RELATIONAL DATA MODEL:

Chapter 6 (Part 1):
Relational Algebra

Outline of Chapter 6

1. Relational Algebra
2. Relational Algebra Example Queries
3. Relational Calculus

1. Relational Algebra

- A basic set of relational model operations constitute the **relational algebra**.

- Sequences of relational algebra operations form **relational algebra expressions** whose result is also a relation

- Two groups of operations:

  set theoretic operations: UNION, INTERSECTION, SET DIFFERENCE, CARTESIAN PRODUCT

  relational model specific operations: SELECT, PROJECT, JOIN

- Extensions of the basic relational model use *additional operators*.

1.1. The SELECT operation (1)

- The **SELECT operation** is used to select a subset of tuples from a relation that satisfy a selection condition.

- Form of the operation: $\sigma_C(R)$

  - **R** is a *relational algebra expression* (the simplest expression is the name of a *database relation*).

  - The **condition C** is an *arbitrary Boolean expression* on the attributes of R.

    The Boolean expression is formed using
    - *clauses* (*atomic comparisons*) of the form
      A *op* B or A *op* c where
      A, B are *attribute names*, c is a *constant*, and
      op is one of the *comparison operators* {=, <, ≤ , >, ≥, ≠}.
    - *the logical operators AND, OR, and NOT.*

1.1. The SELECT operation (2)

- The resulting relation has the *same attributes* as R

- The resulting relation includes *each tuple in r(R) whose attribute values satisfy the condition C*

Examples:

$\sigma_{DNO=4}(EMPLOYEE)$

$\sigma_{SALARY>30000}(EMPLOYEE)$

$\sigma_{(DNO=4 \text{ AND } SALARY>25000) \text{ OR } (DNO=5 \text{ AND } SALARY>30000)}(EMPLOYEE)$

Resulting relation:

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
Franklin	T	Wong	333445555	1955-12-08	638 Voss,Houston,TX	M	40000	888665555	5
Jennifer		Wallace	987654321	1941-06-20	291 Berry,Bellaire,TX	F	43000	888665555	4
Ramesh		Narayan	666884444	1962-09-15	975 FireOak,Humble,TX	M	38000	333445555	5

1.1. The SELECT operation (3)

Remarks:

- Clearly the degree of the resulting relation is the *same* as that of the input relation.
- The resulting relation is *empty* if no tuple satisfies the selection condition.
- The number of tuples (*cardinality*) of the resulting relation is *less than or equal* to that of the input relation: $|\sigma_C(R)| \leq |R|$
- $|\sigma_C(R)| / |R|$ is referred to as the *selectivity* of the selection condition C
- The SELECT operation is *commutative*:
  $\sigma_{C1}(\sigma_{C2}(R)) = \sigma_{C2}(\sigma_{C1}(R))$
- A sequence of SELECT operations can be *combined into a single select operation*:
  $\sigma_{C1}(\sigma_{C2}(R)) = \sigma_{C2 \wedge C1}(R))$

1.2. The PROJECT operation (1)

- The **PROJECT operation** selects some of the columns of a table while discarding the other columns.
- Form of operation: $\Pi_L(R)$

  R is a *relational algebra expression*,
  L is *a list of attributes from the attributes of relation R*.

- The resulting relation has only those attributes of R specified in L and in the same order as they appear in the list.
- Example:

  $\Pi_{LNAME,FNAME,SALARY}(EMPLOYEE)$

Resulting relation →

LNAME	FNAME	SALARY
Smith	John	30000
Wong	Franklin	40000
Zelaya	Alicia	25000
Wallace	Jennifer	43000
Narayan	Ramesh	38000
English	Joyce	25000
Jabbar	Ahmad	25000
Borg	James	55000

1.2. The PROJECT operation (2)

- The PROJECT operation *eliminates duplicate tuples* in the resulting relation so that it remains a mathematical set (no duplicate elements).
- Example:

$\Pi_{SEX,SALARY}(EMPLOYEE)$

Resulting relation →

SEX	SALARY
M	30000
M	40000
F	25000
F	43000
M	38000
M	25000
M	55000

- Note that several female employees have salary 25000, but only a single tuple <F, 25000> is kept in the resulting relation. *Duplicate tuples are eliminated by the Π operator.*

1.2. The PROJECT operation (3)

Remarks:

- The degree of the resulting relation is the number of attributes in the attribute list.
- The number of tuples (*cardinality*) of the resulting relation is less than or equal to that of the input relation:
  $|\Pi_L(R)| \leq |R|$
- If the attribute list L is a superkey of R, the number of tuples in the resulting relation equals the number of tuples in R.
- $\Pi_{L1}(\Pi_{L2}(R)) = \Pi_{L1}(R)$ if $L_1 \subseteq L_2$
- Commutativity does not hold on PROJECT:

  if $\Pi_{L1}(\Pi_{L2}(R))$ is a valid relational algebra expression
  $\Pi_{L2}(\Pi_{L1}(R))$ is not defined unless $L_2 \subseteq L_1$
  (in which case $\Pi_{L1}(\Pi_{L2}(R)) = \Pi_{L2}(\Pi_{L1}(R)) = \Pi_{L2}(R) = \Pi_{L1}(R)$

1.3. Sequences of operations

- We can break complex relational algebra expressions into smaller ones by applying one or more operators at a time and naming the intermediate results.
- This improves readability.
- Example:

  Retrieve the names and salaries of employees who work in department 4:

  $$\Pi_{\text{FNAME,LNAME,SALARY}} (\sigma_{\text{DNO=4}}(\text{EMPLOYEE}))$$

  Alternatively, we specify explicit intermediate relations for each step:

  $$\text{DEPT4\_EMPS} \leftarrow \sigma_{\text{DNO=4}}(\text{EMPLOYEE})$$

  $$\text{RESULT} \leftarrow \Pi_{\text{FNAME,LNAME,SALARY}}(\text{DEPT4\_EMPS})$$

1.4. Attribute renaming (1)

- Attributes can optionally be *renamed* in the resulting relation (this may be required for some operations that will be presented later).
- We can use for this the **attribute renaming operator**:

  Consider the relation schema R(A, B).

  $\rho_{A \rightarrow C}(R)$ returns a relation identical to R except that attribute A is renamed to C. The schema of the resulting relation is (C, B).

- Example: We want to rename LNAME and FNAME to LASTNAME and FIRSTNAME respectively in the result of the previous example.

$$DEPT4\_EMPS \leftarrow \sigma_{DNO=4}(EMPLOYEE)$$

$$TEMPO \leftarrow \Pi_{FNAME,LNAME,SALARY}(DEPT4\_EMPS)$$

$$RESULT \leftarrow \rho_{FNAME \rightarrow FIRSTNAME,\ LNAME \rightarrow LASTNAME}(TEMPO)$$

1.4. Attribute renaming (2)

- Alternatively, attributes can be *renamed* in the resulting left-hand-side relation.

- Example: We want to rename LNAME and FNAME to LASTNAME and FIRSTNAME respectively in the result of the previous example.

$$\text{DEPT4\_EMPS} \leftarrow \sigma_{\text{DNO=4}}(\text{EMPLOYEE})$$

$$\text{RESULT(FIRSTNAME,LASTNAME,SALARY)} \leftarrow \Pi_{\text{FNAME,LNAME,SALARY}}(\text{DEPT4\_EMPS})$$

1.5. Union, Intersection, Set difference operators

- These operations are **binary**: each is applied to two relations.
- In order to apply these operators to relations, the relations have to be *union compatible*.
- Two relations $R(A_1, \ldots, A_n)$ and $S(B_1, \ldots, B_n)$ are **union compatible** if they have the same degree n and $dom(A_i) = dom(B_i)$, $1 \leq i \leq n$.
- Suppose that R and S are union compatible.
  Notation: **Union: $R \cup S$**
  Intersection: $R \cap S$
  Set difference: R - S
- Convention: the **schema** of the relation resulting by any of these operations has the attributes of the first relation R

1.5. Union, Intersection, Set difference operators

- The result of **R ∪ S** is a relation that includes all the tuples that are either in R or in S or in both R and S.
  Duplicates are eliminated.
- Example:

(a)

STUDENT	FN	LN
	Susan	Yao
	Ramesh	Shah
	Johnny	Kohler
	Barbara	Jones
	Amy	Ford
	Jimmy	Wang
	Ernest	Gilbert

INSTRUCTOR	FNAME	LNAME
	John	Smith
	Ricardo	Browne
	Susan	Yao
	Francis	Johnson
	Ramesh	Shah

(b)

FN	LN
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

STUDENT ∪ INSTRUCTOR

1.5. Union, Intersection, Set difference operators

- The result of **R ∩ S** is a relation that includes all the tuples that are in both R and S.
- Example:

(a)

STUDENT	FN	LN
	Susan	Yao
	Ramesh	Shah
	Johnny	Kohler
	Barbara	Jones
	Amy	Ford
	Jimmy	Wang
	Ernest	Gilbert

INSTRUCTOR	FNAME	LNAME
	John	Smith
	Ricardo	Browne
	Susan	Yao
	Francis	Johnson
	Ramesh	Shah

(c)

FN	LN
Susan	Yao
Ramesh	Shah

STUDENT ∩ INSTRUCTOR

1.5. Union, Intersection, Set difference operators

- The result of **R - S** is a relation that includes all the tuples that are in R but not in S.
- Example:

STUDENT	FN	LN
	Susan	Yao
	Ramesh	Shah
	Johnny	Kohler
	Barbara	Jones
	Amy	Ford
	Jimmy	Wang
	Ernest	Gilbert

INSTRUCTOR	FNAME	LNAME
	John	Smith
	Ricardo	Browne
	Susan	Yao
	Francis	Johnson
	Ramesh	Shah

FN	LN
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

STUDENT- INSTRUCTOR

FNAME	LNAME
John	Smith
Ricardo	Browne
Francis	Johnson

INSTRUCTOR - STUDENT

1.5. Union, Intersection, Set difference operators

- Intersection and Union are *commutative* operations (modulo attribute renaming).
  $R \cup S = S \cup R$
  $R \cap S = S \cap R$

- Intersection and Union are *associative* operations.
  $R \cup (S \cup T) = (R \cup S) \cup T$
  $R \cap (S \cap T) = (S \cap R) \cap T$

- Set difference *is not a commutative* operation.
  In general, $R - S \neq S - R$

1.6. The Cartesian Product operation (1)

- The **Cartesian product (Cross Product, Cros Join)** is a binary operation and the input queries do not have to be union compatible.

- Notation: **R x S**

- Given $R(A_1, \ldots, A_n)$ and $S(B_1, \ldots, B_m)$ the *schema* of the relation T resulting by the operation R x S is:
  $T(A_1, \ldots, A_n, B_1, \ldots, B_m)$. It has $n + m$ attributes.

  Notice that the schemas R and S should not have common attributes – no two attributes with the same name can appear in a relation schema.

- The *instance* of Q has one tuple for each combination of tuples – one from R and one from S.

1.6. The Cartesian Product operation (2)

- Example:

R

A_1	A_2
a	b
a’	b’

S

B_1	B_2	B_3
1	2	3
4	5	6

R x S

A_1	A_2	B_1	B_2	B_3
a	b	1	2	3
a	b	4	5	6
a’	b’	1	2	3
a’	b’	4	5	6

1.6. The Cartesian Product operation (3)

- The cardinality |R x S| of R x S is $n_R * n_S$ where n_R is the cardinality of R and n_S is the cardinality of S

- The Cartesian product is a *meaningless operation* on its own. It can *combine related tuples* from two relations *if followed by the appropriate SELECT operation* (in which case it is called *join operation*).

1.7. The Join operation (1)

- The **JOIN operation** is used to combine related tuples from two relations into single tuples.

- The most general form of the join operation is the **THETA JOIN**:

- The theta join is similar to a Cartesian product followed by a selection.

1.7. The Join operation (2)

Theta join

- R(A_1, A_2, ..., A_n), and S(B_1, B_2, ..., B_m)

  R *and* S *should not have common attributes.*

  Notation: $\mathbf{R} \rhd\lhd_{\mathbf{C}} \mathbf{S}$

  C is called **join condition** and is an expression of the form
  $(A_i\ \theta\ B_j)$ AND ... AND $(A_h\ \theta\ B_k)$, $1 \le i,h \le n$, $1 \le j,k \le m$,

  where θ is one of the comparison operators in $\{=, <, \le, >, \ge, \ne\}$, and

  attributes A_i and B_j have the *same domain*.

- The result of $\mathbf{R} \rhd\lhd_{\mathbf{C}} \mathbf{S}$ is a relation Q with n+m attributes having schema $\mathbf{Q(A_1, A_2, ..., A_n, B_1, B_2, ..., B_m)}$

- Q has *one tuple for each combination of tuples – one from R and one from S – whenever the combination satisfies the join condition*.

1.7. The Join operation (3)

- Example:

Retrieve each DEPARTMENT's name and its manager's name:

$T \leftarrow DEPARTMENT \triangleright\!\triangleleft_{MGRSSN=SSN} EMPLOYEE$

$RESULT \leftarrow \Pi_{DNAME,FNAME,LNAME}(T)$

The intermediate result T:

DNAME	DNUMBER	MGRSSN	...	FNAME	MINIT	LNAME	SSN	...
Research	5	333445555	...	Franklin	T	Wong	333445555	...
Administration	4	987654321	...	Jennifer	S	Wallace	987654321	...
Headquarters	1	888665555	...	James	E	Borg	888665555	...

1.7. The Join operation (4)

Equijoin

- The most common join operation involves only equality and is called **equijoin**.
- The join condition C of an equijoin between R and S includes one or more *equality comparisons* involving attributes from R and S.

That is, C is of the form:
$(A_i=B_j)$ AND ... AND $(A_h=B_k)$; $1 \leq i,h \leq n$, $1 \leq j,k \leq m$

A_i, ..., A_h are called the **join attributes of R**
B_j, ..., B_k are called the **join attributes of S**

1.7. The Join operation (5)

Natural join

- In an equijoin $T \leftarrow R \rhd\lhd_C S$, the join attributes of S appear *redundantly* in the result relation Q.

- In a NATURAL JOIN, the *redundant join attributes* of S are *eliminated* from T

- The standard definition of the natural join requires that the join attributes in each pair of join attributes have the *same name* in both relations.
  Thus, the join attributes are implied and the *join condition need not be specified*.

- The notation is simplified.

  Notation: $T \leftarrow R * S$

1.7. The Join operation (6)

Natural join

- Example:

DEPT_LOCS ← DEPARTMENT * DEPT_LOCATIONS

DEPT_LOCS	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE	LOCATION
	Headquarters	1	888665555	1981-06-19	Houston
	Administration	4	987654321	1995-01-01	Stafford
	Research	5	333445555	1988-05-22	Bellaire
	Research	5	333445555	1988-05-22	Sugarland
	Research	5	333445555	1988-05-22	Houston

1.7. The Join operation (7)

Natural join

- If the join attributes do not have the same name in the two relations, a renaming operation is applied first.

- Example:

  We want to join PROJECT and DEPARTMENT on attributes DNUM and DNUMBER respectively using a natural join operator.

$$PROJ\_DEPT \leftarrow PROJECT * \rho_{DNUMBER \rightarrow DNUM}(DEPARTMENT)$$

PROJ_DEPT	PNAME	PNUMBER	PLOCATION	DNUM	DNAME	MGRSSN	MGRSTARTDATE
	ProductX	1	Bellaire	5	Research	333445555	1988-05-22
	ProductY	2	Sugarland	5	Research	333445555	1988-05-22
	ProductZ	3	Houston	5	Research	333445555	1988-05-22
	Computerization	10	Stafford	4	Administration	987654321	1995-01-01
	Reorganization	20	Houston	1	Headquarters	888665555	1981-06-19
	Newbenefits	30	Stafford	4	Administration	987654321	1995-01-01

1.7. The Join operation (8)

Remarks:

- If no combination of tuple satisfies the join condition, the result of the join is an *empty relation*.

- If R has n_R and S has n_S tuples, $R \triangleright\triangleleft_C S$ will have between 0 and $n_R * n_S$ tuples.

- Join selectivity is the expected number of tuples in the join result $R \triangleright\triangleleft_C S$ divided by $n_R * n_S$.
  This is a property of the *join condition*.

- A join can be specified between a relation and itself (*self-join*).
  One can *think of this* as joining *two distinct copies* of the relation, although only one relation actually exists

1.8. Relational Algebra as a query language

- We can use relational algebra expressions to express *requests on the data stored in a database*. We shall generally refer to expressions of relational algebra as **queries**.

- A relational algebra expression (query) Q involves, relation names in a database D as operands. The **answer to Q over D** is the relation resulting by substituting relation names in Q by the corresponding relations in D and performing the operations.

1.8. Complete set of relational algebra operations

- All the operations discussed so far can be described as a sequence of *only* the operations SELECT, PROJECT, UNION, SET DIFFERENCE, and CARTESIAN PRODUCT (and ATTRIBUTE RENAME).
- Two relational algebra expressions are **equivalent** (notation ≡) if they produce the same answer, whenever they are given the same relations as operands.

  Example:

  $R \cap S \equiv R - (R - S)$

  $R \rhd\lhd_C S \equiv \sigma_C (R \times S)$

- Therefore, INTERSECTION and JOIN operations can be equivalently expressed using SELECT, CARTESIAN PRODUCT, and SET DIFFERENCE operations.

1.8. Complete set of relational algebra operations

- The set {σ, Π, $\cup$, - , x } is called a *complete set* of relational algebra operations. Any query language that can equivalently express any expression involving these operations is called **relationally complete**.

- The rest of the operations that we have seen so far ($\cap$, $\triangleright\triangleleft_C$, *) are not necessary for the expressive power of relational algebra but they are convenient to use.

1.9. The DIVISION operation (1)

- The **DIVISION operation** is useful for a special kind of query.

- Example: Retrieve the names of employees that work on **all** the projects.

- Consider two relations **R(Z)** and **S(X)** where $\mathbf{X \subset Z}$
  Let $\mathbf{Y = Z - X}$ (thus $\mathbf{Z = X \cup Y}$)

  Notation: $\mathbf{R \div S}$

- The result is a relation T(Y) that includes a tuple t iff for every tuple t_S in S there is a tuple t_R in R such that $t_R[Y] = t$, and $t_R[X] = t_S$.

  That is, if a tuple t appears in T, the values in t appear in R in combination with every tuple in S.

- Note that this operation can be performed only if $\mathbf{X \subset Z}$.

1.9. The DIVISION operation (2)

- Example:

R	A	B
	a1	b1
	a2	b1
	a3	b1
	a4	b1
	a1	b2
	a3	b2
	a2	b3
	a3	b3
	a4	b3
	a1	b4
	a2	b4
	a3	b4

S	A
	a1
	a2
	a3

T = R ÷ S

T	B
	b1
	b4

1.9. The DIVISION operation (3)

- **÷** can be expressed in terms of **Π, x**, and **-** as follows:

  Consider relations **R(Z)** and **S(X)** where **X ⊂ Z**
  and let **Y = Z – X** .

  $T_1 \leftarrow \Pi_Y(R)$

  $T_2 \leftarrow \Pi_Y((S \times T_1) - R)$

  $T \leftarrow T_1 - T_2$

1.10. Additional Relational operators

- Some common relational database requests *cannot be expressed* using the traditional relational algebra operators.
- New operators have been introduced to deal with this limitation.
- The $\mathcal{F}$ operator is useful for grouping and aggregating tuples. It *enhances the expressive power of the relational algebra*.
- The left outer join, right outer join and full outer join operators are useful in an “extended” relational model where null values are allowed to appear in relations.

1.11. The Grouping/Aggregation operation (1)

- The **aggregate functions** are functions that are applied to collections of values and return single values.

  Common aggregate function used in database applications are:

  SUM, AVERAGE, MAXIMUM, MINIMUM. These functions are applied to collections of numeric values.

  COUNT is used for counting values (or tuples).

- *Grouping tuples* of a relation based on some attributes of the relation (**grouping attributes**) means partitioning the tuples of the relation in groups where each group contains the tuples of the relation with the same combination of values for the grouping attributes.

1.11. The Grouping/Aggregation operation (2)

- Notation: $_{<ga>}\mathcal{F}_{<aa>}$ (R)

  where
  <ga> is a list of grouping attributes, and
  <aa> is a list of aggregated attributes, that is
  an expressions of the form *agg(A)*, where *agg* is an aggregate function and *A* is an attribute of *R*.

- The **schema of the resulting relation** is the list of attributes <ga, aa>.

- The **instance of the resulting relation** is computed by grouping the tuples of R based on the grouping attributes in <ga>, and by applying the aggregate functions *agg* in <aa> to all the values for the respective attributes *A* in each group.

- Notice: an attribute value can occur *many times* in a group.

1.11. The Grouping/Aggregation operation (3)

- Example: Retrieve each department's number, the *number of employees* in the department and *their average salary*.

$$T \leftarrow {}_{DNO}\mathcal{F}_{COUNT(SSN),AVERAGE(SALARY)}(EMPLOYEE)$$

T

DNO	COUNT(SSN)	AVERAGE(SALARY)
5	4	33250
4	3	31000
1	1	55000

To change the name of the attributes (including the aggregated attributes) in the resulting relation the attribute renaming operator can be used:

$$RESULT \leftarrow \rho_{COUNT(SSN) \rightarrow COUNT\_SSN,\ AVERAGE(SALARY) \rightarrow AVERAGE\_SALARY}(T)$$

RESULT

DNO	COUNT_SSN	AVERAGE_SALARY
5	4	33250
4	3	31000
1	1	55000

1.11. The Grouping/Aggregation operation (4)

- If no grouping attributes are specified in the grouping/aggregation operation (i.e. if it is of the form $\mathcal{F}_{<aa>}(R)$), the aggregate functions are applied to the attribute values of *all the tuples in* R.

- Example: Retrieve the *total number of employees*, and *their average salary*.

$$T \leftarrow \mathcal{F}_{\text{COUNT(SSN), AVERAGE(SALARY)}}(\text{EMPLOYEE})$$

$$\text{RESULT} \leftarrow \rho_{\text{COUNT(SSN)} \rightarrow \text{COUNT\_SSN, AVERAGE(SALARY)} \rightarrow \text{AVERAGE\_SALARY}}(T)$$

RESULT

COUNT_SSN	AVERAGE_SALARY
8	35125

1.12. The OUTER JOIN operations (1)

- An extension of the traditional relational model allows *null values in the relations*.
- A join operation R $\triangleright\triangleleft_C$ S *matches* tuples in R with tuples in S that satisfy the join condition.

  Tuples in R *without a matching tuple* in S (and conversely) *are eliminated* from the JOIN result.

  As a consequence, tuples from both relations that have *null values for the join attributes are eliminated* from the JOIN result.

- Some queries require *all* tuples in R (or in S or in both) to appear in the result.
- A set of operations, called **outer joins**, can be used *to keep all the tuples in R, or in S, or in both relations in the result of the JOIN*, whether or not they have matching tuples in the other relation.
- This is possible only when *null values are allowed* in relations.

1.12. The OUTER JOIN operations (2)

- The **LEFT OUTER JOIN** operation (denoted =▷◁) keeps *every tuple in the first (i.e. left) relation R* in **R =▷◁ S**. If no matching tuple is found in S for a tuple in R, the attributes of S in the join result tuple are *"padded" with nulls*.
- Example: List the name of all employees and also the name of the department they manage (if they happen to manage a department).

  T ← EMPLOYEE $=\triangleright\triangleleft_{SSN=MGRSSN}$ DEPARTMENT)

  RESULT ← $\Pi_{FNAME, MINIT, LNAME, DNAME}$(T)

RESULT	FNAME	MINIT	LNAME	DNAME
	John	B	Smith	null
	Franklin	T	Wong	Research
	Alicia	J	Zelaya	null
	Jennifer	S	Wallace	Administration
	Ramesh	K	Narayan	null
	Joyce	A	English	null
	Ahmad	V	Jabbar	null
	James	E	Borg	Headquarters

1.12. The OUTER JOIN operations (3)

- Similarly, the **RIGHT OUTER JOIN** operation (denoted ⟖) keeps *every tuple in the second (i.e. right) relation S* in **R ⟖ S**.

- The **FULL OUTER JOIN** operation (denoted ⟗) keeps *every tuple in both the first and the second relations R and S* in **R ⟗ S.** When no matching tuple is found, they are *"padded" with nulls* as needed.

1.13. Recursive closure operation (1)

- Another type of operation that *cannot* be specified in the basic relational algebra is *recursive closure*.

- Example of a recursive operation:
  Retrieve all supervisees of a specific employee e *at all levels*.

  This query *is not* expressible in basic relational algebra.

  Note that we can express in relational algebra the query: retrieve all supervisees of a specific employee e *at a specific level a or at a lower level*. $\rightarrow \rightarrow$

1.13. Recursive closure operation (2)

Example (cont): let e be the employee ‘James Borg’.

The SSN of his direct supervisees (i.e. those at level one) is:

$$\text{BORG\_SSN} \leftarrow \rho_{\text{SSN}\rightarrow\text{SUPERSSN}}(\Pi_{\text{SSN}}(\sigma_{\text{FNAME='James'} \wedge \text{LNAME='Borg'}}(\text{EMPLOYEE})))$$

$$\text{SUPERVISION} \leftarrow \Pi_{\text{SSN, SUPERSSN}}(\text{EMPLOYEE}))$$

$$\textbf{RESULT1} \leftarrow \Pi_{\text{SSN}}(\text{SUPERVISION} * \text{BORG\_SSN})$$

The SSN of his supervisees at level two is:

$$\text{R1\_SUPER} \leftarrow \rho_{\text{SSN}\rightarrow\text{SUPERSSN}}(\text{RESULT1})$$

$$\textbf{RESULT2} \leftarrow \Pi_{\text{SSN}}(\text{SUPERVISION} * \text{R1\_SUPER})$$

The SSN of his supervisees at both levels (one and two) is:

$$\textbf{RESULT} \leftarrow \textbf{RESULT1} \cup \textbf{RESULT2}$$

1.13. Recursive closure operation (3)

A two-level recursive query.

(Borg's SSN is 888665555)

(SSN) (SUPERSSN)

SUPERVISION	SSN1	SSN2
	123456789	333445555
	333445555	888665555
	999887777	987654321
	987654321	888665555
	666884444	333445555
	453453453	333445555
	987987987	987654321
	888665555	null

RESULT 1	SSN
	333445555
	987654321

(Supervised by Borg)

RESULT 2	SSN
	123456789
	999887777
	666884444
	453453453
	987987987

(Supervised by Borg's subordinates)

RESULT	SSN
	123456789
	999887777
	666884444
	453453453
	987987987
	333445555
	987654321

(RESULT1 ∪ RESULT2)