Chapter - 4

Relational Algebra Operations & Extended Relational Algebra Operations

Outline

- Relational Algebra
  - Unary Relational Operations
  - Relational Algebra Operations From Set Theory
  - Binary Relational Operations
  - Additional Relational Operations
  - Examples of Queries in Relational Algebra

Describing a Relational Database Mathematically: Relational Algebra

- Tables in a relational database as sets of tuples
- Query operators using set theory
- The query language is called relational algebra
- Normally, not used directly - foundation for SQL and query processing
  - SQL adds syntactic sugar

What is an “Algebra”?

- In general, algebra is the study of mathematical symbols and the rules for manipulating these symbols

- Mathematical system consisting of
  - *Operands* - variables or values from which new values can be constructed
  - *Operators* - symbols denoting procedures that construct new values from given values
- Expressions can be constructed by applying operators to atomic operands and/or other expressions
  - Operations can be **composed** - algebra is closed
  - Parentheses are needed to group operators

Relational Algebra -Overview

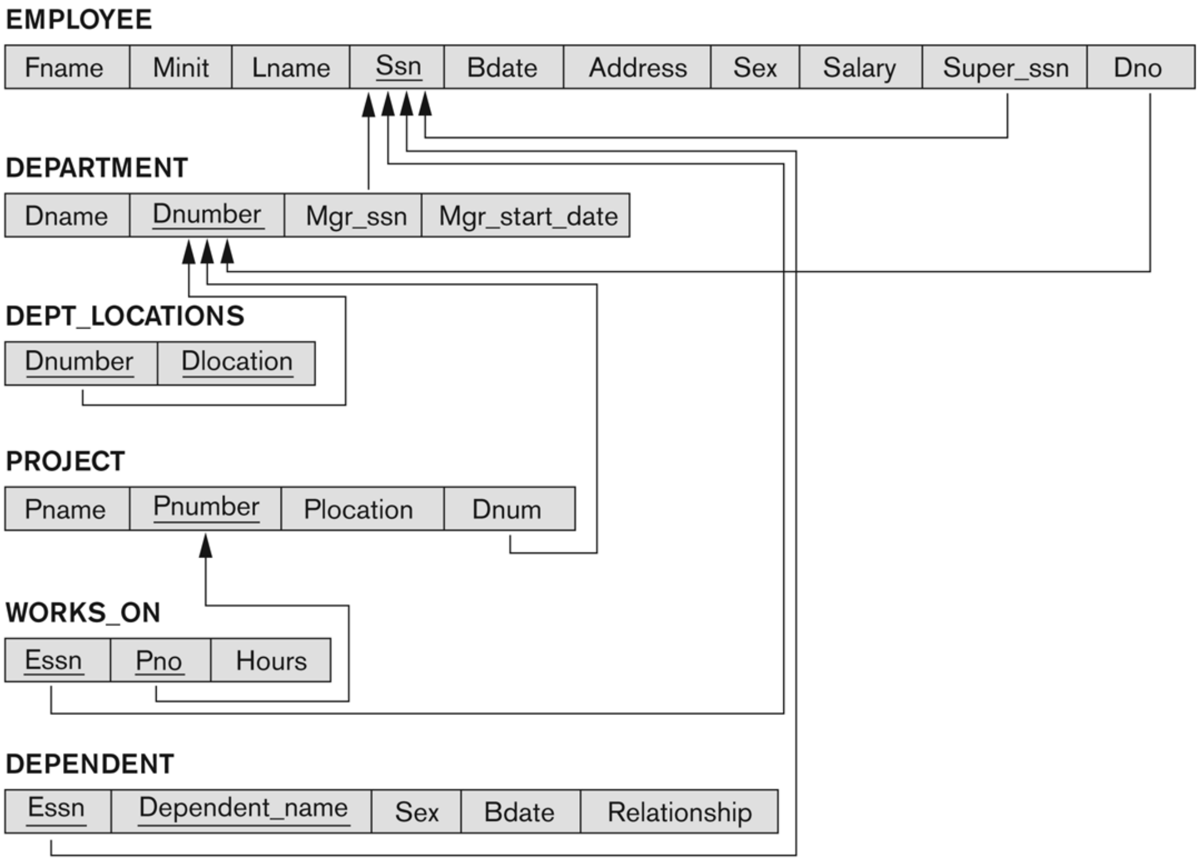
- Relational algebra is the basic set of operations for the relational model
- These operations enable a user to specify **basic retrieval requests** (or **queries**)
- The result of an operation is a *new relation*, which may have been formed from one or more *input* relations
  - This property makes the algebra “closed” (all objects in relational algebra are relations)
- The **algebra operations** thus produce new relations
  - These can be further manipulated using operations of the same algebra
- A sequence of relational algebra operations forms a **relational algebra expression**
  - The result of a relational algebra expression is also a relation that represents the result of a database query (or retrieval request)

Relational Algebra Overview

- Relational Algebra consists of several groups of operations
  - Unary Relational Operations
    - SELECT (symbol: $\boldsymbol{\sigma}$ (sigma))
    - PROJECT (symbol: $\boldsymbol{\pi}$ (pi))
    - RENAME (symbol: ρ (rho))
  - Relational Algebra Operations From Set Theory
    - UNION ($\cup$), INTERSECTION ($\cap$), DIFFERENCE (or MINUS, **–**)
    - CARTESIAN PRODUCT (**x**)
  - Binary Relational Operations
    - JOIN (several variations of JOIN exist)
    - DIVISION
  - Additional Relational Operations
    - OUTER JOINS, OUTER UNION
    - AGGREGATE FUNCTIONS (These compute summary of information: for example, SUM, COUNT, AVG, MIN, MAX)

Relational Schema – COMPANY Database

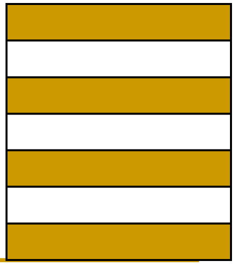
Figure 5.7
Referential integrity constraints displayed on the COMPANY relational database schema.



Unary Relational Operations: SELECT

- The SELECT operation (denoted by σ (sigma)) is used to select a *subset* of the tuples from a relation based on a **selection condition**
  - The selection condition acts as a **filter**
  - Keeps only those tuples that satisfy the qualifying condition
  - Tuples satisfying the condition are *selected* whereas the other tuples are discarded (*filtered out*)
  - The SELECT operation can also be visualized as a *horizontal partition* of the relation into two sets of tuples
    - those tuples that satisfy the condition and are selected, and those tuples that do not satisfy the condition and are discarded
- Examples:
  - Select the EMPLOYEE tuples whose department number is 4:

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



Unary Relational Operations: SELECT

- Select the employee tuples whose salary is greater than $30,000

$$\sigma_{\text{SALARY} > 30{,}000}(\text{EMPLOYEE})$$

- In general, the *select* operation is denoted by

$$\sigma_{\text{<selection condition>}}(R)$$

where

- the symbol **σ** (sigma) is used to denote the *select* operator
- the selection condition (made of a number of clauses) is a Boolean (conditional) expression specified on the attributes of relation R

  <attribute name> <comparison op> <constant value> (or)

  <attribute name> <comparison op> <attribute name>

  where

  - <attribute name> is the name of an attribute of *R*,
  - <comparison op> is normally one of the operators {=, <, ≤, >, ≥, ≠}, and
  - <constant value> is a constant value from the attribute domain

Unary Relational Operations: SELECT

- Clauses can be connected by the standard Boolean operators *and*, *or*, and *not* to form a general selection condition
- For example : To select the tuples for all employees who either work in department 4 and make over $25,000 per year, or work in department 5 and make over $30,000, we can specify the following SELECT operation:

$$\sigma_{(\text{Dno}=4 \textbf{ AND } \text{Salary}>25000) \textbf{ OR } (\text{Dno}=5 \textbf{ AND } \text{Salary}>30000)}(\text{EMPLOYEE})$$

- tuples that make the condition **true** are selected
  - appear in the result of the operation
- tuples that make the condition **false** are filtered out
  - discarded from the result of the operation

Unary Relational Operations: SELECT

- The SELECT operator is **unary**; that is, it is applied to a single relation
- The selection operation is applied to *each tuple individually*
- The **degree** of the relation resulting from a SELECT operation-its number of attributes-is the same as the degree of *R*
- The **number of tuples** in the resulting relation is always *less than or equal to* the number of tuples in *R*, $|\sigma_c(R)| \leq |R|$ for any condition c

Unary Relational Operations: SELECT

- **SELECT Operation Properties**
  - The SELECT operation $\sigma_{\text{<selection condition>}}(R)$ produces a relation S that has the same schema (same attributes) as R
  - SELECT σ is commutative
    - $\sigma_{\text{<condition1>}}(\sigma_{\text{< condition2>}}(R)) = \sigma_{\text{<condition2>}}(\sigma_{\text{< condition1>}}(R))$
  - Because of commutativity property, a cascade (sequence) of SELECT operations may be applied in any order
    - $\sigma_{\text{<cond1>}}(\sigma_{\text{<cond2>}}(\sigma_{\text{<cond3>}}(R)) = \sigma_{\text{<cond2>}}(\sigma_{\text{<cond3>}}(\sigma_{\text{<cond1>}}(R)))$
  - A cascade of SELECT operations may be replaced by a single selection with a conjunction of all the conditions
    - $\sigma_{\text{<cond1>}}(\sigma_{\text{< cond2>}}(\sigma_{\text{<cond3>}}(R)) = \sigma_{\text{<cond1> AND < cond2> AND < cond3>}}(R)))$
  - The number of tuples in the result of a SELECT is less than (or equal to) the number of tuples in the input relation R

Unary Relational Operations: SELECT

$$\sigma_{rating>8}(S2)$$

Selects rows that satisfy selection condition

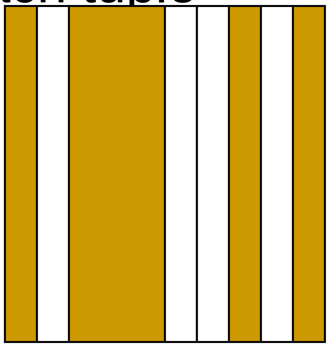
S2

sid	sname	rating	age
28	yuppy	9	35.0
~~31~~	~~lubber~~	~~8~~	~~55.5~~
~~44~~	~~guppy~~	~~5~~	~~35.0~~
58	rusty	10	35.0

sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

Unary Relational Operations: PROJECT

- The SELECT operation - chooses some of the *rows* from the table while discarding other rows.
- The **PROJECT** operation - selects certain *columns* from the table and discards the other columns.
  - If we are interested in only certain attributes of a relation, we use the PROJECT operation to *project* the relation over these attributes only
- The result of the PROJECT operation can be visualized as a *vertical partition* of the relation into two relations
  - The list of specified columns (attributes) is kept in each tuple
  - The other attributes in each tuple are discarded



- PROJECT Operation - denoted by π (pi)

Unary Relational Operations: PROJECT

- Example: To list each employee's first and last name and salary, the following is used

$$\pi_{\text{LNAME, FNAME,SALARY}}(\text{EMPLOYEE})$$

- The general form of the *project* operation is:

$$\pi_{\text{<attribute list>}}(\text{R})$$

  - π (pi) is the symbol used to represent the *project* operation
  - <attribute list> is the desired list of attributes from relation R

- The project operation removes any duplicate tuples
  - This is because the result of the *project* operation must be a set of tuples
  - If duplicates are not eliminated, the result would be a **multiset** or **bag** of tuples rather than a set
  - Mathematical sets *do not allow* duplicate elements

Unary Relational Operations: PROJECT

- **PROJECT Operation Properties**
  - The number of tuples in the result of projection $\pi_{\text{<list>}}(R)$ is always less or equal to the number of tuples in R
  - If the list of attributes includes a *key* of R, then the number of tuples in the result of PROJECT is equal to the number of tuples in R
    - The result of the PROJECT operation has only the attributes specified in <attribute list> *in the same order as they appear in the list*. Hence, its **degree** is equal to the number of attributes in <attribute list>
  - PROJECT is not commutative
    - $\pi_{\text{<list1>}}(\pi_{\text{<list2>}}(R)) = \pi_{\text{<list1>}}(R)$ as long as <list2> contains the attributes in <list1>

Unary Relational Operations: PROJECT

- In SQL, the PROJECT attribute list is specified in the SELECT clause of a query
- For example, the following operation:

$$\pi_{\text{Sex, Salary}}(\text{EMPLOYEE})$$

would correspond to the following SQL query:

SELECT DISTINCT Sex, Salary

FROM EMPLOYEE

- Note : if we remove the keyword **DISTINCT** from this SQL query, then duplicates will not be eliminated

Unary Relational Operations: PROJECT

$$\pi_{age}(S2)$$

- Deletes attributes that are not in *projection list*.

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

age
35.0
55.5

Schema: Result(age)

Duplicates are eliminated
(sets not bags)

Unary Relational Operations: SELECT

Figure 6.1
Results of SELECT and PROJECT operations. (a) $\sigma_{(\text{Dno}=4 \textbf{ AND } \text{Salary}>25000) \text{ OR } (\text{Dno}=5 \textbf{ AND } \text{Salary}>30000)}$ (EMPLOYEE). (b) $\pi_{\text{Lname, Fname, Salary}}$(EMPLOYEE). (c) $\pi_{\text{Sex, Salary}}$(EMPLOYEE).

(a)

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5

(b)

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

(c)

Sex	Salary
M	30000
M	40000
F	25000
F	43000
M	38000
M	25000
M	55000

Relational Algebra Expressions

- We may want to apply several relational algebra operations one after the other
  - Either we can write the operations as a single **relational algebra expression** by nesting the operations, or
  - We can apply one operation at a time and create **intermediate result relations**
- In the latter case, we must give names to the relations that hold the intermediate results

Single expression versus sequence of relational operations (Example)

- To retrieve the first name, last name, and salary of all employees who work in department number 5, we must apply a select and a project operation
- We can write a single relational algebra expression as follows:

$$\pi_{\text{FNAME, LNAME, SALARY}}(\sigma_{\text{DNO=5}}(\text{EMPLOYEE}))$$

- (OR) We can explicitly show the **sequence of operations**, giving a name to each intermediate relation:

$$\text{DEP5\_EMPS} \leftarrow \sigma_{\text{DNO=5}}(\text{EMPLOYEE})$$

$$\text{RESULT} \leftarrow \pi_{\text{FNAME, LNAME, SALARY}}(\text{DEP5\_EMPS})$$

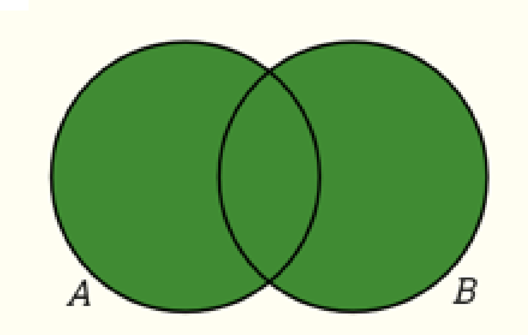
Unary Relational Operations: RENAME

- The RENAME operator is denoted by ρ (rho)
- In some cases, we may want to *rename* the attributes of a relation or the relation name or both
  - Useful when a query requires multiple operations
  - Necessary in some cases (see JOIN operation later)
- The general RENAME operation ρ can be expressed by any of the following forms
    - $\rho_{S\ (B1,\ B2,\ \ldots,\ Bn\ )}(R)$ changes both
    - the relation name to S, *and*
    - the column (attribute) names to B1, B1, …..,Bn
  - $\rho_S(R)$ changes
    - the *relation name* only to S
  - $\rho_{(B1,\ B2,\ \ldots,\ Bn\ )}(R)$ changes
    - the *column (attribute) names* only to B1, B1, …..Bn

Unary Relational Operations: RENAME

- For convenience, we also use a *shorthand* for renaming attributes in an intermediate relation
  - If we write:
    - RESULT $\leftarrow \pi_{\text{FNAME, LNAME, SALARY}}$ (DEP5_EMPS)
    - RESULT will have the *same attribute names* as DEP5_EMPS (same attributes as EMPLOYEE)
  - If we write:
    - RESULT (F, M, L, S, B, A, SX, SAL, SU, DNO)$\leftarrow \pi_{\text{FNAME, LNAME, SALARY}}$ (DEP5_EMPS)
    - The 10 attributes of DEP5_EMPS are *renamed* to F, M, L, S, B, A, SX, SAL, SU, DNO, respectively

Relational Algebra Operations from Set Theory : UNION



- UNION Operation
  - Binary operation, denoted by $\cup$
  - The result of $R \cup S$, is a relation that includes all tuples that are either in R or in S or in both R and S
  - Duplicate tuples are eliminated
  - The two operand relations R and S must be "type compatible" (or UNION compatible)
    - R and S must have same number of attributes (arity)
    - Each pair of corresponding attributes must be type compatible (have same or compatible domains)
- Two relations $R(A1, A2, ..., An)$ and $S(B1, B2, ..., Bn)$ are said to be **union compatible** (or **type compatible**) if they have the same degree n and if $dom(Ai) = dom(Bi)$ for $1 \leq i \leq n$

Union Compatible

- Same number of fields.
- `Corresponding' fields have the same type.

Schema of S1 = Schema of S1

S1(sid,sname,rating,age)

S2(sid,sname,rating,age)

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Relational Algebra Operations from Set Theory : UNION

- Three operations UNION, INTERSECTION, and SET DIFFERENCE on two union-compatible relations *R* and *S* are defined as follows:
  - UNION: The result of this operation, denoted by $R \cup S$, is a relation that includes all tuples that are either in *R* or in *S* or in both *R* and *S*. Duplicate tuples are eliminated
  - INTERSECTION: The result of this operation, denoted by $R \cap S$, is a relation that includes all tuples that are in both *R* and *S*
  - SET DIFFERENCE (or MINUS): The result of this operation, denoted by $R – S$, is a relation that includes all tuples that are in *R* but not in *S*

Relational Algebra Operations from Set Theory : UNION

- Example
  - To retrieve the social security numbers of all employees who either *work in department 5* (RESULT1 below) or *directly supervise an employee who works in department 5* (RESULT2 below)
  - We can use the UNION operation as follows:

$$DEP5\_EMPS \leftarrow \sigma_{DNO=5}(EMPLOYEE)$$

$$RESULT1 \leftarrow \pi_{SSN}(DEP5\_EMPS)$$

$$RESULT2(SSN) \leftarrow \pi_{SUPERSSN}(DEP5\_EMPS)$$

$$RESULT \leftarrow RESULT1 \cup RESULT2$$

  - The union operation produces the tuples that are in either RESULT1 or RESULT2 or both

Relational Algebra Operations from Set Theory : UNION

Union

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

Union Compatable

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
44	guppy	5	35.0
28	yuppy	9	35.0

Compute: S1 U S2

S1(sid,sname,rating,age)

S2(sid,sname,rating,age)

Result(sid,sname,rating,age)

Duplicates

Relational Algebra Operations from Set Theory : UNION

RESULT1

Ssn
123456789
333445555
666884444
453453453

RESULT2

Ssn
333445555
888665555

RESULT

Ssn
123456789
333445555
666884444
453453453
888665555

Figure 6.3
Result of the UNION operation RESULT ← RESULT1 ∪ RESULT2.

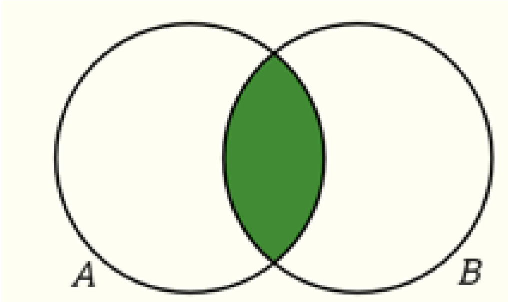
Relational Algebra Operations from Set Theory

- Type Compatibility of operands is required for the binary set operation UNION $\cup$, (also for INTERSECTION $\cap$, and SET DIFFERENCE –)
- R1(A1, A2, ..., An) and R2(B1, B2, ..., Bn) are type compatible if:
  - they have the same number of attributes, and
  - the domains of corresponding attributes are type compatible (i.e. dom(Ai)=dom(Bi) for i=1, 2, ..., n)

- The resulting relation for R1$\cup$R2 (also for R1$\cap$R2, or R1–R2) has the same attribute names as the *first* operand relation R1 (by convention)

Relational Algebra Operations from Set Theory: INTERSECTION

- INTERSECTION is denoted by $\cap$
- The result of the operation $R \cap S$, is a relation that includes all tuples that are in both R and S
  - The attribute names in the result will be the same as the attribute names in R
- The two operand relations R and S must be "type compatible"



Relational Algebra Operations from Set Theory: INTERSECTION

Intersection

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

sid	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0

Compute: S1 ∩ S2

S2

Union Compatible

Duplicates

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

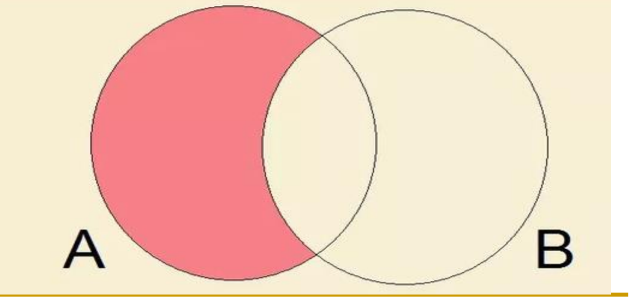
S1(sid,sname,rating,age)
S2(sid,sname,rating,age)

Result(sid,sname,rating,age)

The same schema

Relational Algebra Operations from Set Theory: SET DIFFERENCE

- SET DIFFERENCE (also called MINUS or EXCEPT) is denoted by –
- The result of R – S, is a relation that includes all tuples that are in R but not in S
  - The attribute names in the result will be the same as the attribute names in R
- The two operand relations R and S must be “type compatible”



Relational Algebra Operations from Set Theory: SET DIFFERENCE

Difference

S1

sid	sname	rating	age
22	dustin	7	45.0
~~31~~	~~lubber~~	~~8~~	~~55.5~~
~~58~~	~~rusty~~	~~10~~	~~35.0~~

sid	sname	rating	age
22	dustin	7	45.0

Compute: S1 - S2

S2 Union Compatible

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Take away Duplicates

S1(sid,sname,rating,age)

S2(sid,sname,rating,age)

Result(sid,sname,rating,age)

The same schema

Example to illustrate the result of UNION, INTERSECT, and DIFFERENCE

(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

(c)

Fn	Ln
Susan	Yao
Ramesh	Shah

(d)

Fn	Ln
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

(e)

Fname	Lname
John	Smith
Ricardo	Browne
Francis	Johnson

Figure 6.4
The set operations UNION, INTERSECTION, and MINUS. (a) Two union-compatible relations. (b) STUDENT ∪ INSTRUCTOR. (c) STUDENT ∩ INSTRUCTOR. (d) STUDENT – INSTRUCTOR. (e) INSTRUCTOR – STUDENT.

Some Properties of UNION, INTERSECT, and DIFFERENCE

- Notice that both union and intersection are *commutative* operations; that is
  - $R \cup S = S \cup R$, and $R \cap S = S \cap R$
- Both union and intersection can be treated as n-ary operations applicable to any number of relations as both are *associative* operations; that is
  - $R \cup (S \cup T) = (R \cup S) \cup T$
  - $(R \cap S) \cap T = R \cap (S \cap T)$
- The minus operation is not commutative; that is, in general
  - $R - S \neq S - R$

Relational Algebra Operations from Set Theory: CARTESIAN PRODUCT

- CARTESIAN (or CROSS) PRODUCT Operation
  - This operation is used to combine tuples from two relations in a combinatorial fashion
  - Denoted by R(A1, A2, . . ., An) x S(B1, B2, . . ., Bm)
  - Result is a relation Q with degree n + m attributes
    - Q(A1, A2, . . ., An, B1, B2, . . ., Bm), in that order
  - The resulting relation state has one tuple for each combination of tuples—one from R and one from S.
  - Hence, if R has n_R tuples (denoted as $|R| = n_R$), and S has n_S tuples, then R x S will have $n_R * n_S$ tuples
  - The two operands do NOT have to be "type compatible”

CARTESIAN PRODUCT

Cross-Product

<u>sid</u>	<u>bid</u>	<u>day</u>
22	101	10/10/96
58	103	11/12/96

R1(sid,bid,day)

<u>sid</u>	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S1(sid,sname,rating,age)

Schema of cross product

Result(R1.sid,bid,day,S1.sid,sname,rating,age)

Renaming attribute

CARTESIAN PRODUCT

Pair each tuple t1 of R1 with each tuple t2 of S1.

sid	bid	day
22	101	10/10/96
58	103	11/12/96

R1(sid,bid,day)

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S1(sid,sname,rating,age)

	(sid)	sname	rating	age	(sid)	bid	day
1	22	dustin	7	45.0	22	101	10/10/96
2	22	dustin	7	45.0	58	103	11/12/96
3	31	lubber	8	55.5	22	101	10/10/96
4	31	lubber	8	55.5	58	103	11/12/96
5	58	rusty	10	35.0	22	101	10/10/96
6	58	rusty	10	35.0	58	103	11/12/96

Relational Algebra Operations from Set Theory: CARTESIAN PRODUCT

- Generally, CROSS PRODUCT is **not a meaningful operation**
  - Can become meaningful when followed by other operations
- Example (not meaningful)
  - FEMALE_EMPS $\leftarrow \sigma_{SEX='F'}$(EMPLOYEE)
  - EMPNAMES $\leftarrow \pi_{FNAME, LNAME, SSN}$ (FEMALE_EMPS)
  - EMP_DEPENDENTS $\leftarrow$ EMPNAMES x DEPENDENT
- EMP_DEPENDENTS will contain every combination of EMPNAMES and DEPENDENT
  - whether or not they are actually related

Relational Algebra Operations from Set Theory: CARTESIAN PRODUCT

- To keep only combinations where the DEPENDENT is related to the EMPLOYEE, add a SELECT operation as follows

- Example (meaningful)
  - FEMALE_EMPS $\leftarrow \sigma_{\text{SEX='F'}}$(EMPLOYEE)
  - EMPNAMES $\leftarrow \pi_{\text{FNAME, LNAME, SSN}}$ (FEMALE_EMPS)
  - EMP_DEPENDENTS $\leftarrow$ EMPNAMES x DEPENDENT
  - ACTUAL_DEPS $\leftarrow \sigma_{\text{SSN=ESSN}}$(EMP_DEPENDENTS)
  - RESULT $\leftarrow \pi_{\text{FNAME, LNAME, DEPENDENT\_NAME}}$ (ACTUAL_DEPS)

- RESULT will now contain the name of female employees and their dependents

CARTESIAN PRODUCT - SELECTION

S1.sid R1.sid

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

S1.sid > R1.sid

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	58	103	11/12/96

Binary Relational Operations: JOIN

- JOIN Operation (denoted by $\bowtie$)
  - The sequence of CARTESIAN PRODECT followed by SELECT is used quite commonly to identify and select related tuples from two relations
  - A special operation, called JOIN combines this sequence into a single operation
  - This operation is very important for any relational database with more than a single relation, because it allows us *combine related tuples* from various relations
  - The general form of a join operation on two relations R(A1, A2, . . ., An) and S(B1, B2, . . ., Bm) is:

$$R \bowtie_{\text{<join condition>}} S$$

  - where R and S can be any relations that result from general *relational algebra expressions*

Binary Relational Operations: JOIN

- Example: Suppose that we want to retrieve the name of the manager of each department
  - To get the manager's name, we need to combine each DEPARTMENT tuple with the EMPLOYEE tuple whose SSN value matches the MGRSSN value in the department tuple.
  - We do this by using the join $\bowtie$ operation
  - DEPT_MGR $\leftarrow$ DEPARTMENT $\bowtie_{\text{MGRSSN=SSN}}$ EMPLOYEE
- MGRSSN=SSN is the join condition
  - Combines each department record with the employee who manages the department
  - The join condition can also be specified as DEPARTMENT.MGRSSN= EMPLOYEE.SSN

Joins

$$R \bowtie_c S = \sigma_c (R \times S)$$

- *Condition Join*:

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	58	103	11/12/96

$$S1 \bowtie_{S1.sid < R1.sid} R1$$

- *Result schema* same as that of cross-product.
- Fewer tuples than cross-product, might be able to compute more efficiently
- Sometimes called a *theta-join*.

Some properties of JOIN

- Consider the following JOIN operation:
  - $R(A1, A2, \ldots, An) \bowtie_{R.Ai=S.Bj} S(B1, B2, \ldots, Bm)$
  - Result is a relation Q with degree n + m attributes:
    - Q(A1, A2, . . ., An, B1, B2, . . ., Bm), in that order.
  - The resulting relation state has one tuple for each combination of tuples—r from R and s from S, but *only if they satisfy the join condition* r[Ai]=s[Bj]
  - Hence, if R has n_R tuples, and S has n_S tuples, then the join result will generally have *less than* $n_R * n_S$ tuples
  - Only related tuples (based on the join condition) will appear in the result

Some properties of JOIN

- The general case of JOIN operation is called a Theta-join: $R \bowtie_{\textit{theta}} S$
- The join condition is called *theta*
- *Theta* can be any general boolean expression on the attributes of R and S;
  - For example

    R.Ai<S.Bj AND (R.Ak=S.Bl OR R.Ap<S.Bq)

- Most join conditions involve one or more equality conditions “AND”ed together;
  - For example:

    R.Ai=S.Bj AND R.Ak=S.Bl AND R.Ap=S.Bq

Join : Example

⋈ join   Account ⋈ $_{(\text{Number=Account and Amount>700})}$ Deposit

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Deposit	Account	T-id	Date	Amount
	102	1	10/22/00	500.00
	102	2	10/29/00	200.00
	104	3	10/29/00	1000.00
	105	4	11/2/00	10,000.00

	Number	Owner	Balance	Type	Account	T-id	Date	Amount
	104	M. Jones	1000.00	checking	104	3	10/29/00	1000.00
	105	H. Martin	10,000.00	checking	105	4	11/2/00	10000.00

Binary Relational Operations: EQUIJOIN

- The most common use of join involves join conditions with *equality comparisons* only
- Such a join, where the only comparison operator used is =, is called an EQUIJOIN
  - In the result of an EQUIJOIN we always have one or more pairs of attributes (whose names need not be identical) that have identical values in every tuple

- Special case of condition join where the condition *c* contains only **equalities**
- *Result schema* similar to cross-product
  - but only one copy of fields for which equality is specified

Binary Relational Operations: EQUIJOIN

⋈ = join

Account ⋈ $_{\text{Number=Account}}$ Deposit

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Deposit	Account	Transaction-id	Date	Amount
	102	1	10/22/00	500.00
	102	2	10/29/00	200.00
	104	3	10/29/00	1000.00
	105	4	11/2/00	10,000.00

Number	Owner	Balance	Type	Account	Transaction-id	Date	Amount
102	W. Wei	2000.00	checking	102	1	10/22/00	500.00
102	W. Wei	2000.00	checking	102	2	10/29/00	200.00
104	M. Jones	1000.00	checking	104	3	10/29/00	1000.00
105	H. Martin	10,000.00	checking	105	4	11/2/00	10000.00

Binary Relational Operations: EQUIJOIN

$\bowtie$ join

Account $\bowtie_{\text{Number=Account}}$ Deposit

Note that when the join is based on equality, then we have two identical attributes (columns) in the answer.

Number	Owner	Balance	Type	Account	Trans-id	Date	Amount
102	W. Wei	2000.00	checking	102	1	10/22/00	500.00
102	W. Wei	2000.00	checking	102	2	10/29/00	200.00
104	M. Jones	1000.00	checking	104	3	10/29/00	1000.00
105	H. Martin	10,000.00	checking	105	4	11/2/00	10000.00

EQUIJOIN

Pair each tuple t1 of R1 with each tuple t2 of S1.

sid	bid	day
22	101	10/10/96
58	103	11/12/96

R1(sid,bid,day)

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S1(sid,sname,rating,age)

	(sid)	sname	rating	age	(sid)	bid	day
1	22	dustin	7	45.0	22	101	10/10/96
2	22	dustin	7	45.0	58	103	11/12/96
3	31	lubber	8	55.5	22	101	10/10/96
4	31	lubber	8	55.5	58	103	11/12/96
5	58	rusty	10	35.0	22	101	10/10/96
6	58	rusty	10	35.0	58	103	11/12/96

S1.sid R1.sid

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

S1.sid = R1.sid

sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10/10/96
58	rusty	10	35.0	103	11/12/96

Binary Relational Operations: NATURAL JOIN Operation

- Another variation of JOIN called NATURAL JOIN
  - denoted by *
  - was created to get rid of the second (superfluous) attribute in an EQUIJOIN condition
    - because one of each pair of attributes with identical values is superfluous
  - The standard definition of natural join requires that the two join attributes, or each pair of corresponding join attributes, *have the same name* in both relations
  - If this is not the case, a renaming operation is applied first

Binary Relational Operations : NATURAL JOIN

- Example: To apply a natural join on the DNUMBER attributes of DEPARTMENT and DEPT_LOCATIONS, it is sufficient to write:
  - DEPT_LOCS ← DEPARTMENT * DEPT_LOCATIONS
- Only attribute with the same name is DNUMBER
- An implicit join condition is created based on this attribute:
  DEPARTMENT.DNUMBER=DEPT_LOCATIONS.DNUMBER

- Another example: Q ← R(A,B,C,D) * S(C,D,E)
  - The implicit join condition includes *each pair* of attributes with the same name, "AND"ed together:
    - R.C=S.C AND R.D.S.D
  - Result keeps only one attribute of each such pair:
    - Q(A,B,C,D,E)

Natural Join

- *Natural Join*: Equijoin on *all* common fields.

sid	bid	day
22	101	10/10/96
58	103	11/12/96

R1(sid,bid,day)

$$S1 \bowtie R1$$

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S1(sid,sname,rating,age)

sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10/10/96
58	rusty	10	35.0	103	11/12/96

Question

- How could you express the natural join operation if you didn't have a natural join operator in relational algebra? Consider you have two relations R(A,B,C) and S(B,C,D)

$$\pi_{R.A,R.B,R.C,\ S.D}\left(\sigma_{R.B=S.B \text{ and } R.C=S.C}\ (R \times S)\right)$$

Complete Set of Relational Operations

- The set of operations including SELECT σ, PROJECT π , UNION $\cup$, DIFFERENCE $-$, RENAME ρ, and CARTESIAN PRODUCT X is called a complete set
  - Any other relational algebra expression can be expressed by a combination of these five operations

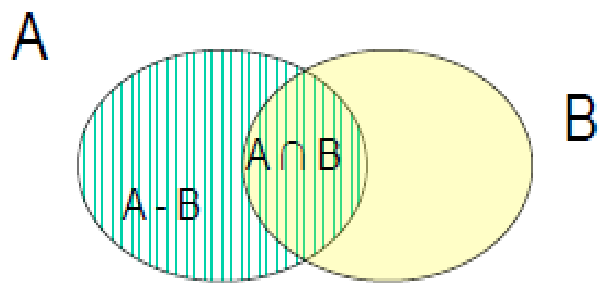
- For example
  - $R \cap S = (R \cup S) - ((R - S) \cup (S - R))$
  - $R \bowtie_{\text{<join condition>}} S = \sigma_{\text{<join condition>}} (R \text{ X } S)$

Challenging Question

- How could you express the intersection operation if you didn't have an Intersection operator in relational algebra?
- [Hint: Can you express Intersection using only the Difference operator?]
  - $A \cap B$ = ???

Challenging Question

- How could you express the intersection operation if you didn't have an Intersection operator in relational algebra?
- [Hint: Can you express Intersection using only the Difference operator?]
  - $A \cap B = A - (A - B)$



Binary Relational Operations: DIVISION

- Binary operator - R ÷ S
- Attributes of S must be a subset of the attributes of R
- attr(R ÷ S) = attr(R) – attr(S)
- t tuple in (R ÷ S) iff (t × S) is a subset of R
- Used to answer questions involving all
  - e.g., Which employees work on all the critical projects?

Works(enum,pnum) Critical(pnum)

Works

enum	pnum
E35	P10
E45	P15
E35	P12
E52	P15
E52	P17
E45	P10
E35	P15

Critical

pnum
P15
P10

Works ÷ Critical

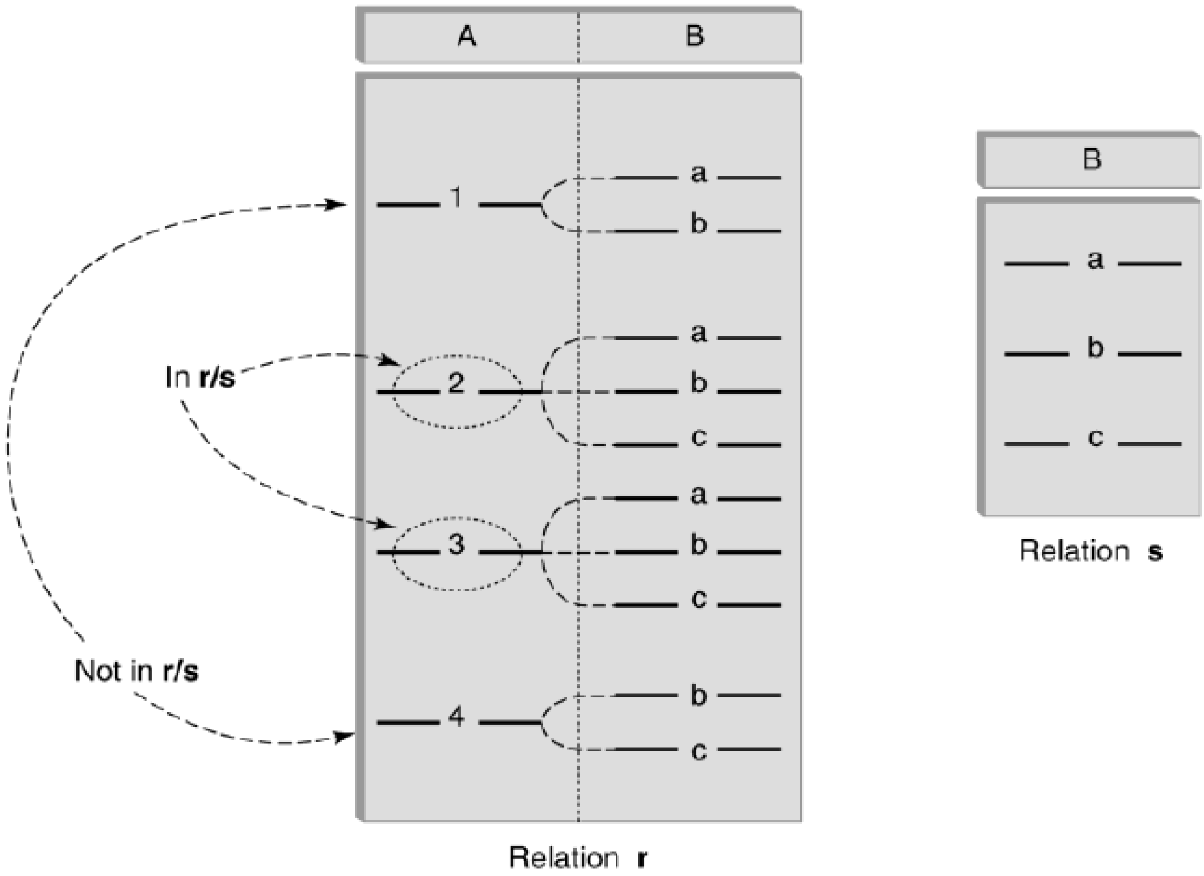
enum
E45
E35

(Works ÷ Critical) × Critical

enum	pnum
E45	P15
E45	P10
E35	P15
E35	P10

  - “Inverse” of cross product

Binary Relational Operations: DIVISION



Binary Relational Operations: DIVISION

$\div$ or / divide $\quad (\pi_{\text{Owner, Type}}\ \text{Account}) \div \text{Account-types}$

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00checking	

Account-types	Type
	checking
	savings

	Owner
	J. Smith

Find account owners who have ALL types of accounts.

Binary Relational Operations: DIVISION

÷ or / divide $(\pi_{\text{Owner, Type}}\, \text{Account}) \div \text{Account-types}$

Account	Number	Owner	Balance	Type
	101	J. Smith	1000.00	checking
	102	W. Wei	2000.00	checking
	103	J. Smith	5000.00	savings
	104	M. Jones	1000.00	checking
	105	H. Martin	10,000.00	checking

Account-types	Type
	checking
	savings

	Owner
	J. Smith

Find account owners who have ALL types of accounts.

Divide Operator

- For R ÷ S where **R** (r1, r2, r3, r4) and **S**(s1, s2)

- Since S has two attributes, there must be two attributes in R (say r3 and r4) that are defined on the same domains, respectively, as s1 and s2.
  - We could say that (r3, r4) is *union-compatible* with (s1, s2)

- The query answer has the remaining attributes (r1, r2) And the answer has a tuple (r1, r2) in the answer if the (r1, r2) value appears with *every* S tuple in R

Binary Relational Operations: DIVISION

- The DIVISION operation - useful for a special kind of query that sometimes occurs in database applications
- Example - *Retrieve the names of employees who work on* ***all*** *the projects that 'John Smith' works on*

$$\text{SMITH} \leftarrow \sigma_{\text{Fname='John' AND Lname='Smith'}}(\text{EMPLOYEE})$$
$$\text{SMITH\_PNOS} \leftarrow \pi_{\text{Pno}}(\text{WORKS\_ON} \bowtie_{\text{Essn=Ssn}} \text{SMITH})$$

$$\text{SSN\_PNOS} \leftarrow \pi_{\text{Essn, Pno}}(\text{WORKS\_ON})$$

$$\text{SSNS(Ssn)} \leftarrow \text{SSN\_PNOS} \div \text{SMITH\_PNOS}$$
$$\text{RESULT} \leftarrow \pi_{\text{Fname, Lname}}(\text{SSNS} * \text{EMPLOYEE})$$

Example of DIVISION

(a)

SSN_PNOS

Essn	Pno
123456789	1
123456789	2
666884444	3
453453453	1
453453453	2
333445555	2
333445555	3
333445555	10
333445555	20
999887777	30
999887777	10
987987987	10
987987987	30
987654321	30
987654321	20
888665555	20

SMITH_PNOS

Pno
1
2

SSNS

Ssn
123456789
453453453

(b)

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

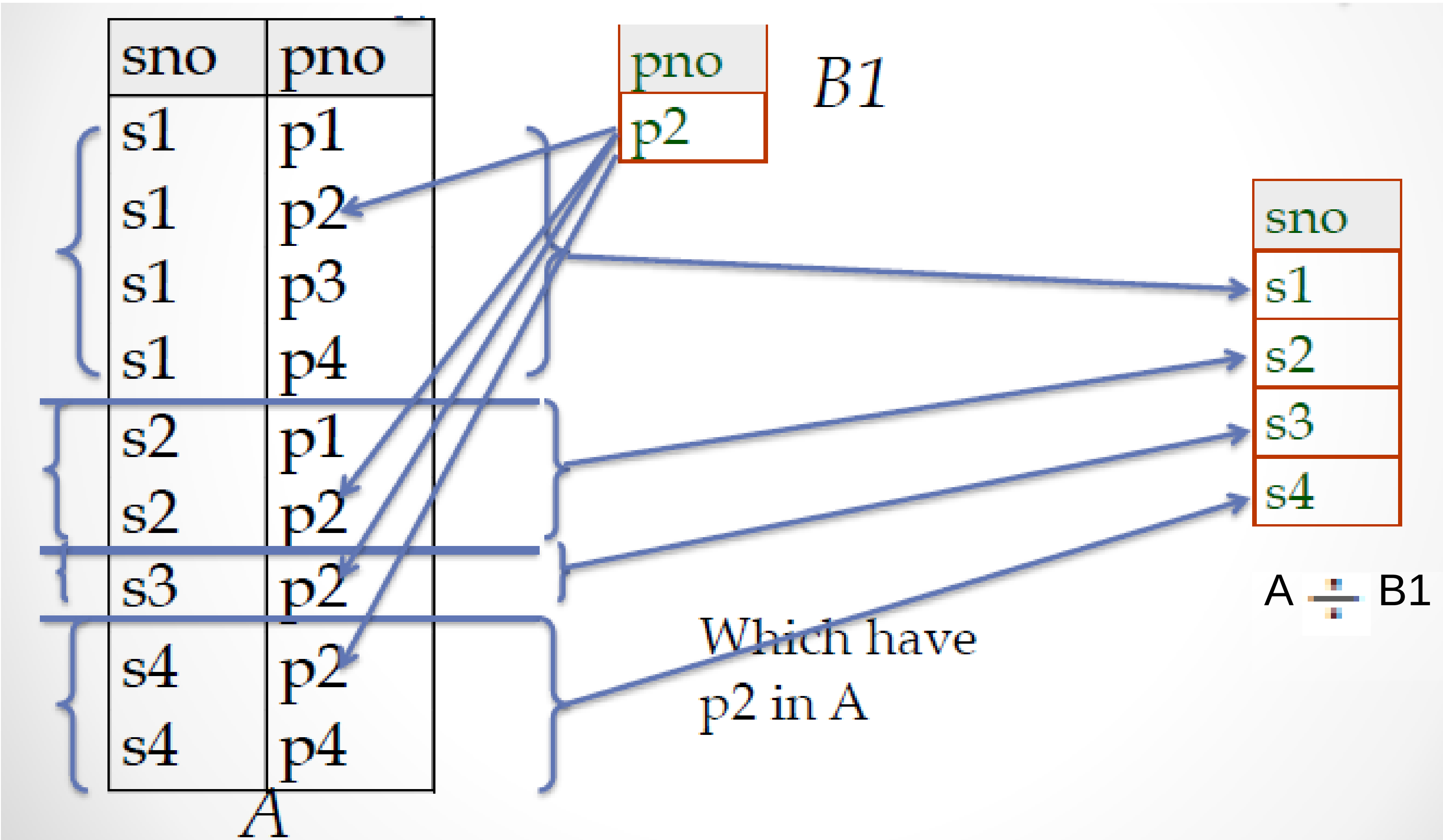
A
a1
a2
a3

T

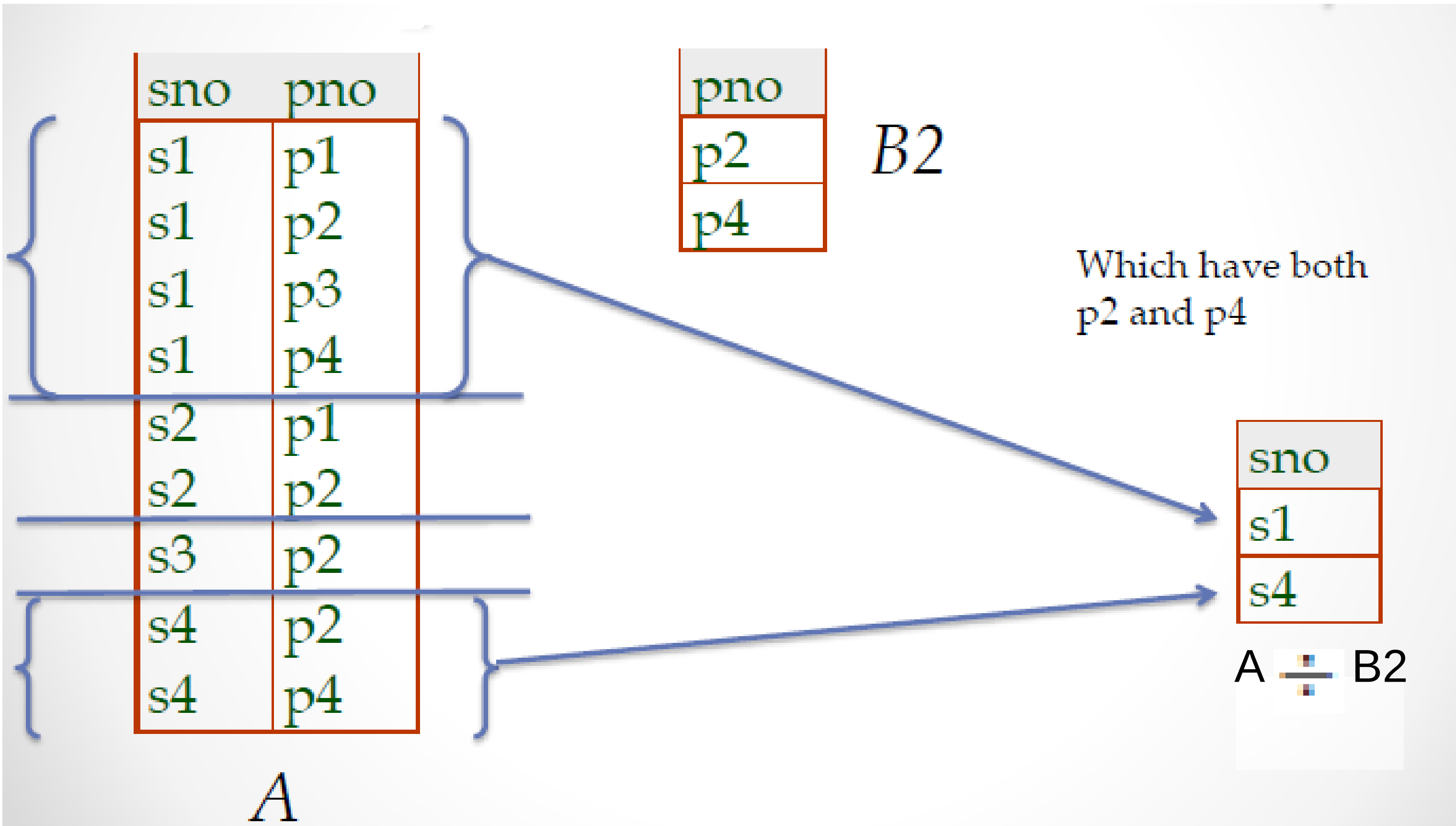
B
b1
b4

Figure 6.8
The DIVISION operation. (a) Dividing SSN_PNOS by SMITH_PNOS. (b) $T \leftarrow R \div S$.

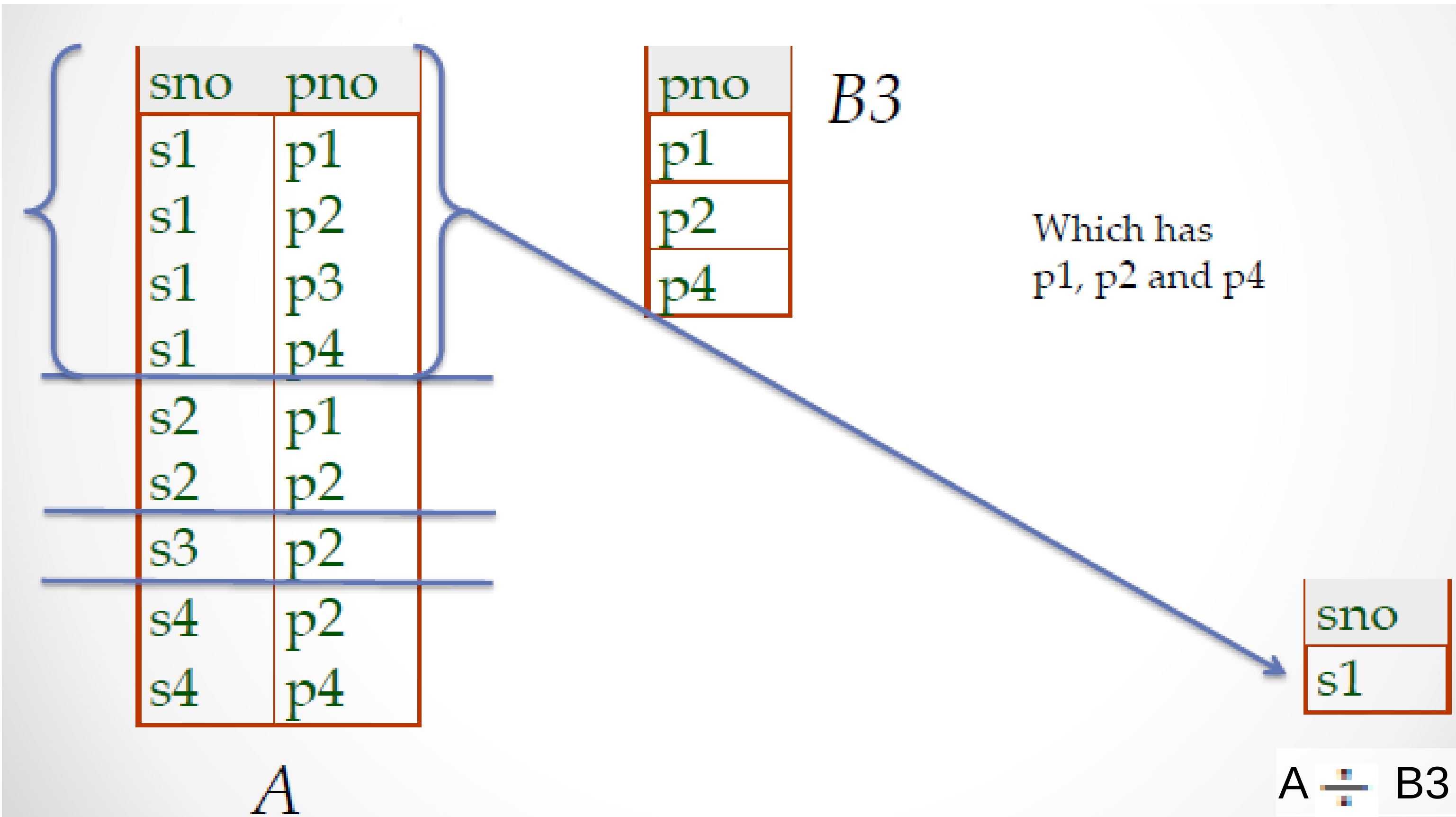
Binary Relational Operations: DIVISION



Binary Relational Operations: DIVISION



Binary Relational Operations: DIVISION



Expressing A/B Using Basic Operators

- Division is not an essential operator, but it provides a useful shorthand
- Example

Disqualified x values: $\pi_x((\pi_x(A) \times B) - A)$

A/B: $\pi_x(A)$ − all disqualified tuples

  - A = ((s1,p1), (s1,p2),(s2,p1),(s3,p2))
  - B = (p1,p2)
  - A/B = ???
  - $\pi_X(A)$ = (s1,s2,s3) – duplicates are removed!
  - $\pi_X(A)$ X B = ((s1,p1),(s1,p2),(s2,p1),(s2,p2),(s3,p1),(s3,p2))
  - ($\pi_X(A)$ X B) – A = ((s2,p2),(s3,p1))
  - $\pi_X((\pi_X(A)$ X B) – A) = (s2,s3) ← disqualified tuples
  - A/B = (s1,s2,s3) – (s2,s3) = (s1)

Recap of Relational Algebra Operations

OPERATION	PURPOSE	NOTATION
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{\text{<selection condition>}}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{\text{<attribute list>}}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{\text{<join condition>}} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{\text{<join condition>}} R_2$, OR $R_1 \bowtie_{(\text{<join attributes 1>}),\ (\text{<join attributes 2>})} R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 *_{\text{<join condition>}} R_2$, OR $R_1 *_{(\text{<join attributes 1>}),\ (\text{<join attributes 2>})} R_2$ OR $R_1 * R_2$
UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$

Recap of Relational Algebra Operations

INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2.	$R_1 \times R_2$
DIVISION	Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

Building Complex Expressions

- Combine operators with parentheses and precedence rules
- Three notations, just as in arithmetic
  - 1. Sequences of assignment statements
  - 2. Expressions with several operators
  - 3. Expression trees

Sequences of Assignments

- Create temporary relation names
  - Renaming can be implied by giving relations a list of attributes
- Example
  - $R3 := R1 \bowtie_C R2$ can be written

    $R4 := R1 \times R2$

    $R3 := \sigma_C (R4)$
- Example: Write $r \div s$ as

  $temp1 := \Pi_{R-S} (r)$

  $temp2 := \Pi_{R-S} ((temp1 \times s) - \Pi_{R-S,S} (r))$

  $result := temp1 - temp2$

Expressions in a Single Assignment

- Example
  - The theta-join $R3 := R1 \bowtie_C R2$ can be written $R3 := \sigma_C (R1 \text{ X } R2)$
- Precedence of relational operators
  - 1. [σ, π, ρ] (highest)
  - 2. [x, ⋈]
  - 3. ∩
  - 4. [∪, —]

Expression Trees

- Leaves are operands - either variables standing for relations or particular, constant relations
- Interior nodes are operators, applied to their child or children

- Example: Tree for a Query
- Using the relations

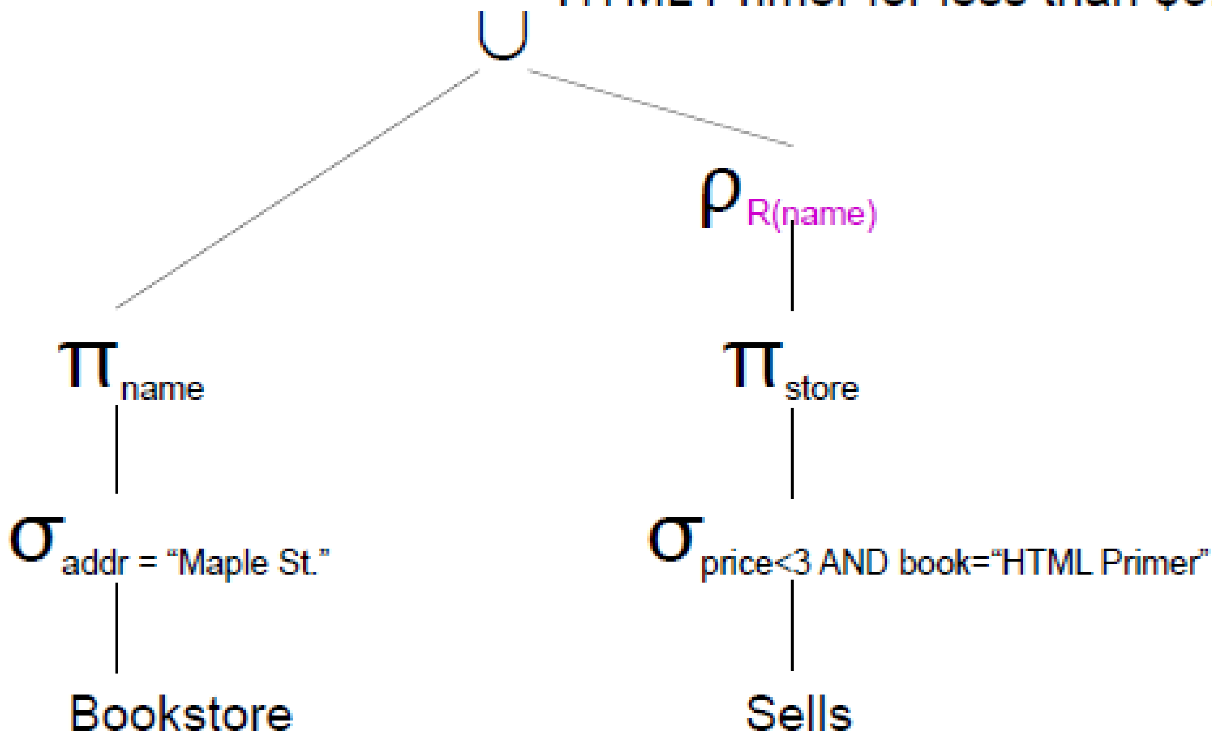
  Bookstore(name, addr) and

  Sells(store, book, price)

  Find the names of all the stores that are either on Maple St. or sell HTML Primer for less than $3

As a Tree:

Bookstore(name, addr) and Sells(store, book, price), find the names of all the stores that are either on Maple St. or sell HTML Primer for less than $3.



Additional Relational Operations: Generalized Projection

- Extends the projection operation by allowing functions of attributes to be included in the projection list
- General form : $\pi_{F1,\ F2,\ ...,\ Fn}\ (R)$
  - where *F*1, *F*2, ..., *Fn* are functions over the attributes in relation *R* and may involve arithmetic operations and constant values
- Example
  - EMPLOYEE (Ssn, Salary, Deduction, Years_service)
- A report may be required to show
  - Net Salary = Salary – Deduction,
  - Bonus = 2000 * Years_service, and
  - Tax = 0.25 * Salary
- Generalized projection combined with renaming
  - REPORT ← ρ(Ssn, Net_salary, Bonus, Tax)($\pi_{\text{Ssn, Salary – Deduction, 2000 * Years\_service 0.25 * Salary}}$(EMPLOYEE))

Additional Relational Operations : Aggregate Functions and Grouping

- A type of request that cannot be expressed in the basic relational algebra is to specify mathematical **aggregate functions** on collections of values from the database
- Examples
  - Retrieving the average or total salary of all employees or the total number of employee tuples
- These functions are used in simple statistical queries that summarize information from the database tuples
- Common functions applied to collections of numeric values include
  - SUM - sum of values
  - AVERAGE - average value
  - MAXIMUM – maximum value
  - MINIMUM – minimum value
- The COUNT function is used for counting tuples or values
  - Count – number of values

Aggregate Function Operation

- Use of the Aggregate Functional operation $\mathcal{F}$
  - $\mathcal{F}_{\text{MAX Salary}}$ (EMPLOYEE) - retrieves the maximum salary value from the EMPLOYEE relation
  - $\mathcal{F}_{\text{MIN Salary}}$ (EMPLOYEE) - retrieves the minimum Salary value from the EMPLOYEE relation
  - $\mathcal{F}_{\text{SUM Salary}}$ (EMPLOYEE) - retrieves the sum of the Salary from the EMPLOYEE relation
  - $\mathcal{F}_{\text{COUNT SSN, AVERAGE Salary}}$ (EMPLOYEE) - computes the count (number) of employees and their average salary
    - Note: count just counts the number of rows, **without removing duplicates**

Using Grouping with Aggregation

- Grouping can be combined with Aggregate Functions
- Example: For each department, retrieve the DNO, COUNT SSN, and AVERAGE SALARY

- A variation of aggregate operation $\mathcal{F}$ allows this:
  - **Grouping attribute** placed to **left of symbol**
  - **Aggregate functions** to **right of symbol**

    ${}_{\text{DNO}}\ \mathcal{F}_{\text{COUNT SSN, AVERAGE Salary}}\ (\text{EMPLOYEE})$

  - Above operation groups employees by DNO (department number) and computes the count of employees and average salary per department

Examples of applying aggregate functions and grouping

Figure 6.10
The aggregate function operation.
(a) $\rho_{R(\text{Dno, No\_of\_employees, Average\_sal})}\left({}_{\text{Dno}}\,\Im\,{}_{\text{COUNT Ssn, AVERAGE Salary}}(\text{EMPLOYEE})\right)$.
(b) ${}_{\text{Dno}}\,\Im\,{}_{\text{COUNT Ssn, AVERAGE Salary}}(\text{EMPLOYEE})$.
(c) $\Im\,{}_{\text{COUNT Ssn, AVERAGE Salary}}(\text{EMPLOYEE})$.

R

(a)

Dno	No_of_employees	Average_sal
5	4	33250
4	3	31000
1	1	55000

(b)

Dno	Count_ssn	Average_salary
5	4	33250
4	3	31000
1	1	55000

(c)

Count_ssn	Average_salary
8	35125

Aggregate Operation – Example

- Relation *r*:

A	*B*	*C*
α	α	7
α	β	7
β	β	3
β	β	10

$\mathcal{F}_{\text{SUM(C)}}\,(r)$

sum-C
27

Examples of applying aggregate functions and grouping

- Consider the following **ACCOUNT** relation

branch-name	*account-number*	*balance*
Perryridge	A-102	400
Perryridge	A-201	900
Brighton	A-217	750
Brighton	A-215	750
Redwood	A-222	700

- ρ $_{branch\text{-}name}\,\mathcal{F}\,_{sum(balance)}\,(ACCOUNT)$

branch-name	*balance*
Perryridge	1300
Brighton	1500
Redwood	700

Additional Relational Operations

- Recursive Closure Operations
  - Another type of operation that, in general, cannot be specified in the basic original relational algebra is **recursive closure**
  - This operation is applied to a **recursive relationship**
  - An example of a recursive operation is to retrieve all SUPERVISEES of an EMPLOYEE **e** at all levels — that is,
    - all EMPLOYEE **e'** directly supervised by **e**
    - all employees **e''** directly supervised by each employee **e'**
    - all employees **e'''** directly supervised by each employee **e''**
    - …

Additional Relational Operations

- It is relatively straightforward in the relational algebra to specify all employees supervised by *e at a specific level* by joining the table with itself one or more times
- However, it is difficult to specify all supervisees at *all* levels
- For example
  - to specify the Ssns of all employees *e* directly supervised—***at level one***—by the employee *e* whose name is 'James Borg',

$$\text{BORG\_SSN} \leftarrow \pi_{\text{Ssn}}(\sigma_{\text{Fname}=\text{'James'} \text{ AND } \text{Lname}=\text{'Borg'}}(\text{EMPLOYEE}))$$
$$\text{SUPERVISION}(\text{Ssn1}, \text{Ssn2}) \leftarrow \pi_{\text{Ssn},\text{Super\_ssn}}(\text{EMPLOYEE})$$
$$\text{RESULT1}(\text{Ssn}) \leftarrow \pi_{\text{Ssn1}}(\text{SUPERVISION} \bowtie_{\text{Ssn2}=\text{Ssn}} \text{BORG\_SSN})$$

Additional Relational Operations

- To retrieve all employees supervised by Borg **at level 2**—that is, all employees *e* supervised by some employee *e* who is directly supervised by Borg—we can apply another **JOIN** to the result of the first query, as follows:

$$\mathrm{RESULT2}(\mathrm{Ssn}) \leftarrow \pi_{\mathrm{Ssn1}}(\mathrm{SUPERVISION} \bowtie_{\mathrm{Ssn2=Ssn}} \mathrm{RESULT1})$$

- To get both sets of employees supervised at levels 1 and 2 by 'James Borg', we can apply the UNION operation to the two results, as follows:

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

Additional Relational Operations

(Borg's SSN is 888665555)

SUPERVISION	SSN1 (SSN)	SSN2 (SUPERSSN)
	123456789	333445555
	333445555	888665555
	999887777	987654321
	987654321	888665555
	666884444	333445555
	453453453	333445555
	987987987	987654321

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 $\cup$ RESULT2)

Outer Join Operations

- JOIN operations – include tuples that satisfy the join condition
  - For example, for a NATURAL JOIN operation *R* * *S*, only tuples from *R* that have matching tuples in *S*-and vice versa-appear in the result.
  - Hence, tuples without a *matching* (or *related*) tuple are eliminated from the JOIN result -
  - Tuples with NULL values in the join attributes are also eliminated
  - This type of join, where tuples with no match are eliminated, is known as an **inner join**
- **Inner Joins**
  - Theta Join
  - Equi Join
  - Natural Join
- **Outer Joins -** An extension of the join operation that avoids **loss of information**

Outer Join Operations

- Three Outer Joins
  - **Left outer Join** - $⟕$
  - **Right outer Join** - $⟖$
  - **Full Outer Join** - $⟗$
- necessary to specify certain types of queries
- For example
  - Display the list of all employee names as well as the name of the departments they manage (*if they happen to manage a department),* if they do not manage one, we can indicate it with a NULL value
- Apply an operation **LEFT OUTER JOIN** to retrieve the result

$$\text{TEMP} \leftarrow (\text{EMPLOYEE} ⟕_{\text{Ssn}=\text{Mgr\_ssn}} \text{DEPARTMENT})$$

$$\text{RESULT} \leftarrow \pi_{\text{Fname, Minit, Lname, Dname}}(\text{TEMP})$$

Outer Join Operations

- Left outer join
  - The left outer join operation keeps every tuple in the first or left relation R in R ⟕ S; if no matching tuple is found in S, then the attributes of S in the join result are filled or “padded” with null values
- Right outer join
  - A similar operation, right outer join, keeps every tuple in the second or right relation S in the result of R ⟖ S
- Full outer join
  - R ⟗ S, keeps all tuples in both the left and the right relations when no matching tuples are found, padding them with null values as needed

Outer Join - Example

- Relation *loan*

loan-number	*branch-name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

- Relation *borrower*

customer-name	*loan-number*
Jones	L-170
Smith	L-230
Hayes	L-155

- **Inner Join**

loan ⋈ *Borrower*

loan-number	*branch-name*	*amount*	*customer-name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith

- **Left Outer Join**

loan ⟕ *Borrower*

loan-number	*branch-name*	*amount*	*customer-name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*

- **Right Outer Join**

loan ⟖ *borrower*

loan-number	*branch-name*	*amount*	*customer-name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	*null*	*null*	Hayes

- **Full Outer Join**

loan ⟗ *borrower*

loan-number	*branch-name*	*amount*	*customer-name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*
L-155	*null*	*null*	Hayes

OUTER UNION Operations

- The outer union operation was developed to take the union of tuples from two relations if the relations are *not type compatible*
- This operation will take the union of tuples in two relations R(X, Y) and S(X, Z) that are **partially compatible**, meaning that only some of their attributes, say X, are type compatible
- The attributes that are type compatible are represented only once in the result, and those attributes that are not type compatible from either relation are also kept in the result relation T(X, Y, Z)

- Example: An outer union can be applied to two relations whose schemas are STUDENT(Name, SSN, Department, Advisor) and INSTRUCTOR(Name, SSN, Department, Rank)
  - Tuples from the two relations are matched based on having the same combination of values of the shared attributes - Name, SSN, Department
  - If a student is also an instructor, both Advisor and Rank will have a value; otherwise, one of these two attributes will be null
  - The result relation STUDENT_OR_INSTRUCTOR will have the following attributes:

STUDENT_OR_INSTRUCTOR (Name, SSN, Department, Advisor, Rank)

Relational Algebra consists of several groups of operations

- **Unary Relational Operations**
  - SELECT (symbol: **σ** (sigma))
  - PROJECT (symbol: **π** (pi))
  - RENAME (symbol: **ρ** (rho))
- **Relational Algebra Operations From Set Theory**
  - UNION (∪), INTERSECTION (∩), DIFFERENCE (**–**), CARTESIAN PRODUCT (**x**)
- **Binary Relational Operations**
  - JOIN (several variations of JOIN exist)
  - DIVISION
- **Additional Relational Operations**
  - OUTER JOINS, OUTER UNION
  - AGGREGATE FUNCTIONS (These compute summary of information) For example, SUM, COUNT, AVG, MIN, MAX

Summary

- The relational model has rigorously defined query languages that are simple and powerful
- Relational algebra is more operational; useful as internal representation for query evaluation plans
- Several ways of expressing a given query; a query optimizer should choose the most efficient version