

# **CHAPTER 8**

## **The Relational Algebra and The Relational Calculus (plus QBE- Appendix C)**

# Chapter Outline

- Relational Algebra
  - Unary Relational Operations
  - Relational Algebra Operations From Set Theory
  - Binary Relational Operations
  - Additional Relational Operations
  - Examples of Queries in Relational Algebra
- Relational Calculus
  - Tuple Relational Calculus
  - Domain Relational Calculus
- Example Database Application (COMPANY)
- Overview of the QBE language (appendix D)

# 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)

# Relational Algebra Overview (continued)

- 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)

# Brief History of Origins of Algebra

- Muhammad ibn Musa al-Khwarizmi (800-847 CE) – from Morocco wrote a book titled al-jabr about arithmetic of variables
  - Book was translated into Latin.
  - Its title (al-jabr) gave Algebra its name.
- Al-Khwarizmi called variables “shay”
  - “Shay” is Arabic for “thing”.
  - Spanish transliterated “shay” as “xay” (“x” was “sh” in Spain).
  - In time this word was abbreviated as x.
- Where does the word Algorithm come from?
  - Algorithm originates from “al-Khwarizmi”
  - Reference: PBS (<http://www.pbs.org/empires/islam/innoalgebra.html>)

# Relational Algebra Overview

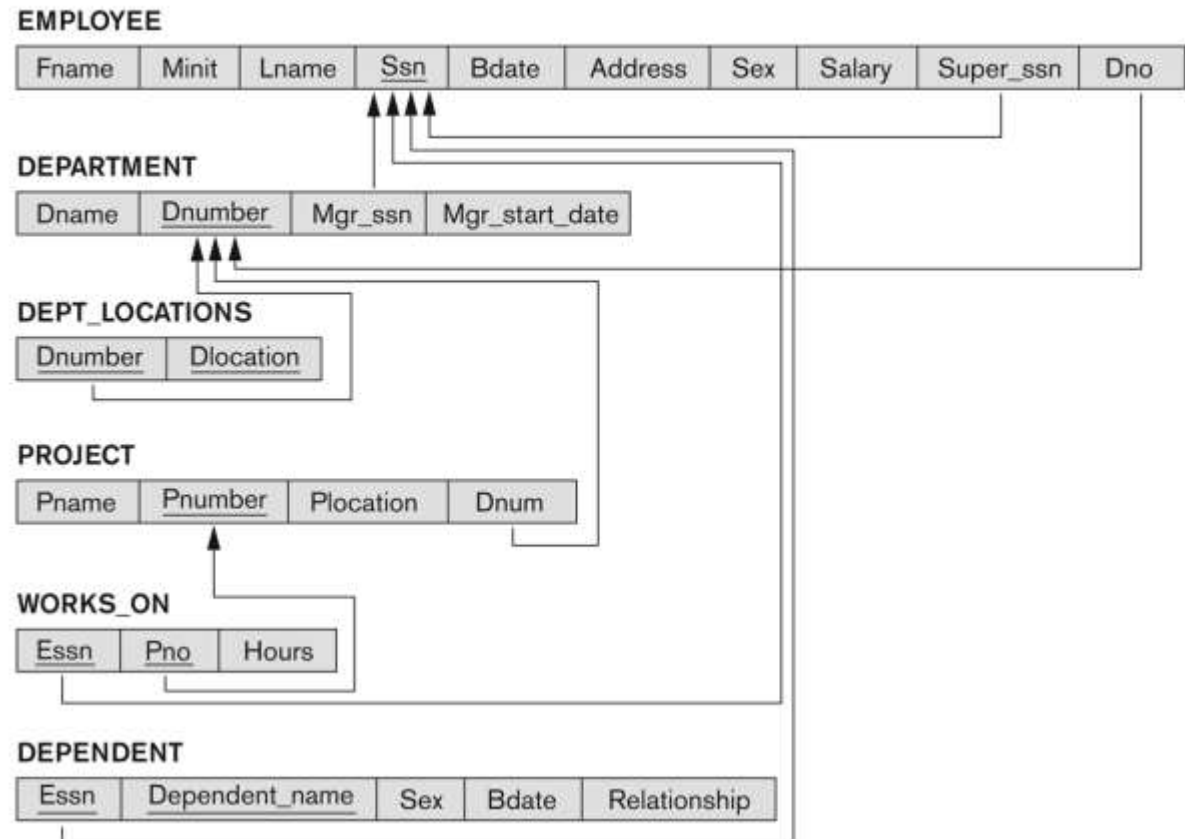
- Relational Algebra consists of several groups of operations
  - Unary Relational Operations
    - SELECT (symbol:  $\sigma$  (sigma))
    - PROJECT (symbol:  $\pi$  (pi))
    - RENAME (symbol:  $\rho$  (rho))
  - Relational Algebra Operations From Set Theory
    - UNION ( $\cup$ ), INTERSECTION ( $\cap$ ), DIFFERENCE (or MINUS,  $-$ )
    - CARTESIAN PRODUCT ( $\times$ )
  - 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)

# Database State for COMPANY

- All examples discussed below refer to the COMPANY database shown here.

**Figure 5.7**

Referential integrity constraints displayed on the COMPANY relational database schema.





# Unary Relational Operations: SELECT

- The SELECT operation (denoted by  $\sigma$  (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*)

- Examples:

- Select the EMPLOYEE tuples whose department number is 4:

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

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

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

# Unary Relational Operations: SELECT

- In general, the *select* operation is denoted by  $\sigma_{\langle \text{selection condition} \rangle}(R)$  where
  - the symbol  $\sigma$  (sigma) is used to denote the *select* operator
  - the selection condition is a Boolean (conditional) expression specified on the attributes of relation R
  - 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 (continued)

## ■ SELECT Operation Properties

- The SELECT operation  $\sigma_{\langle \text{selection condition} \rangle}(R)$  produces a relation S that has the same schema (same attributes) as R
- SELECT  $\sigma$  is commutative:
  - $\sigma_{\langle \text{condition1} \rangle}(\sigma_{\langle \text{condition2} \rangle}(R)) = \sigma_{\langle \text{condition2} \rangle}(\sigma_{\langle \text{condition1} \rangle}(R))$
- Because of commutativity property, a cascade (sequence) of SELECT operations may be applied in any order:
  - $\sigma_{\langle \text{cond1} \rangle}(\sigma_{\langle \text{cond2} \rangle}(\sigma_{\langle \text{cond3} \rangle}(R))) = \sigma_{\langle \text{cond2} \rangle}(\sigma_{\langle \text{cond3} \rangle}(\sigma_{\langle \text{cond1} \rangle}(R)))$
- A cascade of SELECT operations may be replaced by a single selection with a conjunction of all the conditions:
  - $\sigma_{\langle \text{cond1} \rangle}(\sigma_{\langle \text{cond2} \rangle}(\sigma_{\langle \text{cond3} \rangle}(R))) = \sigma_{\langle \text{cond1} \rangle \text{ AND } \langle \text{cond2} \rangle \text{ AND } \langle \text{cond3} \rangle}(R)$
- 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

# The following query results refer to this database state

**Figure 5.6**

One possible database state for the COMPANY relational database schema.

**EMPLOYEE**

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-05-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

**DEPARTMENT**

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1968-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

**DEPT\_LOCATIONS**

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

**WORKS\_ON**

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

**PROJECT**

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

**DEPENDENT**

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

# Unary Relational Operations: PROJECT

- PROJECT Operation is denoted by  $\pi$  (pi)
- This operation keeps certain *columns* (attributes) from a relation and discards the other columns.
  - PROJECT creates a vertical partitioning
    - The list of specified columns (attributes) is kept in each tuple
    - The other attributes in each tuple are discarded
- Example: To list each employee's first and last name and salary, the following is used:

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

# Unary Relational Operations: PROJECT (cont.)

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

$$\pi_{\langle \text{attribute list} \rangle}(\mathbf{R})$$

- $\pi$  (pi) is the symbol used to represent the *project* operation
- $\langle \text{attribute list} \rangle$  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*
    - Mathematical sets *do not allow* duplicate elements.

# Unary Relational Operations: PROJECT (contd.)

## ■ PROJECT Operation Properties

- The number of tuples in the result of projection  $\pi_{\langle \text{list} \rangle}(R)$  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  $R$
- PROJECT is *not* commutative
  - $\pi_{\langle \text{list1} \rangle}(\pi_{\langle \text{list2} \rangle}(R)) = \pi_{\langle \text{list1} \rangle}(R)$  as long as  $\langle \text{list2} \rangle$  contains the attributes in  $\langle \text{list1} \rangle$

# Examples of applying SELECT and PROJECT operations

**Figure 8.1** Results of SELECT and PROJECT operations. (a)  $\sigma_{(Dno=4 \text{ AND } Salary > 25000) \text{ OR } (Dno=3 \text{ AND } Salary > 30000)}(EMPLOYEE)$ . (b)  $\pi_{Lname, Fname, Salary}(EMPLOYEE)$ . (c)  $\pi_{Sex, Salary}(EMPLOYEE)$ .

(a)

Fname	Minit	Lname	Sen	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, Bellairs, 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  $\rho$  (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)

# Unary Relational Operations: RENAME (continued)

- The general RENAME operation  $\rho$  can be expressed by any of the following forms:
  - $\rho_S(B_1, B_2, \dots, B_n)(R)$  changes both:
    - the relation name to  $S$ , *and*
    - the column (attribute) names to  $B_1, B_1, \dots, B_n$
  - $\rho_S(R)$  changes:
    - the *relation name* only to  $S$
  - $\rho_{(B_1, B_2, \dots, B_n)}(R)$  changes:
    - the *column (attribute) names* only to  $B_1, B_1, \dots, B_n$

# Unary Relational Operations: RENAME (continued)

- For convenience, we also use a *shorthand* for renaming attributes in an intermediate relation:
  - If we write:
    - $\text{RESULT} \leftarrow \pi_{\text{FNAME, LNAME, SALARY}}(\text{DEP5\_EMPS})$
    - RESULT will have the *same attribute names* as DEP5\_EMPS (same attributes as EMPLOYEE)
  - If we write:
    - $\text{RESULT}(\text{F, M, L, S, B, A, SX, SAL, SU, DNO}) \leftarrow \rho_{\text{RESULT}(\text{F.M.L.S.B,A,SX,SAL,SU,DNO})}(\text{DEP5\_EMPS})$
    - The 10 attributes of DEP5\_EMPS are *renamed* to F, M, L, S, B, A, SX, SAL, SU, DNO, respectively

**Note:** the  $\leftarrow$  symbol is an assignment operator

# Example of applying multiple operations and RENAME

**Figure 8.2** Results of a sequence of operations. (a)  $\pi_{Fname, Lname, Salary}(\sigma_{Dno=3}(EMPLOYEE))$ .  
(b) Using intermediate relations and renaming of attributes.

(a)

Fname	Lname	Salary
John	Smith	30000
Franklin	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

(b)

TEMP

Fname	Minit	Lname	SSN	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1985-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Ramesh	K	Narayan	666884444	1982-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

R

First_name	Last_name	Salary
John	Smith	30000
Franklin	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

# 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
    - Each pair of corresponding attributes must be type compatible (have same or compatible domains)

# 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:

$$\text{DEP5\_EMPS} \leftarrow \sigma_{\text{DNO}=5} (\text{EMPLOYEE})$$
$$\text{RESULT1} \leftarrow \pi_{\text{SSN}}(\text{DEP5\_EMPS})$$
$$\text{RESULT2}(\text{SSN}) \leftarrow \pi_{\text{SUPERSSN}}(\text{DEP5\_EMPS})$$
$$\text{RESULT} \leftarrow \text{RESULT1} \cup \text{RESULT2}$$

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



## Figure 8.3 Result of the UNION operation $\text{RESULT} \leftarrow \text{RESULT1} \cup \text{RESULT2}$ .

**RESULT1**

Ssn
123456789
333445555
666884444
453453453

**RESULT2**

Ssn
333445555
888665555

**RESULT**

Ssn
123456789
333445555
666884444
453453453
888665555

# 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  $-$ , see next slides)
- $R1(A1, A2, \dots, An)$  and  $R2(B1, B2, \dots, Bn)$  are type compatible if:
  - they have the same number of attributes, and
  - the domains of corresponding attributes are type compatible (i.e.  $\text{dom}(Ai) = \text{dom}(Bi)$  for  $i=1, 2, \dots, n$ ).
- The resulting relation for  $R1 \cup R2$  (also for  $R1 \cap R2$ , or  $R1 - R2$ , see next slides) 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: SET DIFFERENCE (cont.)

- 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”

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

**Figure 8.4** The set operations UNION, INTERSECTION, and MINUS. (a) Two union-compatible relations. (b)  $\text{STUDENT} \cup \text{INSTRUCTOR}$ . (c)  $\text{STUDENT} \cap \text{INSTRUCTOR}$ . (d)  $\text{STUDENT} - \text{INSTRUCTOR}$ . (e)  $\text{INSTRUCTOR} - \text{STUDENT}$ .

(a) STUDENT		INSTRUCTOR		(b)	
Fn	Ln	Fname	Lname	Fn	Ln
Susan	Yao	John	Smith	Susan	Yao
Ramesh	Shah	Ricardo	Browne	Ramesh	Shah
Johnny	Kohler	Susan	Yao	Johnny	Kohler
Barbara	Jones	Francis	Johnson	Barbara	Jones
Amy	Ford	Ramesh	Shah	Amy	Ford
Jimmy	Wang			Jimmy	Wang
Ernest	Gilbert			Ernest	Gilbert

(c)		(d)		(e)	
Fn	Ln	Fn	Ln	Fname	Lname
Susan	Yao	Johnny	Kohler	John	Smith
Ramesh	Shah	Barbara	Jones	Ricardo	Browne
		Amy	Ford	Francis	Johnson
		Jimmy	Wang		
		Ernest	Gilbert		

# 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  $R(A_1, A_2, \dots, A_n) \times S(B_1, B_2, \dots, B_m)$
  - Result is a relation  $Q$  with degree  $n + m$  attributes:
    - $Q(A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m)$ , 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  $R \times S$  will have  $n_R * n_S$  tuples.
  - The two operands do NOT have to be "type compatible"

# Relational Algebra Operations from Set Theory: CARTESIAN PRODUCT (cont.)

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



# Relational Algebra Operations from Set Theory: CARTESIAN PRODUCT (cont.)

- To keep only combinations where the DEPENDENT is related to the EMPLOYEE, we add a SELECT operation as follows
- Example (meaningful):
  - $FEMALE\_EMPS \leftarrow \sigma_{SEX='F'}(EMPLOYEE)$
  - $EMP\_NAMES \leftarrow \pi_{FNAME, LNAME, SSN}(FEMALE\_EMPS)$
  - $EMP\_DEPENDENTS \leftarrow EMP\_NAMES \times DEPENDENT$
  - $ACTUAL\_DEPS \leftarrow \sigma_{SSN=ESSN}(EMP\_DEPENDENTS)$
  - $RESULT \leftarrow \pi_{FNAME, LNAME, DEPENDENT\_NAME}(ACTUAL\_DEPS)$
- RESULT will now contain the name of female employees and their dependents

## Figure 8.5 The CARTESIAN PRODUCT (CROSS PRODUCT) operation.

**FEMALE\_EMPS**

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Alicia	J	Zelaya	999887777	1968-07-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

**EMPNames**

Fname	Lname	Ssn
Alicia	Zelaya	999887777
Jennifer	Wallace	987654321
Joyce	English	453453453

*continued on next slide*

## Figure 8.5 (continued) The CARTESIAN PRODUCT (CROSS PRODUCT) operation.

EMP\_DEPENDENTS

Fname	Lname	Ssn	Essn	Dependent_name	Sex	Bdate	...
Alicia	Zelaya	999887777	333445555	Alice	F	1986-04-05	...
Alicia	Zelaya	999887777	333445555	Theodore	M	1983-10-25	...
Alicia	Zelaya	999887777	333445555	Joy	F	1958-05-03	...
Alicia	Zelaya	999887777	987654321	Abner	M	1942-02-28	...
Alicia	Zelaya	999887777	123456789	Michael	M	1988-01-04	...
Alicia	Zelaya	999887777	123456789	Alice	F	1988-12-30	...
Alicia	Zelaya	999887777	123456789	Elizabeth	F	1967-05-05	...
Jennifer	Wallace	987654321	333445555	Alice	F	1986-04-05	...
Jennifer	Wallace	987654321	333445555	Theodore	M	1983-10-25	...
Jennifer	Wallace	987654321	333445555	Joy	F	1958-05-03	...
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	...
Jennifer	Wallace	987654321	123456789	Michael	M	1988-01-04	...
Jennifer	Wallace	987654321	123456789	Alice	F	1988-12-30	...
Jennifer	Wallace	987654321	123456789	Elizabeth	F	1967-05-05	...
Joyce	English	453453453	333445555	Alice	F	1986-04-05	...
Joyce	English	453453453	333445555	Theodore	M	1983-10-25	...
Joyce	English	453453453	333445555	Joy	F	1958-05-03	...
Joyce	English	453453453	987654321	Abner	M	1942-02-28	...
Joyce	English	453453453	123456789	Michael	M	1988-01-04	...
Joyce	English	453453453	123456789	Alice	F	1988-12-30	...
Joyce	English	453453453	123456789	Elizabeth	F	1967-05-05	...

continued on next slide

## Figure 8.5 (continued) The CARTESIAN PRODUCT (CROSS PRODUCT) operation.

### ACTUAL\_DEPENDENTS

Fname	Lname	Ssn	Essn	Dependent_name	Sex	Bdate	...
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	...


### RESULT

Fname	Lname	Dependent_name
Jennifer	Wallace	Abner

# 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  $R(A_1, A_2, \dots, A_n)$  and  $S(B_1, B_2, \dots, B_m)$  is:
$$R \bowtie_{\langle \text{join condition} \rangle} S$$
  - where  $R$  and  $S$  can be any relations that result from general *relational algebra expressions*.

# Binary Relational Operations: JOIN (cont.)

- 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  operation.
- $\text{DEPT\_MGR} \leftarrow \text{DEPARTMENT} \underset{\text{MGRSSN=SSN}}{\bowtie} \text{EMPLOYEE}$
- 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  $\text{DEPARTMENT.MGRSSN} = \text{EMPLOYEE.SSN}$

## Figure 8.6 Result of the JOIN operation

$\text{DEPT\_MGR} \leftarrow \text{DEPARTMENT} \bowtie_{\text{Mgr\_ssn}=\text{Ssn}} \text{EMPLOYEE.}$

DEPT\_MGR

Dname	Dnumber	Mgr_ssn	...	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	...

# Some properties of JOIN

- Consider the following JOIN operation:
  - $R(A_1, A_2, \dots, A_n) \bowtie_{R.A_i=S.B_j} S(B_1, B_2, \dots, B_m)$
  - Result is a relation Q with degree  $n + m$  attributes:
    - $Q(A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m)$ , 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*  $r[A_i]=s[B_j]$
  - 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:  $R \bowtie_{\theta} S$   
*theta*
- The join condition is called *theta*
- *Theta* can be any general boolean expression on the attributes of R and S; for example:
  - $R.A_i < S.B_j \text{ AND } (R.A_k = S.B_l \text{ OR } R.A_p < S.B_q)$
- Most join conditions involve one or more equality conditions “AND”ed together; for example:
  - $R.A_i = S.B_j \text{ AND } R.A_k = S.B_l \text{ AND } R.A_p = S.B_q$

# Binary Relational Operations: EQUIJOIN

- EQUIJOIN Operation
- 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.
  - The JOIN seen in the previous example was an EQUIJOIN.

# Binary Relational Operations:

## NATURAL JOIN Operation

- 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 (continued)

- Example: To apply a natural join on the DNUMBER attributes of DEPARTMENT and DEPT\_LOCATIONS, it is sufficient to write:
  - $DEPT\_LOCS \leftarrow DEPARTMENT * DEPT\_LOCATIONS$
- 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:  $Q \leftarrow R(A,B,C,D) * S(C,D,E)$ 
  - The implicit join condition includes *each pair* of attributes with the same name, “AND”ed together:
    - $R.C = S.C \text{ AND } R.D = S.D$
  - Result keeps only one attribute of each such pair:
    - $Q(A,B,C,D,E)$

# Example of NATURAL JOIN operation

**Figure 8.7** Results of two natural join operations. (a)  $\text{proj\_dept} \leftarrow \text{project} * \text{dept}$ . (b)  $\text{dept\_locs} \leftarrow \text{department} * \text{dept\_locations}$ .

(a)

**PROJ\_DEPT**

Pname	Pnumber	Plocation	Dnum	Dname	Mgr_ssn	Mgr_start_date
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

(b)

**DEPT\_LOCS**

Dname	Dnumber	Mgr_ssn	Mgr_start_date	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

# Complete Set of Relational Operations

- The set of operations including SELECT  $\sigma$ , PROJECT  $\pi$ , UNION  $\cup$ , DIFFERENCE  $-$ , RENAME  $\rho$ , and CARTESIAN PRODUCT  $\times$  is called a *complete set* because any other relational algebra expression can be expressed by a combination of these five operations.
- For example:
  - $R \cap S = (R \cup S) - ((R - S) \cup (S - R))$
  - $R \bowtie_{\langle \text{join condition} \rangle} S = \sigma_{\langle \text{join condition} \rangle} (R \times S)$

# Binary Relational Operations: DIVISION

## ■ DIVISION Operation

- The division operation is applied to two relations
- $R(Z) \div S(X)$ , where  $X$  subset  $Z$ . Let  $Y = Z - X$  (and hence  $Z = X \cup Y$ ); that is, let  $Y$  be the set of attributes of  $R$  that are not attributes of  $S$ .
- The result of DIVISION is a relation  $T(Y)$  that includes a tuple  $t$  if tuples  $t_R$  appear in  $R$  with  $t_R[Y] = t$ , and with
  - $t_R[X] = t_s$  *for every tuple  $t_s$  in  $S$ .*
- For a tuple  $t$  to appear in the result  $T$  of the DIVISION, the values in  $t$  must appear in  $R$  in combination with *every* tuple in  $S$ .

# Example of DIVISION

**Figure 8.8** The DIVISION operation. (a) Dividing SSN\_PNOS by SMITH\_PNOS. (b)  $T \leftarrow R \div S$ .

(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



# Table 8.1 Operations of Relational Algebra

**Table 8.1** Operations of Relational Algebra

OPERATION	PURPOSE	NOTATION
SELECT	Selects all tuples that satisfy the selection condition from a relation $R$ .	$\sigma_{\langle \text{selection condition} \rangle}(R)$
PROJECT	Produces a new relation with only some of the attributes of $R$ , and removes duplicate tuples.	$\pi_{\langle \text{attribute list} \rangle}(R)$
THETA JOIN	Produces all combinations of tuples from $R_1$ and $R_2$ that satisfy the join condition.	$R_1 \bowtie_{\langle \text{join condition} \rangle} R_2$
EQUIJOIN	Produces all the combinations of tuples from $R_1$ and $R_2$ that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{\langle \text{join condition} \rangle} R_2$ , OR $R_1 \bowtie_{(\langle \text{join attributes } 1 \rangle), (\langle \text{join attributes } 2 \rangle)} R_2$
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.	$R_1 \star_{\langle \text{join condition} \rangle} R_2$ , OR $R_1 \star_{(\langle \text{join attributes } 1 \rangle), (\langle \text{join attributes } 2 \rangle)} R_2$ OR $R_1 \star R_2$

*continued on next slide*

# Table 8.1 Operations of Relational Algebra (continued)

**Table 8.1** Operations of Relational Algebra

OPERATION	PURPOSE	NOTATION
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.	$R_1 \cup R_2$
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 $Z = X \cup Y$ .	$R_1(Z) \div R_2(Y)$

# Query Tree Notation

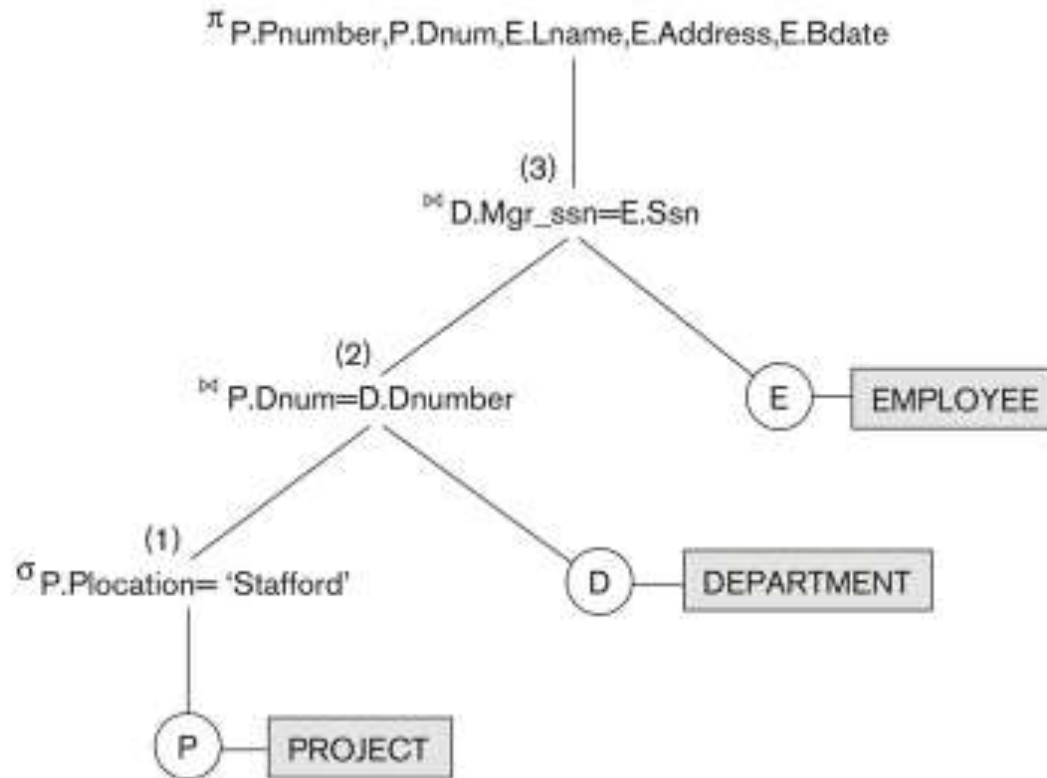
## ■ Query Tree

- An internal data structure to represent a query
- Standard technique for estimating the work involved in executing the query, the generation of intermediate results, and the optimization of execution
- Nodes stand for operations like selection, projection, join, renaming, division, ....
- Leaf nodes represent base relations
- A tree gives a good visual feel of the complexity of the query and the operations involved
- Algebraic Query Optimization consists of rewriting the query or modifying the query tree into an equivalent tree.

**(see Chapter 15)**

# Example of Query Tree

Figure 8.9 Query tree corresponding to the relational algebra expression for Q2.



# 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 of such functions include 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, AVERAGE, MAXIMUM, and MINIMUM.
- The COUNT function is used for counting tuples or values.

# Aggregate Function Operation

- Use of the Aggregate Functional operation  $\mathcal{F}$ 
  - $\mathcal{F}_{\text{MAX Salary}}(\text{EMPLOYEE})$  retrieves the maximum salary value from the EMPLOYEE relation
  - $\mathcal{F}_{\text{MIN Salary}}(\text{EMPLOYEE})$  retrieves the minimum Salary value from the EMPLOYEE relation
  - $\mathcal{F}_{\text{SUM Salary}}(\text{EMPLOYEE})$  retrieves the sum of the Salary from the EMPLOYEE relation
  - $\mathcal{F}_{\text{COUNT SSN, AVERAGE Salary}}(\text{EMPLOYEE})$  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

- The previous examples all summarized one or more attributes for a set of tuples
  - Maximum Salary or Count (number of) Ssn
- 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

# Figure 8.10 The aggregate function operation.

- $\rho_{R(Dno, No\_of\_employees, Average\_sal)}(Dno \Join COUNT Ssn, AVERAGE Salary (EMPLOYEE)).$
- $Dno \Join salary (EMPLOYEE).$
- $\Join COUNT Ssn, AVERAGE Salary (EMPLOYEE).$

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



# Figure 7.1a Results of GROUP BY and HAVING (in SQL). Q24.

Fname	Minit	Lname	<u>Ssn</u>	...	Salary	Super_ssn	Dno		Dno	Count (*)	Avg (Salary)
John	B	Smith	123456789		30000	333445555	5		5	4	33250
Franklin	T	Wong	333445555		40000	888665555	5		4	3	31000
Ramesh	K	Narayan	666884444		38000	333445555	5		1	1	55000
Joyce	A	English	453453453	...	25000	333445555	5				
Alicia	J	Zelaya	999887777		25000	987654321	4				
Jennifer	S	Wallace	987654321		43000	888665555	4				
Ahmad	V	Jabbar	987987987		25000	987654321	4				
James	E	Bong	888665555		55000	NULL	1				

Result of Q24

Grouping EMPLOYEE tuples by the value of Dno

*continued on next slide*

# Additional Relational Operations (continued)

- **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''**; and so on.

# Additional Relational Operations (continued)

- Although it is possible to retrieve employees at each level and then take their union, we cannot, in general, specify a query such as “retrieve the supervisees of ‘James Borg’ at all levels” without utilizing a looping mechanism.
  - The SQL3 standard includes syntax for recursive closure.

# Figure 8.11 A two-level recursive query.

## SUPERVISION

(Borg's Ssn is 888665555)

(Ssn) (Super\_ssn)

Ssn1	Ssn2
123456789	333445555
333445555	888665555
999887777	987654321
987654321	888665555
666884444	333445555
453453453	333445555
987987987	987654321
888665555	null

## RESULT1

Ssn
333445555
987654321

(Supervised by Borg)

## RESULT2

Ssn
123456789
999887777
666884444
453453453
987987987

(Supervised by  
Borg's subordinates)

## RESULT

Ssn
123456789
999887777
666884444
453453453
987987987
333445555
987654321

(RESULT1  $\cup$  RESULT2)

# Additional Relational Operations (continued)

- The OUTER JOIN Operation
  - In NATURAL JOIN and EQUIJOIN, tuples without a *matching* (or *related*) tuple are eliminated from the join result
    - Tuples with null in the join attributes are also eliminated
    - This amounts to loss of information.
  - A set of operations, called OUTER joins, can be used when we want to keep all the tuples in R, or all those in S, or all those in both relations in the result of the join, regardless of whether or not they have matching tuples in the other relation.

# Additional Relational Operations (continued)

- The left outer join operation keeps every tuple in the first or left relation  $R$  in  $R \bowtie\!\!\!\!\!\! \times S$ ; if no matching tuple is found in  $S$ , then the attributes of  $S$  in the join result are filled or “padded” with null values.
- A similar operation, right outer join, keeps every tuple in the second or right relation  $S$  in the result of  $R \bowtie\!\!\!\!\!\! \times S$ .
- A third operation, full outer join, denoted by  $\boxtimes$ , keeps all tuples in both the left and the right relations when no matching tuples are found, padding them with null values as needed.

## Figure 8.12 The result of a LEFT OUTER JOIN operation.

### 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

# Additional Relational Operations (continued)

## ■ 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  $T(X, Y, Z)$ .



# Additional Relational Operations (continued)

- 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)**

# Examples of Queries in Relational Algebra : Procedural Form

- **Q1: Retrieve the name and address of all employees who work for the 'Research' department.**

$\text{RESEARCH\_DEPT} \leftarrow \sigma_{\text{DNAME}='Research'} (\text{DEPARTMENT})$

$\text{RESEARCH\_EMPS} \leftarrow (\text{RESEARCH\_DEPT} \bowtie_{\text{DNUMBER}=\text{DNOEMPLOYEE}} \text{EMPLOYEE})$

$\text{RESULT} \leftarrow \pi_{\text{FNAME}, \text{LNAME}, \text{ADDRESS}} (\text{RESEARCH\_EMPS})$

- **Q6: Retrieve the names of employees who have no dependents.**

$\text{ALL\_EMPS} \leftarrow \pi_{\text{SSN}} (\text{EMPLOYEE})$

$\text{EMPS\_WITH\_DEPS}(\text{SSN}) \leftarrow \pi_{\text{ESSN}} (\text{DEPENDENT})$

$\text{EMPS\_WITHOUT\_DEPS} \leftarrow (\text{ALL\_EMPS} - \text{EMPS\_WITH\_DEPS})$

$\text{RESULT} \leftarrow \pi_{\text{LNAME}, \text{FNAME}} (\text{EMPS\_WITHOUT\_DEPS} * \text{EMPLOYEE})$

# Examples of Queries in Relational Algebra – Single expressions

As a single expression, these queries become:

- **Q1: Retrieve the name and address of all employees who work for the 'Research' department.**

$$\pi_{Fname, Lname, Address} (\sigma_{Dname = 'Research'} (DEPARTMENT \bowtie_{Dnumber = Dno} (EMPLOYEE)))$$

- **Q6: Retrieve the names of employees who have no dependents.**

$$\pi_{Lname, Fname} ((\pi_{Ssn} (EMPLOYEE) - \rho_{Ssn} (\pi_{Essn} (DEPENDENT))) * EMPLOYEE)$$

# Relational Calculus

- A **relational calculus** expression creates a new relation, which is specified in terms of variables that range over rows of the stored database relations (in **tuple calculus**) or over columns of the stored relations (in **domain calculus**).
- In a calculus expression, there is *no order of operations* to specify how to retrieve the query result—a calculus expression specifies only what information the result should contain.
  - This is the main distinguishing feature between relational algebra and relational calculus.

# Relational Calculus (continued)

- Relational calculus is considered to be a **nonprocedural** or **declarative** language.
- This differs from relational algebra, where we must write a *sequence of operations* to specify a retrieval request; hence relational algebra can be considered as a **procedural** way of stating a query.

# Tuple Relational Calculus

- The tuple relational calculus is based on specifying a number of tuple variables.
- Each tuple variable usually ranges over a particular database relation, meaning that the variable may take as its value any individual tuple from that relation.
- A simple tuple relational calculus query is of the form

$$\{t \mid \text{COND}(t)\}$$

- where  $t$  is a tuple variable and  $\text{COND}(t)$  is a conditional expression involving  $t$ .
- The result of such a query is the set of all tuples  $t$  that satisfy  $\text{COND}(t)$ .

# Tuple Relational Calculus (continued)

- Example: To find the first and last names of all employees whose salary is above \$50,000, we can write the following tuple calculus expression:

**$\{t.FNAME, t.LNAME \mid EMPLOYEE(t) \text{ AND } t.SALARY > 50000\}$**

- The condition  $EMPLOYEE(t)$  specifies that the **range relation** of tuple variable  $t$  is  $EMPLOYEE$ .
- The first and last name (PROJECTION  $\pi_{FNAME, LNAME}$ ) of each  $EMPLOYEE$  tuple  $t$  that satisfies the condition  $t.SALARY > 50000$  (SELECTION  $\sigma_{SALARY > 50000}$ ) will be retrieved.

# The Existential and Universal Quantifiers

- Two special symbols called quantifiers can appear in formulas; these are the universal quantifier ( $\forall$ ) and the existential quantifier ( $\exists$ ).
- Informally, a tuple variable  $t$  is bound if it is quantified, meaning that it appears in an  $(\forall t)$  or  $(\exists t)$  clause; otherwise, it is free.
- If  $F$  is a formula, then so are  $(\exists t)(F)$  and  $(\forall t)(F)$ , where  $t$  is a tuple variable.
  - The formula  $(\exists t)(F)$  is true if the formula  $F$  evaluates to true for some (at least one) tuple assigned to free occurrences of  $t$  in  $F$ ; otherwise  $(\exists t)(F)$  is false.
  - The formula  $(\forall t)(F)$  is true if the formula  $F$  evaluates to true for every tuple (in the universe) assigned to free occurrences of  $t$  in  $F$ ; otherwise  $(\forall t)(F)$  is false.



# The Existential and Universal Quantifiers (continued)

- $\forall$  is called the universal or “for all” quantifier because every tuple in “the universe of” tuples must make  $F$  true to make the quantified formula true.
- $\exists$  is called the existential or “there exists” quantifier because any tuple that exists in “the universe of” tuples may make  $F$  true to make the quantified formula true.

# Example Query Using Existential Quantifier

- Retrieve the name and address of all employees who work for the 'Research' department. The query can be expressed as :  
 $\{t.FNAME, t.LNAME, t.ADDRESS \mid EMPLOYEE(t) \text{ and } (\exists d) (DEPARTMENT(d) \text{ and } d.DNAME='Research' \text{ and } d.DNUMBER=t.DNO) \}$
- The only *free tuple variables* in a relational calculus expression should be those that appear to the left of the bar (  $\mid$  ).
  - In above query, t is the only free variable; it is then *bound successively* to each tuple.
- If a tuple *satisfies the conditions* specified in the query, the attributes FNAME, LNAME, and ADDRESS are retrieved for each such tuple.
  - The conditions EMPLOYEE (t) and DEPARTMENT(d) specify the range relations for t and d.
  - The condition d.DNAME = 'Research' is a selection condition and corresponds to a SELECT operation in the relational algebra, whereas the condition d.DNUMBER = t.DNO is a JOIN condition.

# Example Query Using Universal Quantifier

- Find the names of employees who work on *all* the projects controlled by department number 5. The query can be:  
**{e.LNAME, e.FNAME | EMPLOYEE(e) and ( (  $\forall$  x)(not(PROJECT(x)) or not(x.DNUM=5)**  
**OR ( (  $\exists$  w)(WORKS\_ON(w) and w.ESSN=e.SSN and x.PNUMBER=w.PNO)) ) }**
- Exclude from the universal quantification all tuples that we are not interested in by making the condition true *for all such tuples*.
  - The first tuples to exclude (by making them evaluate automatically to true) are those that are not in the relation R of interest.
- In query above, using the expression **not(PROJECT(x))** inside the universally quantified formula evaluates to true all tuples x that are not in the PROJECT relation.
  - Then we exclude the tuples we are not interested in from R itself. The expression not(x.DNUM=5) evaluates to true all tuples x that are in the project relation but are not controlled by department 5.
- Finally, we specify a condition that must hold on all the remaining tuples in R.  
**( (  $\exists$  w)(WORKS\_ON(w) and w.ESSN=e.SSN and x.PNUMBER=w.PNO)**

# Languages Based on Tuple Relational Calculus

- The language **SQL** is based on tuple calculus. It uses the basic block structure to express the queries in tuple calculus:
  - **SELECT** <list of attributes>
  - **FROM** <list of relations>
  - **WHERE** <conditions>
- **SELECT** clause mentions the attributes being projected, the **FROM** clause mentions the relations needed in the query, and the **WHERE** clause mentions the selection as well as the join conditions.
  - SQL syntax is expanded further to accommodate other operations. (See Chapter 8).

# Languages Based on Tuple Relational Calculus (continued)

- Another language which is based on tuple calculus is **QUEL** which actually uses the range variables as in tuple calculus. Its syntax includes:
  - **RANGE OF** <variable name> **IS** <relation name>
- Then it uses
  - **RETRIEVE** <list of attributes from range variables>
  - **WHERE** <conditions>
- This language was proposed in the relational DBMS **INGRES**. (system is currently still supported by Computer Associates – but the QUEL language is no longer there).

# The Domain Relational Calculus

- Another variation of relational calculus called the domain relational calculus, or simply, domain calculus is equivalent to tuple calculus and to relational algebra.
- The language called QBE (Query-By-Example) that is related to domain calculus was developed almost concurrently to SQL at IBM Research, Yorktown Heights, New York.
  - Domain calculus was thought of as a way to explain what QBE does.
- Domain calculus differs from tuple calculus in the type of variables used in formulas:
  - Rather than having variables range over tuples, the variables range over single values from domains of attributes.
- To form a relation of degree  $n$  for a query result, we must have  $n$  of these domain variables— one for each attribute.

# The Domain Relational Calculus (continued)

- An expression of the domain calculus is of the form

$\{ x_1, x_2, \dots, x_n \mid$

**COND**( $x_1, x_2, \dots, x_n, x_{n+1}, x_{n+2}, \dots, x_{n+m}$ ) $\}$

- where  $x_1, x_2, \dots, x_n, x_{n+1}, x_{n+2}, \dots, x_{n+m}$  are domain variables that range over domains (of attributes)
- and COND is a condition or formula of the domain relational calculus.

# Example Query Using Domain Calculus

Retrieve the birthdate and address of the employee whose name is 'John B. Smith'.

■ Query :

$\{uv \mid (\exists q) (\exists r) (\exists s) (\exists t) (\exists w) (\exists x) (\exists y) (\exists z)$   
 $(\text{EMPLOYEE}(qrstuvwxyz) \text{ and } q=\text{'John'} \text{ and } r=\text{'B'} \text{ and } s=\text{'Smith'})\}$

- **Abbreviated notation** **EMPLOYEE(qrstuvwxyz)** uses the variables without the separating commas: **EMPLOYEE(q,r,s,t,u,v,w,x,y,z)**
- Ten variables for the employee relation are needed, one to range over the domain of each attribute in order.
  - Of the ten variables  $q, r, s, \dots, z$ , only  $u$  and  $v$  are free.
- Specify the *requested attributes*, BDATE and ADDRESS, by the free domain variables  $u$  for BDATE and  $v$  for ADDRESS.
- Specify the condition for selecting a tuple following the bar ( | )—
  - namely, that the sequence of values assigned to the variables  $qrstuvwxyz$  be a tuple of the employee relation and that the values for  $q$  (FNAME),  $r$  (MINIT), and  $s$  (LNAME) be 'John', 'B', and 'Smith', respectively.



# QBE: A Query Language Based on Domain Calculus (Appendix C)

- This language is based on the idea of giving an example of a query using “example elements” which are nothing but domain variables.
- Notation: An example element stands for a domain variable and is specified as an example value preceded by the underscore character.
- P. (called **P dot**) operator (for “print”) is placed in those columns which are requested for the result of the query.
- A user may initially start giving actual values as examples, but later can get used to providing a minimum number of variables as example elements.

# QBE: A Query Language Based on Domain Calculus (Appendix C)

- The language is very user-friendly, because it uses minimal syntax.
- QBE was fully developed further with facilities for grouping, aggregation, updating etc. and is shown to be equivalent to SQL.
- The language is available under QMF (Query Management Facility) of DB2 of IBM and has been used in various ways by other products like ACCESS of Microsoft, and PARADOX.
- For details, **see Appendix C** in the text.

# QBE Examples

- QBE initially presents a relational schema as a “blank schema” in which the user fills in the query as an example:

# Example Schema as a QBE Query Interface

## EMPLOYEE

Fname	Minit	Lname	<u>Ssn</u>	Bdate	Address	Sex	Salary	Super_ssn	Dno
-------	-------	-------	------------	-------	---------	-----	--------	-----------	-----

## DEPARTMENT

Dname	<u>Dnumber</u>	Mgr_ssn	Mgr_start_date
-------	----------------	---------	----------------

## DEPT\_LOCATIONS

<u>Dnumber</u>	<u>Dlocation</u>
----------------	------------------

## PROJECT

Pname	<u>Pnumber</u>	Plocation	Dnum
-------	----------------	-----------	------

## WORKS\_ON

<u>Essn</u>	<u>Pno</u>	Hours
-------------	------------	-------

## DEPENDENT

<u>Essn</u>	<u>Dependent_name</u>	Sex	Bdate	Relationship
-------------	-----------------------	-----	-------	--------------

**Figure C.1**

The relational schema of Figure 5.5 as it may be displayed by QBE.

# QBE Examples

- The following domain calculus query can be successively minimized by the user as shown:
- Query :  
 $\{uv \mid (\exists q) (\exists r) (\exists s) (\exists t) (\exists w) (\exists x) (\exists y) (\exists z)$   
 $(\text{EMPLOYEE}(qrstuvwxyz) \text{ and } q=\text{'John'} \text{ and } r=\text{'B'} \text{ and } s=\text{'Smith'})\}$

# Four Successive Ways to Specify a QBE Query

**(a) EMPLOYEE**

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	_123456789	P_9/1/60	P_100 Main, Houston, TX	_M	_25000	_123456789	_3

**(b) EMPLOYEE**

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith		P_9/1/60	P_100 Main, Houston, TX				

**(c) EMPLOYEE**

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith		P_X	P_Y				

**(d) EMPLOYEE**

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith		P.	P.				

**Figure C.2**

Four ways to specify the query Q0 in QBE.

# QBE Examples

- Specifying complex conditions in QBE:
- A technique called the “condition box” is used in QBE to state more involved Boolean expressions as conditions.
- The C.4(a) gives employees who work on either project 1 or 2, whereas the query in C.4(b) gives those who work on both the projects.

# Complex Conditions with and without a condition box as a part of QBE Query

WORKS\_ON

(a)

Essn	Pno	Hours
P.		> 20

WORKS\_ON

(b)

Essn	Pno	Hours
P.	_PX	_HX

CONDITIONS

_HX > 20 and (PX = 1 or PX = 2)
---------------------------------

WORKS\_ON

(c)

Essn	Pno	Hours
P.	1	> 20
P.	2	> 20

**Figure C.3**

Specifying complex conditions in QBE. (a) The query Q0A. (b) The query Q0B with a condition box. (c) The query Q0B without a condition box.



# Handling AND conditions in a QBE Query

**Figure C.4**

Specifying EMPLOYEES who work on both projects. (a) Incorrect specification of an AND condition. (b) Correct specification.

(a)

Essn	Pno	Hours
P_ES	1	
P_ES	2	

(b)

Essn	Pno	Hours
P_EX	1	
P_EY	2	

**CONDITIONS**

_EX = _EY
-----------

# JOIN in QBE : Examples

- The join is simply accomplished by using the same example element (variable with underscore) in the columns being joined from different (or same as in C.5 (b)) relation.
- Note that the Result is set up as an independent table to show variables from multiple relations placed in the result.

# Performing Join with common example elements and use of a RESULT relation

**Figure C.5**

Illustrating JOIN and result relations in QBE. (a) The query Q1. (b) The query Q8.

**(a) EMPLOYEE**

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
_FN		_LN			_Addr				_DX

**DEPARTMENT**

Dname	Dnumber	Mgrssn	Mgr_start_date
Research	_DX		

RESULT			
P.	_FN	_LN	_Addr

**(b) EMPLOYEE**

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
_E1		_E2						_Xssn	
_S1		_S2	_Xssn						

RESULT				
P.	_E1	_E2	_S1	_S2

# AGGREGATION in QBE

- Aggregation is accomplished by using .CNT for count, .MAX, .MIN, .AVG for the corresponding aggregation functions
- Grouping is accomplished by .G operator.
- Condition Box may use conditions on groups (similar to HAVING clause in SQL – see Section 8.5.8)

# AGGREGATION in QBE : Examples

(a) EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
							P.CNT.		

(b) EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
							P.CNT.ALL		

(c) EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
			P.CNT.ALL				P.AVG.ALL		P.G.

(d) PROJECT

Pname	Pnumber	Plocation	Dnum
P.	_PX		

WORKS\_ON

Essn	Pno	Hours
P.CNT.EX	G._PX	

CONDITIONS

CNT_EX > 2
------------

**Figure C.6**

Functions and grouping in QBE. (a) The query Q23. (b) The query Q23A. (c) The query Q24. (d) The query Q26.

# NEGATION in QBE : Example

**Figure C.7**

Illustrating negation by the query Q6.

## EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
P.		P.	_SX						

## DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
_SX				

# UPDATING in QBE : Examples

**(a) EMPLOYEE**

	Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
I.	Richard	K	Marini	653298653	30-Dec-52	98 Oak Forest, Katy, TX	M	37000	987654321	4

**(b) EMPLOYEE**

	Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
D.				653298653						

**(c) EMPLOYEE**

	Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
	John		Smith					U_S*1.1		U.4

**Figure C.8**

Modifying the database in QBE. (a) Insertion. (b) Deletion. (c) Update in QBE.

# Chapter Summary

- Relational Algebra
  - Unary Relational Operations
  - Relational Algebra Operations From Set Theory
  - Binary Relational Operations
  - Additional Relational Operations
  - Examples of Queries in Relational Algebra
- Relational Calculus
  - Tuple Relational Calculus
  - Domain Relational Calculus
- Overview of the QBE language (appendix C)