

Fundamentals of
**DATABASE
SYSTEMS**

FOURTH EDITION

ELMASRI  NAVATHE

Chapter 6

The Relational Algebra and Calculus



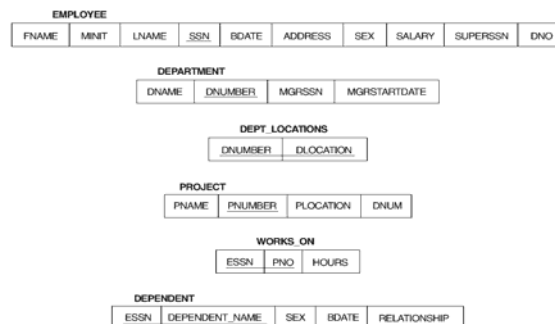
Chapter Outline

- Example Database Application (COMPANY)
- 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 D)

Database State for COMPANY

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

Figure 7.5 Schema diagram for the COMPANY relational database schema; the primary keys are underlined.



Relational Algebra

- The basic set of operations for the relational model is known as the relational algebra. These operations enable a user to specify basic retrieval requests.
- The result of a retrieval is a new relation, which may have been formed from one or more relations. The **algebra operations** thus produce new relations, which can be further manipulated using operations of the same algebra.
- A sequence of relational algebra operations forms a **relational algebra expression**, whose result will also be a relation that represents the result of a database query (or retrieval request).

Unary Relational Operations

● SELECT Operation

SELECT operation is used to select a *subset* of the tuples from a relation that satisfy a **selection condition**. It is a filter that keeps only those tuples that satisfy a qualifying condition – those satisfying the condition are selected while others are discarded.

Example: To select the EMPLOYEE tuples whose department number is four or those whose salary is greater than \$30,000 the following notation is used:

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

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

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

Unary Relational Operations

SELECT Operation Properties

- The SELECT operation $\sigma_{\langle \text{selection condition} \rangle}(R)$ produces a relation S that has the same schema as R
- The SELECT operation σ is **commutative**; i.e.,

$$\sigma_{\langle \text{condition1} \rangle}(\sigma_{\langle \text{condition2} \rangle}(R)) = \sigma_{\langle \text{condition2} \rangle}(\sigma_{\langle \text{condition1} \rangle}(R))$$
- A cascaded SELECT operation **may be applied in any order**; i.e.,

$$\sigma_{\langle \text{condition1} \rangle}(\sigma_{\langle \text{condition2} \rangle}(\sigma_{\langle \text{condition3} \rangle}(R))) = \sigma_{\langle \text{condition2} \rangle}(\sigma_{\langle \text{condition3} \rangle}(\sigma_{\langle \text{condition1} \rangle}(R)))$$
- A cascaded SELECT operation may be replaced by a single selection with a conjunction of all the conditions; i.e.,

$$\sigma_{\langle \text{condition1} \rangle}(\sigma_{\langle \text{condition2} \rangle}(\sigma_{\langle \text{condition3} \rangle}(R))) = \sigma_{\langle \text{condition1} \rangle \text{ AND } \langle \text{condition2} \rangle \text{ AND } \langle \text{condition3} \rangle}(R)$$

Unary Relational Operations (cont.)

Figure 7.8 Results of SELECT and PROJECT operations.

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

(b) $\pi_{LNAME, FNAME, SALARY}(\text{EMPLOYEE})$. (c) $\pi_{SEX, SALARY}(\text{EMPLOYEE})$

(a)

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

(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

Unary Relational Operations (cont.)

● PROJECT Operation

This operation selects certain *columns* from the table and discards the other columns. The PROJECT creates a vertical partitioning – one with the needed columns (attributes) containing results of the operation and other containing the discarded Columns.

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 $\pi_{\langle \text{attribute list} \rangle}(\text{R})$ where π (π) is the symbol used to represent the project operation and $\langle \text{attribute list} \rangle$ is the desired list of attributes from the attributes of relation R.

The project operation *removes any duplicate tuples*, so the result of the project operation is a set of tuples and hence a valid relation.

Unary Relational Operations (cont.)

PROJECT Operation Properties

- The number of tuples in the result of projection $\pi_{\langle \text{list} \rangle}(\text{R})$ 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 is equal to the number of tuples in R.
- $\pi_{\langle \text{list1} \rangle}(\pi_{\langle \text{list2} \rangle}(\text{R})) = \pi_{\langle \text{list1} \rangle}(\text{R})$ as long as $\langle \text{list2} \rangle$ contains the attributes in $\langle \text{list2} \rangle$

Unary Relational Operations (cont.)

Figure 7.8 Results of SELECT and PROJECT operations.

- (a) $\sigma_{(DNO=4 \text{ AND SALARY}>25000) \text{ OR } (DNO=5 \text{ AND SALARY}>30000)}(\text{EMPLOYEE})$.
 (b) $\pi_{\text{LNAME, FNAME, SALARY}}(\text{EMPLOYEE})$. (c) $\pi_{\text{SEX, SALARY}}(\text{EMPLOYEE})$

(a)

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

(b)

LNAME	FNAME	SALARY
Smith	John	30000
Wong	Franklin	40000
Zelaysia	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

© Addison Wesley Longman, Inc. 2006, Elmasri/Navathe, Fundamentals of Database Systems, Third Edition

Unary Relational Operations (cont.)

• Rename Operation

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.

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 (cont.)

- **Rename Operation (cont.)**

The rename operator is ρ

The general Rename operation can be expressed by any of the following forms:

- $\rho_S(B_1, B_2, \dots, B_n) (R)$ is a renamed relation S based on R with column names B_1, B_2, \dots, B_n .
- $\rho_S (R)$ is a renamed relation S based on R (which does not specify column names).
- $\rho_{(B_1, B_2, \dots, B_n)} (R)$ is a renamed relation with column names B_1, B_2, \dots, B_n which does not specify a new relation name.

Unary Relational Operations (cont.)

Figure 7.9 Results of relational algebra expressions.

(a) $\pi_{LNAME, FNAME, SALARY} (\sigma_{DNO=5} (EMPLOYEE))$. (b) The same expression 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	MINT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
	John	B	Smith	123456789	1965-01-09	731 Foyden,Houston,TX	M	30000	333445555	5
	Franklin	T	Wong	333445555	1965-12-08	638 Vines,Houston,TX	M	40000	889999999	5
	Ramesh	K	Narayan	666894444	1962-09-15	975 Fire Oak,Humble,TX	M	38000	333445555	5
	Joyce	A	English	453453453	1972-07-31	5831 Rice,Houston,TX	F	25000	333445555	5

FIRSTNAME	LASTNAME	SALARY
John	Smith	30000
Franklin	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

Relational Algebra Operations From Set Theory

● UNION Operation

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.

Example: To retrieve the social security numbers of all employees who either work in department 5 or directly supervise an employee who works in department 5, 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. The two operands must be “type compatible”.

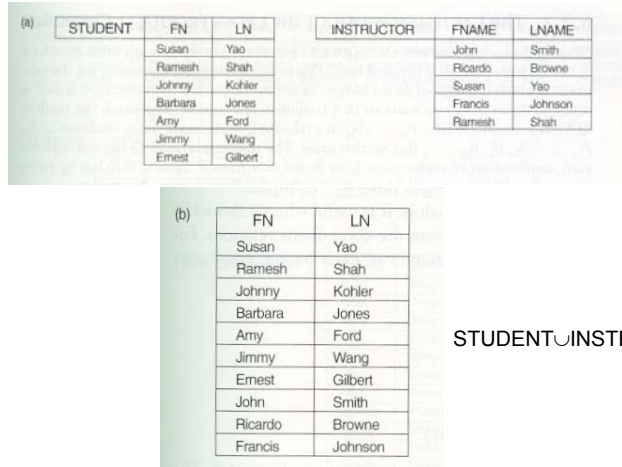
Relational Algebra Operations From Set Theory

● Type Compatibility

- The operand relations $R_1(A_1, A_2, \dots, A_n)$ and $R_2(B_1, B_2, \dots, B_n)$ must have the same number of attributes, and the domains of corresponding attributes must be compatible; that is, $\text{dom}(A_i) = \text{dom}(B_i)$ for $i=1, 2, \dots, n$.
- The resulting relation for $R_1 \cup R_2, R_1 \cap R_2$, or $R_1 - R_2$ has the same attribute names as the *first* operand relation R1 (by convention).

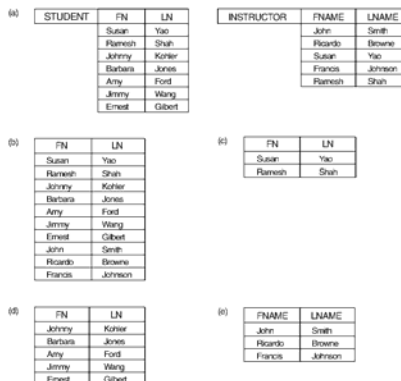
Relational Algebra Operations From Set Theory

- **UNION Example (Place Figure 6.4 a and b below)**



Relational Algebra Operations From Set Theory (cont.) – use Fig. 6.4

Figure 7.11 Illustrating the set operations union, intersection, and difference. (a) Two union compatible relations. (b) $STUDENT \cup INSTRUCTOR$. (c) $STUDENT \cap INSTRUCTOR$. (d) $STUDENT - INSTRUCTOR$. (e) $INSTRUCTOR - STUDENT$.



Relational Algebra Operations From Set Theory (cont.)

- **INTERSECTION OPERATION (Place Figure 6.4 a and c below)**

The result of this operation, denoted by $R \cap S$, is a relation that includes all tuples that are in both R and S. The two operands must be "type compatible"

Example: The result of the intersection operation (figure below) includes only

(a) STUDENT			INSTRUCTOR			(c)	
FN	LN		FNAME	LNAME	FN	LN	
Susan	Yao		John	Smith	Susan	Yao	
Ramesh	Shah		Ricardo	Browne	Ramesh	Shah	
Johnny	Kohler		Susan	Yao			
Barbara	Jones		Francis	Johnson			
Amy	Ford		Ramesh	Shah			
Jimmy	Wang						
Ernest	Gilbert						

$\text{STUDENT} \cap \text{INSTRUCTOR}$

Relational Algebra Operations From Set Theory (cont.)

- **Set Difference (or MINUS) Operation (Place Figure 6.4 a,d and e below)**

The result of this operation, denoted by $R - S$, is a relation that includes all tuples that are in R but not in S. The two operands must be "type compatible".

Example: The figure shows the names of students who are not instructors, and

(a) STUDENT			INSTRUCTOR			(d) STUDENT-INSTRUCTOR	
FN	LN		FNAME	LNAME	FN	LN	
Susan	Yao		John	Smith	Johnny	Kohler	
Ramesh	Shah		Ricardo	Browne	Barbara	Jones	
Johnny	Kohler		Susan	Yao	Amy	Ford	
Barbara	Jones		Francis	Johnson	Jimmy	Wang	
Amy	Ford		Ramesh	Shah	Ernest	Gilbert	
Jimmy	Wang						
Ernest	Gilbert						

$\text{INSTRUCTOR-STUDENT}$

(e)	
FNAME	LNAME
John	Smith
Ricardo	Browne
Francis	Johnson

Relational Algebra Operations From Set Theory (cont.)

- Notice that both union and intersection are *commutative operations*; that is

$$R \cup S = S \cup R, \text{ and } R \cap S = S \cap R$$

- Both union and intersection can be treated as n-ary operations applicable to any number of relations as both are *associative operations*; that is

$$R \cup (S \cup T) = (R \cup S) \cup T, \text{ and } (R \cap S) \cap T = R \cap (S \cap T)$$

- The minus operation is *not commutative*; that is, in general

$$R - S \neq S - R$$

Relational Algebra Operations From Set Theory (cont.)

- **CARTESIAN (or cross product) Operation**

- This operation is used to combine tuples from two relations in a combinatorial fashion. In general, the result of $R(A_1, A_2, \dots, A_n) \times S(B_1, B_2, \dots, B_m)$ 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 Q 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"

Example:

FEMALE_EMPS $\leftarrow \sigma_{SEX='F'}(\text{EMPLOYEE})$

EMPNAMES $\leftarrow \pi_{FNAME, LNAME, SSN}(\text{FEMALE_EMPS})$

EMP_DEPENDENTS $\leftarrow \text{EMPNAMES} \times \text{DEPENDENT}$

Relational Algebra Operations From Set Theory (cont.)

Figure 7.12 An illustration of the CARTESIAN PRODUCT operation.

PEOPLE	FNAME	MINIT	LNAME	SSN	BOATE	ADDRESS	SEX	SALARY	SUPERSSN	DMO
Alice	J	Zelen	98987777	98987777	3027	Clark Spring TX	F	2500	98765432	A
Janeiro	S	Redman	98765432	98765432	201	Bayou Lake TX	F	4000	98987777	A
Jayce	A	Engler	45678901	98765432	503	New Haven TX	F	2000	33445566	S

EMPVALUES	FNAME	LNAME	SSN
Alice	Zelen	98987777	
Janeiro	Redman	98765432	
Jayce	Engler	45678901	

EMP_DEPENDENTS	FNAME	LNAME	SSN	ESSN	DEPENDENT_NAME	SEX	BOATE	...
Alice	Zelen	98987777	33445566	Alice	F	989-08-25	...	
Alice	Zelen	98987777	33445566	Theodore	M	989-10-25	...	
Alice	Zelen	98987777	33445566	Eve	F	989-05-20	...	
Alice	Zelen	98987777	98765432	Alice	M	989-10-25	...	
Alice	Zelen	98987777	12345678	Madame	M	989-07-04	...	
Alice	Zelen	98987777	12345678	Alice	F	989-10-20	...	
Alice	Zelen	98987777	12345678	Eliotabeth	F	989-10-26	...	
Janeiro	Redman	98765432	33445566	Alice	F	989-08-25	...	
Janeiro	Redman	98765432	33445566	Theodore	M	989-10-25	...	
Janeiro	Redman	98765432	33445566	Eve	F	989-05-20	...	
Janeiro	Redman	98765432	98765432	Alice	M	989-10-25	...	
Janeiro	Redman	98765432	12345678	Madame	M	989-07-04	...	
Janeiro	Redman	98765432	12345678	Alice	F	989-10-20	...	
Janeiro	Redman	98765432	12345678	Eliotabeth	F	989-10-26	...	
Jayce	Engler	45678901	33445566	Alice	F	989-08-25	...	
Jayce	Engler	45678901	33445566	Theodore	M	989-10-25	...	
Jayce	Engler	45678901	33445566	Eve	F	989-05-20	...	
Jayce	Engler	45678901	98765432	Alice	M	989-10-25	...	
Jayce	Engler	45678901	12345678	Madame	M	989-07-04	...	
Jayce	Engler	45678901	12345678	Alice	F	989-10-20	...	
Jayce	Engler	45678901	12345678	Eliotabeth	F	989-10-26	...	

ACTUAL DEPENDENTS	FNAME	LNAME	SSN	ESSN	DEPENDENT_NAME	SEX	BOATE
Janeiro	Redman	98765432	98987777	Alice	M	198-10-26	

RESULT	FNAME	LNAME	DEPENDENT_NAME
Janeiro	Redman	Alice	

© Addison Wesley Longman, Inc. 2000, Elmasri/Navathe, Fundamentals of Database Systems, Third Edition

Binary Relational Operations

● JOIN Operation

- The sequence of cartesian product followed by select is used quite commonly to identify and select related tuples from two relations, a special operation, called **JOIN**. It is denoted by a \bowtie
- This operation is very important for any relational database with more than a single relation, because it allows us to process relationships among 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:

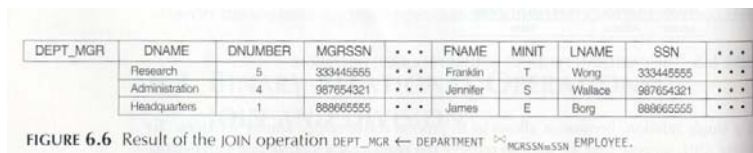
$$R \bowtie_{\langle \text{join condition} \rangle} S$$

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

Binary Relational Operations (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 \bowtie operation.

$\text{DEPT_MGR} \leftarrow \text{DEPARTMENT} \bowtie_{\text{MGRSSN=SSN}} \text{EMPLOYEE}$



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

FIGURE 6.6 Result of the JOIN operation $\text{DEPT_MGR} \leftarrow \text{DEPARTMENT} \bowtie_{\text{MGRSSN=SSN}} \text{EMPLOYEE}$.

Binary Relational Operations (cont.)

- **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 EQUIJOIN.

- **NATURAL JOIN Operation**

Because one of each pair of attributes with identical values is superfluous, a new operation called natural join—denoted by *—was created to get rid of the second (superfluous) attribute in an EQUIJOIN condition.

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 (cont.)

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

(a)

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

(b)

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

FIGURE 6.7 Results of two NATURAL JOIN operations. (a) PROJ_DEPT ← PROJECT * DEPT. (b) DEPT_LOCS ← DEPARTMENT * DEPT_LOCATIONS.

Complete Set of Relational Operations

- The set of operations including **select** σ , **project** π , **union** \cup , **set difference** $-$, and **cartesian product** \times is called a complete set \bowtie 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 (cont.)

● 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 .

Binary Relational Operations (cont.)

INSERT Figure 6.8 (a) (b)

Recap of Relational Algebra Operations

INSERT Table 6.1 here

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.

Additional Relational Operations (cont.)

Insert Figure 6.9 here

Additional Relational Operations (cont.)

Use of the Functional operator \mathcal{F}

$\mathcal{F}_{MAX\ Salary}$ (**Employee**) retrieves the maximum salary value from the Employee relation

$\mathcal{F}_{MIN\ Salary}$ (**Employee**) retrieves the minimum Salary value from the Employee relation

$\mathcal{F}_{SUM\ Salary}$ (**Employee**) retrieves the sum of the Salary from the Employee relation

DNO $\mathcal{F}_{COUNT\ SSN, AVERAGE\ Salary}$ (**Employee**) groups employees by DNO (department number) and computes the count of employees and average salary per department. [Note: count just counts the number of rows, without removing duplicates]

Additional Relational Operations (cont.)

● 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 .
- 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.

Additional Relational Operations (cont.)

Insert Figure 6.10 here

Additional Relational Operations (cont.)

● The OUTER JOIN Operation

- In NATURAL JOIN 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.
- The left outer join operation keeps every tuple in the *first* or *left* relation R in $R \bowtie_{\leftarrow} S$; 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 $R \bowtie_{\rightarrow} S$.
- A third operation, full outer join, denoted by $\bowtie_{\leftarrow\rightarrow}$ keeps all tuples in both the left and the right relations when no matching tuples are found, padding them with null values as needed.

Additional Relational Operations (cont.)

Insert figure 6.11 here

Additional Relational Operations (cont.)

● OUTER UNION Operations

- The outer union operation was developed to take the union of tuples from two relations if the relations are *not union 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 union compatible.
- The attributes that are union compatible are represented only once in the result, and those attributes that are not union 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)

Elmasri/Navathe, Fundamentals of Database Systems, Fourth Edition

Chapter 6-39

Copyright © 2004 Ramez Elmasri and Shamkant Navathe

Examples of Queries in Relational Algebra

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

$RESEARCH_DEPT \leftarrow \sigma_{DNAME='Research'}(DEPARTMENT)$

$RESEARCH_EMPS \leftarrow (RESEARCH_DEPT \bowtie_{DNUMBER=}$
 $DNOEMPLOYEE} EMPLOYEE)$

$RESULT \leftarrow \pi_{FNAME, LNAME, ADDRESS}(RESEARCH_EMPS)$

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

$ALL_EMPS \leftarrow \pi_{SSN}(EMPLOYEE)$

$EMPS_WITH_DEPS(SSN) \leftarrow \pi_{ESSN}(DEPENDENT)$

$EMPS_WITHOUT_DEPS \leftarrow (ALL_EMPS - EMPS_WITH_DEPS)$

$RESULT \leftarrow \pi_{LNAME, FNAME}(EMPS_WITHOUT_DEPS * EMPLOYEE)$

Elmasri/Navathe, Fundamentals of Database Systems, Fourth Edition

Chapter 6-40

Copyright © 2004 Ramez Elmasri and Shamkant Navathe

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 is considered to be a **nonprocedural** 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 $\text{COND}(t)$.

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:

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

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

$\sigma_{\text{SALARY} > 50000}$) 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 is ($\exists t$)(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**.
- If F is a formula, then so is ($\forall t$)(F), where t is a tuple variable. 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**. It 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.

Example Query Using Existential Quantifier

- Retrieve the name and address of all employees who work for the ‘Research’ department.

Query :

**{t.FNAME, t.LNAME, t.ADDRESS | EMPLOYEE(t) and (\exists d)
(DEPARTMENT(d) and d.DNAME=‘Research’ 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 (|). 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 $d.DNAME = \text{‘Research’}$ 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.

Query :

{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
SELECT <list of attributes>
FROM <list of relations>
WHERE <conditions>
block structure to express the queries in tuple calculus where the 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).
- 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.

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.
- An expression of the domain calculus is of the form
 $\{x_1, x_2, \dots, x_n \mid \text{COND}(x_1, x_2, \dots, x_n, x_{n+1}, x_{n+2}, \dots, x_{n+m})\}$
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 :**
 $\{uv \mid (\exists q) (\exists r) (\exists s) (\exists t) (\exists w) (\exists x) (\exists y) (\exists z)$
 $(\text{EMPLOYEE}(qrstuvwxyz) \text{ and } q='John' \text{ and } r='B' \text{ and } s='Smith')\}$
- 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 D)

- This language is based on the idea of giving an example of a query using **example elements**.
- 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.
- 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, PARADOX.
- For details, see Appendix D 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:

Insert Figure D.1

QBE Examples

- The following domain calculus query can be successively minimized by the user as shown:

Query :

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

Insert Figure D.2

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 D.4(a) gives employees who work on either project 1 or 2, whereas the query in D.4(b) gives those who work on both the projects.

Insert Figures D.3, D.4

QBE Examples

- Illustrating join in QBE. The join is simple accomplished by using the same example element in the columns being joined. Note that the Result is set us as an independent table.

Insert Figure D.5