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

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

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

Using Grouping with Aggregation

- 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
  - DNO $\mathcal{F}$ COUNT SSN, AVERAGE Salary (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.

a. $\rho_{R(\text{Dno, No\_of\_employees, Average\_sal})}(\text{Dno } \mathfrak{I} \text{ COUNT Ssn, AVERAGE Salary } (\text{EMPLOYEE}))$.

b. $\text{Dno } \mathfrak{I} \text{ COUNT Ssn, AVERAGE salary}(\text{EMPLOYEE})$.

c. $\mathfrak{I} \text{ COUNT Ssn, AVERAGE Salary}(\text{EMPLOYEE})$.

R

(a)

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

(b)

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

(c)

Count_ssn	Average_salary
8	35125

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

Fname	Minit	Lname	Ssn	...	Salary	Super_ssn	Dno
John	B	Smith	123456789		30000	333445555	5
Franklin	T	Wong	333445555		40000	888665555	5
Ramesh	K	Narayan	666884444		38000	333445555	5
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

Grouping EMPLOYEE tuples by the value of Dno

Dno	Count (*)	Avg (Salary)
5	4	33250
4	3	31000
1	1	55000

Result of Q24

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 ∪ 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 ⟕ 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 ⟖ S.
- A third operation, full outer join, denoted by ⟗ 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.**

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

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

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

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

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

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

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

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