Lecture 6

Relational Algebra

Dr. Vandana Kushwaha

Department of Computer Science
Institute of Science, BHU, Varanasi

Introduction

- The **basic set of database operations** for the R**elational model** can be defined using **Relational Algebra**.
- Historically, the **relational algebra** was developed before the **SQL language**.
- **SQL** is based on **concepts** of **relational algebra.**
- The **relational algebra** is a ***procedural*** **query language.**
- It consists of a **set of operations** that take one or **two relations** as **input** and produce a **new relation** as their **result.**
- The **fundamental operations** in the **relational algebra** are ***select, project, Cartesian product,*** **and** ***rename*****.**
- In addition to the **fundamental operations**, there are several **other operations**—namely, **set union, set intersection, set difference, natural join and division.**

Fundamental Operations

- **Unary operations**
  - The **select, project**, and **rename operations** are called ***unary*** **operations**, because they **operate** on **one relation.**
- **Binary operations**
  - The other four operations ***union*****, intersection,** ***set difference*****, division and** ***Cartesian product*** **operate** on **pairs of relations** and are, therefore, called ***binary*** **operations**.

The SELECT Operation

- The **SELECT operation** is used to choose a ***subset*** of the **tuples** from a **relation** that satisfies a **selection condition**.
- The **SELECT operation** act as a ***filter*** that keeps only those tuples that satisfy a **qualifying condition.**
- The **SELECT operation** can also be visualized as a ***horizontal partition*** of the relation into **two sets of tuples**—
  - those **tuples** that **satisfy the condition** and are **selected**, and
  - those **tuples** that **do not satisfy** the **condition** and are **discarded.**
- **SYNTAX SELECT operation:**

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

- The lowercase Greek letter **sigma (*σ*)** to denote **selection**.
- The **selection condition** appears as a **subscript** to ***σ*.**

The SELECT Operation

- The **Boolean expression** specified in **<selection condition>** is made up of a number of **clauses** of the forms:
  - <attribute name> <comparison op> <constant value>
  - <attribute name> <comparison op> <attribute name>
- **<attribute name>** is the name of an **attribute** of *R.*
- **<comparison op>** is normally one of the **operators {=, <, ≤, >, ≥, ≠},** and
- **<constant value>** is a **constant value** from the **attribute domain**.
- We can combine several conditions into a larger condition by using the connectives
  - ***and*** **(∧)**
  - ***or*** **(∨)**
  - ***not*** **(˥)**

Examples

- *Select the EMPLOYEE tuples whose department is 4*
  - **$\sigma_{Dno=4}$(EMPLOYEE)**
- *Select the EMPLOYEE tuples whose salary is greater than \$30,000*
  - **$\sigma_{Salary>30000}$(EMPLOYEE)**
- *Select the EMPLOYEE tuples whose department is 4 and whose salary is greater than \$25,000*
  - **$\sigma_{(Dno=4 \text{ AND } Salary>25000)}$ (EMPLOYEE)**
- *Select the tuples for all employees who either work in department 4 and make over \$25,000 per year, or work in department 5 and make over \$30,000*
  - **$\sigma_{(Dno=4 \text{ AND } Salary>25000) \text{ OR } (Dno=5 \text{ AND } Salary>30000)}$(EMPLOYEE)**

The PROJECT Operation

- The **PROJECT** operation **selects certain *columns*** from the **table** and **discards** the other columns.
- If we are interested in only certain attributes of a relation, we use the **PROJECT operation** to ***project*** the relation over these attributes only.
- Therefore, the result of the **PROJECT operation** can be visualized as a ***vertical partition*** of the **relation** into **two relations**:
  - one has the **needed columns (attributes)** and contains the result of the operation.
  - and the other contains the discarded columns.
- **SYNTAX PROJECT operation** is $\boxed{\pi_{\text{<attribute list>}}(R)}$
  - **π (pi)** is the symbol used to represent the **PROJECT operation**, and
  - **<attribute list>** is the **desired subset of attributes** from the attributes of **relation *R*.**

Example

- list each employee's first and last name and salary, we can use the PROJECT operation as follows:
  - $\pi_{Lname,\ Fname,\ Salary}(EMPLOYEE)$
- **Combination of Select and Project operations:**
- Retrieve the first name, last name, and salary of all employees who work in department number 5
  - $\pi_{Fname,\ Lname,\ Salary}(\sigma_{Dno=5}(EMPLOYEE))$

Relational Algebra Operations from Set Theory

- **Set theoretic operations** are used to **merge** the **elements** of **two sets** in various ways, including **UNION**, **INTERSECTION**, and **SET DIFFERENCE** (also called **MINUS** or **EXCEPT**).
- These are **binary** operations; that is, each is applied to **two sets.**
- When these operations are adapted to **relational databases**, the **two relations** on which any of these three operations are applied must have the same **type of tuples**.
- This condition has been called ***union compatibility*** or ***type compatibility***.
- Two relations ***R*(*A*1, *A*2, ..., *An*)** and ***S*(*B*1, *B*2, ..., *Bn*)** are said to be **union compatible** (or **type compatible**) if they have the **same degree *n*** and if **dom(*Ai*) = dom(*Bi*) for $1 \le i \le n$.**
- This means that the **two relations** have the **same number of attributes** and each corresponding pair of attributes has the **same domain.**

Relational Algebra Operations from Set Theory

- We can define the three operations **UNION, INTERSECTION**, and **SET DIFFERENCE** on **two union-compatible relations *R* and *S*** as follows:
  - **UNION:** The result of this operation, denoted by ***R*** ∪ ***S***, is a relation that includes **all tuples** that are **either in *R* or in *S* or in both *R* and *S*.**
  - **Duplicate tuples** are **eliminated.**
  - **INTERSECTION:** The result of this operation, denoted by ***R*** ∩ ***S***, is a relation that includes **all tuples that are in both *R* and *S*.**
  - **SET DIFFERENCE (or MINUS):** The result of this operation, denoted by ***R* – *S***, is a relation that includes **all tuples that are in *R* but not in *S*.**
- After applying the above operations the resulting relation has the same attribute names as the ***first*** relation ***R*.**

Example 1

- $\pi_{Fn,\ Ln}$ (STUDENT) $\cup$ $\pi_{Fname,\ Lname}$ (INSTRUCTOR)

(a) STUDENT

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

INSTRUCTOR

Fname	Lname
John	Smith
Ricardo	Browne
Susan	Yao
Francis	Johnson
Ramesh	Shah

STUDENT ∪ INSTRUCTOR

(b)

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

Example 2

- $\pi_{Fn,\ Ln}$ (STUDENT) $\cap$ $\pi_{Fname,\ Lname}$ (INSTRUCTOR)

(a) STUDENT

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

INSTRUCTOR

Fname	Lname
John	Smith
Ricardo	Browne
Susan	Yao
Francis	Johnson
Ramesh	Shah

STUDENT ∩ INSTRUCTOR

(c)

Fn	Ln
Susan	Yao
Ramesh	Shah

Example 3

- $\pi_{Fn,\ Ln}$ (STUDENT) **-** $\pi_{Fname,\ Lname}$ (INSTRUCTOR)

(a) STUDENT

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

INSTRUCTOR

Fname	Lname
John	Smith
Ricardo	Browne
Susan	Yao
Francis	Johnson
Ramesh	Shah

STUDENT – INSTRUCTOR

(d)

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

Example 4

- $\pi_{Fname,\ Lname}$ (INSTRUCTOR) - $\pi_{Fn,\ Ln}$ (STUDENT)

(a) STUDENT

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

INSTRUCTOR

Fname	Lname
John	Smith
Ricardo	Browne
Susan	Yao
Francis	Johnson
Ramesh	Shah

INSTRUCTOR – STUDENT

(e)

Fname	Lname
John	Smith
Ricardo	Browne
Francis	Johnson

CARTESIAN PRODUCT (CROSS PRODUCT) Operation

- **CARTESIAN PRODUCT** /**CROSS PRODUCT/CROSS JOIN** operation is denoted by **X**.
- This is also a **binary set operation**, but the relations on which it is applied **do *not* have to be union compatible.**
- In its binary form, this set operation produces a new element by **combining every member (tuple) from one relation (set) with every member (tuple) from the other relation (set).**
- In general, the result of ***R*(*A*1, *A*2, ..., *An*)** × ***S*(*B*1, *B*2, ..., *Bm*)** is a relation ***Q*** with **degree *n* + *m*** attributes ***Q*(*A*1, *A*2, ..., *An*, *B*1, *B*2, ..., *Bm*),** 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** and ***S*** has n_S **tuples**, then ***R* X *S*** will have $n_R * n_S$ tuples.

Example: CARTESIAN PRODUCT Operation

Employee		
Emp_ID	Emp_name	D-Id
1	Bill	A
2	Sara	C
3	John	A

Department	
Dept_Id	Dname
A	Marketing
B	Sales
C	Legal

Employee X Department				
Emp_ID	Emp_name	Employee.D-Id	Department.Dept_Id	Dname
1	Bill	A	A	Marketing
1	Bill	A	B	Sales
1	Bill	A	C	Legal
2	Sara	C	A	Marketing
2	Sara	C	B	Sales
2	Sara	C	C	Legal
3	John	A	A	Marketing
3	John	A	B	Sales
3	John	A	C	Legal

CROSS JOIN in SQL

Example:

SELECT *

FROM employee e, department d

Employee

EMPNAME	DEPTNO
king	10
blake	NULL
clark	10
martin	20
turner	10
jones	NULL

Department

DEPTNO	DEPTNAME
10	accounting
30	sales
20	operations

X

EMPNAME	DEPTNO	DEPTNO	DEPTNAME
king	10	10	accounting
blake	NULL	10	accounting
clark	10	10	accounting
martin	20	10	accounting
turner	10	10	accounting
jones	NULL	10	accounting
king	10	30	sales
blake	NULL	30	sales
clark	10	30	sales
martin	20	30	sales
turner	10	30	sales
jones	NULL	30	sales
king	10	20	operations
blake	NULL	20	operations
clark	10	20	operations
martin	20	20	operations
turner	10	20	operations
jones	NULL	20	operations

JOIN Operation

- The **JOIN** operation, denoted by $\bowtie$**,** is used to combine ***related tuples*** from two relations into single "longer" tuples.
- The **general form** of a **JOIN operation** on two relations ***R*(*A*1, *A*2, ..., *An*)** and ***S*(*B*1, *B*2, ..., *Bm*)** is ***R*** $\bowtie$ **<join condition> *S***
- The **result of the JOIN** is a **relation *Q*** with ***n* + *m* attributes *Q*(*A*1, *A*2, ..., *An*, *B*1, *B*2,... , *Bm*)** in that order.
- ***Q*** has one tuple for each combination of tuples—one from *R* and one from *S*—*whenever the combination satisfies the **join condition***.
- This is the main difference between **CARTESIAN PRODUCT** and **JOIN.**
- In JOIN, only combinations of tuples *satisfying the join condition* appear in the result.
- Whereas in the **CARTESIAN PRODUCT** *all* combinations of tuples are included in the result.

THETA JOIN

- A general **join condition** is of the form:
- <condition> **AND** <condition> **AND**…..**AND** <condition>
  - Where each **<condition>** is of the form ***Ai* θ *Bj***, ***Ai*** is an attribute of ***R***, ***Bj*** is an attribute of ***S***.
  - ***Ai*** and ***Bj*** have the **same domain**, and
  - **θ (theta)** is one of the **comparison operators {=, <, ≤, >, ≥, ≠}.**
- A **JOIN operation** with such a general join condition is called a **THETA JOIN**.

EQUIJOIN

- A **JOIN**, where the only **comparison operator** used is **=**, is called an **EQUIJOIN**.
- Notice that in the result of an EQUIJOIN we always have one or more pairs of attributes that have ***identical values*** in every tuple.
- **Example:**
- $\boldsymbol{\sigma}_{Employee.D\text{-}Id\,=Department.Dept\_Id}$ (Employee × *Department)*

Emp_ID	Emp_name	Employee.D-Id	Department.Dept_Id	Dname
1	Bill	A	A	Marketing
2	Sara	C	C	Legal
3	John	A	A	Marketing

- The attributes **Employee.D_Id** and **Department.Dept_Id** are identical in every tuple of the equijoin relation.

NATURAL JOIN

- Because one of each pair of attributes with identical values is **superfluous**, a new operation called **NATURAL JOIN,** denoted by * **symbol.**
- It was created to get rid of the second (superfluous) attribute in an **EQUIJOIN condition.**
- Thus **NATURAL JOIN** is basically an **EQUIJOIN followed by the removal of the superfluous attributes.**

Emp_ID	Emp_name	Employee.D-Id	Dname
1	Bill	A	Marketing
2	Sara	C	Legal
3	John	A	Marketing

- **THETA JOIN, EQUIJOIN** and **NATURAL JOIN** are also referred as **inner JOIN** operations.

NATURAL JOIN/INNER JOIN in SQL

- **Inner Join** is the simplest and most common type of join.
- It returns all rows from multiple tables where the **join condition** is **met.**
- Syntax

```
SELECT columns
FROM table1   t1
INNER JOIN table2  t2
ON t1.column = t2.column;
```

OR

```
SELECT columns
FROM table1 t1 , table2  t2
WHERE  t1.column = t2.column;
```

NATURAL /INNER JOIN in SQL

EMPNAME	DEPTNO
king	10
blake	NULL
clark	10
martin	20
turner	10
jones	NULL

DEPTNO	DEPTNAME
10	accounting
30	sales
20	operations

EMPNAME	DEPTNO	DEPTNAME
king	10	accounting
clark	10	accounting
martin	20	operations
turner	10	accounting

```
SELECT e.empname, e.deptno, d.deptname
FROM employee e, department d
WHERE e.deptno = d.deptno
```

NATURAL JOIN/INNER JOIN in SQL

```
SELECT e.empname, e.deptno, d.deptname
FROM employee e, department d
WHERE e.deptno = d.deptno
```

OR

```
SELECT e.empname, e.deptno ,d.deptname
FROM employee e
INNER JOIN department d ON e.deptno=d.deptno;
```

SELF JOIN in SQL

- A **SELF JOIN** is used for joining a table with itself.
- **Syntax**

```
SELECT columns
FROM table t1
INNER JOIN table t2
ON t1.column = t2.column;
```

OR

```
SELECT columns
FROM table t1 , table  t2
WHERE  t1.column = t2.column;
```

Example: SELF JOIN

- Consider the **Employee** schema:
- Each employee in the table has a **manager ID** associated with it.
- We use **SELF JOIN** for referencing the same table twice in a same query by using table alias.
- The **Join Condition** matches the employees with their managers using the manager_id and employee_id columns.

SELECT e.first_name || '''s manager is '

||m.last_name || '.'

FROM employees e, employees m

WHERE e.manager_id = m.employee_id ;

	Column_Name
1	EMPLOYEE_ID
2	FIRST_NAME
3	LAST_NAME
4	EMAIL
5	PHONE_NUMBER
6	HIRE_DATE
7	JOB_ID
8	SALARY
9	COMMISSION_PCT
10	MANAGER_ID
11	DEPARTMENT_ID

Example: SELF JOIN

- The previous query results in:

	E.FIRST_NAME||'''SMANAGERIS'||M.LAST_NAME||'.'
1	William's manager is Cambrault.
2	Lisa's manager is Cambrault.
3	Sundita's manager is Cambrault.
4	Tayler's manager is Cambrault.
5	Harrison's manager is Cambrault.
6	Elizabeth's manager is Cambrault.
7	Alexander's manager is De Haan.
8	Clara's manager is Errazuriz.
9	Mattea's manager is Errazuriz.
10	David's manager is Errazuriz.

OUTER JOIN

- The **outer-join** operation is an extension of the **join operation** to deal with **missing information.**
- Suppose that we have the relations with the following schemas, which contain data on **full-time employees:**
  - *employee* (*employee-name, street, city*)
  - *ft-works* (*employee-name, branch-name, salary*)
- Consider the ***employee*** and ***ft-works*** relations:

employee-name	*street*	*city*
Coyote	Toon	Hollywood
Rabbit	Tunnel	Carrotville
Smith	Revolver	Death Valley
Williams	Seaview	Seattle

employee-name	*branch-name*	*salary*
Coyote	Mesa	1500
Rabbit	Mesa	1300
Gates	Redmond	5300
Williams	Redmond	1500

- Suppose that we want to generate **a single relation** with all the information (street, city, branch name, and salary) about full-time employees.

OUTER JOIN

- A possible approach would be to use the **natural join** operation as follows:
  - **employee * ft-works**
- The result of this expression appears as:

employee-name	*street*	*city*	*branch-name*	*salary*
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
Williams	Seaview	Seattle	Redmond	1500

- Notice that we have **lost** the **street** and **city** information about **Smith**, since the tuple describing Smith is absent from the ***ft-works*** **relation**.
- Similarly, we have **lost** the **branch name** and **salary** information about **Gates,** since the tuple describing Gates is absent from the ***employee*** **relation.**
- We can use the **outer-join operation** to **avoid** this **loss of information.**

OUTER JOIN

- There are actually **three forms** of the outer join operation:
  - **Left outer join**, denoted ⟕
  - **Right outer join**, denoted ⟖
  - **Full outer join**, denoted ⟗

LEFT OUTER JOIN

- The **left outer join** takes **all tuples in the left relation** that did not match with any tuple in the **right relation**, **pads** the tuples with **null values** for all other attributes from the **right relation**, and adds them to the result of the **natural join.**
- **Example:** employee ⟕ ft-works

employee-name	*street*	*city*
Coyote	Toon	Hollywood
Rabbit	Tunnel	Carrotville
Smith	Revolver	Death Valley
Williams	Seaview	Seattle

employee-name	*branch-name*	*salary*
Coyote	Mesa	1500
Rabbit	Mesa	1300
Gates	Redmond	5300
Williams	Redmond	1500

employee-name	*street*	*city*	*branch-name*	*salary*
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
Williams	Seaview	Seattle	Redmond	1500
Smith	Revolver	Death Valley	*null*	*null*

Left Outer Join in SQL

- **Left Outer Join** returns **all rows from the left (first) table** specified in the ON condition and only those rows from the right **right (second) table** where the join condition is met.

```
SELECT columns
FROM table1  t1
LEFT JOIN table2  t2
ON t1.column = t2.column;
```

OR

```
SELECT columns
FROM table1 t1 , table2  t2
WHERE  t1.column = t2.column(+);
```

Left Outer Join in SQL

EMPNAME	DEPTNO
king	10
blake	NULL
clark	10
martin	20
turner	10
jones	NULL

DEPTNO	DEPTNAME
10	accounting
30	sales
20	operations

EMPNAME	DEPTNO	DEPTNO	DEPTNAME
turner	10	10	accounting
clark	10	10	accounting
king	10	10	accounting
martin	20	20	operations
jones	NULL	NULL	NULL
blake	NULL	NULL	NULL

```
SELECT e.empname, e.deptno, d.deptname
FROM employee e
LEFT JOIN department d
ON e.deptno=d.deptno
```

OR

```
SELECT e.empname, e.deptno, d.deptname
FROM employee e, department d
WHERE e.deptno = d.deptno(+)
```

RIGHT OUTER JOIN

- The **right outer join is symmetric** with the **left outer join.**
- It **pads tuples** from the **right relation** that did not match any from the **left relation** with **nulls** and adds them to the result of the **natural join.**
- **Example:** employee ⟖ ft-works

employee-name	*street*	*city*
Coyote	Toon	Hollywood
Rabbit	Tunnel	Carrotville
Smith	Revolver	Death Valley
Williams	Seaview	Seattle

employee-name	*branch-name*	*salary*
Coyote	Mesa	1500
Rabbit	Mesa	1300
Gates	Redmond	5300
Williams	Redmond	1500

employee-name	*street*	*city*	*branch-name*	*salary*
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
Williams	Seaview	Seattle	Redmond	1500
Gates	*null*	*null*	Redmond	5300

Right Outer Join in SQL

- **Right Outer Join** returns **all rows from the right (second) table** specified in the ON condition and only those rows from the **left (first) table** where the join condition is met.

SELECT columns

FROM table1 t1

RIGHT JOIN table2 t2

ON t1.**column** = t2.**column**;

OR

SELECT columns

FROM table1 **t1** , table2 **t2**

WHERE t1.column(+) = **t2.column**;

Right Outer Join in SQL

EMPNAME	DEPTNO
king	10
blake	NULL
clark	10
martin	20
turner	10
jones	NULL

DEPTNO	DEPTNAME
10	accounting
30	sales
20	operations

EMPNAME	DEPTNO	DEPTNO	DEPTNAME
king	10	10	accounting
clark	10	10	accounting
martin	20	20	operations
turner	10	10	accounting
NULL	NULL	30	sales

```
SELECT *
FROM employee e
RIGHT JOIN department d
ON e.deptno=d.deptno
```

OR

```
SELECT *
FROM employee e, department d
WHERE e.deptno(+) = d.deptno
```

FULL OUTER JOIN

- The **full outer join** does **both** of those operations, **padding tuples** from the **left relation** that did not match any from the right relation, as well as **tuple**s from the **right relation** that did not match any from the left relation, and adding them to the result of the join.
- **Example:** employee ⟗ ft-works

employee-name	*street*	*city*
Coyote	Toon	Hollywood
Rabbit	Tunnel	Carrotville
Smith	Revolver	Death Valley
Williams	Seaview	Seattle

employee-name	*branch-name*	*salary*
Coyote	Mesa	1500
Rabbit	Mesa	1300
Gates	Redmond	5300
Williams	Redmond	1500

employee-name	*street*	*city*	*branch-name*	*salary*
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
Williams	Seaview	Seattle	Redmond	1500
Smith	Revolver	Death Valley	*null*	*null*
Gates	*null*	*null*	Redmond	5300

FULL Outer Join in SQL

- The **Full Outer Join** returns all rows from the left hand table and right hand table. It places **NULL** where the join condition is not met.
- **Syntax:**

SELECT columns

FROM table1 t1

FULL JOIN table2 t2

ON t1.**column** = t2.**column**;

FULL Outer Join in SQL

EMPNAME	DEPTNO
king	10
blake	NULL
clark	10
martin	20
turner	10
jones	NULL

DEPTNO	DEPTNAME
10	accounting
30	sales
20	operations

EMPNAME	DEPTNO	DEPTNO	DEPTNAME
turner	10	10	accounting
clark	10	10	accounting
king	10	10	accounting
martin	20	20	operations
jones	NULL	NULL	NULL
blake	NULL	NULL	NULL
NULL	NULL	30	sales

```
SELECT *
FROM employee e
FULL JOIN department d
ON e.deptno=d.deptno
```

RENAME Operation

- **RENAME** operation can **rename** either the **relation name** or the **attribute names**, or both—as a **unary operator.**
- The **symbol ρ (rho)** is used to denote the **RENAME operator**.
- The general **RENAME operation** when applied to a relation ***R*** of **degree *n*.**
- *Suppose* ***S*** is the **new relation name**, and ***B*1, *B*2, ..., *Bn*** are the **new attribute names.**
- **Rename operation** can be denoted by any of the following **three forms**:
  - **ρ*S*(*R*)** : renames the **relation only.**
  - **ρ(*B*1, *B*2, ..., *Bn*)(*R*)** : renames the **attributes only.**
  - **ρ*S*(*B*1, *B*2, ..., *Bn*)(*R*)** : renames **both** the relation and its attributes.
- If the attributes of ***R*** are **(*A*1, *A*2, ..., *An*)** in that order, then each ***Ai* is renamed as *Bi*.**

RENAME Operation

1. **ρ**EMPLOYEE_DETAILS(EMPLOYEE)): **rename** the **EMPLOYEE** relation as **EMPLOYEE_DETAILS.**
2. **ρ**(P, B, C) (Student): rename the **first attribute** of the table **Student** with attributes **A, B, C** to **P.**
3. **ρ**(FirstName, Salary) (**Л**$_{FName,Sal}$(EMPLOYEE))

DIVISION Operation

- The **DIVISION operation**, denoted by **÷,** is useful for a **special kind** of **query** that sometimes occurs in database applications.
- In general the **Division operation** is applied to **two relations R(z) and S(x)** as:
  - **T(y)=R(z) ÷ S(x)** where **x⊆ z and y = z-x**
- **Example:** *Retrieve the names of employees who work on **all** the projects that 'John Smith' works on*.
- First, retrieve the list of **project numbers** that 'John Smith' works on in the intermediate relation **SMITH_PNOS.**

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

DIVISION Operation

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

$SSNS(Ssn) \leftarrow SSN\_PNOS \div SMITH\_PNOS$

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