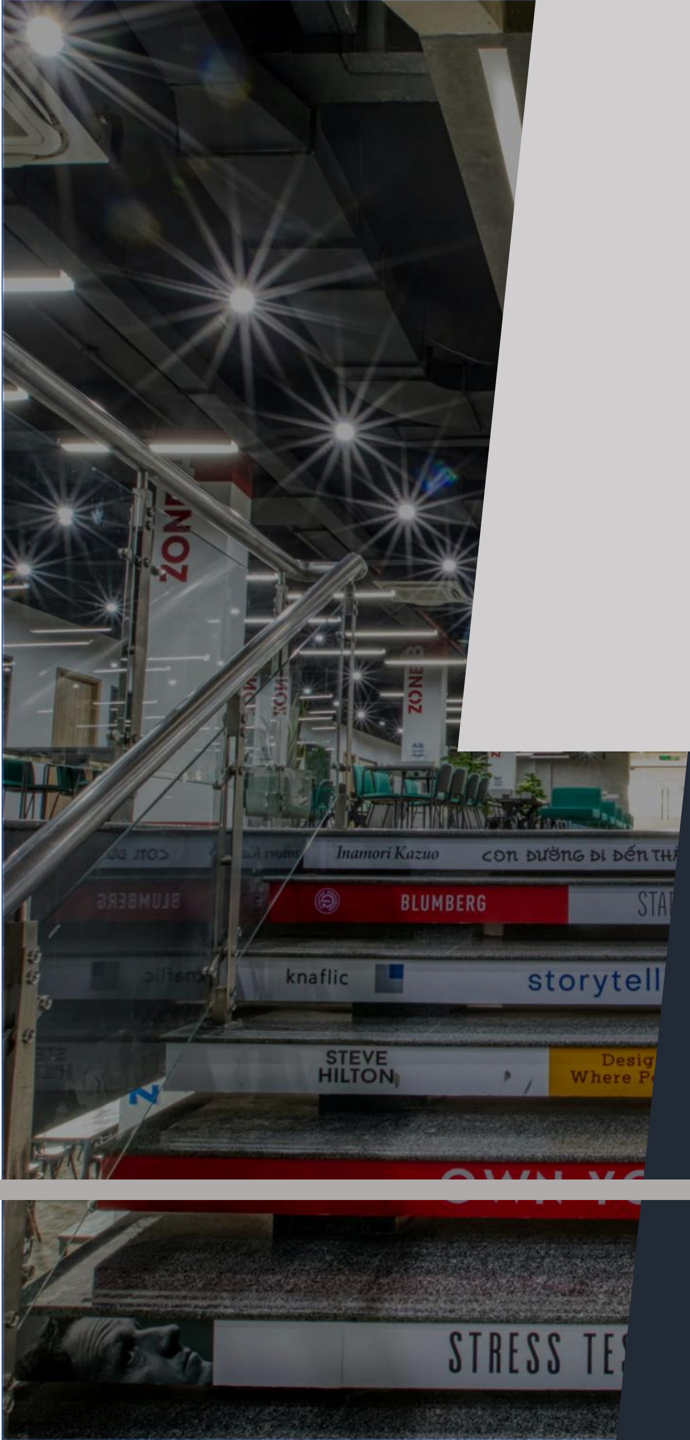


TRƯỜNG ĐẠI HỌC
QUỐC TẾ
MIỀN ĐÔNG
EASTERN
INTERNATIONAL
UNIVERSITY

CSE301 – DATABASE

Relational Algebra

Rename Operation (ρ)

- Symbol: rho (ρ)
  - Notation 1: $\rho_x (E)$

    where the symbol 'ρ' is used to denote the RENAME operator and 'E' is the result of expression or sequence of operation which is saved with the name x.

- SQL:
  - Use the AS keyword in the FROM clause
  - Use the AS keyword in the SELECT clause to rename attributes (or columns)

Rename Operation (ρ)

- SQL: Use the AS keyword in the FROM clause

  E.g. Students AS Students1 **renames** Students to Students1

SELECT column_name FROM table_name AS new_table_name WHERE condition;

- SQL: Use the AS keyword in the SELECT clause to rename attributes (or columns)

  E.g. Roll_No AS S_No renames Roll_No to S_No

SELECT column_name AS new _column_name FROM table_name WHERE condition;

Example: Rename Operation (ρ)

- Suppose we want to do cartesian producct between same table then **one of the table should be renamed with another name**
- R x R

  (Ambiguity will be there)
- $R \times \rho_S(R)$

  (Rename R to S)

R

A	B
α	1
β	2

RxR

R.A	R.B	R.A	R.B
α	1	α	1
α	1	β	2
β	2	α	1
β	2	β	2

$R \times \rho_s(R)$

R.A	R.B	S.A	S.B
α	1	α	1
α	1	β	2
β	2	α	1
β	2	β	2

Rename Operation (ρ)

- <u>Notation 2</u>: $\rho_{X\,(A1,\,A2,\,\ldots,\,An)}(E)$

  It returns the result of expression E under the name X, and with the attributes renamed to $A1, A2, \ldots, An$.

- <u>Notation 3</u>: $\rho_{(A1,\,A2,\,\ldots,\,An)}(E)$

  It returns the result of expression E with the attributes renamed to $A1, A2, \ldots, An$.

Example: Rename Operation (ρ)

Example-1: Query to find the female students from Student relation and rename the relation Student as FemaleStudent and the attributes of Student – RollNo, SName as Sno, Name.

Student

RollNo	SName	Gender
1	Neha	F
2	Suman	F
3	Sohan	M
4	Mohan	M
5	Rohan	M

Rename →

FemaleStudent

SNo	Name
1	Neha
2	Suman

$$\rho_{FemaleStudent(Sno,\ Name)}(\pi_{RollNo,\ SName}(\sigma_{Gender='F'}(Student)))$$

Examples: Rename Operation (ρ)

- Query to rename the attributes Name, Age of the table Person to N, A

$$\rho_{(N,\ A)}\ (\text{Person})$$

- Query to rename the table name Project to Work and its attributes to P, Q, R.

$$\rho_{\text{Work}(P,\ Q,\ R)}\ (\text{Project})$$

- Query to rename the first attribute of the table Student with attributes A, B, C to P.

$$\rho_{(P,\ B,\ C)}\ (\text{Student})$$

- Query to rename the table name Loan to L.

$$\rho_{L}\ (\text{Loan})$$

Banking Example: Relational Algebra

- Understand the Relation and their Relationship:

branch (branch-name, branch-city, assets)

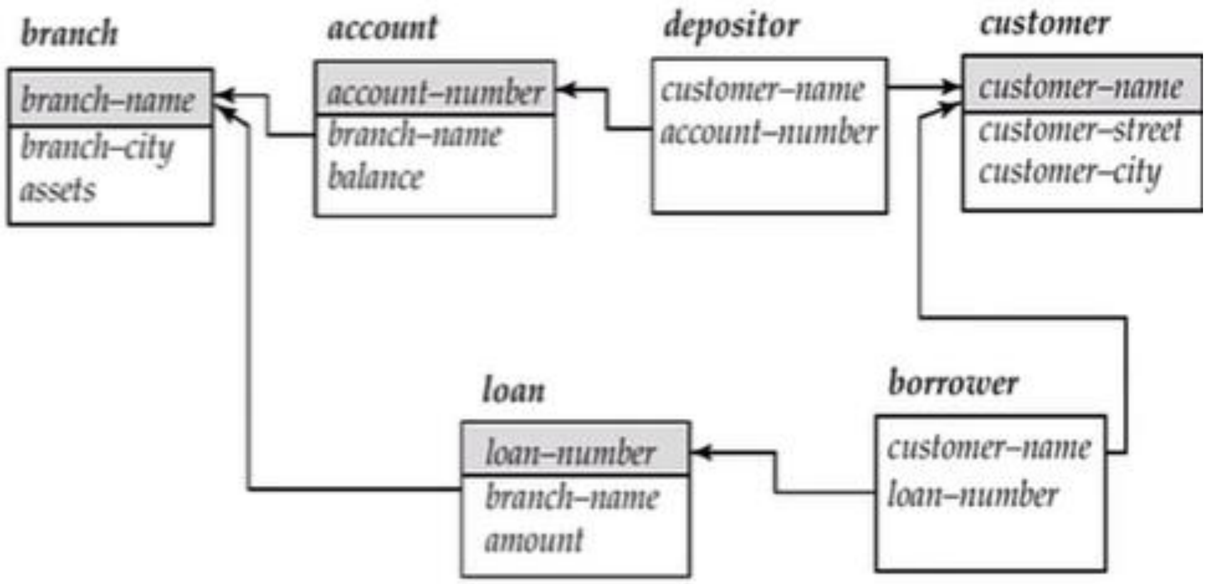
customer (customer-name, customer-street, customer-city)

account (account-number, branch-name, balance)

loan (loan-number, branch-name, amount)

depositor (customer-name, account-number)

borrower (customer-name, loan-number)



Banking Example: Relational Algebra

- Corresponding Data into tables

branch

branch-name	branch-city	assets
Brighton	Brooklyn	7100000
Downtown	Brooklyn	9000000
Mianus	Horseneck	400000
North Town	Rye	3700000
Perryridge	Horseneck	1700000
Pownal	Bennington	300000
Redwood	Palo Alto	2100000
Round Hill	Horseneck	8000000

customer

customer-name	customer-street	customer-city
Adams	Spring	Pittsfield
Brooks	Senator	Brooklyn
Curry	North	Rye
Glenn	Sand Hill	Woodside
Green	Walnut	Stamford
Hayes	Main	Harrison
Johnson	Alma	Palo Alto
Jones	Main	Harrison
Lindsay	Park	Pittsfield
Smith	North	Rye
Turner	Putnam	Stamford
Williams	Nassau	Princeton

depositor

customer-name	account-number
Hayes	A-102
Johnson	A-101
Johnson	A-201
Jones	A-217
Lindsay	A-222
Smith	A-215
Turner	A-305

account

account-number	branch-name	balance
A-101	Downtown	500
A-102	Perryridge	400
A-201	Brighton	900
A-215	Mianus	700
A-217	Brighton	750
A-222	Redwood	700
A-305	Round Hill	350

loan

loan-number	branch-name	amount
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

borrower

customer-name	loan-number
Adams	L-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Jones	L-17
Smith	L-11
Smith	L-23
Williams	L-17

Banking Example 1:

- Find all **loans** made at "Perryridge" **branch**

  $\sigma_{\textit{branch-name}=\text{"Perryridge"}}(\textit{loan})$

- Find all **loans** of over $1200

  $\sigma_{\textit{amount} > 1200}(\textit{loan})$

loan

loan-number	*branch-name*	*amount*
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

branch (*branch-name*, *branch-city*, *assets*)
customer(*customer-name*, *customer-street*, *customer-city*)
account(*account-number*, *branch-name*, *balance*)
loan(*loan-number*, *branch-name*, *amount*)
depositor(*customer-name*, *account-number*)
borrower(*customer-name*, *loan-number*)

Banking Example 2

Find all tuples who have taken **loans** of more than $1200 made by the "Perryridge" **branch**

$$\sigma_{\textit{branch-name}=\text{"Perryridge"} \wedge \textit{amount} > 1200}(\textit{loan})$$

loan-number	*branch-name*	*amount*
L-15	Perryridge	1500
L-16	Perryridge	1300

loan

loan-number	*branch-name*	*amount*
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

Banking Example 3

Find all **loan** numbers and the **amount** of the **loans**

$$\Pi_{loan\text{-}number,\ amount}\ (loan)$$

loan-number	*amount*
L-11	900
L-14	1500
L-15	1500
L-16	1300
L-17	1000
L-23	2000
L-93	500

loan

loan-number	*branch-name*	*amount*
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

branch (<u>branch-name</u>, branch-city, assets)

customer(<u>customer-name</u>, customer-street, customer-city)

account(<u>account-number</u>, branch-name, balance)

loan(<u>loan-number</u>, branch-name, amount)

depositor(<u>customer-name</u>, <u>account-number</u>)

borrower(<u>customer-name</u>, <u>loan-number</u>)

Banking Example 3

Find the **loan** number for each loan of an amount greater than \$1200

loan

loan-number	branch-name	amount
L-11	Round Hill	900
L-14	Downtown	1500
L-15	Perryridge	1500
L-16	Perryridge	1300
L-17	Downtown	1000
L-23	Redwood	2000
L-93	Mianus	500

$$\Pi_{loan\text{-}number}\ (\sigma_{amount\ >\ 1200}\ (loan))$$

loan-number
L-14
L-15
L-16
L-23

branch (branch-name, branch-city, assets)

customer(customer-name, customer-street, customer-city)

account(account-number, branch-name, balance)

loan(loan-number, branch-name, amount)

depositor(customer-name, account-number)

borrower(customer-name, loan-number)

Banking Example 3

- Find those **customers** who lives in "Harrison"

$$\Pi_{customer\text{-}name}\ (\sigma_{customer\text{-}city=\text{“Harrison”}}\ (customer))$$

customer-name
Hayes
Jones

customer

customer-name	*customer-street*	*customer-city*
Adams	Spring	Pittsfield
Brooks	Senator	Brooklyn
Curry	North	Rye
Glenn	Sand Hill	Woodside
Green	Walnut	Stamford
Hayes	Main	Harrison
Johnson	Alma	Palo Alto
Jones	Main	Harrison
Lindsay	Park	Pittsfield
Smith	North	Rye
Turner	Putnam	Stamford
Williams	Nassau	Princeton

branch (<u>*branch-name*</u>, *branch-city*, *assets*)

customer(<u>*customer-name*</u>, *customer-street*, *customer-city*)

account(<u>*account-number*</u>, *branch-name*, *balance*)

loan(<u>*loan-number*</u>, *branch-name*, *amount*)

depositor(<u>*customer-name*</u>, <u>*account-number*</u>)

borrower(<u>*customer-name*</u>, <u>*loan-number*</u>)

Banking Example 4

Find the names of all customers who have a loan, an account, or both, from the bank

$$\Pi_{customer\text{-}name}\ (borrower) \cup \Pi_{customer\text{-}name}\ (depositor)$$

customer-name
Adams
Curry
Hayes
Jackson
Jones
Smith
Williams
Lindsay
Johnson
Turner

branch (*branch-name*, *branch-city*, *assets*)
customer(*customer-name*, *customer-street*, *customer-city*)
account(*account-number*, *branch-name*, *balance*)
loan(*loan-number*, *branch-name*, *amount*)
depositor(*customer-name*, *account-number*)
borrower(*customer-name*, *loan-number*)

Banking Example 5

Find the names of all customers who have a loan and an account at bank.

$$\Pi_{customer\text{-}name}\ (borrower) \cap \Pi_{customer\text{-}name}\ (depositor)$$

customer-name
Hayes
Jones
Smith

branch (*branch-name*, *branch-city*, *assets*)
customer(*customer-name*, *customer-street*, *customer-city*)
account(*account-number*, *branch-name*, *balance*)
loan(*loan-number*, *branch-name*, *amount*)
depositor(*customer-name*, *account-number*)
borrower(*customer-name*, *loan-number*)

Banking Example 6

Find the names of all customers who have an account but no loan from the bank.

$$\Pi_{\textit{customer-name}}\ (\textit{depositor}) - \Pi_{\textit{customer-name}}\ (\textit{borrower})$$

customer-name
Johnson
Lindsay
Turner

branch (<u>branch-name</u>, branch-city, assets)

customer (<u>customer-name</u>, customer-street, customer-city)

account (<u>account-number</u>, branch-name, balance)

loan (<u>loan-number</u>, branch-name, amount)

depositor (<u>customer-name</u>, <u>account-number</u>)

borrower (<u>customer-name</u>, <u>loan-number</u>)

Banking Example 7

Find the **names** of all customers who have a **loan** at the Perryridge **branch.**

borrower × *loan*

customer-name	borrower. loan-number	loan. loan-number	branch-name	amount
Adams	L-16	L-11	Round Hill	900
Adams	L-16	L-14	Downtown	1500
Adams	L-16	L-15	Perryridge	1500
Adams	L-16	L-16	Perryridge	1300
Adams	L-16	L-17	Downtown	1000
Adams	L-16	L-23	Redwood	2000
Adams	L-16	L-93	Mianus	500
Curry	L-93	L-11	Round Hill	900
Curry	L-93	L-14	Downtown	1500
Curry	L-93	L-15	Perryridge	1500
Curry	L-93	L-16	Perryridge	1300
Curry	L-93	L-17	Downtown	1000
Curry	L-93	L-23	Redwood	2000
Curry	L-93	L-93	Mianus	500
Hayes	L-15	L-11		900
Hayes	L-15	L-14		1500
Hayes	L-15	L-15		1500
Hayes	L-15	L-16		1300
Hayes	L-15	L-17		1000
Hayes	L-15	L-23		2000
Hayes	L-15	L-93		500
...	...	...	...	...
...	...	...	...	...
...	...	...	...	...
Smith	L-23	L-11	Round Hill	900
Smith	L-23	L-14	Downtown	1500
Smith	L-23	L-15	Perryridge	1500
Smith	L-23	L-16	Perryridge	1300
Smith	L-23	L-17	Downtown	1000
Smith	L-23	L-23	Redwood	2000
Smith	L-23	L-93	Mianus	500
Williams	L-17	L-11	Round Hill	900
Williams	L-17	L-14	Downtown	1500
Williams	L-17	L-15	Perryridge	1500
Williams	L-17	L-16	Perryridge	1300
Williams	L-17	L-17	Downtown	1000
Williams	L-17	L-23	Redwood	2000
Williams	L-17	L-93	Mianus	500

$$\Pi_{customer\text{-}name}\,(\sigma_{branch\text{-}name=\text{"Perryridge"}}\,(\sigma_{borrower.\,loan\text{-}number\;=\;loan.\,loan\text{-}number}\,(borrower \times loan)))$$

customer-name
Adams
Hayes

branch (<u>branch-name</u>, branch-city, assets)

customer(<u>customer-name</u>, customer-street, customer-city)

account(<u>account-number</u>, branch-name, balance)

loan(<u>loan-number</u>, branch-name, amount)

depositor(<u>customer-name</u>, <u>account-number</u>)

borrower(<u>customer-name</u>, <u>loan-number</u>)

Banking Example 7

Find the **names** of all customers who have a **loan** at the Perryridge **branch.**

$$\Pi_{customer\text{-}name}\ (\sigma_{branch\text{-}name=\text{“Perryridge”}}\ (\sigma_{borrower.\,loan\text{-}number\ =\ loan.\,loan\text{-}number}\ (borrower\ x\ loan)))$$

Banking Example 8

Find the **names** of all customers who have a **loan** at the Perryridge **branch** but do **not** have an **account** at any **branch** of the bank.

$$\Pi_{customer\text{-}name}\,(\sigma_{branch\text{-}name\,=\,\text{"Perryridge"}}\,(\sigma_{borrower.loan\text{-}number\,=\,loan.loan\text{-}number}\,(borrower \times loan))) - \Pi_{customer\text{-}name}\,(depositor)$$

customer-name
Adams

branch (branch-name, branch-city, assets)
customer(customer-name, customer-street, customer-city)
account(account-number, branch-name, balance)
loan (loan-number, branch-name, amount)
depositor(customer-name, account-number)
borrower(customer-name, loan-number)

Banking Example 9

Find the largest **account** balance in the bank

- *Strategy*:

  1. Find those balances that are not largest (as a **temporary relation**)
     - Rename ***account*** relation as ***d*** so that we can compare each account balance with all others

       $$\Pi_{account.balance}(\sigma_{account.balance < d.balance}(account \times \rho_d(account)))$$

  2. Use set difference to find those account balances that were not found in the earlier step.
     - Take **set difference** between relation $\Pi_{balance}(account)$ and **temporary relation** just computed, to obtain the result

$$\Pi_{balance}(account) -$$

$$\Pi_{account.balance}(\sigma_{account.balance < d.balance}(account \times \rho_d\, account)))$$

Banking Example 9

account

account-number	branch-name	balance
A-101	Downtown	500
A-102	Perryridge	400
A-201	Brighton	900
A-215	Mianus	700
A-217	Brighton	750
A-222	Redwood	700
A-305	Round Hill	350

$$\Pi_{balance}(account) - \Pi_{account.\ balance}(\sigma_{account.\ balance\ <\ d.\ balance}(account \times \rho_d\ account)))$$

balance
500
400
900
700
750
700
350

\-

balance
500
400
700
750
350

→

result

balance
900

Join Operator (⋈)

- Join is an Additional / Derived operator which simplify the queries, but does not add any new power to the basic relational algebra.
- Join is a combination of a Cartesian product followed by a selection process.
- Join = cartesian product + Selection
- A Join operation pairs two tuples from different relations, if and only if a given join condition is satisfied.
- Symbol: ⋈
- Notation: $A \bowtie B = \sigma_c (A \times B)$

Difference between Join & Cartesian Product

JOIN (⋈)	Cartesian Product/Cross Product /Cross Join (X)
Combination of tuples that satisfy the filtering or matching conditions	All possible combination of tuples from the relations.
Fewer tuples then cross product, might be able to compute efficiently.	Huge number of tuples and costly to manage.

R

A	B
1	a
2	b

S

B	C
a	3
b	4

R ⋈ S

(Natural Join)

A	B	C
1	a	3
2	b	4

R X S

A	R.B	S.B	C
1	a	a	3
1	a	b	4
2	b	a	3
2	b	b	4

Types of JOIN

- Inner Joins (Join): Only those tuples that satisfy the matching criteria are included, while the rest are excluded.
- Types of Inner Join:
  - ✓ Theta Join
  - ✓ EQUI Join
  - ✓ Natural Join

- Outer Joins (Extension of Join): Along with tuples that satisfy the matching criteria, we also include some or all tuples that do not match the criteria.
- Types of Outer Join:
  - ✓ Left Outer Join
  - ✓ Right Outer Join
  - ✓ Full Outer Join

Theta Join (θ)

- Theta Join combines two or more relations based on some condition.
- The general case of JOIN operation is called a Theta join.
- It is denoted by symbol **θ**
- Notation : $R \bowtie_{\theta} S$
  - Where R is the first relation, S is the second relation and θ is the predicate or condition.
  - **θ** use all kinds of comparison operators such as >, <, >=, <=, =, ≠ etc.
  - $R \bowtie_{\theta} S = \sigma_{\theta} (R \times S)$

Example of Theta Join (θ)

R

Sid	Name	Rating	Age
22	A	7	20
31	B	8	17
58	C	10	16

S

Sid	Bid	Date
22	101	10/10/2015
58	102	02/03/2019
3	103	25/09/2007

$$R \bowtie_{R.sid < S.sid} S$$

R.Sid	Name	Rating	Age	S.Sid	Bid	Date
22	A	7	20	58	102	02/03/2019
31	B	8	17	58	102	02/03/2019

Equivalent Cartesian Product: $\sigma_{S.sid < R.sid} (R \times S)$

Equi Join ($\theta_=$)

- Equi Join is a special case of theta join where the condition can only contain **equality(=)** comparisons.
- When a theta join uses only an equivalence (=) condition, it becomes a Equi Join.
- **Notation:** $R \bowtie_{R.a1 = S.b1 \wedge \ldots\ldots \wedge R.an = S.bn} S$

Example of Equi Join ($\theta_=$)

R

Sid	Name	Rating	Age
22	A	7	20
31	B	8	17
58	C	10	16

S

Sid	Bid	Date
22	101	10/10/2015
58	102	02/03/2019
3	103	25/09/2007

$$\mathbf{R} \bowtie_{S.sid = R.sid} \mathbf{S}$$

R.Sid	Name	Rating	Age	S.Sid	Bid	Date
22	A	7	20	22	101	10/10/2015
58	C	10	16	58	102	02/03/2019

Equivalent Cartesian Product: $\sigma_{S.sid = R.sid} (R \times S)$

Example of Equi Join (θ =)

Student

SID	Name	Std
101	Rohan	11
102	Mira	12

Subjects

Class	Subject
11	Maths
11	Physics
12	English
12	Chemistry

$$\text{Student} \bowtie_{\text{Student.Std} = \text{Subjects.Class}} \text{Subjects}$$

SID	Name	Std	Class	Subject
101	Rohan	11	11	Maths
101	Rohan	11	11	Physics
102	Mira	12	12	English
102	Mira	12	12	Chemistry

Equivalent Cartesian Product: $\sigma_{SID = Class}$ (Student x Subjects)

Natural Join (⋈)

- A Natural Join can be performed only if two relations share at least one common attribute and the attributes must share the same name and domain.
- A comparison operator is not used in a natural join.
- It is same as Equi Join which occurs implicitly by comparing all the common attributes or columns in both the relation, ***but*** difference is that in Natural Join common attributes appears only once.
- In Natural Join, the resulting schema will be changed.

Natural Join (⋈)

- Notation: $R \bowtie S$
- The result of natural join is the set of all combination of tuples in two relations R and S that are equal on their common attributes names.
- A Natural Join of two relations can be obtained by applying a projection operation to Equi Join of two relations. In terms of basic operator:

Natural Join = Cartesian Product + Selection + Projection

- Natural join by default Inner Join because the tuples which does not satisfy the condition of join does not appear in the result set.

Example of Natural Join (⋈)

Relations: R = (A, B, C, D) and S = (B, D, E)

Resulting Schema R ⋈ S = (A, B, C, D, E)

R ⋈ S equals to: $\Pi_{R.A,\ R.B,\ R.C,\ R.D,\ S.E}(\sigma_{R.B = S.B \wedge R.D = S.D}(R \times S))$

R

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

S

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	ε

R ⋈ S

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

Example of Natural Join (⋈)

Courses

CID	Course	Dept
CS01	Database	CS
ME01	Mechanics	ME
EE01	Electronics	EE

HOD

Dept	Head
CS	Rohan
ME	Sara
EE	Jiya

Courses ⋈ HOD

CID	Course	Dept	Head
CS01	Database	CS	Rohan
ME01	Mechanics	ME	Sara
EE01	Electronics	EE	Jiya

Find the equivalent expression using Cartesian Product?

Example of Natural Join ($\bowtie$)

R

Sid	Name	Rating	Age
22	A	7	20
31	B	8	17
58	C	10	16

S

Sid	Bid	Date
22	101	10/10/2015
58	102	02/03/2019
3	103	25/09/2007

R $\bowtie$ S

Sid	Name	Rating	Age	Bid	Date
22	A	7	20	101	10/10/2015
58	C	10	16	102	02/03/2019

Find the equivalent expression using Cartesian Product?

Example of Natural Join (⋈)

Find the names of all customers who have a loan at the bank, along with the loan number and the loan amount.

Using Natural Join:

$$\Pi_{customer\text{-}name,\ loan.loan\text{-}number,\ amount}\ (borrower \bowtie loan)$$

Using Cartesian Product:

$$\Pi_{customer\text{-}name,\ loan.\ loan\text{-}number,\ amount}\ (\sigma_{borrower.\ loan\text{-}number\ =\ loan.\ loan\text{-}number}\ (borrower \times loan))$$

branch (branch-name, branch-city, assets)
customer(customer-name, customer-street, customer-city
account(account-number, branch-name, balance)
loan(loan-number, branch-name, amount)
depositor(customer-name, account-number)
borrower(customer-name, loan-number)

Example of Natural Join (⋈)

Find the names of all branches with customers who have an account in the bank and who live in Harrison.

$\Pi_{branch\text{-}name}(\sigma_{customer\text{-}city=\text{"Harrison"}}(customer \bowtie account \bowtie depositor))$

Note: Natural join is associative

$((customer \bowtie account) \bowtie depositor) = (customer \bowtie (account \bowtie depositor))$

So we can write it $(customer \bowtie account \bowtie depositor)$

branch (branch-name, branch-city, assets)
customer(customer-name, customer-street, customer-city)
account(account-number, branch-name, balance)
loan(loan-number, branch-name, amount)
depositor(customer-name, account-number)
borrower(customer-name, loan-number)

Example of Natural Join (⋈)

Find all customers who have ***both a loan and an account*** at the bank.

Using Natural Join:

$$\Pi_{customer\text{-}name}(\textbf{borrower} \bowtie \textbf{depositor}))$$

Using Set intersection:

$$\Pi_{customer\text{-}name}(\textbf{borrower}) \cap \Pi_{customer\text{-}name}(\textbf{depositor})$$

branch (branch-name, branch-city, assets)
customer (customer-name, customer-street, customer-city)
account(account-number, branch-name, balance)
loan(loan-number, branch-name, amount)
depositor (customer-name, account-number)
borrower (customer-name, loan-number)

Example of Natural Join (⋈)

Find the **names** of all customers who have a **loan** at the Perryridge **branch.**

Using Natural Join:

$$\Pi_{customer\text{-}name}\ (\sigma_{branch\text{-}name=\text{“Perryridge”}}\ (borrower \bowtie loan))$$

Using Cartesian Product:

$$\Pi_{customer\text{-}name}\ (\sigma_{branch\text{-}name=\text{“Perryridge”}}\ (\sigma_{borrower.\ loan\text{-}number\ =\ loan.\ loan\text{-}number}\ (borrower\ x\ loan)))$$

branch (<u>branch-name</u>, branch-city, assets)
customer(<u>customer-name</u>, customer-street, customer-city)
account(<u>account-number</u>, branch-name, balance)
loan(<u>loan-number</u>, branch-name, amount)
depositor(<u>customer-name</u>, <u>account-number</u>)
borrower(<u>customer-name</u>, <u>loan-number</u>)

Example of Natural Join (⋈)

Find the **names** of all customers who have a **loan** at the Perryridge **branch** but do **not** have an **account** at any **branch** of the bank.

$$\Pi_{customer\text{-}name}\,(\sigma_{branch\text{-}name=\text{"Perryridge"}}\,(borrower \bowtie loan)) - \Pi_{customer\text{-}name}\,(depositor)$$

branch (*branch-name*, *branch-city*, *assets*)
customer(*customer-name*, *customer-street*, *customer-city*)
account(*account-number*, *branch-name*, *balance*)
loan(*loan-number*, *branch-name*, *amount*)
depositor(*customer-name*, *account-number*)
borrower(*customer-name*, *loan-number*)

Outer Join

- An Outer Join operation is the extension of the join operation that avoids loss of information.
- It contains matching tuples that satisfy the matching condition, along with some or all tuples that do not satisfy the matching condition.
  - ✓ It is based on both matched and unmatched tuples.
  - ✓ It contains all rows from either one or both relations are present.
- It uses Null values.
- Null signifies that the value is unknown or does not exist.

Outer Join

R

Sid	Course
100	CSE 301
101	CSE 302
102	CSE 303

S

Sid	Name
100	A
102	B
104	C

Types of Outer join:

- Left Outer Join
- Right Outer Join
- Full Outer Join

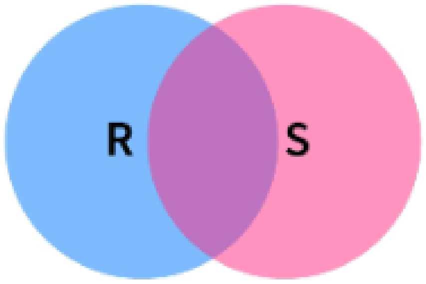
$(R \bowtie S)$ + Extra Information

Sid	Course	Name
100	CSE 301	A
102	CSE 303	B
...	...	...
...	...	...
...	...	...

Extra information comes from either the left or right table, or from both tables.

Left Outer Join (⟕)

- A Left Outer Join returns the matching tuples present in both relations and the tuples which are only present in left relation.
- The tuples of R which does not satisfy join condition will have values as NULL for attributes of S.
- Symbol: ⟕
- Notation: R ⟕ S



Left Outer Join

Example of Left Outer Join (⟕)

R

Sid	Course
100	CSE 301
101	CSE 302
102	CSE 303
103	CSE 404

S

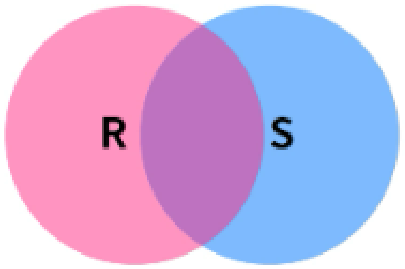
Sid	Name
100	A
102	B
104	C

(R ⟕ S)

Sid	Course	Name
100	CSE 301	A
101	CSE 302	**Null**
102	CSE 303	B
103	CSE 404	**Null**

Right Outer Join (⟖)

- A Right Outer Join returns the matching tuples present in both relations and the tuples which are only present in right relation.
- If the matching tuples are NULL, then the attributes of Left Relation, here R are made NULL in output relation.
- Symbol: ⟖
- Notation: R ⟖ S



Right Outer Join

Example of Right Outer Join ($⟖$)

R

Sid	Course
100	CSE 301
101	CSE 302
102	CSE 303

S

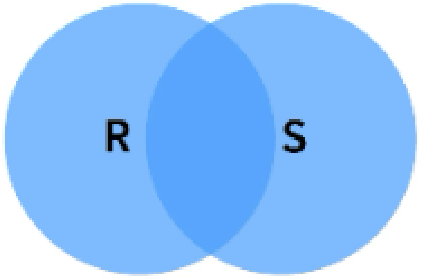
Sid	Name
100	A
102	B
104	C
105	D

(R $⟖$ S)

Sid	Course	Name
100	CSE 301	A
102	CSE 303	B
104	**Null**	C
105	**Null**	D

Full Outer Join (⟗)

- A Full Outer Join returns all the tuples from both relations.
- If there are no matching tuples then, their respective attributes are made NULL in output relation.
- Symbol: ⟗
- Notation: R ⟗ S



Full Outer Join

Example of Full Outer Join (⟗)

R

Sid	Course
100	CSE 301
101	CSE 302
102	CSE 303
103	CSE 404

S

Sid	Name
100	A
102	B
104	C
105	D

(R ⟗ S)

Sid	Course	Name
100	CSE 301	A
101	CSE 302	**Null**
102	CSE 302	B
103	**CSE 404**	**Null**
104	**Null**	C
105	**Null**	D

Example of Natural Join (⋈)

loan

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

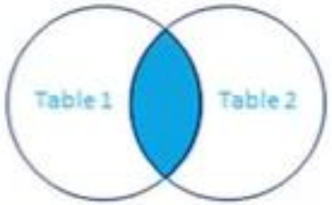
borrower

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

Loan ⋈ borrower

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

Inner Join



Example of Left Outer Join ($⟕$)

loan

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

borrower

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

Loan $⟕$ borrower

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

Example of Right Outer Join ($⟖$)

loan

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

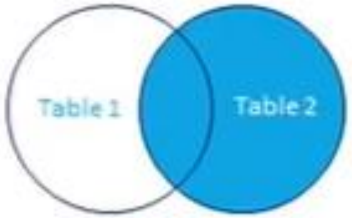
borrower

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

Loan $⟖$ borrower

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

Right Outer Join



Example of Full Outer Join ($⟗$)

loan

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

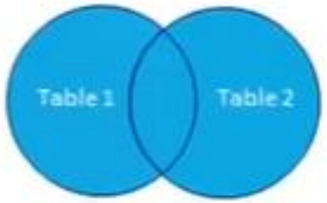
borrower

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

Loan $⟗$ borrower

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

Full Outer Join



Division Operator (÷ , /)

- Division operator is a derived operator, not supported as a primitive operator.
- Suited to queries that include the keyword "all", or "every" like "at all", "for all" or "in all", "at every", "for every" or "in every". E.g.
  - ✓ Find the person that has account in all the banks of a particular city.
  - ✓ Find sailors who have reserved all boats.
  - ✓ Find employees who works on all project of company.
  - ✓ Find students who have registered for every course.

Division Operator (÷ , /)

- "All" or "Every" defines a set which contains some elements, and the final result contains those records who satisfy these requirements.
- Notation: A ÷ B or A / B
- Division operator can be applied if and only if:
  - ✓ Attributes of B is proper subset of Attributes of A.
  - ✓ The relation returned by division operator will have attributes = (All attributes of A – All attributes of B).
  - ✓ The relation returned by division operator will return those tuples from relation A which are associated to every B's tuple.

Division Operator (÷ , /)

R

X	Y
A	1
B	2
A	3
D	4

÷

S

Y
1
3

→

(R ÷ S)

X
A

Example of Division Operator (÷ , /)

A

sno	pno
S1	P1
S1	P2
S1	P3
S1	P4
S2	P1
S2	P2
S3	P2
S4	P2
S4	P4

B1

pno
P2

B2

pno
P2
P4

B3

pno
P1
P2
P4

A/B1

sno
S1
S2
S3
S4

A/B2

sno
S1
S4

A/B3

sno
S1

Division Operator (÷ , /)

Student

Student_Name	Course_Name
Tom	DBMS
John	DS
Tom	DS
Tom	CN
John	DBMS
Amy	CN
Amy	DBMS
Amy	DS

÷

Course

Course_Name
DBMS
DS
CN

(Student ÷ Course)

Student_Name
Tom
Amy

Expressing Division Operator (÷ , /) using Basic Operator

- Division is a **derived operator** (or additional operator).
- Division can be expressed in terms of **Cross Product, Set-Difference and Projection.**

Idea:

- For A/B, compute all x values that are not 'disqualified' by some y value in B.
  - **x** value is disqualified if by attaching **y** value from **B**, we obtain an **xy tuple** that is not in **A**.

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

So $A/B = \Pi_x(A) -$ all disqualified tuples

$A/B = \Pi_x(A) - \Pi_x((\Pi_x(A) \times B) - A)$

a ***a, b, c, d*** ***b, d***

A

x	y
a	1
b	2
a	2
d	4

B

y
1
2

A÷B

x
a

Expressing Division Operator (÷ , /) using Basic Operator

A

sno	pno
S1	P1
S1	P2
S1	P3
S1	P4
S2	P1
S2	P2
S3	P2
S4	P2
S4	P4

B

pno
P2
P4

Disqualified x values: $\Pi_{sno}((\Pi_{sno}(A) \times B) - A)$

$\Pi_{sno}(A)$

sno
S1
S2
S3
S4

B

pno
P2
P4

$(\Pi_{sno}(A) \times B)$

sno	pno
S1	P2
S2	P2
S3	P2
S4	P2
S1	P4
S2	P4
S3	P4
S4	P4

A

sno	pno
S1	P1
S1	P2
S1	P3
S1	P4
S2	P1
S2	P2
S3	P2
S4	P2
S4	P4

$((\Pi_{sno}(A) \times B) - A)$

sno	pno
S2	P4
S3	P4

$\Pi_{sno}((\Pi_{sno}(A)$

sno
S2
S3

sno
S1
S2
S3
S4

–

sno
S2
S3

A/B

sno
S1
S4

A/B = $\Pi_{sno}(A)$ – all disqualified tuples

A/B = $\Pi_{sno}(A) - \Pi_{sno}((\Pi_{sno}(A) \times B) - A)$

Example of Division Operator (÷)

Find all customers who have an account at all branches located 'brooklyn' city

- 1st obtain all branches in Brooklyn by the expression:

$$r1 = \Pi_{branch\text{-}name}\,(\sigma_{branch\text{-}city=\text{“Brooklyn”}}(branch))$$

branch-name
Brighton
Downtown

- 2nd we can find all (customer-name, branch-name) pairs for which the customer has an account at branch by writing

$$r2 = \Pi_{customer\text{-}name,\,branch\ name}(depositor \bowtie account)$$

customer-name	branch-name
Hayes	Perryridge
Johnson	Downtown
Johnson	Brighton
Jones	Brighton
Lindsay	Redwood
Smith	Mianus
Turner	Round Hill

- Now, we need to find customers who appear in $r2$ *with every branch name in* $r1$. So $r2 \div r1$

$$\Pi_{customer\text{-}name,\,branch\_name}(depositor \bowtie account) \div \Pi_{branch\text{-}name}\,(\sigma_{branch\text{-}city=\text{“Brooklyn”}}(branch))$$

customer-name
Johnson

Exercise

A

A	B	C	D	E
α	a	α	a	1
α	a	γ	a	1
α	a	γ	b	1
β	a	γ	a	1
β	a	γ	b	3
γ	a	γ	a	1
γ	a	γ	b	1
γ	a	β	b	1

÷

B

D	E
a	1
b	1

?

$A/B = \prod_{A,B,C}(A) -$ all disqualified tuples

$A/B = \prod_{A.B,C}(A) - \prod_{A,B,C}((\prod_{A,B,C}(A) \times B) - A)$

Assignment Operator(←)

- The **assignment operation (←) provides** a **convenient way** to express complex queries.
  - It **writes query** as a sequential program consisting of:
    - a series of **assignments**
    - followed by an expression whose value is displayed as a **result** of the query.
  - **Assignment** must always be made to a **temporary relation variable.**
    - ***Division operation*** $A \div B = \Pi_X(A) - \Pi_X((\Pi_X(A) \times B) - A)$
    - Write $A \div B$ as

$$\textit{temp1} \leftarrow \Pi_x(A)$$

$$\textit{temp2} \leftarrow \Pi_x((\textit{temp1} \times B) - A)$$

$$\textit{result} \leftarrow \textit{temp1} - \textit{temp2}$$

- The result to the right of the ← is assigned to the relation variable on the left of the ←
- May use variable in subsequent expressions.

Aggregate Function

- **Aggregation function** *takes* ***a collection of values*** *and* ***returns a single value*** *as* ***a result.***

  avg: **average value**
  min: **minimum value**
  max: **maximum value**
  sum: **sum of values**
  count: **number of values**

  - These operations can be applied on **entire relation** or **certain groups of tuples**.
  - It ignore **NULL** values except **count**

- *Generalize form* (g) of **Aggregate operation**:

$$G1, G2, \ldots, Gn \; \mathcal{G}_{F1(A1), F2(A2), \ldots, Fn(An)} (E)$$

- E is any **relational-algebra expression**
- $G_1, G_2 \ldots, G_n$ is a **list of attributes** on which to **group** (can be empty)
- Each F_i is an **aggregate function**
- Each A_i is an **attribute name**

Example of Aggregate Function

Relation **r**

A	B	C
a	a	5
a	b	5
b	c	2
b	d	3

$\mathcal{G}_{sum(c)}(r)$

sum (c)
20

${}_{A}\mathcal{G}_{sum(c)}(r)$

A	sum (c)
a	10
b	5

Example of Aggregate Function

Relation '**account**' grouped by ***branch-name***:

branch_name	*account_number*	*balance*
Perryridge	A-102	400
Perryridge	A-201	900
Brighton	A-217	750
Brighton	A-215	750
Redwood	A-222	700

$_{branch\_name}\mathcal{G}_{sum(balance)}(account)$

branch_name	*sum(balance)*
Perryridge	1300
Brighton	1500
Redwood	700

Aggregate Function

- Result of aggregation ***does not have a name***
  - Can use **rename** operation to give it a name
  - For convenience, we permit renaming as part of aggregate operation using '**as**' keyword

$$_{branch\_name}\mathcal{G}_{\,sum(balance)\ \mathbf{as}\ sum\_balance}(account)$$

NULL Values

- *It is possible for* ***tuples*** *to have a* ***null value***, denoted by ***null***, for some of their attributes
- ***null*** signifies
  - **an unknown value/missing**, or
  - **a value that does not exist**

Emp_Id	Name	Phone_No	Passport_No
001	Rahul	7777777777	111 111 111
002	Anil	8888888888	**NULL**

First_Name	Middle_Name	Last Name
Ranjit	Singh	Thakur
Amit	**NULL**	Chopra

NULL Values

- The result of any arithmetic expression (+, -, *, /) involving ***null*** must return a ***null***.
  - *Eg: 5 + null = null*

    *null * 5 = null*
- Aggregate functions simply ignore **null values** (as in SQL)
  - Eg: Aggregate functions ***avg, min, max, sum*** ignores *null values* **except** for **count**
- For duplication, elimination and grouping, **null** is ***treated like any other value***, and two nulls are assumed to be the same (as in SQL)

NULL Values

- Comparisons (<, <=, >,>=, =, ≠) with **null values** evaluate to *special value* **unknown** (as in SQL)
  - Because we are not sure whether the result is *true* or *false*
  - *Eg: 5 = null, null > 5, 5 > null, null = null* ⟶ **unknown**
- Comparisons in **Boolean expressions** involving **AND, OR, NOT** operations uses ***three-valued logic*** i.e. true (1), false (0), unknown (as in SQL)
- ***Three-valued logic*** using the truth value ***unknown***:
  - **OR:** (*unknown* **or** *true*) = *true*,
    (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown*) = *unknown*
  - **AND:** (*true* **and** *unknown*) = *unknown*,
    (*false* **and** *unknown*) = *false*,
    (*unknown* **and** *unknown*) = *unknown*
  - **NOT:** (**not** *unknown*) = *unknown*
- Result of select predicate is treated as ***false*** if it evaluates to ***unknown***