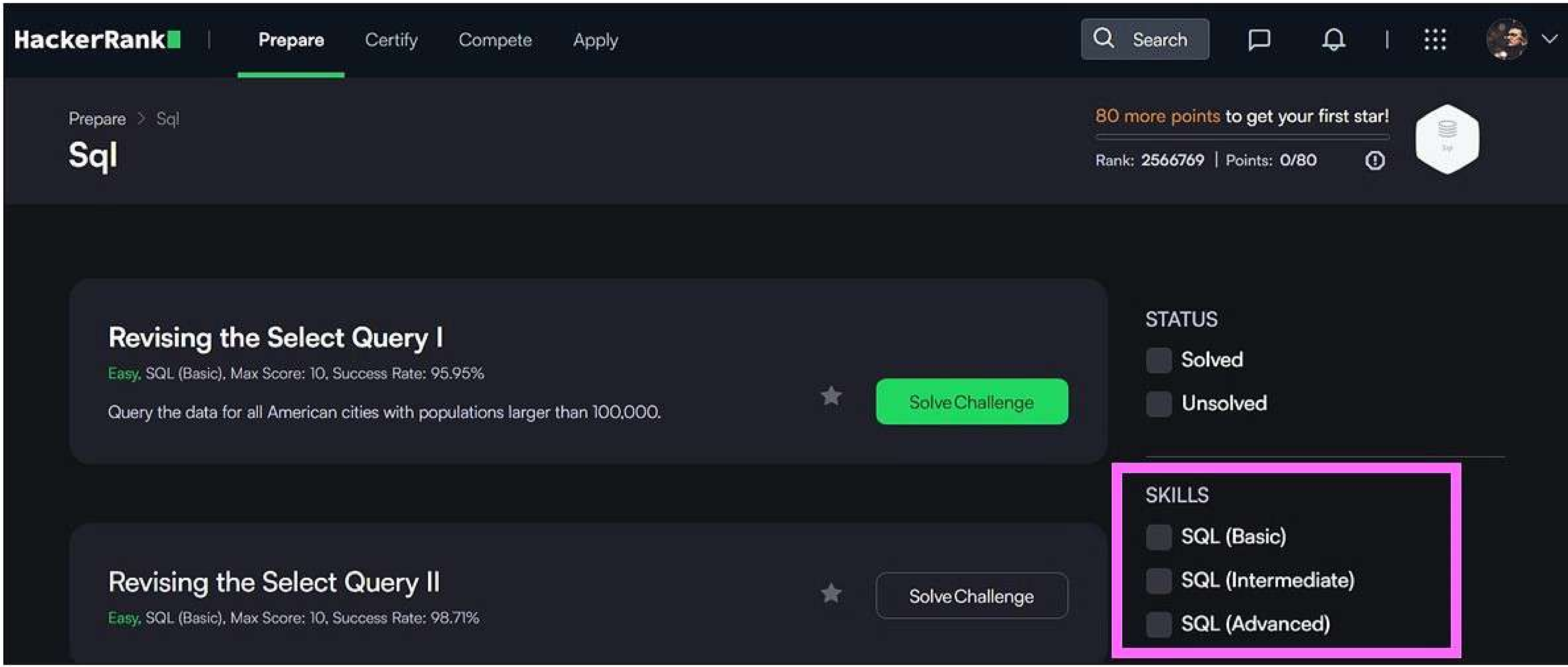
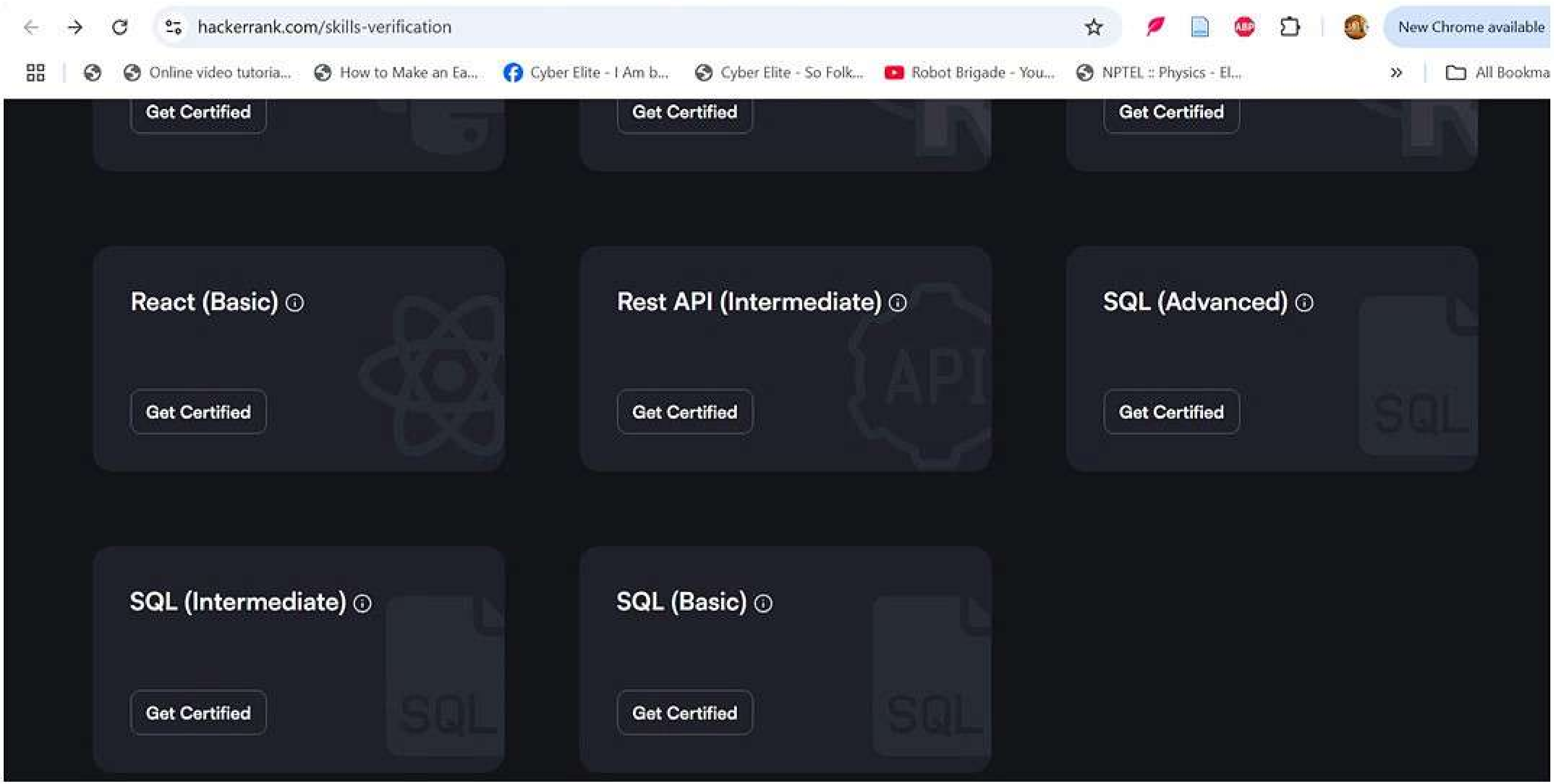
Assignment Task



Assignment Task



Consider a Student Table with lots of Attributes: Stud_Id, Name, Email, Age, Gender, DOJ, Height, Weight…..

Query: Retrieve All student_ID whose age is between 20 and 25

$$\pi_{Student\_ID}(\sigma_{Age>20 \wedge Age<25}(Students))$$

$$\pi_{Student\_ID}(\sigma_{Age>20 \wedge Age<25}(\pi_{Student\_ID, Age}(Students)))$$

$$\{t.Student\_ID \mid t \in Students \wedge 20 < t.Age < 25\}$$

Different Higher Level Strategies can be deployed

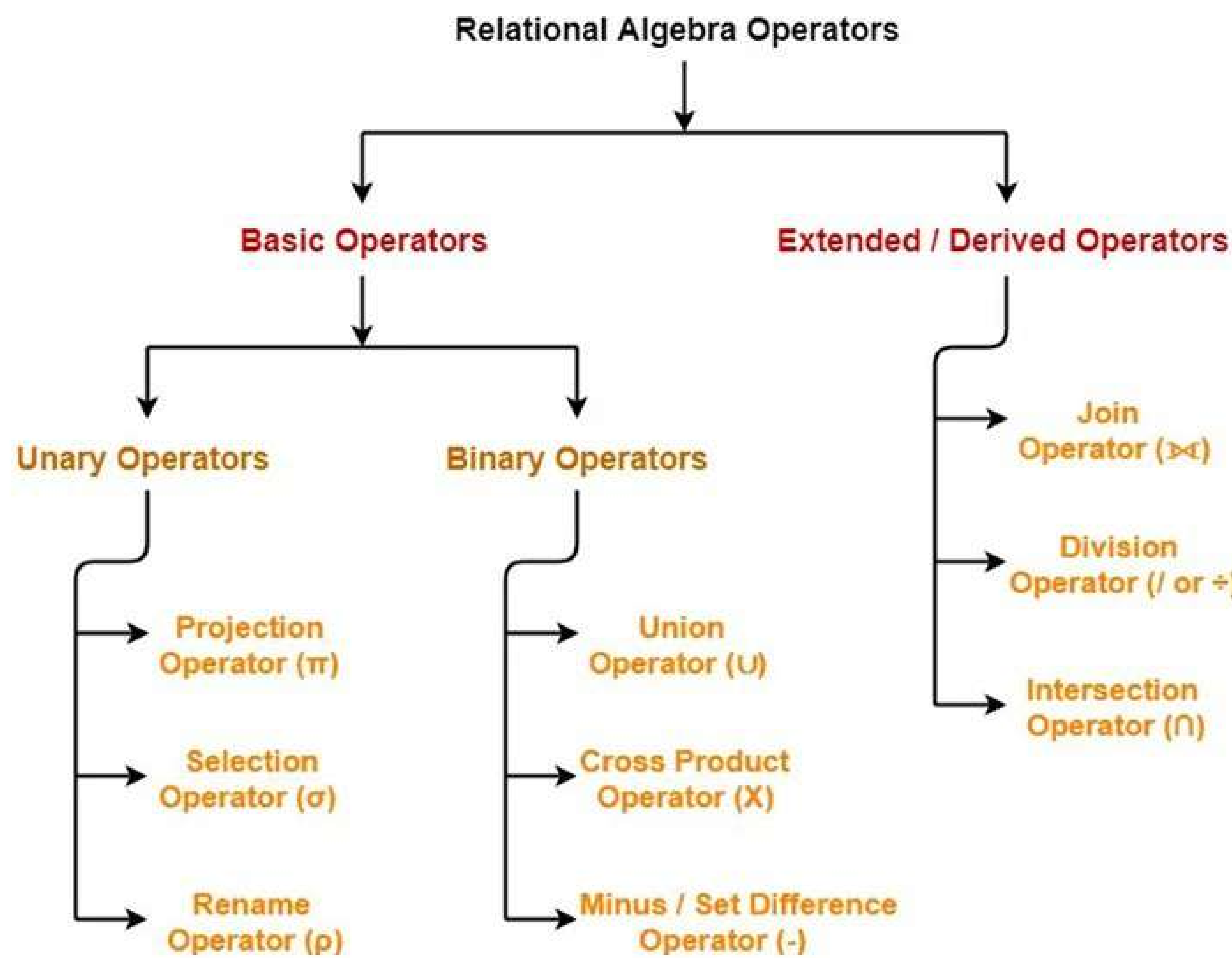
A single RA expression can be typically written in multiple ways.

One way can be *better* than the other, in terms of efficiency

A RC Expression can *typically* be specified in only one way.

You just declare what you want, You don't specify the sequence of operations.

Relational Algebra


Relational Algebra Operators
Basic Operators
Extended / Derived Operators
Unary Operators
Binary Operators
Join
Operator (⋈)
Division
Operator (/ or ÷)
Intersection
Operator (∩)
Projection
Operator (π)
Selection
Operator (σ)
Rename
Operator (ρ)
Union
Operator (∪)
Cross Product
Operator (X)
Minus / Set Difference
Operator (-)

Select Operator (σ)

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

$$\sigma_{(Department='CSE' \wedge Grade \geq 60)}(Students)$$ **Say 60 is the pass mark**

```
SELECT *
FROM Students
WHERE Department = 'CSE' AND Grade >= 60;
```

Select Operator (σ)

$$\sigma_{(Department='CSE' \wedge Grade \geq 60)}(Students)$$

```
SELECT *
FROM Students
WHERE Department = 'CSE' AND Grade >= 60;
```

Select Operator (σ)

It is used to select a subset of tuples from a relation that satisfies the given condition.

It is like a **filter**, tuples passing the filter condition ➔ Makes to the output.

Select operator, returns the entire Tuple in the O/P

Select Operator (σ)

Select operator, **Horizontally partitions** the Table.

Selection

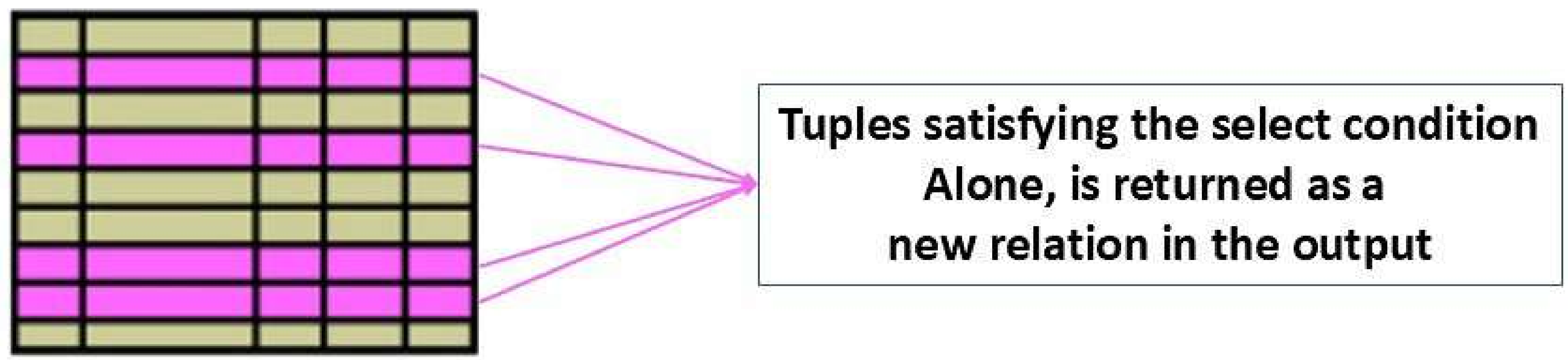


Table 1

Different Operators that can be used in the Select Condition

- $=$ (Equal to)
- $\neq$ (Not equal to)
- $<$ (Less than)
- $>$ (Greater than)
- $\leq$ (Less than or equal to)
- $\geq$ (Greater than or equal to)

Comparison Operator

- $\wedge$ (AND)
- $\vee$ (OR)
- $\neg$ (NOT)

Logical Operator

Can all comparison operators be applied to all attributes ?

$\sigma_{\text{Gender > Male}}$ (Students)

$\sigma_{\text{Color > Red}}$ (Painting)

Are they both Valid ?

Some attributes (Due to its domain nature) doesn't support some comparison operators

If the domains are "**Ordered Values**" then all the comparison operator's are applicable.

Integer, Float, Date, Grades are of ordered domains

Set of all colors, Name, Address, these things have no meaningful order.
Bob > Alice ➔ doesn't make any sense

{=, ≠} alone is applicable for unordered domains

How an Select expression is processed ?

The select condition is applied independently to each individual tuple *t* in R.

All tuple *t*, which satisfies / passes the condition appear in the result.

Select is Unary → Applied to only one single relation.
It is applied to each tuple individually.

Degree of the Output Relation

The $\underbrace{\text{degree of the relation}}_{\text{Number of Attributes in a Relation}}$ resulting from a SELECT operation is the same as the input table.

What about the Number of Tuples in the Resulting Relation ?

It is always lesser than or equal to the number of tuples in R (Input Relation)

Selectivity:

The fraction of tuples selected by a selection condition is referred to as the selectivity of the Condition.

Consider a Students table with 10,000 records.

σ Age > 18 (Students)

- If 9,000 students are older than 18, the selectivity = 9000 / 10000 = 0.9 (90%)
- This means the condition is not very selective (i.e., it retrieves most of the records).

σ Age = 21 (Students)

- If only 500 students are exactly 21, the selectivity = 500 / 10000 = 0.05 (5%)
- This means the condition is highly selective (i.e., it filters out most records)

Is Selection Operator Commutative ?

$$\sigma_{C1}(\sigma_{C2}(R)) = \sigma_{C2}(\sigma_{C1}(R))$$

Are they both same ?

Does applying condition **C1** first and then **C2** gives the same result as applying **C2** first and then **C1**. ?

$$\sigma_{Age>20}(\sigma_{Grade='A'}(Students))$$

$$\sigma_{Grade='A'}(\sigma_{Age>20}(Students))$$

Selection **only filters tuples** and does not change their structure. Since both C1 and C2 remove unwanted tuples, applying them in any order produces the same final set of tuples

Select Operator (σ) allows me to
Choose the desired rows,
What about columns how do I choose them ?

Project Operator (π)

$$\pi_{\text{<attribute list>}}(R)$$

$$\pi_{\text{Student\_ID, Name}}(Students)$$

```sql
SELECT Student_ID, Name FROM
Students;
```

Project Operator (π)

$$\pi_{\text{Student\_ID, Name}}(Students)$$

Are the both Same ?

$$\pi_{\text{Name, Student\_ID}}(Students)$$

The output relation, has the attributes listed in the exact order as specified in the Project condition

Vertical Partitioning of the Table

Projection

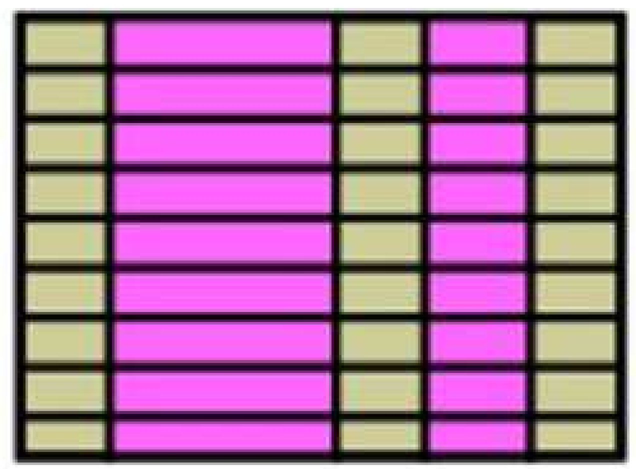


Table 1

Project Operator (π)

$$\underbrace{\pi_{\text{Name, Age}}}_{\text{Display the Desired Columns}}\underbrace{(\sigma_{\text{Grade}>80}(Students))}_{\text{Select the Desired Tuples}}$$

SELECT Name, Age

FROM *Students*

WHERE Grade > 80;

Degree of the Output Relation

Will the Degree change if you use the Project **(π)** Operator ?

$$\pi_{<\underline{\text{attribute list}}>}(R)$$

Degree of Output Relation:
No. of attributes mentioned here

$$\pi_{\text{Student\_ID, Name}}(Students)$$

This expression will output an Two degree relation.

Is Project Operator Commutative ?

Students (Student_ID, Name, Age, Grade)

$$\pi_{\text{Student\_ID, Name}}(\pi_{\text{Student\_ID, Name, Age}}(Students))$$

Are the both Same ?

$$\pi_{\text{Student\_ID, Name, Age}}(\pi_{\text{Student\_ID, Name}}(Students))$$

Is it even Valid at the first place ?

SQL Counterpart

$$\pi_{\text{Student\_ID, Name}}(\pi_{\text{Student\_ID, Name, Age}}(Students))$$

```
SELECT Student_ID, Name FROM (
       SELECT Student_ID, Name, Age FROM Students
)
```

Nested Sub Queries

SQL Counterpart

$$\pi_{\text{Student\_ID, Name}}(\pi_{\text{Student\_ID, Name, Age}}(Students))$$

```
SELECT Student_ID, Name FROM Students
```

Will this alone do the Job ?

Project Operator and Duplicates

Does Project Operator removes duplicates ?

Characters

Name	Gender	Age
Senku Ishigami	Male	15
Hermione	Female	14
Saitama	Male	26
Paul Atreides	Male	15

π_{Gender} (Characters)

Gender
Male
Female

$\pi_{\text{Gender, Age}}$ (Characters)

Gender	Age
Male	15
Female	14
Male	26

This row Appear Only once

Project Operator (π)

$$\pi_{\text{Name, Age}}(\sigma_{\text{Grade}>80}(Students))$$

You have to explicitly Instruct in SQL
To remove duplicates

SELECT DISTINCT Name, Age

FROM *Students*

WHERE Grade > 80;

Select Operator and Duplicates

Does Select Operator remove duplicates ?

Yes & No

Student_ID	Name	Age	Grade
101	Alice	20	85
102	Bob	21	90
103	Alice	20	85
104	David	22	80

$$\sigma_{\text{Age}=20}(\text{Students})$$

Student_ID	Name	Age	Grade
101	Alice	20	85
103	Alice	20	85

Select Operator and Duplicates

The SELECT operation (σ) in relational algebra does not remove duplicates because it only filters tuples based on a condition.

The result contains all tuples that satisfy the given condition, including duplicates if they exist in the original relation.

How do you deal with lengthy RA Expressions

$$\pi_{Student\_ID,Name}(\sigma_{Age>20}(\underline{\pi_{Student\_ID,Name,Age}(Students)}))$$

Store this as an intermediate relation
In a relation named as "**Temp**"

$$\rho_{Temp}(\pi_{Student\_ID,Name,Age}(Students))$$

$$\pi_{Student\_ID,Name}(\sigma_{Age>20}(Temp))$$

Rename Operator (ρ)

$$\rho_{NewName}(Expression)$$

The result of the expression will be renamed as “NewName”

$$\rho_{NewName(A1,A2,\ldots,An)}(Expression)$$

You can also explicitly rename each of the attributes, if you wish

Rename Operator (ρ)

```
SELECT Temp.Student_ID, Temp.Name

FROM (
       SELECT Student_ID, Name, Age
       FROM Students
       WHERE Age > 18
) AS Temp

WHERE Temp.Student_ID LIKE 'S%';
```

```
SELECT Student_ID, Name, Age
FROM Students
WHERE Age > 18 AS Temp
```

Filters students who are above 18 & Store it as Temp

```sql
SELECT Temp.Student_ID, Temp.Name

FROM (
      SELECT Student_ID, Name, Age
      FROM Students
      WHERE Age > 18
) AS Temp

WHERE Temp.Student_ID LIKE 'S%';
```

```sql
SELECT Temp.Student_ID, Temp.Name
FROM Temp
WHERE Temp.Student_ID LIKE 'S%';
```

retrieves the Student_ID and Name for students whose IDs start with the letter S from the alias Temp

Rename Operator (ρ) – What gets the Renamed ??

The rename operator (ρ) in Relational Algebra does not rename the original table permanently. Instead, it creates a temporary table with the new name, for the duration of the query.

The original relation remains unchanged.

Rename Operator (ρ) – What gets the Renamed ??

```
SELECT Student_ID, Name AS Full_Name, Age
FROM Students AS Temp_Students;
```

This is how You rename The attributes in SQL

```
ALTER TABLE Students RENAME TO StudentRecords;

ALTER TABLE Students RENAME COLUMN Name TO Full_Name;

ALTER TABLE Students MODIFY COLUMN Age VARCHAR(10);
```

Set Operations

Union Operator (U)

$$R_1 \cup R_2$$

```
SELECT column1, column2, ...FROM table1
UNION
SELECT column1, column2, ...FROM table2;
```

Can you apply Union operator between any two relations ?

Union Compatible Relations

Same Number of Attributes
Matching Datatypes

Students_AI Table

Student_ID	Name	Year
101	Aakash	2023
102	Bhavna	2023
103	Chaitanya	2024

What about these two rows ?

How many times will they appear in the o/p

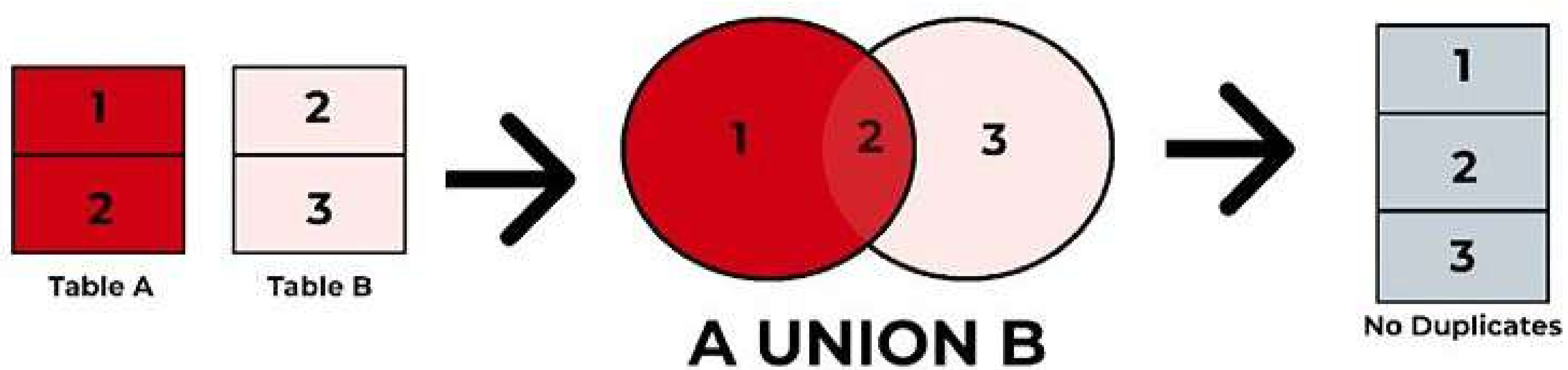
Students_DD Table

Student_ID	Name	Year
102	Bhavna	2023
104	Divya	2024
105	Eshwar	2024

Take Union between these two tables, what will be the Output ?

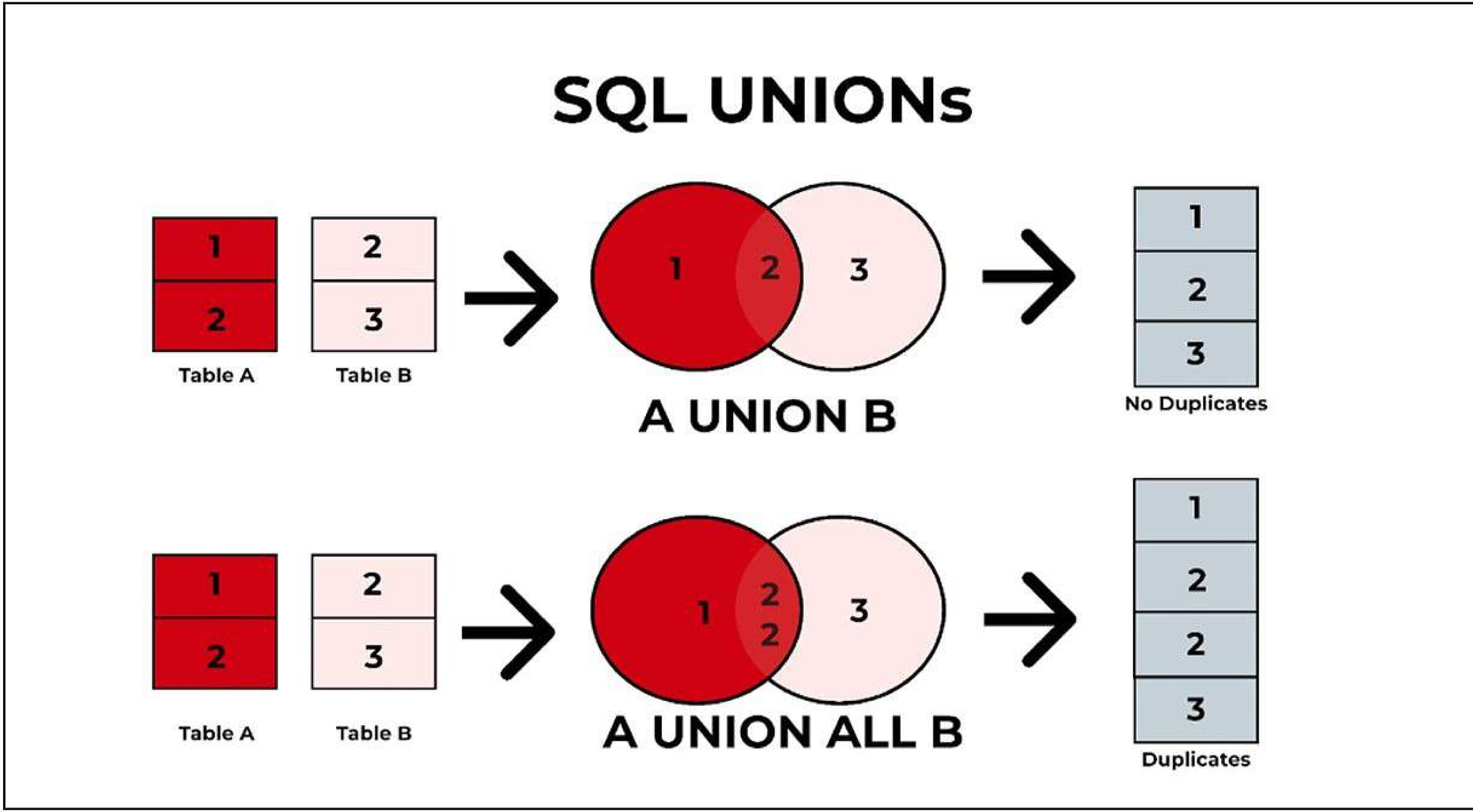
$$\pi_{Student\_ID,Name,Year}(Students\_AI) \cup \pi_{Student\_ID,Name,Year}(Students\_DD)$$

Student_ID	Name	Year
101	Aakash	2023
102	Bhavna	2023
103	Chaitanya	2024
104	Divya	2024
105	Eshwar	2024



What if I want the Duplicates ?

SQL (Alone) provides a special provision to retain duplicates


SQL UNIONs
1
2
Table A
2
3
Table B
1
2
3
A UNION B
1
2
3
No Duplicates
1
2
Table A
2
3
Table B
1
2
2
3
A UNION ALL B
1
2
2
3
Duplicates

Other Set Operators

- INTERSECTION: The result of this operation, denoted by $R \cap S$, 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.

Course_1

C_id	C_name
11	Foundation C
21	C++
31	JAVA

Course_2

C_id	C_name
12	Python
21	C++

Course_1 ∩ Course_2

C_id	C_name
21	C++

Note: Both the Tables should be “Union Compatible” for Intersect to work

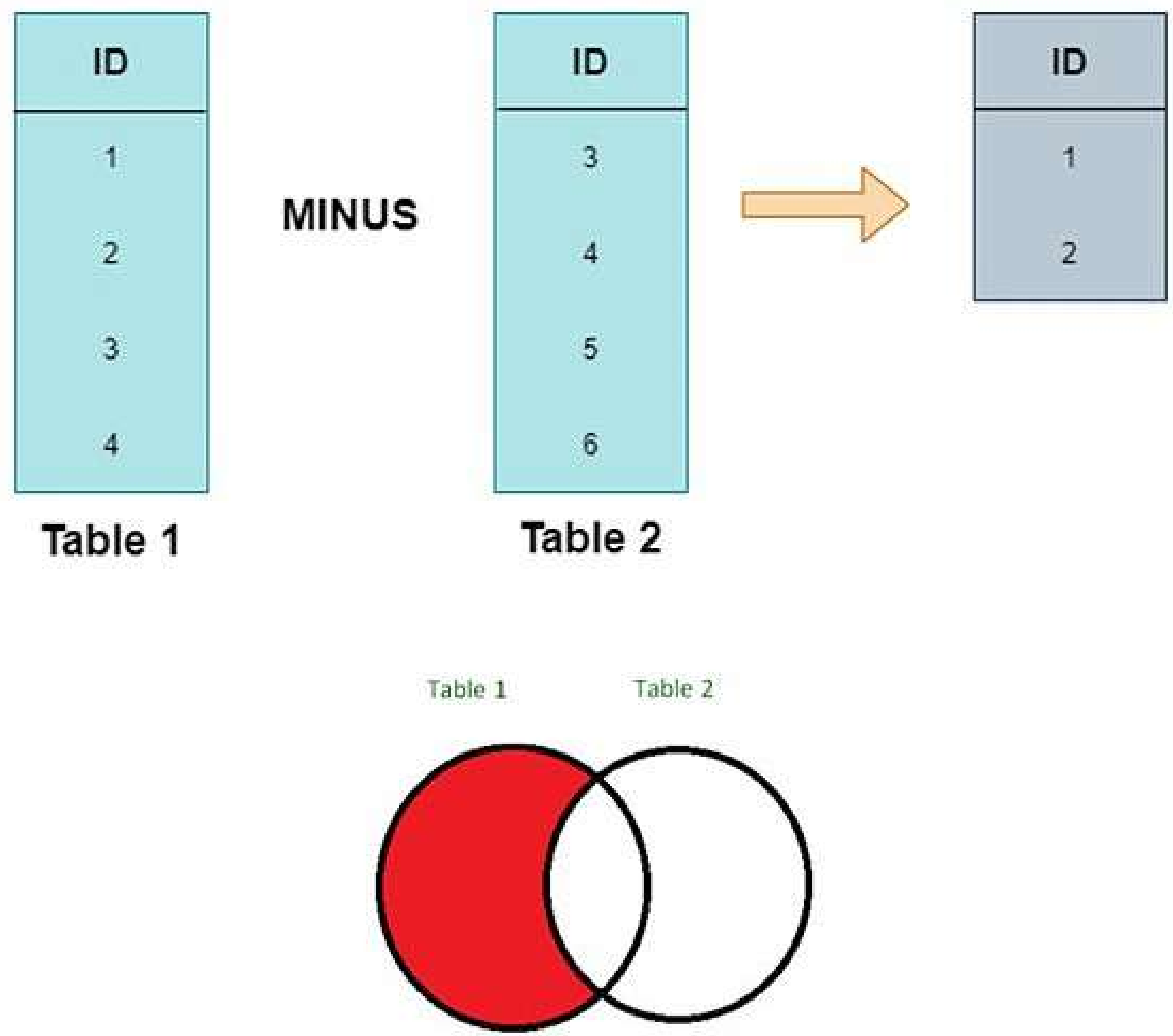
SQL Counterpart

```
SELECT column1, column2, ... FROM TableA
INTERSECT
SELECT column1, column2, ... FROM TableB;
```

```
SELECT Student_ID, Name, Age
FROM Students_AI
WHERE Age > 21

INTERSECT

SELECT Student_ID, Name, Age
FROM Students_DD
WHERE Age > 21;
```



Note: Both the Tables should be “Union Compatible” for Minus to work

SQL Counterpart

```
SELECT column1, column2, ...
FROM Table_A

EXCEPT

SELECT column1, column2, ...
FROM Table_B;
```

What about Commutativity ?

Cross Product

Name
John
Paul
Sara

Department
IT
Marketing

CROSS JOIN

Name	Department
John	IT
Paul	IT
Sara	IT
John	Marketing
Paul	Marketing
Sara	Marketing

Cross Product

1 Column

3 Rows

	Car_model
1	Camry
2	Corolla
3	Prius

1 Column

3 Rows

	Color_name
1	Black
2	Red
3	Silver

CROSS JOIN

1 + 1 = 2 Columns

	Car_model	Color_name
1	Camry	Black
2	Corolla	Black
3	Prius	Black
4	Camry	Red
5	Corolla	Red
6	Prius	Red
7	Camry	Silver
8	Corolla	Silver
9	Prius	Silver

3 x 3 = 9 Rows

A Cartesian product blindly combines all rows in all different ways possible, Most of them may be meaningless

Remember the case, where we wanted To combine related tuples of two Different tables

E_Name	Emp_id	Dept No
Eren	E_180	12
Mikasa	E_181	12

Related Tuples

Dept_Name	Dept_id	Start Year
Scout Regimen	12	1450
Military Corps	13	1300

In which year was the department where Eren works established?

How do you combine tuples belonging two tables But are related ?

You can take a cross product between both the relations.

This will create a very big table, where most of the rows are meaningless.

And then use a Select condition to filter out only the meaningful rows.

Employee

E_Name	Emp_id	Dept No
Eren	E_180	12
Mikasa	E_181	12

Department

Dept_Name	Dept_id	Start Year
Scout Regimen	12	1450
Military Corps	13	1300

Big Table = Employee X Department

E_Name	Emp_id	Dept No	Dept_Name	Dept_Id	Start Year
Eren	E_180	12	Scout Regimen	12	1450
Eren	E_180	12	Military Corps	13	1300
Mikasa	E_181	12	Scout Regimen	12	1450
Mikasa	E_181	12	Military Corps	13	1300

These two rows Are Meaningless.

Big Table

E_Name	Emp_id	Dept No	Dept_Name	Dept_Id	Start Year
Eren	E_180	12	Scout Regimen	12	1450
Eren	E_180	12	Military Corps	13	1300
Mikasa	E_181	12	Scout Regimen	12	1450
Mikasa	E_181	12	Military Corps	13	1300

$$\sigma_{\text{<Dept No = Dept\_Id>}} \text{ (Big Table)}$$

Actual_Emp_Dept__info

E_Name	Emp_id	Dept No	Dept_Name	Dept_Id	Start Year
Eren	E_180	12	Scout Regimen	12	1450
Mikasa	E_181	12	Scout Regimen	12	1450

SINCE, A Cartesian Product followed by a Select Condition is often used,

A dedicated operator was created for this purpose.

$$R \bowtie_{\text{<join condition>}} S$$

$$Employee \bowtie_{\text{Employee.dept\_id=Department.dept\_id}} Department$$

Helps to combine related tuples into "Longer Tuples", so that interesting information could be retrieved across tables.

```sql
SELECT D.Start_Year
FROM Employees E JOIN Departments D
ON E.Dept_No = D.Dept_id
WHERE E.E_Name = 'Eren';
```

Join Condition in SQL

$\sigma_{<Dept\ No\ =\ Dept\_Id>}$ (Big Table)

Do you find something Weird in this table ?

Actual_Emp_Dept__info

E_Name	Emp_id	Dept No	Dept_Name	Dept_Id	Start Year
Eren	E_180	12	Scout Regimen	12	1450
Mikasa	E_181	12	Scout Regimen	12	1450

Two Columns with same values

This is called as superfluous attribute, or Join Attribute

It would be nice, if we can get rid of that

Natural Join (*)

It is just Equijoin, followed by removal of the superfluous attribute

E_Name	Emp_id	Dept No

Dept_Name	Dept_id	Start Year

To apply Natural Join the "Join Attribute" should have exactly same name.

You can always
Rename the attributes
Using the rename
operator

Dept_Name	Dept_No	Start Year

Natural Join (*)

E_Name	Emp_id	Dept No

Dept_Name	Dept_No	Start Year

These two tables are natural join Compatible

To perform Natural Join do I need to specify any Join Condition ?

Employee * Department

```sql
SELECT Start_Year
FROM Employees NATURAL JOIN Departments
WHERE E_Name = 'Eren';
```

This work fine
If the join attribute
Has same name already

```sql
SELECT Start_Year
FROM (SELECT E_Name, Emp_id, Dept_No AS Dept_id FROM Employees) AS E
NATURAL JOIN Departments AS D
WHERE E_Name = 'Eren';
```

What about other Join Conditions (< , >...) ?

Products (information about products):

Product_ID	Product_Name	Price
101	Laptop	1200
102	Tablet	500
103	Smartphone	800

Suppliers (information about suppliers):

Supplier_ID	Supplier_Name	Max_Price
201	Supplier_A	1000
202	Supplier_B	1500

Find products whose price is less than or equal to the maximum price a supplier can handle.

What about other Join Conditions (< , >…) ?

Find products whose price is less than or equal to the maximum price a supplier can handle.

$$Products \bowtie_{Products.Price \leq Suppliers.Max\_Price} Suppliers$$

Product_Name	Price	Supplier_Name	Max_Price
Laptop	1200	Supplier_B	1500
Tablet	500	Supplier_A	1000
Tablet	500	Supplier_B	1500
Smartphone	800	Supplier_A	1000
Smartphone	800	Supplier_B	1500

Any General Join Condition:	**ϴ Join**
Join with Only Equality:	**Equijoin**
Join with implicit join attribute matching:	**Natural Join**

ϴ Join ➔ Equijoin ➔ Natural Join

More General ⟷ Specific / Restricted

Inner Joins

Θ Join ➔ Equijoin ➔ Natural Join

More General ⟷ Specific / Restricted

Hierarchy of Relationships

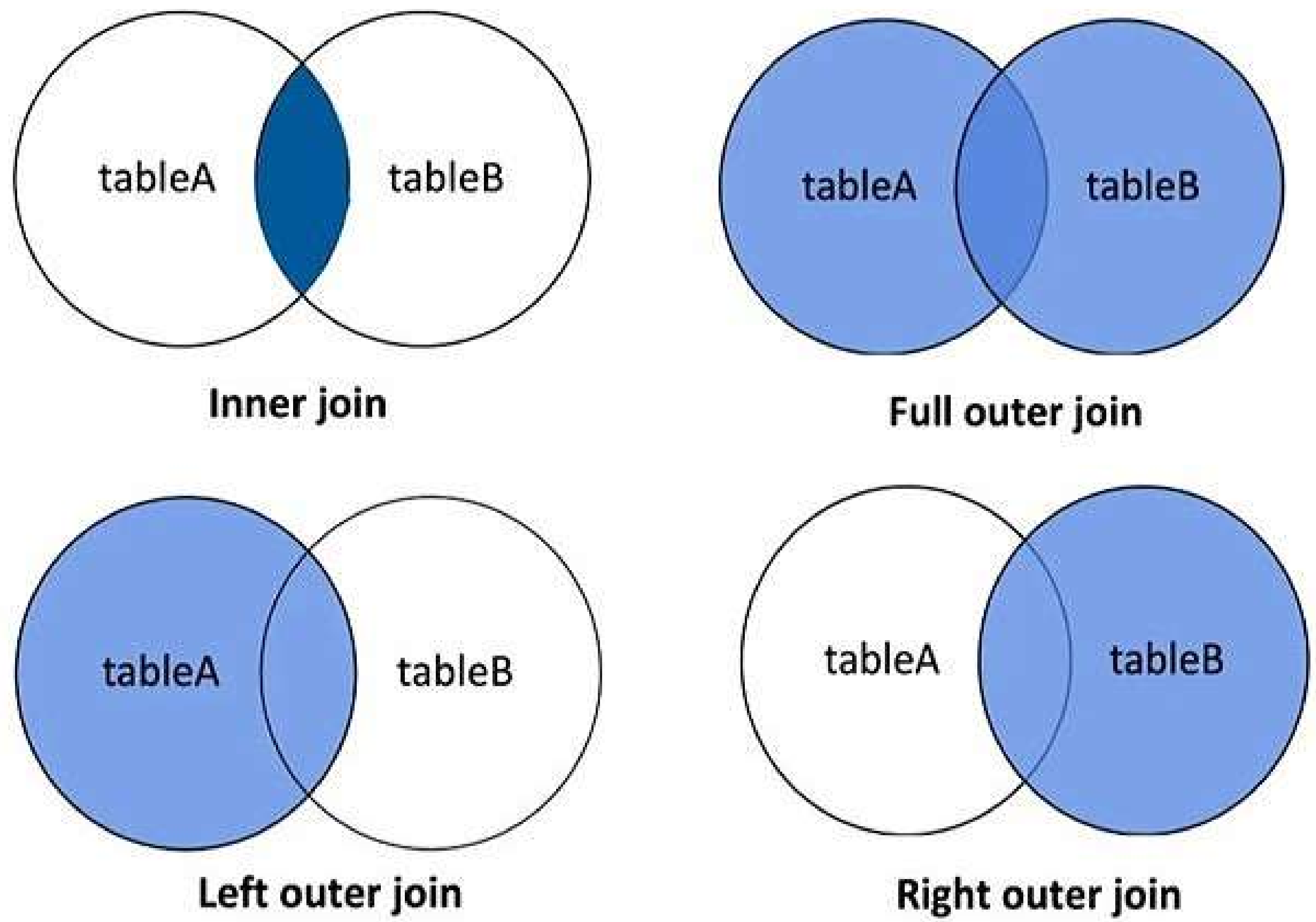
1. **Inner Join** is the broad category.

   2. **Theta Join** is a type of Inner Join that uses general comparison conditions.

   3. **Equi-Join** is a specific type of Theta Join where the comparison operator is `=`.

   4. **Natural Join** is a specialized form of Equi-Join where matching attributes are automatically identified and duplicate columns are removed.

Outer Join

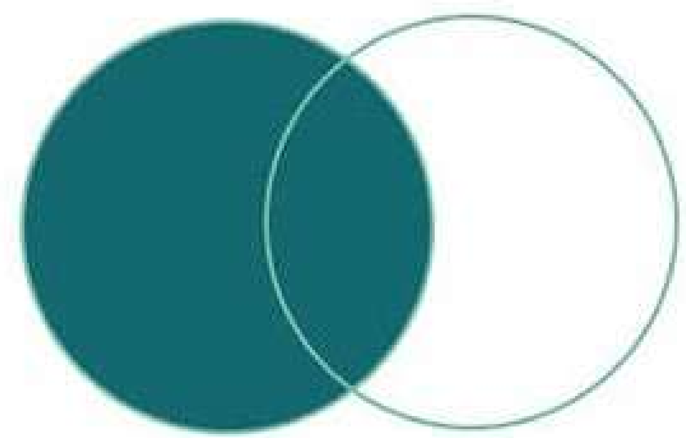

tableA
tableB
Inner join
tableA
tableB
Full outer join
tableA
tableB
Left outer join
tableA
tableB
Right outer join

Example

LEFT OUTER JOIN in SQL

Left Table

Food	Price
Pizza	$8
Burger	$6
French Fries	$4



Right Table

Food	Quantity
Pizza	3
Burger	5
Cake	2

Food	Price	Quantity
Pizza	$8	3
Burger	$6	5
French Fries	$4	Null

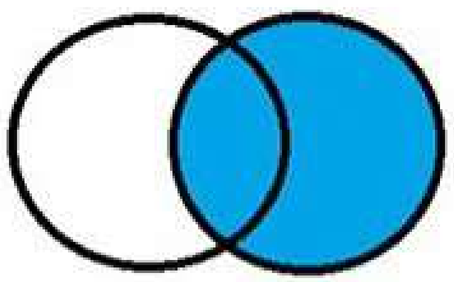
Note: There is a null in this row

INNER Join + all left out rows from the left table

Right Outer Join Example

Student ID	Name
1001	A
1002	B
1003	C
1004	D

Student ID	Department
1004	Mathematics
1005	Mathematics
1006	History
1007	Physics
1008	Computer Science



Student ID	Name	Department
1004	D	Mathematics
1005	NULL	Mathematics
1006	NULL	History
1007	NULL	Physics
1008	NULL	Computer Science

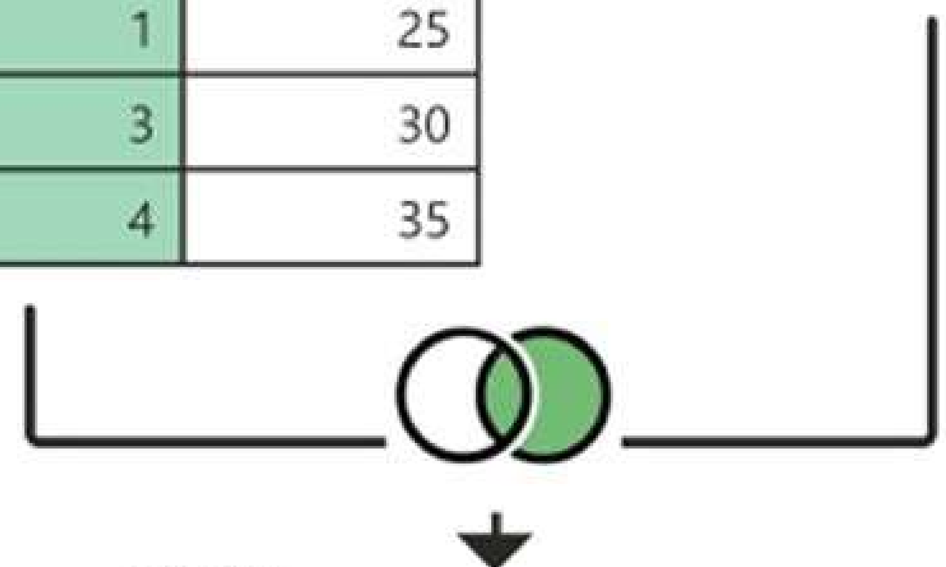
Right Outer Join Example

Left Table

Date	CountryID	Units
1/1/2020	1	40
1/2/2020	1	25
1/3/2020	3	30
1/4/2020	4	35

Right Table

ID	Country
3	Panama



Merged Table

Date	CountryID	Units	Country
1/3/2020	3	30	Panama

Why only one row in the RO Join ?

Remember: Whatever Matches + Whatever that remains (unmatched) in the Right Table

Outer Join = Theta Join + Left / Right or Both

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

Division Operator

R

ColA	ColB
F	1
F	2
F	3
E	1
E	3
S	1
S	2

S

ColB
1
2

$R \div S =$

ColA
F
S

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

Find students who have passed all subjects in their curriculum.

$$\pi_{Student\_ID}(\text{Exam\_Results}) \div \pi_{Subject\_ID}(\text{Curriculum})$$

Find sales representatives who have sold all products in a given region ‘r’.

$$\pi_{Rep\_ID}(\text{Sales}) \div \pi_{Product\_ID}(\sigma_{Region=r}(\text{Products}))$$

Division Operator

$$\pi_{Scholar\_ID}(\text{Publications}) \div \pi_{Journal\_ID}(\sigma_{Tier=1}(\text{Journals}))$$

Find research scholars who have published in all tier 1 journals.

$$\pi_{Student\_ID}(\text{Workshop\_Attendance}) \div \pi_{Workshop\_ID}(\sigma_{Mandatory=True}(\text{Workshops}))$$

Find students who have attended all mandatory workshops for their course.

Operations in Relational Algebra

1. Selection
2. Projection
3. Renaming
4. Cross Product
5. Set Difference
6. Union

1. Joins
2. Intersection
3. Division

Do you find anything special about this collection of operators ?

Functionally Complete Set:

A set of operators is functionally complete if all other relational operations can be expressed using only the operators in this set.

Table 6.1 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 $R_1 \bowtie_{\text{(<join attributes 1>),}\ \text{(<join attributes 2>)}} R_2$
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 $R_1 *_{\text{(<join attributes 1>),}\ \text{(<join attributes 2>)}} R_2$ OR $R_1 * R_2$
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 $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

Does Foreign Key play any role in Joining Tables ?

Typically the join attribute will be a Foreign key.

- Many database systems automatically create an **index** on foreign key columns, making joins faster.
- Even though the join condition doesn't require a foreign key, having one allows the database engine to **optimize join execution**.

Other than this remember that F.Key always has its own job:

A foreign key constraint guarantees that every foreign key value in one table must exist as a primary key value in another table.

Without a foreign key, you might accidentally insert or update a value that doesn't have a corresponding primary key in the referenced table, leading to inconsistent data

AKA, Maintain Referential Integrity

Other than this remember that F.Key always has its own job:

Prevents Orphaned Records

- If you delete a referenced record (e.g., a department), the foreign key constraint prevents the deletion if related records exist in the referencing table (students belonging to that department).

- You can also define **ON DELETE CASCADE**, so deleting a department automatically deletes all its related students.