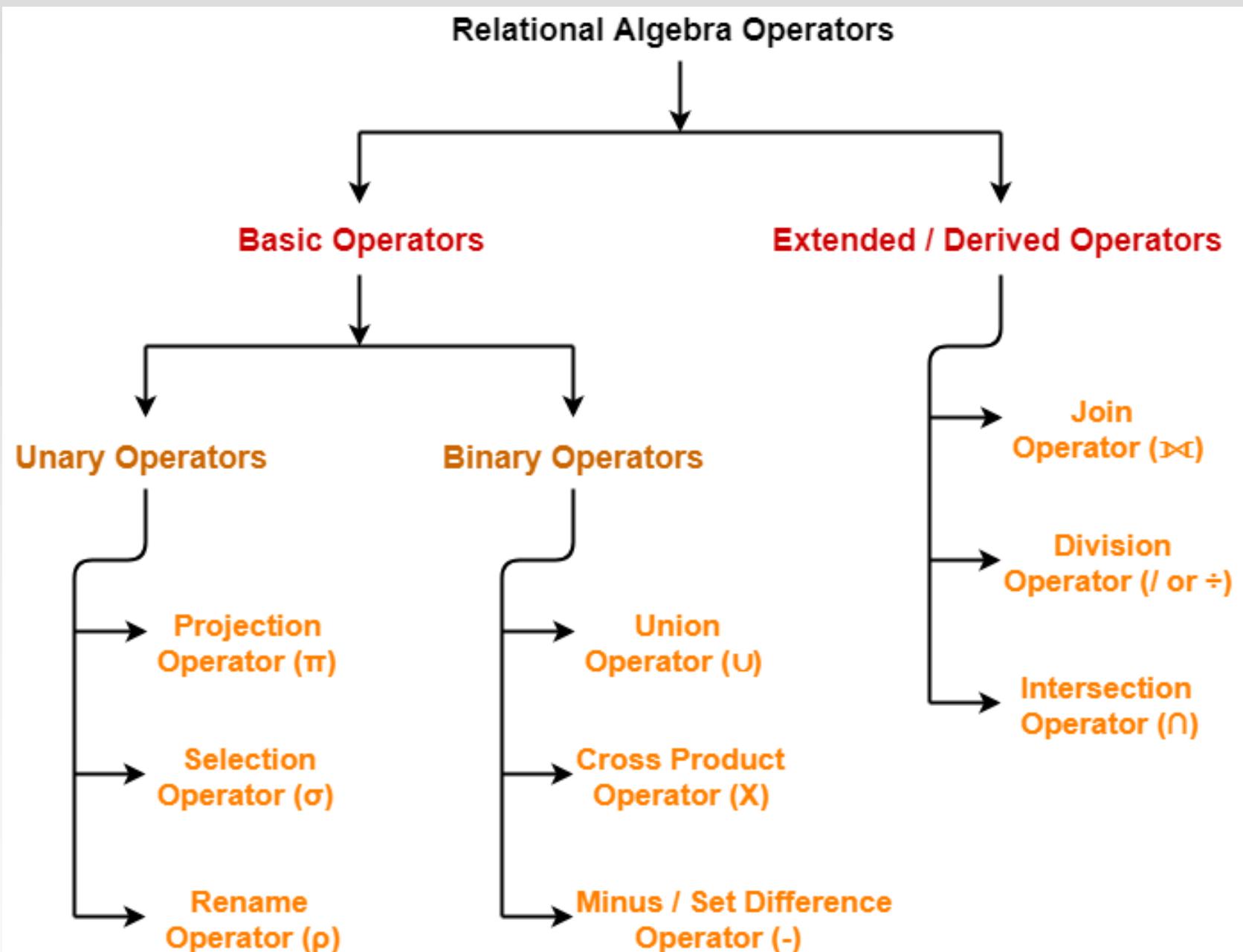


Database Management Systems

Shamim Ahmad



Examples of SQL statements



Projection (Π) → picks columns.

Selection (σ) → picks rows (tuples) based on conditions

Select and Project Operation

SELECT (σ)

The SELECT operation is used for selecting a **subset of the tuples (Horizontal subset ?)** according to a given selection condition.

Select operator selects tuples that satisfy a given predicate

Projection(π)

The projection eliminates all attributes of the input relation but those mentioned in the projection list. The projection method defines a relation that contains values for a list all attributes **(vertical subset ?) of Relation.**

Roll	Name	Department	Fees	Team
1	Bikash	CSE	22000	A
2	Josh	CSE	34000	A
3	Kevin	ECE	36000	C
4	Ben	ECE	56000	D

Select all the student of Team A : $\sigma_{\text{Team} = 'A'}$ (Student)

Roll	Name	Department	Fees	Team
1	Bikash	CSE	22000	A
2	Josh	CSE	34000	A

Roll	Name	Department	Fees	Team
1	Bikash	CSE	22000	A
2	Josh	CSE	34000	A
3	Kevin	ECE	36000	C
4	Ben	ECE	56000	D

Select all the students of department ECE whose fees is greater than equal to 10000 and belongs to Team other than A.

$$\sigma_{\text{Fees} \geq 10000}(\sigma_{\text{Class} \neq 'A'} (\text{Student}))$$

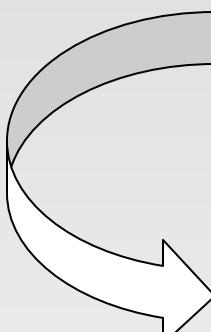
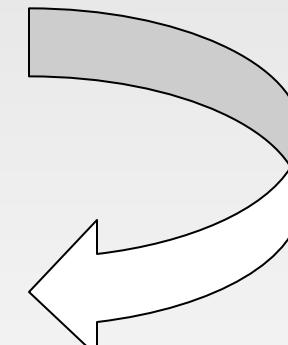
Roll	Name	Department	Fees	Team
1	Bikash	CSE	22000	A
2	Josh	CSE	34000	A
3	Kevin	ECE	36000	C
4	Ben	ECE	56000	D

Select all the students of department ECE whose fees is greater than equal to 10000 and belongs to Team other than A.

$$\sigma_{\text{Fees} \geq 10000}(\sigma_{\text{Class} \neq 'A'} (\text{Student}))$$

Roll	Name	Department	Fees	Team
3	Kevin	ECE	36000	C
4	Ben	ECE	56000	D

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive
4	Alibaba	Active


$$\Pi_{\text{CustomerName}, \text{Status}} (\text{Customers})$$


CustomerName	Status
Google	Active
Amazon	Active
Apple	Inactive
Alibaba	Active

StudentID	Name	Department	GPA
1	Alice	CSE	3.8
2	Bob	EEE	3.5
3	Charlie	CSE	3.9
4	Diana	BBA	3.6

SELECT Name FROM Students;

StudentID	Name	Department	GPA
1	Alice	CSE	3.8
2	Bob	EEE	3.5
3	Charlie	CSE	3.9
4	Diana	BBA	3.6

SELECT Name FROM Students;

Name
Alice
Bob
Charlie
Diana

StudentID	Name	Department	GPA
1	Alice	CSE	3.8
2	Bob	EEE	3.5
3	Charlie	CSE	3.9
4	Diana	BBA	3.6

SELECT Name, Department FROM Students;

StudentID	Name	Department	GPA
1	Alice	CSE	3.8
2	Bob	EEE	3.5
3	Charlie	CSE	3.9
4	Diana	BBA	3.6

SELECT Name, Department FROM Students;

Name	Department
Alice	CSE
Bob	EEE
Charlie	CSE
Diana	BBA

StudentID	Name	Department	GPA
1	Alice	CSE	3.8
2	Bob	EEE	3.5
3	Charlie	CSE	3.9
4	Diana	BBA	3.6

```
SELECT *
FROM Students
WHERE Department = 'CSE';
```

StudentID	Name	Department	GPA
1	Alice	CSE	3.8
2	Bob	EEE	3.5
3	Charlie	CSE	3.9
4	Diana	BBA	3.6

SELECT * FROM Students WHERE Department = 'CSE';

StudentID	Name	Department	GPA
1	Alice	CSE	3.8
3	Charlie	CSE	3.9

StudentID	Name	Department	GPA
1	Alice	CSE	3.8
2	Bob	EEE	3.5
3	Charlie	CSE	3.9
4	Diana	BBA	3.6

```
SELECT *
FROM Students
WHERE GPA > 3.6;
```

StudentID	Name	Department	GPA
1	Alice	CSE	3.8
3	Charlie	CSE	3.9

Operations on Tables

1. DELETE

- Removes **specific rows** from a table (can use WHERE clause).
- Data is deleted**, but the **table structure remains**.
- It is a **DML** (Data Manipulation Language) command.
- Transaction **can be rolled back** (if within a transaction).
- Slower than TRUNCATE (removes row by row).

Example:

```
DELETE FROM Students WHERE City = 'Dhaka';
```

Removes only rows where City = 'Dhaka'. Table Students still exists, with other rows intact

2. TRUNCATE

- Removes **all rows** from a table (**no WHERE clause allowed**).
- Table structure remains**, so you can still insert data later.
- It is a **DDL (Data Definition Language)** command.
- Cannot delete specific rows.
- Faster than DELETE (does not log individual row deletions).
- Cannot usually be rolled back (depends on DBMS, but in SQL Server/Oracle it's not reversible once executed outside a transaction)

```
TRUNCATE TABLE Students;
```

- Removes **all rows** from Students.
- Table Students still exists, but it's now empty.

DROP

- Removes the **entire table** (structure + data).
- It is a **DDL** command.
- Once dropped, the table is gone permanently (unless restored from backup).
- Cannot be rolled back (outside of transactional support).

Example:

```
DROP TABLE Students;
```

- The table Students is completely deleted (data + structure).
- You cannot insert into it anymore unless you recreate it.

Set Operation

The Union Operation

Query

To find the names of all **bank customers** who have
either an account **or** a loan **or** both.

The borrower relation

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

The customer Relation

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

The depositor Relation

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

The Union Operation

Query

To find the names of all **bank customers** who have
either an account **or** a loan **or** both.

The *borrower* relation

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

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

The *depositor* Relation

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

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

The *customer* Relation

<i>customer-name</i>	<i>customer-street</i>	<i>customer-city</i>
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

Example Queries

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

$$\Pi_{customer_name} (borrower) \cup \Pi_{customer_name} (depositor)$$

The *borrower* relation

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

The *depositor* Relation

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

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

```
SELECT * FROM Info UNION SELECT * FROM Demo;
```

Explanation:

UNION combines the results of two SELECT queries. It removes duplicate rows from the final result set (if you want duplicates, you'd use UNION ALL).

For UNION to work:

1. Both queries must select the same number of columns.
2. The data types in each corresponding column must be compatible (e.g., INT with INT, VARCHAR with VARCHAR).

Table: Info

ID	Name	City
1	Alice	Dhaka
2	Rashed	Rajshahi
3	Obony	Khulna

Table: Demo

ID	Name	City
2	Rashed	Rajshahi
4	Karim	Dhaka
5	Nabila	Sylhet

SELECT * FROM Info UNION SELECT * FROM Demo;

ID	Name	City
1	Alice	Dhaka
2	Rashed	Rajshahi
3	Obony	Khulna
4	Karim	Dhaka
5	Nabila	Sylhet

```
SELECT * FROM Info  
UNION ALL  
SELECT * FROM Demo  
ORDER BY Name;
```

- **UNION ALL**

- Combines results from both Info and Demo.
- Unlike UNION, it **does not remove duplicates**.

- **ORDER BY Name**

- After combining, the full result set is sorted by the column Name.
- Requirements**
- Both tables must have the same number of columns.
- Corresponding columns must have compatible data type

Table: Info

ID	Name	City
1	Alice	Dhaka
2	Rashed	Rajshahi
3	Obony	Khulna

Table: Demo

ID	Name	City
2	Rashed	Rajshahi
4	Karim	Dhaka
5	Nabila	Sylhet

```
SELECT * FROM Info UNION ALL  
SELECT * FROM Demo  
ORDER BY Name;
```

ID	Name	City
1	Alice	Dhaka
4	Karim	Dhaka
5	Nabila	Sylhet
3	Obony	Khulna
2	Rashed	Rajshahi
2	Rashed	Rajshahi

Set-Intersection Operation – Example

- Relation r, s :

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r \cap s$

A	B
α	2

Query

Find all customers who have both a loan and an account.

The *borrower* relation

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

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

The *depositor* Relation

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

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

<i>customer-name</i>
Hayes
Jones
Smith

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

```
SELECT * FROM Info  
INTERSECT  
SELECT * FROM Demo;
```

Explanation

- INTERSECT returns only the rows that **exist in both** Info and Demo.
- Duplicate rows are automatically removed (similar to UNION).
- The number of columns and their data types must match in both queries.
- Not all DBMS (**like MySQL**) **directly support INTERSECT** – but Oracle, SQL Server, and PostgreSQL do. In MySQL, you simulate it using INNER JOIN or IN

Table: Info

ID	Name	City
1	Alice	Dhaka
2	Rashed	Rajshahi
3	Obony	Khulna

Table: Demo

ID	Name	City
2	Rashed	Rajshahi
4	Karim	Dhaka
5	Nabila	Sylhet

```
SELECT * FROM Info  
INTERSECT  
SELECT * FROM Demo;
```

ID	Name	City
2	Rashed	Rajshahi

Equivalent **in MySQL** (since **INTERSECT** is not supported)

```
SELECT * FROM Info  
INNER JOIN Demo  
USING (ID, Name, City);
```

Join operation

What is a JOIN?

A **join** combines rows from two (or more) tables based on a related column (the *join key*).

General form

```
SELECT ...
FROM A
JOIN B ON A.key = B.key
```

Without a condition (**CROSS JOIN**), you get the **Cartesian product** (every row of A with every row of B)

INNER JOIN

Returns only the rows where the join condition matches in **both** tables

LEFT OUTER JOIN ALL

Rows from the left table; matching rows from the right; non-matches on the right become NULL

RIGHT OUTER JOIN

Mirror image of LEFT: all rows from the right table, NULLs for missing left rows.

Employees

emp_id	name	dept_id
1	Amina	10
2	Bilal	20
3	Chen	NULL
4	Diana	30

Departments

dept_id	dept_name
10	Sales
20	HR
40	Research

Employees

emp_id	name	dept_id
1	Amina	10
2	Bilal	20
3	Chen	NULL
4	Diana	30

Departments

dept_id	dept_name
10	Sales
20	HR
40	Research

INNER JOIN

emp_id	name	dept_name
1	Amina	Sales
2	Bilal	HR

Employees

emp_id	name	dept_id
1	Amina	10
2	Bilal	20
3	Chen	NULL
4	Diana	30

Departments

dept_id	dept_name
10	Sales
20	HR
40	Research

LEFT OUTER JOIN

emp_id	name	dept_name
1	Amina	Sales
2	Bilal	HR
3	Chen	NULL
4	Diana	NULL

Employees

emp_id	name	dept_id
1	Amina	10
2	Bilal	20
3	Chen	NULL
4	Diana	30

Departments

dept_id	dept_name
10	Sales
20	HR
40	Research

RIGHT OUTER JOI

emp_id	name	dept_name
1	Amina	Sales
2	Bilal	HR
NULL	NULL	Research

Employees

emp_id	name	dept_id
1	Amina	10
2	Bilal	20
3	Chen	NULL
4	Diana	30

Departments

dept_id	dept_name
10	Sales
20	HR
40	Research

FULL OUTER JOIN

emp_id	name	dept_name
1	Amina	Sales
2	Bilal	HR
3	Chen	NULL
4	Diana	NULL
NULL	NULL	Research

Outer Join – Example

□ Relation *loan*

<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

□ Relation *borrower*

<i>customer_name</i>	<i>loan_number</i>
Jones	L-170
Smith	L-230
Hayes	L-155

Outer Join – Example

□ Join

loan \bowtie *borrower*

<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>	<i>customer_name</i>
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith

□ Left Outer Join

loan $\text{L}\bowtie$ *borrower*

<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>	<i>customer_name</i>
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	<i>null</i>

Outer Join – Example

- Right Outer Join

loan \bowtie *borrower*

<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>	<i>customer_name</i>
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	<i>null</i>	<i>null</i>	Hayes

- Full Outer Join

loan $\bowtie\bowtie$ *borrower*

<i>loan_number</i>	<i>branch_name</i>	<i>amount</i>	<i>customer_name</i>
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	<i>null</i>
L-155	<i>null</i>	<i>null</i>	Hayes

Table: Students

ID	Name	City
1	Alice	Dhaka
2	Rashed	Rajshahi
3	Obony	Khulna

Table: Marks

ID	Subject	Score
1	Math	90
2	Math	85
3	Math	70

```
SELECT Students.Name, Students.City, Marks.Score  
FROM Students  
INNER JOIN Marks  
ON Students.ID = Marks.ID;
```

Name	City	Score
Alice	Dhaka	90
Rashed	Rajshahi	85
Obony	Khulna	70

Aspect	Set Operations (UNION, INTERSECT, EXCEPT)	Join Operations (INNER JOIN, LEFT JOIN, etc.)
Combination level	Combine complete result sets (rows stacked vertically).	Combine columns from multiple tables (rows expanded horizontally).
Number of columns	Must be same number and type in both queries.	Can be different ; columns are matched using a condition.

Duplicates	UNION removes duplicates, UNION ALL keeps them.	Duplicates depend on join condition (may produce Cartesian product if not careful).
Purpose	To merge two query results with the same structure .	To merge data across tables based on relationships/keys .
Output	Rows from one query added on top of rows from another.	Rows from one table combined side by side with rows from another.
Typical usage	Compare or merge results of two similar queries/tables .	Combine related data spread across different tables .