Database Management Systems
Relational Algebra: Basics

Vikas Thammanna Gowda

01/31/2025

1 Projection as Applied in Relational Algebra and SQL

1.1 Relational Algebra Projection (π)

Purpose: Retrieves specific columns (attributes) from a relation (table). It focuses on reducing the number of columns in the output.

Mathematical Notation:

$$\pi_{A_1,A_2,\ldots,A_n}(R)$$

where:

- $A_1, A_2, \ldots, A_n$ are the attributes (columns) you want to keep.
- R is the relation (table).

Description:

- Projection eliminates unwanted attributes (columns) but retains only the specified ones.
- Duplicate rows are removed because relations in relational algebra are sets, and sets do not allow duplicates.

Example:

Suppose R is a relation:

$$R = \{(1,'Alice',23),(2,'Bob',30),(3,'Alice',23)\}$$

with attributes **ID**, **Name**, and **Age**.
If you apply $\pi_{\text{Name,Age}}(R)$, the result is:

$$\{('Alice',23),('Bob',30)\}$$

Notice that duplicates are removed.

1.2 SQL Projection (SELECT)

Purpose: Retrieves specific columns from a table, similar to projection in relational algebra.

SQL Syntax:

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

Description:

- SQL's `SELECT` statement works similarly to projection in relational algebra, as it retrieves only specified columns.
- However, SQL does **not** remove duplicates by default (because SQL treats tables as bags (multisets) rather than sets).
- To remove duplicates, you need to use `DISTINCT`.

Example 1 (Without `DISTINCT`):

```
SELECT Name, Age FROM R;
```

Result:

Name	**Age**
Alice	23
Bob	30
Alice	23

Duplicates are retained.

Example 2 (With `DISTINCT`):

```
SELECT DISTINCT Name, Age FROM R;
```

Result:

Name	**Age**
Alice	23
Bob	30

Duplicates are removed, making this behavior identical to relational algebra projection.

1.3 Comparison of Projection in Relational Algebra and SQL

Feature	**Relational Algebra (π)**	**SQL (SELECT)**
Purpose	Select specific columns	Select specific columns
Duplicate Removal	Yes (always)	No (by default)
How to Remove Duplicates?	Always removes duplicates	Use `DISTINCT`

1.4 Key Takeaways

- Projection in relational algebra always removes duplicates, treating relations as sets.
- SQL's `SELECT` does **not** remove duplicates unless explicitly requested using `DISTINCT`.

2 Selection as Applied in Relational Algebra and SQL

2.1 Relational Algebra Selection (σ)

Purpose: Retrieves specific rows (tuples) from a relation based on a condition. It focuses on filtering rows.

Mathematical Notation:

$$\sigma_{\text{condition}}(R)$$

where:

- **condition** is a Boolean expression (e.g., comparisons like $=, >, <, \geq, \neq$ and logical operators like `AND, OR, NOT`) applied to attributes of the relation R.
- R is the relation (table).

Description:

- Selection filters the rows in R based on the given condition.
- It does not change the number of columns; only rows satisfying the condition are included in the output.

Example:

Suppose R is a relation:

$$R = \{(1,'Alice', 23), (2,'Bob', 30), (3,'Charlie', 25)\}$$

with attributes **ID**, **Name**, and **Age**.
If you apply $\sigma_{\text{Age}>25}(R)$, the result is:

$$\{(2,'Bob', 30)\}$$

Only rows meeting the condition (Age > 25) are returned.

2.2 SQL Selection (WHERE Clause)

Purpose: Retrieves specific rows from a table based on a condition, similar to selection in relational algebra.

SQL Syntax:

```
SELECT * FROM TableName WHERE condition;
```

Description:

- SQL's WHERE clause works similarly to selection in relational algebra, as it filters only specific rows based on the given condition.
- The number of columns remains the same; only the rows change.
- Unlike projection, selection in both SQL and relational algebra behaves identically.

Example:

```
SELECT * FROM R WHERE Age > 25;
```

Result:

ID	Name	Age
2	Bob	30

Only rows where Age > 25 are included.

2.3 Relational Algebra Selection with AND, OR, NOT

Operator	Relational Algebra Notation	SQL Equivalent
AND ($\wedge$)	$\sigma_{\text{Age}>20 \wedge \text{Name}='Alice'}(R)$	WHERE Age > 20 AND Name = 'Alice'
OR ($\vee$)	$\sigma_{\text{Age}>25 \vee \text{Name}='Bob'}(R)$	WHERE Age > 25 OR Name = 'Bob'
NOT ($\neg$)	$\sigma_{\neg(\text{Name}='Charlie')}(R)$	WHERE NOT Name = 'Charlie'

Example: Selection with `AND`

```
SELECT * FROM R WHERE Age > 20 AND Name = 'Alice';
```

Relational algebra equivalent:

$$\sigma_{\text{Age}>20 \wedge \text{Name}='Alice'}(R)$$

Result:

ID	Name	Age
1	Alice	23

Example: Selection with `OR`

```
SELECT * FROM R WHERE Age > 25 OR Name = 'Bob';
```

Relational algebra equivalent:

$$\sigma_{\text{Age}>25 \vee \text{Name}='Bob'}(R)$$

Result:

ID	Name	Age
2	Bob	30

Example: Selection with `NOT`

```
SELECT * FROM R WHERE NOT Name = 'Charlie';
```

Relational algebra equivalent:

$$\sigma_{\neg(\text{Name}='Charlie')}(R)$$

Result:

ID	Name	Age
1	Alice	23
2	Bob	30

2.4 Comparison of Selection in Relational Algebra and SQL

Feature	**Relational Algebra (σ)**	**SQL (WHERE)**
Purpose	Select specific rows based on conditions	Select specific rows based on conditions
How It Works?	Uses conditions to filter tuples	Uses WHERE to filter rows
Effect on Columns?	No change in number of columns	No change in number of columns
Duplicate Handling?	No effect (works on sets)	No effect (duplicates are retained unless DISTINCT is used)
Example Notation	$\sigma_{\text{Age}>25}(R)$	`SELECT * FROM R WHERE Age > 25;`

2.5 Key Takeaways

- Selection in relational algebra and SQL both filter rows based on a condition.
- Logical operators (AND, OR, NOT) work the same way in both relational algebra and SQL.
- Relational algebra uses the σ symbol, while SQL uses the `WHERE` clause.
- Unlike projection, SQL selection behaves exactly like relational algebra selection—no extra `DISTINCT` is needed.

3 Combining Projection (π) and Selection (σ)

The combination of projection (π) and selection (σ) is commonly used to both filter rows and reduce the columns in the output.

3.1 Sequence of Operations

1. Apply selection (σ) first to filter rows based on the condition.
2. Apply projection (π) next to retain specific columns from the filtered rows.

Example: Relation R

$$R = \{(1,'Alice', 23), (2,'Bob', 30), (3,'Charlie', 25)\}$$

Attributes: ID, Name, Age.

Query: Select rows where Age > 23 and project only the Name column:

$$\pi_{\text{Name}}(\sigma_{\text{Age}>23}(R))$$

Step 1 (Selection): Filter rows with Age > 23:

$$\{(2,'Bob', 30), (3,'Charlie', 25)\}$$

Step 2 (Projection): Keep only the Name column:

$$\{('Bob'), ('Charlie')\}$$

3.2 Combining Multiple Selection and Projection

You can apply multiple selection conditions before projecting the attributes.

Example:

Select rows where Age $> 20 \wedge$ Name $\neq' Charlie'$, and project Name and Age:

$$\pi_{\text{Name, Age}}(\sigma_{\text{Age}>20 \wedge \text{Name} \neq 'Charlie'}(R))$$

Step 1 (Selection): Filter rows where Age > 20 and Name $\neq' Charlie'$:

$$\{(1,'Alice', 23), (2,'Bob', 30)\}$$

Step 2 (Projection): Retain only the Name and Age columns:

$$\{('Alice', 23), ('Bob', 30)\}$$

3.3 Generalized Workflow for Combining:

1. **Start with Selection:** Focus on filtering the rows using logical conditions.
2. **Apply Projection:** Select the relevant attributes from the filtered rows.
3. **Order of Operations:** Always apply selection first, then projection. This reduces computation cost by working on a smaller subset of data.

Exercise 1: Students

Table: Students

StudentID	Name	Major	GPA	Year
101	*John*	*CS*	3.5	2023
102	*Emma*	*Math*	3.8	2022
103	*Liam*	*CS*	2.9	2024
104	*Olivia*	*Physics*	3.6	2023
105	*Sophia*	*Math*	3.2	2022
106	*Noah*	*CS*	3.7	2023

Query 1: Retrieve students majoring in "CS" with a GPA greater than 3.0.

Relational Algebra:

SQL Equivalent:

Query 2: Retrieve the names of students who are in the year 2023.

Relational Algebra:

SQL Equivalent:

Query 3: Retrieve all details of Math students with a GPA less than 3.5.

Relational Algebra:

SQL Equivalent:

Exercise 2: Products

Table: Products

ProductID	Name	Category	Price	Stock
201	*Laptop*	*Electronics*	1200	50
202	*Chair*	*Furniture*	150	200
203	*Smartphone*	*Electronics*	800	100
204	*Table*	*Furniture*	300	80
205	*Monitor*	*Electronics*	400	70
206	*Desk*	*Furniture*	250	90

Query 1: Retrieve the names of all products in the ”Electronics” category.

Relational Algebra:

SQL Equivalent:

Query 2: Retrieve all details of products priced above 500.

Relational Algebra:

SQL Equivalent:

Query 3: Retrieve the names and stock of products in the ”Furniture” category.

Relational Algebra:

SQL Equivalent:

Exercise 3: Orders

Table: Orders

OrderID	CustomerName	Product	Quantity	Date
301	*Alice*	*Laptop*	1	2025 − 01 − 10
302	*Bob*	*Chair*	4	2025 − 01 − 12
303	*Charlie*	*Table*	2	2025 − 01 − 15
304	*Diana*	*Laptop*	3	2025 − 01 − 17
305	*Emily*	*Smartphone*	5	2025 − 01 − 20
306	*Frank*	*Monitor*	1	2025 − 01 − 22

Query 1: Retrieve the details of orders where the quantity is greater than 2.

Relational Algebra:

SQL Equivalent:

Query 2: Retrieve the names of customers who ordered "Laptop."

Relational Algebra:

SQL Equivalent:

Query 3: Retrieve the names and stock of products in the "Furniture" category.

Relational Algebra:

SQL Equivalent: