Relational algebra is a theoretical framework used in databases to manipulate and query relational data. It provides a set of operations that work on tables (relations) to produce new tables. These operations form the foundation for SQL queries in systems like MySQL.

Relational Algebra Operations and Their MySQL Equivalents

Operation	Symbol	Pronunciation	MySQL Equivalent
Selection	σ	"sigma"	SELECT ... FROM ... WHERE ...
Projection	π	"pi"	SELECT column1, column2, ... FROM ...
Union	∪	"union"	SELECT ... UNION SELECT ...
Intersection	∩	"intersection"	SELECT ... INTERSECT SELECT ... (not in MySQL directly, but achievable)
Difference	−	"minus" or "difference"	SELECT ... EXCEPT SELECT ... (not directly supported in MySQL)
Cartesian Product	×	"cross product"	SELECT ... FROM table1, table2
Natural Join	⋈	"natural join"	SELECT ... FROM table1 NATURAL JOIN table2
Theta Join	⋈θ	"theta join"	SELECT ... FROM table1 JOIN table2 ON condition
Division	÷	"division"	Achieved through nested SELECT queries

Explanation of Each Operation

1. **Selection (σ)**
   - **Symbol**: σ
   - **Pronunciation**: "sigma"
   - **Description**: Filters rows that meet a specified condition.
   - **Example (MySQL)**:

     ```
     SELECT * FROM Songs WHERE genre = 'Pop';
     ```

   - **Relational Algebra**:

     ```
     σ(genre = 'Pop')(Songs)
     ```

2. **Projection (π)**
   - **Symbol**: π
   - **Pronunciation**: "pi"
   - **Description**: Selects specific columns from a table, eliminating duplicates.
   - **Example (MySQL)**:

     ```
     SELECT artist_name, genre FROM Songs;
     ```

   - **Relational Algebra**:

     ```
     π(artist_name, genre)(Songs)
     ```

3. **Union (∪)**
   - **Symbol**: ∪
   - **Pronunciation**: "union"
   - **Description**: Combines results from two tables, removing duplicates.
   - **Example (MySQL)**:

     ```
     SELECT song_name FROM PopSongs
     UNION
     SELECT song_name FROM RockSongs;
     ```

   - **Relational Algebra**:

```
PopSongs ∪ RockSongs
```

4. **Intersection (∩)**
   - **Symbol**: ∩
   - **Pronunciation**: "intersection"
   - **Description**: Returns rows common to both tables (not directly supported in MySQL but achievable using INNER JOIN or EXISTS).
   - **Example (MySQL)**:

     ```
     SELECT song_name FROM PopSongs
     INNER JOIN RockSongs ON PopSongs.song_name = RockSongs.song_name;
     ```

   - **Relational Algebra**:

     ```
     PopSongs ∩ RockSongs
     ```

5. **Difference (−)**
   - **Symbol**: −
   - **Pronunciation**: "minus" or "difference"
   - **Description**: Returns rows in one table that are not present in another.
   - **Example (MySQL)**:

     ```
     SELECT song_name FROM PopSongs
     WHERE song_name NOT IN (SELECT song_name FROM RockSongs);
     ```

   - **Relational Algebra**:

     ```
     PopSongs − RockSongs
     ```

6. **Cartesian Product (×)**
   - **Symbol**: ×
   - **Pronunciation**: "cross product"
   - **Description**: Combines every row of one table with every row of another.
   - **Example (MySQL)**:

     ```
     SELECT * FROM Artists, Genres;
     ```

   - **Relational Algebra**:

     ```
     Artists × Genres
     ```

7. **Natural Join (⋈)**
   - **Symbol**: ⋈
   - **Pronunciation**: "natural join"
   - **Description**: Joins two tables on all columns with the same name.
   - **Example (MySQL)**:

     ```
     SELECT * FROM Songs NATURAL JOIN Artists;
     ```

   - **Relational Algebra**:

     ```
     Songs ⋈ Artists
     ```

8. **Theta Join (⋈θ)**
   - **Symbol**: ⋈θ
   - **Pronunciation**: "theta join"
   - **Description**: Joins tables based on a specified condition.
   - **Example (MySQL)**:

     ```
     SELECT * FROM Songs JOIN Artists ON Songs.artist_id = Artists.artist_id;
     ```

   - **Relational Algebra**:

     ```
     Songs ⋈ (Songs.artist_id = Artists.artist_id) Artists
     ```

9. **Division (÷)**
   - **Symbol**: ÷
   - **Pronunciation**: "division"
   - **Description**: Returns rows in one table that are associated with **all** rows in another table (requires nested queries in MySQL).
   - **Example (MySQL)**:

     ```
     SELECT student_id
     FROM Enrollments
     WHERE course_id IN ('C1', 'C2')
     GROUP BY student_id
     HAVING COUNT(DISTINCT course_id) = 2;
     ```

   - **Relational Algebra**:

     ```
     Enrollments ÷ Courses
     ```