

Relational Calculus & Algebra

Relational Calculus

- Relational Calculus is "declarative". It describes **what** data you want, not the procedure for how to find it.

Relational Tuple Calculus:

- A non-procedural query language based on first-order logic. Where queries are expressed as a set of tuples $\{t \mid P(t)\}$, that satisfy a specified predicate P

Notation:

$$\{t \mid P(t)\}$$

- - t : The tuple variable (represents the row you want to retrieve).
 - \mid : "Such that".
 - $P(t)$: The formula (predicate) that must be true for t to be included in the result

Predicate = A function that returns **true/false** when arguments are given

Proposition = The **expression** obtained when **replacing values** for the arguments of a predicate (a proposition is either true or false)

Basic Operators:

- Let $P(x)$ and $Q(x)$ be two predicates with an argument x

AND:

$$\{x \mid P(x) \wedge Q(x)\}$$

OR:

$$\{x \mid P(x) \vee Q(x)\}$$

NOT:

$$\{x \mid P(x) \sim Q(x)\}$$

Example:

Find all staff members who earn more than £10,000

- We want a tuple (row), s (for staff)
- s must belong to the *Staff* table
- The salary attribute of s must be > 10000

Result: $\{s \mid Staff(s) \wedge s.salary > 10000\}$

Quantifiers (Linking Tables):

- Used when you need to check if data exists in *another* table (similar to a Join in SQL)
 - \exists (**Existential Quantifier**): "There exists..."
 - Use this when you want to say, "There is a row in another table that matches my current row"
 - \forall (**Universal Quantifier**): "For all..."
 - Use this when checking if *every* row in a set meets a condition
-

Relational Algebra

- Relational Algebra is "procedural". It uses operators to construct a new set of data from existing ones, specifying **how** to retrieve data rather than just what to retrieve

Relational Algebra:

- A procedural query language where queries are expressed as a sequence of operations, taking one or more relations as an input, to produce a new relation as an output
-

Unary Operations (One Table)

- These operations work on a single relation at a time

Selection (σ):

- **Syntax:** $\sigma_{condition}(R)$
- **Purpose:** Selects a subset of **rows** (tuples) that satisfy a specific condition
- **SQL Equivalent:** WHERE clause
- **Example:** $\sigma_{salary > 12000}(Staff)$, selects only the staff members earning over £12,000

Projection (π):

- **Syntax:** $\pi_{col1, col2, \dots}(R)$

- **Purpose:** Selects specific **columns** (attributes) and discards the rest
- **SQL Equivalent:** `SELECT DISTINCT col1, col2 ...`
- **Example:** $\sigma_{name,salary}(Staff)$, creates a list containing only names and salaries
- **Note:** Unlike SQL, Relational Algebra explicitly eliminates duplicates in the result

Rename (ρ):

- **Syntax:** $\rho_{NewName}(R)$ or $\rho_{S(A,B,C)}(R)$
- **Purpose:** Gives a temporary name to a relation or its attributes
- **Usage:** Used when you need to join a table with itself (e.g. finding a supervisor from within the same Staff table)

Binary Operations (Two Table)

- These operations combine two relations

Union (\cup):

- **Syntax:** $R \cup S$
- **Purpose:** Combines all tuples from R and S into one set, removing duplicates
- **Constraint:** R and S must be Union Compatible. Meaning:

1. They have the same number of attributes
2. Matching attributes must have the same domain (data type)

Set Difference ($-$):

- **Symbol:** $R - S$
- **Purpose:** Returns tuples that are in R but **NOT** in S
- **Constraint:** Relations must also be Union Compatible
- **Example:** "Find cities with a Property but no Branch" $\rightarrow \pi_{city}(Property) - \pi_{city}(Branch)$

Cartesian Product (\times)

- **Symbol:** $R \times S$
- **Purpose:** Combines **every** tuple in R with **every** tuple in S (concatenation)
- **Size:** If R has 5 rows and S has 3 rows, the result has $5 \times 3 = 15$ rows
- **Note:** This is rarely used alone; it is usually followed by a Selection (σ) to filter meaningful pairs

Derived Operations (Relational Algebra)

- These operations are "derived" because they can be built by combining the basic operations (Selection, Projection, Cartesian Product, Union, Difference). However, they are so common that they have their own symbols

Intersection (\cap):

- **Syntax:** $R \cap S$
 - **Purpose:** Returns a relation containing all tuples that appear in both input relations R and S
 - **Constraint:** Relations must be **Union Compatible** (![Relational Calculus & Algebra > ^a04368](#))
 - **Derivation:** $R \cap S = R - (R - S)$
-

Joins (\bowtie):

- Joins connect data from two tables based on related columns

Theta Join (\bowtie_{θ}):

- **Syntax:** $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$
- **Purpose:** A cartesian Product followed by a Selection condition θ
- **Usage:** General joining with any condition (e.g. $R.a < S.b$)

Equijoin ($=$):

- **Purpose:** A specific type of Theta join where the condition uses only the equality operator ($=$)

Natural Join (\bowtie):

- **Purpose:** An **Equijoin** performed over all attributes with the same name in both relations, followed by the removal of duplicate attribute columns
 - "Join these two tables by matching up every column that has the same name"

Outer Joins:

- **Left Outer Join (\bowtie_{\leftarrow}):** Keeps all rows from the Left table, even if there's no match in the Right (fills with NULL)
 - **Right Outer Join (\bowtie_{\rightarrow}):** Keeps all rows from the Right table
 - **Full Outer Join (\bowtie_{full}):** Keeps all rows from both tables
-

Division (\div):

- **Syntax:** $A \div B$
 - **Definition:** Where B has a subset of attributes of A , and the result consists of the values of attributes in A but not in B that are associated with **every** tuple in B
 - **Requirement:** B must have a subset of attributes from A : $A(X_1, X_2, \dots, Y_1, Y_2, \dots)$ and $B(Y_1, Y_2, \dots)$
-

Summary of Keywords for Question Identification

- "Find ... over X value" → **Selection (σ)**
- "Find the names of..." → **Projection (π)**
- "Find ... who have a loan **OR** an account" → **Union (\cup)**
- "Find ... who have a loan **AND** an account" → **Intersection (\cap)**
- "Find ... who have a loan **BUT NOT** an account" → **Difference ($-$)**
- "Find ... who have a loan **AT** branch X" → **Join (\bowtie) + Selection (σ)**
- "Find ... who have **ALL...**" → **Division (\div)**