ECON485 Introduction to Database Systems

Lecture 05 – Relational Algebra

Relational Algebra

Relational Algebra was introduced by Codd in 1970.

- All relational databases and their query languages, such as SQL are based on this algebra.
- **Simple algebra** is defined with operators (+,-, etc) which work on numbers
- **Set theory** defines operators (union, intersection, difference, and universal set concept) that work on sets.
- **Relational algebra extends set theory concepts**, with operators which work on tables of data.

Relational Algebra

Three types of operators

- Unary operators – Select, Project, and Rename
- Operators from set theory (adapted into tables) – Union, Intersection, Difference, Cartesian Product
- Binary operators – Join, and Division.

Select

Purpose

Select a subset of tuples (rows) that satisfy a given condition.

Notation

$\sigma_p(r)$ where σ is the predicate, r stands for relation which is the name of the table, and p is prepositional logic.

Example

$\sigma_{\text{Type=Winter and OwnerID=2}}(\text{TyreTable})$

	TyreTable	
TyreID	**Type**	**OwnerID**
1	Winter	1
5	Summer	1
6	Summer	1
9	Winter	2
10	Summer	2

TyreID	Type	OwnerID
9	Winter	2

Project

Purpose

Eliminates all attributes (columns) of the input relation but those mentioned in the projection list.

Notation

$\Pi_p(r)$ where Π is the predicate, r stands for relation which is the name of the table, and p is the projection list

Example

Π_{Type} (TyreTable)

	Tyre Table			**Result**
TyreID	**Type**	**Owner ID**		**Type**
1	Winter	1		Winter
2	Winter	1		Winter
5	Summer	2		Summer
6	Summer	2		Summer

Rename

Purpose

Renames attributes (columns) of the input relation as those mentioned.

Notation

$\rho_{(a/b)}R$ where ρ is the predicate, R stands for relation which is the name of the table, and (a/b) denotes "rename a as b"

Example

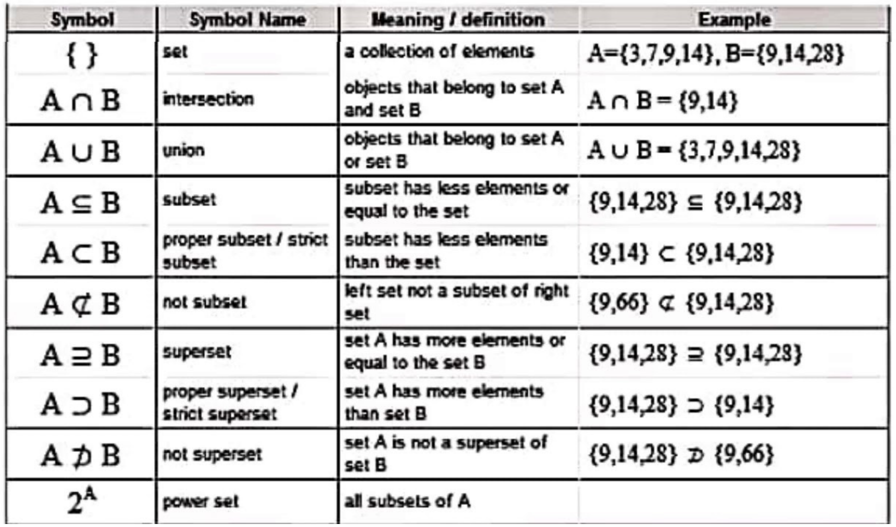
ρ (Type/TyreType) (TyreTable)

Tyre Table				Result		
TyreID	**Type**	**OwnerID**		**TyreID**	**TyreType**	**OwnerID**
1	Winter	1		1	Winter	1
2	Winter	1		2	Winter	1
5	Summer	2		5	Summer	2
6	Summer	2		6	Summer	2

Set Theory Operators

Let’s first recall set theory.

- These operators are adapted into tables which are viewed at one time as either
  - Composed of tuples (rows)
  - Composed of attributes (columns)



Symbol	Symbol Name	Meaning / definition	Example
{ }	set	a collection of elements	A={3,7,9,14}, B={9,14,28}
$A \cap B$	intersection	objects that belong to set A and set B	$A \cap B = \{9,14\}$
$A \cup B$	union	objects that belong to set A or set B	$A \cup B = \{3,7,9,14,28\}$
$A \subseteq B$	subset	subset has less elements or equal to the set	$\{9,14,28\} \subseteq \{9,14,28\}$
$A \subset B$	proper subset / strict subset	subset has less elements than the set	$\{9,14\} \subset \{9,14,28\}$
$A \not\subset B$	not subset	left set not a subset of right set	$\{9,66\} \not\subset \{9,14,28\}$
$A \supseteq B$	superset	set A has more elements or equal to the set B	$\{9,14,28\} \supseteq \{9,14,28\}$
$A \supset B$	proper superset / strict superset	set A has more elements than set B	$\{9,14,28\} \supset \{9,14\}$
$A \not\supset B$	not superset	set A is not a superset of set B	$\{9,14,28\} \not\supset \{9,66\}$
2^A	power set	all subsets of A	

Set Theory Operators

Unions, Intersection, and Differences (in tables)

- A union is symbolized by ∪ symbol. A **∪** B. It includes all tuples (rows) that are in tables A **or** in B. It also eliminates duplicate tuples.
- An intersection is defined by the symbol ∩. A ∩ B. It defines a relation consisting of a set of all tuples (rows) that are in both A and B.
- A difference is defined by the symbol **–**. The result of A **–** B, is a relation which includes all tuples (rows) that are in A but not in B.

For these three operators to work on tables, tables should have the same set of attributes (column names).

- This is called **union-compatibility**.

Set Theory Operators

Cartesian Product (X)

The Cartesian product of two tables is defined different than a cartesian product of two sets.

For the Cartesian product to be defined, the two relations involved must have disjoint headers — that is, they must **not** have a common attribute name.

The Cartesian product of a table of n-tuples (rows with n columns) with a set of m-tuples (rows with m columns) yields a set of "flattened" (n + m)-tuples (rows of n+m columns).

Therefore, Cartesian Product is an operation used to merge columns from two relations (mostly tables).

Generally, a cartesian product is never a meaningful operation when it performs alone. However, **it becomes meaningful when it is followed by other operations**.

Example. Cartesian product of two tables produces a relation, then we use the selection operator on this relation. This means joining all columns of two tables and then filtering them based on the criteria of the selection operator.

Binary Operators

Joins

- A cartesian product followed by a selection is the core idea of a join.
- Join operation denoted by $\bowtie$. Join operation allows joining variously related tuples (rows) from different relations (tables).
- In an inner join, only those tuples that satisfy the matching criteria are included, while the rest are excluded.
  - When the matching criteria is arbitrary, we call an inner join as a theta join, denoted $A \bowtie_{\theta} B$.
  - When the matching criteria is based on an equivalence relationship, we call an inner join as an equi join, denoted as $A \bowtie_{EQUI} B$
  - Natural join $\bowtie$ as a type of inner join can only be performed if there is a common attribute (column) between the relations. The name and type of the attribute must be same.
- In an outer join, along with tuples that satisfy the matching criteria, we also include some or all tuples that do not match the criteria.

Binary Operators

Joins

- In an outer join, along with tuples that satisfy the matching criteria, we also include some or all tuples that do not match the criteria.
  - In the left outer join, operation allows keeping all tuple in the left relation. However, if there is no matching tuple is found in right relation, then the attributes of right relation in the join result are filled with null values.
  - In the right outer join, operation allows keeping all tuple in the right relation. However, if there is no matching tuple is found in the left relation, then the attributes of the left relation in the join result are filled with null values.
  - In a full outer join, all tuples from both relations are included in the result, irrespective of the matching condition.
- There are also concepts like semi-joins, anti-joins, and divisions which we will not cove.

How much does Relational Algebra match SQL?

Relational databases and SQL have advanced much since 1970.

- Today, relational databases view tables as **bags implemented as multisets**. This is a more versatile model then that of a mathematical set which would not allow for repeated entries (ie. rows containing the exact same content).
- Therefore tables are **not exactly** relations.

However, understanding relational algebra improves understanding of basic SQL operations.

How much does Relational Algebra match SQL?

For example, a very popular (and recommended) SQL tutorial exists at – https://www.w3schools.com/sql/

Starts with **SELECT** on a single table which is the SQL implementation of relational algebra **projection** operator (not selection, as the simplest SELECT query just chooses columns).

Then covers **SELECT DISTINCT** on a single table, which resolves the set vs multiset difference we discussed.

Then covers the **WHERE** clause, used with a **SELECT** query, which is a projection operator used with a selection operator (projection choosing columns, selection filtering rows)

Then covers useful concepts like logical operators within the **WHERE** clause, and **ORDER**ing the results **BY** the values of a selected column.

How much does Relational Algebra match SQL?

For example, a very popular (and recommended) SQL tutorial exists at – https://www.w3schools.com/sql/

- INSERT, UPDATE, and DELETE operations are covered before joins.
  - Note that these are required to fill the tables with data and maintain the tables. Relational algebra does not concern itself much with how the tables are filled and maintained.
- The first join example using **INNER JOIN** is an equi join as SQL does not have any syntactic difference between types of inner joins.
- Then covers outer joins as separate types of joins **LEFT JOIN**, **RIGHT JOIN** and **FULL JOIN**.

Next Session

We will continue with examples in SQL.