# DBMS - Joins

We understand the benefits of taking a Cartesian product of two relations, which gives us all the possible tuples that are paired together. But it might not be feasible for us in certain cases to take a Cartesian product where we encounter huge relations with thousands of tuples having a considerable large number of attributes.

**Join** is a combination of a Cartesian product followed by a selection process. A Join operation pairs two tuples from different relations, if and only if a given join condition is satisfied.

We will briefly describe various join types in the following sections.

## Theta (θ) Join

Theta join combines tuples from different relations provided they satisfy the theta condition. The join condition is denoted by the symbol **θ**.

### Notation

R1 ⋈θ R2

R1 and R2 are relations having attributes (A1, A2, .., An) and (B1, B2,.. ,Bn) such that the attributes don’t have anything in common, that is R1 ∩ R2 = Φ.

Theta join can use all kinds of comparison operators.

## Natural Join (⋈)

Natural join does not use any comparison operator. It does not concatenate the way a Cartesian product does. We can perform a Natural Join only if there is at least one common attribute that exists between two relations. In addition, the attributes must have the same name and domain.

Natural join acts on those matching attributes where the values of attributes in both the relations are same.

**LEFT JOIN**

The LEFT JOIN keyword returns all rows from the left table (table1), with the matching rows in the right table (table2). The result is NULL in the right side when there is no match.

**LEFT JOIN Syntax**

SELECT *column\_name(s)*  
FROM *table1*  
LEFT JOIN *table2*  
ON *table1.column\_name*=*table2.column\_name*;

**RIGHT JOIN**

The RIGHT JOIN keyword returns all rows from the right table (table2), with the matching rows in the left table (table1). The result is NULL in the left side when there is no match.

**RIGHT JOIN Syntax**

SELECT *column\_name(s)*  
FROM *table1*  
RIGHT JOIN *table2*  
ON *table1.column\_name*=*table2.column\_name*;

**Inner joint**

The most frequently used and important of the joins is the **INNER JOIN**. They are also referred to as an EQUIJOIN.

The INNER JOIN creates a new result table by combining column values of two tables (table1 and table2) based upon the join-predicate. The query compares each row of table1 with each row of table2 to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row.

## Syntax:

The basic syntax of **INNER JOIN** is as follows:

SELECT table1.column1, table2.column2...

FROM table1

INNER JOIN table2

ON table1.common\_field = table2.common\_field;

**ADDITIONAL BASIC OPERATIONS**

* **Assignment operation**
* **Rename Operation**
* **The Intersect Operation**
* **Division operation**
* **Natural join operation**

**ADDITIONAL BASIC OPERATIONS**

There are number of additional basic operations that are supported in SQL.

**The Assignment Operation**

It is convenient at times to write a relational-algebra expression by assigning parts of it to temporary relation variables. The assignmentoperation, denoted by ←, works like assignment in a programming language. To illustrate this operation, consider the definition of the natural-join operation. We could write *r \_ s* as:

*temp*1 ← *R* × *S*

*temp*2 ← \_*r.A*1 =*s.A*1 ∧*r.A*2 =*s.A*2 ∧*...*∧*r.An* =*s.An* (*temp*1)

*result* = *\_R* ∪ *S* (*temp*2)

The evaluation of an assignment does not result in any relation being displayed to the user. Rather, the result of the expression to the right of the ← is assigned to the relation variable on the left of the ←. This relation variable may be used in subsequent expressions.

With the assignment operation, a query can be written as a sequential program consisting of a series of assignments followed by an expression whose value is displayed as the result of the query. For relational-algebra queries, assignment must always be made to a temporary relation variable. Assignments to permanent relations constitute a database modification. Note that the assignment operation does not provide any additional power to the algebra. It is, however, a convenient way to express complex queries.

**Rename operation (ρ)**

The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter rho *ρ*.

**Notation** − *ρ* x (E), where the result of expression **E** is saved with name of **x**.

Consider the query:

**select** *name*, *course id*

**from** *instructor*, *teaches*

**where** *instructor*.*ID*= *teaches*.*ID*;

The result of this query is a relation with the following attributes:

*name*, *course id*

The names of the attributes in the result are derived from the names of the attributes in the relations in the **from** clause.

We cannot, however, always derive names in this way, for several reasons:

First, two relations in the **from** clause may have attributes with the same name, in which case an attribute name is duplicated in the result. Second, if we used an arithmetic expression in the **select** clause, the resultant attribute does not have a name. Third, even if an attribute name can be derived from the base relations as in the preceding example, we may want to change the attribute name in the result. Hence, SQL provides a way of renaming the attributes of a result relation.

It uses the **as** clause, taking the form:

*old-name* **as** *new-name*

The as clause can appear in both the select and from clauses.

**The Intersect Operation**

The **intersect** operation acts as intersection of a set: it outputs a relation with thew tuples common to both the relation. It automatically eliminates duplicates. For example, if it were the case that 4 sections of ECE-101 were taught in the Fall 2009 semester and 2 sections of ECE-101 were taught in the Spring 2010 semester, then there would be only 1 tuple with ECE-101 in the result.

If we want to retain all duplicates, we must write intersect all in place of intersect.

Example: To find the set of all courses taught in the Fall 2009 as well as in Spring 2010 we write:

(**select** *course id*

**from** *section*

**where** *semester* = ’Fall’ **and** *year*= 2009)

**intersect**

(**select** *course id*

**from** *section*

**where** *semester* = ’Spring’ **and** *year*= 2010);

**Division operation**

The division operator of relational algebra, “÷”, is defined as follows. Let *r* (*R*) and *s*(*S*) be relations, and let *S* ⊆ *R*; that is, every attribute of schema *S* is also in schema *R*. Then *r* ÷ *s* is a relation on schema *R* − *S* (that is, on the schema containing all attributes of schema *R* that are not in schema *S*). A tuple *t* is in *r* ÷ *s* if and only if both of two conditions hold:

* *t* is in *\_R*−*S*(*r* )
* For every tuple *ts* in *s*, there is a tuple *tr* in *r* satisfying both of the following:

1. *tr* [*S*] = *ts* [*S*]
2. *tr* [*R* − *S*] = *t*

**Natural Join Operation**

The **natural join** operation operates on two relations and produces a relation as the result. Unlike the Cartesian product of two relations, which concatenates each tuple of the first relation nwith every tuple of the second, natural join considers only those pairs of tuples with the same value on those attributes that appear in the schemas of both relations.