ST. XAVIER’S COLLEGE

**(Affiliated to Tribhuvan University)**

Maitighar, Kathmandu



DATABASE MANAGEMENT SYSYTEM

Lab Assignment #6

**Submitted by:**

Pankaj Niroula  
013BSCCSIT025

**Submitted to:**

|  |  |
| --- | --- |
| Er. Sanjay Kumar Yadav  Lecturer  Department of Computer Science |  |

Date of Submission: September 13, 2015

# JOIN

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.

Combines attributes of two relations into one.

**R3 = join(R1,D1,R2,D2)**

Given a domain from each relation, *join* considers all possible pairs of tuples from the two relations, and if their values for the chosen domains are equal, it adds a tuple to the result containing all the attributes of both tuples (discarding the duplicate domain D2).

## THETA JOIN

In theta join we apply the condition on input relation(s) and then only those selected

rows are used in the cross product to be merged and included in the output. It means

that in normal cross product all the rows of one relation are mapped/merged with all

the rows of second relation, but here only selected rows of a relation are made cross

product with second relation. It is denoted as under: - R x S

If R and S are two relations then is the condition, which is applied for select

operation on one relation and then only selected rows are cross product with all the

rows of second relation. For Example there are two relations of FACULTY and

COURSE, now we will first apply select operation on the FACULTY relation for

selection certain specific rows then these rows will have across product with

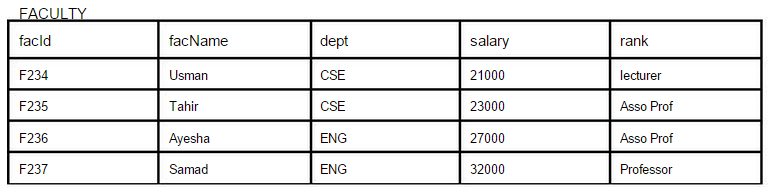
COURSE relation, so this is the difference in between cross product and theta join.

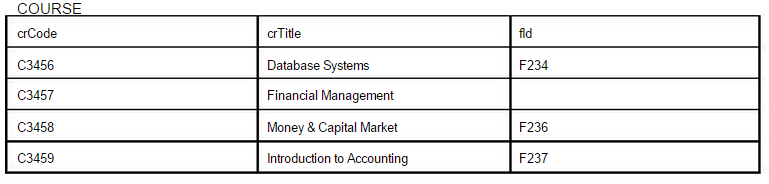
We will now see first both the relation their different attributes and then finally the

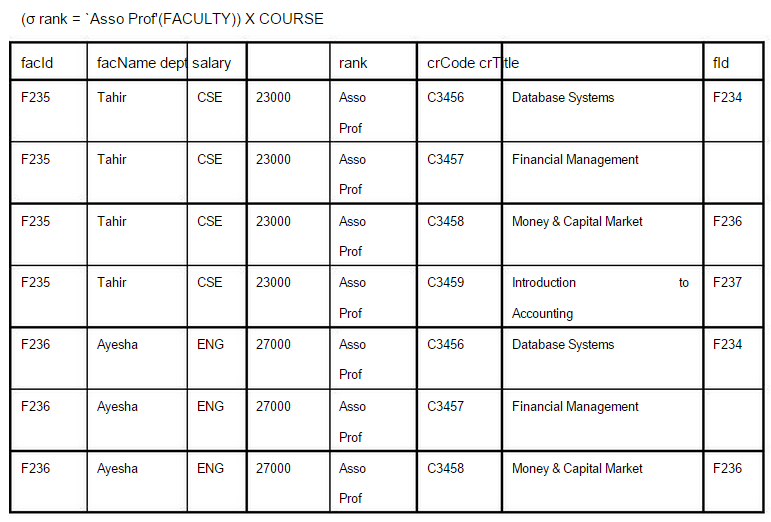
cross product after carrying out select operation on relation.

From this example the difference in between cross product and theta join becomes

clear.







# Natural Join

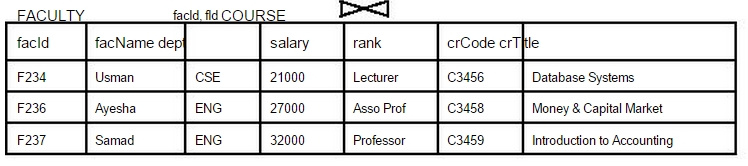
This is the most common and general form of join. If we simply say join, it means the

natural join. It is same as equi­join but the difference is that in natural join, the

common attribute appears only once. Now, it does not matter which common attribute

should be part of the output relation as the values in both are same. For Example if we

take the natural join of faculty and course the output would be as under:



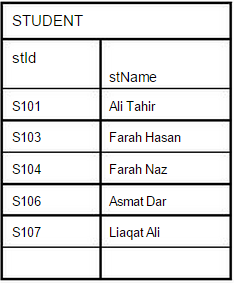
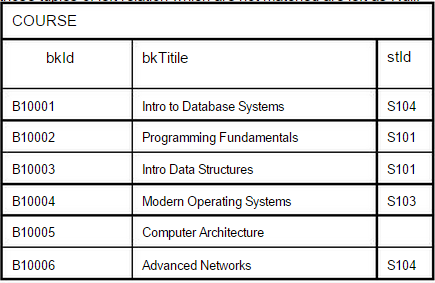
## Right Join

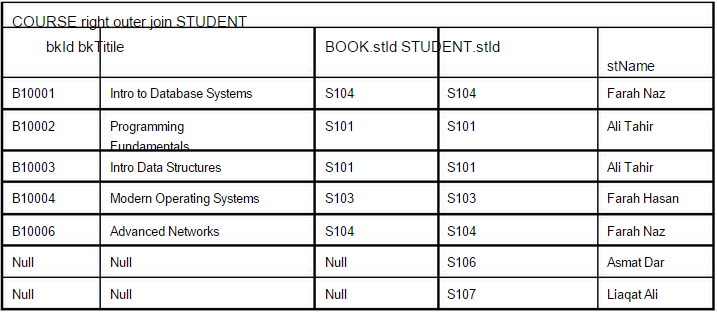
In right outer join all the tuples of right relation remain part of the output relation,

whereas on the left side the tuples, which do not match with the right relation, are left

as null. It means that right outer join will always have all the tuples of right relation

and those tuples of left relation which are not matched are left as Null.

****

****

## Left Join

In left outer join all the tuples of left relation remain part of the output. The tuples that

have a matching tuple in the second relation do have the corresponding tuple from the

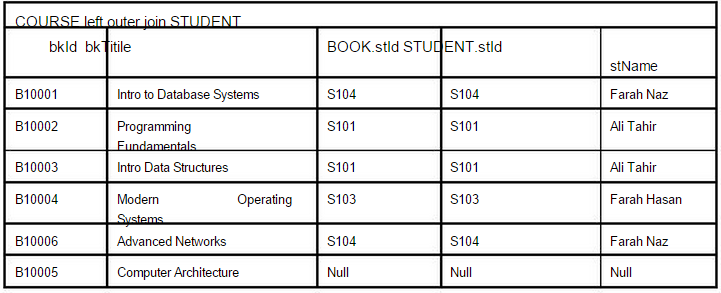
second relation. However, for the tuples of the left relation, which do not have a

matching record in the right tuple have Null values against the attributes of the right

relation. The example is given in figure 5 below. It can be described in another way.

Left outer join is the equi-join plus the non matching rows of the left side relation

having Null against the attributes of right side relation.



## Inner Join

Only those rows from two tables are joined that have same value in the common

attribute. For example, if we have two tables R and S with schemes

R (a, b, c, d) and S (f, r, h, a), then we have `a' as common attribute between these

twit tables. The inner join between these two tables can be performed on the basis of

`a' which is the common attribute between the two. The common attributes are not

required to have the same name in both tables, however, they must have the same

domain in both tables. The attributes in both tables are generally tied in a primary-

foreign key relationship but that also is not required. Consider the following two

tables:

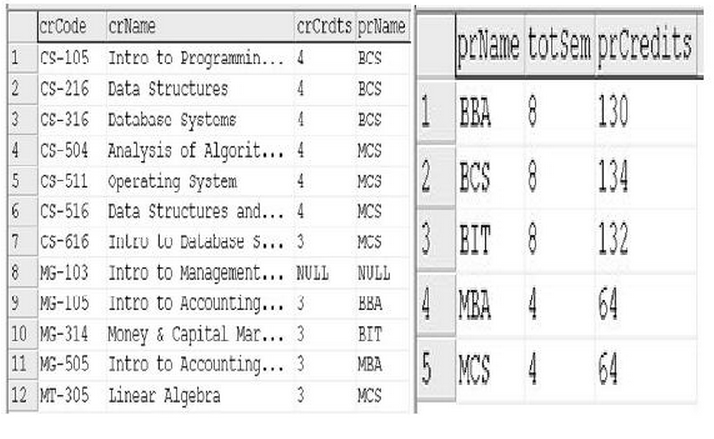


Fig. : COURSE and PROGRAM tables with common attribute prName

The figure shows two tables, COURSE and PROGRAM. The COURSE.prName and

PROGRAM. prName are the common attributes between the two tables; incidentally

the attributes have the same names and definitely the same domains. If we apply

inner join on these tables, the rows from both tables will be merged based on the

values of common attribute, that is, the prName. Like, row one of COURSE has the

value `BCS' in attribute prName. On the other hand, row number 2 in PROGRAM

table has the value `BCS'. So these two rows will merge and form one row of the

resultant table of the inner join operation. As has been said before, the participating

tables of inner join are generally tied in a primary-foreign key link, so the common

attribute is PK in one of the tables. It means the table in which the common attribute

is FK, the rows from this table will not be merged with more that one row from the

other table. Like in the above example, each row from COURSE table will find

exactly one match in PROGRAM table, since the prName is the PK in PROGRAM

table.

The inner join can be implemented using different techniques. One possibility is that

we may find `inner join' operation as such, like:

SELECT \* FROM course INNER JOIN program ON

course.prName = program.prName

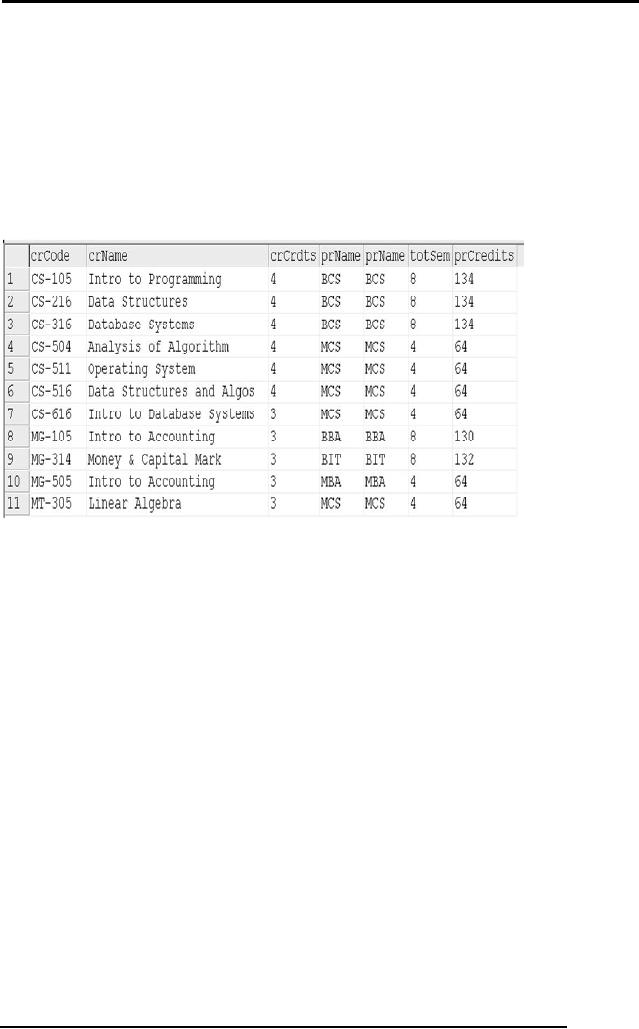
or

·

Select \* FROM Course c INNER JOIN program p ON

c.prName = p.prName

The output after applying inner join on tables of figure 1 will be as follows:



# Rename Operation

The rename operator returns an existing relation under a new name. ρA(B) is the relation B with its name changed to A. For example, find the employees in the same Department as employee 3.

ρemp2.surname,emp2.forenames (

σemployee.empno = 3 ^ employee.depno = emp2.depno (

employee × (ρemp2employee)

)

)

# Assignment Operator

You’ve probably gotten a sense, particularly with division that relational algebra feels

a lot like programming: there are many steps to some expressions, with intermediate

or temporary relations along the way.

For this very reason, we have the assignment operation, which works a lot like assignments

in a programming language. It is notated with the left-pointing arrow ←:

variable ← E

where E is any relational algebra expression.

The assignment operation is more of a notational convenience rather than a real relational

operation — it helps human beings with writing out complex relational expressions

in steps so that they can be more easily understood.

Revisiting the breakdown of the division operation, we can use assignment to rewrite

(1) this way:

temp1 ← ΠR−S(r)

temp2 ← ΠR−S((temp1 × s) − ΠR−S,S(r))

r ÷ s = temp1 − temp2

# 5. Division Operation

The division operation is a binary operation, notated as ÷, on relations r(R) and s(S) such that S ⊆ R. Intuitively, division is a “for all” query — it returns the tuples in r that “match” all of the tuples in s.

Formally, r ÷ s is a relation with schema R − S whose tuples t satisfy both of these conditions:

1. t ∈ ΠR−S(r)

2. ∀ ts ∈ s, ∃ tr ∈ r that satisfies both of:

(a) tr[S] = ts[S]

(b) tr[R − S] = t

Division can also be defined in terms of the relational algebra. Again, we start with

r(R) and s(S) with S ⊆ R:

r ÷ s = ΠR−S(r) − ΠR−S((ΠR−S(r) × s) − ΠR−S,S(r)) (1)

This is quite an eyeful, so we break it down like this:

– First, we take ΠR−S(r). This defines the relation for the first condition of the formal definition: t ∈ ΠR−S(r), or the tuples in r with their s-shared attributes removed.

– Now, from this, we perform set-difference using:ΠR−S((ΠR−S(r) × s) − ΠR−S,S(r))

– So these tuples are removed from ΠR−S(r). What are these tuples exactly? We start with ΠR−S(r) × s: it takes all of the tuples in r and removes any attributes that r shares with s, then pairs those tuples with every tuple in s.

– ΠR−S,S(r) just rearranges the attributes in r so that the attributes that are in r alone are listed first, followed by the attributes that r shares with s. This ensures that the relation of the first difference term, ΠR−S(r) × s, is compatible with ΠR−S,S(r) (same arity, corresponding domains).

– So, the set-difference operation can take place, and the result would be the r-to-s tuple matchups that are not in r.

– If we then remove the s attributes from that relation, resulting in ΠR−S((ΠR−S(r)×s)−ΠR−S,S(r)), we now have the tuples in r for which at least one tuple in s does not match. Since r ÷ s is about tuples in r that have a corresponding match for all tuples in s, then these are precisely the tuples that we don’t want, and so we subtract them from ΠR−S(r).

# 6. Additional Operation

## 6.1 Set Intersection Operation

## The set-intersection operation is a binary operation on relations r and s that is denoted by the traditional intersection symbol, ∩. r ∩ s results in all tuples t such that (t ∈ r) ∧ (t ∈ s).

## Set-intersection is defined in terms of set-difference:s

## r ∩ s = r − (r − s)

## • Thus, set-intersection must follow the same compatibility rules as set-difference: same arity, corresponding domains.

## 6.2 Natural Join Operation

A NATURAL JOIN is a [JOIN operation](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefsqlj29840.html#rrefsqlj29840) that creates an implicit join clause for you based on the common columns in the two tables being joined. Common columns are columns that have the same name in both tables.

A NATURAL JOIN can be an INNER join, a LEFT OUTER join, or a RIGHT OUTER join. The default is INNER join.

If the SELECT statement in which the NATURAL JOIN operation appears has an asterisk (\*) in the select list, the asterisk will be expanded to the following list of columns (in this order):

* All the common columns
* Every column in the first (left) table that is not a common column
* Every column in the second (right) table that is not a common column

An asterisk qualified by a table name (for example, COUNTRIES.\*) will be expanded to every column of that table that is not a common column.

If a common column is referenced without being qualified by a table name, the column reference points to the column in the first (left) table if the join is an INNER JOIN or a LEFT OUTER JOIN. If it is a RIGHT OUTER JOIN, unqualified references to a common column point to the column in the second (right) table.

## Syntax

[TableExpression](https://docs.oracle.com/javadb/10.8.3.0/ref/rreftableexpression.html#rreftableexpression) **NATURAL [ { LEFT | RIGHT } [ OUTER ] | INNER ] JOIN {** [TableViewOrFunctionExpression](https://docs.oracle.com/javadb/10.8.3.0/ref/rrefsqlj33215.html#rrefsqlj33215) **| (** [*TableExpression*](https://docs.oracle.com/javadb/10.8.3.0/ref/rreftableexpression.html#rreftableexpression) **) }**

## Examples

If the tables COUNTRIES and CITIES have two common columns named COUNTRY and COUNTRY\_ISO\_CODE, the following two SELECT statements are equivalent:

**SELECT \* FROM COUNTRIES NATURAL JOIN CITIES**

**SELECT \* FROM COUNTRIES JOIN CITIES**

**USING (COUNTRY, COUNTRY\_ISO\_CODE)**

The following example is similar to the one above, but it also preserves unmatched rows from the first (left) table:

**SELECT \* FROM COUNTRIES NATURAL LEFT JOIN CITIES**