ST. XAVIER’S COLLEGE

**Maitighar, Kathmandu**

**(Affiliated to Tribhuvan University)**



**Database Management System**

**Theory Assignment #6**

**Submitted By**

Sneha Prasai

013BSCCSIT040

**Submitted To**

Er. Sanjay Kumar Yadav

Lecturer

Department of Computer Science

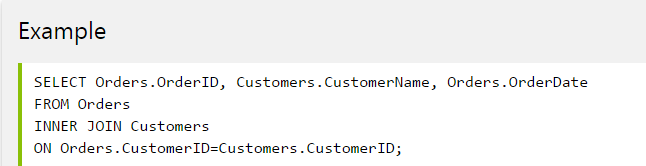
St. Xavier’s College

Maitighar, Kathmandu

# JOIN

A SQL **join** clause combines records from two or more tables in a relational database. It creates a set that can be saved as a table or used as it is. A **JOIN** is a means for combining fields from two tables (or more) by using values common to each.

The most common type of join is: **SQL INNER JOIN (simple join)**. An SQL INNER JOIN return all rows from multiple tables where the join condition is met.



Types of join:

1. Theta Join
2. Equi Join
3. Semi Join
4. Natural Join
5. Outer Join

# Theta Join:

A *theta-join* is any Cartesian product that's filtered by a condition which compares values from both Tables. That is, the general theta-join form is:

<Table\_1.Column> relator <Table\_2.Column>

where the relator is almost always "=", as in this example:

Sellers.seller\_name = Sales.seller\_name

This special case of theta-join — where the relation is equality — is called an *equijoin*.

# 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.

**Left Join**

This join returns all the rows from the left table in conjunction with the matching rows from the right table. If there are no columns matching in the right table, it returns NULL values.

All the tuples from the Left relation, R, are included in the resulting relation. If there are tuples in R without any matching tuple in the Right relation S, then the S-attributes of the resulting relation are made NULL.

**Eg:**

SELECT user.name, course.name

FROM `user`

LEFT JOIN `course` on user.course = course.id;

**Right Join**

This join returns all the rows from the right table in conjunction with the matching rows from the left table. If there are no columns matching in the left table, it returns NULL values.

All the tuples from the Right relation, S, are included in the resulting relation. If there are tuples in S without any matching tuple in R, then the R-attributes of resulting relation are made NULL.

**Eg:**

SELECT user.name, course.name

FROM `user`

RIGHT JOIN `course` on user.course = course.id;

**Inner Join**

In this kind of a JOIN, we get all records that match the condition in both the tables, and records in both the tables that do not match are not reported.

In other words, INNER JOIN is based on the single fact that: ONLY the matching entries in BOTH the tables SHOULD be listed.

Note that a JOIN without any other JOIN keywords (like INNER, OUTER, LEFT, etc) is an INNER JOIN.

**Eg:**

SELECT cFirstName, cLastName, orderDate

FROM customers INNER JOIN orders

USING (custID);

# Rename operation:

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**.

**ASSIGNMENT OPERATOR**

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 this way:

temp1 ← ΠR−S(r)

temp2 ← ΠR−S((temp1 × s) − ΠR−S,S(r))

r ÷ s = temp1 − temp2

**THE DIVISION OPERATION**

Let r(R) and s(S) be relations **r ÷ s: -** the result consists of the restrictions of tuples in r to the attribute names unique to R, i.e. in the Header of r but not in the Header of s, for which it holds that all their combinations with tuples in s are present in r.  
  
Example:

|  |  |  |
| --- | --- | --- |
| Relation or table "r":- | Relation or table "s":- | Therefore, r ÷ s |
| Code: | Code: | Code: |
| | A | B |  \_\_\_\_\_\_\_\_\_  | a | 1 |  | b | 2 |  | a | 2 |  | p | 3 |  | p | 4 | | | B |  \_\_\_\_  | 2 |  | 3 |  \_\_\_\_ | | A |  \_\_\_\_  | b |  | a |  | p |  \_\_\_\_ |

**ADDITIONAL OPERATION**

• “Additional operations” refer to relational algebra operations that can be expressed in terms of the fundamentals — select, project, union, set-difference, cartesian-product, and rename.

• The compositions of these operations are so lengthy, yet so common, that we define new operations for them, based on the fundamentals. Kind of a mathematical “syntactic sugar.”

**SET-INTERSECTION**

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). 1

Set-intersection is defined in terms of set-difference: r ∩ s = r − (r − s)

# Natural join operation:

The natural-join operation is a binary operation on relations r(R) and s(S) that is denoted by the symbol ./. Intuitively, a natural-join “matches” the tuples of r with the tuples of s based on attributes that are both in r and s.

• If we take the relational schemas R and S as sets of attributes, then we can define “attributes that are in both r and s” as R ∩ S = {A1, A2, . . . , An}. With that, we can formally define r ./ s as:

r ./ s = ΠR ∪ S(σr.A1 = s.A1 ∧ r.A2 = s.A2 ∧ ... ∧ r.An = s.An (r × s))

• Note that R ∪ S removes duplicate attribute names, so r ./ s will only have one attribute Ak ∀Ak ∈ R ∩ S.

• Natural join is associative — that is, (a ./ b) ./ c = a ./ (b ./ c).

• When r and s do not have any common attributes — i.e., R ∩ S = ∅ — then r ./ s = r × s.

**REFERENCE :**

<http://cisnet.baruch.cuny.edu/holowczak/classes/3400/relationalalgebra/>

<http://myweb.lmu.edu/dondi/share/db/relational3.pdf>