ST. XAVIER’S COLLEGE

**(Affiliated to Tribhuvan University)**

**Maitighar, Kathmandu**

****

**Database Management System**

**Assignment #6**

**SUBMITTED BY:**

**Hemanchal Joshi**

**013BSCCSIT021**

**SUBMITTED TO:**

|  |  |
| --- | --- |
| **Er. Sanjay Kr. Yadav**  **Lecturer** |  |
| **Department of Computer Science** | |

Submission Date September 10, 2015

**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. Join is a special form of cross product of two tables. SQL Join is used to fetch data from two or more tables, which is joined to appear as single set of data. SQL Join is used for combining column from two or more tables by using values common to both tables. Join Keyword is used in SQL queries for joining two or more tables. Minimum required condition for joining table, is (n-1) where n, is number of tables. A table can also join to itself known as, Self Join. It is a binary operation that allows combining certain selections and a Cartesian product into one operation. The join operation forms a Cartesian product of its two arguments, performs a selection forcing equality on those attributes that appear in both relation schemas, and finally removes duplicate attributes. Following are the different types of joins:

**Theta join**

**Equi Join**

**Semi Join**

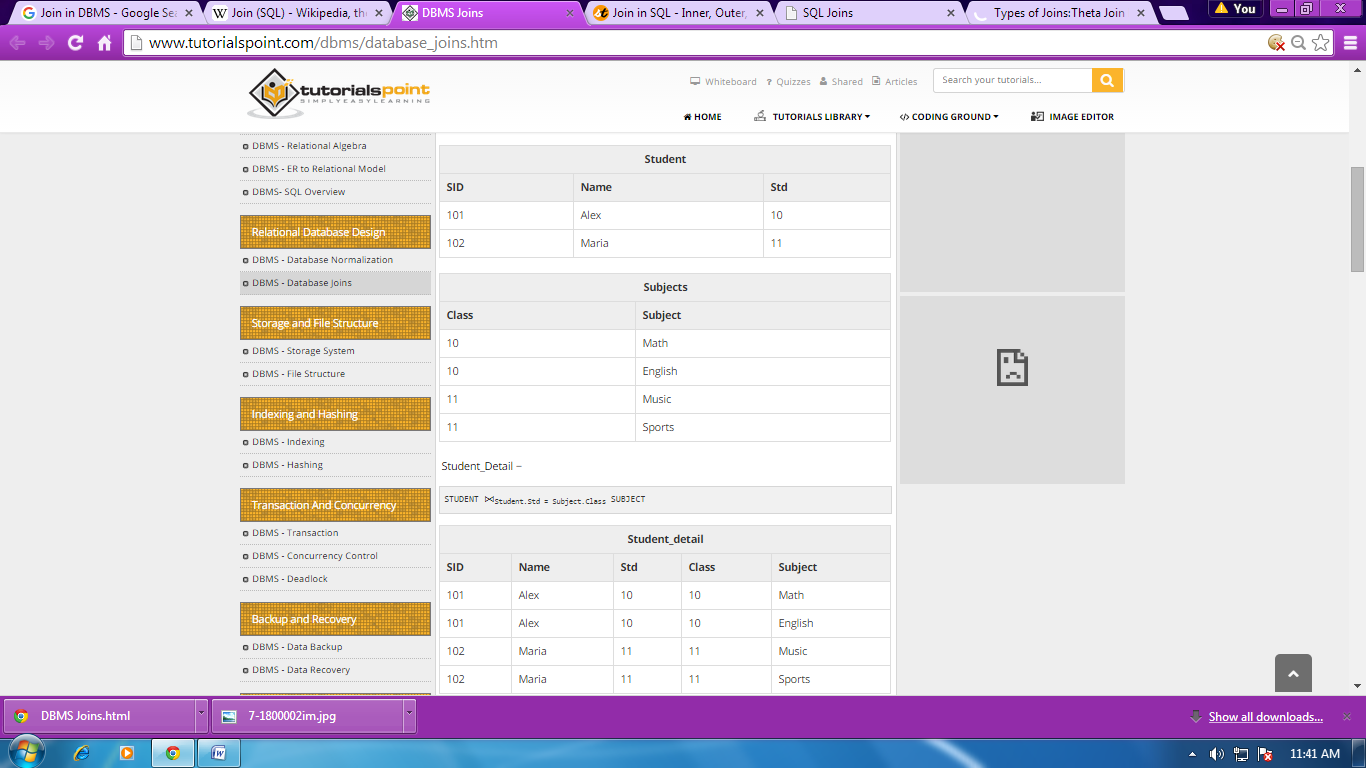
**Natural Join**

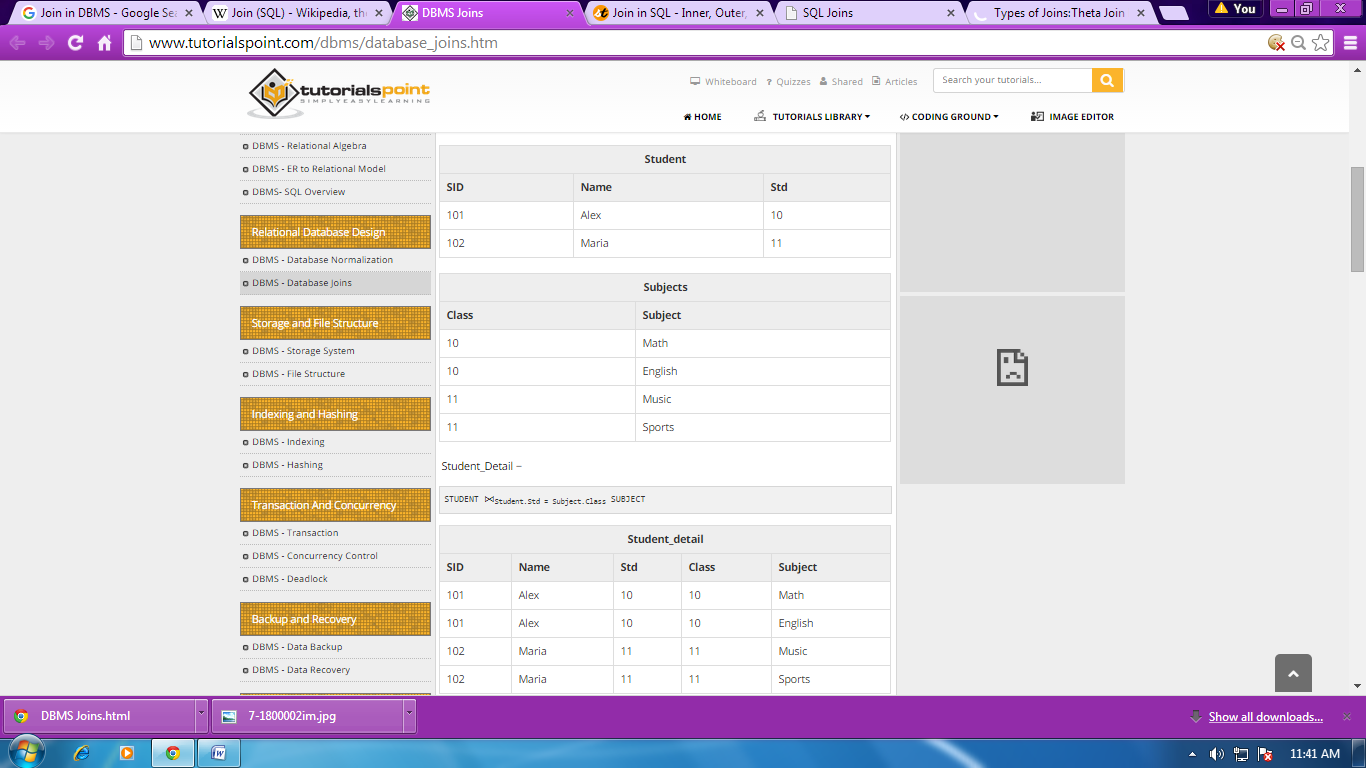
**Outer Joins**

**THETA JOIN**

In theta join we apply the condition on input relation 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: - R1 ⋈θ R2

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:



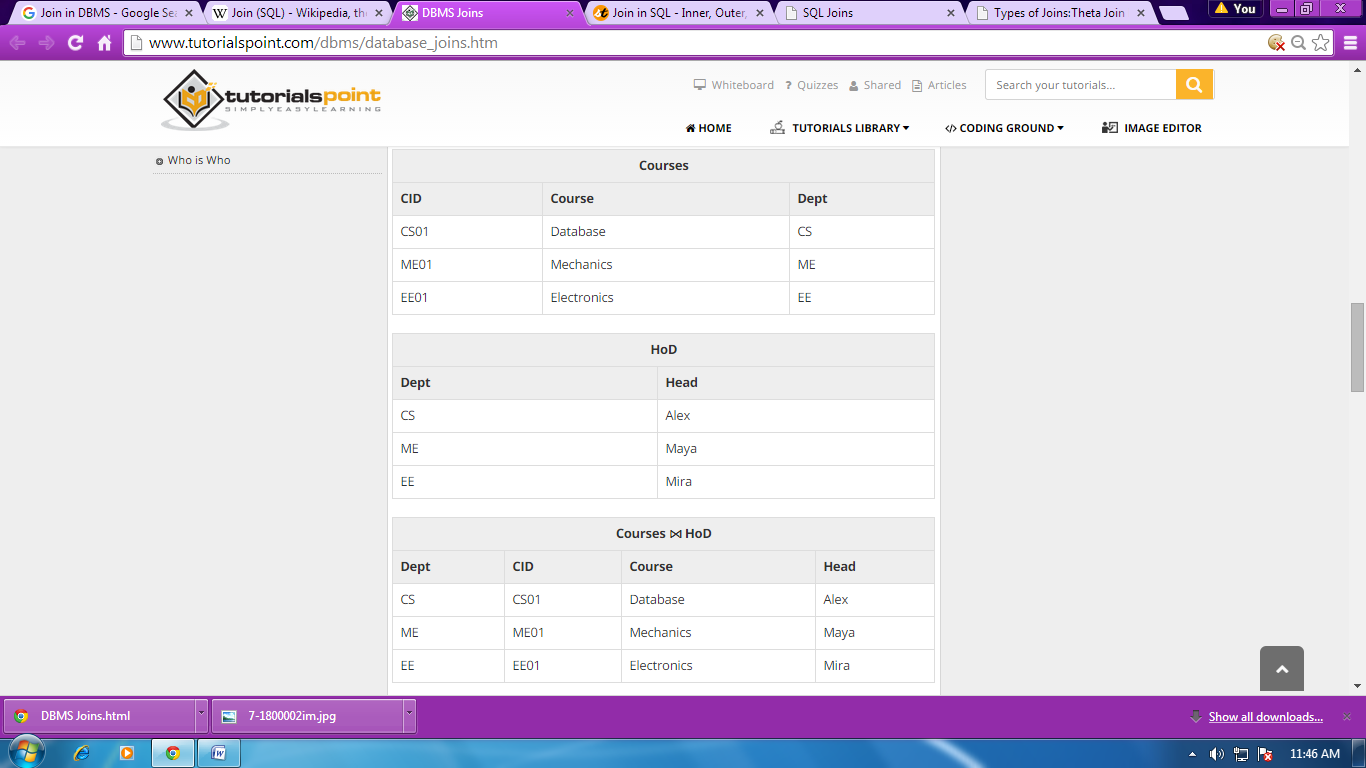


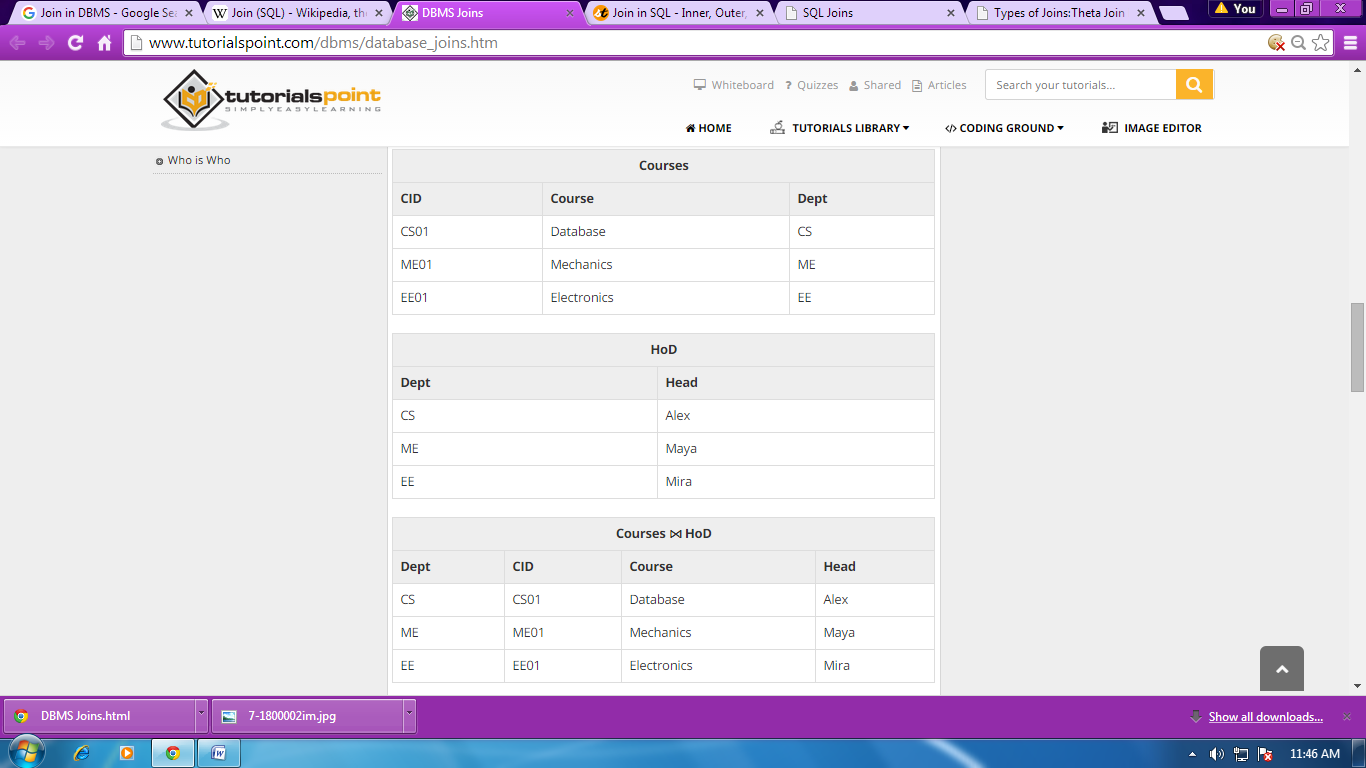
**NATURAL JOIN**

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. 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.

For example:





* **RIGHT JOIN**

In right outer joins all the right tuples of the right relation remain part of the output relation, whereas the left side of the tuples, which do not match with the right relation. The right outer join returns a result table with the matched data of two tables then remaining rows of the right table and null for the left table's columns.

Right Join Syntax is,

select column-name-list

from *table-name1*

**RIGHT OUTER JOIN**

*table-name2*

on table-name1.column-name = table-name2.column-name;

* **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 tuple 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 left outer join returns a result table with the matched data of two tables then remaining rows of the left table and null for the right table's column.

Left Outer Join syntax is,

SELECT column-name-list

from *table-name1*

**LEFT OUTER JOIN**

*table-name2*

on table-name1.column-name = table-name2.column-name;

* **INNER JOIN**

This is a simple JOIN in which the result is based on matched data as per the equality condition specified in the query.

Inner Join Syntax is,

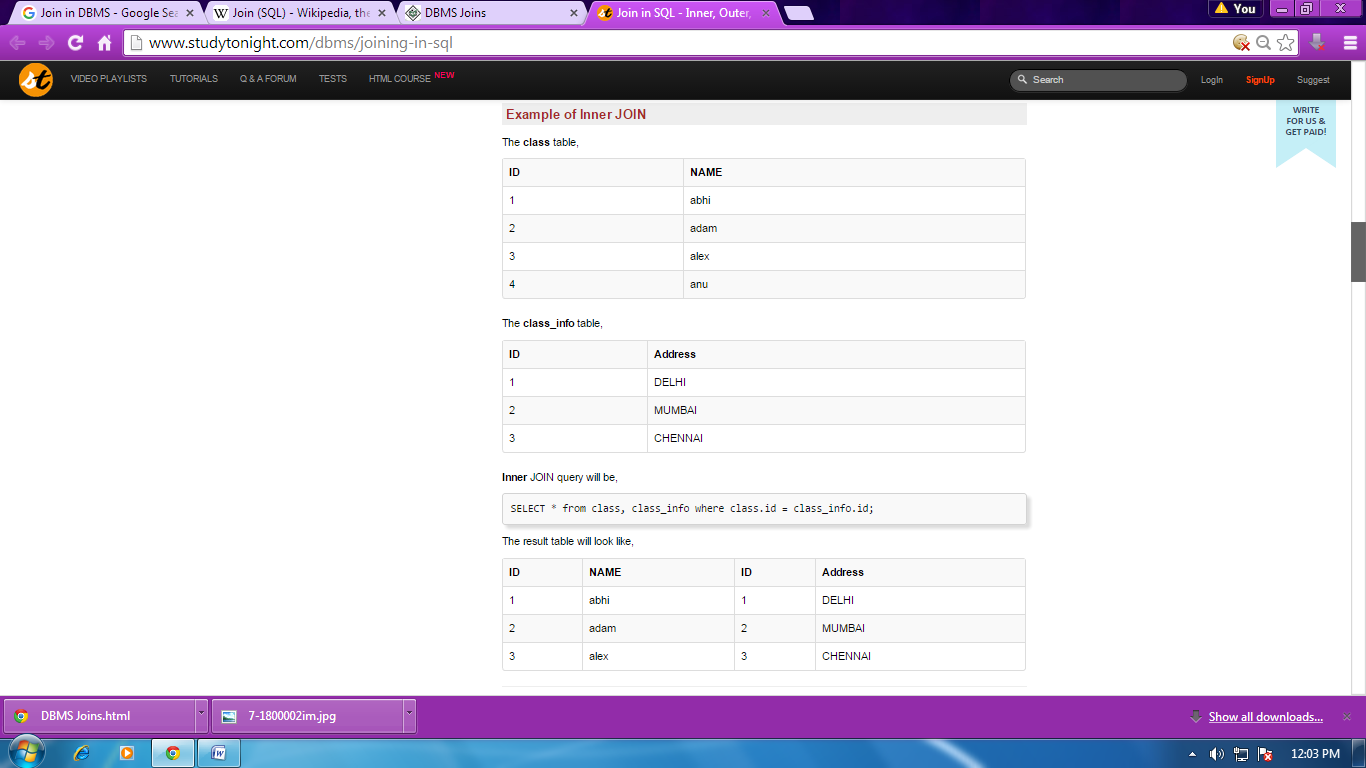
SELECT column-name-list

from *table-name1*

**INNER JOIN**

*table-name2*

WHERE table-name1.column-name = table-name2.column-name;



**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. Formally the semantics of the rename operator is defined as follows: \rho_{a/b}(R) = \{ \ t[a/b] : t \in R \ \}where t[a/b] is defined as the tuple t with the b attribute renamed to a so that: t[a/b] = \{ \ (c, v) \ | \ ( c, v ) \in t, \ c \ne b \ \} \cup \{ \ (a, \ t(b) ) \ \}

**Notation − ρ x (E)**

Where the result of expression E is saved with name of x.

Additional operations are −

* **Set intersection**
* **Assignment**
* **Natural join**

**ASSIGNMENT OPERATION**

The assignment operator is one of the most intuitive to use. It assigns a value to a variable. The only confusion in using this operator could stem from its overloading. All RDBMS overload this operator with an additional function — comparison — in the SQL.

The equals operator (=) is used as an assignment in the following SQL query that updates the price (PROD\_PRICE\_N) column in the PRODUCT table, raising the existing prices by 2 percent:

UPDATE product SET prod\_price\_n

= prod\_price\_n \* 1.02 (10 row(s) affected)

And the same operator would be used for comparing values when used, for example, in the WHERE clause of an SQL statement:

UPDATE product SET prod\_price\_n

= prod\_price\_n \* 1.02 WHERE prod\_id\_n = 1880 (1 row(s)

affected)

This statement assigns a 2 percent increase to a product whose ID is 1880; in the same query, the equals operator (=) is used in its assignment and comparison capacity at the same time.

**DIVISION OPERATION**

The division operation is denoted as **÷.**

Letr(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":

| **A** | **B** |

+--------+-------+

| a | 1 |

| b | 2 |

| a | 2 |

| p | 3 |

| p | 4 |

Relation or table "s":-

Code:

| B |

+------+

| 2 |

| 3 |

Therefore, r ÷ s

Code:

| A |

+------+

| b |

| a |

| p |

**ADDITIONAL OPERATIONS:**

* **SET-INTERSECTION OPERATIONS**

UNION, INTERSECT and EXCEPT operations can be done in SQL corresponding to their operations U, ∩ and **–**in relational algebra only if the domains of the attributes of the relations match and the relations have same arity i.e same number of attributes.

* **NATURAL JOIN OPERATIONS**

It is a binary operation and a combination of certain selections and a Cartesian product into one operation.  
It is denoted as |X|.

It is associative.

It forms a Cartesian product of its two arguments.  
Then performs a selection forcing equality on those attributes those appear in both the relations.  
And finally removes duplicates attributes.  
**r(R):** r is a relation with attributes R.  
**s(S):** s is a relation with attributes S.  
  
If R **∩**S = Ф i.e. they have no attributes in common then **r |X| s = r X s**Example:-  
Table "r":-Code:

| **A** | **B**  | **C**  |

+--------+--------+-------+

| a | b | c |

| d | e | f |

| g | h | i |

Table "s" :- Code:

| **B** | **D** |

+--------+-------+

| b | g |

| p | r |

| e | t |

Therefore, r |X| s :-Code:

| **A**  | **B** | **C** | **D**  |

+-----------+---------+---------+----------+

| a | b | c | g |

| d | e | f | t |

i.e;  
if r (A, B, C), s (B, D) then

Code:

**r|X|s** = **Π r.A, r.B, r.C, s.D (σ r.B = s.B (r X s))**