ST. XAVIER’S COLLEGE

**(Affiliated to Tribhuvan University)**

Maitighar, Kathmandu



DATABASE MANAGEMENT SYSYTEM

Lab Assignment #6

**Submitted by:**

Linus Dhakal  
013BSCCSIT022

**Submitted to:**

|  |  |
| --- | --- |
| Er. Sanjay Kumar Yadav  Lecturer |  |

Department of Computer Science

Date of Submission: September 10, 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.

## THETA 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.

**ILLUSTRATION:**

|  |  |  |
| --- | --- | --- |
| **Student** | | |
| **SID** | **Name** | **Std** |
| 101 | Alex | 10 |
| 102 | Maria | 11 |

|  |  |
| --- | --- |
| **Subjects** | |
| **Class** | **Subject** |
| 10 | Math |
| 10 | English |
| 11 | Music |
| 11 | Sports |

Student\_Detail −

STUDENT ⋈Student.Std = Subject.Class SUBJECT

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Student\_detail** | | | | |
| **SID** | **Name** | **Std** | **Class** | **Subject** |
| 101 | Alex | 10 | 10 | Math |
| 101 | Alex | 10 | 10 | English |
| 102 | Maria | 11 | 11 | Music |
| 102 | Maria | 11 | 11 | Sports |

# 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.

**Syntax:**

SELECT \*

FROM table1

NATURAL JOIN table2;

**Illustration:**

|  |  |  |
| --- | --- | --- |
| **Courses** | | |
| **CID** | **Course** | **Dept** |
| CS01 | Database | CS |
| ME01 | Mechanics | ME |
| EE01 | Electronics | EE |

|  |  |
| --- | --- |
| **HoD** | |
| **Dept** | **Head** |
| CS | Alex |
| ME | Maya |
| EE | Mira |

|  |  |  |  |
| --- | --- | --- | --- |
| **Courses ⋈ HoD** | | | |
| **Dept** | **CID** | **Course** | **Head** |
| CS | CS01 | Database | Alex |
| ME | ME01 | Mechanics | Maya |
| EE | EE01 | Electronics | Mira |

## Right Join

The right 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.

**Syntax:**

select column-name-list

from table-name1

RIGHT OUTER JOIN

table-name2

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

**Illustration:**

The **class** table,

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 1 | abhi |
| 2 | adam |
| 3 | alex |
| 4 | anu |
| 5 | ashish |

The **class\_info** table,

|  |  |
| --- | --- |
| **ID** | **Address** |
| 1 | DELHI |
| 2 | MUMBAI |
| 3 | CHENNAI |
| 7 | NOIDA |
| 8 | PANIPAT |

**Right Join** query will be,

SELECT \* FROM class RIGHT OUTER JOIN class\_info on (class.id=class\_info.id);

The result table will look like,

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **NAME** | **ID** | **Address** |
| 1 | abhi | 1 | DELHI |
| 2 | adam | 2 | MUMBAI |
| 3 | alex | 3 | CHENNAI |
| null | null | 7 | NOIDA |
| null | null | 8 | PANIPAT |

## Left Join

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.

**Syntax:**

SELECT column-name-list

from table-name1

LEFT OUTER JOIN

table-name2

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

Left outer Join Syntax for Oracle is,

select column-name-list

from table-name1,

table-name2

on table-name1.column-name = table-name2.column-name(+);

**Illustration:**

1. The **class** table,

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 1 | abhi |
| 2 | adam |
| 3 | alex |
| 4 | anu |
| 5 | ashish |

The **class\_info** table,

|  |  |
| --- | --- |
| **ID** | **Address** |
| 1 | DELHI |
| 2 | MUMBAI |
| 3 | CHENNAI |
| 7 | NOIDA |
| 8 | PANIPAT |

## 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.

**Syntax:**

SELECT column-name-list

from table-name1

INNER JOIN

table-name2

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

**Illustration:**

The **class** table,

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 1 | abhi |
| 2 | adam |
| 3 | alex |
| 4 | anu |

The **class\_info** table,

|  |  |
| --- | --- |
| **ID** | **Address** |
| 1 | DELHI |
| 2 | MUMBAI |
| 3 | CHENNAI |

# Rename Operation

The REMANE operator returns an existing relation under a new name. This is a Mechanism used for renaming both relations and attributes, this renaming is highly useful when one has to compare a set of tuples of the same relation with other tuples of that relation. It has the following general format :  
  
old\_name as new-name  
  
The as clauses can appear in both the select and from clauses. For example if we want the attribute name loan \_no to be replaced with the name loan\_id, we can write the query as  
  
Select distinct cust\_name, borrower.loan\_no as loan\_no  
From borrower, loan  
Where borrower.loan\_no = loan.loan.no AND  
Branch\_name = KR CIRCLE

# Assignment Operator

Assignment operator causes the user variable on the left hand side of the operator to take on the value to its right. The value on the right hand side may be a literal value, another variable storing a value, or any legal expression that yields a scalar value, including the result of a query (provided that this value is a scalar value). You can perform multiple assignments in the same [**SET**](https://dev.mysql.com/doc/refman/5.0/en/set-statement.html) statement. You can perform multiple assignments in the same statement-

Unlike [**=**](https://dev.mysql.com/doc/refman/5.0/en/assignment-operators.html#operator_assign-equal), the [**:=**](https://dev.mysql.com/doc/refman/5.0/en/assignment-operators.html#operator_assign-value) operator is never interpreted as a comparison operator. This means you can use [**:=**](https://dev.mysql.com/doc/refman/5.0/en/assignment-operators.html#operator_assign-value) in any valid SQL statement (not just in [**SET**](https://dev.mysql.com/doc/refman/5.0/en/set-statement.html) statements) to assign a value to a variable.

mysql> SELECT @var1, @var2;

-> NULL, NULL

mysql> SELECT @var1 := 1, @var2;

-> 1, NULL

mysql> SELECT @var1, @var2;

-> 1, NULL

mysql> SELECT @var1, @var2 := @var1;

-> 1, 1

mysql> SELECT @var1, @var2;

-> 1, 1

mysql> SELECT @var1:=COUNT(\*) FROM t1;

-> 4

mysql> SELECT @var1;

-> 4

You can make value assignments using [**:=**](https://dev.mysql.com/doc/refman/5.0/en/assignment-operators.html#operator_assign-value) in other statements besides [**SELECT**](https://dev.mysql.com/doc/refman/5.0/en/select.html), such as [**UPDATE**](https://dev.mysql.com/doc/refman/5.0/en/update.html), as shown here:

mysql> SELECT @var1;

-> 4

mysql> SELECT \* FROM t1;

-> 1, 3, 5, 7

mysql> UPDATE t1 SET c1 = 2 WHERE c1 = @var1:= 1;

Query OK, 1 row affected (0.00 sec)

Rows matched: 1 Changed: 1 Warnings: 0

mysql> SELECT @var1;

-> 1

mysql> SELECT \* FROM t1;

-> 2, 3, 5, 7

While it is also possible both to set and to read the value of the same variable in a single SQL statement using the[**:=**](https://dev.mysql.com/doc/refman/5.0/en/assignment-operators.html#operator_assign-value) operator, this is not recommended. [Section 9.4, “User-Defined Variables”](https://dev.mysql.com/doc/refman/5.0/en/user-variables.html), explains why you should avoid doing this.

* **[=](https://dev.mysql.com/doc/refman/5.0/en/assignment-operators.html" \l "operator_assign-equal)**

This operator is used to perform value assignments in two cases, described in the next two paragraphs.

Within a [**SET**](https://dev.mysql.com/doc/refman/5.0/en/set-statement.html) statement, **=** is treated as an assignment operator that causes the user variable on the left hand side of the operator to take on the value to its right. (In other words, when used in a [**SET**](https://dev.mysql.com/doc/refman/5.0/en/set-statement.html) statement, **=** is treated identically to [**:=**](https://dev.mysql.com/doc/refman/5.0/en/assignment-operators.html#operator_assign-value).) The value on the right hand side may be a literal value, another variable storing a value, or any legal expression that yields a scalar value, including the result of a query (provided that this value is a scalar value). You can perform multiple assignments in the same [**SET**](https://dev.mysql.com/doc/refman/5.0/en/set-statement.html) statement.

In the **SET** clause of an [**UPDATE**](https://dev.mysql.com/doc/refman/5.0/en/update.html) statement, **=** also acts as an assignment operator; in this case, however, it causes the column named on the left hand side of the operator to assume the value given to the right, provided any**WHERE** conditions that are part of the [**UPDATE**](https://dev.mysql.com/doc/refman/5.0/en/update.html) are met. You can make multiple assignments in the same **SET**clause of an [**UPDATE**](https://dev.mysql.com/doc/refman/5.0/en/update.html) statement.

In any other context, **=** is treated as a [comparison operator](https://dev.mysql.com/doc/refman/5.0/en/comparison-operators.html#operator_equal).

mysql> SELECT @var1, @var2;

-> NULL, NULL

mysql> SELECT @var1 := 1, @var2;

-> 1, NULL

mysql> SELECT @var1, @var2;

-> 1, NULL

mysql> SELECT @var1, @var2 := @var1;

-> 1, 1

mysql> SELECT @var1, @var2;

-> 1, 1

# 5. Division Operation

The SQL divide ( / ) operator is used to divide one expressions or numbers by another.

**Illustration:**

To get data of 'cust\_name',  'opening\_amt',  'receive\_amt',  'outstanding\_amt' and ('receive\_amt'\*5/ 100) as a column heading  'commission' from the customer table with following condition -

**1.** 'outstanding\_amt' is less than or equal to 4000,

the following sql statement can be used :

[view plaincopy to clipboardprint?](http://www.w3resource.com/sql/arithmetic-operators/sql-arithmetic-operators.php)

SELECT cust\_name, opening\_amt, receive\_amt,

outstanding\_amt, (receive\_amt\*5/ 100) commission

FROM customer

WHERE outstanding\_amt<=4000;

Output

CUST\_NAME OPENING\_AMT RECEIVE\_AMT OUTSTANDING\_AMT COMMISSION

---------------------------------------- ----------- ----------- --------------- ----------

Holmes 6000 5000 4000 250

Bolt 5000 7000 3000 350

Karl 4000 6000 3000 300

Steven 5000 7000 3000 350

# 6. Additional Operation

## 6.1 Set Intersection Operation

The SQL INTERSECT operator is used to return the results of 2 or more SELECT statements. However, it only returns the rows selected by all queries or data sets. If a record exists in one query and not in the other, it will be omitted from the INTERSECT results.

**Syntax:**

SELECT expression1, expression2, ... expression\_n

FROM tables

WHERE conditions

INTERSECT

SELECT expression1, expression2, ... expression\_n

FROM tables

WHERE conditions;

**Illustration:**

The following statement combines the results with the **INTERSECT** operator, which returns only those rows returned by both queries:

SELECT product\_id FROM inventories

INTERSECT

SELECT product\_id FROM order\_items;

## 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**