**ST. XAVIER’S COLLEGE**

**(Affiliated to Tribhuvan University)**

Maitighar, Kathmandu



**DBMS Theory Assignment #6**

**Submitted by:**

**Suman Malla**

013BSCCSIT041

**Submitted to:**

|  |  |
| --- | --- |
| Er.Sanjay Kumar Yadav  Lecturer, St. Xavier’s College |  |

**Date of Submission: Thursday, September 10, 2015**

1. **Join**

A [SQL](https://en.wikipedia.org/wiki/SQL) join clause combines [records](https://en.wikipedia.org/wiki/Row_(database)) from two or more [tables](https://en.wikipedia.org/wiki/Table_(database)) in a relational [database](https://en.wikipedia.org/wiki/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](https://en.wikipedia.org/wiki/Field_(computer_science)) from two tables (or more) by using values common to each. ANSI-standard SQL specifies five types of JOIN: INNER, LEFT OUTER, RIGHT OUTER, FULL OUTER and CROSS. As a special case, a table (base table, [view](https://en.wikipedia.org/wiki/View_(database)), or joined table) can JOIN to itself in a self-join.

A programmer writes a join statement to identify the records for joining. If the evaluated predicate is true, the combined record is then produced in the expected format, a record set or a temporary table. It has two types and they are:-

1. **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.

|  |  |  |
| --- | --- | --- |
| **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 |

1. **Natural join**

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) ) }

* **Right join**

|  |  |  |  |
| --- | --- | --- | --- |
| **Courses Right Outer Join HoD** | | | |
| **A** | **B** | **C** | **D** |
| 100 | Database | 100 | Alex |
| 102 | Electronics | 102 | Maya |
| --- | --- | 104 | Mira |

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.

* Left join
* **Inner join**

An inner join requires each record in the two joined tables to have matching records, and is a commonly used join operation in [applications](https://en.wikipedia.org/wiki/Application_software) but should not be assumed to be the best choice in all situations. Inner join creates a new result table by combining column values of two tables (A and B) based upon the join-predicate. The query compares each row of A with each row of B to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied by matching non-NULL values, column values for each matched pair of rows of A and B are combined into a result row.

The result of the join can be defined as the outcome of first taking the [Cartesian product](https://en.wikipedia.org/wiki/Cartesian_product) (or [Cross join](https://en.wikipedia.org/wiki/Join_(SQL)#Cross_join)) of all records in the tables (combining every record in table A with every record in table B) and then returning all records which satisfy the join predicate. Actual SQL implementations normally use other approaches, such as [hash joins](https://en.wikipedia.org/wiki/Hash_join) or [sort-merge joins](https://en.wikipedia.org/wiki/Sort-merge_join), since computing the Cartesian product is slower and would often require a prohibitively large memory space to store.

* **Left Join**

The result of a left outer join (or simply left join) for tables A and B always contains all records of the "left" table (A), even if the join-condition does not find any matching record in the "right" table (B). This means that if the ON clause matches 0 (zero) records in B (for a given record in A), the join will still return a row in the result (for that record)—but with NULL in each column from B. A left outer join returns all the values from an inner join plus all values in the left table that do not match to the right table, including rows with NULL (empty) values in the link field.

**SELECT** \*

**FROM** employee

**LEFT** **OUTER** **JOIN** department **ON** employee.DepartmentID = department.DepartmentID;

1. **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.

Additional operations are −

* Set intersection
* Assignment
* Natural join

1. **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 theWHERE 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.

1. **Division operation**

It 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":-

Code:

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

| **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 |

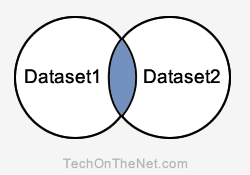
| a |

| p |

+------+

1. **Set intersection operation**

Intersect operation is used to combine two SELECT statements, but it only retuns the records which are common from both SELECT statements. In case of Intersect the number of columns and datatype must be same. MySQL does not support INTERSECT operator.



#### Example of Intersect

The **First** table,

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 1 | abhi |
| 2 | adam |

The **Second** table,

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 2 | adam |
| 3 | Chester |

Intersect query will be,

select \* from First

**INTERSECT**

select \* from second

The result table will look like

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 2 | adam |

1. **Natural join operation**

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.**

**Reference:**

[1]<https://www.google.com.np/search?q=types+of+join&oq=types+of+join&aqs=chrome..69i57j0l5.5347j0j4&sourceid=chrome&es_sm=122&ie=UTF-8>

[2] <http://stackoverflow.com/questions/17946221/sql-join-and-different-types-of-joins>

[3] <http://www.cs.sfu.ca/CourseCentral/354/zaiane/material/notes/Chapter4/node7.html>