**ST. XAVIER’S COLLEGE**

**(Affiliated to Tribhuvan University)**

Maitighar, Kathmandu



**Database Management System Lab Assignment #6z**

**Submitted by:**

Sazjan Neupane

013BSCCSIT036

**Submitted to:**

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

**Date of Submission:** 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. ANSI-standard SQL specifies five types of JOIN: INNER, LEFT OUTER, RIGHT OUTER, FULL OUTER and CROSS.

Join is a special form of cross product of two tables. 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: -

1. Theta Join

2. Equi Join

3. Semi Join

4. Natural Join

5. Outer Joins

**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: -RX S

**Equi­Join**

This is the most used type of join. In equi­join rows are joined on the basis of values of a common attribute between the two relations. It means relations are joined on the basis of common attributes between them; which are meaningful. This means on the basis of primary key, which is a foreign key in another relation. Rows having the same value in the common attributes are joined. Common attributes appear twice in the output. It means that the attributes, which are common in both relations, appear twice, but only those rows, which are selected. Common attribute with the same name is qualified with the relation name in the output. It means that if primary and foreign keys of two relations are having the same names and if we take the equi ­ join of both then in the output relation the relation name will precede the attribute name.

**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 Outer 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.

**Right Outer 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.

**Outer Join**

In outer join all the tuples of left and right relations are part of the output. It means that all those tuples of left relation which are not matched with right relation are left as Null. Similarly all those tuples of right relation which are not matched with left relation are left as Null.

**Semi Join**

In semi join, first we take the natural join of two relations then we project the attributes of first table only. So after join and matching the common attribute of both relations only attributes of first relation are projected.

**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.

**Rename Operation (ρ)**

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

## Assignment Operator

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.

**DIVISION Operator**

The division operation will return a Relation R(X) that includes all tuples t[X] in R(Z) that appear in R1 in combination with every tuple from R2(Y), where Z = X ∪ Y. The division operator is symbolized by:

* R1(Z) ∻ R2(Y)

The division operator is the most difficult to implement in SQL as no SQL command is given for division operation. The division operator would be seen as the opposite of the Cartesian product operator; just as in standard math, the relation between division and multiplication. Therefore a series of current SQL commands have to be utilized in implementation of the division operator. An example of the SQL implementation of division operator:

SELECT surname, forenames

FROM employee X

WHERE NOT EXISTS

(SELECT ‘X’

FROM employee y

WHERE NOT EXISTS

(SELECT ‘X’

FROM employee z

WHERE x.empno = z.empno

AND y.surname = z.surname))

ORDER BY empno

**NATURAL JOIN Operator**

The NATURAL JOIN operation returns results that does not include the JOIN attributes of the second Relation B. It is not required that attributes with the same name be mentioned. The NATURAL JOIN operator is symbolized by:

* A \* ⟗<join condition>B, OR A \* ⟗(<join attributes 1>),

(<join attributes 2>)B

OR A \* B

SQL translation example where attribute dob is Date of Birth and empno is Employee Number:

SELECT A.dob, B.empno

FROM A

NATURAL JOIN B

//where depno =5

We can always use the ‘where’ clause to further restrict our output and stop a Cartesian product output.

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.

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 |