**ST. XAVIER’S COLLEGE**

(Affiliated to Tribhuvan University)

Maitighar, Kathmandu



Database Management System

Theory Assignment #6

**Submitted by Submitted to**

Victor Kumar Sapkota Er. Sanjay Kr. Yadav 013BSCCSIT047 Lecturer

Sem IV Department of Computer Science

**Submitted on:** August 10, 2015

**JOIN**

We understand the benefits of taking a Cartesian product of two relations, which gives us all the possible tuples that are paired together. But it might not be feasible for us in certain cases to take a Cartesian product where we encounter huge relations with thousands of tuples having a considerable large number of attributes.

**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.

An SQL JOIN clause is used to combine rows from two or more tables, based on a common field between them.JOIN is a method to retrieve data from two or more database tables.

**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 ⋈θR2

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.

**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.

**Eg:**

SELECT cFirstName, cLastName, orderDate

FROM customers NATURAL JOIN orders;

**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**.

RENAME TABLE ***tbl\_name*** TO ***new\_tbl\_name***

[, ***tbl\_name2*** TO ***new\_tbl\_name2***] ...

**Assignment Operation**

| **Name** | **Description** |
| --- | --- |
| **=** | Assign a value (as part of a **SET** statement, or as part of the **SET** clause in an **UPDATE** statement) |
| **:=** | Assign a value |

:=

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/5.1/en/set-statement.html) statement. You can perform multiple assignments in the same statement-

Unlike [**=**](https://dev.mysql.com/doc/refman/5.1/en/assignment-operators.html#operator_assign-equal), the [**:=**](https://dev.mysql.com/doc/refman/5.1/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/5.1/en/assignment-operators.html#operator_assign-value) in any valid SQL statement (not just in [**SET**](https://dev.mysql.com/doc/refman/5.1/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

=

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/5.1/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/5.1/en/set-statement.html) statement, **=** is treated identically to [**:=**](https://dev.mysql.com/doc/refman/5.1/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/5.1/en/set-statement.html) statement.

In the **SET** clause of an [**UPDATE**](https://dev.mysql.com/doc/refman/5.1/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/5.1/en/update.html) are met. You can make multiple assignments in the same **SET**clause of an [**UPDATE**](https://dev.mysql.com/doc/refman/5.1/en/update.html) statement.

In any other context, **=** is treated as a [comparison operator](https://dev.mysql.com/doc/refman/5.1/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

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.

Additional Operations

“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)

• Thus, set-intersection must follow the same compatibility rules as set-difference: same arity, corresponding domains.

**Natural Join**

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.

**Division**

• The division operation is a binary operation, notated as ÷, on relations r(R) and s(S) such that S ⊆ R. Intuitively, division is a “for all” query — it returns the tuples in r that “match” all of the tuples in s. 2

• Formally, r ÷ s is a relation with schema R − S whose tuples t satisfy both of these conditions: 1. t ∈ ΠR−S(r) 2. ∀ts∈ s, ∃tr∈ r that satisfies both of: (a) tr[S] = ts[S] (b) tr[R − S] = t

• Division can also be defined in terms of the relational algebra. Again, we start with r(R) and s(S) with S ⊆ R: r ÷ s = ΠR−S(r) − ΠR−S((ΠR−S(r) × s) − ΠR−S,S(r)) (1)

• This is quite an eyeful, so we break it down like this: – First, we take ΠR−S(r). This defines the relation for the first condition of the formal definition: t ∈ ΠR−S(r), or the tuples in r with their s-shared attributes removed. – Now, from this, we perform set-difference using: ΠR−S((ΠR−S(r) × s) − ΠR−S,S(r)) – So these tuples are removed from ΠR−S(r). What are these tuples exactly? We start with ΠR−S(r) × s: it takes all of the tuples in r and removes any attributes that r shares with s, then pairs those tuples with every tuple in s. – ΠR−S,S(r) just rearranges the attributes in r so that the attributes that are in r alone are listed first, followed by the attributes that r shares with s. This ensures that the relation of the first difference term, ΠR−S(r) × s, is compatible with ΠR−S,S(r) (same arity, corresponding domains). – So, the set-difference operation can take place, and the result would be the r-to-s tuple matchups that are not in r. – If we then remove the s attributes from that relation, resulting in ΠR−S((ΠR−S(r)× s)−ΠR−S,S(r)), we now have the tuples in r for which at least one tuple in s does not match. Since r ÷ s is about tuples in r that have a corresponding match for all tuples in s, then these are precisely the tuples that we don’t want, and so we subtract them from ΠR−S(r).

• Phew! When all is said and done, just remember that division, r ÷ s, is all about “for all” — tuples in r that match all tuples in s.

Set-Intersection Operation

Natural Join Operation