ST. XAVIER’S COLLEGE

**(Affiliated to Tribhuvan University)**

**Maitighar, Kathmandu**

****

**Database Management System**

**Theory Lab Assignment #6**

**SUBMITTED BY:**

**Siddhant Rimal**

**013BSCCSIT039**

**SUBMITTED TO**

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

Submission Date: September 13th 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.

The most common type of join is: **SQL INNER JOIN (simple join)**. An SQL INNER JOIN return all rows from multiple tables where the join condition is met.

Example:

SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate

FROM Orders

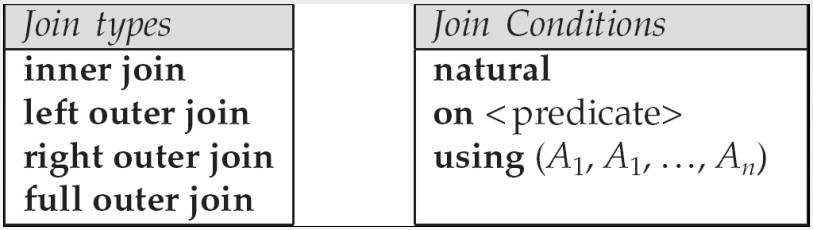
INNER JOIN Customers

ON Orders.CustomerID=Customers.CustomerID

Join operations take two relations and return as a result another relation.  
■ These additional operations are typically used as subquery expressions in the from clause

■ Join condition – defines which tuples in the two relations match, and  
what attributes are present in the result of the join.

■ Join type – defines how tuples in each relation that do not match any  
tuple in the other relation (based on the join condition) are treated.



Types of join:

1. Theta Join
2. Equi Join
3. Semi Join
4. Natural Join
5. Outer Join

# Theta Join:

A *theta-join* is any Cartesian product that's filtered by a condition which compares values from both Tables. That is, the general theta-join form is:

<Table\_1.Column> relator <Table\_2.Column>

where the relator is almost always "=", as in this example:

Sellers.seller\_name = Sales.seller\_name

This special case of theta-join — where the relation is equality — is called an *equijoin*.

# 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 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**.

It can be used in two ways :

* + return the result of expression E in the table named *x*.
  + return the result of expression E in the table named *x* with the attributes renamed to A1, A2,…, An.

It’s benefit can be understood by the solution of the query “ Find the largest account balance in the bank”

It can be solved by following steps:

1. Find out the relation of those balances which are not largest.
   1. Consider Cartesion product of Account with itself i.e. Account × Account
   2. Compare the balances of first Account table with balances of second Account table in the product.
   3. For that we should rename one of the account table by some other name to avoid the confusion
   4. It can be done by following operation

*ΠAccount.balance (σAccount.balance < d.balance(Account× ρd(Account))*

* 1. So the above relation contains the balances which are not largest.

1. Subtract this relation from the relation containing all the balances i.e . *Πbalance (Account)*.
2. So the final statement for solving above query is

*Πbalance (Account)- ΠAccount.balance (σAccount.balance < d.balance(Account× ρd(Account))*

Assignment Operation

The assignment operation (←) provides a convenient way to express complex queries. Particularly with division, that relational algebra feels a lot like programming: there are many steps to some expressions, with intermediate or temporary relations along the way. For this very reason, we have the assignment operation, which works a lot like assignments in a programming language. It is notated with the left-pointing arrow ←:

variable ← E

where E is any relational algebra expression.

● Write query as a sequential program consisting of 4 a series of assignments 4 followed by an expression whose value is displayed as a result of the query.  
●Assignment must always be made to a temporary relation variable.

■ Example:

Write r ÷ s as

temp1 ← ∏R­S (r ) temp2 ← ∏R­S ((temp1 x s ) – ∏R­S,S(r )) result = temp1 – temp2

● The result to the right of the ← is assigned to the relation variable on the left of the ←.

● May use variable in subsequent expressions.

**Division Operation:**

Let r(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.

* denoted by ÷ is used for queries that include the phrase “for all”.
* For example “Find customers who has an account in all branches in branch city Agra”. This query can be solved by following statement.

*ΠCustomer-name. branch-name (* )÷*Πbranch-name (σBranch-city=”Agra”(Branch)*

* The division operations can be specified by using only basic operations as follows: Let r(R) and s(S) be given relations for schema R and S with

r ÷ s = ΠR-S(r) - ΠR-S ((ΠR-S (r) × s) - ΠR-S,S (r))

* The division operations can be specified by using only basic operations as follows: Let r(R) and s(S) be given relations for schema R and S with

r ÷ s = ΠR-S(r) - ΠR-S ((ΠR-S (r) × s) - ΠR-S,S (r))

Example:

|  |  |  |
| --- | --- | --- |
| Relation or table "r":- | Relation or table "s":- | Therefore, r ÷ s |
| Code: | Code: | Code: |
| | A | B |  \_\_\_\_\_\_\_\_\_  | a | 1 |  | b | 2 |  | a | 2 |  | p | 3 |  | p | 4 | | | B |  \_\_\_\_  | 2 |  | 3 |  \_\_\_\_ | | A |  \_\_\_\_  | b |  | a |  | p |  \_\_\_\_ |

* 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 Operation**

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)

Set intersection: v = r ∩ s

●when a tuple is inserted in r we check if it is present in s, and if so we add it to v.

●If the tuple is deleted from r, we delete it from the intersection if it is present.

●Updates to s are symmetric

●The other set operations, union and set difference are handled in a similar fashion.

**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.

* Forms Cartesian product of its two arguments, performs selection forcing equality on those attributes that appear in both relations
* For example consider Borrower and Loan relations, the natural join between them will automatically perform the selection on the table returned by Borrower × Loan which force equality on the attribute that appear in both Borrower and Loan i.e. Loan-no and also will have only one of the column named Loan-No.
* That means = *σBorrower.Loan-no = Loan.Loan-no* (Borrower × Loan).
* The table returned from this will be as follows:

Eliminate rows that does not satisfy the selection criteria

“*σBorrower.Loan-no = Loan.Loan-no”* from Borrower × Loan =

|  |  |  |  |
| --- | --- | --- | --- |
| **Borrower.Cust-name** | **Borrower.Loan-no** | **Loan.Loan-no** | **Loan.Amount** |
| Ram | L-13 | L-13 | 1000 |
| ~~Ram~~ | ~~L-13~~ | ~~L-30~~ | ~~20000~~ |
| ~~Ram~~ | ~~L-13~~ | ~~L-42~~ | ~~40000~~ |
| ~~Shyam~~ | ~~L-30~~ | ~~L-13~~ | ~~1000~~ |
| Shyam | L-30 | L-30 | 20000 |
| ~~Shyam~~ | ~~L-30~~ | ~~L-42~~ | ~~40000~~ |
| ~~Suleman~~ | ~~L-42~~ | ~~L-13~~ | ~~1000~~ |
| ~~Suleman~~ | ~~L-42~~ | ~~L-30~~ | ~~20000~~ |
| Suleman | L-42 | L-42 | 40000 |

And will remove one of the column named Loan-no.

* i.e. =

|  |  |  |
| --- | --- | --- |
| **Cust-name** | **Loan-no** | **Amount** |
| Ram | L-13 | 1000 |
| Shyam | L-30 | 20000 |
| Suleman | L-42 | 40000 |