05. Querying Relations

[PM Jat, DAIICT, Gandhinagar]

Types of JOINs

Relational theory defines various types of JOINs. Mostly this classification is based on the following two aspects:

(1) Nature of join condition, and
(2) The way result is computed for cases of "un-matched" evaluation of join conditions.

Let us look at them one by one.

EQUI JOIN

We have seen a type of JOIN that is most commonly used. Though we have not given any type of name to that, this has been "equi-join", discussed below.

This is called so because it uses only "equal-to" in join condition. Below is one of the example that we have already seen.

$$employee \xrightarrow[eno=mgr\_eno]{\bowtie} department$$

```
SELECT * FROM employee JOIN department ON (eno=mgr_eno);
```

Theta JOIN

It is the generalized form of join.

We have seen the following form of JOIN

$$r_1 \xrightarrow[< join - condition >]{\bowtie} r_2$$

Following is the general form of the join condition:

```
<condition-1> AND|OR <condition-2> … <condition-n>
```

Where each condition is in the form of $A_i\ \theta\ B_j$ and A_i is an attribute from relation r_1 and B_j is an attribute from relation r_2 and θ (theta) is one of the comparison operators $\{=, \geq, \leq, \neq\}$.

A JOIN operation with such a general join condition is called a **THETA JOIN**.

However in most practical cases we are going to see θ as equal ($=$) only.

However, there can be examples where non equal is also used. Here is the one:

$$employee\ e1 \xrightarrow[e1.eno <> e2.eno\ AND\ e1.ename=e2.ename]{\bowtie} employee\ e2$$

```
SELECT * FROM employee e1 JOIN employee e2
       ON (e1.eno <> e2.eno AND e1.ename=e2.ename);
```

Can you guess, what would it result to?

NATURAL JOIN

Consider following JOIN

$$program\ p \overset{\bowtie}{\underset{p.did=d.did}{}} department\ d$$

```
SELECT * FROM program AS p JOIN department AS d ON p.did = d.did;
```

Here is a snapshot of a result snapshot of above join

pid	pname	intake	p.did	d.did	dname
BCS	BTech(CS)	60	CS	CS	Computer Engineering
MCS	MTech(CS)	20	CS	CS	Computer Engineering
BEE	BTech(EE)	40	EE	EE	Electrical Engineering
BME	BTech(ME)	40	ME	ME	Mechanical Engineering
BIT	BTech(IT)	40	CS	CS	Computer Engineering
BEC	BTech(EC)	40	EE	EE	Electrical Engineering

You notice two columns of DID are identical in terms of values! Also, requires explicit including relation identifier for referring to a particular attribute/column as in the following SQL expression. Even if does not matter whether we use `p.did` or `d.did`:

```
SELECT p.did, pname, dname FROM program AS p JOIN department AS d
                                        ON p.did = d.did;
```

To simplify expressions, here, algebra and SQL defines an another type of join called NATURAL JOIN to deal with just this situation.

It is a join where we do not explicitly specify join-condition.

NATURAL JOIN is special case of equi-join where join condition is inbuilt and that is, typically equality of the FK-PK pair.

However, the definition of natural-join makes it little more explicit and defines it as

equi-join based on "common" attributes.

Common in terms of names of attributes.

In the example of program and department. Let us use the symbol $*$ for natural join, Natural Join be expressed as:

$$program * department$$

Here since the common attribute is DID, the implicit join condition becomes is "p.did=d.did" !

In SQL we have a NATURAL JOIN as a keyword, and written as follows:

```
SELECT * FROM program NATURAL JOIN department;
```

Here is snapshot of a result of above join.

did	pid	pname	intake	dname
CS	BCS	BTech(CS)	60	Computer Engineering
CS	MCS	MTech(CS)	20	Computer Engineering
EE	BEE	BTech(EE)	40	Electrical Engineering
ME	BME	BTech(ME)	40	Mechanical Engineering
CS	BIT	BTech(IT)	40	Computer Engineering
EE	BEC	BTech(EC)	40	Electrical Engineering

As a general case, suppose a relation r1 and relation r2 has common attribute names A1, A2, … An.

$$r1 * r2 \equiv r1 \overset{\bowtie}{\underset{r1.a1 = r2.a1\ AND\ r1.a2 = r2.a2\ AND\ \ldots r1.an = r2.an}{}} r2$$

Important to note that to use natural join operand relations has to have valid common sets of attributes!

Example

Consider relation r1 and r2 shown below. Result of JOIN and Natural JOIN is also shown.

r1

a	b	c
a1	b1	c1
a2	(Null)	c2

r2

b	d
b1	d1
b2	d2

```
select * from r1 join r2 on (r1.b=r2.b);
```

a	b	c	b	d
a1	b1	c1	b1	d1

```
select * from r1 natural join r2;
```

b	a	c	d
b1	a1	c1	d1

Let us make some interesting observations:

NATURAL JOIN when No Common Attributes:

Consider student and program relations.

What following expression would evaluate to?

$$student * program$$

```
SELECT * FROM student NATURAL JOIN program;
```

Above SQL statement would produce CROSS PRODUCT of student and program relations.

This is because these relations have no common names of attributes, and as result no join condition. You can however note that relation student has attribute progid and relation program has attribute pid and both actually represent program id. So there are common attributes (semantically) but

names are different. Therefore, it can be acceptable in relational algebra but SQL does not yield desirable results.

NATURAL JOIN when all attributes are common

Consider employee relation of company database, what would be following Natural Join in SQL be producing:

```
SELECT * FROM employee e1 NATURAL JOIN employee e2;
```

Execute the query yourself and observe the result!

INNER JOIN

Consider employee and department relations in company database. Below is a snapshot of both relations.

eno	ename	dob	gender	salary	super_eno	dno
105	Jayesh	10-11-1967	M	55000		1
102	Sanna	23-07-1982	F	35000	105	5
106	Vijay	20-06-1971	M	53000	105	4
101	Sanjay	09-11-1955	M	70000		
108	Priya	19-07-1978	F	45000	106	4
104	Ramesh	15-09-1972	M	38000	106	5
103	Kalpesh	31-07-1982	F	25000	102	5
107	Ahmad	29-03-1979	M	25000	106	4
111	Kirit	08-12-1985	M	40000	102	5

dno	dname	mgr_eno	mgrstartdate
5	Research	102	22-05-1998
4	Administration	106	01-01-2007
1	Headquater	104	19-06-2001

Now consider following JOIN

$$employee\ e \overset{\bowtie}{\underset{e.dno=d.dno}{\quad}} department\ d$$

```
SELECT * FROM employee AS e JOIN department AS d ON (e.dno=d.dno);
```

Following is snapshot of result relation of above join.

eno	ename	dob	gender	salary	super_eno	e.dno	d.dno	dname	mgr_eno	mgrstartdate
105	Jayesh	10-11-1967	M	55000		1	1	Headquater	104	19-06-2001
102	Sanna	23-07-1982	F	35000	105	5	5	Research	102	22-05-1998
106	Vijay	20-06-1971	M	53000	105	4	4	Administration	106	01-01-2007
108	Priya	19-07-1978	F	45000	106	4	4	Administration	106	01-01-2007
104	Ramesh	15-09-1972	M	38000	106	5	5	Research	102	22-05-1998
103	Kalpesh	31-07-1982	F	25000	102	5	5	Research	102	22-05-1998
107	Ahmad	29-03-1979	M	25000	106	4	4	Administration	106	01-01-2007
111	Kirit	08-12-1985	M	40000	102	5	5	Research	102	22-05-1998

Now question is there loss of any tuple in this join from any of operand relations that mean all tuples from both relations are there in joined relation!

Yes, there is loss of an employee tuple "`101, Sanjay, …`"

Is it correct to loose such a tuple?

May be OK in some cases but may not correct in others.

Consider a query: List ENO, ENAME, DNAME of all employees. If perform above join for this query, we would not like to miss an employee. However, it might be OK loosing any tuple from department side if that department has no employee working for.

$$employee\ e \overset{\bowtie}{\underset{e.eno=d.mgr\_eno}{}}\ department\ d$$

```
SELECT * FROM employee AS e JOIN department AS d
         ON (e.eno=d.mgr_eno);
```

Following is snapshot of result

eno	ename	dob	gender	salary	super_eno	dno	dno-2	dname	mgr_eno	mgrstartdate
102	Sanna	23-07-1982	F	35000	105	5	5	Research	102	22-05-1998
106	Vijay	20-06-1971	M	53000	105	4	4	Administration	106	01-01-2007
104	Ramesh	15-09-1972	M	38000	106	5	1	Headquater	104	19-06-2001

Now if we have a query: List (DName, ENO [of manager], and EName [of manager]), then it is OK to loose employee tuples from employee but cannot afford to lose a tuple from department side.

Why is it so?

Is it correct?

Actually Yes!

Consider first case, it is because employee tuple "`101, Sanjay, …`" has NULL in dno attribute which is there in join condition and in this join condition evaluates to FALSE because there is no tuple in department corresponding to this employee tuple.

In **INNER JOIN** tuples are combined and included in result only if join-condition evaluates to TRUE. Use of Null for attributes in join-condition evaluates to FALSE.

All joins discussed so far including Theta Join, Equi-Join, Natural Join are INNER JOINs.

Let us state INNER JOIN through following procedure:

Suppose we want to computer "`r1 INNER JOIN r2 on join-cond`". .

```
result_set = NULL;
For each tuple t1 in relation r1
For each tuple t2 in relation r2
If join-cond(t1,t2) met* then
      derive new tuple t = t1 concatenate t2
      append t to result_set
```

* if value for any attribute of join condition in t1 or t2 is NULL, then the join-cond evaluates to FALSE.

INNER JOIN in SQL

SQL has keyword INNER JOIN and expressed as following:

```
SELECT * FROM student INNER JOIN program ON progid=pid;
```

However, INNER is optional here, therefore it is same as following

```
SELECT * FROM student JOIN program ON progid=pid;
```

Note that it is same; our theta JOIN!

OUTER JOIN

Outer join includes tuples from operand relation even if join condition evaluates to FALSE. Based on side of inclusion of "un-matched" tuples in results, outer joins are classified to following three:

- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN

LEFT OUTER JOIN

Let us understand this operation as following.

Consider computing "`r1 LEFT OUTER JOIN r2 ON join-cond`"

Let this be expressed as following in relational algebra

$$r1 \frac{LEFT \bowtie}{< join - cond >} r2$$

```
result_set =
      all matched tuples (INNER JOIN)
            UNION
      remaining tuples of r1 combined with NULLs in r2 side;
```

SQL has keyword LEFT OUTER JOIN and expressed as following:

```
SELECT * FROM student LEFT OUTER JOIN program ON progid=pid;
```

More Examples

Let us look at few examples.

$$employee\ e \overset{LEFT\ \bowtie}{\underset{e.dno=d.dno}{}} department\ d$$

```
SELECT * FROM employee AS e LEFT OUTER JOIN department AS d
       ON (e.dno=d.dno);
```

Below is snapshot of result relation. Note that INNER JOIN would not have the highlighted tuple in the result!

eno	ename	dob	gender	salary	super_eno	e.dno	d.dno	dname	mgr_eno	mgrstartdate
105	Jayesh	10-11-1967	M	55000	null	1	1	Headquater	104	19-06-2001
102	Sanna	23-07-1982	F	35000	105	5	5	Research	102	22-05-1998
106	Vijay	20-06-1971	M	53000	105	4	4	Administration	106	01-01-2007
101	Sanjay	09-11-1955	M	70000	null	null	null	null	null	null
108	Priya	19-07-1978	F	45000	106	4	4	Administration	106	01-01-2007
104	Ramesh	15-09-1972	M	38000	106	5	5	Research	102	22-05-1998
103	Kalpesh	31-07-1982	F	25000	102	5	5	Research	102	22-05-1998
107	Ahmad	29-03-1979	M	25000	106	4	4	Administration	106	01-01-2007
111	Kirit	08-12-1985	M	40000	102	5	5	Research	102	22-05-1998

Here is another LEFT JOIN and its result snapshot

$$employee\ e \overset{LEFT\ \bowtie}{\underset{e.eno=d.mgr\_eno}{}} department\ d$$

```
SELECT * FROM employee AS e LEFT OUTER JOIN department AS d
       ON (e.eno=d.mgr_eno);
```

Observe the result of this JOIN try understanding the LEFT OUTER JOIN

eno	ename	dob	gender	salary	super_eno	e.dno	d.dno	dname	mgr_eno	mgrstartdate
102	Sanna	23-07-1982	F	35000	105	5	5	Research	102	22-05-1998
106	Vijay	20-06-1971	M	53000	105	4	4	Administration	106	01-01-2007
104	Ramesh	15-09-1972	M	38000	106	5	1	Headquater	104	19-06-2001
101	Sanjay	09-11-1955	M	70000	null	null	null	null	null	null
107	Ahmad	29-03-1979	M	25000	106	4	null	null	null	null
108	Priya	19-07-1978	F	45000	106	4	null	null	null	null
103	Kalpesh	31-07-1982	F	25000	102	5	null	null	null	null
111	Kirit	08-12-1985	M	40000	102	5	null	null	null	null
105	Jayesh	10-11-1967	M	55000	null	1	null	null	null	null

Remember that in SQL OUTER keyword is optional. Therefore, following two are equivalent:

```
SELECT * FROM employee AS e LEFT OUTER JOIN department AS d
        ON (e.eno=d.mgr_eno);
SELECT * FROM employee AS e LEFT JOIN department AS d
        ON (e.eno=d.mgr_eno);
```

RIGHT OUTER JOIN

RIGHT OUTER JOIN is opposite LEFT JOIN.

Consider computing "`r1 RIGHT OUTER JOIN r2 ON join-con`"

Let this be expressed as following in relational algebra

$$r1 \frac{\bowtie RIGHT}{< join - cond >} r2$$

Following algorithm explains result of this JOIN

```
result_set =
      all matched tuples (INNER JOIN)
            UNION
      remaining tuples of r2 combined with NULLs in r1 side;
```

As an example:

$$program\ p \frac{\bowtie RIGHT}{p.did=d.id} department\ d$$

```
SELECT * FROM program AS p RIGHT OUTER JOIN department AS d
       ON (p.did=d.did);
```

Below is result snapshot

pid	pname	intake	p.did	d.did	dname
BCS	BTech(CS)	60	CS	CS	Computer Engineering
MCS	MTech(CS)	20	CS	CS	Computer Engineering
BEE	BTech(EE)	40	EE	EE	Electrical Engineering
BME	BTech(ME)	40	ME	ME	Mechanical Engineering
BIT	BTech(IT)	40	CS	CS	Computer Engineering
BEC	BTech(EC)	40	EE	EE	Electrical Engineering
null	null	null	null	CE	Civil Engineering

FULL OUTER JOIN

Full outer join is both LEFT and RIGHT OUTER JOIN.

Consider computing "`r1 FULL OUTER JOIN r2 ON join-cond`"

Let this be expressed as following in relational algebra

$$r1 \frac{FULL \bowtie}{< join - cond >} r2$$

```
result_set =
      all matched tuples (INNER JOIN)
            UNION
      remaining tuples of r1 combined with NULLs in r2 side;
            UNION
      remaining tuples of r2 combined with NULLs in r1 side;
```

SQL has keyword LEFT OUTER JOIN and expressed as following:

```
SELECT * FROM student FULL OUTER JOIN program ON progid=pid;
```

JOIN SUMMARY

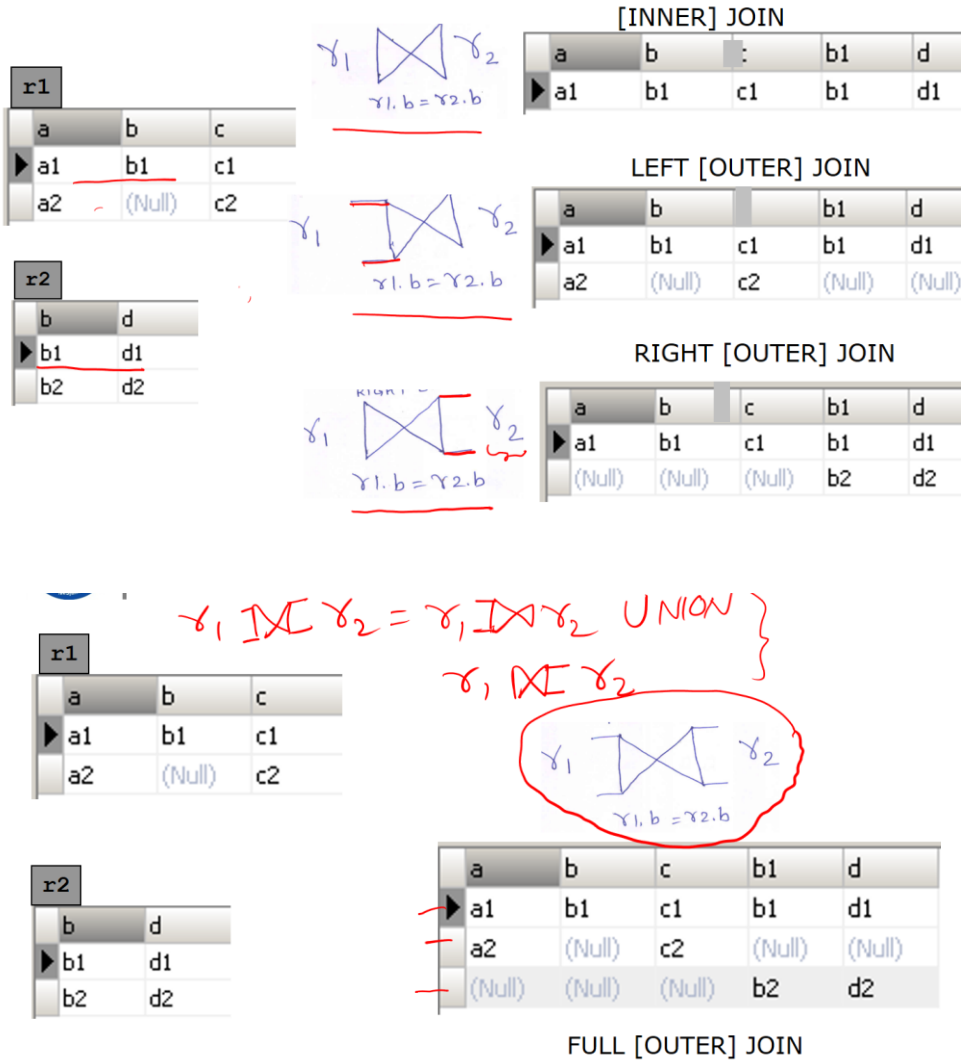
```
--INNER JOIN:
r1 [INNER]* JOIN r2 on (r1.b=r2.b);

-- LEFT OUTER:
r1 LEFT [OUTER]* JOIN r2 on (r1.b=r2.b);

--RIGHT OUTER:
r1 RIGHT [OUTER]* JOIN r2 on (r1.b=r2.b);

--FULL OUTER:
r1 FULL [OUTER]* JOIN r2 on (r1.b=r2.b);
```

* optional



Ordering of JOINS in FROM clause

Join operation is *non-associative*. Ordering of evaluation of joins in a FROM clause of SELECT statement is left to right, if there are multiple joins.

```
SELECT ssn, fname, pname AS Project, dname AS "Controlling Dept",  hours
FROM works_on NATURAL JOIN project NATURAL JOIN department JOIN employee
ON essn = ssn;
```

Above query mean as below –

```
SELECT ssn, fname, pname AS Project, dname AS "Controlling Dept",  hours
FROM (((works_on NATURAL JOIN project) NATURAL JOIN department) JOIN
employee ON essn = ssn);
```

SET Operations

We already understand meaning of following operations

- UNION: $A \cup B$
- INTERSECT: $A \cap B$
- EXCEPT (MINUS): $A - B$

Relational Algebra supports these operations on relations. However, there is compatibility requirement for set operations on relations called as "**Union Compatibility Requirement**", which is defined as following:

- The operand relations R1(A1, A2, ..., An) and R2(B1, B2, ..., Bn) must have the same number of attributes, and the domains of corresponding attributes must be compatible; that is, dom(Ai)=dom(Bi) for i=1, 2, ..., n.
- The resulting relation for $r1 \cup r2$ has the same attribute names as the first operand relation R1
- This applies to Intersection and subtraction as well

Examples

Here are the examples taken from the book Elmasri/Navathe.

(a) Two union-compatible relations.

(b) STUDENT ∪ INSTRUCTOR.

(c) STUDENT ∩ INSTRUCTOR.

(d) STUDENT – INSTRUCTOR

(e) INSTRUCTOR – STUDENT

(a)

STUDENT	FN	LN
	Susan	Yao
	Ramesh	Shah
	Johnny	Kohler
	Barbara	Jones
	Amy	Ford
	Jimmy	Wang
	Ernest	Gilbert

INSTRUCTOR	FNAME	LNAME
	John	Smith
	Ricardo	Browne
	Susan	Yao
	Francis	Johnson
	Ramesh	Shah

(b)

FN	LN
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

(c)

FN	LN
Susan	Yao
Ramesh	Shah

(d)

FN	LN
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

(e)

FNAME	LNAME
John	Smith
Ricardo	Browne
Francis	Johnson

SET Operations in SQL

SQL has keywords -

`UNION` for union

`INTERSECT` for intersection

`EXCEPT` for minus

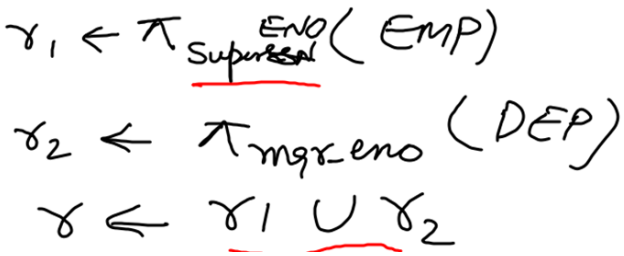
Syntax:

```
SELECT ...
     [UNION/INTERSECT/EXCEPT]
SELECT ... ;
```

Query Examples for SET Operation

- Employee that are either manager or supervisor
- Students either study in BCS or BIT
- Employee that are not manager:
- Employee that are manager also: INTERSECTION

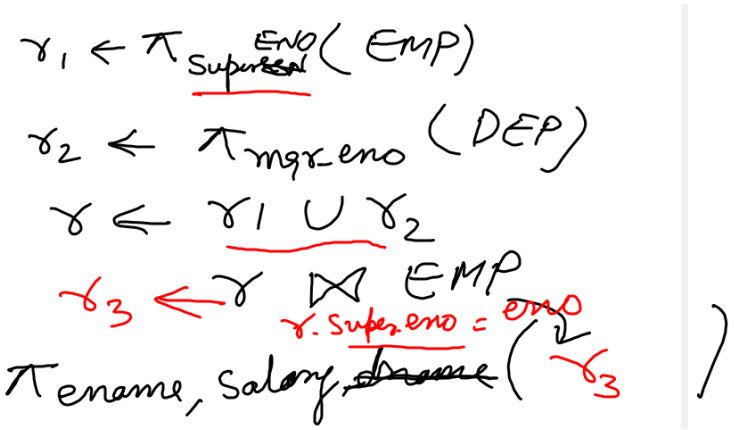
Example ##: List ENOs of supervisors and manager in company database



SQL:

```
select super_eno from employee
union
select mgr_eno from department;
```

Example ##: List ENO, Name, Salary of supervisors and manager in company database



SQL:

```
select eno, ename, salary
    from employee
         natural join
          (select super_eno AS eno from employee
           union
           select mgr_eno from department
         ) as r;
```

Semi Join and Semi-difference (IN and NOT IN of SQL)

Because of union compatibility requirement, direct use of SET operations in SQL is limited.

- INTERSECT is accomplished by NATURAL JOIN or SEMI JOIN/SEMI INTERSECT (IN in SQL)
- EXCEPT typically is accomplished by SEMI Difference (NOT IN of SQL)
- In some cases, UNION can be performed by having OR in tuple SELECTION criteria (in WHERE Clause of SQL).
  For example, consider query "List students of BCS and BIT; and solved as following:

```
SELECT * FROM student WHERE progid='BCS'
UNION
SELECT * FROM student WHERE progid='BIT';
```

  Can also be written as:

```
SELECT * FROM student WHERE progid='BCS' OR progid='BIT';
```

  However, this is typically possible only when we have union based on different options on same relation or so. For example, following UNION may not be possible through OR-ing strategy
  Can also be written as:

```
SELECT super_eno FROM employee
UNION
SELECT mgr_eno FROM department;
```

Note: DISTINCT is implicit in SET operations in SQL

JOIN is basically INTERSECTION problem

JOIN can be understood as: join produces "joined" tuples of "common tuples" in operand relations; and "commonness" is checked based on "join condition".

For example, consider this join.

$$student \overset{\bowtie}{\underset{progid = pid}{}} program$$

Let us understand it as following: Find tuples that are common in both students and program, combine, and produce. Is not it an adaptation of "INTERSECTION"?

Semi JOIN or Semi Intersection [SQL IN]

Semi Intersection

Motivating Example: Consider query "**Customers who have at least one Order**"

Let us say following is schema of customer and order relations

```
CUSTOMER(custid, name, city, state, pin, email)
ORDER(orderno, orderdate, custid)
```

Orders		
OrdrNo	OrdrDate	CustNo
1	30-06-2010	5
2	05-07-2011	4
3	26-07-2011	3
4	21-08-2011	2
5	23-08-2011	5
6	23-08-2011	1

Customers	
CustNo	Name
1	John
2	Smith
3	Allen
4	Russel
5	Harry

It can be solved as "intersection of Customers and Orders"

But the problem is Customer and Orders are not Union Compatible.

Therefore, let us attempt defining a new operation "Customers Matching with Orders"

$$customer \ \frac{MATCHING}{mactching\ cond\ is\ (c.custid = o.custid)}\ order$$

Let us call it semi intersection

$$customer \ \frac{SEMI\_INTERSECTION}{check\_condition\ (c.custid = o.custid)}\ order$$

Algebraically, it can be expressed as following

$$customer * \left(\pi_{custid}(customer) \cap \pi_{custid}(order)\right)$$

Note that this also same as: $\pi_{c.*}(customer * order)$

Therefore, Semi Intersection is also called as **Semi Join**. And Semi-Join is more popularly used in the literature.

General expressions:

$$r1 \ \frac{SEMI\_INTERSECTION}{r1.A = r2.B}\ r2$$

where A and B are same size attribute sets.

It is algebraically equivalent to following

$$r1 * \left(\pi_A(r1) \cap \pi_B(r2)\right)$$

where * stands for Natural Join.

Above SEMI INTERSECTION is also algebraically equivalent to following

$$\pi_{r1.*}(r1 \ \frac{\bowtie}{r1.A=r2.B}\ r2)$$

This means we are performing join but are interested in values of left relation r1 only.

Semi Intersection/Join in SQL

While sounds all this complicated in algebra. SQL implements it very simply.

It provides IN keyword for performing semi intersection.

Above algebraic SEMI INTERSECTION or SEMI JOIN can be expressed as

```
SELECT * FROM r1 WHERE A IN (SELECT B FROM r2);
```

And in generalized form, we can express it as following

```
SELECT * FROM r1
WHERE A1, A2, … Am IN (SELECT B1, B2, .. Bm FROM r2);
```

For our example above we express it as following:

```
SELECT * FROM customer WHERE custid IN
    (SELECT custid FROM order);
```

Note that the same can also be expressed as following:

```
SELECT c.* FROM customer AS c NATURAL JOIN order;
```

And

```
SELECT * FROM customer NATURAL JOIN
     (SELECT custid FROM customer
      INTERSECT
      SELECT custid FROM order) AS r1;;
```

Example ##:

Compute employees (i.e., employee.*) that are both (manager of some department and supervisor of some employee)

#using SET INTERSECTION

$$r1 \leftarrow \pi_{supereno}(EMPLOYEE) \cap \pi_{mgreno}(DEPARTMENT)$$

$$result \leftarrow \pi_{e.*}\left(r1 \overset{\bowtie}{\underset{e.eno = r1.supereno}{}} EMPLOYEE\ e\right)$$

SQL:

```
SELECT e.* FROM employee AS e JOIN
     (SELECT super_eno FROM employee
      INTERSECT
      SELECT mgr_eno FROM department) AS r1
   ON r1.super_eno=e.eno;
```

Above can also be alternatively written as

$$r1(eno) \leftarrow \pi_{supereno}(EMPLOYEE) \cap \pi_{mgreno}(DEPARTMENT)$$

$$result \leftarrow r1 * EMPLOYEE$$

SQL:

```
SELECT * FROM employee NATURAL JOIN
     (SELECT super_eno AS eno FROM employee
      INTERSECT
      SELECT mgr_eno AS eno FROM department) AS r1;
```

#using SEMI JOIN/INTERSECTION (Algebra)

$$r1 \leftarrow EMPLOYEE_e \frac{SEMI\_INTERSECTION}{e.supereno = d.mgreno} DEPARTMENT_d$$

$$result \leftarrow EMPLOYEE_e \frac{SEMI\_INTERSECTION}{e.supereno = r1.eno} r1$$

SQL (Using IN)

```
SELECT * FROM employee WHERE eno IN
    (SELECT super_eno FROM employee
      WHERE super_eno IN (SELECT mgr_eno FROM department));
```

OR, bit mixed (INTERSECTION followed by SEMI-INTERSECTION)

$$r1 \leftarrow \pi_{supereno}(EMPLOYEE) \cap \pi_{mgreno}(DEPARTMENT)$$

$$result \leftarrow EMPLOYEE_e \frac{SEMI\_INTERSECTION}{e.eno = r1.supereno} r1$$

SQL:

```
SELECT * FROM employee WHERE eno IN
    (SELECT super_eno FROM employee
     INTERSECT SELECT mgr_eno FROM department);
```

Exercise ##:

Compute “employees (i.e., employee.*) that work on some project”

$$result \leftarrow EMPLOYEE_e \frac{SEMI\_INTERSECT}{e.eno = w.eno} WORKS\_ON_w$$

SQL (Using IN)

```
SELECT * FROM employee WHERE eno IN
    (SELECT eno_eno FROM works_on);
```

Semi DIFERENCE [SQL NOT IN]

Motivating Example: Consider query "**Customers who have not purchased any item**"

It can be solved as "set difference of Customers and Orders"

But again the problem is Customer and Orders are not Union Compatible.

Therefore, let us attempt defining a new operation "Customers Matching with Orders"

$$customer \ \frac{NOT\ MATCHING}{mactching\ cond\ is\ (c.custid = o.custid)}\ order$$

Let us call it semi intersection

$$customer \ \frac{SEMI\_DIFFERENCE}{check\_condition\ (c.custid = o.custid)}\ order$$

Algebraically, it can be expressed as following

$$customer * \left(\pi_{custid}(customer) - \pi_{custid}(order)\right)$$

General expressions:

$$r1 \ \frac{SEMI\_DIFFERENCE}{r1.A = r2.B}\ r2$$

where A and B are same size attribute sets.

It is algebraically equivalent to following

$$r1 * \left(\pi_A(r1) - \pi_B(r2)\right)$$

where * stands for Natural Join.

Semi DIFFERENCE in SQL

Expressed by `NOT IN`

In generalized form, we can express it as following:

```
SELECT * FROM r1
WHERE A1, A2, … Am IN (SELECT B1, B2, .. Bm FROM r2);
```

For our example here, we express it as following:

```
SELECT * FROM customer WHERE custid NOT IN
    (SELECT custid FROM order);
```

Note that the same can also be expressed as following:

```
SELECT * FROM customer NATURAL JOIN
     (SELECT custid FROM customer
      EXCEPT
      SELECT custid FROM order) AS r1;;
```

Example

Compute employees (i.e., employee.*) that are not managers

#using SET DIFFERENCE

$$r1 \leftarrow \pi_{eno}(EMPLOYEE) - \pi_{mgreno}DEPARTMENT$$

$$result \leftarrow EMPLOYEE * r1$$

SQL:

```
SELECT * FROM employee AS e NATURAL JOIN
     (SELECT eno FROM employee
      EXCEPT
      SELECT mgr_eno FROM department) AS r1;
```

#using SEMI DIFFERENCE

$$result \leftarrow EMPLOYEE_e \frac{SEMI\_DIFFERENCE}{e.eno = d.mgreno} DEPARTMENT_d$$

SQL (using `NOT IN`)

```
SELECT * FROM employee WHERE eno NOT IN
    (SELECT mgr_eno FROM department);
```

Exercise ##:

Compute employees (i.e., employee.*) that are supervisors (supervises some employees) but not managers (that are manager of any department)