Relations – Querying .2

pm jat @ daiict

JOIN Operation in terms of CROSS PRODUCT

- In theory JOIN operation is also explained through CROSS JOIN; where,
- "CROSS JOIN" is CROSS PRODUCT of operand relations

$$r \bowtie_{<join-cond>} s = \sigma_{<join-cond>}(r \times s)$$

CROSS JOIN

S

StudentID	Name	ProgID	CPI
101	Rahul	BCS	7.5
102	Vikash	BIT	8.6
103	Shally	BEE	5.4
104	Alka	BIT	6.8
105	Ravi	BCS	6.5

t1

- Given Student and Program relations,
- Try computing

  STUDENT × PROGRAM

PID	ProgName	Intake	DID
BCS	BTech(CS)	40	CS
BIT	BTech(IT)	30	CS
BEE	BTech(EE)	40	EE
BME	BTech(ME)	40	ME

t2

for t1 ∈ S
for t2 ∈ P
produce t1 + t2
t ≡
append t to resultset

CROSS JOIN

student × program

studid charact	name character varying(20)	progid characte	cpi numeri	pid charac	pname character vary	intake smallin	did chara
101	Rahul	BCS	8.70	BCS	BTech(CS)	30	CS
102	Vikash	BEC	6.80	BCS	BTech(CS)	30	CS
103	Shally	BEE	7.40	BCS	BTech(CS)	30	CS
104	Alka	BEC	7.90	BCS	BTech(CS)	30	CS
105	Ravi	BCS	9.30	BCS	BTech(CS)	30	CS
101	Rahul	BCS	8.70	BEC	BTech(ECE)	40	EE
102	Vikash	BEC	6.80	BEC	BTech(ECE)	40	EE
103	Shally	BEE	7.40	BEC	BTech(ECE)	40	EE
104	Alka	BEC	7.90	BEC	BTech(ECE)	40	EE
105	Ravi	BCS	9.30	BEC	BTech(ECE)	40	EE
101	Rahul	BCS	8.70	BEE	BTech(EE)	40	EE
102	Vikash	BEC	6.80	BEE	BTech(EE)	40	EE
103	Shally	BEE	7.40	BEE	BTech(EE)	40	EE
104	Alka	BEC	7.90	BEE	BTech(EE)	40	EE
105	Ravi	BCS	9.30	BEE	BTech(EE)	40	EE
101	Rahul	BCS	8.70	BME	BTech(ME)	40	ME
102	Vikash	BEC	6.80	BME	BTech(ME)	40	ME
103	Shally	BEE	7.40	BME	BTech(ME)	40	ME
104	Alka	BEC	7.90	BME	BTech(ME)	40	ME
105	Ravi	BCS	9.30	BME	BTech(ME)	40	ME

select * FROM student CROSS JOIN program;

Try interpreting content of each tuple of CROSS JOIN? And compare highlighted tuple with tuples in JOIN result

SELECT * FROM student
CROSS JOIN program

charact	character varying(20)	characte	cpi numeri	pid charac	pname character vary	intake smallin	did chara
101	Rahul	BCS	8.70	BCS	BTech(CS)	30	CS
102	Vikash	BEC	6.80	BCS	BTech(CS)	30	CS
103	Shally	BEE	7.40	BCS	BTech(CS)	30	CS
104	Alka	BEC	7.90	BCS	BTech(CS)	30	CS
105	Ravi	BCS	9.30	BCS	BTech(CS)	30	CS
101	Rahul	BCS	8.70	BEC	BTech(ECE)	40	EE
102	Vikash	BEC	6.80	BEC	BTech(ECE)	40	EE
103	Shally	BEE	7.40	BEC	BTech(ECE)	40	EE
104	Alka	BEC	7.90	BEC	BTech(ECE)	40	EE
105	Ravi	BCS	9.30	BEC	BTech(ECE)	40	EE
101	Rahul	BCS	8.70	BEE	BTech(EE)	40	EE
102	Vikash	BEC	6.80	BEE	BTech(EE)	40	EE
103	Shally	BEE	7.40	BEE	BTech(EE)	40	EE
104	Alka	BEC	7.90	BEE	BTech(EE)	40	EE
105	Ravi	BCS	9.30	BEE	BTech(EE)	40	EE
101	Rahul	BCS	8.70	BME	BTech(ME)	40	ME
102	Vikash	BEC	6.80	BME	BTech(ME)	40	ME
103	Shally	BEE	7.40	BME	BTech(ME)	40	ME
104	Alka	BEC	7.90	BME	BTech(ME)	40	ME
105	Ravi	BCS	9.30	BME	BTech(ME)	40	ME

JOIN and CROSS JOIN (PRODUCT)

- Note that tuples where progid and pid match only represents correct fact set of values.
- It should be easy to establish following:

$\sigma_{<progid=pid>}$(student x program) is equal to
student $\bowtie_{<progid=pid>}$ program

CROSS JOIN

- Following pseudo algorithm for CROSS PRODUCT or CROSS JOIN

  r × s -

```
result_set = NULL;
for each tuple t1 in relation r1
   for each tuple t2 in relation r2
       form new tuple t = t1 + t2
       append t to result_set
```

- Below is how CROSS PRODUCT written in SQL-

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

or

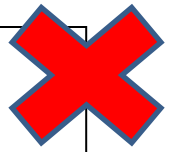
```
SELECT * FROM student, program;
```

Do Not perform JOIN through CROSS Product in SQL!

- When you want to perform JOIN in SQL, use JOIN keyword.
- Even though following two are algebraically same. You should be using second style.

$\sigma_{p.pid=s.pid}(S \times P)$

```
SELECT * FROM program, department
      WHERE program.did = department.did;
```



$S \bowtie_{pid=pid} P$

```
SELECT * FROM program JOIN department
      ON (program.did = department.did);
```

Other (names/types of) Joins

- Recall that join is expressed as $r \bowtie_{\text{<join-cond>}} s$
- Note that Degree of result will be
  degree of R + degree of S, and cardinality of result can be anything between zero rows in r times rows in s.
- Following are the names/references are found in the literature for JOIN.
- Classically this is general form of JOIN and referred as **Theta Join**.
- If Join condition only includes equality check (that is almost the case) then it is called as **Equi-Join**.

JOIN

- Out of following SQL query shown below

```
SELECT * FROM student AS s JOIN
        program AS p ON (s.progid=p.pid)
```

studid charact	name character varying(20)	progid characte	cpi numeri	pid charac	pname character vary	intake smallin	did chara
101	Rahul	BCS	8.70	BCS	BTech(CS)	30	CS
102	Vikash	BEC	6.80	BEC	BTech(ECE)	40	EE
103	Shally	BEE	7.40	BEE	BTech(EE)	40	EE
104	Alka	BEC	7.90	BEC	BTech(ECE)	40	EE
105	Ravi	BCS	9.30	BCS	BTech(CS)	30	CS

- Note the double appearance of column **progid** in join result here.

Natural Join

- Relational model defines NATURAL JOIN that has implicit join condition, that is equality of all common attributes. Below is an SQL example for this.
- This also drops duplicate columns

equality on all common attribut

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

```
pmjat=# SELECT * FROM program NATURAL JOIN department;
 did | pid |    pname     | intake |          dname
-----+-----+--------------+--------+-------------------------
 CS  | BCS | BTech(CS)    |     30 | Computer Engineering
 EE  | BEC | BTech(ECE)   |     40 | Electrical Engineering
 EE  | BEE | BTech(EE)    |     40 | Electrical Engineering
 ME  | BME | BTech(ME)    |     40 | Mechanical Engineering
(4 rows)
```

Natural Join

- Note that SQL Natural Join requires having same name of attributes in both the relations
- However, in practice, relations may not have same name for FK-PK pairs; for example, consider following-
  - progid (in student) and pid (in program), and
  - mgrssn (in department) and ssn (in employee)

  Therefore, we may not always able to use natural join keyword in SQL
- There are often reasons of choosing different names for semantically same attributes. What?

Natural Join - example

- Following will not give correct result ?

  SELECT * FROM student NATURAL JOIN program;

- There are no common attributes, therefore it results into a cross product?

- You need to get back to JOIN –

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

Operations on relations

- SELECTION: $\sigma_{\text{<selection condition>}}(r)$
- PROJECTION: $\pi_{\text{<attribute list>}}(r)$
- JOIN: $r \bowtie_{\text{<join-cond>}} s$
- NATURAL JOIN: $r * s$
  Requires that r and s have a common set of attribute – normally FK-CK pair

Theta JOIN

- There are various types of JOINS
  - Theta JOIN,
  - Equi JOINs
  - Natural JOIN
  - INNER JOIN, OUTER JOIN (LEFT, RIGHT, FULL)
- In theory JOIN can have any condition, and such a join is called there join
- Alone JOIN may refer to Theta JOIN or Natural JOIN

Equi JOIN

- When JOIN condition is only equality

Equi (Theta) JOIN and Natural Join

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 r`

b	a	c	d
b1	a1	c1	d1

Types of JOIN?

- Theta JOIN: can have any boolean expressions in join condition (rarely used)
- EQUI-JOIN: has only equality condition (commonly used)
- Natural JOIN: implicit equality condition on common attributes (and drops duplicate columns)
- INNER JOIN
- OUTER JOIN: LEFT, RIGHT, and FULL

INNER and OUTER JOIN

- Theta Join and Natural Join are inner joins.

- Following result same relation

```
select * from student JOIN program
  ON (progid = pid);
select * from student INNER JOIN program
  ON (progid = pid);
```

INNER and OUTER JOIN

- INNER JOIN does not join, when
  - Tuples from operand relations do not agree on JOIN-Condition, or
  - Null is found in any of joining attributed
- OUTER join still performs join such cases (in above situations).
- There are three types of OUTER JOIN: LEFT, RIGHT, and FULL based on the way non-matching tuples (and NULLs) are joined and included in the result-set.

Recall- Logical Algo of INNER JOIN

- Iterate through tuples of one relation, and look into other relation for matched tuples, and JOIN wherever there is a match.

```
result_set = NULL;
for each tuple t1 in relation r1
  for each tuple t2 in relation r2
    if join-cond met then
       form new tuple t = t1 + t2
       append t to result_set
```

Logical Algo for **LEFT** OUTER **JOIN**

- Perform JOIN for all tuples from LEFT operand, whether match or no match.

```
result_set = NULL;
for each tuple t1 in relation r1
   for each tuple t2 in relation r2
      if join-cond met then
         form new tuple t = t1 + t2
         append t to result_set
for each tuple t1 in relation r1
   if t1 NOT IN (result_set)
       form new tuple t = t1 + <null>
       append t to result_set
```

Logical Algo-2 for **LEFT** OUTER **JOIN**

- Perform JOIN for all tuples from LEFT operand, whether match or no match.

```
result_set = NULL;
for each tuple t1 in relation r1
  if attribs(left-relation) has NULL then
     form new tuple t = t1 + <null>
     append t to result_set
  match=false
  for each tuple t2 in relation r2
    if join-cond met then
       form new tuple t = t1 + t2
       append t to result_set
       match=true
  if not match then
       form new tuple t = t1 + <null>
       append t to result_set
```

Example **INNER JOIN** and **LEFT JOIN**

SELECT * FROM employee AS e [INNER] **JOIN** employee AS s ON(e.superssn=s.ssn);

ename character v	ssn integ	bdate date	gen cha	salary numeric(	super intege	dno smal	ename character va	ssn integer	bdate date	gen chai	salary numeric(	superssn integer	dno small
Franklin	102	1945-	M	40000	105	5	James	105	1927-	M	55000		1
Jennifer	106	1931-	F	43000	105	4	James	105	1927-	M	55000		1
John	101	1955-	M	30000	102	5	Franklin	102	1945-	M	40000	105	5
Alicia	108	1958-	F	25000	106	4	Jennifer	106	1931-	F	43000	105	4
Ramesh	104	1952-	M	38000	102	5	Franklin	102	1945-	M	40000	105	5
Joyce	103	1962-	F	25000	102	5	Franklin	102	1945-	M	40000	105	5
Ahmad	107	1959-	M	25000	106	4	Jennifer	106	1931-	F	43000	105	4

SELECT * FROM employee AS e **LEFT JOIN** employee AS s ON(e.superssn=s.ssn);

ename character v	ssn integ	bdate date	gen cha	salary numeric(	super intege	dno smal	ename character va	ssn integer	bdate date	gen chai	salary numeric(	superssn integer	dno small
James	105	1927-	M	55000		1							
Franklin	102	1945-	M	40000	105	5	James	105	1927-	M	55000		1
Jennifer	106	1931-	F	43000	105	4	James	105	1927-	M	55000		1
John	101	1955-	M	30000	102	5	Franklin	102	1945-	M	40000	105	5
Alicia	108	1958-	F	25000	106	4	Jennifer	106	1931-	F	43000	105	4
Ramesh	104	1952-	M	38000	102	5	Franklin	102	1945-	M	40000	105	5
Joyce	103	1962-	F	25000	102	5	Franklin	102	1945-	M	40000	105	5
Ahmad	107	1959-	M	25000	106	4	Jennifer	106	1931-	F	43000	105	4

Logical Algo for **RIGHT** OUTER **JOIN**

- Perform JOIN for all tuples from LEFT operand, whether match or no match.

```
result_set = NULL;
for each tuple t1 in relation r1
   for each tuple t2 in relation r2
      if join-cond met then
         form new tuple t = t1 + t2
         append t to result_set
for each tuple t2 in relation r2
   if t2 NOT IN (result_set)
       form new tuple t = <null> + t2
       append t to result_set
```

Logical Algo of FULL [OUTER] JOIN

- UNION of result of LEFT JOIN and RIGHT JOIN

- Any of the algo used of LEFT or RIGHT can be used, and remaining rows from other side are also included with null values in corresponding attributes

Logical Algo for **FULL** OUTER **JOIN**

```
result_set = NULL;
for each tuple t1 in relation r1
   for each tuple t2 in relation r2
      if join-cond met then
         form new tuple t = t1 + t2
         append t to result_set
for each tuple t1 in relation r1
   if t1 NOT IN (result_set)
       form new tuple t = t1 + <null>
       append t to result_set
for each tuple t2 in relation r2
   if t2 NOT IN (result_set)
       form new tuple t = <null> + t2
       append t to result_set
```

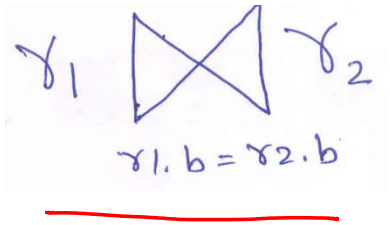
JOINS: INNER, LEFT, RIGHT, and FULL

r1

a	b	c
a1	b1	c1
a2	(Null)	c2

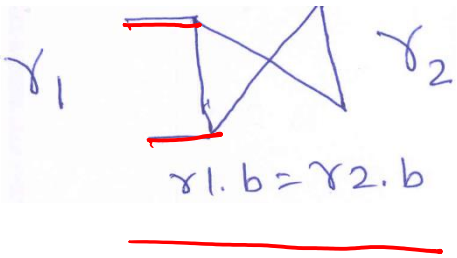
r2

b	d
b1	d1
b2	d2



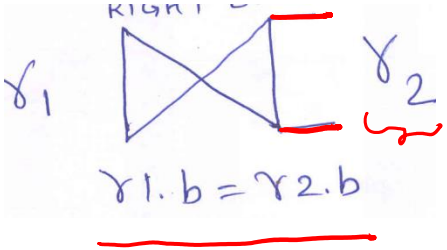
[INNER] JOIN

a	b	c	b1	d
a1	b1	c1	b1	d1



LEFT [OUTER] JOIN

a	b	c	b1	d
a1	b1	c1	b1	d1
a2	(Null)	c2	(Null)	(Null)



RIGHT [OUTER] JOIN

a	b	c	b1	d
a1	b1	c1	b1	d1
(Null)	(Null)	(Null)	b2	d2

JOINS: INNER, LEFT, RIGHT, and FULL

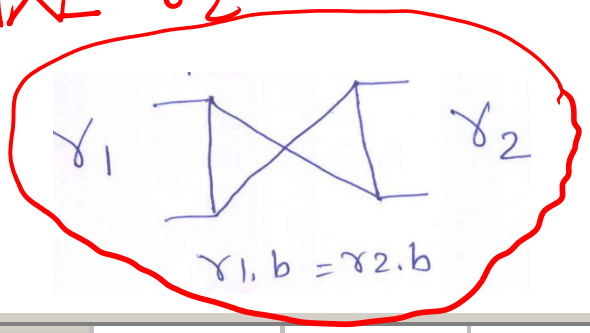
$r_1 \bowtie r_2 = r_1 \bowtie r_2$ UNION $r_1 \bowtie r_2$

r1

a	b	c
a1	b1	c1
a2	(Null)	c2

r2

b	d
b1	d1
b2	d2



a	b	c	b1	d
a1	b1	c1	b1	d1
a2	(Null)	c2	(Null)	(Null)
(Null)	(Null)	(Null)	b2	d2

FULL [OUTER] JOIN

FULL JOIN is equivalent to UNION of LEFT and RIGHT JOIN

JOINs in SQL: INNER, LEFT, RIGHT, and FULL

- INNER:
  `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

Example

list EmpName, Salary, DName

- List of supervisors and manager employees (ename, salary)

$\gamma_1 \leftarrow \pi_{Super\text{-}ENO}(EMP)$

$\gamma_2 \leftarrow \pi_{mgr\_eno}(DEP)$

$\gamma \leftarrow \gamma_1 \cup \gamma_2$

$\gamma_3 \leftarrow \gamma \bowtie EMP$

$\gamma.Super.eno = ENO$

$\pi_{ename, salary, \cancel{dname}}(\gamma_3)$

Example

- List of supervisors and manager employees (ename, salary)

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

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

SET Operations

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

Requirement:
UNION Type Compatibility between operands for these operations

Type Compatibility for Set operations

- 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 ∪ r2 has the same attribute names as the *first* operand relation R1

- This applies to Intersection and subtraction as well

Example – SET operations

(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

Courtesy: Elmasri/Navathe

UNION, INTERSECT, MINUS in SQL

- SQL has keywords -
  - UNION for union
  - INTERSECT for intersection
  - EXCEPT for minus

- Syntax:

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

Examples

- 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

NATURAL JOIN and INTERSECTION

- MATURAL JOIN is basically a INTESECTION problem?
- EMP NATURAL JOIN DEP

  ==> take INTERSECTION of both sets by checking only dno in both sets and combine the tuples

UNION, INTERSECT, MINUS in SQL

- Because of type compatibility, use of these operations directly have limited use in practice
- In most cases UNION can be performed by having OR in tuple SELECTION criteria (in WHERE Clause of SQL)
- INTERSECT is accomplished by NATURAL JOIN or SEMI JOIN (IN in SQL)
- EXCEPT could be accomplished by SEMI Difference (NOT IN of SQL)
- DISTINCT is implied in SET operations in SQL