Operations on Relations [01]

pm @ daiict

Operations on Relations

- Two types of operations
- **Updating Relations**
  – Add tuple(s) to a relations [INSERT]
  – Modify data of tuple(s) of a relation [UPDATE]
  – Delete tuple(s) from a relation [DELETE]
- **Answering queries** by retrieving data from one or more relations
- Operand is “relation instance” for manipulation operations.

Querying relations

- Queries are answered by applying various relational operations on relations. Queries are written as expression in terms relational operators and relations as operands, and result is returned as a relation

- There are languages for writing relational expressions - Relational Algebra, Relational Calculus, SQL

Basic Operations on Relations

- Let us first discuss what is called as "the original operators" –
  - SELECTion
  - PROJECTion
  - Join
  - Cross JOIN
  - Set Operations: UNION, INTERSECTION, and DIFFERENCE
  - Aggregate operations
- Note: Update operations were not part of original relational model

SELECT Operation

- Some people use term “RESTRICT” for SELECT – possibly to avoid conflict with SELECT in SQL - both are different.
- SELECT operation is used to select a *subset* of the tuples from a relation that satisfy a **selection condition**.
- Notation sigma (σ) is used as operator for this operation in relational algebra

SELECT Operation

- In general, the SELECT operation is denoted by

  $\sigma_{\texttt{<tuple selection criteria>}}(\texttt{r})$

  where selection condition is a boolean expression specified on the attributes of relation **r**

- <u>Schema of resultant relation</u>: Same as of operand relation r

- <u>Tuples in resultant relation</u>: set of tuples from r that meet the tuple selection criteria; will be some subset of operand relation. Number of tuples in result can be anything between 0 to n, where **n** is number of tuples in operand relation.

SELECT Operation - Example

- Query: To select the EMPLOYEE tuples whose department number is four.
- Following expression is used to write query in relational algebra-

  $\sigma_{\text{DNO = 4}}$ (EMPLOYEE)
- For Query: To select the EMPLOYEE tuples whose salary is greater than $30,000; We would write-

  $\sigma_{\text{SALARY > 30000}}$ (EMPLOYEE)

PROJECT Operation

- This operation selects certain “columns” of a relation
- Can be viewed as getting vertical subset of a relation
- Relation Algebra uses pi (π) operator for this operation.
- The general form of the project operation is $\pi_{\texttt{<attribute list>}}(\texttt{R})$ where <attribute list> is the desired list of attributes from the attributes of relation R.

PROJECT Operation - Example

- Query: To view each employee's first name, last name, and salary, the following is used:
  $\pi_{\text{LNAME, FNAME, SALARY}}(\text{EMPLOYEE})$

- Result of Project Operation will be –
  - Will be a valid relation – no duplicates (not so in SQL)
  - Will have cardinality same as of the instance of EMPLOYEE (and less, if projection operation has to drop some duplicate tuples in result set)
  - Will have degree equal to number of attributes specified in the list

More examples of Select and Project

$$\sigma_{(DNO=4\ AND\ SALARY>25000)\ OR\ (DNO=5\ AND\ SALARY\ >\ 30000)}(EMP)$$

$$\pi_{FNAME,\ LNAME,\ SALARY}(EMP)$$

$$\pi_{SEX,\ SALARY}(EMP)$$

SELECT and PROJECT operations in SQL

SELECT and PROJECT in SQL - examples

- SELECTion:
```
SELECT * FROM employee
WHERE dno = 5;
```

- PROJECTion:
```
SELECT ssn, fname, lname, salary
   FROM employee;
```

- SELECTion and PROJECTion:
```
SELECT ssn, fname, lname, salary
   FROM employee
   WHERE dno = 5;
```

SQL PROJECT may give duplicate tuples

- SQL PROJECT may give duplicate tuples – not a valid relation. For example, following *query* may not give distinct set of tuples

```
SELECT dno FROM employee;
```

- In order to have distinct tuples, we need to use DISTINCT keyword-

```
SELECT DISTINCT dno FROM employee;
```

SELECT and PROJECT in SQL - examples

- You can order the result (multi) set-

```
SELECT ssn, fname, lname, salary
    FROM employee
    WHERE dno = 5;
    ORDER BY ssn;
```

- Note: There is no ordering operation in Relational Algebra.

Properties of SELECT operation

- The SELECT operation σ <selection condition> (R) produces a relation r(S) that has the same schema as R
- The SELECT operation σ is commutative; i.e.,
  σ<cond1>(σ<cond2> (R)) = σ<cond2>(σ<cond1>(R))
- A cascaded SELECT operation may be applied in any order; i.e.,
  σ<cond1>(σ<cond2>(σ<cond3>(R))
  = σ<cond2>(σ<cond3>(σ<cond1>(R)))
- A cascaded SELECT operation may be replaced by a single selection with a conjunction of all the conditions; i.e.,
  σ<cond1>(σ<cond2>(σ<cond3>(R))
  = σ<cond1> AND <cond2> AND <cond3>(R)))

Properties of PROJECT Operation

- The number of tuples in the result of projection $\pi_{\text{<list>}}(R)$ is always less or equal to the number of tuples in R.
- If the list of attributes includes a key of R, then the number of tuples is equal to the number of tuples in R.
- $\pi_{\text{<list1>}}(\pi_{\text{<list2>}}(R)) = \pi_{\text{<list1>}}(R)$
  as long as <list1> contains the attributes from <list2>

JOIN operations

JOIN Operation

- Consider XIT schema, and want to answer a query, give name of program in which student "Ankit" studies.
- Done through JOIN.
- Binary Operator. Join condition is additional parameter.
- General Syntax: $r \bowtie_{\text{<join-cond>}} s$

JOIN operation

- Following pseudo algorithm should help in understanding the meaning of expression $\mathbf{r} \bowtie_{\texttt{<join-cond>}} \mathbf{s}$ -

```
Result set rs = 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 rs
```

Example JOIN

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

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

- Given STUDENT and PROGRAM relations, following JOIN **student** $\bowtie_{\text{<progid=pid>}}$ **program** results into following relation-

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

JOIN in SQL

- The join $\mathbf{r} \bowtie_{\text{<join-cond>}} \mathbf{s}$ in relational algebra will be written in SQL as following-

SELECT * FROM r JOIN s ON (<join-cond>)

- <u>Example</u> JOIN $\mathbf{student} \bowtie_{\text{<progid=pid>}} \mathbf{program}$ is expressed as following in SQL

SELECT * FROM STUDENT JOIN PROGRAM ON (progid=pid)

CROSS PRODUCT

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

CROSS PRODUCT

- Following pseudo algorithm for CROSS PRODUCT $r \times s$ -

```
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;`

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 PRODUCT

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

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

- In general terms, we can say that
  $\sigma_{\texttt{<join-cond>}}(\texttt{r x s}) = \texttt{r} \bowtie_{\texttt{<join-cond>}} \texttt{s}$

Simplified **Company** schema

EMPLOYEE (ssn, ename, bdate, dno, gender, superssn)
Foreign Keys: dno REFERENCES department (dno),
Foreign Key: superssn REFERENCES employee (ssn)

DEPARTMENT (dno, dname, mgrssn, mgrstartdate)
Foreign Keys: mgrssn REFERENCES employee (essn)

DEP_LOCATIONS (dno, dlocation)
Foreign Keys: dno REFERENCES department (dno),

PROJECT (pno, pname, plocation, dno)
Foreign Keys: dno REFERENCES department (dno),

WORKS_ON (essn, pno, hours)
Foreign Keys: essn REFERENCES employee (essn)
Foreign Keys: pno REFERENCES project (pno)

DEPENDENT (essn, dep_name, gender, bdate date, relationship)
Foreign Keys: essn REFERENCES employee (essn)

Some examples

Interpret tuples in results of two JOINS on EMPLOYEE and DEPARTMENT

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

ename character var	ssn integer	bdate date	gender characte	salary numeric(	superssn integer	dno small	dname character varying(20)	dno smallint	mgrssn integer	mgrstar date
James	105	1927-	M	55000		1	Headquater	1	105	1971-06
Franklin	102	1945-	M	40000	105	5	Research	5	102	1978-05
Jennifer	106	1931-	F	43000	105	4	Administration	4	106	1985-01
John	101	1955-	M	30000	102	5	Research	5	102	1978-05
Alicia	108	1958-	F	25000	106	4	Administration	4	106	1985-01
Ramesh	104	1952-	M	38000	102	5	Research	5	102	1978-05
Joyce	103	1962-	F	25000	102	5	Research	5	102	1978-05
Ahmad	107	1959-	M	25000	106	4	Administration	4	106	1985-01

SELECT * FROM employee AS e JOIN department AS d ON(e.ssn=d.mgrssn);

ename character varyin	ssn integ	bdate date	gen cha	salary numeric(	super intege	dno small	dname character varying(20)	dno small	mgrssn integer	mgrstartdate date
Franklin	102	1945-	M	40000	105	5	Research	5	102	1978-05-22
Jennifer	106	1931-	F	43000	105	4	Administration	4	106	1985-01-01
James	105	1927-	M	55000		1	Headquater	1	105	1971-06-19

Interpret tuples in result of following JOIN

SELECT * FROM employee AS e 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 smalli
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

Interpret tuples in results following JOIN

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

ename character v	ssn integ	bdate date	gen cha	salary numeric(	super integ	dno smal	ename character va	ssn integer	bdate date	gen cha	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

Interpret tuples in results of following JOIN

STUDENT $\bowtie_{<progid=did>}$ PROGRAM $\bowtie_{<p.did=d.did>}$ DEPARTMENT

```
select * from student JOIN program as p ON (progid=pid)
JOIN department as d ON p.did=d.did;
```

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

Other (names/types of) Joins

- Recall that join is expressed as $r \bowtie_{\texttt{<join-cond>}} s$
- 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 - Why do we need ?

- Normally database contains *normalized* relations, and in order to answer a query we may need to collect data from multiple relations

- In example on previous slide, we wanted to list program-name, its department-name. But these attributes are in two different relations (Why in two different relations?) for such situations, where we need to combine data from multiple relations, we need joins.

Good Practice to follow 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.

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

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

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

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

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 natural join r2;`

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
```

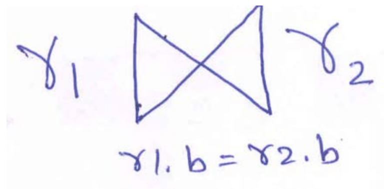
JOINS: INNER, LEFT, RIGHT, and FULL

r1

a	b	c
a1	b1	c1
a2	(Null)	c2

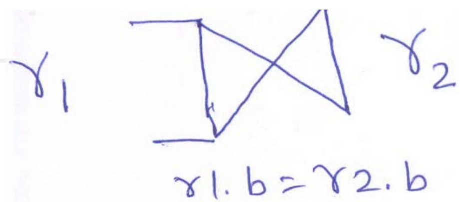
r2

b	d
b1	d1
b2	d2



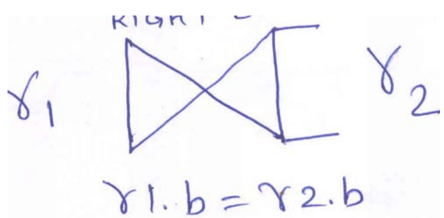
[INNER] JOIN

a	b	c	b1	d
a1	b1	c1	b1	d1



LEFT [OUTER] JOIN

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



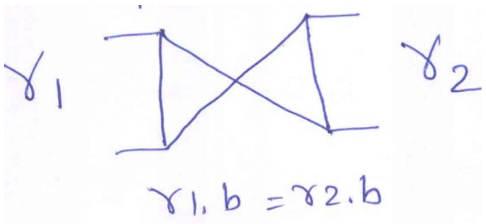
RIGHT [OUTER] JOIN

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

JOINS: INNER, LEFT, RIGHT, and FULL

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

Some Exercises!

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 ∪ B

Requirement:
UNION Type Compatibility between operands for these operations

- INTERSECT
  - A ∩ B
- EXCEPT (MINUS)
  - A - B

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 $\cup$ 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

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