Database Systems

UNIT-2
(Relational Algebra operations: Unary, Binary, Joins, Aggregate

Edited By: Er. Shilpi (AP, CSE)

Relational Algebra

- So far we have seen what a database is, what is the features of database, how to gather requirements and how to put them in ER diagrams, how to convert them into tables and their columns, set their constraints etc. Once we have database ready users will start using them. But how will they access the database? Most of the time they access the data by using some applications. These applications will communicate to database by SQL and DBMS is responsible for managing the application and SQL intact. SQL has its own querying methods to interact with database. But how these queries work in the database? These queries work similar to relational algebra that we have in mathematics. In database we have tables participating in relational algebra.

Relational Query Languages

- Relational query languages use relational algebra to break the user requests and instruct the DBMS to execute the requests.

- It is the language by which user communicates with the database. These relational query languages can be procedural or non-procedural.

Procedural Query Language

- **A procedural query language will have set of queries instructing the DBMS to perform various transactions in the sequence to meet the user request.**

- For example, *get_CGPA* procedure will have various queries to get the marks of student in each subject, calculate the total marks, and then decide the CGPA based on his total marks. This procedural query language tells the database what is required from the database and how to get them from the database. Relational algebra is a procedural query language.

Non-Procedural Query Language

- **Non-procedural queries will have single query on one or more tables to get result from the database.** For example, get the name and address of the student with particular ID will have single query on STUDENT table. Relational Calculus is a non procedural language which informs what to do with the tables, but doesn’t inform how to accomplish this.
- These query languages basically will have queries on tables in the database. In the relational database, a table is known as relation. Records / rows of the table are referred as tuples. Columns of the table are also known as attributes. All these names are used interchangeably in relational database.

Relational Algebra Definitions

- *Domain*: set of relations
- *Basic operators*: select, project, union, set difference, Cartesian (cross) product
- *Derived operators*: set intersection, division, join
- *Procedural*: Relational expression specifies query by describing an algorithm (the sequence in which operators are applied) for determining the result of an expression

Unary Relational Operations

- **SELECT Operation:** used to select a *subset* of the tuples from a relation that satisfy a **selection condition**. It is a filter that keeps only those tuples that satisfy a qualifying condition.

  Examples:

$$\sigma_{\mathbf{DNO = 4}}\ \mathbf{(EMPLOYEE)}$$

$$\sigma_{\mathbf{SALARY > 30,000}}\ \mathbf{(EMPLOYEE)}$$

  – denoted by $\sigma_{\text{<selection condition>}}(R)$ where the symbol $\boldsymbol{\sigma}$ (sigma) is used to denote the select operator, and the selection condition is a *Boolean expression* specified on the attributes of relation R

SELECT Operation Properties

The SELECT operation $\sigma_{\text{<selection condition>}}(R)$ produces a relation S that has the same schema as R

The SELECT operation σ is **commutative;** i.e.,

$$\sigma_{\text{<condition1>}}(\sigma_{\text{< condition2>}}(R)) = \sigma_{\text{<condition2>}}(\sigma_{\text{< condition1>}}(R))$$

A cascaded SELECT operation **may be applied in any order;** i.e.,

$$\sigma_{\text{<condition1>}}(\sigma_{\text{< condition2>}}(\sigma_{\text{<condition3>}}(R))$$
$$= \sigma_{\text{<condition2>}}(\sigma_{\text{< condition3>}}(\sigma_{\text{< condition1>}}(R)))$$

A cascaded SELECT operation may be replaced by a single selection with a conjunction of all the conditions; i.e.,

$$\sigma_{\text{<condition1>}}(\sigma_{\text{< condition2>}}(\sigma_{\text{<condition3>}}(R))$$
$$= \sigma_{\text{<condition1> AND < condition2> AND < condition3>}}(R)))$$

Selection Condition

- Operators: $<, \leq, \geq, >, =, \neq$
- Simple selection condition:
  - <*attribute*> *operator* <*constant*>
  - <*attribute*> *operator* <*attribute*>
  - <*condition*> AND <*condition*>
  - <*condition*> OR <*condition*>
  - NOT <*condition*>

Select Examples

Person

Id	*Name*	*Address*	*Hobby*
1123	John	123 Main	stamps
1123	John	123 Main	coins
5556	Mary	7 Lake Dr	hiking
9876	Bart	5 Pine St	stamps

$$\sigma_{\,Id>3000 \text{ OR } Hobby=\text{'hiking'}}\text{(Person)}$$

$$\sigma_{\,Id>3000 \text{ AND } Id<3999}\text{(Person)}$$

Id	*Name*	*Address*	*Hobby*
5556	Mary	7 Lake Dr	hiking
9876	Bart	5 Pine St	stamps

$$\sigma_{\,\text{NOT}(Hobby=\text{'hiking'})}\text{(Person)}$$

The SQL implementation would translate into:

σ empno=7(EMPLOYEE)
SELECT empno FROM EMPLOYEE WHERE empno=7;

σ dob<’01-Jan-1980′(EMPLOYEE)
SELECT dob FROM EMPLOYEE WHERE DOB < ’01-Jan-1980′;

Unary Relational Operations (cont.)

- **PROJECT Operation:** selects certain *columns* from the table and discards the others.

  Example:

  $$\pi_{\text{LNAME, FNAME, SALARY}}(\text{EMPLOYEE})$$

  The general form of the project operation is:

  π<attribute list>(R) where π is the symbol used to represent the project operation and <attribute list> is the desired list of attributes.

  PROJECT *removes duplicate tuples*, so the result is a set of tuples and hence a valid relation.

PROJECT Operation Properties

The number of tuples in the result of $\pi_{\text{<Attribute list>}}(R)$ is always less or equal to the number of tuples in R.

If attribute list includes a key of R, then the number of tuples is equal to the number of tuples in R.

The SQL implementation would translate into:

$\Pi_{\text{dob, empno}}$**(EMPLOYEE)**

SELECT dob, empno FROM EMPLOYEE

SELECT and PROJECT Operations

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

(b) $\pi_{\text{LNAME, FNAME, SALARY}}$ (EMPLOYEE)

(c) $\pi_{\text{SEX, SALARY}}$ (EMPLOYEE)

(a)

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
Franklin	T	Wong	333445555	1955-12-08	638 Voss,Houston,TX	M	40000	888665555	5
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry,Bellaire,TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 FireOak,Humble,TX	M	38000	333445555	5

(b)

LNAME	FNAME	SALARY
Smith	John	30000
Wong	Franklin	40000
Zelaya	Alicia	25000
Wallace	Jennifer	43000
Narayan	Ramesh	38000
English	Joyce	25000
Jabbar	Ahmad	25000
Borg	James	55000

(c)

SEX	SALARY
M	30000
M	40000
F	25000
F	43000
M	38000
M	25000
M	55000

Unary Relational Operations (cont.)

- **Rename Operation**

We may want to apply several relational algebra operations one after the other. Either we can write the operations as a single **relational algebra expression** by nesting the operations, or we can apply one operation at a time and create **intermediate result relations**. In the latter case, we must give names to the relations that hold the intermediate results.

Example: To retrieve the first name, last name, and salary of all employees who work in department number 5, we must apply a select and a project operation. We can write a single relational algebra expression as follows:

$$\pi_{\text{FNAME, LNAME, SALARY}}(\sigma_{\text{DNO=5}}(\text{EMPLOYEE}))$$

OR We can explicitly show the sequence of operations, giving a name to each intermediate relation:

$$\text{DEP5\_EMPS} \leftarrow \sigma_{\text{DNO=5}}(\text{EMPLOYEE})$$
$$\text{RESULT} \leftarrow \pi_{\text{FNAME, LNAME, SALARY}}(\text{DEP5\_EMPS})$$

Rename Operation

- choose the Date of Birth and Employee Number attributes and RENAME them as ‘Birth_Date’ and ‘Employee_Number’ from the EMPLOYEE relation…

$$\rho_{\text{(Birth\_Date, Employee\_Number)}}(\text{EMP}) \leftarrow \Pi_{\text{dob, empno}}(\text{EMPLOYEE})$$

- **In SQL we would translate the RENAME operator using the SQL ‘AS’ statement:**

  SELECT dob AS ‘Birth_Date’, empno AS ‘Employee_Number’ FROM EMPLOYEE AS EMP;

EMPNO	DOB

BIRTH_DATE	EMPLOYEE_NUMBER

Relational Algebra Operations from Set Theory

- The UNION, INTERSECTION, and MINUS Operations
- The CARTESIAN PRODUCT (or CROSS PRODUCT) Operation

Relational Algebra Operations

Two relations R_1 and R_2 are said to be **union compatible** if they have the same degree and all their attributes (correspondingly) have the same domain.

- The UNION, INTERSECTION, and SET DIFFERENCE operations are applicable on union compatible relations
- The resulting relation has the same attribute names as the first relation

The *UNION operation*

- The result of UNION operation on two relations, R_1 and R_2, is a relation, R_3, that includes all tuples that are either in R_1, or in R_2, or in both R_1 and R_2.
- The UNION operation is denoted by:

$$R_3 = R_1 \cup R_2$$

- The UNION operation eliminates duplicate tuples

FIGURE 6.3

Results of the UNION operation

RESULT1 $\cup$ RESULT2.

SELECT * FROM RESULT1

UNION

SELECT * FROM RESULT2;

RESULT1	SSN
	123456789
	333445555
	666884444
	453453453

RESULT2	SSN
	333445555
	888665555

SSN
123456789
333445555
666884444
453453453
888665555

UNION Example

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

STUDENT $\cup$ INSTRUCTOR:

Select fn, ln from student
UNION
Select fname as fn, lname as ln from instructor;

What would STUDENT $\cap$ INSTRUCTOR be?

The *INTERSECTION* operation

- The result of INTERSECTION operation on two relations, R_1 and R_2, is a relation, R_3, that includes all tuples that are in both R_1 and R_2.

- The INTERSECTION operation is denoted by:

$$R_3 = R_1 \cap R_2$$

- The both UNION and INTERSECTION operations are **commutative** and **associative** operations

INTERSECTION Example

STUDENT $\cap$ INSTRUCTOR:

Select fn, ln from student
INTERSECT
Select fname as fn, lname as ln from instructor;

The *SET DIFFERENCE* Operation

- The result of SET DIFFERENCE operation on two relations, R_1 and R_2, is a relation, R_3, that includes all tuples that are in R_1 but not in R_2.
- The SET DIFFERENCE operation is denoted by:

$$R_3 = R_1 - R_2$$

- The SET DIFFERENCE (or MINUS) operation **is not commutative**

Select fn, ln from student
MINUS
Select fname as fn, lname as ln from instructor;

(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

Relational Algebra Operations From Set Theory (cont.)

- Union and intersection are *commutative operations:*

  $$\mathbf{R \cup S = S \cup R, \text{ and } R \cap S = S \cap R}$$

- Both union and intersection can be treated as n-ary operations applicable to any number of relations as both are *associative operations;* that is

  $$\mathbf{R \cup (S \cup T) = (R \cup S) \cup T, \text{ and}}$$

  $$\mathbf{(R \cap S) \cap T = R \cap (S \cap T)}$$

- The minus operation is *not commutative;* that is, in general

  $$\mathbf{R - S \neq S - R}$$

Cartesian (Cross) Product

- If R and S are two relations, $R \times S$ is the set of all concatenated tuples $<x,y>$, where x is a tuple in R and y is a tuple in S
  - R and S need not be union compatible
- $R \times S$ is <u>expensive to compute</u>:
  - Factor of two in the size of each row; Quadratic in the number of rows

A	*B*
x1	x2
x3	x4

R

C	*D*
y1	y2
y3	y4

S

A	*B*	*C*	*D*
x1	x2	y1	y2
x1	x2	y3	y4
x3	x4	y1	y2
x3	x4	y3	y4

$R \times S$

Cartesian (Cross) Product

- A B C D A B C D
- 1 2 x 6 7 = 1 2 6 7
- 3 4 8 9 1 2 8 9

3 4 6 7

3 4 8 9

>SELECT A.dob, B.empno
from A, B;

Cartesian Product Example

- We want a list of COMPANY’s female employees dependents.

FEMALE_EMPS	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
	Alicia	J	Zelaya	999887777	1968-07-19	3321 Castle,Spring,TX	F	25000	987654321	4
	Jennifer	S	Wallace	987654321	1941-06-20	291 Berry,Bellaire,TX	F	43000	888665555	4
	Joyce	A	English	453453453	1972-07-31	5631 Rice,Houston,TX	F	25000	333445555	5

EMPNAMES	FNAME	LNAME	SSN
	Alicia	Zelaya	999887777
	Jennifer	Wallace	987654321
	Joyce	English	453453453

EMP_DEPENDENTS	FNAME	LNAME	SSN	ESSN	DEPENDENT_NAME	SEX	BDATE	• • •
	Alicia	Zelaya	999887777	333445555	Alice	F	1986-04-05	• • •
	Alicia	Zelaya	999887777	333445555	Theodore	M	1983-10-25	• • •
	Alicia	Zelaya	999887777	333445555	Joy	F	1958-05-03	• • •
	Alicia	Zelaya	999887777	987654321	Abner	M	1942-02-28	• • •
	Alicia	Zelaya	999887777	123456789	Michael	M	1988-01-04	• • •
	Alicia	Zelaya	999887777	123456789	Alice	F	1988-12-30	• • •
	Alicia	Zelaya	999887777	123456789	Elizabeth	F	1967-05-05	• • •
	Jennifer	Wallace	987654321	333445555	Alice	F	1986-04-05	• • •
	Jennifer	Wallace	987654321	333445555	Theodore	M	1983-10-25	• • •
	Jennifer	Wallace	987654321	333445555	Joy	F	1958-05-03	• • •
	Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	• • •
	Jennifer	Wallace	987654321	123456789	Michael	M	1988-01-04	• • •
	Jennifer	Wallace	987654321	123456789	Alice	F	1988-12-30	• • •
	Jennifer	Wallace	987654321	123456789	Elizabeth	F	1967-05-05	• • •
	Joyce	English	453453453	333445555	Alice	F	1986-04-05	• • •
	Joyce	English	453453453	333445555	Theodore	M	1983-10-25	• • •
	Joyce	English	453453453	333445555	Joy	F	1958-05-03	• • •
	Joyce	English	453453453	987654321	Abner	M	1942-02-28	• • •
	Joyce	English	453453453	123456789	Michael	M	1988-01-04	• • •
	Joyce	English	453453453	123456789	Alice	F	1988-12-30	• • •
	Joyce	English	453453453	123456789	Elizabeth	F	1967-05-05	• • •

ACTUAL_DEPENDENTS	FNAME	LNAME	SSN	ESSN	DEPENDENT_NAME	SEX	BDATE
	Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28

RESULT	FNAME	LNAME	DEPENDENT_NAME
	Jennifer	Wallace	Abner

The *DIVISION* Operation

- The DIVISION operation is useful for some queries. For example, “Retrieve the name of employees who work on **all** the projects that ‘John Smith’ works on.”

- The Division operation is denoted by:

$$R_3 = R_1 \div R_2$$

- In general, attributes in R_2 are a subset of attributes in R_1

Examples of Division A/B

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

A

pno
p2

B1

pno
p2
p4

B2

pno
p1
p2
p4

B3

sno
s1
s2
s3
s4

A/B1

sno
s1
s4

A/B2

sno
s1

A/B3

The DIVISION Operation

(a) Dividing SSN_PNOS by SMITH_PNOS.
(b) $T \leftarrow R \div S$.

(a)

SSN_PNOS	ESSN	PNO
	123456789	1
	123456789	2
	666884444	3
	453453453	1
	453453453	2
	333445555	2
	333445555	3
	333445555	10
	333445555	20
	999887777	30
	999887777	10
	987987987	10
	987987987	30
	987654321	30
	987654321	20
	888665555	20

SMITH_PNOS	PNO
	1
	2

SSNS	SSN
	123456789
	453453453

(b)

R	A	B
	a1	b1
	a2	b1
	a3	b1
	a4	b1
	a1	b2
	a3	b2
	a2	b3
	a3	b3
	a4	b3
	a1	b4
	a2	b4
	a3	b4

S	A
	a1
	a2
	a3

T	B
	b1
	b4

Binary Relational Operations

- An important operation for any relational database is the JOIN operation, because it enables us to combine **related tuples** from two relations into single tuple

- The JOIN operation is denoted by:

$$R_3 = R_1 \bowtie_{\text{<join condition>}} R_2$$

- The degree of resulting relation is

$$\text{degree}(R_1) + \text{degree}(R_2)$$

The *JOIN* Operation

- The difference between CARTESIAN PRODUCT and JOIN is that the resulting relation from JOIN consists only those tuples that satisfy the **join condition.**
- The JOIN operation is equivalent to CARTESIAN PRODUCT and then SELECT operation on the result of CARTESIAN PRODUCT operation, if the **select-condition** is the same as the **join condition.**
- **The JOIN operation is denoted by the ⋈ symbol and is used to compound similar tuples from two Relations into single longer tuples. Every row of the first table is joined to every row of the second table. The result is tuples taken from both tables.**
- **The general syntax would be $A \bowtie_{\text{<join condition>}} B$**

JOIN Operation

- *Cartesian product* followed by *select* is commonly used to identify and select related tuples from two relations => called **JOIN**. It is denoted by a $\bowtie$
  - This operation is important for any relational database with more than a single relation, because it allows us to process *relationships* among relations.
  - The general form of a join operation on two relations $R(A_1, A_2, \ldots, A_n)$ and $S(B_1, B_2, \ldots, B_m)$ is:

    $R \bowtie_{\text{<join condition>}} S$

    where R and S can be any relations that result from general *relational algebra expressions*.

Database State for COMPANY

EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO

DEPARTMENT

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE

DEPT_LOCATIONS

DNUMBER	DLOCATION

PROJECT

PNAME	PNUMBER	PLOCATION	DNUM

WORKS_ON

ESSN	PNO	HOURS

DEPENDENT

ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP

JOIN EXAMPLE

- SQL translation example where attribute dob is Date of Birth and empno is Employee Number:

```
SELECT A.dob, A.empno
from A
JOIN B on[where] B.empno=A.empno;
```

Types of JOIN

- THETA JOIN or NON-EQUI JOIN

  This operation results in all combinations of tuples from Relation A and Relation B satisfying a join requirement other than equal to operation: (>,<,!= etc.)

- EQUI JOIN or INNER JOIN

  If the JOIN operation has equality comparison only (that is, = operation), then it is called an EQUIJOIN operation.

- NATURAL JOIN
- SELF JOIN
- OUTER JOINS
  - LEFT OUTER JOIN or LEFT JOIN
  - RIGHT OUTER JOIN or RIGHTT JOIN
  - FULL OUTER JOIN

The *EQUIJOIN* Operation

- If the JOIN operation has equality comparison only (that is, = operation), then it is called an EQUIJOIN operation.

- In the resulting relation on an EQUIJOIN operation, we always have one or more pairs of attributes that have **identical values** in every tuples.

Example EquiJoin

Result of the JOIN operation

Dept_mgr ← Department JOIN $_{\text{Mgrssn=ssn}}$ Employee

SELECT * from DEPARTMENT as D [INNER] JOIN EMPLOYEE as E ON [WHERE] D.Mgrssn=E.ssn;

DEPT_MGR	DNAME	DNUMBER	MGRSSN	...	FNAME	MINIT	LNAME	SSN	...
	Research	5	333445555	...	Franklin	T	Wong	333445555	...
	Administration	4	987654321	...	Jennifer	S	Wallace	987654321	...
	Headquarters	1	888665555	...	James	E	Borg	888665555	...

Example EquiJoin

rollno	name	branch
101	aba	cse
102	sh	ece
103	gh	ece
104	h	cse
105	hhh	cse

<- Stuinfo

Stulib->

rollno	book
101	C
101	DS
102	DS
102	Math
105	AC
106	GG
107	DD

Result of the JOIN operation

Student [INNER] JOIN $_{\text{rollno=rollno}}$ Library

>SELECT * FROM stuinfo AS ss JOIN stulib AS ll ON ss.rollno=ll.rollno;

Resultant Table ->

rollno	name	branch	rollno	book
101	aba	cse	101	C
101	aba	cse	101	DS
102	sh	ece	102	DS
102	sh	ece	102	Math
105	hhh	cse	105	AC

The *NATURAL JOIN* Operation

- **In EQUIJOIN operation, if the two attributes in the join condition have the same name, then in the resulting relation we will have two identical columns. In order to avoid this problem, we define the NATURAL JOIN operation.**
- **The NATURAL JOIN operation returns results that does not include the JOIN attributes of the second Relation B. It is not required that attributes with the same name be mentioned. If no columns have identical names then it is same as INNER JOIN.**
- The NATURAL JOIN operation is denoted by:

$$R_3 = R_1 *_{\text{<attribute list>}} R_2$$

- In R_3 only one of the duplicate attributes from the list are kept

Example Natural Join

rollno	name	branch
101	aba	cse
102	sh	ece
103	gh	ece
104	h	cse
105	hhh	cse

<- Stuinfo

Stulib->

rollno	book
101	C
101	DS
102	DS
102	Math
105	AC
106	GG
107	DD

Result of the JOIN operation

Student NATURAL JOIN $_{\text{rollno=rollno}}$ Library

>SELECT * FROM stuinfo AS ss NATURAL JOIN stulib AS ll WHERE ss.rollno=ll.rollno;

Resultant Table ->

rollno	name	branch	book
101	aba	cse	C
101	aba	cse	DS
102	sh	ece	DS
102	sh	ece	Math
105	hhh	cse	AC

NATURAL JOIN

(a) PROJ_DEPT ← PROJECT * DEPT

>Select * from Project NATURAL JOIN Dept;

(b) DEPT_LOCS ← DEPARTMENT * DEPTLOCATIONS

>Select * from Department NATURAL JOIN Deptlocations;

(a)

PROJ_DEPT	PNAME	PNUMBER	PLOCATION	DNUM	DNAME	MGRSSN	MGRSTARTDATE
	ProductX	1	Bellaire	5	Research	333445555	1988-05-22
	ProductY	2	Sugarland	5	Research	333445555	1988-05-22
	ProductZ	3	Houston	5	Research	333445555	1988-05-22
	Computerization	10	Stafford	4	Administration	987654321	1995-01-01
	Reorganization	20	Houston	1	Headquarters	888665555	1981-06-19
	Newbenefits	30	Stafford	4	Administration	987654321	1995-01-01

(b)

DEPT_LOCS	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE	LOCATION
	Headquarters	1	888665555	1981-06-19	Houston
	Administration	4	987654321	1995-01-01	Stafford
	Research	5	333445555	1988-05-22	Bellaire
	Research	5	333445555	1988-05-22	Sugarland
	Research	5	333445555	1988-05-22	Houston

OUTER JOINs

- In NATURAL JOIN tuples without a *matching* (or *related*) tuple are eliminated from the join result. Tuples with null in the join attributes are also eliminated. This loses information.

- *Outer joins*, can be used when we want to keep all the tuples in R, all those in S, or all those in both relations
  - regardless of whether they have matching tuples in the other relation.

- The left outer join operation keeps every tuple in the *first* or *left* relation R in R $⟕$ S; if no matching tuple is found in S, then the attributes of S in the join result are "padded" with null values.

- A similar operation, right outer join, keeps every tuple in the *second* or *right* relation S in the result of R $⟖$ S.

- A third operation, *full outer join*, denoted by $⟗$ keeps all tuples in both the left and the right relations when no matching tuples are found, padding them with null values as needed.

Outer Join

R

ColA	ColB
A	1
B	2
D	3
F	4
E	5

R LEFT OUTER JOIN R.ColA = S.SColA S

A	1	A	1
D	3	D	3
E	5	E	4
B	2	-	-
F	4	-	-

S

SColA	SColB
A	1
C	2
D	3
E	4

R RIGHT OUTER JOIN R.ColA = S.SColA S

A	1	A	1
D	3	D	3
E	5	E	4
-	-	C	2

Full outer Join

R

ColA	ColB
A	1
B	2
D	3
F	4
E	5

S

SColA	SColB
A	1
C	2
D	3
E	4

R FULL OUTER JOIN S

R.ColA = S.SColA

A	1	A	1
D	3	D	3
E	5	E	4
B	2	-	-
F	4	-	-
-	-	C	2

LEFT OUTER Join

rollno	name	branch
101	aba	cse
102	sh	ece
103	gh	ece
104	h	cse
105	hhh	cse

<- Stuinfo

Stulib->

rollno	book
101	C
101	DS
102	DS
102	Math
105	AC
106	GG
107	DD

Result of the LEFT JOIN operation

Student LEFT JOIN $_{\text{rollno=rollno}}$ Library

>SELECT * FROM stuinfo AS ss LEFT JOIN stulib AS ll ON ss.rollno=ll.rollno;

Resultant Table ->

rollno	name	branch	rollno	book
101	aba	cse	101	C
101	aba	cse	101	DS
102	sh	ece	102	DS
102	sh	ece	102	Math
105	hhh	cse	105	AC
103	gh	ece	(NULL)	(NULL)
104	h	cse	(NULL)	(NULL)

RIGHT OUTER Join

rollno	name	branch
101	aba	cse
102	sh	ece
103	gh	ece
104	h	cse
105	hhh	cse

<- Stuinfo

Stulib->

rollno	book
101	C
101	DS
102	DS
102	Math
105	AC
106	GG
107	DD

Result of the LEFT JOIN operation

Student LEFT JOIN $_{\text{rollno=rollno}}$ Library

>SELECT * FROM stuinfo AS ss RIGHT JOIN stulib AS ll ON ss.rollno=ll.rollno;

Resultant Table ->

rollno	name	branch	rollno	book
101	aba	cse	101	C
101	aba	cse	101	DS
102	sh	ece	102	DS
102	sh	ece	102	Math
105	hhh	cse	105	AC
(NULL)	(NULL)	(NULL)	106	GG
(NULL)	(NULL)	(NULL)	107	DD

FULL OUTER Join

Result of the FULL OUTER JOIN operation

Student FULL OUTER JOIN $_{\text{rollno=rollno}}$ Library

>SELECT * FROM stuinfo AS ss LEFT JOIN stulib AS ll ON ss.rollno=ll.rollno

UNION

SELECT * FROM stuinfo AS ss RIGHT JOIN stulib AS ll ON ss.rollno=ll.rollno;

Resultant Table ->

rollno	name	branch	rollno	book
101	aba	cse	101	C
101	aba	cse	101	DS
102	sh	ece	102	DS
102	sh	ece	102	Math
105	hhh	cse	105	AC
103	gh	ece	(NULL)	(NULL)
104	h	cse	(NULL)	(NULL)
(NULL)	(NULL)	(NULL)	106	GG
(NULL)	(NULL)	(NULL)	107	DD

SELF JOIN OPERATION

A self join is a join in which a table is joined with itself (which is also called Unary relationships), especially when the table has a FOREIGN KEY which references its own PRIMARY KEY. To join a table itself means that each row of the table is combined with itself and with every other row of the table.

The self join can be viewed as a join of two copies of the same table. The table is not actually copied, but SQL performs the command as though it were. **It implements Recursive Relation in Table or we can say Recursive Closure.**

The syntax of the command for joining a table to itself is almost same as that for joining two different tables. To distinguish the column names from one another, aliases for the actual the table name are used, since both the tables have the same name. Table name aliases are defined in the FROM clause of the SELECT statement. See the syntax:

```
SELECT a.column_name, b.column_name... FROM table1 a,
table1 b WHERE a.common_filed = b.common_field;
```

SELF JOIN OPERATION

We have used a table EMPLOYEE, that has one to many relationship. Where SuperSSN is the id of Supervisor corresponding to every employee and also it is the Foreign key referring to Primary Key of Same table i.e SSN.

Employee Table:

SSN	Ename	Salary	DNO	SuperSSN
101	Payal	21600	4	111
104	Aman	25000	4	113
106	Hanish	28000	3	111
111	Ria	30000	5	104
113	Heena	20400	3	111
121	Deeshu	31000	4	113
203	Isha	29000	2	104

Self Join Operation

SQL Syntax to find out name of the Supervisor corresponding to a emp:

Employee Table:

SSN	Ename	Salary	DNO	SuperSSN
101	Payal	21600	4	111
104	Aman	25000	4	113
106	Hanish	28000	3	111
111	Ria	30000	5	104
113	Heena	20400	3	111
121	Deeshu	31000	4	113
203	Isha	29000	2	104

>SELECT e1.SSN AS EID, e1.Ename AS EmpName, e1.SuperSSN AS SupervisorID, e2.Ename AS SupervisorName FROM emp e1, emp e2 WHERE e1.superssn= e2.SSN;

EID	EmpName	SupervisorID	SupervisorName
101	Payal	111	Ria
104	Aman	113	Heena
106	Hanish	111	Ria
111	Ria	104	Aman
113	Heena	111	Ria
121	Deeshu	113	Heena
203	Isha	104	Aman

<- Resultant Table

Recursive Closure/ Self Join Ex.

(Borg's SSN is 888665555)

SUPERVISION	SSN	SUPERSSN
	123456789	333445555
	333445555	888665555
	999887777	987654321
	987654321	888665555
	666884444	333445555
	453453453	333445555
	987987987	987654321

RESULT 1	SSN
	333445555
	987654321

(Supervised by Borg)

RESULT 2	SSN
	123456789
	999887777
	666884444
	453453453
	987987987

(Supervised by Borg's subordinates)

RESULT	SSN
	123456789
	999887777
	666884444
	453453453
	987987987
	333445555
	987654321

(RESULT1 $\cup$ RESULT2)

Recursive Closure/ Self Join Ex.

(Borg's SSN is 888665555)

SUPERVISION	SSN	SUPERSSN
	123456789	333445555
	333445555	888665555
	999887777	987654321
	987654321	888665555
	666884444	333445555
	453453453	333445555
	987987987	987654321

>SELECT ssn FROM supervisor WHERE superssn=888665555;

RESULT 1	SSN
	333445555
	987654321

(Supervised by Borg)

Recursive Closure/ Self Join Ex.

(Borg's SSN is 888665555)

SUPERVISION	SSN	SUPERSSN
	123456789	333445555
	333445555	888665555
	999887777	987654321
	987654321	888665555
	666884444	333445555
	453453453	333445555
	987987987	987654321

>SELECT ssn FROM supervisor WHERE SuperSSN IN(SELECT ssn FROM supervisor WHERE superssn=888665555);

RESULT 2	SSN
	123456789
	999887777
	666884444
	453453453
	987987987

(Supervised by Borg's subordinates)

Recursive Closure/ Self Join Ex.

>>>SELECT ssn FROM supervisor WHERE superssn=888665555 UNION
SELECT ssn FROM supervisor WHERE SuperSSN IN (SELECT ssn FROM supervisor WHERE superssn=888665555);

RESULT	SSN
	123456789
	999887777
	666884444
	453453453
	987987987
	333445555
	987654321

(RESULT1 $\cup$ RESULT2)

Additional Relational Operations

- Aggregate Functions and Grouping
- Recursive Closure Operations
- The OUTER JOIN Operation

Some Functions for “StuLib” Table

- **SELECT COUNT(book) AS ISSUED_BOOKS FROM stulib;**

ISSUED_BOOKS
7

rollno	book	fine
101	C	100
101	DS	60
102	DS	80
102	Math	200
105	AC	130
106	GG	90
107	DD	170

- **SELECT SUM(fine) FROM stulib;**

sum(fine)
830

- **SELECT AVG(fine) FROM stulib;**

AVG(fine)
118.5714

Group By Clause on "StuLib" Table

- **SELECT * FROM stulib GROUP BY rollno;**

rollno	book	fine
101	C	100
102	DS	80
105	AC	130
106	GG	90
107	DD	170

rollno	book	fine
101	C	100
101	DS	60
102	DS	80
102	Math	200
105	AC	130
106	GG	90
107	DD	170

- **SELECT rollno, COUNT(book) FROM stulib GROUP BY rollno;**

rollno	COUNT(book)
101	2
102	2
105	1
106	1
107	1

Having Clause on "StuLib" Table

- SELECT rollno, COUNT(book) FROM stulib GROUP BY rollno ORDER BY COUNT(book);

rollno	book	fine
101	C	100
101	DS	60
102	DS	80
102	Math	200
105	AC	130
106	GG	90
107	DD	170

rollno	COUNT(book)
105	1
106	1
107	1
101	2
102	2

- SELECT rollno, COUNT(book) FROM stulib GROUP BY rollno HAVING COUNT(book)>1 ORDER BY rollno DESC;

rollno	count(book)
102	2
101	2

Aggregate Functions

E.g. SUM, AVERAGE, MAX, MIN, COUNT

(a)

R	DNO	NO_OF_EMPLOYEES	AVERAGE_SAL
	5	4	33250
	4	3	31000
	1	1	55000

(b)

DNO	COUNT_SSN	AVERAGE_SALARY
5	4	33250
4	3	31000
1	1	55000

(c)

COUNT_SSN	AVERAGE_SALARY
8	35125

Aggregate Functions

>>> Select COUNT(SSN), AVG(SALARY) FROM employee;

(c)

COUNT_SSN	AVERAGE_SALARY
8	35125

Aggregate Functions

>>> Select DNO, COUNT(EMPNO) AS NO_OF_EMPLOYEE, AVG(SALARY) AS AVERAGE_SAL FROM employee GROUP BY DNO;

(a)

R	DNO	NO_OF_EMPLOYEES	AVERAGE_SAL
	5	4	33250
	4	3	31000
	1	1	55000

Aggregate Functions

>>> Select DNO, COUNT(SSN) AS COUNT_SSN, AVG(SALARY) AS AVERAGE_SALARY FROM employee GROUP BY DNO;

(b)

DNO	COUNT_SSN	AVERAGE_SALARY
5	4	33250
4	3	31000
1	1	55000

Aggregate operations with JOINs

rollno	name	branch
101	aba	cse
102	sh	ece
103	gh	ece
104	h	cse
105	hhh	cse

<- Stuinfo

Stulib->

rollno	book	fine
101	C	100
101	DS	60
102	DS	80
102	Math	200
105	AC	130
106	GG	90
107	DD	170

>>> SELECT ss.rollno, ss.name, COUNT(ll.book), SUM(ll.fine) FROM stuinfo AS ss JOIN stulib AS ll ON ss.rollno=ll.rollno GROUP BY rollno;

rollno	name	COUNT(ll.book)	SUM(ll.fine)
101	aba	2	160
102	sh	2	280
105	hhh	1	130

Aggregate operations with JOINs

rollno	name	branch
101	aba	cse
102	sh	ece
103	gh	ece
104	h	cse
105	hhh	cse

<- Stuinfo

Stulib->

rollno	book	fine
101	C	100
101	DS	60
102	DS	80
102	Math	200
105	AC	130
106	GG	90
107	DD	170

>>> SELECT ss.rollno, ss.name, COUNT(ll.book), SUM(ll.fine) FROM stuinfo AS ss

JOIN stulib AS ll ON ss.rollno=ll.rollno GROUP BY rollno Having COUNT(ll.book)>1;

rollno	name	COUNT(ll.book)	SUM(ll.fine)
101	aba	2	160
102	sh	2	280

Aggregate operations with JOINs

<- Stuinfo

rollno	name	branch
101	aba	cse
102	sh	ece
103	gh	ece
104	h	cse
105	hhh	cse

Stulib->

rollno	book	fine
101	C	100
101	DS	60
102	DS	80
102	Math	200
105	AC	130
106	GG	90
107	DD	170

>>> SELECT ss.rollno, ss.name, COUNT(ll.book), SUM(ll.fine) FROM stuinfo AS ss

JOIN stulib AS ll ON ss.rollno=ll.rollno GROUP BY rollno Having COUNT(ll.book)>1 order by rollno desc;

rollno	name	count(ll.book)	sum(ll.fine)
102	sh	2	280
101	aba	2	160