ADBMS (ECS-242/602)

Abdul Rehman email id: abdul@northern.edu.pk Whatsapp#03087792217

Lesson Plan 3, 4 (Relational Algebra)

Duration: **3 hours (180 minutes)**

Objectives: **Learning objectives of this lecture are**

- **Join Operator**
- **Rules of Join Operator**
- **Difference between Joins and Cartesian Product**
- **Types Join**
- **Relational Algebra**
- **Use and need of relational algebra**
- **Unary Operators**
- **Binary Operators.**

Abdul Rehman email id: abdul@northern.edu.pk Whatsapp#03087792217

JOIN:

Join operators works almost exactly like cartesian product. That is, row of first table with all rows of second table and so on and so for. But there is a major difference between both. Cartesian product does not have any rule but join have a rule without which join is not possible.

Rule To apply Join:

- JOIN must have condition in it. (Join does not apply without a condition)
- Condition of join must have only columns in it. (not value in it)

For example.

Condition 1 = s.Regno > 101 (in this 101 is a value so not valid for join)

Condition 2 = s.Regno = p.Regno (here no value is used, so this is valid)

Example of Join:

Now let us repeat example of cartesian product. where we wanted to see student names and project names of those students who have been allocated projects.

SQL Query: Select * from Student join Project on Student.RegNo = Project.RegNo;

RegNo	sName	sAge	sCity	pId	pName	Regno
101	NAUMAN	31	RWP	p1	DBS	101
101	NAUMAN	31	RWP	p3	CCN	101
102	SHAHID	32	RWP	p2	PF	102
103	IMRAN	22	ISB	p4	OOP	103

NOTE: condition of JOIN operator is written with on operator. All other conditions are written in where clause sigma operator.

Here we can see that the result of above query is exactly same as we obtained using cartesian product. But here result is accurate in 1st step. But we have to display only two columns so we will change above query accordingly.

SQL Query: Select sName, pName from Student join Project on Student.RegNo = Project.RegNo;

Output:

sName	pName
NAUMAN	DBS
NAUMAN	CCN

SHAHID	PF
IMRAN	OOP

Here is our desired result.

Types of JOIN Operator:

1- Theta Join
2- Equi Join (a particular type of Theta join)
3- Natural Join
4- Semi Join

All above joins works exactly same way except LEFT join. Difference is just in writing style nothing else. Each is explained with example below.

1- Theta Join

Any join between two relations(tables) where condition of join contains operators (>, >=, <, <=, !=) will be considered as Theta Join.

2- Equi Join

Any join between two relations(tables) where condition of join contains operator (=) will be considered as Equi Join.

3- Natural Join

Condition of natural join is not mentioned (it doesn't mean there is no condition just like if all columns are fetched). So natural join condition means all those columns of both table with same name over the operator of "=").

Semi Join (►)

Semi join works exactly like theta join. The difference is it only display columns of left table.

Example:

Select student.* from student join project on student.regno = project.regno

Result of both above queries is same.

4- LEFT Join:

Left join is different from all other joins. It is the only join which cannot be achieved using cartesian product. Mean, all other join can be converted into cartesian product and vice versa, but not LEFT join.

Left join has two phases of displaying output.

1- Display data same as theta join. (only display those records where condition of JOIN is correct)
2- If there exists any record in left table which doesn't match with any record in right table, also display that record but display "NULL" in all columns of right table.

Example:

Select * from student left join project on student.regno = project.regno

RegNo	sName	sAge	sCity	pId	pName	Regno
101	NAUMAN	31	RWP	p1	DBS	101
101	NAUMAN	31	RWP	p3	CCN	101
102	SHAHID	32	RWP	p2	PF	102
103	IMRAN	22	ISB	p4	OOP	103
104	AYESHA	29	LHR	NULL	NULL	NULL

Here we can see, last record which is highlighted in blue doesn't match any of record in project table as there is no project which is assigned to student (104) but still it is displayed because it is in left table. And you can see NULL value is shown in all columns of right table (pid, pname, Regno).

Relational Algebra:

"Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output".

Relation:

- Means Table in Database.
- A database table is something which has rows and columns in it.

Algebra:

- "the part of mathematics in which letters and other general symbols are used to represent numbers and quantities in formulae and equations".

Algebraic Equation:

- An expression combination of operands with some operators between them.
- 5x + 2y = z (Example of Equation)

Abdul Rehman email id: abdul@northern.edu.pk Whatsapp#03087792217

Use of Algebra

- Let us say we have an algebraic equation

$$\frac{5x^2+\sqrt{(3y-2z)}}{2x^3y^2} = t \quad \text{..... (1)}$$

- In above equation we have 3 variables (x, y, z and t).
- Let us say;
  - x = "washing powder"
  - y = "amount of water in liters"
  - z = "time in minutes"
  - t = "Super clean clothes"
- Now above equation takes 2 hours to complete the process so that we could get clean clothes.
- Problem is 2 hours are not acceptable to clients.
- So, we try to Simplify Equation#1.
- Now let us say simplified version of Equation#1 is

$$\frac{2x+3y}{3} = t \quad \text{..... (2)}$$

- Now Equation#2 will generate same result as Equation#1.
- But execution time of Equation#2 is 20 minutes. Which is far less than Equation#1.
- This is the main moto of Algebra to Simplify complex equations into simpler equations so that we can minimize cost of process by minimizing time cost and resource cost as well.

Need of Relational Algebra:

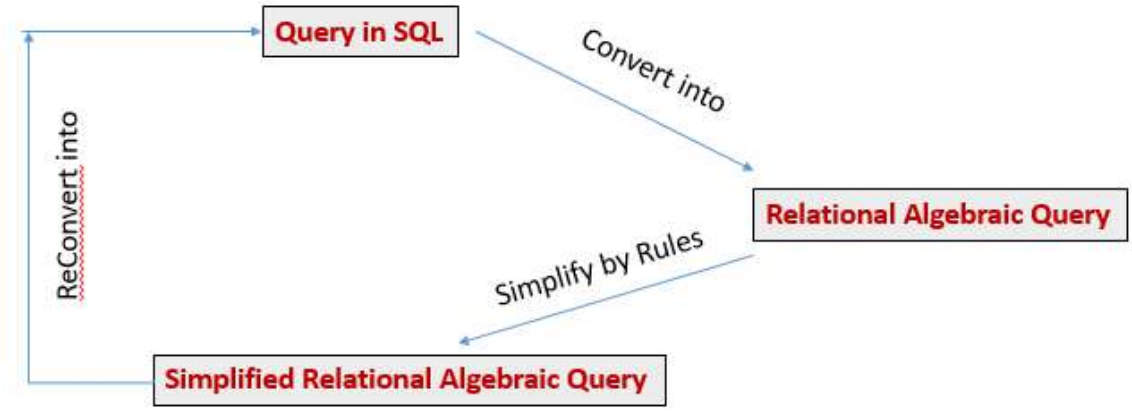
- As we have seen above, Relation means "Table".
- So Relational Algebra means to "Apply algebra over table".
- Why apply algebra over database Tables?
  - To minimize the overall execution cost(time) of a SQL query to generate results.
- In large databases, where number of records are in millions. Or data is placed on different servers.

- Process of retrieval of data becomes slow. Real time queries can take hours to get executed.
- So Relational Algebra is used to minimize time cost of a query so that it can be executed in timely manner.

Here is The Deal:

- We write Query in SQL, but SQL cannot be simplified because it is plain English and no rules.
- We will convert SQL into Relational Algebraic Expression.
- Simplify relational algebraic query and make it less complex.
- Reconvert into SQL and Execute

NOTE: Relational Algebraic Query is not acceptable by SQL Server, so eventually only SQL query will be executed by system.



Operator and Operand:

- Every equation has some operators and some operands.

Operand:

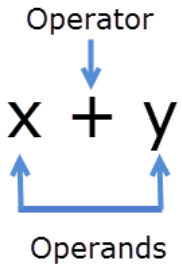
- An object on which operation has to be performed.

Operator:

- An operator is a symbol which represent any specific operations to perform. For example; addition is an operation and "+" is operator used for addition.

Abdul Rehman email id: abdul@northern.edu.pk Whatsapp#03087792217

For Example:



Types of Operators:

1- **Unary Operators** (which applies on a single operand)

For example; x++, here ++ is operator which is applied on a single operand "x".

2- **Binary Operator** (which applies on two operands)

For example;

- x + y, here "+" is operator which is applied on two operands x and y.
- 6 × 9, here "×" is operator which is applied on two operands 6 and 9.

- **Relational algebra has its own operators and operands.**

Operands in Relational Algebra:

- Relations (**Table**) will be only operands used in relational algebra.
- Relational algebra operator will only apply on Database relations (**Tables**) nothing else.

Operators in Relational Algebra:

1- **Unary Operators**

   a. **Projection Operator**

   b. **Selection Operator**

2- **Binary Operators**

   a. **Union**

   b. **Intersection**

   c. **Set Difference**

   d. **Cartesian Product (Cross Product)**

e. Join

Let us discuss each operator in details now

1- Projection Operator (Pi- π)

- Symbol of projection operator is (**π**) pronounced as **"Pi".**
- It is used to project data. (project means display)
- Its SQL keyword is "**SELECT**".
- So we can say that we will replace SELECT keyword with **π.**

Example 1:

Let us say we have a SQL query

SELECT * from Student

- We have to convert this into Relational Query
- First of all, we will write Operand (which is Table) in query.
- After words, we will examine have we read any symbol for keywords written in query.
- Yes, we have studied "Select" keyword which will be replaced by **"π"** operator.
- Then, write all columns which are written with **SELECT** clause at the subscript of "**π**" operator.

NOTE: Unary operator will always be written on LEFT side of operand.

$$(\pi_{*}(\text{STUDENT}))$$

- Above is the relational expression which means display all columns from Student table.

Example 2:

Consider the following database table of student.

RegNumber	Name	CGPA	CITY
101	ASMA	3.4	ISB
102	QADEER	2.3	LHR
103	SHAMS	3.8	RWP

- Let us convert the SQL query into relational query and also display output of the query according to given data.

SQL	SELECT Name, CGPA from Student
Relational Query	($\pi_{Name, CGPA}$ (STUDENT))

Output:

Name	CGPA
ASMA	3.4
QADEER	2.3
SHAMS	3.8

Example 3:

Consider the following database table of student.

Student

RegNumber	Name	CGPA	CITY
101	ASMA	3.4	ISB
102	QADEER	2.3	LHR
103	SHAMS	3.8	RWP

- Let us convert the SQL query into relational query and also display output of the query according to given data.

SQL	SELECT Count(*) from Student
Relational Query	($\pi_{Count(*)}$ (STUDENT))

Output:

No Column
3

2- Selection Operator (Sigma - σ)

- Symbol of selection operator is (**σ**) pronounced as "**Sigma**".
- It is used to get specific rows from the table.
- Its SQL keyword is "**WHERE**".
- So we can say that we will replace **WHERE** keyword with **σ**.
- Unary operator will always be written on left side of operand(table).

Example 1:

Now Consider the same table we used above.

Student

ADBMS (ECS-242/602)

Abdul Rehman **email id: abdul@northern.edu.pk** **Whatsapp#03087792217**

RegNumber	Name	CGPA	CITY
101	ASMA	3.4	ISB
102	QADEER	2.3	LHR
103	SHAMS	3.8	RWP

Let us say we have an SQL statement to be converted into Relational Algebraic query.

SELECT * FROM STUDENT WHERE CGPA<3.0

- First of all, we will see how many Keywords have we studies.
- By looking at the above SQL query, we can say that we have studies two keywords. "**SELECT**" and "**WHERE**".
- Now at a time, only one operator can be applied, so which one of them would be applied first?
- For now, we will always give priority to "**Sigma**" operator.
- Pi operator will always be executed at the end of query.

$$(\pi_{Name}\ (\sigma_{CGPA\ <\ 3.0}\ (STUDENT)))$$

- In above relational query, first sigma operator is applied on student table.
- Condition in **WHERE** clause will be written in subscript of Sigma operator.
- Above query shows the sequence of execution of SQL query.
    - At first, Sigma will be applied on student table which will bring all those students whose CGPA is less than 3.0.
    - Here as for now, Pi operator is not executed, so it will not affect any of the columns.

RegNumber	Name	CGPA	CITY
102	QADEER	2.3	LHR

- Here we see after applying sigma, only those students have arrived whose CGPA is less than 3.0. but all columns of table are projected.
- Now Pi will be applied on the result of Sigma not on the "Student" table as a whole.
- So after Pi will be applied, only Name column will be displayed. Shown as below

Name
QADEER

Example 2:

- As we know, **WHERE** clause can have logical operators like **"AND / OR".**
- "AND" will be replaced by Cap (^) symbol.
- "OR" will be replaced by (∨) symbol.
- "**BETWEEN**, **IN**" will be written as it is but in subscript of Sigma operator.

Now let us consider given table of Student.

Student

RegNumber	Name	CGPA	CITY
101	ASMA	3.4	ISB
102	QADEER	2.3	LHR
103	SHAMS	3.8	RWP
104	BILAL	2.8	RWP
105	NAZIA	3.9	ISB

- Let us convert given SQL statement into Relational Algebraic query.

SQL	**SELECT** Name, CGPA **FROM STUDENT** **WHERE** CITY = 'ISB' or CGPA > 3.5
Relational Step 1: Apply Sigma	$(\sigma_{\text{CITY='ISB'} \vee \text{CGPA>3.5}}(\text{STUDENT}))$
Relational Step 2: Apply Pi	$(\pi_{\text{Name,CGPA}}(\sigma_{\text{CITY='ISB'} \vee \text{CGPA>3.5}}(\text{STUDENT})))$ Final Query

Output Step 1:

RegNumber	Name	CGPA	CITY
101	ASMA	3.4	ISB
103	SHAMS	3.8	RWP
105	NAZIA	3.9	ISB

- All columns are projected because Pi is not applied yet, only sigma is applied which displayed those students whose CITY is 'Islamabad' or their CGPA is more than 3.5

Output Step 2 (Final Step):

Name	CGPA
ASMA	3.4
SHAMS	3.8
NAZIA	3.9

- As Pi has only two columns Name and CGPA so above is the final output of Query.

Abdul Rehman email id: abdul@northern.edu.pk Whatsapp#03087792217

Example 3:

- Consider the student table given below
- **Student**

RegNumber	Name	CGPA	CITY
101	ASMA	3.4	ISB
102	QADEER	2.3	LHR
103	SHAMS	3.8	RWP
104	BILAL	2.8	RWP
105	NAZIA	3.9	ISB

- Convert given SQL statement into the Relational Algebraic query.

```
SELECT NAME FROM STUDENT
WHERE CGPA = (SELECT MAX(CGPA) FROM STUDENT)
```

- Here we can see that there are two Select statements in above SQL statement.
- We will start with main SELECT which will display "NAME" column.

$$(\pi_{NAME}(\sigma_{CGPA = ?}(STUDENT)))$$

- As we can, there is a question mark after CGPA which is another SQL Select Query.
- Now let us convert only inner Select statement and substitute it with question mark.

SubQuery	$(\pi_{Max(CGPA)}(STUDENT))) = X$
Substituting Subquery into Main Query	$(\pi_{NAME}(\sigma_{CGPA = X}(STUDENT)))$

- Here we have written x which is equivalent to inner query.
- This is because to make query look like less complicated.
- Otherwise, we can write whole value as shown below

$$(\pi_{Name}(\sigma_{CGPA=(\pi_{Name}(\sigma_{CGPA=}(Student)))}(Student)))$$

Output:

- Final output of above query is given below.

Name
NAZIA

NOTE:

$(\pi_{*}(\sigma_{CGPA<3.0}(STUDENT))) = (\sigma_{CGPA<3.0}(STUDENT))$
Above Two queries are same and there SQL query will also be same which is SELECT * FROM STUDENT WHERE CGPA < 3.0
- Reason is when Pi has all columns (*) then we can skip writing Pi. - It doesn't mean there is no Pi operator. - It means Pi operator has all columns projected. - It is just to make Relational Query look like simple.

Union Operator

- Union is represented by "**union**" between multiple SQL queries.
- Rest of the query remains same.

Examples:

1	Relational Query	$(\pi_{NAME}(STUDENT)) \cup (\pi_{NAME}(EMPLOYEE))$
	SQL Query	SELECT * FROM STUDENT union SELECT * FROM PROJECT
2	Relational Query	$(\pi_{REG,NAME}(\sigma_{CGPA>3.0}(STUDENT)) \cup (\pi_{EMPID,NAME}(\sigma_{CITY='ISB'}(EMPLOYEE))$
	SQL Query	SELECT REG, NAME FROM STUDENT WHERE CGPA > 3.0 UNION SELECT EMPID, NAME FROM EMPLOYEE WHERE CITY='ISB'

Intersection Operator

- Intersection is represented by "**intersect**" between multiple SQL queries.
- Rest of the query remains same.

Examples:

1	Relational Query	$(\pi_{REG}(STUDENT)) \cap (\pi_{EMPID}(EMPLOYEE))$

	SQL Query	**SELECT * FROM STUDENT Intersect SELECT * FROM PROJECT**
2	Relational Query	**($\pi_{REG,NAME}$ ($\sigma_{CGPA > 3.0}$ (STUDENT)) ∩ ($\pi_{EMPID, NAME}$ ($\sigma_{CITY='ISB'}$ (EMPLOYEE))**
	SQL Query	**SELECT** REG, NAME **FROM STUDENT WHERE** CGPA > 3.0 **Intersect SELECT** EMPID, NAME **FROM EMPLOYEE WHERE** CITY='ISB'

Set Difference Operator

- Set difference is represented by "**except**" between multiple SQL queries.
- Rest of the query remains same.

Examples:

1	Relational Query	**(π_{REG} (STUDENT)) — (π_{EMPID} (EMPLOYEE))**
	SQL Query	**SELECT * FROM STUDENT Except SELECT * FROM PROJECT**
2	Relational Query	**($\pi_{REG,NAME}$ ($\sigma_{CGPA > 3.0}$ (STUDENT)) — ($\pi_{EMPID, NAME}$ ($\sigma_{CITY='ISB'}$ (EMPLOYEE))**
	SQL Query	**SELECT** REG, NAME **FROM STUDENT WHERE** CGPA > 3.0 **Except SELECT** EMPID, NAME **FROM EMPLOYEE WHERE** CITY='ISB'

SQL Query of Cartesian Product:

- Cartesian product is represented by comma symbol "**,**" between multiple tables in SQL.
- Rest of the query remains same.

Examples:

1	Relational Query	**(Student × Project)**
	SQL Query	**SELECT * FROM STUDENT , PROJECT**
2	Relational	**($\pi_{S.NAME}$ (Student)) × Project**

	Query	
	SQL Query	**SELECT** s.Name, p.* **From Student s, Project p**
3	Relational Query	$(\pi_{S.NAME,P.PNAME}\ (\sigma_{S.RegNo=P.RegNo}\ (\textbf{Student} \times \textbf{Project})))$
	SQL Query	**SELECT** S.NAME, P.PNAME **FROM STUDENT S, PROJECT P WHERE** S.REGNO = P.REGNO
4	Relational Query	$(\pi_{S.NAME,S.AGE,P.PNAME}\ (\sigma_{S.RegNo=P.RegNo \wedge S.AGE>21}\ ((\textbf{Student} \times \textbf{Project}) \times \textbf{Course})))$
	SQL Query	**SELECT** S.NAME, S.AGE, P.PNAME **FROM STUDENT S, PROJECT P, COURSE C WHERE** S.REGNO=P.REGNO AND S.AGE > 31

Equi Join:

- Any join between two relations(tables) where condition of join contains operators (=) will be considered as Equi Join.

Examples:

1	Relational Query	$(\textbf{Student} \bowtie_{s.RegNo=p.RegNo} \textbf{Project})$
	SQL Query	**SELECT * FROM STUDENT s join PROJECT p on s.regNo = p.regNo**
2	Relational Query	$(\pi_{S.NAME}\ (\textbf{Student} \bowtie_{s.RegNo=p.RegNo} \textbf{Project}))$
	SQL Query	**SELECT s.Name FROM STUDENT s join PROJECT p on s.regNo = p.regNo**
3	Relational Query	$(\pi_{S.NAME,P.PNAME}\ (\sigma_{S.age>30}\ (\textbf{Student} \bowtie_{s.RegNo=p.RegNo} \textbf{Project})))$
	SQL Query	**SELECT s.Name, p.Name FROM STUDENT s join PROJECT p on s.regNo = p.regNo where s.age>30**
4	Relational Query	$(\pi_{S.NAME,s.age,\ P.PNAME}\ (\sigma_{S.age>30\ \wedge\ s.city\ =\ 'isb'}\ (\textbf{Student} \bowtie_{s.RegNo=p.RegNo})))$
	SQL Query	**SELECT s.Name,s.age, p.Name FROM STUDENT s join PROJECT p on s.regNo = p.regNo where s.age>30 and s.city='isb'**

Abdul Rehman email id: abdul@northern.edu.pk Whatsapp#03087792217

Theta Join:

- Any join between two relations(tables) where condition of join contains operators (>, >=, <, <=, !=) will be considered as Theta Join.

  Example: Select * from student s inner join project p on s.regno>p.regno

  (Student $\bowtie_{s.RegNo > p.RegNo}$ Project)

Natural Join:

Condition of natural join is not mentioned.

It doesn't mean there is no condition just like if π operator is not mentioned it means all columns are fetched.

So natural join condition means all those columns of both table with same name over the operator of "=").

Example: Select * from student s inner join project p on s.regno=p.regno

(Student $\bowtie$ Project)

Semi Join:

- Semi join works exactly like theta join.
- **The difference is it only display columns of left table.**

Example 1: Select S.* from student s inner join Project p on s.regno=p.rogno

(Student $\blacktriangleright_{s.RegNo=p.RegNo}$ Project)

($\pi_{s.*}$ (Student $\bowtie_{s.RegNo=p.RegNo}$ Project))

Example 2: Select p.* from student s inner join Project p on s.regno=p.rogno

(Project $\blacktriangleright_{s.RegNo=p.RegNo}$ Student)

($\pi_{P.*}$ (Project $\bowtie_{s.RegNo=p.RegNo}$ Student))

Left Join:

Select * from student s left join project p on s.regno=p.regno

(Student $⟕_{s.RegNo=p.RegNo}$ Project)

Lab Manual:

Read the material of Lab15. Solve Task and Exercise given in Lab15 on Lab Manual.