05. Querying Relations

[PM Jat, DAIICT, Gandhinagar]

What is a Query?

Here is an example of query
"**get me ID, Name of students studying in program 'BCS' and have CPI > 7.5**"

Queries are expressed in a query language. Such a query is submitted to RDBMS, and that in turn runs the query on specified relations and result is presented to the user.

Following are three query languages that can be used for expressing the query.

- Relational Algebra
- Relational Calculus
- Structured Query Language (SQL)

As examples, below are query expressions for above query in these languages.

Relational Algebra: $\sigma_{progid='BCS' \ AND \ cpi>7.5}(STUDENT)$

Relational Calculus: {t| STUDENT(t) AND t.progid='BCS' AND t.cpi > 7.5) }

SQL: SELECT * FROM STUDENT WHERE progid='BCS' AND CPI > 7.5;

Note that first two are pure mathematical expressions. Being its ability to express queries mathematically has been foremost strength of relational model.

However, for real work, most RDBMS only support SQL for submitting queries.

In this course we learn Relational Algebra and SQL side by side. We first learn how queries are expressed in relational algebra, and then map algebraic queries to SQL queries.

Relational Operations (Querying):

Relational Algebra allows expressing queries as algebraic expressions, where operands are relations and operators are "relational operations".

Here is list of relational operations

- SELECT (σ)
- PROJECT (π)
- Join (⋈)
- Cross JOIN (×)
- Set Operations: UNION, INTERSECTION, and DIFFERENCE
- Aggregate operations

While SELECT and PROJECT are unary operators, others are binary.

Note following in relation to relational operations:

- Algebraic operations are immutable. That is these operations are READ only and do not change operand relation.

- Relational algebra operations have closure property that means result of an algebraic expression is a valid relation
- Result being valid relation also mean, it is a set with no duplicate tuples.
- Result relation do have its own schema (and instance). However, are NOT stored in database, just in working memory.
- Query expressions can be sequencing operations such that result of an expression becomes an operand for next operation.

Let us now attempt understanding these operations one by one.

SELECT operation (σ):

SELECT is a unary operation, and represented by sigma **(σ)**. This operation is applied on a relation to get its subset (subset of tuples). Tuple selection criteria is parameter to this operation.

It is expressed as:

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

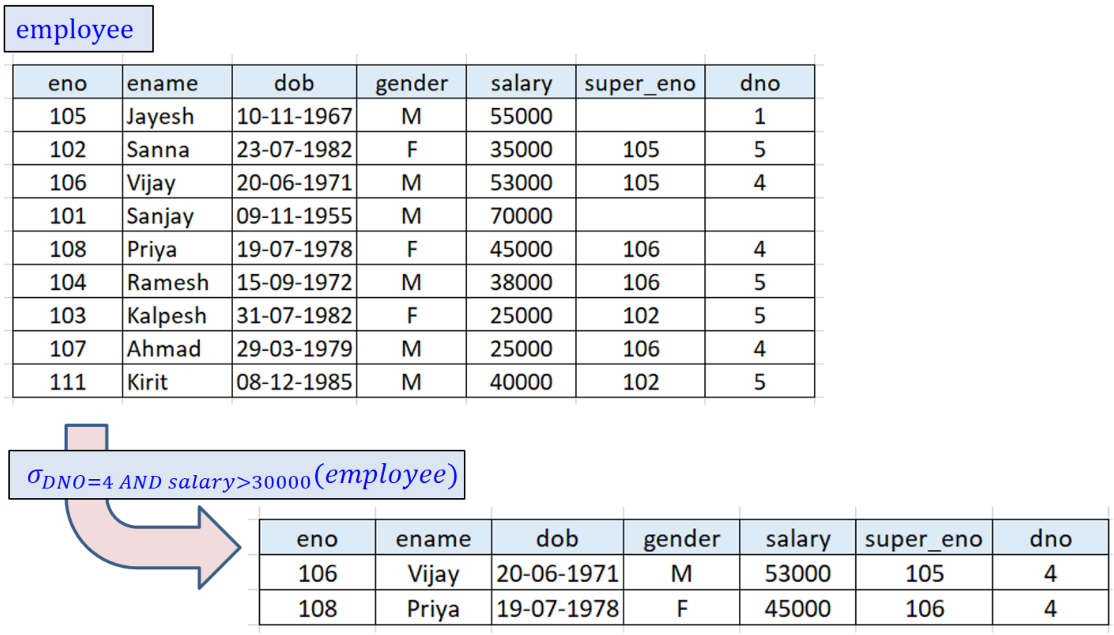
Here r is operand relation.

Example:

Consider a relation **employee** as shown below. To select the employee tuples where department number is four, and have salary > 30000, we would express it in relational algebra, as following:

$$\sigma_{dno=4\ AND\ salary\ >\ 30000}(employee)$$

Figure below depicts the result, when we apply this operation on the employee relation. It could be read as following: "produce a new relation by reading **employee** relation, and selecting only the tuple that meet the specified condition"



employee

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

$\sigma_{DNO=4\ AND\ salary>30000}(employee)$

eno	ename	dob	gender	salary	super_eno	dno
106	Vijay	20-06-1971	M	53000	105	4
108	Priya	19-07-1978	F	45000	106	4

Note that the SELECT operation does not truncate the operand relation stored in database. It only takes operand relation as input and produces the resultant relation in temporary storage.

The Resultant relation of SELECT operation has:

- Schema is same as the operand relation
- "Degree" is same as the operand relation. Degree means – "Number of Attributes"
- Cardinality can be anything from 0 to N, where N is cardinality of operand relation. Cardinality means - number of tuples.

PROJECT Operation (π)

PROJECT is also an unary operation. Represented by PI($\boldsymbol{\pi}$). This operation selects certain "columns" of operand relation. It can be viewed as getting vertical subset of a relation.

The general form of the PROJECT operation is

$$\pi_{<attribute-list>}(r)$$

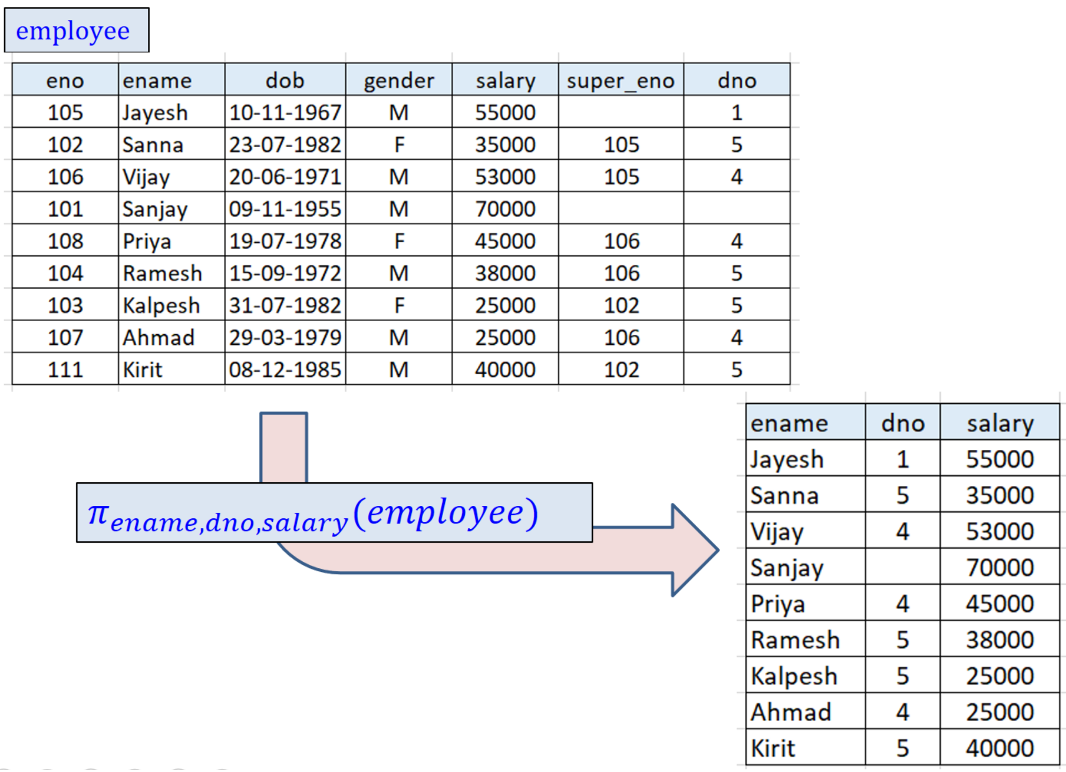
Where <attribute list> is the desired list of attributes (columns) from the attributes of relation R.

Example:

Consider the same relation **`employee`**. Select only employee name, date of birth, and salary columns from employee relation. In relational algebra, we would express it as following:

$$\pi_{ename,dno,\,salary}(employee)$$

Figure below depict the meaning of the expression here. It could be read as following: "produce a new relation by reading **employee** relation, and selecting only the tuple that meet the specified condition".



employee

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

$\pi_{ename,dno,salary}(employee)$

ename	dno	salary
Jayesh	1	55000
Sanna	5	35000
Vijay	4	53000
Sanjay		70000
Priya	4	45000
Ramesh	5	38000
Kalpesh	5	25000
Ahmad	4	25000
Kirit	5	40000

Result of PROJECT operation –

- is a valid relation – no duplicate tuples
- has cardinality same as of the instance of EMPLOYEE (and less, if projection operation has to drop some duplicate tuples in result set)
- has degree equal to number of attributes specified in the list

SELECT and PROJECT in SQL:

Querying expressions in SQL has following form:

```
SELECT <attribute-list>
         FROM <relation-name> | <relation-expression>
         [WHERE <predicate>]
         [ … more ]
```

Every SQL query statement begins with "SELECT" keyword, and then has more parts of it.

SELECT part specifies attributes that are to be projected

FROM clause is used to specify operand relation. It is important to note here is that this part can be a "relational expression" too, and that results to a relation

WHERE clause "predicate" specifies tuple selection criteria from the operand relation. Each tuple of operand relation is evaluated for specified condition, if the tuple meets the condition, then it is added to the resultant relation.

Order of evaluation of a SQL statement goes as following:

(1) FROM clause

(2) WHERE CLAUSE – produce the selected tuples from relation specified in FROM

(3) PROJECT as per attribute list

Note: This is not complete form of SQL statement. We shall add more to it at later stage.

Note that SELECT statement of SQL is not only SELECT operation of relational algebra. It is a SQL command for executing all "Query Answering" operations

Examples:

Here is SQL versions of a SELECT algebraic query that we have already discussed:

$\sigma_{dno=4\ AND\ salary\ >\ 30000}(employee)$

```
SELECT * FROM EMPLOYEE WHERE DNO=4 AND SALARY > 30000;
```

'' as attribute list here implies all attributes of the operand relation.*

Here is the PROJECT query, expressed in SQL

$$\pi_{ename,dno,\,salary}(employee)$$

```sql
SELECT ename, dno, salary FROM employee;
```

While projecting through SQL one thing is important to note that SQL project might produce duplicate tuples. Consider following expression in Algebra and SQL

$$\pi_{dno}(employee)$$

```sql
SELECT dno FROM employee;
```

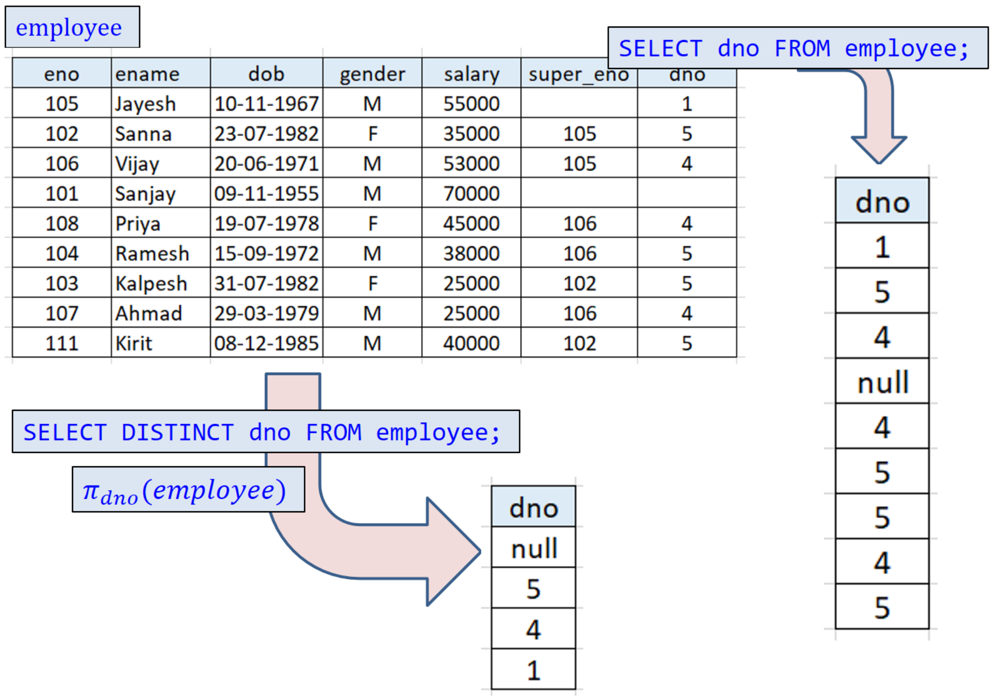
Algebraic expression evaluates to a single column relation with no duplicate tuples. In this query DNO is non-unique column and would result duplicate tuples. PROJECT operation, however should produce distinct tuples only, and that is true for Relational Algebra.

However, SQL version produces duplicate tuples. It shall be presenting the column DNO as it is. Therefore, algebraically equivalent SQL would be as following

$$\pi_{dno}(employee)$$

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

Figure below shows up sample operand relation and its result!



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

dno
1
5
4
null
4
5
5
4
5

dno
null
5
4
1

By default, SQL does not remove duplicates, primarily due to performance reasons as duplicate removal is bit expensive to execute. However, if user wants distinct values, can use the distinct tag. Note that DISTINCT here applies to the tuple not to specific attribute.

More examples:

- Query ##: List of Supervisors (that is employees that supervise some employees)

$$\pi_{super\_eno}(employee)$$

```
SELECT distinct super_eno FROM employee;
```

- Query ##: List (eno, name) of employees having salary > 50000

  Relational Algebra:

$$\pi_{eno,\,ename}\left(\sigma_{SALARY>50000}(employee)\right)$$

Note that this query solution is sequencing two operations. Projection follows Selection. Output of SELECT operation becomes input to PROJECT operation.

Here is SQL expression for the same query:

```
SELECT eno, ename FROM employee WHERE salary > 50000;
```

Ordering of Operations

It should be easy to notice that order of operation is more explicit in algebraic expressions. SQL queries too have order of operations, implicitly defined. We shall learn about that later.

Here it is to be noted that in some cases order of sequencing of operations may not matter while in other cases it can! For instance, following is incorrect for obvious reasons.

$$\sigma_{SALARY>50000}\left(\pi_{eno,\,ename}(employee)\right)$$

Often, we would also be splitting expression for making the expressions in the cases of complex queries.

$$r \leftarrow \sigma_{SALARY>50000}(employee)$$

$$result \leftarrow \pi_{eno,\,ename}(r)$$

Exact translation of this twostep solution in SQL would be as following

```
SELECT r.eno, r.ename FROM (SELECT * FROM employee
       WHERE salary > 5000 ) AS r;
```

Note however that this is mathematically equivalent to following

```
SELECT eno, ename FROM employee WHERE salary > 5000;
```

We can also explicitly refer attributes by relation, or its alias name (in later query) as following

```
SELECT employee.eno, employee.ename FROM employee
       WHERE salary > 5000;
```

```
SELECT e.eno, e.ename FROM employee AS e WHERE salary > 5000;
```

- Query ##: List (ID, Name, CPI) of B Tech CS Students having cpi > 7.5

$$\pi_{studid,\, name,\, cpi}\left(\sigma_{progid='BCS'\ AND\ cpi>7.5}(student)\right)$$

```
SELECT studid, name, cpi from student
       WHERE progid='BCS' AND cpi > 7.5;
```

It should also be noted from here that query writing is solved by identifying appropriate relational operations and proper sequencing of them. The query here is solved in following steps:

  - Apply Selection on Student (prodid = 'bcs' and cpi > 7.5)
  - Apply projection on result (id, name, cpi)

- Query ##: List (ID, Name, CPI) of B Tech CS and B Tech IT students having cpi > 7.5

$$\pi_{studid,\, name,\, cpi}\left(\sigma_{(progid='BCS'\ OR\ progid='BIT')\ AND\ cpi>7.5}(student)\right)$$

```
SELECT studid, name, cpi from student
       WHERE (progid='BCS' or progid='BIT') AND cpi > 7.5;
```

This query solution varies only in terms of tuple selection condition part from previous query

- Query ##: List eno, name of employees who are supervised by eno=105

$$\pi_{eno,\, ename}\left(\sigma_{super_{eno}=105}(employee)\right)$$

```
SELECT eno, ename FROM employee WHERE super_eno=105;
```

Recap of SELECT and PROJECT operations

SELECT operation

- General form: $\sigma_{<tuple\ selection\ criteria>}(r)$
- Results a subset of tuples meeting the selection criteria from the operand relation r

PROJECT operation

- General form: $\pi_{<attribute-list>}(r)$
- Results a relation that is a projection of specified attributes from the operand relation r
- Removes duplicates tuple if project results

More about SQL

ORDER BY

Recall general form of SQL:

```
SELECT <attribute-list>
         FROM <relation-name> | <relation-expression>
         [WHERE <predicate>]
         [ORDER BY <ordering attributes> [DESC] ];
         [ … more ]
```

We have added one more clause here, that is "ORDER BY"

ORDER BY clause is used to specify order in the result is to be arranged.

Here is an example

```
SELECT eno, ename, salary FROM employee AS e
       WHERE dno = 5
       ORDER BY salary DESC;
```

Note that Algebra has no concept of ordering. Relational Algebra sees a relation as set of tuple.

Expressions in WHERE predicate

Following expressions should be self-explanatory:

- `SELECT * FROM employee WHERE salary BETWEEN 30000 AND 50000`
- `SELECT * FROM programs WHERE did IN ('CS', 'EC')`
- `SELECT * FROM applicants WHERE (mark1+marks2+marks3)/3 >= 50`
- `SELECT * FROM employee WHERE ename like '*Shankar*'`

JOIN operations

Here is a motivating example of JOIN.

Consider given snapshot of student and program relations as shown below. Suppose we want to get consider answer of query "**List all attributes of students whose did= 'CS'**"

studid	name	progid	cpi
101	Rahul	BCS	8.7
102	Vikash	BEC	6.8
103	Shally	BEE	7.4
104	Alka	BEC	7.9
105	Ravi	BCS	9.3

pid	pname	intake	did
BCS	BTech(CS)	30	CS
BEC	BTech(ECE)	40	EE
BEE	BTech(EE)	40	EE
BME	BTech(ME)	40	ME

This query looks like a SELECT operation. But question is how do we specify selection criteria `did='CS'`. All required attributes are in relation student but attribute did is in relation program.

How do we solve this?

Recall that FK progid refers to corresponding program tuple. We can always find out referred tuple for a FK value. Operation that relational algebra provides for such situation is JOIN

Below is a result of JOIN of student and program

studid	name	progid	cpi	pid	pname	intake	did
101	Rahul	BCS	8.7	BCS	BTech(CS)	30	CS
102	Vikash	BEC	6.8	BEC	BTech(ECE)	40	EE
103	Shally	BEE	7.4	BEE	BTech(EE)	40	EE
104	Alka	BEC	7.9	BEC	BTech(ECE)	40	EE
105	Ravi	BCS	9.3	BCS	BTech(CS)	30	CS

Let us call this relation as SP. Try interpreting each tuple in this result.

- It has attributes from both the relations.
- For all tuples values in progid and pid are same. Do you any rationale behind this?
- If you look at a tuple say SID=103. The value of attribute pname shows name of the program in which student 103 studies. Value in DID column indicates department id of the student Shally.

Join in expressed as following in relational algebra

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

As seen here, join is a binary operation. It receives join-condition as parameter. Join condition actually tells how to combine tuples from operand relations to compute tuples for resultant relation.

For example, join of student and program, can be expressed as

$$student \frac{\bowtie}{progid = pid} program$$

This JOIN can be simply thought of combining tuples from two relations with the matched FK-PK pairs. This is basically combining referred tuples with referencing tuples.

Same join can be expressed in SQL as following:

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

Once we have computed JOIN, then we can apply necessary filter to select desired tuples, and finally necessary projection to get desired result:

$$\pi_{student.*}\left(\sigma_{did='CS'}\left(student \underset{progid=pid}{\bowtie} program\right)\right)$$

Same join can be expressed in SQL as following:

```
SELECT s.* FROM student AS s JOIN program ON progid=pid
        WHERE did='CS';
```

Note the s.* in attribute list for projection. Simple * would include all attributes from the join. If we want to show only student's attribute then we write student.* or s.* for short through alias s.

Example ##:

Another example from company database.

Consider following relations employee and department

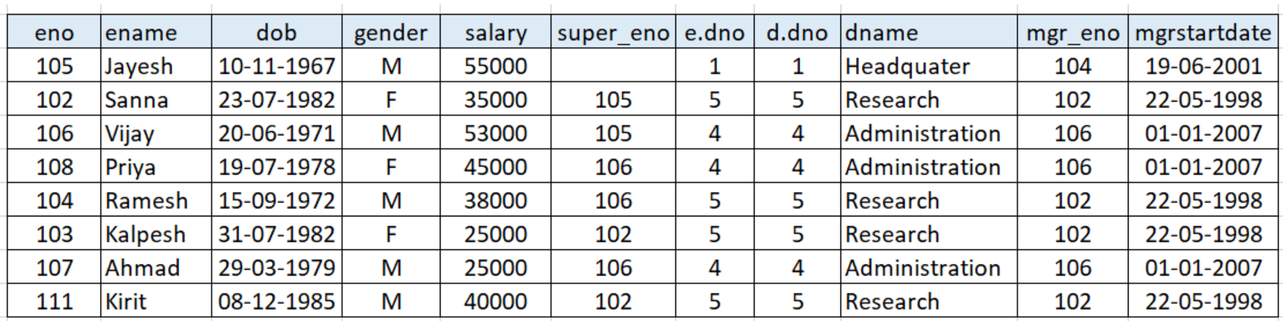
eno	ename	dob	gender	salary	super_eno	dno
105	Jayesh	10-11-1967	M	55000		1
102	Sanna	23-07-1982	F	35000	105	5
106	Vijay	20-06-1971	M	53000	105	4
101	Sanjay	09-11-1955	M	70000		
108	Priya	19-07-1978	F	45000	106	4
104	Ramesh	15-09-1972	M	38000	106	5
103	Kalpesh	31-07-1982	F	25000	102	5
107	Ahmad	29-03-1979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

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

JOIN of employee and department

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

Here is snapshot of the expressed JOIN expression



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

Try understanding of content of tuples here

- What does value of dname ‘Research’ mean to employee ‘Ramesh’?
- What does value of mgr_eno=102 mean to the employee ‘Ramesh’?

Question: Is there any loss of tuples while joining?

JOIN in SQL

Suppose we want join relation r1 and r2, then general SQL expression is as following:

```
SELECT * FROM r1 JOIN r2 ON <join-condition>;
```

Our EMPLOYEE-DEPARTMENT JOIN can be expressed as following:

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

This SQL statement also shows FROM clause of SQL having a relational expression and its result becomes operand for WHERE clause.

Example Query ##: List (ENO, ENAME, DNAME) of all employees with dno=4

$$\pi_{eno,ename,dname}\left(\sigma_{dno=4}\left(employee \underset{e.dno=d.dno}{\bowtie} department\right)\right)$$

```
SELECT eno,ename,dname FROM employee AS e
     JOIN department AS d ON (e.dno = d.dno)
     WHERE dno = 4;
```

There is another way, we can express this query

$$\pi_{eno,ename,dname}\left(\sigma_{dno=4}\,(employee) \underset{e.dno=d.dno}{\bowtie} department\right)$$

```
SELECT eno,ename,dname FROM
     (SELECT * FROM employee WHERE dno = 4) AS e
     JOIN department AS d ON (e.dno=d.dno);
```

Which one is better?

Though in execution terms one is better than the other, right now we would simply say that either is fin as long as queries are algebraically correct. Rest will be taken care by RDBMS query optimizer.

Let us consider another join of employee and department, and appreciate how join condition sometime is crucial.

$$employee \underset{eno=mgr\_eno}{\overset{\bowtie}{\overline{\qquad\qquad}}} department$$

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

Here is snapshot of the resultant relation.

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

Try understanding of content of tuples here

- What does value of dname 'Research' mean to employee 'Sanna'?
- What does value of mgr_eno=102 mean to the employee 'Sanna'?

Exercise ##: Interpret tuples of result of following JOIN

$$employee\ (e) \underset{e.super\_eno=s.eno}{\overset{\bowtie}{\overline{\qquad\qquad}}} employee\ (s)$$

```sql
SELECT * FROM employee AS e
    JOIN employee AS s ON e.super_eno=s.eno;
```

Here is snapshot of resultant relation. You might notice all attributes are appearing twice.

Try figuring out, why is it so, and what does this mean?

eno	ename	dob	gender	salary	super_eno	dno	eno-2	ename-2	dob-2	gender-2	salary-2	super_eno-2	dno-2
102	Sanna	23-07-1982	F	35000	105	5	105	Jayesh	10-11-1967	M	55000		1
106	Vijay	20-06-1971	M	53000	105	4	105	Jayesh	10-11-1967	M	55000		1
108	Priya	19-07-1978	F	45000	106	4	106	Vijay	20-06-1971	M	53000	105	4
104	Ramesh	15-09-1972	M	38000	106	5	106	Vijay	20-06-1971	M	53000	105	4
103	Kalpesh	31-07-1982	F	25000	102	5	102	Sanna	23-07-1982	F	35000	105	5
107	Ahmad	29-03-1979	M	25000	106	4	106	Vijay	20-06-1971	M	53000	105	4
111	Kirit	08-12-1985	M	40000	102	5	102	Sanna	23-07-1982	F	35000	105	5

Let us say all attributes suffixed with "-2" are from "s" side.

Try understanding of content of tuples in this relation

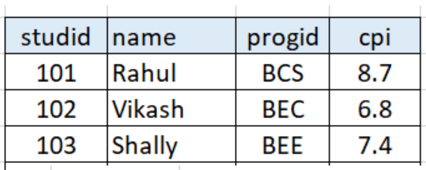
- Which gender represents gender of employee "Sanna". Is it "F" or "M"?
- What does value "Jayesh" (ename-2) mean to ename='Sanna'?
- What does value 550000 in (salary-2) mean to ename= 'Sanna'?
- How employees "Vijay" and "Jayesh" are related?

More Queries from Company database:

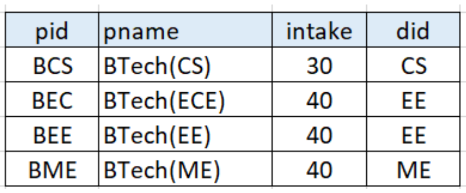
- How do you find who supervises whom?
- How do you find who works for what department?
- How do you find out who works on what project?
- How do you find out what are the projects managed by a department?

Example

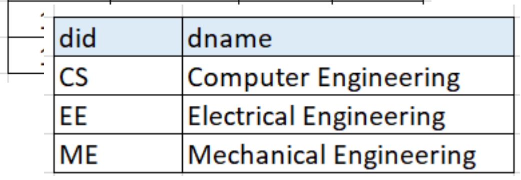
Consider XIT database. Snapshot is shown here for your reference.



studid	name	progid	cpi
101	Rahul	BCS	8.7
102	Vikash	BEC	6.8
103	Shally	BEE	7.4



pid	pname	intake	did
BCS	BTech(CS)	30	CS
BEC	BTech(ECE)	40	EE
BEE	BTech(EE)	40	EE
BME	BTech(ME)	40	ME



did	dname
CS	Computer Engineering
EE	Electrical Engineering
ME	Mechanical Engineering

Suppose we want to figure out what is name of department of in which student id = 102 is studying?

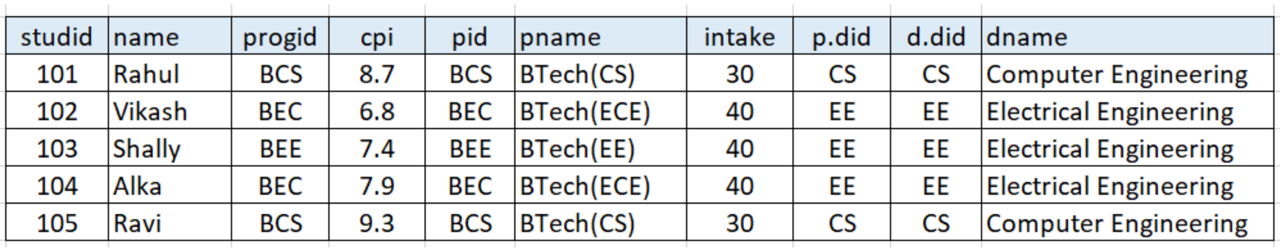
Now again the situation where we want to access tuple that contains DNAME and want to access that tuple for given SID. For this purpose, we want to have a tuple set where SID and DNAME are there. This would require travelling from SID to DNAME, and that can only be doing by joining STUDENT with PROGRAM and then with DEPARTMENT.

Below is how we would compute joined tuple-set.

$$student \underset{progid=pid}{\bowtie} program \underset{p.did=d.did}{\bowtie} department$$

```sql
SELECT * FROM student JOIN program AS p ON progid=pid
       JOIN department AS d ON p.did=d.did;
```

Below is snapshot of resultant relation.



studid	name	progid	cpi	pid	pname	intake	p.did	d.did	dname
101	Rahul	BCS	8.7	BCS	BTech(CS)	30	CS	CS	Computer Engineering
102	Vikash	BEC	6.8	BEC	BTech(ECE)	40	EE	EE	Electrical Engineering
103	Shally	BEE	7.4	BEE	BTech(EE)	40	EE	EE	Electrical Engineering
104	Alka	BEC	7.9	BEC	BTech(ECE)	40	EE	EE	Electrical Engineering
105	Ravi	BCS	9.3	BCS	BTech(CS)	30	CS	CS	Computer Engineering

Again, try understanding tuples in this relation

- How Vikash and Electrical Engineering is related?
- How Vikash and EE is related?

Required result can be computed as following:

$$r1 \leftarrow student \underset{progid=pid}{\overset{\bowtie}{\overline{\qquad\qquad}}} program \underset{p.did=d.did}{\overset{\bowtie}{\overline{\qquad\qquad}}} department$$

$$result \leftarrow \pi_{dname}(\sigma_{studid=104}\ (r1)\ )$$

The same can be expressed in SQL as following

```
SELECT dname FROM student JOIN program AS p ON progid=pid
     JOIN department AS d ON p.did=d.did
     WHERE studid='102';
```

JOIN explained in algorithmic terms

JOIN operation can further be explained in terms of an algorithm.

Consider JOIN expression given below

$$r1 \underset{< join - condition >}{\overset{\bowtie}{\overline{\qquad\qquad\qquad}}} r2$$

Following algorithm computes result of above JOIN expression. Note that algorithm no means tells how actually JOIN operation is executed by RDBMS, it simple tells what shall be result of said join expression.

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

JOIN – Recap

- When do we need JOIN? JOIN is often required

  >> When we want to deference a foreign KEY.

  >> When we want combine tuples of a relation with “related tuples”

- How do we express JOIN

Queries Writing Approach (within the operations that we have learned so far):

- Collect all attributes and relations that are either there in WHERE clause and in SELECT clause.
- Identify Relations that have required attributes
- Perform JOIN through relevant foreign keys
- Apply appropriate filter, and finally
- Perform appropriate projection
- Of course, more operations to come

Exercises

- List (prog-name, deptt-name) of programs offered at XIT
- List student name, program name of students having cpi > 7.5
- List employees (eno, pname) who work on projects controlled by dno=5

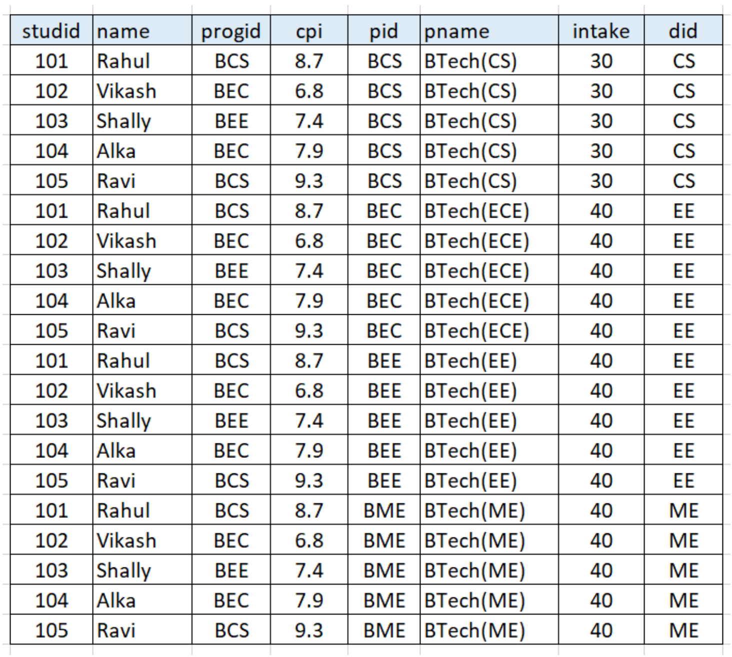
CROSS PRODUCT

Hope you also know cross product of two sets.

Algebraically it is expressed as $r1 \times r2$

Example

For example $student \times program$ results following:



studid	name	progid	cpi	pid	pname	intake	did
101	Rahul	BCS	8.7	BCS	BTech(CS)	30	CS
102	Vikash	BEC	6.8	BCS	BTech(CS)	30	CS
103	Shally	BEE	7.4	BCS	BTech(CS)	30	CS
104	Alka	BEC	7.9	BCS	BTech(CS)	30	CS
105	Ravi	BCS	9.3	BCS	BTech(CS)	30	CS
101	Rahul	BCS	8.7	BEC	BTech(ECE)	40	EE
102	Vikash	BEC	6.8	BEC	BTech(ECE)	40	EE
103	Shally	BEE	7.4	BEC	BTech(ECE)	40	EE
104	Alka	BEC	7.9	BEC	BTech(ECE)	40	EE
105	Ravi	BCS	9.3	BEC	BTech(ECE)	40	EE
101	Rahul	BCS	8.7	BEE	BTech(EE)	40	EE
102	Vikash	BEC	6.8	BEE	BTech(EE)	40	EE
103	Shally	BEE	7.4	BEE	BTech(EE)	40	EE
104	Alka	BEC	7.9	BEE	BTech(EE)	40	EE
105	Ravi	BCS	9.3	BEE	BTech(EE)	40	EE
101	Rahul	BCS	8.7	BME	BTech(ME)	40	ME
102	Vikash	BEC	6.8	BME	BTech(ME)	40	ME
103	Shally	BEE	7.4	BME	BTech(ME)	40	ME
104	Alka	BEC	7.9	BME	BTech(ME)	40	ME
105	Ravi	BCS	9.3	BME	BTech(ME)	40	ME

Try "interpreting" tuples of the result relation?

- What does values Vikash, BEC, and BCS in second tuple represents?

Logically following algorithmic shows computation of result of CROSS PRODUCT

```
result_set = NULL;
For each tuple t1 in relation r1
For each tuple t2 in relation r2
      derive new tuple t = t1 concatenate t2
      append t to result_set
```

CROSS PRODUCT in SQL

Expressed as

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

This SQL statement is equivalent to JOIN express

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

JOIN expressed in terms of CROSS PRODUCT

Mathematically JOIN can also be expressed following

$$r1 \underset{<join-condition>}{\overset{\bowtie}{\overline{\qquad\qquad}}} r2 \leftarrow \sigma_{<join-condition>}(r1 \times r2)$$

For our student-program example, it would be as following

$$student \underset{<progid=pid>}{\overset{\bowtie}{\overline{\qquad\qquad}}} program \leftarrow \sigma_{<progid=pid>}(student \times program)$$

Same can be expressed in SQL as following

```sql
SELECT * FROM student, program WHERE progid=pid;
```

Above SQL is equivalent to following SQL JOIN

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

However, expressing JOIN through CROSS JOIN in SQL is not a preferred way.

Important drawback of this is expressivity - JOIN in SQL gets mixed with “SELECT”. Other is query optimizer may not detect a join, and hence slower to execute. Though should not be issue with modern query optimizers.