Relational Database

Relational Algebra – SQL

Bipin C. DESAI

Attributes and Domains

An object or entity is characterized by its properties (attributes or data elements). The set of allowable values for an attribute is the domain of the attribute.

Domain. We define a domain, D_i, as a set of values of the same data type.
Each Attribute is defined on some underlying domain; more than one attribute may share a domain.

Tuples, Relations and Their Schemes

A relation consists of a homogeneous set of tuples.

Since each tuple in a relation represents an identifiable instance of an entity (object type), duplicate tuples are not allowed.

The number of attributes in the relation gives the **degree** or **arity** of the relation.

The **cardinality** of an instance of a relation, at a point in time, is derived from the count of the tuples in the instance.

The cardinality could change over time

Relation Representation

APPLICANT:

Name	Age	Profession
John Doe	55	Analyst
Mirian Taylor	31	Programmer
Abe Malcolm	28	Receptionist
Adrian Cook	33	Programmer
Liz Smith	33	Manager

Key. A subset of attributes X of a relation R(**R**), X ∈ **R**, with the following time independent properties is called the **key** of the relation:

Unique Identification: The values of X uniquely identify a tuple. $s[X] = t[X] \Rightarrow s = t$.

Non-redundancy: No proper subset of X has the unique identification property.

There may be more than one key in a relation; all such keys are known as **candidate keys**.

One of the candidate keys is chosen as the **primary key**; the others are known as alternate keys.
An attribute that forms part of a candidate key of a relation is called a **prime attribute**.

EMPLOYEE (*Emp#, Emp_Name, Profession*)
PRODUCT (*Prod#, Prod_Name, Prod_Details*)
JOB_FUNCTION (*Job#, Title*)
ASSIGNMENT (*Emp#, Prod#, Job#*)

EMPLOYEE

Emp#	Name	Profession
101	Jones	Analyst
103	Smith	Programmer
104	Lalonde	Receptionist
106	Letitia	VP Marketing
107	Evan	VP R & D
110	Drew	VP Operation
112	Smith	Manager

PRODUCT

Prod#	Prod_Name	Prod_Details
HEAP1	HEAP_SORT	ISS module
BINS9	BINARY_SEARCH	ISS/ R module
FM6	FILE_MANAGER	ISS/ R-PC subsys
B++1	B++_TREE	ISS/ R turbo sys
B++2	B++_TREE	ISS/ R-PC turbo

JOB_FUNCTION

Job#	Title
1000	CEO
700	Chief Programmer
800	Manager
600	Analyst

ASSIGNMENT

Emp#	Prod#	Job#
107	HEAP1	800
101	HEAP1	600
110	BINS9	800
103	HEAP1	700
101	BINS9	700
110	FM6	800
107	B++1	800

The attributes ***Emp#***, ***Prod#***, and ***Job#*** in the relation ASSIGNMENT are known as **foreign** keys.

A null value for an attribute:

- a value that is either not known at the time, or
- the value is known but not recorded, or
- no value is applicable for some tuples

P:

Id	Name
101	Jones
@	Smith
104	Lalonde
105	Letitia
107	Evan
110	Drew
@	Smith

P:

Id	Name
101	Jones
103	Smith
104	Lalonde
105	Letitia
107	Evan
110	Drew
112	Smith

Emp#	Name	Manager
101	Jones	@
103	Smith	110
104	Lalonde	107
107	Evan	110
110	Drew	112
112	Smith	112

Integrity rule 1 (Entity Integrity). If attribute *A* of relation R(R) is a component of the primary key of R(R), then *A* cannot accept null values.

Integrity Rule 2 (Referential Integrity). Given two relations R and S. Suppose R refers the relation S via a set of attribute which forms the primary key of S and, hence, this set of attributes forms a foreign key in R. Then, the value of the foreign key in a tuple in R must either be equal to the primary key of a tuple of S or be entirely null.

- All tuples which contain references to the deleted tuple should also be deleted. **cascading** deletion

- A tuple which is referred by other tuples in the database cannot be deleted.

- In the third option, the tuple is deleted, however, the foreign key attributes of all referencing tuples are set to null(otherwise “dangling” pointers!)

- All tuples which contain references to the deleted tuple should also be deleted. This is **cascading** deletion

- A tuple which is referred by other tuples in the database cannot be deleted.

- In the third option, the tuple is deleted, however, the foreign key attributes of all referencing tuples are set to null (otherwise “dangling” pointers!)

Query Languages

- The Relational model supports simple, powerful query languages which:
  - Have formal foundation based on logic.
  - Allows for implementation which can be optimized.
- Allow data access and modification.
- These languages **are not general purpose** programming languages, however, most DBMS vendors have added their own enhancements to improve its functionality.

Relational Algebra, Calculus

Relational Algebra(RA) and **Relational Calculus(RC)** are the foundation for implemented languages (e.g. SQL)

- RA is operational, and is useful for representing the plan of execution of a query.
- RC is declarative allowing users to describe what they want. (i.e. it is non-operational)

- **Understanding RA & RC is vital to the understanding of SQL and query processing!**

Your textbook may not cover RC!

Relational algebra is a collection of operations to manipulate relations.

Basic Operations: Three of these four basic operations - **union**, **intersection** and **difference** - require that operand relations be **union-compatible**. (Same number (and order)of attributes on identical (at least compatible) domains

Q:

Id	Name
101	Jones
105	Letitia
107	Evan
110	Drew

P:

Id	Name
101	Jones
103	Smith
104	Lalonde
105	Letitia
107	Evan
110	Drew
112	Smith

UNION ($\cup$) If we assume that P(**P**) and Q(**Q**) are two union-compatible relations, then:

The union of P(**P**) and Q(**Q**) is the set-theoretic union of P(**P**) and Q(**Q**).

The resultant relation, R = P $\cup$ Q, has tuples drawn from P and Q, such that

$$R = \{t \mid t \in P \vee t \in Q\} \text{ and}$$

$$\max(P, Q) \leq R \leq P + Q$$

Union operation is associative and commutative

$$P \cup Q \cup S = P \cup (Q \cup S) = (P \cup Q) \cup S = (P \cup S) \cup Q$$

Q:

Id	Name
101	Jones
105	Letitia
107	Evan
110	Drew

P:

Id	Name
101	Jones
103	Smith
104	Lalonde
105	Letitia
107	Evan
110	Drew
112	Smith

P ∪ Q:

Id	Name
101	Jones
103	Smith
104	Lalonde
105	Letitia
107	Evan
110	Drew
112	Smith

Q ∪ P:

Id	Name
101	Jones
103	Smith
104	Lalonde
105	Letitia
107	Evan
110	Drew
112	Smith

DIFFERENCE (-)

The difference operation removes common tuples from the first relation.

$R = P - Q$ such that

$R = \{ t \mid t \in P \;\&\; t \notin Q\}$ and $0 \leq R \leq P$

Difference operation is non-associative and non-commutative.

Is $P - Q = Q - P$?

Is $P - (Q - S) = (P - Q) - S$?

Q:

Id	Name
101	Jones
105	Letitia
107	Evan
110	Drew

P:

Id	Name
101	Jones
103	Smith
104	Lalonde
105	Letitia
107	Evan
110	Drew
112	Smith

P - Q:

Id	Name
103	Smith
104	Lalonde
112	Smith

Q - P:

Id	Name

INTERSECTION ($\cap$)

The intersection operation selects the common tuples from the two relations.

$R = P \cap Q$ where

$R = \{ t \mid t \in P \;\&\; t \in Q\}$ and $0 \leq R \leq \min(P , Q)$

The intersection operation is really unnecessary as it can be very simply expressed as:

$$P \cap Q = P - (P - Q)$$

$$Q \cap P = Q - (Q - P)$$

Is $P - (P - Q) = Q - (Q - P)$?

P:

Id	Name
103	Smith
104	Lalonde
105	Letitia
107	Evan
110	Drew
112	Smith

Q:

Id	Name
101	Jones
105	Letitia
107	Evan
110	Drew

$P \cap Q$:

Id	Name
105	Letitia
107	Evan
110	Drew

$Q \cap P$:

Id	Name
105	Letitia
107	Evan
110	Drew

P:

Id	Name
103	Smith
104	Lalonde
105	Letitia
107	Evan
110	Drew
112	Smith

Q:

Id	Name
101	Jones
105	Letitia
107	Evan
110	Drew

P - Q:

Id	Name
103	Smith
104	Lalonde
112	Smith

Q – P :

Id	Name
101	Jones

P-(P- Q)

Id	Name
105	Letitia
107	Evan
110	Drew

Q-(Q- P)

Id	Name
105	Letitia
107	Evan
110	Drew

RENAMING (ρ)

The renaming operation ρ* is used to rename relations or its attributes. The operation:

$$\rho(R(\text{modattributes}), \text{rel\_exp})$$

takes a relation expression and the result is named R with some of the attributes, specified in the modattributes, are renamed

The format of modattributes is:

modattributes ::= <oldname → newname>|

<position → newname> <,modattributes>

*ρ *rho* is the 17th letter of the Greek alphabet

Employee(Emp#, Ename, Address, Phone, DOB)

$\rho(Q_{\text{EmpID} \rightarrow \text{Emp\#, Ename} \rightarrow \text{Name}}\ \Pi_{\text{Emp\#,Ename}}\ \text{EMPLOYEE})$

Q:

Id	Name
101	Jones
105	Letitia
107	Evan
110	Drew

CARTESIAN PRODUCT ($\times$)

The extended cartesian or simply the cartesian product of two relations is the concatenation of tuples belonging to the two relations. $R = P \times Q$

The scheme of the result relation is given by: $\mathbf{R} = \mathbf{P} || \mathbf{Q}$.

The degree of the result relation is given by:
$|\mathbf{R}| = |\mathbf{P}| + |\mathbf{Q}|$.

Q:

Id	Name
101	Jones
105	Letitia
107	Evan
110	Drew

P:

Id	Name
101	Jones
103	Smith
104	Lalonde
105	Letitia
107	Evan
110	Drew
112	Smith

Id	Name	Id	Name
101	Jones	101	Jones
101	Jones	103	Smith
101	Jones	104	Lalonde
…			
	…		
		…	
			….
110	Drew	112	Smith

PROJECTION ($\prod$) $\prod_X$ R

It should be noted that the projection operation reduces the arity if the number of attributes in X is less than the arity of the relation. It may also reduce the cardinality of the result relation since duplicate tuples are removed.

P:

Id	Name
101	Jones
103	Smith
104	Lalonde
105	Letitia
107	Evan
110	Drew
112	Smith

$\pi_{Name}P$:

Name
Jones
Smith
Lalonde
Letitia
Evan
Drew

SELECTION (σ)

The selection operation,yields a "horizontal subset" of a given relation. Any finite number of predicates connected by boolean operators may be specified in the selection operation.

P:

Id	Name
101	Jones
103	Smith
104	Lalonde
105	Letitia
107	Evan
110	Drew
112	Smith

$\sigma_{Id<106}$ P:

Id	Name
101	Jones
103	Smith
104	Lalonde
105	Letitia

JOIN (⊳⊲)

The join operator, as the name suggests, allows the combining of two relations to form a single new relation.

- first compute the cartesian product
- followed by selecting those tuples where the common attribute(s) has(have) the same value(s).

Project (Proj#, Pname, Pleader)   Assign(P#, E#)

Employee(Emp#, Ename, Address, Phone, DOB)

- **Get Emp# of employees working on Proj# comp353.**
- **Get complete details of employee working on comp353.**
- **Get complete detatils of employes working on the Database project.**
- **Get complete details of employees working on both comp353 and comp354**
- **Get Emp# (complete details) of employees working on two projects.**
- **Get names of employees working in projects where Ma is the project leader.**

Project (Proj#, Pname, Pleader)   Assign(P#, E#)

Employee(Emp#, Ename, Address, Phone, DOB,)

❖ **Get Emp# of employees working on Proj# comp353.**

$$\Pi_{E\#}\ (\sigma_{P\#=comp353}(Assign))$$

❖ **Get complete details of employee working on comp353.**

$$Employee\ \triangleright\!\triangleleft_{\ Emp\#=E\#}\ \Pi_{E\#}\ (\sigma_{P\#=comp353}(Assign))$$

Project (Proj#, Pname, Pleader) **Assign(P#, E#)**

Employee(Emp#, Ename, Address, Phone, DOB,)

- **Get complete details of employees working on the Database project(s) – a project name.**

$$X = \Pi_{E\#} ((Assign) \bowtie_{Proj\#=P\#} (\sigma_{Pname=\text{“Database”}}(Project)))$$

$$Employee \bowtie_{Emp\#=E\#} X$$

Combining in one RA expression:

$$Employee \bowtie_{Emp\#=E\#} \Pi_{E\#} ((Assign) \bowtie_{Proj\#=P\#} (\sigma_{Pname=\text{“Database”}}(Project)))$$

Project (Proj#, Pname, Pleader)   Assign(P#, E#)

Employee(Emp#, Ename, Address, Phone, DOB,)

- **Get complete details of employees working on both comp353 and comp354**

$$\text{Employee} \rhd\lhd_{\text{Emp\#=E\#}} \Pi_{\text{E\#}} (\sigma_{\text{P\#=comp353}}(\text{Assign})) \cap$$
$$\text{Employee} \rhd\lhd_{\text{Emp\#=E\#}} \Pi_{\text{E\#}} (\sigma_{\text{P\#=comp354}}(\text{Assign}))$$
$$\text{or Employee} \rhd\lhd_{\text{Emp\#=E\#}} (\Pi_{\text{E\#}} (\sigma_{\text{P\#=comp353}}(\text{Assign})) \cap \Pi_{\text{E\#}} (\sigma_{\text{P\#=comp354}}(\text{Assign}))$$

Project (Proj#, Pname, Pleader)   Assign(P#, E#)

Employee(Emp#, Ename, Address, Phone, DOB,)

- **Get complete details of employees working on (any)two projects**

  $\rho(A1(P1\#, E1\#), Assign)$

  $X = A1 \times Assign \qquad Y = \Pi_{E\#} (\sigma_{P\# \neq P1\# \wedge E\# = E1\#} (X))$

  $Employee \bowtie_{Emp\# = E\#} Y$

Combining in one RA expression:

$Employee \bowtie_{Emp\# = E\#} (\Pi_{E\#} (\sigma_{P\# \neq P1\# \wedge E\# = E1\#} (\rho(A1(P1\#, E1\#), Assign) \times Assign)))$

Project (Proj#, Pname, Pleader)   Assign(P#, E#)

Employee(Emp#, Ename, Address, Phone, DOB,)

- **Get complete details of employees working in projects where Ma(an employee name) is the project leader.**

$$X= (\sigma_{Ename=\text{“}Ma\text{”}}(Employee)))$$

$$Y=\Pi_{Proj\#}\ (Project \rhd\lhd_{Emp\#=Pleader}\ X)$$

$$Z=\Pi_{E\#}\ (\ Assign\ \rhd\lhd_{Proj\#=P\#}\ Y)$$

$$Employee\ \rhd\lhd_{Emp\#=E\#}\ Z$$

$$Employee\ \rhd\lhd_{Emp\#=E\#}\ (\Pi_{E\#}\ (\ Assign\ \rhd\lhd_{Proj\#=P\#}\ (\Pi_{Proj\#}\ (Project \rhd\lhd_{Emp\#=Pleader}\ (\sigma_{Ename=\text{“}Ma\text{”}}(Employee)))\ ))))$$

Complete set of RA operations

$$\{\sigma, \; \Pi, \; \cup, \; -, \; \times\}$$

The above is a complete set of RA operations.

The others could be expressed as a sequence

of operations from this set.

$R \cap S \equiv (R \cup S) - ((R - S) \cup (S-R))$

$R \bowtie_B S \equiv \sigma_B (R \times S)$ etc

Definition: **Theta-join**. The **theta-join** of two relations P(**P**) and Q(**Q**) is defined as

$$R = P \bowtie_B Q$$

such that $R = \{t \mid t_1 || t_2 \wedge t_1 \in P \wedge t_2 \in Q \wedge B\}$

where B is a selection predicate consisting of terms of the form: $(t_1[A_i]\ \theta\ t_2[B_i])$ for i = 1, 2, ..., n,

where θ_i is some comparison operator ($\theta \in \{=, \neq, <, \leq, >, \geq\}$), and A_i and B_i are some domain compatible attributes of the relation schemes **P** and **Q** respectively.

$$0 \leq |R| \leq |P| * |Q|$$

$$|\mathbf{R}| = |\mathbf{P}| + |\mathbf{Q}|$$

Two common and very useful variants of the join are the **equi-join** and the **natural-join**.

If two relations that are to be joined have no domain compatible attributes, then the natural join operation is equivalent to a simple cartesian product.

-The equi-join and the theta joins are "horizontal subsets" of the cartesian product.

-The natural join is equivalent to an equi-join with a subsequent projection to eliminate the duplicate attributes.

SQL is both the data definition and data manipulation language of the relational database systems

Create table <relation> (<attribute list>, <integrity constraint list>)

Where the <attribute list> is specified as:

<Attribute list> ::= <attribute name> (<data type>)[not null][,<attribute list>]

and <integrity constraints list> is specified as:

<Integrity constraint list> ::= <integrity1>|<integrity constraint list>

and <integrity> could be a primary key(a1, a2, ... am) , not null or a named constraint.

```
create table EMPLOYEE
    (Empl_No   integer not null,
     Name      char(25),
     Skill     char(20),
     Pay_Rate  decimal(10,2)
     Primary key Empl_No )
```

```
select [distinct] <target list>
from  <relation list>
[where <predicate>]
```

Database Schema

- Employee(Name, Sin, Dept#, MGRSIN)
- Dept(Dname, Dept#, MgrSin, Bcode)
- Project(Pname, Proj#, Dept#, Lab)
- Assign(Proj#, EmpSin, Hours)
- EmpDet(Sin, Address, Salary, DOB)
- EmplDepd(Sin, DepName, HowR)

Queries

Employee(Name, Sin, Dept#, MGRSIN)
Dept(Dname, Dept#, MgrSin, Bcode)
Project(Pname, Proj#, Dept#, Lab)

- Names of Employees in Dept 101?

```
select Name
from Employee
where Dept#= 101
```

- Details of Employee in Dept. 101?

```
select *
from Employee
where Dept#= 101
```

Employee(Name, Sin, Dept#, MGRSIN)
Dept(Dname, Dept#, MgrSin, Bcode)
Project(Pname, Proj#, Dept#, Lab)

- For all projects in the Software Engg. Lab, find the DName, Manager's name?

```
select Dname, Name
from Employee e, Dept d, Project p
where Lab = 'Software Engg.'
        and p.Dept#=d.Dept#
        and d.MgrSin=e.Sin
```

- For all projects in the Software Engg. Lab, find the DName, Manager's address etc.

Employee(Name, Sin, Dept#, MGRSIN)
Dept(Dname, Dept#, MgrSin, Bcode)
Project(Pname, Proj#, Dept#, Lab)
EmpDet(Sin, Address, Salary, DOB)

- For all projects in the Software Engg. Lab, find the DName, Manager's name, address etc.

```
select Dname, Name,Address, Salary,DOB
from Employee e, Dept d, Project p, EmpDet t
where Lab = 'Software Engg.'
          and p.Dept#=d.Dept#
          and d.MgrSin=e.Sin
          and e.Sin=t.Sin
```

Query Tree

Consider the relation:

Employee(Emp#, Ename, City, Phone, YOB,)

Suppose we want to find Emp# and names of employees who live in NDG(an address) and who were born in 1971(YOB).

We can express this query in one of the following ways:

$\Pi_{\text{Emp\#, Ename}} (\sigma_{\text{City='NDG'} \wedge \text{YOB =1971}}(\textbf{EMPLOYEE}))$

or

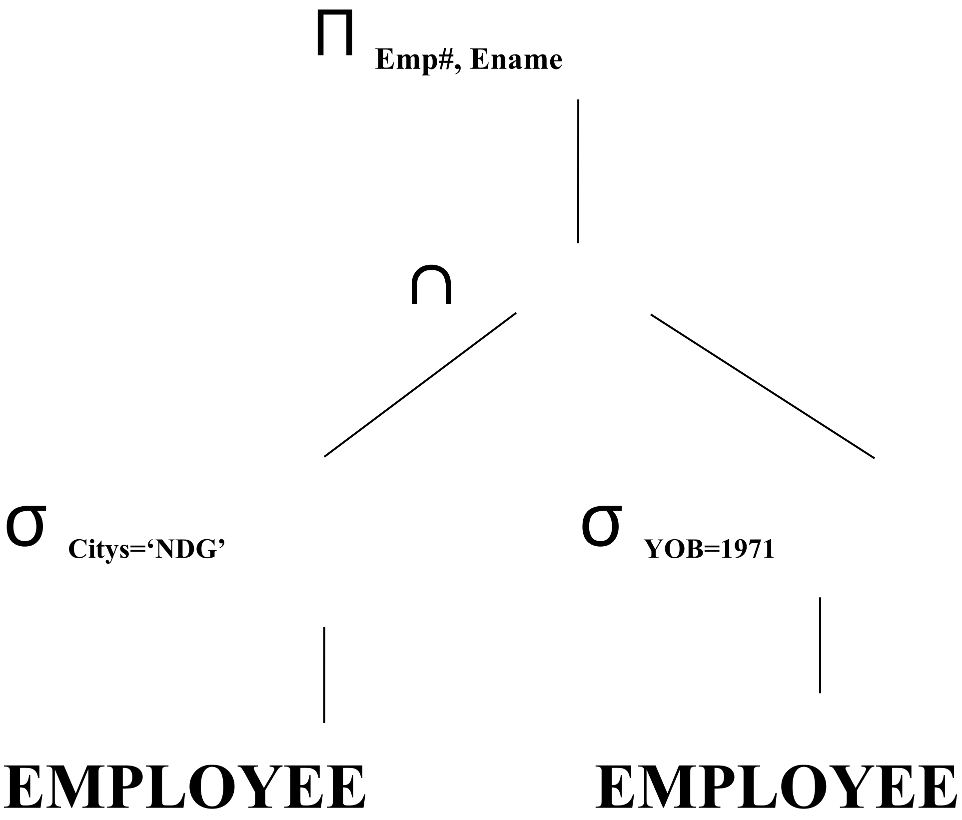
$\Pi_{\text{Emp\#, Ename}} (\sigma_{\text{City='NDG'}}(\textbf{EMPLOYEE}) \cap \sigma_{\text{YOB =1971}}(\textbf{EMPLOYEE}))$

or

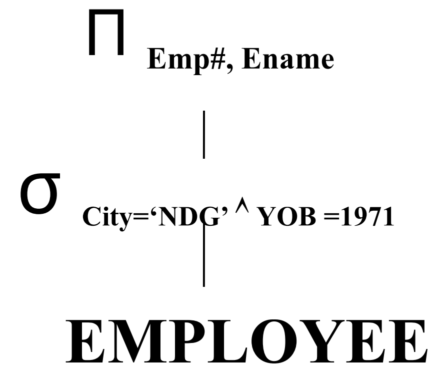
$\Pi_{\text{Emp\#, Ename}}\ \sigma_{\text{City='NDG'}}(\textbf{EMPLOYEE}) \cap$

$\Pi_{\text{Emp\#, Ename}}\ \sigma_{\text{YOB =1971}}(\textbf{EMPLOYEE})$

$\Pi_{\text{Emp\#, Ename}} (\sigma_{\text{City='NDG'} \wedge \text{YOB}=1971}(\text{EMPLOYEE}))$

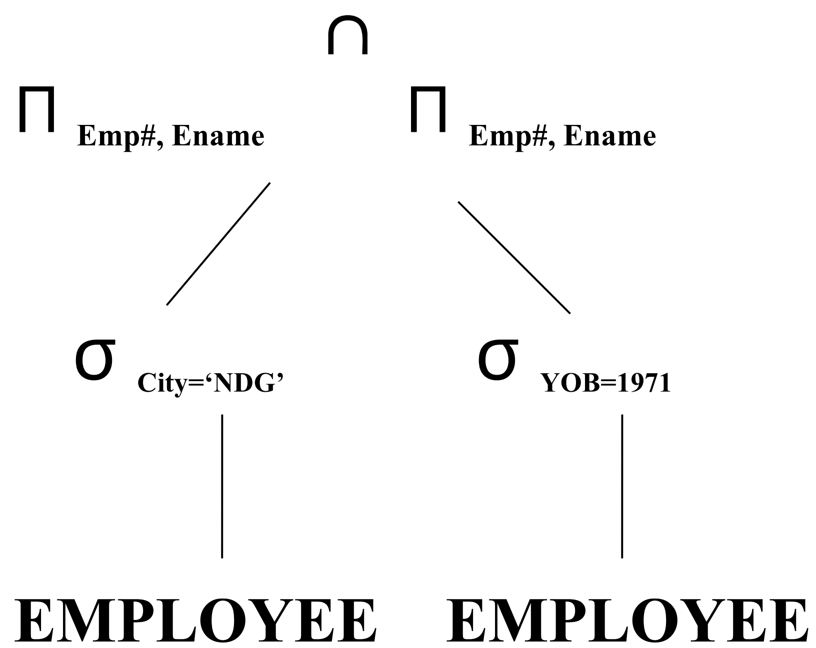


$\Pi_{\text{Emp\#, Ename}} (\sigma_{\text{City='NDG'}}(\text{EMPLOYEE}) \cap \sigma_{\text{YOB}=1971}(\text{EMPLOYEE}))$



$\Pi_{\text{Emp\#, Ename}}\ \sigma_{\text{City='NDG'}}(\text{EMPLOYEE}) \cap$

$\Pi_{\text{Emp\#, Ename}}\ \sigma_{\text{YOB}=1971}(\text{EMPLOYEE})$



Project (Proj#, Pname, Pleader) 100 projects
Empl (Emp#, Ename, City, Ph, DOB,) 500 employees
Assign (P#, E#) 1500 assignments

Get complete details of employees working on the DB project(s).

Suppose there are 10 DB projects, distribution is uniform.

Av. of 3 projects for each employee; Av. of 15 employees per project

Number of assignments to DB project is 150 (10% of assignments).

If no employee works on more than one DB project,

then maximum number of tuples in output would be 150.

Two possible ways of expressing this query

$$\Pi_{Emp\#,\ Ename,\ City,\ Ph,\ DOB}(\sigma_{E\#=Emp\#}(\sigma_{P\#=Proj\#}(\sigma_{Pname=\text{“DB”}}(Empl \times Assign \times Project))))$$

$$Empl \ \rhd\lhd_{Emp\#=E\#} (\Pi_{E\#}((Assign) \rhd\lhd_{Proj\#=P\#} (\sigma_{Pname=\text{“DB”}}(Project))))$$

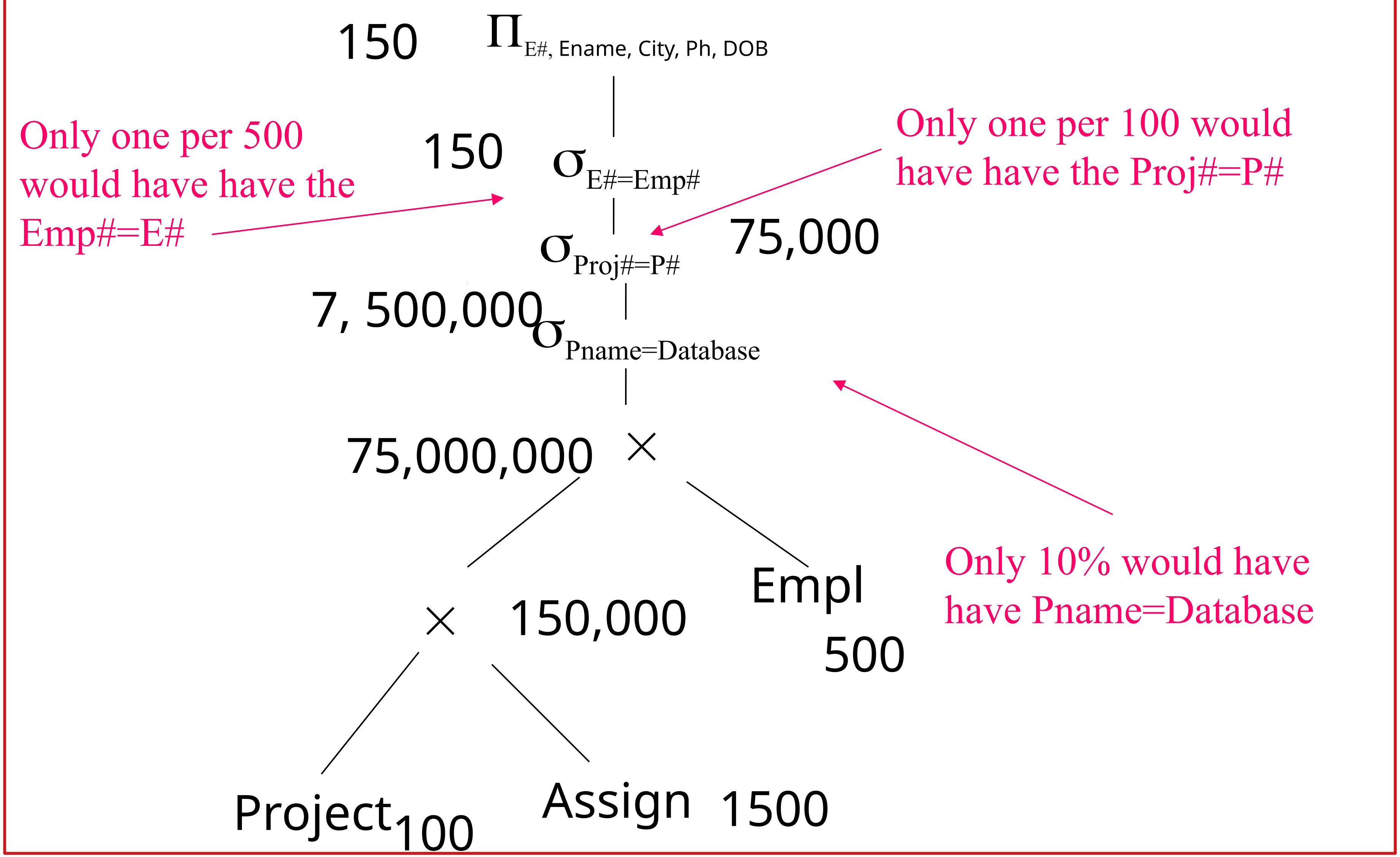
Pr	Pn
P1	DB
P2	X
.	
.	

P	E
P1	E1
P2	E2
.	
.	

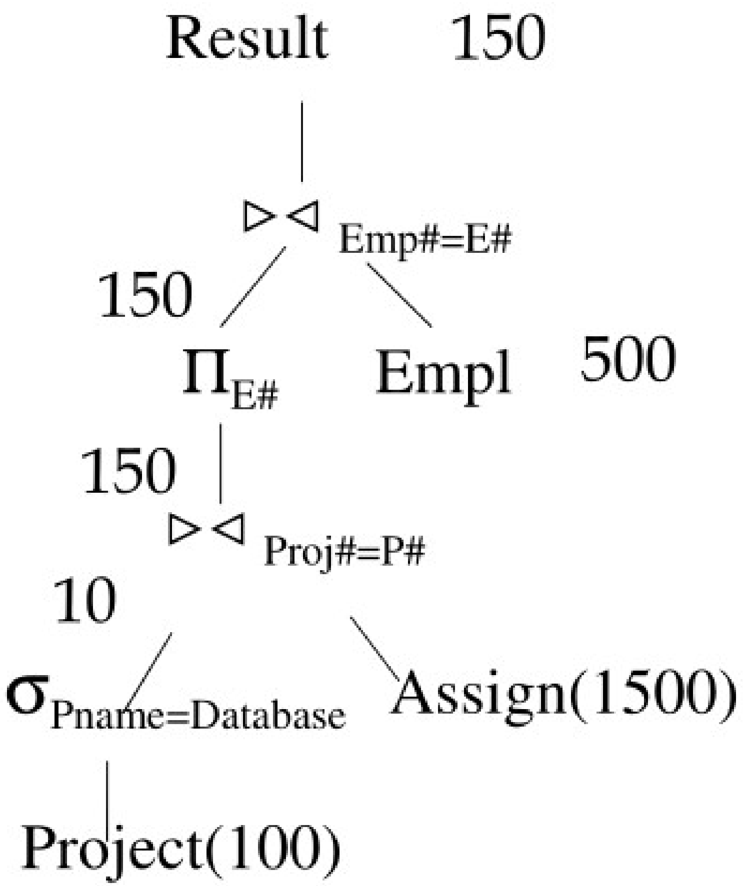
Em	En
E1	N1
E2	N2
.	
.	

Pr	Pn	P	E	Em	En
P1	DB	P1	E1	E1	N1
				E2	N2
		....			
		P2	E2	E1	N1
				E2	N2
		....			
P2	X	P1	E1	E1	N1
				E2	N2
		....			
		P2	E2	E1	N1
				E2	N2
		....			

$$\Pi_{E\#,\ Ename,\ City,\ Ph,\ DOB}(\sigma_{P\#=Database}\ (Empl \times Assign \times Project))$$



$$\text{Empl} \bowtie_{\text{Emp\#=E\#}} (\Pi_{\text{E\#}} ((\text{Assign}) \bowtie_{\text{Proj\#=P\#}} (\sigma_{\text{P\#=Database}}(\text{Project}))))$$



Division (÷)

P(P):

A	B
a_1	b_1
a_1	b_2
a_2	b_1
a_3	b_1
a_4	b_2
a_5	b_1
a_5	b_2

Q(Q):

B
b_1
b_2

R(R)(result):

A
a_1
a_5

The result of dividing P by Q is the relation R which has two tuples. For each tuple in R, its product with the tuples of Q must be in P. In our example (a_1,b_1) and (a_1,b_2) must both be tuples in P; the same is true for (a_5,b_1) and (a_5,b_2).

The Cartesian product of Q and R is a subset of P.

P(**P**):

A	B
a_1	b_1
a_1	b_2
a_2	b_1
a_3	b_1
a_4	b_2
a_5	b_1
a_5	b_2

Q(**Q**):

B
b_1

R(**R**) is:

A
a_1
a_2
a_3
a_5

Q(**Q**):

B
b_1
b_2
b_3

R(R) is:

A

Q(**Q**):

B

R(R) is:

A
a_1
a_2
a_3
a_4
a_5

The division operation is useful where a query involves the phrase "*for all objects having all of the specified properties*" .

Project (Proj#, Pname, Pleader) **Assign(P#, E#)**
Employee(Emp#, Ename, Address, Phone, DOB,)

Get complete details of employees working on *all* Database projects.

Find the Proj# of all Database project as DBPROJNO

$$\rho_{1 \rightarrow P\#}\ (\text{DBPROJNO}, \Pi_{\text{Proj\#}}(\sigma_{\text{Pname=Database}}\text{Project}))$$

Find the specified details for the required employees by dividing **Assign** by DBPROJNO and join the result with **Employee**.

$$\text{ASSIGN} \div \text{DBPROJNO} \bowtie_{\text{Emp\#=E\#}} \textbf{Employee}$$

Project (Proj#, Pname, Pleader)   Assign(P#, E#)
Employee(Emp#, Ename, Address, Phone, DOB,)

Get complete details of employees working exactly on all DB projects.

Find the Proj# of all Database project as DBPROJNO

$\rho_{1 \rightarrow P\#}$ (DBPROJNO, $\Pi_{Proj\#}(\sigma_{Pname=Database}$Project))

Find those employees who work on all DB projects by dividing Assign by DBPROJNO (some of them work on other projects as well!).

ALLDB =ASSIGN $\div$ DBPROJNO

Find those tuples not involving assignments to DB projects

NOTDBONLY =ASSIGN – DBPROJNO $\times$ ALLDB

Required employees: ONLYDB = ALLDB - $\Pi_{E\#}$ NOTDBONLY

Result is: ONLYDB $\bowtie_{Emp\#=E\#}$ **Employee**

Division is not a basic operation

We can re-write **P(AB)÷Q(B)** by :

$\Pi_A P - \Pi_A(\Pi_A P \times Q - P)$

A	B
a_1	b_1
a_1	b_2
a_2	b_1
a_3	b_1
a_4	b_2
a_5	b_1
a_5	b_2

B
b_1
b_2

$\Pi_A P$

A
a_1
a_2
a_3
a_4
a_5

$\Pi_A P \times Q$

A	B
a_1	b_1
a_1	b_2
a_2	b_1
a_2	b_2
a_3	b_1
a_3	b_2
a_4	b_1
a_4	b_2
a_5	b_1
a_5	b_2

$\Pi_A P \times Q - P$

A	B
a_2	b_2
a_3	b_2
a_4	b_1

$\Pi_A(\Pi_A P \times Q - P)$

A
a_2
a_3
a_4

$\Pi_A P - \Pi_A(\Pi_A P \times Q - P)$

A
a_1
a_5

Some special characters used in DB & HTML codes

∧∨∃¬∃∀Σ Π∪ ∩ ⊆⊂ ⊃ ⊇⌈ ⌉⌊ ⌋≡≠∈∉→ σ π ρ θ ϕ Γ×÷ ≤ ≥∣ ▷◁

∧ ∧ ∨ ∨ ∃ ∃ ¬∃ ¬ ∃ ∀ ∀
Σ ∑ ∏ ∏ ∪ ∪ ∩ ∩ ⊆ ⊆ ⊂ ⊂
⊃ ⊃ ⊇ ⊇ ⌈ &rcel; ⌉ ⌈ ⌊ ⌋ ⌋ ⌊
≡ ≡ ≠ ≠ ∈ ∈ ∉ ∉ → →
σ σ π π ρ ρ θ θ ϕ φ
ΓΓ × × ÷ ÷ ≤ ≤ ≥≥
∣ ∣ ▷◁ ⋈

see confsys.encs.concordia.ca/CrsMgr/html-symbols.html