Database Systems – Models and Query Languages

Hasan M. Jamil

Department of Computer Science
University of Idaho
USA

The Relational Model

A Brief Review of Set Theory:

- A *set* is a well-defined collection of objects.
- Represented by a list of elements called *members*.
- The *intension* of a set defines the permissible occurrences by specifying a membership condition.
- The *extension* specifies one possible occurrences by an explicit list of members.

Example:

Intension of set G:
$\{$g | g is an odd positive integer less than 20$\}$
Extension of set G:
$\{1, 3, 5, 7, 9, 11, 13, 15, 17, 19\}$

Example:

Let $G = \{$Oracle, IBM$\}$ be the set of companies producing hierarchical databases, and let $H = \{$Oracle, MicroSoft, Sun$\}$ be the set of companies producing relational databases. Also let $K = \{$Relational, Hierarchical$\}$ be the set of database systems. Then

$G \cap H = \{$Oracle$\}$ – companies producing both

$G \cup H = \{$Oracle, IBM, MicroSoft, Sun$\}$ – companies producing one of the two or both

Example:

The cartesian product of two sets, say G and K (denoted $G \times K$), is defined in terms of ordered pairs of 2-tuples. The product $G \times K$ is the set consisting of all ordered pairs (g,k)* such that $g \in G$ and $k \in K$. Hence

$G \times K = \{$(Oracle, Relational), (Oracle, Hierarchical), (IBM, Relational), (IBM, Hierarchical)$\}$

whereas,

$K \times G = \{$(Relational, Orcale), (Hierarchical, Oracle), (Relational, IBM), (Hierarchical, IBM)$\}$

and hence are completely different sets.

Note: Cartesian product creates (pairing) *relations*, or relationships, among sets of objects. Hence for any two given sets, a pairing relation is a subset of the cartesian product of the sets involved in the relationship.

*Note that g and k are used as variables here.

Example:

The difference of two sets G and H (denoted G $-$ H) is the set containing all elements that are members of G but not of H. Thus

$G - H = \{\text{IBM}\}$

$H - G = \{\text{MicroSoft, Sun}\}$

and hence $G - H \neq H - G$ in general.*

*Whereas $G \cup H = H \cup G$, and $G \cap H = H \cap G$.

Relational Databases

- A set of attributes define an object or entity.
- Attributes in relational model corresponds to fields in file systems.
- A set of permissible values is said to consitute a domain.
- Attributes have an underlying domain from which it is allowed to draw its values.

Example:

Let *Age* be an attribute, and *P_age*:{x | x is a positive integer and $0 \leq x \leq 150$} be a set of permissble values (intension of a set). Then *P_age* may be used as a domain for the attribute *Age*.

Definition: A domain D is a set of values of the same data type.

Domains may be simple or composite. For any two domains D_i and D_j, $D_i \cap D_j$ need not be empty.

Entity types having n attributes may be represented by an ordered set of these attributes, called an n-tuple.

Let e be an entity being described using n attributes. If the attributes $A_1, \ldots, A_n$ draw values from associated domains $D_1, \ldots, D_n$ respectively, then the representation of the entity e must be a member of the set $D_1 \times \ldots \times D_n$, as this set contains all possible ordered n-tuples, or relationships among the attribute elements.

Relations

If G = {Oracle, IBM} is a set of vendors and K = {Relational, Hierarchical}is a set of products, a relation over G and K is always a subset of all possible relationships we can form among G and K which is precisely G $\times$ K.

Mathematically, for domains $D_1, \ldots, D_n$, a relation r is defined as

$$r \subseteq D_1 \times \ldots \times D_n$$

Every relation has two components – schema and instances.

Schema:

- Every relation in a database has a structure or type, called the scheme.
- Every relation scheme has a name.
- A relation scheme is a list of attribute names and associated domains.
- Scheme is time invariant.
- The length of the list of attributes is called the degree (or arity) of the relation.

Instance:

- The instance of a relation is a possible and permissble set of relationships of domain elements.
- Its time varying.
- The size of the instance is called the cardinality (the number of n-tuples) of the relation.

If P be a relation scheme, then the instance corresponding to P is usually denoted by p. A relation is usually represented by p(P), instance p over scheme P.

The database schema is the set of relation schemes in the database and the database instance is the set of relation instances in the database.

Consistent Relations

A relation is said be consistent if it satisfies the following two integrity rules:

Entity Integrity: An attribute *A* of a relation $r(R)$ cannot accept null values if it is prime (is a member of a candidate key).

Referential Integrity: If a set of attributes X in $r(R)$ is a foreign key for another relation $s(S)$ (i.e., X is the primary key of $s(S)$) then X cannot accept a value non existent in $s(S)$ (i.e., the value must be a member of $\Pi_X(s)$), or must be entirely null.

Basic Relational Operations

Union compatibility of relations:

Two relations P and Q are said to be union compatible if they have the same degree and the corresponding domains are identical.

Example:

Let P = $\{$Id, Name$\}$ and Q = $\{$Id, Name$\}$ be the schemes of two rekations P and Q respectively. Let the instances of P and Q be as follows:

p:

Id	Name
101	Jones
103	Smith
104	Pierre
107	Evan
110	Drew
112	Smith

q:

Id	Name
103	Smith
104	Pierre
106	Byron
110	Drew

Degree of P, denoted deg(P), is 2 and cardinality, denoted $|p|$, is 6. P and Q are union compatible. How about Q?

Union ($\cup$):

Let P and Q be two union compatible relations. Then

$r = p \cup q = \{t \mid t \in p \text{ or } t \in q\}$,
$\max(|p|,|q|) \leq |r| \leq |p| + |q|$,
$R = P$ or Q, and
$deg(R) = deg(P)$ or $deg(Q)$.

r:

Id	Name
101	Jones
103	Smith
104	Pierre
106	Byron
107	Evan
110	Drew
112	Smith

$deg(R) = 2$, $|r| = 7$.

Basic Operators:

<u>Intersection ($\cap$)</u>:

Let P and Q be two union compatible relations. Then

$r = p \cap q = \{t \mid t \in p \text{ and } t \in q\}$,
$0 \leq |r| \leq \min(|p|,|q|)$,
R = P or Q, and
deg(R) = deg(P) or deg(Q).

r:

Id	Name
103	Smith
104	Pierre
110	Drew

deg(R) = 2, |r| = 3.

Difference ($-$):

Let P and Q be two union compatible relations. Then

$r = p - q = \{t \mid t \in p \text{ and } t \notin q\}$,
$0 \leq |r| \leq |p|$,
R = P or Q, and
deg(R) = deg(P) or deg(Q).

r:

Id	Name
101	Jones
107	Evan
112	Smith

deg(R) = 2, $|r| = 3$.

Note that ($r' = q - p$) $\neq$ r as

r':

Id	Name
106	Byron

deg(R') = 2, $|r'| = 1$.

Cartesian product ($\times$):

Let Q be the relation shown before and S = {Software} be another relation* as shown.

s:

Software
Word
FoxPro

Then

$r = q \times s = \{t_1 \mid\mid t_2 \mid t_1 \in q \text{ and } t_2 \in s\}$,
$|r| = |q| * |s|$,
$R = Q \mid\mid S$, and
$deg(R) = deg(Q) + deg(S)$.

r:

Id	Name	Software
103	Smith	Word
104	Pierre	Word
106	Byron	Word
110	Drew	Word
103	Smith	FoxPro
104	Pierre	FoxPro
106	Byron	FoxPro
110	Drew	FoxPro

$deg(R) = 3$, $|r| = 8$.

*Note that Cartesian product does not assume union compatibility.

Additional Operators:

Projection (Π):

Let t be a tuple and A be an attribute of t, the value of A being a. Then the projection of t over A is a, denoted t[A]=a.

The project of a relation p over an attribute A is defined as

Π_A (p) = {a | t[A]=a and t $\in$ p}.

Similarly, the projection of p over a set of attributes X is defined as

r = Π_X (p) = {|| a_i | t[A_i]=a_i, $a_i \in$ X, and t $\in$ p},
$0 \leq |r| \leq |p|$,
R = X, and
deg(R) = length(X).

$r=\Pi_{Name}$ (p):

Name
Jones
Smith
Pierre
Evan
Drew

deg(R) = 1, |r| = 5.

Whereas,

$r'=\Pi_{Id}$ (p):

Id
101
103
104
107
110
112

deg(R') = 1, |r'| = 6.

Note: Projection operation reduces the arity of a relation, and possibly reduces the cardinality. Hence projection yields a vertical subset of a relation.

Also, projection may be used to reorder the attributes of a relation.

Selection (σ):

Given a relation p and a predicate expression C, we say that a tuple t satisfies C if the predicate C is true on t.

r = σ_C (p) = $\{$t | t $\in$ p and C(t)=true$\}$,
0 $\leq$ |r| $\leq$ |p|,
R = P, and
deg(R) = deg(P).

r=$\sigma_{Id>106}$ (p):

Id	Name
107	Evan
110	Drew
112	Smith

deg(R) = 2, |r| = 3.

Note: Selection operation yields a horizontal subset of a relation. Its a restriction operation. Usually affects the cardinality but not the degree of a relation.

Joins ($\bowtie$):

Theta Join ($\bowtie_\theta$):

Let student and takes be two relations as shown below. Then

students:

Sid	Name	City
101	Jones	Troy
103	Smith	Troy
104	Pierre	Troy
105	Stan	Westpoint
107	Evan	Moscow
110	Drew	Pullman
112	Smith	Troy

takes:

Cid	Sid	Semester	Pid
4503	101	Fall	34
4503	103	Summer	45
8503	112	Winter	45
4503	112	Summer	45
8503	110	Winter	50
2402	104	Fall	34
3703	103	Winter	55

r = student $\bowtie_\theta$ takes = $\{$ u $\|$ v $|$ u $\in$ students and v $\in$ takes and θ(u,v) = true$\}$ where θ is a valid logical expression over u and v,
$0 \leq |r| \leq |students| * |takes|$,
deg(R) = deg(students) + deg(takes)

Example:

r =
students $\bowtie_{students.Sid > takes.Sid \wedge (Semester = Fall \vee Pid \neq 45)}$ takes =

s.Sid	Name	City	Cid	t.Sid	Sem	Pid
103	Smith	Troy	4503	101	Fall	34
104	Pierre	Troy	4503	101	Fall	34
104	Pierre	Troy	3703	103	Winter	55
105	Stan	Westpoint	4503	101	Fall	34
105	Stan	Westpoint	2402	104	Fall	34
105	Stan	Westpoint	3703	103	Winter	55
107	Evan	Moscow	4503	101	Fall	34
107	Evan	Moscow	2402	104	Fall	34
107	Evan	Moscow	3703	103	Winter	55
110	Drew	Pullman	4503	101	Fall	34
110	Drew	Pullman	2402	104	Fall	34
110	Drew	Pullman	3703	103	Winter	55
112	Smith	Troy	4503	101	Fall	34
112	Smith	Troy	8503	110	Winter	50
112	Smith	Troy	2402	104	Fall	34
112	Smith	Troy	3703	103	Winter	55

Equi-Join ($\bowtie_=$):

θ may only involve equality (=) conditions.

r =
students $\bowtie_{students.Sid=takes.Sid \wedge (Semester=Fall \vee Pid=45)}$ takes =

s.Sid	Name	City	Cid	t.Sid	Sem	Pid
101	Jones	Troy	4503	101	Fall	34
103	Smith	Troy	4503	103	Summer	45
104	Pierre	Troy	2402	104	Fall	34
112	Smith	Troy	8503	112	Winter	45
112	Smith	Troy	4503	112	Summer	45

Natural-Join ($\bowtie$):

θ in natural join may only involve conjunction of equality ($=$) conditions on common attributes followed by a projection to eliminate duplicate columns.

Technically,

r = p $\bowtie$ q = $\Pi_{P \cup Q}$ ($\sigma_{p.A_1=q.A_1 \wedge \ldots \wedge p.A_n=q.A_n}$ p $\times$ q), where P $\cap$ Q = $\{A_1, \ldots, A_n\}$.

r = students $\bowtie$ takes =

s.Sid	Name	City	Cid	Sem	Pid
101	Jones	Troy	4503	Fall	34
103	Smith	Troy	4503	Summer	45
112	Smith	Troy	8503	Winter	45
112	Smith	Troy	4503	Summer	45
110	Drew	Pullman	8503	Winter	50
104	Pierre	Troy	2402	Fall	34
103	Smith	Troy	3703	Winter	55

Division (÷):

Let takes and db_courses be the following relations. Then

takes:

Cid	Sid
4503	101
4503	103
4503	104
8503	112
4503	112
8503	110
2402	101
2402	104
3703	101
3703	103
3703	107
8503	103
3203	112
3203	107

db_courses:

Cid
4503
8503

r = takes ÷ db_courses = { t | t=takes[Takes - Db_courses] and for all tuple t' ∈ db_courses, there exists a tuple t'' ∈ takes such that t'[Db_courses] = t''[Db_courses] and t'[Takes - Db_courses] = t}

$0 \leq |r| \leq |\Pi_{Takes-Db\_courses}(\text{takes})|$,

deg(R) = length(Takes - Db_courses)*

Example:

r = takes ÷ db_courses =

Sid
112
103

*Takes is the scheme corresponding to the relation takes, and so on.

Example:

Now, if we take the following relation as db_courses,

db_courses:

Cid
4503
8503
3203

then

r = takes $\div$ db_courses =

Sid
112

Whereas taking

db_courses:

Cid
4503
8503
3203
2402

we have

r = takes $\div$ db_courses = $\emptyset$

Finally, if we consider

db_courses:

Cid
4503

we have

r = takes $\div$ db_courses =

Sid
101
103
104
112

Relational Query Languages

The Example Database

Works:

Project#	Emp#
Comp460	101
Comp354	103
Comp343	104
Comp354	104
Comp231	106
Comp278	106
Comp360	106
Comp354	106
Comp460	106
Comp231	107
Comp360	107
Comp278	110
Comp360	112
Comp354	112

Employee:

Emp#	Name
101	Jones
103	Smith
104	Pierre
106	Byron
107	Evan
110	Drew
112	Smith

Project:

Project#	PName	Coor.
Comp231	Pascal	107
Comp278	Object	110
Comp360	DBase	107
Comp354	OS	104
Comp460	DBase	101

Data Definition Language

The SQL data-definition language (DDL) allows the specification of information about relations, including:

- The schema for each relation.
- The domain of values associated with each attribute.
- Integrity constraints And as we will see later, also other information such as
  - The set of indices to be maintained for each relations.
  - Security and authorization information for each relation.
  - The physical storage structure of each relation on disk.

Domain Types in SQL

The SQL data-definition language (DDL) allows the specification of information about relations, including:

- **char(n)**. Fixed length character string, with user-specified length n.
- **varchar(n)**. Variable length character strings, with user-specified maximum length n.
- **int**. Integer (a finite subset of the integers that is machine-dependent).
- **smallint**. Small integer (a machine-dependent subset of the integer domain type).
- **numeric(p,d)**. Fixed point number, with user-specified precision of p digits, with d digits to the right of decimal point. (ex., numeric(3,1), allows 44.5 to be stores exactly, but not 444.5 or 0.32)
- **real, double precision**. Floating point and double-precision floating point numbers, with machine-dependent precision.
- **float(n)**. Floating point number, with user-specified precision of at least *n* digits.
- More are covered in Chapter 4.

SQL create table Construct

An SQL relation is defined using the create table command:

create table *r* $(A_1 D_1, A_2 D_2, \ldots, A_n D_n,$
(*integrity-constraint*$_1$)
...
(*integrity-constraint*$_k$));

- r is the name of the relation
- each A_i is an attribute name in the schema of relation r
- D_i is the data type of values in the domain of attribute A_i

create table *instructor* (

ID	**char(5)**,
name	**varchar(20) not null**,
dept_name	**varchar(20)**,
salary	**numeric(8,2)**;

Constraints: SQL create table Construct

- **not null**
- **primary key** $(A_1, \ldots, A_n)$
- **foreign key** $(A_m, \ldots, A_n)$ **references** *r*

create table *instructor* (
ID **char(5)**,
name **varchar(20) not null**,
dept_name **varchar(20)**,
salary **numeric(8,2)**,
primary key (*ID*),
foreign key (*dept_name*) **references** *department*);

primary key declaration on an attribute automatically ensures **not null**.

More Examples

```
create table student (
    ID              char(5),
    name            varchar(20) not null,
    dept_name       varchar(20),
    tot_credit      numeric(3,0),
    primary key (ID),
    foreign key (dept_name) references
                    department);
```

```
create table takes (
    ID              varchar(5),
    course_id       varchar(8),
    sec_id          varchar(8),
    semester        varchar(6),
    year            numeric(4,0),
    grade           varchar(2),
    primary key (ID, course_id, sec_id, semester, year),
    foreign key (ID) references student,
    foreign key (course_id, sec_id, semester, year)
                    references section);
```

Note: *sec_id* can be dropped from **primary key** above, to ensure a *student* cannot be registered for two sections of the same course in the same semester.

More Still

```
create table course (
   course_id      varchar(8),
   title          varchar(50),
   dept_name      varchar(20),
   credits        numeric(2,0),
   primary key (course_id),
   foreign key (dept_name) references
                  department);

create table persons (
   id             varchar(8) not null
                  primary key,
   lastname       varchar(50),
   firstname      varchar(20),
   age            int);

create table persons (
   id             varchar(8) not null
   lastname       varchar(50) not null,
   firstname      varchar(20),
   age            int,
   constraint pk_persons primary key
                  (id, lastname));
```

Query 1:

Find the employee number of employees working on project Comp360.

RA:

$$\Pi_{Emp\#}(\sigma_{Project\#=Comp360}(Works))$$

TRC:

$$\{t[Emp\#] \mid t \in Works \wedge t[Project\#] = Comp360\}$$

SQL:

select Emp#
from Works
where Project#=Comp360

Response:

Emp#
106
107
112

Query 2:

Find the name and employee number of employees working on project Comp360.

RA:

$$Employee \bowtie (\Pi_{Emp\#}(\sigma_{Project\#=Comp360}(Works)))$$

TRC:

$$\{t \mid t \in Employee \wedge \exists u(u \in Works \wedge u[Project\#] = Comp360 \wedge t[Emp\#] = u[Emp\#])\}$$

SQL:

```
select Emp#, Name
from Works, Employee
where Project#=Comp360 and
  Works.Emp#=Employee.Emp#
```

Response:

Emp#	
106	Byron
107	Evan
112	Smith

Query 3:

Find the name and employee number of employees working on any DBase project.

RA:

$$Employee \bowtie (\Pi_{Emp\#}(Works \bowtie (\sigma_{Pname=DBase}(Project))))$$

TRC:

$$\{t \mid t \in Employee \wedge \exists u, v(u \in Works \wedge v \in Project \wedge u[Project\#] = v[Project\#] \wedge t[Emp\#] = u[Emp\#] \wedge v[Pname] = DBase)\}$$

SQL:

select Emp#, Name
from Works, Employee, Project
where Pname=DBase and
  Works.Emp#=Employee.Emp# and
  Works.Project#=Project.Project#

Response:

Emp#	
101	Jones
106	Byron
107	Evan
112	Smith

Query 4:

Find the name and employee number of employees who work on all the projects.

RA:

$Employee \bowtie (Works \div (\Pi_{Project\#}(Project)))$

TRC:

$$\{t \mid t \in Employee \wedge \exists u(u \in Works \wedge t[Emp\#] = u[Emp\#] \wedge \forall p(p \in Project \Rightarrow \exists v(v \in Works \wedge p[Project\#] = v[Project\#] \wedge u[Emp\#] = v[Emp\#])))\}$$

SQL:

```
select *
from Employee E
where (
   (select Project#
   from Works W
   where E.Emp#=W.Emp#)
  contains
  (select Project#
   from Project)
)
```

Response:

Emp#	
106	Byron

Query 5:

Find the name and employee number of employees who do not work on project Comp460.

RA:

$Employee \bowtie ((\Pi_{Emp\#}(Works) -$
$(\Pi_{Emp\#}(\sigma_{Project\#=Comp460}(Works)))$

TRC:

$\{t \mid t \in Employee \wedge \forall u(u \in Works \wedge p[Project\#] = Comp460 \Rightarrow$
$t[Emp\#] \neq u[Emp\#])\}$

SQL:

(**select** *
from Employee)
minus
(**select** Employee.Emp#, Employee.Name
from Employee, Works
where Employee.Emp#=Works.Emp# and
Works.Project#=Comp460)

Response:

Emp#	Name
103	Smith
104	Pierre
107	Evan
110	Drew
112	Smith

Query 6:

Find the name and employee number of employees who are assigned to projects as coordinators only.

SQL:

```
select *
from Employee e
where (
   (select p.Project#
   from Project p
   where e.Emp#=p.Coor)
   contains
   (select p.Project#
   from Works w
   where e.Emp#=w.Emp#)
)
```

Response:

Emp#	Name
101	Jones
107	Evan
110	Drew

Advanced Examples: Consider the following relational database called *university*.

students:

Sid	Name	Age	City
101	Jones	25	Troy
103	Smith	20	Troy
104	Pierre	28	Troy
105	Stan	30	Westpoint
107	Evan	22	Moscow
110	Drew	18	Pullman
112	Smith	25	Troy

courses:

Cid	Title	Credits	Group
4503	Databases	3	DB
2402	C	2	PL
8503	Adv Databases	3	DB
3703	OS	3	Sys
3203	Data Structures	3	Foun

takes:

Cid	Sid	Semester	Year	Pid
4503	101	Fall	97	34
4503	103	Summer	98	45
8503	112	Winter	98	45
4503	112	Summer	98	45
8503	110	Winter	97	50
2402	101	Fall	96	34
2402	104	Fall	97	34
3703	101	Winter	97	55
3703	103	Winter	97	55
3703	107	Fall	97	55
3203	112	Fall	98	45

professors:

Pid	Name	Office	Dept
34	Smith	BU102	CS
45	Turing	BU311	CS
50	Sue	ER201	ERC
55	Probst	BU333	CS
72	Alagar	BU222	CS

Query 1: List all the students who live in Troy.

RA: $\sigma_{City="Troy"}(students)$

TRC: $\{t | t \in students \wedge t[city] = "Troy\}$

DRC: $\{< s, n, a, c > \mid < s, n, a, c > \in students \ \wedge$
$c = "Troy\}$

or

$\{< s, n, a > \mid < s, n, a, "Troy > \in students\}$

SQL: select *
from students
where City="Troy"

Query 2: List names of all the student who live in Troy.

RA: $\Pi_{Name}(\sigma_{City=''Troy''}(students))$

TRC: $\{t|\exists u(u \in students \ \wedge \ t = u[name] \ \wedge$
$u[city] = ''Troy'')\}$

DRC: $\{< n > |\exists s, a(< s, n, a, ''Troy'' >\in students)\}$

SQL: select Name
from students
where City=" Troy"

Query 3: List names of all students who took a course in Summer of 1998.

RA: $\Pi_{Name}((\sigma_{Year=98 \wedge Semester="Summer"}(takes)) \bowtie students)$

TRC: $\{t | \exists u, v(u \in students \ \wedge \ v \in takes \ \wedge$
$t = u[name] \ \wedge \ u[sid] = v[sid] \ \wedge$
$v[semester] = "Summer" \ \wedge \ v[year] = 98)\}$

DRC: $\{< n > | \exists s, a, c(< s, n, a, c > \in students \ \wedge$
$\exists ci, p(< ci, s, "Summer", 98, p > \in takes))\}$

SQL: select Name
from students, takes
where takes.Sid = students.Sid and Semester = "Summer" and Year = 98

<u>Query 4:</u> List names of all students who took a 2 credit course.

RA: $\Pi_{Name}((\sigma_{Credits=2}(courses)) \bowtie takes \bowtie students)$

TRC: $\{t | \exists u, v, w(u \in students \ \wedge \ v \in takes \ \wedge$
$w \in courses \ \wedge \ t = u[name] \ \wedge \ u[sid] =$
$v[sid] \ \wedge$
$v[cid] = w[cid] \ \wedge \ w[credits] = 2)\}$

DRC: $\{< n > | \exists s, a, c(< s, n, a, c > \in students \ \wedge$
$\exists ci, se, y, p(< ci, s, se, y, p > \in takes \ \wedge$
$\exists ti, g(< ci, ti, 2, g > \in courses)))\}$

SQL: select Name
from students, takes, courses
where takes.Cid = students.Cid and takes.Cid
= courses.Cid and Credits = 2

Query 5: List names of all students who took a course with Professor Turing.

RA: $\Pi_{Name}((\sigma_{Name=''Turing''}(professors)) \bowtie takes \bowtie students)$

TRC: $\{t | \exists u, v, w(u \in students \ \wedge \ v \in takes \ \wedge \ w \in professors \ \wedge \ t = u[name] \ \wedge \ u[sid] = v[sid] \ \wedge \ v[pid] = w[pid] \ \wedge \ w[name] = ''Turing'')\}$

DRC: $\{< n > | \exists s, a, c(< s, n, a, c > \in students \ \wedge \ \exists ci, se, y, p(< ci, s, se, y, p > \in takes \ \wedge \ \exists o, d(< p, ''Turing'', o, d > \in professors)))\}$

SQL: select students.Name
from students, takes, professors
where takes.Sid = students.Sid and takes.Pid = professors.Pid and professors.Name = "Turing"

Query 6: List names of all students who took a database course or a systems course.

RA: $\Pi_{Name}((\sigma_{Group=''DB'' \wedge Group=''Sys''}(courses)) \bowtie takes \bowtie students)$

TRC: $\{t|\exists u, v, w(u \in students \ \wedge \ v \in takes \ \wedge$
$w \in courses \ \wedge \ t = u[name] \ \wedge$
$u[sid] = v[sid] \ \wedge \ v[cid] = w[cid] \ \wedge$
$(w[group] = ''DB'' \vee w[group] = ''Sys''))\}$

DRC: $\{< n > |\exists s, a, c(< s, n, a, c > \in students \ \wedge$
$\exists ci, se, y, p(< ci, s, se, y, p > \in takes \ \wedge$
$\exists ti, cr, g(< ci, ti, cr, g > \in courses \ \wedge$
$(g = ''Sys'' \vee g = ''DB'')))))\}$

SQL: (select students.Name
from students, takes, courses
where takes.Sid = students.Sid and takes.cid
= courses.cid and courses.Group = "DB")
union
(select students.Name
from students, takes, courses
where takes.Sid = students.Sid and takes.cid
= courses.cid and courses.Group = "Sys")

Query 7: List names of all students who took a database course or a systems course with Professor Turing.

RA: $\Pi_{Name}(((\sigma_{Group="DB" \vee Group="Sys"}(courses)) \bowtie takes \bowtie (\sigma_{Name="Turing"}(Professors))) \bowtie students)$

TRC: $\{t | \exists u, v, w, x(u \in students \wedge v \in takes \wedge w \in courses \wedge x \in professors \wedge t = u[name] \wedge u[sid] = v[sid] \wedge v[cid] = w[cid] \wedge v[pid] = x[pid] \wedge x[name] = "Turing" \wedge (w[group] = "DB" \vee w[group] = "Sys"))\}$

DRC: $\{< n > | \exists s, a, c(< s, n, a, c > \in students \wedge \exists ci, se, y, p(< ci, s, se, y, p > \in takes \wedge \exists ti, cr, g(< ci, ti, cr, g > \in courses \wedge \exists o, d(< p, "Turing", o, d > \in professors \wedge (g = "Sys" \vee g = "DB")))))\}$

SQL:
```
(select students.Name
from students, takes, courses, professors
where takes.Sid = students.Sid and takes.cid = courses.cid and courses.Group = "DB" and professors.Pid = takes.Pid and professors.Name = "Turing")
union
(select students.Name
from students, takes, courses
where takes.Sid = students.Sid and takes.cid = courses.cid and courses.Group = "Sys" and professors.Pid = takes.Pid and professors.Name = "Turing")
```

Query 8: List names of students who never took a course.

RA: $\Pi_{Name}(students) - \Pi_{Name}(students \bowtie takes)$

TRC: $\{t | \exists u(u \in students \ \wedge \ t = u[name] \ \wedge$
$\forall v(v \in takes \Rightarrow v[sid] \neq u[sid])\}$

DRC: $\{< n > | \exists s, a, c(< s, n, a, c > \in students \ \wedge$
$\forall ci, si, se, y, p(< ci, si, se, y, p > \in takes \Rightarrow$
$si \neq s))\}$

SQL:
```
select Name
from students
where Sid not in
          (select Sid
          from takes)
```

Query 9: List names of professors who teach only in the Summer.

RA: $\Pi_{Name}(professors \bowtie takes) - \Pi_{Name}((professors \bowtie (\sigma_{Semester \neq "Summer"}(takes))))$

TRC: $\{t | \exists u(u \in professors \ \wedge \ t = u[name] \ \wedge$
$\exists w(w \in takes \ \wedge \ w[pid] = u[pid] \ \wedge$
$\forall v(v \in takes \ \wedge \ v[pid] = u[pid] \Rightarrow$
$v[semester] = "Summer")))\}$

DRC: $\{< n > | \exists p, o, d(< p, n, o, d > \in professors \ \wedge$
$\exists c, s, se, y(< c, s, se, y, p > \in takes \ \wedge$
$\forall ci, si, sem, ye(< ci, si, sem, ye, p > \in takes \Rightarrow$
$sem = "Summer")))\}$

SQL:
```
select Name
from professors
where Pid not in
          (select Pid
          from takes
          where Semester ≠ "Summer"
          group by pid
          having count(*) > 0)
and Pid in
          (select Pid
          from takes)
```

Query 10: List names of students who took all the database courses.

RA: $\Pi_{Name}(students \bowtie ((\Pi_{Cid,Sid}(takes)) \div (\Pi_{Cid}(\sigma_{Group="DB"}(courses)))))$

TRC: $\{t | \exists u(u \in students \;\wedge\; t = u[name] \;\wedge\; \forall v(v \in courses \;\wedge\; v[group] = "DB" \Rightarrow \exists x(x \in takes \;\wedge\; x[sid] = u[sid] \;\wedge\; x[cid] = v[cid])))\}$

DRC: $\{< n > | \exists s, a, c(< s, n, a, c > \in students \;\wedge\; \forall ci, t, cr(< ci, t, cr, "DB" > \in courses \Rightarrow \exists sem, y, p(< ci, s, sem, y, p > \in takes)))\}$

SQL:
```
select Name
from students as S
where
        (select Cid
        from takes
        where takes.Sid=S.Sid)
contains
        (select Cid
        from courses
        where Group = "DB")
```

Query 11: List names of students who took all the courses taught by Professor Turing.

RA: $\Pi_{Name}(students \bowtie ((\Pi_{Cid,Sid}(takes)) \div (\Pi_{Cid}(((\Pi_{Pid}(\sigma_{Name=''Turing''}(professors)) \bowtie takes))))))$

TRC: $\{t | \exists u(u \in students \wedge t = u[name] \wedge \exists w(w \in professors \wedge w[name] = ''Turing'' \wedge \forall v(v \in takes \wedge v[pid] = w[pid] \Rightarrow \exists x(x \in takes \wedge x[sid] = u[sid] \wedge x[cid] = v[cid])))))\}$

DRC: $\{< n > | \exists s, a, c(< s, n, a, c > \in students \wedge \exists p, o, d(< p, ''turing'', o, d > \in professors \wedge \forall ci, si, se, y(< ci, si, se, y, p > \in takes \Rightarrow \exists sem, ye, pi(< ci, s, sem, ye, pi > \in takes)))\}$

SQL:
```
select Name
from students as S
where
        (select Cid
        from takes
        where takes.Sid=S.Sid)
        contains
        (select Cid
        from takes, professors
        where takes.Pid = professors.Pid and
        Name = "Turing")
```

Safety of Calculus Expressions

Consider the expression below that generates infinitely many tuples.

$\{t \mid \neg(t \in students)\}$

Domain of a formula: Domain of a formula F, denoted $dom(F)$, is the set of all values* referenced by F.

A calculus expression e is <u>*safe*</u> if all values that appear in the result are values from $dom(F)$ such that $e = \{t \mid F(t)\}$.

*The values mentioned in F or the values that appear in the tuples in all the relations mentioned in F.

Expressive power of languages

The expressive power of safe tuple calculus, and safe domain calculus are equvalent in expressive power to the (basic) relational algebra.

Safe TRC $\equiv$ Safe DRC $\equiv$ Basic RA

Extended Relational Algebra Operators

There are four extended operators:

- Rename: $\rho_s(r)$ and $\rho_{s(A_1,\ldots,A_k)}(r)$.
  - $\rho_{FallCourses}(\sigma_{semester="Fall"}(Takes))$.
  - $\rho_{FallCourses(CID,Sem,Year)}(\sigma_{sem="Fall"}(Takes))$ where Takes(CourseID, Semester, Year) is the scheme.
- Assignment: $s \leftarrow r$.
  - $FallCourses \leftarrow \sigma_{semester="Fall"}(Takes)$ is similar to $\rho_{FallCourses}(\sigma_{semester="Fall"}(Takes))$ except that FallCourses is now stored, scheme remains as in Takes.
  - $FCourses \leftarrow$ $\rho_{FallCourses(CID,Sem,Year)}(\sigma_{semester="Fall"}(Takes))$ also renames the attributes of Takes.

- Group By: ${}_{L}\mathcal{G}_{f_1,f_2,\ldots,f_n}(r)$ or ${}_{L}\mathcal{G}_{f_1\ as\ A_1,f_2\ as\ A_2,\ldots,f_n\ as\ A_n}(r)$.
  - ${}_{Year}\mathcal{G}_{Count(Year)\ as\ Total}(Takes)$

- Outer Joins:
  - Left Outer Join: $r ⟕ s$
    * Students ⟕ Takes will retain all students who never took a class too, padded with null.
  - Right Outer Join: $r ⟖ s$
    * Students ⟖ Takes will retain all courses that were never taken by a student too, padded with null.
  - Full Outer Join: $r ⟗ s$
    * Students ⟗ Takes will retain all courses that was never taken by a student and the courses that were never taken by a student, padded both sides with null.

Joins in SQL

select name, SID
from students natural join takes
where semester="Fall"

select name, SID
from students inner join on students.sid=takes.sid
takes
where semester="Fall"

select name, SID
from students join on students.sid=takes.sid
takes
where semester="Fall"

Joins in SQL: Continued

select name, SID
from students left outer join takes
where semester="Fall"

select name, SID
from students left outer join on
students.sid=takes.sid takes
where semester="Fall"

select name, SID
from students right outer join on
students.sid=takes.sid takes
where semester="Fall"

select name, SID
from students full outer join on
students.sid=takes.sid takes
where semester="Fall"