Relational Algebra

Hugh Darwen

hughdarwen@gmail.com
web.onetel.com/~hughdarwen/M359/

A lecture derived from HD's course at Warwick University.
Terms and concepts not used in M359 are shown in this colour.

Anatomy of a Relation

StudentId	Name	CourseId
S1	Anne	C1

attribute name

attribute values

n-*tuple*, or *tuple*.
This is a 3-tuple.
The tuples constitute the *body* of the relation.
The number of tuples in the body is the *cardinality* of the relation.

Heading (a set of attributes)
The *degree* of this heading is 3, which is also the degree of the relation.

ENROLMENT Example

ENROLMENT (a relation variable, or *relvar*)

StudentId	Name	CourseId
S1	Anne	C1
S1	Anne	C2
S2	Boris	C1
S3	Cindy	C3
S4	Devinder	C1

Predicate: StudentId is called Name and is enrolled on CourseId

Note *redundancy*: S1 is *always* called Anne!

Splitting ENROLMENT

IS_CALLED

StudentId	Name
S1	Anne
S2	Boris
S3	Cindy
S4	Devinder
S5	Boris

Student StudentId is called Name

IS_ENROLLED_ON

StudentId	CourseId
S1	C1
S1	C2
S2	C1
S3	C3
S4	C1

Student StudentId is enrolled on course CourseId

Relations and Predicates (1)

Consider the predicate: StudentId is called Name

… is called --- is the *intension* (meaning) of the predicate.

The parameter names are arbitrary. "S is called N" means the same thing (has the same intension).

The *extension* of the predicate is the set of *true* propositions that are *instantiations* of it:
{ S1 is called Anne, S2 is called Boris, S3 is called Cindy, S4 is called Devinder, S5 is called Boris }

Each tuple in the body (extension) of the relation provides the values to substitute for the parameters in one such instantiation.

Relations and Predicates (2)

Moreover, each proposition in the extension has exactly one corresponding tuple in the relation.

This 1:1 correspondence reflects the *Closed-World Assumption*:

> A tuple representing a true instantiation is in the relation.
> A tuple representing a false one is out.

The Closed-World Assumption underpins the operators we are about to meet.

Relational Algebra

Operators that operate on relations and return relations.

In other words, operators that are *closed over* relations. Just as arithmetic operators are closed over numbers.

Closure means that every invocation can be an operand, allowing expressions of arbitrary complexity to be written. Just as, in arithmetic, e.g., the invocation b-c is an operand of a+(b-c).

The operators of the relational algebra are relational counterparts of *logical* operators: AND, OR, NOT, EXISTS. Each, when invoked, yields a relation, which can be interpreted as the extension of some predicate.

Logical Operators

Because relations are used to represent predicates, it makes sense for relational operators to be counterparts of operators on predicates. We will meet examples such as these:

Student StudentId is called Name **AND** StudentId is enrolled on course CourseId.

Student StudentId is enrolled **on some course**.

Student StudentId is enrolled on course CourseId AND StudentId is **NOT** called Devinder.

Student StudentId is NOT enrolled on any course **OR** StudentId is called Boris.

Meet The Operators

Logic	Relational counterpart
AND	**join** restriction (**select …where**) extension SUMMARIZE *and some more*
EXISTS	**project … over**
OR	**union**
(AND) NOT	**(semi)difference**
	rename

join (= AND)

StudentId is called Name **AND** StudentId is enrolled on CourseId.

IS_CALLED **join** IS_ENROLLED_ON

IS_CALLED

Name	StudentId
Anne	S1
Boris	S2
Cindy	S3
Devinder	S4
Boris	S5

IS_ENROLLED_ON

StudentId	CourseId
S1	C1
S1	C2
S2	C1
S3	C3
S4	C1

IS_CALLED **join** IS_ENROLLED_ON

StudentId	Name	CourseId
S1	Anne	C1
S1	Anne	C2
S2	Boris	C1
S3	Cindy	C3
S4	Devinder	C1

Seen this before? Yes, this is our original ENROLMENT. The JOIN has reversed the split. (And has “lost” the second Boris.)

Definition of **join**

Let *s* = *r1* **join** *r2*. Then:

The heading *Hs* of *s* is the union of the headings of *r1* and *r2*.

The body of *s* consists of those tuples having heading *Hs* that can be formed by taking the union of *t1* and *t2,* where *t1* is a tuple of *r1* and *t2* is a tuple of *r2*.

If *c* is a common attribute, then it must have the same domain in both *r1* and *r2*. (I.e., if it doesn't, then *r1* **join** *r2* is undefined.)

Note: **join**, like AND, is both commutative and associative.

rename

Sid1 is called Name

IS_CALLED **rename** (StudentId as Sid1)

StudentId	Name
S1	Anne
S2	Boris
S3	Cindy
S4	Devinder
S5	Boris

Sid1	Name
S1	Anne
S2	Boris
S3	Cindy
S4	Devinder
S5	Boris

Definition of **rename**

Let *s* = *r* **rename** (*A1* **as** *B1*, … *An* **as** *Bn*)

The heading of *s* is the heading of *r* except that attribute *A1* is renamed to *B1* and so on.

The body of *s* consists of the tuples of *r* except that in each tuple attribute *A1* is renamed to *B1* and so on.

rename and **join**

Sid1 is called Name AND so is Sid2

IS_CALLED **rename** (StudentId **as** Sid1) **join**
IS_CALLED **rename** (StudentId **as** Sid2)

Sid1	Name	Sid2
S1	Anne	S1
S2	Boris	S2
S2	Boris	S5
S5	Boris	S2
S3	Cindy	S3
S4	Devinder	S4
S5	Boris	S5

Special Cases of **join**

What is the result of R JOIN R?

R

What if all attributes are common to both operands?

It is called “intersection”.

What if no attributes are common to both operands?

It is called “Cartesian product”

Interesting Properties of **join**

It is *commutative*: *r1* **join** *r2* ≡ *r2* **join** *r1*

It is *associative*: (*r1* **join** *r2*) **join** *r3* ≡ *r1* **join** (*r2* **join** *r3*)
So **Tutorial D** allows **join**{*r1*, *r2*, …} (note the braces)

We note in passing that these properties are important for *optimisation* (in particular, of query evaluation).

Of course it is no coincidence that logical AND is also both commutative and associative.

Projection (= EXISTS)

Student StudentId is enrolled **on some course**.

project IS_ENROLLED_ON **over** StudentId

Given:

StudentId	CourseId
S1	C1
S1	C2
S2	C1
S3	C3
S4	C1

To obtain:

StudentId
S1
S2
S3
S4

Definition of Projection

Let *s* = **project** *r* **over** *A1, ..., An* }
(=, in **Tutorial D**, *r* { ALL BUT *B1, ..., Bm* }, where
B1, ..., Bm are the attributes not mentioned in *A1, ..., An*)

The heading of *s* is the subset of the heading of *r* given by { *A1, ... An* }.

The body of *s* consists of each tuple that can be formed from a tuple of *r* by removing from it the attributes named *B1, ... Bm.*

Note that the cardinality of *s* can be less than that of *r* but cannot be more than that of *r.*

How ENROLMENT Was Split

relation IS_CALLED
StudentId: StudentIds,
Name: Names })
primary key *StudentId*
IS_CALLED **:=** **project** ENROLMENT **over** *StudentId, Name*

relation IS_ENROLLED_ON
StudentId: StudentIds,
CourseId: CourseIds
primary key *StudentId, CourseId*
IS_ENROLLED_ON **:=** **project** ENROLMENT
over *StudentId, CourseId*

Special Case of AND (1)

StudentId is called Boris

Can be done using JOIN and projection, like this:

project (IS_CALLED JOIN
RELATION { TUPLE { Name 'Boris' } })
over *StudentId*

but it's easier using *restriction* (and projection again):

project (**select** IS_CALLED **where** Name = 'Boris')
over StudentId

result:

StudentId
S2
S5

"EXISTS Name such that StudentId is called Name AND Name is Boris"

A More Useful Restriction

Sid1 has the same name as Sid2 (**AND** Sid2 ≠ Sid1).

project (**select** ((IS_CALLED **rename** (*StudentId* **as** *Sid1*))
join
(IS_CALLED **rename** (*StudentId* **as** *Sid2*)))
where *Sid1* **<** *Sid2*) **over** *Sid1*, *Sid2*

Result:

Sid1	Sid2
S2	S5

Hopelessly difficult using JOIN instead of WHERE! (Why?)

Definition of Restriction

Let *s* = **select** *r* **where** *c,* where *c* is a conditional expression on attributes of *r*.

The heading of *s* is the heading of *r*.

The body of *s* consists of those tuples of *r* for which the condition *c* evaluates to TRUE.

So the body of *s* is a subset of that of *r*.

Special Cases of Restriction

What is the result of R **where** TRUE?

R

What is the result of R **where** FALSE?

The empty relation with the heading of R.

Extension

StudentId is called Name **AND** Name begins with the letter Initial.

EXTEND IS_CALLED **ADD**
(SUBSTRING (Name, 0, 1) AS Initial)

Result:

StudentId	Name	Initial
S1	Anne	A
S2	Boris	B
S3	Cindy	C
S4	Devinder	D
S5	Boris	B

Definition of Extension

Let *s* = **EXTEND** *r* **ADD** (*formula-1* AS *A1*, … *formula-n* AS *An*)

The heading of *s* consists of the attributes of the heading of *r* plus the attributes *A1* … *An*. The declared type of attribute *Ak* is that of *formula-k.*

The body of *s* consists of tuples formed from each tuple of *r* by **add**ing *n* additional attributes *A1* to *An*. The value of attribute *Ak* is the result of evaluating *formula-k* on the corresponding tuple of *r*.

OR

StudentId is called Name **OR** StudentId is enrolled on CourseId.

StudentId	Name	CourseId
S1	Anne	C1
S1	Boris	C1
S1	Zorba	C1
S1	Anne	C4
S1	Anne	C943

and so on *ad infinitum* (almost!)

NOT SUPPORTED!

UNION (restricted OR)

StudentId is called Devinder **OR** StudentId is enrolled on C1.

StudentId
S1
S2
S4

(**project** (**select** IS_CALLED **where** Name = ‘Devinder’)
over StudentId)
union
(**project** (**select** IS_ENROLLED_ON **where** CourseId = ‘C1’)
over StudentId)

Definition of UNION

Let *s* = *r1* **union** *r2*. Then:

r1 and *r2* must have the same heading.

The heading of *s* is the common heading of *r1* and *r2*.

The body of *s* consists of each tuple that is *either* a tuple of *r1* *or* a tuple of *r2*.

Is UNION commutative? Is it associative?

NOT

StudentId is **NOT** called Name

StudentId	Name
S1	Boris
S1	Quentin
S1	Zorba
S1	Cindy
S1	Hugh

and so on *ad infinitum* (almost!)

NOT SUPPORTED!

Restricted NOT

StudentId is called Name **AND** is **NOT** enrolled on any course.

StudentId	Name
S5	Boris

In **Tutorial D** (but not in M359!)
IS_CALLED **NOT MATCHING** IS_ENROLLED_ON

Definition of NOT MATCHING

Let *s* = *r1* **NOT MATCHING** *r2*. Then:

The heading of *s* is the heading of *r1*.

The body of *s* consists of each tuple of *r1* that matches no tuple of *r2* on their common attributes.

It follows that in the case where there are no common attributes, *s* is equal to *r1* if *r2* is empty, and otherwise is empty .

DIFFERENCE

M359 teaches *r1* **difference** *r2* instead of *r1* NOT MATCHING *r2*.

difference is set difference and requires *r1* and *r2* to have the same heading (as in *r1* **union** *r2*).

Most textbooks (following Codd) teach **difference** and do not even define NOT MATCHING, in spite of its greater generality.

Either can be defined in terms of the other.

DIFFERENCE Example

As before:
StudentId is called Name **AND** is **NOT** enrolled on any course.

StudentId	Name
S5	Boris

IS_CALLED **join** (
(**project** IS_CALLED **over** StudentId)
difference
(**project** IS_ENROLLED_ON **over** StudentId))