Relational Algebra-Part I

Hi! I would like to discuss with you about the relational algebra, the basis of SQL language.

A brief introduction

- Relational algebra and relational calculus are formal languages associated with the relational model.
- Informally, relational algebra is a (high-level) procedural language and relational calculus a non-procedural language.
- However, formally both are equivalent to one another.
- A language that produces a relation that can be derived using relational calculus is relationally complete.
- Relational algebra operations work on one or more relations to define another relation without changing the original relations.
- Both operands and results are relations, so output from one operation can become input to another operation.
- Allows expressions to be nested, just as in arithmetic. This property is called closure.

What? Why?

- Similar to normal algebra (as in 2+3*x-y), except we use relations as values instead of numbers.
- Not used as a query language in actual DBMSs. (SQL instead.)
- The inner, lower-level operations of a relational DBMS are, or are similar to, relational algebra operations. We need to know about relational algebra to understand query execution and optimization in a relational DBMS.
- Some advanced SQL queries requires explicit relational algebra operations, most commonly *outer join*.

- SQL is *declarative*, which means that you tell the DBMS *what* you want, but not *how* it is to be calculated. A C++ or Java program is *procedural*, which means that you have to state, step by step, exactly how the result should be calculated. Relational algebra is (more) procedural than SQL. (Actually, relational algebra is mathematical expressions.)
- It provides a formal foundation for operations on relations.
- It is used as a basis for implementing and optimizing queries in DBMS software.
- DBMS programs add more operations which cannot be expressed in the relational algebra.
- Relational calculus (tuple and domain calculus systems) also provides a foundation, but is more difficult to use. We'll skip these for now.

The Basic Operations in Relational Algebra

- Basic Operations:
  - Selection (σ): choose a subset of rows.
  - Projection (π): choose a subset of columns.
  - Cross Product ($\times$): Combine two tables.
  - Union ($\cup$): unique tuples from either table.
  - Set difference ($-$): tuples in R1 not in R2.
  - Renaming (ρ): change names of tables & columns
- Additional Operations (for convenience):
  - Intersection, joins (*very useful*), division, outer joins, aggregate functions, etc.

Now we will see the various operations in relational algebra in detail. So get ready to be in a whole new world of algebra which posses the power to extract data from the database.

Selection Operation σ

The **select** command gives a programmer the ability to choose tuples from a relation (rows from a table). Please do not confuse the Relational Algebra **select** command with the more powerful SQL **select** command that we will discuss later.

Idea: choose tuples of a relation (rows of a table)

Format: **σselection-condition**(R). Choose tuples that satisfy the ***selection condition***. Result has identical schema as the input.

σMajor = 'CS' (Students)

This means that, the desired output is to display the name of students who has taken CS as Major. The Selection condition is a Boolean expression including **=, ≠,** <, ≤, **>,** ≥, and, or, not.

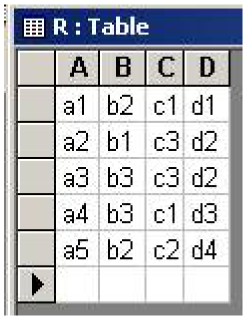
Students

SID	Name	GPA	Major
456	John	3.4	CS
457	Carl	3.2	CS
678	Ken	3.5	Math

Result

SID	Name	GPA	Major
456	John	3.4	CS
457	Carl	3.2	CS

Once again, all the Relational Algebra select command does choose tuples from a relation. For example, consider the following relation R (A, B, C, D):


R : Table

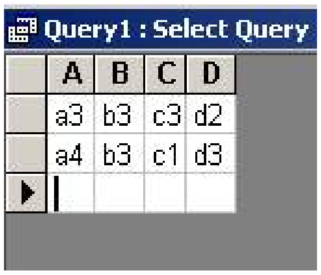
A	B	C	D
a1	b2	c1	d1
a2	b1	c3	d2
a3	b3	c3	d2
a4	b3	c1	d3
a5	b2	c2	d4

I call this an "abstract" table because there is no way to determine the real world model that the table represents. All we know is that attribute (column) A is the primary key and that fact is reflected in the fact that no two items currently in the A column of R are the same. Now using a popular variant of Relation Algebra notation…if we were to do the Relational Algebra command

Select R where B > 'b2' Giving R1;

or another variant
`R1 = R where B > 'b2';`

We would create a **relation R1** with the ***exact*** same attributes and attribute domains (column headers and column domains) as **R**, but we would select only the tuples where the B attribute value is greater than ‘b2’. This table would be



A	B	C	D
a3	b3	c3	d2
a4	b3	c1	d3

(I wrote the query in Microsoft Access XP). Important things to know about using the Relational Algebra **select** command is that the Relation produced always has the exact same attribute names and attribute domains as the original table – we just delete out certain columns.

Let us Consider the following relations

SP

S#	P#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S2	P1	300
S2	P2	400
S3	P2	200

S

S#	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris

P

P#	PNAME	COLOUR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London

Fig.1

Now based on this consider the following examples

e.g. SELECT S WHERE CITY = 'PARIS'

S#	SNAME	STATUS	CITY
S2	Jones	10	Paris
S3	Blake	30	Paris

e.g. SELECT SP WHERE (S# = S1 and P# = P1)

S#	P#	QTY
S1	P1	300

The resulting relation has the same attributes as the original relation.

The selection condition is applied to each tuple in turn - it cannot therefore involve more than one tuple.

PROJECT Operation

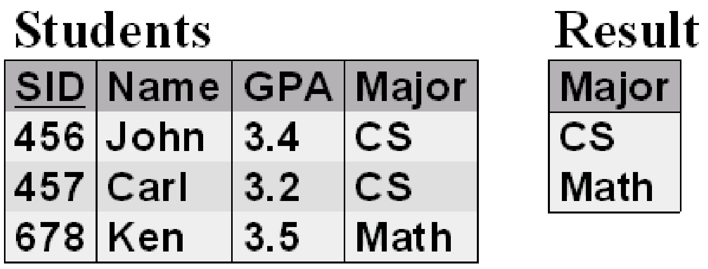
The Relational Algebra **project** command allows the programmer to choose attributes (columns) of a given relation and delete information in the other attributes

Idea: Choose certain attributes of a relation (columns of a table)

Format: $\square_{\textbf{Attribute\_List}}$ (Relation)

Returns: a relation with the same tuples as (Relation) but limited to those attributes of interest (in the attribute list).selects some of the *columns* of a table; it constructs a vertical subset of a relation; implicitly removes any duplicate tuples (so that the result will be a relation).

πMajor(Students)



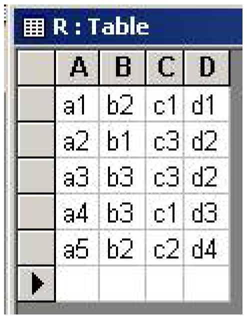
Students

SID	Name	GPA	Major
456	John	3.4	CS
457	Carl	3.2	CS
678	Ken	3.5	Math

Result

Major
CS
Math

For example, given the original abstract relation R(A, B, C, D):



R : Table

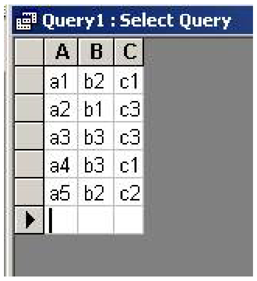
A	B	C	D
a1	b2	c1	d1
a2	b1	c3	d2
a3	b3	c3	d2
a4	b3	c1	d3
a5	b2	c2	d4

We can pick out columns A, B, and C with the following:

Project R over [A, B, C] giving R2;

R2 = R[A, B, C];

This would give us a relation R2(A, B, C) with the D attribute gone:



Query1 : Select Query

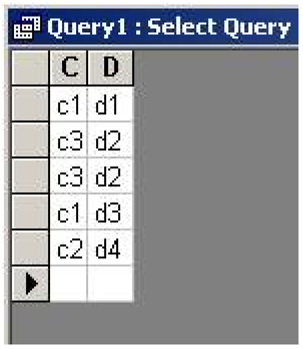
A	B	C
a1	b2	c1
a2	b1	c3
a3	b3	c3
a4	b3	c1
a5	b2	c2

There is one slight problem with this command. Sometimes the result might contain a duplicate tuple. For example, what about

project R over [C, D] giving R3;

R3 = R[C, D];

What would R3(C, D) look like?



Query1 : Select Query

C	D
c1	d1
c3	d2
c3	d2
c1	d3
c2	d4

There are two (c3, d2) tuples. This is not allowed in a "legal" relation. What is to be done? Of course, in Relational Algebra, all duplicates are deleted. Now consider the following examples based on the following examples based on the Fig.1

e.g. PROJECT S OVER CITY

CITY
London
Paris

e.g. PROJECT S OVER SNAME, STATUS

SNAME	STATUS
Smith	20
Jones	10
Blake	30

Sequences of operations

Now we can see the sequence of operations based on both selection and Projection operations.

E.g. part names where weight is less than 17:

TEMP <- SELECT P WHERE WEIGHT < 17

RESULT <- PROJECT TEMP OVER PNAME

TEMP

P#	PNAME	COLOUR	WEIGHT	CITY
P1	Nut	Red	12	London
P4	Screw	Red	14	London

RESULT

PNAME
Nut
Screw

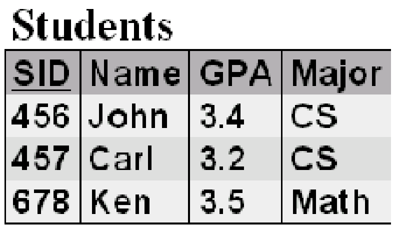
or (nested operations)

PROJECT (SELECT P WHERE WEIGHT < 17) OVER PNAME

Renaming Operation

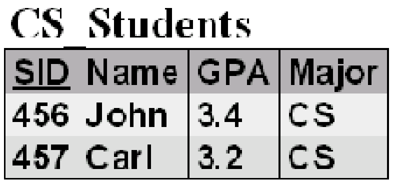
Format: $\rho S(R)$ or $\rho S(A1, A2, \ldots)(R)$: change the name of relation R, and names of attributes of R

$\rho CS\_Students(\sigma Major = \text{‘CS’} \quad Students))$



Students

SID	Name	GPA	Major
456	John	3.4	CS
457	Carl	3.2	CS
678	Ken	3.5	Math



CS_Students

SID	Name	GPA	Major
456	John	3.4	CS
457	Carl	3.2	CS

Consider the scenario

- $p_{List1}(s_{cond1}\ (R1))$
- $p_{List2}(s_{cond1}\ (R1))$
- $s_{cond2}\ (p_{List1}(s_{cond1}(R1)))$

It would be useful to have a notation that allows us to “save” the output of an operation for future use.

- $Tmp1 \leftarrow s_{cond1}\ (R1)$
- $Tmp2 \leftarrow p_{List1}(Tmp1)$
- $Tmp3 \leftarrow p_{List2}(Tmp2)$
- $Tmp4 \leftarrow s_{cond2}(Tmp2)$

(Of course, we would come up with better names than Tmp1, Tmp2….)

The resulting temporary relations will have the same attribute names as the originals. We might also want to change the attribute names:

- To avoid confusion between relations.
- To make them agree with names in another table. (You’ll need this soon.)

For this we will define a Rename operator (rho):

$$\mathbf{rS(B_1,B_2,\ldots, B_n)}\ (\ R(A_1,A_2,\ldots,A_n))$$

where S is the new relation name, and $B_1 \ldots B_n$ are the new attribute names. Note that the degree(S) = degree(R) = n.

Examples:

$\mathbf{r}_{\mathbf{EmpNames(LAST,FIRST)}}$ ($\mathrm{p}_{\mathbf{LNAME,FNAME}}$(Employee))

$\mathbf{r}_{\mathbf{Works\_On2(SSN,PNUMBER,HOURS)}}$ (Works_On(ESSN,PNO,HOURS))

<u>The Cartesian Product</u>

The *cartesian product* of two tables combines each row in one table with each row in the other table.

The **Cartesian product** of *n* domains, written $dom(A_1) \times dom(A_2) \times \ldots dom(A_n)$, is defined as follows.

$$(A_1 \times A_2 \times \ldots A_n = \{(a_1, a_2, \ldots, a_n) \mid a_1 \in A_1 \textbf{ AND } a_2 \in A_2 \textbf{ AND } \ldots \textbf{ AND } a_n \in A_n\}$$

We will call each element in the Cartesian product a **tuple**. So each $(a_1, a_2, \ldots, a_n)$ is known as a tuple. Note that in this formulation, the order of values in a tuple matters (we could give an alternative, but more complicated definition, but we shall see later how to swap attributes around in a relation).

Example: The table **E** (for **EMPLOYEE**)

enr	**ename**	**dept**
1	Bill	A
2	Sarah	C
3	John	A

Example: The table **D** (for **DEPARTMENT**)

dnr	**dname**
A	Marketing
B	Sales

C	Legal

SQL	Result					Relational algebra
	enr	**ename**	**dept**	**dnr**	**dname**	
select * from E, D	1	Bill	A	A	Marketing	E **X** D
	1	Bill	A	B	Sales	
	1	Bill	A	C	Legal	
	2	Sarah	C	A	Marketing	
	2	Sarah	C	B	Sales	
	2	Sarah	C	C	Legal	
	3	John	A	A	Marketing	
	3	John	A	B	Sales	
	3	John	A	C	Legal	

Division

- R ÷ S
  - Defines a relation over the attributes C that consists of set of tuples from R that match combination of *every* tuple in S.

- Expressed using basic operations:

  T1 ← ΠC(R)

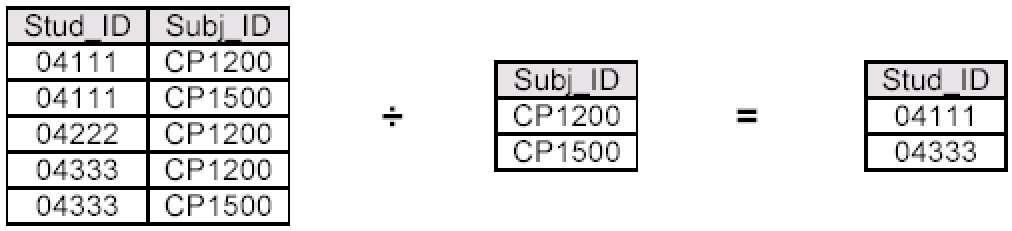
  T2 ← ΠC((S X T1) – R)

  T ← T1 – T2

The division operation (÷) is useful for a particular type of query that sometimes occurs in database applications. For example, if I want to organise a study group,I would like to find people who do the same subjects I do. The division operator provides you with the facility to perform this query without the need to "hard code" the subjects involved.

My_Subjects = ðsubj_code (ó Stud_ID = ‘04111’ (Enrolment))

Study_Group = Enrolment ÷ My_Subjects



Stud_ID	Subj_ID
04111	CP1200
04111	CP1500
04222	CP1200
04333	CP1200
04333	CP1500

÷

Subj_ID
CP1200
CP1500

=

Stud_ID
04111
04333

<u>Joins</u>

Theta-join:-

The **theta-join** operation is the most general join operation. We can define theta-join in terms of the operations that we are familiar with already.

$$R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$$

So the join of two relations results in a subset of the Cartesian product of those relations. Which subset is determined by the *join condition*: θ. Let's look at an example. The result of

Professions $\bowtie_{\text{Job = Job}}$ Careers is shown below.

```
   Name |  Job      | Job       |  Pays
 -------------------------------------------
   Joe   |  Garbageman | Garbageman |  50000
   Sue   |  Doctor     | Doctor     |  40000
   Joe   |  Surfer     | Surfer     |   6500
```

Equi-join:-

The join condition, θ, can be any well-formed logical expression, but usually it is just the conjunction of equality comparisions between pairs of attributes, one from each of the joined relations. This common case is called an **equi-join**. The example given above is an example of an equi-join.

Outer Joins:-

- A join operation is **complete**, if all the tuples of the operands contribute to the result.
- Tuples not participating in the result are said to be **dangling**
- Outer join operations are variants of the join operations in which the dangling tuples are appended with NULL fields. They can be categorized into **left**, **right**, and **full** outer joins.

INSTRUCTOR $_{\text{left}}\bowtie_{\text{right}}$ ADMINISTRATOR

Name	**Number**	**Adm**
Don	3412	Dean
Nel	2341	NULL
NULL	4123	Secretary

INSTRUCTOR $_{\text{left}}\bowtie$ ADMINISTRATOR

Name	**Number**	**Adm**
Don	3412	Dean
Nel	2341	NULL

We will discuss further about relational algebra in the next lecture.

<u>Review Questions</u>

1. Explain the basic operations in Relational Algebra
2. Explain the difference between union and intersection
3. Explain Project operation
4. Explain Renaming operation
5. Explain Cartesian Product