LAB 4 Notes

The Relational Algebra

- Any questions on the project (Discuss)
- In the previous lab we discussed the Conceptual Database Design Phase and the ER Diagram
- Today we will mainly discuss how to convert an ER model into the Relational model of a specific database.

Ch.1: Overview of Database Systems Ch.2: Introduction to Database Design Ch.3: The Relational Model Ch.4: Relational Algebra Ch.5: SQL Ch.8: Storage and Indexing	Ch.9: Storing Data: Disks and Files Ch.10: Tree-Structured Indexing Ch.11: Hash-Based Indexing Ch.12: Overview of Query Evaluation Ch.13: External Sorting Ch.14: Evaluation of Relational Operators Ch.15: A Typical Relational Query Optimizer Ch.16: Overview of Transaction

Outline
1) Glace at Relational Algebra Operators.

Selection

The **selection** operation selects tuples from a relation that fit some criteria, creating a new relation with the selected tuples. We will use the notation

$\sigma_C(R) = \{ t \mid C \text{ is true for } t \}$

where σ is the selection operator, *C* is the **selection condition**, and *R* is a relation. The selection condition is a well-formed logical expression built from the following rules:

- a comparison operation between attribute names or attribute values, and
- the standard logical connectives: **AND**, **OR**, and **NOT**.

Some example conditions are given below for a relation with `Name` and `Age` attributes.

```
Name = 'Sue'
Name = 'Sue' AND Age > 23
NOT (Name = 'Sue' AND Age > 23)
```

Let's look at some example of selection, and the meaning will become clear. Consider the relation `Professions`.

Professions
```
Name  | Job
------------------
Joe   | Garbageman
Sue   | Doctor
Joe   | Surfer
```

Now consider the following selections and their results.

$\sigma_{\text{Name = Sue}}(\text{Professions}) =$

$\{t \mid t.\text{Name} = \text{Sue}\} =$

$\{(\text{Sue, Doctor})\}$

$\sigma_{\text{Name = Sue OR Job = Surfer}}$(`Professions`) =

{t | t.Name = Sue **OR** \$t.Job = Surfer} =

{(Sue, Doctor), (Joe, Surfer)}

$\sigma_{\text{Name = Sue AND Job = Surfer}}$(`Professions`) =

{t | t.Name = Sue **AND** t.Job = Surfer} =

{}

What does selection do in terms of the table metaphor? It merely selects those rows from the table that satisfy the selection condition, ignoring the rest. Note that the selected rows form a new table (possibly an empty table).

Projection

The **projection** operation projects out a list of attributes from a relation. For example, suppose we have a relation with the schema $R(A_1, A_2, ..., A_N)$ and we want only the first M attributes

$$\pi_{A1, A2, ..., AM}(R) = \{ (t[A_1], t[A_2], ... t[A_M]) \mid t \in R \}$$

where π is the projection operator, $A_1, A_2, ..., A_M$ is a list of the first M attributes, and R is a relation. In general, we can project any of the attributes in a relation in any order. Let's look at some examples from the `Professions` relation depicted above.

- π_{Job}(`Professions`) would produce the following relation.

```
    Job
--------------
    Garbageman
    Doctor
    Surfer
```

- π_{Name}(`Professions`) would produce the following relation (assuming we retain duplicates)

```
    Name
--------
    Joe
    Sue
    Joe
```

  or this table (assuming we eliminate duplicates)

```
    Name
--------
    Joe
    Sue
```

Cartesian product of relations

The **Cartesian product** operation is similar to that for sets. Basically the Cartesian product produces a relation consisting of all possible pairings of tuples as follows. Assume we have relations $R(A_1, A_2, ..., A_N)$ and $S(B_1, B_2, ..., B_M)$ Then

$R \times S = \{((a_1, a_2, ..., a_N, b_1, b_2, ..., b_M) \mid (a_1, a_2, ..., a_N) \in R$ **AND** $(b_1, b_2, ..., b_M) \in S \}$

Note that $R \times S$ is not the same as $S \times R$ because the order of attributes differs.

Let's look at an example. Assume that in addition to the `Professions` relation, we have a `Salaries` relation.

Salaries

Job	Pays
Garbageman	50000
Doctor	40000
Surfer	6500

The result of `Professions` × `Careers` is depicted below.

Name	Job	Job	Pays
Joe	Garbageman	Garbageman	50000
Joe	Garbageman	Doctor	40000
Joe	Garbageman	Surfer	6500
Sue	Doctor	Garbageman	50000
Sue	Doctor	Doctor	40000
Sue	Doctor	Surfer	6500
Joe	Surfer	Garbageman	50000
Joe	Surfer	Doctor	40000
Joe	Surfer	Surfer	6500

Note that we have two attributes now with the same name, `Job`, we will assume that one of the attributes is renamed appropriately.

Union, Intersection, Difference

Since a relation is just a set (or multiset), the set (or multiset) algebra operations, **union**, **intersection**, and **difference**, are also present in the relational algebra, with one constraint. These operations are only permitted between relations that are **union compatible**. Two relations are union compatible if they have the same number of attributes, and if the i^{th} attribute in each relation has the same domain. Basically, the two relations must have the same schemas, modulo renaming of the attributes, which makes a lot of sense since you really do not want two completely different kinds of tuples in the same relation.

A complete set of operations

We now have a complete set of relational algebra operations. Any other operator that we might introduce, such as a *join*, is merely for our notational convenience.

Joins

In general, a **join** is an operation that glues relations together. There are several kinds of joins.

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` $\bowtie_{\texttt{Job = Job}}$ `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.

Natural join

Note that in the result of an equi-join, the join attributes are duplicated. A **natural join** is an equi-join that projects away duplicated attributes. If θ is omitted from a $\bowtie$ we will assume that the operation is a natural join. Let

$R = (A_1,\ldots,A_n,X_1,\ldots,X_m)$

and

$S = (X_1,\ldots,X_m,B_1,\ldots,B_k)$

Then

$R \bowtie S = \pi_{A1,\ldots,An,X1,\ldots,Xm,B1,\ldots,Bk}(R \bowtie_{X1 = X1 \textbf{ AND } \ldots \textbf{ AND } Xm = X1} S)$

(We assume that the join attributes have been made distinct via renaming appropriately.)

Let's look at an example. The result of

`Professions` $\bowtie$ `Careers`

is shown below.

Name	Job	Pays
Joe	Garbageman	50000
Sue	Doctor	40000
Joe	Surfer	6500

Reordering columns in a table

How do I go about swapping columns in a relation? I use projection? Assume I have relation

$S = (A_1, A_2, A_3)$

I want a relation that is just like *S* but with exactly the opposite order of attributes. Then I would do

$\pi_{A3,\ A2,\ A1}(S)$

the result is *S* with the columns swapped.

Examples of Relational Algebra

Consider the following relations (depicted as tables).

```
  STUDENTS
    name | subject
   ----------------
    joe  |  CP1500
    joe  |  CP1200
    sue  |  CP3020
```

PARENTOF

```
Parent| Name
----------------
pam   | joe
pam   | sue
ann   | pam
eric  | ann
```

The Cartesian product of these relations,

PARENTOF × STUDENTS

would result in the following relation.

```
    parent | name | name | subject
 ----------------------------------
     pam   |  joe |  joe | CP1500
     pam   |  joe |  joe | CP1200
     pam   |  joe |  sue | CP3020
     pam   |  sue |  joe | CP1500
     pam   |  sue |  joe | CP1200
     pam   |  sue |  sue | CP3020
     ann   |  pam |  joe | CP1500
     ann   |  pam |  joe | CP1200
     ann   |  pam |  sue | CP3020
     eric  |  ann |  joe | CP1500
     eric  |  ann |  joe | CP1200
     eric  |  ann |  sue | CP3020
```

The equi-join (on the `name` attribute),

PARENTOF $\bowtie_{name = name}$ STUDENTS

would result in the following relation.

```
    parent | name | name | subject
 ----------------------------------
     pam   |  joe |  joe | CP1500
     pam   |  joe |  joe | CP1200
     pam   |  sue |  sue | CP3020
```

Finally, the natural join,

PARENTOF $\bowtie$ STUDENTS

would yield the following.

```
    parent | name | subject
 ----------------------------------
     pam   |  joe | CP1500
     pam   |  joe | CP1200
     pam   |  sue | CP3020
```

Now let's consider several examples using these relations.

Reordering columns example

Suppose I want a relation like STUDENTS, but with the subject first, then the name. I would do

$\pi_{\texttt{subject, name}}(\texttt{STUDENTS})$

What are the names of the students?:

$\texttt{STUDENTNAMES} = \pi_{\texttt{name}}(\texttt{STUDENTS})$

Who is taking *CP1500*?:

$\texttt{CP1500} = \pi_{\texttt{name}}(\sigma_{\texttt{subject} = \mathit{CP1500}}(\texttt{STUDENTS}))$

Who is the parent of *joe*?:

$\texttt{JOES\_PARENTS} = \pi_{\texttt{parent}}(\sigma_{\texttt{name} = \mathit{joe}}(\texttt{PARENTOF}))$

The above three examples were all operations on a single table. We must use a join to combine information from two or more tables.

Who is the parent of a student taking *CP1500*?: In this example, we make use of the result of a previous query, the CP1500 relation is computed above.

$\texttt{CP1500\_PARENTS} = \pi_{\texttt{name}}(\texttt{PARENTOF} \bowtie \texttt{CP1500})$

Who is the grandparent of of a student taking *CP1500*?:

$\texttt{CP1500\_GRANDPARENTS} = \pi_{\texttt{name}}(\texttt{PARENTOF} \bowtie \texttt{CP1500\_PARENTS})$