

SQL meets Maths

With only minor additions and adaptations, yesterday's bit of pure maths becomes a useful theory of databases.

The plan for today's lectures is to

- start with a simple SQL query
- develop the maths until we can express that same query in relational algebra
- along the way, see what algebraic rules we can use to manipulate it

This material is from [E&N §6.1 - 6.5, §8.4]

- Assignment 1 is on the course web page, due 5pm, Friday 5th September. No marks for late submissions.
 - both paper and electronic submission required
 - please read assignment sheet carefully and understand what is required
 - see the “pick and pack” example
- Mid-semester exam is on Wednesday 10th September, starting 12:50, for one hour, at Melville Hall.
 - last years mid-test paper is on the course web page

An Example Query

Our toy organisation wants to review employees in finance and administration departments.

(Database from Lab 1)

```
SELECT enumber, fname, lname, dname
FROM employee, department
WHERE (dname='Administration' OR dname='Finance')
AND dnumber=dno;
```

We will express this query as mathematics.

Set Size

We write $|A|$ for the number of elements in a set A .

Here's a few cute facts about set size.

- $|A \times B| = |A| \times |B|$
- $|\prod_{i \in I} A_i| = \prod_{i \in I} |A_i|$
- ie $|\bullet|$ is a product *homomorphism*
- $|A \cup B| \leq |A| + |B|$
- $|A \cap B| \leq |A|$ and $|A \cap B| \leq |B|$
- $|A - B| \geq |A| - |B|$

Projections

Recall that an indexed set is a set whose elements have names.

We can use those names to extract the elements.

$$\pi_{city}\{city = Paris, custID = 11\} = \{city = Paris\}$$

$$\pi_{1,3}(2, 11, 42) = (2, 42)$$

We specify a subset of the index set (names), and in return, we get the corresponding subset of the indexed set (name=value pairs). **This called a projection.**

(Note that the second, tuple version, does some renaming)

About Projection

- We can apply a projection to a set of indexed sets (a relation). Write $\pi_i\{A, B, C\}$ for $\{\pi_iA, \pi_iB, \pi_iC\}$.
- When does $\pi_X(\pi_YA)$ make sense?
- When $X \subseteq Y$, in which case $\pi_X(\pi_YA) = \pi_XA$.
- Its quicker to pass through the relation just once.
- $|\pi_XA| \leq |A|$ Why not =?
- When we lose some of the named elements of each tuple, two that were distinct may cease to be.
- eg $|\pi_1\{(3, 1), (3, 2)\}| = |\{3\}| = 1 < 2 = |\{(3, 1), (3, 2)\}|$

Products: Names and Numbers

There is some ambiguity in the ideas of relational database theory. Here is an attempt to deal with it “nicely”.

- For the generalised idea of product, the “tuples” need not be ordered.

$$\{city = Paris, custID = 42\} = \{custID = 42, city = Paris\}$$

- But sometimes its nice to have them in some order.
- So, we will think of each index set I as being indexed by $\{1, \dots, |I|\}$.
- Then if $I_1 = custID$ and $I_2 = city$ then we can write
 - $(42, Paris)$ for $\{custID = 42, city = Paris\}$
 - π_2 for π_{city}

Products: Flat and Structured

The product of two relations can have two attributes with the same name.

If so, we write `table.attribute` to disambiguate.

```
SELECT lname, deptName
FROM employee, department
WHERE employee.deptID = department.deptID
```

Products: Flat and Structured

Allowing both qualified (`table.attribute`) and unqualified (`attribute`) forms means that our products are both flat and structured.

The first way is technically correct: let

$$t \in Q = \prod_{i \in \{employee, department\}} R_i$$

then to get t 's name we must take $\pi_{lname}(\pi_{employee}(t))$.

That is,

$t = \{employee = \{lname = 'Smith', \dots\}, department = \{\dots\}\}$
is *structured*.

Back to the Example: Products and SQL

Throw away the `WHERE` clause from our example query, and we have a product and a projection.

```
SELECT fname, lname, dname
FROM employee, department;
```

Mathematically, this is

$$\pi_{fname, lname, dname}(employee \times department)$$

Products: Flat and Structured

- In practice it is unnecessarily verbose to always give the table name to qualify the attribute.
- So, we (ambiguously) think of Q as having the attributes of both $R_{employee}$ and $R_{department}$. That is, it is indexed by the union of the index sets of those two relations.
- ie, the relation is *flat*: $t = \{lname = 'Smith', \dots\}$
- Also unnecessarily verbose is $Q = blah\ blah\ blah$ above. Instead we will write

$$Q = employee \times department$$

Observations

- $|employee| = 9$, $|department| = 3$,
 $|employee \times department| = 27$
- Everybody works for every department!?!

Look at the middle columns resulting from the following query.

```
SELECT *,
department.dnumber, department.dname
FROM employee, department;
```

This shows us what foreign keys are for. To get tuples of employee data augmented by that employees department data, we add

```
WHERE employee.dno = department.dnumber
```

Select

- To do this in our mathematical query, we must define a new operation on relations, *select*.
- Do not confuse SQL's `SELECT` with the mathematical *select*. They are related, but not the same.
- Select is the only really new thing we need. The rest of relational algebra can be defined from maths we have already done.
- So, we will look at select next lecture.