Lecture 8

Database Construction and Usage

SQL DDL and DML
Relational Algebra

Queries:
SQL and Relational Algebra

Querying

- To *query* the database means asking it for information.
  - ”List all courses that have lectures in room VR”
- Unlike a modification, a query leaves the database unchanged.

SQL

- SQL = Structured Query Language
  - The querying parts are really the core of SQL. The DDL and DML parts are secondary.
- Very-high-level language.
  - Specify *what* information you want, not *how* to get that information (like you would in e.g. Java).
- Based on Relational Algebra

”Algebra”

- An *algebra* is a mathematical system consisting of:
  - Operands: variables or values to operate on.
  - Operators: symbols denoting functions that operate on variables and values.

Relational Algebra

- An algebra whose operands are relations (or variables representing relations).
- Operators representing the most common operations on relations.
  - Selecting rows
  - Projecting columns
  - Composing (joining) relations

Selection

- Selection = Given a relation (table), choose what tuples (rows) to include in the result.

$\sigma_C(T)$	**`SELECT * FROM T WHERE C;`**

  - Select the rows from relation T that satisfy condition C.
  - σ = sigma = greek letter **S** = **S**election

Example:

GivenCourses =

course	*per*	*teacher*
TDA357	3	Niklas Broberg
TDA357	2	Graham Kemp
TIN090	1	Devdatt Dubhashi

```
SELECT *
FROM   GivenCourses
WHERE  course = 'TDA357';
```

Result =

What?

Example:

GivenCourses =

course	*per*	*teacher*
TDA357	3	Niklas Broberg
TDA357	2	Graham Kemp
TIN090	1	Devdatt Dubhashi

```
SELECT *
FROM   GivenCourses
WHERE  course = 'TDA357';
```

Result =

course	*per*	*teacher*
TDA357	3	Niklas Broberg
TDA357	2	Graham Kemp

Projection

- Given a relation (table), choose what attributes (columns) to include in the result.

$\pi_X(\sigma_C(T))$ **`SELECT X FROM T WHERE C;`**

  - Select the rows from table T that satisfy condition C, and project columns X of the result.
  - π = pi = greek letter **p** = **p**rojection

Example:

GivenCourses =

course	*per*	*teacher*
TDA357	3	Niklas Broberg
TDA357	2	Graham Kemp
TIN090	1	Devdatt Dubhashi

```
SELECT course, teacher
FROM   GivenCourses
WHERE  course = 'TDA357';
```

Result =

What?

Example:

GivenCourses =

course	*per*	*teacher*
TDA357	3	Niklas Broberg
TDA357	2	Graham Kemp
TIN090	1	Devdatt Dubhashi

```
SELECT course, teacher
FROM   GivenCourses
WHERE  course = 'TDA357';
```

Result =

course	*teacher*
TDA357	Niklas Broberg
TDA357	Graham Kemp

The confusing **SELECT**

Example:

GivenCourses =

course	*per*	*teacher*
TDA357	3	Niklas Broberg
TDA357	2	Graham Kemp
TIN090	1	Devdatt Dubhashi

```
SELECT course, teacher
FROM   GivenCourses;
```

Result =

What?

The confusing **SELECT**

Example:

GivenCourses =

course	*per*	*teacher*
TDA357	3	Niklas Broberg
TDA357	2	Graham Kemp
TIN090	1	Devdatt Dubhashi

```
SELECT course, teacher
FROM   GivenCourses;
```

Result =

course	*teacher*
TDA357	Niklas Broberg
TDA357	Graham Kemp
TIN090	Devdatt Dubhashi

Quiz: **SELECT** is a projection??

Mystery revealed!

```
SELECT course, teacher
  FROM GivenCourses;
```

$$\pi_{code,teacher}(\sigma(GivenCourses))$$
$$=\pi_{code,teacher}(GivenCourses)$$

- In general, the SELECT clause could be seen as corresponding to projection, and the WHERE clause to selection (don’t confuse the naming though).

Quiz!

- What does the following expression compute?

Courses

code	*name*
TDA357	Databases
TIN090	Algorithms

GivenCourses

course	*per*	*teacher*
TDA357	3	Niklas Broberg
TDA357	2	Graham Kemp
TIN090	1	Devdatt Dubhashi

```
SELECT *
FROM   Courses, GivenCourses
WHERE  teacher = 'Niklas Broberg';
```

FROM Courses, GivenCourses

code	*name*	*course*	*per*	*teacher*
TDA357	Databases	TDA357	3	Niklas Broberg
TDA357	Databases	TDA357	2	Graham Kemp
TDA357	Databases	TIN090	1	Devdatt Dubhashi
TIN090	Algorithms	TDA357	3	Niklas Broberg
TIN090	Algorithms	TDA357	2	Graham Kemp
TIN090	Algorithms	TIN090	1	Devdatt Dubhashi

WHERE teacher = 'Niklas Broberg'

code	*name*	*course*	*per*	*teacher*
TDA357	**Databases**	**TDA357**	**3**	**Niklas Broberg**
TDA357	Databases	TDA357	2	Graham Kemp
TDA357	Databases	TIN090	1	Devdatt Dubhashi
TIN090	**Algorithms**	**TDA357**	**3**	**Niklas Broberg**
TIN090	Algorithms	TDA357	2	Graham Kemp
TIN090	Algorithms	TIN090	1	Devdatt Dubhashi

Answer:

```
SELECT *
FROM   Courses, GivenCourses
WHERE  teacher = 'Niklas Broberg';
```

code	*name*	*course*	*per*	*teacher*
TDA357	Databases	TDA357	3	Niklas Broberg
TIN090	Algorithms	TDA357	3	Niklas Broberg

The result is all rows from **`Courses`** combined in all possible ways with all rows from **`GivenCourses`**, and then keep only those where the **`teacher`** attribute is Niklas Broberg.

Cartesian Products

- The *cartesian product* of relations R_1 and R_2 is all possible combinations of rows from R_1 and R_2.
  - Written $R_1 \times R_2$
  - Also called *cross-product*, or just *product*

```
SELECT *
FROM   Courses, GivenCourses
WHERE  teacher = 'Niklas Broberg';
```

$$\sigma_{\text{teacher} = \text{'Niklas Broberg'}}(\text{Courses} \times \text{GivenCourses})$$

Quiz!

List all courses, with names, that Niklas Broberg is responsible for.

Courses(code,name)

GivenCourses(course,per,teacher)

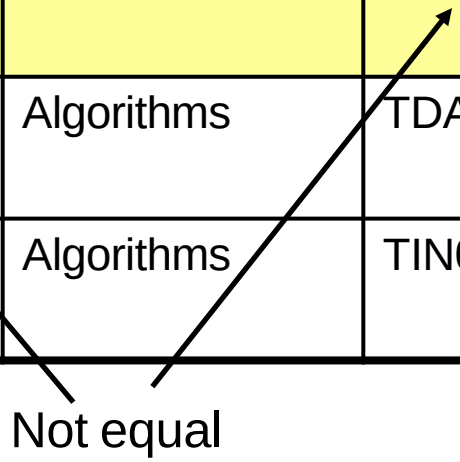
course -> Courses.code

```
SELECT *
FROM   Courses, GivenCourses
WHERE  teacher = 'Niklas Broberg'
  AND  code = course;
```

code	*name*	*course*	*per*	*teacher*
TDA357	Databases	TDA357	3	Niklas Broberg

code = course

code	*name*	*course*	*per*	*teacher*
TDA357	**Databases**	**TDA357**	**3**	**Niklas Broberg**
TDA357	Databases	TDA357	2	Graham Kemp
TDA357	Databases	TIN090	1	Devdatt Dubhashi
TIN090	**Algorithms**	**TDA357**	**3**	**Niklas Broberg**
TIN090	Algorithms	TDA357	2	Graham Kemp
TIN090	Algorithms	TIN090	1	Devdatt Dubhashi



Joining relations

- Very often we want to join two relations on the value of some attributes.
  - Typically we join according to some reference, as in:

```
SELECT *
FROM   Courses, GivenCourses
WHERE  code = course;
```

- Special operator $\bowtie_C$ for joining relations.

$$R_1 \bowtie_C R_2 = \sigma_C(R_1 \times R_2)$$

```
SELECT *
FROM   R1 JOIN R2 ON C;
```

Example

Courses

code	*name*
TDA357	Databases
TIN090	Algorithms

GivenCourses

course	*per*	*teacher*
TDA357	3	Niklas Broberg
TDA357	2	Graham Kemp
TIN090	1	Devdatt Dubhashi

```
SELECT *
FROM   Courses JOIN GivenCourses
  ON   code = course;
```

code	*name*	*course*	*per*	*teacher*
TDA357	Databases	TDA357	3	Niklas Broberg
TDA357	Databases	TDA357	2	Graham Kemp
TIN090	Algorithms	TIN090	1	Devdatt Dubhashi

Natural join

- ”Magic” version of join.
  - Join two relations on the condition that all attributes in the two that share the same name should be equal.
  - Remove all duplicate columns
  - Written $R_1 \bowtie R_2$ (like join with no condition)

Example

Courses

code	*name*
TDA357	Databases
TIN090	Algorithms

GivenCourses

code	*per*	*teacher*
TDA357	3	Niklas Broberg
TDA357	2	Graham Kemp
TIN090	1	Devdatt Dubhashi

```
SELECT *
FROM   Courses NATURAL JOIN GivenCourses;
```

code	*name*	*per*	*teacher*
TDA357	Databases	3	Niklas Broberg
TDA357	Databases	2	Graham Kemp
TIN090	Algorithms	1	Devdatt Dubhashi

Sets or Bags?

- Relational algebra formally applies to sets of tuples.
- SQL, the most important query language for relational databases is actually a bag language.
  - SQL will eliminate duplicates, but usually only if you ask it to do so explicitly.
- Some operations, like projection, are much more efficient on bags than sets.

Sets or Bags?

R(A,B)

A	B
1	2
5	6
1	3

SQL

```
SELECT A
FROM R
```

A
1
5
1

Bag

Relational Algebra

$\pi_A(R)$

A
1
5

Set
(no repeating values)