

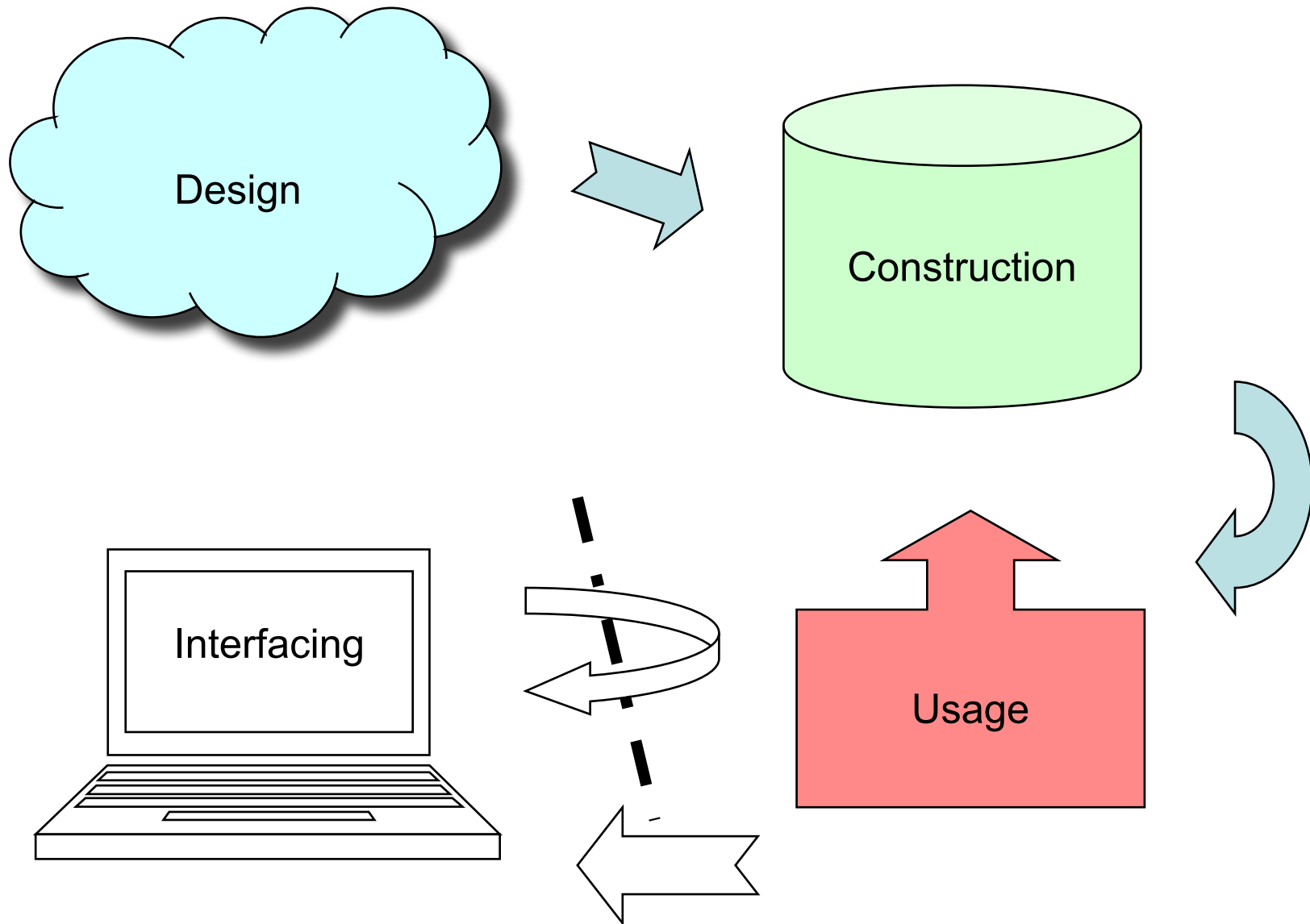
Database Construction and Usage

SQL DDL and DML
Relational Algebra

Announcement

- Sign up to the Google Group for updates!
 - <https://groups.google.com/forum/#!forum/tda357-ht2016>
- Fill in the doodles
 - No-one signed up == no TA attending
 - More rooms are added if needed

Course Objectives



Connecting to PostgreSQL

- Chalmers postgresql server:

```
psql -h ate.ita.chalmers.se -U <username> <dbname>
```

- Local postgresql server:

```
psql <dbname>
```

- Semicolon and postgres prompt:

```
steven=> select 1+1
```



```
steven-> ;
```



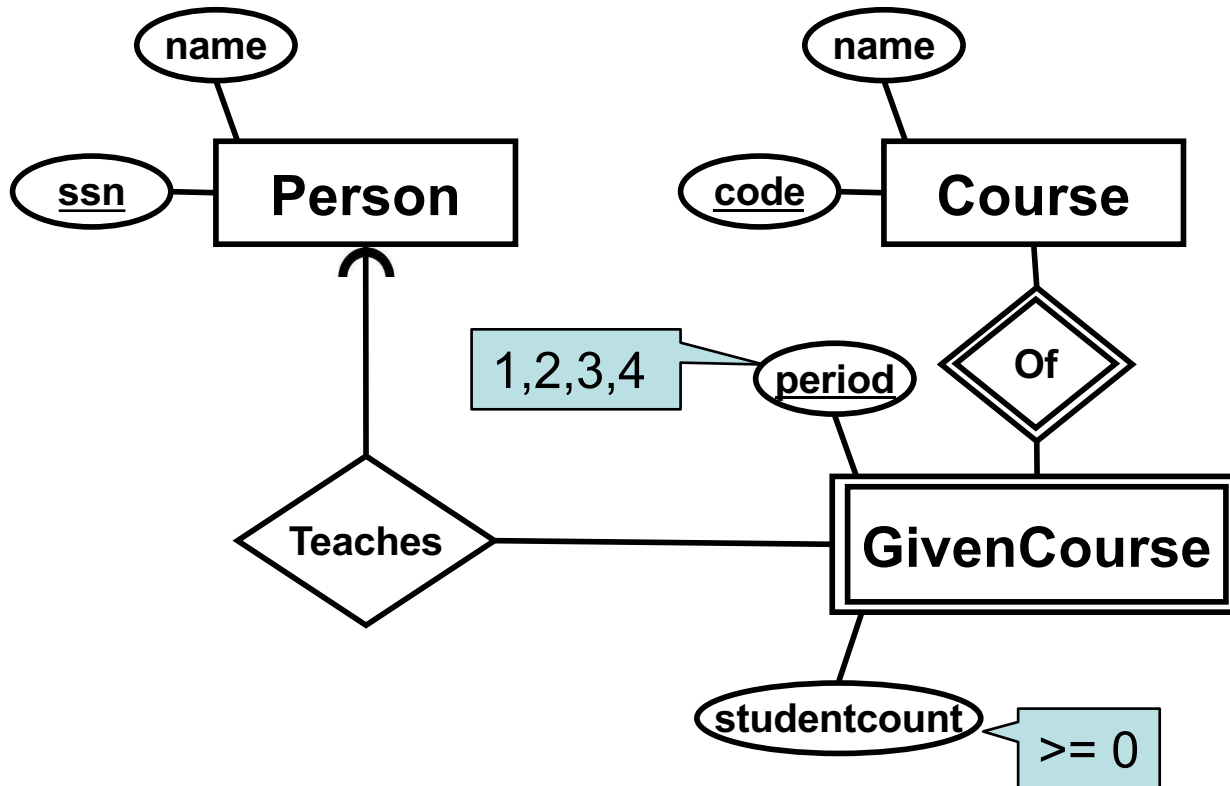
Lines should end with ';', otherwise statements are continued on the next line. Note the prompt change!

Case convention

- SQL is completely case insensitive. Upper-case or Lower-case makes no difference. We will use case in the following way:
 - **UPPERCASE** marks keywords of the SQL language.
 - **lowercase** marks the name of an attribute.
 - **Capitalized** marks the name of a table.

SQL Data Definition Language

Working example



`Person(ssn, name)`

`Course(code, name)`

`GivenCourse(code, period, studentcount, teacher)`

`code -> Course.code`

`teacher -> Person.ssn`

Creating and dropping tables

- Relations become tables, attributes become columns.

```
CREATE TABLE Tablename (  
    <list of table elements>  
);
```

- Get all info about a created table:

```
\d+ Tablename;
```

PostgreSQL specific!

- Remove a created table:

```
DROP TABLE Tablename;
```

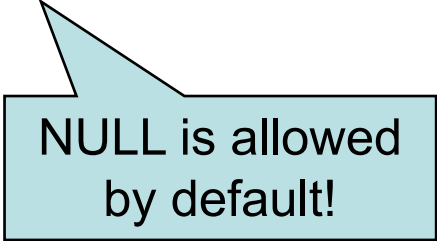

Table declaration elements

- The basic elements are pairs consisting of a column name and a type.
- Most common SQL types:
 - INT or INTEGER (synonyms)
 - REAL or FLOAT (synonyms)
 - CHAR(n) = fixed-size string of size n .
 - VARCHAR(n) = variable-size string of up to size n .
 - TEXT = string of unrestricted length

Example

Example:

```
CREATE TABLE Courses (  
    code CHAR(6) ,  
    name TEXT NOT NULL  
);
```



NULL is allowed
by default!

Created the table courses:

code	name
------	------

Declaring keys

- An attribute or a list of attributes can be declared PRIMARY KEY or UNIQUE
 - PRIMARY KEY: (At most) One per table, never NULL. Efficient lookups in all DBMS.
 - UNIQUE: Any number per table, can be NULL. Could give efficient lookups (may vary in different DBMS).
- Both declarations state that all other attributes of the table are functionally determined by the given attribute(s).

Example

```
CREATE TABLE Courses(  
    code CHAR(6),  
    name TEXT NOT NULL,  
    PRIMARY KEY (code)  
);
```

Foreign keys

- Referential constraints are handled with references, called *foreign keys*.
 - FOREIGN KEY *attribute*
REFERENCES *table(attribute)*.

FOREIGN KEY course

REFERENCES Courses (code)

Foreign keys

- General:

```
FOREIGN KEY course REFERENCES Courses (code)
```

- If course is Primary Key in Courses:

```
FOREIGN KEY course  
REFERENCES Courses
```

- Give a name to the foreign key:

```
CONSTRAINT ExistsCourse  
FOREIGN KEY course  
REFERENCES Courses
```

Example

```
CREATE TABLE GivenCourses (  
    course          CHAR(6),  
    period          INT,  
    numStudents    INT,  
    teacher        INT NOT NULL,  
    PRIMARY KEY (course, period),  
    FOREIGN KEY (course) REFERENCES Courses(code)  
    FOREIGN KEY (teacher) REFERENCES People(ssn)  
);
```

Example

```
CREATE TABLE GivenCourses (  
    course CHAR(6) REFERENCES Courses,  
    period          INT,  
    numStudents     INT,  
    teacher         INT REFERENCES People(ssn) NOT  
    NULL,  
    PRIMARY KEY (course, period)  
);
```


Value constraints

- Use CHECK to insert simple value constraints.
 - CHECK (*some test on attributes*)

CHECK (period IN (1,2,3,4))

Example

```
CREATE TABLE GivenCourses (  
    course CHAR(6) REFERENCES Courses,  
    period          INT CHECK (period IN (1,2,3,4)),  
    numStudents     INT,  
    teacher         INT REFERENCES People(ssn) NOT  
    NULL,  
    PRIMARY KEY (course, period)  
);
```

Example

```
CREATE TABLE GivenCourses (  
    course CHAR(6) REFERENCES Courses,  
    period          INT,  
    numStudents     INT,  
    teacher         INT REFERENCES People(ssn) NOT  
    NULL,  
    PRIMARY KEY (course, period),  
    CONSTRAINT ValidPeriod CHECK (period in (1,2,3,4))  
);
```

Example

- Legal:
 - INSERT INTO GivenCourses
VALUES ('TDA357', 2, 199, 1);
- Not Legal:
 - INSERT INTO GivenCourses
VALUES ('TDA357', 7, 199, 1);
- ERROR: new row for relation
"givencourses" violates check constraint
"givencourses_period_check"DETAIL:
Failing row contains (TDA357, 7, 199, 1).

SQL Data Manipulation Language: Modifications

Inserting data

```
INSERT INTO tablename  
VALUES (values for attributes);
```

```
INSERT INTO Courses  
VALUES ('TDA357', 'Databases');
```

<i>code</i>	<i>name</i>
TDA357	Databases

Deletions

```
DELETE FROM tablename  
WHERE test over rows;
```

```
DELETE FROM Courses  
WHERE code = 'TDA357' ;
```

Updates

```
UPDATE  tablename  
SET     attribute = ...  
WHERE   test over rows
```

```
UPDATE  GivenCourses  
SET     teacher = 'Graham Kemp'  
WHERE   course  = 'TDA357'  
        AND     period = 2;
```


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** = **Selection**

Example:

GivenCourses =

<u>course</u>	<u>per</u>	teacher
TDA357	3	Niklas Broberg
TDA357	2	Graham Kemp
TIN090	1	Devdatt Dubhashi

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

Result =

What?

Example:

GivenCourses =

<i>course</i>	<i>per</i>	<i>teacher</i>
TDA357	3	Niklas Broberg
TDA357	2	Graham Kemp
TIN090	1	Devdatt Dubhashi

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

Result =

<i>course</i>	<i>per</i>	<i>teacher</i>
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 =

<u>course</u>	<u>per</u>	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 =

<i>course</i>	<i>per</i>	<i>teacher</i>
TDA357	3	Niklas Broberg
TDA357	2	Graham Kemp
TIN090	1	Devdatt Dubhashi

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

Result =

<i>course</i>	<i>teacher</i>
TDA357	Niklas Broberg
TDA357	Graham Kemp

The confusing **SELECT**

Example:

GivenCourses =

<u>course</u>	<u>per</u>	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 =

<u>course</u>	<u>per</u>	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;
```

$$\begin{aligned} & \pi_{\text{code,teacher}}(\sigma(\text{GivenCourses})) \\ &= \pi_{\text{code,teacher}}(\text{GivenCourses}) \end{aligned}$$

- 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

<u>code</u>	name
TDA357	Databases
TIN090	Algorithms

GivenCourses

<u>course</u>	<u>per</u>	teacher
TDA357	3	Niklas Broberg
TDA357	2	Graham Kemp
TIN090	1	Devdatt Dubhashi

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

FROM Courses, GivenCourses

<i>code</i>	<i>name</i>	<i>course</i>	<i>per</i>	<i>teacher</i>
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'**

<i>code</i>	<i>name</i>	<i>course</i>	<i>per</i>	<i>teacher</i>
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' ;
```

<i>code</i>	<i>name</i>	<i>course</i>	<i>per</i>	<i>teacher</i>
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;

<i>code</i>	<i>name</i>	<i>course</i>	<i>per</i>	<i>teacher</i>
TDA357	Databases	TDA357	3	Niklas Broberg

code = course

<i>code</i>	<i>name</i>	<i>course</i>	<i>per</i>	<i>teacher</i>
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

Not equal



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

<u>code</u>	name
TDA357	Databases
TIN090	Algorithms

GivenCourses

<u>course</u>	<u>per</u>	teacher
TDA357	3	Niklas Broberg
TDA357	2	Graham Kemp
TIN090	1	Devdatt Dubhashi

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

<i>code</i>	<i>name</i>	<i>course</i>	<i>per</i>	<i>teacher</i>
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

<u>code</u>	name
TDA357	Databases
TIN090	Algorithms

GivenCourses

<u>code</u>	<u>per</u>	teacher
TDA357	3	Niklas Broberg
TDA357	2	Graham Kemp
TIN090	1	Devdatt Dubhashi

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

<i>code</i>	<i>name</i>	<i>per</i>	<i>teacher</i>
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)

Next time, Lecture 6

More Relational Algebra, SQL,
Views