GRUNDLÆGGENDE PROGRAMMERING

SQL I: Introduction to SQL and Databases!

Claus Brabrand

(((brabrand@itu.dk)))

Associate Professor, Ph.D.

(((Software and Systems)))

IT University of Copenhagen

AGENDA

- **Introduction to Databases:**

TEORI

- 1) Create a database: 'CREATE TABLE'
- 2) Insert data into database: 'INSERT INTO'
- 3) Query a database: 'SELECT FROM'

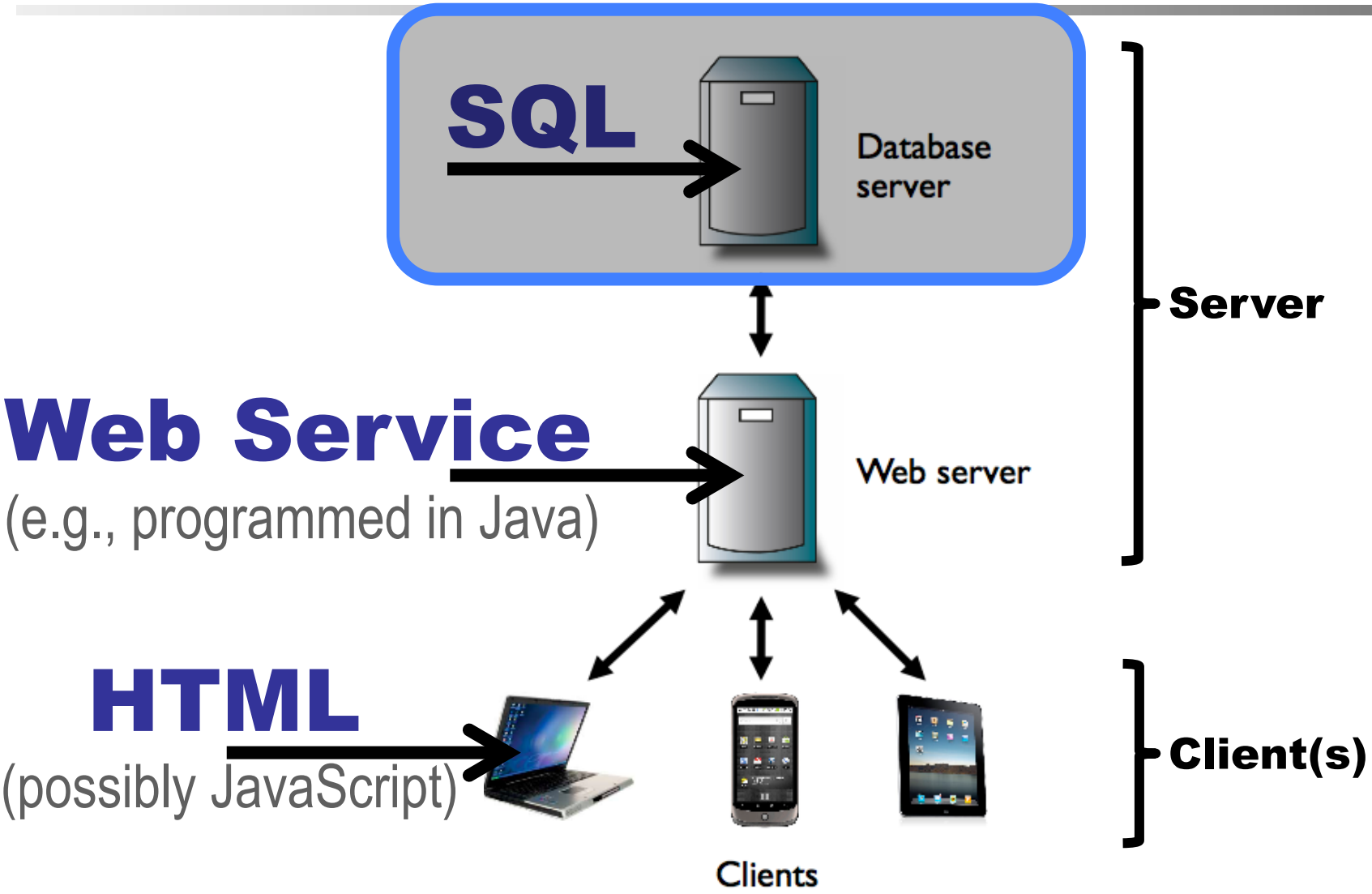
- **How to create + connect to a MySQL database**

- **Introduction to Databases:**

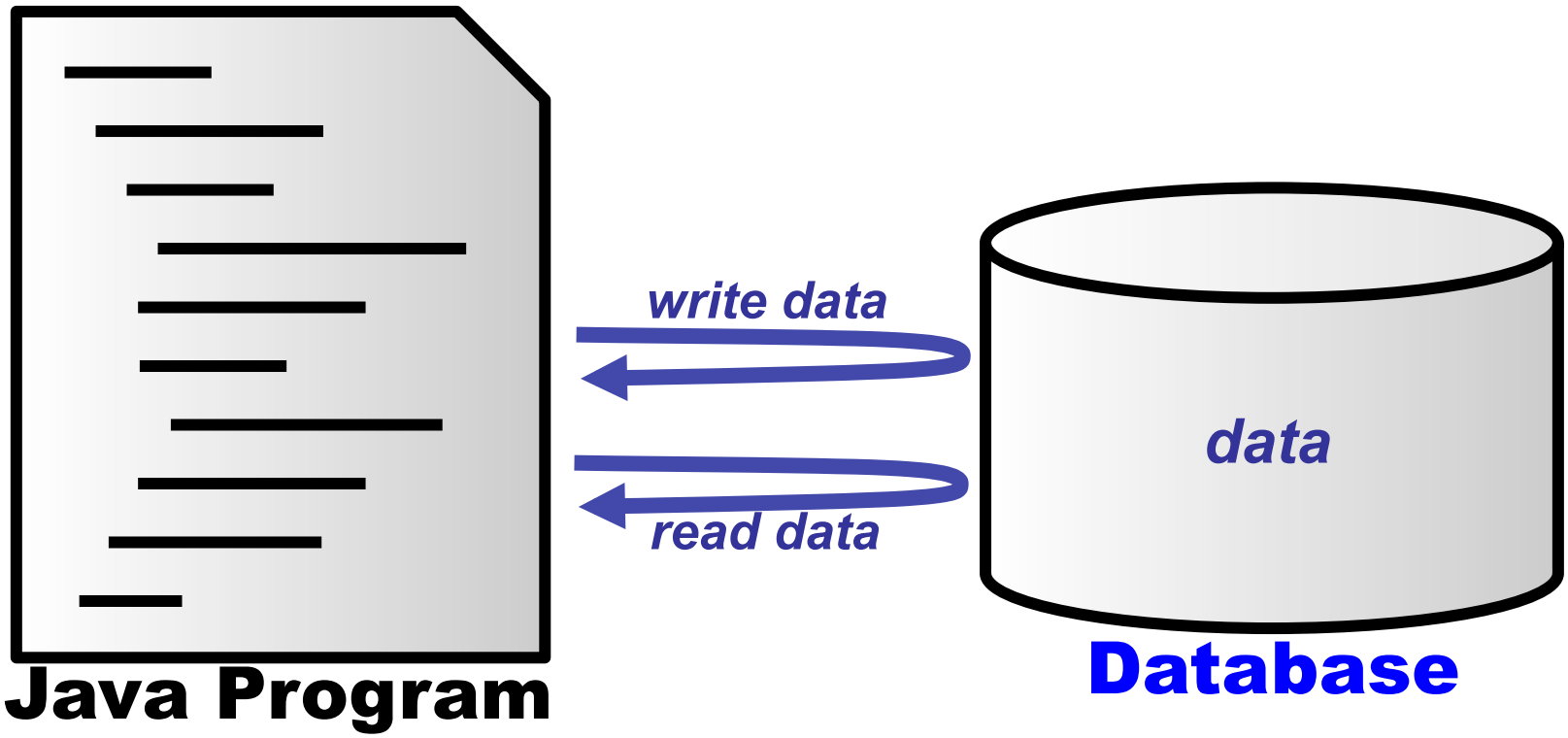
DEMO

- 1) Create a database: 'CREATE TABLE'
- 2) Insert data into database: 'INSERT INTO'
- 3) Query a database: 'SELECT FROM'

Web Service Architecture



Java Program with Database



SQL (Database Language)

- **SQL** is a (relational) **Database Language**:
  - Originally "SEQUEL" (Structured English Query Language)
  - Invented in the 1970s and popular since 1980s
  - Most common way to access (relational) databases
  - (Other kinds of databases: *"object-oriented"*, *"spatial"*)

Example:

```
SELECT first_name FROM users WHERE last_name = 'Hansen';
```

- We will use **MySQL** (free SQL implementation):
  - ***robust***, ***efficient***, ***free*** (www.mysql.com)
  - ...and used a lot in the ***software industry!***

Entities (aka, Tables)

A relational database is built on...:

- ***Entities (aka, Tables):***

(Sort of kind of like an **Excel spreadsheet !**)

stud_name	stud_id	address	course_id	course_name	teacher
Anna	1234	Somewhere 3	GRPRO	Grundlæggende Programmering	Claus
Bent	0001	Homestreet 4	GRPRO	Grundlæggende Programmering	Claus
Carl	0002	Nowhere 9b	KKRT	IT-Kulturkontekstsreflektionsteori	TBA
Anna	1234	Somewhere 3	BPAK	Projektarbejde og Kommunikation	Mathias

row: known as a ***record***

column:
has a ***name*** (here: '`stud_id`')
and a ***type*** (here: '`int`')

(svarer til "`int stud_id;`" i Java)

a table cell is called a ***field***
(and it contains the actual data)

- ...and ***relations*** between entities/tables:

Relations (between Entities)

- **One-to-one** relations:

CUSTOMERS:

customer	creditcard_id
Anna	1234567890
Bent	2345678901
Carl	3456789012

one-to-one

CREDIT_CARDS:

creditcard_id	type	expires
1234567890	VISA	07/17
2345678901	Master Card	09/16
3456789012	VISA	11/17

- **One-to-many** relations:

ZIPCODES:

zip code	area name
0999	Copenhagen C
2300	Copenhagen S
1257	Copenhagen K

one-to-many

STREETS:

street_name	zip code	speed limit
Amagerbrogade	2300	50
Amalienborg Slotsplads	1257	40
Emil Holms Kanal	0999	20
Rued Langgards Vej	2300	50

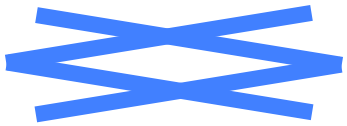
"A street has one zipcode", whereas
"A zipcode has many streets"

Problem with Many-to-Many

- **Many-to-many** relations:

STUDENTS:

student	student_id
Anna	1234
Bent	0001
Carl	0002



many-to-many

COURSES:

course_id	course_name
GRPRO	Grundlæggende Programmering
KKRT	IT-Kulturkontekstsreflektionsteori
BPAK	Projektarbejde og Kommunikation

- **Note**: ***many-to-many*** relations ***cannot*** be captured in/between tables!
  - **Solution:** make a new table which will ***be*** the many-to-many relation; **(i.e., how students↔courses)**:

ENROLLMENT:

student_id	course_id
1234	GRPRO
1234	KKRT
0001	GRPRO
0001	BPAK
0002	GRPRO

Solution: New Table for M-to-M

- **SOLUTION:** create a ***new table*** which then ***becomes the many-to-many relation:***

STUDENTS:

student	s_id
Anna	1234
Bent	0001
Carl	0002

1-to-many

ENROLLMENT:

s_id	c_id
1234	GRPRO
1234	KKRT
0001	GRPRO
0001	BPAK
0002	GRPRO

many-to-1

COURSES:

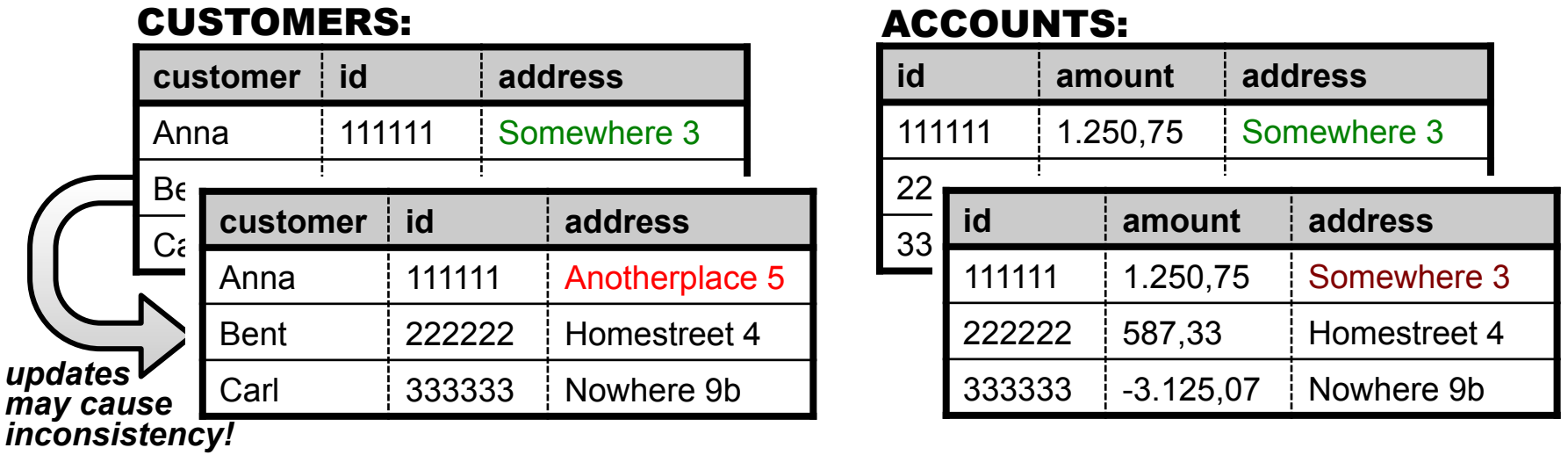
c_id	course_name
GRPRO	Grundlæggende Programmering
KKRT	IT-Kulturkontekstsreflektionsteori
BPAK	Projektarbejde og Kommunikation

- The "enrollment" table now ***is*** the ***many-to-many relation*** (how: **students↔courses**)!

"A student has many courses"
"A course has many students"

Redudance and Normalization

- Data redundancy is dangerous!:
  - Updates might lead to *data inconsistency!:*



- We ***normalize*** a DB to avoid data redundancy
  - A good DB has little or no data redundancy!

Normalization Example (I/III)

- Let's make a database containing info about students and the courses they are attending:

- **Data**:
  - student name
  - student id (unique for each student)
  - home address
  - course id (unique for each course)
  - course name
  - course teacher

Normalization Example (II/III)

- We might be tempted to make ***one big table:***

stud_name	stud_id	address	course_id	course_name	teacher
Anna	1234	Somewhere 3	GRPRO	Grundlæggende Programmering	Claus
Bent	0001	Homestreet 4	GRPRO	Grundlæggende Programmering	Claus
Carl	0002	Nowhere 9b	BPAK	Projektarbejde og Kommunikation	Mathias
Anna	1234	Somewhere 3	KKRT	IT-Kulturkontekstsreflektionsteori	TBA

  - However, such a table would have ***lots of redundancy!***

Normalization Example (II/III)

- We might be tempted to make ***one big table:***

stud_name	stud_id	address	course_id	course_name	teacher
Anna	**1234**	**Somewhere 3**	**GRPRO**	**Grundlæggende Programmering**	**Claus**
Bent	0001	Homestreet 4	**GRPRO**	**Grundlæggende Programmering**	**Claus**
Carl	0002	Nowhere 9b	BPAK	Projektarbejde og Kommunikation	Mathias
Anna	**1234**	**Somewhere 3**	KKRT	IT-Kulturkontekstsreflektionsteori	TBA

  - However, such a table would have ***lots of redundancy!***

- We can eliminate redundancy by breaking it into independent tables and relations:

STUDENTS:

stud_name	stud_id	address
Anna	1234	Somewhere 3
Bent	0001	Homestreet 4
Carl	0002	Nowhere 9b

COURSES:

course_id	course_name	teacher
GRPRO	Grundlæggende Programmering	Claus
KKRT	IT-Kulturkontekstsreflektionsteori	TBA
BPAK	Projektarbejde og Kommunikation	Mathias

Normalization Example (III/III)

STUDENTS:

stud_name	stud_id	address
Anna	1234	Somewhere 3
Bent	0001	Homestreet 4
Carl	0002	Nowhere 9b

COURSES:

course_id	course_name	teacher
GRPRO	Grundlæggende Programmering	Claus
KKRT	IT-Kulturkontekstsreflektionsteori	TBA
BPAK	Projektarbejde og Kommunikation	Mathias

- Now, all we need is ***m-to-m relation*** (table) that says ***"how students↔courses are related":***

ENROLLMENT:

student_id	course_id
1234	GRPRO
0001	GRPRO
0002	BPAK
1234	KKRT

BENEFITS:

- **reduced redundancy!** :-)
- courses can exist without students :-)
- students can exist without courses :-)

Note: we could have further reduced redundancy in our example (e.g., teacher could get own table)

Exercise: Eliminate Redundancy !

- **Exercise:**
  Eliminate redundancy in the following table:

pizza	name	description	ordered_by	creditcard	date
1	Margherita	Tomato and Cheese	Marco	724124825	23/10/2014
15	Carnivore	Meatballs, Beef, Chicken, and Bacon	Stefano	123456789	23/10/2014
22	Vegetariana	Rucola, Cucumber, and Brussels sprouts	Filippo	329942217	22/10/2014
1	Margherita	Tomato and Cheese	Marco	724124825	22/10/2014
15	Carnivore	Meatballs, Beef, Chicken, and Bacon	Filippo	329942217	21/10/2014
1	Margherita	Tomato and Cheese	Marco	724124825	24/12/2013

- **...by *"normalization"*** (i.e., break it into):
  - **independent tables;** *and*
  - **relations between them**

AGENDA

- **Introduction to Databases:**

TEORI

- 1) Create a database: 'CREATE TABLE'
- 2) Insert data into database: 'INSERT INTO'
- 3) Query a database: 'SELECT FROM'

- **How to create + connect to a MySQL database**

- **Introduction to Databases:**

DEMO

- 1) Create a database: 'CREATE TABLE'
- 2) Insert data into database: 'INSERT INTO'
- 3) Query a database: 'SELECT FROM'

SQL: Main Commands

- **Data definition:**
  - CREATE TABLE // creates a new table
  - DROP TABLE // deletes a table
- **Data manipulation:**
  - SELECT .. FROM .. WHERE // retrieves information
  - INSERT INTO .. VALUES (..) // insert record(s)
  - DELETE FROM // delete record(s)
  - UPDATE // changes record(s)

Example:

```
SELECT first_name FROM users WHERE last_name = 'Hansen';
```

- **SQL** commands are ***case in-sensitive*** (but it's common to use upper-case)
- Names of tables and records are ***case sensitive*** (let's use lower-case)

Table Creation (by Example)

- **Example:** 'mailing_list'

```
CREATE TABLE mailing_list (
    name VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL
)
```

mailing_list:

name	email
Claus	brabrand@itu.dk
Barack	obama@hotmail.com
John	jdoe@notmail.com

name of field (here: 'email')

type of field 'email' (here up to max 100 chars of variable length)

field in record cannot be 'null' (i.e., must have a value)

'null' is a *special* SQL value: ≠ **0** (zero) **or** "" (empty string)

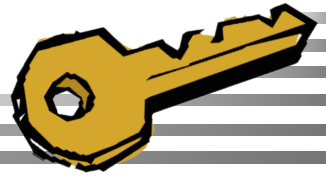
- **Another example:** 'phone_numbers'

```
CREATE TABLE phone_numbers (
    email VARCHAR(100) NOT NULL,
    phone VARCHAR(20) NOT NULL
)
```

phone_numbers:

email	phone
brabrand@itu.dk	7218 5076
obama@hotmail.com	212-555-0000
jdoe@notmail.com	123-456-7890

Primary Keys



- Primary key is a field which ***uniquely*** identifies a record (table row)
  - 'email' is a good candidate here:

```
CREATE TABLE mailing_list (
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) PRIMARY KEY
)
```

mailing_list:

name	email
Claus	brabrand@itu.dk
Barack	obama@hotmail.com
John	jdoe@notmail.com
John	john1928@yahoo.com

(Note: a primary key field automatically implies that it cannot be null)

- ***Primary keys*** are declared at table creation
- **Note**: without ***primary keys***, records cannot be uniquely identified (so we need them!)

Special: Auto Increment

- MySQL special function: AUTO_INCREMENT
  - ...that we can use to ***auto-generate*** unique values:

```
CREATE TABLE students (
    stud_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(50) NOT NULL,
    age INT NOT NULL,
)
```

students:

stud_id	name	age
1	Anna	20
2	Brian	99
3	Claire	87

- This simplifies insertion of new records:
  - every time a new record is inserted it automatically gets a fresh number (1, 2, 3, ...) so we don't need to bother

Question: Primary Keys?

STUDENTS:

stud_name	stud_id	address
Anna	1234	Somewhere 3
Brian	0001	Homestreet 4
Claire	0002	Nowhere 9b

COURSES:

course_id	course_name	teacher
DSDS	Introduction to Scripting...	Claus
GSD	Global Software Development	Claus
DC	Digital Culture	Somebody

ENROLLMENT:

student_id	course_id
1234	DSDS
1234	GSD
0001	DSDS
0001	DC
0002	DSDS

Common SQL Types

- Common SQL Types:

Type	Description	Example value
INT	Integer (number)	42
DOUBLE	Floating-point (decimal) number	3.1415
VARCHAR(80)	Text up to 80 characters	"John Doe"
VARCHAR	Text up to 255 characters	"John Doe"
TEXT	Long text (max 65535 characters)	"Once upon a time, ..."
DATE	Date	2014-10-23
TIME	Time	12:59:59
DATETIME	Date & Time	2014-10-23 12:59:59

- ...and many more (see MySQL specification)

Inserting Records (by Example)

- Insert records:

```
INSERT INTO mailing_list (name, email)
  VALUES ('Claus', 'brabrand@itu.dk') ;

INSERT INTO mailing_list (name, email)
  VALUES ('Barack', 'obama@hotmail.com') ;

INSERT INTO mailing_list (name, email)
  VALUES ('John', 'johndoe@notmail.com') ;
```

mailing_list:

name	email
Claus	brabrand@itu.dk
Barack	obama@hotmail.com
John	jdoe@notmail.com

```
INSERT INTO phone_numbers (email, phone)
  VALUES ('brabrand@itu.dk', '72185076') ;

INSERT INTO phone_numbers (email, phone)
  VALUES ('obama@hotmail.com', '212-555-0000')

INSERT INTO phone_numbers (email, phone)
  VALUES ('jdoe@notmail.com', '123-456-7890') ;
```

phone_numbers:

email	phone
brabrand@itu.dk	7218 5076
obama@hotmail.com	212-555-0000
jdoe@notmail.com	123-456-7890

Info Retrieval (I)

students:

name	id	address
Anna	1234	Somewhere 3
Brian	0001	Homestreet 4
Claire	0002	Nowhere 9b

- Information retrieval:

```
SELECT * FROM students ;
```

name	id	address
Anna	1234	Somewhere 3
Brian	0001	Homestreet 4
Claire	0002	Nowhere 9b

```
SELECT name, address FROM students ;
```

name	address
Anna	Somewhere 3
Brian	Homestreet 4
Claire	Nowhere 9b

Info Retrieval (II)

students:

name	id	address
Anna	1234	Somewhere 3
Brian	0001	Homestreet 4
Claire	0002	Nowhere 9b

- Information retrieval:

```
SELECT name FROM students WHERE id > 1;
```

name
Anna
Claire

```
SELECT id, address FROM students WHERE name = 'Brian' ;
```

id	address
0001	Homestreet 4

```
SELECT * FROM students WHERE id = '1234' ;
```

name	id	address
Anna	1234	Somewhere 3

Info Retrieval (III)

students:

name	id	address
Anna	1234	Somewhere 3
Brian	0001	Homestreet 4
Claire	0002	Nowhere 9b

- Information retrieval:

```
SELECT * FROM students ORDER BY id ;
```

name	id	address
Brian	0001	Homestreet 4
Claire	0002	Nowhere 9b
Anna	1234	Somewhere 3

```
SELECT * FROM students ORDER BY name DESC ;
```

desc (aka descending) => backwards

name	id	address
Claire	0002	Nowhere 9b
Brian	0001	Homestreet 4
Anna	1234	Somewhere 3

Deleting and Updating records

- Deleting records:

```
DELETE FROM mailing_list WHERE name='John' ;
```

mailing_list:

name	email
Claus	brabrand@itu.dk
Barack	obama@hotmail.com
John	jdoe@notmail.com

mailing_list:

name	email
Claus	brabrand@itu.dk
Barack	obama@hotmail.com

- Updating records:

```
UPDATE mailing_list SET email='barack@gmail.com' WHERE name='Barack' ;
```

mailing_list:

name	email
Claus	brabrand@itu.dk
Barack	obama@hotmail.com

mailing_list:

name	email
Claus	brabrand@itu.dk
Barack	barack@gmail.com

Dropping Tables

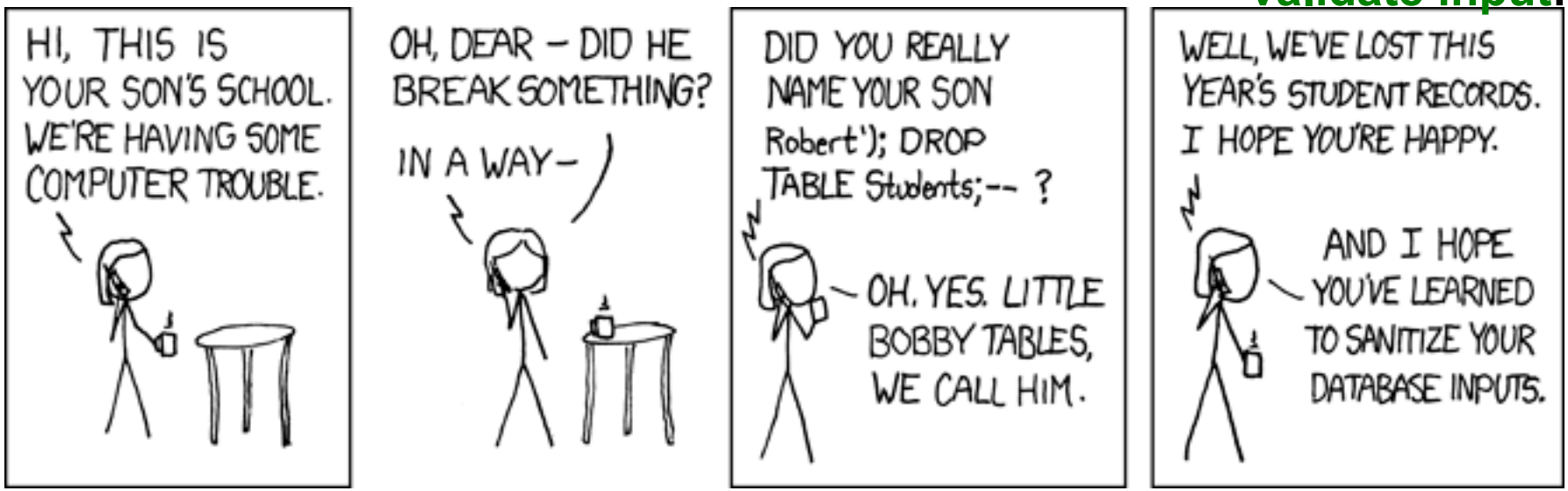
- Dropping tables:

```
DROP TABLE mailing_list ;
```

mailing_list:

name	email
Claus	brabrand@itu.dk
Barack	obama@notmail.com
John	jdoe@notmail.com

Remember to **validate input**!



SQL Injection Attacks :-)

AGENDA

- **Introduction to Databases:**

TEORI

- 1) Create a database: 'CREATE TABLE'
- 2) Insert data into database: 'INSERT INTO'
- 3) Query a database: 'SELECT FROM'

- **How to create + connect to a MySQL database**

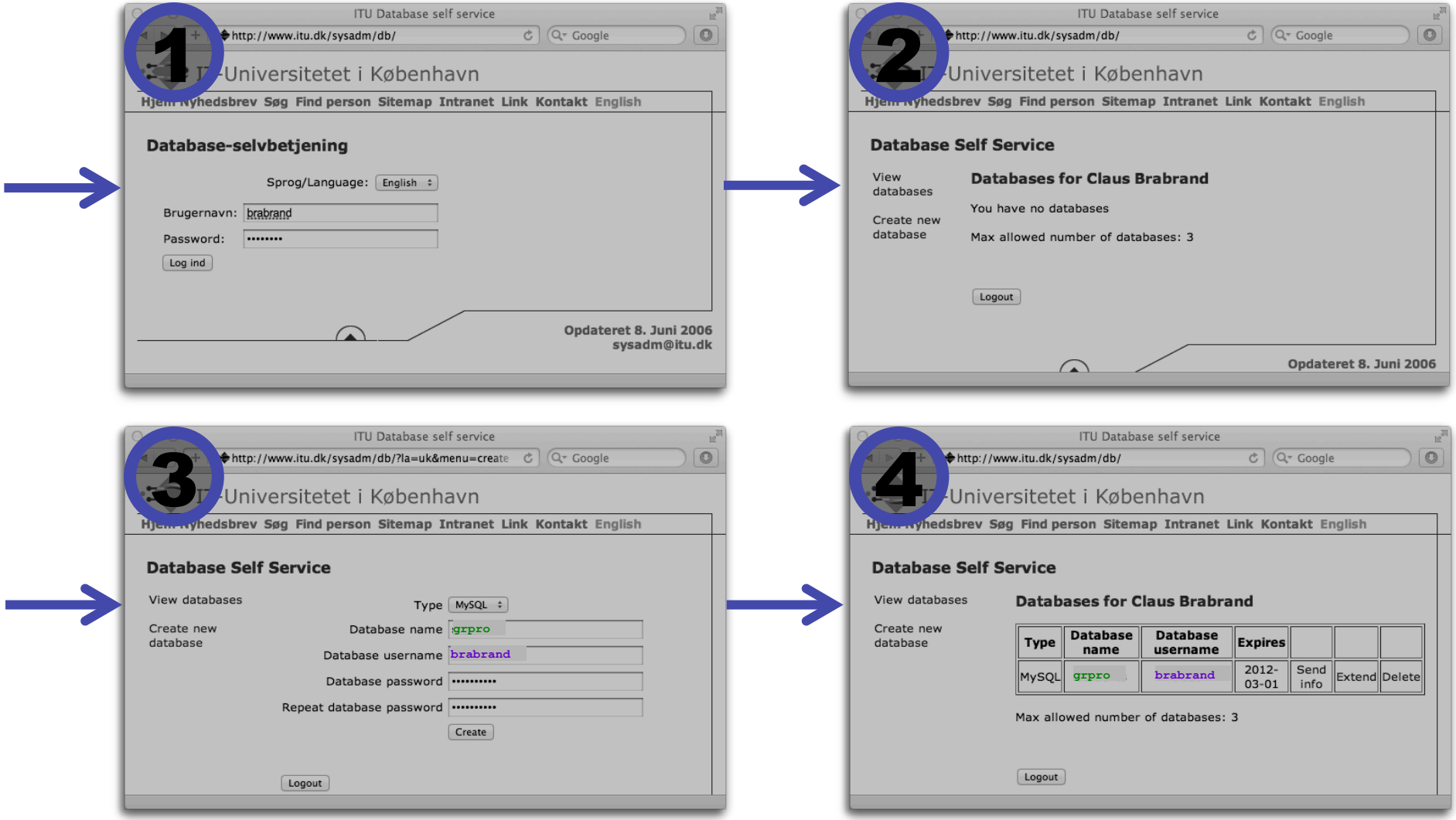
- **Introduction to Databases:**

DEMO

- 1) Create a database: 'CREATE TABLE'
- 2) Insert data into database: 'INSERT INTO'
- 3) Query a database: 'SELECT FROM'

(http://www.itu.dk/sysadm/db/)

SQL Setup



SQL Log in

- Start a "`terminal`" and write the following:

```
Last login: Fri Nov  4 11:18:54 on ttys000
adm2-66:~ brabrand$ ssh brabrand@ssh.itu.dk
brabrand@ssh.itu.dk's password:
Last login: Fri Nov  4 11:19:10 2011 from 130.226.142.243

[brabrand@cypher ~]$ mysql -h mysql.itu.dk -u brabrand -p grpro
Enter password:
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9696431
Server version: 5.0.67 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

you write the blue stuff and the computer writes the gray stuff

AGENDA

- **Introduction to Databases:**

TEORI

- 1) Create a database: 'CREATE TABLE'
- 2) Insert data into database: 'INSERT INTO'
- 3) Query a database: 'SELECT FROM'

- **How to create + connect to a MySQL database**

- **Introduction to Databases:**

DEMO

- 1) Create a database: 'CREATE TABLE'
- 2) Insert data into database: 'INSERT INTO'
- 3) Query a database: 'SELECT FROM'

SQL Demo: Create & Insert

```
mysql> CREATE TABLE mailing_list ( name VARCHAR(100) NOT NULL,
->                                 email VARCHAR(100) NOT NULL ) ;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO mailing_list (name, email)
->     VALUES ('Claus', 'brabrand@itu.dk') ;
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO mailing_list (name, email)
->     VALUES ('Barack', 'obama@hotmail.com') ;
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO mailing_list (name, email)
->     VALUES ('John', 'jdoe@notmail.com') ;
Query OK, 1 row affected (0.00 sec)

mysql>
```

SQL Demo: Select

```
mysql> SELECT * FROM mailing_list ;
+--------+--------------------+
| name   | email              |
+--------+--------------------+
| Claus  | brabrand@itu.dk    |
| Barack | obama@hotmail.com  |
| John   | jdoe@notmail.com   |
+--------+--------------------+
3 rows in set (0.00 sec)

mysql> SELECT name FROM mailing_list ;
+--------+
| name   |
+--------+
| Claus  |
| Barack |
| John   |
+--------+
3 rows in set (0.00 sec)
```

SQL Demo: Select (cont'd)

```
mysql> SELECT email FROM mailing_list WHERE name='Barack';
+--------------------+
| email              |
+--------------------+
| obama@hotmail.com  |
+--------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM mailing_list ORDER BY name DESC ;
+--------+--------------------+
| name   | email              |
+--------+--------------------+
| John   | jdoe@notmail.com   |
| Claus  | brabrand@itu.dk    |
| Barack | obama@hotmail.com  |
+--------+--------------------+
3 rows in set (0.00 sec)

mysql>
```

SQL Demo: Delete

```
mysql> DELETE FROM mailing_list WHERE name='John';
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM mailing_list ;
+--------+--------------------+
| name   | email              |
+--------+--------------------+
| Claus  | brabrand@itu.dk    |
| Barack | obama@hotmail.com  |
+--------+--------------------+
2 rows in set (0.00 sec)

mysql>
```

SQL Demo: Update

```
mysql> UPDATE mailing_list SET email='barack@gmail.com'
->                             WHERE name='Barack' ;
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM mailing_list ;
+--------+-------------------+
| name   | email             |
+--------+-------------------+
| Claus  | brabrand@itu.dk   |
| Barack | barack@gmail.com  |
+--------+-------------------+
2 rows in set (0.00 sec)

mysql>
```

SQL Demo: Show Columns

```
mysql> SHOW COLUMNS FROM mailing_list ;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| name  | varchar(100) | NO   |     | NULL    |       |
| email | varchar(100) | NO   |     | NULL    |       |
+-------+--------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
Query OK, 1 row affected (0.00 sec)

mysql>
```

AGENDA

- **Bonus: More SQL commands**

SHOW TABLES & DESCRIBE

- Show tables in database:

```
SHOW TABLES ;
```

Tables_in_my_database
mailing_list
phone_numbers
students

- Show information about a particular table:

```
DESCRIBE students ;
```

Field	Type	Null	Key	Default	Extra
name	varchar(80)	NO		NULL	
age	int(11)	NO		NULL	

- Quit:

```
QUIT ;
```

LIMIT and ORDER-BY

- We can limit the number of records displayed:

- Retrieve a hundred recipes:

```
SELECT * FROM recipes LIMIT 100 ;
```

- Retrieve the youngest student:

```
SELECT * FROM students ORDER BY age LIMIT 1 ;
```

- Retrieve three (alphabetically) last courses:

```
SELECT * FROM courses ORDER BY name DESC LIMIT 3 ;
```

COUNT

- Given the table:

```
SELECT * FROM students ;
```

stud_id	name	age
1234	Anna	20
5678	Brian	25
9999	Claire	23

- **COUNT** (will count number of records):

```
SELECT COUNT(*) FROM students ;
```

COUNT(*)
3

- **COUNT AS** (gives the count a name):

```
SELECT COUNT(*) AS num_courses FROM students ;
```

num_courses
3

MIN, MAX, SUM, AVG

stud_id	name	age
1234	Anna	20
5678	Brian	25
9999	Claire	23

- **MIN** (minimum):

```
SELECT MIN(age) FROM students ;
```

MIN(age)
20

- **MAX** (maximum):

```
SELECT MAX(age) AS oldest FROM students ;
```

oldest
25

- **SUM**:

```
SELECT SUM(age) FROM students ;
```

SUM(age)
68

- **AVG** (average):

```
SELECT AVG(age) AS average FROM students ;
```

average
22.6667

Thx

- Questions? -

Next time: "JOIN" & "GROUP BY"

Exercises 9

- **Exercise 9.1**:
  - Create own MySQL database
    (hosted on the ITU MySQL server)
  - Log into it via the MySQL text client
- **Exercise 9.2**:
  - Create SQL tables for ITU Courses and Teachers
  - ...and populate your database with real'ish data
- **Exercise 9.3**:
  - Try out the various queries from these slides