introduction to

SCRIPTING, DATABASES, SYSTEM ARCHITECTURE

SQL I: Relational Databases & Intro to SQL

Claus Brabrand

(((brabrand@itu.dk)))

Associate Professor, Ph.D.

(((Software and Systems)))

IT University of Copenhagen

Agenda

- **PHP Recap** (list of constructs)
- **Introduction to Databases** **[in Theory]:**

DEMO

  - 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** **[in Practice]:**

TRY !

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

PHP Index

Constructions, Types, Operators, Control Structures, Functions, HTML input fields, Regular Expressions, ...

HTML Form Input Fields

Simple input fields:

- <input type='text' />
- <input type='password' />
- <input type='hidden' />
- <input type='radio' />
- <input type='checkbox' />
- <input type='submit' />
- <input type='reset' />

Composite input fields:

- <textarea>...</textarea>
- <select>...</select>
- <select multiple='multiple'>
  ...
  </select>

Form:

- <form method='get/post'
  action='script-URL'>
  ...
  </form>

PHP Index

Meta Constructs:

- <?php ... ?> (php tag)
- <!-- html comment -->
- // php one line comment
- # php one line comment
- /* php multi-line comment */
- include('my_script.php')
- var_dump($x)

Variable Operations:

- $x (read value of variable)
- $x = 3 (write / assignment)
- $x[3] (read from array)
- $x[3] = ... (write to array)
- $x['bla'] (assoc.array read)
- $x['bla'] = ... (ass.ary. write)
- $x++ (increase $x by 1)
- $x-- (decrease $x by 1)

PHP Index (II)

Read Input from Fields:

- $_REQUEST['x'] (read input)
- $_GET['x'] (read get input)
- $_POST['x'] (read post input)

Control Structures:

- if (conditional statement)
- if-else (cond. statement)
- while (loop / iteration)
- for (loop / iteration)
- foreach (loop / iteration)

Special Control Struct's:

- return (used in functions)

Types:

- 87 (integer)
- 3.14 (floating point)
- 'bla' (string, single quote)
- "bla" (string, double quote)
- true (boolean)
- array (sequence of values)

PHP Index (III)

Arithmetic operators:

- + (addition)
- - (subtraction)
- * (multiplication)
- / (division)

Logical operators:

- ! (not / negation)
- && (and / conjunction)
- || (or / disjunction)

Comparative operators:

- == (equality / equal to)
- != (inequality / not equal to)
- < (less than)
- <= (less than or equal to)
- > (greater than)
- >= (greater or equal to)

String Operators:

- . (string concatenation)

PHP Index (IV)

Functions declaration:

- function f($x, $y) { ... }

Functions:

- echo ...
- isset(...)
- array(...)
- sizeof(...)
- sort(...)
- preg_match('/^...$/')

Function call/invocation:

- f(42, 'blah');

Functions:

- date(...)
- round(...)
- strlen(...)
- strtoupper(...)
- mail(...)
- number_format(...)

Regular Expressions (regexp)

```
R   :   x              // matches text 'x' itself
    :   \c             // escaping (matches char 'c')
    :   '.' (dot)      // matches one character
    :   [aeiou]        // character class: one of chars
    :   [a-z]          // any character in interval
    :   [^...]         // negated char class
    :   R1 R2          // sequence (matches R1 then R2)
    :   R1|R2          // alternative (either R1 or R2)
    :   R+             // one-or-more repetitions of R
    :   R*             // zero-or-more repetitions of R
    :   R?             // optional R (empty text or R)
    :   R{n}           // n times repetitions of R
    :   R{n,}          // at least n repetitions of R
    :   R{n,m}         // n to m repetitions of R
    :  ( R )           // parentheses for grouping
```

Agenda

- **PHP Recap** (list of constructs)
- **Introduction to Databases** **[in Theory]:**

DEMO

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

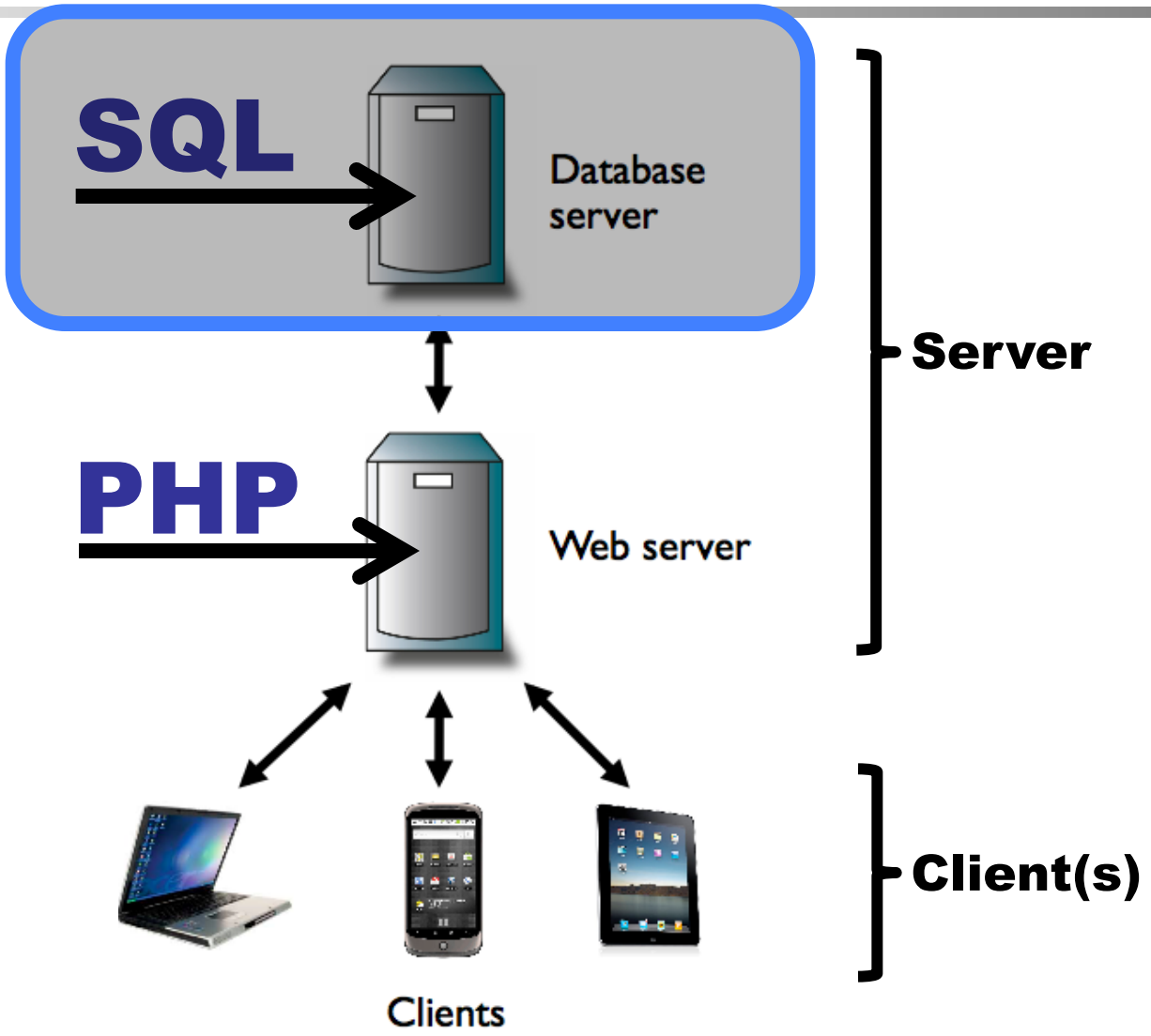
- **How to create + connect to a MySQL database**
- **Introduction to Databases** **[in Practice]:**

TRY !

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

Web Service Architecture

SQL clients:
- text client
- web client



SQL

- 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 DBs
  - (other kinds of DBs: 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 in the software industry!

Web Service Design Process

4-Step design process...:

1) design data model:

- what info should be stored?
- how should it be represented?

Will use:

- ER (Entity-Relationship) diagrams
- Normalization

2) develop data transactions:

- how is data ***read*** (from the DB)?
- how is data ***written*** (to the DB)?
- how is data ***modified*** (in the DB)?

Q: *"does our database allow us to retrieve (get) the info we want?"*

3) develop site map and forms:

- use HTML (and CSS) for the user interface (UI)

4) program it all in SQL + PHP:

- **SQL:** perform DB transactions
- **PHP:** "glue" UI + DB together

Structure and Terminology

A relational database is built on...:

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

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

stud_name	stud_id	address	course_id	course_name	teacher
Anna	1234	Somewhere 3	DSDS	Introduction to Script...	Claus
Brian	0001	Homestreet 4	DSDS	Introduction to Script...	Claus
Claire	0002	Nowhere 9b			
Anna	1234	Somewhere 3	DC	Digital Culture	Somebody

a ***row*** (aka. ***record***) contains the actual data

a ***column*** has a name (e.g. 'stud_id') and a "type" (e.g. 'integer')

a table cell is called a ***field***

- ...and ***relations*** between tables

Table Relationships

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

CARD_OWNERS:

customer	creditcard_id
Anna	1234567890
Anna	2345678901
Brian	3456789012

one-to-one

CARD_INFO:

creditcard_id	type
1234567890	VISA
2345678901	Master Card
3456789012	VISA

- **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

Table Relationships (cont'd)

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

STUDENTS:

student	student_id
Anna	1234
Brian	0001
Claire	0002

COURSES:

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

- **Note**: **many-to-many** relations cannot be captured directly in tables!
  - Thus, we have to make a table for a many-to-many relation:

    (A student has many courses)

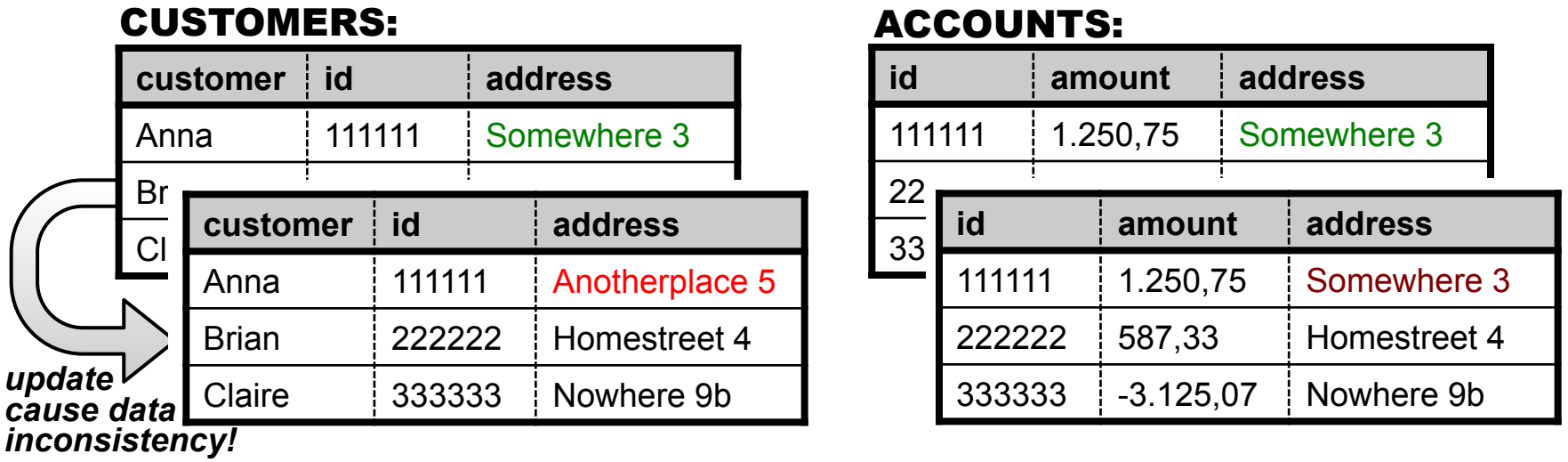
    (A course has many students)

ENROLLMENT:

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

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**	**DSDS**	**Introduction to Script..**	**Claus**
Brian	0001	Homestreet 4	**DSDS**	**Introduction to Script..**	**Claus**
Claire	0002	Nowhere 9b			
Anna	**1234**	**Somewhere 3**	DC	Digital Culture	Somebody

  - However, it has ***lots*** of redundancy!

- We can eliminate redundancy by breaking it into independent constituents:

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

Normalization Example (III/III)

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

- Now all we need is another table that says "which students and courses are related":

ENROLLMENT:

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

Benefits:
- reduced redundancy!
- courses can exist without students
- students can exist without courses

Note: we could further reduce the redundancy in the example (e.g., teacher could get its own table)

Agenda

- **PHP Recap** (list of constructs)
- **Introduction to Databases** **[in Theory]:**
  - **1)** Create a database: 'CREATE TABLE'
  - **2)** Insert data into database: 'INSERT INTO'
  - **3)** Query a database: 'SELECT FROM'

DEMO

- **How to create + connect to a MySQL database**
- **Introduction to Databases** **[in Practice]:**
  - **1)** Create a database: 'CREATE TABLE'
  - **2)** Insert data into database: 'INSERT INTO'
  - **3)** Query a database: 'SELECT FROM'

TRY !

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)

- SQL commands are ***case insensitive*** (but let's 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)

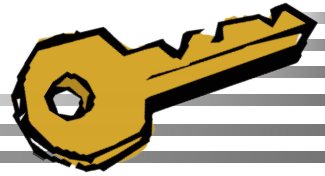
- **Another example:** 'phone_numbers'

```
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

Students-Courses Example

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

Q: Primary keys?

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	2011-11-04
TIME	Time	08:29:59
DATETIME	Date & Time	2011-11-04 08:29:59

- ...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

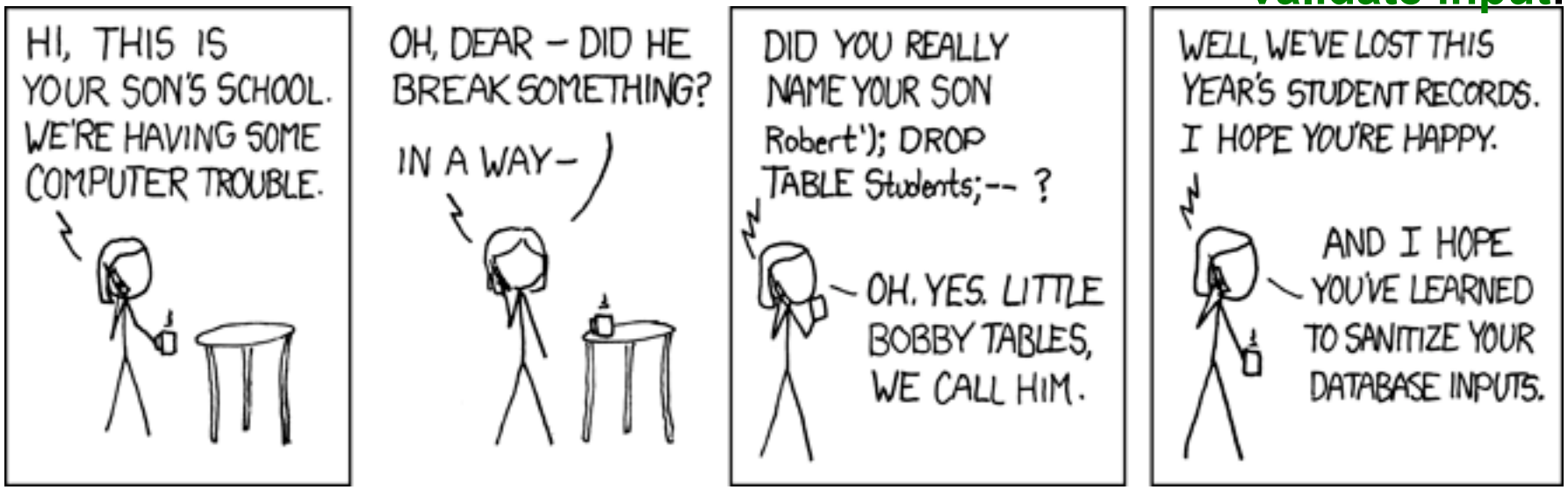
- Dropping tables:

```
DROP TABLE mailing_list ;
```

mailing_list:

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

Remember to **validate input**!



Agenda

- **PHP Recap** (list of constructs)
- **Introduction to Databases** **[in Theory]:**

DEMO

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

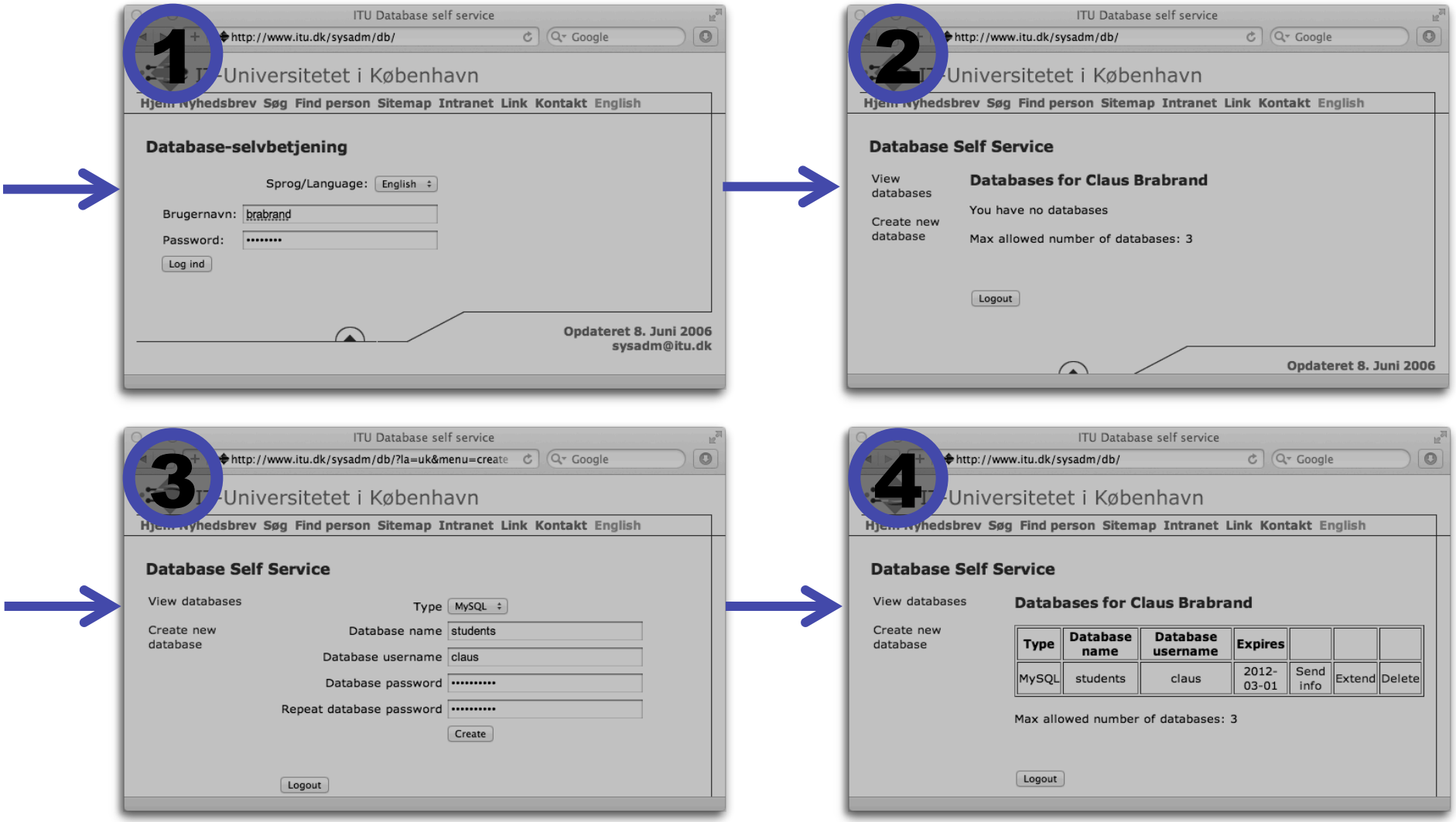
- **How to create + connect to a MySQL database**
- **Introduction to Databases** **[in Practice]:**

TRY !

  - 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 claus -p students
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

- **PHP Recap** (list of constructs)
- **Introduction to Databases** **[in Theory]:**

DEMO

  - 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** **[in Practice]:**

TRY !

  - 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>
```

Assignment A6

- **Exercise 6.1:**
  - Create own MySQL database
    (hosted on the ITU MySQL server)
  - Log into it via the MySQL text client
- **Exercise 6.2:**
  - Answer some basic SQL questions from the lecture
- **Exercise 6.3:**
  - Create SQL tables for ITU Courses and Teachers
- **Exercise 6.4:**
  - Add students to your database