GRUNDLÆGGENDE PROGRAMMERING

SQL II: Joins, Keys, Group-By, & Java+SQL !

Claus Brabrand

(((brabrand@itu.dk)))

Associate Professor, Ph.D.

(((Software and Systems)))

IT University of Copenhagen

A G E N D A

- **Recap**
- **Join** (samkøring af data)
- **Indices**
- **Left and Right Join**
- **Group by**
- **Java + SQL !**

SQL: Main Commands

- **Data definition:**
  - CREATE TABLE // creates a new table
  - DROP TABLE // deletes a table
  - SHOW TABLES // see tables defined
  - DESCRIBE // info about a table

- **Data manipulation:**
  - INSERT INTO .. VALUES (..) // insert record(s)
  - **SELECT .. FROM .. WHERE** **// retrieves info !!!**
  - DELETE FROM .. WHERE // delete record(s)
  - UPDATE .. SET .. WHERE // changes record(s)

Information Retrieval

students:

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

- **SELECT-FROM:**

```
SELECT * FROM students ;
```

→

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

- **SELECT-FROM-WHERE:**

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

→

name	address
Anna	Somewhere 3
Claire	Nowhere 9b

A G E N D A

- Recap
- Join (samkøring af data)
- Indices
- Left and Right Join
- Group by
- Java + SQL !

Lov om offentlige myndigheders registre

DATATILSYNET

Lov nr. 294 af 8. juni 1978:

- § 4. Registre, der føres for en statslig myndighed, ***må kun oprettes i henhold til godkendelse*** af vedkommende minister efter forhandling med finansministeren. [...]
- *Stk 3.* Bestemmelserne [...] ***gælder tilsvarende ved samkøring af registre***, der er oparbejdet med henblik på varetagelse af forskellige opgaver. (Dette gælder dog ikke samkøring, der udelukkende foretages med henblik på uddrag i statistisk eller videnskabeligt øjemed.)

Join (1/3)

mailing_list:

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

phone_numbers:

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

```
SELECT *
FROM mailing_list , phone_numbers ;
```

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

"join" operation:

gives all combos!

Join (2/3)

mailing_list:

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

phone_numbers:

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

```
SELECT *
FROM mailing_list , phone_numbers
WHERE mailing_list.email = phone_numbers.email ;
```

	name	email	email	phone
✓	Claus	brabrand@itu.dk	brabrand@itu.dk	7218 5076
✗	~~Barack~~	~~obama@hotmail.com~~	~~brabrand@itu.dk~~	~~7218 5076~~
✗	~~John~~	~~jdoe@notmail.com~~	~~brabrand@itu.dk~~	~~7218 5076~~
✗	~~Claus~~	~~brabrand@itu.dk~~	~~obama@hotmail.com~~	~~212-555-0000~~
✓	Barack	obama@hotmail.com	obama@hotmail.com	212-555-0000
✗	~~John~~	~~jdoe@notmail.com~~	~~obama@hotmail.com~~	~~212-555-0000~~
✗	~~Claus~~	~~brabrand@itu.dk~~	~~jdoe@notmail.com~~	~~123-456-7890~~
✗	~~Barack~~	~~obama@hotmail.com~~	[illegible]	[illegible]
✓	John	jdoe@notmail.com		

"join" operation:

gives all combos!

now, only the rows we want :-)

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

Join (3/3)

mailing_list:

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

phone_numbers:

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

```
SELECT name, mailing_list.email, phone
FROM mailing_list , phone_numbers
WHERE mailing_list.email = phone_numbers.email ;
```

	name ✓	email ✓	email ✗	phone ✓
✓	Claus	brabrand@itu.dk	brabrand@itu.dk	7218 5076
✗	~~Barack~~	~~obama@hotmail.com~~	~~brabrand@itu.dk~~	~~7218 5076~~
✗	~~John~~	~~jdoe@notmail.com~~	~~brabrand@itu.dk~~	~~7218 5076~~
✗	~~Claus~~	~~brabrand@itu.dk~~	~~obama@hotmail.com~~	~~212-555-0000~~
✓	Barack	obama@hotmail.com	obama@hotmail.com	212-555-0000
✗	~~John~~	~~jdoe@notmail.com~~	~~obama@hotmail.com~~	~~212-555-0000~~
✗	~~Claus~~	~~brabrand@itu.dk~~	~~jdoe@notmail.com~~	~~123-456-7890~~
✗	~~Barack~~	~~obama@hotmail.com~~	jdo	[illegible]
✓	John	jdoe@notmail.com	jdo	

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

"join" operation:

gives all combos!

now, only the rows we want :-)

and only the col's we want :-)

EXERCISE 1: Join

- Database:

crimescenes:

place	item	dna_found
Living Room	Table	ACCTC
Living Room	Lamp	AGTGC
Hallway	Chainsaw	ACCTC
Back Alley	Knife	ACTGC
Bathroom	Shampoo	ACCTC
Kitchen	Milk Carton	ACTTC

suspects:

name	dna_profile
Jack the Ripper	ACTGC
Jason	ACGTC
Amagermanden	ACTTC

+ 200.000 more people

- ***Join*** to determine (possibly multiple) perpetrators?

→

name	place	item
Jack the Ripper	Back Alley	Knife
Amagermanden	Kitchen	Milk Carton

EXERCISE 2: Join

customers:

cpr	name	phone	address
321085-1111	Anna	212-555-7755	Somewhere 4
221188-2222	Brian	212-555-5050	Anystreet 7
010190-3333	Claire	212-555-0707	Some Other Place 15
020290-4444	Danny	212-555-9999	Anywhere 1a

orders:

cpr	total
010190-3333	1,525.24
321085-1111	195.56
040587-5555	1,020.30
221188-2222	897.20

- Find names & phone #s of ***"good customers"***
  ("good customer" = spent more than 500 DKK):

→

name	phone	total
Brian	212-555-5050	897.20
Claire	212-555-0707	1,525.24

Three-way Join

- Three-way joins:

```
SELECT students.id, students.name
  FROM students , courses , enrollment
    WHERE students.id = enrollment.id
      AND enrollment.id = courses.id ;
```

- The above query can be ***abbeviated*** to (via "**AS**" aliases):

```
SELECT s.id, s.name
  FROM students AS s , courses AS c , enrollment AS e
    WHERE s.id = e.id
      AND e.id = c.id ;
```

A G E N D A

- **Recap**
- **Join** **(samkøring af data)**
- **Indices**
- **Left and Right Join**
- **Group by**
- **Java + SQL !**

How fast is a query?

- A ***join*** of two tables can take a very long time!
  - every record in one table needs to be compared to every record in the other table

```
SELECT * FROM table1 , table2 WHERE table.id1 = table.id2 ;
```

- If both tables have 10,000 records (not uncommon!), we get 10,000 x 10,000 = 100,000,000 comparisons
  - **This can easily take 20 seconds !**
- However, if `id1` and `id2` have been ***indexed***
  - **This only takes 0.05 seconds !**

  - (Note: the bigger the table, the more important with indexing)

Types of Indices (= Index'es)

- Three types of indices:

type	needs to be unique?	is allowed to be null?
PRIMARY KEY	*yes*	*no*
UNIQUE	*yes*	*yes*
INDEX	*no*	*yes*

  - (Note: only one PRIMARY KEY allowed per table)

- **Example:**

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

students:

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

When to use Indices

Indices are best used on columns that are...:

- ...frequently used in the **`WHERE`** part:

```
SELECT * FROM table WHERE age > 20 ;
```

- ...frequently used in the **`ORDER BY`** part:

```
SELECT * FROM table ORDER BY name ;
```

- ...used as part of joins:

```
SELECT * FROM table1 , table2 WHERE table.id1 = table.id2 ;
```

Foreign Keys

mailing_list:

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

phone_numbers:

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

- Recall:

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

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

Ensure "referential integrity" :
i.e., that `phone_number.email` is always in `mailing_list.email` !

- We want to **link**:
  - `mailing_list.email` ⟵ `phone_numbers.email`

- We can use a ***foreign key:***

```
CREATE TABLE mailing_list (
    name VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL
) ENGINE=InnoDB;
```

```
CREATE TABLE phone_numbers (
    email VARCHAR(100) NOT NULL,
    phone VARCHAR(20) NOT NULL,
    FOREIGN KEY(email) REFERENCES
        mailing_list(email)
) ENGINE=InnoDB;
```

NB: some (annoying) stuff we have to write
b/c we use Foreign Keys [long story...]

A G E N D A

- Recap
- Join (samkøring af data)
- Indices
- Left and Right Join
- Group by
- Java + SQL !

Recall "Join"

- Database:

suspects:

name	dna_profile
Jack the Ripper	ACTGC
Jason	ACGTC
Amagermanden	ACTTC

crimescenes:

place	item	dna_found
Living Room	Table	ACCTC
Living Room	Lamp	AGTGC
Hallway	Chainsaw	ACCTC
Back Alley	Knife	ACTGC
Bathroom	Shampoo	ACCTC
Kitchen	Milk Carton	ACTTC

- *Join* to determine (possibly multiple) perpetrators?

```
SELECT name, place, item
  FROM suspects , crimescenes
    WHERE dna_profile = dna_found ;
```

→

name	place	item
Jack the Ripper	Back Alley	Knife
Amagermanden	Kitchen	Milk Carton

One problem with joins

- Database:

suspects:

name	dna_profile
Jack the Ripper	ACTGC
Jason	ACGTC
Amagermanden	ACTTC

crimescenes:

place	item	dna_found
Living Room	Table	ACCTC
Living Room	Lamp	AGTGC
Hallway	Chainsaw	ACCTC
Back Alley	Knife	ACTGC
Bathroom	Shampoo	ACCTC
Kitchen	Milk Carton	ACTTC

- Suppose we instead would like to generate:

That is, always generate a record from the right table (not just when we happen to have a match)

```
SELECT place, item, name
FROM
  suspects RIGHT JOIN crimescenes
  ON dna_profile = dna_found ;
```

place	item	name (if match)
Living Room	Table	NULL
Living Room	Lamp	NULL
Hallway	Chainsaw	NULL
Back Alley	Knife	Jack the Ripper
Bathroom	Shampoo	NULL
Kitchen	Milk Carton	Amagermanden

LEFT JOIN vs. RIGHT JOIN

- Database:

crimescenes:

place	item	dna_found
Living Room	Table	ACCTC
Living Room	Lamp	AGTGC
Hallway	Chainsaw	ACCTC
Back Alley	Knife	ACTGC
Bathroom	Shampoo	ACCTC
Kitchen	Milk Carton	ACTTC

suspects:

name	dna_profile
Jack the Ripper	ACTGC
Jason	ACGTC
Amagermanden	ACTTC

- Same thing for LEFT JOIN (just swap arguments):

```
SELECT place, item, name FROM
  crimescenes LEFT JOIN suspects
  ON suspects.dna_profile =
     crimescene.dna_found ;
```

||

```
SELECT place, item, name FROM
  suspects RIGHT JOIN crimescenes
  ON suspects.dna_profile =
     crimescene.dna_found ;
```

→

place	item	name (if match)
Living Room	Table	NULL
Living Room	Lamp	NULL
Hallway	Chainsaw	NULL
Back Alley	Knife	Jack the Ripper
Bathroom	Shampoo	NULL
Kitchen	Milk Carton	Amagermanden

Join (normal, left, right)

a:

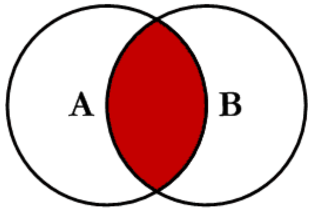
name	id
Anna	1
Brian	2
Claire	3

b:

id	course
1	GRPRO
2	KKRT
4	BPAK

- Normal join:

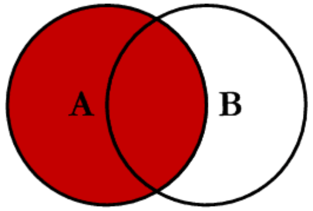
```
SELECT name, a.id, course
  FROM a , b WHERE a.id = b.id ;
```



name	id	course
Anna	1	GRPRO
Brian	2	KKRT

- Left join:

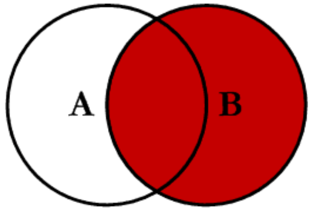
```
SELECT name, a.id, course
  FROM a LEFT JOIN b ON a.id = b.id ;
```



name	id	course (if match)
Anna	1	GRPRO
Brian	2	KKRT
Claire	3	NULL

- Right join:

```
SELECT name, b.id, course
  FROM a RIGHT JOIN b ON a.id = b.id ;
```



name (if match)	id	course
Anna	1	GRPRO
Brian	2	KKRT
NULL	4	BPAK

RIGHT JOIN

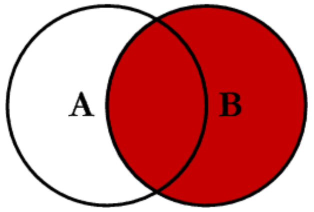
crimescenes:

place	item	dna_found
Living Room	Table	ACCTC
Living Room	Lamp	AGTGC
Hallway	Chainsaw	ACCTC
Bedroom	Knife	ACTGC
Bathroom	Shampoo	ACCTC
Kitchen	Milk Carton	ACTTC

suspects:

name	dna_profile
Jack the Ripper	ACTGC
Jason	ACGTC
Amagermanden	ACTTC

```
SELECT place, item, name
FROM suspects RIGHT JOIN crimescenes
     ON suspects.dna_profile = crimescene.dna_found ;
```



place	item	name (if match)
Living Room	Table	NULL
Living Room	Lamp	NULL
Hallway	Chainsaw	NULL
Bedroom	Knife	Jack the Ripper
Bathroom	Shampoo	NULL
Kitchen	Milk Carton	Amagermanden

We, of course, have that:

`x RIGHT JOIN y`

||

`y LEFT JOIN x`

A G E N D A

- Recap
- Join (samkøring af data)
- Indices
- Left and Right Join
- Group by
- Java + SQL !

GROUP BY

expenses:

expense	dept	year	amount
salary	research	2011	490000
salary	sales	2012	1500000
salary	research	2012	500000
coffee	research	2013	800
coffee	sales	2013	300
salary	sales	2013	1600000
salary	research	2013	510000

- Given table:
  - CREATE TABLE
  - INSERT INTO
- Let's calculate total expenses ***per dept.***:

```
SELECT dept, SUM(amount) FROM expenses GROUP BY dept ;
```

"GROUP BY" causes the records to be sorted and grouped wrt. their `dept` value:

expense	dept	year	amount
salary	research	2011	490000
salary	research	2012	500000
coffee	research	2013	800
salary	research	2013	510000
salary	sales	2012	1500000
coffee	sales	2013	300
salary	sales	2013	1600000

THEN the result is produced which involves calculating the 'SUM(amount)' values:

dept	SUM(amount)
research	1500800
sales	3100300

GROUP BY

- The sum of expenses grouped by ***expense and department***:

expenses:

expense	dept	year	amount
salary	research	2011	490000
salary	sales	2012	1500000
salary	research	2012	500000
coffee	research	2013	800
coffee	sales	2013	300
salary	sales	2013	1600000
salary	research	2013	510000

```
SELECT expense, dept, SUM(amount) AS total FROM expenses
  GROUP BY expense, dept ;
```

The table is first grouped by expense:

exp.	dept	year	amount
coffee	research	2013	800
coffee	sales	2013	300
salary	research	2011	490000
salary	sales	2012	1500000
salary	research	2012	500000
salary	sales	2013	1600000
salary	research	2013	510000

THEN, the table is sub-grouped by dept.:

exp.	dept	year	amount
coffee	research	2013	800
coffee	sales	2013	300
salary	research	2011	490000
salary	research	2012	500000
salary	research	2013	510000
salary	sales	2012	1500000
salary	sales	2013	1600000

FINALLY, the result is produced:

exp.	dept	total
coffee	research	800
coffee	sales	300
salary	research	1500000
salary	sales	3100000

GROUP BY

expenses:

expense	dept	year	amount
salary	research	2011	490000
salary	sales	2012	1500000
salary	research	2012	500000
coffee	research	2013	800
coffee	sales	2013	300
salary	sales	2013	1600000
salary	research	2013	510000

- The sum of expenses grouped by ***expense and department 2012-13***:

```
SELECT expense, dept, SUM(amount) AS total FROM expenses
  WHERE 2012 <= year AND year <= 2013 GROUP BY expense, dept ;
```

The table is first grouped by expense:

exp.	dept	year	amount
coffee	research	2013	800
coffee	sales	2013	300
~~salary~~	~~research~~	~~2011~~	~~490000~~
salary	sales	2012	1500000
salary	research	2012	500000
salary	sales	2013	1600000
salary	research	2013	510000

THEN, the table is sub-grouped by dept.:

exp.	dept	year	amount
coffee	research	2013	800
coffee	sales	2013	300
~~salary~~	~~research~~	~~2011~~	~~490000~~
salary	research	2012	500000
salary	research	2013	510000
salary	sales	2012	1500000
salary	sales	2013	1600000

FINALLY, the result is produced:

exp.	dept	total
coffee	research	800
coffee	sales	300
salary	research	1010000
salary	sales	3100000

EXERCISE 3

- What is the ***total amount spent per expense***?

expenses:

expense	dept	year	amount
salary	research	2011	490000
salary	sales	2012	1500000
salary	research	2012	500000
coffee	research	2013	800
coffee	sales	2013	300
salary	sales	2013	1600000
salary	research	2013	510000

expense	SUM(amount)
coffee	1100
salary	4600000

EXERCISE 4

- What is the ***average amount per expense***?

expenses:

expense	dept	year	amount
salary	research	2011	490000
salary	sales	2012	1500000
salary	research	2012	500000
coffee	research	2013	800
coffee	sales	2013	300
salary	sales	2013	1600000
salary	research	2013	510000

expense	AVG(amount)
coffeee	550
salary	920000

EXERCISE 5

- What are the ***total expenses each year***?

expenses:

expense	dept	year	amount
salary	research	2011	490000
salary	sales	2012	1500000
salary	research	2012	500000
coffee	research	2013	800
coffee	sales	2013	300
salary	sales	2013	1600000
salary	research	2013	510000

year	SUM(amount)
2011	490000
2012	2000000
2013	2111100

EXERCISE 6

- What are the ***expenses per year per department***?

expenses:

expense	**dept**	**year**	**amount**
salary	research	2011	490000
salary	sales	2012	1500000
salary	research	2012	500000
coffee	research	2013	800
coffee	sales	2013	300
salary	sales	2013	1600000
salary	research	2013	510000

year	**dept**	**total**
2011	research	490000
2012	sales	1500000
2012	research	500000
2013	research	510800
2013	sales	1600300

EXERCISE 7

- What are the ***expenses each year in 2012-13***?

expenses:

expense	dept	year	amount
salary	research	2011	490000
salary	sales	2012	1500000
salary	research	2012	500000
coffee	research	2013	800
coffee	sales	2013	300
salary	sales	2013	1600000
salary	research	2013	510000

year	SUM(amount)
2012	2000000
2013	2111100

EXERCISE 8

- What is the ***highest expenditure for each dept?***

expenses:

expense	dept	year	amount
salary	research	2011	490000
salary	sales	2012	1500000
salary	research	2012	500000
coffee	research	2013	800
coffee	sales	2013	300
salary	sales	2013	1600000
salary	research	2013	510000

dept	MAX(amount)
research	510000
sales	1600000

EXERCISE 9

- What is the ***highest exp. for each dept per year***?

expenses:

expense	dept	year	amount
salary	research	2011	490000
salary	sales	2012	1500000
salary	research	2012	500000
coffee	research	2013	800
coffee	sales	2013	300
salary	sales	2013	1600000
salary	research	2013	510000

dept	year	MAX(amount)
research	2011	490000
research	2012	500000
research	2013	510000
sales	2012	1500000
sales	2013	1600000

A G E N D A

- Recap
- Join (samkøring af data)
- Indices
- Left and Right Join
- Group by
- **Java + SQL !**

Java + SQL !

```
SELECT * FROM mailing_list ;
```

mailing_list:

name	email
Barack	obama@hotmail.com
Michelle	m.obama@hotmail.com

```java
import java.sql.*; // Import required packages

public class MySQL {
  // JDBC driver name and database URL
  static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
  static final String database_name = "grpro";
  static final String DB_URL = "jdbc:mysql://mysql.itu.dk/" + database_name;

  //  Database credentials
  static final String USER = "brabrand";
  static final String PASS = "********";

  public static void main(String[] args) {
    Connection connection = null;
    Statement statement = null;
    try {
      DriverManager.registerDriver(new com.mysql.jdbc.Driver()); // Register driver
      connection = DriverManager.getConnection(DB_URL, USER, PASS); // Open connection
      statement = connection.createStatement(); // Create statement

      String sql = "SELECT * FROM mailing_list"; // NB: implicit semi-colon!
      ResultSet rs = statement.executeQuery(sql); // *** EXECUTE QUERY! ***
      // STEP 5: Extract data from result set
      while (rs.next()) { // Retrieve data by column name
        String email = rs.getString("email");
        String name = rs.getString("name");
        // int id  = rs.getInt("id");
        System.out.println("Name: '" + name + "', Email: '" + email + "'"); // Display data
      }
      rs.close(); // close query
      connection.close(); // close connection
    } catch(Exception e) {
      e.printStackTrace(); // handle errors
    }
  }
}
```

Compile and Run

- Jar (you have to download it):

```
mysql.jar
```

- Compile:

```
%> javac -cp mysql.jar MySQL.java
```

- Run:

```
%> java -cp mysql.jar:. MySQL
Name: 'Barack', Email: 'obama@hotmail.com'
Name: 'Michelle', Email: 'm.obama@hotmail.com'
```

SQL Injection Attacks :-)

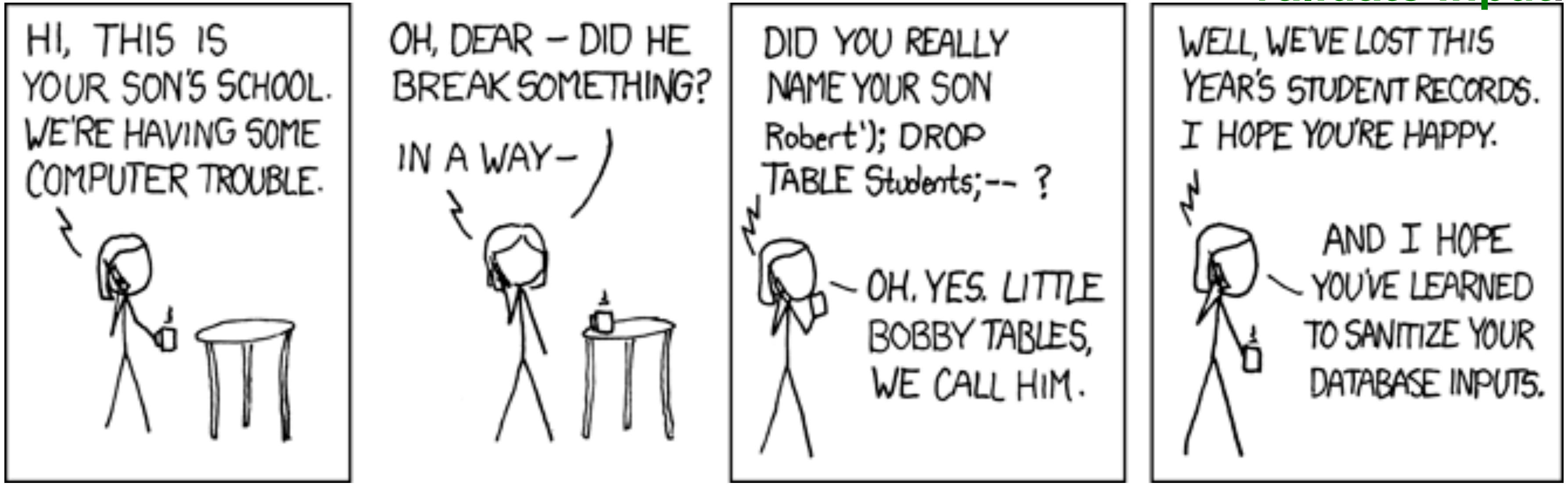
- "Inject" following SQL command:

```
DROP TABLE mailing_list ;
```

mailing_list:

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

Remember to **validate input**!


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

SQL Injection Attacks :-)

Exercises 9 (for Oct 28, 2014)

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 three tables (each with its own '`id`' column):
  - "`Courses`", "`Students`", and "`Enrollment`"
- ...and populate your database with sample real'ish data

Exercise 9.3:

- Try out the various queries from these slides (GRPRO-15/16)

Exercises 9 (for Oct 28, 2014)

Exercise 9.4: (Three-way joins)

- Join data from your ***three tables*** as explained below:
  - "`Courses`", "`Students`", and "`Enrollment`"
- a) Select all students attending "**Grundlæggende Programmering**"

(don't simply use the key: "GRPRO")

student	course
Bent	Grundlæggende Programmering
Anna	Grundlæggende Programmering

- b) Compute the student-teacher relation:

student	teacher
Bent	Claus
Carl	Mathias
Anna	TBA
Anna	Claus

Exercise 9.5:

- Find some (any) data online and turn it into at least two tables.
- Perform a **join** on it. Finally send your SQL commands to me. **:-)**

Thx!

Questions?