COMP207 Database Development

Lecture 2

Basic SQL

Kortext download

- You can download the book and read it offline using the provided apps

Slides

- Will upload slides from last year next week

SQL Database

- Data is organised in **tables** (also called **relations**)

Attributes (or columns)

Table name

Tuple (or row)

Students

name	number	progra
Anna	20171989	G402
John	20174378	G702
…	…	…

Module

code	name
COMP207	Database Development
COMP219	Artificial Intelligence
…	…

All elements in a column are of the same datatype

Other things

- **Views** – intuitively, saved **queries**

This lecture – also in Lab

- **Indices** – intuitively, search trees

Later lecture – also in Lab

- **Procedures** – intuitively, like programming language procedures

Won't cover in this course – wait for COMP283

- **Triggers** – procedures that runs when certain things happens

SQL Language

- SQL databases are accessed using SQL language
- The standard for the SQL language is updated every few years (86, 89, 92, 99, 03, 06, 08, 11, 16 and 19)
- The implementations does not follow the standard that well though

SQL Language Parts

- Data Definition Language (DDL)
  - Create/alter/delete databases, **tables** and their attributes

This lecture

- Data Manipulation Language (DML)
  - Add/remove/update and query **rows** in tables

The following 9-10 lectures…

- Transact-SQL
  - Intuitively, do a **sequence** of SQL statements

Data Definition Language (DDL)

```
CREATE DATABASE databasename;
```

– Creates a database named databasename

```
CREATE TABLE Table_name (
   column1 datatype,
   column2 datatype,
   column3 datatype,
  ...
);`
```

– Creates a table named table_name with 3 columns, named column1,column2,column3, each with a datatype of datatype

Datatypes

- INT – integers
- FLOAT – decimal numbers
- CHAR(x) – x is an integer, fixed length string
- VARCHAR(x) – x is an integer, variable length string
- DATE – dates
- DATETIME – for time and dates
- XML – for XML files

Will talk abt. XML later in course

- BLOB – binary files (e.g. programs)
- Others for other precisions

Datatype usage

```
CREATE TABLE Students2 (
   name VARCHAR(100),
   birthday DATE,
   student_no INT
);
```

- Creates a table named students with a name, birthday and student_no columns
- It has **schema** students(name, birthday, student_no)
  - i.e. name(name_of_attribute1, name_of_attribute2,…) in general
  - Used when wanting to talk about the database

Primary keys

```
CREATE TABLE Students2 (
   name VARCHAR(100),
   birthday DATE,
   student_no INT,
   CONSTRAINT PK_st PRIMARY KEY (student_no)
);
```

- **Primary keys** are how the rows are sorted on disk
  - Also requires uniqueness, i.e. only 1 row with a given student_no
  - Makes searching for student_no faster

Data Definition Language (DDL)

```
DROP TABLE Students2;
```

– Deletes the students table

```
ALTER TABLE Students2 ADD email VARCHAR(100);
```

– Adds an email attribute to the student

Exact command depends on implementation

```
ALTER TABLE Students2 MODIFY COLUMN
email VARCHAR(200);
```

– Changes the email attribute to be a string of length at most 200

```
ALTER TABLE Students2 DROP COLUMN email;
```

– Removes the email attribute from the students table

Data Manipulation Language (DML)

- Insert rows into a table
- Query a table
- Delete/update rows from a table

Insert

Students

name	number	programme
Anna	20171989	G402
John	20174378	G702

```
INSERT INTO Students
VALUES ('Oliver',20171112,'G402');
```

Students

name	number	programme
Anna	20171989	G402
John	20174378	G702
Oliver	20171112	G402

Insert

Students

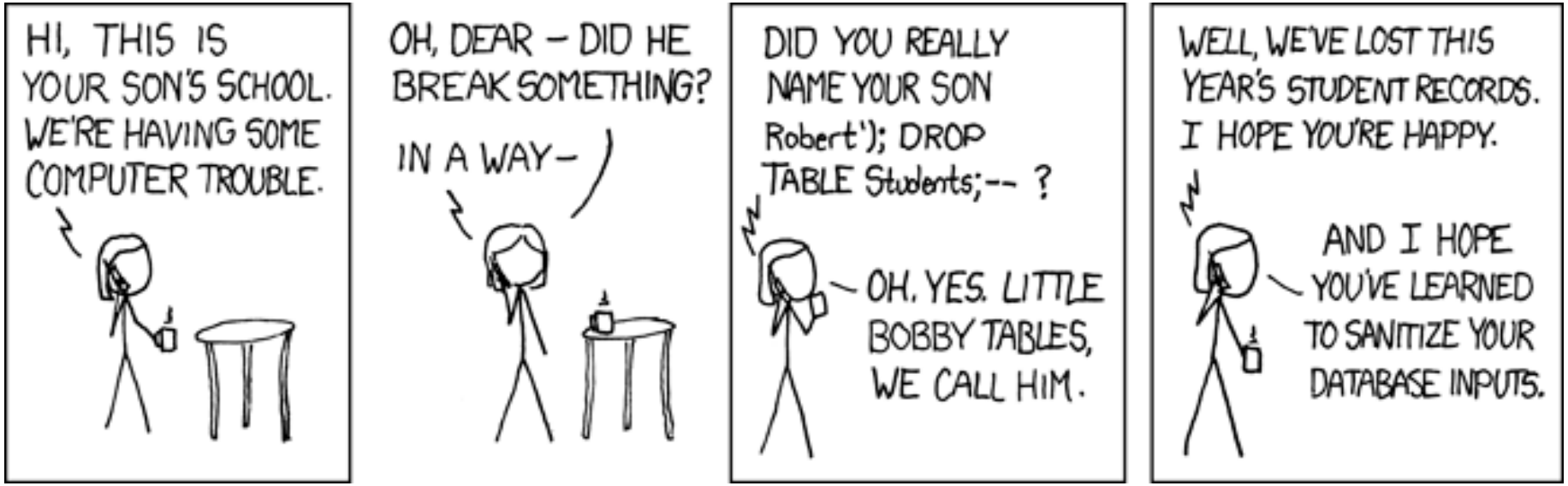
name	number	programme
Anna	20171989	G402
John	20174378	G702
Oliver	20171112	G402

```
INSERT INTO Students(programme,name)
VALUES('G702','Danny');
```

Students

name	number	programme
Anna	20171989	G402
John	20174378	G702
Oliver	20171112	G402
Danny	*null*	G702

SQL Injections



- This is the most common exploit online (~50% of all)
- What happens is: You make an website with a form, you take what people inputs into the form and insert the fields into your SQL database

Insert

Students

name	number	programme
Anna	20171189	G402
John	20174378	G702
Olive	20171112	G402
Danny	*null*	G702

```
INSERT INTO Students(programme,name)
VALUES('G702','Robert'); DROP TABLE students; --');
```

Queries

Students

name	number	programme
~~Anna~~	~~20171989~~	~~G402~~
~~John~~	~~20174378~~	~~G702~~
Oliver	~~20171112~~	G402
~~Danny~~	~~*null*~~	~~G702~~

```
SELECT name, programme
FROM Students
WHERE number=20171112;
```

name	programme
Oliver	G402

Algebra

- Algebra = branch of math

- Example: Algebra with numbers
  - Addition (written: +) = function that takes two numbers and returns a number
  - Logarithm (written: log) = function that takes one number and returns another
    - Has a subscript – the base
  - …
- Can be composed, e.g.: $\log_2 (3 + 8) + 4$

Relational algebra

- Algebra with tables ≈ SQL SELECT queries
  - Exception: uses set semantics (i.e. removes duplicates)

- Relational algebra is crucial for optimization
  - Later in the course

Selection (σ)

- $\sigma_{\text{condition}}(R)$ = set of all tuples in R that satisfy the condition

SQL query

```
SELECT *
FROM Modules
WHERE year = 2 AND sem = 1;
```

The SELECT keyword in SQL has nothing to do with the selection operator!

translates into

$\sigma_{\text{year=2 AND sem=1}}(\text{Modules})$

Relational algebra expression

Modules

code	year	sem
COMP105	1	1
COMP201	2	1
COMP202	2	2
COMP207	2	1

=

code	year	sem
COMP201	2	1
COMP207	2	1

Conditions

- Conditions can contain:
  - AND
  - OR
  - NOT
  - =,<,<=,>=,>,<> (or != for the last)
  - BETWEEN
    - E.g. “Price BETWEEN 10 AND 20”
  - LIKE
    - For string matching
    - _ matches any 1 letter and % any number of letters
    - E.g. “Name LIKE ‘O%r’” and “Name LIKE ‘O____r’” matches Oliver

Conditions: IN

```
SELECT *
FROM Students
WHERE name IN (SELECT name
               FROM Lecturers);
```

Students

name	number	programme
Anna	20171989	G402
John	20174378	G702
Oliver	20171112	G702
Danny	*null*	G702

Lecturers

name	module
John	COMP105
Sebastian	COMP201

name	number	programme
John	20174378	G702

Projection (π)

- $\pi_{\text{attribute list}}(\text{R})$ = restricts R to the attributes in attribute list

SQL query

```
SELECT sem, module
FROM Modules;
```

translates into

Attribute order matters

$\pi_{\text{sem, module}}(\text{Modules})$

Relational algebra expression

=

Modules

code	year	sem
COMP105	1	1
COMP201	2	1
COMP202	2	2
COMP207	2	1

sem	code
1	COMP105
1	COMP201
2	COMP202
1	COMP207

SQL vs. Relational Algebra

- $\pi_{\text{attribute list}}(R)$ = restricts R to the attributes in attribute list

SQL query

```
SELECT name
FROM Students;
```

name
Anna
John
Anna
Oliver
Danny

Gives

Does not translate into

$\pi_{\text{name}}(\text{Students})$

Relational algebra expression

=

name
Anna
John
Oliver
Danny

Students

name	number	programme
Anna	20171989	G402
John	20174378	G702
Oliver	20171112	G702
Danny	*null*	G702
Anna	20171234	G702

DISTINCT

- $\pi_{\text{attribute list}}(R)$ = restricts R to the attributes in attribute list

SQL query

```
SELECT DISTINCT name
FROM Students;
```

translates into

$\pi_{\text{name}}(\text{Students})$

Relational algebra expression

=

Students

name	number	programme
Anna	20171989	G402
John	20174378	G702
Oliver	20171112	G702
Danny	*null*	G702
Anna	20171234	G702

name
Anna
John
Oliver
Danny

Renaming (ρ)

- $\rho_{A1 \rightarrow B1, A2 \rightarrow B2, \ldots}(R)$ = renames attribute **A1** to **B1**, attribute **A2** to **B2**, …

SQL query

```
SELECT module AS module_code, year, sem
FROM Modules2;
```

Modules2

code	year	sem
COMP105	1	1
COMP201	2	1

translates into

$\rho_{module \rightarrow module\_code}(Modules2)$ =

Relational algebra expression

module_code	year	sem
COMP105	1	1
COMP201	2	1

Cartesian Product (×)

- $R_1 \times R_2$ = pairs each tuple in R_1 with each tuple in R_2

SQL query

```
SELECT *
FROM Modules, Lecturers;
```

translates into

Modules × Lecturers

Relational algebra expression

Modules2

code	year	sem
COMP105	1	1
COMP201	2	1

Lecturers

name	module
John	COMP105
Sebastian	COMP201

=

code	year	sem	name	module
COMP105	1	1	John	COMP105
COMP105	1	1	Sebastian	COMP201
COMP201	2	1	John	COMP105
COMP201	2	1	Sebastian	COMP201

Combining Operators

- Operators can be combined:

$$\pi_{\text{name,module}}(\sigma_{\text{code=module AND year=2}}(\text{Modules2} \times \text{Lecturers}))$$

Modules

code	year	sem
COMP105	1	1
COMP201	2	1

Lecturers

name	module
John	COMP105
Sebastian	COMP201

~~code~~	~~year~~	~~sem~~	name	module
~~COMP105~~	~~1~~	~~1~~	~~John~~	~~COMP105~~
~~COMP105~~	~~1~~	~~1~~	~~Sebastian~~	~~COMP201~~
~~COMP201~~	~~2~~	~~1~~	~~John~~	~~COMP105~~
~~COMP201~~	~~2~~	~~1~~	Sebastian	COMP201

```
SELECT name, module
FROM Modules2, Lecturers
WHERE code=module AND year=2;
```

Non-relational Algebra parts

- Sorted tables
  - Add ORDER BY name at the end to sort by name e.g.

- Limit on output size
  - Basically all implementations can do it…
  - … each with their own command

- Aggregations

Will talk more about these towards the end of the course

  - For e.g. finding the average grade for each course

Views

- Intuitively, saved queries or virtual tables

```
CREATE VIEW Year_2_modules AS
SELECT module, name
FROM Modules2, Lecturers
WHERE code=module AND year=2;
```

Year_2_modules

name	module
Sebastian	COMP201

```
SELECT name
FROM Year_2_modules;
```

name
Sebastian

More on Views

- Views can be updated

```
CREATE OR REPLACE VIEW Year_2_modules AS
SELECT module, name as lecturer
FROM Modules, Lecturers
WHERE code=module AND year=2;
```

Year_2_modules

name	module
Sebastian	COMP201

~~Year_2_modules~~

~~lecturer~~	~~module~~
~~Sebastian~~	~~COMP201~~

- … and removed

```
DROP VIEW Year_2_modules;
```

Data Manipulation Language (DML)

- Insert rows into a table
- Query a table
- Delete/update rows from a table

DELETE

- How to remove John

```
DELETE FROM Students
WHERE number=20174378;
```

Students

name	number	programme
Anna	20171989	G402
Oliver	20171112	G702
Danny	*null*	G702
Anna	20171234	G702
Anna	20171234	G702

- Warning:

```
DELETE FROM Students;
WHERE number=20174378;
```

UPDATE

- How to change Oliver

```
UPDATE Students
SET programme='G402'
WHERE number=20171112;
```

Students

name	number	programme
Anna	20171989	G402
Oliver	20171113	G402
Danny	*null*	G702
Anna	20171234	G702

- Relative changes

```
UPDATE Students
SET number=number+1
WHERE number=20171112;
```

Examples

- Write a query to find Oliver's programme
- Write a command to insert John again (number=20171112, programme=G702)
- Write a query to find when each lecturer's module is running

Modules

code	year	sem
COMP105	1	1
COMP201	2	1
COMP202	2	2
COMP207	2	1

Students

name	number	programme
Anna	20171989	G402
Oliver	20171113	G402
Danny	*null*	G702
Anna	20171234	G702

Lecturers

name	module
John	COMP105
Sebastian	COMP201

Answer 1

- Write a query to find Oliver's programme

```
SELECT programme
FROM Students
WHERE name='Oliver';
```

Modules

code	year	sem
COMP105	1	1
COMP201	2	1
COMP202	2	2
COMP207	2	1

Students

name	number	programme
Anna	20171989	G402
Oliver	20171113	G402
Danny	*null*	G702
Anna	20171234	G702

Lecturers

name	module
John	COMP105
Sebastian	COMP201

Answer 2

- Write a command to insert John again (number=20171112, programme=G702)

```
INSERT INTO Students
VALUES ('John',20174378,'G702');
```

Modules

code	year	sem
COMP105	1	1
COMP201	2	1
COMP202	2	2
COMP207	2	1

Students

name	number	programme
Anna	20171989	G402
Oliver	20171113	G402
Danny	*null*	G702
Anna	20171234	G702

Lecturers

name	module
John	COMP105
Sebastian	COMP201

Answer 3

- Write a query to find when each lecturer's module is running

```
SELECT *
FROM Modules, Lecturers
WHERE code=module;
```

Modules

code	year	sem
COMP105	1	1
COMP201	2	1
COMP202	2	2
COMP207	2	1

Students

name	number	programme
Anna	20171989	G402
Oliver	20171113	G402
Danny	*null*	G702
Anna	20171234	G702

Lecturers

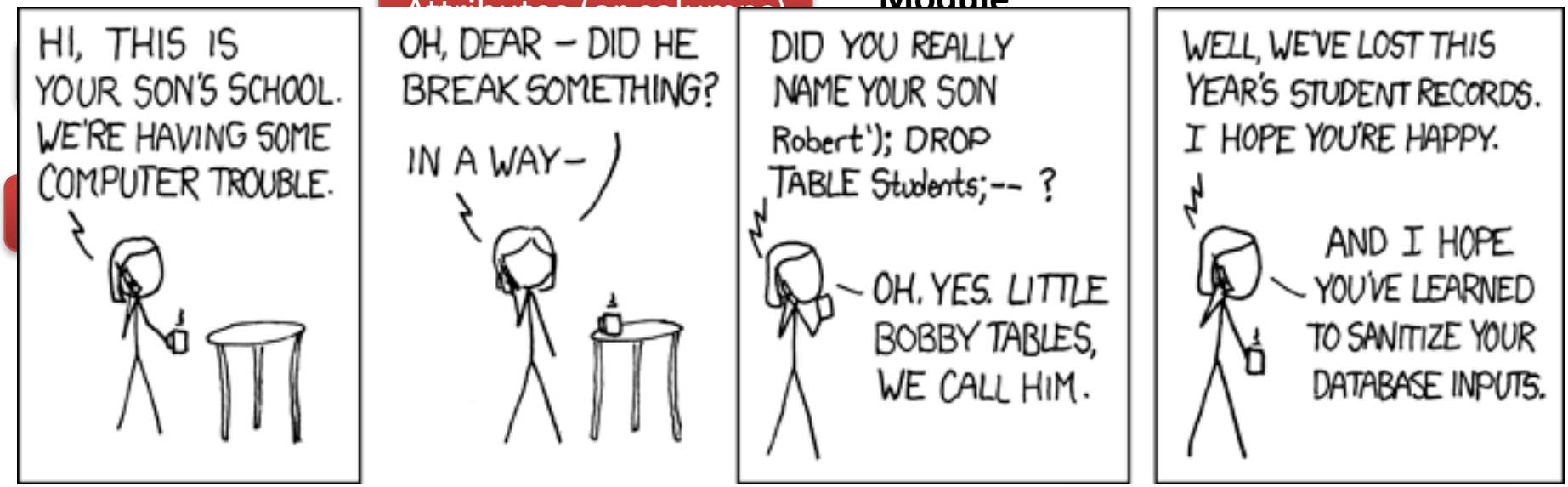
name	module
John	COMP105
Sebastian	COMP201

Summery

- SQL databases

Attributes (or columns)

Module




```
SELECT module, name
FROM Modules, Lecturers
WHERE code=module AND year=2;
```