

COMPSCI280

Enterprise Software Development

Ulrich Speidel
ulrich@cs.auckland.ac.nz

Lectures

- Tue 3-4 pm – Venue: G20 Science Centre (PLT1)
- Wed 3-4 pm – Venue: G23 Science Centre (MLT1)
- Fri 3-4 pm – Venue: Engineering 1.439

Course outline first half – week 1

- Week 1: Basic concepts of a SQL database
- Lecture 1 (1 March): Course intro. DB server, client, queries, storing data in tables, fields, data definition language (CREATE TABLE statements), inserts, simple select queries without joins or where clause, datasets
- Lecture 2 (2 March): Finding stuff in tables: the where clause, keys
- Lecture 3 (4 March): Updates, changing tables, design considerations (data types)

Lecture 1

- DB servers and clients
- storing data in tables
- fields
- data definition language (CREATE TABLE statements)
- Inserts
- simple select queries without joins or where clause
- datasets

Storing data in files

- Writing data to a file from a computer program is easy, as is reading from a file
- BUT: There are many cases when this is not ideal
 - when we have a lot of data to store (file size)
 - when the data is complex and many *relations* exist within the data
 - when several different programs (possibly on different computers) need simultaneous write and read access to the data
 - when we need to do common operations on the data, e.g., sorting, filtering according to certain criteria
 - when access to the data needs to be restricted for security and consistency reasons
- In such cases, we need a database in the form of a Database Management System (DBMS)

Example: Data storage in files

- Many e-mail programs still store e-mail messages in text files, with one file per mailbox
- Result: many messages per file, potentially huge files (gigabytes!) that don't fit into the computer's RAM memory and need to be read/written piecewise (=slow!)
- Difficult to locate individual messages in the big file: need additional files for indexing etc.
- E-mail program must provide functionality such as finding all messages from a particular sender, sorting messages, etc.
- Done this way for historical reasons – nowadays, one would use a DBMS for this.

Features of a DBMS

- A DBMS separates the data from the application(s) that create and use the data
- It provides a uniform way of defining the data, adding, updating and reading data that is application-independent
- Several applications may work with the same database
- Example: the UofA student database is used by everything from student services online to the program that creates your own personal databases for this course, and eventually the application that generates your official academic record when you apply for a job after uni.

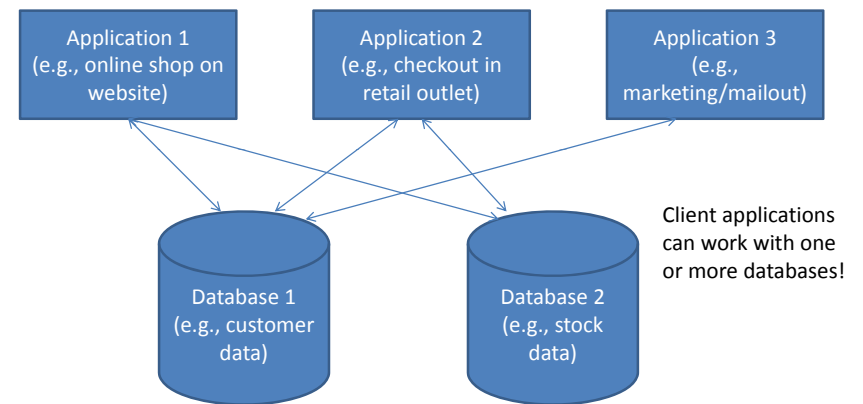
DBMS vs. RDBMS

- Most databases in use today follow the *relational* model
- The relational model recognizes that certain pieces of data form a relation, e.g., that a student always has a birth date and a student ID number.
- An RDBMS lets you define complex relations between items of data. Most of the time this is far easier to grasp in practice than in theory
- Perhaps the best-known RDBMS are those that use SQL, the *Structured Query Language*, to manage their data
- Well-known examples are MySQL, Microsoft SQL server, and Oracle, but there are many other SQL-based systems in use
- Relational database model was proposed by Edgar Frank Codd in 1970

Database servers vs. clients

- What if more than one application needs access to the same data? Then we cannot make the DBMS part of the application itself.
- What if the applications are on different computers?
- Answer: the database needs to be on a *server*, and the applications that need to access the database act as *clients*.

DB servers and clients



Storing data in tables

- A common way for relational data to be stored in DBMS is in *tables*:

- In formal database speak, a table defines a *relation*. In databases, tables typically have names – think of it like a variable that stores the relation.

Article number	Description	Items in stock	Price
777-8888	Rusty bolt	21	\$0.99
777-8889	Wet bus ticket	3	\$3.20
777-8890	Can of worms	102	\$102.95
777-8891	White elephant	1	\$1,533,087.33
777-8895	Good news	0	\$1,000.00

E.g., the table here might be called **stock**.

- Note that a table consists of structure (columns, also called *fields* or, more formally, *attributes*) and data (rows, also known as *records*, or more formally *tuples*).
- Note also that the table above is not “normalized” – we’ll get to that later...

Tables in relational databases

- Columns (fields) are generally defined to be of a particular data type (string, integer, float, binary object, date, ...). This helps the RDBMS to keep storage space to a minimum and also tells it how to handle the data (e.g., sort it)
- The order of columns (fields) is in principle not important, except when we want to change the definition of a table (e.g. insert a column), or in some clients that return a row as an untyped array with elements in column order (so if we want to read the value of the 3rd column in that row, we need to read the 3rd element of the array etc.)
- The order that rows (records) are stored in is similarly not important and cannot be relied upon. Even if records are added in a particular sequence, the RDBMS may change this order during updates. We’ll discuss later what you do if you need your records in a particular order.

Talking to a database with SQL

- SQL (Structured Query Language) is pronounced “es-queue-ell” but also known as “sequel”. Both are OK.
- Dialects of SQL exist and almost every RDBMS has its own
- We use SQL to:
 - create databases
 - create tables in a database
 - insert records into tables in a database
 - read data from a database
 - update records in tables of a database
 - ... and a few more things

Trying this out

- You should have received an e-mail before this lecture giving you the access data to your own database on the server `studdb-mysql.fos.auckland.ac.nz`
- Log into the server `login.fos.auckland.ac.nz` with PuTTY and your normal university password. Once logged in, issue the command:

```
mysql -h studdb-mysql -u your_api -p
```

and enter the password you received in the mail.

This logs you into the database server `studdb-mysql.fos.auckland.ac.nz` using the mysql command line client installed on `login.fos.auckland.ac.nz`
- Then enter the command

```
use stu_your_api_COMPSCI_280_C_S1_2011;
```
- All further SQL statements can now be entered from the command line of the mysql client

SQL preliminaries

- Whitespace is treated similarly to Java or C source code
- Can break an SQL statement across lines
- End of an SQL statement is denoted by a semicolon to separate it from the next statement – very similar to Java or C!
- On most SQL RDBMS, SQL is nOt CaSE SenSITive! Don't let that lead you to an inconsistent SQL coding style, though!
- The rules that apply for the selection of table and field names are broadly the same as for variable names in Java or C: start with a letter and continue with alphanumeric characters, avoid reserved words, and you will be fine. Most other things are possible but usually need to go into some sort of backquotes etc.

Creating a database in SQL

- This step is done for you on `studdb-mysql` but on a real job, you may need to do this:

```
create database name_of_your_db;
```
- To get rid of a database, use:

```
drop database name_of_your_db;
```
- You may also need to set up user privileges on a DB. Many RDBMS allow you to specify access privileges down to table and field level!

Creating a table in SQL

- This is done with the CREATE TABLE statement
- As a minimum, it specifies the name of the table and the name and type of each field:

```
CREATE TABLE stock (  
    `Article number` varchar(8),  
    Description varchar(100),  
    `Items in stock` int,  
    Price double  
);
```

- Note that we have put field names with spaces into backticks here.
- A **varchar** is a string of characters up to the specified length

Considerations when creating a table

Databases can grow large, so in order to avoid them taking up too much space and to help the DBMS find data quickly:

- Consider the data type for each field seriously: E.g., 1234 takes up different amounts of storage space depending on what it is stored as:
 - INT: typically 4 bytes
 - VARCHAR: typically 5 bytes
 - DOUBLE: typically 8 bytes
 - ... and this is just one example!
- Avoid replicating data in different fields
- Other considerations may be the choice of DB engine, e.g., under MySQL
- More on this topic later (see: indexes/keys, normalization, computed fields, and views)

Putting data into a table

- This is done with the INSERT statement. There are different forms of the statement, but this is the most common one:

```
INSERT  
    INTO <table_name>  
        (<field_list>)  
    VALUES  
        (<value_list>);
```

- By specifying a list of fields and values to be inserted into them, we do not need to remember the order of the fields in the table, and we do not need to specify values for fields we want to leave empty

INSERT example

- The following inserts a line into our **stock** table:

```
INSERT  
    INTO stock  
        (`Article number`, `Description`,  
        `Items in stock`, `Price`)  
    VALUES  
        ("777-8888", "Rusty bolt", 21, "0.99");
```

- Note: Can put numbers into quotes but don't have to! Sometimes there are good reasons to put all values into quotes, especially if they are supplied by a remote user (e.g., in a web application)

Simplified INSERT

- The following also inserts a line into our **stock** table:

```
INSERT
  INTO stock
  VALUES
    ("777-8889", "Wet bus ticket", 3.20);
```

- But: must know the order of our columns here, and must always specify values for all columns

Getting data back out of our SQL DB

- This is done with the SELECT statement
- There are many ways to use SELECT, and we will discuss quite a few more in this course
- To see all data in a table, use:

```
SELECT * FROM <table_name>;
```

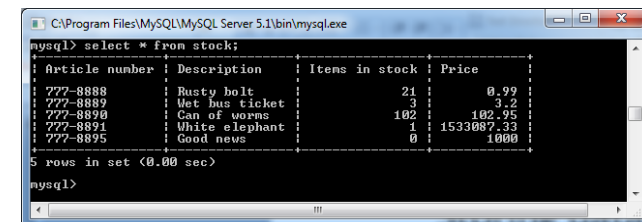
SELECT example

- To get all records from our **stock** table:

```
SELECT * FROM stock;
```

Datasets

- SELECT queries return datasets
- You can think of a dataset as a table that doesn't get stored and that contains the results of the SELECT query



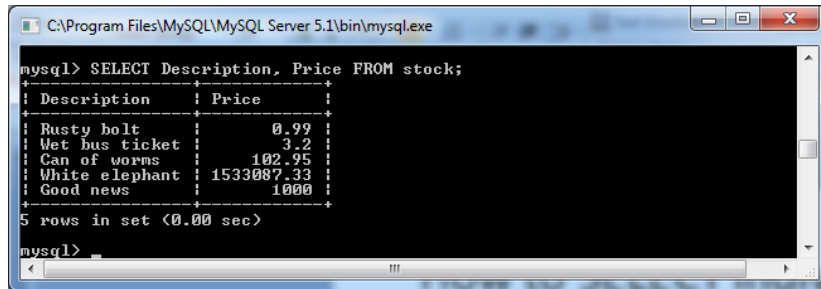
```
mysql> select * from stock;
+-----+-----+-----+-----+
| Article number | Description | Items in stock | Price |
+-----+-----+-----+-----+
| 777-8888      | Rusty bolt | 21              | 0.99  |
| 777-8889      | Wet bus ticket | 3              | 3.2   |
| 777-8890      | Can of worms | 102             | 102.95|
| 777-8891      | White elephant | 1              | 1533087.33|
| 777-8895      | Good news | 0               | 1000  |
+-----+-----+-----+-----+
5 rows in set (0.00 sec)

mysql>
```

How to SELECT individual fields only

- Say we only wanted the description and the price of an item. Then we can use the following SELECT query:

```
SELECT Description, Price FROM stock;
```



```
mysql> SELECT Description, Price FROM stock;
+-----+-----+
| Description | Price |
+-----+-----+
| Rusty bolt  | 0.99  |
| Wet bus ticket | 3.2   |
| Can of worms | 102.95|
| White elephant | 1533087.33|
| Good news   | 1000  |
+-----+-----+
5 rows in set (0.00 sec)

mysql>
```

Much more on SELECT ...

- ... in lectures to come!

Today's lab sheet

- ...is on the web
- Today: Connecting to a database server via command line, creating a table with CREATE TABLE, populating it with data using INSERT, and reading the data back out with SELECT
- Is not assessed, but its content is examinable in test and exam!