Introduction to Databases

1 Introduction to the relational data model and simple SQL queries

1. What is a database? In the *widest possible sense*, any collection of data can be called a "database". Examples, then, are the University Library, a telephone book, or the World Wide Web.

In a *narrower sense* we use the word "database" to refer to a **computer-based** collection of **non-perishable** data. We can no longer use the library as an example, but there are still many different kinds of databases in this sense: a single file, or a directory of files, or the World Wide Web. The adjective "non-perishable" is important as it emphasises the difference between databases and the data ordinarily processed by computer programs; the latter is erased when the program terminates.

In an *even narrower sense* the word "database" refers to a computer-based **structured** collection of non-perishable data. In this interpretation we no longer admit arbitrary files, and also web-pages do not qualify. Instead we insist that all items of data are stored in a specific format. The two best-known examples are **relational databases**, where all data is stored as lines in **tables**, and **XML databases**, where all data is stored as **labelled trees**.

Finally, in the *narrowest sense* of relational database terminology, we denote with "database" a **collection of tables** (about which we will say more below). In practice, the tables of a database will describe *one organisation* or *one aspect of an organisation*. So the University, for example, maintains a database which describes students' exam results, another database for its employees, yet another to do with its buildings and equipment, and many more. **Throughout these notes we will use the word "database" in sense of relational database terminology**.

2. What is a database management system (DBMS)? This is a computer program which maintains a database. This general task can be divided into many different subtasks. The main ones:

Access The DBMS provides functionality for users to access the data in the database. Instead of "accessing data in the database" one usually says "querying the database".

Safety It contains many safeguards against unwanted loss of data, e.g. from hardware failure.

Security It has mechanisms with which we can control who is allowed to see which parts of the database.

Efficiency It contains facilities which help to return results to queries quickly.

This course will use the public domain DBMS PostgreSQL. Other common DBMSs are "MySQL" (also public domain), "DB2" (an IBM product), Oracle, "Microsoft Access", and "SQL Server" (also by Microsoft). DBMSs differ in the functionality they offer to the user but they all implement the basic elements which will be covered in this course. It is worth pointing out that the course is not a training programme in a particular DBMS or in the language SQL, but it tries to give an introduction to the fundamental elements of database theory and practice.

3. Tables. No doubt you know what a table looks like but just to focus the discussion, here is an example (taken from the existing table "`staff`" in the course database):

firstname	**lastname**	**office**	**phone**
Achim	Jung	213	44776
Robert	Hendley	236	44761
Martin	Escardo	212	42797
⋮	⋮	⋮	⋮

Because tables are central to relational databases, a lot of terminology has developed around this concept. For example, instead of "table" people also use the word "relation". This explains why databases which consist of tables are called "relational", and it refers to a mathematical description of this form of databases. Although we will sometimes use "relation" instead of "table", we have no intention of pursuing this connection.

Tables have **rows** and **columns**. For these, too, there are various synonyms. A row is also called an "entry" (because it is something that is usually *entered* into the table in one operation), or a "tuple" (because it consists of

a sequence of fields), or a "record" (because it *records* information about one specific event or entity), or a "line" (because tables are always displayed with horizontal rows and vertical columns).
Instead of "column" you may read the expression "attribute" (because one column contains the values of one attribute of the individual entries).
In a database, every table has a name to identify it uniquely; in our example that name is "staff". Normally, we only work with the tables of one database at any one time, and so we only need to keep the table names within a database different. This is exactly as with the names of fields in Java class definitions.
Columns, too, are named; in our example the names are in bold font at the top of the table. Column names need to be different only within a single table; in other words, the same column name can appear in different tables within the *same* database. It is good practice to use the same name only when the columns involved are indeed referring to the same real-world attribute.
All values appearing in a column are of the same **type** where this word has pretty much the same meaning as in programming. In the above table, for example, "firstname" is always a string, whereas "office" is always an integer.

Pause. Do you think that "integer" is a good choice of type for "office"?

It is OK for a table to have no entries at all (for example, after it's just been created). In this case we would only see the first line with the column names. A table with no columns, on the other hand, seems pretty pointless (and PostgreSQL will not allow you to create such a thing).

4. The relational data model. It may be good to repeat what was said above about our approach to databases (called the "relational" one):

All data can be represented in the form of tables.

This is a pretty bold statement, and when it was first made by Edgar F. "Ted" Codd in 1970, it was considered to be radical and outrageous by the database community. However, history has proved him right and by now this data model is dominating the field completely.
When you contemplate Codd's postulate you would probably quickly come up with the following questions:

a. How does one perform the translation from real-world data into the table format?

b. How does one retrieve information from a collection of tables?

We first consider the second and later the first question.

2 Querying a database with SQL

5. Simple select queries. OK, let's get started. Consider the example table shown above. We want to answer the question:

> What is John's phone number?

Of course, we can look it up easily ourselves but let's now imagine that the "staff" table is stored in a database management system and that it is way too big to print out and consult manually. Instead we want to use SQL to retrieve the answer for us. This is what we should write:

```
SELECT phone FROM staff WHERE firstname = 'John';
```

You can understand immediately how this works but let us point out some important aspects of this query language:

- SQL is quite old (development started in the early seventies at IBM) and as was common at the time, it does not distinguish between uppercase and lowercase characters. A consequence of this is that you can not use one of the SQL-keywords as a name for a table or an attribute. Some weird error messages will be returned to you if you don't abide by this rule. This is a bit sad because SQL contains so many keywords; more than one hundred, in fact.

  Although it makes no difference to SQL, it is good programming style to distinguish notationally between SQL-keywords and application specific identifiers. Using all uppercase for the former and all lowercase for the latter is a widely shared practice.

  The string `John` between single quotes, on the other hand, is a string constant and case *does matter* here.

- Observe that we use a single equal sign to indicate equality between "`firstname`" and "`'John'`". This is similar to the practice in Mathematics but in contrast with the convention of Java, where "=" means assignment and "==" means equality.
- Observe that the query is terminated by a semicolon. This is the same as with Java commands, so you should be used to it.

6. How does a database management system evaluate a query? Conceptually, what happens is that the system searches the stored table checking each time whether the `firstname` field matches exactly the string `'John'`. For all the records that succeed the check, the `phone` field is printed to the output. On the technical level, database management systems employ all kinds of clever tricks to speed up the search.
Since there could well be more than one member of staff whose first name is "John", it is possible that the system has to output many phone numbers. The format, in which this is done, is **as another table**. This result table has only one column and the column name will be inherited from "`staff`", namely, it will be "`phone`". It will have as many rows as there are rows in the staff table where the first name is "John". Indeed, here is the output from the course database:

```
 phone
-------
 43816
 42590
 43705
```

In this simplest of queries we thus meet again Codd's dictum that all data can be represented in the form of tables. The output from a query certainly qualifies as "data", hence it is in this format. In other words:

Queries map tables to a table.

This is in complete analogy with arithmetic, where calculations produce numbers from numbers, and it gives you one indication why the relational data model has been so successful.

7. Some technical information on simple `SELECT` queries. A query can request more than one attribute from a table, as in the following:

```
SELECT name, office, phone FROM staff WHERE firstname = 'John';
```

This will produce a three-column table with three entries.
You can even request all the information stored about particular entries. Try:

```
SELECT * FROM staff WHERE phone = 44774;
```

If you run this on the version of `staff` created for this course, then you will find that it contains many more attributes than the four shown here.
An output column can also contain *computed* values, for example, the cost of an item in a different currency. In this case it is advisable to name the column explicitly. Imagine we have a table which lists the price of products in Pounds and we wish to quote the price in Euros. This is what we could write:

```
SELECT price * 1.42 AS "Price in Euros",  productname FROM productlist;
```

Note that the star in this case is interpreted as a multiplication sign, not as a wildcard. The column header for the price column has been given explicitly as a string in double quotes.

Pause. Why is it necessary to write out column headers between double quotes but string constants in the `WHERE`-part are between single quotes? (There is no good answer to this.)

Although an attribute may have type "`int`", once a value is sent to the output, it is transformed into a string. The resulting strings can be concatenated together using the operator "`||`". This is in contrast to Java's output routine `System.out.print` where strings are concatenated by the operator "+". For example, we might find it useful to translate the phone numbers in the staff table into their full national format:

```
SELECT '0121 41' || phone AS "National phone number" FROM staff;
```

Funny, that mix between single and double quotes...
The `WHERE`-condition is a lot like a condition that you might use inside a Java `if`-statement. Basic comparisons between numbers are given by

- `=` for equality
- `<=` for "less than or equal to"; likewise >=, <, and >
- `<>` for "not equal", which is different from Java's `!=`

Most data in databases is stored in (variations of) the string format. SQL, therefore, has some powerful comparison operators for this as well; apart from the test for equality (denoted by "=") it also offers the operator "`LIKE`". This is used in a condition as follows:

```
<some string> LIKE <some pattern>
```

Here `<some string>` would typically be a column name, while `<some pattern>` would be a string between single quotes. The pattern can contain the symbol "_" ("underscore") which is a wildcard for an arbitrary character, and the symbol "`%`" which is a wildcard for any substring (including the empty substring). For example, the query

```
SELECT name FROM staff WHERE firstname LIKE 'Ann%';
```

will find all Ann's, all Anne's, all Anna's and all Annemarie's.
Finally, we point out the operator "`iLIKE`", which behaves like "`LIKE`" except that it likes to ignore case.

Edgar F Codd (1923 – 2003), the British computer scientist who in 1970 suggested the relational data model. He received the Turing Award in 1981.

8. Exercise Using the `fundamentals` database on the PostgreSQL system, develop some simple SQL queries to answer the following questions.

a. Who occupies office number 211?

b. Where is the office of Dr Mark Ryan? (NB: This requires to test for equality of a string. Remember that strings are enclosed in single quotes, and that equality only holds if the two strings are equal character by character. Check this statement by adding some blanks at the end of the string `'Ryan'` in your query or by changing the case of a letter.)

c. Which staff have offices on the first floor?

d. Who are the professors in the School?

e. Leave the program `psql` by typing `\q`.

9. Remember that, when you are working with PostgreSQL inside a Unix shell, you are working with several different command environments. Here is a summary:

	UNIX	`psql`	`less`
start up	login with username and password	`psql -h mod-fund-databases`	is started by `psql` if output is too big for single screen
commands	things like: "`cd`" for "change directory" "`ls`" for "list all files"	1) SQL commands 2) "house keeping" commands (starting with `\`)	`<space>` for next page "`u`" for previous page
help facilities	`man` pages; call as `man <command-name>`	`\?`	`h`
how to exit	`exit`	`\q`	`q`