University of Sheffield

COM1001 SPRING SEMESTER

Professor Phil McMinn

p.mcminn@sheffield.ac.uk

SQL Databases

Transience vs Persistence

When a program terminates, the memory containing program data (i.e., values in variables) is **erased**. We say that this data is **transient**.

Most useful programs need certain data to **persist** outside of the application, so it can be re-loaded when the program is used again.

Web applications have a particular need for persistence – as soon as a user finishes using a web application, their session data is lost. Also, web applications may be responsible for managing a lot of data that is not feasible to load into memory for each HTTP request and response.

Commerce application:
Customer information, current orders, stock levels, etc.

Online learning environment:
Student logins and information, details of learning materials, assessments and marks.

One way to achieve persistence, as well as a means to efficiently store and retrieve information, is to store data in a database.

Databases

A database consists of **data** and **rules** pertaining to its organisation.

Access and modification of the data is handled by a **Database Management System (DBMS)**.

A DBMS can run as a server, accepting multiple connections at once involving requests for data and/or updates to that data.

Types of Database

Databases generally fall into two categories:

Relational

- Data is stored in **structured 2D tables**
- Require the use of a language called **SQL** to interact with the database
- Several decades of development has led to some very mature, fast and reliable DBMSs

Commercial:
Oracle, MS SQL Server

Free, Open Source:
MySQL, Postgres, SQLite

NoSQL

- Data can be of a form chosen by the DBMS (e.g., graphs, documents)
- SQL is *not* used (hence the name!) – the method of obtaining and updating data is DBMS-specific
- Less mature

MongoDB, Cassandra, HBase, Neo4J … and many many more

SQLite

Simple and self-contained, **little to no configuration** required

Often the choice for web developers for developing web applications

> For deployment, developers tend to prefer an enterprise database that is better optimised for heavy concurrent access, such as PostgreSQL

Forms the basis of many desktop and mobile applications

> e.g. Chrome and Safari, and numerous other well-known applications

Pre-installed on Mac OS and Linux.
For other types of OS see https://www.sqlite.org

Using SQLite

```
codio@north-mister:~/workspace$ sqlite3
SQLite version 3.22.0 2018-01-22 18:45:57
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
```

SQLite works as a server (more later), but we can interact with it directly at the Terminal too, using the `sqlite3` command

SQLite achieves persistence by writing **everything to a file.**

If we don't specify a file, it will work in an in-memory, transient mode. That means that everything will be lost when we quit the session.

SQLite achieves persistence by writing **everything to a file.**

If we don't specify a file, it will work in an in-memory, transient mode. That means that everything will be lost when we quit the session.

We can specify a file in two ways:

1. Supply a file name at the terminal:

```
codio@north-mister:~/workspace$ sqlite3 my_database.sqlite3
SQLite version 3.22.0 2018-01-22 18:45:57
Enter ".help" for usage hints.
sqlite>
```

2. Use the `.open` command in SQLite:

```
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open my_database.sqlite3
sqlite>
```

To exit, and get back to the shell, we use the `.quit` command:

```
sqlite> .quit
codio@north-mister:~/workspace$
```

SQLite is a Relational Database

Data in a relational database is organised into **tables**

Each **row** represents a **record** of information

Each **column** denotes a named **field** of the **record**

first_name	surname	gender	date_of_birth	country	position	club
Dominic	Calvert-Lewin	M	1997-03-16	England	Forward	Everton
Mary	Earps	F	1993-03-07	England	Goalkeeper	Manchester United
Harry	Kane	M	1993-07-28	England	Forward	Bayern Munich
Ashley	Lawrence	F	1995-06-11	Canada	Midfielder	Chelsea
Son	Heung-min	M	1992-07-08	South Korea	Forward	Tottenham Hotspur
Carpenter	Ellie	F	2000-04-28	Austrailia	Defender	Lyon
Bruno	Fernandes	M	1994-09-08	Portugal	Midfielder	Manchester United
Sam	Kerr	F	1993-09-10	Austrailia	Midfielder	Chelsea
Kevin	De Bruyne	M	1991-06-28	Belgium	Midfielder	Manchester City
Alexia	Putellas	F	1994-02-04	Spain	Midfielder	Barcelona
Jarrad	Braithwaite	M	2002-06-27	England	Defender	Everton
Lauren	James	F	2001-09-29	England	Forward	Chelsea

Each Table Requires a “Key”

Each table requires one or more columns whose rows will contain unique values, and so can uniquely identify a row. These column(s) are called the **primary key** of the table.

A primary key could be a name in table of people, for example, but often a name is not unique enough – two people may share the same name.

We could combine names with other pieces of information, e.g. a date of birth, but is also not guaranteed to be unique.

first_name	surname	gender	date_of_birth	country	position	club
Dominic	Calvert-Lewin	M	1997-03-16	England	Forward	Everton
Mary	Earps	F	1993-03-07	England	Goalkeeper	Manchester United
Harry	Kane	M	1993-07-28	England	Forward	Bayern Munich
Ashley	Lawrence	F	1995-06-11	Canada	Midfielder	Chelsea
Son	Heung-min	M	1992-07-08	South Korea	Forward	Tottenham Hotspur
Carpenter	Ellie	F	2000-04-28	Austrailia	Defender	Lyon
Bruno	Fernandes	M	1994-09-08	Portugal	Midfielder	Manchester United
Sam	Kerr	F	1993-09-10	Austrailia	Midfielder	Chelsea
Kevin	De Bruyne	M	1991-06-28	Belgium	Midfielder	Manchester City
Alexia	Putellas	F	1994-02-04	Spain	Midfielder	Barcelona
Jarrad	Braithwaite	M	2002-06-27	England	Defender	Everton
Lauren	James	F	2001-09-29	England	Forward	Chelsea

Try to think of some real world examples of this… (e.g., your UCard number)

Often, we just invent an ID number.

first_name	surname	gender	date_of_birth	country	position	club
Dominic	Calvert-Lewin	M	1997-03-16	England	Forward	Everton
Mary	Earps	F	1993-03-07	England	Goalkeeper	Manchester United
Harry	Kane	M	1993-07-28	England	Forward	Bayern Munich
Ashley	Lawrence	F	1995-06-11	Canada	Midfielder	Chelsea
Son	Heung-min	M	1992-07-08	South Korea	Forward	Tottenham Hotspur
Carpenter	Ellie	F	2000-04-28	Austrailia	Defender	Lyon
Bruno	Fernandes	M	1994-09-08	Portugal	Midfielder	Manchester United
Sam	Kerr	F	1993-09-10	Austrailia	Midfielder	Chelsea
Kevin	De Bruyne	M	1991-06-28	Belgium	Midfielder	Manchester City
Alexia	Putellas	F	1994-02-04	Spain	Midfielder	Barcelona
Jarrad	Braithwaite	M	2002-06-27	England	Defender	Everton
Lauren	James	F	2001-09-29	England	Forward	Chelsea

Creating a Table with SQL

We “talk” to relational databases using a language called **SQL** (**S**tructured **Q**uery **L**anguage)

All relational databases use SQL. However, each DBMS implements SQL slightly differently.

```
CREATE TABLE players (
    id INTEGER PRIMARY KEY,
    first_name TEXT,
    surname TEXT,
    gender TEXT,
    date_of_birth TEXT,
    country TEXT,
    position TEXT,
    club TEXT
);
```

Tables are created using `CREATE TABLE` SQL statements. Between the brackets of the `CREATE TABLE ... (...)` statement go the specifics of the table’s columns.

By convention, table names are plurals.

The `id` column is annotated with `PRIMARY KEY` to indicate it is such

Each entry for each column takes the form of the column’s name, followed by its type.

Each column’s information is separated by a comma.

SQL statements are terminated with a semicolon

Although SQLite is case insensitive, by convention SQL keywords appear in `UPPERCASE` and entity names in `lowercase`. Entity names are separated with `under_scores`.

Data Types in SQLite

```
CREATE TABLE players (
    id INTEGER PRIMARY KEY,
    first_name TEXT,
    surname TEXT,
    gender TEXT,
    date_of_birth TEXT,
    country TEXT,
    position TEXT,
    club TEXT
);
```

SQLite has **five** main data types:

`INTEGER`

`NUMERIC` — The difference between `NUMERIC` and `INTEGER` and `REAL` is subtle, and partly exists to maintain compatibility with other DBMSs. It's ignorable from the point of view of this module.

`REAL` — Floating point numbers

`TEXT` — Strings

`BLOB` — `BLOB` stands for Binary Large OBject. `BLOB` database fields store binary data like images and other types of document in a database. We won't be using them in this module.

What's missing?

`INTEGER`

`NUMERIC`

`REAL`

`TEXT`

`BLOB`

There is no `BOOLEAN` type.
We need to use an `INTEGER` instead, where `0` = `FALSE` and 1 = `TRUE`

SQLite does not have special types to manage date and time.
(This is in contrast to many other DBMSs.) Instead we must use the `TEXT` field. SQLite does have a number of built-in functions that can manipulate these date/time `TEXT` fields, but we have the option of using Ruby for that anyway.

The Database Schema

The tables, columns, types and other specifics like their primary keys are collectively referred to as the database's **schema**. More on database schemas later in the module.

SQLite will give us back the schema for our database, in SQL, if we use the `.schema` command:

```
sqlite> .schema
CREATE TABLE players (
    id INTEGER PRIMARY KEY,
    first_name TEXT,
    surname TEXT,
    gender TEXT,
    date_of_birth TEXT,
    country TEXT,
    position TEXT,
    club TEXT
);
sqlite>
```

The `.tables` command gives a list of tables:

```
sqlite> .tables
players
sqlite>
```

`.schema`, `.tables`, `.open`, and any command that starts with a period are special SQLite commands.

They are not part of SQL and will not necessarily work with other DBMSs.

Adding, Retrieving, Updating and Deleting Rows in a Table

Adding Rows. Rows are added to a table using the `INSERT` SQL statement as shown here

Retrieving Rows. `SELECT` queries return the records matching the clause following the `WHERE` keyword. The "*" here means "all columns". We can specify certain column names instead and get back partial records with only the field values for those columns.

Updating Rows. Similarly, we can update certain records using `UPDATE` as shown here, i.e. those that satisfy the `WHERE` clause.

Deleting Rows. And finally, we can delete certain records as shown here using the `DELETE` statement, also qualified using `WHERE`.

```
INSERT INTO players VALUES(1, "Dominic", "Calvert-Lewin",
                           "M", "1997-03-16", "England",
                           "Forward", "Everton");

SELECT * FROM players WHERE id = 1;

UPDATE players SET club = "Everton" WHERE surname = "Kane";

DELETE FROM players WHERE position = "Midfielder";
```

The Importance of WHERE

The `WHERE` clause is a predicate used to identify rows in the table to retrieve (using a `SELECT` query), **update** (using `UPDATE`) or **delete** (using `DELETE`).

Adding new rows (through `INSERT`) **does not require the identification of any existing rows**, so `WHERE` is *not used* with `INSERT`.

For any `SELECT`, `UPDATE` or `DELETE` statement, omitting the `WHERE` clause is equivalent to requesting all the records in the table.

So guess what happens with "`DELETE from players`"?

More complex WHERE queries

WHERE clauses can contain multiple conditions

```
WHERE club = "Manchester City" AND position = "Midfielder";

WHERE club = "Manchester City" OR club = "Manchester United"

WHERE club LIKE "%Manchester%"
```

LIKE is an operator for TEXT columns. It will return rows of the table where the value of a field matches the following specifier (i.e., “%Manchester%”)

The % symbols are wildcards – they can match any character. So this WHERE clause matches players with clubs with “Manchester” in their name.

WHERE club = “Manchester City” AND position = “Midfielder”

id	first_name	surname	gender	date_of_birth	country	position	club
1	Dominic	Calvert-Lewin	M	1997-03-16	England	Forward	Everton
2	Mary	Earps	F	1993-03-07	England	Goalkeeper	Manchester United
3	Harry	Kane	M	1993-07-28	England	Forward	Bayern Munich
4	Ashley	Lawrence	F	1995-06-11	Canada	Midfielder	Chelsea
5	Son	Heung-min	M	1992-07-08	South Korea	Forward	Tottenham Hotspur
6	Carpenter	Ellie	F	2000-04-28	Austrailia	Defender	Lyon
7	Bruno	Fernandes	M	1994-09-08	Portugal	Midfielder	Manchester United
8	Sam	Kerr	F	1993-09-10	Austrailia	Midfielder	Chelsea
9	Kevin	De Bruyne	M	1991-06-28	Belgium	Midfielder	Manchester City
10	Alexia	Putellas	F	1994-02-04	Spain	Midfielder	Barcelona
11	Jarrad	Braithwaite	M	2002-06-27	England	Defender	Everton
12	Lauren	James	F	2001-09-29	England	Forward	Chelsea

1 row retrieved, updated or deleted, depending on whether this WHERE is used in a SELECT, UPDATE or DELETE

`WHERE club = “Manchester City” OR club = “Manchester United”`

id	first_name	surname	gender	date_of_birth	country	position	club
1	Dominic	Calvert-Lewin	M	1997-03-16	England	Forward	Everton
2	Mary	Earps	F	1993-03-07	England	Goalkeeper	Manchester United
3	Harry	Kane	M	1993-07-28	England	Forward	Bayern Munich
4	Ashley	Lawrence	F	1995-06-11	Canada	Midfielder	Chelsea
5	Son	Heung-min	M	1992-07-08	South Korea	Forward	Tottenham Hotspur
6	Carpenter	Ellie	F	2000-04-28	Austrailia	Defender	Lyon
7	Bruno	Fernandes	M	1994-09-08	Portugal	Midfielder	Manchester United
8	Sam	Kerr	F	1993-09-10	Austrailia	Midfielder	Chelsea
9	Kevin	De Bruyne	M	1991-06-28	Belgium	Midfielder	Manchester City
10	Alexia	Putellas	F	1994-02-04	Spain	Midfielder	Barcelona
11	Jarrad	Braithwaite	M	2002-06-27	England	Defender	Everton
12	Lauren	James	F	2001-09-29	England	Forward	Chelsea

3 rows retrieved, updated or deleted, depending on whether this `WHERE` is used in a `SELECT`, `UPDATE` or `DELETE`

(Equivalent to result to `WHERE club LIKE “%Manchester%”`)

id	first_name	surname	gender	date_of_birth	country	position	club
1	Dominic	Calvert-Lewin	M	1997-03-16	England	Forward	Everton
2	Mary	Earps	F	1993-03-07	England	Goalkeeper	Manchester United
3	Harry	Kane	M	1993-07-28	England	Forward	Bayern Munich
4	Ashley	Lawrence	F	1995-06-11	Canada	Midfielder	Chelsea
5	Son	Heung-min	M	1992-07-08	South Korea	Forward	Tottenham Hotspur
6	Carpenter	Ellie	F	2000-04-28	Austrailia	Defender	Lyon
7	Bruno	Fernandes	M	1994-09-08	Portugal	Midfielder	Manchester United
8	Sam	Kerr	F	1993-09-10	Austrailia	Midfielder	Chelsea
9	Kevin	De Bruyne	M	1991-06-28	Belgium	Midfielder	Manchester City
10	Alexia	Putellas	F	1994-02-04	Spain	Midfielder	Barcelona
11	Jarrad	Braithwaite	M	2002-06-27	England	Defender	Everton
12	Lauren	James	F	2001-09-29	England	Forward	Chelsea

week3/football_players_example/football_players.sqlite3

Databases – Summary

Databases allow data to persist between and during user sessions of a web application, allowing data to be manipulated without having to load it all into memory at once.

- A database is managed by a **Database Management System (DBMS)**

SQL DBMSs store data in 2D tables.

- They use a language called **SQL** to insert, retrieve (“select”), update and delete data in them.

SQLite is an SQL DBMS that is easy to use.

- SQLite is the DBMS we’ll be using on this module.