Dr Tomasz Kurowski
t.j.kurowski@cranfield.ac.uk

# Database Design and Implementation

Following the database design discussion this morning, you are now going to implement the proposed microbial growth database using a relational database management system called **SQLite**. Unlike most traditional RDMSes, SQLite does not depend on a separate server application (which stores and manages the data) and client application (which remotely connects to the server), but it stores databases in local files, which can be accessed using a simple local executable – one could call it an "embedded" system. This makes it less suitable for databases meant to be shared by many users, but it will make your work easier as you learn the basics of what is otherwise a full-fledged relational database system.

## SQLite setup

For most uses SQLite does not require installation or significant setup. Simply download and extract the appropriate precompiled binaries for your system (Linux, Windows, or macOS), either from Canvas or the official website: https://sqlite.org/download.html

The official website is also a good place to look up the documentation, which you can consult later in the practical. The following pages should be particularly useful:

- Overview: https://sqlite.org/docs.html
- Command line shell: https://sqlite.org/cli.html
- SQL as understood by SQLite: https://sqlite.org/lang.html
- Datatypes: https://sqlite.org/datatype3.html
- Aggregate functions: https://sqlite.org/lang_aggfunc.html
- Core functions: https://sqlite.org/lang_corefunc.html
- Error codes: https://sqlite.org/rescode.html

## Running SQLite

You should be able to run the extracted binary (there will be several in the archive, but the one we will be using is called sqlite3) in the command line simply by typing in **sqlite3** (or **./sqlite3** from a local directory on Linux or macOS), followed by the name of a database file, for `example`:

```
sqlite3 practical.sqlite
```

The above will open a command line interface which can be used to interact with the database stored in a file named `practical.sqlite` (and create the file if it does not exist). You can then type in and execute both SQL commands (e.g. `CREATE DATABASE...`) as well as special "dot commands" which control the command line interface itself, such as the `.exit` command which will close it.

## Foreign Key enforcement - IMPORTANT

By default, SQLite is slightly more relaxed than most relational database management systems when it comes to enforcing foreign key constraints. As a result, it will allow users to violate those constraints, making it possible to create tables which are not - strictly speaking - correct. To avoid this, make sure to use this command every time you open your SQLite interface (or at the start of your SQL scripts):

```
PRAGMA foreign_keys = true;
```

This will enable strict foreign key constraint enforcement.

## Example: Student marks

You may remember a fun practical from the Java module which required you to implement an application for storing and retrieving student marks. Just to try out basic SQL syntax, we will create something similar as a relational database. Open the `sqlite3` command line interface (name your database file something like `student_marks.sqlite`) and create a table for storing our student data:

```
CREATE TABLE students (
    student_id TEXT,
    student_name TEXT,
    PRIMARY KEY(student_id)
);
```

You can verify that the table was created by using the `.tables` command, which will list all the tables in the current database. You can also have a look at its schema (it should be the same as your command) by using the `.schema` command. Both of those are useful for investigating database contents. The table could be deleted by using the `DROP TABLE students` command, but let us try to populate it instead:

```
INSERT INTO students VALUES('s802060', 'Tomasz Kurowski');
```

or, more explicitly:

```
INSERT INTO students (student_id, student_name)
VALUES('s802060', 'Tomasz Kurowski');
```

The latter approach is useful in case you want to omit certain fields. For now, insert several more students into the database. (Note that you cannot insert the same student twice – the primary key has to be unique!) You can then display the student data by using the SELECT command:

```
SELECT * FROM students;
```

or, more specifically, if you want to display only student names:

```
SELECT student_name FROM students;
```

You can use the aggregator function COUNT to count the number of students in the database:

```
SELECT COUNT(*) FROM students;
```

Now let's create another table, which will store marks achieved by each student:

```
CREATE TABLE marks (
    student_id TEXT,
    module TEXT,
    mark INTEGER,
    PRIMARY KEY (student _id, module),
     FOREIGN KEY (student_id) REFERENCES students (student_id)
);
```

Populate the table with marks. Once you have entered several, you can use the SELECT command and aggregator functions to investigate the contents of your table. Can you tell what the following commands will return?

```
SELECT AVG(mark) FROM marks WHERE module = 'Java';
SELECT AVG(mark) FROM marks WHERE student_id = 's802060';
```

Notice that the `marks` table alone is insufficient to provide you with certain information, for example the name of a student who achieved the highest mark in the Java module. To do that we need to JOIN the `student` table with the `marks` table:

```
SELECT student_name, MAX(mark), module
FROM students
JOIN marks ON students.student_id = marks.student_id
WHERE module = 'Java';
```

Note that we are JOINing the two tables where their student_ids (their foreign keys) match. You can also UPDATE values in a table, as show below:

```
UPDATE marks SET mark = 100
WHERE student_id = 's802060' AND module = 'Java';
```

Try SELECTing a few other metrics of your choice until you feel comfortable with the basic commands. You can then move on to the main tasks today – implementing our microbial database!

## SQL scripts

While typing in commands one at a time into a command line interface can be useful, particularly for exploratory purposes, for tasks such as creating a database it is usually better to create a script containing all your commands to be executed all at once instead. For the task ahead, create a text file (you can use the editor of your choice, e.g. Notepad++ on Windows machines) with the `.sql` extension, for example `practical.sql`. Place all your SQL commands inside that file (make sure to include `PRAGMA foreign_ keys = true;`) and execute it using the following commands:

- On Linux / maxOS: `./sqlite3 practical.sqlite < practical.sql`
- On Windows: `sqlite3 practical.sqlite ".read practical.sql"`

You can also execute a script from within the SQLite command line interface by using the `.read` command.

## Creating a microbial growth database

Your main task today will be creating an SQL script which will contain all the statements needed to create the database tables we designed for our microbial growth data this morning.

The database we designed includes five tables (names in bold) listed below along with their attributes.

(PK – Primary Key, FK – Foreign Key)

**1 experiments**

- experiment_id
- organism
- medium
- temperature
- PK: experiment_id
- FK: organism

**2 datapoints**

- experiment_id
- time
- cfu
- PK: (experiment_id, time)
- FK: experiment_id

**3 authors**

- author_id
- name
- PK: author_ id

**4 organisms**

- organism
- is_fungus
- PK: organism

**5 experiments_authors**

- author_id
- experiment_id
- PK:(author_id, experiment_id)
- FK: author_id
- FK: experiment_id

Based on your own judgement, the SQLite documentation, the original data spreadsheet and legend (both on Canvas), assign a type to each attribute in the database before creating new tables.

Below is a statement which can be used to create the **datapoints** table. Use this as an example of how to form the SQL statements for the remaining four tables:

```
CREATE TABLE IF NOT EXISTS datapoints (
     experiment_id TEXT,
     time REAL NOT NULL,
     cfu REAL NOT NULL,
     PRIMARY KEY (experiment _id, time),
     FOREIGN KEY (experiment_id)
          REFERENCES experiments (experiment_ID)
);
```

**Important!**

1. The tables should be created (and more importantly: populated) in a specific order due to references.

2. Mark every attribute which should never be empty with **NOT NULL**

3. Foreign Keys and the attribute they reference must have the same type.

Try to manually populate each of the database tables with a few rows from the microbial data spreadsheet on Canvas to test your constraints.

### Side note – GUI tools

There are many free visual/GUI based database design tools which can be used to facilitate and ease the development of databases. One such tool is SQLite workbench (https://www.sqlite-workbench.com/), a free Web-based application which allows you to upload your SQLite database file and interact with it, as well as create diagrams of your database – this sort of visualisation is very useful for technical documentation! Try it.

If you prefer a desktop-based tool, DB Browser for SQLite, available at https://sqlitebrowser.org/ will allow you to work on your database files on your own machine. It may also be useful for populating your database.

Another such tool is MySQL Workbench, a more advanced stand-alone application, developed by Oracle. Although it does not directly support SQLite (it can be done via plug-ins), it allows you to design your database schema by dragging and dropping tables, specifying column types, all in a graphical environment providing a diagrammatic view of your database. You may find it a practical tool to use later on (for example, during the group project), so we would recommend you have a look at it as well if you have the time. Its ER diagram visualisation options are particularly powerful.

More detailed information about MySQL Workbench can be found here: https://www.mysql.com/products/workbench/

### Advanced task: Populating the database

Manually populating a database would be very tedious. Manually populating a database would be very tedious. Many programming languages provide you with means of connecting to relational databases. For example, if you've done the Java module, you will be familiar with the OpenCSV library, which you could use together with a library called JDBC (Java Database Connectivity) to read a CSV file and populate SQLite database (see the "Populating the database using Java" section below for an example). In Python, you could use the csv and sqlite3 modules (both part of the standard library). There are multiple options in JavaScript. You could use a sophisticated parsing package like fast-csv (https://www.npmjs.com/package/fast-csv), but for a simple example we can use a synchronous library called csv-load-sync, alongside the better-sqlite3 package that will let us connect to our database. You can install both using `npm install`. Make sure to have a look at their documentation:

Csv-load-sync: https://www.npmjs.com/package/csv-load-sync

Better-sqlite3: https://www.npmjs.com/package/better-sqlite3

A simple script that will insert datapoints into our database could look like this:

```
// Load dependencies
const better_sqlite3 = require('better-sqlite3');
const { load } = require('csv-load-sync');
// Input CSV file
const input_csv_file = 'microbial_growth_data.csv';
// Open database
const db = better_sqlite3(input_db_file);
// Load all rows (synchronously)
const rows = load(input_csv_file);
// Loop over each row
rows.forEach((row) => {
    // Create query template
const qry= 'INSERT INTO datapoints (experiment_id,time,cfu) VALUES (?,?,?)';
    // Run query
    db.prepare(qry).run(row['Experiment'],row['Time'], row['CFU']);
});
// Close database
db.close();
```

Note that inserting the datapoints is not enough – you will also need to insert authors, organisms, experiments, etc.! Entering your entire dataset is comprises the first "extra task" at the end of this practical.

## (Optional) Populating the database using Java – JDBC

If you want to use Java to populate your database, the simplest approach would be to use the JDBC API. You can find its documentation at: https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/

As JDBC is not dependent on any specific RDBMS, it requires a driver specific to your database of choice. There is no official driver for SQLite, but an unofficial one is available from https://github.com/xerial/sqlite-jdbc.

The driver file has also been uploaded to Canvas. You need to add it to your project's libraries; in Netbeans this can be achieved by right-clicking **Libraries** in your project tab, then selecting it through **Add JAR/Folder...**

The example code below demonstrates a simple SELECT query on an SQLite database stored in a file named `practical.sqlite`, then loops over the results and prints them out.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class JavaJDBC {
    public static void main(String[] args) {
        try {
          String url = "jdbc:sqlite:C:/Users/e802060/practical.sqlite";
            String query = "SELECT experiment_id, organism FROM experiments;";
            Connection conn = DriverManager.getConnection(url);
            try (Statement stmt = conn.createStatement()) {
                ResultSet result = stmt.executeQuery(query);
                while (result.next()) {
                     System.out.println("EXPERIMENT ID: " + result.getString(1));
                     System.out.println("ORGANISM: " + result.getString(2));
                }
            }
        conn.close();
        } catch (SQLException ex) {
            System.out.println("Could not connect to the database!");
        }
    }
}
```

## Extra Task 1

Convert the microbial growth data spreadsheet file available on Canvas (microbial_growth_data_2025.xls) into a CSV file (this can be done using Excel) and implement a script which will parse the microbial growth data and INSERT it into the database that you have created.

## Extra Task 2

Create a script or application which asks the user for an experiment id, then prints out information about the experiment (the authors, medium used, temperature used, and all the data points).