Databases — Part 1

The objective of this sequence of works is to understand the concept of databases and to handle it with the SQL language: we will manipulate this language thanks to the `sqlite3` module in Python.

We use databases all the time, but never manipulate them directly. There is usually an interface between you and the database. For example there is a database of all the students, teachers, etc. in the school. But you don't ask directly to the database what is your timetable or what are the students that follow a given course that you also follow. You get this information thanks to your teachers, or thanks to a software (or an application).

As a second example, you might be interested in knowing how to go to Leuven by train. Then, you go on https://www.belgiantrain.be/en, and here, the application asks you where your journey starts, where it ends, and when you want it to happen. For example you can select a journey from Uccle Vivier d'Oie (just next to the school) to Leuven, and the application will tell you that there is a direct S train. But you do not write a SQL request. What happens is (1) the application transforms what you typed into a SQL request, (2) the application gets the answer from this request and (3) the application shows the possible answers thanks to a "user-friendly" interface. Here is a possible scenario:

1. The application asks us: "Departure from".
2. We answer "Uccle Vivier d'Oie"
3. The application asks us: "Arrival at".
4. We answer "Leuven".
5. The application sends a SQL request to the database:

```
SELECT timeStart, timeStop FROM trains
WHERE stationStart = "Uccle Vivier d'Oie" AND stationStop = "Leuven";
```

6. The database gives the answers $\{(15\text{h}47, 17\text{h}22); (18\text{h}21, 19\text{h}58)\}$
7. The application prints on our screen:

```
Results for the journey Uccle Vivier d'Oie -> Leuven
First train  : 15h47  ->  17h22
Second train : 18h21  ->  19h58
```

Before examining what is a SQL request... we have to understand what is a database. In this work, we will examine a very simple database: a library, which manages books, and can lend them to its users (the Platon library for instance). As a simple example, we will work on a library containing 2 copies of "L'avare", 3 copies of "Les fleurs du mal", 1 copy of "Jane Eyre" and 2 copies of "King Lear". Of course, to be able to search books in the library, we need all the information about those books. We will thus create a table containing different fields, see Table 1. Each row of our table will be a book, and each column will show the values of each field for the current book.

We see that we have a problem: we can't distinguish the different books we have. We have seen in last works that we must be able to refer to each item in our data uniquely. We will thus add identifiers for each of our books: an integer. All the identifiers for our books will be different (and will be numbers from 1 to the number of books we have): this field is the primary key for our books, see Table 2. For the sake of simplicity, in our works, the primary key will always be a natural number.

Now, of course, we want to be able to lend those books. Will we add a column to each row, telling who currently has the book? It is possible, see Table 3, but this is a bad idea.

Author	Title	Genre
Molière	L'avare	Theater
Molière	L'avare	Theater
Baudelaire	Les fleurs du mal	Poetry
Baudelaire	Les fleurs du mal	Poetry
Baudelaire	Les fleurs du mal	Poetry
Brontë	Jane Eyre	Novel
Shakespeare	King Lear	Theater
Shakespeare	King Lear	Theater

Table 1: Our 8 books.

Identifier	Author	Title	Genre
1	Molière	L'avare	Theater
2	Molière	L'avare	Theater
3	Baudelaire	Les fleurs du mal	Poetry
4	Baudelaire	Les fleurs du mal	Poetry
5	Baudelaire	Les fleurs du mal	Poetry
6	Brontë	Jane Eyre	Novel
7	Shakespeare	King Lear	Theater
8	Shakespeare	King Lear	Theater

Table 2: Our unique 8 books.

Id.	Author	Title	Genre	Borrower	From
1	Molière	L'avare	Theater	Bob	07/01/2024
2	Molière	L'avare	Theater	—	—
3	Baudelaire	Les fleurs du mal	Poetry	—	—
4	Baudelaire	Les fleurs du mal	Poetry	Alice	12/02/2024
5	Baudelaire	Les fleurs du mal	Poetry	—	—
6	Brontë	Jane Eyre	Novel	—	—
7	Shakespeare	King Lear	Theater	—	—
8	Shakespeare	King Lear	Theater	—	—

Table 3: Our unique 8 books with borrowings.

If our database grows, having everything in the same table will lead to requests that will take a lot of time, especially if we want to keep track of the history (requests will search through a big table, although it is mostly filled with empty cells, see Table 4). It is thus a good idea to structure our data by adding a second table for our users and a third table for the borrowings, see Tables 5 and 6.

Id.	Author	Title	Genre	Borrower1	From1	To1	Borrower2	From2	To2	...
...	...	...	...	...	...	...	...	...	...	...

Table 4: Our books with multiple borrowings.

Identifier	Surname	First name
1	Smith	Alice
2	Johnson	Bob

Table 5: Our users.

Identifier	Borrower id.	Book id.	From	To
1	2	1	07/01/2024	—
2	1	4	12/02/2024	—

Table 6: Our borrowings.

This borrowing table (Table 6) transforms our database into a relational database. It is impossible to understand this table without the two others. For each borrowing:

- to know who is the borrower, we need the user table (Table 5), and
- to know what is the book borrowed, we need the book table (Table 2).

Exercise 1

Here is the database of another library containing four tables (Tables 7, 8, 9 and 10).

Identifier	Surname	First name
1	Smith	Alice
2	Johnson	Bob
3	Taylor	Charles
4	Davies	Diana

Table 7: Exercise 1: The 4 users.

Identifier	Author	Title	Genre id.
1	Baudelaire	Les fleurs du mal	1
2	Steinbeck	The Grapes of Wrath	2
3	Molière	L'avare	3
4	Corneille	Le Cid	3
5	Hugo	Les misérables	2
6	Prévert	Paroles	1

Table 8: Exercise 1: The 6 books.

Identifier	Genre
1	Poetry
2	Novel
3	Theater

Table 9: Exercise 1: The 3 literary genres.

Identifier	Borrower id.	Book id.	From	To
1	4	2	01/01/2024	—
2	1	4	02/01/2024	08/01/2024
3	4	5	04/01/2024	16/01/2024

Table 10: Exercise 1: The 3 borrowings.

1. How can we interpret the fact that the "To" field for the 1st borrowing is empty?
2. What is the title of the 4th book?
3. What is the genre of the 5th book?
4. Who is the borrower in the 2nd borrowing? What book was borrowed?
5. List the genres of the borrowed books.

Exercise 2

What structure would you build to handle a database for the students at EEB1? Give the list of tables you would use, with the different fields, and the different relations between your tables.

Exercise 3

Uccle's communal council wants to realize a website so that each Uccloise and each Ucclois may send a message to the communal house to give feedback about road problems they might encounter. Each person must be able to create an account on the website (identifier, name, password), and must be able to write a message. The communal council must be able to read the information and must have an account for that.

What structure would you build to handle a database for this task? Give the list of tables you would use, with the different fields, and the different relations between your tables.

Appendix: insight into the SQL language

Here is a possible sequence of SQL commands to create the tables of exercise 1. We will see in the next works how to execute those commands in Python with `sqlite3`.

```
CREATE TABLE IF NOT EXISTS users(
    identifier INTEGER PRIMARY KEY,
    surname TEXT NOT NULL,
    first_name TEXT NOT NULL
);
INSERT INTO users(surname,first_name) VALUES ('Smith', 'Alice');
INSERT INTO users(surname,first_name) VALUES ('Johnson', 'Bob');
INSERT INTO users(surname,first_name) VALUES ('Taylor', 'Charles');
INSERT INTO users(surname,first_name) VALUES ('Davies', 'Diana');

CREATE TABLE IF NOT EXISTS genres(
    identifier INTEGER PRIMARY KEY,
    genre TEXT NOT NULL
);
INSERT INTO genres(genre) VALUES ('Poetry');
INSERT INTO genres(genre) VALUES ('Novel');
INSERT INTO genres(genre) VALUES ('Theater');

CREATE TABLE IF NOT EXISTS books(
    identifier INTEGER PRIMARY KEY,
    author TEXT NOT NULL,
    title TEXT NOT NULL,
    genre_id INTEGER NOT NULL,
    CONSTRAINT fk_genre FOREIGN KEY(genre_id) REFERENCES genres(identifier)
);
...

CREATE TABLE IF NOT EXISTS borrowings(
    identifier INTEGER PRIMARY KEY,
    borrower_id INTEGER NOT NULL,
    book_id INTEGER NOT NULL,
    date_from DATE NOT NULL,
    date_to DATE,
    CONSTRAINT fk_borrower_id FOREIGN KEY(borrower_id) REFERENCES users(identifier
        ),
    CONSTRAINT fk_book_id FOREIGN KEY(book_id) REFERENCES books(identifier)
);
...
```

Almost everything speaks for itself, except one feature: when typing `INTEGER PRIMARY KEY`, `sqlite3` will automatically manage the ids in ascending order, starting at 1. Thus, you do not need to manage the identifiers, and you do not need to make sure that they are all different: the library does it for you. You can see that when inserting, you do not specify the id value.

Remark: you could also use the SQL keyword `AUTO_INCREMENT`, but this does not work in `sqlite3`.

Here are a few commands that can be executed to get information from those tables:

```
SELECT surname, first_name FROM users;
```

This request will give us the names of all users (without their identifier).

```
SELECT * FROM books WHERE genre_id=2;
```

This request will give us the list of all novels.

And of course it is also possible to modify some rows of those tables:

```
UPDATE borrowings SET date_to='2024-02-23' WHERE identifier=1;
```

This request will update the first borrowing to tell that the book was given back today.