Introduction to databases

Relevance and background

What does this course cover?

- Overview of [relational] databases and DBMSs
- SQL language for
  - retrieving data
  - updating data
  - inserting data
  - deleting data
  - creating databases and tables
- Managing integrity using constraints
- Programmatically connecting to and using a database using Java
- Testing integrity and acceptance of [bad] data
- Basic database administration

What is a database (system)?

- Organised and persistent collection of data
- We use a Database management system (DBMS) to access the data

What can we do with a DBMS?

- Retrieve data in a structured way
- Insert new data and delete old data
- Update existing data
- Create new tables (schemas) for future data

Why do we need a database?

- We don't want to manage data and persistens ourselves
- A database does the job faster, better and more securely
- We want to separate code from data
- We might want centralised data for a distributed system
- We might want data integrity
- We might want transactions (commit and rollback)
- A database offers a standardised interface to the data, e.g. SQL

Note: Sometimes we use the word "database" to refer to a database management system

What would a database look like?

Imagine we have a collection of books which we'd like to build an application around. The application should accept new books, be able to search books and delete books.

A simple database could have a table for books:

Author	Title	ISBN	Publisher
John Smith	Life	0-0-0-0-0-1	Bonnier
James Woody	Love	0-0-0-0-0-2	Bonnier
Joan Carmen	Guns	0-0-0-0-0-3	Bonnier
Johnanna Boyd	Code	0-0-0-0-0-4	Bonnier
Eva Peron	Cars	0-0-0-0-0-5	Books R us

The books database

Having the table books in a database allows us to search for books, update books, delete books or insert new books.

This can be done in a standardised way using the SQL query language.

Our application can talk to the database using SQL over an API.

The data about books are now independent of our application.

The database knows nothing about who or what is talking to it.

We can thus easily switch from i.e. a command line interface to some other client for our application.

We need to learn some SQL for our database

Next lecture we'll look at SQL for retrieving data ("selecting data").

Teaser:

What do you think this SQL statement would retrieve?

```
SELECT author, title FROM books
 WHERE publisher = 'Bonnier';
```

Terms and concepts

Database - A set of tables for storing data in an organised way

Database Management System (DBMS) - A set of computer programs that controls the creation, maintenance and usage of the database

SQL/Structured Query Language - A Language for Working with a DBMS

Table - A set of related data in a database

Note: Sometimes we say "database" for short when we actually talk about a database management system

Retrieving data

SQL SELECT

What we will look at in this lecture

- How is data organised in a database?
- What is a table?
- What are types?
- `SELECT` statement (SQL)
- `WHERE` clause - specifying criteria
- `ORDER BY`
- `*` (wildcard for “all columns”)
- Boolean expressions with `AND`, `OR`, comparisons
- `LIKE`

What we will not learn

How to solve Sudokos using SQL and SELECT

```
WITH RECURSIVE
  input(sud) AS (
    VALUES('53..7....6..195....98....6.8...6...34..8.3..17...2...6.6....28....419..5....8..79')
  ),
  digits(z, lp) AS (
    VALUES('1', 1)
    UNION ALL SELECT
    CAST(lp+1 AS TEXT), lp+1 FROM digits WHERE lp<9
  ),
  x(s, ind) AS (
    SELECT sud, instr(sud, '.') FROM input
    UNION ALL
    SELECT
      substr(s, 1, ind-1) || z || substr(s, ind+1), instr( substr(s, 1, ind-1) || z || substr(s, ind+1), '.' )
     FROM x, digits AS z
    WHERE ind>0
      AND NOT EXISTS (
            SELECT 1
              FROM digits AS lp
             WHERE z.z = substr(s, ((ind-1)/9)*9 + lp, 1)
                OR z.z = substr(s, ((ind-1)%9) + (lp-1)*9 + 1, 1)
                OR z.z = substr(s, (((ind-1)/3) % 3) * 3 + ((ind-1)/27) * 27 + lp + ((lp-1) / 3) * 6, 1)
          )
  )
SELECT s FROM x WHERE ind=0; /* works in sqlite version > 8.2 */
```

Organisation of data

In order to understand how to fetch data, we need to know how data is organised in a database.

Data are organised in a database in something called tables. In our example database there is currently one table called books.

Data in two or more tables can be related. For instance, a table can hold information that is further described in a second table.

We will look at that when we talk about normalisation.

For now, we have only one single table.

Organisation of data inside a table

A table has rows of data. A row contains data in fields (fields are also known as columns). Fields have types and can have modifiers (or constraints).

Our simple book table is arranged as such:

```
author TEXT, title TEXT, isbn TEXT PRIMARY KEY, publisher TEXT
```

It means that each row has the following fields and types (and modifiers)

```
fieldname    type     modifier
author       TEXT
title        TEXT
isbn         TEXT     PRIMARY KEY
publisher    TEXT
```

Field types

You can read about the type system of SQLite3 here: https://www.sqlite.org/datatype3.html

Basically we can think of these different types:

`INTEGER`

`REAL`

`TEXT`

`BLOB`

As a reference/comparison the PostgreSQL types are here: http://www.postgresql.org/docs/9.0/static/datatype.html

Other types that would be handy

We'd also like to have the concepts of Boolean, Date and Time.

SQLite provides functions for the dates and times but uses integers for the actual data. Boolean is used by SQLite using the integers 1 for true and 0 for false.

Modifiers/Constraints

A field can have either a value of a type or the special value `NULL`.

If we don't want to permit `NULL` values we can use the modifier `NOT NULL` which will cause a constraint violation if someone tries to insert the value `NULL` for such a field.

`PRIMARY KEY` is a (kind of) constraint for a unique column. All primary key values must be different from each other.

We'll get back to constraints in a later lecture.

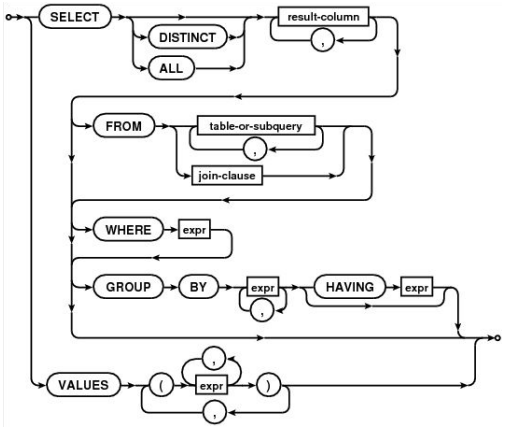
SELECT

In order to retrieve data using SQL, we use the SELECT statement. It has the following structure in its simplest form:

```
SELECT column[,column]* FROM table [WHERE criteria];
```

For instance, returning to our books example, we could do:

```
sqlite> SELECT title FROM books;
Life
Love
Guns
Code
Cars
```



SELECT more than one field/column

We can retrieve more than one field:

```
sqlite> SELECT author, title FROM books;
John Smith|Life
James Woody|Love
Joan Carmen|Guns
Johnanna Boyd|Code
Eva Peron|Cars
```

The WHERE clause

Often we want to specify a criteria for the data we want to retrieve. What if we only want the books that have “Bonnier” for publisher?

```
sqlite> SELECT author, title FROM books WHERE publisher='Bonnier';
John Smith|Life
James Woody|Love
Joan Carmen|Guns
Johnanna Boyd|Code
```

Headers

Warning: SQLite specific!

We can include the column names of our result by setting the headers flag:

```
sqlite> .headers on
sqlite> SELECT author, title FROM books WHERE publisher='Bonnier';
author|title
John Smith|Life
James Woody|Love
Joan Carmen|Guns
Johnanna Boyd|Code
sqlite>
```

Format output

Warning: SQLite specific!

```
sqlite> .mode column
sqlite> SELECT author, title FROM books WHERE publisher='Bonnier';
author      title
----------  ----------
John Smith  Life
James Wood  Love
Joan Carme  Guns
Johnanna B  Code
sqlite>
```

More formatting

Warning! SQLite specific!

```
sqlite> .width 14 14
sqlite> SELECT author, title FROM books WHERE publisher='Bonnier';
author          title
--------------  --------------
John Smith      Life
James Woody     Love
Joan Carmen     Guns
Johnanna Boyd   Code
sqlite>
```

Ordering of the result

We can add an ordering clause at the end of our query like so:

```
sqlite> SELECT author, title FROM books WHERE publisher='Bonnier'
   ...> ORDER BY title;
author          title
--------------  --------------
Johnanna Boyd   Code
Joan Carmen     Guns
John Smith      Life
James Woody     Love
sqlite>
```

Reverse order

```
sqlite> SELECT author, title FROM books WHERE publisher='Bonnier'
   ...> ORDER BY title DESC;
author          title
--------------  --------------
James Woody     Love
John Smith      Life
Joan Carmen     Guns
Johnanna Boyd   Code
sqlite>
```

SELECT all fields

Sometimes we want to fetch all columns from a table. We can use * in order to say “all columns”.

```
sqlite> SELECT * FROM books;
author          title           isbn         publisher
--------------  --------------  -----------  ----------
John Smith      Life            0-0-0-0-0-1  Bonnier
James Woody     Love            0-0-0-0-0-2  Bonnier
Joan Carmen     Guns            0-0-0-0-0-3  Bonnier
Johnanna Boyd   Code            0-0-0-0-0-4  Bonnier
Eva Peron       Cars            0-0-0-0-0-5  Books R us
sqlite>
```

Boolean expressions

We can create criterias using AND, OR combined with <, >, =

```
sqlite> SELECT * FROM books WHERE publisher='Books R us' OR title='Life';
author          title           isbn         publisher
--------------  --------------  -----------  ----------
John Smith      Life            0-0-0-0-0-1  Bonnier
Eva Peron       Cars            0-0-0-0-0-5  Books R us

sqlite> SELECT * FROM books WHERE publisher='Bonnier' AND
        isbn  > '0-0-0-0-0-2';
author          title           isbn         publisher
--------------  --------------  -----------  ----------
Joan Carmen     Guns            0-0-0-0-0-3  Bonnier
Johnanna Boyd   Code            0-0-0-0-0-4  Bonnier
```

Like it or not

We can select text according to a substring expression using LIKE:

```
sqlite> SELECT * FROM books WHERE title LIKE "c%";
author          title           isbn         publisher
--------------  --------------  -----------  ----------
Johnanna Boyd   Code            0-0-0-0-0-4  Bonnier
Eva Peron       Cars            0-0-0-0-0-5  Books R us


sqlite> SELECT * FROM books WHERE author LIKE "jo%";
author          title           isbn         publisher
--------------  --------------  -----------  ----------
John Smith      Life            0-0-0-0-0-1  Bonnier
Joan Carmen     Guns            0-0-0-0-0-3  Bonnier
Johnanna Boyd   Code            0-0-0-0-0-4  Bonnier
```

A few words about LIKE

In SQLite the like expression is case insensitive*. In PostgreSQL it is not (PostgreSQL has the `ILIKE` operator for case insensitive matching).

There is a performance hit when using `LIKE`. There are much faster matching mechanisms in various databases.

* By default, SQLite3 uses case insensitive LIKE for the ASCII charset. Note that

```
SELECT "ö" LIKE "Ö";
```

returns 0 (false) because it doesn't treat ÅÄÖ case-insentitively.

Summary

A database consists of tables. Tables have rows. Each row has columns with name and type. Data is inserted into a table row by row.

To retrieve data from a table we use SELECT. The basic form is:

```
SELECT <column>[,<column>]* FROM <table> [WHERE <condition>];
```

For instance:

```
SELECT author, title FROM books WHERE title = 'Cars';
```

The condition can be complex, e.g. `title='Cars' AND publisher='Bonnier';`

What's up next?

Next lecture, we'll look at the SQLite3 database and how to get started with it.

If you want to prepare, install SQLite3 on your computer.

Ubuntu:

```
sudo apt-get install sqlite3
```

MacOS:

Follow the instructions here (make sure to install at least version 3):

http://www.tutorialspoint.com/sqlite/sqlite_installation.htm

Read

http://www.w3schools.com/sql/sql_select.asp

http://zetcode.com/db/sqlite/select/

https://en.wikibooks.org/wiki/Structured_Query_Language/Snippets#Basic_Syntax (The basic syntax is enough for this course, and note that you have to refer to the SQLite3 manual to see what constructs are supported!)

https://www.sqlite.org/lang_select.html (in particular SELECT_CORE)

SQL WHERE clause

Where everybody knows your name

Boolean expressions

In SQL, an expression has a value and a type (or it's `null`, representing no value and no type).

When we want to make a selection of rows according to some criteria, we use a Boolean expression. Such an expression is a claim about the world which is either True or False.

In SQLite3 True is represented by 1 (one) and False is represented by 0 (zero).

This expression is typically used in the "`WHERE` clause" of an SQL statement.

Criteria for a SELECT

A simple form of a SELECT statement can be expressed as:

SELECT <* or comma separated list of column names>

FROM <tablename> [WHERE <Boolean expression>];

Example:
`SELECT title, author FROM book WHERE publisher_id = 3;`

column list: **`title, author`**
table name: **`book`**
Boolean expression: **`publisher_id = 3`**

Compound Boolean expressions

The Boolean expression for a WHERE clause can be a simple "predicate" (Boolean expression), but often is a compound expression with logical operators:

```
sqlite> SELECT make, color, license_number FROM cars
......> WHERE color IN ('Blue', 'Black', 'Brown')
......>   AND make = 'Dodge'
......>   AND license_number LIKE 'N%';
Dodge|Brown|NGR 230
Dodge|Blue|NCT 331
Dodge|Brown|NZN 420
Dodge|Brown|NQL 251
Dodge|Black|NEZ 751
```

Compound Boolean expressions

The basic Boolean operators are:

- AND (both operands need to evaluate to True)
- OR (*at least one* operand needs to evaluate to True)

```
sqlite> SELECT make, color, license_number FROM cars
......> WHERE make = 'Dodge' AND license_number LIKE 'N%';
sqlite> SELECT make, color, license_number FROM cars
......> WHERE make = 'Dodge' OR make = 'Volvo';
```

Tests (operands) always need to be a complete Boolean Expression:

```
sqlite> SELECT make, color, license_number FROM cars
......> WHERE make = 'Dodge' OR 'Volvo';
```

Boolean column type

You can use the Boolean type for a column:

```
CREATE TABLE author(author_id INTEGER PRIMARY KEY NOT NULL,
                    name TEXT,
                    got_nobel_prize BOOLEAN DEFAULT 0 NOT NULL);

sqlite> SELECT name FROM author WHERE got_nobel_prize;
Selma Lagerlöf
sqlite> SELECT name FROM author WHERE NOT got_nobel_prize;
Henrik and Rikard
```

As you see, you don't need to compare a BOOLEAN value to True or False

In SQLite3, the BOOLEAN type will be treated as NUMERIC
(0 used for false, 1 used for true)

Operators which produce Boolean values

The following operators produce a `BOOLEAN` value:

- `=  <  >  <=  >=  IS  <>  !=`
- `IN`
- `BETWEEN`
- `AND  OR`
- `NOT`
- `LIKE  GLOB`
- `CASE-WHEN-THEN-ELSE-END`
- `HAVING`

Some examples - CASE-WHEN-ELSE-END

```
sqlite> SELECT name ||
......> CASE WHEN got_nobel_prize THEN
......>  ' (winner)'
......> ELSE
......>  ' (loser)'
......> END
......> FROM author;
Henrik and Rikard (loser)
Selma Lagerlöf (winner)
```

The || operator concatenates text (in SQLite).

Some examples - BETWEEN

```
sqlite> SELECT title FROM book WHERE title BETWEEN 'D' AND 'K';
Java direkt med Swing
Databasteknik

sqlite> SELECT title FROM book WHERE
   ...> title BETWEEN 'Databasteknik' AND 'Java direkt med Swing';
Java direkt med Swing
Databasteknik

sqlite> SELECT publisher_id, name FROM publisher WHERE
   ...> publisher_id BETWEEN 2 AND 4;
2|Juneday
3|Mayday! Mayday!
4|Oh Really
```

Some examples - HAVING

```
sqlite> SELECT count(*) AS number_of_titles, name AS publisher FROM book
NATURAL JOIN publisher GROUP BY publisher;
number_of_titles      publisher
--------------------  --------------------
1                     Juneday
2                     Studentlitteratur
sqlite> SELECT count(*) AS number_of_titles, name AS publisher FROM book
NATURAL JOIN publisher GROUP BY publisher HAVING number_of_titles > 1;
number_of_titles      publisher
--------------------  --------------------
2                     Studentlitteratur
```

You cannot use WHERE on an aggregate value.

SQL Extra lecture

CREATE TABLE ‹tablename›...

Creating a new table

Since a database consists of one or more tables, how is a table created in the first place?

You have actually seen examples of that inside all files provided for the exercises. But we have never looked more closely at the so called CREATE TABLE statement.

We will not examine this at the exam, but for the curious student, here's an extra lecture with the basics about creating tables.

Simplest form of a create table statement

```
CREATE TABLE [IF NOT EXISTS] <table_name>
     (<column_name> <type> [modifier] [,<column_name> <type> [modifier]]*);
```

For instance:

```
CREATE TABLE IF NOT EXISTS cars (car_id INTEGER,
                                 make TEXT,
                                 license_num TEXT PRIMARY KEY);
```

Used once - but make it accessible

The create table statement(s) is of course run only once, when you set up the database initially. But why not save that statement in a file, so that you later on easily can re-create the table (or tables)?

When you make a dump of a table (or database) the SQL statements created for you always start with the create table statement for each table.

Without it, the dump wouldn't work, since it would only contain insert statements but without any guarantee that the table to insert into exists...

Read more about the create statement

The full story can be read here:

https://www.sqlite.org/lang_createtable.html

Bonus:

If you'd like to try it in MySQL/MariaDB:

https://mariadb.com/kb/en/mariadb/create-table/

PostgreSQL:

https://www.postgresql.org/docs/9.6/static/sql-createtable.html

Getting started

sqlite3

Recap of L02

A database consists of tables. Tables have rows. Each row has columns with name and type. Data is inserted into a table row by row.

To retrieve data from a table we use SELECT. The basic form is:

```
SELECT <column>[,<column>]* FROM <table> [WHERE <condition>];
```

For instance:

```
SELECT author, title FROM books WHERE title = 'Cars';
```

The condition can be complex, e.g. `title='Cars' AND publisher='Bonnier';`

Install sqlite3

Ubuntu:
```
sudo apt-get install sqlite3
```

Cygwin:
Use the cygwin installer and select the sqlite3 package sqlite3-3.9.2-1 or later.

MacOS:
Follow the instructions here:
http://www.tutorialspoint.com/sqlite/sqlite_installation.htm

Create a database for your user

You can start the sqlite3 shell by typing:

```
sqlite3 my_books
```

This will create the database my_books. You will run sqlite3 as your UNIX user, so there is no need to create any users inside the database.

Note that this will create the database as a file named my_books in the current directory (the directory where you ran the command line).

Note: If you use a more complex dbms such as PostgreSQL or MySQL, it's a little more complicated to set up users and rights.

Your user may now create a new database table

```
sqlite> create table books(author TEXT, title TEXT,
                           isbn TEXT PRIMARY KEY,
                           publisher TEXT);
```

OK, I'm set up. Whachamadowithit?

Now, after creating a database, you can either connect to it (while inside the sqlite3 interactive shell):

```
sqlite> .open my_books
```

...Or you can use computer shell to connect to your database:

```
rikard@ggslaptop:~$ sqlite3 my_books
SQLite version 3.8.2 2013-12-06 14:53:30
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite>
```

Accessing the database from the command line

It is very convenient to be able to connect to a database directly from the command line.

Now we can actually script SQL commands (as the user) and get results directly! E.g.:

```
$ echo "SELECT author, title FROM books WHERE publisher='Bonnier';" | sqlite3 my_books
John Smith|Life
James Woody|Love
Joan Carmen|Guns
Johnanna Boyd|Code
```

(the command is issued on one single line)

Investigating a table

```
sqlite> .schema books
CREATE TABLE books(author TEXT, title TEXT,
                   isbn TEXT PRIMARY KEY, publisher TEXT);
sqlite>
```

Listing tables in database

```
sqlite> .tables
books

sqlite>
```

Provided you have either opened the database using

```
sqlite3 my_books
```

or opened the database from within sqlite3

```
.open my_books
```

What's up next?

Review the SELECT (retrieving data) lecture and practise SELECT statements on the my_books database (provided by the teacher).

The next lecture on SQL will focus on UPDATE (changing data in a table)

The my_books database

Create a textfile called my_books.sql with the following content:

```
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE IF NOT EXISTS books(author TEXT, title TEXT, isbn TEXT PRIMARY
KEY, publisher TEXT);
INSERT INTO "books" VALUES('John Smith','Life','0-0-0-0-0-1','Bonnier');
INSERT INTO "books" VALUES('James Woody','Love','0-0-0-0-0-2','Bonnier');
INSERT INTO "books" VALUES('Joan Carmen','Guns','0-0-0-0-0-3','Bonnier');
INSERT INTO "books" VALUES('Johnanna Boyd','Code','0-0-0-0-0-4','Bonnier');
INSERT INTO "books" VALUES('Eva Peron','Cars','0-0-0-0-0-5','Books R us');
COMMIT;
```

Load the my_books.sql into SQLite3

In the same directory as the my_books.sql file do the following:

```
sqlite3 my_books < my_books.sql
```

It means: create a new database called `my_books` using the SQL statements in the file called `my_books.sql`

Login to the database using:

```
sqlite3 my_books
```

Read

https://www.sqlite.org/cli.html

http://zetcode.com/db/sqlite/introduction/

http://zetcode.com/db/sqlite/tool/

Getting started

postgresql

Install postgresql 9

Ubuntu:

```
sudo apt-get install postgresql-9.4
```

MacOS:

Follow the instructions here:

http://www.postgresql.org/download/macosx/

Create a user

You need to tell postgres that your pc user can login to the database.

Ubuntu:

```
sudo su postgres # you need to be user 'postgres'
```

```
createuser -d -l <your-user-name> # in order to run this
```

The latter command creates a user with "create table" privileges, who also may login as the unix user with the same name.

Create a user

Wait, what, what, now?

```
sudo su postgres
```

You run the "switch user" command as your super user, in order to "become" the postgres user (who has privileges to do stuff in the database)

```
createuser -d -l <your-user-name>
```

As "postgres" you run the command to create a user with "your name". The flags mean "with create table privileges" and "with right to login" from the shell (with your unix user name as the database user name).

What was all this for?

Now, you have a user in postgres with the same name as your UNIX user name. You may login to postgres with the psql command without having to specify a database user name (it will now accept the default which is your UNIX user name):

```
rikard@ggslaptop:~$ psql postgres
psql (9.3.10)
Type "help" for help.

postgres=>
```

Your user may now create a new database table

```
CREATE TABLE books(author TEXT, title TEXT, isbn TEXT
PRIMARY KEY, publisher TEXT);
```

OK, I'm set up. Whachamadowithit?

Now, after creating a database, you can either connect to it (while still inside the psql interactive shell):

```
postgres=> \c my_books;
You are now connected to database "my_books" as user "rikard".
my_books=>
```

...Or you can logout, and from the shell connect to your database:

```
rikard@ggslaptop:~$ psql my_books rikard
psql (9.3.10)
Type "help" for help.

my_books=>
```

Why all the fuzz?

It is very convenient to have a UNIX user being able to connect to a database it has privileges for, directly from the command line (or environment) without having to being asked for password...

Now we can actually script SQL commands (as the user) and get results directly! E.g.:

```
rikard@ggslaptop:~$ psql my_books rikard -c "SELECT author, title FROM books WHERE
Publisher='Bonnier';"
    author     | title
---------------+-------
 John Smith    | Life
 James Woody   | Love
 Joan Carmen   | Guns
 Johnanna Boyd | Code
(4 rows)
```

Investigating a table

```
my_books=> \d books;
     Table "public.books"
  Column   | Type | Modifiers
-----------+------+-----------
 author    | text |
 title     | text |
 isbn      | text | not null
 publisher | text |
Indexes:
    "books_pkey" PRIMARY KEY, btree (isbn)
```

Listing tables in database

```
my_books=> \dt
        List of relations
 Schema | Name  | Type  | Owner
--------+-------+-------+--------
 public | books | table | rikard
(1 row)
```

Getting started with MariaDB

Crash course with examples

Creating a user

We'll assume that you have the root user's password. This is how you create a new user:

```
$ sudo mysql -uroot -p
[sudo] password for rikard:     # ← sudo password
Enter password:                 # ← mysql root password
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 73
Server version: 10.0.29-MariaDB-0ubuntu0.16.04.1 Ubuntu 16.04

Copyright (c) 2000, 2016, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

Creating a user

While in the interactive shell, create a user, e.g. “phpuser”

```
MariaDB [(none)]> CREATE USER 'phpuser'@'localhost' IDENTIFIED BY 'somepassword';
Query OK, 0 rows affected (0.00 sec)

MariaDB [(none)]>
```

Creating a user

Next, create a database

```
MariaDB [(none)]> CREATE DATABASE my_reddit;
Query OK, 1 row affected (0.00 sec)

MariaDB [(none)]>
```

Creating a user

Next, assign privileges to the user ‘phpuser’

```
MariaDB [(none)]> GRANT SELECT, INSERT, UPDATE ON my_reddit.* TO
'phpuser'@'localhost';
Query OK, 0 rows affected (0.00 sec)

MariaDB [(none)]>
```

If your user will be the php/apache user on your system, it might be wise to limit the number of privileges to the minimum necessary for your application.

Creating a table in the my_reddit database

Next, let’s create a simple table in the my_reddit database, and add some data:

```
MariaDB [(none)]> USE my_reddit;
Database changed

MariaDB [my_reddit]> CREATE TABLE links(id int(11) NOT NULL AUEMENT PRIMARY KEY,
                                        url varchar(2083) NOT NULL);
Query OK, 0 rows affected (0.04 sec)

MariaDB [my_reddit]> INSERT INTO links(url) VALUES('http://wiki.juneday.se');
Query OK, 1 row affected (0.00 sec)

MariaDB [my_reddit]>
```

Let's try to use our user from the command line

Let's see if we can retrieve the data from my_reddit.links from the command line:

```
$ echo 'SELECT * FROM links;' | mysql -uphpuser -psomepassword my_reddit
id   url
1    http://wiki.juneday.se
$
```

That seemed to work fine!

Testing the database from PHP

Make sure you have installed:

- PHP
- Apache2
- libapache2-mod-php7.0 (or similar)
- php7.0-pgsql (or other supplier of PDO for database access from PHP)

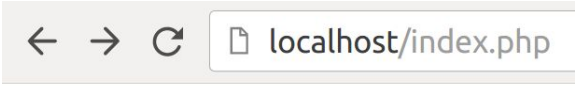
Testing the database from PHP

Example PHP file (e.g. /var/www/html/index.php):

```
<!DOCTYPE html>
<html><head><title>Testing MariaDB from Database</title></head>
<body>
<?php
$servername = "localhost"; $username = "phpuser"; $password = "somepassword"; $dbname = "my_reddit";
$url = "mysql:host=" . $servername . ";dbname=" . $dbname;
try {
    $dbh = new PDO($url, $username, $password);
    foreach($dbh->query('SELECT url from links') as $row) {
        $url = $row["url"];
        echo "<a href=\"" . $url . "\">" . $url . "</a>" . "<br>";
    }
    $dbh = null;
} catch (PDOException $e) {
    print "Error!: " . $e->getMessage() . "<br/>"; die();
}
?>
</body></html>                                    <!--Source: http://php.net/manual/en/pdo.connections.php -->
```

Testing the database from PHP

Open your browser and go to http://127.0.01/index.php (if that's what you called your file was!)



http://wiki.juneday.se

Source: http://php.net/manual/en/pdo.connections.php

You can run PHP as a server in the command line

```
 $ ls
index.php
$ php -S localhost:8000
PHP 7.0.18-0ubuntu0.16.04.1 Development Server started at
Thu Jul  6 09:48:53 2017
Listening on http://localhost:8000
Document root is /home/rikard/programming/php/cli-test
Press Ctrl-C to quit.
[Thu Jul  6 09:48:59 2017] 127.0.0.1:59238 [200]: /
```

Some more PDO examples

```
try {
  $dbh = new PDO($url, $username, $password);
  print "Got connection: " .
      $dbh->getAttribute(constant("PDO::ATTR_CONNECTION_STATUS")) .
      " Server version: " .
      $dbh->getAttribute(constant("PDO::ATTR_SERVER_VERSION")) .
      "<br />";
  foreach($dbh->query('SELECT url from links') as $row) {
      $url = $row["url"];
      echo "<a href=\"" . $url . "\">" . $url . "</a>" . "<br>";
  }
  $dbh = null;
} catch (PDOException $e) {
  print "Error!: " . $e->getMessage() . "<br/>"; die();
}
```

Some handy MariaDB/MySQL commands

To log in to the interactive shell of a database:

```
$ mysql -u phpuser -p my_reddit
Enter password:
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 79
Server version: 10.0.29-MariaDB-0ubuntu0.16.04.1 Ubuntu 16.04

Copyright (c) 2000, 2016, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [my_reddit]>
```

Some handy MariaDB/MySQL commands

To see the tables of a database:

```
MariaDB [my_reddit]> SHOW TABLES;
+---------------------+
| Tables_in_my_reddit |
+---------------------+
| links               |
+---------------------+
1 row in set (0.00 sec)

MariaDB [my_reddit]>
```

Some handy MariaDB/MySQL commands

To investigate a table:

```
MariaDB [my_reddit]> DESCRIBE links;
+-------+---------------+------+-----+---------+----------------+
| Field | Type          | Null | Key | Default | Extra          |
+-------+---------------+------+-----+---------+----------------+
| id    | int(11)       | NO   | PRI | NULL    | auto_increment |
| url   | varchar(2083) | NO   |     | NULL    |                |
+-------+---------------+------+-----+---------+----------------+
2 rows in set (0.00 sec)

MariaDB [my_reddit]>
```

Some handy MariaDB/MySQL commands

To investigate a table:

```
MariaDB [my_reddit]> DESC links;
+-------+---------------+------+-----+---------+----------------+
| Field | Type          | Null | Key | Default | Extra          |
+-------+---------------+------+-----+---------+----------------+
| id    | int(11)       | NO   | PRI | NULL    | auto_increment |
| url   | varchar(2083) | NO   |     | NULL    |                |
+-------+---------------+------+-----+---------+----------------+
2 rows in set (0.00 sec)

MariaDB [my_reddit]>
```

Some handy MariaDB/MySQL commands

To backup a database:

```
$ sudo mysqldump -uroot -p my_reddit > my_reddit_backup.sql
```

To restore a database:

```
$ sudo mysql -uroot -p my_reddit < my_reddit_backup.sql
```

Comparison between postgresql, mysql and sqlite

Here's a good syntax and usage comparison between three dbms:s:

http://hyperpolyglot.org/db

If you are familiar with postgresql or sqlite, then that link could be of great use. Regardless, it is useful to see that there are (sometimes subtle) differences between the syntax and workings of different systems.

Changing data

SQL UPDATE

Recap L03

Create database from command line (if the database doesn't exist):

```
$ sqlite3 database_name
```

Start sqlite3 with a database that exists:

```
$ sqlite3 database_name
```

Run commands from a file without starting the interactive shell:

```
$ sqlite3 database_name < file_with_commands
OR:
$ cat file_with_commands | sqlite3 database_name
```

Recap L03 SELECT

```
SELECT <column>[,<column>]* FROM <table> [WHERE <condition>];
```

For instance:

```
SELECT author, title FROM books WHERE title = 'Cars';
```

The condition can be complex, e.g.

```
title='Cars' AND publisher='Bonnier';
```

Data changes

Data often changes so we have to have a way to update the data in our database.

This lecture will guide you through the basics of the UPDATE statement in SQL.

Let's look at a table of cars

```
sqlite> .schema cars
CREATE TABLE cars (make TEXT, color TEXT, licensenumber TEXT PRIMARY KEY);

sqlite> select * from cars;
make        color       license
----------  ----------  -------
Volvo       Green       ABC 123
Honda       Blue        ABC 124
Porsche     Green       BBC 666
Ferrari     Red         FST 667
```

If you want your own cars database, here's how

Put the following in a file called cars.sql

```
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE cars (make TEXT, color TEXT, licensenumber TEXT PRIMARY KEY);
INSERT INTO "cars" VALUES('Volvo','Green','ABC 123');
INSERT INTO "cars" VALUES('Honda','Blue','ABC 124');
INSERT INTO "cars" VALUES('Porsche','Green','BBC 666');
INSERT INTO "cars" VALUES('Ferrari','Red','FST 667');
COMMIT;
```

Then execute the following command in the same directory as the file:

```
sqlite3 my_cars < cars.sql
```

Open the database with `sqlite3 my_cars`

The UPDATE statement

The basic form of the UPDATE SQL statement is:

```
UPDATE <table> SET <col>=<value>[,<col>=<value>]* WHERE <criteria>;
```

The WHERE clause is fairly important. If we leave it out, all rows will be changed!

Let's change the color of the Porsche to Yellow:

```
UPDATE cars SET color='Yellow' WHERE licensenumber='BBC 666';
```

Confirm:

```
sqlite> SELECT * FROM cars WHERE licensenumber='BBC 666';
Porsche     Yellow      BBC 666
```

Updating more than one column

You can update more than one column:

```
UPDATE cars SET color='Grey', licensenumber='AAA 111'

   WHERE licensenumber='ABC 123';
```

(Change both color and license number of the Volvo)

```
sqlite> select * from cars;
Volvo       Grey        AAA 111
Honda       Blue        ABC 124
Porsche     Yellow      BBC 666
Ferrari     Red         FST 667
```

Read

http://www.w3schools.com/sql/sql_update.asp

http://zetcode.com/db/sqlite/datamanipulation/

What's next?

Next, we will look at how we would DELETE data from a table.

Deleting data

SQL DELETE

Deleting some data from a table

In a database, the data may have to be deleted. For instance, the data could be wrong or simply outdated. If we don't need any trace of the data ever being in the table, we use the SQL DELETE statement.

```
DELETE FROM <table> WHERE <col>=<value>;
```

As with the UPDATE statement, it is imperative to include the WHERE clause, since leaving it out would delete all rows from the table.

Revisiting the cars table

Remember our car database?

```
sqlite> select * from cars;
make        color       license
----------  ----------  -------
Volvo       Grey        AAA 111
Honda       Blue        ABC 124
Porsche     Yellow      BBC 666
Ferrari     Red         FST 667
sqlite>
```

We want to delete and forget all about the Honda. Let's do it!

```
DELETE FROM cars WHERE licensenumber = 'ABC 124';
```

Why did we choose LicenseNumber for criteria?

The LicenseNumber (as the observant student has noticed) is `PRIMARY KEY`. Remember that a property for `PRIMARY KEY` columns was that their values must be unique.

If we had several cars whose `color` was Blue, then the following would have deleted every blue car in the table:

```
DELETE FROM cars WHERE color='Blue';
```

Once again, be very cautious when performing `UPDATE` and `DELETE` statements!

Read

http://www.w3schools.com/sql/sql_delete.asp

What's next?

Now that we know how to fetch, update and delete data, we're going to look at how to `INSERT` new data into a table.

Adding new data

SQL INSERT

When new data arrives

Data is produced very quickly. It is therefore not uncommon to store new data in a database table. It could be a new Customer registering, or a new Order being placed. Or as with our example databases, a new Book is added to the books table or a new Car is added to the Cars table.

With SQL this is performed using the INSERT INTO statement:

```
INSERT INTO <table> (<col>[,<col>]*) VALUES (<val>[,<val>]*);
```

Example:

```
INSERT INTO cars (make, color, licensenumber)
    VALUES ('Berlingo','Red','HES 000');
```

The order matters

Let's look at the previous INSERT INTO statement again:

```
INSERT INTO cars (make, color, licensenumber)
    VALUES ('Berlingo','Red','HES 000');
```

We listed the column names and then the values in the corresponding order.

If you have very large tables (with many columns), this is of course a source for errors if you get mixed up concerning the order of the columns/values.

Read

http://www.w3schools.com/sql/sql_insert.asp

What's next?

Now that we know how to retrieve (`SELECT`), change (`UPDATE`), remove (`DELETE`), and add (`INSERT`) data, we are going to look at a more realistic data model involving more than one table.

This is called normalisation in a fancy language but we'll simply call it "decomposing large tables" and "combining (joining) data from linked tables".

Referencing data from linked tables

Normalised data - SQL JOIN

Why the need of more tables?

If we look at the structure of our previous example table books, we might notice that the table is self-contained and in a sense flat. All the data is contained in the same table.

This leads to duplication of data. For instance, the Publisher Bonnier is duplicated many times.

Books table

```
sqlite> select * from books;
author           title       isbn          publisher
---------------  ----------  ------------  ----------
John Smith       Life        0-0-0-0-0-0-1  Bonnier
James Woody      Love        0-0-0-0-0-0-2  Bonnier
Joan Carmen      Guns        0-0-0-0-0-0-3  Bonnier
Johnanna Boyd    Code        0-0-0-0-0-0-4  Bonnier
Eva Peron        Cars        0-0-0-0-0-0-5  Books R us
```

What's the problem with duplication of data?

One problem with duplication is obviously storage. Another one could be efficiency (it takes longer to search strings than e.g. integers). Another problem is to do with consistency.

What if we add a book but misspell Bonnier as Bonier?

Then we'd have inconsistency because we think of it as published by Bonnier but the database will consider it as published by "Bonier" instead (a completely different beast).

Fixing the duplication of Publisher

Let's assume that there is a second table alongside books, which is called publishers:

```
sqlite> .schema publishers
CREATE TABLE publishers (publisher_id INTEGER PRIMARY KEY, name TEXT);
sqlite> SELECT * FROM publishers;
publisher_id     name
---------------  ----------
1                Bonnier
2                Books R us
```

Consistency achieved!

Now we have only one canonical publisher called Bonnier, which is good.

But how do we reference the `publishers` table from the `books` table?

Let's change books to have it reference publisher

What if we, instead of storing the publishers' names in the table, stored the ID of the publisher from the publishers table?

```
sqlite> select * from books;
author           title       isbn           publisherid
---------------  ----------  -------------  -----------
John Smith       Life        0-0-0-0-0-0-1  1
James Woody      Love        0-0-0-0-0-0-2  1
Joan Carmen      Guns        0-0-0-0-0-0-3  1
Johnanna Boyd    Code        0-0-0-0-0-0-4  1
Eva Peron        Cars        0-0-0-0-0-0-5  2
```

For reference, how did I create that table?

```
sqlite> CREATE TABLE books2(author TEXT, title TEXT,
  isbn TEXT PRIMARY KEY, publisherid INTEGER);
sqlite> INSERT INTO books2 (author, title, isbn)
  SELECT author, title, isbn FROM books;
sqlite> UPDATE books2 SET publisherid = 1 WHERE isbn < '0-0-0-0-0-5';
sqlite> UPDATE books2 SET publisherid = 2 WHERE isbn = '0-0-0-0-0-5';
sqlite> DROP TABLE books;
sqlite> ALTER TABLE books2 RENAME TO books;
```

How to list the publishers' names?

But if we now have this:

```
sqlite> select * from books;
author           title       isbn         publisherid
---------------  ----------  -----------  -----------
John Smith       Life        0-0-0-0-0-1  1
James Woody      Love        0-0-0-0-0-2  1
Joan Carmen      Guns        0-0-0-0-0-3  1
Johnanna Boyd    Code        0-0-0-0-0-4  1
Eva Peron        Cars        0-0-0-0-0-5  2
```

How do we list all books including the publisher names?

Making the connection

This is one way of listing all books with their corresponding publishers:

```
sqlite> SELECT books.author, books.title, books.isbn, publishers.name
  FROM books, publishers WHERE books.publisherid = publishers.publisher_id;
author           title       isbn         name
---------------  ----------  -----------  ----------
John Smith       Life        0-0-0-0-0-1  Bonnier
James Woody      Love        0-0-0-0-0-2  Bonnier
Joan Carmen      Guns        0-0-0-0-0-3  Bonnier
Johnanna Boyd    Code        0-0-0-0-0-4  Bonnier
Eva Peron        Cars        0-0-0-0-0-5  Books R us
sqlite>
```

But we are not pleased with the header "name".

Fixing the column header

We can use aliases using the SQL operator AS:

```
sqlite> SELECT books.author, books.title, books.isbn,
             publishers.name  AS Publisher
          FROM books, publishers
         WHERE books.publisherid = publishers.publisher_id;
author           title       isbn         Publisher
---------------  ----------  -----------  ----------
John Smith       Life        0-0-0-0-0-1  Bonnier
James Woody      Love        0-0-0-0-0-2  Bonnier
Joan Carmen      Guns        0-0-0-0-0-3  Bonnier
Johnanna Boyd    Code        0-0-0-0-0-4  Bonnier
Eva Peron        Cars        0-0-0-0-0-5  Books R us
sqlite>
```

Using aliases

We can shorten the previous statement using aliases:

```
sqlite> SELECT b.author, b.title, b.isbn,
               p.name AS Publisher
          FROM books b, publishers p
         WHERE b.publisherid = p.publisher_id;
author           title       isbn         publisher
---------------  ----------  -----------  ----------
John Smith       Life        0-0-0-0-0-1  Bonnier
James Woody      Love        0-0-0-0-0-2  Bonnier
Joan Carmen      Guns        0-0-0-0-0-3  Bonnier
Johnanna Boyd    Code        0-0-0-0-0-4  Bonnier
Eva Peron        Cars        0-0-0-0-0-5  Books R us
sqlite>
```

Using SQL JOIN

There is an alternative syntax to use, called JOIN. We think it is good if you have seen it:

```
sqlite> SELECT author, title, isbn, name AS Publisher
          FROM books
          JOIN publishers
            ON books.publisherid = publishers.publisher_id;
author           title       isbn         Publisher
---------------  ----------  -----------  ----------
John Smith       Life        0-0-0-0-0-1  Bonnier
James Woody      Love        0-0-0-0-0-2  Bonnier
Joan Carmen      Guns        0-0-0-0-0-3  Bonnier
Johnanna Boyd    Code        0-0-0-0-0-4  Bonnier
Eva Peron        Cars        0-0-0-0-0-5  Books R us
sqlite>
```

Summary

We moved the Publisher name to its own table consisting of `publisher_id` and `name`.

We changed the `books` table to include an id from the `publishers` table rather than the publisher name itself.

We saw how we can join two tables in a `SELECT` query so that we could see all books and their respective publisher name, connecting the `publisherid` of the `books` table with the corresponding `publisher_id` of the `publishers` table in order to get the name of the publisher.

Read

http://zetcode.com/db/sqlite/joins/

http://www.w3schools.com/sql/sql_join_left.asp

http://www.w3schools.com/sql/sql_alias.asp

http://www.w3schools.com/sql/sql_join.asp

https://en.wikipedia.org/wiki/Database_normalization

Decomposing a table

Splitting up a table into tables - towards normalization

Too much information

Consider a table for books:

```
sqlite> .schema book
CREATE TABLE book(title TEXT, author TEXT, publisher TEXT,
                  isbn TEXT PRIMARY KEY NOT NULL);

sqlite> SELECT * from book;
title                author               publisher            isbn
-------------------  -------------------  -------------------  ------------
Databases            Manda Tory           IT-Literature Inc    0000001
SQL                  Manda Tory           IT-Literature Inc    0000002
Java                 D. Fault Values      IT-Literature Inc    0000003
More Java            M.T. Objects         IT-Literature Inc    0000004
Swedish Flowers      Lynn E. Uss          Biology Books AB     0000005
Java direkt med Swin Jan Skansholm        Studentlitteratur    978914410431
Databasteknik        Thomas Padron-Mccart Studentlitteratur    978914404449
Programming in Java  Henrik and Rikard    Juneday              1234567
```

Too much information

- What if we want to store information about a new publisher (without any books yet)?
- What if we stop selling some books from a publisher, what to do?

```
sqlite> .schema book
CREATE TABLE book(title TEXT, author TEXT, publisher TEXT,
                  isbn TEXT PRIMARY KEY NOT NULL);

sqlite> INSERT INTO book(....???); -- add new publisher
sqlite> UPDATE book SET ....???;   -- remove books but keep publisher
```

Duplicate information

- How is the publisher IT Literature Inc stored?

```
sqlite> SELECT * FROM book WHERE publisher = 'IT-Literature Inc';
title                author               publisher            isbn
-------------------  -------------------  -------------------  ------------
Databases            Manda Tory           IT-Literature Inc    0000001
SQL                  Manda Tory           IT-Literature Inc    0000002
Java                 D. Fault Values      IT-Literature Inc    0000003
More Java            M.T. Objects         IT-Literature Inc    0000004

-- The publisher's name is repeated four times...
```

Rule of thumb

- One type of “thing” per table - The book table should be about books
- One such thing per row - One book per row
- One row per thing - The “publisher” is actually in four rows...

```
title               author              publisher            isbn
------------------  ------------------  -------------------  ------------
Databases           Manda Tory          IT-Literature Inc    0000001
SQL                 Manda Tory          IT-Literature Inc    0000002
Java                D. Fault Values     IT-Literature Inc    0000003
More Java           M.T. Objects        IT-Literature Inc    0000004
```

See for instance http://www.databasteknik.se/webbkursen/normalisering/ (Swedish)

We can’t store new publishers without books

- One type of “thing” per table - The book table contains also publishers
- One such thing per row
- One row per thing - A publisher needs at least one book...

```
title               author              publisher            isbn
------------------  ------------------  -------------------  ------------
Databases           Manda Tory          IT-Literature Inc    0000001
SQL                 Manda Tory          IT-Literature Inc    0000002
Java                D. Fault Values     IT-Literature Inc    0000003
More Java           M.T. Objects        IT-Literature Inc    0000004
                                        Infological Books Ltd
```

We need a table for publishers, if we want to be able to store information about publishers apart from books

We have to spell publisher’s name correctly

- Adding more than one book requires careful spelling of the publisher

```
title               author              publisher            isbn
------------------  ------------------  -------------------  ------------
Databases           Manda Tory          IT-Literature Inc    0000001
SQL                 Manda Tory          IT-Literature Inc    0000002
Java                D. Fault Values     IT-Literature Inc    0000003
More Java           M.T. Objects        IT-Litterature Inc   0000004    --OOPS!
                                        Infological Books Ltd
```

We need a table for publishers, to save the name about a publisher on one row (and spell it correctly once)

Create a publisher table

- One type of “thing” per table - The table should be about publishers
- One such thing per row - One publisher per row
- One row per thing - One publisher per row as well
- We can store/keep publishers without books!

```
CREATE TABLE publisher(publisher_id INTEGER PRIMARY KEY NOT NULL,
                       name TEXT UNIQUE NOT NULL);
sqlite> SELECT * FROM publisher;
publisher_id        name
------------------  -------------------
1                   Studentlitteratur
2                   Juneday
3                   Mayday! Mayday!
4                   Oh Really
5                   IT-Literature Inc
6                   Biology Books AB
```

Change the book table - reference publisher

- A book still “has” a publisher, only we refer to it by the ID of the other tbl
- The publisher_id column is called a “foreign key”
- The column in the foreign table must have unique values - Why?

```
CREATE TABLE book (title TEXT, author TEXT, isbn TEXT PRIMARY KEY NOT NULL,
                   publisher_id INTEGER);
sqlite> SELECT * FROM book;
title                author                isbn           publisher_id
-------------------  --------------------  -------------  ------------
Java direkt med Swing  Jan Skansholm         9789144104317  1
Databasteknik        Thomas Padron-Mccarthy  9789144044491  1
Programming in Java  Henrik and Rikard     1234567        2
Swedish Flowers      Lynn E. Uss           0000005        6
More Java            M.T. Objects          0000004        5
Java                 D. Fault Values       0000003        5
SQL                  Manda Tory            0000002        5
Databases            Manda Tory            0000001        5
```

Retrieving book info including publisher

- We have to JOIN the two tables
- Both tables have column “publisher_id” - NATURAL JOIN is possible

```
sqlite> SELECT title, author, isbn, name AS publisher FROM
   ...> book NATURAL JOIN publisher;
title                author                isbn           publisher
-------------------  --------------------  -------------  ----------------------
Java direkt med Swing  Jan Skansholm         9789144104317  Studentlitteratur
Databasteknik        Thomas Padron-Mccarthy  9789144044491  Studentlitteratur
Programming in Java  Henrik and Rikard     1234567        Juneday
Swedish Flowers      Lynn E. Uss           0000005        Biology Books AB
More Java            M.T. Objects          0000004        IT-Literature Inc
Java                 D. Fault Values       0000003        IT-Literature Inc
SQL                  Manda Tory            0000002        IT-Literature Inc
Databases            Manda Tory            0000001        IT-Literature Inc
```

We still have duplicate authors

- We do the same with the author column - move it to its own table
- Authors “know” nothing about books - one thing per table, per row

```
CREATE TABLE author(author_id INTEGER PRIMARY KEY NOT NULL, name TEXT NOT NULL);

sqlite> SELECT * FROM author;
author_id            name
-------------------  --------------------
1                    Jan Skansholm
2                    Thomas Padron-Mccarthy
3                    Henrik and Rikard
4                    Lynn E. Uss
5                    M.T. Objects
6                    D. Fault Values
7                    Manda Tory
8                    Selma Lagerlöf
```

Change book table - refer also to author table

- author → author_id
- author_id FK in book, must be unique in author table

```
CREATE TABLE book(title TEXT, author_id INTEGER, isbn TEXT PRIMARY KEY NOT NULL,
                  publisher_id INTEGER);

sqlite> SELECT * FROM book;
title                author_id            isbn           publisher_id
-------------------  --------------------  -------------  ------------
Java direkt med Swing  1                    9789144104317  1
Databasteknik        2                    9789144044491  1
Programming in Java  3                    1234567        2
Swedish Flowers      4                    0000005        6
More Java            5                    0000004        5
Java                 6                    0000003        5
SQL                  7                    0000002        5
Databases            7                    0000001        5
```

Combining the three tables

- author → author_id, publisher → publisher_id
- author_id, publisher_id FKs in book, must be unique in other tables

```
CREATE TABLE book(title TEXT, author_id INTEGER, isbn TEXT PRIMARY KEY NOT NULL,
                  publisher_id INTEGER);

sqlite> SELECT title, author.name AS author, isbn, publisher.name AS publisher FROM
   ...> book NATURAL JOIN author JOIN publisher
   ...> ON publisher.publisher_id = book.publisher_id;
title                author                isbn           publisher
-------------------  --------------------  -------------  ----------------------
Java direkt med Swing  Jan Skansholm         9789144104317  Studentlitteratur
Databasteknik        Thomas Padron-Mccarthy  9789144044491  Studentlitteratur
Programming in Java  Henrik and Rikard      1234567        Juneday
Swedish Flowers      Lynn E. Uss           0000005        Biology Books AB
More Java            M.T. Objects          0000004        IT-Literature Inc
Java                 D. Fault Values       0000003        IT-Literature Inc
SQL                  Manda Tory            0000002        IT-Literature Inc
Databases            Manda Tory            0000001        IT-Literature Inc
```

Combining the three tables (alternative version)

- author → author_id
- author_id FK in book, must be unique in author table

```
CREATE TABLE book(title TEXT, author_id INTEGER, isbn TEXT PRIMARY KEY NOT NULL,
                  publisher_id INTEGER);

sqlite> SELECT title, isbn, publisher, author FROM book
   ...> NATURAL JOIN (SELECT publisher_id, name AS publisher from publisher)
   ...> NATURAL JOIN (SELECT author_id, name AS author FROM author);
title                author                isbn           publisher
-------------------  --------------------  -------------  ----------------------
Java direkt med Swing  Jan Skansholm         9789144104317  Studentlitteratur
Databasteknik        Thomas Padron-Mccarthy  9789144044491  Studentlitteratur
Programming in Java  Henrik and Rikard      1234567        Juneday
Swedish Flowers      Lynn E. Uss           0000005        Biology Books AB
More Java            M.T. Objects          0000004        IT-Literature Inc
Java                 D. Fault Values       0000003        IT-Literature Inc
SQL                  Manda Tory            0000002        IT-Literature Inc
Databases            Manda Tory            0000001        IT-Literature Inc
```

Pause - part 1

Alternative design (bonus info)

- book is about the book title and isbn
- author is about the author name
- publisher is about the publisher name
- book_publisher is about who publishes what book(s)
- book_author is about who authored what book

```
sqlite> .schema book
CREATE TABLE book(title TEXT, isbn TEXT PRIMARY KEY NOT NULL);
sqlite> .schema book_author
CREATE TABLE book_author(isbn TEXT, author_id INTEGER);
sqlite> .schema book_publisher
CREATE TABLE book_publisher(isbn TEXT, publisher_id INTEGER);
```

Query gets a little more complicated, though

```
sqlite> SELECT title, book.isbn, a.name AS author_name, publisher.name AS pub FROM book
NATURAL JOIN book_author NATURAL JOIN author a NATURAL JOIN book_publisher JOIN publisher
ON book_publisher.publisher_id = publisher.publisher_id;
title                   isbn                  author_name     pub
----------------------  --------------------  --------------  ----------------------
Java direkt med Swing   9789144104317         Jan Skansholm   Studentlitteratur
Databasteknik           9789144044491         Thomas Padron-  Studentlitteratur
Programming in Java     1234567               Henrik and Rik  Juneday
Swedish Flowers         0000005               Lynn E. Uss     Biology Books AB
More Java               0000004               M.T. Objects    IT-Literature Inc
Java                    0000003               D. Fault Value  IT-Literature Inc
SQL                     0000002               Manda Tory      IT-Literature Inc
Databases               0000001               Manda Tory      IT-Literature Inc

--OR (handle the fact that there are two "name" columns):
sqlite> SELECT title, isbn, name AS author, publisher FROM book
   ...> NATURAL JOIN book_author NATURAL JOIN author a NATURAL JOIN book_publisher
   ...> NATURAL JOIN (SELECT publisher_id, name AS publisher FROM publisher);
```

Pause part 2

We can always create a "view" (bonus info!)

```
sqlite> CREATE VIEW ALL_ABOUT_BOOK AS
   ...> SELECT title, book.isbn, a.name AS author_name, publisher.name as pub FROM book
   ...> NATURAL JOIN book_author NATURAL JOIN author a NATURAL JOIN book_publisher
   ...> JOIN publisher ON book_publisher.publisher_id = publisher.publisher_id;

sqlite> SELECT * FROM ALL_ABOUT_BOOK;
title                   isbn                  author_name     pub
----------------------  --------------------  --------------  ----------------------
Java direkt med Swing   9789144104317         Jan Skansholm   Studentlitteratur
Databasteknik           9789144044491         Thomas Padron-  Studentlitteratur
Programming in Java     1234567               Henrik and Rik  Juneday
Swedish Flowers         0000005               Lynn E. Uss     Biology Books AB
More Java               0000004               M.T. Objects    IT-Literature Inc
Java                    0000003               D. Fault Value  IT-Literature Inc
SQL                     0000002               Manda Tory      IT-Literature Inc
Databases               0000001               Manda Tory      IT-Literature Inc
```

Another alternative (bonus info!)

```
-- We can make a table of the connection between books, authors and publishers
CREATE TABLE publication(isbn TEXT, publisher_id INTEGER, author_id INTEGER);

sqlite> SELECT isbn, title, publisher, author FROM book
   ...> NATURAL JOIN publication
   ...> NATURAL JOIN (SELECT publisher_id, name AS publisher FROM publisher)
   ...> NATURAL JOIN (SELECT author_id, name AS author FROM author);

isbn            title                 publisher           author
--------------  --------------------  ------------------  ----------------------
9789144104317   Java direkt med Swing  Studentlitteratur   Jan Skansholm
1234567         Programming in Java    Juneday             Henrik and Rikard
0000005         Swedish Flowers       Biology Books AB    Lynn E. Uss
0000004         More Java             IT-Literature Inc   M.T. Objects
0000003         Java                  IT-Literature Inc   D. Fault Values
0000002         SQL                   IT-Literature Inc   Manda Tory
0000001         Databases             IT-Literature Inc   Manda Tory
9789144044491   Databasteknik         Studentlitteratur   Thomas Padron-Mccarthy
```

Another alternative (bonus info!)

```
-- This makes for a nice model - book, author, publisher and publication
CREATE TABLE book(title TEXT, isbn TEXT PRIMARY KEY NOT NULL);
CREATE TABLE author(author_id INTEGER PRIMARY KEY NOT NULL, name TEXT NOT NULL);
CREATE TABLE publisher(publisher_id INTEGER PRIMARY KEY NOT NULL, name TEXT NOT NULL);
CREATE TABLE publication(isbn TEXT, publisher_id INTEGER, author_id INTEGER);

sqlite> SELECT isbn, title, publisher, author FROM book
   ...> NATURAL JOIN publication
   ...> NATURAL JOIN (SELECT publisher_id, name AS publisher FROM publisher)
   ...> NATURAL JOIN (SELECT author_id, name AS author FROM author);

isbn            title                  publisher           author
--------------  ---------------------  ------------------  ----------------------
9789144104317   Java direkt med Swing  Studentlitteratur   Jan Skansholm
1234567         Programming in Java    Juneday             Henrik and Rikard
0000005         Swedish Flowers        Biology Books AB    Lynn E. Uss
0000004         More Java              IT-Literature Inc   M.T. Objects
0000003         Java                   IT-Literature Inc   D. Fault Values
0000002         SQL                    IT-Literature Inc   Manda Tory
0000001         Databases              IT-Literature Inc   Manda Tory
9789144044491   Databasteknik          Studentlitteratur   Thomas Padron-Mccarthy
```

Further reading

- http://www.databasteknik.se/webbkursen/er/index.html (Swedish)
- http://www.databasteknik.se/webbkursen/er2relationer/index.html (Swedish)
- http://www.databasteknik.se/webbkursen/normalisering/index.html (Swedish)

Extra - More on JOIN and stuff

A joint venture

Find names that don't have a unique initial

How do we list all the names of students whose name starts with the same letter as some other student?

```
sqlite> SELECT DISTINCT name FROM students ORDER BY name;
name
Chip
Dale
Dewey
Donald
Goofy
Hewey
Louie
Mickey
Minnie
Pluto
Scrooge
```

Self-join

```
SELECT DISTINCT s1.name
           FROM students s1
           JOIN students s2
             ON SUBSTR(s1.name, 1, 1) = SUBSTR(s2.name, 1, 1)
            AND s1.student_id != s2.student_id -- why do we need this?
       ORDER BY s1.name;
name
Dale
Dewey
Donald
Mickey
Minnie
```

Finding names with unique initial?

```
SELECT name FROM
   ( SELECT name, SUBBSTR(name,1,1) AS "letter", COUNT(*) AS number
       FROM students
   GROUP BY letter
     HAVING number = 1 );
name
Chip
Goofy
Hewey
Louie
Pluto
Scrooge
```

What colors doesn't any car have?

```
CREATE TABLE color(color_id INTEGER PRIMARY KEY, color TEXT UNIQUE NOT
NULL);
CREATE TABLE car(id INTEGER PRIMARY KEY, license TEXT UNIQUE NOT NULL,
  color_id INTEGER, FOREIGN KEY(color_id) REFERENCES color(color_id));
```

```
sqlite> select * from car;
id          license     color_id
----------  ----------  ----------
1           AAA 001     1
2           AAA 002     2
3           AAA 003     3
```

```
sqlite> select * from color;
color_id    color
----------  ----------
1           Red
2           Blue
3           White
4           Green
5           Silver
6           Gold
7           Pink
8           Grey
9           Black
10          Orange
11          Purple
```

Use an outer join (keep all rows in first table)

```
sqlite>  SELECT color
           FROM color
LEFT OUTER JOIN car
             ON car.color_id = color.color_id
          WHERE license is null;
color
----------
Green
Silver
Gold
Pink
Grey
Black
Orange
Purple
```

Huh?

```
sqlite>  SELECT color, license
           FROM color
LEFT OUTER JOIN car        -- OUTER JOIN: keep all rows from color!
             ON car.color_id = color.color_id;
color       license
----------  ----------
Red         AAA 001
Blue        AAA 002
White       AAA 003
Green       NULL
Silver      NULL
Gold        NULL
Pink        NULL
Grey        NULL
Black       NULL
Orange      NULL
Purple      NULL
```

So we filter using WHERE license is null

```
sqlite>  SELECT color
           FROM color
LEFT OUTER JOIN car
             ON car.color_id = color.color_id
          WHERE license is null;
color
----------
Green
Silver
Gold
Pink
Grey
Black
Orange
Purple
```

SQLite3 doesn't have RIGHT JOINs

Since SQLite3 doesn't have RIGHT JOINs, we use LEFT OUTER JOIN and put the table whose rows we are interested in to the left of the JOIN

```
sqlite> SELECT color, license
          FROM color LEFT OUTER JOIN car ON car.color_id = color.color_id;
               left                      right
```

All rows in the right table (car) will get NULL values on its columns, since there is no matching car in the result for some color rows!

```
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE color(color_id INTEGER PRIMARY KEY, color TEXT UNIQUE NOT NULL);
INSERT INTO "color" VALUES(1,'Red');
INSERT INTO "color" VALUES(2,'Blue');
INSERT INTO "color" VALUES(3,'White');
INSERT INTO "color" VALUES(4,'Green');
INSERT INTO "color" VALUES(5,'Silver');
INSERT INTO "color" VALUES(6,'Gold');
INSERT INTO "color" VALUES(7,'Pink');
INSERT INTO "color" VALUES(8,'Grey');
INSERT INTO "color" VALUES(9,'Black');
INSERT INTO "color" VALUES(10,'Orange');
INSERT INTO "color" VALUES(11,'Purple');

CREATE TABLE car(id INTEGER PRIMARY KEY, license TEXT UNIQUE NOT NULL, color_id INTEGER,
FOREIGN KEY(color_id) REFERENCES color(color_id));
INSERT INTO "car" VALUES(1,'AAA 001',1);
INSERT INTO "car" VALUES(2,'AAA 002',2);
INSERT INTO "car" VALUES(3,'AAA 003',3);
COMMIT;
```

NULL

Databases - representing absence of a value

Database structure

In a database we store values in tables. A table has columns with names and types, and rows of data:

```
sqlite> .schema
CREATE TABLE book(title TEXT, author TEXT, isbn TEXT PRIMARY
KEY NOT NULL);
sqlite> SELECT * FROM book;
title                 author         isbn
--------------------  -------------  -------------
Java direkt med Swing  Jan Skansholm  9789144104317
Databasteknik         Thomas Padron  9789144044491
```

Inserting a book without title

Let's insert a book without a title:

```
sqlite> INSERT INTO book (author, isbn)
...>    VALUES('Henrik and Rikard', '1234567');
```

What is stored for the above row in the column title?

Inserting a book without title

Let's investigate what was stored:

```
sqlite> SELECT * FROM book WHERE isbn = '1234567';
title       author             isbn
----------  -----------------  ----------
            Henrik and Rikard  1234567
```

It's empty? What does that mean? Empty string?

No, in this case there is no value at all, which is called NULL.

Making NULL values apparent (in SQLite3)

Let's make NULL values more obvious:

```
sqlite> .nullvalue NULL
sqlite> SELECT * FROM book WHERE isbn = '1234567';
title       author             isbn
----------  -----------------  ----------
NULL        Henrik and Rikard  1234567
```

What about empty strings?

An empty string is still a string. We can prove that using the `is` operator:

```
sqlite> SELECT '' IS '';                    remember: 0 means false
'' IS ''                                    and 1 means true
----------
1
sqlite> SELECT '' IS 'not empty';
'' IS 'not empty'
-----------------
0
sqlite> SELECT '' IS NULL;
'' IS NULL
----------
0
```

So, how can we understand the meaning of NULL?

NULL simply means "absence of a value". This is quite useful. It means that we can allow some columns to represent the lack of a value. We can use this to select rows where some column lacks a value:

```
sqlite> SELECT author, isbn FROM book WHERE title IS NULL;
author             isbn
-----------------  ----------
Henrik and Rikard  1234567
```

Checking for NULL is also useful when performing certain JOIN operations, e.g. when you want to check for the absence of references between tables, like "What publishers have no books in our book table".

Adding publisher as a foreign key

```
sqlite> CREATE TABLE book(title TEXT, author TEXT,
...>    isbn TEXT PRIMARY KEY NOT NULL, publisher_id INTEGER);
sqlite> CREATE TABLE
...>    publisher(publisher_id INTEGER PRIMARY KEY NOT NULL,
...>              name TEXT UNIQUE NOT NULL);
```

In this new design, every book has a reference to the publisher table (the publisher_id).

Adding publisher as a foreign key - JOINing

```
sqlite> SELECT title, author, name AS publisher
          FROM book
  NATURAL JOIN publisher;

title                 author         publisher
--------------------  -------------  -----------------
Java direkt med Swing  Jan Skansholm  Studentlitteratur
Databasteknik          Thomas Padron  Studentlitteratur
Programming in Java    Henrik and Ri  Juneday
```

But, can we find out if there are publishers without books, and who these publishers are?

What if a publisher has no books?

```
sqlite>   SELECT title, name as publisher
            FROM publisher
 LEFT OUTER JOIN book ON book.publisher_id = publisher.publisher_id;
title          publisher
-------------  -----------------
Databasteknik  Studentlitteratur
Java direkt m  Studentlitteratur
Programming i  Juneday
NULL           Mayday! Mayday!
NULL           Oh Really
```

The publishers “Mayday! Mayday!” and “Oh Really” don’t have any titles, as shown when using LEFT OUTER JOIN (meaning “show columns from the left table, regardless!”).

Using the previous information

Since we saw that title became NULL for some publishers, we can use this fact:

```
sqlite> SELECT name as publisher
...>    FROM publisher LEFT OUTER JOIN book
...>     ON book.publisher_id = publisher.publisher_id
...>   WHERE title IS NULL;
publisher
---------------
Mayday! Mayday!
Oh Really
```

The above answers the question: “What publishers have no books”. That’s a use case for NULL!

Constraints

Garbage in, garbage out

GIGO

> *On two occasions I have been asked, "Pray, Mr. Babbage, if you put into the machine wrong figures, will the right answers come out?" … I am not able rightly to apprehend the kind of confusion of ideas that could provoke such a question.*
>
> -- Charles Babbage, Passages from the Life of a Philosopher

A question of sanitation

Let's pretend we have a table for storing data on persons:

```
sqlite> .schema persons
CREATE TABLE persons (last_name TEXT, first_name TEXT, born TEXT, sex TEXT);

sqlite> SELECT * from persons;
last_name        first_name       born        sex
---------------  ---------------  ----------  -----
Nilsson          Tommy            1960-03-11  Man
Norum            John             1964-02-23  Man
Jett             Joan             1958-09-22  Woman
Wilson           Ann              1950-06-19  Woman
sqlite>
```

Types too blunt

As you might have noticed, all the columns were of type text. While this is convenient, it is not a great choice for in particular the Born and Sex columns.

Why?

Garbage in

Suppose we add persons via either a web interface or a command line interface. The clerk who enters new persons into the system forgets what the valid values for Sex and Born were.

The following person is entered into the table:

"Doe", "Jane", "11/09/14","Female"

Inconsistent data

The state of the table is now:

```
sqlite> SELECT * from persons;
last_name        first_name       born        sex
---------------  ---------------  ----------  -------
Nilsson          Tommy            1960-03-11  Man
Norum            John             1964-02-23  Man
Jett             Joan             1958-09-22  Woman
Wilson           Ann              1950-06-19  Woman
Doe              Jane             11/09/14    Female
```

Ordering by birth date

Let's order all persons by Born:

```
sqlite> SELECT * FROM persons ORDER BY born;
last_name        first_name       born        sex
---------------  ---------------  ----------  -------
Doe              Jane             11/09/14    Female
Wilson           Ann              1950-06-19  Woman
Jett             Joan             1958-09-22  Woman
Nilsson          Tommy            1960-03-11  Man
Norum            John             1964-02-23  Man
```

The very young Jane now is listed as the oldest person.

Selecting all women

```
sqlite> SELECT * FROM persons WHERE sex='Woman';
last_name        first_name       born        sex
---------------  ---------------  ----------  -------
Jett             Joan             1958-09-22  Woman
Wilson           Ann              1950-06-19  Woman
```

No Jane...

How do we fix this?

What about:

```
sqlite> SELECT * FROM persons WHERE sex='Woman' OR sex='Female';
```

What's the problem with this?

Relax

I'll need some information first

Just the basic facts

Can you show me where it hurts?

Look for inconsistencies

We could list all variations of Sex:

```
sqlite> SELECT DISTINCT sex FROM persons;
sex
---------------
Man
Woman
Female
sqlite>
```

Two variations for Woman. Not good.

Guess what the DISTINCT keyword does? It is not part of the exam.

Statistics, please

Which version is in majority?

```
sqlite> SELECT sex, COUNT(*) FROM persons GROUP BY sex;
sex              COUNT(*)
---------------  ---------------
Female           1
Man              2
Woman            2
```

Only one “Female”

COUNT combined with GROUP BY is sometimes quite useful. They are not part of the exam, and just used here for reference.

Fixing the problem

Wouldn’t it be better if we only accepted valid input into the persons table?

This is where constraints come into play.

Let’s start with the values for Sex.

Warning! The syntax for constraints used in this lecture are specific to SQLite3. If you are using a different dbms, check the manual for equivalent constraints.

Using an enumeration for the Sex column

```
CREATE TABLE persons2(last_name TEXT, first_name TEXT, born TEXT,
  sex TEXT CHECK( sex IN ('Woman','Man')));
```

Let's try to violate that constraint!

```
sqlite> INSERT INTO persons2 VALUES('Joplin', 'Janis', '1943-01-19',
  'Female');
Error: CHECK constraint failed: persons2
sqlite>
```

Using a function to check dates

We want to force dates to have the format 'YYYY-mm-dd HH:MM:SS':

```
CREATE TABLE persons2(last_name TEXT, first_name TEXT,
 born DATETIME CHECK(born IS datetime(Born)),
 Sex text      CHECK(Sex IN ('Woman','Man'))
);
/* datetime converts a string to a date of the format above if it can */
```

Violation:

```
sqlite> INSERT INTO persons2 VALUES('Joplin', 'Janis', '01/19/43','Woman');
Error: CHECK constraint failed: persons2
sqlite>
```

What about foreign keys?

If we go back to our books example where we normalised the Publisher to its own table, wouldn't it be great to only accept a valid publisher_id for a book?

We don't want a book to have an invalid publisherid, i.e. one that is not an ID in the publishers table.

```
sqlite> select * from publishers;
1|Bonnier
2|Books R us
sqlite>
```

What would it mean to store a book with PublisherID 666? Inconsistency.

Creating a foreign key constraint

Let's make sure that PublisherID in the books table really references a valid publisher_id in the publishers table:

```
sqlite> CREATE TABLE "books2"( author TEXT, title TEXT,
  isbn TEXT PRIMARY KEY,
  publisherid INTEGER,
  FOREIGN KEY(publisherid) REFERENCES publishers(publisher_id)
);
```

Test: Insert a book with an invalid PublisherID:

```
sqlite> PRAGMA foreign_keys=true;
sqlite> INSERT INTO books2 VALUES ('Tage Danielsson', 'Tankar från roten',
'0-0-0-0-0-6', 666);
Error: FOREIGN KEY constraint failed
```

Primary keys are enforced to be unique

In the books table, ISBN was selected as the PRIMARY KEY with only unique values.

Test: Insert a book with an already existing ISBN:

```
sqlite> INSERT INTO books2 VALUES ('Tage Danielsson', 'Tankar från roten',
'0-0-0-0-0-1',2);
Error: UNIQUE constraint failed: books2.ISBN
sqlite>
```

But... we can sanitize input at the application layer

Yes. But can you guarantee that nobody enters data into the database using some other means?

Does it hurt to sanitize input in more than one place?

As a tester, it is your job to find potential future problems, at all layers.

Discussion

What different strategy could we have used for correctness of the values for Sex? Hint: there are only two legal sexes in Sweden. What datatype is typically used for modelling one of two values?

What if Sweden implements a third legal sex. Which strategy seems better for the future?

The persons table could actually be used to represent Authors. What changes would we need in order to implement that in the books table?

Read

http://zetcode.com/db/sqlite/constraints/

https://www.sqlite.org/lang_createtable.html#constraints

https://www.sqlite.org/lang_datefunc.html

https://www.sqlite.org/pragma.html#pragma_foreign_keys

The persons table with constraints

```
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE persons (last_name TEXT, first_name TEXT, born TEXT, sex TEXT);
INSERT INTO "persons" VALUES('Nilsson','Tommy','1960-03-11','Man');
INSERT INTO "persons" VALUES('Norum','John','1964-02-23','Man');
INSERT INTO "persons" VALUES('Jett','Joan','1958-09-22','Woman');
INSERT INTO "persons" VALUES('Wilson','Ann','1950-06-19','Woman');
INSERT INTO "persons" VALUES('Doe','Jane','11/09/14','Female');
CREATE TABLE persons2(last_name TEXT, first_name TEXT, born DATETIME
check(born IS datetime(born)), sex TEXT CHECK( sex IN ('Woman','Man')));
INSERT INTO "persons2" VALUES('Joplin','Janis','1943-01-19
00:00:00','Woman');
COMMIT;
```

The books2 table with constraints

```
PRAGMA foreign_keys=ON;
BEGIN TRANSACTION;
CREATE TABLE "books2"(author TEXT, title TEXT,
                      isbn TEXT PRIMARY KEY,
                publisherid INTEGER,
                FOREIGN KEY(publisherid)
                 REFERENCES publishers(publisher_id)
);
INSERT INTO "books2" VALUES('John Smith','Life','0-0-0-0-0-1',1);
INSERT INTO "books2" VALUES('James Woody','Love','0-0-0-0-0-2',1);
INSERT INTO "books2" VALUES('Joan Carmen','Guns','0-0-0-0-0-3',1);
INSERT INTO "books2" VALUES('Johnanna Boyd','Code','0-0-0-0-0-4',1);
INSERT INTO "books2" VALUES('Eva Peron','Cars','0-0-0-0-0-5',2);
COMMIT;
```