IS5102
Database Management Systems
Lecture 7: Introduction to SQL

Alexander Konovalov
alexander.konovalov@st-andrews.ac.uk

(with thanks to Susmit Sarkar)

2021

Before

- Data Modeling
- ER models and Relational Models
- Relational algebra

- Overview of the SQL Query Language
- Data Definition
- Basic Query Structure

History

- (early 1970s) IBM Sequel Language as part of System R project
- (early 1980s) Evolved and renamed to Structured Query Language (SQL)
- (1986) ANSI and ISO published an SQL standard
  - SQL-86; SQL-89; SQL-92
  - SQL:1999
  - SQL:2003; SQL:2006; SQL:2008; SQL:2011; SQL:2016

Not all features supported in all systems

SQLite documentation: see https://www.sqlite.org/docs.html
in particular, "About", "Distinctive features" and "Quirks" under "Overview Documents"

You have most likely used SQLite today: https://www.sqlite.org/famous.html

- Data Definition Language (**DDL**)
  define, modify and delete relations
- Data Manipulation Language (**DML**)
  insert, modify and delete tuples; perform **queries**
- Integrity Constraints
  enforced by forbidding updates violating them
- Data Control Language (**DCL**)
  Transactions, authorisation, ...

Allow the specification of **information about relations**, e.g.:

- Schema of each relation
- Domain of values for each attribute
- Integrity constraints

Also allow additional information, e.g.:

- Indices for each relation
- Security and authorization information for relation
- Physical storage structure of relation on disk

Domain Types in SQL standard

- `CHAR(n)`: fixed length character string, length `n`
- `VARCHAR(n)`: variable length character string, max length `n`
- `INT` (or `INTEGER`): integer (machine-dependent size)
- `SMALLINT`: "small" integer (machine-dependent size)
- `NUMERIC(p,d)`: fixed-point number with user-specified precision
  (`p` digits of which `d` are to the right of the decimal point)
- `REAL` and `DOUBLE`: floating point numbers (machine-dependent precision)
- `FLOAT(n)`: floating point number (at least `n` digit precision)

Domain Types in SQL standard

Temporal data

- `DATE`: 4 year digits (yyyy)+ 2 month digits(mm) + 2 day digits (dd).
  Format 'yyyy-mm-dd' e.g. `2012-10-04`
- `TIME`: 2 hour digits(hh) + 2 minute digits(mm + 2 second digits(ss) in 24 hour notation: 'hh:mm:ss'
- `TIMESTAMP`: Combination of the above
- More on this later...

Practical implementations:

- "rigidly typed":
  MySQL et al.: `http://www.w3schools.com/sql/sql_datatypes.asp`
- "flexibly typed":
  SQLite: `https://www.sqlite.org/datatype3.html`

An SQL relation is defined using the `CREATE TABLE` command:

```
CREATE TABLE r (
    A1 D1,
    A2 D2,
    ...,
    An Dn,
    (integrity-constraint1),
    ...,
    (integrity-constraintk)
);
```

- `r` is the name of the relation
- `Ai` is an attribute name in the schema of relation `r`
- `Di` is the data type of values in domain of attribute `Ai`

CREATE TABLE Example

```
CREATE TABLE department (
    dept_id     CHAR(5),
    dept_name   VARCHAR(20),
    building    VARCHAR(15),
    budget      NUMERIC(12,2)
);
```

We are following the **SQL Style Guide** by Simon Holywell:

https://www.sqlstyle.guide

SQLite documentation entry for `CREATE TABLE`:

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

- Add tuples to a relation

```
INSERT INTO department
VALUES ('CS', 'Computer Science', 'Jack Cole', 1500000),
       ('MATH','Mathematics and Statistics', 'Maths', 900000),
       ('PHYS', 'Physics and Astronomy', 'Physics', 1500000);
```

- Remove all tuples from a relation

```
DELETE FROM department;
```

- Remove a relation from a database

```
DROP TABLE department;
```

Obligatory XKCD: https://xkcd.com/327/

Queries are performed using the SELECT command, which lists the attributes requested as a result of the query.

A typical SQL query has the form:

```
SELECT A1, A2, ..., An
  FROM r1, r2, ..., rm
 WHERE P;
```

In this query:

- Ai is an attribute
- ri is a relation
- P is a predicate

The result of an SQL query is a relation.

Simple Query Example

An asterisk * is a wildcard to denote all attributes in a relation. It can be used in

```
SELECT * FROM department;
```

to inspect the content of the `department` table.

Example of an SQL script

```
CREATE TABLE department (
    dept_id     CHAR(5),
    dept_name   VARCHAR(20),
    building    VARCHAR(15),
    budget      NUMERIC(12,2)
);

INSERT INTO department
VALUES ('CS', 'Computer Science', 'Jack Cole', 1500000),
       ('MATH','Mathematics and Statistics', 'Maths', 900000),
       ('PHYS', 'Physics and Astronomy', 'Physics', 1500000);

SELECT * FROM department;
```

- Put the code from the example of an SQL script into the text file called `depts.sql`
- Open the terminal in the directory containing the file `depts.sql`, and execute it with:
  - `sqlite3 --init depts.sql` to connect to a transient in-memory database, or
  - `sqlite3 uni.db --init depts.sql` to connect to a database `uni.db` (new database will be created, if it does not exist)
- This will run commands from the `depts.sql` script and keep SQLite shell open for further exploration. You can enter, for example:
  - `.tables` to check that it outputs `department`
  - `.schema` to show corresponding `CREATE` commands
  - `.dump` to dump the database in an SQL text format
- Then enter `.quit` (or `.q`) to exit SQLite shell
- Add lines

  ```
  .mode column
  .headers on
  ```

  on top of the script to change output formatting, and try this procedure again

- You have now established a repeatable and reproducible workflow for further SQL practice
  - It recreates tables each time, so you can easily modify their schemas and test changes
  - You can also rerun the script entering `.read depts.sql` in the SQLite shell
  - Repeat this process several times. What goes wrong?
  - Fix this by adding the line `DROP table department;` on top of the script
    - In this case, the next restart will cause `DROP` to report "no such table" errors
    - That clearly happens because the database is empty - can be ignored in this case

Running in the DB Browser

This is the equivalent of the previous workflow for the DB Browser for SQLite

- Put the code from the example of an SQL script into the text file called `depts.sql` and open it from the "Execute SQL" tab (alternatively, copy and paste it into the code editor in the "Execute SQL" tab)
  - Customise DB Browser: use monospace font (e.g. Monaco) for indentation, close some tabs
- Use "New Database" button to create the new database. Select "Cancel" in the form for editing table definition (later use "Open Database" to connect to an existing database)
- Run SQL code using "Execute all" button or the keyboard shortcut
- Inspect "Database Structure" and "Browse Data" tabs

Running in the DB Browser - 2

- Run the same SQL code again. What goes wrong?
- Fix this by adding the line `DROP table department;` on top of the SQL script
- You can now modify SQL code and run it again
  - Resetting tables is more cumbersome though:
    - Change to a new database (or delete an old one and pick the same name for the new one)
    - Comment out `DROP` command(s) for the first run of the script

Know where the docs are

- SQLite
  - https://sqlite.org/docs.html
- Command Line Shell For SQLite
  - https://sqlite.org/cli.html
- DB Browser for SQLite
  - https://github.com/sqlitebrowser/sqlitebrowser/wiki