Databases - SQL - 1

Gordon Royle

School of Mathematics & Statistics
University of Western Australia

SQL - Part 1

This lecture will cover

- The Client-Server Model
- Structured Query Language
  - Data Definition Language — how to create, alter and delete *tables*
  - Data Manipulation Language — how to write queries on a single table

Client-server

Databases are almost always accessed in a *client-server* fashion

- The *server* is a program that controls the data, "listens for" requests, and then *serves* the requestors
- The *clients* are programs that make the requests and receive the responses
- Client programs can be on the same machine or (more usually) remote

Typically, the server will handle requests from many different client programs "at the same time".

One of the most important tasks of a DBMS is to ensure that each user appears to be *isolated* from the others, even if they are working with the *same table*

SQL implementations

There are numerous *implementations* of SQL, and equally many *client programs*, ranging from simple command-line-interface (CLI) programs to sophisticated GUI-based interfaces.

In these lectures, I will use

- MySQL (v5.6.12) as the server
- The terminal program `mysql` as the client

Database structure

A single MySQL server manages a two-level hierarchy

- At the top-level there are a number of *databases*
- Each database contains a number of *tables*

Client programs always maintain a notion of "*the current database*" and all names and commands are interpreted relative to this database.

Looking around the database

- `SHOW DATABASES;`
  This shows all the databases on the server
- `USE world;`
  This makes `world` the current database
- `SHOW TABLES;`

```
mysql> show tables;
+-----------------+
| Tables_in_world |
+-----------------+
| City            |
| Country         |
| CountryLanguage |
+-----------------+
3 rows in set (0.00 sec)
```

Looking round the database

The `DESCRIBE` command tells you about a table's schema:

```
mysql> DESCRIBE City;
+-------------+----------+------+-----+---------+----------------+
| Field       | Type     | Null | Key | Default | Extra          |
+-------------+----------+------+-----+---------+----------------+
| ID          | int(11)  | NO   | PRI | NULL    | auto_increment |
| Name        | char(35) | NO   |     |         |                |
| CountryCode | char(3)  | NO   | MUL |         |                |
| District    | char(20) | NO   |     |         |                |
| Population  | int(11)  | NO   |     | 0       |                |
+-------------+----------+------+-----+---------+----------------+
5 rows in set (0.01 sec)
```

This table/relation has *five attributes*, so each *row* comprises an `ID`, a `Name`, a `CountryCode`, a `District` and a `Population`.

Making tables

There are three commands that work with *an entire table* at once:

These commands create, alter, or delete the *relation schema*, rather than working with the data items (the rows).

- `CREATE TABLE ...`
- `ALTER TABLE ...`
- `DROP TABLE ...`

REMEMBER: These commands all have `TABLE` in their name — leaving this out is a common error!

Creating

The new table must be *named* and its *schema* defined

```
CREATE TABLE City (
  ID INT,
  Name CHAR(35),
  CountryCode CHAR(3),
  District CHAR(20),
  Population INT);
```

So each attribute has been given a name and a type — `INT` is an *integer* type for storing numbers, while `CHAR(35)` is a *character string* of up to 35 characters.

Altering or dropping

Altering a table means *adding* or *deleting* attributes:

- `ALTER TABLE City ADD COLUMN Area INT;`
  Adds a new column `Area` to the right of the table
- `ALTER TABLE City DROP COLUMN Area;`
  Deletes the column `Area`
- `DROP TABLE City;`
  Deletes the entire relation

Inserting Data

There are two main ways to insert data into a table

- `INSERT INTO` *table-name* `VALUES` (*values*)
- `LOAD DATA`

The Australian city of Darwin is missing from the table `City`[1], so let's add it:

```
INSERT INTO City VALUES (4080, 'Darwin', 'AUS', 150000);
```

Notice: the *integer types* are included just as numbers, while the two *string types* are *delimited* by either single or double quotes, which are not part of the actual data.

[1] The data in this sample DB is very old as MySQL no longer maintain it.

The Relation Instance

Suppose the table contained only the Australian cities:

ID	Name	CountryCode	Population
130	Sydney	AUS	3276207
131	Melbourne	AUS	2865329
132	Brisbane	AUS	1291117
133	Perth	AUS	1096829
134	Adelaide	AUS	978100
135	Canberra	AUS	322723
136	Gold Coast	AUS	311932
137	Newcastle	AUS	270324
138	Central Coast	AUS	227657
139	Wollongong	AUS	219761
140	Hobart	AUS	126118
141	Geelong	AUS	125382
142	Townsville	AUS	109914
143	Cairns	AUS	92273
4080	Darwin	AUS	150000

Terminology

At the risk of overkill, a reminder about how the formal/informal terminology compares:

- The *table / relation* is called `City`.
- The *relation schema* dictates four *columns / attributes* with the types as previously indicated.
- Each *row / tuple* contains the data for one `City`.
- This *relation instance* is the collection of 15 rows making up the *current contents* of the table.

First Normal Form

There is an extensive theory of *database normalization* which is used to design databases that are free of *structural* error and/or inefficiency.

A database is in *first normal form*[2] if it meets the following criterion:

- Each tuple contains an *atomic value* of the correct type for each attribute,

The word *atomic* means "indivisible", but in this context should not be taken literally — it just means that each row must contain *exactly one value* of the correct type for the column.

[2] There are additional more technical criteria that we'll meet later

Atomic values

EXAMPLE If the attribute is of type `INT`, then each row must contain *just one* integer value in that column.

Name	Packet Weight	Price
Spearmint Leaves	250	5.00
Spearmint Leaves	750	12.00

The value "250" is a single integer.

Name	Packet Weight	Price
Spearmint Leaves	{250,750}	{5.00,12.00}

The value $\{250, 750\}$ is a *set of integers* and not a single integer.

Querying

The most fundamental database task is *querying* the database.

For this purpose, the most important statement is the `SELECT` statement, which can be extremely simple or very complicated due to its many optional parts.

`SELECT` *columns*
`FROM` *tables*
`WHERE` *conditions*
`GROUP BY` *group columns*
`HAVING` *more conditions*
`ORDER BY` *sort columns*
`LIMIT` *number*

The `world.sql` database

On the MySQL website, they supply a sample database that has three tables `City`, `Country` and `CountryLanguage` for training and experimentation.

- `City` contains information about the name, country and population of an individual city
  Each row of the table represents one city
- `Country` contains a wealth of information about a country
  Each row of this table represents one country
- `CountryLanguage` contains information about a language spoken in a country.
  Each row of this table represents a country/language pair.

Some cities

```
mysql> SELECT * FROM City LIMIT 10;
+----+----------------+-------------+---------------+------------+
| ID | Name           | CountryCode | District      | Population |
+----+----------------+-------------+---------------+------------+
|  1 | Kabul          | AFG         | Kabol         |    1780000 |
|  2 | Qandahar       | AFG         | Qandahar      |     237500 |
|  3 | Herat          | AFG         | Herat         |     186800 |
|  4 | Mazar-e-Sharif | AFG         | Balkh         |     127800 |
|  5 | Amsterdam      | NLD         | Noord-Holland |     731200 |
|  6 | Rotterdam      | NLD         | Zuid-Holland  |     593321 |
|  7 | Haag           | NLD         | Zuid-Holland  |     440900 |
|  8 | Utrecht        | NLD         | Utrecht       |     234323 |
|  9 | Eindhoven      | NLD         | Noord-Brabant |     201843 |
| 10 | Tilburg        | NLD         | Noord-Brabant |     193238 |
+----+----------------+-------------+---------------+------------+
10 rows in set (0.00 sec)
```

Some large cities

What query will list all the cities with population over 200000?

Some large cities

What query will list all the cities with population over 200000?

```
SELECT Name, Population
FROM City
WHERE Population > 200000;
+-----------+------------+
| Name      | Population |
+-----------+------------+
| Kabul     |    1780000 |
| Qandahar  |     237500 |
| Amsterdam |     731200 |
| Rotterdam |     593321 |
...
```

Selection

Let's analyse this a bit more carefully:

- `SELECT Name, Population`
  This indicates *which columns* should be in the repsonse
- `FROM City`
  This indicates *from which table(s)* the rows should come
- `WHERE Population > 200000`
  This indicates *from which rows* you want to extract the columns

This statement is purely *declarative* — it specifies *what* the user wants, but does not specify *how* the DBMS should accomplish this.

The mental model

The right mental model is to imagine the DBMS as a *row-processing machine*, converting an existing table (or tables) into new tables.

- The DBMS *constructs* the candidate output rows from the input tables,
- The DBMS *tests* each row to see if passes the conditions,
- The DBMS *extracts* the desired columns from the successful rows,
- The DBMS *produces* a new table containing the results of this process.

Tables in, tables out!

Country

The Country relation (table) has a large number of columns; we'll just use a few of them:

```
SELECT Name, SurfaceArea, Population
FROM Country
LIMIT 10;
+----------------------+-------------+------------+
| Name                 | SurfaceArea | Population |
+----------------------+-------------+------------+
| Aruba                |      193.00 |     103000 |
| Afghanistan          |   652090.00 |   22720000 |
| Angola               |  1246700.00 |   12878000 |
| Anguilla             |       96.00 |       8000 |
| Albania              |    28748.00 |    3401200 |
| Andorra              |      468.00 |      78000 |
| Netherlands Antilles |      800.00 |     217000 |
| United Arab Emirates |    83600.00 |    2441000 |
| Argentina            |  2780400.00 |   37032000 |
| Armenia              |    29800.00 |    3520000 |
+----------------------+-------------+------------+
```

Population Density

```
SELECT Name, Population / SurfaceArea
FROM Country
LIMIT 10;
+----------------------+---------------------------+
| Name                 | Population / SurfaceArea  |
+----------------------+---------------------------+
| Aruba                |                533.678756 |
| Afghanistan          |                 34.841816 |
| Angola               |                 10.329670 |
| Anguilla             |                 83.333333 |
| Albania              |                118.310839 |
| Andorra              |                166.666667 |
| Netherlands Antilles |                271.250000 |
| United Arab Emirates |                 29.198565 |
| Argentina            |                 13.318947 |
| Armenia              |                118.120805 |
+----------------------+---------------------------+
10 rows in set (0.00 sec)
```

Dense countries

What query will list countries ordered by population density?

Dense countries

What query will list countries ordered by population density?

```
SELECT Name, Population/SurfaceArea AS Density
FROM Country
ORDER BY Density DESC;
+----------------------------------------------+---------------+
| Name                                         | Density       |
+----------------------------------------------+---------------+
| Macao                                        | 26277.777778 |
| Monaco                                       | 22666.666667 |
| Hong Kong                                    |  6308.837209 |
| Singapore                                    |  5771.844660 |
| Gibraltar                                    |  4166.666667 |
```

CountryLanguage

```
mysql> DESCRIBE CountryLanguage;
+-------------+---------------+------+-----+---------+-------+
| Field       | Type          | Null | Key | Default | Extra |
+-------------+---------------+------+-----+---------+-------+
| CountryCode | char(3)       | NO   | PRI |         |       |
| Language    | char(30)      | NO   | PRI |         |       |
| IsOfficial  | enum('T','F') | NO   |     | F       |       |
| Percentage  | float(4,1)    | NO   |     | 0.0     |       |
+-------------+---------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

CountryLanguage

```
mysql> SELECT * FROM CountryLanguage LIMIT 10;
+-------------+------------+------------+-------------+
| CountryCode | Language   | IsOfficial | Percentage |
+-------------+------------+------------+-------------+
| ABW         | Dutch      | T          |         5.3 |
| ABW         | English    | F          |         9.5 |
| ABW         | Papiamento | F          |        76.7 |
| ABW         | Spanish    | F          |         7.4 |
| AFG         | Balochi    | F          |         0.9 |
| AFG         | Dari       | T          |        32.1 |
| AFG         | Pashto     | T          |        52.4 |
| AFG         | Turkmenian | F          |         1.9 |
| AFG         | Uzbek      | F          |         8.8 |
| AGO         | Ambo       | F          |         2.4 |
+-------------+------------+------------+-------------+
10 rows in set (0.00 sec)
```