Operating systems fundamentals - B07

David Kendall

Northumbria University

What is SQL?

- Structured Query Language
- Used to ‘talk’ to a database server
- A standard language for querying and manipulating data
- Used as a front-end to many database management systems, e.g. MySQL, Oracle, PostgreSQL, Sybase, . . .
- SQL is a very high-level programming language (declarative)
- Three main parts: data definition language, data manipulation language, user privileges

When do you need a database?

- Multiple simultaneous changes to data (concurrency)
- Data changes frequently
- Large data sets where you need to be able to select items of current interest
- Share large data set among many users
- Rapid queries
- Web interface to data, especially dynamic data

SQL features

data definition language

- define relational schemata
- create/alter/delete tables and their attributes

data manipulation language

- insert/delete/modify tuples in tables
- query one or more tables

user privileges

- who can do what to which?

Ways to use SQL

- Console command (**mysql -u** *user* **-p**)
- GUI interfaces are often available
- Interfaces to many programming languages, e.g. Python, Perl, PHP, R, . . .
- SQLite — use SQL without a database server
- We'll use MySQL with commands from the console in this module
- MySQL is the most widely used open-source DataBase Management System (DBMS) available today

Some basic database concepts

- A database server can contain many databases
- Databases are collections of tables
- Tables are two-dimensional with rows (tuples, observations) and columns (attributes, variables)
- Limited mathematical and summary operations available
- Very good at combining information from several tables

Basic interactions with the database server

- Login

  ```
  mysql -u root -p
  Enter password:
  ```

- What databases is the server managing?

  ```
  SHOW DATABASES;
  ```

- What tables are there in a database?

  ```
  SHOW TABLES in <database name>;
  ```

- What columns are there in a table?

  ```
  SHOW COLUMNS in <table name>;
  ```

- What are the types of data in a table?

  ```
  DESCRIBE <table name>;
  ```

- How do I create a database?

  ```
  CREATE DATABASE <database name>;
  ```

- How do I select a particular database to work on?

  ```
  USE <database name>;
  ```

Tables in SQL

- Assume we have a database called `menagerie` that contains a table called `pet`

pet

name	**owner**	**species**	**sex**	**birth**	**death**
Fluffy	Harold	cat	f	1993-02-04	
Claws	Gwen	cat	m	1994-03-17	
Slim	Benny	snake	m	1996-04-29	1997-03-11
Buffy	Harold	dog	f	1989-05-13	

- A **table** (or **relation**) is a *multiset* of *tuples* having the *attributes* specified by the *schema*
- We need to break this definition down . . .

Tables in SQL

pet

name	**owner**	**species**	**sex**	**birth**	**death**
Fluffy	Harold	cat	f	1993-02-04	
Claws	Gwen	cat	m	1994-03-17	
Slim	Benny	snake	m	1996-04-29	1997-03-11
Buffy	Harold	dog	f	1989-05-13	

- A **multiset** is an unordered list (a set with duplicate elements allowed)
  - List: $[1, 1, 2, 3]$
  - Set: $\{1, 2, 3\}$
  - Multiset: $\{1, 1, 2, 3\}$

Tables in SQL

pet

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	1993-02-04	
Claws	Gwen	cat	m	1994-03-17	
Slim	Benny	snake	m	1996-04-29	1997-03-11
Buffy	Harold	dog	f	1989-05-13	

- An **attribute** (or **column**) is a typed data entry, present in each tuple in the relation
- Attributes must have an *atomic* data type in standard SQL, e.g. a number, a string, a date, . . . but *not* a list, set, table, . . .
- Each value in the table must have the correct type for the attribute, or it may be **NULL**, indicating that the value is not known or not available, e.g. the value of the **death** attribute for **Fluffy** is **NULL** (presumably because Fluffy is still alive)
- The number of attributes in a table gives the table's **arity**

Tables in SQL

pet

name	**owner**	**species**	**sex**	**birth**	**death**
Fluffy	Harold	cat	f	1993-02-04	
Claws	Gwen	cat	m	1994-03-17	
Slim	Benny	snake	m	1996-04-29	1997-03-11
Buffy	Harold	dog	f	1989-05-13	

- A **tuple** (or **row**) is a single entry in the table, having the attributes specified by the schema
- Also sometimes referred to as a **record**
- The number of rows in a table gives the table's **cardinality**

Data types in SQL

- Atomic types:
  - Characters: **CHAR(20)**, **VARCHAR(50)**
  - Numbers: **INT**, **BIGINT**, **SMALLINT**, **FLOAT**
  - Others: **MONEY**, **DATE**, **DATETIME**
- Every attribute must have an atomic type, i.e. tables are *flat*

Table schemas and table creation

- The **schema** of a table is its name, its attributes, and their types
- The table schema is specified when the table is created, e.g.

```
mysql> CREATE TABLE pet (name VARCHAR(20),
    -> owner VARCHAR(20), species VARCHAR(20),
    -> sex CHAR(1), birth DATE, death DATE);
```

Keys and Key constraints

- A *key* is a *minimal subset of attributes* that acts as a unique identifier for a tuple in a relation
- A key is an implicit constraint on which tuples can be in the relation
- If two tuples agree on the value of the key, they must be the same tuple!
- A possible key for **pet** is underlined below, assuming that there is no pet with the same name, owner, and birth date

pet key

pet(name VARCHAR(20), owner VARCHAR(20), species VARCHAR(20), sex CHAR(1), birth DATE, death DATE)

SQL queries — SELECT

- Basic form (there are many more variations)

  SELECT <attributes>
  FROM <one or more tables>
  WHERE <conditions>

- This is known as an **SFW** query, e.g.

```
  mysql> SELECT name, species, birth
      -> FROM pet
      -> WHERE owner = 'harold';
+--------+---------+------------+
| name   | species | birth      |
+--------+---------+------------+
| Fluffy | cat     | 1993-02-04 |
| Buffy  | dog     | 1989-05-13 |
+--------+---------+------------+
```

- Note it is the final semi-colon (**;**) that causes the command to be executed — this is generally the case when entering SQL commands

SQL queries — SELECT

- You can select *ALL* attributes from a table using *, e.g.

```
mysql> SELECT *
    -> FROM pet
    -> WHERE birth > ’1994-01-01’;
+-------+-------+---------+------+------------+------------+
| name  | owner | species | sex  | birth      | death      |
+-------+-------+---------+------+------------+------------+
| Claws | Gwen  | cat     | m    | 1994-03-17 | NULL       |
| Slim  | Benny | snake   | m    | 1996-04-29 | 1997-03-11 |
+-------+-------+---------+------+------------+------------+
```

Some syntax details (UNIX environment)

- SQL commands and attributes are *not* case-sensitive, e.g.
    - Same: SELECT, Select, select
    - Same: CREATE, Create, crEAtE
    - Same: BIRTH, Birth, birth
- Names of databases, tables, and values *are* case-sensitive, e.g.
    - Different: PET, Pet, pet
    - Different: 'HAROLD', 'Harold', 'harold'
- A common convention is to use UPPERCASE letters for SQL commands, e.g. SELECT, FROM, WHERE, CREATE and lowercase letter for the names of databases, tables, attributes …e.g. pet, owner
- Use single quotes for constants, e.g.

'abc'	YES
"abc"	NO

Exercise — SELECT

- Write a query to show the name and species of all female pets

Exercise — SELECT

- Write a query to show the name and species of all female pets
-
  ```
  mysql> SELECT name, species
      -> FROM pet
      -> WHERE sex = ’f’;
  +--------+----------+
  | name   | species  |
  +--------+----------+
  | Fluffy | cat      |
  | Buffy  | dog      |
  +--------+----------+
  ```

INSERT command

- The **INSERT** command is used to add a row to a table, e.g.

```
mysql> INSERT INTO pet
    -> VALUES (’Puffball’, ’Diane’, ’hamster’, ’f’,
    -> ’1999-03-30’, NULL);
Query OK, 1 row affected (0.04 sec)

mysql> SELECT * FROM pet;
+----------+--------+---------+------+------------+------------+
| name     | owner  | species | sex  | birth      | death      |
+----------+--------+---------+------+------------+------------+
| Fluffy   | Harold | cat     | f    | 1993-02-04 | NULL       |
| Claws    | Gwen   | cat     | m    | 1994-03-17 | NULL       |
| Slim     | Benny  | snake   | m    | 1996-04-29 | 1997-03-11 |
| Buffy    | Harold | dog     | f    | 1989-05-13 | NULL       |
| Puffball | Diane  | hamster | f    | 1999-03-30 | NULL       |
+----------+--------+---------+------+------------+------------+
```

ALTER TABLE and UPDATE commands

- The **ALTER TABLE** command can be used to add a column to a table, e.g.

```
mysql> ALTER TABLE pet ADD COLUMN weight INT;
Query OK, 0 rows affected (0.63 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM pet;
+----------+--------+---------+------+------------+------------+--------+
| name     | owner  | species | sex  | birth      | death      | weight |
+----------+--------+---------+------+------------+------------+--------+
| Fluffy   | Harold | cat     | f    | 1993-02-04 | NULL       |   NULL |
| Claws    | Gwen   | cat     | m    | 1994-03-17 | NULL       |   NULL |
| Slim     | Benny  | snake   | m    | 1996-04-29 | 1997-03-11 |   NULL |
| Buffy    | Harold | dog     | f    | 1989-05-13 | NULL       |   NULL |
| Puffball | Diane  | hamster | f    | 1999-03-30 | NULL       |   NULL |
+----------+--------+---------+------+------------+------------+--------+
```

- The **UPDATE** command can be used to update the value of an entry, e.g.

```
mysql> UPDATE pet SET weight = 2345 WHERE name = 'Fluffy';
Query OK, 1 row affected (0.04 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

New data set

- We can update our pet database with the weights of our pets, ending up with, e.g.

```
mysql> SELECT * FROM pet;
+----------+--------+---------+------+------------+------------+--------+
| name     | owner  | species | sex  | birth      | death      | weight |
+----------+--------+---------+------+------------+------------+--------+
| Fluffy   | Harold | cat     | f    | 1993-02-04 | NULL       |   2345 |
| Claws    | Gwen   | cat     | m    | 1994-03-17 | NULL       |   4321 |
| Slim     | Benny  | snake   | m    | 1996-04-29 | 1997-03-11 |  10123 |
| Buffy    | Harold | dog     | f    | 1989-05-13 | NULL       |   6543 |
| Puffball | Diane  | hamster | f    | 1999-03-30 | NULL       |     45 |
+----------+--------+---------+------+------------+------------+--------+
```

- We're assuming here that all weights are expressed as an integer number of grams
- We'll use this table in our examples from now on

Aggregating data

- Sometimes we want a summary of the data rather than the details
- SQL has several aggregate functions to help with this, e.g.

COUNT	counts the number of rows in a table or view
AVG	calculates the average of a set of values
MIN	gets the minimum value in a set of values
MAX	gets the maximum value in a set of values
SUM	calculates the sum of values

- Examples to follow . . .

Aggregate function examples

- COUNT

```
mysql> SELECT COUNT(owner) FROM pet;
+--------------+
| COUNT(owner) |
+--------------+
|            5 |
+--------------+
```

- Why is the answer 5?
- What if we want the number of *different* owners?

```
mysql> SELECT COUNT(DISTINCT owner) FROM pet;
+-----------------------+
| COUNT(DISTINCT owner) |
+-----------------------+
|                     4 |
+-----------------------+
```

Aggregate function examples

- AVG

```
mysql> SELECT AVG(weight) FROM pet;
+-------------+
| AVG(weight) |
+-------------+
|   4675.4000 |
+-------------+
```

- Use GROUP BY to average over an attribute

```
mysql> SELECT species, AVG(weight)
    -> FROM pet GROUP BY species;
+---------+-------------+
| species | AVG(weight) |
+---------+-------------+
| cat     |   3333.0000 |
| dog     |   6543.0000 |
| hamster |     45.0000 |
| snake   |  10123.0000 |
+---------+-------------+
```

Aggregate function examples

- MIN

```
mysql> SELECT MIN(weight) FROM pet;
+-------------+
| MIN(weight) |
+-------------+
|          45 |
+-------------+
```

- Use GROUP BY to get the minimum over an attribute

```
mysql> SELECT owner, MIN(weight) FROM pet GROUP BY owner;
+--------+-------------+
| owner  | MIN(weight) |
+--------+-------------+
| Benny  |       10123 |
| Diane  |          45 |
| Gwen   |        4321 |
| Harold |        2345 |
+--------+-------------+
```

Aggregate function examples

- SUM

```
mysql> SELECT SUM(weight) FROM pet;
+-------------+
| SUM(weight) |
+-------------+
|       23377 |
+-------------+
```

- Use GROUP BY to get the sum over an attribute

```
mysql> SELECT owner, SUM(weight) FROM pet GROUP BY owner;
+--------+-------------+
| owner  | SUM(weight) |
+--------+-------------+
| Benny  |       10123 |
| Diane  |          45 |
| Gwen   |        4321 |
| Harold |        8888 |
+--------+-------------+
```

Exercise

- Write a query to list every species and the total weight of pets of that species.

Exercise

- Write a query to list every species and the total weight of pets of that species.
- Answer

```
mysql> SELECT species, SUM(weight)
    -> FROM pet GROUP BY species;
+---------+-------------+
| species | SUM(weight) |
+---------+-------------+
| cat     |        6666 |
| dog     |        6543 |
| hamster |          45 |
| snake   |       10123 |
+---------+-------------+
```

Ordering data

- If we want to order the results of our last exercise by total weight, we can do it like this . . .
- ORDER BY

```
mysql> SELECT species, SUM(weight)
    -> FROM pet GROUP BY species
    -> ORDER BY SUM(weight);
+---------+-------------+
| species | SUM(weight) |
+---------+-------------+
| hamster |          45 |
| dog     |        6543 |
| cat     |        6666 |
| snake   |       10123 |
+---------+-------------+
```

- Add `DESC` after `ORDER BY SUM(weight)` to get the results in descending order of weight

Getting data from multiple tables

- Suppose we decide that we want to record information about special events in the lives of our pets.
- We create a new table called `event` to store this information, e.g.

```
mysql> SELECT * FROM event;
+--------+------------+----------+-----------------------------+
| name   | date       | type     | remark                      |
+--------+------------+----------+-----------------------------+
| Fluffy | 1995-05-15 | litter   | 4 kittens, 3 female, 1 male |
| Claws  | 1995-09-27 | litter   | 6 kittens, 2 female, 4 male |
| Claws  | 1998-03-17 | birthday | new flea collar             |
| Buffy  | 1993-06-23 | litter   | 5 puppies, 2 female, 3 male |
| Buffy  | 1994-06-19 | litter   | 3 puppies, 3 female         |
| Slim   | 1993-08-03 | vet      | swallowed tennis ball       |
+--------+------------+----------+-----------------------------+
```

- What can we do if we need a list of all those owners whose pets have had a litter?

Getting data from multiple tables

- The litter information is in the event table
- The owner information is in the pet table
- We need both tables to get the information about owners whose pets have had a litter

```
mysql> SELECT owner
    -> FROM pet JOIN event
    -> ON (pet.name = event.name)
    -> WHERE event.type = ’litter’;
+--------+
| owner  |
+--------+
| Harold |
| Gwen   |
| Harold |
| Harold |
+--------+
```

- Notice the use of the keyword JOIN to combine the data from the pet and the event tables (giving the CROSS PRODUCT)
- The ON clause ensures that we only have rows in the result where the pet name is the same as the event name

Getting data from multiple tables

- What if we just want a list of all the *different* owners whose pets have had a litter?

```
mysql> SELECT DISTINCT owner
    -> FROM pet JOIN event
    -> ON (pet.name = event.name)
    -> WHERE event.type = ’litter’;
+--------+
| owner  |
+--------+
| Harold |
| Gwen   |
+--------+
```

- Notice the use of the keyword DISTINCT

Using sub-queries

- A query returns a view (table) as its result
- The result can be used like a table in a larger query
- E.g. to find the owners and the names of their lightest pet to have had a litter, we can try . . .

```
mysql> SELECT pet.owner, pet.name, weight, type
    -> FROM pet JOIN event ON (pet.name = event.name)
    ->          JOIN (SELECT owner, MIN(weight) AS minw
    ->                FROM pet GROUP BY owner) weights
    ->          ON (weights.owner = pet.owner AND weights.minw = pet.weight)
    -> WHERE event.type = 'litter';
+--------+--------+--------+--------+
| owner  | name   | weight | type   |
+--------+--------+--------+--------+
| Harold | Fluffy |   2345 | litter |
+--------+--------+--------+--------+
```

- Notice how we are able to give a name, `weights`, to the result of the sub-query and then to use it in the specification of a JOIN
- This is really beyond the scope of this short introduction to SQL but points the way to what is possible

Acknowledgments

- The examples here are based on the examples in the MySQL tutorial
- Some of the other material is adapted from
  - Spector, P., Introduction to SQL, Slides, University of California, Berkeley
  - Bailis, P., CS145: Introduction to Databases, Stanford University