122COM: Databases

David Croft

Coventry University

david.croft@coventry.ac.uk

2017

Overview

1 Databases
- SQL
- SQLite

2 Code
- Dynamic queries
  - SQL injection

3 Recap

4 Further reading

SQL

C

Database (noun) - a collection of information that is organized so that it can easily be accessed, managed, and updated.

SQL

C

Database (noun) - a collection of information that is organized so that it can easily be accessed, managed, and updated.

- Pronounced S-Q-L or Sequel.
  - Structured Query Language.
- Used to query relational databases.
- Theoretically it doesn’t matter what underlying database is.
  - MS SQL Server, Oracle, PostgreSQL, MySQL, SQLite.
  - In reality lots of minor variations.

Relational Databases

Built around tables.

- Can be imagined like a spreadsheet.

Row/record →

id	forename	surname	job
0	Malcolm	Reynolds	Captain
4	Zoe	Washburne	Co-captain
11	Hoban	Washburne	Pilot
23	Kaywinnet	Frye	Mechanic

↑

Column/attribute

C

Many types of query.

- SELECT - Get information from the database.
- INSERT - Add information to the database.
- DELETE - Remove information.

Also used for database administration.

- CREATE - Create a whole new table/schema/function.
- ALTER - Modify a table/schema/function.
- DROP - Delete a whole table/schema/function.

SELECT

C

Used to retrieve information from the database.

SELECT

C

Used to retrieve information from the database.

id	forename	surname	job
0	Malcolm	Reynolds	Captain
4	Zoe	Washburne	Co-captain
11	Hoban	Washburne	Pilot
23	Kaywinnet	Frye	Mechanic

SELECT

C

Used to retrieve information from the database.

id	forename	surname	job
0	Malcolm	Reynolds	Captain
4	Zoe	Washburne	Co-captain
11	Hoban	Washburne	Pilot
23	Kaywinnet	Frye	Mechanic

```
SELECT * FROM staff;
```

* means everything.

SELECT

C

Used to retrieve information from the database.

id	forename	surname	job
0	Malcolm	Reynolds	Captain
4	Zoe	Washburne	Co-captain
11	Hoban	Washburne	Pilot
23	Kaywinnet	Frye	Mechanic

```
SELECT * FROM staff;
```

∗ means everything.

#	id	forename	surname	job
1	0	Malcolm	Reynolds	Captain
2	4	Zoe	Washburne	Co-captain
3	11	Hoban	Washburne	Pilot
4	23	Kaywinnet	Frye	Mechanic

SELECT

C

Used to retrieve information from the database.

SELECT

C

Used to retrieve information from the database.

id	forename	surname	job
0	Malcolm	Reynolds	Captain
4	Zoe	Washburne	Co-captain
11	Hoban	Washburne	Pilot
23	Kaywinnet	Frye	Mechanic

SELECT

C

Used to retrieve information from the database.

id	forename	surname	job
0	Malcolm	Reynolds	Captain
4	Zoe	Washburne	Co-captain
11	Hoban	Washburne	Pilot
23	Kaywinnet	Frye	Mechanic

```
SELECT * FROM staff WHERE surname = "Washburne";
```

Only return the records `WHERE` something is true.

SELECT

C

Used to retrieve information from the database.

id	forename	surname	job
0	Malcolm	Reynolds	Captain
4	Zoe	Washburne	Co-captain
11	Hoban	Washburne	Pilot
23	Kaywinnet	Frye	Mechanic

```
SELECT * FROM staff WHERE surname = "Washburne";
```

Only return the records `WHERE` something is true.

#	id	forename	surname	job
1	4	Zoe	Washburne	Co-captain
2	11	Hoban	Washburne	Pilot

count()

I

What if we want to now how many records there are?

- count() function.
- More efficient.
  - Minimum amount of data.

count()

I

What if we want to now how many records there are?

- count() function.
- More efficient.
  - Minimum amount of data.

id	forename	surname	job
0	Malcolm	Reynolds	Captain
4	Zoe	Washburne	Co-captain
11	Hoban	Washburne	Pilot
23	Kaywinnet	Frye	Mechanic

count()

I

What if we want to now how many records there are?

- count() function.
- More efficient.
  - Minimum amount of data.

id	forename	surname	job
0	Malcolm	Reynolds	Captain
4	Zoe	Washburne	Co-captain
11	Hoban	Washburne	Pilot
23	Kaywinnet	Frye	Mechanic

```
SELECT count(*) FROM staff;
```

## count()

I

What if we want to now how many records there are?

- count() function.
- More efficient.
  - Minimum amount of data.

id	forename	surname	job
0	Malcolm	Reynolds	Captain
4	Zoe	Washburne	Co-captain
11	Hoban	Washburne	Pilot
23	Kaywinnet	Frye	Mechanic

```
SELECT count(*) FROM staff;
```

#	count(*)
1	4

INSERT

I

Used to add information to the database.

INSERT

Used to add information to the database.

id	forename	surname	job
0	Malcolm	Reynolds	Captain
4	Zoe	Washburne	Co-captain
11	Hoban	Washburne	Pilot
23	Kaywinnet	Frye	Mechanic

INSERT

I

Used to add information to the database.

id	forename	surname	job
0	Malcolm	Reynolds	Captain
4	Zoe	Washburne	Co-captain
11	Hoban	Washburne	Pilot
23	Kaywinnet	Frye	Mechanic

```
INSERT INTO staff VALUES (42, 'Simon', 'Tam', 'Doctor');
```

INSERT

I

Used to add information to the database.

id	forename	surname	job
0	Malcolm	Reynolds	Captain
4	Zoe	Washburne	Co-captain
11	Hoban	Washburne	Pilot
23	Kaywinnet	Frye	Mechanic

```
INSERT INTO staff VALUES (42, 'Simon', 'Tam', 'Doctor');
```

id	forename	surname	job
0	Malcolm	Reynolds	Captain
4	Zoe	Washburne	Co-captain
11	Hoban	Washburne	Pilot
23	Kaywinnet	Frye	Mechanic
42	Simon	Tam	Doctor

INSERT again

I

Don’t have to supply values for all the columns.

- Depends on the table design.

INSERT again

I

Don't have to supply values for all the columns.

- Depends on the table design.

```
INSERT INTO staff (forename, id, surname)
    VALUES ('River', 43, 'Tam');
```

INSERT again

Don't have to supply values for all the columns.

- Depends on the table design.

```
INSERT INTO staff (forename, id, surname)
        VALUES ('River', 43, 'Tam');
```

id	forename	surname	job
0	Malcolm	Reynolds	Captain
4	Zoe	Washburne	Co-captain
11	Hoban	Washburne	Pilot
23	Kaywinnet	Frye	Mechanic
42	Simon	Tam	Doctor
43	River	Tam	

Databases

C

Why use databases at all?
Why not just use dictionaries and lists or similar?

Databases...

- Have structure.
  - Easy to organise the data.
- Scale.
  - Can handle a LOT of data.
- Multi-user.
  - Can have lots of people working on the same data.
- Fault tolerant.
  - Can recover if things go wrong.

SQLite

Using SQLite3 in labs.

- Not a fully featured database.
  - But has all the basic features.
  - SQL.
- Good for small/non-urgent databases.
  - $\leq$ gigabytes of data.
- Efficient
  - Don’t need to waste resources on a ’real’ database.
- Convenient.
  - Don’t need to install, configure, manage a ’real’ database.
  - Portable, 1 file.
- No network.
  - Single user only.

Databases
SQL
SQLite
Code
Dynamic queries
SQL injection
Recap
Further reading

Python

C

How to use SQL queries in Python?

```
import sqlite3 as sql                                   # sqlite module

con = sql.connect( 'firefly.sqlite' )                  # open database
cur = con.cursor()

cur.execute( '''SELECT * FROM staff;''' )              # run query
for row in cur:                                          # loop over results
    print( row )

con.close()                                              # close database
```

lec_select.py

Python

C

How to use SQL queries in Python?

```python
import sqlite3 as sql                                    # sqlite module

con = sql.connect( 'firefly.sqlite' )                   # open database
cur = con.cursor()

cur.execute( '''SELECT * FROM staff;''' )               # run query
for row in cur:                                          # loop over results
    print( row )

con.close()                                              # close database
```

lec_select.py

```
(0, 'Malcolm', 'Reynolds', 'Captain')
(4, 'Zoe', 'Washburne', 'Co-captain')
(11, 'Hoban', 'Washburne', 'Pilot')
(23, 'Kaywinnet', 'Frye', 'Mechanic')
```

C++ I

How to use SQL queries in C++?

```
#include "libsqlite.hpp"                             // sqlite library

int main()
{
    sqlite::sqlite db( "firefly.sqlite" );          // open database

    auto cur = db.get_statement();                    // create query
    cur->set_sql( "SELECT * FROM staff;" );
    cur->prepare();                                    // run query

    while( cur->step() )                             // loop over results
        cout « cur->get_int(0) « " " « cur->get_text(1) « endl;
}
```

lec_select.cpp

```
0 Malcolm
4 Zoe
11 Hoban
23 Kaywinnet
```

Break

Static queries

I

So far looked at static queries.

- Same query is run every time.
- Real power is in dynamic queries.
    - Code creates changes the SQL to ask new questions.

Dynamic queries

I

```python
import sqlite3 as sql

con = sql.connect('firefly.sqlite')
cur = con.cursor()

question = input('Who is the...')

cur.execute('''SELECT forename, surname FROM staff
               WHERE job = ?;''', (question,))

for row in cur:
    print('%s %s' % row)
```

lec_dynamic.py

```
Who is the...Captain
Malcolm Reynolds
```

Dynamic queries C++

I

Using sqlitepp.

- 3rd party wrapper around default SQLite3 API.
- Simplified use.

```cpp
sqlite::sqlite db( "firefly.sqlite" );

string question;
cout « "Who is the...";
cin » question;

auto s = db.get_statement();
s->set_sql( "SELECT forename, surname FROM staff "
             "WHERE job = ?;" );
s->prepare();
s->bind( 1, question );

while( s->step() )
{
    string forename = s->get_text(0);
    string surname = s->get_text(1);
    cout « forename « " " « surname « endl;
}
```

lec_dynamic.cpp

Bad dynamic queries

Dynamic queries should **ALWAYS** use placeholders (i.e. ?).

```
cur.execute('''SELECT forename, surname FROM staff
               WHERE job = ?;''', (question,))
```

Dynamic queries must **NEVER** be created by manipulating strings.

```
cur.execute('''SELECT forename, surname FROM staff
               WHERE job = "%s";''' % question )
cur.execute('''SELECT forename, surname FROM staff
               WHERE job = "{}";'''.format( question) )
```

- User could input anything, e.g. SQL commands!.
  - `Captain"; DROP TABLE staff; -`
- Sanitise your inputs.

Bad dynamic queries

Dynamic queries should **ALWAYS** use placeholders (i.e. ?).

```
cur.execute('''SELECT forename, surname FROM staff
               WHERE job = ?;''', (question,))
```

Dynamic queries must **NEVER** be created by manipulating strings.

```
cur.execute('''SELECT forename, surname FROM staff
               WHERE job = "%s";''' % question )
cur.execute('''SELECT forename, surname FROM staff
               WHERE job = "{}";'''.format( question) )
```

- User could input anything, e.g. SQL commands!.
  - `Captain"; DROP TABLE staff; -`
- Sanitise your inputs.
- Always use placeholders.

Bad dynamic queries

I

Dynamic queries should **ALWAYS** use placeholders (i.e. ?).

```
cur.execute('''SELECT forename, surname FROM staff
               WHERE job = ?;''', (question,))
```

Dynamic queries must **NEVER** be created by manipulating strings.

```
cur.execute('''SELECT forename, surname FROM staff
               WHERE job = "%s";''' % question )
cur.execute('''SELECT forename, surname FROM staff
               WHERE job = "{}";'''.format( question) )
```

- User could input anything, e.g. SQL commands!.
    - `Captain"; DROP TABLE staff; -`
- Sanitise your inputs.
- Always use placeholders.
    - No exceptions.

Bad dynamic queries

I

Dynamic queries should **ALWAYS** use placeholders (i.e. ?).

```
cur.execute('''SELECT forename, surname FROM staff
               WHERE job = ?;''', (question,))
```

Dynamic queries must **NEVER** be created by manipulating strings.

```
cur.execute('''SELECT forename, surname FROM staff
               WHERE job = "%s";''' % question )
cur.execute('''SELECT forename, surname FROM staff
               WHERE job = "{}";'''.format( question) )
```

- User could input anything, e.g. SQL commands!.
  - `Captain"; DROP TABLE staff; -`
- Sanitise your inputs.
- Always use placeholders.
  - No exceptions.
  - NO EXCEPTIONS!

Bad dynamic queries

I

Dynamic queries should **ALWAYS** use placeholders (i.e. ?).

```
cur.execute('''SELECT forename, surname FROM staff
               WHERE job = ?;''', (question,))
```

Dynamic queries must **NEVER** be created by manipulating strings.

```
cur.execute('''SELECT forename, surname FROM staff
               WHERE job = "%s"''' % question)
cur.execute('''SELECT forename, surname FROM staff
               WHERE job = "{}";'''.format( question) )
```

NO EXCEPTIONS

- User could input anything, e.g. SQL commands!.
    - `captain"; DROP TABLE staff; -`
- Sanitise your inputs.
- Always use placeholders.
    - No exceptions.
    - NO EXCEPTIONS!

SQL injection

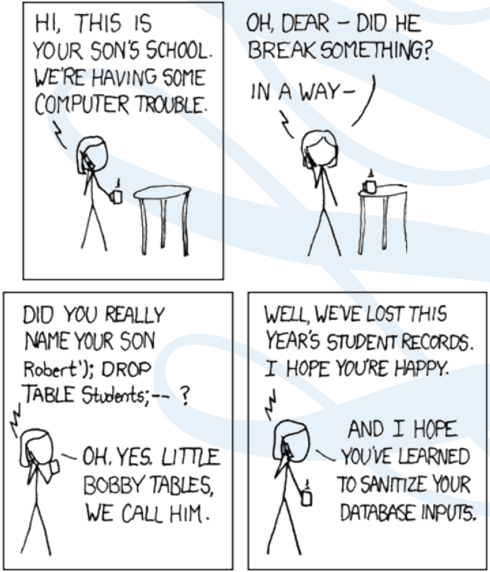
A

Around since at least 1998.

Notable SQL injection attacks.

- 2015 TalkTalk - 160,000 customers' details.
- 2014 Hold security - found 420,000 vulnerable websites.
- 2012 Yahoo - 450,000 logins.
- 2011 MySql - mysql.com compromised.
- 2008 Heartland Payment - 134,000,000 credit cards.

Many, many more.


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

https://xkcd.com/327/

Recap

- SQL used to query databases.
- Databases are...
  - fault tolerant.
  - multi user.
  - scalable.
- Always use place holders in dynamic queries.
  - Say no to SQL injection!

Why do I care?

- Everyone
  - Structured Query Language (SQL) is widely used, most in demand language[1].
  - Should be aware of and able to defend against SQL injection.
  - Experience in using 3rd party libraries/modules in software.
- Computing - SQL is a vital for much of the web. Heard of LAMP servers?, the M is for MySQL.
- Ethical Hackers - need to understand SQL injection.
- ITB - SQL is widely used in business applications, especially for generating reports.
- Games Tech & MC- SQL is used in games, i.e. for save games.

[1] According to Indeed.com

Further reading

A

- Introduction to SQL - `http://www.w3schools.com/sql/sql_intro.asp`
- SQL injection hall of shame - `http://codecurmudgeon.com/wp/sql-injection-hall-of-shame/`
- Efficient inserting - the `executemany()` method.

The End