

122COM: Databases

David Croft

Coventry University

david.croft@coventry.ac.uk

2016

Overview

1 Databases

- SQL
- SQLite

2 Python

- Dynamic queries
 - SQL injection
- Efficient inserting

3 Recap

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.
- 4th generation language.
- Used to query relational databases.
- Doesn't matter what underlying database is.
 - MS SQL Server, Oracle, PostgreSQL, MySQL, SQLite.
 - In reality, minor variations.

Relational Databases

C

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

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.

Used to retrieve information from the database.

Databases

SQL

SQLite

Python

Dynamic queries

SQL injection

Efficient inserting

Recap

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

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;
```


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;
```

#	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

Used to retrieve information from the database.

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

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';
```

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';
```

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



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

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



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

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;
```


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



Used to add information to the database.

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

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');
```

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

I

Used to add information to the database.



Used to add information to the database.

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

Used to add information to the database.

```
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	

Why use databases at all?

- Databases...
 - have structure.
 - scale.
 - multi-user.
 - fault tolerant.
- Can include SQL queries in other languages.



Using SQLite3 in labs.

- Not really a database.
 - Behaves like one.
 - 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, managed a 'real' database.
 - Portable, 1 file.
- No network.
 - Single user only.

How to use SQL queries in Python?

```
import sqlite3 as sql

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

cur.execute(''SELECT * FROM staff;'')
for row in cur:
    print(row)

con.close()
```

lec_select.py

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

Multiple queries.

```
import sqlite3 as sql

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

cur.execute('SELECT count(*) FROM staff;')
print(cur.fetchone()[0])

cur.execute('SELECT * FROM staff;')
for row in cur:
    pass

con.close()
```

lec_multi.py

Static queries



So far looked at static queries.

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

```
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 should ALWAYS use place holders (i.e. ?).

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

Dynamic queries should NEVER be created by manipulating strings.

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

- User could input anything.
 - Captain"; DROP TABLE staff; --
- Sanitise inputs.

Dynamic queries should ALWAYS use place holders (i.e. ?).

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

Dynamic queries should NEVER be created by manipulating strings.

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

- User could input anything.
 - Captain"; DROP TABLE staff; --
- Sanitise inputs.
- Always use placeholders.

Dynamic queries should ALWAYS use place holders (i.e. ?).

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

Dynamic queries should NEVER be created by manipulating strings.

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

- User could input anything.
 - Captain"; DROP TABLE staff; --
- Sanitise inputs.
- Always use placeholders.
 - No exceptions.

Dynamic queries should ALWAYS use place holders (i.e. ?).

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

Dynamic queries should NEVER be created by manipulating strings.

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

- User could input anything.
 - Captain"; DROP TABLE staff; --
- Sanitise inputs.
- Always use placeholders.
 - No exceptions.
 - NO EXCEPTIONS!

Bad dynamic queries

1

Dynamic queries should ALWAYS use place holders (i.e. ?).

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

Dynamic queries should NEVER be created by concatenating strings.

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

Is it possible to use anything.

- Captain": DROP TABLE staff; --

- Sanitise inputs.

- Always use placeholders.

- No exceptions.

- NO EXCEPTIONS!

NO EXCEPTIONS

Around since at least 1998.

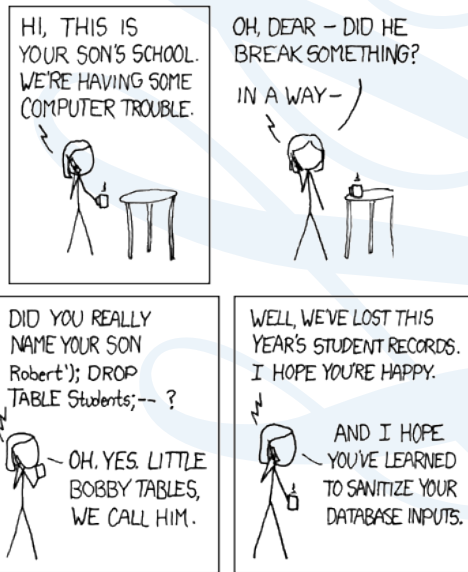
Notable SQL injection attacks.

- 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.

SQL injection

A



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

cur.execute('''INSERT INTO staff (forename, surname)
              VALUES (?,?)''', ('River', 'Tam'))

con.commit()
con.close()
```

lec_single_insert.py

commit() command.

- Have modified database.
- Tell database to save changes.
- `revert()` command to undo everything done since `commit()`.

What is you want to insert a lot of records?

- Could run multiple small INSERT statements.
 - Slow.
- Could run one big INSERT statement.
 - Fast.

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

people = [('Simon', 'Tam', 'Doctor'), ('River', 'Tam', None)]

cur.executemany('INSERT INTO staff (forename, surname, job)
                VALUES (?, ?, ?)', people)

cur.commit()
con.close()
```

Quiz

<https://tophat.com/>
Join code: 094769

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!
 - Inserting data
 - Avoid small inserts.
 - Use big inserts.

The End