

NEWS COMPUTING

Top Programming Languages 2022 >

Python's still No. 1, but employers love to see SQL skills

BY STEPHEN CASS | 23 AUG 2022 | 4 MIN READ | 📌



Relational data

- SQL, SQLite
- pandas

Two tables

Table: city		
name	population	established
'Copenhagen'	775033	800
'Aarhus'	273077	750
'Berlin'	3711930	1237
'Munich'	1464301	1158
'Reykjavik'	126100	874
'Washington D.C.'	693972	1790
'New Orleans'	343829	1718
'San Francisco'	884363	1776

Table: country			
name	population	area	capital
'Denmark'	5748769	42931	'Copenhagen'
'Germany'	82800000	357168	'Berlin'
'USA'	325719178	9833520	'Washington, D.C.'
'Iceland'	334252	102775	'Reykjavik'

SQL

- SQL = Structured Query **Language**
- **Database = collection of tables**
- ANSI and ISO standards since 1986 and 1987, respectively
- Widespread used SQL databases (can handle many tables/rows/users):
Oracle, MySQL, Microsoft SQL Server, PostgreSQL and IBM DB2
- **SQLite** is a very lightweight version storing a database in one file
- SQLite is included in both iOS and Android mobil phones

Table: country			
name	population	area	capital
'Denmark'	5748769	42931	'Copenhagen'
'Germany'	82800000	357168	'Berlin'
'USA'	325719178	9833520	'Washington, D.C.'
'Iceland'	334252	102775	'Reykjavik'



The Course “Database Systems” gives a more in-depth introduction to SQL (MySQL)

SQL examples

Table: country			
name	population	area	capital
'Denmark'	5748769	42931	'Copenhagen'
'Germany'	82800000	357168	'Berlin'
'USA'	325719178	9833520	'Washington, D.C.'
'Iceland'	334252	102775	'Reykjavik'

- CREATE TABLE country (name, population, area, capital)
- INSERT INTO country VALUES ('Denmark', 5748769, 42931, 'Copenhagen')
- UPDATE country SET population=5748770 WHERE name='Denmark'
- SELECT name, capital FROM country WHERE population >= 1000000
> [('Denmark', 'Copenhagen'), ('Germany', 'Berlin'), ('USA', 'Washington, D.C.')]
- SELECT * FROM country WHERE capital = 'Berlin'
> [('Germany', 82800000, 357168, 'Berlin')]
- SELECT country.name, city.name, city.established FROM city, country
WHERE city.name=country.capital AND city.population < 500000
> ('Iceland', 'Reykjavik', 874), ('USA', 'Washington, D.C.', 1790)
- DELETE FROM country WHERE name = 'Germany'
- DROP TABLE country

sqlite-example.py

```
import sqlite3

connection = sqlite3.connect('example.sqlite') # creates file if necessary
c = connection.cursor()

countries = [('Denmark', 5748769, 42931, 'Copenhagen'),
             ('Germany', 82800000, 357168, 'Berlin'),
             ('USA', 325719178, 9833520, 'Washington, D.C.'),
             ('Iceland', 334252, 102775, 'Reykjavik')]

cities = [('Copenhagen', 775033, 800),
          ('Aarhus', 273077, 750),
          ('Berlin', 3711930, 1237),
          ('Munich', 1464301, 1158),
          ('Reykjavik', 126100, 874),
          ('Washington, D.C.', 693972, 1790),
          ('New Orleans', 343829, 1718),
          ('San Francisco', 884363, 1776)]

c.execute('CREATE TABLE country (name, population, area, capital)')
c.execute('CREATE TABLE city (name, population, established)')
c.executemany('INSERT INTO country VALUES (?, ?, ?, ?)', countries)
c.executemany('INSERT INTO city VALUES (?, ?, ?)', cities)

connection.commit() # make sure data is saved to database
connection.close()
```

SQLite

SQLite query examples

sqlite-example.py

```
for row in c.execute('SELECT * FROM city'):
    print(row)

for row in c.execute(
    '''SELECT country.name, city.name, city.established FROM city, country
       WHERE city.name=country.capital AND city.population < 700000'''):
    print(row)
```

Python shell

```
| ('Copenhagen', 775033, 800)
| ('Aarhus', 273077, 750)
| ('Berlin', 3711930, 1237)
| ('Munich', 1464301, 1158)
| ('Reykjavik', 126100, 874)
| ('Washington, D.C.', 693972, 1790)
| ('New Orleans', 343829, 1718)
| ('San Francisco', 884363, 1776)
| ('Iceland', 'Reykjavik', 874)
| ('USA', 'Washington, D.C.', 1790)
```

SQL injection

Right way

```
c.execute('INSERT INTO users VALUES (?)', (user,))
```

sqlite-unsafe-example.py

```
import sqlite3
connection = sqlite3.connect('users.sqlite')
c = connection.cursor()
c.execute('CREATE TABLE users (name)')
while True:
    user = input("New user: ")
    c.executescript('INSERT INTO users VALUES ("%s")' % user)
    connection.commit()
    print(list(c.execute('SELECT * FROM users')))
```

can execute a string
containing several
SQL statements



Insecure: NEVER
use % on user input

Python shell

```
> New user: gerth
| [('gerth',)]
> New user: guido
| [('gerth',), ('guido',)]
> New user: evil"); DROP TABLE users; --
| sqlite3.OperationalError: no such table: users
```

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.

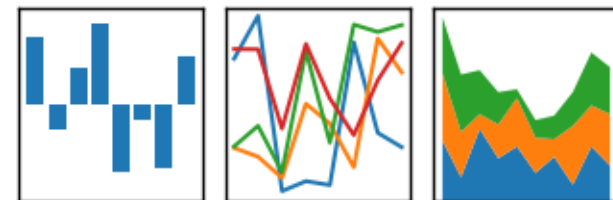


AND I HOPE
YOU'VE LEARNED
TO SANITIZE YOUR
DATABASE INPUTS.

Pandas

- Comprehensive Python library for data manipulation and analysis, in particular tables and time series
- Pandas **data frames** = tables
- Supports interaction with SQL, CSV, JSON, ...
- Integrates with Jupyter, numpy, matplotlib, ...

pandas
 $y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$



Reading tables

students.csv

Name	City
"Donald Duck"	"Copenhagen"
"Goofy"	"Aarhus"
"Mickey Mouse"	"Aarhus"

- Pandas provide functions for reading different data formats, e.g. SQLite and .csv files, into pandas.DataFrames

pandas-example.py

```
import pandas as pd
import sqlite3
connection = sqlite3.connect("example.sqlite")
countries = pd.read_sql_query("SELECT * FROM country", connection)
cities = pd.read_sql_query("SELECT * FROM city", connection)
students = pd.read_csv("students.csv")
students.to_sql('students', connection, if_exists='replace')
print(students)
```

Python shell

	Name	City
0	Donald Duck	Copenhagen
1	Goofy	Aarhus
2	Mickey Mouse	Aarhus

Selecting columns and rows

Table: country			
name	population	area	capital
'Denmark'	5748769	42931	'Copenhagen'
'Germany'	82800000	357168	'Berlin'
'USA'	325719178	9833520	'Washington, D.C.'
'Iceland'	334252	102775	'Reykjavik'

Python shell

```
> countries['name'] # select column
> countries.name # same as above
> countries[['name', 'capital']] # select multiple columns, note double-[]
> countries.head(2) # first 2 rows
> countries[1:3] # slicing rows, rows 1 and 2
> countries[:2] # slicing rows, rows 0 and 1
> countries.at[1, 'area'] # indexing cell by (row, column name)
> cities[(cities['name'] == 'Berlin') | (cities['name'] == 'Munich')] # select rows
> pd.DataFrame([[1, 2], [3, 4], [5, 6]], columns=['x', 'y']) # create DF from list
> pd.DataFrame(np.random.random((3, 2)), columns=['x', 'y']) # from numpy
> ...
```

Merging and creating a new column

`pandas-example.py`

```
res = pd.merge(countries, cities, left_on="capital", right_on="name")

res.rename(columns={'name_x': 'country'})

res['%pop in capital'] = res['population_y'] / res['population_x']

res.sort_values('%pop in capital', ascending=False)

print(res[['country', '%pop in capital']])
```

Python shell

	country	country	%pop in capital
3	Iceland	Iceland	0.377260
0	Denmark	Denmark	0.134817
1	Germany	Germany	0.044830
2	USA	USA	0.002131

Googlefinance > Pandas > Matplotlib

googlefinance-example.py

```
from googlefinance.client import get_price_data # pip install googlefinance.client
param = {
    'q': "GOOGL", # Stock symbol (ex: "AAPL", "MSFT", "FB")
    'i': "86400", # Interval size in seconds ("86400" = 1 day intervals)
    'x': "NASDAQ", # Stock exchange symbol on which stock is listed (ex: "NASDAQ")
    'p': "1Y" # Period (Ex: "1Y" = 1 year)
}
df = get_price_data(param) # get price data as dataframe
import matplotlib.pyplot as plt
plt.plot(df['Close'])
plt.show()
```

