ToiletPaper #140

SQL Queries ohne Datenbank auf CSV-Dateien ausführen mit q

Autoren: Fionn Fuchs & Maximilian Konzack / Werkstudent & Software Engineer / Standort Leipzig

✗ Problem

Hast du schon einmal auf eine CSV-Datei auf deinem Bildschirm gestarrt und gehofft, es wäre eine Datenbank, auf der du einfach SQL Queries ausführen kannst? Uns ging es auf jeden Fall schon oft so, dass wir nicht direkt CSVs verarbeiten wollten, sondern lieber SQL-Anfragen.

✓ Lösung

Mit q [1] lassen sich herkömmliche SQL Queries auf CSVs und anderen tabellenartigen Dateien ausführen, ohne dafür zusätzlich eine Datenbank anzubinden oder gar SQLite zu verwenden. Bei Bedarf kann q sogar eine SQLite-Datenbank exportieren. Die gängigsten SQL Queries werden von q unterstützt. Bemerkenswert ist, dass q sich als Unix-Befehl nutzen lässt. Dabei liest q standardmäßig von STDIN und schreibt nach STDOUT, was eine Verkettung mit anderen Unix-Befehlen ermöglicht.

Installation:
Alle gängigen Betriebssysteme werden unterstützt und können über den üblichen Weg installiert werden [2].
Disclaimer: Wir haben q nur über homebrew auf macOS getestet.

```
brew install q
```

➔ Beispiel

Beispiel 1

Nehmen wir die folgende Tabelle als CSV-Datei:

```
ID,name,geburtsjahr,haarfarbe
1,Max Mustermann,1994,braun
2,Lars Agne,1983,blond
3,Otto Normal,1995,braun
4,Lieschen Müller,1987,schwarz
5,John Doe,1980,braun
```

Wir möchten herausfinden, wie viele Personen braune Haare haben. Bei so einer kleinen Tabelle ist das natürlich trivial, aber in echten Datensätzen wäre das deutlich umfangreicher und unübersichtlicher.

Mit q führen wir direkt eine SQL Query auf dieser CSV-Datei aus, um anhand der Haarfarbe zu selektieren. Wir nehmen an, dass das ; der Zeichentrenner ist und dass die erste Zeile des CSVs die Spaltennamen beschreibt. Deshalb benutzen wir *-d*, um den Delimiter zu setzen und *-H*, um die erste Zeile zu überspringen.

```
$> q -H -d ";" "SELECT COUNT(ID) FROM personen.csv WHERE haarfarbe = 'braun'"

- Result -
3
```

Beispiel 2

Mit q lassen sich auch mehrere CSV-Dateien über Joins verbinden. Dazu nehmen wir zusätzlich zur obigen CSV aus Beispiel 1 noch diese CSV-Datei zur Hand:

```
ID,email
1,max.mustermann@example.com
2,lars.agne@example.com
3,otto.normal@example.com
4,lieschen.mueller@example.com
5,john.doe@example.com
```

Nun wollen wir beide Dateien verknüpfen, um alle E-Mail-Adressen von Personen mit braunen Haaren ausgeben lassen. Das sieht dann so aus:

```
$> q -H -d ";" "SELECT personen.name, emails.email FROM personen.csv personen
            JOIN emails.csv emails ON (personen.id = emails.id)
            WHERE personen.haarfarbe = 'braun'"

- Result -
Max Mustermann    max.mustermann@example.com
Otto Normal       otto.normal@example.com
John Doe          john.doe@example.com
```

Beispiel 3

Q behandelt text as data und daher können wir auch Ausgaben aus Unix-Befehlen mit q verarbeiten. Um beispielsweise alle aktuellen Prozesskommandos aufzulisten, die am 24. Dezember gestartet wurden, können wir das so in q ausführen:

```
$> ps aux | q -H "SELECT COMMAND FROM - WHERE STARTED = '24Dez20'"

- Result -
/System/Library/CoreServices/Santa
...
```

Δ Performance und Limitierungen

Die aktuelle Version von q ist im Ausführen von SQL-Queries auf CSV-Dateien deutlich schneller als vergleichbare Go-basierende Tools wie Textql und Octosql [3]. Uns würde ein Vergleich von q mit xsv, das CSVs indizieren und verarbeiten kann, interessieren.
Uns ist jedoch aufgefallen, dass es q beispielsweise nicht ermöglicht, FROM auf einer Subquery auszuführen. Außerdem benutzt q den SQLite als SQL-Dialekt. Weitere Einschränkungen sind auf der Website von q [1] zu finden.

+ Weiterführende Aspekte

- [1] http://harelba.github.io/q/
- [2] http://harelba.github.io/q/#installation
- [3] https://github.com/harelba/q/blob/master/test/BENCHMARK.md
- [4] https://github.com/BurntSushi/xsv