SQL with Python in the Middle

```
$ spyql "IMPORT getpass SELECT getpass.getuser() AS author,
date.today() AS date, 'FOSDEM' AS conference TO json" | jq

{
  "author": "dcmoura",
  "date": "2022-02-06",
  "conference": "FOSDEM"
}
```

SQL with Python in the Middle

```
$ spyql "IMPORT getpass SELECT getpass.getuser() AS author,
date.today() AS date, 'FOSDEM' AS conference TO json" | yq eval -P

author: dcmoura
date: "2022-02-06"
conference: FOSDEM
```

SQL with Python in the Middle

```
$ spyql "IMPORT getpass SELECT getpass.getuser() AS author,
date.today() AS date, 'FOSDEM' AS conference TO csv"

author,date,conference
dcmoura,2022-02-06,FOSDEM
```

SQL with Python in the Middle

```
$ spyql "IMPORT getpass SELECT getpass.getuser() AS author,
date.today() AS date, 'FOSDEM' AS conference TO sql"

INSERT INTO "table_name"("author","date","conference")
VALUES (‘dcmoura','2022-02-06','FOSDEM');
```

SQL with Python in the Middle

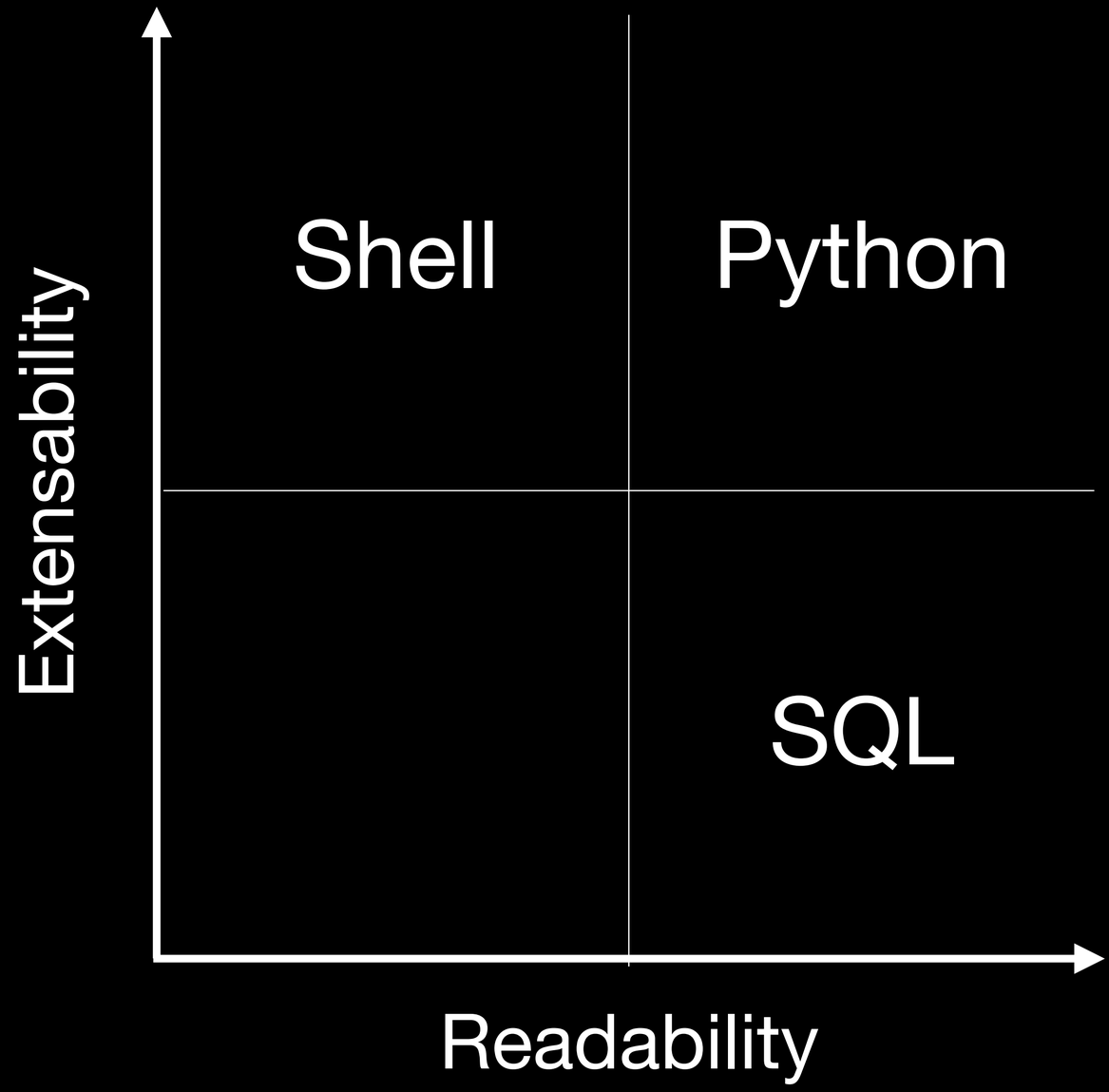
```
$ spyql "IMPORT getpass SELECT getpass.getuser() AS author,
date.today() AS date, 'FOSDEM' AS conference TO pretty"

author    date        conference
--------  ----------  ------------
dcmoura   2022-02-06  FOSDEM
```

```
$ http https://fosdem.org/2022/schedule/event/python_spyql/ |
pup 'head meta[name="DC.Title"] attr{content}' |
spyql "
    IMPORT getpass
    SELECT
        col1 AS title,
        getpass.getuser() AS author,
        date.today() AS date
    FROM text
    TO json" | jq

{
  "title": "SPyQL - SQL with Python in the middle",
  "author": "dcmoura",
  "date": "2022-02-06",
}
```

```
$ echo "https://fosdem.org/2022/schedule/event/python_spyql/" |
spyql "
    IMPORT getpass, pyquery AS pq
    SELECT
        pq.PyQuery(col1)('meta[name=\'DC.Title\']').attr['content']
          AS title,
        getpass.getuser() AS author,
        date.today() AS date
    FROM text
    TO json" | jq

{
  "title": "SPyQL - SQL with Python in the middle”,
  "author": “dcmoura",
  "date": "2022-02-06",
}
```

Making command-line data processing more:

- intuitive
- readable
- powerful



SPyQL is data-driven

Input / Output

Format	Input	Output
CSV	Auto/manual	Yes
JSON	JSON lines	Yes
Raw text	Yes	
SQL		Bulk INSERTs
Pretty		Yes
Plots		Yes
SPy	Yes	Yes

SPyQL is data-driven

NULL Type: tries not to break!

```
abs(NULL - 1) is NULL

float("I am not a number") is NULL

mydict["I_do_not_exist"] is NULL
mydict->I_do_not_exist is NULL

mydict["I don’t exist"]["and neither do I"] is NULL
```

SPyQL is data-driven

Data context on error messages

```
$ spyql "SELECT 1/(json->overall-1) FROM json" < smpl.json > /dev/null

ERROR    could not evaluate SELECT expression #1: 1/(json['overall']-1)
        at data row #98: [{'overall': 1.0, 'verified': True,
'reviewTime': '02 20, 2016', 'reviewerID': 'A2M08SO0PJKPAV', 'asin':
'0001712799', 'style': {'Format:': ' Hardcover'}, 'reviewerName':
'Emily', 'reviewText': "Completly boring!!! Yes it's a childerns book
that they will be able to read beacuse 60% of the book is the word Up.
\nThis one never gets picked for story time just sits on the shelf.\nWe
love Dr Seuss books but this one is disappointing.", 'summary': "Don't
waste your money", 'unixReviewTime': 1455926400}]
ZeroDivisionError: float division by zero
```

SPyQL is data-driven

Small memory footprint

- Does not load the full dataset into memory
  - Loads 1 line at a time
- Streaming-ready
  - Writes output ASAP

SPyQL is shell-friendly

- pipable: stdin → stdout
- queries are 1-liners
- respects the order of the input

Querying 1M JSON lines / 700MB

Book reviews from Amazon https://jmcauley.ucsd.edu/data/amazon/

```
$ head -1 smpl.json | jq -c

{"overall":5,"verified":false,"reviewTime":"08 12,
2005","reviewerID":"A1C6M8LCIX4M6M","asin":"000171
3353","style":{"Format:":"
Paperback"},"reviewerName":"June
Bug","reviewText":"This book is a winner with both
of my boys.  They really enjoy the pictures and
the story.  It's a classic.","summary":"Children's
favorite","unixReviewTime":1123804800}
```

Benchmark: averaging a JSON field

CPU time (secs) for 1M JSON lines / 700MB

```
python3 -c "import json, sys, statistics; dec = json.JSONDecoder(); print(
statistics.mean((dec.decode(line)['overall'] for line in sys.stdin)))" < smpl
```

```
python3 -c "import pandas as pd, sys; print(pd.read_json(sys.stdin,
lines=True)['overall'].mean())" < smpl.json
```

```
jq -n '[inputs.overall] | add/length' smpl.json
```

```
spyql "SELECT avg_agg(json->overall) FROM json" < smpl.json
```

5 10 15 20 25

Benchmark: averaging a JSON field

Memory peak (GB) for 1M JSON lines / 700MB

```
python3 -c "import json, sys, statistics; dec = json.JSONDecoder(); print(
statistics.mean((dec.decode(line)['overall'] for line in sys.stdin)))" < smpl
```

```
python3 -c "import pandas as pd, sys; print(pd.read_json(sys.stdin,
lines=True)['overall'].mean())" < smpl.json
```

```
jq -n '[inputs.overall] | add/length' smpl.json
```

```
spyql "SELECT avg_agg(json->overall) FROM json" < smpl.json
```

1 2 3 4

Benchmark: averaging a CSV field

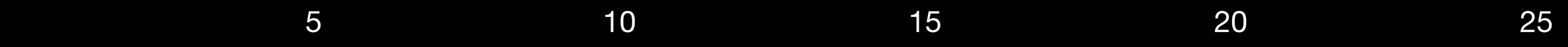
CPU time (secs) for 1M CSV lines / 420MB

```
python3 -c "import sys, statistics, csv; print(statistics.mean(
(float(line['overall']) for line in csv.DictReader(sys.stdin))))" < smpl.csv
```

```
python3 -c "import pandas as pd, sys; print(pd.read_csv(sys.stdin,
usecols=['overall'], squeeze=True, engine='c').mean())" < smpl.csv
```

```
sed 1d smpl2.csv | awk -F '§' '{ sum += $2 } END { print sum / NR }'
```

```
spyql "SELECT avg_agg(overall) FROM csv" < smpl.csv
```



Benchmark: averaging a CSV field

Memory peak (GB) for 1M CSV lines / 420MB

```
python3 -c "import sys, statistics, csv; print(statistics.mean(
(float(line['overall']) for line in csv.DictReader(sys.stdin))))" < smpl.csv
```

```
python3 -c "import pandas as pd, sys; print(pd.read_csv(sys.stdin,
usecols=['overall'], squeeze=True, engine='c').mean())" < smpl.csv
```

```
sed 1d smpl2.csv | awk -F '§' '{ sum += $2 } END { print sum / NR }'
```

```
spyql "SELECT avg_agg(overall) FROM csv" < smpl.csv
```

1 2 3 4

Top 5 reviewers

spyql - CSV

```
$ spyql "SELECT
   reviewerName AS reviewer,
   count_agg(*) AS num_reviews,
   avg_agg(overall) AS avg_score,
   avg_agg(len(ifnull(reviewText, ''))) AS avg_review_len
  FROM csv
  GROUP BY 1 ORDER BY 2 DESC LIMIT 5 TO pretty" < smpl.csv

  reviewer          num_reviews    avg_score    avg_review_len
  --------------  -------------  -----------  ----------------
  Amazon Customer         60966      4.37854           281.368
  Kindle Customer         15703      4.40247           262.154
  Chris                     672      4.26786           557.394
  Sarah                     621      4.34622           539.3
  John                      612      4.26961           379.609
```

Top 5 reviewers

spyql - JSON

```
$ spyql "SELECT
   json->reviewerName AS reviewer,
   count_agg(*) AS num_reviews,
   avg_agg(json->overall) AS avg_score,
   avg_agg(len(ifnull(json->reviewText, ''))) AS avg_review_len
  FROM json
  GROUP BY 1 ORDER BY 2 DESC LIMIT 5 TO pretty" < smpl.json
```

reviewer	num_reviews	avg_score	avg_review_len
Amazon Customer	60966	4.37854	281.368
Kindle Customer	15703	4.40247	262.154
Chris	672	4.26786	557.394
Sarah	621	4.34622	539.3
John	612	4.26961	379.609

Top 5 reviewers

jq - JSON

```
$ jq -cs '
    group_by(.reviewerName) |
    map({
        reviewer: .[0].reviewerName,
        num_reviews: length,
        avg_score: map(.overall) | (add/length),
        avg_reviews_len: map(.reviewText | length) | (add/length)}) |
    sort_by(.num_reviews) | reverse |
    .[0:5][]' smpl.json
```

```
{"reviewer":"Amazon Customer","num_reviews":60966,"avg_score":4.37854,"avg_len":281.368}
{"reviewer":"Kindle Customer","num_reviews":15703,"avg_score":4.40247,"avg_len":262.153}
{“reviewer":"Chris","num_reviews":672,"avg_score":4.26786,"avg_len":557.394}
{“reviewer":"Sarah","num_reviews":621,"avg_score":4.34622,"avg_len":539.299}
{"reviewer":"Linda","num_reviews":612,"avg_score":4.41667,"avg_len":233.284}
```

Top 5 reviewers
awk - csv

```
$ sed 1d smpl2.csv | awk 'BEGIN {FS = "§"; OFS = "|"}
{
  num_reviews[$8]++
  sum_score[$8] += $2
  sum_len[$8] += length($1)
}
END {
  for (r in sum_len)
    print r, num_reviews[r], sum_score[r]/num_reviews[r], sum_len[r]/num_reviews[r]
}' |
sort -t "|" -rnk2 | head -5
```

```
Amazon Customer|60966|4,37854|282,305
Kindle Customer|15703|4,40247|263,008
Chris|672|4,26786|560,057
Sarah|621|4,34622|541,519
Linda|612|4,41667|233,943
```

Scaling down K8s pods

Helping out automating tasks

```
$ kubectl get pods -o json |
jq -c '.items[].metadata.labels' |
spyql "
  IMPORT os
  SELECT
    json->app,
    os.system(
      'kubectl scale deployment {} -replicas=0'.format(json->app)
    ) AS exec_status
  FROM json
  WHERE json->app is not NULL
    and json->app.startswith('data-pipe')"
```

Partial aggregations

aka in SQL as analytical/window functions

```
$ spyql "SELECT sum_agg(col1) FROM [100,20,3] TO json"
{"sum_agg_col1": 123}


$ spyql "SELECT PARTIALS sum_agg(col1) FROM [100,20,3] TO json"
{"sum_agg_col1": 100}
{"sum_agg_col1": 120}
{"sum_agg_col1": 123}
```

Kafka streaming stats

Leveraging partial aggregates

```
$ kafkacat -q -b mykafka.com -t my.topic.with.data |
spyql "IMPORT time SELECT time.time() AS ts FROM json TO spy" |
spyql "
  SELECT PARTIALS
    round((ts - first_agg(ts))*1000) AS elapsed_ms,
    round(1.0/(ts-lag_agg(ts))) AS msgs_per_sec,
    round(3.0/(ts-lag_agg(ts,3))) AS msgs_per_sec_avg3msgs,
    count_agg(1) AS n_msgs
  FROM spy TO json”

{"elapsed_ms": 0, "msgs_per_sec": null, "msgs_per_sec_avg3msgs": null, "n_msgs": 1}
{"elapsed_ms": 0, "msgs_per_sec": 6754, "msgs_per_sec_avg3msgs": null, "n_msgs": 2}
{"elapsed_ms": 0, "msgs_per_sec": 10755, "msgs_per_sec_avg3msgs": null, "n_msgs": 3}
{"elapsed_ms": 0, "msgs_per_sec": 8542, "msgs_per_sec_avg3msgs": 10438, "n_msgs": 4}
```

github.com/dcmoura/spyql

daniel.c.moura@gmail.com

```
$ spyql -Oheader=False "
  SELECT 'Thank you' + ' :-D' * col1
  FROM [2**x for x in range(4)] TO pretty"

-----------------------------------
Thank you :-D
Thank you :-D :-D
Thank you :-D :-D :-D :-D
Thank you :-D :-D :-D :-D :-D :-D :-D
-----------------------------------
```