

DB or no DB?

Advanced Databases

Pierre Senellart



30 September 2022

Is a DBMS necessary?

- DBMSs are good at managing very large amounts of data

Is a DBMS necessary?

- DBMSs are good at managing very large amounts of data... but sometimes data we deal with are not that large!

Is a DBMS necessary?

- DBMSs are good at managing very **large amounts of data**... but sometimes data we deal with are **not that large!**
- DBMSs provide a standard **declarative** query language with smart query **optimization**

Is a DBMS necessary?

- DBMSs are good at managing very **large amounts of data**... but sometimes data we deal with are **not that large!**
- DBMSs provide a standard **declarative** query language with smart query **optimization**... but sometimes it is fine to describe **precisely how** the computation is to be performed!

Is a DBMS necessary?

- DBMSs are good at managing very **large amounts of data**... but sometimes data we deal with are **not that large!**
- DBMSs provide a standard **declarative** query language with smart query **optimization**... but sometimes it is fine to describe **precisely how** the computation is to be performed!
- DBMSs provide different views of data, **isolation between users**, concurrency control

Is a DBMS necessary?

- DBMSs are good at managing very **large amounts of data**... but sometimes data we deal with are **not that large!**
- DBMSs provide a standard **declarative** query language with smart query **optimization**... but sometimes it is fine to describe **precisely how** the computation is to be performed!
- DBMSs provide different views of data, **isolation between users**, concurrency control... but sometimes there is **only a single user**

Is a DBMS necessary?

- DBMSs are good at managing very **large amounts of data**... but sometimes data we deal with are **not that large!**
- DBMSs provide a standard **declarative** query language with smart query **optimization**... but sometimes it is fine to describe **precisely how** the computation is to be performed!
- DBMSs provide different views of data, **isolation between users**, concurrency control... but sometimes there is **only a single user**
- DBMSs provide updating features, ensure constraints are not violated while updating, ensure updated data is in a consistent state

Is a DBMS necessary?

- DBMSs are good at managing very **large amounts of data**... but sometimes data we deal with are **not that large!**
- DBMSs provide a standard **declarative** query language with smart query **optimization**... but sometimes it is fine to describe **precisely how** the computation is to be performed!
- DBMSs provide different views of data, **isolation between users**, concurrency control... but sometimes there is **only a single user**
- DBMSs provide updating features, ensure constraints are not violated while updating, ensure updated data is in a consistent state... but sometimes a dataset **is not to be modified**

Possible alternatives

- In memory** Ad-hoc management of data stored in main memory, within some programming language – if the data fits within memory. Will be illustrated by the **pandas** Python library.
- On disk** Ad-hoc management of data on disk, stored in files, either through programming or through the use of external tools. Will be illustrated by the **Unix command line** tools.

When a DBMS is necessary

- When the data needs to be used by **other applications**
- When data **updating**, transactions, concurrency control, user isolation, etc., is important
- When queries become **complex**, and are more manageable and easily optimized in a declarative query language like SQL than in an ad-hoc language
- When data volumes are **too large** for simple in-memory storage or for ad-hoc disk accesses, when indexes are required

Data management concepts

Even when not using a DBMS, data management concepts are important:

- expressing operations in terms of **formal operators** such as selections, projections, joins, etc., allows to **better understand and describe** what needs to be performed
- paying attention to integrity constraints allows catching up potential **errors in data formats**
- the notion of **physical and logical independence** may still be relevant in how to design a computation

Assumption

- Data still follows the relational data model (similar processes may be followed for other data models)
- Data will be stored in simple text files with newline-separated rows, and delimiter-separated attribute values on each row
- Data available in extension as files, one per table

Plan

Introduction

pandas

Unix command line tools

pandas

- Rich library for expressing **complex manipulation of tabular data** in Python
- Data tables available in Python in the form of a **DataFrame object**
- Heavily inspired by the way data tables are handled in statistical computing language such as **R** and **SAS**
- **Not** a **declarative** language: the way an expression is written is the way it will be executed (with minor optimizations)
- Results in code **less verbose** than SQL, but also somewhat **more cryptic**
- All data is stored **in main memory**: does not scale to large datasets
- As this is Python code, **arbitrary** code can be written, interfaces with other libraries (e.g., for deep learning)

The DataFrame object

- Representation of a **relation** (tabular data, fixed number of columns/attributes, names for attributes, etc.)
- Each **row** can be assigned an **index**, i.e., a name; similar concept to that of primary key – if no name assigned, rows are referred to by a sequential numbering
- Relies on Series objects, which are unidimensional arrays of data; each column is such a series
- The DataFrame is said to have two axes: axis 0 is the rows, axis 1 the columns
- As there are row indexes, there are **column indexes**, i.e., attribute names

Constructing a DataFrame

```
import pandas as pd
```

```
# Literal DataFrame
```

```
guest = pd.DataFrame(  
    data={  
        'name': ['John Smith', 'Alice Black', 'John Smith'],  
        'email': ['john.smith@gmail.com',  
                 'alice@black.name',  
                 'john.smith@ens.fr']  
    },  
    index = pd.Index([1, 2, 3], name='id'))
```

```
# Read DataFrame from CSV file, first column as row index
```

```
reservation = pd.read_csv('reservation.csv', index_col=0)
```

Renaming (1/2)

| Guest | | |
|-------|-------------|----------------------|
| id | name | email |
| 1 | John Smith | john.smith@gmail.com |
| 2 | Alice Black | alice@black.name |
| 3 | John Smith | john.smith@ens.fr |

| Reservation | | | | |
|-------------|-------|------|------------|--------|
| id | guest | room | arrival | nights |
| 1 | 1 | 504 | 2017-01-01 | 5 |
| 2 | 2 | 107 | 2017-01-10 | 3 |
| 3 | 3 | 302 | 2017-01-15 | 6 |
| 4 | 2 | 504 | 2017-01-15 | 2 |
| 5 | 2 | 107 | 2017-01-30 | 1 |

$$\rho_{id \rightarrow \text{guest}}(\text{Guest})$$

```
guest.index.name='guest'
```

(This changes the guest DataFrame)

Renaming (2/2)

| Guest | | | Reservation | | | | |
|-------|-------------|----------------------|-------------|-------|------|------------|--------|
| id | name | email | id | guest | room | arrival | nights |
| 1 | John Smith | john.smith@gmail.com | 1 | 1 | 504 | 2017-01-01 | 5 |
| 2 | Alice Black | alice@black.name | 2 | 2 | 107 | 2017-01-10 | 3 |
| 3 | John Smith | john.smith@ens.fr | 3 | 3 | 302 | 2017-01-15 | 6 |
| | | | 4 | 2 | 504 | 2017-01-15 | 2 |
| | | | 5 | 2 | 107 | 2017-01-30 | 1 |

$\rho_{\text{email} \rightarrow \text{e-mail}}(\text{Guest})$

```
guest.rename(columns={'email': 'e-mail'})
```

Projection

| Guest | | |
|-------|-------------|----------------------|
| id | name | email |
| 1 | John Smith | john.smith@gmail.com |
| 2 | Alice Black | alice@black.name |
| 3 | John Smith | john.smith@ens.fr |

| Reservation | | | | |
|-------------|-------|------|------------|--------|
| id | guest | room | arrival | nights |
| 1 | 1 | 504 | 2017-01-01 | 5 |
| 2 | 2 | 107 | 2017-01-10 | 3 |
| 3 | 3 | 302 | 2017-01-15 | 6 |
| 4 | 2 | 504 | 2017-01-15 | 2 |
| 5 | 2 | 107 | 2017-01-30 | 1 |

$$\Pi_{\text{email}, \text{id}}(\text{Guest})$$

```
guest[['email']]
```

(The row index always comes first)

Selection

| Guest | | | Reservation | | | | |
|-------|-------------|----------------------|-------------|-------|------|------------|--------|
| id | name | email | id | guest | room | arrival | nights |
| 1 | John Smith | john.smith@gmail.com | 1 | 1 | 504 | 2017-01-01 | 5 |
| 2 | Alice Black | alice@black.name | 2 | 2 | 107 | 2017-01-10 | 3 |
| 3 | John Smith | john.smith@ens.fr | 3 | 3 | 302 | 2017-01-15 | 6 |
| | | | 4 | 2 | 504 | 2017-01-15 | 2 |
| | | | 5 | 2 | 107 | 2017-01-30 | 1 |

$\sigma_{\text{arrival} > 2017-01-12 \wedge \text{guest} = 2}(\text{Reservation})$

```
reservation[(reservation.arrival > '2017-01-12') & \
             (reservation['guest'] == 2)]
```

Cross product

| Guest | | | Reservation | | | | |
|-------|-------------|----------------------|-------------|-------|------|------------|--------|
| id | name | email | id | guest | room | arrival | nights |
| 1 | John Smith | john.smith@gmail.com | 1 | 1 | 504 | 2017-01-01 | 5 |
| 2 | Alice Black | alice@black.name | 2 | 2 | 107 | 2017-01-10 | 3 |
| 3 | John Smith | john.smith@ens.fr | 3 | 3 | 302 | 2017-01-15 | 6 |
| | | | 4 | 2 | 504 | 2017-01-15 | 2 |
| | | | 5 | 2 | 107 | 2017-01-30 | 1 |

$$\Pi_{id}(\text{Guest}) \times \Pi_{name}(\text{Guest})$$

```
guest[[]].reset_index().merge(\n    guest[['name']],how='cross').\n    drop_duplicates().sort_values(['name','id'])
```

Union

| Guest | | | Reservation | | | | |
|-------|-------------|----------------------|-------------|-------|------|------------|--------|
| id | name | email | id | guest | room | arrival | nights |
| 1 | John Smith | john.smith@gmail.com | 1 | 1 | 504 | 2017-01-01 | 5 |
| 2 | Alice Black | alice@black.name | 2 | 2 | 107 | 2017-01-10 | 3 |
| 3 | John Smith | john.smith@ens.fr | 3 | 3 | 302 | 2017-01-15 | 6 |
| | | | 4 | 2 | 504 | 2017-01-15 | 2 |
| | | | 5 | 2 | 107 | 2017-01-30 | 1 |

$$\begin{aligned} & \Pi_{\text{room}}(\sigma_{\text{guest}=2}(\text{Reservation})) \\ \cup & \Pi_{\text{room}}(\sigma_{\text{arrival}=2017-01-15}(\text{Reservation})) \end{aligned}$$

```
pd.concat([
    reservation[reservation.guest==2].\
        reset_index()[['room']],
    reservation[reservation.arrival=='2017-01-15'].\
        reset_index()[['room']]
]).drop_duplicates()
```

Difference

| Guest | | | Reservation | | | | |
|-------|-------------|----------------------|-------------|-------|------|------------|--------|
| id | name | email | id | guest | room | arrival | nights |
| 1 | John Smith | john.smith@gmail.com | 1 | 1 | 504 | 2017-01-01 | 5 |
| 2 | Alice Black | alice@black.name | 2 | 2 | 107 | 2017-01-10 | 3 |
| 3 | John Smith | john.smith@ens.fr | 3 | 3 | 302 | 2017-01-15 | 6 |
| | | | 4 | 2 | 504 | 2017-01-15 | 2 |
| | | | 5 | 2 | 107 | 2017-01-30 | 1 |

$$\Pi_{\text{room}}(\sigma_{\text{guest}=2}(\text{Reservation}))$$
$$\setminus \Pi_{\text{room}}(\sigma_{\text{arrival}=2017-01-15}(\text{Reservation}))$$

```
r1=reservation[reservation.guest==2].\  
    reset_index()[['room']]  
r2=reservation[reservation.arrival=='2017-01-15'].\  
    reset_index()[['room']]  
r1.merge(r2,how='outer',indicator=True).\  
    query('_merge=="left_only"')[['room']].\  
    drop_duplicates()
```

Join

| Guest | | | Reservation | | | | |
|-------|-------------|----------------------|-------------|-------|------|------------|--------|
| id | name | email | id | guest | room | arrival | nights |
| 1 | John Smith | john.smith@gmail.com | 1 | 1 | 504 | 2017-01-01 | 5 |
| 2 | Alice Black | alice@black.name | 2 | 2 | 107 | 2017-01-10 | 3 |
| 3 | John Smith | john.smith@ens.fr | 3 | 3 | 302 | 2017-01-15 | 6 |
| | | | 4 | 2 | 504 | 2017-01-15 | 2 |
| | | | 5 | 2 | 107 | 2017-01-30 | 1 |

Reservation $\bowtie_{\text{guest=id}}$ Guest

```
pd.concat([\n    reservation.reset_index().\n        set_index('guest', drop=False),\n    guest],\n    axis=1)
```

Aggregation

| Guest | | | Reservation | | | | |
|-------|-------------|----------------------|-------------|-------|------|------------|--------|
| id | name | email | id | guest | room | arrival | nights |
| 1 | John Smith | john.smith@gmail.com | 1 | 1 | 504 | 2017-01-01 | 5 |
| 2 | Alice Black | alice@black.name | 2 | 2 | 107 | 2017-01-10 | 3 |
| 3 | John Smith | john.smith@ens.fr | 3 | 3 | 302 | 2017-01-15 | 6 |
| | | | 4 | 2 | 504 | 2017-01-15 | 2 |
| | | | 5 | 2 | 107 | 2017-01-30 | 1 |

$$\sigma_{\text{avg} > 3}(\gamma_{\text{room}}^{\text{avg}}[\lambda x. \text{avg}(x)](\Pi_{\text{room}, \text{nights}}(\text{Reservation})))$$

```
reservation.groupby('room')[['nights']].\
mean().\
query('nights>3').\
sort_values(by='room')
```

But also

`df.sort_values` to order results (similar to **ORDER BY** in SQL)

`df.head` to limit the number of answers (similar to **LIMIT** in SQL)

Plan

Introduction

pandas

Unix command line tools

Command line tools

- **Classical tools** used within a Unix/Linux/WSL shell to manipulate text files
- **Disk-based** and **pipelined** file manipulation: usually no important use of memory
- **Scales** to very large files, but no indexing capabilities, only linear (and pipelined) processing of files

Pipelines

- If a and b are two commands, then $a | b$:
 - simultaneously launches commands a and b
 - **redirects the standard output** of a to the **standard input** of b
- This means b is provided (on its standard input) the standard output of a , **as it is produced**
- **Blocking** (with buffering): if b stops reading its input, a stops producing an output (and conversely)
- Central point of Unix/Linux command line philosophy, very convenient
- Can be chained: $a | b | c | d | \dots$

Reading from standard input or arguments

- Most common Unix commands can read their input:
 - either through their **standard input** (useful for pipelines)
 - or within a **file** given as **argument**
- **Process substitution**: $a <(b)$ launches command b and provides to command a a filename (actually a **temporary named pipe**) which, if read, provides the output of the b command
- Often, when a filename is expected as argument, $-$ means to read from standard input instead

Getting help on a command

- Try *command* --help or *command* -h
- **man** is a documentation integrated within Unix/Linux:
“man *command*” displays a manual page about a command
- **man -k** searches a man entry by keyword
- **whatis** short summary of a command

Format of input files

- Rows separated by newlines
- Attributes separated by a special delimiter character, “,” in examples; commands use a flag (-d, -t, -F) to indicate the delimiter
- No header line; can be removed using `tail -n +2 file.csv`
- More complex CSV files: see the `csvkit` or `csvquote` tools

awk

- Programming language dedicated to the processing of tabular data within text files
- General form of an awk program:

```
BEGIN { instructions1 }  
condition { instructions2 }  
END { instructions3 }
```
- instructions1 is executed at the beginning of a file, the **BEGIN** block can be omitted
- condition is a condition for each line to be processed; if omitted, defaults to matching every line
- instructions2 is executed at each line matched by conditions; if omitted defaults to print the line
- instructions3 is executed at the end of a file, the **END** block can be omitted

Projection

| Guest | | |
|-------|-------------|----------------------|
| id | name | email |
| 1 | John Smith | john.smith@gmail.com |
| 2 | Alice Black | alice@black.name |
| 3 | John Smith | john.smith@ens.fr |

| Reservation | | | | |
|-------------|-------|------|------------|--------|
| id | guest | room | arrival | nights |
| 1 | 1 | 504 | 2017-01-01 | 5 |
| 2 | 2 | 107 | 2017-01-10 | 3 |
| 3 | 3 | 302 | 2017-01-15 | 6 |
| 4 | 2 | 504 | 2017-01-15 | 2 |
| 5 | 2 | 107 | 2017-01-30 | 1 |

$$\Pi_{\text{email}, \text{id}}(\text{Guest})$$

```
awk -F, '{print $3 "," $1}' guest.csv
```

ou

```
cut -d, -f3,1 guest.csv
```

(cut ne permet pas de réordonner ou de répéter des colonnes)

Selection

| Guest | | | Reservation | | | | |
|-------|-------------|----------------------|-------------|-------|------|------------|--------|
| id | name | email | id | guest | room | arrival | nights |
| 1 | John Smith | john.smith@gmail.com | 1 | 1 | 504 | 2017-01-01 | 5 |
| 2 | Alice Black | alice@black.name | 2 | 2 | 107 | 2017-01-10 | 3 |
| 3 | John Smith | john.smith@ens.fr | 3 | 3 | 302 | 2017-01-15 | 6 |
| | | | 4 | 2 | 504 | 2017-01-15 | 2 |
| | | | 5 | 2 | 107 | 2017-01-30 | 1 |

$\sigma_{\text{arrival} > 2017-01-12 \wedge \text{guest} = 2}(\text{Reservation})$

```
awk -F, '$4 > "2017-01-12" && $2 == 2' reservation.csv
```

Cross product

| Guest | | | Reservation | | | | |
|-------|-------------|----------------------|-------------|-------|------|------------|--------|
| id | name | email | id | guest | room | arrival | nights |
| 1 | John Smith | john.smith@gmail.com | 1 | 1 | 504 | 2017-01-01 | 5 |
| 2 | Alice Black | alice@black.name | 2 | 2 | 107 | 2017-01-10 | 3 |
| 3 | John Smith | john.smith@ens.fr | 3 | 3 | 302 | 2017-01-15 | 6 |
| | | | 4 | 2 | 504 | 2017-01-15 | 2 |
| | | | 5 | 2 | 107 | 2017-01-30 | 1 |

$$\Pi_{\text{id}}(\text{Guest}) \times \Pi_{\text{name}}(\text{Guest})$$

```
join -t, -j 2 -o '1.1,2.1' \  
  <(cut -d, -f1 guest.csv) \  
  <(cut -d, -f2 guest.csv) | \  
  sort -t, -k2,1 | uniq
```

Union

| Guest | | | Reservation | | | | |
|-------|-------------|----------------------|-------------|-------|------|------------|--------|
| id | name | email | id | guest | room | arrival | nights |
| 1 | John Smith | john.smith@gmail.com | 1 | 1 | 504 | 2017-01-01 | 5 |
| 2 | Alice Black | alice@black.name | 2 | 2 | 107 | 2017-01-10 | 3 |
| 3 | John Smith | john.smith@ens.fr | 3 | 3 | 302 | 2017-01-15 | 6 |
| | | | 4 | 2 | 504 | 2017-01-15 | 2 |
| | | | 5 | 2 | 107 | 2017-01-30 | 1 |

$$\Pi_{\text{room}}(\sigma_{\text{guest}=2}(\text{Reservation})) \\ \cup \Pi_{\text{room}}(\sigma_{\text{arrival}=2017-01-15}(\text{Reservation}))$$

```
cat <(awk -F, '$2==2 {print $3}' reservation.csv) \  
<(awk -F, '$4=="2017-01-15" {print $3}' reservation.csv) | \  
sort -u
```

Difference

| Guest | | | Reservation | | | | |
|-------|-------------|----------------------|-------------|-------|------|------------|--------|
| id | name | email | id | guest | room | arrival | nights |
| 1 | John Smith | john.smith@gmail.com | 1 | 1 | 504 | 2017-01-01 | 5 |
| 2 | Alice Black | alice@black.name | 2 | 2 | 107 | 2017-01-10 | 3 |
| 3 | John Smith | john.smith@ens.fr | 3 | 3 | 302 | 2017-01-15 | 6 |
| | | | 4 | 2 | 504 | 2017-01-15 | 2 |
| | | | 5 | 2 | 107 | 2017-01-30 | 1 |

$$\Pi_{\text{room}}(\sigma_{\text{guest}=2}(\text{Reservation}))$$
$$\setminus \Pi_{\text{room}}(\sigma_{\text{arrival}=2017-01-15}(\text{Reservation}))$$

```
join -v2 \  
<(awk -F, '$2==2 {print $3}' reservation.csv|sort) \  
<(awk -F, '$4=="2017-01-15" {print $3}' reservation.csv|sort)
```

Join

| Guest | | | Reservation | | | | |
|-------|-------------|----------------------|-------------|-------|------|------------|--------|
| id | name | email | id | guest | room | arrival | nights |
| 1 | John Smith | john.smith@gmail.com | 1 | 1 | 504 | 2017-01-01 | 5 |
| 2 | Alice Black | alice@black.name | 2 | 2 | 107 | 2017-01-10 | 3 |
| 3 | John Smith | john.smith@ens.fr | 3 | 3 | 302 | 2017-01-15 | 6 |
| | | | 4 | 2 | 504 | 2017-01-15 | 2 |
| | | | 5 | 2 | 107 | 2017-01-30 | 1 |

Reservation $\bowtie_{\text{guest=id}}$ Guest

```
sort -t, -k2 reservation.csv | \  
  join -t, -1 2 -2 1 - <(sort guest.csv)
```

Aggregation

| Guest | | | Reservation | | | | |
|-------|-------------|----------------------|-------------|-------|------|------------|--------|
| id | name | email | id | guest | room | arrival | nights |
| 1 | John Smith | john.smith@gmail.com | 1 | 1 | 504 | 2017-01-01 | 5 |
| 2 | Alice Black | alice@black.name | 2 | 2 | 107 | 2017-01-10 | 3 |
| 3 | John Smith | john.smith@ens.fr | 3 | 3 | 302 | 2017-01-15 | 6 |
| | | | 4 | 2 | 504 | 2017-01-15 | 2 |
| | | | 5 | 2 | 107 | 2017-01-30 | 1 |

$$\sigma_{\text{avg} > 3}(\gamma_{\text{room}}^{\text{avg}}[\lambda x. \text{avg}(x)](\Pi_{\text{room}, \text{nights}}(\text{Reservation})))$$

```
awk -F, '{s[$3]+=$5;++c[$3]}
END {for (r in s) print r "," s[r]/c[r]}' \
reservation.csv | \
awk -F, '$2>3' | sort -t, -n
```

But also

- `sort` to order results (similar to **ORDER BY** in SQL)
- `head` to limit the number of answers (similar to **LIMIT** in SQL)