Other Query Languages

Properties

- So far
  - Relational Algebra on relations
    - OutputRelation = Operator (InputRelation)
    - OutputRelation = InputRelation1 operator InputRelation2
    - Can be concatenated:
      - Output relation is input of next operator

$$\Pi_{sid}(\sigma_{ctype='local' \vee ctype='regional'}(C) \bowtie P)$$

$$\Pi_{sid}(\sigma_{ctype='local'}(C) \bowtie P) \cup \Pi_{sid}(\sigma_{ctype='regional'}(C) \bowtie P)$$

    - Indicates "somewhat" an execution plan – stepwise sequence of operations to come to end-result
  - SQL on relations
    - Declarative; little indication HOW to retrieve the data
  - Cypher
    - SQL operators (conditions, projections, aggregations)
    - Path pattern (somewhat a join)
    - Declarative; again little indication HOW to get the data

PigLatin

- Was popular when large-scale data processing started
  - Map/reduce distributed data processing framework
  - We will talk about this later in this course

Example (Alan Gates, Yahoo)

- users(name, age), pagelog(url, uname)
- Find the top 5 most popular pages for users aged 18-25
- SQL:

```
SELECT  url, count(*) as clicks
FROM users U, pagelog P
WHERE U.name = P.uname
AND U.age >= 18
AND U.age <= 25
GROUP BY URL
ORDER BY clicks desc
LIMIT 5                                   -- FETCH FIRST 5 ROWS ONLY
```

- *Partial solution in Relational Algebra:*

$$\pi_{url,count(*)}(GROUPBY_{url}((\sigma_{age\geq 18 \wedge age\leq 25}(users) \bowtie pagelog))$$

In Pig Latin

```
Users = load 'users' as (name,age);
Fltrd = filter Users by age >= 18 and age <= 25;
Pages = load 'pages' as (uname, url);
Jnd = join Fltrd by name, Pages by uname;
Grpd = group Jnd by url;
Smmd = foreach Grpd generate ($0), COUNT($1) as clicks;
Srtd = order Smmd by clicks desc;
Top5 = limit Srtd 5;
store Top5 into 'top5sites'
```

PigLatin

- Step by step listing of operations
  - Similar to linear algebra
- Intermediate results are explicitly stated

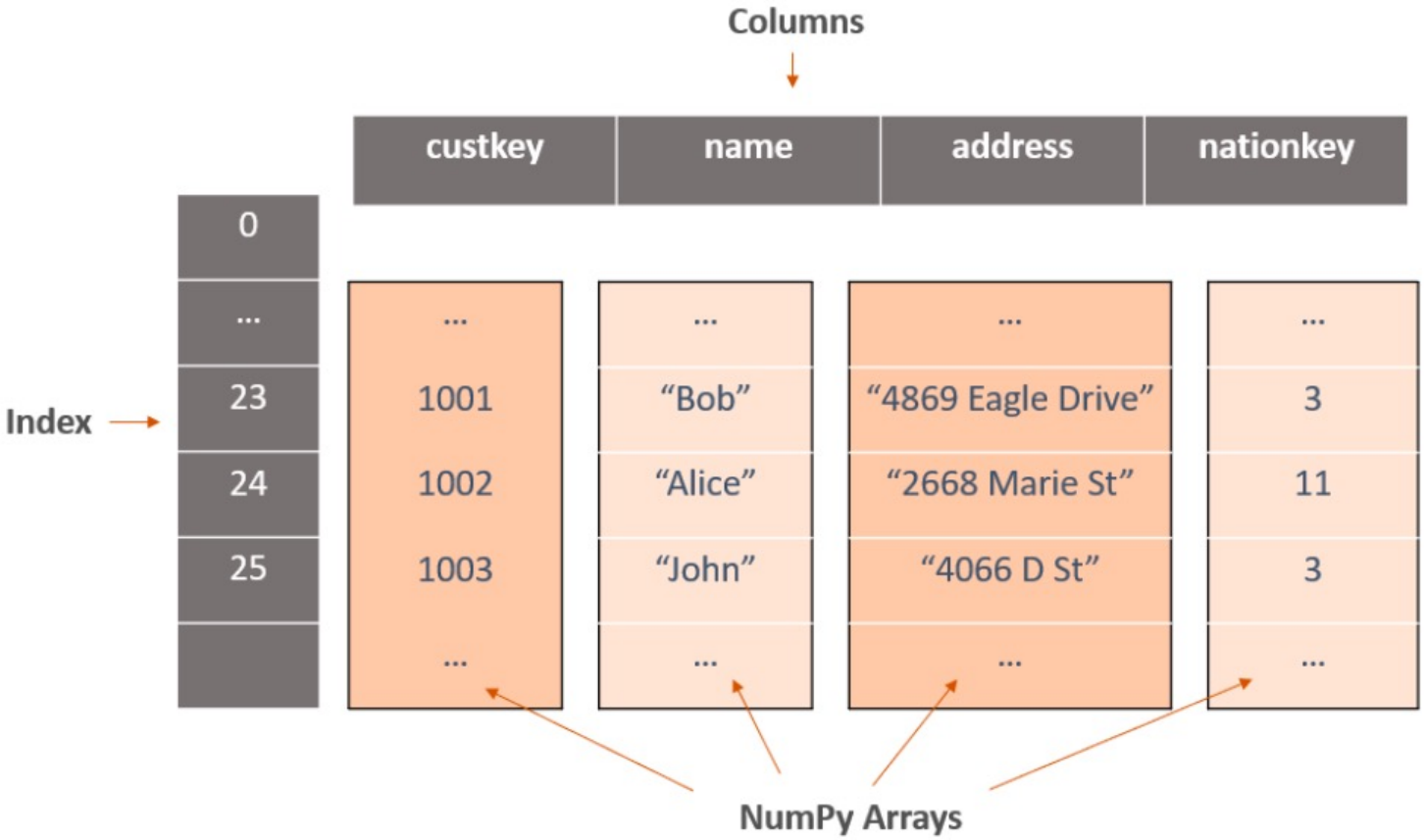
Pandas

- A Python library for data science

  Import pandas as pd

- Has the concept of a dataframe
  - Similar to a table
- You can load data from a csv file or a table from a relational DB into a dataframe object
- Then you perform transformations and manipulations

Dataframe


Columns
custkey
name
address
nationkey
0
...
23
24
25
Index
1001
"Bob"
"4869 Eagle Drive"
3
1002
"Alice"
"2668 Marie St"
11
1003
"John"
"4066 D St"
3
NumPy Arrays

Where to get the data

From local file

Skaters = pd.read_csv('skaters.csv', …)

From a database

Participates = pd.read_sql_table('participates', conn)

Simple operations

Projection

df1 = skaters [[‘sid’, ‘age’]]

Selection

df2 = df1 [df1[‘age’] <18]

Inner part returns a table with True or False rows

Outer part returns all rows that are True

Further Relational Operations

Inner Join

df3 = participates.merge(df2, left_on='sid',right_on='sid')

Outer Join

df4 = participates.merge(df2, left_on='sid', right_on='sid',
how="right")

GROUP BY

df5 = df3.groupbyby=['cid']).agg('count')

Mix relational and linear algebra

```
df6 = product[['price','discount']].to_numpy()
df7 = df6.transpose()
```

Step-by-step vs nested

- Complex queries can be listed step-by-step similar to PigLatin
  - Intermediate dataframes
- Nested notation similar to relational algebra possible

```
participates.merge(skaters[skaters['age'] < 20],
                   left_on='sid', right_on='sid')
```

Pandas Programs

- Dataframe operations can be mixed with any other Python code
- **Object-oriented API**
  - Method calls on dataframe objects
  - Return in most cases again a dataframe object

Older Stuff

- Relational Operators in Object-oriented programming languages
  - Similar in concept to what happens in Pandas
  - Used in programs after retrieving data from a database
  - Object-relational mapping
    - Django ORM
- XML documents (trees vs. graphs)
  - XQuery