QUERY

Prof. Faber Henrique

Query

$\pi_{e.name}$ ($\sigma_{e.salary > m.salary}$ (ρ_e(employee) $\bowtie_{e.manager = m.name}$ ρ_m(employee)))

(a) Relational Algebra version

RANGE employee e;
RANGE employee m;
GET w (e.name): $\exists$m((e.manager = m.name) $\wedge$ (e.salary > m.salary))

(b) Relational Calculus version


```
select e.name
from employee e, employee m
where e.manager = m.name and e.salary > m.salary
```

(c) Sequel (SQL) version

Figure 2. Three versions of the query, ''Find names of employees who earn more than their managers.''

Query

SQL É UBIQUO

Query

- ORMs
- Schemas
- Controle de Concorrência
- Manipulação de dados

Query

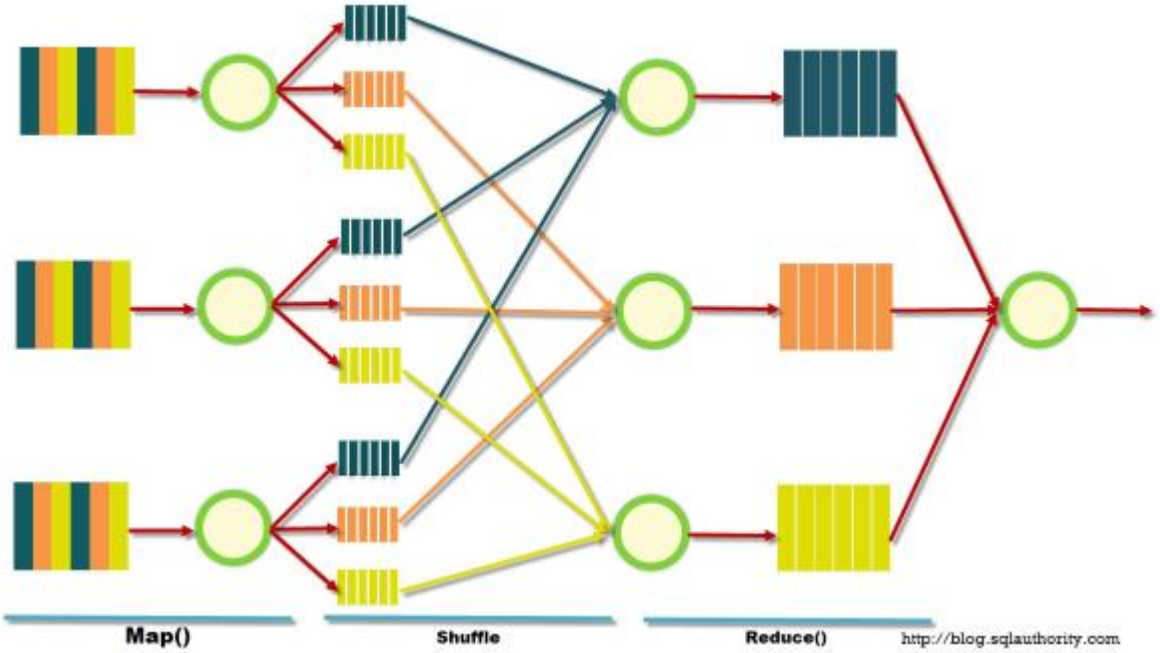
- 5 abordagens
  - NOMP
  - MapReduce
  - Abordagens específica do SGBD
  - Padronização
  - Derivações do SQL

Query

- Not is My Problem
  - Key-Value
  - Schema free
  - Rest API

Query

- MapReduce



Query

- SGBD

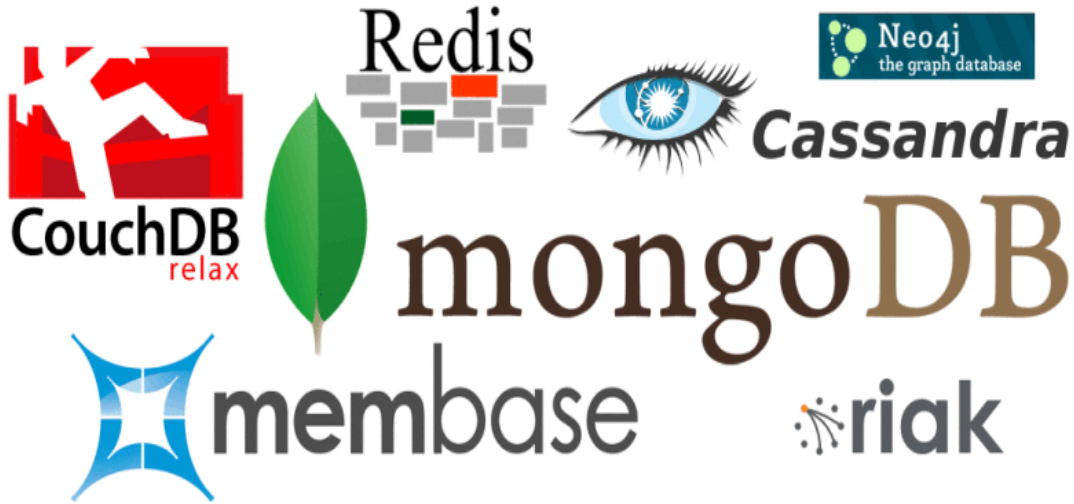
```
db.collection.command(data)
```

```
db.restaurants.find( { "cuisine": "Fish and chips" } )
```

mongoDB

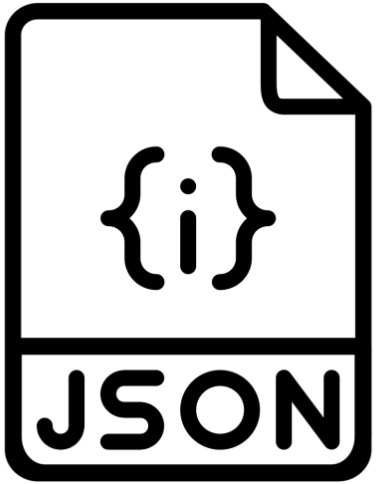
Query

- SGBD


Redis
Neo4j
the graph database
Cassandra
CouchDB
relax
mongoDB
membase
riak

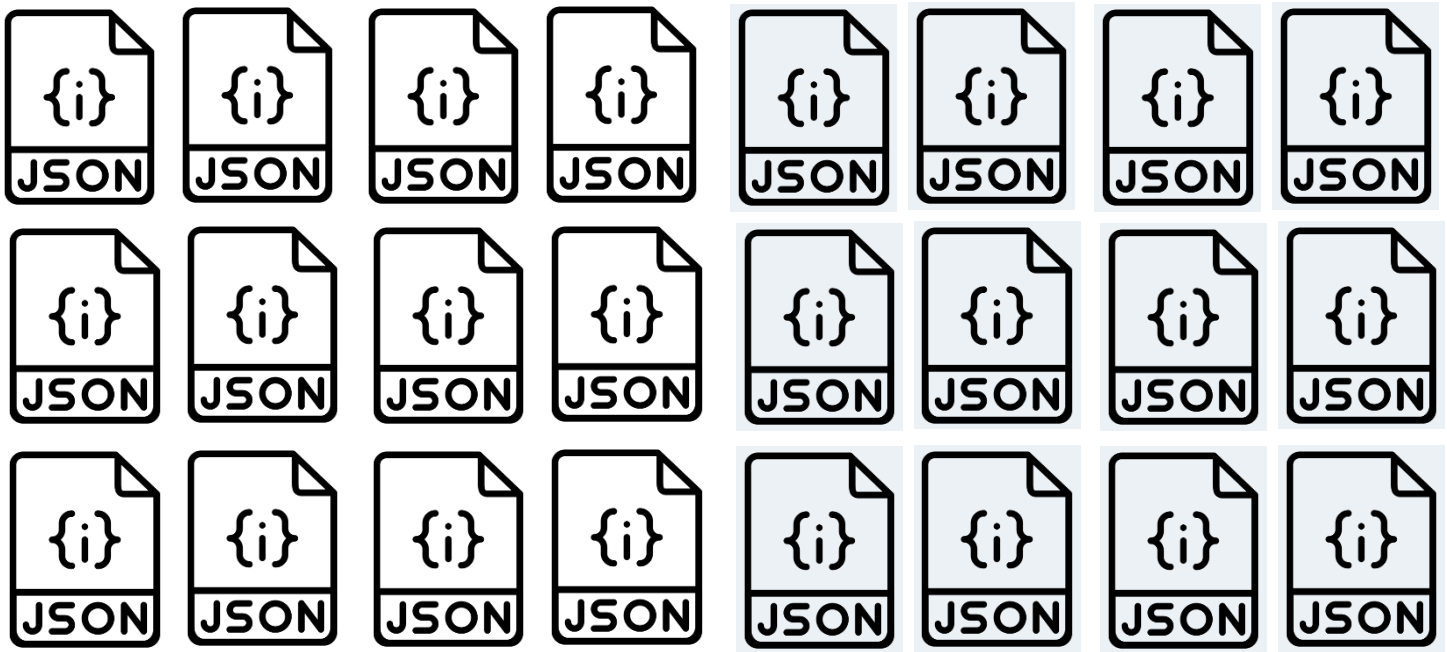
Query

- Padronização



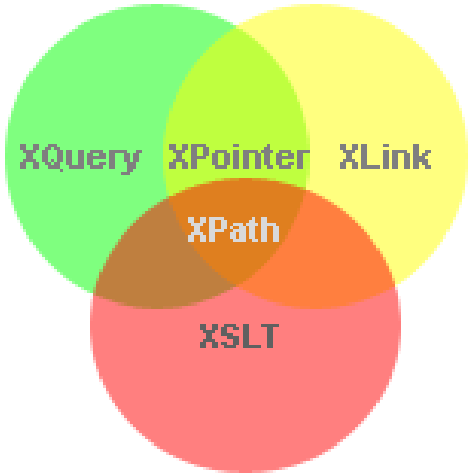
Query

- Padronização



Query

- Padronização



Query

- Padronização

```
xquery version "1.0-ml";
declare namespace rest =
"http://example.com/namespace/restaurants";
{
    for $restaurant in
doc("restaurants.xml")/rest:restaurants/rest:restaurant
    return
    {
        $restaurant/rest:name/text()
    }
}
```

Query

- Derivações do SQL

$$\pi_{e.name}\left(\sigma_{e.salary > m.salary}\left(\rho_{e}(employee) \bowtie_{e.manager = m.name} \rho_{m}(employee)\right)\right)$$

  - CQL

```
SELECT name FROM restaurants WHERE cuisine='Fish and chips';
```

  - https://www.couchbase.com/products/n1ql

Query

Query

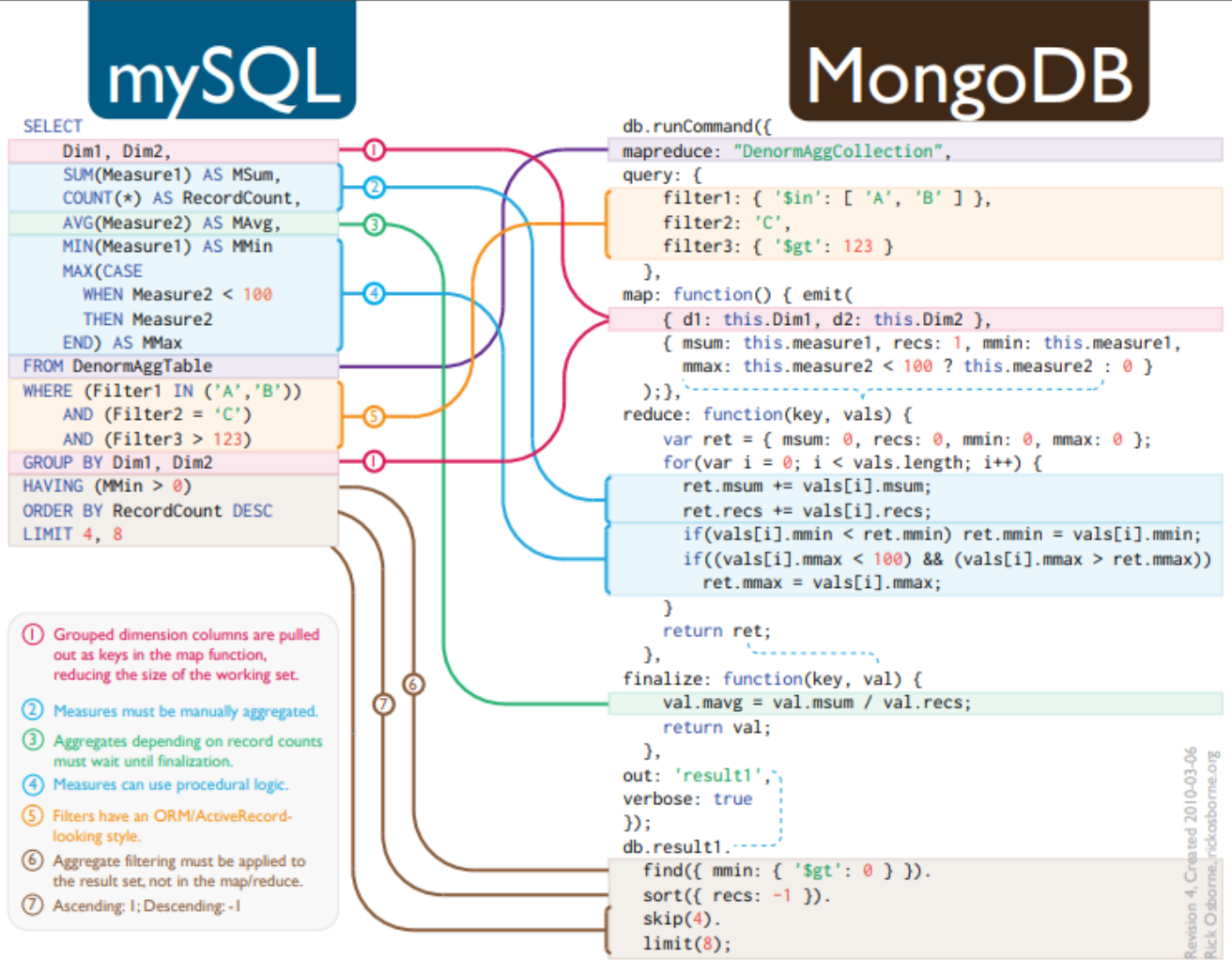
- Create Database
  - Couch → REST
    - PUT➔ Create
    - POST ➔ Insert
    - GET ➔ Select
    - DELETE ➔ Delete

curl -X PUT http://127.0.0.1:5984/fruit

curl -X POST –D '{"Abacaxi":"190"}'http://127.0.0.1:5984/fruit

Query


mongoDB


mySQL
MongoDB
SELECT
Dim1, Dim2,
SUM(Measure1) AS MSum,
COUNT(*) AS RecordCount,
AVG(Measure2) AS MAvg,
MIN(Measure1) AS MMin
MAX(CASE
WHEN Measure2 < 100
THEN Measure2
END) AS MMax
FROM DenormAggTable
WHERE (Filter1 IN ('A','B'))
AND (Filter2 = 'C')
AND (Filter3 > 123)
GROUP BY Dim1, Dim2
HAVING (MMin > 0)
ORDER BY RecordCount DESC
LIMIT 4, 8
db.runCommand({
mapreduce: "DenormAggCollection",
query: {
filter1: { '$in': ['A', 'B'] },
filter2: 'C',
filter3: { '$gt': 123 }
},
map: function() { emit(
{ d1: this.Dim1, d2: this.Dim2 },
{ msum: this.measure1, recs: 1, mmin: this.measure1,
mmax: this.measure2 < 100 ? this.measure2 : 0 }
);},
reduce: function(key, vals) {
var ret = { msum: 0, recs: 0, mmin: 0, mmax: 0 };
for(var i = 0; i < vals.length; i++) {
ret.msum += vals[i].msum;
ret.recs += vals[i].recs;
if(vals[i].mmin < ret.mmin) ret.mmin = vals[i].mmin;
if((vals[i].mmax < 100) && (vals[i].mmax > ret.mmax))
ret.mmax = vals[i].mmax;
}
return ret;
},
finalize: function(key, val) {
val.mavg = val.msum / val.recs;
return val;
},
out: 'result1',
verbose: true
});
db.result1.
find({ mmin: { '$gt': 0 } }).
sort({ recs: -1 }).
skip(4).
limit(8);
1 Grouped dimension columns are pulled out as keys in the map function, reducing the size of the working set.
2 Measures must be manually aggregated.
3 Aggregates depending on record counts must wait until finalization.
4 Measures can use procedural logic.
5 Filters have an ORM/ActiveRecord-looking style.
6 Aggregate filtering must be applied to the result set, not in the map/reduce.
7 Ascending: 1; Descending: -1
Revision 4, Created 2010-03-06
Rick Osborne, rickosborne.org

OBRIGADO.