Back to the future :
SQL 92 for Elasticsearch ?

@LucianPrecup

@nosqlmatters #nosql14

2014-09-04

whoami

- CTO of Adelean (http://adelean.com/, http://www.elasticsearch.com/about/partners/)
- Integrate search, nosql and big data technologies to support ETL, BI, data mining, data processing and data visualization use cases.

Poll - How many of you …

- Know SQL ?
- Are familiar with the NoSQL theory ?
- Are familiar with Elasticsearch ?
- Lucene ? Solr ?
- Used a NoSQL database or product ?
- Are remembering SQL 92 ?

SQL 92 ? NoSQL ?

SQL ? SQL 92 ? RDBMS ?

- SQL
  - Structured Query Language
  - Based on relational algebra
- **Designed for RDMBSes**
  - Relational Database Management Systems
- SQL 92
  - 700 pages of specification
  - Standardization
  - No vendor lock in ?

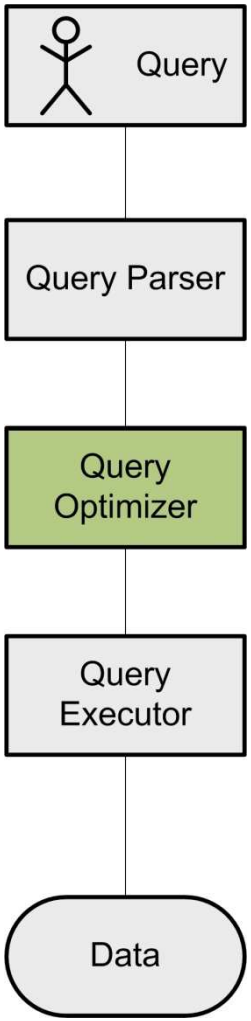
NoSQL ? Elasticsearch ?

- NoSQL
  - At first : the name of an event
  - Distributed databases
  - Horizontal scaling
- Standardization ?
- **Polyglot persistence**
- The language
  - Low level : speak the “raw data ” language
- Elasticsearch Query DSL

Why this presentation ?

- The title is voluntarily provocative
  - Back in ‘92, the dream (or nightmare) of any database vendor was to be SQL 92 compliant
- Good occasion to do a comparison
  - And who knows : the history might repeat :-)
- Elasticsearch users often ask questions about how to express a SQL query with Elasticsearch
  - However this will not going to be exhaustive about the subject

The "Query Optimizer"

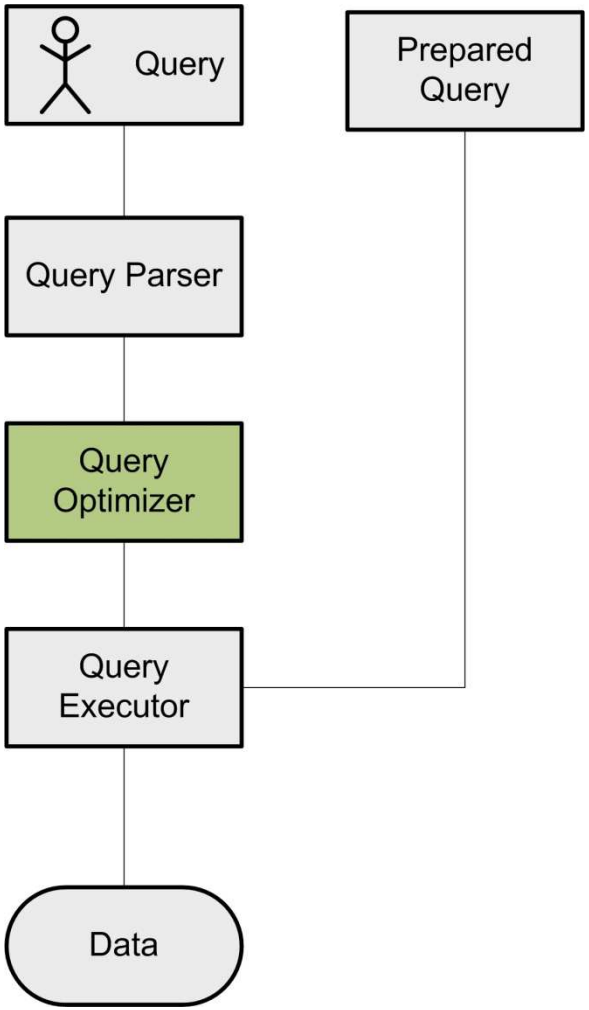



```
SELECT DISTINCT offer_status FROM offer;
                    III
SELECT offer_status FROM offer GROUP by offer_status;
```

```
SELECT O.id, O.label
FROM offer O
WHERE O.offer_status IN (
            SELECT S.id FROM offer_status S)
                    III
SELECT O.id, O.label
FROM offer O, offer_status S
WHERE O.offer_status = S.id
```

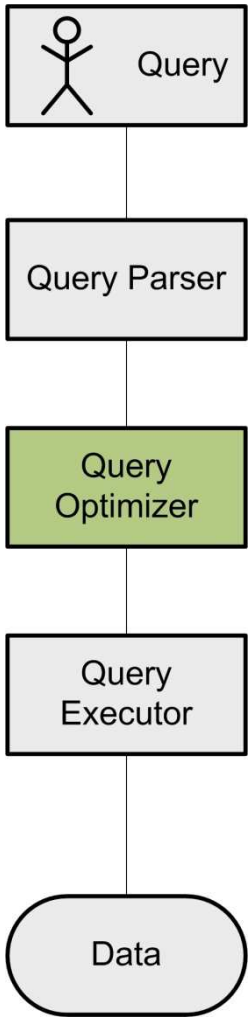
The "Query Optimizer"

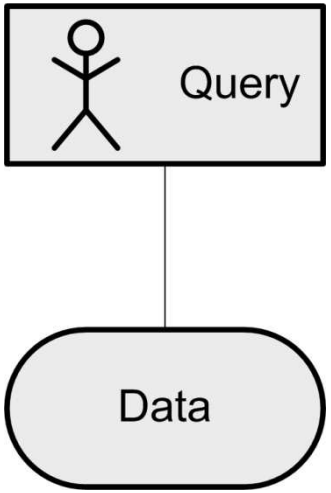
SQL/RDBMS →Power to the DBA



The "Query Optimizer"

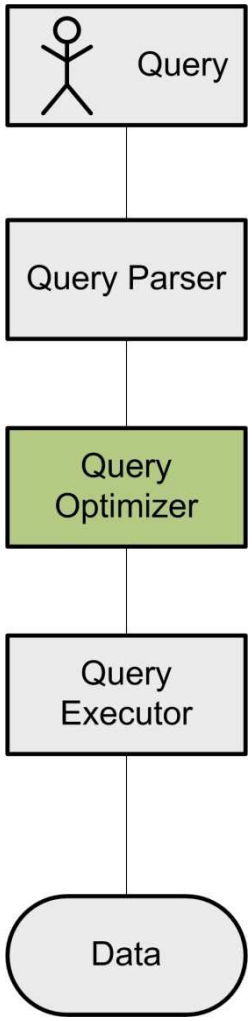
SQL/RDBMS →Power to the DBA **NoSQL**



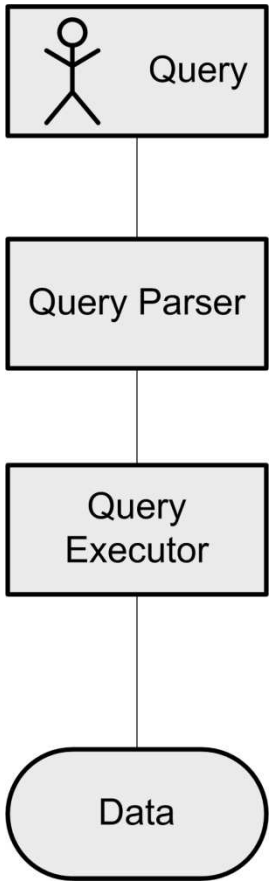


The "Query Optimizer"

SQL/RDBMS →Power to the DBA

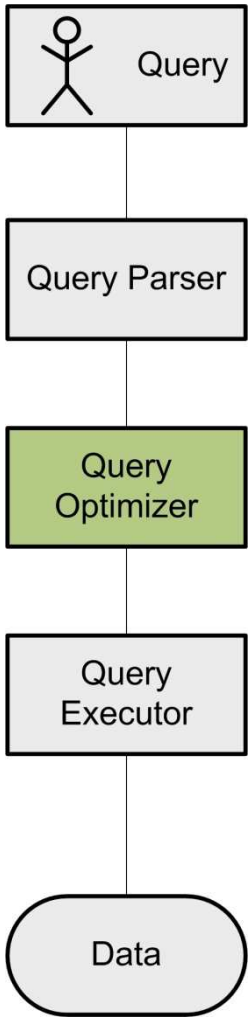


NoSQL

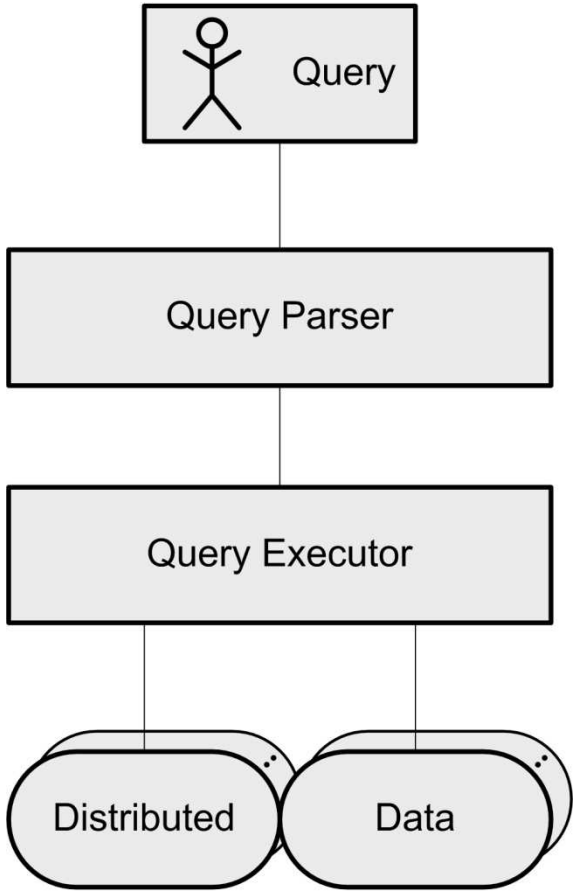


The "Query Optimizer"

SQL/RDBMS →Power to the DBA

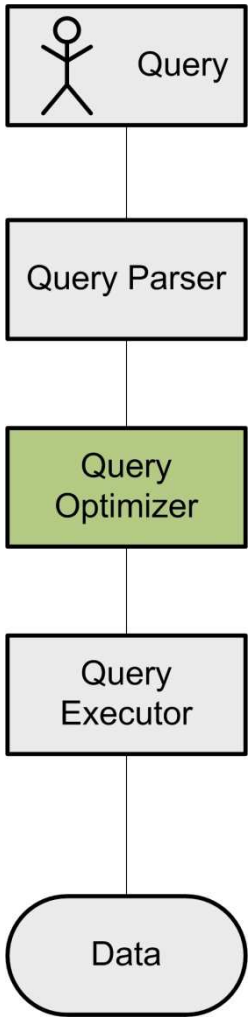


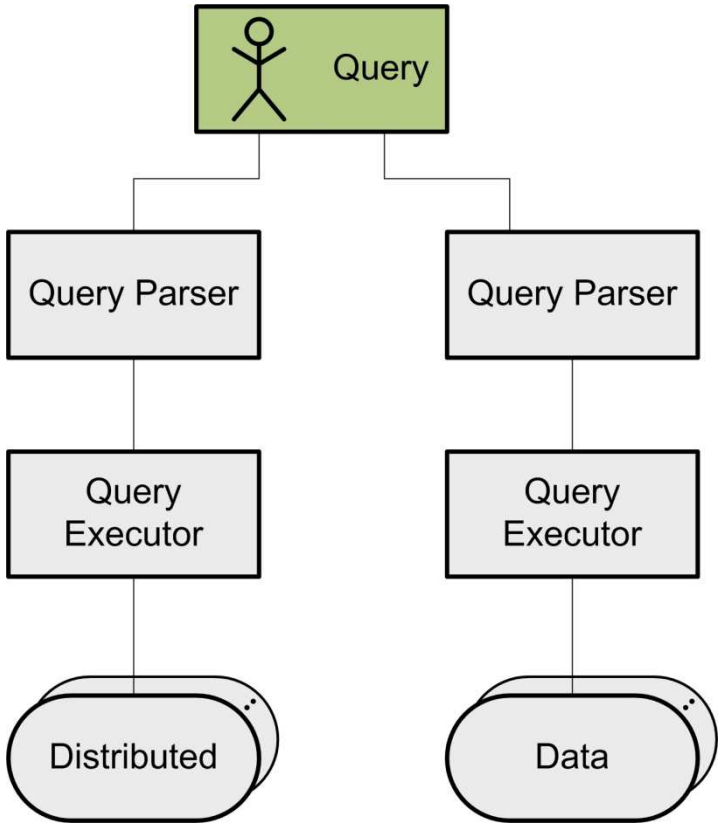
NoSQL



The "Query Optimizer"

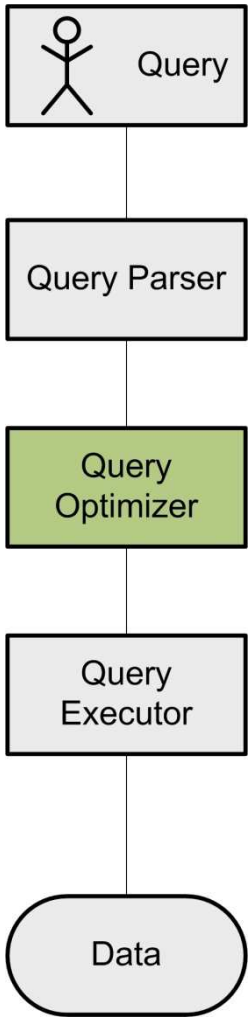
SQL/RDBMS →Power to the DBA **NoSQL**



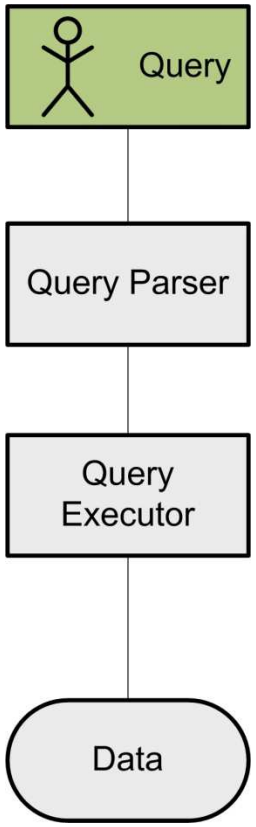


The "Query Optimizer"

SQL/RDBMS →Power to the DBA



NoSQL →Power to the developer

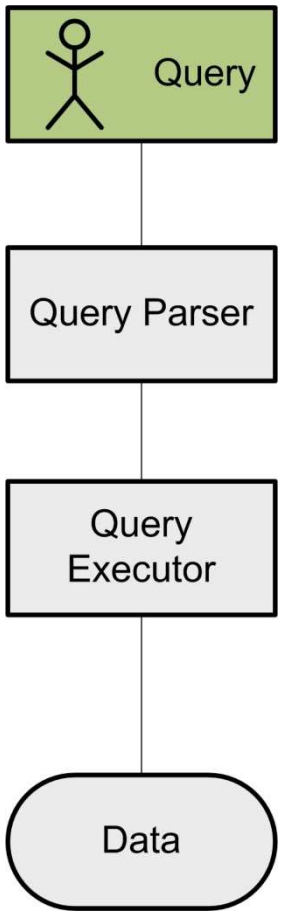


“With great power comes great responsibility”

- The developer has to :
  - Deal with query optimization
  - Deal with data storage
  - Take care about data consistency
  - …

- But the developer can do better than the query optimizer → adjusting (the data) to the (very) specific needs

Great responsibility … with Elasticsearch



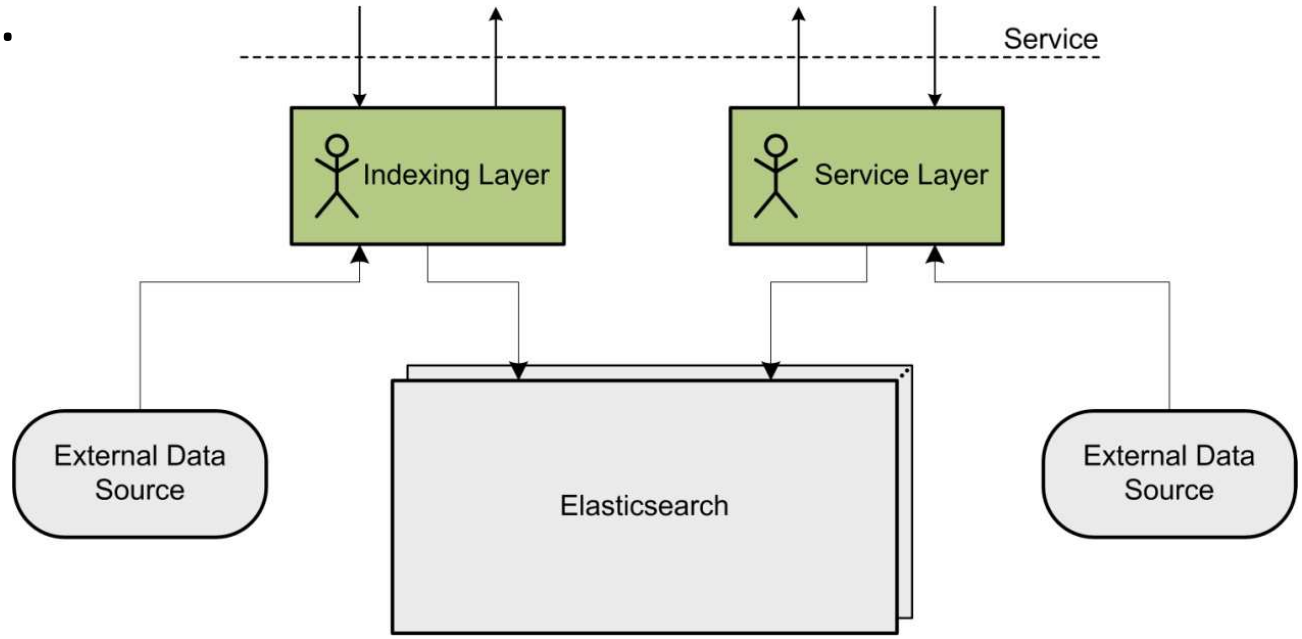

```
"fields": ["@timestamp"],
 "from": 0, "size": 1,
 "sort": [{ "@timestamp": { "order": "desc" }}],
 "query": { "match_all": {} },
 "filter": { "and": [
    {"term": {"account": "you@me.org"}},
    {"term": {"protocol": "http"}}
  ]
 }
```

III

```
"from": 0, "size": 0,
 "query": { "filtered": {"query": {"match_all": {}},
    "filter": { "bool": { "must": [
         {"term": {"account": "you@me.org"}},
         {"term": {"protocol": "http"}}
       ]}}}
 },
 "aggs": {"LastTimestamp": {"max": {"field": "@timestamp"}}}
```

What SQL 92 for Elasticsearch would imply ?

- Syntax → not important
- Focus on functionality
- Take advantage of the fact that the database is no longer the center of the information system. The service layer is.



Side by side - pagination

As we will use this difference in some choices

- Statement.execute()
- do while ResultSet.next()
  - ResultSet.get()
- Otherwise: no standard for pagination in SQL 92

- Pagination is at the core of search engines
- Top n results are returned fast and use cases usually stop to that

Side by side - decimals

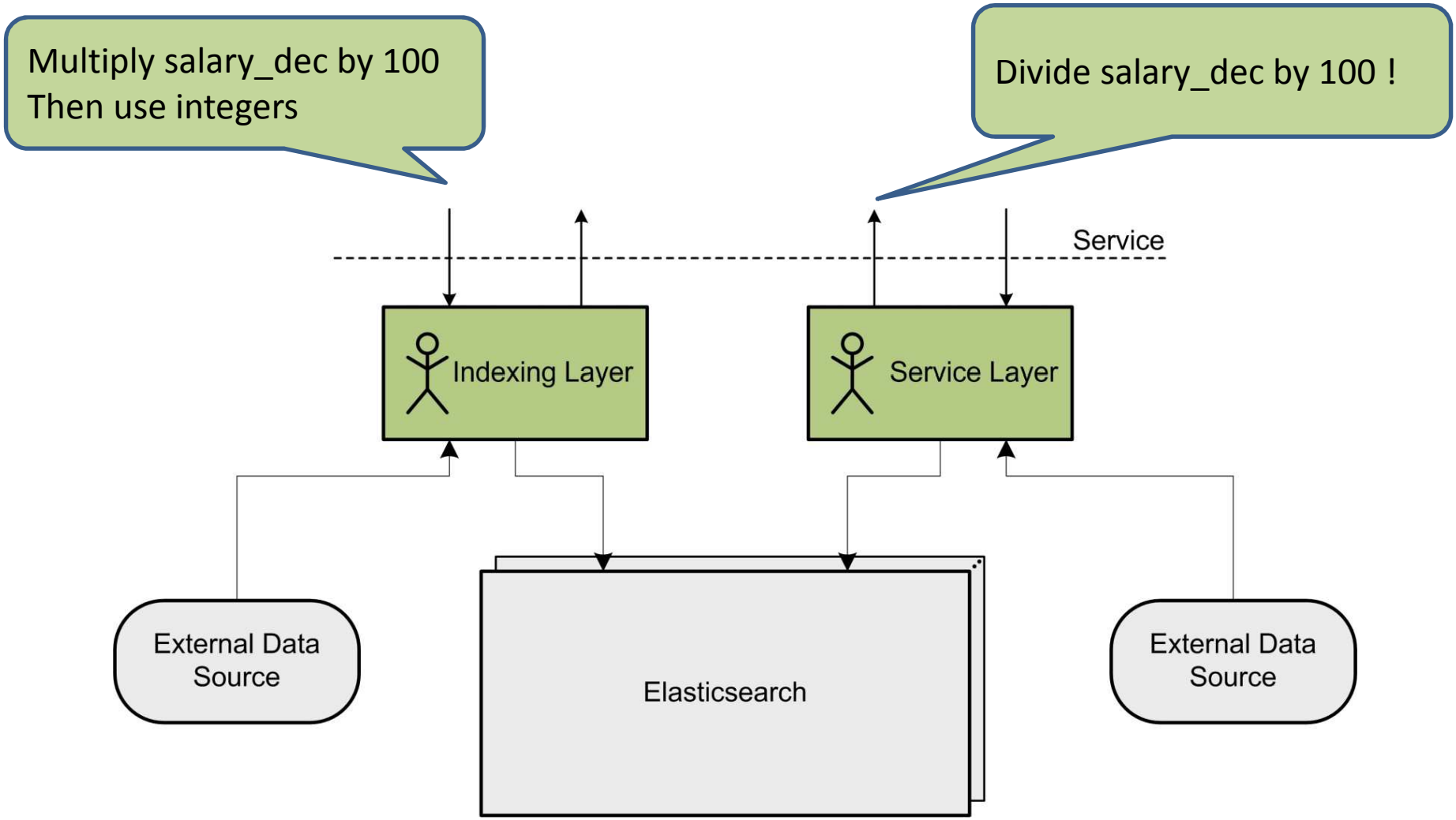
As SQL 92 introduced some new types

```
CREATE TABLE test_decimal(
    salary_dec DECIMAL(5,2),
    salary_double DOUBLE);
INSERT INTO test_decimal(
    salary_dec, salary_double)
    values (0.1, 0.1); X 10
SELECT SUM(salary_dec)
    FROM test_decimal;
→1.00
SELECT SUM(salary_double)
    FROM test_decimal;
→0.9999999999999999
```

This fits
But 0.00001 **X 10** does not
→ 0.00010000000000000002

```
PUT test_index/test_decimal/_mapping
"test_decimal" : {
    "salary_float" : {"type" : "float" },
    "salary_double" : {"type" : "double" },
    "salary_string" : {"type" : "string", "index": "not_analyzed"
}
POST test_index/test_decimal
{"salary_float" : 0.1,"salary_double" : 0.1,"salary_string" :
    "0.1"} X 10
POST test_index/test_decimal/_search
"size": 0,   "aggs": {
   "FloatTotal": {"sum": { "field" : "salary_float" }},
   "DoubleTotal": {"sum": { "field" : "salary_double" }}
}
→  "FloatTotal": {"value": 1.0000000149011612},
    "DoubleTotal": {"value": 1}
```

Decimals for Elasticsearch – the solution

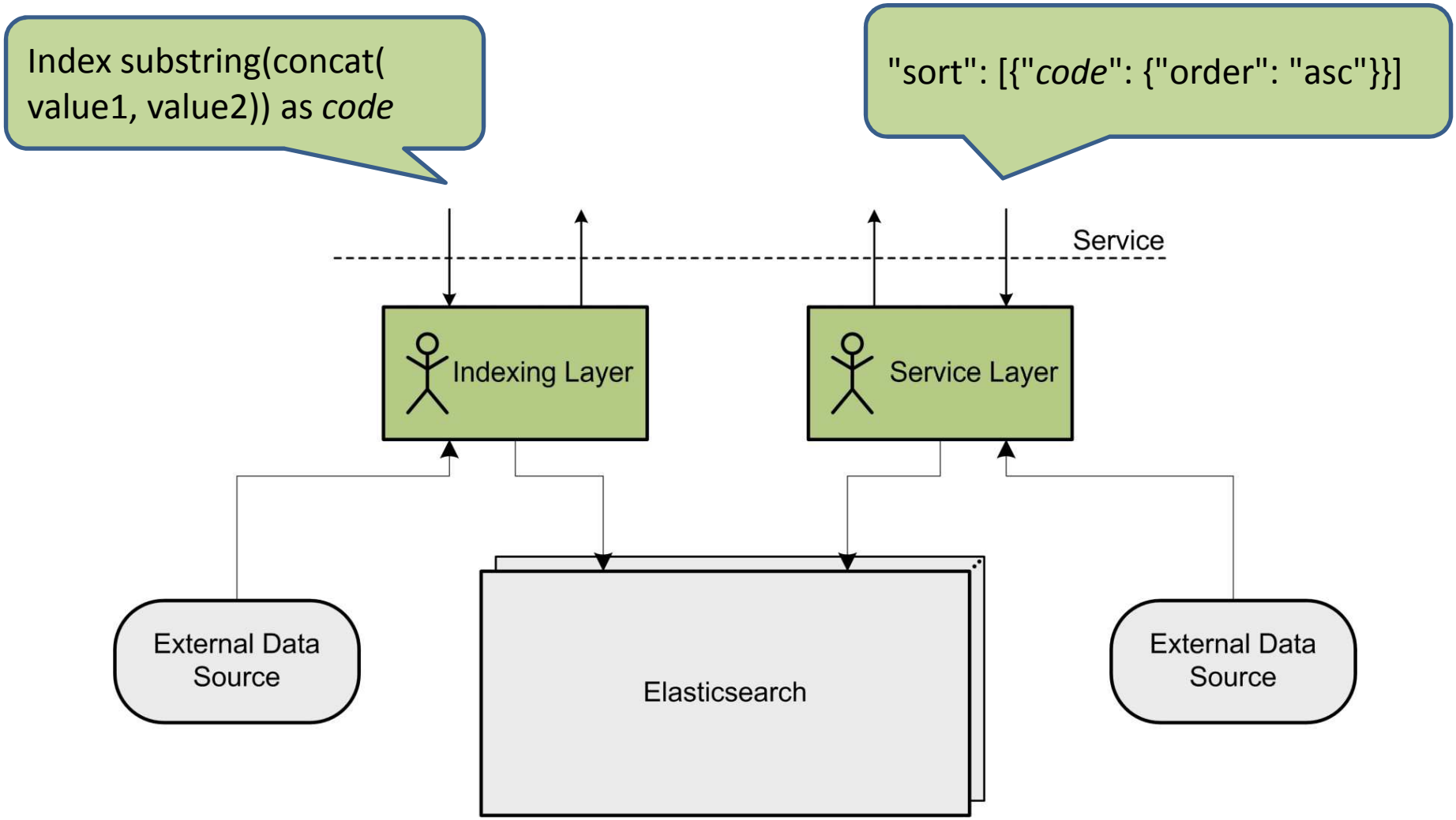


Side by side – order by

- SELECT * FROM offer ORDER BY price;
- SELECT (price_ex + price_vat) AS price FROM offer ORDER BY price;
- SELECT substring(concat(value1, value2)) AS code FROM table ORDER BY code

- "query": {"match_all": {}},
  "sort": [{"price": {"order": "asc"}}]
- "function_score": {"boost_mode": "replace",
  "script_score": {"script":
  "doc['price_ex'].value + doc['price_vat'].value"}}
- Let's do the computations at index time !
- Watch out for order by + pagination + distributed

Order by - computations at index time



Side by side - count

The simplest aggregation

- SELECT COUNT(*) FROM offer;
- POST index/_count
  {"query" : {"match_all": {}}}

- SELECT COUNT(*) FROM offer WHERE price > 10;
- POST index/_count
  "query": {"filtered": {
  "filter": {"range": {"price": {"from": 10}}}}}

- POST index/_search
  "size": 0,
  "aggs": {"Total": {"value_count": { "field" :
  "price" }}}

Side by side - other aggregations

- SELECT SUM(price) FROM offer;
- SELECT AVG(price) FROM offer;
- SELECT MAX(price) FROM offer;

- POST index/_search
  "size": 0,
  "aggs": {"Total": {"sum": { "field" : "price" }}}
- POST index/_search
  "size": 0,
  "aggs": {"Average": {"avg": { "field" : "price" }}}
- POST index/_search
  "size": 0,
  "aggs": {"Maximum": {"max": { "field" : "price" }}}

Side by side – distinct and group by

- SELECT DISTINCT offer_status FROM offer;
- SELECT * FROM offer GROUP BY offer_status;

- "size": 0,
  "aggs": {"Statuses": {"terms": {
  "field" : "offer_status.raw" }}}

Side by side – distinct and group by

- SELECT * FROM offer GROUP BY offer_status;

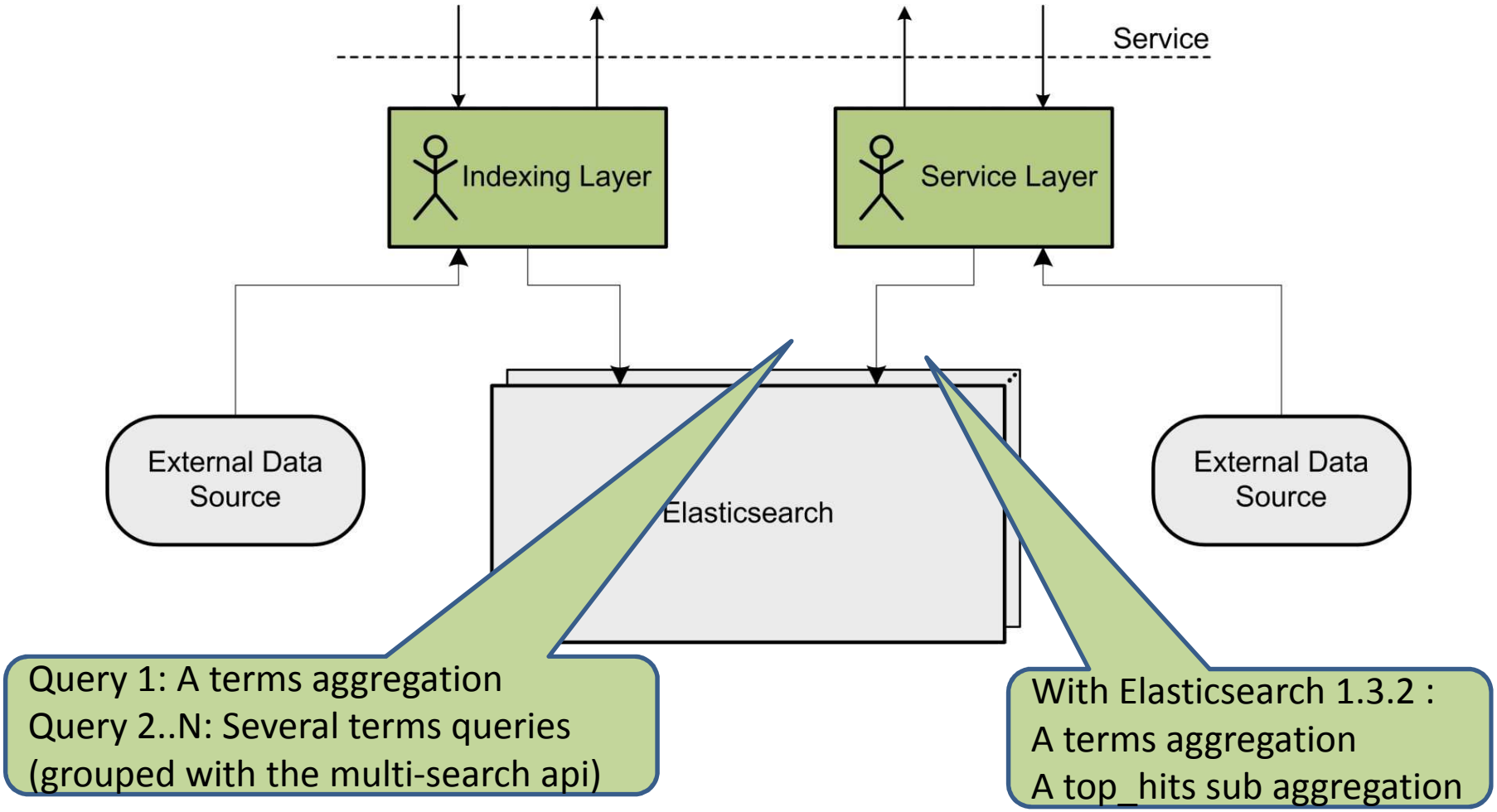
- "size": 0,
  "aggs": {"Statuses": {"terms": { "field" :
  "offer_status.raw" }}}

- "query": {"filtered": {
  "filter": {"term": {"offer_status.raw": "on_line"}}}

  "query": {"filtered": {
  "filter": {"term": {"offer_status.raw": "off_line"}}}

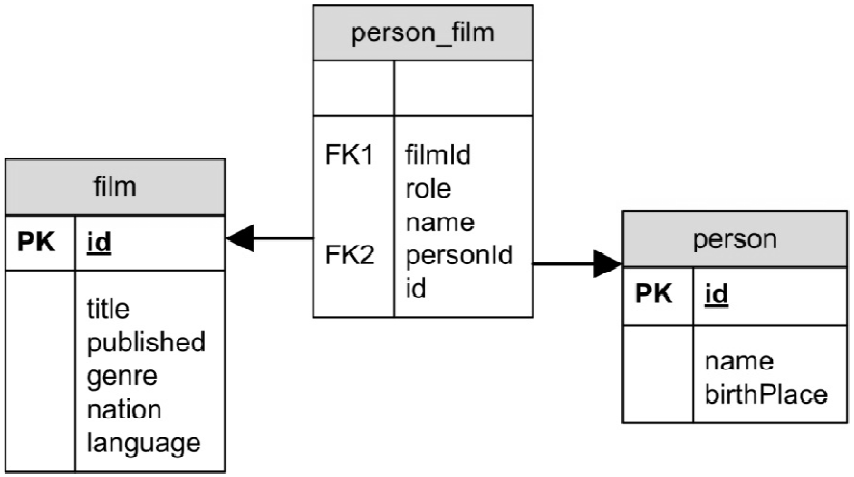
- "size": 0,
  "aggs": {"Statuses": {"terms": { "field" :
  "offer_status.raw" },
  "aggs": {"Top hits": {"top_hits": {"size": 10}}}}}

Implementing GROUP BY



Side by side – joins

Normalized database

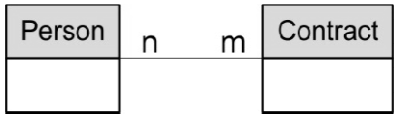


Elasticsearch document

```
{"film" : {
    "id" : "183070",
    "title" : "The Artist",
    "published" : "2011-10-12",
    "genre" : ["Romance", "Drama", "Comedy"],
    "language" : ["English", "French"],
    "persons" : [
      {"person" : { "id" : "5079",  "name" : "Michel Hazanavicius",  "role" : "director" }},
      {"person" : { "id" : "84145",  "name" : "Jean Dujardin",  "role" : "actor" }},
      {"person" : { "id" : "24485",  "name" : "Bérénice Bejo",  "role" : "actor" }},
      {"person" : { "id" : "4204",  "name" : "John Goodman",  "role" : "actor" }}
    ]
}}
```

The issue with joins :-)

- Let's say you have two relational entities: Persons and Contracts
  - A Person has zero, one or more Contracts
  - A Contract is attached to one or more Persons (eg. the Subscriber, the Grantee, …)



- Need a search services :
  - S1: getPersonsDetailsByContractProperties
  - S2: getContractsDetailsByPersonProperties
- Simple solution with SQL:

```
SELECT P.* FROM P, C WHERE P.id = C.pid AND C.a = 'A‘
SELECT C.* FROM P, C WHERE P.id = C.pid AND P.a = 'A'
```

The issue with joins - solutions

- Solution 1
  - Index Persons with Contracts together for S1
    {"person" : { "details" : …, … , "contracts" : ["contract" :{"id" : 1, …}, …] }}
  - Index Contracts with Persons together for S2
    {"contract" : { "details" : …, …, "persons" : ["person" :{"id" : 1, "role" : "S", …}, …]}}
- Issues with solution 1:
  - A lot of data duplication
  - Have to get Contracts when indexing Persons and vice-versa
- Solution 2
  - Elasticsearch's Parent/Child
- Issues with solution 2:
  - Works in one way but not the other (only one parent for n children, a 1 to n relationship)
- Solution 3
  - Index Persons and Contracts separately
  - Launch **two Elasticsearch queries** to get the response
  - For S1 : First get all Contract ids by Contract properties, then get Persons by Contract ids (terms query or mget)
  - For S2 : First get all Persons ids by Person properties, then get Contracts by Person ids (terms query or mget)
  - The response to the second query can be returned "as is" to the client (pagination, etc.)

Side by side - having

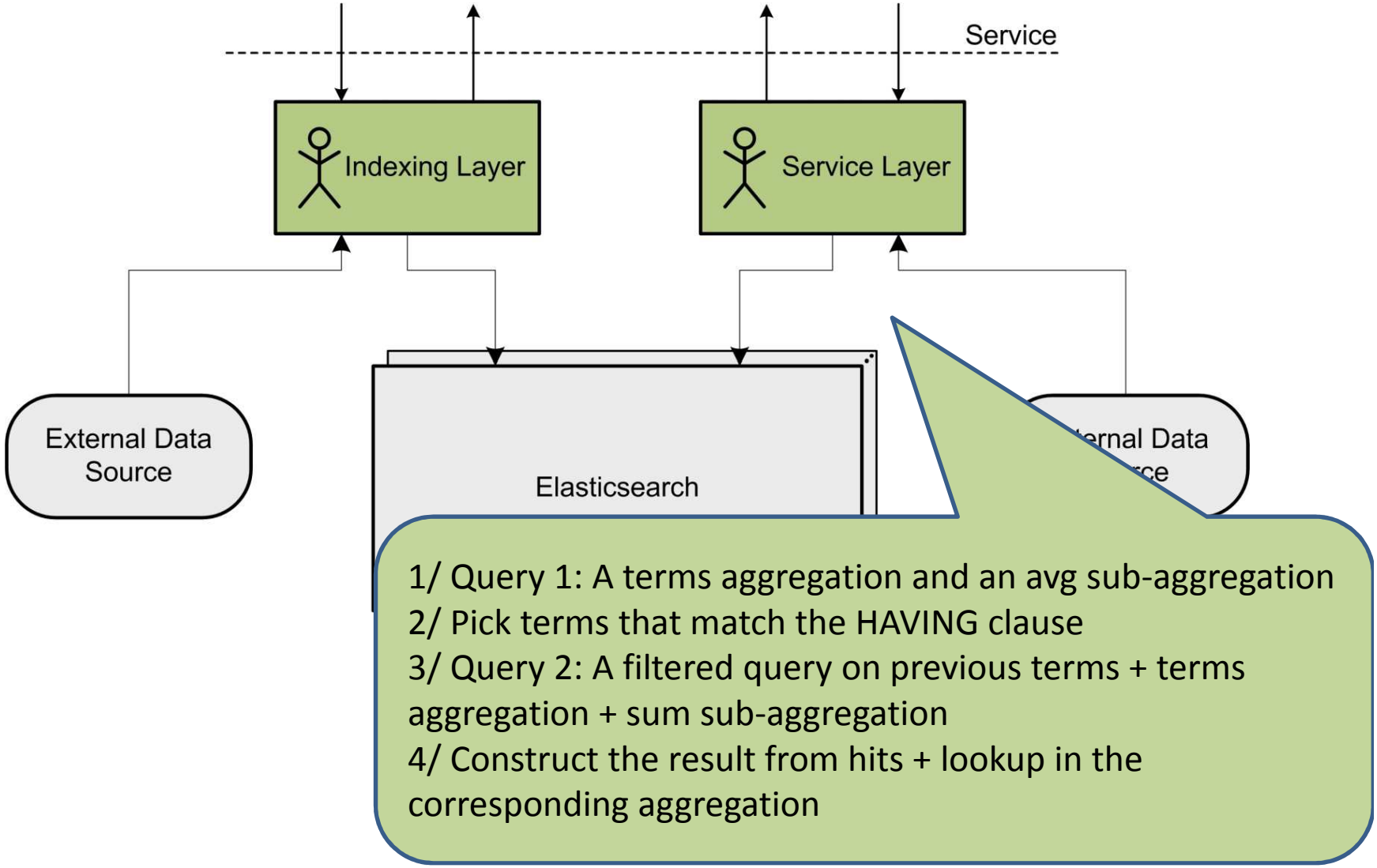
Also specified by SQL 92

- SELECT *, SUM(price)
  FROM offer
  GROUP BY offer_status
  HAVING AVG(price) > 10;

- ```
  "size": 0,
  "aggs": {
   "Status": {"terms": {"field": "offer_status"},
    "aggs": {
     "Average": {"avg": {"field": "price_ht"}}}}
  }
  ```

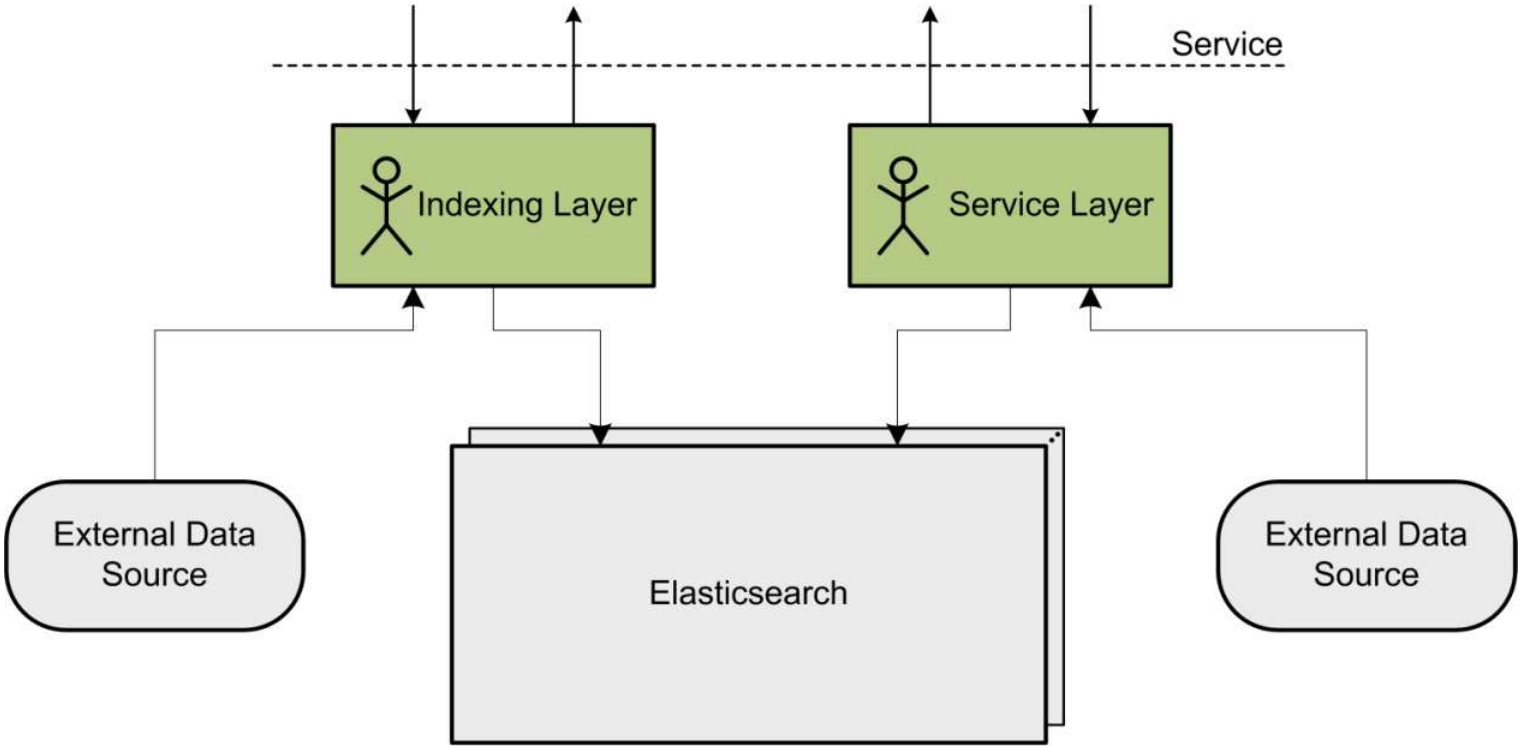
- ```
  "query": {
     "filtered": {"filter": {
         "terms": {"offer_status": ["on_line"]}}}},
   "aggs": {
     "Status": {"terms": {"field": "offer_status"},
     "aggs": {
       "Total": {"sum": {"field": "price_ht"}}}}}
  ```

Implementing HAVING



Conclusion

- The service layer is the center of the system
- The developer has the power :-)



Thank you

Q & A