Objectives

- Elasticsearch
- MongoDB

(Poor) Example code

- `for(; iter.hasNext();) { ... }`
- `!StringUtils.isNotEmpty(str)`

Review

- What data storage/search mechanisms have we talked about so far?
  - How do we access them?
  - What can we do with them?
  - What are their relative strengths and limitations?

Comparing Relational Databases and Elasticsearch

Operation	SQL	Elasticsearch
Interface	SQL interactive interface	REST API
Connectors	JDBC, ODBC	Clients for many programming languages

CRUD Operations Comparison

Operation	SQL	Elasticsearch
Create		
Read		
Update		
Delete		

CRUD: Good buzzword!

CRUD Operations Comparison

Operation	SQL	Elasticsearch
Create	INSERT INTO	PUT
Read	SELECT	GET
Update	UPDATE	PUT
Delete	DELETE FROM	DELETE

CRUD: Good buzzword!

DB Popularity

http://db-engines.com/en/ranking

322 systems in ranking, March 2017

Rank Mar 2017	Rank Feb 2017	Rank Mar 2016	DBMS	Database Model	Score Mar 2017	Score Feb 2017	Score Mar 2016
1.	1.	1.	Oracle	Relational DBMS	1399.50	-4.33	-72.51
2.	2.	2.	MySQL	Relational DBMS	1376.07	-4.23	+28.36
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1207.49	+4.04	+71.00
4.	4.	↑ 5.	PostgreSQL	Relational DBMS	357.64	+3.96	+58.01
5.	5.	↓ 4.	MongoDB	Document store	326.93	-8.57	+21.60
6.	6.	6.	DB2	Relational DBMS	184.91	-2.99	-3.02
7.	↑ 8.	7.	Microsoft Access	Relational DBMS	132.94	-0.45	-2.09
8.	↓ 7.	8.	Cassandra	Wide column store	129.19	-5.19	-1.14
9.	9.	↑ 10.	SQLite	Relational DBMS	116.19	+0.88	+10.42
10.	10.	↓ 9.	Redis	Key-value store	113.01	-1.03	+6.79
11.	11.	11.	Elasticsearch	Search engine	106.23	-2.08	+26.06
12.	12.	↑ 13.	Teradata	Relational DBMS	73.53	-2.06	-0.53
13.	13.	↓ 12.	SAP Adaptive Server	Relational DBMS	70.13	-1.61	-6.52
14.	14.	14.	Solr	Search engine	63.99	-3.70	-5.38
15.	15.	15.	HBase	Wide column store	58.98	-0.26	+6.57
16.	↑ 17.	↑ 17.	FileMaker	Relational DBMS	54.57	-0.62	+6.64

Overarching Question

In one day:
Walmart processes over 40 Petabytes of data
600 TB of data add to Facebook
500 million tweets on Twitter
…

How to **store**, **query** and **process** these data efficiently?

Likely not a one-size-fits-all solution; need a combination

Limitations with Relational Database

- Overhead for complex select, update, delete operations
  - Select: Joining too many tables creates a huge table
  - Update: Each update may affect other tables
  - Delete: Must guarantee the consistency of data
- Mix of unstructured data is not well-supported
- Doesn't scale well with very large data

NoSQL is a good solution to deal with these problems.

Elasticsearch Architecture

- Node
  - A running ES instance
    - a process running on a machine
- Cluster
  - **Distributed** ES system made of several ***nodes***
  - Dynamic master election, no single node fail
    - fail as a whole
  - Communication between nodes and data distribution and balancing is automatically handled
  - View as a whole from outside

Elasticsearch Architecture

- Index
  - Multiple index support
  - Multiple types inside indices
- Shard
  - Building blocks of index
    - index is divided into shards
  - Each shard is an Apache Lucene index
  - Shards will be placed on different machines
  - ES sends queries to relevant shards and merges results
- Replica
  - Each shard can have 0 or more replicas
  - True copy of primary shard
  - Increase system fault tolerance and search performance

Elasticsearch Search

```
curl -XGET 'localhost:9200/bank/
_search?pretty' -H 'Content-Type:
application/json' –d'{
"query" : { "term" :
{ "account_number" : 995 } } }'
```

Elasticsearch Search

```
curl -XGET 'localhost:9200/bank/_search?pretty' -H 'Content-Type: application/json' -d'{
  "query" : {
      "constant_score" : {
          "filter" : {
              "term" :
{ "account_number" : 995
              }
          }
      }
  }
}'
```


Programmatic API Example: JDBC

```
public User getUser(int user_id) {
   Connection con = getConnection();
   PreparedStatement pstmt;
   ResultSet rs;
   User u = new User();
   try {
       pstmt = con.prepareStatement("SELECT username FROM
users WHERE id=?");
       pstmt.setInt(1, user_id);
       rs = pstmt.executeQuery();
       while (rs.next()) {
           String username = rs.getString(1);
           u = getUser(username);
       }
       pstmt.close();   rs.close();  con.close();
   } catch (SQLException e) {
       …
   }
   return u;
}
```

Programmatic API Example: Elasticsearch

```
response = client.prepareSearch(ES_INDEX_NAME)
.setTypes(ES_TYPE_NAME)
.setQuery(query)
.addFields("id", CITY_FIELD_NAME, INSULA_ID_FIELD_NAME,
INSULA_NAME_FIELD_NAME, PROPERTY_ID_FIELD_NAME,
"property.property_number", "property.property_name",
PROPERTY_TYPES_FIELD_NAME, "edr_id", "bibliography",
WRITING_STYLE_IN_ENGLISH_FIELD_NAME,
LANGUAGE_IN_ENGLISH_FIELD_NAME, "cil", "description",
"comment", "content_translation", "measurements")
.setSize(NUM_RESULTS_TO_RETURN)
.addSort("edr_id", SortOrder.ASC)
.execute().actionGet();

for (SearchHit hit : response.getHits()) {
    inscriptions.add(hitToInscription(hit));
}
```

(truncated)

What design pattern is this using?

MONGODB

Overview

- An open source and document-oriented database
- Data is stored in JSON-like documents
  - NoSQL
- Designed with both scalability and developer agility
- Dynamic schemas
  - collections hold a bunch of documents
  - Number of fields, content and size of the document can differ from one document to another

Terminology

- Database → Database
- Table → Collection
- Record / Row → Document
- Column → Field

mongodb

- show dbs
- show collections

- To switch database
  - use database

CRUD

- Create
  - db.collection.insert(<document>)
  - db.collection.save(<document>)
- Read
  - db.collection.find(<query>, <projection>)
  - db.collection.findOne(<query>, <projection>)
- Update
  - db.collection.update(<query>, <update>, <options>)
- Delete
  - db.collection.remove(<query>, <justOne>)

CRUD Examples

```
> db.user.insert({
    first: "John",
    last : "Doe",
    age: 39
})
```

```
> db.user.find ({
    "first" : "John",
    "last" : "Doe",
    "age" : 39
});
```

```
> db.user.update(
    {"_id" :
ObjectId("51…")},
    {
      $set: {
        age: 40,
        salary: 7000}
    }
)
```

```
> db.user.remove({
    "first": /^J/
})
```

```
DELETE * FROM user WHERE
first LIKE “J%”;
```


Search: find()

- db.bank.findOne()
- db.bank.find()
- db.bank.find().pretty()
- db.bank.find({lastname: ”Johnson”})