NextGen Scientific Publication
American International Journal of Computer Science and Technology

Volume 1 Issue 5, Pg. No. 17-26, AIJCST-V1I5P103, 2019
https://doi.org/10.63282/3117-5481/AIJCST-V1I5P103

Original Article

Evolution of Data Management in Enterprise-from File System to Data Lakes

***Romeo Mehra**

Sr. IT Consultant/Lead, USA.

Abstract:

This paper will capture the evolution of high-level technical overview of the data technologies progressed from legacy file system to refined data stores adhering to specialized use cases that include managing both structured and unstructured data (including images, text, documents etc.). It starts with a brief overview of the data technologies that evolved over time and finally analyzes the strength and weakness with best fit use case. Overall, it covers the evolution with a detailed progression and uses comparative approach for real life use cases.

Keywords:

RDBMS, Dataware House, Data Lake, No SQL, Search Based Index.

Article History:

Received: 25.07.2019

Revised: 10.08.2019

Accepted: 22.08.2019

Published: 08.09.2019

1. Introduction

The primary factors the evolution of storage and retrieval architectures for data have been volume, variety, velocity and structure. This article will capture the evolution of these data technologies with focus on these driving factors. Throughout the journey of progression, the key technical aspects are covered and comparative analysis follows.

2. Technology Evolution

2.1. File based data systems

With the advent of computing storing data in files was most standard and primitive way to store the data. Initially it started with storing based direct storage on magnetic tapes and punched cards. The storage and retrieval were tightly coupled with application logic.

Following is the snapshot of how data is stored and accessed in flat file
Advantages:

- No complex architecture and maintenance required on storage and infrastructure
- Easy to migrate

Limitations of file structure:

- Write custom logic to update multiple files where data is related for e.g if John's salary is updated the application should write a logic to get employee id and update the salary.
- If another user is updating the salary at the same time the last recent one gets updated. There is no Isolation
- If the server crashes and data files are not manually backed up No Durability
- Inconsistent if one file gets updated and the other is missed in the program; the data is in inconsistent state.

Copyright by the Author(s). This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International (CC BY-NC-ND 4.0) License (https://creativecommons.org/licenses/by-sa/4.0/)

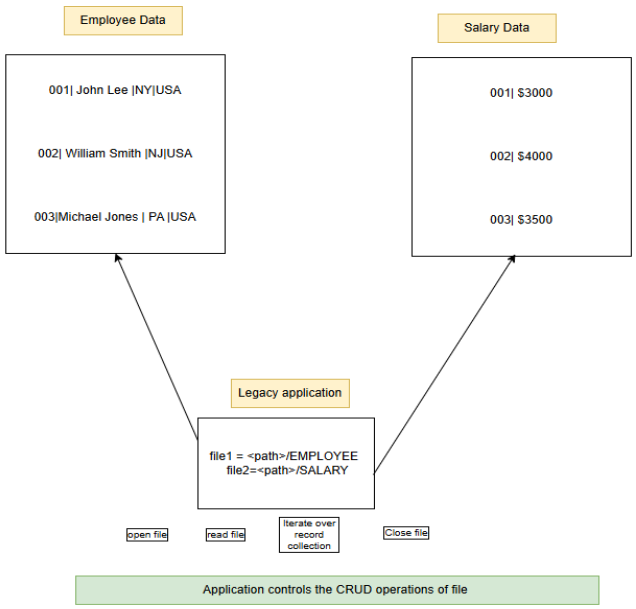


Figure 1. File-Based Data Management Using a Legacy Application with Separate Employee and Salary Records

2.2. Relational Model-RDBMS

Relational database was conceptualized by E.F Codd in 1970. This gave data storage a much-centralized approach. For a database management system to be truly described as relational he has described 12 rules. Major ones were Relational structure in form of tables, having an online catalog/metadata, Physical and logical data independence, managing data integrity constraints such as creating of primary, foreign and unique keys, setting default values for an attribute(column).

2.2.1. Transactions in RDBMS

Follow the standard Atomicity, Consistency, Integrity, Durability (ACID) rules which were missing in the file-based data systems, Data was structured and integrity was managed well.

2.2.2. Logical Representation of Data

The data is stored in tabular format

- Each tuple is a Row.
- Each Attribute is a Column.
- Relation is a Table.

Table 1. Employee Data in Relational Database Structure

EmployeeID	Name	State	Country
001	John Lee	NY	USA
002	William Smith	NJ	USA
003	Michael Jones	PA	USA

The data is structured using a relational concept as mentioned by Codd. Following constraints manage the:

- Primary key: The primary index and unique identifier (could be a business key or auto generated identifier)
- Foreign key: The reference key to another table primary key

OLTP (Online transaction processing) used relational concepts in SQL to join the tables based on keys and relations to retrieve the current state of data for business application.

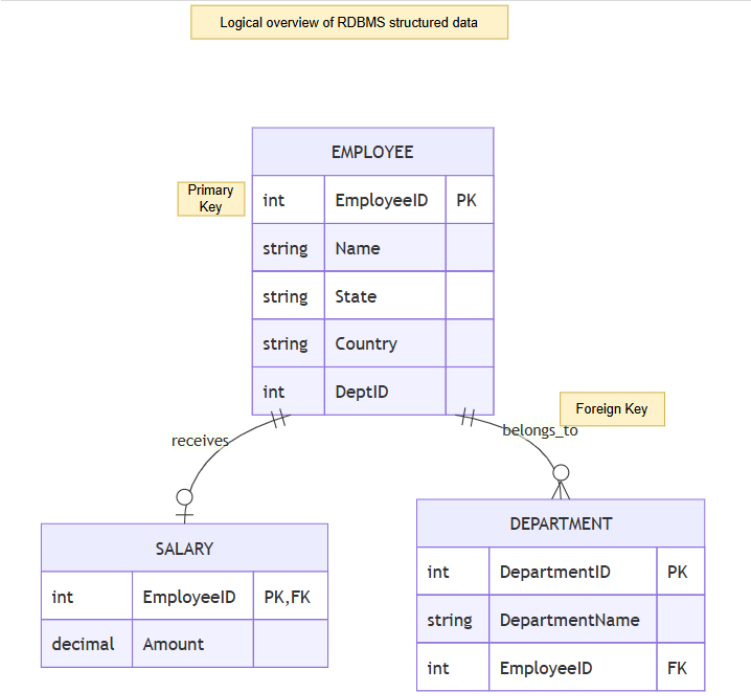


Figure 2. Logical overview of RDBMS Structured data

2.3. Data Warehouse

As the data grew in volume and there was a greater interest to understand and build trends and understand the patterns related to growth, decline and understand the history. Databases could not keep up with continuous historical data volumes and real time retrieval. The Business and Intelligence applications moved from Online transaction processing to Online Analytical Processing.

- Multiple data sources files, business database, images feed into a single Datawarehouse making it as a single source of truth.
- Data source can be file, structured tables and stored in different formats (ROLAP and MOLAP).

2.3.1. Logical overview of OLAP schema

Table 2. Employee Dimension table

EmployeeID	Name	Department	State	Country
001	John Lee	IT	NY	USA
002	William Smith	Engineering	NJ	USA
003	Michael Jones	Sales	PA	USA

Dimension: - Data context

Table 3. Salary Fact table

EmployeeID	Year	Salary
001	2015	50000
001	2016	55000
002	2015	30000
002	2016	40000
003	2016	42000

Fact: Numeric measures

Cubes:

These data points are aggregated over by dimensions and stored in disk by different dimensions for ready to view results (In relational world its SQL GROUP BY)

Totals by dimension:

- 1 D -> Year
- 2 D-> Year + Country, Year + Department
- 3 D-> Year + State + Country, Year+State+Department

Physical Storage
Stored in continuous series of values in a flat array structure disk

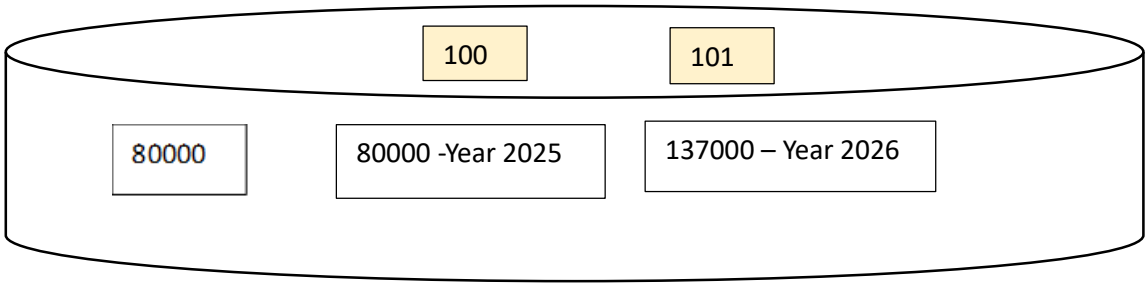


Figure 3. Logical Representation of Data Storage across Multiple Years

The aggregates are stored in disk in continuous address to reduce the computation overheads.

NoSQL:

While RDBMS and Datawarehouse systems were able to address the problems where the data had well defined structure, Unstructured data management was still an unsolved problem till late 2000s. In 2009 the Eric Evans and Johan Oskarsson came up with the term NoSQL for non-relational databases "Not only SQL". The document could have been stored either in-memory and disk.

The storage formats can be:

1. Key value : is defined by primary key and the value is rest of the columns

```
{
 "employee:001": {
  "EmployeeID": "001",
  "Name": "John Lee",
  "State": "NY",
  "Country": "USA"
 },
 "employee:002": {
  "EmployeeID": "002",
  "Name": "William Smith",
  "State": "NJ",
  "Country": "USA"
 },
 "employee:003": {
  "EmployeeID": "003",
  "Name": "Michael Jones",
  "State": "PA",
  "Country": "USA"
 }
}
```

2. Wide column store
   - "001" ——▶ [name: "John Lee", ts: 17180000] [state: "NY", ts: 17180000] [country: "USA", ts: 17180000]
   - "002" ——▶ [name: "William Smith", ts: 17180010] [state: "NJ", ts: 17180010] [country: "USA", ts: 17180010]
   - "003" ——▶ [name: "Michael Jones", ts: 17180020] [state: "PA", ts: 17180020] [country: "USA", ts: 17180020]

2.5. Search Engine storage -Apache Lucene

Databases solved the structured data access. Data warehousing handled management of massive historical data but searching on text data was unsolved Apache lucene text based pattern. The document is mapped and indexed for rapid revtrival.

DocID 0:
- EmployeeID: "001"
- Name: "John Lee"
- State: "NY"
- Country: "USA"

DocID 1:
- EmployeeID: "002"
- Name: "William Smith"
- State: "NJ"
- Country: "USA"

DocID 2:
- EmployeeID: "003"
- Name: "Michael Jones"
- State: "PA"
- Country: "USA"

Common Lucene Data stores: Elasticsearch, Apache SOLR, AWS OpenSearch

Batch ->Streaming. Thick client desktop-based apps shifted to thin client (web based) app.

2.6. Graph databases

RDBMS reach their peak capacity with joins when there too many interconnected relationships, this is where the graph databases come into picture. In a graph database each data node points to its neighboring nodes in the disk. They are best use for social networks, fraud detection and recommendations.

Graph Store:

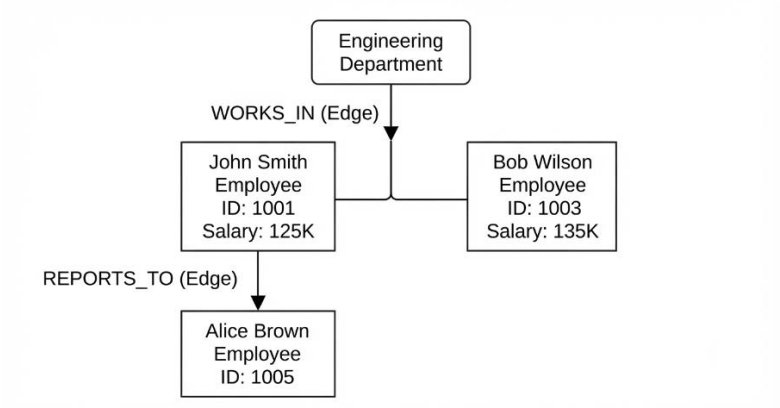


Figure 4. Employee Hierarchy and Reporting Structure

Graph database storage is based on graph theory. An entity is the vertex /node and relationship is the edge for e.g

- John = vertex
- Engineering = vertex
- WORKS_IN = edge
- Examples are Neo4j, Amazon Neptune

2.7. Data Lake:

The data lake were discovered in 2010 by James Dixon, with massive growth of unstructured data , a cost efficient and optimized storage mechanism was needed to handle the data volume and variety. The Datawarehouse and data marts were not able to handle large chunks of unstructured data. Another factor that acted as a catalyst for data lake growth is cheaper storage on public cloud providers (AWS and Azure)

2.7.1. Data processing

The data is pushed by batch processes that follow ELT pattern(Extract Load and Transform)

- Ingestion: Talend, Informatica , AWS Glue , Azure data factory
- Processing: Spark framework from various formats and finally convert to parquet format.
- Extraction: Engines like Amazon Athena, Databricks

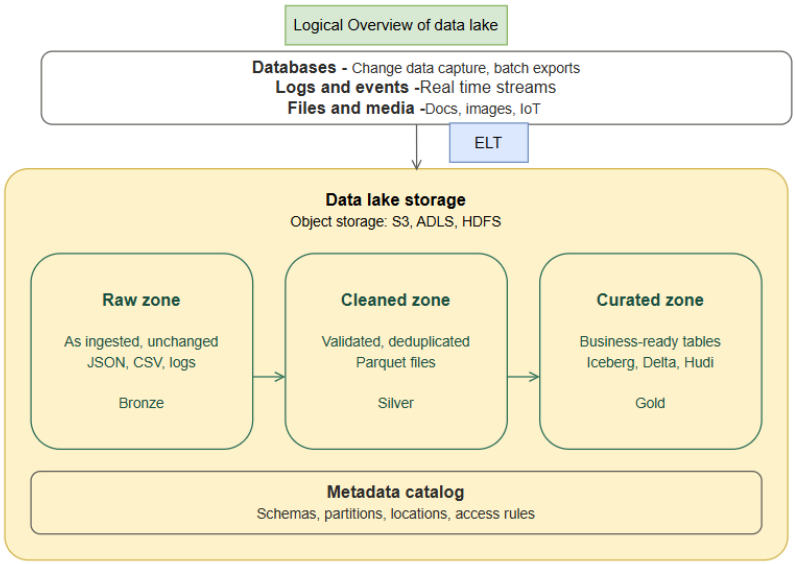


Figure 5. Logical Overview of Data Lake

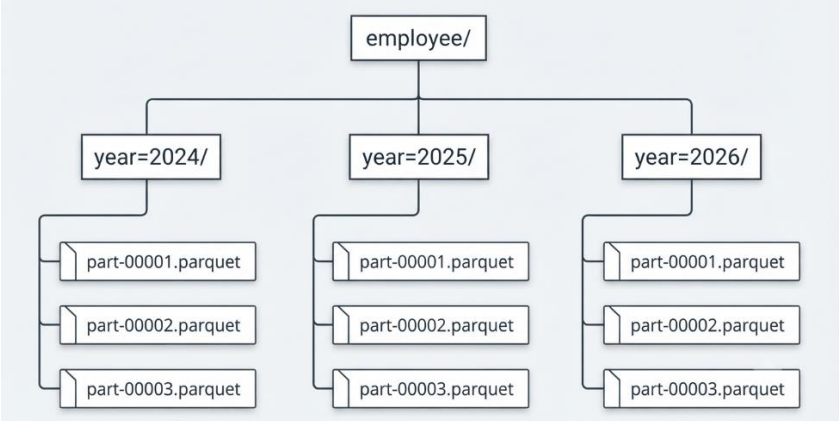


Figure 6. Physical storage in Data Lake

2.8. LakeHouse

Its an emerging technology that involves creating transactional table formats. One of the frameworks that are recently developed by Uber is Hudi. Lake house keeps original parquet files in data lake intact and creates meta data records to create the transactions.

3. Selection Matrix

3.1. Criteria

Following are selection criteria for selecting the data source.

Factor	Usage/Impact
Storage-How much data the technology can accommodate and how storage grows	Supports growth in dataset size and retention requirements
Data model-How data is represented and organized	Must match the structure and relationships in the workload
Access pattern-How data is typically read or retrieved	Point lookup, joins, scans, traversal, aggregation, and other patterns favor different architectures
Latency-Time required to retrieve or process data	Critical for real-time, interactive, and operational workloads
Throughput-Volume of data or operations processed per unit time	Important for high-volume workloads and large-scale ingestion
Consistency / transactions-Guarantees for correctness, atomicity, and isolation	Important for financial and operational workloads
Availability / fault tolerance-Ability to remain operational during failures	Important for enterprise systems requiring resilience
Cost-Infrastructure, storage, compute, licensing, and operational cost	Affects total cost of ownership
Concurrency-Ability to support simultaneous users and operations	Important for shared enterprise workloads
Data variety-Ability to handle structured, semi-structured, and unstructured data	Important for heterogeneous and modern data platforms
Real-time capability-Ability to ingest and process continuously	Important for event-driven and time-sensitive workloads
Ecosystem / integration-Compatibility with applications, tools, platforms, and services	Influences integration effort and adoption
Governance / security-Access control, auditing, lineage, compliance, and data management capabilities	Critical for enterprise and regulated data
Maturity-Technology maturity, community, tooling, and available skills	Influences adoption risk and long-term maintainability

4. Comparative Analysis of Datastores

Scale:

- General 1= Worst 5=Best
- Cost 1= Most expensive 5=Cheapest
- Complexity 1= Highly Complex 5= Simplest
- Structure 1= Lacks structure 5=Schema enforcement and highly structured

	Storage	Latency	Throughput	Scalability	Cost	Complexity	Flexibility	Structure	Leaders
File processing	2	2	3	1	5	4	4	1	Flat files csv , text
Relational DB	3	4	3	2	4	4	2	5	Oracle, MS-SQL

									Server, Postgres SQL
Data Warehouse	4	3	5	4	2	3	2	5	
HDFS	5	1	5	5	3	1	4	1	Hadoop, Spark
No SQL DB	4	5	5	5	3	3	5	2	MongoDB, Couchbase DB
Graph DB	2	4	3	2	2	2	4	3	Neo4j , Tigergraph
Data Lake	5	1	5	5	5	2	5	1	AWS Lakeformation
Lake House	5	3	5	5	4	3	4	4	Databricks

5. Decision Support framework

***** = Highly suitable
**** = Very suitable
*** = Suitable with conditions
** = Limited suitability
* = Generally unsuitable

Use case	Relational DB	Document DB	Graph DB	Data Warehouse	Data Lake	Lakehouse	Streaming
Customer management	*****	****	***	**	**	***	**
Financial transactions	*****	***	**	**	*	**	***
Enterprise OLTP	*****	****	***	*	*	**	**
BI & reporting	****	**	**	*****	****	*****	***
Historical analytics	****	**	**	*****	*****	*****	***
Fraud detection	****	***	*****	****	****	****	*****
Recommendation systems	***	***	*****	***	***	****	****
IoT / telemetry	**	***	**	***	*****	****	*****
Large-scale batch processing	**	**	*	****	*****	*****	***
Real-time analytics	***	***	***	***	***	****	*****
Enterprise knowledge / relationships	***	***	*****	***	****	****	**
Data integration / raw data	***	****	***	***	*****	*****	****
Machine-learning data preparation	***	***	***	****	*****	*****	****

6. Conclusion

The chart below shows the timeline and the data progression, while legacy data systems are still used and are not fully obsolete but the use cases have reduced and their usage has become more specific. As illustrated with the timeline data management has progressed from file systems to Lakehouse architectures. Choosing a data store for a particular use case needs a complex and expert

analysis. This paper has tried to simplify most of these major criteria in using selection matrix and a use case mapper for business and organizations to use as a guiding principle for their new initiatives.

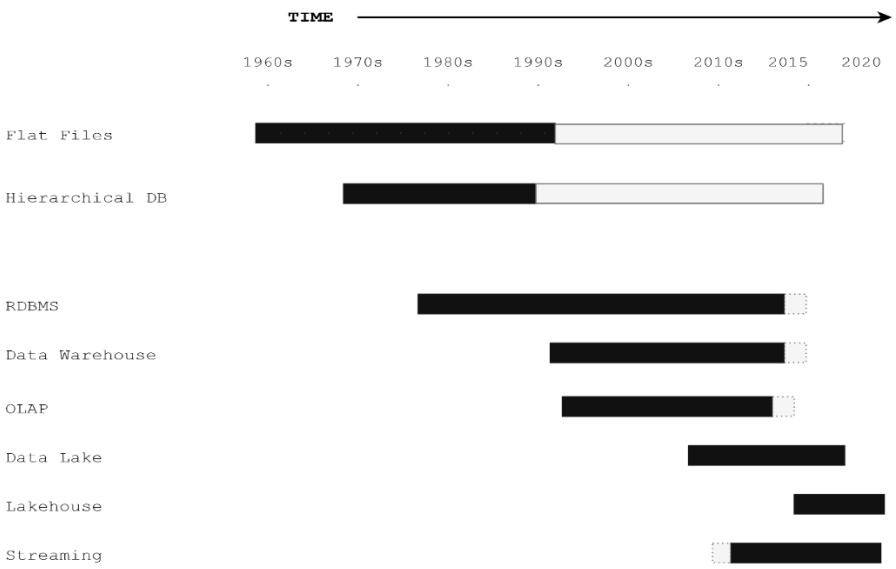


Figure 7. Timeline

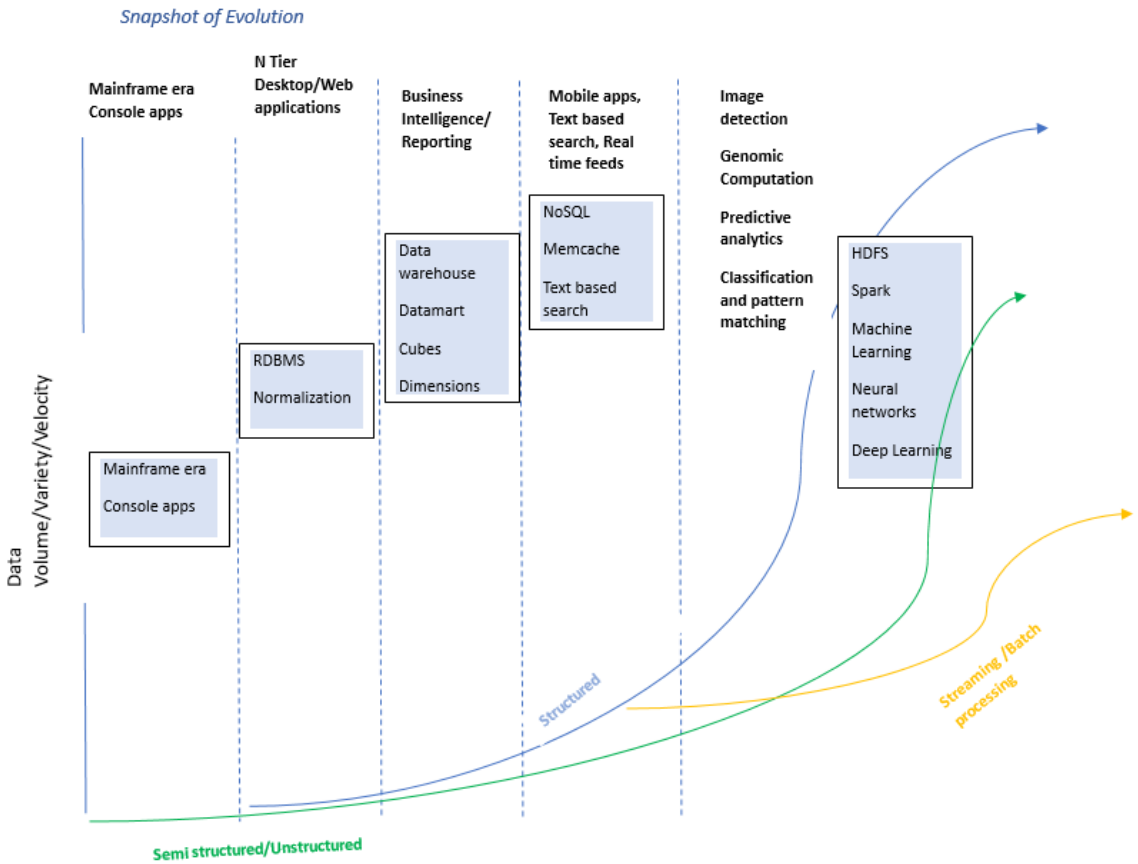


Figure 8. Snapshot of Evolution

References

[1] E. F. Codd, “A Relational Model of Data for Large Shared Data Banks,” Communications of the ACM, 1970.References
[2] Codd’s 12 rules
[3] Ralph Kimball “White Paper: An Architecture for Data Quality “
[4] Kevin D. Foote A Brief History of Non-Relational Databases
[5] Apache Lucene features
[6] "Pentaho, Hadoop, and Data Lakes," *James Dixon's Blog*, 14 October 2010