CSM14303 Seminar on Big Data Management

PAPER SUMMARY:

Multi-model Databases: A New Journey to Handle the Variety of Data

1. Introduction:

Suppose we have data in multiple different structures. Usually, there are two strategies to store and analyse this data:

- Store the data in different DBMSs corresponding to the different data models -> Installing and managing different systems are complex
- Transform all the different models into one single format, e.g. the relational format, and store the data in a DBMS corresponding to that model -> Causes overhead and inefficiency

=> Third option: Build one single unified backend for all the different models.

2. Preliminaries on Data Models:

2.1. Relational Model:

Basically, we store data in tables (relations). Each table contains many rows (tuples), and each row usually has a unique id and other data columns.

=> PostgreSQL, MySQL, SQLite, etc.

2.2. XML/JSON:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<library>
    <book id="1">
        <title>The Great Gatsby</title>
        <author>F. Scott Fitzgerald</author>
        <year>1925</year>
        <genre>Fiction</genre>
    </book>
    <book id="2">
        <title>To Kill a Mockingbird</title>
        <author>Harper Lee</author>
        <year>1960</year>
        <genre>Fiction</genre>
    </book>
</library>
```

```json
{
  "library": [
    {
      "id": "1",
      "title": "The Great Gatsby",
```

```
      "author": "F. Scott Fitzgerald",
      "year": 1925,
      "genre": "Fiction"
    },
    {
      "id": "2",
      "title": "To Kill a Mockingbird",
      "author": "Harper Lee",
      "year": 1960,
      "genre": "Fiction"
    }
  ]
}
```


2.3. Key/Value:

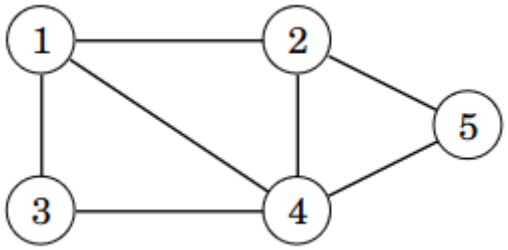

```
127.0.0.1:6379> GET sess:OP3PTcNk_2W-VZGeiOKHcDlDfBYQRM0j
"{\"cookie\":{\"originalMaxAge\":315360000000,\"expires\":\"2033-11-07T21:18:32.429Z\",\"secure\":true,\"httpOnly\":false,\"domain\":\"localhost:3000\",\"path\":\"/\",\"sameSite\":\"lax\"
},\"userId\":1}"
```


=> Redis

2.4. Graph:

A graph is a data structure that consists of nodes and edges. Each edge connects two nodes.

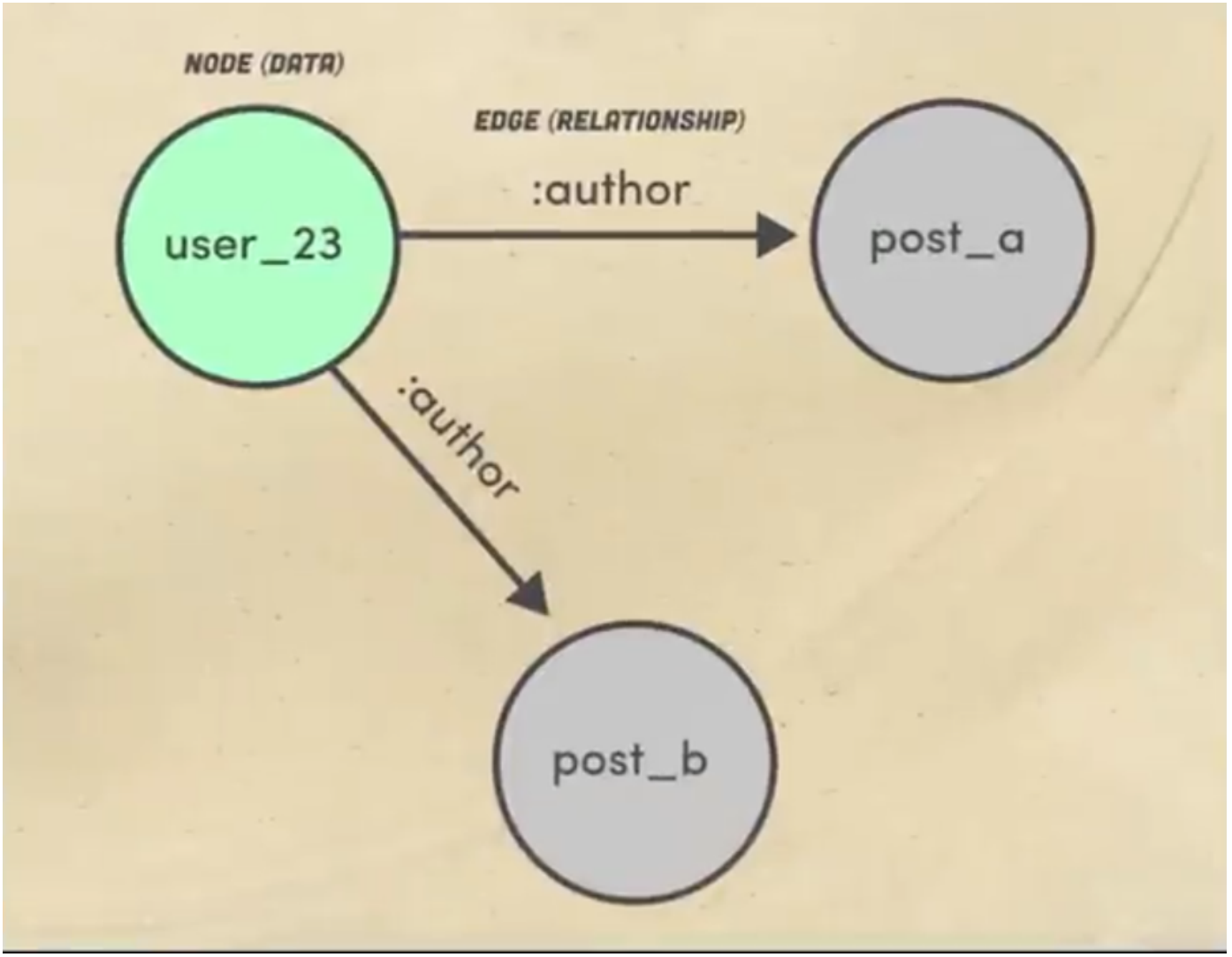
For example, the following graph has five nodes and seven edges:



A neighbor of node is another node connected to it by an edge. In the example, the neighbors of the node 1 are the nodes 2, 3 and 4. The degree of a node is the number of its neighbors. For example, the degree of the node 1 is 3, because it has 3 neighbors.

A path between two nodes is a route along the edges of the graph. Here are some of the possible paths from the node 1 to the node 5:

1 → 2 → 5
1 → 4 → 5
1 → 3 → 4 → 5
1 → 3 → 4 → 2 → 5



=> Neo4j, Dgraph

3. Taxonomy and Comparative Study:

3.1. A Brief History:

- Mid-1960s: Data was stored in File Systems.
- Early 1980s: Relational DBs were increasingly adopted by enterprises
- 1990s: Databases geared towards specific non-relational models of data were developed, e.g. object-oriented databases, XML databases, spatial databases, or RDF databases.
- 2010s: NoSQL DBs like Cassandra, HBase, CouchDB, OrientDB, Neo4j, Asterix, ArangoDB, and MongoDB were created.

=> Trend: NoSQL DBs store an increasing number of data models.

3.2. Taxonomy of multi-model databases:

Multi-model databases can be categorised according to various criteria:

- By the original/core data model:

Table I. Classification of multi-model databases

Original Type	Representatives
Relational	PostgreSQL, SQL Server, IBM DB2, Oracle DB, Oracle MySQL, Sinew
Column	Cassandra, CrateDB, DynamoDB, HPE Vertica
Key/value	Riak, c-treeACE, Oracle NoSQL DB
Document	ArangoDB, Couchbase, MongoDB, Cosmos DB, MarkLogic
Graph	OrientDB
Object	Caché
Other	Not yet multi-model – NuoDB, Redis, Aerospike Multi-use-case – SAP HANA DB, Octopus DB

- By the trategy used to extend the original model to other models or to combine multiple models:

+ Adoption of a completely new storage strategy suitable for the new data model(s)

+ Extension of the original storage strategy for the purpose of the new data model(s)

+ Creating of a new interface for the original storage strategy

+ No change in the original storage strategy

Table II. A strategy for extension towards multiple models

Approach	DBMS	Type
New storage strategy	PostgreSQL	relational
	SQL server	relational
	IBM DB2	relational
	Oracle DB	relational
	Cassandra	column
	CrateDB	column
	DynamoDB	column
	Riak	key/value
	Cosmos DB	document
Extension of the original storage strategy	MySQL	relational
	HPE Vertica	column
	ArangoDB	document
	MongoDB	document
	OrientDB	graph
	Caché	object
New interface for the original storage strategy	Sinew	relational
	c-treeACE	key/value
	Oracle NoSQL Database	key/value
	Couchbase	document
	MarkLogic	document

- By the data models supported in the particular multi-model DBMSs:

Table III. Overview of supported data models in multi-model DBMSs

Type	DBMS	Relational	Column	Key/value	JSON	XML	Graph	RDF	Nested data/UDT/object	Popularity (2018)
Relational	PostgreSQL	√		√	√	√			√	*****
	SQL Server	√			√	√	√		√	*****
	IBM DB2	√				√	√	√	√	*****
	Oracle DB	√			√	√		√	√	*****
	Oracle MySQL	√		√					√	*****
	Sinew	√		√						*
Column	Cassandra		√				√		√	****
	CrateDB	√	√		√		√			*
	DynamoDB		√	√	√		√		√	***
	HPE Vertica		√		√		√			***
Key/value	Riak			√	√	√	√			**
	c-treeACE	√		√			√			*
	Oracle NoSQL DB	√		√			√	√		***
Document	ArangoDB			√	√		√			**
	Couchbase			√	√					****
	MongoDB			√	√				√	*****
	Cosmos DB		√	√	√					***
	MarkLogic				√	√		√	√	*****
Graph	OrientDB			√	√		√			***
Object	InterSystems Caché	√			√	√			√	*

- By their features:

Table VI. Comparison of multi-model single-database DBMSs (yes/no features. In the lower part of the table we include also systems which are not (yet) multi-model.)

Type	DBMS	Data distribution	Flexible schema	Queries across models	Version for cloud	Multi-model transactions
Relational	PostgreSQL	×	√	√	√	×
	SQL Server	√	√	√	√	×
	IBM DB2	√	√	√	√	×
	Oracle DB	√	×	√	√	×
	Oracle MySQL	√	×	√	√	×
	Sinew	–	√	√	×	–
Column	Cassandra	√	×	√	√	×
	CrateDB	√	√	√	√	×
	DynamoDB	√	√	√	√	×
	HPE Vertica	√	√	√	√	×
Key/value	Riak	√	×	√	√	×
	c-treeACE	√	√	–	×	–
	Oracle NoSQL DB	√	×	√	√	×
Document	ArangoDB	√	√	√	√	×
	Couchbase	√	√	√	√	×
	MongoDB	√	√	√	√	×
	Cosmos DB	√	√	–	√	×
	MarkLogic	√	√	√	√	×
Graph	OrientDB	√	√	√	√	×
Object	Caché	√	√	–	√	–
Other	NuoDB	√	–	–	√	–
	Redis	√	–	–	√	–
	Aerospike	√	√	–	√	–

- By their query optimisation strategies:

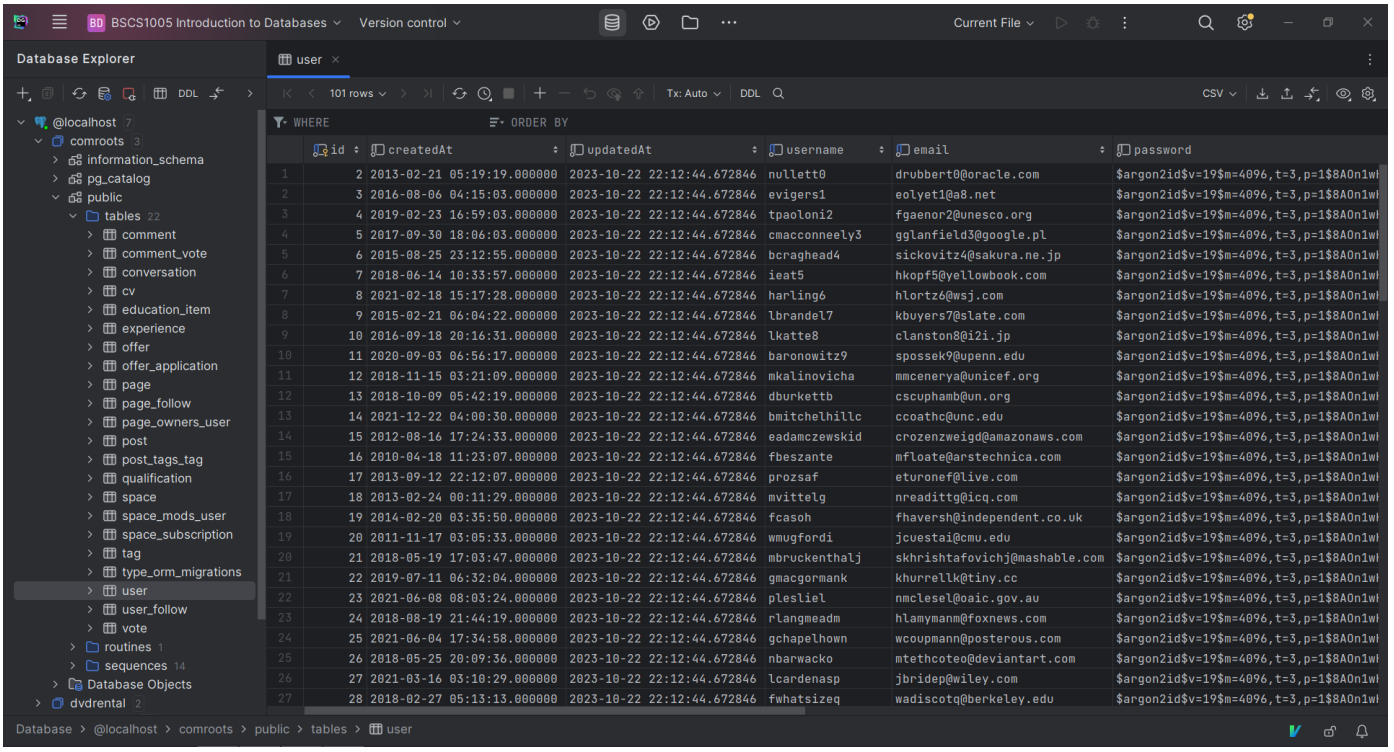
Table VIII. Query optimization strategies in multi-model databases)

Optimization	DBMS	Type
Inverted index	PostgreSQL	relational
	Cosmos DB	document
B-tree, B+-tree	SQL server	relational
	Oracle DB	relational
	Oracle MySQL	relational
	Cassandra	column
	Oracle NoSQL DB	key/value
	Couchbase	document
	MongoDB	document
Materialization	HPE Vertica	column
Hashing	DynamoDB	column
	ArangoDB	document
	MongoDB	document
	Cosmos DB	document
	OrientDB	graph
Bitmap index	Oracle DB	relational
	Caché	object
Function-based index	Oracle DB	relational
Native XML index	Oracle DB	relational
	SQL server	relational
	DB2	relational
	MarkLogic	document

**4. A closer look at Multi-Model Database Representatives:

4.1. Relational Stores (DBMS):

- **PostgreSQL**:



On top of primitives, PostgreSQL has extended support to the following data types, to name a few:

+ XML

+ Key/value (hstore)

+ JSON

```
CREATE TABLE customer (
  id      INTEGER PRIMARY KEY,
  name    VARCHAR(50),
  address VARCHAR(50),
  orders  JSONB
);

INSERT INTO customer
VALUES (1, 'Mary', 'Prague',
 '{"Order_no":"0c6df508",
   "Orderlines":[
    {"Product_no":"2724f", "Product_Name":"Toy",  "Price":66},
    {"Product_no":"3424g", "Product_Name":"Book", "Price":40}]
  }');

INSERT INTO customer
VALUES (2, 'John', 'Helsinki',
 '{"Order_no":"0c6df511",
   "Orderlines":[
    {"Product_no":"2454f", "Product_Name":"Computer", "Price":34}]
  }');
```

id integer	name character varying (50)	address character varying (50)	orders jsonb
1	Mary	Prague	{"Orderlines":[{"Price":66,"Product_Name":"Toy","Product_no":"2724f"},{"Price":40,"Product_Name":...
2	John	Helsinki	{"Orderlines":[{"Price":34,"Product_Name":"Computer","Product_no":"2454f"}],"Order_no":"0c6df511"}

Fig. 4. An example of storing multi-model data in PostgreSQL

- **Microsoft SQL Server**:
+ Started in late 1980s as a relational DBMS
+ Started supporting XML in 2000
+ Started supporting JSON in 2016
+ Since 2016, possible to run SQL queries on external data in Hadoop or Azure blob storage
- **IBM DB2**:
+ First release of IBM DB2 as an object-relational DBMS in early 1980s
+ Started supporting XML in 2007
+ Started supporting RDF graphs in 2012
- **Oracle DB**:
+ Released in 1979 as an object-relational DBMS
+ Started supporting XML in 2001
+ Started supporting JSON and RDF in 2013
- **MySQL**:
+ Released in 1995 as a relational DBMS.
+ In 2014, MySQL cluster was released, enabling data sharding and replication
+ Started supporting key/value data in 2011
- **Sinew**:

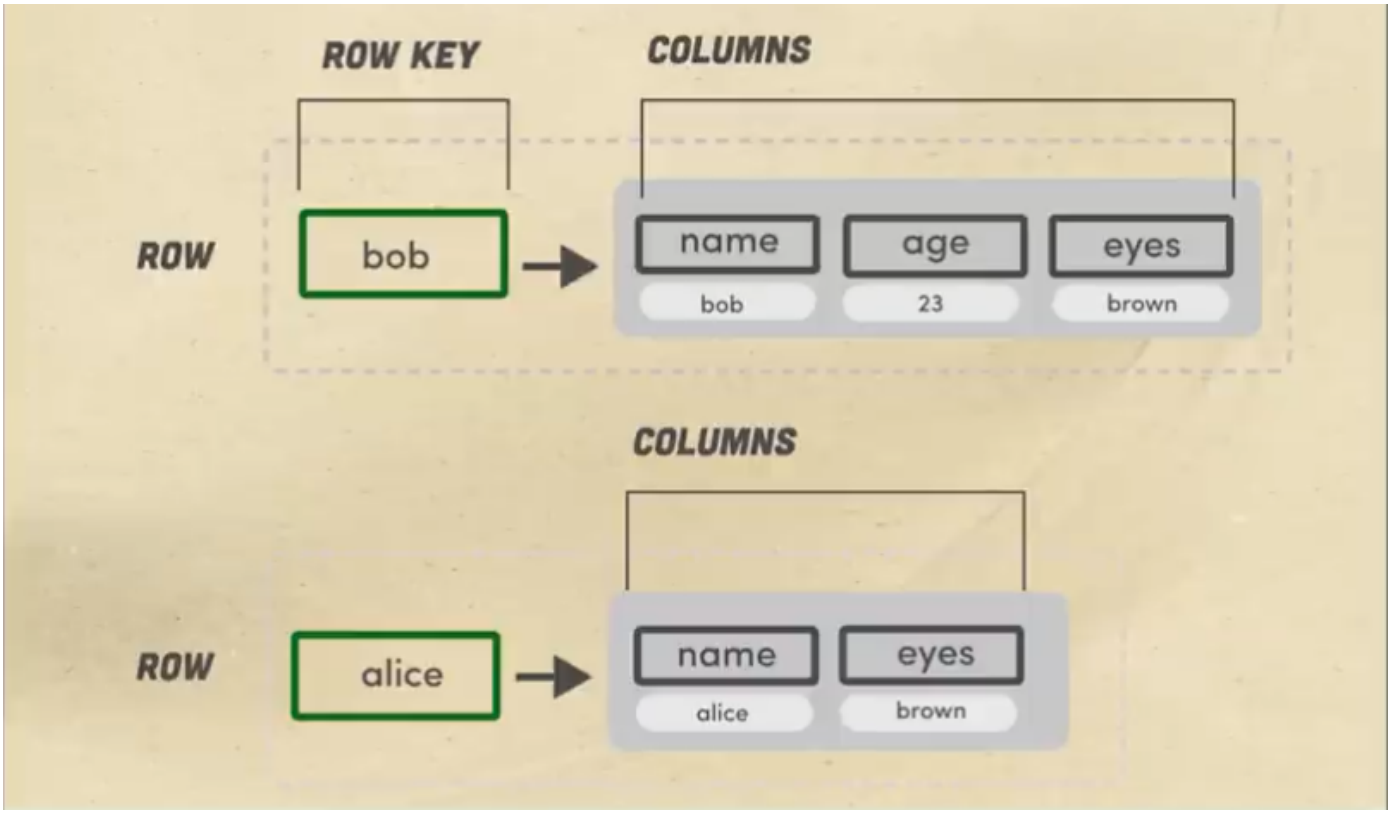
Created a new layer above above a traditional relational DBMS to enable querying multi-model data (key/value, relational, nested document etc.) without a pre-defined schema. Physically, the data is still stored in an underlying relational DBMS.

4.2. Column Stores:

Can be understood as either:

- Column-oriented DBMSs, which store store data tables as columns instead of rows e.g. HPE Vertica.

- Wide-column DBMSs: Our main focus. Similar to key-value, but instead of one value associated being with one key, each key points to multiple columns. Each of those columns, in turn, contain ordered rows. Wide-column DBs are schemaless, and so can handle unstructured data. The trade-off, however, is that we cannot do joins.



- **Apache Cassandra**:

+ Uses a language similar to SQL called CQL :

+ Apart from scalar data types (like text or int), it supports
three types of collections (list, set and map), tuples, and user defined data types (which can consist of any data types), together with respective operations for storing and retrieval of the data.

+ Started supporting JSON in 2015:

```
create keyspace myspace
WITH REPLICATION = { 'class' : 'SimpleStrategy',
                     'replication_factor' : 3 };

CREATE TYPE myspace.orderline (
   product_no text,
   product_name text,
   price float
   );

CREATE TYPE myspace.myorder (
   order_no text,
   orderlines list<frozen <orderline>>
   );

CREATE TABLE myspace.customer (
   id INT PRIMARY KEY,
   name text,
   address text,
   orders list<frozen <myorder>>
   );
```

```
INSERT INTO myspace.customer JSON
' {"id":1,
  "name":"Mary",
  "address":"Prague",
  "orders" : [
  { "order_no":"0c6df508",
    "orderlines":[
    { "product_no" : "2724f",
      "product_name" : "Toy",
      "price" : 66 },
      { "product_no" : "3424g",
        "product_name" :"Book",
        "price" : 40 } ] } ]
}';

INSERT INTO myspace.customer JSON
' {"id":2,
  "name":"John",
  "address":"Helsinki",
  "orders" : [
  { "order_no":"0c6df511",
    "orderlines":[
    { "product_no" : "2454f",
      "product_name" : "Computer",
      "price" : 34 } ] } ]
}';
```

Fig. 6. An example of storing multi-model data in Cassandra

- **CrateDB**:

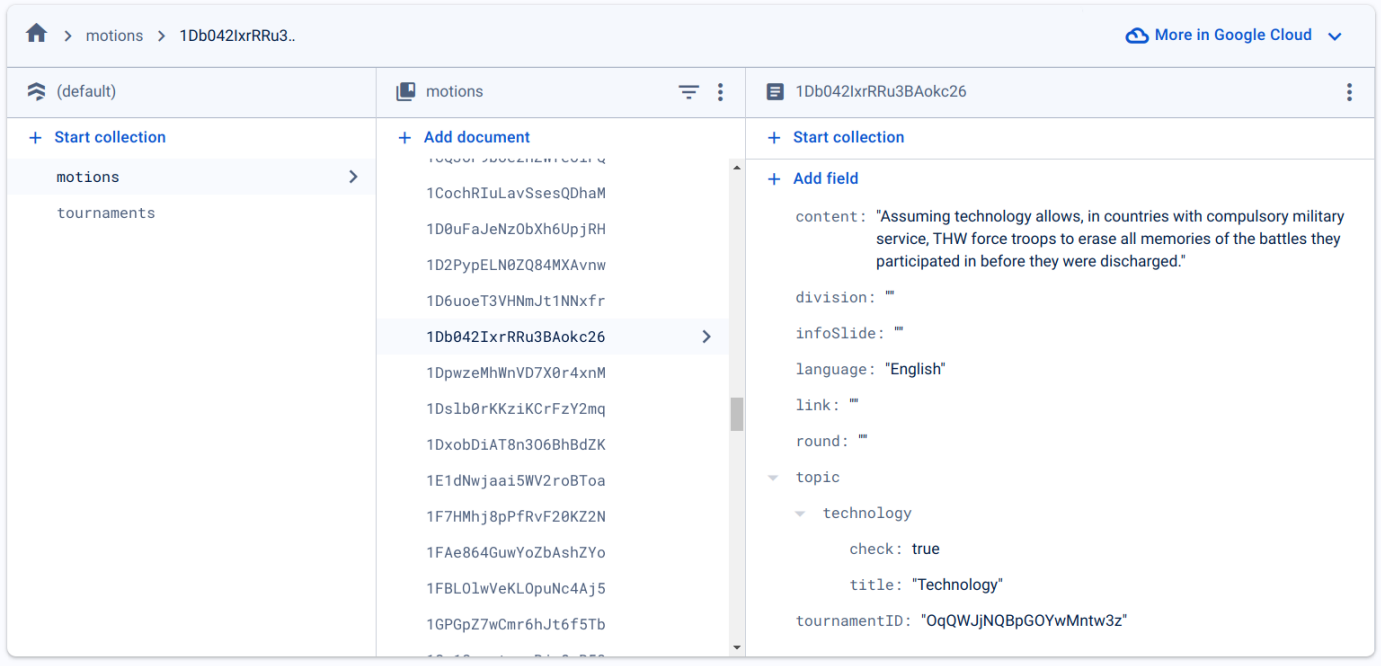
+ Released in 2016 as a distributed column-oriented SQL DB with dynamic schema which can store also nested JSON documents, arrays, and BLOBs

+ Each row of a table in CrateDB is a semi-structured document. Every table in CrateDB is sharded across the nodes of a cluster, whereas each shard is a Lucene index. Operations on documents are atomic.

- **AWS DynamoDB**:

+ Released in 2012 as a cloud database which supports both (JSON) documents and key/value flexible data models

+ In DynamoDB, a table is schemaless and it corresponds to a collection of items. An item is a collection of attributes and it is identified by a primary key. An attribute consists of a name, a data type, and a value. The data type can be a scalar value (string, number, Boolean etc.), a document (list or map), or a set of scalar values. The data items in a table do not have to have the same attributes.



+ DynamoDB stores data in partitions. A partition is an allocation of storage for a table, backed by solid state drives (SSDs) and automatically replicated across multiple Availability Zones within an AWS Region.
+ If your table has a simple primary key (partition key only), DynamoDB stores and retrieves each item based on its partition key value. To write an item to the table, DynamoDB uses the value of the partition key as input to an internal hash function. The output value from the hash function determines the partition in which the item will be stored.
+ If the table has a composite primary key (partition key and sort key), DynamoDB calculates the hash value of the partition key in the same way as described in Data distribution: Partition key. However, it tends to keep items which have the same value of partition key close together and in sorted order by the sort key attribute's value. The set of items which have the same value of partition key is called an item collection.
- **HPE Vertica**:
+ A high-performance analytics engine designed to manage Big Data
+ Column-oriented storage organisation
+ Has support for a standard SQL interface enriched with analytics capabilities
+ In 2013: Extended with flex tables, which do not require schema definitions, can store semi-structured data (e.g. JSON, CSV, etc.) and be queried using SQL. Creating flex tables is similar to creating classical tables, except column definitions are optional.
+ Besides the flex table itself, Vertica creates also associated keys table (with self-describing columns key_name, frequency, and data_type_guess) and a default view for the main flex table.

4.3. Key/Value Stores:

- **Riak**:
+ Released in 2009 as a classic key/value DBMS
+ Can be used as a document store with querying capabilities with Riak Search and Riak Data Types
+ Currently supports JSON, XML, plain text, and Riak Data Types extractors, but it is possible to implement an own extractor as well.
- **c-treeACE**:
+ Offers both NoSQL and SQL in a single database
+ Supports both relational and non-relational APIs
- **Oracle NoSQL DB**:
+ A key/value DBMS which supports SQL
+ Provides RDF support using Oracle Graph module
+ Can store both relational and JSON data:

```
create table Customers (
  id integer,
  name string,
  address string,
  orders array (
    record (
      order_no string,
      orderlines array (
        record (
          product_no string,
          product_name string,
          price integer ) ) )
    ),
  primary key (id)
);

import -table Customers -file customer.json
```

Content of file **customer.json**:

```
{  "id":1,
   "name":"Mary",
   "address":"Prague",
   "orders" : [
     { "order_no":"0c6df508",
       "orderlines":[
         { "product_no" : "2724f",
           "product_name" : "Toy",
           "price" : 66 },
         { "product_no" : "3424g",
           "product_name"  :"Book",
           "price" : 40 } ] } ]
}
{  "id":2,
   "name":"John",
   "address":"Helsinki",
   "orders" : [
     {"order_no":"0c6df511",
      "orderlines":[
       { "product_no" : "2454f",
         "product_name" : "Computer",
         "price" : 34  } ] } ]
}
```

Fig. 7. An example of storing multi-model data in Oracle NoSQL Database

```
sql-> select * from Customers;
+----+------+----------+-----------------------------+
| id | name | address  |           orders            |
+----+------+----------+-----------------------------+
|  2 | John | Helsinki | order_no          | 0c6df511 |
|    |      |          | orderlines                  |
|    |      |          |     product_no    | 2454f    |
|    |      |          |     product_name  | Computer |
|    |      |          |     price         | 34       |
+----+------+----------+-----------------------------+
|  1 | Mary | Prague   | order_no          | 0c6df508 |
|    |      |          | orderlines                  |
|    |      |          |     product_no    | 2724f    |
|    |      |          |     product_name  | Toy      |
|    |      |          |     price         | 66       |
|    |      |          |                             |
|    |      |          |     product_no    | 3424g    |
|    |      |          |     product_name  | Book     |
|    |      |          |     price         | 40       |
+----+------+----------+-----------------------------+
```

Fig. 8. An example of storing multi-model data in Oracle NoSQL Database – the resulting table

4.4. Document Stores:

- **ArangoDB**:

+ Created as a multi-model system

+ Supports key/value, document, and graph data

+ Primarily serves documents to clients. Documents are represented in the JSON format and grouped in collections.

+ A document collection always has a primary key attribute key and in the absence of further secondary indices the document collection behaves like a simple key/value store. Special edge collections store documents as well, but they include two special attributes, from and to, which enable to create relations between documents. Hence two documents (vertices) stored in document collections are linked by a document (edge) stored in an edge collection. This is ArangoDB's graph data model.

+ ArangoDB query language (AQL) allows complex queries. Despite the different data models, it is similar to SQL. In case of the key/value store the only operations that are possible are single key lookups and key/value pair insertions and updates. In case of the document store queries can range from a simple "query by example" to complex "joins" using many collections, usage of functions

(including user-defined ones) etc. For the purpose of graph data various types of traversing graph structures and shortest path searches are available.

- **Couchbase**:

+ Is both key/value and document DBMS with an SQL-based

+ Documents (in JSON) are stored in data containers called buckets without any pre-defined schema.

- **MongoDB**:

+ Stores data in JSON documents, which support graph data in addition to simple key/value pairs and table-like structures

+ Documents have a flexible schema and so their collections do not enforce document structure (except for field iduniquely identifying each document). Operations are atomic at the document level.

- **Azure CosmosDB**:

+ A cloud, schema-less, originally document database which supports ACID compliant transactions

+ Supports document (JSON), key/value, graph, and columnar data models

+ Uses a SQL-like query language for the above data models

4.4.1 XML Stores:

- **MarkLogic**:

+ Originally a native XML DB, it has been extended to support JSON, RDF, binary, and textual.

+ Models JSON documents as XML documents, i.e., trees, which allows for querying using XPath:

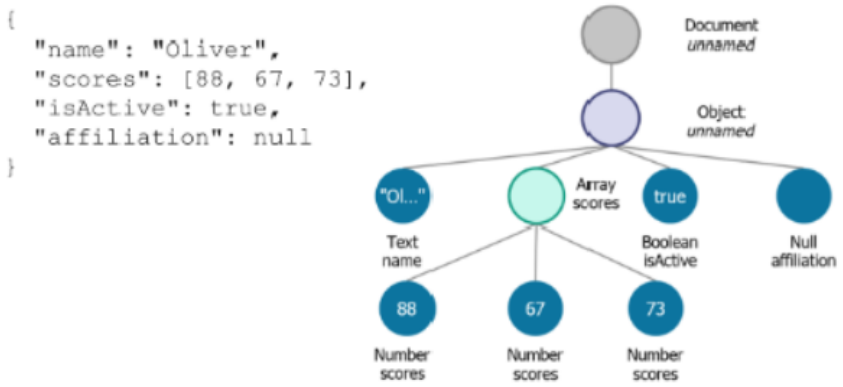


Fig. 10. An example of modeling JSON data as trees in MarkLogic (source: https://developer.marklogic.com/features/json)

4.5. Graph Stores:

- **OrientDB**:

+ An open source NoSQL DBMS supporting graph,key/value, document, and object models

+ The most smallest logical unit of storage in Orient DB is a record. Each record has a unique ID and correspondings to a document, a BLOB, a vertex, or an edge.

+ Classes contain and define records. Classs can be schema-full, schema-less, or schema-mixed. Classes

can have two types of relationships:

* Referenced relationships: stored as physical links: the target record ID is stored in the source record

* Embedded relationships: embedded records stored wtihin a container record and only accessible via that container record

```
CREATE CLASS orderline EXTENDS V
CREATE PROPERTY orderline.product_no STRING
CREATE PROPERTY orderline.product_name STRING
CREATE PROPERTY orderline.price FLOAT

CREATE CLASS order EXTENDS V
CREATE PROPERTY order.order_no STRING
CREATE PROPERTY order.orderlines EMBEDDEDLIST orderline

CREATE CLASS customer EXTENDS V
CREATE PROPERTY customer.id INTEGER
CREATE PROPERTY customer.name STRING
CREATE PROPERTY customer.address STRING

CREATE CLASS orders EXTENDS E

CREATE CLASS knows EXTENDS E
```


```
CREATE VERTEX order CONTENT {
   "order_no":"0c6df508",
   "orderlines":[
    { "@type":"d",
      "@class":"orderline",
      "product_no":"2724f",
      "product_name":"Toy",
      "price":66 },
    { "@type":"d",
      "@class":"orderline",
      "product_no":"3424g",
      "product_name":"Book",
      "price":40}]
}
```

```
CREATE VERTEX order CONTENT {
   "order_no":"0c6df511",
   "orderlines":[
    { "@type":"d",
      "@class":"orderline",
      "product_no":"2454f",
      "product_name":"Computer",
      "price":34 }]
}
```

```
CREATE VERTEX customer CONTENT {
  "id" : 1,
  "name" : "Mary",
  "address" : "Prague"
  }

CREATE VERTEX customer CONTENT {
  "id" : 2,
  "name" : "John",
  "address" : "Helsinki"
  }
```

```
CREATE EDGE orders FROM
   (SELECT FROM customer WHERE name = "Mary")
   TO
   (SELECT FROM order WHERE order_no = "0c6df508")

CREATE EDGE orders FROM
   (SELECT FROM customer WHERE name = "John")
   TO
   (SELECT FROM order WHERE order_no = "0c6df511")

CREATE EDGE knows FROM
   (SELECT FROM customer WHERE name = "Mary")
   TO
   (SELECT FROM customer WHERE name = "John")
```

Fig. 13. An example of storing multi-model data in OrientDB

+ Supports data querying using either the graph-traversal language Gremlin or a version of SQL extended for graph traversal

4.6. Other Stores:

4.6.1. Object Stores:

- **InterSystems Caché**:

+ An object database which stores data in sparse, multidimensional arrays capable of carrying hierarchically structured data. Since 2016, it also supports JSON/XML documents.

+ Data can be accessed using several APIs: via objectsbased upon the ODMG standard (involving inheritance and polymorphism, embed-ded objects, collections etc.), SQL (including DDL, transactions, referential integrity,triggers, stored procedures etc. with various object enhancements), or direct (andhighest-performance) manipulation of its multidimensional data structures.

4.6.2.Multi-Use-Case Stores:

- **SAP HANA DB**: an in-memory, column-oriented, relational DBMS.It exploits and combines the advantages of a row (OLTP) and columnar (OLAP) storagestrategy together with in-memory processing in order to provide a highly efficient anduniversal data management tool.

- **OctopusDB**: tries to mimic OLTP, OLAP, streamingand other types of database systems. For this purpose it does not have any fixed hardcoded (e.g., row or columnar) store, but it records all database operations to a sequen-tial primary log by creating appropriate logical log records. It later creates arbitraryphysical representations of the log (called storage views), depending on the workload.

4.6.3. Not (Yet) Multi-Model:

- **NuoDB**: a NewSQL DBMS which works in the cloud.
- **Redis**: a NoSQL key/value DBMS. The value can be a string, a list of strings, an (un)ordered set of strings, a hash table, etc.
- **Aerospike**: a key/value store with support for maps and lists for the value

4.6.4. No Longer Available: FoundationDB, Akiban Server, etc.

5. Challenges and Open Problems:

- **Multi-model Query Processing and Optimisation**:

+ A truly multi-model DBMS requires one unified cross-model data processing language that can be compiled and optimised for different models. Although these languages exist, they currently are immature.
+ Current query optimisation strategies are mostly tailored for RDBMS with a fixed relational schema, while a multi-model DBMS would have to support flexible and diverse schemas.
+ Cloud deployments for DBs are becoming ever more popular, so multi-model DBMSs need to utilise distributed database management and parallel database programming to fulfill scalability, simplicity, and flexibility requirements.

- **Multi-model Schema Design and Optimisation**: In the world of multi-model systems we encounter contradictory requirements for the distinct models and thus it calls for a new solution for multi-model schema design to balance and trade-off the diverse requirements of multi-model data.
- **Multi-model Evolution**:

As the volume of data increases, data schemas have to evolve and thus causing changes to data instances, queries, indices, or even storage strategies, etc. This is even more true for multi-model DBMSs. There are generally 2 kinds of schema modifications:

+ **Intra-model changes**: For these, we can apply existing strategies for single-models.
+ **Inter-model changes**: In the case of multi-model databases the distinct models cover separate parts of the reality which are interconnected using references, foreign keys, or similar entities. Hence the evolution management has to be solved across all the supported data models. In addition, the challenge of query rewrite, i.e. propagation of changes to queries, also becomes more complex in case of inter-model changes which require changes in data access constructs.

- **Multi-model Extensbility**:

+ **Intra-model extensbility**: extending one of the models with new constructs, e.g., extending the XML model with the support for the query on IDs and IDREF(S)
+ **Inter-model extensbility**: adding new constructs expressing relations between the models, e.g. the ability to express a CHECK constraint from the relational model across both relational and XML data
+ **Extra-model extensbility**: adding a whole new model, together with respective data and query, e.g. adding time series data with the support of time series analysis

6. Conclusion:

The specific V-characteristics of Big Data bring many challenging tasks to be solved to provide efficient and effective management of the data. In this survey we focus on the variety challenge of Big Data

which requires concurrent storage and management of distinct data types and formats. Multi-model DBMSs analyzed in this survey correspond to the “one size fits a bunch” viewpoint. Considering the Gartner survey which shows the high near-future representation and the existing large amount of multi-model systems, this approach has demonstrated its meaningfulness and practical applicability. On the other hand, this survey also shows that there still remains a long journey towards a mature and robust multi-model DBMS comparable with verified solutions from the world of relational databases. One intention of this survey is to promote research and industrial efforts to catch the opportunities and address challenges in developing a full-fledged multi-model database system.