Modern Database Systems

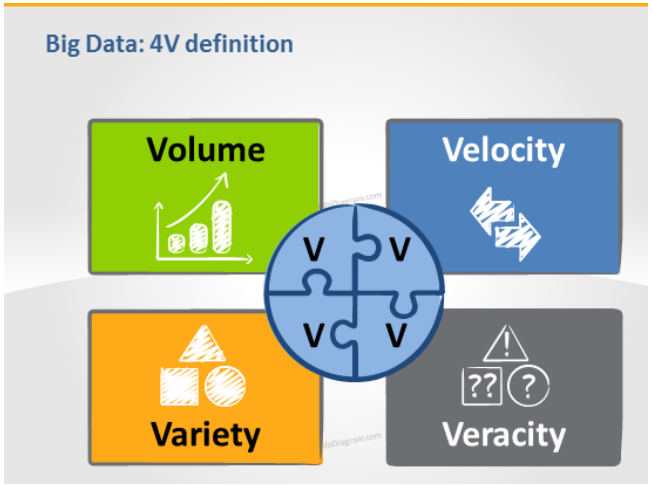
Multi-model databases

Doc. RNDr. Irena Holubova, Ph.D.
Irena.Holubova@matfyz.cuni.cz

Based on the tutorial “Multi-model Data Management: What's New and What's Next?”, Jiaheng Lu and Irena Holubova, EDBT’17, Venice, Italy.

Big Data V-Characteristics

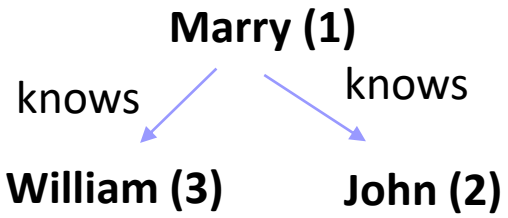
- Volume – scale
- Variety – complexity
- Velocity – speed
- …
- Veracity – uncertainty
- …
- Value
- Validity
- Volatility
- …



A Grand Challenge on Variety

- Tree data (XML, JSON)
- Graph data (RDF, property graphs, networks)
- Tabular data (CSV)
- Temporal and spatial data
- Text
- …

An example of multi-model data



Social network graph

```
{ "Order_no":"0c6df508",
  "Orderlines": [
    { "Product_no":"2724f"
      "Product_Name":"Toy",
      "Price":66 },
    { "Product_no":"3424g",
      "Product_Name":"Book",
      "Price":40 } ]
}
```

Order JSON document

"1" -- > "34e5e759"

"2"-- > "0c6df508"

Key/value pairs
(Customer_ID , Order_no)

Customer relation

Customer_ID	Name	Credit_limits
1	Mary	5,000
2	John	3,000
3	William	2,000

NoSQL Database Types

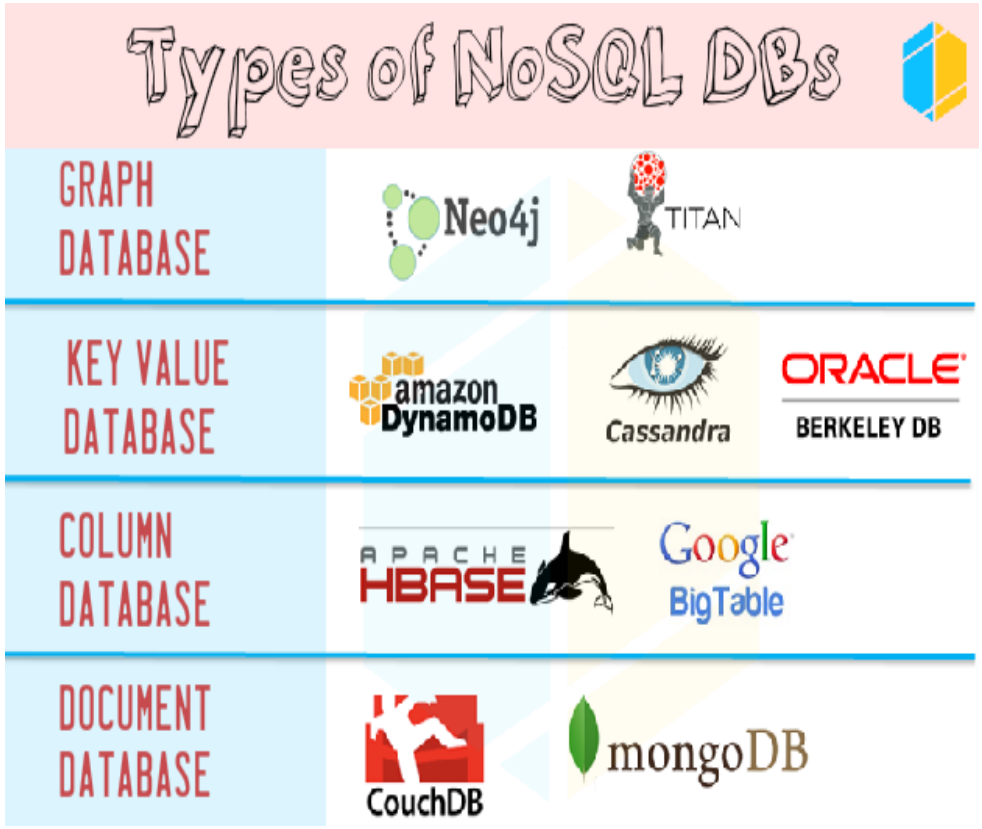
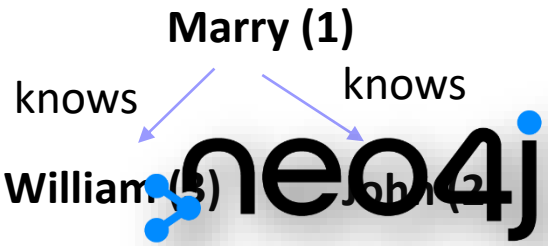


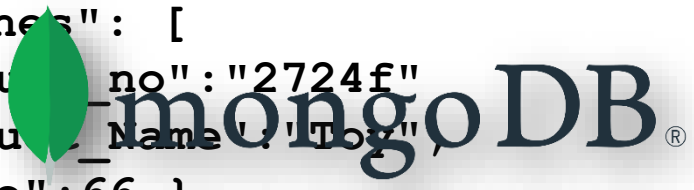
Photo downloaded from: http://www.vikramtakkar.com/2015/12/nosql-types-of-nosql-database-part-2.html

An example of multi-model data



Social network graph

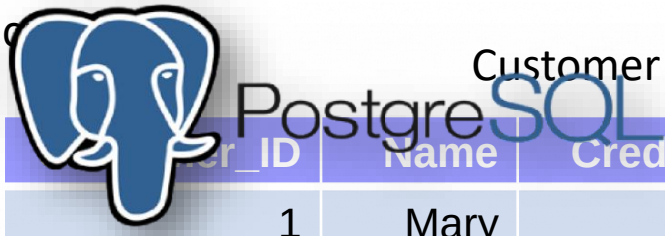
```
{ "Order_no":"0c6df508",
  "Orderlines": [
    { "Product_no":"2724f"
      "Product_Name":"Toy",
      "Price":66 },
    { "Product_no":"3424g",
      "Product_Name":"Book",
      "Price":40 } ]
}
```



Order JSON doc

Key/value pairs
(Customer_ID , Order_no)

Customer relation



Customer_ID	Name	Credit_limits
1	Mary	5,000
2	John	3,000
3	William	2,000

Polyglot Persistence

- Idea: Use the right tool for the job
- If you have structured data with some differences
  - Use a document store
- If you have relations between entities and want to efficiently query them
  - Use a graph database
- If you manage the data structure yourself and do not need complex queries
  - Use a key/value store

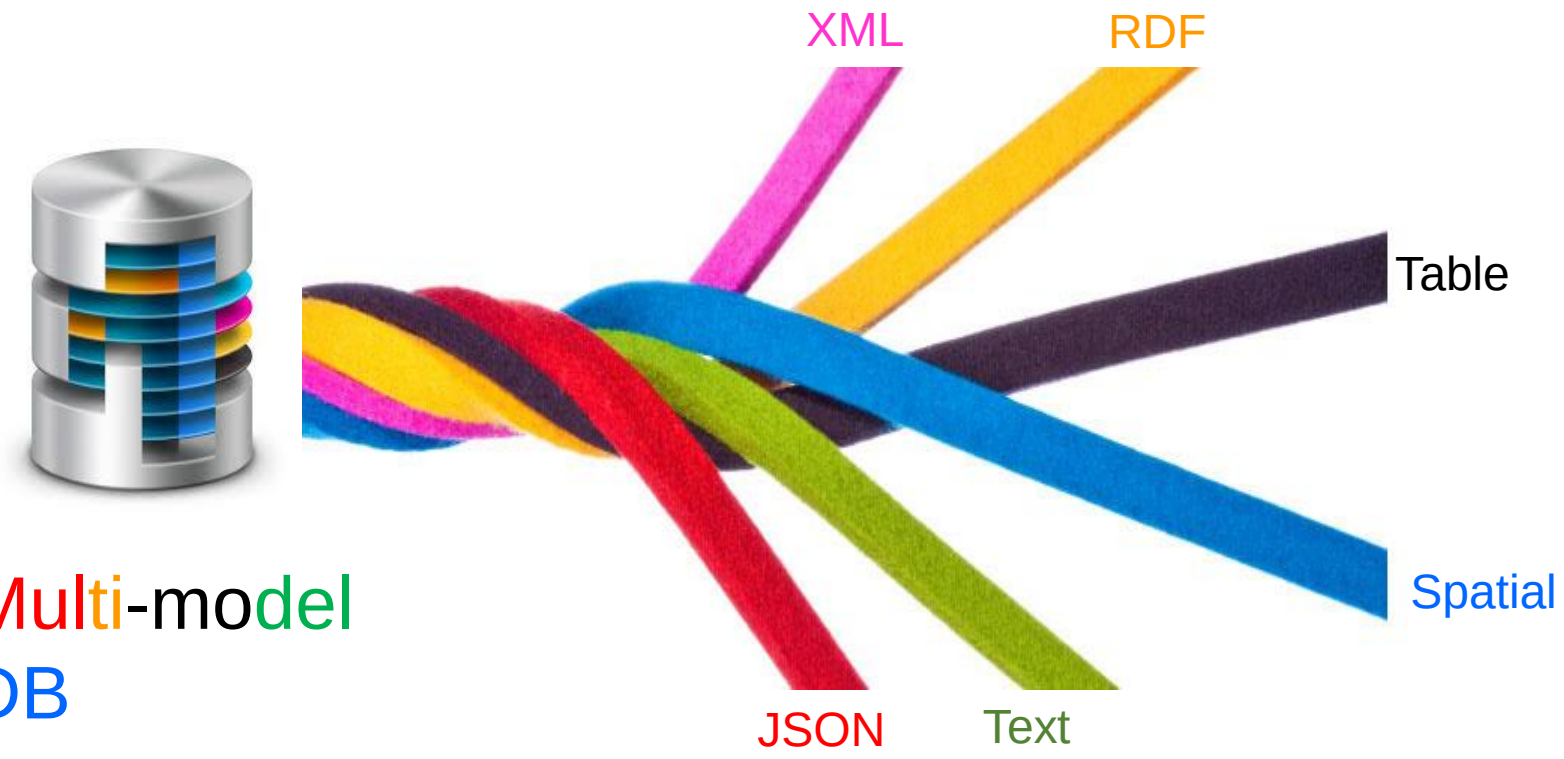
Pros and Cons of Polyglot Persistence

- Handles multi-model data
- Helps apps to scale well
- A rich experience

- Requires the company to hire people to integrate different databases
- Developers need to learn different databases
- How to handle cross-model queries and transactions?

Multi-model DB

- One unified database for multi-model data



Multi-model Databases

- A multi-model database is designed to support multiple data models against a single, integrated backend
- Example of data models: document, graph, relational, key/value

Three Arguments

1. One size **cannot** fit **all**
2. One size **can** fit all
3. One size **fits a bunch**

One size cannot fit all

"SQL analytics, real-time decision support, and data warehouses cannot be supported in one engine."

M. Stonebraker and **U. Cetintemel**. "One Size Fits All": An Idea Whose Time Has Come and Gone (Abstract). In ICDE, 2005.

One size can fit all

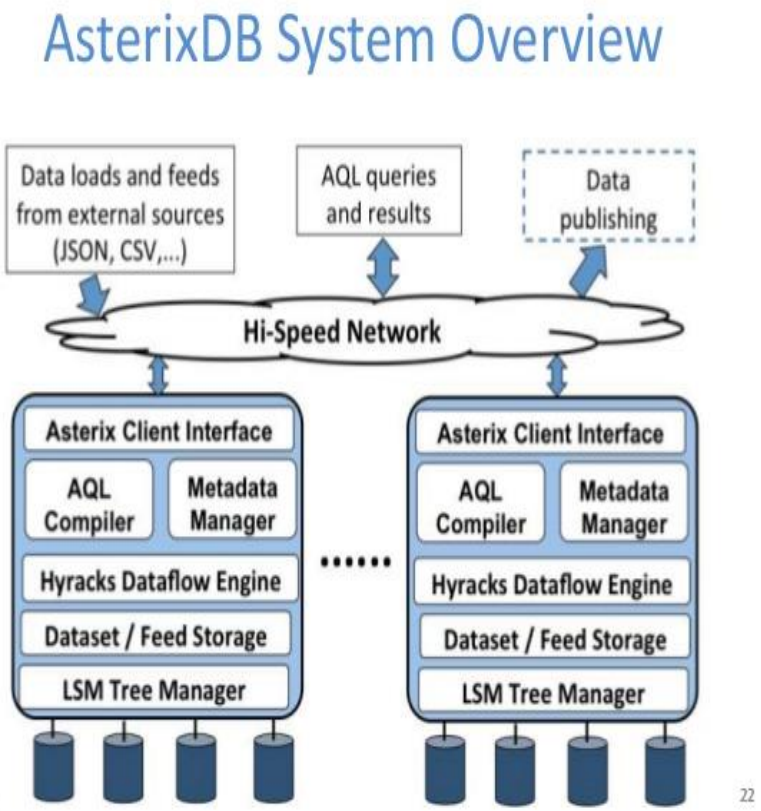
- OctopusDB suggests a unified, one-size-fits-all data processing architecture for OLTP, OLAP, streaming systems, and scan-oriented database systems

J. Dittrich, A. Jindal: Towards a One Size Fits All Database Architecture. CIDR 2011: 195-198

One size can fit all

- All data is collected in a central log
  - Insert and update-operations = log-entries
- Based on that log, it defines several types of optional storage views
- Query optimization, view maintenance, index selection, as well as the store selection problems suddenly become a single problem: storage view selection

One size can fit a bunch



- Semistructured data model
  - Both schema-less and schema-full
  - Textural, temporal, spatial,… data
- SQL-like query language

AsterixDB: A Scalable, Open Source BDMS. PVLDB 7(14): 1905-1916 (2014)

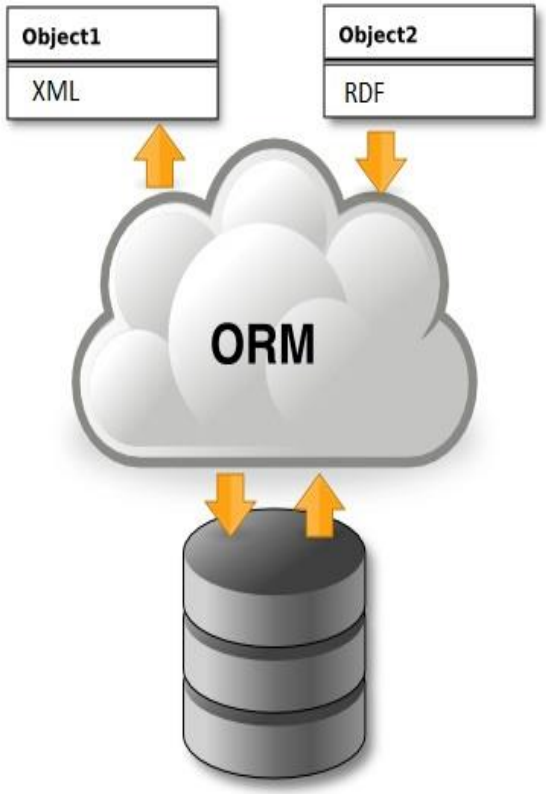
Multi-model Databases:
One size fits multi-model data

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

https://dbdb.io/

Multi-model databases are not new!

- Can be traced to object-relational databases (ORDBMS)
- ORDBMS framework allows users to plug in their domain and/or application specific data models as user-defined functions/types/indexes



Most of DBs will become multi-model databases in 2017

- By 2017, all leading operational DBMSs will offer multiple data models, relational and NoSQL, in a single DBMS platform.

-- Gartner report for operational databases 2016

e.g. MongoDB supports multi-model in the recent release 3.4 (**NOV 29, 2016**)

Pros and Cons of Multi-model databases

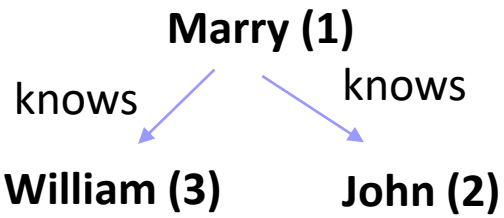
- Handle multi-model data
- One system implements fault tolerance
- Data consistency
- Unified query language for multi-model data

- A complex system
- Immature and developing
- Many challenges and open problems

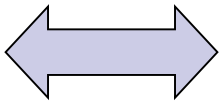
Two examples of multi-model databases

- ArangoDB is a multi-model, open-source database with flexible data models
  - **Documents**, graphs, key/values
- Stores all data as documents
- Vertices and edges of graphs are documents → allows to mix all three data models

An example of multi-model data and query

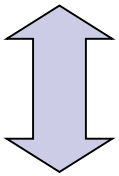


Graph-key/value join



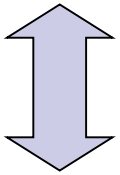
"1" --> "34e5e759"

"2"--> "0c6df508"



Relation-graph join

Key/value-JSON join



Customer_ID	Name	Credit_limits
1	Mary	5,000
2	John	3,000
3	William	2,000

```
{ "Order_no":"0c6df508",
  "Orderlines": [
    { "Product_no":"2724f"
      "Product_Name":"Toy",
      "Price":66 },
    { "Product_no":"3424g",
      "Product_Name":"Book",
      "Price":40 } ] }
```

Recommendation query:

Return all product_no-s which are ordered by a friend of a customer whose credit_limit>3000

An example of multi-model data and query

```
LET CustomerIDs = (
   FOR Customer IN Customers
   FILTER Customer.CreditLimit > 3000
   RETURN Customer.id)
LET FriendIDs = (
   FOR CustomerID IN CustomerIDs
       FOR Friend IN 1..1 OUTBOUND CustomerID Knows
   RETURN Friend.id)
FOR Friend in FriendIDs
FOR Order in 1..1 OUTBOUND Friend Customer2Order
RETURN Order.orderlines[*].Product_no
```

Recommendation query:
Return all product_no-s which are ordered by a friend of a customer whose credit_limit>3000

ArangoDB

- Supporting graph, document, key/value and **object** models
- The relationships are managed as in graph databases with direct connections between records
- It supports schema-less, schema-full and schema-mixed modes
- Queries: SQL extended for graph traversal


```
SELECT expand( out("Knows").Orders.orderlines.
       Product_no )
FROM Customers
WHERE CreditLimit > 3000
```

Recommendation query:
Return all product_no-s which are ordered by a friend of a customer whose credit_limit>3000

Classification of Multi-model Systems

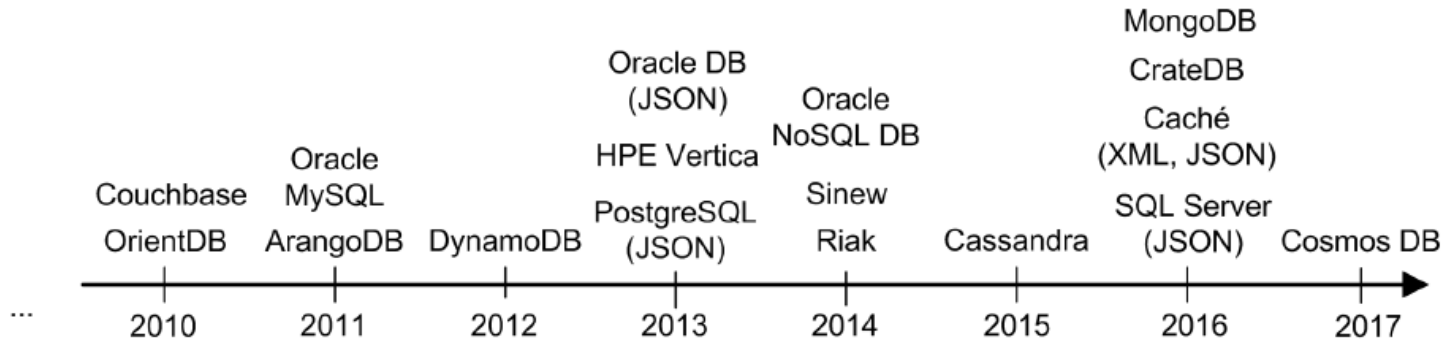
- Basic approach: on the basis of original (or core) data model

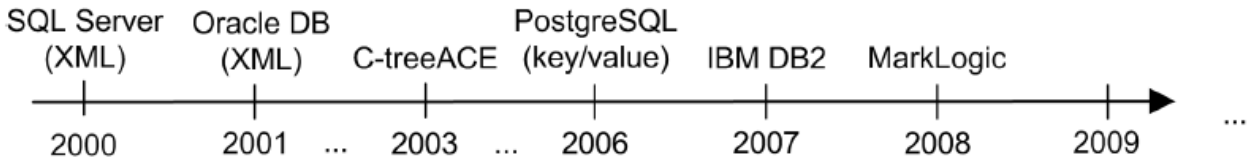
Popular = considered models

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, MarkLogic, MongoDB, Cosmos DB
Graph	OrientDB
Object	InterSystems Caché
Special	• Not yet multi-model – NuoDB, Redis, Aerospike • Multi-use-case – SAP HANA DB, Octopus DB

Timeline

- When a particular system became multi-model
  - Original data format (model) was extended
  - First released directly as a multi-model DBMS





Extension towards Multiple Models

Types of strategies:

1. Adoption of a completely new storage strategy suitable for the new data model(s)
   - e.g., XML-enabled databases
2. Extension of the original storage strategy for the purpose of the new data model(s)
   - e.g., ArangoDB - special edge collections bear information about edges in a graph
3. Creating of a new interface for the original storage strategy
   - e.g., MarkLogic - stores JSON data in the same way as XML data
4. No change in the original storage strategy
   - Storage and processing of data formats simpler than the original one

- Types of cross-model transitions
  - Inter-model references
  - Model embedding
  - Cross-model redundancy

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

Overview of Supported Data Models

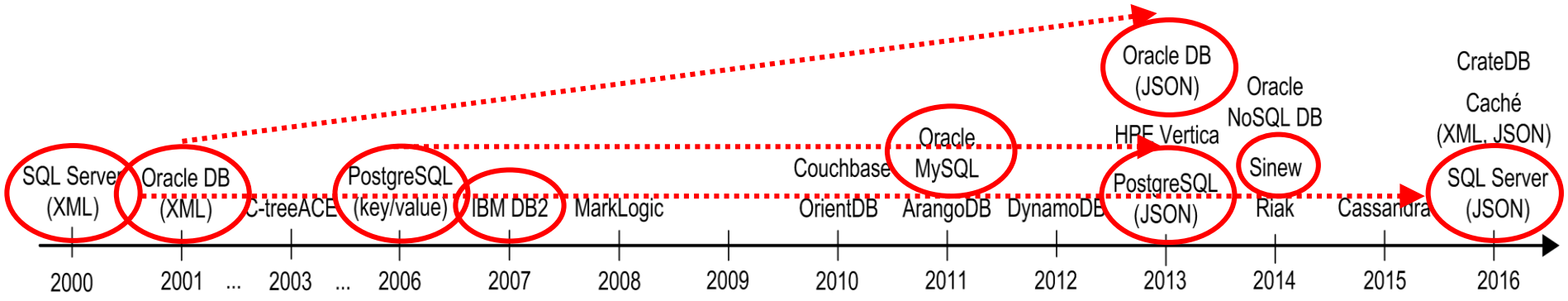
??

Type	DBMS	Relational	Column	Key/value	Document (JSON)	XML	Graph	Nested data/UDT/object
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	√			√	√		√

Relational Multi-model DBMSs

- Biggest set of multi-model databases
  - The most popular type of databases
  - SQL has been extended towards other data formats (e.g, SQL/XML)
  - Simplicity and universality of the relational model

Type	DBMS	Relational	Column	Key/value	Document (JSON)	XML	Graph	Nested data/UDT/object
Relational	PostgreSQL	√		√	√	√		
	SQL Server	√			√	√		
	IBM DB2	√				√		
	Oracle DB	√			√	√		
	Oracle MySQL	√		√				
	Sinew	√		√				



Relational Multi-model DBMSs

Storage – PostgreSQL Example

```
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"}

Relational Multi-model DBMSs

Storage – PostgreSQL Example

```
SELECT json_build_object('id',id,'name',name,'orders',orders)
FROM customer;
```

json_build_object json
{"orders":{"Orderlines":[{"Price":66,"Product_Name":"Toy","Product_no":"2724f"},{"Price":40,"Product_Name":"Book","Product_no":"3...
{"orders":{"Orderlines":[{"Price":34,"Product_Name":"Computer","Product_no":"2454f"}],"Order_no":"0c6df511"},"id":2,"name":"John"}

```
SELECT jsonb_each(orders) FROM customer;
```

jsonb_each record
(Order_no,"""0c6df508""")
(Orderlines,"[{""Price"": 66, ""Product_no"": ""2724f"", ""Product_Name"": ""To...
(Order_no,"""0c6df511""")
(Orderlines,"[{""Price"": 34, ""Product_no"": ""2454f"", ""Product_Name"": ""Co...

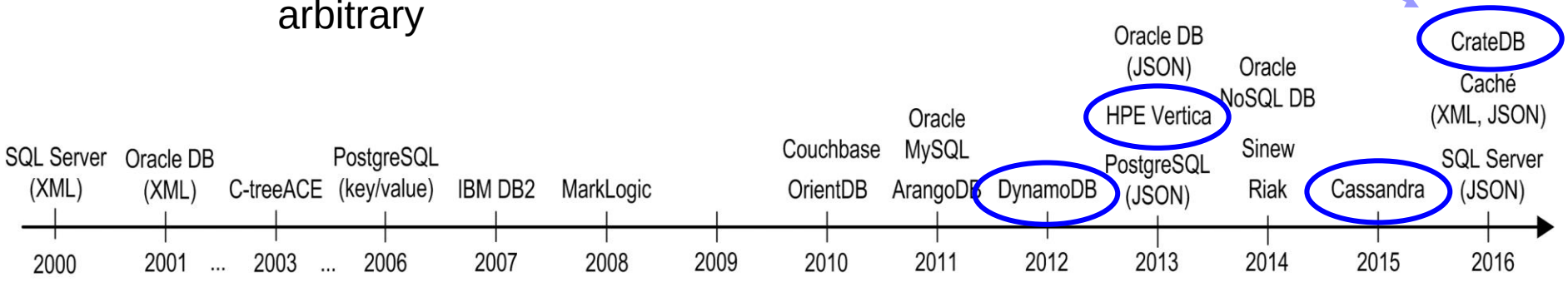
```
SELECT jsonb_object_keys(orders) FROM customer;
```

jsonb_object_keys text
Order_no
Orderlines
Order_no
Orderlines

Column Multi-model DBMSs

- Two meanings:
  - Column-oriented (columnar, column) DBMS stores data tables as columns rather than rows
    - Not necessarily NoSQL
  - Column-family (wide-column) DBMS = a NoSQL database which supports tables having distinct numbers and types of columns
    - Underlying storage strategy is arbitrary

Type	DBMS	Relational	Column	Key/value	Document (JSON)	XML	Graph	Nested data/UDT/object
Column	Cassandra		✓					✓
	CrateDB	✓	✓		✓			
	DynamoDB		✓	✓	✓			
	HPE Vertica		✓		✓			



Column Multi-model DBMSs

Storage – Cassandra Example

```
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>>
    );
```

value replaced as a whole

Column Multi-model DBMSs

Storage – Cassandra Example

```
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 } ] } ]
  }';
```

Column Multi-model DBMSs

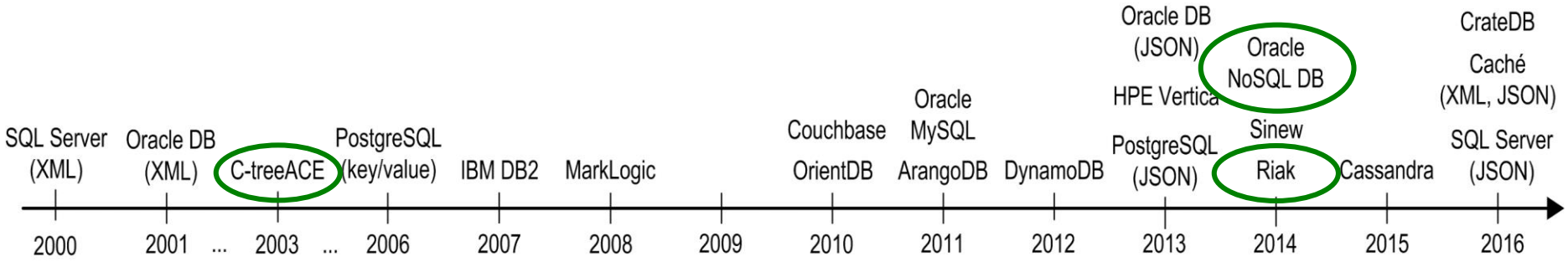
Storage – Cassandra Example

```
CREATE TABLE myspace.users (
   id text PRIMARY KEY,
   age int,
   country text
   );

INSERT INTO myspace.users (id, age, state) VALUES ('Irena', 37, 'CZ');

SELECT JSON * FROM myspace.users;


[json]
-----------------------------------------
{"id": "Irena", "age": 37, "country": "CZ"}
```

Key/Value Multi-model DBMSs

- The simplest type of NoSQL database
  - Get / put / delete + key
  - Often extended with more advanced features
- Multi-model extensions:
  - More complex indices over the value part + new APIs (e.g., JSON, SQL, ...)

Type	DBMS	Relational	Column	Key/value	Document (JSON)	XML	Graph	Nested data/UDT/object
Key/value	Riak			√	√	√		
	c-treeACE	√		√				
	Oracle NoSQL DB	√		√				



Key/Value Multi-model DBMSs

Storage – Oracle NoSQL DB Example

```
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
```

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  } ] } ]
}
```

ORACLE NOSQL DATABASE

Key/Value Multi-model DBMSs

Storage – Oracle NoSQL DB Example

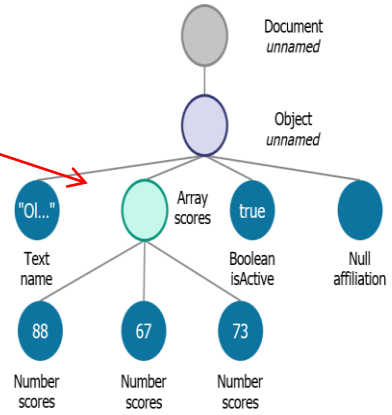
```
sql-> select * from Customers
   -> ;
 +----+------+----------+-----------------------------+
 | id | name | address  |           orders            |
 +----+------+----------+-----------------------------+
 |  2 | John | Helsinki | order_no          | 0c6df511 |
 |    |      |          | orderlines                   |
 |    |      |          |     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       |
 +----+------+----------+-----------------------------+
```

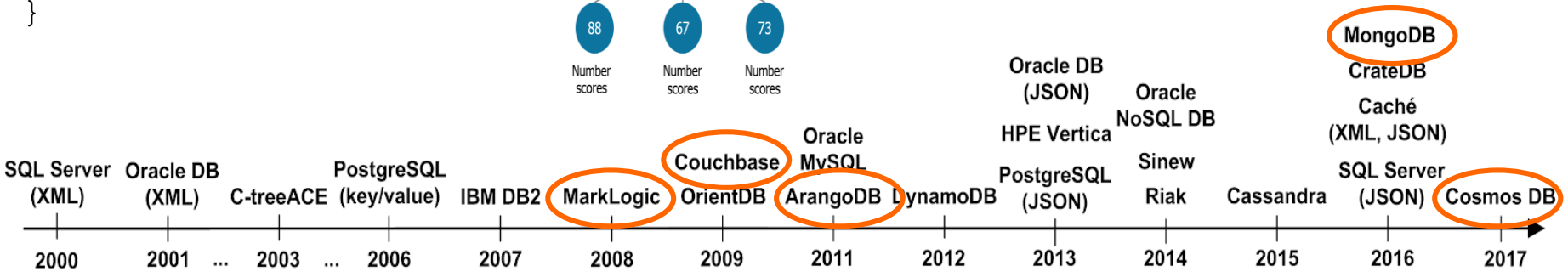
Document Multi-model DBMSs

- Distinct strategies:
  - ArangoDB: special edge collection
  - MarkLogic: stores JSON data as XML

Type	DBMS	Relational	Column	Key/value	Document (JSON)	XML	Graph	Nested data/UDT/object
Document	ArangoDB			√	√		√	
	Couchbase			√	√			
	MongoDB			√	√		√	
	Cosmos DB		√	√	√		√	
	MarkLogic				√	√		

```
{
   "name": "Oliver",
   "scores": [88, 67, 73],
   "isActive": true,
   "affiliation": null
}
```





Document Multi-model DBMSs

Storage – MarkLogic Example

JavaSript:

```
declareUpdate();
xdmp.documentInsert("/myJSON1.json",
{
  "Order_no":"0c6df508",
   "Orderlines":[
    { "Product_no":"2724f",
      "Product_Name":"Toy",
      "Price":66 },
    {"Product_no":"3424g",
     "Product_Name":"Book",
     "Price":40}]
}
);
```

XQuery:

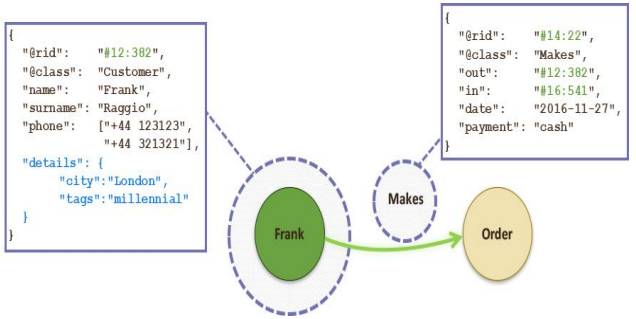
```
xdmp:document-insert("/myXML1.xml",
<product no="3424g">
  <name>The King's Speech</name>
  <author>Mark Logue</author>
  <author>Peter Conradi</author>
</product>
);
```

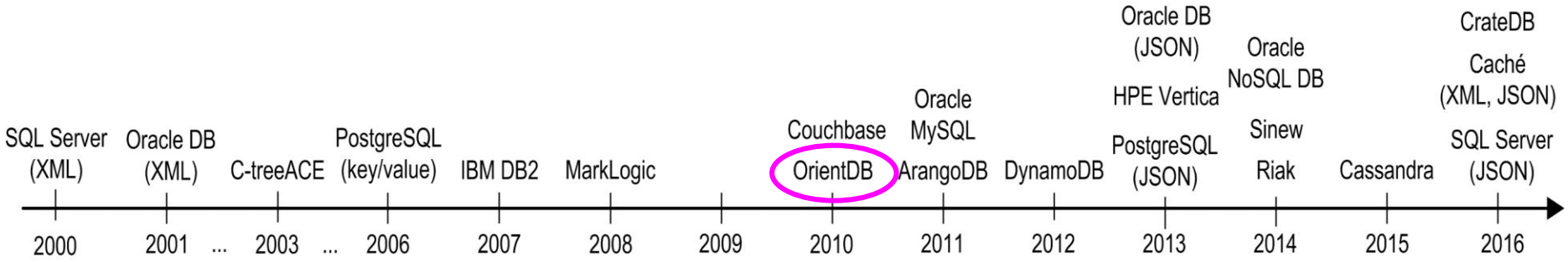
MarkLogic

Graph Multi-model DBMSs

- Based on an object database = native support for multiple models
  - Element of storage = record = document / BLOB / vertex / edge
- Classes – define records
- Classes can have relationships:
  - Referenced – stored similarly to storing pointers between two objects in memory
  - Embedded – stored within the record that embed

Type	DBMS	Relational	Column	Key/value	Document (JSON)	XML	Graph	Nested data/UDT/object
Graph	OrientDB			√	√		√	√





Graph Multi-model DBMSs

Storage – OrientDB Example

```
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
```

Graph Multi-model DBMSs

Storage – OrientDB Example

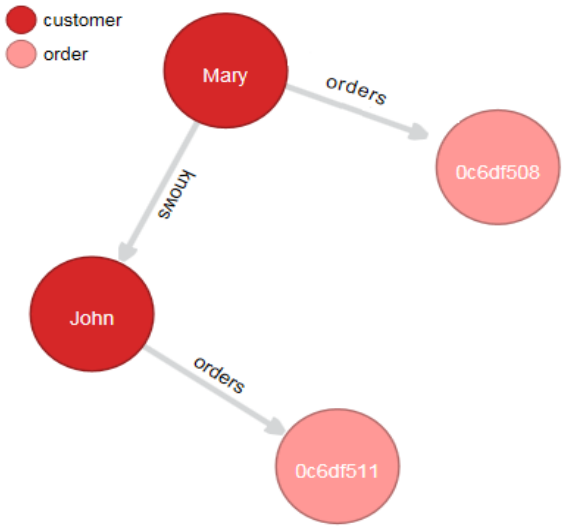
```
CREATE VERTEX order CONTENT {
   "order_no":"0c6df508",
   "orderlines":[
    { "@type":"d",
      "@class":"orderline",
      "product_no":"2724f",
      "product_name":"Toy",
      "price":66 },
    { "@type":"d",
      "@class":"orderline",
      "product_no":"3424g",
      "product_name":"Book",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

Graph Multi-model DBMSs

Storage – OrientDB Example

```
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")
```



Multi-model Query Languages

Classification of Approaches

1. Simple API
   - Store, retrieve, delete data
     - Typically key/value, but also other use cases
   - **DynamoDB** – simple data access + querying over indices using comparison operators
2. SQL Extensions and SQL-Like Languages
   - Most common
     - In most types of systems (relational, column, document, …)

Multi-model Query Languages

Classification of Approaches

3. SPARQL Query Extensions
   - **IBM DB2** - SPARQL 1.0 + subset of features from SPARQL 1.1
     - `SELECT`, `GROUP BY`, `HAVING`, `SUM`, `MAX`, …
     - Probably no extension for relational data
       - But: RDF triples are stored in table → SQL queries can be used over them too
4. XML Query Extensions
   - **MarkLogic** – JSON can be accessed using XPath
     - Tree representation like for XML
     - Can be called from XQuery and JavaScript
5. Full-text Search
   - In general quite common
   - **Riak** – Solr index + operations
     - Wildcards, proximity search, range search, Boolean operators, grouping, …

Type	DBMS	SQL extension
Relational	PostgreSQL	Getting an array element by index, an object field by key, an object at a specified path, containment of values/paths, top-level key-existence, deleting a key-value pair / a string element / an array element with specified index / a field / an element with specified path, ...
	SQL Server	JSON: export relational data in the JSON format, test JSON format of a text value, JavaScript-like path queries SQLXML: SQL view of XML data + XML view of SQL relations
	IBM DB2	SQL/XML + embedding SQL queries to XQuery expressions
	Oracle DB	SQL/XML + JSON extensions (`JSON_VALUE`, `JSON_QUERY`, `JSON_EXISTS`, ...)
Document	Couchbase	Clauses `SELECT`, `FROM` (multiple buckets), ... for JSON
	Cosmos DB	Clauses `SELECT`, `FROM` (with inner join), `WHERE` and `ORDER BY` for JSON
	ArangoDB	key/value: insert, look-up, update document: simple QBE, complex joins, functions, ... graph: traversals, shortest path searches
Key/value	Oracle NoSQL DB	SQL-like, extended for nested data structures
	c-treeACE	Simple SQL-like language
Column	Cassandra	`SELECT`, `FROM`, `WHERE`, `ORDER BY`, `LIMIT` with limitations
	CrateDB	Standard ANSI SQL 92 + nested JSON attributes
Graph	OrientDB	Classical joins not supported, the links are simply navigated using dot notation; main SQL clauses + nested queries
Object	Caché	SQL + object extensions (e.g. object references instead of joins)

SQL Extensions and SQL-Like Languages

PostgreSQL Example (relational)

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"}

```
{"Order_no":"0c6df508",
   "Orderlines":[
    { "Product_no":"2724f",
      "Product_Name":"Toy",
      "Price":66 },
    {"Product_no":"3424g",
     "Product_Name":"Book",
     "Price":40}]
}
```

```
SELECT name,
  orders->>'Order_no' as Order_no,
  orders#>'{Orderlines,1}'->>'Product_Name' as
Product_Name
FROM customer
where orders->>'Order_no' <> '0c6df511';
```

name character varying (50)	order_no text	product_name text
Mary	0c6df508	Book

ORACLE NOSQL DATABASE

SQL Extensions and SQL-Like Languages

Oracle NoSQL DB Example (key/value)

```
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       |
 +----+------+----------+-----------------------------+
```

ORACLE NOSQL DATABASE

SQL Extensions and SQL-Like Languages

Oracle NoSQL DB Example (key/value)

```
sql-> SELECT c.name, c.orders.order_no, c.orders.orderlines[0].product_name
   -> FROM customers c
   -> where c.orders.orderlines[0].price > 50;
 +------+----------+--------------+
 | name | order_no | product_name |
 +------+----------+--------------+
 | Mary | 0c6df508 | Toy          |
 +------+----------+--------------+

sql-> SELECT c.name, c.orders.order_no,
   -> [c.orders.orderlines[$element.price >35]]
   -> FROM customers c;
 +------+----------+------------------------+
 | name | order_no |        Column_3        |
 +------+----------+------------------------+
 | Mary | 0c6df508 | product_no   | 2724f   |
 |      |          | product_name | Toy     |
 |      |          | price        | 66      |
 |      |          |                        |
 |      |          | product_no   | 3424g   |
 |      |          | product_name | Book    |
 |      |          | price        | 40      |
 +------+----------+------------------------+
 | John | 0c6df511 |                        |
 +------+----------+------------------------+
```

XML Query Extensions

MarkLogic Example

JavaScript:

```
declareUpdate();
xdmp.documentInsert("/myJSON1.json"
,
{
  "Order_no":"0c6df508",
   "Orderlines":[
    { "Product_no":"2724f",
      "Product_Name":"Toy",
      "Price":66 },
    {"Product_no":"3424g",
     "Product_Name":"Book",
     "Price":40}]
}
);
```

XQuery:

```
xdmp:document-insert("/myXML1.xml",
<product no="3424g">
  <name>The King's Speech</name>
  <author>Mark Logue</author>
  <author>Peter Conradi</author>
</product>
);
```

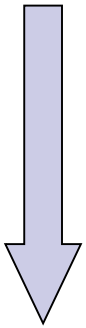
XQuery:

```
let $product := fn:doc("/myXML1.xml")/product
let $order := fn:doc("/myJSON1.json")
  [Orderlines/Product_no = $product/@no]
return $order/Order_no
```

Result: 0c6df508

Multi-model Query Processing

- Depends highly on the way the system was extended
  - No change
  - New interface
    - e.g. MarkLogic
  - Extension of the original storage strategy
    - e.g. ArangoDB
  - A completely new storage strategy
    - e.g. Oracle native support for XML



changes in the query processing approaches

- General tendencies:
  - Exploit the existing storage strategies as much as possible
  - Exploit the verified approaches to query optimization

JavaSript:

```
declareUpdate();
xdmp.documentInsert("/myJSON1.json"
,
{
  "Order_no":"0c6df508",
   "Orderlines":[
    { "Product_no":"2724f",
      "Product_Name":"Toy",
      "Price":66 },
    {"Product_no":"3424g",
     "Product_Name":"Book",
     "Price":40}]
}
);
```

XQuery:

```
xdmp:document-insert("/myXML1.xml",
<product no="3424g">
  <name>The King's Speech</name>
  <author>Mark Logue</author>
  <author>Peter Conradi</author>
</product>
);
```

XQuery:

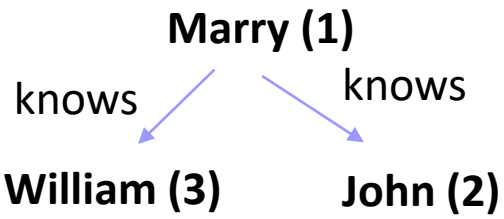
```
let $product := fn:doc("/myXML1.xml")/product
let $order := fn:doc("/myJSON1.json")
  [Orderlines/Product_no = $product/@no]
return $order/Order_no
```

Result: 0c6df508

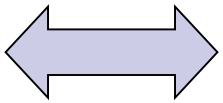
MarkLogic™

MarkLogic Multiple Models

- Indexes both XML and JSON data in the same way
- Schema-less data
- Universal index - optimized to allow text, structure and value searches to be combined into
  - Word indexing
  - Phrase indexing
  - Relationship indexing
  - Value indexing
- Other user-defined indices
  - Range indexing
  - Word lexicons
  - Reverse indexing
  - Triple index

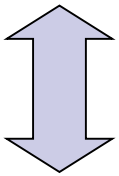


Graph-key/value join



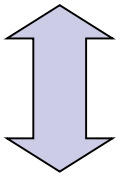
"1" --> "34e5e759"

"2"--> "0c6df508"



Relation-graph join

Key/value-JSON join



Customer_ID	Name	Credit_limits
1	Mary	5,000
2	John	3,000
3	William	2,000

```
{ "Order_no":"0c6df508",
  "Orderlines": [
    { "Product_no":"2724f"
      "Product_Name":"Toy",
      "Price":66 },
    { "Product_no":"3424g",
      "Product_Name":"Book",
      "Price":40 } ] }
```

Recommendation query:
Return all product_no-s which are ordered by a friend of a customer whose credit_limit>3000

ArangoDB

```
LET CustomerIDs = (
   FOR Customer IN Customers
   FILTER Customer.CreditLimit > 3000
   RETURN Customer.id)
LET FriendIDs = (
   FOR CustomerID IN CustomerIDs
        FOR Friend IN 1..1 OUTBOUND CustomerID Knows
   RETURN Friend.id)
FOR Friend in FriendIDs
FOR Order in 1..1 OUTBOUND Friend Customer2Order
RETURN Order.orderlines[*].Product_no
```

Recommendation query:
Return all product_no-s which are ordered by a friend of a customer whose credit_limit>3000

ArangoDB Multiple Models

- Supported models:
  - Document - original
  - Key/value - special type of document without complex value part
  - Tables - special type of document with regular structure
  - Graph - relations between documents
    - Edge document collection – two special attributes _from and _to
- So we still need to efficiently process queries over documents
- Indices
  - Primary = hash index for document keys
  - Edge = hash index, which stores the union of all _from and _to attributes
    - For equality look-ups
  - User-defined - (non-)unique hash/skiplist index, (non-)unique sparse hash/skiplist index, geo, fulltext, ...

Query Optimization Strategies

- B-tree/B+-tree index - the most common approach
  - Typically in relational databases
- Native XML index - support of XML data
  - Typically an ORDPATH-based approach
- Hashing - can be used almost universally
- ...
- But: still no universally acknowledged optimal or sub-optimal approach
  - Approaches are closely related to the way the system was extended

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

Multi-model Databases

Challenges

Open data model

Unified query language

Schema evolution and model evolution

Multi-model databases

Multi-model index structure

Multi-model transactions

Multi-model main memory structure

References

- Neither Fish Nor Fowl: the Rise of Multi-model Databases. The 451 Group, 2013.
- D. Feinberg, M. Adrian, N. Heudecker, A. M. Ronthal, and T. Palanca. Gartner Magic Quadrant for Operational Database Management Systems, 12 October 2015.
- J. Lu, Z. H. Liu, P. Xu, and C. Zhang. UDBMS: road to unification for multi-model data management. CoRR, abs/1612.08050, 2016
- J. Lu: Towards Benchmarking Multi-model Databases. CIDR 2017
- S. Abiteboul et al: Research Directions for Principles of Data Management, Dagstuhl Perspectives Workshop 16151 (2017)