Multi-model Data Management

Jiaheng Lu and Irena Holubová

University of Helsinki and Charles University, Prague
Outline

• Introduction to multi-model databases (25 minutes)
• Multi-model data storage (25 minutes)
• Multi-model data query languages (15 minutes)
• Multi-model query optimization (5 minutes)
• Multi-model database benchmarking (5 minutes)
• Open problems and challenges (10 minutes)
Outline

• Introduction to multi-model databases
• Multi-model data storage
• Multi-model data query languages
• Multi-model query optimization
• Multi-model database benchmarking
• Open problems and challenges
A grand challenge on Variety

• Big data: Volume, Variety, Velocity, Veracity
• Variety: tree data (XML, JSON), graph data (RDF, property graphs, networks), tabular data (CSV), temporal and spatial data, text etc.
Motivation: one application to include multi-model data

An E-commerce example with multi-model data
NoSQL database types


Multiple NoSQL databases

- MongoDB: Sales, Social media
- Neo4j: Recommendations
- Redis: Shopping-cart
- MongoDB: Customer, Catalog

Polyglot Persistence

• “One size cannot fit all”: use multiple databases for one application
• 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

- Handle multi-model data
- Help your apps to scale well
- A rich experience to manage multiple databases

- Requires the company to hire people to integrate different databases
- Implementers need to learn different databases
- Hard to handle inter-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.

• Document, graph, relational, and key-value models are examples of data models that may be supported by a multi-model database.
What is the difference between Multi-model and Multi-modal

• **Multi-model**: graph, tree, relation, key-value, ...

• **Multi-modal**: video, image, audio, eye gaze data, physiological signals, ...
Three arguments on one DB engine for multiple applications

• 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 database 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.

• Jens Dittrich, Alekh 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, i.e., all insert and update-operations create logical log-entries in that log.
- Based on that log, define several types of optional storage views
- The query optimization, view maintenance, and index selection problems suddenly become a single problem: storage view selection

One size can fit a bunch: AsterixDB [1]

A parallel semi-structured data management system with its own storage, indexing, run-time, language, and query optimizer, supporting JSON, CSV data

Support SQL++ [2] and AQL (AsterixDB query language)

One size can fit a bunch: AsterixDB

• AsterixDB’s data model is **flexible**

• **Open**: you can store objects there that have those fields as well as any/all other fields that your data instances happen to have at insertion time.

• **Closed**: you can choose to pre-define any or all of the fields and types that objects to be stored in it will have
A simple survey

How many of you agree that

1. One size cannot fit all?
2. One size can fit all?
3. One size fits a bunch?
4. ???

Multi-model databases: One size fits multi-data-model
Multi-model databases are not new!

- Can be traced to object-relational database (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

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
- One system guarantees inter-model 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 for documents, graphs, and key-values.

• They store all data as documents.

• Since vertices and edges of graphs are documents, this allows to mix all three data models (key-value, JSON and graph)
An example of multi-model data and query

Social network graph

"1" -- > "34e5e759"
"2" -- > "0c6df508"

Shopping-cart key-value pairs
Customer_ID → Order_no

Order JSON document

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

Customer relation

<table>
<thead>
<tr>
<th>Customer_ID</th>
<th>Name</th>
<th>Credit_limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mary</td>
<td>5,000</td>
</tr>
<tr>
<td>2</td>
<td>John</td>
<td>3,000</td>
</tr>
<tr>
<td>3</td>
<td>William</td>
<td>2,000</td>
</tr>
</tbody>
</table>
An example of multi-model data and query

<table>
<thead>
<tr>
<th>Customer_ID</th>
<th>Name</th>
<th>Credit_limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mary</td>
<td>5,000</td>
</tr>
<tr>
<td>2</td>
<td>John</td>
<td>3,000</td>
</tr>
<tr>
<td>3</td>
<td>Anne</td>
<td>2,000</td>
</tr>
</tbody>
</table>

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

```json
{"Order_no":"0c6df508", "Orderlines": [ { "Product_no":"2724f", "Product_Name":"Toy", "Price":66 }, { "Product_no":"3424g", "Product_Name":"Book", "Price":40 } ]
```
An example of multi-model query (ArangoDB)

Description: Return all products which are ordered by a friend of a customer whose credit_limit>3000

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

Result: ["2724f", "3424g"]
• 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-hybrid** modes.

• Query with **SQL** extended for graph traversal.
Description: Return all products which are ordered by a friend of a customer whose credit_limit>3000

```
Select expand(out("Knows").Orders.orderlines.Product_no)
from Customers where Credit_limit > 3000
```

Result: ["2724f", "3424g"]
Outline

• Introduction to multi-model databases
• Multi-model data storage
  • Multi-model data query languages
  • Multi-model query optimization
  • Multi-model database benchmarking
• Open problems and challenges
Classification and Timeline

<table>
<thead>
<tr>
<th>Relational</th>
<th>PostgreSQL, SQL Server, IBM DB2, Oracle DB, Oracle MySQL, Sinew</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column</td>
<td>Cassandra, CrateDB, DynamoDB, HPE Vertica</td>
</tr>
<tr>
<td>Key/value</td>
<td>Riak, c-treeACE, Oracle NoSQL DB</td>
</tr>
<tr>
<td>Document</td>
<td>ArangoDB, Couchbase, MarkLogic</td>
</tr>
<tr>
<td>Graph</td>
<td>OrientDB</td>
</tr>
<tr>
<td>Object</td>
<td>InterSystems Caché</td>
</tr>
<tr>
<td>Special</td>
<td>• Not yet multi-model – NuoDB, Redis, Aerospike</td>
</tr>
<tr>
<td></td>
<td>• Multi-use-case – SAP HANA DB, Octopus DB</td>
</tr>
</tbody>
</table>

### Classification and Timeline

<table>
<thead>
<tr>
<th>Relational</th>
<th>PostgreSQL, SQL Server, IBM DB2, Oracle DB, Oracle MySQL, Sinew</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column</td>
<td>Cassandra, CrateDB, DynamoDB, HPE Vertica</td>
</tr>
<tr>
<td>Key/value</td>
<td>Riak, c-treeACE, Oracle NoSQL DB</td>
</tr>
<tr>
<td>Document</td>
<td>ArangoDB, Couchbase, MarkLogic</td>
</tr>
<tr>
<td>Graph</td>
<td>OrientDB</td>
</tr>
<tr>
<td>Object</td>
<td>InterSystems Caché</td>
</tr>
<tr>
<td>Special</td>
<td>• Not yet multi-model – NuoDB, Redis, Aerospike</td>
</tr>
<tr>
<td></td>
<td>• Multi-use-case – SAP HANA DB, Octopus DB</td>
</tr>
</tbody>
</table>

![Database Classification Timeline](image-url)
Relational Multi-Model DBMSs

Storage

• Biggest set:
  1. Most popular type of DBMSs
  2. Extended to other models long before Big Data arrival
  3. Relational model enables simple extension

• PostgreSQL
  • Many NoSQL features: materialized views (data duplicities), master/slave replication
  • Data types: XML, HSTORE (key/value pairs), JSON / JSONB (JSON)

• SQL Server
  • Data types: XML, NVARCHAR (JSON)
  • SQLXML (not SQL/XML)
  • Function OPENJSON: JSON text $\rightarrow$ relational table
    • Pre-defined schema and mapping rules / without a schema (a set of key/value pairs)
Relational Multi-Model DBMSs

Storage

- **IBM DB2**
  - PureXML – native XML storage (or shredding into tables)
  - DB2-RDF – RDF graphs
    - Direct primary – triples + associated graph, indexed by subject
    - Reverse primary – triples + associated graph, indexed by object
    - Direct secondary – triples that share the subject and predicate within an RDF graph
    - Reverse secondary – triples that share the object and predicate within an RDF graph
  - Datatypes – mapping of internal integer values for SPARQL data types

- **Oracle DB**
  - Data types: XMLType (or shredded into tables), VARCHAR / BLOB / CLOB (JSON)
    - is_json check constraint
Relational Multi-Model DBMSs

Storage

• **Oracle MySQL**
  - Memcached API (2011): key/value data access
    - Default: key/value pairs are stored in rows of the same table
    - Key prefix can be defined to specify the table to be stored
  - Strength: combination with relational data access
  - MySQL cluster (2014): sharding and replication

• **Sinew**
  - Idea: a new layer above a relational DBMS that enables SQL queries over multi-structured data without having to define a schema
    - Relational, key-value, nested document etc.
  - Logical view = a universal relation
    - One column for each unique key in the data set
    - Nested data is flattened into separate columns

**Relational Multi-Model DBMSs**

Storage – PostgreSQL Example

```sql
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 }
   ]
  });
```

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>address</th>
<th>orders</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mary</td>
<td>Prague</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;{Orderlines&quot;:[&quot;Price&quot;:66,&quot;Product_Name&quot;:&quot;Toy&quot;,&quot;Product_no&quot;:&quot;2724f&quot;],[&quot;Price&quot;:40,&quot;Product_Name&quot;:...}</td>
</tr>
<tr>
<td>2</td>
<td>John</td>
<td>Helsinki</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;{Orderlines&quot;:[&quot;Price&quot;:34,&quot;Product_Name&quot;:&quot;Computer&quot;,&quot;Product_no&quot;:&quot;2454f&quot;],&quot;Order_no&quot;:&quot;0c6df511&quot;}</td>
</tr>
</tbody>
</table>
Relational Multi-Model DBMSs

Storage – PostgreSQL Example

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

SELECT jsonb_each(orders) FROM customer;

SELECT jsonb_object_keys(orders) FROM customer;

# Relational Multi-Model DBMSs

<table>
<thead>
<tr>
<th>Formats</th>
<th>Storage strategy</th>
<th>Query languages</th>
<th>Indices</th>
<th>Scale out</th>
<th>Flexible schema</th>
<th>Comb. data</th>
<th>Cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td>PostgreSQL</td>
<td>relational, key/value, JSON, XML</td>
<td>relational tables - text or binary format + indices</td>
<td>SQL ext.</td>
<td>inverted</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>SQL Server</td>
<td>relational, XML, JSON, ...</td>
<td>text, relational tables</td>
<td>SQL ext.</td>
<td>B-tree, full-text</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>IBM DB2</td>
<td>relational, XML, RDF</td>
<td>native XML type / relations for RDF</td>
<td>Extended SQL / XML / SPARQL 1.0/1.1</td>
<td>XML paths / B+ tree, fulltext</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Oracle DB</td>
<td>relational, XML, JSON</td>
<td>relational, native XML</td>
<td>SQL/XML, JSON SQL ext.</td>
<td>bitmap, B+ tree, function-based, XMLIndex</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Oracle MySQL</td>
<td>relational, key/value</td>
<td>relational</td>
<td>SQL, memcached API</td>
<td>B-tree</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Sinew</td>
<td>relational, key/value, nested document, ...</td>
<td>logically a universal relation, physically partially materialized</td>
<td>SQL</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Y</td>
</tr>
</tbody>
</table>
## Classification and Timeline

<table>
<thead>
<tr>
<th>Classification</th>
<th>Relational</th>
<th>Column</th>
<th>Key/Value</th>
<th>Document</th>
<th>Graph</th>
<th>Object</th>
<th>Special</th>
</tr>
</thead>
</table>
|                | PostgreSQL, SQL Server, IBM DB2, Oracle DB, Oracle MySQL, Sinew | Cassandra, CrateDB, DynamoDB, HPE Vertica | Riak, c-treeACE, Oracle NoSQL DB | ArangoDB, Couchbase, MarkLogic | OrientDB | InterSystems Caché | • Not yet multi-model – NuoDB, Redis, Aerospike  
• Multi-use-case – SAP HANA DB, Octopus DB |

![Database Timeline Diagram](image_url)
Column Multi-Model DBMSs

Storage

• Two meanings:
  1. **Column-oriented** (columnar, column) DBMS stores data tables as columns rather than rows
     • Not necessarily NoSQL, usually in analytics tools
  2. **Column** (wide-column) DBMS = a NoSQL database which supports tables having distinct numbers and types of columns
     • Underlying storage strategy can be columnar, or any other

• **Cassandra**
  • Column store with sparse tables
  • SSTables (Sorted String Tables) – proposed in Google system Bigtable
  • SQL-like query and manipulation language CQL
  • Scalar data types (text, int), collections (list, set, map), tuples, and UDTs
  • 2015: JSON format (schema of tables must be defined)
     • Keys ↔ column names
     • JSON values ↔ column values
Column Multi-Model DBMSs

Storage

• **CrateDB**
  - Distributed columnar SQL database, dynamic schema
    - Built upon Elasticsearch, Lucene, ...
  - Nested JSON documents, arrays, BLOBs
  - Row of a table = (nested) structured document
    - Operations on documents are atomic

• **DynamoDB**
  - Document (JSON) and key/value flexible data models
  - (Schemaless) table = collection of items
    - Item (uniquely identified by a primary key) = collection of attributes
      - Attribute = name + data type + value
      - Data type: value (string, number, Boolean ...), document (list or map), set of scalar values
  - Data items in a table need not have the same attributes
Column Multi-Model DBMSs

Storage

- **HPE Vertica**
  - High-performance analytics engine
  - Storage organization: column oriented + SQL interface + analytics capabilities
  - 2013 – flex tables
    - Do not require schema definitions
    - Enable to store semi-structured data (JSON, CSV,...)
    - Support SQL queries
    - Loaded data stored in internal map (set of key/value pairs) = virtual columns
      - Selected keys can be materialized = real table columns
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>>
);
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 } ] } ]
}';
```
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"}
<table>
<thead>
<tr>
<th></th>
<th>Formats</th>
<th>Storage strategy</th>
<th>Query languages</th>
<th>Indices</th>
<th>Scale out</th>
<th>Flexible schema</th>
<th>Comb. data</th>
<th>Cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cassandra</strong></td>
<td>text, user-defined type</td>
<td>sparse tables</td>
<td>SQL-like CQL</td>
<td>inverted, B+ tree</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td><strong>CrateDB</strong></td>
<td>relational, JSON, BLOB, arrays</td>
<td>columnar store based on Lucene and Elasticsearch</td>
<td>SQL</td>
<td>Lucene</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td><strong>DynamoDB</strong></td>
<td>key/value, document (JSON)</td>
<td>column store</td>
<td>simple API (get / put / update) + simple queries over indices</td>
<td>hashing</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td><strong>HPE Vertica</strong></td>
<td>JSON, CSV</td>
<td>flex tables + map</td>
<td>SQL-like</td>
<td>for materialized data</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>
### Classification and Timeline

<table>
<thead>
<tr>
<th>Classification</th>
<th>PostgreSQL, SQL Server, IBM DB2, Oracle DB, Oracle MySQL, Sinew</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational</td>
<td><img src="image1.png" alt="Image" /></td>
</tr>
<tr>
<td>Column</td>
<td>Cassandra, CrateDB, DynamoDB, HPE Vertica</td>
</tr>
<tr>
<td>Key/value</td>
<td>Riak, c-treeACE, Oracle NoSQL DB</td>
</tr>
<tr>
<td>Document</td>
<td>ArangoDB, Couchbase, MarkLogic</td>
</tr>
<tr>
<td>Graph</td>
<td>OrientDB</td>
</tr>
<tr>
<td>Object</td>
<td>InterSystems Caché</td>
</tr>
<tr>
<td>Special</td>
<td>• Not yet multi-model – NuoDB, Redis, Aerospike</td>
</tr>
<tr>
<td></td>
<td>• Multi-use-case – SAP HANA DB, Octopus DB</td>
</tr>
</tbody>
</table>

![Diagram](image2.png)
Key/Value Multi-Model DBMSs

Storage

• **Riak**
  - 2009: classical key/value DBMS
  - 2014: document store with querying capabilities
    - Riak Data Types – conflict-free replicated data type
      - Sets, maps (enable embedding), counters,...
    - Riak Search – integration of Solr for indexing and querying
      - Indices over particular fields of XML/JSON document, plain text, ...

• **c-treeACE**
  - **No+SQL** = both NoSQL and SQL in a single database
  - Key/value store + support for relational and non-relational APIs
  - Record-oriented Indexed Sequential Access Method (ISAM) structure
    - Operations with records, their sets, or files in which they are stored
Key/Value Multi-Model DBMSs

Storage

• **Oracle NoSQL DB**
  • Built upon the Oracle Berkeley DB
    • Released in 2011
  • Key/value store which supports table API = SQL (since 2014)
    • Data can be modelled as:
      • Relational tables
      • JSON documents
      • Key/value pairs
  • Definition of tables must be provided
    • Table and attribute names, data types, keys, indices, ...
    • Data types: scalar types, arrays, maps, records, child tables (nested subtables)
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:

```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  
        }  
      ]  
    }  
  ]  
}
```
### Key/Value Multi-Model DBMSs

#### Storage – Oracle NoSQL DB Example

```sql
sql-> select * from Customers
  -> ;
+-----------------------------+------------+----------+----------------+-----------------+-----------------+-----------------+-----------------+
<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>address</th>
<th>orders</th>
<th>orderlines</th>
<th>product_no</th>
<th>product_name</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>John</td>
<td>Helsinki</td>
<td>order_no</td>
<td>0c6df511</td>
<td>2454f</td>
<td>Computer</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>orderlines</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>product_no</td>
<td>2454f</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>product_name</td>
<td>Computer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>price</td>
<td>34</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
+-----------------------------+------------+----------+----------------+-----------------+-----------------+-----------------+
| 1  | Mary     | Prague   | order_no  | 0c6df508      | 3424g       | Book           | 40    |
|    |          |          | orderlines|               |             |                |       |
|    |          |          | product_no| 3424g         |             |                |       |
|    |          |          | product_name| Book        |             |                |       |
|    |          |          | price      | 40            |             |                |       |
+-----------------------------+------------+----------+----------------+-----------------+-----------------+-----------------+
```

# Key/Value Multi-Model DBMSs

<table>
<thead>
<tr>
<th></th>
<th>Formats</th>
<th>Storage strategy</th>
<th>Query languages</th>
<th>Indices</th>
<th>Scale out</th>
<th>Flexible schema</th>
<th>Comb. data</th>
<th>Cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td>Riak</td>
<td>key/value, XML, JSON</td>
<td>key/value pairs in buckets</td>
<td>Solr</td>
<td>Solr</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>c-treeACE</td>
<td>key/value + SQL API</td>
<td>record-oriented ISAM</td>
<td>SQL</td>
<td>ISAM</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
<td>N</td>
</tr>
<tr>
<td>Oracle NoSQL DB</td>
<td>key/value, (hierarchical) table API</td>
<td>key/value</td>
<td>SQL</td>
<td>B-tree</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>

## Classification and Timeline

<table>
<thead>
<tr>
<th>Relational</th>
<th>PostgreSQL, SQL Server, IBM DB2, Oracle DB, Oracle MySQL, Sinew</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column</td>
<td>Cassandra, CrateDB, DynamoDB, HPE Vertica</td>
</tr>
<tr>
<td>Key/value</td>
<td>Riak, c-treeACE, Oracle NoSQL DB</td>
</tr>
<tr>
<td>Document</td>
<td>ArangoDB, Couchbase, MarkLogic</td>
</tr>
<tr>
<td>Graph</td>
<td>OrientDB</td>
</tr>
<tr>
<td>Object</td>
<td>InterSystems Caché</td>
</tr>
<tr>
<td>Special</td>
<td>• Not yet multi-model – NuoDB, Redis, Aerospike</td>
</tr>
<tr>
<td></td>
<td>• Multi-use-case – SAP HANA DB, Octopus DB</td>
</tr>
</tbody>
</table>

### Timeline Diagram

- **Relational Databases:** PostgreSQL, SQL Server, IBM DB2, Oracle DB, Oracle MySQL, Sinew
- **Column Databases:** Cassandra, CrateDB, DynamoDB, HPE Vertica
- **Key/Value Databases:** Riak, c-treeACE, Oracle NoSQL DB
- **Document Databases:** ArangoDB, Couchbase, MarkLogic
- **Graph Databases:** OrientDB
- **Object Databases:** InterSystems Caché
- **Special Databases:**
  - Not yet multi-model – NuoDB, Redis, Aerospike
  - Multi-use-case – SAP HANA DB, Octopus DB

The timeline extends from 2000 to 2016, showcasing the chronological appearance of these databases.
Document Multi-Model DBMSs

Storage

• Document DB = key/value, where value is complex
  • Multi-model extension is natural

• ArangoDB
  • Denoted as native multi-model database
  • Key/value, (JSON) documents and graph data
    • Document collection – always a primary key attribute
      • No secondary indices → simple key/value store
    • Edge collection – two special attributes from and to
      • Relations between documents

• Couchbase
  • Key/value + (JSON) document
    • No pre-defined schema
  • SQL-based query language
  • Memcached buckets – support caching of frequently-used data
    • Reduce the number of queries

Multi-Model DBMSs

Storage

• **MarkLogic**
  • Originally XML
    • Since 2008: JSON
    • Currently: RDF, textual, binary data
  • Models a JSON document similarly to an XML document = a tree
    • Rooted at an auxiliary document node
    • Nodes below: JSON objects, arrays, and text, number, Boolean, null values
→ unified way to manage and index documents of both types
{  "name": "Oliver",  "scores": [88, 67, 73],  "isActive": true,  "affiliation": null}
Document Multi-Model DBMSs

Storage – MarkLogic Example

JavaScript:

```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:

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

<table>
<thead>
<tr>
<th>Formats</th>
<th>Storage strategy</th>
<th>Query languages</th>
<th>Indices</th>
<th>Scale out</th>
<th>Flexible schema</th>
<th>Comb. data</th>
<th>Cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArangoDB</td>
<td>document store allowing references</td>
<td>SQL-like AQL</td>
<td>mainly hash (eventually unique or sparse)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Couchbase</td>
<td>document store + append-only write</td>
<td>SQL-based N1QL</td>
<td>B+tree, B+trie</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>MarkLogic</td>
<td>storing like hierarchical XML data</td>
<td>XPath, XQuery, SQL-like</td>
<td>inverted + native XML</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>

## Classification and Timeline

<table>
<thead>
<tr>
<th>Classification</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational</td>
<td>PostgreSQL, SQL Server, IBM DB2, Oracle DB, Oracle MySQL, Sinew</td>
</tr>
<tr>
<td>Column</td>
<td>Cassandra, CrateDB, DynamoDB, HPE Vertica</td>
</tr>
<tr>
<td>Key/value</td>
<td>Riak, c-treeACE, Oracle NoSQL DB</td>
</tr>
<tr>
<td>Document</td>
<td>ArangoDB, Couchbase, MarkLogic</td>
</tr>
<tr>
<td>Graph</td>
<td>OrientDB</td>
</tr>
<tr>
<td>Object</td>
<td>InterSystems Caché</td>
</tr>
<tr>
<td>Special</td>
<td>• Not yet multi-model – NuoDB, Redis, Aerospike</td>
</tr>
<tr>
<td></td>
<td>• Multi-use-case – SAP HANA DB, Octopus DB</td>
</tr>
</tbody>
</table>
Graph Multi-Model DBMSs

Storage

- **OrientDB**
  - Data models: graph, document, key/value, object
  - Element of storage = a record corresponding to document / BLOB / vertex / edge
    - Having a unique ID
  - Classes – contain and define records
    - Schema-less / schema-full / schema-mixed
    - Can inherit (all properties) from other classes
      - Class properties are defined, further constrained or indexed
  - Classes can have relationships:
    - Referenced relationships – stored similarly to storing pointers between two objects in memory
      - LINK, LINKSET, LINKLIST, LINKMAP
    - Embedded relationships – stored within the record that embed
      - EMBEDDED, EMBEDDEDSET, EMBEDDEDLIST, EMBEDDEDMAP

<table>
<thead>
<tr>
<th>Formats</th>
<th>Storage strategy</th>
<th>Query languages</th>
<th>Indices</th>
<th>Scale out</th>
<th>Flexible schema</th>
<th>Comb. data</th>
<th>Cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td>OrientDB</td>
<td>graph, document, key/value, object</td>
<td>key/value pairs + object-oriented links</td>
<td>Gremlin, SQL ext.</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>
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
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 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 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")
## Classification and Timeline

<table>
<thead>
<tr>
<th>Relational</th>
<th>PostgreSQL, SQL Server, IBM DB2, Oracle DB, Oracle MySQL, Sinew</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column</td>
<td>Cassandra, CrateDB, DynamoDB, HPE Vertica</td>
</tr>
<tr>
<td>Key/value</td>
<td>Riak, c-treeACE, Oracle NoSQL DB</td>
</tr>
<tr>
<td>Document</td>
<td>ArangoDB, Couchbase, MarkLogic</td>
</tr>
<tr>
<td>Graph</td>
<td>OrientDB</td>
</tr>
<tr>
<td><strong>Object</strong></td>
<td>InterSystems Caché</td>
</tr>
<tr>
<td>Special</td>
<td>• Not yet multi-model – NuoDB, Redis, Aerospike</td>
</tr>
<tr>
<td></td>
<td>• Multi-use-case – SAP HANA DB, Octopus DB</td>
</tr>
</tbody>
</table>

### Timeline

- **SQL Server (XML)**
- **Oracle DB (XML)**
- **C-treeACE (key/value)**
- **PostgreSQL**
- **IBM DB2**
- **MarkLogic**
- **Couchbase**
- **OrientDB**
- **Oracle MySQL**
- **ArangoDB**
- **DynamoDB**
- **Oracle DB (JSON)**
- **HPE Vertica**
- **Oracle NoSQL DB**
- **PostgreSQL (JSON)**
- **Sinek**
- **Cassandra**
- **CBS**
- **SQL Server (JSON)**

**Key Dates:**
- 2000: Introduction of SQL Server (XML)
- 2001: Introduction of Oracle DB (XML)
- 2003: Introduction of C-treeACE (key/value)
- 2006: Introduction of PostgreSQL
- 2007: Introduction of IBM DB2
- 2008: Introduction of MarkLogic
- 2009: Introduction of Couchbase
- 2010: Introduction of OrientDB
- 2011: Introduction of Oracle MySQL
- 2012: Introduction of ArangoDB
- 2013: Introduction of DynamoDB
- 2014: Introduction of Oracle DB (JSON)
- 2015: Introduction of HPE Vertica
- 2016: Introduction of Oracle NoSQL DB
- 2017: Introduction of PostgreSQL (JSON)
- 2018: Introduction of Sinek
- 2019: Introduction of Cassandra
- 2020: Introduction of CBS
- 2021: Introduction of SQL Server (JSON)
Object Multi-Model DBMSs

Storage

• Object model = storing any kind of data → multi-model extension is natural

• InterSystems Caché
  • Stores data in sparse, multidimensional arrays
    • Capable of carrying hierarchically structured data
  • Access APIs: object (ODMG), SQL, direct manipulation of multidimensional data structures
    • Schemaless and schema-based storage strategy is available
  • 2016: JSON, XML

<table>
<thead>
<tr>
<th>Formats</th>
<th>Storage strategy</th>
<th>Query languages</th>
<th>Indices</th>
<th>Scale out</th>
<th>Flexible schema</th>
<th>Comb. data</th>
<th>Cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caché</td>
<td>object, SQL or multi-dimensional, document (JSON, XML) API</td>
<td>multi-dimensional arrays</td>
<td>SQL with object extensions</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
<td>N</td>
</tr>
</tbody>
</table>
Not (yet) multi-model

• **NuoDB** – NewSQL cloud DBMS
  • Data is stored in and managed through objects called Atoms
    • Self-coordinating objects (data, indices or schemas)
  • Atomicity, Consistency and Isolation are applied to Atom interaction
    • Replacing the SQL front-end would have no impact

• **Redis** – NoSQL key/value DBMS
  • Support for strings + a list of strings, an (un)ordered set of strings, a hash table, ... + respective operations
  • Redis Modules – add-ons which extend Redis to cover most of the popular use cases

• **Aerospike** – NoSQL key/value DBMS
  • Support for maps and lists in the value part that can nest
  • 2012 - Aerospike acquired AlchemyDB
    • Aim: to integrate its index, document store, graph database, and SQL functionality
Outline

• Introduction to multi-model databases
• Multi-model data storage
• **Multi-model data query languages**
• Multi-model query optimization
• Multi-model database benchmarking
• Open problems and challenges
Classification of Approaches

• Simple API
  • Store, retrieve, delete data
    • Typically key/value, but also other use cases
  • DynamoDB – simple data access + querying over indices using comparison operators

• SQL Extensions and SQL-Like Languages
  • Most common
  • PostgreSQL – SQL extension for JSON
  • Cassandra – CQL = subset of SQL, lots of limitations
  • OrientDB – Gremlin or SQL extended for graph traversal
  • SQL Server – SQLXML + similar extension for JSON
    • Not SQL/XML standard!
Classification of Approaches

- **IBM DB2** – SQL/XML + further extensions for XML
- **Oracle DB** – SQL/XML + further extensions for JSON
- **ArangoDB** – AQL = SQL-like + concept of loops
- **InterSystems Caché** – SQL + object concepts
  - Instances of classes accessible as rows of tables
  - Inheritance is “flattened”
- **Couchbase** – N1QL = SQL-like for JSON
- **CrateDB** – standard ANSI SQL 92 + usage of nested JSON attributes

<table>
<thead>
<tr>
<th>Database</th>
<th>Type</th>
<th>Features and Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>PostgreSQL</td>
<td>relational</td>
<td>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,...</td>
</tr>
<tr>
<td>SQL Server</td>
<td>relational</td>
<td>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</td>
</tr>
<tr>
<td>IBM DB2</td>
<td>relational</td>
<td>SQL/XML + e.g. embedding SQL queries to XQuery expressions</td>
</tr>
<tr>
<td>Oracle DB</td>
<td>relational</td>
<td>SQL/XML + JSON extensions (JSON_VALUE, JSON_QUERY, JSON_EXISTS,...)</td>
</tr>
<tr>
<td>Couchbase</td>
<td>document</td>
<td>Classical clauses such as SELECT, FROM (multiple buckets), ... for JSON</td>
</tr>
<tr>
<td>ArangoDB</td>
<td>document</td>
<td>key/value: insert, look-up, update document: simple QBE, complex joins, functions, ... graph: traversals, shortest path searches</td>
</tr>
<tr>
<td>Oracle NoSQL DB</td>
<td>key/value</td>
<td>SQL-like, extended for nested data structures</td>
</tr>
<tr>
<td>c-treeACE</td>
<td>key/value</td>
<td>SQL-like language</td>
</tr>
<tr>
<td>Cassandra</td>
<td>column</td>
<td>SELECT, FROM, WHERE, ORDER BY, LIMIT with limitations</td>
</tr>
<tr>
<td>CrateDB</td>
<td>column</td>
<td>Standard ANSI SQL 92 + usage nested JSON attributes</td>
</tr>
<tr>
<td>OrientDB</td>
<td>graph</td>
<td>Classical joins not supported, the links are simply navigated using dot notation; main SQL clauses + nested queries</td>
</tr>
<tr>
<td>Caché</td>
<td>object</td>
<td>SQL + object extensions (e.g. object references instead of joins)</td>
</tr>
</tbody>
</table>
SQL Extensions and SQL-Like Languages
PostgreSQL Example (relational)

```
{
"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';
```
### SQL Extensions and SQL-Like Languages

**Oracle NoSQL DB Example (key/value)**

```sql
sql-> SELECT c.name, c.orders.order_no, c.orders.orderlines[0].product_name
    -> FROM customers c
    -> WHERE c.orders.orderlines[0].price > 50;
```

<table>
<thead>
<tr>
<th>name</th>
<th>order_no</th>
<th>product_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>0c6df508</td>
<td>Toy</td>
</tr>
</tbody>
</table>

```sql
sql-> SELECT c.name, c.orders.order_no,
    -> c.orders.orderlines[$element.price >35]
    -> FROM customers c;
```

<table>
<thead>
<tr>
<th>name</th>
<th>order_no</th>
<th>Column_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>0c6df508</td>
<td>product_no 2724f</td>
</tr>
<tr>
<td></td>
<td></td>
<td>product_name Toy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>price 66</td>
</tr>
<tr>
<td></td>
<td></td>
<td>product_no 3424g</td>
</tr>
<tr>
<td></td>
<td></td>
<td>product_name Book</td>
</tr>
<tr>
<td></td>
<td></td>
<td>price 40</td>
</tr>
</tbody>
</table>

```sql
sql-> SELECT * FROM Customers
```
Classification of Approaches

• **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

• **XML Query Extensions**
  - **MarkLogic** – JSON can be accessed using XPath
    - Tree representation like for XML
    - Can be called from XQuery and JavaScript

• **Full-text Search**
  - In general quite common
  - **Riak** – Solr index + operations
    - Wildcards, proximity search, range search, Boolean operators, grouping, ...
XML Query Extensions

MarkLogic Example

JavaSript:

```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:

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

XQuery:

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

Result: 0c6df508
Outline

• Introduction to multi-model databases
• Multi-model data storage
• Multi-model data query languages
• Multi-model query optimization
• Multi-model database benchmarking
• Open problems and challenges
Classification of Approaches

• Inverted Index
  • PostgreSQL – data in jsonb: GIN index = (key, posting list) pairs
    • But also B-tree and hash index

• B-tree, B+ tree
  • Cassandra
    • Primary key = always indexed using inverted index (auxiliary table)
    • Secondary index = memory mapped B+trees (range queries)
  • SQL Server – no special index for JSON (B-tree or full-text indices)
  • Couchbase – B+tree / B+trie (a hierarchical B+tree-based Trie) = a shallower tree hierarchy

• Oracle DB
  • Shredded XML data = B+tree index
  • To index fields of a JSON object = virtual columns need to be created for them first + B+tree index

• Oracle MySQL – mostly classical B-trees (spatial data R-trees)

• Oracle NoSQL DB – secondary indices = distributed, shard-local B-trees
  • Indexing over simple, scalar as well as over non-scalar and nested data values
Classification of Approaches

• Materialization
  • **HPE Vertica** – flex table can be processed using SQL commands + custom views can be created
    • SELECT invokes `maplookup()` function
    • Promoting virtual columns to real columns improves query performance

• Hashing
  • **OrientDB**
    • SB trees – B-tree optimized for data insertions and range queries
    • Extendible hashing – significantly faster
  • **ArangoDB**
    • Primary index – hash index for document `_key` attributes of all documents in a collection
    • Edge index – hash index for `_from` and `_to` attributes
    • User-defined indices – hash, unsorted (can be unique or sparse) → no range queries
  • **DynamoDB**
    • Primary key index: partition key (determine partition) + sort key (within partition)
    • Secondary index: global (involving partition key) and local (within a partition)
Classification of Approaches

• Bitmap
  • InterSystems Caché – a series of highly compressed bitstrings to represent the set of object IDs = indexed value
    • Extended with bitslice index for numeric data fields used for a SUM, COUNT, or AVG
  • Oracle DB – can be created for a value returned by json_exists

• Function based
  • Oracle DB – indexes the function on a column = the product of the function
    • Can be created for SQL function json_value
    • For XML data deprecated
Classification of Approaches

• Native XML
  • MarkLogic
    • Universal index – inverted index for each word (or phrase), XML element and JSON property and their values
      • Further optimized using hashing
    • Index of parent-child relationships
    • (User-specified) range indices – for efficient evaluation of range queries
      • An array of document ids and values sorted by document ids + an array of values and document ids sorted by values
      • Path range index – to index JSON properties defined by an XPath expression
  • DB2 – XML region index, XML column path index, XML index
  • Oracle DB – XMLIndex = path index + order index + value index
    • Position of each node is preserved using a variant of the ORDPATHS numbering schema
Query Optimization – Inverted Index

PostgreSQL Example (GIN – Generalized Inverted Index)

• Two types:
  • Default (jsonb_ops) - key-exists operators ?, ?& and ?| and path/value-exists operator @>
    • Independent index items for each key and value in the data
  • Non-default (jsonb_path_ops) - indexing the @> operator only
    • Index items only for each value in the data
      • A hash of the value and the key(s) leading to it

• Example: {"foo": {"bar": "baz"} }
  • Default: three index items representing foo, bar, and baz separately
    • Containment query looks for rows containing all three of these items
  • Non-default: single index item (hash) incorporating foo, bar, and baz
    • Containment query searches for specific structure
Outline

• Introduction to multi-model databases
• Multi-model data storage
• Multi-model data query languages
• Multi-model query optimization
• Multi-model database benchmarking
• Open problems and challenges
Some Big data benchmarking initiatives

• **HiBench**, Yan Li et al., Intel
• **Yahoo Cloud Serving Benchmark (YCSB)**, Brian Cooper et al., Yahoo!
• **Berkeley Big Data Benchmark**, Pavlo et al., AMPLab
• **BigDataBench**, Jianfeng Zhan, Chinese Academy of Sciences
• **Bigframe**
• **LDCS** graph and RDF benchmarking
• ......
New challenges for multi-model databases

• Cross-model query processing
  • Complex joins of cross-model data
• Cross-model transaction
  • Transactions support cross-model
• Open schema data and model evolution
  • Query data with varied schemas and models
UniBench: A unified benchmark for multi-model data

An E-commerce application involving multi-model data

J. Lu: Towards Benchmarking Multi-Model Databases. CIDR 2017
Workloads

- Workload A: Data Insertion and reading
- Workload B: Cross-model query
- Workload C: Cross-model Transaction
On-going work on multi-model benchmarking

• Flexible schema management
• Model evolution
• HTAP (Hybrid Transaction/Analytical Processing)

• The data and code (on-going update) can be downloaded at:
• http://udbms.cs.helsinki.fi/?projects/ubench
Outline

• Introduction to multi-model databases
• Multi-model data storage
• Multi-model data query languages
• Multi-model query optimization
• Multi-model database benchmarking

• Open problems and challenges

Six challenges

- Open data model
- Unified query language
- Schema evolution and model evolution
- Multi-model index structure
- Multi-model transactions
- Multi-model main memory structure
Open data model

A flexible data model to accommodate multi-model data
Providing a convenient unique interface to handle data from different sources
A new unified query language can query multi-model data together
Multi-model query language

• SQL extension embedding data model specific languages
  • ORACLE: SQL/XML, SQL/JSON, SQL/SPARQL
• Graph extension
  • AQL ArangoDB language
• XQuery extension
  • MarkLogic
• JSON extension
  • MongoDB $graphLookup
Model evolution

- Relational table (Legacy data)
- JSON document (New data)

Model mapping among different models of data
Multi-model index structures

- Inter-model indexes to speedup the inter-model query processing

- A new index structure for graph, document and relational joins
Multi-model main memory structure

• As the in-memory technology going forward, disk based index and data storage model are constantly being challenged.

• Building up just-in-time multi-model data structure is a new challenge on main memory multi-model database.

• For example, In-memory virtual column[1] --> In-memory virtual model

Multi-model transaction

• How to process inter-model transactions?
• Graph data and relational data may have different requirements on the consistency models

An example of multi-model data hybrid consistency models
Some theoretical challenges on multi-model databases

• Schema language for multi-model data and schema extraction

• Multi-model query language: expressive power or higher complexity of query language (involving logic, complexity and automata theories)

• Query evaluation and optimization on inter-model
Conclusion

Classification of multi-model data management

Conclusion

• Multi-model database is not new
  • Can be traced to ORDBMS
  • A number of DBs can manage multiple models of data
  • By 2017, most of leading operational DBs will support multi-models.

• Multi-model database is new and open
  • New query language for multi-model data
  • New query optimization and indexes
  • Open data model and model evolution
  • ...
• Slides and papers are available at:
  • http://udbms.cs.helsinki.fi/?tutorials

• Open multi-model datasets
  • http://udbms.cs.helsinki.fi/?datasets

• Multi-model database benchmark
  • http://udbms.cs.helsinki.fi/?projects/ubench

Contact us:
jiahenglu@gmail.com