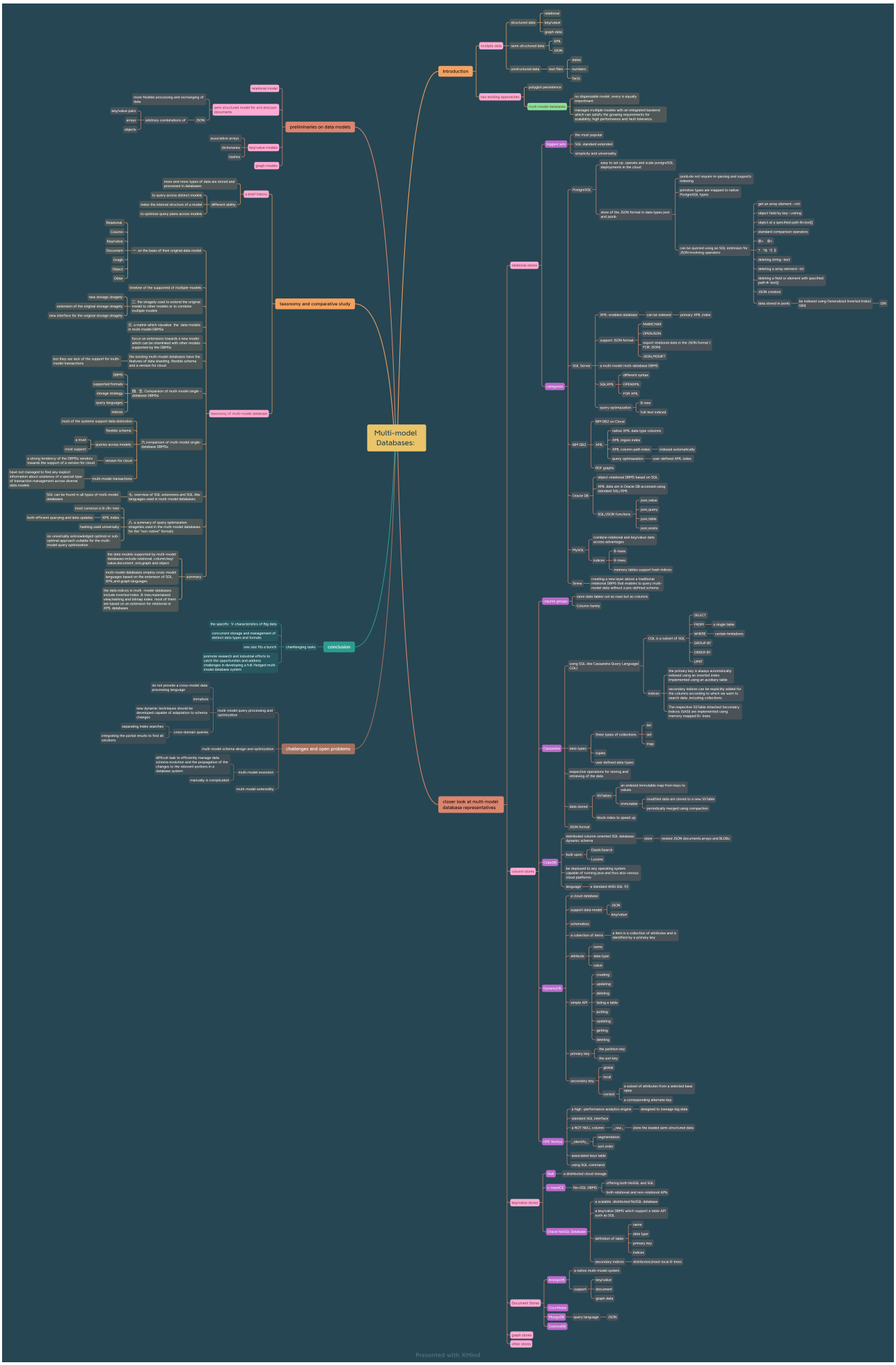
以下是我在阅读论文的过程中画的思维导图,放大可查看详情:



A New Journey to Handle the Variety of Data

10205101530-赵晗瑜-论文笔记和总结

以下是我读完后对论文的总结与体悟：

首先，什么是**muti-model**呢？

文章在开头处便给出了定义:

> *It is worthy to mention the difference between multi-modal databases and multimodel databases. The former means the multi-media databases where the types of data may include speech, images, videos, handwritten text and fingerprints. But the latter stands for a system to manage data with different models such as relational, tree, graph and*

> *object models. The scope of this survey restricts to the latter one, i.e. multi-model databases.*

multi-model即为管理有着不同模型的数据的系统，像是关系模型，树模型，图模型和对象模型等，而这篇论文即针对这一方面进行阐述。

那么，现在常用的存储的逻辑模型有哪几种呢？

文中给出了四种，分别是：*relation model ,semi-structured Model for JSON and XML Documents, key/value model ,Graph model*，文章先是简要地概述了这四种常见的模型（并且这些模型是被大多数multi-model database所支持的），然后就开始从很多不同的角度对*multi-model DBMSs*进行了分类和对比，分类的维度有很多种，比如文中的表格有按照对*multi-model*的扩展策略来分、按照*multi-model DBMSs*支持的数据模型来分、按照对*SQL*扩展的支持来划分等等，从多个角度对*multi-model DBMSs*的应用和可扩展性、语言特性、是否支持云端存储、索引、支持的数据格式、存储的策略等多个角度进行了对比分析，最后得出了这样的结论：

> *The data models supported by multi-model databases include relational, column, key/value, document, XML, graph, and object.*
> *—Multi-model databases employ cross-model languages based on the extension of SQL, XML, and graph languages.*
> *—The data indices in multi-model databases include inverted index, B-tree, materialized view, hashing, and bitmap index. Most of them are based on an extension for relational or XML databases.*
>
> *The existing multi-model databases have the features of data sharding, flexible schema, and a version for cloud. But they still lack of the support for multi-model transactions.*

可见，能够支持*multi-model*的数据模型覆盖广泛，*multi-model*所使用的语言也是针对*single model*语言的扩展，数据的索引也是基于关系型和*XML*数据库的扩展，可以看出，*multi-model*已经有很好的闪光点，像是*data sharding(*数据碎片）*,flexible schema, a version for cloud*，但是现有的*multi-model database*缺乏对于*multi-model*事物（*transaction*）的支持。

随后文章用大量的篇幅对*multi-database*进行了多维度的分类:

第一种分类是根据原本的数据模型来分类：

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

从上图和文中的分析可以得出以下结论：

- multi-model database可以对关系数据库，所有四种非关系数据库和其他诸如对象数据库进行相应的拓展

第二种分类是按照多模型拓展的策略来分的，拓展的方法分别是：

- 适用于新模型的新的存储策略
- 对原存储策略的拓展
- 在不改变原策略的基础上提供一个针对新策略的接口

针对这种分类，我们可以得出：第一种分类策略的代表支持XML并且利用XML进行高效存储和查询的数据库；第二种分类策略的代表是ArangoDB，这是一种document数据库；第三种分类策略的代表是Sinew，它在关系存储策略之上开辟了一个新的层

第三种分类是按照在**multi-model**数据库中支持的数据模型有哪些来分类的

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	√			√	√			√	*

从上图和文中的分析可以得出以下结论：

从图中可以观察到所有的Document都支持JSON数据模型，同时，文中着重强调了RDF模型，指出RDF模型对于研究multi-model有很大作用，而支持RDF存储作为系统的一部分的DBMSs是真正的multi-model，文中还给出了针对RDF支持的系统的研究链接。

第四种分类是从支持的格式、存储策略、查询语言、索引等多方面对**multi-model single database DBMSs**的对比

Type	DBMS	Supported formats	Storage strategy	Query languages	Indices
Relational	PostgreSQL	relational, key/value, JSON, XML	relational tables – text or binary format + indices	extended SQL	inverted
	SQL Server	relational, XML, JSON, ...	text, relational tables	extended SQL	B-tree, full-text
	IBM DB2	relational, XML	native XML type	extended SQL/XML	XML paths / B+ tree, full-text
	Oracle DB	relational, XML, JSON, RDF	relational	SQL/XML or JSO extension of SQL	bitmap, B+tree, function-based, `XMLIndex`
	Oracle MySQL	relational, key/value	relational	SQL, memcached API	B-tree
	Sinew	relational, key/value, nested document, ...	logically a universal relation, physically partially materialized	SQL	–
Column	Cassandra	text, user-defined type	sparse tables	SQL-like CQL	inverted, B+ tree
	CrateDB	relational, JSON, BLOB, arrays	columnar store based on Lucene and Elasticsearch	SQL	Lucene
	DynamoDB	key/value, document (JSON)	column store	simple API (get/put/update) + simple queries over indices	hashing
	HPE Vertica	JSON, CSV	flex tables + map	SQL-like	for materialized data
Key/value	Riak	key/value, XML, JSON	key/value pairs in buckets	Solr	Solr
	c-treeACE	key/value + SQL API	record-oriented ISAM	SQL	ISAM
	Oracle NoSQL DB	key/value, (hierarchical) table API, RDF	key/value	SQL	B-tree
Document	ArangoDB	key/value, document, graph	document store allowing references	SQL-like AQL	mainly hash (eventually unique or sparse)
	Couchbase	key/value, document, distributed cache	document store + append-only write	SQL-based N_1QL	B+tree, B+trie
	MongoDB	document, graph	BSON format + indices	JSON-based query language	B-tree, hashed, geospatial
	Cosmos DB	document, key-value, graph, column	JSON format + indices	SQL-like query language	forward and inverted index mapping
	MarkLogic	XML, JSON, binary, text, ...	storing like hierarchical XML data	XPath, XQuery, SQL-like	inverted + native XML

Type	DBMS	Supported formats	Storage strategy	Query languages	Indices
Graph	OrientDB	graph, document, key/value, object	key/value pairs + object-oriented links	Gremlin, extended SQL	SB-tree, extendible hashing, Lucene
Object	Caché	object, SQL or multi-dimensional, document (JSON, XML) API	multi-dimensional arrays	SQL with object extensions	bitmap, bit-slice, standard
Other	NuoDB	relational	key/value	–	–
	Redis	flat lists, sets, hash tables	key/value	–	–
	Aerospike	key/value	key/value	–	–

上图对系统的关键特性进行了总结，这两张表的侧重点在于：

- 支持的数据格式
- 处理多样的数据时采取的存储策略
- 支持的查询语言是什么以及为了查询优化支持什么样的索引，而这个点在第七种分类中有更为详细的介绍

第五种分类是从数据分布、**schema**的灵活性、跨模型的查询、云版本、**multi-model**事务等多方面对**multi-model single database DBMSs**的对比

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	√	√	–	√	–

从上图和文中的分析可以得出以下结论：

- 绝大多数系统都支持data distribution
- 灵活的schema并不是所有系统都有，但是对于NoSQL来说是
- 跨multi-model的查询是multi-model database 所必须的，以此所有系统都提供了对于此的支持
- 但是，有一个很大的问题就是对于跨多模的事物管理的处理，这是一个非常棘手的问题

第六种分类是从**multi-model databases**能够支持的**SQL**扩展的语言出发进行的分类

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)

从上图和文中的分析可以得出以下结论：

- 在所有类型的multi-model database中都可以看到SQL语言的支持，可以看出SQL应用的普遍

第七种分类是从**multi-model databases**中的查询优化策略出发进行的分类

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

从上图和文中分析可以得到以下结论：

- 最普遍的优化查询的策略是B+/B-树索引
- 与此同时，hashing也是在不同种类的DBMSs使用的普遍策略
- 但是，总的来说没有一个通用的multi-model查询优化的合适的办法

综上所述，针对文章从多个维度对**multi-model databases**进行的讨论，可以得出以下结论：

- 支持multi-model databases的数据模型包括key/value,column,relational,document,XML,graph和object
- multi-model databases应用了在SQL、XML、graph language基础上拓展的跨模型的语言
- 在multi-model databases中使用的数据索引包括inverted index, B-tree,materialized view,hashing和bitmap index。而这些中的大多数都支持对于relational或XML数据库的扩展

然后，文章对不同的**multi-model databases**进行了详细的介绍，重点介绍每一类代表的关键闪光点

Relational Stores

文章第一部分介绍的便是关系数据库存储——multi-model system的最大集合之一。

文中提到了关系型数据的拓展性、通用性和易用性几个特点。

文中首先介绍了PosgreSQL——一个非常经典和具有年代感的关系型DBMS，此处提到了json和jsonb这类multi-model数据组织类型，并简单对比了两者的优缺点：

- json插入快但取用慢
- jsonb作为json的解析二进制格式，插入慢但取用快

此处还介绍了存储在PosgreSQL中的数据类型可以通过一种SQL扩展的语言进行查询得到，如-> int,->string,#>text,@>,@<,?,?&,?|,||,-int,#-text[]等。

下图便是一个查询的例子：

```
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"}

沿着jsonb，本小节还介绍了其索引相关的知识，简单提到了**GIN**以及B-tree，特别是**GIN**索引结构的两种情况：posting tree和posting list，而这与HashMap的存储策略非常类似。

接着，文章又介绍了SQL Server、DB2、Oracle、MySQL对Relational Data和key/value Data的multi-model操作支持，对XML、JSON数据类型的支持以及对应的数据操作、索引机制。

最后文章简单介绍了基于以上传统Relational database所研发的没有固定数据模型的Sinew数据库，它可以支持上述提到的多种数据模型的灵活操作。

Column Stores

在介绍Column Store之前，文章先对“column store”进行了解读：

- 单列存储：一个列里面只存储单个属性
- 列簇存储：一个列里面存储多个属性，以列簇的形式呈现

该部分主要对列簇式存储数据库进行了探讨

文章首先介绍了Cassandra并提出了以下几个特性：

- 存储机制

  内部的数据存储在SSTable中，SStable会进一步划分为block，SStable是immutable的，因此修改的数据会存储到一个新的SStable并且使用compaction策略进行合并。

  同时，其同样支持json格式，但需要提前指定schema，如下图是一个json schema从里向外层层嵌套预定义以及对应的数据插入例子：

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

- 查询语言

  可以把它当作SQL的子集，支持SQL常见操作。

  但是在针对列簇数据进行where查询和索引时会受到一些限制

- 索引机制

  primary key:using an inverted index

secondary indices:使用B+ tree以支持范围查询

虽然合适的索引可以增加查询效率，但是并不推荐针对频繁修改、删除、大范围查询的数据表进行索引的建立

然后，文章简单介绍了Column-oriented数据库CrateDb查询层面的Lucence索引支持，这是一种动态的schema数据库，

然后，文章介绍的DynamoDB可作为云部署的、具有动态Schema、分布式的高性能数据库，它支持json和key/value的灵活的数据模型，同时还介绍了在Dynamo中一个item由哪些部分组成。

然后，介绍了Dynamo中使用哪些API对item进行各种操作。

然后，介绍了在Dynamo中使用的两种primary key：

- **partition key**
- **sort key**

最后文章提到了HPE高性能分布式数据库中的virtual column和real column之间的转换，如virtual column可以转换成real column以获得更高的查询性能。

Dynamo和HPE都提到了相应的SQL操作、索引、针对json和其他数据模型的支持。

Key/value Stores

键值数据库是最简单的非关系数据库，只需要通过id进行API的调用来实现数据的存储和获取。

Riak作为经典的kv数据库，其基于Sorl的集成提供了高效的索引机制，另外其还具有分布式、高可用、高扩展等特性。

c-treeACE同时支持SQL和NoSQL两种操作模式。

最后介绍的Oracle NoSQL除了具备json、index外，还添加了RDF、Graph module、Child table等机制

Document Stores

文档数据库可以看作键值数据库的增强模式，其索引与kv数据库类似。综合来看，文档数据库更加贴近multi-model database。

文章首先介绍了ArangDB这一原生多模型、开源的数据库，其强大的AQL查询支持在一个单个查询中混合使用kv、doc和graph三种数据模型。

此外，其具有着和json类似的文档存储组织的存储方式；

同时，文章还介绍了其文档之间的关系图模型和对应图算法的操作，如图遍历和求最短路径；

同时，文章还介绍了Hashing和SkipList索引方式及其之间的差别：

- Hashing不支持范围查询而SkipList支持

随后，文章介绍了Couchbase这一基于内存的高性能文档数据库，其特性总结如下：

- 对于json存储格式的数据无需预先创建schema，这使得其数据操作更加flexible
- 其针对频繁操作的document进行了缓存，可以提供更加高效的服务响应
- 其基于SQL的查询语言N_1QL提供了对json、kv和geographical data的方便访问
- 其提供了B+ tree和B+ trie两种索引，其中B+ trie在tree上构造更短的depth来获得更好的查询效率

随后，文章介绍了MongoDB这一最流行的文档数据库，文章介绍了其flexible schema、json数据查询、Manual references和DBRefs references、BSON作为JSON的二进制高效表示等主要特性。

随后文章介绍了Cosmos DB和MarkLogic这两款数据库，其除了最基本的文档数据库所具有的特性之外，还具有着其各自的特性：

- Cosmos DB的（constistent/lazy/none）索引更新策略
- MarkLogic的基于树状存储结构为XML和JSON提供的统一索引管理方式
- MarkLogic还提供了kv互换下的两种范围索引数据结构：kv和vk型

Gragh Stores

图数据库以图论为数学基础，各种复杂的算法都集中在图结构中

文章针对OrientDB这一图数据库的代表从存储结构、数据操作、索引等关键点进行了分析。

OrientDB支持针对云环境的支持，这意味着其能够提供DAAS平台层面的服务。即其支持类型继承以及类间关系的建立如引用。

文章介绍了两种类间关系模式：

1. 引用（Refreneced）

   引用通过存储对应的物理连接来维持关系

2. 嵌入（Embedded）

   嵌入通过直接将实际数据存储在对应的数据对象下形成共生依赖

下图是在OrientDB中存储multi-model data的一个例子：

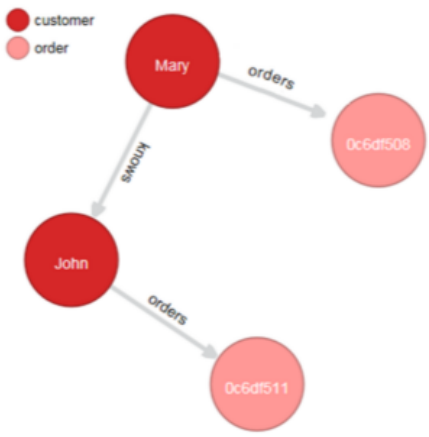
```
CREATE CLASS orderline EXTENDS V
CREATE PROPERTY orderline.product_no STRING
CREATE PROPERTY orderline.product_name STRING
CREATE PROPERTY orderline.price FLOAT

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")
```

Other Stores

除了上述几种主流的数据存储模型之外，还存在一些潜在的多模数据存储模型，如object-model、multi-use-case stores

- object-model

  从面向对象的角度来看，其可以存储任何类型，因为“万物皆对象”。

  object-model data model的代表是InterSystems Cache，其所支持的schemaless、schema-based的存储策略对于数据定义操作提供了更大的灵活性

- Multi-use-case

  multi-use-case的核心思想是：“one size fits all”

接着，文章说了一些挑战和问题

文章将遇到的挑战和问题划分成了四类：

- multi-model查询处理和优化
- multi-model schema的设计和优化
- multi-model的演变问题
- multi-model的扩展性问题

最后：

文章对比了各种数据模型的典型的数据库产品，并重点针对数据存储模型、查询操作语言、索引、可扩展性等特性进行阐述，但存在一个问题是：

我们还没有一个较为成熟的、可针对多种数据模型提供一个统一操作接口的数据库技术

而在大数据的背景下，针对volume、velocity、variety、veracity、value 5v问题急需一个统一的数据库技术来操作多种数据库模型，打破各个数据模型之间的壁垒，进而才能提高数据处理的效率、降低数据处理的难度、节约企业数据处理的成本。

我非常相信多年以后一定能有这些问题的解决办法，或许某一天就会出现一个这样统一的数据库技术针对各种数据模型进行统一的处理。

希望早日到达“one size fits a bunch”的境地。