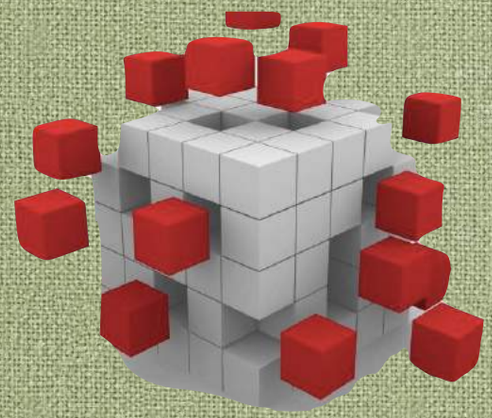
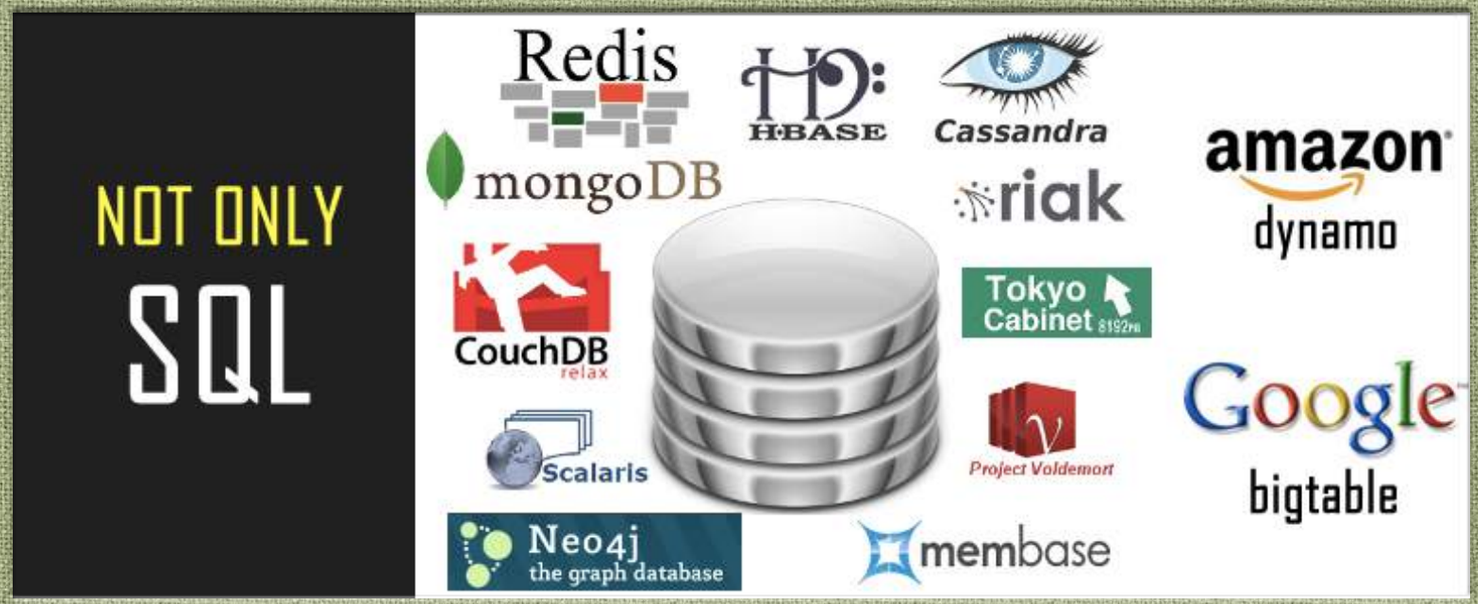


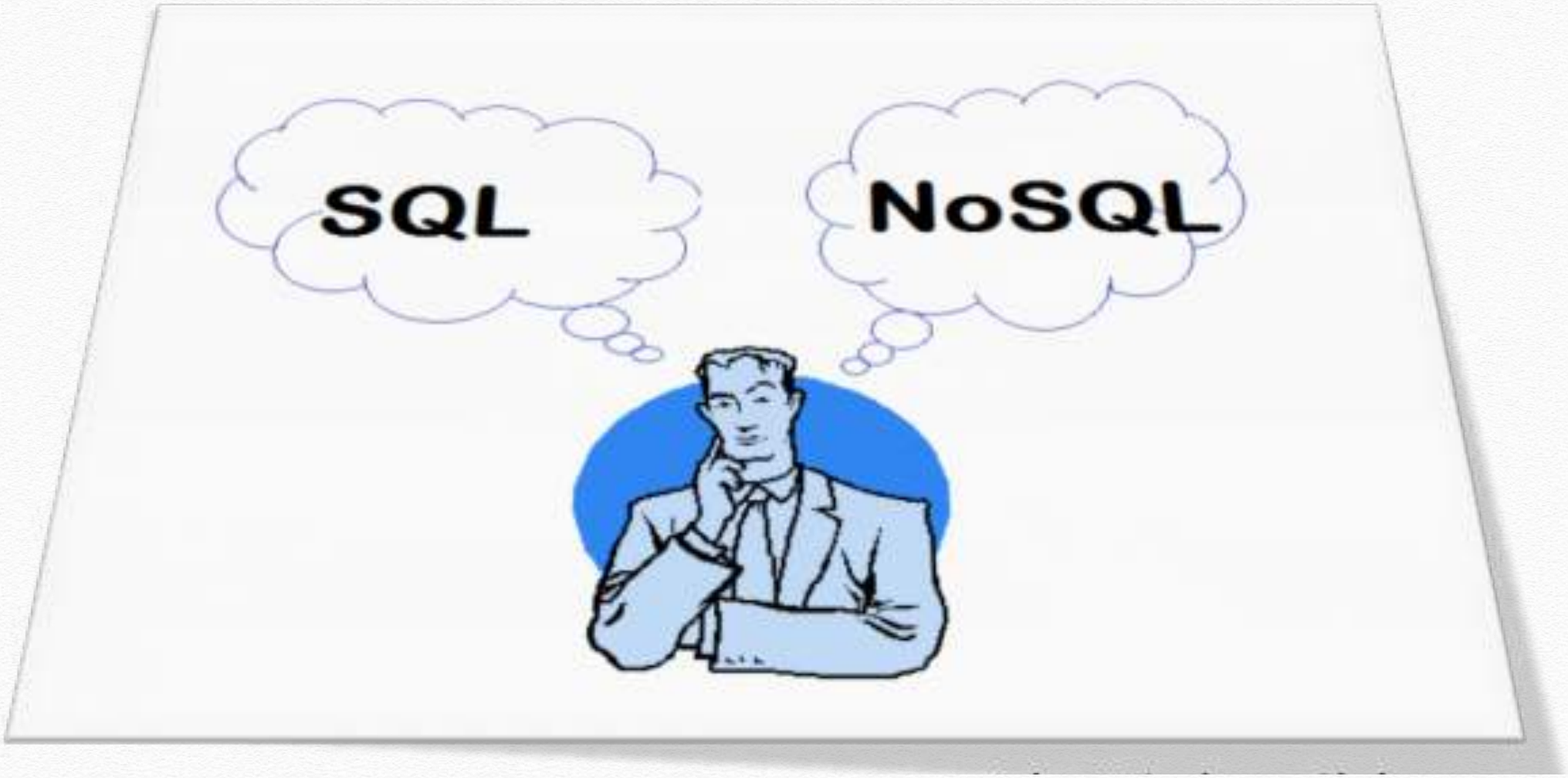
Not Only SQL



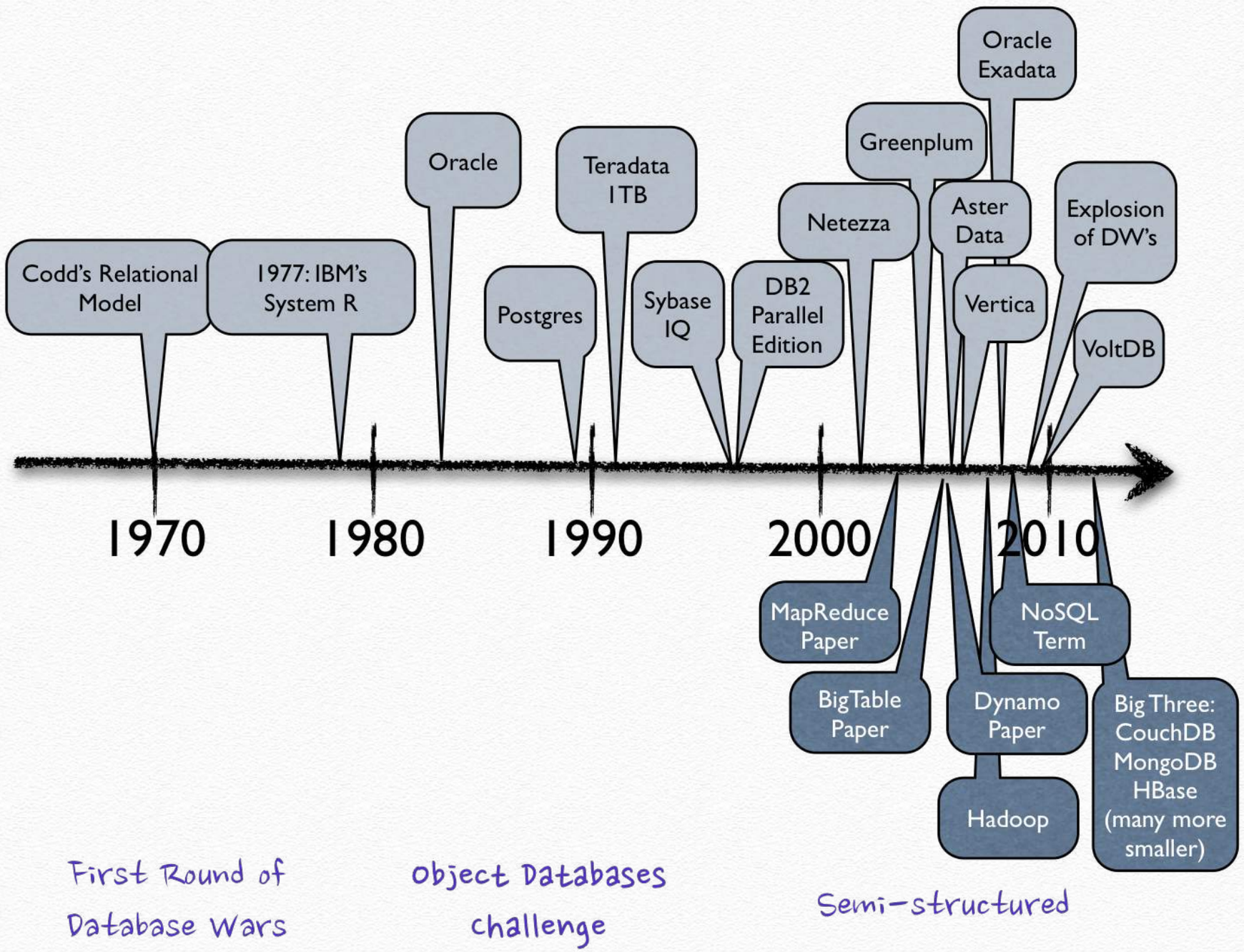
NoSQL Databases

27/05/2020 - Big Data 2020

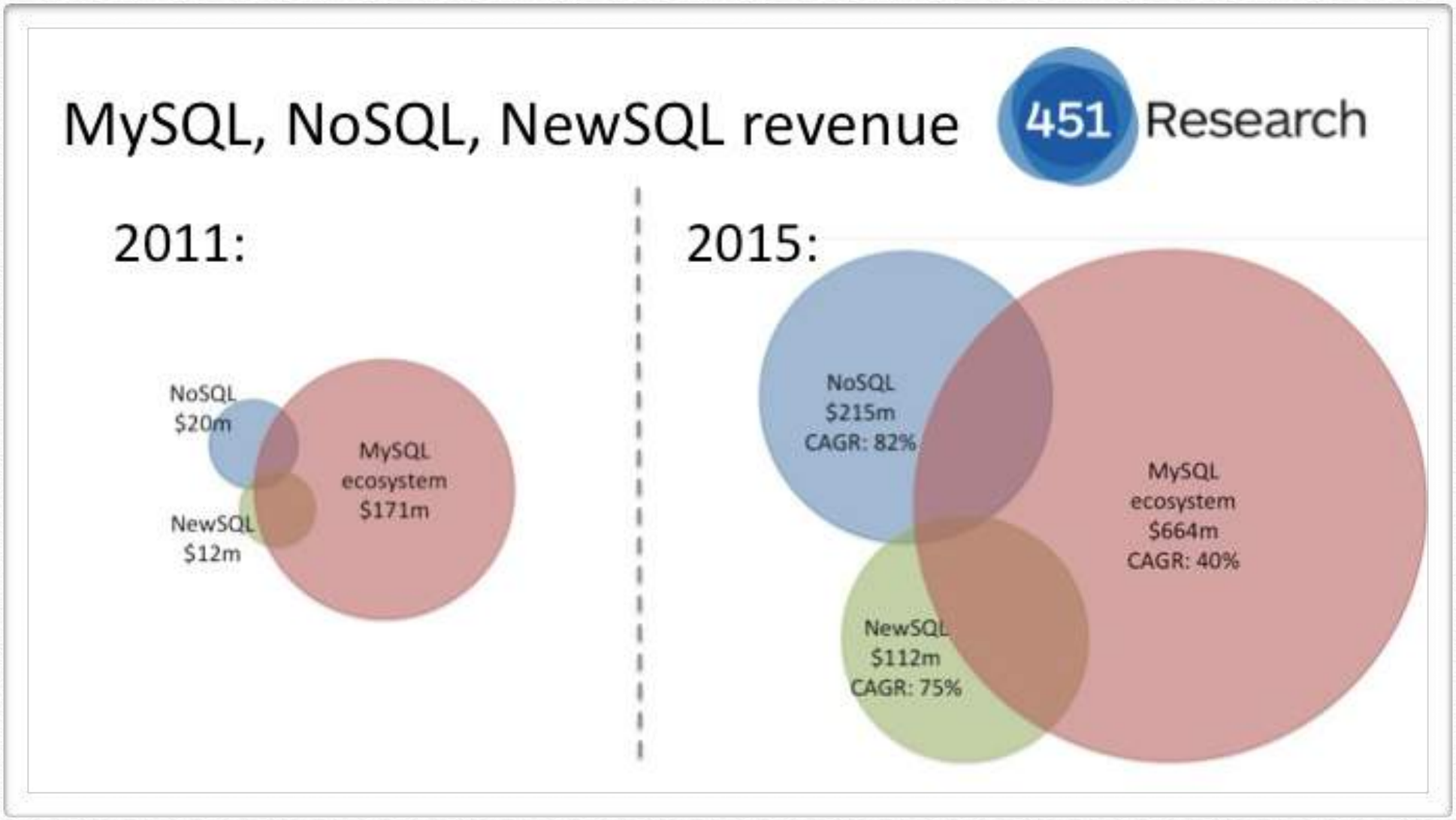
What's the matter



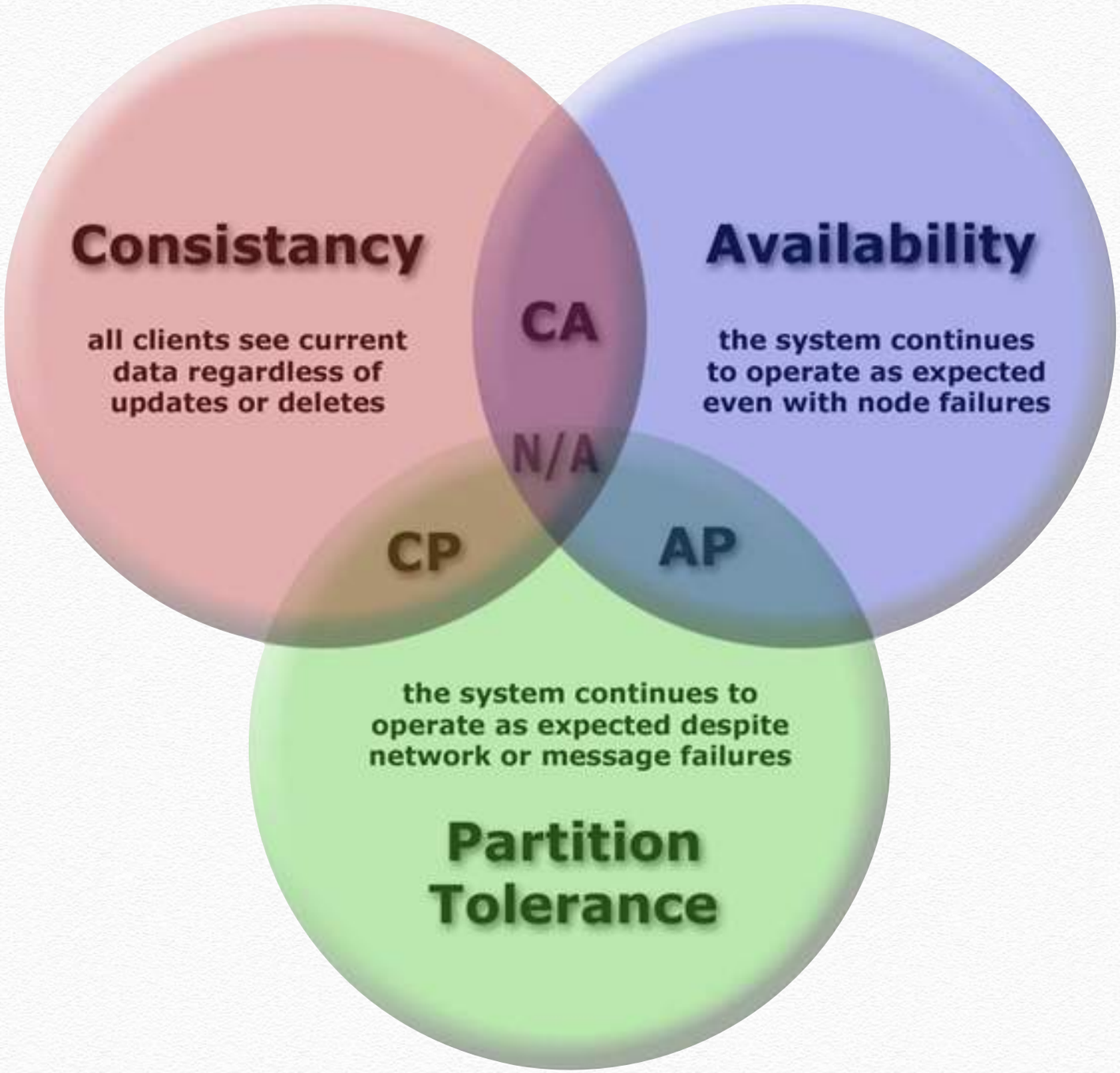
What's the matter



What's the matter



CAP Theorem


Consistancy
all clients see current data regardless of updates or deletes
Availability
the system continues to operate as expected even with node failures
CA
N/A
CP
AP
the system continues to operate as expected despite network or message failures
Partition Tolerance

How to save this programming language user object in a database?

```
{
  "id" : 1234,
  "name" : {
    "first" : "foo",
    "last" : "bar"
  },
  "topics": [
    "skating",
    "music"
  ]
}
```

Relational databases – tables

- data are stored in **tables** with typed **columns**
- all records in a table are **homogenously structured** and have the same columns and data types
- tables are **flat** (no hierchical data in a table)
- columns have primitive data types: **multi-valued data** are **not supported**

Relational databases – schemas

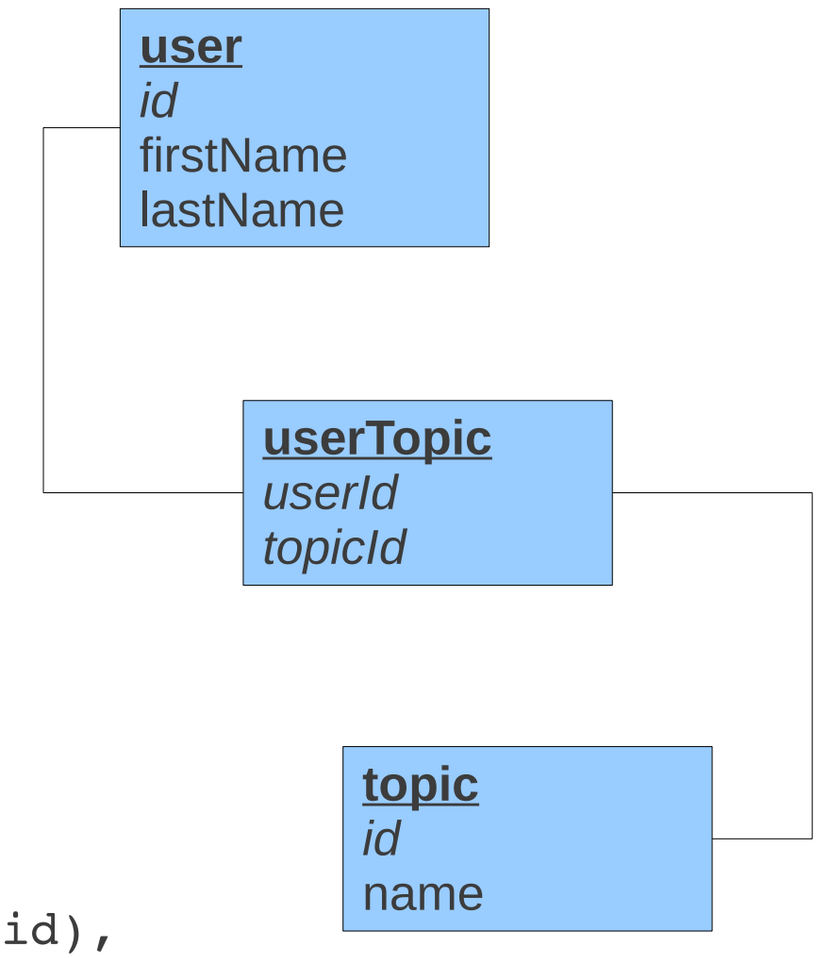
- relational databases have a **schema** that defines which tables, columns etc. there are
- users are **required to define the schema** elements before data can be stored
- inserted **data must match the schema** or the database will reject it

Saving the user object in a relational database

- we cannot store the object as it is in a relational table, we must first **normalise**
- for the example, we end up with **3 database tables** (user, topic**,** and an n:m mapping table between them)
- note that the object in the programming language now has **a different schema** than we have in the database

Schema we may have come to

```
CREATE TABLE `user` (
  id INTEGER NOT NULL,
  firstName VARCHAR(40) NOT NULL,
  lastName VARCHAR(40) NOT NULL,
  PRIMARY KEY(id)
);
CREATE TABLE `topic` (
  id INTEGER NOT NULL auto_increment,
  name VARCHAR(40) NOT NULL,
  PRIMARY KEY(id),
  UNIQUE KEY(name)
);
CREATE TABLE `userTopic` (
  userId INTEGER NOT NULL,
  topicId INTEGER NOT NULL,
  PRIMARY KEY(userId, topicId),
  FOREIGN KEY(userId) REFERENCES user(id),
  FOREIGN KEY(topicId) REFERENCES topic(id)
);
```



Now we can save the user object

```
BEGIN;

-- insert the user
INSERT INTO `user` (id, firstName, lastName)
VALUES (1234, "foo", "bar");

-- insert topics (must ignore duplicate keys)
INSERT INTO `topic` (name) VALUES ("skating");
INSERT INTO `topic` (name) VALUES ("music");

-- insert user-to-topics mapping
INSERT INTO `userTopic` (userId, topicId)
SELECT 1234, id FROM `topic`
WHERE name IN ("skating", "music");

COMMIT;
```

Relational databases criticisms (I)

- lots of **new databases have emerged** in the past few years, often because...
  - ...**object-relational mapping can be complex** or costly
  - ...relational databases do not play well with **dynamically structured data** and **often-varying schemas**

Relational databases criticisms (II)

- lots of **new databases have emerged** in the past few years, often because...
  - ...**overhead** of SQL parsing and full-blown query engines **may be significant for simple access patterns** (primary key access, BLOB storage etc.)
  - ...**scaling to many servers** with the **ACID guarantees** provided by relational databases **is hard**

NoSQL and NewSQL databases

- many of the recent databases are labelled
  - **NoSQL (the non-relational ones)** or
  - **NewSQL (the relational ones)**
- because they **provide alternative solutions** for some of the mentioned problems
- especially the NoSQL ones often **sacrifice features** that relational databases have in their DNA

NoSQL database characteristics

- NoSQL databases have multiple (but not necessarily all) of these characteristics:
  - **non-relational**
  - **schema-free**
  - **open source**
  - **simple APIs**
- several, but not all of them, are **distributed** and **eventually consistent**

Simple APIs

- NoSQL databases often **provide simple interfaces** to store and query data
- in many cases, the APIs offer access to **low-level data manipulation and selection** methods
- queries capabilities are often limited so **queries can be expressed in a simple way**
- SQL is not widely used

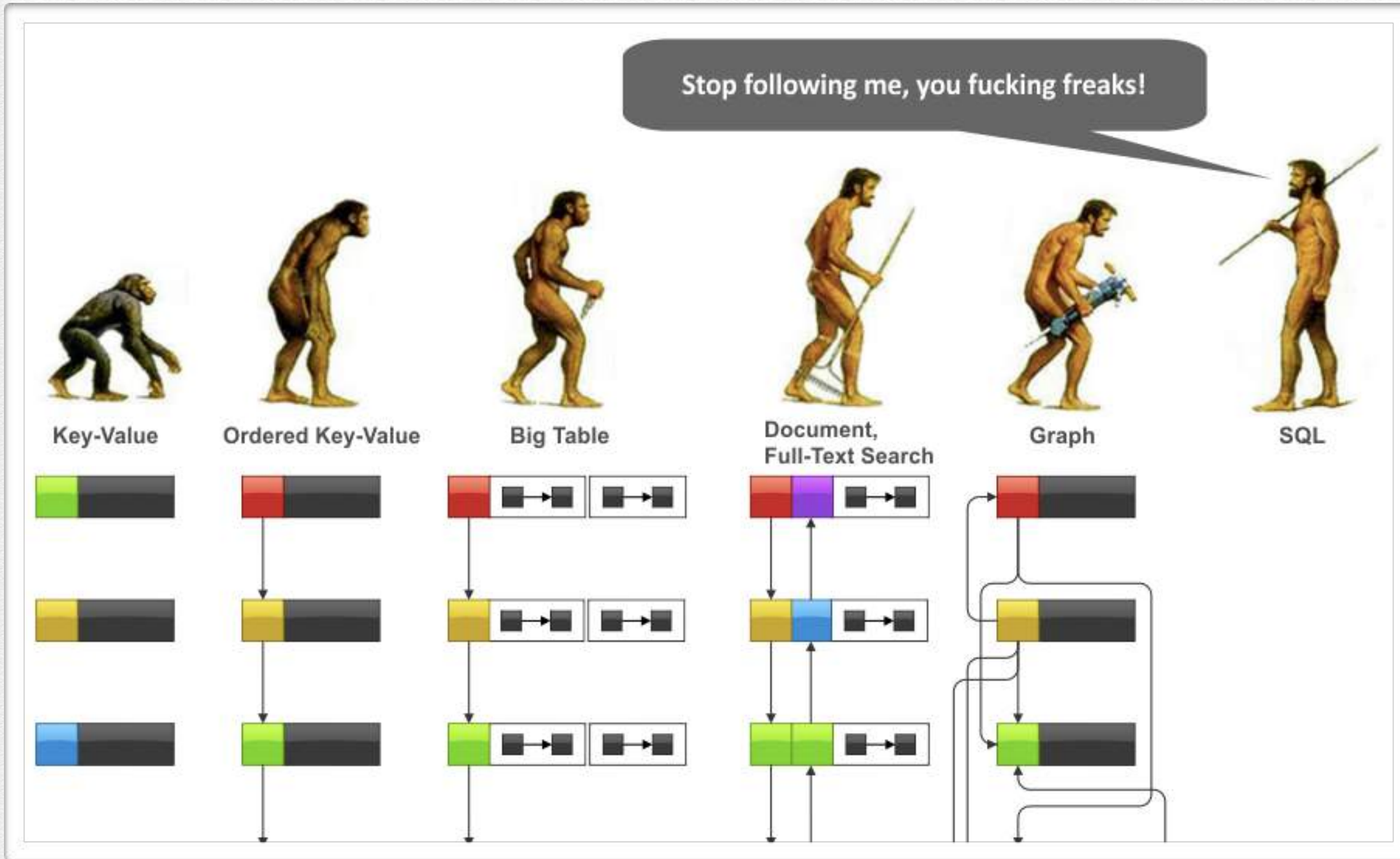
Simple APIs

- many NoSQL databases have simple text-based protocols or **HTTP REST APIs with JSON inside**
- databases with HTTP APIs are **web-enabled** and can be run as **internet-facing services**
- several vendors provide **database-as-a-service** offers

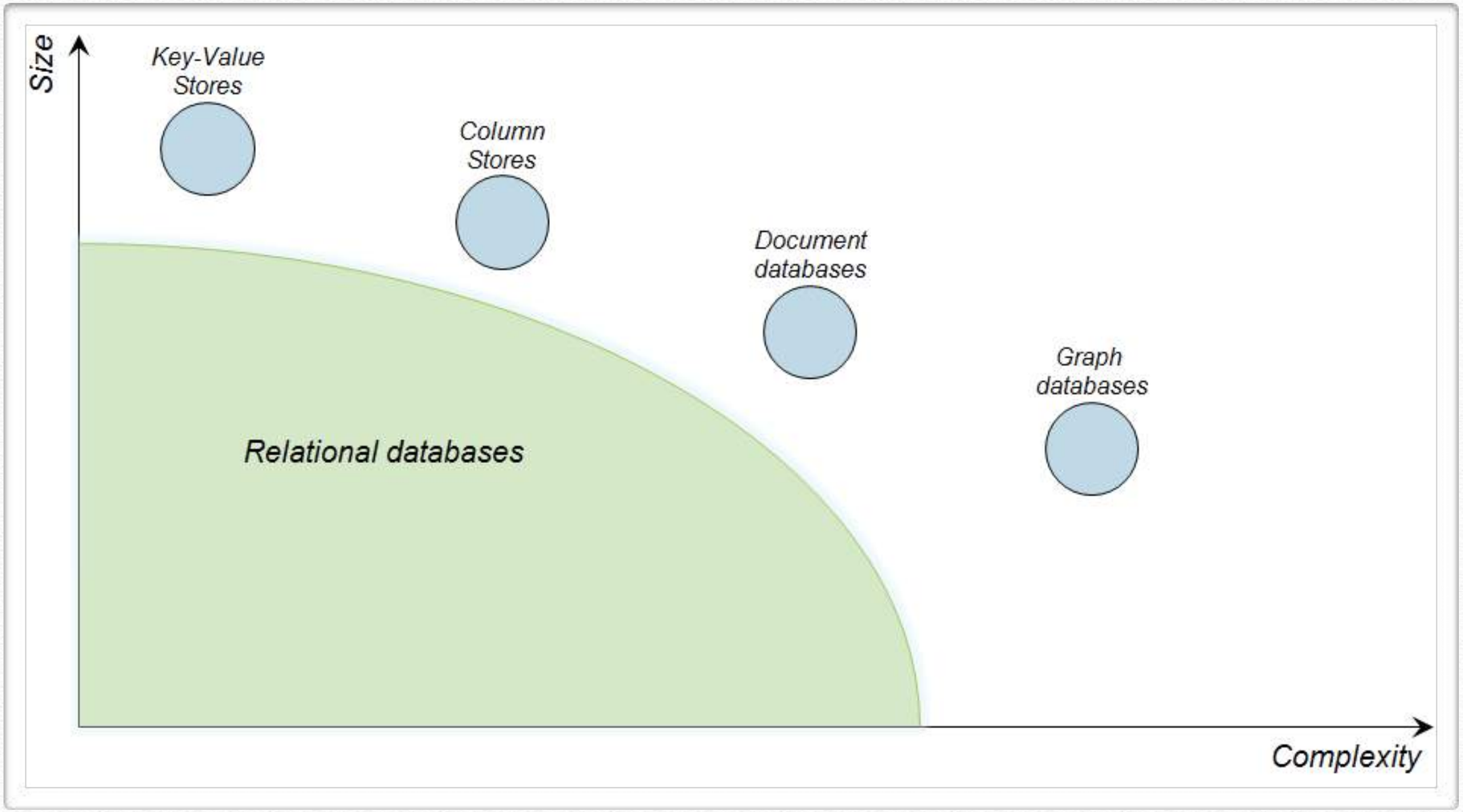
Distributed

- several NoSQL databases (not all!) can be run in a **distributed** fashion, **providing auto-scalability** and **failover capabilities**
- in a distributed setup, ACID features are often sacrificed for scalability and throughput
- **replication** between distributed nodes is often **lazy**, meaning the database is **eventually consistent**

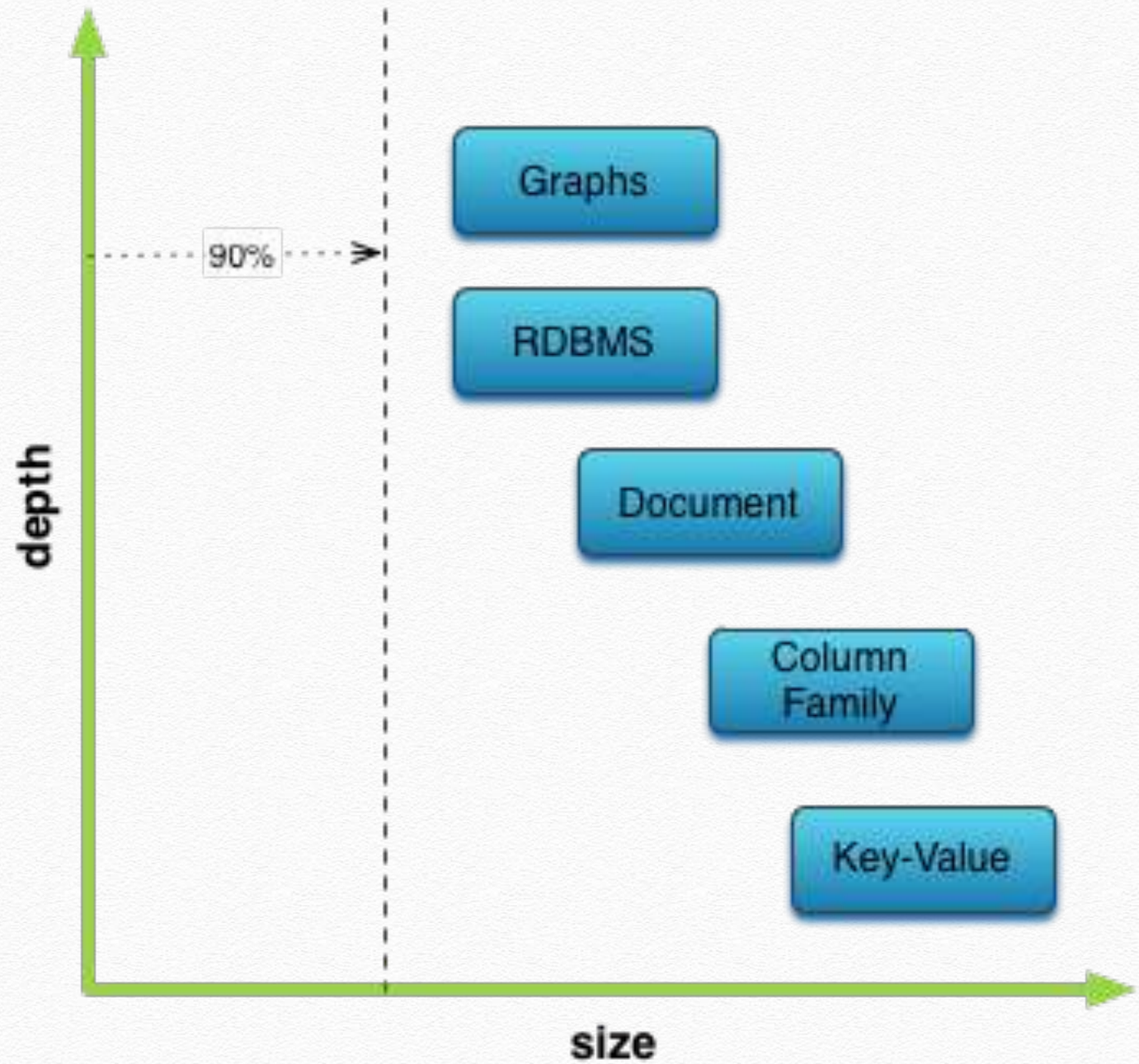
NoSQL Family



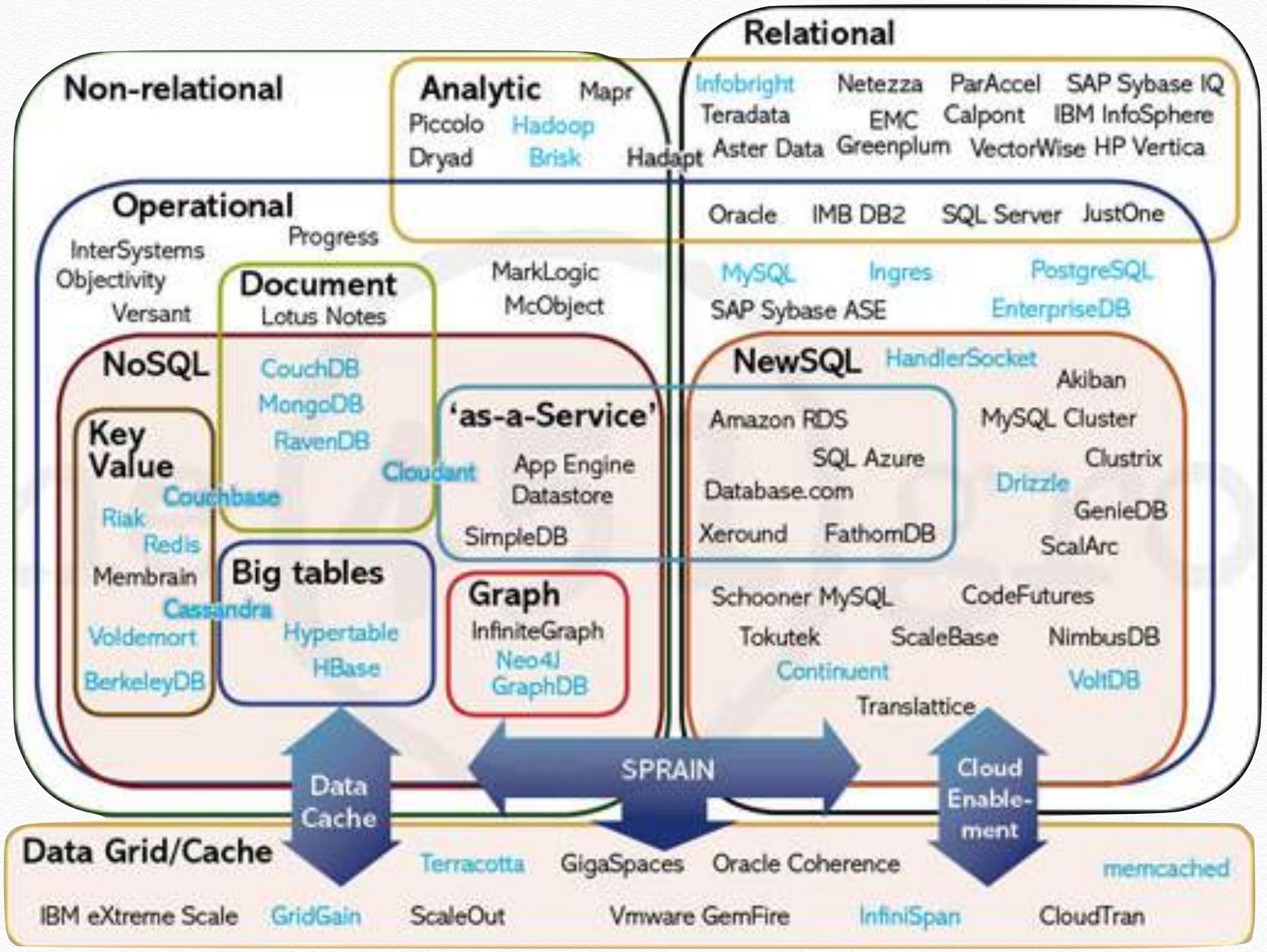
Which one ?


Size
Key-Value Stores
Column Stores
Document databases
Graph databases
Relational databases
Complexity

Which one ?


Graphs
90%
RDBMS
depth
Document
Column
Family
Key-Value
size

NoSQL Family



Key-value stores – principle

- in a key-value store, a **value** is mapped to a **unique key**
- to **store** data, supply both key and value:
  `> store.set("user-1234", "...");`
- to **retrieve a value**, supply its key:
  `> value = store.get("user-1234");`
- keys are organised in **databases**, **buckets**, **keyspaces** etc.

Key-value stores – values

- key-value stores treat value data as **indivisible BLOBs** by default (some operations will treat values as numeric)
- for the store, the values **do not have a known structure** and will **not be validated**
- as no structure is known, values can only be queried via their keys, not by values or sub-parts of values

Key-value stores – basic operations

- key-value stores **are very efficient** for basic operations on keys, such as **set**, **get**, **del**, **replace**, **incr**, **decr**
- many stores also provide **automatic ttl-based expiration of values** (useful for caches)
- some provide **key enumeration** to retrieve the full or a restricted list of keys

key-values examples

key-values examples

redis

Project Voldemort
A distributed database.
zoie · bobo · cleo · decomposer · norbert

https://www.youtube.com/watch?v=Hbt56gFj998

https://www.project-voldemort.com/voldemort/

Amazon DynamoDB

Amazon Web Services

Compute

- **EC2** — Virtual Servers in the Cloud
- **Lambda** — Run Code in Response to Events
- **EC2 Container Service** — Run and Manage Docker Containers

Storage & Content Delivery

- **S3** — Scalable Storage in the Cloud
- **Elastic File System** PREVIEW — Fully Managed File System for EC2
- **Storage Gateway** — Integrates On-Premises IT Environments with Cloud Storage
- **Glacier** — Archive Storage in the Cloud
- **CloudFront** — Global Content Delivery Network

Database

- **RDS** — MySQL, Postgres, Oracle, SQL Server, and Amazon Aurora
- **DynamoDB** — Predictable and Scalable NoSQL Data Store
- **ElastiCache** — In-Memory Cache
- **Redshift** — Managed Petabyte-Scale Data Warehouse Service

Networking

Administration & Security

- **Directory Service** — Managed Directories in the Cloud
- **Identity & Access Management** — Access Control and Key Management
- **Trusted Advisor** — AWS Cloud Optimization Expert
- **CloudTrail** — User Activity and Change Tracking
- **Config** — Resource Configurations and Inventory
- **CloudWatch** — Resource and Application Monitoring

Deployment & Management

- **Elastic Beanstalk** — AWS Application Container
- **OpsWorks** — DevOps Application Management Service
- **CloudFormation** — Templated AWS Resource Creation
- **CodeDeploy** — Automated Deployments

Analytics

- **EMR** — Managed Hadoop Framework
- **Kinesis** — Real-time Processing of Streaming Big Data
- **Data Pipeline** — Orchestration for Data-Driven Workflows
- **Machine Learning** — Build Smart Applications Quickly and Easily

Application Services

- **SQS** — Message Queue Service
- **SWF** — Workflow Service for Coordinating Application Components
- **AppStream** — Low Latency Application Streaming
- **Elastic Transcoder** — Easy-to-use Scalable Media Transcoding
- **SES** — Email Sending Service
- **CloudSearch** — Managed Search Service

Mobile Services

- **Cognito** — User Identity and App Data Synchronization
- **Mobile Analytics** — Understand App Usage Data at Scale
- **SNS** — Push Notification Service

Enterprise Applications

- **WorkSpaces** — Desktops in the Cloud
- **WorkDocs** — Secure Enterprise Storage and Sharing Service
- **WorkMail** PREVIEW — Secure Email and Calendaring Service

Amazon DynamoDB

Amazon DynamoDB Tables

Filter: Explore Table | Create Table | Modify Throughput | Delete Table | Export / Impo

Name	Status	Hash Key
Forum	ACTIVE	Name
ProductCatalog	ACTIVE	Id
Reply	ACTIVE	Id
Thread	ACTIVE	ForumName
my-favorite-movies-table	ACTIVE	name

Amazon DynamoDB

- **Relational Data Model Concepts**
  - In a **relational database**, a **table** has a predefined schema
    - the ***table name***
    - ***primary key***
    - list of its ***column names*** and their data types.
  - All records stored in the table must have the **same set of columns**.

Amazon DynamoDB

Bycicle

id	price	title	type	brand	color	description
205	500	20-Bike-205	Hybrid	C	{black, red}	D1
203	300	10-Bike-203	Road	B	{black, green, red}	D2
202	200	21-Bike-202	Road	A	{black, green}	D3
201	100	18-Bike-201	Road	D	{black, red}	D4
204	400	18-Bike-204	Mountain	B	{red}	D5

RDB

Book

id	price	title	authors	dimensions	ISBN	InPublicatio	pages
102	20	T1	{A1, A2}	8.5 x 11.0 x 0.8	222-2222222222	1	600
103	2000	T2	{A1, A2, A3}	8.5 x 11.0 x 1.5	333-3333333333	0	600
101	2	T3	{A1}	8.5 x 11.0 x 1.5	111-1111111111	1	500

Amazon DynamoDB

Products

id	price	category	title	type	brand	color	descript	authors	dimensi	ISBN	InPublic	pages
205	500	**Bycicle**	20-	Hybrid	C	{black,	D1	**NULL**	**NULL**	**NULL**	**NULL**	**NULL**
203	300	**Bycicle**	10-	Road	B	{black,	D2	**NULL**	**NULL**	**NULL**	**NULL**	**NULL**
202	200	**Bycicle**	21-	Road	A	{black,	D3	**NULL**	**NULL**	**NULL**	**NULL**	**NULL**
201	100	**Bycicle**	18-	Road	D	{black,	D4	**NULL**	**NULL**	**NULL**	**NULL**	**NULL**
204	400	**Bycicle**	18-	Mountai	B	{red}	D5	**NULL**	**NULL**	**NULL**	**NULL**	**NULL**
102	20	**Book**	T1	**NULL**	**NULL**	**NULL**	**NULL**	{A1,	8.5 x	222-22	1	600
103	2000	**Book**	T2	**NULL**	**NULL**	**NULL**	**NULL**	{A1, A2,	8.5 x	333-33	0	600
101	2	**Book**	T3	**NULL**	**NULL**	**NULL**	**NULL**	{A1}	8.5 x	111-11	1	500

RDB

Amazon DynamoDB

- **Data Model Concepts**
  - a **database** is a collection of **tables**;
  - a **table** is a collection of **items**;
  - each **item** is a collection of **attributes**.
- A DynamoDB table **is schema-less**
  - Individual items in a DynamoDB table can have **any number of attributes**
  - Each attribute in an item is a **name-value pair**.
  - An attribute can be **single-valued** or **multi-valued set**.
  - There is a **primary-key** attribute.

Amazon DynamoDB

- For example, consider storing a catalog of products in DynamoDB. You can create a table, **ProductCatalog**, with the **Id** attribute as its ***primary key***.

ProductCatalog (**Id**, ...)

Amazon DynamoDB

- You can store various kinds of **product items** in the table.

```
{
   Id = 201
   ProductName = "18-Bicycle 201"
   Description = "201 description"
   BicycleType = "Road"
   Brand = "Brand-Company A"
   Price = 100
   Gender = "M"
   Color = [ "Red", "Black" ]
   ProductCategory = "Bicycle"
}
```

```
{
   Id = 101
   ProductName = "Book 101 Title"
   ISBN = "111-1111111111"
   Authors = [ "Author 1", "Author 2" ]
   Price = 2
   Dimensions = "8.5 x 11.0 x 0.5"
   PageCount = 500
   InPublication = 1
   ProductCategory = "Book"
}
```

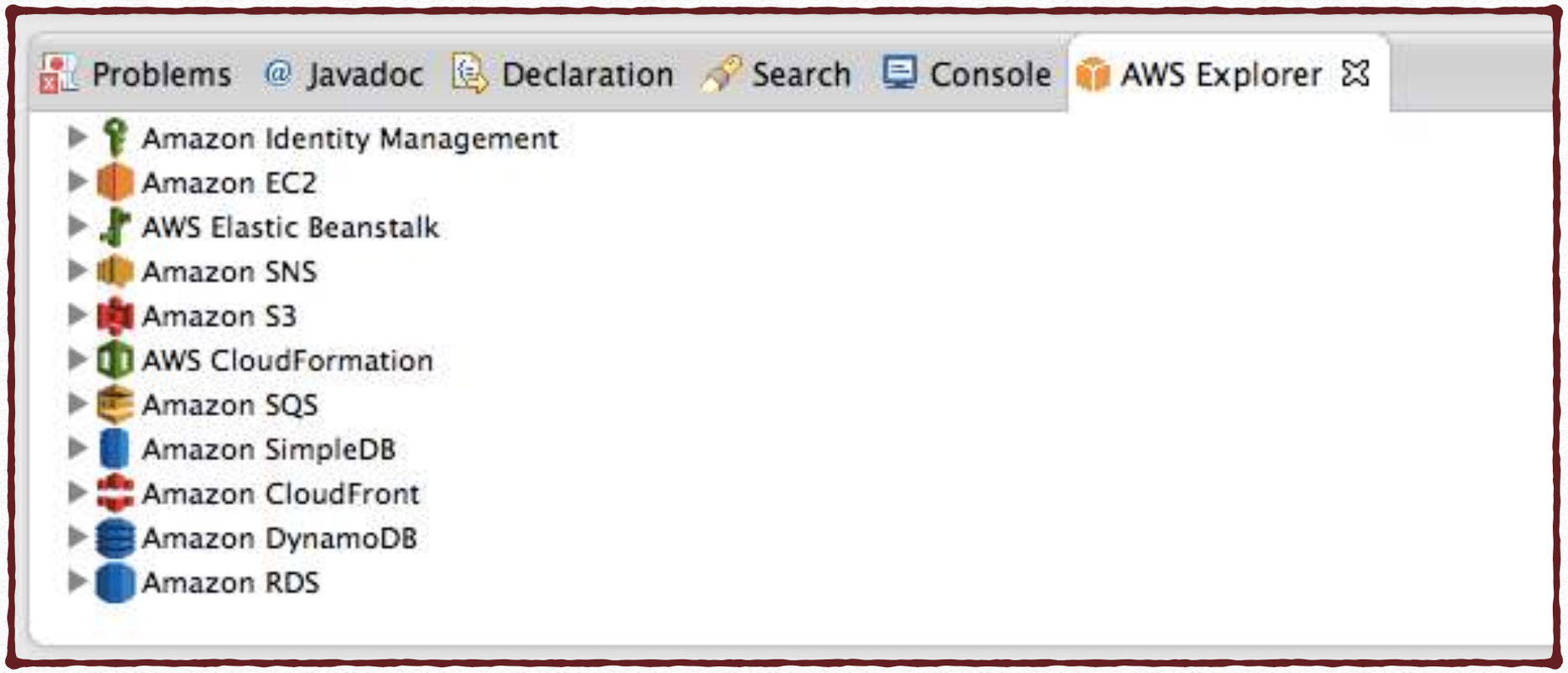
Running Java Examples for DynamoDB

- Download and install the **AWS Toolkit for Eclipse**. This toolkit includes the **AWS SDK** for Java, along with preconfigured templates for building applications.
- From the Eclipse menu, create a new **AWS Java Project**
- You will now need to create a default **credential profiles file.**

AwsCredentials.properties

```
aws_access_key_id = <Your Access Key ID>
aws_secret_access_key = <Your Secret Key>
```

Running Java Examples for DynamoDB



Running Java Examples for DynamoDB

```
static AmazonDynamoDBClient client;

    private static void init() throws Exception {
        /*
         * The ProfileCredentialsProvider will return your [default]
         * credential profile by reading from the credentials file located at
         * (~/.aws/credentials).
         */
        AWSCredentials credentials = null;
        try {
            //credentials = new ProfileCredentialsProvider().getCredentials();
            credentials = new PropertiesCredentials(
                    AmazonDynamoDBSample.class.getResourceAsStream("AwsCredentials.properties"));
        } catch (Exception e) {
            throw new AmazonClientException(
                    "Cannot load the credentials from the credential profiles file. " +
                    "Please make sure that your credentials file is at the correct " +
                    "location (~/.aws/credentials), and is in valid format.",
                    e);
        }
        client = new AmazonDynamoDBClient(credentials);
        Region usWest2 = Region.getRegion(Regions.US_WEST_2);
        client.setRegion(usWest2);
    }
```

Running Java Examples for DynamoDB

```
public class CreateTablesUploadData {

    static AmazonDynamoDBClient client;

    static String productCatalogTableName = “ProductCatalog”;

    . . .

    public static void main(String[] args) throws Exception {

        try {
           init();
           deleteTable(productCatalogTableName);
           waitForTableToBeDeleted(productCatalogTableName);

            // Parameter1: table name
            // Parameter2: reads per second
            // Parameter3: writes per second
            // Parameter4/5: hash key and type
            // Parameter6/7: range key and type (if applicable)

            createTable(productCatalogTableName, 10L, 5L, "Id", “N");
            waitForTableToBecomeAvailable(productCatalogTableName);
            uploadSampleProducts(productCatalogTableName);
        } catch (AmazonServiceException ase) {
            System.err.println("Data load script failed: " + ase);
            ase.printStackTrace();
        }
    }
```

Running Java Examples for DynamoDB

```
private static void createTable(String tableName, long readCapacityUnits, long writeCapacityUnits,
            String hashKeyName, String hashKeyType) {

        createTable(tableName, readCapacityUnits, writeCapacityUnits, hashKeyName,  hashKeyType, null, null);
    }

private static void createTable(String tableName, long readCapacityUnits, long writeCapacityUnits,
            String hashKeyName, String hashKeyType, String rangeKeyName, String rangeKeyType) {

        try {
            System.out.println("Creating table " + tableName);
            ArrayList<KeySchemaElement> ks = new ArrayList<KeySchemaElement>();
            ArrayList<AttributeDefinition> attributeDefinitions = new ArrayList<AttributeDefinition>();

            ks.add(new KeySchemaElement().withAttributeName(hashKeyName).withKeyType(KeyType.HASH));

            attributeDefinitions.add(new AttributeDefinition().withAttributeName(
                      hashKeyName).withAttributeType(hashKeyType));

            if (rangeKeyName != null){
                  ks.add(new KeySchemaElement().withAttributeName(
                      rangeKeyName).withKeyType(KeyType.RANGE));
              attributeDefinitions.add(new AttributeDefinition().withAttributeName(
                      rangeKeyName).withAttributeType(rangeKeyType));
            }
```

Running Java Examples for DynamoDB

```
        CreateTableRequest request = new CreateTableRequest()
            .withTableName(tableName)
            .withKeySchema(ks)
            .withProvisionedThroughput(provisionedthroughput);


        request.setAttributeDefinitions(attributeDefinitions);

        client.createTable(request);

    } catch (AmazonServiceException ase) {
        System.err.println("Failed to create table " + tableName + " " + ase);
    }
}
```

Running Java Examples for DynamoDB

```
private static void uploadSampleProducts(String tableName) {

        try {
            // Add books.
            Map<String, AttributeValue> item = new HashMap<String, AttributeValue>();
            item.put("Id", new AttributeValue().withN("101"));
            item.put("Title", new AttributeValue().withS("Book 101 Title"));
            item.put("ISBN", new AttributeValue().withS("111-1111111111"));
            item.put("Authors", new AttributeValue().withSS(Arrays.asList("Author1")));
            item.put("Price", new AttributeValue().withN("2"));
            item.put("Dimensions", new AttributeValue().withS("8.5 x 11.0 x 0.5"));
            item.put("PageCount", new AttributeValue().withN("500"));
            item.put("InPublication", new AttributeValue().withN("1"));
            item.put("ProductCategory", new AttributeValue().withS("Book"));

            PutItemRequest itemRequest = new PutItemRequest().withTableName(tableName).withItem(item);
            client.putItem(itemRequest);
            item.clear();

            // Add Bicycle.

            item.put("Id", new AttributeValue().withN("201"));
            item.put("Title", new AttributeValue().withS("18-Bike-201")); // Size, followed by some title.
            item.put("Description", new AttributeValue().withS("201 Description"));
            item.put("BicycleType", new AttributeValue().withS("Road"));
            item.put("Brand", new AttributeValue().withS("Mountain A")); // Trek, Specialized.
            item.put("Price", new AttributeValue().withN("100"));
            item.put("Gender", new AttributeValue().withS("M")); // Men's
            item.put("Color", new AttributeValue().withSS(Arrays.asList("Red", "Black")));
            item.put("ProductCategory", new AttributeValue().withS("Bicycle"));

            itemRequest = new PutItemRequest().withTableName(tableName).withItem(item);
            client.putItem(itemRequest);


        }   catch (AmazonServiceException ase) {
            System.err.println("Failed to create item in " + tableName);
        }

    }
```

Running Java Examples for DynamoDB

```
public class ObjectPersistenceQueryScanExample {

    static AmazonDynamoDBClient client;
    . . .

     public static void main(String[] args) throws Exception {
        try {

          init();
            DynamoDBMapper mapper = new DynamoDBMapper(client);

            // Get a book - Id=101
            GetBook(mapper, 101);
        } catch (Throwable t) {
            System.err.println("Error running the ObjectPersistenceQueryScanExample: " + t);
            t.printStackTrace();
        }
    }

    private static void GetBook(DynamoDBMapper mapper, int id) throws Exception {
        System.out.println("GetBook: Get book Id='101' ");
        System.out.println("Book table has no range key attribute, so you Get (but no query).");
        Book book = mapper.load(Book.class, 101);
        System.out.format("Id = %s Title = %s, ISBN = %s %n", book.getId(), book.getTitle(), book.getISBN() );
        }

}
```

Running Java Examples for DynamoDB

```
@DynamoDBTable(tableName="ProductCatalog")
    public static class Book {
        private int id;
        private String title;
        private String ISBN;
        private int price;
        private int pageCount;
        private String productCategory;
        private int inPublication;

        . . .
    }
```

```
@DynamoDBHashKey(attributeName="Id")
public int getId() { return id; }
public void setId(int id) { this.id = id; }

@DynamoDBAttribute(attributeName="Title")
public String getTitle() { return title; }
public void setTitle(String title) { this.title = title; }

@DynamoDBAttribute(attributeName="ISBN")
public String getISBN() { return ISBN; }
public void setISBN(String ISBN) { this.ISBN = ISBN; }

@DynamoDBAttribute(attributeName="Price")
public int getPrice() { return price; }
public void setPrice(int price) { this.price = price; }

@DynamoDBAttribute(attributeName="PageCount")
public int getPageCount() { return pageCount; }
public void setPageCount(int pageCount) { this.pageCount = pageCount;}

@DynamoDBAttribute(attributeName="ProductCategory")
public String getProductCategory() { return productCategory; }
public void setProductCategory(String productCategory) { this.productCategory = productCategory; }

@DynamoDBAttribute(attributeName="InPublication")
public int getInPublication() { return inPublication; }
public void setInPublication(int inPublication) { this.inPublication = inPublication; }
```

Documents – principle

- documents are **self-contained**, **aggregate data structures**
- they consist of attributes (name-value pairs)
- attribute values have **data types, which can also be nested/hierarchical**

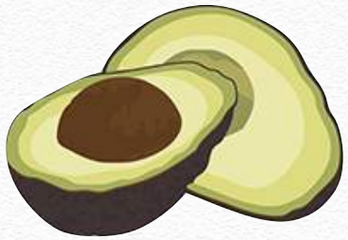
Objects vs. documents

- **programming language objects** can often be **stored easily** in documents
- lists/arrays, and sub-objects from programming language objects **do not need to be normalised** and **re-assembled** later
- **one programming language object** is often **one document in the database**

Document stores

- document stores have a **type system**, so they can **perform** some **basic validation** on data
- as **each document carries an implicit schema**, document stores can **access all document attributes** and **sub-attributes individually**, offering lots of query power

document-store examples



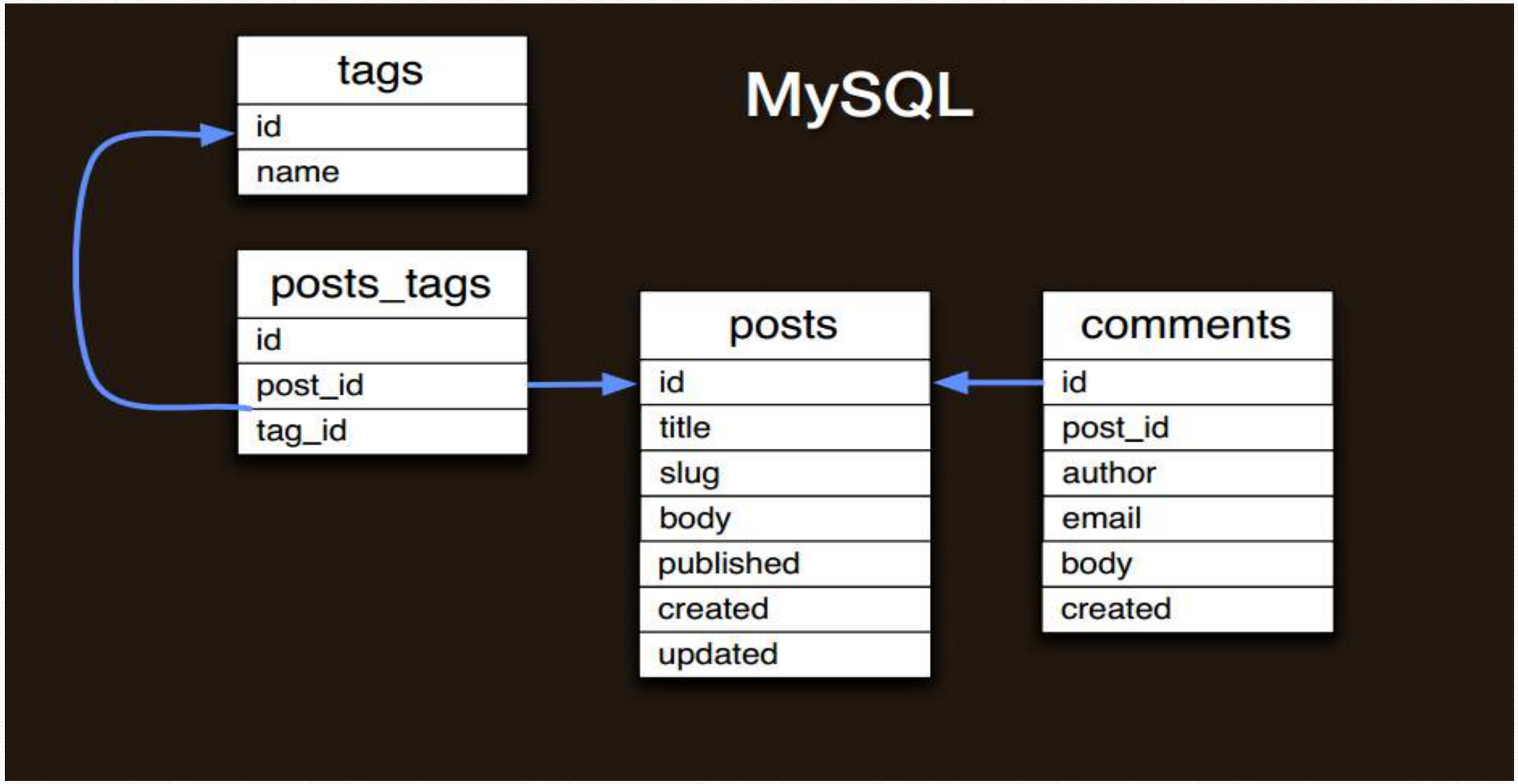
ArangoDB

multi-model NoSQL

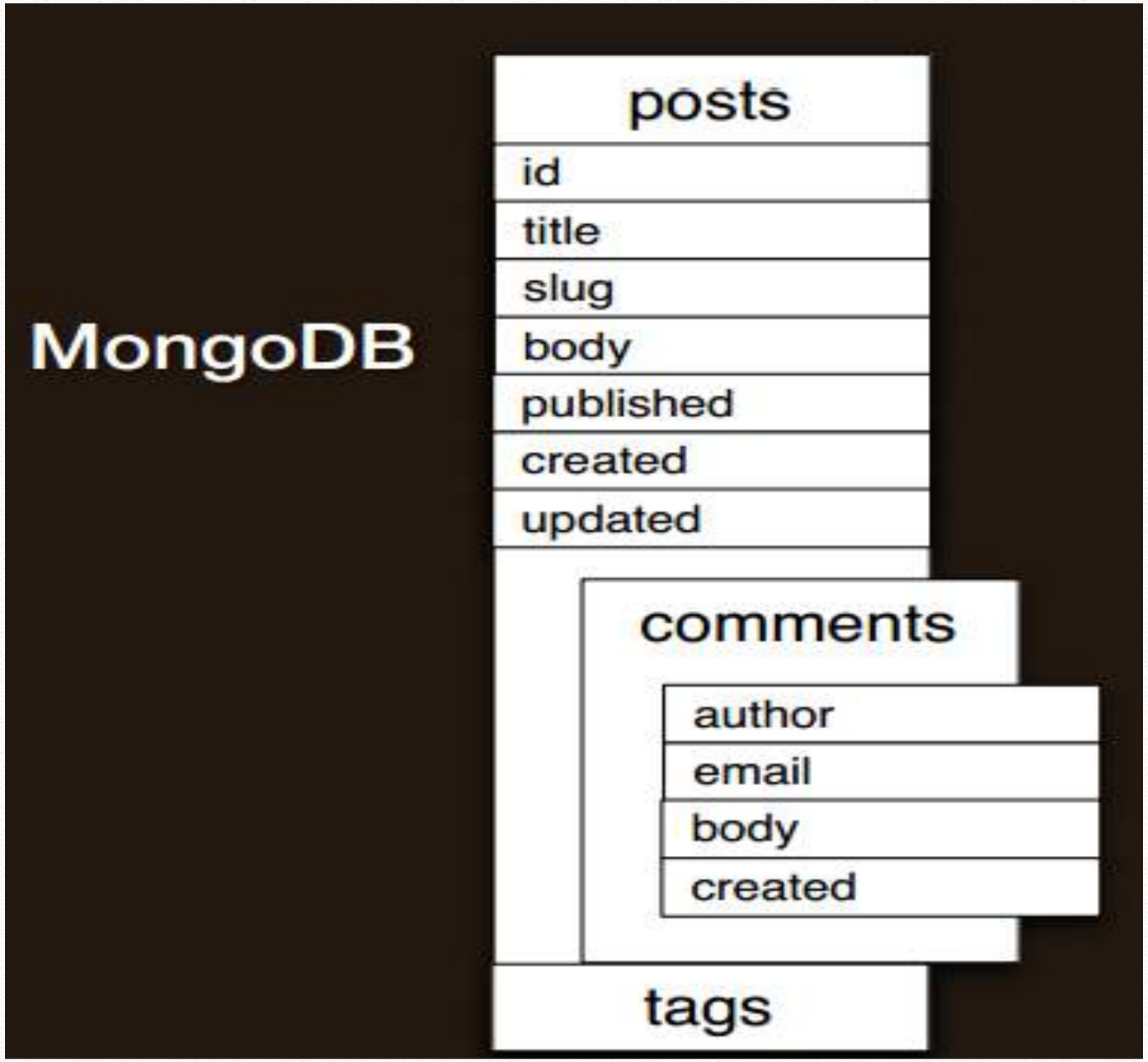
MongoDB

RDBMS (mysql, postgres)	MongoDB
Tables	Collections
Records/rows	Documents/objects
Queries return record(s)	Queries return a cursor

MongoDB


MySQL
tags
id
name
posts_tags
id
post_id
tag_id
posts
id
title
slug
body
published
created
updated
comments
id
post_id
author
email
body
created

MongoDB


MongoDB
posts
id
title
slug
body
published
created
updated
comments
author
email
body
created
tags

MongoDB

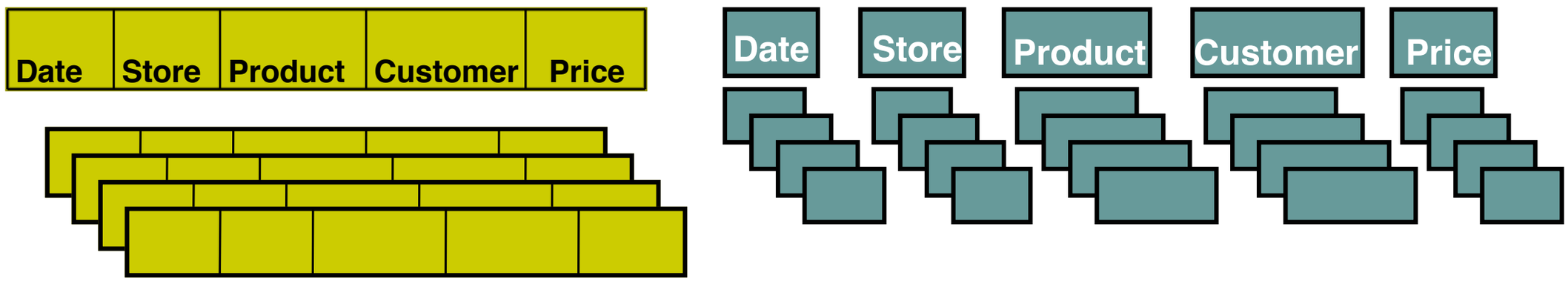
MongoDB

```
{
    "_id" : ObjectId("4c03e856e258c2701930c091"),
    "title" : "Welcome to MongoDB",
    "slug" : "welcome-to-mongodb",
    "body" : "Today, we're gonna totally rock your world...",
    "published" : true,
    "created" : "Mon May 31 2010 12:48:22 GMT-0400 (EDT)",
    "updated" : "Mon May 31 2010 12:48:22 GMT-0400 (EDT)",
    "comments" : [
        {
            "author" : "Bob",
            "email" : "bob@example.com",
            "body" : "My mind has been totally blown!",
            "created" : "Mon May 31 2010 12:48:22 GMT-0400 (EDT)"
        }
    ],
    "tags" : [
        "databases", "MongoDB", "awesome"
    ]
}
```

https://www.youtube.com/watch?v=pWbMrx5rVBE

- Web tutorial to **configure** MongoDB

column-store



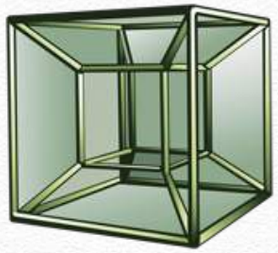
+ easy to add/modify a record

- might read in unnecessary data

+ only need to read in relevant data

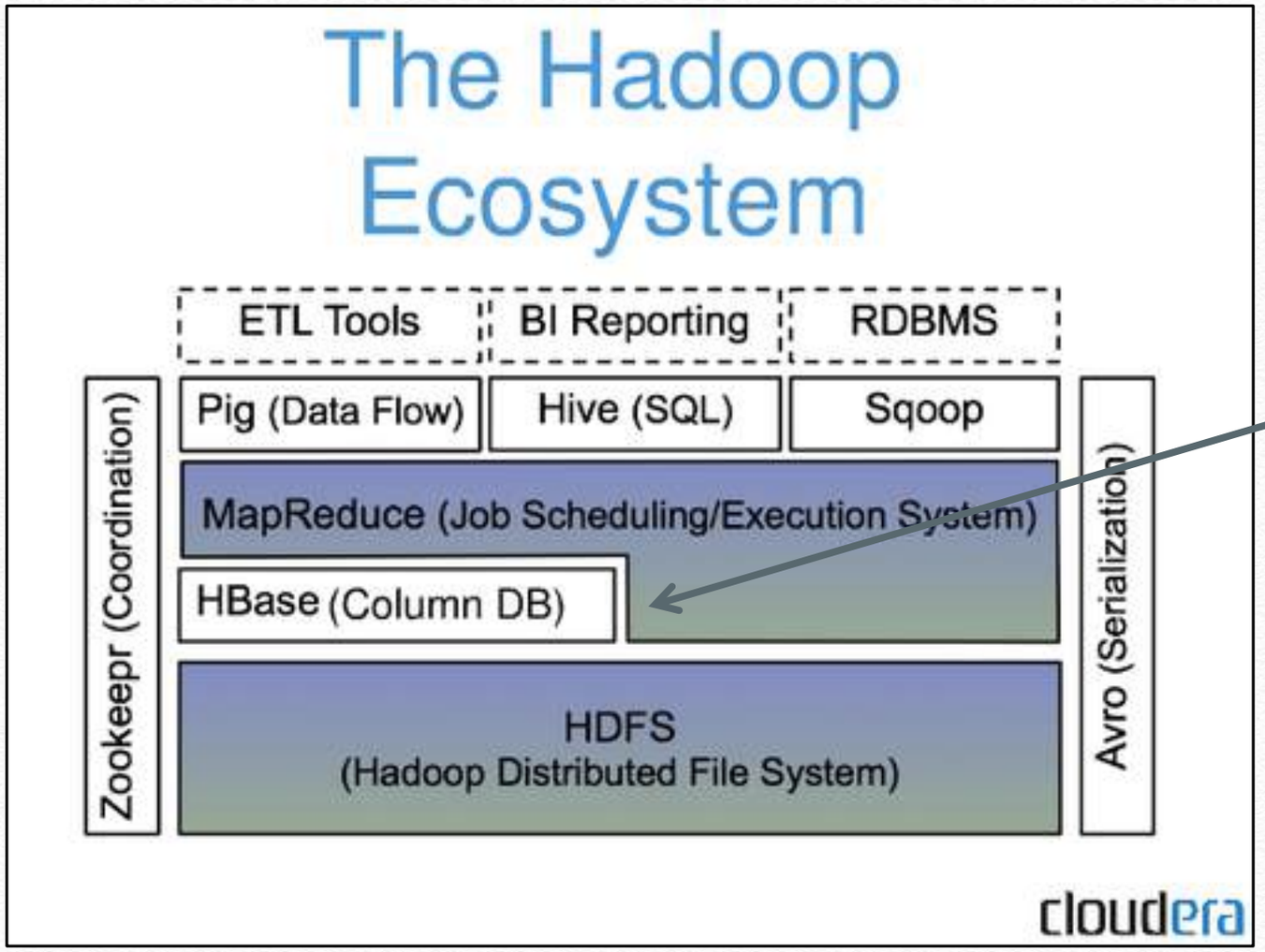
- tuple writes require multiple accesses

column-store examples



HYPERTABLE INC

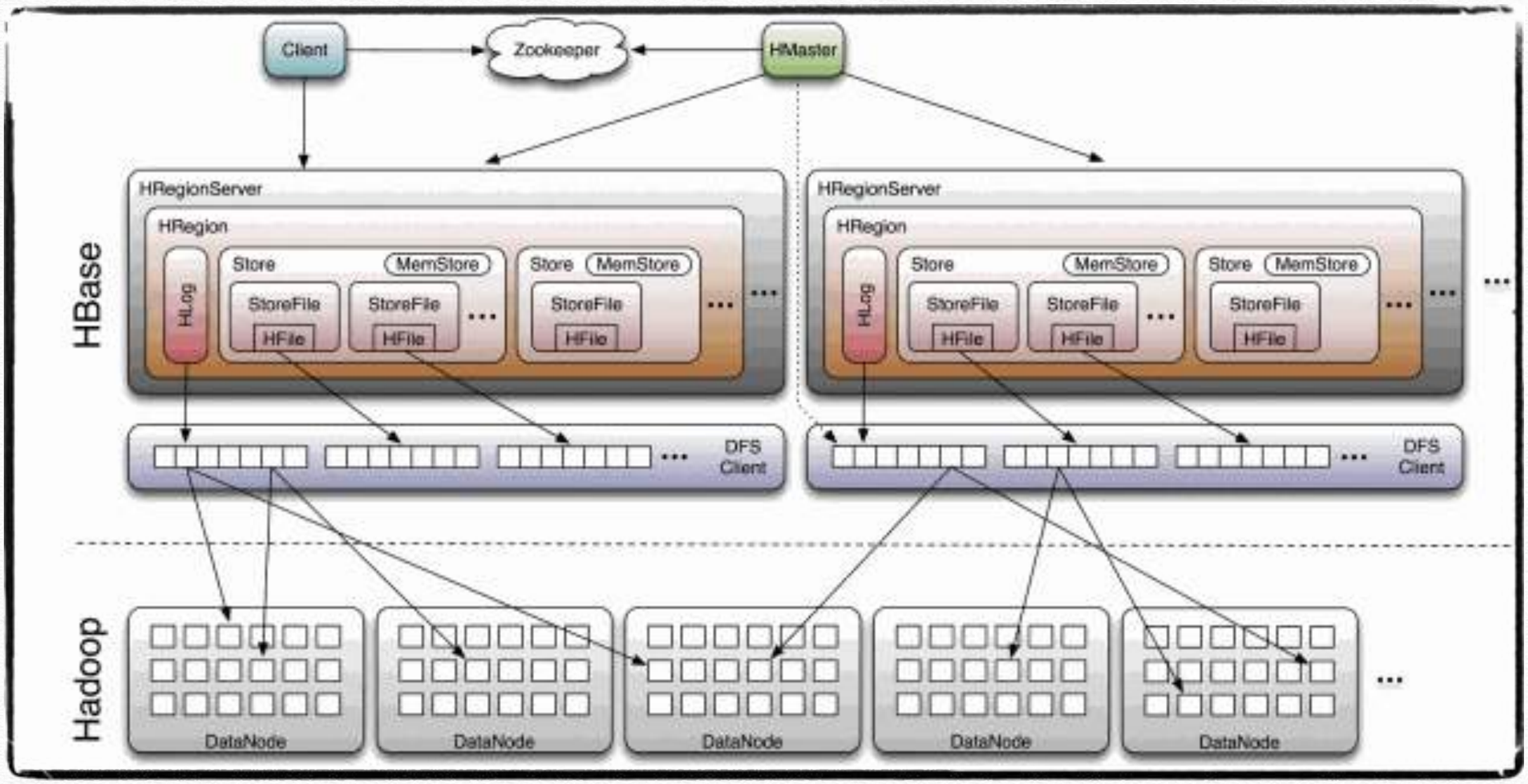
Apache HBase



HBase is built on top of HDFS

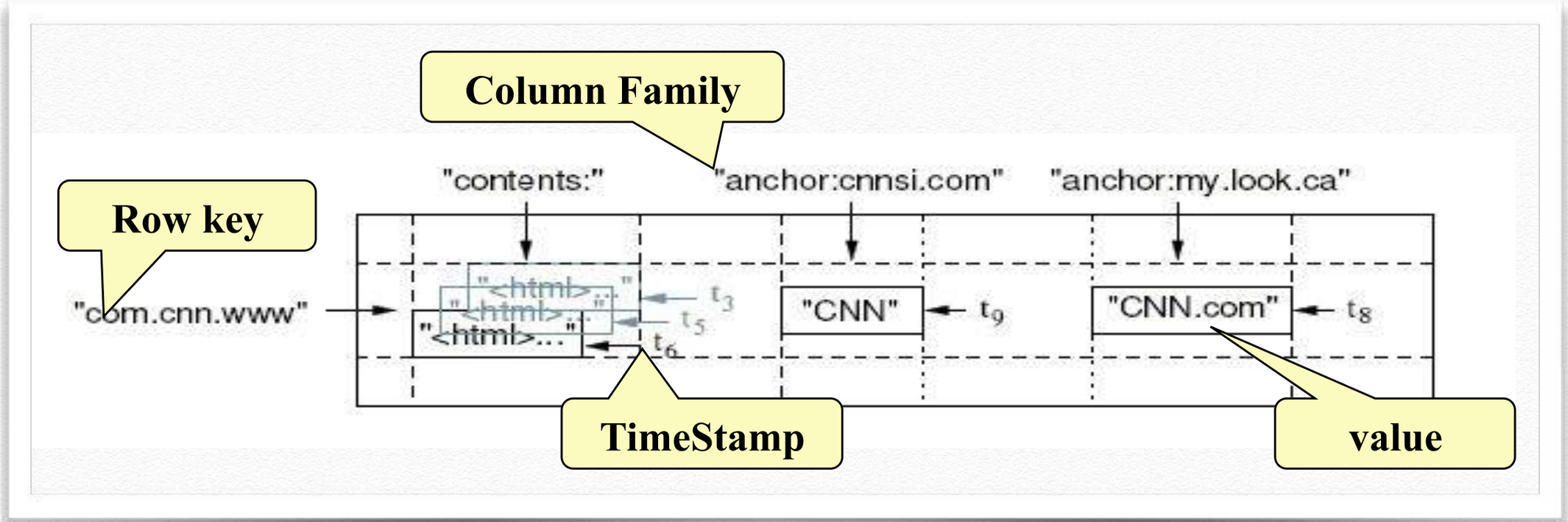
HBase files are internally stored in HDFS

Apache HBase



Apache HBase

- **HBase is based on Google's Bigtable model (Key-Value pairs)**



Apache HBase

- **Logical View of HBase**

Implicit PRIMARY KEY in RDBMS terms

Data is all `byte[]` in HBase

Row key	Data
cutting	info: { 'height': '9ft', 'state': 'CA' } roles: { 'ASF': 'Director', 'Hadoop': 'Founder' }
tlipcon	info: { 'height': '5ft7, 'state': 'CA' } roles: { 'Hadoop': 'Committer'@ts=2010, 'Hadoop': 'PMC'@ts=2011, 'Hive': 'Contributor' }

Different types of data separated into different "column families"

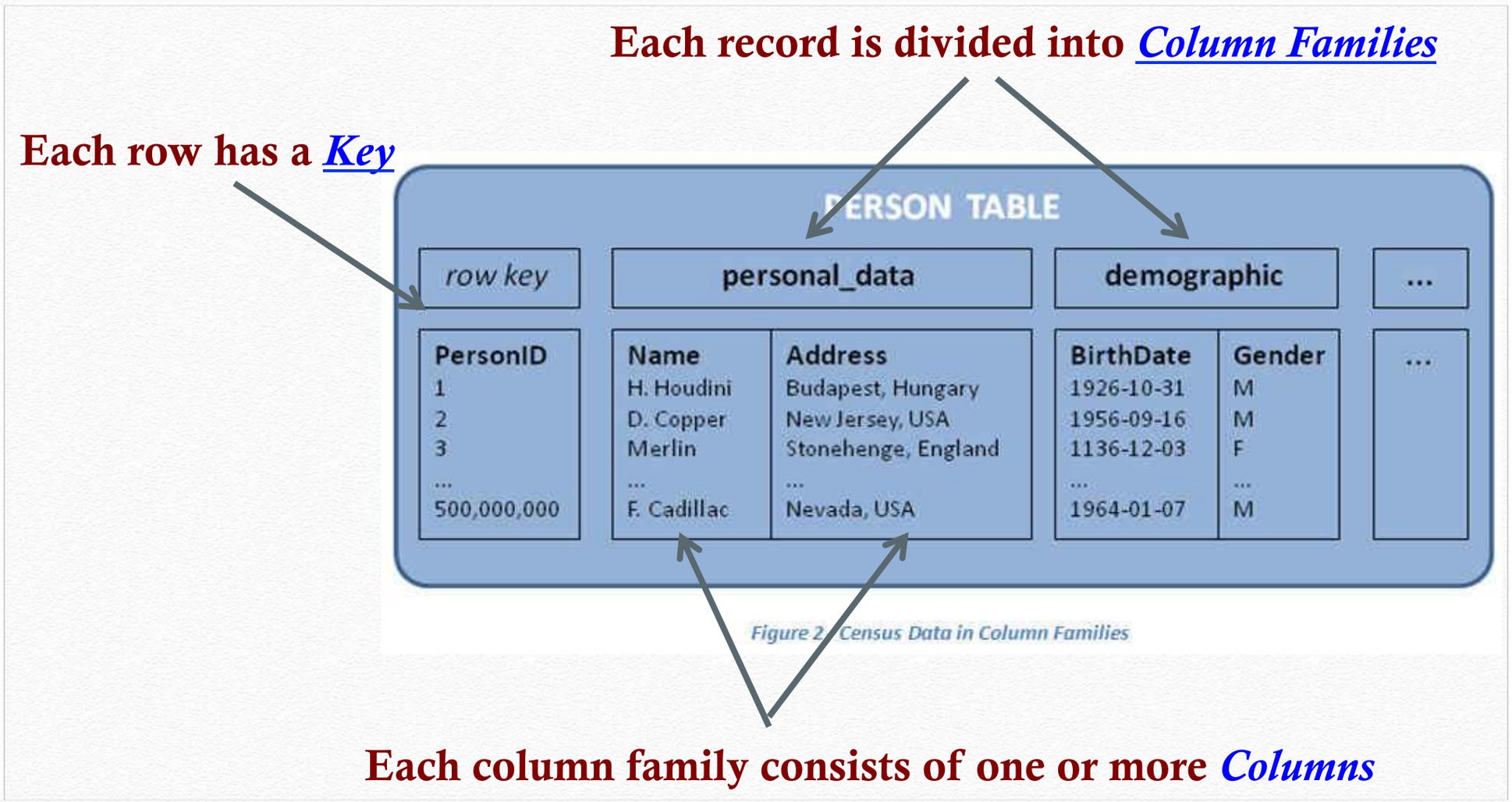
Different rows may have different sets of columns(table is *sparse*)

A single cell might have different values at different timestamps

Useful for *-To-Many mappings

Apache HBase

- **Keys and Column Families**



Apache HBase

- **Tables can be very sparse**

Row Key	Time Stamp	ColumnFamily `contents`	ColumnFamily `anchor`
"com.cnn.www"	t9		`anchor:cnnsi.com` = "CNN"
"com.cnn.www"	t8		`anchor:my.look.ca` = "CNN.com"
"com.cnn.www"	t6	`contents:html` = "<html>..."	
"com.cnn.www"	t5	`contents:html` = "<html>..."	
"com.cnn.www"	t3	`contents:html` = "<html>..."	

Apache HBase

- **The example of products**

```
{
   Id = 201
   ProductName = "18-Bicycle 201"
   Description = "201 description"
   BicycleType = "Road"
   Brand = "Brand-Company A"
   Price = 100
   Color = [ "Red", "Black" ]
   ProductCategory = "Bicycle"
}
```

```
{
   Id = 101
   ProductName = "Book 101 Title"
   ISBN = "111-1111111111"
   Authors = [ "Author 1", "Author 2" ]
   Price = 2
   Dimensions = "8.5 x 11.0 x 0.5"
   PageCount = 500
   InPublication = 1
   ProductCategory = "Book"
}
```

Apache HBase

Products

row key	t	general	bike	book
row1	t1	general:**Id** = 201 general:**ProductName** = "18-Bicycle 201"	bike:**Description** = "D1" bike:**BicycleType** = "Road" bike:**Brand** = "Brand-Company A" bike:**Price** = 100 bike:**Color** = ["Red", "Black"]	
row2	t2	general:**Id** = 101 general:**ProductName** = “Book 101 Title"		book:**ISBN** = "111-1111111111" book:**authors** = ["A1", "A2"] book:**price** = 2 book:**dimensions** = “8.5 x 11.0 x 0.5" book:**Pages** = 500 book:**InPublication** = 1

HBASE Running

- **Hbase shell:**

```
hbase(main):005:0> create 'products', 'general', 'bike', 'book'
0 row(s) in 1.0970 seconds

hbase(main):006:0> list
TABLE
products
1 row(s) in 0.0380 seconds

hbase(main):007:0> put 'products', 'row1', 'general:Id', 201
0 row(s) in 0.0940 seconds

hbase(main):009:0> put 'products', 'row1', 'general:ProductName', '18-Bicycle 201'
0 row(s) in 0.0200 seconds

hbase(main):011:0> scan 'products'
ROW          COLUMN+CELL
 row1        column=general:Id, timestamp=1400685808533, value=201
 row1        column=general:ProductName, timestamp=1400685851499, value=18-Bicycle 201
1 row(s) in 0.0310 seconds
```

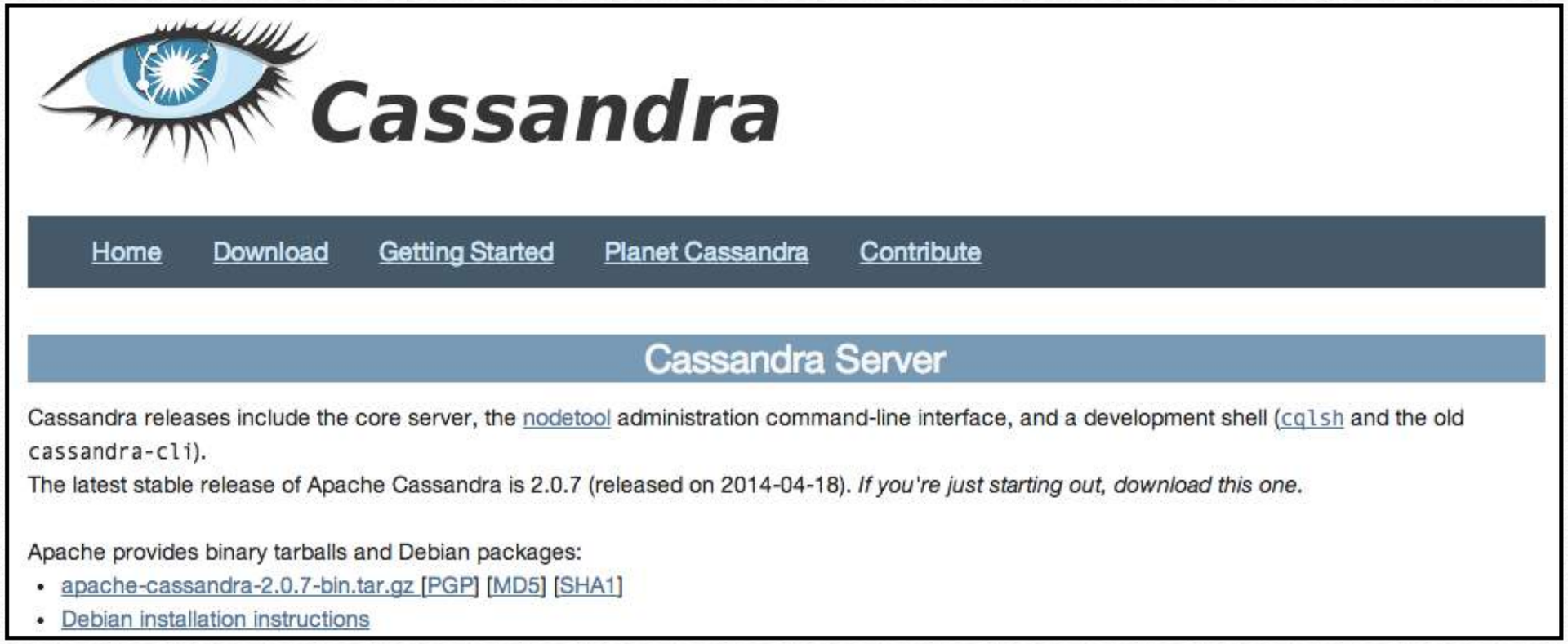
HBASE Running

- **Hbase shell:**

```
hbase(main):007:0> put 'products', 'row2', 'general:Id', 101
0 row(s) in 0.0940 seconds

hbase(main):009:0> put 'products', 'row2', 'general:ProductName', ‘Book 101 Title'
0 row(s) in 0.0200 seconds

hbase(main):014:0> scan 'products'
ROW          COLUMN+CELL
 row1        column=general:Id, timestamp=1400685808533, value=201
 row1        column=general:ProductName, timestamp=1400685851499, value=18-Bicycle 201
 row2        column=general:Id, timestamp=1400686106206, value=101
 row2        column=general:ProductName, timestamp=1400686122543, value=Book 101 Title
2 row(s) in 0.0290 seconds
```

Cassandra Configuration

- Download a **release** of **apache Cassandra:**
- **apache-cassandra-3.11.6-bin.tar.gz**



Cassandra

Home Download Getting Started Planet Cassandra Contribute

Cassandra Server

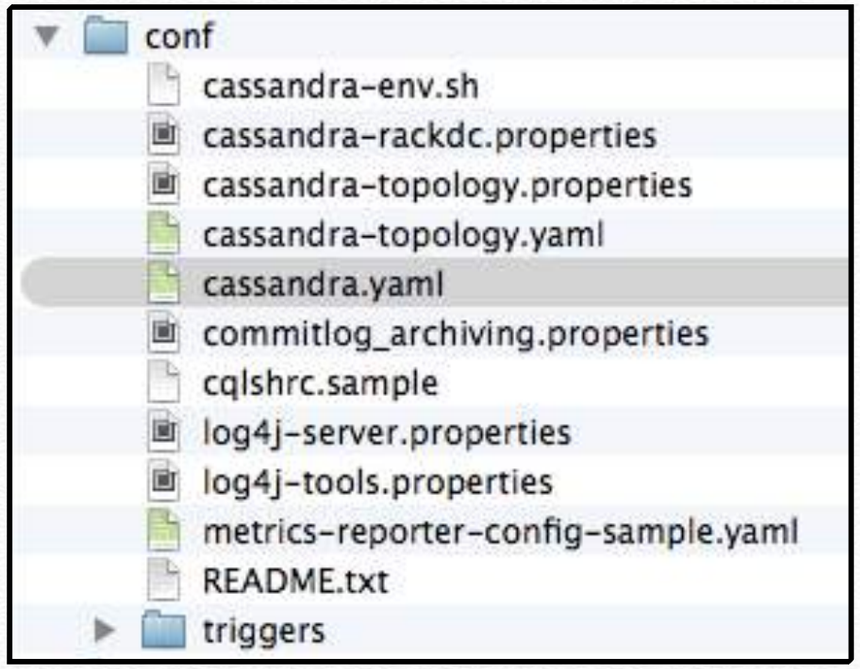
Cassandra releases include the core server, the nodetool administration command-line interface, and a development shell (`cqlsh` and the old `cassandra-cli`).
The latest stable release of Apache Cassandra is 2.0.7 (released on 2014-04-18). *If you're just starting out, download this one.*

Apache provides binary tarballs and Debian packages:

- apache-cassandra-2.0.7-bin.tar.gz [PGP] [MD5] [SHA1]
- Debian installation instructions

Cassandra Configuration

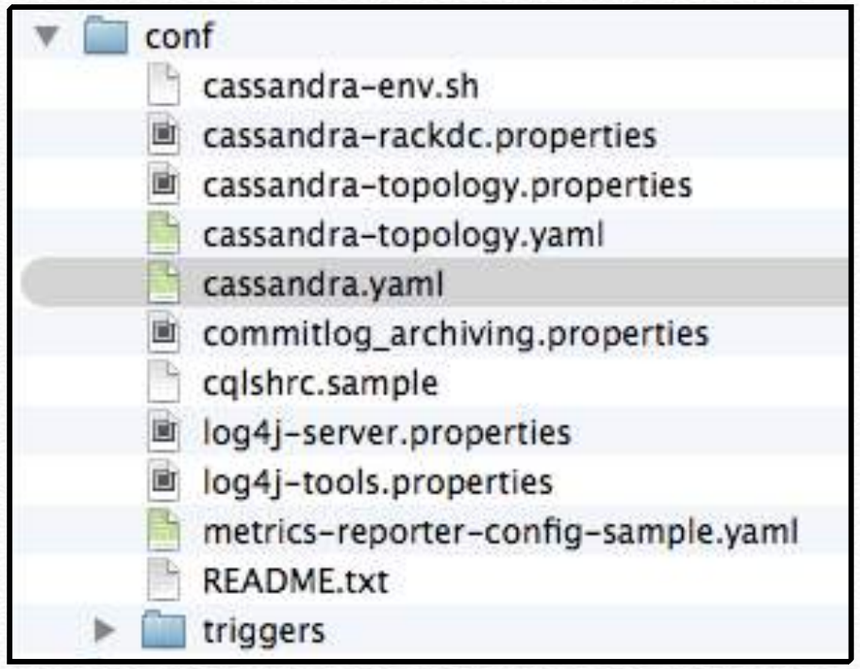
- Go to the **conf** directory in the **cassandra-home** directory
- set the **cassandra.yaml** file as follows




```yaml
# Directories where Cassandra should store data on disk.  Cassandra
# will spread data evenly across them, subject to the granularity of
# the configured compaction strategy.
data_file_directories: <home>/dsc-cassandra-3.11.6/lib/cassandra/data

# commit log
commitlog_directory: <home>/dsc-cassandra-3.11.6/lib/cassandra/commitlog

# saved caches
saved_caches_directory: <home>/dsc-cassandra-3.11.6/lib/cassandra/saved_caches
```

Cassandra Configuration

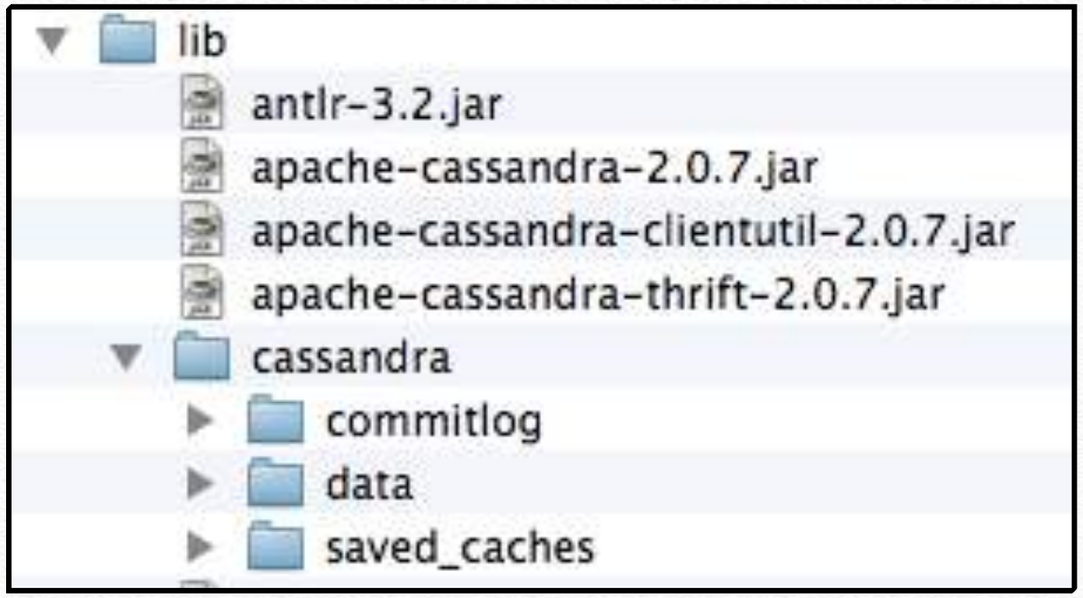
- Go to the **conf** directory in the **cassandra-home** directory
- set the **log4j-server.properties** file as follows




```
log4j.appender.R.File=<home>/dsc-cassandra-3.11.6/log/cassandra/system.log
```

Cassandra Configuration

- Go to the **cassandra-home** directory
- create the following directories



Cassandra Running

- Running cassandra:

```
$:~cassandra-*/bin/cassandra -f
```

- **Warning:**

```
The service should start in the foreground and log gratuitously
to the console. Assuming you don't see messages with scary words
like "error", or "fatal", or anything that looks like a Java
stack trace, then everything should be working.

You have to keep open the terminal window and execute cassandra
commands in another terminal window.

Press "Control-C" in the first terminal window to stop Cassandra.
```

Cassandra Running

- Running cassandra:

```
$:~cassandra-*/bin/cassandra
```

- **Warning:**

```
If you start up Cassandra without the "-f" option, it
will run in the background.

You can stop the process by killing it, using
'pkill -f CassandraDaemon'
```

Cassandra Running

- Running cqlsh:

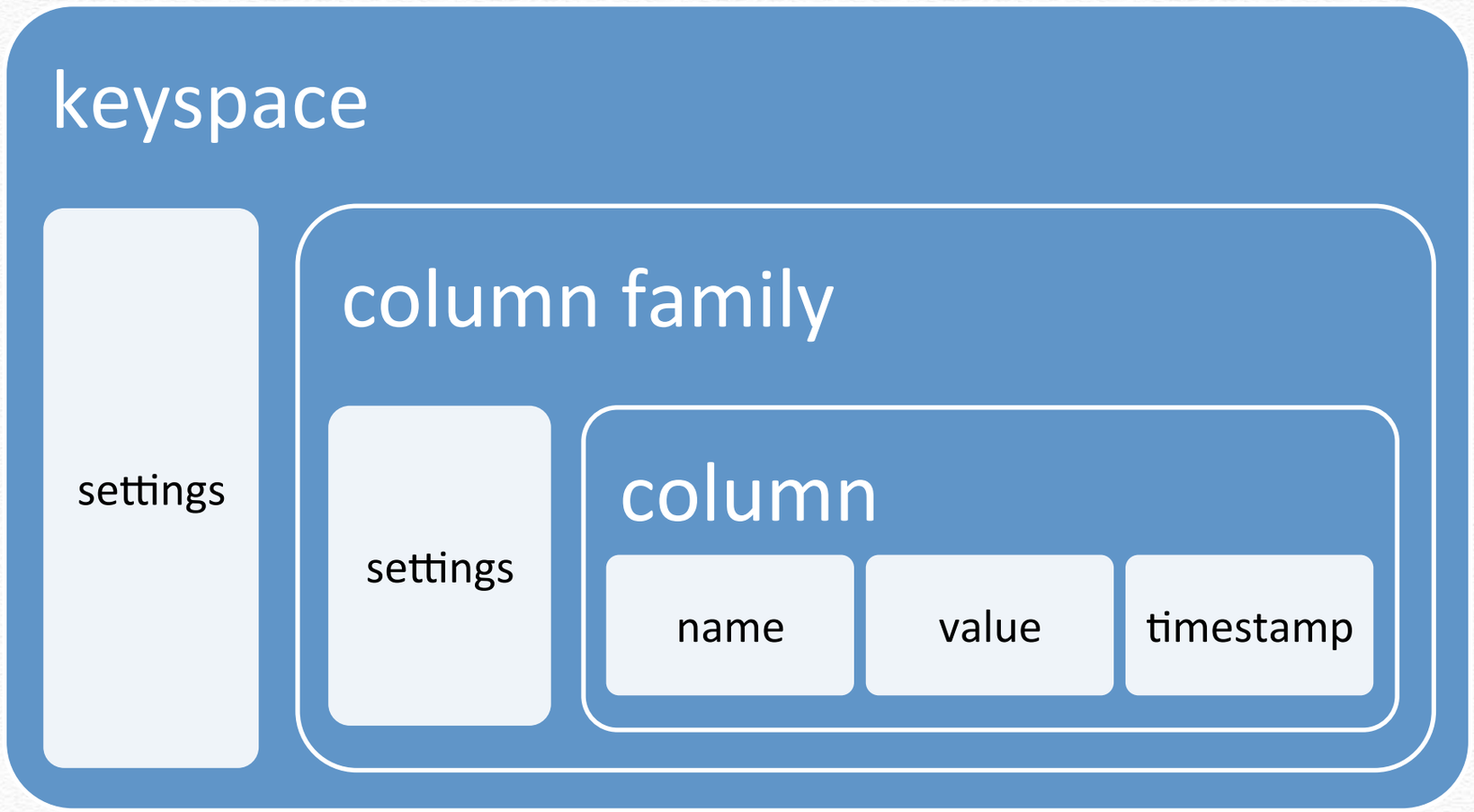
```
$:~cassandra-*/bin/cqlsh
```

- **it is an interactive command line interface for Cassandra. cqlsh allows you to execute CQL (Cassandra Query Language) statements against Cassandra. Using CQL, you can define a schema, insert data, execute queries.**

```
Connected to Test Cluster at localhost:9160.
[cqlsh 4.1.1 | Cassandra 2.0.7 | CQL spec 3.1.1 | Thrift protocol 19.39.0]
Use HELP for help.

cqlsh>
```

Apache Cassandra

- **Cassandra is quite similar to HBase**



Apache Cassandra

- **The example of products**

```
{
   Id = 201
   ProductName = "18-Bicycle 201"
   Description = "201 description"
   BicycleType = "Road"
   Brand = "Brand-Company A"
   Price = 100
   Color = "Red"
   ProductCategory = "Bicycle"
}
```

```
{
   Id = 101
   ProductName = "Book 101 Title"
   ISBN = "111-1111111111"
   Authors = "Author 1"
   Price = 2
   Dimensions = "8.5 x 11.0 x 0.5"
   PageCount = 500
   InPublication = 1
   ProductCategory = "Book"
}
```

Apache Cassandra

keyspace: Products

columnfamily: Bycicle

id	price	title	type	brand	color	description
201	100	18-Bike-201	Road	D	{black, red}	D4

columnfamily: Book

id	price	title	authors	dimensions	ISBN	InPublication	pages
101	2	T3	{A1}	8.5 x 11.0 x 1.5	111-1111111111	1	500

Cassandra Running

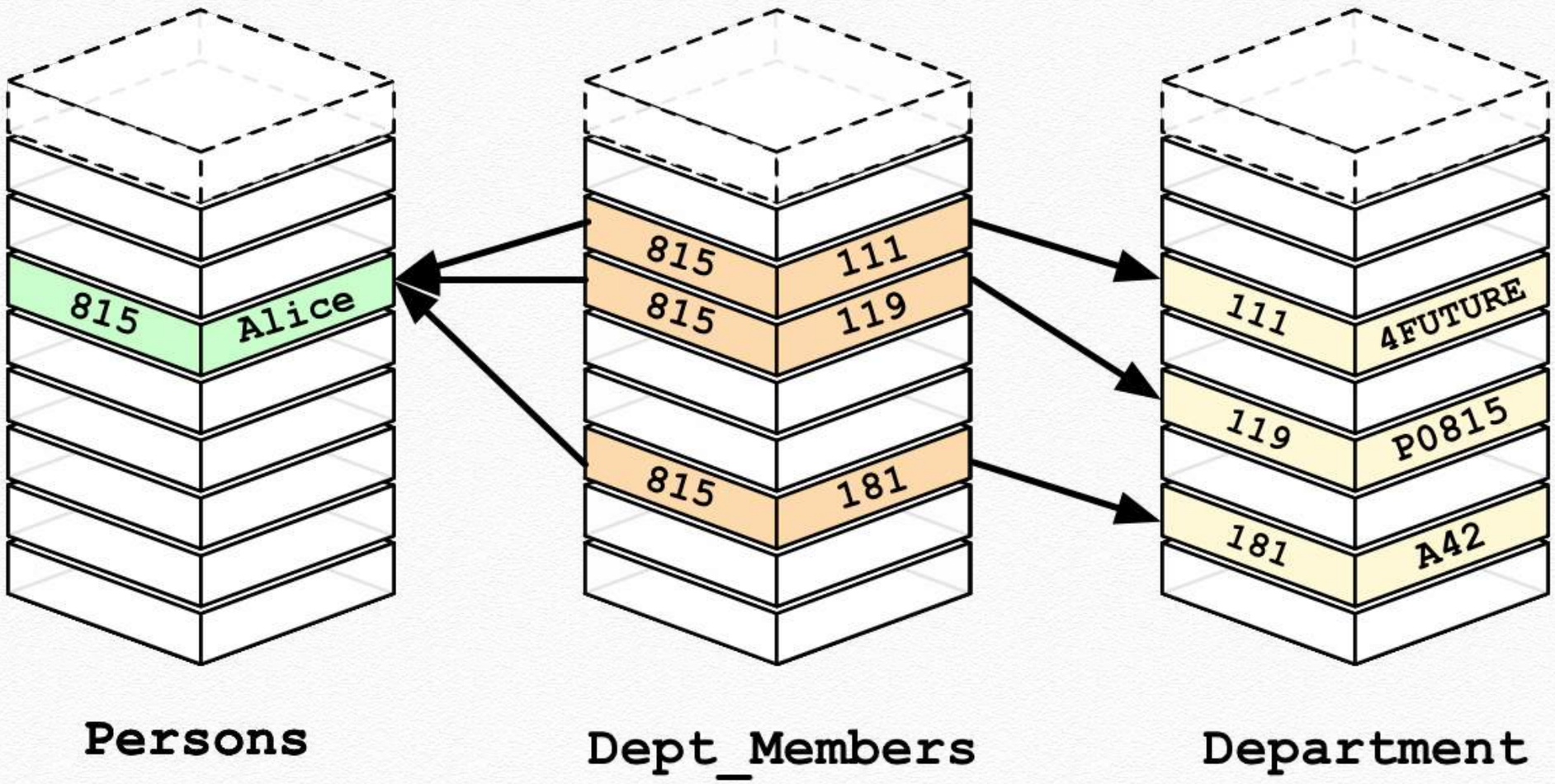
```
Connected to Test Cluster at localhost:9160.
[cqlsh 4.1.1 | Cassandra 2.0.7 | CQL spec 3.1.1 | Thrift protocol 19.39.0]
Use HELP for help.

cqlsh> CREATE KEYSPACE products
       WITH REPLICATION =
                        { 'class' : 'SimpleStrategy', 'replication_factor' : 1 };

cqlsh> USE products;

cqlsh:products> CREATE TABLE bycicle (
                   id int PRIMARY KEY,
                   price int,
                   title text,
                   type text,
                   brand text,
                   color text,
                   description text
                );
```

```
cqlsh:products> CREATE COLUMNFAMILY bycicle (
                   id int PRIMARY KEY,
                   price int,
                   title text,
                   type text,
                   brand text,
                   color text,
                   description text
                );
```

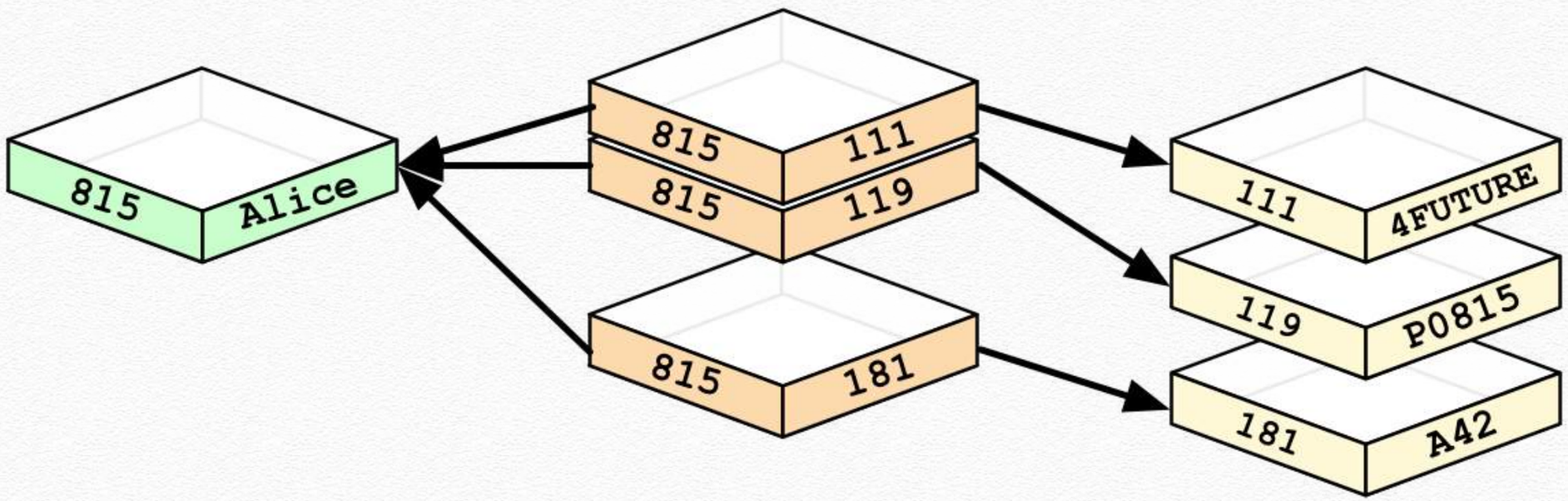
Cassandra Running

```
cqlsh:products> INSERT INTO bycicle
                          (id,  price, title, type, brand, color, description)
                     VALUES (201, 100, '18-Bike-201', 'Road', 'D', 'red', ‘D4');


cqlsh:products> SELECT * FROM bycicle;

 id  | brand | color | description | price  | title       | type
-----+-------+-------+-------------+--------+-------------+------
 201 |     D |   red |          D4 |    100 | 18-Bike-201 | Road

(1 rows)

cqlsh:products> ALTER COLUMNFAMILY bycicle ADD nation text;

cqlsh:products> SELECT * FROM bycicle;

 id  | brand | color | description | nation | price  | title       | type
-----+-------+-------+-------------+--------+--------+-------------+------
 201 |     D |   red |          D4 |   null |    100 | 18-Bike-201 | Road

(1 rows)
```

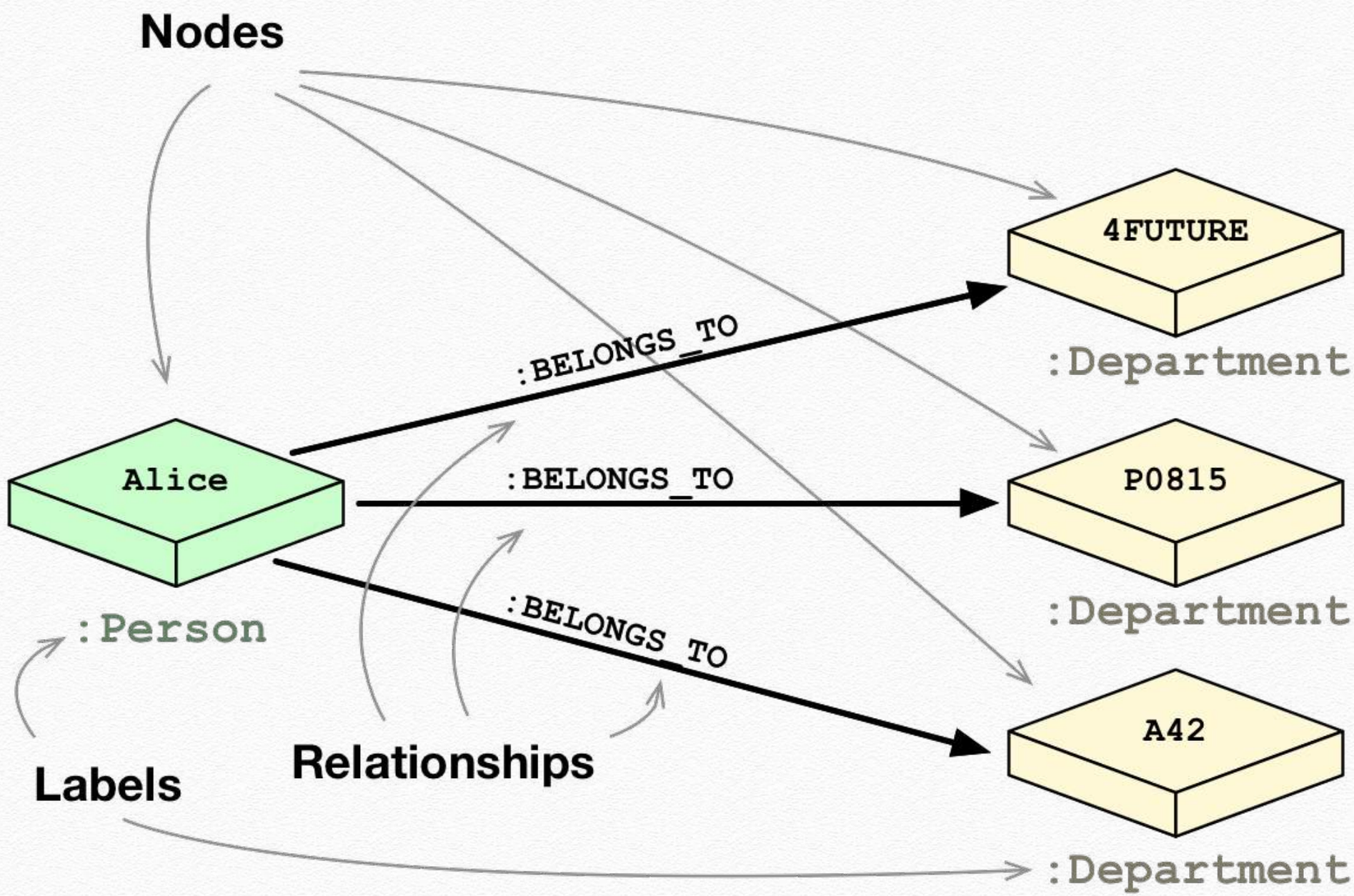
Graph database



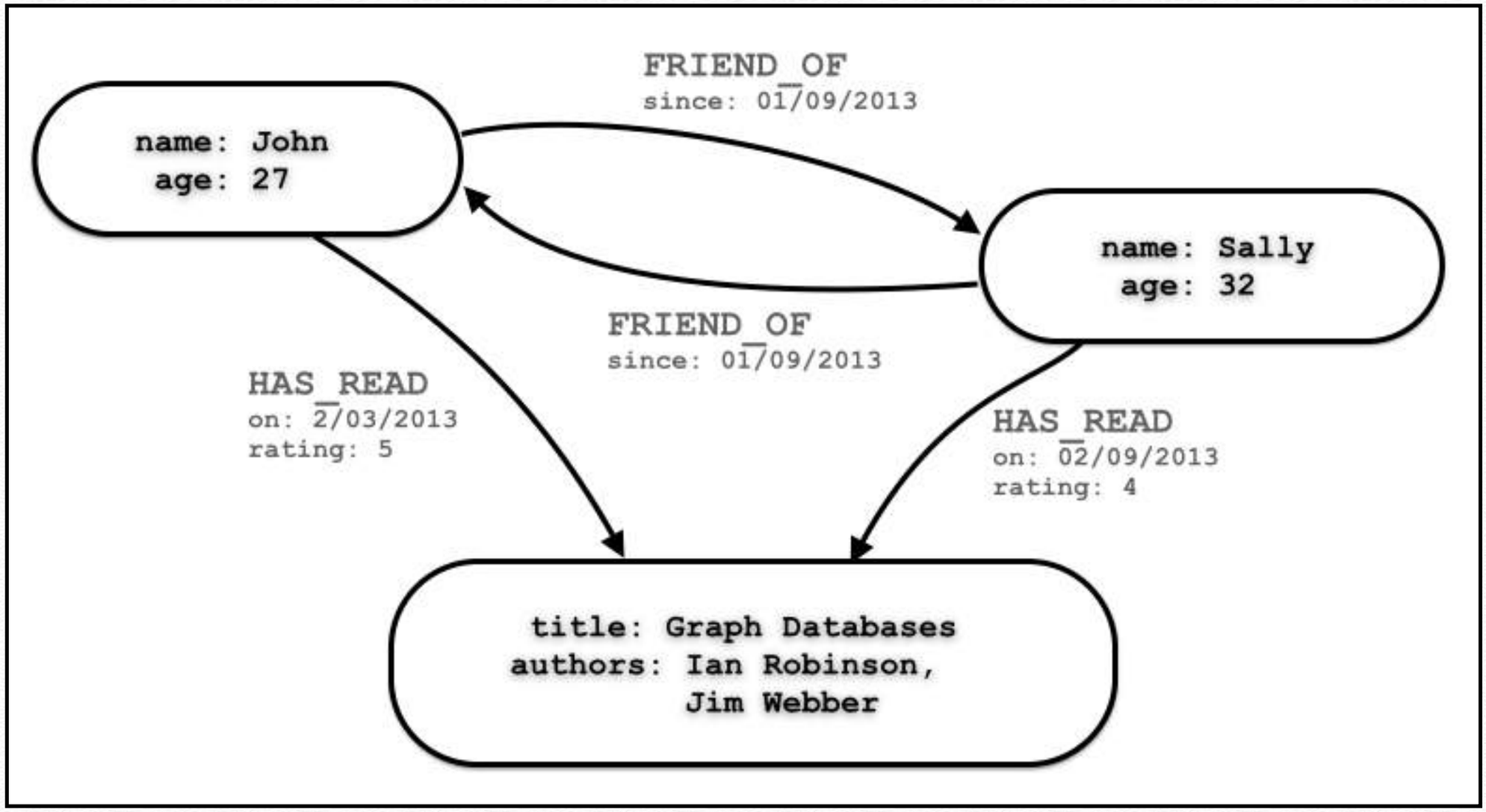
Graph database



Graph database



Graph database



Property Model

- Represents data in Nodes, Relationships and Properties
- Both Nodes and Relationships contains properties
- Relationships connects nodes
- Properties are key-value pairs
- Nodes are represented using circle and Relationships are represented using arrow keys.
- Relationships have directions: Unidirectional and Bidirectional.

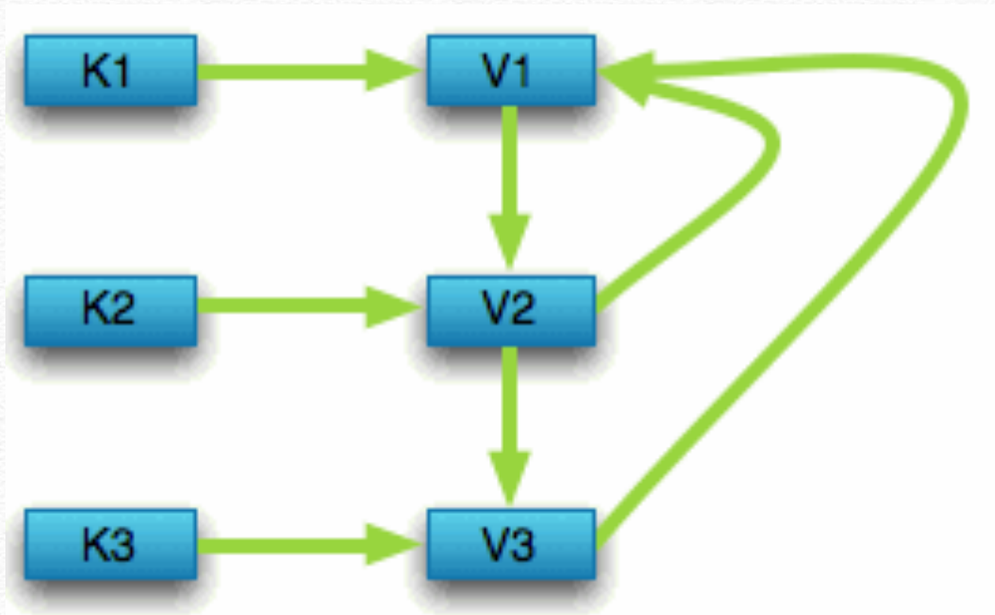
Property Model

RDBMS Vs Graph Database

S.No.	RDBMS	Graph Database
1.	Tables	Graphs
2.	Rows	Nodes
3.	Columns and Data	Properties and its values
4.	Constraints	Relationships
5.	Joins	Traversal

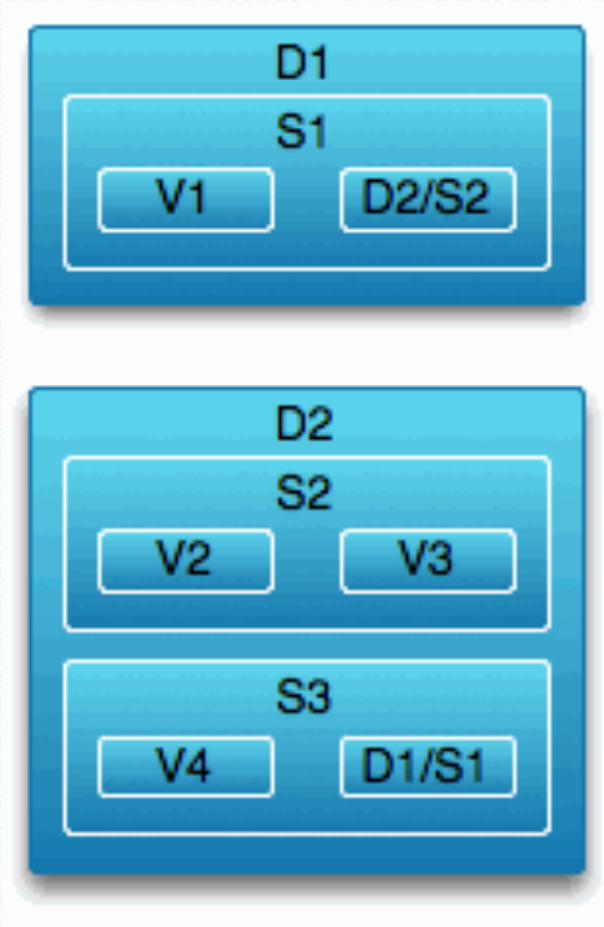
Relate Key-Value Stores with Graph Databases

Key-Value Store

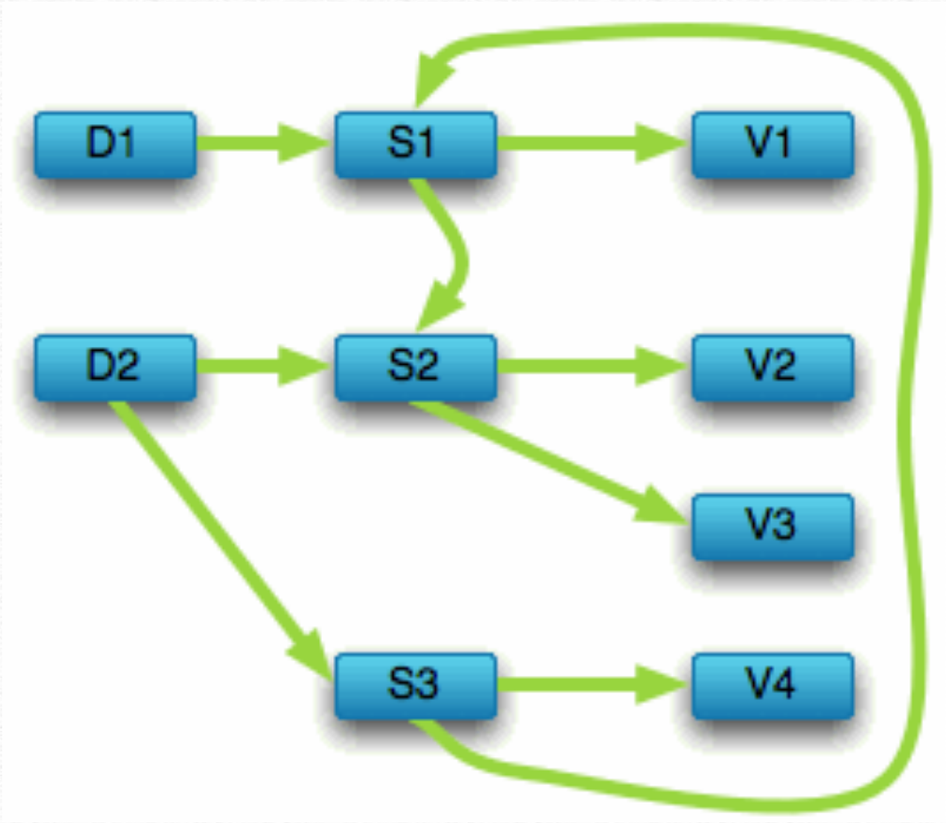


Graph Database

Navigate Document Stores with Graph Databases

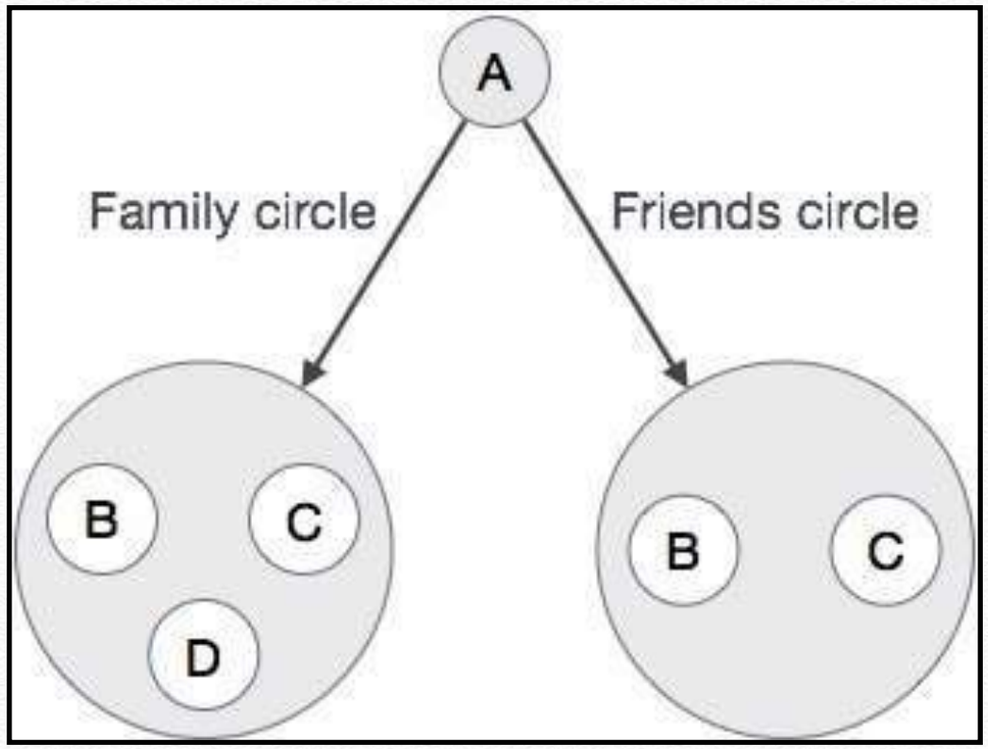


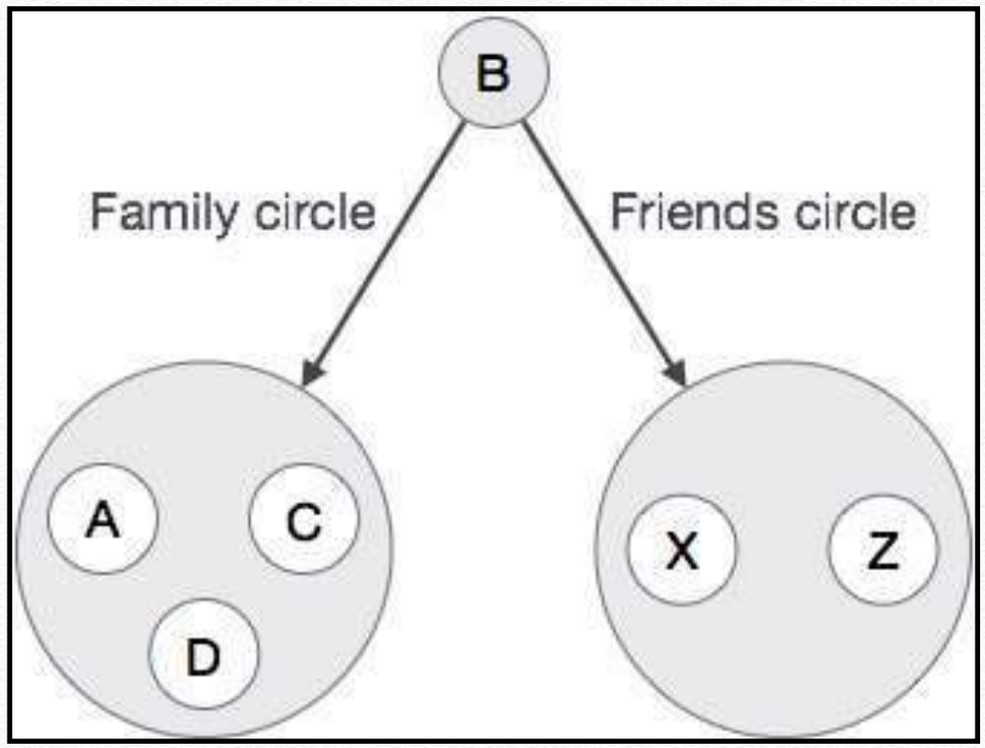
Document Store



Graph Database

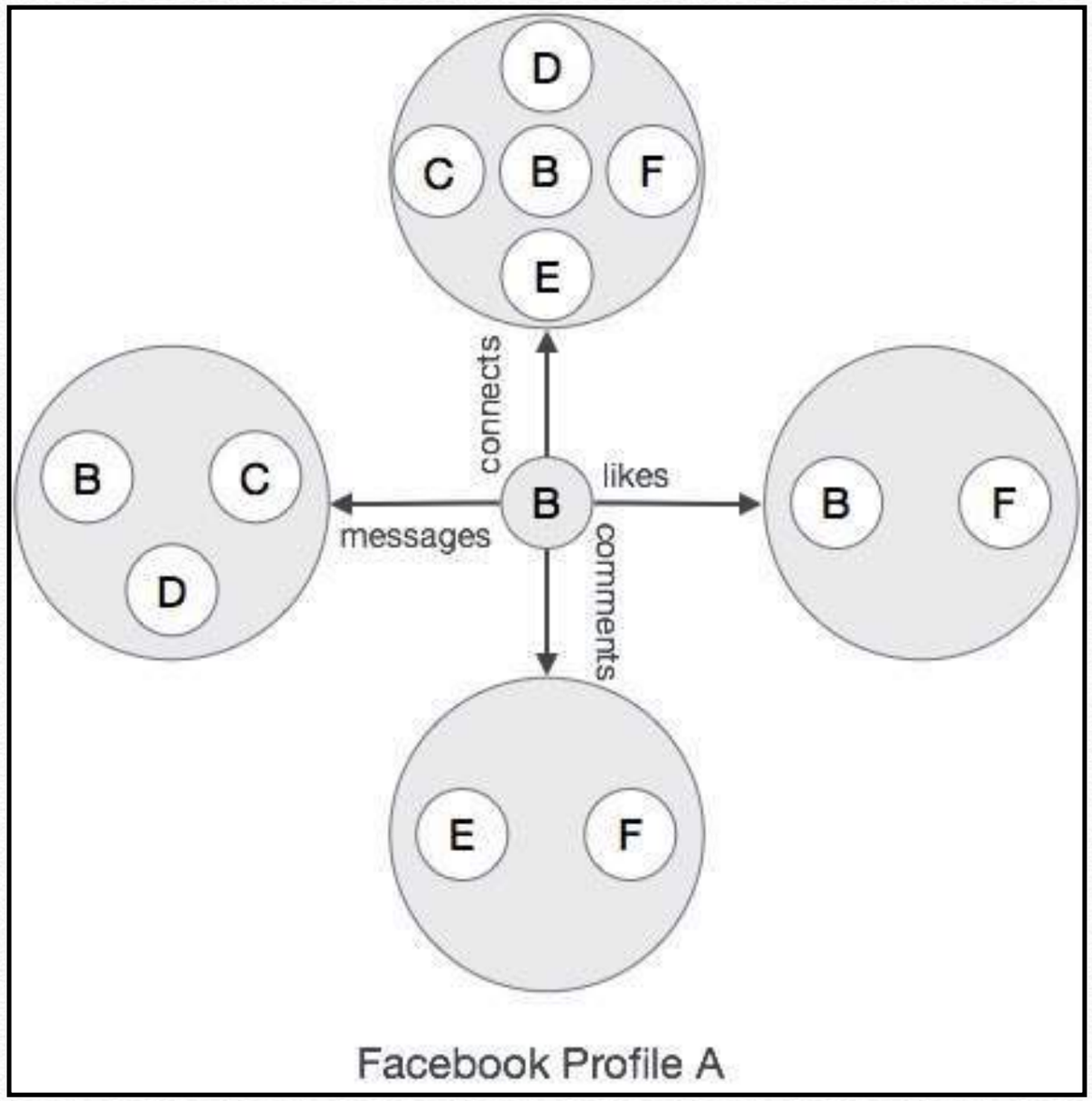
Need for Graph database





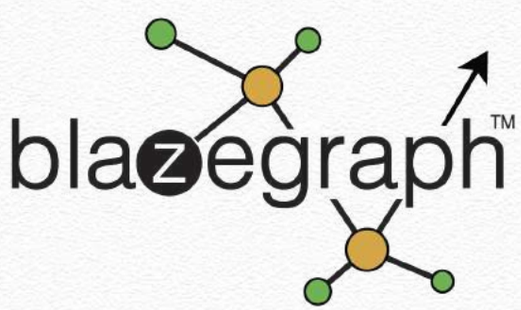
google+

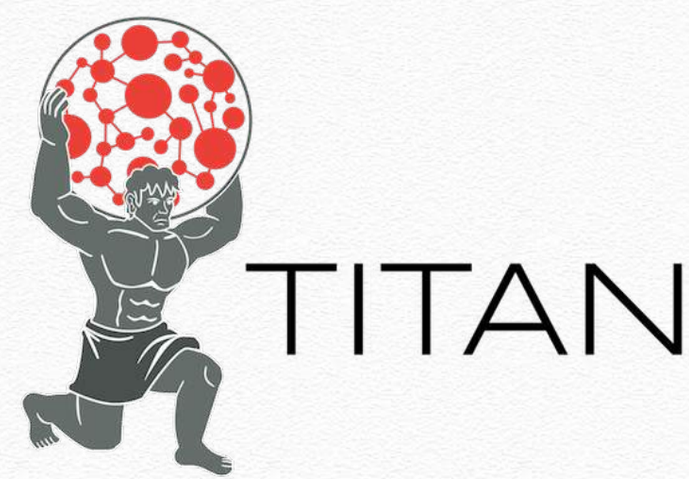
Need for Graph database



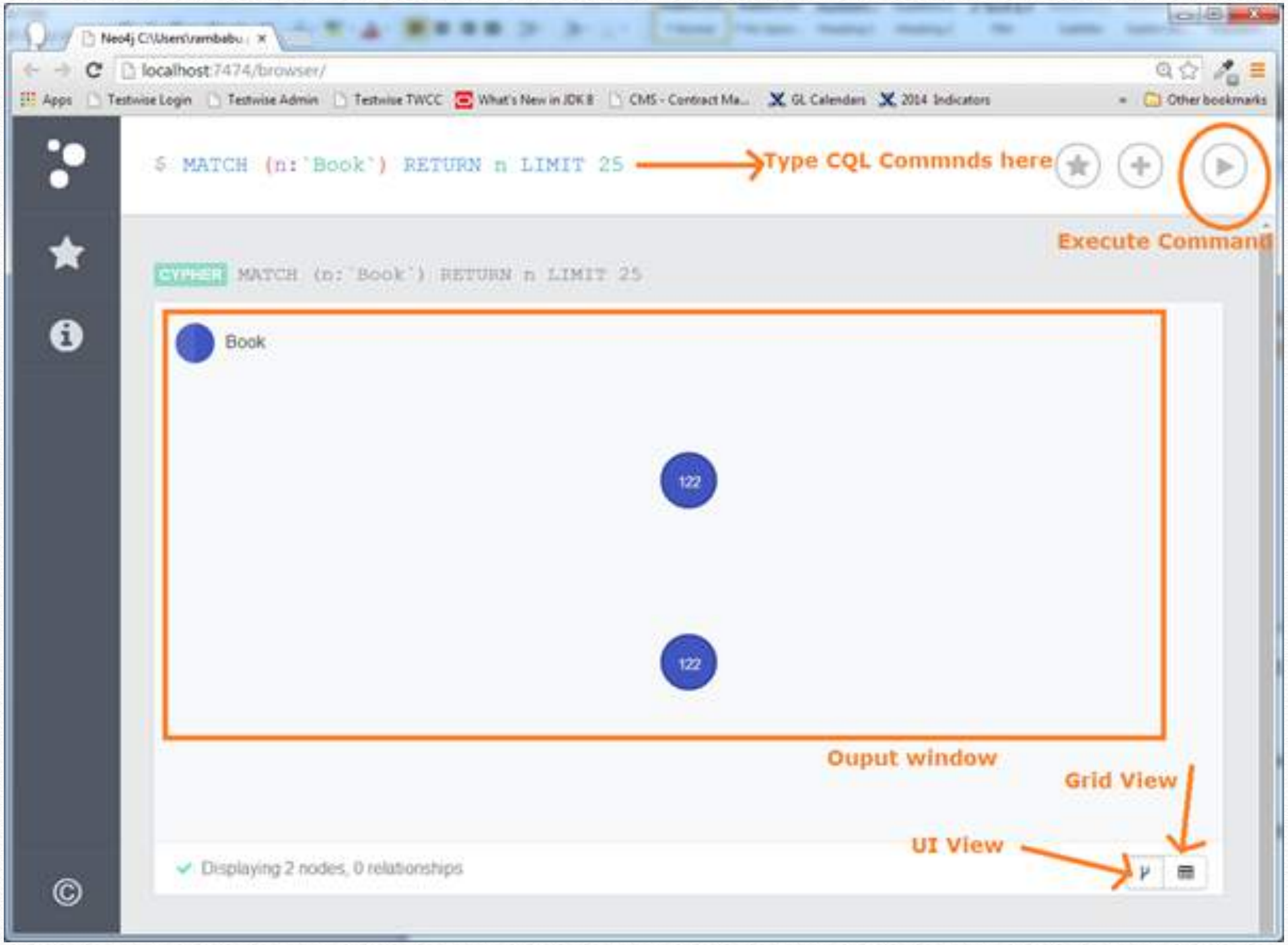
facebook

graph database examples

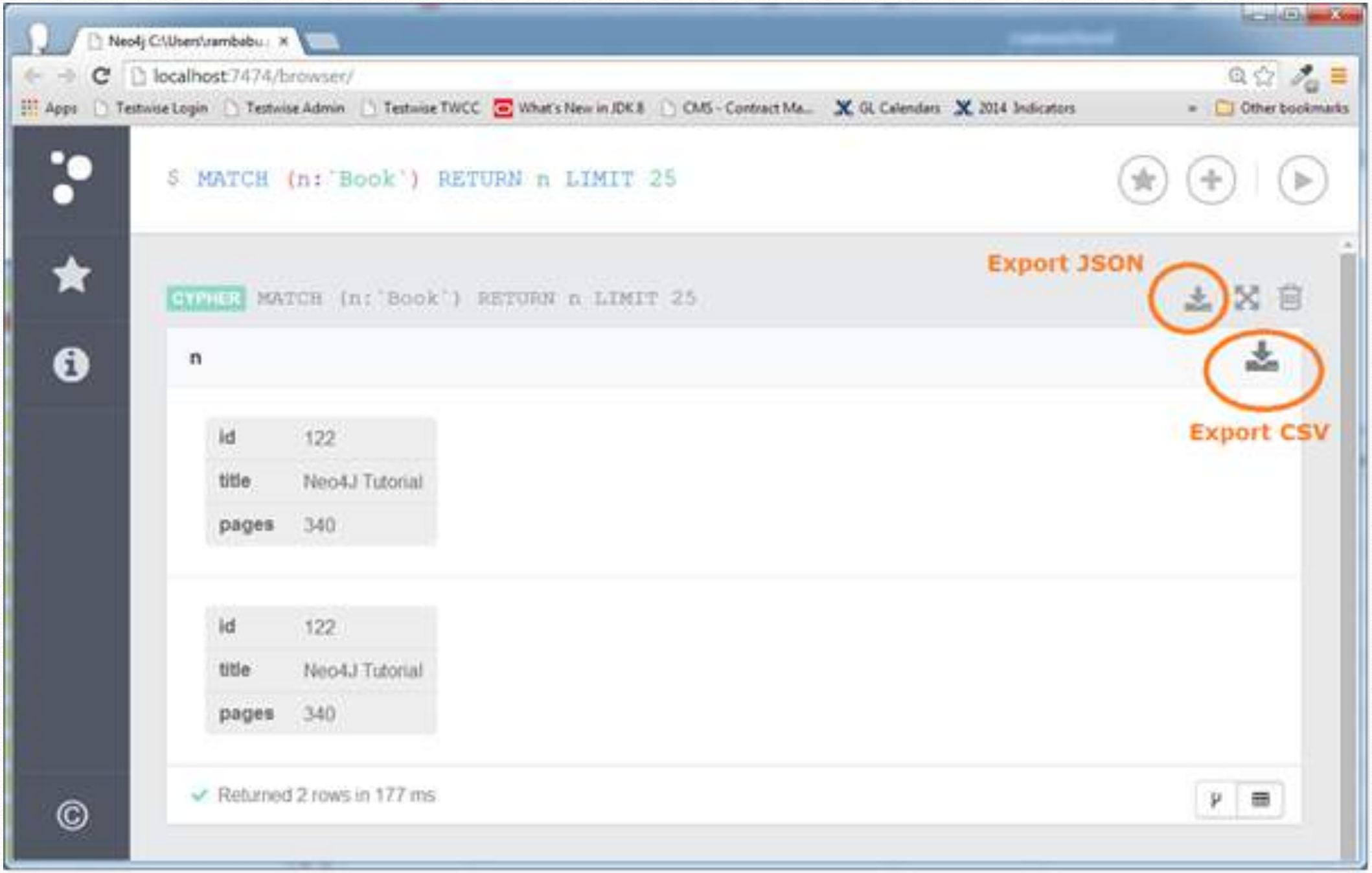




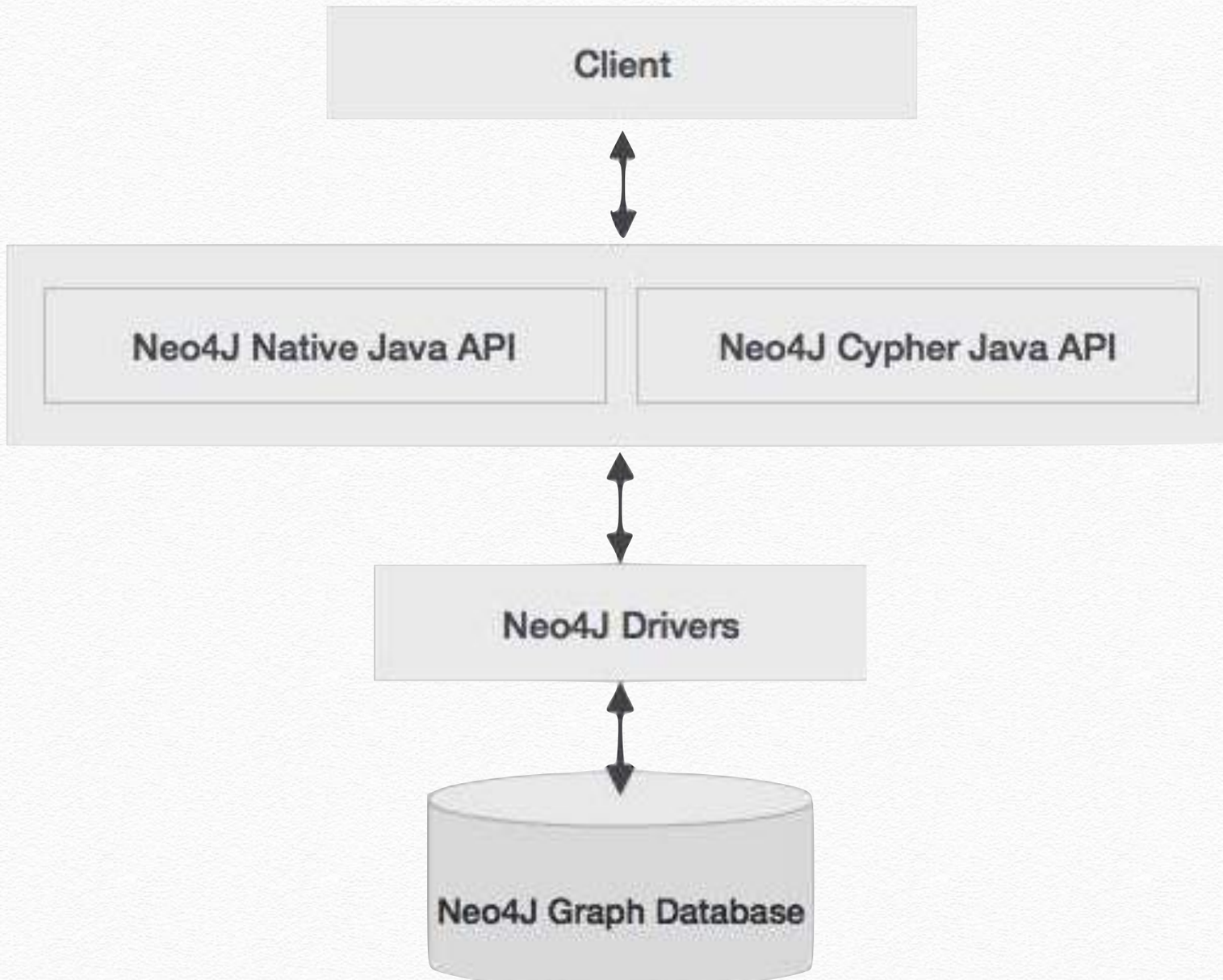
Neo4J


localhost:7474/browser/
$ MATCH (n:`Book`) RETURN n LIMIT 25
Type CQL Commnds here
Execute Command
CYPHER MATCH (n:`Book`) RETURN n LIMIT 25
Book
122
122
Ouput window
Grid View
UI View
Displaying 2 nodes, 0 relationships

Neo4J


localhost:7474/browser/
$ MATCH (n:`Book`) RETURN n LIMIT 25
Export JSON
CYPHER MATCH (n:`Book`) RETURN n LIMIT 25
n
Export CSV
id 122
title Neo4J Tutorial
pages 340
id 122
title Neo4J Tutorial
pages 340
Returned 2 rows in 177 ms

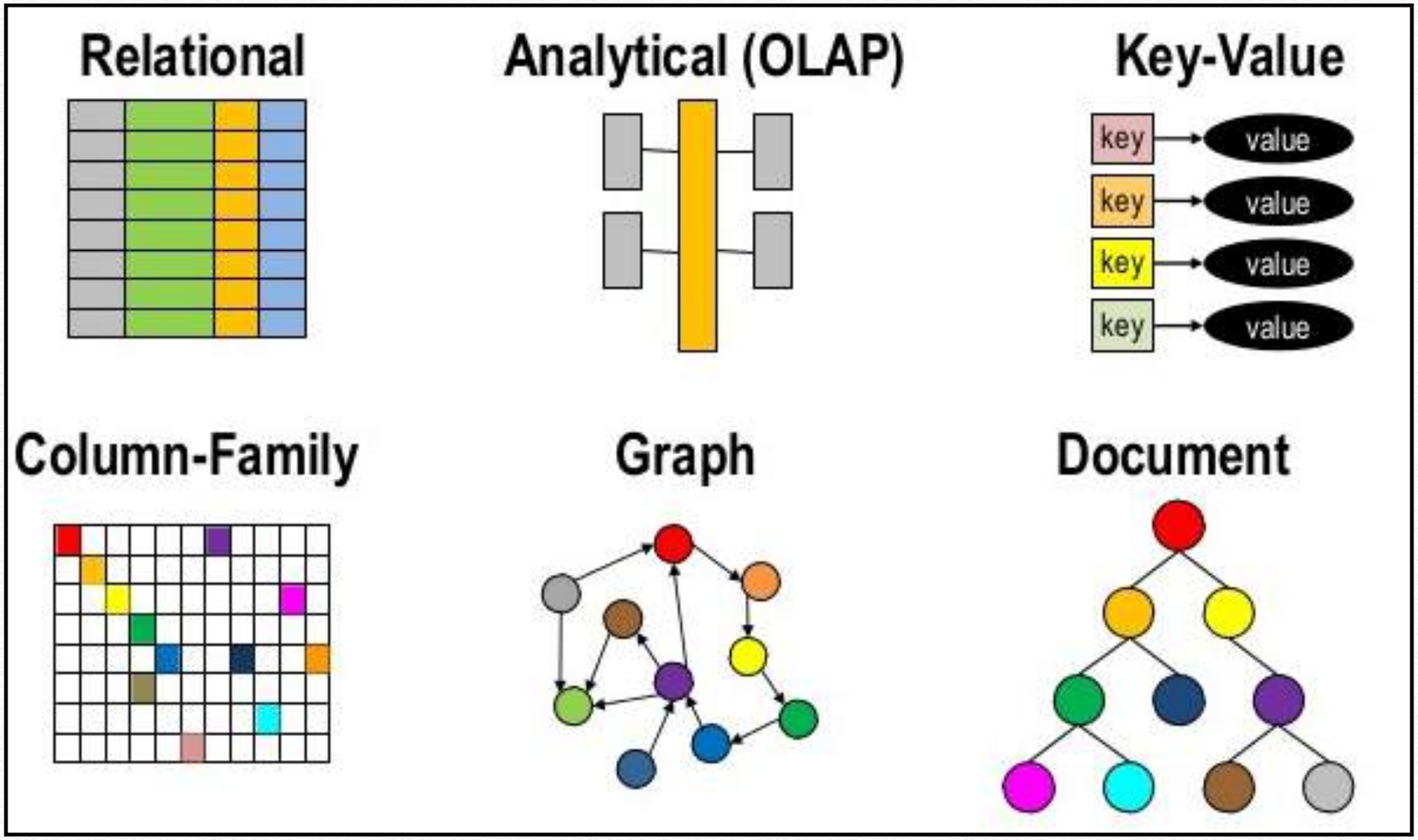
Neo4J - JAVA

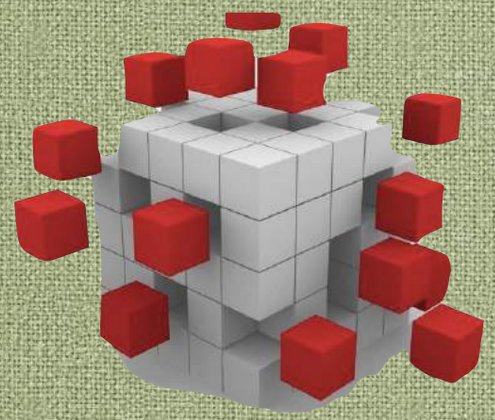

Client
Neo4J Native Java API
Neo4J Cypher Java API
Neo4J Drivers
Neo4J Graph Database

https://www.youtube.com/watch?v=Go3P73-KV30

- Web tutorial to **configure** Neo4J

Summary


Relational
Analytical (OLAP)
Key-Value
key
value
key
value
key
value
key
value
Column-Family
Graph
Document



Not Only SQL

NoSQL Databases

27/05/2020 - Big Data 2020