Doc. RNDr. Irena Holubová, Ph.D. & PROFINIT

DATA SCIENCE

NDBI048

Modern Database Systems

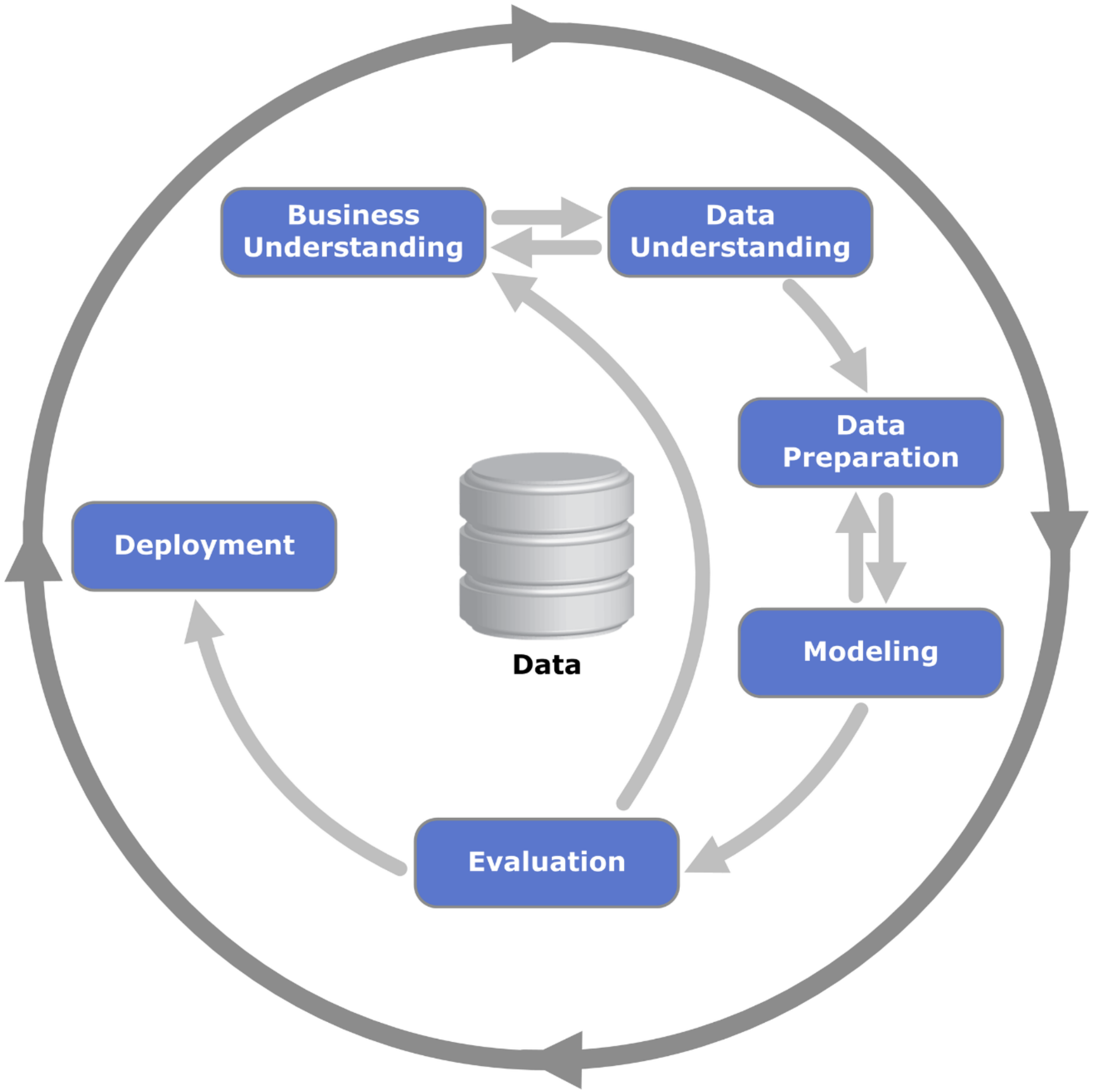
https://www.ksi.mff.cuni.cz/~holubova/NDBI048/

OUTLINE

- NoSQL databases
  - Key/value
  - Column
  - Document
  - Graph
- NewSQL databases
- Array databases
- Multi-model databases

CRISP-DM PHASES

I. Business Understanding
II. Data Understanding
III. Data Preparation
IV. Modeling
V. Evaluation
VI. Deployment



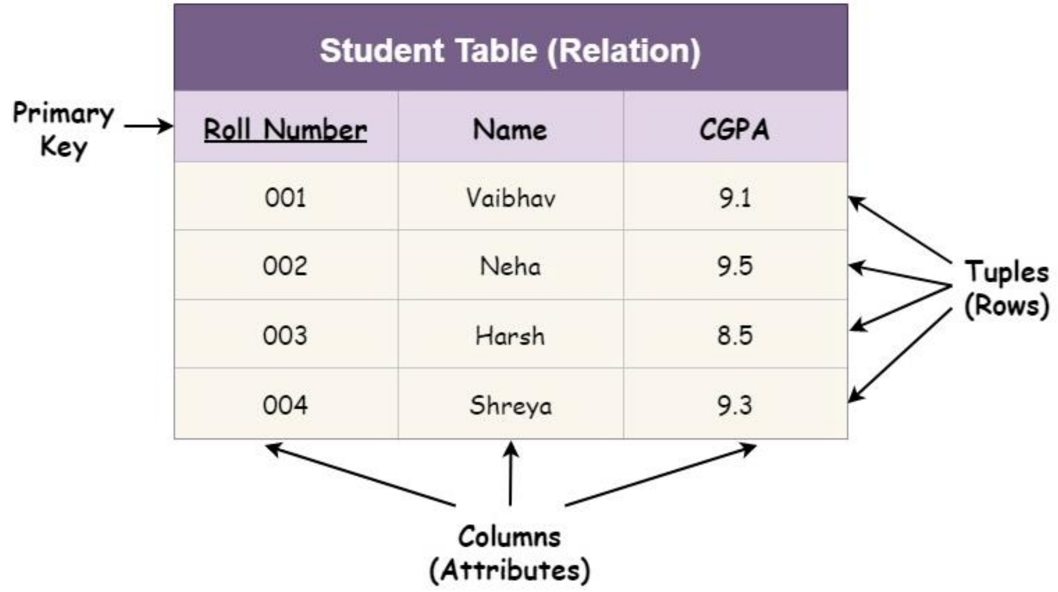
https://www.datascience-pm.com/crisp-dm-2/

NOSQL DATABASES

DATABASE = RELATIONAL DATABASE?

- A common assumption for many years
- Relational databases are able to store and process various data structures
- Advantages:
  - Simplicity
    - of the model
    - of the respective query language
  - After so many years mature and verified database management systems (DBMSs)
  - Strong mathematical background
  - …

RELATIONAL MODEL



- Proposed by E.F. Codd in 1970
  - Paper: "A relational model of data for large shared data banks"
  - IBM Research Labs
- Basic idea:
  - Storing of object and their mutual associations in tables (relations)
    - A **relation** R from X to Y is a subset of the Cartesian product X × Y.
  - Row in a table (member of relation) = object/association
  - Column (attribute) = attribute of an object/association
  - Table (relational) schema = name of the schema + list of attributes and their types
  - Schema of a relational database = set of relational schemas

RELATIONAL MODEL

- Basic integrity constraints
  - Unique identification of a row
    - Super key vs. key
  - Simple type attributes
  - NULL values
    - No “holes”
- Keys/foreign keys

BUT THE RELATIONAL MODEL WAS NOT THE FIRST ONE…

time

- First generation: navigational
  - Hierarchical model
  - Network model
- Second generation: relational
- Third generation: post-relational
  - Extensions of relational model
    - Object-relational
  - New models reacting to popular technologies
    - Object
    - XML
    - NoSQL (key/value, column, document, graph, …) - Big Data
    - Array databases
  - Multi-model systems
  - …
  - NewSQL
    - Back to the relations

TYPES OF NOSQL DATABASES

Core:

- Key-value databases
- Document databases
- Column-family (column-oriented/columnar) stores
- Graph databases

Non-core:

- Object databases
- XML databases
- …

http://nosql-database.org/

KEY/VALUE DATABASES

KEY-VALUE STORE

BASIC CHARACTERISTICS

- The simplest NoSQL data stores
- A simple hash table (map), primarily used when all access to the database is via primary key
- A table in RDBMS with two columns, such as ID and NAME
  - ID column being the key
  - NAME column storing the value
    - A BLOB that the data store just stores
- Basic operations:
  - Get the value for the key
  - Put a value for a key
  - Delete a key from the data store
- Simple → great performance, easily scaled
- Simple → not for complex queries, aggregation needs

KEY-VALUE STORE

REPRESENTATIVES

riak

redis

MemcachedDB

MEMCACHED

ORACLE

BERKELEY DB

HamsterDB
embedded database

amazon
web services
Amazon DynamoDB

not
open-source

open-source
version

Project
Voldemort

KEY-VALUE STORE

QUERYING

- We can query by the key
- To query using some attribute of the value column is (typically) not possible
  - We need to read the value to figure out if the attribute meets the conditions
- What if we do not know the key?
  - Some systems enable to retrieve the list of all keys
    - Expensive
  - Some support searching inside the value
    - Using, e.g., a kind of full-text index
      - The data must be indexed first
      - Riak search (see later)

KEY-VALUE STORE
RIAK

- Open source, distributed database
  - First release: 2009
  - Implementing principles from Amazon's Dynamo
- OS: Linux, BSD, Mac OS X, Solaris
- Language: Erlang, C, C++, some parts in JavaScript
- Built-in MapReduce support
- Stores keys into buckets = a namespace for keys
  - Like tables in a RDBMS, directories in a file system, …
  - Have a set of common properties for its contents
    - e.g., number of replicas

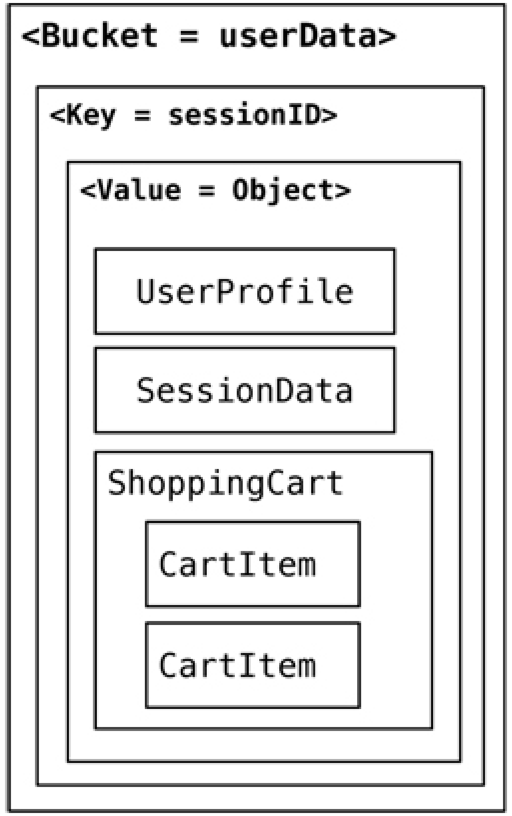
http://basho.com/riak/

RIAK BUCKETS

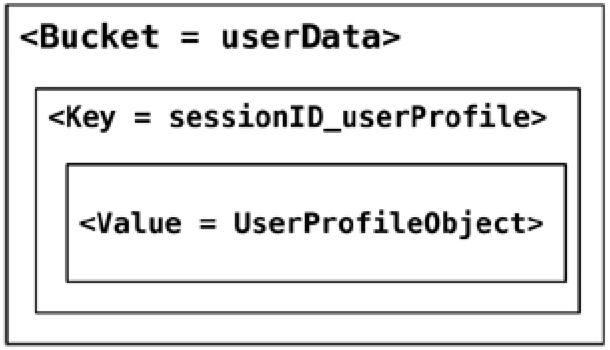
Oracle	Riak
database instance	Riak cluster
table	bucket
row	key-value
rowid	key

namespace for keys

Terminology in Oracle vs. Riak



Single object for all data, everything in a single bucket



Adding type of data to the key, still everything in a single bucket

Separate buckets for different types of data

KEY-VALUE STORE

EXAMPLE

```
Bucket bucket = getBucket(bucketName);
IRiakObject riakObject =
    bucket.store(key, value).execute();
```

```
Bucket bucket = getBucket(bucketName);
IRiakObject riakObject =
    bucket.fetch(key).execute();
byte[] bytes = riakObject.getValue();
String value = new String(bytes);
```

KEY-VALUE STORE

SUITABLE USE CASES

Storing Session Information

- Every web session is assigned a unique session_id value
- Everything about the session can be stored by a single PUT request or retrieved using a single GET
- Fast, everything is stored in a single object

User Profiles, Preferences

- Every user has a unique user_id, user_name + preferences such as language, colour, time zone, which products the user has access to, …
- As in the previous case:
  - Fast, single object, single GET/PUT

Shopping Cart Data

- Similar to the previous cases

KEY-VALUE STORE

WHEN NOT TO USE

Relationships among Data

- Relationships between different sets of data
- Some key-value stores provide link-walking features
  - Not usual

Multioperation Transactions

- Saving multiple keys
  - Failure to save any one of them → revert or roll back the rest of the operations

Query by Data

- Search the keys based on something found in the value part

Operations by Sets

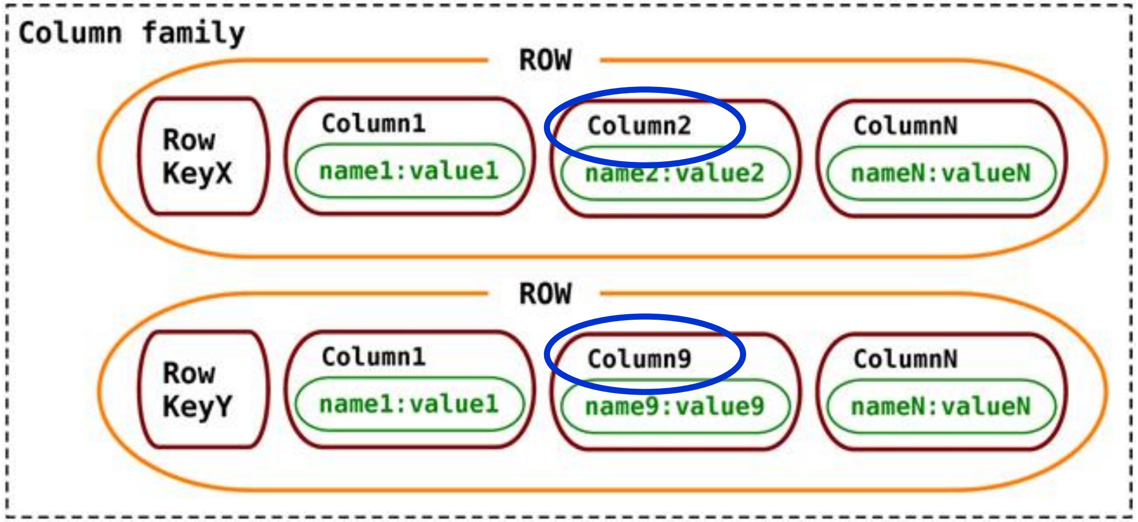
- Operations are limited to one key at a time
- No way to operate upon multiple keys at the same time

COLUMN DATABASES

COLUMN-FAMILY STORES

BASIC CHARACTERISTICS

- Also "columnar" or "column-oriented"
- Column families = rows that have many columns associated with a row key
- Column families are groups of related data that is often accessed together
  - e.g., for a customer we access all profile information at the same time, but not orders



COLUMN-FAMILY STORES
REPRESENTATIVES

Google's BigTable

H·BASE

Cassandra

HYPERTABLE

SimpleDB

APACHE CASSANDRA

cassandra

- Developed at Facebook
- Initial release: 2008
- Stable release: 2013
  - Apache Licence
- Written in: Java
- OS: cross-platform
- Operations:
  - CQL (Cassandra Query Language)
  - MapReduce support
    - Can cooperate with Hadoop (data storage instead of HDFS)

http://cassandra.apache.org/

CASSANDRA
TERMINOLOGY

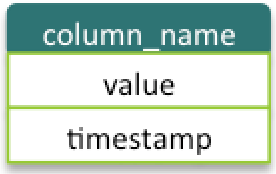
RDBMS	Cassandra
database instance	cluster
database	keyspace
table	column family
row	row
column (same for all rows)	column (can be different per row)

Usually one per application

- Column = basic unit, consists of a **name-value** pair
  - Name serves as a key
  - Stored with a **timestamp** (expired data, resolving conflicts, …)

3-tuple

- Row = a collection of columns attached or linked to a key
- Column family = a collection of similar rows
  - Rows do not have to have the same columns



CASSANDRA

DATA MODEL – EXAMPLE

```
{ "pramod-sadalage" : {
    firstName: "Pramod",
    lastName: "Sadalage",
    lastVisit: "2012/12/12" }
  "martin-fowler" : {
    firstName: "Martin",
    lastName: "Fowler",
    location: "Boston" } }
```

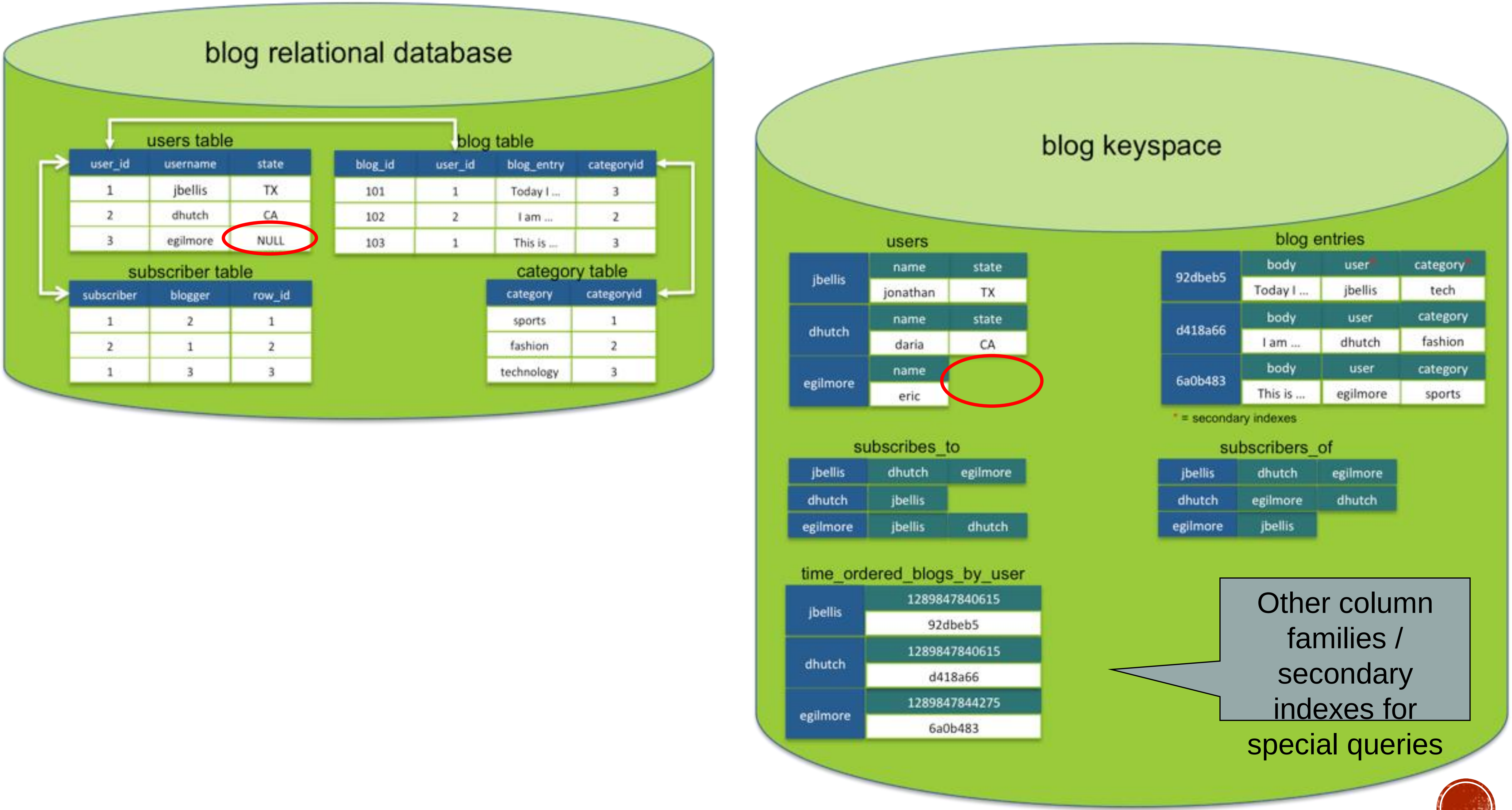
- pramod-sadalage row and martin-fowler row with different columns; both rows are a part of a column family

```
{ name: "firstName",
  value: "Martin",
  timestamp: 12345667890 }
```

- Column key of firstName and the value of Martin

CASSANDRA
COLUMN-FAMILIES VS. RELATIONS

- We do not need to model all of the columns up front
  - Each row is not required to have the same set of columns
  - Usually we assume similar sets of columns
    - Related data
    - Can be extended when needed
- No formal foreign keys
  - Joining column families at query time is usually not supported
  - We need to pre-compute the query / use a secondary index


blog relational database
users table
user_id username state
1 jbellis TX
2 dhutch CA
3 egilmore NULL
blog table
blog_id user_id blog_entry categoryid
101 1 Today I ... 3
102 2 I am ... 2
103 1 This is ... 3
subscriber table
subscriber blogger row_id
1 2 1
2 1 2
1 3 3
category table
category categoryid
sports 1
fashion 2
technology 3
blog keyspace
users
jbellis name state jonathan TX
dhutch name state daria CA
egilmore name eric
blog entries
92dbeb5 body user* category* Today I ... jbellis tech
d418a66 body user category I am ... dhutch fashion
6a0b483 body user category This is ... egilmore sports
* = secondary indexes
subscribes_to
jbellis dhutch egilmore
dhutch jbellis
egilmore jbellis dhutch
subscribers_of
jbellis dhutch egilmore
dhutch egilmore dhutch
egilmore jbellis
time_ordered_blogs_by_user
jbellis 1289847840615 92dbeb5
dhutch 1289847840615 d418a66
egilmore 1289847844275 6a0b483
Other column families / secondary indexes for special queries

CASSANDRA

COLUMN-FAMILIES

- Can define metadata about columns
  - Actual columns of a row are determined by client application
  - Each row can have a different set of columns
- Static – similar to a relational database table
  - Rows have the same set of columns
  - Not required to have all of the columns defined
- Dynamic – takes advantage of Cassandra's ability to use arbitrary application-supplied column names
  - Pre-computed result sets
  - Stored in a single row for efficient data retrieval
  - Row = a snapshot of data that satisfy a given query
    - Like a materialized view

CASSANDRA

COLUMN-FAMILIES

static

row key	columns …			
jbellis	name	email	address	state
	jonathan	jb@ds.com	123 main	TX
dhutch	name	email	address	state
	daria	dh@ds.com	45 2nd St.	CA
egilmore	name	email		
	eric	eg@ds.com		

dynamic

row key	columns …			
jbellis	dhutch	egilmore	datastax	mzcassie
dhutch	egilmore			
egilmore	datastax	mzcassie		

Users that subscribe to a particular user's blog

CASSANDRA

WORKING WITH A TABLE – SET

order

CREATE TABLE users (
user_id text PRIMARY KEY,
first_name text,
last_name text,
emails set<text>);

```
user_id | emails
---------+-----------------------------------------------------------------
 frodo   | {"baggins@caramail.com","f@baggins.com","fb@friendsofmordor.org"}
```

INSERT INTO users (user_id, first_name, last_name, emails)
VALUES('frodo', 'Frodo', 'Baggins', **{**'f@baggins.com', 'baggins@gmail.com'**}**);

UPDATE users SET emails = emails **+** **{**'fb@friendsofmordor.org'**}**
WHERE user_id = 'frodo';

SELECT user_id, emails FROM users WHERE user_id = 'frodo';

UPDATE users SET emails = emails - **{**'fb@friendsofmordor.org'**}**
WHERE user_id = 'frodo';

UPDATE users SET emails = **{}** WHERE user_id = 'frodo';

CASSANDRA

WORKING WITH A TABLE – LIST

ALTER TABLE users ADD top_places **list<text>**;

UPDATE users SET top_places = **[** 'rivendell', 'rohan' **]**
WHERE user_id = 'frodo';

UPDATE users SET top_places = **[** 'the shire' **]** **+** top_places
WHERE user_id = 'frodo';

UPDATE users SET top_places = top_places **+** **[** 'mordor' **]**
WHERE user_id = 'frodo';

UPDATE users SET top_places**[2]** = 'riddermark'
WHERE user_id = 'frodo';

DELETE top_places**[3]** FROM users WHERE user_id = 'frodo';

UPDATE users SET top_places = top_places - **[**'riddermark'**]**
WHERE user_id = 'frodo';

CASSANDRA

WORKING WITH A TABLE – MAP

```
ALTER TABLE users ADD todo map<timestamp, text>;

UPDATE users SET todo = { '2012-9-24' : 'enter mordor',
'2012-10-2 12:00' : 'throw ring into mount doom' }
WHERE user_id = 'frodo';

UPDATE users SET todo['2012-10-2 12:00'] =
'throw my precious into mount doom'
WHERE user_id = 'frodo';

INSERT INTO users (user_id, todo) VALUES ('frodo', {
'2013-9-22 12:01' : 'birthday wishes to Bilbo',
'2013-10-1 18:00' : 'Check into Inn of Prancing Pony' });

DELETE todo['2012-9-24'] FROM users
WHERE user_id = 'frodo';
```

CASSANDRA
WORKING WITH A TABLE

DROP TABLE timeline;
- Delete a table including all data

TRUNCATE timeline;
- Remove all data from a table

CREATE INDEX userIndex ON timeline (posted_by);
- Create a (secondary) index
- Allow efficient querying of other columns than key

DROP INDEX userIndex;
- Drop an index

CASSANDRA
QUERYING

- No joins, just simple conditions
  - For simple data reads

SELECT * FROM users
WHERE firstname = 'jane' and lastname='smith'
ALLOW FILTERING;

- Filtering (WHERE)

SELECT * FROM emp
WHERE empID IN (130,104)
ORDER BY deptID DESC;

- Ordering (ORDER BY)

CASSANDRA
QUERYING

SELECT select_expression
FROM keyspace_name.table_name
WHERE relation **AND** relation ...
GROUP BY columns
ORDER BY (clustering_key (ASC | DESC)...)
LIMIT n
ALLOW FILTERING

- select_expression:
  - List of columns
  - DISTINCT
  - COUNT
  - Aliases (AS)
  - TTL(column_name)
  - WRITETIME(column_name)

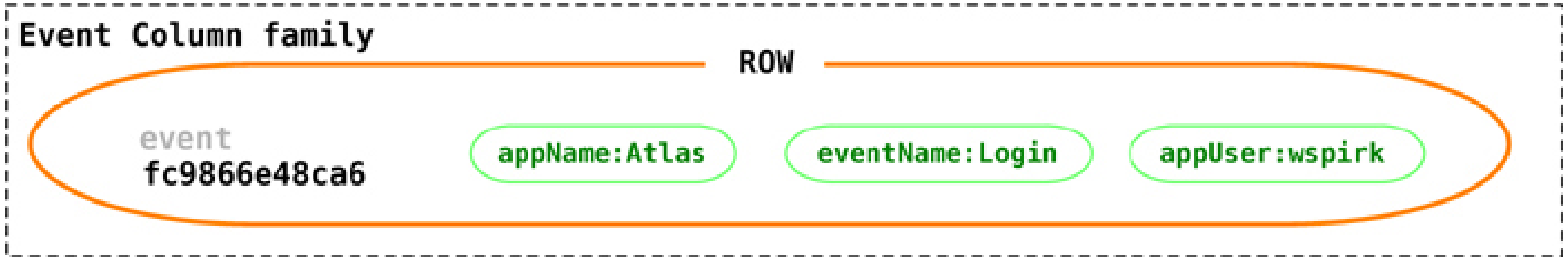
- relation:
  - column_name (= | < | > | <= | >=) key_value
  - column_name IN ((key_value,...))
  - (column_name, ...) (= | < | > | <= | >=)
  - (term | TOKEN (term, ...))

hash

- term:
  - constant
  - set/list/map

COLUMN-FAMILY STORES

SUITABLE USE CASES



Event Logging

- Ability to store any data structures → good choice to store event information

Content Management Systems, Blogging Platforms

- We can store blog entries with tags, categories, links, and trackbacks in different columns
- Comments can be either stored in the same row or moved to a different keyspace
- Blog users and the actual blogs can be put into different column families

COLUMN-FAMILY STORES

WHEN NOT TO USE

Systems that Require ACID Transactions

- Column-family stores are <u>not</u> just a special kind of RDBMSs with variable set of columns!

Aggregation of the Data Using Queries

- (Such as SUM or AVG)
- Have to be done on the client side

For Early Prototypes

- We are not sure how the query patterns may change
- As the query patterns change, we have to change the column family design

DOCUMENT DATABASES

Document Databases

Basic Characteristics

- Documents are the main concept
  - Stored and retrieved
  - XML, JSON, …
- Documents are
  - Self-describing
  - Hierarchical tree data structures
  - Can consist of maps, collections (lists, sets, …), scalar values, nested documents, …
- Documents in a collection are expected to be similar
  - Their schema can differ
- Document databases store documents in the value part of the key-value store
  - Key-value stores where the value is examinable

DOCUMENT DATABASES
DATA – EXAMPLE

```
{ "firstname": "Martin",
  "likes": [ "Biking",
             "Photography" ],
  "lastcity": "Boston",
  "lastVisited": }
```

```
{ "firstname": "Pramod",
  "citiesvisited": [ "Chicago", "London", "Pune", "Bangalore" ],
  "addresses": [
    { "state": "AK",
      "city": "DILLINGHAM",
      "type": "R"    },
    { "state": "MH",
      "city": "PUNE",
      "type": "R" }  ],
  "lastcity": "Chicago" }
```

DOCUMENT DATABASES
DATA – EXAMPLE

- Data are similar, but have differences, e.g., in attribute names
  - Still belong to the same collection
- We can represent
  - A list of cities visited as an array
  - A list of addresses as a list of documents embedded inside the main document

DOCUMENT DATABASES

REPRESENTATIVES

Lotus Notes
Storage Facility

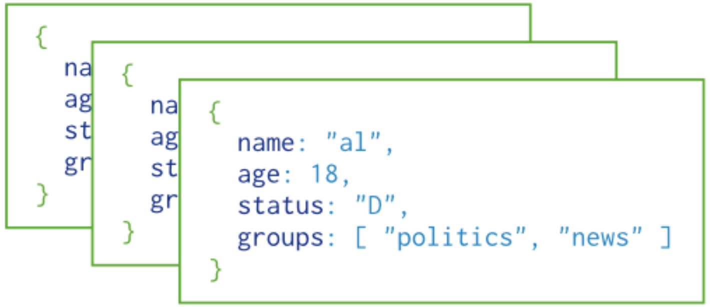
MONGODB

- Initial release: 2009
- Written in C++
  - Open-source
- Cross-platform
- JSON documents
  - Dynamic schemas
- Features:
  - High performance – indices
  - High availability – replication + eventual consistency + automatic failover
  - Automatic scaling – automatic sharding across the cluster
  - MapReduce support

http://www.mongoDB.org/

MONGODB

TERMINOLOGY



Collection

Oracle	MongoDB
database instance	MongoDB instance
schema	database
table	collection
row	document
rowid	_id
join	DBRef

Terminology in Oracle and mongoDB

- Each mongoDB instance has multiple databases
- Each database can have multiple collections
- When we store a document, we have to choose database and collection

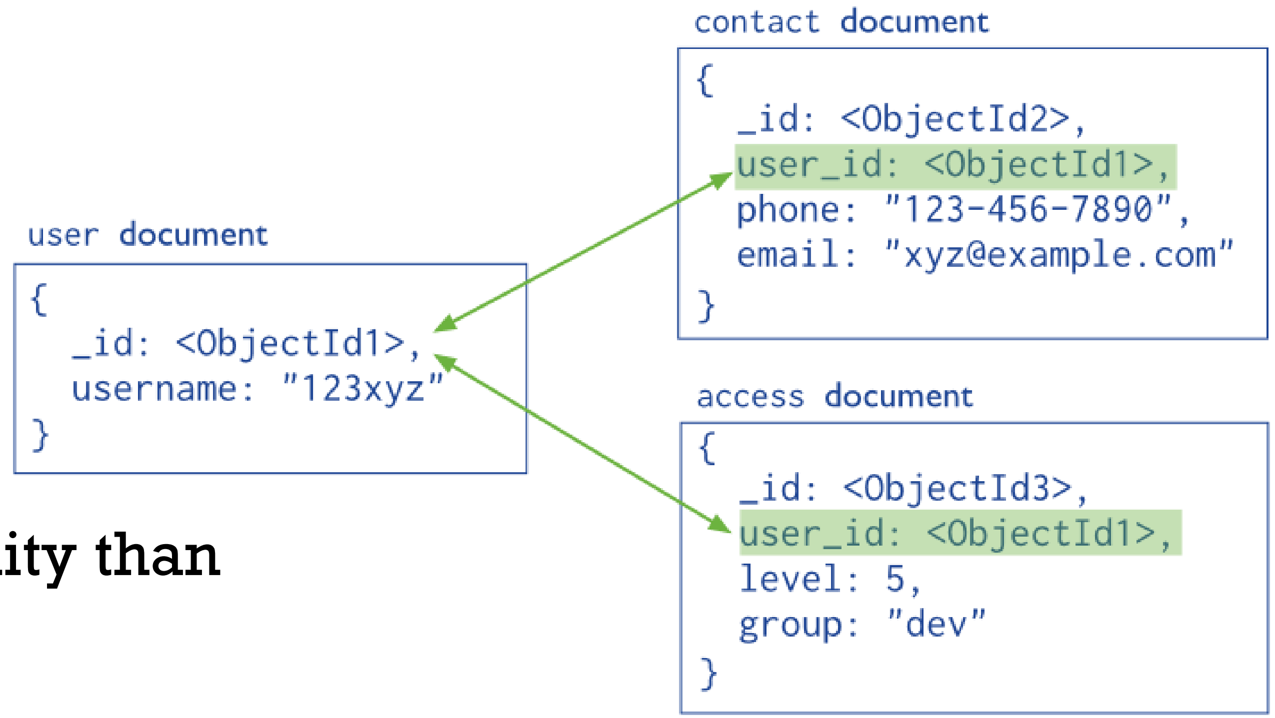
MONGODB

DOCUMENTS

- Use JSON
- Stored as BSON
  - Binary representation of JSON
- Have maximum size: 16MB (in BSON)
  - Not to use too much RAM
  - GridFS tool divides larger files into fragments
- Restrictions on field names:
  - `_id` is reserved for use as a primary key
    - Unique in the collection
    - Immutable
    - Any type other than an array
  - The field names cannot start with the `$` character
    - Reserved for operators
  - The field names cannot contain the `.` character
    - Reserved for accessing fields

MONGODB

DATA MODEL – REFERENCES



- References provide more flexibility than embedding
- Use normalized data models:
  - When embedding would result in duplication of data not outweighted by read performance
  - To represent more complex many-to-many relationships
  - To model large hierarchical data sets
- Disadvantages:
  - Can require more roundtrips to the server (follow up queries)

MONGODB

DATA MODEL – EMBEDDED DATA

- Related data in a single document structure
  - Documents can have subdocuments (in a field of array)
  - Applications may need to issue less queries
- Denormalized data models
- Allow applications to retrieve and manipulate related data in a single database operation

```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
              phone: "123-456-7890",
              email: "xyz@example.com"
            },
  access: {
             level: 5,
             group: "dev"
          }
}
```

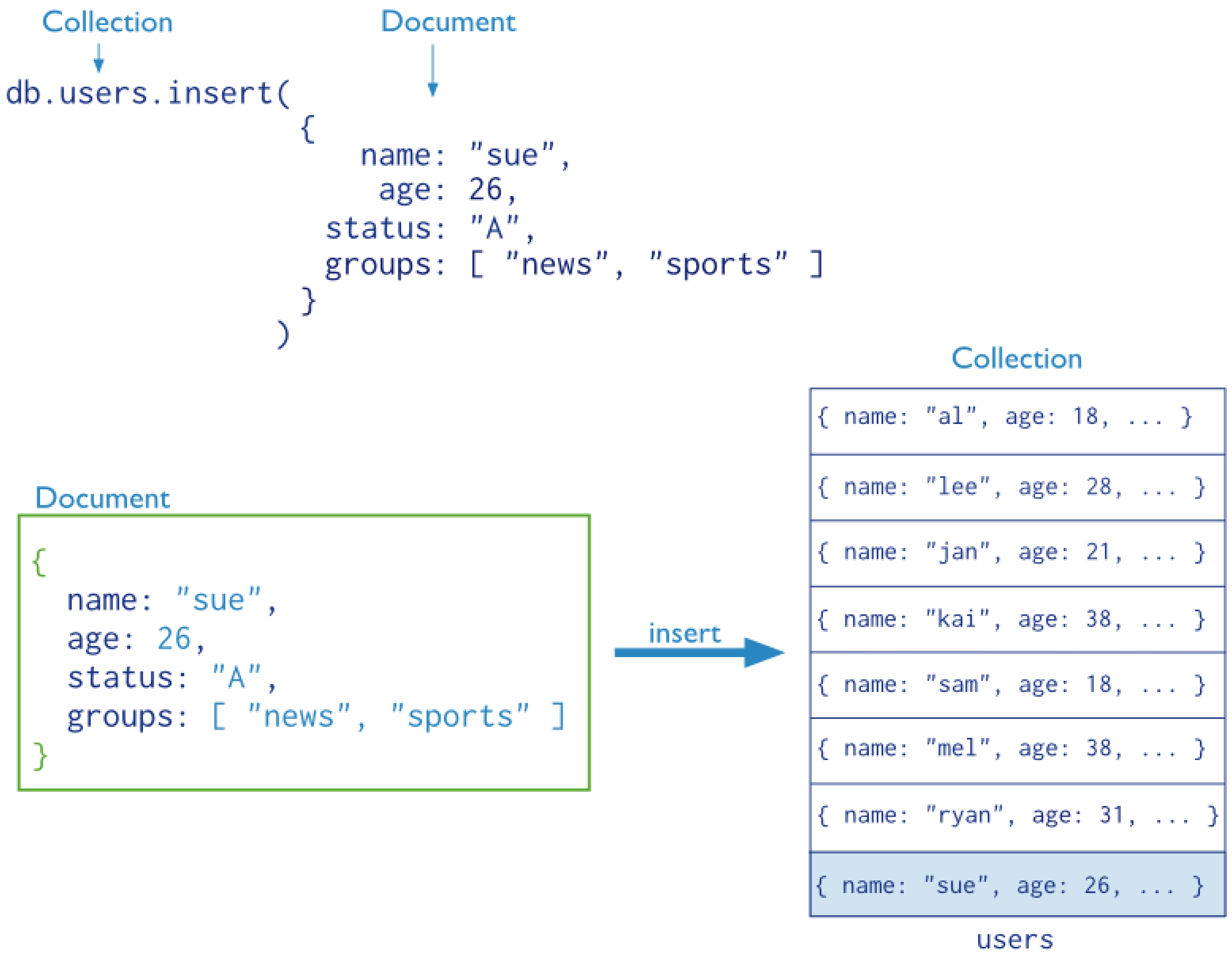
Embedded sub-document (contact)

Embedded sub-document (access)

MONGODB

DATA MODIFICATION

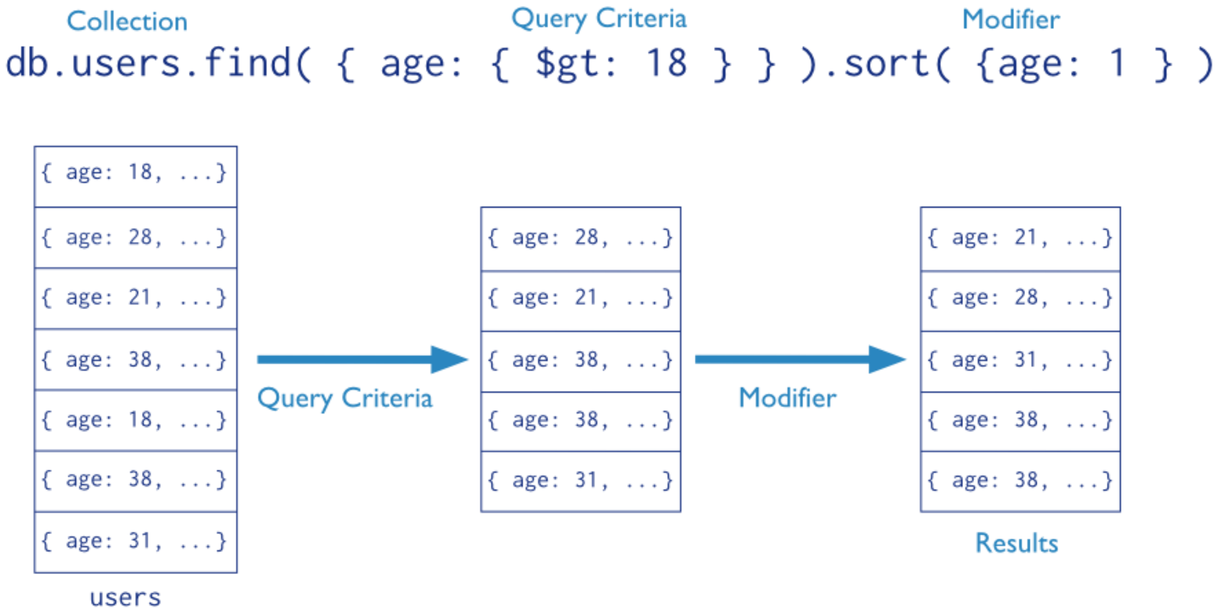
- Operations: create, update, delete
  - Modify the data of a single collection of documents
- For update / delete: criteria to select the documents to update / remove



MONGODB

QUERY

- Targets a specific collection of documents
- Specifies criteria that identify the returned documents
- May include a projection that specifies the fields from the matching documents to return
- May impose limits, sort orders, …



DOCUMENT DATABASES

SUITABLE USE CASES

Event Logging

- Many different applications want to log events
  - Type of data being captured keeps changing
- Events can be sharded (i.e. divided) by the name of the application or type of event

Content Management Systems, Blogging Platforms

- Managing user comments, user registrations, profiles, web-facing documents, …

Web Analytics or Real-Time Analytics

- Parts of the document can be updated
- New metrics can be easily added without schema changes
  - E.g. adding a member of a list, set,…

E-Commerce Applications

- Flexible schema for products and orders
- Evolving data models without expensive data migration

DOCUMENT DATABASES

WHEN NOT TO USE

Complex Transactions Spanning Different Operations

- Atomic cross-document operations
  - Some document databases do support (e.g., RavenDB)

Queries against Varying Aggregate Structure

- Design of aggregate is constantly changing → we need to save the aggregates at the lowest level of granularity
  - i.e. to normalize the data

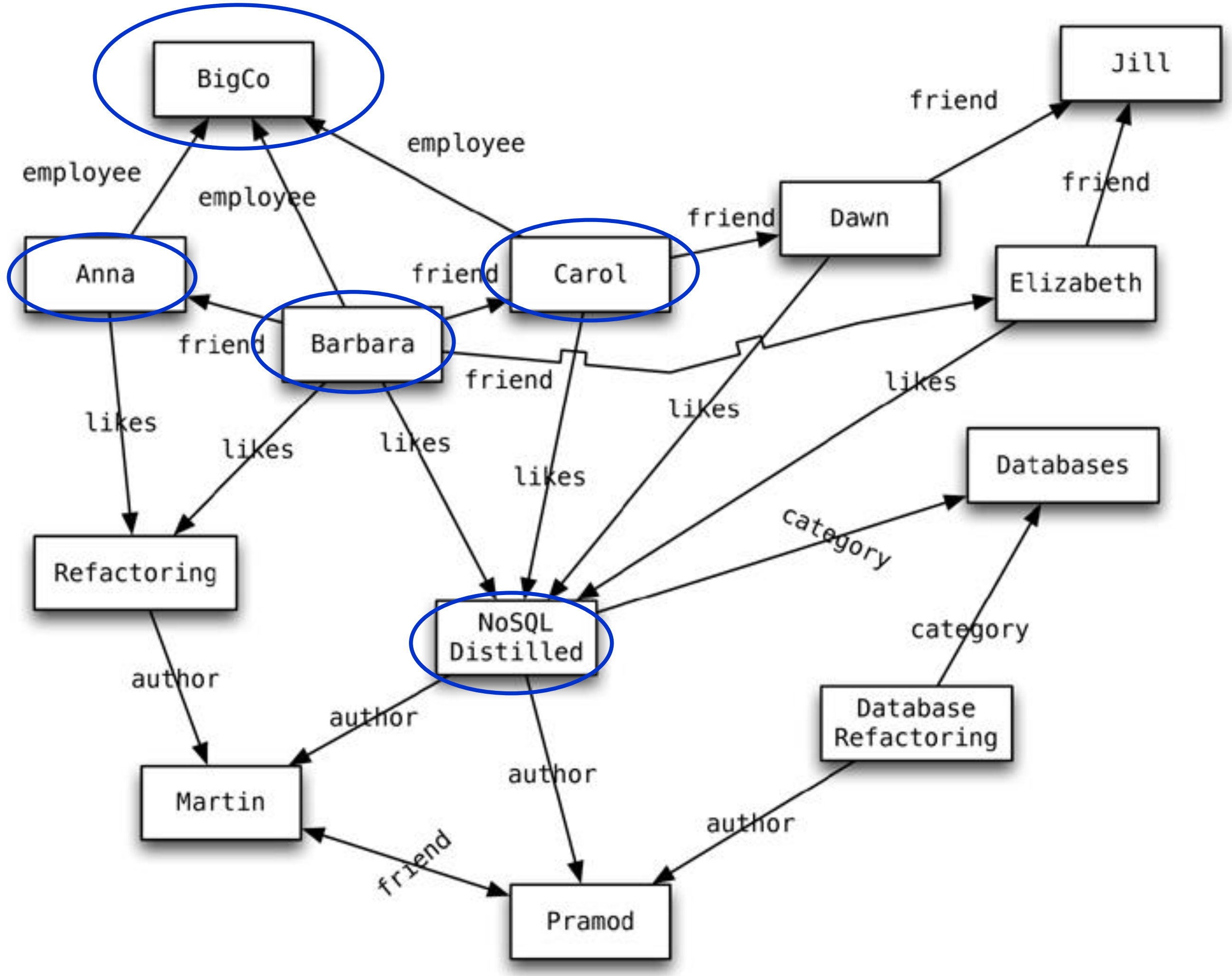
GRAPH DATABASES

GRAPH DATABASES

BASIC CHARACTERISTICS

- To store entities and relationships between these entities
  - Node is an instance of an object
  - Nodes have properties
    - e.g., name
  - Edges have directional significance
  - Edges have types
    - e.g., likes, friend, …
- Nodes are organized by relationships
  - Allow to find interesting patterns
  - e.g., "Get all people (= nodes in the graph) employed by Big Co that like (book called) NoSQL Distilled"

EXAMPLE:



BigCo
Jill
friend
employee
employee
employee
friend
Dawn
friend
friend
Anna
Carol
Elizabeth
friend
Barbara
friend
friend
likes
likes
likes
likes
likes
likes
likes
Databases
category
Refactoring
NoSQL Distilled
category
author
author
Database Refactoring
author
Martin
author
friend
Pramod

GRAPH DATABASES

RDBMS VS. GRAPH DATABASES

- When we store a graph-like structure in RDBMS, it is for a single type of relationship
  - "Who is my manager"
- Adding another relationship usually means a lot of schema changes
- In RDBMS we model the graph beforehand based on the Traversal we want
  - If the Traversal changes, the data will have to change
  - In graph databases the relationship is not calculated at query time but persisted

GRAPH DATABASES
REPRESENTATIVES

Neo4j
the graph database

InfiniteGraph
The Distributed Graph Database for the Cloud and Beyond

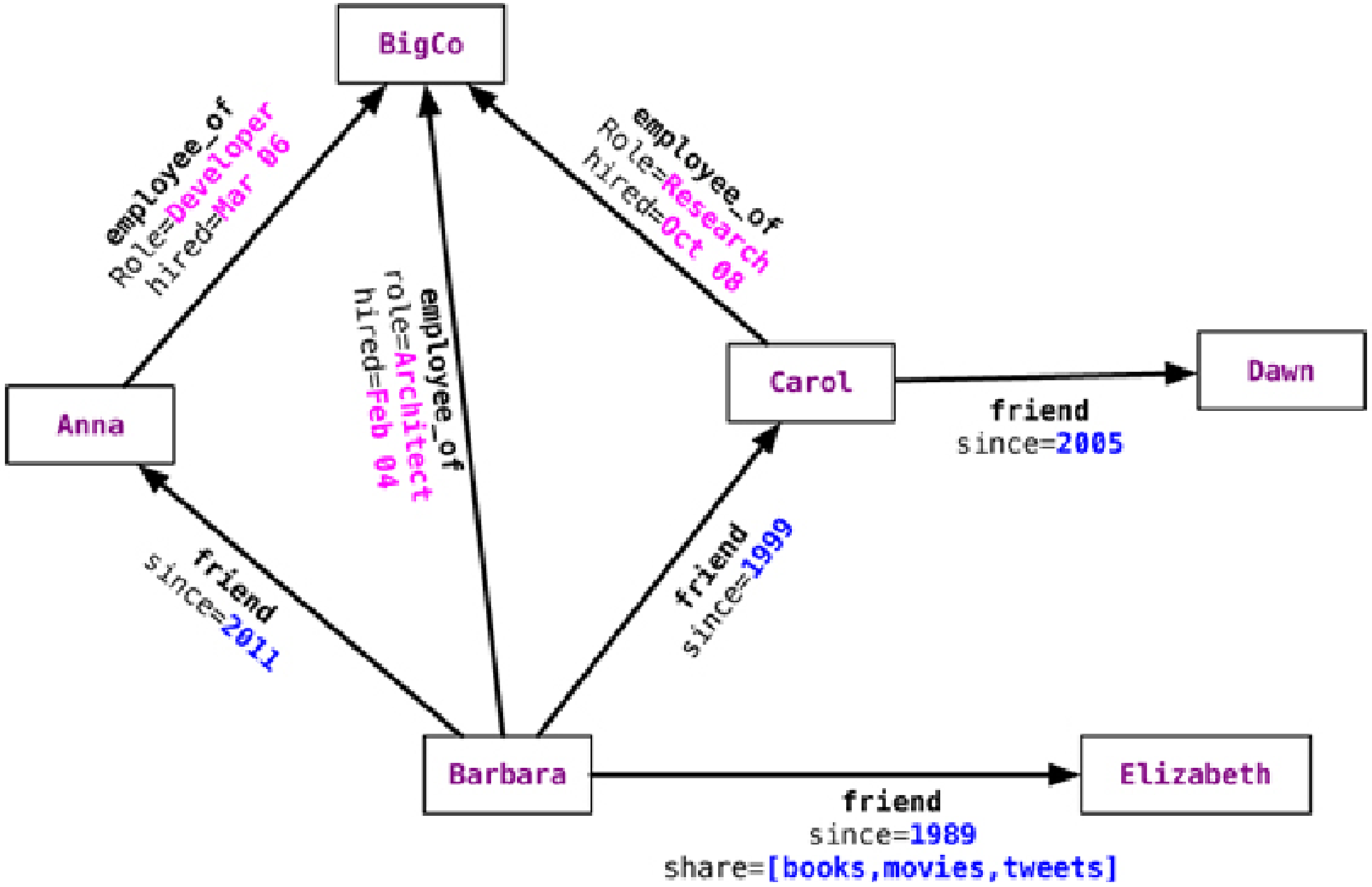
OrientDB®

FlockDB

GRAPH DATABASES
BASIC CHARACTERISTICS

- Nodes can have different types of relationships between them
  - To represent relationships between the domain entities
  - To have secondary relationships
    - Category, path, time-trees, quad-trees for spatial indexing, linked lists for sorted access, …
- There is no limit to the number and kind of relationships a node can have
  - Except for upper limits of a particular system, if any
- Relationships have type, start node, end node, own properties
  - e.g., since when did they become friends

EXAMPLE:



BigCo
Anna
Barbara
Carol
Dawn
Elizabeth
employee_of
Role=Developer
hired=Mar 06
employee_of
role=Architect
hired=Feb 04
employee_of
Role=Research
hired=Oct 08
friend
since=2005
friend
since=2011
friend
since=1999
friend
since=1989
share=[books,movies,tweets]

EXAMPLE: NEO4J

```
Node martin = graphDb.createNode();
martin.setProperty("name", "Martin");
Node pramod = graphDb.createNode();
pramod.setProperty("name", "Pramod");

martin.createRelationshipTo(pramod, FRIEND);
pramod.createRelationshipTo(martin, FRIEND);
```

- We have to create a relationship between the nodes in both directions
  - Nodes know about INCOMING and OUTGOING relationships

GRAPH DATABASES

QUERY

- Properties of a node/edge can be indexed
- Indices are queried to find the starting node to begin a traversal

creating index

adding nodes

```
Transaction transaction = graphDb.beginTx();
try {
    Index<Node> nodeIndex = graphDb.index().forNodes("nodes");
    nodeIndex.add(martin, "name", martin.getProperty("name"));
    nodeIndex.add(pramod, "name", pramod.getProperty("name"));
    transaction.success(); }
finally {
    transaction.finish(); }
```

retrieving a node

```
Node martin = nodeIndex.get("name", "Martin").getSingle();
allRelationships = martin.getRelationships();
```

getting all its relationships

GRAPH DATABASES

QUERY – FINDING PATHS

- We are interested in determining if there are multiple paths, finding all of the paths, the shortest path, …

```
Node barbara = nodeIndex.get("name", "Barbara").getSingle();
Node jill    = nodeIndex.get("name", "Jill").getSingle();
PathFinder<Path> finder1 = GraphAlgoFactory.allPaths(
        Traversal.expanderForTypes(FRIEND,Direction.OUTGOING),
                                   MAX_DEPTH);
Iterable<Path> paths = finder1.findAllPaths(barbara, jill);

PathFinder<Path> finder2 = GraphAlgoFactory.shortestPath(
        Traversal.expanderForTypes(FRIEND,Direction.OUTGOING),
                                   MAX_DEPTH);
Iterable<Path> paths = finder2.findAllPaths(barbara, jill);
```

GRAPH DATABASES

SUITABLE USE CASES

Connected Data

- Social networks
- Any link-rich domain is well suited for graph databases

Routing, Dispatch, and Location-Based Services

- Node = location or address that has a delivery
- Graph = nodes where a delivery has to be made
- Relationships = distance

Recommendation Engines

- "your friends also bought this product"
- "when invoicing this item, these other items are usually invoiced"

GRAPH DATABASES

WHEN NOT TO USE

- When we want to update all or a subset of entities
  - Changing a property on all the nodes is not a straightforward operation
  - e.g., analytics solution where all entities may need to be updated with a changed property
- Some graph databases may be unable to handle lots of data
  - Distribution of a graph is difficult

NEWSQL AND ARRAY DATABASES

NEWSQL DATABASES

- Idea (from 2011): scalable storage + all functionality known from traditional relational databases
  - Not just SQL access, but classical relational model, ACID properties, …
  - Previously ScalableSQL

Aslett, M.: *What We Talk about When We Talk about NewSQL*. 452 Group, 2011. http://blogs.the451group.com/information_management/2011/04/06/what-we-talk-about-when-we-talk-about-newsql/

Stonebraker, M.: *New SQL: An Alternative to NoSQL and Old SQL for New OLTP Apps*, 2011. https://cacm.acm.org/blogs/blog-cacm/109710-new-sql-an-alternative-to-nosql-and-old-sql-for-new-oltp-apps/fulltext

NEWSQL DATABASES

- Approaches:
  - Distributed systems which add advantages of relational model + ACID
    - e.g. Clustrix, ScaleArc, MemSQL, VoltDB, …
  - Relational DBMSs extended towards horizontal scalability
    - e.g. TokuDB, JustOne DB, ..
- Cloud: NewSQL as a Service
  - Special type of a cloud service = scalable relational DBMS
    - e.g. Amazon Relational Database Service, Microsoft Azure Database, …

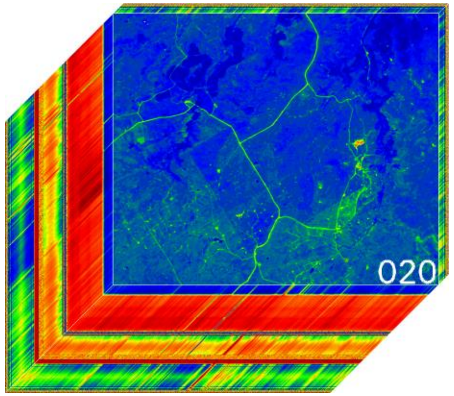
NEWSQL DATABASES

- Why do we need them?
  1. There are applications which work with relational databases + they need to solve new increase of data volumes
     - Transformation to any NoSQL data model would be too expensive
  2. There are application which still need strong data consistency + horizontal scalability
- Consequence: Again NewSQL does not mean the end of traditional SQL (relational) DBMSs
  - An alternative approach – we need alternatives and there will occur other

↓

Stonebraker, M. et al.: *The end of an architectural era: (it's time for a complete rewrite)*. VLDB '07.

ARRAY DATABASES



- Database systems specific for data represented as one- or multi-dimensional arrays
- Usually: We need to represent the respective values in time and/or space
  - Biology, chemistry, physics, geology, …
  - Complex research analyses of natural events
    - e.g. astronomical measurements, changes of climate, satellite pictures of the Earth, oceanographic data, human genome, …
- Example: Each satellite picture is a 2D-array (longitude + latitude) with values informing about the particular positions
  - Next dimensions: time when the picture was taken, characteristics of the tool taking the picture, …

ARRAY DATABASES

- In general:
  - Big Data of a specific type
  - Data <u>not</u> suitable for flat 2D relations
    - Some RDBMSs support arrays
    - Too simple operations for these purposes
      - Not efficient

SciDB

rasdaman
raster data management

MULTI-MODEL DATABASES

POLYGLOT PERSISTENCE

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

PROS AND CONS OF POLYGLOT PERSISTENCE

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

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

MULTI-MODEL DATABASE

- One unified database for multi-model data

MULTI-MODEL DATABASES

ORACLE

mongoDB

MariaDB

APACHE DRILL

ArangoDB

OrientDB

FOUNDATIONDB

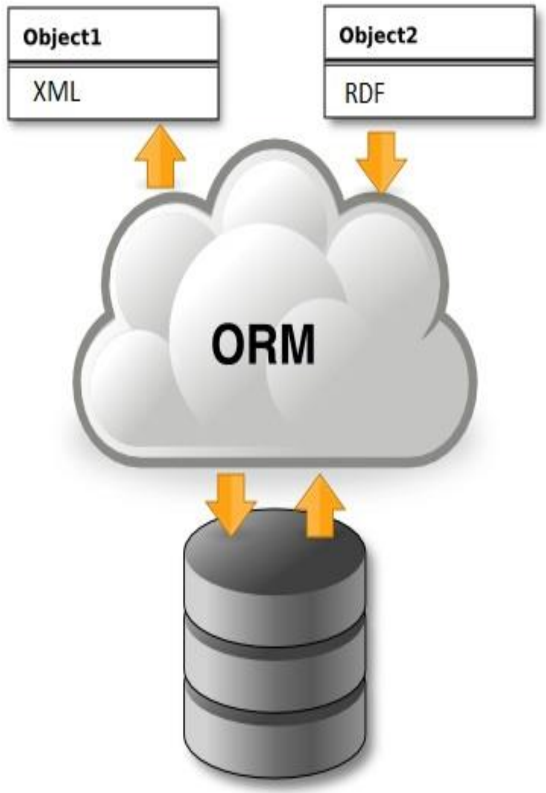
AsterixDB

MarkLogic

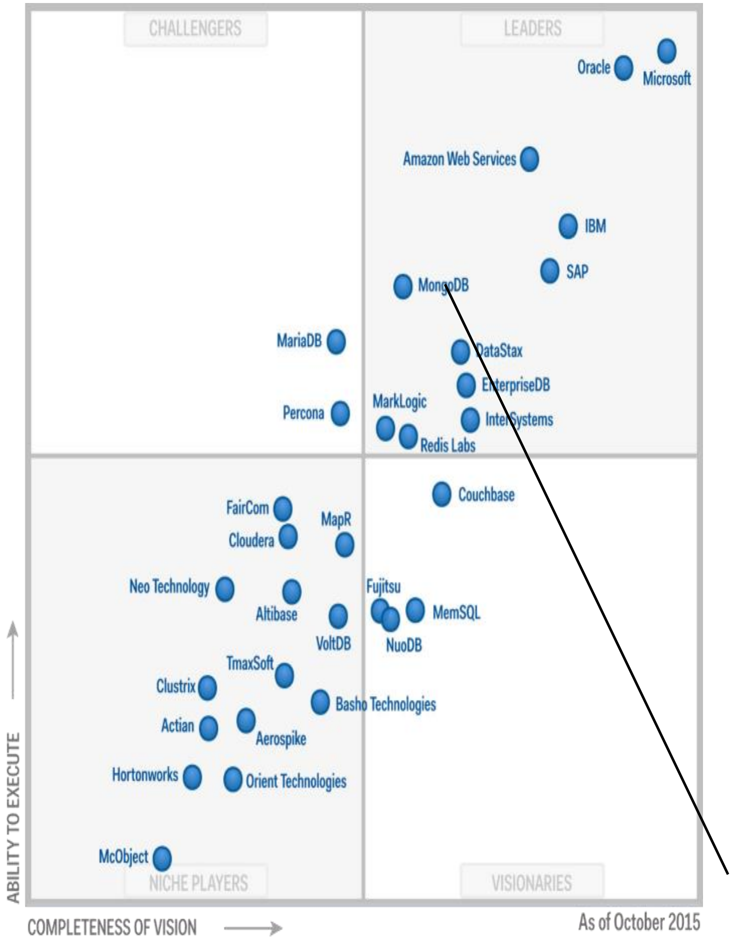
…

MULTI-MODEL DATABASES ARE NOT NEW!

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



MOST OF DBS WILL BECOME MULTI-MODEL DATABASES IN 2017



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

-- Gartner report for operational databases 2016

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

PROS AND CONS OF MULTI-MODEL DATABASES

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

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

TWO EXAMPLES OF MULTI-MODEL DATABASES

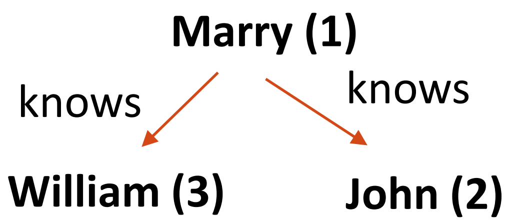
ArangoDB

OrientDB

ArangoDB

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

AN EXAMPLE OF MULTI-MODEL DATA AND QUERY



Social network graph

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

Order JSON document

"1" -- > "34e5e759"

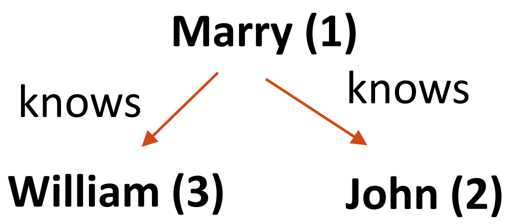
"2"-- > "0c6df508"

Key/value pairs
(Customer_ID , Order_no)

Customer relation

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

AN EXAMPLE OF MULTI-MODEL DATA AND QUERY



Graph-key/value join

"1" -- > "34e5e759"

"2"-- > "0c6df508"

Relation-graph join

Key/value-JSON join

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

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

Recommendation query:

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

AN EXAMPLE OF MULTI-MODEL DATA AND QUERY

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

Recommendation query:

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

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

OrientDB

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

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

CLASSIFICATION OF MULTI-MODEL SYSTEMS

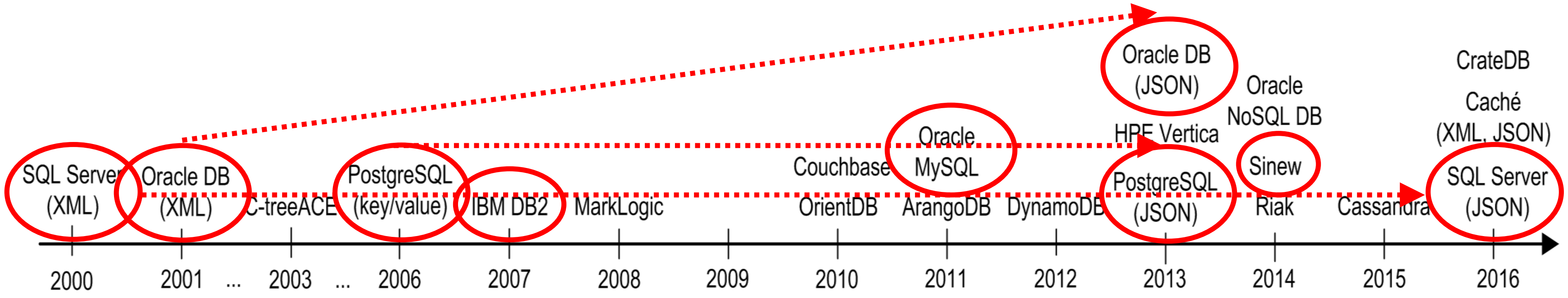
- Basic approach: on the basis of original (or core) data model

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

RELATIONAL MULTI-MODEL DBMSS

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

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



RELATIONAL MULTI-MODEL DBMSS

STORAGE – POSTGRESQL EXAMPLE

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

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

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

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

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

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

jsonb_object_keys text
Order_no
Orderlines
Order_no
Orderlines

RELATIONAL MULTI-MODEL DBMSS

STORAGE – POSTGRESQL EXAMPLE

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

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

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

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

RELATIONAL MULTI-MODEL DBMSS

QUERYING – POSTGRESQL EXAMPLE

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

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

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

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

REFERENCES

- http://nosql-database.org/
- Pramod J. Sadalage – Martin Fowler: NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence
- Eric Redmond – Jim R. Wilson: Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement
- Sherif Sakr – Eric Pardede: Graph Data Management: Techniques and Applications
- Shashank Tiwari: Professional NoSQL
- Neither Fish Nor Fowl: the Rise of Multi-model Databases. The 451 Group, 2013.
- D. Feinberg, M. Adrian, N. Heudecker, A. M. Ronthal, and T. Palanca. Gartner Magic Quadrant for Operational Database Management Systems, 12 October 2015.
- J. Lu, Z. H. Liu, P. Xu, and C. Zhang. UDBMS: road to unification for multi-model data management. CoRR, abs/1612.08050, 2016
- J. Lu: Towards Benchmarking Multi-model Databases. CIDR 2017
- S. Abiteboul et al: Research Directions for Principles of Data Management, Dagstuhl Perspectives Workshop 16151 (2017)