Big Data, Techniques and Platforms
Class 4/8: NoSQL

Francesca Bugiotti

CentraleSupélec

October 26, 2023

Objectives

- Understand what is a Data Model
- Review the Relational Data Model
- Understand what is a NoSQL database
- Assign its own category to a NoSQL database

Plan

Ingredients

Stockage Utils

Usually

Not usually

Sometimes

Usually

Usually

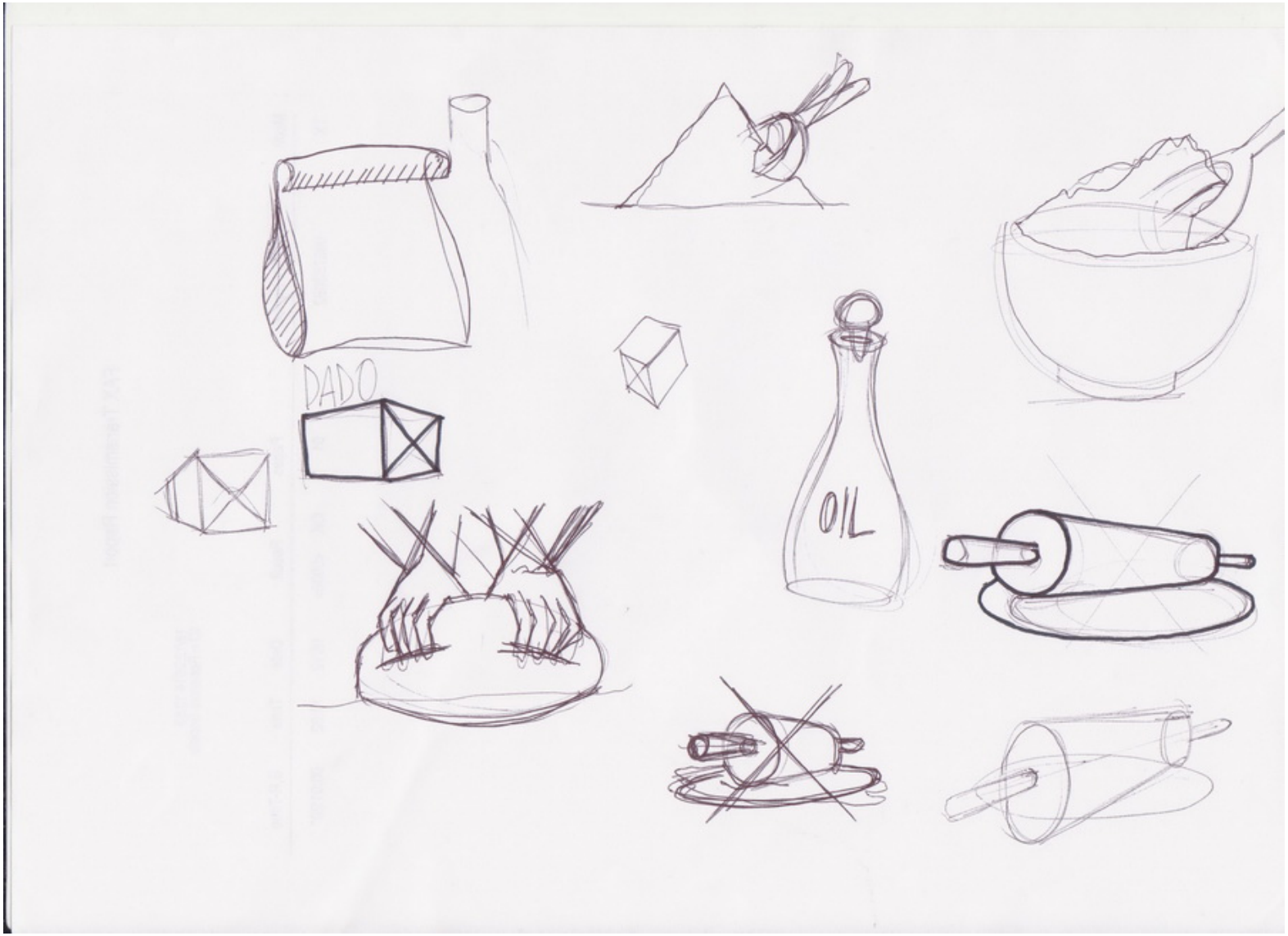

Kneading on the spot
using fresh
ingredients
Let's Pizza
Freshly baked
Ready to eat
Let's Pizza

Usually

Sometimes

Sometimes

Better



They are all pizzas






They are all yests

The point

How much control?

- Do you want simply a result?
  - Be aware: even this is not always so easy in NoSQL era
- Do you want to be able to have intuitions?
- Do you want to control and analyze the whole process?

Variables

How can I control?

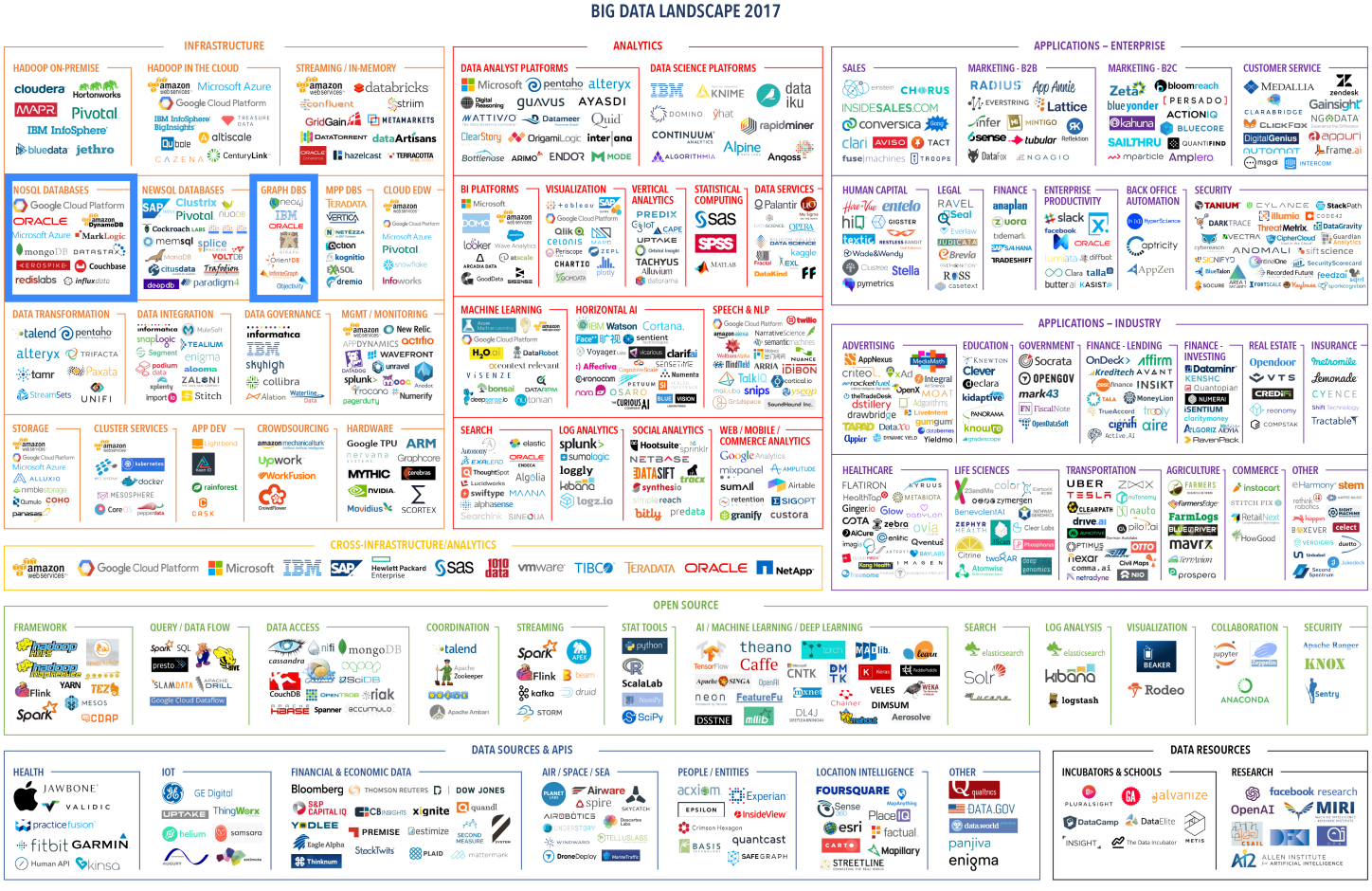
- Knowledge
- Time
- Access

Big data is not just "Big"

Combining data from different sources

- Machines
  - real-time data, sensor data, trackers, streaming, etc.
- People
  - social media, personal documents, etc.
- Organizations
  - structured knowledge bases, data-warehouses, etc.

Technologies



Data management in Big Data

- How to ingest data
- How to store data
- How to ensure data quality
- How to identify the operations to be performed
- How to be efficient
- How to scale up data volume, variety, velocity, and access
- How to keep data secure

Data management in Big Data

- How to store data

Data management in Big Data

- How to store data
- How to scale up data volume, variety, velocity, and access

Big Data = Variety of Data Models

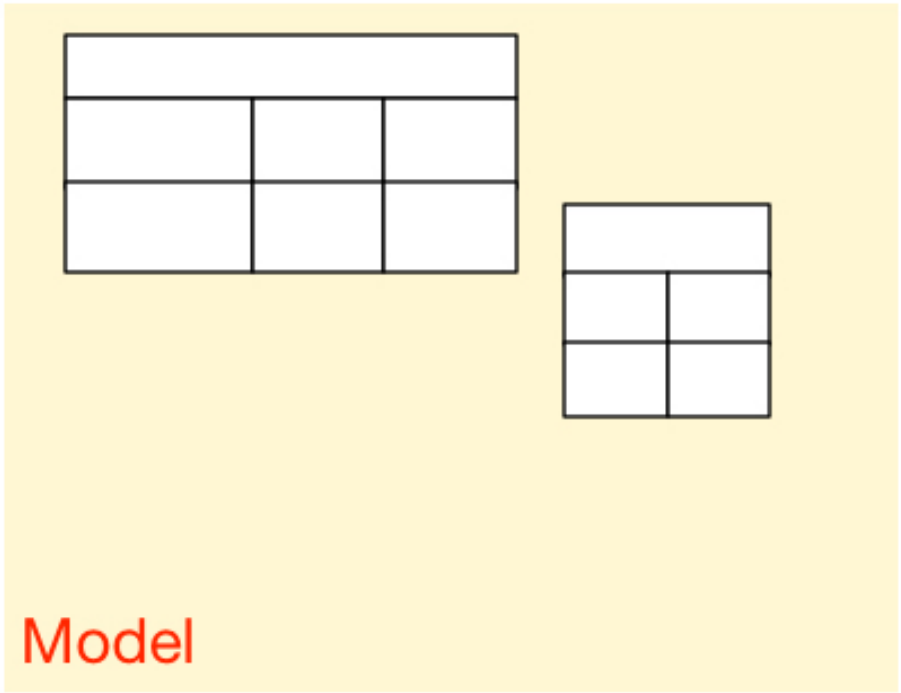
Identify the different data models that are used in the application

What is a data model?

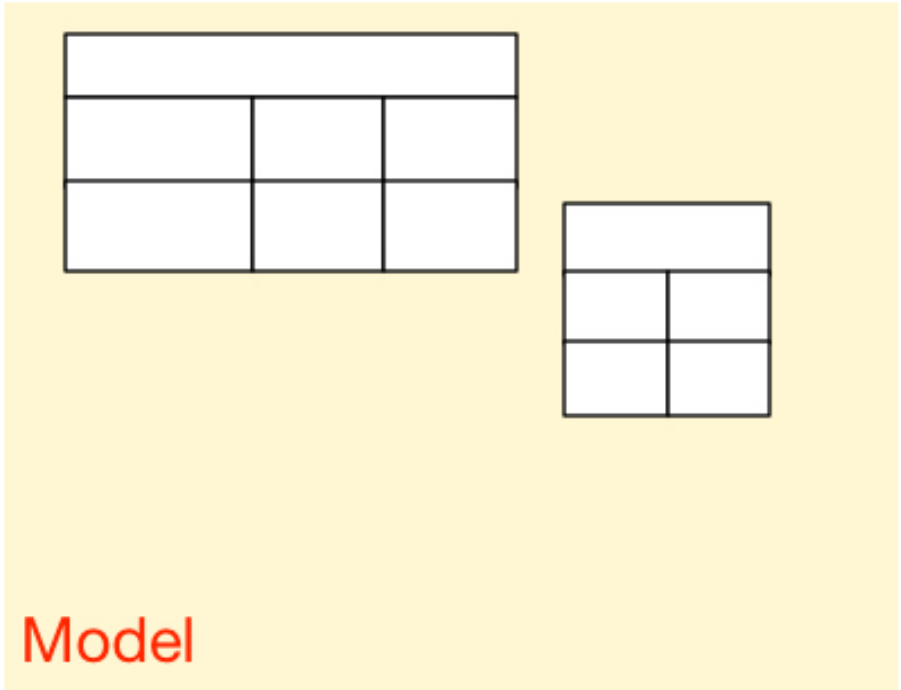
- Structure
- Operations
- Constraints

Plan

Relational Data model



Relational Data model



Structure

- Tables
- Columns

Relational Data model

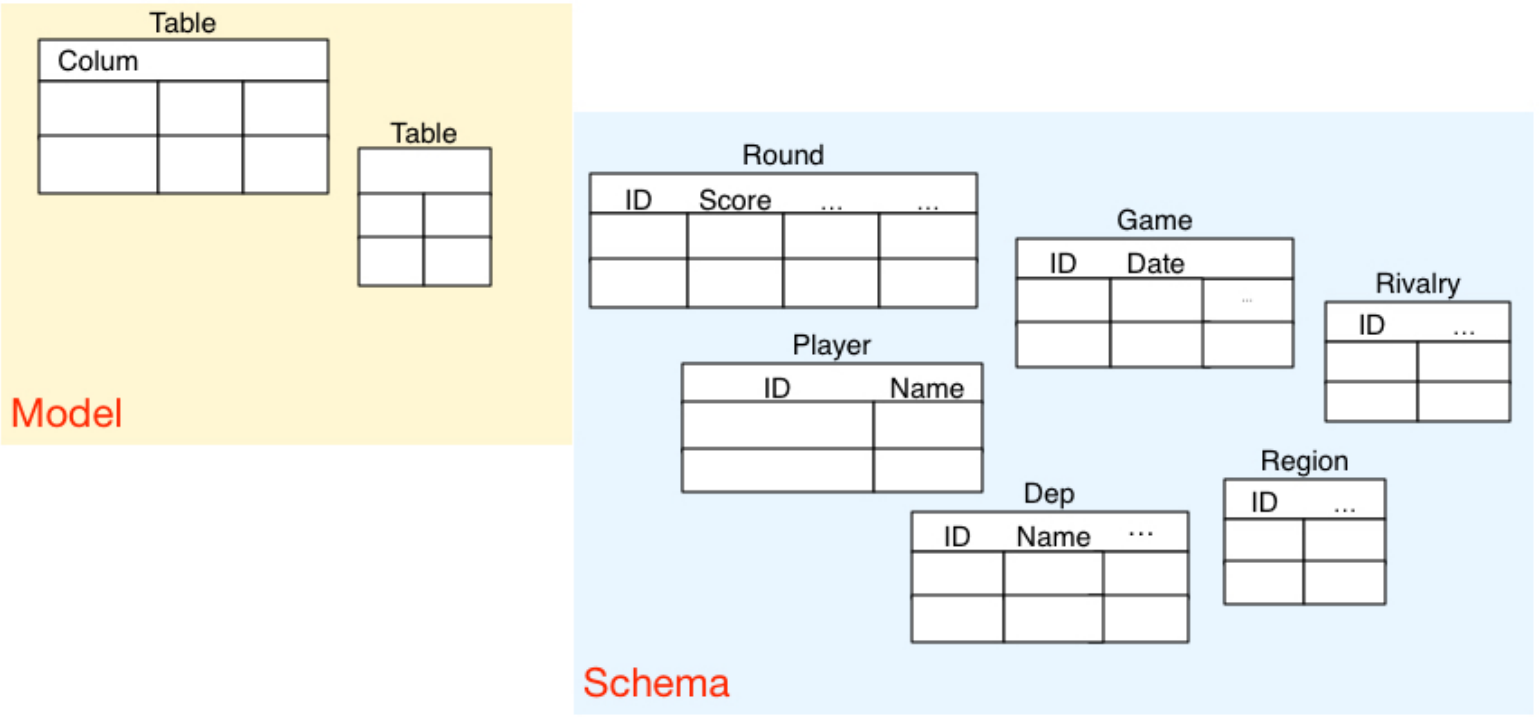
Volges	Epinal	88	01
	...		
Moselle	Metz	51	01

01	Grand Est
03	Occitanie

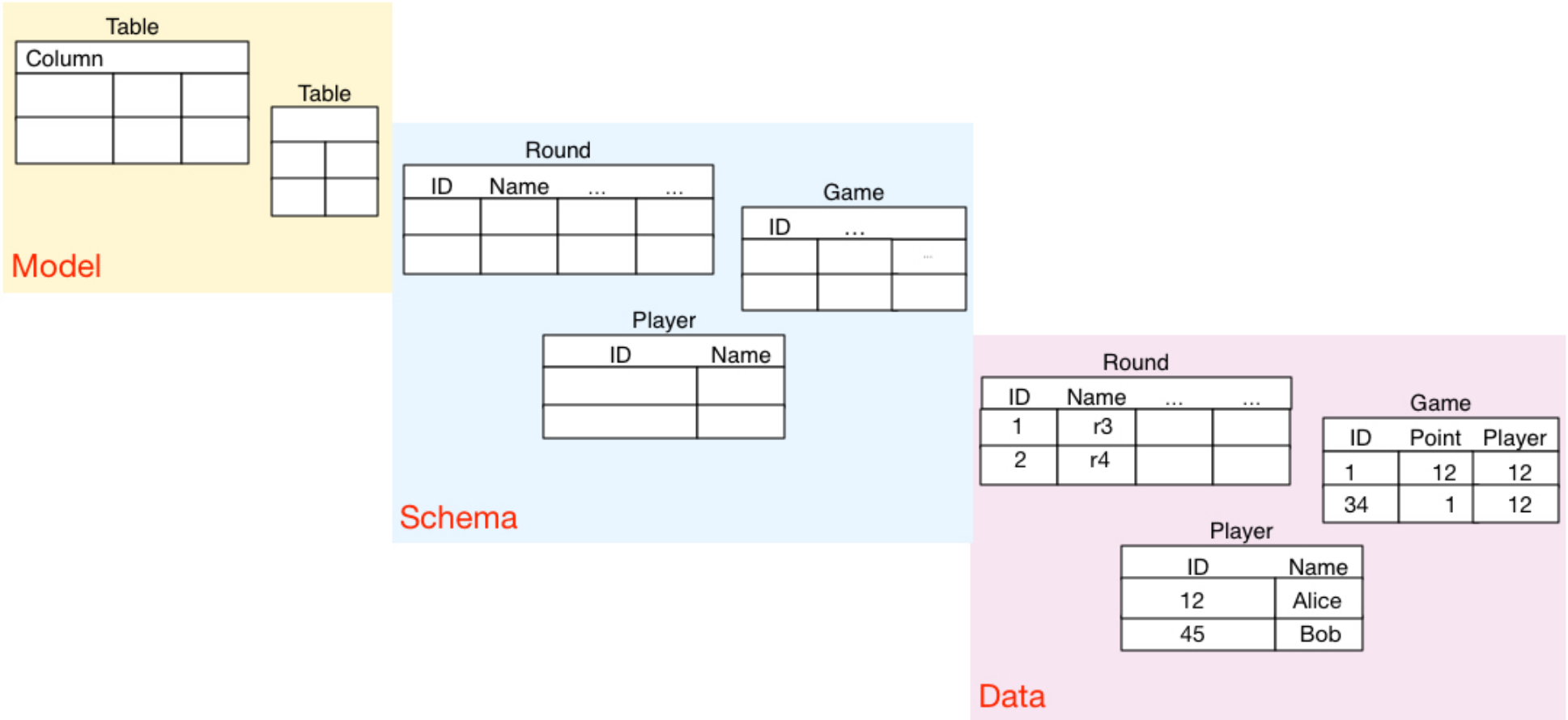
Structure

- Tuples
- Atomic values

Relational Schema



Relational Data



Relational Data model

Relational modeling

It is possible to define tables in different ways but there are rules that govern the relational data modeling

Relational Data model

Game

PNick	PFName	PLName	...	PCity1	PCity1Reg	PCity2	PCity2Reg
Alice	Alice	Foo	…	Paris	Île-de-France	Lyon	Rhône-Alpes
Bob	Bob	Green	…	Nice	Aquitaine	Marseille	Aquitaine
Pt	Peter	Old	…	null	null	null	null

Data

All data in a big table?

Relational Data model

Game

PNick	PFName	PLName	...	PCity1	PCity1Reg	PCity2	PCity2Reg
Alice	Alice	Foo	...	Paris	Île-de-France	Lyon	Rhône-Alpes
Bob	Bob	Green	...	Lyon	Rhône-Alpes	Marseille	Aquitaine
Pt	Peter	Old	...	null	null	null	null

Data

All data in a big table? What happens if..

- we add an address?

Relational Data model

Game

PNick	PFName	PLName	...	PCity1	PCity1Reg	PCity2	PCity2Reg
Alice	Alice	Foo	...	Paris	Île-de-France	Lyon	Rhône-Alpes
Bob	Bob	Green	...	Nice	Aquitaine	Marseille	Aquitaine
Pt	Peter	Old	...	null	null	null	null

Data

All data in a big table? What happens if..

- we add an address?
- some data is missing?

Relational Data model

Game

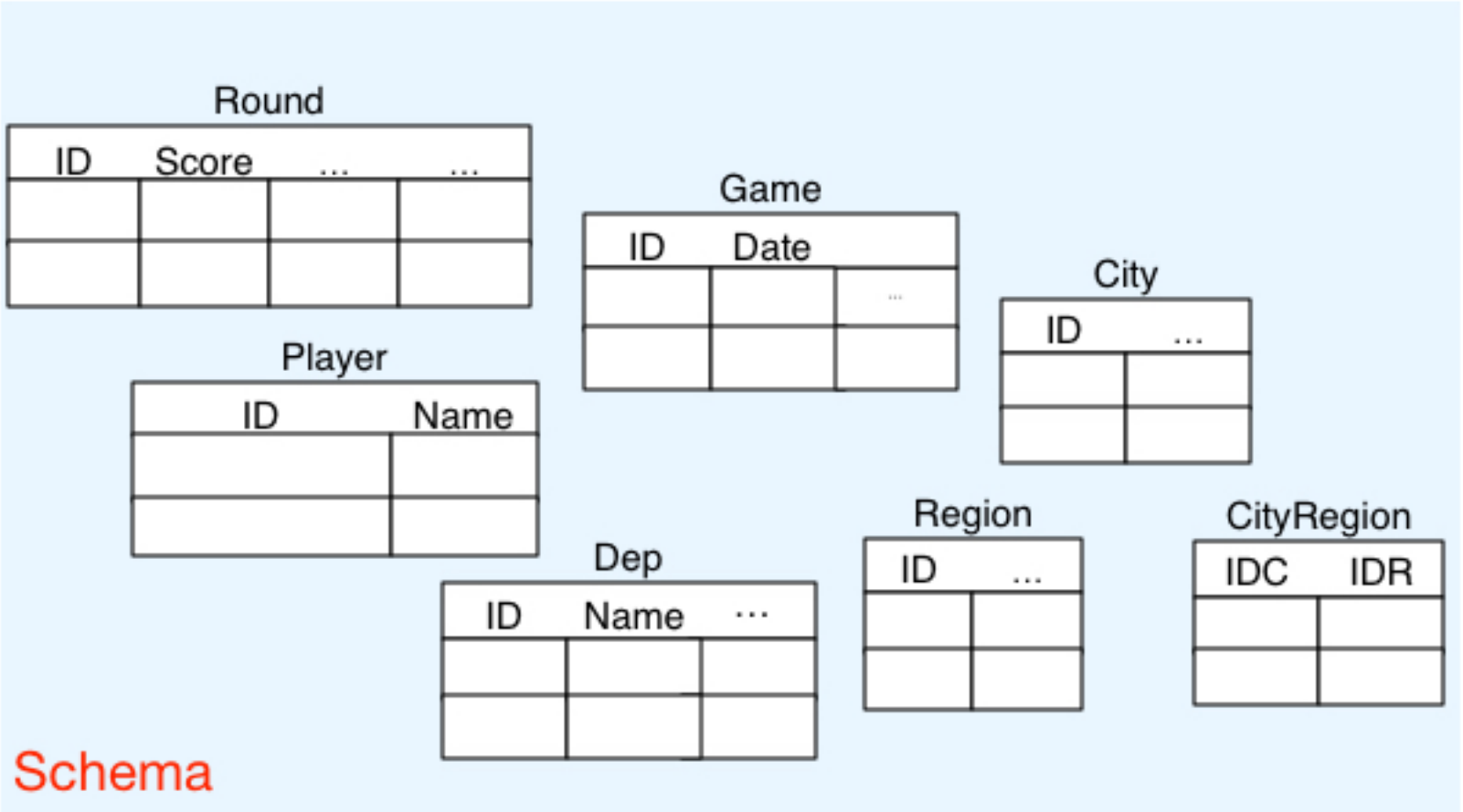
PNick	PFName	PLName	...	PCity1	PCity1Reg	PCity2	PCity2Reg
Alice	Alice	Foo	...	Paris	Île-de-France	Lyon	Auvergne- Rhône-Alpes
Bob	Bob	Green	...	Lyon	Auvergne- Rhône-Alpes	Marseille	Nouvelle-Aquitaine
Pt	Peter	Old	...	null	null	null	null

Data

All data in a big table? What happens if..

- we add an address?
- some data is missing?
- we update data?

Relational Data model



Relational Data model

Conceptual data modeling

- Each independent concept corresponds to a different table
- Constraints across tables that guarantee the data integrity and the links

Relational Data model

Candidate Key

uniquely identifies a tuple within a table. It can be composed by values belonging to one or multiple columns.

- No distinct tuples with the same values for its attributes
- It is minimal

Primary Key

in a DBMS is elected to be used as default tuple identifier for a table.

Round

ID	Score	WordSet	Timing
rdf	45	we4	343
rde	23	3e4	321
fe4	56	4356re	654

Relational Data model

Foreign Key

is a value of a column (or a collection of values) in one table that uniquely identifies a row of another table or the same table

Round

ID	Score	WordSet	Timing	Game
rdf	23	we4	343	g34
...	...	...	...	...
fe4	56	4356re	654	g34
...	...	...	...	...
rde	45	3e4	321	g16

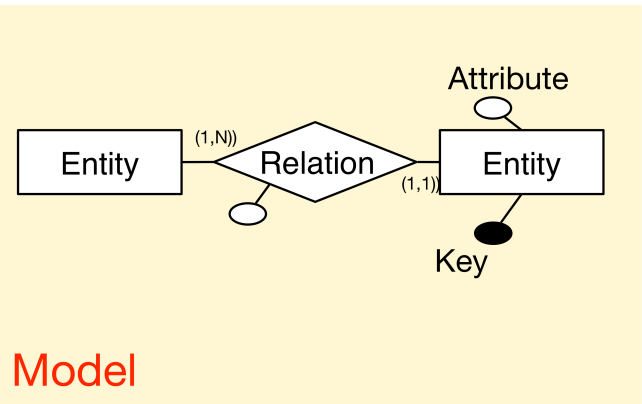
Game

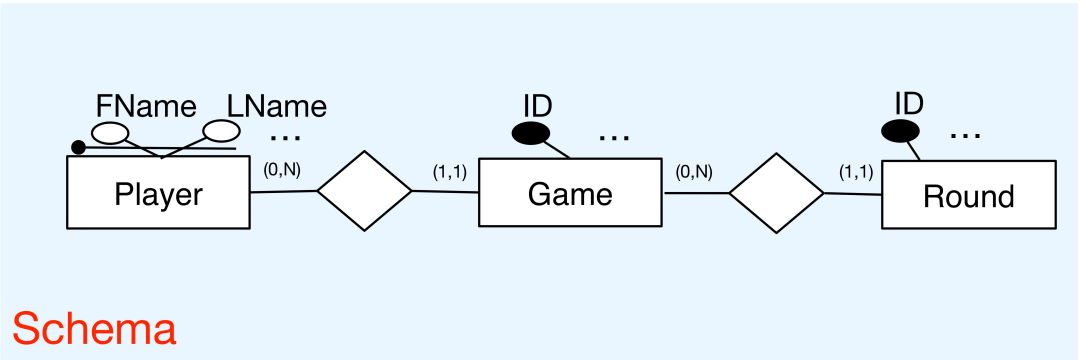
GID	PNick	...	...
g34	goodP	...	...
...	...	...	...
g16	theBest	...	...

Entity Relationship data model

Used to define a conceptual model

- Relationship
- Entity
- Attribute
- Key-attribute/Key-attributes
- Cardinalities





Relational Data model

Operations

Relational algebra operations (Codd [3] 1972) on tables (relations)

- SELECT: extracts tuples from a relation that satisfy a given restriction
- PROJECT: extracts specified attributes (columns) from a relation
- PRODUCT: builds the Cartesian product of two relations
- UNION: builds the set-theoretic union of two tables
- INTERSECT: builds the set-theoretic intersection of two tables
- DIFFERENCE: builds the set difference of two tables
- JOIN: connects two tables by their common attributes

Relational Data model

SQL Query Language

Structured Query Language (SQL) is the standard language for handling Relational data

- Data Definition Language
- Data Manipulation Language
- Data Control Language

Data Definition Language

Commands

```
CREATE TABLE (...)
ALTER TABLE (...)
DROP  TABLE (...)
...
```

To create, modify, delete, etc. a table.

Example

Table creation

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ...
);
```

Example

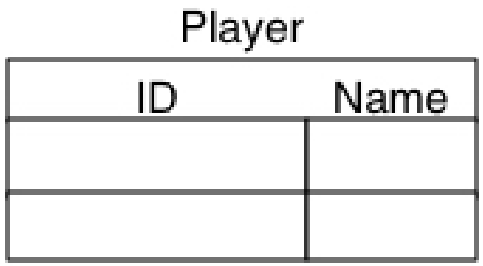
Table creation

```
CREATE TABLE Player
(
ID int,
name varchar(32)
);
```

Example

Table creation

```
CREATE TABLE Player
(
ID int,
name varchar(32)
);
```



Data Manipulation

```
SELECT (...)
DELETE (...)
UPDATE (...)
INSERT (...)
...
```

Example

Insert Data

```
INSERT INTO table_name (column1, column2, ...)
VALUES (value1, value2, value3, ...);
```

Example

Insert Data

```
INSERT INTO table_name (column1, column2, ...)
VALUES (value1, value2, value3, ...);

INSERT INTO Round (ID, Score, Wordset, Timing)
VALUES (rdf, 45, we4, 343);
```

Round

ID	Score	WordSet	Timing

Example

Insert Data

```
INSERT INTO table_name (column1, column2, ...)
VALUES (value1, value2, value3, ...);

INSERT INTO Round (ID, Score, Wordset, Timing)
VALUES (rdf, 45, we4, 343);
```

Round

ID	Score	WordSet	Timing

Round

ID	Score	WordSet	Timing
rdf	45	we4	343

Example

Data Manipulation

```
SELECT column1, column2, ...
FROM table_name;
```

Example

Data Manipulation

```
SELECT column1, column2, ...
FROM table_name;

SELECT *
  FROM Round
```

Round

ID	Score	WordSet	Timing
rdf	45	we4	343
rde	23	3e4	321
fe4	56	4356re	654

Example

Data Manipulation

```
SELECT ID, Score, Timing
  FROM Round
```

Round

ID	Score	WordSet	Timing
rdf	45	we4	343
rde	23	3e4	321
fe4	56	4356re	654

Example

Data Manipulation

```
SELECT ID, Score, Timing
  FROM Round
```

Round

ID	Score	WordSet	Timing
rdf	45	we4	343
rde	23	3e4	321
fe4	56	4356re	654

Round

ID	Score		Timing
rdf	45		343
rde	23		321
fe4	56		654

Example

Data Manipulation

```
SELECT ID, Score, Timing
  FROM Round
  WHERE Score > 40
```

Round

ID	Score	WordSet	Timing
rdf	45	we4	343
rde	23	3e4	321
fe4	56	4356re	654

Example

Data Manipulation

```
SELECT ID, Score, Timing
  FROM Round
  WHERE Score > 40
```

Round

ID	Score	WordSet	Timing
rdf	45	we4	343
rde	23	3e4	321
fe4	56	4356re	654

Round

ID	Score		Timing
rdf	45		343
fe4	56		654

Example

Data Manipulation

```
SELECT ID, Timing
  FROM Round,
  WHERE Score > 40
```

Round

ID	Score	WordSet	Timing
rdf	45	we4	343
rde	23	3e4	321
fe4	56	4356re	654

Example

Data Manipulation

```
SELECT ID, Timing
  FROM Round,
  WHERE Score > 40
```

Round

ID	Score	WordSet	Timing
rdf	45	we4	343
rde	23	3e4	321
fe4	56	4356re	654

Round

ID			Timing
rdf			343
fe4			654

Example

Data Manipulation

```
SELECT ID, Timing
  FROM Round,
  WHERE Score > 40,
  AND Timing < 400
```

Round

ID	Score	WordSet	Timing
rdf	45	we4	343
rde	23	3e4	321
fe4	56	4356re	654

Example

Data Manipulation

```
SELECT ID, Timing
  FROM Round,
  WHERE Score > 40,
  AND Timing < 400
```

Round

ID	Score	WordSet	Timing
rdf	45	we4	343
rde	23	3e4	321
fe4	56	4356re	654

Round

ID			Timing
rdf			343

Example

Data Manipulation

```
SELECT ID, GID, PNick
  FROM Round, Game
  WHERE Round.Game = Game.GID
```

Round

ID	Score	WordSet	Timing	Game
rdf	23	we4	343	g34
...	...	...	...	...
fe4	56	4356re	654	g34
...	...	...	...	...
rde	45	3e4	321	g16

Game

GID	PNick	...	...
g34	goodP	...	...
...	...	...	...
g16	theBest	...	...

Example

Data Manipulation

```
SELECT ID, GID, PNick
  FROM Round, Game
  WHERE Round.Game = Game.GID
```

Round

ID	Score	WordSet	Timing	Game
rdf	23	we4	343	g34
...	...	...	...	...
fe4	56	4356re	654	g34
...	...	...	...	...
rde	45	3e4	321	g16

Game

GID	PNick	...	...
g34	goodP	...	...
...	...	...	...
g16	theBest	...	...

ID	GID	PNick
rdf	23	goodP
...	...	...
fe4	56	goodP
...	...	...
rde	45	theBest

Example

Data Manipulation

```
SELECT ID, GID, PNick
  FROM Round
  INNER JOIN Game ON Round.Game=Game.GID;
```

Round

ID	Score	WordSet	Timing	Game
rdf	23	we4	343	g34
...	...	...	...	...
fe4	56	4356re	654	g34
...	...	...	...	...
rde	45	3e4	321	g16

Game

GID	PNick	...	...
g34	goodP	...	...
...	...	...	...
g16	theBest	...	...

Example

Data Manipulation

```
SELECT ID, GID, PNick
  FROM Round
  INNER JOIN Game ON Round.Game=Game.GID;
```

Round

ID	Score	WordSet	Timing	Game
rdf	23	we4	343	g34
...	...	...	...	...
fe4	56	4356re	654	g34
...	...	...	...	...
rde	45	3e4	321	g16

Game

GID	PNick	...	...
g34	goodP	...	...
...	...	...	...
g16	theBest	...	...

ID	GID	PNick
rdf	23	goodP
...	...	...
fe4	56	goodP
...	...	...
rde	45	theBest

Examples

Data Control

```
CREATE TABLE Player
(
ID int,
name varchar(32) NOT NULL,
...
);
```

```
CREATE TABLE Player
(
ID int,
name varchar(32) NOT NULL,
...
PRIMARY KEY (ID)
);
```

And at last, what is a Database?

According to the classical literature a database is a

- Organized
- Shared
- Persistent
- Big

collection of data

And a Database Management System?

A database management system (DBMS) is a computer software system

Provides utilities for:

- Creating/managing large collections of data
- Accessing data collections
- Managing persistence
- Handling concurrent access
- Assuring reliability and privacy

Plan

Why?

In the last thirty years relational databases have been the default choice for data storage

- Winner with respect to multiple alternatives during the years:
  - deductive databases in the 1980's
  - object databases in the 1990's
  - XML databases in the 2000's

Why?

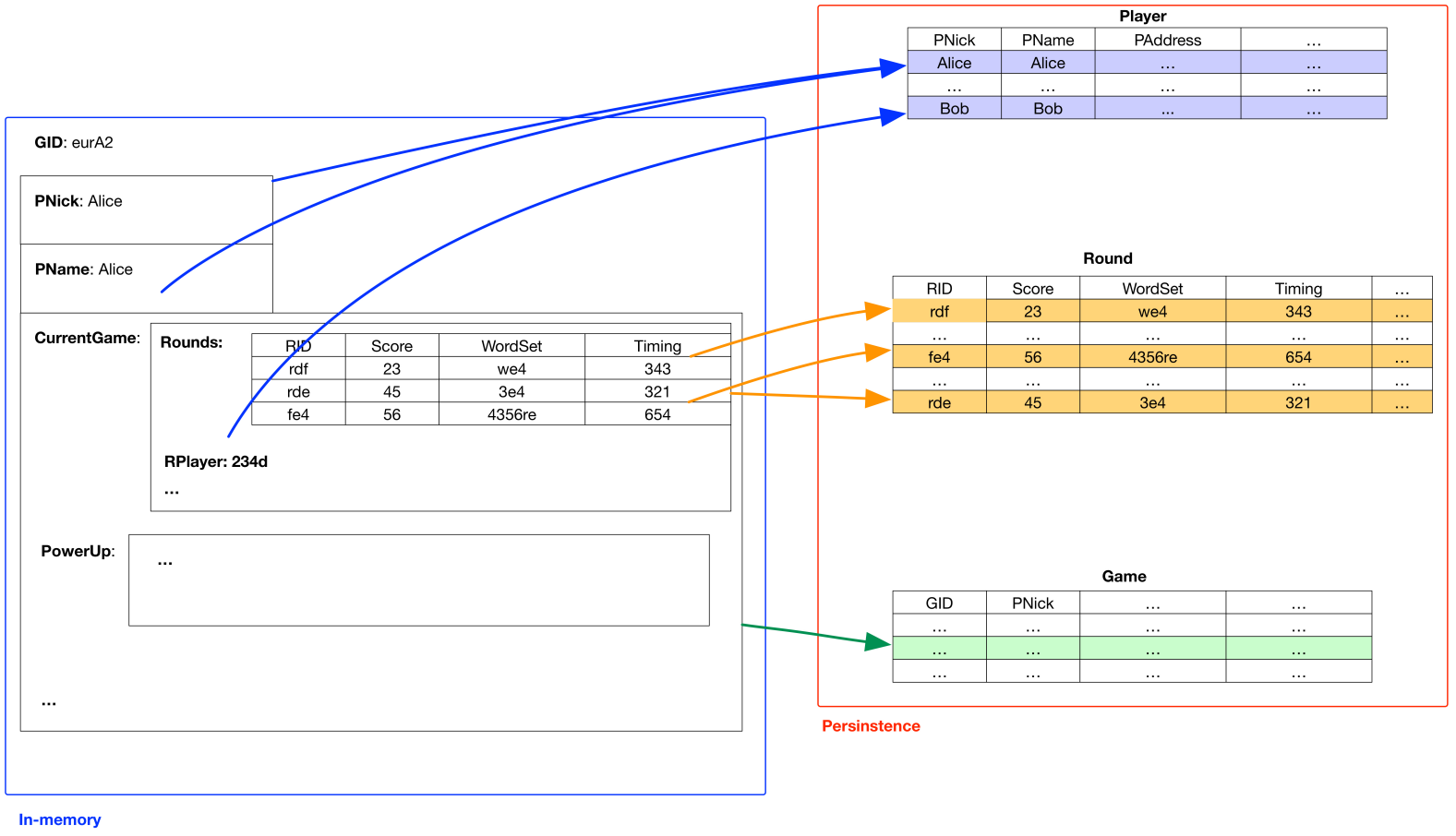
In the last thirty years relational databases have been the default choice for data storage

- Winner with respect to multiple alternatives during the years:
  - deductive databases in the 1980's
  - object databases in the 1990's
  - XML databases in the 2000's
- A Data Scientist starting a new project:
  - which Relational database to use?
    - DB2, Oracle, SQLServer, MySQL, PostgreSQL, etc.
  - often no choice
    - Licenses
    - Triggers
    - Know-how

Provided Values

- Effective and efficient management of persistent data
- Concurrency control
- Data integration
- Standard data model
- Standard query language

Impedance Mismatch



Difference between the persistent data model and the in-memory data structures

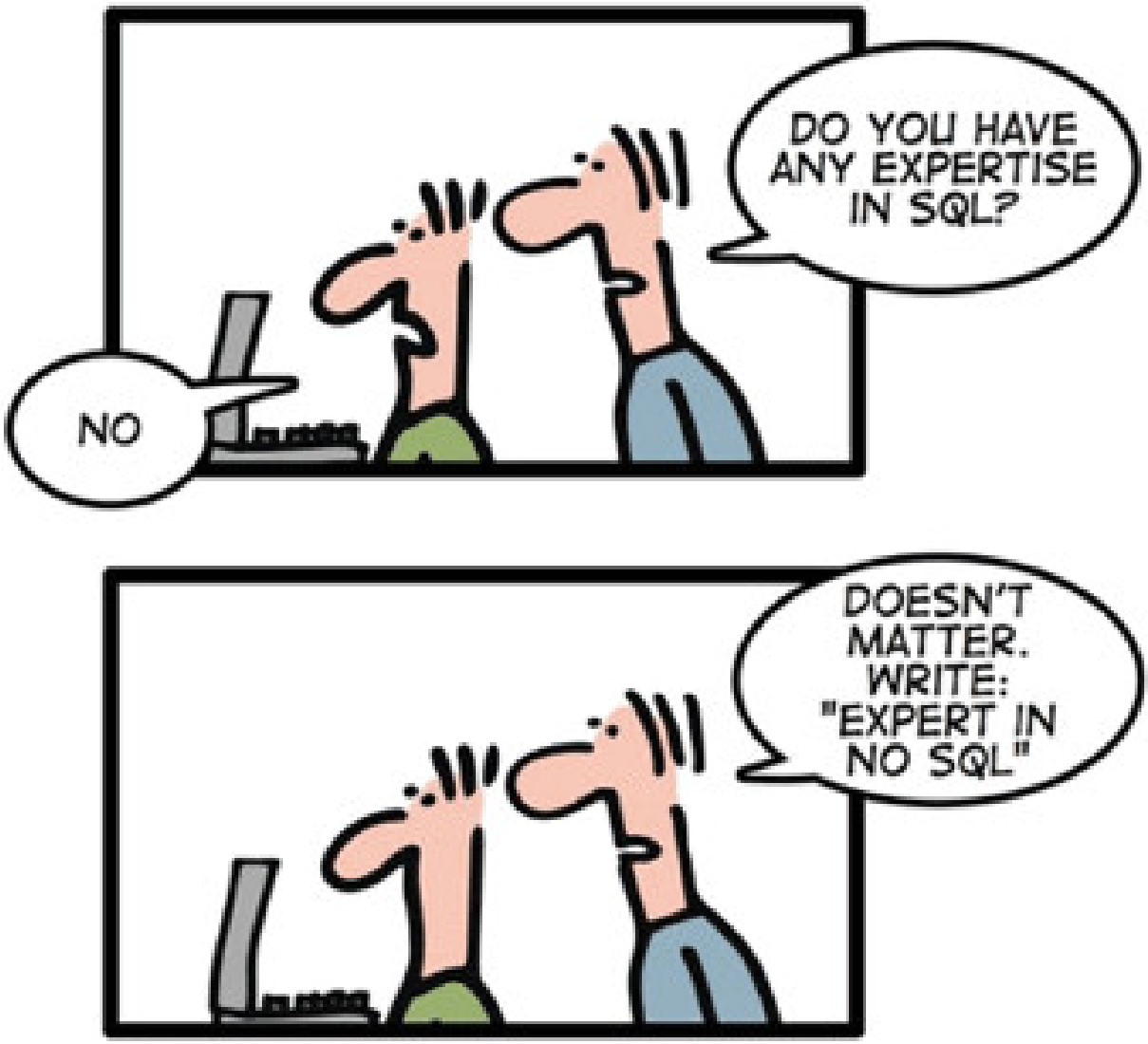
Clusters

- New opportunities:
  - Data Volume
  - Commodity Hardware
- Relational databases were not designed to run on clusters
  - They do not perform well on clusters
- Increasing mismatch
  - Google Bigtable
  - Amazon DynamoDB

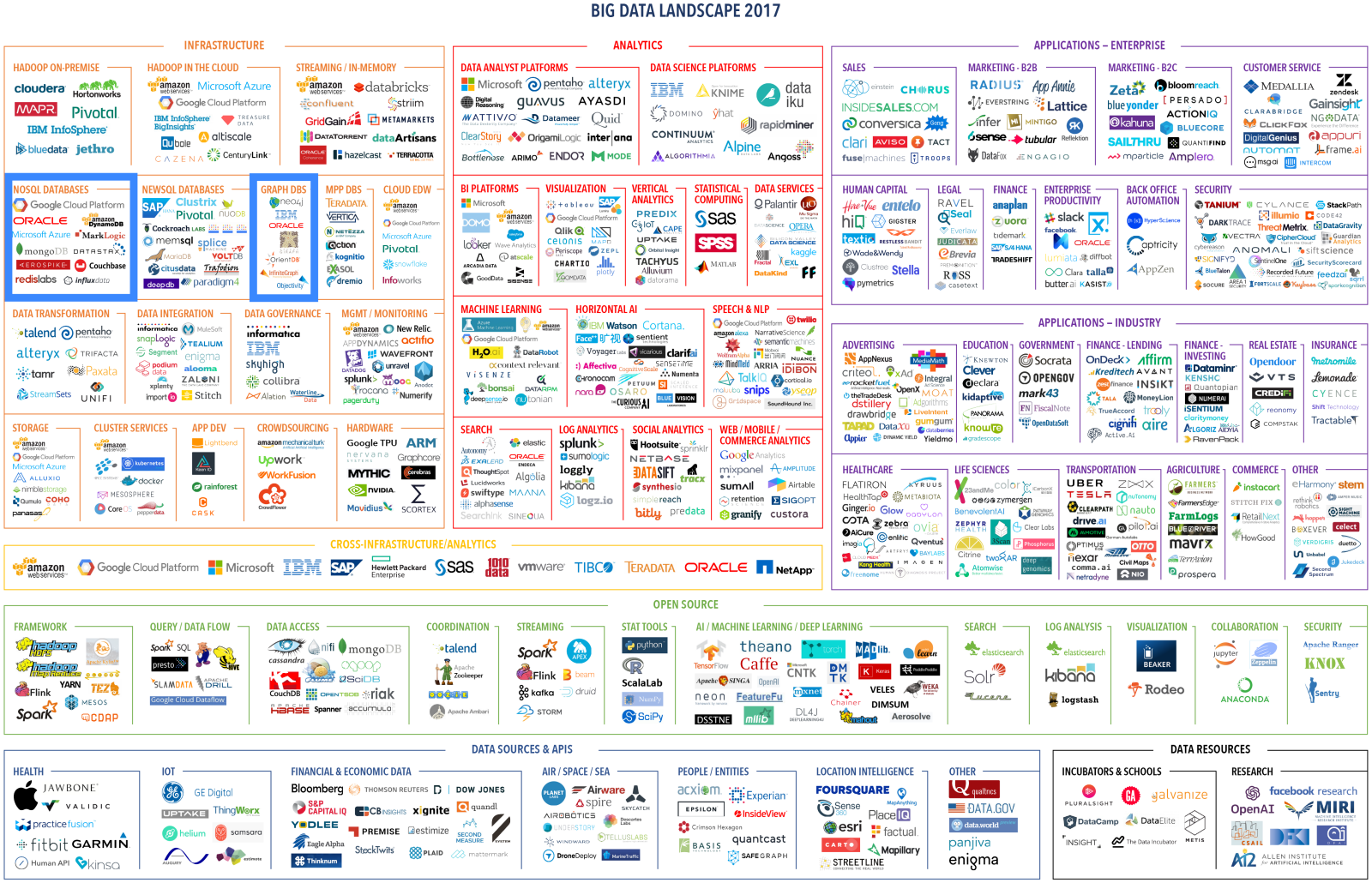
NoSQL

- Term appeared in the late 90s
  - Open source relational database [Strozzi NoSQL]
  - Tables as AScII files, without SQL
- Current interpretation:
  - 11 June, 2009: San Francisco meet-up
  - Hashtag chosen NoSQL
  - Main Features:
    - Not using the Relational Model and SQL
    - Open source projects (mostly)
    - Running on clusters
    - Schema-less

NoSQL


DO YOU HAVE ANY EXPERTISE IN SQL?
NO
DOESN'T MATTER. WRITE: "EXPERT IN NO SQL"

NoSQL


BIG DATA LANDSCAPE 2017
INFRASTRUCTURE
HADOOP ON-PREMISE
HADOOP IN THE CLOUD
STREAMING / IN-MEMORY
NOSQL DATABASES
NEWSQL DATABASES
GRAPH DBS
MPP DBS
CLOUD EDW
DATA TRANSFORMATION
DATA INTEGRATION
DATA GOVERNANCE
MGMT / MONITORING
STORAGE
CLUSTER SERVICES
APP DEV
CROWDSOURCING
HARDWARE
ANALYTICS
DATA ANALYST PLATFORMS
DATA SCIENCE PLATFORMS
BI PLATFORMS
VISUALIZATION
VERTICAL ANALYTICS
STATISTICAL COMPUTING
DATA SERVICES
MACHINE LEARNING
HORIZONTAL AI
SPEECH & NLP
SEARCH
LOG ANALYTICS
SOCIAL ANALYTICS
WEB / MOBILE / COMMERCE ANALYTICS
APPLICATIONS – ENTERPRISE
SALES
MARKETING - B2B
MARKETING - B2C
CUSTOMER SERVICE
HUMAN CAPITAL
LEGAL
FINANCE
ENTERPRISE PRODUCTIVITY
BACK OFFICE AUTOMATION
SECURITY
APPLICATIONS – INDUSTRY
ADVERTISING
EDUCATION
GOVERNMENT
FINANCE - LENDING
FINANCE - INVESTING
REAL ESTATE
INSURANCE
HEALTHCARE
LIFE SCIENCES
TRANSPORTATION
AGRICULTURE
COMMERCE
OTHER
CROSS-INFRASTRUCTURE/ANALYTICS
OPEN SOURCE
FRAMEWORK
QUERY / DATA FLOW
DATA ACCESS
COORDINATION
STREAMING
STAT TOOLS
AI / MACHINE LEARNING / DEEP LEARNING
SEARCH
LOG ANALYSIS
VISUALIZATION
COLLABORATION
SECURITY
DATA SOURCES & APIS
HEALTH
IOT
FINANCIAL & ECONOMIC DATA
AIR / SPACE / SEA
PEOPLE / ENTITIES
LOCATION INTELLIGENCE
OTHER
DATA RESOURCES
INCUBATORS & SCHOOLS
RESEARCH
V2 – Last updated 5/3/2017
© Matt Turck (@mattturck), Jim Hao (@jimrhao), & FirstMark (@firstmarkcap)
mattturck.com/bigdata2017
FIRSTMARK
EARLY STAGE VENTURE CAPITAL

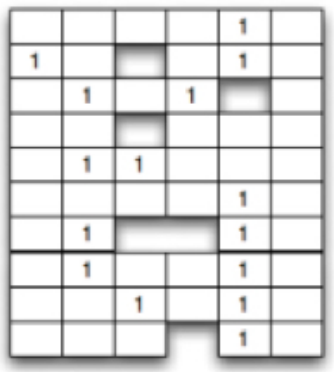
NoSQL


NoSQL Databases
amazon DynamoDB
Google Cloud Platform
ORACLE
Microsoft Azure
MarkLogic
mongoDB
DATASTAX
AEROSPIKE
Couchbase
SequoiaDB
redislabs
influxdata

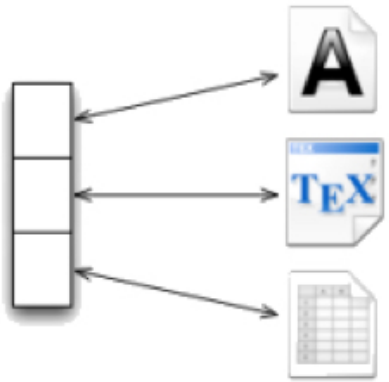
NoSQL

Key-Value

Graph



Column-Family



Document

Data Models For NoSQL Systems

- Document
  - MongoDB
- Key-value
  - Redis
- Column-family
  - Cassandra
- Graph
  - Neo4j

Pay Attention

Many possible data representations are possible

There is no theory that can state which data representation is the best of all for a given data store

Pay Attention

collection Player

id	*document*
8*mary	{ _id:"mary", username:"mary", firstName:"Mary", games[1]: { game:"Game:2345", opponent:"Player:rick" }, games[2]: { game:"Game:7425", opponent:"Player:ann" } }

collection Player

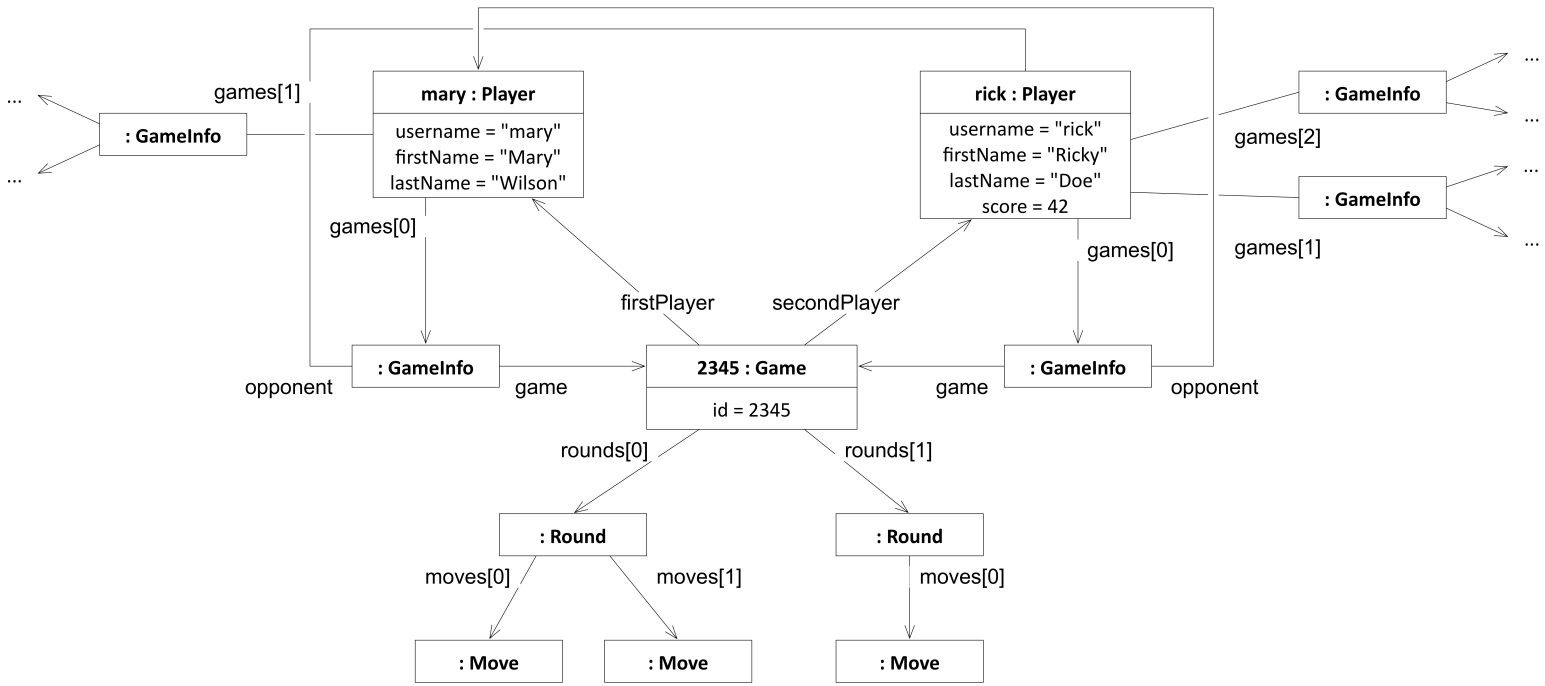
id	*document*
7*mary	{ _id:"mary", username:"mary", firstName:"Mary", games: [{ game:"Game:2345", opponent:"Player:rick"}, { game:"Game:7425", opponent:"Player:ann"}] }

Figure: Two possible data representations in MongoDB

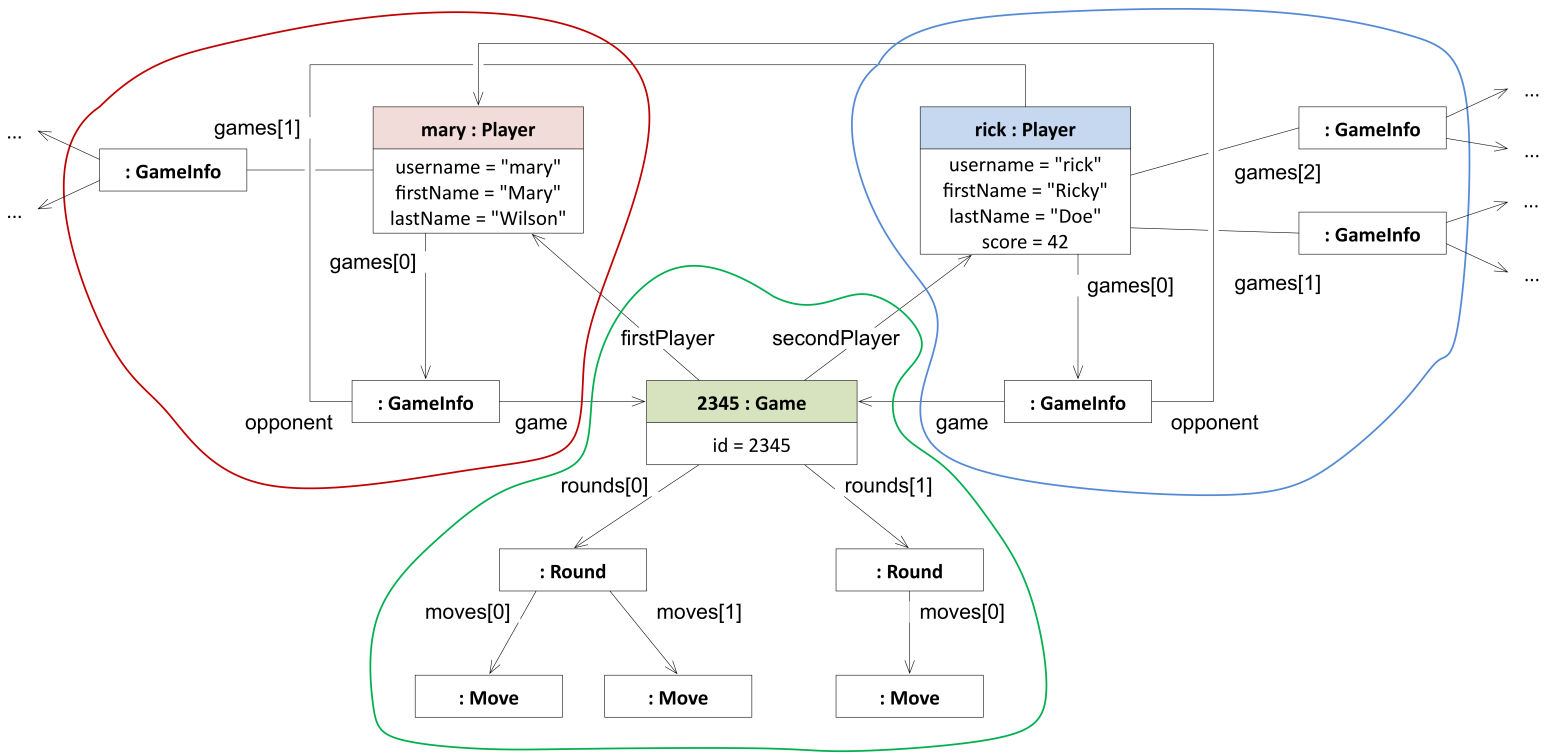
Aggregates

- Data as units that have a complex structure
  - More structure than just a set of tuples
    - A complex record with fields inside
    - A record with a composite access structure
- Aggregates in Domain Driven Design [4]:
  - A collection of related objects that we treat as a unit
  - A unit for data manipulation and management of consistency
- Advantages of aggregates:
  - easier for programmer to work with
  - easier for database systems to handle operations on a cluster

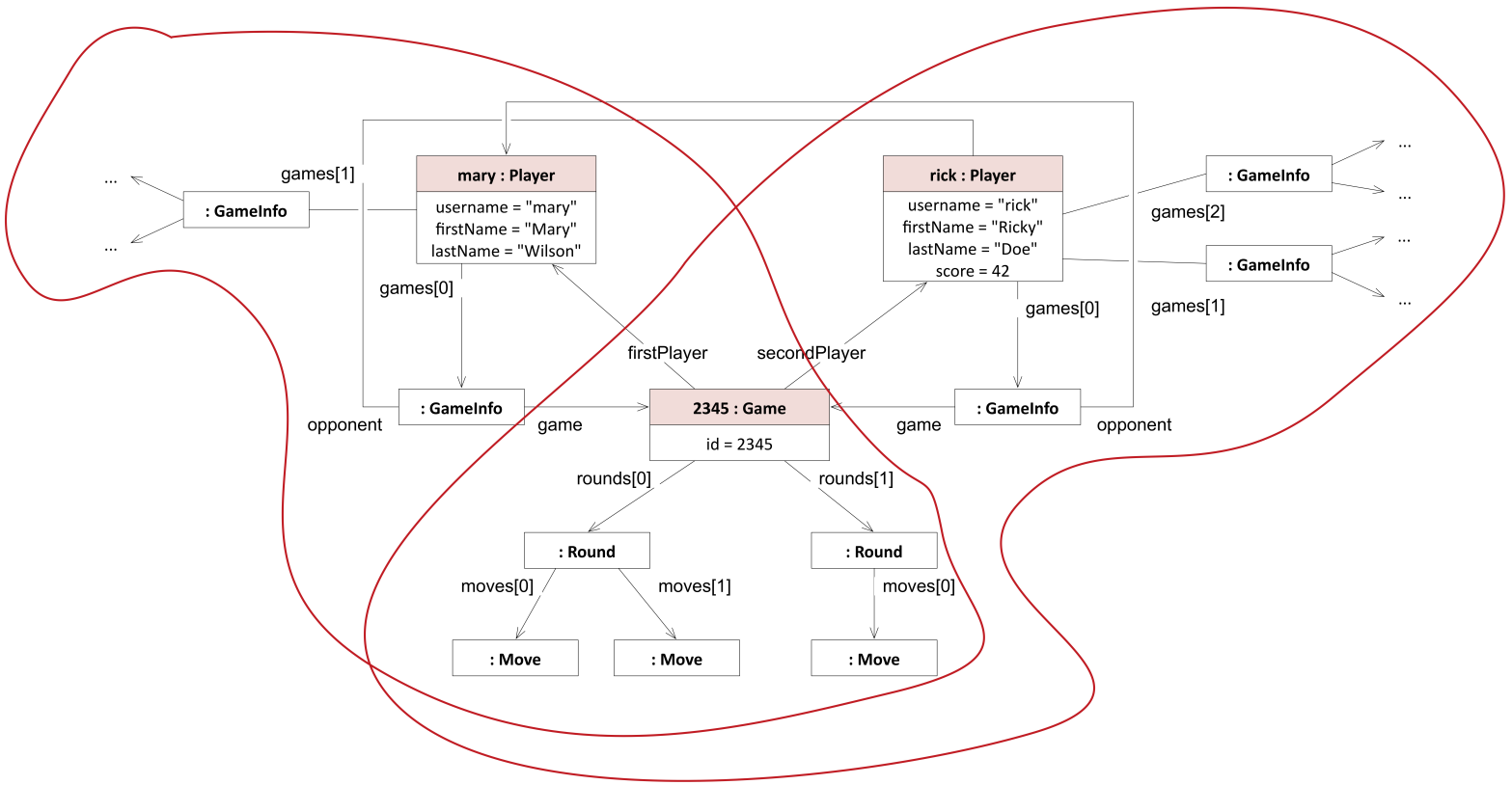
Aggregates [1]



Aggregates


...
games[1]
: GameInfo
mary : Player
username = "mary"
firstName = "Mary"
lastName = "Wilson"
games[0]
: GameInfo
opponent
game
2345 : Game
id = 2345
firstPlayer
secondPlayer
rounds[0]
rounds[1]
: Round
: Round
moves[0]
moves[1]
moves[0]
: Move
: Move
: Move
rick : Player
username = "rick"
firstName = "Ricky"
lastName = "Doe"
score = 42
games[0]
: GameInfo
game
opponent
games[2]
: GameInfo
games[1]
: GameInfo
...

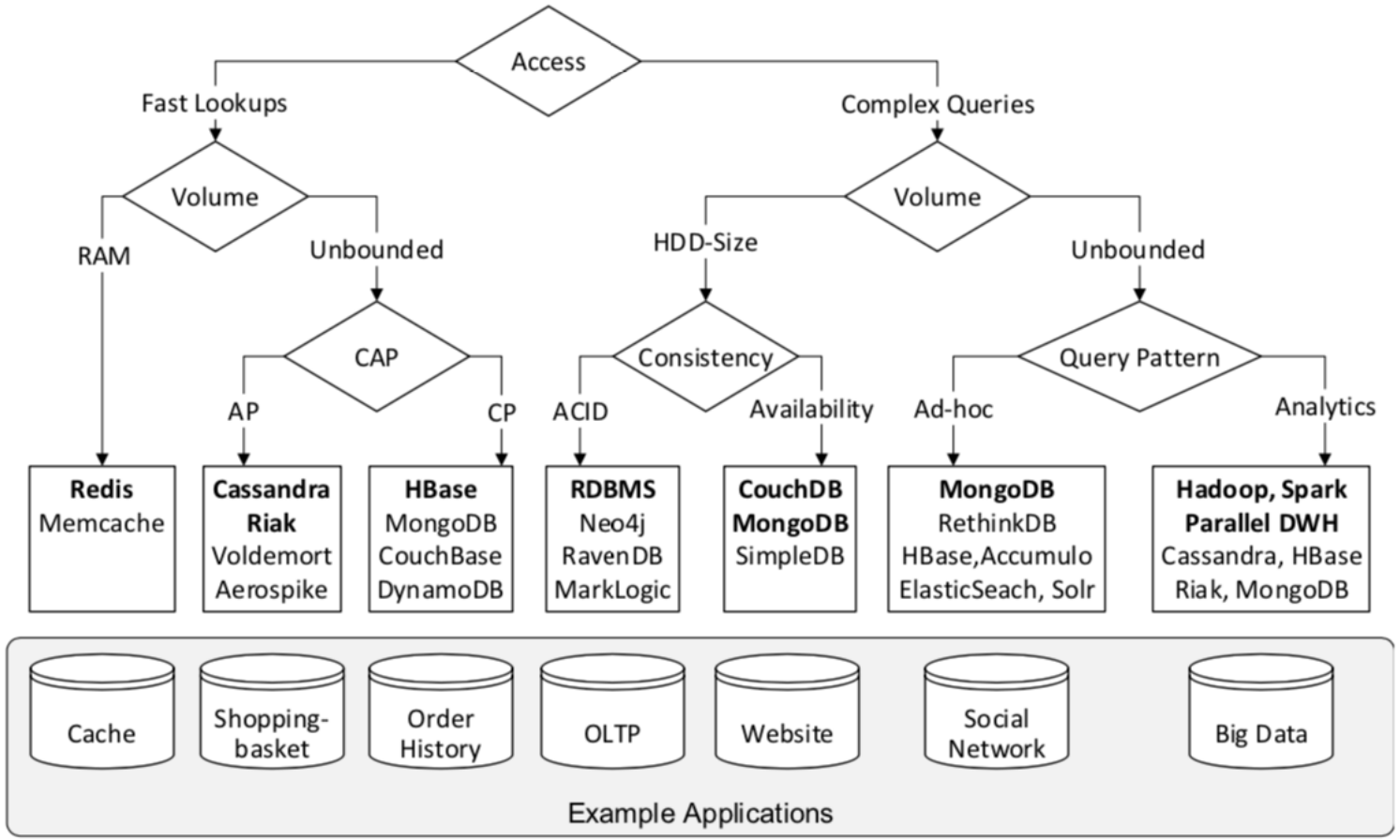
Aggregates


mary : Player
username = "mary"
firstName = "Mary"
lastName = "Wilson"
rick : Player
username = "rick"
firstName = "Ricky"
lastName = "Doe"
score = 42
2345 : Game
id = 2345
: GameInfo
: Round
: Move
games[0]
games[1]
games[2]
firstPlayer
secondPlayer
opponent
game
rounds[0]
rounds[1]
moves[0]
moves[1]

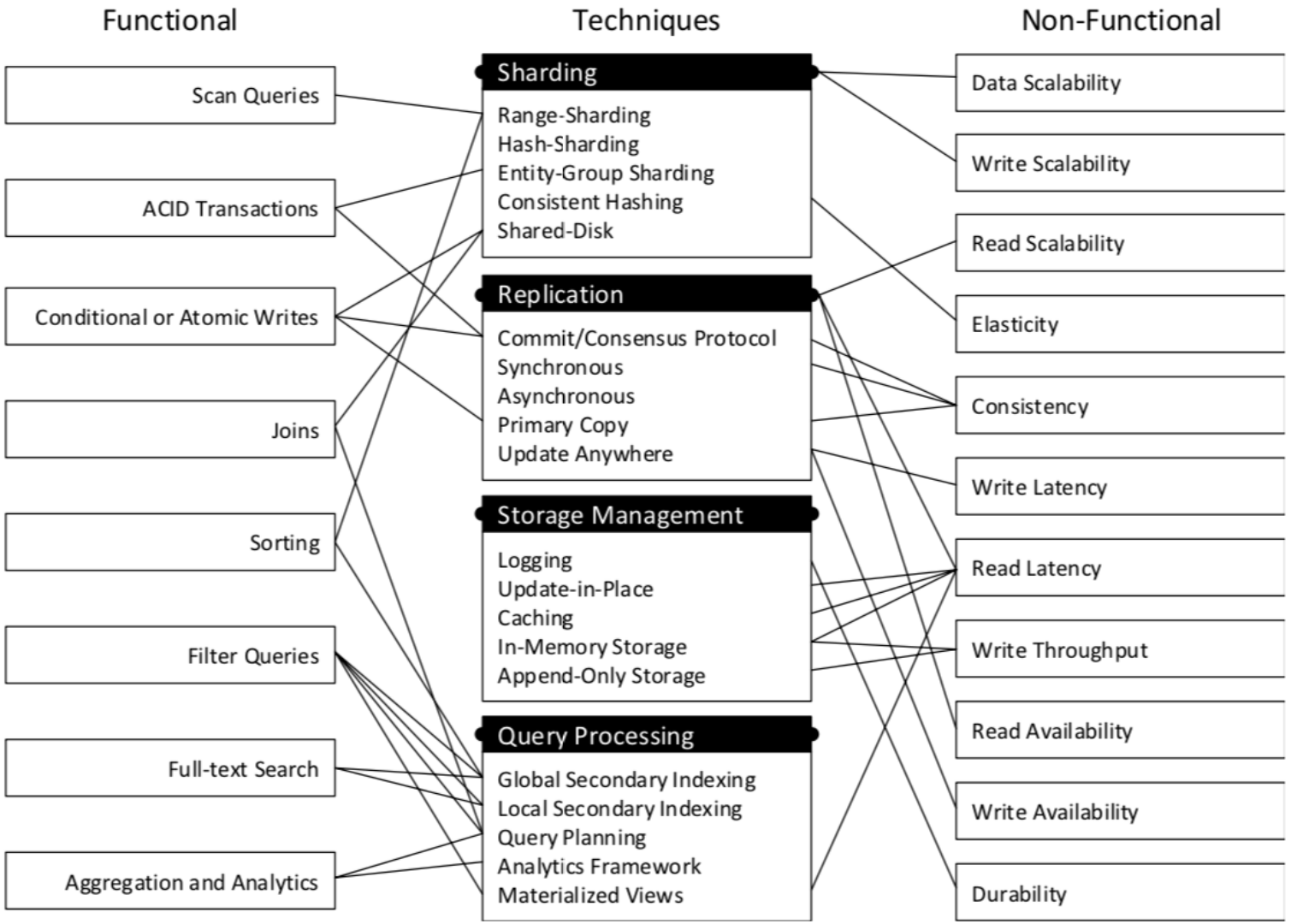
Design Strategy

- No universal strategy for designing aggregates boundaries
- The design depends on data manipulation
  - Access on a single round
  - Access on a player with all his games and rounds
- Context specific:
  - Some applications will run on a data representation
  - Some others on another data representation
- Focus on the Data Access Unit
  - very good for clusters
  - not easy to change the data interaction strategy

Not easy to classify and multiple options possible[6]



Not easy to classify and multiple options possible[6]



Not easy to classify and multiple options possible[6]

Query Capabilities

Not easy to classify and multiple options possible[6]

Table 1 Querying capabilities

NoSQL Data Stores		Querying					License
		Map Reduce	REST	Query	Other API	Other features	
Key-value stores	**Redis** http://redis.io	No	Third-party APIs	Does not provide SQL-like querying	CLI and API in several languages	Server-side scripting support using Lua	Open source: BSD (Berkeley Software Distribution)
	Memcached http://memcached.org	No	Third-party APIs	Does not provide SQL-like querying	CLI and API in several languages. Binary and ASCII protocols for custom client development	No server-side scripting support	Open source: BSD 3-clause license
	BerkeleyDB http://www.oracle.com/us/products/database/berkeley-db/overview/index.html	No	Yes	SQLite	CLI and API in several languages	No secondary indices, no server-side-scripting support	Closed source: Oracle Sleepycat license
	Voldemort http://www.project-voldemort.com/voldemort	Yes	Under development	No	Clients for several languages		Open source: Apache 2.0 license
	Riak http://basho.com/riak	Yes	Yes	Riak search, secondary indices	CLI and API in several languages	Provides filtering through key filters. Configurable secondary indexing. Provides Solr search capabilities. Provides server-side scripting	Open source: Apache 2.0 license
Column family stores	**Cassandra** http://cassandra.apache.org	Yes	Third party APIs	Cassandra query language	CLI and API in several languages. Supports Thrift interface	Secondary indexing mechanisms include column families, super-columns, collections	Open source: Apache 2.0 license
	HBasehttp://hbase.apache.org	Yes	Yes	No, could be used with Hive	Java/Any Writer	Server-side scripting support. Several secondary indexing mechanisms	Open source: Apache 2.0 license
	DynamoDB (Amazon service) http://aws.amazon.com/dynamodb	Amazon Elastic MapReduce	Yes	Proprietary	API in several languages	Provides secondary indexing based on attributes other than primary keys	Closed source: Pricing as pay-per-use basis
	Amazon SimpleDB (Amazon service) http://aws.amazon.com/simpledb	No	Yes	Amazon proprietary	Amazon proprietary API	Automatic indexing for all columns	Closed source: Pricing as pay-per-use basis
Document stores	**MongoDB** http://www.mongodb.org	Yes	Yes	Proprietary	CLI and API in several languages	Server-side scripting and secondary indexing support. A powerful aggregation framework	Open source: Free GNU AGPL v3.0 license
	CouchDB http://couchdb.apache.org	Yes	Yes	SQL like UnQL, under development	API in several languages	Server-side scripting and secondary indexing support	Open source: Apache 2.0 license
	Couchbase Server http://www.couchbase.com	Yes	Yes	No	Memcached API + protocol (binary and ASCII) in several languages	Server-side scripting and secondary indexing support	Open source: Free Community Edition. Paid Enterprise Edition
Graph databases	**Neo4J** http://www.neo4j.org	No	Yes	Cypher, Gremlin and SparQL	CLI and API in several languages	Server-side scripting and secondary indexing support	Open source license: NTCL + (A)GPLv3
	HyperGraphDB www.hypergraphdb.org/	No	Yes	SQL like querying	Currently has Java API. Could be used with Scala	Provides a search engine and Seco scripting IDE	Open source license: GNU LGPLv3
	Allegro Graph http://www.franz.com/agraph/allegrograph	No	Yes	SparQL and Prolog	API in several languages	Support for Solr indexing and search	Closed source: free, developer and enterprise versions
NewSQL	**VoltDB** http://voltdb.com/	No	Yes	SQL	CLI and API in several languages. JDBC support	Stored procedures are written in Java. Tables cannot join with themselves, and all joined tables must be partitioned over the same value	Open source AGPL v3.0 license. Commercial enterprise edition
	Spanner	Yes	NA	SQL like language	NA	Tables are partitioned into hierarchies, which describe locality relationship between tables	Google internal use only
	Clustrix http://www.clustrix.com/	No	No	SQL	Wire protocol compatible with MySQL		Closed source. Available as a service in the AWS marketplace, as an appliance, and as standalone software
	NuoDB http://www.nuodb.com/	No	No	SQL	CLI and drivers for most common data access APIs (JDBC, ODBC, ADO.NET). Also provides a C++ API	No support for stored procedures	Closed source. Pro and Developers editions. Available as a service in the AWS marketplace

Not easy to classify and multiple options possible[6]

Partition Replication and Consistency

Not easy to classify and multiple options possible[6]

Table 2 Partitioning, replication, consistency, and concurrency control capabilities

NoSQL Data Stores		Partitioning	Replication	Consistency	Concurrency control
Key-value stores	**Redis**	Not available (planned for Redis Cluster release). It can be implemented by a client or a proxy.	Master–slave, asynchronous replication.	Eventual consistency. Strong consistency if slave replicas are solely for failover.	Application can implement optimistic (using the WATCH command) or pessimistic concurrency control.
	Memcached	Clients' responsibility. Most clients support consistent hashing.	No replication. Repcached can be added to memcached for replication.	Strong consistency (single instance).	Application can implement optimistic (using CAS with version stamps) or pessimistic concurrency control.
	BerkeleyDB	Key-range partitioning and custom partitioning functions. Not supported by the C# and Java APIs at this time.	Master–slave.	Configurable.	Readers-writer locks.
	Voldemort	Consistent hashing.	Masterless, asynchronous replication. Replicas are located on the first R nodes moving over the partitioning ring in a clockwise direction.	Configurable, based on quorum read and write requests.	MVCC with vector clock.
	Riak	Consistent hashing.	Masterless, asynchronous replication. The built-in functions determine how replicas distribute the data evenly.	Configurable, based on quorum read and write requests.	MVCC with vector clock.
Column family stores	**Cassandra**	Consistent hashing and range partitioning (known as order preserving partitioning in Cassandra terminology) is not recommended due to the possibility of hot spots and load balancing issues.	Masterless, asynchronous replication. Two strategies for placing replicas: replicas are placed on the next R nodes along the ring; or, replica 2 is placed on the first node along the ring that belongs to another data centre, with the remaining replicas on the nodes along the ring in the same rack as the first.	Configurable, based on quorum read and write requests.	Client-provided timestamps are used to determine the most recent update to a column. The latest timestamp always wins and eventually persists.
	HBase	Range partitioning.	Master–slave or multi-master, asynchronous replication. Does not support read load balancing (a row is served by exactly one server). Replicas are used only for failover.	Strong consistency.	MVCC.
	DynamoDB	Consistent hashing.	Three-way replication across multiple zones in a region. Synchronous replication.	Configurable.	Application can implement optimistic (using incrementing version numbers) or pessimistic concurrency control.
	Amazon SimpleDB	Partitioning is achieved in the DB design stage by manually adding additional domains (tables). Cannot query across domains.	Replicas within a chosen region.	Configurable.	Application can implement optimistic concurrency control by maintaining a version number (or a timestamp) attribute and by performing a conditional put/delete based on the attribute value.
Document Stores	**MongoDB**	Range partitioning based on a shard key (one or more fields that exist in every document in the collection). In addition, hashed shard keys can be used to partition data.	Master–slave, asynchronous replication.	Configurable. Two methods to achieve strong consistency: set connection to read only from primary; or, set *write concern* parameter to "Replica Acknowledged".	Readers–writer locks.
	CouchDB	Consistent hashing.	Multi-master, asynchronous replication. Designed for off-line operation. Multiple replicas can maintain their own copies of the same data and synchronize them at a later time.	Eventual consistency.	MVCC. In case of conflicts, the winning revision is chosen, but the losing revision is saved as a previous version.
	Couchbase Server	A hashing function determines to which bucket a document belongs. Next, a table is consulted to look up the server that hosts that bucket.	Multi-master.	Within a cluster: strong consistency. Across clusters: eventual consistency.	Application can implement optimistic (using CAS) or pessimistic concurrency control.
Graph databases	**Neo4J**	No partitioning (cache sharding only).	Master–slave, but can handle write requests on all server nodes. Write requests to slaves must synchronously propagate to master.	Eventual consistency.	Write locks are acquired on nodes and relationships until committed.

Not easy to classify and multiple options possible[6]

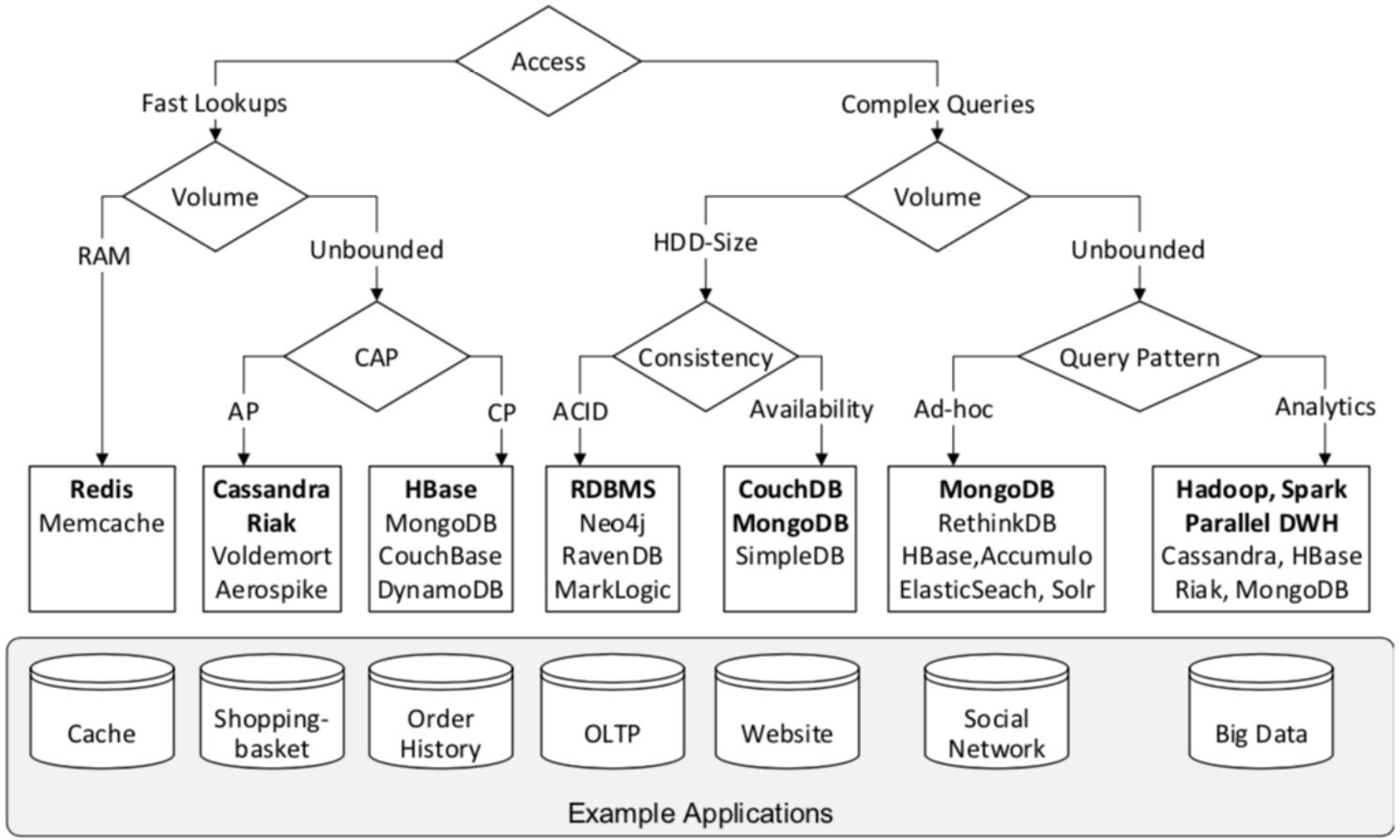
Security

Not easy to classify and multiple options possible[6]

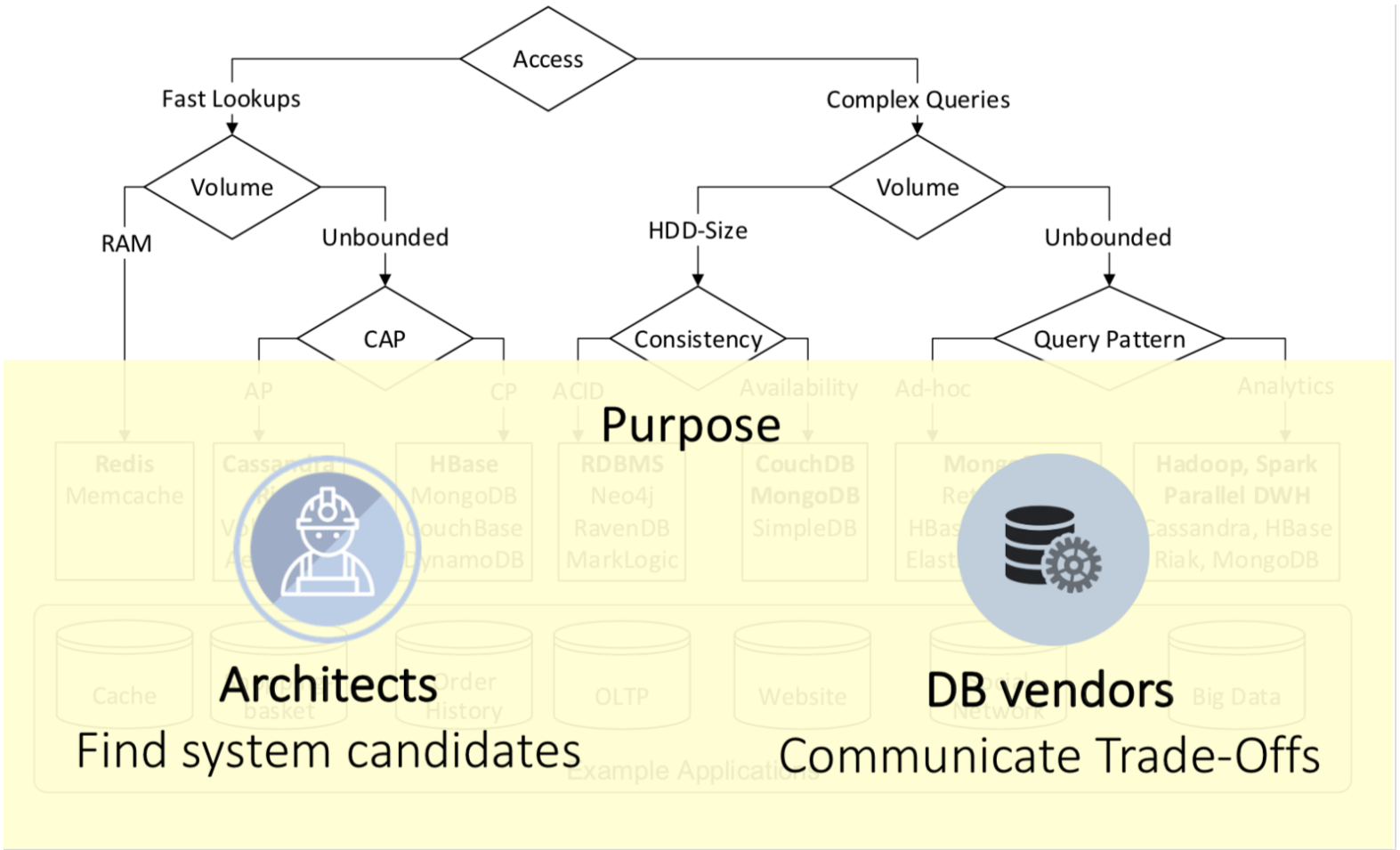
Table 3 Security features

NoSQL Data Stores		Encryption			Authentication	Authorization	Auditing
		Data at Rest	Client/Server	Server/Server			
Key-Value stores	**Redis**	No	No	No	Admin password sent in clear text for admin functions. Data access does not support authentication	No	No
	Memcached	NA, Memcache does store data on disk	No	No	Binary protocol supports Simple Authentication and Security Layer (SASL) authentication	No	No
	BerkeleyDB	Yes, the database needs to be created using encryption	NA, embedded data store	No	No	No	No
	Voldemort	Possibly if BerkeleyDB is used as the storage engine	No	No	No	No	No
	Riak	No	REST interface supports HTTPS. Binary protocol is not encrypted	Multiple data-centre replication can be done over HTTPS	No	No	No
Column Family Stores	**Cassandra**	Enterprise Edition only. Commit log is not encrypted	Yes, SSL based	Yes, configurable: all server-to-server communication, only between datacentres or between servers in the same rack	Yes, store credentials in a system table. Possible to provide pluggable implementations	Yes, similar to the SQL GRANT/REVOKE approach.	Enterprise Edition only. Based on log4j framework.
						Possible to provide pluggable implementations	Logging categories include ADMIN, ALL, AUTH, DML, DDL, DCL, and QUERY. Possible to disable logging for specific keyspaces
	HBase	No, planned for future release	Yes	Communication of HBase nodes with the HDFS and Zookeeper clusters can be secured. Not clear whether the HBase nodes communicate via a secure channel	Yes, RPC API based on SASL, supporting Kerberos. REST API uses a HTTP gateway, which authenticates with the data store as one single user, and executes all operations on his/her behalf	Yes, permissions include read, write, create and admin. Granularity of table, column family, or column	No, planned for future release
	Amazon DynamoDB	No	Yes, HTTPS	NA	Integration with Identity and Access Management (IAM) services. The requests need to be signed using HMAC-SHA256	Allow the creation of policies that associate users and operations on domains. Possible to define policies for temporary access	Integrates with Amazon Cloud Watch service. Access information about latencies for operations, amount of data stored, and requests throughput
	Amazon SimpleDB	See DynamoDB					No
Document Stores	**MongoDB**	No, a third-party partner (Gazzang) provides an encryption plug-in	Yes, SSL-based	Yes	Yes, store credentials in a system collection. REST interface does not support authentication. Enterprise Edition supports Kerberos	Yes, permissions include read, read/write, dbAdmin, and userAdmin. Granularity of collections	No
	CouchDB	NA	Yes, SSL-based	Possible using HTTPS connections	Yes, HTTP authentication using cookies or BASIC method. Oauth supported	Three levels of users: server admin, database admin, and database member. Complex authorization can be done in validation functions	No
	Couchbase Server	No	No	No, planned for future release	Yes, SASL authentication – each bucket is differentiated by its name and password. REST API for administrative function uses HTTP BASIC authentication	No	No
Graph databases	**Neo4J**	No	Yes, SSL-based	No	No, developers can create a SecurityRule and register with the server	No	No

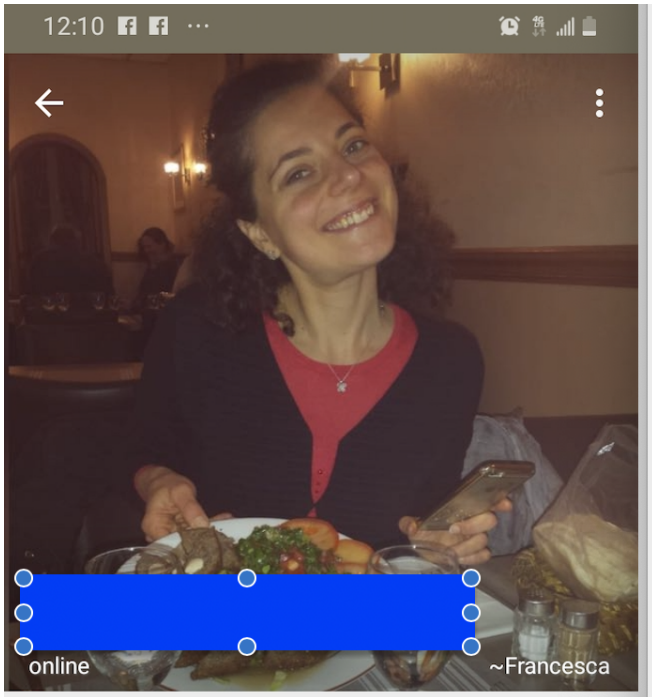
Not easy to classify and multiple options possible[6]

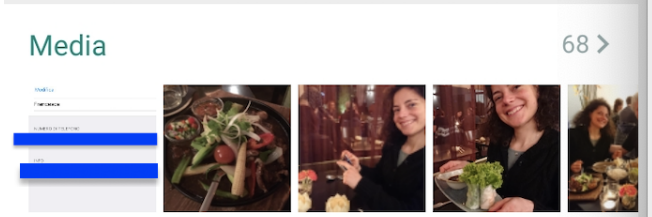


Not easy to classify and multiple options possible[6]



In practice? How do they work?

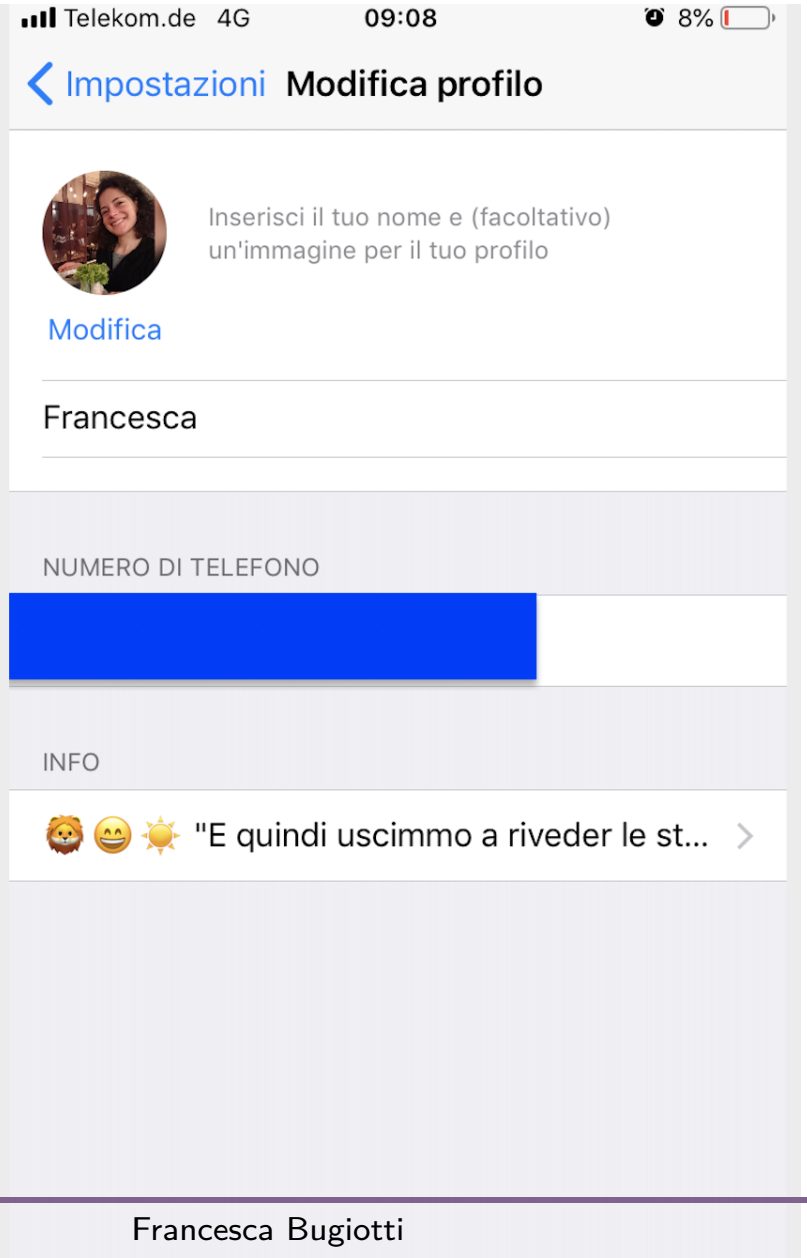




Mute notifications

Custom notifications

Media visibility



Plan

Document Database Word

Document Database

The database is a collection of documents

- The database stores and retrieves documents
- A key-value data-store where the value is an examinable document
- Documents:
  - can be XML, JSON, BSON, and so on
  - are self-describing, hierarchical tree data structures
  - can consist of scalar values, collections, and maps
  - are structurally similar but not identical to each other

Operations:

- get, put, delete a document given its ID
- query the content of the document

Example

```
{
  "firstname": "Pramod",
  "gamesPlayed": [ "GameStar",
                   "GamePuzzle",
   "GameCities"],
  "addresses": [
    { "state": "France",
      "city": "Paris",
      "Department": "Essonne"
    },
    { "state": "France",
      "city": "Lille",
      "type": "Nord" }
  ],
}
```

Document Databases

- Strongly aggregate oriented
  - A lot of aggregates
  - Each aggregate has a key
- Data Model:
  - A set of (key,document) pairs
  - Document: an aggregate instance
- It is possible to query/navigate the aggregate
- Access to an aggregate:
  - queries based on the fields of the aggregate

Example

collection Player

id	*document*
8*mary	{ _id:"mary", username:"mary", firstName:"Mary", games[1]: { game:"Game:2345", opponent:"Player:rick" }, games[2]: { game:"Game:7425", opponent:"Player:ann" } }

collection Player

id	*document*
7*mary	{ _id:"mary", username:"mary", firstName:"Mary", games: [{ game:"Game:2345", opponent:"Player:rick"}, { game:"Game:7425", opponent:"Player:ann"}] }

Figure: Two possible data representations in MongoDB

MongoDB

Database structure

- Each MongoDB instance has multiple databases
  - Similar to a database schema in a RDBMS
- Each database can have multiple collections
  - Similar to a table in a RDBMS
- When we store a document, we have to choose which database and collection this document belongs to

```
db.collection.insert(document)
```

MongoDB

Document Structure

- The schema of the data in a collection can differ across documents
  - It is possible to say that a collection is a set of similar documents
- In documents no empty attributes
  - Empty values correspond to not-included attributes
- New attributes can be created without the need to define them or to change the existing documents
- It is possible to include special attributes that reference other documents

Example

```
{
  _ID: 2330
  "firstname": "Alice",
  "lastname": "Foo",
  "gamesPlayed": [ "GameStar",
                   "GamePuzzle",
   "GameCities"],
  "addresses": [
    { "state": "France",
      "city": "Paris",
      "Department": "Essonne"
    },
    { "state": "France",
      "city": "Lille",
      "type": "Nord" }
  ],
  "birthday":"25 August"
}
```

```
{
  _ID: 2332,
  "firstname": "Pramod",
  "gamesPlayed": [ "GameStar",
                   "GamePuzzle",
                   "GameCities"],
  "addresses": [
    { "state": "France",
      "city": "Paris",
      "Department": "Essonne"
    },
    { "state": "France",
      "city": "Lille",
      "type": "Nord" }
  ],
}
```

Example

```
{ _ID: 2330,
  "firstname": "Alice",
  "lastname": "Foo"
  "gamesPlayed": [ "GameStar",
                   "GamePuzzle",
                   "GameCities"],
  "games": [ 4657,
             2345,
             56984],
  "birthday":"25 August"
}
```

```
{ _ID: 2345,
  "gameName": "GameStar",
  "rounds": [ 4564, 45696],
  "globalScore": 45565},
  "opponent": 445,
  "globalTime": "55769"
}
```

Query features

In General

You can query the data inside the document without having to retrieve the whole document by its key and then introspect the document

Different document DBMS provide different query features

MongoDB query language

Expressed via JSON with simple constructs

find() operation

```
db.collection.find( <query filter>, <projection>
).<cursor modifier>
```

```
/* in games
{
   ''gameId'':''99'',
   ''playerId'':''1234fb'',
   ''gameDate'':''2014-04-25'',
   ''rounds'':[
      { ''round'': {''id'':27, ''score'':''45''}, ''time'': 32.45 }
      { ''round'': {''id'':55, ''score'':''75''}, ''time'': 41.33 }
   ],
} */

db.games.find()
```

Matches every document in the collection games

MongoDB query language

Expressed via JSON with simple constructs

`find()` operation

```
db.collection.find( <query filter>, <projection>
).<cursor modifier>
```

```
/* in games
{
   ''gameId'':''99'',
   ''playerId'':''1234fb'',
   ''gameDate'':''2014-04-25'',
   ''rounds'':[
      { ''round'': {''id'':27, ''score'':''45''}, ''time'': 32.45 }
      { ''round'': {''id'':55, ''score'':''75''}, ''time'': 41.33 }
   ],
} */

db.games.find({''playerId'':''1234fb''})
```

Finds all the documents where the value for `playerId` is `234fb`

MongoDB query language

Expressed via JSON with simple constructs

`find()` operation

```
db.collection.find( <query filter>, <projection>
).<cursor modifier>
```

```
/* in games
{
   ''gameId'':''99'',
   ''playerId'':''1234fb'',
   ''gameDate'':''2014-04-25'',
   ''rounds'':[
      { ''round'': {''id'':27, ''score'':''45''}, ''time'': 32.45 }
      { ''round'': {''id'':55, ''score'':''75''}, ''time'': 41.33 }
   ],
} */

db.games.find({''playerId'':''1234fb''}, {''gameId'':1,''gameDate'':1})
```

The query returns only the attributes `_id` (returned by default), `gameID`, and `gameDate`

MongoDB query language

Expressed via JSON with simple constructs

find() operation

```
db.collection.find( <query filter>, <projection>
).<cursor modifier>
```

```
/* in games
{
   ''gameId'':''99'',
   ''playerId'':''1234fb'',
   ''gameDate'':''2014-04-25'',
   ''rounds'':[
      { ''round'': {''id'':27, ''score'':''45''}, ''time'': 32.45 }
      { ''round'': {''id'':55, ''score'':''75''}, ''time'': 41.33 }
   ],
} */

db.games.find({''rounds.time'':32.45},{playerID:1, rounds.$ : 1 })
```

We can query also according the value of some attribute contained in an array.

MongoDB query language

Expressed via JSON with simple constructs

`find()` operation

```
db.collection.find( <query filter>, <projection>
).<cursor modifier>
```

```
/* in games
{
   ''gameId'':''99'',
   ''playerId'':''1234fb'',
   ''gameDate'':''2014-04-25'',
   ''rounds'':[
      { ''round'': {''id'':27, ''score'':''45''}, ''time'': 32.45 }
      { ''round'': {''id'':55, ''score'':''75''}, ''time'': 41.33 }
   ],
} */

db.games.find({''rounds'' : { ''$elemMatch'' : {''round.score'' : {''$gte'' : 45}}}})
```

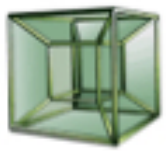
We can query also embedded documents:

- we can impose a condition on the `score` value

MongoDB[5]


Sharding
Range-Sharding
Hash-Sharding
Entity-Group Sharding
Consistent Hashing
Shared Disk
Replication
Trans-action Protocol
Sync. Replica-tion
Async. Replica-tion
Primary Copy
Update Anywhere
Storage Management
Logging
Update-in-Place
Caching
In-Memory
Append-Only Storage
Query Processing
Global Index
Local Index
Query Planning
Analytics
Materialized Views

Column-family databases



Column-family databases I

Also called Extensible Record Stores, they come after the Google's success with BigTable [2]

Their basic data model consists of rows and columns

- Rows are analogues to documents
  - Variable number of columns (called also attributes, fields, qualifiers, etc.) having a name
  - Any type for the columns
  - Rows are grouped into collections (called also tables)
  - Values associated with timestamps
- Columns are grouped together and form column groups (super columns, column families, etc.)
  - a column group is often a namespace for a column
    - columns with the same name in different column groups

Column-Family Databases

- Strongly aggregate oriented
  - A lot of aggregates
  - Each aggregate has a key
- Data Model:
  - A two level map structure
    - A set of (row-key, aggregate) pairs
    - Each aggregate is a group of pairs (column-key, value)
- Structure of the aggregate is visible
- Columns can be organized in families
  - Data usually accessed together
- Access to an aggregate:
  - a row as a whole
  - part of a column

Example

[Item per object in DynamoDB]

table	*_id*	*username*	*firstName*	*lastName*	*games*
`Player`	`mary1994`	`mary1994`	`Mary`	`Wilson`	`{ ... }`

[Cell per object in Cassandra]

table	*id*	*value*
`Player`	`mary1994`	"username":"mary1994", "firstName":"Mary", ...

Column-family databases I

Operations:

- put, get, scan, delete of a row
- put, get, delete of fields of a row
- access to groups of fields of a row in a column group
- access to rows using a given timestamp

Cassandra columns

The basic unit of storage in Cassandra is the column

- A column is a name-value pair
  - The name behaves as the key
  - The value is a simple byte-array
  - A column is always stored with a timestamp
- The timestamp is used to expire data, resolve write conflicts, deal with stale data, and do other things

```
{
key:‘‘FirstName’’,
value:‘‘Alice’’,
timestamp: ‘‘13645585’’
}
```

Cassandra rows

- A row is a collection of columns attached or linked to a key
- A collection of similar rows makes a column family
- Column families are defined into keyspaces
- Keyspaces have to be created so that column families can be assigned to them:

```
create keyspace game
```

Column Families

```
//column family
{
//row
"myAlice" : {
  firstName: "Alice",
  lastName: "Foo",
  lastGame: "2014/1/12"
  } //row
  "bobGreen" : {
  firstName: "Bob",
  lastName: "Green",
  location: "Boston"
 }
}
```

Cassandra query features

Cassandra does not have a rich query interface, because of this it is necessary to carefully choose the columns and the column families

- Cassandra client that includes the get, set, and delete operations
- Indexing Features
- Cassandra Query Language (CQL)

Cassandra query features

- Cassandra client

```
use games;
CREATE COLUMN FAMILY Player
WITH comparator = UTF8Type
AND key_validation_class=UTF8Type
AND column_metadata = [
     {column_name: city, validation_class: UTF8Type}
 {column_name: name, validation_class: UTF8Type}];
   SET Customer['myAlice']['city']='Boston';
   SET Customer['myAlice']['name']='Alice Foo';
   GET Customer['myAlice'];
   GET Customer['myAlice']['name'];
   DEL Customer['myAlice']['city'];
   DEL Customer['myAlice'];
```

Cassandra Query Language

The Cassandra Query Language is a

- SQL-like query language

```
CREATE COLUMNFAMILY Player ( KEY varchar PRIMARY KEY,
name varchar,
city varchar);
INSERT INTO Player (KEY,name,city)
VALUES ('myAlice',
        'Alice Foo',
        'Boston');
SELECT * FROM Customer
SELECT name, city FROM Player
SELECT name, city FROM Player
 WHERE city='Boston'
```

Cassandra [5]


Sharding
Range-Sharding
Hash-Sharding
Entity-Group Sharding
Consistent Hashing
Shared Disk
Replication
Trans-action Protocol
Sync. Replica-tion
Async. Replica-tion
Primary Copy
Update Anywhere
Storage Management
Logging
Update-in-Place
Caching
In-Memory
Append-Only Storage
Query Processing
Global Index
Local Index
Query Planning
Analytics
Materialized Views

Key-Value Databases Word

Key-Value Databases

A simple hash table accessible only through its primary key

- Basically a table with two columns: the key and the value
- The value is a blob that the data store just stores: it can be text, JSON, XML, or anything else

Operations:

- get the value for the key
- put a value for a key
- delete a key from the database

Key-Value Databases

- Strongly aggregate oriented
  - A lot of aggregates
  - Each aggregate has a key
- Data Model:
  - A set of (key,value) pairs
  - Value: an aggregate instance
- It is not possible to query/navigate the aggregate
  - the aggregate is a blob of meaningless bits
- Access to an aggregate
  - lookup based on its key

Example

[A first data representation in Oracle NoSQL]

key	*value*
/Player/mary1994/-/username	mary1994
/Player/mary1994/-/firstName	Mary
/Player/mary1994/-/lastName	Wilson
/Player/mary1994/-/games	...

[A second data representation in Oracle NoSQL]

key	*value*
/Player/mary1994/-/username	mary1994
/Player/mary1994/-/firstName	Mary
/Player/mary1994/-/lastName	Wilson
/Player/mary1994/-/games/0/id	2345
/Player/mary1994/-/games/0/opponent	rick_the_good
/Player/mary1994/-/games/0/gameDetails	...
/Player/mary1994/-/games/1/id	7425
/Player/mary1994/-/games/1/opponent	ann x
/Player/mary1994/-/games/1/gameDetails	...

Example

[Key-value per object in Redis]

key	*value*
`Player:mary1994`	"username":"mary1994", "firstName":"Mary", ...

[Key-value per field in Redis]

key	*value*
`Player:mary1994/username`	`mary1994`
`Player:mary1994/firstName`	`Mary`
`Player:mary1994/lastName`	`Wilson`
`Player:mary1994/games`	`...`

[Key-hash per field in Redis]

key	*value*

`4*Player:mary1994`	`username:mary1994`
	`firstName:Mary`
	`lastName:Wilson`
	`games:...`

Query Features

General

- Only query by the key
- It is not possible to use some attribute of the value column
- If the key is not known:
  - Some systems allow the search inside the value
    - e.g., Riak Search
  - The key needs to be suitably chosen
    - e.g., session ID for storing session data

Example

[Key-value per object in Redis]

key	*value*
`Player:mary1994`	"username":"mary1994", "firstName":"Mary", ...

[Key-value per field in Redis]

key	*value*
`Player:mary1994/username`	`mary1994`
`Player:mary1994/firstName`	`Mary`
`Player:mary1994/lastName`	`Wilson`
`Player:mary1994/games`	`...`

[Key-hash per field in Redis]

key	*value*

`4*Player:mary1994`	`username:mary1994`
	`firstName:Mary`
	`lastName:Wilson`
	`games:...`

Query Features

General

- Only query by the key
- It is not possible to use some attribute of the value
- If the key is not known:
  - Some systems allow the search inside the value
    - e.g., Riak Search
  - The key needs to be suitably chosen
    - e.g., session ID for storing session data

Redis

Database Structure

Redis is an open source in-memory data structure store

It can be used as a:

- database
- cache
- message broker

Redis

Database Structure

Redis is an open source in-memory data structure store

It can be used as a:

-
-
- message broker

Redis

Databases Structure

- Stores key-value pairs

The value is not limited to a simple string:

- Binary-safe strings
- Lists
- Sets
- Sorted sets
- Hashes
- Bit arrays
- HyperLogLogs

Redis

Databases Structure

- Stores key-value pairs

The value is not limited to a simple string:

- Binary-safe strings

Redis

Databases Structure

- Stores key-value pairs

The value is not limited to a simple string:

- Binary-safe strings
- Lists
- Sets
- Sorted sets
- Hashes
- Bit arrays
- HyperLogLogs

Redis

Databases Structure

- Stores key-value pairs

The value is not limited to a simple string:

-
-
-
-
-
-
- HyperLogLogs

Redis

HyperLogLog

probabilistic data structure used to count unique values

Counting unique values with exact precision requires an amount of memory proportional to the number of unique values:

- trades memory consumption for precision

Query Features

Redis commands

provides basic store, fetch, delete operations

Redis APIs

There are modules/APIs that allow to integrate the connection to Redis into the code

Query Features

Redis commands

`redis-cli` is the Redis command line interface

- send commands to Redis
- read the replies sent by the server
- operations run directly from the terminal

Query Features

Simple operations

```
SET "my key" "my string to store"
GET "my key"
```

Key-Value Databases Word

```
funny-ecran:src francy$ redis-cli
127.0.0.1:6379> SET a b
OK
127.0.0.1:6379> get a
"b"
127.0.0.1:6379>
```

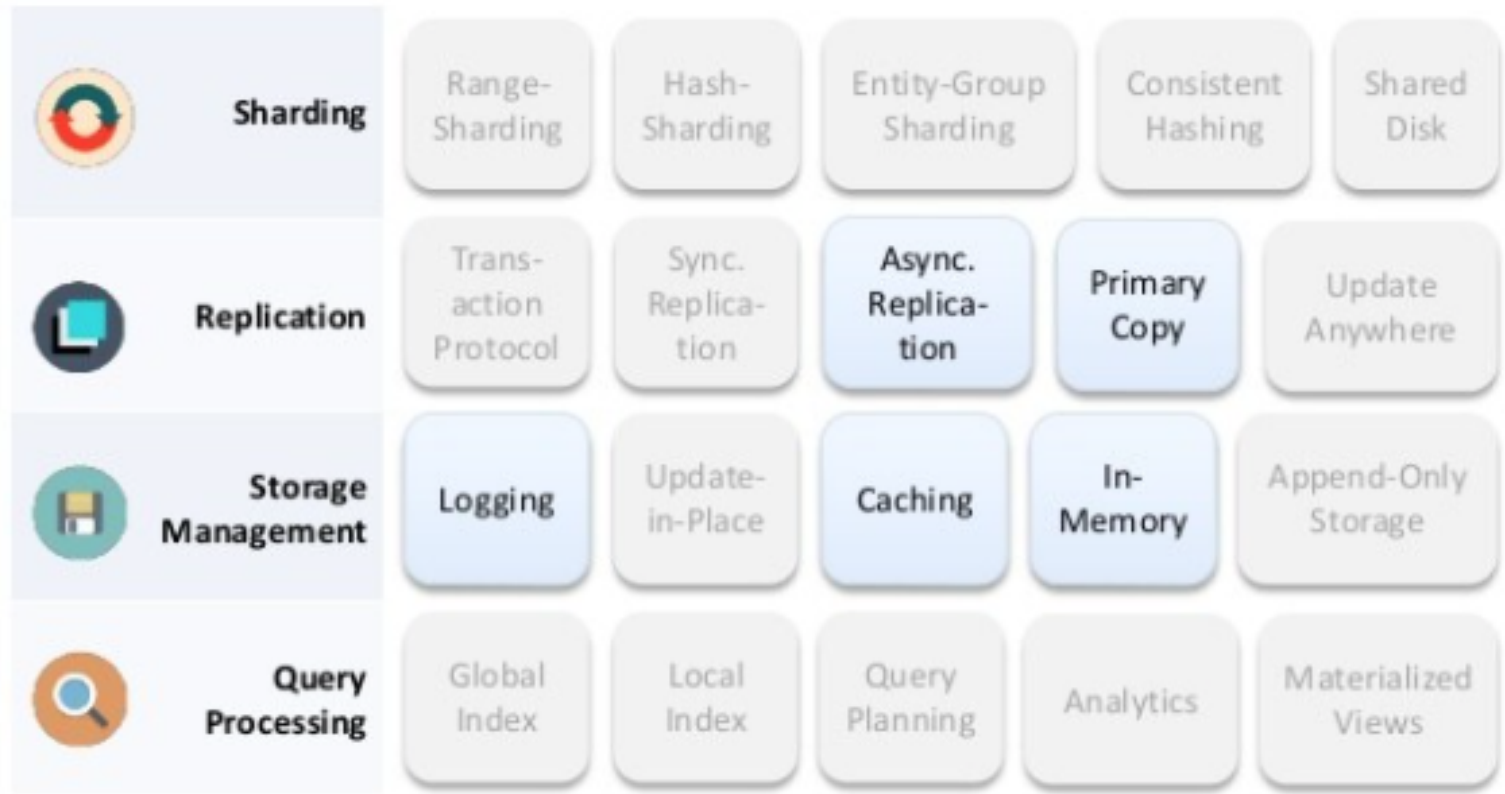
Query Features

Other commands

- `GETSET "key" "value"`
- `RPUSH "key" "value" ["value" ...]`
- ...

`https://redis.io/commands`

Redis [5]


Sharding
Range-Sharding
Hash-Sharding
Entity-Group Sharding
Consistent Hashing
Shared Disk
Replication
Trans-action Protocol
Sync. Replica-tion
Async. Replica-tion
Primary Copy
Update Anywhere
Storage Management
Logging
Update-in-Place
Caching
In-Memory
Append-Only Storage
Query Processing
Global Index
Local Index
Query Planning
Analytics
Materialized Views

Graph Database Word

Graph Databases

- Different point of view
  - Complex relations require complex joins
- Goal:
  - Capture data consisting of complex relations
  - Data naturally modeled as graphs
  - Social Networks, Web data, etc.
  - No joins but navigations

Graph Databases

Allow to store entities and relationships between entities

- Nodes
- Edges

A node is an instance of an entity and the relations between entities are modeled as edges

- Nodes and edges can have properties and types
- Edges have directional significance

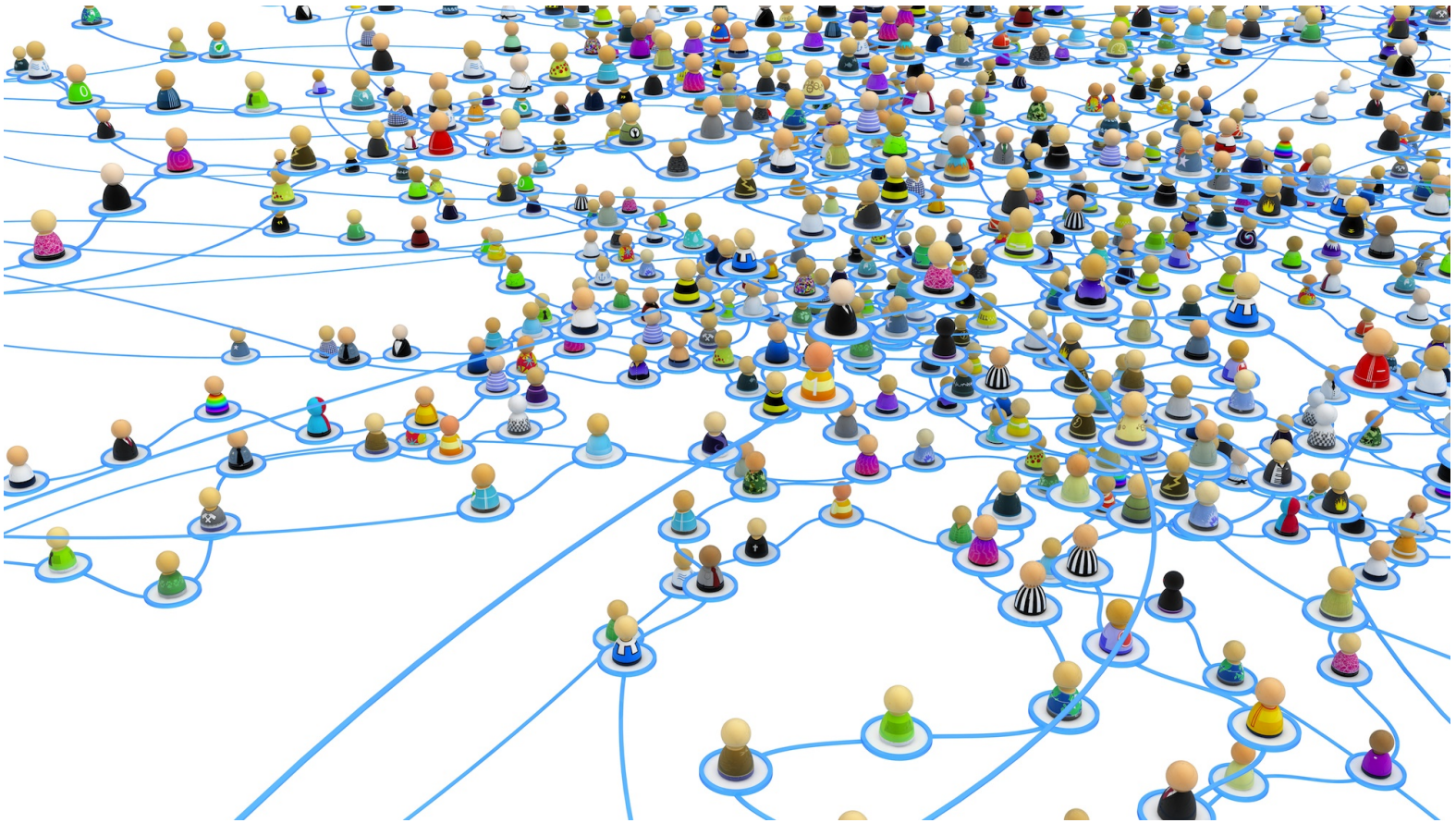
Operations:

- Traversals
  - edges following on the basis of some condition
- Shortest path
  - going from one node to another following the shortest path
- Optimal path
  - we want to include some nodes

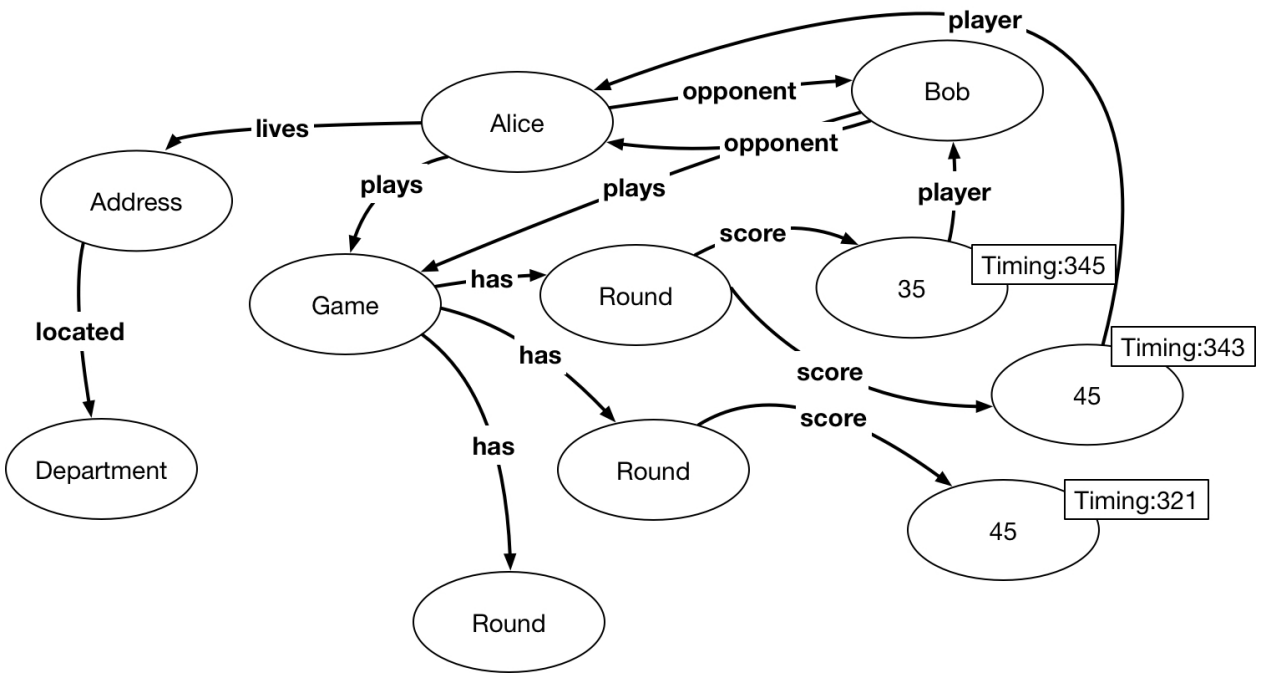
Graph Databases

- Not aggregate oriented
  - data represented as nodes connected with edges
- Data Model:
  - Nodes connected by edges
  - Many variations:
    - Neo4j stores java objects to nodes and edges in a schemaless fashion
    - Infinite Graph stores Java objects, which are subclasses of built in types as nodes and edges
    - FlockDB simply nodes and edges with no additional attributes
- Access to data:
  - Navigation through the network of edges
  - Need of a starting node
  - Nodes can be indexed by an attribute

Graph Databases



Example


player
Alice
opponent
Bob
lives
opponent
Address
plays
plays
player
score
Timing:345
Game
has
Round
35
located
has
Timing:343
score
45
Department
has
score
Round
Timing:321
45
Round

Neo4j data model

Neo4j is a highly scalable native graph database

- Represents data in nodes, relationships and properties
- Both nodes and relationships contain properties
- Relationships connect nodes
- Properties are key-value pairs
- Relationships have directions: unidirectional and bidirectional

Neo4j data model

- How we can create the nodes and assign them properties:

```
Node alice = graphDb.createNode();
  alice.setProperty("name", "Alice");

Node bob = graphDb.createNode();
bob.setProperty("name", "Bob");
```

Relationships

- Relationships are first class citizens in graph databases:
  - Type
  - Start node
  - End node
  - Properties
- The properties can be used to query the graph
- Design work to model relationships

Neo4j data model

- How we can create relationships:

```
alice.createRelationshipTo(bob, OPPONENT);

bob.createRelationshipTo(alice, OPPONENT);
```

Query features

General

- Specific graph query languages
  - Gremlin: a domain specific language for traversing graphs
  - Cypher an SQL-like query language used in Neo4j

Neo4j

- Cypher
- Features for:
  - traversing the graph
  - querying the graph properties
  - navigating nodes using language bindings

Traversals

General

- Powerful when it is necessary to traverse the graph at any depth and specify a starting node for the traversal
- Very powerful when you do not know how manage the depth of the query
- Very powerful when you have to go more levels down
- It is also possible to make the traversal top-down

Traversals

Neo4j

- The `friendsTraverser`
  - finds all the nodes that are related to Alice where the relationship type is `OPPONENT`
  - The nodes can be at any depth—opponent

```
Node alice = nodeIndex.get("name", "Alice").getSingle();
Traverser friendsTraverser = alice.traverse(Order.BREADTH_FIRST,
StopEvaluator.END_OF_GRAPH,
ReturnableEvaluator.ALL_BUT_START_NODE,
EdgeType.OPPONENT,
Direction.OUTGOING);
```

Traversals

Neo4j

Suppose that we want to discover if Alice and Bob have ever been opponents in a game

- We retrieve all the paths (all the times they were opponents)

```
Node alice = nodeIndex.get("name", "Alice").getSingle();
Node bob = nodeIndex.get("name", "Bob").getSingle();
PathFinder<Path> finder = GraphAlgoFactory.allPaths(
Traversal.expanderForTypes(FRIEND, Direction.OUTGOING)
    ,MAX_DEPTH);
Iterable<Path> paths = finder.findAllPaths(alice, bob);
```

Finds all the paths and their distance

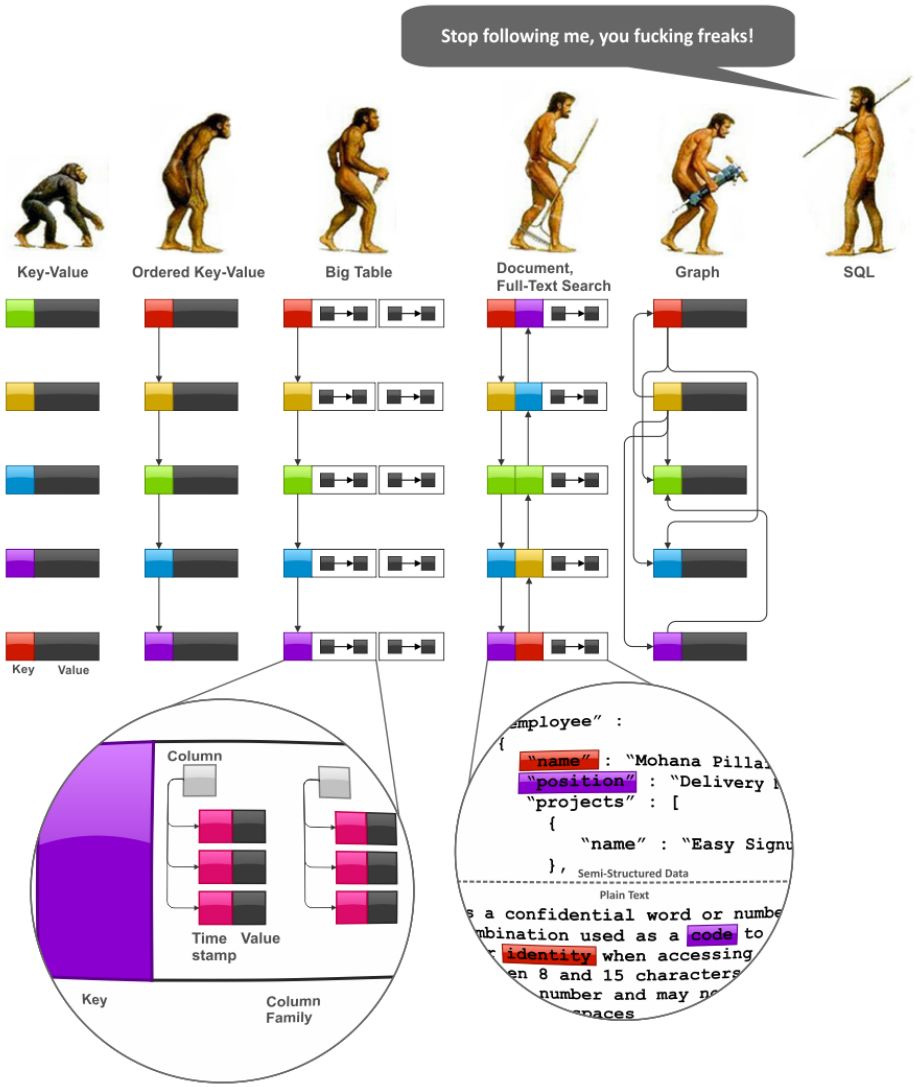
Traversals

- If there are multiple paths the shortest path is found:

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

Finds the shortest path

Comparing [7]...


Stop following me, you fucking freaks!
Key-Value
Ordered Key-Value
Big Table
Document, Full-Text Search
Graph
SQL
Key
Value
Column
Time stamp
Value
Key
Column Family
"name" : "Mohana Pilla
"position" : "Delivery
"projects" : [
"name" : "Easy Signu
Semi-Structured Data
Plain Text
code
identity
when accessing

References I


Francesca Bugiotti, Luca Cabibbo, Paolo Atzeni, and Riccardo Torlone.
Database design for nosql systems.
In *Conceptual Modeling - 33rd International Conference, ER 2014, Atlanta, GA, USA, October 27-29, 2014. Proceedings*, pages 223–231, 2014.


Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Michael Burrows, Tushar Chandra, Andrew Fikes, and Robert Gruber.
Bigtable: A distributed storage system for structured data (awarded best paper!).
In *7th Symposium on Operating Systems Design and Implementation (OSDI '06), November 6-8, Seattle, WA, USA*, pages 205–218, 2006.

References II


E. F. Codd.
Relational completeness of data base sublanguages.
IBM Research Report, RJ987, 1972.


Eric Evans.
Domain-Driven Design: Tacking Complexity In the Heart of Software.
Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.


Felix Gessert and Norbert Ritter.
Scalable data management: Nosql data stores in research and practice.
In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 1420–1423, 2016.

References III

Katarina Grolinger, Wilson Higashino, Abhinav Tiwari, and Miriam Capretz.
Data management in cloud environments: Nosql and newsql data stores.
Journal of Cloud Computing: Advances, Systems and Application, 2, 12 2013.

NoSQL Data Modeling Techniques.
`https://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling-techniques/`.
Accessed 2016.