NoSQL Solutions

History, Concepts and offerings

14 setp 2017

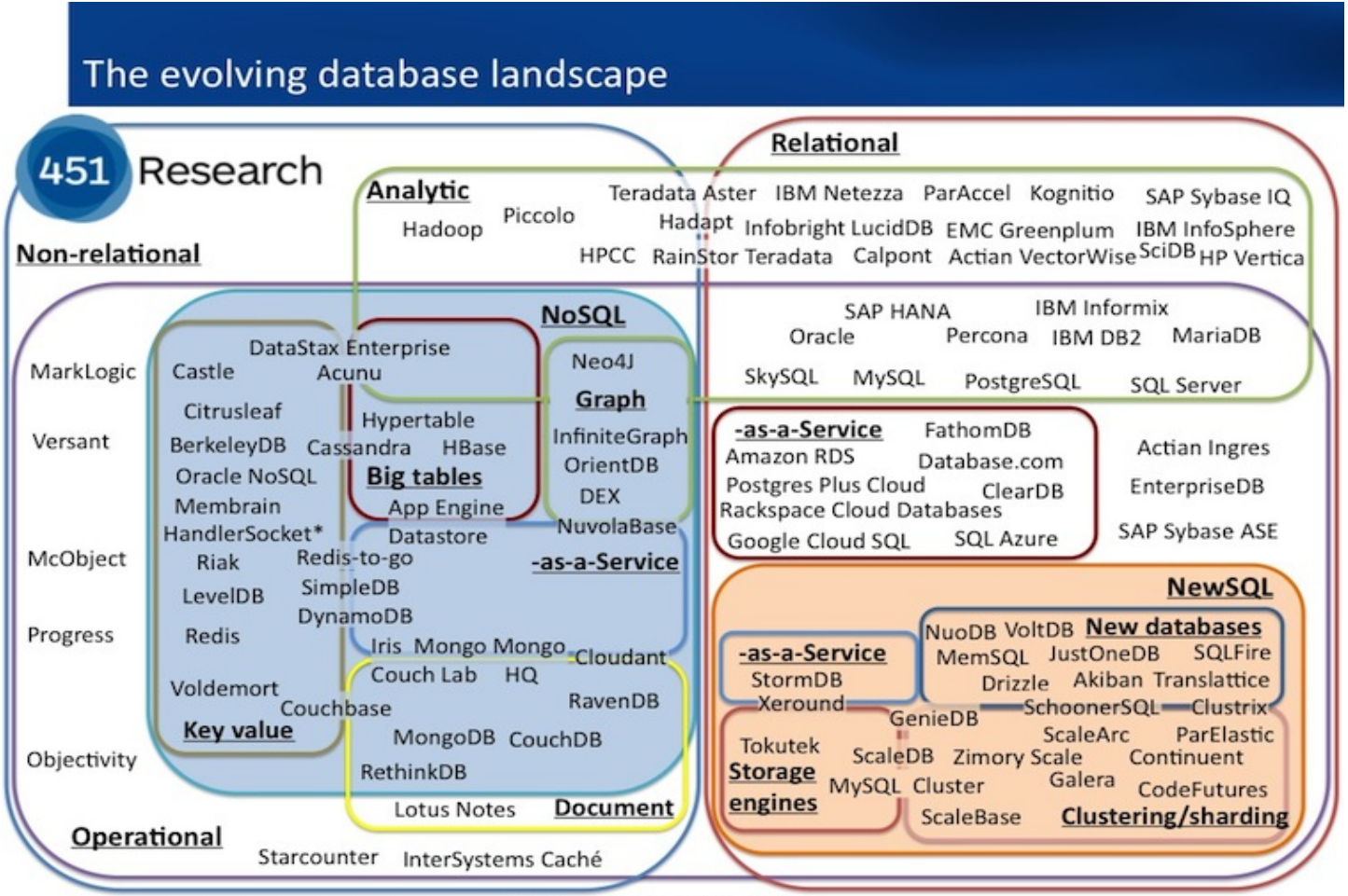
Click to edit Master subtitle sty

Christophe Menichetti
Lead Architect (HPDA/AI) – Cognitive Systems

IBM Montpellier Client Center (IBMCCMPL)

GEND

GEND



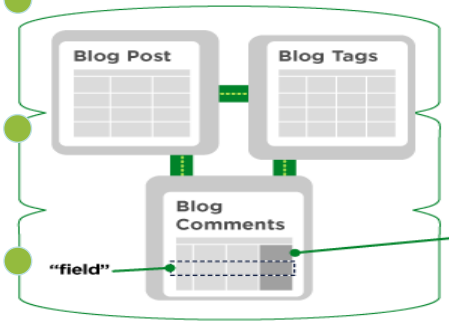
Why NoSQL ?

Imagine ecommerce company, with web site where you can buy items and post comments

RELATIONAL VS. NON-RELATIONAL DATABASES

upwork

- Highly Structured Organisation with efficient structured language (SQL)
- ACID properties to guarantee important transactions (ex financial core operations)
- Software developpers know the structure and do not worry about data schema



A relational database table organizes structured data fields into defined columns.

A non-relational database does not incorporate the table model. Instead, data can be stored in a single document file.

Why NoSQL ?

Ideal for: user session logs, ecommerce, shopping cart, recommandations, product catalog, analytics (where ACID is not mandatory)

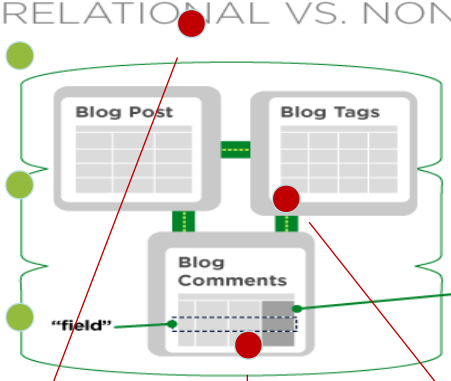
RELATIONAL VS. NON-RELATIONAL DATABASES

upwork

Highly Structured Organisation with efficient structured language (SQL)

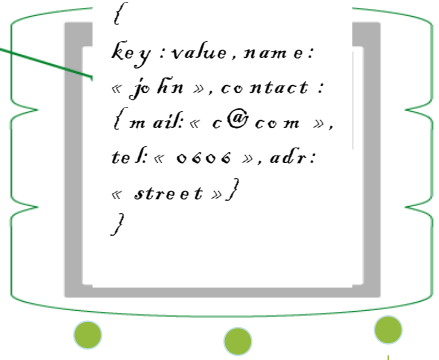
ACID properties to guarantee important transactions (ex financial core operations)

Software developpers know the structure and do not worry about data schema



A non-relational database does not incorporate the table model. Instead, data can be stored in a single document file.

A relational database table organizes structured data fields into defined columns.



NOT FLEXIBLE : if your business change you must add new table, with new relationships

NOT SCALABLE : it's really hard (or cost expenseive) to scale (deploy on cluster) join, primary key, foreign key, the whole structure making SQL compliant but rigid

NOT OPTIMISED FOR WEB / MOBILE :
Because comments could be blank, you could find many « blank » in your table, becoming a parse table, expensive in terms of storage

JSON format for flexibility

Schema on read (not on write)

No foreign key, primary key for scalability

NoSQL Databases Store data differently than RDBMS – simple example

Relational databases generally store rows and columns in tables (70's technology, storage was expensive!)

- Minimal amount of data stored and data integrity is enforced with constraints
- *SQL> select name,OccupationName from Character, Occupation where Occupation.occupationNum = Character.occupationNum and Name='Harry'*

Name	OccupationName
Harry	*Wizard*

Character Table

SerialNum	Name	RoleNum
111	Luke	1
112	Yoda	1
113	Darth Vader	3
114	Palpatine	3
115	Hans Solo	2

Unique key

Foreign key

Role Table

occupationNum	OccupationName
1	Jedi
2	Common
3	Sith

Unique key

Most NoSQL Databases do not use joins

- Flexible schema
- Easy to adapt to changes in input data
- Data is denormalized (duplicated)
- Solutions may not be suitable where strong data credibility or compliance is required
- Data consistency and integrity are the responsibilities of the developer, not database or DBA or data modeler

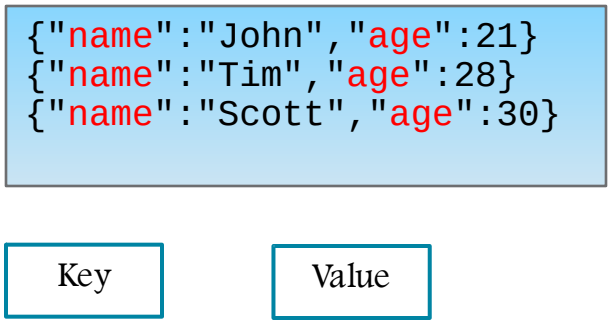
- Result: **NoSQL uses more data (duplicated), queries usually less complex = more storage, IO from memory or disk**

Character Collection

{_id}:'111'	{Name}:'Luke'	{OccupationName}:'Jedi'
{_id}:'112'	{Name}:'Yoda'	{OccupationName}:'Jedi'
{_id}:'113'	{Name}:'Darth Vader'	{PlanetName}:"Dagoba" {OccupationName}:'Sith'
{_id}:'114'	{Name}:'Palpatine'	{PlanetName}:'Coriant' {OccuptionName}:'Sith'
{_id}:'115'	{Name}:'Hans Solo'	{OccupationName}:common,'

Basic Translation Terms/Concepts

Mongo/NoSQL Terms	Traditional SQL Terms
FIND	SELECT
SAVE	INSERT
REMOVE	DELETE
UPDATE	UPDATE
ensureIndex	CREATE INDEX
SORT()	ORDER BY
LIMIT	LIMIT/FIRSTN



Gartner Magic Quadrant for DBMS (2013 vs 2015)

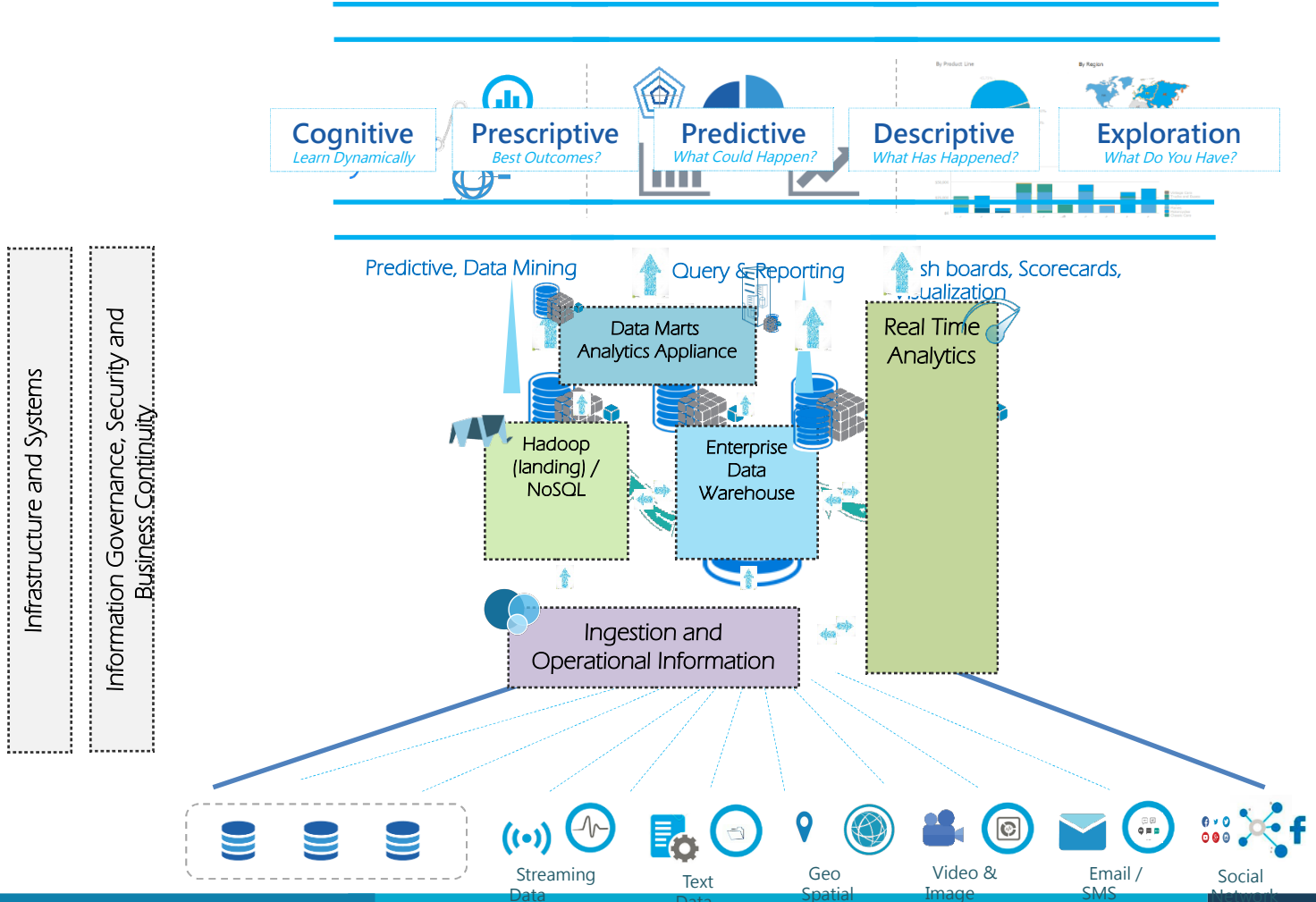
Open Source is on the move….

Open Source DB Momentum – another view: popularity

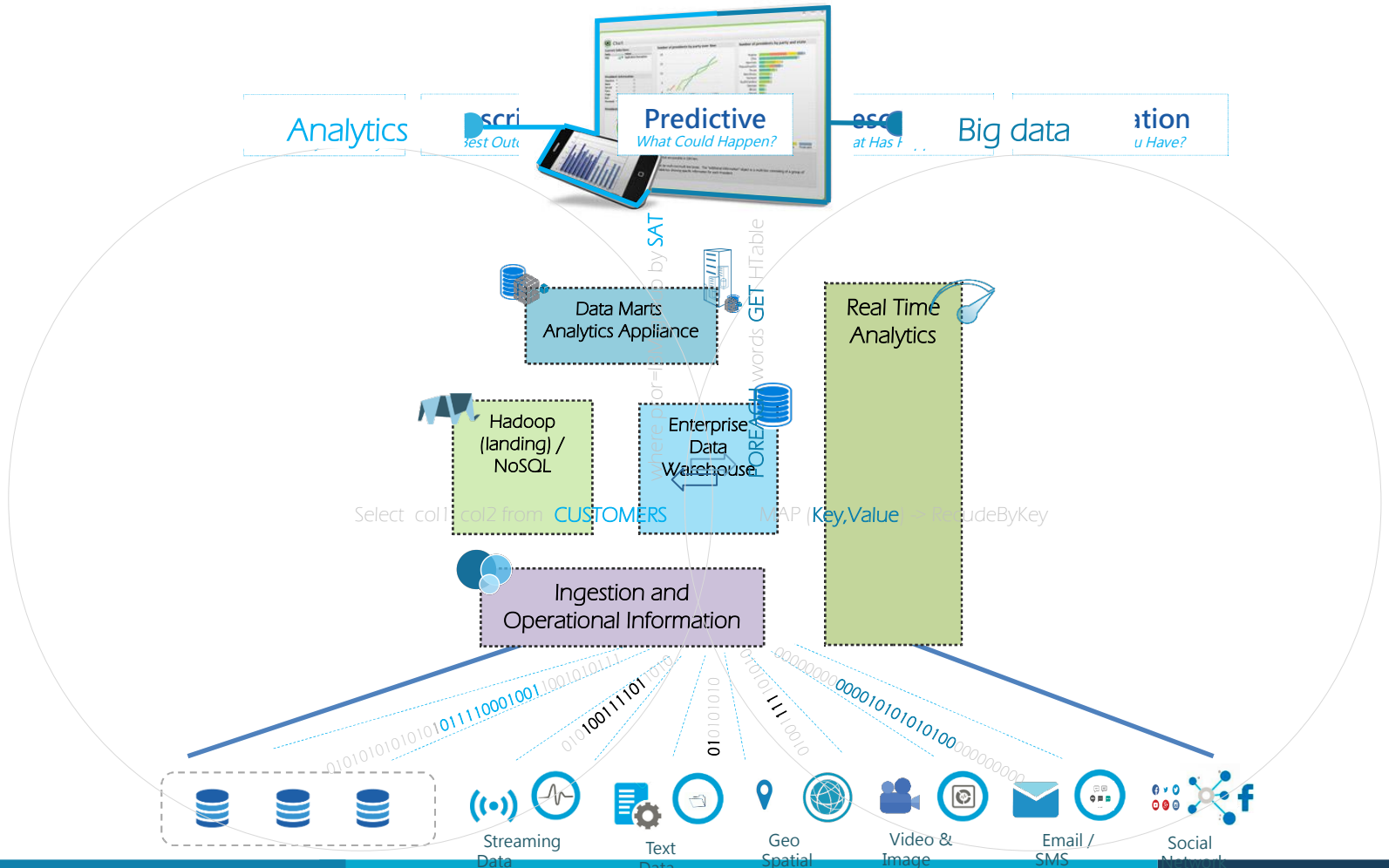
RANK	DBMS	MODEL	SCORE	GROWTH (20 MO)
1.	Oracle	RDBMS	1,442	-5%
2.	MySQL	RDBMS	1,294	2%
3.	Microsoft SQL Server	RDBMS	1,131	-10%
4.	MongoDB	Document Store	277	172%
5.	PostgreSQL (EDB, Splendid Data, 2nd Quadrant)	RDBMS	273	40%
6.	DB2	RDBMS	201	11%
7.	Microsoft Access	RDBMS	146	-26%
8.	Cassandra	Wide Column	107	87%
9.	SQLite	RDBMS	105	19%

Source: http://db-engines.com/en/ranking

Big Data and Analytics Architecture



Big Data and Analytics Architecture



GEND

Il existe plusieurs types de DB

- Bases de Données Relationnelles (SQL)

- Bases de Données non Relationnelles (noSQL)

  - Documents

  - Key-Value

  - Graph

  - Colonnes

Différences fondamentales entre RDMS et NoSQL

SQL -> ACID : Atomique, Consistante, Isolée, Durable

NoSQL -> BASE : Basic Availability, Soft-state, Eventual consistency

Modèles de données des bases NoSQL

Document databases : Enregistrement des données au format « documents »

Elles sont utilisées pour la gestion de contenus, les plateformes blog, l'analyse de données Web, l'analyse temps-réel et les applications de e-commerce.

Graph databases : Emphase sur les connections enregistrées sous forme de graphe

Elles sont dédiées à la gestion de la relation entre des données ou objets comme dans les réseau sociaux, les problématiques de logistiques, les systèmes de recommandation ou le routage d'informations.

Key-value databases : Modèle de données simple associant une valeur à une clef

Elles sont largement utilisées pour enregistrer des informations liées à une session utilisateur, les profiles, les préférences ou le panier d'achats.

Column family databases : Enregistrement sous forme de table avec un très grand nombre de colonnes

Elles sont spécialisées dans la lecture intensive comme pour l'enregistrement agrégé de logs et des données venant d'objets

Une même base peut couvrir plusieurs usages avec plusieurs modèles de données !

Source: TechTarget

Base de données NoSQL: Key-value databases : REDIS

L'usage de base de Redis, c'est de stocker des valeurs associées à des clés. Le serveur Redis est comme un gigantesque Hash Map.

```
"cle1" => valeur1
"cle2" => valeur2
etc
```

La clé doit être une chaîne de caractères. La valeur peut être n'importe quoi, vraiment, mais généralement c'est un gros bloc de texte. On utilise des commandes simple comme SET, GET, INCR, DECR, etc

```
127.0.0.1:6379> GET une_cle
(nil)
127.0.0.1:6379> SET une_cle "Une valeur"
OK
127.0.0.1:6379> GET une_cle
"Une valeur"
127.0.0.1:6379> SET une_cle 780708
OK
127.0.0.1:6379> GET une_cle
"780708"
```

Base de données NoSQL: Base de type document : MongoDB

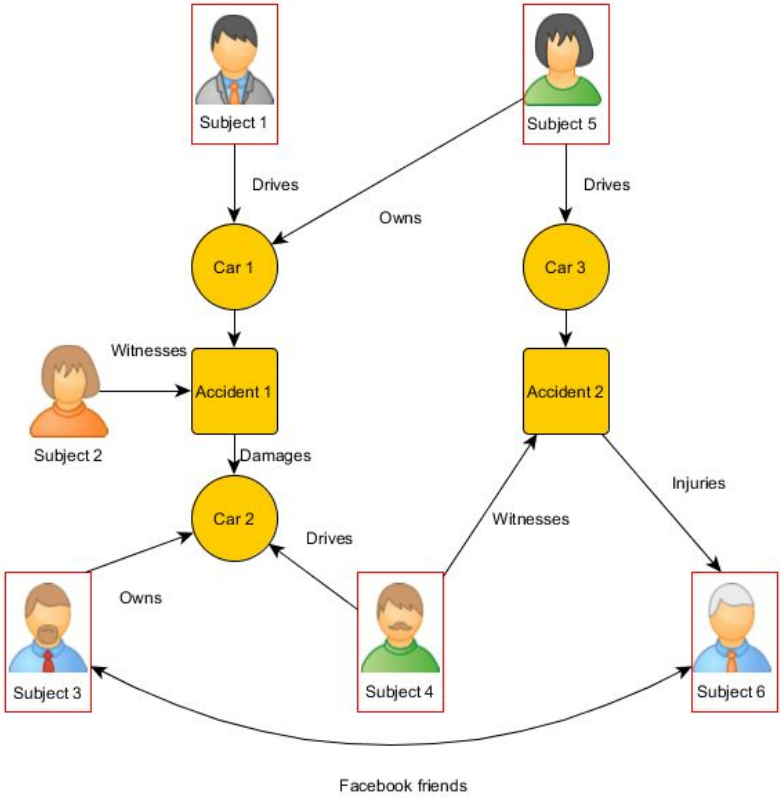
Un objet contient donc des paires clés/valeurs et ou des tableaux séparés par des virgules.

```
{"nom": "Phoenix", "prenom" : "Joaquim",
        "couleurs_preferees" : ["vert", "marron"]}
```

Sachant que la valeur d'une paire clé/valeur peut être un objet et que le tableau peut contenir des objets, on peut ordonner les informations sur plusieurs niveaux:

```
{"acteur": { "nom": "Phoenix", "prenom" : "Joaquim" },
  "roles" : [
              {"personnage" : "Theodore", "film" : "her"},
              {"personnage" : "Leonard Kraditor","film" : "Two Lovers "}
             ]
}
```

Base de données NoSQL: Représentation Graph : Neo4J



En **base de données orientée graphe**, la table est représentée par un ensemble de nœuds mis en relation entre eux

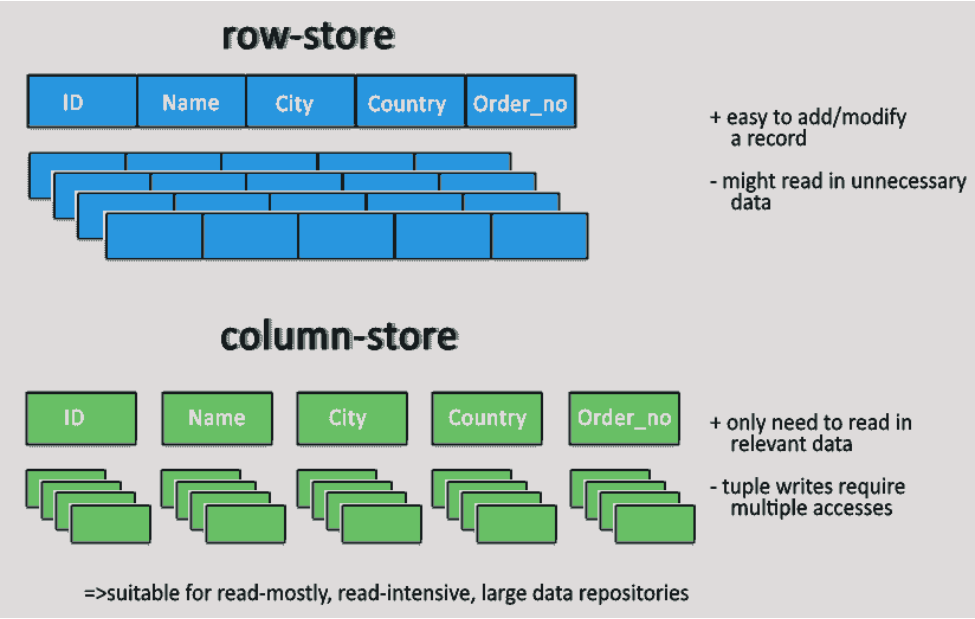
’’Subject:
identifiant : sujet 1
nom : Dupont
prénom : Jean
adresse : 96 rue des lilas
numéro client: 010203’’.

’’Car :
Identifiant : AB 123 CD
nom : Renault
modèle : Scénic"

Et la relation entre Subject 1 et Car 1 est matérialisée par un arc partant du nœud Subject 1 vers le nœud Dupont Jean, nommée ’’Drives".

Base de données NoSQL: Représentation Colonne : Cassandra

Contrairement aux moteurs orientés documents à la structure libre, les bases en colonnes stockent les données par colonnes. Cette structure permet en outre d'ajouter plus facilement une colonne à une table. Ces bases sont plébiscitées pour leur capacité à monter en charge et à accueillir une forte volumétrie de données.



Atomicity – Consistency – Isolation – Durability

- **Atomicity:** All tasks within a transaction are performed or none of them are. If one element of a transaction fails the entire transaction fails.
- **Consistency:** The transaction does not violate any protocols or rules defined in the system and the database must remain in a consistent state at the beginning and end of a transaction; there are never any half-completed transactions.
- **Isolation:** No transaction has access to any other transaction that is in an intermediate or unfinished state. This is required for both performance and consistency of transactions within a database.
- **Durability:** Once the transaction is complete, it will persist as complete and cannot be undone; it will survive system failure, power loss and other types of system breakdowns.

Basically Available – Soft State – Eventual consistency

- **Basically Available:** There will be a response to any request, BUT, that response could still be ‘failure’ to obtain the requested data or the data may be in an inconsistent or changing state, much like waiting for a cheque to clear in your bank account.
- **Soft state:** The state of the system could change over time, so even during times without input there may be changes going on due to ‘eventual consistency,’ thus the state of the system is always ‘soft.’
- **Eventual consistency:** The system will ***eventually*** become consistent once it stops receiving input. The data will propagate to everywhere it should sooner or later, but the system will continue to receive input and is not checking the consistency of every transaction before it moves onto the next one.

Trade offs when selecting your DB

Brewer's CAP Theorem (1998)

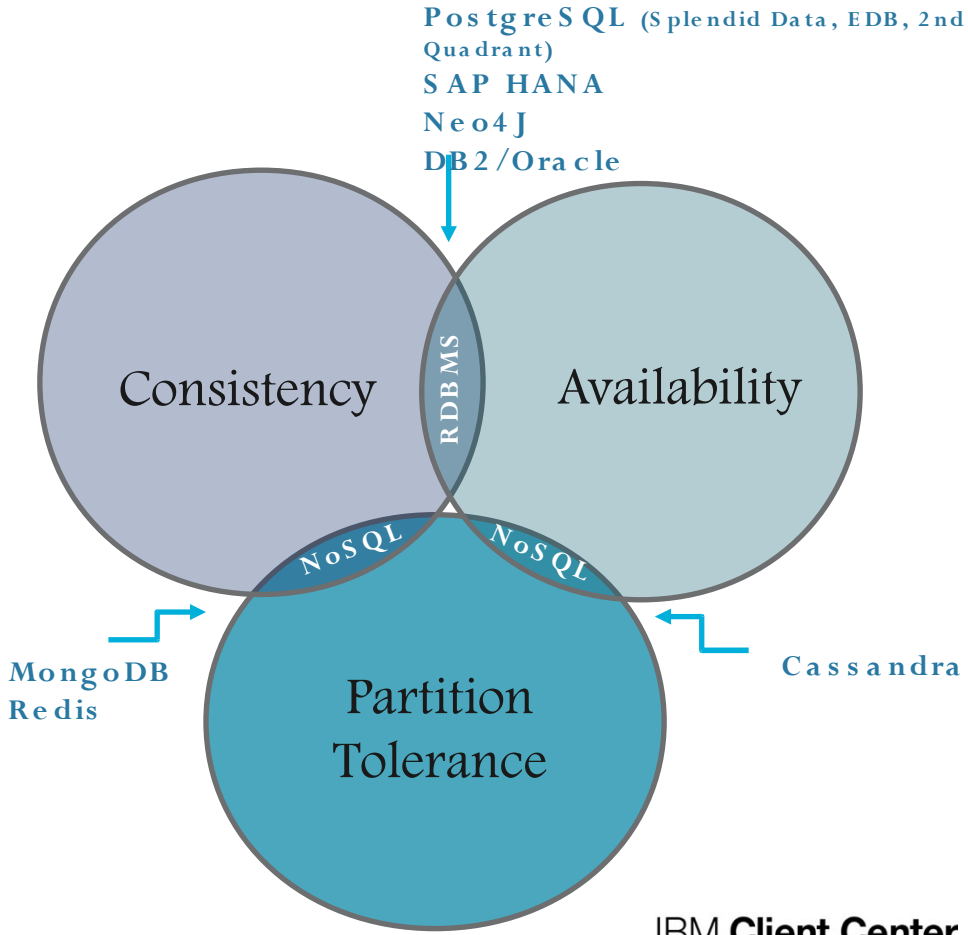
Pick any two*:

Consistency – All nodes see the same data at the same time

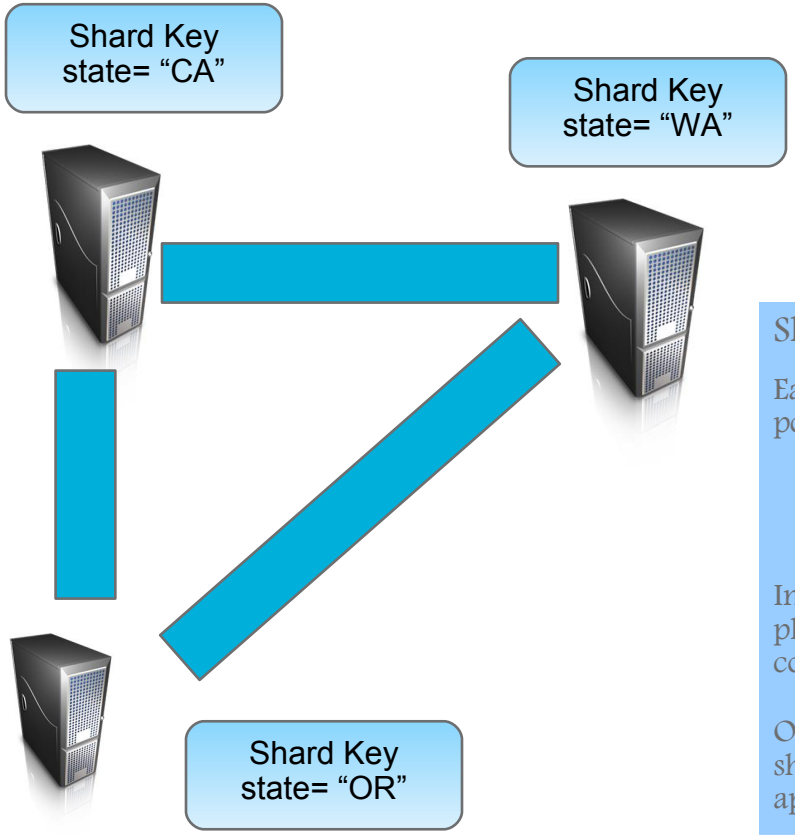
Availability – every request receives a response about whether it succeeded or failed

Partition Tolerance - The system will continue to function despite arbitrary partitioning (access to nodes) due to network failure.

*Simplification for illustration: In reality it's not this absolute. Details here are not critical for this audience, but the trade-off concept is.

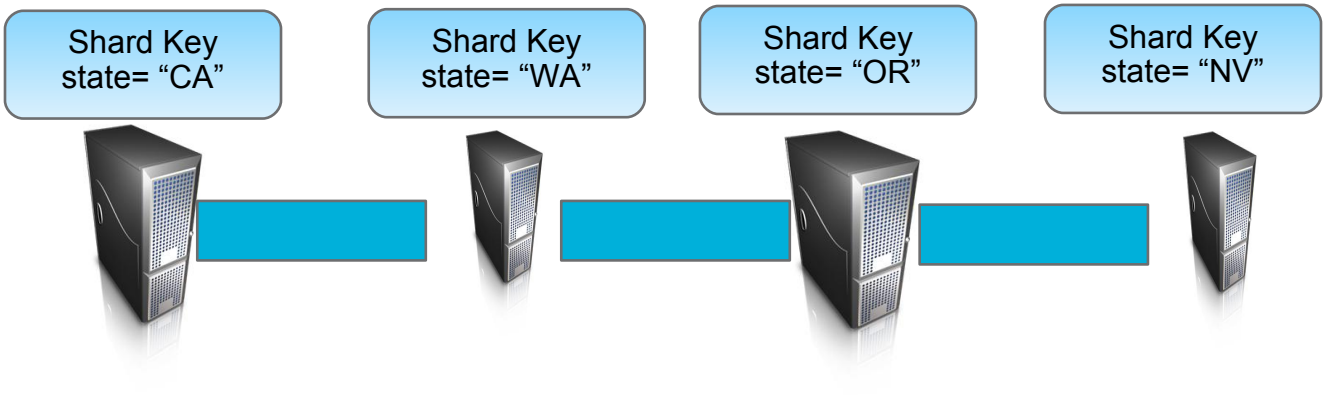


Difference between Sharding Data VS Replication



Sharding	Replication
Each node holds a portion of the data · Hash · Expression	Same data on each node
Inserted data is placed on the correct node	Data is copied to all nodes
Operations are shipped to applicable nodes	Work on local copy and modification are propagated

Scaling Out Adding a Shard



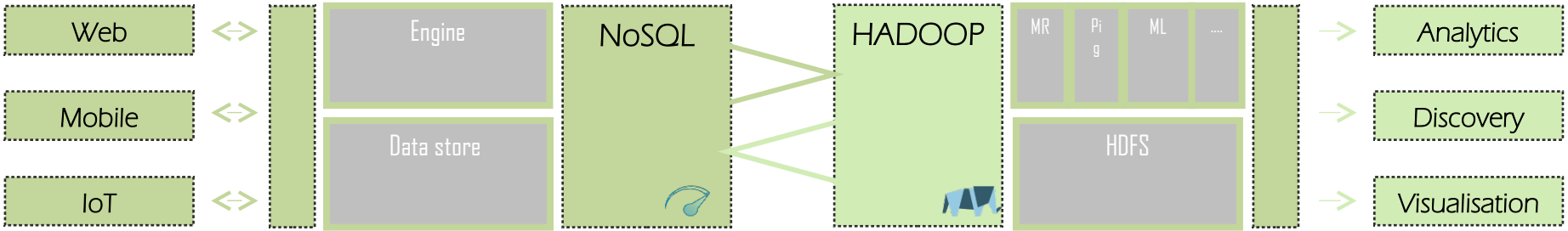
1. Send command to local node
2. New shard dynamically added, data re-distributed (if required)

GEND

HADOOP OR NoSQL ?

Performance (for huge volume)

Batch (for huge volume)



Specialized engine to retrieve data and update

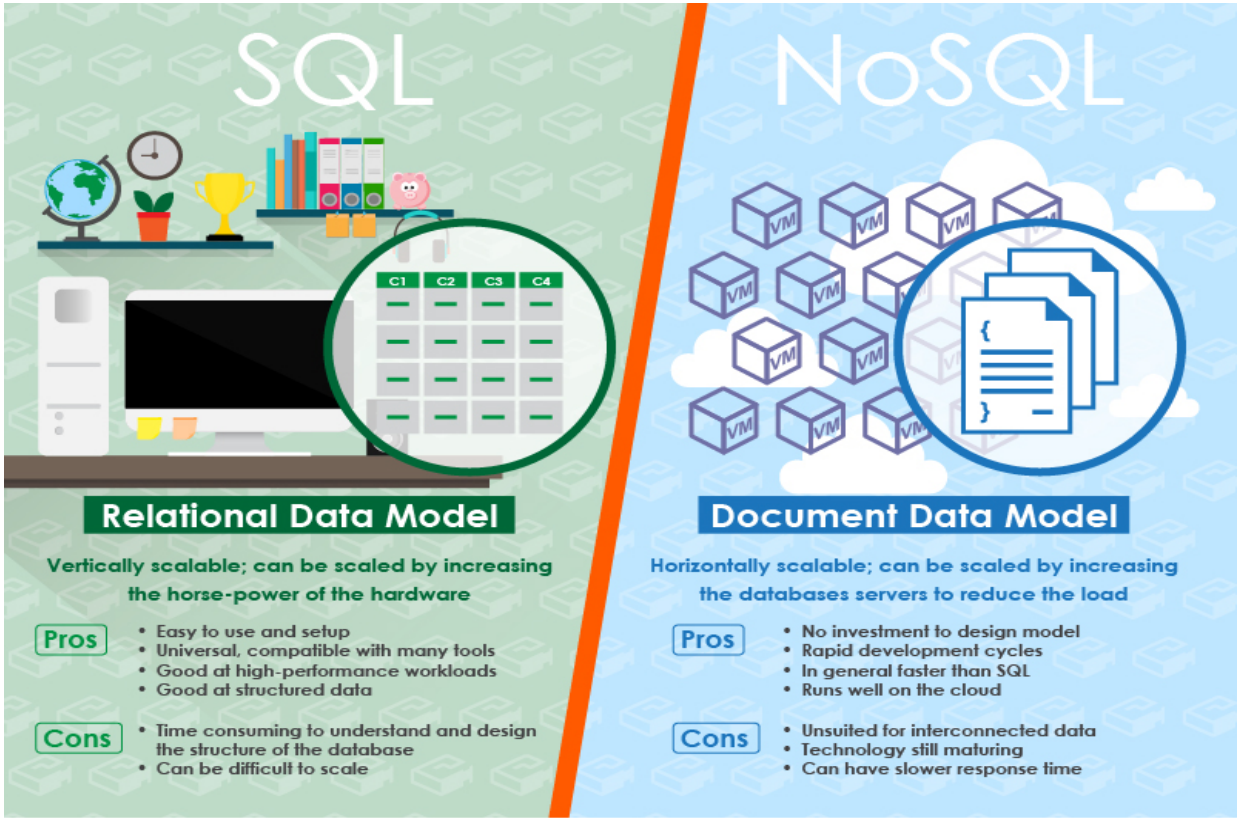
Multiple engines for multiple workloads (ML, Script, graph)

SQL OR NoSQL ?

Transactions
(consitency, reliablity)

Mature
(best for OLTP)

Vertical Scaling



Storage optimised
(for huge volume)

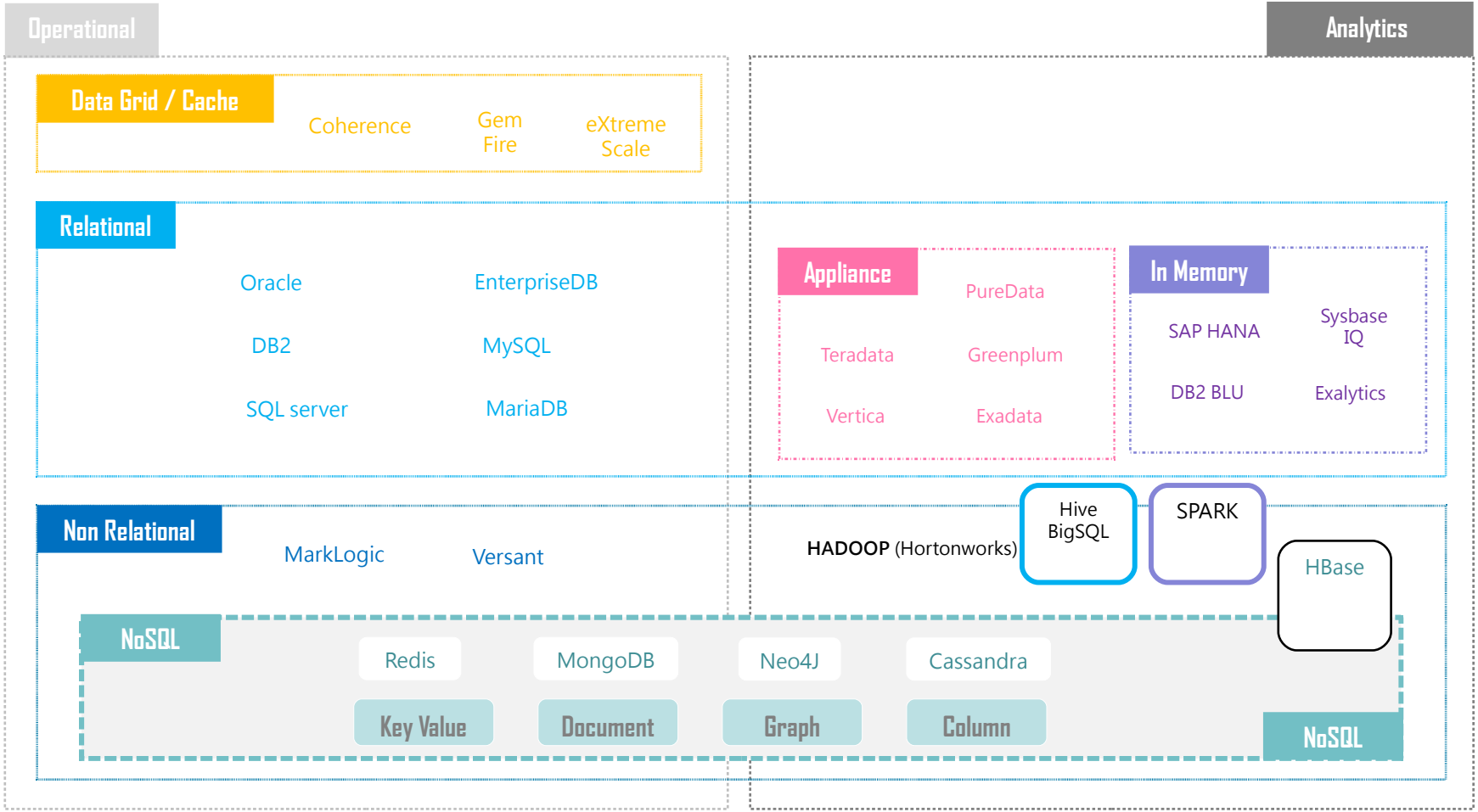
Flexible
(best for mobile/web)

Horizontal Scaling

Cas d'usages pour les bases SQL / NoSQL

	Nom	Classification	Optimisée pour	Cas d'usage / Type de données	Support de CAPI-Flash
RDBMS	PostGreSQL (EnterpriseDB, SpendidData, Dalibo)	Base de données relationnelle Open source	Données structurées. Requêtes lourdes, back-office	Migration de bases de données Oracle, compatibilité PL/SQL, syntaxe et sémantique SQL Oracle. Charge de travail transactionnelle avec un TCO avantageux	Non
	MariaDB	Base de données relationnelle Open source	Données structurées. Requêtes simples, grand nombre de transactions, portails web	Migration de MySQL (Oracle) Sites web intra/internet dynamiques sur stack LAMP (Linux Apache MariaDB PHP/Python)	Non
NoSQL	MongoDB	NoSQL - Document	Modèle document. Données semi structurées ou non structurées	Vue 360 de clients, gestion de contenus, catalogues, Internet des objets …	Non
	Redis	NoSQL – Clef-Valeur	Chaines de caractères, Listes, statistiques, images, vidéos	Cache de données, données de session utilisateur, fils de données, panier d 'achat	Oui
	Cassandra	NoSQL - Colonnes	Environnements nécessitant de très hautes performances et de grands volumes de données	Messageries, Détection de fraude, Internet des objets, capteurs, logs, ticket de communication (telco)	Oui
	Neo4J	NoSQL - Graphe	Données enregistrées sous forme de feuilles, nœuds, attributs (graphes)	Détection de fraude, réseau sociaux, Application localisées, machine learning	Oui

HADOOP, NoSQL, IN MEMORY PUZZLE



THANK YOU FOR YOUR ATTENTION

christophe.menichetti@fr.ibm.com

septembre@2017

YouTube - IBM Client Center Montpellier

Pinterest ibm-client-center-Montpellier

Twitter @IBMCCMPL

Website ibm.com/ibm/clientcenter/montpellier/

More Details

LEARNING MORE

www.bigdatauniversity.com
www.ibm.com/bigdata

BRIEFING / CONFERENCES

Any Projects

WORKSHOPS

Big Data Solutions Sizing
Big Data Design Thinking

TEST / BENCHMARK

Related Subjects

COGNITIVE

HPC / GPU and FPGA acceleration

What is Wrong With RDBMS?

- Nothing. One size fits all? Not really.
- Impedance mismatch.
  - Object Relational Mapping doesn't work quite well.
- Rigid schema design.
- Harder to scale.
- Replication.
- Joins across multiple nodes? Hard.
- How does RDMS handle data growth? Hard.
- Need for a DBA.
- Many programmers are already familiar with it.

ACID Semantics

- Atomicity: All or nothing.
- Consistency: Consistent state of data and transactions.
- Isolation: Transactions are isolated from each other.
- Durability: When the transaction is committed, state will be durable.

Any data store can achieve Atomicity, Isolation and Durability but do you always need consistency? No.

Enter CAP Theorem

- Also known as Brewer's Theorem by Prof. Eric Brewer, published in 2000 at University of Berkeley.
- "Of three properties of a shared data system: data consistency, system availability and tolerance to network partitions, only two can be achieved **at any given moment**."
- Proven by Nancy Lynch et al. MIT labs.

- http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf

CAP Semantics

- **C**onsistency: Clients should read the same data. There are many levels of consistency.
  - Strict Consistency – RDBMS.
  - Tunable Consistency – Cassandra.
  - Eventual Consistency – Amazon Dynamo.
- **A**vailability: Data to be available.
- **P**artial Tolerance: Data to be partitioned across network segments due to network failures.

BASE, an ACID Alternative

Almost the opposite of ACID.

- Basically available: Nodes in the a distributed environment can go down, but the whole system shouldn't be affected.
- Soft State (scalable): The state of the system and data changes over time.
- Eventual Consistency: Given enough time, data will be consistent across the distributed system.

A Clash of cultures

ACID:

- Strong consistency.
- Less availability.
- Pessimistic concurrency.
- Complex.

BASE:

- Availability is the most important thing. Willing to sacrifice for this (CAP).
- Weaker consistency (Eventual).
- Best effort.
- Simple and fast.
- Optimistic.