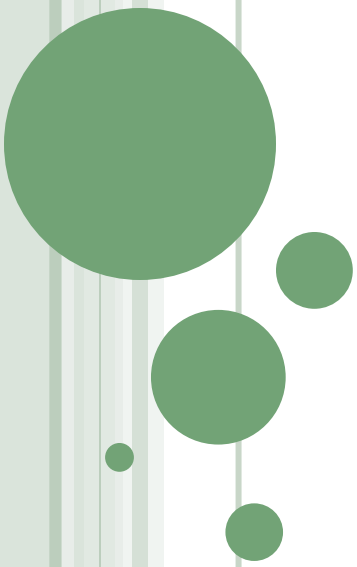
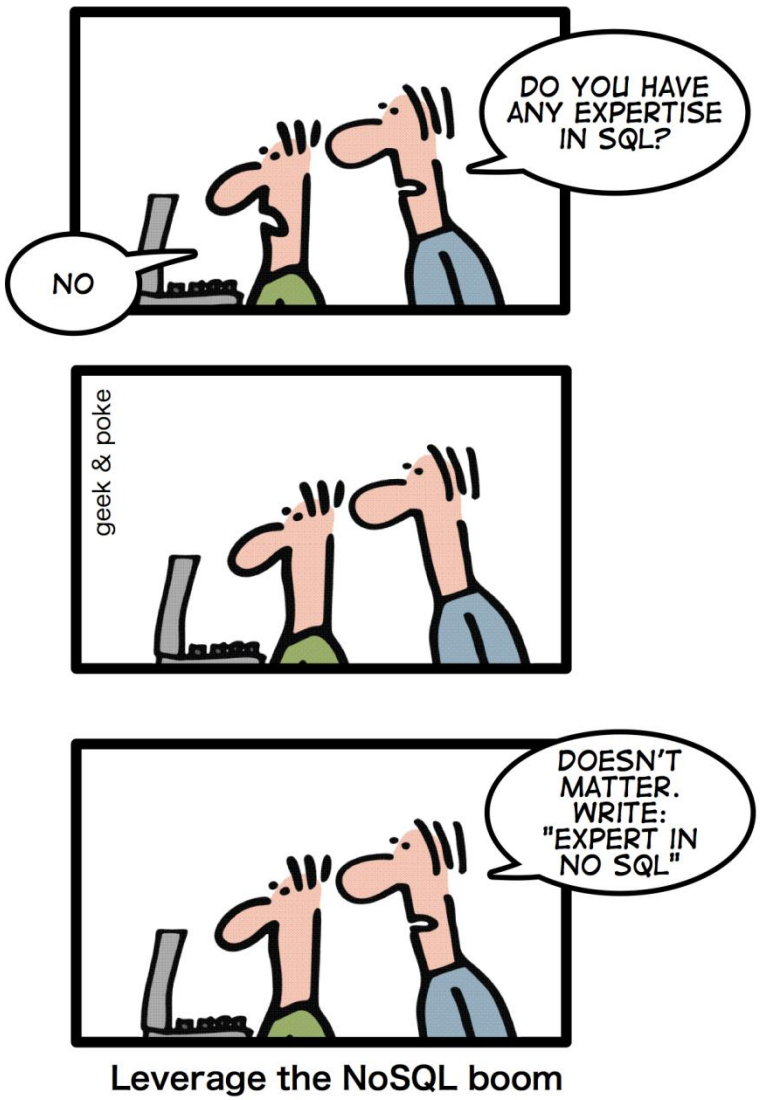


NoSQL Introducción

Enrique Barra

HOW TO WRITE A CV



http://geekandpoke.typepad.com/geekandpoke/2011/01/nosql.html?

NoSQL = Not Only SQL

Historia del término NoSQL

- El término fue acuñado en 1998 por Carlo Strozzi y resucitado en 2009 por Eric Evans

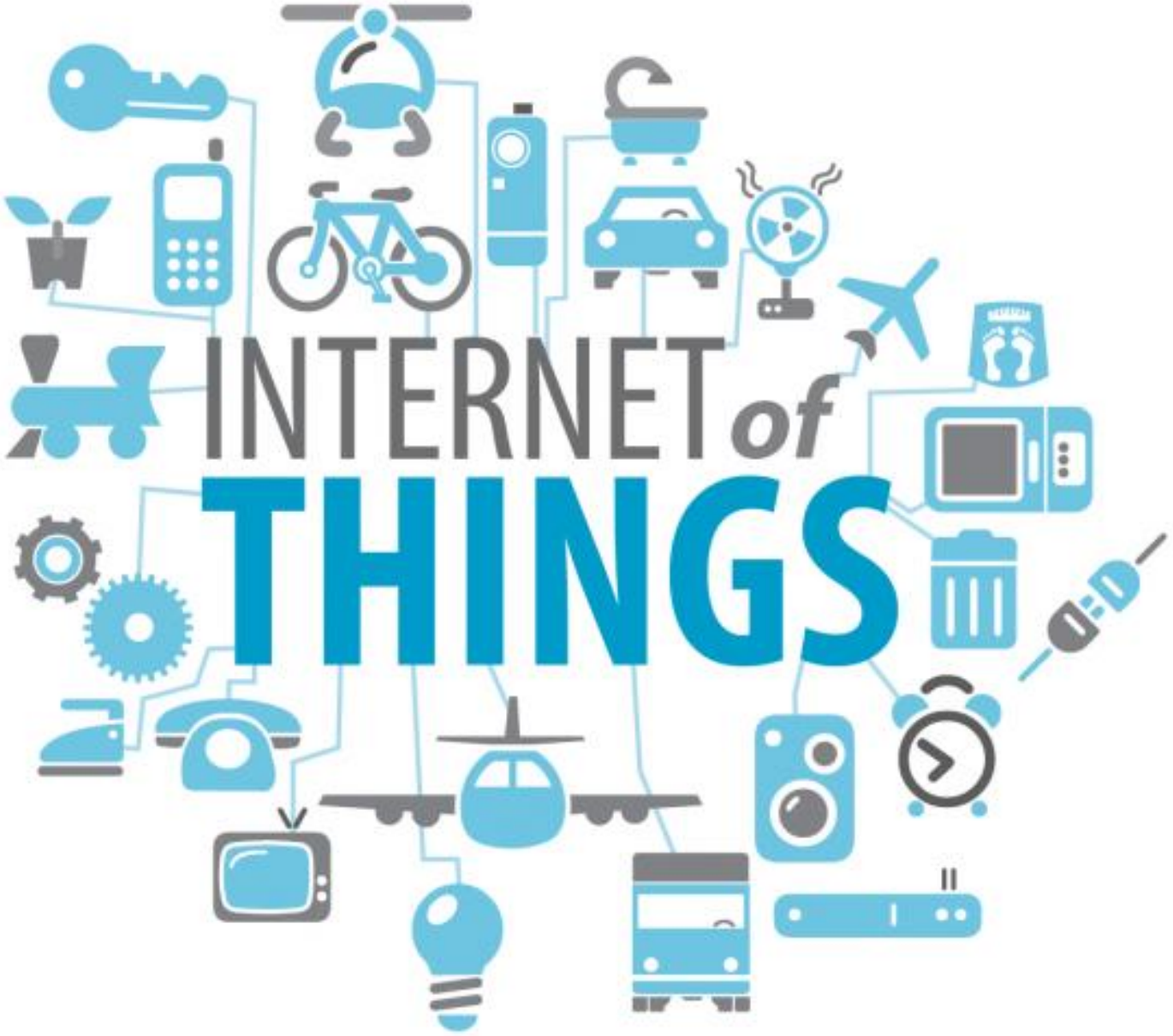
- Evans sugiere mejor referirse a esta familia de BBDD de nueva generación como **“Big Data”** mientras que Strozzi considera ahora que **NoREL** es un mejor nombre

Contexto

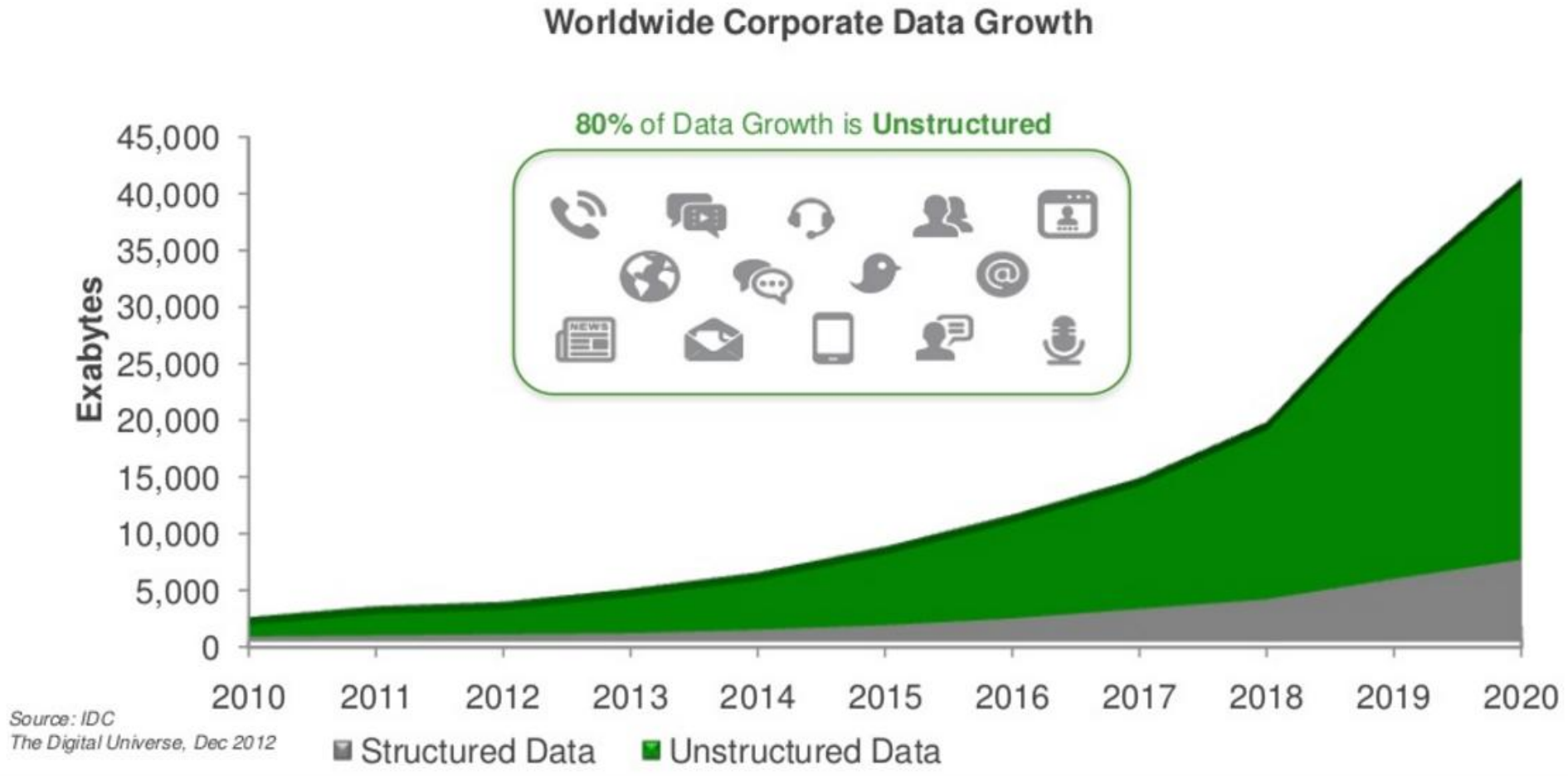
- Los dispositivos y las aplicaciones han evolucionado mucho en los últimos años
- Han aparecido nuevos dispositivos “smart”: smartphone, smart tv, wearables, coches autónomos, bombillas, …
- Las compañías han crecido vertiginosamente en:
  - Cantidad de usuarios concurrentes
  - Volúmenes de datos procesados y almacenados
  - Explotación de datos semi-estructurados y no estructurados
- La computación en la nube ha facilitado el acceso a servicios de Internet que explotan inmensos **volúmenes de datos**

https://upload.wikimedia.org/wikipedia/commons/4/46/BigData_2267x1146_white.png

http://www.smartinsights.com/internet-marketing-statistics/happens-online-60-seconds/



http://bigstartups.in/wp-content/uploads/2016/01/Internet-of-Things.png


Worldwide Corporate Data Growth
80% of Data Growth is Unstructured
Exabytes
45,000
40,000
35,000
30,000
25,000
20,000
15,000
10,000
5,000
0
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
Structured Data
Unstructured Data
Source: IDC
The Digital Universe, Dec 2012

RANKING

316 systems in ranking, October 2016

Rank Oct 2016	Rank Sep 2016	Rank Oct 2015	DBMS	Database Model	Score Oct 2016	Score Sep 2016	Score Oct 2015
1.	1.	1.	Oracle	Relational DBMS	1417.10	-8.46	-49.85
2.	2.	2.	MySQL	Relational DBMS	1362.65	+8.62	+83.69
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1214.18	+2.62	+90.95
4.	↑ 5.	4.	MongoDB	Document store	318.80	+2.81	+25.54
5.	↓ 4.	5.	PostgreSQL	Relational DBMS	318.69	+2.34	+36.56
6.	6.	6.	DB2	Relational DBMS	180.56	-0.62	-26.25
7.	7.	↑ 8.	Cassandra	Wide column store	135.06	+4.57	+6.05
8.	8.	↓ 7.	Microsoft Access	Relational DBMS	124.68	+1.36	-17.16
9.	↑ 10.	↑ 10.	Redis	Key-value store	109.54	+1.75	+10.75
10.	↓ 9.	↓ 9.	SQLite	Relational DBMS	108.57	-0.05	+5.90
11.	11.	↑ 14.	Elasticsearch	Search engine	99.12	+2.64	+28.89
12.	12.	↑ 13.	Teradata	Relational DBMS	76.23	+3.17	+2.79
13.	13.	↓ 11.	SAP Adaptive Server	Relational DBMS	69.48	+0.32	-16.16
14.	14.	↓ 12.	Solr	Search engine	66.57	-0.39	-12.50
15.	15.	15.	HBase	Wide column store	58.19	+0.38	+0.95

RDBMS vs. NoSQL

- Los RDBMS tradicionales nos permiten definir la estructura de un esquema que demanda reglas rígidas y garantizan ACID

- Las aplicaciones web y sistemas de información modernos presentan desafíos muy distintos a los sistemas empresariales tradicionales:
  - Datos a escala web (web-scale)
  - Alta frecuencia de lecturas y escrituras
  - Cambios de esquema de datos frecuentes
  - Las aplicaciones sociales no necesitan el mismo nivel de ACID

- Consecuencia → aparición de soluciones NoSQL
  - Cassandra, MongoDB, Jackrabbit , CouchDB, BigTable, Dynamo o Neo4j

Características principales

- (No todas las BBDD NoSQL tienen todas las características)

- Guardan **datos persistentes** (no sólo cachés)
- “Fáciles” de usar en clústers de balanceo de carga convencionales → facilitan **escalabilidad horizontal**
- No tienen esquemas fijos y permite la migración del esquema sin tener que ser reiniciadas o paradas. **Tienen esquema dinámico**
- Suelen tener un **sistema de consultas propio** en vez de usar un lenguaje de consultas estándar (recordemos, NoSQL)
- Tienen propiedades ACID en un nodo del clúster y son “**eventualmente consistentes**” en el clúster
- **No hay joins** ya que es una operación compleja que combina entradas de dos o más tablas, requieren esquemas fijos y consistencia fuerte

Características principales (Arquitectura)

- A menudo ofrecen sólo **garantías de consistencia débiles**, como por ejemplo *eventual consistency*, o transacciones restringidas a elementos de datos simples

- Emplean una **arquitectura distribuida** (y tolerante a fallos), donde los datos se guardan de modo redundante en distintos servidores, a menudo usando tablas hash distribuidas

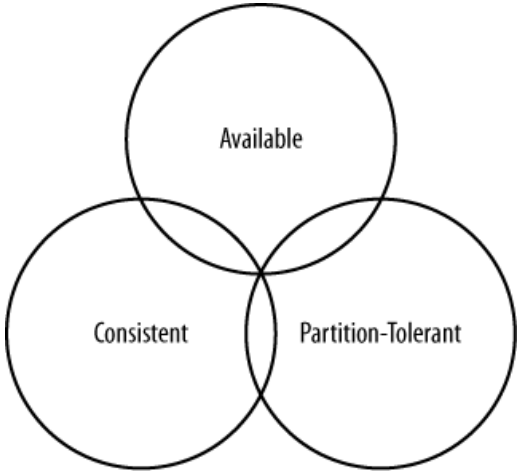
- Suelen ofrecer **estructuras de datos sencillas** como arrays asociativos o almacenes de pares clave-valor

El teorema CAP

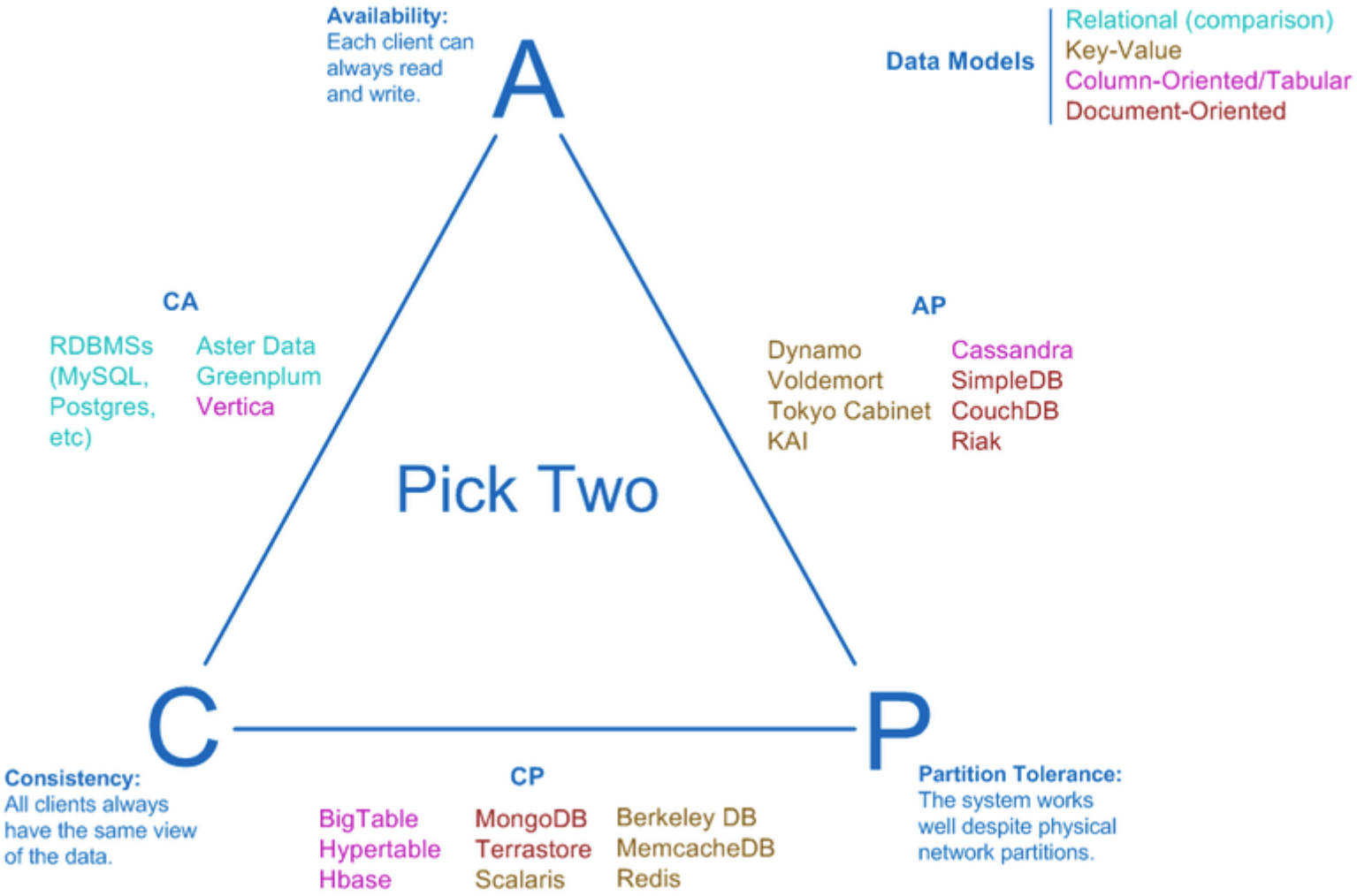
- **Teorema de Brewer**: “es imposible para un sistema computacional distribuido ofrecer simultáneamente las siguientes tres garantías”:
  - ***Consistencia*** (*Consistency*) – todos los nodos ven los mismos datos al mismo tiempo. Siempre que un dato es actualizado, todos los usuarios tienen acceso a esa última versión
  - ***Disponibilidad*** *(Availability)* – garantiza que cada petición recibe una respuesta acerca de si tuvo éxito o no. Cualquier operación puede ser ejecutada sin demora
  - ***Tolerancia a la partición*** *(Partition)* – los datos son distribuidos a través de dos o más nodos de la red y el sistema puede seguir funcionando, incluso, cuando algunos de estos nodos son totalmente inaccesibles

Equivalente a:

> “You can have it good, you can have it fast, you can have it cheap: pick two.”



CAP en RDBMS y NoSQL

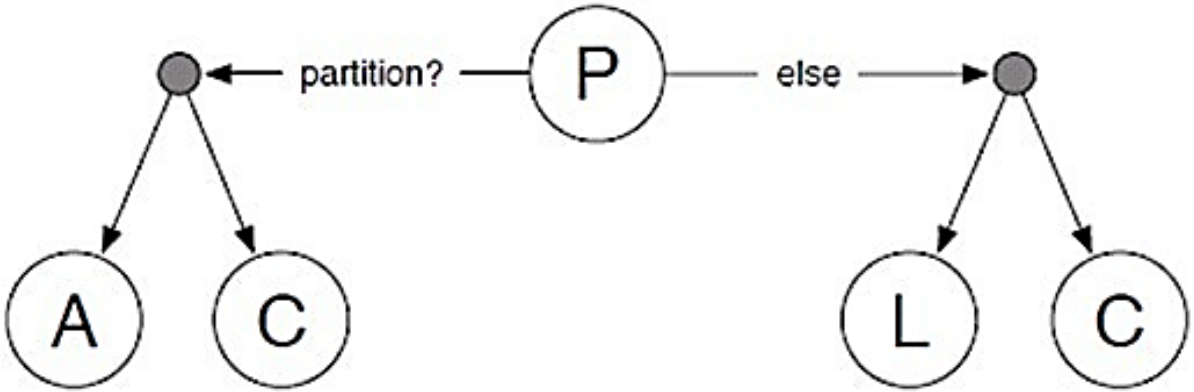


http://zohararad.github.io/presentations/big-data-introduction/img/nosql-visual-guide.png

Teorema PACELC

- “Si hay una Partición (P), un sistema escoge entre disponibilidad (A) y consistencia (C), sino (E, de else) cuando el sistema está operando normalmente ante ausencia de particiones, el sistema hace concesiones entre latencia (L) y consistencia (C).

- También llamado PACELCA, porque en el segundo caso siempre está la A



Fuente: http://eincs.com/images/2013/06/truth-of-cap-theorem-pacelc.jpg

Clasificación de sistemas de BBDD

- Considerando tanto el teorema de CAP como PACELC, podemos clasificar a algunos sistemas de bases de datos de la siguiente manera (D. J. Abadi, "Consistency Tradeoffs in Modern Distributed Database System Design," *IEEE Computer Society,* vol. 45, no. 2, pp. 37-42, 2012.)

- ***Sistemas PA/EL*:**
  - Si una partición ocurre, sacrifica consistencia por disponibilidad.
  - En operaciones normales, sacrificar consistencia por baja latencia.
  - Ejemplos de este caso pueden ser Dynamo, Cassandra y Riak.

- ***Sistemas PC/EC:***
  - No sacrifica consistencia, sacrificará disponibilidad y costos de latencia.
  - Son sistemas ACID.
  - Ejemplos de este caso serían HBase, VoltDB/H-Store y Megastore.

Clasificación de sistemas II

- **Sistemas PA/EC:**
  - Se garantiza que lecturas y escrituras sean consistentes.
  - Ejemplo: MongoDB.

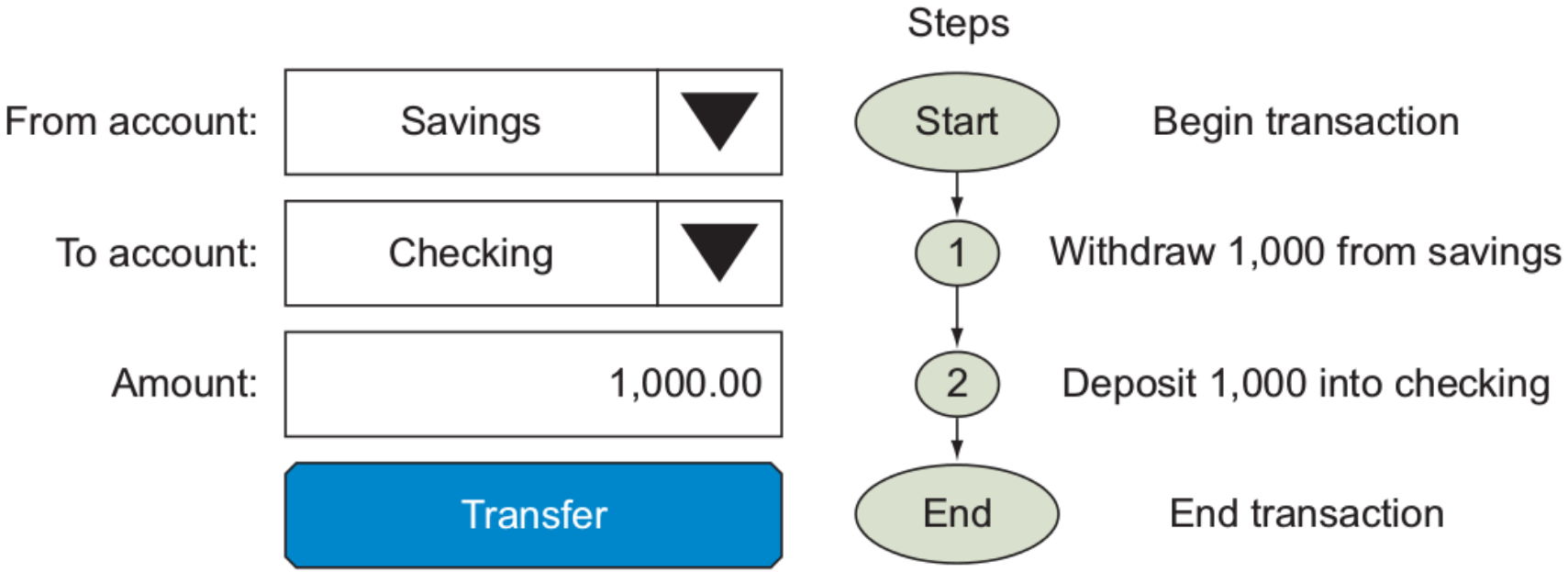
- **Sistemas PC/EL:**
  - Sacrifica consistencia por latencia en condiciones normales. Si ocurre una partición, se cambia disponibilidad por consistencia.
  - Ejemplo: PNUTS.

Recordemos ACID

- ***Atomicity - Consistency - Isolation – Durability***
- **A**tomicidad, **C**onsistencia, aislamiento y **D**urabilidad
- Una secuencia de operaciones (transacción):
  - Se ejecutará del todo o nada **(A)**
  - Una vez completada, la BD quedará en un estado en el que no se viola ninguna restricción de integridad **(C)**
  - Las transacciones concurrentes son independientes y no se afectan unas a otras **(I)**
  - Las modificaciones efectuadas por una transacción podrán recuperarse ante fallas del sistema **(D)**

- En el mundo relacional estamos familiarizados con las transacciones ACID, que garantizar la consistencia y estabilidad de las operaciones pero requieren lockings sofisticados

EJEMPLO TRANSACCIÓN ACID



McCreary, D. & Kelly, A. (2013). *Making Sense of NoSQL: A guide for managers and the rest of us. Shelter Island, NY:* Manning Publications Co.

BASE

- Las BBDD NoSQL son repositorios de almacenamiento más optimistas, siguen el **modelo BASE**:
  - **Basic availability** – el almacén funciona la mayoría del tiempo incluso ante fallos gracias al almacenamiento distribuido y replicado
  - **Soft state** – los almacenes no tienen por qué ser consistentes ni sus réplicas en todo momento.
    - El programador puede verificar esa consistencia.
  - **Eventual consistency** – la consistencia se da eventualmente. Si no ocurren nuevas actualizaciones sobre un determinado dato, en algún momento todos los usuarios tendrán acceso a la versión más actual de este

- **BASE es una alternativa flexible a ACID** para aquellos almacenes de datos que no requieren una adherencia estricta a las transacciones, tienen modelos de datos más flexibles

Conceptos asociados a BBDD distribuidas

- **Almacenes basados en columnas y filas**. RDBMS almacenan las filas de modo continuo en disco, mientras que algunas NoSQL guardan columnas de modo continuo

- **Replicación maestro-maestro** es un método de replicación de BBDD que permite almacenar datos en un grupo de nodos y su actualización por cualquier miembro del grupo
- **Replicación maestro-esclavo** donde un sólo elemento se designa como “maestro” de un datastore y es el único nodo que permite modificar datos

- **Particionado** es la división de una BBDD lógica y sus partes constituyentes en un conjunto de partes independientes
- **Sharding**: es una partición horizontal en una BBDD donde las filas de una tabla se mantienen de modo separado

- **Modelo de consistencia**: contrato entre el programador y el sistema sobre las garantías de consistencia (write & read consistency: one, all, quorum, etc.)

Soluciones NoSQL

Taxonomía de soluciones NoSQL

- Los principales tipos de BBDD de acuerdo con su implementación son los siguientes:
  - **Almacenes de Clave-Valor**
  - **Almacenes de Familia de Columnas**
  - **Almacenes de documentos**
  - **Grafos**

Soluciones Empresariales NoSQL

- **BigTable** (column oriented) es un sistema de gestión de base de datos creado por Google distribuido, de alta eficiencia y propietario.
  - Google Cloud Datastore (App Engine NoSQL Data Storage) motor de BBDD NoSQL factorizado de Google App Engine
    - BBDD de columnas que soporta transacciones ACID, tiene alta disponibilidad a través de centros de replicación y consultas SQL-like

- **Amazon DynamoDB** (key-value oriented) es un servicio de bases de datos NoSQL rápido y totalmente gestionado que permite almacenar y recuperar de manera fácil y económica cualquier cantidad de datos y atender cualquier nivel de tráfico

- Microsoft NoSQL **Windows Azure Tables**
  - http://msdn.microsoft.com/en-us/library/windowsazure/jj553018.aspx

- Comparativa:
  - http://www.theregister.co.uk/2013/05/16/google_datastore/

¿Qué tipo de BBDD elijo?

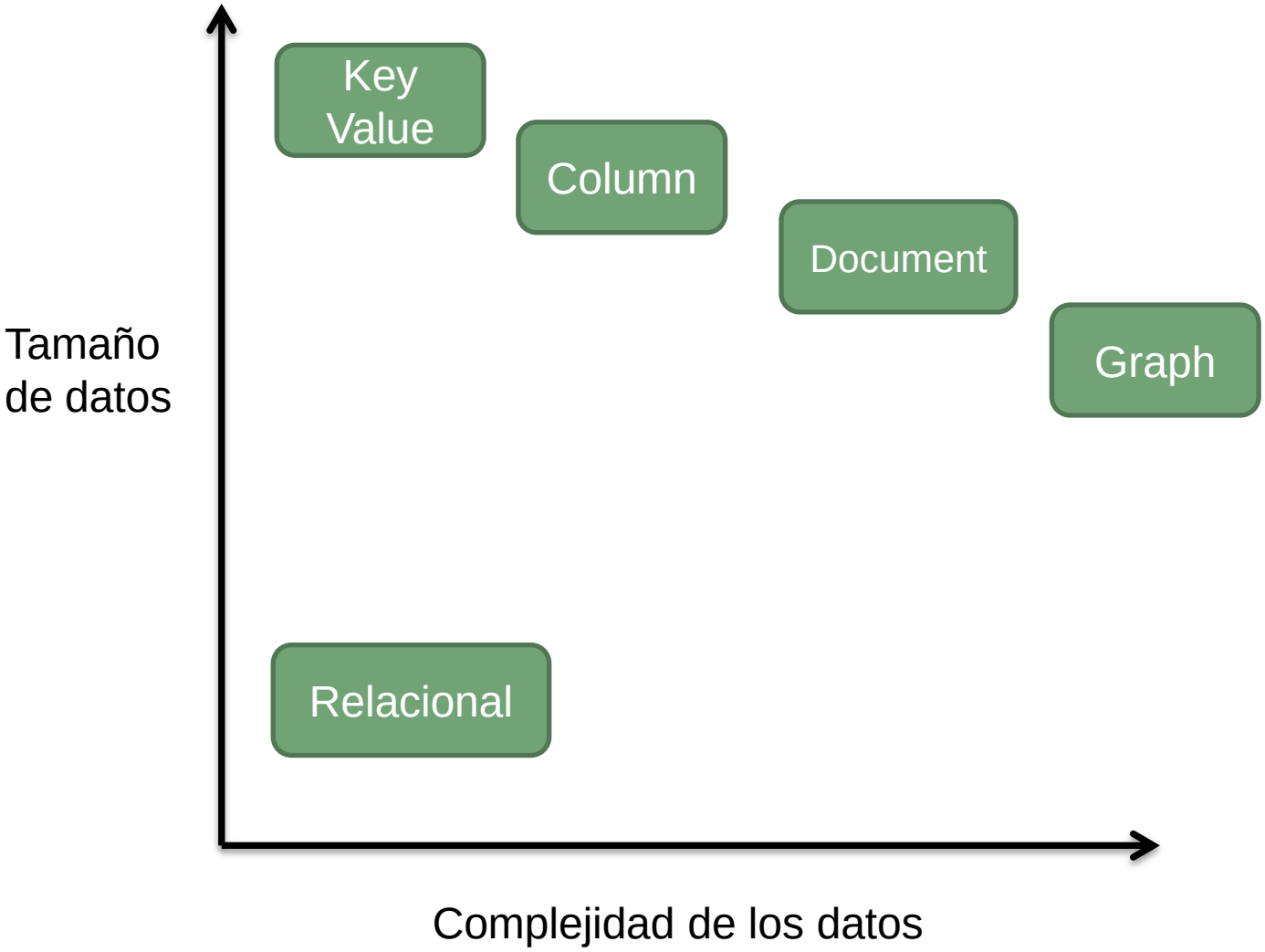
- Algunas respuestas pueden encontrarse en:
  - 35+ Use Cases For Choosing Your Next NoSQL Database
    - http://highscalability.com/blog/2011/6/20/35-use-cases-for-choosing-your-next-nosql-database.html
  - Which freaking database should I use?
    - http://www.infoworld.com/print/199184

- **¡¡¡NoSQL no es la panacea!!!**
  - Si tus datos son relacionales, quedarte con tu RDBMS sería la opción correcta
  - Evitar el hype, no elegir NoSQL porque esté de moda

¿Cuándo necesito NoSQL?

- Desafíos de gestión de información son difíciles de resolver con tecnología de bases de datos relacionales:
  - BBDD **no escala a tu tráfico** a un coste aceptable
  - El **tamaño de tu esquema de datos** ha crecido desproporcionalmente.
  - Generas **muchos datos temporales** que no corresponden al almacén de datos principal (carritos de compra, personalización de portales)
  - **BBDD ha sido desnormalizada** por razones de rendimiento o por conveniencia para utilizar los datos en una aplicación
  - **Dataset tiene** grandes cantidades de texto o imágenes, **BLOB**
  - **Consultas contra datos que no implican relaciones jerárquicas** sencillas; recomendaciones o consultas de inteligencia de negocio.
    - Ejemplo: “toda la gente en una red social que no ha comprado un libro este año que ha dejado de seguir a gente que si ha comprado un libro“
  - Usas **transacciones locales** que no necesitan ser durables, e.j. Like.
    - Los sites AJAX tienen muchos casos de uso de este estilo.

¿Qué tipo de BBDD elijo?



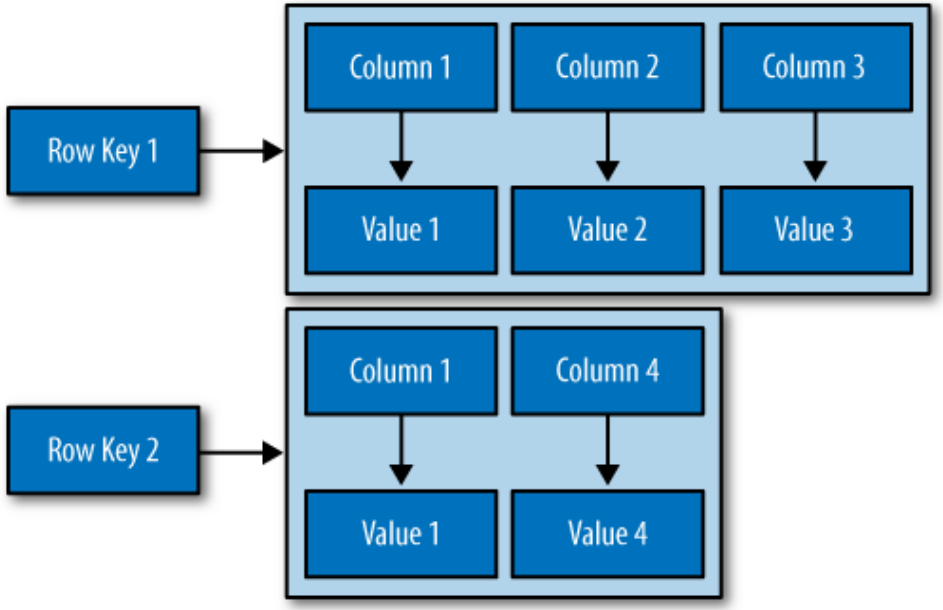
Características BBDD Clave-Valor

- Su precursor fue **Amazon Dynamo**
  - Basadas en DHT (Distributed Hash Tables)
- Modelo de datos muy sencillo: colección de pares clave/valor
- Ejemplos: Dynomite, Voldemort, Tokyo
- Buenas en:
  - manejo de datos no estructurados
  - alta disponibilidad de los datos
  - Las operaciones de lectura y escritura tienen un desempeño altísimo
- Nota: En general, no hay forma de recuperar un registro basándose en el contenido de su valor

	Key	Value
Image name →	image-12345.jpg	Binary image file
Web page URL →	http://www.example.com/my-web-page.html	HTML of a web page
File path name →	N:/folder/subfolder/myfile.pdf	PDF document
MD5 hash →	9e107d9d372bb6826bd81d3542a419d6	The quick brown fox jumps over the lazy dog
REST web service call →	view-person?person-id=12345&format=xml	<Person><id>12345</id .</Person>
SQL query →	SELECT PERSON FROM PEOPLE WHERE PID="12345"	<Person><id>12345</id .</Person>

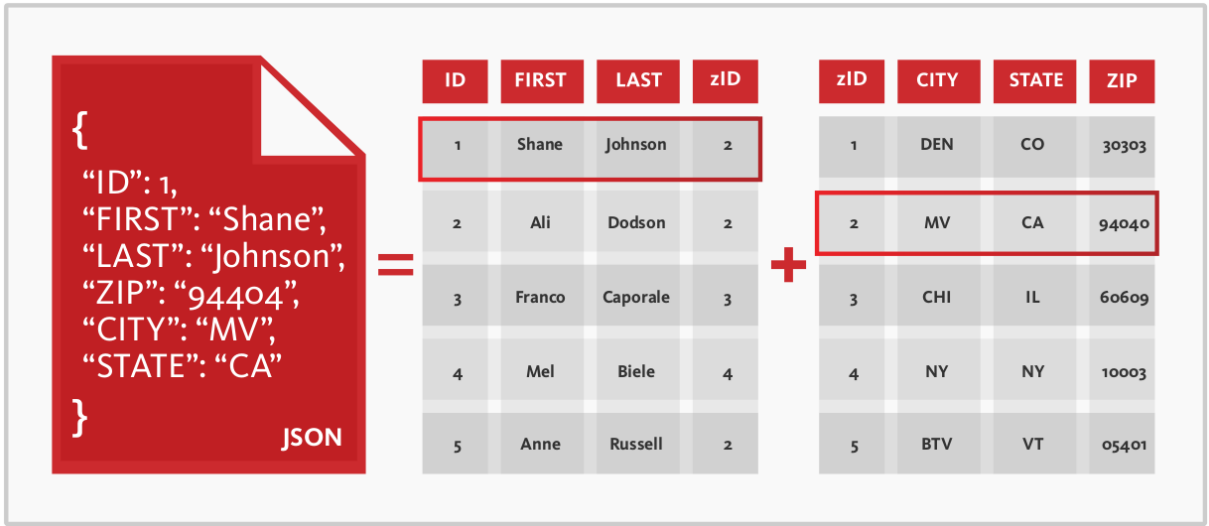
BBDD ORIENTADAS A COLUMNAS

- Su precursor fue **Google BigTable**
- Modelo de datos: familia de columnas, esto es, un modelo tabular donde cada fila puede tener una configuración diferente de columnas
  - Guardan datos por columna en vez de las BBDD tradicionales que lo hacen por filas
- Ejemplos: HBase, Hypertable, Cassandra, Riak
- Buenas en:
  - Gestión de tamaño
  - Cargas de escrituras masivas orientadas al stream
  - Alta disponibilidad
  - MapReduce
  - Optimizadas para operaciones a nivel de columnas (contar, sumar, promediar, etc.)



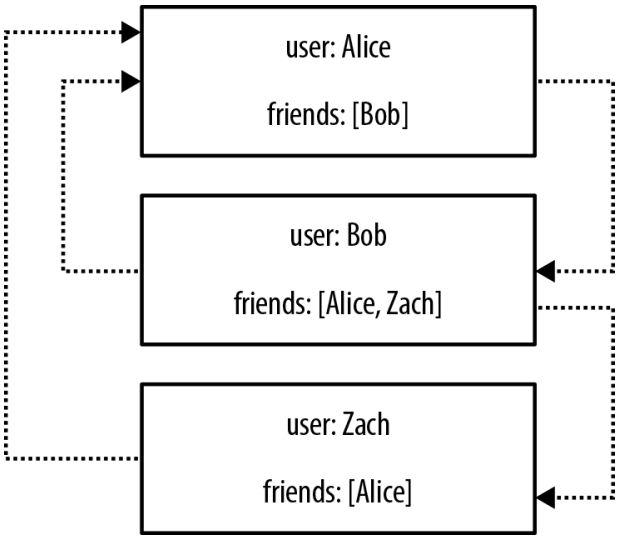
BBDD orientadas a Documentos

- La precursora fue **Lotus Notes**
- Modelo de datos: colecciones de documentos (JSON, XML, BSON) que contienen colecciones de claves-valor
- Ejemplos: CouchDB, **MongoDB**
- Buenas en:
  - Permiten trabajar con datos más complejos, admitiéndose documentos (objetos) anidados
  - Se corresponde con la manera en que se modelan los objetos y sus propiedades en los lenguajes OO
  - Orientas a la web: CRUD



BBDD orientadas a Grafos

- Inspiradas por Euler y la teoría de grafos
- Modelo de datos: nodos (interrelacionados unos con otros), relaciones con pares clave valor en ambos
- Ejemplos: AllegroGraph, VertexBD, **Neo4j**
- Buenas en:
  - Aquellos contextos en los que las relaciones son fundamentales (ej: redes sociales)
  - Aplican algoritmos de búsqueda optimizados para grafos



Person	
ID	Person
1	Alice
2	Bob
...	...
99	Zach

PersonFriend	
PersonID	FriendID
1	2
2	1
2	99
...	...
99	1

¿Dónde se usa NoSQL?

- Google (BigTable, LevelDB)
- LinkedIn (Voldemort)
- Facebook (Cassandra)
- Twitter (Hadoop/Hbase, FlockDB, Cassandra)
- Netflix (SimpleDB, Hadoop/HBase, Cassandra)
- CERN (CouchDB)