# NoSQL: la siguiente generación de bases de datos

18 Julio de 2013, 9:30-10:45

Curso de Verano Big Data & Data Science, Universidade de Santiago de Compostela

http://eventos.citius.usc.es/bigdata/

**Dr. Diego Lz. de Ipiña Glz. de Artaza**

DeustoTech-INTERNET, DeustoTech – Deusto Institute of Technology, Universidad de Deusto

dipina@deusto.es

http://paginaspersonales.deusto.es/dipina

http://www.morelab.deusto.es

# Abstract

El 95% de la información que se está generando actualmente en la red se corresponde con información no estructurada y las bases de datos relacionales tradicionales no son adecuadas para el manejo de esta información. Los nuevos sistemas de gestión de datos desarrollados para su manejo, que utilizan infraestructuras de supercomputación altamente distribuidas, reciben habitualmente el nombre de NoSQL. Este curso explorará algunos ejemplos significativos de las principales taxonomías de los almacenes de datos NoSQL: de clave-valor (Cassandra), orientados a documentos (CouchDB) o grafos (Neo4j). La finalidad del curso será identificar en qué situaciones una BBDD NoSQL puede reemplazar o complementar las BBDD relacionales, particularmente cuando se quieren gestionar ingentes cantidades de información semi-estructurada.

# Agenda

1. Introducción a NoSQL
2. Principios de las BBDD NOSQL: CAP y BASE
3. Taxonomía de las BBDD NoSQL
4. Apache Cassandra
   - Características
   - Instalación
   - Modelo de datos
   - Uso programático
5. CouchDB
6. MongoDB
7. Neo4j
8. Persistencia políglota

# Ejemplos de este Curso

- Los ejemplos asociados al curso, junto al software necesario para su ejecución, puede descargarse de:
  - https://www.dropbox.com/sh/yz51jkr9y6t3c51/9QSS0xvhlq/noSQL
- Los ejemplos asociados al curso pueden descargarse de:
  - http://paginaspersonales.deusto.es/dipina/resources/examples-nosql.rar

# Introducción a NoSQL

- **NoSQL** – "not only SQL" – es una categoría general de sistemas de gestión de bases de datos que difiere de los RDBMS en diferente modos:
  - No tienen schemas, no permiten JOINs, no intentan garantizar ACID y escalan horizontalmente
  - Tanto las bases de datos NoSQL como las relacionales son tipos de **Almacenamiento Estructurado**.
- 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

# Introducción a NoSQL

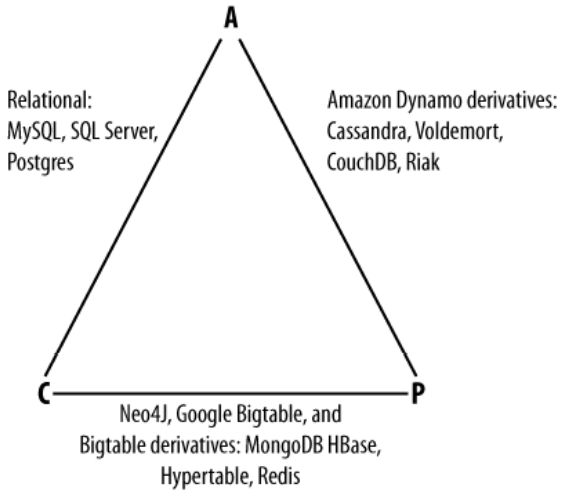
- La **principal diferencia radica en cómo guardan los datos** (por ejemplo, *almacenamiento de un recibo*):
  - En una RDBMS tendríamos que partir la información en diferentes tablas y luego usar un lenguaje de programación en la parte servidora para transformar estos datos en objetos de la vida real.
  - En NoSQL, simplemente guardas el recibo:
    - NoSQL es **libre de schemas**, tú no diseñas tus tablas y su estructura por adelantado
- **¡¡¡NoSQL no es la panacea!!!**
  - Si tus datos son relacionales, quedarte con tu RDBMS sería la opción correcta

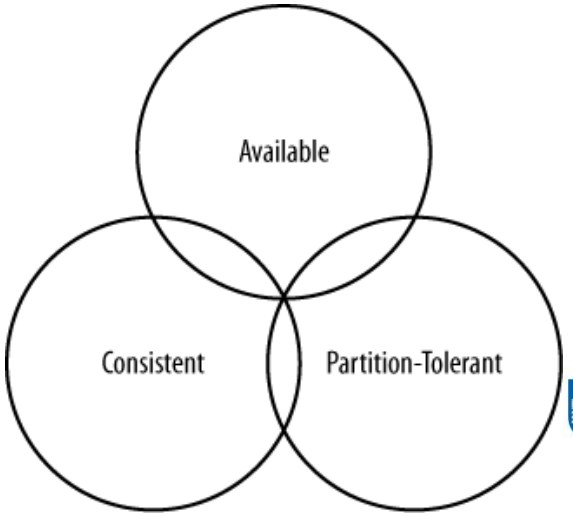
# Características principales

- Fáciles de usar en clústers de balanceo de carga convencionales → facilitan **escalabilidad horizontal**
- Guardan **datos persistentes** (no sólo cachés)
- **No tienen esquemas fijos y permite la migración del esquema** sin tener que ser reiniciadas o paradas
- Suelen tener un **sistema de consultas propio** en vez de usar un lenguaje de consultas estándar
- Tienen propiedades ACID en un nodo del clúster y son "**eventualmente consistentes**" en el clúster

# El teorema CAP

- Teorema de Brewer: “es imposible para un sistema computacional distribuido ofrecer simultáneamente las siguientes tres garantías”:
  - ***Consistencia*** – todos los nodos ven los mismos datos al mismo tiempo
  - ***Disponibilidad*** *(Availability)* – garantiza que cada petición recibe una respuesta acerca de si tuvo éxito o no
  - ***Tolerancia a la partición*** *(Partition)* – el sistema continua funcionando a pesar de la pérdida de mensajes
- Equivalente a:
  - **“You can have it good, you can have it fast, you can have it cheap: pick two.”**



# ACID vs. BASE

- En el mundo relacional estamos familiarizados con las transacciones ACID, que garantizar la consistencia y estabilidad de las operaciones pero requieren lockings sofisticados:
  - **ACID** = **A**tomicidad, **C**onsistencia, (**I**solation) aislamiento y **D**urabilidad
- 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-sate** – los almacenes no tienen porque ser consistentes ni sus réplicas en todo momento.
    - El programador puede verificar esa consistencia.
  - **Eventual consistency** – la consistencia se da eventualmente
- **BASE es una alternativa flexible a ACID** para aquellos almacenes de datos que no requieren un adherencia estricta a las transacciones

# 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 (e.j. sistemas bancarios):
  - Datos a escala web
  - Alta frecuencia de lecturas y escrituras
  - Cambios de esquema de datos frecuentes
  - Las aplicaciones sociales (no bancarias) no necesitan el mismo nivel de ACID
- Consecuencia → aparición de soluciones NoSQL
  - Cassandra, MongoDB, Jackrabbit , CouchDB, BigTable, Dynamo o Neo4j

# ¿Por qué necesitas NoSQL?

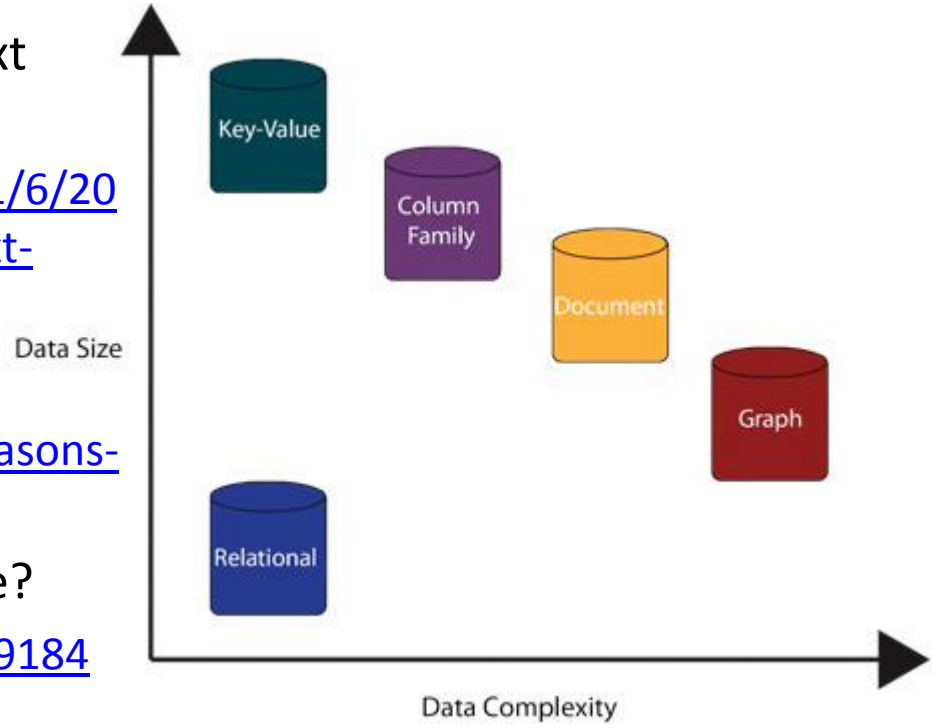
- Desafíos de gestión de información son difíciles de resolver con tecnología de bases de datos relacionales:
  - BBDD **no escala a tu tráfico** a un coste aceptable
  - El **tamaño de tu esquema de datos** ha crecido desproporcionalmente.
  - Generas **mucho datos temporales** que no corresponden al 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: "all people in a social network who have not purchased a book this year who are once removed from people who have“
  - Usas **transacciones locales** que no necesitan ser durables, e.j. Like.
    - Los sites AJAX tienen muchos casos de uso de este estilo.

# Arquitectura de las BBDD NoSQL

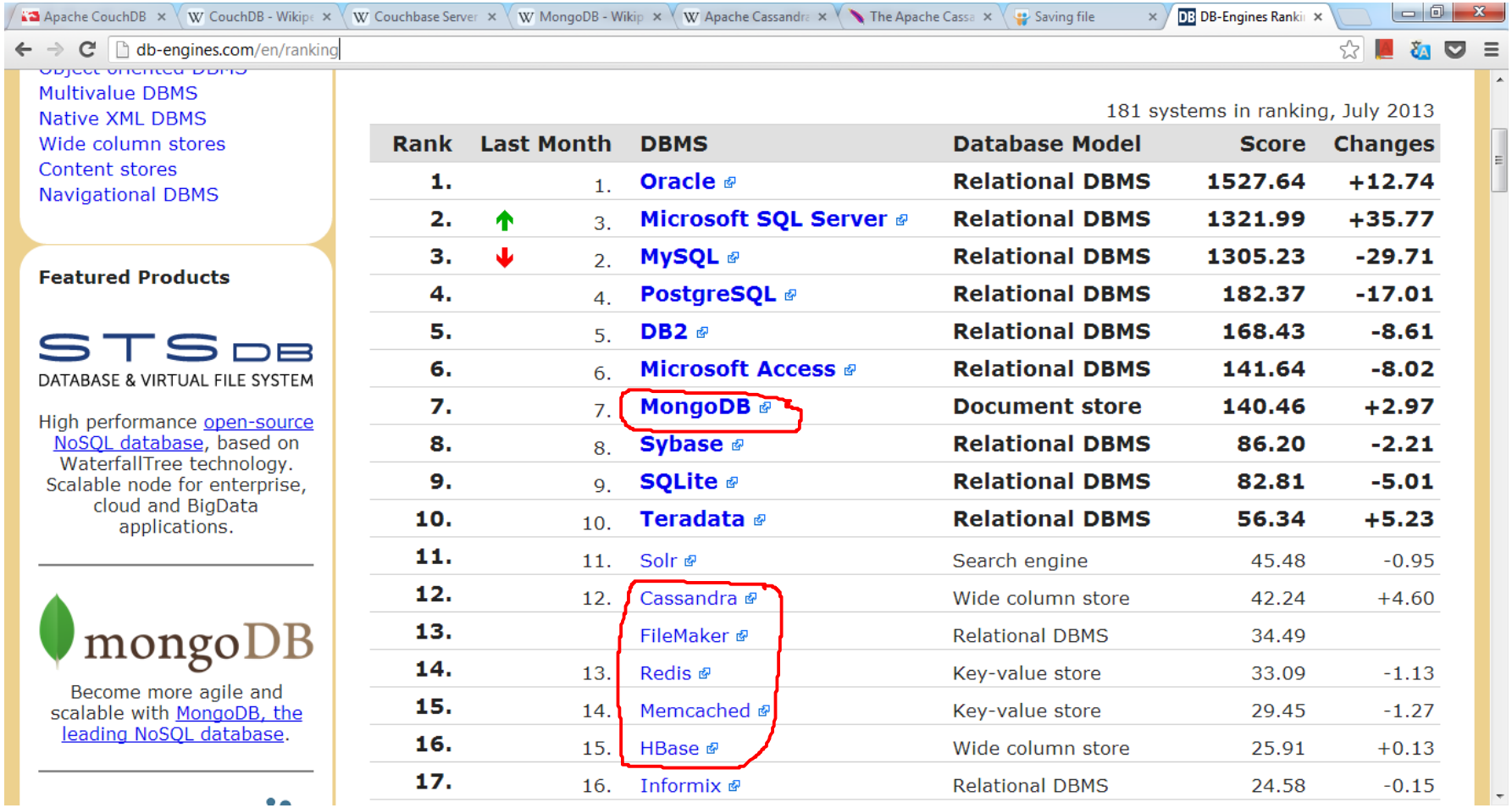
- A menudo ofrecen sólo **garantías de consistencia débiles**, como por ejemplo *eventual consistency*, o transacciones restringidas a elementos de datos simples
- Emplean una **arquitectura distribuida**, donde los datos se guardan de modo redundante en distintos servidores, a menudo usando tablas hash distribuidas
- Suelen ofrecer **estructuras de datos sencillas** como arrays asociativos o almacenes de pares clave-valor

# ¿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
  - Five Reasons to Use NoSQL
    - http://facility9.com/2010/09/five-reasons-to-use-nosql/
  - Which freaking database should I use?
    - http://www.infoworld.com/print/199184



# DB-Engines Ranking



db-engines.com/en/ranking

Multivalue DBMS
Native XML DBMS
Wide column stores
Content stores
Navigational DBMS

**Featured Products**

STSDB
DATABASE & VIRTUAL FILE SYSTEM

High performance open-source NoSQL database, based on WaterfallTree technology. Scalable node for enterprise, cloud and BigData applications.

mongoDB

Become more agile and scalable with MongoDB, the leading NoSQL database.

181 systems in ranking, July 2013

| Rank | Last Month | DBMS | Database Model | Score | Changes |
|---|---|---|---|---|---|
| 1. | 1. | Oracle | Relational DBMS | 1527.64 | +12.74 |
| 2. | 3. | Microsoft SQL Server | Relational DBMS | 1321.99 | +35.77 |
| 3. | 2. | MySQL | Relational DBMS | 1305.23 | -29.71 |
| 4. | 4. | PostgreSQL | Relational DBMS | 182.37 | -17.01 |
| 5. | 5. | DB2 | Relational DBMS | 168.43 | -8.61 |
| 6. | 6. | Microsoft Access | Relational DBMS | 141.64 | -8.02 |
| 7. | 7. | MongoDB | Document store | 140.46 | +2.97 |
| 8. | 8. | Sybase | Relational DBMS | 86.20 | -2.21 |
| 9. | 9. | SQLite | Relational DBMS | 82.81 | -5.01 |
| 10. | 10. | Teradata | Relational DBMS | 56.34 | +5.23 |
| 11. | 11. | Solr | Search engine | 45.48 | -0.95 |
| 12. | 12. | Cassandra | Wide column store | 42.24 | +4.60 |
| 13. | | FileMaker | Relational DBMS | 34.49 | |
| 14. | 13. | Redis | Key-value store | 33.09 | -1.13 |
| 15. | 14. | Memcached | Key-value store | 29.45 | -1.27 |
| 16. | 15. | HBase | Wide column store | 25.91 | +0.13 |
| 17. | 16. | Informix | Relational DBMS | 24.58 | -0.15 |

# Soluciones Empresariales NoSQL

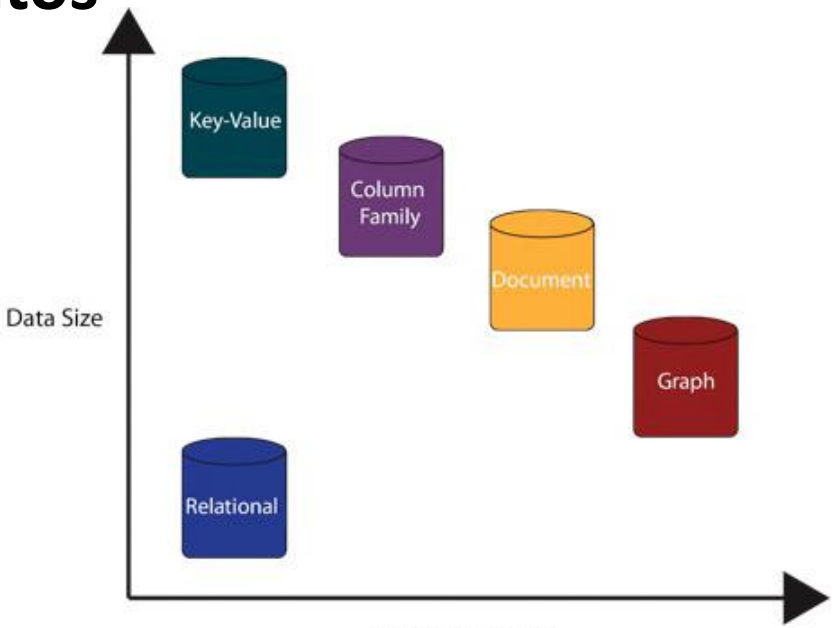
- **BigTable** (column oriented) es un sistema de gestión de base de datos creado por Google distribuido, de alta eficiencia y propietario.
  - Google Cloud Datastore (App Engine NoSQL Data Storage) motor de BBDD NoSQL factorizado de Google App Engine
    - BBDD de columnas que soporta transacciones ACID, tiene alta disponibilidad a través de centros de replicación y consultas SQL-like
- **Amazon DynamoDB** (row 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
- Comparativa:
  - http://www.theregister.co.uk/2013/05/16/google_datastore/
- Microsoft NoSQL **Windows Azure Tables**
  - http://msdn.microsoft.com/en-us/library/windowsazure/jj553018.aspx

# 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
- **Consistencia eventual**: si no se realizan nuevas actualizaciones a un elemento de datos, todos los accesos del elemento devolverán el último valor actualizado
- **Sharding**: es una partición horizontal en una BBDD donde las filas de una tabla se mantienen de modo separado
- **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.
- **Modelo de consistencia**: contrato entre el programador y el sistema sobre las garantías de consistencia (write & read consistency: one, all, quorum, etc.)

# Taxonomía de soluciones NoSQL

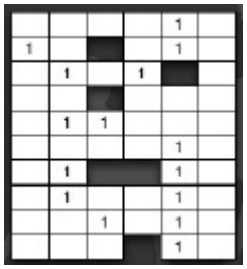
- Los principales tipos de BBDD de acuerdo con su implementación son los siguientes:
  - **Almacenes de Clave-Valor**
  - **Almacenes de Familia de Columnas**
  - **Almacenes de documentos**
  - **Grafos**



# Características BBDD orientadas a Clave-Valor

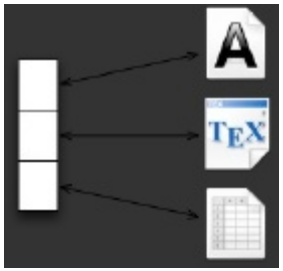
- Su precursor fue Amazon Dynamo
  - Basadas en DHT (Distributed Hash Tables)
- Modelo de datos: colección de pares clave/valor
- Ejemplos: Dynomite, Voldemort, Tokyo

# 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 orientas al stream
  - Alta disponibilidad
  - MapReduce

# BBDD orientadas a Documentos

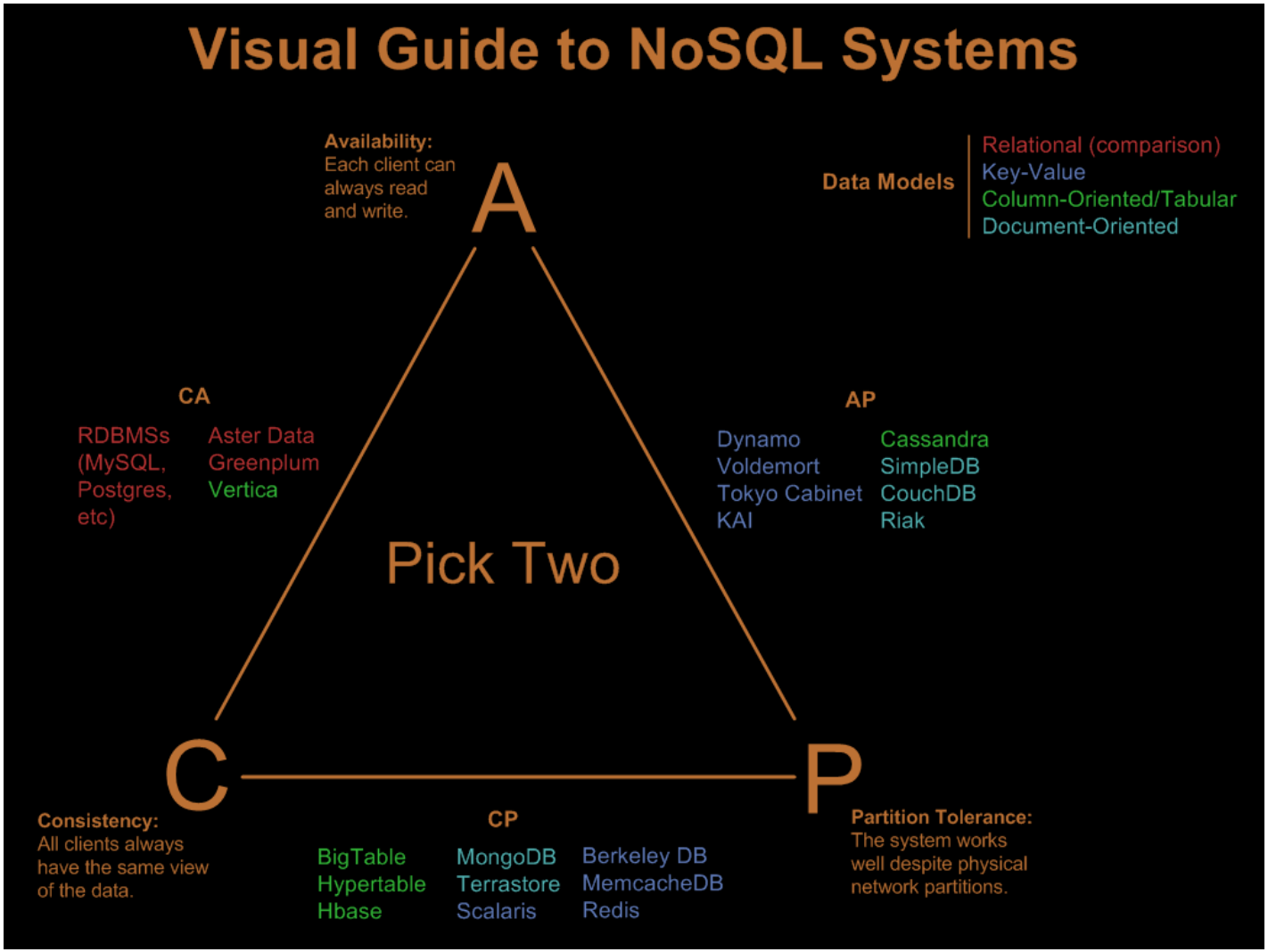


- La precursora fue Lotus Notes
- Modelo de datos: colecciones de documentos (JSON, XML, BSON) que contienen colecciones de claves-valor
- Ejemplos: CouchDB, MongoDB
- Buenas en:
  - Modelado de datos natural
  - Amigables al programador
  - Desarrollo rápido
  - Orientas a la web: CRUD

# BBDD orientadas a Grafos

- Inspiradas por Euler y la teoría de grafos
- Modelo de datos: nodos, relaciones con pares clave valor en ambos
- Ejemplos: AllegroGraph, VertexBD, Neo4j
- Buenas en:
  - Modelar directamente un dominio en forma de grafo, una manera común de representar y entender datasets
  - Ofrecer excelente rendimiento cuando los datos están interconectados y no tabulares
  - Realizar operaciones transaccionales que exploten las relaciones entre entidades

# El teorema de CAP



# Apache Cassandra

- Es un almacén altamente escalable, eventualmente consistente y distribuido de estructuras clave-valor.
  - Iniciado por Facebook
  - Código abierto
  - Proyecto apache
    - Licencia: Apache License 2.0
  - Escrito en Java
  - Multiplataforma
  - Versión actual: 1.2.6
  - Web: http://cassandra.apache.org/
- Documentación: http://www.datastax.com/documentation/cassandra/1.2/

# ¿Quién usa Apache Cassandra?

- Algunos usuarios famosos de Cassandra son:
  - Digg
  - Netflix
  - Rackspace
  - Twitter
  - …

# Ventajas de Cassandra para desarrolladores Web



- Cassandra está desarrollada para ser un **servidor distribuido**, pero puede también ejecutarse como un nodo simple:
  - Escalabilidad horizontal (añade nuevo hardware cuando sea preciso)
  - Rápidas respuestas aunque la demanda crezca
  - Elevadas velocidades de escritura para gestionar volúmenes de datos incrementales
  - Almacenamiento distribuido
  - Capacidad de cambiar la estructura de datos cuando los usuarios demandan más funcionalidad
  - Una API sencilla y limpia para tu lenguaje de programación favorito
  - Detección automática de fallos
  - No hay un punto de fallo único (cada nodo conoce de los otros)
  - Descentralizada
  - Tolerante a fallos
  - Permite el uso de Hadoop para implementar Map Reduce
  - Hinted hand off

# Desventajas de Cassandra

- Hay algunas desventajas que un sistema de almacenamiento tan escalable ofrece en contrapartida:
  - No hay joins (a cambio de más velocidad)
  - No permite ordenar resultados en tiempo de consulta
  - No tiene SQL
    - Pero desde la versión 0.8 tenemos CQL

# Instalación de Cassandra

- Documentación en:
  - Cassandra Wiki: GettingStarted, http://wiki.apache.org/cassandra/GettingStarted
- Requisitos:
  - Java 1.6 en adelante
- Las últimas versiones estables disponibles en:
  - http://cassandra.apache.org/download/

# Instalación de Cassandra

- Disponible desde: http://cassandra.apache.org/download/
  - Descargar `apache-cassandra-1.2.6-bin.tar.gz` o similar
- Descomprimir en tu sistema con Winrar o usando gzip en Linux
- Asociar a la variable de entorno PATH la localización de la carpeta `bin` dentro de Apache Cassandra:
  - En Windows podría quedar en: `C:\Programming\Java\apache-cassandra-1.2.6\bin`

# Ejecutando un nodo de Cassandra

- Arrancar Apache Cassandra, ejecutando:
  - `cassandra -f`
    - `-f` le dice a Cassandra que se ejecute en foreground para ver así los logs del sistema
- Tenemos un cluster con un solo nodo ejecutándose en el puerto 9160
- La configuración de este nodo la podemos encontrar en: `conf/cassandra.yaml`
- Ejecutar los clientes de consola para asegurarnos que todo está bien:
  - `cassandra-cli`
  - `cqlsh`

# Ring, clúster y el protocolo Gossip



- Cassandra usa un protocolo Gossip para permitir comunicación dentro de un ring, de tal modo que cada nodo sabe de otros nodos
  - Permite soportar descentralización y tolerancia a la partición
- Cassandra está diseñada para ser distribuida en varias máquinas que aparecen como una simple máquina a los ojos de los clientes
  - La estructura más externa de Cassandra es el cluster o ring
    - Un nodo tiene una réplica para diferentes rangos de datos, si algo va mal una réplica puede responder
      - El parámetro `replication_factor` en la creación de un KeySpace indica cuántas máquinas en el clúster recibirán copias de los mismos datos.

# Teorema CAP en Cassandra

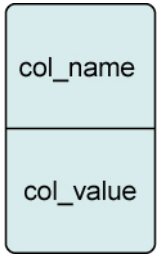
- Las bases de datos derivadas de Amazon Dynamo incluyen Cassandra, Voldemort, CouchDB y Riak
  - Centradas más en disponibilidad y tolerancia a fallos
    - Permiten Consistencia Eventual
      - Donde “eventual” significa milisegundos
  - Por tanto Cassandra es AP:
    - “To primarily support Availability and Partition Tolerance, your system may return inaccurate data, but the system will always be available, even in the face of network partitioning”

# Modelo de Datos en Cassandra

- Diseñado para datos distribuidos de modo escalable → sacrifica ACID por ventajas en rendimiento, disponibilidad y gestión operacional
- Los modelos que se crean son desnormalizados:
  - Se suele crear una column family por cada consulta (query) a realizar
  - Varias filas en un column family suelen dar respuesta a una consulta
- Los conceptos básicos son:
  - **Clúster**: son las máquinas que componen una instancia de Cassandra
    - Pueden contener varios Keyspaces
  - **Keyspace**: espacio de nombres para un conjunto de ColumFamily, asociado a una aplicación
    - Suele vincularse con una BBDD en el modelo relacional
  - **ColumFamily**: contienen varias columnas
    - Suelen vincularse con una tabla en el modelo relacional
  - **SuperColumn**: columnas que ellas mismas tienen sub-columnas
  - **Column**: compuestas de un nombre, valor y timestamp

# Column

- Una **columna** es un par nombre-valor que también contiene un timestamp
  - Los nombres y columnas son arrays de bytes
  - El timestamp registra la última vez que una columna es accedida
  - Unidad atómica
    - `name:value:timestamp`
    - `Email:dipina@deusto.es:123456789`




- Ejemplo en JSON:

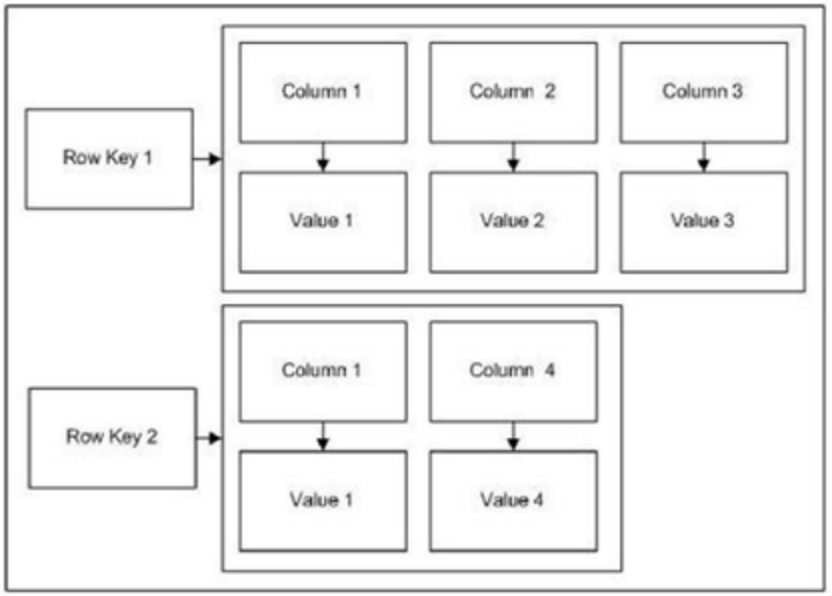
```
{
  "name": "Email",
  "value": "dipina@deusto.es",
  "timestamp”: 123456789
}
```

# Column Family

- Es un contenedor de columnas
  - Análogo al concepto de tabla en RDBMS
- Contiene una lista ordenada de columnas
  - Cuando se crea de manera configurativa una familia de columnas se indica cómo se ordenarán las columnas de cada fila (cogiendo el nombre)
    - ASCII, UTF-8, Long, UUID
- Cada familia de columnas se guarda en un fichero, y el fichero está ordenado por clave de fila
- Una familia de columnas tiene un …

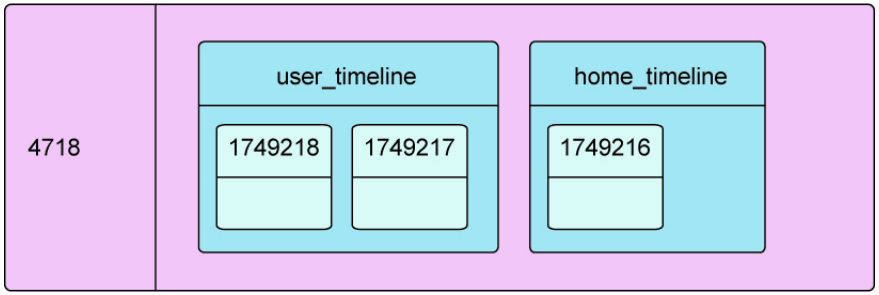
  conjunto de filas con un conjunto de …

  columnas similar pero no idéntico
- Pueden ser de tipo SUPER o STANDARD

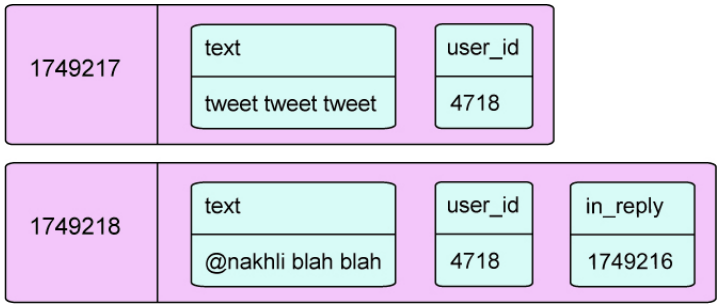


# KeySpaces

- Un espacio de claves o KeySpace es un esquema de alto nivel que contiene familias de columnas.
  - Supercolumn Family “Estado de Usuario”




  - Column Family “Entradas en Twitter: tweets”:



# Configuración de un Keyspace

- Los atributos básicos que puedes asociar a un keyspace son:
  - **Replication factor**: cuánto quieres pagar en rendimiento a favor de consistencia
  - **Replica placement strategy**: indica cómo se colocan las réplicas en el anillo: SimpleStrategy, OldNetworkTopologyStrategy y NetworkTopologyStrategy
    - Revisar: http://answers.oreilly.com/topic/2408-replica-placement-strategies-when-using-cassandra/
  - **Column families**: al menos una por Keyspace, es un contenedor de filas, que contienen columnas

# Particionado y Consistencia

- Cassandra ofrece soporte para particionado distribuido de datos
  - `RandomPartitioner` usa la función MD5 para distribuir filas en el cluster
  - `Murmur3Partitioner` es similar pero usa la función Murmur3_128#
  - `OrderPreservingPartitionioner` te permite ejecutar consultas de rangos, pero exige más trabajo eligiendo node tokens
    - Más info en: http://www.datastax.com/docs/1.2/cluster_architecture/partitioners
- Cassandra tiene **consistencia reconfigurable**
  - http://www.datastax.com/docs/1.2/dml/data_consistency

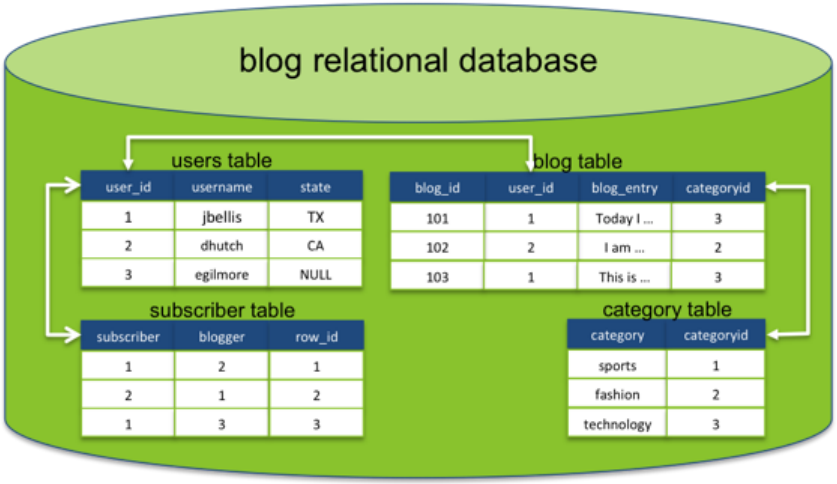
# Diferencias entre un RDBMS y Cassandra

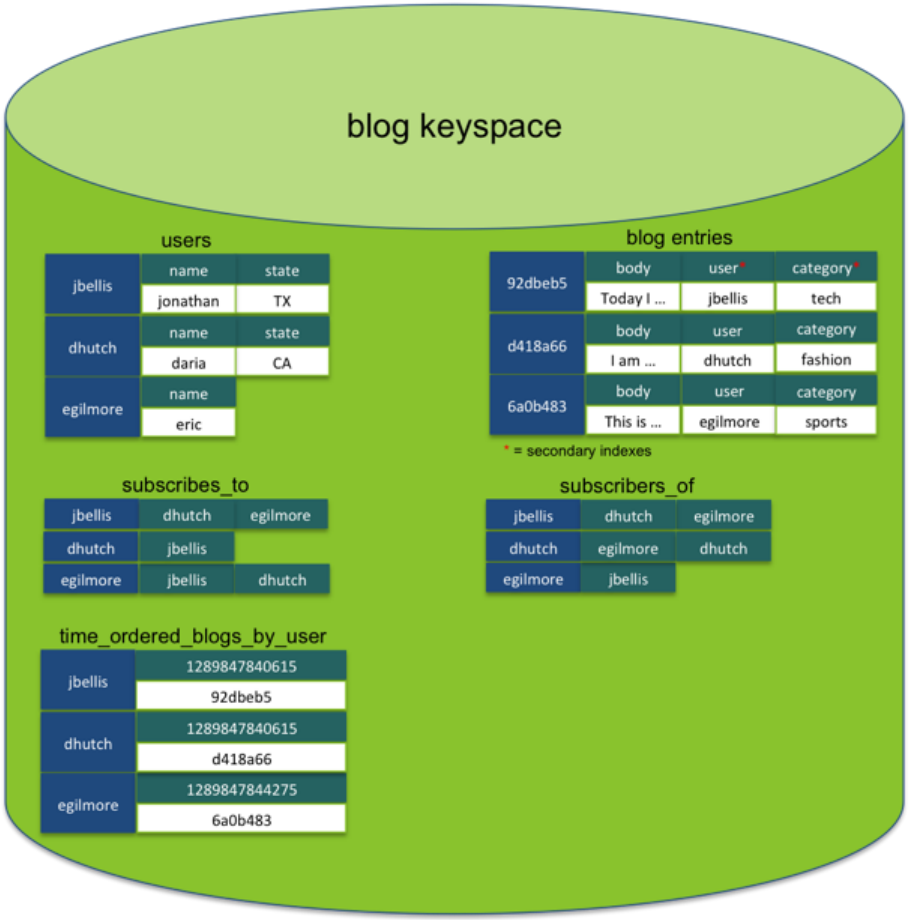


- No hay lenguaje de consulta, tiene una API accesible a través de Thrift
  - Verdad a medias: si existe desde la versión 0.8 CQL
- No hay integridad referencial, no hay joins, se puede emular almacenando claves a otras filas en un column family
- Índices secundarios en una tabla dan lugar a nuevos column families que mapean un campo a la clave de la primera column family
  - Verdad a medias: desde Cassandra 0.7 se pueden crear índices secundarios: http://www.datastax.com/dev/blog/whats-new-cassandra-07-secondary-indexes
- Las filas son agrupadas y ordenadas (distribuidas en el cluster) por el `Partitioner`
  - `RandomPartitioner`, `OrderPreservingPartitioner`, `CollatingOrderPreservingPartitioner`
- Cassandra tiene mejor rendimiento con datos desnormalizados
- Modelas las consultas y luego defines la estructura de datos a su alrededor
  - Cassandra tiene un modelo orientado a consultas
    - ¿qué preguntas tienes?

Cassandra

# RDBS vs. KeySpace



# Soporte Multi-lenguaje

- Cassandra utilizaba hasta ahora la librería Thrift (http://thrift.apache.org/) para proveer una API independiente del lenguaje de programación
  - Thrift soporta un gran número de lenguajes incluyendo: C++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, JavaScript, Node.js, Smalltalk, y Ocaml
  - Hay muchas librerías clientes disponibles:
    - Pycassa para Python: http://pycassa.github.com/pycassa/index.html
    - Hector para Java: https://github.com/rantav/hector
- Desde Cassandra 1.2 lo recomendable es hacer uso de Cassandra native protocol y CQL 3:
  - Datasax/java-driver: https://github.com/datastax/java-driver
  - Python driver: https://pypi.python.org/pypi/cql/1.0.3
  - Más drivers y librerías en: http://www.datastax.com/download/clientdrivers

# Acceso a CQL

- CQL puede usarse desde línea de comandos o programáticamente
  - Existen drivers en diferentes lenguajes de programación
    - http://crlog.info/2011/06/13/cql-creating-a-simple-keyspace/
- Desde Apache Cassandra 1.2.6, el cliente `cqlsh` está instalado en `$CASSANDRA_HOME/bin/cqlsh` para instalaciones tarball o `/usr/bin/cqlsh` para instalaciones de paquetes Linux
- Sintaxis:
  - http://cassandra.apache.org/doc/cql3/CQL.html

# Un poco de CQL

```
cqlsh> CREATE KEYSPACE test with REPLICATION = {'class' : 'SimpleStrategy', 'replication_factor': 1};
cqlsh> USE test;

cqlsh> CREATE COLUMNFAMILY users (
   ...     key varchar PRIMARY KEY,
   ...     full_name varchar,
   ...     birth_date int,
   ...     state varchar
   ... );

cqlsh> CREATE INDEX ON users (birth_date);
cqlsh> CREATE INDEX ON users (state);

cqlsh> INSERT INTO users (key, full_name, birth_date, state) VALUES ('bsanderson', 'Brandon Sanderson',
    1975, 'UT');
cqlsh> INSERT INTO users (key, full_name, birth_date, state) VALUES ('prothfuss', 'Patrick Rothfuss',
    1973, 'WI');
cqlsh> INSERT INTO users (key, full_name, birth_date, state) VALUES ('htayler', 'Howard Tayler', 1968,
    'UT');

cqlsh> SELECT key, state FROM users;
        key | state |
 bsanderson |    UT |
  prothfuss |    WI |
    htayler |    UT |

cqlsh> SELECT * FROM users WHERE state='UT' AND birth_date > 1970 ALLOW FILTERING;;
        KEY | birth_date |         full_name | state |
 bsanderson |       1975 | Brandon Sanderson |    UT |
```

# Twissandra-j



- Es un ejemplo en Java creado para aprender y demostrar el uso de Cassandra y las características de CQL 3.0
  - Presenta una web similar a Twitter
  - Está basado en el famoso ejemplo Twissandra hecho en Python y Django sobre Cassandra: https://github.com/twissandra/twissandra
- Para su ejecución:
  - Compilar el código: `mvn install`
  - Arrancar Cassandra: `cassandra -f`
  - Inicializar el Keyspace: `python cqlsh < twissandra-j/schema.cql`
  - Ejecutar el contenedor Karaf:
    - `bin/karaf`
    - `features:addurl mvn:org.opennms.twissandra/twissandra-cassandra/1.0-SNAPSHOT/xml/features`
    - `features:install twissandra-cassandra`
- Visitar la página web: `http://localhost:8181/twissandra`
- Los dos ficheros más importantes a revisar son:
  - `twissandra-j-master\persistence-cassandra\src\main\java\org\opennms\twissandra\persistence\cassandra\internal\`**`CassandraTweetRepository.java`**
    - Muestra cómo acceder a la base de datos
  - `twissandra-j-master/`**`schema.cql`**
    - CQL script que muestra cómo crear el modelo de datos

# Modelo de datos



```
-- User storage
CREATE TABLE users (username text PRIMARY KEY, password text);

-- Users user is following
CREATE TABLE following (
  username text,
  followed text,
  PRIMARY KEY(username, followed));

-- Users who follow user
CREATE TABLE followers (
  username  text,
  following text,
  PRIMARY KEY(username, following));

-- Tweet storage
CREATE TABLE tweets (
  tweetid uuid PRIMARY KEY,
  username text,
  body text);
```

# Modelo de datos



```
-- Materialized view of tweets created by user
CREATE TABLE userline (
  tweetid  timeuuid,
  username text,
  body     text,
  PRIMARY KEY(username, tweetid));

-- Materialized view of tweets created by user, and users she follows
CREATE TABLE timeline (
  username  text,
  tweetid   timeuuid,
  posted_by text,
  body      text,
  PRIMARY KEY(username, tweetid));
```

# Cassandra (1.2.6)

- **Written in:** Java
- **Main point:** Best of BigTable and Dynamo
- **License:** Apache
- **Protocol:** Thrift & custom binary CQL3
- Tunable trade-offs for distribution and replication (N, R, W)
- Querying by column, range of keys (Requires indices on anything that you want to search on)
- BigTable-like features: columns, column families
- Can be used as a distributed hash-table, with an "SQL-like" language, CQL (but no JOIN!)
- Data can have expiration (set on INSERT)
- Writes can be much faster than reads (when reads are disk-bound)
- Map/reduce possible with Apache Hadoop
- All nodes are similar, as opposed to Hadoop/HBase
- Cross-datacenter replication
- **Best used:** When you write more than you read (logging). If every component of the system must be in Java. ("No one gets fired for choosing Apache's stuff.")
- **For example:** Banking, financial industry (though not necessarily for financial transactions, but these industries are much bigger than that.) Writes are faster than reads, so one natural niche is data analysis.

# CouchDB

CouchDB relax

- CouchDB es una base de datos open source **orientada a documentos, accesible mediante una API RESTful** que hace uso extensivo de JavaScript Object Notation (**JSON**)
  - "**Couch**" es el acrónimo de"**Cluster Of Unreliable Commodity Hardware**"
    - Su misión es ser muy escalable, con alta disponibilidad y robustez, incluso cuando se ejecuta en hardware convencional
  - Creada como "**database of the Web**"

*"Django may be built for the Web, but CouchDB is built of the Web. I've never seen software that so completely embraces the philosophies behind HTTP. CouchDB makes Django look old-school in the same way that Django makes ASP look outdated."*

—Jacob Kaplan-Moss, Django developer

# Características de CouchDB

- **Documentos JSON** – todo lo que se guarda en CouchDB son simplemente documentos JSON.
- **Interfaz RESTful** – desde la creación a la replicación a la inserción de datos, toda la gestión de datos en CouchDB puede ser realizada vía HTTP.
- **Replicación N-Master** – puedes hacer uso de un número ilimitado de 'masters', dando lugar a topologías de replicación muy interesantes.
- **Escrita para ejecutarse offline** – CouchDB puede replicarse en dispositivos (e.j. teléfonos Android) que pueden quedarse sin conexión y gestionar sincronización de datos cuando el dispositivo está online de nuevo
- **Filtros de replicado** – puedes filtrar de modo preciso los datos que quieres replicar a distintos nodos.
  - http://wiki.apache.org/couchdb/Replication#Filtered_Replication

# ¿Quién usa CouchDB?

- Un largo listado de software y websites que hacen uso de CouchDB puede encontrarse en:
  - http://wiki.apache.org/couchdb/CouchDB_in_the_wild
  - Algunos ejemplos notables:
    - BBC
    - Credit Suisse

# Conceptos clave en CouchDB: Documentos



- Las BBDD CouchDB guardan **colecciones de documentos** nombrados de modo unívoco y proporcionan una API RESTful JSON que permite a las aplicaciones leer y modificar estos documentos
  - Cada documento puede tener campos no definidos en otros documentos:
    - Los documentos no están asociados a un esquema de bases de datos estricto
- Cada documento contiene metadatos (datos sobre datos) como el **identificador unívoco del documento** (`id`) y su **número de revisión** (`rev`)
  - Los campos de un documento pueden ser de varios tipos como strings, números, booleanos, colecciones, etc.
- Cuando se hacen **cambios sobre un documento CouchDB se crea una nueva versión del documento**, denominado revisión
  - Se mantiene un historial de modificaciones gestionado automáticamente por la BBDD
- CouchDB **no dispone de mecanismos de bloqueo (locking)** ante escrituras

# Conceptos clave en CouchDB: Vistas



- Son el mecanismo para añadir estructura a datos semi-estructurados
  - El modelo de vistas en CouchDB usa JavaScript aplicando el modelo MapReduce:
    - La **función Map** de la vista recibe un documento como argumento y realiza una serie de cálculos para determinar qué datos deberían ser disponibles en la vista
    - Si la vista tiene una **función Reduce**, es usada para agregar los resultados. A partir de un conjunto de pares clave/valor devuelve un sólo valor.
  - Las vistas son el método para agregar y realizar informes sobre los documentos de un repositorio, siendo creados en demanda para agregar y agregar documentos.

# Conceptos clave en CouchDB: Vistas



- CouchDB es desestructurado en naturaleza, adolece de un esquema estricto pero provee beneficios en términos de flexibilidad y escalabilidad:
  - Los datos se guardan en un espacio de almacenamiento plano, algo así como un repositorio de datos desnormalizados.
  - El modelo de vistas sirve para añadir estructura a los datos de modo que pueda agregarse para añadir significado útil
- Las **vistas se definen en documentos de diseño que pueden ser replicados a través de varias instancias**
  - Sólo se replican datos, aunque esos datos a menudo (código JavaScript) puede corresponder a aplicaciones.
  - Para garantizar un alto rendimiento, el **motor de vistas mantiene índices de sus vistas e incrementalmente las actualiza** para reflejar los cambios en la base de datos.

# Map/Reduce en CouchDB

- Usar Map/Reduce tiene ventajas sobre consultas SQL porque pueden ser distribuidas entre varios nodos, algo que no puede hacerse con RDBMS.
- Las bases de datos NoSQL utilizan map/reduce para consultar e indexar la BBDD
  - `map` consiste en extraer los datos a procesar
  - `reduce` se centra en la agregación de los mismos.

# Ejemplo de Vista en CouchDB



```
map: function(doc) {
  if (doc._attachments) {
    emit("with attachment", 1);
  }
  else {
    emit("without attachment", 1);
  }
}

reduce: function(keys, values) {
   return sum(values);
}
```

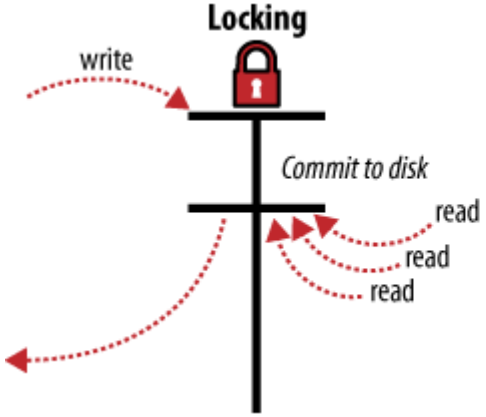
# Conceptos clave: Distribuida

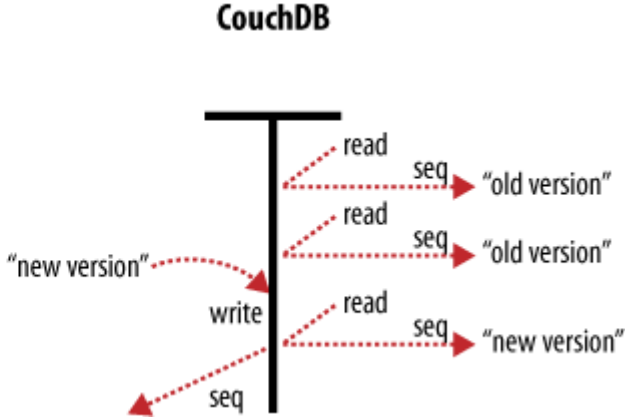
- CouchDB es un **sistema distribuido de base de datos** basado en nodos
  - Un número variable de nodos CouchDB (servidores y clientes offline) pueden tener "copias de réplicas" independientes de la misma BBDD, donde las aplicaciones pueden tener interactividad completa con la BBDD (consultar, añadir, editar y borrar)
    - Cuando vuelven a estar online o de modo planificado, los cambios de las bases de datos son replicados bidireccionalmente.
- CouchDB tiene **gestión de conflictos incorporada de serie**, haciendo que el proceso de replicación sea incremental y rápido, copiando sólo documentos y campos individuales modificados desde la última replicación.
  - Utiliza Multi-Version Concurrency Control (MVCC)

# Gestión de Conflictos MVCC



- Los documentos en CouchDB son versionados, de modo similar a como se realiza en sistemas de control de versiones como Subversion.
  - Si cambias un valor de un documento, realmente creas una nueva versión del mismo que coexiste con la versión antigua
- Las peticiones se ejecutan en paralelo haciendo que los servidores permitan una alta concurrencia
  - Una petición de lectura verá siempre la versión más reciente de la BBDD

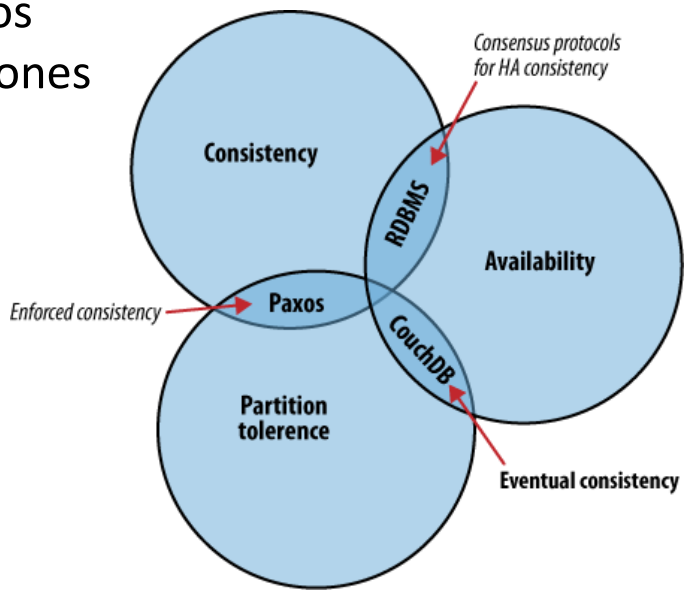


# Teorema CAP en CouchDB



- El teorema CAP dice que en BBDD distribuidas sólo dos propiedades pueden cumplirse:
  - *Consistency*
    - Todas las bases de datos cliente ven los mismos datos, incluso cuando se producen actualizaciones concurrentes.
  - *Availability*
    - Todos los clientes pueden acceder a los datos.
  - *Partition tolerance*
    - La base de datos puede partirse a múltiples servidores
- **CouchDB sacrifica consistencia inmediata a cambio de obtener un mayor rendimiento a través de distribución de la información**



# Instalación y ejecución

- Ir a http://couchdb.apache.org/, seleccionar "Download" y elegir la plataforma de instalación
  - En Windows el último instalador es: `setup-couchdb-1.3.1_R15B03-1.exe`
  - Instalación en Linux:
    - Tan simple como: `sudo aptitude install couchdb`
    - http://wiki.apache.org/couchdb/Installing_on_Ubuntu
- Para lanzar el servidor de CouchDB ejecutar el comando `couchdb` en el directorio `bin` de instalación.

# CouchDB Futon

- Administrador gráfico de bases de datos en CouchDB
  - `http://localhost:5984/_utils/`
- Hace uso internamente de la librería http://127.0.0.1:5984/_utils/script/jquery.couch.js
  - http://daleharvey.github.io/jquery.couch.js-docs/symbols/
- Lo primero que hay que hacer es hacer click en Fix Me para asegurarnos que sólo usuarios autorizados pueden acceder a CouchDB
  - Nosotros usaremos la combinación `admin/citius`

# La API RESTful JSON



- CouchDB ofrece una **API REST como mecanismo para manipular datos de una BBDD**:
  - **POST** – crea un nuevo registro
  - **GET** – lee registros
  - **PUT** – actualiza un registro
  - **DELETE** – borra un registro
- Esta API accesible vía HTTP GET y POST retorna datos en JSON:
  - Una ventaja de este enfoque es que puede usarse una framework AJAX como Prototype o jQuery para crear una aplicación web, sin necesidad de hacer uso de un lenguaje de parte servidora
  - La herramienta de línea de comando CURL pueden ser usada como cliente de línea de comandos HTTP:
    - Descargable de: http://curl.haxx.se/
    - Permite realizar peticiones GET, POST, PUT, y DELETE, mostrando la respuesta HTTP recibida del servidor web

# Probando la API RESTful de CouchDB con curl

- `$ curl http://127.0.0.1:5984/`
  - Respuesta: `{"CouchDB":"Welcome","version":"1.1.1"}`
- O explícitamente define el tipo de petición realizada a través del parámetro `-X` de curl:
  - `$ curl -X GET http://127.0.0.1:5984/_all_dbs`
    - Respuesta:
      `["_replicator","_users","miscursos","test_suite_reports","testdb","users"]`
- Para crear dos nuevas bases de datos, ejecutaríamos los comandos:
  - `$ curl -uadmin:citius -X PUT http://127.0.0.1:5984/fruit`
    - Respuesta: `{"ok":true}`
  - `$ curl -uadmin:citius -X PUT http://127.0.0.1:5984/vegetables`
- Si ejecutamos ahora:
  - `$ curl -X GET http://127.0.0.1:5984/_all_dbs`
  - Obtendríamos:
    `["_replicator","_users","fruit","miscursos","test_suite_reports","testdb","users"]`
- Si intentamos volver a crear una BBDD ya existente, recibimos un error:
  - `$ curl -X PUT http://127.0.0.1:5984/fruit`
    - Respuesta: `{"error":"file_exists","reason":`
      `"The database could not be created, the file already exists."}`

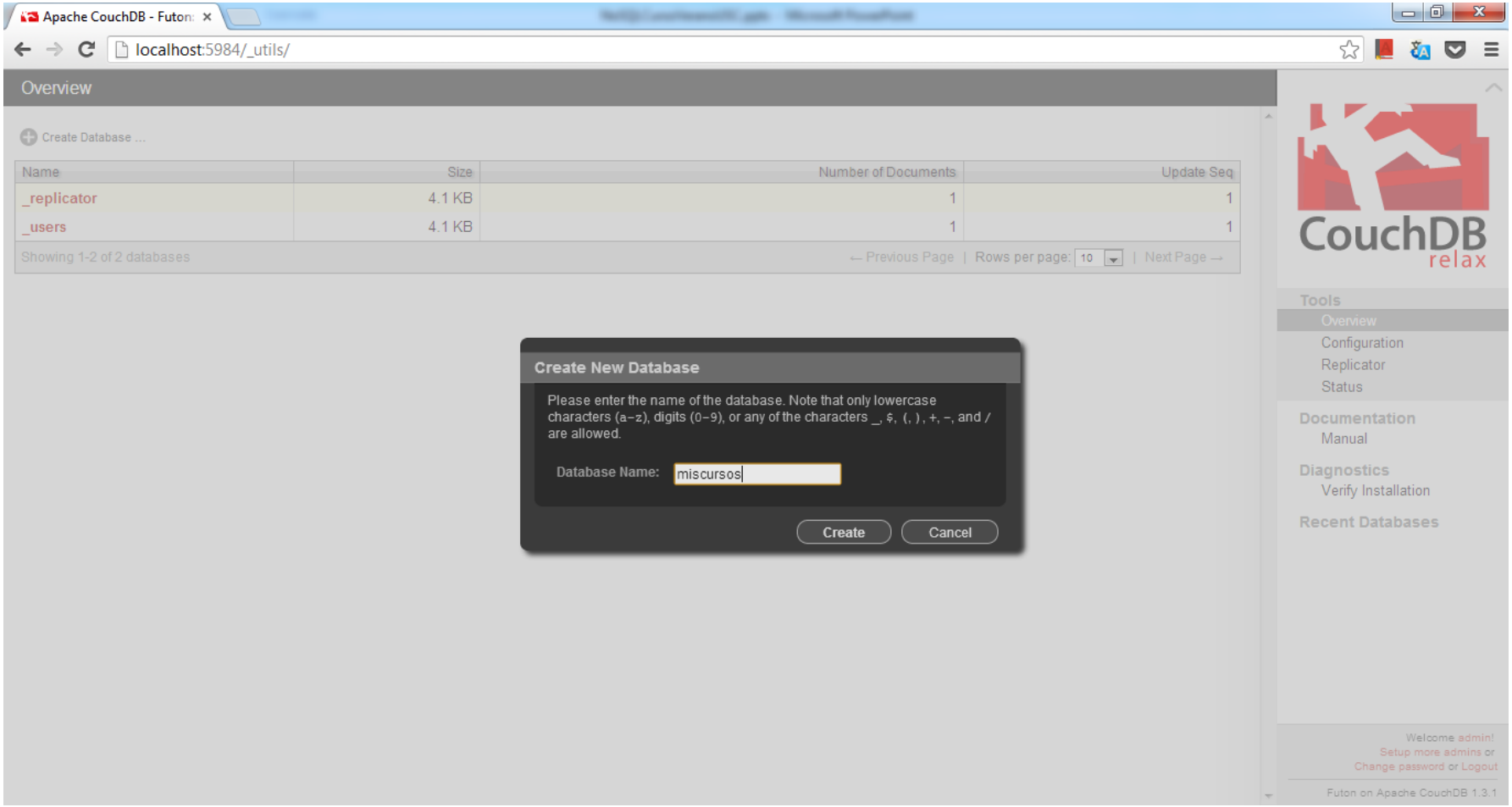
# Probando la API RESTful de CouchDB con curl

- Podemos borrar una base de datos con el siguiente comando:
  - `$ curl -uadmin:citius -X DELETE http://127.0.0.1:5984/vegetables`
  - Respuesta: `{"ok":true}`
- Para crear un documento:
  - `$ curl -uadmin:citius -X PUT http://127.0.0.1:5984/fruit/apple -H "Content-Type: application/json" -d {}`
  - Respuesta: `{"ok":true,"id":"apple","rev":"1-967a00dff5e02add41819138abb3284d"}`
- Para recuperarlo:
  - `$ curl -X GET http://127.0.0.1:5984/fruit/apple`
  - Respuesta: `{"_id":"apple","_rev":"1-967a00dff5e02add41819138abb3284d"}`
- Para recuperar información de la BBDD:
  - `$ curl -X GET http://127.0.0.1:5984/fruit`
  - Respuesta:
    `{"db_name":"fruit","doc_count":1,"doc_del_count":0,"update_seq":1,"purge_seq":0,"compact_running":false,"disk_size":4179,"instance_start_time":"1321991208171560","disk_format_version":5,"committed_update_seq":1}`

# Programando CouchDB

- Gracias a la API RESTful, los desarrolladores pueden conectarse a CouchDB usando cualquier software que soporte HTTP
- La mayoría de los lenguajes modernos ofrecen algún tipo de interfaz HTTP, implicando que CouchDB puede ser usada en cualquier proyecto de desarrollo.
  - Revisar la siguiente página para diferentes clientes programáticos a CouchDB:
    - https://code.google.com/p/couchdb-python/
    - http://wiki.apache.org/couchdb/Getting_started_with_Java

# Primeros pasos en CouchDB



# Primeros Pasos en CouchDB

- En Futon:
  1. Crearemos la base de datos haciendo click en “Create Database”, de nombre "`miscursos`"
  2. Hacer click en “`New Document`”
     - Vete añadiendo campos a través de “`ADD Field`”
       - `"name" : “Curso NoSQL”`
       - `"price" : 30`
     - Guarda cada campo
     - Guarda el documento
  3. Modificar el documento ya existente añadiendo el campo "`type`" con valor "`course`"
     - La nueva versión del documento debería empezar por 2

# Primeros Pasos en Curl

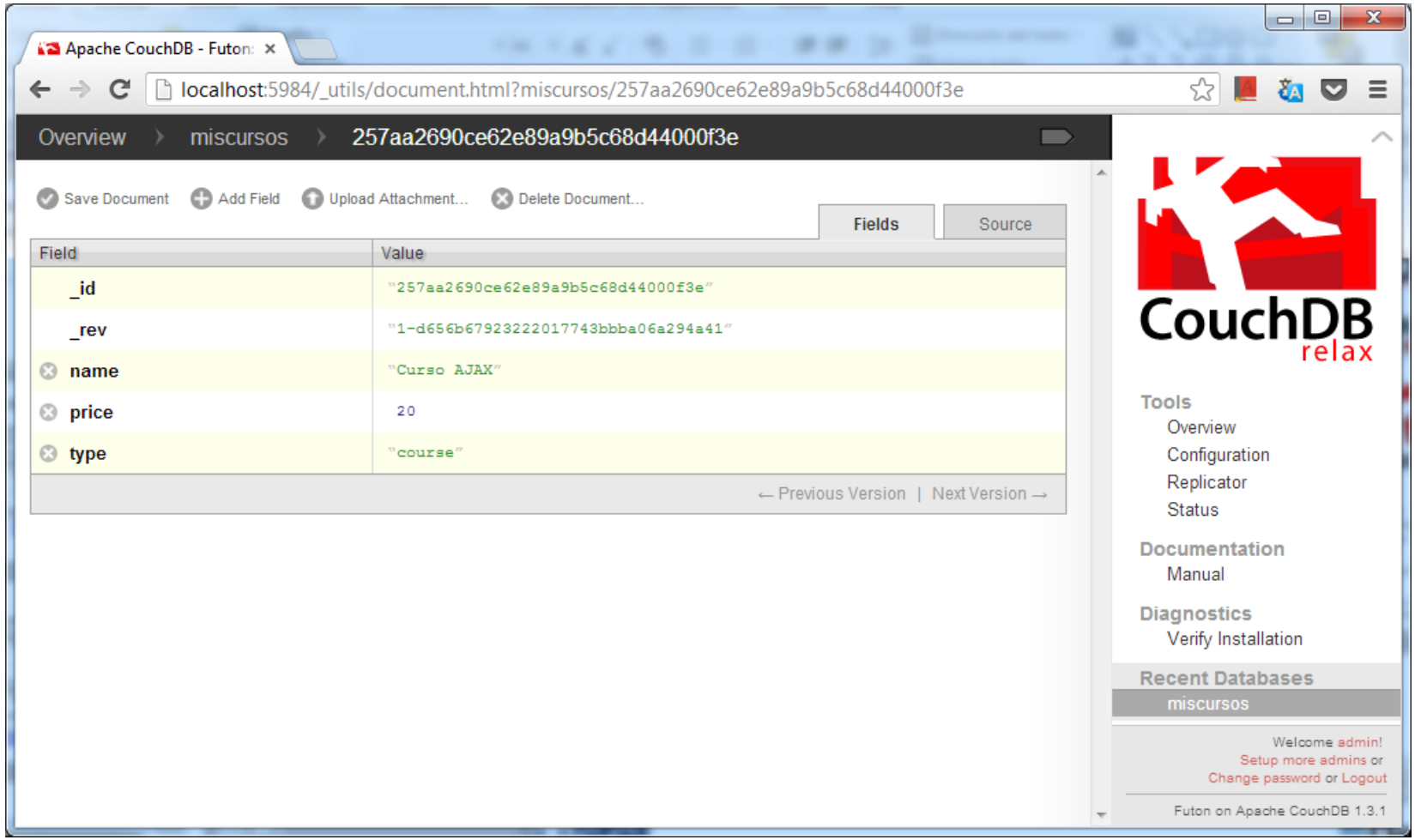


1. Creemos un documento `persona.json` con el siguiente contenido:

```
{
  "forename": "Diego",
  "surname":  "Lopez-de-Ipina",
  "type":     "person"
}
```

2. Usamos el siguiente comando de CURL para subir el documento a CouchDB:
   - `curl -X POST http://127.0.0.1:5984/miscursos/ -d @persona.json -H "Content-Type: application/json"`
   - Se devolvería:
     - `{"ok":true,"id":"b54b3496d090c43a4266180ecb002a92","rev":"1-93c73d298af443f623db6861f90e9f6e"}`
3. Para recuperar todos los documentos:
   - `curl -X GET http://127.0.0.1:5984/miscursos/_all_docs`

# Vista de un documento en Futon



# Creando una función Map

- Seleccionar Temporary View en the Drop View dentro de Futon
  - La función map tendría el siguiente código:

```
function (doc) {
  if (doc.type == "course" && doc.name) {
    emit(doc.name, doc);
  }
}
```

  - Accede a ella a través del navegador como:
    - http://127.0.0.1:5984/_utils/database.html?miscursos/_design/courses/_view/courses

# Creando un Reduce



- Nos aseguraremos de tener al menos dos cursos con precio
- Definir la función de mapeo como:

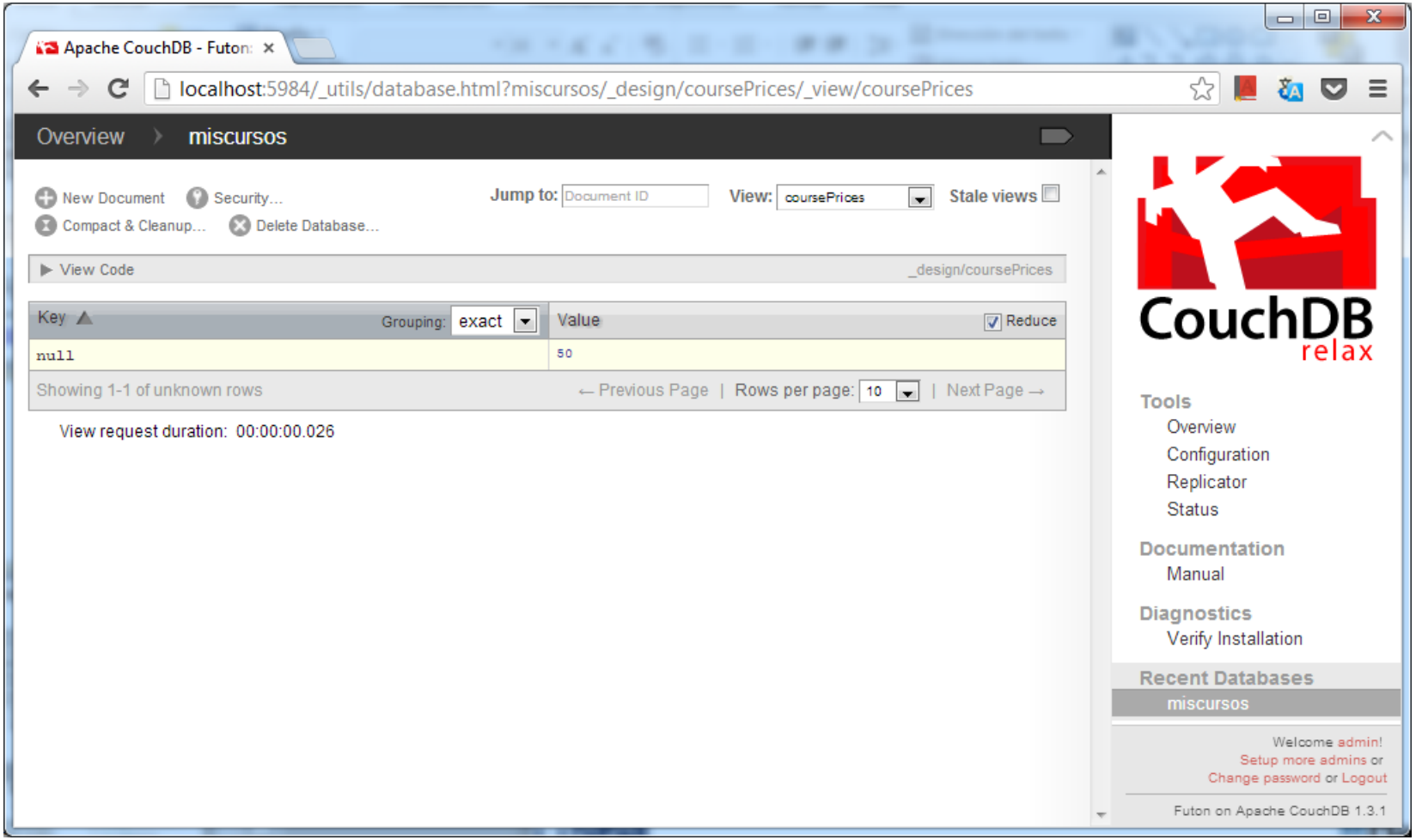
```
function (doc) {
    if (doc.type === "course" && doc.price) {
        emit(doc.id, doc.price);
    }
}
```

- Definir la función reduce como:

```
function (keys, prices) {
    return sum(prices);
}
```

- Guardarlo como `coursePrices`, asegurarse de hacer click en `Reduce`
- Ir a http://127.0.0.1:5984/_utils/database.html?miscursos/_design/courseprices/_view/courseprices

# Creando un Reduce



# Documentos de Diseño en CloudDB



- Los documentos de diseño son un tipo especial de documento en CouchDB que contiene código de aplicación:
  - Vistas MapReduce, validaciones, funciones `show`, `list` y `update`
  - Se suele crear un documento de diseño por cada aplicación
- El **documento de diseño es un documento CouchDB con un ID que comienza con `_design/`:**
  - Se replica como otros documentos en la BBDD y soporta gestión de conflictos a través del parámetro `rev`
    - CouchDB mira las vistas y otras funciones de aplicación en él
    - Los contenidos estáticos de la aplicación aparecen como `_attachments` en el documento de diseño

# Estructura interna de un Documento de Diseño

- Están compuestos de:
  - Funciones de validación
  - Definición de vistas
  - Funciones `show`, `list` y `update`
  - Attachments
- Una BBDD CouchDB puede tener varios documentos de diseño.
  - `_design/calendar`
  - `_design/contacts`
- Para recuperar un documento de diseño haz un GET con el siguiente patrón de URL:
  - `http://localhost:5984/mydb/_design/calendar`
  - `http://127.0.0.1:5984/mydb/_design/contacts`

# Vistas: funciones map



- Las funciones vistas en CouchDB son strings guardados en el campo `views` del documento de diseño
- Los resultados de un vista se guardan en un B-tree, al igual que todo documento
  - Estos árboles permiten realizar búsquedas de filas por clave y recuperar rangos de filas
- Todas las funciones `map` tienen un único parámetro `doc`:

```
function(doc) {
   if(doc.date && doc.title) {
     emit(doc.date, doc.title);
   }
}
```

- Las funciones `map` son funciones libres de efectos laterales que toman un documento como argumento y emiten pares clave/valor:
  - Generan una lista ordenada por la clave de las filas
- Tenemos varios parámetros para recuperar resultados:
  - Una sola fila: `_design/docs/_view/date?key="2009/01/30 18:04:11"`
  - Varias filas: `_design/docs/_view/date?startkey="2010/01/01 00:00:00"&endkey="2010/02/00 00:00:00"&descending=true`

# Vistas: funciones `reduce`



- Las funciones `reduce` operan sobre las filas ordenadas emitidas por las funciones `map`
- Dada la manera en que los árboles B-tree están estructurados, podemos cachear los resultados intermedios de un reduce en los nodos no-hoja
- El formato de una función `reduce` en una vista es el siguiente:

```
function(keys, values, rereduce) {
  return sum(values)
}
```

- La función `reduce` se ejecuta sobre cada nodo del árbol para calcular el resultado final que es un valor escalar:
  - Cuando se ejecuta sobre las hojas del árbol (que contiene filas del mapa), el parámetro `rereduce` es `false`
  - Cuando se ejecuta en los nodos internos del árbol, el valor de `rereduce` es `true`

# Configuración de vistas

- Se definen en el documento de diseño como sigue:

```
{
  "_id": "_design/application",
  "_rev": "1-C1687D17",
  "views": {
    "viewname": {
      "map": "function(doc) { ... }",
      "reduce": "function(keys, values) { ... }"
    },
    "anotherview": {
      "map": "function(doc) { ... }",
      "reduce": "function(keys, values) { ... }"
    }
  }
}
```

- Para acceder a ellas se realiza un HTTP GET a `/database/_design/application/_view/viewname`

# Recetas: De SQL a MapReduce

- `SELECT field FROM table WHERE value="searchterm"`
  - La función map sería:

```
function(doc) {
    if(doc.age && doc.name) {
        emit(doc.age, doc.name);
    }
}
```

  - Y la consulta:

```
/ladies/_design/ladies/_view/age?key=5
```

- Documentación: http://guide.CouchDB.org/draft/cookbook.html

# Validaciones



- CouchDB usa la función `validate_doc_update` para evitar que documentos inválidos o no autorizados sean procesados.
- CouchDB envía las funciones y documentos a un intérprete de JavaScript
- La función `validate_doc_update` se ejecuta para cada documento a crear o actualizar
  - Si lanza una excepción, la actualización es rechazada
- El formato de una función de validación es el siguiente:

```
function(newDoc, oldDoc, userCtx) {
   function require(field, message) {
    message = message || "Document must have a " + field;
    if (!newDoc[field]) throw({forbidden : message});
   };
   if (newDoc.type == "post") {
    require("title");
    require("created_at");
    require("body");
    require("author");
   }
}
```

# Replicación en CouchDB



- **La replicación sincroniza dos copias de la misma BBDD**, permitiendo que los usuarios tengan baja latencia de acceso a datos independientemente de su localización
  - La replicación consiste en enviar una petición HTTP a un servidor CouchDB incluyendo una BBDD de origen y otra de destino, haciendo que CouchDB envíe los cambios del origen al destino:
    - `POST /_replicate HTTP/1.1 {"source":"database","target":http://example.org/database}`

# Replicación en CouchDB

- Para enviar cambios se sigue utilizando la misma llamada
  - Cuando solicitas a CouchDB que replique una BBDD en otra, comparará ambas para encontrar qué documentos en el origen difieren de los del destino y luego enviará todos los cambios al destino en batch
    - Las bases de datos en CouchDB tienen un número de secuencia que es incrementado cada vez que la BBDD se cambia
- Hay diferentes modos de replicación:
  - Replicación normal, en demanda
  - Replicación continua, cada vez que produce un cambio éste se propaga
    - Añades `"continuous":true` al objeto de lanzamiento de replicación
    - CouchDB hará uso de la `_changes` API sobre cambios en documentos

# Replicación con Futon

- Abre tu navegador en Futon: http://127.0.0.1:5984/_utils/
  - Haz clic en `Replicator`
  - Selecciona BBDD a replicar
  - Haz clic en `continuous`
  - Haz clic en el botón `Replicate`
- Alternativamente desde línea de comando:
  - `curl -uadmin:citius -X DELETE http://127.0.0.1:5984/miscursos_rep2`
  - `curl -uadmin:citius -X PUT http://127.0.0.1:5984/miscursos_rep2`
  - `curl -uadmin:citius -X POST http://127.0.0.1:5984/_replicate -H "Content-Type: application/json" -d @replication-example.json`
- Donde `replication-example.json` contiene:
  - `'{"source":"http://127.0.0.1:5984/miscursos", "target":http://admin:citius@127.0.0.1:5984/miscursos_rep2, "continuous":true}'`

# Gestión de Conflictos

- Cuando replicas dos BBDD en CouchDB y se identifican conflictos, CouchDB los señala en los documentos afectados con el atributo `"_conflicts":true`
  - La versión que es seleccionada como última es la *versión ganadora.*
  - La *revision perdedora* es almacenada como versión anterior.
  - CouchDB no hace un merge de las revisiones conflictivas.
    - Tu aplicación es la que resuelve programáticamente los conflictos
  - La replicación garantiza que los conflictos se detecten y que cada instancia de CouchDB haga la misma selección respecto a ganadores y perdedores, independientemente de las instancias que existan.

# Notificaciones de cambios

- CouchDB ofrece un mecanismo de subscripciones a cambios mediante la API `_changes`
- La API `_changes` suministra información en el siguiente formato:
  - `{"seq":12,"id":"foo","changes":[{"rev":"1-23202479633c2b380f79507a776743d5"}]}`
    - Donde `seq` es un valor que va cambiando cada vez que se produce un cambio en una BBDD
    - `id` es el identificador del documento
    - El campos `changes` suele contener una descripción de las últimas modificaciones

# Probando la API _changes

- Creamos una BBDD:
  - `$ curl -X PUT http://127.0.0.1:5984/db-changes`
- Revisamos si ha habido cambios:
  - `$ curl -X GET http://127.0.0.1:5984/db-changes /_changes`
    - Devuelve:
      - `{"results":[], "last_seq":0}`
- Añadimos un documento:
  - `$ curl -X PUT http://127.0.0.1:5984/db-changes/test -d '{"name":"Anna"}'`
- Revisamos si ha habido cambios de nuevo:
  - `{"results":[{"seq":1,"id":"test","changes":[{"rev":"1-aaa8e2a031bca334f50b48b6682fb486"}]}], "last_seq":1}`
- Podemos usar el parámetro `since` para filtrar resultados y `longpoll` para quedarse bloqueado hasta un nuevo cambio:
  - `$ curl -X GET $HOST/db/_changes?feed=longpoll&since=1`

# Probando la API _changes

- Long polling requiere abrir una nueva conexión HTTP por cada cambio notificado.
- La API *continuous changes* permite establecer una conexión permanente entre tu aplicación y CouchDB para recibir cambios:
  - `$ curl -X GET "$HOST/db/_changes?feed=continuous&since=3"`
- Podemos definir filtros para refinar los cambios en los que estamos interesados. Por ejemplo:

```
{
    "_id": "_design/app",
    "_rev": "1-b20db05077a51944afd11dcb3a6f18f1",
    "filters": {
       "important":
         "function(doc, req) {
              if(doc.priority == 'high') { return true; }
              else { return false; }
         }"
    }
}
```

- Para ver los cambios filtrados invocaríamos:

```
$ curl "$HOST/db/_changes?filter=app/important"
```

# CouchApp

- A CouchApp es una framework para la creación de aplicaciones JavaScript con CouchDB
  - Como son aplicaciones JavaScript y HTML5 son servidas directamente desde CouchDB
  - Como efecto lateral consigues la flexibilidad y escalabilidad de CouchDB
    - http://couchapp.org/
- Para crear una CouchApp, necesitas un modo para meter JavaScript, HTML y otros recursos en CouchDB
  - La herramienta CouchApp escrita en Python es la recomendada para generar plantillas de código en tu aplicación y volcarlos a una instancia de CouchDB:
    - http://couchapp.org/page/couchapp-python

# CouchApp

- CouchApp te simplifica la labor de crear aplicaciones web a partir de CouchDB:
  - Fragmenta un documento de diseño en un conjunto de directorios y ficheros del sistema local, representando views, shows, validations, attachments, etc. que serán enviados al servidor CouchDB
- Una aplicación de Couch, CouchApp, sigue más o menos el patrón de diseño Modelo/Vista/Controlador:
  - CouchDB es el Modelo
  - El árbol DOM representando el HTML mostrado al usuario sería la vista
  - El JavaScript sería el Controller

# Ejemplos de CouchApp

- Revisar documentación en:
  - http://couchapp.org/page/index
    - Incluye varios ejemplos de aplicaciones
- Instalación de aplicación de ejemplo AddressBook:

```
couchapp push addressbook
  http://admin:citius@localhost:5984/addre
  ssbook
```

  - Ir a: http://localhost:5984/addressbook/_design/addressbook/index.html

# CouchDB (V1.3.1)

- **Written in:** Erlang
- **Main point:** DB consistency, ease of use
- **License:** Apache
- **Protocol:** HTTP/REST
- Bi-directional (!) replication, continuous or ad-hoc, with conflict detection, thus, master-master replication. (!)
- MVCC - write operations do not block reads
- Previous versions of documents are available
- Crash-only (reliable) design
- Needs compacting from time to time
- Views: embedded map/reduce
- Formatting views: lists & shows
- Server-side document validation possible
- Authentication possible
- Real-time updates via '`_changes`' (!)
- Attachment handling
- thus, CouchApps (standalone js apps)
- **Best used:** For accumulating, occasionally changing data, on which pre-defined queries are to be run. Places where versioning is important.
- **For example:** CRM, CMS systems. Master-master replication is an especially interesting feature, allowing easy multi-site deployments.

# MongoDB

- Similar a CouchDB
- Pretende combinar lo mejor de los almacenes clave/valor, bases de datos de documentos y RDBMS
- Hace uso de JSON y tiene su propio lenguaje de consultas
- Implementada en C++
- Usada por SourceForge, Bit.ly, Foursquare o GitHub
- URL: http://www.mongodb.org/

# MongoDB

- **MongoDB** (de la palabra en ingles "hu**mongo**us" que significa enorme) es un sistema de base de datos NoSQL orientado a documentos
- MongoDB **guarda estructuras de datos en documentos tipo BSON** (*Binary JSON* (JSON Binario) con un esquema dinámico , haciendo que la integración de los datos en ciertas aplicaciones sea mas fácil y rápida.

# Características Principales



- **Consultas Ad hoc**
  - MongoDB soporta la búsqueda por campos, consultas de rangos y expresiones regulares.
  - Las consultas pueden devolver un campo específico del documento pero también puede ser una función JavaScript definida por el usuario.
- **Indexación**
  - Cualquier campo en un documento de MongoDB puede ser indexado, al igual que es posible hacer índices secundarios.
    - El concepto de índices en MongoDB es similar a los encontrados en base de datos relacionales.
- **Replicación**
  - MongoDB soporta el tipo de **replicación maestro-esclavo.**
    - El maestro puede ejecutar comandos de lectura y escritura.
    - El esclavo puede copiar los datos del maestro y sólo se puede usar para lectura
      - El esclavo tiene la habilidad de poder elegir un nuevo maestro en caso del que se caiga el servicio con el maestro actual.

# Características Principales



- **Balanceo de carga**
  - MongoDB se puede escalar de forma horizontal usando el concepto de "shard".
    - El desarrollador elije una clave shard, la cual determina como serán distribuidos los datos en una colección. Los datos son divididos en rangos (basado en la clave shard) y distribuidos a través de múltiples shard.
    - Un shard es un maestro con uno o más esclavos.
  - MongoDB tiene la capacidad de ejecutarse en múltiple servidores, balanceando la carga y/o duplicando los datos
- **Almacenamiento de archivos**
  - MongoDB puede ser utilizado con un sistema de archivos, tomando la ventaja de la capacidad que tiene MongoDB para el balanceo de carga y la replicación de datos utilizando múltiples servidores para almacenamiento
  - GridFS está incluida en los drivers de MongoDB y disponible para los lenguajes de programación que soporta MongoDB.
- **Agregación**
  - La función MapReduce puede ser utilizada para el procesamiento por lotes de datos y operaciones de agregación.
    - Permite que los usuarios puedan obtener el tipo de resultado que se obtiene cuando se utiliza el comando SQL "group-by".
- **Ejecución de JavaScript del lado del servidor**

# Casos de uso de MongoDB

- Almacenamiento y registro de eventos
- Para sistemas de manejo de documentos y contenido
- Comercio Electrónico
- Juegos
- Problemas de alto volumen
- Aplicaciones móviles
- Almacén de datos operacional de una página Web
- Manejo de contenido
- Almacenamiento de comentarios
  - Votaciones
  - Registro de usarios
  - Perfiles de usuarios
  - Sesiones de datos
- Proyectos que utilizan metodologías de desarrollo iterativo o ágiles
- Manejo de estadísticas en tiempo real

# Manipulación de Datos: colecciones y documentos

- MongoDB guarda la estructura de los datos en documentos tipo JSON con un esquema dinámico llamado BSON, lo que implica que no existe un esquema predefinido.
- Los elementos de los datos son llamados **documentos** y se guardan en **colecciones**
- Una colección puede tener un número indeterminado de documentos
  - Las **colecciones son como tablas** y los **documentos como filas**
  - Cada documento en una colección puede tener diferentes campos.
- La estructura de un documento es simple y compuesta por "key-value pairs" parecido a las matrices asociativas en un lenguaje de programación
  - Como valor se pueden usar números, cadenas o datos binarios como imágenes o cualquier otro "key-value pairs".

# Ejemplo de documento en MongoDB



```
{
    "_id": ObjectId("4efa8d2b7d284dad101e4bc7"),
    "Last Name": "PELLERIN",
    "First Name": "Franck",
    "Age": 29,
    "Address": {
        "Street": "1 chemin des Loges",
        "City": "VERSAILLES"
    }
}
```

# Utilidades de MongoDB



- Las siguientes herramientas pueden ser utilizadas para el manejo y la administración de MongoDB:
  - **`mongo`**: Shell interactivo que permite a los desarrolladores:
    - Ver, insertar, eliminar y actualizar datos en su base de datos.
    - Replicar información, configurar los Shards, apagar los servidores y ejecutar JavaScript.
  - **`mongostat`**: herramienta de línea de comandos que muestra en resumen una lista de estadísticas de una instancia de MongoDB en ejecución.
  - **`mongotop`**: herramienta de línea de comandos para dar seguimiento a la cantidad de tiempo que dura una la lectura o escritura de datos en una instancia.
  - **`mongosniff`**: herramienta de línea de comandos que hace sniffing en el tráfico de la red que va desde y hacia MongoDB.
  - **`mongoimport/mongoexport`**: herramienta de línea de comandos que facilita la importación exportación de contenido desde JSON, CSV o TSV.
  - **`mongodump/mongorestore`:** herramienta de línea de comandos para la creación de una exportación binaria del contenido de la base de datos.

# Documentación e instalación

- La documentación completa de MongoDB puede encontrarse en:
  - http://docs.mongodb.org/manual/
- Instrucciones para instalar MongoDB en Windows:
  - Descargar la última versión de: http://docs.mongodb.org/manual/tutorial/install-mongodb-on-windows/
  - Crear directorio de datos: `mkdir data\db`
  - Ejecutar el comando: **`mongod.exe --dbpath ..\data\db`**
  - Ejecutar el cliente `mongo` y los siguientes comandos en JavaScript:

```
%INSTALL_DIR%\mongodb-win32-x86_64-2.4.5\bin>mongo
MongoDB shell version: 2.4.5
connecting to: test
> db.test.save( {a:1} )
> db.test.find()
{ "_id" : ObjectId("4fe6e41b184d3a26629be9b6"), "a" : 1 }
>
```

# Usando MongoDB



- `mongo` es un shell JavaScript completo, cualquier función JavaScript, sintáxis o clase puede usarse en el shell

```
> j = { name : "mongo" };
{ "name" : "mongo" }
> t = { x : 3 };
{ "x" : 3 }
> db.things.save(j);
> db.things.save(t);
> db.things.find();
{ "_id" : ObjectId("51e50d3b70f9b7c7fdbd8d90"), "name" : "mongo" }
{ "_id" : ObjectId("51e50d3b70f9b7c7fdbd8d91"), "x" : 3 }
> for (var i = 1; i <= 20; i++) db.things.save({x : 4, j : i});
> db.things.find();
{ "_id" : ObjectId("51e50d3b70f9b7c7fdbd8d90"), "name" : "mongo" }
{ "_id" : ObjectId("51e50d3b70f9b7c7fdbd8d91"), "x" : 3 }
...
Type "it" for more
> // Iterate through the remaining items
> it
{ "_id" : ObjectId("51e50d3b70f9b7c7fdbd8da4"), "x" : 4, "j" : 19 }
{ "_id" : ObjectId("51e50d3b70f9b7c7fdbd8da5"), "x" : 4, "j" : 20 }
```

# Usando MongoDB

```
> // Store the cursor of the DB in a variable
> var cursor = db.things.find();
> while (cursor.hasNext()) printjson(cursor.next());
{ "_id" : ObjectId("51e50d3b70f9b7c7fdbd8d90"), "name" : "mongo" }
{ "_id" : ObjectId("51e50d3b70f9b7c7fdbd8d91"), "x" : 3 }
...
> // Use functional features of JavaScript
> db.things.find().forEach(printjson);
{ "_id" : ObjectId("51e50d3b70f9b7c7fdbd8d90"), "name" : "mongo" }
{ "_id" : ObjectId("51e50d3b70f9b7c7fdbd8d91"), "x" : 3 }
...
> // cursors like an array
> var cursor = db.things.find();
> printjson(cursor[4]);
{ "_id" : ObjectId("51e50d3b70f9b7c7fdbd8d94"), "x" : 4, "j" : 3 }
>
```

# Documentos de consulta

- Documentos que indican el patrón de claves y valores que deben ser localizados (proyección)
- Ejemplos:
  - `SELECT * FROM things WHERE name="mongo"`
    - **`db.things.find({name:"mongo"}).forEach(printjson);`**
  - `SELECT * FROM things WHERE x=4`
    - **`db.things.find({x:4}).forEach(printjson);`**
  - `SELECT j FROM things WHERE x=4`
    - **`db.things.find({x:4}, {j:true}).forEach(printjson);`**
  - Recuperar el primer elemento que cumple alguna restricción:
    - **`printjson(db.things.findOne({name:"mongo"}));`**
  - Limitar el número de resultados:
    - **`db.things.find().limit(3);`**

# GridFS

- GridFS es una especificación para guardar ficheros grandes en MongoDB
  - Aunque MongoDB permite guardar datos binarios en objetos BSON, su límite en tamaño es de 16 MB
- Ofrece un mecanismo para dividir transparentemente un fichero grande en varios documentos
  - Cada fichero tiene un objeto de metadatos en la colección `files` y uno o más objetos chunk en la colección `chunks`
- URL: http://www.mongodb.org/display/DOCS/GridFS

# Programando MongoDB

- MongoDB tiene soporte variado de lenguajes y librerías cliente:
  - http://docs.mongodb.org/ecosystem/drivers/java/
- Para programar en Java mirar:
  - http://docs.mongodb.org/ecosystem/drivers/java/
  - http://docs.mongodb.org/ecosystem/tutorial/getting-started-with-java-driver/
- Algunos buenos ejemplos en:
  - http://www.mkyong.com/tutorials/java-mongodb-tutorials/

# Comparación MongoDB y CouchDB

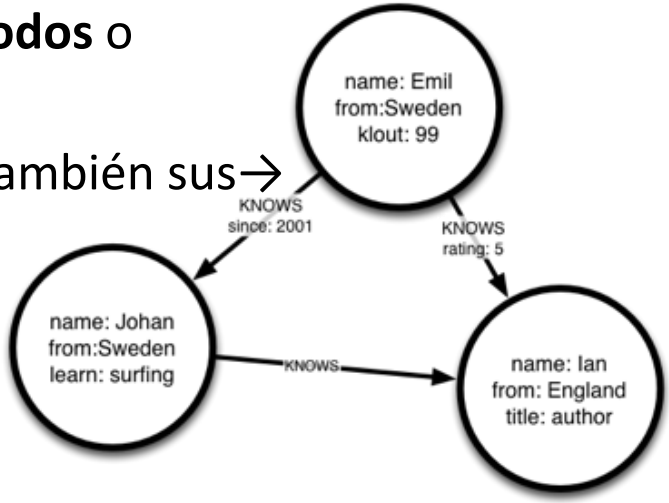
| | Couchdb | Mongodb |
|---|---|---|
| Data Model | Document-Oriented (JSON) | Document-Oriented (BSON) |
| Interface | HTTP/REST | Custom protocol over TCP/IP |
| Object Storage | Database contains Documents | Database contains Collections Collections contains Documents |
| Query Method | Map/Reduce (javascript + others) creating Views + Range queries | Map/Reduce (javascript) creating Collections + Object-Based query language |
| Replication | Master-Master with custom conflict resolution functions | Master-Slave |
| Concurrency | MVCC (Multi Version Concurrency Control) | Update in-place |
| Written In | Erlang | C++ |

# MongoDB (2.4.5)

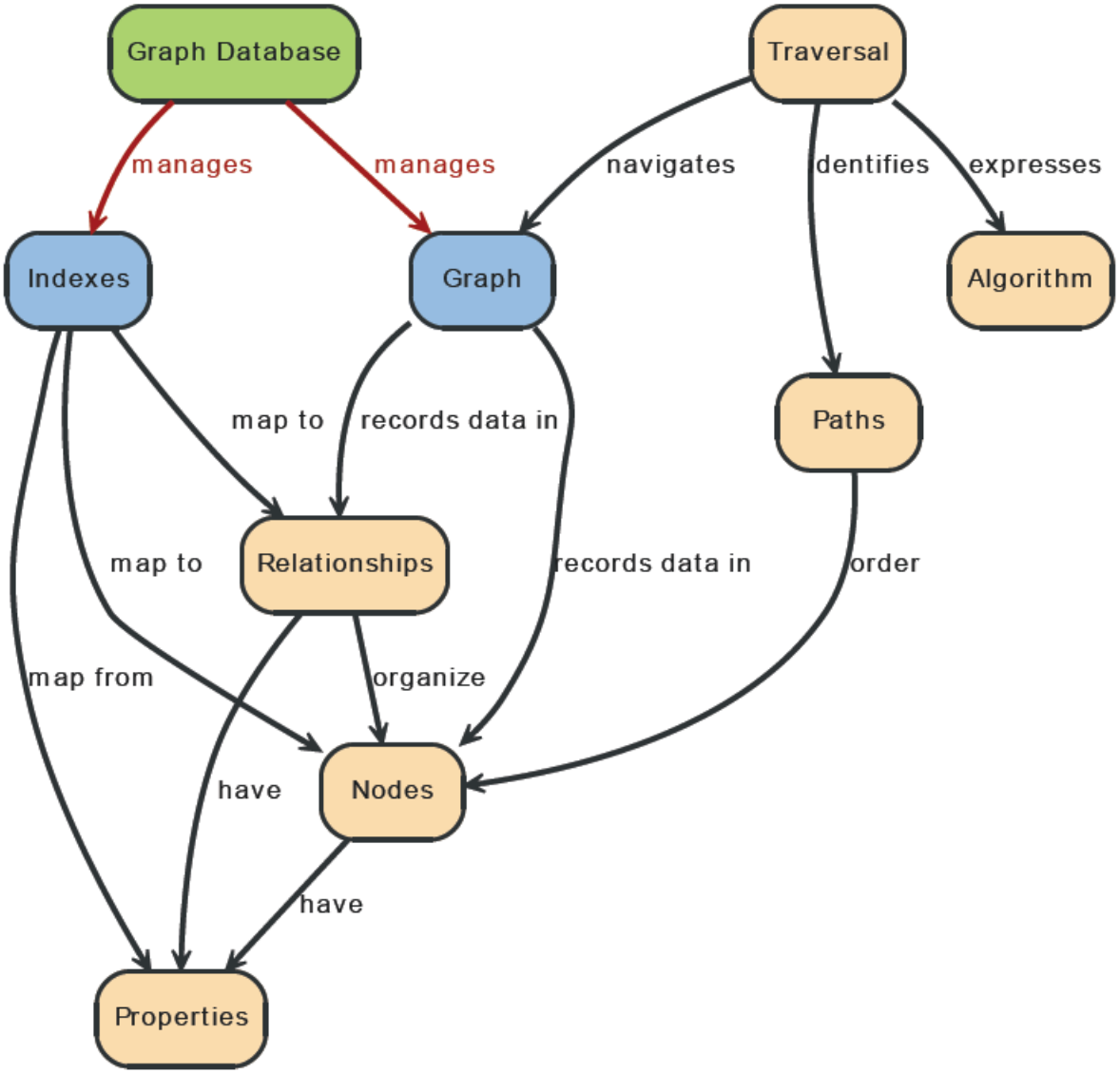
- **Written in:** C++
- **Main point:** Retains some friendly properties of SQL. (Query, index)
- **License:** AGPL (Drivers: Apache)
- **Protocol:** Custom, binary (BSON)
- Master/slave replication (auto failover with replica sets)
- Sharding built-in
- Queries are javascript expressions
- Run arbitrary javascript functions server-side
- Better update-in-place than CouchDB
- Uses memory mapped files for data storage
- Performance over features
- Journaling (with --journal) is best turned on
- On 32bit systems, limited to ~2.5Gb
- An empty database takes up 192Mb
- GridFS to store big data + metadata (not actually an FS)
- Has geospatial indexing
- Data center aware
- **Best used:** If you need dynamic queries. If you prefer to define indexes, not map/reduce functions. If you need good performance on a big DB. If you wanted CouchDB, but your data changes too much, filling up disks.
- **For example:** For most things that you would do with MySQL or PostgreSQL, but having predefined columns really holds you back.

# Neo4j: una BBDD orientada a grafos

- Una BBDD de grafos almacena grafos, la estructura de datos más genérica
  - Un **Grafo** —almacena datos en→ **Nodos** —que tienen→ **Propiedades**
  - Los **Nodos** —están organizados en→ **Relaciones** —que también tienen→ **Propiedades**
  - Un **Recorrido** (traversal) —navega→ un **Grafo**; ella—identifica→ **Caminos** —que ordenan→ **Nodos**
  - Un **Índice** —mapea → **Propiedades** —bien a→ **Nodos** o **Relaciones**
  - Una **BBDD de Grafos** —gestiona un→ **Grafo** y —también sus→ **Índices**



# Conceptos de una BBDD de Grafos



# Neo4j

- **BBDD de grafos** especialmente potente cuando queremos **modelar datos conectados**
  - Usa el modelo basado en grafo de propiedades compuesto de nodos y relaciones con propiedades
- Cualquier dominio puede ser representado como un grafo y por tanto gestionado por Neo4j
  - *"Una base de datos tradicional, puede responderte rápidamente el promedio de ingresos de todos los que asisten a esta charla, pero una base de datos de grafos, puede decirte quién de ellos es más probable que te invite una cerveza"*

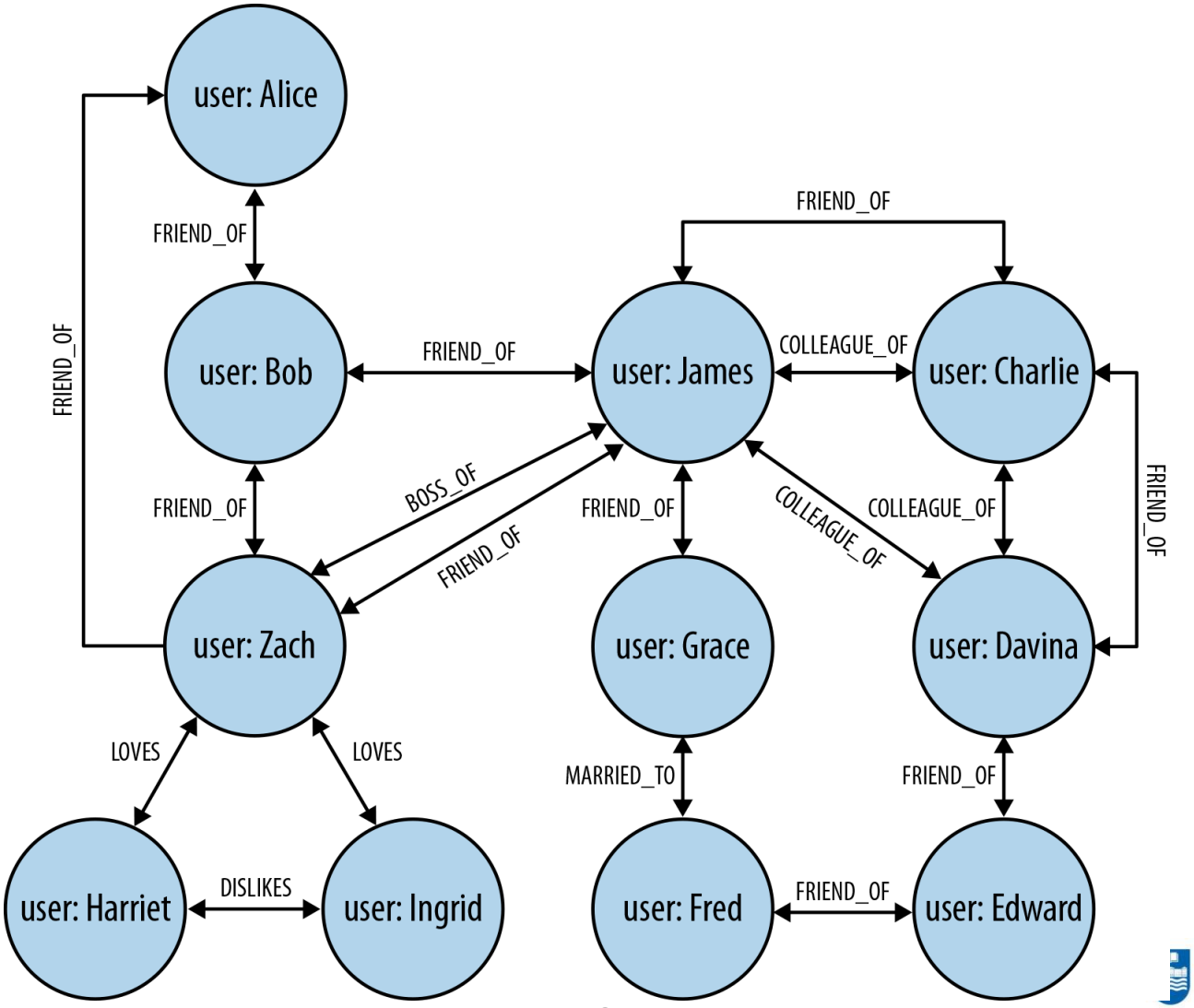
# Características de Neo4j

- Una Base de datos de grafos con:
  - **Alto desempeño y alta disponibilidad** (Escalamiento de lectura)
  - **Soporte** sólido y real para transacciones **ACID**
  - **Escalable**: 32 miles de millones de Nodos, 32 miles de millones de Relaciones, 64 miles de millones de Propiedades
  - **Servidor con una API REST o usable como una biblioteca Java**

# Aplicabilidad de Neo4j

- Es excelente para **modelar datos complejos y conectados**:
  - Recomendaciones
  - Inteligencia de negocios
  - Computación Social
  - Geoespacial
  - Administración de sistemas
  - Indexación de datos
  - Logística

# Ejemplo de una BBDD Grafos



# Instalar y ejecutar Neo4j

1. Descargar `neo4j-community-1.9.1` de: http://www.neo4j.org/download
2. Extraerlo en tu disco duro
3. Arrancar el servidor:
   - Windows: `bin\Neo4j.bat`
   - Linux/Mac: `bin/neo4j start`

- En un navegador abrir: http://localhost:7474/webadmin/
  - Incluye un Dashboard, data browser, console, etc.
- Desde REST podemos acceder al nodo raíz:

```
curl -v http://localhost:7474/db/data
```

# Drivers de lenguajes

- Soporte para diversos lenguajes de programación:
  - http://www.neo4j.org/develop/drivers
- Por ejemplo, podemos hablar con Neo4j:
  - A través de REST: Neo4j REST API
    - http://docs.neo4j.org/chunked/milestone/rest-api.html
  - Java API
    - http://docs.neo4j.org/chunked/milestone/tutorials-java-embedded.html

# Lenguaje de Traversal en Neo4j

- Un lenguaje declarativo a la SQL
- Detalles sobre el lenguaje en:
  - http://docs.neo4j.org/chunked/stable/cypher-query-lang.html
- Ejemplo:

```
// búsqueda de un amigo llamado Emil
start a=(1) match (a)--(b) where
b.name="Emil" return b
```

# Estructura de las consultas Cypher

- **START**: puntos de comienzo del grafo, obtenidos mediante búsquedas por ID o índices
- **MATCH**: el patrón del grafo a encontrar asociados a los puntos START
- **WHERE**: criterios de filtrado
- **RETURN**: qué devolver
- Ejemplos:

```
start andreas=(1) match (andreas)-
[:KNOWS]->(friends)-[:KNOWS]->(foafs)
return foafs
```

# HTTP Console

- Un lugar para experimentar con la API REST
- Ejemplos:
  - `get /` - La URL raíz del servidor
  - `get /db/data` – La raíz del acceso a datos
  - `get /db/data/node/0` – Nodo 0
  - `get /db/data/node/0/relationships/in` – relaciones entrantes
- Más detalles de la REST API en:
  - http://docs.neo4j.org/chunked/stable/rest-api.html

# Neo4j (1.9.2)



- **Written in:** Java
- **Main point:** Graph database - connected data
- **License:** GPL, some features AGPL/commercial
- **Protocol:** HTTP/REST (or embedding in Java)
- Standalone, or embeddable into Java applications
- Full ACID conformity (including durable data)
- Both nodes and relationships can have metadata
- Integrated pattern-matching-based query language ("Cypher")
- Also the "Gremlin" graph traversal language can be used
- Indexing of nodes and relationships
- Nice self-contained web admin
- Advanced path-finding with multiple algorithms
- Indexing of keys and relationships
- Optimized for reads
- Has transactions (in the Java API)
- Scriptable in Groovy
- Online backup, advanced monitoring and High Availability is AGPL/commercial licensed
- **Best used:** For graph-style, rich or complex, interconnected data. Neo4j is quite different from the others in this sense.
- **For example:** For searching routes in social relations, public transport links, road maps, or network topologies.

# ¿NoSQL or not NoSQL?

- En NoSQL, generalmente los datos son recuperados de manera mucho más rápida que en un RDBMS, sin embargo las consultas que se pueden hacer son más limitadas y requieren trasladar complejidad a la aplicación
- RDBMS para escribir usan locks y redos para garantizar ACID, pero NoSQL no soporta a menudo Atomicy, Consistency o Durability
  - Si quieres soporte transaccional integral debes seguir usando RDBMS
- Aplicaciones que generan informes emplean consultas complejas para las que NoSQL no es muy adecuado
- Aplicando MapReduce, las bases de datos NoSQL pueden paralelizar operaciones complejas como agregaciones estadísticas, filtros, agrupaciones o ordenación.
- Desde un punto de vista de sistemas deberíamos considerar la combinación de SQL y NoSQL:
  - LinkedIn comenzó sólo con un RDBMS, pero desarrolló su propia BBDD NoSQL (Voldemort)
  - Facebook tienen una arquitectura híbrida con Memcached y MySQL junto a un OLTP (envío de mensajes al Wall), y Cassandra/HBase para la búsqueda en la bandeja de entrada

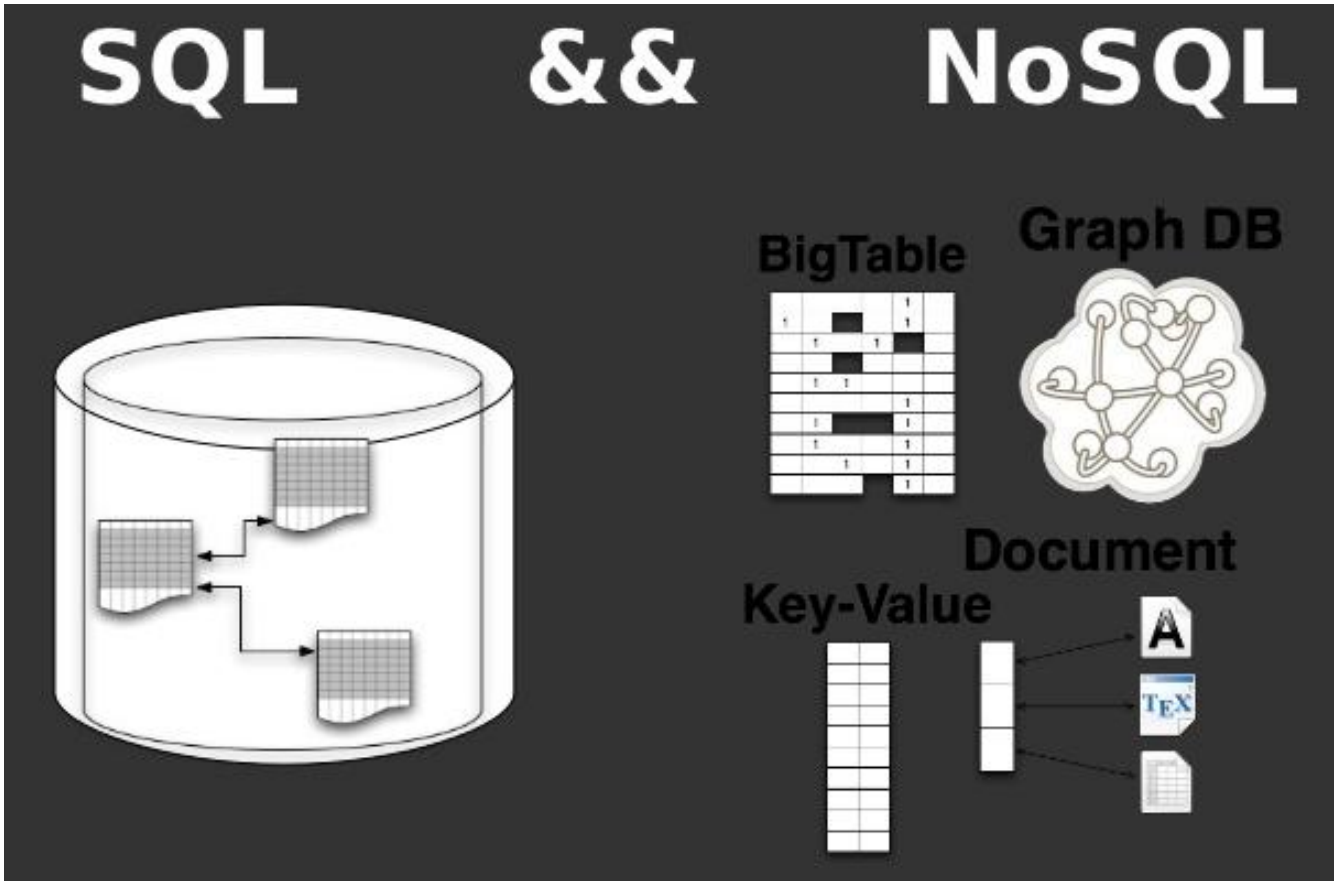
# ¿NoSQL or not NoSQL?

- Los principales problemas de NoSQL son:
  - Su complejidad:
    - Instalación
    - Consultas (comprender bien MapReduce)
    - Los modelos de datos usados
  - Su falta de madurez
- ¿Dónde usarlas?
  - Datos sociales
  - Procesado de datos (Hadoop)
  - Búsqueda (Lucene)
  - Caching (Memcache)
  - Data Warehousing
- ¿Qué problema quieres resolver?
  - Transacciones
  - Grandes volúmenes de datos (Exabytes)
  - Estructura de los datos

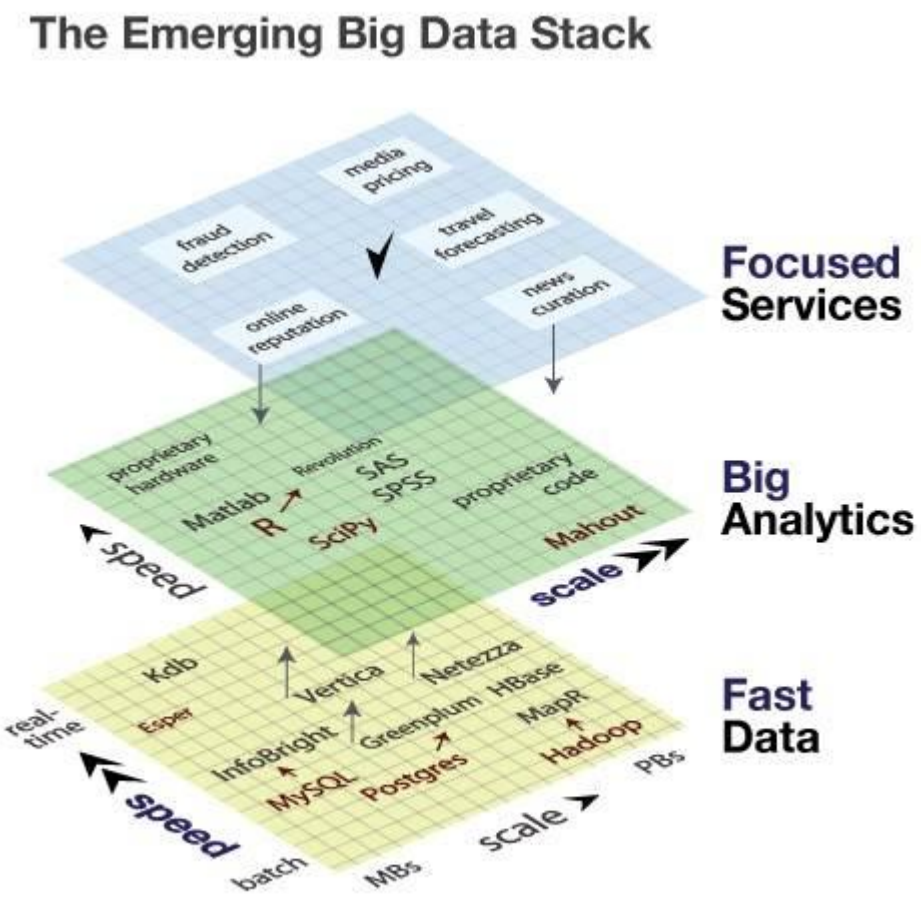
# Persistencia Políglota

- **Toda empresa va a acabar teniendo una variedad de tecnologías de almacenamiento para diferentes tipos de datos**
  - El código de nuestros sistemas va a tener acceso a diferentes repositorios de datos
  - Muchos datos seguirán guardándose en almacenes relacionales pero debemos empezar a preguntarnos cómo queremos utilizar los datos y sólo entonces decidir qué tecnología es mejor
    - Usar NoSQL para una característica particular de una aplicación
      - Rápido procesado batch
      - Logging distribuido
      - Para grandes tablas
    - Usar una RDBMS para reporting
  - Se complica la lógica de la aplicación y el despliegue
  - Responsabilidades administrativas adicionales
- Hay que **¡elegir la herramienta adecuada para cada trabajo!**

# Persistencia Políglota



# The Emerging Big Data Stack



# Conclusión

- Las BBDD NoSQL son una clara alternativa a los RDBMS
  - Sobre todo para algunas aplicaciones sociales y web que requieren elevada escalabilidad
- No son idóneas para todo, de hecho en la mayoría de los casos las RDBMS deberían seguir siendo la primera opción:
  - La capacidad de hacer JOIN y las garantías ACID son muy importantes para muchas aplicaciones
- Es muy posible que los RDBMS actuales evolucionen para incorporar capacidades de NoSQL
  - Lo que se va a estilar es ¡¡¡"persistencia polígota"!!!!

# The end

# Referencias

- **Cassandra**
  - "NoSQL – Not only SQL (Introduction to Apache Cassandra)"
    - http://www.scriptandscroll.com/3508/technology/nosql-not-only-sql-introduction-to-apache-cassandra/#.TtonPmMk6nA
  - DataSax company:
    - http://www.datastax.com/about-us/about-datastax
  - Getting started with CQL:
    - http://www.datastax.com/docs/0.8/dml/using_cql
  - http://cassandra.apache.org/

# Referencias

- **CouchDB**
  - Exploring CouchDB, Joe Lennon,
    - http://www.ibm.com/developerworks/opensource/library/os-CouchDB/index.html
  - CouchDB tutorial
    - http://net.tutsplus.com/tutorials/getting-started-with-couchdb/
  - CouchDB for geeks:
    - http://www.slideshare.net/svdgraaf/CouchDB-for-geeks?from=share_email
  - CouchDB site:
    - http://CouchDB.apache.org/
  - CouchApp.org: The 'Do It Yourself' Evently Tutorial
    - http://couchapp.org/page/evently-do-it-yourself
  - CouchApp.org: What the HTTP is CouchApp?
    - http://wiki.couchapp.org/page/what-is-couchapp
  - Tutorial: Using JQuery and CouchDB to build a simple AJAX web application
    - http://blog.edparcell.com/using-jquery-and-CouchDB-to-build-a-simple-we
  - CouchApp site:
    - http://couchapp.org/page/getting-started

# Referencias

- **MongoDB**
  - MongoDB reference manual, http://docs.mongodb.org/manual/MongoDB-reference-manual.pdf
  - mongoDB official page, http://www.mongodb.org/

- **Neo4j**
  - Welcome to the matrix ! With Neo4j, Benoît Simard, http://sim51.github.io/prez-neo4j/#/step-3
  - Neo4j – the graph database, http://www.neo4j.org/

# Referencias

- **NoSQL vs. RDBMS**
  - Riyaz -- Thanks for the question regarding "NOSQL vs. RDBMS databases", version 10r2
    - http://asktom.oracle.com/pls/asktom/f?p=100:11:0::::P11_QUESTION_ID:2664632900346253817
  - NoSQL or not NoSQL?
    - http://www.slideshare.net/ruflin/nosql-or-not-nosql/download
  - Comparativa de diferentes soluciones NoSQL:
    - http://kkovacs.eu/cassandra-vs-mongodb-vs-couchdb-vs-redis
  - SQL vs. NoSQL
    - http://www.linuxjournal.com/article/10770

# Referencias


- NoSQL, no problem. An introduction to NoSQL databases, Robert Rees, 27 September 2010
  - http://www.thoughtworks.com/insights/articles/nosql-comparison
- Modelo BASE vs. ACID:
  - http://www.johndcook.com/blog/2009/07/06/brewer-cap-theorem-base/
  - http://databases.about.com/od/otherdatabases/a/Abandoning-Acid-In-Favor-Of-Base.htm
- The future is Polyglot Persistence
  - http://martinfowler.com/articles/nosql-intro.pdf
- Cassandra vs MongoDB vs CouchDB vs Redis vs Riak vs HBase vs Couchbase vs Neo4j vs Hypertable vs ElasticSearch vs Accumulo vs VoltDB vs Scalaris, Kristof Kovács
  - http://kkovacs.eu/cassandra-vs-mongodb-vs-couchdb-vs-redis

# NoSQL: la siguiente generación de bases de datos

18 Julio de 2013, 9:30-10:45

Curso de Verano Big Data & Data Science, Universidade de Santiago de Compostela

http://eventos.citius.usc.es/bigdata/

**Dr. Diego Lz. de Ipiña Glz. de Artaza**

DeustoTech-INTERNET, DeustoTech – Deusto Institute of Technology, Universidad de Deusto

dipina@deusto.es

http://paginaspersonales.deusto.es/dipina

http://www.morelab.deusto.es