Bases de Datos No Convencionales

5. BD de Familias de Columnas: Cassandra

Universidad
Rey Juan Carlos

Bibliografía

- Dan Sullivan (2015). **NoSQL for Mere Mortals.** Addison-Wesley Professional
- Sam R. Alapati (2017). **Expert Apache Cassandra Administration.** Apress
- Sandeep Yarabarla (2017) **Learning Apache Cassandra**. Packt
- Avinash Lakshman , Prashant Malik (2010), **Cassandra: a decentralized structured storage system**, ACM SIGOPS Operating Systems Review, v.44 n.2, April 2010
- Artem Chebotko, Andrey Kashlev y Shiyong Lu (2015), **A Big Data Modeling Methodology for Apache Cassandra**, IEEE BigData Congress

Índice

1. Introducción
2. Arquitectura
3. CQL
4. Modelado

Índice

1. > Introducción
2. Arquitectura
3. CQL
4. Modelado

Introducción

- Ejemplo

```
CREATE TABLE empleado (
  nombre varchar,
  direccion varchar,
  despacho varchar,
  provincia varchar,
  PRIMARY KEY (nombre));

INSERT INTO empleado (nombre, direccion,
despacho, nacionalidad)
  VALUES ('Juan', 'C/ Corta 1', 'D-231',
  'Madrid');

SELECT * FROM empleado WHERE nombre = 'Juan';
```

Introducción

- Desarrollado inicialmente en Facebook
- Liberado como proyecto open-source por Facebook en 2008
- "*Cassandra - A Decentralized Structured Storage System*", Avinash Lakshman y Prashant Malik (2010)
- Actualmente, "top-level project" de Apache
- Jonathan Ellis y Matt Pfeil fundan DataStax en 2010

Introducción

Introducción

- BD de familias de columnas son las BD NoSQL más complejas en cuanto a su estructura
- Terminología común con bases de datos relacionales (incluso **CQL** "parece" SQL), pero hay importantes diferencias: no confundir

Introducción

- Componentes:
  - Columna: unidad básica de almacenamiento. Tiene un nombre y un valor (a veces también una marca de tiempo):
    - Nombre: "Juan"
  - Fila: conjunto de columnas. Las filas pueden tener las mismas o diferentes columnas
    - Nombre: "Juan", Apellido:"Pérez", Cargo: "Responsable RRHH"
    - Nombre: "María", Apellido: "García", Teléfono: "9999"
  - Familia de columnas: conjunto de columnas relacionadas, a menudo utilizadas conjuntamente (Nombre + Apellido).
  - Como en el resto de BD NoSQL, no hay obligación de esquema predefinido: pueden añadirse columnas, filas con diferentes columnas...
    - No es que valgan NULL

Introducción

- Similitudes y diferencias con:
  - Clave valor
    - Pueden añadirse nuevas claves y valores de igual forma que se pueden añadir columnas a familias de columnas
    - En el caso de las familias de columnas, para identificar al valor necesitamos el identificador de la fila, más la columna
  - Documentales
    - No todos los documentos tienen los mismos campos, y éstos pueden añadirse cuando sea necesario. Correspondencia fila-documento

Introducción

- Similitudes y diferencias con:
  - Relacionales:
    - Ambas identificador único por fila y concepto de fila/columna
    - Las BD de familia de columnas no están normalizadas. Toda la información relativa a un concepto está guardada en una única fila. Elimina o reduce la necesidad de joins
    - Los lenguajes de consulta son similares, las BD de familias de columnas tienen sentencias similares a SQL y otras específicas

Índice

1. Introducción
2. > Arquitectura
3. CQL
4. Modelado

Arquitectura

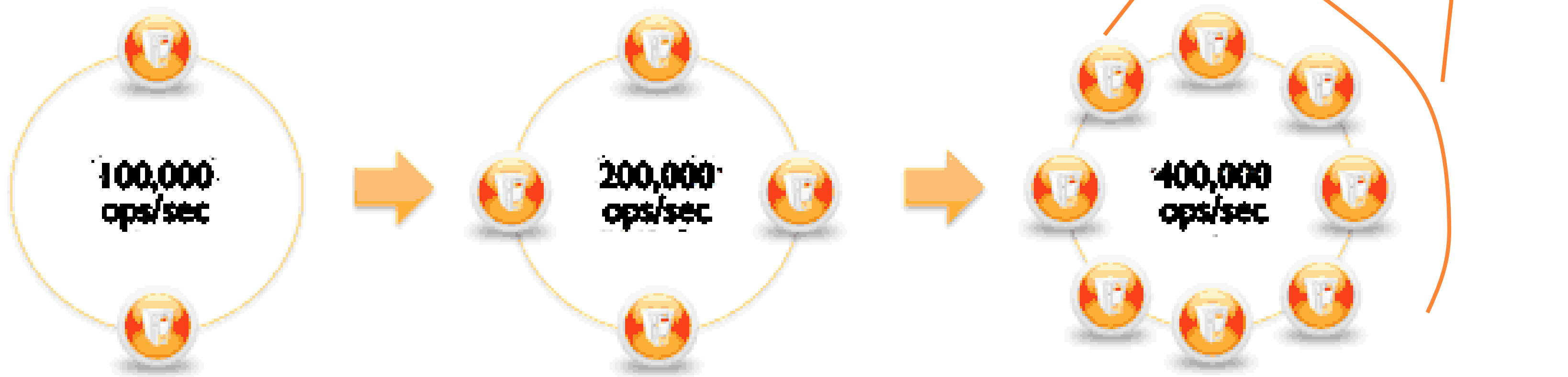
- Características
  - Cassandra es una BD diseñada para trabajar de manera distribuida en muchos nodos sin un único punto de fallo (Point Of Failure)
  - Diseño basado en que los fallos existen
  - Sistema distribuido peer-to-peer, datos distribuidos en todos los nodos del clúster
  - Intercambio de información frecuente entre los nodos
  - Log en cada nodo que captura las escrituras-> los datos se indexan y se escriben a una estructura en memoria (memtable)->cuando se llena, se escribe a disco (SSTables)
  - Todas las escrituras se particionan y replican en el cluster
  - Los usuarios pueden acceder a los datos a través de cualquier nodo para leer o escribir. Ese nodo se convierte en coordinador para esa operación
  - Normalmente, un keyspace por aplicación

Arquitectura

- Estructuras básicas
  - Columna: par clave-valor
  - Fila: conjunto de columnas identificadas por una PK
  - Partición: conjunto de filas con la misma **clave de partición**
  - Nodo: componente básico de la infraestructura de Cassandra, que almacena los datos
  - DataCenter: colección de nodos relacionados. Normalmente atiende a un conjunto concreto de peticiones (ej. DataCenter para Europa)
  - Clúster: uno o más DataCenters

Arquitectura

- Características
  - Distribución automática y transparente en los nodos que forman el cluster
  - Tolerancia a fallos, configurable
  - Escalabilidad lineal



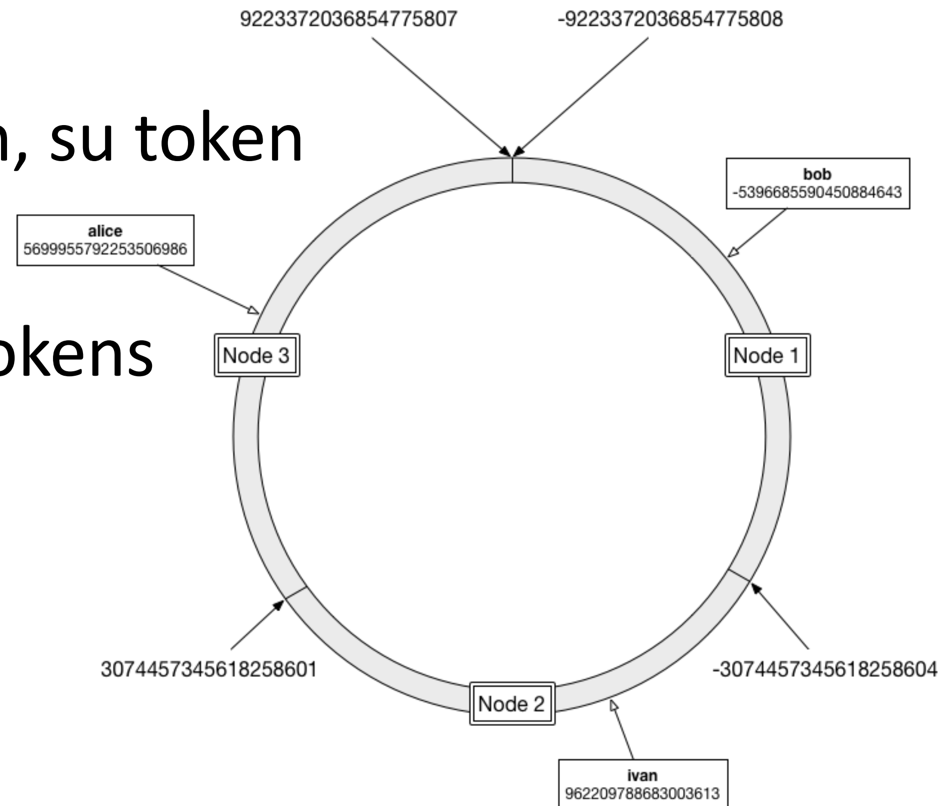
Fuente:datastax.com

Arquitectura

- Replicación de datos
  - Cassandra almacena réplicas en varios nodos.
  - El número total de réplicas se denomina **factor de replicación** (cada copia en un nodo diferente)
  - Configurable para cada datacenter
  - Se asigna el nodo de cada dato de acuerdo a una función de partición

Arquitectura

- Distribución de datos en Cassandra
  - Cada clave de partición, su token
  - Cada nodo, rango de tokens



Fuente: Learning Apache Cassandra

Arquitectura

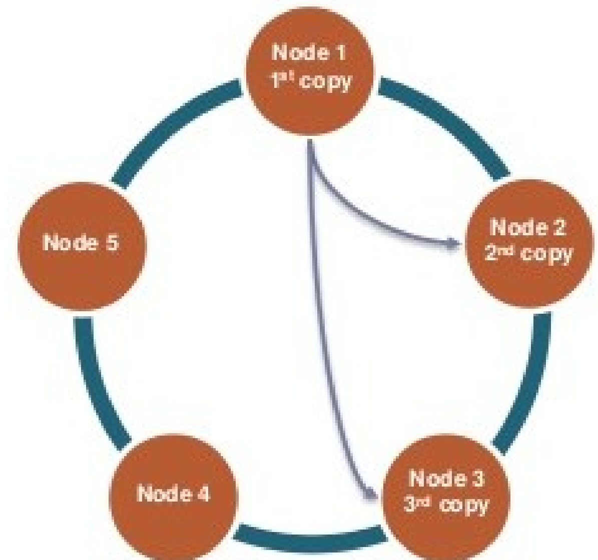
- Particionadores
  - Función hash
  - Determina, a partir de la clave de partición, el nodo correspondiente
  - Varias alternativas

Arquitectura

- Replicación de datos
  - Dos aspectos fundamentales a tener en cuenta para decidir el número de réplicas en cada datacenter:
    - Poder satisfacer las lecturas localmente
    - **Posibles fallos**
  - La estrategia de replicación se define para cada keyspace (keyspace ≈ base de datos)
  - Estrategias más habituales:
    - **Dos réplicas** en cada datacenter: tolera el fallo de un nodo y permitiría lecturas locales con un nivel de consistencia ONE
    - **Tres réplicas** en cada datacenter: tolera el fallo de un nodo y un nivel de consistencia de LOCAL_QUORUM o varios fallos por datacenter con un nivel de consistencia ONE
  - Pueden existir diferentes números de réplicas de los mismos datos en distintos datacenter

Arquitectura

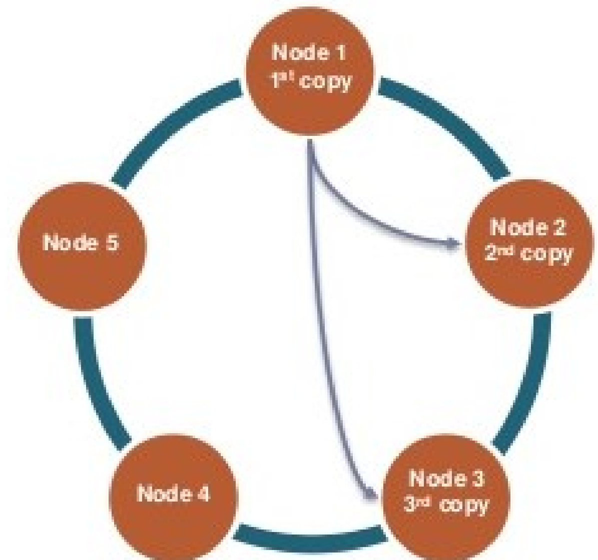
- Ejemplo
  - La sentencia puede llegar a cualquier nodo
  - Replicación a nivel de keyspace (db)
  - 3 réplicas en el datacenter



Fuente: datastax.com

Arquitectura

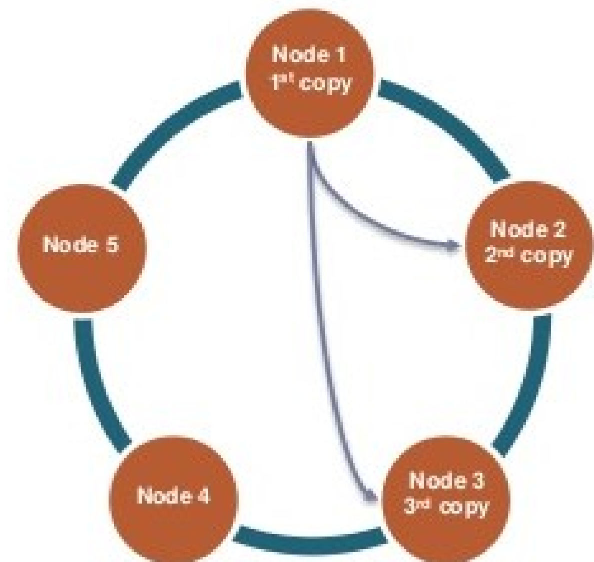
- Recordatorio:
  - Consistencia inmediata:
    - Cualquier escritura disponible para cualquier lectura
  - Consistencia eventual
    - Temporalmente inconsistente



Fuente: datastax.com

Arquitectura

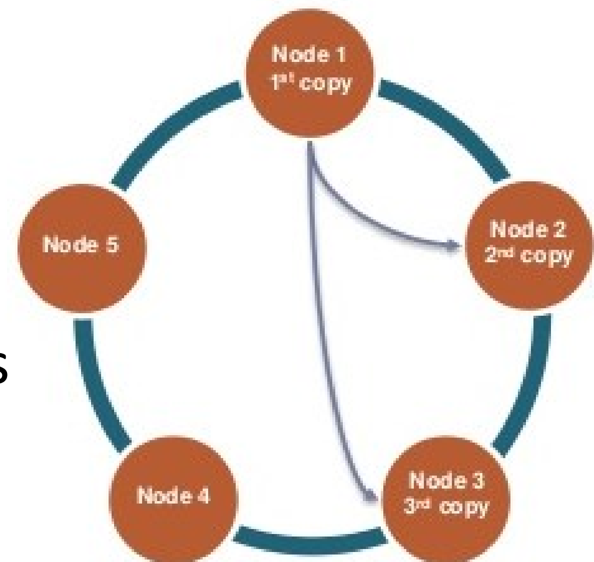
- Consistencia configurable
  - Además del número de réplicas, debemos configurar el nivel de consistencia
  - Cuántas réplicas implicadas
  - Posibles niveles
    - ONE
    - ALL
    - QUORUM



Fuente: datastax.com

Arquitectura

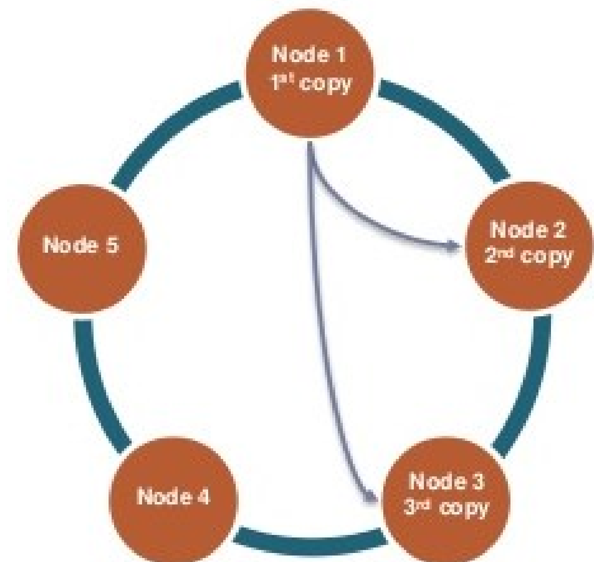
- Consistencia configurable:
  - Consistencia eventual con ONE
    - Lecturas y escrituras ONE
    - Suficiente con una confirmación
      - En escritura
      - En lectura
    - Podemos leer datos no actuales
    - Admite dos nodos no disponibles



Fuente: datastax.com

Arquitectura

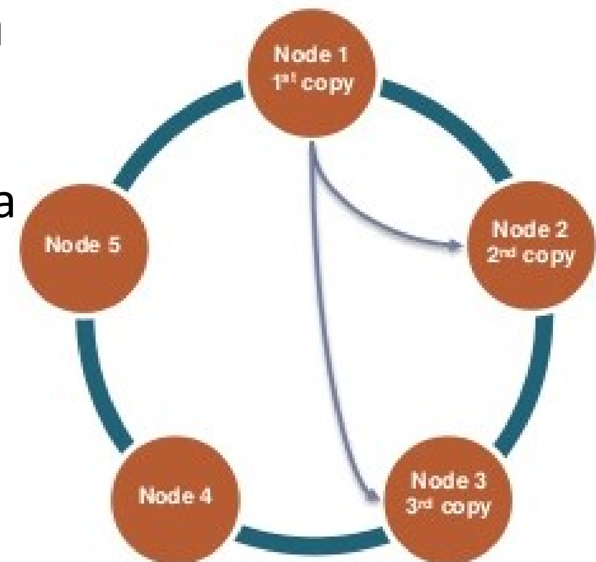
- Consistencia configurable:
  - Consistencia inmediata con ALL
    - Necesaria confirmación de todas (las 3)
      - En escritura
      - En lectura
    - Suficiente sólo en un caso
    - Baja tolerancia a fallos
    - Peor rendimiento
      - Espera a que todos respondan



Fuente: datastax.com

Arquitectura

- Consistencia configurable:
  - Consistencia inmediata con QUORUM
    - Basta con la mayoría
    - Garantiza consistencia inmediata
    - Admite caída de sólo un nodo
      - El más nuevo si no hay coincidencia
  - Otras
    - LOCAL, EACH...



Fuente: datastax.com

Arquitectura

- Virtual Nodes
  - A partir de Cassandra 1.2
  - Intermediarios entre los rangos de tokens y los nodos físicos
  - Cada nodo virtual, un rango de tokens
  - Cada nodo físico, un conjunto de nodos virtual
  - Simplifica la redistribución de nodos

Fuente: Learning Apache Cassandra

Índice

1. Introducción
2. Arquitectura
3. > CQL
4. Modelado

CQL

- Keyspace: estructura lógica donde Cassandra almacena los datos
  - Consta de un conjunto de tablas (+ vistas materializadas, funciones, ...)
  - ≈Base de datos
  - Define cómo se replican los datos
    - Cassandra recomienda 1 keyspace/aplicación
    - Si varios requisitos de replicación, varios keyspaces

CQL

- Replication:
  - SimpleStrategy: aplica el mismo factor de replicación para todos los data centers. Es la opción utilizada en desarrollo. Para un único nodo, el factor de replicación sería 1.

```
{'class': 'SimpleStrategy',
'replication_factor' : 3};
```

  - NetworkTopologyStrategy: para opciones de replicación más complejas, permiten aplicar factores de replicación diferentes:

```
{'class': 'NetworkTopologyStrategy', 'DC1' :
1, 'DC2' : 3}
```

CQL

- Se puede modificar (ALTER) o borrar (DROP) keyspaces
- Consultar los existentes:

```
DESC[RIBE] keyspaces;
```

- Utilizar un keyspace por defecto:

```
USE nombre_keyspace;
```

- Podemos consultar todas las tablas (o las del keyspace en el que estemos)

```
DESC[RIBE] tables;
```

- Para hacer referencia a una tabla desde otro keyspace:

```
mi_keyspace.mi_tabla
```

CQL

Casi todos los ejemplos sacados de: Learning Apache Cassandra

- Creación de tablas:

```
CREATE TABLE [IF NOT EXISTS] nombreTabla
  (
  nombreCol tipo [STATIC][PRIMARY KEY],
  ...
  [PRIMARY KEY (clave_partición,
               [clave_de_clustering])]
  ) [WITH opciones_de_tabla]
```

CQL

- Definición de las columnas:
  - Nombre de la columna
  - Tipo
  - Modificadores:
    - Primary key: identifica una fila. Todas las tablas deben tener clave primaria
    - Static: tiene el mismo valor para todas las filas que comparten la misma *partition key*. No puede ser *primary key*

CQL

- Definición de las columnas:
  - Tipos de datos
    - Strings: (text, ascii)
    - Enteros: (in, bigint, smallint...)
    - Boolean
    - Decimales y coma flotante (float, double, decimal)
    - UUID (uuid, timeuuid)
    - Blob
    - Colecciones
    - ...

CQL

- INSERT
  - Inserta una o más columnas para una fila (al menos la clave primaria)

```
INSERT INTO tabla (lista_de_columnas)
VALUES (valores)
[IF NOT EXISTS]
[USING TTL seconds | TIMESTAMP microseg]
```

  - Opciones:
    - USING TIMESTAMP: modifica el tiempo de ejecución de la sentencia (por defecto, el actual)
    - USING TTL: time-to-live (segundos) para los valores (por defecto, valor 0, no expiran)
  - También puede insertarse en formato JSON

CQL

- INSERT
  - Ejemplos:

```
INSERT INTO usuarios (nombreusuario", "email", "edad")
VALUES ('antonio', 'antonio@gmail.com', 25);
```

  - Inserción parcial

```
INSERT INTO usuarios (nombreusuario, edad)
VALUES ('bea', 27);
```

  - Una inserción con TTL

```
INSERT INTO usuarios (nombreusuario, edad)
VALUES ('federico', 60)
USING TTL 10;
```

  - Consulta válida

```
SELECT * FROM usuarios LIMIT 2;
```

  - Consulta con error

```
SELECT * FROM usuarios WHERE nombreusuario > 'antonio
```

  - Claves primarias, de partición y de clustering

CQL

- Primary key:
  - La clave primaria especifica tanto la posición como el orden de los datos almacenados
  - Tiene dos partes: Partition key (obligatorio) y clustering key (opcional)
    - Partition key: determina qué nodo almacenará la fila. Puede estar compuesta, en cuyo caso separa los datos en diferentes particiones
    - Clustering key o clustering column/s: ordena los datos en la partición
  - Es una decisión de diseño muy importante
  - La partition key agrupa filas en cada réplica. Las clustering columns especifican cómo se ordenan en la réplica

CQL

- Partition Key
  - Conforma la primera parte de la primary key. Es obligatorio:

```
CREATE TABLE t (k text PRIMARY KEY)
```

  - Una partición de tabla es un conjunto de filas que tienen el mismo valor de su partition key
  - Puede estar formada por una o más columnas
  - Ejemplo:

```
CREATE TABLE t (
    a int,
    b int,
    c int,
    d int,
    PRIMARY KEY ((a, b), c, d)
);
```

CQL

- Partition Key

```
SELECT * FROM t;
   a | b | c | d
  ---+---+---+---
   0 | 0 | 0 | 0     // fila 1
   0 | 0 | 1 | 1     // fila 2
   0 | 1 | 2 | 2     // fila 3
   0 | 1 | 3 | 3     // fila 4
   1 | 1 | 4 | 4     // fila 5
```

  - Las filas 1 y 2 comparten la misma partición
  - Las filas 3 y 4 comparten la misma partición
  - La fila 5 está en otra partición

  – Cassandra garantiza que todas las filas de la misma partición se almacenan en el mismo nodo. Por tanto, conviene elegir cuidadosamente la partition key para agrupar datos que se recuperen juntos

CQL

- Cláusula SELECT
  - Determina qué columnas devuelve la consulta, así como las transformaciones que deben aplicarse al resultado antes de devolverlo.

```
SELECT COUNT (*) FROM usuarios;
SELECT nombreusuario, email FROM
usuarios;
SELECT JSON nombreusuario, email
FROM usuarios;
SELECT nombreusuario AS nombre, email
FROM usuarios;
```

CQL

- Cláusula WHERE
  - Especifica qué filas vamos a recuperar. Sólo permite:
    - Columnas que forman la clave primaria (o índice sec.)
    - Para claves de partición, relaciones entre claves de clustering restringidas a aquellas que busquen en un conjunto contiguo de filas
  - Consulta que busca por clave primaria:

```
SELECT * FROM usuarios
WHERE nombre='antonio';
```

  - Sin embargo, la siguiente consulta da error:

```
SELECT * FROM usuarios
WHERE edad=25;
```

  - Necesita un índice secundario en la columna edad (o ALLOW FILTERING)

CQL

- Clave primaria compuesta
  - Pueden especificarse varias columnas para la clave de partición
  - De esta forma dividimos los datos en diferentes particiones

```
CREATE TABLE datos_sensor (
       anyo int,
       sensor text,
       evento timeuuid,
       dato text,
PRIMARY KEY ((anyo, sensor), evento)
       );
```

  - En este caso: partition key (anyo,sensor) y clustering column (evento), si estimamos partición muy grande

CQL

- Si la clave de partición es compuesta, debemos especificar todas las columnas

```
INSERT INTO datos_sensor (anyo, sensor, evento, dato)
VALUES (2019, 'sensor 1', NOW(), 'dato 001');
INSERT INTO datos_sensor (anyo, sensor, evento, dato)
VALUES (2019, 'sensor 1', NOW(), 'dato 002');
```

  - Consulta errónea

```
SELECT * FROM datos_sensor WHERE anyo = 2019;
```

  - Consulta correcta

```
SELECT * FROM datos_sensor WHERE anyo = 2019 AND sensor = 'sensor 1';
```

CQL

- "Compound primary key" y clustering columns
  - Primary key = Partition Key + Clustering Columns
  - Partition Key: puede ser simple or composite
  - Compound Primary Key: cuando tiene clustering column/s
  - Las clustering columns son opcionales. Su orden define el orden de una partición en la tabla
  - Por cada partición, Cassandra ordena físicamente las filas basándose en esas columnas (por defecto, ascendente)

```
CREATE TABLE t (
    a int,
    b int,
    c int,
    PRIMARY KEY (a, b));
```

  - a es partition key y b clustering column

CQL

- "Compound primary key" y clustering columns
- Tras insertar datos:

```
SELECT * FROM t;
  a | b | c
 ---+---+---
  0 | 0 | 4      // fila 1
  0 | 1 | 9      // fila 2
  0 | 2 | 2      // fila 3
  0 | 3 | 3      // fila 4
```

  – Las consultas que implican rangos de clustering columns son muy rápidas:

```
SELECT * FROM t where a=0 and b>1 and b<=3;
```

CQL

- Más ejemplos, con dos columnas de clustering:

```
CREATE TABLE mensajes (
  nombreusuario text,
  fecha date,
  hora time,
  cuerpo text,
  PRIMARY KEY (nombreusuario, fecha, hora));
```

  - Inserciones desordenadas

CQL

- Comprobamos el orden correcto

```
SELECT * FROM mensajes;
```

- Consulta correcta

```
SELECT * FROM mensajes
WHERE nombreusuario = 'antonio' AND fecha < '2016-11-20';
```

- Consulta errónea

```
SELECT * FROM mensajes
WHERE nombreusuario = 'antonio'
    AND fecha > '2016-11-20' AND  hora > '12:00:00';
```

  - Si se pone una condición de no igualdad en una columna de clustering, todas las anteriores condiciones de columnas de clustering deben ser de igualdad

- Consulta correcta

```
SELECT * FROM mensajes
WHERE nombreusuario = 'antonio'
    AND fecha = '2016-11-21' AND hora > '12:00:00';
```

CQL

- Columnas static
  - Una columna static lo es dentro de cada partición
  - Debe existir alguna clustering column

```
CREATE TABLE t (
  k text,
  s text STATIC,
  i int,
  PRIMARY KEY (k, i));
INSERT INTO t (k, s, i) VALUES ('k', 'Primer valor', 0);
INSERT INTO t (k, s, i) VALUES ('k', 'Segundo valor', 1);
SELECT * FROM t;
k |               s | i
----------------------------
k  | "Segundo valor" | 0
k  | "Segundo valor" | 1;
```

Universidad Rey Juan Carlos

CQL

- Ejemplo static

```
CREATE TABLE mensajes (
nombreusuario text,
fecha date,
hora time,
correo text STATIC,
cuerpo text,
PRIMARY KEY (nombreusuario, fecha, hora));
```

  - Insertamos usuario con email

```
INSERT INTO mensajes (nombreusuario, fecha, hora,
correo, cuerpo)
VALUES ('antonio', '2020-04-24', '09:00:00',
'antonio@mail.com', 'Antonio Update 7');

SELECT * FROM mensajes;
```

CQL

- UPDATE
  - En realidad, es un UPSERT (si no existía, se crea). Funciona como un INSERT
  - También se pueden indicar TIMESTAMP y TTL (ojo, lo borra)
  - Pueden actualizarse varias filas y varias columnas

```
UPDATE usuarios SET edad = 26,
email='antonio@nuevo.com'
WHERE nombreusuario = 'antonio';

UPDATE usuarios SET edad=18
WHERE nombreusuario IN ('juan','luis','ana');

UPDATE usuarios USING TTL 5 SET edad = 90
WHERE nombreusuario = 'antonio';
```

CQL

- ALTER TABLE
  - Cambiar (algunos) tipos de columnas, añadir nuevas columnas, cambiar algunas propiedades...

```
ALTER TABLE clientes
   ADD direccion varchar;
ALTER TABLE clientes
   WITH comment = 'tabla con datos de
clientes' AND default_time_to_live = 1000;
```

- DESCRIBE nombretabla;
  - Muestra las propiedades de una tabla

CQL

- DROP TABLE nombre [IF EXISTS]
  - Antes hay que eliminar posibles vistas materializadas basadas en la tabla
- TRUNCATE [TABLE] nombretabla
  - Borra todos los datos de la tabla

CQL

- Borra filas:

```
DELETE FROM usuarios WHERE nombre='bea';
```

- Borra columnas de filas:

```
DELETE edad FROM usuarios WHERE
nombreusuario = 'gemelo3';
```

- Borra varios valores:

```
DELETE FROM usuarios WHERE nombreusuario
IN ('gemelo1', 'gemelo2');
```

CQL

- Ordenación en las consultas
  - Por defecto, las particiones se encuentran ordenadas ascendentemente por sus clustering columns
  - En una consulta, podemos reordenar la salida, siempre y cuando implique un orden total (aunque es mejor que la tabla esté ordenada previamente por las ordenaciones más habituales)

```
WITH CLUSTERING ORDER BY
(clustering_column1 ASC,
clustering_column2 DESC)
```

CQL

- Índices
  - Podemos crear índices secundarios para poder hacer consultas por campos no clave
  - CREATE INDEX nombreÍndice ON [Keyspace.]nombreTabla(nombreColumna);
  - DROP INDEX [IF EXISTS] [Keyspace.]nombreÍndice

CQL

- Vistas Materializadas
  - Tabla creada a partir de otra existente utilizando otra clave primaria
  - Alternativa al uso de índices

```
CREATE MATERIALIZED VIEW MENSAJES_ANYO AS
    SELECT FECHA, HORA, NOMBREUSUARIO, CUERPO
    FROM MENSAJES
    WHERE NOMBREUSUARIO IS NOT NULL
    AND FECHA IS NOT NULL AND HORA IS NOT NULL
    PRIMARY KEY (FECHA, HORA, NOMBREUSUARIO);
```

  - Los atributos de la PK de la tabla base deben serlo también de la vista
  - La cláusula WHERE debe incluir todas las columnas de clave primaria con la opción IS NOT NULL
  - Muchas otras restricciones

CQL

- Vistas Materializadas

```
DROP MATERIALIZED VIEW nombre_VM;
```

- Pueden crearse múltiples MV basadas en la misma tabla
- Cuando se actualice la tabla base se actualizan las MV basadas en ella (asíncrono)
- Escrituras son más lentas

CQL

- Más de consultas
  - Problemas para realizar consultas que impliquen un filtro en los resultados (no devolver todos los registros del conjunto obtenido)
  - El motivo es que las consultas que devuelven todo el resultado tienen un rendimiento predecible, y es proporcional al tamaño de los datos del resultado

```
CREATE TABLE emails (
    emailId int,
    time int,
    from text,
    content text,
    PRIMARY KEY(emailId, time));
```

  - Funciona

```
SELECT * from emails;
```

  - No funciona:

```
SELECT * FROM emails WHERE time = 1418306451235;
Bad Request: Cannot execute this query as it might involve data
filtering and thus may have unpredictable performance. If you want
to execute this query despite the performance unpredictability,
use ALLOW FILTERING.
```

Universidad Rey Juan Carlos

CQL

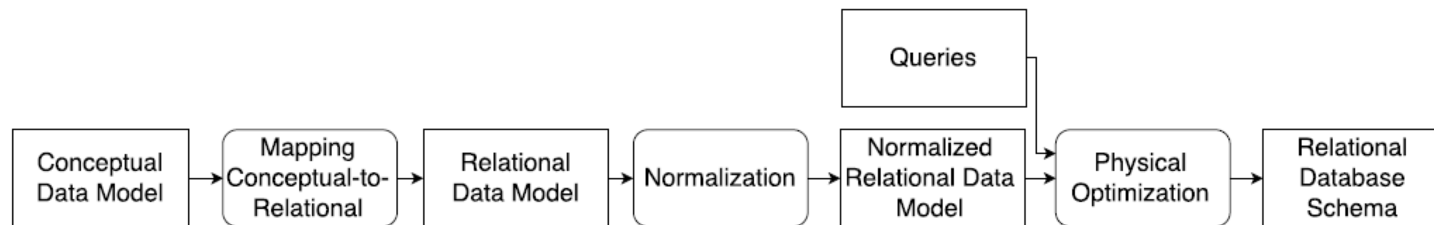
- Más de consultas
  - Motivo: Cassandra no ejecuta una consulta si no puede garantizar un uso eficiente de los recurso, aunque estén implicadas pocas filas
  - Sin embargo, si sabemos que más del 90% de las filas cumplirán la condición, podemos especificar la cláusula ALLOW FILTERING, en otro caso, Cassandra recuperará muchas filas para obtener unas pocas. En este caso, podemos crear un índice secundario
  - La mejor estrategia depende de los datos y las consultas

Índice

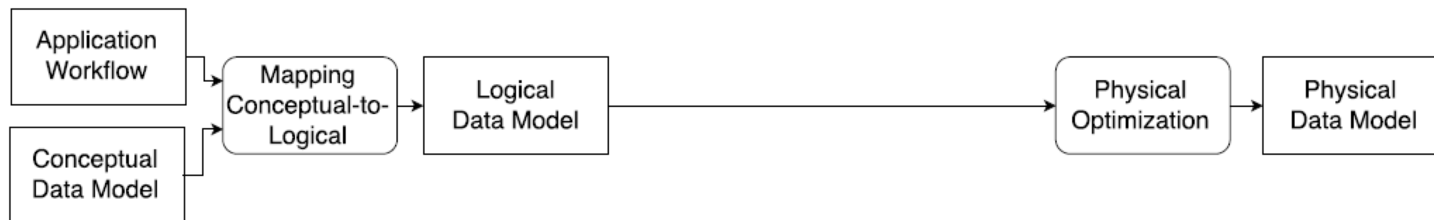
1. Introducción
2. Arquitectura
3. CQL
4. > Modelado

Modelado

- Proceso de modelado data-driven -> query-driven
  - Considerar el uso de la aplicación al mismo nivel que el modelado de los datos
  - Eliminación de la normalización: duplicación de datos



(a) Relational Data Modeling

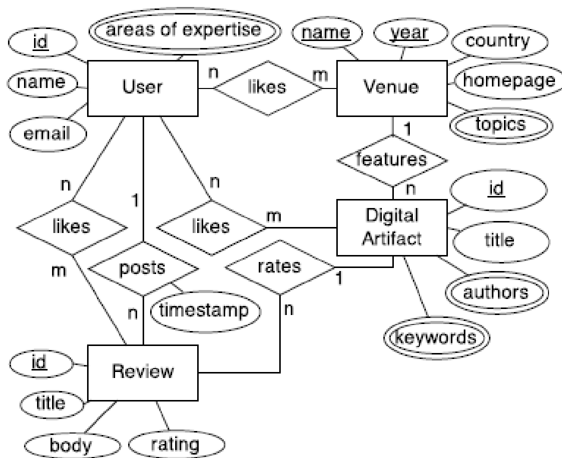


(b) Cassandra Data Modeling

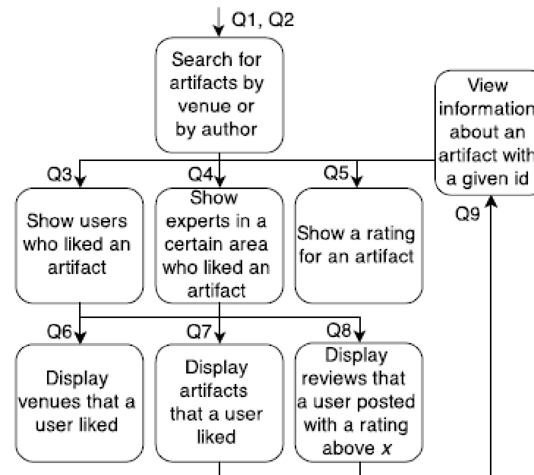
Fuente :Chebotko et al.

Modelado

- Proceso de modelado data-driven -> query-driven
  - A partir de los dos modelos, reglas de transformación
  - ER y workflow



(a) Conceptual data model



Q1: Find artifacts published in a venue with a given name after a given year. Order results by year (DESC).
Q2: Find artifacts published by a given author. Order results by year (DESC).
Q3: Find users who liked a given artifact.
Q4: Find users who liked a given artifact and who have expertise in a certain area.
Q5: Find an average rating of a given artifact.
Q6: Find venues that a given user liked.
Q7: Find artifacts published after a certain year that a given user liked. Order results by year (DESC).
Q8: Find reviews posted by a given user with a rating >=*x*. Order results by rating (DESC).
Q9: Find information about an artifact with a given id.

(b) Application workflow

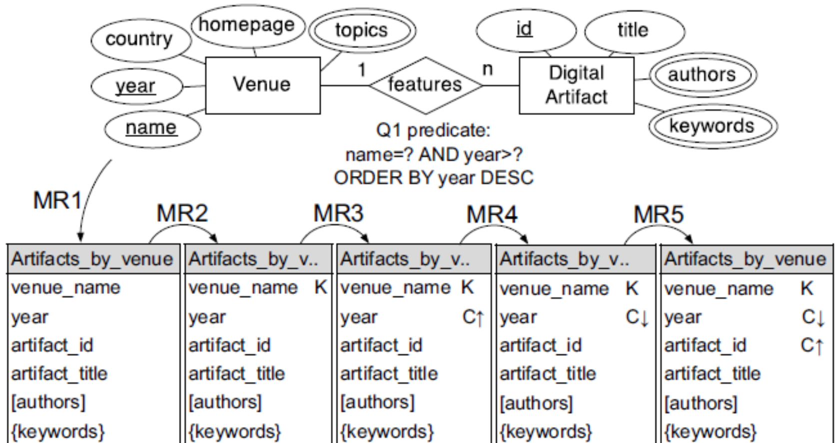
Fuente :Chebotko et al.

Modelado

- Diagramas de Chebotko y ejemplo

Table schema

Table Name			
column name 1	*CQL-Type*	K	Partition key column
column name 2	*CQL-Type*	C↑	Clustering key column (ASC)
column name 3	*CQL-Type*	C↓	Clustering key column (DESC)
column name 4	*CQL-Type*	S	Static column
column name 5	*CQL-Type*	IDX	Secondary index column
column name 6	*CQL-Type*	++	Counter column
[column name 7]	*CQL-Type*		Collection column (list)
{column name 8}	*CQL-Type*		Collection column (set)
<column name 9>	*CQL-Type*		Collection column (map)
column name 10	*CQL-Type*		Regular column



Fuente :Chebotko et al.