Tutorial de Cassandra

Comandos básicos de Cassandra

Apache Cassandra puede descargarse e instalarse desde https://cassandra.apache.org/download/. En el siguiente enlace teneis una guia de como instalar y empezar a utilizar Cassandra. En este tutorial de primeros pasos asumimos que se utiliza la máquina virtual facilitada en clase, pero puede seguirse utilizando cualquier instalación de Cassandra.

Arrancar el servicio de Cassandra

Para arrancar la base de datos se deberá ejecutar el siguiente comando[1]:

```
sudo service cassandra start
```

Parar el servicio Cassandra

La siguiente sentencia parará el motor de la base de datos:

```
sudo service cassandra stop
```

Conectarse al intérprete de comandos de Cassandra

Para conectarse al intérprete de comandos es necesario ejecutar el siguiente comando en un terminal.

```
cqlsh
```

[1] El password de la máquina virtual es cassandra.

Crear una base de datos

En el momento de la creación hay que establecer valor para los atributos:

- *Replica placement strategy*: este atributo permite indicar la manera en que se deben gestionar las réplicas (la estratégia de gestión de réplicas a utilizar). Para el tutorial utilizaremos el valor "SimpleStrategy", que aloja la primera réplica en el nodo indicado y las siguientes en los nodos colindantes del anillo siguiendo las agujas del reloj.
- *Replication factor*: indica el número de réplicas para cada fila. En el tutorial trabajaremos con un valor de replicación de 1, es decir, sin réplicas.

```
CREATE KEYSPACE tweetssandra WITH REPLICATION = { 'class'
:
'SimpleStrategy', 'replication_factor' : '1' };
```

Podemos consultar que el KEYSPACE[2] tweetssandra se ha creado correctamente:

```
DESCRIBE KEYSPACES;

SELECT * FROM system.schema_keyspaces;
```

Indicamos que queremos usar la base de datos tweetssandra:

```
USE tweetssandra;
```

Comprobamos que la base de datos está vacía:

```
DESCRIBE TABLES;
```

Eliminar la base de datos

Para eliminar la base de datos podemos ejecutar el siguiente comando:

```
DROP KEYSPACE tweetssandra;
```

[2] El *Keyspace* vendría a ser el equivalente a una base de datos en un modelo relacional. Es el contenedor de un conjunto de familias de columnas y permite configurar la estrategia de replicación, el factor de replicación y la durabilidad de las escrituras de las mismas. Por otro lado, las familias de columnas (o tablas) son contenedores de una colección ordenada de filas, donde cada fila contiene un conjunto ordenado de columnas. Vendrían a ser el equivalente a las relaciones en el modelo relacional.

Primeros pasos con Cassandra

Para ejemplificar el uso de Cassandra vamos a crear una base de datos que contenga información de Twitter. Para simplificar el ejemplo, supondremos que estamos interesados en almacenar información sobre usuarios de Twitter y, para cada usuario, sus seguidores, los tweets realizados y el número de veces que ha sido retweeteado.

Creación de tablas

Lo primero que haremos será crear una tabla (o familia de columnas) de usuarios para almacenar los datos de los usuarios de Twitter. La clave primaria de la tabla será el nombre (login) de usuario. Por defecto la clave primaria se utilizará en Cassandra como clave de partición, es decir, para identificar en qué nodo donde deberá almacenarse cada registro.

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

A continuación definimos una tabla para indicar los seguidores. Definiremos la siguiente estructura, donde para cada usuario (username) se indican sus seguidores (follower).

```
CREATE TABLE followers (
      username text,
      follower text,
      since timestamp,
      PRIMARY KEY (username, follower)
);
```

Podemos observar que la tabla *follower* tiene una clave compuesta, ya que un usuario tendrá, en el caso general, más de un seguidor. Una clave compuesta en Cassandra está formada por una clave de partición y una clave de agrupación. La clave de partición se utiliza para ubicar el nodo donde se almacenará el registro. La clave de agrupación indica el orden en que se ordenarán los datos en disco dentro de cada nodo. Por defecto, el primer campo de una clave primaria se utilizará como clave de partición y el resto como clave de agrupación, pero puede configurarse de forma distinta.

Posteriormente podríamos crear una tabla para almacenar los tweets:

```
CREATE TABLE tweets (
      tweet_id uuid PRIMARY KEY,
      username text,
      body text
);
```

Un posible diseño de la tabla que permite almacenar el timeline de Twitter podría ser el siguiente:

```
CREATE TABLE timeline (
  user_id text,
  tweet_id uuid,
  author text,
  body text,
  PRIMARY KEY (user_id, tweet_id)
);
```

Una de las particulares de Cassandra es que no es posible interrelacionar familias de columnas ni realizar productos cartesianos, lo cual tiene implicaciones en el diseño de las familias de columnas. En este sentido, hay que observar que si una consulta implicase a datos de varias familias de columnas, obligaría a realizar las búsquedas sobre cada una de ellas, introduciendo una gran ineficiencia. Es por ello, que un principio de diseño que debe seguirse cuando se crea una familia de columnas es tener en cuenta las consultas que se van a realizar, de manera que se creará una única familia de columnas por cada consulta que se tenga que realizar conteniendo toda la información que sea necesaria consultarse y organizada de acuerdo a la forma de acceso que requiere la consulta. De esta forma se evita la ineficiencia de tener que buscar en varias familias de columnas.

En el ejemplo anterior se han seguido estos principios de diseño creando familias de columnas específicas para los diferentes tipos de consultas que se quieren realizar, asegurando que toda la información necesaria se encuentra en cada familia de columna, y que no será necesaria consultar otras familias de columnas.

Otro aspecto importante a destacar es acerca de la formación de la clave primaria de las familias de columnas. La clave primaria de una familia de columnas está formada tal como se ha mencionado más arriba por dos tipos de clave: la clave de partición y la clave de agrupamiento. La clave de partición influye en el lugar donde se almacenarán las filas de la familia de columna, de manera que todas aquellas que tengan la misma

clave de partición se almacenan juntas. En cambio, la clave de agrupamiento influye en el ordenamiento de las filas que se encuentran almacenadas juntas. Por ejemplo en la familia de columnas followers antes definidas, todos los seguidores de un mismo usuario estarán almacenados juntos y ordenados de acuerdo al nombre del seguidor. Como ya se ha comentado la clave de agrupamiento afecta a la ordenación. En este sentido, por defecto los valores se ordenan de forma ascendente por el valor de la clave de agrupamiento. Sin embargo es posible configurar la familia de columnas para ordenarlos de manera descendente. Para ello se usa la cláusula WITH CLUSTERING ORDER BY que indica explícitamente que los valores se almacenarán ordenados de manera descendente. Por ejemplo considerar el siguiente ejemplo:

```
CREATE TABLE followers (
      username text,
      follower text,
      since timestamp,
      PRIMARY KEY (username, follower))
WITH CLUSTERING ORDER BY (follower DESC)
```

Con esta definición se está indicando que cuando se almacenen los seguidores de un mismo usuario, estos se deben ordenar de manera descendente de acuerdo al nombre del seguidor.

Tanto la clave de partición como la clave de agrupamiento puede estar formada por más de una columna. Para representarlo, si hubiera más de una clave de partición se encierran entre paréntesis y separadas por comas, y si hubiera más de una clave agrupamiento, simplemente se listan separadas por comas. Por ejemplo considerar el siguiente ejemplo:

```
PRIMARY KEY ((username, id-tweet), location-tweet,
date-tweet)
```

En el ejemplo (username,id-tweet) constituyen las claves de partición de la clave primaria mientras que las claves location-tweet y date-tweet constituyen las claves de agrupamiento.

Inserción de datos

Para insertar datos se puede utilizar la sentencia INSERT, parecido a como hacemos con SQL. Para ejemplificarlo creamos un par de usuarios, e indicamos que uno es seguidor del otro y creamos un par de tweets:

```
INSERT INTO users (username, password)
  VALUES ('jvaldez', 'cafe');

INSERT INTO users (username, password)
  VALUES ('conchita', 'colombia');

INSERT INTO followers (username, follower, since)
  VALUES ('jvaldez', 'conchita', '2013-01-28 15:30');

INSERT INTO tweets(tweet_id, username, body)
         VALUES     (01234567-0123-0123-0123-0123456789ab,
'jvaldez', 'El sabor de un buen café');

INSERT INTO tweets(tweet_id, username, body)
         VALUES     (01234567-0123-0123-0123-0123456789bc,
'conchita', 'El mejor café del mundo');
```

Modificación de tablas

Supongamos que descubrimos que nos hemos olvidado de la columna *name* en la tabla *users*. Podemos añadirla mediante la sentencia ALTER TABLE:

```
ALTER TABLE users ADD name text;

UPDATE users SET name = 'Juan Valdez' WHERE
username='jvaldez';

INSERT INTO users (username, password, name)
  VALUES ('FNC', 'federacion',  'Federación Nacional de
Cafeteros');

INSERT INTO followers (username, follower, since)
  VALUES ('jvaldez', 'FNC', '2013-01-28 15:30');

INSERT INTO followers (username, follower, since)
  VALUES ('conchita', 'FNC', '2013-01-28 15:30');
```

Creación de contadores

En Cassandra pueden utilizarse columnas de tipo contador para almacenar números que sólo pueden cambiarse incrementalmente. Por ejemplo, podríamos utilizar un contador para contar el número de veces que un usuario es retweeteado.

Para ello debemos crear el contador en una tabla dedicada, donde hay como mínimo una columna de tipo contador. Todas las columnas de la tabla que no sean de tipo contador deberán formar parte de la clave primaria.

```
CREATE TABLE user_retweeted_counts
  (counter_value counter,
     user_id text PRIMARY KEY,
);
```

Si el tweet de un usuario es retweeteado podríamos incrementar el contador con la siguiente sentencia:

```
UPDATE user_retweeted_counts
  SET counter_value = counter_value + 1
  WHERE user_id ='jvaldez';
```

Columnas de tipo colección

Las columnas de tipo colección permiten almacenar diferentes valores en una sola columna y son útiles para cuando queremos desnormalizar una pequeña cantidad de datos. Básicamente podemos encontrar 3 tipos de colecciones en Cassandra: set, list y map.

Las colecciones de tipo **set** nos permiten definir un conjunto de valores, por tanto, los valores no estarán ordenados ni permitirán repetición. Como ejemplo, podríamos modificar la tabla *users* para añadir las direcciones de correo de cada usuario:

```
ALTER TABLE users ADD emails set<text>;

UPDATE users SET emails = emails +
     {'jv@cafeAdictos.org','jvaldez@FNC.org'}
WHERE username = 'jvaldez';
```

Las colecciones de tipo **list** nos permiten definir un conjunto ordenado de valores, que pueden estar repetidos. Como ejemplo, podríamos modificar la tabla *users* para añadir la lista de ciudades donde ha vivido cada usuario.

```
ALTER TABLE users ADD lived_in list<text>;

UPDATE users SET lived_in=
     [ 'Bogota', 'Medellin', 'Bogota' ]
WHERE username = 'jvaldez';

UPDATE users SET lived_in=
     [ 'Bogota', 'Medellin', 'Bogota' ]
WHERE username = 'conchita';
```

Las colecciones de tipo **map** nos permiten definir pares de datos. Dejamos a vuestro cargo realizar ejemplos con este tipo de colección.

Creación de índices

En Cassandra se puede crear índices mediante la sentencia CREATE INDEX ON. A continuación indexamos el cuerpo de los tweets.

```
CREATE INDEX ON tweets (body);
```

Consulta de filas

Las consultas se usan de forma parecida a SQL. Podríamos obtener todos los usuarios mediante la siguiente sentencia:

```
SELECT * FROM users;
```

Para obtener los seguidores de Juan Valdez podríamos ejecutar la siguiente sentencia:

```
SELECT follower FROM followers
  WHERE username = 'jvaldez';
```

La cláusula WHERE se utiliza para filtrar las filas. Se puede aplicar sólo sobre columnas que forman parte de la clave partición o que están indexadas.

En caso de que queramos consultar filas en columnas de algún tipo "colección" se puede utilizar la condición CONTAINS en la cláusula WHERE. La siguiente consulta nos devolvería los usuarios que han vivido en Bogotá.

```
SELECT username FROM users
WHERE lived_in CONTAINS 'Bogota';
```

No obstante, la sentencia anterior nos dará un error porque el campo *lived_in* no está indexado. Por tanto, habrá que indexarlo y después preguntar de nuevo. Para ello necesitaremos una versión posterior a 2.1 de Cassandra, ya que las versiones anteriores no permiten indexar campos de tipo colección.

```
CREATE INDEX ON users (lived_in);

SELECT username FROM users
WHERE lived_in CONTAINS 'Bogota';
```

Un caso particular que hay que tener en cuenta en las consultas es cuando es necesario representar condiciones en la cláusula WHERE sobre columnas que no forman parte de la clave de partición o no están indexadas. En estos casos es posible evitar esta restricción si se usa la cláusula ALLOW FILTERING. Por ejemplo considerar la siguiente familia de columnas:

```
CREATE TABLE Imagenes-Tweets (username text, id-tweet
text,id-imagen text, año int) PRIMARY
KEY(username,id-imagen, id-tweet)
```

La siguiente sentencia busca toda la información sobre imágenes que fueron enviadas en tweets por diferentes usuarios a partir del año 2010.La búsqueda se realiza sobre el campo año aunque no forme parte de la clave de partición.

```
SELECT username, id-tweet,id-imagen, año FROM
Imagenes-Tweets WHERE año>2010 ALLOW FILTERING
```

En el caso de la máquina virtual, la versión es anterior a la 2.1, por tanto no podríamos ejecutar la consulta. Se puede consultar la versión de Cassandra mediante CQL utilizando la siguiente sentencia:

```
SHOW VERSION
```

Eliminar datos

Se pueden eliminar datos mediante la consulta DELETE. A continuación eliminamos el usuario *FNC* .

```
DELETE FROM users WHERE username='FNC';
```

También podemos eliminar campos de una fila, como por ejemplo los lugares donde ha vivido *conchita*.

```
DELETE lived_in FROM users WHERE username='conchita';
```

Eliminar tablas

Para eliminar tablas podemos utilizar la sentencia DROP TABLE. A Continuación eliminamos todas las tablas de la base de datos:

```
DESCRIBE TABLES;
DROP TABLE users;
DROP TABLE tweets;
DROP TABLE followers;
DROP TABLE user_retweeted_counts;
```

Comprobamos que no queda ninguna tabla mediante:

```
DESCRIBE TABLES;
```