Creación del modelo de datos

Ejecución de scripts de CQL

Para ejecutar instrucciones CQL podemos simplemente lanzar una shell de CQLSH en un contenedor de Cassandra como vimos en la sección de creación del cluster:

```
docker exec -it cass1 cqlsh
```

Pero pronto nos daremos cuenta de que hay instrucciones que tienen una sintaxis demasiado compleja para ser ejecutadas desde la shell. Por ello, lo más cómodo es escribir las instrucciones CQL en un fichero de texto y ejecutarlas desde la shell de CQLSH.

Los *scripts* de CQL son ficheros de texto con extensión `.cql` que contienen instrucciones CQL. Para ejecutar un *script* de CQL desde la shell de CQLSH utilizamos la sentencia `SOURCE`:

```
SOURCE 'path/to/script.cql';
```

Pero si lo que queremos es ejecutar un *script* de CQL desde la línea de comandos utilizaremos el siguiente comando:

```
cqlsh -f path/to/script.cql
```

Como en nuestro ejemplo vamos a utilizar contenedores Docker, vamos a hacer lo siguiente.

Cuando creamos el fichero `docker-compose.yml` indicamos que el directorio `./scripts` del host se montase en el directorio `/scripts` de los contenedores. Por lo tanto, si queremos ejecutar un *script* de CQL desde la línea de comandos, lo que tenemos que hacer en primer lugar es copiar el *script* en el directorio `scripts` del host para luego ejecutar el siguiente comando:

```
docker exec -it cass1 cqlsh -f /scripts/script.cql
```

Creación de un *keyspace*

Para crear un *keyspace* utilizamos la sentencia `CREATE KEYSPACE`:

```
CREATE KEYSPACE <keyspace_name>
  WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : <n> };
```

Existe también la sentencia opcional `AND DURABLE_WRITES = <verdadero o falso>` que indica que si los datos se han de escribir o no en el disco. Esta opción está activada por defecto.

Por ejemplo, para crear un *keyspace* llamado `my_keyspace` con una estrategia de replicación `SimpleStrategy` y un factor de replicación de 1 en el *datacenter 1* y 3 en el *datacenter 2* utilizaríamos la siguiente sentencia:

```
CREATE KEYSPACE my_keyspace
  WITH REPLICATION = { 'class' : 'NetworkTopologyStrategy', 'dc1' : 1, 'dc2' : 3
};
```

Para indicar que vamos a utilizar el *keyspace* `my_keyspace` utilizamos la sentencia `USE`:

```
USE my_keyspace;
```

Modificar un *keyspace*

Para modificar un *keyspace* utilizamos la sentencia `ALTER KEYSPACE`:

```
ALTER KEYSPACE <keyspace_name>
  WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : <n> };
```

Borrar un *keyspace*

Para borrar un *keyspace* utilizamos la sentencia `DROP KEYSPACE`:

```
DROP KEYSPACE <keyspace_name>;
```

Creación de una tabla

Para crear una tabla utilizamos la sentencia `CREATE TABLE`:

```
CREATE TABLE <table_name> [     IF NOT EXISTS ] (
  <column_name> <type> PRIMARY KEY,
  <column_name> <type>,
  ...
);
```

Veamos en detalle con un ejemplo:

```
CREATE TABLE monkey_species (
    species text PRIMARY KEY,
    common_name text,
    population varint,
    average_weight float,
    average_height float
) WITH comment = 'Tabla que almacena información sobre especies de monos';
```

En este ejemplo hemos creado una tabla llamada `monkey_species` con las siguientes columnas:

- `species`: clave primaria de tipo `text`.
- `common_name`: columna de tipo `text`.
- `population`: columna de tipo `varint`.
- `average_weight`: columna de tipo `float`.
- `average_height`: columna de tipo `float`.

También hemos añadido un comentario a la tabla.

Otro ejemplo algo más complejo:

```
CREATE TABLE timeline (
    user_id uuid,
    posted_month int,
    posted_time timeuuid,
    body text,
    posted_by text,
    PRIMARY KEY (user_id, posted_month, posted_time)
) WITH COMPACTION = { 'class' : 'LeveledCompactionStrategy' };
```

En este ejemplo hemos creado una tabla llamada `timeline` en la que definimos la clave primaria como una clave primaria compuesta por tres columnas:

- `user_id`: columna de tipo `uuid`.
- `posted_month`: columna de tipo `int`.
- `posted_time`: columna de tipo `timeuuid`.

No hemos especificado cual es la partition key y cual es la clustering key. Si queremos que `user_id` sea la partition key y `posted_month` y `posted_time` sean las clustering keys deberíamos haber definido la clave primaria de la siguiente forma:

```
PRIMARY KEY ((user_id) posted_month, posted_time)
```

al encerrar una o más columnas entre paréntesis indicamos que son la partition key. En este caso `user_id` es la partition key y `posted_month` y `posted_time` son las clustering keys.

Si tenemos una clustering key compuesta por varias columnas podemos indicar también la ordenación de las mismas. Por ejemplo, si queremos que `posted_month` sea descendente y `posted_time` ascendente deberíamos haber definido la clave primaria de la siguiente forma:

```
PRIMARY KEY ((user_id) posted_month, posted_time ) WITH CLUSTERING ORDER BY
(posted_month DESC, posted_time ASC)
```

Si hay alguna columnas cuyos valores no van a cambiar podemos indicarlo con el modificador `static`. Por ejemplo, si queremos que `posted_by` sea una columna estática deberíamos haber definido la tabla de la siguiente forma:

```
CREATE TABLE timeline (
    user_id uuid,
    posted_month int,
    posted_time timeuuid,
    body text,
    posted_by text STATIC,
    PRIMARY KEY (user_id, posted_month, posted_time)
) WITH COMPACTION = { 'class' : 'LeveledCompactionStrategy' };
```

Nótese que: Una **primary key ha de ser única** pero ni la PK (partition key) ni la CK (clustering key) han de ser únicas por separado.

Modificar una tabla

Para modificar una tabla utilizamos la sentencia `ALTER TABLE`:

```
ALTER TABLE <table_name>
  ADD <column_name> <type>,
  DROP <column_name>,
  ALTER <column_name> TYPE <type>,
  RENAME <column_name> TO <new_column_name>,
  WITH <option> = <value>,
  ...
```

Un ejemplo sería:

```
ALTER TABLE monkey_species
  ADD average_lifespan int,
  DROP average_height,
  ALTER average_weight TYPE float,
  RENAME common_name TO name,
  WITH comment = 'Tabla que almacena información sobre especies de monos';
```

Si quisiésemos eliminar todos los registros de una tabla podemos utilizar la sentencia `TRUNCATE`:

```
TRUNCATE [ TABLE ] <table_name>;
```

Para eliminar una tabla utilizamos la sentencia `DROP TABLE`:

```
DROP TABLE [ IF EXISTS ] <table_name>;
```

Ver la definición de una tabla

Para ver la definición de una tabla utilizamos la sentencia `DESCRIBE` indicando a qué tabla nos referimos:

```
DESCRIBE TABLE <table_name>;
```

Por ejemplo, para ver la estructura de la tabla `sbd.miembros`:

```
DESCRIBE TABLE sbd.miembros;
```

Importar datos ds CSV a una tabla

```
COPY keypsace.tableName (col1,col2,col3.....) FROM 'file/file.csv' WITH
DELIMITER=',' AND HEADER=TRUE;
```

`IF NOT EXISTS`

La utilidad de esta cláusula es evitar que se produzca un error si intentamos crear un *keyspace* o una tabla que ya existe. Si la cláusula `IF NOT EXISTS` está presente y el *keyspace* o la tabla ya existen, la sentencia no tiene ningún efecto.