**CASSANDRA**

Cassandra es una base de datos NoSQL. No se podrá usar Join. Es una base de datos clave-valor. En cassandra se creará una tabla por cada consulta que se quiera hacer.

Sus principales características son:

* Es un sistema descentralizado: todos los nodos tienen el mismo rol, no existe un nodo maestro.
* Replicación: en el clúster y en múltiples datacenters.
* Escalabilidad lineal: las lecturas y escrituras aumentan a medida que se agregan más nodos al clúster.
* Tolerancia a fallas: la información puede estar en más de un nodo (redundancia) y en múltiples centros de datos.
* Consistencia: posibilidad de obtener el registro con la fecha de grabación de un nodo, del clúster o de todos los datacenter.

Una base de datos en Cassandra es representada por una keyspace, que almacenará las tablas. Para crear las tablas será necesario definir la clave partición y la clave clúster:

* Clave de partición: su función es definir el nodo en donde se almacenará el registro. Esto lo hace utilizando una función hash.
* Clave de clúster: Una vez definido el nodo en que se almacenará el registro, el siguiente paso será guardarlo de acuerdo al orden definido por el usuario, la clave clúster idealmente debe de ser una marca de tiempo (timestamp).

Por ejemplo, supongamos un clúster de 10 nodos utilizado para almacenar 1GB de información cada día. Nuestra aplicación es Twitter y cada Tweet tendrá: el id del usuario, la marca de tiempo y el texto. Esta tabla la podemos definir:

*CREATE TABLE tweet (*

*usuario bigint,*

*timestamp timestamp,*

*tweet text,*

*PRIMARY KEY (usuario, timestamp)*

*) WITH CLUSTERING ORDER BY (timestamp DESC);*

En la tabla anterior el id de usuario será la clave partición, esto significa que cuando un nodo reciba un registro para escribir aplicará la siguiente función hash(usuario) = {1, 2, 3, …, 10}, es decir que los posibles valores serán del 1 al 10 (nodos en el clúster). De esta manera todos los registros de un usuario estarán en un solo nodo.

La clave de clúster (timestamp) es la que se encarga de ordenar los registros en el nodo, en este caso de manera descendiente (el más reciente de primero).

Cada nodo Cassandra almacenará los registros pertenecientes a diferentes claves de partición (en nuestro ejemplo pueden haber 10 nodos pero 10,000 usuarios). Entonces, Cassandra define el almacenamiento por clave partición en forma horizontal.

Para describir esto, a nuestra tabla ejemplo le vamos a agregar más campos para tener un mejor detalle de este concepto:

*CREATE TABLE tweet (*

*usuario bigint,*

*timestamp timestamp,*

*tweet text,*

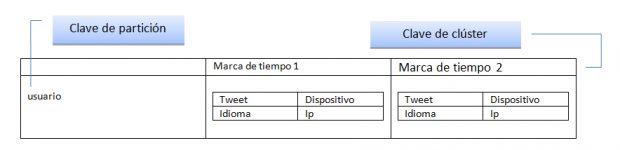
*dispositivo text,*

*idioma text,*

*ip text,*

*PRIMARY KEY (usuario, timestamp)*

*) WITH CLUSTERING ORDER BY (timestamp DESC);*



En el ejemplo anterior observamos que para cada clave partición (usuario) se creará una fila con sus tweets, ordenados de forma descendiente definidos por la clave clúster (timestamp). En este caso los valores de cada celda serán los campos tweet, dispositivo, idioma e ip.

Cada tabla tendrá entones una fila con N registros para cada usuario. La capacidad máxima de registros por fila teóricamente es de 2 billones.

Para optimizar Cassandra, es necesario agregar más valores a la clave de partición, esto lo podemos hacer separando cada usuario por mes (lógicamente a cada mes le debe preceder un año):

*CREATE TABLE tweet (*

*usuario bigint,*

*year int,*

*month int,*

*timestamp timestamp,*

*tweet text,*

*dispositivo text,*

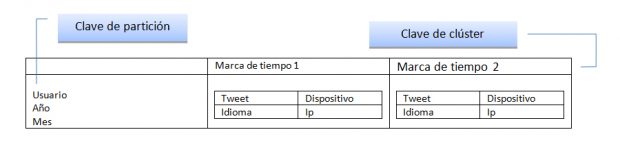
*idioma text,*

*ip text,*

*PRIMARY KEY ((usuario, year, month), timestamp)*

*) WITH CLUSTERING ORDER BY (timestamp DESC);*

La clave de partición (usuario, año, mes) crea diferentes filas de registros por cada usuario/año/mes. La tabla ahora será de la siguiente forma:



La función hash de la clave partición será: hash(usuario, año, mes) = {1,2,3,…,10}.

Para poder consultar la información es obligatorio definir las claves partición y la clave clúster (si la tabla las usa). Si al final de la consulta ponemos ALLOW FILTERING, nos va a permitir no definir la primary key, pero es menos eficiente.

En el siguiente ejemplo para obtener los tweets enviados por un usuario en los últimos 10 minutos debemos especificar el usuario, año, mes y marca de tiempo:

SELECT \* FROM tweet WHERE usuario = 100 AND year = 2018 AND month = 1 AND timesteamp > ‘2018-01-28 20:12:00’ AND timestamp <= ‘2018-01-28 20:22:00’;

**El Modelo de Datos de Cassandra**

**Keyspace**

Un keyspace puede ser visto como el contenedor más exterior para datos en Cassandra. Todos los datos en Cassandra deberían vivir dentro de un keyspace. Esto puede ser como una base de datos en el Sistema de gestión de bases de datos relacionales, que es una colección de tablas. En el caso de Cassandra, un keyspace es una colección de familias de columnas.

*CREATE KEYSPACE personas*

*WITH REPLICATION = {*

*'class' : 'SimpleStrategy',*

*'replication\_factor' : 2*

*} ;*

* **Replication Factor**: Por ejemplo, un factor de replicación de 2 nos garantiza que habrá dos copias de la información de los nodos en el clúster. Escoger el valor dos para el factor de replicación está bien para cubrir fallas en un nodo en el ambiente de desarrollo y puede ser lo mínimo para ambientes de producción, ya que si un nodo cae el otro se encargará de manejar todas las peticiones, así que es algo que debemos pensar con detenimiento antes de implementarlo.
* **Replication Strategies:**
  + Simple: más simple. Se encarga de copiar la información al siguiente nodo hasta que se cumpla el factor de replicación definido. Selecciona el primer nodo para la primera réplica de datos y coloca la segunda réplica en el siguiente nodo en el anillo.
  + Network Topology Strategy: es más inteligente, es la mejor opción si deseamos distribuir la información a lo largo de múltiples centros de datos.

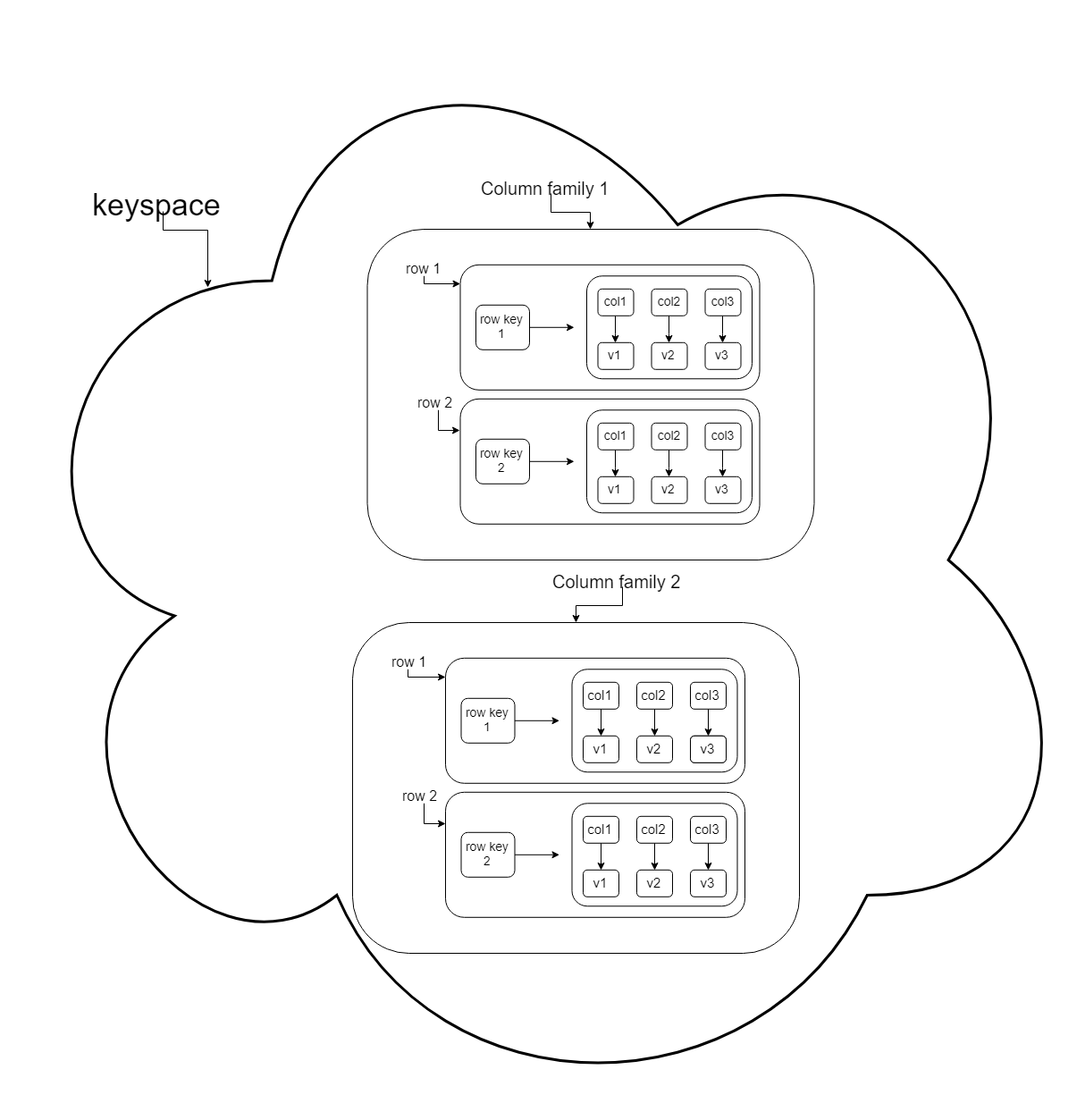
**Familia de Columnas**

Una familia de columnas puede ser vista como una colección de filas, y cada fila es una colección de columnas. Es análoga a una tabla en RDBMS pero tiene algunas diferencias. Las familias de columnas son definidas, pero no es necesario para cada fila tener todas las columnas, y las columnas pueden ser añadidas o eliminadas de una fila cuando se requiera.

**Columna**

La columna es la unidad básica de datos en Cassandra.

La siguiente imagen muestra todos estos componentes.



**Filas**: un conjunto de filas constituye una familia de columna.

**Clave fila (clave de partición)**: identifica a una fila en una familia de columna.

**Fila**: almacena pares de claves y valores de columnas.

**Clave de columna**: identifica un valor de columna en una fila.

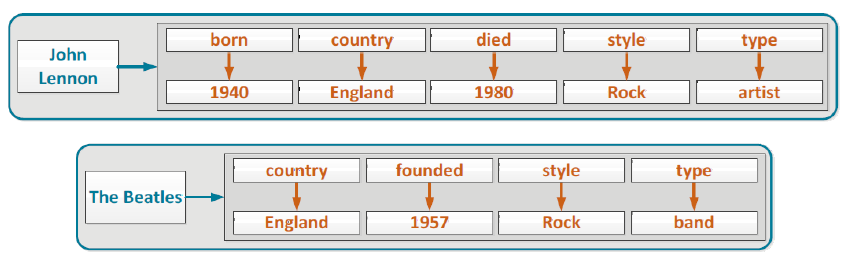
**Valor de columna**: almacena un valor o colección de valores.

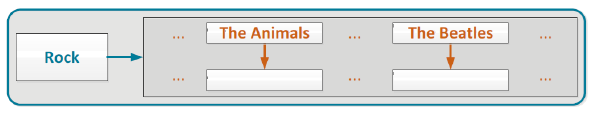
Las claves de columnas se ordenan por defecto.

Una fila puede ser recuperada si se conoce su clave. Igualmente, un valor de columna puede ser recuperado si se conoce su clave de fila y su clave de columna.

Una fila puede ser descrita como:

* “Skinny row”: tamaño casi fijo y un número relativamente pequeño de columnas.



* “Wide row”: tamaño relativamente grande de clave de columnas (100, 1000), este número puede ir aumentando cuando se insertan nuevos datos.

**Partition-key, Composite-key y Clustering-key**

**Partition Key**

La Partition Key no es otra cosa que la “primera parte” de la Primary key e identifica de forma unívoca a una partición. Por otro lado, la Primary key puede ser simple (una única columna) o estar compuesta por más de una columna (Composite Key). En el caso de una tabla en la que la Primary key es simple, la Partition Key es la única columna que existe en la clave primaria, por lo tanto, coinciden Primary key y Partition Key.

La Partition Key es especialmente importante en Cassandra, ya que es la clave responsable de distribuir los datos a lo largo del total de nodos disponibles.

Ejemplo 1: como ejemplo básico, podríamos modelar una tabla de oficinas identificadas de manera unívoca por su código de oficina utilizando una Primary key simple, que coincidiría con la Partition Key, de la siguiente manera:

*create table offices\_by\_office\_code (*

*office\_code int Primary key,*

*country\_code text,*

*state\_province text,*

*city text,*

*address text*

*);*

**Composite Key (o Compound Key)**

Como ya hemos visto, la Primary key puede tener más de una columna, en estos casos hablamos de Composite Key (o Compound Key). Una característica interesante de la Composite Key es que únicamente la “primera parte” es considerada como la Partition Key.

Continuando con nuestro ejemplo anterior, inicialmente teníamos una tabla en la que identificamos las oficinas por su código de oficina (columna “office\_code”), que era precisamente la Partition Key. Por lo tanto, los datos se iban a distribuir entre todos los nodos utilizando este campo. Esto podría significar un problema si queremos realizar una búsqueda de oficinas por país, ya que la consulta requiere acceder a todos los nodos para buscar oficinas de un país en concreto. La solución pasa por modelar una tabla de forma que tengamos una Partition Key que nos asegure una correcta distribución de los datos entre todos los nodos, y que incluya los campos por los que vamos a realizar la búsqueda en el orden apropiado para la consulta.

Ejemplo 2: podríamos modelar una tabla para almacenar las distintas oficinas físicas de las que dispone una empresa en todo el mundo. En este caso tendríamos una Composite Key, pero la Partition Key sería únicamente las columnas “country\_code” y “state\_province”, que tendría sentido si la empresa cuenta con aproximadamente el mismo número de oficinas en cada ubicación. Esta solución exige que si queremos consultar una oficina en concreto por su “office\_code”, necesitamos conocer también su “country\_code”, su “state\_province” y su “city”:

*create table offices\_by\_location (*

*country\_code text,*

*state\_province text,*

*city text,*

*office\_code text,*

*address text,*

*Primary key ((country\_code, state\_province), city, office\_code)*

*);*

En la tabla “offices\_by\_location”, y asumiendo que no se han definido otros índices, las posibles consultas que se podrían realizar serían aquellas que utilicen las columnas…

* “country\_code” AND “state\_province”
* “country\_code” AND “state\_province” AND “city”
* “country\_code” AND “state\_province” AND “city” AND “office\_code”

Y algunos ejemplos de consultas que no se podrían realizar…

* En general, cualquiera que no incluya “country\_code” AND “state\_province” (es decir, la Partition Key)
* “country\_code” AND “state\_province” AND “office\_code” (porque no respeta el orden de columnas que se ha definido en la Primary key)

**Clustering Key**

Cuando alguna de las consultas que vamos a realizar implica obtener los datos ordenados ascendentemente o descendentemente por alguna de las columnas entra en juego la Clustering Key. Ya hemos visto que la “primera parte” de una clave primaria compuesta (Composite Key) es la Partition Key. Pues bien, la Clustering Key la conforman el resto de columnas que forman parte de la Primary key pero no de la Partition Key, es decir, la “segunda parte” de la Primary key.

Si echamos un vistazo al ejemplo 2, es fácil identificar que en este caso nuestra Clustering Key está formada por las columnas “city” y “office\_code”, es decir, dentro de cada partición Cassandra está ordenando los campos por estas dos columnas.

Por lo tanto, utilizando el modelo de tabla del ejemplo 2 podríamos lanzar una consulta para obtener todas las oficinas de una ubicación ordenadas por ciudad:

SELECT \* FROM offices\_by\_location WHERE country\_code = 'ES' AND state\_province = '28' ORDER BY city;

Pero por ejemplo no podríamos utilizar la siguiente consulta para obtener todas las oficinas de una ciudad ubicación ordenadas por su dirección:

SELECT \* FROM offices\_by\_location WHERE country\_code = 'ES' AND state\_province = '28' AND city = 'Madrid' ORDER BY address; (MAL)

Resumen de estos tres conceptos:

* La Primary key es equivalente a la Partition Key en las tablas cuya clave primaria esta formada por una única columna.
* La Composite Key (o Compound Key) es simplemente una Primary key formada por varias columnas.
* La Partition Key identifica de forma unívoca a una partición
* La Partition Key es la clave responsable de distribuir los datos a lo largo del total de nodos disponibles.
* La Clustering Key es la clave responsable de ordenar los datos dentro de una misma partición
* La Partition Key y la Clustering Key pueden estar compuestas por más de una columna

**INSTALACIÓN**

* Instalar DataStax DevCenter -> https://academy.datastax.com/planet-cassandra/cassandra
* Crear conexión. Crearemos conexion en LocalHost, protocol port: 9042
* Si da error.
  + Seguir los pasos de la siguiente página -> https://jonathanmelgoza.com/blog/como-instalar-y-configurar-apache-cassandra-en-windows/
    - Comprobar: data\_file\_directories, commitlog\_directory y la variable de entorno JAVA\_HOME señalando a la carpeta jdk
  + Hacer los siguientes pasos -> https://stackoverflow.com/questions/39893193/cassandra-cql-shell-window-got-disappears-after-installation-in-windows
    - Step 1: Open file: DataStax-DDC\apache-cassandra\conf\cassandra.yaml
    - Step 2: Uncomment the cdc\_raw\_directory and set new value to (for windows)

cdc\_raw\_directory: "C:/Program Files/DataStax-DDC/data/cdc\_raw"

* + - Step 3: Goto Windows Services and Start the DataStax DDC Server 3.9.0 Service
* Realizar de nuevo el test de conexión, ya debe funcionar.
* (nose si debe de estar bien configurado las variables de entorno java\_home y path)
* El servicio DataStax debe estar iniciado.
* Ya debe funcionar DataStax y y CQL SHELL