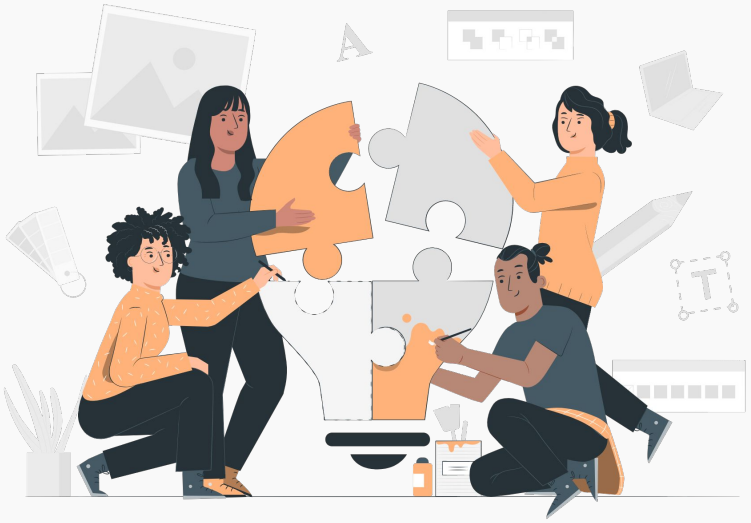
BASES DE DATOS ORIENTADAS A COLUMNAS

Mario Julián Mora
mario.mora@u.icesi.edu.co

Mónica Rojas
mmrojas@icesi.edu.co



Unidad 4 – Bases de datos NoSQL

UNIVERSIDAD ICESI

- Su precursor fue Google BigTable

- **BigTable** (column oriented) es un sistema de gestión de base de datos creado por Google distribuido, de alta eficiencia y propietario.

- Tablas multidimensionales
- Se divide en columnas -> tabletas
- Cada tableta 200 Mb
- En cada nodo 100 tabletas

BD ORIENTADAS A COLUMNAS

- 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
  - Alta disponibilidad
  - MapReduce

DB ENGINES RANKING

416 systems in ranking, November 2023

Rank Nov 2023	Rank Oct 2023	Rank Nov 2022	DBMS	Database Model	Score Nov 2023	Score Oct 2023	Score Nov 2022
1.	1.	1.	Oracle	Relational, Multi-model	1277.03	+15.61	+35.34
2.	2.	2.	MySQL	Relational, Multi-model	1115.24	-18.07	-90.30
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	911.42	+14.54	-1.09
4.	4.	4.	PostgreSQL	Relational, Multi-model	636.86	-1.96	+13.70
5.	5.	5.	MongoDB	Document, Multi-model	428.55	-2.87	-49.35
6.	6.	6.	Redis	Key-value, Multi-model	160.02	-2.95	-22.03
7.	7.	7.	Elasticsearch	Search engine, Multi-model	139.62	+2.48	-10.70
8.	8.	8.	IBM Db2	Relational, Multi-model	136.00	+1.13	-13.56
9.	9.	↑ 10.	SQLite	Relational	124.58	-0.56	-10.05
10.	10.	↓ 9.	Microsoft Access	Relational	124.49	+0.18	-10.53
11.	11.	↑ 12.	Snowflake	Relational	121.00	-2.24	+10.84
12.	12.	↓ 11.	Cassandra	Wide column, Multi-model	109.17	+0.34	-8.96
13.	13.	13.	MariaDB	Relational, Multi-model	102.09	+2.43	-2.82

- https://db-engines.com/en/ranking

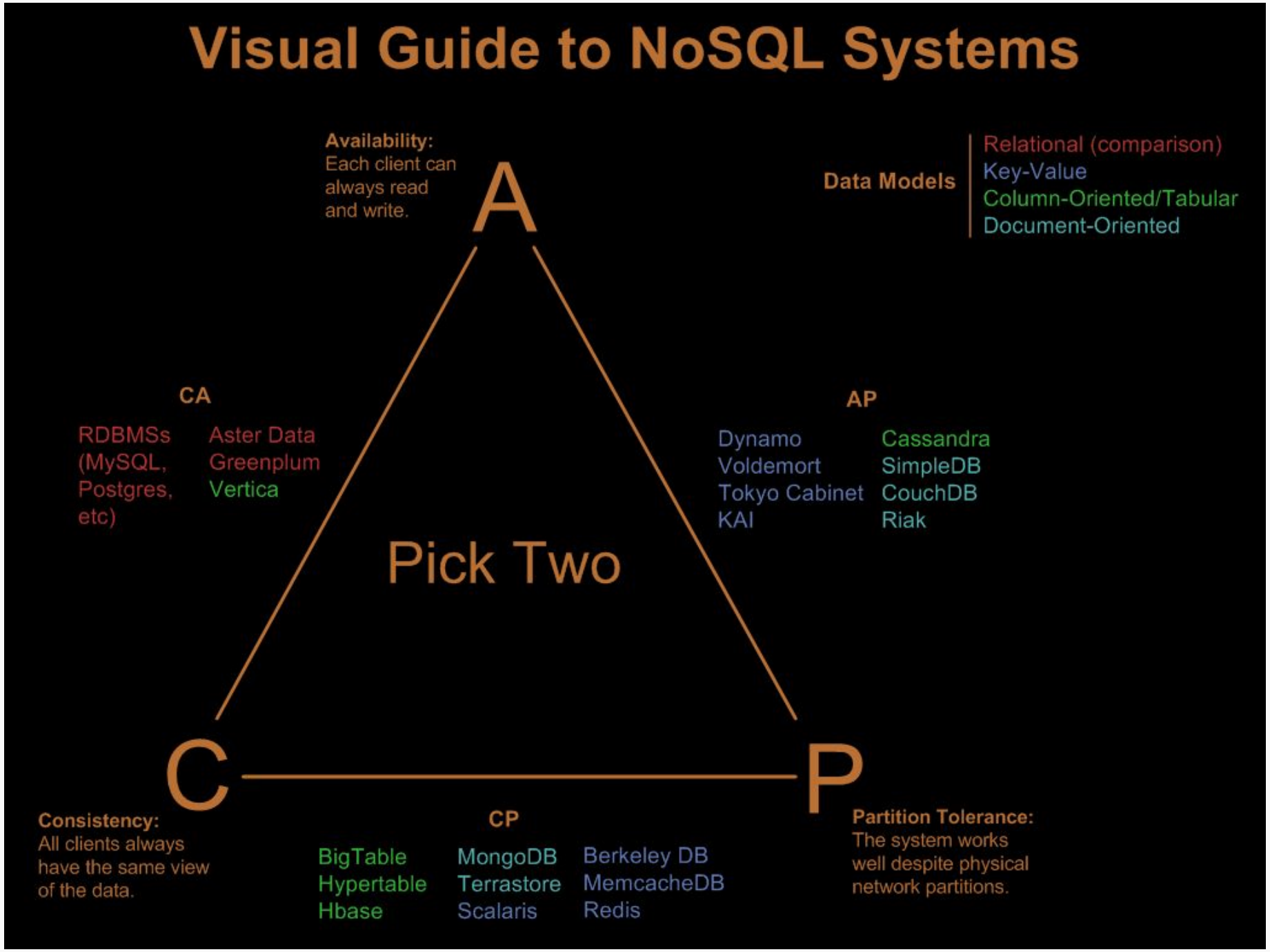
DB RANKING – WIDE COLUMN

☐ include secondary database models — 13 systems in ranking, November 2023

Rank Nov 2023	Rank Oct 2023	Rank Nov 2022	DBMS	Database Model	Score Nov 2023	Score Oct 2023	Score Nov 2022
1.	1.	1.	Cassandra	Wide column, Multi-model	109.17	+0.34	-8.96
2.	2.	2.	HBase	Wide column	34.16	-0.53	-6.25
3.	3.	3.	Microsoft Azure Cosmos DB	Multi-model	34.11	-0.18	-5.64
4.	4.	4.	Datastax Enterprise	Wide column, Multi-model	7.26	+0.48	-1.42
5.	5.	↑ 8.	ScyllaDB	Wide column, Multi-model	6.56	+0.38	+1.61
6.	6.	↓ 5.	Microsoft Azure Table Storage	Wide column	6.08	+0.31	+0.06
7.	7.	↓ 6.	Google Cloud Bigtable	Multi-model	4.26	-0.35	-0.97
8.	8.	↓ 7.	Accumulo	Wide column	4.14	-0.47	-0.84
9.	9.	9.	HPE Ezmeral Data Fabric	Multi-model	1.24	-0.07	-0.02
10.	10.	10.	Amazon Keyspaces	Wide column	0.98	-0.13	+0.21
11.	11.	11.	Elassandra	Wide column, Multi-model	0.52	-0.01	-0.05
12.	12.	12.	Alibaba Cloud Table Store	Wide column	0.26	+0.04	-0.07
13.	13.	13.	SWC-DB	Wide column, Multi-model	0.05	+0.04	-0.02

- https://db-engines.com/en/ranking

Visual Guide to NoSQL Systems



MODELO DE DATOS

- Es una tripleta (nombre de columna, valor, marca de tiempo)

- Cada columna en Cassandra posee una marca horaria, la cual graba la última fecha en que la columna fue actualizada, la marca es de las columnas y no de los registros. Este dato no puede consultarse, se usa únicamente para resolución de conflictos en el servidor.

MODELO DE DATOS

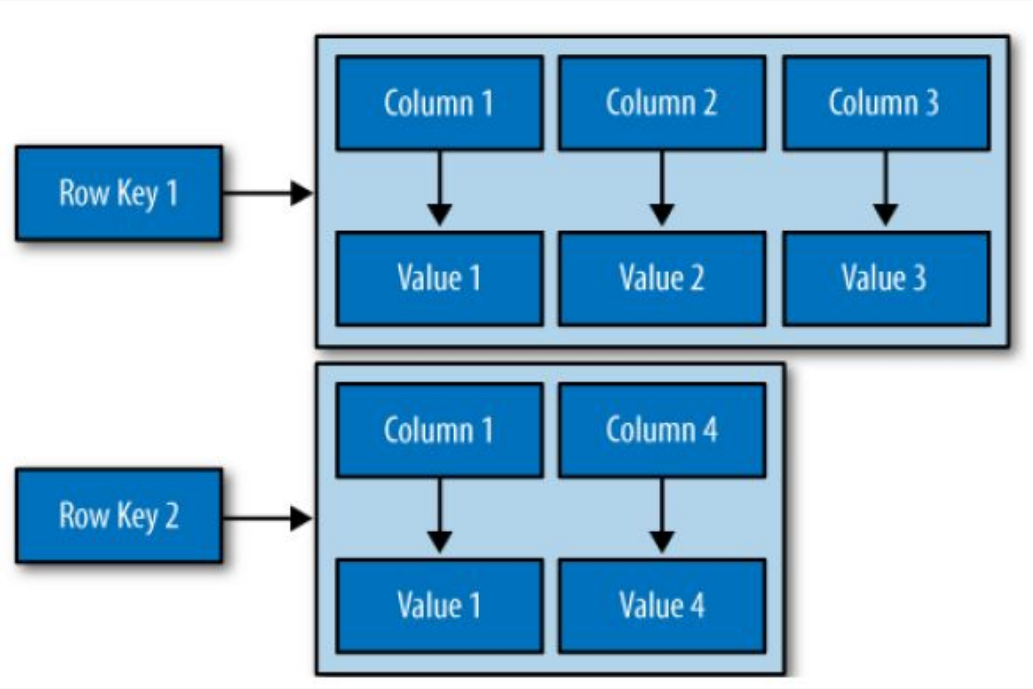
- Cassandra define una familia de columnas para asociar datos similares.
- Por ejemplo, se puede tener una familia de columnas llamada Usuario, otra llamada Hotel, otra llamada Registro y muchas más.
- Una familia de columnas es análoga a una tabla en el modelo relacional.

MODELO DE DATOS

No se necesita almacenar un valor para cada columna si no se tiene información.

Por ejemplo, algunas personas tienen un segundo número de teléfono y otras no.

En vez de colocar NULL para los valores que no conocemos, lo cual gasta espacio, simplemente no se tiene en cuenta la columna para ese registro. Con esto se tiene una estructura de arreglos multidimensional como esta:



CUANDO UTILIZARLAS

- Soluciones de tipo analítico

- ¿Por qué?

Porque al tener almacenada la información de las columnas junta, se reduce notablemente los requisitos globales de E/S del disco, y disminuye el volumen de datos que hay que cargar desde él.

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: 3.11(2019-02-11)
  - Web: http://cassandra.apache.org/download/
  - Documentación: https://docs.datastax.com/en/landing_page/doc/landing_page/cassandra.html

VENTAJAS CASSANDRA

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

DESVENTAJAS CASSANDRA

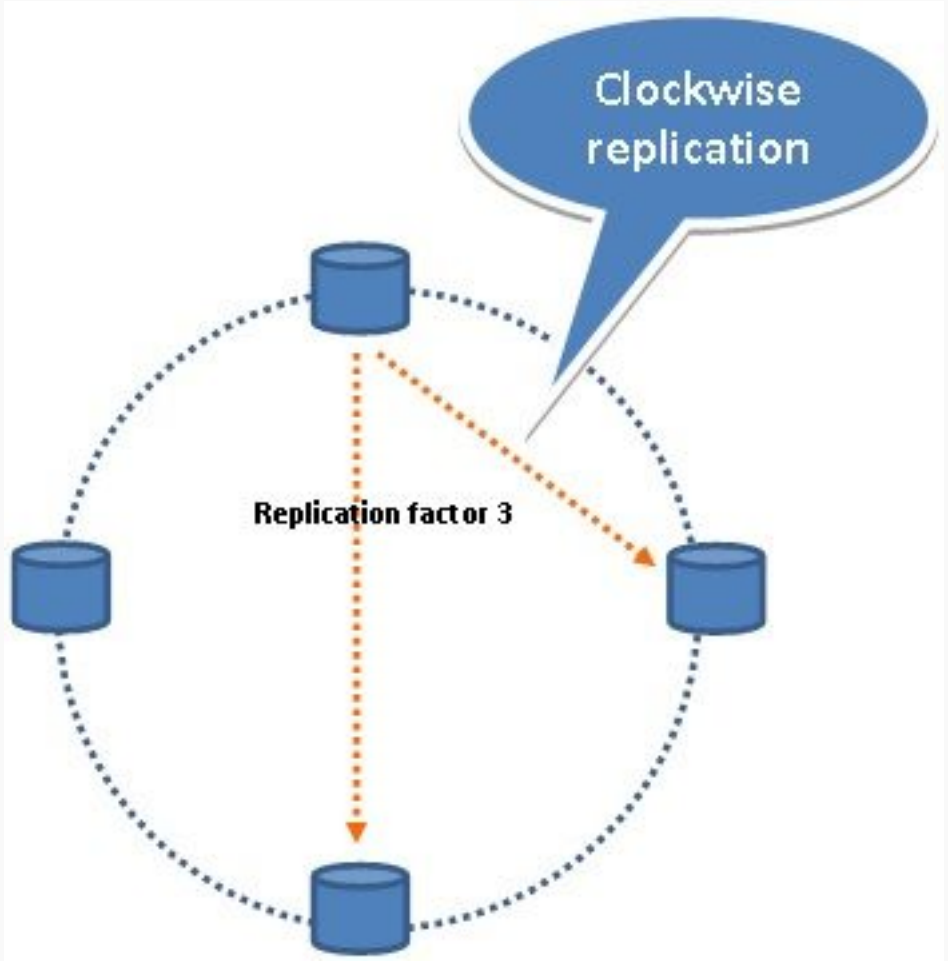
- Hay algunas desventajas que un sistema de almacenamiento tan escalable ofrece en contrapartida:
  - No hay joins (a cambio de más velocidad)
  - No permite ordenar resultados en tiempo de consulta
  - No tiene SQL
    - Pero desde la versión 0.8 se tiene CQL

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 un único nodo.
  - La estructura más externa de Cassandra es el clúster 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.

CASSANDRA ARCHITECTURE- SIMPLE STRATEGY

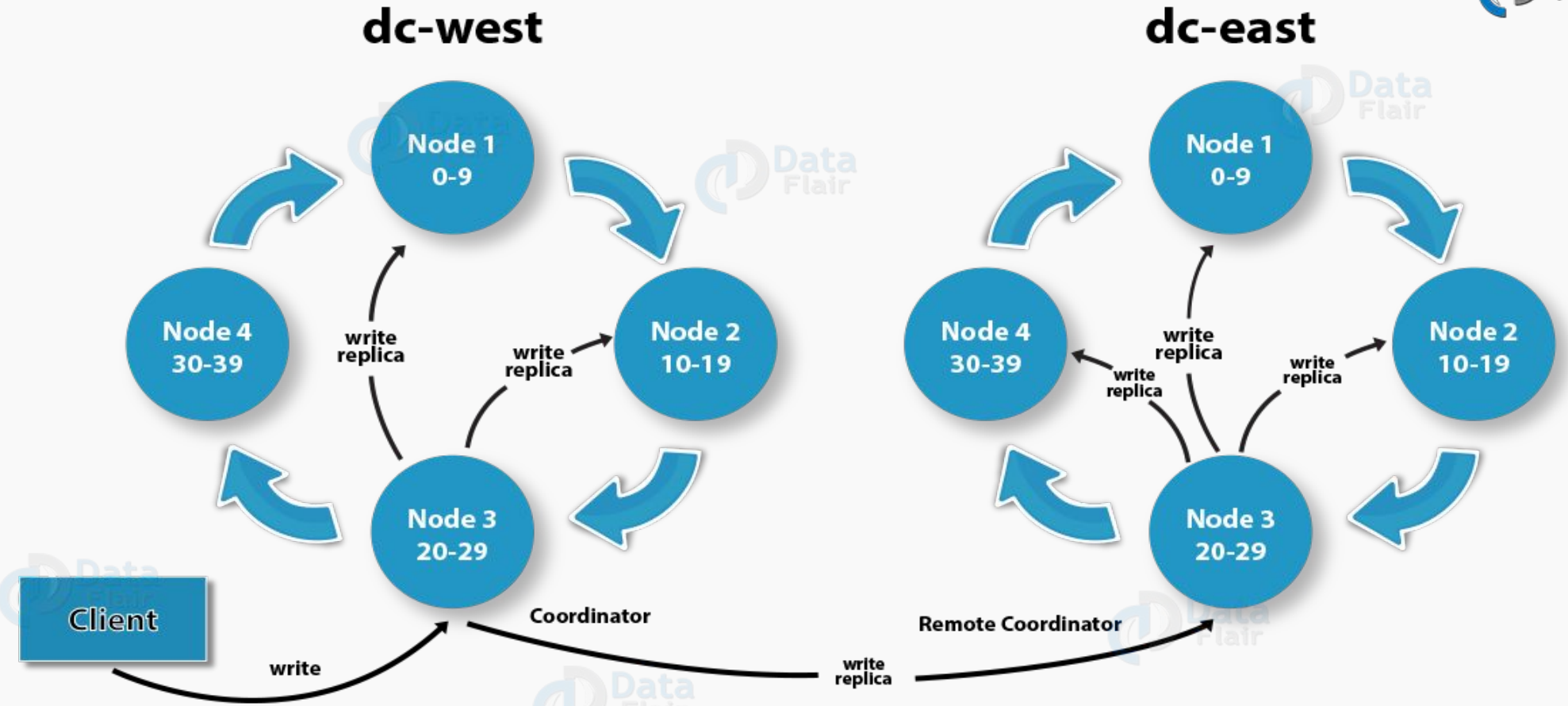
Guarda las copias en un solo datacenter.



https://docs.datastax.com/en/cassandra-oss/3.0/cassandra/architecture/archDataDistributeReplication.html

CASSANDRA ARCHITECTURE- NETWORK TOPOLOGY STRATEGY

Se utiliza para diferentes datacenters.



https://data-flair.training/blogs/cassandra-architecture/

TEOREMA CAP EN CASSANDRA

- Centradas en disponibilidad y tolerancia a fallos
- Permiten Consistencia Eventual

  (Donde “eventual” significa milisegundos)

- 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”

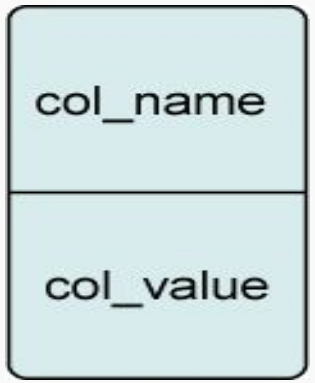
MODELO DE DATOS EN CASSANDRA

- 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 asociarse con una BD en el modelo relacional)
  - **ColumFamily**: contienen varias columnas
    (Suelen asociarse con una tabla en el modelo relacional)
  - **SuperColumn**: columnas que ellas mismas tienen sub-columnas
  - **Column**: compuestas de un nombre, valor y timestamp

COLUMN

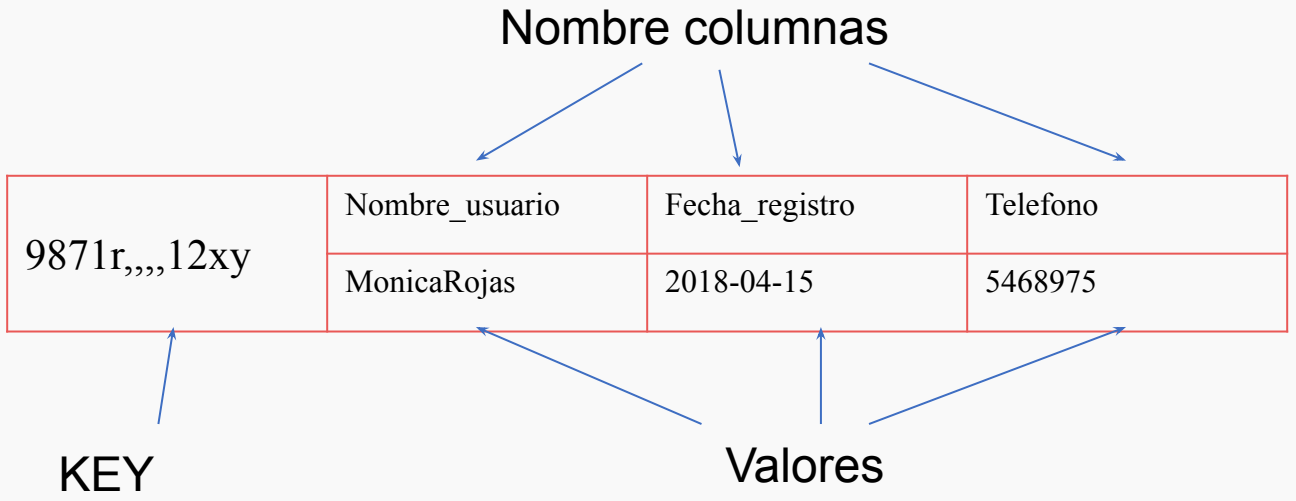
- Una **columna** es un par nombre-valor que también contiene un timestamp
  - Los nombres y columnas son arrays de bytes
  - El timestamp registra la última vez que una columna es accedida
  - Unidad atómica
    - `name:value:timestamp`
    - `Email:monroj@icesi.edu.co:123456789`
- Ejemplo en JSON:

```
{
   "name": "Email",
   "value": "monroj@icesi.edu.co",
   "timestamp”: 123456789
}
```



FILA

- Agregación de columnas con un nombre para referenciarlo.
- Equivale a la fila en el modelo relacional.



FILA

- Tiene un ROW Key, o clave primaria de las filas.

Dos partes:

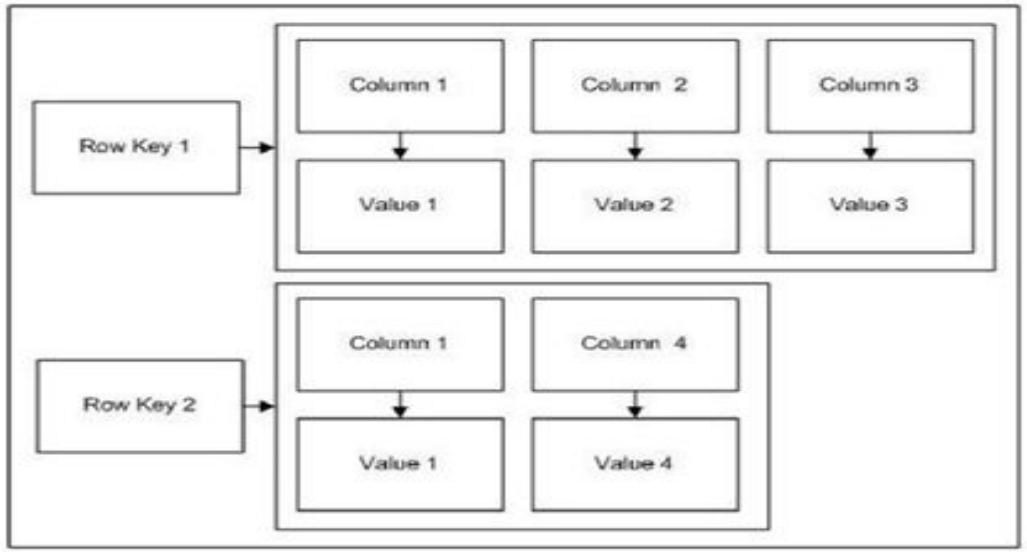
- Partition key (de particionado): las filas con el mismo valor de clave de particionado se almacenan en la misma partición del disco (físicamente juntas).
- Clustering key (de agrupamiento): determina el orden físico en el que se almacenan las filas.

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.
    - ASCII, UTF-8, Long, UUID
- Cada familia de columnas se guarda en una partición, y la partición está ordenado por la clustering key.
- Una familia de columnas tiene un …

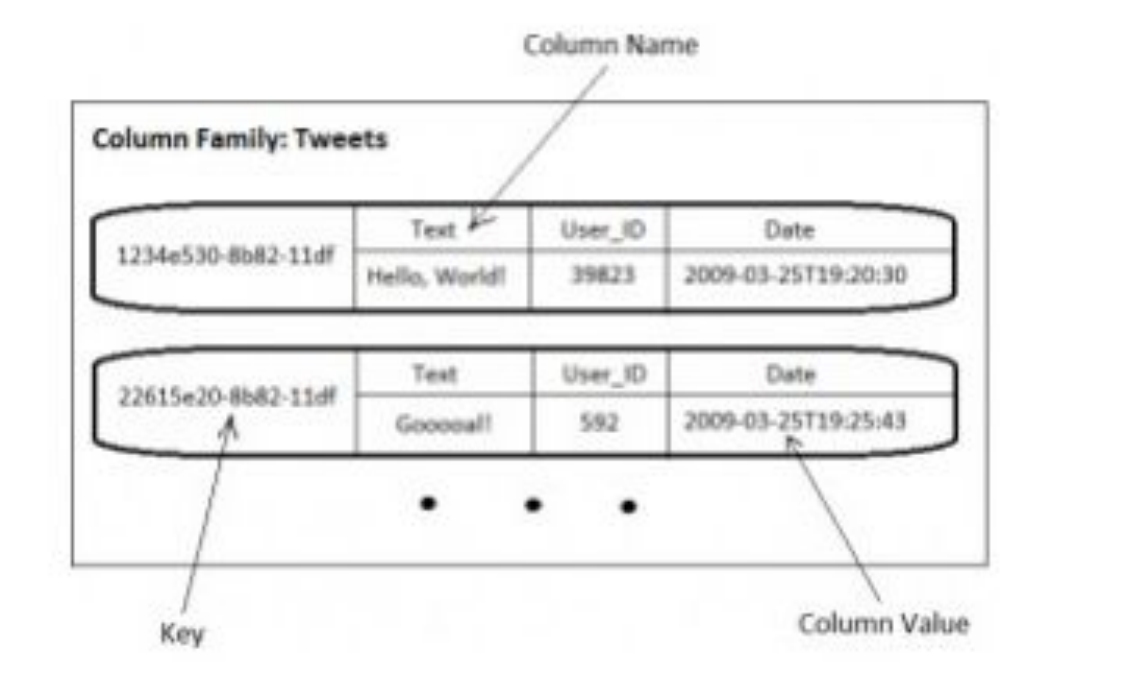
  conjunto de filas con un conjunto de …

  columnas similar pero no idéntico



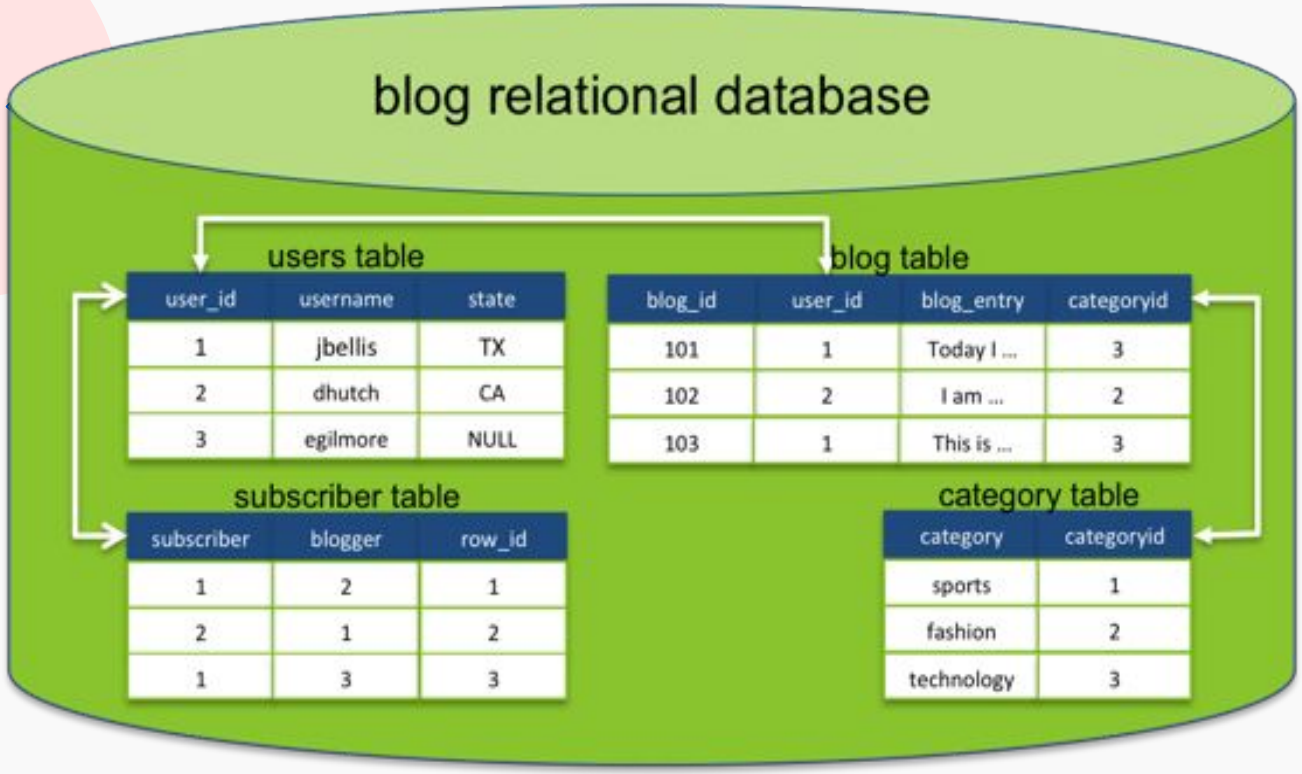
KEYSPACES

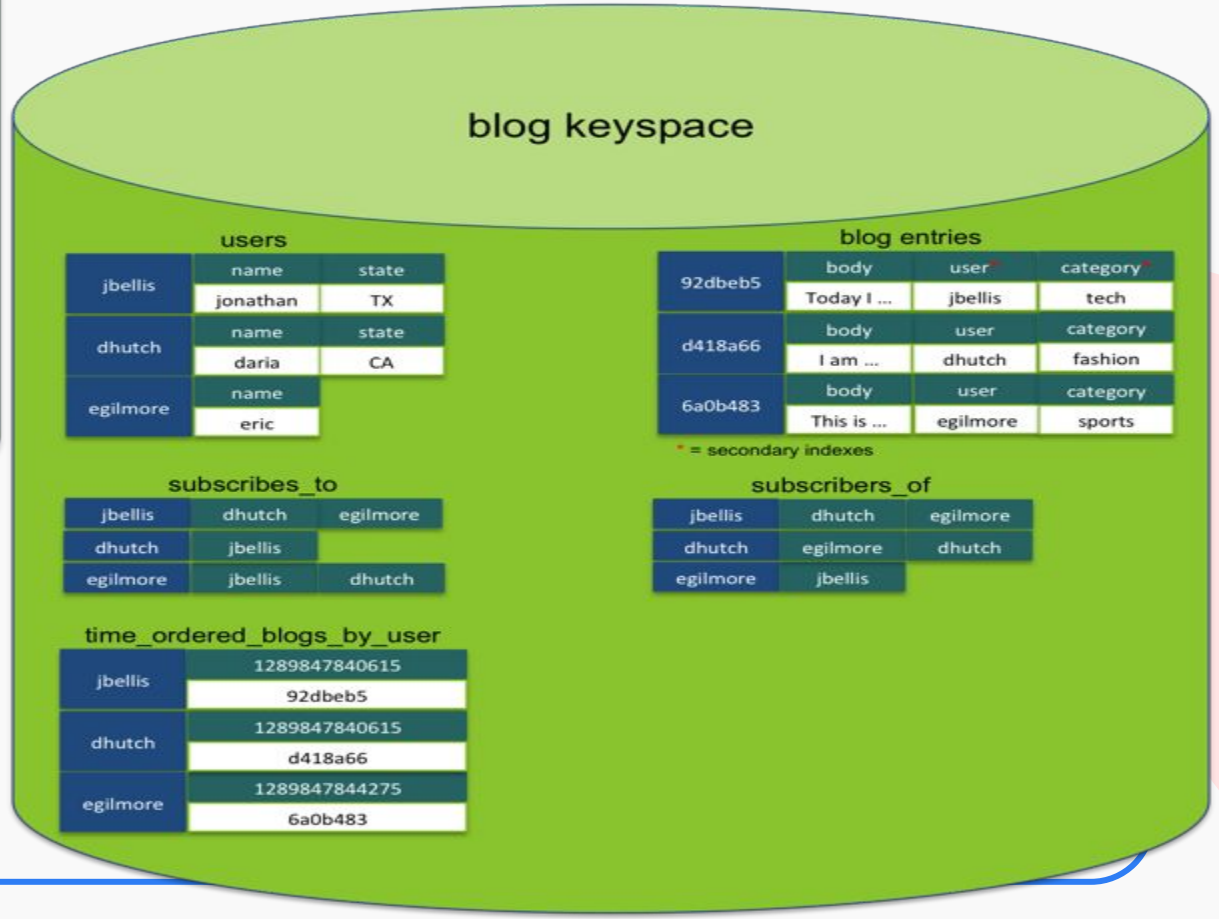
Un espacio de claves o KeySpace es un esquema de alto nivel que contiene familias de columnas.



Column Family “Entradas en Twitter: tweets”:

RDBS VS. KEYSPACE





COMPARACIÓN

Modelo Relacional	Modelo Cassandra
Base de datos	Keyspace
Tabla	Column Family (CF)
Primary key	Row key
Atributo	Column name /key
Valor	Column value

INSTALACIÓN DE CASSANDRA

- Documentación en:
  https://cassandra.apache.org/doc/latest/cassandra/getting_started/index.html

  - Requisitos:
    - Java.
    - Python.
- Las últimas versiones estables disponibles en:
  - http://cassandra.apache.org/download/

AMBIENTES ALTERNATIVOS

Instalación versión de Datastax:

https://www.datastax.com/blog/getting-started-apache-cassandra-windows-easy-way

Datastax Community Edition (Win)

https://datastax-community-edition.software.informer.com/2.2/

En línea, creando un espacio en Astra:

https://astra.datastax.com/register

Pasos para la configuración [link]

Data Flair

Cassandra

Data Definition Command

KEYSPACE

A. CREATE KEYSPACE

B. USE

C. ALTER KEYSPACE

D. DROP KEYSPACE

E. CREATE TABLE

F. ALTER TABLE

G. TRUNCATE TABLE

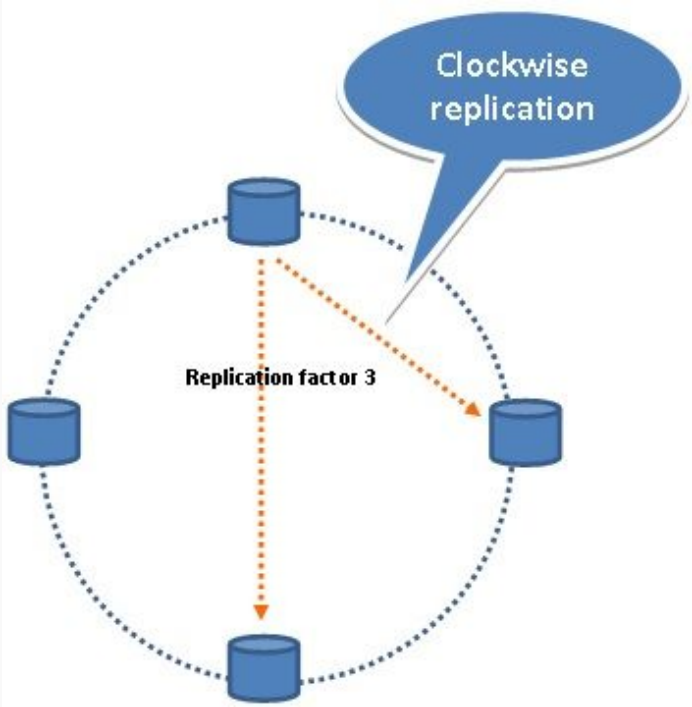
H. DROP TABLE

I. CREATE INDEX

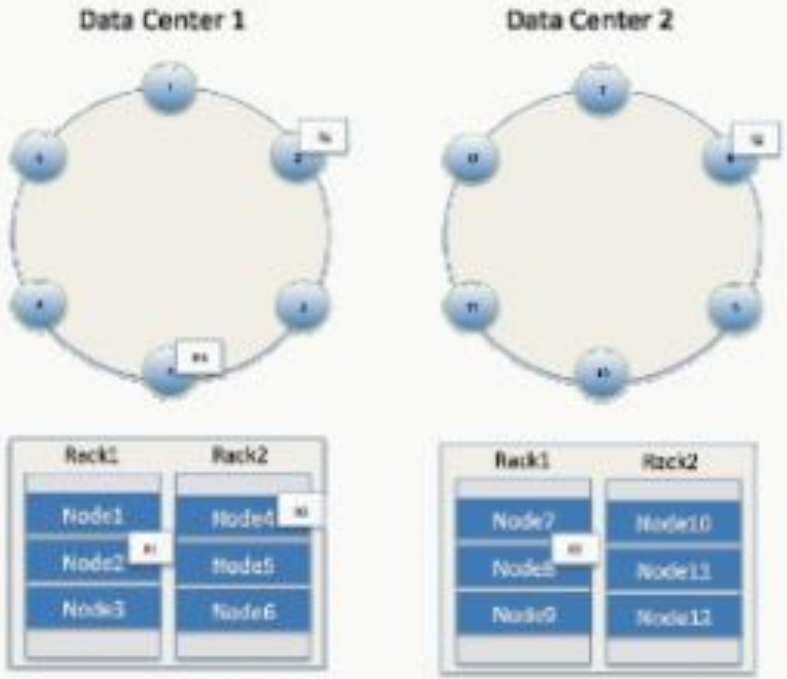
J. DROP INDEX

CREACIÓN DE UN KEYSPACE

SimpleStrategy: se define cuando deseamos que los datos se almacenen en un solo data center.



NetworkTopologyStrategy: se define si se planea desplegar el cluster en múltiples data centers.



https://docs.datastax.com/en/cassandra-oss/3.x/cassandra/architecture/archDataDistributeReplication.html

CREAR Y USAR UN KEYSPACE

Nombre del keyspace

```
CREATE KEYSPACE prueba WITH replication =
{'class': 'SimpleStrategy', 'replication_factor': '1'};
```

Define la estrategia de replicación (SimpleStrategy o NetworkTopologyStrategy)

Número de réplicas de los datos en múltiples nodos, solo si es SimpleStrategy.

```
USE prueba;
```

```
For more details, please visit https://support.apple.com/kb/HT208050.
$ cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.10 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh> CREATE KEYSPACE prueba WITH replication
   ... = {'class': 'SimpleStrategy', 'replication_factor': '1'};
cqlsh> USE prueba;
```

```
DESCRIBE KEYSPACES;  //Muestra los keyspaces disponibles en la base de datos.
```

CREACIÓN DE TABLAS (FAMILIAS DE COLUMNAS)

- En CQL 3.x, las familias de columnas se crean con una sintaxis de creación de tablas (CREATE TABLE) similar a SQL.
- Una tabla debe tener al menos una clave primaria.
- Una clave primaria identifica la ubicación y el orden de los datos almacenados. La clave principal se define cuando se crea la tabla y no se puede modificar.

```
CREATE TABLE [IF NOT EXISTS] [keyspace_name.]table_name (
  column_definition [, ...]
  PRIMARY KEY (column_name [, column_name ...])
[WITH table_options
  | CLUSTERING ORDER BY (clustering_column_name order])
  | ID = 'table_hash_tag'

   | COMPACT STORAGE]
```

https://docs.datastax.com/en/cql-oss/3.3/cql/cql_reference/cqlCreateTable.html

TIPOS DE DATOS

Tipo de blob

Tipo de datos	Descripción
`blob`	Representa bytes arbitrarios.

Tipo booleano

Tipo de datos	Descripción
`boolean`	Representa`true`o bien`false`.

Tipos relacionados con el tiempo

Tipo de datos	Descripción
`timestamp`	Representa una marca temporal.
`timeuuid`	Representa un objetoUUID versión 1.

Tipos numéricos

Tipo de datos	Descripción
`bigint`	Representa un largo firmado de 64 bits.
`counter`	Representa un contador de enteros firmado de 64 bits. Para obtener más información, consulte Counters.
`decimal`	Representa un decimal de precisión variable.
`double`	Representa un punto flotante IEEE 754 de 64 bits.
`float`	Representa un punto flotante IEEE 754 de 32 bits.
`int`	Representa un int firmado de 32 bits.
`varint`	Representa un entero de precisión arbitraria.

Tipos de cadena

Tipo de datos	Descripción
`ascii`	Representa una cadena de caracteres ASCII.
`text`	Representa una cadena de caracteres UTF-8.
`varchar`	Representa una cadena con codificación UTF-8 (`varchar`es un alias para`text`).

EJEMPLO

Crear la familia de columnas con el nombre USERS, con clave primaria id.

```
CREATE TABLE users (
    id text,
    firstname text,
    lastname text,
    year int,
    height float,
    phone int,

    PRIMARY KEY (id));
```

Partition key

- En este caso, al ser el id del usuario único, todas las particiones de la tabla contienen una sola fila.
- No hay clustering key.

partición 1	001	Pedrito	Castro	1.57	1972	Pedrito@hotmail.com
partición 2	002	Maria	Fernandez	1.72	1980	maria@correo.com
partición 3	003	Manuela	Rojas	1.62	1980	manuela@correo.com

EJEMPLO 2

Crear la familia de columnas con el nombre USERS_YEAR, con clave primaria year.

```
CREATE TABLE users_year (
    id text,
    firstname text,
    lastname text,
    year int,
    height float,
    phone int,
    Email text,

    PRIMARY KEY (year));
```

Partition key

- Las réplicas de las filas de una partición se encuentran físicamente juntas.
- Las diferentes particiones de las filas de una familia de columnas (tabla) pueden residir físicamente en nodos diferentes

partición 1	1972	Pedrito	Castro	1.57	001	Pedrito@hotmail.com
partición 2	1980	Maria	Fernandez	1.72	002	maria@correo.com
	1980	Manuela	Rojas	1.62	003	manuela@correo.com

REVISAR LA ESTRUCTURA DE LA TABLA

1. DESCRIBE users;

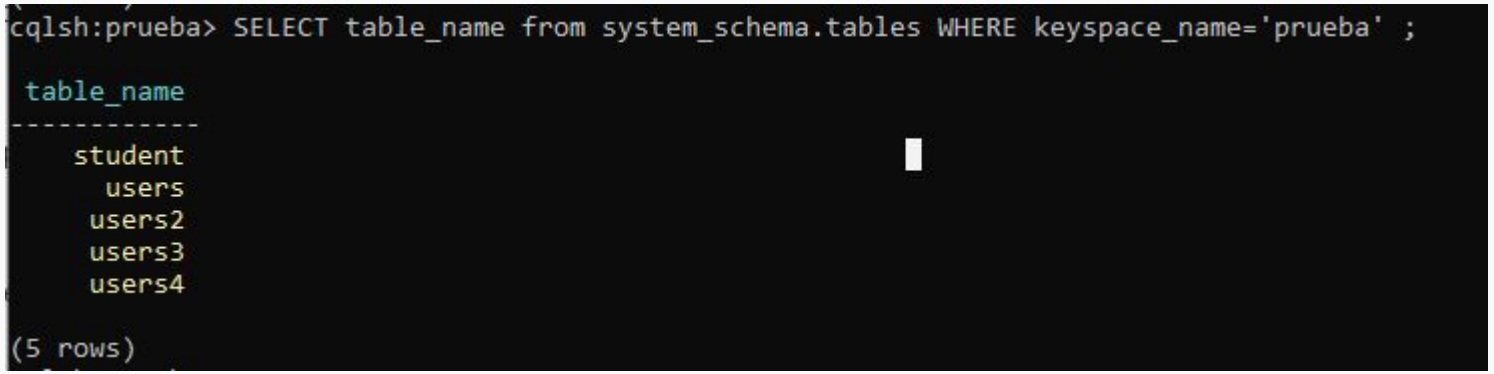
```
cqlsh:prueba> describe users;

CREATE TABLE prueba.users (
    id text PRIMARY KEY,
    firstname text,
    height float,
    lastname text,
    phone int,
    year int
) WITH additional_write_policy = '99p'
    AND bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND cdc = false
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_threshold': '32',
 'min_threshold': '4'}
    AND compression = {'chunk_length_in_kb': '16', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND crc_check_chance = 1.0
    AND default_time_to_live = 0
    AND extensions = {}
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair = 'BLOCKING'
    AND speculative_retry = '99p';
cqlsh:prueba>
```

AGREGAR NUEVO CAMPO A LA TABLA

1. ALTER TABLE users ADD email text;
2. DESCRIBE users;

Consulta al diccionario de datos:

SELECT * from system_schema.tables
WHERE keyspace_name='practica' ;




```
DESCRIBE TABLES; //Muestra los tablas que hay en el keyspace actual
```

MODIFICACIÓN

Para modificar tablas o keyspaces se utiliza la cláusula ALTER.

- Modificación de keyspace:

```
ALTER KEYSPACE BD_tienda_online WITH REPLICATION = { 'class' : 'SimpleStrategy',
'replication_factor' : 2 };
```

- Modificación de tablas:

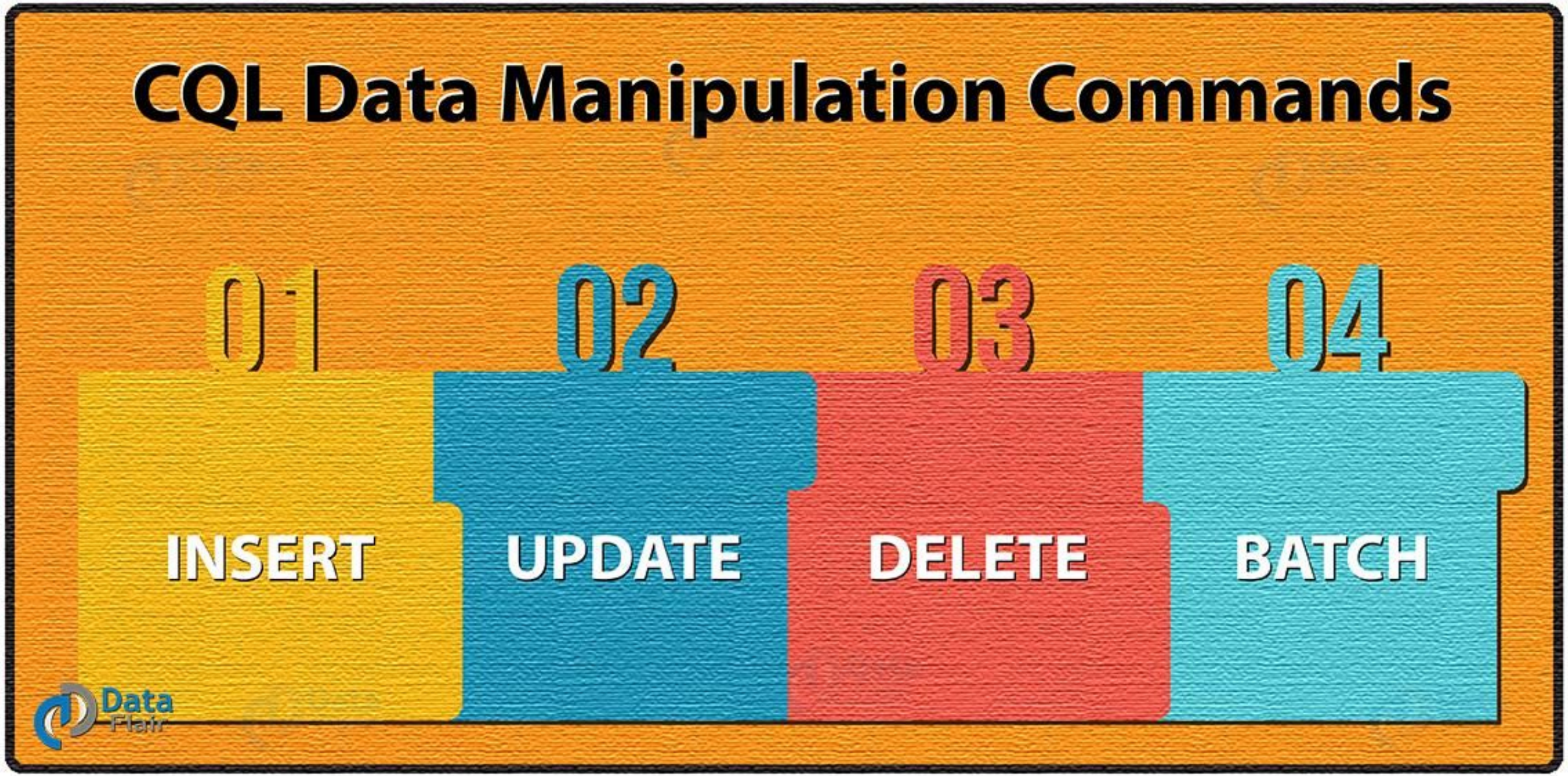
```
ALTER TABLE users ADD apellido1 int;

ALTER TABLE users ALTER apellido1 TYPE text;

ALTER TABLE users DROP appelido1;
```

- El ALTER también se puede utilizar para modificar usuarios o tipos definidos.

BORRADO

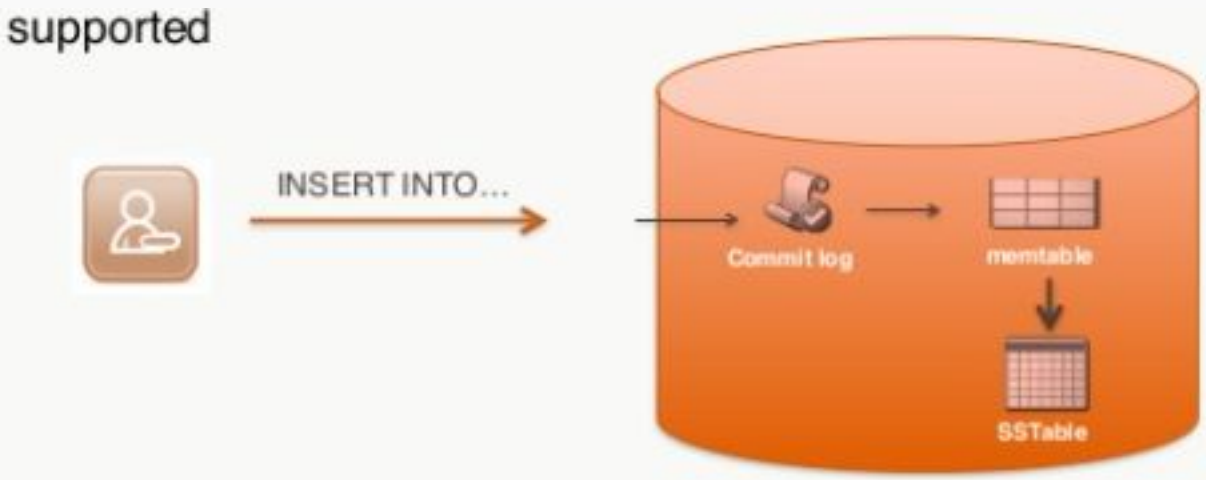
Para borrar cualquier elemento de la base de datos (tabla, índice, usuario…) se utiliza la cláusula DROP.
Ejemplos:

```
DROP TABLE usuario;
DROP INDEX indiceNombreUsuario;
DROP TRIGGER disparador1X;
DROP TYPE tipo2Y;
DROP KEYSPACE prueba;
…
```



https://data-flair.training/blogs/cql-data-manipulation-commands/

INSERTAR FILAS



INSERTAR FILAS

Ejemplo:

```
INSERT INTO users(id, firstname, lastname, height, year, email) VALUES ('001', 'Pedrito', 'Castro', 1.57, 1972, 'Pedrito@hotmail.com');

INSERT INTO users(id, firstname, lastname, height, year, email) VALUES  ('002','Maria', 'Fernandez', 1.72, 1980,'maria@correo.com');
```

Ver la tabla con todos los datos: `SELECT * FROM users;`

Inserte los datos de 5 usuarios más

```
INSERT INTO users(id, firstname, lastname, height, year, email) VALUES ('003', 'Manuela',  'Rojas', 1.62, 1980, 'manuela123@yahoo.com');

INSERT INTO users(id, firstname, phone,year) VALUES ('004', 'Camila', 324228,1978);
```

USING TTL

Esta opción permite realizar un insert por tiempo limitado, dicho tiempo se define en segundos y en el momento en el que se cumpla ese tiempo, Cassandra automáticamente eliminará ese dato introducido.

```
INSERT INTO users(id, firstname, phone, email) VALUES ('012', 'John', 798221, 'john2018@outlook.com') USING TTL 600;
```

id	apellido	edad	email	estatura	nombre	telefono
004	null	null	null	null	Camila	324228
012	null	null	john2018@outlook.com	null	John	798221
002	Fernandez	null	maria@correo.com	1.72	Maria	3156745
001	Castro	47	Pedrito@hotmail.com	1.57	Pedrito	null
003	null	null	manuela123@yahoo.com	null	Manuela	235644

MODELAMIENTO

Objetivos:

Regla 1: Distribuir los datos por todo el clúster

- Es deseable que cada nodo del clúster tenga un volumen de datos similar (equilibrio).
- Como las filas se distribuyen en base a la **partition key** es conveniente escoger una clave primaria adecuada.

Regla 2: Minimizar el número de particiones a leer

- Las particiones son grupos de filas que comparten la misma partition key.
- Cuantas menos particiones tengan que ser leídas, más rápida será la lectura.

https://www.datastax.com/blog/basic-rules-cassandra-data-modeling

ACTUALIZAR UN VALOR EN LA TABLA

Ejemplo:

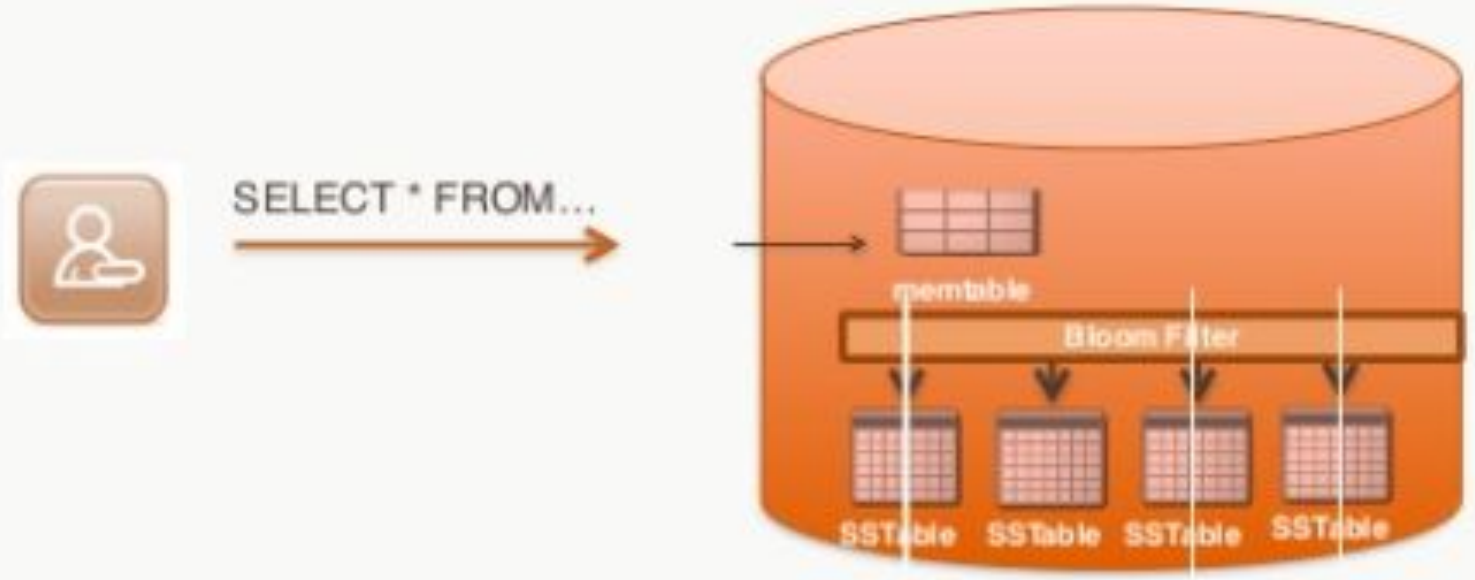
```
UPDATE users SET email='Pedrito123@gmail.com' WHERE id='001';
```

ver cambio en la tabla:

```
SELECT * FROM users;
```

id	apellido	edad	email	estatura	nombre	telefono
004	null	null	null	null	Camila	324228
012	null	null	john2018@outlook.com	null	John	798221
002	Fernandez	null	maria@correo.com	1.72	Maria	3156745
001	Castro	47	Pedrito123@gmail.com	1.57	Pedrito	null
003	null	null	manuela123@yahoo.com	null	Manuela	235644

CONSULTAS


SELECT * FROM...
memtable
Bloom Filter
SSTable
SSTable
SSTable
SSTable

CONSULTAS

Proyección. Un * indica que se devuelven todas las columnas.

```
SELECT id
FROM users
WHERE id='001'
ORDER BY nombre;
```

Tabla (familia de columnas) consultada. Solo se puede consultar una tabla (no hay JOINs).

Criterios de búsqueda a cumplir. En el WHERE, sólo se pueden utilizar columnas que **formen parte de la partition key** o sobre las que se haya definido un índice secundario. Si la partition key es compuesta y se incluye en el WHERE, han de incluirse todos sus campos.

Criterios de ordenamiento. Siempre han de ser sobre columnas que estén incluidas en la clustering key.).

CONSULTAS

1. SELECT firstname FROM users WHERE id IN ('001','002');

```
cqlsh:prueba> SELECT nombre FROM usuarios WHERE id IN ('001','002');

 nombre
---------
 Pedrito
   Maria
```

2. SELECT phone FROM users WHERE id = '003';

```
cqlsh:prueba> SELECT phone FROM users WHERE id = '012';

 phone
--------
 798221
```

3. SELECT firstname , email FROM users WHERE id = '003';

```
cqlsh:prueba> SELECT nombre, email FROM usuarios WHERE id = '003';

 nombre  | email
---------+----------------------
 Manuela | manuela123@yahoo.com
```

TALLER

1. Muestre los nombres de todos los usuarios.
2. Muestre los nombres y apellidos de los usuarios nacidos antes del año 1980.

No se puede mostrar la segunda consulta… claro year no es parte de la PRIMARY KEY

Probemos entonces con USERS_YEAR. Creen la tabla USERS_YEAR y carguen los datos.

¿Qué ocurre?

- Como tiene el mismo partition key se sobreescriben (upsert)

```
cqlsh:prueba> select * from users2;

 year | email                 | firstname | height | id  | lastname | phone
------+-----------------------+-----------+--------+-----+----------+--------
 1972 |  Pedrito@hotmail.com |   Pedrito |   1.57 | 001 |   Castro |   null
 1978 |                 null |    Camila |   null | 004 |     null | 324228
 1980 | manuela123@yahoo.com |   Manuela |   1.62 | 003 |    Rojas |   null

(3 rows)
cqlsh:prueba> INSERT INTO users2(id, firstname, lastname, height, year, email)
, 1980,'maria@correo.com');
cqlsh:prueba> select * from users2;

 year | email                 | firstname | height | id  | lastname  | phone
------+-----------------------+-----------+--------+-----+-----------+--------
 1972 | Pedrito@hotmail.com |   Pedrito |   1.57 | 001 |    Castro |   null
 1978 |                 null |    Camila |   null | 004 |      null | 324228
 1980 |    maria@correo.com |     Maria |   1.72 | 002 | Fernandez |   null
```

CONSULTAS

Qué resultado darían las siguientes consultas:

- SELECT id FROM users_year

  WHERE year = 1980;

- SELECT id FROM users_year

  WHERE year = 1980 and id='003';

CONSULTAS

Order by:

- Sólo puede utilizarse sobre la clustering key.
- Indispensable que las partition key aparezcan en la claúsula where.
- Sólo tiene sentido para ordenar particiones.

Limit:

- Número máximo de filas a retornar.

ALLOW FILTERING

- Puede utilizarse para realizar consultas en las que en el WHERE no haya condiciones en la partition key.
- No recomendado a menos que sea estrictamente necesario: acceso secuencial a los datos

CONSULTAS

Dada la definición de la tabla USERS3

```
CREATE TABLE users3 (
        id text,
        firstname text,
        lastname text,
        year int,
        height float,
        phone int,
        email text,

        PRIMARY KEY ((id, year), lastname));
```

Qué resultado darían las siguientes consultas:

- SELECT id FROM users3

  WHERE year = 1980;

- SELECT id FROM users3

  WHERE year = 1980 and id='003';

CONSULTAS

Dada la definición de la tabla USERS

```
CREATE TABLE users4 (
        id text,
        firstname text,
        lastname text,
        year int,
        height float,
        phone int,
        email text,

        PRIMARY KEY ( year, lastname));
```

Qué resultado darían las siguientes consultas:

- SELECT id FROM users4

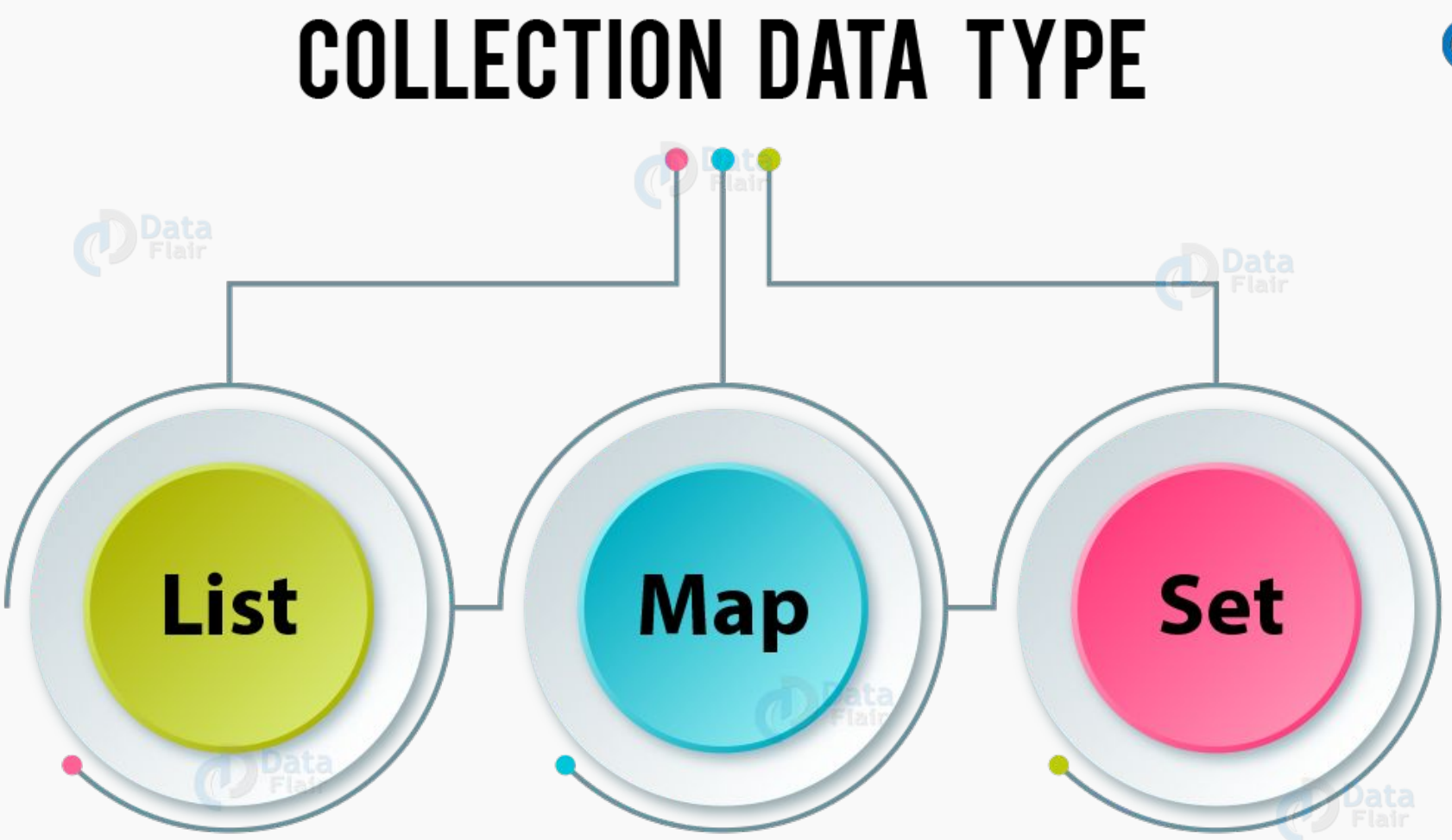
  WHERE year >1980;

- SELECT id FROM users4

  WHERE year =1980;

Muestre el id y el apellido de los usuarios de 1980 ordenados por el apellido

COLECCIONES



Restricciones: https://docs.datastax.com/en/cql-oss/3.3/cql/cql_using/useCollections.html

LIST

- Los valores se almacenan en forma de lista.
- Un valor puede almacenarse varias veces.
- El orden de la lista no se puede cambiar.

```
CREATE TABLE <table name>(
column1 PRIMARY KEY,
column2 list <data type>,
column3 list <data type>,
.....
);
```

EJEMPLO

```
CREATE TABLE student
        (EN int,
        NAME text,
        EMAIL LIST<text>,
        PRIMARY KEY(EN),
);
```

```
INSERT INTO student (EN, NAME, EMAIL) VALUES(001,'Andres',['ayush@gmail.com']);
INSERT INTO student (EN, NAME, EMAIL) VALUES(002,'Lucia',['aarav@ymail.com']);
INSERT INTO student (EN, NAME, EMAIL) VALUES(003,'Carlos',['kabir@hotmail.com']);
```

```
UPDATE student
SET EMAIL=EMAIL+['ayush@hotmail.com']
where EN=001;
```

en	email	name
1	['ayush@gmail.com', 'ayush@hotmail.com']	Andres
2	['aarav@ymail.com']	Lucia
3	['kabir@hotmail.com']	Carlos

SET

- Consiste en un grupo de elementos con valores únicos.
- Los elementos del conjunto vuelven en orden después de la ejecución.

```
CREATE TABLE<table name> (
column1 PRIMARY KEY,
column2 set <data type>,
column3 set <data type>
.....
);
```

EJEMPLO

Modifique la tabla STUDENT, adicione el atributo: `BRANCH de tipo SET<text>`

INSERT INTO student (EN, NAME, BRANCH) VALUES(004,'Alejandro',{'electrical engineering'});

INSERT INTO student (EN, NAME, BRANCH) VALUES(005,'Adriana',{'Computer engineering'});

INSERT INTO student (EN, NAME, BRANCH) VALUES(006,'Fabian',{'Applied Physics'});

```
UPDATE student
SET BRANCH=BRANCH+{'Electrical and Electronics'}
where EN=005;
```

Adicione además ‘Architect’ y ‘Zoo’, para ver el orden en que quedan almacenados. E ingrese nuevamente ‘Zoo’ (se ingresa nuevamente?)

MAP

- Se almacena un par de elementos clave-valor.
- Para cada clave, sólo puede existir un valor y no se pueden almacenar duplicados. Tanto la clave como el valor se designan con un tipo de datos.
- Con el tipo de mapa, puede almacenar información relacionada con la marca de tiempo en los perfiles de usuario.
- Cada elemento del mapa se almacena internamente como una columna de Cassandra que puede modificar, reemplazar, eliminar y consultar.
- Cada elemento puede tener un tiempo de vida individual y vencer cuando finaliza el TTL.

```
CREATE TABLE<table name>
(column1 PRIMARY KEY,
column2 map <type, data type>, column3 map
<type, data type>
.....
);
```

EJEMPLO

Modifique la tabla STUDENT, adicione el atributo `SUBJECT MAP<text,text>`

```
INSERT INTO student(EN, SUBJECT) VALUES(007,{'physics':'mathematics'});
INSERT INTO student(EN, SUBJECT)VALUES(008,{'operating system':'robotics'});
INSERT INTO student(EN, SUBJECT) VALUES(009,{'power system':'machines'});
```

BORRAR FILAS DE LA TABLA

Ejemplo:

1. DELETE FROM users WHERE id = ‘002’;

```
cqlsh:prueba> select * from users;

 id  | email                 | firstname | height | lastname  | phone  | year
-----+-----------------------+-----------+--------+-----------+--------+------
 004 |                  null |    Camila |   null |      null | 324228 | 1978
 002 |      maria@correo.com |     Maria |   1.72 | Fernandez |   null | 1980
 001 | Pedrito123@gmail.com |   Pedrito |   1.57 |    Castro |   null | 1972
 003 | manuela123@yahoo.com |   Manuela |   1.62 |     Rojas |   null | 1980

(4 rows)
cqlsh:prueba> delete from users where id='002';
cqlsh:prueba> select * from users;

 id  | email                 | firstname | height | lastname | phone  | year
-----+-----------------------+-----------+--------+----------+--------+------
 004 |                  null |    Camila |   null |     null | 324228 | 1978
 001 | Pedrito123@gmail.com |   Pedrito |   1.57 |   Castro |   null | 1972
 003 | manuela123@yahoo.com |   Manuela |   1.62 |    Rojas |   null | 1980

(3 rows)
cqlsh:prueba>
```

REFERENCIAS

- Bases de datos orientado a columnas. Adrian Garcete. Universidad Católica.

  http://jeuazarru.com/wp-content/uploads/2014/10/dbco.pdf

- La información de Cassandra, tomada de: NoSQL: la siguiente generación de bases de datos. Dr. Diego Lz. de Ipiña Glz. de Artaza. DeustoTech
  http://eventos.citius.usc.es/bigdata/

- Cassandra

  http://cassandra.apache.org

- Getting started with CQL:

  http://www.datastax.com/docs/0.8/dml/using_cql

- Modelo de datos Cassandra

  https://sites.google.com/site/uegoman/modelo-de-datos-de-cassandra

- Academy Datastax

https://academy.datastax.com/#/catalog/c5b626ca-d619-45b3-adf2-a7d2b940a7ee