Asignatura	Datos del alumno	Fecha
Bases de Datos para el Big Data	Apellidos:	
	Nombre:	

Actividad: Trabajando con Cassandra

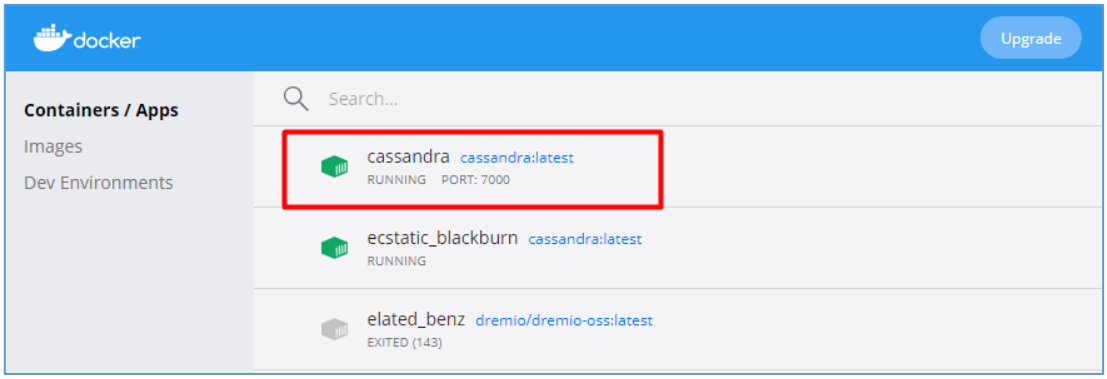
Objetivos

Esta actividad te permitirá profundizar en el uso de Cassandra como motor de base de datos. Realizarás una serie de ejercicios guiados que te ayudarán a conocer las diferentes funcionalidades que posee Cassandra como motor de base de datos.

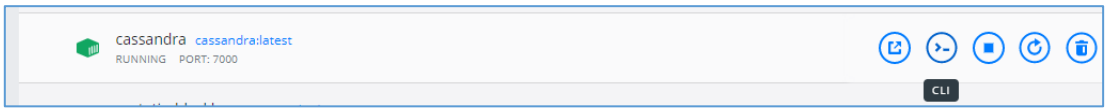
Descripción

Para realizar esta actividad, se asume que tienes instalado el contenedor de Cassandra en Docker (revisar el tema 8). Sobre dicho contenedor, los pasos que debes llevar a cabo son los siguientes:

- Inicia Docker

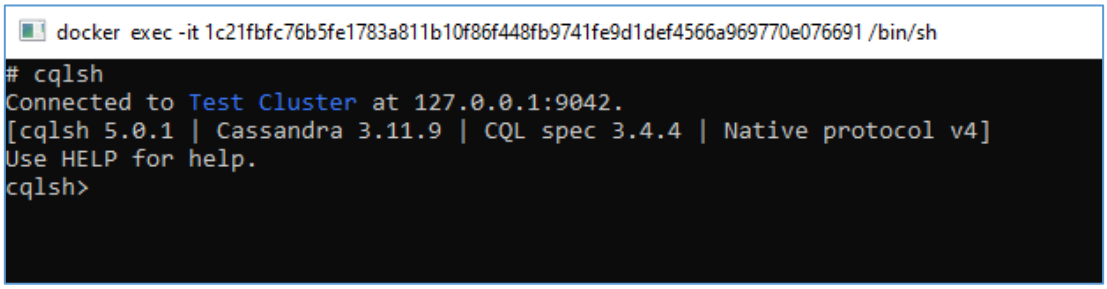


- Inicia el contenedor de Cassandra



- Inicia el cliente de Cassandra: **cqlsh**

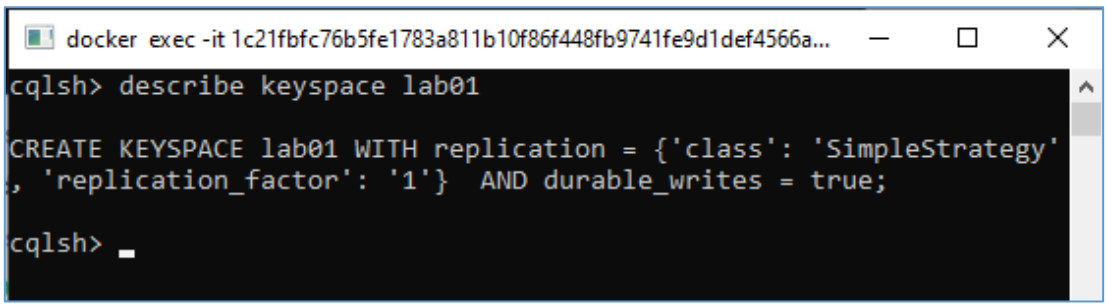
Asignatura	Datos del alumno	Fecha
Bases de Datos para el Big Data	Apellidos:	
	Nombre:	



Ejecuta las siguientes instrucciones:

1. **Aprende a trabajar con Keyspaces**
   - Consulta los **keyspaces** existentes.
     - `DESCRIBE KEYSPACES;`
   - Crea un **keyspaces** cualquiera.
     - `CREATE KEYSPACE Lab00_TEST WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1};`
   - Borra el **keyspace** recientemente creado.
     - `DROP KEYSPACE Lab00_TEST;`
2. **Manos la obra con la BD:**

- Crea una base de datos
  - `CREATE KEYSPACE Lab01 WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1};`
  - Comprueba que está creada
  - `DESCRIBE KEYSPACE Lab01`



- Usa la keyspace recién creada:
  - `USE lab01` *(como en MongoDB, ¿recuerdas?)*

Asignatura	Datos del alumno	Fecha
Bases de Datos para el Big Data	Apellidos:	
	Nombre:	

```
cqlsh> use lab01
   ... ;
cqlsh:lab01> _
```

- Crea una tabla dentro del **keyspace**. Llama dicha tabla "*empleado*" y define su la clave primaria.
  - CREATE TABLE **empleado** (IdEmpleado int, IdDept int, nombre varchar, apellido varchar, PRIMARY KEY (IdEmpleado, IdDept));

```
docker exec -it 1c21fbfc76b5fe1783a811b10f86f448fb9741fe9d1def4566a969770e076691 /bin/sh
cqlsh:lab01> CREATE TABLE empleado (IdEmpleado int, IdDept int, nombre varchar, apellido varchar, PRIMARY KEY (IdEmpleado, IdDept));
```

- Inserta tres registros en la tabla *empleado*.
- INSERT INTO **empleado** (IdEmpleado, IdDept, nombre, apellido) VALUES (333, 30, 'Adrián', 'Paz');
- INSERT INTO **empleado** (IdEmpleado, IdDept, nombre, apellido) VALUES (338, 15, 'Alejandra', 'Paz');
- INSERT INTO **empleado** (IdEmpleado, IdDept, nombre, apellido) VALUES (241, 30, 'Alma', 'Dura');
- **Reto:** vuelve a realizar los tres *insert* anteriores y explica qué ocurre.
- Consulta los datos de la nueva tabla.
  - SELECT * FROM empleado;

```
cqlsh:lab01> select * from empleado;

 idempleado | iddept | apellido | nombre
------------+--------+----------+-----------
        241 |     30 |     Dura |      Alma
        338 |     15 |      Paz | Alejandra
        333 |     30 |      Paz |    Adrián

(3 rows)
cqlsh:lab01>
```

- Añade un campo nuevo a la tabla creada y vuelve a consultarla. Observa el valor por defecto del nuevo campo.
  - ALTER TABLE empleado ADD *direccion* text;
  - SELECT * FROM empleado;

```
cqlsh:lab01> ALTER TABLE empleado ADD direccion text;
cqlsh:lab01> select * from empleado;

 idempleado | iddept | apellido | direccion | nombre
------------+--------+----------+-----------+-----------
        241 |     30 |     Dura |      null |      Alma
        338 |     15 |      Paz |      null | Alejandra
        333 |     30 |      Paz |      null |    Adrián

(3 rows)
cqlsh:lab01>
```

- Ahora añade valores al campo nuevo *dirección*.
  - UPDATE empleado SET *direccion* = 'Logroño' WHERE IdEmpleado = 241 AND IdDept = 30;
- Añade otro campo nuevo llamado *código_postal*.
  - ALTER TABLE empleado ADD *codigo_postal* text;

```
cqlsh:lab01> UPDATE empleado SET direccion = 'Logroño' WHERE IdEmpleado = 241 AND IdDept = 30;
cqlsh:lab01> select * from empleado;

 idempleado | iddept | apellido | codigo_postal | direccion | nombre
------------+--------+----------+---------------+-----------+-----------
        241 |     30 |     Dura |          null |   Logroño |      Alma
        338 |     15 |      Paz |          null |      null | Alejandra
        333 |     30 |      Paz |          null |      null |    Adrián
```

- Prueba a eliminar el campo *código_postal* de la tabla.
  - ALTER TABLE empleado DROP *codigo_postal*;
  - SELECT * FROM empleado;

```
cqlsh:lab01> ALTER TABLE empleado DROP codigo_postal;
cqlsh:lab01> select * from empleado;

 idempleado | iddept | apellido | direccion | nombre
------------+--------+----------+-----------+-----------
        241 |     30 |     Dura |   Logroño |      Alma
        338 |     15 |      Paz |      null | Alejandra
        333 |     30 |      Paz |      null |    Adrián

(3 rows)
cqlsh:lab01>
```

- Algunas instrucciones más. Borra un registro usando condiciones.
  - DELETE FROM empleado WHERE IdEmpleado = 333;
  - SELECT * FROM empleado;

```
cqlsh:lab01> DELETE FROM empleado WHERE IdEmpleado = 333;
cqlsh:lab01> select * from empleado;

 idempleado | iddept | apellido | direccion | nombre
------------+--------+----------+-----------+-----------
        241 |     30 |     Dura |   Logroño |      Alma
        338 |     15 |      Paz |      null | Alejandra

(2 rows)
cqlsh:lab01>
```

Asignatura	Datos del alumno	Fecha
Bases de Datos para el Big Data	Apellidos:	
	Nombre:	

- Crea una nueva tabla llamada *lab02_temp* y luego consúltala.
  - CREATE TABLE **lab02_temp** (Id int, Ciudad text, nombre varchar, apellido varchar, PRIMARY KEY (Id));
  - SELECT * FROM **lab02_temp**;

```
cqlsh:lab01> CREATE TABLE lab02_temp (Id int, Ciudad text, nombre varchar, apellido varchar, PRIMARY KEY (Id));
cqlsh:lab01> select * from lab02_temp;

 id | apellido | ciudad | nombre
----+----------+--------+--------

(0 rows)
cqlsh:lab01>
```

- DROP TABLE **lab02_temp**;

```
cqlsh:lab01> drop table lab02_temp;
cqlsh:lab01> select * from lab02_temp;
InvalidRequest: Error from server: code=2200 [Invalid query] message="unconfigured table lab02_temp"
cqlsh:lab01>
```

```
cqlsh:lab01> describe tables

temporal  empleado

cqlsh:lab01>
```

- Utiliza el comando DESCRIBE TABLES; para ver las tablas existentes. En caso de que no haya ninguna tabla en el *keyspace*, este devolverá <empty>.

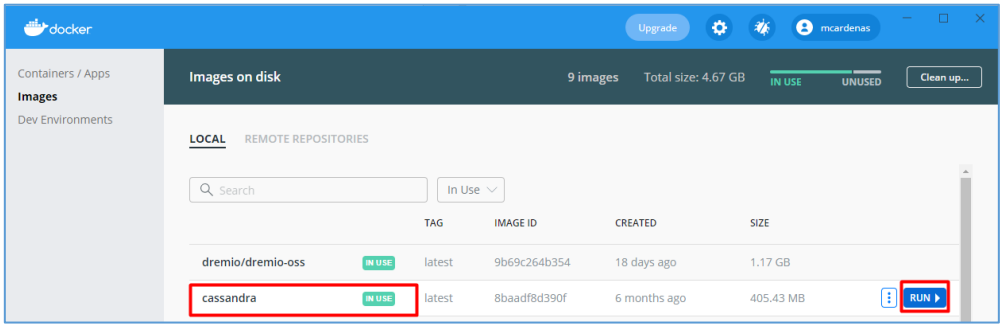
3. Trabaja con ficheros

- Utiliza el fichero "***data_unirlab_001.csv***", el cual tiene cuatro campos: *dni, nombre, apellido, edad, email*.
- El fichero CSV lo vas a cargar en Cassandra. Para ello debes subirlo a tu contenedor de Docker. Para ello, debes compartir ficheros entre tu máquina (host) y tu contenedor.

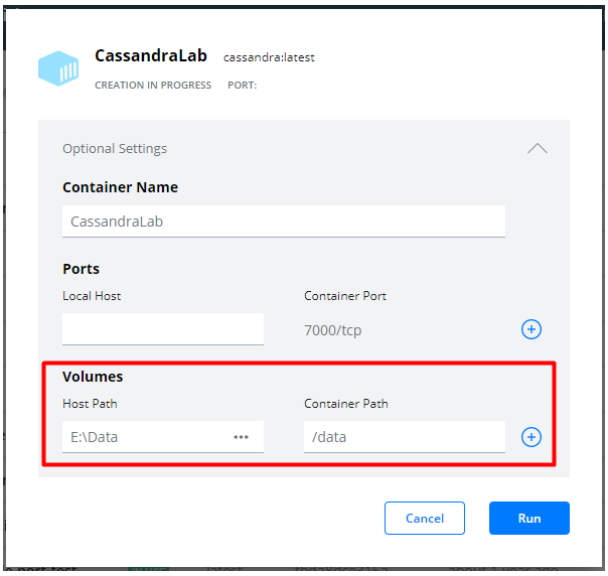
4. Pasos para compartir ficheros.

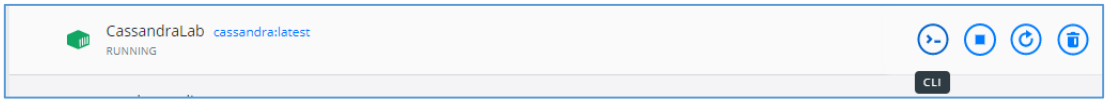
- Detén el contenedor que estabas ejecutando previamente.
- Crea un director en "**E:/Data**" (Windows, por ejemplo) luego copia aquí el fichero CSV.
- Localiza la imagen de Cassandra en tu Docker Desktop.

- Accede a la opción “RUN” que está a la derecha de la imagen.

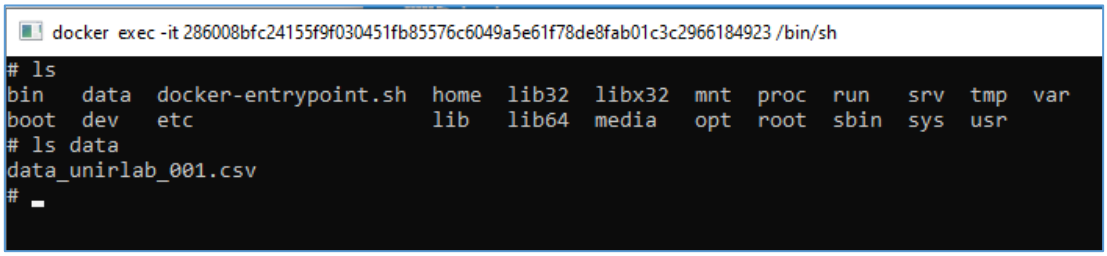


- Ejecuta la imagen y en la ventana siguiente, indica una ruta que has definido en el paso anterior. En la misma ventana, asigna una ruta interna dentro del contenedor. Docker lo que hará es crear un puente entre host-contenedor para poder ver el fichero, por ejemplo: “**/data**”





- Abre la terminar del contenedor y en él busca la ruta que has mapeado previamente.



Si todo ha ido bien, debes poder ver el fichero CSV desde el contenedor.

5. **Carga el fichero**

- El primer paso es crear el keyspace. Asumimos que se usará una keyspace nueva diferente a la de los ejercicios anteriores.
  - **CREATE KEYSPACE DBClientes WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1}** ;
- Accede a dicho keyspace para trabajar sobre él.
  - **USE DBClientes**;
- Crea la tabla donde se alojarán los datos del fichero.
  - **CREATE TABLE usuario (dni text, nombre text, apellido varchar, edad int, email text, PRIMARY KEY (dni));**
- Después de crear la tabla, cargar los datos del fichero CSV. Ten presente el separador que este incluye para indicarlo a continuación.
  - **COPY usuario (dni, nombre, apellido, edad, email) FROM '/data/data_unirlab_001.csv' WITH DELIMITER = ';' AND HEADER=TRUE;**
- El siguiente será el mensaje que se observe.
  - n rows imported in 0.202 seconds.
- Ejecute la instrucción siguiente y compruebe que los datos están en dicha tabla.
  - **SELECT * FROM usuario;**
- **Reto**: ejecuta lo siguiente.
  - **SELECT * FROM usuario LIMIT 10;**

Asignatura	Datos del alumno	Fecha
Bases de Datos para el Big Data	Apellidos:	
	Nombre:	

```
docker exec -it 286008bfc24155f9f030451fb85576c6049a5e61f78de8fab01c3c2966184923 /bin/sh
cqlsh:dbclientes> CREATE TABLE usuario ( dni text, nombre text, apellido varchar,
edad int, email text, PRIMARY KEY (dni) );
cqlsh:dbclientes> COPY usuario (dni, nombre, apellido, edad, email) FROM '/data/da
ta_unirlab_001.csv' WITH DELIMITER = ';'  AND HEADER=TRUE;
Using 1 child processes

Starting copy of dbclientes.usuario with columns [dni, nombre, apellido, edad, ema
il].
Processed: 12 rows; Rate:      24 rows/s; Avg. rate:      34 rows/s
12 rows imported from 1 files in 0.351 seconds (0 skipped).
cqlsh:dbclientes> select * from usuario allow filtering;

 dni  | apellido | edad | email         | nombre
------+----------+------+---------------+----------
 135a |   Ruanda |   52 | lr@email.com |     Luis
 127a |   Torres |   23 | at@email.com |      Ana
 134B |  Jimenez |   19 | oj@email.com |    Oscar
 131a |  Batista |   45 | rb@email.com |   Romulo
 124B |     Diaz |   23 | cd@email.com | Catalino
 123A |    Lopez |   28 | jp@email.com |     Juan
 130B |  Fuentes |   74 | bf@email.com |    Berto
 128B | Altamira |   69 | ja@email.com |    Julia
 129a |   Garcia |   85 | jg@email.com |    Jairo
 133B |   Caldas |   56 | fc@email.com |    Fredy
 125a | Gonzalez |   44 | pg@email.com |    Pablo
 126B |     Ruiz |   52 | dr@email.com |     Dina

(12 rows)
cqlsh:dbclientes>
```

6. Es tu turno de explorar

- **Reto**: ejecuta lo siguiente.
  - **SELECT * FROM usuario WHERE dni = '125a';**

Ten en cuenta:

- **No** uses comillas dobles
- **No copies y pegues** las instrucciones, escríbelas para que practiques el uso del lenguaje CQL.
- Las **comillas simples** no son las que propone Word, edita y pon las comillas simples directamente en la terminal.

```
cqlsh:dbclientes> SELECT * FROM usuario WHERE dni = '125a';

 dni  | apellido | edad | email         | nombre
------+----------+------+---------------+--------
 125a | Gonzalez |   44 | pg@email.com |  Pablo

(1 rows)
cqlsh:dbclientes>
```

Asignatura	Datos del alumno	Fecha
Bases de Datos para el Big Data	Apellidos:	
	Nombre:	

Reto: ejecuta lo siguiente.

- **SELECT * FROM usuario WHERE edad > 1 allow filtering;**

Explica qué significa la instrucción “allow filtering”.

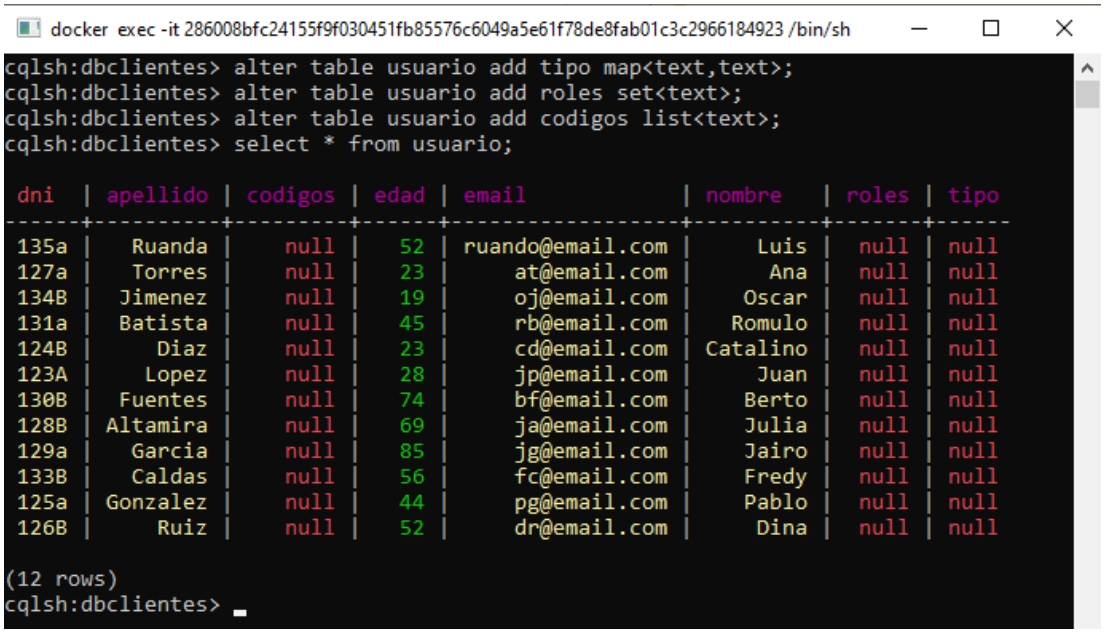
```
cqlsh:dbclientes> SELECT * FROM usuario WHERE edad > 50;
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execu
te this query as it might involve data filtering and thus may have unpredictable p
erformance. If you want to execute this query despite the performance unpredictabi
lity, use ALLOW FILTERING"
cqlsh:dbclientes> SELECT * FROM usuario WHERE edad > 50 allow filtering;

 dni  | apellido | edad | email         | nombre
------+----------+------+---------------+--------
 135a |   Ruanda |   52 | lr@email.com  |   Luis
 130B |  Fuentes |   74 | bf@email.com  |  Berto
 128B | Altamira |   69 | ja@email.com  |  Julia
 129a |   Garcia |   85 | jg@email.com  |  Jairo
 133B |   Caldas |   56 | fc@email.com  |  Fredy
 126B |     Ruiz |   52 | dr@email.com  |   Dina

(6 rows)
cqlsh:dbclientes>
```

- **Reto**: ejecuta lo siguiente.
  - **UPDATE FROM usuario SET email = ‘ruando@email.com’ WHERE dni = ‘135a;**

```
cqlsh:dbclientes> update usuario set email = 'ruando@email.com' where dni = '135a';

cqlsh:dbclientes> SELECT * FROM usuario;

 dni  | apellido | edad | email             | nombre
------+----------+------+-------------------+----------
 135a |   Ruanda |   52 | ruando@email.com |     Luis
 127a |   Torres |   23 |     at@email.com |      Ana
 134B |  Jimenez |   19 |     oj@email.com |    Oscar
 131a |  Batista |   45 |     rb@email.com |   Romulo
 124B |     Diaz |   23 |     cd@email.com | Catalino
 123A |    Lopez |   28 |     jp@email.com |     Juan
 130B |  Fuentes |   74 |     bf@email.com |    Berto
 128B | Altamira |   69 |     ja@email.com |    Julia
 129a |   Garcia |   85 |     jg@email.com |    Jairo
 133B |   Caldas |   56 |     fc@email.com |    Fredy
 125a | Gonzalez |   44 |     pg@email.com |    Pablo
 126B |     Ruiz |   52 |     dr@email.com |     Dina

(12 rows)
cqlsh:dbclientes>
```

- **Reto**: ejecuta las siguientes instrucciones y explica qué hace cada una de ellas.
  - **ALTER TABLE usuario ADD tipo map<text, text>;**
  - **ALTER TABLE usuario ADD roles set<text>;**
  - **ALTER TABLE usuario ADD codigos list<int>;**

Asignatura	Datos del alumno	Fecha
Bases de Datos para el Big Data	Apellidos:	
	Nombre:	



Revisa los apuntes del tema 7 y completa los siguientes retos:

- En la nueva columna **tipo** guarda los siguientes valores:
  - Usuario 135a = {'area': 'general', 'metodo': 'acceso', 'ventas': 'no'}
  - Usuario 130B = {'area': 'local', 'metodo': 'consulta', 'ventas': 'no'}
  - Usuario 129a = {'area': 'privada', 'metodo': 'lectura', 'ventas': 'si'}
- En la nueva columna **roles** guarda los siguientes valores:
  - Usuario 128B = {'supervisor', 'delegado', 'usuario'}
  - Usuario 123A = {'admin', 'dbadmin'}
  - Usuario 134B = {'lector', 'editor', 'comentarista', 'literato', 'usuario', '*'}
- En la nueva columna **códigos** guarda los siguientes valores:
  - Usuario 126B = [1,2,6,5,8,7]
  - Usuario 124B = [8,9,7,6,5,2]
  - Usuario 133B = [0,5,6,3,4]

Asignatura	Datos del alumno	Fecha
Bases de Datos para el Big Data	Apellidos:	
	Nombre:	

Uso de UDT

- El primer paso es crear el keyspace. Asumimos que se usará una keyspace nueva diferente a la de los ejercicios anteriores.

```
CREATE KEYSPACE DBUsuarios2 WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1};
```

- Accede a dicho keyspace para trabajar sobre él.

```
USE DBUsuarios2;
```

- Crea el siguiente tipo de datos.

```
CREATE TYPE telefono (
    cod_pais int,
    numero text,
);
```

- Crea este otro tipo de datos.

```
CREATE TYPE direccion (
    calle text,
    ciudad text,
    poblacion text,
    pais text,
    cp text,
    telefonos map<text, telefono>
);
```

- Ahora la siguiente tabla.

```
CREATE TABLE usuario (
    nombre text PRIMARY KEY,
    direcciones map<text, frozen<direccion>>
);
```

- Ahora inserta datos en la nueva tabla.

```
INSERT INTO usuario (nombre, direcciones)
   VALUES ('Pepe Armando', {
     'casa': {
        calle: 'General Ricardo',
        ciudad: 'Madrid',
        poblacion: 'Madrid',
        pais: 'Espana',
        cp: '28050',
        telefonos: {'movil_personal': {cod_pais: 35, numero: 'X'},
```

```
                    'movil_oficina': {cod_pais: 34, numero:
'666888963'}}
      },
      'oficina': {
         calle: 'Plaza Castilla',
         ciudad: 'Madrid',
         poblacion: 'Centro',
         pais: 'Madrid',
         cp: '28036',
         telefonos: {'fax': {cod_pais: 33, numero: 'X'}}
      }
   }
);
```

- Modifica un tipo de datos creado.

```
ALTER TYPE direccion RENAME cp TO CodPostal;
```

- Crea una tabla que contenga un elemento JSON.

```
INSERT INTO documentos_2010 JSON '{"\"cod\"": 1, "descripcion":
"Codigo 1"}';
```