



1. EXPLORANDO CASSANDRA

1. EXPLORANDO CASSANDRA.....	1
1.1. Instalación del software.....	2
1.1.1. Instalación del JDK 11.....	2
1.1.2. Instalación de Cassandra.....	3
1.2. Iniciando Cassandra.....	3
1.3. Modelado de datos en Cassandra.....	5
1.3.1. Particiones.....	6
Ejemplo.....	7
Ejemplo.....	7
1.3.2. Diferencias de diseño entre un RDBMS y Cassandra.....	7
1.3.3. Diseñar para realizar un almacenamiento óptimo.....	8
1.3.4. Diseño conceptual en RDBMS.....	8
1.3.5. Diseño orientado a las consultas.....	8
1.3.6. Diseño lógico en Cassandra.....	10
1.3.7. Diseño físico.....	11
1.3.8. Evaluación y refinamiento de modelos de datos.....	12
1.3.8.1. Calcular el tamaño de las particiones.....	13
Ejemplo.....	13
1.3.8.2. Calcular el tamaño del espacio de almacenamiento.....	13
Ejemplo.....	14
Ejemplo.....	14
1.4. Arquitectura básica de Cassandra.....	16
1.5. Definición del esquema de la base de datos.....	17
1.5.1. Creación de un KeySpace y objetos.....	17
Ejemplo.....	18
Ejemplo.....	18
1.5.2. Poblado de datos.....	22
1.5.3. Actividades a realizar.....	24
1.6. Interactuando con Cassandra desde una aplicación Java.....	24
1.6.1. Instalación y configuración de IntelliJ Idea.....	24
1.6.2. Actividades a realizar.....	31



En este ejercicio práctico se realizarán diversas actividades para instalar, configurar, y hacer uso de Cassandra, una de las principales bases de datos NoSQL orientada a familias de columnas.

Cassandra es una base de datos NoSQL, open source, distribuida. Su modelo de datos está representado por un formato de almacenamiento columnar particionado que permite entre otras funcionalidades, consistencia eventual.

Cassandra fue diseñada originalmente por Facebook. Emplea una arquitectura dirigida por eventos **staged event-driven architecture** (SEDA), realiza una combinación del esquema de almacenamiento distribuido de Amazon Dynamo y técnicas de replicación de Google Bigtable. Los principales requerimientos que implementa Cassandra son:

- Replicación multi-master completa
- Disponibilidad global con baja latencia
- Escalabilidad con hardware simple
- Incremento lineal del throughput
- Balanceo de cargas en línea, crecimiento del cluster
- Consultas particionadas empleando la Key como criterio
- Esquemas flexibles

1.1. Instalación del software

1.1.1. Instalación del JDK 11

OpenJDK

- A. Acceder al sitio: <https://jdk.java.net/java-se-ri/11>
- B. Hacer click en : **Linux/x64 Java Development Kit** El nombre del archivo **tar.gz** tiene la siguiente estructura: **openjdk-<version>_linux-x64_bin.tar.gz**
- C. En una terminal, cambiarse al directorio de descarga y descomprimir el archivo.

```
tar -xvf openjdk-<version>_linux-x64_bin.tar.gz
```

- D. Mover la carpeta al directorio **/opt** En esta carpeta es donde generalmente se realiza la instalación de software adicional.

```
sudo mv jdk-<version> /opt
```

- E. Configurar las variables de entorno **JAVA_HOME** y **PATH**, agregar las siguientes líneas en el archivo **/etc/bash.bashrc**

```
#Entorno Java
export JAVA_HOME=/opt/jdk-<version>
```

```
export PATH=${JAVA_HOME}/bin:${PATH}
```

- F. Para que la configuración tome efecto de forma segura, reiniciar sesión. Ejecutar el siguiente comando para verificar resultados.

```
java -version
openjdk version "<version>" yyyy-mm-dd
OpenJDK Runtime Environment XX.X (build 11+28)
OpenJDK 64-Bit Server VM XX.X (build XX+XX, mixed mode)
```

1.1.2. Instalación de Cassandra



Sitio oficial de Cassandra: <https://cassandra.apache.org>

- A. Ejecutar el siguiente comando en la terminal para obtener el archivo **.tar.gz** El archivo se descargara en el directorio donde se ejecute el comando.

```
curl -OL https://dlcdn.apache.org/cassandra/4.1.4/apache-cassandra-4.1.4-bin.tar.gz
```

- B. El siguiente comando se encargará de extraer los archivos dentro del directorio **apache-cassandra-<version>/**

```
tar xzvf apache-cassandra-<version>-bin.tar.gz
```

- Mover la carpeta al directorio **/opt** En esta carpeta es donde generalmente se realiza la instalación de software adicional.

```
sudo mv apache-cassandra-<version> /opt
```

1.2. Iniciando Cassandra

Ejecutar el siguiente comando para iniciar Cassandra. Se autentica con el usuario actual de Linux.

```
cd /opt/apache-cassandra-<version>/ && bin/cassandra
```

El proceso puede tardar algunos segundos y mostrar algunos mensajes.

```
INFO [main] 2023-03-02 17:09:45,261 ColumnFamilyStore.java:484 -
```



```

Initializing system_auth.roles
INFO [main] 2023-03-02 17:09:45,268 ColumnFamilyStore.java:484 - Initializing
system_auth.role_members
INFO [main] 2023-03-02 17:09:45,277 ColumnFamilyStore.java:484 - Initializing
system_auth.role_permissions
INFO [main] 2023-03-02 17:09:45,283 ColumnFamilyStore.java:484 - Initializing
system_auth.resource_role_permissions_index
INFO [main] 2023-03-02 17:09:45,291 ColumnFamilyStore.java:484 - Initializing
system_auth.network_permissions
INFO [main] 2023-03-02 17:09:45,296 StorageService.java:1789 - JOINING: Finish
joining ring
INFO [main] 2023-03-02 17:09:45,303 ColumnFamilyStore.java:1009 - Enqueuing
flush of system.local, Reason: INTERNALLY_FORCED, Usage: 0.596KiB (0%) on-heap,
0.000KiB (0%) off-heap
INFO [PerDiskMemtableFlushWriter_0:1] 2023-03-02 17:09:45,397 Flushing.java:145
- Writing Memtable-local@677991700(0.084KiB serialized bytes, 4 ops, 0.596KiB
(0%) on-heap, 0.000KiB (0%) off-heap), flushed range = [null, null)

```

Para verificar que se ha iniciado Cassandra de forma correcta se utiliza el siguiente comando.

```
bin/nodetool status
```

La columna de Status debe reportar (Up/Down)

```

Datacenter: datacenter1
=====
Status=Up/Down
| / State=Normal/Leaving/Joining/Moving
-- Address     Load      Tokens  Owns (effective)  Host ID          Rack
UN 127.0.0.1   104.33 KiB  16        100.0%           f228e133-dad3-4aea-b331-758f0716a843  rack1

```

Finalmente, para conectarse al nodo disponible

```
bin/cqlsh localhost
```

```

Connected to Test Cluster at localhost:9042
[cqlsh 6.1.0 | Cassandra 4.1.0 | CQL spec 3.4.6 | Native protocol v5]
Use HELP for help.
cqlsh>

```



Cassandra almacena sus datos en tablas, cada tabla consiste de registros y columnas. CQL (Cassandra Language Query) se emplea para realizar el acceso a los datos.

Para utilizar CQL es necesario conectarse a un cluster utilizando alguno de los siguientes métodos:

- Utilizando un shell llamado `cqlsh`.
- Utilizando un *client driver* para Cassandra.

En la sección anterior se utilizó `cqlsh`, para saber en que cluster nos encontramos actualmente se ejecuta la siguiente sentencia.

```
cqlsh> select cluster_name, listen_address from system.local;  
  
cluster_name | listen_address  
-----+-----  
Test Cluster | 127.0.0.1  
  
(1 rows)
```

En caso de que no se especifique el nombre del cluster en el comando `/bin/cqlsh`, por default es `localhost`. Para salir de la base se ejecuta el comando `exit`.

1.3. Modelado de datos en Cassandra

El modelado de datos de Cassandra no es el modelado clásico que se realiza a través de los conceptos de modelo relacional en el que se realiza la identificación de entidades y relaciones. Las tablas son normalizadas, y se emplean FKs para hacer referencia a otras tablas. Las consultas que se realizan en este modelo están **orientadas o dirigidas a la estructura de las tablas**. Los datos relacionados se consultan a través de operaciones **join**.

En Cassandra el modelo de datos está dirigido a las consultas, no a la estructura de los datos. Los patrones de acceso a datos así como las consultas que son requeridas por la aplicación determinan la estructura y organización de los datos. Dicha estructura se emplea para diseñar a su vez la estructura de las **tablas**. En resumen: **Modelar en torno a las consultas**.

- Generalmente cada consulta accede a una sola tabla. Esto significa que el diseño de las tablas se realiza para hacer que dicha consulta acceda a los datos de forma rápida, sin la necesidad de consultar otras tablas. Las operaciones Join **no** están soportadas en Cassandra.
- Cada tabla puede contener datos de varias entidades que son requeridas por la consulta. Esto implica una mejora importante en cuanto a reducción de tiempos ya que todos los datos se encuentran en el mismo lugar.

- Debido a que una entidad puede asociarse con otras entidades y una consulta puede requerir acceder a varias entidades relacionadas entre sí, es posible que **una misma entidad pueda ser incluida en múltiples tablas.**

1.3.1. Particiones

- En Cassandra los datos se almacenan en los diferentes nodos de un clúster. Se emplea el concepto de **partition key** para distribuir los registros de las tablas en un determinado nodo empleando la técnica de **hashing distribution..**
- Todos los datos de una partición siempre se encuentran en un mismo nodo.
- En una consulta siempre se debe especificar el valor de la partition key.
- Cada partición contiene múltiples registros que son agrupados con base al valor de la partition key.
- El partition key se forma a partir del primer atributo de la llave primaria de una tabla.

```
create table t (
    id int,
    k int,
    v text,
    primary key (id)
);
```

En el ejemplo anterior, se emplea el atributo **id** para crear la **partition key**.

Si la tabla define una PK compuesta, el segundo atributo se le conoce como **clustering key**, y se emplea para organizar y ordenar el registro dentro de la partición.

Tanto partition keys como clustering keys pueden ser formadas por más de una columna.

```
create table t (
    id1 int,
    id2 int,
    c1 text,
    c2 text
    k int,
    v text,
    primary key ((id1,id2),c1,c2)
);
```

En el ejemplo, **id1** y **id2** se agrupan entre paréntesis para formar una *partition key*. **c1** y **c2** se emplean para formar a una *clustering key*.

Los atributos restantes de la tabla pueden ser indexados para mejorar el desempeño en cuanto acceso a datos.

Ejemplo

Revista	
PK	<u>id</u>
	nombre
	frecuenciaPublicacion

partition key ←

Q1. Listar todas las revistas incluyendo su nombre y la frecuencia de publicación

Notar que la consulta solo requiere el nombre y frecuencia de publicación. Por lo tanto, la tabla solo define estos 2 atributos.

Ejemplo

Revista	
PK	<u>editorial</u>
CK	id
	frecuenciaPublicacion
	nombre

partition key ←

clustering key ←

Q2. Mostrar los nombres, frecuencia de publicación de las revistas para cada editorial

1.3.2. Diferencias de diseño entre un RDBMS y Cassandra

- No Joins. Si se requiere realizar esta operación se tienen 2 opciones: Realizarla del lado de la aplicación, o realizar desnormalización. La segunda opción es preferida por Cassandra.
- No existe integridad referencial. Generalmente en Cassandra se agrega una columna tipo ID para relacionarse con otras entidades, pero sin la existencia de integridad referencial.
- Desnormalización. En Cassandra se obtiene una mejor funcionalidad cuando los datos están desnormalizados. A partir de la versión 3.0 Cassandra incorpora soporte de **materialized views <materialized-views>** las cuales permiten crear múltiples vistas desnormalizadas. Cassandra se encarga de administrar estas vistas, mantenerlas sincronizadas con las tablas base.
- Query-first design. En el modelo relacional se inicia con el diseño conceptual, se identifican entidades y se establecen relaciones. Las consultas se adaptan al modelo realizado, sin importar que se requieran múltiples tablas y operaciones Join. En Cassandra no se inicia con el modelo de datos, se inicia con un **query model**

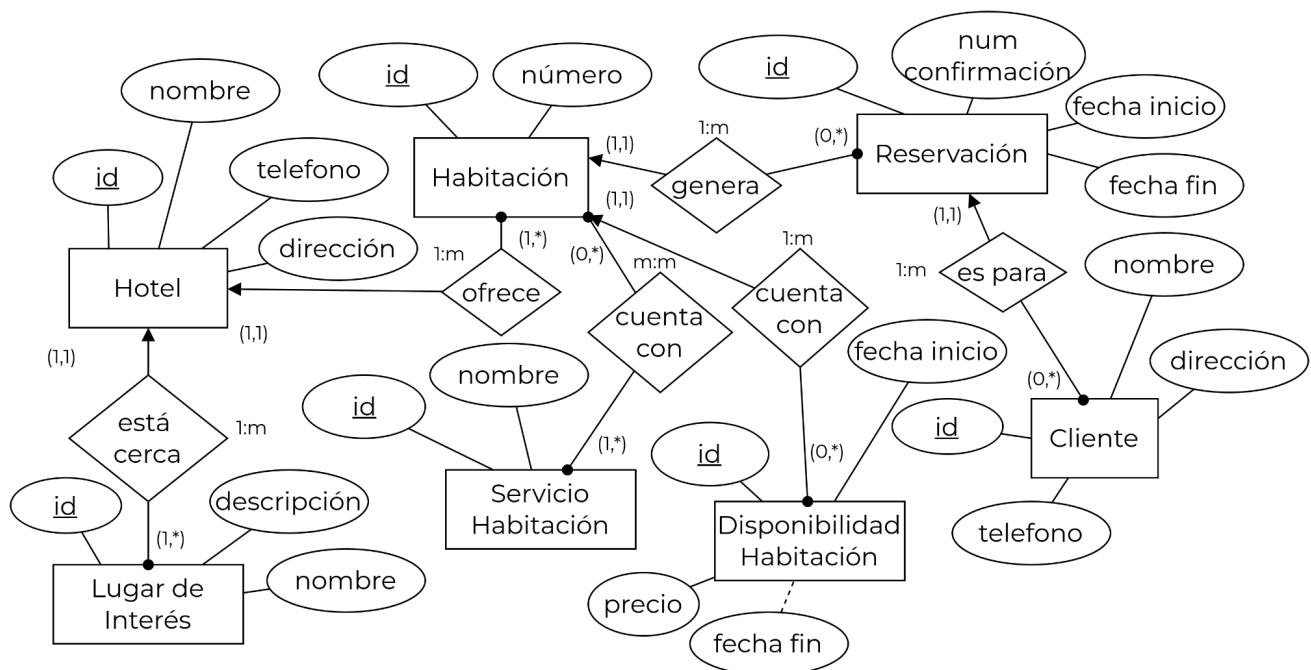
(modelo de consultas). Primero se diseñan las consultas y los datos se adaptan a dichas consultas, generalmente una tabla por consulta.

1.3.3. Diseñar para realizar un almacenamiento óptimo

- Los datos de las tablas en Cassandra se almacenan en diferentes archivos. Cada tabla puede incluir una columna que permita relacionar sus datos con otras entidades.
- Un aspecto importante del diseño es minimizar el número de particiones que se deben escanear para resolver una consulta. Cada partición se encuentra en un **solo** sitio. De lo anterior, una consulta que hace uso de una sola partición representa la mejor opción en cuanto a desempeño.

1.3.4. Diseño conceptual en RDBMS

Para entender el proceso de modelado en Cassandra, considerar el siguiente modelo ER asociado con Hoteles y reservaciones el cual muestra un diseño típico orientado a la estructura de las entidades empleado en los RDBMS



1.3.5. Diseño orientado a las consultas

Con base al modelo ER anterior, suponer que como resultado del proceso de análisis con el usuario final, la aplicación de Hoteles requiere realizar las siguientes consultas:

- Q1. Encontrar hoteles que se encuentren cerca de un cierto lugar de interés.
- Q2. Obtener los datos de un cierto hotel, mostrar nombre y ubicación.

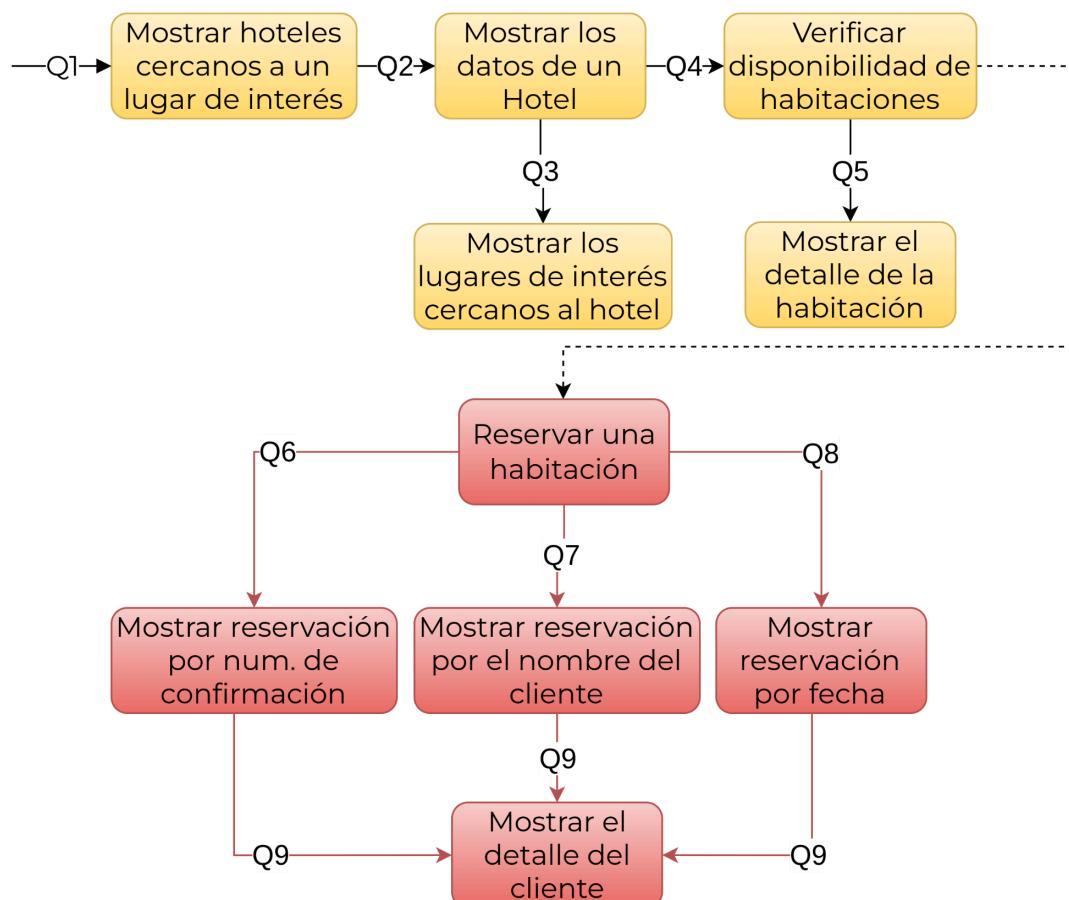
- Q3. Encontrar los puntos de interés para un hotel en específico.
- Q4. Encontrar una habitación disponible para un cierto periodo de tiempo.
- Q5. Obtener los servicios disponibles para una habitación.

Otro conjunto de respuestas asociadas al acceso de las reservaciones por parte de los clientes:

- Q6. Mostrar los datos de la reserva con base en su número de confirmación.
- Q7. Mostrar los datos de la reserva por hotel, fecha, y nombre del cliente.
- Q8. Mostrar los datos de la reserva por el nombre del cliente.
- Q9. Mostrar el detalle de un cliente.

Observar que todas estas consultas fueron obtenidas a partir del análisis de un **workflow** o flujo de trabajo, en este caso, el proceso que realiza un cliente para realizar una reserva.

Los flujos de trabajo pueden ser representados por diagramas divididos por acciones. Cada acción. Para pasar de una acción a otra se emplea un conector y su correspondiente consulta **Qn** que permite realizar la transición.



Cada caja de texto en el diagrama representa un paso del **workflow** o flujo de trabajo. Cada paso realiza una actividad que desbloquea a la siguiente. Por ejemplo, El primer

paso *Mostrar hoteles cercanos a un lugar de interés* le permite a la aplicación conocer los datos de los lugares, así como los identificadores de los hoteles. Estos identificadores serán empleados por Q2 para obtener el detalle del hotel.

1.3.6. Diseño lógico en Cassandra

Una vez que se ha realizado el diseño orientado o dirigido por las consultas, y se ha realizado su definición, el siguiente paso es el diseño de las tablas. Para ello será necesario la creación de un modelo lógico que contenga una tabla por cada consulta.

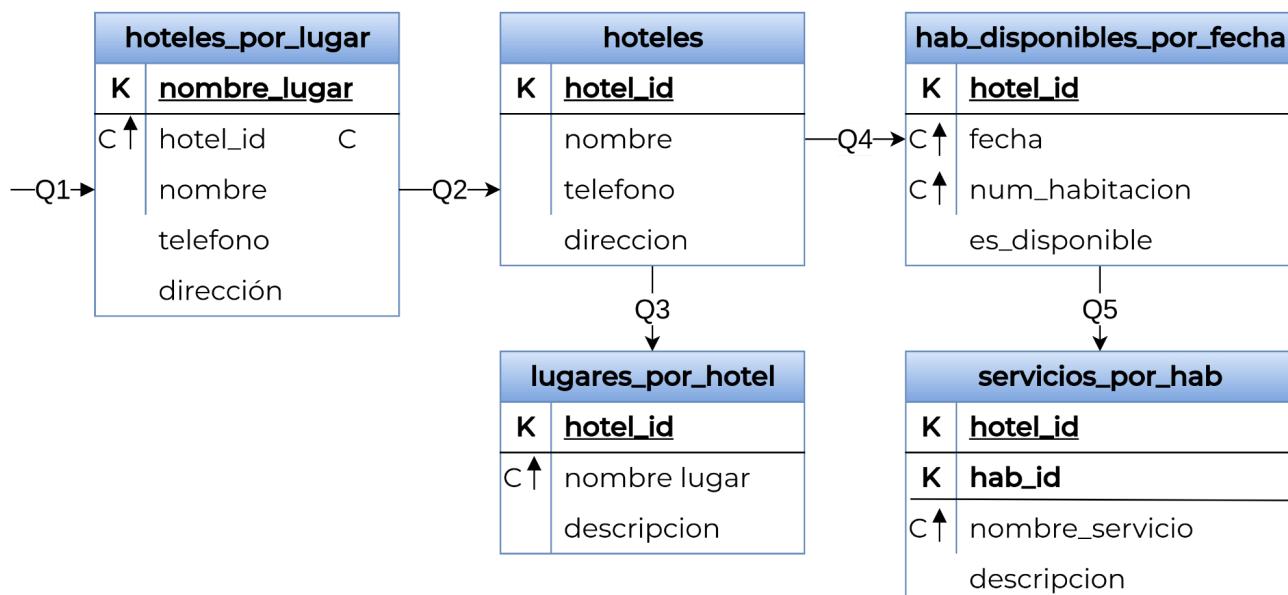
El nombre de la tabla se establece al identificar el nombre de la entidad principal o primaria que contiene los datos de la consulta Q. Si la consulta incluye predicados de los atributos de la tabla, se puede emplear la palabra **por**. Por ejemplo: **clientes_por_email**

El siguiente paso es la asignación de la **llave primaria** de la tabla así como de columnas que actuarán como de **clustering columns**. En subconjunto estos atributos deben garantizar unicidad y ordenamiento.

El diseño de la llave primaria es muy importante debido a que sus valores determinarán la cantidad de datos que se almacenarán en cada partición, así como su organización en disco.

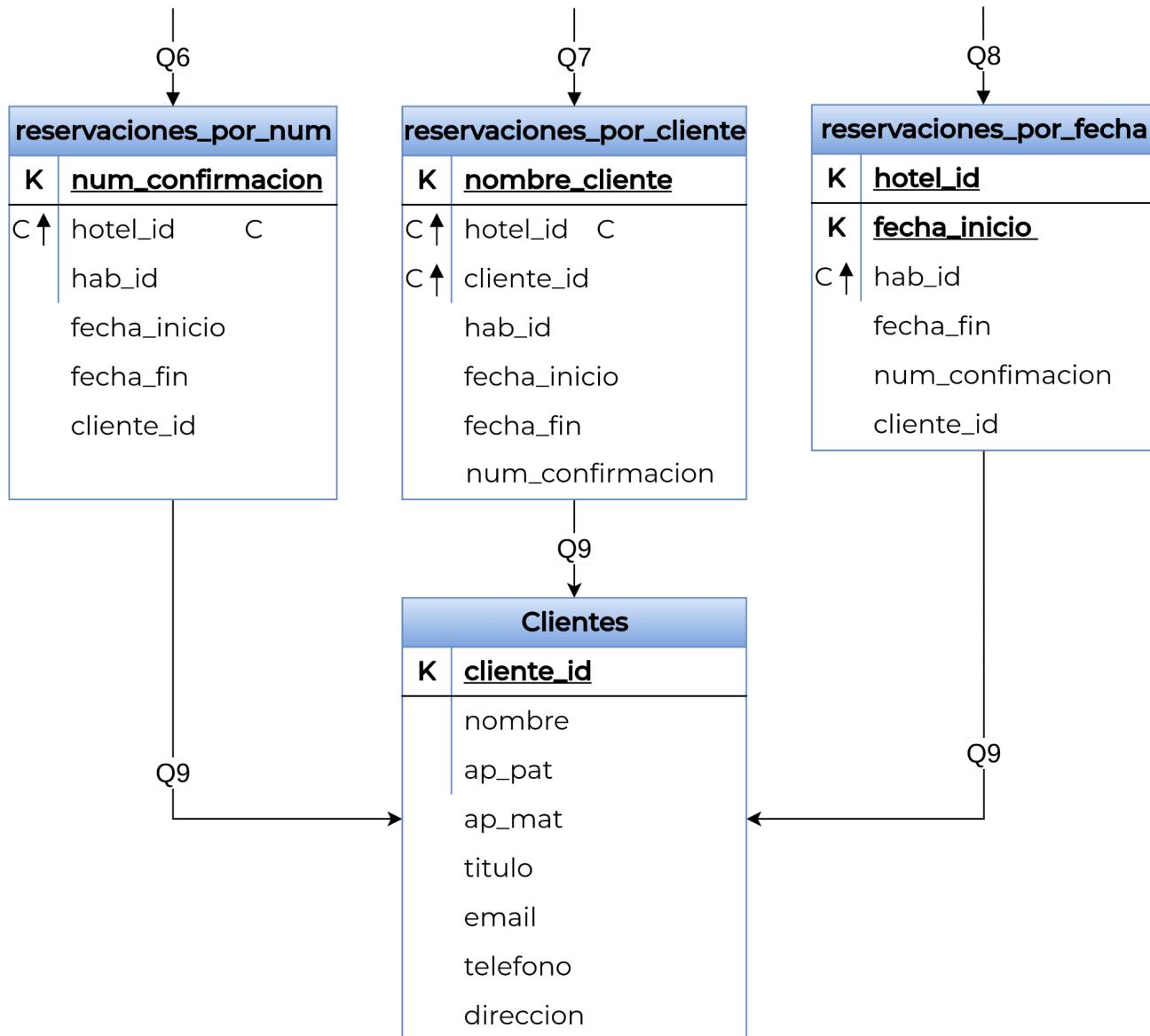
La tabla puede contener atributos adicionales requeridos por la consulta. Estos atributos pueden marcarse como **static** (sus valores son compartidos por todos los registros de una misma partición).

El siguiente diagrama representa al diseño lógico para Cassandra del caso de estudio de hoteles.



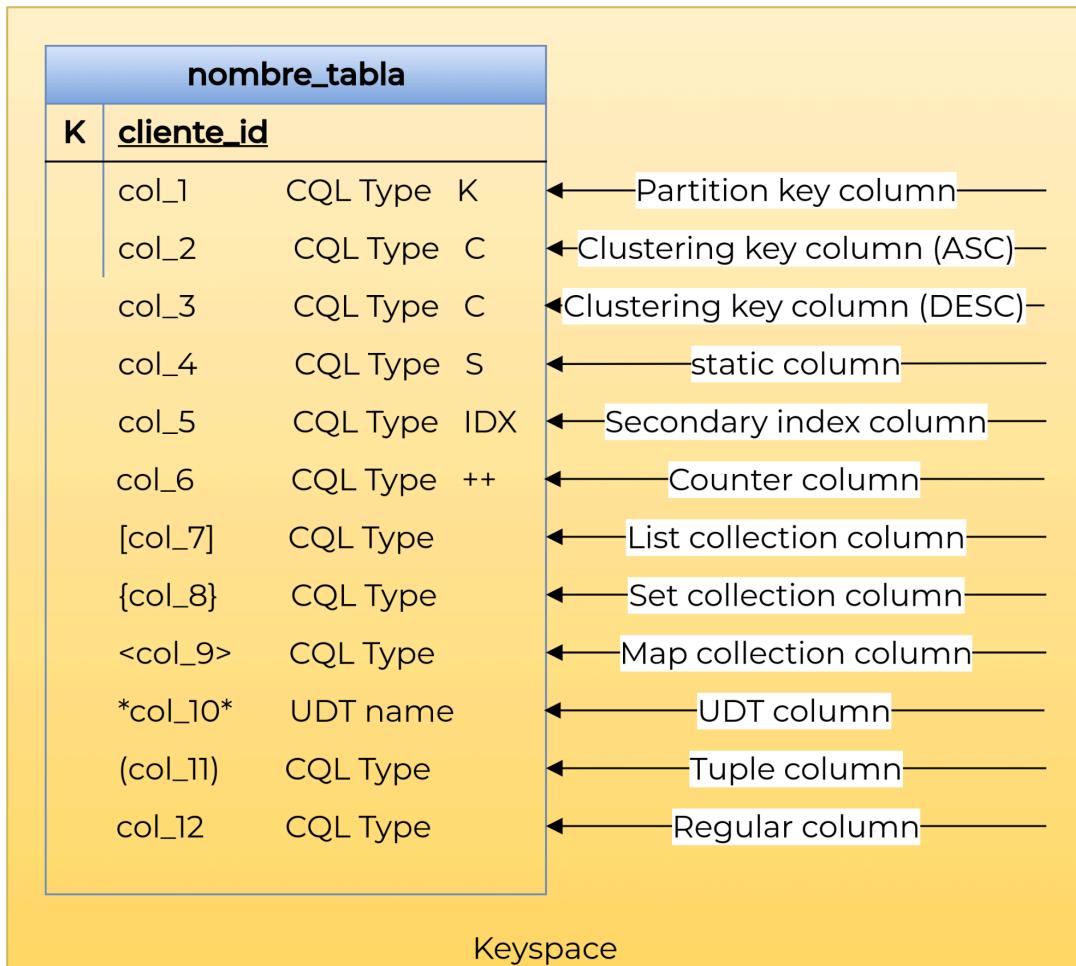
Las columnas que forman a la PK se identifican por la letra **K** (partition columns). Los valores identificados por la letra **C↑** y por **C↓** representan *clustering columns*

Para la parte del acceso a las reservaciones se tiene el siguiente modelo



1.3.7. Diseño físico

Posterior al diseño lógico, se realiza el diseño físico. Una de las actividades a realizar es la asignación de los tipos de datos de cada columna. Cassandra ofrece diversos tipos de datos: CQL types (Cassandra Query Language Types). Las tablas deberán estar contenidos en un KeySpace



En una aplicación pudieran existir varios keyspace, cada uno contiene tablas con funcionalidades similares. El siguiente ejemplo muestra el modelo lógico con su asignación de tipos de datos

hoteles_por_lugar			hoteles			hab_disponibles_por_fecha			*address*		
K	nombre_lugar	text	K	hotel_id	text	K	hotel_id	text	K	hotel_id	text
C	hotel_id	text		nombre	text	C↑	fecha	date	C	calle	text
	nombre	text		telefono	text	C↑	num_hab	smallint		ciudad	text
	telefono	text		*direccion*	address		es_disponible	boolean		estado	text
	direccion	address								CP	text
										Pais	text

1.3.8. Evaluación y refinamiento de modelos de datos

Posterior al diseño físico, se realiza un par de pasos más para evaluar y refinar el diseño lógico para obtener el mejor desempeño posible

1.3.8.1. Calcular el tamaño de las particiones

El tamaño de una partición depende del número de valores N_v (cells o celdas) que contenga. La capacidad máxima que soporta Cassandra es de 2 billones. En cuanto a espacio de almacenamiento, el tamaño máximo es de 100M, idealmente debe ser de alrededor de 10MB por partición. Para calcular el número de particiones que pudieran requerirse se puede calcular a través de la siguiente fórmula

$$N_v = N_r(N_c - N_{pk} - N_s) + N_s$$

El número de valores N_v o celdas en la partición es igual al número de columnas estáticas N_s más el producto del número de registros N_r con el número de valores por registro ($N_c - N_{pk} - N_s$), donde:

N_c Número de columnas

N_s Número de columnas estáticas.

N_{pk} Número de columnas que integran a la PK incluyendo clustering keys

N_r Número de registros.

La expresión que está en paréntesis se le conoce como *número de valores por registro*.

Ejemplo

Calcular el tamaño de la partición para la tabla **hab_disponibles_por_fecha**:

$N_v = N_r(4 - 3 - 0) + 0 = N_r$, es decir, el tamaño de una partición corresponde con el número de registros.

¿Cuántos registros tendría la tabla? Suponer que se desea guardar 2 años de disponibilidad diaria y se tienen 5000 hoteles con un promedio de 100 habitaciones.

Debido a que existirá una partición para cada **hotel_id**, el tamaño de la partición será

$N_r = 100 * 730 = 73,000$ *registros por partición*

1.3.8.2. Calcular el tamaño del espacio de almacenamiento

Este cálculo se refiere a estimar el espacio de almacenamiento requerido para almacenar los datos de una tabla S_t . Se emplea la siguiente fórmula

$$S_t = \sum_i sizeOf(c_{ki}) + \\ \sum_j sizeOf(c_{sj}) + N_r (\sum_k sizeOf(c_{rk}) + \sum_l sizeOf(c_{cl})) + N_v * sizeOf(t_{avg})$$

c_k : partition key columns

c_s : static columns

c_r : columnas regulares

c_c : clustering columns

t_{avg} : tamaño promedio de los metadatos de cada celda de la tabla. Aprox, 8 bytes

$sizeOf$: tamaño en bytes

Ejemplo

Para la tabla **hab_disponibles_por_fecha** se tiene el siguiente cálculo. Asumir que se cuenta con 73,000 registros, suponer que **hotel_id** ocupa 5 bytes de espacio (5 caracteres). Para las columnas regulares se tiene fecha, **num_hab**, **es_disponible** con tamaños de 4, 2 y 1 bytes respectivamente.

$$S_t = 5 + 0 + 73000 * (4 + 2 + 1) + 73000 * 8$$

$$S_t = 5 + 0 + 0.51MB + 0.58MB$$

Recordar que cada partición debe incluirse completamente en un solo nodo.

Ejemplo

Suponer que se tienen 10 registros de libros para ser almacenados en una tabla llamada **libros**.

ISBN	Título	Autor	año publicación	Editorial
439023483	Anna Karenina	Leo Tolsoy	2015	Fingerprint Publishing
439554934	Harry Potter and the Chamber of secrets	J.K. Rowling	2014	Bloomsbury
316015849	Harry Potter and the Philosopher's Stone	J.K. Rowling	2014	Bloomsbury
61120081	Harry Potter and the Prisoner of Azkaban	J.K. Rowling	2014	Bloomsbury
743273567	Lord of the rings: The fellowship of the ring	J.R.R. Tolkien	2012	Bloomsbury
525478817	Lord of the rings: The return of the king	J.R.R. Tolkien	2012	Harpercollins
618260307	Lord of the rings: The two towers	J.R.R. Tolkien	2013	Harpercollins
316769177	The adventures of Tom Sawyer	Mark Twain	2012	Harpercollins
1416524797	The mysterious Stranger	Mark Twain	2012	Amazon
679783261	War and Peace	Jane Austen	2014	Amazon
1594480001	The Kite Runner	Leo Tolsoy	2014	Fingerprint Publishing

Los siguientes ejemplos muestran el impacto de seleccionar diferentes columnas como partition key

partition key = autor

439554934	Harry Potter and the Chamber of secrets	J.K. Rowling	2014	Bloomsbury
316015849	Harry Potter and the Philosopher's Stone	J.K. Rowling	2014	Bloomsbury
61120081	Harry Potter and the Prisoner of Azkaban	J.K. Rowling	2014	Bloomsbury
743273567	Lord of the rings: The fellowship of the ring	J.R.R. Tolkien	2012	Bloomsbury
525478817	Lord of the rings: The return of the king	J.R.R. Tolkien	2012	Harpercollins
618260307	Lord of the rings: The two towers	J.R.R. Tolkien	2013	Harpercollins
316769177	The adventures of Tom Sawyer	Mark Twain	2012	Harpercollins
1416524797	The mysterious Stranger	Mark Twain	2012	Amazon
679783261	War and Peace	Jane Austen	2014	Amazon
1594480001	The Kite Runner	Leo Tolstoy	2014	Fingerprint Publishing
439023483	Anna Karenina	Leo Tolstoy	2015	Fingerprint Publishing

Partition key = editorial

439554934	Harry Potter and the Chamber of secrets	J.K. Rowling	2014	Bloomsbury
316015849	Harry Potter and the Philosopher's Stone	J.K. Rowling	2014	Bloomsbury
61120081	Harry Potter and the Prisoner of Azkaban	J.K. Rowling	2014	Bloomsbury
743273567	Lord of the rings: The fellowship of the ring	J.R.R. Tolkien	2012	Bloomsbury
525478817	Lord of the rings: The return of the king	J.R.R. Tolkien	2012	Harpercollins
618260307	Lord of the rings: The two towers	J.R.R. Tolkien	2013	Harpercollins
316769177	The adventures of Tom Sawyer	Mark Twain	2012	Harpercollins
1416524797	The mysterious Stranger	Mark Twain	2012	Amazon
679783261	War and Peace	Jane Austen	2014	Amazon
1594480001	The Kite Runner	Leo Tolstoy	2014	Fingerprint Publishing
439023483	Anna Karenina	Leo Tolstoy	2015	Fingerprint Publishing

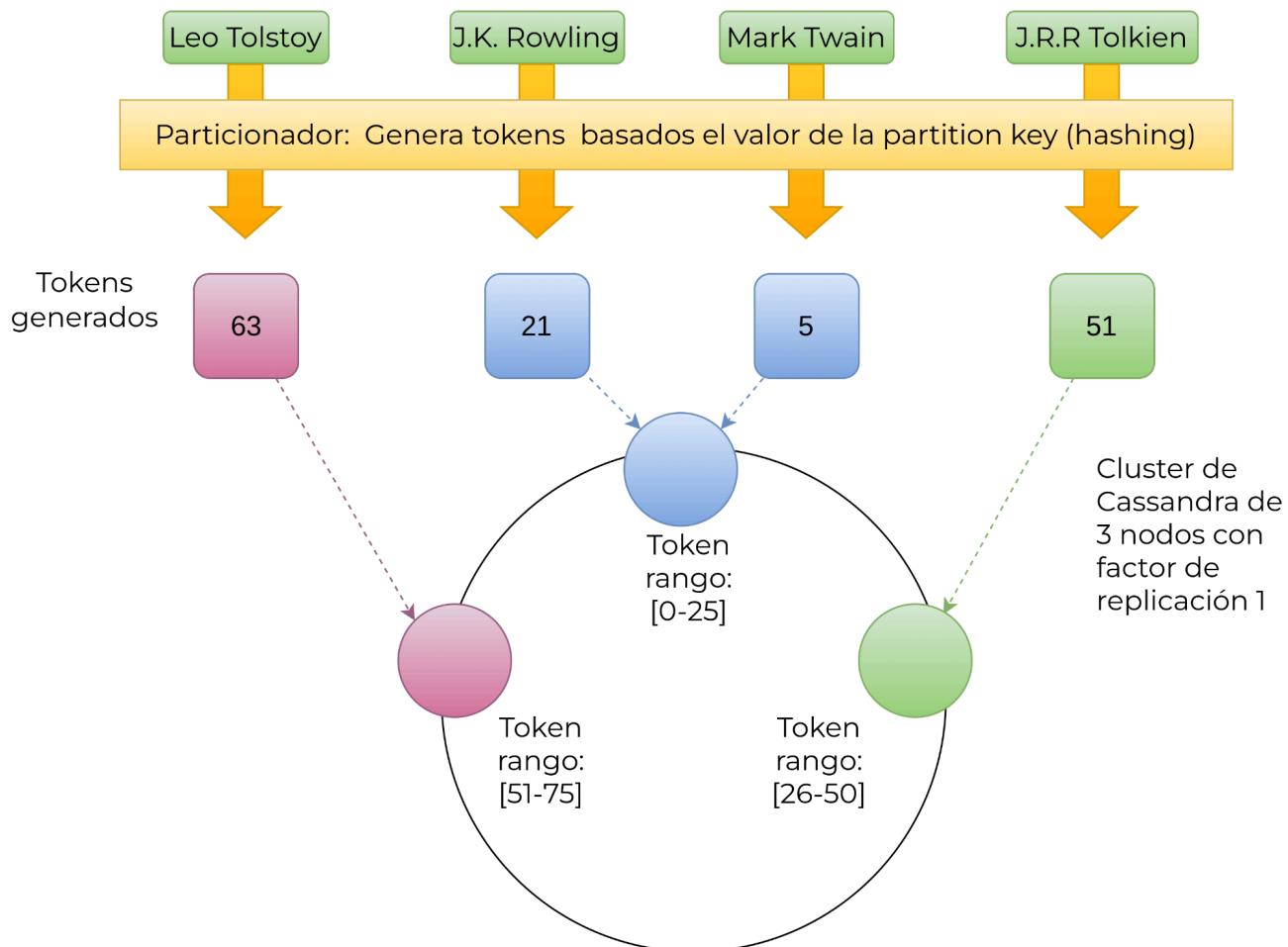
partition key = año publicación, editorial

1416524797	The mysterious Stranger	Mark Twain	2012	Amazon
316769177	The adventures of Tom Sawyer	Mark Twain	2012	Harpercollins
743273567	Lord of the rings: The fellowship of the ring	J.R.R. Tolkien	2012	Bloomsbury
525478817	Lord of the rings: The return of the king	J.R.R. Tolkien	2012	Harpercollins
618260307	Lord of the rings: The two towers	J.R.R. Tolkien	2013	Harpercollins
679783261	War and Peace	Jane Austen	2014	Amazon
439554934	Harry Potter and the Chamber of secrets	J.K. Rowling	2014	Bloomsbury
316015849	Harry Potter and the Philosopher's Stone	J.K. Rowling	2014	Bloomsbury
61120081	Harry Potter and the Prisoner of Azkaban	J.K. Rowling	2014	Bloomsbury
1594480001	The Kite Runner	Leo Tolstoy	2014	Fingerprint Publishing
439023483	Anna Karenina	Leo Tolstoy	2015	Fingerprint Publishing

Esquemas incorrectos de particionado:

- Muy pequeña: una partición por renglón
- Muy grande: su tamaño no permite almacenarla en un solo nodo.

La siguiente imagen muestra la forma en la que Cassandra realiza la distribución de datos. Se considera al autor como partition key empleando los datos del ejemplo anterior



Cada nodo administra un rango de tokens

- Nodo 1: Tokens 1 - 25
- Nodo 2: Tokens 25 - 70
- Nodo 3: Tokens 51 - 75

Cuando se inserta un registro, se obtiene el valor del partition key, se emplea un algoritmo de hashing para generar el token. Este mecanismo de almacenamiento distribuido de datos le permite a Cassandra responder de forma rápida y eficiente.

Con base al valor del partition key en la consulta, Cassandra podrá conocer de forma directa al nodo donde se encuentra el dato.

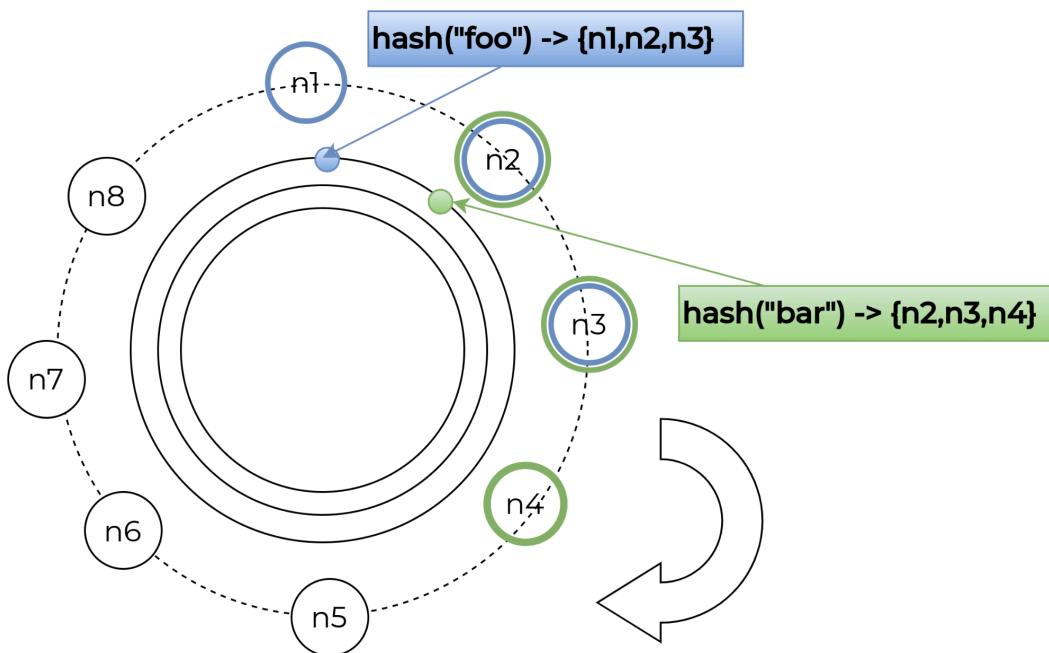
1.4. Arquitectura básica de Cassandra

En los ejemplos anteriores se empleó un factor de replicación RF de 1 por simplicidad, así como los rangos de tokens con valores muy bajos.

- En clusters reales el factor de replicación ideal es de 3.
- El rango de tokens es de -2^{63} a $2^{63}-1$
- Cada nodo puede atender varios rangos de tokens empleando **vnodes**.

$range(t1, t2] \rightarrow \{n_2, n_3, n_4\}$
 $range(t2, t3] \rightarrow \{n_3, n_4, n_5\}$

Posterior al cálculo del token, se realiza la asignación de nodo y sus réplicas en nodos consecutivos realizando un recorrido en sentido a las manecillas del reloj



1.5. Definición del esquema de la base de datos

Posterior a la definición y refinamiento del modelo físico, el siguiente paso es la implementación del esquema de la BD en CQL (Cassandra Query Language)

CQL almacena los datos en tablas. Cada tabla define un esquema que a su vez define la forma o estructura de los datos. Las tablas se organizan en KeySpaces. Las configuraciones que se le apliquen al KeySpace se aplican de forma directa a todas las tablas que contiene. Uno de los parámetros más importantes es el factor de replicación FR.

1.5.1. Creación de un KeySpace y objetos

Para crear un KeySpace se utiliza la siguiente sentencia:

```
create keyspace [ if not exists ] keyspace_name with options
```

Las opciones son:

- **replication**
- **durable_writes** Habilita o deshabilita el uso de commit logs

Ejemplo

```
create keyspace k1
with replication = {'class': 'SimpleStrategy', 'replication_factor' : 1};
```

Para cambiarse entre KeySpaces:

```
USE <nombre_keyspace>
```

Observar que en el prompt aparece el nombre del keyspace en el que nos encontramos actualmente.

```
cqlsh:k1>
```

Ejemplo

- A. Crear un Keyspace para el caso de estudio de hoteles

```
create keyspace hotel
with replication = {'class': 'SimpleStrategy', 'replication_factor' : 1};
```

- B. Consultar keyspaces

```
desc keyspaces
```

- C. Modificar el nivel de replicación a 3 (Solo ejecutar cuando se tenga un cluster).

```
alter keyspace hotel
with replication = {'class': 'SimpleStrategy', 'replication_factor' : 3};
```

- D. Cambiarse al keyspace **hotel**.

```
USE hotel
```

- E. Definir el esquema de las tablas para el sistema de reservaciones de hotel

```
create type hotel.direccion (
    calle text,
    ciudad text,
    estado text,
    codigo_postal text,
    pais text
);

create table hotel.hoteles_por_lugar (
    nombre_lugar text,
    hotel_id text,
    nombre_hotel text,
    telefono_hotel text,
    direccion frozen<direccion>,
    primary key ((nombre_lugar), hotel_id)
) with comment = 'Q1. Encontrar hoteles cerca de un lugar de interés'
and clustering order by (hotel_id asc) ;

create table hotel.hoteles (
    id text primary key,
    nombre text,
    telefono text,
    direccion frozen<direccion>,
    lugares_interes set <text>
) with comment = 'Q2. Encontrar el detalle de un hotel';

create table hotel.puntos_interes_por_hotel (
    nombre_lugar text,
    hotel_id text,
    descripcion text,
    primary key ((hotel_id), nombre_lugar) )
with comment = 'Q3. Encontrar puntos de interes cercanos a un lugar';

create table hotel.habs_disponibles_por_fecha (
    hotel_id text,
    fecha date,
    num_habitacion smallint,
    es_disponible boolean,
    primary key ((hotel_id), fecha, num_habitacion) )
with comment = 'Q4. Encontrar habitaciones disponibles por fecha';
```

```
create table hotel.servicios_por_habitacion (
    hotel_id text,
    num_habitacion smallint,
    nombre_servicio text,
    descripcion text,
    PRIMARY key ((hotel_id, num_habitacion), nombre_servicio) )
    with comment = 'Q5. Encontrar servicios por habitacion';
```

- La cláusula `frozen` indica que los elementos de una colección no podrán ser modificados posterior a su creación. Si se desea realizar cambios, se deberá reemplazar toda la colección en lugar de modificar alguno de sus elementos.

F. Mostrar las tablas del keyspace

```
select table_name
from system_schema.tables where keyspace_name = 'hotel';
```

G. Crear el keyspace para el esquema de reservaciones

```
create keyspace reservacion with replication =
{'class': 'SimpleStrategy', 'replication_factor' : 1};
```

H. Crear las tablas para las consultas de reservaciones

```
use reservacion

create type direccion (
    calle text,
    ciudad text,
    estado text,
    codigo_postal text,
    pais text
);
```

```
create table reservaciones_por_num_confirmacion (
    num_confirmacion text,
    hotel_id text,
    fecha_inicio date,
    fecha_fin date,
    num_habitacion smallint,
    cliente_id uuid,
    primary key (num_confirmacion) )
with comment = 'Q6. find reservations by confirmation number';

create table reservaciones_por_hotel_fecha (
    hotel_id text,
    fecha_inicio date,
    fecha_fin date,
    num_habitacion smallint,
    num_confirmacion text,
    cliente_id uuid,
    primary key ((hotel_id, fecha_inicio), num_habitacion) )
with comment = 'Q7. Encontrar reservaciones por hotel y fecha';

create table reservaciones_por_cliente (
    nombre_completo text,
    hotel_id text,
    fecha_inicio date,
    fecha_fin date,
    num_habitacion smallint,
    num_confirmacion text,
    cliente_id uuid,
    primary key ((nombre_completo), hotel_id) )
with comment = 'q8. find reservations by guest name';

create table clientes (
    cliente_id uuid primary key,
    nombre text,
    ap_paterno text,
    ap_materno text,
    titulo text,
    emails set,
    num_telefono list,
    direcciones map<text, frozen<direccion>>,
    num_confirmacion text )
with comment = 'Q9. Encontrar clientes';
```

1.5.2. Poblado de datos

Los siguientes ejemplos muestran la sintaxis para agregar algunos registros las tablas de este caso de estudio

- Para `hoteles_por_lugar`

```
insert into hoteles_por_lugar(
    nombre_lugar,hotel_id,nombre_hotel,telefono_hotel,direccion)
values(
    'Xcaret','100','Hilton','33823828',
{
    calle: 'av central',
    ciudad:'Playa del Carmen',
    estado:'Quintana Roo',
    codigo_postal: '82731',
    pais: 'México'
},
);
;
```

- Para `hoteles`

```
insert into hoteles (id,nombre,telefono,direccion,lugares_interes)
values (
    '100',
    'Hilton',
    '33823828',
{
    calle: 'av central',
    ciudad:'Playa del Carmen',
    estado:'Quintana Roo',
    codigo_postal: '82731',
    pais: 'México'
},
    {'Cancun','Xcaret','Tulum'}
);
;
```

- Para `puntos_interes_por_hotel`

```
insert into puntos_interes_por_hotel (nombre_lugar,hotel_id,descripcion)
values(
  'Xcaret',
  '100',
  'Xcaret lugar turístico'
);
```

- Para `reservaciones_por_num_confirmacion`

```
use reservacion

insert into reservaciones_por_num_confirmacion(
  num_confirmacion,hotel_id,fecha_inicio,fecha_fin,num_habitacion,cliente_id)
values(
  '3230',
  '100',
  '2024-01-15',
  '2024-01-20',
  1401,
  uuid()
);
```

- Para clientes

```
insert into clientes (
  cliente_id,nombre,ap_paterno,ap_materno,titulo,emails,num_telefono,
  direcciones,num_confirmacion
) values (
  uuid(), 'Adolfo', 'López', 'Ruiz', 'Dr',
  {'m1@mail.com','ado@gmail.com','ado@unnam.mx'},
  ['55678982','5524343211'],
  {'dir_1' :
    { calle:'av central',
      ciudad:'CDMX',
      estado:'CDMX',
      codigo_postal:'02342',
      pais: 'mx'
    },
    'dir_2':
    { calle : 'sur 39',
      ciudad : 'CDMX',
      estado : 'CDMX',
      }
  }
);
```

```

        codigo_postal: '10342',
        pais: 'mx'
    },
},
'0000923'
);

```

1.5.3. Actividades a realizar

C1. Incluir en el reporte el siguiente código

- Insertar un registro para cada una de las tablas del caso de estudio de reservaciones y lugares de interés

1.6. Interactuando con Cassandra desde una aplicación Java

En esta sección se ilustra el proceso para interactuar con Cassandra desde una aplicación Java. Se emplearán las siguientes herramientas:

- IntelliJ Idea como IDE de desarrollo
- Se empleará **Gradle** como herramienta para administrar la compilación, ejecución y administración de las dependencias del código fuente muy utilizada en la industria del desarrollo de software. Para conocer un poco de Gradle, leer el siguiente [enlace](#)
- Se empleará la misma versión del JDK empleada para correr Cassandra

1.6.1. Instalación y configuración de IntelliJ Idea

- A. Realizar la descarga de **IntelliJ IDEA Community Edition** de este [enlace](#). Es posible instalar la versión IntelliJ IDEA Ultimate. Para este caso se requiere obtener una licencia de estudiante en este [enlace](#). Cualquiera de las 2 opciones funciona para efectos del ejercicio.
- B. En una terminal, cambiarse al directorio donde se realizó la descarga, descomprimir el archivo y mover el software a la carpeta `/opt`. No olvidar sustituir `<version>` con el valor correspondiente

```

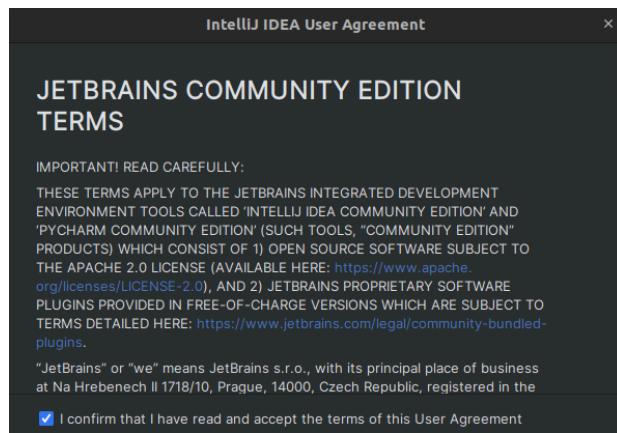
tar -xvf ideaIC-<version>.tar.gz
sudo mv idea-IC-233.14808.21 /opt

```

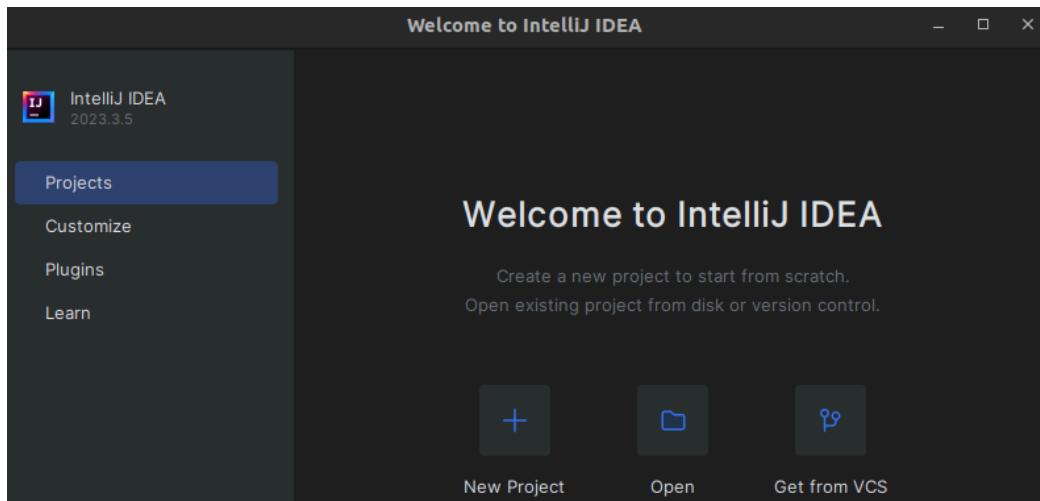
- C. Iniciar IntelliJ IDEA desde la terminal. Se puede crear un acceso directo si así se desea

```
/opt/idea-IC-<version>/bin/idea.sh &
```

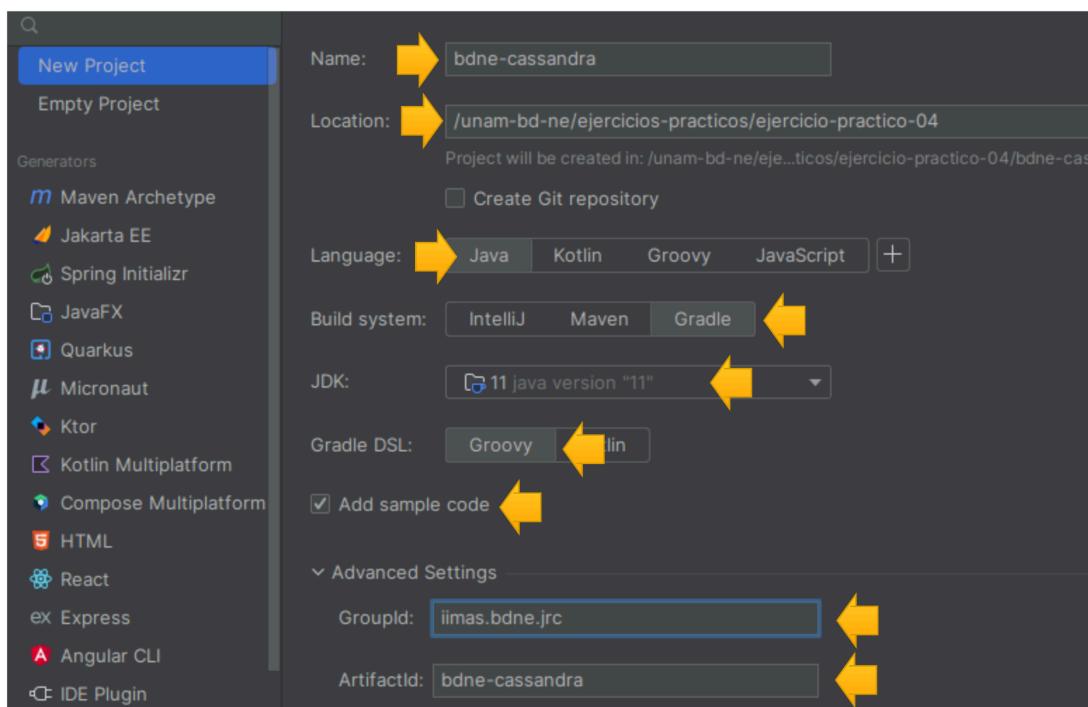
- D. Aceptar los términos de uso



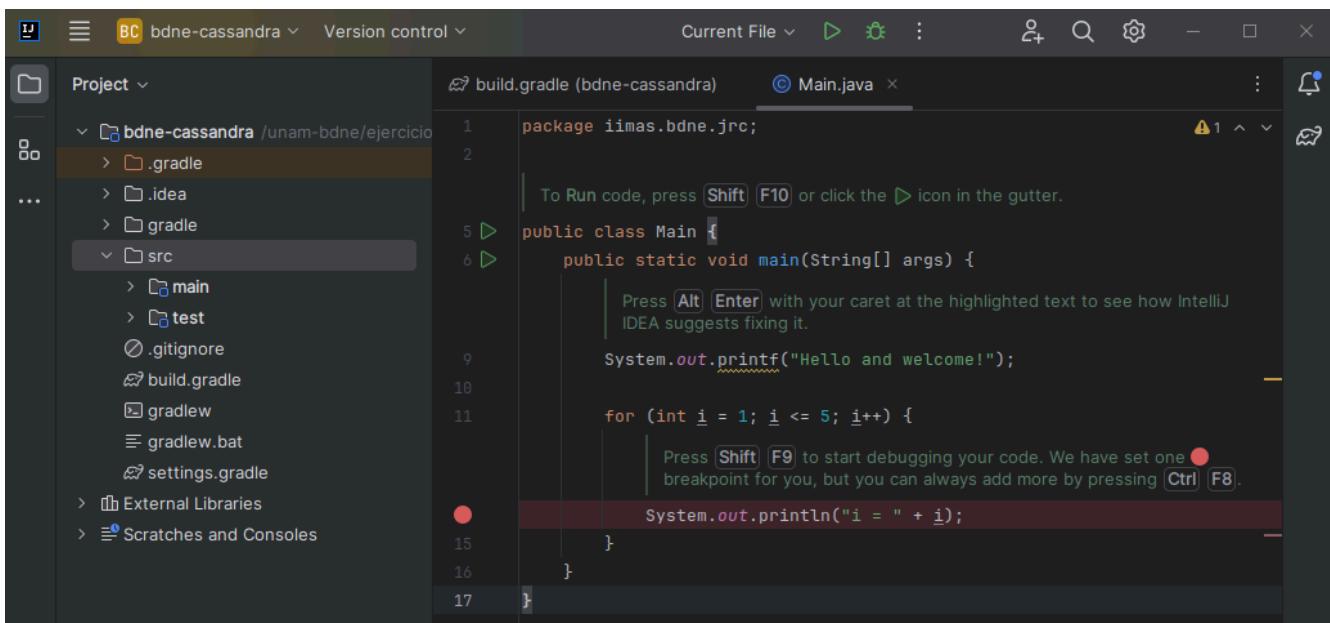
- E. Compartir datos con JetBrains (Data sharing). No es necesario habilitar esta opción.
F. Pantalla de inicio. Seleccionar New Project.



- G. Configurar un nuevo proyecto Gradle con base a las siguientes configuraciones. El proyecto se almacenará en la carpeta de trabajo para este ejercicio práctico. Notar que el valor del atributo groupId tiene la forma **iimas.bdne.<iniciales>**. Actualizar el valor **jrc** con las iniciales de cada estudiante (nombre, apellido paterno y apellido materno)



H. Aparecerá una ventana similar a la siguiente.



- Por default, un proyecto Gradle está formado por la estructura que se muestra del lado izquierdo.
- Las clases Java se ubican dentro de la carpeta **src/main/java**.
- Las clases de prueba, generalmente hacen uso de JUnit, se encuentran dentro de **src/test/java**.
- El archivo de configuración de gradle se llama **build.gradle** y se encuentra en la carpeta raíz del proyecto.

Por default se crea una clase **Main** a modo de ejemplo para validar que el proyecto y su configuración inicial es correcta. La clase solo muestra un mensaje de bienvenida. Se puede ejecutar haciendo click en el botón **play**, en la esquina superior derecha.



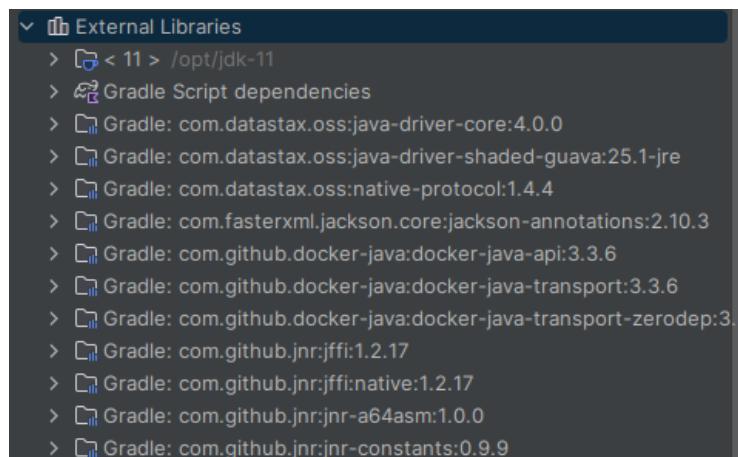
- I. El siguiente paso es la configuración de las dependencias (archivos jar) que requiere el proyecto para trabajar con cassandra
- Abrir el archivo **build.gradle** y agregar las siguientes dependencias

```
dependencies {
    testImplementation 'org.junit.jupiter:junit-jupiter-api:5.8.1'
    testRuntimeOnly 'org.junit.jupiter:junit-jupiter-engine:5.8.1'
    implementation 'com.datastax.oss:java-driver-core:4.1.0'
    implementation 'com.datastax.oss:java-driver-query-builder:4.1.0'
    runtimeOnly 'ch.qos.logback:logback-classic:1.5.0'
    runtimeOnly 'org.slf4j:slf4j-api:2.0.12'
}
```

- Notar que se habilita un ícono en la parte superior derecha. Hacer click en él para aplicar los cambios.



- J. Al aplicar los cambios notar del lado izquierdo en el árbol que contiene la estructura del proyecto, en la sección External dependencies, se deberá apreciar que se han agregado varias dependencias



- K. Crear el directorio `src/main/resources` en caso de no existir y agregar un archivo llamado `logback.xml` con el siguiente contenido

```
<configuration>
    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%d{HH:mm:ss.SSS} %magenta(%-5level) %cyan(%-45logger) : %msg%n</pattern>
        </encoder>
    </appender>
    <root level="WARN">
        <appender-ref ref="STDOUT"/>
    </root>
    <logger name="com.datastax" level="INFO"/>
    <logger name="iimas.bdne" level="DEBUG"/>
</configuration>
```

- A nivel general, este archivo configura el formato de los mensajes que salen en la consola. Existen varios niveles de detalle. Notar que existe un logger llamado `iimas.bdne` que corresponde con la estructura de paquetes donde se encuentran todas las clases de este ejercicio. Para todas estas clases, los mensajes que se enviarán a consola tendrán nivel de `debug`.

L. Crear las siguientes clases

A nivel general, este ejercicio incluye la programación necesaria para crear un Keyspace llamado `video`, una tabla llamada `videos`, la inserción y consulta de datos.

Las clases se organizan en los siguientes paquetes dentro de `iimas.bdne.<iniciales>`

- `iimas.bdne.<iniciales>.domain`
 - `Video`
- `iimas.bdne.<iniciales>.repository`
 - `VideoRepository`
 - `KeyspaceRepository`
- `iimas.bdne.<iniciales>`
 - `Application`
 - `CassandraConnector`

Los siguientes ejemplos no incluyen las sentencias import, agregarlas con la ayuda de IntelliJ IDEA.

Clase `CassandraConnector`

Se encarga de realizar conexiones con la base de datos

```

public class CassandraConnector {
    private CqlSession session;

    public void connect(String node, Integer port, String dataCenter) {
        CqlSessionBuilder builder = CqlSession.builder();
        builder.addContactPoint(new InetSocketAddress(node, port));
        builder.withLocalDatacenter(dataCenter);
        session = builder.build();
    }
    public CqlSession getSession() {
        return this.session;
    }
    public void close() {
        session.close();
    }
}

```

- Clase **KeyspaceRepository** Se encarga de crear y cambiarse a un Keyspace

```

public class KeyspaceRepository {
    private final CqlSession session;

    public KeyspaceRepository(CqlSession session) {
        this.session = session;
    }
    public void createKeyspace(String keyspaceName, int numberOfReplicas) {
        CreateKeyspace createKeyspace = SchemaBuilder.createKeyspace(keyspaceName)
            .ifNotExists()
            .withSimpleStrategy(numberOfReplicas);

        session.execute(createKeyspace.build());
    }
    public void useKeyspace(String keyspace) {
        session.execute("USE " + CqlIdentifier.fromCql(keyspace));
    }
}

```

- Clase **Video**

Las instancias de esta clase representan a cada uno de los registros de la tabla **video**.

```

public class Video {

    private UUID id;
}

```

```
private String title;
private Instant creationDate;

public Video(UUID id, String title, Instant creationDate) {
    this.id = id;
    this.title = title;
    this.creationDate = creationDate;
}
public Video(String title, Instant creationDate) {
    this.title = title;
    this.creationDate = creationDate;
}

public UUID getId() {
    return id;
}
public void setId(UUID id) {
    this.id = id;
}
public String getTitle() {
    return title;
}
public void setTitle(String title) {
    this.title = title;
}
public Instant getCreationDate() {
    return creationDate;
}
public void setCreationDate(Instant creationDate) {
    this.creationDate = creationDate;
}
@Override
public String toString() {
    return "[id:" + id.toString() + ", title:" + title + ", creationDate: " +
creationDate.toString() + "]";
}
```

- Clase **VideoRepository**

Esta clase se encarga de insertar y consultar registros a la tabla videos. Por ahora, se consultan todos los videos existentes y se obtiene una lista.

```

public class VideoRepository {
    private static final String TABLE_NAME = "videos";
    private final CqlSession session;
    public VideoRepository(CqlSession session) {
        this.session = session;
    }
    public void createTable() {
        CreateTable createTable = SchemaBuilder.createTable(TABLE_NAME).ifNotExists()
            .withPartitionKey("video_id", DataTypes.UUID)
            .withColumn("title", DataTypes.TEXT)
            .withColumn("creation_date", DataTypes.TIMESTAMP);
        session.execute(createTable.build());
    }
    public UUID insertVideo(Video video) {
        UUID videoId = UUID.randomUUID();
        video.setId(videoId);
        //construye una sentencia parametrizada
        RegularInsert insertInto = QueryBuilder.insertInto(TABLE_NAME)
            .value("video_id", QueryBuilder.bindMarker())
            .value("title", QueryBuilder.bindMarker())
            .value("creation_date", QueryBuilder.bindMarker());
        SimpleStatement insertStatement = insertInto.build();
        PreparedStatement preparedStatement = session.prepare(insertStatement);
        //asigna los valores de los parámetros (valores de las columnas)
        BoundStatement statement = preparedStatement.bind()
            .setUuid(0, video.getId())
            .setString(1, video.getTitle())
            .setInstant(2, video.getCreationDate());
        //ejecuta la sentencia
        session.execute(statement);
        return videoId;
    }

    public List<Video> selectAll() {
        //genera una sentencia que obtiene todos los videos
        Select select = QueryBuilder.selectFrom(TABLE_NAME).all();
        ResultSet resultSet = session.execute(select.build());
        List<Video> result = new ArrayList<>();
        resultSet.forEach(x -> result.add(
            new Video(x.getUuid("video_id"), x.getString("title"), x.getInstant("creation_date"))
        ));
        return result;
    }
}

```

- Clase **Application**

Esta es la clase principal donde inicia la ejecución (método **main**). En ella se realiza la invocación de todas las clases anteriores. La aplicación crea el keyspace, la tabla videos, inserta datos y finalmente muestra los videos existentes almacenados.

```

public class Application {
    private static final Logger LOG = LoggerFactory.getLogger(Application.class);
    private static final String KEYSPACE="video";
    private static final String TABLE_NAME="videos";
    public static void main(String[] args) {
        new Application().run();
    }

    public void run() {
        LOG.debug("Realizando conexión con el servidor Cassandra");
        CassandraConnector connector = new CassandraConnector();
        connector.connect("127.0.0.1", 9042, "datacenter1");
        CqlSession session = connector.getSession();

        LOG.debug("Creando keyspace {}",KEYSPACE);
        KeyspaceRepository keyspaceRepository = new KeyspaceRepository(session);
        keyspaceRepository.createKeyspace(KEYSPACE, 1);
        keyspaceRepository.useKeyspace(KEYSPACE);

        LOG.debug("Insertando un nuevo Video");
        VideoRepository videoRepository = new VideoRepository(session);
        videoRepository.createTable();
        videoRepository.insertVideo(new Video("Video 1", Instant.now()));
        videoRepository.insertVideo(new Video("Video 2",
            Instant.now().minus(1, ChronoUnit.DAYS)));

        LOG.debug("Obteniendo los videos existentes");
        List<Video> videos = videoRepository.selectAll();
        videos.forEach(x -> LOG.info(x.toString()));

        LOG.debug("Muy importante: cerrar la sesión con el servidor");
        connector.close();
    }
}

```

1.6.2. Actividades a realizar

Ejecutar el ejemplo anterior. Se deberá obtener una salida similar a la siguiente. **C2.**
Incluir su propia salida en el reporte.

```

> Task :Application.main()
16:52:50.987 DEBUG iimas.bdne.jrc.Application : Realizando conexión con el servidor
Cassandra
16:52:51.236 INFO com.datastax.oss.driver.internal.core.DefaultMavenCoordinates : DataStax Java driver for
Apache Cassandra(R) (com.datastax.oss:java-driver-core) version 4.1.0
16:52:51.533 INFO com.datastax.oss.driver.internal.core.time.Clock : Using native clock for microsecond
precision
16:52:51.871 DEBUG iimas.bdne.jrc.Application : Creando keyspace video
16:52:53.026 WARN com.datastax.oss.driver.internal.core.session.PoolManager : [s0] Detected a keyspace change
at runtime (<none> => video). This is an anti-pattern that should be avoided in production (see

```

```
'advanced.request.warn-if-set-keyspace' in the configuration).
16:52:53.030 DEBUG iimas.bdne.jrc.Application : Insertando un nuevo Video
16:52:54.223 DEBUG iimas.bdne.jrc.Application : Obteniendo los videos existentes
16:52:54.248 INFO iimas.bdne.jrc.Application : [id:5a99b25f-c0ae-4290-b892-bb721b625032,
title:Video 2, creationDate: 2024-03-18T22:52:54.220Z]
16:52:54.248 INFO iimas.bdne.jrc.Application : [id:08b1932e-28d6-4c08-8835-9f3c214b96d9,
title:Video 1, creationDate: 2024-03-19T22:52:54.170Z]
16:52:54.248 DEBUG iimas.bdne.jrc.Application : Muy importante: cerrar la sesión con el
servidor
```

Modificar la aplicación para realizar las siguientes acciones. **C3. Incluir en el reporte** el código generado y la salida del programa en la que se muestra la invocación del código nuevo.

- Agregar un método en **VideoRepository** que permita obtener un video con base a su id.
- Agregar un método en **VideoRepository** que permita eliminar un video a partir de su Id.
- Agregar código en **Application** para invocar estas 2 operaciones

Siguiendo la estructura del keyspace video y su tabla, agregar las clases necesarias para crear un nuevo keyspace hotel y así como una tabla **hoteles_por_lugar** vista anteriormente y agregar un registro. **C4. Incluir en el reporte** únicamente en el reporte la salida de la ejecución del programa en la que se pueda apreciar la creación del keyspace, tabla, inserción y consulta de los registros de la tabla.

1.7. Configuración de un cluster en Cassandra

A nivel general el proceso de configuración es similar al de otras bases de datos NoSQL. En esta ocasión se creará un cluster de 4 nodos, cada uno de ellos se representa a través de un contenedor Docker.

En ejercicios prácticos anteriores los contenedores fueron creados y administrados de forma manual. En esta ocasión se hará uso de Docker compose.

Docker Compose es una herramienta que permite crear un archivo con ciertas instrucciones, empleado para automatizar la administración de contenedores, por ejemplo, creación, configuración y eliminación.

- A. crear y actualizar un archivo llamado **docker-compose.yml**. Modificar las iniciales (**jrc-**) con el valor correspondiente. guardar el archivo en **/unam-bd-ne/ejercicios-practicos/ejercicio-practico-04/**
- B. Agregar un cuarto nodo.

```
networks:  
  cassandra:  
services:  
  jrc-cassandra-n1:  
    image: cassandra:latest  
    container_name: jrc-cassandra-n1  
    hostname: jrc-cassandra-n1  
    networks:  
      - cassandra  
    ports:  
      - "9042:9042"  
    environment: &environment  
      CASSANDRA_SEEDS: "jrc-cassandra-n1,jrc-cassandra-n2"  
      CASSANDRA_CLUSTER_NAME: MyTestCluster  
      CASSANDRA_DC: DC1  
      CASSANDRA_RACK: RACK1  
      CASSANDRA_ENDPOINT_SNITCH: GossipingPropertyFileSnitch  
      CASSANDRA_NUM_TOKENS: 128  
  jrc-cassandra-n2:  
    image: cassandra:latest  
    container_name: jrc-cassandra-n2  
    hostname: jrc-cassandra-n2  
    networks:  
      - cassandra  
    ports:  
      - "9043:9042"  
    environment: *environment  
    depends_on:  
      jrc-cassandra-n1:  
        condition: service_started  
  jrc-cassandra-n3:  
    image: cassandra:latest  
    container_name: jrc-cassandra-n3  
    hostname: jrc-cassandra-n3  
    networks:  
      - cassandra  
    ports:  
      - "9044:9042"  
    environment: *environment  
    depends_on:  
      jrc-cassandra-n2:  
        condition: service_started
```

C. Revisar en clase el significado de las instrucciones anteriores

- D. Para crear los contenedores a partir del archivo anterior, abrir una terminal y cambiarse al directorio donde se encuentra el archivo `docker-compose.yml`. Ejecutar:

```
cd /unam-bd-ne/ejercicios-practicos/ejercicio-practico-04/  
sudo docker compose up -d
```

- E. Acceder a alguno de los contenedores y revisar el archivo de configuración

```
sudo docker exec -it jrc-cassandra-n1 bash  
more /opt/cassandra/conf/cassandra.yaml
```

- F. Para acceder a `cqlsh` en alguno de los nodos del cluster:

```
sudo docker exec -it jrc-cassandra-n1 cqlsh
```

- G. Algunos comandos para verificar el status del cluster

```
#a Nivel s.o.  
/opt/cassandra/bin/nodetool status  
  
#en cqlsh  
cqlsh server-internal-ip-address 9042  
describe cluster  
SELECT cluster_name, listen_address FROM system.local;
```

- H. Ejecutar las instrucciones anteriores para validar el statu y las características del cluster. **C5. Incluir la salida** en el reporte

- I. Seleccionar una tabla cualquiera, crear un Keyspace con una replicación de 3, conectarse a cada uno de los nodos y verificar que el dato aparezca en los 3 nodos. **C6. Incluir en el reporte** el código y salida de ejecución de las consultas en cada nodo.