Gestión/Almacenamiento De Información No Estructurada: PEC 3 BASES DE DATOS COLUMNARES: CASSANDRA

Sergio Soler Rocha
Universidad Nacional de Educación a Distancia

1. Ejercicio 1

DataStax Astra DB es un servicio de base de datos en la nube que ofrece una implementación completamente gestionada de Apache Cassandra, un sistema de gestión de bases de datos NoSQL distribuido conocido por su escalabilidad y rendimiento robusto. Astra DB elimina la complejidad asociada con la configuración, gestión y operación de clusters Cassandra, proporcionando una solución sin servidor que permite a los desarrolladores concentrarse en la creación de aplicaciones sin preocuparse por el mantenimiento de la infraestructura subyacente.

Descripción de la Base de Datos y Keyspaces

Base de Datos: La base de datos creada en DataStax Astra DB se denomina `Pec3`. Su propósito principal es gestionar y almacenar información relacionada con viajes turísticos. Esta base de datos almacena datos sobre clientes, destinos, fechas de viaje, y otros detalles relevantes que permiten a la empresa turística ofrecer servicios personalizados y gestionar incidencias de manera eficiente.

Keyspaces: En Apache Cassandra, un keyspace es el equivalente a una base de datos en sistemas de gestión de bases de datos relacionales. Sirve como contenedor de alto nivel para la configuración de replicación y las tablas. En nuestro caso, hemos creado un keyspace llamado `agenciaviajes` dentro de la base de datos `Pec3`. Este keyspace está diseñado específicamente para almacenar y organizar datos relacionados con los viajes de los clientes. Contiene tablas que registran los viajes por cliente, los detalles de los viajes, y los hitos asociados a cada viaje.

Configuración de Replicación

Estrategia de Replicación: Para el keyspace `agenciaviajes`, se ha configurado la estrategia de replicación `SimpleStrategy` con un `replication_factor` de 3. Esta configuración se ha elegido por varias razones:

1. **Simplicidad**: `SimpleStrategy` es adecuada para entornos de desarrollo o para un cluster que opera en una sola región de datos. Es fácil de configurar y administrar.
2. **Alta Disponibilidad**: Un `replication_factor` de 3 significa que cada pieza de datos se replica tres veces en diferentes nodos del cluster. Esto asegura una alta disponibilidad de los datos, ya que la base de datos puede tolerar la falla de hasta dos nodos antes de que los datos se vuelvan inaccesibles.
3. **Resiliencia de Datos**: La replicación múltiple también protege contra la pérdida de datos. En caso de falla de un nodo, Cassandra puede recuperar los datos desde otros nodos donde los datos están replicados.

Uso Actual del Período de Facturación

- **Solicitudes de Lectura**: 2

- **Solicitudes de Escritura**: 7
- **Consumo de Almacenamiento**: 25.83 KB
- **Transferencia de Datos**: 9.26 MB

Estos datos reflejan un uso relativamente moderado de recursos, lo que es típico para un entorno de desarrollo o prueba inicial. Estas métricas ayudan a monitorizar el rendimiento y el costo, asegurando que la base de datos se gestione de manera eficiente en términos de costos y rendimiento.

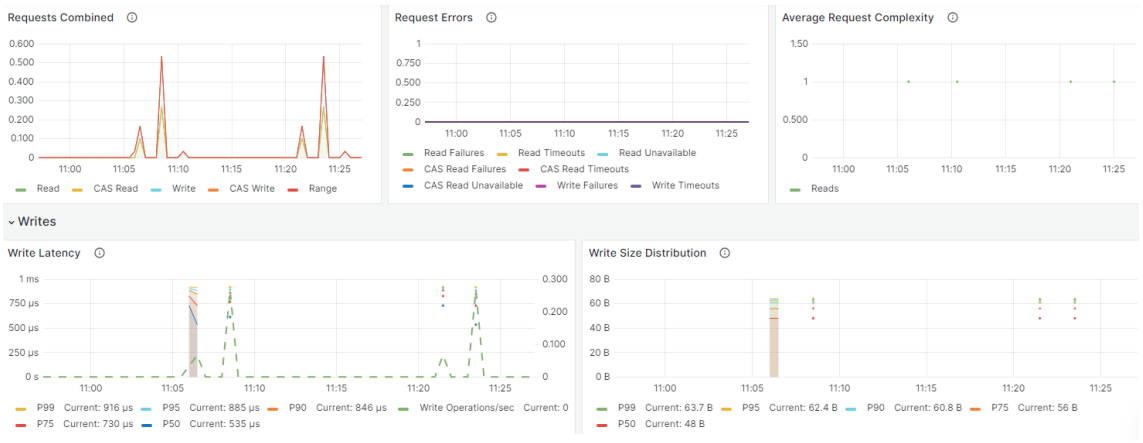


Figura 1: Dashboard 1

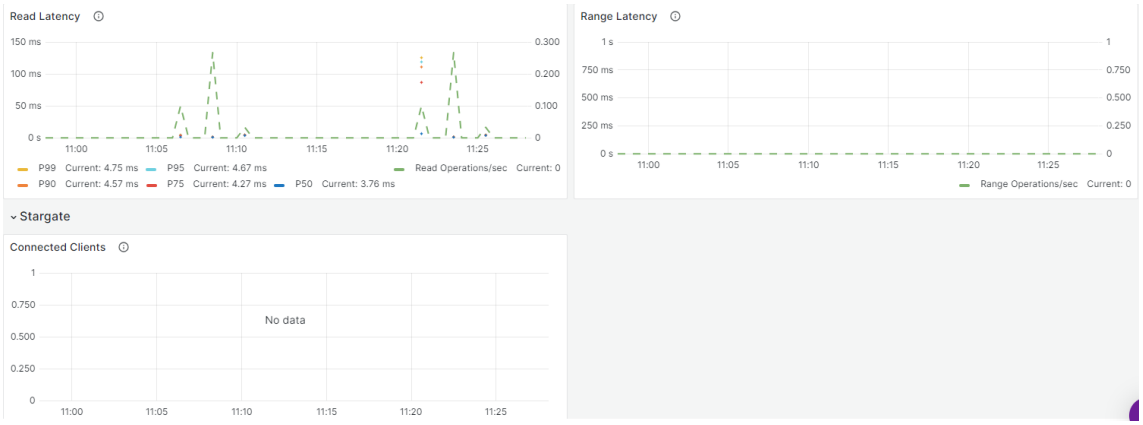


Figura 2: Dashboard 2

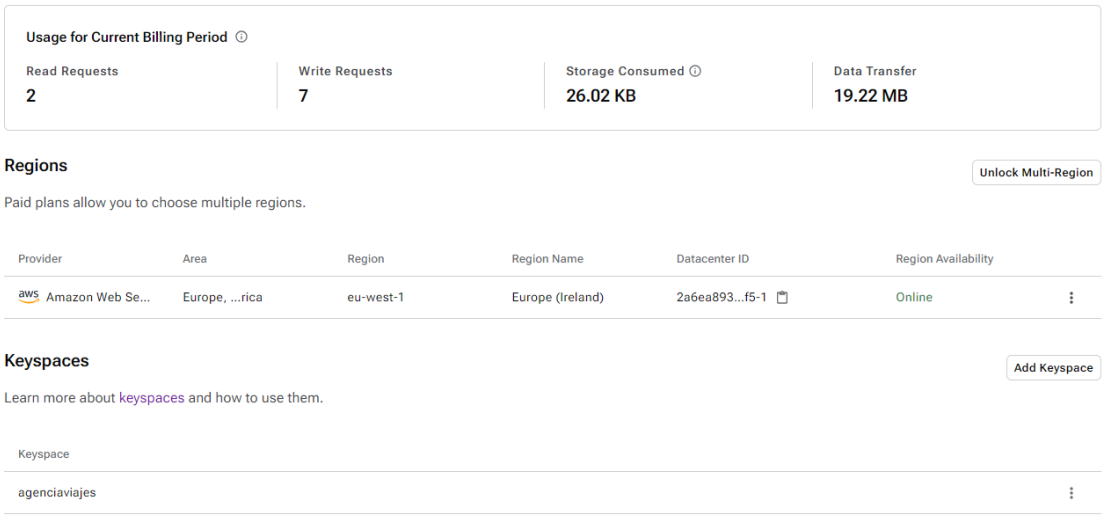


Figura 3: Información de la base de datos

2. Ejercicio 2

2.1. Apartado 2.1

El modelo de datos está diseñado para optimizar las consultas más frecuentes y asegurar una distribución eficiente de los datos entre los nodos del cluster Cassandra. Se utilizará la estrategia de replicación `SimpleStrategy` con un `replication_factor` de 3 para proporcionar alta disponibilidad y tolerancia a fallos dentro de un único centro de datos.

Definición de las Tablas

Dentro del keyspace `agenciaviajes`, se proponen las siguientes tablas:

- **viajes_por_cliente**: Para almacenar información sobre los viajes asociados a cada cliente.
- **clientes_por_viaje**: Para registrar qué clientes están asociados a cada viaje específico y sus hitos relevantes.

Código para la Creación de Tablas

Creación del Keyspace

```
CREATE KEYSPACE agenciaviajes WITH replication = {
 'class': 'SimpleStrategy',
 'replication_factor' : 3
};
```

Tabla viajes_por_cliente

```
CREATE TABLE agenciaviajes.viajes_por_cliente (
idCliente text,
fechaPartida date,
idViaje int,
lugarPartida text,
lugarDestino text,
numDias int,
PRIMARY KEY (idCliente, fechaPartida)
) WITH CLUSTERING ORDER BY (fechaPartida DESC);
```

Tabla clientes_por_viaje

```
CREATE TABLE agenciaviajes.clientes_por_viaje (
idViaje int,
idCliente text,
hitos map<text, text>,
PRIMARY KEY (idViaje, idCliente)
) WITH CLUSTERING ORDER BY (idCliente ASC);
```

2.2. Apartado 2.2

Inserciones en la Tabla viajes_por_cliente

```
-- Insertar datos para Bertoldo, viaje de Madrid a París
INSERT INTO agenciaviajes.viajes_por_cliente (idCliente, fechaPartida, idViaje, lugarPartida,
lugarDestino, numDias)
VALUES ('Bertoldo', '2020-05-02', 3, 'Madrid', 'París', 2);

-- Insertar datos para Herminia, viaje de Barcelona a París
INSERT INTO agenciaviajes.viajes_por_cliente (idCliente, fechaPartida, idViaje, lugarPartida,
lugarDestino, numDias)
VALUES ('Herminia', '2020-05-02', 3, 'Barcelona', 'París', 2);

-- Insertar datos para Bertoldo, viaje de Madrid a Tokyo
INSERT INTO agenciaviajes.viajes_por_cliente (idCliente, fechaPartida, idViaje, lugarPartida,
lugarDestino, numDias)
VALUES ('Bertoldo', '2020-06-01', 4, 'Madrid', 'Tokyo', 7);
```

Inserciones en la Tabla clientes_por_viaje

```
-- Insertar hitos para Bertoldo en el viaje a París
INSERT INTO agenciaviajes.clientes_por_viaje (idViaje, idCliente, hitos)
VALUES (3, 'Bertoldo', {'Versailles': '3-05 contacta agencia'});

-- Insertar hitos para Herminia en el viaje a París
INSERT INTO agenciaviajes.clientes_por_viaje (idViaje, idCliente, hitos)
VALUES (3, 'Herminia', {'Versailles': '3-05 contacta agencia'});

-- Insertar hitos para Bertoldo en el viaje a Tokyo
INSERT INTO agenciaviajes.clientes_por_viaje (idViaje, idCliente, hitos)
VALUES (4, 'Bertoldo', {'Transbordo': 'Frankfurt, puerta C27'});
```

2.3. Apartado 2.3

Consulta 1: Viajes de Bertoldo

Para recuperar información sobre todos los viajes realizados por Bertoldo, ordenados por la fecha de partida en orden descendente, utilizamos la siguiente consulta:

```
SELECT lugarPartida, lugarDestino, fechaPartida, numDias
FROM agenciaviajes.viajes_por_cliente
WHERE idCliente = 'Bertoldo'
ORDER BY fechaPartida DESC;
```

Resultado de la Consulta 1

A continuación se muestra la salida de la consulta para los viajes de Bertoldo:

lugarpartida	lugardestino	fechapartida	numdias
Madrid	Tokyo	2020-06-01	7
Madrid	París	2020-05-02	2

(2 rows)

Figura 4: Resultados de los viajes de Bertoldo

Consulta 2: Hitos del viaje con identificador 3

Para obtener información sobre los clientes asociados al viaje con identificador 3, incluyendo los hitos registrados, utilizamos la siguiente consulta:

```
SELECT idCliente, hitos
FROM agenciaviajes.clientes_por_viaje
WHERE idViaje = 3
ORDER BY idCliente ASC;
```

Resultado de la Consulta 2

A continuación se muestra la salida de la consulta para los hitos del viaje con identificador 3:

idcliente	hitos
Bertoldo	{'Versailles': '3-05 contacta agencia', 'regreso': '2020-05-04'}
Herminia	{'Versailles': '3-05 contacta agencia'}

Figura 5: Hitos del viaje con identificador 3

2.4. Apartado 2.4

Descripción de la Actualización

Se añadirá un nuevo hito al registro del viaje de Bertoldo con el identificador de viaje 3. Este hito, denominado ‘regreso‘, registrará el evento ocurrido el 4 de mayo de 2020.

Código para la Actualización

Para añadir este nuevo hito al viaje, utilizaremos la siguiente instrucción CQL que actualiza el mapa de hitos en la tabla `clientes_por_viaje`:

```
UPDATE agenciaviajes.clientes_por_viaje
SET hitos['regreso'] = '2020-05-04'
WHERE idViaje = 3 AND idCliente = 'Bertoldo';
```

Explicación del Código

La instrucción `UPDATE` se utiliza para modificar una fila existente en la tabla `clientes_por_viaje`. La columna `hitos`, que es un tipo de dato mapa en Cassandra, se actualiza añadiendo un nuevo par clave-valor sin afectar otros pares existentes en el mapa. La clave es ‘regreso‘ y el valor es ‘2020-05-04‘. El `WHERE` especifica que la actualización debe aplicarse solo al registro donde `idViaje` es 3 y `idCliente` es 'Bertoldo', garantizando que no se afecten otros registros accidentalmente.

2.5. Apartado 2.5

Solución sin Modificar el Modelo de Datos Existente

Descripción y Código

Utilizando el modelo de datos actual, la consulta requeriría el uso de `ALLOW FILTERING` para filtrar por fecha de partida, lo cual es ineficiente:

```
SELECT * FROM agenciaviajes.viajes_por_cliente
WHERE fechaPartida = '2020-05-02' ALLOW FILTERING;
```

Ventajas e Inconvenientes

Ventajas: - No requiere cambios en la estructura de la base de datos.
Inconvenientes: - Alto costo en términos de rendimiento debido al escaneo completo de la tabla. - No recomendado para producción, especialmente en datasets grandes.

Solución con Modificación del Modelo de Datos

Propuesta de Nuevo Modelo

Crear una nueva tabla que particione los datos por fecha de partida:

```
CREATE TABLE agenciaviajes.viajes_por_fecha (
fechaPartida date,
idCliente text,
idViaje int,
lugarPartida text,
lugarDestino text,
numDias int,
PRIMARY KEY (fechaPartida, idCliente)
);
```

Código para la Consulta

```
SELECT idCliente, idViaje, lugarPartida, lugarDestino, numDias
FROM agenciaviajes.viajes_por_fecha
WHERE fechaPartida = '2020-05-02';
```

Ventajas e Inconvenientes

Ventajas: - Consultas altamente eficientes por fecha de partida. - Mejor escalabilidad y gestión del rendimiento.
Inconvenientes: - Requiere modificaciones en el modelo de datos y posiblemente duplicación de datos. - Necesidad de mantener la sincronización entre múltiples tablas al actualizar datos.

3. Ejercicio 3

En este documento se propone un modelo de datos diseñado específicamente para un sistema de gestión de videos y comentarios utilizando Apache Cassandra. La estructura de este modelo se basa en optimizar las consultas más frecuentes del sistema, asegurando eficiencia y escalabilidad.

Modelo de Datos Propuesto

El modelo de datos incluye cuatro tablas principales: `usuarios`, `videos`, `comentarios_por_usuario` y `comentarios_por_video`. A continuación, se describen las tablas y se justifican las decisiones tomadas en su diseño.

3.1. Tabla de Usuarios

```
CREATE TABLE usuarios (
usuario_id UUID PRIMARY KEY,
nombre TEXT,
email TEXT
);
```

Esta tabla almacena la información básica de los usuarios. Se utiliza `usuario_id` como clave primaria para garantizar la unicidad y permitir consultas eficientes por ID de usuario.

3.2. Tabla de Videos

```
CREATE TABLE videos (
video_id UUID PRIMARY KEY,
titulo TEXT,
descripcion TEXT
);
```

Similar a la tabla de usuarios, cada video es identificado unívocamente por un `video_id`. Esto facilita la recuperación de detalles del video directamente mediante su ID.

3.3. Tabla de Comentarios por Usuario

```
CREATE TABLE comentarios_por_usuario (
usuario_id UUID,
timestamp TIMESTAMP,
video_id UUID,
comentario TEXT,
PRIMARY KEY (usuario_id, timestamp)
) WITH CLUSTERING ORDER BY (timestamp DESC);
```

Esta tabla indexa los comentarios por usuario, facilitando la recuperación de todos los comentarios realizados por un usuario específico, ordenados del más reciente al más antiguo. Se utiliza `usuario_id` como parte de la clave de partición y `timestamp` como clave de agrupamiento para ordenarlos cronológicamente.

3.4. Tabla de Comentarios por Video

```
CREATE TABLE comentarios_por_video (
video_id UUID,
timestamp TIMESTAMP,
usuario_id UUID,
comentario TEXT,
PRIMARY KEY (video_id, timestamp)
) WITH CLUSTERING ORDER BY (timestamp DESC);
```

De manera análoga, esta tabla facilita la recuperación de todos los comentarios de un video, ordenados también del más reciente al más antiguo. La eficiencia se logra mediante el uso de `video_id` como clave de partición.

3.5. Creación del Keyspace

El keyspace denominado `gestion_videos` se creó con una estrategia de replicación simple, ideal para entornos de desarrollo y pruebas, pero que debe ser ajustada para entornos de producción donde se requieren configuraciones de replicación más robustas. El comando utilizado para crear este keyspace es el siguiente:

```
CREATE KEYSPACE gestion_videos WITH replication = {
 'class': 'SimpleStrategy',
 'replication_factor' : 3
};
```

Población de Datos

El objetivo de esta sección es describir el proceso de inserción de datos en la base de datos de gestión de videos, enfocándose en cómo se maneja la inserción de comentarios para maximizar la eficiencia de las consultas futuras, lo que conlleva múltiples escrituras por una sola acción de inserción de comentario.

Usuarios y Videos

Inicialmente, se poblaron las tablas `usuarios` y `videos` con un mínimo de tres registros cada una. A continuación, se detalla la inserción:

- Usuarios: Se insertaron tres usuarios con identificadores únicos y datos básicos como nombre y correo electrónico.
- Videos: De manera similar, se insertaron tres registros de videos, cada uno con un título y una descripción.

Comentarios

Cada usuario realizó dos comentarios en los videos, lo que resultó en múltiples acciones de escritura en la base de datos. Por cada comentario realizado por un usuario, se realizaron las siguientes inserciones:

- **En la tabla** `comentarios_por_usuario`: Esta tabla captura cada comentario relacionado con el usuario que lo hizo, ordenado por el timestamp para recuperaciones eficientes basadas en el tiempo.
- **En la tabla** `comentarios_por_video`: Similarmente, cada comentario también se registra en relación con el video comentado, permitiendo una rápida recuperación de todos los comentarios asociados a un video específico.

Ejemplo de inserción: Un comentario de Üsuario 1"sobre "Video 1resulta en:

```
INSERT INTO comentarios_por_usuario (usuario_id, timestamp, video_id, comentario)
VALUES (UUID de Usuario 1, timestamp actual, UUID de Video 1, 'Excelente video!');
INSERT INTO comentarios_por_video (video_id, timestamp, usuario_id, comentario)
VALUES (UUID de Video 1, timestamp actual, UUID de Usuario 1, 'Excelente video!');
```

Consultas Ejecutadas

Consulta 1: Comentarios de un Usuario

La siguiente consulta SQL extrae todos los comentarios realizados por un usuario específico, ordenados del más reciente al más antiguo. Se utiliza la tabla `comentarios_por_usuario`.

```
SELECT * FROM gestion_videos.comentarios_por_usuario
WHERE usuario_id = 9fbfd90e-08dd-45e0-9a6f-68c551ab30fe;
```

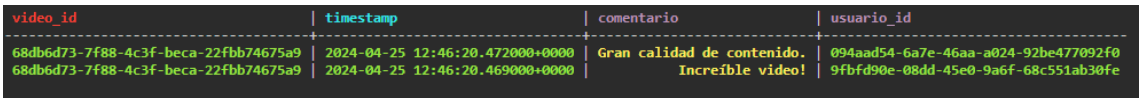
usuario_id	timestamp	comentario	video_id
9fbfd90e-08dd-45e0-9a6f-68c551ab30fe	2024-04-25 12:44:21.354000+0000	Muy bien explicado.	192c67b4-4168-4d81-a428-028c269224f8
9fbfd90e-08dd-45e0-9a6f-68c551ab30fe	2024-04-25 12:44:21.351000+0000	Increíble video!	68db6d73-7f88-4c3f-beca-22fbb74675a9
9fbfd90e-08dd-45e0-9a6f-68c551ab30fe	2024-04-25 12:36:18.347000+0000	Gran video!	68db6d73-7f88-4c3f-beca-22fbb74675a9

Figura 6: Captura de la consulta 1

Consulta 2: Comentarios de un Video

Esta consulta recupera todos los comentarios hechos en un video específico, igualmente ordenados del más reciente al más antiguo. Se utiliza la tabla `comentarios_por_video`.

```
SELECT * FROM gestion_videos.comentarios_por_video
WHERE video_id = 68db6d73-7f88-4c3f-beca-22fbb74675a9;
```



video_id	timestamp	comentario	usuario_id
68db6d73-7f88-4c3f-beca-22fbb74675a9	2024-04-25 12:46:20.472000+0000	Gran calidad de contenido.	094aad54-6a7e-46aa-a024-92be477092f0
68db6d73-7f88-4c3f-beca-22fbb74675a9	2024-04-25 12:46:20.469000+0000	Increíble video!	9fbfd90e-08dd-45e0-9a6f-68c551ab30fe

Figura 7: Captura de la consulta 2