Ejercicio de Aplicación #4 – Apache Cassandra

Caso de Uso: Registro de llamadas – Telecomunicaciones

Una empresa necesita almacenar información sobre llamadas: número de origen, destino, duración y fecha. El sistema debe estar optimizado para consultas por número y fecha, y debe ser altamente escalable.

1. Diseño de la estructura de la base de datos

```
CREATE TABLE llamadas_por_cliente (
   numero_origen TEXT,
   fecha DATE,
   id_llamada UUID,
   numero_destino TEXT,
   duracion_min INT,
   PRIMARY KEY ((numero_origen), fecha, id_llamada)
);
```

- numero_origen es clave de partición → permite distribuir la carga por cliente.
- fecha y id_llamada son claves de ordenamiento → permiten búsquedas por rango temporal.

2. Inserción de datos de ejemplo

```
INSERT INTO llamadas_por_cliente (numero_origen, fecha, id_llamada, numero_destino,
duracion_min)
VALUES ('+56911111111', '2025-07-26', uuid(), '+56922222222', 15);

INSERT INTO llamadas_por_cliente (numero_origen, fecha, id_llamada, numero_destino,
duracion_min)
VALUES ('+56911111111', '2025-07-26', uuid(), '+56933333333', 8);

INSERT INTO llamadas_por_cliente (numero_origen, fecha, id_llamada, numero_destino,
duracion_min)
VALUES ('+56922222222', '2025-07-25', uuid(), '+56911111111', 12);
```

3. Consultas implementadas

a) Todas las llamadas por número específico:

```
SELECT * FROM llamadas_por_cliente
WHERE numero_origen = '+56911111111';
```

b) Llamadas en un rango de fechas:

```
SELECT * FROM llamadas_por_cliente
WHERE numero_origen = '+56911111111'
  AND fecha >= '2025-07-01' AND fecha <= '2025-07-31';
```

c) Total de minutos por cliente (agregado del lado del cliente):

```
SELECT fecha, duracion_min FROM llamadas_por_cliente
WHERE numero_origen = '+56911111111'
  AND fecha >= '2025-07-01' AND fecha <= '2025-07-31';
```

4. Comparación con base de datos relacional

Cassandra vs Relacional:
- Modelo: desnormalizado vs normalizado.
- Escalabilidad: horizontal vs vertical.
- Escritura: muy rápida vs moderada.
- Agregaciones: manuales en Cassandra vs nativas en SQL.
- Ideal para: alto volumen y rendimiento vs lógica compleja.

Plus: Tabla de usuarios

```
CREATE TABLE usuarios (
   user_id UUID PRIMARY KEY,
   nombre TEXT,
   email TEXT,
   fecha_registro DATE
);
```

```
INSERT INTO usuarios (user_id, nombre, email, fecha_registro)
VALUES (uuid(), 'Juan Pérez', 'juan@mail.com', '2025-01-15');

INSERT INTO usuarios (user_id, nombre, email, fecha_registro)
VALUES (uuid(), 'Ana Torres', 'ana@mail.com', '2025-02-05');

INSERT INTO usuarios (user_id, nombre, email, fecha_registro)
VALUES (uuid(), 'Luis Soto', 'luis@mail.com', '2025-03-10');
```

Consulta:

```
SELECT * FROM usuarios;
```