Introducción a Cassandra con Java

About me


seresco

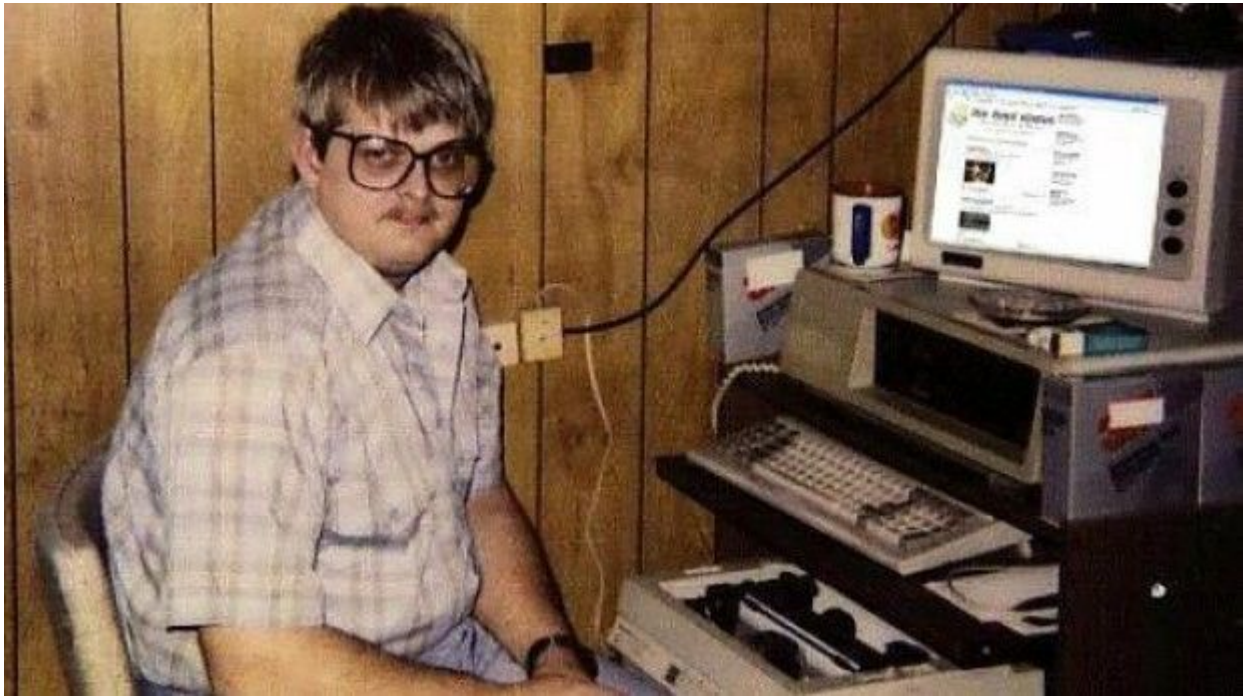

Optare
Solutions


krasis


INFORMÁTICA
El Corte Inglés


HAC LUCE
UNIVERSIDADE DA CORUÑA

¿De dónde venimos?



¿De dónde venimos?

- **Bases de datos relacionales**
- **ACID:**
  - **Atomicidad**
  - **Consistencia**
  - **Isolation (Aislamiento)**
  - **Durabilidad**
- **Tablas (Columnas y filas)**
- **Modelo entidad - relación**

¿Quienes somos?

¿Quienes somos?

- **Más y más datos**
- **Distintas formas de procesar la información**
- **ACID ya no es nuestro amigo**
- **Nuevos actores en juego**
  - **Google**
  - **Facebook**
  - **Twitter**

NoSQL (Not Only SQL)

- **Comienzos del 2000**
- **Distintos tipos**
  - **Documentos (MongoDB, CouchBase, CouchDB,etc..)**
  - **Clave-valor (Redis, DynamoDB, Voldemort)**
  - **Grafos (Neo4j, Titan, Allegro)**
  - **Wide-column (Cassandra, HBase, Google BigTable)**
- **Escalables horizontalmente**
- **Mejoras de rendimiento**

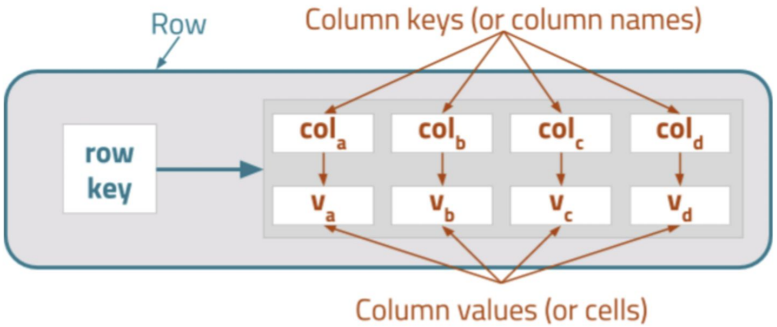
Cassandra

- **Inicialmente desarrollada por Facebook**
- **Su historia:**
  - **Open Source en 2008**
  - **Apache Incubator en 2009**
  - **Ciudadano de primer nivel de Apache en 2010**

Versión	Fecha liberación original	Última version	Fecha liberación	Estado[21]
0.6	2010-04-12	0.6.13	2011-04-18	Ya no tiene soporte
0.7	2011-01-10	0.7.10	2011-10-31	Ya no tiene soporte
0.8	2011-06-03	0.8.10	2012-02-13	Ya no tiene soporte
1.0	2011-10-18	1.0.12	2012-10-04	Ya no tiene soporte
1.1	2012-04-24	1.1.12	2013-05-27	Ya no tiene soporte
1.2	2013-01-02	1.2.19	2014-09-18	Ya no tiene soporte
2.0	2013-09-03	2.0.17	2015-09-21	Ya no tiene soporte
2.1	2014-09-16	2.1.13	2016-02-08	Tiene soporte
2.2	2015-07-20	2.2.5	2016-02-08	Tiene soporte
3.0	2015-11-09	3.0.5	2016-04-11	Tiene soporte
3.2	2015-11-09	3.2.1	2016-01-18	Tiene soporte
3.7	2016-06-13	3.7	2016-06-13	Última liberación
Leyenda: Versión antígua · Versión antígua, soportada · **Última versión** · Última versión prevista				

Cassandra II

- Lenguaje propio de consulta CQL
- Comunicación Peer -to-Peer entre nodos
  - Elimina el single point of failure
  - No usa esquema master - slave
- Escalabilidad lineal
- Escala horizontalmente



Empezamos! Show me code!

- **¿Como nos conectamos?**

```
return Cluster.builder()
                .addContactPoint("server-1")
                .build();
```

- **Se puede establecer la consistencia de los resultados a nivel cluster:**

```
return Cluster.builder()
    .addContactPoint("server-1")
    .withQueryOptions(new QueryOptions()
        .setConsistencyLevel(ConsistencyLevel.ONE))
    .withRetryPolicy(DefaultRetryPolicy.INSTANCE).build();
```

- **Consejo #1: Una única instancia por cluster para toda la aplicación.**
- **Consejo #2: Usa múltiples “contact points”**

Creando la sesión

- **La sesión es nuestro punto de contacto con Cassandra**

```
Session session = cluster.connect();
```

- **Consejo #3: Usa una sesión por keyspace o si usas una única sesión especifica el keyspace para cada query.**

```
String query = "SELECT * FROM vigojug.sensors";
```

Creando consultas

- **Se crean a partir de la sesión**

```
Session session = Connection.getInstance().getSession();
session.execute("SELECT * FROM vigojug.sensors");
```

- **Consejo #4: ¿Quieres ejecutar más de una vez la misma consulta? Usa PreparedStatements.**

```
String preparedQuery = "SELECT * FROM vigojug.sensors where sensorId = ?;";
 PreparedStatement preparedStatement = session.prepare(prepareQuery);
 BoundStatement boundStatement = preparedStatement.bind(
                UUID.fromString("48b71ba2-11c0-431a-8c4a-bc8aed95635b"));
 session.execute(boundStatement);
```

Creando consultas (II)

- **Consejo #5: Puedes modificar el nivel de consistencia para cada consulta**

```
String preparedQuery = "SELECT * FROM vigojug.sensors where sensorId = ?;";
PreparedStatement preparedStatement = session.prepare(prepareQuery);
BoundStatement boundStatement = preparedStatement.bind(
        UUID.fromString("48b71ba2-11c0-431a-8c4a-bc8aed95635b"));
boundStatement.setConsistencyLevel(ConsistencyLevel.ONE);
session.execute(boundStatement);
```

Obteniendo resultados

- **Tras la consulta se obtiene un ResultSet**

```
ResultSet resultSet = session.execute(boundStatement);
for(Row row : resultSet ) {
   UUID sensorId = row.getUUID("sensorId");
   Date creationDate = row.getTimestamp("creationDate");
}
```

- **Consejo #6: Se puede establecer la cantidad de registros recuperados.**
  - Por sentencia: `boundStatement.setFetchSize(100);`
  - Por conexión al cluster:

```
.withQueryOptions(new QueryOptions().setFetchSize(100))
```

Consultas asíncronas

- **Se pueden realizar consultas asíncronas**

```
ResultSetFuture future = session.executeAsync("SELECT release_version FROM system.local");

while (!future.isDone()) {
    logger.debug("Waiting for request to complete");
}
ResultSet rs = future.get();
logger.debug("Got response: {}", rs.one().getString("release_version"));
```

- **Consejo #7: Enlazando con la charla de Saul..usad RxJava la respuesta.**

Consultas asíncronas

- **Consejo #7: Enlazando con la charla de Saul..usad RxJava la respuesta.**

```
public static Observable<ResultSet> queryAllAsObservable(Session session, String query, Object...
partitionKeys) {
    List<ResultSetFuture> futures = sendQueries(session, query, partitionKeys);
    Scheduler scheduler = Schedulers.io();
    List<Observable<ResultSet>> observables = Lists.transform(futures, (ResultSetFuture future) ->
Observable.from(future, scheduler));
    return Observable.merge(observables);
}
```

Q&A

Gracias a todos!!