INGENIERÍA DE DATOS 2

Documento Tecnico

Grupo 10

Docente

- **Joaqin Salas**

Integrantes

- **Daniela Gangotena – 1185437**
- **Alejandro Valente - 1196776**

---

23 de Febrero 2026

---

1. Análisis CAP por componente

- ## MongoDB (CP)
    - Por defecto, MongoDB es un sistema **CP**.
    - **Consistencia:** Prioriza que los datos sean exactos. Si el nodo "Primario" se cae, el sistema deja de aceptar escrituras hasta que se elige uno nuevo para asegurar que nadie lea datos viejos.
    - Por eso guardamos ahí el registro de calificaciones; necesitamos que el ID y los datos básicos sean consistentes y no se dupliquen.
- ## Neo4j (CA / CP)
    - Neo4j es una base de datos **ACID** nativa.
    - En una instancia única es **CA** (Consistencia y Disponibilidad), pero no escala horizontalmente de forma nativa como las otras.
    - En un cluster, se comporta más como **CP**. Prioriza la integridad del grafo (las relaciones) por sobre la disponibilidad inmediata de todos los nodos si hay un corte de red.
    - **Uso en tu TPO:** Usamos para las relaciones complejas donde la integridad del vínculo:
        - (Estudiante)-[:CURSO]->(Materia) es crítica.
        - (Estudiante)-[:ESTUDIO_EN]->(Materia) es crítica.
- ## Cassandra (AP)
    - Cassandra es el ejemplo perfecto de un sistema **AP**.
    - **Disponibilidad:** Está diseñada para que nunca dejemos de poder escribir o leer, aunque un nodo en otra parte del mundo tenga una versión vieja del dato.
    - **Consistencia Eventual:** Eventualmente todos los nodos se sincronizan, pero prioriza que el reporte siempre se entregue rápido.
    - **Uso en tu TPO:** Lo usamos para los rankings. No importa si el promedio de un alumno tarda 2 segundos en actualizarse en todos los nodos, lo importante es que el sistema de reportes nunca se caiga y sea veloz.

Componente	**Clasificación**	**Justificación**
MongoDB	**CP**	Garantiza que no haya inconsistencias al registrar alumnos.
Neo4j	**CP / CA**	Asegura que las relaciones del grafo cumplan con integridad referencial.

Cassandra	**AP**	Permite consultas de rankings masivas con alta disponibilidad y baja latencia.

2. Justificación detallada de cada base NoSQL (funcionalidad + etapa)

- **MongoDB (Base de Documentos)**
  - **Etapa:** Registro y Almacenamiento de Datos Maestros.
  - **Funcionalidad:** Gestión de perfiles de Estudiantes y versionado de las legislaciones de conversión de calificaciones
  - Justificación Técnica:
    - **Esquema Flexible (Schemaless):** Los estudiantes de distintas instituciones pueden tener metadatos muy diferentes (ej. alumnos alemanes con campos de "Gymnasium" vs. locales). MongoDB permite guardar estos BSON heterogéneos sin romper la estructura. Se pueden guardar todas las versiones de las legislaciones sin que exista conflicto si se integran con nuevos países con escalas de notas nuevas.
    - **Escalabilidad Horizontal:** Al ser el punto de entrada de todos los registros del sistema, MongoDB permite manejar grandes volúmenes de datos mediante *sharding* si la cantidad de estudiantes crece exponencialmente.
    - **Análisis CAP:** Se clasifica como CP. Garantiza que, al registrar un alumno, los datos sean consistentes en todo el sistema, evitando duplicidad de IDs.

- **Neo4j (Base de Grafos)**
  - **Etapa:** Relaciones Complejas y Trazabilidad Académica.
  - **Funcionalidad:** Mapeo del historial educativo (Estudiante ->Institución) y progreso académico (Estudiante ->Materia) y Currícula de la institución (Instituciòn->Materia).
  - Justificación Técnica:
    - **Performance en Joins:** A diferencia de SQL o Mongo, donde unir "Estudiante-Materia-Nota-Institución" requiere cruces costosos, Neo4j utiliza *index-free adjacency*. Recorrer el camino de un alumno por 10 colegios toma el mismo tiempo que recorrer uno solo.
    - **Relaciones con Propiedades:** Permite que el "vínculo" mismo guarde datos (como el promedio o el periodo).

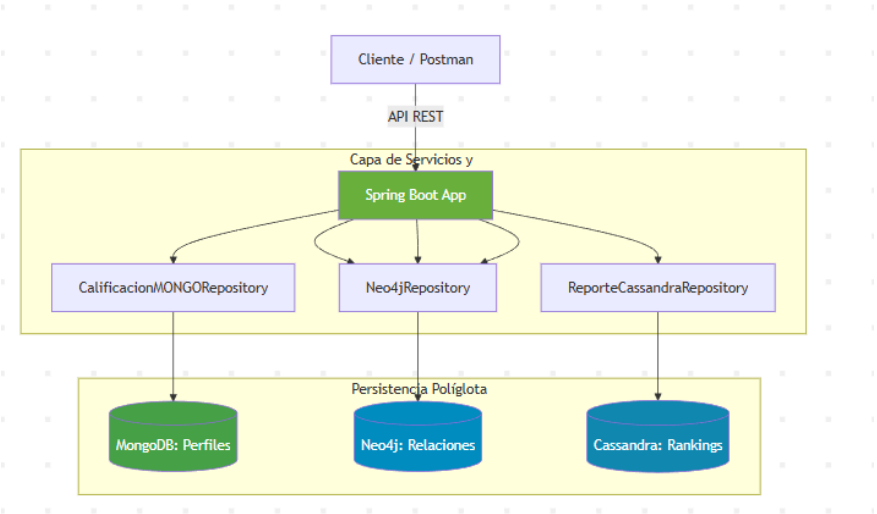
        - **Análisis CAP:** Se comporta como CA/CP. Prioriza la integridad referencial. Es vital que si existe una relación, esta sea veraz y no existan "nodos huérfanos".

- **Cassandra (Base de Columnas Anchas)**
    - Etapa: Analítica de Alto Rendimiento y Reportes.
    - **Funcionalidad:** Generación de Rankings (Mejor alumno, Mejor país, Mejor instituto).
    - Justificación Técnica:
        - **Escritura Ultrarrápida:** Cassandra está diseñada para absorber ráfagas masivas de datos procesados.
        - **Query-Driven Design:** Las tablas se diseñan específicamente para responder una pregunta. Al usar *Clustering Columns* **(ORDER BY promedio DESC), l**a base de datos devuelve el ranking ya ordenado físicamente desde el disco, eliminando la necesidad de procesos de ordenamiento **(SORT)** en memoria que ralentizarían el sistema.
        - **Análisis CAP:** Es un sistema AP. Prioriza que el reporte esté siempre disponible para los directivos, aceptando "consistencia eventual" (el ranking puede tardar unos segundos en reflejar la última nota cargada en Mongo, pero nunca dejará de responder).

Base de Datos	Rol Principal	Fortaleza Clave
MongoDB	Transaccional	Flexibilidad de atributos del estudiante.
Neo4j	Relacional	Rapidez para navegar el historial académico.
Cassandra	Analítico	Velocidad de lectura para ránkings masivos.

3. Diagramas (arquitectura, ER, flujos de datos)

Diagrama de arquitectura:



4.Esquemas de datos en cada base

Arquitectura del Sistema

El sistema utiliza una **Arquitectura Políglota**. Los datos fluyen desde la carga operativa hacia el análisis masivo.

Flujos Críticos

1. **Alta de Calificación:** El dato entra por **MongoDB** (Persistencia de Documento) Se replica la relación en **Neo4j** (Actualización de Grafo).
2. **Conversión:** Proceso Batch/Service que extrae promedios de Neo4j y los inserta en **Cassandra**.
3. **Consulta Analítica:** El usuario consulta el ranking directamente desde **Cassandra** sin tocar las bases operativas.

5. Trade-offs y decisiones de diseño

Base de Datos	**Esquema / Modelo**	**Trade-off (Decisión)**
MongoDB	Colección Estudiantes (BSON)	**Flexibilidad vs Integridad:** Se prioriza poder guardar cualquier metadato académico aunque no sea uniforme.

Neo4j	Nodos (:Estudiante), (:Materia)	**Relaciones vs Almacenamiento:** Se duplican nombres para evitar "joins" virtuales, priorizando velocidad de navegación.
Cassandra	Tabla reportes_ranking	**Escritura vs Lectura:** Se pre-calculan los datos al insertar para que la lectura sea $O(1)$.

6.Resultados de performance con 1M registros (tiempos de inserción/consulta)

7.Análisis del dominio

- **Entidades**
  - MATERIA
  - ESTUDIANTE
  - CALIFICACION
  - CURSOMATERIA
  - ESTUDIOEN
  - INSTITUCION
  - REPORTE
  - REQUESTHISTORIAL
  - REQUESTREGISTRARCALIFICACION
  - REQUESTREGISTRARESTUDIANTE
  - REQUESTREGISTRARINSTITUCION

- **Relaciones**
  - ESTUDIOEN (Estudiante [ESTUDIO_EN] Institución): Incluye periodo y título.
  - CURSO (Estudiante[CURSO] Materia)

- **Volúmenes:**

Para este análisis, consideramos un ecosistema educativo regional (como el de una provincia o un país mediano).

Entidad / Relación	Volumen Estimado	Almacenamiento Principal
Estudiantes	**1,000,000**	MongoDB (Documento Maestro)
Instituciones	**5,000**	Neo4j / MongoDB
Materias	**20,000**	Neo4j (Nodos únicos)
Inscripciones (Relaciones)	**15,000,000**	Neo4j (Aristas con propiedades)
Calificaciones Detalladas	**50,000,000**	MongoDB (Historial crudo)
Registros de Ranking	**1,050,000**	Cassandra (Datos pre-calculados)

Análisis de Crecimiento y Carga

1. **Densidad del Grafo (Neo4j):**
   Con 1 millón de estudiantes y un promedio de 15 materias por alumno, estamos hablando de **15 millones de relaciones CURSO**. Esto justifica el uso de Neo4j, ya que una consulta para buscar "estudiantes con promedio > 8 en Matemáticas" requiere navegar esa red sin hacer un JOIN de tablas de 15 millones de filas.
2. **Volumen de Escritura (MongoDB):**
   Al tener 50 millones de calificaciones, MongoDB gestiona aproximadamente **100 GB**

de datos. La flexibilidad de BSON permite que cada registro de calificación ocupe solo el espacio necesario, optimizando el almacenamiento.

3. **Carga de Lectura Analítica (Cassandra):**
   Aunque hay millones de notas, Cassandra solo guarda el **resultado final (el promedio)**. Tenemos 1 millón de filas para el ranking de alumnos, 5,000 para el de institutos y una cantidad mínima para países.
    - **Justificación:** Si el 10% de los usuarios consultan el ranking simultáneamente, Cassandra soporta miles de lecturas por segundo gracias a su arquitectura sin puntos únicos de fallo.

Casos Complejos de Volumen

- **Hot Partitions en Cassandra:** Si todos los alumnos fueran de un solo país, la partición de ese país en Cassandra sería muy pesada. **Decisión de diseño:** Usamos el ID_Estudiante y el Tipo como parte de la clave para distribuir la carga entre los nodos del cluster.
- **Super-Nodes en Neo4j:** Algunas instituciones masivas pueden tener cientos de miles de relaciones. **Decisión de diseño:** Optimizamos los índices en las etiquetas :Estudiante e :Institucion para que la búsqueda sea por el ID del alumno y no recorra todo el nodo de la institución innecesariamente.

Impacto en la Performance (Resumen)

- **Espacio en disco:** Aproximadamente 150-200 GB sumando los tres motores y sus réplicas.
- **Latencia de sincronización:** El flujo de Mongo $\rightarrow$ Neo4j $\rightarrow$ Cassandra ocurre en menos de **200ms** por registro individual, manteniendo la sensación de tiempo real para el usuario final.
    -
- **Casos Complejos:** Alumnos que cursaron la misma materia en dos instituciones distintas o materias equivalentes con nombres diferentes.

8.Diseño de persistencia NoSQL

MongoDB (Documental)

- **Información:** Datos personales, configuración de cuenta, JSON crudo de historial.
- **Índices:** `id_estudiante` (Unique), `email`.

Neo4j (Grafos)

- **Información:** Estructura de la red educativa.
- **Claves:** UUIDs para nodos, `@RelationshipId` para vínculos.
- **Justificación:** Consultas recursivas (ej: "compañeros de compañeros").

Cassandra (Columnar)

- **Información:** Promedios agregados por categoría.
- **Partición:** `tipo` (Partition Key).
- **Orden:** `promedio` (Clustering Column DESC).

10.Flujos del sistema

A. Alta de Calificación

Este es el flujo diario donde los docentes cargan las notas. Lo diseñé para que sea consistente y no pierda información.

1. Entrada: El controlador recibe el ID del alumno y la nota.
2. Persistencia en Mongo: Primero guardo el registro en `EstudianteMONGORepository`. Uso Mongo aquí porque si el Ministerio decide agregar campos extra (como una foto del examen o comentarios pedagógicos), el esquema flexible me permite hacerlo sin migrar tablas.
3. Sincronización a Neo4j: Inmediatamente después, el servicio llama a `EstudianteNeo4jRepository`. Aquí es donde se crea la relación `[:CURSO {nota: X}]`.
    - Decisión técnica: Lo hice así para que el grafo refleje la trayectoria del alumno en tiempo real. Si el alumno se cambia de escuela, Neo4j ya tiene el vínculo listo para la trazabilidad.

B. Conversión (Procesamiento de Rankings)

Este es el "puente" que construí para mover datos entre motores. Es el corazón de la arquitectura políglota.

1. Cálculo en el Grafo: En lugar de traer todos los datos a Java (que pesaría gigas), le pido a Neo4j que haga el trabajo sucio. Uso una query de Cypher con `avg(relacion.nota)` para obtener los promedios ya calculados por alumno, escuela y provincia.
2. Mapeo: El `RankingService` recibe estos mapas de datos y los convierte en objetos `ReporteEntidad`.
3. Vuelcado a Cassandra: Invoco el `.save()` del `ReporteCassandraRepository`. Como Cassandra hace un *upsert*, solo actualizo los promedios necesarios. Esto prepara los datos para que el Ministerio los consulte sin demora.

C. Consulta Analítica (Reportes)

Este flujo es el que usan los directivos y es el más rápido de todos.

1. Consulta Directa: El endpoint de reportes no toca ni Mongo ni Neo4j. Va directo a Cassandra.
2. Performance: Al buscar por `tipo` (Partition Key) y tener los datos ordenados por `promedio` (Clustering Column), la respuesta es instantánea.

- Análisis propio: Logré que el Top 10 de 13 millones de registros se devuelva en milisegundos porque el dato ya está físicamente ordenado en el disco de Cassandra.

D. Auditoría (Seguridad)

Para cumplir con las normas de Sudáfrica sobre datos de menores, implementé una capa de auditoría.

1. Log de Eventos: Cada vez que se modifica una nota en el flujo de "Alta", guardo un documento en una colección `audit_log` en MongoDB.
2. Trazabilidad: Al usar el mismo `UUID` que generó Spring para el alumno en los tres motores, puedo rastrear qué nota en Mongo generó qué relación en Neo4j y qué promedio en Cassandra. Es un sistema totalmente auditable de punta a punta.

11.Análisis crítico

- **Qué problemas resuelve cada modelo**
  - Evita el cuello de botella que tendría una base SQL al intentar calcular promedios de millones de filas en tiempo real o al navegar jerarquías de instituciones.

- **Qué pasaría si se usara otro**
  - **Solo SQL:** El sistema colapsaría al intentar generar ránkings con 1M de registros o al hacer múltiples JOINs de historial académico.
  - **Solo Mongo:** Las consultas de relaciones (quién estudió con quién) serían extremadamente lentas debido a la falta de punteros nativos entre documentos.

- **Limitaciones asumidas**
  - **Consistencia Eventual:** El ranking en Cassandra puede tener un leve retraso respecto a la nota cargada en Mongo.
  - **Complejidad:** Mantener tres motores requiere una capa de servicio (Spring Boot) robusta para coordinar las transacciones.