|  |  |
| --- | --- |
| *!!!* | **Esta prueba sólo la pueden realizar los estudiantes que han aprobado la Evaluación Continua** |

**Ficha técnica de la prueba de síntesis**

* No es necesario que escribas tu nombre. Una vez resuelta la prueba final, solo se aceptan documentos en formato .doc, .docx (Word) y .pdf.
* Comprueba que el código y el nombre de la asignatura corresponden a la asignatura de la que te has matriculado.
* Tiempo total: **1 hora** Valor de cada pregunta: **2,5 puntos**
* ¿Se puede consultar material durante la prueba?  ¿Qué materiales están permitidos? **Todos**
* ¿Puede utilizarse calculadora? ¿De qué tipo?
* Si hay preguntas tipo test, ¿descuentan las respuestas erróneas?  ¿Cuánto?
* Indicaciones específicas para la realización de esta prueba de síntesis: **Ver texto a continuación.**

### Esta prueba de síntesis tiene 6 preguntas. Cada pregunta vale 2,5 puntos.

### Para cada pregunta, entre paréntesis, se indicará el bloque temático sobre el qué trata.

### Cada persona deberá escoger y resolver las 4 preguntas con las que se sienta más cómoda.

### Las preguntas sobre Cassandra, Neo4j y MongoDB planteadas en esta prueba usan las bases de datos que has utilizado en la práctica. Por ello, te recomendamos tener la máquina virtual y Neo4j arrancados para poder trabajar con las preguntas relacionadas.

### Enunciados

Juan Luis Acebal Rico

### Pregunta 1 (Bases de datos NoSQL)

Explica, bajo tu punto de vista y utilizando tus propias palabras, las ventajas e inconvenientes de las bases de datos de tipo *schemaless*. Indica de forma justificada en qué escenarios pueden ser útiles y en qué escenarios deberían evitarse.

La respuesta no puede ocupar más de una página.

La primera ventaja es que tienen un esquema flexible, es decir no tienes que definir de forma explicita y corsetada el esquema de los datos, además es más rápido hacer modificaciones, inserciones, etc ya que, no tenemos que hacer atención a estar dentro de un esquema definido como es el caso de una bd SQL, donde se definen las tablas antes de la inserción de datos.

Además creo que es más escalable, ya que una base de datos por ejemplo de clientes de una gran multinacional, con millones de líneas, cuando cambian las necesidades de negocio, pueden cambiar los campos necesitados, y requeriría adaptar quizás cientos de tablas, y si no hay esquema, eso hace que puedas aumentar la cantidad de datos, sobretodo horizontalmente, posibilitando mucho más volumen de datos. Por último, se pueden adaptar a datos semiestructurados, donde la definición de los campos normalmente viene definido en cada dato, en este caso pongo de ejemplo JSON, si guardamos los datos en JSON, un cliente puede tener unos campos y otro, otros campos, y si con el tiempo cambian las necesidades de negocio, se puede utilizar otros campos nuevos y dejar de usar algunos de los existentes.

Sin embargo, hay inconvenientes, para transacciones ACID, entre otras, no lo veo la mejor opción, ya que puede provocar inconsistencia de los datos, por ejemplo, para transacciones financieras, o movimientos bancarios, donde tiene que ser claro, si la transacción se cumple y esta debe ser un esquema rígido y concreto.

Otros inconvenientes pueden ser a la hora de buscar datos mas complejos que impliquen algunos a muchos JOINS, puede ser complicadas o difíciles de gestionar esas consultas. Además diría que la complicación de acceder a los datos, además de la falta de integridad, puede ocasionar que de forma frecuente tengamos información duplicada, eso puede aumentar el costo de almacenamiento, y crear situaciones difíciles, tales como un almacen de datos que no se puede eliminar, pero que tiene una cantidad de información muy grande para una compañía, pero con mucha información duplicada, y que sea onpremise o cloud, tiene un costo y que se tiene que pagar. Por último, no es aún maduro, las bases de datos sin esquema definido, para ciertos entornos se usan, pero para otros como para bussisnes inteligence, que son datos SQL y con esquemas fijos, está aún sin uso. Yo las utilizaría en proyectos no maduros, tales como un nuevo proyecto que aún no se sabe las necesidades a largo plazo, aplicaciones que se necesita analizar o majerar datos en tiempo real, para datos que no está estructurado o semi estructurado, tales como sensores o logs de una aplicación, o contenido que varía en el tiempo, y no lo usaría en banca, negocios, que se requiere una consistencia en las transacciones, y en relaciones complejas que requieran un análisis avanzado. Tampoco en cualquier entorno con requisitos y controles de integridad definidos.

### Pregunta 2 (Modelos de datos NoSQL y Distribución de datos)

Una empresa nacional quiere crear un sistema que permita analizar las colaboraciones que realizan sus trabajadores entre sí. Para ello, quiere almacenar todas las conversaciones entre trabajadores. Para cada conversación se almacenará información sobre el tipo de conversación (chat, videoconferencia o email), los participantes de la conversación y las interacciones de la misma. Además, la empresa quiere analizar los datos, no sólo a nivel de usuario, sino también a nivel de departamento, sucursal y área geográfica. Por lo que se quiere guardar una jerarquía que indique a qué departamento y sucursal pertenece cada usuario y en qué área geográfica trabaja.

Los datos serán accesibles sólo por los analistas de la empresa, que se encuentran ubicados en una sede de Barcelona.

Debido a las características del sistema se desea que este ofrezca una alta consistencia. La disponibilidad es importante, pero no tanto como la consistencia de los datos en esta primera etapa del proyecto, donde el objetivo es obtener información precisa de las colaboraciones.

**Se pide contestar a las siguientes preguntas de forma argumentada** para el caso anterior:

1. Describe las ventajas e inconvenientes de utilizar cada uno de los siguientes modelos: Relacional, NoSQL de agregación y NoSQL en grafo.
2. Describe la arquitectura de fragmentación y distribución que propondrías.
3. Indica qué estratégia de replicación propondrías: básicamente, si se guardarán los datos replicados, qué datos se replicarán, donde y cómo se gestionarán dichas réplicas.

1)

Relacional. Aquí tenemos la fuerte consistencia de los datos y las transacciones ACID. Ideal para datos estructurados y con relaciones definidas y claras. Además es un sector maduro desde hace ya decenas de años donde hay mucha cantidad de herramientas y experiencia. Como inconvenientes no es flexible para datos semiestructurados o cambios frecuentes en el esquema. Y tienen un escalado horizontal complicado.

NoSQL agregacion. Aquí tenemos que es muy flexible y es fácil de usar en datos semiestructurados, además es muy adecuado para usarse en datos basados en agregaciones y datos que tienen una alta variedad. Tiene un escalado horizontal sencillo. En contra, es complicado o no está tan maduro las transacciones mas complejas, y eso hace también que sea complicado manejar relaciones muy profundas.

NoSQL grafo. Muy interesante para relaciones complejas que tienen relaciones entre sus entidades. Las consultas de grafos son interesantes también para conocer las vías de interconexión entre entidades (ya que todas las entidades tienen el mismo nivel), y como inconveniente no es eficiente si los datos están muy interconectados, y el escalado es mas complicado que con otro tipo de bases de datos.

2)

La arquitectura de fragmentación es como dividimos los datos entre diferentes nodos, para que a la hora de hacer consultas y agrupaciones tengamos más velocidad. Por ejemplo podemos dividir los datos por área geográfica y sucursal

3)

Podemos replicar todo el cunjunto de datos, debido a la importancia dde la alta consistencia, con nodos de solo lectura. El cluster principal estaría en el centro de datos, y las replicas remotas en otras zonas geográficas, es decir, mantener una alta consistencia equilibrándolo con disponibilidad.

### Pregunta 3 (Distribución de datos)

Supón que tenemos un sistema de gestión de réplicas mediante quórums con las siguientes configuraciones:

C1: N=10, W=5, R=4

C2: N=10, W=5, R=5

C3: N=10, W=6, R=5

C4: N=10, W=10, R=1

C5: N=10, W=1, R=1

C6: N=10, W=1, R=12

**Se pide**:

1. Indica y justifica, para cada configuración, qué tipo de consistencia provee (final en el tiempo, fuerte o estricta). Si hay alguna configuración errónea, identificala y comenta el porqué.
2. Supón que te piden implementar un sistema de banca online, donde se requiere una consistencia fuerte. Indica y justifica la configuración que escogerías.

1.

C1 (N=10, W=5, R=4): 5+4=9 < 10 , consistencia eventual.

C2 (N=10, W=5, R=5): 5+5=10 = N , no garantiza lectura del último valor (no fuerte).

C3 (N=10, W=6, R=5): 6+5=11 > 10 , consistencia fuerte.

C4 (N=10, W=10, R=1): 10+1=11 > 10, fuerte (aunque poco práctico).

C5 (N=10, W=1, R=1): 1+1=2 < 10 , eventual.

C6 (N=10, W=1, R=12): inválida (R> N).

2.

Para banca online (fuerte), elegiría C3 (W=6, R=5) porque W+R> N y no fuerza a escribir en todas las réplicas.

### Pregunta 4 (MongoDB)

### Teniendo en cuenta la base de datos que has utilizado en la práctica, se pide crear una consulta en MongoDB que devuelva las 10 cuentas (*accounts*) con más productos asociados (*products*).

### Pregunta 5 (Cassandra)

De acuerdo a  las siguientes dos tablas en Cassandra, creadas a partir de los datos utilizados en la práctica.

--  Tabla 1

CREATE TABLE AGGREGATE\_1

(

YEAR\_OCC TEXT,

MONTH\_OCC TEXT,

DAY\_OCC TEXT,

TIME\_OCC TEXT,

SUB\_AREA TEXT,

WEAPON\_DESC TEXT,

TOTAL\_COUNT INT,

PRIMARY KEY (WEAPON\_DESC, YEAR\_OCC, MONTH\_OCC, DAY\_OCC, TIME\_OCC, SUB\_AREA)

);

COPY CRIMES.AGGREGATE\_1 (YEAR\_OCC, MONTH\_OCC, DAY\_OCC, TIME\_OCC, SUB\_AREA, WEAPON\_DESC, TOTAL\_COUNT) FROM

'/home/student/Escritorio/Cassandra/data/YEAR\_MONTH\_DAY\_TIME\_AREA\_SUB\_AREA\_WEAPON.csv'

WITH DELIMITER=',' AND HEADER=TRUE;

-- Tabla 2

CREATE TABLE AGGREGATE\_2

(

YEAR\_OCC TEXT,

MONTH\_OCC TEXT,

DAY\_OCC TEXT,

TIME\_OCC TEXT,

SUB\_AREA TEXT,

WEAPON\_DESC TEXT,

TOTAL\_COUNT INT,

PRIMARY KEY (WEAPON\_DESC, YEAR\_OCC, MONTH\_OCC)

);

COPY CRIMES.AGGREGATE\_2 (YEAR\_OCC, MONTH\_OCC, DAY\_OCC, TIME\_OCC, SUB\_AREA, WEAPON\_DESC, TOTAL\_COUNT) FROM

'/home/student/Escritorio/Cassandra/data/YEAR\_MONTH\_DAY\_TIME\_AREA\_SUB\_AREA\_WEAPON.csv'

WITH DELIMITER=',' AND HEADER=TRUE;

Se ejecutan las dos consultas siguientes y se obtienen los resultados que se muestran a continuación:

cqlsh:crimes> SELECT WEAPON\_DESC, YEAR\_OCC, sum(TOTAL\_COUNT)

          ... FROM AGGREGATE\_1

          ... WHERE WEAPON\_DESC = 'ASSAULT WEAPON/UZI/AK47/ETC'

          ... GROUP BY WEAPON\_DESC, YEAR\_OCC;

 weapon\_desc                 | year\_occ | system.sum(total\_count)

-----------------------------+----------+-------------------------

 ASSAULT WEAPON/UZI/AK47/ETC |     2020 |                      **26**

 ASSAULT WEAPON/UZI/AK47/ETC |     2021 |                      **12**

 ASSAULT WEAPON/UZI/AK47/ETC |     2022 |                      **20**

 ASSAULT WEAPON/UZI/AK47/ETC |     2023 |                      **16**

cqlsh:crimes> SELECT WEAPON\_DESC, YEAR\_OCC, sum(TOTAL\_COUNT)

          ... FROM AGGREGATE\_2

          ... WHERE WEAPON\_DESC = 'ASSAULT WEAPON/UZI/AK47/ETC'

          ... GROUP BY WEAPON\_DESC, YEAR\_OCC;

 weapon\_desc                 | year\_occ | system.sum(total\_count)

-----------------------------+----------+-------------------------

 ASSAULT WEAPON/UZI/AK47/ETC |     2020 |                      **17**

 ASSAULT WEAPON/UZI/AK47/ETC |     2021 |                       **8**

 ASSAULT WEAPON/UZI/AK47/ETC |     2022 |                      **10**

 ASSAULT WEAPON/UZI/AK47/ETC |     2023 |                      **10**

Aún habiendo cargado los mismos datos en los dos agregados, los resultados de las consultas son diferentes. **¿Podrías explicar qué ha pasado y porque los resultados son diferentes en las dos consultas?**

Aquí pasa es que no estamos en SQL, normalmente en sql cuando una clave es primaria, y está definida como única, dependiendo del motor de sql utilizado, normalmente dará error la inserción, en cassandra sobreescribe si tenemos mas de una fila con la misma clave única. En casandra la ultima inserción con la misma clave primaria remplaza la fila que existía.

El primer agregado usa como clave primaria (WEAPON\_DESC, YEAR\_OCC, MONTH\_OCC, DAY\_OCC, TIME\_OCC, SUB\_AREA) y el Segundo agregado usa como clave primaria (WEAPON\_DESC, YEAR\_OCC, MONTH\_OCC), esto quiere decir que cada combinación de esas 6 columnas para el primer agregado, crean cada fila única, y para el segundo agregado, es la combinación de esas 3 columnas. Por tanto es normal que el primer agregado sea por días y subareas y el segundo agregado sea por meses, aparte en ambos la descripción del arma. A la hora de hacer el sum, como cualquier otra función de agregado, el primer agregado tendrá mas filas y mostrará más resultados, después al usar group by, lo que hace es usar solamente las filas únicas, de año y weapon\_desc, que son las que realmente se insertaron en ambas tablas, las filas únicas con la misma combinación de primary key, el resto en el proceso de inserción se sobrescribieron aunque en la consulta usemos año y weapon, al contabilizar ambos agregados, el primero hemos perdido menos registros que en el segundo agregado, y luego el group by si que hace el sumatorio por años pero con menos registros iniciales. Me quedaría por decir que el segundo agregado es demasiado disperso o amplio, cassandra debido a su forma de tratar los datos debe tener una clave primaria a medida de lo que realmente son los datos.

### Pregunta 6 (Neo4j)

Hay interés por conocer las autorías comunes de dirección y producción en una misma película de la saga, es decir, las películas (o series) cuyo director haya sido también su productor.

En ese sentido, **se solicita el listado de películas (mostradas alfabéticamente por orden ascendente) dirigidas y producidas (en solitario o no) por una misma persona**. La primera columna debe presentar el título de la película, la segunda el nombre del productor y director y, la tercera, un listado con el resto de productores de la película (si existen) entre los cuales no se ha de incluir el nombre del director/productor de la columna anterior, ya que sería redundante.