BadCats - Week4

Practical

[CREATE A MIGRATION FILE WITH THE REQUIRED SQL STATEMENTS FOR CREATING THE TABLES]

1) Mohamed Ahmed Hassan (Mohamed Hassan) [in Udacity Courses before]

https://github.com/mohamedlolx/Migration-Practical-Task.git

Mohamed Hassan | LinkedIn

2) Yousef elsayed Elhamaki [ ]

https://github.com/yousefhamaki/migration.git

3) Omar Elsayed [please change it to your full name]

< please add your weekly task link here >

4) Shaimaa Adel

Migrations assignment

5) Burham B. Soliman

BadCats - Week4

Practical

[CREATE A MIGRATION FILE WITH THE REQUIRED SQL STATEMENTS FOR CREATING THE TABLES]

1) Mohamed Ahmed Hassan (Mohamed Hassan) [in Udacity Courses before]

https://github.com/mohamedlolx/Migration-Practical-Task.git

Mohamed Hassan | LinkedIn

2) Yousef elsayed Elhamaki [ ]

https://github.com/yousefhamaki/migration.git

Database Migration File structure @ GitHub

Theoretical

[COME UP WITH A SCENARIO WHERE USING NoSQL DATABASES MAKES MORE SENSE]

1) Mohamed Ahmed Hassan(Mohamed Hassan) [in Udacity Courses before]

[Mohamed Ahmed Hassan]

Mohamed Hassan | LinkedIn

mohamedlolx (Mohamed Hassan) (github.com)

BadCats - Week4

[Scenarios where using No SQL databases makes more sense]

Scenario 1

- Imagine a database to model building a furniture instruction where you would have the object "furniture" which would have a list of "materials" and have a list of "instructions"
- Furniture - would simply have a name that has a list of Materials and Instructions.
- Materials - would be a name + quantity, maybe we can even have a "Material Category" table as well.
- Instructions - would simply be an ordered list of texts.
- Create a table called "Furniture", "Material" and "Instruction" and the appropriate columns.

Maybe think of extending the data stored to include information on how many people are required to build it? Difficulty level? How much time would it take?

I think NoSQL is the suitable for this scenario

Scenario 2

Imagine a database to model a user database with basic information (name, email, phone number, etc), but you also want to have the flexibility of being able to add any custom fields as you wish.

Think of different systems consuming this user database, each system will want to have their own custom attribute to be attached to the user

My inclination would go the RDBMS way:

- Create a table for "USER" with columns: ID, name, email, phone.
- Create a table for "USER_ATTRIBUTE" with columns: ID, USER_ID, attr_name, attr_type, attr_value.

BadCats - Week4

- The USER_ATTRIBUTE will allow that customization and flexibility without having to shut down the system, alter the database and restart it.

  I think NoSQL is the suitable for this scenario

Notes:
There are several possible reasons that companies go NoSQL:

- The most common scenario is probably when one database server is no longer enough to handle your load.
- NoSQL solutions are much more suited to distribute load over database servers.

- That means that you have multiple slave databases that watches a master database for changes and replicate them to themselves.
- Reads are made from the slaves, and writes are made to the master.

- This works to a certain level, but it has the annoying side-effect that the slaves will always lag slightly behind, so there is a delay between the time of writing and the time that the object is available for reading, which is complex and error-prone to handle in your application.

- Also, the single master eventually becomes a bottleneck no matter how powerful it is. Plus, it's a single point of failure.

- NoSQL generally deals with this problem by sharding.
-
- Overly simplified it means that users with user_id: 1-1000000 are on server A, and users with user_id: 1000001-2000000 are on server B and so on.

- This solves the problems that relational replication has, but the drawback is that features such as aggregate queries (SUM, AVG etc) and traditional transactions are sacrificed.

BadCats - Week4

[Relational Database Vs NoSQL]

NoSQL Databases excel in their ability to store data in a non-structured form as documents or key-value pairs. They allow for denormalized storage.

Relational Databases, on the other hand, require the data to be stored in a structured and normalized way.

While it may seem that NoSQL Databases save a lot of time in database definition initially in the development process, the fact remains that a well-defined database schema with a Relational Database can give a sizable performance advantage in some cases.

What are Relational Databases?

- Relational Databases excel at their ability to store structured data.
- They provide comprehensive querying layers and store data in the minimum footprint possible by denormalizing the data.
- Since data is denormalized, accessing them requires complex joins.
- Relational Databases have existed for ages and they are proven
- n a wide variety of applications.
- There are several licensed and open-source Relational Databases out there and all of them are mature enough to be considered for production-grade applications.

Relational Databases strictly comply with ACID guarantees and hence are a good choice for transactional data.

Relational Databases are mainly based on a single node design.

That said, lately, most of the popular ones have been adding cluster support through the use of Sharding.

But none of them are as elegant as their multi-node NoSQL counterparts since partition tolerance is built into their foundation.

Sharding leads to increased costs and requires close management.

It is safe to say that most Relational Databases prioritize consistency and availability over partition tolerance.

This means Relational Databases are not very good at handling a large amount of data since the data they have to be stored in a single system or need constant babysitting because of their limitations when it comes to multi-node operation.

What are NoSQL Databases?

NoSQL Databases are great at storing semi-structured or unstructured data since they don't enforce a concrete schema for tables.

This means data attributes can be added on the fly without changing the structure of the entire table or adding redundant elements to the rest of the rows.

Since there is no particular structure enforced by the database, they are also not very good at join queries.

NoSQL Databases recommend data to be stored in a format in which it will be frequently accessed.

This helps in defining the database very close to the UI layer or where the data will be actually used or reported.

NoSQL Databases are great at scaling horizontally and partition tolerance is built into their foundation.

NoSQL Databases do well in scenarios where sub-second response time for high data volume is required.

NoSQL Databases achieve this by compromising consistency and referential

integrity.

Most NoSQL Databases support only eventual consistency and are hence not a great choice for transactional operations.

NoSQL Use Cases:

1- Content Management (CM):

Content management is a set of processes for the collection, managing, delivery, retrieval, and publishing of information in any format including text, images, audio, and video.

NoSQL databases can offer a better option for storing multimedia content with their flexible and open-ended data model.

For example, Forbes built a custom content management system based on MongoDB in just a few months, giving them greater agility at a lower cost.

2-Real-Time Big Data Processing:

Big data refers to a dataset that is too large to be handled through traditional processing systems.

Systems that store and retrieve big data in real time use stream processing to ingest new data while analyzing historical data, a series of functions that are ideally suited to NoSQL databases.

Zoom used DynamoDB (on-demand mode) to enable its data to scale without performance issues, even as the service saw usage spike in the early days of the COVID-19 pandemic.

3-Internet of Things (IoT):

IoT devices have embedded software and sensors that are connected to the internet or communications networks, and are able to collect and share data without human

intervention.

With billions of devices generating untold amounts of data, IoT NoSQL databases provide scalability and more flexible schema to IoT service providers.

One such service is Freshub, which switched to MongoDB from MySQL to better deal with its large, dynamic, non-uniform datasets.

4-Mobile Applications:

With billions of smartphone users, scalability is becoming the biggest challenge for businesses that deliver services on mobile devices.

A NoSQL DBMS, with more flexible data models, is often the perfect solution.

For instance, The Weather Channel uses a MongoDB database for handling millions of requests per minute while also processing user data and delivering weather updates.

5-NoSQL Databases Solve Modern Problems:
Data management has become one of the most important issues facing businesses today. NoSQL databases are designed to**.**

Some popular NoSQL databases include:

1-MongoDB.	2-Amazon DynamoDB.	3-Apache Cassandra.
4-Apache CouchDB.	5-Apache HBase.	6-RavenDB.
7-Elasticsearch.	8-Redis.	9-OrientDB.
10-Neo4j.	11-Cosmos DB.	12-HyperTable.

BadCats - Week4

Practical :

https://github.com/mohamedlolx/Migration-Practical-Task.git

Theoritical Please Have A look on the Original One:

https://drive.google.com/file/d/1Ah02CkcflxMmoH67Mk0J-wR3LnasBYfU/view?usp=sharing

Sources: Relational Database Vs NoSQL: 7 Critical Aspects (hevodata.com)

SQL vs. NoSQL Databases: What's the Difference? | IBM

NoSQL Databases | When to Use NoSQL vs SQL | ServerWatch

BadCats - Week4

2) Yousef Hamaki [please change it to your full name]

< please put your weekly task here >

3) Omar Elsayed [please change it to your full name]

< please put your weekly task here >

4) Burham B. Soliman

- Advertising system, which gathering information about clients in a random way, then re-use it later to show up the best matched advertises that related to user gathered data.