DSO553 - Overview

Overview:

- Big Data Landscape
- What is NoSQL
- MongoDB

References:

- And this link as well
- Database Design

Instructions:
Below are a mixture of typical NoSQL questions you may see in an interview. Lets review them together.

Question 1 - What are the differences between NoSQL and RDBMS?

RDBMS
-Primary key: unique and acts as foreign key in other tables for joins
-Predefined schema
-Values are domain-specific, column-homogenous, and not multi-value
-Names of columns within a table are unique
-ACID transactions:
Example --- I pay someone $5 for coffee, and $1000 for rent. Transaction occur separately and one at a time while preserving consistency of the data. Data integrity.
-Vertical Scaling: scaling up, it's about increasing the capacity of your existing hardware

NoSQL
-No Joins
-Schema-less
-Various data types e.g. Unstructured data(video, images, sensor data)
-Storage Types for given use case e.g. document, graph, column, key-value
-BASE → most important point is that the system is eventually consistent. In addition, data is mostly available and at some date, depending on business logic of implementation, eventually consistent in a distributed system. In fact, there are tools that can predict at which point in time a system can reach consistency.

-Horizontal scaling: Solve a traffic jam by constructing more lanes
Vertical scaling: Classroom can add more chairs but has a threshold/limit

Web-scale: huge volume of data due to the web

Q2 - What do you understand by "Polyglot Persistence" in NoSQL?

-**3Vs:** Velocity, Volume, Variety (extra: Veracity)
*More clarity on above:

- Volume: The overall size of the data set
- Velocity: The rate at which the data arrives and also how fast it needs to be processed
- Variety: The wide range of data that the data set may contain—that is, web logs, audio, images, sensor or device data, and unstructured text, among many others types

-Choosing right tools for the kind of data + goal trying to achieve
Polyglot persistence => Picking the right language for a particular problem can be more productive rather than trying to fit all the aspects of that problem into a single language. Hence, polyglot persistence is the term which is used to define this hybrid approach to data persistence.

Pros of Polyglot Persistence
-Performance. The main advantage is the best performance of the application because each component manages their data using the most appropriate data model and system.
Cons:
-complexity managing data retrieval → requires more resources
-data duplication between data stores/storages/databases

Q3 - List the different kinds of NoSQL data stores.
-document, graph, search, key-value, column
-Unstructured data, eventually consistent, schema-less

Key-value:
Values - are singular in nature for the most part and are not “multi-valued” with exception to Redis
Very performant and fast
Simple queries on keys, not values

Column:
Column-based and great for said simple aggregations e.g. maximums, minimums, sums, averages
No deep analytical queries
No joins

Graph:
Nodes, Relationships, Properties
Focused on/built for systems that have lots of relationships (or “joins” in relational system)
Examples: Product recommendations, Social networking
Built for fast traversals -> going from one node to another

Document:
-collections ~ tables(sql), documents ~ rows(sql), key:value pairs
-complex data types such as arrays of objects (e.g. list of dictionaries) which is known as the process of denormalization
-Example: product catalogs, metadata, logs

Q4 - What is CAP theorem? How is it applicable to NoSQL systems and data storage ?

distributed systems: stuff that runs on multiple servers

CAP Theorem:
Consistency: all nodes/servers in a system have the same view of the data at any time
Availability: in the event of some node failure, database remains operational and still sends/receives responses (though responses might be inaccurate or stale)
Partition tolerance: database stays operational if system/network fails and where one group of servers are unable to communicate with another group of servers (think about the geographical examples I gave regarding servers not being co-located)

Q5 - What do you mean by eventual consistency in NoSQL stores?

For banks, consistency is King, so some sort of latency (i.e. lag) penalty is unavoidable and acceptable.

However, for many websites, such as social networks, emails, chatrooms, and certain e-commerce operations (Amazon), this worldwide synchronous consistency is unnecessary. Ultimately, it doesn’t matter if my friend in Australia can see my tweet a few seconds before my friend in America. As long as both friends can see the tweet eventually, I’m happy.

Social feed - not everyone sees the same thing at once, but no biggie

Fine-tuning the consistency to meet your needs

EC is OK
-Social media, chatrooms, notifications, emails, weather reporting
EC is bad
-stock market trading, venmo payments, hospital data

Q6 - What kinds of questions should you be asking when you begin to think about your Big Data implementation?

-Performance: speed, how fast do you need your system to be? Latency - speed of event occuring
-Availability: do you need 100% uptime?
-Scalability: data storage + how much compute power do you need?
-Flexibility: types of data + changing of data structure AND how quickly can you add more resources
-Cost: what can you afford and what can you justify

Q7 - What is the Hadoop ecosystem and what necessitated its appearance?

-distributed, powerful, widely used, can parallelize jobs, data storage mechanism
-moving data from storage to where you can run computations/queries, so that queries run where data exists and not the other way around
-Pre-Hadoop: lots of data loss because raw files were transformed during ETL (extract-transform-load) process

Power of Hadoop:
Economical
Massive scalability
Reliable
Schema on read

Distributed storage(HDFS) - where all raw files get stored. **Storage**
Distributed processing (MapReduce) - process large data sets using parallel algorithms. **Processing.**

Think of Hadoop vs Relational architecture like a restaurant:
Relational System - come to kitchen, give you the cans of meat, tuna, vegetables sliced up, all nice and neat and ready to cook. Easy to do, everything predetermined, but limited
Hadoop - Give you the livestock, bag of tomatoes, bag of potatoes, etc. takes much longer and messy. but lots of flexibility and drive to insight

---

Q8 - What is a Columnar/Wide-Column database and what is a good use case?

Q9 - What is the biggest case for using Graph database and name examples ?

Q10 - What is a Key-Value database and what is a good use case?

Q11 - What is a Document database and what is a good use case?

Q12 - What are some of the "limits" of Relational Databases
See slides

1 - Large number of reads & writes
2 - Low latency response times at high volumes
3 - Flexibility in structure and data type

Q13 - OLTP vs OLAP architecture? What's the difference and can you give a real world example of the differences?

OLTP - online transaction processing
OLAP - online analytical processing

OLTP:
OLTP systems are "classical" systems that process data transactions. They are all around you. In the bank, the ATM or the computer system used by the bank teller to

record a transaction is an OLTP system, usually a database. If you text someone from your smartphone, you are working with another OLTP system. The cash register at your local supermarket runs off another OLTP system, and on it goes.

OLAP examples:

systems work with very large amounts of data. Preserving the accuracy and integrity of transactions is not their purpose; this is up to OLTP. OLAP is here to allow us to find trends, crunch numbers, and get the big picture.

Other Examples:

OLAP:

-identify sales for each dept each month
-identify top 10 selling books
-find number of classes that have had fewer than 10 students the last 10 years

OLTP:

-update account balance
-enroll in course
-add a book to amazon shopping cart