**What is Eventual Consistency?**

*Eventual consistency:*

*I watch the weather report and learn that it's going to rain tomorrow.*

*I tell you that it's going to rain tomorrow.*

*Your neighbor tells his wife that it's going to be sunny tomorrow.*

*You tell your neighbor that it is going to rain tomorrow.*

*Eventually, all of the servers (you, me, your neighbor) know the truth (that it's going to rain tomorrow), but in the meantime the client (his wife) came away thinking it is going to be sunny, even though she asked after one or more of the servers (you and me) had a more up-to-date value.*

*As opposed to Strict Consistency / ACID compliance:*

*Your bank balance is $50.*

*You deposit $100.*

*Your bank balance, queried from any ATM anywhere, is $150.*

*Your daughter withdraws $40 with your ATM card.*

*Your bank balance, queried from any ATM anywhere, is $110.*

*At no time can your balance reflect anything other than the actual sum of all of the transactions made on your account to that exact moment.*

*The reason why so many NoSQL systems have eventual consistency is that virtually all of them are designed to be distributed, and with fully distributed systems there is super-linear overhead to maintaining strict consistency (meaning you can only scale so far before things start to slow down, and when they do you need to throw exponentially more hardware at the problem to keep scaling).*

*https://stackoverflow.com/questions/10078540/eventual-consistency-in-plain-english*

**The Brewer's Theorem, most commonly known as the CAP theorem, states that in the presence of a Network Partition (the P in CAP), a system's designer has to choose between Consistency (the C in CAP) and Availability (the A in CAP). Can you think about examples of CP, AP and CA systems?**

<https://howtodoinjava.com/hadoop/brewers-cap-theorem-in-simple-words/>

**How would you explain the recent rise in interest for NoSQL?**

https://support.rackspace.com/how-to/choosing-between-rdbms-and-nosql/

**How does NoSQL tackle scalability challenges?**

*noSQL databases give up a massive amount of functionality that a SQL database gives you by it's very nature.*

*Things like automatic enforcement of referential integrity, transactions, etc. These are all things that are very handy to have for some problems, and which require some interesting techniques to scale outside of a single server (think about what happens if you need to lock two tables for an atomic transaction, and they are on different servers!).*

*noSQL databases don't have all that. If you need that stuff, you need to do it yourself, but if you DON'T need it (and there are a lot of applications that don't), then boy howdy are you in luck. The DB doesn't have to do all of these complex operations and locking across much of the dataset, so it's really easy to partition the thing across many servers/disks/whatever and have it work really fast.*

**In which case would you use a document database like MongoDB instead of a relational database like MySQL or PostgreSQL?**

https://support.rackspace.com/how-to/choosing-between-rdbms-and-nosql/