Pitfalls of Relational Databases

Relational Databases Background

- Tables holding many rows of structured data (pre-defined schema)
- Rows of one table can have a relation to rows of another if they share a common key
- Built in query optimizer that returns results using the declarative SQL language

Popular RDBMS: MySQL, PostgreSQL

Other Implementation Details

- Use B-Trees
- Support transactions with 2 phase locking for isolation
- All reads and writes go to disk

Scaling a relational database

- Vertical Scaling
  - Increasing the power of the hardware running the database (traditional approach)
- Horizontal Scaling
  - Adding more computers/nodes of similar power, distributing workload
  - Means that we have to shard/partition our dataset, this is where things get hard

The Problems with Sharding

Imagine that I’m a bank, and I have a table called Accounts

Let’s transfer $1000 from Trump to Putin, who are on different partitions

Id	Name	Balance
1	Trump	100

Id	Name	Balance
2	Putin	1000000

The Problems with Sharding

Imagine that I’m a bank, and I have a table called Accounts

Let’s transfer $1000 from Trump to Putin, who are on different partitions

Id	Name	Balance
1	Trump	100

Id	Name	Balance
2	Putin	1000000

In order to do so, we need to make sure that the operation either succeeds on both partitions, or fails on both partitions - requires a distributed transaction! We will go into this more in a subsequent video but for now just know that this means it’s a slow operation. Cross partition writes are very slow.

The Problems with Sharding Continued

I am on Facebook messenger, and I want to load the screen that shows me all of the chats that I am in. To do so, I would run a join operation.

UserId	ChatId
1	1
1	3
1	5
1	7

ChatId	ChatName
1	Cool Kids
2	Bad coders
3	Good coders
4	Poorly endowed

ChatId	ChatName
5	Well endowed
6	1 plate club
7	2 plate club
8	Not natty

To fetch all chats for user Jordan with UserID=1, we need to aggregate results from many partitions, too many network calls!

The Relational Philosophy

- One copy of every piece of data (reduce duplication)
- Each table has one preset schema
- Fetch related data via joins
- Hide concurrency bugs and partial failures via transactions

Issues with Relational Databases at Scale

- Splitting related data up over partitions/different tables becomes very problematic once network delay becomes involved
  - The need for checking each partition or using distributed transactions greatly slows things down
- The locking needed by transactions in order to enforce isolation is too slow
- B-Trees are very slow for writing compared to some in memory buffer

Moving Away from Relational Databases

The term NoSQL has arisen as a general term for any database that is not both relational and using the SQL query language.

In reality, NoSQL databases are more stripped down that relational databases, and give the developer more opportunities to choose one that fits the needs for their application, because sometimes it is better at huge scale to abandon some of the features of relational databases in exchange for greater performance.

NoSQL design patterns

- Most importantly, objects are generally self-contained documents
  - More locality on disk for both reads and writes (good for when accessing whole document)
  - Easier to shard
  - Schemaless
  - Data duplication (needs to be handled in application code)
- Graph databases
  - Good for many to many relationships, everything can be related
  - Schemaless

Relational Databases Conclusion

- Very intuitive data model that is easy to understand
- However, tend to scale poorly when sharded
  - On writes to many shards may need distributed transactions
  - On reads to many shards involves many network calls
- Transaction abstraction and locking is slow
- B-Trees are slow for writes (since they go to disk)
- Set schema
- Generally use single leader replication

As a result, many developers have chosen to use “NoSQL” databases, which relax some of these requirements and diverge from the data patterns of SQL, in the hopes of improving performance for their application. We will examine some of said databases in subsequent videos.

Relational Databases Conclusion

With all this said, this doesn’t mean that SQL is infeasible. For many read-heavy applications with highly related data, relational databases are a good solution. Many companies (see VoltDB, Google Spanner) have even tried to improve the scalability of the relational model, which some have dubbed NewSQL.

There is no one size fits all database, and it is only by knowing how each one of these popular databases work that we can determine which to use for a given application.