RX-M Cloud Native Consulting

Building Cloud Native Applications on Cloud Foundry

An in depth look at the microservices architecture pattern, containers and Cloud Foundry

6: Stateless Services and Polyglot Persistence

Objectives

- Describe the range of state managers are available for use in microservice based systems
- Explain the differences between various database types
  - Relational
  - Graph
  - Key/Value
  - Document
  - Column
- Explore the importance of Schemas

Relational Databases

- Relational databases still manage most of the world's critical data
- ACID Transactions
- Perfect for shared database integration [Hohpe and Woolf]
  - One database hosting data for a range of applications
- Single Source of Truth
- Years of refinement and evolution
  - No data platform type is more widely supported
- One system to manage
  - Concurrency is notoriously difficult to get right
  - Errors can trap even the most careful programmers
  - Relational systems scale vertically, sidestepping many concurrency concerns and managing most of the rest for you

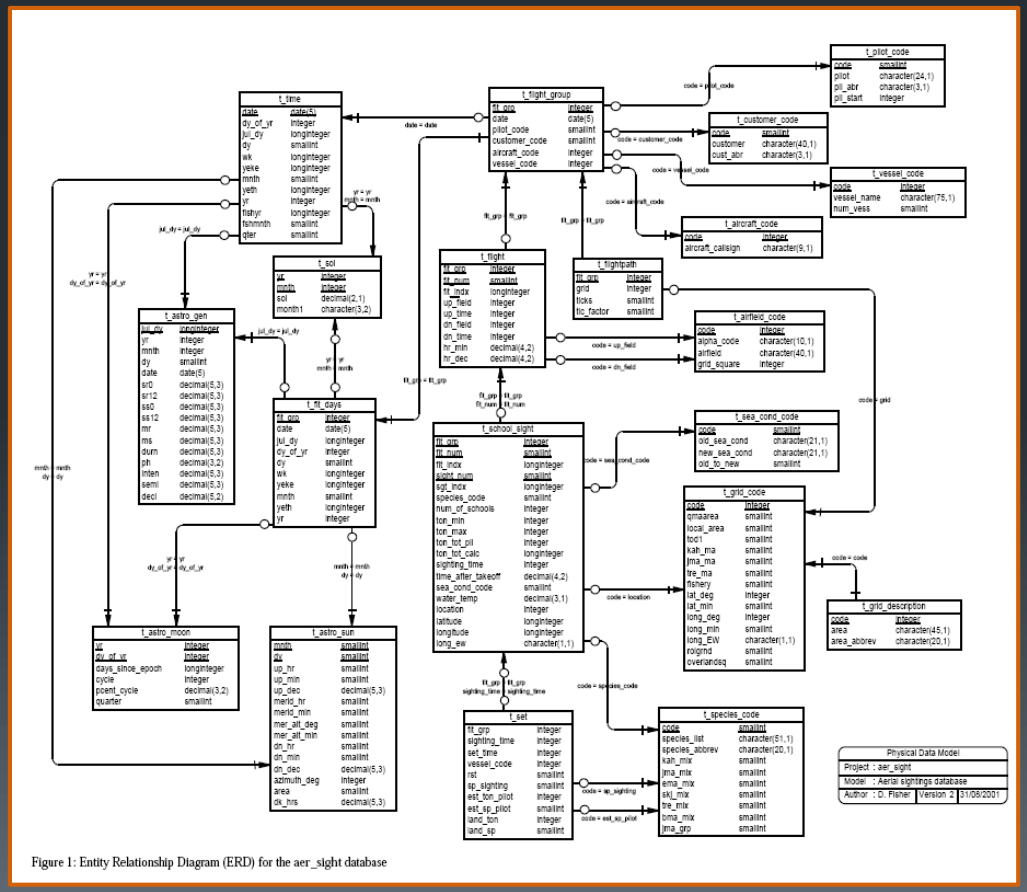
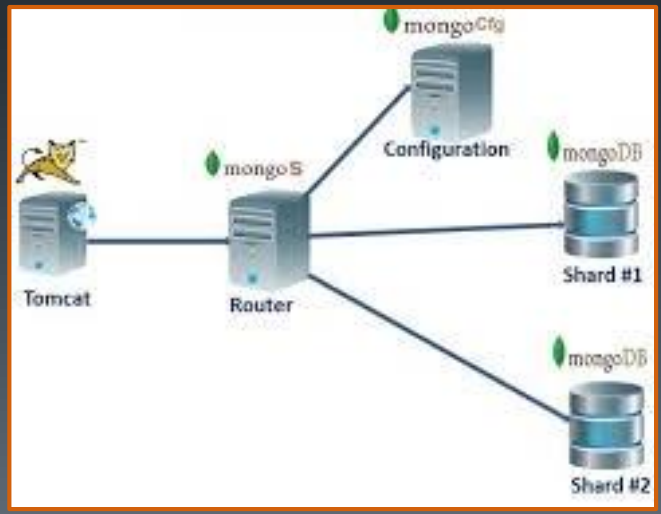


Figure 1: Entity Relationship Diagram (ERD) for the aer_sight database

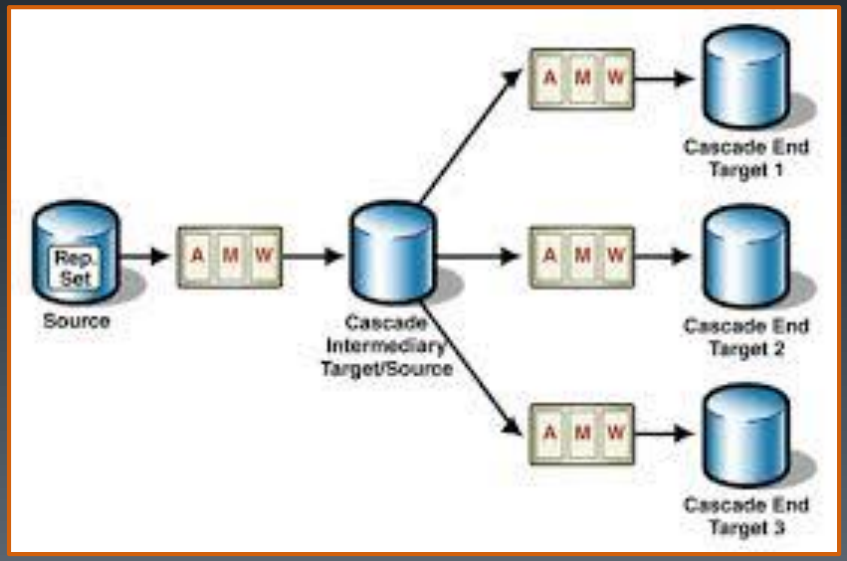
Database Scale Out

- There are two basic means used to distributed data
  - Replication
    - Copies the same data to multiple nodes
  - Partitioning
    - Vertical – sets of columns are placed on different nodes
    - Horizontal (aka Sharding) – sets of rows are placed on different nodes
- Replication and Partitioning can be combined to create an array of data distribution patterns
- Single server databases are the easiest to manage
  - Not everyone is Google
  - If a quality server and a fast database can handle your data you may be much better off keeping things simple
    - Single server solutions run on reliable hardware (not commodity)
    - Redundant power supplies and network interfaces, RAID disks, monitoring hardware, robust manufacturing QA, etc.
  - In memory cache and hot standby solutions allow single server models to run fast and reliably
  - Distributed systems create unwanted overhead, don't do it if you don't have to!
- If you plan to shard do so early and during a slow time (when you have plenty of head room)
  - Enabling partitioning on many systems will cause heavy data rebalancing and replication traffic



Replication Benefits

- Read Scale: replication enables scale out architectures to be designed involving a master server and many replicas sharing application read load
- Availability: system redundancy imparted by replication allows for highly available solutions
- Geographic Distribution: Replicas can be placed in geographically diverse locations, close to their user
- Task Offloading: Replicas present excellent platforms for backups, analytics, application development and testing environments



Why NoSQL

- Impedance mismatch
  - the difference between the relational model and the in-memory data structures is often substantial
  - This requires data translation
  - Relational data is organized into tables and rows (relations and tuples)
    - A tuple is a set of name -value pairs and a relation is a set of tuples
  - SOA services use richer data structures with nested records and lists
    - Represented as documents in XML, JSON, etc.
  - Reducing the number of server round trips makes a rich structure desirable
- Relational systems (due to joins and ACID) are difficult to scale horizontally
  - Join performance and highly structured data also makes them ill suited for today's "big data" environments
- A cluster of small machines can use commodity hardware
  - Scaling out
  - Cheaper
  - More resilient
- NoSQL systems process aggregates not tuples and are typically designed for clustering and unstructured data
  - Good support for nonuniform data
- SQL is a powerful and well understood language
  - Requires significant complexity in the underlying platform
  - Not present in NoSQL platforms
  - SQL like solutions are present on many NoSQL solutions
    - Hive
    - Cassandra CQL

NoSQL Data Models

NoSQL Attributes:

- Not using the relational model
- Running well on clusters
- Open-source
- Built for the 21st century web estates
- Schemaless

KeyValue

- Redis
- Memcached
- GemFire (VMware/Pivotal)
- Apache Geode
- Riak
- BerkeleyDB
- Hazelcast
- LevelDB

Document

- AWS DynamoDB
- MongoDB
- CouchDB
- Couchbase
- MarkLogic
- OrientDB
- ArangoDB
- RavenDB

Column

- Cassandra
- AWS SimpleDB
- HBase
- Accumulo
- Hypertable

Graph

- Neo4J
- FlockDB
- Virtuoso
- Giraph
- Neptune
- Titan (can use DynamoDB as backend)
- OrientDB
- ArangoDB

AWS Solutions in orange

NoSQL Considerations

- NoSQL databases have no explicit schema
  - Schemaless databases have implicit schemas
  - Changing the structure of data during the life of an application has an impact on the application
- Many NoSQL databases operate on aggregates
  - An aggregate may be a document, column set, block, etc.
- NoSQL databases work best as application databases
  - Application databases serve a single application or application component
  - Application databases are often wrapped in SOA services
  - A strength of NoSQL solutions is their ability to map directly to application constructs
  - NoSQL databases do not always work well in multitenant (application) scenarios where more general solutions are more broadly applicable
  - Polyglot Persistence using different data stores in different circumstances
- Choosing the best NoSQL solution involves:
  - Selecting a programming model and choosing a well aligned data storage model
  - Selecting the simplest storage solution that provides the scale, performance and growth potential required

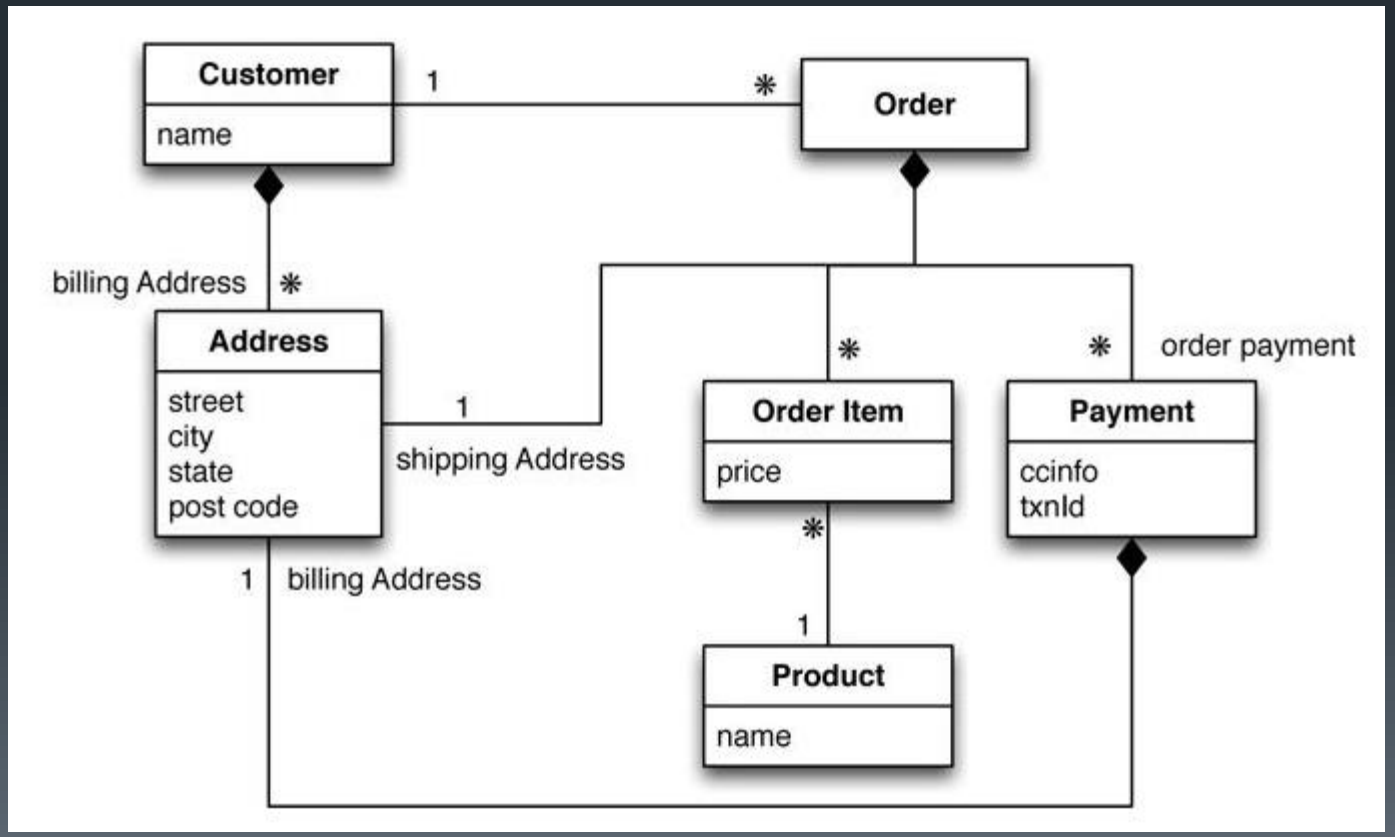
There are two primary reasons for considering NoSQL

- To handle data access with sizes and performance that demand a cluster
- To improve the productivity of application development by using a more convenient data interaction style

--Fowler 2012

Aggregates

- In Domain-Driven Design [Evans], an aggregate is a collection of related objects that we wish to treat as a unit
- Choosing aggregates is an important part of the design process in distributed applications
- Where are the aggregate boundaries?
- Which datum will be repeated if any?
- How can aggregates be structured to minimize server round trips?

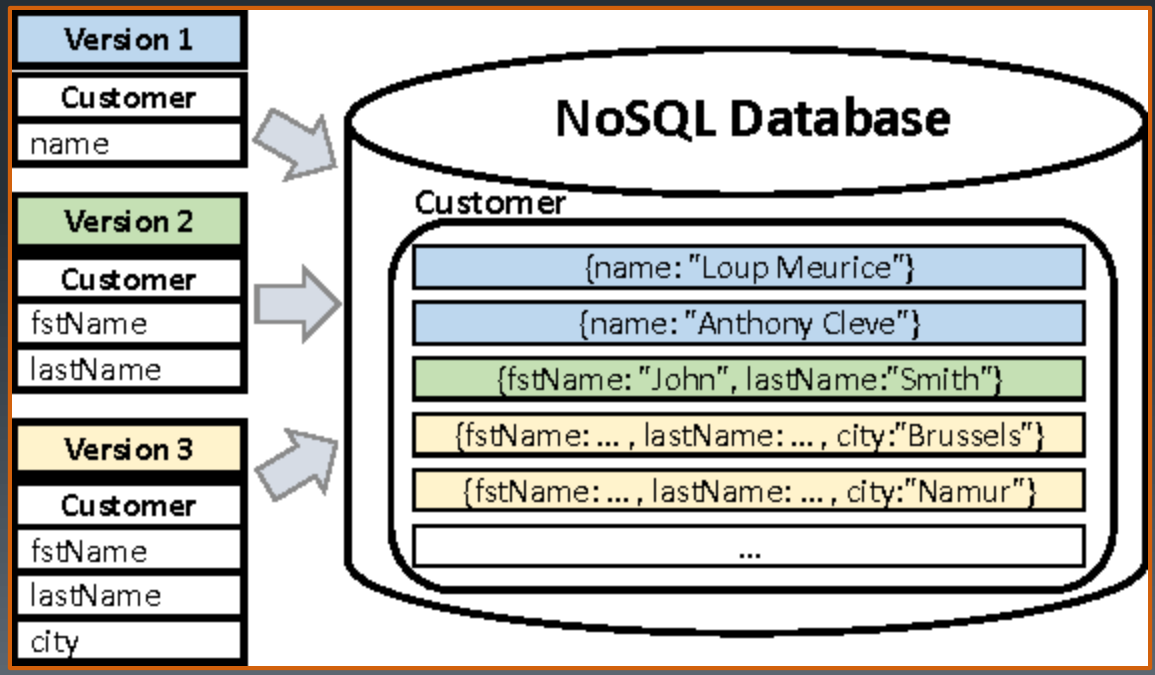


NoSQL Transactions

- ACID transactions offer the pinnacle of consistency
- Most NoSQL database offer no transaction support
  - Graph databases are the exception
- However, transaction-less NoSQL systems offer atomic aggregate updates
- If aggregates equate to joined rows in SQL similar transaction semantics can be achieved

Schemaless Systems

- NoSQL databases often have no explicit schema
- Programs that accesses data in such a system use an implicit schema
  - Schemaless databases shift the schema into the application code that accesses it
- If multiple applications access the same database the implicit schema must be coordinated
  - Error prone and difficult to scale/maintain
  - Microservice systems encapsulate all database interaction for a given aggregate within a single service
- Schemaless data stores allow data to evolve over time



Consistency

- Highly distributed clusters introduce new consistency concerns as compared to Relational ACID type systems
  - **write-write conflict**: two people updating the same data item at the same time
    - Can produce **Lost Updates**
    - We both increment 6 and instead of going to 8 it goes to 7
  - **sequential consistency:** ensuring that all nodes apply operations in the same order
- Systems can take a pessimistic or optimistic approach to consistency
  - pessimistic prevents conflicts from occurring
  - optimistic lets conflicts occur, but detects them and retries
- Conditional updates test the value just before updating it to see if it's changed since last read
- Some systems save all write-write conflict updates allowing users (or application code) to resolve the conflict
- Any update that affects multiple aggregates leaves an inconsistency window
  - Amazon SimpleDB reports an inconsistency windows usually less than 1 second
- Replication consistency is the process of ensuring that the same data item has the same value when read from different replicas
  - Quorums
  - Eventual Consistency
- read-your-writes consistency means that, once you've made an update, you're guaranteed to continue seeing that update

Sticky Session

- A session tied to one node (aka. session affinity)
- ensures that as long as you keep read-your-writes consistency on a node, you'll get it for sessions too
- sticky sessions reduce the ability of the load balancer to do its job

Quorums

- write quorum
  - W > N/2
  - The number of nodes participating in the write (W) must be more than the half the number of nodes involved in replication (N)
  - The number of replicas is often called the replication factor
- read quorum
  - How many nodes you need to contact to be sure you have the most up-to-date change
  - You can have a strongly consistent read if R + W > N
- These inequalities are written with a peer-to-peer distribution model in mind
- Most suggest a replication factor of 3
  - This allows a single node to fail while still maintaining quora for reads and writes
  - With automatic rebalancing the cluster will create a third replica quickly
- The number of nodes participating in an operation vary with the operation
  - When writing a quorum may be required for some types of updates but not others, depending on the value of consistency and availability

Key Value Model

- Each aggregate has a key or ID that's used to get the data
- The aggregate is opaque to the database
  - Usually seen as a blob
  - Aggregates can be anything
  - Usually size limited
- Some dbs support meta data to define relationships, expiration times, etc.
- These solutions are very simple, and thus tend to be very fast
- Sharding is based on key hash

Use:
- Storing Session Info
- User profiles/preferences
- Shopping carts

Don't Use:
- Data Relationships
- Multioperation Transactions
- Query by Data
- Operations on sets

Memcached

- A general-purpose distributed memory K/V caching system
- Used to speed up dynamic database/API driven websites
- Runs on Unix, Linux, Windows and Mac OS X
- Keys are up to 250 bytes long and values can be at most 1 megabyte
- The client library knows all servers
  - Servers do not communicate with each other
  - Clients read/set values by computing a hash of the key to determine the server to use
- When the table is full, subsequent inserts cause older data to be purged in least recently used (LRU) order
- Applications using Memcached typically fall back to a database request on cache miss
- Originally developed by Danga Interactive for LiveJournal, now used by YouTube, Reddit, Zynga, Facebook, Twitter, Tumblr, Wikipedia, etc.
- Engine Yard and Jelastic are using Memcached as the part of their platform as a service technology stack
- Heroku offers a managed Memcached service built on Couchbase Server as part of their platform as a service
- Google App Engine, AppScale, Windows Azure and Amazon Web Services also offer Memcached

```
function get_foo(int userid) {
   /* first try the cache */
   data = memcached_fetch("userrow:" + userid);
   if (!data) {
      /* not found : request database */
      data = db_select("SELECT * FROM users WHERE userid = ?", userid);
      /* then store in cache until next get */
      memcached_add("userrow:" + userid, data);
   }
   return data;
}
```

Document Model

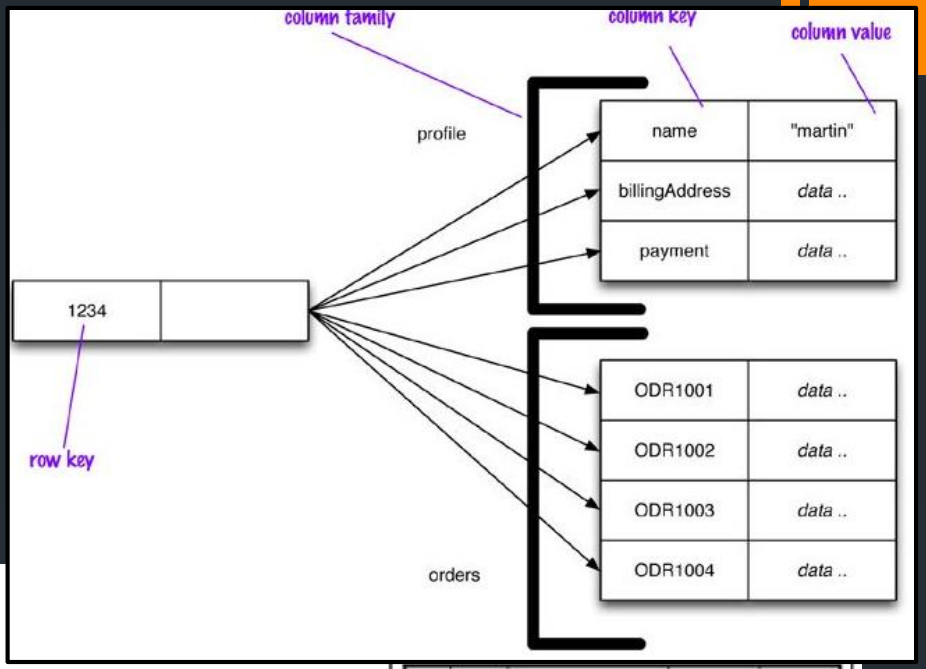
- Each aggregate has a key or ID that's used to get the data
- The aggregate is visible to the database
  - Usually a JSON or XML document
  - Aggregates can have any structure support by the document type
- Document dbs support limited searches on document data
- These solutions are simple, fast but supply useful aggregate search features
  - You can look up data by something other than aggregate key
- Sharding is based on key hash
- In practice the distinction between KeyValue and Document dbs blurs heavily in some implementations

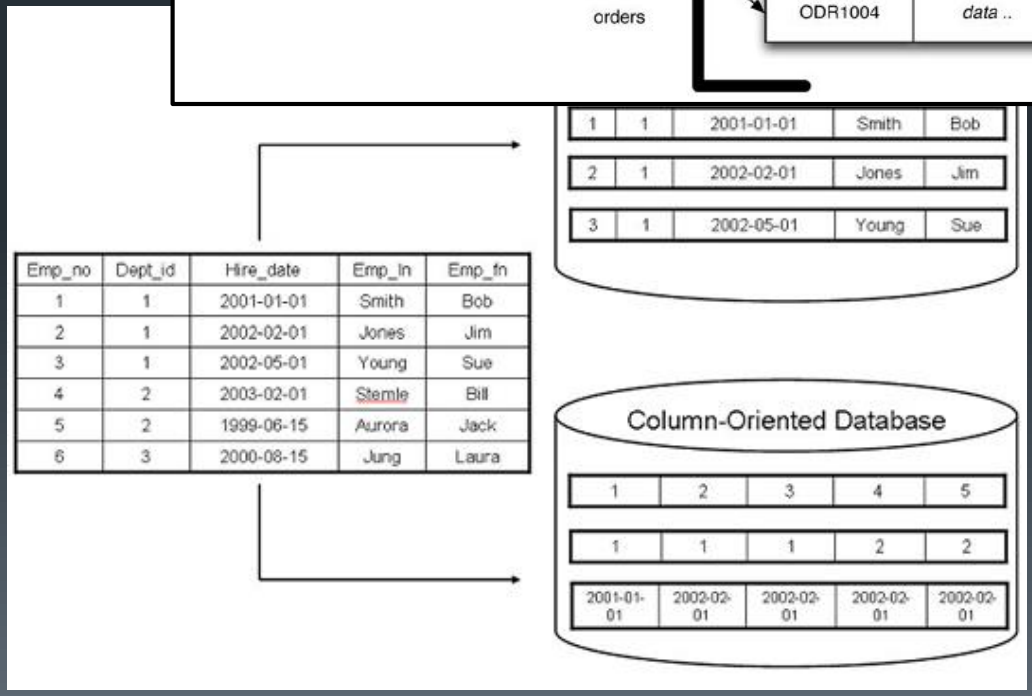
Use:
- Event Logging
- CMS/Blogging
- Web Analytics
- E-Commerce

Don't Use:
- Complex transactions
- Queries against varying aggregate structure

Column

- Google documented BigTable in a famous white paper [Chang etc.]
  - Their core data store at the time (previously used for search)
  - Ues sparse columns and no schema
  - A two-level map
  - Influenced HBase and Cassandra
- Column Families
  - Stores groups of columns together
  - Each column has to be part of a single column family
  - Columns are the unit for access
  - Column stores make summing columns very fast due to column layout on disc versus row layout
  - Column families can be treated like tables
- Columns can be added freely
- Skinny Rows
  - Few columns
  - Same columns used across the many rows
  - The column family defines a record type
  - Each row is a record, and each column is a field
- Wide Rows
  - Have many columns (perhaps thousands), with rows having very different columns
  - A wide column family models a list, with each column being one element in that list
  - A consequence of wide column families is that a column family may define a sort order for its columns





Use:
- Event Logging
- CMS/Blogging
- Counters
- Expiring usage (columns)

Don't Use:
- Situations where the column families change frequently

Cassandra Case Study

- Invented at Facebook by the author of Amazon's DynamoDB, Apache Cassandra is an open source distributed database management system designed to handle large amounts of data across many commodity servers
  - Cassandra's distributed architecture is specifically tailored for multiple-data center deployment, for redundancy, for failover and disaster recovery
- Cassandra is implemented as a DHT producing high availability with no single point of failure
- Cassandra implements asynchronous masterless replication
- University of Toronto researchers studying NoSQL systems report that Cassandra achieved the highest throughput for the maximum number of nodes in all of their experiments
- Based on the Chord DHT and uses an MD5 128 bit hash key
- Cassandra's data model is a partitioned row store with tunable consistency
  - Partition keys can be hashed randomly or in order for systems benefitting from partition key locality
  - Rows are organized into tables
  - The first component of a table's primary key is the partition key
  - Within a partition, rows are clustered by the remaining columns of the key
  - Other columns may be indexed separately from the primary key
  - Tables may be created, dropped, and altered at runtime without blocking updates and queries
- Cassandra does not support joins or subqueries, except for batch analysis via Hadoop
- Cassandra emphasizes denormalization through features like collections
- Memcached competitive performance without a two level architecture

Facebook statistics for a >50TB, 150 node cluster

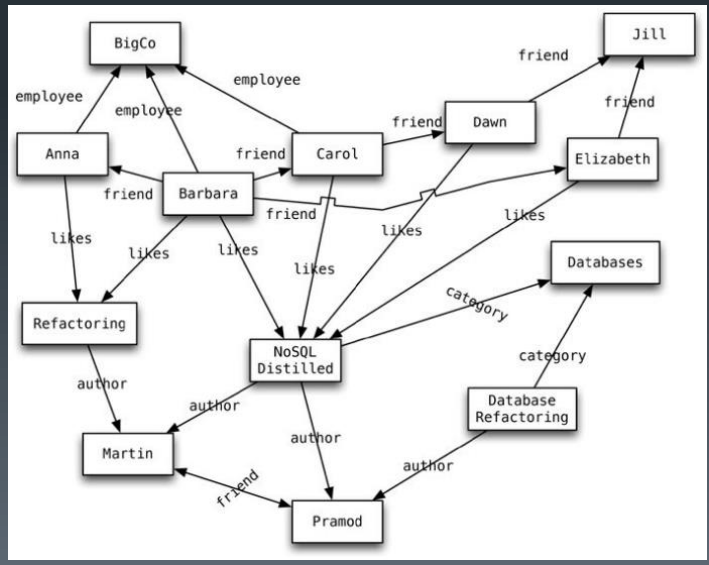
Latency Stat	Search Interactions	Term Search
Min	7.69ms	7.78ms
Median	15.69ms	18.27ms
Max	26.13ms	44.41ms

Graph

- Graph databases are motivated by a different frustration with relational databases and thus have an opposite model
  - small records with complex interconnections
- Nodes & Edges (aka. Arcs)
- Queries like:
  - Select books in the Databases category written by someone whom a friend of mine likes
- Edge traversal is cheap
  - Graph databases shift most of the work of navigating relationships from query time to insert time
  - Good only if querying performance is more important than insert speed
- Usually single server based
- Examples
  - FlockDB - nodes and edges with no mechanism for additional attributes
  - Neo4J - attach Java objects as properties to nodes and edges
  - Infinite Graph - stores Java objects, which are subclasses of its built-in types, as nodes and edges

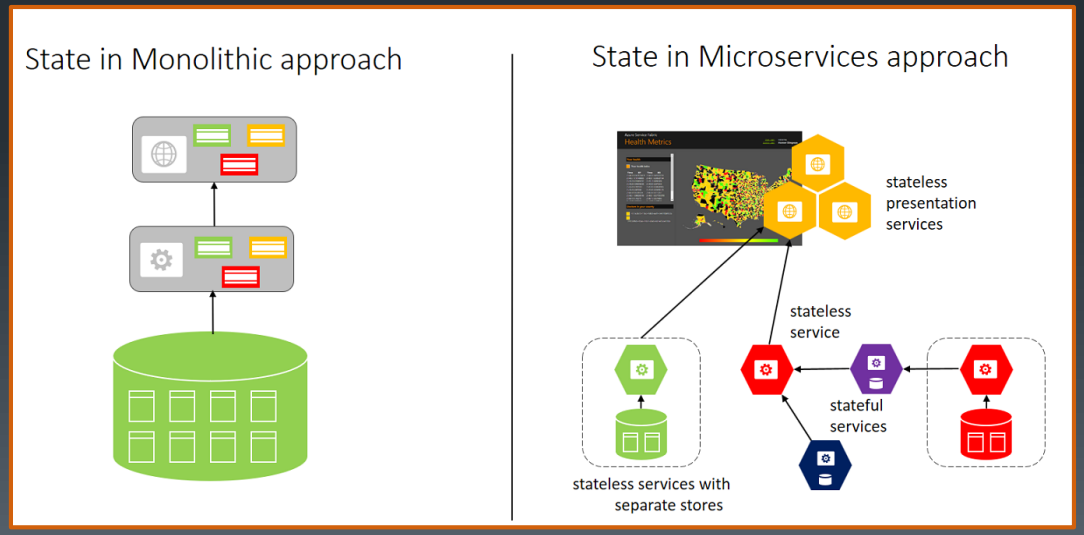
Use:
- Connected Data (social graphs)
- Routing/Dispatch (location based systems
- Recommendation Engines

Don't Use:
- Situations where multiple nodes must be updated



Cloud native state management, patterns and practices

- Cloud native applications divided microservices into 2 camps
  - Stateless
  - Stateful
- Stateful services use volumes to store durable data
- State managers are typically cloud native cluster aware systems
  - Redis, MongoDB, Cassandra, etc.
- State managers are owned by a single service and live inside the bounded context of that service



Summary

- A range of state managers are available for use in microservice based systems
  - Relational
  - Graph
  - Key/Value
  - Document
  - Column
- Schemas are critical components of most data stores and should be documented
- Aggregates describe the elements of storage in NoSQL systems
- A range of consistency tuning features are available with different storage solutions

Lab 6

- Building a stateful service