NoSQL “Interview Questions”

**Relational Database Questions**

Basic Operations

◦ Create – This is how you create new tables

◦ Insert – This is how you add data to tables

◦ Select – This is how you query data to get results

◦ Update – This is how you change existing data

◦ Join (Inner, Outer, Left, Right, etc.) – This is how you combine tables

◦ Union All/Union/ Union Distinct – used to combine the result-set of two or more SELECT statements. Returns distinct set across the two.

◦ Minus – This is how you make sure to not return certain data in queries

◦ Intersect – Returns distinct rows that are output by both the left and right input queries operator

◦ Normalization (1NF, 2NF and 3NF) – This is how you coerce your tables into better structure. Allows for cleaner queries and joins.

◦ Transaction(Concurrency Control) – ACID

Atomic – Transactions either fully complete or fully fail

Consistent – Transactions follow rules of the DB

Isolated – Other transactions don’t see partially complete transaction

Durable – Once a transaction is completed, it persists

◦ Indexing\* - B tree, Hash

Allows for faster queries by formatting the data by a column.

◦ Truncate vs Delete

Delete – Can be rolled back after

Truncate – Can not be rolled back

◦ Difference Where vs. Having

Where applies to a single row

Having applies to a grouped collection.

**NoSQL Questions**

* What is NoSQL?
  + Not only SQL. Does not follow the classic relational form that RDBMS have.
* Relational Database vs. NoSQL
  + NoSQL is more flexible, does not have a schema.
  + Ran on Clusters usually
  + Loses traditional ACID consistency.
  + Can better match needs
* Impedence mismatch
  + Data in-memory does not match the database structure.
* Polyglot persistence
  + Using multiple storage techniques to store the data.
* Aggregate-oriented database
  + Store commonly accessed together data on the same node.
  + Indexed by a key
* Key-value database
  + Store aggregates and index them by key to return value.
  + Can store anything in aggregates
* Document database
  + Like Key-Value but can also query internal structure of the document
* Column family database
  + Not all rows have to have the same columns.
  + Allows for good write performance.
  + Good for accessing many rows and not many columns.
  + First level is key/row identifier, second level is value/column identifier
  + Organized into column families
    - Store commonly accessed rows together in the same family.
* Graph database
  + Nodes (objects) and edges (relationships) representation of data.
  + Good for complex relationships.
  + Runs on a single server, not distributed
* Replication vs sharding
  + Replication – Copy the data to other nodes.
    - Master-Slave – One node takes all writes (and usually all reads) and then propagates it to slaves. If master goes down, automatically picks new slave to be master.
    - Peer-to-peer
      * All nodes take read/write requests and then data is propagated around.
        + Can result in inconsistencies (write-write)
      * Allows for good performance
  + Sharding – split the data into several nodes
    - Improves reads and writes
    - Can be combined with replication for more resilience
* CAP Theorem
  + Consistency – All nodes have up to date data
  + Availability – All running nodes can respond
  + Partition Tolerant – Can still respond even if communication between nodes breaks
* Eventual Consistency
  + It means eventual consistency…
* Map-Reduce
  + First generate “map” to all the key-value pairs of interest
  + Then “reduce” them down to one single key-value pair and return that

**MongoDB Questions**

* MongoDB’s type
  + Document Database
* MongoDB’s characteristics
  + Schemaless
  + Can query internal structure of documents
  + Aggregate
  + Easy scaling – does load balancing for you
  + CRUD
  + Indexing
* Alternative databases
  + DynamoDB
  + CouchDB
* Supported programming languages
  + C, C++, C#, Java, Node.js, Perl, PHP, Python, Ruby, Scala, Go and Erlang
* Index
  + Allows for faster querying. B-tree is default.
* Aggregation Operations(aggregation pipeline)
  + Can build complex aggregations from single pieces
  + i.e
    - $match – find criteria
    - $project – change or add field
    - Math, i.e. $add, $subtract, $multiply
    - Strings, i.e $concat, $substr
* Sharding
  + Partition the data across multiple nodes. All data only exists in one place
    - Same as sharding above
* Replication
  + Copy the data across multiple nodes. All data exists in multiple places
    - Same as replication above
* GridFS
  + Way to store and receive large documents.
  + Divides large documents into chunks and stores each as a separate document
  + Reassembles upon querying
* ObjectId
  + Unique identifier for each document
* Consistency
  + Read consistency by usually not allowing slaves to be read from
  + Write consistency usually by slaves only taking reads from primary master.
  + Both dependent on replica set size.

**Cassandra Question**

* Cassandra data model
  + Column Family.
  + Stores in two levels
    - Row/key – primary indicator
    - Column/value – secondary
  + Not all rows have to have the same columns
  + Group related data together as “column families”
  + Fast writes because can add columns to any row without affecting the other rows
  + Highly scalable and highly available
* Tunable consistency (Quorum)
  + Can set level of consistency by changing values for read and write quorum.
* Difference between RDBMS and Cassandra
  + No joins in Cassandra
  + No referential integrity in Cassandra
    - No primary-foreign key relationships
  + Query-First design
    - Design queries first, then structure DB around them.
* CAP Theorem
  + Consistency
  + Availability
  + Partition tolerant – This one must exist
* Cassandra query language
  + The method of making requests
  + Can only query primary key
    - Otherwise must either build secondary index – not preferred or create materialized views
* Cqlsh
  + The shell for Cassandra
* Compaction
  + Removes stale data from SSTable for better read performance.
  + Upon deletion, data is not immediately deleted, is given a “tombstone” which is a marker for future deletion.
    - This helps consistency as the tombstone is propagated to nodes that may be offline at the initial delete request.
* Super column
  + A map of columns.
  + Keeps related data together
  + Have to pull all columns even if you only need one.
* Column family
  + A grouping of rows that are commonly accessed together or have related data
* Memtable
  + Cassandra way of initially storing data in-memory. Once the Memtable gets too big, it flushes the data to the SSTable.
* Commit Log
  + Cassandra way of initially storing data. This is crash-recoverable.
* SSTable
  + Primary storage for data in Cassandra.
* Keyspace
  + The equivalent of a SQL Database. This is where the column families reside.