# NoSQL Document Databases

Concepts

# JSON, CSV, Relational Database

- Recall the differences between JSON and CSV when it comes to relational database tables
- CSV mimics a table
  - Easy to import
  - Easy to export
- Flat JSON is similar to CSV
- Nested JSON
  - Outermost layers mimic a table
  - Each nested layer mimics a separate table

# Several Tables of Sales Data

- Five tables
  - Stores, customers, products, sales, line items
- CSV
  - Five CSV files, one for each table
- JSON
  - One JSON file, level of nesting for each table

# Collection of JSON Objects

- Instead of having five relational tables
- We create a collection of JSON objects
- The JSON objects hold the same data as the five relational tables, but organized differently

# Querying Our Collection of JSON Objects

- We create indices on important key-value pairs in our JSON objects to speed up queries.
- We query using a SQL-like language.
- Queries return entire JSON objects instead of rows.

# NoSQL Document Databases

- Collection of documents
  - Originally XML was common
  - Then JSON was common
  - Now JSON-like is common
- Index key-value commonly used items in the documents
- Queries return entire documents instead of rows

# Normalization

- Normalized
  - 3NF (third normal form)
  - Each element of data is only stored once
  - Update data in only one place
- Denormalized
  - Each piece of data is stored multiple times for ease of use in analytics
  - Update data in multiple places
  - Data stored in multiple places must be in sync

# Transactional Databases

Transactional databases

- Executes the business
- OLTP (online transaction processing)
- Normalized, 3NF
- Scale up very limited

# Analytical Databases

Analytical databases

- Evaluates the execution of the business
- OLAP (online analytical processing)
- Denormalized
- Focus on both current and historical data
- Scales up very well

# POV: Points of Views

- We may want to analyze our data from several different POVs
  - Store, customer, product, quarter, month, etc.
- Create a separate document for each POV
- When we perform analytics, we choose the POV that best suits our needs
- Denormalize: multiple copies of the same data from a different POV

# Updating Data

- Denormalized
- Must update data in every POV
- Refresh frequency
  - Every X minutes
  - Once every X hours
  - Once a day
  - Once a week
  - Etc.

# The End

# NoSQL Document Databases

Business Cases

# Sales Relational Database

Tables

- Stores
- Customers
- Products
- Sales
- Line items

# Several POVs

- Load data into our NoSQL document database from the relational tables
- Several POVs
  - Store POV
  - Customer POV
  - Products POV
  - Day of week POV
  - Month POV
  - Quarter POV
  - Etc.

# Store POV

- Multiple ways to organize
- Stores
  - Customers
    - Sales
      - Line items
        - Products
- Stores
  - Month
    - Sales totals
- Etc.

# Customer POV

- Multiple ways to organize
- Customers
  - Products
    - Sales
- Customers
  - Sales
    - Products
- Etc.

# New Sales Data

Need to update all of the POVs that contain sales data

- Store POV
- Customer POV
- Products POV
- Day of week POV
- Month POV
- Quarter POV
- Etc.

# Airline POVs

- Flights POV
- Customers POV
- Aircraft POV
- Pilots POV
- Flight attendants POV

# University POVs

- Courses POV
- Student POV
- Instructor POV
- Department POV
- Semester POV

# Social Media POVs

- Followers POV
- Followed POV
- Subject matter POV
- Age grouping POV
- Geographic grouping POV
- Political POV

# Customer Website

- Customer logs into a website
- With relational database, we might have to pull from several database tables
- With a NoSQL document database, we could pull one document with all customer data in it
- Issues
  - Hard to keep data fresh in the documents
  - Transactional or analytical or hybrid?

# The End

# NoSQL Key-Value Databases

Concepts

# NoSQL Key-Value and NoSQL In-Memory

- NoSQL key-value concepts
- NoSQL in-memory concepts
- Perfect match
  - NoSQL key-value and NoSQL in-memory

# Python Dictionary

- Key with value
- Key
  - Scalar: integer, string, tuple, etc.
  - Must be unique
- Value
  - Scalar, list, dictionary, etc.
  - Nesting
- Key-value pair is often called a slot in slang

# Python Dictionary Update Logic

- If key does not exist:
  - Add new key-value pair
- If key exists:
  - Overwrite the key’s value

# NoSQL Key-Value

- Database structured in key-value pairs
- Keys must be unique
- Values
  - Black boxes
  - Anything
    - Binary (data, images, audio, video, etc.)
    - Text
    - JSON objects
    - Etc.

# NoSQL Update Logic

- If key does not exist:
  - Add new key-value pair
- If key exists:
  - Overwrite the key’s value
- Familiar? Same logic as Python dictionary update

# Exhaustive Search

- Queries by key
  - Expected method of querying
  - Extremely fast
- Queries not by key
  - Not expected
  - Exhaustive search of entire database—slow
  - Since data is black box, custom code to search

# Why Not Just Use Python Dictionaries?

- Python dictionaries are limited to one process on one OS
- Everything that needs to access the Python dictionary needs to be running on the same OS
- With NoSQL key-value databases, they run on a server where multiple processes on multiple OSs can access
- Scale up—NoSQL key-value database runs on a cluster of servers

# When Values Are JSON Objects

- Compare and contrast
  - NoSQL key-value where values are JSON objects
  - NoSQL document where documents are JSON-like objects
- Very similar
- The differences comes with the implementation
- Visit the topic of NoSQL in-memory

# Memory

- Memory
  - RAM (random access memory)
  - Fast
  - Expensive
  - Volatile (lost on shutdown, reboot, power loss, etc.)

# Storage

- Storage
  - Slower than memory
  - Cheaper than memory
  - Nonvolatile (not lost on shutdown, reboot, power loss, etc.)

# Storage (cont.)

- HDD (hard disk drive)
  - Slowest, cheapest, moving parts—energy, high failure
- SSD (solid state drive)
  - Faster than HDD, costs more than HDD, no moving parts—less energy
- Storage servers
  - NAS (network attached storage)
  - SAN (storage area network)
  - Object store

# Database Implementation

- Storage-based databases
  - Data is stored in a storage layer
  - Data is cached locally in memory to speed up access
  - On shutdown, reboot, power loss, data is not lost
- In-memory databases
  - Data is stored in-memory
  - Can be distributed among memory in several nodes of a cluster
  - Data is backed to storage
  - On shutdown, reboot, power loss, data that has not been written to storage layer is lost

# In-Memory Advantages/Disadvantages

- Advantages
  - Faster
  - So fast that indices usually do not help
- Disadvantages
  - Memory more expensive
  - Possible loss of data that is not backed to store

# NoSQL Key-Value and NoSQL In-Memory

- Common to combine key-value with in-memory
- Advantages
  - Fast
  - Non-key queries may be tolerated since the exhaustive search occurs in memory
- Disadvantages
  - Data has to fit in memory
- NoSQL key-value is generally assumed to be in-memory
  - We will assume this from now on in this course

# NoSQL Key-Value vs. NoSQL Document

- NoSQL key-value (assume in-memory)
  - Good for databases that can fit in memory (or memory budget)
  - Typical query is by key
  - Non-key queries are carefully considered
  - Need it as fast as possible
- NoSQL document
  - Good for databases that are too big for memory (or budget)
  - Databases that could fit in memory but have a lot of non-key queries

# The End

# NoSQL Key-Value Databases

Business Cases

# Web Server Session Data

- We are all familiar with web browser cookies
  - Client-side cookies
- Web servers also store session data
  - Server-side cookies
- Each web session gets a unique SID (session ID)
- For each web page (or web API) the web server has to pull session data based on SID and update it

# Web Server Session Data (cont.)

- In mid-1990s, companies originally used relational database tables to store session data
- In late 1990s, dot-com boom, companies soon found relational databases were not fast enough
- Companies started writing home brew in-memory key value stores by SID
- This is the origin of the key-value in-memory database

# Current Stock Quotes

- We want to store current stock quotes
  - Last sales price, bid, bid size, ask, ask size, open, previous close, volume, etc.
- Stock symbol is a unique key
- Few, if any, queries other than by stock symbol
- Few thousand stock symbols
  - Fits into memory

# Current Stock Quotes (cont.)

- Fast moving
  - Quotes stream fast
  - Fast updates
  - Fast queries

# Multiplayer Game Player Stats

- Multiplayer games have leader boards
- Players have a unique key
- Few million players
- Players are constantly earning points
- Need to update players' points, stats, etc. as quickly as possible
- Want to display players' stats as quickly as possible

# Live Weather Data

- We have weather sensors all over the world.
  - Temperature, precipitation, snow on ground, wind, pressure, humidity, etc.
- Each weather station has a unique key.
- There are about 100,000 weather stations around the world.
- We want to disseminate weather data from sensors quickly.

# The End

# SQL, NoSQL

Business Case Comparison

# Relational Databases Using SQL

- Default go-to database is the relational database using SQL
- Often due to lack of knowledge and understanding of NoSQL
  - Even some enterprise architects seem clueless
- Not a one-size-fits-all solution
- Limiting to relational database with SQL is sometimes like the proverbial trying to fit a square peg into a round hole

# Corporate Resources in Place

- Relational databases typically have a lot of corporate resources in place
  - Auditing
  - Backup
  - Encryption
  - Cataloging
  - Etc.
- SOR (system of record)

# NoSQL Instead of Relational Database With SQL

- Sometimes a relational database with SQL simply will not work at all
- Must use a NoSQL that will work
- NoSQL is our SOR
  - Challenge to make sure we have all the auditing, backup, encryption, cataloging, etc. in place

# NoSQL Supplementing Relational Database With SQL

- Sometimes a relational database can store our data, just not in the most convenient format for analytics
- Store data in relational database as our SOR
  - Leverage corporate resources
- Store data in NoSQL as a supplemental database for analytical purposes

# Sales

- Sales tables in relational database as SOR
  - Stores, customers, products, sales, line items
- For analytical purposes, create a NoSQL document database with documents from several POVs

# Weather

- Need current weather, time series weather, and permanent official record
- Relational database for official record as SOR
  - Readings every 15 minutes
- NoSQL time series to hold streaming data that is constantly coming in
  - Fast enough for use to store it as it comes in and special purpose for time series
- NoSQL key-value for current (as we detailed previously)

# Huge Table-Oriented Dataset

- Huge dataset that naturally fits into relational tables
- Too big for our transactional relational databases to hold and process
- NoSQL wide column
  - Scales up
  - SQL on top of NoSQL
- Biggest issue is that NoSQL is now SOR
  - Have to build out corporate quality resources ourselves

# Graph Data

- In our NoSQL graph database business cases, we saw numerous cases where data is most naturally represented as a graph
- Graphy features easily store in relational database tables
  - Nodes, relationships, labels, attributes, weights, etc.
- Analytics on graphy features not easy from tables
- Use relational database as SOR and use NoSQL graph database for analytics

# Web Server Session Data

- As detailed earlier, web servers would use a NoSQL key-value in-memory solution
- When a user’s session is completed, we may want to save the session data to relational tables
- Data in NoSQL would be lost while data in relational tables would be SOR

# Financial Transactions

- We have financial transactions that need both security and speed
  - Security and speed are conflicting requirements
- Block chain aspects of NoSQL ledger database would meet requirements
- In this case, NoSQL is not as convenient
  - Analytics typically do not care about hashes
- Relational database is more convenient for analytics
  - If too big, go to NoSQL wide column

# The End