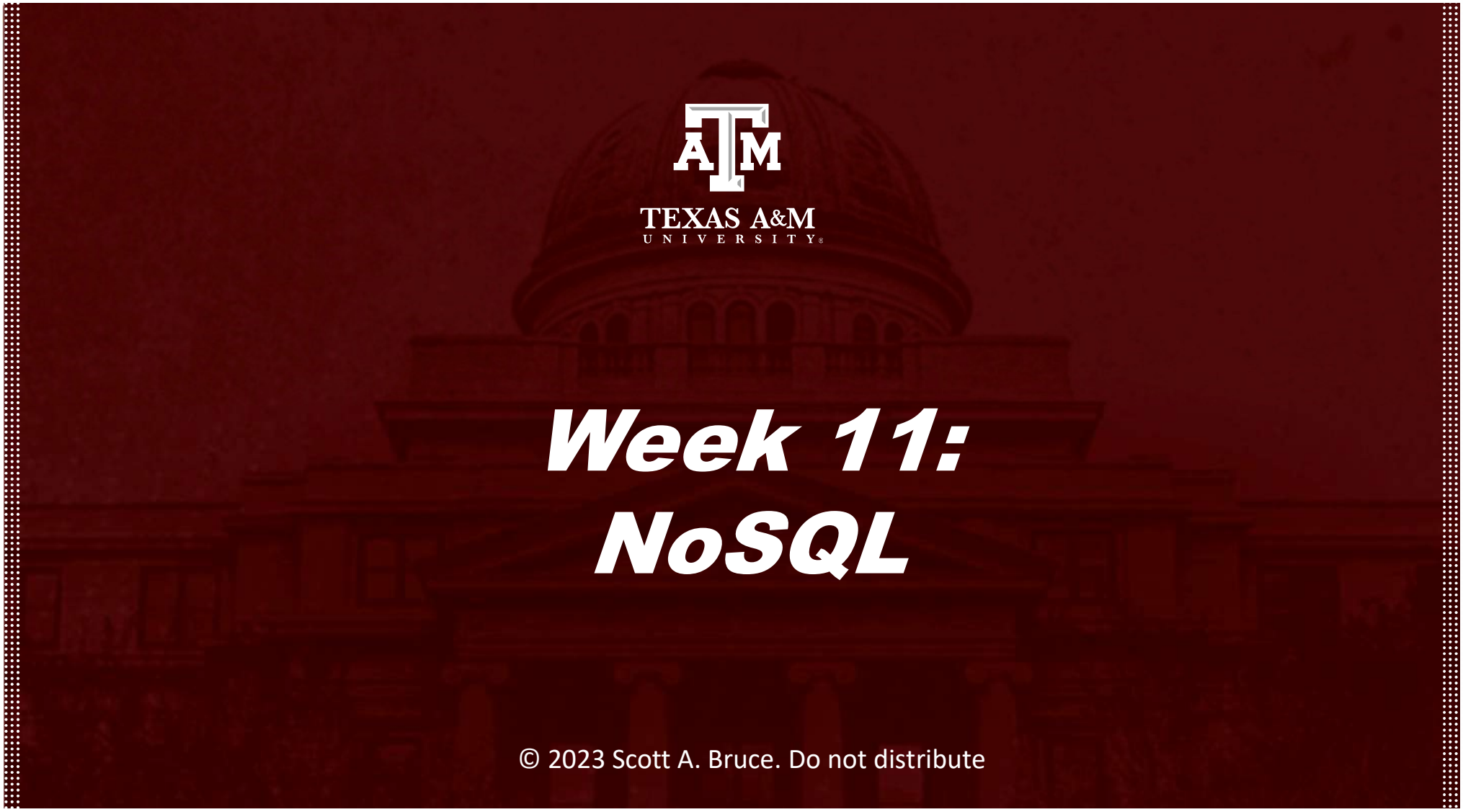


Week 11: NoSQL

Source

1. This module contains material from the LinkedIn Learning courses

- NoSQL Essential Training by Melanie McGee
- Data Science Foundations: Choosing the Right Database by Kumaran Ponnambalam

(https://linkedinlearning.tamu.edu/)

2. Textbook
NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence by P. Sadalage, M Fowler
>> Chapter 8-12

Motivation: NoSQL

- It often stores data in a denormalized format, which incurs extra storage costs due to redundancy but offers faster query speeds.
- NoSQL databases aren't restricted to a rows-and-columns approach.
- A distributed system (partitioning or sharding): data is stored across multiple servers enhancing accessibility

Why NoSQL?

Data structure becomes more complex

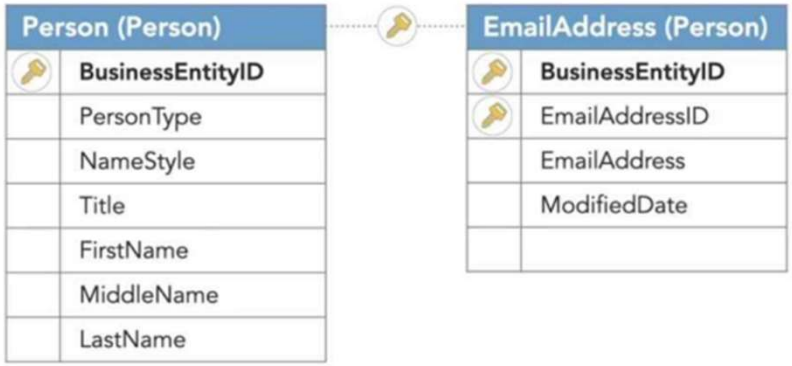
- Dynamic changing relations such as tree diagram or complex data formation such as tweets largely stored in ‘unstructured’ text
- Painful to store big data in a structured schema like in RDBMS Sparse data problem: null is available in RDMS (negatively affects to storage size)
- Do we need ‘relationship’ if all stored in one row?

Relational vs. NoSQL

TEXAS A&M UNIVERSITY

Relational DBs

- Structured
- Reliable and consistent
- Store data in 'tables' with columns and rows
- Relationships are **explicit** and managed by the RDMBS (it's not the programmer's job. RDMBS will enforce it)
- Well-suited for most business transactions



Relational Database Person Entity

BusinessEntityID	PersonType	NameStyle	Title	FirstName
1	EM	0	NULL	Ken
2	EM	0	Ms.	Terri
3	EM	0	Mr.	Roberto

Relational Database EmailAddress Entity

BusinessEntityID	EmailAddressID	EmailAddress
1	1	ken0@adventure-works.com
2	2	terri0@adventure-works.com
3	3	roberto0@adventure-works.com
3	**4**	**roberto.the.great@gmail.com**

Relational vs. NoSQL

TEXAS A&M UNIVERSITY

NoSQL DBs (a.k.a. non-relational, "not only SQL")

- Flexible and fluid design (semi-Structured or schema-agnostic)
- Columns and data types may change from one record to the next
- Scalable (accommodate many concurrent users interacting with DB)
- No explicit RDBMS enforced relationships
- Suitable for big data analysis, social networking, website personalization, etc.

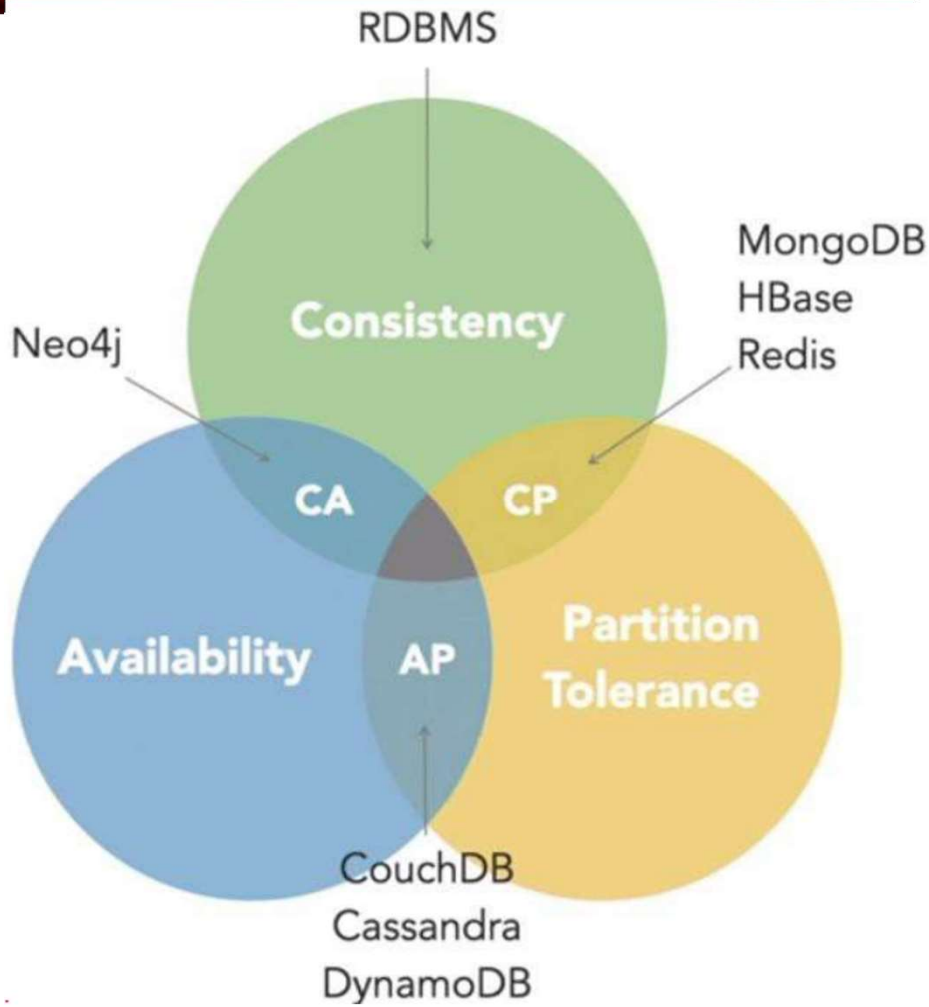
CAP Theorem for Distributed Databases

TEXAS A&M UNIVERSITY

How to compare SQL and NoSQL DBs?*

- Idea: You can't have it all, so pick 2.
- **Consistency** – data throughout the system is the same
- **Availability** – all working nodes in the distributed system return a valid response for any request, without exception
- **Partition Tolerance**
  - A **partition** is a communications break within a distributed system—a lost or temporarily delayed connection between two nodes.
  - **Partition tolerance** means that the cluster must continue to work despite any number of communication breakdowns between nodes in the system.

*Source: https://www.ibm.com/cloud/learn/cap-theorem

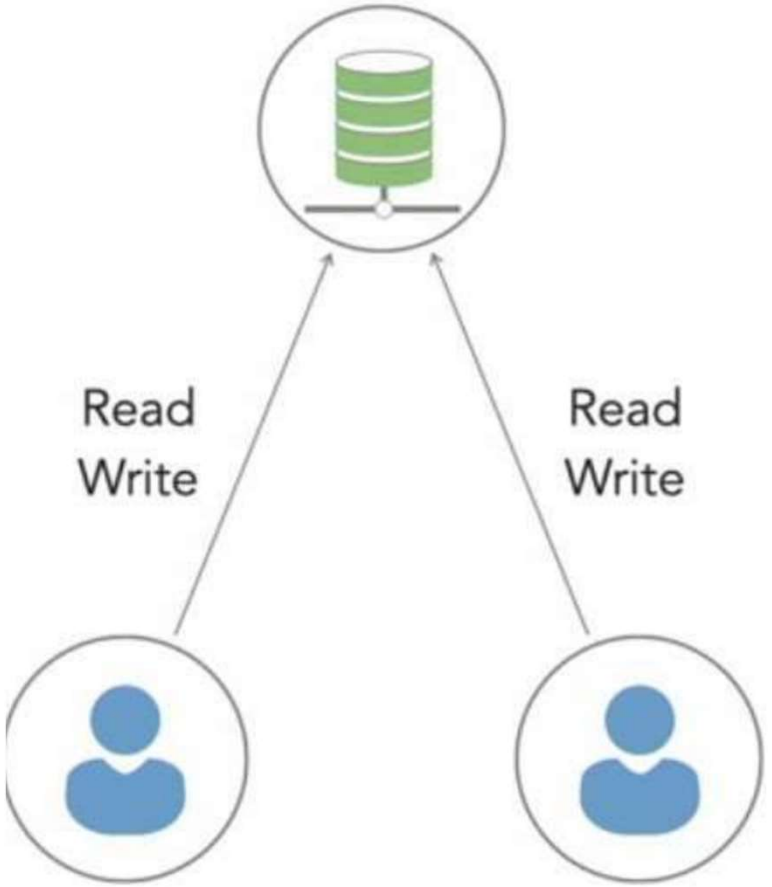


CAP Theorem for Distributed Databases

TEXAS A&M UNIVERSITY

Relational DBs

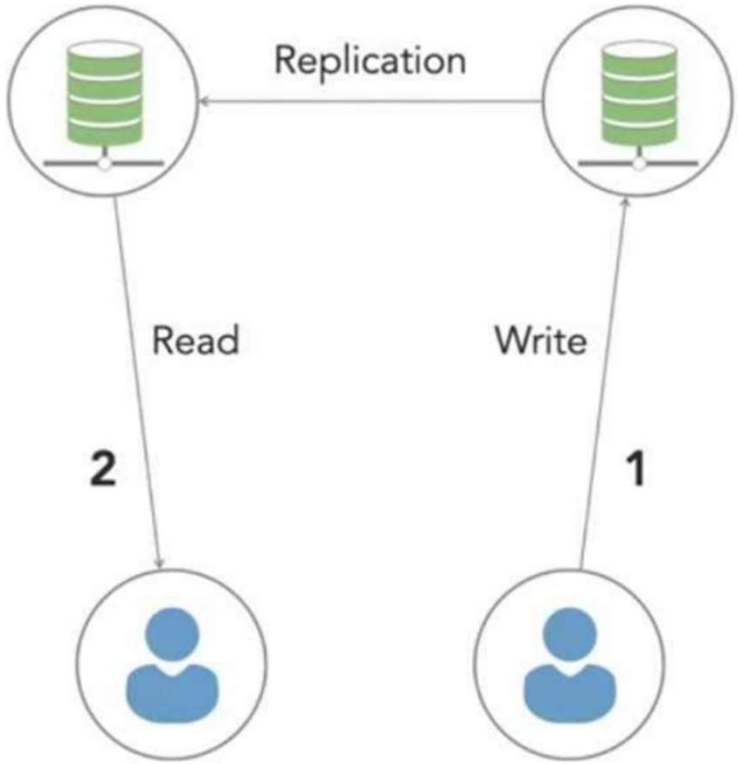
- **Extreme consistency** - every read receives the most recent write or an error.
- Unable to access data during updates (ACID compliance by locking data during transaction), so **availability also suffers**.
- **Transaction**: modifying data over multi-table with multi-command transactions available; you can decide if you want to keep the change or not by using commit/rollback; strong ACID compliance.
- **Lack partition tolerance** - data typically exist on a single machine.
- **No Replication**



CAP Theorem for Distributed Databases

NoSQL AP Systems

- **Eventual** consistency – most accurate data makes it to all machines after replication.
- **Highly available** (not locked during updates).
- **Partition tolerant** (can still function if nodes can't communicate). When a partition occurs, all nodes remain available but those at the wrong end of a partition might return an older version of data than others. (When the partition is resolved, AP databases typically resync the nodes to repair all inconsistencies in the system.*)
- **Multi-Master** Replication;
- Examples: CouchDB, **Cassandra**, DynamoDB.



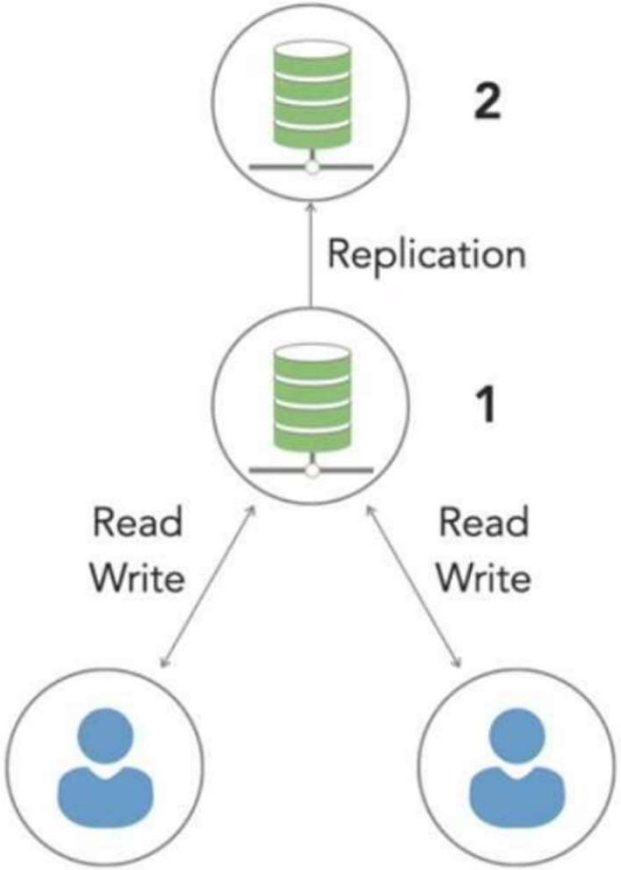
CAP Theorem for Distributed Databases

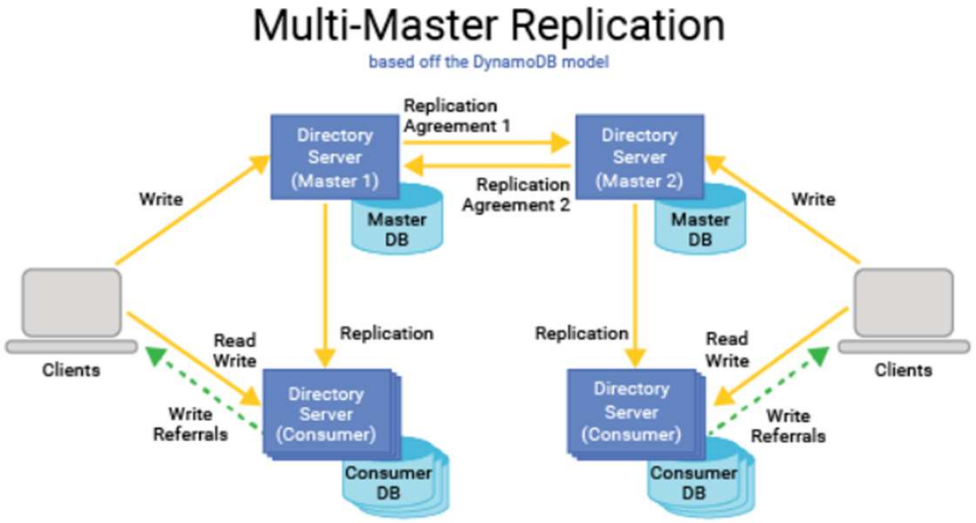
TEXAS A&M UNIVERSITY

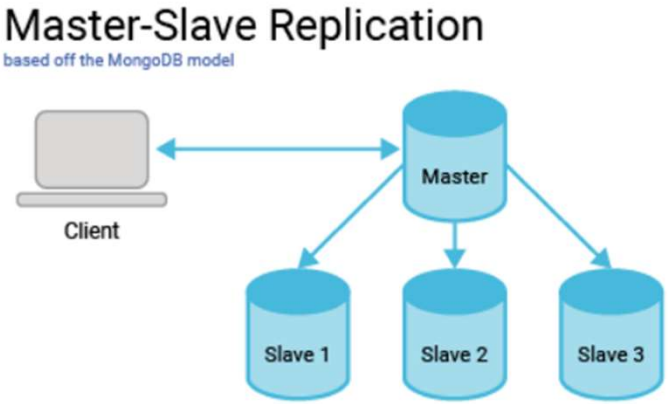
NoSQL CP Systems

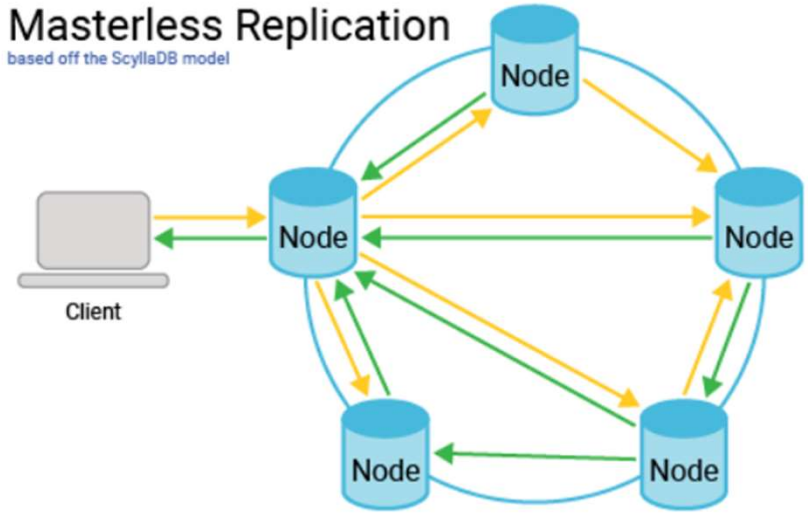
- **Excellent consistency** – reads and writes on machine where most consistent and accurate data exist (replicated to other machines after updates).
- **Not highly available** (locked during updates)
- **Partition tolerant** (can still function if nodes can't communicate). When a partition occurs between any two nodes, the system has to shut down the non- consistent node (i.e., make it unavailable) until the partition is resolved.*
- **Master-Slave** Replication
- Examples: **MongoDB,** HBase, **Redis**, relational DBs with failover

*Source: https://www.ibm.com/cloud/learn/cap-theorem









Source: https://www.scylladb.com/glossary/database-replication/

CAP Theorem for Distributed Databases

TEXAS A&M UNIVERSITY

NoSQL CA Systems*

- Delivers **consistency** and **availability** across all nodes.
- It can't do this if there is a partition between any two nodes in the system, however, and therefore can't deliver fault tolerance.
- We listed this type last for a reason—in a distributed system, partitions can't be avoided. So, while we can discuss a CA distributed database in theory, for all practical purposes, a CA distributed database can't exist.
- However, this doesn't mean you can't have a CA database for your distributed application if you need one. Many relational databases, such as PostgreSQL, deliver consistency and availability to some extent and can be deployed to multiple nodes using replication.
- neo4j: **constrained distribution** due to the complexity of graph data; **Causal Consistency** https://neo4j.com/docs/operationsmanual/current/clustering/introduction

*Source: https://www.ibm.com/cloud/learn/cap-theorem

Consistency

When is **consistency needed**?

- Inventory – customer can't purchase an item we don't have to sell.
- Financial accounting and transactions – account balance accuracy.
- Customer service – accurate customer data when reaching out.

When can consistency be desirable but **not required**?

- Batch data analysis – good to have recent data, but analysis not likely to be significantly impacted if missing latest updates.
- Search queries – availability more important than consistency. Display what's available!
- Fraud mitigation – fraud detection system should be up and running even if data are not perfect across all nodes.

Real world examples: Understanding CAP Theorem: Balancing Consistency, Availability, and Partition Tolerance in Distributed Systems – FactorBytes

Main Types of NoSQL databases

TEXAS A&M UNIVERSITY

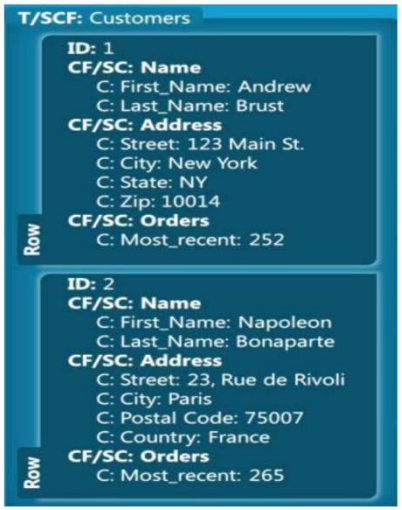
1. Key-value:

Key	Value
as12B5	10001
Emp-10023-Name	Mike Smith
Emp-10023-Qual	['BS', 'MS']
Cart-01999282	['Keyboard']

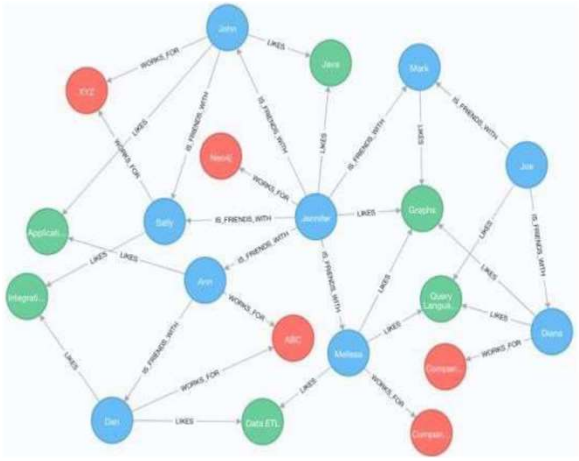
2. Document:

```
{
  "name":"Mike",
   "age":40,
    "married":"yes",
   "contacts" : {
     "phone": 1234556,
     "email": "m1@gm.com"
   }
}
```

3. Wide Column:



4. Graph



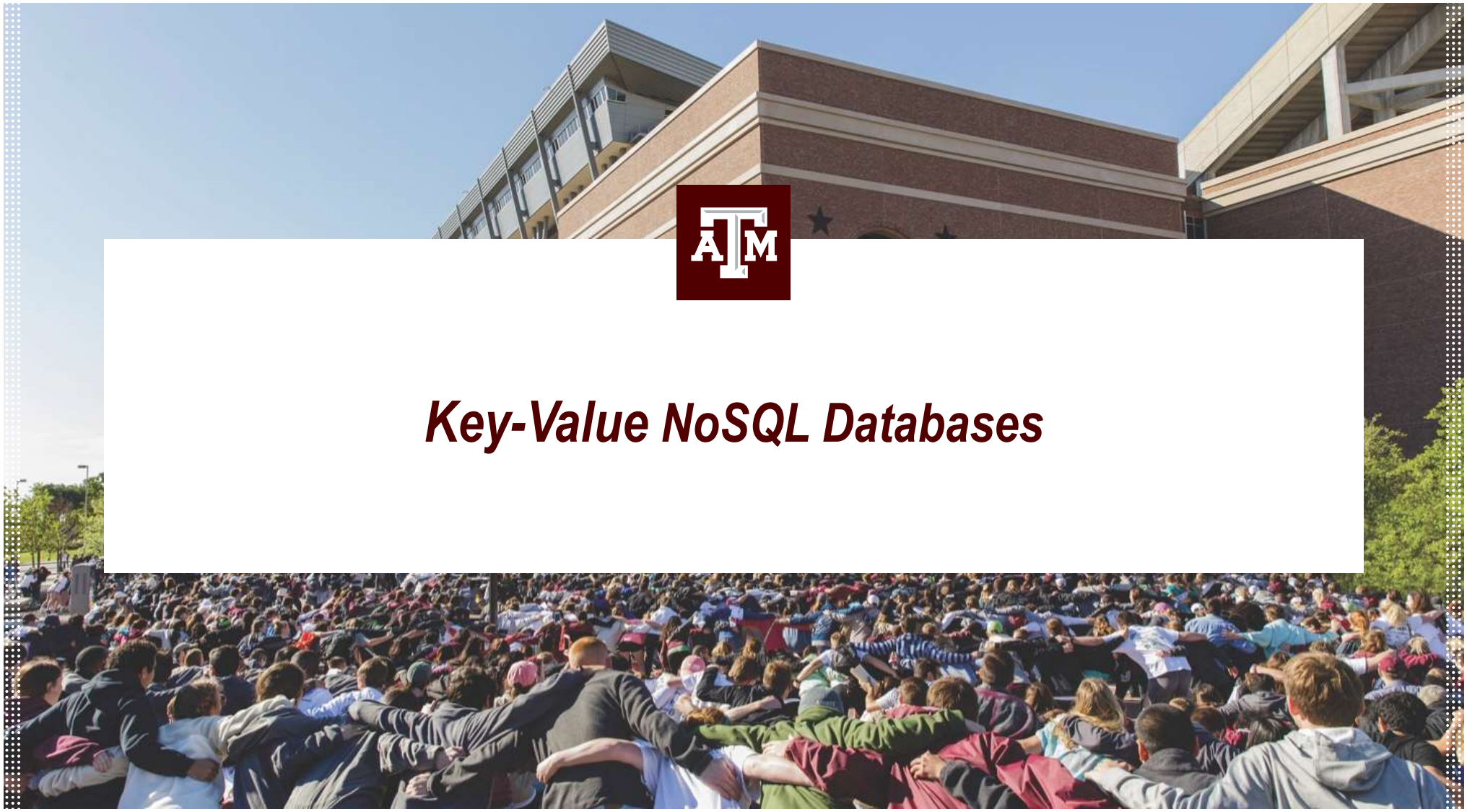
Main Types of NoSQL databases

TEXAS A&M UNIVERSITY

1. **Key-value:** The simplest type of NoSQL DB; A unique key is paired with a value
2. **Document:** A general-purpose database storing data in flexible documents, typically in JSON, BSON, or XML formats.
3. **Wide-column:** (column-oriented): Stores and reads data in rows but organized into columns. Column names and formatting can vary from row to row in a single table.
4. **Graph:** Stores data as nodes and relationship (edges) between nodes.

We will explore each in some level of detail in what follows.

Read https://cloud.google.com/discover/what-is-nosql



Key-Value NoSQL Databases

Key-Value NoSQL Databases

TEXAS A&M UNIVERSITY

Key characteristics

- Examples: **Redis**, Amazon DynamoDB
- Data has a **key** and **value;** They are **schema-free:** Every record can have different keys
- All key-value stores can **query by the key:**

→ What if we don't know the key? Lots of thought has to be given to the design of the key.

- Customable key expiration
- Key-value databases don't care what is stored in the value part of the key-value pair. It can be a blob, text, JSON, XML, and so on

→ It supports **complex data type**, which allow storing related data **under one key** but in multiple fields. Memory caching of data for **fast access** to stored values. https://redis.io/learn/howtos/solutions/microservices/caching

Suitable Use Cases

All relevant data about a subject is saved as the value for a single key: storing (via a PUT or SET command) or retrieving (via a GET command) the data for that subject becomes as simple as a single command referencing that key.

Storing Session Information: SessionID and Session details: This single-request operation makes it very fast, as everything about the session is stored in a single object.

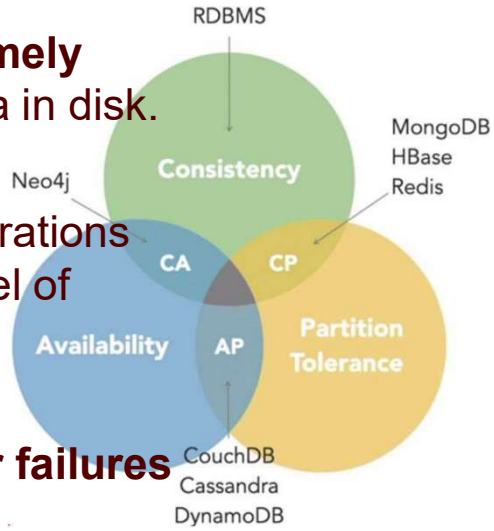
User Profiles, Preferences: UserID and User Profiles including username, or some other attribute, as well as preferences such as language, color, timezone, which products the user has access to, and so on.

E-commerce websites Shopping Cart Data: All the shopping cart information can be put into the value where the key is the userid and available across machines, sessions, etc.

Key-Value NoSQL Databases
Redis

TEXAS A&M UNIVERSITY

Redis stores all data in memory rather than on disk, making data access **extremely fast**, optimal for real-time use: other DBs (mongodb, Cassandra, etc.) store data in disk.



Consistency

Consistency is applicable only for operations on **a single key**, since these operations are either a get, put, or delete on a single key. The ***eventually consistent*** model of consistency is implemented

Transactions

Redis allows for **multi-operation transactions**, but does **not offer rollback for failures** occurring after the transaction starts executing.

Scaling

Typically **horizontal sharding** is implemented

→ Data is splitted by rows or keys and distributes these subsets across multiple servers

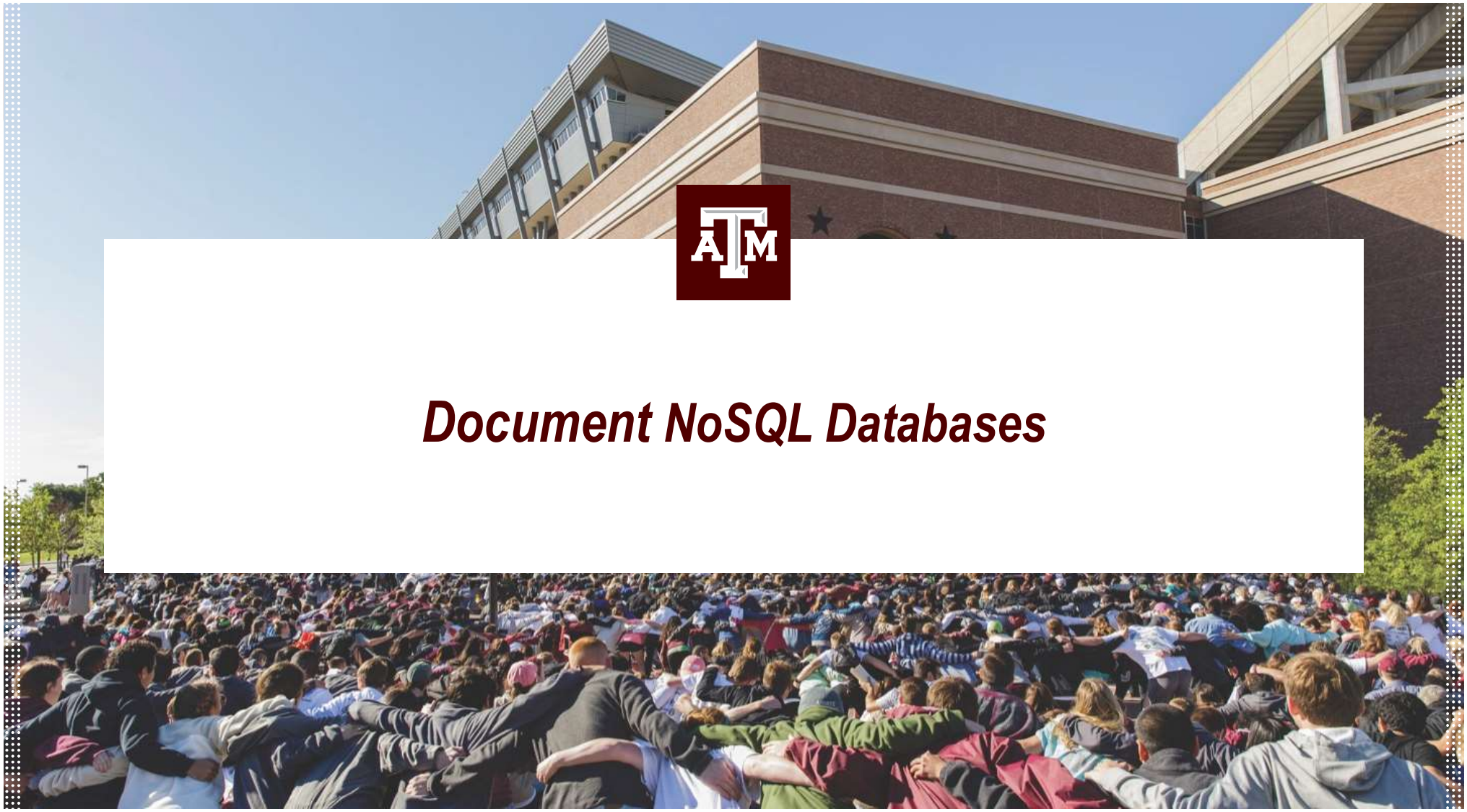
→ Sharding also introduces some problems: If a node storing a certain range of keys goes down, the data on that node can become unavailable.

Key-Value NoSQL Databases
Redis

Try It Yourself

- Redis Sandbox: https://redis.io/try/sandbox
- Get a cloud Redis: Getting Started
- Caching on API: https://github.com/haseungyeon3/STAT624_redis

Data types: https://redis.io/docs/latest/develop/data-types/



Document NoSQL Databases

Document NoSQL Databases

TEXAS A&M UNIVERSITY

Key characteristics

- Examples: **MongoDB**, Elasticsearch, CouchDB
- Most popular NoSQL database type.
- Treats records as “documents” stored in JSON like documents (e.g.,XML, JSON, BSON) which are hierarchical tree data structures.
- **Dynamic schemas**

→ Documents can have different attributes and data types:
The documents with different schema can still belong to the same collection

→ In documents, there are NO empty attributes:
In RDBMS every row in a table follows the same schema and if it does not have data it is marked as empty or set to null.

```
{
"firstname": "Martin",
  "likes": [ "Biking",   "Photography" ],
  "lastcity": "Boston",
  "lastVisited":
}
{
  "firstname": "Pramod",
  "citiesvisited": [ "Chicago", "London",
"Pune", "Bangalore" ],
  "lastcity": "Chicago"
}
```

Document NoSQL Databases

TEXAS A&M UNIVERSITY

Suitable Use Cases

Since document databases have **no predefined schemas** and usually understand JSON documents, they work well as a central data store for event storage

Content Management Systems, Blogging Platforms: (e.g., applications for publishing websites, managing user comments, user registrations, profiles, web-facing documents.)

Web Analytics or Real-Time Analytics: Document databases can store data for real-time analytics; it's very easy to store page views or unique visitors, and new metrics can be easily added without schema changes.

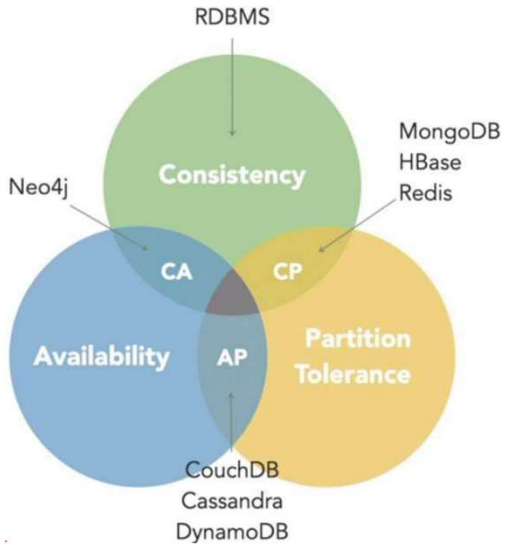
E-Commerce Applications: A flexible schema for products and orders, as well as the ability to evolve their data models without expensive database refactoring or data migration

Document NoSQL Databases MongoDB

TEXAS A&M UNIVERSITY

Consistency

Consistency is configured by using the **replica sets** and choosing to wait for the writes to be replicated/synced to all the slaves or a given number of slaves. (WriteConcern)



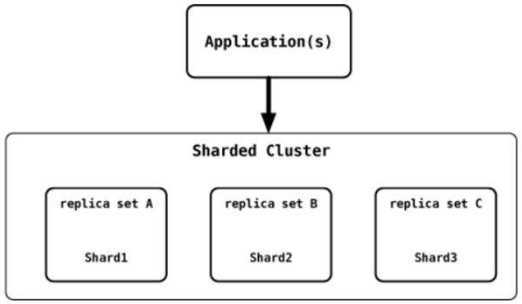
Transactions

Tranditional NoSQL does not support multi-operation transactions and roll back. MongoDB (since version 4.0) does support **multi-operation transactions** and **rollback features**, making it possible to revert grouped changes if any part fails.

Scaling/Availability

MongoDB's primary scaling strategies are sharding (horizontal) for growing performance and availability and replica sets for reliability, while vertical upgrades are used only when horizontal scaling is not feasible or needed initially.

https://www.mongodb.com/docs/manual/core/sharding-scaling-strategies/



Document NoSQL Databases
MongoDB

TEXAS A&M UNIVERSITY

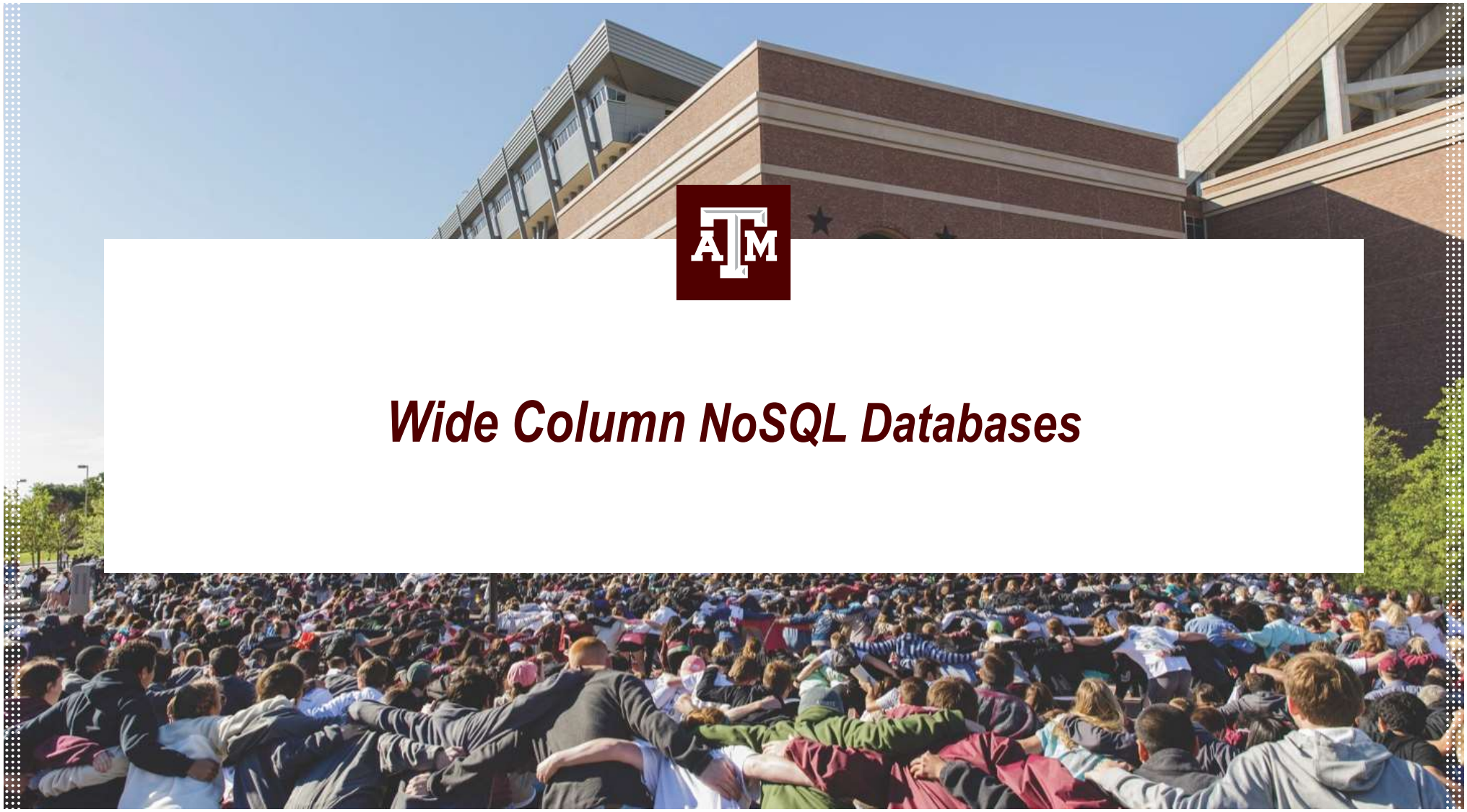
1. Get a cloud mongodb: MongoDB Atlas Tutorial | MongoDB

→ Connect to shell:

mongosh "mongodb+srv://cluster0.72xcx.mongodb.net/" --apiVersion 1 --username <db_username>

→ Connect to Python: Pymongo (Canvas>>Week 11)

2. Sharding simulation with Docker (Canvas>>Week 11)



Wide Column NoSQL Databases

Wide Column NoSQL Databases

TEXAS A&M UNIVERSITY

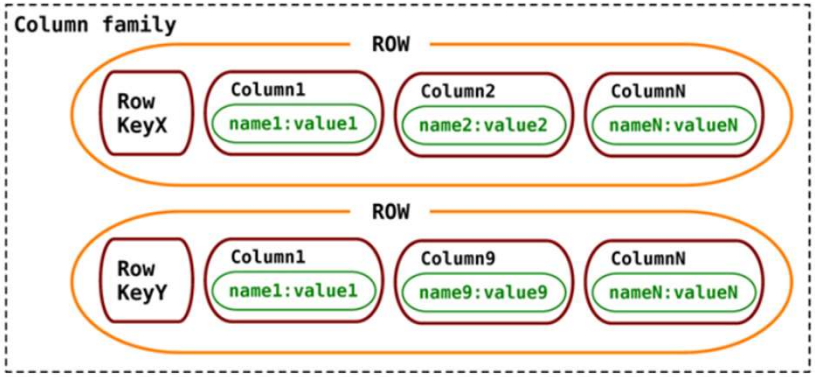
Key characteristics

• RDBMS store data row-by-row (each row contains several columns stored together), while **wide column DBs** store data **column-by-column (**Each row contains columns organized as key-value pairs)**:**

→ A row is a **collection of columns** linked to a key (row key- a unique key to identify the specific row); a collection of similar rows makes a **column family**. Column families are groups of related data that is often accessed together.

• Examples: **Cassandra**, Hbase

• Consist of tables, rows, and columns, but **columns can vary** (column families): columns can be added to any row at any time without having to add it to other rows.




```
//column family
{
//row
 "pramod-sadalage" : {
    firstName: "Pramod",
    lastName: "Sadalage",
    email:{'alice@example.com',
'a.work@example.com'},
    lastVisit: "2012/12/12"
 }
//row
 "martin-fowler" : {
    firstName: "Martin",
    lastName: "Fowler",
    location: "Boston"
 }
}
```

Wide Column NoSQL Databases

TEXAS A&M UNIVERSITY

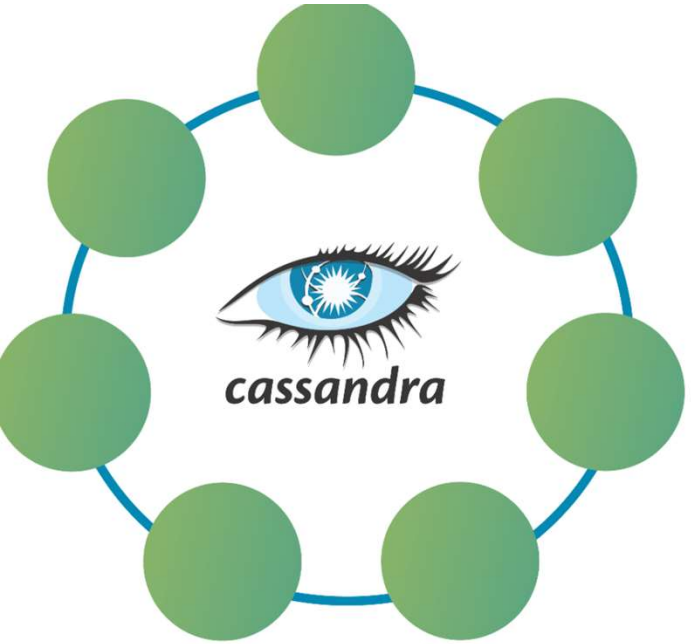
Key characteristics

- When modeling, focus on required queries, not data

→ Design your tables to support queries that quickly retrieve only the exact subset of data required rather than fetching whole rows

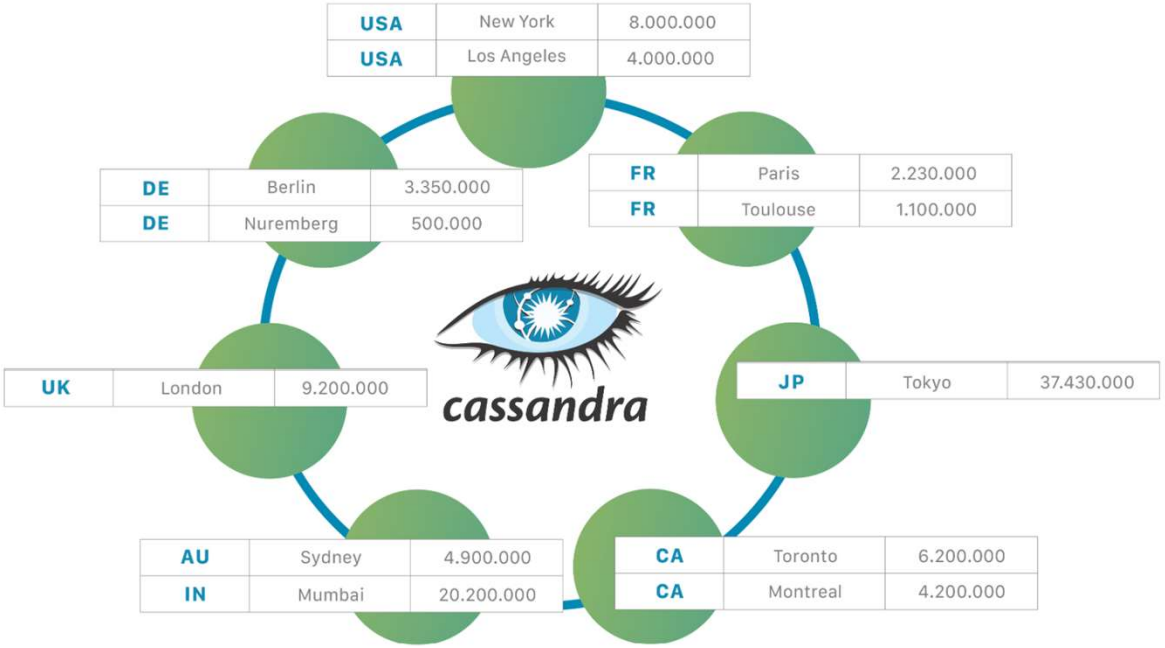
→ primary key (partition key and clustering columns) design: Data in each row is sorted by column names. Queries that retrieve a specific *column* repeatedly benefit from having that column **as part of the row key**, enabling fast lookups.

- Optimally stores large volumes of data. Empty column data not stored.
- Similar query language to SQL (see CQL)
- CQL does not allow joins or subqueries, and its where clauses are typically simple.



COUNTRY	CITY	POPULATION
USA	New York	8.000.000
USA	Los Angeles	4.000.000
FR	Paris	2.230.000
DE	Berlin	3.350.000
UK	London	9.200.000
AU	Sydney	4.900.000
DE	Nuremberg	500.000
CA	Toronto	6.200.000
CA	Montreal	4.200.000
FR	Toulouse	1.100.000
JP	Tokyo	37.430.000
IN	Mumbai	20.200.000

Partition Key



https://cassandra.apache.org/_/cassandra-basics.html

Scaling an existing Cassandra cluster is a matter of adding more nodes.

Wide Column NoSQL Databases
Cassandra

TEXAS A&M UNIVERSITY

- Fast,easily scalable: **NO master**-any read and write can be handled by **any** node
- **Cassandra** puts the column families into **keyspaces (**like a database in RDBMS)

Consistency

- Highly depends on the replication factor (RF): it replicates data across multiple nodes based on RF.

When a write/read request is received, it contacts some or all nodes depending on the configured RF

- When a write is received by a node, the data is first recorded in a commit log, then written to an in-memory structure known as **memtable:**

Nodetool: https://cassandra.apache.org/doc/latest/cassandra/troubleshooting/use_nodetool.html

Transactions

NO traditional ACID transactions with full rollback, commit, or locking mechanisms.

Atomic write at the **row level**: inserting or updating columns for a given row key will be treated as a single write and will either succeed or fail.

Availability

Highly available (no master in the cluster). The availability increased by reducing the consistency level of the requests.

Wide Column NoSQL Databases
Cassandra

Suitable Use Cases

Event Logging: Store application events using a row key formatted as appname:timestamp.

Content Management Systems & Blogging Platforms: Use Cassandra to store blog entries with metadata such as tags, categories, links, and trackbacks in separate columns. Blog users and blog content can be organized into distinct column families for better data separation and scalability.

Counters for Analytics: In web applications, tracking and categorizing page visitors is essential for analytics. Cassandra supports this through the CounterColumnType, which can be defined when creating a column family.

When Not to Use Cassandra

Traditional ACID Transactions Cassandra does not support full ACID compliance

Query-Based Aggregation limited support for complex aggregation operations.

Early Prototypes or Tech Spikes Cassandra can be costly to adapt during early development phases (less ideal for rapidly evolving prototypes)

Wide Column NoSQL Databases
Cassandra

TEXAS A&M UNIVERSITY

Try it Yourself

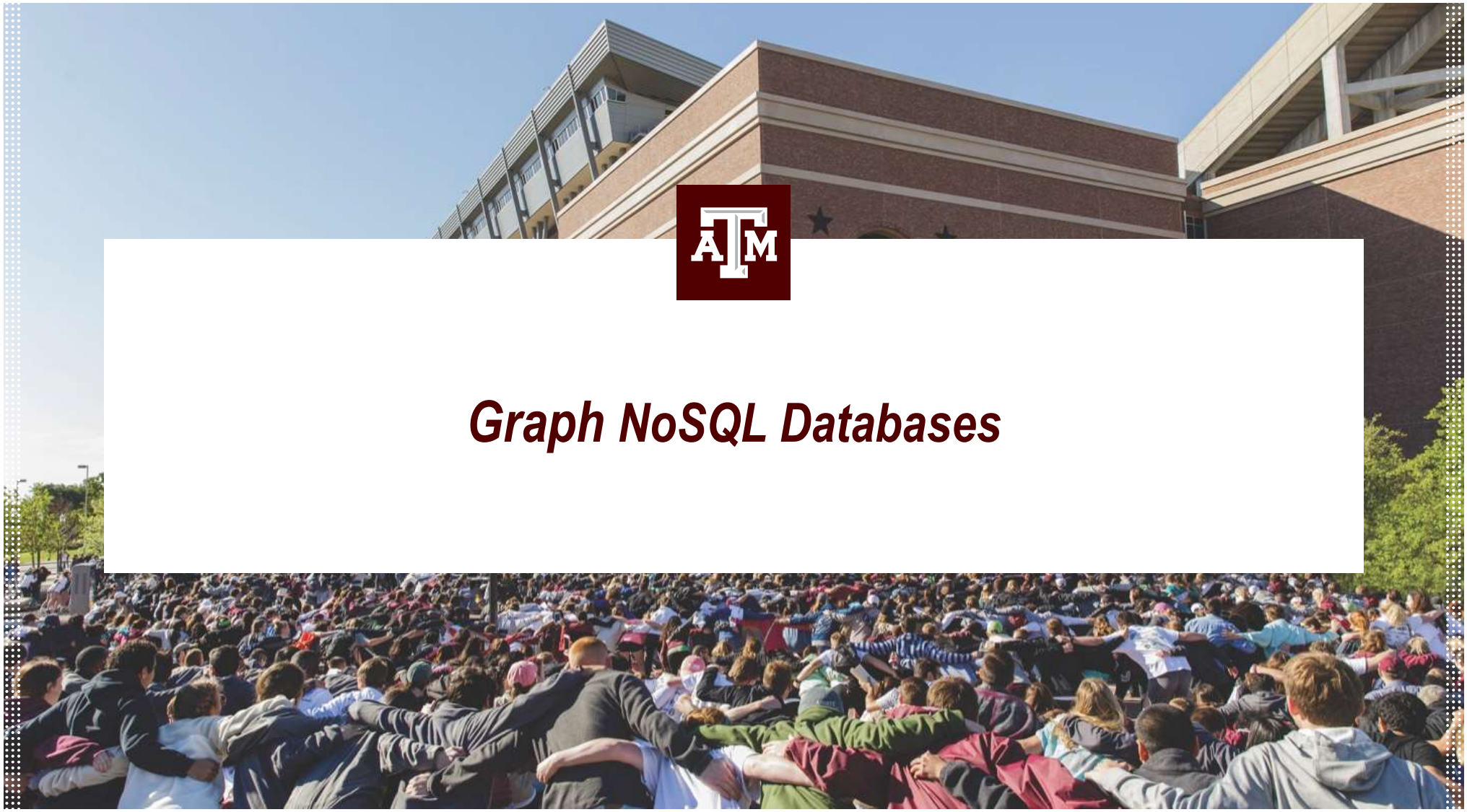
1. Apache Cassandra | Apache Cassandra Documentation
2. Get Started with Apache Cassandra (https://cassandra.apache.org/_/quickstart)

>> 'docker exec –it <container name> bash'

>> 'cqlsh'

Cassandra for machine learning:

https://community.ibm.com/community/user/blogs/sri-kamani/2023/03/31/machine-learning-using-cassandra-and-scikit-learn



Graph NoSQL Databases

Graph NoSQL databases

TEXAS A&M UNIVERSITY

Key characteristics

- Graph databases recognize entities and relationships between them.
- In the graph database world, these entities are called nodes and the relationships between them are called edges (all of these terms come from mathematical graph theory).
- New edges can be added (or old ones removed) at any time, allowing one-to-many and many-to-many relationships to be expressed easily.
- Constructs like friends, followers, degrees of separation, lists, endorsements, status messages and responses to them are very naturally accommodated in graph databases.
- Best used for relationship-type analysis.
- A subset of triple stores (subject, predicate (relationship), and object)
- A popular graph NoSQL DB is Neo4j

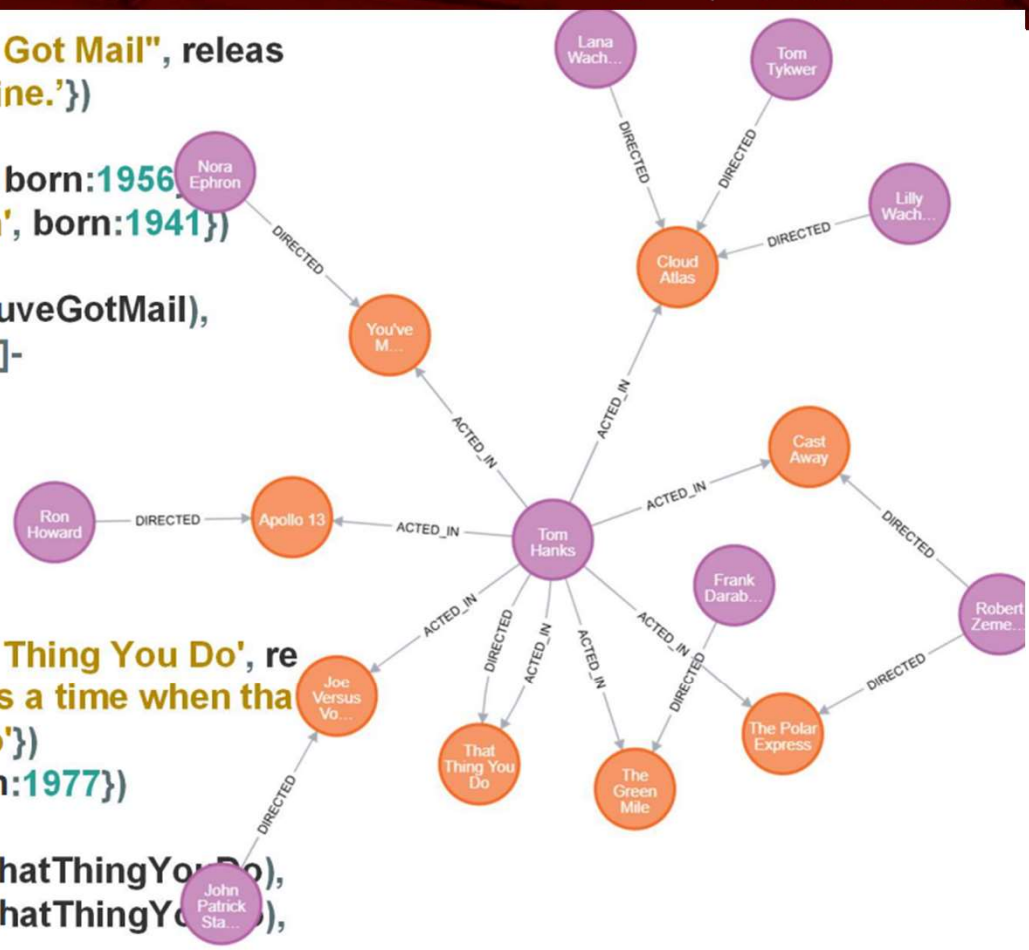
Neo4j Example: Movie Graph

TEXAS A&M UNIVERSITY

```
CREATE (YouveGotMail:Movie {title:"You've Got Mail", releas
ed:1998, tagline:'At odds in life... in love on-line.'})
…
CREATE (TomH:Person {name:'Tom Hanks', born:1956})
CREATE (NoraE:Person {name:'Nora Ephron', born:1941})
CREATE
(TomH)-[:ACTED_IN {roles:['Joe Fox']}]->(YouveGotMail),
(MegR)-[:ACTED_IN {roles:['Kathleen Kelly']}]-
>(YouveGotMail),
…
(NoraE)-[:DIRECTED]->(YouveGotMail)

…

CREATE (ThatThingYouDo:Movie {title:'That Thing You Do', re
leased:1996, tagline:'In every life there comes a time when tha
t thing you dream becomes that thing you do'})
CREATE (LivT:Person {name:'Liv Tyler', born:1977})
CREATE
(TomH)-[:ACTED_IN {roles:['Mr. White']}]->(ThatThingYouDo),
(LivT)-[:ACTED_IN {roles:['Faye Dolan']}]->(ThatThingYouDo),
…
(TomH)-[:DIRECTED]->(ThatThingYouDo)
```



Tutorials - Getting Started

Which NoSQL DB is right for the job?

TEXAS A&M UNIVERSITY

1. Key-value
2. Document
3. Wide-column
4. Graph

Social Networking

"I'm creating a social networking site where I want to quickly traverse relationships between users, their actions, and the like. I should consider a …"

Which NoSQL DB is right for the job?

TEXAS A&M UNIVERSITY

1. Key-value
2. Document
3. Wide-column
4. Graph

Leaderboard

"I have a game site with large amounts of data. I need a leaderboard, which is a simple lookup query."

Which NoSQL DB is right for the job?

TEXAS A&M UNIVERSITY

1. Key-value
2. Document
3. Wide-column
4. Graph

Huge Amounts of Data and Known Queries

"I have a database with a huge amount of data. I know the queries up front, and can model my data around them. Which NoSQL solution might be most appropriate?"

Which NoSQL DB is right for the job?

TEXAS A&M UNIVERSITY

1. Key-value
2. Document
3. Wide-column
4. Graph

General-Purpose CMS

"I am creating a large CMS and have a need for a wide variety of general data types. Which NoSQL database would likely be appropriate?"

Challenge: Relational or NoSQL?

TEXAS A&M UNIVERSITY

Entrepreneurial Drive-Share App

We're entrepreneurs with a team of agile developers. We're creating a new web application that allows time sharing of luxury vehicles. Drivers rent from dealers; they review each other; and the like. The app will have an international audience and should be able scale quickly.

Challenge: Relational or NoSQL?

TEXAS A&M UNIVERSITY

Enterprise Financial App

We work for a large enterprise bank. We're creating an internal auditing application of our financial data and live banking systems. Would relational or NoSQL be best suited here?

Challenge: Relational or NoSQL?

TEXAS A&M UNIVERSITY

Unstructured Big Data

We work for a consulting firm that is using vast amounts of unstructured data to look at consumer spending over the course of a year. Which solution is right for us?

Recap: *General considerations for deciding*

TEXAS A&M UNIVERSITY

Relational

- Strong consistency
- Precise
- Hard to change (structured collection of tables)
- Might be available

NoSQL

- Weak consistency
  (ok to have stale data)
- Approximate okay
- Easy to change
- Must be available

Recap: *General considerations for deciding*

TEXAS A&M UNIVERSITY

Relational

- Defined schemas
- Data integrity
- Scalability and availability are less of a concern than consistency
- Team has SQL expertise

NoSQL

- Agile projects/evolving data
- Availability and/or scalability are more important than consistency
- Speed

Recap: *General considerations for deciding*

TEXAS A&M UNIVERSITY

Relational

- Team fluency and training
- More time defining (and redefining) schema
- Potentially slower to market
- Complex queries
  - SQL is a mature language

NoSQL

- Team fluency and training
- Less time upfront (dynamic schema)
- Often associated with agile/rapid development
- Developer often tasked with writing own code to help with consistency

Recap: *General considerations for deciding*

TEXAS A&M UNIVERSITY

Relational

- Cost of training
- Often expensive at scale
- Queries are easier and less expensive to write

NoSQL

- Cost of training
- Can be expensive with server sprawl
- Potentially more affordable, but depends on added complexity of the system

DB	Strengths	Shortcomings	Applications
Key-value	• Ultrafast Access • Rich data types • Auto-expire keys • Scalability	• No Search on values • Small values only • Consistency across nodes • No tables	• Session Cache • Shopping cart • Scorecard • Real-time queue
Document	• Complex data types • Rich querying • Full text search • Scalability	• No transactions • No media storage • Joins not optimal • No attribute-based security (user-role-based)	• Blogs • Catalogs • RDBMS alternatives • Searchable repository
Wide column	• CQL-like SQL • Optimal updates • Scalability	• Limited transactions • No joins • Query by key only • No order by • Only small blobs	• Data warehouse • Real-time counters • Customer 360 • Write intensive apps • Machine learning
Graph	• Transactions • Relation-based queries • Deep/complex relationships • Relation attributes	• Complex data • No aggregation • No subqueries • Large volume queries	• Social media • Network topology • Recommendation engines • Location services • Search

Vector Databases

Main ideas:

1. Represent complex data points (e.g. images, text, sounds) as numerical arrays embedded in a high dimensional space (i.e. vector).
2. Index vectors for efficient similarity/distance based search and retrieval (e.g. approximate nearest neighbor algorithms).
3. AI/ML models already generate efficient vector embeddings of complex data (e.g. autoencoders) that can be used and stored for such purposes.
4. Designed for semantic search, recommendation systems, image search, personalization systems, anomaly/fraud detection.

Food Discovery Demo – Qdrant

Food Discovery with Qdrant