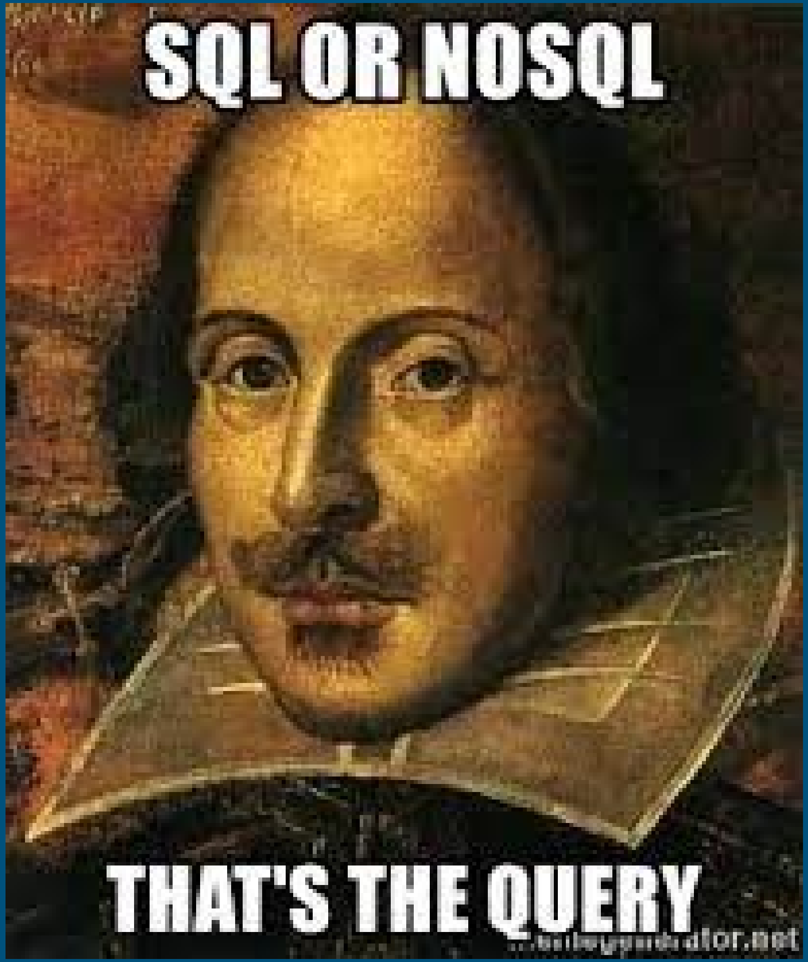
Databases 11

Non-relational databases: overview, general introduction to Neo4j

I. Overview of NoSQL databases

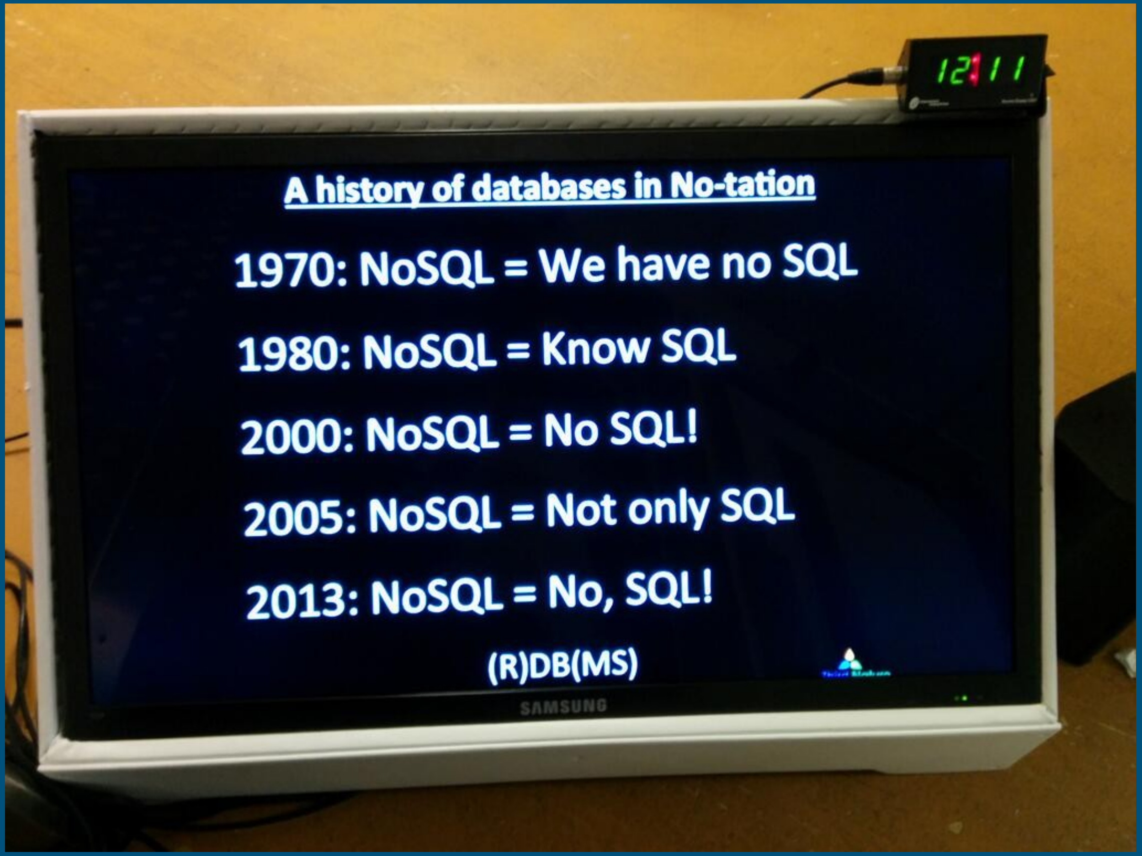
What is NoSQL?

- SQL – is a **language** for the relational data model
- NoSQL – is **a way of data organization and managing** which is different from relational model



How NoSQL-DB and DBMS were created?

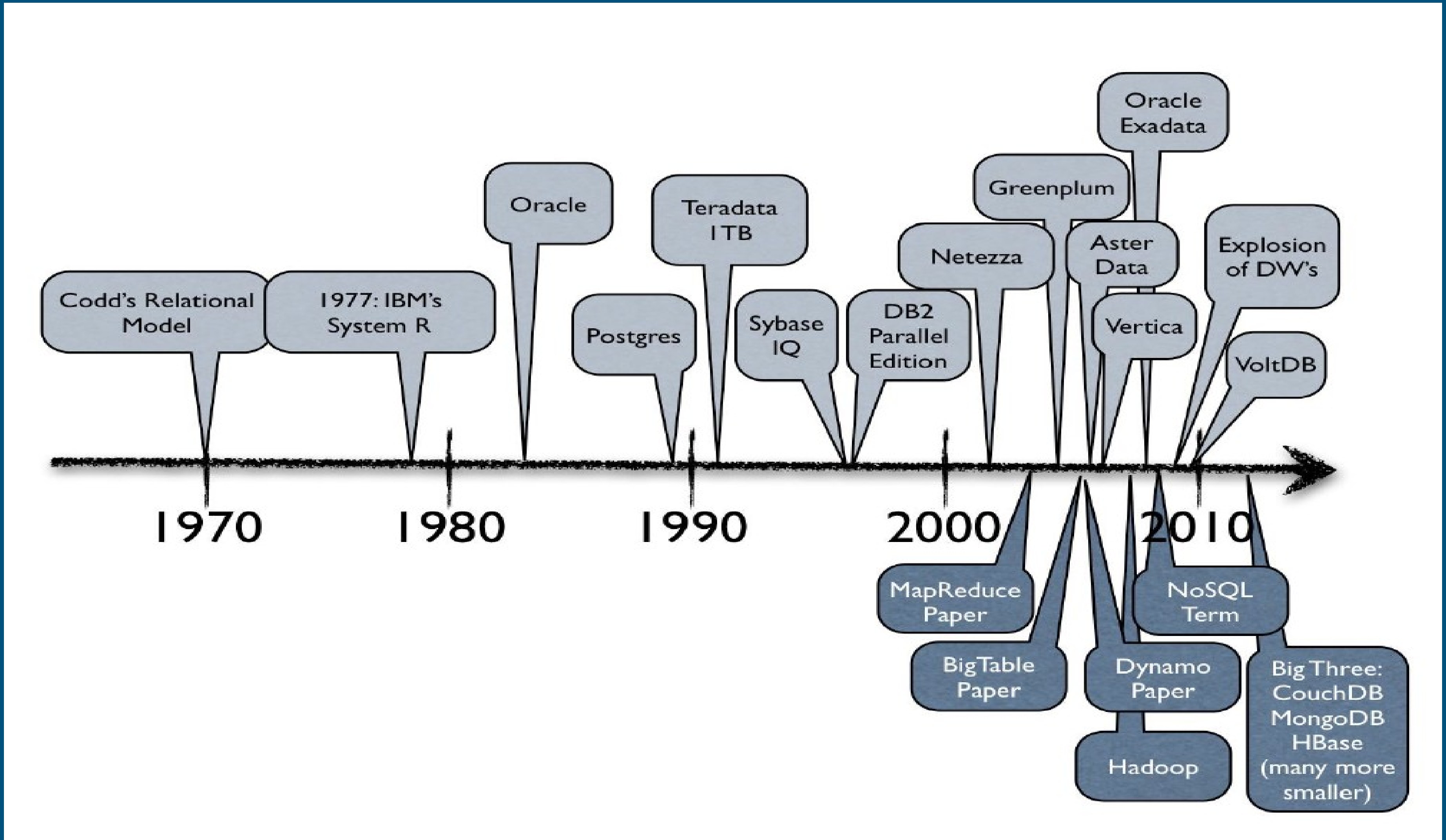
In fact, the first data organization systems did not comply with the provisions of the relational model. NoSQL is an organization for storing and processing data that has existed almost since the first production computers - after all, the data had to be stored somehow!



A little bit of history

- At the very beginning - master files
- Next (1960s) - network and hierarchical databases:
  - article by GE engineers Charles W Bachman, S. B. Williams “A general purpose programming system for random access memories” (1964/1965), data management system Integrated Data Store (IDS) (1963)
  - article by IBM engineer - William C. McGee “Generalized File Processing”, (1969), Information Management System (IBM IMS)
  - Conceptual heirs of hierarchical databases:
    - File systems
    - XML
  - Network databases have “turned” into graph databases
- 1969 – Edgar Codd publishes “A Relational Model of Data for Large Shared Data Banks” for IBM internal use (published 1970)

A little bit of history



Term NoSQL

The term 'NoSQL' appeared in 1998. It was used by developer Carlo Strozzi for his lightweight DBMS, which, although not using SQL, implemented a relational model.

Some say that now instead of the term NoSQL it would be more correct to use the term NoREL

Trends in the development of database models in 1990

- **BigData** – large volumes of data, scalability
- **Web** – ensuring data availability for any resource
- **Agile** – speed of development (MVP is completed in 3-4 months)

RDB principles: ACID

- Atomicity
- Consistency
- Isolation
- Durability

IBM Information Management System (DBSM) has been started to support ACID since 1973

CAP - theorem

According to the theorem, there are three main properties that characterize any database (DBMS):

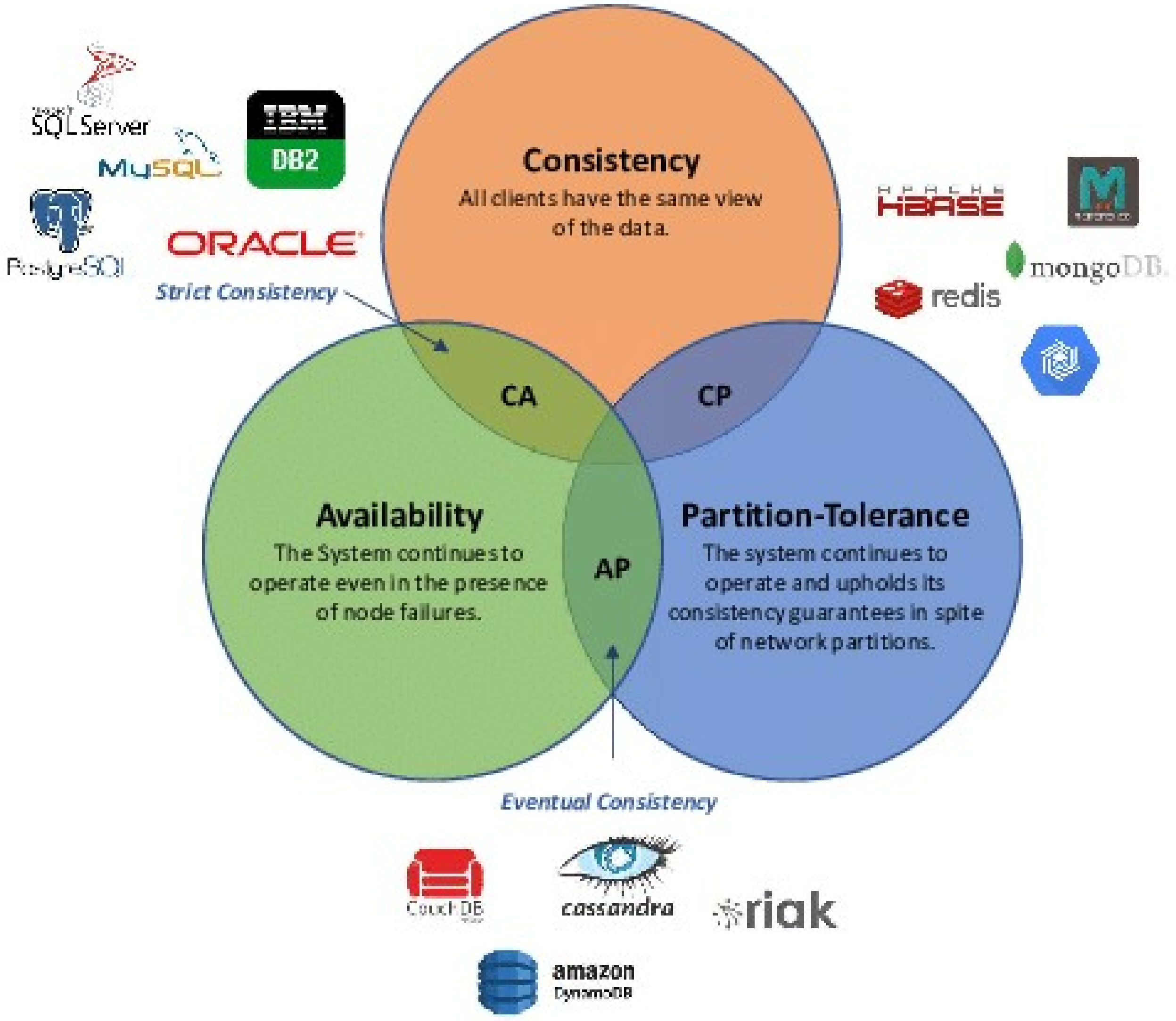
- Consistency
- Availability
- Partition tolerance

and we can maximize no more than two of them

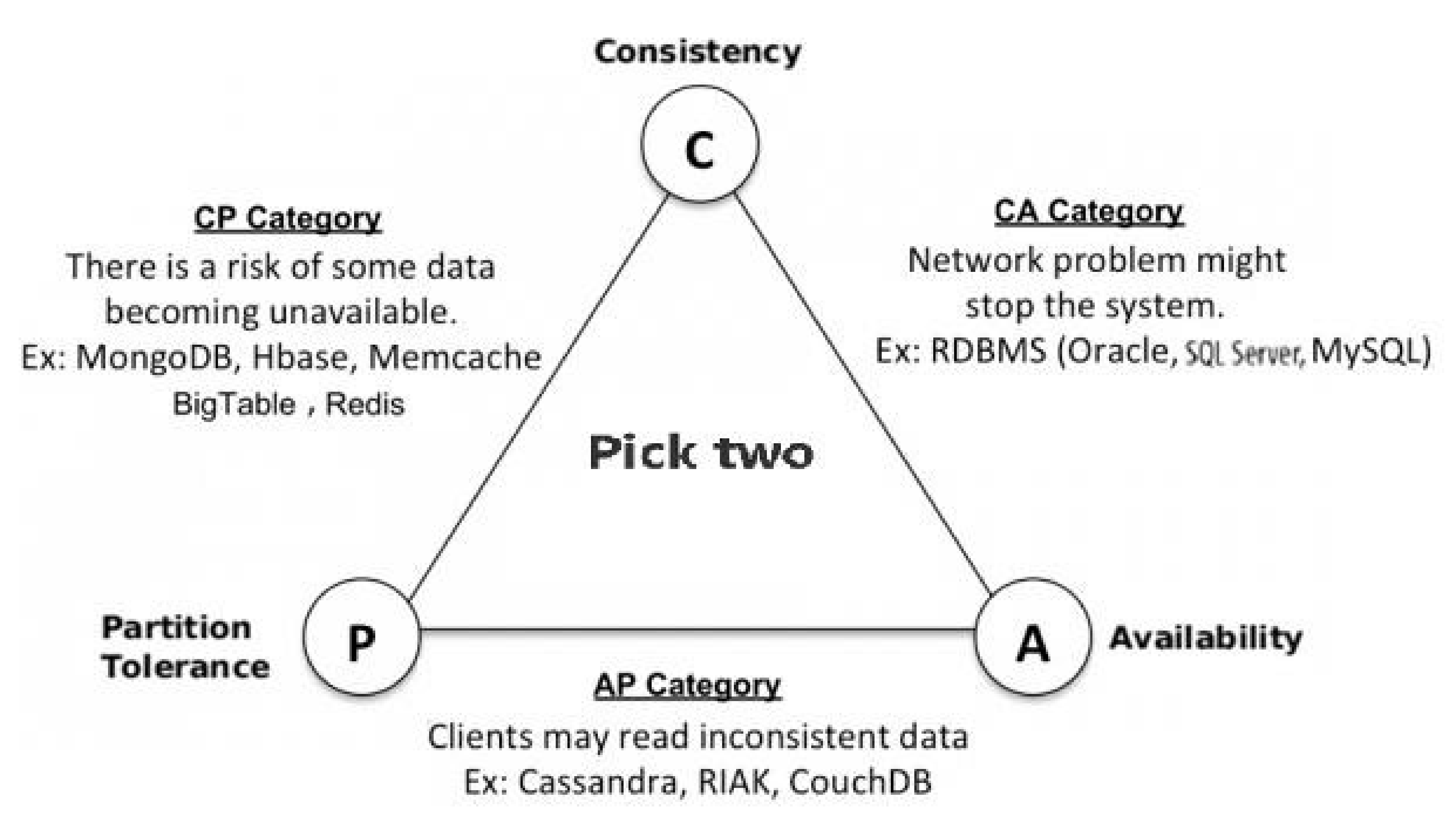
Application:

allows you to classify all database models according to pairs of specified properties.

Proposed by E. Brewer in ~2000


Consistency
All clients have the same view of the data.
Availability
The System continues to operate even in the presence of node failures.
Partition-Tolerance
The system continues to operate and upholds its consistency guarantees in spite of network partitions.
CA
CP
AP
Strict Consistency
Eventual Consistency
SQL Server
MySQL
IBM
DB2
PostgreSQL
ORACLE
HBASE
mongoDB
redis
CouchDB
cassandra
riak
amazon
DynamoDB

CAP: disadvantages of pairs



Alternative for ACID - BASE

The BASE model appeared as a “consequence” of the CAP theorem. It is “guided” when designing NoSQL DBMS and contrasted with ACID

BASE:

Basically Available

- Guaranteed response to every data request

Soft State

- Data may change in the absence of input operations

Eventual Consistency

- Ultimately (when input operations stop) changes in the data will be pushed to all database nodes

NoSQL: classification

There are many categories of NoSQL systems, for example:

- Key-Value Stores
- Column-oriented DBMS
- Document Stores
- Graph DBMS
- RDF Stores (data presented in triplets: subject-predicate-object)
- Native XML DBMS
- Content Stores (storing entire content, including metadata)
- Search Engines (search engines, e.g. Elasticsearch)

NoSQL: classification

The following four main types of NoSQL models are considered:

- GraphDB
- Document DB
- Key-Value Stores
- Column-oriented DB (special case - Wide-column DB)

- Let’s take a closer look at each of these types

Column-oriented vs Wide-column

Column-oriented database:

- Primary Use: Optimized for aggregation operations and analytical queries that read large amounts of data from specific columns across an entire table.
- Data storage: Data is stored in columns, allowing for fast data aggregations and compression as columns contain the same type of data.
- examples: Apache ClickHouse, Vertica, SAP HANA

Column-oriented vs Wide-column

Wide-column database:

- Main purpose: designed for scalability and flexibility in applications that require processing large amounts of data with different structures.
- Data storage: Data is organized in tables, where each row can have a different number of columns and structure. Columns are grouped into column families that are stored together, improving read and write performance.
- examples: Apache Cassandra, Google Bigtable, Apache HBase.

Column-oriented vs Wide-column

- **Storage structure:** In column-oriented databases, the structure is fixed and uniform, each record or row contains data for each column. In wide-column databases, each row can have a different set of columns, which gives additional flexibility in data management.
- **Column Families:** Wide-column databases use the concept of column families, where columns that are logically related to each other are stored together. This distinguishes them from column-oriented databases, where columns are not typically grouped in this way.

NoSQL: Key-Value Stores / overview

Rank			DBMS	Database Model	Score		
Apr 2024	Mar 2024	Apr 2023			Apr 2024	Mar 2024	Apr 2023
1.	1.	1.	Redis	Key-value, Multi-model	156.44	-0.56	-17.11
2.	2.	2.	Amazon DynamoDB	Multi-model	77.57	-0.15	+0.12
3.	3.	3.	Microsoft Azure Cosmos DB	Multi-model	29.85	-0.54	-5.23
4.	4.	4.	Memcached	Key-value	20.74	-0.07	-1.25
5.	↑ 6.	5.	etcd	Key-value	7.64	-0.02	-0.99
6.	↓ 5.	6.	Hazelcast	Key-value, Multi-model	6.87	-0.79	-0.89
7.	7.	7.	Aerospike	Multi-model	6.10	-0.41	-0.30
8.	8.	8.	Ehcache	Key-value	5.23	-0.32	-0.91
9.	9.	↑ 10.	Riak KV	Key-value	4.44	-0.51	-0.75
10.	↑ 14.	↑ 20.	InterSystems IRIS	Multi-model	4.39	+0.57	+1.19

NoSQL: Key-Value Stores / Characteristics

Advantages:

- The simplest type of the others. The principle of operation is similar to the associative array on hash tables
- There is no data schema, value can be any object, the key can be an array rather than a single value

Disadvantages:

- Due to the need to scan the entire hash table when reading, it may not scale very well
- Suitable only for very simple data - it is impossible to implement connections using the model itself
- Difficulty building complex queries

NoSQL: Wide-column DB / Overview

Rank			DBMS	Database Model	Score		
Apr 2024	Mar 2024	Apr 2023			Apr 2024	Mar 2024	Apr 2023
1.	1.	1.	Cassandra	Wide column, Multi-model	103.86	-0.72	-7.94
2.	2.	2.	HBase	Wide column	31.25	-0.35	-6.55
3.	3.	3.	Microsoft Azure Cosmos DB	Multi-model	29.85	-0.54	-5.23
4.	4.	4.	Datastax Enterprise	Wide column, Multi-model	6.31	-0.76	-0.48
5.	5.	5.	ScyllaDB	Wide column, Multi-model	5.27	-0.42	-0.58
6.	6.	6.	Microsoft Azure Table Storage	Wide column	4.92	-0.43	-0.76
7.	↑ 8.	7.	Accumulo	Wide column	3.61	-0.17	-1.57
8.	↓ 7.	8.	Google Cloud Bigtable	Multi-model	3.58	-0.29	-1.34
9.	↑ 10.	↑ 10.	Amazon Keyspaces	Wide column	0.92	+0.06	+0.18
10.	↓ 9.	↓ 9.	HPE Ezmeral Data Fabric	Multi-model	0.89	-0.16	-0.33

NoSQL: Wide-column DB / Characteristics

Advantages:

- Flexibility in physical data storage – table rows are distributed across different servers, a row can contain columns of different data types
- Fast processing of queries related to retrieving data from a column
- Good compression (due to serialization of columns, i.e. arrays storing the same type of data)

Disadvantages:

- Long (compared to RDBMS) processing of queries that retrieve entire rows. Long processing of JOIN queries, etc. – where line-by-line reading is required
- Long (compared to RDBMS) writing data to a string
- (almost complete) inability to create composite indexes

NoSQL: Document DB / overview

Rank			DBMS	Database Model	Score		
Apr 2024	Mar 2024	Apr 2023			Apr 2024	Mar 2024	Apr 2023
1.	1.	1.	MongoDB	Document, Multi-model	423.96	-0.57	-17.93
2.	2.	2.	Amazon DynamoDB	Multi-model	77.57	-0.15	+0.12
3.	3.	3.	Databricks	Multi-model	76.33	+1.99	+15.36
4.	4.	4.	Microsoft Azure Cosmos DB	Multi-model	29.85	-0.54	-5.23
5.	5.	5.	Couchbase	Document, Multi-model	18.46	-0.69	-5.30
6.	6.	6.	Firebase Realtime Database	Document	15.00	-0.07	-3.22
7.	7.	7.	CouchDB	Document, Multi-model	10.26	-1.47	-4.36
8.	8.	8.	Google Cloud Firestore	Document	8.96	-1.01	-2.13
9.	9.	↑ 10.	Realm	Document	7.71	+0.01	-0.57
10.	10.	↓ 9.	MarkLogic	Multi-model	6.50	-0.57	-1.84

NoSQL: Document DB / Characteristics

Advantages:

- No data schema support
- Convenient format (MongoDB – JSON, one of the web standards)
- Fast write/read
- Convenient replication and sharding

Disadvantages:

- No data schema support!!!
- Poor support for complex SQL-like queries

NoSQL: Graph DB / Обзор

Rank			DBMS	Database Model	Score		
Apr 2024	Mar 2024	Apr 2023			Apr 2024	Mar 2024	Apr 2023
1.	1.	1.	Neo4j	Graph	44.47	+0.11	-7.13
2.	2.	2.	Microsoft Azure Cosmos DB	Multi-model	29.85	-0.54	-5.23
3.	3.	3.	Aerospike	Multi-model	6.10	-0.41	-0.30
4.	4.	4.	Virtuoso	Multi-model	4.20	-0.19	-2.04
5.	5.	5.	ArangoDB	Multi-model	3.77	-0.45	-1.03
6.	6.	6.	OrientDB	Multi-model	3.27	-0.11	-0.79
7.	↑ 8.	↑ 9.	GraphDB	Multi-model	3.10	+0.19	+0.76
8.	↓ 7.	↑ 11.	Memgraph	Graph	3.00	-0.09	+0.88
9.	9.	↓ 7.	Amazon Neptune	Multi-model	2.58	-0.24	-0.11
10.	10.	10.	NebulaGraph	Graph	2.12	-0.24	-0.05

NoSQL: Graph DB / Characteristics

Advantages:

- It is based on a well-developed mathematical model
- It is convenient to build specific models of the subject area - knowledge bases
- Convenient data model extension
- Speed of query processing due to the specifics of the data model

Disadvantages:

- Heterogeneous query languages – no industry standard
- Does not support transaction abstraction
- It is difficult to calculate queries with aggregation

SQL vs NoSQL: реальное положение дел

Rank			DBMS	Database Model	Score		
Apr 2024	Mar 2024	Apr 2023			Apr 2024	Mar 2024	Apr 2023
1.	1.	1.	Oracle	Relational, Multi-model	1234.27	+13.21	+5.99
2.	2.	2.	MySQL	Relational, Multi-model	1087.72	-13.77	-70.06
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	829.80	-16.01	-88.73
4.	4.	4.	PostgreSQL	Relational, Multi-model	645.05	+10.15	+36.64
5.	5.	5.	MongoDB	Document, Multi-model	423.96	-0.57	-17.93
6.	6.	6.	Redis	Key-value, Multi-model	156.44	-0.56	-17.11
7.	7.	↑ 8.	Elasticsearch	Search engine, Multi-model	134.78	-0.01	-6.29
8.	8.	↓ 7.	IBM Db2	Relational, Multi-model	127.49	-0.26	-18.00
9.	9.	↑ 12.	Snowflake	Relational	123.20	-2.18	+12.07
10.	10.	↓ 9.	SQLite	Relational	116.01	-2.15	-18.53

NoSQL: advantages

- Allows you to store large volumes of semi-structured information in a more or less coherent form
- Ease of scaling (compared to RDB and RDBMS)
- Convenient replication and sharding
- Speed of development

NoSQL: disadvantages

- No domain diagram design required
- Encourage accumulation of inconsistent information in the database
- For complex subject areas: the data schema tends to be relational (for the corresponding data categories). At the same time, the advantages of the relational model remain unavailable
- Eventual consistency and the BASE principle introduce risks of error

And finally... Vector databases

- At its core, this is not some new type of database, but something like an add-on over existing storage systems. Key-value databases can be used as a “backend” in such databases
- The goal is to optimize the storage and processing of vector embeddings
- Conceptually includes three parts: data store, vector indexing mechanism, query mechanism
- Indexing and queries are processed using special libraries that are not related to database technology. The standard option is FAISS (Facebook AI Similarity Search). An important difference from query results in other types of databases is that the correctness of the result is probabilistic.

Examples

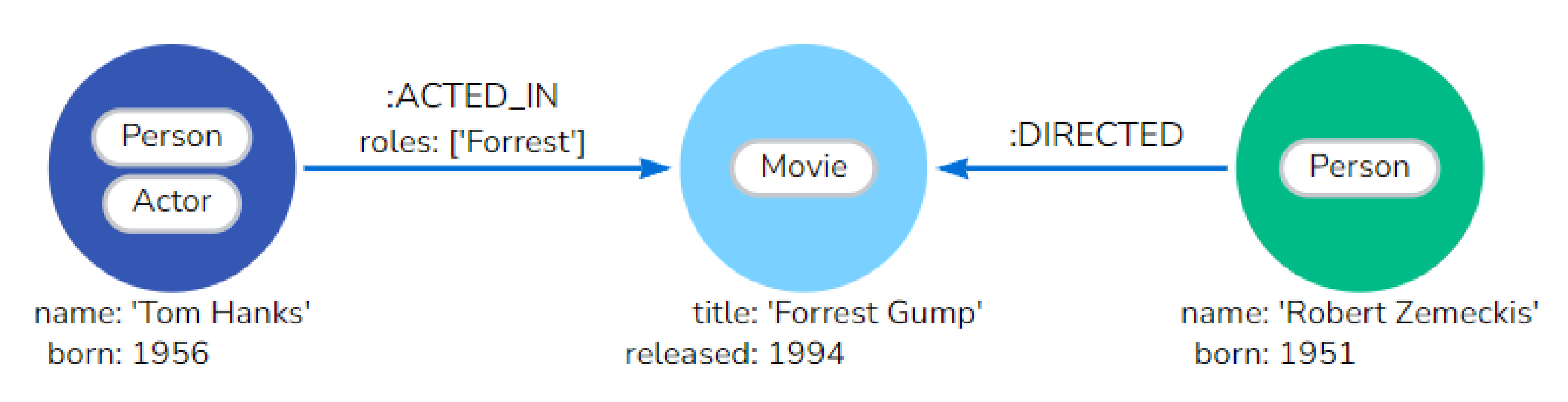
- Milvus
- Pincone
- Weaviate

II. Introduction to Neo4j

Neo4j: data model

Data and relationships between them are represented in the form of 3 model objects:

- nodes
- relationships
- properties.



Neo4j: Nodes

- Nodes represent entities or objects in a graph. They are similar to records or objects in a traditional database.
- Each node can have one or more labels that define its role or type in the graph.
- Tags help you organize and query data efficiently.
- Nodes can have properties, which are key-value pairs that provide additional information about the node. Properties can be indexed for faster searching.
- Nodes can have labels
- Example syntax for creating a node:

```
CREATE (:Person:Actor {name: 'Tom Hanks', born: 1956})
```

Neo4j: Properties

- Properties are key-value pairs that are used to store information about nodes and relationships.
- The property value can:
  - contain different data types such as number, string or boolean.
  - contain a list (array) consisting, for example, of strings, numbers or Boolean values (the components must belong to the same type).
- Example of properties with values of different types:

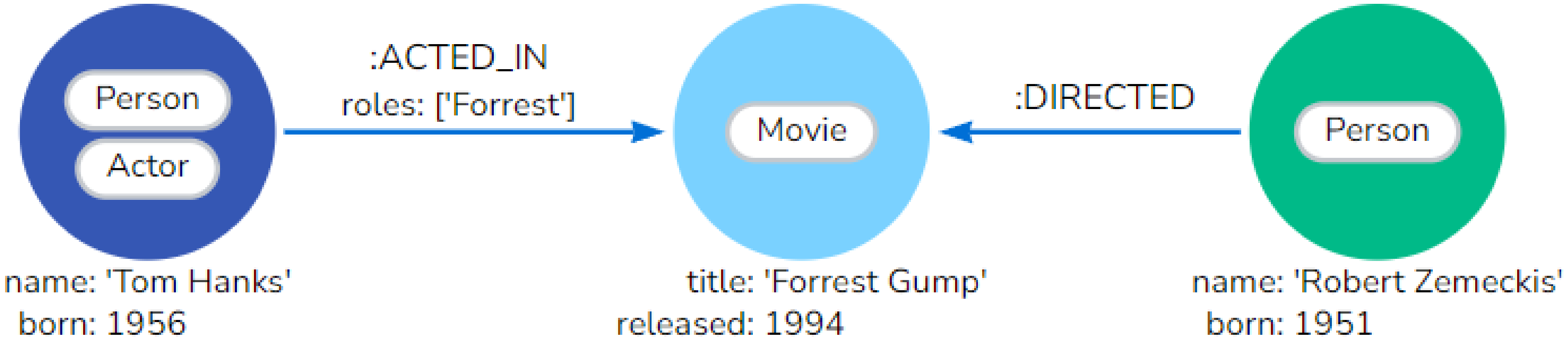
```
CREATE (:Example{a: 1, c: 'This is an example string', b: 3.14}, f: [1, 2, 3])
```

Neo4j: Relationships

- A relationship describes how a source node and a target node are related. A node can have a connection with itself.
- Relationships are always directed and have a start node and an end node. They represent a relationship between two nodes and can have a specific type or label.
- Like nodes, relationships can have properties, which provide additional information about the relationship.
- The syntax for creating a relationship is:

```
CREATE ()-[:ACTED_IN {roles: ['Forrest'], performance: 5}]->()
```

Neo4j: creation of elementary DB




```
CREATE (:Person:Actor {name: 'Tom Hanks', born: 1956})-
[:ACTED_IN {roles: ['Forrest']}]->(:Movie {title: 'Forrest Gump'})<-
[:DIRECTED]-(:Person {name: 'Robert Zemeckis', born: 1951})
```

Neo4j: CQL (Cypher Query Language)

- Keywords similar to SQL are used: MATCH, WHERE, CREATE, DELETE, RETURN - and others
- Example of pattern matching:

```
MATCH (me)-[:KNOWS*1..2]-(remote_friend)
WHERE me.name = 'Filipa’
RETURN remote_friend.name
```

- Aggregating functions are allowed: COUNT, SUM, MIN, MAX
- It is acceptable to use standard comparison operators and logical operators: >, <, =, AND, OR, NOT:

```
MATCH (node:Label)
WHERE node.property > 10 RETURN node
ORDER BY node.property
```

- In addition to CREATE, constructs such as SET, DELETE, and MERGE are used to define data. Example of using MERGE:
  - MERGE (m:Movie {title: 'The Matrix', year: 1999})
  - Used to avoid data duplication (in the example above, the node will not be created if there is already a node with similar properties in the database)

Neo4j: extra opportunities

- It is allowed to create indexes:

```
CREATE INDEX example_index_1 FOR (a:Actor) ON (a.name)
```

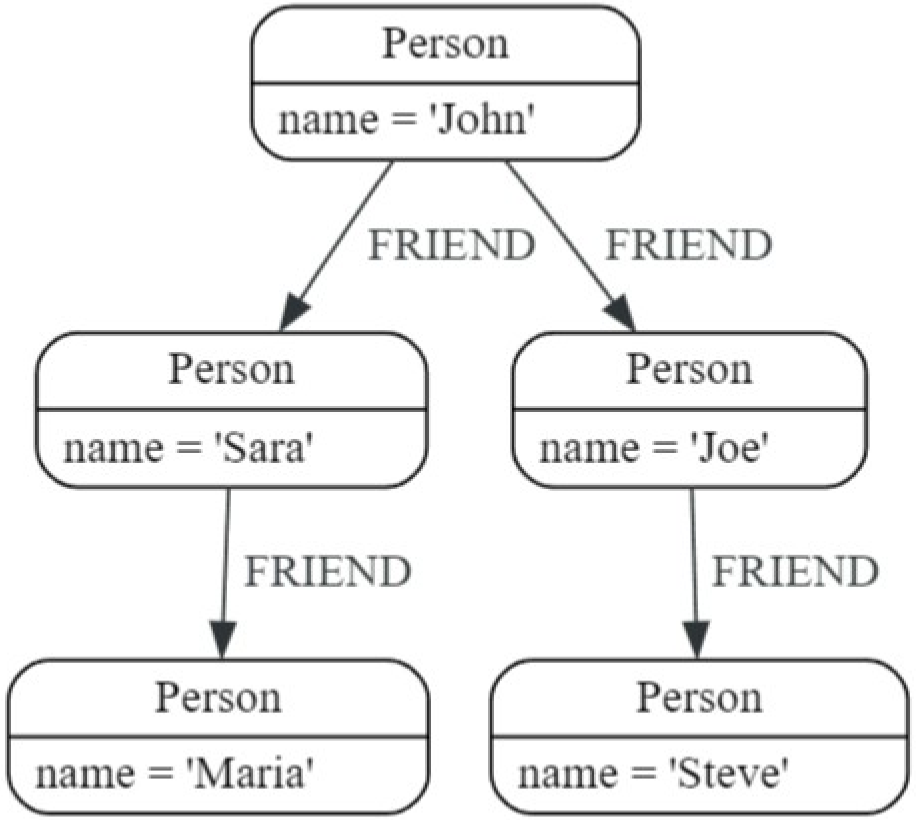
- It is allowed to impose restrictions

```
CREATE CONSTRAINT constraint_example_1 FOR (movie:Movie)
REQUIRE movie.title IS UNIQUE
```

- Creation of custom functions is allowed

Neo4j: example

```
MATCH (john:Person {name: 'John’})
MATCH (john)-[:FRIEND]->(friend)
RETURN friend.name AS friendName
```



How does it work

`MATCH (john:Person {name: 'John'})`	**john** ({name: 'John'})
`MATCH (john)-[:FRIEND]->(friend)`	**john** / **friend** ({name: 'John'}) / ({name: 'Sara'}) ({name: 'John'}) / ({name: 'Joe'})
`RETURN friend.name AS friendName`	**friendName** 'Sara' 'Joe'

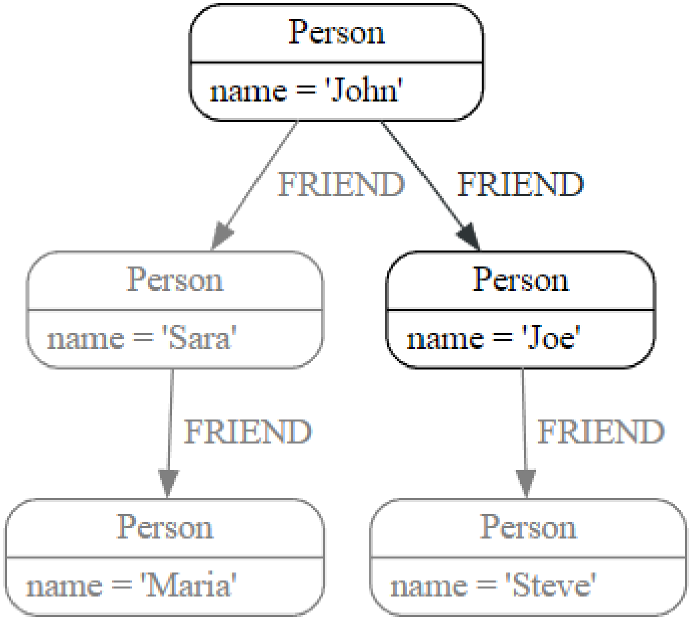
Another example

```
MATCH (j:Person)
WHERE j.name STARTS WITH "J«
CREATE (j)-[:FRIEND]->(jj:Person {name: "Jay-jay"})
```

- the query finds all nodes whose name property begins with "J", and for each such node creates another node with the name property set to "Jay-jay".

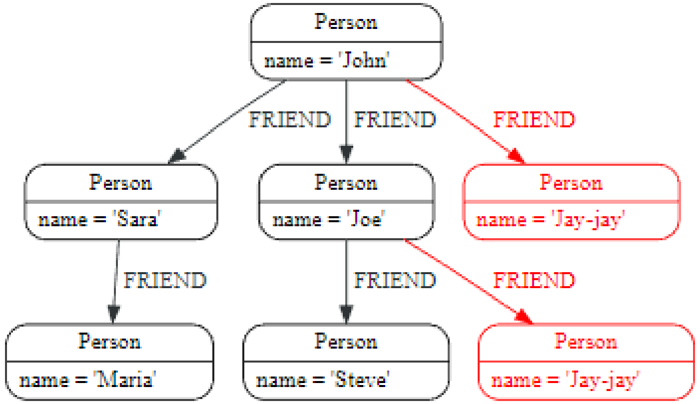
<table>
<tr><th>Clause</th><th>Table of intermediate results after the clause</th><th>State of the graph after the clause, changes in red</th></tr>
<tr>
<td>

```
MATCH (j:Person) WHERE
j.name STARTS WITH "J"
```

</td>
<td>
<table>
<tr><th>j</th></tr>
<tr><td>({name: 'John'})</td></tr>
<tr><td>({name: 'Joe'})</td></tr>
</table>
</td>
<td>


</td>
</tr>
<tr>
<td>

```
CREATE (j)-[:FRIEND]->
(jj:Person {name: "Jay-
jay"})
```

</td>
<td>
<table>
<tr><th>j</th><th>jj</th></tr>
<tr><td>({name: 'John'})</td><td>({name: 'Jay-jay'})</td></tr>
<tr><td>({name: 'Joe'})</td><td>({name: 'Jay-jay'})</td></tr>
</table>
</td>
<td>


</td>
</tr>
</table>

- Recommendations for interpreting relational data into graph data from the official guide Neo4j (*Tutorial: Import Relational Data Into Neo4j - Developer Guides*):

When deriving a graph model from a relational model, you should keep a couple of general guidelines in mind.

1. A *row* is a *node*.
2. A *table name* is a *label name*.
3. A *join or foreign key* is a *relationship*.