

Базы данных

Лекция 11

Нереляционные БД: обзор, общее знакомство с Neo4j

МФТИ, 2024

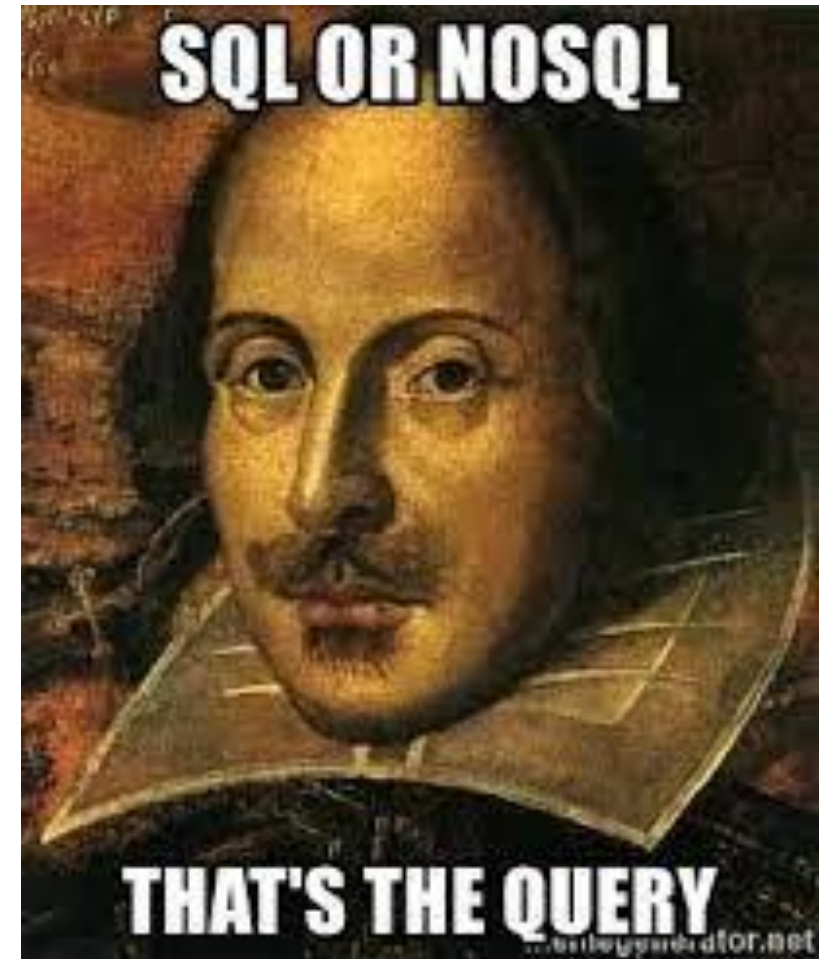
Игорь Шевченко

[@igorshvch](https://github.com/igorshvch)

I. Обзор NoSQL - баз данных

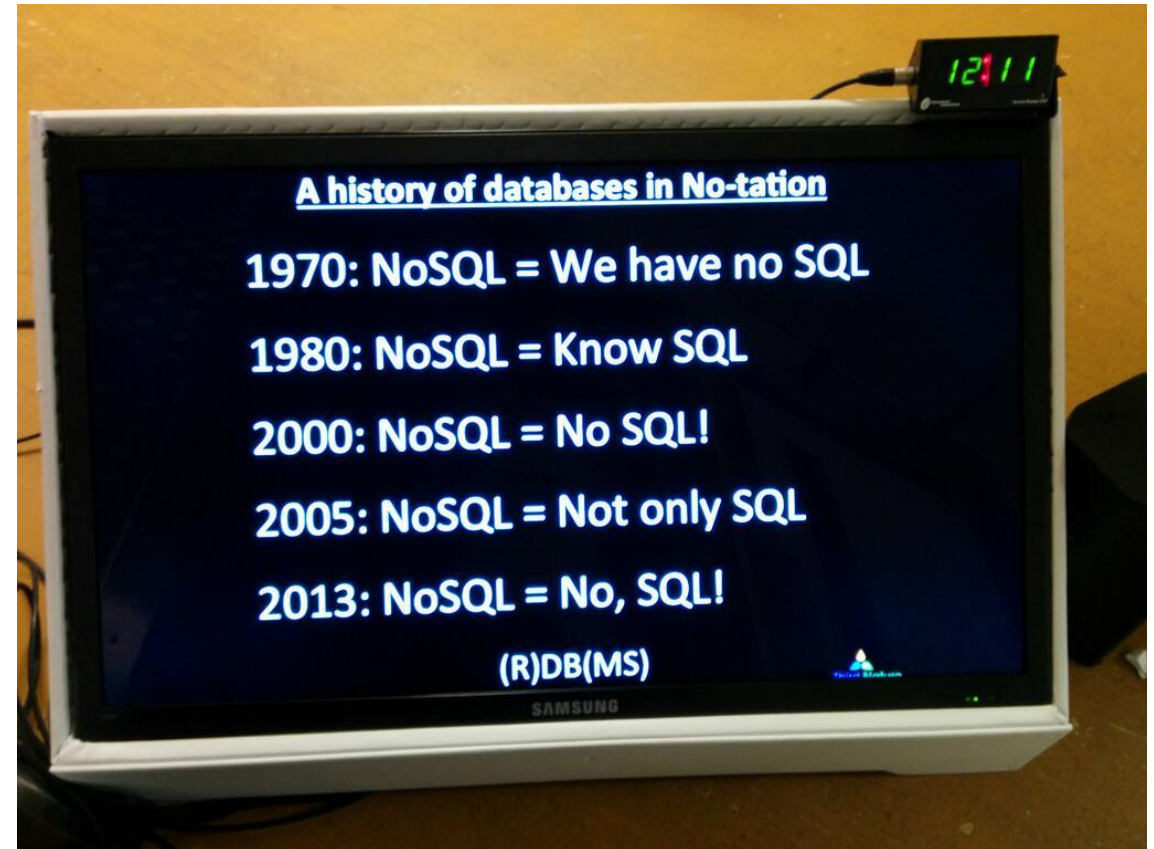
NoSQL: что это такое?

- SQL – **язык** для реляционной модели данных
- NoSQL – **способ организации и управления данными**, отличная от реляционной модели



Как появились NoSQL-БД и СУБД?

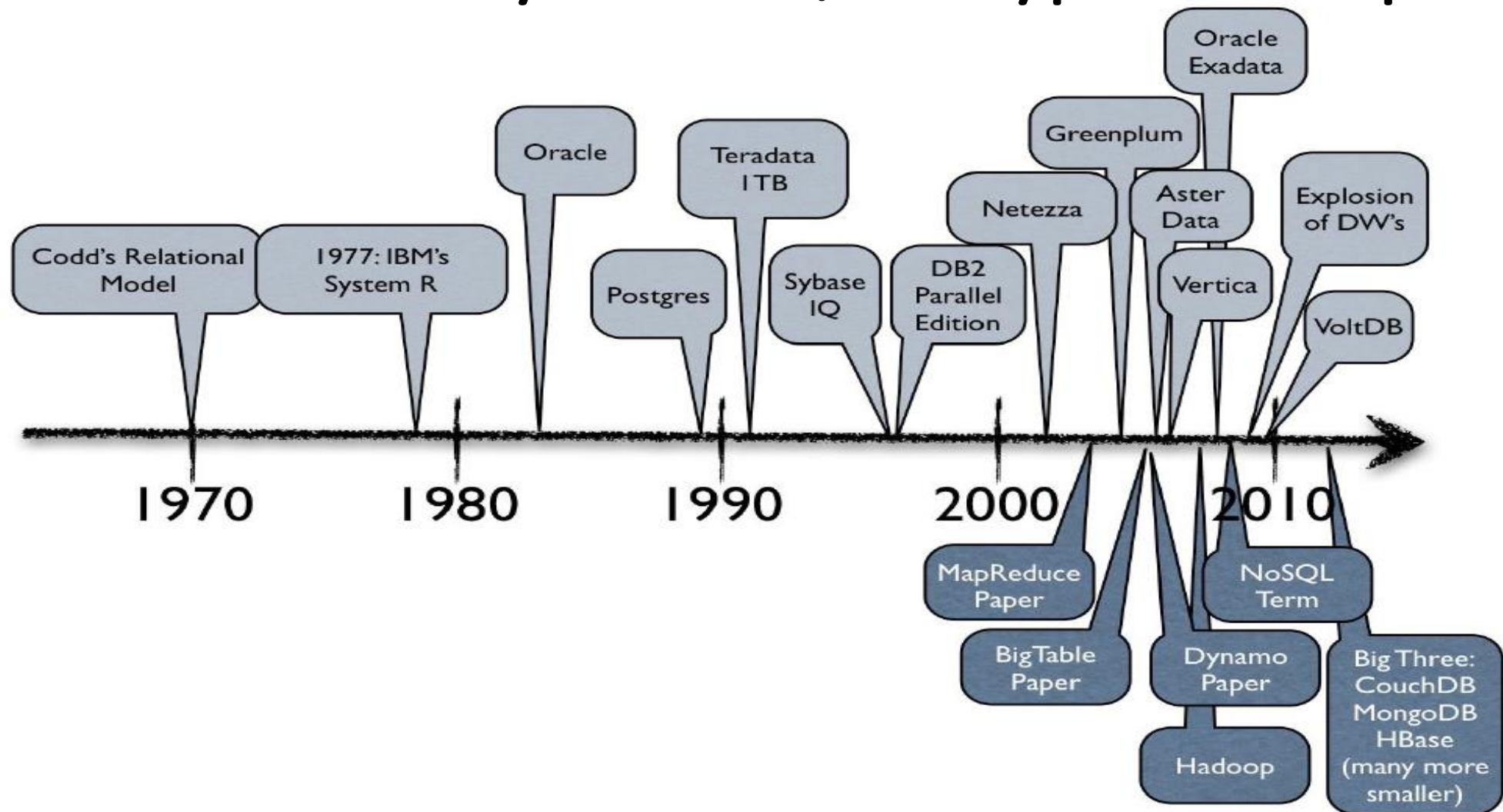
Фактически первые системы организации данных не отвечали положениям реляционной модели. NoSQL – это организация хранения и обработки данных, которая существует практически с первых серийных образцов компьютеров – ведь данные надо было как-то хранить!



Небольшое отступление/экскурс в историю

- В самом начале - мастер-файлы
- Дальше (1960-е) – сетевые и иерархические базы данных:
 - статья инженеров GE *Charles W Bachman, S. B. Williams* «*A general purpose programming system for random access memories*» (1964/1965 г.), система менеджмента данных Integrated Data Store (IDS) (1963 г.)
 - статья инженера IBM - *William C. McGee* «*Generalized File Processing*», (1969 г.), система управления данными Information Management System (IBM IMS)
 - Концептуальные наследники иерархических БД:
 - Файловые системы
 - XML
 - Сетевые БД «превратились» в графовые БД
- 1969 – Эдгар Кодд издает для внутреннего пользования IBM “*A Relational Model of Data for Large Shared Data Banks*” (опубликована в 1970)

Небольшое отступление/экскурс в историю



Кстати: термин NoSQL



Термин 'NoSQL' появился в 1998 году. Его использовал разработчик Carlo Strozzi для своей легковесной СУБД, которая хотя и не использовала SQL, реализовывала реляционную модель.

Некоторые говорят, что сейчас вместо термина NoSQL правильнее было бы использовать термин NoREL

Тенденции развития моделей БД в 1990

- **BigData** – большие объемы данных, масштабируемость
- **Web** – обеспечение доступности данных для любых ресурсов
- **Agile** – быстрота разработки (MVP делают за 3-4 месяца)

Принципы работы РБД: ACID

- **Atomicity — Атомарность**
- **Consistency — Согласованность**
- **Isolation — Изолированность**
- **Durability — Надежность**

IBM Information Management System (СУБД) начала поддерживать ACID с 1973 года

Обобщение: теорема CAP (Брюера)

Согласно теореме, есть три основных свойства, характеризующих любую БД (СУБД):

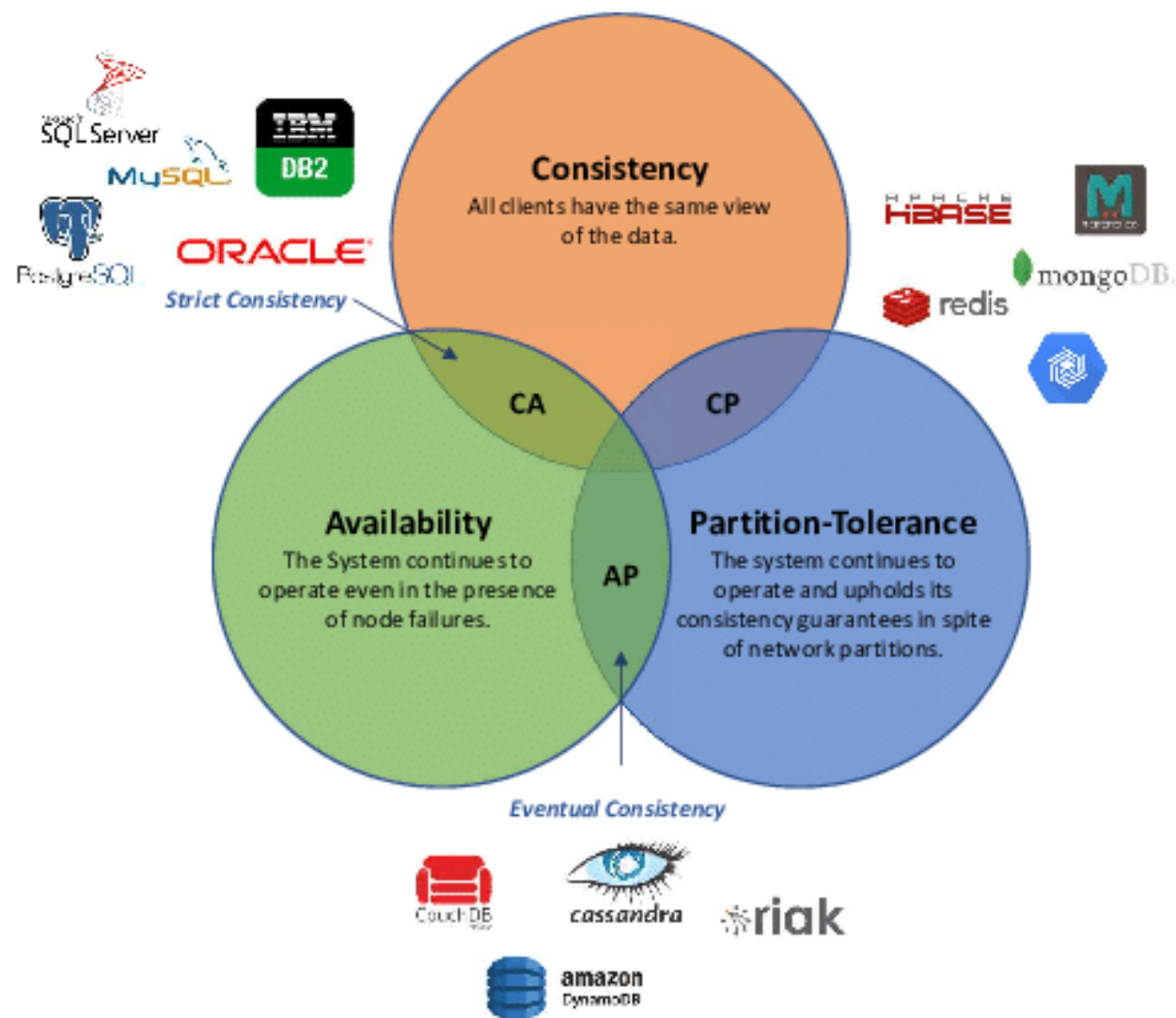
- *Consistency* (согласованность данных)
 - *Availability* (доступность)
 - *Partition tolerance* (устойчивость к разделению)
- **и максимизировать мы можем не более двух из них**

Применение:

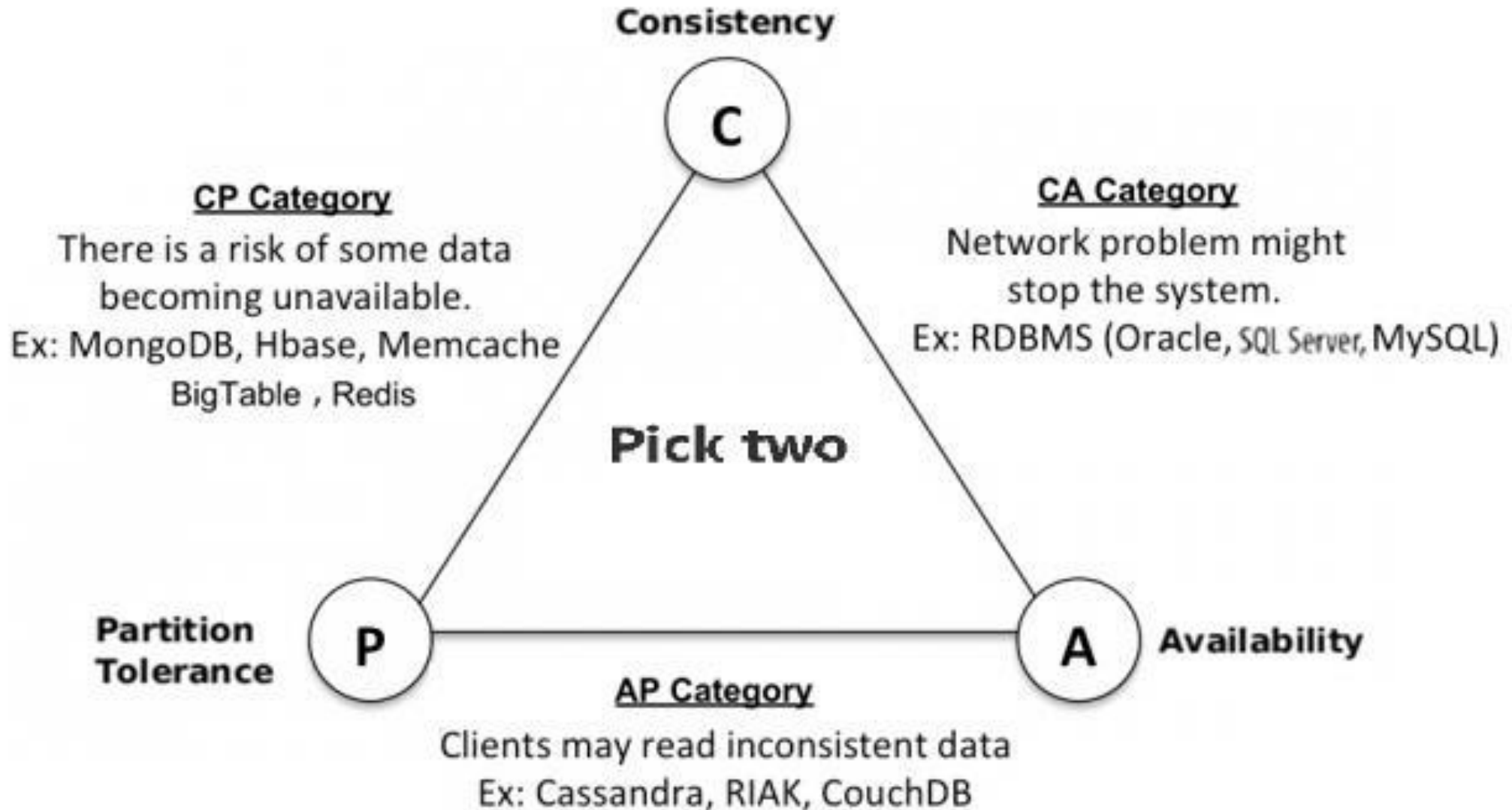
позволяет классифицировать все модели БД по парам указанных свойств.

Предложена Э. Брюером в ~2000 году

Обобщение: теорема CAP (Брюера)



CAP: недостатки максимизации параметров



Альтернатива ACID - BASE

- Модель BASE появилась как «следствие» CAP-теоремы. Ей «руководствуются» при проектировании NoSQL-СУБД и противопоставляют ее ACID
- BASE:
 - **Basically Available (базовая доступность)**
 - Гарантирован ответ на каждый запрос к данным
 - **Soft State (неустойчивое состояние)**
 - Данные могут меняться в отсутствие операций ввода
 - **Eventual Consistency (согласованность в конечном счете)**
 - В конечном итоге (при прекращении операций ввода) изменения в данных будут прокинуты во все узлы БД

NoSQL: классификация

Существует множество категорий NoSQL-систем, например:

- Key-Value Stores
- Column-oriented DBMS
- Document Stores
- Graph DBMS
- RDF Stores (данные представлены в тройках: субъект-предикат-объект)
- Native XML DBMS
- Content Stores (хранение контента целиком, включая мета-данные)
- Search Engines (поисковые движки, напр., Elasticsearch)

NoSQL: классификация

В качестве основных видов NoSQL моделей рассматривают четыре следующих:

- 1) Graph DB
- 2) Document DB
- 3) Key-Value Stores
- 4) Column-oriented DB (частный случай - Wide-column DB)

- рассмотрим подробнее каждый из указанных видов

Отличия Column-oriented от Wide-column

- Column-oriented базы данных:
 - основное назначение: оптимизированы для операций агрегации и аналитических запросов, которые читают большие объемы данных из определенных столбцов по всей таблице.
 - хранение данных: данные хранятся по столбцам, что позволяет выполнять быстрые агрегации и сжатие данных, так как столбцы содержат данные одного типа.
 - примеры: Apache ClickHouse, Vertica, SAP HANA.
- Wide-column базы данных:
 - основное назначение: предназначены для масштабируемости и гибкости в приложениях, требующих обработки больших объемов данных с различной структурой.
 - хранение данных: данные организованы в таблицах, где каждая строка может иметь различное количество столбцов и структуру. Столбцы группируются в семейства столбцов, которые хранятся вместе, что улучшает производительность чтения и записи.
 - примеры: Apache Cassandra, Google Bigtable, Apache HBase.

Отличия Column-oriented от Wide-column

- **Структура хранения:** В column-oriented базах данных структура фиксирована и однородна, каждая запись или строка содержит данные для каждого столбца. В wide-column базах данных каждая строка может иметь различный набор столбцов, что придает дополнительную гибкость в управлении данными.
- **Семейства столбцов:** Wide-column базы данных используют понятие семейств столбцов, где столбцы, логически связанные между собой, хранятся вместе. Это отличает их от column-oriented баз данных, где столбцы обычно не группируются таким образом.

NoSQL: Key-Value Stores / обзор

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

По данные ресурса [DB-Engines](#) от австрийской консалтинговой компании Solid IT

NoSQL: Key-Value Stores / Характеристики

















Плюсы:

- Наиболее простой вид из прочих. Принцип работы аналогичен ассоциативному массиву на хэш-таблицах
- Отсутствует какая-либо схема данных, value может быть любой объект, в качестве ключа может выступать не единичное значение, а массив

Минусы:

- За счет необходимости просмотра всей хэш-таблицы при чтении может не очень хорошо масштабироваться
- Подходят только для очень простых данных – реализовать связи за счет самой модели невозможно
- Сложности с построением сложных запросов

NoSQL: Wide-column DB / Обзор

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 / Характеристики















Плюсы:

- Гибкость в физическом хранении данных – строки таблиц распределяются по разным серверам, в строке могут быть колонки разных типов данных
- Быстрота обработки запросов, связанных с извлечением данных из колонки
- Хорошее сжатие (за счет сериализации колонок, т.е. массивов, хранящих однотипные данные)

Минусы:

- Долгая (в сравнении с РСУБД) обработка запросов, извлекающих строки целиком. Долгая обработка JOIN-запросов и т.п. – где требуется построчное считывание
- Долгая (в сравнении с РСУБД) запись данных в строку
- (почти полная) невозможность создания композитных индексов

NoSQL: Document DB / Обзор

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 / Характеристики





















Плюсы:

- Нет поддержки схемы данных
- Удобный формат (MongoDB – JSON, один из стандартов для web)
- Быстрые запись/чтение
- Удобные репликация и шардирование

Минусы:

- Нет поддержки схемы данных!!!
- Плохая поддержка сложных SQL-подобных запросов

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 / Характеристики

Плюсы:

- В основе - проработанная математическая модель
- Удобно строить специфические модели предметной области – базы знаний
- Удобное расширение модели данных
- Быстрота обработки запросов за счет специфики модели данных

Минусы:

- Разнородные языки запросов – нет стандарта по отрасли
- Не поддерживают абстракцию транзакций
- Сложно обсчитывать запросы с агрегированием

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: достоинства

- Позволяют хранить большие объемы слабоструктурированной информации в более-менее связном виде
- Удобство масштабирования (в сравнении с РБД и РСУБД)
- Удобство репликации и шардирования
- Быстрота разработки

NoSQL: недостатки

- Не требуется проектирование схемы предметной области
- Поощряют накопление несогласованной информации в БД
- Для сложных предметных областей: схема данных стремится к реляционной. При этом преимущества реляционной модели остаются недоступными
- Согласованность в конечном счёте и BASE-принцип вносят риски ошибок

И напоследок... Векторные базы данных

- По сути своей – это не какой-то новый тип баз данных, а нечто вроде надстройки над уже существующими системами хранения. В качестве «backend'а» в таких БД могут использоваться key-value БД
- Цель – оптимизировать хранение и обработку vector embeddings
- Концептуально включают в себя три части: data store, vector indexing mechanism, query mechanism
- Индексация и запросы обрабатываются с использованием специальных библиотек, не относящихся к технологии баз данных. Стандартный вариант – FAISS (Facebook AI Similarity Search). Важное отличие от результатов запроса в иных типах БД – корректность результата носит **вероятностный** характер

Векторные базы данных: примеры

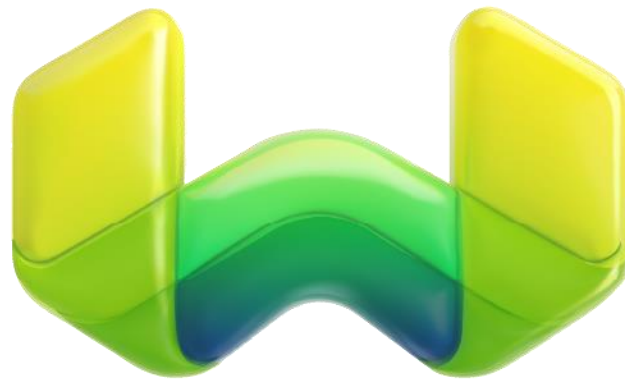
- Milvus



- Pinecone



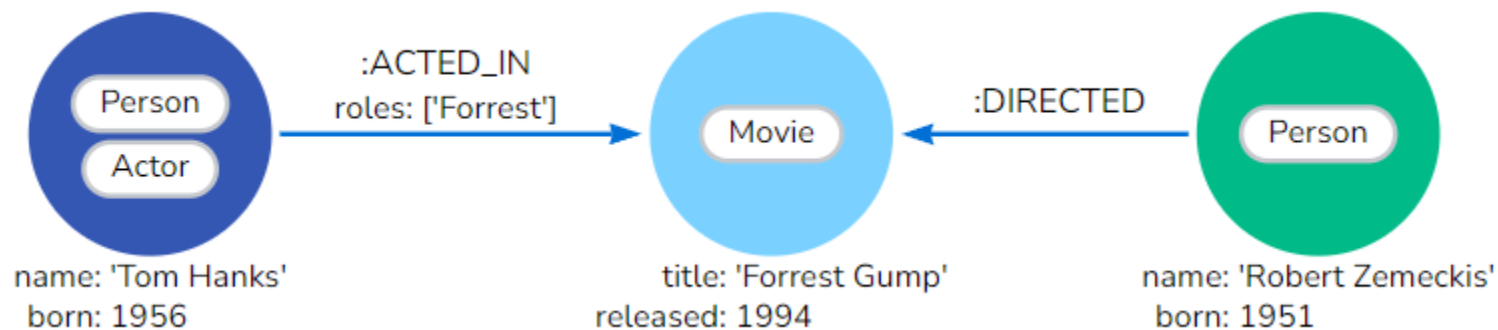
- Weaviate



II. Общее ЗНАКОМСТВО С Neo4j

Neo4j: модель данных

- Данные и отношения между ними представлены в виде 3 объектов модели:
 - узлов (nodes)
 - отношений (relationships)
 - свойств (properties).



Neo4j: Nodes

- Узлы представляют сущности или объекты в графе. Они похожи на записи или объекты в традиционной базе данных.
- Каждый узел может иметь одну или несколько меток, которые определяют его роль или тип в графе.
- Метки помогают эффективно организовывать и запрашивать данные.
- Узлы могут иметь свойства, которые представляют собой пары ключ-значение, предоставляющие дополнительную информацию об узле. Свойства могут быть проиндексированы для более быстрого поиска.
- Узлы могут иметь метки (labels)
- Пример синтаксиса создания узла:

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

Neo4j: Properties

- Свойства - это пары ключ-значение, которые используются для хранения данных об узлах и отношениях.
- Значение свойства может:
 - содержать различные типы данных, такие как число, строка или булево значение.
 - содержать список (массив), состоящий, например, из строк, чисел или булевых значения (компоненты должны принадлежать к одному типу).
- Пример свойств со значениями разных типов:

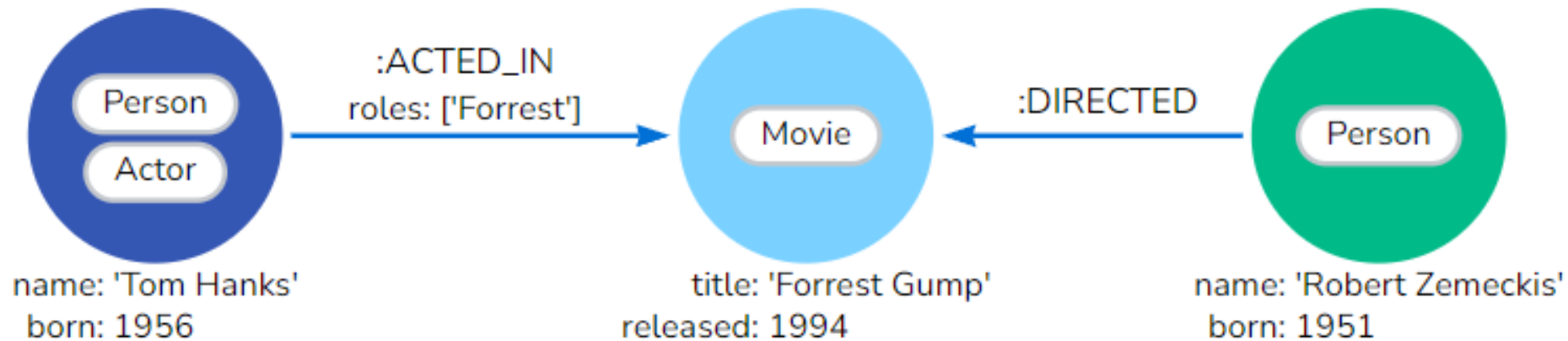
```
CREATE (:Example{a: 1, c: 'This is an example  
string', b: 3.14}, f: [1, 2, 3])
```

Neo4j: Relationships

- Отношение описывает, как связаны между собой исходный узел и целевой узел. Узел может иметь связь с самим собой.
- Отношения всегда направлены и имеют начальный узел и конечный узел. Они представляют собой отношения между двумя узлами и могут иметь определенный тип или метку (label).
- Как и узлы, отношения могут иметь свойства (properties), которые предоставляют дополнительную информацию об отношениях.
- Синтаксис создания отношения:

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

Neo4j: пример создания элементарной БД



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

Neo4j: некоторые аспекты языка CQL (Cypher Query Language)

- Используются ключевые слова, похожие на SQL: MATCH, WHERE, CREATE, DELETE, RETURN - и другие
- Пример pattern matching:

```
MATCH (me)-[:KNOWS*1..2]-(remote_friend)
WHERE me.name = 'Filipa'
RETURN remote_friend.name
```
- Допустимо использовать агрегирующие функции, например: COUNT, SUM, MIN, MAX
- Допустимо использовать стандартные операторы сравнения и логические операторы: >, <, =, AND, OR, NOT:

```
MATCH (node:Label)
WHERE node.property > 10 RETURN node
ORDER BY node.property
```
- Помимо CREATE для определения данных используются такие конструкции, как SET, DELETE, MERGE. Пример использования MERGE:
 - MERGE (m:Movie {title: 'The Matrix', year: 1999})
 - Используется для исключения дублирования данных (в примере выше узел не будет создан, если в БД уже есть узел с аналогичными свойствами)

Neo4j: дополнительные возможности

- Допускается создание индексов:

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

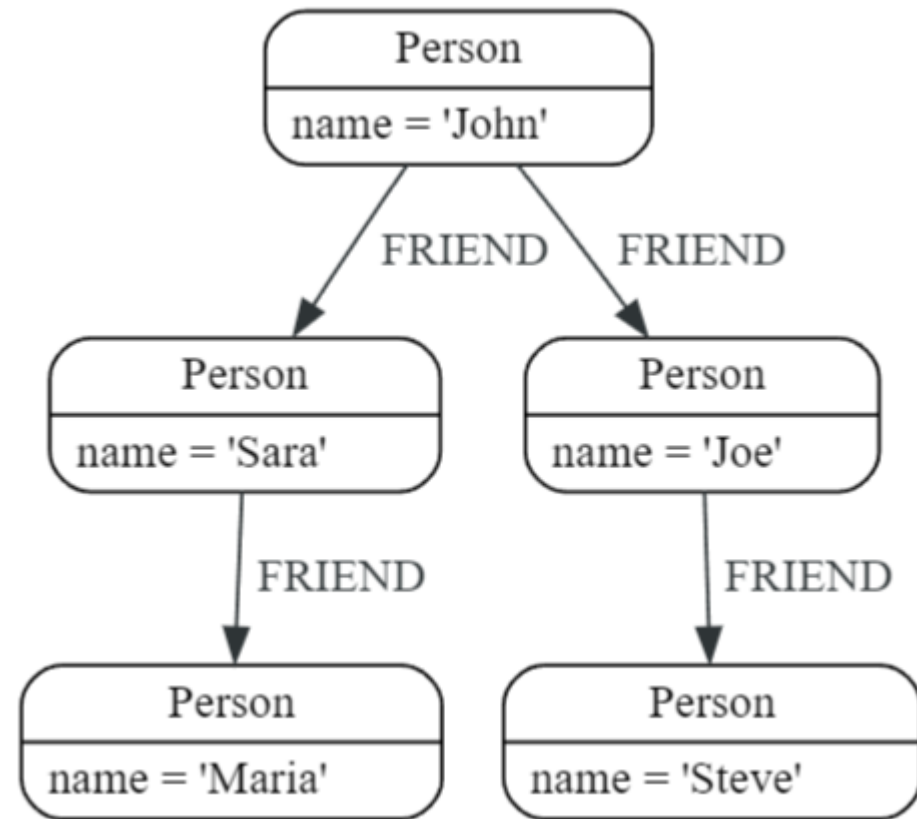
- Допускается наложение ограничений на

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

- Допускается создание пользовательских функций

Neo4j: пример работы с запросами

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



Как отработает запрос из предыдущего примера

```
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'

Еще пример запроса

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

- запрос находит все узлы, у которых свойство name начинается с "J", и для каждого такого узла создает другой узел со свойством name, установленным в "Jay-jay".

Clause	Table of intermediate results after the clause	State of the graph after the clause, changes in red						
<pre>MATCH (j:Person) WHERE j.name STARTS WITH "J"</pre>	<table><tr><td>j</td></tr><tr><td>{{name: 'John'}}</td></tr><tr><td>{{name: 'Joe'}}</td></tr></table>	j	{{name: 'John'}}	{{name: 'Joe'}}	<pre>graph TD John[Person name = 'John'] -- FRIEND --> Sara[Person name = 'Sara'] John -- FRIEND --> Joe[Person name = 'Joe'] Sara -- FRIEND --> Maria[Person name = 'Maria'] Joe -- FRIEND --> Steve[Person name = 'Steve']</pre>			
j								
{{name: 'John'}}								
{{name: 'Joe'}}								
<pre>CREATE (j)-[:FRIEND]-> (jj:Person {name: "Jay-jay"})</pre>	<table><tr><td>j</td><td>jj</td></tr><tr><td>{{name: 'John'}}</td><td>{{name: 'Jay-jay'}}</td></tr><tr><td>{{name: 'Joe'}}</td><td>{{name: 'Jay-jay'}}</td></tr></table>	j	jj	{{name: 'John'}}	{{name: 'Jay-jay'}}	{{name: 'Joe'}}	{{name: 'Jay-jay'}}	<pre>graph TD John[Person name = 'John'] -- FRIEND --> Sara[Person name = 'Sara'] John -- FRIEND --> Joe[Person name = 'Joe'] John -- FRIEND --> JayJay1[Person name = 'Jay-jay'] Sara -- FRIEND --> Maria[Person name = 'Maria'] Joe -- FRIEND --> Steve[Person name = 'Steve'] Joe -- FRIEND --> JayJay2[Person name = 'Jay-jay']</pre>
j	jj							
{{name: 'John'}}	{{name: 'Jay-jay'}}							
{{name: 'Joe'}}	{{name: 'Jay-jay'}}							

- Рекомендации по интерпретации реляционных данных в графовые с из официального руководства 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*.

Что почитать?

- Официальная документация:

<https://neo4j.com/docs/>

- Google...