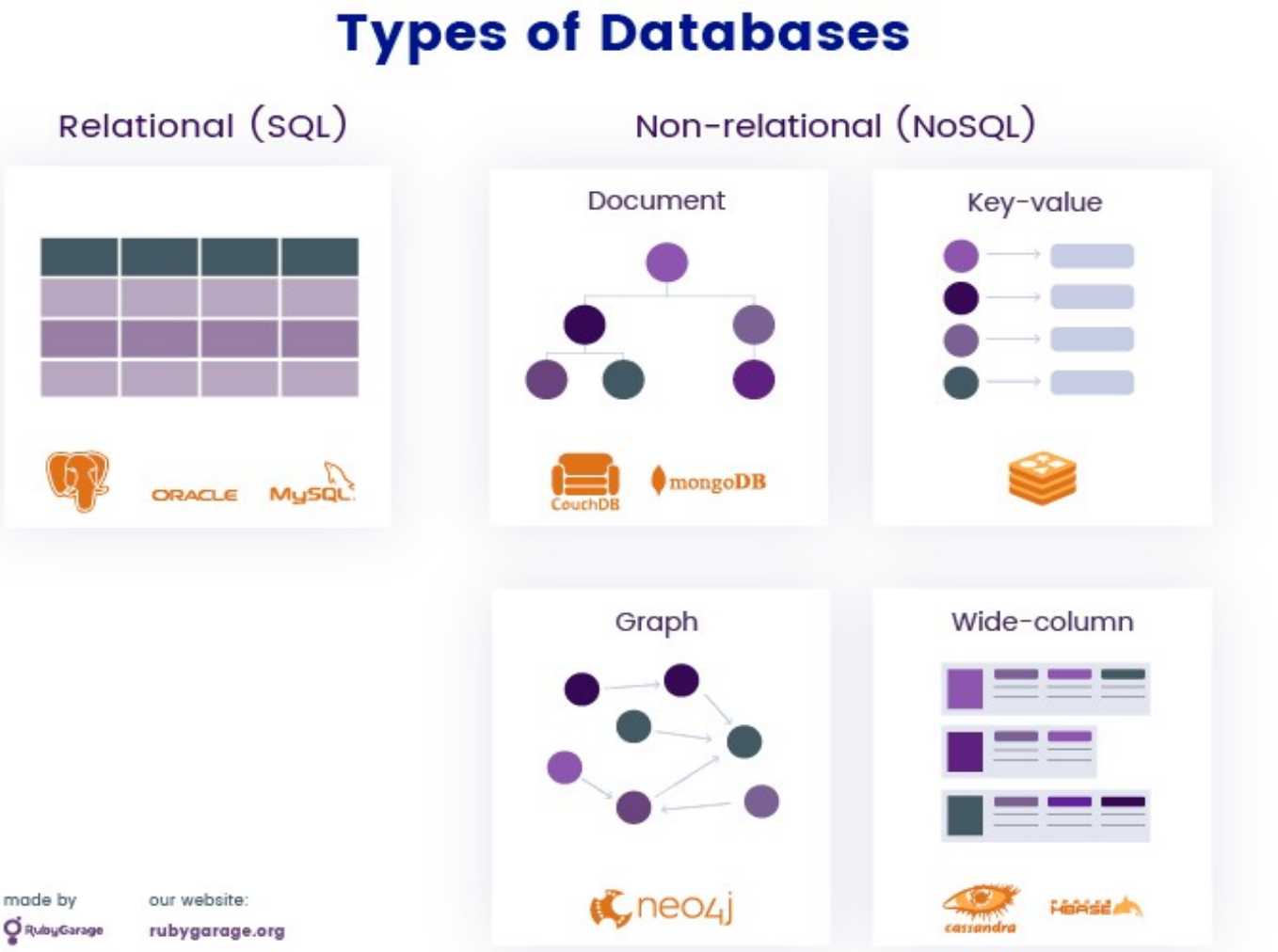
پایگاه داده و داده‌ها سنگین

انواع پایگاه داده:



محدودیت‌های SQL در چیست؟

به دلیل رشد علم و فناوری با سرعت بالا و قدیمی بودن SQL شاید در گذشته بهترین سیستم برای مدیریت اطلاعات به‌شمار می‌رفت ولی به دلیل دو محدودیت بزرگ مجبوریم این روزها آن را با سیستمی نوین (مانند NoSQL) عوض کنیم. این دو محدودیت عبارتند از:

- **مقیاس‌پذیری یا Scalability**: کاربر در سیستم SQL مجبور است داده‌های خود را فقط به صورت Relational در قالب کلاسیک و گفته شده ذخیره کند و این برای هر نوع خواسته‌ای از طرف کاربر قابل مقیاس بندی نیست. در مواردی بنا به تجربه دیده شده مجبور به ایجاد داده‌های بی‌مورد یا موارد مشابه از طرف برنامه‌نویس شده یا برای قالب بندی کردن و طراحی شمای پایگاه داده مورد نظر مدت‌ها باید توسط کارشناسان طرحهایی پیاده‌سازی و اجرا گردد که این خود یک محدودیت بزرگ برای دنیای داده‌های بزرگ امروزیست که تصور کنید با این هزینه سرورهای پرسرعت را بتوانیم پشت سر بگذاریم و برای طراحی داده‌ها در دو سرور نیازمند طراحی‌های وقت گیر هم باشیم!
- **پیچیدگی یا Complexity**: این پیچیدگی طراحی پایگاه داده‌های SQL بقدری می‌باشد که در برنامه‌نویسی فرد یا افراد جدای برنامه‌نویس را لازم دارد تا بر اساس داده‌های موجود بهترین نوع چیدمان جداول را طراحی نمایند که برای نیازهای مورد نظر قابل قبول باشد که توضیح این پیچیدگی در محدودیت قبلی و مقیاس بندی هم توضیح داده شده بود.

تاریخچه NOSQL :

کارلو استروزی (Carlo Strozzi) نخستین‌بار در سال ۱۹۹۸عبارت NoSQL را برای اشاره به پایگاه‌هاي داده‌اي سبک و اپن‌سورس رابطه‌اي به کار گرفت که از رابط SQL استفاده نمی‌کردند. هرچند بعدها وی به این نکته اشاره کرد که این عبارت و مفهوم پشت آن، کاملاً از مدل رابطه‌اي جدا شده و دیگر بهتر است آن را NoREL یا Not Only Relational بنامیم.

Document

نحوه کارکرد :

داده را بر اساس کلید و مقدار ذخیره می‌کنند ولی در مدل‌هایی مانند xml,json,bson که به عنوان Document شناخته می‌شوند.

کاربرد :

- فروشگاه آنلاین
- سیستم مدیریت محتوا
- پلتفرم تجزیه و تحلیل داده
- پلتفرم وبلاگی
- ذخیره داده با حجم بزرگ

مشکلات :

- برای کارهای با کوئری پیچیده مناسب نیست

نمونه‌ها :

- Mongodb
- Unqlite

Key-Value

نحوه کارکرد :

بر اساس کلید-مقدار کار می‌کند و داده را به صورت شی دودویی (binari Object) ذخیره می‌کند بنابراین تشخیص نوع داده بر عهده کاربر است. برای ایجاد کلید از hash استفاده می‌کند.

کاربردها :

- ذخیره داده‌های session لاگین کاربران
- مدیریت پروفایل‌های بدون ساختار کاربران
- ذخیره داده‌های سبد خرید فروشگاه

مشکلات :

- عدم امکان زدن کوئری بر اساس مقدار
- عدم امکان زدن کوئری رابطه‌ای معنادار بین داده‌ها
- عملیات روی چندین کلید منحصر به فرد
- برای داده‌هایی که نیاز به بروز رسانی دارد مناسب نیست

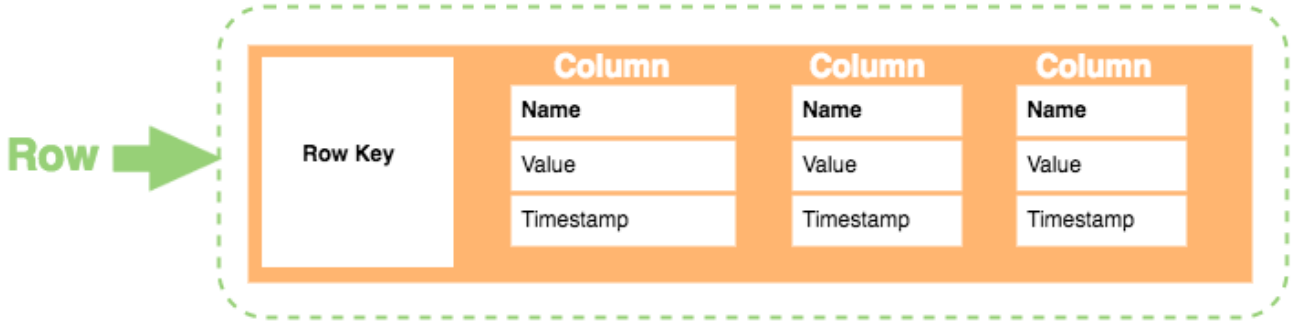
نمونه‌ها :

- Redis

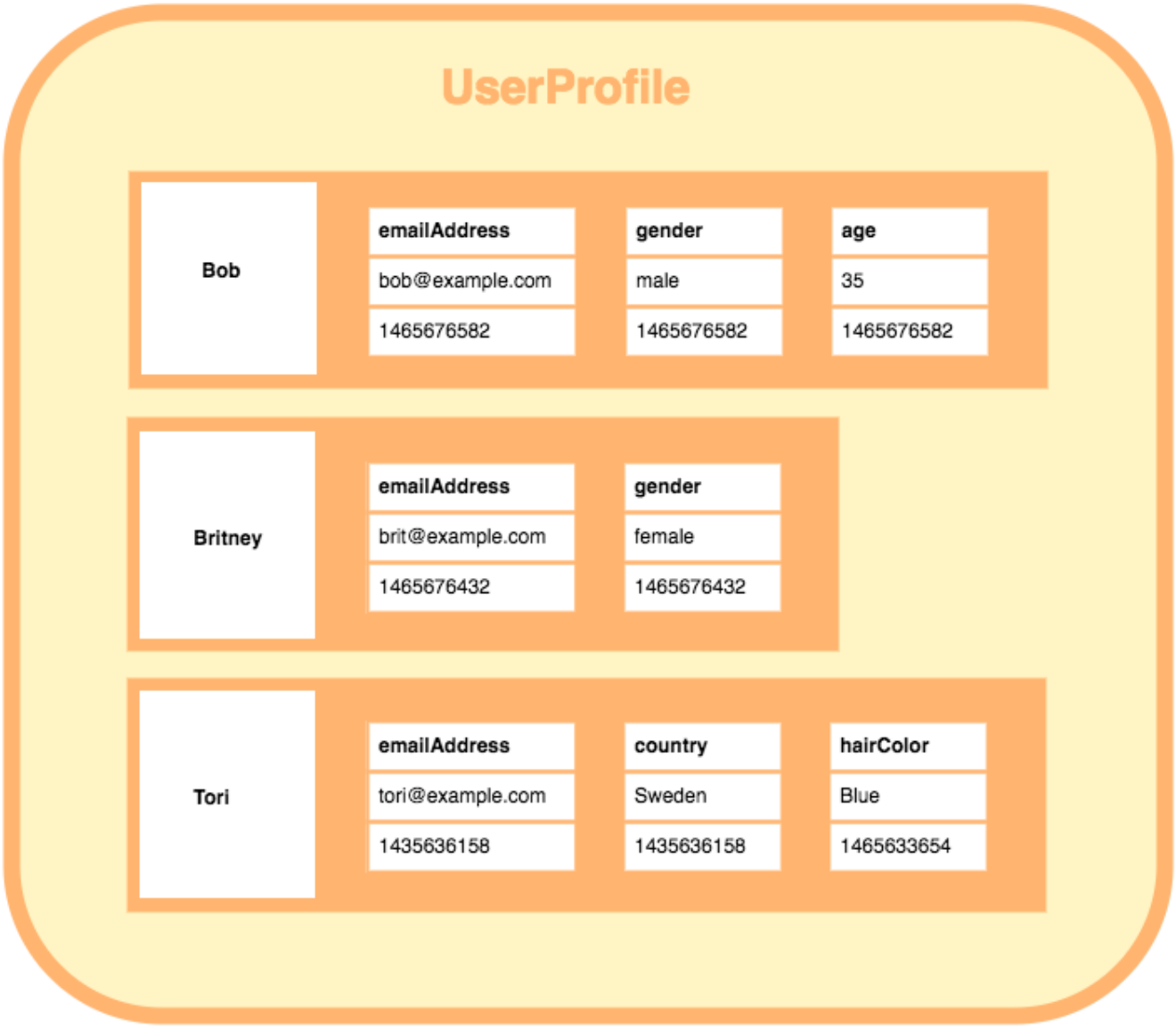
Column

نحوه عملکرد :

هر ردیف تعداد نامشخصی ستون دارد و نیاز نیست همه ستون‌ها در همه ردیف‌ها وجود داشته باشند و امکان کوئری زدن بر اساس ستون فراهم است. مزیت اصلی نسبت به **SQL** این است که چون دیتای هر ستون را میتوان به صورت یک دیتای مجزا ذخیره کرد پس توضیع پذیری افقی امکان پذیر است و بقیه مزیت‌های **NOSQL** نیز صدق می‌کند و همینطور نیاز نیست در زمان اضافه کردن یک ستون به یک ردیف دیتای اضافی و نامعتبر به بقیه ردیف‌ها اضافه شود.



مثال ذخیره داده پروفایل:



کاربردها :

- سیستم مدیریت محتوا
- پلتفرم وبلاگی
- سرویس ذخیره داده با تاریخ انقضا
- سیستم‌هایی که نیاز به ثبت داده سنگین دارند مانند ذخیره لاگ

مشکلات :

- برای وقتی که تعریف درستی از کارکرد دیتابیس نداریم مناسب نیست
- وقتی نیاز به کوئری پیچیده داریم یا روش کوئری زدن تغییر می‌کند نباید از این دیتابیس استفاده کنیم.

نمونه‌ها :

- Accumulo

Graph

نحوه عملکرد :

بر اساس Entity-attribute-value کار می‌کنندکه در آن‌ها موجودیت همان گره‌های گراف است که یک سری خصوصیت دارد و میتوان به صورت مجازی در لحظه هر رابطه‌ای را تعریف کرد و نیازی نیست در

ابتدای طراحی دیتابیس روابط ثابت و محدودی بین جداول داشته باشیم. در زمان کوئری زدن نیز در این نوع دیتابیس میتوان به سادگی کوئری‌هایی که در سیستم SQL و رابطه ای به شدت سخت و طاقت فرصا بود را در مدت کوتاهی ایجاد کرد. در این دیتابیس هم گره و هم رابطه میتواند خصوصیت داشته باشند.

کاربرد :

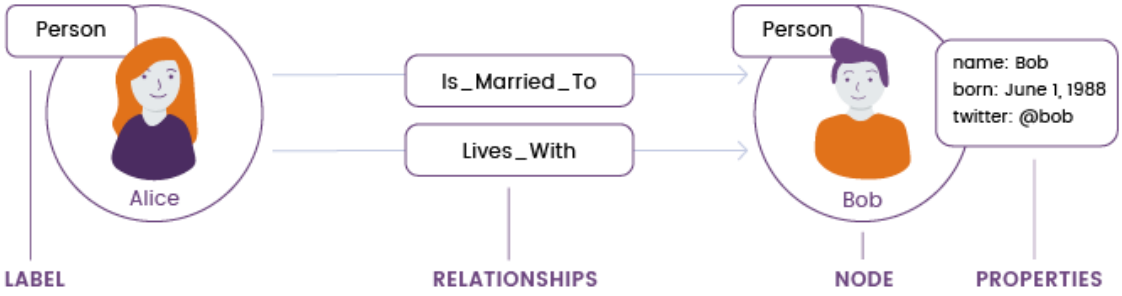
- سیستم شناسایی کلاهبرداری آنلاین
- جستجوی مبتنی بر گراف
- انجام تسک‌های مختلف در حوزه شبکه
- شبکه اجتماعی

مزیت‌ها نسبت به دیتابیس رابطه‌ای :

- **محدودیت ذخیره سازی** : دیتابیس‌های رابطه‌ای نمیتوانند میزان زیاد داده را هندل کنند
- **سرعت** : عملکرد دیتابیس‌های رابطه‌ای وقتی نیاز به میزان زیاد عملیات خواندن/نوشتن دارید زجرآور می‌شود.
- **کمبود روابط** : دیتابیس‌های رابطه‌ای نمی‌توانند رابطه‌هایی بجز رابطه‌های استاندارد یک به یک ، یک به چند و چند به چند را توصیف کنند.
- **تنوع** : پایگاه‌داده های رابطه‌ای در هنگام برخورد با انواع داده هایی که نمی توانند با استفاده از طرح پایگاه‌داده (database schema) توصیف شوند، انعطاف پذیری ندارند.
- **توسعه‌پذیری** : توسعه پذیری افقی برای دیتابیس‌های رابطه‌ای ناکارآمد است.

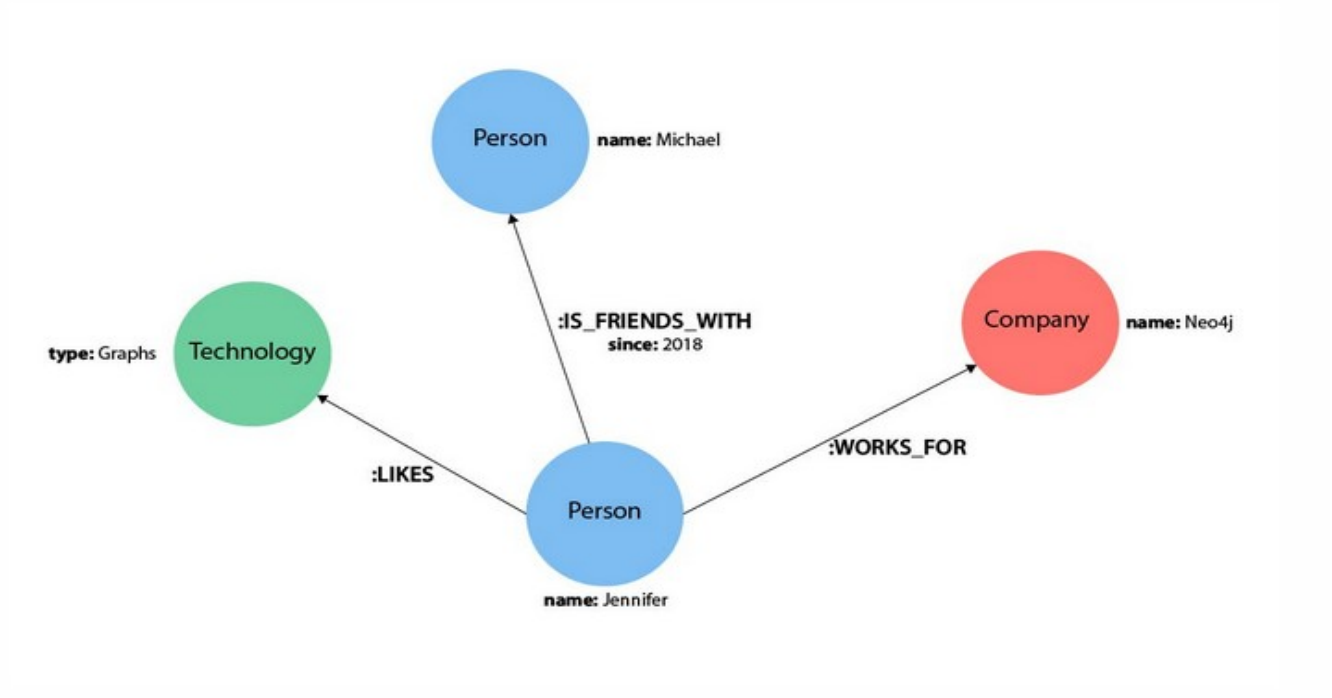
تعاریف اصلی :

- **Nodes** : (معادل vertices در نظریه گراف) : این‌ها دیتای اصلی هستند که از طریق روابط متصل هستند. یک گره می‌تواند یک یا چند برچسب(label) داشته باشد(که نقش آن را توصیف می‌کند) و خواص (ویژگی)
- **Relationship** : (معادل edge در نظریه گراف). یک رابطه دو گره را به هم متصل می‌کند که به نوبه خود می‌توانند چند رابطه داشته باشند. رابطه‌ها میتوانند یک یا چندخاصیت داشته باشند.
- **Label :** این‌ها برای دسته بندی گره‌ها استفاده می‌شوند و هر گره می‌تواند به چندین برچسب اختصاص داده شود.برچسب‌ها برای یافتن سریع گره در گراف ایندکس شده‌اند.
- **Properties** : این‌ها ویژگی‌های هردوی گره‌ها و رابطه‌ها هستند.



چند مثال :

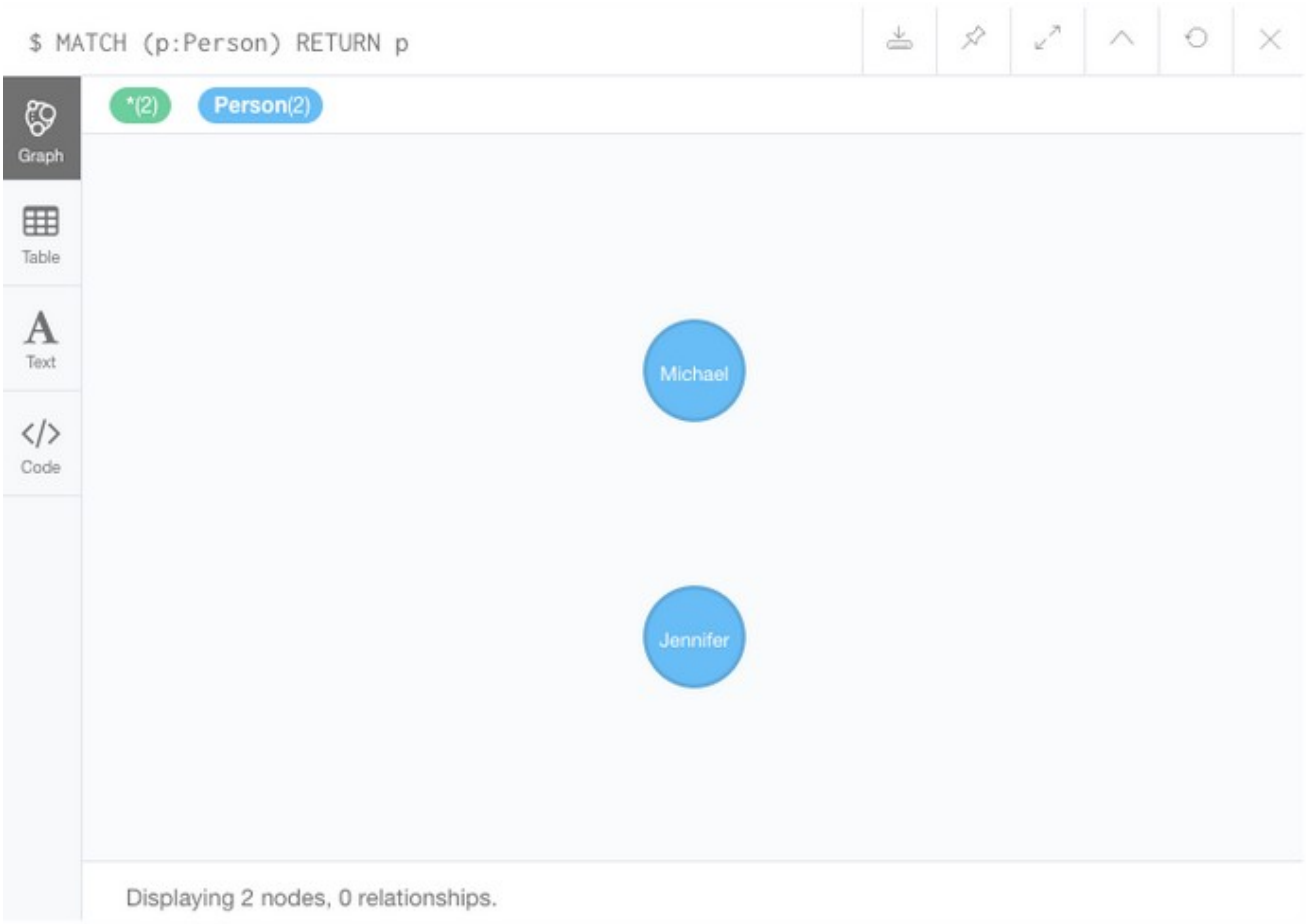
گراف :



مثال کوئری :

```
MATCH (p:Person)
RETURN p
```

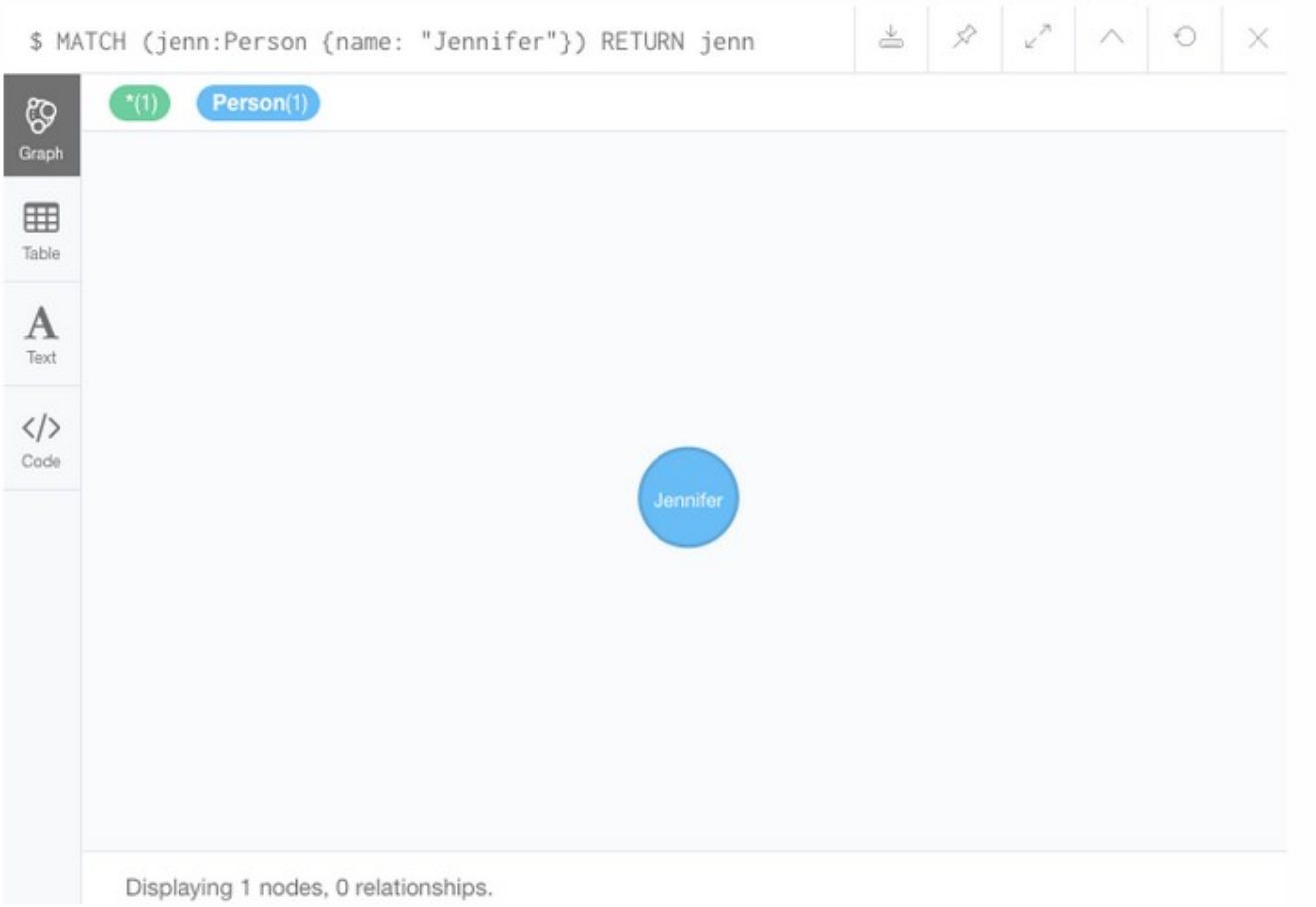
نتیجه :



کوئری :

```
MATCH (jenn:Person {name: 'Jennifer'})
RETURN jenn
```

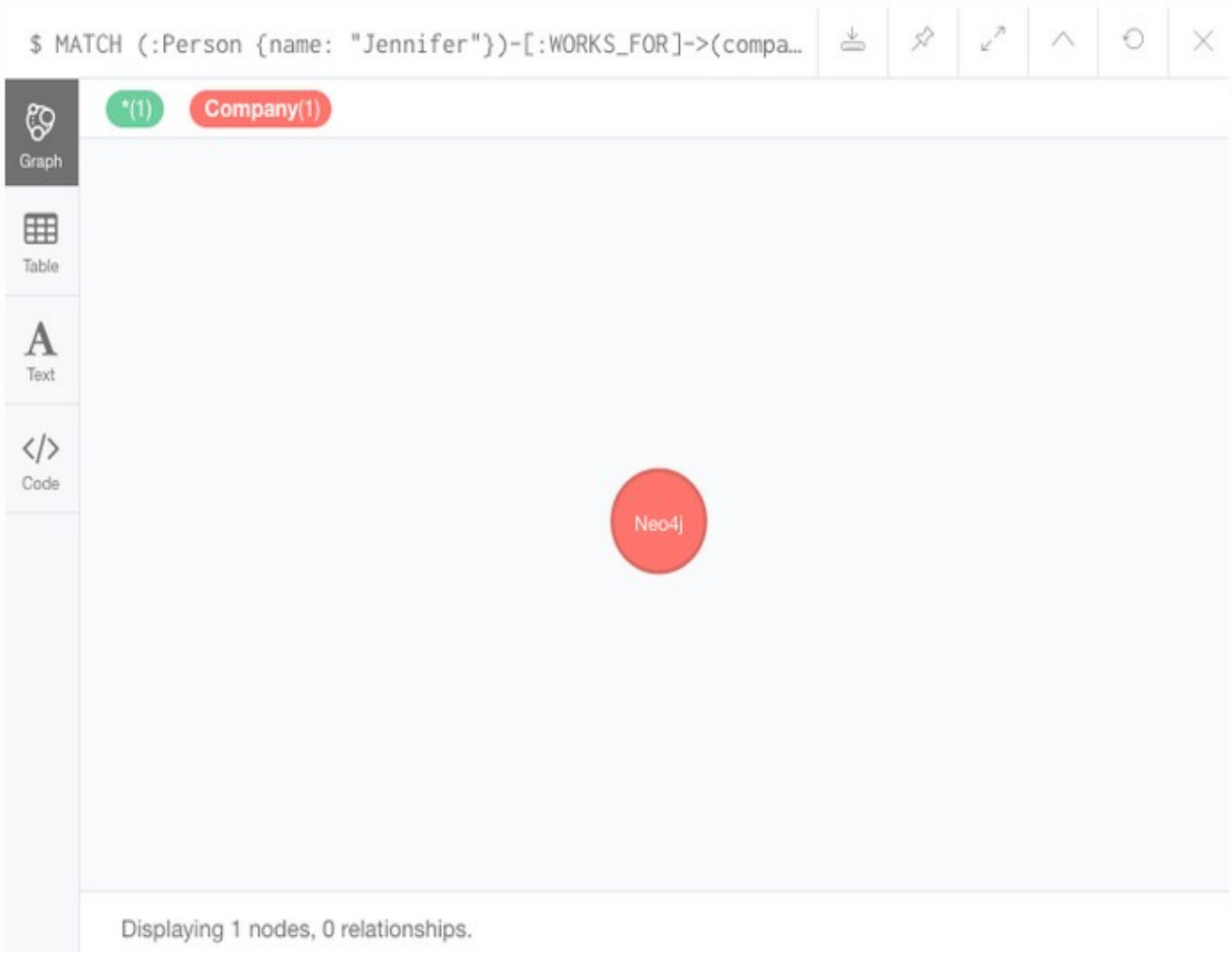
نتیجه :


$ MATCH (jenn:Person {name: "Jennifer"}) RETURN jenn
Graph
Table
Text
Code
*(1)
Person(1)
Jennifer
Displaying 1 nodes, 0 relationships.

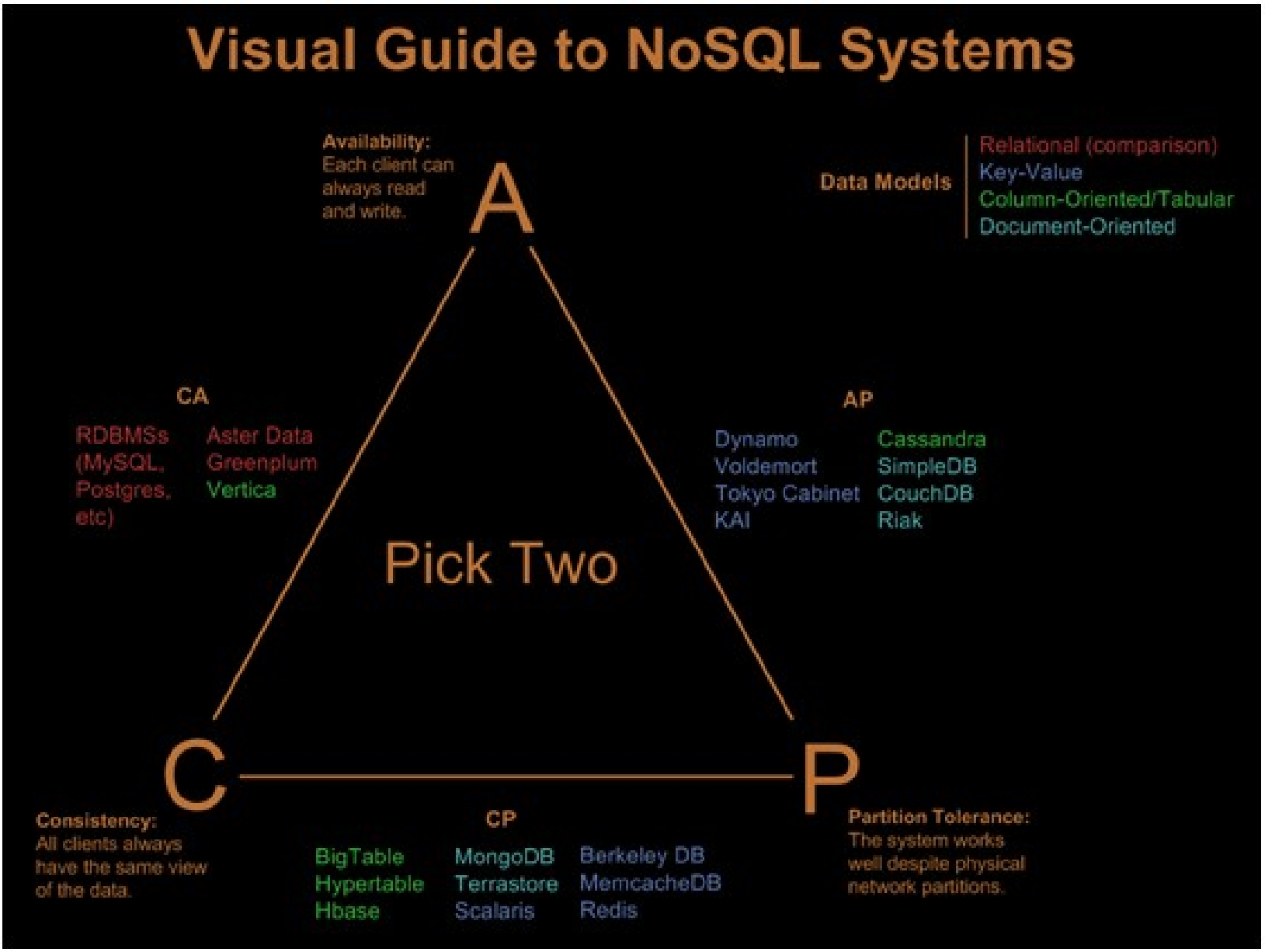
کوئری :

```
MATCH (:Person {name: 'Jennifer'})-[:WORKS_FOR]->(company:Company
RETURN company
```

نتیجه :



Visual Guide to NoSQL Systems



So let's briefly compare Neo4j to other relational and non-relational databases:

	Neo4j	Relational databases	NoSQL databases
Data storage	Graph storage structure	Fixed, predefined tables with rows and columns	Connected data not supported at the database level
Data modeling	Flexible data model	Database model must be developed from a logical model	Not suitable for enterprise architectures
Query performance	Great performance regardless of number and depth of connections	Data processing speed slows with growing number of joins	Relationships must be created at the application level
Query language	**Cypher**: native graph query language	**SQL**: complexity grows as the number of joins increases	Different languages are used but none is tailored to express relationships
Transaction support	Retains ACID transactions	ACID transaction support	BASE transactions prove unreliable for data relationships
Processing at scale	Inherently scalable for pattern-based queries	Scales through replication, but it's costly	Scalable, but data integrity isn't trustworthy

منابع :

http://sokanacademy.com

http://bigdata.ir

https://rubygarage.org