

# Computação na nuvem

## ISEL – LEIRT / LEIC / LEIM

### Armazenamento em Bases de Dados NoSQL

- Do modelo relacional aos modelos distribuídos
- Tipos de Bases de Dados NoSQL
- Exemplos com Google *Firestore* e API Java (slides 7.2)

José Simão [jsimao@cc.isel.ipl.pt](mailto:jsimao@cc.isel.ipl.pt) ; [jose.simao@isel.pt](mailto:jose.simao@isel.pt)

Luís Assunção [lass@isel.ipl.pt](mailto:lass@isel.ipl.pt) ; [luis.assuncao@isel.pt](mailto:luis.assuncao@isel.pt)

# Modelo Relacional - Inquestionável !

---

- Durante as últimas décadas existiram mudanças significativas nas tecnologias e linguagens de desenvolvimento de aplicações;
- No entanto, para armazenar dados era inquestionável usar bases de dados relacionais;
- Ao definir a arquitetura de uma aplicação a única questão que se colocava era decidir:
  - Qual a tecnologia a usar?  
Oracle? SQLserver? Postgres? MySQL? SQLite? etc.

# Modelo Relacional - Aspectos relevantes

---

- **Persistência de dados:** Capacidade de persistir grandes quantidades de dados, incomportáveis de manter em memória
- **Concorrência:** Permitir que múltiplos utilizadores acedam e alterem os dados sem conflitos (ex. evitar duas reservas do mesmo quarto)
- **Transações:** Permitir que uma sequência de ações, incluindo alterações de dados, sejam executadas atomicamente;
- **Integração:** Permitir que aplicações em diferentes tecnologias possam partilhar (aceder e alterar) dados num único repositório;
- **Standards:** Com a álgebra relacional e a linguagem standard SQL é possível usar BDs relacionais em múltiplos cenários aplicacionais;
- **Impedance Mismatch:** Limitações de representação no modelo relacional (*Relation, Tuple, Attributes*) de modelos de informação em memória onde podemos ter representações agregadas, de dados, mais complexas.
  - Os frameworks **Object-relational mapping (ORM)** resolvem parte do problema mas introduzem perdas de desempenho e podem introduzir falhas de consistência (má utilização: *cache* de dados nos objetos).

# Modelo Relacional - *Impedance mismatch*

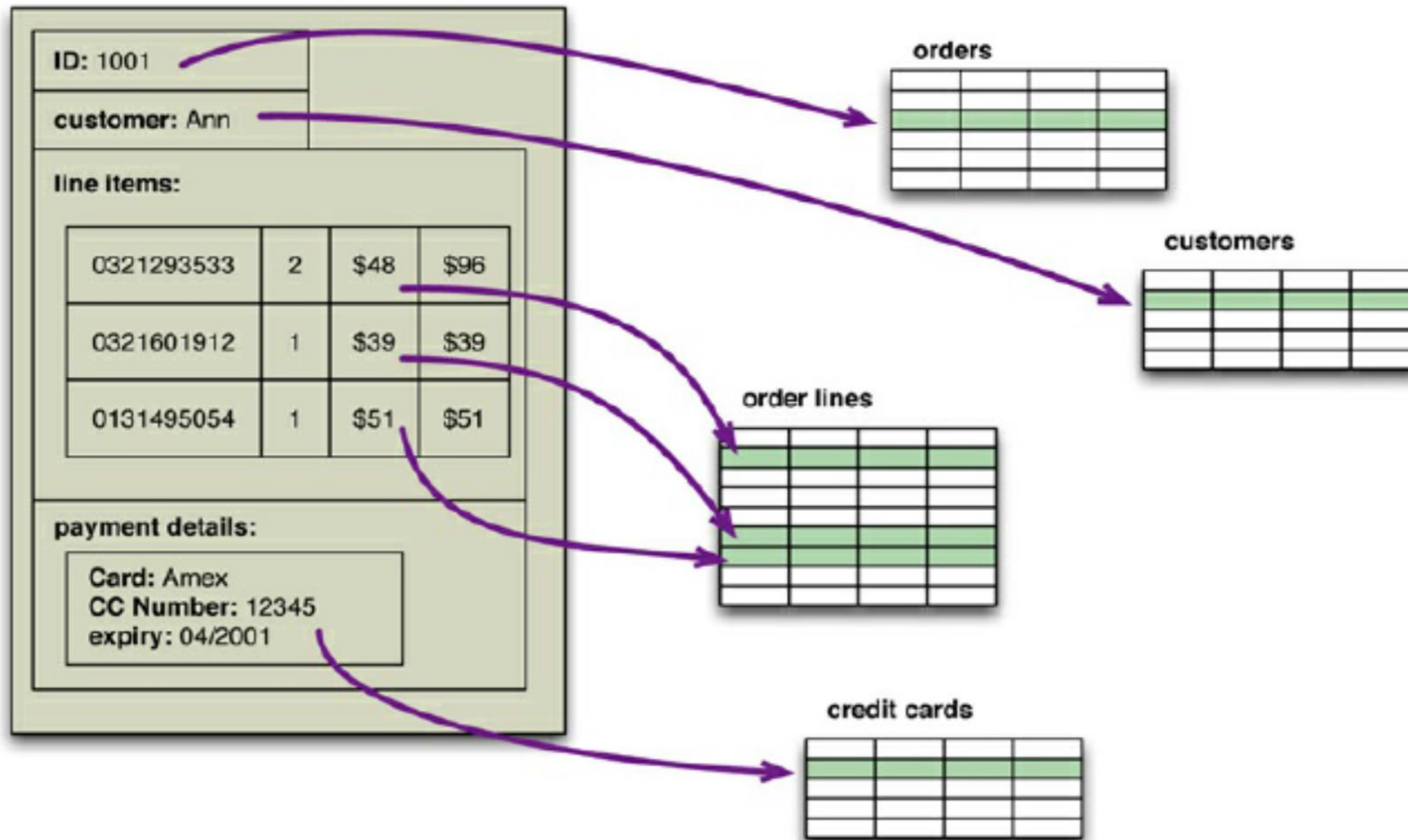


Figure 1.1. An order, which looks like a single aggregate structure in the UI, is split into many rows from many tables in a relational database

Extraído de: *NoSQL Distilled, A Brief Guide to the Emerging World of Polyglot Persistence*, Pramod J. Sadalage, Martin Fowler, Addison-Wesley.

# Modelo Relacional e *Clusters* de computadores

- O aparecimento de *clusters* de computadores possibilita a (escalabilidade horizontal (*scale-out*) versus escalabilidade vertical (*scale-up*))
- Num cenário (*scale-up*) as maiores exigências de processamento implicam computadores de custo elevado existindo um limite de crescimento;
- A alternativa é usar (*scale-out*) com uma grande quantidade de pequenos computadores ligados em *cluster*, permitindo também maior resiliência a falhas;
- No entanto, o modelo relacional não é apropriado para trabalhar em *clusters*, ou quando trabalham (*Oracle RAC, Microsoft SQL Server, etc.*) assumem a partilha de subsistemas de discos (pontos únicos de falha) e têm custos de licenciamento altos, baseados no número de nós do *cluster*.

As limitações entre o modelo relacional e a existência de *clusters* de grandes dimensões levaram a novas iniciativas para armazenar dados. Em particular, empresas como a Google e Amazon que criaram e influenciaram modelos alternativos.

# Armazenamento em Clusters de computadores

O armazenamento de dados distribuídos em larga escala é conseguido com o recurso a *clusters* de grande dimensão, garantindo:

- **Escalabilidade elevada:** milhares de máquinas
- **Custos reduzidos:** o custo de 10 máquinas de menor capacidade é inferior a uma máquina com capacidade idêntica
- **Flexibilidade:** Os dados não têm de ser estruturados segundo um *schema* como no modelo relacional, permitindo formas flexíveis de agregar os dados (*key-value, document, column, graph*)
- **Disponibilidade:** Replicação intensiva garantindo menores *bottlenecks* e suporte para tolerância a falhas

**Mas, não esquecer o teorema CAP**  
(impossibilidade de ter 100% de *Consistency, Availability, Partitions*)

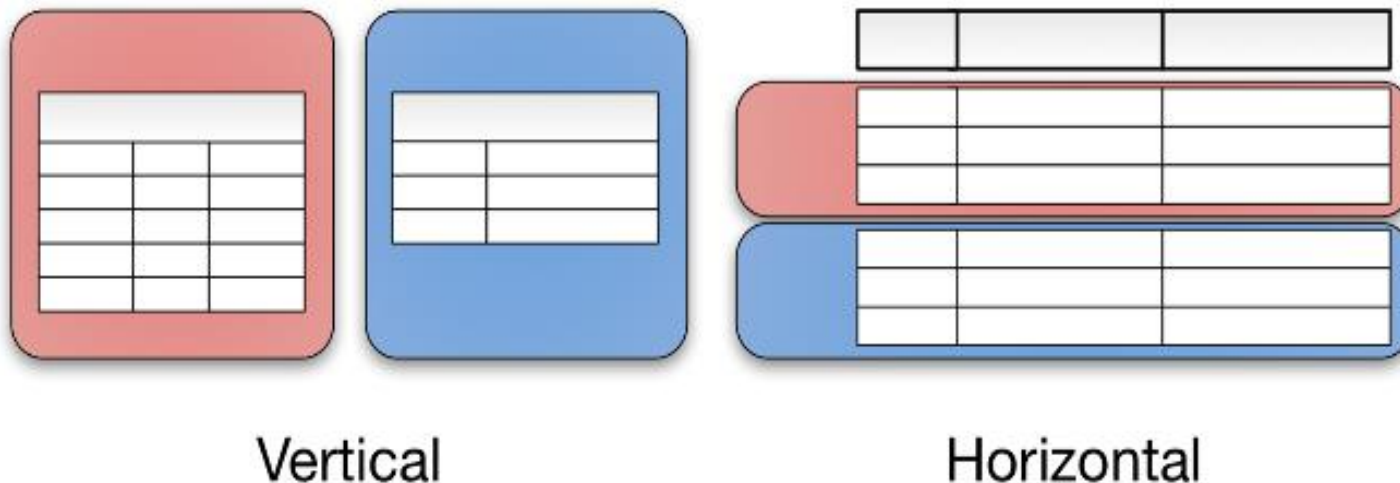
# Distribuição de dados

---

- **Sharding:** Fragmentação de dados em múltiplos servidores de um cluster, em que cada servidor contém um subconjunto dos dados. Por exemplo, as colunas ou linhas de uma tabela estão fragmentadas em múltiplos servidores
  - **Replication:** Cópias dos dados existem em múltiplos servidores, permitindo múltiplos acessos aos mesmos dados em diferentes servidores
    - ❖ A replicação de dados pode ser feita:
      - **Master-slave:** Um nó *master* gere as escritas, enquanto os nós *slave* só suportam leituras sincronizando-se com o *master*
      - **Peer-to-peer:** Escritas em qualquer nó, necessitando de coordenação para evitar conflitos na atualização das várias réplicas
      - A replicação *Master-slave* reduz os conflitos de atualização dos dados mas, por outro lado, a replicação *peer-to-peer* evita a sobrecarga de escritas num único ponto de falha.
- Alguns sistemas de armazenamento de dados usam as duas técnicas em simultâneo

# *Sharding* no modelo relacional

- Nos nós de um cluster, a distribuição de dados pode ser realizada em dois modos:
  - Vertical:** Armazenar tabelas e colunas em diferentes nós
  - Horizontal:** Armazenar linhas da mesma tabela em múltiplos nós





# O que significa *NoSQL*?

- Não existe uma definição clara: *Not SQL*; *Not only SQL*, etc.
- Em 1998, **Carlo Strozzi** usou o termo NoSQL para designar uma base de dados relacional *open-source* leve e sem suporte da linguagem SQL
- Em 2009, **Johan Oskarsson**, reintroduziu o termo quando organizou um evento para discutir "*open source distributed, non relational databases*" quando se assistia ao crescimento de sistemas de armazenamento distribuídos, por ex., *Google's Bigtable* e *Amazon DynamoDB*, que não suportavam as propriedades transacionais ACID (Atomicidade, Consistência, Isolamento, Durabilidade) do modelo relacional

NoSQL

**Bases de Dados *Schemaless*: Os dados são armazenados como coleções de agregados, de acordo com os requisitos da aplicação**

# Características nas Bases de Dados *NoSQL*

---

- Não usam o modelo relacional
- Suportam execução em clusters de múltiplos computadores
- Múltiplas soluções *open-source*
- ***Schemaless***: Não existe um *schema* rígido como no modelo relacional
- Flexibiliza a persistência de dados (*Polyglot Persistence*\*)

\* *Polyglot Persistence*: Utilização híbrida de bases de dados especializadas (NoSQL e Relacionais) consoante os diferentes requisitos da aplicação

# ***BASE: Basically Available, Soft State, Eventually Consistent***

---

***BA (Basically Available):*** podem existir falhas parciais nalgumas partes do sistema distribuído, continuando o resto do sistema a funcionar, principalmente na presença de réplicas

***S (Soft state):*** Os dados podem estar desatualizados (expirados), sendo necessário atualizá-los com processamento posterior

***E (Eventually consistent):*** Os dados podem estar inconsistentes durante alguns intervalos de tempo. Por exemplo, na presença de múltiplas réplicas, algumas podem estar inconsistentes durante algum tempo. O tempo para atualizar as réplicas depende da carga do sistema, da velocidade de comunicação na rede e da latência

# Aggregate Data Models

---

- Um modelo de dados define como percebemos e manipulamos os dados na perspetiva da aplicação
- Um modelo de armazenamento de dados descreve como a base de dados armazena e manipula os dados internamente
- Os sistemas NoSQL suportam modelos de dados orientados a agregados de dados (*aggregate orientation*) que podem ser classificados em 4 categorias:
  - *key-value*
  - *document*
  - *column-family*
  - *graph*

# Tipos de Bases de Dados NoSQL

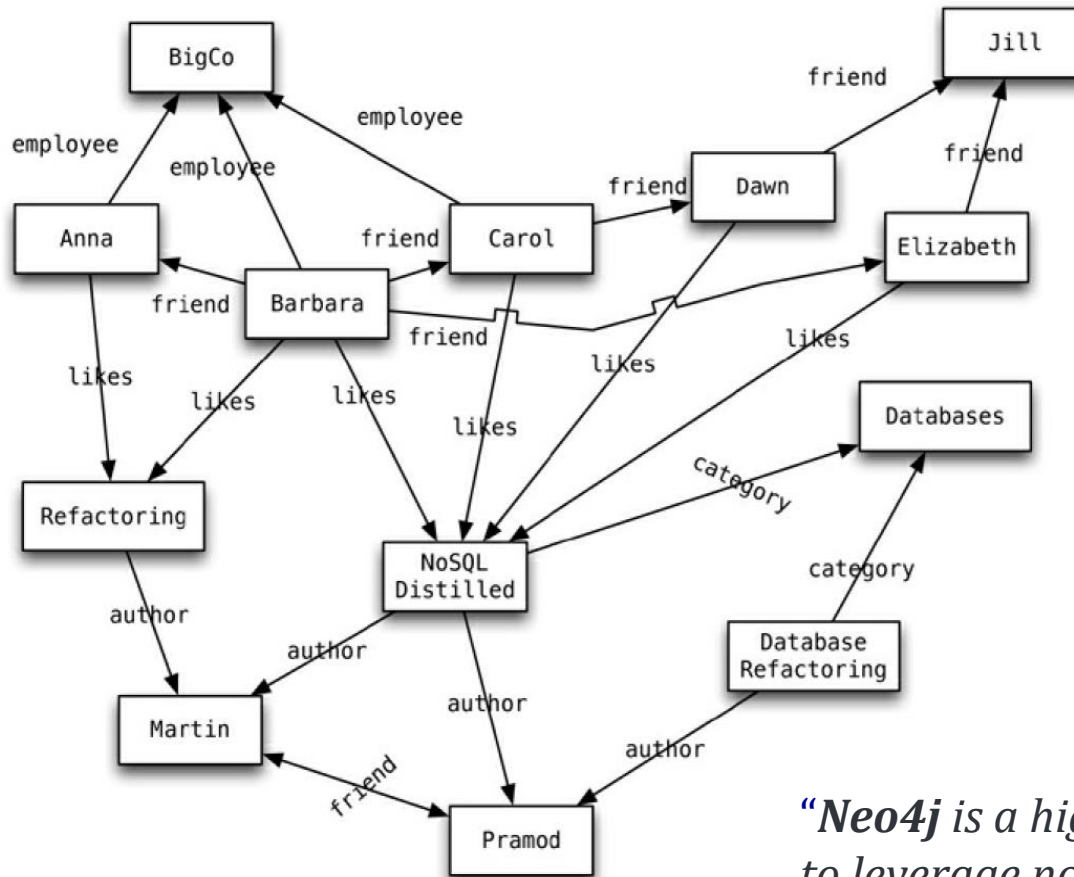
Existem 4 tipos (*Key-value; Document; Column-family; Graph*)

- Todos partilham a noção que existe uma *key* usada para obter (*lookup*) um agregado de dados.
- **Key-Value:** O agregado é opaco, permitindo unicamente obter o total do agregado através de uma *key*
- **Document:** permite *queries* baseadas na estrutura interna do agregado (documento)
- **Column-family:** Permite estruturar o agregado em famílias de colunas permitindo à base de dados usar estratégias de *sharding* das diferentes famílias de colunas.

Key	Column family 1				...	Column family N			
	c1	c2	...	cn		c1	c2	...	cn
	val11	val12	...	val1n	...	valN1	valN2	...	valNn

# Graph databases

- Organizam os dados em grafos (nós e arcos) permitindo associações complexas entre entidades



**NoSQL Distilled.** A Brief Guide to the Emerging World of Polyglot Persistence. Pramod J. Sadalage. Martin Fowler

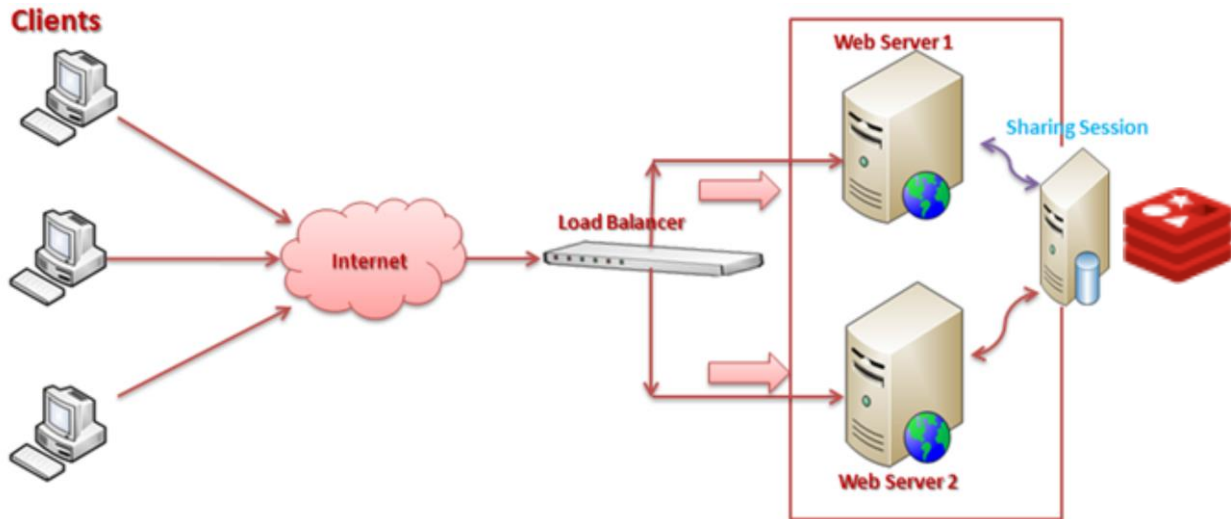
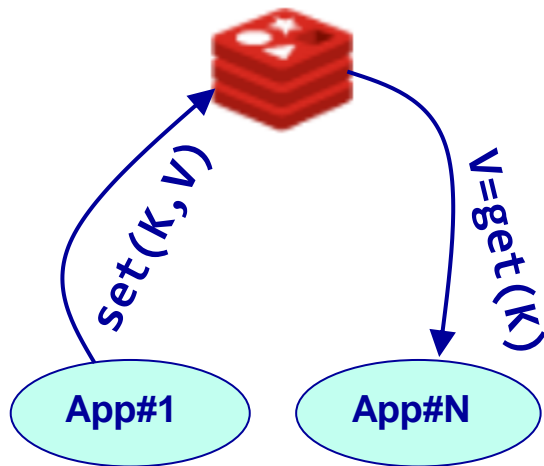
*“Neo4j is a highly scalable native graph database, built to leverage not only data but also data relationships”*

<https://neo4j.com/product/neo4j-graph-database/>

# REDIS: Um exemplo (*Key, Value*)

“*RE*mote *DI*ctionary *S*erver” - <https://redis.io/>

- O Redis é um serviço que permite armazenamento *in-memory* baseado no conceito de dicionário ou dados estruturados em pares (*Key, Value*)
- Pode ser usado como base de dados NoSQL, como *cache* ou *message broker* permitindo armazenar estruturas de dados, tais como: *strings, hashes, lists, sets, geospatial indexes, etc.*
- Para além do armazenamento em memória, o Redis permite também persistir os dados em ficheiro.



Demo:  
Cliente Redis em Java (*ClientRedis.zip*)

Cenário de partilha de estado de sessão em aplicações Web com balanceamento de carga

<https://medium.com/volosoft/docker-web-farm-example-with-using-redis-haproxy-and-asp-net-core-web-api-8e3f81217fd2>

---

*Google Cloud Platform*

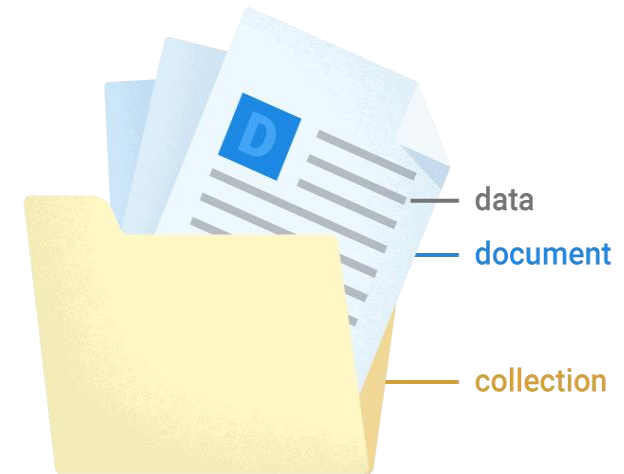
*Serviço Firestore*

(Armazenar dados *NoSQL* como coleções de documentos)



# Modelo de dados do *Firestore*

- A base de dados é um conjunto de coleções
- Coleções contêm documentos, os quais representam agregados de campos (*fields*) de vários tipos:
  - inteiros, booleanos, *arrays*, pontos geográficos, *maps*, *reference* para outros documentos
- Documentos da mesma coleção podem ter número e tipo de campos diferentes
- Um documento pode referir novas coleções
  - até um máximo de 100 níveis de profundidade
- Documentos devem ser pequenos
  - máximo 1 MiB (*mebibyte=1024\*1024 bytes*)




 users

 alovelace

`first : "Ada"`

`last : "Lovelace"`

`born : 1815`

 aturing

`first : "Alan"`

`last : "Turing"`

`born : 1912`

# Ativar serviço *Firestore* num projeto GCP

- Cada projeto pode ter uma base dados *Firestore*, que pode ser usada em modo *Datastore* (versão anterior) ou modo nativo (versão usada nos slides)

	Native mode	Datastore mode
	Enable all of Cloud Firestore's features, with offline support and real-time synchronization.  <a href="#">SELECT NATIVE MODE</a>	Leverage Cloud Datastore's system behavior on top of Cloud Firestore's powerful storage layer.  <a href="#">SELECT DATASTORE MODE</a>
API	Firestore	Datastore
Scalability	Automatically scales to millions of concurrent clients	Automatically scales to millions of writes per second
App engine support	Not supported in the App Engine standard Python 2.7 and PHP 5.5 runtimes	All runtimes
Max writes per second	10,000	No limit
Real-time updates	✓	✗
Mobile/web client libraries with offline data persistence	✓	✗

# Localização da Base de dados

✓ Select a Cloud Firestore mode — 2 Choose where to store your data

You selected Cloud Firestore in Native mode. Now choose a database location.

The location of your database affects its cost, availability, and durability. Choose a regional location (lower write latency, lower cost) or a multi-region location (higher availability, higher cost). [Learn more](#)

⚠ Choose carefully: your location selection is permanent and will also apply to this project's App Engine app

Select a location

Multi-region (99.999% SLA)

eur3 (Europe)

nam5 (United States)

Regional (99.99% SLA)

asia-east2 (Hong Kong)

asia-northeast1 (Tokyo)

asia-northeast2 (Osaka)

✓ Select a Cloud Firestore mode — 2 Choose where to store your data

You selected Cloud Firestore in Native mode. Now choose a database location.

The location of your database affects its cost, availability, and durability. Choose a regional location (lower write latency, lower cost) or a multi-region location (higher availability, higher cost). [Learn more](#)

⚠ Choose carefully: your location selection is permanent and will also apply to this project's App Engine app

Select a location


eur3 (Europe)


To improve performance, store your data close to the users and services that need it


CREATE DATABASE

BACK


# Base de dados como conjunto de coleções

 Firestore


 Data

 Indexes

Data

/ 

Root

 START COLLECTION

## Start a collection

A collection is a set of one or more documents that contain data. Start a collection at this path by adding its first document. [Learn more](#)

### Give the collection an ID

Parent path  
/




Collection ID \*  
Users


Choose an ID that describes the documents you'll add to this collection.

### Add its first document ?

Document ID  
alovelace

Assigned automatically. Customize if needed.

Field name	Field type	Field value	
first	string	Ada	
last	string	Lovelace	
born	string	1815	



SAVE

SAVE & ADD ANOTHER

CANCEL

## Field Types

### Add a field

Field name	Field type	Field value
address	string	

number

boolean

map

array

null

timestamp

geopoint

reference

CANCEL

SAVE FIELD

# Coleções e documentos

- Coleção **Users** tem dois documentos: **alovelace** e **aturing**
- O documento **aturing** tem uma coleção **moreinfo** com um documento **film**

