

Trabalhando com banco de dados

Bancos de dados relacionais (SQL)

Bancos de dados relacionais (SQL)

- Utiliza um sistema de queries (SELECT, UPDATE, DELETE...);
- Utiliza a estrutura de linhas, colunas e tabelas para armazenar os registros:
 - Tabelas -> Domínio (Usuário, Produto);
 - Linhas -> Registros (cada usuário, cada produto);
 - Colunas -> Propriedades de cada registro (Nome, Email, Descrição...);

Results Messages

	id	emp_name	emp_age	emp_salary	join_date	phone
1	1	Mike	35	5000.50	2016-01-01 00:00:00.000	042-4521-458452
2	2	Michale	30	4500.00	2016-05-01 00:00:00.000	021-1234-574845
3	3	Jimmy	27	3000.75	2016-05-03 00:00:00.000	001-4521-458452
4	4	Shaun	30	3500.30	2018-01-20 00:00:00.000	042-2544-458452
5	5	Ben	45	5000.00	2015-03-15 00:00:00.000	001-3355-457484
6	6	John	28	2600.00	2018-04-12 19:35:27.067	092-1414-458452
7	7	NULL	40	5500.00	2018-03-20 00:00:00.000	042-3333-458452
8	8	Jay	36	5500.00	2018-03-25 00:00:00.000	021-4242-145854
9	9	Haynes	25	3300.00	2018-03-20 00:00:00.000	033-1133-457457

Bancos de dados relacionais (SQL)



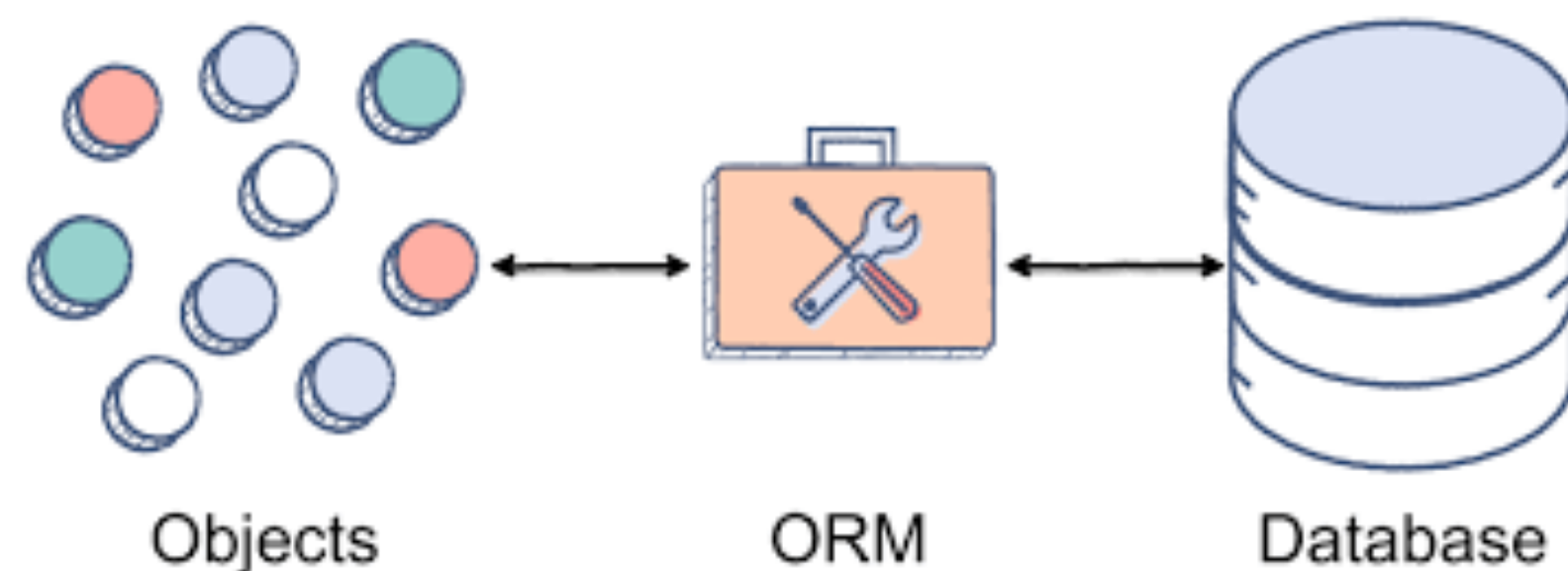
Bancos de dados relacionais (SQL)

- Na maioria das arquiteturas, sempre há uma camada de acesso a dados;
- Necessidade de manter a flexibilidade entre os bancos;
- Abstração entre a linguagem da aplicação e a linguagem de banco.

Banco de dados relacionais (SQL)

ORM para mapeamento de entidades em Node.js

- **Object Relational Mapper:** técnica que permite fazer uma relação entre os objetos presentes na aplicação com os dados/tabelas presentes no banco de dados.



<https://sequelize.org/>

Banco de dados relacionais (SQL)

Query Builder

- Conjunto de métodos disponíveis através de uma biblioteca para auxiliar na construção de queries no banco de dados.



<http://knexjs.org/>

Knex.js

- Biblioteca para construção de queries;
- Similar ao Entity Framework;
- Compatível com: Postgres, MSSQL, MySQL, MariaDB, SQLite3, Oracle, e Amazon Redshift;
- Utiliza uma mesma sintaxe para todos os bancos, abstraindo consultas simples através de métodos.
- Separação de ambientes (dev, staging, production, test...).

Knex.js - estrutura inicial com migrations

```
exports.up = function(knex) {  
  return knex.schema.createTable('products', table => {  
    table.increments('id').primary()  
    table.string('name')  
    table.string('description')  
    table.decimal('price')  
    table.timestamps(false, true) //created_at & updated_at  
  })  
};  
  
exports.down = function(knex, Promise) {  
  return knex.schema.dropTableIfExists('products')  
};
```

Knex.js - configurações

```
module.exports = {  
  development: {  
    client: 'sqlite3',  
    connection: {  
      filename: './dev.sqlite3'  
    },  
    useNullAsDefault: true  
  },  
}
```

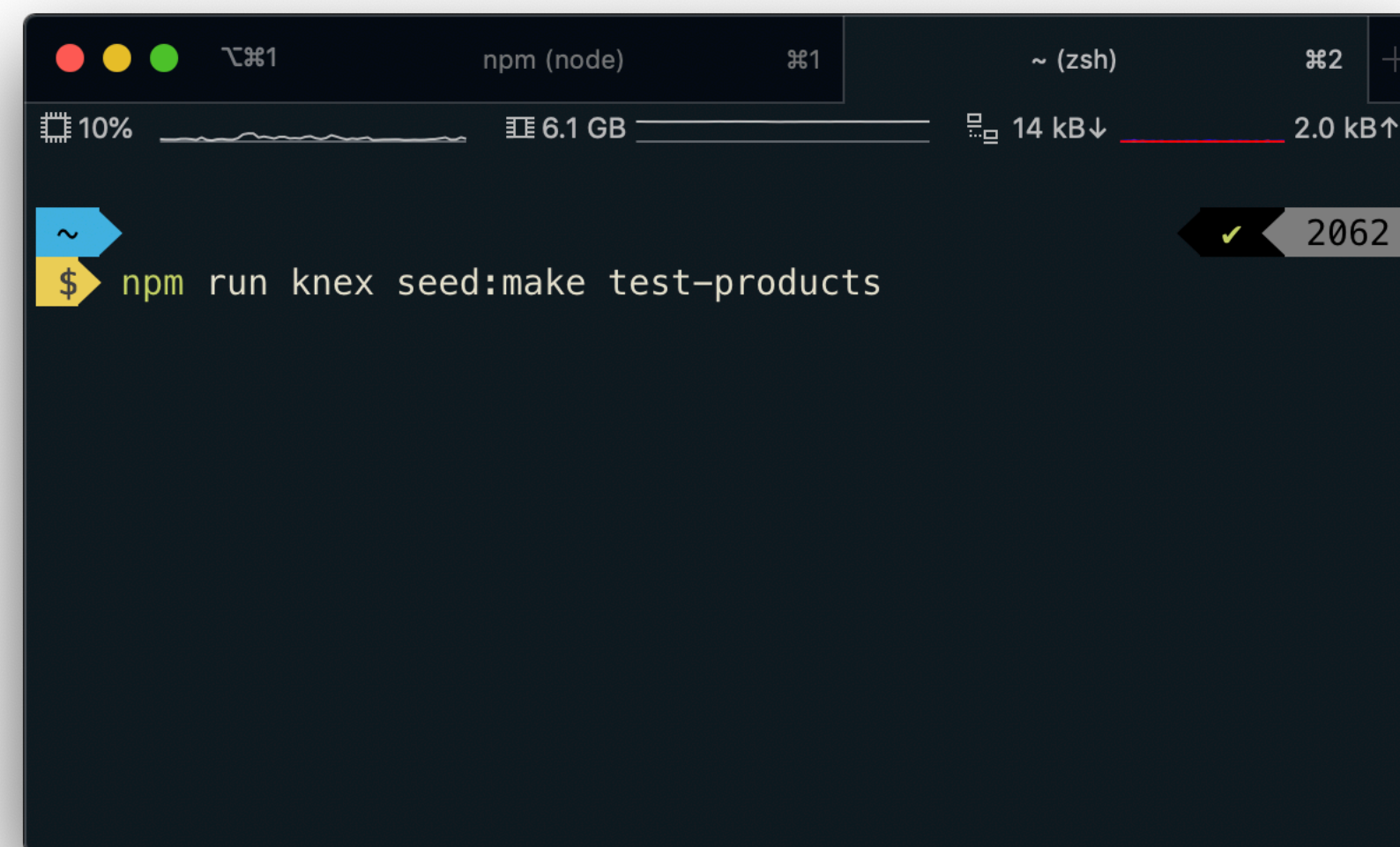
Knex.js - SQLite

- Banco relacional;
- Self-contained;
- Ótimo para ambiente de desenvolvimento e aplicações pequenas;
- Pode ser entregue juntamente com a aplicação;
- Cliente open-source para SQLite: <https://sqlitebrowser.org/dl/>



Knex.js - Seeds

- Possibilidade de gerar dados iniciais de exemplo no banco através de linha de comando.



A screenshot of a terminal window with a dark background. The window title bar shows 'npm (node)' and '~ (zsh)'. The terminal displays a command prompt '\$' followed by the command 'npm run knex seed:make test-products'. The command has been executed successfully, as indicated by a green checkmark icon and the number '2062' in the top right corner of the terminal pane. The terminal also shows system status indicators at the top: '10%' CPU usage, '6.1 GB' memory usage, and '14 kB↓ 2.0 kB↑' network activity.

Knex.js - Seeds

```
1 exports.seed = function(knex, Promise) {  
2   // Deletes ALL existing entries  
3   return knex('products').del()  
4     .then(function () {  
5     // Inserts seed entries  
6     return knex('products').insert([  
7       { id: 1, name: 'iPhone X', description: 'Amazing cellphone', price: 999.00 },  
8       { id: 2, name: 'Smartwatch', description: 'Amazing smartwatch', price: 349.00 }  
9     ]);  
10  });  
11 };  
12
```

Knex.js - Seeds

- Popular os dados via linha de comando



A screenshot of a macOS terminal window. The title bar shows the window name as '..s/nodejs-knex (zsh)' and the current directory as '~ (zsh)'. The terminal displays the command `npm run knex seed:run` with a yellow cursor at the end. The top status bar shows system metrics: 8% battery, a signal strength indicator, 5.7 GB of memory used, and network activity (14 kB down, 2.0 kB up). On the right side of the terminal, there is a small black box with a green checkmark and the number '2064'.

Knex.js - GET (SELECT)

```
const knex = require('knex')
const knexConfigs = require('../knexfile')
const db = knex(knexConfigs.development)

const TABLE_NAME = 'products'

module.exports = {
  get() {
    return db(TABLE_NAME).select('*')
  },
}
```


Knex.js - (INSERT)

```
module.exports = {  
  insert(product) {  
    return db(TABLE_NAME).insert(product);  
  }  
}
```

Knex.js - (DELETE)

```
module.exports = {  
  delete(product) {  
    return db(TABLE_NAME)  
      .where('id', product.id)  
      .del();  
  },  
}
```

Knex.js - (UPDATE)

```
module.exports = {  
  update(product) {  
    return db(TABLE_NAME)  
      .where('id', product.id)  
      .update({  
        name: product.name,  
        description: product.description,  
        price: product.price  
      });  
  }  
}
```

Knex.js - módulo de produtos como Serviço

```
const knex = require('knex')
const knexConfigs = require('../knexfile')
const db = knex(knexConfigs.development)

const TABLE_NAME = 'products'

module.exports = {
  get() {
    return db(TABLE_NAME).select('*')
  },
  insert(product) {
    return db(TABLE_NAME).insert(product);
  },
  delete(product) {
    return db(TABLE_NAME)
      .where('id', product.id)
      .del();
  },
  update(product) {
    return db(TABLE_NAME)
      .where('id', product.id)
      .update({
        name: product.name,
        description: product.description,
        price: product.price
      });
  }
}
```


Implementação com View Engine

Knex.js

Exemplo com View Engine

```
1 const Products = require('../store/Products')
2
3 /* GET home page */
4 router.get('/', function(req, res) {
5   Products.get()
6     .then(function(products) {
7       res.render('index', { products })
8     })
9 })
```

Knex.js

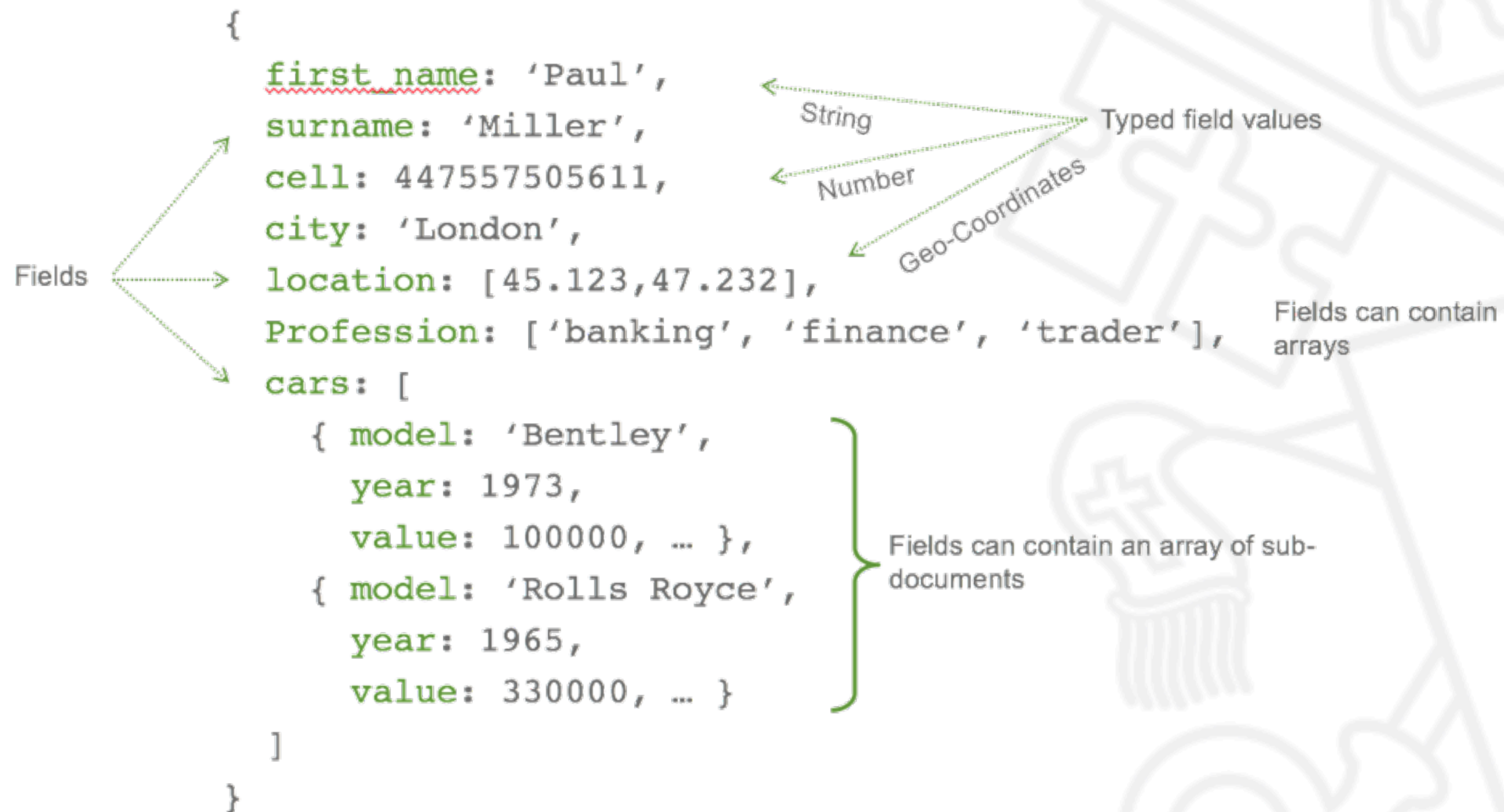
Exemplo com View Engine

```
1 <ul>
2     {{#each products}}
3         <li>
4             <p><b>{{this.name}}</b>: {{this.description}}</p>
5             <p><b>Price</b>: ${{this.price}}</p>
6         </li>
7     {{/each}}
8 </ul>
```

Bancos de dados não-relacionais (No-SQL)

Bancos de dados não-relacionais (No-SQL)

- Not Only SQL;
- Armazena os registros em formato de documentos;
- Cada documento é composto por um conjunto de chave:valor, similar ao JSON.



Bancos de dados não-relacionais (No-SQL)



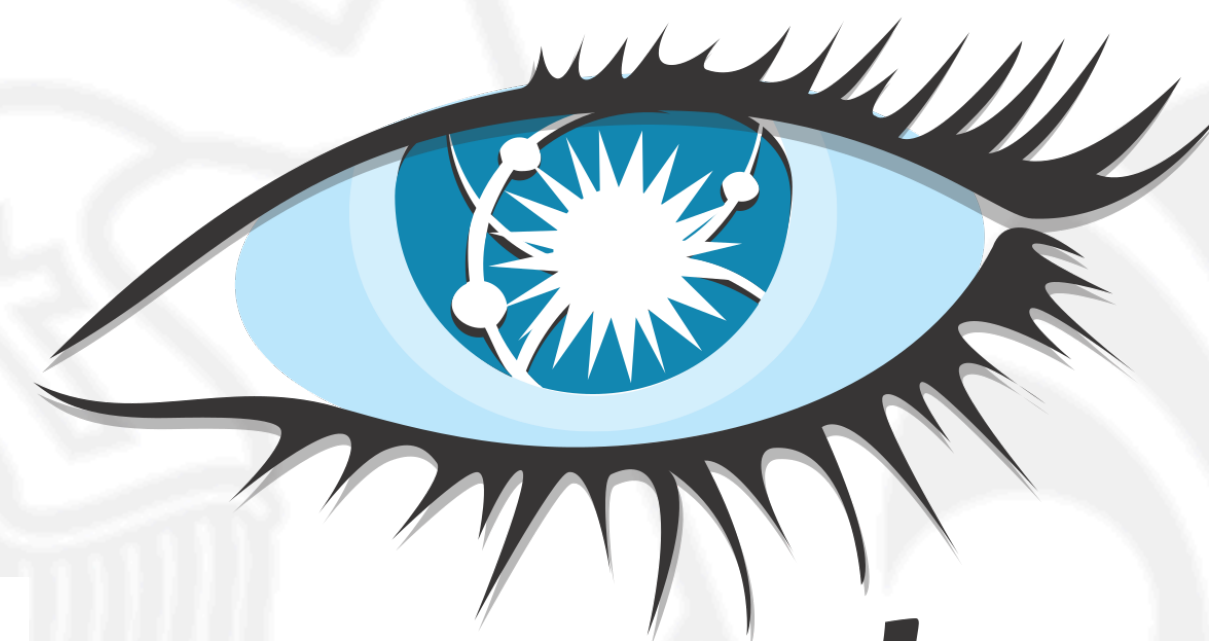
mongoDB



CouchDB



Azure Cosmos DB



cassandra