Aviso legal

Serviços para Aplicações Web e Móveis

Acesso a dados

Fábio Marques (fabio@ua.pt)
Mestrado em Informática Aplicada

Escola Superior de Tecnologia e Gestão de Águeda
Universidade de Aveiro

Acesso a dados

- Introdução
- Abordagem relacional
- Abordagem orientada a objetos
- BD não estruturadas

- São ferramentas que permitem o armazenamento de dados
- Têm um papel fundamental no desenvolvimento web e móvel disponibilizando uma forma estruturada e organizada de armazenar, gerir e aceder aos dados
- Uma base de dados bem desenhada é crucial para o desempenho, a escalabilidade e a fiabilidade de uma aplicação

- Estruturadas
  - Exemplos de SGBD: `MySQL`, `MSSQL`, `PostgreSQL`
  - Comandos SQL: `SELECT`, `UPDATE`, `INSERT INTO`, `DELETE` (lógico e físico)
  - Abordagem relacional: utiliza-se a linguagem SQL para dizer à BD "o que" fazer.
  - Abordagem orientada a objeto: utilizam-se os mecanismos orientados a objetos (classes e métodos) que indicam "como" fazer
- Não estruturadas
  - `NoSQL` (`Non-SQL`)
  - Exemplos de BD documentais: `MongoDB`, `CouchDB`, `RavenDB`

- O Node.js tem suporte para acesso a diferentes sistemas de gestão de bases de dados (SGBD)
- Para os podermos utilizar é necessário instalar o driver correspondente ao SGBD que pretendemos utilizar

Database	**Driver**	**comando npm**
MS SQL Server	mssql	npm install mssql
Oracle	oracledb	npm install oracledb
MySQL	MySQL2	npm install mysql2
PostgreSQL	pg	npm install pg
SQLite	node-sqlite3	npm install node-sqlite
…	…	…

```
const mysql = require('mysql2');
let config = {
  host: "dbhost",
  user: "username",
  password: "password"
  database: "database"
};
let con = mysql.createConnection(config);
con.connect(function(err) {
  if (err) throw err; // Error handling
    console.log("Connected!");
});

module.exports = con;
```

[1] https://www.npmjs.com/package/mysql2

```
pool.getConnection((err, connection) => { // Using a pool of connections
  if (err) {
    if (err.code === 'PROTOCOL_CONNECTION_LOST') {
      console.error('Database connection was closed.');
    }
    if (err.code === 'ER_CON_COUNT_ERROR') {
      console.error('Database has too many connections.');
    }
    if (err.code === 'ECONNREFUSED') {
      console.error('Database connection was refused.');
    }
    console.error('Error connecting to database:', err);
  }
  if (connection) {
    connection.release();
  }
  return; });
```

- Se não tiver acesso a um SGBD, instale um à sua escolha
- Crie um novo utilizador e uma nova base de dados (que será utilizada nos exercícios)
- Crie uma tabela com os campos que achar necessários para dar suporte ao exercício que temos vindo a desenvolver
- Preencha a tabela com dados de teste
- No projeto da biblioteca, instale o driver para o SGBD que escolheu.
- Estabeleça a ligação à base de dados

A primeira forma é invocar o método `.query(sqlString)`

```
let sql = "SELECT * FROM books";
const [rows, fields] = await con.query(sql);
```

A segunda forma é utilizando preparedStatements recorrendo ao método `.execute(sqlString, values)`

```
const sql = 'SELECT * FROM `users` WHERE `name` = ? AND `age` > ?';
const values = ['Page', 45];

const [rows, fields] = await connection.execute(sql, values);
```

Qualquer uma das formas é válida, mas a segunda é mais segura e eficiente. Qualquer das instruções pode ser utilizada para executar qualquer tipo de instrução SQL.

Utilizando a base de dados que criou e a tabela que definiu, implemente uma função que:

1. devolva todos os livros da base de dados
2. devolva um livro específico da base de dados
3. adicione um novo livro à base de dados
4. atualize um livro existente na base de dados
5. elimine um livro da base de dados

- Object-Relational Mapping (ORM)
- Permite o mapeamento entre um registo numa base de dados relacional e um objeto
- Para além da instalação da biblioteca ORM a utilizar é também necessário instalar o driver para o SGBD escolhido

Biblioteca	**comando npm**
Sequelize	npm install sequelize
Bookshelf.js	npm install knex / npm install bookshelf
Waterline	npm install waterline
…	…

- Técnica que permite relacionar objetos com relações nas bases de dados relacionais
- Tem como principal objetivo permitir a manipulação de dados de forma mais natural e orientada a objetos, em detrimento da manipulação de dados através de SQL

Principais conceitos:

- Entidades/Objetos: representam as tabelas da base de dados
- Atributos: representam as colunas das tabelas. Cada atributo guarda um valor para um objeto

- Mapeamento: utilizam o mapeamento para definir a forma como os objetos da aplicação se relacionam com as tabelas da base de dados
- Relações: permitem definir relações entre objetos/entidades que espelham as relações entre tabelas
- Operações CRUD: fornecem métodos para realizar as operações CRUD em objetos sem a necessidade de escrever SQL
- Linguagem de consulta: permite a utilização de uma linguagem de consulta orientada a objetos

Aspeto positivo → reduz a quantidade de código escrito tornando o sistema mais robusto

Aspeto negativo → algumas ferramentas ORM não permitem o processamento de dados em massa

SEQUELIZE: ESTABELECER A LIGAÇÃO À BD

```
const Sequelize = require("sequelize");

const sequelize = new Sequelize(
  'database_name',
  'username',
  'password',
  {
      host: 'hostname',
      port: portnumber,
      dialect: 'sgbdname' //mysql, postgres, sqlite, mssql
  }
);

module.exports = sequelize;
```

Modelos são a representação das tabelas da base de dados. Cada modelo é uma classe que representa uma tabela da base de dados e cada propriedade do modelo representa uma coluna na tabela.

```
module.exports = function Book(sequelize) {
  return sequelize.define('books', {
    title: {
      type: DataTypes.STRING,
      allowNull: false
    },
    author: {
      type: DataTypes.STRING,
      allowNull: false
    },
```

```
      release_date: {
        type: DataTypes.DATEONLY,
      },
      subject: {
        type: DataTypes.INTEGER,
      }
    });
  }
```

É possível definir relações entre modelos. Por exemplo, um livro pode ter um ou mais autores.

Também é possível definir migrações para a base de dados, permitindo gerir as alterações à estrutura da base de dados.

```
sequelize.sync() // sincronizacao de todos os modelos com a base de dados
  .then() => {
    console.log('Database synchronized');
  }
  .catch(err => {
    console.error('Error synchronizing database:', err);
  });
...

Book.findAll().then((books) => {res.json(books);})
  .catch(next);
...
book.findOne({where: {id,},});
```

Utilizando a base de dados que criou e a tabela que definiu, adapte a aplicação que está a desenvolver para utilizar o Sequelize. Implemente uma função que:

1. devolva todos os livros da base de dados
2. devolva um livro específico da base de dados
3. adicione um novo livro à base de dados
4. atualize um livro existente na base de dados
5. elimine um livro da base de dados

- O Node.js tem suporte para acesso a diferentes bases de dados documentais.
- Para os podermos utilizar é necessário instalar o driver correspondente

Database	Driver	comando npm
MongoDB	mongodb	npm install mongodb
Cassandra	cassandra-driver	npm install cassandra-driver
LevelDB	leveldb	npm install level levelup leveldown
RavenDB	ravendb	npm install ravendb
Redis	redis	npm install redis
CouchDB	nano	npm install nano
…	…	…