code
URJC

Tema 8

Bases de datos con Node

Micael Gallego
micael.gallego@gmail.com
@micael_gallego

Bases de datos con Node

- **Bases de datos NoSQL**
- MongoDB
- Node y MongoDB
- Bases de datos SQL
- PostgreSQL
- Node y PostgreSQL

Bases de datos

code URJC

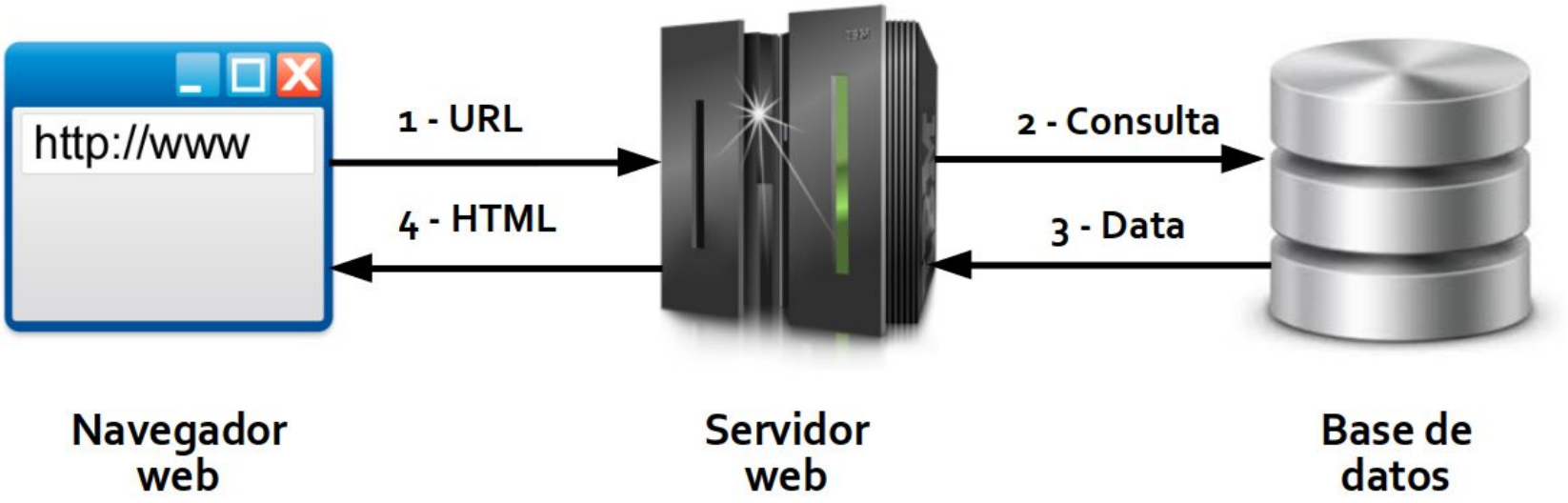
- **Bases de datos (*Database systems*)**
  - Son programas utilizados para almacenar información y permitir un acceso posterior a ella
  - Varios programas (servidor web, aplicación gráfica, ...) pueden acceder a la información de forma **concurrente** a través de la red

Bases de datos

- **Bases de datos (*Database systems*)**
  - La información está **centralizada**, **actualizada** y es más sencillo realizar actualizaciones y copias de seguridad
  - La información puede estar en forma de texto, números, ficheros, XML, etc…
  - Existen muchos tipos de bases de datos, pero las más usadas son las **Bases de datos SQL**

Bases de datos SQL

- Despliegue típico de una aplicación web con base de datos



Bases de datos NoSQL

- Las bases de datos NoSQL son bases de datos con **modelos de datos diferentes al relacional**
- **NoSQL** es un término que referencia a esta diferencia con las BBDD relacionales, pero no hay una definición formal
- Actualmente se entiende NoSQL como **"Not Only SQL"**

Bases de datos NoSQL

code URJC

- **Características comunes**
  - No tienen esquema de datos. La estructura de los datos se define en cada inserción
  - No se basan en SQL (aunque pueden tener lenguajes de consulta)
  - Son escalables mediante clusterización
  - Están pensadas para las necesidades de las aplicaciones web actuales
  - La mayoría son open-source

Bases de datos NoSQL

code URJC

- **¿Por qué existen las bases de datos NoSQL?**
  - Las bases de datos **relacionales no son escalables** porque no se pueden clusterizar
  - Algunos lenguajes requieren otros tipos de datos en vez de los usados en las BBDD relacionales (***impedance mismatch***)
  - La necesidad de **crear un esquema antes de guardar datos** dificulta el desarrollo

Bases de datos NoSQL

Bases de datos NoSQL

code URJC

- **Persistencia políglota (Polyglot persistence)**
  - Hay que **decidir** la base de datos a utilizar en cada aplicación
  - Diferentes aplicaciones tienen **diferentes necesidades**
  - Hay aplicaciones que pueden tener varias bases de datos a la misma vez
    - **Redis** para sesiones o colas de mensajes
    - **ElasticSearch** para logs
    - **MongoDB** para documentos

Bases de datos con Node

code URJC

- Bases de datos NoSQL
- **MongoDB**
- Node y MongoDB
- Bases de datos SQL
- PostgreSQL
- Node y PostgreSQL

MongoDB

https://www.mongodb.com/

MongoDB

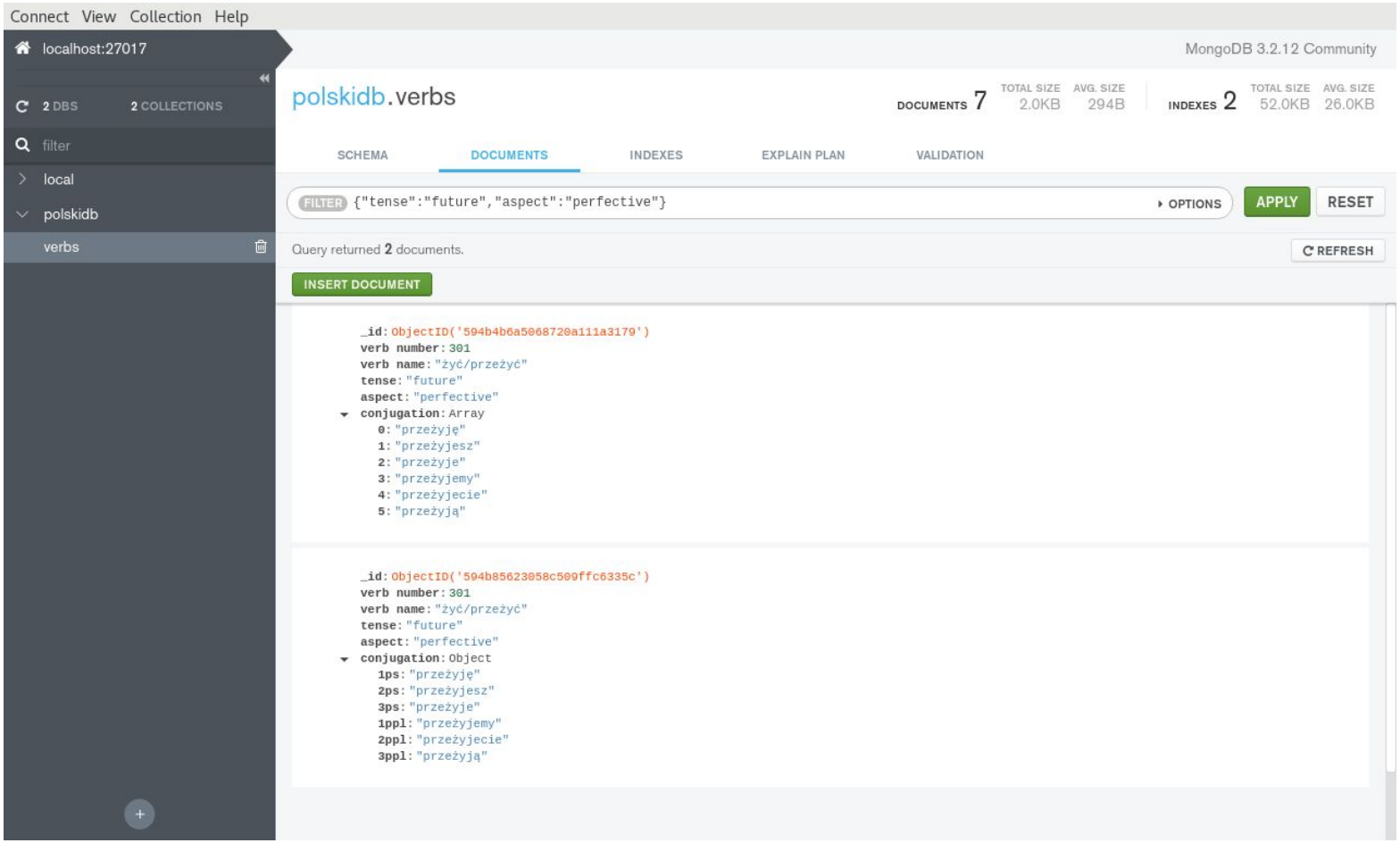
- Instalar MongoDB (CE) en el sistema:

https://docs.mongodb.com/manual/installation/#mongodb-community-edition-installation-tutorials

- Instalar un cliente gráfico (opcional)

https://www.mongodb.com/products/compass

MongoDB



https://www.mongodb.com/products/compass

MongoDB

code URJC

- En MongoDB se manejan **colecciones**, que guardan **documentos**.
  - Una **base de datos** Mongo tiene un conjunto de **colecciones**
  - Una **colección** tiene un conjunto de **documentos**
  - Un **documento** es un binario serializado en formar **BSON** (*Binary jSON*)

https://mongodb.github.io/node-mongodb-native/3.3/api/Collection.html

MongoDB

code URJC

- BSON

https://docs.mongodb.com/manual/reference/bson-types

- Double
- String
- Object
- Array
- Binary data
- Undefined
- ObjectId
- Boolean
- Date
- Null
- Regular Expression
- DBPointer
- JavaScript
- Symbol
- JavaScript (with scope)
- 32-bit integer
- Timestamp
- 64-bit integer
- Decimal128
- Min key
- Max key

MongoDB

- BSON

https://docs.mongodb.com/manual/reference/bson-types

- Double
- **String**
- **Object**
- Array
- Binary data
- **Undefined**
- ObjectId
- **Boolean**
- Date
- **Null**
- Regular Expression
- DBPointer
- JavaScript
- **Symbol**
- JavaScript (with scope)
- 32-bit integer
- Timestamp
- 64-bit integer
- Decimal128
- Min key
- Max key

- JSON
  - Boolean
  - Null
  - Undefined
  - Number
  - String
  - Symbol (ES6)
  - Object: Object, Array, Function…

MongoDB

- Operaciones sobre una BBDD MongoDB:
  - Añadir una colección
  - Eliminar una colección
  - Añadir un documento a una colección
  - Buscar documentos en una colección
  - Actualizar un documento en una colección
  - Eliminar un documento de una colección
  - Manejo de arrays

MongoDB

- Comandos en la shell básicos
  - Listar colecciones

```
$ mongo <DB_NAME> --eval "db.getCollectionNames()"
```

  - Borrar colección

```
$ mongo <DB_NAME> --eval "db.<COLLECTION_NAME>.drop()"
```

  - Listar documentos de colección

```
$ mongo <DB_NAME> --eval "db.<COLLECTION_NAME>.find()"
```

  - Borrar base de datos

```
$ mongo <DB_NAME> --eval "db.dropDatabase()"
```

Bases de datos con Node

code URJC

- Bases de datos NoSQL
- MongoDB
- **Node y MongoDB**
- Bases de datos SQL
- PostgreSQL
- Node y PostgreSQL

Node y MongoDB

code URJC

- Instalar el driver oficial de MongoDB en tu aplicación Node

```
$ npm install --save mongodb
```

Node y MongoDB

code URJC

ejem1

```
const MongoClient = require('mongodb').MongoClient;
const url = "mongodb://localhost:27017/customersDB";

async function main() {

   const conn = await MongoClient.connect(url, {
       useUnifiedTopology: true,
       useNewUrlParser: true
   });

   const customers = conn.db().collection('customers');

   await customers.insertOne({
       'firstName': 'Jack',
       'lastName': 'Bauer'
   });

   conn.close();
}

main();
```

Node y MongoDB

code URJC

ejem2

- Añadir un documento:

```
await customers.insertOne({
    firstName: 'Jack',
    lastName: 'Bauer'
});
```

  - Obtener el id del elemento:

```
const { insertedId } = await customers.insertOne({
    firstName: 'Jack',
    lastName: 'Bauer'
});

console.log('Customer inserted with id:', insertedId);
```

Al objeto que se inserta también se le añade un atributo **_id** con el nuevo id

Node y MongoDB

code URJC

ejem2

- Añadir varios documentos:

```
await customers.insertMany([
    { firstName: 'Jack', lastName: 'Bauer' },
    { firstName: 'Juan', lastName: 'Pérez' }
]);
```

  - Obtener los ids

```
const { insertedIds } = await customers.insertMany([
    { firstName: 'Jack', lastName: 'Bauer' },
    { firstName: 'Juan', lastName: 'Pérez' }
]);

console.log('Customers inserted with ids:', insertedIds);
// Customers inserted with ids: { '0':
5df95777099a07467dbe941d, '1': 5df95777099a07467dbe941e }
```

Node y MongoDB

- Consulta de documentos

ejem2

```
const result = await customers.find({ firstName:'Juan'}).toArray();

console.log('Customers with firstName = "Juan":', result);
```

```
[
  {
    _id: 5df959757517c34d58bd80f5,
    firstName: 'Juan',
    lastName: 'Pérez'
  }
]
```

Node y MongoDB

- Consultas de documentos
  - **`collection.find({})`**: devuelve todos los documentos
  - **`collection.find({field: value})`**: documentos que tengan el campo especificado con el valor especificado
  - **`collection.find({field:{$in:['value1','value2']})`**: documentos cuyo campo especificado tenga uno de los valores indicados
  - **`collection.find({field1:value1, field2:value2})`**: AND lógico
  - **`collection.find({$or:[{field1:value1},{field2:value2}]})`**: OR lógico

https://docs.mongodb.com/manual/reference/operator/query

Node y MongoDB

- Obtener un documento por su id

ejem2

```
const ObjectId = require('mongodb').ObjectId;

...

const customer = await customers.findOne({ _id: new ObjectId(id)});

console.log('Customer with id:', customer);
```

Node y MongoDB

code URJC

- Actualizar un documento:

ejem2

```
await customers.updateOne(
    { _id: new ObjectId(id) },
    { $set: { firstName: 'Pedro', age: 45 } }
);
```

- Actualizar varios documentos:

```
const { matchedCount } = await customers.updateMany(
    { firstName: 'Juan' },
    { $set: { firstName: 'John' } }
);

console.log(`Updated ${matchedCount} customers with name "Juan"`);
```

Node y MongoDB

code URJC

- Eliminar un documento:

ejem2

```
await customers.deleteOne({ _id: new ObjectId(id) });
```

- Eliminar varios documentos:

```
const { deletedCount } = await customers.deleteMany({ firstName: 'Juan' });

console.log(`Deleted ${deletedCount} customers with name "Juan"`);
```

MongoDB + Express

code URJC

ejem3

- Una API REST suele implementarse con persistencia
- En Node es habitual implementar una API REST con Express y Mongo

MongoDB + Express

code URJC

ejem3

- Consideraciones
  - Cambiar el mapa en memoria y el generador de identificadores únicos (uuid) por la conexión a Mongo
  - Cambiar la propiedad _id generada por Mongo por id
  - Crear una función para la conexión con la base de datos

Ejercicio 1

- Añade persistencia con Mongo a la API de gestión de Items del tema anterior

Documentos con arrays y objetos

code URJC

- Mongo permite insertar documentos con objetos anidados en propiedades y en arrays

```
[
  {
    item: 'journal',
    instock: [
      { warehouse: 'A', qty: 5 },
      { warehouse: 'C', qty: 15 }
    ],
    data: { num: 233, unit: 'C' }
  },
  {
    item: 'notebook',
    instock: [{ warehouse: 'C', qty: 5 }],
    data: { num: 456, unit: 'C' }
  },
  {
    item: 'paper',
    instock: [
      { warehouse: 'A', qty: 60 },
      { warehouse: 'B', qty: 15 }
    ],
    data: { num: 800, unit: 'C' }
  }
]
```

Documentos con arrays y objetos

- Existen mecanismos para manipular los objetos anidados y los arrays
  - Manipulación de objetos anidados
  - Manipulación de arrays de objetos

```
const cursor = db.collection('inventory').find({
  'instock.qty': 5,
  'instock.warehouse': 'A'
 });
```

https://docs.mongodb.com/manual/tutorial/query-array-of-documents/

Consultas avanzadas

- Devolver solo ciertos campos en las búsquedas

```
collection("customers").find(
    query,
    { projection: { _id: 0, firstName: 1, lastName: 0} }
);
```

  - 0 para no incluirlo, 1 para incluirlo (el campo _id se incluye por defecto)

Documentos con arrays

- Manejo de arrays: buscar documentos (I)
  - **`Collection.find({array:[“value1”, “value2”]})`**: documentos que tengan un array llamado “array” con los valores “value1” y “value2” (valores exactos en el orden especificado)
  - **`Collection.find({array:“value1”})`**: documentos que tengan un array llamado “array” que contenga el valor “value1”
  - **`Collection.find({array:{ $all: ["value1", "value2"] }})`**: documentos que tengan un array llamado “array” que contenga los valores “value1” y value2” (en cualquier orden y con cualquier otro valor extra)

https://docs.mongodb.com/manual/tutorial/query-arrays
https://docs.mongodb.com/manual/reference/operator/update-array/

Documentos con arrays

- Manejo de arrays: buscar documentos (II)
  - **`Collection.find({array: { $gt:10, $lt:20 }})`**: documentos que tengan un array llamado "array" que contenga un valor mayor que 10 y uno menor que 20 (o uno que cumpla ambos)
  - **`Collection.find({array: { $elemMatch: { $gt:10, $lt:20 }}})`**: documentos que tengan un array llamado "array" que contenga al menos un valor mayor que 10 y menor que 20
  - **`Collection.find({"array.2": { $gt: 10 } })`**: documentos que tengan un array llamado "array" que contenga un elemento mayor que 10 en la tercera posición
  - **`Collection.find({"array": { $size: 5 }})`**: documentos que tengan un array llamado "array" con tamaño 5

Documentos con arrays

- Manejo de arrays: adición de elementos
  - Añadir un elemento

```
var query = { lastName: /^T/ };
var newValues = { $push: { arrayProperty: newElement } };

collection.updateMany(
     query, // those documents with a “lastName” property starting with “T”
     newValues // “newElement” object will be pushed to “arrayProperty” array
);
```

Si queremos que los elementos del array tengan un identificador propio para poder manejarlos individualmente más adelante, tendremos que añadírselo explícitamente con la clase **ObjectId** de mongodb

```
var ObjectID = require('mongodb').ObjectID;
newElement._id = new ObjectID();
```

Documentos con arrays

- Manejo de arrays: adición de elementos
  - Añadir múltiples elementos

```
var query = { lastName: /^T/ };
var newValues = { $push: { arrayProperty: { $each: [ 10, 20, 33 ] } } };

collection.updateOne(
    query, // first document with a “lastName” property starting with “T”
    newValues // each one of the elements in the $each array will be pushed
              // to “arrayProperty” array
);
```

Documentos con arrays

- Manejo de arrays: borrado de elementos

```
var query = { lastName: /^T/ };
var newValues = { $pull: { arrayProperty: { $in: ["elem1","elem2"]} } };

collection.updateOne(
    query, // first document with a “lastName” property starting with “T”
    newValues // each one of the elements in the $in array will be removed
              // from “arrayProperty” array
);
```

Documentos con arrays

ejem3

- Manejo de arrays: actualización de elementos

```
var query = { lastName: /^T/, arrayProperty: 90 };
// Must include the array property inside the query!

var newValues = { $set: { "arrayProperty.$": 99 };

collection.updateOne(
    query, // first document with a “lastName” property starting with “T”
           // and an “arrayProperty” array containing value 90
    newValues // substitute 90 value of the “arrayProperty” array to 99
);
```

Consultas avanzadas en MongoDB

code URJC

- **Aggregation framework**:
  - Las operaciones de agregación agrupan valores de múltiples documentos y pueden realizar diversas operaciones en los datos agrupados para devolver un único resultado.
  - También nos permite combinar objetos de varias colecciones en la misma consulta

```
collection1.aggregate([{
   $lookup: {
      from: 'collection2',
      localField: 'fieldFromCollection1',
      foreignField: 'fieldFromCollection2',
      as: 'outputArray'
}}]
```

Archivos en MongoDB

- Manejo de archivos con MongoDB:
  - MongoDB es una BBDD muy utilizada para el guardado de archivos.
  - Para ello se usa **GridFS**: es la especificación de MongoDB para el manejo de archivos de gran tamaño.

https://docs.mongodb.com/manual/core/gridfs/

Object Document Mapper (ODM)

code URJC

- La librería de acceso a Mongo permite ejecutar sentencias sobre la base de datos
- Un **ODM** es una librería que nos permite un acceso a la base de datos de más alto nivel y más idiomático del lenguaje de programación
- Para las bases de datos relacionales a estas librerías se las conoce como **ORM** (Object Relational Mapper)

https://en.wikipedia.org/wiki/Object-relational_mapping

Object Document Mapper (ODM)

- En Node.js el ODM más usado para MongoDB es Mongoose

https://mongoosejs.com/

Mongoose

- Mongoose permite definir la **estructura de los datos** que se guardarán en MongoDB (**esquema**)
- **Validará** que esa estructura se cumple
- Permite incluir **valores por defecto** para las propiedades sin valor
- Cuando se modifican sólo algunos atributos de un objeto, únicamente se **guardan los cambios** (mejorando la eficiencia)

Mongoose

- Instalar dependencia

```
$ npm install --save mongoose
```

Mongoose

```
const MongoClient = require('mongodb').MongoClient;
const url = "mongodb://localhost:27017/customersDB";

async function main() {

   const conn = await MongoClient.connect(url, {
       useUnifiedTopology: true,
       useNewUrlParser: true
   });

   const customers = conn.db().collection('customers');

   await customers.insertOne({
       'firstName': 'Jack',
       'lastName': 'Bauer'
   });

   conn.close();
}

main();
```

Sin mongoose

```
const mongoose = require('mongoose');

const url = "mongodb://localhost:27017/customersDB";

async function main() {

   await mongoose.connect(url, {
       useUnifiedTopology: true,
       useNewUrlParser: true,
       useFindAndModify: false
   });

   var customerSchema = new mongoose.Schema({
       firstName: String,
       lastName: String
   });

   var Customer = mongoose.model('Customer', customerSchema);

   let customer = new Customer({ firstName: 'Jack', lastName: 'Bauer' })

   await customer.save();

   conn.close();
}
main();
```

Mongoose

ejem5

- Añadir un documento

```
await customers.insertOne({
    firstName: 'Jack',
    lastName: 'Bauer'
});
```



```
let customer = new Customer({
   firstName: 'Jack',
   lastName: 'Bauer'
});

await customer.save();
```

Mongoose

- Consulta de documentos

ejem5

```
const result = await customers.find({ firstName:'Juan'}).toArray();
```



```
const result = await Customer.find({ firstName: 'Juan' }).exec();
```

Mongoose

- Obtener un documento por su id

ejem5

```
const ObjectId = require('mongodb').ObjectId;
...
const customer = await customers.findOne({ _id: new ObjectId(id)});
```



```
const customer = await Customer.findById(id);
```

Mongoose

code URJC

- Actualizar un documento:

ejem5

```
await customers.updateOne(
    { _id: new ObjectId(id) },
    { $set: { firstName: 'Pedro', age: 45 } }
);
```



```
await Customer.findByIdAndUpdate(id,
    { $set: { firstName: 'Pedro', age: 45 }}
);
```

Mongoose

- Eliminar un documento:

ejem5

```
await customers.deleteOne({ _id: new ObjectId(id) });
```



```
await Customer.findByIdAndDelete(id);
```

Mongoose

code URJC

ejem6

- La implementación de una API REST con mongoose en vez de con el driver oficial de Mongo es muy similar

Ejercicio 2

- Cambia la persistencia de la API de Items para que use Mongoose

Bases de datos con Node

code URJC

- Bases de datos NoSQL
- MongoDB
- Node y MongoDB
- **Bases de datos SQL**
- MySQL
- Node y MySQL

Bases de datos SQL

- Una base de datos **relacional** almacena la información en tablas* con filas y columnas (campo)

Tabla Libros

idLibro	titulo	precio
1	Bambi	3
2	Batman	4
3	Spiderman	2

Tabla Autores

idAutor	nombre	nacionalidad
1	Antonio	Español
2	Gerard	Frances

Tabla RelacionLibroAutor

idLibro	idAutor
1	1
2	2
3	2

* A las tablas se las denominaba “relaciones”, de ahí el nombre de base de datos relacional

Bases de datos SQL

- Una base de datos **relacional** almacena la información en tablas* con filas y columnas (campo)

Tabla Libros

idLibro	titulo	precio
1	Bambi	3
2	Batman	4
3	Spiderman	2

Tabla Autores

idAutor	nombre	nacionalidad
1	Antonio	Español
2	Gerard	Frances

Tabla RelacionLibroAutor

idLibro	idAutor
1	1
2	2
3	2

La información se relaciona mediante identificadores (id)

Bases de datos SQL

- SQL (*Standard Query Language*)

```
SELECT titulo, precio
FROM Libros
WHERE precio > 2
```

Bases de datos SQL

- **SQL (*Standard Query Language*):** Lenguaje de consulta estándar que permite:
  - Consulta de los datos seleccionados con diferentes criterios y realizando operaciones (medias, sumas,…)
  - Inserción, Actualización y borrado de la información
  - Creación, alteración y borrado de las tablas y sus campos
  - Gestión de usuarios y sus privilegios de acceso

Sentencia SQL SELECT

- También conocido como ***statement* o *query*** (consulta)
- Permite **recuperar** la información de una o varias tablas
- Especifica uno o más **campos**, una o más **tablas** y un **criterio** de selección
- La base de datos devuelve los campos indicados de aquellas **filas** que cumplan el **criterio** de selección

Sentencia SQL SELECT

code URJC

Situación en la base de datos

Tabla Libros

idLibro	titulo	precio
1	Bambi	3
2	Batman	4
3	Spiderman	2

Consulta

```
SELECT titulo, precio
FROM Libros
WHERE precio > 2
```

Conjunto de resultados (*ResultSet*)

titulo	precio
Bambi	3
Batman	4

Cláusula WHERE

- Operador LIKE (Comparación de cadenas)

```
SELECT titulo, precio
FROM Libros
WHERE titulo LIKE 'Ba%'
```

- Operadores relacionales (<,=,...) lógicos (AND, OR)

```
SELECT titulo, precio
FROM Libros
WHERE precio > 3 AND titulo LIKE '%Man'
```

Uniones (JOINs)

- Se pueden unir varias tablas en una consulta

Tabla Libros

idLibro	titulo	precio
1	Bambi	3
2	Batman	4
3	Spiderman	2

Tabla Autores

idAutor	nombre	nacionalidad
1	Antonio	Español
2	Gerard	Frances

Tabla RelacionLibroAutor

idLibro	idAutor
1	1
2	2
3	2

Uniones (JOINs)

- Se pueden unir varias tablas en una consulta

Tabla Libros

idLibro	titulo	precio
1	Bambi	3
2	Batman	4
3	Spiderman	2

Tabla Autores

idAutor	nombre	nacionalidad
1	Antonio	Español
2	Gerard	Frances

Tabla RelacionLibroAutor

idLibro	idAutor
1	1
2	2
3	2

Uniones (JOINs)

```
SELECT titulo, precio, nombre
FROM Libros, Autores, RelacionLibroAutor
WHERE Libros.idLibro = RelacionLibroAutor.idLibro
   AND Autores.idAutor = RelacionLibroAutor.idAutor
```

titulo	precio	nombre
Bambi	3	Antonio
Batman	4	Gerard
Spiderman	2	Gerard

Productos

- Existen muchos productos de bases de datos relacionales, comerciales y software libre

Bases de datos con Node

code URJC

- Bases de datos NoSQL
- MongoDB
- Node y MongoDB
- Bases de datos SQL
- **MySQL**
- Node y MySQL

MySQL

- http://www.mysql.org/
- Sistema gestor de base de datos multiplataforma
- Desarrollado en C
- Licencia código abierto GPL
- Soporte de un subconjunto de SQL 99
- Herramienta interactiva para hacer consultas y crear bases de datos
- Propiedad de ORACLE

MySQL

- Instalar MySQL

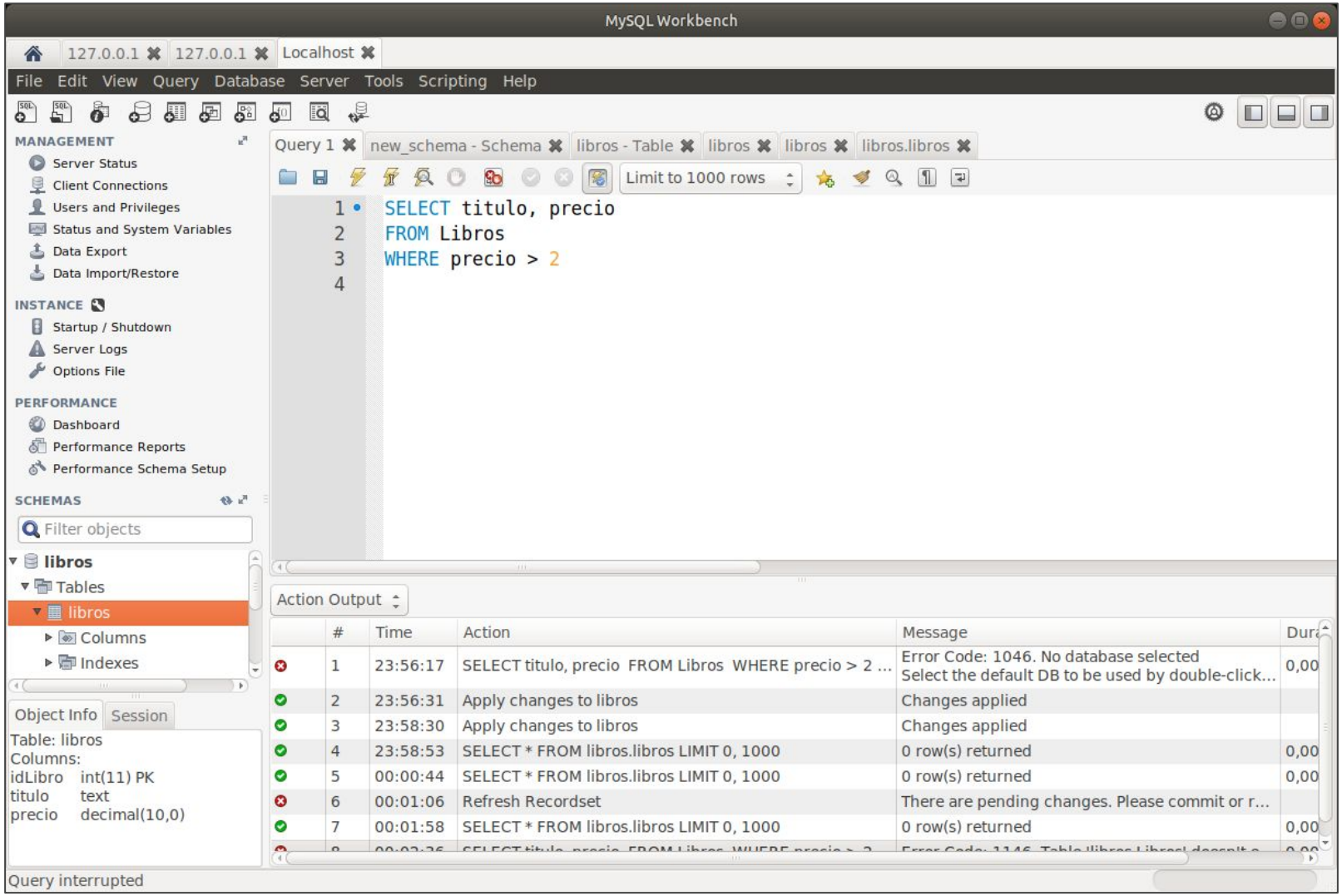
https://dev.mysql.com/downloads/mysql/

- Instalar cliente gráfico (opcional)

MySQL Workbench

https://www.mysql.com/products/workbench/

MySQL Workbench



Bases de datos con Node

code URJC

- Bases de datos NoSQL
- MongoDB
- Node y MongoDB
- Bases de datos SQL
- MySQL
- **Node y MySQL**

Node y MySQL

code URJC

- Existen varias librerías para conectarse a MySQL desde Node
  - **Mysql**
    - Más madura (14.9k *)
    - No soporta async/await
    - https://github.com/mysqljs/mysql
  - **Mysql2**
    - Desarrollo más reciente. Más rápida
    - Soporte de async/await
    - https://github.com/sidorares/node-mysql2

Node y MySQL

- Usaremos mysql2 para poder usar promesas

```
$ npm install --save mysql2
```

Node y MongoDB

code URJC

ejem7

```
const mysql = require('mysql2/promise');

async function main() {

   const conn = await mysql.createConnection({
       host: 'localhost',
       user: 'root',
       password: 'pass',
       database: 'customersDB'
   });

   console.log("Connected to MySQL");

   await conn.execute(
       'INSERT INTO customers SET firstName = ?, lastName = ?',
       ['Jack', 'Bauer']
   );

   console.log("Customer inserted");

   await conn.close();

   console.log("Connection closed");
}

main();
```

Node y MongoDB

code URJC

ejem8

- Añadir una fila (*row*):

```
await conn.execute(
   'INSERT INTO customers SET firstName = ?, lastName = ?',
   ['Jack', 'Bauer']
);
```

  - Obtener el id de la fila:

```
const [{ insertId }] = await conn.execute(
    'INSERT INTO customers SET firstName = ?, lastName = ?',
    ['Jack', 'Bauer']
);

console.log('Customer inserted with id:', insertId);
```

Node y MongoDB

code URJC

- Consulta de filas

ejem8

```
const [rows, fields] = await conn.execute(
    'SELECT * FROM customers WHERE firstName = ?',
    ['Jack']
);

console.log('Customers with firstName = "Jack":', rows);
```

Node y MongoDB

code URJC

- Obtener una fila por su id

ejem8

```
const [rows, fields] = await conn.execute(
   'SELECT * FROM customers WHERE id = ?',
   [id]
);

console.log('Customer with id:', rows[0]);
```

Node y MongoDB

code URJC

ejem8

- Actualizar una fila:

```
await conn.execute(
   'UPDATE customers SET firstName = ? WHERE id = ?',
   ['Pedro', id]
);
```

- Actualizar varias filas:

```
const [{ affectedRows }] = await conn.execute(
   'UPDATE customers SET firstName = ? WHERE firstName = ?',
   ['John', 'Pedro']
);
```

Node y MongoDB

code URJC

- Eliminar una fila:

ejem8

```
await conn.execute('DELETE FROM customers WHERE id = ?', [id]);
```

- Eliminar varias filas:

```
const [{ affectedRows }] = await conn.execute(
   'DELETE FROM customers WHERE firstName = ?',
   ['John']
);

console.log(`Deleted ${affectedRows} customers with name "John"`);
```

Ejercicio 3

- Cambia la persistencia de la API de Items para que use MySQL

Object Relational Mapper (ORM)

- La librería de acceso a MySQL permite ejecutar sentencias SQL sobre la base de datos
- Un **ORM** es una librería que nos permite un acceso a la base de datos de más alto nivel y más idiomático del lenguaje de programación

https://en.wikipedia.org/wiki/Object-relational_mapping

Object Relational Mapper (ORM)

- En Node.js el ORM más usado es Sequelize

https://sequelize.org/

Sequelize

- Permite definir la **estructura de los datos** en JavaScript
- La **tabla es creada por Sequelize** con esa información
- Puede añadir campos de registro de actualización (createdAt y updatedAt)
- **Consultas** con objetos literales (como en Mongo)
- Soporta varias bases de datos relacionales

Sequelize

- Instalar dependencia

```
$ npm install --save sequelize
```

- Instalar el driver de la base de datos (uno de ellos)

```
$ npm install --save pg pg-hstore # Postgres
$ npm install --save mysql2
$ npm install --save mariadb
$ npm install --save sqlite3
$ npm install --save tedious # Microsoft SQL Server
```

Sequelize

code URJC

ejem9

```
const { Sequelize, Model, DataTypes } = require('sequelize');

async function main() {

   const sequelize = new Sequelize('customersDB', 'root', 'pass', {
       dialect: 'mysql'
   })

   class Customer extends Model { }

   Customer.init({
       firstName: DataTypes.STRING,
       lastName: DataTypes.STRING
   }, { sequelize, modelName: 'customer' });

   await sequelize.sync();

   await Customer.create({
       firstName: 'Jack',
       lastName: 'Bauer'
   });

   await sequelize.close();
}

main();
```

Sequelize

ejem10

- Añadir una fila (*row*):

```
await Customer.create(
   { firstName: "Jack", lastName: "Bauer" }
);
```

  - Obtener el id de la fila:

```
const { dataValues: { id }} = await Customer.create(
   { firstName: "Jack", lastName: "Bauer" }
);

console.log('Customer inserted with id:', id);
```

Sequelize

- Consulta de filas

ejem10

```
const rows = await Customer.findAll({ where:{ firstName: 'Jack'}});

console.log('Customers with firstName = "Jack":', toPlainObj(rows));
```

Sequelize

- Obtener una fila por su id

ejem10

```
const rows = await Customer.findAll({ where: { id } });

console.log('Customer with id:', toPlainObj(rows[0]));
```

Sequelize

code URJC

ejem10

- Actualizar una fila:

```
await Customer.update(
   { firstName: 'Pedro' },
   { where: { id } }
);
```

- Actualizar varias filas:

```
const affectedRows = await Customer.update(
   { firstName: 'John' },
   { where: { firstName: 'Pedro' }}
);

console.log(`Updated ${affectedRows} customers with name "Pedro"`);
```

Sequelize

code URJC

- Eliminar una fila:

ejem10

```
await Customer.destroy({ where: { id } });
```

- Eliminar varias filas:

```
const affectedRows = await Customer.destroy(
   { where: { firstName: 'Jack' } }
);

console.log(`Deleted ${affectedRows} customers with name "John"`);
```

Relaciones

- Sequelize permite definir **relaciones** entre entidades del modelo **(1:1, 1:N, M:N)**
- Esas relaciones se traducen en campos en las tablas para relacionar los elementos en las **consultas**
- Sequelize puede **cargar varias filas** relacionadas en las consultas de forma sencilla

https://sequelize.org/v5/manual/associations.html

Relaciones

- **Relación M:N**
  - Un autor puede tener varios libros
  - Un libro puede tener varios autores

Tabla Libros

idLibro	titulo	precio
1	Bambi	3
2	Batman	4
3	Spiderman	2

Tabla Autores

idAutor	nombre	nacionalidad
1	Antonio	Español
2	Gerard	Frances

Tabla RelacionLibroAutor

idLibro	idAutor
1	1
2	2
3	2

La información se relaciona mediante identificadores (id)

Ejercicio 4

- Cambia la persistencia de la API de Items para que use MySQL con sequelize

Bases de datos con Node

code URJC

- **Bases de datos NoSQL**
- MongoDB
- Node y MongoDB
- Bases de datos SQL
- PostgreSQL
- Node y PostgreSQL
- Estructura de aplicaciones

Estructura de aplicaciones

code URJC

- Las aplicaciones web/APIs REST con express y un sistema de persistencia se deben organizar de forma adecuada
- Existen proyectos de ejemplo y recomendaciones
  - https://github.com/madhums/node-express-mongoose
  - https://medium.com/better-programming/best-node-js-boilerplate-to-speed-up-your-project-development-a9eca7b07f90
  - https://www.freecodecamp.org/news/how-to-write-a-production-ready-node-and-express-app-f214f0b17d8c/