ACÁMICA

¡Bienvenidos a Data Science!

Agenda

¿Cómo anduvieron?

Bases de datos

Break

Lab

Cierre

¿Cómo anduvieron?

Introducción a Base de Datos

¿Qué es una Base de datos?

Una base de datos
es un conjunto de información ordenada de modo sistemático para su posterior recuperación, análisis y/o transmisión.

Desde backend surge la necesidad de guardar la información. Hasta ahora guardamos información en:

- Variables
- Arrays

Si reiniciamos el server se pierde la información.

- Archivos

No recomendado. Debemos acceder al disco todo el tiempo, es lento y no nos provee ninguna funcionalidad.

YouTube AR

Buscar

Historia de las Bases de Datos

Base de datos • Ventajas

- Se puede **obtener información** en forma más sencilla, flexible y estructurada.
- **Compartir información** simultáneamente con otros usuarios o otras bases de datos.
- Facilita la **estandarización** de procesos, nombres de registros, etc.
- Permite **controlar** la duplicidad, triplicidad de almacenamiento de espacio en disco (redundancia)
- Permite la **sincronización de datos**.
- Aumenta la **productividad**.
- **Centraliza** la información para diversos sistemas que trabajen sobre esa DB (permite la centralización de datos).
- Independientes de los programas y/o aplicaciones (genera independencia de los datos).
- Son **portables**. Basta con copiarlas, importarlas.

Base de datos • Desventajas

- Requiere de **mucho espacio** en disco.
- **Mantenimiento**.
- Suba de costos.
- Requieren de capacitación, asesoría y acompañamiento para enseñar su manejo.
- Si la BD crece mucho puede llegar a ponerse lenta, lo que afecta las búsquedas y la recuperación de información.
- Un fallo en la BD afecta a todo el entorno.

¿Qué es una Base de datos **relacional** y **no relacional**?

Relacional (SQL)

Se basan en la organización de la información en trozos pequeños, que se relacionan entre ellos mediante la relación de identificadores.

VS

No Relacional (No SQL)

A diferencia de las relacionales, no tienen un identificador que sirva de relación entre un conjunto de datos y otros.

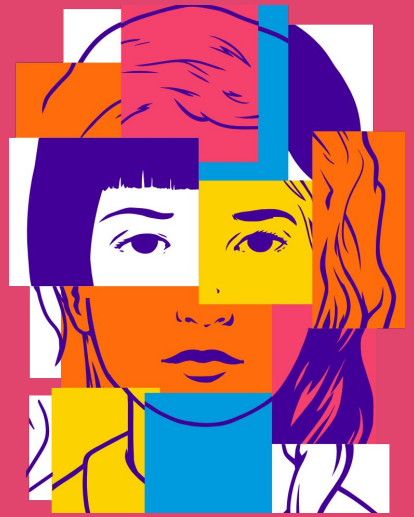
Empecemos con las no relacionales

VS

No Relacional (No SQL)

A diferencia de las relacionales, no tienen un identificador que sirva de relación entre un conjunto de datos y otros.

MongoDB

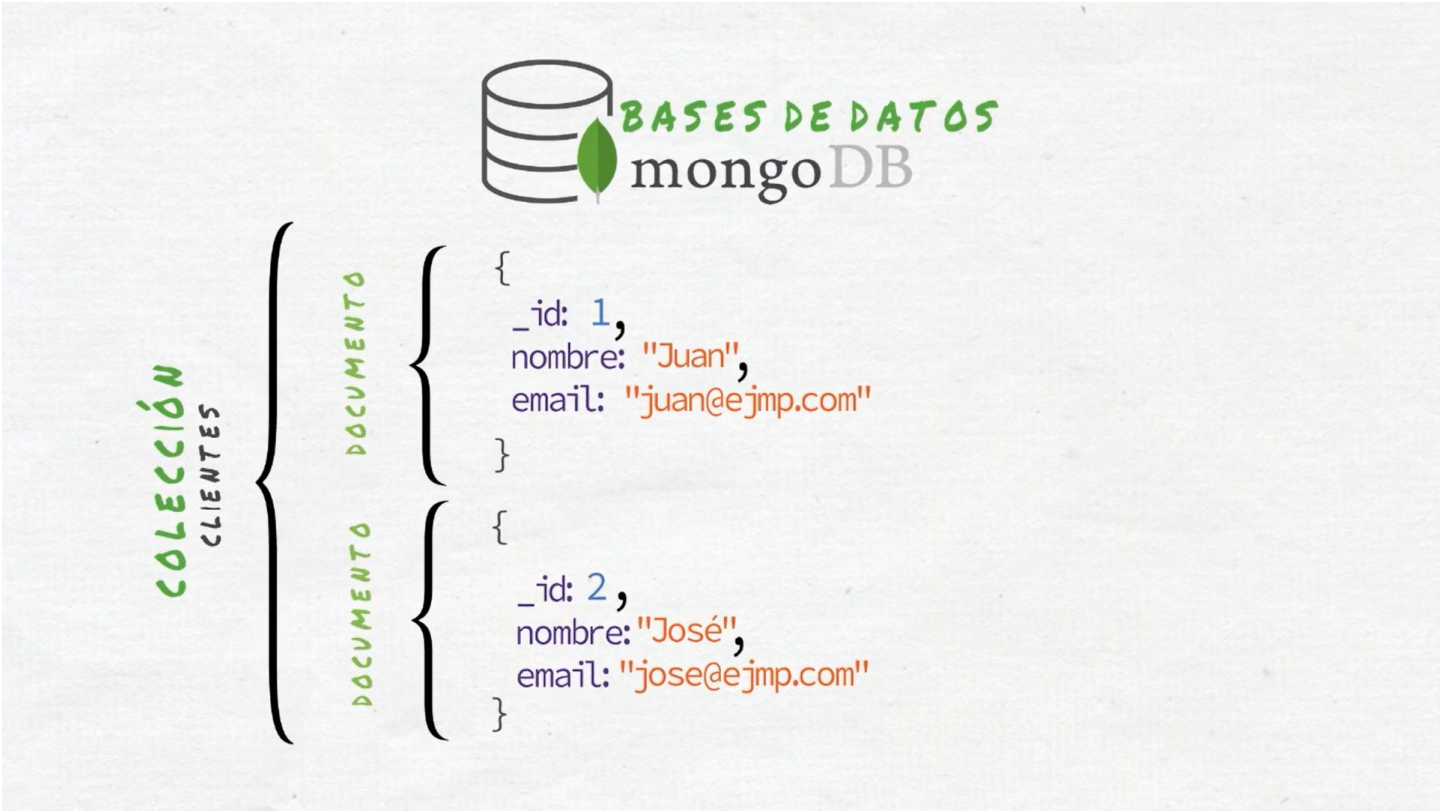


MongoDB
es un sistema para la gestión de bases de datos no relacional. Utiliza un modelo de documentos basado en JSON.

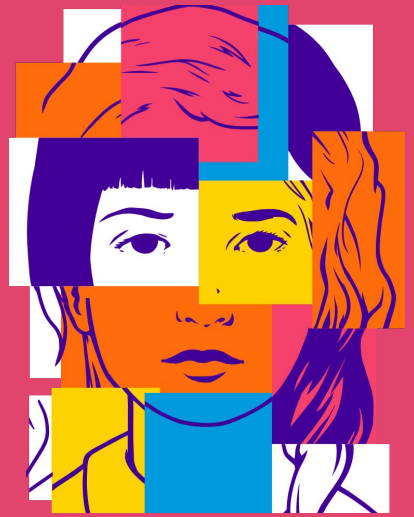
¿Cómo se compone una base de datos MongoDB?

MongoDB utiliza **colecciones** para agrupar información y dentro de ellas almacena **documentos**.

Composición



Bases de datos relacionales



Relacional (SQL)

Se basan en la organización de la información en trozos pequeños, que se relacionan entre ellos mediante la relación de identificadores.

VS

No Relacional (No SQL)

A diferencia de las relacionales, no tienen un identificador que sirva de relación entre un conjunto de datos y otros.

Composición

- **Tablas**: Permitirán almacenar nuestros datos.
- **Campos**: Es el nombre que identifica cada uno de los datos en tus entidades
- **Registros**: Es un conjunto de datos almacenados.
- **Consultas:** Permitirán acceder a los datos almacenados, ordenarlos y filtrarlos por diferentes criterios.
- **Vistas:** Mostrarán de forma más eficaz nuestros datos.

Composición (Tablas)

Campos

Registros

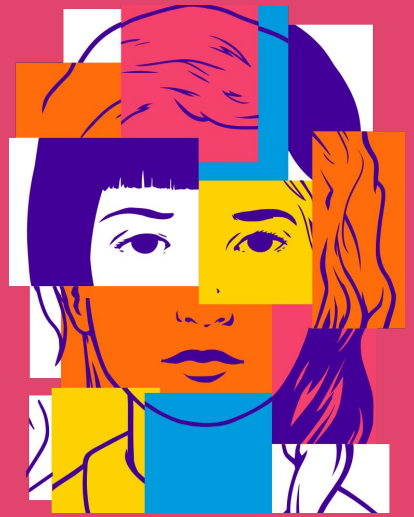
apellido	nombre	dni	etc
Einstein	Albert	12345678	etc
Turing	Alan Mathison	23456789	etc

Composición (Tablas)

CLAVE PRIMARIA
(PK)

id	apellido	nombre	dni	etc
1	Einstein	Albert	12345678	etc
2	Turing	Alan Mathison	23456789	etc

SQL



SQL

(Structured Query Language)
Lenguaje de Consulta Estructurado

Mediante este lenguaje vamos a poder hacer consultas a nuestras tablas en nuestra base de datos.

SQL

(Structured Query Language)
Lenguaje de Consulta Estructurado

Podemos:

- SQL puede hacer consultas en una base de datos
- SQL puede insertar registros en una base de datos
- SQL puede actualizar registros en una base de datos
- SQL puede eliminar registros de una base de datos
- SQL puede crear nuevas bases de datos
- SQL puede crear nuevas tablas en una base de datos
- SQL puede crear procedimientos almacenados en una base de datos
- SQL puede crear vistas en una base de datos
- SQL puede establecer permisos en tablas, procedimientos y vistas
-

SQL • Create

Permite crear tablas.

```
CREATE TABLE persona (
    id INT PRIMARY KEY AUTO_INCREMENT,
    apellido VARCHAR (60),
    nombre VARCHAR (60) NOT NULL,
    dni INT UNSIGNED NOT NULL
)
```

id	apellido	nombre	dni

SQL • Alter

Permite modificar la estructura de una tabla

```
ALTER TABLE persona
  ADD fecha_nac date NOT NULL
```

SQL • Alter

Permite modificar la estructura de una tabla

```
ALTER TABLE persona
 ADD fecha_nac date NOT NULL
```

```
ALTER TABLE persona
 DROP fecha_nac
```

SQL • Alter

Permite modificar la estructura de una tabla

```
ALTER TABLE persona
 ADD fecha_nac date NOT NULL
```

```
ALTER TABLE persona
 DROP fecha_nac
```

```
 ALTER TABLE persona

 MODIFY COLUMN dni varchar(10) NOT NULL;
```

SQL • Select

> Permite consultar y obtener información de nuestras tablas.

```
SELECT * FROM persona;
```

id	apellido	nombre	dni	pais
1	Einstein	Albert	123456	Alemania
2	Turing	Alan Mathison	234566	Inglaterra

SQL • Select

> Selecciono los campos que necesito

```
SELECT apellido, nombre from persona
```

id	apellido	nombre	dni	pais
1	Einstein	Albert	123456	Alemania
2	Turing	Alan Mathison	234566	Inglaterra

SQL • Select

Resultado

```
SELECT apellido, nombre from persona
```

id	apellido	nombre	dni	pais
1	Einstein	Albert	123456	Alemania
2	Turing	Alan Mathison	234566	Inglaterra

SQL • Select + Where

> Permite filtrar la información de nuestras tablas mediante operadores.

```
SELECT * FROM persona
    WHERE id = 1;
```

id	apellido	nombre	dni	pais
1	Einstein	Albert	123456	Alemania
2	Turing	Alan Mathison	234566	Inglaterra
3	Lovelace	Ada	989812	Inglaterra

SQL • Select + Where

Resultado.

```
SELECT * FROM persona
    WHERE id = 1;
```

id	apellido	nombre	dni	pais
1	Einstein	Albert	123456	Alemania
2	Turing	Alan Mathison	234566	Inglaterra
3	Lovelace	Ada	989812	Inglaterra

SQL • Select + Where

Operador =

```
SELECT apellido, nombre from persona
    WHERE pais = 'Inglaterra'
```

id	apellido	nombre	dni	pais
1	Einstein	Albert	123456	Alemania
2	Turing	Alan Mathison	234566	Inglaterra
3	Lovelace	Ada	989812	Inglaterra

SQL • Select + Where

Combinación = , >

```
SELECT apellido, nombre from persona
    WHERE pais = 'Inglaterra' and id > 2
```

id	apellido	nombre	dni	pais
1	Einstein	Albert	123456	Alemania
2	Turing	Alan Mathison	234566	Inglaterra
3	Lovelace	Ada	989812	Inglaterra

SQL • Select + Where

Resultado

```
SELECT * from persona
    WHERE pais = 'Inglaterra' and id > 2
```

id	apellido	nombre	dni	pais
1	Einstein	Albert	123456	Alemania
2	Turing	Alan Mathison	234566	Inglaterra
3	Lovelace	Ada	989812	Inglaterra

SQL • Select + Where (Operadores)

Distintos tipos de operadores que podemos usar para filtrar la información

```
=, !=, <, >, <= o >=
```

Podemos encadenar filtros con operaciones lógicas

```
AND, OR
```

SQL • Select + Where (Like)

Operador Like
El carácter % funciona como comodín para la búsqueda

```
SELECT apellido, nombre from persona
    WHERE pais LIKE 'Ing%'
```

id	apellido	nombre	dni	pais
1	Einstein	Albert	123456	Alemania
2	Turing	Alan Mathison	234566	Inglaterra
3	Lovelace	Ada	989812	Inglaterra

SQL • Select + Where (Like)

Resultado

```
SELECT apellido, nombre from persona
    WHERE pais LIKE 'Ing%'
```

id	apellido	nombre	dni	pais
1	Einstein	Albert	123456	Alemania
2	Turing	Alan Mathison	234566	Inglaterra
3	Lovelace	Ada	989812	Inglaterra

SQL • Insert

Permite insertar registros en nuestra tabla.

```
INSERT INTO persona
    VALUES (NULL, “Lovelace”, “Ada”, 989812, “Inglaterra”)
```

id	apellido	nombre	dni	pais
1	Einstein	Albert	123456	Alemania

SQL • Insert

Resultado

```
INSERT INTO persona
    VALUES (NULL, “Lovelace”, “Ada”, 989812, “Inglaterra”)
```

id	apellido	nombre	dni	pais
1	Einstein	Albert	123456	Alemania
2	Lovelace	Ada	989812	Inglaterra

SQL • Insert

Resultado

```
INSERT INTO persona (nombre, apellido, pais)
    VALUES (“Ada”,“Lovelace”,”Inglaterra”)
```

id	apellido	nombre	dni	pais
1	Einstein	Albert	123456	Alemania
2	Lovelace	Ada	null	Inglaterra

SQL • Update

Permite actualizar registros en nuestra tabla.

```
UPDATE persona
    SET pais = null
```

id	apellido	nombre	dni	pais
1	Einstein	Albert	123456	Alemania
2	Turing	Alan Mathison	234566	Inglaterra
3	Lovelace	Ada	989812	Inglaterra

SQL • Update

Permite actualizar registros en nuestra tabla.

```
UPDATE persona
    SET pais = null
```

id	apellido	nombre	dni	pais
1	Einstein	Albert	123456	null
2	Turing	Alan Mathison	234566	null
3	Lovelace	Ada	989812	null

SQL • Update + Where

> Al igual que el SELECT, podemos utilizar el WHERE para indicar qué registros queremos modificar

```
UPDATE persona
    SET pais = 'Colombia' WHERE id > 2
```

id	apellido	nombre	dni	pais
1	Einstein	Albert	123456	null
2	Turing	Alan Mathison	234566	null
3	Lovelace	Ada	989812	null

SQL • Update + Where

> Al igual que el SELECT, podemos utilizar el WHERE para indicar qué registros queremos modificar

```
UPDATE persona
    SET pais = 'Colombia' WHERE id > 2
```

id	apellido	nombre	dni	pais
1	Einstein	Albert	123456	null
2	Turing	Alan Mathison	234566	2
3	Lovelace	Ada	989812	Colombia

SQL • Delete

Permite eliminar registros en nuestra tabla.

```
DELETE FROM persona
```

id	apellido	nombre	dni	pais
1	Einstein	Albert	123456	Argentina
2	Turing	Alan Mathison	234566	Colombia
3	Lovelace	Ada	989812	México

SQL • Delete

Resultado

```
DELETE FROM persona
```

id	apellido	nombre	dni	cod_pais

SQL • Delete + Where

Podemos usar el WHERE para indicar que registros queremos eliminar.

```
DELETE FROM persona WHERE pais = 'Argentina'
```

id	apellido	nombre	dni	pais
1	Einstein	Albert	123456	Argentina
2	Turing	Alan Mathison	234566	México
3	Lovelace	Ada	989812	México

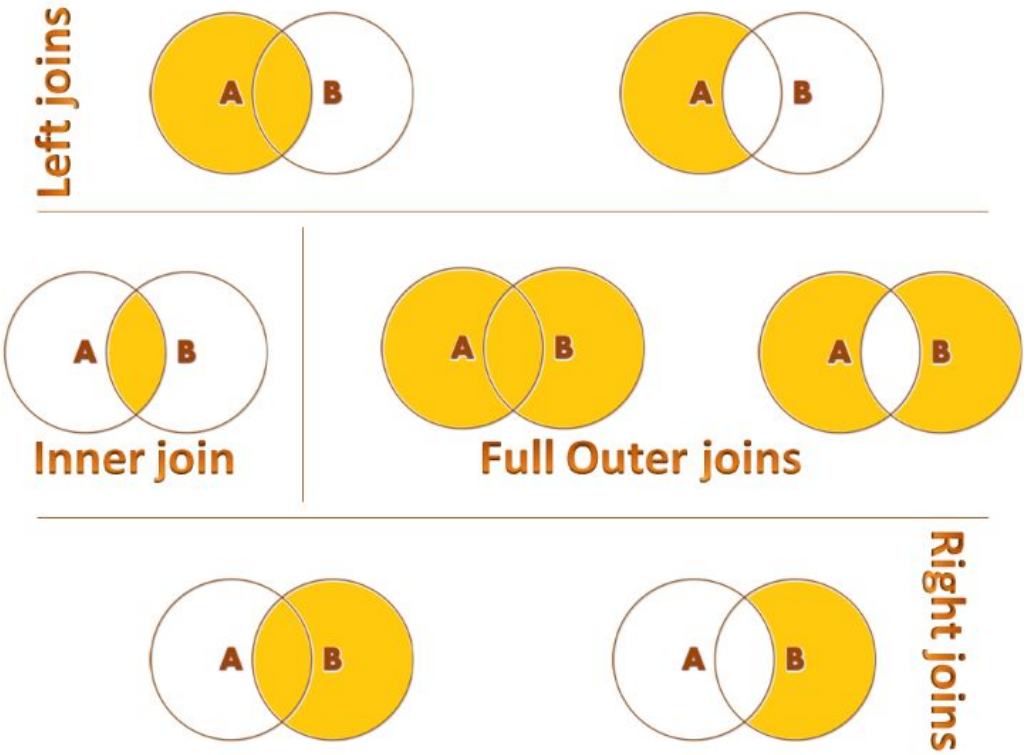
SQL • Delete + Where

Podemos usar el WHERE para indicar que registros queremos eliminar.

```
DELETE FROM persona
    WHERE cod_pais = 2
```

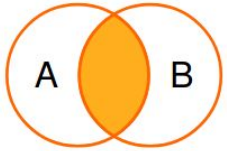
id	apellido	nombre	dni	cod_pais
1	Einstein	Albert	123456	1

Sentencias importantes con SQL

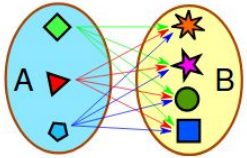


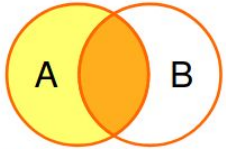
Joins del SQL

Select (campos)
From A Inner Join B
On A.Clave = B.Clave

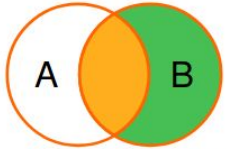


Select (campos)
From A Cross Join B

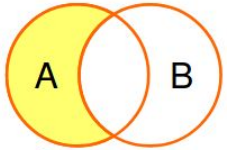




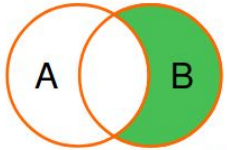
Select (campos)
From A Left Join B
On A.Clave = B.Clave



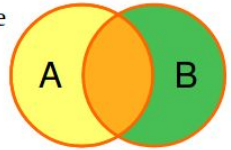
Select (campos)
From A Right Join B
On A.Clave = B.Clave



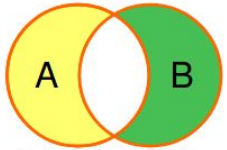
Select (campos)
From A Left Join B
On A.Clave = B.Clave
Where B.Clave is Null



Select (campos)
From A Right Join B
On A.Clave = B.Clave
Where A.Clave is Null



Select (campos)
From A Full Join B
On A.Clave = B.Clave

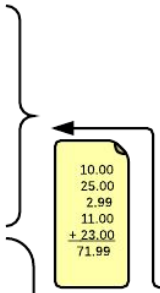


Select (campos)
From A Right Join B
On A.Clave = B.Clave
Where (A.Clave is Null) Or (B.Clave is Null)

SQL GROUP By Clause

transactions

id	paid_out	description
1	10.00	Amazon
2	25.00	Amazon
3	2.99	Amazon
4	11.00	Amazon
5	23.00	Amazon
6	25.00	Michaels
7	300.00	Michaels
8	1.00	Michaels
9	11.00	Michaels
10	2.99	Michaels
11	25.00	Michaels
12	1.00	Michaels
13	25.00	Michaels
14	10.00	Michaels




```
SELECT
    SUM(paid_out) AS paid_out_total,
    description
FROM transactions
GROUP BY description;
```

paid_out_total	description
71.99	Amazon
400.99	Michaels

Buenas prácticas

Índices

Un índice es una estructura sobre una columna de una tabla que permite localizar de forma rápida las filas que estás buscando.

Los índices añaden mucha velocidad a tus búsquedas pero ojo, no abuses de ellos, escribir muchos índices hará que tu base de datos crezca mucho en tamaño.

Elige cuales son tus campos claves de cada una de tus tablas y añade índices a ellos.

- Modelar entidades/tablas antes de crear.
- Usar nombres acordes a las entidades.
- Documentar
- A partir de la documentación crear las entidades/tablas.
- Guardar queries importantes o de uso diario.
- Tratar de automatizar carga de datos.

Modelar entidades/tablas antes de crear.

Puedes empezar en un boceto con papel y lápiz o utilizar alguna herramienta como:

- draw.io : online (open source)
- lucidchart : online (licencia limitada)
- visio parte del paquete office (licencia Microsoft)

Usar nombres acordes a las entidades.

Esto para su entendimiento.
Recordemos que no estamos sólos desarrollando.

A partir de la documentación, crear las entidades/tablas.

Manos a la obra con la creación.

Guardar querys importantes o de uso diario.

Muchas veces, a la hora de hacer pruebas, vamos a necesitar tener nuestros querys para hacer consultas.
Por esto, deberíamos tener guardados para evitar su reescritura.

Tratar de automatizar la carga de datos.

Esta carga de datos nos servirá para los pasajes a otros ambientes de testing y producción.

¿Cómo? Mediante un archivo con formato **csv**.

Podemos tener un módulo administrativo o una api para la carga del archivo y ejecuten los inserts.

¡BREAK!

Ph. Credit: Drew Coffmann

Programamos

todos/as

ACÁMICA