

# Tecnología II: Manipulación de datos



red.es



*"El FSE invierte en tu futuro"*  
**Fondo Social Europeo**

# Índice

## 1. Conceptos básicos

- 1.1 La importancia de documentarse
- 1.2 Tipos de datos: string, integer y float
- 1.3 Booleanos
- 1.4 Variables
- 1.5 Funciones
- 1.6 Condicionales
- 1.7 Bucles
- 1.8 Comentarios
- 1.9 Expresiones regulares

## 2. JS: Objetos JSON y API

- 2.1 JS: Manipulación de objetos
- 2.2 JS: Eventos y capas de datos
- 2.3 API REST

## 3. Bases de datos SQL

- 3.1 SQL vs NoSQL
- 3.2 SGBD
- 3.3 CRUD
- 3.4 Primary key y foreign key
- 3.5 CREATE TABLE
- 3.6 INSERT, SELECT, UPDATE,DELETE
- 3.7 Operadores: AND, OR
- 3.8 Condicionales: Where, between like y negación
- 3.9 CASE
- 3.10 Tablas Cruzadas

## 4. SQL y Reporting

- 4.1 Campos calculados
- 4.2 Funciones GDS

## 5. Ejercicios

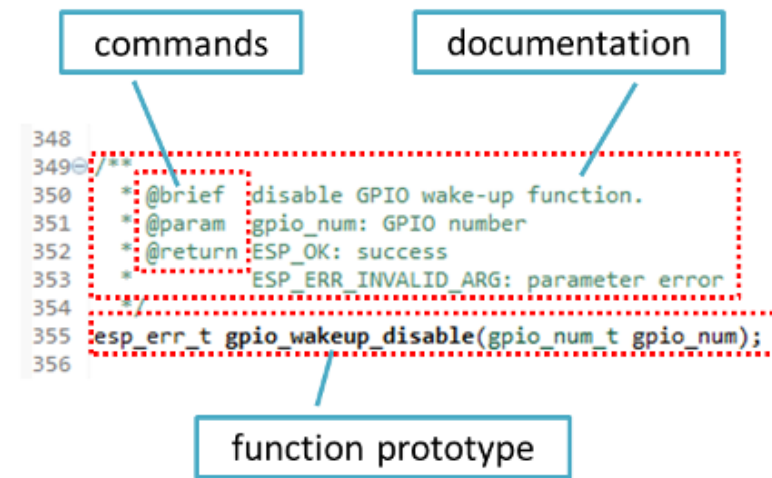
# 1. Conceptos básicos

## 1.1 La importancia de documentarse

La documentación de los programas es un aspecto sumamente importante, tanto en el desarrollo de la aplicación como en el mantenimiento de la misma. Es un error muy común no leer ni documentar programas y perder la posibilidad de la reutilización de parte del programa en otras aplicaciones, sin necesidad de conocerse el código al dedillo.

La documentación de un programa empieza a la vez que la construcción del mismo y finaliza justo antes de la entrega del programa o aplicación al cliente.

La documentación de un programa puede ser interna y externa. La documentación interna es la contenida en líneas de comentarios. La documentación externa incluye análisis, diagramas de flujo y/o pseudocódigos, manuales de usuario con instrucciones para ejecutar el programa y para interpretar los resultados.



## 1.2 Tipos de datos

Casi todos los lenguajes de programación que vamos a usar como profesionales de los datos comparten unos tipos de datos básicos:

### **STRING**

Una string hace referencia a cadenas de texto. Sus valores se suelen representar siempre situados entre dobles comillas.

“una cadena de texto”

### **NÚMEROS INT**

Hace referencia a números enteros (integer).

“745”

### **NÚMEROS FLOAT**

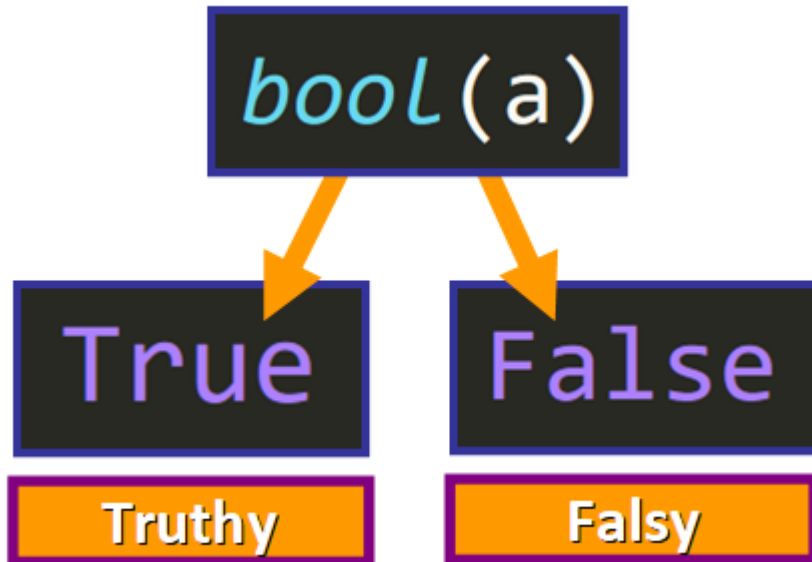
Hace referencia a números decimales

“981.4”

## 1.3 Booleanos

También en muchos lenguajes de programación existe el tipo de dato **booleano**, boolean o bool que indican que una variable solo podrá tener dos valores: **verdadero y falso**.

Son muy utilizados para construir funciones y consultas basadas en condiciones.



```
> String(true) == "true"
< true
> Boolean("true")
< true
> "true"==true
< false
>
```

## 1.4 Variables

Una variable **es donde se almacenan y se recuperan los datos de un programa**. Así de simple. En programación, la utilizamos para guardar datos y estados, asignar ciertos valores de variables a otras, representar valores de expresiones matemáticas y mostrar valores por pantallas.

```
#String
coro_cancion = "Tommy used to work on the docks"
#Int
edad = 22
#Float
valor_pi = 3.1416
#Bool
estado = True
```

```
variables.js  x
1  var cadena="hola esto es un String"
2  var entero=5
3  var decimal=4.5
4  var caracter='c'
5  var booleana=true
6  var array=['elemnto1','elemento2','elemento3']
```

## 1.5 Funciones

Las **funciones** son un elemento muy utilizado en la programación. Empaquetan y ‘aíslan’ del resto del programa una parte de código que realiza alguna tarea específica.

Son por tanto un conjunto de instrucciones que ejecutan una tarea determinada y que hemos encapsulado en un formato estándar para que nos sea muy sencillo de manipular y reutilizar.

AVG() - La **media de los valores**

COUNT() - El **número de filas**

MAX() - El **valor más grande**

MIN() - El **valor más pequeño**

SUM() - La **suma de los valores**



## 1.6 Condicionales

Un condicional en la programación es una sentencia o grupo de sentencias que puede ejecutarse o no en función del valor de una condición.

Los tipos más conocidos de condicionales son el SI (IF) y el SEGÚN (case o switch), aunque también podríamos mencionar al lanzamiento de errores como una alternativa más moderna para evitar el "anidamiento" de condicionales.

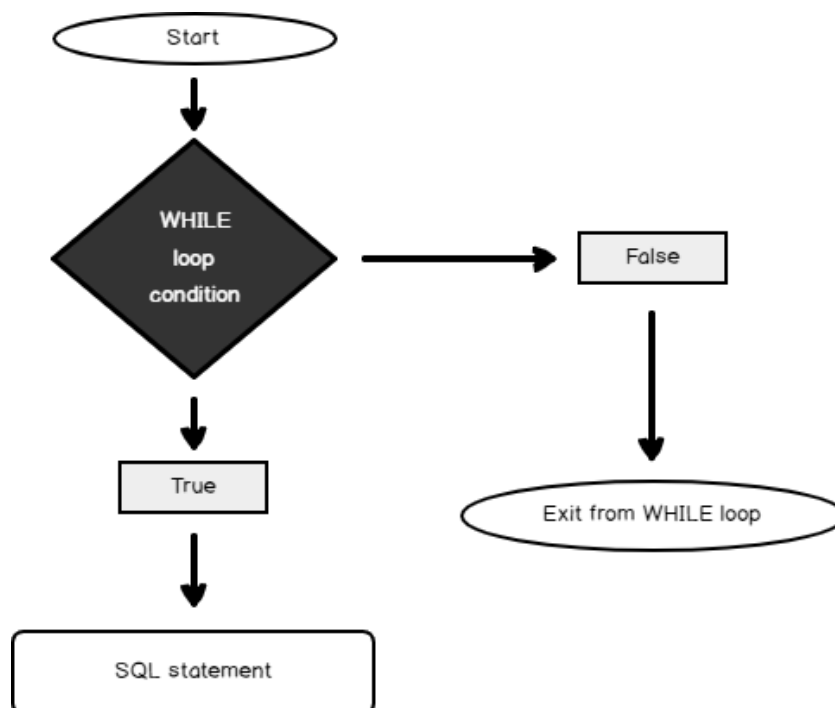


```
SELECT
    column_name(s)
    CASE condition_field
        WHEN condition_field_value_1 THEN result_1
        WHEN condition_field_value_2 THEN result_2
        ...
    ELSE
    END AS
FROM
    table_name;
```

## 1.7 Bucles

Los bucles o ciclos de programación se utilizan en los programas de código para establecer sentencias o trozos de código que se repiten o se iteran. Este se repita hasta que una condición deja de cumplirse y da lugar al siguiente trozo de código.

### EJEMPLOS MÁS UTILIZADOS: WHILE, FOR, FOREACH



```
let ranks = ['A', 'B', 'C'];  
for (let i = 0; i < ranks.length; i++) {  
  console.log(ranks[i]);  
}
```

## 1.8 Expresiones regulares

Una expresión regular es una cadena de caracteres utilizada para describir o encontrar patrones dentro de otras cadenas o campos, en base al uso de delimitadores y ciertas reglas de sintaxis.

Hay muchas expresiones regulares. Es imposible saberlas todas, ni mucho menos. Sin embargo, para optimizar y facilitar el trabajo de analítica, es bueno conocer algunos ejemplos muy utilizados a la hora de crear patrones en la extracción de datos.

.

Cualquier carácter (menos salto de línea).

^

Que empiece por. Ej. `^wwGA: ANALÍTICA AVANZADA`

\$

Que termine por. Ej. `/ $` (que termine con la barra de directorio)

**[a-z]**

Cualquier letra de la a a la z (minúsculas). Otros ej.: `[a-m]` de la a a la m.

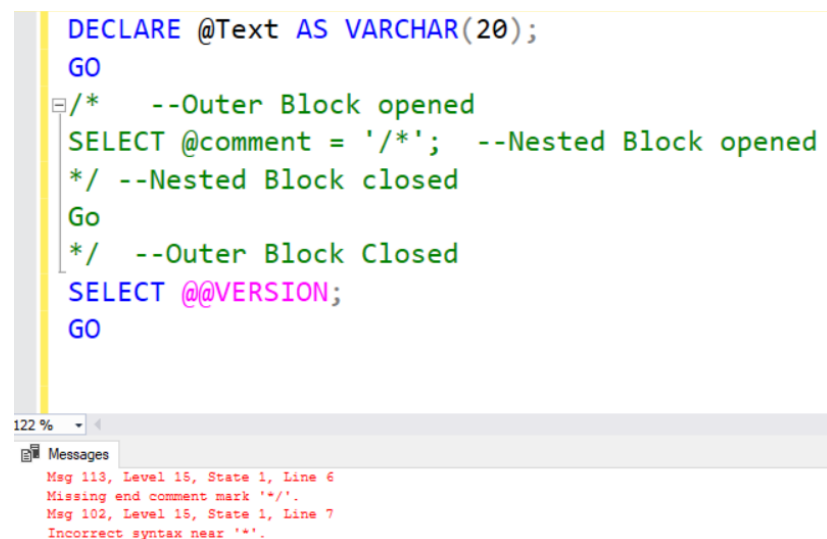
**[0-9]**

Cualquier número del 0 al 9. Otro ej.: `[1-3]`. No confundir con `{1,3}` que es el número de repeticiones de otro carácter, no el carácter numérico en sí mismo.

## 1.9 Comentarios

De la misma forma que habíamos hablado de la importancia de la documentación en todos los lenguajes y los programas, también es imprescindible incorporar comentarios a nuestro código a la hora de escribirlo.

Unos buenos comentarios van a permitir que todo el equipo de profesionales pueda leer y entender el código con mayor rapidez y agilidad. Asimismo, incluir comentarios permite y facilita la comprensión del código después de mucho tiempo sin consultarlo.

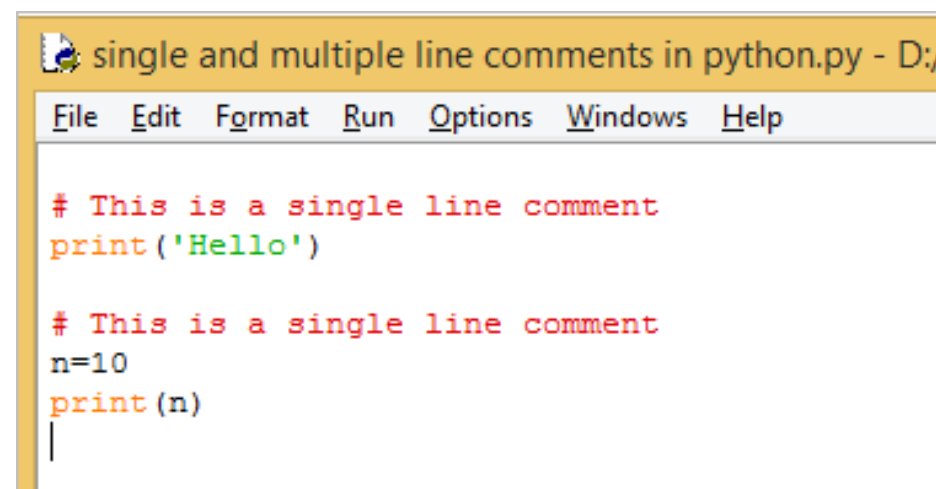


```
DECLARE @Text AS VARCHAR(20);
GO
/*  --Outer Block opened
SELECT @comment = '/*';  --Nested Block opened
*/ --Nested Block closed
Go
*/  --Outer Block Closed
SELECT @@VERSION;
GO
```

122 %

Messages

Msg 113, Level 15, State 1, Line 6  
Missing end comment mark '\*/'.  
Msg 102, Level 15, State 1, Line 7  
Incorrect syntax near '/\*'.



```
single and multiple line comments in python.py - D:/I

File Edit Format Run Options Windows Help

# This is a single line comment
print('Hello')

# This is a single line comment
n=10
print(n)
|
```

## 2. Objetos JSON y API Rest

## 2.1 JS y manipulación de datos

**JavaScript Object Notation (JSON)** es un formato basado en texto estándar para representar datos estructurados en la sintaxis de objetos de JavaScript. <https://www.json.org/json-es.html>

Es comúnmente utilizado para transmitir datos en aplicaciones web (por ejemplo: enviar algunos datos desde el servidor al cliente, así estos datos pueden ser mostrados en páginas web, o vice versa).

```
▼ {  
  "id": 1001,  
  "type": "donut",  
  "name": "Cake",  
  "description": "https://www.jqueryscript.net",  
  "price": 2.55,  
  ▶ "available": { 2 items },  
  ▼ "topping": [  
    ▼ {  
      "id": 5001,  
      "type": "None"  
    },  
    ▼ {  
      "id": 5002,  
      "type": "Glazed"  
    }  
  ]  
}
```

```
superHeroes.homeTown  
superHeroes['active']
```

```
superHeroes['members'][1]['powers'][2]
```

## 2.2 Eventos y capas de datos

Es muy común encontrarse en herramientas de analítica de datos de páginas webs y apps objetos JSON que lanzan eventos en cada acción que se quiera recoger.

Estas acciones o eventos llevan incorporados muchos valores, llaves o parámetros adicionales a los que podemos acceder y extraer fácilmente para nuestro objetivo final.

Data Layer values after this Message:

```

1 {
2   gtm: {start: 1519042442115, uniqueEventId: 16},
3   event: 'gtm.dom',
4   ecommerce: {
5     purchase: {
6       actionField: {
7         id: '3000021331',
8         revenue: 87,
9         tax: '7.7300',
10        shipping: '0.0000',
11        coupon: '',
12        coupon_value: -87,
13        customer_id: 'd57ed3e4'
14      },
15      products: [
16        {
17          id: '600100100-8099',
18          name: '',
19          price: '99.0000',
20          quantity: '1.0000',
21          price_with_tax: 107.79,
22          price_without_tax: 99
23        },

```

```

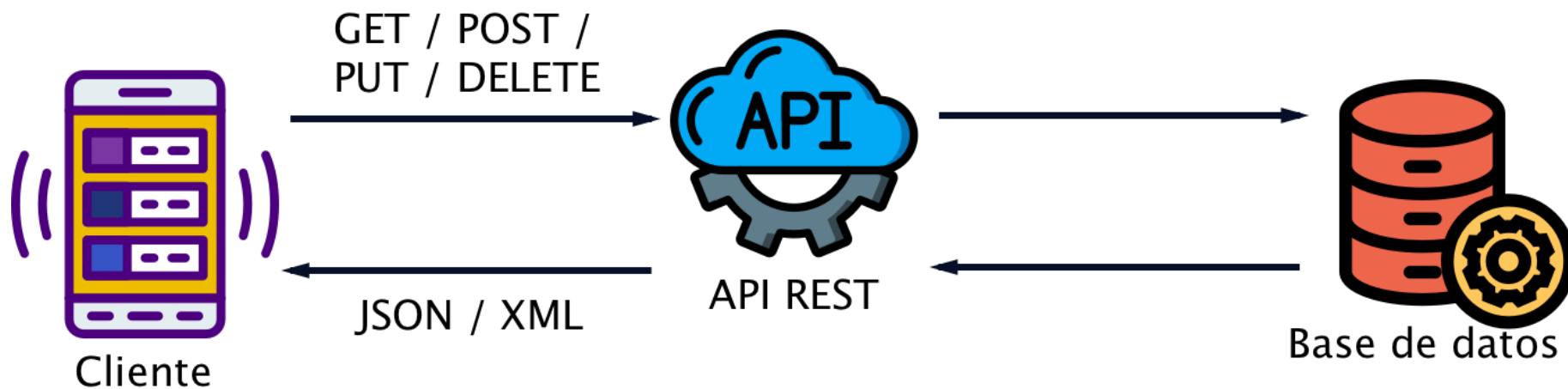
<script>
dataLayer = [{
  'transactionId': 'A123456',
  'transactionTotal': 175.00,
  'transactionTax': 0.00,
  'transactionShipping': 25.00,
}];
transactionProducts = [{
  'sku': 'MLP-RD123',
  'name': 'Rainbow Dash Hoodie',
  'category': 'Hoodies',
  'price': 50.00,
  'quantity': 1
},{
  'sku': 'MLP-RD456',
  'name': 'Rainbow Dash Super Hoodie',
  'category': 'Hoodies',
  'price': 100.00,
  'quantity': 1
}];
dataLayer.push({'transactionProducts':transactionProducts});
</script>

```

## 2.3 JS y API REST

Una **API**, o *interfaz de programación de aplicaciones*, es un conjunto de reglas que definen cómo pueden las aplicaciones o los dispositivos conectarse y comunicarse entre sí.

Una **API REST** es una API que cumple los principios de diseño del estilo de arquitectura REST o transferencia de estado representacional. Por este motivo, las API REST a veces se conocen como API RESTful.





# 3. Bases de datos

## SQL

## 3.1 SQL vs NoSQL

La gestión de las bases de datos es por lo tanto fundamental para todos los trabajos de estas áreas. Un sistema de gestión de bases de datos (SGBD) es un programa que permite a uno o varios usuarios acceder a una base de datos. Permite manejar los accesos diferenciados (identificación, seguridad) y permite interpretar las búsquedas para ingresar, modificar, invertir o suprimir datos. Se pueden diferenciar 2 grandes familias de SGBD : los SGBD SQL y los SGBD NoSQL.

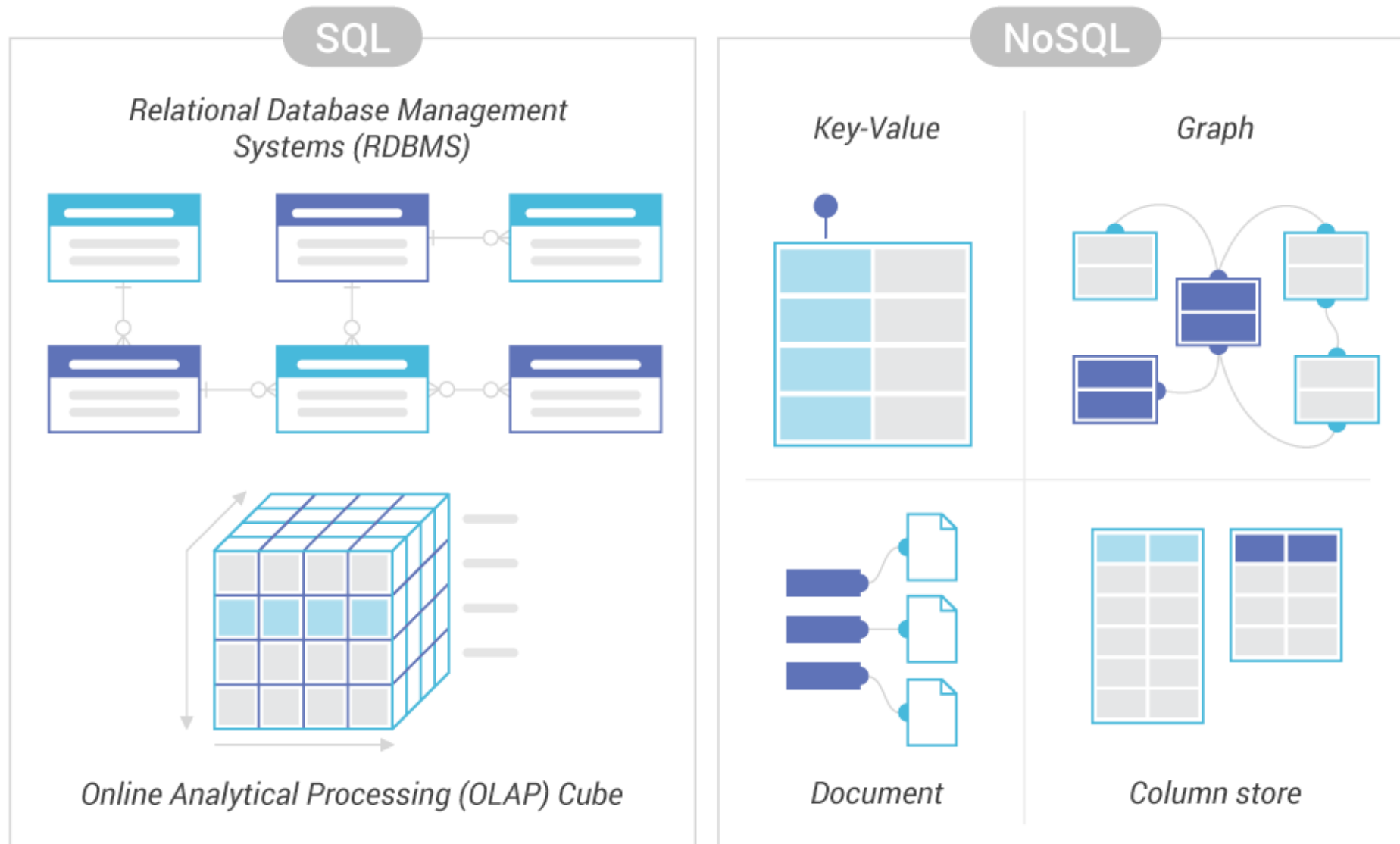
**SQL:** Relacionales. Esquema fijo y datos clasificados.

- Tipo y validez de datos muy importante
- Necesidad recurrente de escritura y modificaciones de datos sobre elementos específicos (SQL permite modificar fácilmente líneas específicas)
- Necesidad de búsquedas complejas

**NoSQL:** No relacionales. Modulares. No necesitan esquema fijo.

- Necesidad de múltiples búsquedas de lectura.
- Grandes conjuntos de datos (Big Data)
- Datos distribuidos (varias fuentes)

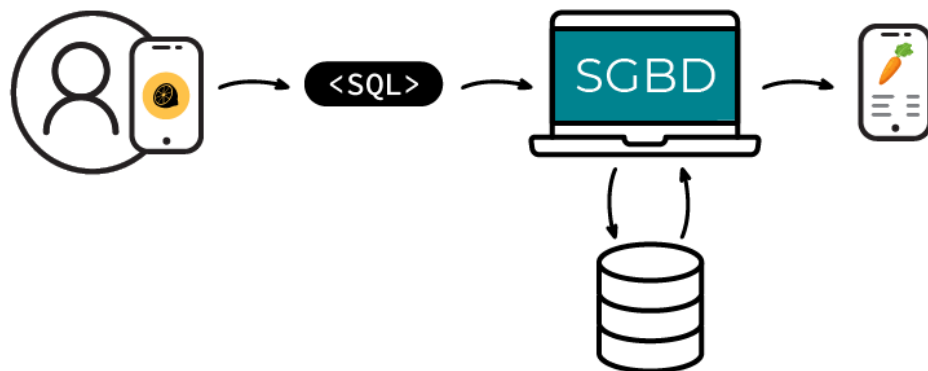
## 3.1 SQL vs NoSQL



## 3.2 SGBD

Un sistema de gestión de bases de datos (SGBD, por sus siglas en inglés) o **DataBase Management System** (DBMS) es una colección de **software muy específico, orientado al manejo de base de datos**, cuya función es servir de **interfaz** entre la base de datos, el usuario y las distintas aplicaciones utilizadas.

Su uso permite realizar un mejor control a los administradores de sistemas y, por otro lado, también obtener mejores resultados a la hora de realizar consultas que ayuden a la gestión empresarial mediante la generación de la tan perseguida ventaja competitiva.



## 3.3 CRUD

En programación solemos usar el término CRUD para referirnos a las operaciones básicas que puedes realizar sobre un conjunto de datos y por sus siglas son:

Crearlos, ya sabes, nuevos registros, cuando hablamos de bases de datos esto quiere decir insertar información.

Leerlos, r por Read, esto quiere decir consultar esa información, ya sea un registro o una colección de estos registros.

Actualizarlos, u por Update, que significa tomar un registro que ya existe en la base de datos y modificar alguna de las columnas.

Por último eliminar registros, d por Delete, que significa tomar un registro y quitarlo del almacén.



## 3.4 Primary key y foreign key

La restricción **PRIMARY KEY** identifica de forma única cada registro en una tabla.

Las claves primarias deben contener valores únicos y no pueden contener valores **NULL**.

Una tabla solo puede tener una clave principal, que puede consistir en campos simples o múltiples.

### SQL PRIMARY KEY EN CREATE TABLE

El siguiente SQL crea una **PRIMARY KEY** en la columna «ID» cuando se crea la tabla “Personas”

```
CREATE TABLE Personas ( ID int NOT NULL PRIMARY KEY, Apellidos varchar(255) NOT NULL,  
Nombre varchar(255), Edad int );
```

## 3.4 Primary key y foreign key

Una **FOREIGN KEY** es una clave (campo de una columna) que sirve para relacionar dos tablas.

El campo **FOREIGN KEY** se relaciona o vincula con la **PRIMARY KEY** de otra tabla de la bbdd

La tabla secundaria es la que contiene la **FOREIGN KEY** y la tabla principal contiene la **PRIMARY KEY**.

La **FOREIGN KEY** es una restricción que no permite que se agreguen o inserten datos que no válidos en la columna de foreign key, ya que los valores que se van a insertar deben ser valores que se encuentren o ya estén en la tabla con la que se quiere relacionar.

El siguiente comando crea una FOREIGN KEY en la columna “ID PERSONA” cuando se crea la tabla pedidos.

```
CREATE TABLE Pedidos ( PedidoID int NOT NULL PRIMARY KEY, NumeroPedido int NOT NULL, PersonalID int  
FOREIGN KEY REFERENCES Personas(PersonalID) );
```

## 3.5 CREATE TABLE

Las **tablas se utilizan para almacenar datos en la base de datos**. Las tablas tienen nombres únicos dentro de una base de datos y un esquema.

Cada tabla contiene una o más columnas y cada columna tiene un tipo de datos asociado que define el tipo de datos que puede almacenar, por ejemplo, números, cadenas o datos temporales.

```
SQL> CREATE TABLE CUSTOMERS(
  ID    INT                NOT NULL,
  NAME  VARCHAR (20)       NOT NULL,
  AGE   INT                NOT NULL,
  ADDRESS CHAR (25) ,
  SALARY DECIMAL (18, 2),
  PRIMARY KEY (ID)
);
```

| Field   | Type          | Null | Key | Default | Extra |
|---------|---------------|------|-----|---------|-------|
| ID      | int(11)       | NO   | PRI |         |       |
| NAME    | varchar(20)   | NO   |     |         |       |
| AGE     | int(11)       | NO   |     |         |       |
| ADDRESS | char(25)      | YES  |     | NULL    |       |
| SALARY  | decimal(18,2) | YES  |     | NULL    |       |



## 3.6 INSERT, SELECT, UPDATE Y DELETE

### **INSERT**

Nos permite insertar nuevos datos en nuestra tabla

```
INSERT INTO nombre_tabla  
VALUES (valor1, valor2, valor3, .)
```

### **SELECT**

Nos permite seleccionar ciertos datos de nuestra tabla. Especialmente útil en el análisis de datos

```
SELECT * FROM nombretabla
```

.

### **UPDATE**

Permite actualizar valores presentes en la base de datos o tabla.

```
UPDATE nombre_tabla  
SET columna1 = valor1, columna2 = valor2  
WHERE columna3 = valor3
```

### **DELETE**

Se utiliza para borrar cierto valor en la base de datos.

```
DELETE * FROM nombre_tabla;
```

## 3.7 OPERADORES: AND y OR

Los **operadores AND y OR** se utilizan para filtrar resultados con 2 condiciones.

El operador **AND** mostrará los resultados cuando se cumplan las 2 condiciones.

Condición1 AND condición2

El operador **OR** mostrará los resultados cuando se cumpla alguna de las 2 condiciones.

Condicion1 OR condicion2

| nombre  | apellido1 | apellido2 |
|---------|-----------|-----------|
| ANTONIO | PEREZ     | GOMEZ     |
| ANTONIO | GARCIA    | BENITO    |

```
SELECT * FROM personas  
WHERE nombre = 'ANTONIO'  
OR apellido1 = 'GARCIA'
```

```
SELECT * FROM personas  
WHERE nombre = 'ANTONIO'  
AND (apellido1 = 'GARCIA' OR apellido1 = 'LOPEZ')
```

## 3.8 Condicionales: WHERE

La cláusula WHERE se utiliza para hacer filtros en las consultas, es decir, seleccionar solamente algunas filas de la tabla que cumplan una determinada condición.

El valor de la condición debe ir entre comillas simples ".

Por ejemplo:

Seleccionar las personas cuyo nombre sea ANTONIO

| nombre  | apellido1 | apellido2 |
|---------|-----------|-----------|
| ANTONIO | PEREZ     | GOMEZ     |
| ANTONIO | GARCIA    | BENITO    |

```
SELECT * FROM personas  
WHERE nombre = 'ANTONIO'
```

## 3.8 Condicionales: Between

El operador BETWEEN se utiliza en la cláusula WHERE para seleccionar valores entre un rango de datos.

Sintaxis de SQL BETWEEN

SELECT columna

FROM tabla WHERE columna

BETWEEN valor1 AND valor2

Ejemplo de SQL BETWEEN

Dada la siguiente tabla 'Products'

| ProductID | ProductName                  | SupplierID | CategoryID | Unit                | Price |
|-----------|------------------------------|------------|------------|---------------------|-------|
| 1         | Chais                        | 1          | 1          | 10 boxes x 20 bags  | 18    |
| 2         | Chang                        | 1          | 1          | 24 - 12 oz bottles  | 19    |
| 3         | Aniseed Syrup                | 1          | 2          | 12 - 550 ml bottles | 10    |
| 4         | Chef Anton's Cajun Seasoning | 1          | 2          | 48 - 6 oz jars      | 22    |
| 5         | Chef Anton's Gumbo Mix       | 1          | 2          | 36 boxes            | 21.35 |

SELECT \* FROM Products

WHERE Price BETWEEN 18 AND 22;

Nos va a devolver 3 filas (18, 19 y 22)

## 3.8 Condicionales: like

El operador **LIKE** se utiliza en la cláusula WHERE para buscar por un patrón.

Sintaxis de SQL LIKE

```
SELECT columna(s) FROM tabla WHERE columna LIKE '%patron%'
```

Ejemplos del uso de SQL LIKE

Dada la siguiente tabla 'personas'

| nombre  | apellido1 | apellido2 |
|---------|-----------|-----------|
| ANTONIO | PEREZ     | GOMEZ     |
| ANTONIO | GARCIA    | RODRIGUEZ |
| PEDRO   | RUIZ      | GONZALEZ  |

Si quiero seleccionar los nombres que empiezan por 'AN' en la tabla 'personas', ejecutaría el comando siguiente:

```
SELECT * FROM personas  
WHERE nombre LIKE 'AN%'
```

## 3.9 Condicionales: CASE

La sentencia case en el SQL se utiliza principalmente en un caso con expresiones de igualdad. La sentencia de SQL CASE generalmente está dentro de una lista de selección para alterar la salida. Lo que realiza es evaluar una lista de condiciones y devuelve una de las múltiples expresiones de resultado posibles.

CASE

WHEN *condition1* THEN *result1*

WHEN *condition2* THEN *result2*

WHEN *conditionN* THEN *resultN*

ELSE *result*

END;

SELECT OrderID, Quantity,

CASE

WHEN Quantity > 30 THEN 'The quantity is greater than 30'

WHEN Quantity = 30 THEN 'The quantity is 30'

ELSE 'The quantity is under 30'

END AS QuantityText

FROM OrderDetails;

## 3.10 Tablas cruzadas

### LEFT JOIN

La sentencia **LEFT JOIN** combina los valores de la primera tabla con los valores de la segunda tabla. Siempre devolverá las filas de la primera tabla, incluso aunque no cumplan la condición.

```
SELECT t.col1, t.col2,...,t.col from table1 t left join table2 a on t.id = a.id;
```

### RIGHT JOIN

La sentencia **RIGHT JOIN** combina los valores de la primera tabla con los valores de la segunda tabla. Siempre devolverá las filas de la segunda tabla, incluso aunque no cumplan la condición.

```
SELECT t.col1, t.col2,...,t.col from table1 t right join table2 a on t.id = a.id;
```

### INNER JOIN

La sentencia **INNER JOIN** es el sentencia JOIN por defecto, y consiste en combinar cada fila de una tabla con cada fila de la otra tabla, seleccionado aquellas filas que cumplan una determinada condición.

```
SELECT t.col1, t.col2,...,t.col from table1 t inner join table2 a on t.id = a.id;
```

# 4. SQL y reporting



## 4.1 Campos calculados

### Ejemplos

`SUM(Cantidad)`: suma los valores del campo Cantidad.

`PERCENTILE(Usuarios por día, 50)`: devuelve el percentil 50 de todos los valores del campo Usuarios por día.

`ROUND(Ingresos por usuario, 0)`: redondea el valor del campo Ingresos por usuario a un número sin decimales (es decir, con 0 posiciones decimales).

`SUBSTR(Campaign, 1, 5)`: devuelve los 5 primeros caracteres del valor del campo Campaña.

`REGEXP_EXTRACT(Cadena de valores delimitados por barras verticales, R'^([a-zA-Z_]*) (\\|)')`: extrae el primer valor de una cadena delimitada por barras verticales.

`DATETIME_DIFF(Start Date, End Date)`: calcula el número de días que hay entre la fecha de inicio y la de finalización.

`PARSE_DATETIME("%d/%m/%Y %H:%M:%S", DateTimeText)`- crea una fecha a partir de un campo de texto.

`TOCITY(ID de criterio, "CRITERIA_ID")`: muestra el nombre de la ciudad asociada a un ID de criterio de segmentación geográfica de Google Ads que sea válido.

**MÁS INFORMACIÓN SOBRE CAMPOS CALCULADOS EN LOOKER STUDIO**

## 4.2 Funciones

| Tipo ^      | Nombre        | Descripción   | Sintaxis   |
|-------------|---------------|---|--|
| Condicional | CASE          | <p>Evalúa la condición <code>condition</code> de cada cláusula WHEN sucesiva y devuelve el primer resultado <code>result</code> donde <code>condition</code> es true. Las cláusulas WHEN y ELSE restantes no se evalúan. Si todas las condiciones son false o NULL, devuelve <code>else_result</code> (si está presente). Si no está presente, devuelve NULL. <a href="#">Más información</a></p> | <pre>CASE   WHEN condition   THEN result   [WHEN condition   THEN result]   [...]   [ELSE   else_result] END</pre>                                   |
| Condicional | CASE (simple) | <p>Compara <code>input_expression</code> con el valor <code>expression_to_match</code> de cada cláusula WHEN sucesiva y devuelve el primer resultado <code>result</code> cuando esta comparación devuelve true. <a href="#">Más información</a></p>   | <pre>CASE input_expression   WHEN   expression_to_match   THEN result   [WHEN   expression_to_match   THEN result]   [...]   [ELSE result] END</pre> |

**CONSULTA TODAS LAS FUNCIONES EN LOOKER STUDIO**

# 5. Ejercicios

## 5. Ejercicios

### E.1

Instalación de MySQL y creación de dos tablas: Usuarios y Productos.

### E.2

Ejecutar al menos dos cruzados de tablas para asignar los productos creados por cada uno de los usuarios teniendo en cuenta el ID.

### E.3

Mejora el dashboard que hiciste ayer con expresiones regulares y campos calculados en Looker Studio



red.es

Centro de  
Referencia Nacional  
en Comercio Electrónico  
y Marketing  
CRN  
Digital



UNIÓN EUROPEA

*"El FSE invierte en tu futuro"*

**Fondo Social Europeo**

