

**BBDD**

## UT4: DML (parte 2)

José Antonio  
Benítez Chacón



# Índice

1

ORACLE

2

Combinaciones de consultas

3

Vistas

4

SQL Avanzado:  
STRING\_AGG

5

SQL Avanzado: RANK

6

SQL Avanzado: LAG y LEAD

7

SQL Avanzado: WITH

8

SQL Avanzado: WITH  
Recurso

9

SQL Avanzado: PIVOT



# 1. ORACLE

**Oracle Database** es un sistema de gestión de bases de datos relacional (RDBMS) utilizado ampliamente en entornos empresariales por su robustez, escalabilidad y seguridad. A diferencia de MariaDB, que es de código abierto y está basado en MySQL, Oracle introduce algunas diferencias clave en la sintaxis SQL, los tipos de datos y la gestión de usuarios.

## Principales diferencias entre MariaDB y Oracle

### 1) Tipos de datos

**VARCHAR** vs. **VARCHAR2**: En Oracle, se usa VARCHAR2(n) en lugar de VARCHAR(n). VARCHAR2 es preferido porque maneja de forma eficiente la memoria.

**DECIMAL** vs. **NUMBER**: Oracle usa NUMBER(p,s) en lugar de DECIMAL(p,s), permitiendo mayor precisión numérica.

**TEXT** vs. **CLOB**: Para almacenar grandes bloques de texto, Oracle usa CLOB en lugar de TEXT.

**DATETIME** vs. **DATE**: En Oracle, DATE almacena fecha y hora, mientras que en MariaDB se usa DATETIME.



# 1. ORACLE

## 2) Generación de Claves Primarias

En MariaDB, se usa AUTO\_INCREMENT en columnas PRIMARY KEY.

En Oracle, se emplean **secuencias** y **GENERATED AS IDENTITY** (desde Oracle 12c).

```
CREATE TABLE empleados (  
    id NUMBER GENERATED AS IDENTITY PRIMARY KEY,  
    nombre VARCHAR2(100)  
);
```

```
CREATE TABLE empleados (  
    id NUMBER PRIMARY KEY,  
    nombre VARCHAR2(100)  
);  
  
CREATE SEQUENCE empleado_seq START WITH 1;  
INSERT INTO empleados (id, nombre) VALUES (empleado_seq.NEXTVAL, 'Juan');
```



# 1. ORACLE

## 3) Concatenación y NULL

En MariaDB, la concatenación se realiza con CONCAT (ej: CONCAT('Hola',' mundo');

En Oracle se puede utilizar también CONCAT, pero, además, se puede utilizar el operador ||

```
select concat('1','2') nume1,  
       '3' || '4' nume2 from dual;
```

A-Z NUME1	A-Z NUME2
12	34

Además, si concatenamos con el valor null:

En MariaDB, NULL || 'texto' devuelve 'texto'.

En Oracle, NULL || 'texto' sigue siendo NULL. Para evitarlo, usa NVL() o COALESCE().

En Oracle, la función **NVL()** equivale al IFNULL() de MariaDB.

## 4) Texto

En MariaDB, las cadenas de texto pueden ir entre comillas simples ('Hola') o entre comillas dobles ("Hola")

En ORACLE, las cadenas de texto **SOLO pueden ir entre comillas simples**: 'Hola' ✅, "Hola" ❌

NOTA: Usad **SIEMPRE** comillas simples y así no habrá problema con las consultas.



# 1. ORACLE

## 5) Consultas sin tablas

En MariaDB puedes utilizar una consulta sin tablas.

En Oracle, para este tipo de consultas, necesitas usar la tabla dual.

## 6) Subconsultas en el from

En Oracle, las subconsultas en la cláusula from (que simulan “tablas”) no necesitan un alias forzosamente.

## 7) Función DECODE (en Oracle)

En Oracle, la función DECODE es similar a un CASE WHEN (también existen).

Su sintaxis es **DECODE**(valor\_a\_analizar, valor1, salida1, valor2, salida2... salida por defecto)

```
SELECT Case when nume = 1 then 'uno' else 'dos' end
numero_texto,
decode(nume,1,'uno','dos') numero_texto2 from
(SELECT 1 nume from dual
UNION
SELECT 2 nume from dual);
```

A:Z NUMERO_TEXTO	A:Z NUMERO_TEXTO2
uno	uno
dos	dos



# 1. ORACLE

## 8) Limitación de resultados

En MariaDB se utiliza **limit X offset Y** (siendo la parte del offset opcional).

En Oracle, **offset Y rows fetch first x rows only** (siendo la parte del offset opcional).

También se puede controlar utilizando ROWNUM como filtro en la cláusula where:

```
select * from jugadores  
offset 10 rows fetch first 5 rows only
```

```
select * from jugadores  
where rownum < 5;
```

## 9) Manejo de fechas

En Oracle, para el formateo de fechas utilizamos el **TO\_CHAR()** para pasar fechas a texto con un formato y **TO\_DATE()** para pasar cadenas de texto a fechas siguiendo un formato. Además, se usa SYSDATE en lugar de current\_date() o now()

```
SELECT TO_CHAR(SYSDATE, 'DD/MM/YYYY HH24:MI:SS') fecha_a_texto,  
TO_DATE('2021-12-03 13:02:05', 'YYYY-MM-DD HH24:MI:SS') texto_a_fecha FROM  
dual;
```

A-Z FECHA_A_TEXTO	TEXT0_A_FECHA
21/02/2025 15:27:03	2021-12-03 13:02:05.000



# 1. ORACLE

## 10) Años entre dos fechas

En Oracle no existe Timestampdiff, pero sí existe una función para calcular los meses pasados entre dos fechas (**MONTHS\_BETWEEN**). Bastaría con dividir por 12 después:

```
SELECT trunc(MONTHS_BETWEEN(SYSDATE, to_date('1982-01-15', 'YYYY-MM-DD'))/12) anyos
FROM dual;
```

123 ANYOS
43





## 2. Combinaciones de consultas

En SQL hay ciertas cláusulas que sirven para combinar datos de diferentes consultas.

NOTA: Han de coincidir las columnas de todas las consultas implicadas.

### UNION

Sirve para combinar los resultados de dos consultas. En caso de que hubiese registros duplicados (todas las columnas son iguales) los ignoraría en el resultado final.

```
select * from jugadores
where
habilidad_principal='ataque';
```

id	nombre	tipo	id_equipo	fecha_nacimiento	habilidad_principal
2	Nabil Fekir	futbolista	2	1993-07-18	ataque
8	Gerard Moreno	futbolista	7	1992-04-07	ataque
10	Mikel Oyarzabal	futbolista	9	1997-04-21	ataque
12	Luis Suárez	futbolista	11	1987-01-24	ataque

```
select * from jugadores
where mod(id,6)=0;
```

id	nombre	tipo	id_equipo	fecha_nacimiento	habilidad_principal
6	Jan Oblak	futbolista	5	1993-01-07	portería
12	Luis Suárez	futbolista	11	1987-01-24	ataque
18	Rue	tributo	[NULL]	2005-04-10	sigilo
24	Haymitch Abernath	tributo	[NULL]	1976-03-25	cuchillo

```
select * from jugadores
where
habilidad_principal='ataque'
union
select * from jugadores
where mod(id,6)=0
order by 1;
```

id	nombre	tipo	id_equipo	fecha_nacimiento	habilidad_principal
2	Nabil Fekir	futbolista	2	1993-07-18	ataque
6	Jan Oblak	futbolista	5	1993-01-07	portería
8	Gerard Moreno	futbolista	7	1992-04-07	ataque
10	Mikel Oyarzabal	futbolista	9	1997-04-21	ataque
12	Luis Suárez	futbolista	11	1987-01-24	ataque
18	Rue	tributo	[NULL]	2005-04-10	sigilo
24	Haymitch Abernathy	tributo	[NULL]	1976-03-25	cuchillo



## 2. Combinaciones de consultas

### UNION ALL

Es como UNION, pero admite duplicados:

```
select * from jugadores
where habilidad_principal='ataque'
union all
select * from jugadores
where mod(id,6)=0
order by 1;
```

123 id	A-Z nombre	A-Z tipo	123 id_equipo	🕒 fecha_nacimiento	A-Z habilidad_principal
2	Nabil Fekir	futbolista	2	1993-07-18	ataque
6	Jan Oblak	futbolista	5	1993-01-07	portería
8	Gerard Moreno	futbolista	7	1992-04-07	ataque
10	Mikel Oyarzabal	futbolista	9	1997-04-21	ataque
12	Luis Suárez	futbolista	11	1987-01-24	ataque
12	Luis Suárez	futbolista	11	1987-01-24	ataque
18	Rue	tributo	[NULL]	2005-04-10	sigilo
24	Haymitch Abernathy	tributo	[NULL]	1976-03-25	cuchillo

### EXCEPT

Quita de la primera consulta los resultados coincidentes de la segunda. En algunas BBDD se utiliza exclude en lugar de except (como en PostgreSQL).

```
select * from jugadores
where habilidad_principal='ataque'
except
select * from jugadores
where mod(id,6)=0
order by 1;
```

123 id	A-Z nombre	A-Z tipo	123 id_equipo	🕒 fecha_nacimiento	A-Z habilidad_principal
2	Nabil Fekir	futbolista	2	1993-07-18	ataque
8	Gerard Moreno	futbolista	7	1992-04-07	ataque
10	Mikel Oyarzabal	futbolista	9	1997-04-21	ataque



## 2. Combinaciones de consultas

### INTERSECT

Devuelve la intersección de las consultas, es decir, los registros que están (exactamente igual) en ambas:

```
select * from jugadores
where habilidad_principal='ataque'
intersect
select * from jugadores
where mod(id,6)=0
order by 1;
```

123 id ▼	A-Z nombre ▼	A-Z tipo ▼	123 id_equipo ▼	🕒 fecha_nacimiento ▼	A-Z habilidad_principal ▼
12	Luis Suárez	futbolista	11	1987-01-24	ataque



# 3. Vistas

Una **vista** es una consulta almacenada que funciona como una tabla virtual. Permite acceder a datos de una o más tablas sin almacenar los resultados de manera física en la base de datos. Además, sirven para facilitar la organización y el acceso a los datos (un usuario podría no tener acceso a las tablas que conforman una vista, pero sí a la vista en sí, pudiendo visualizar solo los datos que se le permiten)

## Creación de una vista:

```
CREATE VIEW nombre_vista AS
SELECT columna1, columna2, ...
FROM tabla_base
WHERE condicion;
```

## Ejemplos:

```
create view futbolistas as
select left(j.nombre, instr(j.nombre, ' ')-1) nombre,
right(j.nombre, char_length(j.nombre) -
instr(j.nombre, ' ')) apellido,
e.nombre equipo
from jugadores j
join equipos e on (j.id_equipo = e.id);
```

```
create or replace view futbolistas as
select substr(j.nombre, instr(j.nombre, ' ')) nombre,
substr(j.nombre, instr(j.nombre, ' ')+1,
length(j.nombre) - instr(j.nombre, ' ')) apellido,
e.nombre equipo
from c##futjdh.jugadores j
join c##futjdh.equipos e on (j.id_equipo = e.id)
where e.nombre != 'Celta de Vigo';
```



# 3. Vistas

Para utilizar una vista utilizamos la sintaxis como si se tratase de una tabla normal y corriente:

```
select * from futbolistas;
```

**Modificación de una vista:**

```
create or replace view futbolistas as
select left(j.nombre, instr(j.nombre, ' ')-1) nombre,
right(j.nombre, char_length(j.nombre) - instr(j.nombre, ' ')) apellido,
e.nombre equipo
from jugadores j
join equipos e on (j.id_equipo = e.id)
where e.nombre != 'Celta de Vigo';
```

**Eliminación de una lista:**

```
drop view futbolistas;
```



## 4. SQL Avanzado: STRING\_AGG

Ya hemos visto ciertas operaciones de agregación como pueden ser SUM, COUNT o AVG.

Pero, ¿qué ocurre si al agrupar registros queremos agrupar campos de formato cadena? Para eso sirve la función **STRING\_AGG** (en PostgreSQL y en Oracle a partir de la versión 23c).

En MariaDB no existe STRING\_AGG, pero una operación similar es **GROUP\_CONCAT**:

```
select tipo, group_concat(nombre SEPARATOR ', ') jugadores
from futbol_jdh_toki.jugadores j
group by tipo;
```

	AZ tipo	AZ jugadores
1	futbolista	Sergio Ramos, Luis Suárez, Iago Aspas, Mikel Oyarzabal, Iñaki Williams, Gerard Moreno, Carlos Soler, Jan Oblak, Karim Benzema, Alexia Putellas, Lionel Messi, Nabil Fekir
2	tributo	Clove, Glimmer, Foxface, Marvel, Thresh, Rue, Johanna Mason, Finnick Odair, Cato, Peeta Mellark, Katniss Everdeen, Haymitch Abernathy

En Oracle tendríamos (hasta la 23c) **LISTAGG** para hacer lo mismo:

```
select tipo, listagg(nombre, ', ') jugadores
from c##futjdh.jugadores
group by tipo
```



# 5. SQL Avanzado: RANK

La función **RANK()** (y **DENSE\_RANK()**) sirve para ordenar resultados dentro de grupos (particiones). La sintaxis es:

**RANK() OVER** (**PARTITION BY** columnas\_que\_determinan\_la\_particion  
**ORDER BY** columnas\_para\_la\_ordenacion) as ranking

```
select j.*, rank() over (partition by tipo order by fecha_nacimiento) ranking
from jugadores j;
```

	123 ID	A:2 NOMBRE	A:2 TIPO	123 ID_EQUIPO	FECHA_NACIMIENTO	A:2 HABILIDAD_PRINCIPAL	123 RANKING
1	1	Sergio Ramos	futbolista	1	1986-03-30 00:00:00.000	defensa	1
2	12	Luis Suárez	futbolista	11	1987-01-24 00:00:00.000	ataque	2
3	3	Lionel Messi	futbolista	4	1987-06-24 00:00:00.000	regate	3
4	11	Iago Aspas	futbolista	10	1987-08-01 00:00:00.000	remate	4
5	5	Karim Benzema	futbolista	3	1987-12-19 00:00:00.000	remate	5
6	8	Gerard Moreno	futbolista	7	1992-04-07 00:00:00.000	ataque	6
7	6	Jan Oblak	futbolista	5	1993-01-07 00:00:00.000	portería	7
8	2	Nabil Fekir	futbolista	2	1993-07-18 00:00:00.000	ataque	8
9	4	Alexia Putellas	futbolista	4	1994-02-04 00:00:00.000	mediocampo	9
10	9	Iñaki Williams	futbolista	8	1994-06-15 00:00:00.000	velocidad	10
11	7	Carlos Soler	futbolista	6	1997-01-02 00:00:00.000	mediocampo	11
12	10	Mikel Oyarzabal	futbolista	9	1997-04-21 00:00:00.000	ataque	12
13	24	Haymitch Abernathy	tributo	[NULL]	1976-03-25 00:00:00.000	cuchillo	1
14	17	Johanna Mason	tributo	[NULL]	1997-09-30 00:00:00.000	hacha	2
15	16	Finnick Odair	tributo	[NULL]	1998-07-15 00:00:00.000	tridente	3
16	19	Thresh	tributo	[NULL]	1998-12-01 00:00:00.000	fuerza	4
17	23	Marvel	tributo	[NULL]	1999-05-05 00:00:00.000	lanza	5
18	20	Clove	tributo	[NULL]	1999-06-18 00:00:00.000	cuchillos	6
19	15	Cato	tributo	[NULL]	1999-11-22 00:00:00.000	combate cuerpo a cuerpo	7
20	21	Glimmer	tributo	[NULL]	2000-02-14 00:00:00.000	arco	8
21	14	Peeta Mellark	tributo	[NULL]	2000-03-12 00:00:00.000	estrategia	9
22	13	Katniss Everdeen	tributo	[NULL]	2000-05-08 00:00:00.000	arco	10
23	22	Foxface	tributo	[NULL]	2001-08-20 00:00:00.000	astucia	11
24	18	Rue	tributo	[NULL]	2005-04-10 00:00:00.000	sigilo	12



# 5. SQL Avanzado: RANK

La diferencia entre rank() y dense\_rank() es que el segundo no se salta números en caso de empate. Ejemplo:

```
select j.*, rank() over (partition by tipo order by habilidad_principal) ranking,  
dense_rank() over (partition by tipo order by habilidad_principal) dense_ranking  
from "C##FUTJDH".jugadores j;
```

123 ID	AZ NOMBRE	AZ TIPO	123 ID_EQUIPO	FECHA_NACIMIENTO	AZ HABILIDAD_PRINCIPAL	123 RANKING	123 DENSE_RANKING
2	Nabil Fekir	futbolista	2	1993-07-18 00:00:00.000	ataque	1	1
12	Luis Suárez	futbolista	11	1987-01-24 00:00:00.000	ataque	1	1
10	Mikel Oyarzabal	futbolista	9	1997-04-21 00:00:00.000	ataque	1	1
8	Gerard Moreno	futbolista	7	1992-04-07 00:00:00.000	ataque	1	1
1	Sergio Ramos	futbolista	1	1986-03-30 00:00:00.000	defensa	5	2
7	Carlos Soler	futbolista	6	1997-01-02 00:00:00.000	mediocampo	6	3
4	Alexia Putellas	futbolista	4	1994-02-04 00:00:00.000	mediocampo	6	3
6	Jan Oblak	futbolista	5	1993-01-07 00:00:00.000	porteria	8	4
3	Lionel Messi	futbolista	4	1987-06-24 00:00:00.000	regate	9	5
5	Karim Benzema	futbolista	3	1987-12-19 00:00:00.000	remate	10	6
11	Iago Aspas	futbolista	10	1987-08-01 00:00:00.000	remate	10	6
9	Iñaki Williams	futbolista	8	1994-06-15 00:00:00.000	velocidad	12	7
13	Katniss Everdeen	tributo	[NULL]	2000-05-08 00:00:00.000	arco	1	1
21	Glimmer	tributo	[NULL]	2000-02-14 00:00:00.000	arco	1	1
22	Foxface	tributo	[NULL]	2001-08-20 00:00:00.000	astucia	3	2
15	Cato	tributo	[NULL]	1999-11-22 00:00:00.000	combate cuerpo a cuerpo	4	3
24	Haymitch Abernathy	tributo	[NULL]	1976-03-25 00:00:00.000	cuchillo	5	4
20	Clove	tributo	[NULL]	1999-06-18 00:00:00.000	cuchillos	6	5
14	Peeta Mellark	tributo	[NULL]	2000-03-12 00:00:00.000	estrategia	7	6
19	Thresh	tributo	[NULL]	1998-12-01 00:00:00.000	fuerza	8	7
17	Johanna Mason	tributo	[NULL]	1997-09-30 00:00:00.000	hacha	9	8
23	Marvel	tributo	[NULL]	1999-05-05 00:00:00.000	lanza	10	9
18	Rue	tributo	[NULL]	2005-04-10 00:00:00.000	sigilo	11	10
16	Finnick Odair	tributo	[NULL]	1998-07-15 00:00:00.000	tridente	12	11





## 6. SQL Avanzado: LAG y LEAD

Las funciones **LAG** y **LEAD** se utilizan para acceder a filas anteriores o siguientes (respectivamente) dentro de una misma consulta sin necesidad de hacer subconsultas o JOIN adicionales.

NOTA: también pueden usar la cláusula “partition by”.

```
select m.*,  
lag(hora_muerte) over(order by hora_muerte) hora_muerte_anterior,  
lead(hora_muerte) over(order by hora_muerte) hora_muerte_siguiente  
from "C##FUTJDH".muertes m;
```

123 ID	123 ID_JUGADOR	123 ID_ASESINO	AZ CAUSA	AZ HORA_MUERTE	AZ HORA_MUERTE_ANTERIOR	AZ HORA_MUERTE_SIGUIENTE
1	3	1	combate	00:30:12	[NULL]	01:15:40
2	17	7	asesinato	01:15:40	00:30:12	02:10:50
3	6	21	emboscada	02:10:50	01:15:40	04:05:33
4	8	10	combate	04:05:33	02:10:50	06:25:48
5	16	18	emboscada	06:25:48	04:05:33	07:10:00
6	4	2	asesinato	07:10:00	06:25:48	08:40:12
7	10	19	combate	08:40:12	07:10:00	11:20:30
8	1	15	venganza	11:20:30	08:40:12	13:10:45
9	18	5	combate	13:10:45	11:20:30	14:25:30
10	9	20	emboscada	14:25:30	13:10:45	15:35:10
11	7	24	asesinato	15:35:10	14:25:30	17:45:12
12	5	13	combate	17:45:12	15:35:10	19:20:00
13	20	11	asesinato	19:20:00	17:45:12	20:35:40
14	11	15	combate	20:35:40	19:20:00	21:10:30
15	15	13	asesinato	21:10:30	20:35:40	23:10:50
16	22	23	asesinato	23:10:50	21:10:30	23:20:50
17	2	23	trampa	23:20:50	23:10:50	23:55:00
18	23	[NULL]	accidente	23:55:00	23:20:50	[NULL]



# 7. SQL Avanzado: WITH

La cláusula **WITH** en SQL se usa para definir CTE (Common Table Expressions), que permiten escribir subconsultas reutilizables y mejorar la legibilidad. Se pueden encadenar las definiciones.

```
WITH
tributos AS (SELECT * from "C##FUTJDH".jugadores j
              WHERE j.tipo='tributo'),
habilidades_tributos AS (SELECT habilidad_principal hab,
                               substr(habilidad_principal,1,3) abreviatura from tributos
                          GROUP BY habilidad_principal
                          ORDER BY 1)
SELECT * FROM habilidades_tributos;
```

A-Z HAB	A-Z ABREVIATURA
arco	arc
astucia	ast
combate cuerpo a cuerpo	com
cuchillo	cuc
cuchillos	cuc
estrategia	est
fuerza	fue
hacha	hac
lanza	lan
sigilo	sig
tridente	tri



# 8. SQL Avanzado: WITH Recursivo

La cláusula **WITH RECURSIVE** (en ORACLE solo **WITH**) se utiliza en SQL para llamadas recursivas:

Ejemplo: partiendo de una tabla empleados con el siguiente contenido:

ID	NOMBRE	ID_JEFE
1	1 María (CEO)	(null)
2	2 Luis	1
3	3 Ana	2
4	4 Lucía	3
5	5 Diego	3
6	6 Carlos	1
7	7 Elena	1
8	8 Pedro	7

ID	NOMBRE	ID_JEFE	NIVEL
1	1 María (CEO)	(null)	1
2	2 Luis	1	2
3	3 Ana	2	3
4	4 Lucía	3	4
5	5 Diego	3	4
6	6 Carlos	1	2
7	7 Elena	1	2
8	8 Pedro	7	3

```
WITH EmpleadoJerarquia (id, nombre, id_jefe, nivel) AS (  
  -- Caso base: Selecciona el CEO (empleado sin jefe)  
  SELECT id, nombre, id_jefe, 1  
  FROM empleados  
  WHERE id_jefe IS NULL  
  
  UNION ALL  
  
  -- Caso recursivo: Encuentra los empleados bajo el jefe  
  SELECT e.id, e.nombre, e.id_jefe, ej.nivel + 1  
  FROM empleados e  
  JOIN EmpleadoJerarquia ej ON e.id_jefe = ej.id  
)  
  
SELECT * FROM EmpleadoJerarquia  
ORDER BY id;
```



# 9. SQL Avanzado: PIVOT

**PIVOT** es una cláusula exclusiva de Oracle y SQL Server que sirve para transponer la información de una tabla (pasar las filas a columnas)

```
select al.nombre, asi.nombre asignatura, nota from
alumnos asignaturas alas
join asignaturas asi on (alas.id_asignatura = asi.id)
join alumnos al on (al.id = alas.id_alumno);
```

NOMBRE	ASIGNATURA	NOTA
1 Carlos	Matemáticas	8,5
2 Carlos	Biología	7
3 Carlos	Física	4,5
4 Lucía	Matemáticas	7,3
5 Lucía	Biología	6,3
6 Lucía	Física	10
7 Marina	Filosofía	10
8 Marina	Historia	10
9 Javier	Historia	8
10 Javier	Filosofía	(null)
11 Sofía	Historia	7,9
12 Sofía	Filosofía	(null)
13 Daniel	Historia	9,9
14 Daniel	Filosofía	8,1
15 Elena	Matemáticas	5,5
16 Elena	Biología	6
17 Elena	Física	6,5
18 Pablo	Matemáticas	7

```
with
tablaAux as(select al.nombre, asi.nombre asignatura, nota from
alumnos asignaturas alas
join asignaturas asi on (alas.id_asignatura = asi.id)
join alumnos al on (al.id = alas.id_alumno))
SELECT *
FROM tablaAux
PIVOT (
-- Función de agregación sobre los valores
MAX(nota)
FOR asignatura IN ('Matemáticas' AS Matemáticas, 'Biología' AS Biología,
'Física' AS Física, 'Historia' AS Historia, 'Filosofía' AS Filosofía)
);
```

NOMBRE	MATEMÁTICAS	BIOLOGÍA	FÍSICA	HISTORIA	FILOSOFÍA
1 Carlos	8,5	7	4,5	(null)	(null)
2 Lucía	7,3	6,3	10	(null)	(null)
3 Marina	(null)	(null)	(null)	10	10
4 Javier	(null)	(null)	(null)	8	(null)
5 Sofía	(null)	(null)	(null)	7,9	(null)
6 Daniel	(null)	(null)	(null)	9,9	8,1
7 Elena	5,5	6	6,5	(null)	(null)
8 Pablo	7	(null)	8,3	(null)	(null)
9 Clara	(null)	(null)	3,4	(null)	(null)
10 Adrián	10	9,5	10	(null)	(null)



## 9. SQL Avanzado: PIVOT

La sintaxis de **PIVOT** es la siguiente:

```
SELECT * -- o lista de campos a mostrar y transponer
FROM tabla
PIVOT (
    -- Función de agregación sobre los valores
    MAX(valor) -- Valor a poner en cada columna según la distribución siguiente
    FOR columna_a_transponer IN ('Valor1' AS Columna1, 'Valor2' AS Columna2, 'Valor3' AS Columna3 --, etc.
)
);
```

Con esto, mostramos todos los valores de la tabla indicados en la consulta, pero los campos “valor” y la columna que hay que transponer, aparecerán en el modo columnar visto en la diapositiva anterior

