

# Unidad Didáctica 2: Uso de Bases de Datos Relacionales

## Parte 1: El Lenguaje SQL: Consultas y Actualización (DML)

### U.D. 2.1

## UD2.1 El lenguaje SQL: consultas y actualización (DML)

---

### Objetivos:

- Presentar la sintaxis del lenguaje SQL (sólo el Lenguaje de Manipulación).
- Ver algunos ejemplos sencillos para clarificar la semántica del SQL.
- Presentar la base de datos CICLISMO.
- Realizar de menor a mayor complejidad consultas SQL sobre la base de datos CICLISMO.

## UD 2.1 El lenguaje SQL: consultas y actualización (DML)

---

### 1.- Introducción a SQL

- 2.- SELECT. Consultas sencillas sobre una tabla
- 3.- SELECT. Consultas simples sobre varias tablas
- 4.- SELECT. Consultas complejas: Subconsultas
- 5.- SELECT. Con. complejas: Agrupación (GROUP BY)
- 6.- SELECT. Operadores conjuntistas
- 7.- SELECT. Operador JOIN
- 8.- Lenguaje de Manipulación de Datos (DML)
  - 8.1.- Instrucción INSERT
  - 8.2.- Instrucción DELETE
  - 8.3.- Instrucción UPDATE
- 9.- Instrucciones de control de transacciones

# 1. Introducción a SQL

---

## Lenguaje SQL:

- El lenguaje SQL (**S**tructured **Q**uery **L**anguage - lenguaje de consulta estructurado) es un lenguaje de acceso a bases de datos relacionales.
- **Permite**, entre otras cosas
  - crear y modificar esquemas de bases de datos y
  - especificar las operaciones sobre bases de datos.
- Aúna características del álgebra y el cálculo relacional.

# Sublenguajes de SQL

---

- **Lenguaje de Definición de Datos** (LDD –DDL en inglés) para crear y modificar esquemas de BD.
- **Lenguaje de Manipulación de Datos** (LMD -DML en inglés) para consulta y actualización de las BD.

## **Consta de las instrucciones:**

- SELECT (consulta)
- INSERT (inserción de tuplas)
- DELETE (borrado de tuplas)
- UPDATE (modificación de tuplas)
- Otras instrucciones (no se estudiarán)
- **Lenguaje de Control** para cambiar dinámicamente propiedades de la BD. (Sólo se citará alguna instrucción)

# Lenguaje de Manipulación de Datos (LMD)

---

Se presentan las instrucciones que se pueden ejecutar desde un intérprete de SQL, lo que se denomina *SQL interactivo*.

SQL es un lenguaje muy expresivo y, en general, permite muchas formas de expresar las misma órdenes.

Las instrucciones que componen el DML son las siguientes:

- **SELECT**: permite la declaración de consultas para la recuperación de información de una o más tablas de una base de datos.
- **INSERT**: realiza la inserción de una o varias filas sobre una tabla.
- **DELETE**: permite efectuar el borrado de una o varias filas de una tabla.
- **UPDATE**: realiza una modificación de los valores de una o más columnas de una o varias filas de una tabla.

**EQUIPO**(**nomeq**: d\_eq, **director**: d\_dir)

Clave Primaria: {nomeq}

**CICLISTA**(**dorsal**: d\_dor, **nombre**: d\_nom, **edad**: d\_edad, **nomeq**: d\_eq))

Clave Primaria: {dorsal}

CAj: {nomeq} hace referencia a EQUIPO

VNN: {nomeq}

**ETAPA**(**netapa**:d\_n°, **km**:d\_km, **salida**:d\_sal, **llegada**:d\_lleg, **dorsal**:d\_dor)

Clave Primaria: {netapa}

CAj: {dorsal} hace referencia a CICLISTA

**PUERTO**(**nompuerto**:d\_nom,**altura**:d\_alt,**categoria**:d\_cat,  
**netapa**:d\_n°,**dorsal**: d\_dor)

Clave Primaria: {nompuerto}

CAj: {netapa} hace referencia a ETAPA

CAj: {dorsal} hace referencia a CICLISTA

VNN: {netapa}

**MAILLOT**(**codigo**: d\_código, **tipo**: d\_tipo, **premio**: d\_pre, **color**: d\_col)

Clave Primaria: {codigo}

**LLEVAR**(**dorsal**: entero, **netapa**: d\_n°, **codigo**: d\_código)

Clave Primaria: {netapa, codigo}

CAj: {netapa} hace referencia a ETAPA

CAj: {dorsal} hace referencia a CICLISTA

CAj: {codigo} hace referencia a MAILLOT

VNN: {dorsal}

## UD 2.1 El lenguaje SQL: consultas y actualización (DML)

---

1.- Introducción a SQL

2.- SELECT. Consultas sencillas sobre una tabla

3.- SELECT. Consultas simples sobre varias tablas

4.- SELECT. Consultas complejas: Subconsultas

5.- SELECT. Con. complejas: Agrupación (GROUP BY)

6.- SELECT. Operadores conjuntistas

7.- SELECT. Operador JOIN

8.- Lenguaje de Manipulación de Datos (DML)

8.1.- Instrucción INSERT

8.2.- Instrucción DELETE

8.3.- Instrucción UPDATE

9.- Instrucciones de control de transacciones



## 2. SELECT.

### Consultas sencillas sobre una tabla

---

**EJEMPLO:** Obtener el nombre y la edad de todos los ciclistas.

```
SELECT nombre, edad  
FROM Ciclista;
```

---

**EJEMPLO: Obtener edades de las que hay ciclistas.**

```
SELECT DISTINCT edad  
FROM Ciclista;
```

**EJEMPLO: Obtener toda la información de los equipos.**

```
SELECT *  
FROM Equipo;
```

---

EJEMPLO: Obtener el nombre, la altura y la categoría de todos los puertos ordenados por altura y categoría.

```
SELECT nompuerto, altura, categoria  
FROM Puerto  
ORDER BY altura, categoria ;
```

EJEMPLO: Obtener el nombre y la altura de todos los puertos de 1ª categoría **ordenados de menor a mayor altura.**

---

1. ¿En qué tablas se encuentra la información?
2. ¿Qué condición deben cumplir las filas resultantes?
3. ¿Que información queremos visualizar?
4. ¿Queremos ordenar el resultado por alguna columna?

```
SELECT nompuerto, altura  
FROM Puerto  
WHERE categoria = '1'  
ORDER BY altura ;
```

# Sintaxis

```
SELECT [ALL | DISTINCT] {expresion1, expresion2, ... expresionn} | *  
FROM Tabla  
[WHERE expresión_condicional]  
[ORDER BY {columna1, columna2, ... columnam}]
```

- **ALL** : Permite la aparición de filas idénticas (valor por defecto).
- **DISTINCT**: No permite la aparición de filas idénticas.
- La *expresión\_condicional* está formada por un conjunto de predicados combinados con las conectivas lógicas AND, OR y NOT. Los predicados utilizados permiten comparar columnas:
  - predicados de **comparación**: =, <>, >, <, >=, <=.
  - predicado **LIKE**: permite comparar una tira de caracteres con un patrón.
  - predicado **BETWEEN**: permite comprobar si un escalar está en un rango.
  - predicado **IN**: permite comprobar si el valor está dentro de un conjunto.
  - predicado **IS NULL**: permite comprobar si el valor es nulo.

# Alias y orden decreciente

---

Nombre y edad de los ciclistas del Bora ordenados de mayor a menor edad. La cabecera de la columna de nombres debe ser “Bora”

```
SELECT nombre AS Bora, edad  
FROM Ciclista  
WHERE nomeq = 'Bora'  
ORDER BY edad DESC;
```

---

Uso de operadores aritméticos: + (suma), – (diferencia), \* (producto), / (división), etc.

EJEMPLO: Obtener de los maillots el tipo y el premio en euros (supongamos que está en dólares) ( $1\$ = 0.8\text{€}$ ) de aquellos maillots cuyo premio supere los 100 euros.

```
SELECT tipo, premio / 0.8  
FROM Maillot  
WHERE premio / 0.8 > 100;
```

EJEMPLO: Obtener el número de las etapas donde el nombre de la ciudad de llegada tenga por segunda letra una “O” o donde el nombre de la ciudad de salida lleve dos o más ‘A’s

```
SELECT netapa
```

```
FROM Etapa
```

```
WHERE llegada LIKE ‘_O%’ OR salida LIKE ‘%A%A%’
```



---

**Uso de LIKE en casos especiales (si la cadena a buscar contiene un carácter 'comodín')**

**EJEMPLO:** Obtener el nombre y la edad de los ciclistas que pertenezcan a equipos cuyo nombre contenga la cadena "100%".

```
SELECT nombre, edad  
FROM Ciclista  
WHERE nomeq LIKE '%100\%%%' ESCAPE '\'
```

---

EJEMPLO: Obtener el nombre de los ciclistas cuya edad está entre 20 y 30 años.


```
SELECT nombre FROM Ciclista  
WHERE edad BETWEEN 20 AND 30;
```

El predicado **BETWEEN** es equivalente a una condición con comparaciones de la siguiente forma:

$$exp \text{ BETWEEN } exp_1 \text{ and } exp_2 \equiv (exp \geq exp_1) \text{ and } (exp \leq exp_2)$$

Ejemplo de consulta **incorrecta** (error de sintaxis)

```
SELECT nomeq  
FROM Equipo  
WHERE director = null
```



La consulta **correcta** sería

```
SELECT nomeq  
FROM Equipo  
WHERE director IS NULL
```

## Comparación con el valor nulo

---

Las comparaciones entre cualquier valor y NULL resultan en *indefinido*.

Ejemplo:

```
SELECT *  
FROM T  
WHERE atrib1 > atrib2
```

Si en una fila se diera el caso que atrib<sub>1</sub> = 50 y atrib<sub>2</sub> fuera nulo, el resultado de la comparación sería indefinido y por tanto dicha fila no se incluiría en la selección.

# Consultas con Funciones Agregadas

**{ AVG | MAX | MIN | SUM | COUNT } ( [ ALL | DISTINCT ] *expresión\_escalar* ) | COUNT(\*)**

- Las funciones agregadas no se pueden anidar.
- Para las funciones SUM y AVG los argumentos deben ser numéricos.
- Utilizando DISTINCT los valores repetidos son eliminados antes de realizar el cálculo.
- La función especial COUNT(\*), en la que no está permitido incluir DISTINCT ni ALL, da como resultado el número de filas de la selección.
- Los cálculos se realizan después de la selección y aplicar las condiciones.
- Los valores nulos son eliminados antes de realizar los cálculos (incl. COUNT).
- Si el número de filas de la selección es 0, la función COUNT devuelve el valor 0 y las otras funciones el valor nulo.

# Funciones agregadas en consultas no agrupadas

## EJEMPLO:

```
SELECT 'Núm. de ciclistas =', COUNT(*), 'Media Edad =', AVG(edad)
FROM Ciclista
WHERE nomeq = 'Banesto';
```

En consultas no agrupadas, la selección sólo podrá incluir referencias a funciones agregadas o literales ya que las funciones van a devolver un único valor.

## EJEMPLO INCORRECTO:

```
SELECT nombre, AVG(edad)
FROM Ciclista
WHERE nomeq = 'ONCE';
```

## UD 2.1 El lenguaje SQL: consultas y actualización (DML)

---

- 1.- Introducción a SQL
- 2.- SELECT. Consultas sencillas sobre una tabla
- 3.- SELECT. Consultas simples sobre varias tablas
- 4.- SELECT. Consultas complejas: Subconsultas
- 5.- SELECT. Con. complejas: Agrupación (GROUP BY)
- 6.- SELECT. Operadores conjuntistas
- 7.- SELECT. Operador JOIN
- 8.- Lenguaje de Manipulación de Datos (DML)
  - 8.1.- Instrucción INSERT
  - 8.2.- Instrucción DELETE
  - 8.3.- Instrucción UPDATE
- 9.- Instrucciones de control de transacciones

### 3. Instrucción SELECT.

#### Consultas simples sobre varias tablas

---

Si la información que se desea obtener está almacenada en varias tablas, la consulta debe incluir dichas tablas en la cláusula FROM.



EJEMPLO: Obtener pares de números de etapas y nombres de puertos ganados por el mismo ciclista.

---

1. ¿En qué tablas se encuentra la información?

FROM Etapa, Puerto

2. ¿Qué condición deben cumplir las filas resultantes?

WHERE etapa.dorsal = puerto.dorsal;

3. ¿Qué información queremos visualizar?

SELECT etapa.netapa, nompuerto

En esta expresión es obligatorio que la referencia a la columna *dorsal* de *Etapa* y *Puerto* sea calificada con el nombre de la tabla, si no es **ambigua**.

SELECT **etapa**.netapa, nompuerto

FROM Etapa, Puerto

WHERE **etapa**.dorsal = **puerto**.dorsal ;

## Ejemplo: Algebra vs SELECT

R

a1	a2	n
A	20	1
B	30	2
C	20	3

S

b1	n
X	3
Y	4
X	1

$R \otimes_n S$

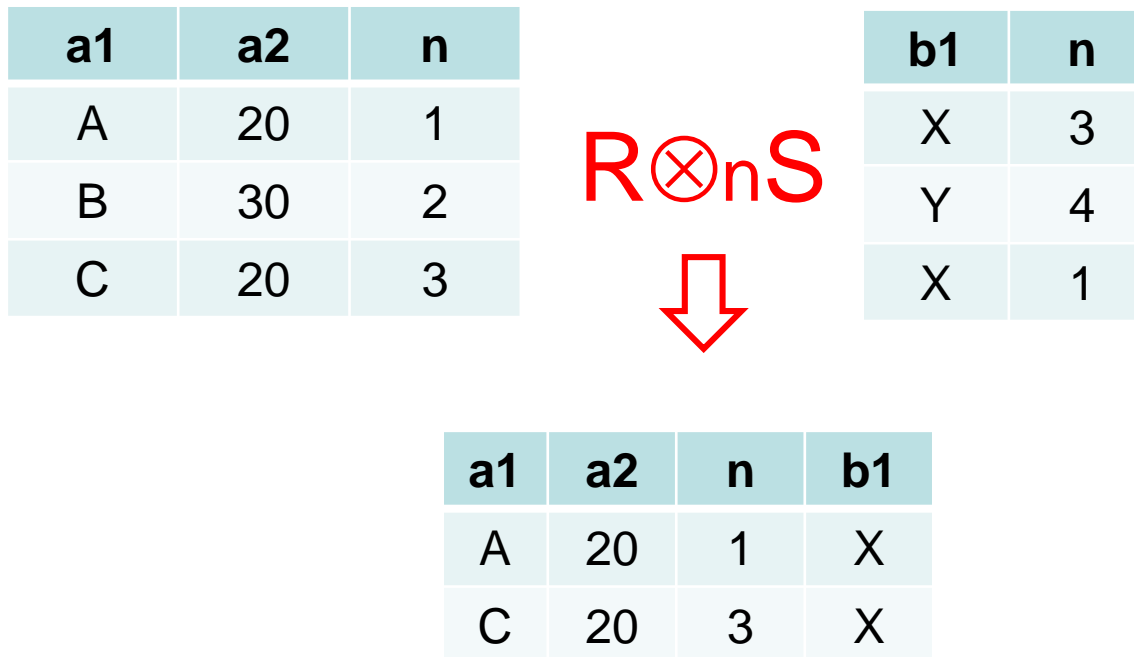
$R \otimes_n S[a2, b1]$

**SELECT \***  
**FROM R, S**  
**WHERE R.n = S.n;**

**SELECT a2, b1**  
**FROM R, S**  
**WHERE R.n = S.n;**

## Ejemplo: Algebra vs SELECT

Combina la información contenida en dos relaciones uniendo las tuplas que tienen en mismo valor en los atributos con el mismo nombre



## Ejemplo: Algebra vs SELECT

**SELECT \***  
**FROM R, S**  
**WHERE R.n = S.n;**

R

a1	a2	n
A	20	1
B	30	2
C	20	3

S

b1	n
X	3
Y	4
X	1



R.a1	R.a2	R.n	S.b1	S.n
A	20	1	X	3
A	20	1	Y	4
A	20	1	X	1
B	30	2	X	3
B	30	2	Y	4
B	30	2	X	1
C	20	3	X	3
C	20	3	Y	4
C	20	3	X	1



R.a1	R.a2	R.n	S.b1	S.n
A	20	1	X	1
C	20	3	X	3

## Ejemplo: Algebra vs SELECT

R

a1	a2	n
A	20	1
B	30	2
C	20	3

S

b1	n
X	3
Y	4
X	1

$R \otimes_n S$

a1	a2	n	b1
A	20	1	X
C	20	3	X

**SELECT \***  
**FROM R, S**  
**WHERE R.n = S.n;**

R.a1	R.a2	R.n	S.b1	S.n
A	20	1	X	1
C	20	3	X	3

## Ejemplo: Algebra vs SELECT

R

a1	a2	n
A	20	1
B	30	2
C	20	3

S

b1	n
X	3
Y	4
X	1

$R \otimes_n S[a2, b1]$

a2	b1
20	X

(En el álgebra relacional no hay tuplas repetidas en una relación)

**SELECT a2, b1  
FROM R, S  
WHERE R.n = S.n;**

R.a2	S.b1
20	X
20	X

## Uso de claves ajenas en consultas con varias tablas

---

La consulta de varias tablas corresponde al **producto cartesiano**.

Si no se eligen bien las condiciones, el número de filas resultantes puede ser muy grande.



**¡Es importante recordarlo!**

Lo más frecuente es una igualdad entre la clave ajena y la clave primaria de la tabla a la que se hace referencia (aunque no siempre es así).

---

EJEMPLO: Obtener los nombres de los ciclistas pertenecientes al equipo dirigido por 'Alvaro Pino'.

```
SELECT C.nombre  
FROM Ciclista C, Equipo E  
WHERE C.nomeq = E.nomeq AND E.director = 'Alvaro Pino';
```



¡Es muy importante recordar la selección de las filas que interesan!

Las variables de recorrido permiten dar otro nombre a una tabla.

**Sintaxis:**      **FROM tabla [AS] *variable\_recorrido***



---

EJEMPLO: Obtener pares nombre de ciclista y número de etapa, de tal forma que dicho ciclista haya ganado dicha etapa. Además la etapa debe superar los 150 km. de recorrido.

```
SELECT C.nombre, E.netapa  
FROM Ciclista C, Etapa E  
WHERE C.dorsal = E.dorsal AND E.km > 150;
```

## Obtención de filas repetidas.

Al combinar varias tablas, una misma fila de una tabla R puede aparecer relacionada con varias filas de otra tabla S. Si la consulta pide solo información de R, se pueden obtener **filas repetidas**, que en la mayoría de los casos se deben eliminar.

EJEMPLO: Obtener número y longitud de las etapas que tienen puertos de montaña.

```
SELECT DISTINCT E.netapa, km  
FROM Etapa E, Puerto P  
WHERE E.netapa = P.netapa;
```

- Cuando se va a trabajar con una tabla para hacer consulta entre diferentes tuplas de ella, entonces se utilizan necesariamente las variables de recorrido.

**[tabla | *variable\_recorrido*].columna**



Es una instancia de la tabla. Es virtual

- Permiten dar nombres alternativos a la *misma* tabla dentro de una consulta. La manera de declarar una variable de recorrido es:

**from tabla [as] *variable\_recorrido***

EJEMPLO: Obtener el nombre de los ciclistas compañeros de equipo de 'Egan Bernal' que sean más jóvenes que él.

---

1. ¿En qué tablas se encuentra la información?

Ciclista

Como se requiere comparar con tuplas de la misma tabla, se necesita tener varias imágenes de ella

Ciclista C1, Ciclista C2

2. ¿Qué condición deben cumplir las filas resultantes?

C2.nombre='Egan Bernal' AND

C1.nomeq = C2.nomeq AND

C1.edad < C2.edad

3. ¿Qué información queremos visualizar?

C1.nombre

EJEMPLO: Obtener el nombre de los ciclistas compañeros de equipo de 'Egan Bernal' que sean más jóvenes que él.

---

```
SELECT C1.nombre  
FROM Ciclista C1, Ciclista C2  
WHERE    C2.nombre='Egan Bernal' AND  
         C1.nomeq = C2.nomeq AND  
         C1.edad < C2.edad;
```

## UD 2.1 El lenguaje SQL: consultas y actualización (DML)

---

- 1.- Introducción a SQL
- 2.- SELECT. Consultas sencillas sobre una tabla
- 3.- SELECT. Consultas simples sobre varias tablas
- 4.- SELECT. Consultas complejas: Subconsultas
- 5.- SELECT. Con. complejas: Agrupación (GROUP BY)
- 6.- SELECT. Operadores conjuntistas
- 7.- SELECT. Operador JOIN
- 8.- Lenguaje de Manipulación de Datos (DML)
  - 8.1.- Instrucción INSERT
  - 8.2.- Instrucción DELETE
  - 8.3.- Instrucción UPDATE
- 9.- Instrucciones de control de transacciones

## 4. Instrucción SELECT.

### Consultas complejas: Subconsultas

---

¿Qué es una subconsulta?

Una subconsulta es una consulta encerrada entre paréntesis, que se incluye dentro de otra consulta.

Cuando

información buscada



una tabla

condición de búsqueda



otras tablas

EN ALGUNOS CASOS se pueden utilizar las

subconsultas



## EJEMPLO: Obtener número y longitud de las etapas que tienen puertos de montaña.

```
SELECT DISTINCT E.netapa, km  
FROM Etapa E, Puerto P  
WHERE E.netapa = P.netapa;
```

### Usando subconsultas:

```
SELECT netapa, km  
FROM Etapa  
WHERE netapa IN (SELECT netapa  
FROM Puerto)
```

Consulta principal

Subconsulta que  
devuelve  
números de  
etapas

EJEMPLO: Obtener los nombres de los ciclistas pertenecientes al equipo dirigido por 'Alvaro Pino'.

## Usando igualdades:

```
SELECT C.nombre  
FROM Ciclista C, Equipo E  
WHERE C.nomeq = E.nomeq AND E.director = 'Alvaro Pino';
```

## Usando subconsultas:

```
SELECT C.nombre  
FROM Ciclista C  
WHERE C.nomeq = ( SELECT E.nomeq FROM Equipo E  
                  WHERE E.director = 'Alvaro Pino' );
```

*La información que se pide (nombre del ciclista) no está en la tabla de la subconsulta (Equipo)*

*Nombre del equipo dirigido por Álvaro Pino*

*Se puede usar porque la subconsulta retorna SEGURO un único valor.*

# Predicados que aceptan subconsultas

---

Las subconsultas pueden aparecer en las condiciones de búsqueda como argumentos de los predicados siguientes:

- predicados de **comparación** (=, <>, >, <, >=, <=).
- **IN**: comprueba que un valor pertenece a una colección dada mediante una subconsulta.
- **EXISTS**: equivalente al cuantificador existencial, comprueba si una subconsulta devuelve alguna fila.

## Predicados de comparación (=, <>, >, <, >=, <=)

---

### SINTAXIS:

*expresión **predicado de comparación** expresión*

Las subconsultas pueden ser argumentos de un predicado de comparación siempre que

- devuelvan una **única fila**, y
- la columna resultante de la subconsulta **coincida en tipo** con el otro lado del predicado de comparación.

Si el resultado de la **subconsulta está vacío**, se convierte a una fila con el valor nulo en todas las columnas.

EJEMPLO: Obtener los nombres de los puertos cuya altura es mayor que la media de altura de los puertos de 2ª categoría.

---


1. ¿En qué tablas se encuentra la información?

**Puerto** ==> **FROM Puerto**

2. ¿Qué condición deben cumplir las filas resultantes?

**altura > AVG(altura) de los Puertos de segunda categoría**

*Es un valor - una columna  
con una fila*



**==> WHERE altura > (SELECT AVG(altura) FROM Puerto  
WHERE categoria = '2' );**



*Compara cada valor de altura con el valor obtenido  
en avg(altura)*

EJEMPLO: Obtener los nombres de los puertos cuya altura es mayor que la media de altura de los puertos de 2ª categoría.

---

1. ¿En qué tablas se encuentra la información?

**Puerto** ==> **FROM Puerto**

2. ¿Qué condición deben cumplir las filas resultantes?

**altura > AVG(altura) de los Puertos de segunda categoría**

**==> WHERE altura > (SELECT AVG(altura) FROM Puerto  
WHERE categoria = '2' );**

3. ¿Qué información queremos visualizar?

**nompuerto** ==> **SELECT nompuerto** ==> *1 columna  
con n filas*

EJEMPLO: Obtener los nombres de los puertos cuya altura es mayor que la media de altura de los puertos de 2ª categoría.

---

```
SELECT nompuerto
FROM Puerto
WHERE altura > (SELECT AVG(altura)
                FROM Puerto
                WHERE categoria = '2' );
```

EJEMPLO: Obtener los nombres de los puertos cuya altura es mayor que la media de altura de los puertos de 2ª categoría.

~~SELECT **nompuerto** FROM **Puerto**  
WHERE **altura** > (SELECT **altura** FROM **Puerto**  
WHERE **categoria** = '2' );~~

*Es un valor a la vez*

*1 columna con n filas*

==> No puede hacer la comparación

INCORRECTO: (error de ejecución).



EJEMPLO: Obtener los nombres de los puertos cuya altura es mayor que la media de altura de los puertos de 2ª categoría.

---

INCORRECTO: (error de ejecución):

```
SELECT nompuerto  
FROM Puerto  
WHERE altura > AVG (SELECT altura FROM Puerto  
WHERE categoría = '2');
```

EJEMPLO: Obtener el nombre de la ciudad de salida y de llegada de las etapas donde estén los puertos con mayor pendiente.

---

```
SELECT DISTINCT E.salida, E.llegada
FROM Etapa E, Puerto P
WHERE E.netapa = P.netapa
      AND pendiente = (SELECT MAX(pendiente)
                        FROM Puerto) ;
```

### SINTAXIS:

*expresión* [NOT] IN (*expresión\_tabla*)

EJEMPLO: Obtener el nº de las etapas ganadas por ciclistas con edad superior a los 30 años.

```
SELECT netapa
FROM Etapa
WHERE dorsal IN ( SELECT dorsal
                  FROM Ciclista
                  WHERE edad > 30);
```

## SUBCONSULTAS ENCADENADAS

---

EJEMPLO: Obtener el número de las etapas ganadas por ciclistas que pertenezcan a equipos cuyo director tenga un nombre que empiece por 'A'.

```
SELECT etapa FROM Etapa
WHERE dorsal IN
    (SELECT dorsal FROM Ciclista
     WHERE nomeq IN (SELECT nomeq FROM Equipo
                     WHERE director LIKE 'A%'));
```

# Predicado exists

---

## Sintaxis:

EXISTS (*expresión\_tabla*)

- El predicado EXISTS se evalúa a cierto si la expresión\_tabla (un SELECT) devuelve al menos una fila.
- En general, **IN y EXISTS son intercambiables** y se pueden eliminar haciendo consultas a múltiples tablas e igualando por claves ajenas.

(esto **NO** es cierto para NOT IN y NOT EXISTS)

EJEMPLO: Obtener el nombre de los ciclistas que han ganado alguna etapas.

---

```
SELECT DISTINCT nombre  
FROM Ciclista C, Etapa E  
WHERE E.dorsal = C.dorsal;
```

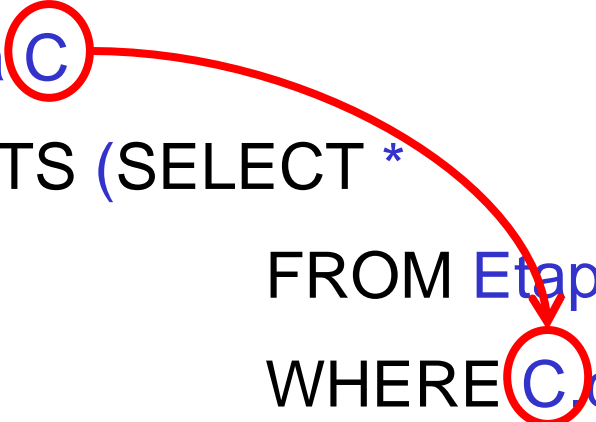
```
SELECT nombre  
FROM Ciclista  
WHERE dorsal IN (SELECT dorsal  
                  FROM Etapa E);
```

EJEMPLO: Obtener el nombre de los ciclistas que han ganado alguna etapas.

---

```
SELECT nombre
FROM Ciclista
WHERE dorsal IN (SELECT dorsal
                  FROM Etapa E);
```


```
SELECT nombre
FROM Ciclista C
WHERE EXISTS (SELECT *
              FROM Etapa E
              WHERE C.dorsal = E.dorsal);
```



EJEMPLO: Obtener el nombre de aquellos ciclistas que han llevado un maillot de un premio menor de 8000 eur.

---

```
SELECT C.nombre FROM Ciclista C
WHERE EXISTS ( SELECT *
                FROM Maillot M, Llevar L
                WHERE M.premio < 8000 AND M.codigo = L.codigo
                AND C.dorsal = L.dorsal)
```



**O bien:**

```
SELECT C.nombre FROM Ciclista C
WHERE C.dorsal IN ( SELECT L.dorsal
                   FROM Llevar L, Maillot M
                   WHERE M.premio < 8000
                   AND L.codigo = M.codigo)
```



EJEMPLO: Obtener el nombre de los ciclistas que **no** han ganado etapas.

---

~~SELECT nombre  
FROM Ciclista C, Etapa E  
WHERE C.dorsal <> E.dorsal ;~~

SELECT nombre  
FROM Ciclista C  
WHERE dorsal NOT IN (SELECT dorsal  
FROM Etapa E);

EJEMPLO: Obtener el nombre de los ciclistas que **no** han ganado etapas.

---

```
SELECT nombre
FROM Ciclista C
WHERE dorsal NOT IN (SELECT dorsal
                     FROM Etapa E
                     WHERE E.dorsal IS NOT NULL);
```

```
SELECT nombre
FROM Ciclista C
WHERE NOT EXISTS (SELECT *
                  FROM Etapa E
                  WHERE E.dorsal = C.dorsal);
```

EJEMPLO: Obtener el nombre de los ciclistas que **no** han ganado etapas.

---

La expresión: WHERE NOT EXISTS (SELECT \* FROM ...)

equivale a: WHERE 0 = (SELECT COUNT(\*) FROM ...)

```
SELECT nombre
```

```
FROM Ciclista C
```

```
WHERE 0 = (SELECT COUNT(*)
```

```
        FROM Etapa E
```

```
        WHERE E.dorsal = C.dorsal);
```

## Evaluación de los predicados con subconsultas vacías

PREDICADO	EVALUACIÓN
[expresión subconsulta] $\alpha$ [expresión subconsulta]	INDEFINIDO
expresión IN (subconsulta)	FALSO
EXISTS (subconsulta)	FALSO

Siendo  $\alpha$  cualquiera de: =, <>, >, <, >=, <=

# Cuantificación universal

Uso de EXISTS en condiciones que necesitan cuantificación universal  
(NO DISPONIBLE EN SQL de Oracle)

$$\forall x (F(x) \rightarrow G(x)) \equiv \neg \exists x (F(x) \wedge \neg G(x))$$

**Ejemplo:**

*“Obtener el nombre del ciclista que ha ganado todas las etapas de más de 200 km.”*

Consulta equivalente:

*“Obtener el nombre del ciclista tal que **no** existe una etapa de más de 200 km. que él **no** haya ganado”*

Nota: Se sabe que en esta base de datos hay etapas de más de 200 km.

“Obtener el nombre del ciclista tal que *no* existe una etapa de más de 200 km. que él *no* haya ganado (que la haya ganado otro)”

```
SELECT nombre
FROM Ciclista C
WHERE NOT EXISTS ( SELECT *
                    FROM Etapa E
                    WHERE km > 200 AND
                           C.dorsal <> E.dorsal );
```

Nota: Se sabe que en esta base de datos hay etapas de más de 200 km.

Problema: ¿Qué pasa si no hay etapas de más de 200 Km.?

```
SELECT nombre
FROM Ciclista C
WHERE NOT EXISTS ( SELECT *
                    FROM Etapa E
                    WHERE km > 200 AND
                           C.dorsal <> E.dorsal );
```

FALSO para toda etapa

CIERTO para todo ciclista

Se mostrarán los nombres de todos los ciclistas !!!

Solución:

Añadir la condición de que exista alguna etapa de más de 200 Km.

“Obtener el nombre del ciclista que ha ganado todas las etapas de más de 200 km.”

---

```
SELECT nombre
FROM Ciclista C
WHERE NOT EXISTS (SELECT * FROM Etapa E
                  WHERE km > 200 AND
                        C.dorsal <> E.dorsal )
AND EXISTS ( SELECT *
             FROM Etapa E
             WHERE km > 200) ;
```



# Cuantificación universal

Uso de COUNT en condiciones que necesitan cuantificación universal  
(NO DISPONIBLE EN SQL de Oracle)

$$\forall x (F(x) \rightarrow G(x)) \equiv$$

$$COUNT(\{x | F(x) \wedge G(x)\}) = COUNT(\{x | F(x)\})$$

**Ejemplo:**

*“Obtener el nombre del ciclista que ha ganado todas las etapas de más de 200 km.”*

Consulta equivalente:

*“Obtener el nombre del ciclista tal que el número de etapas de más de 200 km. que ha ganado coincide con el número total de etapas de más de 200 km.”*

Nota: Se sabe que en esta base de datos hay etapas de más de 200 km.

“Obtener el nombre del ciclista tal que el número de etapas de más de 200 km. que ha ganado coincide con el número total de etapas de más de 200 km.”

```
SELECT nombre
FROM Ciclista C
WHERE
    (SELECT COUNT(*)
     FROM Etapa E
     WHERE km > 200 AND E.dorsal = C.dorsal )
    =
    (SELECT COUNT(*)
     FROM Etapa E
     WHERE km > 200);
```

Nota: Se sabe que en esta base de datos hay etapas de más de 200 km.

Problema: ¿Qué pasa si no hay etapas de más de 200 Km.?

```
SELECT nombre  
FROM Ciclista C  
WHERE
```

```
(SELECT COUNT(*)  
FROM Etapa E  
WHERE km > 200 AND E.dorsal = C.dorsal )
```

FALSO para toda etapa

```
=  
(SELECT COUNT(*)  
FROM Etapa E  
WHERE km > 200);
```

0=0 para  
todo ciclista

Se mostrarán los nombres de todos los ciclistas !!!

Solución: Añadir la condición de que exista alguna etapa de más de 200 Km.

*“Obtener el nombre del ciclista que ha ganado todas las etapas de más de 200 km.”*

---

*“Obtener el nombre del ciclista tal que el número de etapas de más de 200 km. que ha ganado coincide con el número total de etapas de más de 200 km. y hay etapas de más de 200km.”*

```
SELECT nombre
FROM Ciclista C
WHERE
    (SELECT COUNT(*) FROM Etapa E
    WHERE km > 200 AND E.dorsal = C.dorsal )
    =
    (SELECT COUNT(*) FROM Etapa E
    WHERE km > 200)
AND
    (SELECT COUNT(*) FROM Etapa E
    WHERE km > 200) > 0;
```

“Obtener el nombre de los equipos tales que todos sus corredores hayan llevado algún maillot o hayan ganado algún puerto”

$$\neg (A \vee B) = \neg (A) \wedge \neg (B)$$

```
SELECT E.nomeq
FROM Equipo E
WHERE NOT EXISTS (SELECT * FROM Ciclista C
                  WHERE E.nomeq = C.nomeq
                  AND NOT EXISTS (SELECT * FROM Llevar L
                                WHERE C.dorsal=L.dorsal )
                  AND NOT EXISTS (SELECT * FROM Puerto P
                                WHERE C.dorsal=P.dorsal ) )
AND EXISTS (SELECT * FROM Ciclista C
            WHERE E.nomeq = C.nomeq)
```

“Obtener el nombre de los equipos tales que todos sus corredores hayan llevado algún maillot o hayan ganado algún puerto”

Con COUNT

```
SELECT E.nomeq
FROM Equipo E
WHERE (SELECT COUNT(*) FROM Ciclista C
      WHERE E.nomeq = C.nomeq
      AND (EXISTS (SELECT * FROM Llevar L
                  WHERE C.dorsal=L.dorsal )
      OR EXISTS (SELECT * FROM Puerto P
                  WHERE C.dorsal=P.dorsal )))
      = (SELECT COUNT(*) FROM Ciclista C
        WHERE E.nomeq = C.nomeq)
      AND (SELECT COUNT(*) FROM Ciclista C
          WHERE E.nomeq = C.nomeq) > 0;
```

“Obtener el nombre de los ciclistas que han ganado todos los puertos de una etapa y además han ganado esa etapa.”

---

```
SELECT DISTINCT C.nombre
FROM Ciclista C, Etapa E
WHERE E.dorsal = C.dorsal
      AND NOT EXISTS ( SELECT * FROM Puerto P
                        WHERE P.netapa = E.netapa
                           AND C.dorsal <> P.dorsal );
AND EXISTS ( SELECT * FROM Puerto P
             WHERE P.netapa = E.netapa );
```

Porque puede haber etapas sin puertos

## UD 2.1 El lenguaje SQL: consultas y actualización (DML)

---

- 1.- Introducción a SQL
- 2.- SELECT. Consultas sencillas sobre una tabla
- 3.- SELECT. Consultas simples sobre varias tablas
- 4.- SELECT. Consultas complejas: Subconsultas
- 5.- SELECT. Con. complejas: Agrupación (GROUP BY)
- 6.- SELECT. Operadores conjuntistas
- 7.- SELECT. Operador JOIN
- 8.- Lenguaje de Manipulación de Datos (DML)
  - 8.1.- Instrucción INSERT
  - 8.2.- Instrucción DELETE
  - 8.3.- Instrucción UPDATE
- 9.- Instrucciones de control de transacciones



## 5.- SELECT. Consultas complejas: Agrupación

Un grupo lo forma un conjunto de filas con el mismo valor para el conjunto de columnas por las que se agrupa.

EJEMPLO: Obtener el nombre de cada equipo y la edad media de sus ciclistas.

nomeq	edad
Banesto	22
ONCE	25
PDM	32
Banesto	25
Kelme	28
ONCE	30
Kelme	29
Banesto	28

⇒

nomeq	edad
Banesto	22
Banesto	25
Banesto	28
ONCE	25
ONCE	30
PDM	32
Kelme	28
Kelme	29

⇒

nomeq	AVG(edad)
Banesto	25
ONCE	27,5
PDM	32
Kelme	28,5

## 5.- SELECT. Consultas complejas: GROUP BY

Un grupo lo forma un conjunto de filas con el mismo valor para el conjunto de columnas incluidas en el GROUP BY.

Ejemplo: Obtener el nombre de cada equipo y la edad media de los ciclistas de dicho equipo:

```
SELECT nomeq, AVG(edad)  
FROM Ciclista  
GROUP BY nomeq;
```

	nomeq	edad	
	Banesto	22	←
	ONCE	25	←
	PDM	32	←
	Banesto	25	←
	Kelme	28	←
	ONCE	30	←
	Kelme	29	←
	Banesto	28	←

Las **funciones agregadas** en las consultas agrupadas funcionan de forma diferente que en las consultas normales, devolviendo **un valor por cada grupo** formado.

	nomeq	edad	
	Banesto	22	
	Banesto	25	
	Banesto	28	
	ONCE	25	
	ONCE	30	
	PDM	32	
	Kelme	29	
	Kelme	28	



Un valor  
Por grupo

```
SELECT nomeq, AVG(edad)
FROM Ciclista
GROUP BY nomeq;
```

**La solución, es:**

nomeq	edad
Banesto	25
ONCE	27,5
PDM	32
Kelme	28,5

## 5.- SELECT. Consultas complejas: GROUP BY

---

### SINTAXIS

```
SELECT [ALL | DISTINCT] {expresión1, expresión2,..., expresiónn|*}  
FROM tabla1, tabla2 ..., tablan  
[WHERE condición]  
[GROUP BY columna1, columna2,..., columnan  
          [HAVING expresión_condicional]]  
[ORDER BY columna1, columna2,..., columnan]
```

# GROUP BY, WHERE y HAVING

---

La cláusula **HAVING** sólo puede ir en consultas agrupadas y es similar a **WHERE**, pero actúa en orden diferente:

- 1º) Condición **WHERE** (se usa para las filas)
- 2º) Agrupamiento y cálculo de valores agregados
- 3º) Condición **HAVING** (se usa para los grupos)

En la cláusula **HAVING** sólo podrán aparecer directamente columnas por las que se agrupa, funciones agregadas o constantes.

## EJEMPLO INCORRECTO:

---

```
SELECT nomeq, nombre, AVG(edad)
FROM Ciclista
GROUP BY nomeq;
```

La regla sintáctica que utilizan los sistemas relacionales para asegurar el correcto funcionamiento de las consultas agrupadas es el siguiente: “en la **selección** de una consulta agrupada, solo pueden aparecer referencias a **columnas por las que se agrupa**, referencias a **funciones agrupadas** o **constantes**”

# GROUP BY, WHERE y HAVING

---

Si se incluye la cláusula **WHERE**, la aplicación de esta cláusula se produce **previamente** a la agrupación.

5 ----> SELECT nomeq, AVG(edad)

1 ----> FROM Ciclista

2 ----> WHERE edad > 25

3 ----> GROUP BY nomeq;

4 ----> HAVING nomeq LIKE "K%"



---

En las consultas agrupadas se pueden anidar las funciones agregadas.

Ej. Obtener el valor de la edad media del equipo con mayor media de edad de sus ciclistas.

```
SELECT MAX(AVG(edad))  
FROM Ciclista  
GROUP BY nomeq;
```

---

Obtener el nombre de cada equipo y la edad media de sus ciclistas con **más de 25 años**, de aquellos equipos con **más de 3 corredores** mayores de 25 años.

```
SELECT nomeq, AVG(edad)
FROM Ciclista
WHERE edad > 25
GROUP BY nomeq
HAVING COUNT(dorsal) > 3;
```

Obtener el **nombre del ciclista** y el número de puertos que ha ganado, siendo la **media de la pendiente** de éstos superior a 10.

```
SELECT C.nombre, COUNT(P.nompuerto)
FROM Ciclista C, Puerto P
WHERE C.dorsal = P.dorsal
GROUP BY C.dorsal, C.nombre /*Agrupar por CP */
HAVING AVG (P.pendiente) >10;
```

Obtener el **nombre de los ciclistas** que han ganado alguna etapa y que pertenezcan a un equipo que tenga más de 5 corredores, indicando cuántas etapas ha ganado.

```
SELECT C.nombre, count(*)
FROM ciclista C, etapa E
WHERE C.dorsal = E.dorsal
      AND 5 < ( SELECT count(*)
                FROM Ciclista C2
                WHERE C2.nomeq = C.nomeq )
GROUP BY C.nombre, C.dorsal;
```

## UD 2.1 El lenguaje SQL: consultas y actualización (DML)

---

- 1.- Introducción a SQL
- 2.- SELECT. Consultas sencillas sobre una tabla
- 3.- SELECT. Consultas simples sobre varias tablas
- 4.- SELECT. Consultas complejas: Subconsultas
- 5.- SELECT. Con. complejas: Agrupación (GROUP BY)
- 6.- SELECT. Operadores conjuntistas
- 7.- SELECT. Operador JOIN
- 8.- Lenguaje de Manipulación de Datos (DML)
  - 8.1.- Instrucción INSERT
  - 8.2.- Instrucción DELETE
  - 8.3.- Instrucción UPDATE
- 9.- Instrucciones de control de transacciones

## 6. SELECT. Operadores conjuntistas

---

Existen varias formas de combinar dos tablas en el lenguaje SQL dando lugar a una “expresión de tabla”:

- ✓ → Incluir varias tablas en la cláusula **FROM**.
- ✓ → Uso de **subconsultas** en las condiciones de las cláusulas **WHERE** o **HAVING**.
- Combinaciones **conjuntivistas** de tablas: utilizando operadores de la teoría de conjuntos para combinar las tablas.
- **Concatenaciones** de tablas: utilizando diferentes formas variantes del operador concatenación del Álgebra Relacional.

# Operadores conjuntistas

---

Corresponden a los operadores *unión*, *diferencia* e *intersección* del Álgebra Relacional.

- UNION
- EXCEPT (MINUS en Oracle)
- INTERSECT

Permiten combinar tablas que tengan esquemas compatibles.

# UNION

*expresión\_tabla* **UNION** [**ALL**] *término\_tabla*

Realiza la unión de las filas de las tablas provenientes de las dos expresiones.

Se permitirán o no duplicados según se incluya o no la opción **ALL**.

## Ejemplo:

*Obtener el nombre de todo el personal de la vuelta.*

(SELECT nombre FROM Ciclista)

**UNION**

(SELECT director FROM Equipo)



---

Obtener el nombre de los ciclistas que han llevado un maillot o han ganado un puerto o una etapa.

```
SELECT nombre
FROM Ciclista
WHERE dorsal IN
    (SELECT dorsal FROM Llevar
     UNION
     SELECT dorsal FROM Puerto
     UNION
     SELECT dorsal FROM Etapa)
```

# INTERSECCION

---

*expresión\_tabla* **INTERSECT** *término\_tabla*

Realiza la intersección de las filas de las tablas provenientes de las dos expresiones.

## **Ejemplo:**

*Obtener los nombres de las personas que son tanto ciclistas como directores de equipo.*

```
(SELECT nombre FROM Ciclista)  
INTERSECT  
(SELECT director FROM Equipo)
```

# DIFERENCIA

---

*expresión\_tabla* **EXCEPT** *término\_tabla*

Realiza la diferencia de las filas de las tablas provenientes de las dos expresiones.

## **Ejemplo:**

*Obtener los nombres que aparecen en la tabla de ciclistas y no en la de directores.*

```
(SELECT nombre FROM Ciclista)  
EXCEPT  
(SELECT director FROM Equipo)
```

## UD 2.1 El lenguaje SQL: consultas y actualización (DML)

---

- 1.- Introducción a SQL
- 2.- SELECT. Consultas sencillas sobre una tabla
- 3.- SELECT. Consultas simples sobre varias tablas
- 4.- SELECT. Consultas complejas: Subconsultas
- 5.- SELECT. Con. complejas: Agrupación (GROUP BY)
- 6.- SELECT. Operadores conjuntistas
- 7.- SELECT. Operador JOIN
- 8.- Lenguaje de Manipulación de Datos (DML)
  - 8.1.- Instrucción INSERT
  - 8.2.- Instrucción DELETE
  - 8.3.- Instrucción UPDATE
- 9.- Instrucciones de control de transacciones

## 7. Instrucción SELECT. Operador JOIN

---

### CONCATENACIONES DE TABLAS

Corresponden a variantes del operador concatenación de Álgebra Relacional.

1. Producto cartesiano (**CROSS JOIN**)
2. Concatenación interna (**INNER JOIN**)
3. Concatenación externa (**OUTER JOIN**)

## 1. Producto Cartesiano (CROSS JOIN)

*referencia\_tabla1* **CROSS JOIN** *referencia\_tabla2*

=

SELECT \*

FROM *referencia\_tabla1*, *referencia\_tabla2*

## 2. Concatenación Interna

Sintaxis completa (3 formas de uso):

referencia\_tabla

[**NATURAL**] [**INNER**] **JOIN**

referencia\_tabla

[**ON** condición | **USING** (*columna<sub>1</sub>*, *columna<sub>2</sub>*, ..., *columna<sub>n</sub>*)]

Forma 1: *tabla1* [INNER] JOIN *tabla2* ON *expr\_condicional*

---

≡ SELECT \*

FROM *tabla1, tabla2*

WHERE *expresión\_condicional*

**EJEMPLO:** Obtener los nombres de los puertos y el número de la etapa en la que están, si la etapa anterior tiene más de 200 km.

```
SELECT nompuerto, P.netapa  
FROM Puerto P JOIN Etapa E ON P.netapa= E.netapa+1  
WHERE E.km>200
```



Forma 2: *tabla1* [INNER] JOIN *tabla2* USING (c1, ... cn)

---

```
≡ SELECT *  
    FROM tabla1, tabla2  
    WHERE tabla1.c1 = tabla2.c1  
          AND tabla1.c2 = tabla2.c2  
          AND..... AND tabla1.cn = tabla2.cn
```

**EJEMPLO:** Obtener los nombres de los puertos, el número de la etapa en la que están y la longitud de la etapa, para los puertos de altura superior a 800.

```
SELECT nompuerto, netapa, km  
FROM Puerto JOIN Etapa USING (netapa)  
WHERE altura>800
```

### Forma 3: *tabla1* NATURAL INNER JOIN *tabla2*

≡ *tabla1* JOIN *tabla2* USING ( c1, c2, ..., cn)

donde *tabla1* y *tabla2* tienen *n* atributos

(Es un JOIN por todos los atributos comunes de las tablas)

**EJEMPLO:** Obtener los nombres de los ciclistas pertenecientes al equipo dirigido por 'Alvaro Pino'.

```
SELECT nombre  
FROM Ciclista NATURAL JOIN Equipo  
WHERE director = 'Alvaro Pino';
```

**EJEMPLO:** Obtener los nombres de los puertos, el número de la etapa en la que están y la longitud de la etapa, para los puertos de altura superior a 800.

```
SELECT nompuerto, netapa, km  
FROM Puerto JOIN Etapa USING (netapa)  
WHERE altura>800
```

Sería incorrecto:

```
SELECT nompuerto, netapa, km  
FROM Puerto NATURAL JOIN Etapa  
WHERE P.altura>800
```

concatenaría también por **dorsal**

Obtener el dorsal y nombre de los ciclistas junto al total de maillots llevados por éste.

---

```
SELECT C.dorsal, C.nombre, COUNT (DISTINCT L.codigo)
FROM Ciclista C, Llevar L
WHERE C.dorsal = L.dorsal
GROUP BY C.dorsal, C.nombre
```

```
SELECT dorsal, ciclista.nombre, COUNT (DISTINCT llevar.codigo)
FROM Ciclista NATURAL INNER JOIN Llevar
GROUP BY dorsal, ciclista.nombre
```

### 3. Concatenación Externa

Útil cuando se desea que las filas que no se combinen también aparezcan en el resultado final

referencia\_tabla

[**NATURAL**] {**LEFT** | **RIGHT** | **FULL**} [**OUTER**] **JOIN**

referencia\_tabla

[**ON** condición| **USING** (*columna<sub>1</sub>*, *columna<sub>2</sub>*,..., *columna<sub>n</sub>*) ]

Tabla1 **LEFT JOIN** Tabla2 **ON** exp\_condicional

Concatenación interna de *Tabla1* y *Tabla2* UNION tuplas de *Tabla1* que no están en la concatenación interna con valores nulos en el resto de columnas

**FULL:** se muestran las tuplas no concatenadas de *tabla1* y *tabla2*

## EJEMPLO:

Obtener, para **cada ciclista**, su dorsal, su nombre, el código de cada maillot que ha llevado y el número de etapa en la que lo ha llevado.

```
SELECT C.dorsal, nombre, codigo, netapa  
FROM Ciclista C LEFT JOIN Llevar L ON C.dorsal = L.dorsal
```

Obtener el nombre de todos los ciclistas y la cantidad de etapas que ha ganado cada uno.

---

```
SELECT nombre, COUNT(netapa)
FROM Ciclista NATURAL LEFT JOIN Etapa
GROUP BY dorsal, nombre
```

Obtener el nombre de todos los equipos indicando cuántos ciclistas tiene cada uno.

---

```
SELECT equipo.nomeq, COUNT(dorsal)
FROM Equipo LEFT JOIN Ciclista
                ON equipo.nomeq= ciclista.nomeq
GROUP BY equipo.nomeq
```

```
(SELECT nomeq, count(*)
FROM ciclista
GROUP BY nomeq)
UNION
(SELECT nomeq, 0
FROM equipo
WHERE nomeq NOT IN (SELECT nomeq FROM ciclista));
```



## UD 2.1 El lenguaje SQL: consultas y actualización (DML)

---

- 1.- Introducción a SQL
- 2.- SELECT. Consultas sencillas sobre una tabla
- 3.- SELECT. Consultas simples sobre varias tablas
- 4.- SELECT. Consultas complejas: Subconsultas
- 5.- SELECT. Con. complejas: Agrupación (GROUP BY)
- 6.- SELECT. Operadores conjuntistas
- 7.- SELECT. Operador JOIN
- 8.- Lenguaje de Manipulación de Datos (DML)
  - 8.1.- Instrucción INSERT
  - 8.2.- Instrucción DELETE
  - 8.3.- Instrucción UPDATE
- 9.- Instrucciones de control de transacciones

## 8. Lenguaje de Manipulación de Datos (DML)

---

Las instrucciones que modifican los datos sólo pueden aplicarse a una tabla cada vez.

- **INSERT** (para la inserción de tuplas enteras)
- **DELETE** (para el borrado de tuplas enteras)
- **UPDATE** (para la modificación de uno o más atributos en una o más tuplas)

## UD 2.1 El lenguaje SQL: consultas y actualización (DML)

---

- 1.- Introducción a SQL
- 2.- SELECT. Consultas sencillas sobre una tabla
- 3.- SELECT. Consultas simples sobre varias tablas
- 4.- SELECT. Consultas complejas: Subconsultas
- 5.- SELECT. Con. complejas: Agrupación (GROUP BY)
- 6.- SELECT. Operadores conjuntistas
- 7.- SELECT. Operador JOIN
- 8.- Lenguaje de Manipulación de Datos (DML)
  - 8.1.- Instrucción INSERT
  - 8.2.- Instrucción DELETE
  - 8.3.- Instrucción UPDATE
- 9.- Instrucciones de control de transacciones

## 8.1. Instrucción INSERT

```
INSERT INTO tabla [(column1, columna2,..., columnan)]  
    { DEFAULT VALUES |  
      VALUES (átomo1, átomo2,... átomon) |  
      expresión_tabla }
```

- Si no se incluye la lista de columnas se deberán insertar filas completas de *tabla*.
- Si se incluye **DEFAULT VALUES** se insertará una única fila con los valores por defecto en cada columna (según la definición de *tabla*).
- Con **VALUES**(átomo<sub>1</sub>, átomo<sub>2</sub>,..., átomo<sub>n</sub>), un átomo es una expresión escalar del tipo de datos apropiado (texto, entero,...)
- Con **expresión\_tabla**, se insertarán las filas resultantes de la ejecución de la expresión SELECT .

---

```
INSERT INTO tabla [(column1, columna2,..., columnan)]  
    { DEFAULT VALUES |  
      VALUES (átomo1, átomo2,... átomon) |  
      expresión_tabla }
```

Ejemplo de inserción de una **tupla completa**:

Añadir un ciclista de dorsal 101, nombre 'Joan Peris', y del equipo 'Kelme' , de 27 años.

```
INSERT INTO Ciclista  
VALUES (101, 'Joan Peris', 27,'Kelme');
```

---

Ejemplo de inserción de una **tupla incompleta**:

Añadir un ciclista de dorsal 101, nombre 'Joan Peris', y del equipo 'Kelme' (no sabemos la edad).

```
INSERT INTO Ciclista (dorsal, nombre, nomeq)  
VALUES (101, 'Joan Peris', 'Kelme');
```

---

Ejemplo de **inserción múltiple**:

Añadir a la tabla 'Ciclista\_ganador', (con el mismo esquema que Ciclista), la información de los ciclistas que hayan ganado alguna etapa.

```
INSERT INTO Ciclista_ganador
SELECT *
FROM Ciclista
WHERE dorsal IN (SELECT dorsal FROM etapa)
```

## UD 2.1 El lenguaje SQL: consultas y actualización (DML)

---

- 1.- Introducción a SQL
- 2.- SELECT. Consultas sencillas sobre una tabla
- 3.- SELECT. Consultas simples sobre varias tablas
- 4.- SELECT. Consultas complejas: Subconsultas
- 5.- SELECT. Con. complejas: Agrupación (GROUP BY)
- 6.- SELECT. Operadores conjuntistas
- 7.- SELECT. Operador JOIN
- 8.- Lenguaje de Manipulación de Datos (DML)
  - 8.1.- Instrucción INSERT
  - 8.2.- Instrucción DELETE
  - 8.3.- Instrucción UPDATE
- 9.- Instrucciones de control de transacciones



## 8.2. Instrucción DELETE

---

**DELETE FROM** *tabla* [ **WHERE** condición ]

Si se incluye la cláusula WHERE se eliminarán aquellas tuplas que hagan cierta la condición; si no se incluye, se eliminarán todas las tuplas de la tabla.

### Ejemplo:

Eliminar la información del ciclista 'M. Indurain' ya que se ha jubilado.

```
DELETE FROM Ciclista
```

```
WHERE nombre = 'M. Indurain'
```

## UD 2.1 El lenguaje SQL: consultas y actualización (DML)

---

- 1.- Introducción a SQL
- 2.- SELECT. Consultas sencillas sobre una tabla
- 3.- SELECT. Consultas simples sobre varias tablas
- 4.- SELECT. Consultas complejas: Subconsultas
- 5.- SELECT. Con. complejas: Agrupación (GROUP BY)
- 6.- SELECT. Operadores conjuntistas
- 7.- SELECT. Operador JOIN
- 8.- Lenguaje de Manipulación de Datos (DML)
  - 8.1.- Instrucción INSERT
  - 8.2.- Instrucción DELETE
  - 8.3.- Instrucción UPDATE
- 9.- Instrucciones de control de transacciones

## 8.3. Instrucción UPDATE

---

**UPDATE** *tabla*

**SET** *asignación*<sub>1</sub>, *asignación*<sub>2,...</sub>, *asignación*<sub>n</sub>

[ **WHERE** *expresión\_condicional* ]

donde una *asignación* es de la forma:

*columna* = {DEFAULT | NULL | *expresión\_escalar*}

### **Ejemplo:**

Incrementar un 10% los premios de los maillots.

**UPDATE** Maillot

**SET** premio = premio \* 1.10

## Ejemplo:

Los ciclistas del Kelme se cambian todos al equipo *K10* recientemente creado.

```
UPDATE Ciclista  
SET nomeq = 'K10'  
WHERE nomeq='Kelme' ;
```

## UD 2.1 El lenguaje SQL: consultas y actualización (DML)

---

- 1.- Introducción a SQL
- 2.- SELECT. Consultas sencillas sobre una tabla
- 3.- SELECT. Consultas simples sobre varias tablas
- 4.- SELECT. Consultas complejas: Subconsultas
- 5.- SELECT. Con. complejas: Agrupación (GROUP BY)
- 6.- SELECT. Operadores conjuntistas
- 7.- SELECT. Operador JOIN
- 8.- Lenguaje de Manipulación de Datos (DML)
  - 8.1.- Instrucción INSERT
  - 8.2.- Instrucción DELETE
  - 8.3.- Instrucción UPDATE
- 9.- Instrucciones de control de transacciones

## 9. Instrucciones de control de transacciones

---

### Inicio de transacción:

Implícito (no hay instrucción) en cada inicio de sesión y cuando termina otra transacción.

### Fin de transacción:

2 instrucciones posibles:

- **COMMIT**: indica el fin de una transacción que se desea confirmar.
- **ROLLBACK**: indica el fin de una transacción que se descarta.

## Ejemplo:

El nombre del equipo 'Banesto' pasa a ser 'BanQué'

```
UPDATE Equipo SET nombre = '¿BanQué?'  
WHERE nomeq = 'Banesto';
```

```
UPDATE Ciclista SET nomeq = '¿BanQué?'  
WHERE nomeq = 'Banesto';
```

```
COMMIT;
```



Fin de la transacción aceptando los cambios

## Ejemplo:

El nombre del equipo 'Banesto' pasa a ser 'BanQué'

-- falta diferir la comprobación de la C. Ajena de Ciclista a Equipo

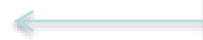
```
UPDATE Equipo SET nombre = '¿BanQué?'
```

```
WHERE nomeq = 'Banesto';
```

```
UPDATE Ciclista SET nomeq = '¿BanQué?'
```

```
WHERE nomeq = 'Banesto';
```

```
COMMIT;
```



Fin de la transacción aceptando los cambios



## Ejemplo:

El nombre del equipo 'Banesto' pasa a ser 'BanQué'

```
SET CONSTRAINT ca_ciclista_equipo DEFERRED;
```

```
UPDATE Equipo SET nombre = '¿BanQué?'
```

```
WHERE nomeq = 'Banesto';
```

```
UPDATE Ciclista SET nomeq = '¿BanQué?'
```

```
WHERE nomeq = 'Banesto';
```

```
COMMIT;
```



Fin de la transacción aceptando los cambios

---

# Repaso

# Repaso

Operador	Álgebra Relacional	SQL
Selección	$R \text{ Donde } F$	SELECT ... FROM R WHERE F
Proyección	$R [A_i, A_j, \dots, A_k]$	SELECT $A_i, A_j, \dots, A_k$ FROM R
Producto Cartesiano	$R_1 \times R_2, \dots \times R_n$	SELECT ... FROM $R_1, R_2, \dots, R_n$ SELECT...FROM $R_1$ CROSS JOIN $R_2, \dots,$ CROSS JOIN $R_n$
Concatenación	$R_1 \ R_2$	SELECT... FROM $R_1$ NATURAL JOIN $R_2$
Unión	$R_1 \cup R_2$	SELECT * FROM $R_1$ UNION SELECT * FROM $R_2$
Diferencia	$R_1 - R_2$	SELECT * FROM $R_1$ EXCEPT SELECT * FROM $R_2$
Intersección	$R_1 \cap R_2$	SELECT * FROM $R_1$ INTERSECT SELECT * FROM $R_2$

## 1. Nombre del ciclista más joven

---

```
SELECT nombre  
FROM Ciclista  
WHERE edad = ( SELECT MIN(edad) FROM Ciclista )
```

2. Obtener el número de las etapas y la ciudad de salida de aquellas etapas que no tengan puertos de montaña.

---

```
SELECT netapa, salida
FROM Etapa
WHERE NOT EXISTS (SELECT *
                   FROM Puerto
                   WHERE Puerto.netapa=Etapa.netapa )
```

3. Obtener el nombre de la ciudad de **salida** y de **llegada** de la etapa donde está el puerto con **mayor pendiente**.

---

```
SELECT DISTINCT E.salida, E.llegada
FROM Etapa e, Puerto p
WHERE E.netapa=P.netapa AND
      P.pendiente=( SELECT MAX(pendiente)
                    FROM puerto )
```

4. Obtener el nombre de los ciclistas que **han ganado todos los puertos** de una etapa y además han ganado esa **misma etapa**.

```
SELECT DISTINCT C.nombre
```

```
FROM Ciclista C, Etapa E
```

```
WHERE E.dorsal=C.dorsal AND
```

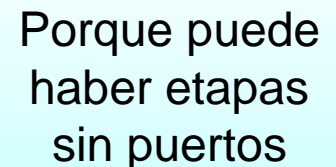
```
    NOT EXISTS ( SELECT * FROM Puerto P
```

```
                WHERE P.netapa=E.netapa
```

```
                AND C.dorsal <> P.dorsal )
```

```
    AND EXISTS ( SELECT * FROM Puerto P
                  WHERE P.netapa=E.netapa )
```

Porque puede  
haber etapas  
sin puertos



5. Obtener el **color** de aquellos **maillots** que **sólo** han sido llevados por ciclistas de un **mismo equipo**.

---

```
SELECT DISTINCT color
FROM Maillot M, Llevar L, Ciclista C
WHERE M.codigo = L.codigo AND
      C.dorsal = L.dorsal AND
      NOT EXISTS( SELECT * FROM Llevar L2, Ciclista C2
                  WHERE C2.dorsal = L2.dorsal AND
                        C2.nomeq <> C.nomeq AND
                        L2.codigo = M.codigo)
```



6. Obtener el nombre de los ciclistas que pertenezcan a un equipo que tenga más de cinco corredores indicando cuántas etapas ha ganado cada ciclista.

```
SELECT C.nombre, COUNT(E.dorsal)
FROM Ciclista C LEFT JOIN Etapa E ON (C.dorsal=E.dorsal)
WHERE 5 < ( SELECT COUNT(*)
            FROM Ciclista C2
            WHERE C2.nomeq=C.nomeq )
GROUP BY C.dorsa, C.nombre
```

# ¡ Observar la diferencia con el anterior !

---

Obtener el nombre de los ciclistas **que han ganado alguna etapa** y que pertenezcan a un equipo que tenga más de 5 corredores, indicando cuántas etapas ha ganado.

```
SELECT C.nombre, count(*)
FROM Ciclista C, Etapa E
WHERE C.dorsal = E.dorsal
      AND 5 < ( SELECT count(*)
                FROM Ciclista C2
                WHERE C2.nomeq = C.nomeq )
GROUP BY C.dorsal, C.nombre;
```

7. Nombre de los ciclistas que **no han llevado todos los maillots** que ha **llevado el ciclista** de dorsal 1.

---

*Ciclistas C tales que hay algún maillot llevado por el dorsal 1 que no ha sido llevado por el ciclista C*

SELECT C.nombre

FROM Ciclista C

WHERE EXISTS

(SELECT \* FROM Llevar L

WHERE L.dorsal = 1

AND NOT EXISTS (SELECT \*

FROM Llevar L2

WHERE L2.dorsal=C.dorsal

AND L2.codigo=L.codigo))

---

8. Obtener el nombre de los equipos y la media de la edad de sus ciclistas, de los equipos que tengan la media de edad máxima.

```
SELECT C.nomeq, AVG(C.edad)
FROM Ciclista C
GROUP BY C.nomeq
HAVING AVG(C.edad) = (SELECT MAX(AVG(D.edad))
                      FROM Ciclista D
                      GROUP BY D.nomeq)
```

9. Obtener la ciudad de **salida** de la etapa que tiene **más puertos** de montaña.

---

```
SELECT E.salida
FROM Puerto P, Etapa E
WHERE P.netapa = E.netapa
GROUP BY E.netapa, E.salida
HAVING COUNT(*) = ( SELECT MAX(COUNT(*))
                    FROM Puerto P2
                    GROUP BY P2.netapa )
```

10. Obtener el nombre de aquellos **equipos** tales que sus ciclistas **sólo** hayan ganado **puertos de 1ª categoría**.

```
SELECT E.nomeq
FROM Equipo E
WHERE NOT EXISTS
    ( SELECT * FROM Ciclista C, Puerto P
      WHERE C.dorsal = P.dorsal
        AND P.categoria <> '1'
        AND C.nomeq = E.nomeq )
AND EXISTS ( SELECT *
             FROM Ciclista C2, Puerto P2
             WHERE C2.dorsal = P2.dorsal
               AND C2.nomeq = E2.nomeq);
```



Algún ciclista del equipo E haya ganado algún puerto de 1ª Categoría