

Práctica 3:

Lenguaje SQL

1ª Parte: Manipulación de Bases de Datos

Objetivos:

- Presentar la sintaxis del lenguaje SQL (sólo del Lenguaje de Manipulación).
- Ver algunos ejemplos sencillos para clarificar la semántica del SQL.
- Presentar (o recordar) las bases de datos CICLISMO y MÚSICA.
- Realizar de menor a mayor complejidad consultas SQL sobre dichas bases de datos.
- Realizar todo lo anterior usando la herramienta ISQL del sistema de gestión de bases de datos ORACLE.

1. Lenguaje de Manipulación del SQL

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 cuatro instrucciones que componen el lenguaje de manipulación de datos 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.

1.1. Consultas: instrucción select

```
select [all | distinct] comalista_item_seleccionado | *  
from comalista_referencia_tabla  
[where expresión_condicional]  
[group by comalista_referencia_col]  
[having expresión_condicional]  
[order by comalista_referencia_col]
```

- *comalista_item_seleccionado*: información a obtener de la base de datos.
- **from** *comalista_referencia_tabla*: especifica de qué tablas se obtiene la información buscada.
- **where** *expresión_condicional*: expresa una condición que deben cumplir las filas de la consulta resultante.
- **group by** *comalista_referencia_col*: permite formar consultas agrupadas para extraer información global sobre los grupos formados.
- **having** *expresión_condicional*: condición sobre los grupos formados.
- **order by** *comalista_referencia_col*: ordena por una o varias columnas.

1.1.1. Condiciones en consultas simples

```
3  select [all | distinct] comalista_ítem_seleccionado | *  
1  from tabla  
2  [where expresión_condicional]  
4  [order by comalista_referencia_col]
```

- **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.

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

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

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

```
SELECT nombre, altura FROM Puerto  
WHERE categoria = 1;
```

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:

$$exp \text{ between } exp_1 \text{ and } exp_2 \equiv (exp \geq exp_1) \text{ and } (exp \leq exp_2)^5$$

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%’;
```

EJEMPLO: Obtener el nombre de los puertos de 1ª, 2ª o 3ª categoría.

```
SELECT nompuerto FROM Puerto
WHERE categoría IN ( 1, 2, 3 );
```

(*) También el predicado in es derivado y la expresión equivalente es:

$$exp \text{ in } (exp_1, exp_2, \dots, exp_n) \equiv (exp=exp_1) \text{ or } (exp=exp_2) \text{ or...or } (exp=exp_n)$$

EJEMPLO: Obtener todos los datos de aquellos ciclistas de los que se desconocía su edad.

```
SELECT * FROM Ciclista
WHERE edad IS NULL;
```

COMPARACIÓN DE VALORES NULOS

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 atrib1 = 50 y atrib2 fuera nulo, el resultado de la comparación sería indefinido y por tanto dicha fila no se incluiría en la selección.

Ejemplo de consulta incorrecta (error de sintaxis)

```
select nomeq  
from Equipo  
where director = null;
```

MÁS EJEMPLOS DE COMPARACIONES

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

EJEMPLO: Obtener de los maillots el tipo y el premio en dólares (supongamos que está en pesetas) (\$1 = 150 ptas.) de aquellos maillots cuyo premio supere los 100 dólares.

```
SELECT tipo, premio / 150 FROM Maillot
```

```
WHERE premio / 150 > 100;
```

Uso de LIKE

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 nombreq LIKE '%100\%%' ESCAPE '\';
```

CONSULTAS DE VALORES AGREGADOS

La sintaxis de una referencia a una función agregada es la siguiente:

{ 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.
- distinct indica que los valores redundantes sean eliminados antes de que se realice el cálculo correspondiente.
- La función especial count(*), en la que no está permitido incluir distinct ni all, da como resultado el cardinal del conjunto 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';
```

CONSULTAS SIMPLES SOBRE VARIAS TABLAS

Cuando la información que se desea obtener de la base de datos se encuentra almacenada en más de una tabla se hace indispensable el declarar una consulta que manipule estas tablas.

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

```
select etapa.netapa, nompuerto
from Etapa, Puerto
where etapa.dorsal = puerto.dorsal;
```

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. En general,

```
[tabla | variable_recorrido].columna
```

Las variables de recorrido permiten dar un nombre alternativo a la *misma* tabla dentro de una consulta. La manera de declarar una variable de recorrido es:

```
from tabla [as] variable_recorrido
```

USO DE CLAVES AJENAS EN CONSULTAS DE 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.

Si existen claves ajenas, lo más normal es una igualdad entre la clave ajena y los atributos correspondientes de la tabla a la que se hace referencia.

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';
```

EJEMPLO: Obtener pares nombre de ciclista, 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;
```

CONSULTAS COMPLEJAS: SUBCONSULTAS

Si la información que se está buscando está incluida en una tabla y la condición de búsqueda de esta información requiere acceder a otras tablas, entonces (EN ALGUNOS CASOS) se pueden utilizar las subconsultas para expresar este tipo de condiciones.

EJEMPLO: El mismo ejemplo anterior: Obtener los nombres de los ciclistas pertenecientes al equipo dirigido por 'Alvaro Pino'. Teníamos 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');
```

Esto es posible porque la información que se requiere, nombre del ciclista, no está en la tabla de la subconsulta (Equipo) y porque la subconsulta retorna un único valor.

PREDICADOS QUE ACEPTAN SUBCONSULTAS:

Las subconsultas pueden aparecer en las condiciones de búsqueda, tanto de la cláusula `where` como `having`, como argumentos de algunos predicados.

Los predicados que pueden llevar como argumentos subconsultas son los siguientes:

- predicados de comparación (`=`, `<>`, `>`, `<`, `>=`, `<=`).
- `in`: comprueba que un valor *pertenece* a una colección dada mediante una subconsulta.
- `match`: comprueba si un valor es idéntico a algún valor de una colección.
- predicados de comparación cuantificados (`any` y `all`): permitir comparar un valor con un conjunto de valores.
- `exists`: equivalente al cuantificador existencial, comprueba si una subconsulta devuelve alguna fila.
- `unique`: devuelve cierto si la consulta no tiene filas repetidas.

PREDICADOS DE COMPARACIÓN (=, <>, >, <, >=, <=)

Cada uno de los dos lados de un predicado de comparación debe ser una única tupla formada por el mismo número de columnas. Es decir:

$$(A1, A2, \dots, An) \text{ predicado_comparación } (B1, B2, \dots, Bn)$$

Las subconsultas pueden ser argumentos siempre y cuando devuelvan una única fila y el número de columnas coincida en número y tipo con el otro lado del predicado de comparación. Llamaremos *constructor_fila* a una lista de atributos entre paréntesis o una subconsulta.

$$\textit{constructor_fila} \text{ predicado_comparación } \textit{constructor_fila}$$

- En el caso de que la subconsulta esté vacía, se convierte a una fila con valores nulos en todas las columnas.
- Para poder comparar dos *constructor_fila* de más de una columna, existe una forma definida de realizar esta comparación para cada uno de (=, <>, >, <, >=, <=).
- En general, sin embargo, se verán subconsultas de una única columna, como el ejemplo anterior.

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 categoría = 2 );
```

INCORRECTO: (error de ejecución):

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

Predicado in

constructor_fila [not] in (expresión_tabla)

A la derecha de IN puede aparecer más de una fila y por eso se denomina *expresión_tabla*.

EJEMPLO: Obtener el n° 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 netapa FROM Etapa
WHERE dorsal IN (SELECT dorsal FROM Ciclista
WHERE nomeq IN (SELECT nomeq FROM Equipo
WHERE director LIKE 'A%')));
```

Predicados de comparación cuantificados (any y all)

constructor_fila *predicado_comparación* {all | any | some}
(*expresión_tabla*)

- El predicado de comparación cuantificado con ALL se evalúa a cierto si lo es para todas las filas de la expresión de tabla (si la tabla está vacía también se evalúa a cierto).
- El predicado de comparación cuantificado con ANY o SOME se evalúa a cierto si lo es para alguna fila de la expresión de tabla (si la tabla está vacía se evalúa a falso).

(*) el predicado in es idéntico al predicado de comparación cuantificado =any.

EJEMPLO 8) Obtener el nombre de los puertos y de los ciclistas que los hayan ganado que tengan la mayor pendiente.

```
SELECT P.nompuerto, C.nombre FROM Puerto P, Ciclista C
WHERE P.dorsal = C.dorsal AND
P.pendiente >= ALL (SELECT P1.pendiente FROM Puerto P1 );
```

EJEMPLO 9) Obtener el nombre de los puertos y de los ciclistas que los hayan ganado, cumpliendo que el puerto no sea el que tenga la menor pendiente.

```
SELECT P.nompuerto, C.nombre FROM Puerto P, Ciclista C
WHERE P.dorsal = C.dorsal AND
P.pendiente > ANY (SELECT P1.pendiente FROM Puerto P1 );
```

(*) Cualquier ANY se puede convertir en un ALL cambiando la condición a su condición negada y añadiendo un NOT.

Predicado exists

exists (expresión_tabla)

El predicado exists se evalúa a cierto si la expresión select devuelve al menos una fila.

EJEMPLO: Obtener el nombre de aquellos ciclistas que han llevado un maillot de un premio menor de 50000 ptas. .

```
SELECT C.nombre FROM Ciclista C, Llevar L
WHERE C.dorsal = L.dorsal AND
EXISTS (SELECT *
        FROM Maillot M
        WHERE M.premio < 50000 AND M.codigo = L.codigo);
```

(*) En el ejemplo se ve además una referencia externa desde la subconsulta a la tabla C externa a él, lo que suele ser muy habitual (aunque no exclusivo) a las subconsultas con EXISTS.

(*) En general, IN y EXISTS son intercambiables y se pueden eliminar haciendo consultas a múltiples tablas e igualando por claves ajenas.

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

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

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

```
WHERE EXISTS (SELECT * FROM ...)
```

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

```
WHERE NOT EXISTS (SELECT * FROM ...)
```

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

Uso de EXISTS para cuantificaciones universales (NO HAY EN SQL):

EJEMPLO: Obtener el nombre del ciclista (si lo hay) 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 );
```

CQC: Se ha tenido que convertir en: “Obtener el nombre del ciclista tal que *no* existe una etapa de más de 200 km. que él *no* haya ganado”

Equivalencias fórmulas lógicas generales (CRT) y SQL:

Las expresiones lógicas generales (p.ej. CRT) se pueden convertir a SQL de la siguiente manera:

Exp. Lógica		Paso intermedio	SQL
$p \rightarrow q$	\Rightarrow	$\neg p \vee q$	(NOT p) OR q
$\forall x p$	\Rightarrow	$\neg \exists x \neg p$	NOT EXISTS (...negar....)
$\forall x (p \rightarrow q)$	\Rightarrow	$\neg \exists x \neg (p \rightarrow q) \equiv \neg \exists x \neg (\neg p \vee q)$ $\equiv \neg \exists x (p \wedge \neg q)$	

“Coletillas” en consultas con cuantificación universal:

EJEMPLO: Obtener el nombre del ciclista (si lo hay) que ha ganado todas las etapas de más de 200 km.

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

¿Qué pasa si no hay etapas de más de 200 km?

!!!SALDRÍAN TODOS LOS CICLISTAS!!!

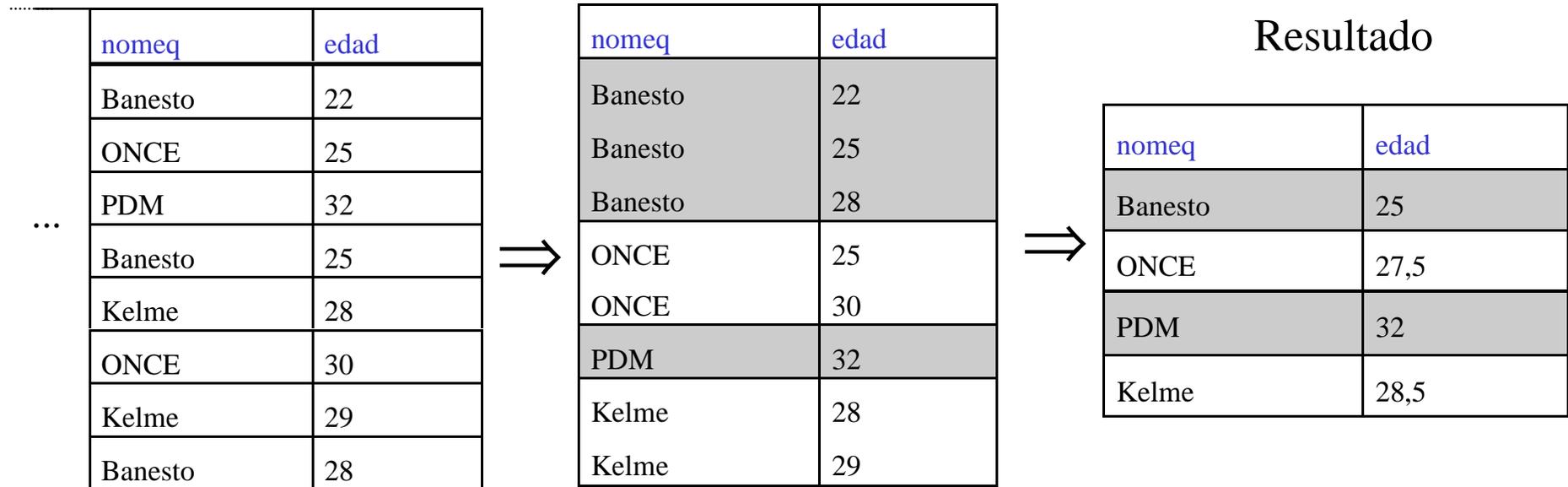
Solución:

```
SELECT C.nombre FROM Ciclista C
WHERE NOT EXISTS (SELECT * FROM Etapa E
                  WHERE E.km > 200 AND C.dorsal <> E.dorsal )
AND EXISTS (SELECT * FROM ETAPA E2 WHERE E2.km > 200);
```

CONSULTAS COMPLEJAS: AGRUPACIÓN

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;
```



Relación Selección-Agrupamiento

Un grupo se puede entender como un conjunto de filas con el mismo valor para el conjunto de columnas por las que se agrupa (las incluidas en la cláusula group by).

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

De una consulta agrupada sólo se puede realizar una selección que devuelve un valor, simple o compuesto, por cada grupo formado en la consulta.

EJEMPLO INCORRECTO:

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

La regla sintáctica que aplican los sistemas relacionales para asegurar el buen funcionamiento de las consultas agrupadas es la siguiente:

“en la selección de una consulta agrupada, sólo pueden aparecer referencias a columnas por las cuales se agrupa, referencias a funciones agregadas o literales”.

GROUP y WHERE

Si se incluye la cláusula where la aplicación de esta cláusula se produce previamente a la agrupación.

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

GROUP, WHERE y HAVING

La cláusula HAVING, que sólo puede ir en consultas agrupadas, es similar a WHERE pero el orden es el siguiente:

- 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 referencias a columnas por las cuales se agrupan o a funciones agregadas.

EJEMPLO: 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;
```

EJEMPLO: 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 siempre por CP */
HAVING AVG (P.pendiente) >10;
```

COMBINACIONES DE TABLAS

Existen otras formas de combinar varias tablas en consultas y todas ellas, junto con las ya vistas, dan lugar a una “expresión de tabla”.

Existen, en definitiva, varias formas de combinar dos tablas en el lenguaje SQL:

- Incluir varias tablas en la cláusula from.
- Uso de subconsultas en las condiciones de las cláusulas where o having.
- **Combinaciones conjuntistas de tablas:** utilizan para combinar las tablas operadores de la teoría de conjuntos.
- **Concatenaciones de tablas:** combinan dos tablas utilizando diferentes formas variantes del operador concatenación del Álgebra Relacional.

COMBINACIONES CONJUNTISTAS DE TABLAS

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

- UNION
- EXCEPT
- 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 los ciclistas de 'Banesto' y de la 'ONCE'.

(SELECT nombre FROM Ciclista WHERE nomeq = 'Banesto')

UNION

(SELECT nombre FROM Ciclista WHERE nomeq = 'ONCE')

CONCATENACIONES DE TABLAS

Corresponden a variantes del operador concatenación del Álgebra Relacional.

- Producto cartesiano (cross join)
- Concatenación interna

referencia_tabla [natural] [inner] join *referencia_tabla*

[on *expresión_condicional* | using (*comalista_columna*)]

- Concatenación externa

referencia_tabla [natural]

{ left [outer] |

right [outer] |

full [outer] } JOIN *referencia_tabla*

[on *expresión_condicional* | using (*comalista_columna*)]

- Concatenación unión

... FROM T1 UNION JOIN T2 ≡ ... FROM

$$\left[\begin{array}{l} \text{select t1.*, null, null, ..., null from t1} \\ \text{union all} \\ \text{select null, null, ..., null, t2.* from t2} \end{array} \right]_{32}$$

ÁLGEBRA RELACIONAL -- SQL

A.R.	SQL ESTÁNDAR	SQL de ORACLE'8
• \cup	• UNION	• UNION
• $-$	• EXCEPT	• MINUS
• \cap	• INTERSECT	• INTERSECT
• \times	• CROSS JOIN	• (no hace falta, poner una coma)
• $ >< $	• NATURAL JOIN	• (no está, bastan = 's en el WHERE)
Otros		
• \cup (dups.)	• UNION ALL	• UNION ALL
• $><$	• Left/Right/Full JOIN	• WHERE TX.a1(+) = TY.a2 (eq. right join)
	• UNION JOIN	• WHERE TX.a1 = TY.a2 ₃₃ ⁽⁺⁾ (eq. left join)

INTRODUCCIÓN DE INFORMACIÓN: INSTRUCCIÓN INSERT

```
insert into tabla [(comalista_columna)]
```

```
{ default values | values (comalista_átomos) | expresión_tabla}
```

- Si no se incluye la lista de columnas se deberán insertar filas completas de *tabla*.
- Si se incluye la opción default values se insertará una única fila en la tabla con los valores por defecto apropiados en cada columna (según la definición de *tabla*).
- En la opción values(*comalista_átomos*) los átomos vienen dados por expresiones escalares.
- En la opción *expresión_tabla*, se insertarán las filas resultantes de la ejecución de la expresión (SELECT).

Ejemplo: Añadir un ciclista de dorsal 101, nombre 'Joan Peris', edad 20 años y del equipo 'Kelme'.

```
insert into Ciclista
```

```
values (101, 'Joan Peris', 20, 'Kelme');
```

MODIFICACIÓN DE LA INFORMACIÓN: INSTRUCCIÓN UPDATE

update tabla

set *comalista_asignaciones*

[where *expresión_condicional*]

donde una *asignación* es de la forma:

columna = { default | null | *expresión_escalár* }

Si se incluye la cláusula where sólo se aplicará a las filas que hagan cierta la condición.

EJEMPLO: Incrementar un 10% la pendiente del puerto ‘Aitana’ al haberse cerrado la carretera que había en buen estado y ser necesario subir por otra peor.

```
UPDATE Puerto SET pendiente = pendiente * 1.10
```

```
WHERE nompuerto = ‘Aitana’ ;
```

ELIMINACIÓN DE INFORMACIÓN: INSTRUCCIÓN DELETE

delete from tabla [where *expresión_condicional*]

Si se incluye la cláusula where se eliminarán aquellas que hagan cierta la condición.

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

DELETE FROM Ciclista WHERE nombre = 'M. Indurain';

MÁS EJEMPLOS DEL ESQUEMA 'CICLISMO'

1) 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 );
```

MÁS EJEMPLOS DEL ESQUEMA 'CICLISMO'

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

```
SELECT e.salida, e.llegada
```

```
FROM etapa e, puerto p
```

```
WHERE e.netapa=p.netapa AND
```

```
p.pendiente=(select MAX(pendiente) from puerto );
```

MÁS EJEMPLOS DEL ESQUEMA 'CICLISMO'

3) ¿Quién es el ciclista más joven? .

```
SELECT nombre
```

```
FROM ciclista c
```

```
WHERE edad = ( SELECT MIN(edad) FROM ciclista );
```

4) Obtener el nombre de los ciclistas que han ganado todos los puertos de una etapa y además han ganado esa misma etapa. (Ej. 17 boletín)

```
SELECT 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.dorsal=c.dorsal
            AND p.netapa=e.netapa );
```

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 c.dorsal=l.dorsal AND m.codigo=l.codigo
AND NOT EXISTS( SELECT * FROM llevar l2, ciclista c2
                WHERE c2.dorsal=l2.dorsal AND
                c2.nombre<>c.nombre AND l2.codigo=l.codigo);
```

6) Obtener el nombre de los ciclistas que pertenezcan a un equipo que tenga más de cinco corredores indicando el número de etapas ganadas por cada uno.

```
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;
```

7) Obtener el nombre de los equipos que tengan la media de edad máxima de todos los equipos.

```
SELECT C.nomeq, AVG(C.edad)
FROM ciclista C
GROUP BY C.nomeq
HAVING AVG(C.edad) >= ALL
                                (SELECT AVG(D.edad)
                                FROM ciclista D
                                GROUP BY D.nomeq);
```

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

```
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) );
```