1.- Introducción

Una de las grandes ventajas de las BDs es la de facilitar la realización de consultas de una manera rápida, flexible y eficaz.

Para la realización de consultas, inserciones, modificaciones y borrados sobre la BD utilizaremos el lenguage SQL (Structured Query Language), concretamente sus sentencias DML (Data Manipulation Language).

Cada comando SQL se compone de clásulas.

Algunas convenciones que se utilizan a la hora de describir la sintaxis de las sentencias SQL son:

- Utilizar “[ ]” para indicar opcionalidad.
- Utilizar “{ }” para indicar una o varias opciones entre las que debe escogerse.
- “|” como separador entre opciones.

2.- Inserción de registros

Para insertar registros se utiliza la sentencia INSERT. La estructura básica de esta sentencia es:

```
INSERT INTO table_name (lista_columnas) VALUES (valores_para_columnas);
```

Una versión reducida de su sintaxis puede ser:

```
INSERT [INTO] table_name
    [(col_name1, col_name2, ...)]
    VALUES (value_col1, value_col2, ...), (...), (...), ...
```

También podemos hacer un INSERT utilizando los valores devueltos por una sentencia SELECT:

```
INSERT [INTO] table_name [(lista_columnas)] SELECT (columnas) FROM ...
```

3.- Modificación de registros

Para actualizar la información de 1 o múltiples registros disponemos de la sentencia UPDATE. Su estructura básica es:

```
UPDATE table_name SET col_name = expresion, col_name = expresion... WHERE where_condition
```

Por ejemplo:

```
UPDATE jugadores SET salario=salario+1000 WHERE equipo = 5
```

También se puede utilizar una consulta SELECT para realizar un UPDATE:

```
UPDATE jugadores SET equipo_id=(select id from equipo where nombre = 'Obradoiro') where id_jugador = 3;
```

4.- Eliminación de registros

```
DELETE FROM table_name
```

```
[WHERE where_condition]
[ORDER BY cols…]
[LIMIT row_count]
```

```
DELETE FROM jugadores WHERE equipo_id = 2;
```

Si no usamos el filtro WHERE se borran todos los registros de la tabla indicada, al igual que si realizáramos un TRUNCATE TABLE table_name.

Igual que en las actualizaciones, podemos usar subconsultas en la cláusula WHERE para filtrar los borrados:

```
DELETE FROM jugadores WHERE equipo = (SELECT id FROM equipo WHERE nombre = 'Obradoiro')
```

5.- Consultas

Para realizar consultas en la BD que deben devolver registros, utilizamos la sentencia SELECT. Una versión simplificada de la sentencia SELECT es:

```
SELECT [DISTINCT] [lista_columnas | * | expr_select] FROM tabla1, tabla2…
[WHERE where_conditions]
[GROUP BY cols [ASC | DESC]]
[HAVING having_conditions]
[ORDER BY [col1 [ASC | DESC]]  [, col2 [ASC | DESC]]  [,…]
[LIMIT row_count]
```

El significado de las distintas cláusulas es:

- DISTINCT: para mostrar sólo valores distintos.
- *: indica que se desea recuperar todas las columnas.
- expr_select: es una expresión que basándose en el valor devuelto de una o más cols. realiza algún cálculo o operación de transformación y devuelve el resultado.
- FROM: indica la tabla o tablas que hay que utilizar para obtener los datos.
- WHERE: permite incluir condiciones sobre nuestra consulta para filtrar las filas que queremos obtener.
- GROUP BY: permite agrupar los registros devueltos según uno o varios campos.
- HAVING: permite añadir condiciones sobre agrupaciones de registros.
- ORDER BY: ordena los datos devueltos según uno o varios campos.
- ASC / DESC: tipo de orden: ascendente o descendente.
- LIMIT: indica el número máximo de tuplas que queremos que nos devuelva la consulta.

5.1.- Consultas básicas

La consulta más sencilla que podemos hacer es una consulta que devuelve todos los campos de una tabla:

```
SELECT *  FROM equipos
```

Si deseamos sólo seleccionar 2 columnas escribimos:

```
SELECT nombre,apellido FROM jugadores
```

Podemos crear “alias” tanto para las columnas como para las tablas:

```
SELECT j.nombre AS 'nombre jugador' FROM jugadores AS j WHERE j.equipo= 1;
```

Como veremos a medida que nuestras consultas se vuelven más complejas y largas, los alias serán muy útiles. (La cláusula “AS” se puede omitir. Si el alias contiene espacios en blanco, tiene que ir entre comillas).

5.1.1- ORDER BY

Permite especificar el orden en el que deseamos que nos sean devueltas las filas. Si no se especifica, las filas se devuelven sin ordenar.

Se puede especificar más de una columna para realizar la ordenación y, para cada una de ellas se puede añadir ASC o DESC en función de si queremos una ordenación ascendente o descendente. (Si se omite esta cláusula, el orden por defecto suele ser ascendente).

```
SELECT nombre, apellido, equipo FROM jugadores ORDER BY equipo;
```

Podemos especificar varias columnas para la ordenación:

```
SELECT nombre, apellido, equipo FROM jugadores ORDER BY equipo, nombre;
```

Y podemos especificar un tipo de orden para cada columna por la que ordenamos:

```
SELECT nombre, apellido, equipo FROM jugadores ORDER BY equipo, nombre desc;
```

5.1.2.- DISTINCT

Una consulta SELECT por defecto devuelve todas las filas que cumplan los criterios indicados en WHERE.
Si queremos evitar las filas duplicadas podemos utilizar la cláusula DISTINCT.

```
SELECT equipo FROM jugadores;  # Devuelve filas repetidas

SELECT DISTINCT equipo FROM jugadores;   # Devuelve filas diferentes
```

5.1.3.- LIMIT

Permite restingir el número de filas devueltas en una consulta.

```
SELECT * FROM jugadores LIMIT 5;
```

También podemos expresar un rango, por ejemplo, para devolver 6 filas desde la cuarta fila.

```
SELECT * FROM jugadores LIMIT 3,6;
```

5.1.4.- Expresiones

En una consulta se pueden realizar operaciones con los datos, por ejemplo, se puede solicitar el producto de 2 columnas. Para realizar este tipo de operaciones se utilizan las “expresiones” que permiten combinar valores literales con las columnas y con operadores de varios tipos.

También en las condiciones de la cláusula WHERE se pueden utilizar las expresiones.
Una expresión es una combinación de operadores, operandos y paréntesis cuyo resultado es un único valor.

Los operandos pueden ser nombre de columnas, constantes u otras expresiones.
Los operadores actúan sobre datos homogéneos, numéricos o alfanuméricos.

Aritméticos	+	Suma
	-	Resta
	*	Producto
	/	División
	** ó ^	Potencia
Relacionales o de comparación	<	Menor
	<=	Menor o igual
	>	Mayor
	>=	Mayor o igual
	=	Igual
	<> !=	Distinto
	!<	No menor
	!>	No mayor
Lógicos	AND	Devuelven verdadero o falso
	NOT	
	OR	
	XOR	

5.1.5.- Funciones

Las funciones aportan funcionalidades avanzadas a las consultas.
Algunas de las funciones más útiles son:

Funciones de control de flujo:

- CASE: permite devolver diferentes valores en función del resultado de una comparación:

```
SELECT nombre, apellido, altura,
    CASE
        WHEN altura > 2 THEN 'PIVOT'
        WHEN altura < 1.90 THEN 'BASE'
        ELSE ''
    END
FROM jugadores;
```

- IF(condition, true_value, false_value): devuelve un valor si la condición es verdadera u otro valor si es falsa.

- IFNULL(expr, alt_value): si la expresión es null devuelve el valor especificado, si no lo es, devuelve la expresión.

- ISNULL(value): devuelve 1 ó 0 si el valor es nulo.

Funciones sobre cadenas:

- CONCAT(val1, val2,…): concatena 2 o más valores:

- LEFT(cadena, num_chars): devuelve *num_chars* caracteres de la cadena comenzando por la izquierda.

- LENGTH(cadena_texto): devuelve el tamaño de una cadena de texto.

- LOCATE(cadena_busq, cadena): devuelve la posición de la primera ocurrencia de *cadena_busq* en cadena. Realiza una búsqueda no-sensitiva. Si no se encuentra la cadena, devuelve 0.

- LOWER(cadena_texto): convierte una cadena a minúsculas.

- LTRIM(cadena_texto): elimina los espacios en blanco de la cadena por la izquierda.

- MID(cadena, inicio, longitud): extrae una cadena de otra cadena.

- REPLACE(cadena_texto, antes, ahora): reemplaza todas las ocurrencias de *antes* con *ahora* en *cadena_texto*.

- RIGHT(cadena_texto, num_chars): devuelve num_chars caracteres de cadena_texto por la derecha.

- RTRIM(cadena_texto): elimina los espacios en blanco de una cadena por la derecha.

- SUBSTR(cadena_texto, inicio, fin): extrae una cadena de texto de otra cadena. SUBSTR, SUBSTRING y MID son sinónimos.

- TRIM(cadena_texto): elimina los espacios en blanco al principio y al final de la cadena.

- UPPER(cadena_texto): convierte una cadena a mayúsculas.

Funciones numéricas:

- ABS(número): devuelve el valor absoluto.

- COUNT(expresión): devuelve el número de filas devueltas por una consulta. No cuenta los valores nulos.

- MAX(expresión): devuelve el mayor valor del conjunto.

- MIN(expresión): devuelve el mínimo valor del conjunto.

- SUM(expresión): devuelve la suma de la expresión.

- TRUNCATE(número, decimales): trunca el número al número especificado de decimales.

Funciones de fecha:

- ADDDATE(fecha, INTERVAL cantidad unidad) ó ADDDATE(fecha, dias): añade una cantidad de tiempo a una fecha.

"unidad" puede ser: DAY, WEEK, MONTH, YEAR, HOUR, MINUTE, SECOND…

- CURRENT_DATE(): devuelve la fecha actual.

- CURRENT_TIME(): devuelve la hora actual

- DATE(fecha_y_hora): devuelve sólo la fecha (sin hora) de una expresión

- DATE_FORMAT(fecha, patrón): formatea una fecha según el patrón especificado.

Algunos patrones comunes:

Patrón	Descripción
%d	Día del mes en número
%H	Hora en formato (0-23)
%i	Minutos (0-59)
%s	Segundos (0-59)
%Y	Año (4 dígitos)
%y	Año (2 dígitos)
...	

- DAY(fecha): extrae el día en formato numérico.

- HOUR(fecha): extrae la hora en formato numérico.

- LAST_DAY(fecha): devuelve el último día del mes.

- MONTH(fecha): extrae el mes en formato numérico.

- NOW(): devuelve la fecha y hora actuales.

- SUBDATE(fecha, INTERVAL cantidad unidad) ó SUBDATE(fecha, días): igual que ADDDATE pero para restar una cantidad de tiempo.

- TIME(fecha_y_hora): devuelve la hora (sin la fecha).

-TIMEDIFF(fecha1, fecha2): devuelve la diferencia entre 2 fechas que deben estar en el mismo formato.

- YEAR(fecha): extrae el año.

5.1.6.- Cláusula WHERE

Podemos establecer condiciones de búsqueda en la sentencia SELECT utilizando la cláusula WHERE. Estas condiciones se denominan **predicados**.
Un predicado expresa una condición que se cumple o no sobre un conjunto de dos valores o expresiones y el resultado de su evaluación puede ser verdadero, falso o desconocido.

Si alguno de los operadores, o ambos, son nulos, el resultado de la evaluación del predicado es desconocido.

Cuando añadimos un predicado a la cláusula WHERE de un SELECT, sólo se recuperan las filas para las que el predicado es verdadero y se descartan las filas para las que el predicado es falso o desconocido.

Predicados de comparación: expresan condiciones de comparación entre 2 valores utilizando operadores lógicos.

Por ejemplo:

x = y x es igual a y
x <> y ó x != y x no es igual a y
x < y x es menor que y
x > y x es mayor que y
x >= y x es mayor o igual que y
x <= y x es menor o igual que y

Predicado NULL: sirve para consultar si el valor de una columna en una fila determinada es nulo o no. Si es nulo devuelve verdadero.

nom_columna IS [NOT] NULL

SELECT * FROM equipos WHERE web IS NULL

Predicado IN: pertenencia a un conjunto. Sirve para comprobar si el resultado de la evaluación de una expresión está incluido en la lista de valores especificados tras la palabra IN.

expresion [NOT] IN (valor1, valor2,…)

SELECT * FROM equipos WHERE ciudad NOT IN (‘Valencia’, ‘Madrid’)

La lista de valores podemos obtenerla también mediante una subconsulta:

expresion [NOT] IN (subconsulta)

Predicado [NOT] BETWEEN exp1 AND exp2: comprueba si un valor está incluido entre otros dos (ambos inclusive) o no.

SELECT * FROM partidos WHERE fecha BETWEEN '2011-11-01' AND '2011-11-31'

Predicado LIKE: sirve para realizar búsquedas por cadenas de texto parciales utilizando patrones.

nom_columna [NOT] LIKE literal_alfanumérico

El literal alfanumérico debe ir entre comillas y puede incluir cualquier carácter pero hay 2 con un significado especial:

- '_': es un comodín, sirve "cualquier carácter" en esta posición (pero sólo 1 carácter).
- '%': es un comodín que indica que sirve cualquier cadena de texto.

Si el literal_alfanumérico no contiene ningún '_' ó '%', el predicado A LIKE B es equivalente a A = B.

Predicado REGEXP/RLIKE: expresiones regulares.

nom_columna [NOT] REGEXP cte_alfanumérica

Es equivalente a LIKE pero es más potente porque permite usar expresiones regulares.

Una expresión regular es una cadena formada por caracteres literales y otros con una función especial. Por ejemplo si utilizamos la exp. Regular "Hola|Hello" hace que la búsqueda coincida si el valor es "Hola" ó "Hello".

Otro ejemplo puede ser "B[an]*s" que podemos utilizarla para "hacer match" contra "Bananas", "Baaaas", "Bs" o cualquier otra cadena que empiece con "B", termine con "s" y contenga cualquier número de "a" ó "n" entre la "B" y la "s".

Algunos caracteres especiales son:

- ^: la coincidencia debe producirse al principio de la cadena:
  - SELECT * FROM jugadores WHERE nombre REGEXP "^A"
- $: la coincidencia debe producirse al final de la cadena:
  - SELECT * FROM jugadores WHERE nombre REGEXP "A$"
- .: coincidencia con cualquier carácter:
  - SELECT * FROM jugadores WHERE nombre REGEXP "j.a"
- ?: opcionalidad. Coincidencia con 0 ó 1 ocurrencias del carácter precedente:
  - SELECT * FROM jugadores WHERE nombre REGEXP "^R?a"
- *: cero o más repeticiones del carácter precedente:
  - SELECT * FROM jugadores WHERE nombre REGEXP "ca*"
- +: 1 o más repeticiones del carácter precendente:
  - SELECT * FROM jugadores WHERE nombre REGEXP "ca+"
- |: coincidencia de una entre varias secuencias posibles:
  - SELECT * FROM jugadores WHERE nombre REGEXP '(Vic|Ju)'
- {n} ó {n,m}: n repeticiones o entre n y m repeticiones del carácter precedente:
  - SELECT * FROM jugadores WHERE apellido REGEXP 't{2}'
- [a-z], [^a-z]: cualquier carácter en el rango:
  - SELECT * FROM jugadores WHERE nombre REGEXP '[0-9]'
  - Que no contengan vocales:
  - SELECT * FROM jugadores WHERE nombre REGEXP '^[^aeiou].*$'

A veces ocurre que alguno de estos caracteres especiales forman parte de la cadena que queremos buscar, en ese caso se dice que hay que "escapar" ese carácter y para ello se le antepone '\\'.

Por ejemplo, para seleccionar los equipos cuya web sea del tipo “http://www.dominio.extensión”, siendo la extensión de 3 caracteres:

```
SELECT * FROM equipos WHERE web REGEXP '^http://www\\.[^\\.]+\\..{3}';
```

Predicados compuestos: son combinaciones de predicados simples utilizando los operadores lógicos AND, OR, XOR y NOT.

5.1.7.- Funciones de agrupación

Son funciones que devuelven un único valor como resultado de aplicar una determinada operación a los valores contenidos en una columna.

Algunas de las más útiles son:

- AVG: devuelve la media de los valores.
- MAX: devuelve el valor máximo.
- MIN: devuelve el valor mínimo.
- SUM: devuelve la suma.
- COUNT: devuelve el número de filas.

Si la colección de valores para agregar es vacía, COUNT devuelve 0 y el resto de funciones devuelven null.

AVG y SUM sólo admiten argumentos numéricos. MAX y MIN pueden recibir argumentos de cualquier tipo.

AVG, MAX, MIN y SUM tienen un resultado del mismo tipo que el argumento. (Si hacemos max(enteros) obtenemos un resultado entero, si hacemos max(fecha) obtenemos un resultado de tipo fecha).

Por ejemplo:

- Calcular el número de jugadores que miden más de 2m:

```
SELECT COUNT(id_jugador) FROM jugadores WHERE altura > 2;
```

- Calcular el salario medio de todos los jugadores:

```
SELECT AVG(salario) as 'Salario medio' FROM jugadores;
```

- Encontrar el salario más alto, el más bajo y la diferencia entre ambos:

```
SELECT MAX(salario) 'máximo', MIN(salario) 'mínimo', MAX(salario) - MIN(salario) 'diferencia' FROM jugadores;
```

- Halar el número de ciudades en las que hay equipos registrados:

```
SELECT COUNT( DISTINCT ciudad) FROM equipos;
```

- Obtener el importe de los salarios mensuales:

```
SELECT SUM(salario)/12 'Salario mensual' FROM jugadores;
```

Antes de aplicar las funciones, se construyen 1 o más grupos de filas. Los grupos se realizan según la cláusula GROUP BY. Si no se emplea el GROUP BY se construye 1 sólo grupo que incluye todas las filas que devuelve la consulta y tras aplicar la función de agregación, se devolverá una única fila.

5.1.8.- GROUP BY

Cuando utilizamos una función de agrupación podemos formar grupos de filas según un determinado criterio utilizando la cláusula GROUP BY de manera que la función se aplique para cada grupo.

La cláusula GROUP BY siempre va después de la cláusula WHERE (si existe).

```
GROUP BY col1, col2…
```

Donde col1 y col2 son las: **columnas de agrupamiento**.

La clásula GROUP BY forma grupos de manera que todas las filas de cada grupo tienen valores idénticos para las columnas de agrupamiento de manera que para cada grupo se obtiene 1 fila en el resultado de la consulta.

En una consulta que utiliza GROUP BY, todas las columnas seleccionadas en el SELECT deben estar incluidas en una función de agrupamiento o estar entre las columnas de agrupamiento de la cláusula GROUP BY, de otra manera la consulta será errónea.

- Seleccionar el número de jugadores por equipo:

```
SELECT equipo, count(*) FROM jugadores GROUP BY equipo;
```

- Seleccionar el salario mínimo y máximo de los jugadores por equipo:

```
SELECT equipo, min(salario), max(salario) FROM jugadores group by equipo;
```

- Consulta NO válida:

```
SELECT equipo, apellido, min(salario), max(salario) FROM jugadores group by equipo;
```

apellido no está dentro de ninguna función de agrupación ni tampoco está entre las columnas de agrupamiento.

5.1.9.- HAVING

Sirve para filtrar los resultados de una función de agrupamiento.

```
HAVING condición
```

Después de haber obtenido la fila resultado para cada grupo realizado durante la agrupación, se descartan aquellas que no cumplan la condición impuesta por HAVING.

- Seleccionar los equipos que tengan una media de salario mayor que 70.000:

```
SELECT equipo, AVG(salario) FROM jugadores group by equipo HAVING AVG(salario) > 70000;
```

También podemos utilizar los alias:

SELECT equipo, AVG(salario) as salario_medio FROM jugadores group by equipo HAVING salario_medio > 70000;

5.2.- Subconsultas

En algunas ocasiones puede suceder que no conocemos de antemano el valor de una condición en un predicado de la cláusula WHERE o HAVING ya que depende del valor de otra consulta. Por ejemplo, si queremos saber qué jugadores cobran más que Gasol necesitamos también una sonculta para hallar el salario de Gasol. En estos casos se utilizan las consultas subordinadas o subconsultas. Nuestro ejemplo quedaría así:

SELECT * FROM jugadores WHERE salario > (SELECT salario FROM jugadores WHERE apellido = ‘Gasol’);

El ‘select’ entre paréntesis es una subconsulta.
Una subconsulta puede contener a su vez otra subconsulta. A las consultas subordinadas también se les llama **anidadas** y puede haber múltiples niveles de anidamiento.

Las subconsultas se pueden utilizar con:

- Predicados básicos de comparación.
- Predicados cuantificados (ANY|SOME, ALL)
- Predicados EXISTS
- Predicado IN

5.2.1.- Cuantificador ALL

El predicado cuantificado es verdadero si la comparación es verdadera para todos y cada uno de los valores devueltos por la subconsulta. Por ejemplo: obtener el nombre de los jugadores que ganan más que todos los del equipo 2.

SELECT nombre FROM jugadores WHERE salario > ALL (SELECT salario FROM jugadores WHERE equipo = 2);

5.2.2.- Cuantificador ANY o SOME

El predicado cuantificado es verdadero si la comparación es verdadera para uno cualquiera de los valores devueltos por la subconsulta. Por ejemplo: seleccionar los jugadores que ganan más que alguno del equipo 5.

SELECT nombre FROM jugadores WHERE salario > ANY (SELECT salario FROM jugadores WHERE equipo = 5);

5.2.3.- Predicado IN

Podemos utilizar una subconsulta que devuelva un conjunto de valores para utilizarlos en un predicado IN.
Por ejemplo: obtener los jugadores que juegan en algún equipo de Zaragoza.

SELECT * FROM jugadores WHERE equipo IN (SELECT id_equipo FROM equipos WHERE ciudad = ‘Zaragoza’);

5.2.4.- Predicado EXISTS

Devuelve ‘verdadero’ si la subconsulta subsiguiente es no vacía y falso en caso contrario. Por ejemplo: obtener los datos de los jugadores pero sólo si hay más de 10 equipos.

```
SELECT * FROM jugadores WHERE EXISTS (SELECT count(*) FROM equipos HAVING count(*) > 10);
```

5.3.- Consultas correlacionadas

Las subconsultas que hemos visto hasta ahora la sentencia subordinada puede evaluarse independientemente de sus consultas antecedentes. En las consultas correlacionadas, las subconsultas hacen referencia a alguna columna de las tablas mencionadas en la cláusula FROM de alguna de sus consultas antecedentes.

Por ejemplo: obtener los datos de jugadores que miden más que la media del equipo.

```
SELECT * FROM jugadores j1 WHERE altura > (SELECT AVG(altura) FROM jugadores j2 WHERE j2.equipo = j1.equipo);
```

Observa la utilización de alias para identificar las columnas correctamente.

Otro ejemplo: obtener los datos de equipos con más de 5 jugadores:

```
SELECT * FROM equipo e WHERE 5 < (SELECT count(*) FROM jugadores j WHERE j.equipo = e.id_equipo);
```

5.3.1.- Predicado EXISTS en consultas correlacionadas

Se usa para verificar cuando un valor de la consulta antecedente existe en los valores devueltos por la subconsulta.

Obtener los datos de los equipos que tengan algún jugador que supere los 2m de altura:

```
SELECT * FROM equipos e where exists (
        select id_jugador from jugadores j where e.id_equipo = j.equipo
   and altura > 2
);
```

5.3.2.- Consultas con tablas derivadas

Las consultas derivadas se utilizan en la claúsula FROM.

Por ejemplo: obtener la suma de los salarios de los jugadores que superan los 2m.

```
SELECT sum(dv.salario) FROM (
                select * from jugadores j where altura > 2
        ) dv;
```

5.4.- Consultas con múltiples tablas

Cuando consultamos una BD, en la mayoría de las ocasiones necesitamos información que se encuentra distribuida en más de 1 tabla. En estos casos tenemos que incluir los nombres de todas las tablas que necesitamos en la claúsula FROM.

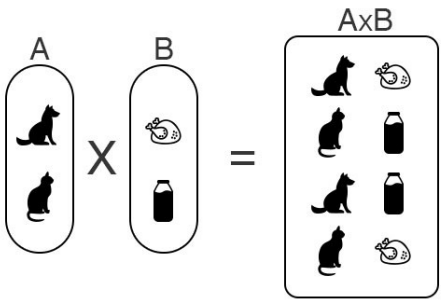
Cuando especificamos 2 tablas en nuestra consulta, se realiza una operación "**join**" entre las filas de las 2 tablas.

A continuación veremos que hay varias maneras de realizar un join entre 2 tablas.

5.4.1.- CROSS JOIN

Es el producto cartesiano entre 2 tablas. Cada fila de la primera tabla es operada con todas las filas de la segunda tabla.

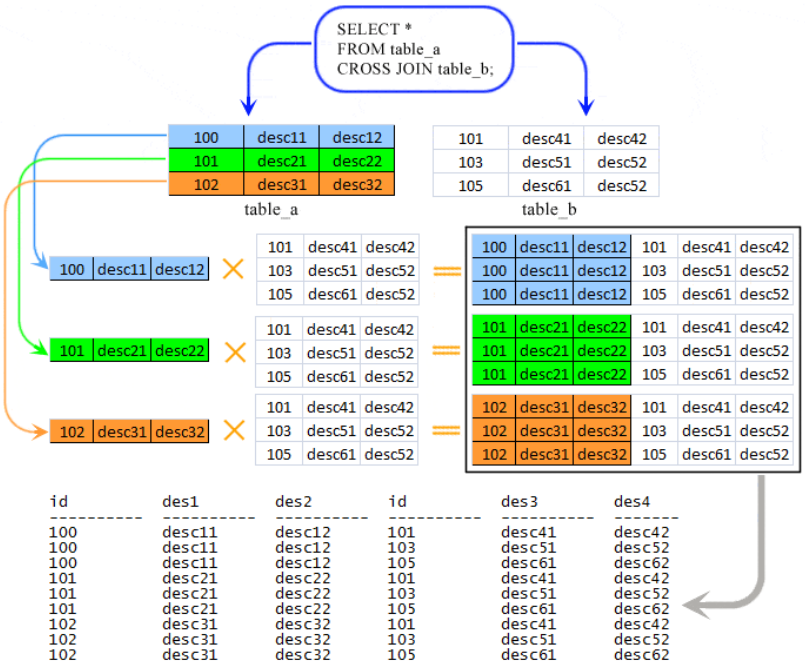
El producto cartesiano de 2 conjuntos :



Cartesian Product of Two Sets.

https://www.codewars.com/kata/cartesian-product

Un ejemplo de producto cartesiano entre 2 tablas:



id	des1	des2	id	des3	des4
100	desc11	desc12	101	desc41	desc42
100	desc11	desc12	103	desc51	desc52
100	desc11	desc12	105	desc61	desc62
101	desc21	desc22	101	desc41	desc42
101	desc21	desc22	103	desc51	desc52
101	desc21	desc22	105	desc61	desc62
102	desc31	desc32	101	desc41	desc42
102	desc31	desc32	103	desc51	desc52
102	desc31	desc32	105	desc61	desc62

https://www.w3resource.com/mysql/advance-query-in-mysql/mysql-cross-join.php

Esto puede crear conjuntos resultado muy grandes con los consiguientes riesgos o perjuicios. Por ejemplo, si tenemos 2 tablas relativamente pequeñas, t1 y t2, con 1.000 filas cada una, la siguiente consulta devolverá... 1.000.000 de filas!

```
SELECT * FROM t1 CROSS JOIN t2
```

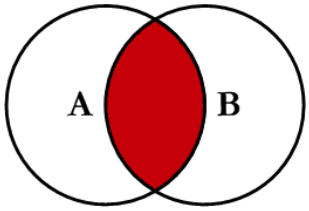
5.4.2.- INNER JOIN

INNER JOIN realiza el producto cartesiano entre aquellas filas que tienen el mismo valor en alguna de sus columnas. Las filas resultado han de tener el mismo valor en las 2 tablas para la columna utilizada al hacer el join.

Por ejemplo, si queremos obtener el nombre de cada jugador y el nombre del equipo al que pertenece podemos hacer:

```
SELECT j.nombre, j.apellido, e.nombre FROM jugadores j INNER JOIN equipos e ON j.equipo = e.id_equipo
```

Como se ve, las filas incluidas en el resultado son aquellas que cumplen la condición impuesta en la claúsula ON: en este caso el join entre ambas tablas se realiza a través del ID de cada equipo.



inner join

Si hay un jugador sin equipo, este jugador no aparecerá en las filas resultado.

5.4.3.- LEFT JOIN (left outer join)

LEFT JOIN funciona igual que un INNER JOIN pero el conjunto de filas resultado incluirá además las filas de la tabla A que no tengan correspondencia en la tabla B, en cuyo caso, las columnas de la tabla B en el resultado tomarán valores nulos.

Por ejemplo, actualicemos un jugador para dejarlo “sin equipo”:

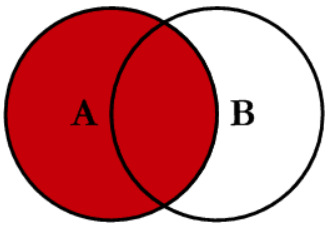
```
UPDATE jugadores set equipo = null where id_jugador = 6  # Mirza Teletovic
```

Si ahora ejecutamos la setencia del punto 5.4.2 veremos que devuelve 16 filas en vez de las 17 que devolvía.

Si queremos incluir en nuestro resultado aquellos jugadores que no tienen equipo, podemos hacer:

```
SELECT j.nombre, j.apellido, e.nombre FROM jugadores j LEFT JOIN equipos e ON j.equipo = e.id_equipo
```

Esta consulta nos devuelve de nuevo a todos los jugadores, incluido aquellos sin equipo.



left join

5.4.4.- RIGHT JOIN (right outer join)

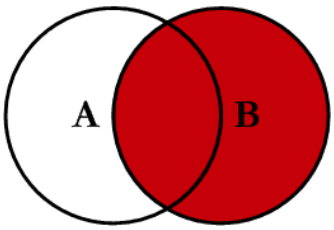
Es exactamente igual que LEFT JOIN pero en este caso son las filas de la tabla B las que se incluirán aunque no tengan correspondencia en la tabla A.

Por ejemplo, creemos un nuevo equipo que, de momento, no tendrá asignado ningún jugador:

```
INSERT INTO equipos VALUES (7,'Lakers', 'Los Angeles', null,5);
```

Este equipo no aparecerá en el resultado del INNER JOIN o del LEFT JOIN anteriores. Sin embargo si ejecutamos un RIGHT JOIN, sí aparecerá:

```
SELECT j.nombre, j.apellido, e.nombre FROM jugadores j RIGHT JOIN equipos e ON j.equipo = e.id_equipo
```



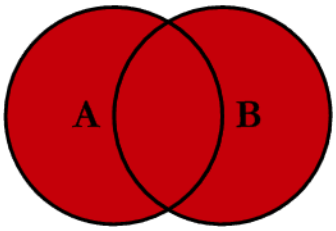
right join

Observa, sin embargo, que el jugador sin equipo ya no aparece en el resultado del RIGHT JOIN.

5.4.5.- FULL JOIN (full outer join)

Es una combinación de un LEFT JOIN y un RIGHT JOIN. Este tipo de join no está soportado en MySQL.

```
SELECT j.nombre, j.apellido, e.nombre FROM jugadores j FULL OUTER JOIN equipos e ON j.equipo = e.id_equipo
```



full join

5.5.- Unión, Intersección y Diferencia

SQL soporta 3 operaciones sobre conjuntos de resultados: la unión, la intersección y la diferencia.
Para poder utilizar estas operaciones, los conjuntos de filas deben tener el mismo número de campos y que éstos sean de dominios compatibles.

5.5.1.- Unión

Devuelve un conjunto con las filas de los subconjuntos.

```
SELECT …
UNION [ALL | DISTINCT]
SELECT ...
[ UNION [ALL | DISTINCT]
  SELECT… ]
```

DISTINCT: es el valor por defecto, y quiere decir que el resultado incluirá sólo filas únicas (Si una fila existía en ambos conjuntos, sólo será devuelta 1 vez)
ALL: el UNION devolverá todas las filas de ambos conjuntos, incluyendo duplicados.

Los nombres de las columnas serán los de las columnas del primer SELECT.
Todas las SELECT deben tener el mismo número y tipo de columnas.

Para ordenar un UNION simplemente hay que añadir la cláusula “ORDER BY” al final de la consulta.

5.5.2.- Intersección

El resultado de una intersección son las filas comunes a ambos conjuntos de resultados.

```
SELECT …

INTERSECT

SELECT …
```

MySQL no implementa esta operación, aunque existe una solución, por ejemplo:

Para obtener el listado de equipos que han jugado como locales y visitantes:

```
SELECT elocal FROM partidos
INTERSECT
SELECT evisitante FROM partidos;
```

En MySql podemos resolverlo con un join:

```
SELECT p1.elocal FROM partidos p1 inner join partidos p2 on p1.elocal = p2.evisitante;
```

5.5.3.- Diferencia

Esta operación devuelve el conjunto de filas NO comunes entre ambos conjuntos.

Por ejemplo, para obtener los nombres (nombre de pila) de jugadores del equipo 1 que no coincidan con nombres de jugadores en el equipo 2:

```
SELECT nombre FROM equipo where id_equipo = 1
```

```
MINUS
SELECT nombre FROM jugadores where id_equipo = 2
```

MySQL no implementa esta operación, aunque existen soluciones, por ejemplo utilizando una subconsulta:

```
SELECT nombre FROM jugadores WHERE id_equipo = 1
AND nombre NOT IN (SELECT nombre FROM jugadores WHERE id_equipo = 2);
```