**MySQl**

SELECT

Sintaxis.

Select campos from tabla where condición.

Clausulas.

**DISTINCT de SQL para resultados distintos**

SELECT DISTINCT columna1, columna2, ...

FROM nombre\_tabla;

**LIMIT y OFFSET en SQL**

**SELECT columnas**

**FROM tabla**

**LIMIT cantidad\_filas OFFSET inicio;**

**Patrones de búsqueda en SQL**

**LIKE**

La cláusula LIKE se usa para buscar valores que coincidan con un patrón específico en una columna. Es posible utilizar dos caracteres especiales con la cláusula LIKE. El símbolo '%' que representa cualquier cantidad de caracteres o el símbolo '\_' que representa un solo carácter.

SELECT \* FROM actor

WHERE first\_name LIKE 'E%';

SELECT \* FROM actor

WHERE first\_name LIKE 'E\_';

**IN**

SELECT columnas

FROM tabla

WHERE columna IN

(valor1, valor2, valor3, ...);

**IS NULL e IS NOT NULL en SQL**

Los operadores IS NULL e IS NOT NULL de SQL te permiten filtrar y realizar acciones específicas en función de la presencia o ausencia de valores NULL en las columnas de la tabla.

SELECT Name, Continent, IndepYear

FROM country

WHERE IndepYear IS NULL;

**Agrupamiento de resultados con GROUP BY en SQL**

¿Qué es GROUP BY?

GROUP BY es una cláusula de SQL que se usa para agrupar registros en un conjunto de resultados en función de los valores de una o más columnas. Se utiliza a menudo con funciones de agregado (como pueden ser COUNT(), MAX(), MIN(), SUM() y AVG()).

SELECT columna1, columna2, ...

FROM nombre\_tabla

WHERE condición

GROUP BY columna1, columna2, ...

ORDER BY columna1, columna2, ...;

Con SELECT podemos seleccionar una o varias columnas de datos.

FROM table\_references nos vale para seleccionar una o varias tablas que almacenan las columnas previamente indicadas.

En WHERE where\_condition se especifican la o las condiciones que se deben cumplir para los valores solicitados.

Dentro de la claúsula WHERE tenemos los siguientes condicionantes que son importantes y útiles:

GROUP BY: Nos permite agrupar los datos obtenidos de la base de datos.

HAVING: Permite hacer cálculos y condiciones más complejas que no se pueden hacer con la cláusula WHERE.

ORDER BY: Ordenación por columna.

LIMIT: Especificación del número de filas devueltas.

Sintaxis En su forma más simple, la sintaxis para la instrucción SELECT en MySQL es: SELECT campos FROM tablas [WHERE condiciones];

Eliminar Resultados repetidos con Distinct

En ciertos casos obtendremos como resultado datos que tendrán apariencia de estar repetidos, esto es solo apariencia porque la sentencia SELECT recupera las columnas de cada registro de la tabla especificada en el FROM, sin embargo, estos resultados pueden ser el mismo valor para cada uno y puede darse el caso que queramos excluir de los resultados esos valores aparentemente repetidos, para eso necesitamos agregar una clausura a nuestra sentencia SELECT, la clausura DISTINCT.

su sintaxis:

SELECT DISTINCT campos

FROM tabla

La cláusula GROUP BY agrupa un conjunto de filas en un grupo de filas de resumen por valores de columnas o expresiones. La cláusula GROUP BY devuelve una fila para cada grupo. En otras palabras, reduce el número de filas en el conjunto de resultados.

La clausura GROUP BY resume un grupo de filas o resultados en una única fila, esto quiere decir que si tienes varias filas con los mismos datos, el resultado que veras es un registro que resume a todos los iguales y estos son considerados como grupos.

**GROUP BY vs DISTINCT**

Cuando estamos iniciando con el lenguaje SQL, tendemos a confundir el funcionamiento de la clausura GROUP BY con la clausura DISTINCT, pero debemos recordar que DISTINCT elimina filas repetidas del resultado, y como acabamos de decir el GROUP\_BY agrupa las filas repetidas y las trata como un solo grupo.

GROUP BY y SUM, MAX, AVG, COUNT, MIN

Es muy común utilizar el GROUP BY con las funciones de agregación como son MAX, MIN, SUM, COUNT y SUM. Especialmente para realizar cálculos sobre los grupos y más.

Sintaxis

SELECT

col\_1,

col\_2,

col\_n,

funciones\_de\_agregacion

FROM

table

WHERE

condiciones

GROUP BY

col\_1,

col\_2,

col\_n

HAVIGN

expresion

La cláusula GROUP BY debe aparecer después de las cláusulas FROM y WHERE.

Después de GROUP BY colocamos separadas por coma las columnas o expresiones que serán utilizadas como criterio para agrupar las filas.

Ejemplo:

GROUP BY con funciones de agregación.

Las funciones de agregación permiten realizar cálculos de un conjunto de filas y devolver un solo valor. La cláusula GROUP BY a menudo se usa con una función de agregación para realizar cálculos y devolver un solo valor para cada subgrupo.

Por ejemplo, se desea conocer cuántos pedidos hay en cada estado, podemos agregar la función COUNT de la siguiente manera:

SELECT

estado, COUNT(\*)

FROM

ordenes

GROUP BY estado;

GROUP BY y HAVING

También existe una clausura llamada HAVING la cual funciona de forma similar al WHERE, la diferencia es que esta se aplica a los grupos después de que el WHERE a hecho su trabajo.

Adicionalmente el WHERE no permite utilizar funciones de agregación (SUM, AVG, etc.) en cambio el HAVING si te permite utilizar estas, lo cual es bien conveniente en muchos de los casos.

Hay que destacar que el HAVING solo funciona con el GROUP BY y este debe colocarse luego de el.

SELECT

idEmpresasTransporte,

estado,

count(\*) as 'Total'

FROM

pedido

WHERE

estado = 'en proceso'

GROUP BY

idEmpresasTransporte, estado

HAVING

count(\*) > 250

ORDER BY

idEmpresasTransporte

ORDER BY

Cuando realizamos una consulta a una tabla en una base de datos utilizamos la sentencia SELECT, este resultado que obtenemos siempre viene ordenado por el orden en que fueron registrado en la tabla.

Para poder cambiar este orden necesitamos agregar ORDER BY a el SELECT, y este nos permitirá establecer el orden de los resultados.

Sintaxis

SELECT

campos

FROM

tabla

ORDER BY

columna1 [ASC|DESC],

columna2 [ASC|DESC]

En la sintaxis podemos ver que el ORDER BY está ubicado después de las tablas del FROM, es decir al final de la consulta.

Otra cosa para notar es que podemos especificar las columnas por las cuales serán ordenados los resultados, podemos especificar una o más columnas y no necesariamente tienen que estar ubicadas en el SELECT (algo que no es permitido en otros SGBD).

Y Por último las palabras ASC y DESC, las cuales indican como deben ordenarse las filas o registros, ASC indica que los resultados serán ordenados de menor a mayor, entiéndase que si estamos tratando números serán desde el valor más pequeño hasta el más alto, y se hablamos de letras entonces serán ordenados desde la a hasta la z, en cuanto a la palabra DESC es el comportamiento contrario.

ORDER BY con expresiones

También podemos ordenar los resultados utilizando cálculos y expresiones aritméticas.

Por ejemplo:

SELECT

IdPedido,

IdProducto,

PrecioUnidad \* Cantidad

FROM

pedido\_detalle

ORDER BY

PrecioUnidad \*Cantidad ASC

Para evitar tener que poner la expresión en el SELECT y volver a repetirla en el ORDER BY podemos utilizar alias y así hacer el código más comprensible.

SELECT

IdPedido,

IdProducto,

PrecioUnidad \* Cantidad AS total

FROM

pedido\_detalle

ORDER BY

total ASC

Ordenar con una lista personalizada

Para este ejemplo utilizaremos la tabla Empleado, y usaremos la función llamada FIELD().

SELECT

nombre,

apellido,

pais

FROM empleado

ORDER BY FIELD(

pais,

'EE.UU.',

'Reino Unido'

)

Nombre Apellidos Pais

Nancy Davolio EE.UU.

Andrew Fuller EE.UU.

Janet Leverling EE.UU.

Margaret Peacock EE.UU.

Laura Callahan EE.UU.

Steven Buchanan Reino Unido

Michael Suyama Reino Unido

Robert King Reino Unido

Anne Dodsworth Reino Unido

La función FIELD() trabaja mapeando los elementos que se le pasan como argumentos. Y en el orden que los establezcamos, ese será el orden en que eran ordenados por el ORDER BY.

Hay que dejar claro que el primer parámetro es el nombre del campo que será usado para ordenar.

Operaciones con dos tablas o más – Join

Para realizar un select con dos o más tablas hay que tener en cuenta las llaves y relaciones que existen entre las tablas de la base de datos, este proceso consiste en una especie de unión de tablas que en SQL es llamada JOIN, del cual existen diferentes tipos de uniones de tablas.

Tipo de Join Función

INNER JOIN MySQL Recupera todos los registros con coincidencias en las tablas involucradas, excluye los que no tiene coincidencias

LEFT JOIN MySQL Recupera todos los registros con coincidencias en las tablas involucradas y todos los de la tabla ubicada a la izquierda del join, excluye los que no tiene coincidencias

RIGHT JOIN MySQL Recupera todos los registros con coincidencias en las tablas involucradas y todos los de la tabla ubicada a la derecha del join, excluye los que no tiene coincidencias

CROSS JOIN MySQL Recupera el producto cruzado de todos los registros de las tablas ivnolucradas.

NATURAL JOIN MySQL Recupera todos los registros con coincidencias en las tablas involucradas ,no requiere el uso de la palabra ON, excluye los que no tiene coincidencias

SELF JOIN MySQL Une una tabla consigo misma, utilizando alias, puede usa cualquiera de los join mencionado más arriba.

Cada tipo de JOIN ofrece su propia ventaja y debe ser utilizado según la necesidad.

Inner Join MySQL

Es el más sencillo de todos los join en MySQL y el más fácil, ya que se basa en las igualdades de los valores de una o más columnas específicas de las tablas involucradas.

La sintaxis del INNER JOIN es la siguiente:

SELECT tabla1.columna(s), tabla2.columna(s)

FROM tabla1

INNER JOIN tabla2

ON tabla1.columna = tabla2.columna

También podemos agregar más clausuras:

SELECT tabla1.columna(s), tabla2.columna(s)

FROM tabla1

INNER JOIN tabla2

ON tabla1.columna = tabla2.columna

WHERE condición

GROUP BY tabla1.columna(s) | tabla2.columna(s)

ORDER BY tabla1.columna(s) | tabla2.columna(s)

Una de las principales diferencias entre el SELECT con una tabla y el SELECT con dos o más tablas es que en el SELECT con una sola tabla no es obligatorio utilizar la notación objeto.atributo (donde el objeto es la tabla y el atributo es el campo), sin embargo cuando se ven involucrada más de una tabla entonces se hace necesario utilizar la notación objeto.atributo.

Left Join MySQL

El LEFT JOIN es idéntico al INNER JOIN, incluso la sintaxis es la misma, la única diferencia es que en vez de utilizar la palabra INNER esta es cambiada por LEFT. Otra diferencia es su funcionamiento, a diferencia del INNER JOIN donde solo se recuperan las filas que coinciden en valor en la clausura ON, en el LEFT JOIN son recuperados todos los registros de la tabla ubicada en la clausura FROM y solo aquellas filas de la tabla ubicada en el LEFT JOIN que coinciden en valor en la clausura ON.

De forma resumida podemos decir que el LEFT JOIN recupera todas las filas de la tabla ubicada en el FROM y solo lo que coincide de la tabla ubicada en el LEFT JOIN.

Su sintaxis

SELECT tabla1.columna(s), tabla2.columna(s)

FROM tabla1

LEFT JOIN tabla2

ON tabla1.columna = tabla2.columna

Right Join MySQL

El RIGHT JOIN realiza la acción inversa al LEFT JOIN, el cual incluye todos los registros de la tabla colocada a la izquierda del JOIN, pero solo incluye los registros que coinciden en la clausura ON.

Al realizar lo inverso entonces quiere decir que recupera todos los registros de la tabla colocada a la derecha del JOIN y excluye aquellos que no tienen ninguna coincidencia en el ON de la tabla colocada a la izquierda del JOIN.

La sintaxis es la siguiente

SELECT tabla1.columna(s), tabla2.columna(s)

FROM tabla1

RIGHT JOIN tabla2

ON tabla1.columna = tabla2.columna

Cross Join MySQL

El CROSS JOIN presenta el producto cartesiano de las filas de las tablas cruzadas. El resultado tendrá todos los registros de la tabla ubicada en el FROM izquierda combinados con cada uno de los registros de la tabla ubicada en CROSS JOIN.

Algo que debemos destacar es que no se utiliza la clausura ON para establecer condiciones.

Tal y como lo muestra la siguiente ilustración.

Una consulta utilizando CROSS JOIN quedaría así.

SELECT \* FROM almacenes CROSS JOIN empleado

No hay mucho que decir sobre este join, pero es posible que le encuentres alguna utilidad.

Natural Join

El NATURAL JOIN es una simplificación de INNER JOIN con la característica de que cuando los campos de las tablas que serán utilizadas en el JOIN, dígase los campos por los cuales serán comparados los valores en la clausura ON, tienen el mismo nombre se puede omitir el uso de la clausura ON, y el NATURAL JOIN se encargará de comparar las coincidencias y el resultado será el mismo que un INNER JOIN con la clausura ON.

Alias En Las tablas

Existe la posibilidad de renombrar el nombre de las tablas, pero a diferencia de las columnas cuyo propósito es una mejor presentación de los datos, cuando se renombra una tabla se hace para realizar consultas con más facilidad.

Self Join

El self join no es más que hacer un join a una tabla consigo misma.

¿Cuál sería la finalidad?, simple, comparar los datos de una tabla consigo mismo, en los joins se ha mostrado como unir tablas mediante campos relacionado, y recuperar valores por coincidencias, pero cuando se necesita comparar los datos de una tabla con los datos de sí misma, entonces tenemos solo dos (2) opciones:

1. Crear otra tabla y copiar los registros de la primera en la segunda y luego comparar.

2. Utilizar self join

Operadores SET

Los operadores SET (Conjunto) combinan los resultados de dos o más consultas en un único resultado. Las consultas que contienen operadores SET (Conjunto) son llamadas consultas compuestas (compound queries).

Operadores SET

Operador Obtiene

UNION Todas las filas distintas seleccionadas por ambas consultas

UNION ALL Todas las filas seleccionadas por ambas consultas, incluyendo todos los duplicados

INTERSECT Todas las filas distintas seleccionadas por ambas consultas

MINUS Todas las filas distintas seleccionadas por la primera sentencia SELECT que no se encuentren en la segunda sentencia SELECT

Todos los operadores SET tienen la misma precedencia. Si una sentencia SQL contiene múltiples operadores SET, el servidor de Oracle evalúa estos de izquierda a derecha. Se puede hacer uso de los paréntesis para especificar un orden explícito de evaluación como pueden ser las consultas que usan el operador INTERSECT con operadores SET.

Operador UNION

El operador UNION obtiene todas las filas seleccionadas por ambas consultas.

Use el operador UNION para obtener todas las filas de múltiples tablas y eliminar cualquier fila duplicada.

Normas a seguir

El número de columnas y tipo de datos deben ser idénticos en todas las sentencias SELECT usadas en las consultas. Los nombres de las columnas no necesitan ser idénticas.

La UNION opera sobre todas las columnas inicialmente seleccionadas.

Los valores NULOS no son ignorados durante la verificación de duplicados

El operador IN tiene una mayor precedencia que el operador UNION

Por defecto, el resultado es ordenado ascendentemente por la primera columna de la cláusula SELECT

El siguiente ejemplo muestra el uso de UNION

SELECT nombre, apellido1 , Apellido2, ife, curp

FROM empleados

UNION

SELECT nombre, apellido1 , apellido2, ife, curp

FROM clientes

ORDER BY nombre, apellido1 , apellido2;

Operador UNION ALL

Use el operador UNION ALL para obtener todas las filas de múltiples consultas.

Normas a seguir

Diferente a UNION, las filas duplicadas no son eliminadas y el resultado no es ordenado por defecto.

La palabra reservada DISTINCT no puede ser usada

Nota: Con excepción de las normas anteriores, UNION y UNION ALL es lo mismo.

SELECT nombre, apellido1 , Apellido2, ife, curp

FROM empleados

UNION ALL

SELECT nombre, apellido1 , apellido2, ife, curp

FROM clientes

ORDER BY nombre, apellido1 , apellido2;

Operador INTERSECT

Use el operador INTERSECT para obtener todas las filas comunes de múltiples consultas.

Normas a seguir

El número de columnas y tipos de datos de las columnas seleccionadas inicialmente por la sentencia SELECT, deben ser idénticos en todas las sentencias SELECT usadas en la consulta. El nombre de las consultas no necesita ser idéntico.

Invertir el orden de las tablas intersectadas no modifica el resultado

INTERSECT no ignora los valores NULOS

SELECT nombre RFC FROM clientes\_2015

INTERSECT

SELECT nombre, RFC FROM clientes\_2016

Operador MINUS

Use el operador MINUS para obtener las filas de la primera consulta que no estén presentes en la segunda consulta (la primera sentencia SELECT MENOS la segunda sentencia SELECT)

Normas a seguir

El número de columnas y tipos de datos de las columnas de la sentencia SELECT inicial deben ser idénticas en todas las sentencias SELECT usadas en la consulta. Los nombres de las columnas no necesitan ser idénticas.

Todas las columnas en la cláusula WHERE deben estar en la cláusula SELECT para que el operador MINUS trabaje.

Ejemplo: la siguiente consulta combina el resultado de las 2 consultas, y devuelve solo los elementos de la primera consulta que no se encuentren en la segunda, es decir todos aquellos clientes que lo fueron en el 2015 pero no lo fueron en el 2016,

SELECT nombre, rfc FROM clientes\_2015

MINUS

SELECT nombre, rfc FROM clientes\_2016

Vistas

Una vista es una tabla virtual cuyo contenido está definido por una consulta.

Una vista es sencillamente un objeto de base de datos que presenta datos de tablas. Se trata de una consulta SQL que está permanentemente almacenada en la Base de datos y a la que se le asigna un nombre, de modo que los resultados de la consulta almacenada son visibles a través de la vista, y SQL permite acceder a estos resultados como si fueran de hecho una tabla real en la base de datos.

Las tablas y las vistas comparten el mismo espacio de nombres en la base de datos, por lo tanto, una base de datos no puede contener una tabla y una vista con el mismo nombre.

Las vistas suelen utilizarse para centrar, simplificar y personalizar la percepción de la base de datos para cada usuario. Las vistas pueden emplearse como mecanismos de seguridad, que permiten a los usuarios obtener acceso a los datos por medio de la vista, pero no les conceden el permiso de obtener acceso directo a las tablas subyacentes de la vista. Las vistas se pueden utilizar para realizar particiones de datos y para mejorar el rendimiento cuando se copian, se importan y se exportan datos.

Mediante vistas es posible presentar datos de distintos servidores. Por ejemplo, para combinar datos de distintos servidores remotos o en un servidor de múltiples procesadores, cada uno de los cuales almacenan datos para una región distinta de su organización, puede crear consultas distribuidas o paralelas aumentando la eficiencia de las consultas.

Mediante diversas cláusulas es factible crear, modificar, eliminar y administrar vistas. La sintaxis básica para estas cláusulas es genérica entre diversos gestores de base de datos. Sin embargo en lo particular cada gestor implementa la administración de estas de forma diferente.

La sintaxis básica de una vista:

CREATE VIEW nombre\_vista AS consulta;

Sintaxis

CREATE VIEW

CREATE [OR REPLACE]

VIEW nombre\_vista AS sentencia\_select

[WITH [CASCADED | LOCAL] CHECK OPTION]

WITH READ ONLY

Ejemplo: Municipios del estado de Veracruz, México

CREATE OR REPLACE VIEW veracruz AS

SELECT M.idMunicipio, M.idEstado, M.municipio

FROM municipios M, estados E

WHERE (E.idEstado = M.idEstado) AND (E.estado = 'Veracruz')

La sintaxis para eliminar una vista es:

DROP VIEW nombreVista

[ CASCADE CONSTRAINTS ];

Eliminamos la vista denominada 'caballoDeHierro':

DROP VIEW caballoDeHierro;

Al borrar una vista no perderá los datos, porque la vista está basada en tablas subyacentes de la B.D.

Únicamente el creador o un usuario con el privilegio DROP ANY VIEW puede eliminar una vista.

Si se elimina una tabla a la que hace referencia una vista, la vista no se elimina, hay que eliminarla explícitamente.

Actualización de vistas

Algunas vistas pueden actualizarse mediante sentencias INSERT, DELETE o UPDATE sobre ellas, considerando las siguientes reglas:

No se debe especificar DISTINCT para que las filas duplicadas no se eliminen de los resultados de las consultas.

La cláusula FROM debe especificar sólo una tabla actualizable.

Cada elemento de selección debe ser una referencia de columna simple ( no puede contener expresiones, columnas calculadas, ni funciones de columna).

La cláusula WHERE no debe incluir una subconsulta.

La consulta no debe incluir una cláusula GROUP BY o HAVING.

La consulta no debe incluir sentencias: UNION, UNION ALL, INTERSECT, MINUS

Si una columna NOT NULL no contiene un DEFAULT, no es posible insertar mediante vistas

Claúsula WITH CHECK OPTTION

Asegura que un comando DML sobre la vista mantiene el dominio de la misma.

CREATE OR REPLACE VIEW veracruz AS

SELECT M.idMunicipio, M.idEstado, M.municipio

FROM municipios M, estados E

WHERE (E.idEstado = M.idEstado) AND (E.estado = 'Veracruz')

WITH CHECK OPTION

De esta vista usted no puede modificar el campo idEstado, porque al hacerlo ya no aparecería en la vista. El resto de los campos de la vista son actualizables

Si usas el comando INSERT solo sería válido si el valor del campo idEstado fuera 30. O sea el correspondiente al estado de Veracruz

Claúsula WITH READ ONLY

Agregando la opción WITH READ ONLY en la definición de la vista, se asegura que no ocurran operaciones DML.

CREATE OR REPLACE VIEW veracruz AS

SELECT M.idMunicipio, M.idEstado, M.municipio

FROM municipios M, estados E

WHERE (E.idEstado = M.idEstado) AND (E.estado = 'Veracruz')

WITH READ ONLY

**Contenidos**

**1 Operaciones de conjuntos y MySQL**

**2 Unión**

**3 Intersección**

**4 Diferencia**

**5 Producto cartesiano**

**6 Concatenación natural**

**7 División**

**7.1 La división del álgebra relacional con**

**EXISTS**

**7.2 Otros usos**

**7.3 Alternativas**

**7.4 Eficiencia**

Operaciones de conjuntos y MySQL

Un operador sobre conjuntos combina el resultado de dos sentencias select en un único resultado. Dependiendo del tipo de operación esas sentencias deberán cumplir unos requisitos en cuanto al resultado que dan.

Los operadores de conjuntos definidos para el álgebra relacional, base sobre la que se cimenta SQL, son la unión, intersección, diferencia, producto cartesiano y división. En otros motores de base de datos se dispone de algunos operadores como UNION,

INTERSECT y MINUS.

En MySQL solo está disponible el operador UNION, aunque la operativa de los otros puede simularse con otros operadores de los que ya hemos visto unos cuantos.

Unión

Al utilizar el operador UNION entre dos sentencias select, el resultado final estará compuesto por todas aquellas filas que aparecen en el resultado de como mínimo una de las select. El operador UNION elimina filas duplicadas en el resultado final. El operador UNION ALL opera de igual modo que el operador UNION, pero no

elimina filas duplicadas en el resultado final.

Supongamos que queremos saber el nombre de los profesores que son ASO6 o imparten asignaturas de 6 créditos.

Veamos primero el resultado de cada consulta por separado.

1) Nombre de los profesores cuya categoría es ASO6.

select nombre from profesores where categoria='ASO6';

2) Nombre de los profesores que imparten asignaturas de 6 créditos.

select nombre

from profesores p, imparte i, asignaturas

where p.dni=i.dni and asignatura=codigo and creditos=6;

Nombre de los profesores que son ASO6 o imparten asignaturas de 6 créditos.

1 de 10

select nombre from profesores where categoria='ASO6'

**UNION**

select nombre

from profesores p, imparte i, asignaturas

where p.dni=i.dni and asignatura=codigo and creditos=6;

**nombre**

RAFAEL ROMERO

EVA GOMEZ

La misma consulta pero solucionada con UNION ALL

select nombre from profesores where categoria='ASO6'

**UNION ALL**

select nombre

from profesores p, imparte i, asignaturas

where p.dni=i.dni and asignatura=codigo and creditos=6;

**nombre**

RAFAEL ROMERO

EVA GOMEZ

EVA GOMEZ

RAFAEL ROMERO

**Intersección**

Aunque no existe tal operador en MySQL, normalmente se puede sustituir por consultas ya conocidas.

DNI de los profesores que imparten y preparan.

IMPARTE[dni]

**∩**

(COORDINADORES[dni])

Lo que debería solucionarse como

*select dni from imparte*

***INTERSECT***

*select dni from prepara;*

en realidad, en MySQL, debemos expresarlo como:

select **distinct** i.dni

2 de 10

from imparte i, prepara pp

where pp.dni=i.dni;

**dni**

21111222

21333444

Se usa el modificador distinct para que la expresión sustituya realmente la operativa de la intersección: lasmoperaciones de conjuntos no devuelven duplicados (al igual que union all es una extensión para que se puedan resolver cierto tipo de consultas que sí necesitan los duplicados). No obstante, lo usaremos o no dependiendo de si lo necesitamos o no.

**Diferencia**

El operador MINUS tampoco está implementado en MySQL pero es fácilmente sustituible por expresiones NOT IN.

En todo caso, el resultado final estará compuesto sólo por aquellas filas que aparecen en el resultado de la primera select y no aparecen en el resultado de la segunda.

Nombre de los profesores que son TEU y no imparten asignaturas de 6 créditos.

PROFESORES donde categoría='TEU' [nombre]

-

(PROFESORES x IMPARTE x ASIGNATURA

donde (PROFESORES.dni = IMPARTE.dni y codigo=asignatura y créditos=6)

[nombre])

Lo que debería solucionarse como

*select nombre from profesores where categoria='TEU'*

***MINUS***

*select nombre*

*from profesores p, imparte i, asignaturas*

*where p.dni=i.dni and asignatura=codigo and creditos=6;*

se soluciona así:

select nombre from profesores where categoria='TEU'

**and nombre NOT IN**

(select nombre

from profesores p, imparte i, asignaturas

where p.dni=i.dni and asignatura=codigo and creditos=6);

3 de 10

**nombre**

MANUEL PALOMAR

Esta sería la consulta exactamente equivalente a la diferencia de conjuntos tal y como está definida en el álgebra relacional, con relaciones compatibles, pero esto es SQL, podemos simplificar la consulta y obtendremos el mismo resultado:

select nombre from profesores

where categoria='TEU'

and **dni NOT IN**

(select dni

from imparte i, asignaturas

where asignatura=codigo and creditos=6);

**Producto cartesiano**

En SQL, y como ya se ha dicho anteriormente, el producto cartesiano se realiza con una *select* "sin *where*".

select dni, codigo

**from profesores, asignaturas**

**dni codigo**

21111222 DGBD

21222333 DGBD

21333444 DGBD

21111222 FBD

21222333 FBD

21333444 FBD

21111222 FP

21222333 FP

21333444 FP

21111222 HI

21222333 HI

21333444 HI

21111222 PC

21222333 PC

21333444 PC

No obstante, el producto cartesiano, como cualquier otro tipo de consulta, puede filtrar las filas resultado a nuestra conveniencia.

Dni de los profesores que imparten 2 o más asignaturas

select distinct i1.dni

from imparte i1, imparte i2

where i1.dni = i2.dni

4 de 10

and i1.asignatura != i2.asignatura;

Del producto cartesiano de una tabla por sí misma solo nos interesan las filas en las que el profesor "de la izquierda" es el mismo que el "de la derecha"; sí, además, las asignaturas son diferentes, la conclusión no puede ser otra que ese profesor cumple con la condición.

Aunque para el ejemplo que hemos propuesto hay una forma más amigable de solucionar este enunciado, la que mostramos a continuación, el producto cartesiano es una opción que podemos utilizar cuando creamos conveniente.

select dni from imparte group by dni having count(\*) >= 2;

Nombre de los profesores que imparten asignaturas de 6 créditos y no son TEU.

PROFESORES x IMPARTE x ASIGNATURA

donde (PROFESORES.dni = IMPARTE.dni y codigo=asignatura y créditos=6)

[nombre]

-

(PROFESORES donde categoría='TEU' [nombre])

select nombre

from profesores p, imparte i, asignaturas

where p.dni=i.dni and asignatura=codigo and creditos=6

**and nombre NOT IN**

(select nombre from profesores where categoria='TEU');

**nombre**

RAFAEL ROMERO