Unidad 5

El lenguaje SQL. DML y DCL

Bases de datos

Índice

Introducción

En la unidad anterior vimos el primero de los lenguajes que usa SQL y como era su estructura. En esta unidad veremos como terminar de interpretar estos sublenguajes y sus múltiples funciones.

En la primera instancia de la unidad veremos cómo se usa el DML o lenguaje de manipulación de datos que nos dará la opción de consultar los registros de una tabla, insertar registros sobre una tabla, modificarlos y eliminarlos.

Más tarde veremos lo que son las vistas, del Lenguaje DDL y como estas funcionan en base a las consultas aprendidas con **SELECT**.

Por último, se usará el lenguaje DCL para establecer un control sobre los datos con el uso de permisos y privilegios.

Al finalizar esta unidad

- Dominaremos la sentencia SELECT.
- Sabremos utilizar adecuadamente las funciones de agregación.
- Seremos capaces de realizar composiciones internas y externas.
- Conoceremos las combinaciones de consultas y subconsultas.
- Tendremos destreza en el uso de funciones integradas y alias.
- Sabremos manejar la sintaxis del DML para inserciones, actualizaciones y borrados.
- Entenderemos como especifica el DCL la asignación de permisos de usuarios.
- Estaremos familiarizados con el concepto de transacción.

5.1. Lenguaje de manipulación de datos (DML)

Dentro de los sublenguajes que usa SQL teníamos el lenguaje de manipulación de datos, data manipulation language, o DML. Este lenguaje nos permite que se pueda acceder a la información que se almacena en una base de datos y modificarla. Así como puede visualizar y modificar la información, también puede añadir de esta, porque si no, no habría información que visualizar. Pero generalmente en mayor uso para este lenguaje es el de la consulta de información, también lo es para SQL, y esta tarea se suele ejecutar con la sentencia SELECT.

5.1.1. La sentencia SELECT

```
SELECT [DISTINCT] campos
       FROM tablas
       [WHERE condición]
       [GROUP BY campos
              [HAVING condición_agrupación]]
       [ORDER BY campos];
```

Cuando nos referimos a las tablas seleccionadas, estas pueden ser una o varias, pero primero vamos a ir viendo esto sobre simplemente una tabla.

5.1.2. Consulta de registros sobre una tabla

Para poder realizar consultas sobre tablas en una cierta base de datos, hemos creado una de ejemplo de Pokémon, de la cual podemos ver su estructura a continuación:

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| pokemondb          |
| sys                |
+--------------------+
5 rows in set (0,00 sec)

mysql> use pokemondb;
Database changed
mysql> show tables;
+---------------------+
| Tables_in_pokemondb |
+---------------------+
| pokemon             |
| pokemon_tipo        |
| tipo                |
| tipo_ataque         |
+---------------------+
```

```
mysql> desc pokemon;
+----------------+-------------+------+-----+---------+----------------+
| Field          | Type        | Null | Key | Default | Extra          |
+----------------+-------------+------+-----+---------+----------------+
| numero_pokedex | int         | NO   | PRI | NULL    | auto_increment |
| nombre         | varchar(15) | NO   |     | NULL    |                |
| peso           | double      | NO   |     | NULL    |                |
| altura         | double      | NO   |     | NULL    |                |
+----------------+-------------+------+-----+---------+----------------+
4 rows in set (0,01 sec)

mysql> desc pokemon_tipo;
+----------------+------+------+-----+---------+-------+
| Field          | Type | Null | Key | Default | Extra |
+----------------+------+------+-----+---------+-------+
| numero_pokedex | int  | NO   | PRI | NULL    |       |
| id_tipo        | int  | NO   | PRI | NULL    |       |
+----------------+------+------+-----+---------+-------+
2 rows in set (0,01 sec)
mysql> desc tipo;
+----------------+-------------+------+-----+---------+----------------+
| Field          | Type        | Null | Key | Default | Extra          |
+----------------+-------------+------+-----+---------+----------------+
| id_tipo        | int         | NO   | PRI | NULL    | auto_increment |
| nombre         | varchar(10) | NO   |     | NULL    |                |
| id_tipo_ataque | int         | NO   | MUL | NULL    |                |
+----------------+-------------+------+-----+---------+----------------+
3 rows in set (0,00 sec)

mysql> desc tipo_ataque;
+----------------+------------+------+-----+---------+----------------+
| Field          | Type       | Null | Key | Default | Extra          |
+----------------+------------+------+-----+---------+----------------+
| id_tipo_ataque | int        | NO   | PRI | NULL    | auto_increment |
| tipo           | varchar(8) | NO   |     | NULL    |                |
+----------------+------------+------+-----+---------+----------------+
2 rows in set (0,00 sec)
```

Imagen 1. Base de datos para los ejemplos.

En la definición de la sentencia SELECT que vimos anteriormente, campos hace referencia a los **campos** de la tabla que queremos que aparezcan en la selección, estos se separan por una coma en caso de querer poner varios, y aparecerán en el orden que hemos seleccionado. Si usamos el símbolo *, se seleccionarán todos los campos de la tabla. Por otro lado, la opción **DISTINCT** elimina valores repetidos.

En la estructura de **WHERE, condición** es la condición simple o compuesta de varias que vamos a usar para filtrar en primera instancia los resultados que queremos obtener. Estas condiciones pueden ser comparaciones de expresiones, valores, variables, etc. Los operadores disponibles en esta sentencia son:

- **De comparación: <, <=, >, >=, <>, =.** Menor, menor o igual, mayor, distinto e igual respectivamente. Se comparan dos valores, pueden ser con valores fijos o expresiones o con valores de otro campo.

  En el siguiente ejemplo vamos a seleccionar el número de pokedex y los nombres de los Pokémon que pesen 20 kg.

```
mysql> SELECT numero_pokedex, nombre
    -> FROM pokemon
    -> WHERE peso = 20;
+----------------+-----------+
| numero_pokedex | nombre    |
+----------------+-----------+
|             30 | Nidorina  |
|             61 | Poliwhirl |
|             74 | Geodude   |
+----------------+-----------+
3 rows in set (0,04 sec)
```

Imagen 2. Sentencia SELECT con condición 1.

  El operador usado para la distinción también se puede simbolizar con != (no igual).

  - BETWEEN ... AND. Este tipo de comparación ejerce una comparativa en un intervalo cerrado, donde los valores límite también están incluidos.

    En el siguiente ejemplo, vamos a seleccionar la altura y el nombre de los Pokémon que midan entre 10 y 30 metros.

```
mysql> SELECT altura, nombre
    -> FROM pokemon
    -> WHERE altura BETWEEN 10 AND 30;
Empty set (0,00 sec)
```

Imagen 3. Sentencia SELECT con condición 2.

Podemos ver que no salen resultados, pero esto no es un fallo, sino que simplemente no hay una concordancia para la condición solicitada.

- **Lógicos: AND, OR, NOT. Y, o y no, respectivamente.** Estos son los operadores lógicos basados en el álgebra de Boole, que nos ayudan a hacer concatenaciones de condiciones, y que conllevan los siguientes usos:

  - Dos expresiones lógicas que se unen por medio del operador AND serán ciertas (true) si ambas son ciertas. Es decir, se tienen que cumplir ambas exigencias.

AND	Cierto	Falso
Cierto	Cierto	Falso
Falso	Falso	Falso

Cuadro 1. Tabla de verdad para el operador AND.

» Si dos expresiones lógicas se usen por el operador OR, la expresión será cierta si una de las dos es cierta.

OR	Cierto	Falso
Cierto	Cierto	Cierto
Falso	Cierto	Cierto

Cuadro 2. Tabla de verdad para el operador OR.

» Cuando se usa una expresión precedida del operador NOT, este invierto su significado, por lo que será cierta si la expresión es falsa.

En el siguiente ejemplo vamos a seleccionar el identificador del tipo de los Pokémon con identificador uno y número de pokedex también uno.

```
mysql> SELECT id_tipo
    -> FROM pokemon_tipo
    -> WHERE numero_pokedex = 1
    -> AND id_tipo = 1;
Empty set (0,00 sec)
```

Imagen 4. Sentencia SELECT con condición 3.

Si combinamos los operadores **AND y OR**, el primero prevalece sobre el segundo, con esto nos referimos a que se evalúa primero la expresión con dicho operador.

> **De concatenación:** Usamos el operador || para separar cadenas de caracteres y concatenarlas y que estas aparezcan especificadas en la selección solicitada.

Hay ocasiones en las que el SGBD no admite dicho operador, y se debe de usar el operador +.

> **IS NULL.** Esta comparación simplemente prueba si el valor es nulo, lo opuesto, comprobar que no es nulo, sería con el operador IS NOT NULL.

En el ejemplo siguiente seleccionamos todos los tipos que no tienen el nombre nulo.

Aquí hay que recordar que un valor nulo, es cuando no hay valor, pero un valor a 0 si es un valor, es decir, hay un valor, que es 0.

```
mysql> SELECT *
    -> FROM tipo
    -> WHERE nombre IS NOT NULL;
+---------+-----------+----------------+
| id_tipo | nombre    | id_tipo_ataque |
+---------+-----------+----------------+
|       1 | Agua      |              2 |
|       2 | Bicho     |              1 |
|       3 | Dragón    |              2 |
|       4 | Eléctrico |              2 |
|       5 | Fantasma  |              1 |
|       6 | Fuego     |              2 |
|       7 | Hielo     |              2 |
|       8 | Lucha     |              1 |
|       9 | Normal    |              1 |
|      10 | Planta    |              2 |
|      11 | Psíquico  |              2 |
|      12 | Roca      |              1 |
|      13 | Tierra    |              1 |
|      14 | Veneno    |              1 |
|      15 | Volador   |              1 |
+---------+-----------+----------------+
```

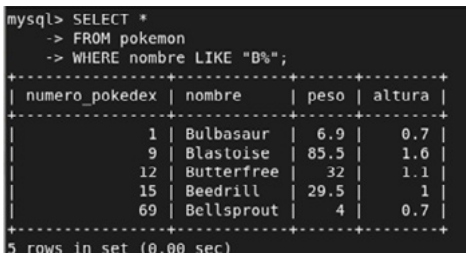
Imagen 5. Sentencia SELECT con IS NOT NULL.

> **LIKE:** es el operador que compara por cadenas de caracteres aproximados, no coincide todo el contenido. Para ello se apoya en el uso de comodines % y _, que sustituyen a una cadena de caracteres y un único carácter respectivamente. Tenemos las siguientes reglas que definen estos usos:

Texto que empiece por una cadena	campo LIKE "cadena%"
Texto que acabe por una cadena	campo LIKE "%cadena".
Texto que contenga una cadena	campo LIKE "%cadena%".

Cuadro 3. Uso básico del operador LIKE.

En el siguiente ejemplo vamos a seleccionar todos los datos sobre los Pokémon cuyo nombre empieza por la letra B.



```
mysql> SELECT *
    -> FROM pokemon
    -> WHERE nombre LIKE "B%";
+----------------+------------+------+--------+
| numero_pokedex | nombre     | peso | altura |
+----------------+------------+------+--------+
|              1 | Bulbasaur  |  6.9 |    0.7 |
|              9 | Blastoise  | 85.5 |    1.6 |
|             12 | Butterfree |   32 |    1.1 |
|             15 | Beedrill   | 29.5 |      1 |
|             69 | Bellsprout |    4 |    0.7 |
+----------------+------------+------+--------+
5 rows in set (0.00 sec)
```

Imagen 6. Sentencia SELECT con LIKE.

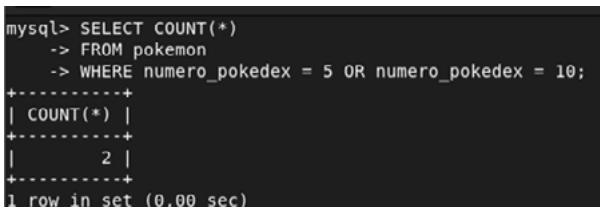
Hay veces en las que los comodines usados no son % y _, sino que se usan * y ?, respectivamente.

5.1.3. Funciones de agregación

En la sentencia **SELECT** seleccionábamos ciertos campos, donde también se pueden mostrar expresiones de agregación resultado de usar funciones específicas. Esto quiere decir, cálculos de los registros de la tabla a los que afecta la consulta. Las funciones que tenemos son las siguientes;

- **COUNT(*|campo).** Esta función nos muestra el número de registros que cumplen con la función, si en vez de usar el comodín *, usamos el nombre de un campo en concreto, no se tendrán en cuenta los valores que sean nulos en dicho campo.

En el siguiente ejemplo seleccionamos el número de registros que tienen número de pokedex 5 o 10.



```
mysql> SELECT COUNT(*)
    -> FROM pokemon
    -> WHERE numero_pokedex = 5 OR numero_pokedex = 10;
+----------+
| COUNT(*) |
+----------+
|        2 |
+----------+
1 row in set (0.00 sec)
```

Imagen 7. Sentencia SELECT con COUNT.

- **AVG(campo).** Realiza la media aritmética de un campo numérico o un intervalo.
- **SUM(campo).** Realiza la suma de valores de un campo numérico o un intervalo.
- **MAX(campo).** Nos muestra el valor máximo de un campo.
- **MIN(campo).** Nos muestra el valor mínimo de un campo.

Si lo que queremos es que el resultado de estas funciones se nos muestre divido por los resultados de cada registro dependiendo de otros campos, es decir, que el resultado de cada función es para un valor de un campo en particular, debemos de usar la cláusula **GROUP BY**, que tiene la siguiente forma:

```
SELECT campos, función
        FROM tabla
GROUP BY campos;
```

Si añadimos campos en SELECT, estos por obligación deben de aparecer en GROUP BY.

En el siguiente ejemplo seleccionamos cuantos registros tenemos para cada uno de los tipos de ataque.

```
mysql> SELECT id_tipo_ataque, COUNT(*)
    -> FROM tipo
    -> GROUP BY id_tipo_ataque;
+----------------+----------+
| id_tipo_ataque | COUNT(*) |
+----------------+----------+
|              1 |        8 |
|              2 |        7 |
+----------------+----------+
2 rows in set (0,00 sec)
```

Imagen 8. GROUP BY.

Dentro de estas agrupaciones y ordenaciones, se puede usar además la cláusula HAVING para dar una condición sobre el resultado de la agrupación, esto sería un funcionamiento parecido a WHERE, pero sobre GROUP BY.

En el siguiente ejemplo seleccionamos lo mismo que antes, pero queremos que el número de registros que se nos muestre sea el que es mayor de 8.

```
mysql> SELECT id_tipo_ataque, COUNT(*)
    -> FROM tipo_ataque
    -> GROUP BY id_tipo_ataque
    -> HAVING COUNT(*) = 8;
Empty set (0,04 sec)
```

Imagen 9. HAVING.

También podemos usar la cláusula **WHERE** antes de **GROUP BY** y **HAVING** para filtrar la consulta antes de realizar las agrupaciones.

En la imagen siguiente vemos como seleccionamos el peso y la altura de los Pokémon, pero solo de los que su altura es menor que 10. Posteriormente agrupamos por ambos campos y seleccionamos que el conteo de nombres (registros totales que nos interesan), sea menor que 30. La consulta devuelve muchos valores.

```
mysql> select peso, altura, COUNT(nombre)
    -> FROM pokemon
    -> WHERE altura < 10
    -> GROUP BY peso, altura
    -> HAVING COUNT(nombre) < 30;
+-------+--------+---------------+
| peso  | altura | COUNT(nombre) |
+-------+--------+---------------+
|   6.9 |    0.7 |             1 |
|    13 |      1 |             1 |
|   100 |      2 |             1 |
|   8.5 |    0.6 |             1 |
|    19 |    1.1 |             1 |
|  90.5 |    1.7 |             1 |
|     9 |    0.5 |             2 |
|  22.5 |      1 |             1 |
|  85.5 |    1.6 |             1 |
|   2.9 |    0.3 |             1 |
|   9.9 |    0.7 |             1 |
|    32 |    1.1 |             1 |
|   3.2 |    0.3 |             1 |
|    10 |    0.6 |             1 |
|  29.5 |      1 |             3 |
|   1.8 |    0.3 |             1 |
|    30 |    1.1 |             2 |
|  39.5 |    1.5 |             1 |
|   3.5 |    0.3 |             1 |
|  18.5 |    0.7 |             1 |
```

Imagen 10. Sentencia SELECT con GROUP BY y HAVING.

Sin contar con las agrupaciones, se puede usar también la cláusula **ORDER BY** para ordenar los resultados por campos. Generalmente la ordenación se realiza de manera ascendente y por campos consecutivamente (si especificamos más de uno). Si queremos que el orden sea descendente usaremos **DESC** al final de esta. Para que la consulta sea eficiente y no altere el rendimiento de la base de datos, un correcto uso de **ORDER BY** es que los campos que se usen para ordenar también se encuentren ordenados con respecto al índice de la tabla.

En el siguiente ejemplo queremos todos los tipos que tenemos ordenados por el identificar del tipo de ataque en primera instancia y por el nombre en segunda.

```
mysql> SELECT *
    -> FROM tipo
    -> ORDER BY id_tipo_ataque, nombre;
+---------+-----------+----------------+
| id_tipo | nombre    | id_tipo_ataque |
+---------+-----------+----------------+
|       2 | Bicho     |              1 |
|       5 | Fantasma  |              1 |
|       8 | Lucha     |              1 |
|       9 | Normal    |              1 |
|      12 | Roca      |              1 |
|      13 | Tierra    |              1 |
|      14 | Veneno    |              1 |
|      15 | Volador   |              1 |
|       1 | Agua      |              2 |
|       3 | Dragón    |              2 |
|       4 | Eléctrico |              2 |
|       6 | Fuego     |              2 |
|       7 | Hielo     |              2 |
|      10 | Planta    |              2 |
|      11 | Psíquico  |              2 |
+---------+-----------+----------------+
```

Imagen 11. Sentencia SELECT con ORDER BY.

Se puede observar que primero ha ordenado por orden numérico y después ha ordenado por orden alfabético.

5.1.4. Consulta de registros sobre varias tablas. Composiciones

Las consultas en SQL nos permiten que se puedan realizar sobre varias tablas al mismo tiempo para así obtener un resultado más preciso o concreto. Esta operación se llama composición o JOIN y sigue la siguiente estructura:

```
SELECT [DISTINCT] campos
        FROM tabla1, tabla2, …, tablaN
        WHERE condición_relación
```

En el siguiente ejemplo, seleccionamos el nombre de los Pokémon y el identificador del tipo, que se encuentran en dos tablas distintas:

```
mysql> SELECT nombre, id_tipo
    -> FROM pokemon, pokemon_tipo
    -> WHERE pokemon.numero_pokedex = pokemon_tipo.numero_pokedex;
+------------+---------+
| nombre     | id_tipo |
+------------+---------+
| Bulbasaur  |      10 |
| Bulbasaur  |      14 |
| Ivysaur    |      10 |
| Ivysaur    |      14 |
| Venasaur   |      10 |
| Venasaur   |      14 |
| Charmander |       6 |
| Charmeleon |       6 |
| Charizard  |       6 |
| Charizard  |      15 |
| Squirtle   |       1 |
| Wartortle  |       1 |
| Blastoise  |       1 |
| Caterpie   |       2 |
| Metapod    |       2 |
| Butterfree |       2 |
| Butterfree |      15 |
```

Imagen 12. JOIN de dos tablas.

Es importante deducir, como se debe de presuponer, que para poder realizar correctamente estas comparativas es necesario conocer el diagrama de la base de datos, pues solo así podremos saber que campos establecen las relaciones.

Si nos fijamos, vemos que hemos usado la siguiente estructura en algunos campos: **tabla.campo**. Esto se realiza para distinguir campos con el mismo nombre en distintas tablas.

Podemos realizar el mismo ejemplo, pero con tres tablas, de modo que ahora solo nos muestre los nombres de los Pokémon y el nombre del tipo.

```
mysql> SELECT pokemon.nombre, tipo.nombre
    -> FROM pokemon, pokemon_tipo, tipo
    -> WHERE pokemon.numero_pokedex = pokemon_tipo.numero_pokedex
    -> AND pokemon_tipo.id_tipo = tipo.id_tipo;
+------------+------------+
| nombre     | nombre     |
+------------+------------+
| Squirtle   | Agua       |
| Wartortle  | Agua       |
| Blastoise  | Agua       |
| Psyduck    | Agua       |
| Golduck    | Agua       |
| Poliwag    | Agua       |
| Poliwhirl  | Agua       |
| Poliwrath  | Agua       |
| Tentacool  | Agua       |
| Tentacruel | Agua       |
| Slowpoke   | Agua       |
| Slowbro    | Agua       |
| Seel       | Agua       |
| Dewgong    | Agua       |
| Shellder   | Agua       |
| Cloyster   | Agua       |
| Krabby     | Agua       |
| Kingler    | Agua       |
| Horsea     | Agua       |
| Seadra     | Agua       |
```

Imagen 13. JOIN de tres tablas.

Como podemos ver, se presentan muchas duplicidades porque los Pokémon en este caso pueden tener hasta dos tipos, y se nos muestran a muchos de ellos repetidos en la selección y desordenados. Para que esto no ocurra, ordenamos como hemos visto anteriormente por el nombre de los Pokémon.

```
mysql>
mysql> SELECT pokemon.nombre, tipo.nombre
    -> FROM pokemon, pokemon_tipo, tipo
    -> WHERE pokemon.numero_pokedex = pokemon_tipo.numero_pokedex
    -> AND pokemon_tipo.id_tipo = tipo.id_tipo
    -> ORDER BY pokemon.nombre;
+------------+------------+
| nombre     | nombre     |
+------------+------------+
| Abra       | Psíquico   |
| Aerodactyl | Volador    |
| Aerodactyl | Roca       |
| Alakazam   | Tierra     |
| Arbok      | Veneno     |
| Arcanine   | Fuego      |
| Articuno   | Volador    |
| Articuno   | Hielo      |
| Beedrill   | Bicho      |
| Beedrill   | Veneno     |
| Bellsprout | Planta     |
| Bellsprout | Veneno     |
| Blastoise  | Agua       |
| Bulbasaur  | Veneno     |
| Bulbasaur  | Planta     |
| Butterfree | Volador    |
| Butterfree | Bicho      |
| Caterpie   | Bicho      |
| Chansey    | Normal     |
| Charizard  | Fuego      |
```

Imagen 14. JOIN con ORDER BY.

Más adelante en SQL, con el estándar SQL-92 se incluyeron nuevas clausulas para realizar **JOIN**. Esta sigue la estructura **INNER JOIN**. Vamos a ver el ejemplo anterior de dos tablas con dicha estructura:

```
mysql> SELECT pokemon.nombre, tipo.nombre
    -> FROM pokemon
    -> INNER JOIN pokemon_tipo
    -> ON pokemon.numero_pokedex = pokemon_tipo.numero_pokedex
    -> INNER JOIN tipo
    -> ON pokemon_tipo.id_tipo = tipo.id_tipo;
+------------+------------+
| nombre     | nombre     |
+------------+------------+
| Squirtle   | Agua       |
| Wartortle  | Agua       |
| Blastoise  | Agua       |
| Psyduck    | Agua       |
| Golduck    | Agua       |
| Poliwag    | Agua       |
| Poliwhirl  | Agua       |
| Poliwrath  | Agua       |
| Tentacool  | Agua       |
| Tentacruel | Agua       |
| Slowpoke   | Agua       |
| Slowbro    | Agua       |
| Seel       | Agua       |
| Dewgong    | Agua       |
| Shellder   | Agua       |
| Cloyster   | Agua       |
| Krabby     | Agua       |
| Kingler    | Agua       |
| Horsea     | Agua       |
| Seadra     | Agua       |
```

Imagen 15. INNER JOIN.

Con esta implementación, se resolvió también un problema que se presentaba anteriormente, que era que los registros que no tuvieran relación quedaban excluidos.

Para estas nuevas cláusulas también existe **OUTER JOIN** que nos permitirá la inclusión de los registros que se quedan fuera de la comparación. Existen tres subtipos:

- **LEFT (OUTER) JOIN.** En el resultado se incluirán los registros de ala tabla de la izquierda que no tengan relación con registros de la tabla de la derecha,
- **RIGHT (OUTER) JOIN.** Se mostrarán por pantalla los registros resultados de la comparación y también los registros de la tabla de la derecha que no tengan relación con los de la otra tabla.
- **FULL (OUTER) JOIN.** Se incluyen todos los registros de ambas tablas, tengan o no relación entre sí.

5.1.5. Consulta de registros sobre varias tablas. Combinación de consultas

Podemos combinan el resultado de distintas consultas de SQL independiente una de la otra usando los siguientes operadores:

- **UNION [ALL].** Se unen todos los resultados de ambas consultas y se muestran seguidos. Los resultados que sean repetidos se eliminarán por defecto a no ser que se use la opción **ALL**.

En el siguiente ejemplo vamos a listar los nombres de los Pokémon y los identificadores de los tipos.

```
mysql> SELECT nombre
    -> FROM pokemon
    -> UNION
    -> SELECT id_tipo
    -> FROM pokemon_tipo;
+------------+
| nombre     |
+------------+
| Bulbasaur  |
| Ivysaur    |
| Venasaur   |
| Charmander |
| Charmeleon |
| Charizard  |
| Squirtle   |
| Wartortle  |
| Blastoise  |
| Caperpie   |
| Metapod    |
| Butterfree |
| Weedle     |
| Kakuna     |
| Beedrill   |
| Pidgey     |
| Pidgeotto  |
```

Imagen 16. UNION.

```
| Aerodactyl |
| Snorlax    |
| Articuno   |
| Zapdos     |
| Moltres    |
| Dratini    |
| Dragonair  |
| Dragonite  |
| Mewtwo     |
| Mew        |
| 1          |
| 2          |
| 3          |
| 4          |
| 5          |
| 6          |
| 7          |
| 8          |
| 9          |
| 10         |
| 11         |
| 12         |
| 13         |
| 14         |
| 15         |
+------------+
166 rows in set (0,01 sec)
```

Imagen 17. Final de la consulta.

Podemos comprobar que solo aparece la opción nombre, ya que esto es ideal para campos que usen el mismo nombre en distintas tablas o distintas bases de datos.

Es importante tener en cuenta que los campos seleccionados de las consultas deben de ser el mismo número, no se puede seleccionar un campo en la primera consulta y 3 en la segunda, por ejemplo.

- **MINUS.** Se muestran todos los datos que muestra la primera consulta, pero quitando los que coincidan con la segunda.
- **INTERSECT.** Se muestran todos los datos que serían resultado a ambas consultas, es decir, coincidentes.

5.1.6. Subconsultas

También podemos usar en SQL las subconsultas, que se trata de realizar una consulta con comparación, pero el dato usado para la comparación viene devuelto por otra consulta.

No existe un número limitado de consultas que ir anidando, estas son infinitas, pero claro, cuantas más consultas, mayor es la complejidad. Se puede realizar esta operación usando los siguientes operadores:

- **Operadores de comparación.** La subconsulta devuelve un único valor, o si no, nos dará error.

En el siguiente ejemplo, queremos el nombre del tipo del Pokémon Squirtle,

```
mysql> SELECT nombre
    -> FROM tipo
    -> WHERE id_tipo = (SELECT id_tipo
    -> FROM pokemon_tipo, pokemon
    -> WHERE pokemon_tipo.numero_pokedex = pokemon.numero_pokedex
    -> AND nombre = "Squirtle");
+--------+
| nombre |
+--------+
| Agua   |
+--------+
1 row in set (0,00 sec)
```

Imagen 18. Subconsultas.

Podemos ver que se ha invertido el orden de las consultas a conforme lo estábamos realizando a lo largo de la unidad, pero es que esto es una cosa que se puede hacer sin ningún problema.

Además, si nos fijamos, también hemos usado JOIN en la subconsulta, esto tampoco es un mayor problema, pues una subconsulta es a efectos una consulta normal y corriente.

- **IN.** Se comprueba que el valor se encuentre en unos cuantos valores obtenidos en la subconsulta y entonces se devuelve el resultado.

En el ejemplo de a continuación, podemos ver que lo que solicitamos es el nombre de todos los Pokémon de tipo agua.

```
mysql> SELECT nombre
    -> FROM pokemon
    -> WHERE numero_pokedex in (SELECT numero_pokedex
    -> FROM pokemon_tipo, tipo
    -> WHERE pokemon_tipo.id_tipo = tipo.id_tipo
    -> AND nombre = "AGUA");
+------------+
| nombre     |
+------------+
| Squirtle   |
| Wartortle  |
| Blastoise  |
| Psyduck    |
| Golduck    |
| Poliwag    |
| Poliwhirl  |
| Poliwrath  |
| Tentacool  |
| Tentacruel |
| Slowpoke   |
| Slowbro    |
| Seel       |
| Dewgong    |
| Shellder   |
| Cloyster   |
| Krabby     |
| Kingler    |
```

Imagen 19. Subconsultas con IN.

Si nos fijamos, vemos que el operador **IN** si que permite que haya varios resultados de la consulta, pues el número de pokédex de los Pokémon de tipo agua, es superior a uno. Aquí también se podrían descartar problemas con los valores nulos.

El uso de operadores de comparación o del operador IN lo decidirá el usuario que ejecute la consulta dependiendo siempre del número de resultados que desea que esta proporcione.

El uso de operadores de comparación o del operador IN lo decidirá el usuario que ejecute la consulta dependiendo siempre del número de resultados que desea que esta proporcione.

5.1.7. Alias

Se pueden asignar alias a los campos para que los resultados se expresen de otro modo,

Para realizar esta acción usamos el operador **AS** después del campo y seguido del nombre que queremos ponerle entrecomillado.

Por ejemplo, si queremos mostrar todos lo Pokémon pero que el campo nombre use el término Pokémon.

```
mysql> SELECT nombre AS "Pokémon"
    -> FROM pokemon;
+------------+
| Pokémon    |
+------------+
| Bulbasaur  |
| Ivysaur    |
| Venasaur   |
| Charmander |
| Charmeleon |
| Charizard  |
| Squirtle   |
| Wartortle  |
| Blastoise  |
| Caperpie   |
| Metapod    |
| Butterfree |
| Weedle     |
| Kakuna     |
| Beedrill   |
| Pidgey     |
| Pidgeotto  |
| Pidgeot    |
| Rattata    |
| Raticate   |
| Spearow    |
| Fearow     |
| Ekans      |
```

Imagen 20. Alias.

Como podemos ver, no han cambiado los resultados de la consulta, simplemente el nombre del campo, pero solo para la visualización.

Se pueden usar concatenaciones también para diferentes campos situados en diferentes tablas y con usos de JOIN. Por ejemplo, el mismo de antes, pero añadiendo el tipo:

```
mysql> SELECT pokemon.nombre AS "Pokémon" , tipo.nombre AS "Tipo"
    -> FROM pokemon, pokemon_tipo, tipo
    -> WHERE pokemon.numero_pokedex = pokemon_tipo.numero_pokedex
    -> AND pokemon_tipo.id_tipo = tipo.id_tipo
    -> ORDER BY pokemon.nombre;
+------------+------------+
| Pokémon    | Tipo       |
+------------+------------+
| Abra       | Psíquico   |
| Aerodactyl | Volador    |
| Aerodactyl | Roca       |
| Alakazam   | Tierra     |
| Arbok      | Veneno     |
| Arcanine   | Fuego      |
| Articuno   | Volador    |
| Articuno   | Hielo      |
| Beedrill   | Bicho      |
| Beedrill   | Veneno     |
| Bellsprout | Planta     |
| Bellsprout | Veneno     |
| Blastoise  | Agua       |
| Bulbasaur  | Veneno     |
| Bulbasaur  | Planta     |
| Butterfree | Volador    |
| Butterfree | Bicho      |
| Caperpie   | Bicho      |
| Chansey    | Normal     |
| Charizard  | Fuego      |
```

Imagen 21. Varios Alias.

5.1.8. Funciones integradas

No solo podemos usar expresiones aritméticas o de concatenación en los SGBD, sino que también tenemos una serie de funciones integradas (complementarios a las de agregación), que nos facilitan el uso de los datos obtenidos de la consulta.

Las siguientes funciones son las más usadas.

Función	Parámetros	Valor devuelto
ABS	Número	Valor absoluto del número
MOD	Número 1, Número 2	Módulo o resto de la división del primero número entre el segundo.
POSITION	Cadena de caracteres 1, Cadena de caracteres 2	Posición de la primera cadena dentro de la segunda.
CHAR_LENGTH	Cadena de caracteres	Tamaño de la cadena.
SUBSTRING	Cadena de caracteres, Posición de inicio, Número de caracteres	Cadena de caracteres dentro de la cadena recibida como parámetro. El segundo de los parámetros define la posición de la cadena original a partir de la que se extraerá de la cadena que será devuelta. El tercer parámetro será el número de caracteres de la cadena resultante.
UPPER	Cadena de caracteres	Cadena de caracteres original en mayúsculas.
LOWER	Cadena de caracteres	Cadena de caracteres original en minúsculas.

Imagen 22. Funciones de SQL.

5.1.9. Inserción de registros

> **Campo a campo:** solo se va a insertar un único registro.

```
INSERT INTO tabla [ (campo1[, campo2, …, campoN])]
  VALUES (valor1[, valor2, …, valorN]);
```

En el siguiente ejemplo, vamos a añadir un nuevo Pokémon a nuestra tabla.

```
mysql> INSERT INTO pokemon
    -> (numero_pokedex, nombre, peso, altura)
    -> VALUES
    -> ("152", "Chikorita", "6", "1");
Query OK, 1 row affected (0,04 sec)

mysql> SELECT *
    -> FROM pokemon
    -> WHERE numero_pokedex = 152;
+---------------+-----------+------+--------+
| numero_pokedex | nombre    | peso | altura |
+---------------+-----------+------+--------+
|            152 | Chikorita |    6 |      1 |
+---------------+-----------+------+--------+
1 row in set (0,00 sec)
```

Imagen 23. Sentencia INSERT 1.

La lista de campos es opcional, y si no se pone, hace referencia a los campos de toda la tabla, pero nunca está demás y es recomendable especificarlos.

También podemos ver que hemos entrecomillado los valores, pero realmente eso solo es necesario en las cadenas de caracteres y no en valores definido que tienen un formato, como los integer.

El siguiente ejemplo añade otro registro más pero ahora sin entrecomillar los números que no son necesarios.

```
mysql> INSERT INTO pokemon
    -> (numero_pokedex, nombre, peso, altura)
    -> VALUES
    -> (153, "Bayleef", 16, 1);
Query OK, 1 row affected (0,01 sec)

mysql> SELECT *
    -> FROM pokemon
    -> WHERE numero_pokedex = 153;
+----------------+---------+------+--------+
| numero_pokedex | nombre  | peso | altura |
+----------------+---------+------+--------+
|            153 | Bayleef |   16 |      1 |
+----------------+---------+------+--------+
1 row in set (0,01 sec)
```

Imagen 24. Sentencia INSERT 2.

> Partiendo de una consulta. Se pueden insertar valores que se hayan obtenido directamente de una consulta.

```
INSERT INTO tabla [(campo1[, campo2, …, campoN])]
  consulta;
```

5.1.10. Modificación de registros

Para poder modificar un registro contamos con la sentencia **UPDATE** que permite que se actualicen los campos concretos de registros que cumplan con cierta condición. La condición es como en cualquier clausula **WHERE**, y si no se especifica, se actualizan todos los valores del campo.

Su estructura es la siguiente:

```
UPDATE tabla
        SET campo1 = valor1[.
        campo2 = campo2,
        …
        campoN = valorN]
        [WHERE condición];
```

En el siguiente ejemplo se modifica el campo identificador de uno de los registros que añadimos anteriormente:

```
mysql> UPDATE pokemon
    -> SET numero_pokedex = 154
    -> WHERE numero_pokedex = 153;
Query OK, 1 row affected (0,04 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Imagen 25. Sentencia UPDATE.

Si hacemos las correspondientes comprobaciones….

```
mysql> SELECT *
    -> FROM pokemon
    -> WHERE numero_pokedex = 153;
Empty set (0,00 sec)

mysql> SELECT *
    -> FROM pokemon
    -> WHERE numero_pokedex = 154;
+----------------+---------+------+--------+
| numero_pokedex | nombre  | peso | altura |
+----------------+---------+------+--------+
|            154 | Bayleef |   16 |      1 |
+----------------+---------+------+--------+
```

Imagen 26. Comprobación de UPDATE.

Si se usa un criterio menos restrictivo, puede que afecte a varios registros.

Y, además, se pueden incluir varias condiciones, como podemos ver en el siguiente ejemplo donde modificamos la altura en un rango concreto;

```
mysql> UPDATE pokemon
    -> SET altura = 1
    -> WHERE altura BETWEEN 0.8 AND 1.2;
Query OK, 32 rows affected (0,04 sec)
Rows matched: 54  Changed: 32  Warnings: 0

mysql> SELECT *
    -> FROM pokemon
    -> WHERE altura = 1;
+----------------+------------+------+--------+
| numero_pokedex | nombre     | peso | altura |
+----------------+------------+------+--------+
|              2 | Ivysaur    |   13 |      1 |
|              5 | Charmeleon |   19 |      1 |
|              8 | Wartortle  | 22.5 |      1 |
|             12 | Butterfree |   32 |      1 |
|             15 | Beedrill   | 29.5 |      1 |
|             17 | Pidgeotto  |   30 |      1 |
|             22 | Fearow     |   38 |      1 |
|             26 | Raichu     |   30 |      1 |
|             28 | Sandslash  | 29.5 |      1 |
|             30 | Nidorina   |   20 |      1 |
|             33 | Nidorino   | 19.5 |      1 |
|             38 | Ninetales  | 19.9 |      1 |
|             40 | Wigglytuff |   12 |      1 |
|             41 | Zubat      |  7.5 |      1 |
|             44 | Gloom      |  8.6 |      1 |
|             45 | Vileplume  | 18.6 |      1 |
```

Imagen 27. UPDATE con condición 1.

Por otro lado, se pueden actualizar varios campos a la vez como podemos ver en este ejemplo en el que se modifica el identificador y la altura de Bayleef.

```
mysql> UPDATE pokemon
    -> SET numero_pokedex = 153,
    -> altura = 1.2
    -> WHERE nombre = "Bayleef";
Query OK, 1 row affected (0,02 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT *
    -> FROM pokemon
    -> WHERE nombre = "Bayleef";
+----------------+---------+------+--------+
| numero_pokedex | nombre  | peso | altura |
+----------------+---------+------+--------+
|            153 | Bayleef |   16 |    1.2 |
+----------------+---------+------+--------+
1 row in set (0,00 sec)
```

Imagen 28. UPDATE con condición 2.

Debemos de tener en cuenta que, si modificamos los registros de una tabla, también debemos de modificar los de las tablas con las que se relaciona o si no, la base de datos no sería eficiente. Lo mismo es válido para la inserción y la eliminación de registros.

5.1.11. Eliminación de registros

La sentencia DELETE tiene una estructura parecida a UPDATE, pero en este caso su sintaxis es la siguiente:

```
DELETE FROM tabla
        [WHERE condición];
```

Si no se especifica una condición, se borrarán todos los registros de la tabla.

```
mysql> DELETE FROM pokemon
    -> WHERE numero_pokedex = 153;
Query OK, 1 row affected (0,05 sec)

mysql> SELECT *
    -> FROM pokemon
    -> WHERE numero_pokedex > 152;
Empty set (0,00 sec)
```

Imagen 29. Sentencia DELETE.

Se pueden usar en todas las sentencias las subconsultas a la hora de actualizar o borrar registros.

Hay que llevar especial cuidado y atención en el borrado de claves primarias y foráneas, así como en su actualización.

Cabe destacar, por último, que el borrado se hace de todo el registro y no de solo un campo de este.

5.2. DDL. Definición de Vistas

Se pueden usar las consultas SELECT simplificadas o con nombres distintos, usando las vistas, que funcionan de forma similar a los alias.

Creación de una vista:

```
CREATE VIEW vista (campos)
        AS consulta;
```

Este uso de vistas hace que cuando ejecutemos una consulta sobre la vista, se ejecute realmente sobre los datos obtenidos en esa consulta contenida en la vista y se visualice con los nombres solicitados en la creación de la vista.

Aquí tenemos un ejemplo básico de creación de vistas:

```
mysql> CREATE VIEW vPokemons (Pokedex, Pokemon, Peso, Altura)
    -> AS
    -> SELECT *
    -> FROM pokemon;
Query OK, 0 rows affected (0,05 sec)

mysql> SELECT *
    -> FROM vPokemons;
+---------+------------+-------+--------+
| Pokedex | Pokemon    | Peso  | Altura |
+---------+------------+-------+--------+
|       1 | Bulbasaur  |   6.9 |    0.7 |
|       2 | Ivysaur    |    13 |      1 |
|       3 | Venasaur   |   100 |      2 |
|       4 | Charmander |   8.5 |    0.6 |
|       5 | Charmeleon |    19 |      1 |
|       6 | Charizard  |  90.5 |    1.7 |
|       7 | Squirtle   |     9 |    0.5 |
|       8 | Wartortle  |  22.5 |      1 |
|       9 | Blastoise  |  85.5 |    1.6 |
|      10 | Caperpie   |   2.9 |    0.3 |
|      11 | Metapod    |   9.9 |    0.7 |
|      12 | Butterfree |    32 |      1 |
|      13 | Weedle     |   3.2 |    0.3 |
|      14 | Kakuna     |    10 |    0.6 |
|      15 | Beedrill   |  29.5 |      1 |
|      16 | Pidgey     |   1.8 |    0.3 |
|      17 | Pidgeotto  |    30 |      1 |
|      18 | Pidgeot    |  39.5 |    1.5 |
```

Imagen 30. CREATE VIEW.

Borrado de una vista:

```
DROP VIEW vista;
```

En el siguiente ejemplo borramos la anterior vista y comprobamos como ya no funciona:

```
mysql> DROP view vPokemons;
Query OK, 0 rows affected (0,02 sec)

mysql> SELECT *
    -> FROM vPokemons;
ERROR 1146 (42S02): Table 'pokemondb.vPokemons' doesn't exist
mysql>
```

Imagen 31. DELETE VIEW.

5.3. Lenguaje de Control de Datos (DCL)

El último de los lenguajes que nos quedaba por explicar es el de control de datos. Este lenguaje se basa en la asignación de permisos y privilegios de seguridad los usuarios sobre ciertos objetos de la base de datos. Estos también permiten que se gestionen las transacciones de la base de datos.

5.3.1. Control de acceso a los datos

- **Concesión de privilegios.** Son los permisos que se pueden conceder de cara a los objetos de la base de datos, desde una tabla a toda la base de datos. Los permisos que se almacenen en las tablas pueden ser:
  - SELECT
  - INSERT
  - UPDATE
  - DELETE

  Y si usamos la opción WITH GRANT OPTION, damos permiso al usuario para que también pueda otorgar dichos privilegios.

  La sintaxis de la sentencia es;

```
GRANT [CONNECT|RESOURCE|DBA|ALL PRIVILEGES|SELECT|UPDATE|INSERT|DELETE]
   ON objeto
   TO usuarios
            [WITH GRANT OPTION];
```

- **Revocación de privilegios.** Parecido a la sentencia anterior, pero con efecto contrario:

```
REVOKE [CONNECT|RESOURCE|DBA|ALL PRIVILEGES|SELECT|UPDATE|INSERT|DELETE]
   FROM usuarios
   [ON objetos];
```

Cuando revoquemos privilegios no es necesario que indiquemos los objetos a los que se los quitamos siempre, si no se indican, se eliminarán todos.

Tampoco se incluye el permiso de otorgación en la revocación de privilegios.

www.universae.com