CONSULTAS DEL LENGUAJE SQL

Insertar registros en la tabla

Para insertar datos en una tabla utilizamos la orden INSERT. Con ella podemos ir añadiendo registros uno a uno, o añadir de golpe tantos registros como deseemos.

La sintaxis genérica de INSERT para crear un nuevo registro es la siguiente:

```
INSERT INTO NombreTabla (Columna1, ..., ColumnaX) VALUES (Valor1, ..., ValorX)
```

… siendo ...

- **NombreTabla:** la tabla en la que se van a insertar las filas o registros.
- **(Campo1, …, CampoX):** representa el campo o campos en los que vamos a introducir valores.
- **(Valor1, …, ValorX):** representan los valores que se van a almacenar en cada campo.

Ejemplo:

```
INSERT INTO Productos (Nombre, Precio, Marca, Categoria, Presentacion, Stock, Disponible) VALUES
('iPhone 5', 499.99, 'Apple', 'Smartphone', '16GB', 500, false);
```

Los campos deben coincidir exactamente igual a como han sido definidos mediante el comando **CREATE**.

Los valores se deben corresponder con cada uno de los campos que aparecen en la lista de campos, tanto en el tipo de dato que contienen como en el orden en el que se van a asignar. Es decir, si se indican una serie de campos en un orden determinado, la lista de valores debe especificar los valores a almacenar en dichos campos, en el mismo orden exactamente. Si un campo no está en la lista, se almacenará dentro de éste el valor **NULL**.

Valores NULL

La expresión **NULL** significa "dato desconocido" o "valor inexistente". No es lo mismo que un valor 0, una cadena vacía o una cadena de texto literal con la palabra **NULL**.

A veces, puede desconocerse o no existir el dato correspondiente a algún campo de un registro. En estos casos decimos que el campo puede contener valores nulos. Por ejemplo, en la tabla de Productos se puede tener valores nulos en el campo Precio porque es posible que para algunos productos no se haya establecido el precio para la venta.

En contraposición, tenemos campos que no pueden estar vacíos jamás, por ejemplo, los campos que identifican cada registro, como los códigos de identificación, que son clave primaria.

Por defecto, es decir, si no lo aclaramos en la creación de la tabla, los campos permiten valores nulos.

Por ejemplo:

```
INSERT INTO Productos (Nombre, Precio, Marca, Categoria, Presentacion, Stock) VALUES ('iPhone
7S', NULL, 'Apple', 'Smartphone', '16GB', 500);
```

Nótese que para el campo Precio el valor **NULL** no es una cadena de caracteres, por lo que no se coloca entre comillas. También si un campo acepta valores nulos, podemos ingresar **NULL** cuando no conocemos el valor.

Además, si una columna fue definida en el comando **CREATE** con la especificación **NULL** (como ser en el campo Disponible), puede no incluirse en el listado de campos (puede entenderse como que el campo "no es obligatorio" de asignar un valor).

Seleccionar registros de una tabla

La sentencia **SELECT** va a permitir realizar operaciones de selección, ordenación, agrupación y filtrado de registros, es decir:

- Se utiliza para seleccionar información registrada en una tabla.
- Para seleccionar todas la columnas se utiliza el * (asterisco)
- La cláusula WHERE se utiliza para establecer un criterio de búsqueda

Sintaxis #1: Seleccionar de la tabla **Productos** los valores de todas las columnas de todos los registros

```
SELECT * FROM Productos;
```

Sintaxis #2: Seleccionar de la tabla **Productos** los valores de la columna **Nombre** de todos los registros

```
SELECT Nombre FROM Productos;
```

Sintaxis #3: Seleccionar de la tabla **Productos** todos los valores de la columna **Nombre** y **Precio** de todos los registros

```
SELECT Nombre, Precio FROM Productos;
```

Sintaxis #4: Seleccionar de la tabla **Productos** la columna **Nombre** del registro cuyo valor de la columna **idProducto** es **igual a 1**

```
SELECT Nombre FROM Productos WHERE idProducto =
1;
```

Sintaxis #5: Seleccionar de la tabla **Productos** las columnas **Nombre** y **Precio** los registros cuyo valor de la columna **Precio** sea **Mayor a 150**

```
SELECT Nombre, Precio FROM Productos WHERE Precio > 150;
```

Condiciones Lógicas

Para crear expresiones lógicas disponemos de varios operadores de comparación. Estos operadores se aplican a cualquier tipo de columna: fechas, cadenas, números, etc, y devuelven valores lógicos: verdadero o falso (o bien 1 ó 0).

Si uno o los dos valores a comparar son **NULL**, el resultado es **NULL**, excepto con el operador <=> que es usado para una comparación con **NULL** segura.

El operador <=> funciona igual que el operador =, salvo que si en la comparación una o ambas de las expresiones es nula el resultado no es NULL. Si se comparan dos expresiones nulas, el resultado es verdadero.

Tabla de Operadores Lógicos

Operador	Descripción
=	Igual
<	Menor
>	Mayor
>=	Mayor o igual
<=	Menor o igual
<>	No es igual

Los operadores IS NULL e IS NOT NULL sirven para verificar si una expresión determinada es o no nula.

Ejemplos de aplicación:

```
SELECT * FROM Productos WHERE Precio >= 500 OR Stock >= 100;
```

```
SELECT Nombre, Presentacion, Precio, Stock FROM Productos WHERE Stock < 20 AND Precio >= 100;
```

```
SELECT Nombre, Presentacion FROM Productos WHERE Precio IS NOT NULL;
```

Cláusulas Especiales

Sentencia BETWEEN

Entre los operadores de MySQL, hay uno para comprobar si una expresión está comprendida en un determinado rango de valores. La sintaxis es:

- **BETWEEN** mínimo **AND** máximo
- **NOT BETWEEN** mínimo **AND** máximo

Ejemplo de aplicación:

```
SELECT * FROM Productos WHERE Precio BETWEEN 100 AND 200;
```

```
SELECT * FROM Productos WHERE Precio NOT BETWEEN 100 AND 200;
```

Sentencia IN

Los operadores **IN** y **NOT IN** sirven para averiguar si el valor de una expresión determinada está dentro de un conjunto indicado

- IN (<expr1>, <expr2>, <expr3>,...)
- NOT IN (<expr1>, <expr2>, <expr3>,...)

El operador **IN** devuelve un valor verdadero si el valor de la expresión es igual a alguno de los valores especificados en la lista. El operador **NOT IN** devuelve un valor falso en el mismo caso. Por ejemplo:

```
SELECT 10 IN(2, 4, 6, 8, 10);
```

Ejemplo de aplicación:

```
SELECT * FROM Productos WHERE Nombre IN ('iPhone', 'iPad');
```

```
SELECT * FROM Producto WHERE Nombre NOT IN ('Galaxy S5', 'TV');
```

Sentencia LIKE

El operador LIKE se usa para hacer comparaciones entre cadenas y patrones. El resultado es verdadero (1) si la cadena se ajusta al patrón, y falso (0) en caso contrario. Tanto si la cadena como el patrón son NULL, el resultado es NULL.

La sintaxis es:

```
SELECT * FROM Productos LIKE <patrón> [ESCAPE 'carácter_escape']
```

Carácter	Descripción
%	Coincidencia con cualquier número de caracteres, incluso ninguno.
_	Coincidencia con un único carácter.

La comparación es independiente del tipo de los caracteres, es decir, **LIKE** no distingue mayúsculas de minúsculas, salvo que se indique lo contrario (ver operadores de casting):

Como siempre que se usan caracteres concretos para crear patrones, se presenta la dificultad de hacer comparaciones cuando se deben buscar precisamente esos caracteres concretos. Esta dificultad se suele superar mediante secuencias de escape. Si no se especifica nada en contra, el carácter que se usa para escapar es '\'. De este modo, si queremos que nuestro patrón contenga los caracteres '%' o '_', los escaparemos de este modo: '\%' y '_':

Como en cualquier otro lenguaje, los paréntesis se pueden usar para forzar el orden de la evaluación de determinadas operaciones dentro de una expresión. Cualquier expresión entre paréntesis adquiere mayor precedencia que el resto de las operaciones en el mismo nivel de paréntesis.

Ejemplo de aplicación:

```
-- Listar todos los nombres que contienen TV
SELECT * FROM Productos WHERE Nombre LIKE '%TV%';
```