PROGRAMA DE EMPREGO

FORMAWEB IV

DATA:	**Marzo 2022**
CURSO:	PROGRAMA DE EMPREGO FORMAWEB IV
MÓDULO	MF0966_3. Consulta e manipulación de información contida en xestores de datos
UNIDADE FORMATIVA	UF2214: Implementación e uso dunha base de datos
NºEXP:	36/00004/2021

Introdución

O Structured Query Language ou SQL (polas súas siglas en inglés **Structured Query Language**) é unha linguaxe declarativa para o acceso a bases de datos relacionais que permite especificar varios tipos de operacións sobre elas. Unha das súas características é a xestión de álxebra e cálculo relacional que permiten realizar consultas para poder recuperar facilmente a información de interese das bases de datos, así como realizar cambios nela.

SQL é unha linguaxe de acceso a bases de datos que aproveita a flexibilidade e o poder dos sistemas relacionais e, polo tanto, permite unha gran variedade de operacións.

Compoñentes SQL

A linguaxe SQL está formada por comandos, cláusulas, operadores e funcións agregadas. Estes elementos combínanse nas instrucións para crear, actualizar e manipular as bases de datos.

Comandos

Hai tres tipos de comandos SQL:

As **DLL (Data Definition Language)** que permiten crear e definir novas bases de datos, campos e índices. O **DML (Data Manipulation Language)** que permite xerar consultas para ordenar, filtrar e extraer datos da base de datos. Os **DCL (Data Control Language)** que se encargan de definir os permisos sobre os datos

Linguaxe de definición de datos (DDL)

Comando	Descrición
CREATE	Úsase para crear novas táboas, campos e índices
DROP	Úsase para soltar táboas e índices
ALTER	Utilízase para modificar táboas engadindo campos ou cambiando a súa definición.

A linguaxe de definición de datos (en inglés Data Definition Language, ou DDL), é a que se encarga de modificar a estrutura dos obxectos da base de datos. Inclúe comandos para modificar, eliminar ou definir as táboas nas que se almacenan os datos da base de datos. Hai catro operacións básicas: CREATE, ALTER, DROP e TRUNCATE.

CREAR

Este comando crea un obxecto dentro do xestor de bases de datos. Pode ser unha base de datos, táboa, índice, procedemento almacenado ou vista.

Exemplo (crear unha táboa):

```
CREATE TABLE Empleado (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    Nombre VARCHAR(50),
    Apellido VARCHAR(50),
    Direccion VARCHAR(255),
    Ciudad VARCHAR(60),
    Telefono VARCHAR(15),
    Peso VARCHAR (5),
    Edad (2),
    Actividad Específica (100),
    idCargo INT
)
```

ALTER

Este comando permítelle modificar a estrutura dun obxecto. Pode engadir/eliminar campos a unha táboa, modificar o tipo de campo, engadir/eliminar índices a unha táboa, modificar un activador, etc.

Exemplo (engadir columna a unha táboa):

```
# ALTER TABLE 'NOMBRE_TABLA' ADD NUEVO_CAMPO INT;
# ALTER TABLE 'NOMBRE_TABLA' DROP COLUMN NOMBRE_COLUMNA;
```

SOLAR

Este comando elimina un obxecto da base de datos. Pode ser unha táboa, vista, índice, disparador, función, procedemento ou calquera outro obxecto que admita o motor de base de datos. Pódese combinar coa instrución ALTER.

Exemplo:

```
# DROP TABLE 'NOMBRE_TABLA';
# DROP SCHEMA 'ESQUEMA;'
# DROP DATABASE 'BASEDATOS';
```

TRONCAR

Este comando trunca todo o contido dunha táboa. A vantaxe sobre o comando DROP é que se queres eliminar todo o contido da táboa, é moito máis rápido, especialmente se a táboa é moi grande. A desvantaxe é que TRUNCATE só é útil cando se quere eliminar absolutamente todos os rexistros, xa que a cláusula WHERE non está permitida. Aínda que, nun principio, esta instrución parece ser DML (Data Manipulation Language), en realidade é un DDL, xa que internamente, o comando TRUNCATE solta a táboa e recréaa e non executa ningunha transacción.

Exemplo:

```
# TRUNCATE TABLE 'NOMBRE_TABLA';
```

Linguaxe de manipulación de datos (DML)

Comando	Descrición
SELECT	Utilízase para consultar rexistros de bases de datos que cumpren un determinado criterio
INSERT	Úsase para cargar lotes de datos na base de datos nunha única operación.
UPDATE	Utilízase para modificar os valores dos campos e rexistros especificados. Utilízase para modificar as táboas engadindo campos ou cambiando a definición dos campos.
DELETE	Úsase para eliminar rexistros dunha táboa

Definición

Unha linguaxe de manipulación de datos (Data Manipulation Language, ou DML en inglés) é unha linguaxe proporcionada polo sistema de xestión de bases de datos que permite aos usuarios realizar as tarefas de consulta ou manipulación dos datos, organizados pola base de datos.modelo de datos adecuado. A linguaxe de manipulación de datos máis popular na actualidade é SQL, que se usa para recuperar e manipular datos nunha base de datos relacional.

INSERT

Unha instrución SQL INSERT engade un ou máis rexistros a unha (e só unha) táboa nunha base de datos relacional.

Forma básica:

```
# INSERT INTO ''tabla'' (''columna1'', [''columna2,... '']) VALUES
(''valor1'', [''valor2,...''])
```

O número de columnas e valores debe ser o mesmo. Se non se especifica unha columna, asignaráselle o valor predeterminado. Os valores especificados (ou implícitos) pola instrución INSERT deben cumprir todas as restricións aplicables. Se se produce un erro de sintaxe ou se infrinxe algunha das restricións, non se engade a fila e devólvese un erro.

Exemplo:

```
# INSERT INTO agenda_telefonica (nombre, numero) VALUES ('Roberto Mendez',
4886850);
```

Cando se especifican todos os valores dunha táboa, pódese usar a declaración abreviada:

```
# INSERT INTO ''VALUES (''valor1'', [''valor2,...''])
```

Exemplo (supoñendo que "nome" e "número" son as únicas columnas da táboa "axenda"):

```
# INSERT INTO agenda_telefonica VALUES ('Pedro Permuy', 080473968);
```

UPDATE

Unha instrución SQL UPDATE úsase para modificar os valores dun conxunto de rexistros existente nunha táboa.

Exemplo:

```
# UPDATE mi_tabla SET campo1 = 'nuevo valor campo1' WHERE campo2 = 'N';
```

DELETE

Unha instrución SQL DELETE elimina un ou máis rexistros existentes nunha táboa.

Forma básica:

```
# DELETE FROM 'tabla' WHERE 'columna1' = 'valor1'
```

Exemplo:

```
# DELETE FROM My_table WHERE field2 = 'N';
```

Cláusulas

As cláusulas son condicións de modificación utilizadas para definir os datos que quere seleccionar ou manipular.

Comando	Descrición
FROM	Utilízase para especificar a táboa da que se deben seleccionar os rexistros
GROUP BY	Utilízase para separar os rexistros seleccionados en grupos específicos
HAVING	Úsase para expresar unha condición que debe cumprir cada grupo
ORDER BY	Utilízase para ordenar os rexistros seleccionados segundo unha orde específica
WHERE	Utilízase para determinar os rexistros seleccionados na cláusula FROM

Operadores

Operadores lóxicos

Operador	Use
AND	É o "e" lóxico. Avalía dúas condicións e devolve un valor de verdade só se ambas son verdadeiras.
OR	É o "ou" lóxico. Avalía dúas condicións e devolve un valor verdadeiro se algunha é verdadeira.
NOT	Negación lóxica. Devolve o valor oposto da expresión.

Operadores de comparación

Operador	Use
<	Menor que
>	Máis grande cá
<>	distinto de
<=	menor ou igual a
>=	Maior ou igual
BETWEEN	Intervalo
LIKE	Comparación
IN	Especificar

Funcións de agregación

As funcións agregadas utilízanse dentro dunha cláusula SELECT en grupos de rexistros para devolver un único valor que se aplica a un grupo de rexistros.

Comando	Descrición
AVG	Úsase para calcular a media dos valores dun determinado campo
COUNT	Utilízase para devolver o número de rexistros na selección
SUM	Úsase para devolver a suma de todos os valores dun campo dado
MAX	Utilízase para devolver o valor máis alto dun campo especificado
MIN	Utilízase para devolver o valor máis baixo dun campo especificado

Consultas

Consultas de selección

As consultas de selección úsanse para indicarlle ao motor de datos que devolva información das bases de datos, esta información devólvese en forma de conxunto de rexistros. Este conxunto de rexistros é escribible.

Básico

A sintaxe básica dunha consulta de selección é:

```
# SELECT Campos FROM Tabla;
# SELECT Nombre, Telefono FROM Clientes;
```

Ordenar rexistros

Podes especificar a orde na que queres recuperar os rexistros das táboas usando a cláusula **ORDER BY** :

```
# SELECT CodigoPostal, Nombre, Telefono FROM Clientes ORDER BY Nombre;
```

Os rexistros pódense ordenar por máis dun campo:

```
# SELECT CodigoPostal, Nombre, Telefono FROM Clientes ORDER BY CodigoPostal,
Nombre;
```

E pode especificar a orde dos rexistros: ascendente a través da cláusula (**ASC** - este valor tómase por defecto) ou descendente (**DESC**):

```
# SELECT CodigoPostal, Nombre, Telefono FROM Clientes ORDER BY CodigoPostal
DESC , Nombre ASC;
```

Consultas con predicado

1. ALL Se non se inclúe ningún dos predicados, asúmese ALL. O motor de base de datos selecciona todos os rexistros que cumpren as condicións da instrución SQL:

   ```
   # SELECT ALL FROM Empleados;
   # SELECT * FROM Empleados;
   ```

2. TOP Devolve un determinado número de rexistros que se sitúan entre o principio ou o final dun intervalo especificado por unha cláusula ORDER BY. Supoñamos que queremos recuperar os nomes dos primeiros 25 alumnos do curso de 1994:

   ```
   # SELECT TOP 25 Nombre, Apellido FROM Estudiantes ORDER BY Nota DESC;
   ```

   Se non se inclúe la cláusula ORDER BY, a consulta devolverá un conxunto arbitrario de 25 rexistros da táboa Estudiantes . O predicado TOP non elixe entre valores iguais. No exemplo anterior, se a nota media número 25 e a 26 son iguais, a consulta devolverá 26 rexistros. Se pode utilizar a palabra reservada PERCENT para devolver un certo porcentaxe de rexistros que caen ao principio ou o final dun rango especificado pola cláusula ORDER BY. Supoñamos que en lugar dos 25 primeiros estudiantes queremos o 10 por cento dol curso:

   ```
   # SELECT TOP 10 PERCENT Nombre, Apellido FROM Estudiantes ORDER BY Nota
   DESC;
   ```

3. DISTINCT Salta os rexistros que conteñen datos duplicados nos campos seleccionados. Para que os valores de cada campo enumerados na instrución SELECT se inclúan na consulta, deben ser únicos:

   ```
   # SELECT DISTINCT Apellido FROM Empleados;
   ```

4. DISTINCTROW Devolve os diferentes rexistros dunha táboa; A diferenza do predicado anterior que só miraba o contido dos campos seleccionados, este mira o contido de todo o rexistro independentemente dos campos indicados na cláusula SELECT:

```
# SELECT DISTINCTROW Apellido FROM Empleados;
```

Criterios de selección

Operadores lóxicos

Os operadores lóxicos soportados por SQL son:

> **AND, OR, XOR, Eqv, Imp, Is** e **Not.**

Excepto os dous últimos, todos teñen a seguinte sintaxe:

```
<expresión1> operador <expresión2>
```

Onde expresión1 e expresión2 son as condicións a avaliar, o resultado da operación varía dependendo do operador lóxico:

```
# SELECT * FROM Empleados WHERE Edad > 25 AND Edad < 50;
# SELECT * FROM Empleados WHERE (Edad > 25 AND Edad < 50) OR Sueldo = 100;
# SELECT * FROM Empleados WHERE NOT Estado = 'Soltero';
# SELECT * FROM Empleados WHERE (Sueldo > 100 AND Sueldo < 500) OR (Provincia
= 'Madrid' AND Estado = 'Casado');
```

operador BEWEEN

Para indicar que queremos recuperar os rexistros segundo o intervalo de valores dun campo, utilizaremos o operador **Entre** :

```
# SELECT * FROM Pedidos WHERE CodPostal Between 28000 And 28999;
// (Devolve os pedidos realizados na provincia de Madrid)

# SELECT IIf(CodPostal Between 28000 And 28999, 'Provincial', 'Nacional')
FROM Editores;
// (Devolve o valor 'Provincial' se o código postal se atopa no intervalo, do
contrario 'Nacional')
```

operador LIKE

Utilízase para comparar unha expresión de cadea cun patrón nunha expresión SQL. A súa sintaxe é:

```
expresión LIKE modelo
```

operador IN

Este operador devolve aqueles rexistros cuxo campo indicado coincide cun dos indicados nunha lista. A súa sintaxe é:

```
// expresión [Not] In(valor1, valor2, . . .)

# SELECT * FROM Pedidos WHERE Provincia In ('Madrid', 'Barcelona',
'Sevilla');
```

cláusula WHERE

A cláusula WHERE pódese usar para determinar que rexistros das táboas listadas na cláusula FROM aparecerán nos resultados da instrución SELECT. WHERE é opcional, pero cando apareza debe seguir FROM:

```
# SELECT Apellidos, Salario FROM Empleados
WHERE Salario > 21000;
# SELECT Id_Producto, Existencias FROM Productos
WHERE Existencias <= Nuevo_Pedido;
```

Agrupación de rexistros (agregación)

AVG

Calcula a media aritmética dun conxunto de valores contidos nun campo especificado dunha consulta:

```
AVG(expr)
```

A función Avg non inclúe ningún campo Nulo no cálculo. Un exemplo de como funciona **AVG** :

```
# SELECT AVG(Gastos) AS Promedio FROM Pedidos WHERE Gastos > 100;
```

MAX, MIN

Devolven o mínimo ou o máximo dun conxunto de valores contidos nun campo específico dunha consulta. A súa sintaxe é:

```
MIN ( expr )
MAX ( expr )
```

Un exemplo do seu uso:

```
# SELECT Min(Gastos) AS ElMin FROM Pedidos
WHERE Pais = 'Francia';
# SELECT Max(Gastos) AS ElMax FROM Pedidos
WHERE Pais = 'Francia';
```

SUM

Devolve a suma do conxunto de valores contidos nun campo específico dunha consulta. A súa sintaxe é:

```
SUM ( expr )
```

Por exemplo:

```
# SELECCIONAR Suma (Prezo unitario * Cantidade) AS Total FROM OrderDetail;
```

GROUP BY

Combina os rexistros con valores idénticos, na lista de campos especificados, nun único rexistro:

```
# SELECT campos FROM tabla WHERE criterio GROUP BY campos del grupo
```

Todos os campos da lista de campos SELECT deben incluírse na cláusula GROUP BY ou como argumentos para unha función agregada de SQL:

```
# SELECT Id_Familia, Sum(Stock) FROM Productos GROUP BY Id_Familia;
```

HAVING é semellante a WHERE, determina que rexistros se seleccionan. Unha vez agrupados os rexistros mediante GROUP BY, HAVING determina cal deles mostrar.

```
# SELECT Id_Familia Sum(Stock) FROM Productos GROUP BY Id_Familia HAVING
Sum(Stock) > 100 AND NombreProducto Like BOS*;
```

Manexando varias táboas

Partindo da definición das seguintes táboas:

1. **Mesa de clientes**

```
+------+--------+----------+
| cit  |  nome   | teléfono |
+------+--------+----------+
|  1   | Xosé   |   111    |
|  2   | maría  |   222    |
|  3   | manual |   333    |
|  4   | Xesús  |  4444    |
+------+--------+----------+
```

2. **Accións da táboa**

```
+-----+-------+--------+----------+
| axuda | cit | acción | cantidade |
+-----+-------+--------+----------+
| 1 | 2 | REDHAT | 10 |
| 2 | 4 | NOVELA | 20 |
| 3 | 4 | SOL    | 30 |
| 4 | 5 | FORD | 100 |
+-----+-----+--------+----------+
```

Consultas mediante JOIN

JOIN

A instrución SQL JOIN úsase para vincular varias táboas. Permitiranos obter unha lista dos campos que teñen coincidencias en ambas táboas:

```
# select nombre, telefono, accion, cantidad from clientes join acciones on clientes.cid=acciones.cid;
```

resultante:

```
+--------+----------+--------+----------+
| nome | teléfono | acción | cantidade |
+--------+----------+--------+----------+
| maría | 222 | REDHAT | 10 |
| Xesús | 4444 | NOVELA | 20 |
| Xesús | 4444 | SOL | 30 |
+--------+----------+--------+----------+
```

LEFT JOIN

A instrución LEFT JOIN daranos o resultado anterior máis os campos da táboa á esquerda do **JOIN** que non teñan coincidencias na táboa da dereita:

```
# select nome, telefono, accion, cantidade from clientes left join acciones on clientes.cid=acciones.cid;
```

co resultado:

```
+--------+----------+--------+----------+
| nome | telefono | acción | cantidade |
+--------+----------+--------+----------+
| Xosé | 111 | NULL | NULL |
| maría | 222 | REDHAT | 10 |
| manual | 333 | NULL | NULL |
| Xesús | 4444 | NOVELA | 20 |
| Xesús | 4444 | SOL | 30 |
+--------+----------+--------+----------+
```

RIGHT JOIN

Operación idéntica como no caso anterior pero coa táboa que se inclúe na consulta á dereita do **JOIN** :

```
# select nombre, telefono, accion, cantidad from clientes left join acciones
on clientes.cid=acciones.cid;
```

cuxo resultado será:

```
+--------+----------+--------+----------+
| nome | telefono | acción | cantidade |
+--------+----------+--------+----------+
| maría | 222 | REDHAT | 10 |
| Xesús | 4444 | NOVELA | 20 |
| Xesús | 4444 | SOL | 30 |
| NULL | NULL | FORD | 100 |
+--------+----------+--------+----------+
```

UNION e UNION ALL

Podemos combinar o resultado de varias afirmacións con UNION ou UNION ALL. UNION non nos mostra os resultados duplicados, pero UNION ALL si os mostra:

```
# select nome, telefono, accion, cantidade from clientes left join acciones
on clientes.cid=acciones.cid where accion is null union select nome,
telefono, accion, cantidade from clientes right join acciones on
clientes.cid=acciones.cid where nome is null;
```

que mostrará:

```
+--------+----------+--------+----------+
| nome | telefono | acción | cantidade |
+--------+----------+--------+----------+
| Xosé | 111 | NULL | NULL |
| manual | 333 | NULL | NULL |
| NULL | NULL | FORD | 100 |
+--------+----------+--------+----------+
```

Vistas

As vistas (“views”) en SQL son un mecanismo que permite xerar un resultado a partir dunha consulta (consulta) almacenada, e executar novas consultas sobre este resultado coma se dunha táboa normal se tratara. As vistas teñen a mesma estrutura que unha táboa: filas e columnas. A única diferenza é que só se almacena a definición deles, non os datos.

A cláusula CREATE VIEW permite a creación de vistas. A cláusula asigna un nome á vista e permítelle especificar a consulta que a define. A súa sintaxe é:

```
# CREATE VIEW id_vista [(columna,…)]AS especificación_consulta;
```

Opcionalmente, pódese asignar un nome a cada columna da vista. Se se especifica, a lista de nomes de columna debe ter o mesmo número de elementos que o número de columnas producidas pola consulta. Se se omite, cada columna da vista1 leva o nome da columna correspondente na consulta.

Referencias

SQL na Wikipedia http://es.wikipedia.org/wiki/SQL

Tutorial de SQL http://www.unalmed.edu.co/~mstabare/Sql.pdf

SQL - JOIN Basic http://ariel.esdebian.org/27200/sql-join-basico

Comandos SQL - http://www.postgresql.org/docs/9.1/static/sql-commands.html

---

MRZ 2022

galicia Xacobeo 21·22 CONCELLO DE VIGO OBRADOIROS DE EMPREGO DE GALICIA UNIÓN EUROPEA XUNTA DE GALICIA