DML

Índice de contenido

1.- INTRODUCCIÓN AL SQL.

- DDL/DML

2.- LENGUAJE DE MANIPULACIÓN DE DATOS (DML).

2.1.- DELETE

(p1b)

- Borra registros de una tabla.
- Ejemplo: DELETE FROM "tabla" WHERE "columna1" = "valor1"
- `Ejemplo: DELETE FROM My_table WHERE field2 = 'N';`

2.2.- DO

(p1c)

- Ejecuta la expresión, pero no muestra ningún resultado.
- Es la abreviación de SELECT *expr*

2.3.- INSERT

(p1c)

2.3.1.- Formas simples:

- Inserta uno o más registros a una tabla.
- Forma básica: INSERT INTO "tabla" ("columna1", ["columna2,... "]) VALUES ("valor1", ["valor2,..."]
- Las cantidades de columnas y valores deben ser las mismas.
- Si una columna no se especifica, le será asignado el valor por omisión.
- Los valores especificados (o implícitos) por la sentencia `INSERT` deberán satisfacer todas las restricciones aplicables.
- Si ocurre un error de sintaxis o si alguna de las restricciones es violada, no se agrega la fila y se devuelve un error.
- Ejemplo: INSERT INTO agenda_telefonica (nombre, numero) VALUES ('Roberto Jeldrez', '4886850');
- Cuando se insertan todos los valores de una tabla, no es necesario poner los nombres de los campos (columnas). Ejemplo (basado en el anterior): INSERT INTO agenda_telefonica VALUES ('Roberto Jeldrez', '4886850');

2.3.2.- Insertar varias filas con una sentencia.

- Se pueden insertar varias filas (registros) a la vez con una única sentencia.Ejemplo:

```
INSERT INTO agenda_telefonica VALUES ('Roberto Fernández',
'4886850'), ('Alejandro Sosa', '4556550');
(dando por supuesto que la tabla anterior tiene solamente dos
columnas).
```

- El ejemplo anterior equivaldría a las siguientes sentencias SQL:

```
INSERT INTO agenda_telefonica VALUES ('Roberto Fernández',
'4886850');
      INSERT INTO agenda_telefonica VALUES ('Alejandro Sosa',
      '4556550');
```

2.3.3.- Copiar filas de una tabla a otra tabla:

```
INSERT INTO phone_book2
      SELECT *
      FROM   phone_book
      WHERE  name IN ('John Doe', 'Peter Doe');
```

- La sentencia anterior inserta en la tabla "phone_book2" los datos de todas las columnas (campos) de la tabla phone_book, pero solamente los que cumplan con la condición especificada (que sean los nombres especificados).
- Otro ejemplo con solamente algunos campos:

```
INSERT INTO phone_book2 ( [name], [phoneNumber] )
      SELECT [name], [phoneNumber]
      FROM   phone_book
      WHERE  name IN ('John Doe', 'Peter Doe');
```

2.4.- LOAD DATA INFILE

- Lee registros desde un fichero de texto y los copia en una tabla rápidamente.
- Ejemplo:

LOAD DATA INFILE 'data.txt' INTO TABLE db2.my_table;

- Ejemplo (cuando el archivo no contiene datos de todas las columnas):

LOAD DATA INFILE 'persondata.txt' INTO TABLE db2.mytable(col1,col2);

- (No es de las más importantes).

2.5.- REPLACE

(p4c)

- Funciona como INSERT, pero si la clave primaria del registro que se inserta ya se encuentra en la tabla, entonces REEMPLAZA el registro de la tabla por el nuevo registro.
- Ejemplo:

REPLACE INTO tabla1 (campo1,campo2,campo3) VALUES (15,'18','Lavavajillas');

2.6.- SELECT

(P4B-P5A)

- Se usa para seleccionar registros de una o más tablas.
- ORDEN DE LAS CLÁUSULAS:

  SELECT

  FROM

  WHERE

  GROUP BY

  HAVING

- Ejemplos de uso:

SELECT 1+1; (devuelve 2).

(situaciones especiales).

SELECT columna1, columna2

FROM tabla1;

(permite seleccionar una o varias columnas)

SELECT columna1 AS aliasdecolumna1

FROM tabla1;

(a las columnas se les puede poner alias).

SELECT tabla1.columna1 AS aliasdecolumna1, tabla2.co11 AS aliasdecol1

FROM tabla1, tabla2;

(se pueden seleccionar columnas de varias tablas).

(se pueden poner varios alias).

SELECT c1
FROM tabla1 AS aliasdetabla1;

(las tablas también pueden llevar alias).

```
SELECT college, region, seed FROM tournament
ORDER BY region, seed;

(se pueden ordenar los registros seleccionados).
```

```
SELECT a, COUNT(b)
FROM test_table
ORDER BY a DESC;

(lo ordena de manera descendente).
```

```
SELECT marca, COUNT(modelo)
FROM coches
GROUP BY marca DESC;

(COUNT: cuenta el número de datos en la columna 'modelo')
(GROUP BY: los agrupa por el campo 'marca').
(Por tanto: mostrará ordenados por marca, todas las marcas y el número de
modelos que tiene cada marca).
```

```
SELECT col_name
FROM tbl_name
WHERE col_name > 0;

(solamente selecciona los registros que cumplen la condición dada por
WHERE).
```

Uso de Having:

Permite poner condiciones, como WHERE, pero estas condiciones se aplican después del ORDER BY y GROUP BY .

Además, HAVING puede llevar funciones de agregación (max, min, count, avg...), mientras que WHERE no puede.

```
SELECT user, MAX(salary)
FROM users
GROUP BY user

HAVING MAX(salary)>10;
```

```
SELECT c1,c2
FROM t1
INTO OUTFILE 'file_name';
```

(copia datos en un fichero de texto).

2.6.1.- JOIN[1]

(pág 6C).

- La sentencia **JOIN** en SQL permite combinar registros de dos o más tablas en una base de datos relacional.
- Tipos de JOIN:
    - El `NATURAL [LEFT] JOIN` de dos tablas se define semánticamente equivalente a un `INNER JOIN` o `LEFT JOIN` con una cláusula `USING` que nombra todas las columnas que existen en ambas tablas.
    - `INNER JOIN` y `,` (coma) son semánticamente equivalentes en la ausencia de una condicicón de join: ambos producen un producto Cartesiano entre las tablas especificadas (esto es, cada registro en la primera tabla se junta con cada registro en la segunda tabla).
    - `RIGHT JOIN` funciona análogamente a `LEFT JOIN`. Para mantener el código portable entre bases de datos, se recomienda que use `LEFT JOIN` en lugar de `RIGHT JOIN`.
    - `STRAIGHT_JOIN` es idéntico a `JOIN`, excepto que la tabla de la izquierda se lee siempre antes que la de la derecha. Esto puede usarse para aquéllos casos (escasos) en que el optimizador de join pone las tablas en orden incorrecto.

Algunos ejemplos de join:

```
mysql> SELECT * FROM table1,table2 WHERE
table1.id=table2.id;
```

[1]Parte del texto ha sido obtenido de la Wikipedia. (http://es.wikipedia.org).

```
mysql> SELECT * FROM table1 LEFT JOIN table2 ON
table1.id=table2.id;

mysql> SELECT * FROM table1 LEFT JOIN table2 USING
(id);

mysql> SELECT * FROM table1 LEFT JOIN table2 ON
table1.id=table2.id LEFT JOIN table3 ON
table2.id=table3.id;

mysql> SELECT * FROM table1 USE INDEX (key1,key2)
->          WHERE key1=1 AND key2=2 AND key3=3;

mysql> SELECT * FROM table1 IGNORE INDEX (key3)
->          WHERE key1=1 AND key2=2 AND key3=3;
```

2.6.2.- UNION

(p7a)

- Se usa para combinar el resultado de un número de comandos SELECT en un conjunto de resultados.
- Ejemplo:

```
(SELECT a FROM tbl_name WHERE a=10 AND B=1)
          UNION
          (SELECT a FROM tbl_name WHERE a=11 AND B=2)
          ORDER BY a;
```

2.6.3.- Subconsultas.

(p7c)

- Una subconsulta es un comando `SELECT` dentro de otro comando.
- `Ejemplo:`

```
SELECT *
FROM t1
WHERE column1 = (SELECT column1 FROM t2);
```

- Una subconsulta puede retornar un escalar (un valor único), un registro, una columna o una tabla (uno o más registros de una o más columnas)
- En su forma más sencilla, una subconsulta es una subconsulta escalar que retorna un único valor.

2.6.4.- Uso de subconsultas en subconsultas.

- Ejemplo:

```
SELECT * FROM t1 AS t
          WHERE 2 = (SELECT COUNT(*) FROM t1 WHERE t1.id = t.id);
```

2.6.5.- Subconsultas con ANY, IN y SOME.

(p8a)

La palabra clave `ANY` , que debe seguir a un operador de comparación, significa “return `TRUE` si la comparación es `TRUE` para `ANY` (cualquiera) de los valores en la columna que retorna la subconsulta.” Por ejemplo:

```
SELECT s1 FROM t1 WHERE s1 > ANY (SELECT s1 FROM t2);
```

Suponga que hay un registro en una tabla t1 que contiene (10). La expresión es TRUE si la tabla t2 contiene (21,14,7) ya que hay un valor 7 en t2 que es menor que 10. La expresión es FALSE si la tabla t2 contiene (20,10), o si la tabla t2 está vacía. La expresión es UNKNOWN si la tabla t2 contiene (NULL,NULL,NULL).

La palabra IN es un alias para = ANY. Por lo tanto, estos dos comandos son lo mismo:

```
SELECT s1 FROM t1 WHERE s1 = ANY (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 IN    (SELECT s1 FROM t2);
```

2.6.6.- Subconsultas con ALL.

(p8b)

- La palabra ALL, que debe seguir a un operador de comparación, significa "return TRUE si la comparación es TRUE para ALL todos los valores en la columna que retorna la subconsulta." Por ejemplo:

```
SELECT s1 FROM t1 WHERE s1 > ALL (SELECT s1 FROM t2);
```

Suponga que hay un registro en la tabla t1 que contiene (10). La expresión es TRUE si la tabla t2 contiene (-5,0,+5) ya que 10 es mayor que los otros tres valores en t2. La expresión es FALSE si la tabla t2 contiene (12,6,NULL,-100) ya que hay un único valor 12 en la tabla t2 mayor que 10. La expresión es UNKNOWN si la tabla t2 contiene (0,NULL,1).

Finalmente, si la tabla t2 está vacía, el resultado es TRUE. Puede pensar que el resultado debería ser UNKNOWN, pero lo sentimos, es TRUE. Así, aunque extraño, el siguiente comando es TRUE cuando la tabla t2 está vacía:

```
SELECT * FROM t1 WHERE 1 > ALL (SELECT s1 FROM t2);
```

2.6.7.- Subconsultas de registro.

(p8c)

- Una *subconsulta de registro* es una variante de subconsulta que retorna un único registro y por lo tanto retorna más de un valor de columna.
- Por ejemplo, la siguiente consulta responde a la petición, "encuentra todos los registros en la tabla t1 que también existen en la tabla t2":

```
SELECT column1,column2,column3
FROM t1
WHERE (column1,column2,column3) IN
(SELECT column1,column2,column3 FROM t2);
```

2.6.8.- EXISTS y NOT EXISTS.

(p8d)

Ejemplos:

¿Qué clase de tienda hay en una o más ciudades?

```
SELECT DISTINCT store_type FROM Stores
WHERE EXISTS (SELECT * FROM Cities_Stores
```

```
WHERE Cities_Stores.store_type = Stores.store_type);
```

- ¿Qué clase de tienda no hay en ninguna ciudad?

```
SELECT DISTINCT store_type FROM Stores
WHERE NOT EXISTS (SELECT * FROM Cities_Stores
WHERE Cities_Stores.store_type = Stores.store_type);
```

2.6.9.- Subconsultas en la cláusula FROM.

- Las subconsultas son legales en la cláusula `FROM` de un comando `SELECT`.
- `SELECT ... FROM (subquery) [AS] name ...`
- La cláusula `[AS]` *name* es obligatoria, ya que cada tabla en la cláusula `FROM` debe tener un nombre. Cualquier columna en la lista selecta de la `subquery` debe tener nombre único. Puede encontrar esta sintaxis descrita en este manual, dónde se usa el término "tablas derivadas."

- Ejemplo:

```
INSERT INTO t1 VALUES (1,'1',1.0);
          INSERT INTO t1 VALUES (2,'2',2.0);

          SELECT sb1,sb2,sb3
          FROM (SELECT s1 AS sb1, s2 AS sb2, s3*2 AS sb3 FROM t1) AS sb
          WHERE sb1 > 1;
```

Resultado: `2, '2', 4.0`.

- Aquí hay otro ejemplo: suponga que quiere conocer la media de un conjunto de sumas para una tabla agrupada.

```
SELECT AVG(sum_column1)
          FROM (SELECT SUM(column1) AS sum_column1
          FROM t1 GROUP BY column1) AS t1;
```

2.7.- TRUNCATE

```
TRUNCATE TABLE tbl_name
```

`TRUNCATE TABLE` vacía una tabla completamente. Lógicamente, esto es equivalente a un comando `DELETE` que borre todos los registros, pero hay diferencias prácticas bajo ciertas circunstáncias.

2.8.- UPDATE

Sintaxis para una tabla:

```
UPDATE [LOW_PRIORITY] [IGNORE] tbl_name
SET col_name1=expr1 [, col_name2=expr2 ...]
[WHERE where_definition]
[ORDER BY ...]
[LIMIT row_count]
```

El comando `UPDATE` actualiza columnas en registros de tabla existentes con nuevos valores. La cláusula `SET` indica qué columna modificar y los valores que puede recibir.

La cláusula `WHERE` , si se da, especifica qué registros deben actualizarse. De otro modo, se actualizan todos los registros. Si la cláusula `ORDER BY` se especifica, los registros se actualizan en el orden que se especifica. La cláusula `LIMIT` es el límite de registros a actualizar.

```
mysql> UPDATE persondata SET age=age+1;
```

Las asignaciones `UPDATE` se avalúna de izquierda a derecha. Por ejemplo, el siguiente comando dobla la columna `age` y luego la incrementa:

```
mysql> UPDATE persondata SET age=age*2, age=age+1;
```

Puede realizar operaciones `UPDATE` que cubran varias tablas. La parte `table_references` lista las tablas involucradas en el join. Aquí hay un ejemplo:

```
UPDATE items,month SET items.price=month.price
WHERE items.id=month.id;
```

2.9.- FUNCIONES DE AGREGACIÓN.

2.9.1.- AVG.

os.

- `AVG([DISTINCT] expr)`

Retorna el valor medio de *expr*. La opción `DISTINCT` puede usarse desde MySQL 5.0.3 para retornar la media de los valores distintos de `expr`.

```
mysql> SELECT student_name, AVG(test_score)
    ->        FROM student
    ->        GROUP BY student_name;
```

2.9.2.- COUNT

Retorna el contador del número de valores no `NULL` en los registros recibidos por un comando `SELECT`.

```
mysql> SELECT student.student_name,COUNT(*)
    ->        FROM student,course
    ->        WHERE student.student_id=course.student_id
    ->        GROUP BY student_name;
```

ejemplo:

```
mysql> SELECT COUNT(*) FROM student;
```

2.9.3.- MIN,MAX

`MIN([DISTINCT]` *expr*`),MAX([DISTINCT]` *expr*`)`

Retornas los valores máximos y mínimos de `expr`.

```
mysql> SELECT student_name, MIN(test_score), MAX(test_score)
    ->        FROM student
```

```
    ->        GROUP BY student_name;
```

2.9.4.- SUM

Retorna la suma de `expr`.

Suponga que una tabla llamada `sales` tiene las columnas `year`, `country`, `product`, y `profit` para guardar las ventas productivas:

```
CREATE TABLE sales
(
    year    INT NOT NULL,
    country VARCHAR(20) NOT NULL,
    product VARCHAR(32) NOT NULL,
    profit  INT
);
```

Los contenidos de la tabla pueden resumirse por año con un simple `GROUP BY` como este:

```
mysql> SELECT year, SUM(profit) FROM sales GROUP BY year;
+------+-------------+
| year | SUM(profit) |
+------+-------------+
| 2000 |        4525 |
| 2001 |        3010 |
+------+-------------+
```