Sumario

Comando DML Insert

Una sentencia *INSERT* de SQL agrega uno o más registros a una (y sólo una) tabla en una base de datos relacional.

Forma básica

```
INSERT INTO 'tablatura' ('columnaA',['columnaB,... '])
      VALUES ('valor1', ['valor2,...'])
```

O también se puede utilizar como:

```
INSERT INTO tablatura VALUES ('valor1','valor2')
```

Las cantidades de columnas y valores deben ser iguales. 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 especifican todos los valores de una tabla, se puede utilizar la sentencia acortada:

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

Ejemplo (asumiendo que 'nombre' y 'número' son las únicas columnas de la tabla 'agenda_telefonica'):

```
INSERT INTO agenda_telefonica VALUES ('Jhonny Aguilar', 080473968);
```

Formas avanzadas

Una característica de SQL (desde SQL-92) es el uso de *constructores de filas* para insertar múltiples filas a la vez, con una sola sentencia SQL:

```
INSERT INTO ''tabla'' (''columna1'', ['']columna2,... ''])
VALUES (''valor1a'', [''valor1b,...'']),
(''value2a'', [''value2b,...'']),...;
```

Esta característica es soportada por DB2, PostgreSQL (desde la versión 8.2), MySQL, y H2.

Ejemplo (asumiendo que 'nombre' y 'número' son las únicas columnas en la tabla 'agenda_telefonica'):

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

Que podía haber sido realizado por las sentencias

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

Notar que las sentencias separadas pueden tener semántica diferente (especialmente con respecto a los triggers), y puede tener diferente rendimiento que la sentencia de inserción múltiple.

Para insertar varias filas en MS SQL puede utilizar esa construcción:

```
INSERT INTO phone_book SELECT 'John Doe', '555-1212'
UNION ALL SELECT 'Peter Doe', '555-2323';
```

Tenga en cuenta que no se trata de una sentencia SQL válida de acuerdo con el estándar SQL (SQL: 2003), debido a la cláusula subselect incompleta.

Para hacer lo mismo en Oracle se usa la Tabla DUAL, siempre que se trate de solo una simple fila:

```
INSERT INTO phone_book
SELECT 'John Doe', '555-1212' FROM DUAL
UNION ALL
SELECT 'Peter Doe','555-2323' FROM DUAL
```

Una implementación conforme al estándar de esta lógica se muestra el siguiente ejemplo, o como se muestra arriba (no aplica en Oracle):

```
INSERT INTO phone_book
SELECT 'John Doe', '555-1212' FROM LATERAL ( VALUES (1) ) AS t(c)
UNION ALL
SELECT 'Peter Doe','555-2323' FROM LATERAL ( VALUES (1) ) AS t(c)
```

Copia de filas de otras tablas

Un INSERT también puede utilizarse para recuperar datos de otros, modificarla si es necesario e insertarla directamente en la tabla. Todo esto se hace en una sola sentencia SQL que no implica ningún procesamiento intermedio en la aplicación cliente. Un SUBSELECT se utiliza en lugar de la cláusula VALUES. El SUBSELECT puede contener JOIN, llamadas a funciones, y puede incluso consultar en la misma TABLA los datos que se inserta. Lógicamente, el SELECT se evalúa antes que la operación INSERT esté iniciada. Un ejemplo se da a continuación.

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

Una variación es necesaria cuando algunos de los datos de la tabla fuente se está insertando en la nueva tabla, pero no todo el registro. (O cuando los esquemas de las tablas no son iguales.)

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

El SELECT produce una tabla (temporal), y el esquema de la tabla temporal debe coincidir con el esquema de la tabla donde los datos son insertados.

Comando DML Delete

Para eliminar filas se usa la sentencia DELETE. La sintaxis genérica de DELETE para eliminar un registro existente es la siguiente:

```
DELETE FROM NombreTabla WHERE NombreCampo Operador Valor];
```

Los filtros que podemos utilizar dentro de la cláusula WHERE son los mismos que se pueden ver en las instrucciones SELECT ó bien UPDATE.

Ejemplo de aplicación:

```
DELETE FROM Productos WHERE idProducto = 1;
```

Si no aplicamos ningún filtro se eliminarán todos los registros sin ninguna limitación.

Si se borran mucho registros y se usa el cliente MySQL workbench es necesario quitar la protección de borrado masivo.
Ejemplo de como desactivar safe updates:

```
Set sql_safe_updates=0;
```

La instrucción DELETE consume muchos recursos si se la usa para eliminar todos los registros, por este motivo si se opta por vaciar completamente la tabla es recomendable utilizar la instrucción TRUNCATE TABLE, la cual elimina los registros en su totalidad dejando vacía la tabla y de manera menos traumática para el servidor de base de datos, con la siguiente sintaxis:

TRUNCATE TABLE NombreTabla;

Comando DML Update

Podemos modificar valores de las filas de una tabla usando la sentencia UPDATE. La sintaxis genérica de UPDATE para actualizar un registro existente es la siguiente:

```
UPDATE NombreTabla SET NombreCampo = "Valor1" [, NombreCampo2 = "Valor2"]
[ WHERE NombreCampo Operador Valor]
```

Es posible del mismo modo, actualizar el valor de más de una columna separándolas en la sección SET mediante comas como delimitador.

Actualizaciones selectivas

Mediante la cláusula WHERE se puede establecer una condición y sólo las filas/registros que cumplan esa condición serán
actualizadas.
Ejemplo de aplicación:

```
UPDATE Productos SET Nombre = 'iPhone 6' WHERE Nombre = 'iPhone Seis';
UPDATE Productos SET Precio = '499.99' WHERE idProducto = 10;
UPDATE Productos SET Stock = 600 , Precio = '499.99' WHERE idProducto = 15;
```

NOTA: Si el resultado es (0 row(s) affected) quiere decir que no hay registros que tengan el valor de la columna indicada en el WHERE

Funciones de Agrupamiento

Existen en SQL funciones que nos permiten contar registros, calcular sumas, promedios, obtener valores máximos y mínimos. Estas funciones se denominan Funciones de Agrupamiento porque operan sobre conjuntos de registros, no con datos individuales.
Tienen la característica de agrupar los resultados en un solo registro de salida.

Función COUNT()

Retorna la cantidad de valores que contiene un campo especificado. Por ejemplo, si se quiere saber la cantidad de productos que hay en la tabla Productos:

```
SELECT COUNT(idProductos) FROM Productos;
```

También es posible combinarla con la cláusula WHERE. Por ejemplo si se quiere saber cuántos productos tienen por Nombre "iPhone" de en tabla Productos:

```
SELECT COUNT(idProductos) FROM Productos WHERE Nombre LIKE "%iPhone%";
```

Función SUM()

Retorna la suma de los valores que contiene el campo especificado. Por ejemplo, si se quiere saber el stock de productos que hay en la tabla Productos:

```
SELECT SUM(Stock) FROM Productos;
```

También es posible combinarla con la cláusula WHERE. Por ejemplo si se quiere saber el stock de productos "iPad" que hay para vender:

```
SELECT SUM(Stock) FROM Productos WHERE Nombre LIKE "%iPad%";
```

Funciones MIN() y MAX()

Para averiguar el valor máximo o mínimo de un campo usamos las funciones max() y min() respectivamente. Ejemplo, se quiere saber cuál es el mayor precio de todos los productos:

```
SELECT MAX(Precio) FROM Productos;
```

También si se quiere saber cuál es el valor mínimo de los productos "TV":

```
SELECT MIN(Precio) FROM Productos WHERE Nombre LIKE "%TV%";
```

Función AVG()

Retorna el valor promedio de los valores del campo especificado. Por ejemplo, si se quiere saber el promedio del precio de los productos "Impresora":

SELECT AVG(Precio) FROM Productos WHERE Nombre LIKE '%Impresora%';

Tener en cuenta que no debe haber espacio entre el nombre de la función y el paréntesis pues puede confundirse con una referencia a una tabla o campo. Es decir…

Correcto :)
SELECT COUNT(*) FROM Productos;

Incorrecto :(
SELECT COUNT   (*) FROM Productos;

Agrupación con GROUP BY

La agrupación es un concepto básico de Base de Datos, La cláusula GROUP BY, como su traducción lo indica tiene como propósito agrupar información de acuerdo a un criterio en común. Por lo general se utiliza con funciones de agrupación o de agregación (COUNT, MIN, MAX, AVG, SUM). Por ejemplo usando la función SUM…

La función GROUP BY permite hacer esto de manera automática a partir de tomar un valor o dato común. Por ejemplo:

```
SELECT Categoria, SUM(Stock) FROM Productos GROUP BY Categoria;
```

El comportamiento de la función GROUP BY dependerá de la función de agrupación que se esté utilizando.

Agrupación con HAVING

La cláusula HAVING permite hacer selecciones en situaciones en las que no es posible usar WHERE. Dado que se establece un criterio sobre un valor dado por una función de agrupamiento y no por valores de registros.
Un ejemplo completo sería:

```
SELECT Categoria, SUM(Stock) FROM Productos GROUP BY Categoria HAVING SUM(Stock)
> 250;
```