Apunte De SQL

Tablas

Una base de datos a menudo contiene uno o más tablas. Cada tabla es identificada por un nombre (por ejemplo "Clientes" o "Órdenes"). Las tablas contienen registros (filas) con datos.

El siguiente es un ejemplo de una tabla llamada "Personas":

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

La tabla contiene tres registros (uno para cada persona) y cinco columnas (P_Id, Apellido, Nombre, Dirección, and Ciudad).

Sentencias SQL

La mayoría de las acciones que se realizan en una base de datos se hacen con sentencias SQL.

La siguiente declaración SQL se selecciona todos los registros en La Tabla "Personas":

SELECT * FROM Personas

Tenga en mente que... • SQL no es sensible a las mayúsculas

Punto y coma después de la sentencia SQL?

Algunos sistemas de base de datos requieren un punto y coma al final de las sentencias SQL.

El punto y coma es una manera estándar para separas cada sentencia SQL en un sistema de base de datos que permite ejecutar más de una sentencia SQL para ser ejecutadas en la misma llamada al servidor.

SQL DML y DDL

SQL puede ser dividido en dos partes: The Data Manipulatión Language (DML) (lenguaje de manipulación de datos) y el Data Definition Language (DDL).(lenguaje de manipulación de datos)

Las consultas y modificaciones son parte del DML:

- **SELECT** – extrae los datos de la Base de datos
- **UPDATE** – modifica los datos de la Base de datos
- **DELETE** – borras los datos de la Base de datos

- **INSERT INTO** – inserta nuevos datos en una base de datos

La DDL permite crear o suprimir tablas en la base de datos. También definir índices (llaves), especificar los enlaces entre tablas, e impone limitaciones entre tablas. Las más importantes declaraciones DDL son:

- **CREATE DATABASE** – crea una nueva base de datos
- **ALTER DATABASE** – modifica una base de datos
- **CREATE TABLA** – crea una nueva tabla
- **ALTER TABLA** – modifica una tabla
- **DROP TABLA** - Borra una tabla
- **CREATE INDEX** – crea un índice
- **DROP INDEX** – borra un índice

La sentencia SELECT

La sentencia SELECT es utilizada para extraer datos de una base de datos.

El resultado es almacenado en una tabla de resultados, llamada el resultado-set.

Sintaxis

```
SELECT column_name(s)
FROM tabla_name
```

y

```
SELECT * FROM tabla_name
```

Note: SQL no es sensible a las mayúsculas. SELECT es lo mismo que select.

Un ejemplo SELECT

La Tabla “Personas”:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Se seleccionará el contenido de las columnas llamadas "Apellido" y "Nombre" de la tabla.

Se usa la siguiente sentencia SELECT:

```
SELECT Apellido,Nombre FROM Personas
```

El conjunto de resultados es:

Apellido	Nombre
Hansen	Ola
Svendson	Tove
Pettersen	Kari

SELECT * Ejemplo

Seleccionaremos todas las columnas de la tabla “Personas”.

Se usa la siguiente sentencia SELECT:

SELECT * FROM Personas

Tip: el asterisco (*) es el camino rápido para seleccionar todas las columnas

El conjunto de resultados es:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

La navegación por el set de resultados

La mayoría de los sistemas de base de datos permite la navegación en el set de resultados con las siguientes funciones: Mover-a-First-Record, Get-Record-Contenido, Mover-a-Next-Record, etc.

La sentencia SELECT DISTINCT

En una tabla, algunas de las columnas podrían contener valores duplicados.

Este no es un problema, sin embargo, a veces, usted querrá lista sólo los diferentes (distintos) valores de una tabla.

La palabra clave DISTINCT puede ser utilizada para regresar sólo distintos (diferentes) valores.

Sintaxis:

SELECT DISTINCT column_name(s)
FROM tabla_name

Ejemplo

La Tabla “Personas”:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Se seleccionaran solo los valores distintos de la columna llamada "Ciudad" desde la tabla.

Se usa la siguiente sentencia SELECT:

```
SELECT DISTINCT Ciudad FROM Personas
```

El conjunto de resultados es:

Ciudad
Sandnes
Stavanger

La clausula top

La clausula TOP es usada para especificar la cantidad de registros que retornara la consulta

Puede ser muy útil para tablas de cientos de registros, donde la ejecución puede impactar sobre la performance del

Nota: No todos los sistemas de Base de datos soportan la clausula top.

SQL Server Sintaxis

```
SELECT TOP number|percent column_name(s)
FROM tabla_name
```

SELECT TOP su equivalencia en MySQL y Oracle

MySQL Sintaxis

```
SELECT column_name(s)
FROM tabla_name
LIMIT number
```

Ejemplo

```
SELECT *
FROM Personas
```

```
LIMIT 5
```

Oracle Sintaxis

```
SELECT column_name(s)
FROM tabla_name
WHERE ROWNUM <= number
```

Ejemplo

```
SELECT *
FROM Personas
WHERE ROWNUM <=5
```

Ejemplo

La Tabla “Personas”:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Tom	Vingvn 23	Stavanger

Se pedirá solo los dos primeros registros de la tabla.

Se usa la siguiente sentencia SELECT:

```
SELECT TOP 2 * FROM Personas
```

El conjunto de resultados es:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

Se seleccionará solo el 50% de los registros de la tabla.

Se usa la siguiente sentencia SELECT:

```
SELECT TOP 50 PERCENT * FROM Personas
```

El conjunto de resultados es:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes

2	Svendson	Tove	Borgvn 23	Sandnes

ORDER BY

La palabra clave ORDER BY es utilizada para ordenar el set de resultado por una columna específica.

La palabra clave ORDER BY ordenar los registros en orden ascendente por defecto.

Si usted quiere ordenar los registros en un orden descendente, se usa el DESC clave.

Sintaxis

```
SELECT column_name(s)
FROM tabla_name
ORDER BY column_name(s) ASC|DESC
```

Ejemplo

La Tabla "Personas":

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Tom	Vingvn 23	Stavanger

Se seleccionara todas las personas desde la tabla y se ordenará la lista por su apellido.

Se usa la siguiente sentencia SELECT:

```
SELECT * FROM Personas
ORDER BY Apellido
```

El conjunto de resultados es:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
4	Nilsen	Tom	Vingvn 23	Stavanger
3	Pettersen	Kari	Storgt 20	Stavanger
2	Svendson	Tove	Borgvn 23	Sandnes

Ejemplo ORDER BY DESC

Se seleccionara todas las personas desde la tabla y se ordenará la lista por su apellido de manera descendente.

Se usa la siguiente sentencia SELECT:

```
SELECT * FROM Personas
ORDER BY Apellido DESC
```

El conjunto de resultados es:

P_Id	Apellido	Nombre	Dirección	Ciudad
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Tom	Vingvn 23	Stavanger
1	Hansen	Ola	Timoteivn 10	Sandnes

La clausula WHERE

La cláusula WHERE es utilizada para extraer sólo los registros que cumplen un criterio.

Sintaxis

```
SELECT column_name(s)
FROM tabla_name
WHERE column_name operator value
```

Ejemplo

La Tabla “Personas”:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Se seleccionará solo las personas que viven en la ciudad de "Sandnes".

Se usa la siguiente sentencia SELECT:

```
SELECT * FROM Personas
WHERE Ciudad='Sandnes'
```

El conjunto de resultados es:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

Comillas en los campos de textos

SQL usa comillas simples en los valores de los campos de textos (la mayoría de los sistemas de base de datos aceptan también las comillas dobles). Mientras tanto los valores numéricos no están encerrados entre comillas.

Valores de textos:

Correcto:

```
SELECT * FROM Personas WHERE Nombre='Tove'
```

Incorrecto:

```
SELECT * FROM Personas WHERE Nombre=Tove
```

Valores numéricos:

Correcto:

```
SELECT * FROM Personas WHERE Year=1965
```

Incorrecto:

```
SELECT * FROM Personas WHERE Year='1965'
```

Operadores permitidos en la clausula WHERE

Operator	Descripción
=	Igual
<>	Distinto
>	Mas grande que
<	Menor que
>=	Mayor o igual
<=	Menor o igual
BETWEEN	Incluido en un rango
LIKE	Filtrado por un patrón
IN	Utilizado cuando se conoce exactamente los valores que se espera que retorne de una columna

Note: En algunas versiones el símbolo de distinto no es e <> sino que podría escribirse como !=

El operador LIKE

Es usado para buscar por un patrón en una columna.

SQL LIKE Sintaxis

```
SELECT column_name(s)
FROM tabla_name
WHERE column_name LIKE pattern
```

EJEMPLO

La Tabla “Personas”:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Se seleccionará las personas que viven en la ciudad que comienza con “s”.

Se usa la siguiente sentencia SELECT:

```
SELECT * FROM Personas
WHERE Ciudad LIKE 's%'
```

El signo "%" puede ser usado para definir un patrón que puede ser usado antes y después del patrón

El conjunto de resultados es:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Se seleccionarán las personas que viven en la ciudad que termina con una “s”.

Se usa la siguiente sentencia SELECT:

```
SELECT * FROM Personas
WHERE Ciudad LIKE '%s'
```

El conjunto de resultados es:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

Se seleccionarán las personas que viven en la ciudad que contiene un patrón “tav”

Se usa la siguiente sentencia SELECT:

```
SELECT * FROM Personas
WHERE Ciudad LIKE '%tav%'
```

El conjunto de resultados es:

P_Id	Apellido	Nombre	Dirección	Ciudad
3	Pettersen	Kari	Storgt 20	Stavanger

Es posible seleccionar las personas que viven en la ciudad que no contienen el patrón “tav”, para ello se utiliza la palabra NOT

Se usa la siguiente sentencia SELECT:

```
SELECT * FROM Personas
WHERE Ciudad NOT LIKE '%tav%'
```

El conjunto de resultados es:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

Comodines

Los comodines pueden sustituir uno o más caracteres cuando se busca en una base de datos.

Son usados con el operador SQL LIKE.

Lista de comodines usados:

Wildcard	Descripción
%	Un sustituto para cero o mas caracteres
_	Un sustituto para estrictamente un carácter
[charlist]	Algún carácter en la lista mostrada
[^charlist] o [!charlist]	Algún carácter que no está en la lista mostrada

Ejemplo

Tabla "Personas":

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Usando el comodín %

Se seleccionan las personas que viven en una ciudad que comience con “sa”.

Se usa la siguiente sentencia SELECT:

```
SELECT * FROM Personas
WHERE Ciudad LIKE 'sa%'
```

El conjunto de resultados es:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

Se seleccionan las personas que viven en una ciudad que contiene el patrón “nes” dentro de las ciudades.

Se usa la siguiente sentencia SELECT:

```
SELECT * FROM Personas
WHERE Ciudad LIKE '%nes%'
```

El conjunto de resultados es:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

Usando el comodín _

Se seleccionara las personas con el nombre que comienza con un carácter y continúa con “_la”.

Se usa la siguiente sentencia SELECT:

```
SELECT * FROM Personas
WHERE Nombre LIKE '_la'
```

El conjunto de resultados es:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes

Se seleccionara las personas con el nombre que comienza con “s” y continúa con un carácter, sigue con “end” otro carácter y continúa con “on”.

Se usa la siguiente sentencia SELECT:

```
SELECT * FROM Personas
WHERE Apellido LIKE 'S_end_on'
```

El conjunto de resultados es:

P_Id	Apellido	Nombre	Dirección	Ciudad
2	Svendson	Tove	Borgvn 23	Sandnes

Usando los comodines de una lista

Se seleccionara las personas con los apellidos que comienza con “b” o “s” o “p.

Se usa la siguiente sentencia SELECT:

```
SELECT * FROM Personas
WHERE Apellido LIKE '[bsp]%'
```

El conjunto de resultados es:

P_Id	Apellido	Nombre	Dirección	Ciudad
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Se seleccionara las personas con el nombre que no comienza con b" o "s" o "p".

Se usa la siguiente sentencia SELECT:

```
SELECT * FROM Personas
WHERE Apellido LIKE '[!bsp]%'
```

El conjunto de resultados es:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes

El operador IN

El operador IN permite especificar múltiples valores en una clausula WHERE

```
SELECT column_name(s)
FROM tabla_name
WHERE column_name IN (value1,value2,...)
```

Ejemplo

La Tabla “Personas”:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Se seleccionara las personas con el apellido igual “Hansen" o "Pettersen" .

Se usa la siguiente sentencia SELECT:

```
SELECT * FROM Personas
WHERE Apellido IN ('Hansen','Pettersen')
```

El conjunto de resultados es:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

El operador BETWEEN

El operador BETWEEN selecciona un rango de datos entre dos valores. El valor puede ser números , textos , o fechas.

```
SELECT column_name(s)
FROM tabla_name
WHERE column_name
BETWEEN value1 AND value2
```

Ejemplo

La Tabla “Personas”:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Se seleccionara las personas con apellido alfabéticamente entre "Hansen" y"Pettersen".

Se usa la siguiente sentencia SELECT:

```
SELECT * FROM Personas
WHERE Apellido
BETWEEN 'Hansen' AND 'Pettersen'
```

El conjunto de resultados es:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes

De acuerdo a los sistemas de base de datos utilizados los valores mostrados pueden incluir los valores solicitados o excluirlos , o puede incluir el primero y excluir el último.

Para mostrar las personas que están fuera del rango se usa el NOT BETWEEN:

```
SELECT * FROM Personas
WHERE Apellido
NOT BETWEEN 'Hansen' AND 'Pettersen'
```

El conjunto de resultados es:

P_Id	Apellido	Nombre	Dirección	Ciudad
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Operador Exist

La sentencia IN se utiliza para enlazar la consulta interna y la consulta externa en una instrucción de subconsulta. IN no es la única forma de hacerlo – uno puede utilizar muchos operadores tales como >, <, o =. EXISTS es un operador especial que describiremos en esta sección.

EXISTS simplemente verifica si la consulta interna arroja alguna fila. Si lo hace, entonces la consulta externa procede. De no hacerlo, la consulta externa no se ejecuta, y la totalidad de la instrucción SQL no arroja nada.

La sintaxis para EXISTS es

```
SELECT "nombre1_columna" FROM "nombre1_tabla"WHERE EXISTS (SELECT * FROM
"nombre2_tabla" WHERE [Condición])
```

Por favor note que en vez de *, puede seleccionar una o más columnas en la consulta interna. El efecto será idéntico.

Utilizamos tablas de ejemplos:

Tabla información

store_name	Sales	Date
Los Angeles	1500 €	05-Jan-1999
San Diego	250 €	07-Jan-1999
Los Angeles	300 €	08-Jan-1999
Boston	700 €	08-Jan-1999

Tabla Geografía

region_name	store_name
East	Boston
East	New York
West	Los Angeles
West	San Diego

Se utilizará la siguiente consulta SQL:

```
SELECT SUM(Sales) FROM Store_Information
 WHERE EXISTS
(SELECT * FROM Geografia WHERE region_name = 'West')
```

El set- resultado es :

SUM(Sales)

2750

Al principio, esto puede parecer confuso, debido a que la subsequencia incluye la condición [region_name = 'West'], aún así la consulta sumó los negocios para todas las regiones. Si observamos de cerca, encontramos que debido a que la subconsulta arroja más de 0 filas, la condición EXISTS es verdadera, y la condición colocada dentro de la consulta interna no influencia la forma en que se ejecuta la consulta externa.

El predicado EXISTS (con la palabra reservada NOT opcional) se utiliza en comparaciones de verdad/falso para determinar si la subconsulta devuelve algún registro. Supongamos que deseamos recuperar todos aquellos clientes que hayan realizado al menos un pedido:

```
SELECT Clientes.Compañía, Clientes.Teléfono FROM Clientes WHERE EXISTS
(SELECT FROM Pedidos WHERE Pedidos.IdPedido = Clientes.IdCliente)
```

Clausula ANY

Se puede utilizar el predicado ANY o SOME, los cuales son sinónimos, para recuperar registros de la consulta principal, que satisfagan la comparación con cualquier otro registró recuperado en la subconsulta. El ejemplo siguiente devuelve todos los productos cuyo precio unitario es mayor que el de cualquier producto vendido con un descuento igual o mayor al 25 por ciento:

```
SELECT * FROM Productos WHERE PrecioUnidad > ANY
(SELECT PrecioUnidad FROM DetallePedido WHERE Descuento >= 0 .25);
```

El predicado ALL se utiliza para recuperar únicamente aquellos registros de la consulta principal que satisfacen la comparación con todos los registros recuperados en la subconsulta. Si se cambia ANY por ALL en el ejemplo anterior, la consulta devolverá únicamente aquellos productos cuyo precio unitario sea mayor que el de todos los productos vendidos con un descuento igual o mayor al 25 por ciento. Esto es mucho más restrictivo.

ANY significa que, para que una fila de la consulta externa satisfaga la condición especificada, la comparación se debe cumplir para al menos un valor de los devueltos por la subconsulta.

Por cada fila de la consulta externa se evalúa la comparación con cada uno de los valores devueltos por la subconsulta y si la comparación es True para alguno de los valores ANY es verdadero, si la comparación no se cumple con ninguno de los valores de la consulta, ANY da False a no ser que todos los valores devueltos por la subconsulta sean nulos en tal caso ANY dará NULL.
Si la subconsulta no devuelve filas ANY da False incluso si expresión es nula.

Ejemplo:

```
SELECT * FROM empleados
WHERE cuota > ANY (SELECT cuota  FROM empleados empleados2
where empleados.oficina = empleados2.oficina);
```

Obtenemos los empleados que tengan una cuota superior a la cuota de alguno de sus compañeros de oficina, es decir los empleados que no tengan la menor cuota de su oficina. En este caso hemos tenido un alias de tabla en la subconsulta (*empleados2*) para poder utilizar una referencia externa.

Clausula ALL

Con el modificador ALL, para que se cumpla la condición, la comparación se debe cumplir con cada uno
de los valores devueltos por la subconsulta.
Si la subconsulta no devuelve ninguna fila ALL da True.

```
SELECT * FROM empleados WHERE cuota > ALL (
SELECT cuota FROM empleados empleados2
WHERE empleados.oficina = empleados2.oficina);
```

En el ejemplo se obtendrá los empleados que tengan una cuota superior a todas las cuotas de la oficina del empleado. Se podría pensar que se obtiene el empleado de mayor cuota de su oficina pero no lo es, aquí hay un problema, la cuota del empleado aparece en el resultado de subconsulta por lo tanto > no se cumplirá para todos los valores y sólo saldrán los empleados que no tengan oficina (para los que la subconsulta no devuelve filas).

Para salvar el problema tendríamos que quitar del resultado de la subconsulta la cuota del empleado modificando el WHERE:

```
SELECT * FROM empleados
WHERE empleados.oficina = empleados2.oficina
AND empleados.numemp <> empleados2.numemp);
```

De esta forma saldrían los empleados que tienen una cuota mayor que cualquier otro empleado de su misma oficina.
O bien

```
SELECT * FROM empleados
WHERE empleados.oficina = empleados2.oficina
AND empleados.cuota <> empleados2.cuota);
```

Para no considerar los empleados que tengan la misma cuota que el empleado. En este caso saldría los empleados con la mayor cuota de sus oficina, pero si dos empleados tienen la misma cuota superior, saldrían, hecho que no sucedería con la otra versión.

Cuando la comparación es una igualdad, = ANY es equivalente a IN y <> ALL es equivalente a NOT IN

Operador AND & OR

El operador AND muestra un registro si la primera y la segunda condición son verdaderas.

El operador OR muestra un registro en donde la primera condición o la segunda condición son verdaderas.

Ejemplo AND

La Tabla “Personas”:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Seleccionar solo las personas que su nombre sea igual a "Tove" y el apellido sea igual a "Svendson":

Se usa la siguiente sentencia SELECT:

```
SELECT * FROM Personas
WHERE Nombre='Tove'
AND Apellido='Svendson'
```

El conjunto de resultados es:

P_Id	Apellido	Nombre	Dirección	Ciudad
2	Svendson	Tove	Borgvn 23	Sandnes

OR Operator Ejemplo

Seleccionar solo las personas que su nombre sea igual a "Tove" o "Ola":

Se usa la siguiente sentencia SELECT:

```
SELECT * FROM Personas
WHERE Nombre='Tove'
OR Nombre='Ola'
```

El conjunto de resultados es:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes

Combinando AND & OR

Se puede combinar AND y OR (usar paréntesis parar formar expresiones complejas).

Seleccionar solo las personas con el apellido igual a "Svendson" y el nombre igual a "Tove" o "Ola":

Se usa la siguiente sentencia SELECT:

```
SELECT * FROM Personas WHERE
Apellido='Svendson'
AND (Nombre='Tove' OR Nombre='Ola')
```

El conjunto de resultados es:

P_Id	Apellido	Nombre	Dirección	Ciudad
2	Svendson	Tove	Borgvn 23	Sandnes

Alias

Usted puede dar otro nombre a una tabla o una columna utilizando un alias.

Esto puede ser muy útil cuando los nombres de las tablas o columnas tienen un nombre muy largo o complejo.

El nombre de Alias puede ser cualquiera, pero usualmente debería ser corto

Uso de Alias para tablas

```
SELECT column_name(s)
```

```
FROM tabla_name
AS alias_name
```

Uso de Alias para columnas

```
SELECT column_name AS alias_name
FROM tabla_name
```

EJEMPLO

Asumamos que tenemos una tabla llamada "personas" y otra tabla llamada "Product_Orden". Se dará alias de "p" un "po" respectivamente.

Se listará todos los pedidos que "Ola Hansen" es realizó.

Se usa la siguiente sentencia SELECT:

```
SELECT po.OrderID, p.Apellido, p.Nombre
FROM Personas AS p,
Product_Orden AS po
WHERE p.Apellido='Hansen'
WHERE p.Nombre='Ola'
```

La misma sentencia SELECT sin alias:

```
SELECT Product_Orden.OrderID, Personas.Apellido, Personas.Nombre
FROM Personas,
Product_Orden
WHERE Personas.Apellido='Hansen'
WHERE Personas.Nombre='Ola'
```

Se verá que cuando existen dos sentencias Select, utilizar alias hace que las consultas sean más fáciles en la lectura y la escritura.

JOIN

La palabra clave JOIN es usada en las sentencias SQL para consultar tablas de dos o más tablas, basada sobre una relación entre columnas de estas tablas

Una tabla en la base de datos menudo son relacionadas a otra con una clave (PK, FK)

Una Clave Primaria PK es una columna (o combinación de columnas) con un único valor distinto para cada fila. Cada valor de la PK debe ser único en cada tabla. El propósito es el de vincular datos juntos cruzando tablas, sin repetir todos los datos en cada tabla.

La Tabla “Personas”:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes

2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Notar que la columna P_Id es la clave primaria en la tabla “Personas”. Esto significa que dos filas no pueden tener el mismo P_Id. La P_Id distingue dos personas inclusive si tienen el mismo nombre.

Tabla “Orden":

O_Id	**OrderNo**	**P_Id**
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

Notar que la columna O_Id es la clave primaria en la tabla Orden y que la columna P_id hace referencia a las personas en la tabla Personas sin usar sus nombres.

La relación entre las dos tablas está dada por la columna "P_Id".

Diferentes tipos de SQL JOINs

Antes de continuar con ejemplos , se listaran los tipos de JOIN que pueden ser usados y la diferencia entre ellos.

- **JOIN**: Retorna las filas cuando hay al menos una concordancia entre ambas tablas
- **LEFT JOIN**: Retorna todas las filas desde la tabla de la izquierda, aunque no haya coincidencias en la tabla derecha
- **RIGHT JOIN**: Retorna todas las filas desde la tabla de la derecha ,aunque no haya coincidencias en la tabla izquierda
- **FULL JOIN**: Retorna filas cuando hay una coincidencia en una de las tablas

Palabra clave INNER JOIN

La palabra clave INNER JOIN retorna filas cuando hay al menos una coincidencia en ambas tablas.

Sintaxis

```
SELECT column_name(s)
FROM tabla_name1
INNER JOIN tabla_name2
ON tabla_name1.column_name=tabla_name2.column_name
```

PS: INNER JOIN es lo mismo que JOIN.

SQL INNER JOIN Ejemplo

La Tabla “Personas”:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

La tabla "Orden":

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

Se listara las personas con una orden.

Se usa la siguiente sentencia SELECT:

```
SELECT Personas.Apellido, Personas.Nombre, Orden.OrderNo
FROM Personas
INNER JOIN Orden
ON Personas.P_Id=Orden.P_Id
ORDER BY Personas.Apellido
```

El conjunto de resultados es:

Apellido	Nombre	OrderNo
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678

La palabra clave INNER JOIN retorna las filas si hay al menos una coincidencia en ambas tablas. Si hay filas en "Personas" que no tienen coincidencias en "Orden", estas filas NO serán listadas.

Palabra clave LEFT JOIN

La palabra clave LEFT JOIN retorna todas las filas desde la tabla de la izquierda (tabla_name1), siempre que no haya coincidencias en la tabla derecha (tabla_name2).

Sintaxis

```
SELECT column_name(s)
FROM tabla_name1
LEFT JOIN tabla_name2
ON tabla_name1.column_name=tabla_name2.column_name
```

Nota: En algunas bases de datos la palabra clave LEFT JOIN es llamada LEFT OUTER JOIN.

LEFT JOIN Ejemplo

La Tabla "Personas":

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

La tabla "Orden":

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

Se listara todas las personas y su orden , si esta existe.

Se usa la siguiente sentencia SELECT:

```
SELECT Personas.Apellido, Personas.Nombre, Orden.OrderNo
FROM Personas
LEFT JOIN Orden
ON Personas.P_Id=Orden.P_Id
ORDER BY Personas.Apellido
```

El conjunto de resultados es:

Apellido	Nombre	OrderNo

Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678
Svendson	Tove	

La palabra clave LEFT JOIN retorna todas las filas desde la tabla izquierda (Personas), aunque no haya coincidencia en la tabla derecha (Orden).

Palabra clave RIGHT JOIN

La palabra clave RIGHT JOIN retorna todas las filas desde la tabla derecha (tabla_name2),aunque no haya coincidencia en la tabla izquierda (tabla_name1).

RIGHT JOIN Sintaxis

```
SELECT column_name(s)
FROM tabla_name1
RIGHT JOIN tabla_name2
ON tabla_name1.column_name=tabla_name2.column_name
```

Nota: En algunas bases de datos la palabra calve RIGHT JOIN es llamada RIGHT OUTER JOIN.

RIGHT JOIN Ejemplo

La Tabla “Personas”:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

La tabla "Orden":

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

Se listara todos los órdenes y personas contenidas si estas existen.

Se usa la siguiente sentencia SELECT:

```
SELECT Personas.Apellido, Personas.Nombre, Orden.OrderNo
FROM Personas
RIGHT JOIN Orden
ON Personas.P_Id=Orden.P_Id
ORDER BY Personas.Apellido
```

El conjunto de resultados es:

Apellido	**Nombre**	**OrderNo**
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678
		34764

La palabra clave RIGHT JOIN retorna todas las filas desde la tabla derecha (Orden), aunque no existan coincidencias en la tabla izquierda (Personas).

Palabra clave FULL JOIN

La palabra clave FULL JOIN retorna las filas cuando hay coincidencia en una de las tablas.

Sintaxis

```
SELECT column_name(s)
FROM tabla_name1
FULL JOIN tabla_name2
ON tabla_name1.column_name=tabla_name2.column_name
```

FULL JOIN Ejemplo

La Tabla “Personas”:

P_Id	**Apellido**	**Nombre**	**Dirección**	**Ciudad**
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

La tabla “Orden”:

O_Id	**OrderNo**	**P_Id**
1	77895	3
2	44678	3

3	22456	1
4	24562	1
5	34764	15

Se listara todas las personas y sus órdenes y todos los órdenes con sus personas.

Se usa la siguiente sentencia SELECT:

```
SELECT Personas.Apellido, Personas.Nombre, Orden.OrderNo
FROM Personas
FULL JOIN Orden
ON Personas.P_Id=Orden.P_Id
ORDER BY Personas.Apellido
```

El conjunto de resultados es:

Apellido	**Nombre**	**OrderNo**
Hansen	Ola	22456
Hansen	Ola	24562
Pettersen	Kari	77895
Pettersen	Kari	44678
Svendson	Tove	
		34764

La palabra clave FULL JOIN retorna todas las filas de la tabla izquierda (Personas), y todas las filas de las tabla derecha (Orden).Si hay filas en "Personas" que no tienen concordancia en "Orden", o si hay filas en "Orden" que no tienen correlación en "Personas", estas filas serán listadas también.

El operador UNION

El operador UNION es usado para combinar el resultado de dos o más sentencias SELECT.

Nótese que cada sentencia SELECT dentro de la unión debe tener el mismo número de columnas. Las columnas deben tener igual tipo de datos , también las columna de cada sentencia SELECT debe ser del mismo orden.

Sintaxis

```
SELECT column_name(s) FROM tabla_name1
UNION
SELECT column_name(s) FROM tabla_name2
```

Nota: El operador UNION selecciona solo los valores distintos por defecto. También permite duplicar valores usando el término UNION ALL.

Sintaxis

```
SELECT column_name(s) FROM tabla_name1
UNION ALL
SELECT column_name(s) FROM tabla_name2
```

La columna nombres en el resultado de una UNION son siempre igual a la columna nombres de la primer sentencia SELECT en la UNION.

UNION Ejemplo

"Empleados_Norway":

E_ID	**E_Nombre**
01	Hansen, Ola
02	Svendson, Tove
03	Svendson, Stephen
04	Pettersen, Kari

"Empleados_USA":

E_ID	**E_Nombre**
01	Turner, Sally
02	Kent, Clark
03	Svendson, Stephen
04	Scott, Stephen

Se listara todos los diferentes empleados en Norway and USA.

Se usa la siguiente sentencia SELECT:

```
SELECT E_Nombre FROM Empleados_Norway
UNION
SELECT E_Nombre FROM Empleados_USA
```

El conjunto de resultados es:

E_Nombre
Hansen, Ola
Svendson, Tove
Svendson, Stephen
Pettersen, Kari
Turner, Sally

Kent, Clark
Scott, Stephen

Note: este comando no puede ser usado para listar todos los empleados en Norway and USA. En el ejemplo hay dos empleados con el mismo nombre y solo es mostrado uno de ellos, porque el comando unión devuelve los valores distintos.

UNION ALL Ejemplo

Se listará todos los empleados en Norway and USA:

```
SELECT E_Nombre FROM Empleados_Norway
UNION ALL
SELECT E_Nombre FROM Empleados_USA
```

Resultado

E_Nombre
Hansen, Ola
Svendson, Tove
Svendson, Stephen
Pettersen, Kari
Turner, Sally
Kent, Clark
Svendson, Stephen
Scott, Stephen

Funciones de agregado

Las funciones de agregado retornan un simple valor, calculado desde el valor en la columna.

Funciones de agregados más comunes:

- AVG() – devuelve el valor promedio
- COUNT() – devuelve el numero de filas
- FIRST() – devuelve el primer valor
- LAST() – devuelve el ultimo valor
- MAX() – devuelve el mayor valor
- MIN() – devuelve el menor valor
- SUM() – devuelve la suma

Funciones escalares

Las funciones escalares devuelven un valor único, basado en el valor de entrada.

Funciones escalares más comunes:

- UCASE() – convierte el campo en mayúsculas
- LCASE() - convierte el campo en minúsculas
- MID() – extrae caracteres desde un texto
- LEN() – retorna la longitud del texto
- ROUND() – redondea un numero con decimales específicos
- NOW() – retorna la fecha y hora actual
- FORMAT() – formatea el campo a mostrarse

La Función AVG()

El AVG() devuelve el valor promedio de una columna numérica..

Sintaxis

```
SELECT AVG(column_name) FROM tabla_name
```

SQL AVG() Ejemplo

Tabla "Orden":

O_Id	Orden_Fecha	Orden_Precio	Clientes
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Hansen
4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

Se buscará el promedio de los valores de los campos de "Orden_Precio".

Se usa la siguiente sentencia SELECT:

```
SELECT AVG(Orden_Precio) AS Promedio FROM Orden
```

El conjunto de resultados es:

Promedio
950

Se buscará los clientes que tienen un valor superior al promedio Orden_Precio valor.

Se usa la siguiente sentencia SELECT:

```
SELECT Clientes FROM Orden
WHERE Orden_Precio>(SELECT AVG(Orden_Precio) FROM Orden)
```

El conjunto de resultados es:

Clientes
Hansen
Nilsen
Jensen

SQL COUNT(column_name) Sintaxis

La función COUNT (column_name) retorna el número de valores (los valores NULL no serán contados) de una columna específica:

```
SELECT COUNT(column_name) FROM tabla_name
```

SQL COUNT(*) Sintaxis

La función COUNT(*) retorna el número de registros en la tabla:

```
SELECT COUNT(*) FROM tabla_name
```

COUNT (DISTINCT column_name) Sintaxis

La función COUNT (DISTINCT column_name) retorna el número de los valores distintos de una columna específica:

```
SELECT COUNT(DISTINCT column_name) FROM tabla_name
```

Note: COUNT (DISTINCT) trabaja con ORACLE y Microsoft SQL Server, pero no con Microsoft Access.

COUNT (column_name)

Se tienen la siguiente tabla “Orden":

O_Id	**Orden_Fecha**	**Orden_Precio**	**Clientes**
1	2008/11/12	1000	Hansen

2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Hansen
4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

Se contará el numero de ordenes de la tabla "Clientes Nilsen".

Se usa la siguiente sentencia:

```
SELECT COUNT(Clientes) AS ClientesNilsen FROM Orden
WHERE Clientes='Nilsen'
```

El resultado de la sentencia será 2, porque el cliente Nilsen ha hecho 2 órdenes en total:

ClientesNilsen
2

SQL COUNT(*) Ejemplo

Si se omite la clausula WHERE:

```
SELECT COUNT(*) AS NumerodeOrden FROM Orden
```

El conjunto de resultados es:

NumerodeOrden
6

Da como resultado el número total de filas en la tabla.

COUNT(DISTINCT column_name) Ejemplo

Se contara el número de clientes únicos en la tabla "Órdenes".

Utilizamos la siguiente declaración SQL

```
SELECT COUNT(DISTINCT Clientes) AS NumerodeClientes FROM Orden
```

El conjunto de resultados es:

NumerodeClientes
3

Es el número de clientes (Hansen, Nilsen, y Jensen) en la tabla "Órdenes".

La función FIRST()

La función FIRST() retorna el primer valor de la columna seleccionada.

FIRST() Sintaxis

```
SELECT FIRST(column_name) FROM tabla_name
```

Ejemplo FIRST()

Se tienen la siguiente tabla “Orden" tabla:

O_Id	Orden_Fecha	Orden_Precio	Clientes
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Hansen
4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

Se mostrara el primer valor de la columna "Orden_Precio".

Se usa la siguiente sentencia SQL:

```
SELECT FIRST(Orden_Precio) AS PrimerOrden_Precio FROM Orden
```

El conjunto de resultados es:

PrimerOrden_Precio
1000

La función LAST()

La función LAST() retorna el ultimo valor de la columna seleccionada .

Sintaxis LAST()

```
SELECT LAST(column_name) FROM tabla_name
```

Ejemplo LAST()

Se tienen la siguiente tabla “Orden":

O_Id	Orden_Fecha	Orden_Precio	Clientes
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen

3	2008/09/02	700	Hansen
4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

Se mostrará el último valor de la columna "Orden_Precio".

Se usa la siguiente sentencia SQL :

```
SELECT LAST(Orden_Precio) AS UltimaOrden_Precio FROM Orden
```

El conjunto de resultados es:

UltimaOrden_Precio
100

La función MAX()

La función MAX() retorna el valor más grande de la columna seleccionada

Sintaxis MAX()

```
SELECT MAX(column_name) FROM tabla_name
```

Ejemplo MAX()

Se tienen la siguiente tabla “Orden" tabla:

O_Id	**Orden_Fecha**	**Orden_Precio**	**Clientes**
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Hansen
4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

Se mostrará el valor más alto de la columna "Orden_Precio".

Se usa la siguiente sentencia SQL:

```
SELECT MAX(Orden_Precio) AS MaximoOrden_Precio FROM Orden
```

El conjunto de resultados es:

MaximoOrden_Precio
2000

La función MIN()

La función MIN() retorna el valor más bajo de la columna seleccionada.

Sintaxis MIN()

```
SELECT MIN(column_name) FROM tabla_name
```

SQL MIN() Ejemplo

Se tienen la siguiente tabla “Orden":

O_Id	Orden_Fecha	Orden_Precio	Clientes
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Hansen
4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

Se mostrará el valor más bajo de la columna "Orden_Precio".

Se usa la siguiente sentencia SQL:

```
SELECT MIN(Orden_Precio) AS MinimoOrden_Precio FROM Orden
```

El conjunto de resultados es:

MinimoOrden_Precio
100

La sentencia GROUP BY

La sentencia GROUP BY es usada en conjunción con la función de agregado para agrupar el conjunto de resultado de uno o más columnas.

Sintaxis GROUP BY

```
SELECT column_name, aggregate_function(column_name)
FROM tabla_name
WHERE column_name operator value
GROUP BY column_name
```

Ejemplo GROUP BY

Se tienen la siguiente tabla “Orden":

O_Id	Orden_Fecha	Orden_Precio	Clientes
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Hansen
4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

Se realizará la suma total (total order) de cada cliente.

Se tendrá que usar la sentencia GROUP BY para agrupar a los clientes.

Se usa la siguiente sentencia SQL:

```
SELECT Clientes,SUM(Orden_Precio) FROM Orden
GROUP BY Clientes
```

El conjunto de resultados es:

Clientes	SUM(Orden_Precio)
Hansen	2000
Nilsen	1700
Jensen	2000

Se verá qué pasa si se omite la sentencia GROUP BY:

```
SELECT Clientes,SUM(Orden_Precio) FROM Orden
```

El conjunto de resultados es:

Clientes	SUM(Orden_Precio)
Hansen	5700
Nilsen	5700
Hansen	5700
Hansen	5700
Jensen	5700
Nilsen	5700

El set de resultado obtenido no es el esperado.

Explicación de por qué la anterior sentencia SELECT no puede ser utilizada: La sentencia SELECT mostrada tiene dos columnas especificadas (cliente y SUM(Orden_Precio).

El "SUM(Orden_Precio)" devuelve un valor simple (que es la suma total de la columna "Orden_Precio"), mientras que "cliente" devuelve 6 valores (un valor para cada fila en la tabla "Órdenes"). Por lo tanto no nos dan el resultado correcto. Sin embargo, se ha visto que el grupo por declaración soluciona este problema

GROUP BY más de una columna

Se puede usar también la sentencia GROUP BY sobre más de una columnas:

```
SELECT Clientes,Orden_Fecha,SUM(Orden_Precio) FROM Orden
GROUP BY Clientes, Orden_Fecha
```

La clausula HAVING

La cláusula HAVING fue agregado al SQL porque la palabra clave WHERE no puede ser usado con funciones de agregado.

Sintaxis HAVING

```
SELECT column_name, aggregate_function(column_name)
FROM tabla_name
WHERE column_name operator value
GROUP BY column_name
HAVING aggregate_function(column_name) operator value
```

Ejemplo HAVING

Se tienen la siguiente tabla “Orden" tabla:

O_Id	Orden_Fecha	Orden_Precio	Clientes
1	2008/11/12	1000	Hansen
2	2008/10/23	1600	Nilsen
3	2008/09/02	700	Hansen
4	2008/09/03	300	Hansen
5	2008/08/30	2000	Jensen
6	2008/10/04	100	Nilsen

Se mostrara si algún cliente tiene una orden total menor a 2000.

Se usa la siguiente sentencia SQL:

```
SELECT Clientes,SUM(Orden_Precio) FROM Orden
GROUP BY Clientes
```

```
HAVING SUM(Orden_Precio)<2000
```

El conjunto de resultados es:

Clientes	SUM(Orden_Precio)
Nilsen	1700

Se seleccionará si los clientes "Hansen" o "Jensen" tienen una total de orden de más de 1500.

Se agrega un clausula WHERE ordinaria para la sentencia SQL:

```
SELECT Clientes,SUM(Orden_Precio) FROM Orden
WHERE Clientes='Hansen' OR Clientes='Jensen'
GROUP BY Clientes
HAVING SUM(Orden_Precio)>1500
```

El conjunto de resultados es:

Clientes	SUM(Orden_Precio)
Hansen	2000
Jensen	2000

La función UCASE()

La función UCASE() convierte el valor del campo a mayúscula.

Sintaxis UCASE()

```
SELECT UCASE(column_name) FROM tabla_name
```

Ejemplo UCASE()

Se tienen la siguiente tabla “Personas":

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Se seleccionará el contenido de las columnas "Apellido" y "Nombre", y convierte la columna "Apellido" en mayúscula.

Se usa la siguiente sentencia SELECT:

```
SELECT UCASE(Apellido) as Apellido,Nombre FROM Personas
```

El conjunto de resultados es:

Apellido	**Nombre**
HANSEN	Ola
SVENDSON	Tove
PETTERSEN	Kari

La función LCASE()

La función LCASE() convierte el valor del campo a minúscula.

Sintaxis LCASE()

```
SELECT LCASE(column_name) FROM tabla_name
```

Ejemplo LCASE()

Se tienen la siguiente tabla “Personas":

P_Id	**Apellido**	**Nombre**	**Dirección**	**Ciudad**
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Se seleccionará el contenido de las columnas "Apellido" y "Nombre", y convierte la columna "Apellido" en minúscula.

Se usa la siguiente sentencia SELECT:

```
SELECT  LCASE(Apellido) as Apellido,Nombre FROM Personas
```

El conjunto de resultados es:

Apellido	**Nombre**
hansen	Ola
svendson	Tove
pettersen	Kari

La función MID()

La función MID() es usado para extraer caracteres de un campo texto.

Sintaxis MID()

```
SELECT MID(column_name,start[,length]) FROM tabla_name
```

Parametro	Descripción
column_name	Requerido. El campo a extraer los caracteres
start	Requerido. Especifica la posición de inicio (comienza en 1)
length	Opcional. El numero de caracteres que retorna, si se omite, La función MID() retorna el reto del texto.

Ejemplo MID()

Se tienen la siguiente tabla “Personas" tabla:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Se mostrará como extraer los primeros cuatro caracteres de la columna "Ciudad".

Se usa la siguiente sentencia SELECT:

```
SELECT MID(Ciudad,1,4) as SmallCiudad FROM Personas
```

El conjunto de resultados es:

SmallCiudad
Sand
Sand
Stav

La función LEN()

La función LEN() retorna el tamaño del texto.

Sintaxis LEN()

```
SELECT LEN(column_name) FROM tabla_name
```

Ejemplo LEN()

Se tienen la siguiente tabla “Personas":

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes

2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Se seleccionará el tamaño de los valores en la columna "Dirección".

Se usa la siguiente sentencia SELECT:

```
SELECT LEN(Dirección) as TamañodeDirección FROM Personas
```

El conjunto de resultados es:

TamañodeDirección
12
9
9

La función ROUND()

La función ROUND() es usada para redondear un campo numérico a un numero decimal especifico.

SQL ROUND() Sintaxis

```
SELECT ROUND(column_name,decimals) FROM tabla_name
```

Parametro	**Descripción**
column_name	Requerido. Campo a redondear.
decimal	Requerido. Numero de decimales a utilizar.

Ejemplo ROUND()

Se tienen la siguiente tabla “Productos":

Prod_Id	**Producto**	**Unidad**	**Unidadprecio**
1	Jarlsberg	1000 g	10.45
2	Mascarpone	1000 g	32.56
3	Gorgonzola	1000 g	15.67

Se mostrará los nombres de productos y precios redondeados a números enteros.

Se usa la siguiente sentencia SELECT:

```
SELECT Producto, ROUND(Unidadprecio,0) as Unidadprecio FROM Productos
```

El conjunto de resultados es:

Producto	**Unidadprecio**
Jarlsberg	10
Mascarpone	33
Gorgonzola	16

La función NOW()

La función NOW() retorna la Fecha y hora actual.

Sintaxis NOW()

```
SELECT NOW() FROM tabla_name
```

Ejemplo NOW()

Se tienen la siguiente tabla “Productos":

Prod_Id	**Producto**	**Unidad**	**Unidadprecio**
1	Jarlsberg	1000 g	10.45
2	Mascarpone	1000 g	32.56
3	Gorgonzola	1000 g	15.67

Se mostrará el producto y los precios por hoy.

Se usa la siguiente sentencia SELECT:

```
SELECT Producto, Unidadprecio, Now() as PorDia FROM Productos
```

El conjunto de resultados es:

Producto	**Unidadprecio**	**PorDia**
Jarlsberg	10.45	10/7/2008 11:25:02 AM
Mascarpone	32.56	10/7/2008 11:25:02 AM
Gorgonzola	15.67	10/7/2008 11:25:02 AM

La función FORMAT()

La función FORMAT() es usada para formatear como un campo es mostrado.

Sintaxis FORMAT()

```
SELECT FORMAT(column_name,format) FROM tabla_name
```

Parámetro	Descripción
column_name	Requerido. El campo a ser formateado.
format	Requerido. Especifica el formato.

Ejemplo FORMAT()

Se tienen la siguiente tabla “Productos":

Prod_Id	Producto	Unidad	Unidadprecio
1	Jarlsberg	1000 g	10.45
2	Mascarpone	1000 g	32.56
3	Gorgonzola	1000 g	15.67

Se mostraran los productos y precios con la fecha de hoy (con la fecha mostrada con el siguiente formato "YYYY-MM-DD").

Se usa la siguiente sentencia SELECT:

```
SELECT Producto, Unidadprecio, FORMAT(Now(),'YYYY-MM-DD') as PorDia
FROM Productos
```

El conjunto de resultados es:

Producto	Unidadprecio	PorDia
Jarlsberg	10.45	2008-10-07
Mascarpone	32.56	2008-10-07
Gorgonzola	15.67	2008-10-07

La sentencia INSERT INTO

La sentencia INSERT INTO es usada para insertar una nueva fila a una tabla.

Sintaxis INSERT INTO

Se puede escribir esta sentencia de dos formas.

La primera forma no especifica los nombres de las columnas donde los datos serán insertados, solo los valores:

```
INSERT INTO tabla_name
VALUES (value1, value2, value3,...)
```

La segunda forma específica ambos, los nombres de las columnas y los valores a insertar:

```
INSERT INTO tabla_name (column1, column2, column3,...)
VALUES (value1, value2, value3,...)
```

SQL INSERT INTO Ejemplo

Se tienen la siguiente tabla “Personas":

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Se insertará una nueva fila en la tabla "Personas".

Se usara la siguiente sentencia SQL:

```
INSERT INTO Personas
VALUES (4,'Nilsen', 'Johan', 'Bakken 2', 'Stavanger')
```

La tabla “Personas" quedará así:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger

Insertar datos son en columnas específicas

Es posible agregar solo datos a algunas columnas específicas.

La siguiente sentencia SQL agregará una nueva fila, pero solo agregara datos en las columnas "P_Id", "Apellido" y "Nombre":

```
INSERT INTO Personas (P_Id, Apellido, Nombre)
VALUES (5, 'Tjessem', 'Jakob')
```

La tabla "Personas" lucirá como la siguiente:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

4	Nilsen	Johan	Bakken 2	Stavanger
5	Tjessem	Jakob		

La sentencia UPDATE

La sentencia UPDATE es usada para modificar los registros en una tabla.

```
UPDATE tabla_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

Nota: Aviso, sobre la cláusula WHERE en la actualización. La cláusula WHERE especifica que registros deben actualizarse. Si omite la cláusula WHERE, se actualizarán todos los registros.

Ejemplo UPDATE

La Tabla “Personas”:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger
5	Tjessem	Jakob		

Se modificará la persona "Tjessem, Jakob" en la tabla "Personas".

Se usará la siguiente sentencia SQL:

```
UPDATE Personas
SET Dirección='Nissestien 67', Ciudad='Sandnes'
WHERE Apellido='Tjessem' AND Nombre='Jakob'
```

La tabla "Personas" lucirá así:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger
5	Tjessem	Jakob	Nissestien 67	Sandnes

Sentencia UPDATE

Se debe ser cuidadoso cuando se modifican registros. Si se omite la clausula Where ocurre lo siguiente:

```
UPDATE Personas
SET Dirección='Nissestien 67', Ciudad='Sandnes'
```

La tabla "Personas" modificada:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Nissestien 67	Sandnes
2	Svendson	Tove	Nissestien 67	Sandnes
3	Pettersen	Kari	Nissestien 67	Sandnes
4	Nilsen	Johan	Nissestien 67	Sandnes
5	Tjessem	Jakob	Nissestien 67	Sandnes

La sentencia DELETE

La sentencia DELETE es usada para borrar filas en una tabla.

Sintaxis DELETE

```
DELETE FROM tabla_name
WHERE some_column=some_value
```

Note: Aviso. La cláusula WHERE especifica que registros se deben suprimirse. Si omite la cláusula WHERE, serán eliminados todos los registros

Ejemplo DELETE

La Tabla “Personas”:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger
5	Tjessem	Jakob	Nissestien 67	Sandnes

Se borrara la persona “Tjessem, Jakob" en la tabla "Personas".

Se usa la siguiente sentencia SQL:

```
DELETE FROM Personas
```

```
WHERE Apellido='Tjessem' AND Nombre='Jakob'
```

La tabla "Personas":

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger

Delete Todas la Filas

Se puede borrar todas las filas sin borrar la tabla. Esto significa que la estructura de la tabla, atributos e índices permanecerán intactos:

```
DELETE FROM tabla_name


DELETE * FROM tabla_name
```

Note: Se debe tener cuidado cuando se borra los registros. No se puede volver atrás esta sentencia.

La sentencia CREATE DATABASE

La sentencia CREATE DATABASE es usado para crear una base de datos.

SQL CREATE DATABASE Sintaxis

```
CREATE DATABASE databasE_Nombre
```

Ejemplo CREATE DATABASE

Se creará una base de datos llamada "my_db".

Se usa la siguiente sentencia CREATE DATABASE:

```
CREATE DATABASE my_db
```

Las tablas serán agregadas con la sentencia CREATE TABLA.

La sentencia CREATE TABLA

La sentencia CREATE TABLA es usada para crear una tabla en una base de datos.

Sintaxis CREATE TABLA

```
CREATE TABLA tabla_name
(
column_name1 data_type,
column_name2 data_type,
column_name3 data_type,
....
)
```

El tipo de datos especifica qué tipo de datos puede mantener la columna. Para una referencia completa de todos los tipos de datos disponibles en MS Access, MySQL, y SQL Server, ir a Tipos referencia al final del documento.

Ejemplo CREATE TABLA

Se creara una tabla llamada "Personas" que contiene cinco columnas: P_Id, Apellido, Nombre, Dirección, y Ciudad.

Se usara la siguiente sentencia CREATE TABLA:

```
CREATE TABLA Personas
(
P_Id int,
Apellido varchar(255),
Nombre varchar(255),
Dirección varchar(255),
Ciudad varchar(255)
)
```

La columna P_Id es de tipo entero y contendrá un número. Las columnas Apellido, Nombre, Dirección, y Ciudad son del tipo varchar con un máximo tamaño de 255 caracteres.

La tabla "Personas" vacía se verá así:

P_Id	Apellido	Nombre	Dirección	Ciudad

La tabla vacía se llenará con la sentencia INSERT INTO.

SQL Restricciones

Las restricciones son usadas para limitar los tipos de datos que pueden ir en una tabla.,

Las restricciones pueden ser especificadas cuando una tabla es creada (con la sentencia CREATE TABLA) o después de crear la tabla con la sentencia ALTER TABLA.

Lista de restricciones:

- NOT NULL
- UNIQUE

- PRIMARY KEY
- FOREIGN KEY
- CHECK
- DEFAULT

Restricción NOT NULL

La restricción NOT NULL fuerza a la columna a no aceptar valores nulos.

La restricción NOT NULL fuerza a los campos a contener siempre. Esto significa que no se puede insertar un Nuevo registro o modificar un registro sin agregar un valor a este campo.

La siguiente restricción SQL la columna "P_Id" y la columna "Apellido" no aceptan valores nulos

```
CREATE TABLA Personas
(
P_Id int NOT NULL,
Apellido varchar(255) NOT NULL,
Nombre varchar(255),
Dirección varchar(255),
Ciudad varchar(255)
)
```

Restricción UNIQUE

La restricción UNIQUE identifica la unicidad de cada registro en la tabla de la base de datos.

Las restricciones UNIQUE y PRIMARY KEY proveen la garantía de unicidad para una columna o grupo de columna.

Una restricción de PRIMARY KEY automáticamente tiene una restricción UNIQUE definida sobre esto.

Note que usted puede tener muchas restricciones por tablas , pero solo una clave primaria por tabla.

Restricción UNIQUE sobre CREATE TABLE

La siguiente sentencia SQL crea una restricción UNIQUE sobre la columna "P_Id" cuando la tabla "Personas" es creada:

MySQL:

```
CREATE TABLA Personas
(
P_Id int NOT NULL,
Apellido varchar(255) NOT NULL,
Nombre varchar(255),
```

```
Dirección varchar(255),
Ciudad varchar(255),
UNIQUE (P_Id)
)
```

SQL Server / Oracle / MS Access:

```
CREATE TABLA Personas
(
P_Id int NOT NULL UNIQUE,
Apellido varchar(255) NOT NULL,
Nombre varchar(255),
Dirección varchar(255),
Ciudad varchar(255)
)
```

Para permitir nombres de una restricción UNIQUE, y para definir una restricción UNIQUE en varias columnas, se usa la siguiente Sintaxis SQL:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLA Personas
(
P_Id int NOT NULL,
Apellido varchar(255) NOT NULL,
Nombre varchar(255),
Dirección varchar(255),
Ciudad varchar(255),
CONSTRAINT uc_PersonID UNIQUE (P_Id,Apellido)
)
```

SQL UNIQUE restricción sobre ALTER TABLE

Para crear una restricción UNIQUE sobre la columna "P_Id" cuando la tabla es creada se usa la siguiente sentencia SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLA Personas
ADD UNIQUE (P_Id)
```

Para permitir nombres de una restricción UNIQUE, y para definir una restricción UNIQUE en varias columnas, se usa la siguiente Sintaxis SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLA Personas
```

```
ADD CONSTRAINT uc_PersonID UNIQUE (P_Id,Apellido)
```

Borrar la restricción UNIQUE

Para borrar una restricción UNIQUE se usa:

MySQL:

```
ALTER TABLA Personas
DROP INDEX uc_PersonID
```

SQL Server / Oracle / MS Access:

```
ALTER TABLA Personas
DROP CONSTRAINT uc_PersonID
```

Restricción PRIMARY KEY

La restricción PRIMARY KEY identifica cada registro como único en una tabla de una base de datos.

Primary keys debe contener valores únicos.

La columna de primary key no contiene valores nulos.

Cada tabla debe tener una clave primaria , y cada tabla puede tener solo una clave primaria.

Restricción PRIMARY KEY sobre CREATE TABLA

La siguiente sentencia crea una PRIMARY KEY sobre la columna "P_Id" cuando la tabla "Personas" es creado:

MySQL:

```
CREATE TABLA Personas
(
P_Id int NOT NULL,
Apellido varchar(255) NOT NULL,
Nombre varchar(255),
Dirección varchar(255),
Ciudad varchar(255),
PRIMARY KEY (P_Id) )
```

SQL Server / Oracle / MS Access:

```
CREATE TABLA Personas
(
```

```
P_Id int NOT NULL PRIMARY KEY,
Apellido varchar(255) NOT NULL,
Nombre varchar(255),
Dirección varchar(255),
Ciudad varchar(255)
)
```

Para dar un nombre de una restricción PRIMARY KEY, y para definir una restricción PRIMARY KEY en múltiples columnas, utilizar la siguiente Sintaxis SQL:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLA Personas
(
P_Id int NOT NULL,
Apellido varchar(255) NOT NULL,
Nombre varchar(255),
Dirección varchar(255),
Ciudad varchar(255),
CONSTRAINT pk_PersonID PRIMARY KEY (P_Id,Apellido)
)
```

Restricción PRIMARY KEY en ALTER TABLE

Para crear una restricción PRIMARY KEY sobre la columna "P_Id" cuando la tabla es recientemente creada se usa la siguiente sentencia SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLA Personas
ADD PRIMARY KEY (P_Id)
```

Para dar nombres a una restricción PRIMARY KEY , y para definir una restricción PRIMARY KEY en varias columnas, utilizar la siguiente Sintaxis SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLA Personas
ADD CONSTRAINT pk_PersonID PRIMARY KEY (P_Id,Apellido)
```

Note: Si se usa la sentencia ALTER TABLA para agregar una clave primaria, las columnas con clave primaria deben haber sido declarados para que no contengan valores nulos cuando la tabla es creada inicialmente.

Borrar una restricción PRIMARY KEY

Para borrar la restricción PRIMARY KEY, usa la siguiente sentencia SQL:

MySQL:

```
ALTER TABLA Personas
DROP PRIMARY KEY
```

SQL Server / Oracle / MS Access:

```
ALTER TABLA Personas
DROP CONSTRAINT pk_PersonID
```

Restricción FOREIGN KEY

Una FOREIGN KEY en una tabla apunta a una PRIMARY KEY en otra tabla.

Ejemplo de la FOREIGN KEY:

La Tabla “Personas”:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

La tabla “Orden”:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	2
4	24562	1

La columna "P_Id" en la tabla "Orden" apunta a la columna "P_Id" en la tabla “Personas".

La columna "P_Id" en la tabla "Personas" es la PRIMARY KEY en la tabla "Personas".

La columna "P_Id" en la tabla "Orden" es una FOREIGN KEY en la tabla "Orden".

La restricción FOREIGN KEY es usada para prevenir acciones que podrían destruir las relaciones entre tablas.

La restricción FOREIGN KEY también previene que datos inválidos sean insertados en la columna de clave foránea, porque esto has sido uno de los valores contenidos en la tabla a la que apunta.

Restricción FOREIGN KEY sobre CREATE TABLA

La siguiente sentencia crea una FOREIGN KEY sobre la columna "P_Id" cuando la tabla "Orden" es creada:

MySQL:

```
CREATE TABLA Orden
(
O_Id int NOT NULL,
OrderNo int NOT NULL,
P_Id int,
PRIMARY KEY (O_Id),
FOREIGN KEY (P_Id) REFERENCES Personas(P_Id)
)
```

SQL Server / Oracle / MS Access:

```
CREATE TABLA Orden
(
O_Id int NOT NULL PRIMARY KEY,
OrderNo int NOT NULL,
P_Id int FOREIGN KEY REFERENCES Personas(P_Id)
)
```

Para permitir la designación de una FOREIGN KEY, y para la definición de una FOREIGN KEY en varias columnas, utilice el siguiente Sintaxis SQL:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLA Orden
(
O_Id int NOT NULL,
OrderNo int NOT NULL,
P_Id int,
PRIMARY KEY (O_Id),
CONSTRAINT fk_PerOrden FOREIGN KEY (P_Id)
REFERENCES Personas(P_Id)
)
```

Restricciones FOREIGN KEY sobre ALTER TABLA

Para crear una restricción FOREIGN KEY sobre la columna "P_Id" cuando la tabla "Orden" ya ha sido creada se utiliza la siguiente sentencia:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLA Orden
ADD FOREIGN KEY (P_Id)
REFERENCES Personas(P_Id)
```

Para permitir el nombramiento de la restricción FOREIGN KEY y para definir la restricción FOREIGN KEY sobre múltiples columnas, se usa las siguiente sintaxis SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLA Orden
ADD CONSTRAINT fk_PerOrden
FOREIGN KEY (P_Id)
REFERENCES Personas(P_Id)
```

Borrar una restricción FOREIGN KEY

Para borrar una restricción FOREIGN KEY, se usa la siguiente sentencia SQL:

MySQL:

```
ALTER TABLA Orden
DROP FOREIGN KEY fk_PerOrden
```

SQL Server / Oracle / MS Access:

```
ALTER TABLA Orden
DROP CONSTRAINT fk_PerOrden
```

Restricción CHECK

La restricción CHECK es usada para limitar el rango de valores que puede ser ingresado en una columna.

Si se define la restricción CHECK sobre una columna permitirá solo asegurar los valores para esta columna.

Si se define una restricción CHECK sobre una tabla, esto puede limitar los valores en ciertas columnas basadas sobre valores en otras columnas en las filas.

Restricción CHECK sobre CREATE TABLA

La siguiente sentencia SQL crea una restricción CHECK sobre la columna "P_Id" cuando la tabla "Personas" es creada. La restricción CHECK especifica que la columna "P_Id" debe solo incluir enteros mayores a 0.

My SQL:

```
CREATE TABLA Personas
(
P_Id int NOT NULL,
Apellido varchar(255) NOT NULL,
Nombre varchar(255),
Dirección varchar(255),
Ciudad varchar(255),
```

```
CHECK (P_Id>0)
)
```

SQL Server / Oracle / MS Access:

```
CREATE TABLA Personas
(
P_Id int NOT NULL CHECK (P_Id>0),
Apellido varchar(255) NOT NULL,
Nombre varchar(255),
Dirección varchar(255),
Ciudad varchar(255)
)
```

Para permitir el nombramiento de la restricción CHECK y para definir la restricción CHECK sobre múltiples columnas, se usa las siguiente sintaxis SQL:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLA Personas
(
P_Id int NOT NULL,
Apellido varchar(255) NOT NULL,
Nombre varchar(255),
Dirección varchar(255),
Ciudad varchar(255),
CONSTRAINT chk_Person CHECK (P_Id>0 AND Ciudad='Sandnes')
)
```

Restricción CHECK sobre ALTER TABLA

Para crear una restricción sobre la columna “P_Id" cuando la tabla ya ha sido creada se usa las siguiente sentencia:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLA Personas
ADD CHECK (P_Id>0)
```

Para permitir el nombramiento de la restricción CHECK y para definir la restricción CHECK sobre múltiples columnas, se usa las siguiente sintaxis SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLA Personas
ADD CONSTRAINT chk_Person CHECK (P_Id>0 AND Ciudad='Sandnes')
```

Borrar una restricción CHECK

Para borrar una restricción CHECK usa la siguiente sentencia:

SQL Server / Oracle / MS Access:

```
ALTER TABLA Personas
DROP CONSTRAINT chk_Person
```

Restricción DEFAULT

La restricción DEFAULT es usada para insertar un valor por defecto dentro de una columna.

Los valores por defecto será argado para todo registro nuevo, si no hay otro valor especificado.

Restricción DEFAULT sobre CREATE TABLA

La siguiente sentencia SQL crea una restricción DEFAULT sobre la columna "Ciudad" cuando la tabla "Personas" es creada:

My SQL / SQL Server / Oracle / MS Access:

```
CREATE TABLA Personas
(
P_Id int NOT NULL,
Apellido varchar(255) NOT NULL,
Nombre varchar(255),
Dirección varchar(255),
Ciudad varchar(255) DEFAULT 'Sandnes'
)
```

La restricción DEFAULT puede también ser usado para insertar valores del sistema como ser la función GETDATE ():

```
CREATE TABLA Orden
(
O_Id int NOT NULL,
OrderNo int NOT NULL,
P_Id int,
Orden_Fecha date DEFAULT GETDATE()
)
```

Restricción DEFAULT sobre ALTER TABLA

Para crear una restricción DEFAULT sobe la columna "Ciudad" cuando la tabla ya ha sido creada, se usara la siguiente sentencia SQL:

MySQL:

```
ALTER TABLA Personas
```

```
ALTER Ciudad SET DEFAULT 'SANDNES'
```

SQL Server / Oracle / MS Access:

```
ALTER TABLA Personas
ALTER COLUMN Ciudad SET DEFAULT 'SANDNES'
```

Borrar DROP una restricción DEFAULT

Para borrar una restricción DEFAULT, usa la siguiente sentencia SQL:

MySQL:

```
ALTER TABLA Personas
ALTER Ciudad DROP DEFAULT
```

SQL Server / Oracle / MS Access:

```
ALTER TABLA Personas
ALTER COLUMN Ciudad DROP DEFAULT
```

Índices

Un índice puede ser creado en una tabla para buscar datos más rápido y eficientemente.

El usuario no puede ver los índices, ellos son solos usados para acelerar las búsqueda.

Nota: modificando una tabla con índices toma más tiempo que modificarla sin índice (porque los índices también necesitan modificarse). También se debería solo crear índices sobre columnas y tablas que serán frecuentemente consultadas.

Sintaxis CREATE INDEX

Crear un índice sobre una tabla. Se permiten valores duplicados:

```
CREATE INDEX index_name
ON tabla_name (column_name)
```

Sintaxis CREATE UNIQUE INDEX

Crear un índice sobre una tabla. No se permiten valores duplicados:

```
CREATE UNIQUE INDEX index_name
ON tabla_name (column_name)
```

Note: La sintaxis para crear índices entre varios base de datos diferentes. Por lo tanto: se chequea las sintaxis para crear los índices en la base de datos.

Ejemplo CREATE INDEX

La sentencia SQL creara un índice llamado "PIndex" sobre la columna "Apellido" en la Tabla “Personas”:

```
CREATE INDEX PIndex
ON Personas (Apellido)
```

Si se quiere crear un índice sobre una combinación de columnas , se puede listar los nombres de columnas entre paréntesis separados por comas:

```
CREATE INDEX PIndex
ON Personas (Apellido, Nombre)
```

La sentencia borrar Índices

Sentencia para borrar INDICES en una tabla.

Sintaxis DROP INDEX para MS Access:

```
DROP INDEX index_name ON tabla_name
```

Sintaxis DROP INDEX para MS SQL Server:

```
DROP INDEX tabla_name.index_name
```

Sintaxis DROP INDEX para DB2/Oracle:

```
DROP INDEX index_name
```

Sintaxis DROP INDEX para MySQL:

```
ALTER TABLA tabla_name DROP INDEX index_name
```

Sentencia DROP TABLA

La sentencia DROP TABLA es usada para borrar una tabla.

```
DROP TABLA tabla_name
```

La sentencia DROP DATABASE

La sentencia DROP DATABASE es usada para borrar una Base de Datos.

```
DROP DATABASE databasE_Nombre
```

La sentencia TRUNCATE TABLA

¿Qué pasa si sólo quiere borrar los datos dentro de la tabla, y no la tabla en sí

Entonces, se usa la sentencia TRUNCATE TABLA:

```
TRUNCATE TABLA tabla_name
```

La sentencia ALTER TABLA

La sentencia ALTER TABLA es usada para agregar, borrar, o modificar columnas en una tabla existente.

Sintaxis ALTER TABLA

Agregar una columna en una tabla:

```
ALTER TABLE tabla_name
ADD column_name datatype
```

Para borrar una columna en una tabla. La Sintaxis (algunos sistemas de base de datos no permiten borrar columnas):

```
ALTER TABLA tabla_name
DROP COLUMN column_name
```

Cambiar los tipos de datos de una columna:

```
ALTER TABLA tabla_name
ALTER COLUMN column_name datatype
```

Ejemplo ALTER TABLA

La Tabla “Personas”:

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Se añadirá una columna llamada "DateOfBirth" en la tabla "Personas".

Sentencia SQL:

```
ALTER TABLA Personas
ADD DateOfBirth date
```

Se observa que la nueva columna, "DateOfBirth", es de tipo date. El tipo de datos especifica qué tipo de datos la columna puede contener.

La tabla "Personas”:

P_Id	Apellido	Nombre	Dirección	Ciudad	DateOfBirth
1	Hansen	Ola	Timoteivn 10	Sandnes	
2	Svendson	Tove	Borgvn 23	Sandnes	
3	Pettersen	Kari	Storgt 20	Stavanger	

Ejemplo cambio tipo de Dato

Cambiar el tipo de datos de la columna llamada "DateOfBirth" en la tabla "Personas".

Sentencia SQL:

```
ALTER TABLA Personas
ALTER COLUMN DateOfBirth year
```

Se observa que la columna "DateOfBirth" es del tipo year y alojara un año con dos dígitos o cuatro dígitos.

Ejemplo DROP COLUMN

Se quiere borrar la columna "DateOfBirth" en la tabla "Personas".

Sentencia SQL:

```
ALTER TABLA Personas
DROP COLUMN DateOfBirth
```

La tabla "Personas":

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

El campo AUTO INCREMENT

A menudo se quiere que el valor de una clave primaria se incremente automáticamente cuando un registro nuevo es insertado.

Se creara un campo en la tabla auto incrementado

Sintaxis para MySQL

Sentencia SQL para definir que la columna "P_Id" como clave primaria y autoincremental en la Tabla "Personas":

```
CREATE TABLA Personas
(
P_Id int NOT NULL AUTO_INCREMENT,
Apellido varchar(255) NOT NULL,
Nombre varchar(255),
Dirección varchar(255),
Ciudad varchar(255),
```

```
PRIMARY KEY (P_Id))
```

MySQL usa la palabra clave AUTO_INCREMENT para llevar a cabo el incremento automático del campo.

Por defecto, el valor de comienzo para AUTO_INCREMENT es 1, y esto incrementara en uno el valor en cada nuevo registro.

Para que la secuencia AUTO_INCREMENT comience con otro valor, use la siguiente sentencia SQL:

```
ALTER TABLA Personas AUTO_INCREMENT=100
```

Para insertar un nuevo registro en la tabla "Personas", No tendremos a especificar un valor para la columna "P_Id (un valor único será agregado automáticamente)):

```
INSERT INTO Personas (Nombre,Apellido)
VALUES ('Lars','Monsen')
```

La sentencia SQL insertara un nuevo registro en la tabla "Personas". La columna "P_Id" debería ser cargado con un valor único. La columna "Nombre" debería insertar "Lars" y la columna "Apellido" se insertará "Monsen".

Sintaxis para SQL Server

La siguiente sentencia SQL define la columna "P_Id" para ser una clave primaria autoincremental en la Tabla “Personas”:

```
CREATE TABLA Personas
(
P_Id int PRIMARY KEY IDENTITY,
Apellido varchar(255) NOT NULL,
Nombre varchar(255),
Dirección varchar(255),
Ciudad varchar(255)
)
```

El MS SQL Server usa la palabra clave IDENTITY para generar un rasgo autoincremental.

Por defecto, el valor IDENTITYcomienza en 1, y será incrementado de a uno en cada nuevo registro.

Para especificar que la columna "P_Id" debería comenzar con un valor de 10 e incrementar de a 5, cambiar las identidad a IDENTITY(10,5).

Para insertar un nuevo registro dentro de la tabla "Personas", no se especificará un valor para la columna "P_Id" (un único valor será agregado automáticamente):

```
INSERT INTO Personas (Nombre,Apellido)
VALUES ('Lars','Monsen')
```

La sentencia SQL insertara un nuevo registro dentro de la tabla "Personas". La columna "P_Id" debería ser asignado un único valor. La columna "Nombre" seria cargada con "Lars" y la columna "Apellido" se carga con "Monsen".

Sintaxis para Access

La siguiente sentencia SQL define la columna "P_Id" para ser clave primaria auto incremental en la Tabla “Personas”:

```
CREATE TABLA Personas
(
P_Id PRIMARY KEY AUTOINCREMENT,
Apellido varchar(255) NOT NULL,
Nombre varchar(255),
Dirección varchar(255),
Ciudad varchar(255)))
```

El MS Access usa la palabra clave AUTOINCREMENT para declarar el cambio

Por defecto, el valor de comienzo para AUTO_INCREMENT es 1, y esto incrementara en uno el valor en cada nuevo registro.

Para especificar que la columna "P_Id" columna debería comenzar con el valor 10 y incrementar en 5, Cambiar el autoincremento a AUTOINCREMENT (10,5).

Para insertar un nuevo registro en la tabla "Personas", no se debe especificar un valor para la columna "P_Id" (un único valor será agregado automáticamente):

```
INSERT INTO Personas (Nombre,Apellido)
VALUES ('Lars','Monsen')
```

La sentencia SQL insertara un nuevo registro en la tabla "Personas". La columna "P_Id" debería ser cargado con un valor único. La columna "Nombre" debería insertar "Lars" y la columna "Apellido" se insertará "Monsen".

Sintaxis en Oracle

El código Oracle es un poco más complicado.

Usted tendrá que crear un campo auto-incremento con la secuencia objeto (este objeto genera una secuencia de números).

Usar la siguiente Sintaxis CREATE SEQUENCE:

```
CREATE SEQUENCE seq_person
MINVALUE 1
START WITH 1
INCREMENT BY 1
CACHE 10
```

El código anterior crea una secuencia objeto llamado seq_person, que comienza con 1 y se incrementará en 1. También caché hasta 10 valores para el rendimiento. La opción caché especifica cuales valores de secuencia serán almacenados en la memoria de acceso más rápido.

Para insertar un nuevo récord en la tabla "personas" , tendremos que usar la función nextval (esta función recupera el siguiente valor de seq_person secuencia):

```
INSERT INTO Personas (P_Id,Nombre,Apellido)
VALUES (seq_person.nextval,'Lars','Monsen')
```

La sentencia SQL insertara un nuevo registro en la tabla "Personas". La columna "P_Id" debería ser cargado con un valor único. La columna "Nombre" debería insertar "Lars" y la columna "Apellido" se insertará "Monsen".

Sentencia CREATE VIEW

En SQL, una vista es una tabla virtual basada en un set de resultado de una sentencia SQL.

Una vista contiene filas y columnas como una tabla real. El campo en una vista son campos desde una o más tablas reales en una base de datos.

Se puede agregar a una función SQL, sentencias WHERE, y JOIN para una vista y presente los datos como si los datos vinieran de una tabla simple.

Sintaxis CREATE VIEW

```
CREATE VIEW view_name AS
SELECT column_name(s)
FROM tabla_name
WHERE condition
```

Nota: una vista siempre muestra datos actualizados. El motor de la base de datos recrea los datos usando las sentencias de vista de SQL.

Ejemplos CREATE VIEW

La vista "Current Product List" lista todos los productos actives (productos que nos está descontinuados) desde la tabla “Productos" . La vista es creada con las sentencia SQL:

```
CREATE VIEW [Current Product List] AS
SELECT ProductID,Producto
FROM Productos
WHERE Discontinued=No
```

Se puede consultar la vista así:

```
SELECT * FROM [Current Product List]
```

Una vista que selecciona todas lo productos en el tabal "Productos" con una unidad de precio más alto que el promedio de precios:

```
CREATE VIEW [Productos Above Average Price] AS
SELECT Producto,Unidadprecio
FROM Productos
WHERE Unidadprecio>(SELECT AVG(Unidadprecio) FROM Productos)
```

Se puede consultar la vista así:

```
SELECT * FROM [Productos Above Average Price]
```

Esta vista calcula el total de ventas para cada categoría en 1997. Nótese que esta vista selecciona estos datos desde otra vista llamada "Product Sales for 1997":

```
CREATE VIEW [Category Sales For 1997] AS
SELECT DISTINCT CategoryName,Sum(ProductSales) AS CategorySales
FROM [Product Sales for 1997]
GROUP BY CategoryName
```

Se puede consultar la vista así:

```
SELECT * FROM [Category Sales For 1997]
```

Se puede agregar una condición en la consulta. Se verá el total de ventas solo en la categoría "Beverages":

```
SELECT * FROM [Category Sales For 1997]
WHERE CategoryName='Beverages'
```

Modificando Vistas

Se puede adaptar una vista usando la siguiente sintaxis:

SQL CREATE OR REPLACE VIEW Sintaxis

```
CREATE OR REPLACE VIEW view_name AS
SELECT column_name(s)
FROM tabla_name
WHERE condition
```

Se puede agregar la columna"Category" para las vista "Current Product List". Se modificará la vista así:

```
CREATE VIEW [Current Product List] AS
SELECT ProductID,Producto,Category
FROM Productos
WHERE Discontinued=No
```

Borrar una VISTA

Se puede borrar una vista con la sentencia DROP VIEW.

Sintaxis DROP VIEW

```
DROP VIEW view_name
```

SQL Dates

La mayoría de las dificultades La parte más difícil cuando se trabaja con fechas es el asegurarse de que el formato de la fecha que se están tratando de insertar, coincide con el formato de la columna fecha en la base de datos.

Siempre y cuando los datos contienen sólo la porción de fecha , Las consultas trabajarán como se esperaba. Sin embargo, si contiene una porción tiempo, se vuelve complicada.

Antes de hablar sobre las complicaciones de consulta para fechas, se verá las más importantes funciones incorporadas para trabajar con fechas.

Funciones fecha MySQL

Función	**Descripción**
NOW()	Retorna la actual fecha y hora
CURDATE()	Retorna la actual fecha
CURTIME()	Retorna la actual hora
DATE()	Extrae la parte de la fecha de una expresión fecha.
EXTRACT()	Retorna una simple parte de una fecha /hora
DATE_ADD()	Agrega una especificada intervalo de tiempo
DATE_SUB()	Sustrae una especificada intervalo de tiempo
DATEDIFF()	Retorna el número de días entre dos fechas
DATE_FORMAT()	Muestra date/time en diferente formatos

SQL Server Date Funciones

Función	**Descripción**
GETDATE()	Retorna la actual fecha y hora
DATEPART()	Retorna una simple parte de una fecha /hora
DATEADD()	Agrega o Sustrae una especificada intervalo de tiempo
DATEDIFF()	Retorna el número de días entre dos fechas
CONVERT()	Muestra date/time en diferente formatos

SQL Tipos de datos Fecha

MySQL Viene con los siguientes tipos de datos para almacenar una fecha o una fecha/hora valor en la base de datos:

- DATE - format YYYY-MM-DD
- DATETIME - format: YYYY-MM-DD HH:MM:SS
- TIMESTAMP - format: YYYY-MM-DD HH:MM:SS
- YEAR - format YYYY o YY

SQL Server Viene con los siguientes tipos de datos para almacenar una fecha o una fecha/hora valor en la base de datos:

- DATE - format YYYY-MM-DD
- DATETIME - format: YYYY-MM-DD HH:MM:SS
- SMALLDATETIME - format: YYYY-MM-DD HH:MM:SS
- TIMESTAMP - format: un único numero

Note: los tipos fecha son elegidos para una columna cuando se crea una nueva tabla en una base de datos..

Trabajando con fechas

Se puede comparar dos fechas fácilmente si no está el tiempo involucrado.

Tabla “Orden":

OrderId	**Producto**	**Orden_Fecha**
1	Geitost	2008-11-11
2	Camembert Pierrot	2008-11-09
3	Mozzarella di Giovanni	2008-11-11
4	Mascarpone Fabioli	2008-10-29

Se selecciona los registros con un Orden_Fecha de "2008-11-11".

Se usa la siguiente sentencia SELECT:

```
SELECT * FROM Orden WHERE Orden_Fecha='2008-11-11'
```

El conjunto de resultados es:

OrderId	**Producto**	**Orden_Fecha**
1	Geitost	2008-11-11
3	Mozzarella di Giovanni	2008-11-11

Se asume ahora que la tabla "Orden" incluye los valores de tiempo en la columna "Orden_Fecha":

OrderId	Producto	Orden_Fecha
1	Geitost	2008-11-11 13:23:44
2	Camembert Pierrot	2008-11-09 15:45:21
3	Mozzarella di Giovanni	2008-11-11 11:12:01
4	Mascarpone Fabioli	2008-10-29 14:56:59

Se usa la misma sentencia SELECT:

```
SELECT * FROM Orden WHERE Orden_Fecha='2008-11-11'
```

Pero no se obtendrá el resultado, porque la consulta la realice sobre la fecha sin la parte de tiempo.

Tip: Si se quiere mantener la consulta simple y fácil de mantener, no se debe permitir el uso de tiempo en los datos.

Valores NULL

Si una columna en una tabla es opcional, se puede insertar un nuevo registro o modificar uno existente sin agregar un valor a la columna. Esto significa que el campo será salvado con un valor NULL.

Los valores NULL son tratados de manera diferente de otros valores

.NULL es usado para mantener un lugar para valores desconocidos o inaplicables.

Note: No es posible comparar NULL y valor 0; no son equivalentes.

Trabajando con valores NULL Valúes

Tabla "Personas":

P_Id	Apellido	Nombre	Dirección	Ciudad
1	Hansen	Ola		Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari		Stavanger

Se supone que la columna "Dirección" en la tabla "Personas" es opcional. Esto significa que si se inserta un nuevo registró sin un valor para la columna "Dirección", la columna "Dirección" será guardada con valores NULL .

Como se puede probar el valor NULL?

Esto no es posible comparar el valor NULL con los operadores de comparación =, <, o <>. Se tendrá que usar los operadores IS NULL e IS NOT NULL.

SQL IS NULL

Cómo podemos seleccionar sólo los registros con valores NULL en la " Dirección " en la columna

Se usa el operador IS NULL:

```
SELECT Apellido, Nombre, Dirección FROM Personas
WHERE Dirección IS NULL
```

El conjunto de resultados es:

Apellido	Nombre	Dirección
Hansen	Ola	
Pettersen	Kari	

Tip: usar siempre IS NULL para ver los valores NULL.

SQL IS NOT NULL

Cómo podemos seleccionar sólo los registros con valores no NULL en la " Dirección " en la columna?

Se usa el operador IS NOT NULL:

```
SELECT Apellido, Nombre, Dirección FROM Personas
WHERE Dirección IS NOT NULL
```

El conjunto de resultados es:

Apellido	Nombre	Dirección
Svendson	Tove	Borgvn 23

Función SQL ISNULL (), NVL (), IFNULL () and COALESCE ()

Tabla “Productos":

P_Id	Producto	Unidadprecio	UnidadsInStock	UnidadsOnOrder
1	Jarlsberg	10.45	16	15
2	Mascarpone	32.56	23	
3	Gorgonzola	15.67	9	20

Se supone que la columna “UnidadsOnOrder" es opcional y podría contener valores NULL.

Se tienen la siguiente sentencia SELECT:

```
SELECT Producto,Unidadprecio*(UnidadsInStock+UnidadsOnOrder)
FROM Productos
```

En el ejemplo anterior, si algún de los valores de "UnidadsOnOrder" son NULL el resultado es NULL.

La función ISNULL () de Microsoft es usada para especificar cómo se quiere tratar los valores NULL.

Las funciones NVL (), IFNULL (), y COALESCE () puede ser usado también para obtener el mismo resultado.

En este caso se busca valores NULL a valores cero.

Si la "UnidadsOnOrder" es NULL esto no afectará el cálculo, porque ISNULL () retorna un cero si el valor es NULL:

SQL Server / MS Access

```
SELECT Producto,Unidadprecio*(UnidadsInStock+ISNULL(UnidadsOnOrder,0))
FROM Productos
```

Oracle

Oracle no tiene una función ISNULL (), aunque se puede usar la función NVL () para obtener el mismo resultado:

```
SELECT Producto,Unidadprecio*(UnidadsInStock+NVL(UnidadsOnOrder,0))
FROM Productos
```

MySQL

MySQL posee una función ISNULL() , pero ,. Esta trabaja un poco diferente que la función ISNULL () de Microsoft.

En MySQL se usa la función IFNULL () :

```
SELECT Producto,Unidadprecio*(UnidadsInStock+IFNULL(UnidadsOnOrder,0))
FROM Productos
```

O se puede usar la función COALESCE ():

```
SELECT Producto,Unidadprecio*(UnidadsInStock+COALESCE(UnidadsOnOrder,0))
FROM Productos
```

Microsoft Access Datos Tipos

Datos tipo	Descripción	Storage
Text	Usado para texto o combinación de texto y numero. 255 caracteres como máximo.	
Memo	Memo es usado para textos de gran tamaño. Almacena hasta 65,536 caracteres. **Note:** No se puede ordenar el campo memo , por otro lado se puede realizar una búsqueda	
Byte	Permite números enteros de 0 a 255	1 byte
Integer	Permite números enteros de -32,768 a 32,767	2 bytes
Long	Permite números enteros de -2,147,483,648 a 2,147,483,647	4 bytes
Single	Precisión simple coma flotante. Manejara decimales	4 bytes
Double	Precisión doble coma flotante. Manejara decimales	8 bytes
Currency	Usado para moneda. Almacena hasta 15 dígitos enteros más 4 lugares decimales.	8 bytes
AutoNumber	Campo auto numérico, permite dar a cada registro su propio numero , usualmente comienza en 1	4 bytes
Date/Time	Usado para Fechas y Horas	8 bytes
Yes/No	Campo lógico que puede mostrarse como Yes/No, True/False, o On/Off. En el código se usa la constante True y False (equivalente a -1 and 0). **Note:** los valores Null no son alojados en estos campos	1 bit
Ole Object	Puede almacenar fotos, videos, u otros BLOBs (Binary Large OBjects)	hasta 1GB
Hyperlink	Contiene enlaces a otros archives	

MySQL Datos Tipos

En MySQL hay 3 tipos principales: text, number, and Date/Time types.

Text types:

Datos tipo	Descripción
CHAR(size)	Tiene una longitud fija string (puede contener letras, números y caracteres especiales). El tamaño fijo está especificado en paréntesis. Puede almacenar hasta 255 caracteres
VARCHAR(size)	Tiene una longitud fija string (puede contener letras, números y caracteres especiales). El elemento fijo posee una longitud variable string (puede contener letras, números y caracteres especiales). El tamaño máximo está especificado en paréntesis. Puede almacenar hasta 255 caracteres. Nota: Si se pone un mayor valor que 255 se convertirá en un tipo de texto

TINYTEXT	Almacena un string con un valor máximo de of 255 caracteres
TEXT	Almacena un string con un valor máximo de of 65,535 caracteres
BLOB	BLOBs (Binary Large OBjects). Almacenas hasta 65,535 bytes
MEDIUMTEXT	Almacena un string con un tamaño máximo de 16,777,215 caracteres
MEDIUMBLOB	Para BLOBs (Binary Large OBjects). Almacena hasta 16,777,215 bytes de datos
LONGTEXT	Almacena un string con un tamaño máximo de 4,294,967,295 caracteres
LONGBLOB	Para BLOBs (Binary Large OBjects). Almacena hasta 4,294,967,295 bytes de datos
ENUM(x,y,z,etc.)	Permite ingresar una lista de posibles valores. Se puede listar hasta 65535 valores en un ENUM list. Si un valor insertado no está en la lista, se insertará un valor en blanco.

Tipos numéricos:

Datos tipo	**Descripción**
TINYINT(size)	-128 a 127 normal. 0 a 255 Sin signo*. El máximo número de dígitos puede ser especificado entre paréntesis
SMALLINT(size)	-32768 a 32767 normal. 0 a 65535 Sin signo*. El máximo número de dígitos puede ser especificado entre paréntesis
MEDIUMINT(size)	-8388608 a 8388607 normal. 0 a 16777215 Sin signo*. El máximo número de dígitos puede ser especificado entre paréntesis
INT(size)	-2147483648 a 2147483647 normal. 0 a 4294967295 Sin signo*. El máximo número de dígitos puede ser especificado entre paréntesis
BIGINT(size)	-9223372036854775808 a 9223372036854775807 normal. 0 a 18446744073709551615 Sin signo*. El máximo número de dígitos puede ser especificado entre paréntesis
FLOAT(tamaño)	Un pequeño número con coma flotante. El máximo número de dígitos puede ser especificado con el parámetro size. El máximo número de dígitos a las derecha de la coma es especificado con el parámetro d
DOUBLE(tamaño)	Un número grande con coma flotante. El máximo número de dígitos puede ser especificado con el parámetro size. El máximo número de dígitos a las derecha de la coma es especificado con el parámetro d
DECIMAL(tamaño)	UN DOBLE almacenados en una cadena, permitiendo un punto decimal. El máximo número de dígitos puede ser especificado con el parámetro size. El máximo número de dígitos a las derecha de la coma es especificado con el parámetro d

* Los tipos entero tienen una opción extra llamado Sin signo o UNSIGNED. Normalmente, el entero va desde un valor negativo a un valor positivo. Añadir el atributo UNSIGNED moverá el rango hasta que comienza en cero en lugar de un número negativo

Date types:

Data tipo	Descripción
DATE()	A date. Format: YYYY-MM-DD **Note:** El rango soportado es desde '1000-01-01'a '9999-12-31'
DATETIME()	*A date and time combinación. Format: YYYY-MM-DD HH:MM:SS **Note:** El rango soportado es desde '1000-01-01 00:00:00' a '9999-12-31 23:59:59'
TIMESTAMP()	*A timestamp. TIMESTAMP es un valor almacenado con un numero de segundos ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD HH:MM:SS **Note:** El rango soportado es desde '1970-01-01 00:00:01' UTC a '2038-01-09 03:14:07' UTC
TIME()	A time. Format: HH:MM:SS **Note:** El rango soportado es desde '-838:59:59' a '838:59:59'
YEAR()	Un año en format de 2 o 4 dígitos. **Note:** Valores permitidos en format de 4 dígitos: 1901 a 2155. Valores permitidos en 2 dígitos: 70 a 69, representando los años de 1970 a 2069

Incluso si FECHAHORA y TIMESTAMP regresan el mismo formato, trabajan muy diferente. En una consulta inserción (INSERT) o ACTUALIZACIÓN (UPDATE), el TIMESTAMP ajusta automáticamente a la fecha y hora actual. TIMESTAMP acepta también diversos formatos, como YYYYMMDDHHMMSS, mientras, AAAAMMDD, o YYMMDD..

SQL Server Datos Tipos

Character strings:

Data tipo	Descripción	Storage
char(n)	Longitud fija, cadena de caracteres. Max 8,000 caracteres	n
varchar(n)	Longitud Variable, cadena de caracteres. Max 8,000 caracteres	
varchar(max)	Longitud Variable, cadena de caracteres. Max 1,073,741,824 c caracteres	
text	Longitud Variable, cadena de caracteres. Max 2GB de texto	

Unicode strings:

Data tipo	Descripción	Storage
nchar(n)	Longitud fija, cadena de caracteres Unicode data. Max 4,000 caracteres	
nvarchar(n)	Longitud Variable Unicode data. Max 4,000 caracteres	

nvarchar(max)	Longitud Variable Unicode data. Max 536,870,912 caracteres	
ntext	Longitud Variable Unicode data. Max 2GB of text data	

Binary types:

Data tipo	**Descripción**	**Storage**
bit	Permite valores 0, 1, o NULL	
binary(n)	Longitud Fija binary data. Max 8,000 bytes	
varbinary(n)	Longitud Variable binary data. Max 8,000 bytes	
varbinary(max)	Longitud Variable binary data. Max 2GB	
image	Longitud Variable binary data. Max 2GB	

Number types:

Data tipo	**Descripción**	**Storage**
tinyint	Permite números enteros de 0 a 255	1 byte
smallint	Permite números enteros de -32,768 a 32,767	2 bytes
int	Permite números enteros de -2,147,483,648 a 2,147,483,647	4 bytes
bigint	Permite números enteros de -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807	8 bytes
decimal(p,s)	Fija la precisión y la escala de números Permite números enteros de -10^38 +1 a 10^38 –1. El p parámetro indica el número máximo de dígitos que puede ser almacenado (tanto a la izquierda y la derecha del punto decimal). p debe ser un valor de 1 a 38. Por defecto es de 18. El parámetro indica el número máximo de dígitos almacenados a la derecha del punto decimal. Debe ser un valor de 0 a p. El valor predeterminado es 0	5-17 bytes
numeric(p,s)	Fija la precisión y la escala de números Permite números enteros de -10^38 +1 a 10^38 –1. El p parámetro indica el número máximo de dígitos que puede ser almacenado (tanto a la izquierda y la derecha del punto decimal). p debe ser un valor de 1 a 38. Por defecto es de 18. El parámetro indica el número máximo de dígitos almacenados a la derecha del punto decimal. Debe ser un valor de 0 a p. El valor predeterminado es 0	5-17 bytes

smallmoney	Tipo moneda de -214,748.3648 a 214,748.3647	4 bytes
money	Tipo moneda de -922,337,203,685,477.5808 a 922,337,203,685,477.5807	8 bytes
float(n)	Numero flotante -1.79E + 308 a 1.79E + 308. El parámetro n indica si el campo debería almacenar 4 o 8 bytes. float(24) posee un 4-byte campo y float(53) posee un 8-byte campo. El valor predeterminado de n es 53.	4 o 8 bytes
real	Precisión Flotante número datos de -3.40E + 38 a 3.40E + 38	4 bytes

Date types:

Data tipo	**Descripción**	**Storage**
datetime	A partir de enero 1, 1753 a Diciembre 31, 9999 con una precisión de 3.33 milisegundos	8 bytes
datetime2	A partir de enero 1, 0001 y Diciembre 31, 9999 con una precisión de 100 nanosegundos	6-8 bytes
smalldatetime	A partir de enero 1, 1900 a Junio 6, 2079 con una precisión de 1 minuto	4 bytes
date	Guardar una fecha sólo. A partir de enero 1, 0001 a Diciembre 31, 9999	3 bytes
time	Almacenar un tiempo sólo con una precisión de 100 nanosegundos	3-5 bytes
datetimeoffset	El mismo datetime2 con la adición de una zona horaria compensado	8-10 bytes
timestamp	Almacena un número único que se actualiza cada vez una fila obtiene creado o modificados. La timestamp valor se basa en un reloj interno y no corresponden a tiempo real. Cada tabla puede tener sólo una variable timestamp	

Other data types:

Data tipo	**Descripción**
sql_variant	Tiendas hasta 8,000 bytes de datos de diferentes tipos de datos, excepto texto, ntext, y timestamp
uniqueidentifier	Almacena un identificador único global (GUID)
xml	Almacena datos formateado en XML. Maximum 2GB
cursor	almacena una referencia a un cursor utilizadas para operaciones base de datos
tabla	Almacena un resultado-set para su posterior procesamiento

DCL

Seguridad

DCL comandos son utilizadas para imponer seguridad en un ambiente de una base de datos multiusuario. Hay Dos tipos de DCL comandos son GRANT y REVOKE. Sólo el administrador de base de datos o propietario de la objeto de base de datos puede imponer o remover los privilegios sobre la base de datos.

SQL GRANT QL GRANT

SQL GRANT es un comando usado para dar acceso o privilegios sobre los objetos de una Base de datos a un usuario.

Sintaxis GRANT:

```
GRANT privilegE_Nombre
ON object_name
TO {user_name |PUBLIC |rolE_Nombre}
[WITH GRANT OPTION];
```

- ***privilegE_Nombre*** Es el derecho de acceso o privilegio concedido al usuario. Algunos de los derechos de acceso son ALL, EXECUTE, y SELECT.
- ***object_name*** Es el nombre de una base de datos como objeto TABLA, VIEW, STORED PROC y SEQUENCE.
- ***user_name*** Es el nombre del usuario a quien se concede un derecho de acceso.
- ***PUBLIC*** Se utiliza para conceder derechos de acceso a todos los usuarios.
- ***ROLES*** Son un conjunto de privilegios agrupados.
- ***WITH GRANT OPTION*** - Permite al usuario a conceder derechos de acceso a otros usuarios.

Por Ejemplo:

```
GRANT SELECT ON employee TO user1
```

Este comando concede un permiso SELECT sobre la tabla employee para el user1.

Debe usar la opción WITH GRANT con cuidado, porque si otorga privilegio SELECT al empleado user1 sobre tabla a usando la WITH GRANT , entonces el user1 puede conceder a su vez privilegio SELECT sobre tabla a otro usuario.

DENY

Crea una entrada en el sistema de seguridad que deniega un permiso de una cuenta de seguridad en la base de datos actual e impide que la cuenta de seguridad herede los permisos a través de los miembros de su grupo o función.

Sintaxis

Permisos de la instrucción:

```
DENY{ALL | instrucción[,...n]}

TO cuentaSeguridad[,...n]
```

Permisos del objeto:

```
DENY
        {ALL [PRIVILEGES] | permiso[,...n]}
        {
                [(columna[,…n])] ON {tabla | vista}
                | ON {tabla | vista}[(columna[,…n])]
                | ON {procedimientoAlmacenado | procedimientoExtendido}
        }
TO cuentaSeguridad[,…n]
[CASCADE]
```

Comando REVOKE:

El comando REVOKE remueve los derechos de accesos o privilegios a los objetos de base de datos.

La Sintaxis REVOKE:

```
REVOKE privilegE_Nombre
ON object_name
FROM {user_name |PUBLIC |rolE_Nombre}
```

Revoke Ejemplo

```
REVOKE SELECT ON employee FROM user1
```

Es te comando removerá los privilegios se selección sobre la tabla empleados para el usuario user1.

Cuando son revocados los privilegios select sobre la tabla para un usuario, el usuario no estará habilitado para realizar consultas select otra vez.

Por otro lado, si el usuario ha recibido privilegios SELECT sobre la tabla desde más de un usuario, puede realizar una selección desde la tabla hasta que todos los que dieron los privilegios los revoquen.

Un Administrador puede revocar privilegios que no fueron dados por él.

Privilegios y roles:

Privileges: Privilegios define los derechos de acceso a un usuario en una base de datos. Hay dos tipos de privilegios

1) Privilegios de sistemas - estos permite al usuario a CREATE, ALTER, o DROP objetos de la base de datos .
2) Privilegios de Objetos – estos permiten al usuario a realizar las siguientes operaciones EXECUTE, SELECT, INSERT, UPDATE, o DELETE desde las base de datos .

Privilegios de sistemas para CREATE:

System Privileges	Descripción
CREATE object	Permite al usuario crear el objeto especificado en su propio esquema.
CREATE ANY object	Permite al usuario crear el objeto especificado en cualquier esquema.

Privilegios de sistemas para **DROP.**

System Privileges	Descripción
ALTER object	Permite al usuario modificar un objeto especificado en su propio esquema.
ALTER ANY object	Permite al usuario modificar el objeto especificado en cualquier esquema.

Privilegios de sistemas para DROP**.**

System Privileges	Descripción
DROP object	Permite al usuario borrar un objeto especificado en su propio esquema.
DROP ANY object	Permite al usuario borrar el objeto especificado en cualquier esquema.

Privilegios de Objetos:

Object Privileges	Descripción
INSERT	Permite al usuario insertar filas en una tabla.
SELECT	Permite al usuario seleccionar datos desde una base de datos.
UPDATE	Permite al usuario modificar datos de una tabla.

EXECUTE	Permite al usuario ejecutar un stored procedure o una función.

Roles

Los roles son una colección de privilegios o derechos de acceso. Cuando hay muchos usuarios en una base de datos se torna difícil conceder o revocar privilegios a los usuarios. Por lo tanto, si se definen roles , se puede conceder o revocar privilegios a los usuarios, de manera automática. Se pueden crear roles o utilizar los roles en el sistema predefinidos.

Privilegios dados por medio de roles:

System Role	Privileges Granted a the Role
CONNECT	CREATE TABLA, CREATE VIEW, CREATE SYNONYM, CREATE SEQUENCE, CREATE SESSION etc.
RESOURCE	CREATE PROCEDURE, CREATE SEQUENCE, CREATE TABLA, CREATE TRIGGER etc. El principal uso de los recursos de roles es el de restringir el acceso a bases de datos objetos.
DBA	ALL SYSTEM PRIVILEGES

Creando Roles:

Sintaxis create role:

```
CREATE ROLE rolE_Nombre
[IDENTIFIED BY password];
```

Ejemplo: Para crear roles llamados “Testing” con un password como "pwd",se utiliza:

```
CREATE ROLE testing
[IDENTIFIED BY pwd];
```

Es más fácil que dar privilegios GRANT o REVOKE para el usuario A los usuarios mediante un rol en lugar de asignar un privilegio directamente a cada usuario. Si un rol es identificado por una contraseña, entonces, cuando se concede o revoca privilegios a un rol, definitivamente se tienen que identificar con la contraseña

Generar privilegios GRANT o REVOKE con roles.

Ejemplo: Dar privilegios CREATE TABLA a un usuario mediante los roles llamado testing:

Primero se crea el rol testing

```
CREATE ROLE testing
```

Segundo, conceder el privilegio CREATE TABLA al ROLE testing. Se puede agregar más privilegios a los roles.

```
GRANT CREATE TABLA TO testing;
```

Tercero, conceder un rol a un usuario.

```
GRANT testing TO user1;
```

Para revocar un privilegio de CREATE TABLA desde el rol testing:

```
REVOKE CREATE TABLA FROM testing;
```

Sintaxis para borrar un ROLE:

```
DROP ROLE rol E_Nombre;
```

DROP ROLE testing;

Eliminación de registros con claves foráneas

Una de las principales bondades de las claves foráneas es que permiten eliminar y actualizar registros en cascada.

Con las restricciones de clave foránea podemos eliminar un registro de la tabla cliente y a la vez eliminar un registro de la tabla venta usando sólo una sentencia DELETE. Esto es llamado eliminación en cascada, en donde todos los registros relacionados son eliminados de acuerdo a las relaciones de clave foránea. Una alternativa es no eliminar los registros relacionados, y poner el valor de la clave foránea a NULL (asumiendo que el campo puede tener un valor nulo). En este caso, no se puede poner el valor de la clave foránea id_cliente en la tabla venta, ya que se ha definido como NOT NULL. Las opciones estándar cuando se elimina un registro con clave foránea son:

- ON DELETE RESTRICT
- ON DELETE NO ACTION
- ON DELETE SET DEFAULT
- ON DELETE CASCADE
- ON DELETE SET NULL

ON DELETE RESTRICT es la acción predeterminada, y no permite una eliminación si existe un registro asociado, como se mostró en el ejemplo anterior. ON DELETE NO ACTION hace lo mismo.

ON DELETE SET DEFAULT - se supone que pone el valor de la clave foránea al valor por omisión (DEFAULT) que se definió al momento de crear la tabla.

Si se especifica ON DELETE CASCADE, y una fila en la tabla padre es eliminada, entonces se eliminarán las filas de la tabla hijo cuya clave foránea sea igual al valor de la clave referenciada en la tabla padre.

Si se especifica ON DELETE SET NULL, las filas en la tabla hijo son actualizadas automáticamente poniendo en las columnas de la clave foránea el valor NULL. Si se especifica una acción SET NULL, debe asegurarse de no declarar las columnas en la tabla como NOT NULL.

A continuación se muestra un ejemplo de eliminación en cascada:

```
ALTER TABLE venta ADD FOREIGN KEY(id_cliente)
REFERENCES cliente(id_cliente) ON DELETE CASCADE;
```

Ejemplo se verá cómo están los registros antes de ejecutar la sentencia DELETE:

La tabla "Personas":

P_Id	**Apellido**	**Nombre**	**Dirección**	**Ciudad**
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger
4	Nilsen	Johan	Bakken 2	Stavanger

Tabla ventas

Id_factura	Id cliente	cantidad
1	1	23
2	3	39
3	2	81

Ahora eliminaremos a Pettersen de la base de datos:

```
DELETE FROM cliente WHERE id_cliente=3
```

La tabla "Personas":

P_Id	**Apellido**	**Nombre**	**Dirección**	**Ciudad**
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
4	Nilsen	Johan	Bakken 2	Stavanger

```
SELECT * FROM cliente
```

Tabla ventas

Id_factura	Id cliente	cantidad

1	1	23
3	2	81

Con la eliminación en cascada, se ha eliminado el registro de la tabla venta al que estaba relacionado Pettersen.

Restricciones de integridad referencial en cascada

Las restricciones de integridad referencial en cascada permiten definir las acciones que SGBD lleva a cabo cuando un usuario intenta eliminar o actualizar una clave a la que apuntan las claves externas existentes.

Las cláusulas REFERENCES de las instrucciones CREATE TABLE y ALTER TABLE admiten las cláusulas ON DELETE y ON UPDATE. Las acciones en cascada también se pueden definir mediante el cuadro de diálogo Relaciones de clave externa.

- [ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT }]
- [ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT }]

NO ACTION es el valor predeterminado si no se especifica ON DELETE u ON UPDATE.

ON DELETE NO ACTION

Especifica que si se intenta eliminar una fila con una clave a la que hacen referencia las claves externas de las filas existentes en otras tablas, se produce un error y se revierte la instrucción DELETE.

ON UPDATE NO ACTION

Especifica que si se intenta actualizar un valor de clave en una fila a cuya clave hacen referencia las claves externas de filas existentes en otras tablas, se produce un error y se revierte la instrucción UPDATE.

CASCADE, SET NULL y SET DEFAULT permiten la eliminación o actualización de valores de clave de modo que se pueda realizar un seguimiento de las tablas definidas para tener relaciones de clave externa en la tabla en la que se realizan las modificaciones. Si las acciones referenciales en cascada se han definido también en las tablas de destino, las acciones en cascada especificadas se aplican para las filas eliminadas o actualizadas. No se puede especificar CASCADE para ninguna de las claves externas o principales que tengan una columna **timestamp**.

ON DELETE CASCADE

Especifica que si se intenta eliminar una fila con una clave a la que hacen referencia claves externas de filas existentes en otras tablas, todas las filas que contienen dichas claves externas también se eliminan.

ON UPDATE CASCADE

Específica que si se intenta actualizar un valor de clave de una fila a cuyo valor de clave hacen referencia claves externas de filas existentes en otras tablas, también se actualizan todos los valores que conforman la clave externa al nuevo valor especificado para la clave.

ON DELETE SET NULL

Nota:

CASCADE no se puede especificar si una columna **timestamp** es parte de una clave externa o de la clave con referencia.

Especifica que si se intenta eliminar una fila con una clave a la que hacen referencia las claves externas de las filas existentes de otras tablas, todos los valores que conforman la clave externa de las filas a las que se hace referencia se establecen en NULL. Todas las columnas de clave externa de la tabla de destino deben aceptar valores NULL para que esta restricción se ejecute.

ON UPDATE SET NULL

Especifica que si se intenta actualizar una fila con una clave a la que hacen referencia las claves externas de las filas existentes de otras tablas, todos los valores que conforman la clave externa de las filas a las que se hace referencia se establecen en NULL. Todas las columnas de clave externa de la tabla de destino deben aceptar valores NULL para que esta restricción se ejecute.

ON DELETE SET DEFAULT

Especifica que si se intenta eliminar una fila con una clave a la que hacen referencia las claves externas de las filas existentes de otras tablas, todos los valores que conforman la clave externa de las filas a las que se hace referencia se establecen como predeterminados. Todas las columnas de clave externa de la tabla de destino deben tener una definición predeterminada para que esta restricción se ejecute. Si una columna acepta valores NULL y no se ha establecido ningún valor predeterminado explícito, NULL se convierte en el valor predeterminado implícito de la columna. Todos los valores distintos de NULL que se establecen debido a ON DELETE SET DEFAULT deben tener unos valores correspondientes en la tabla principal para mantener la validez de la restricción de la clave externa.

ON UPDATE SET DEFAULT

Especifica que si se intenta actualizar una fila con una clave a la que hacen referencia las claves externas de las filas existentes de otras tablas, todos los valores que conforman la clave externa de la fila a los que se hace referencia se establecen en sus valores predeterminados. Todas las columnas externas de la tabla de destino deben tener una definición predeterminada para que esta restricción se ejecute. Si una columna se convierte en NULL, y no hay establecido ningún valor predeterminado explícito, NULL deviene el valor predeterminado implícito de la columna. Todos los valores no NULL que se establecen debido a ON UPDATE SET DEFAULT deben tener unos valores correspondientes en la tabla principal para mantener la validez de la restricción de clave externa.

TRANSACCIONES

Dentro del mundo real las bases de datos se deben actualizar constantemente, pero habitualmente estas no son realizadas por una sola operación sino, que es un conjunto de varias de ellas que se ejecutan como ser :

■ Añadir el nuevo orden de la tabla.
■ Actualización de las ventas para el vendedor que tomó el pedido.
■ Actualización de las ventas totales de la oficina del vendedor.
■ Actualización de la cantidad a la parte total de los pedidos de productos.

Para dejar la base de datos en un estado de consistencia, los cambios deben producirse como una unidad. Si un fallo del sistema u otro error crea una situación en la que algunas de las actualizaciones son procesadas y otros no, la integridad de la base de datos se perderá.

Así mismo si otro usuario calcula porcentajes totales o parcialmente a través de la secuencia de cambios, los cálculos serán incorrectos. La secuencia de cambios debe ser un todo o nada en la base de datos propuesta. SQL ofrece precisamente esta capacidad a través de sus características de procesamiento de transacción.

Una transacción es una secuencia de uno o varios comandos SQL que juntos forman una unidad lógica de trabajo. La sentencia SQL que forma la operación es estrechamente interdependiente y relacionada con acciones. Cada declaración en la transacción realiza una parte de una tarea, pero todos ellos son necesarios para completar la tarea. La transacción le indica al SMDB que debe ejecutar la totalidad de las sentencia en la secuencia dispuestas para garantiza la integridad de la base de datos. Las declaraciones en una transacción se ejecutan como una unidad atómica de trabajo en la base de datos. el DBMS debe asegurarse de que cuando falta la recuperación es completa, la base de datos no refleja una operación parcial.
En SQL92, no hay declaración de iniciar la transacción.

COMMIT y ROLLBACK
Las transacciones se apoyan en las sentencias SQL donde agrupan el conjunto de sentencias que integra el procesamiento de transacciones

■ COMMIT.

El comando COMMIT señala el éxito final de una transacción. Se le dice al DBMS que la transacción se ha completado, todos los estados que conforman la transacción se han

ejecutado, y es la base de datos consistente.

■ ROLLBACK.

El ROLLBACK señalas que la transacción NO se realizo con éxito. Le dice al DBMS que el usuario no desea completar la transacción, el DBMS debe hacer restauración de todas las modificaciones introducidas a la base de datos durante la operación para dejarla tal cual la operación comenzó.

El COMMIT y ROLLBACK se ejecutan con las sentencias SQL, como SELECT, INSERT, y UPDATE.

Sentencias para transacciones

Sentencias definen la estructura de una transacción

- begin
- commit
- rollback

Begin y commit

- begin

  Inicia la transacción

- commit

  Finaliza la transacción

  Todas las modificaciones quedan en firme

Sintaxis:

- begin [transaction_name]
- commit [transaction_name]

Ejemplo: @nombre son variables que se utilizan en la transacción

-- @amount es el monto que será transferido.

-- @from_account es la cuenta que se debita.

-- @to_account es la cuenta que se acredita.

begin tran Comienza la transacción

update accounts

set balance = balance - @amount

(Modifica la los datos de una cuenta descontando el monto)

```
        where account = @from_account
    update accounts
        set balance = balance + @amount
        where account = @to_account
    commit tran  (ratifica que la transacción fue realizada)
 Rollback
```

(Modifica la los datos de una cuenta sumando el monto)

- Rollback termina una transacción

Deshace las modificaciones que se hayan hecho

La ejecución continúa con la instrucción siguiente a rollback

Sintaxis:

rollback [transaction_name]

Ejemplo:

```
   -- If @from_account está por debajo de  0, abortar la transferencia.
   begin tran
   update accounts
        set balance = balance - @amount
        where account = @from_account
update accounts
        set balance = balance + @amount
where account = @to_account
   if (select balance from accounts
        where account = @from_account) < 0
            rollback tran
        else
            commit tran
```

(Modifica la los datos de una cuenta descontando el monto)

(Modifica la los datos de una cuenta sumando el monto)

Si la cuenta esta sin fondos la transacción no se realiza mediante la sentencia rollback tran

Si tiene fondo se ratifica la operación commit tran

La norma ANSI / ISO modelo de transacciones de
La norma ANSI / ISO SQL estándar define un modelo de transacciones de SQL y las funciones de los estados COMMIT y ROLLBACK. La mayoría, aunque no todos, los productos comerciales de SQL usar este modelo de operación, que se basa en la operación de apoyo en las primeras versiones de DB2. El estándar SQL especifica que una transacción comienza automáticamente con la primera sentencia SQL ejecutada por un usuario o un programa. La operación continúa a través de sentencias SQL posteriores hasta que se termina en una de cuatro maneras:
■ COMMIT. Un comando COMMIT finaliza la transacción con éxito, por lo que su base de datos de los cambios permanentes. Una nueva transacción comienza inmediatamente después de que el comando COMMIT.
■ ROLLBACK. Un ROLLBACK aborta la transacción, el respaldo de su base de datos de cambios. Una nueva transacción comienza inmediatamente después de la ROLLBACK.
■ El éxito del programa de terminación. Para los programas de SQL, también con éxito el programa de terminación de la transacción termina con éxito, como si un comando COMMIT había sido ejecutado. Debido a que el programa está terminado, no hay ninguna nueva transacción para comenzar.
■ terminación anormal del programa. Para los programas de SQL, también la terminación anormal del programa aborta la operación, como si un ROLLBACK había sido ejecutado. Debido a que el programa está terminado, no hay ninguna nueva transacción para comenzar.

-Las declaraciones de transacciones de SQL
-Las declaraciones de transacciones de SQL controlar debidamente las operaciones en la base de datos de acceso. Este subconjunto de SQL es también llamado el Lenguaje de control de datos de SQL (SQL DCL).

Hay 2 SQL-declaraciones de transacciones:

* Declaración COMMIT - comprometerse (persistente hacer) todos los cambios de la transacción actual
* ROLLBACK - deshacer (anular) todos los cambios de la transacción actual

SQL Statement	**Sintaxis**
AND / OR	SELECT column_name(s) FROM tabla_name WHERE condition AND|OR condition
ALTER TABLA	ALTER TABLA tabla_name ADD column_name datatype o ALTER TABLA tabla_name DROP COLUMN column_name
AS (alias)	SELECT column_name AS column_alias FROM tabla_name o SELECT column_name FROM tabla_name AS tabla_alias
BETWEEN	SELECT column_name(s) FROM tabla_name WHERE column_name BETWEEN value1 AND value2
CREATE DATABASE	CREATE DATABASE databasE_Nombre
CREATE TABLA	CREATE TABLA tabla_name (column_name1 data_type, column_name2 data_type, column_name2 data_type, ...)
CREATE INDEX	CREATE INDEX index_name ON tabla_name (column_name) o CREATE UNIQUE INDEX index_name ON tabla_name (column_name)
CREATE VIEW	CREATE VIEW view_name AS SELECT column_name(s) FROM tabla_name WHERE condition
DELETE	DELETE FROM tabla_name

	WHERE some_column=some_value o DELETE FROM tabla_name (**Note:** Borra la tabla entera!!) DELETE * FROM tabla_name (**Note:** Borra la tabla entera!!)
DROP DATABASE	DROP DATABASE databasE_Nombre
DROP INDEX	DROP INDEX tabla_name.index_name (SQL Server) DROP INDEX index_name ON tabla_name (MS Access) DROP INDEX index_name (DB2/Oracle) ALTER TABLA tabla_name DROP INDEX index_name (MySQL)
DROP TABLA	DROP TABLA tabla_name
GROUP BY	SELECT column_name, aggregate_function(column_name) FROM tabla_name WHERE column_name operator value GROUP BY column_name
HAVING	SELECT column_name, aggregate_function(column_name) FROM tabla_name WHERE column_name operator value GROUP BY column_name HAVING aggregate_function(column_name) operator value
IN	SELECT column_name(s) FROM tabla_name WHERE column_name IN (value1,value2,..)
INSERT INTO	INSERT INTO tabla_name VALUES (value1, value2, value3,....) *o* INSERT INTO tabla_name (column1, column2, column3,...) VALUES (value1, value2, value3,....)
INNER JOIN	SELECT column_name(s) FROM tabla_name1 INNER JOIN tabla_name2 ON tabla_name1.column_name=tabla_name2.column_name
LEFT JOIN	SELECT column_name(s) FROM tabla_name1 LEFT JOIN tabla_name2 ON tabla_name1.column_name=tabla_name2.column_name
RIGHT JOIN	SELECT column_name(s) FROM tabla_name1

	RIGHT JOIN tabla_name2 ON tabla_name1.column_name=tabla_name2.column_name
FULL JOIN	SELECT column_name(s) FROM tabla_name1 FULL JOIN tabla_name2 ON tabla_name1.column_name=tabla_name2.column_name
LIKE	SELECT column_name(s) FROM tabla_name WHERE column_name LIKE pattern
ORDER BY	SELECT column_name(s) FROM tabla_name ORDER BY column_name [ASC|DESC]
SELECT	SELECT column_name(s) FROM tabla_name
SELECT *	SELECT * FROM tabla_name
SELECT DISTINCT	SELECT DISTINCT column_name(s) FROM tabla_name
SELECT INTO	SELECT * INTO new_tabla_name [IN externaldatabase] FROM old_tabla_name *o* SELECT column_name(s) INTO new_tabla_name [IN externaldatabase] FROM old_tabla_name
SELECT TOP	SELECT TOP number|percent column_name(s) FROM tabla_name
TRUNCATE TABLA	TRUNCATE TABLA tabla_name
UNION	SELECT column_name(s) FROM tabla_name1 UNION SELECT column_name(s) FROM tabla_name2
UNION ALL	SELECT column_name(s) FROM tabla_name1 UNION ALL SELECT column_name(s) FROM tabla_name2
UPDATE	UPDATE tabla_name SET column1=value, column2=value,... WHERE some_column=some_value
WHERE	SELECT column_name(s) FROM tabla_name WHERE column_name operator value

Bibliografía

www.sql.org

SQL, the complete reference - James R. Groff, Paul N. Weinberg - 2002 - 1050 páginas

SQL in a nutshell - Kevin E. Kline, Daniel Kline, Brand Hunt - 2004 - 691 páginas

SQL: A BEGINNER'S GUIDE 3/E - Andy Oppel, Robert Sheldon - 2008 - 534 páginas