Las claves deben ser unicas, y minimas. Pueden ser PK o FK

Los esquemas son agrupamiento de objetos en la base de datos.

Los índices sirven para mejorar la perfomance.

SELECT campos, ctes

FROM tablas, vistas

WHERE condición sobre fila

GROUP BY condición de agrupamiento

HAVING condiciónsobregrupos

ORDER BY (default ASC)

Todas las columnas del select, deben estar en:

* Clausula GROUP BY

o

* Funcion de grupo

Los alias de columna solo son visibles en el orderby

Los stringse concatenan con +

Los SUBSELECT siempre deben devolver una sola columna y una sola fila ,salvo en el EXISTS(). No se puede usar en el FROM, y puede devolver NULL. NO SE PUEDE ANIDAR DENTRO DE UNA FUNCION DE GRUPO!!!

Operadores:

IS NULL /IS NOT NULL

BETWEEN N AND M

LIKE ‘%XXX%’

IN (C1,C2,C3,C4,C5)

EXISTS( SELECT …… )

DISTINCT: trae solo los distintos, comparando TODA la fila de resultados (solo columnas especificadas en el SELECT)

TOP

UNION/UNION ALL: pega 2 conjuntos uno arriba del otro sin repetidos. Puede haber solo un ORDER BY. Con UNION ALL trae repetidos.

Funciones de grupo

COUNT ( )

MAX( ) /MIN( )

ISNULL(campo,valorDefault)

SUM() /AVG ()

CONVERT(TipodeDatoDestino, columna, codigo) ->Codigo 112=yyyymmdd

DATEPART(Intervalo, Fecha) ->Intervalo YEAR,MONTH,DAY……

NO SE PUEDEN ANIDAR FUNCIONES DE GRUPO

JOINS

INNER Trae solo los que matchean según el JOIN

OUTER JOINS

LEFT JOIN Trae todos los de la tabla izquierda (FROM) y los que no matcheen el JOIN les pone NULL

RIGHT JOIN Trae todos los de la table derecha (JOIN) y los que no matcheen el join les pone null

FULL JOIN Trae todos de las 2 tablas y los que no matcheen les pone null

PL-SQL

Vistas

Selectprecompilado, guardado en el motor. Es más performante y seguro ya que se puede limitar al momento de creación, las columnas disponibles

CREATE VIEW Nombre [ nombreColVista1, nombreColVista2 ….]

AS

SELECT ……….

Funciones

Devuelve siempre un valor. No puede modificar tablas pero si puede usarse en un select

CREATE FUNCTION Nombre [ (@PARAM1 tipoDeDato,@PARAM2tipoDeDato, ….)]

RETURNS TipoDeDato

AS

BEGIN

…..

….

RETURN @retorno

END

StoredProcedures

No esta obligado a devolver valor, puede modificar tablas pero no usarse en un select.

CREATE PROCEDURE Nombre[ (@PARAM1 tipoDeDato,@PARAM2 tipoDeDato [OUTPUT], ….)]

AS

BEGIN

END

Variables

DECLARE @VARtipoDeDato

SET @VAR =VALOR

SELECT @VAR=COLUMNA

IF Condicion

BEGIN

END

ELSE

BEGIN

END

CASE WHEN Condicion THEN Valor

WHEN Condicion2 THEN Valor2

ELSE Valor3

END

CASE Condicion

When ValorCond Then Valor

When ValorCond2 Then Valor2

WHILE Condicion

BEGIN

END

RETURN->Sale de la rutina en ejecucion.

@@Error->Si <>0->ERROR

@@RowCount ->Cantidad de filas afectadas

@@TranCount->Cantidad de transacciones abiertas

Cursores

Permite recorrer uno a uno el resultado del SELECT.

DECLARE nombre CURSOR [ LOCAL | GLOBAL ]

[ FORWARD\_ONLY | SCROLL ]

[ STATIC | DYNAMIC | FAST\_FORWARD ]

[ READ\_ONLY | SCROLL\_LOCKS ]

FOR SELECT ……….

[ FOR UPDATE [ OF nombreColumna [ ,...n ] ] ]

OPEN nombre

FETCH …..

….

….

CLOSE nombre

DEALLOCATE nombre

LOCAL: El cursor es visible por el bloque de T-SQL actual (funcion, procedimiento etc).

GLOBAL: El cursor es visible por cualquier objeto de la base de datos, mientras dure la conexion actual. El cursor se desaloca automáticamente al cerrarse la conexión.

FORWARD\_ONLY: El FETCH solo puede ser NEXT, o sea no se puede ir y volver, siempre se avanza.

(Por default es FORWARD\_ONLY)

SCROLL: El FETCH puede ser en cualquier sentido.

STATIC: El cursor es una “foto” del resultado del select, por lo que si durante el recorrido se hacen modificaciones en la table estas no se veran reflejados en el cursor.

DYNAMIC: El cursor sufre los cambios hechos en la tabla. No se puede hacer FETCH ABSOLUTE

FAST\_FORWARD: Establece al cursor como FORWARD\_ONLY, READ\_ONLY y con mejoras de performance.

READ\_ONLY:prohibeupdates a traves del cursor, por ende haciendolo de solo lectura.

SCROLL\_LOCKS:Lockea las filas a medida que las recorre el cursor, asi cuando el cursor esta parado en una fila, se pueda hacer una actualizacion sobre la misma.

FOR UPDATE [of nombreColumna1, ….] : habilita columnas updeteables a traves del cursor. Si se especifica el FOR UPDATE pero no se especifican columnas , todas son updateables.

FETCH

          [ [ NEXT | PRIOR | FIRST | LAST

                    | ABSOLUTE { n | @nvar }

                    | RELATIVE { n | @nvar }

               ]

               FROM

          ]

{ { [ GLOBAL ] nombreCursorGlobal } | @nombreCursorLocal}

[ INTO @Variable1 [ ,...n ] ]

NEXT: Obtiene la siguiente fila y posiciona el cursor ahi

PRIOR: obtiene la fila anterior y posiciona el cursor ahi

FIRST: obtiene la primer fila y posiciona el cursor ahi

LAST: obtiene la ultima fila y posiciona el cursor ahi

ABSOLUTE@N :

Si N>0 entonces obtiene la fila numero N comenzando desde el principio del conjunto.

Si N<0 entonces obtiene la fila numero N comenzando desde el final del conjunto hacia atrás.

Si N=0 entonces no devuelve nada.

RELATIVE @N

Si N>0 entonces obtiene la fila numero N comenzando desde la fila apuntada actualmente por el cursor.

Si N<0 entonces obtiene la fila numero N comenzando desde lafila apuntada actualmente por el cursor pero haciaatrás.

Si N=0 entonces no devuelve nada.

INTO @Variable1,@Variable2,…..

Es la lista de variables donde se guardan las columnas que trae el FETCH. Debe haber la misma cantidad y cada una debe estar ordenada en sus tipos de datos respecto al cursor.

@@FETCH\_STATUS , devuelve 0 si el fetch se hizo bien, devuelve <>0 si hubo error o EOF.

Triggers

Se ejecuta automaticamente ante un evento. Forma parte de la transacción del evento (aunque sea implícita)

No posee parámetros y no se puede controlar su ejecución. En el contexto de un trigger se dispone de 2 tablas, INSERTED (con los datos insertados o updateados), DELETED (con los datos eliminados o updateados), las cuales son temporales y de solo lectura. Para saber si hubo un update un registro debe estar en ambas tablas.

CREATE TRIGGER nombre

ON { tabla | vista }

{ FOR | AFTER | INSTEAD OF }

{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }

AS

BEGIN

CODIGO T-SQL

END

FOR/AFTER: Se ejecuta cuando todas las operaciones que dispararon el triggerterminaron de ejecutarse.

INSTEAD OF: Se ejecuta en lugar de las operaciones que dispararon el trigger.

INSERT, UPDATE,DELETE: Se especifica para que evento queremos que se dispare el trigger

Transacciones

Son un conjunto de operaciones que se ejecutan atómicamente con el fin de mantener la base de datos consistente.

Cumplen con ser ACID:

* Aislamiento: asegura que una transacción puede trabajar con un conjunto de datos sin que otra transacción puede modificárselo, permitiendo así la concurrencia.
* Atomicidad: Se ejecuta o todo o nada
* Consistencia: No se rompe la integridad referencial de la base de datos.
* Durabilidad: una vez finalizada la transacción los datos van a quedar guardados.

BEGIN TRANSACTION [nombre]: Incrementa en uno @@TRANCOUNT

COMMIT TRANSACTION [nombre]: Decremente en uno @@TRANCOUNT, si llego a 0 ->guarda los datos.

ROLLBACK TRANSACTION [nombre] [nombresavepoint] :

* Si no se especifica nombresavepoint, setea @@TRANCOUNT en cero y deshace hasta el BEGIN TRANSACTION mas externo.
* Si se especifica nombresavepoint, no modifica @@TRANCOUNT , solo deshace hasta el SAVE TRANSACTION que corresponda.

SAVE TRANSACTION [nombresavepoint] : Marca un savepoint en la transaccion actual

En toda operación se abren transacciones, si no se hizo un BEGIN TRANSACTION, se abre una transacción implícita que se cierra al terminar de ejecutar la operación.

Arboles

Los arboles se recorren por niveles, de arriba hacia abajo, de izquierda a derecha.

Hay 3 tipos de recorridos:

* Pre-order: Llega e imprime el valor del nodo.
* In-Order: Imprime al momento de cambiar de rama
* Post-Order: Imprime cuando no va a volver a pasar mas.

El árbol puede tener 3 caracteristicas:

* Balanceado: Todos los subárboles tienen el mismo peso, o la diferencia entre ellos es indivisible (1)
* Completo: Todos los elementos cumplen el grado del árbol o son hojas.
* Perfectamente Balanceado: Todos los subárboles están balanceados y todas las hojas están al mismo nivel.

Grafos

En los grafos existen 2 conceptos importantes de recorrido

* Paso: Es la conexión que hay entre 2 elementos del grafo, la cual tiene un sentido y debe respetarse.
* Camino: Es la conexión que hay entre 2 elementos del grafo, sin tener en cuenta el sentido de la misma.

La longitud de un paso es la cantidad de pasos (aristas) que deben hacerse para llegar al destino. Los cíclicos tienen longitud 0.

Algoritmo de búsqueda de paso en grafos

Hay 2 algoritmos:

* DeepthFirst: Se toma la primera rama y se le pregunta. Si no es el elemento, se le va “hasta el fondo” yendo por esa rama hasta el final.
* Wide First: Se pregunta primero a todos los nodos del nivel siguiente, si no lo tiene ninguno baja por cada una de las ramas barriendo de la misma forma. Hace los caminos mas cortos.

Algoritmos de Ordenamiento

* Burbuja: Compara uno con el siguiente, si es mayor el siguiente los intercambia, asi barriendo todos. Es O (n2)
* Minimo: Compara siempre el primero contra todos. Es O (n2)

Los algoritmos que superan la linealidad no son ejecutables, ya que para un usuario, si tarda 1 minuto en buscar 10 registros esperara que tarde 2 minutos para buscar 20 registros

* QuickSort: Toma uno como pivote y compara contra todos, armando 2 subconjuntos (mayores y menores). Es O(n.ln(n))
* B-Sort Igual que el QS pero toma como pivote el del medio, asumiendo que esta ordenado
* Heap-Sort:Se arma un árbol binario y completo, donde se cumple la regla de que los padres son SIEMPRE mayores a los hijos.