HERRAMIENTAS DE SISTEMAS DE GESTION DE BASES DE DATOS

1. INTRODUCCION.

El objetivo del presente apunte es mostrar algunas de las herramientas que los sistemas de gestión poseen en la actualidad para la resolución de problemas en las implementaciones de las bases de datos, como por ejemplo:

- Mantenimiento de la integridad de entidad y la integridad referencial.
- Restricciones al modelo de datos, inherentes a las reglas del negocio, que deberían sino ser implementadas en las aplicaciones que usan la base.
- Mantenimiento de las redundancias impuestas por cuestiones de performance.
- Restricción de acceso a usuarios a datos para los cuales no posee autorización.

Para los ejemplos se utilizó una base de datos Sybase con las siguientes tablas:

cliente (nro_cliente , nombre , descuento , telefono , ...)
encab_factura (nro_factura , nro_cliente , fecha , ...)
detalle_factura (nro_factura , item , importe , ...)

teniendo en cuenta que los descuentos de los clientes no pueden ser superiores al 30 %, y que si no se posee el teléfono del mismo se debe incluir la leyenda “No posee teléfono”.

Aclaraciones
La sintaxis de algunos comandos ha sido simplificada a efectos de facilitar la comprensión de los mismos.
Este trabajo se encuentra en elaboración, por lo que agradecemos las observaciones que puedan aportar los alumnos.

2. USER DATA TYPES.

DEFINICION

Consiste en un nombre provisto por el usuario, un tipo de datos del sistema y una propiedad opcional sobre el tipo de dato.

USO

Se utilizan en lugar de los tipos de datos del sistema porque centralizan la definición de los tipos de datos y porque permiten que se les asocie defaults y reglas directamente a ellos, sin tener que hacerlo para todas las columnas en los cuales se utiliza.

SINTAXIS

```
sp_addtype  type_name,  datatype  [  ,  {  null  /  not null  }  ]
```

EJEMPLO

```
sp_addtype  “tipo_telefono”,  char ( 30 ),  not null
```

SINTAXIS DE BORRADO

```
sp_droptype  type_name
```

3. RULES.

DEFINICION

Especifica una restricción al dominio de valores aceptables para una columna en particular o para cualquier columna de un tipo definido por el usuario. Puede ser una lista o conjunto de valores, un rango de valores o una máscara de edición.

USO

Las rules son creadas con el comando create rule con un nombre y luego son asociadas a la columna correspondiente con el comando sp_bindrule. Esto permite el no tener que crear una rule para cada columna sino crear aquellas que describan el comportamiento del repertorio de atributos y luego ir asociando la regla correspondiente a todos los campos que la deban cumplir.

SINTAXIS

create rule rule_name as condition_expression

sp_bindrule rule_name, { column_name / user_data_type }

EJEMPLOS

create rule descuento_cliente as descuento between 0 and 30

sp_bindrule “descuento_cliente”, “cliente.descuento”

Estas instrucciones crean una regla para que los clientes no tengan un descuento mayor al 30 %.

SINTAXIS DE DESASOCIACION Y BORRADO

sp_unbindrule { column_name / user_data_type }

drop rule rule_name

4. DEFAULTS.

DEFINICION

Es una cláusula que se le agrega a las columnas o a los tipos definidos por el usuario, indicando el valor que tomará ésta si al insertar una nueva fila no se le especifica un valor.

USO

Al ser insertada una nueva fila en una tabla, el usuario tiene la posibilidad de especificar qué columnas desea actualizar con los nuevos valores, por lo que el resto pasarán a tener valores null. Si esto no responde al modelo y el valor puede ser completado con uno por defecto, se le pude informar a la base de datos que lo haga automáticamente por medio de los defaults.

SINTAXIS

Se puede efectuar de dos maneras:

- En la creación de la tabla:
  create table table_name (column_name datatype default constant_expression, …….)

- Creando el default y luego asociándolo a las columnas deseadas:
  create default default_name as constant_expression
  sp_bindefault default_name, { column_name / user_data_type }

EJEMPLOS

```
create table  cliente (   nro_cliente    integer          not null,
                          nombre         char ( 30 )      not null,
                          descuento      integer          default  0,
                          telefono       char ( 30 )      default  “No posee teléfono”,
                          .....
                      )
```

```
create default  descuento_dft  as  0

create default  telefono_dft  as  “No posee teléfono”

sp_bindefault  “descuento_dft”,  “cliente.descuento”

sp_bindefault  “telefono_dft”,  “cliente.telefono”
```

Los dos ejemplos dados anteriormente cumplen la misma función. La diferencia radica en que en el segundo caso el default queda definido bajo un nombre para poder ser reusado en otras columnas de la base de datos.

SINTAXIS DE DESASOCIACION Y BORRADO

```
sp_unbindefault  column_name

drop default  default_name
```

5. STORED PROCEDURES.

DEFINICION

Conjunto de instrucciones Transact-SQL y de control que se almacenan en la base de datos y que pueden ser ejecutadas al ser invocadas por su nombre.

USO

Normalmente es usado para almacenar instrucciones que son realizadas con frecuencia, o para almacenar procedimientos de actualización y consulta de la base de datos. Habilitando a los usuarios el uso de stored procedures, en lugar del acceso directo a la base de datos se centralizan los procesos de actualización, minimizándose los errores y pudiéndose efectuar un control más estricto del software que se ejecuta sobre la base de datos.

BENEFICIOS

- Se pueden almacenar en una versión compilada, reduciéndose el tamaño.
- Se ejecutan más rápidamente que el mismo query interactivo.
- Reducen el tráfico de la red al ser invocados sólo por su nombre y no pasar el query completo.
- Refuerzan la consistencia interna de la base de datos.
- Ayudan a mantener la seguridad.
- Promueven el desarrollo de aplicaciones modulares.
- Reduces los errores de operación.
- Automatizan transacciones complejas o críticas.

SINTAXIS DE CREACION

```
create procedure  procedure_name
  [ (  @parameter_name  datatype  [ ( length )  /  ( precision [ , scale ] ) ]  [ = default ] [ output ]
  [ ,  @parameter_name  datatype  [ ( length )  /  ( precision [ , scale ] ) ]  [ = default ] [ output ] ]…… ) ]
  as  Transact-SQL_statements
```

EJEMPLOS

El siguiente stored procedure verifica la existencia del cliente al cual se le quiere ingresar una factura, retornando 0 si se pudo efectuar correctamente o 1 si no existía el cliente.

```
create procedure  inserto_encab_factura ( @nro_fac  integer  , @nro_cli  integer  ,  @fecha  date )
as
if  0  =  ( select  count ( * )  from  cliente  where  nro_cliente  = @nro_cli )
   begin
      print  “No existe el cliente correspondiente a la factura”
      return  1
   end
insert  encab_factura  ( nro_factura  ,  nro_cliente  ,  fecha )
values ( @nro_fac  ,  @nro_cli  ,  @fecha )
return  0
```

El siguiente stored procedure efectúa el borrado de un cliente, pero efectuando también el borrado de las facturas que éste posee en su cuenta corriente.

```
create procedure  borro_cliente ( @nro_cli )
as
delete  from  cliente  where  nro_cliente  =  @nro_cli
delete  from  detalle_factura
        where  exists  ( select  *  from encab_factura
                                  where  nro_cliente  =  @nro_cli  and
                                         encab_factura.nro_factura  =  detalle_factura.nro_factura )
delete  from  encab_factura  where  nro_cliente  =  @nro_cli
return  0
```

SINTAXIS DE BORRADO

```
drop procedure  procedure_name
```

6. TRIGGERS.

DEFINICION

Tipo de stored procedure que se ejecuta automáticamente cuando un usuario trata de efectuar un cierto tipo de modificación sobre una tabla especifica.

USO

Normalmente pueden ser utilizados para:

- Reforzar la integridad referencial.
- Proporcionar métodos de borrado y actualización en cascada.
- Mantener actualizados datos redundantes en caso de existir.
- Implementar un sistema de log de aquellas acciones que se desean registrar (sobre las actualizaciones únicamente, ya que los triggers no se activan por la instrucción select).

SINTAXIS DE CREACION

Existen dos sintaxis válidas para los triggers debido a que, de acuerdo a la forma en que se activan, se pueden clasificar en:

- los que se activan al tratar algún tipo de actualización sobre la tabla en general (inserción , borrado o modificación) sin hacer distinción de campos.
- los que se activan al efectuar la modificación sobre un/os campo/s en particular (inserción o modificación). Esto permite especificar más de una acción dependiendo de los campos que son actualizados en la tabla especificada.

En Sybase:

```
create trigger  trigger_name  on  table_name  for  {  insert  /  update  /  delete  }
    as  Transact-SQL_statements
```

ó

```
create trigger  trigger_name  on  table_name  for  {  insert  /  update  }
    as
if  update (column_name)  [ [ {and/or}  update (column_name) ] …]  Transact-SQL_statements
[ [ if  update (column_name)  [ [ {and/or}  update (column_name) ] …]  Transact-SQL_statements ] …]
```

Para las instrucciones usadas dentro de los triggers existen dos palabras reservadas : inserted y deleted, que denotan tablas de referencia estructuralmente iguales a las tablas para las cuáles el trigger se encuentra definido y que contienen los nuevos valores a ser insertados (para insert o update) o eliminados (para delete o update). Estas tablas pueden ser examinadas pero no actualizadas por el trigger.

En Oracle:

```
Create Trigger  trigger_name  { after / before }  { delete or insert or update }  on  table_name
[ for each row ]  [ when condition ]
declare  declaración_de_variables
begin
      SQL_statements
end
```

Un trigger puede ejecutarse después de aplicar la actualización que lo dispara o antes (opción { after / before }).
Asimismo se puede ejecutar para cada una de las filas que se está actualizando (opción [for each row]) , o luego de efectivizarse toda la instrucción que lo disparo.
También puede definirse un filtro para la ejecución (opción when).

EJEMPLOS

Para Sybase:

```
create trigger  trigger1  on  encab_factura  for  delete  as
delete from detalle_factura  where  deleted.nro_factura  =  detalle_factura.nro_factura
```

Este trigger elimina las ocurrencias de registros en la tabla de detalle de facturas cuando se intenta borrar su registro correspondiente en el de encabezamiento de facturas.

```
create trigger  trigger2  on  encab_factura  for  insert  as
if  0  =  ( select  count ( * )  from  cliente  where  inserted.nro_cliente  =  cliente.nro_cliente )
begin
  print  “No existe el cliente, por favor ingréselo antes de pasar las facturas”
  rollback trigger
end
```

Este trigger verifica la integridad referencial con la tabla de clientes: no pueden existir facturas de clientes que no existen.

```
create trigger trigger3 on cliente for update as
if update ( descuento )
begin
  if inserted.descuento > 30
     begin
        print "No se puede dar más de un 30 % de descuento a los clientes"
        rollback trigger
     end
end
```

Este trigger limita los valores del descuento asignado a un cliente a un 30 %, avisando al operador si esta tratando de sobrepasarlo y eliminado la transacción.

SINTAXIS DE BORRADO

```
drop trigger trigger_name
```

OTRAS INSTRUCCIONES ASOCIADAS

rollback trigger	Deshace el trabajo realizado hasta el momento en el trigger.
begin transaction	Informa el comienzo de una transaccion.
rollback transaction	Deshace el trabajo realizado hasta el momento en la transacción.
commit transaction	Registra las actualizaciones hechas en la base de datos por la transacción.

Nota: Se debe tener cuidado cuando se arma la estructura de triggers respecto de los anidamientos, ya que si estos efectúan actualizaciones sobre otras tablas, esto a su vez desencadenaría otros triggers y el efecto cascada puede irse de control. Muchas implementaciones permiten limitar la cantidad de llamadas de triggers a triggers para limitar este efecto.

7. CONSTRAINTS.

DEFINICION

Es una restricción a los datos que se pueden ingresar en la tabla a la cuál se encuentra asociada. Los tipos de constraints que se pueden definir son:

- de integridad referencial o claves foráneas.
- de clave primaria.
- de unicidad de la clave candidata.
- de chequeo de valores ingresados.

USO

Se utilizan para implementar en la base de datos algunas de las restricciones que el modelo de datos tiene como así también algunas de las características del negocio al cual reflejan.

SINTAXIS

La definición de los constraints se efectúa durante la creación de la tabla a la cuál se encuentra relacionada:

```
create table  table_name
( column_name  { data_type  /  user_data_type }
  [ default  { constant_expression  /  null } ]
  [ { null  /  not null } ]
  [ [ [ constraint  constraint_name ]
     { { unique  /  primary key }
        /  references  reference_table  [ ( reference_column ) ]
       /   check  ( check_condition )
      } ] ……
    ] , ……
  [ [ [ constraint  constraint_name ]
     { { unique  /  primary key }
        /  foreign key  ( column_name [ [ ,  column_name ] ……] )
           references  reference_table  [ ( reference_column [ [ ,  reference_column ] ….] )  ]
       /   check  ( check_condition )
      } ] ……
    ]
)
```

Si no se le especifica constraint la base de datos le asigna uno con el nombre de la tabla y columna.
Cabe aclarar que cuando se indica una restricción sobre un solo atributo de la tabla, la definición acompaña a la definición del campo. Pero cuando la restricción es sobre varios campos, ésta se especifica al final de las definiciones de los campos.
La cláusula unique indica columnas que deben tomar valores distintos para cada fila, y pueden tener valores nulos. Este constraint crea un índice unique que solamente puede ser eliminado si el constraint es eliminado usando alter table.
La cláusula primary key indica columnas que deben tomar valores distintos para cada fila, y no pueden tener valores nulos.

EJEMPLOS

Las siguientes instrucciones crean las tablas en la base de datos indicando las claves primarias, claves foráneas, restricciones a los valores a ingresar en los campos y claves candidatas.

```
create table  cliente
(
   nro_cliente                  integer        not null
    constraint pk_cliente                      primary key
    constraint ch_pk_cliente                   check ( nro_cliente > 0 ) ,
   nombre                       char ( 30 )    not null
    constraint u_clientenom                    unique ,
   descuento                    integer        default  0
    constraint ch_d_cliente                    check ( descuento between 0 and 30 ) ,
   teléfono                     char ( 30 )    default “No posee teléfono” ,
   .........
)

create table  encab_factura
(
   nro_factura                  integer        not null
    constraint pk_encabfac                     primary key
    constraint ch_pk_encab                     check ( nro_factura > 0 ) ,
   nro_cliente                  integer        not null
    constraint fk_encabfaccli                  references cliente ( nro_cliente ) ,
   fecha                        date           not null ,
   .........
)
```

```
create table  detalle_factura
(
   nro_factura                integer              not null
     constraint fk_detfact                        references encab_factura ,
   item                       integer              not null ,
   importe                    float ,
   .........
     constraint pk_detfactura ( nro_factura , item )
)
```

SINTAXIS DE BORRADO

```
alter table  table_name  drop constraint  constraint_name
```

OTRAS INSTRUCCIONES ASOCIADAS

```
alter table  ,  drop table
create index  ,  drop index
create rule  ,  drop rule
sp_addtype  ,  sp_droptype
sp_primarykey  ,  sp_commonkey , sp_foreignkey
```

8. VISTAS.

DEFINICION

Es una instrucción select almacenada que en el contexto de su utilización, se comporta como una tabla pero no mantiene información. Puede seleccionar desde más de una tabla y puede ser usada para incluir información parcial de la(s) tabla(s). Es una forma alternativa para acceder a la información de una o más tablas.

USO Y BENEFICIOS

- Son usadas como mecanismo de seguridad mediante el permiso de acceso a las vistas y no a las tablas propiamente dichas.
- Simplifican el manejo de los datos.
- Proveen independencia lógica de los datos.
- Brindan al usuario una percepción simplificada y personalizada de la base de datos.

SINTAXIS DE CREACION

```
create view  view_name  [ ( column_name  [ , column_name ] …. ) ]
  as  select_statement
```

EJEMPLOS

```
create view  facturas  ( cliente  ,  factura  ,  importe )
  as  select nro_cliente , encab_factura.nro_factura , sum ( detalle_factura.importe )
            from  encab_factura  ,  detalle_factura
            where  encab_factura.nro_factura  =  detalle_factura.nro_factura
            group by  nro_cliente  ,  encab_factura.nro_factura
```

Esta instrucción deja almacenada en la base de datos una vista llamada facturas cuya consulta brindará el número de cliente , el número de la factura y el importe total de la factura en los campos cliente , factura e importe.

```
create view  clientes_bsas
  as  select nro_cliente ,  nombre
            from  cliente
            where  provincia  =  “Buenos Aires”
```

Esta vista exhibirá únicamente los clientes que tienen domicilio en Buenos Aires y con las columnas nro_cliente y nombre.

SINTAXIS DE BORRADO

drop view view_name

9. TRANSACCIONES.

Las transacciones definidas por el usuario proveen un mecanismo para agrupar instrucciones Transact-SQL de tal manera que estas son tratadas como una unidad. De este modo todas las instrucciones incluidas en la transacción son ejecutadas, o en caso contrario ninguna es ejecutada.

Las instrucciones utilizadas para definir las transacciones son:

begin transaction [transaction_name]	Especifica el comienzo de una transacción definida por el usuario.
commit transaction [transaction_name]	Hace efectiva la transacción finalizando la misma.
rollback transaction [transaction_name]	Deshace la transacción definida sin que ninguna de las modificaciones efectuadas durante la misma tenga efecto.

10. COMANDOS TRANSACT-SQL.

IF ... ELSE

```
if  logical_expression  statement
        [ else  statement ]
```

donde statement puede ser una instrucción Transact-SQL o un conjunto de ellas encerradas por la estructura begin ..... end

RETURN

```
return  [ integer_expression ]
```

Produce la finalización incondicional del stored procedure donde se encuentra la instrucción. Opcionalmente permite el retorno de un valor entero para indicar que la finalización del stored procedure se realizó en forma satisfactoria (usualmente el valor 0), o si hubo algún problema durante el mismo.

WHILE

```
while  logical_expression  statement
```

BEGIN … END

```
begin
    statement
    ……
end
```

Encierra una serie de comandos Transact-SQL para que instrucciones de control tales como if ... else tengan efecto sobre la totalidad de ellas.

BREAK

Causa la salida de una estructura while.

CONTINUE

Causa que la iteración en una instrucción while comience nuevamente ignorándose las instrucciones posteriores, en caso de existir.