UTN
Facultad Regional Córdoba

TUP

Tecnicatura Universitaria en Programación

LABORATORIO DE COMPUTACIÓN II

Unidad Temática N°3:
Objetos de Bases de Datos

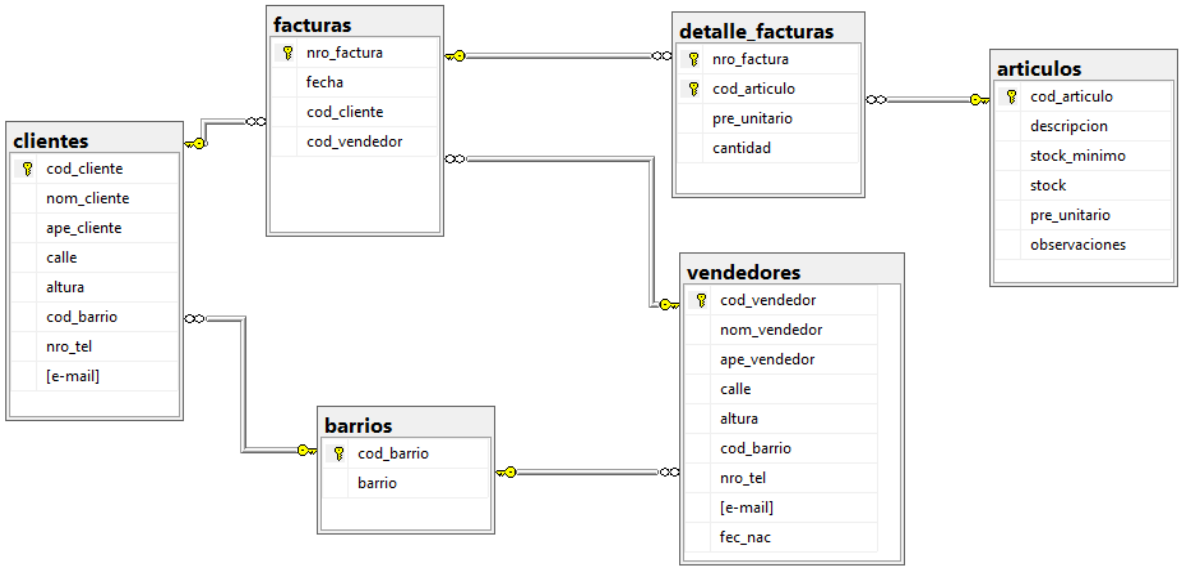
Guía de estudio
1° Año – 2° Cuatrimestre

Índice

Problema 3.1: Vistas

Desde el área de programación del sistema de información de una librería mayorista se solicita la creación de una serie de vistas desde la base de datos de la facturación de este negocio.

El diagrama de la base de datos se muestra en la figura a continuación.

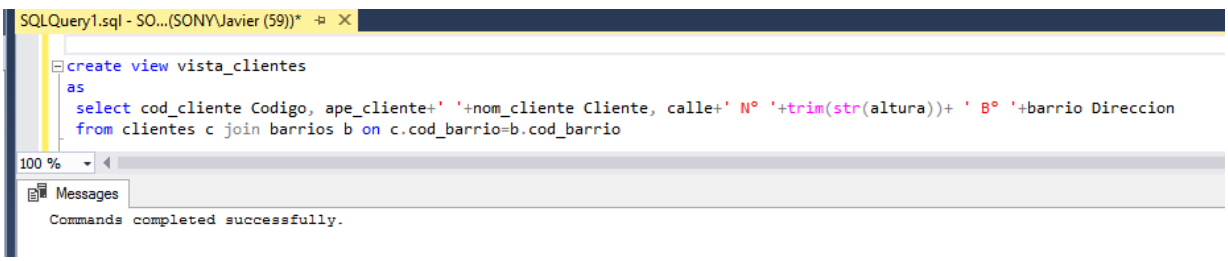


1. **El código y nombre completo de los clientes, la dirección (calle y número) y barrio.**

La sintaxis básica para crear una vista es la siguiente:

```
create view NOMBREVISTA
as
   SENTENCIASSELECT
    from TABLA;
```

Se creará con esta sentencia la vista llamada “vista_clientes”:



Para ver la información contenida en la vista creada anteriormente:

```
select * from vista_clientes;
```

Los campos y expresiones de la consulta que define una vista DEBEN tener un nombre. Se debe colocar nombre de campo cuando es un campo calculado o si hay 2 campos con el mismo nombre.

Los nombres de los campos y expresiones de la consulta que define una vista DEBEN ser únicos (no puede haber dos campos o encabezados con igual nombre)

Se pueden realizar consultas a una vista como si se tratara de una tabla teniendo en cuenta que el nombre de la columna de esta nueva "tabla", es el alias que utilizamos en la consulta por la que se creó la vista. Si el alias contiene caracteres especiales (espacios, guiones medios, signos como %,#,? etc.) al consultar la vista deberemos encerrar estos nombres de columnas entre corchetes.

Listar los clientes que comiencen con A

```
select Cliente,Direccion
from vista_clientes
where Cliente like 'A%';
```

Para quitar una vista se emplea:

```
drop view NOMBREVISTA
```

Para modificar una vista puede hacerlo con "alter view". En el ejemplo siguiente se altera vista_clientes para separar el barrio de la calle y nro.:

```
alter view vista_clientes
as
 select cod_cliente Codigo, ape_cliente+' '+nom_cliente Cliente,
      calle+' Nº '+trim(str(altura)) Direccion,
      barrio Barrio
 from clientes c join barrios b on c.cod_barrio=b.cod_barrio
```

Si crea una vista con "select *" y luego agrega campos a la estructura de las tablas involucradas, los nuevos campos no aparecerán en la vista; esto es porque los campos se seleccionan al ejecutar "create view"; debe alterar la vista.

Continuando con el problema nro. 1.1

2. **Cree una vista que liste la fecha, la factura, el código y nombre del vendedor, el artículo, la cantidad e importe, para lo que va del año. Rotule como FECHA, NRO_FACTURA, CODIGO_VENDEDOR, NOMBRE_VENDEDOR, ARTICULO, CANTIDAD, IMPORTE.**
3. **Modifique la vista creada en el punto anterior, agréguele la condición de que solo tome el mes pasado (mes anterior al actual) y que también muestre la dirección del vendedor.**
4. **Consulta las vistas según el siguiente detalle:**
    a. **Llame a la vista creada en el punto anterior pero filtrando por importes inferiores a $120.**
    b. **Llame a la vista creada en el punto anterior filtrando para el vendedor Miranda.**
    c. **Llama a la vista creada en el punto 4 filtrando para los importes menores a 10.000.**
5. **Elimine las vistas creadas en el punto 3**

Problema N° 3.2: Procedimientos Almacenados

Desde el área de programación del sistema de información de la librería nos solicitan la creación de objetos en la base de datos que permitan ingresar parámetros en el momento de la ejecución de los mismos, dichos objetos en SQL Server son denominados Procedimientos Almacenados:

Para crear un procedimiento almacenado empleamos:

```
CREATE PROCEDURE nombreprocedimiento
 AS instrucciones;
```

Con las siguientes instrucciones se crea un procedimiento almacenado llamado "pa_aarticulos_precios" que muestra todos los articulos cuyo precio unitario es menor a 100:

```
CREATE PROC pa_articulos_precios
 AS
  SELECT * FROM articulos
  WHERE pre_unitario <100;
```

Para ejecutar el procedimiento almacenado creado anteriormente tipeamos:

```
EXEC pa_articulos_precios;
```

Los procedimientos almacenados se eliminan con "DROP PROCEDURE".

Para eliminar el procedimiento almacenado llamado " pa_articulos_precios ":

```
DROP PROCEDURE pa_articulos_precios;
```

Para que un procedimiento almacenado admita parámetros de entrada se deben declarar variables como parámetros al crearlo. La sintaxis es:

```
create proc NOMBREPROCEDIMIENTO
 @NOMBREPARAMETRO TIPO =VALORPORDEFECTO
 as SENTENCIAS;
```

Los parámetros se definen luego del nombre del procedimiento, comenzando con un signo arroba (@). Los parámetros son locales al procedimiento. Pueden declararse varios parámetros por procedimiento, se separan por comas.

Cuando el procedimiento es ejecutado, deben explicitarse valores para cada uno de los parámetros (en el orden que fueron definidos), a menos que se haya definido un valor por defecto luego de definir un parámetro y su tipo; tal valor es el que asume el procedimiento al ser ejecutado si no recibe parámetros. El valor por defecto puede ser "null" o una constante, también puede incluir comodines si el procedimiento emplea "like".

Se crea, en el ejemplo un procedimiento que muestre los artículos cuyo precio unitario sea menor a un precio que se ingresará en el momento en que se ejecute el mismo:

```
CREATE PROC pa_articulos_precios2
@precio money
  AS
   SELECT * FROM articulos
   WHERE pre_unitario <@precio
```

Luego se ejecuta con la siguiente sentencia:

```
exec pa_articulos_precios2 100';
```

Si se quiere listar los artículos cuyo precio esté entre dos valores.

```
CREATE PROC pa_articulos_precios3
@precio1 money,
@precio2 money
  AS
   SELECT * FROM articulos
   WHERE pre_unitario between @precio1 and @precio2

exec pa_ articulos_precios3 @precio1=50, @precio2=100;
```

Cuando se pasa valores con el nombre del parámetro, el orden en que se colocan puede alterarse.

Para que un procedimiento almacenado devuelva un valor se debe declarar una variable con la palabra clave "output" al crear el procedimiento:

```
create procedure NOMBREPROCEDIMIENTO
 @PARAMETROENTRADA TIPO =VALORPORDEFECTO,
 @PARAMETROSALIDA TIPO=VALORPORDEFECTO output
 as
  SENTENCIAS
  select @PARAMETROSALIDA=SENTENCIAS;
```

Crear un procedimiento almacenado que muestre la descripción de un artículo de código determinado (enviado como parámetro de entrada) y nos retorne el total facturado para ese artículo y el promedio ponderado de los precios de venta de ese artículo

```
 Create proc pa_ventas_articulo
@codigo int,
@total decimal(10,2) output,
@precioprom decimal(6,2) output
as
 select descripcion from articulos
 where cod_articulo=@codigo
 select @total=sum(cantidad*pre_unitario)
 from detalle_facturas
 where cod_articulo=@codigo
 select @precioprom=sum(pre_unitario)/sum(cantidad)
 from detalle_facturas
 where cod_articulo=@codigo
```

Ejecutar el procedimiento:

```
  declare @s decimal(12,2), @p decimal(10,2)
 execute pa_ventas_articulo 5, @s output, @p output
 select @s total, @p  'Precio promedio'
```

Crear un procedimiento que muestre todos los libros de un autor determinado que se ingresa como parámetro:

```
create procedure pa_articulos_precios
 @precio decimal(12,2)=null
as
if @precio is null
begin
 select 'Debe indicar un precio'
 return
end;
```

```
select descripcion from  articulos where precio< @precio
```

Crear un procedimiento almacenado que ingresa registros en la tabla "articulos". Los parámetros correspondientes al pre_unitario DEBEN ingresarse con un valor distinto de "null", los demás son opcionales. El procedimiento retorna "1" si la inserción se realiza, es decir, si se ingresan valores para pre_unitario y "0", en caso que pre_unitario sea nulo:

```
create procedure pa_articulos_ingreso
@descripcion nvarchar (50) NULL ,
@stock_minimo smallint NULL ,
@pre_unitario decimal(10, 2) NOT NULL ,
@observaciones nvarchar (50)=null,
as
 if (@pre_unitario is null)
  return 0
 else
 begin
  insert into articulos
values(@descripcion,@stock_minimo,@pre_unitario,@observaciones)
  return 1
 end;
```

Para ver el resultado, debemos declarar una variable en la cual se almacene el valor devuelto por el procedimiento; luego, ejecutar el procedimiento asignándole el valor devuelto a la variable, finalmente mostramos el contenido de la variable:

```
 declare @retorno int
 exec     @retorno=pa_articulos_ingreso     @descripcion     ='Regla
50cm',@pre_unitario=75
 select 'Ingreso realizado=1' = @retorno

 exec @retorno=pa_articulos_ingreso
 select 'Ingreso realizado=1' = @retorno;
```

1. **Cree los siguientes SP:**

   a. **Detalle_Ventas: liste la fecha, la factura, el vendedor, el cliente, el artículo, cantidad e importe. Este SP recibirá como parámetros de E un rango de fechas.**

   b. **CantidadArt_Cli : este SP me debe devolver la cantidad de artículos o clientes (según se pida) que existen en la empresa.**

   c. **INS_Vendedor: Cree un SP que le permita insertar registros en la tabla vendedores.**

d. **UPD_Vendedor: cree un SP que le permita modificar un vendedor cargado.**

e. **DEL_Vendedor: cree un SP que le permita eliminar un vendedor ingresado.**

2. **Modifique el SP 1-a, permitiendo que los resultados del SP puedan filtrarse por una fecha determinada, por un rango de fechas y por un rango de vendedores; según se pida.**

3. **Ejecute los SP creados en el punto 1 (todos).**

4. **Elimine los SP creados en el punto 1.**

Problema N° 3.3: Funciones definidas por el usuario

5. **Cree las siguientes funciones:**
   a. **Hora: una función que les devuelva la hora del sistema en el formato HH:MM:SS (tipo carácter de 8).**
   b. **Fecha: una función que devuelva la fecha en el formato AAAMMDD (en carácter de 8), a partir de una fecha que le ingresa como parámetro (ingresa como tipo fecha).**
   c. **Dia_Habil: función que devuelve si un día es o no hábil (considere como días no hábiles los sábados y domingos). Debe devolver 1 (hábil), 0 (no hábil)**
6. **Modifique la f(x) 1.c, considerando solo como día no hábil el domingo.**
7. **Ejecute las funciones creadas en el punto 1 (todas).**
8. **Elimine las funciones creadas en el punto 1.**

BIBLIOGRAFÍA

Gorman K., Hirt A., Noderer D., Rowland-Jones J., Sirpal A., Ryan D. & Woody B (2019) Introducing Microsoft SQL Server 2019. Reliability, scalability, and security both on premises and in the cloud. Packt Publishing Ltd. Birmingham UK

Microsoft (2021) SQL Server technical documentation. Disponible en: https://docs.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver15

Opel, A. & Sheldon, R. (2010). Fundamentos de SQL. Madrid. Editorial Mc Graw Hill

Varga S., Cherry D., D'Antoni J. (2016). Introducing Microsoft SQL Server 2016 Mission-Critical Applications, Deeper Insights, Hyperscale Cloud. Washington. Microsoft Press