	UNIVERSIDAD DON BOSCO **FACULTAD DE INGENIERÍA** **ESCUELA DE COMPUTACION**
CICLO II-2020	**GUIA DE LABORATORIO Nº9** **Nombre de la practica**: Creación de Vistas y Procedimientos Almacenados **Lugar de ejecución**: Laboratorio de Informática **Tiempo estimado**: 2 horas y 30 minutos **Materia**: MODELAMIENTO DE BASE DE DATOS

I. Objetivos

Qué el estudiante:

1. Diseñar y crear Vistas para la selección de información

2. Implementar la programación con comandos SQL

II. Introducción Teórica

Vistas

Las vistas tiene una tendencia a ser utilizadas mucho o poco: en raras ocasiones se utilizan en su justa medida. Las vistas se podrían utilizar para:

a. Reducir la complejidad aparente de la base de datos para los usuarios finales

b. Prevenir la selección de columnas confidenciales a la vez que permite el acceso a otros datos importantes En el fondo, una vista no es más que una consulta almacenada. lo extraordinario es que podemos combinar y hacer corresponder datos desde tablas base (o desde otras vistas) para crear lo que, en general, funciona como cualquier otra tabla base.

Al crear una vista, Microsoft SQL Server comprueba la existencia de los objetos a los que se hace referencia en su definición. El nombre de la vista debe ajustarse a las normas para los identificadores. Opcionalmente, es posible especificar un nombre de propietario para la vista. Debe establecer una convención de denominación coherente para distinguir las vistas de las tablas. Por ejemplo, puede agregar la palabra “vista” como sufijo de cada objeto vista que cree. De este modo podrá distinguir fácilmente entre objetos similares (tablas y vistas) al consultar la vista **INFORMATION_SCHEMA.TABLES**.

Sintaxis

CREATE VIEW *nombreVista* [(*columna* [,*n*])] [WITH {ENCRYPTION | SCHEMABINDING | VIEW_METADATA} [,n]] AS *instrucciónSelect*

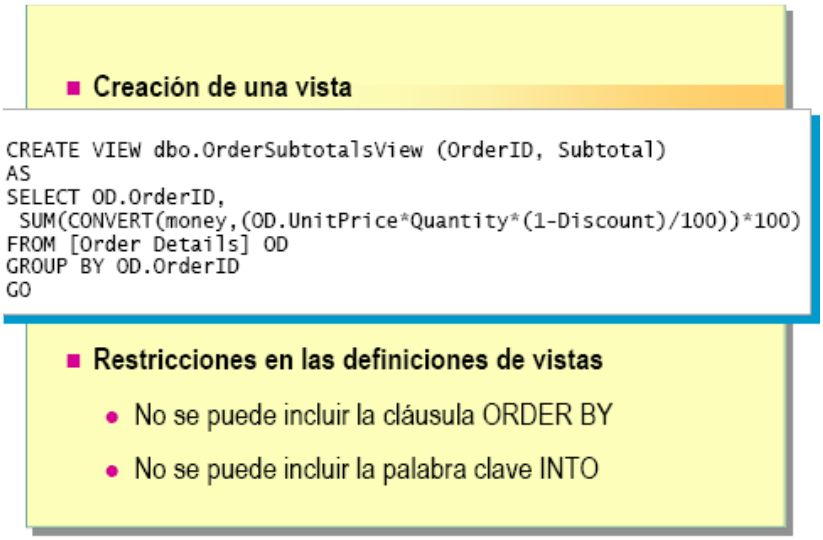
[WITH CHECK OPTION]

Para poder ejecutar la instrucción CREATE VIEW es necesario ser miembro de la función de administradores del sistema (**sysadmin**), de la función propietario de la base de datos (**db_owner**) o de la función administrador de lenguaje de definición de datos (**db_ddladmin**), o bien tener el permiso CREATE VIEW. También es necesario tener el permiso SELECT en todas las tablas o vistas a las que la vista haga referencia.

Para evitar situaciones en las que el propietario de una vista y el propietario de las tablas subyacentes sean distintos, se recomienda que el usuario **dbo** (propietario de base de datos) sea el propietario de todos los objetos de la base de datos. Especifique siempre el usuario **dbo** como propietario al crear el objeto pues, de lo contrario, usted, es decir, su nombre de usuario, será el propietario.

El contenido de una vista se especifica con una instrucción SELECT. Con algunas excepciones, las vistas pueden ser tan complejas como se requiera. Debe especificar los nombres de columna en las situaciones siguientes:

- Alguna de las columnas de la vista se deriva de una expresión aritmética, de una función integrada o de una constante.
- Hay columnas con el mismo nombre en las tablas que se van a combinar.



Ejemplo

Este es un ejemplo de una vista que crea una columna (**Subtotal**) que calcula los subtotales de un pedido a partir de las columnas **UnitPrice**, **Quantity** y **Discount** de la tabla **Order Details**.

```
CREATE VIEW dbo.OrderSubtotalsView (OrderID, Subtotal)
AS
      SELECT OD.OrderID,
      SUM(CONVERT (money,(OD.UnitPrice*Quantity*(1-Discount)/100))*100)
      FROM [Order Details] OD
      GROUP BY OD.OrderID
GO
```

En este ejemplo se consulta la vista para ver los resultados.

```
SELECT * FROM OrderSubtotalsView
```

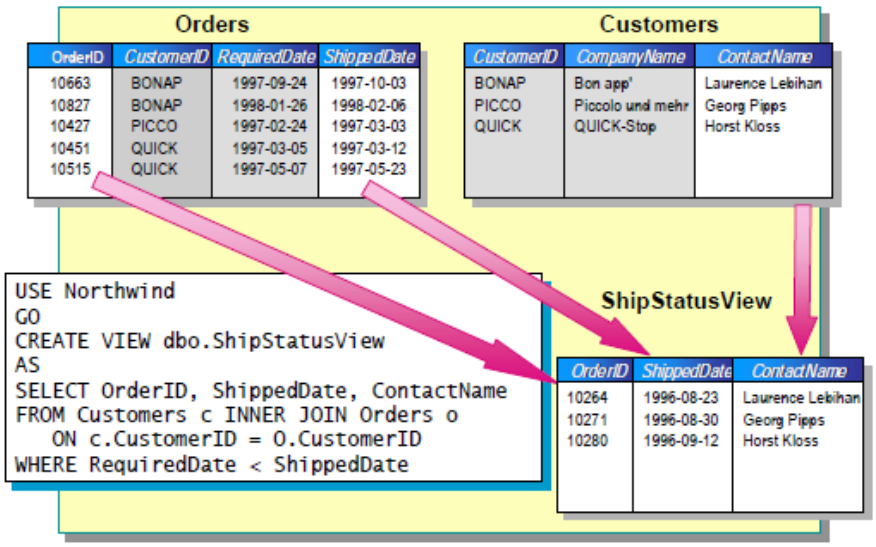
Resultado

	OrderID	Subtotal
1	10248	440.00
2	10249	1863.40
3	10250	1552.60
4	10251	654.06
5	10252	3597.90
6	10253	1444.80
7	10254	556.62
8	10255	2490.50

El resultado da un total de 830 filas

Vista de tablas combinadas

A menudo se crean vistas para conseguir una forma más práctica de ver información centralizada de dos o más tablas combinadas.




```
CREATE VIEW dbo.ShipStatusView
AS
	SELECT OrderID, ShippedDate, ContactName
	FROM Customers c INNER JOIN Orders o
	ON c.CustomerID=o.CustomerID
	WHERE RequiredDate<ShippedDate
GO
```

Resultado

	OrderID	ShippedDate	ContactName
1	10280	1996-09-12 00:00:00.000	Christina Berglund
2	10924	1998-04-08 00:00:00.000	Christina Berglund
3	10970	1998-04-24 00:00:00.000	Martín Sommer
4	10827	1998-02-06 00:00:00.000	Laurence Lebihan
5	10663	1997-10-03 00:00:00.000	Laurence Lebihan
6	10578	1997-07-25 00:00:00.000	Victoria Ashworth
7	10726	1997-12-05 00:00:00.000	Ann Devon
8	10264	1996-08-23 00:00:00.000	Maria Larsson
9	10807	1998-01-30 00:00:00.000	Paolo Accorti
10	10777	1998-01-21 00:00:00.000	André Fonseca

El resultado da un total de 37 filas

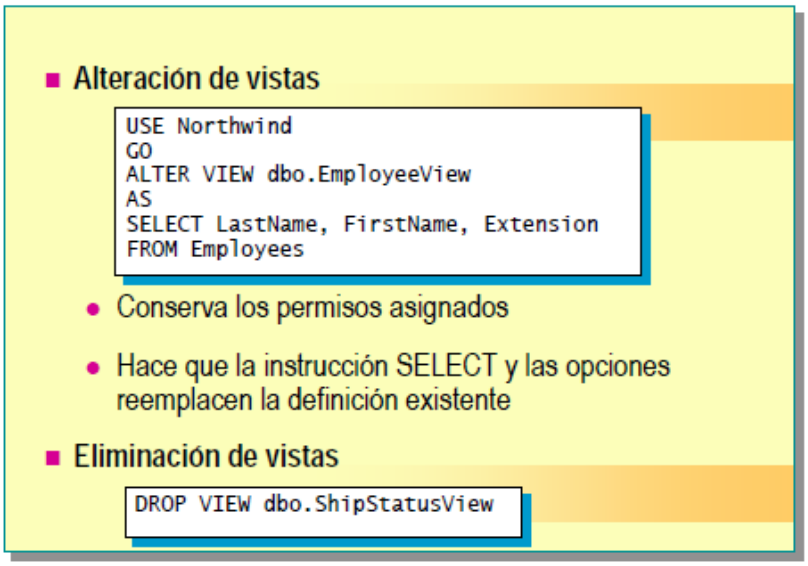
Modificación y eliminación de vistas

A menudo, las vistas se alteran como respuesta a las peticiones de información adicional por parte de los usuarios o a causa de cambios en la definición de las tablas subyacentes. Para alterar una vista puede quitarla y volverla a crear, o bien puede ejecutar la instrucción **ALTER VIEW.**

Eliminación de vistas

Si ya no necesita una vista, puede quitar su definición de la base de datos con la instrucción DROP VIEW. Al quitar una vista se quita su definición y todos los permisos que tenga asignados. Además, si los usuarios consultan vistas que hagan referencia a la vista quitada, obtendrán un mensaje de error. Sin embargo, al quitar una tabla que hace referencia a una vista, ésta no se quita automáticamente. Es necesario quitarla de forma explícita. Se utiliza la instrucción DROP VIEW.

Ejemplo.



Ejemplo

Agregar un nuevo campo (CompanyName de la tabla Customers) a la consulta ShipStatusView

```
ALTER VIEW dbo.ShipStatusView
AS
      SELECT OrderID, ShippedDate, ContactName, CompanyName
      FROM Customers c INNER JOIN Orders o
      ON c.CustomerID=o.CustomerID
      WHERE RequiredDate<ShippedDate
GO
```

Resultado

	OrderID	ShippedDate	ContactName	CompanyName
1	10280	1996-09-12 00:00:00.000	Christina Berglund	Berglunds snabbköp
2	10924	1998-04-08 00:00:00.000	Christina Berglund	Berglunds snabbköp
3	10970	1998-04-24 00:00:00.000	Martín Sommer	Bólido Comidas preparadas
4	10827	1998-02-06 00:00:00.000	Laurence Lebihan	Bon app'
5	10663	1997-10-03 00:00:00.000	Laurence Lebihan	Bon app'
6	10578	1997-07-25 00:00:00.000	Victoria Ashworth	B's Beverages
7	10726	1997-12-05 00:00:00.000	Ann Devon	Eastern Connection

Eliminar la vista ShipStatusView

```
DROP VIEW ShipStatusView
```

PROCEDIMIENTOS ALMACENADOS

Transact-SQL no es realmente un lenguaje de programación similar a las herramientas de tercera y cuarta generación sin embargo permite utilizar SQL para realizar tareas complejas que requieren saltos, bucles, decisiones. **Transact-SQL** se utiliza a menudo en la creación de procedimientos almacenados y triggers (desencadenadores) de tal forma que las aplicaciones clientes que se conectan a SQL Server solo se preocupan por la presentación de los datos para el usuario final, mientras que la lógica de los procesos se maneja en el servidor.

El control de flujo en Transact-SQL

INSTRUCCIÓN	DESCRIPCIÓN
IF…ELSE	Define una decisión.
GOTO etiqueta	Define un salto incondicional
WAITFOR	Establece un tiempo para la ejecución de una instrucción. El tiempo puede ser un intervalo de retardo o un instante especificado de ejecución (una hora concreta del día)
WHILE	Bucle básico de SQL
BREAK	Acompaña al bucle WHILE y le indica finalizarlo inmediatamente.
CONTINUE	Acompaña al bucle WHILE y le indica continuar con la siguiente iteración.
RETURN [n]	Salida incondicional del procedimiento o proceso por lotes, se puede definir un número entero como estado devuelto y puede asignarse a cualquier variable.
BEGIN...END	Utilizado en conjunto con IF..ELSE o WHILE para agrupar un conjunto de instrucciones.
CASE	Implementada en la instrucción SELECT y UPDATE y permite realizar consultas y actualizaciones condicionales.
PRINT	Es una instrucción que se utiliza para imprimir un dato en la pantalla, la sintaxis es: **PRINT** "cadena"; cadena puede ser también una variable de tipo varchar. Por ejemplo: **PRINT** 'Hola a todos'

Los procedimientos almacenados son conjuntos de sentencias en leguaje Transact SQL que pueden almacenarse en el propio servidor. Los procedimientos almacenados de SQL Server, son más potentes, porque permiten almacenar funciones y procedimientos compuestos por varias instrucciones, introducir saltos, bucles, etc.

Creación de procedimientos almacenados (Store Procedures)

La instrucción general para crear procedimientos almacenados es la siguiente:

CREATE PROC nombre_proc parametros
AS
INSTRUCCION SQL

Es necesario aclarar, que un procedimiento almacenado puede recibir parámetros de entrada y devolver parámetros de salida.

Por ejemplo, podemos crear un procedimiento para recuperar el nombre de un cliente (), cuyo código se pasa por parámetro.

```
CREATE PROCEDURE ObtenerNombre @cliente_id nchar(5) AS
SELECT ContactName
FROM Customers WHERE CustomerID = @cliente_id
```

Con esta sentencia, se crea un procedimiento almacenado, de nombre ObtenerNombre, al que se le pasa un parámetro, llamado @cliente_id, de tipo nchar tamaño 5, que realiza una consulta para obtener el nombre del cliente de la tabla Customers, cuyo código (CustomerID) coincida con el parámetro. De esta forma, si queremos obtener el nombre del Cliente cuyo código sea 'ALFKI', deberemos ejecutar el procedimiento pasándole como argumento este valor.

Las llamadas a procedimientos almacenados se pueden realizar de las siguientes formas:

- Pasando los argumentos en el mismo orden que en el que se han declarado.

```
EXEC ObtenerNombre 'ALFKI'
```

- Pasando los argumentos nombrados. En este caso no hace falta que los parámetros vayan en el mismo orden.

```
EXEC ObtenerNombre @cliente_id='ALFKI'
```

Parámetros por referencia

Si ejecutamos las anteriores sentencias, obtendremos el resultado directamente en la ventana que tengamos abierta en SQL Server. Pero ¿qué pasa si queremos obtener un parámetro de salida, como resultado de la ejecución del procedimiento? La solución para este caso es utilizar la palabra reservada OUTPUT para los argumentos de salida.

Si por ejemplo, queremos obtener el número de títulos que tenemos en la base de datos, crearemos el siguiente procedimiento almacenado:

```
CREATE PROCEDURE num_productos @productos int OUTPUT AS
SELECT * FROM Products
SELECT @productos = @@ROWCOUNT
RETURN (0)
```

Vamos a estudiar el anterior ejemplo:
Detrás de la palabra reservada PROCEDURE damos el nombre del procedimiento almacenado, y a continuación proporcionamos los parámetros, junto con su tipo (que en este caso es entero), y diremos si éstos son de salida, en cuyo caso especificamos la palabra reservada **OUTPUT** a continuación. Tras la palabra reservada **AS** se codifica el cuerpo del procedimiento.

Primero contamos todas las filas de la tabla Products, realizando un SELECT * FROM Products.
A continuación devolvemos en el parámetro @productos el valor obtenido, utilizando *@@ROWCOUNT.*

Para ejecutar este procedimiento ejecutamos las siguientes sentencias (las tres al mismo tiempo):

```
DECLARE @productos int
EXEC num_productos @productos OUTPUT
SELECT [Total de productos] = @productos
```

En el resultado de la consulta nos mostrará los registros de la tabla Products y en una nueva fila el valor de la variable @productos en una columna llamada Total de productos

	ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit	UnitPrice	Unit
71	71	Flotemysost	15	4	10 - 500 g pkgs.	21,50	26
72	72	Mozzarella di Giovanni	14	4	24 - 200 g pkgs.	34,80	14
73	73	Röd Kaviar	17	8	24 - 150 g jars	15,00	101
74	74	Longlife Tofu	4	7	5 kg pkg.	10,00	4
75	75	Rhönbräu Klosterbier	12	1	24 - 0.5 l bottles	7,75	125
76	76	Lakkalikööri	23	1	500 ml	18,00	57
77	77	Original Frankfurter grüne Soße	12	2	12 boxes	13,00	32

	Total de productos
1	77

Borrar Procedimientos Almacenados

Si queremos borrar un procedimiento almacenado, ejecutaremos la sentencia DROP PROCEDURE, seguido del nombre del procedimiento. Por ejemplo, si queremos borrar el procedimiento almacenado, creado en el anterior ejemplo, escribiremos el Código:

```
DROP PROCEDURE num_productos
```

III. Requerimientos

Nº	Cantidad	Descripción
1	1	Guía de Laboratorio #9
2	1	Maquina con SQL Server

IV. Procedimiento

Parte 1: Iniciando sesión desde SQL Server Managment Studio

1. Hacer clic en el botón **Inicio**

2. Hacer clic en la opción **Todos los programas** y hacer clic en **Microsoft SQL Server**

Para conectarse con el servidor de base de datos elija los siguientes parámetros de autenticación:

- **Tipo de servidor:** Database Engine
- **Nombre del servidor:** Colocar el nombre del servidor local, por ejemplo PC**NumMaquina-SALA#**
  Nota: NumMaquina es el número de la maquina local
- **Autenticación**:
- **Login:**
- **Password:**

3. Luego, presione Ctrl+N o de clic en **"New Query"** en la barra de trabajo estándar.

Parte 2: Ejemplos de Vistas

Ejercicio 1

En este ejercicio va a crear una vista, de datos almacenados en la base de datos **Northwind**.

La vista debe mostrar los productos de un proveedor en particular, el nombre de la vista será: **productos_proveedor_SuCarnet** (**cambie la palabra SuCarnet por su número de carnet en los ejercicios de la Guía**)

1. Seleccionar la base de datos **Northwind**

2. Seleccionar la tabla **Products**

3. Seleccionar las columnas:
   ProductID, ProductName, SupplierID, CategoryID, QuantityPerUnit, UnitPrice, Discontinued

4. Definir una restricción WHERE la cual es: SupplierID = 14

```
CREATE VIEW productos_proveedor_SuCarnet
AS
     SELECT ProductID, ProductName, SupplierID, CategoryID,QuantityPerUnit, UnitPrice,
     Discontinued
     FROM Products
     WHERE SupplierID = 14
GO
```

5. Consulte la vista para asegurarse de que ha recibido el conjunto de resultados esperado.

```
SELECT * FROM productos_proveedor_SuCarnet
```

6. Modificar la vista anterior y ahora se debe mostrar también el nombre del proveedor (CompanyName) el cual está almacenado en la tabla Suppliers

7. Eliminar la vista

Ejercicio 2

1. Tomado en cuenta la misma base de datos Northwind, debe crear una vista donde se muestren las ventas de un producto las cuales se han realizado en el año 1997, los campos que podría mostrar son los siguientes:
   a. Nombre del producto (ProductName)
   b. Fecha de pedido (OrderDate)
   c. Fecha de envió (ShippedDate)
   d. Calcular el subtotal (UnitPrice*Quantity)

2. Ordenar los resultados por el campo ProductName

3. Nombre de la vista: PedidosProductos1997_Sucarnet

Ejercicio 3

1. Se desea crear una vista la cual muestre la sumatoria de los subtotal

2. Los campos o cálculos a utilizar son:
   a. Nombre del producto (ProductName)
   b. Calcular el Subtotal (UnitPrice*Quantity), el campo UnitPrice es el que está almacenado en la tabla Order Details
   c. Calcular el Total (Subtotal-(Subtotal*Discount))

3. Para los puntos b y c debe implementar la función SUM

4. Los resultados se agruparan por el campo ProductName

5. Ordenar los resultados por el campo ProductName

6. Nombre de la vista TotalProductos_SuCarnet

Parte 3: Ejemplos de Procedimientos almacenados

Ejercicio 1

1. Utilice la bases de datos **NorthWind**

2. Vamos a crear un procedimiento almacenado que ingrese un nuevo registro a la tabla **Categories** solo si el Código o el Nombre de la categoría que se le envían como parámetros no existan en la tabla.

```
CREATE PROCEDURE sp_Insertar_CategoriasSuCarnet
@ID INT,
@NombreCategoria VARCHAR(15)
AS
IF(SELECT COUNT(*) FROM Categories
       WHERE CategoryID=@ID OR CategoryName=@NombreCategoria)=0
       INSERT INTO Categories(CategoryName)
       VALUES(@NombreCategoria)
ELSE
PRINT 'Error la categoría ya existe'
```

3. Ejecutemos el procedimiento anterior comprobando si funciona correctamente, ejecutar cada instrucción individualmente:

```
--Ejecutar primero
EXEC sp_Insertar_CategoriasSuCarnet 1, 'Alimentos'
--Segundo
EXEC sp_Insertar_CategoriasSuCarnet 9, 'Alimentos'
--Tercero
EXEC sp_Insertar_CategoriasSuCarnet 10, 'Alimentos'
```

Nota: si los datos del ejemplo anterior existen en la tabla probar con otros datos

4. Hacer un SELECT y verifique cuantos registros se agregaron a la tabla Categories

```
SELECT * FROM Categories
```

	CategoryID	CategoryName	Description	Pict
1	1	Beverages	Soft drinks, coffees, teas, beers, and ales	0x1
2	2	Condiments	Sweet and savory sauces, relishes, spreads, and ...	0x1
3	3	Confections	Desserts, candies, and sweet breads	0x1
4	4	Dairy Products	Cheeses	0x1
5	5	Grains/Cereals	Breads, crackers, pasta, and cereal	0x1
6	6	Meat/Poultry	Prepared meats	0x1
7	7	Produce	Dried fruit and bean curd	0x1
8	8	Seafood	Seaweed and fish	0x1
9	9	Alimentos	NULL	NU

Ejercicio 2

1. En el siguiente procedimiento hacemos uso de la sentencia CASE para verificar si existen clientes en una determinada ciudad la cual su dato se pasa como parámetro.

```
CREATE PROCEDURE sp_Hay_ClientesSuCarnet
@ciudad varchar(15)
AS
SELECT
CASE (SELECT COUNT(*) FROM Customers WHERE City=@ciudad)
WHEN 0 THEN 'No hay clientes en la ciudad de ' + @ciudad
ELSE 'Hay clientes en la ciudad de ' + @ciudad
END
```

2. Ejecutemos el procedimiento anterior comprobando si funciona correctamente:

```
EXEC sp_Hay_ClientesSuCarnet 'Barcelona'
EXEC sp_Hay_ClientesSuCarnet 'New York'
```

Ejercicio 3.

1. En el siguiente procedimiento se calcula el total de ventas de los pedidos que están almacenados en la tabla **Order Details**

```
CREATE PROCEDURE PROCE_TotalSuCarnet
AS
    SELECT OrderID,
    SUM(CONVERT (money,(UnitPrice*Quantity*(1-Discount)/100))*100)
    AS Total
    FROM [Order Details]
    GROUP BY OrderID
GO
```

La función CONVERT () es una función general que convierte una expresión de un tipo de dato a otro.

2. Ejecutar el procedimiento

```
EXEC  PROCE_TotalSuCarnet
```

3. Ahora vamos a alterar o modificar el procedimiento para que reciba como parámetro de entrada una orden o pedido especifico

```
ALTER PROCEDURE PROCE_TotalSuCarnet
@Cod_orden int
AS
       SELECT OrderID,
       SUM(CONVERT (money,(UnitPrice*Quantity*(1-Discount)/100))*100)
AS Total
FROM [Order Details]
WHERE OrderID=@Cod_orden
GROUP BY OrderID
GO
```

4. Ejecutar el procedimiento

```
EXEC PROCE_TotalSuCarnet 10248
```

5. Ahora ejecutamos el mismos procedimiento con otro código de pedido

```
EXEC PROCE_TotalSuCarnet 10242
```

Analice y comente la diferencia entre los resultados obtenidos, como se dará cuenta el OrderID 10242 no existe en la tabla, pero al ejecutarse se muestran las columnas con datos vacíos

6. Se puede modificar el procedimiento de la siguiente manera, en donde se muestra un mensaje indicando que no existe esa orden

```
ALTER PROCEDURE PROCE_TotalSuCarnet
@Cod_orden int
AS
DECLARE @total INT --Declaracion de variables locales
SELECT @total=COUNT(orderid) FROM [Order Details] WHERE OrderID=@Cod_orden
IF (@total>=1) --Evalua condicion
  BEGIN
       SELECT OD.OrderID,
       SUM(CONVERT (money,(OD.UnitPrice*Quantity*(1-Discount)/100))*100) as Total
       FROM [Order Details] OD
       WHERE OrderID=@Cod_orden
       GROUP BY OD.OrderID
  END
ELSE
  BEGIN
       PRINT 'La orden no existe y el código es: ' +CONVERT(varchar(10),@Cod_orden)
  END
GO
```

7. Ejecutar el procedimiento

```
--Codigo de pedido que si existe en la tabla Order Details
EXEC PROCE_TotalSuCarnet 10248

--Codigo de pedido que no existe en la tabla Order Details
EXEC PROCE_TotalSuCarnet 10242
```

8. Verifique y analice los resultados

9. Guardar el script con el nombre **Guia9_BasedeDatosSucarnet.sql**

V. Ejercicio complementario

Creación de Vistas

Ejercicio 1

La base de datos a utilizar es: **Northwind**

Crear las siguientes **vistas**:

a. Mostrar el código del producto, el nombre del producto y el precio por unidad de todos los productos de la empresa.

b. Mostrar todos los productos cuya categoría sea Beverages.

c. Mostrar los datos del cliente y las fechas de las ordenes que estos han realizado

d. Cuantos productos existen por cada categoría

e. Mostrar el promedio de los precios unitarios de las categorías: Produce y Confections

Ejercicio 2

Crear un procedimiento almacenado que calcule y muestre la edad de un empleado (Tabla Employees, campo BirthDate), puede utilizar la siguiente instrucción para calcular la edad:

(CAST(DATEDIFF(dd, BirthDate,GETDATE())/365.25 as int))

Y por medio de una sentencia CASE deberá calcular y mostrar si el empleado esta por retirarse (mayores de 60 y menores o igual a 70 años), si ya está jubilado (mayores a 70 años) o le falta otros años para trabajar (menores o iguales a 60 años)

El procedimiento recibe como parámetros el código del empleado (EmployeeID) si el empleado no existe deberá mostrar un mensaje indicándolo.

VI. Análisis de resultados

Uso de Procedimiento almacenado

```
CREATE DATABASE Bodega
GO

USE Bodega
GO

CREATE TABLE PRODUCTO
(
idprod char(7) PRIMARY KEY,
descripcion varchar(25),
existencias int, --cantidad de producto existentes
precio  decimal(10,2) not null, --precio costo
preciov  decimal(10,2) not null, --precio venta
ganancia as preciov - precio, --campo calculado para calcular la ganancia
CHECK(preciov>precio) --precio venta tiene que ser mayor al precio de compra
)
GO


CREATE TABLE PEDIDO
(
idpedido char(7),
idprod char(7),
cantidad int  --cantidad de unidades vendidas del producto en el pedido
FOREIGN KEY(idprod) REFERENCES PRODUCTO(idprod)
)
```

La tabla PRODUCTO contiene información general de los productos y la tabla PEDIDO contiene la información del pedido que se realiza de un cierto producto, llenar las tablas con información.

Ejercicios:

1. Crear un procedimiento almacenado que ingrese los valores en la tabla PRODUCTOS, y deberá verificar que el código y nombre del producto no exista para poder insertarlo, en caso que el código o el nombre del producto ya exista enviar un mensaje que diga “ESTE PRODUCTO YA HA SIDO INGRESADO”.

2. Crear un procedimiento almacenado que permita realizar un pedido EN LA TABLA PEDIDO, este procedimiento deberá verificar si el código del producto ingresado existe en la tabla PRODUCTO en caso de que no se encuentre deberá mostrar un mensaje así como se muestra a continuación “ESTE PRODUCTO NO EXISTE “, además si la cantidad a pedir del producto es mayor a la existencia del producto deberá mostrar un mensaje que diga “EXISTENCIA DEL PRODUCTO INSUFICIENTE”.
   En caso que la cantidad a pedir sea menor o igual deberá modificar (o actualizar) el valor de la existencia del producto.

VI. Referencia Bibliográfica

1. La Biblia de SQL Server 2005
   Madrid, España: Anaya, 2006
   Autor: Mike Gundelerloy y Joseph L. Jorden
   Biblioteca UDB – Clasificación: 005.361 G975 2006

2. Fundamentos: de Programación con SQL Server 2005
   Madrid, España: ANAYA, 2006
   Autor: Roberto Vieira
   Biblioteca UDB – Clasificación: 005.361 V665

3. SQL Server 2008
   Madrid, España: ANAYA, 2009
   Autor: Francisco Charte Ojeda
   Biblioteca UDB – Clasificación: 005.361 Ch436 2009