Programación
Laboratorios

Microsoft

Módulo 2: Introducción al Lenguaje SQL

Ejercicio 1
Escribiendo Sentencias SELECT básicas

En este ejercicio escribirá varias sentencias SELECT que devuelven registros de la tabla Production.Product de la base de datos **AdventureWorks**.

1. Escribir y ejecutar una sentencia SELECT que devuelva todos los registros y todas las columnas de la tabla Production.Product y ordenar el resultado en forma ascendente por la columna Name (Puede ejecutar el procedimiento almacenado de sistema ***sp_help*** sobre la tabla Production.Product para encontrar los nombres correctos de las columnas.)

2. Escribir y ejecutar una sentencia SELECT que devuelva los productos de la tabla Production.Product cuya subcategoría (columna ProductSubcategoryID) es la número 1.

3. Escribir y ejecutar la misma sentencia SELECT del punto 2 mostrando solo los campos ProductId, Name y ProductNumber. Renombre los 3 campos como Codigo, Nombre y NumeroProducto respectivamente.

4. Escribir y ejecutar una sentencia SELECT que devuelva los productos cuyo precio (Columna ListPrice) este comprendido entre $500 y $1000. Ordenar en forma descendente por precio.

5. Escribir y ejecutar una sentencia SELECT que devuelva los productos cuyo número de producto (columna ProductNumber) comience con LJ.

6. Escribir y ejecutar una sentencia SELECT que devuelva los productos de color blanco o negro (columna Color – White/Black)

7. Escribir y ejecutar una sentencia SELECT que devuelva los productos cuya columna Color está en nulo.

8. Escribir y ejecutar una sentencia SELECT que devuelva solo los colores posibles para los productos. Los colores deben aparecer solo una vez en la lista.

9. Escribir y ejecutar una sentencia SELECT que devuelva el margen (ListPrice - StandardCost) de todos los productos de la tabla Production.Product

10. Escribir y ejecutar una sentencia SELECT que devuelva los 10 productos más caros (Columna ListPrice)

11. Escribir y ejecutar una sentencia SELECT que devuelva la cantidad de productos por subcategoría (columna ProductSubcategoryID). Como segundo paso modificar la instrucción para que muestre solo las subcategorías que tienen más de 20 productos asociados.

12. Modificar la sentencia del punto 11 para incluir el nombre de la subcategoría tomándolo de la tabla Production.ProductSubCategory

13. Escribir y ejecutar una sentencia SELECT que devuelva el código y nombre del producto, con el nombre de la subcategoría a la que pertenece y el nombre del modelo asociado. Use las tablas tabla Production.ProductSubCategory y tabla Production.Model para encontrar los valores buscados.

14. Escribir y ejecutar una sentencia SELECT que devuelva el código y nombre del producto, con el nombre de la subcategoría a la que pertenece. Ordenado por categoría. Después modifique la sentencia para que muestre todos los productos aunque estos no estén asociados a una subcategoría.

15. Escribir y ejecutar una sentencia SELECT que devuelva los datos Productid, TransactionDate, TransactionType, Quantity, ActualCost de las tabla Production.TransactionHistory unido a los mismo datos de la tabla Production.TransactionHistoryArchive. Ordenar el resultado por ProductId.

Respuestas

1. SELECT * FROM Production.Product ORDER BY Name

   o

   ```
   SELECT ProductID, [Name], ProductNumber, MakeFlag, FinishedGoodsFlag, Color,
   SafetyStockLevel, ReorderPoint, StandardCost, ListPrice, [Size],
   SizeUnitMeasureCode, WeightUnitMeasureCode, Weight, DaysToManufacture,
   ProductLine, Class, Style, ProductSubcategoryID, ProductModelID,
   SellStartDate, SellEndDate, DiscontinuedDate, rowguid, ModifiedDate
   FROM Production.Product
   ORDER BY Name

   Exec sp_help 'Production.Product'
   ```

2. SELECT * FROM Production.Product WHERE ProductSubCategoryId=1

3. SELECT ProductId as Codigo, Name as Nombre, ProductNumber as NumeroProducto
   FROM Production.Product
   WHERE ProductSubCategoryId=1

4. SELECT * FROM Production.Product
   WHERE ListPrice Between 500 and 1000
   ORDER BY ListPrice DESC

   o

   SELECT * FROM Production.Product
   WHERE ListPrice >= 500 and ListPrice <= 1000
   ORDER BY ListPrice DESC

5. SELECT * FROM Production.Product
   WHERE ProductNumber like 'LJ%'

6. SELECT * FROM Production.Product

```
WHERE color in ('White','Black')
ORDER BY Color
```

o

```
SELECT * FROM Production.Product
WHERE color = 'White' or color = 'Black'
ORDER BY Color
```

7.
```
SELECT * FROM Production.Product
WHERE color is null
```

8.
```
SELECT Distinct Color FROM Production.Product
ORDER BY color
```

9.
```
SELECT Productid, Name, Margen=ListPrice - StandardCost
FROM Production.Product
```

o

```
SELECT Productid, Name, ListPrice - StandardCost as Margen
FROM Production.Product
```

10.
```
SELECT TOP 10  * FROM Production.Product
ORDER BY ListPrice DESC
```

o

```
SELECT TOP 10 WITH TIES * FROM Production.Product
ORDER BY ListPrice DESC
```

11.
```
SELECT ProductSubcategoryID, Count(*) as CantidadProductos
FROM Production.Product
GROUP BY ProductSubcategoryID
ORDER BY ProductSubcategoryID
```

```
SELECT ProductSubcategoryID, Count(*) as CantidadProductos
FROM Production.Product
GROUP BY ProductSubcategoryID
HAVING Count(*) > 20
ORDER BY ProductSubcategoryID
```

12.
```
SELECT p.ProductSubcategoryID, c.Name, Count(*) as CantidadProductos
FROM Production.Product p INNER JOIN Production.ProductSubCategory c
ON p.ProductSubcategoryID = c.ProductSubcategoryID
GROUP BY p.ProductSubcategoryID, c.Name
HAVING Count(*) > 20
ORDER BY p.ProductSubcategoryID
```

13.
```
SELECT p.ProductId, p.Name as NombreProducto, c.Name as NombreCategoria,
m.Name as NombreModelo
FROM Production.Product p INNER JOIN Production.ProductSubCategory c
ON p.ProductSubcategoryID = c.ProductSubcategoryID
INNER JOIN Production.ProductModel m ON p.ProductModelID=m.ProductModelID
```

14.
```
SELECT p.ProductId, p.Name as NombreProducto, c.Name as NombreCategoria
FROM Production.Product p INNER JOIN Production.ProductSubCategory c
ON p.ProductSubcategoryID = c.ProductSubcategoryID
ORDER BY c.Name

SELECT p.ProductId, p.Name as NombreProducto, c.Name as NombreCategoria
FROM Production.Product p LEFT JOIN Production.ProductSubCategory c
ON p.ProductSubcategoryID = c.ProductSubcategoryID
ORDER BY c.Name
```

15.
```
SELECT Productid, TransactionDate, TransactionType, Quantity, ActualCost
FROM Production.TransactionHistory
UNION ALL
SELECT Productid, TransactionDate, TransactionType, Quantity, ActualCost
FROM Production.TransactionHistoryArchive
ORDER BY Productid
```

Ejercicio 2
Escribiendo Sentencias INSERT

En este ejercicio escribirá varias sentencias INSERT para grabar información en tablas de la base de datos **AdventureWorks**.

1. Ejecutar el procedimiento almacenado de sistema llamado ***sp_help*** para determinar que columnas de la tabla Person.StateProvince aceptan valores nulos o tienen valores predeterminados. No es necesario ingresar valores para estas columnas. También verifique si alguna columna está marcada con la propiedad IDENTITY. Puede hacer esta verificación usando el SQL Server Management Studio.

2. Escribir y ejecutar una sentencia INSERT que agregue registros en la tabla Person.StateProvince. Solo ingrese la información para las columnas que no aceptan nulo, no tienen valores predeterminado, ni están marcadas con la propiedad IDENTITY. Ingrese para el país Argentina (AR) las provincias Buenos Aires, Córdoba y Mendoza. Los código de cada provincia deberán ser BA, CO y MZ respectivamente. Para el territorio ingrese el valor 1.

3. Escribir y ejecutar una consulta que determine el último valor StateProvinceId agregado en el paso anterior.

4. Verificar que los valores fueron agregados a la tabla.

5. Levantar el archivo <carpeta de instalación>\Modulo2***ProvinciasArgentinas.sql*** dentro del SQL Server Management Studio. Ejecutarlo. Este procedimiento crea una nueva tabla llamada ProvinciasArgentinas en la base de datos AdventureWorks y le agrega varios registros.

6. Escribir y ejecutar una sentencia INSERT de manera de ingresar todos los valores de esta tabla ProvinciasArgentinas a la tabla StateProvinceId en un solo paso.

7. Verificar que los valores fueron agregados a la tabla.

Respuestas

1. exec sp_help 'Person.StateProvince'
   La columna StateProvinceId: Identity
   La columna IsOnlyStateProvinceFlag: Default: 1
   La columna ModifiedData: Default: GetDate()
   La columna rowguid: Default: NewId()
   Ninguna columna soporta valores nulos.

2. INSERT INTO Person.StateProvince (StateProvinceCode, CountryRegionCode, Name, TerritoryId)
   VALUES ('BA', 'AR', 'Buenos Aires', 1)

   INSERT INTO Person.StateProvince (StateProvinceCode, CountryRegionCode, Name, TerritoryId)
   VALUES ('CO', AR', 'Córdoba', 1)

INSERT INTO Person.StateProvince (StateProvinceCode, CountryRegionCode, Name, TerritoryId)
VALUES ('MZ', AR', 'Mendoza', 1)

3. SELECT @@identity

   o

   SELECT SCOPE_IDENTITY()

4. SELECT * FROM Person.StateProvince
   [WHERE countryregioncode='AR'] -- Optativo

5. Código del archivo ProvinciasArgentinas.sql

```
USE AdventureWorks
GO

CREATE TABLE [dbo].[ProvinciasArgentinas](
[CodigoProvincia] [char](2) COLLATE Modern_Spanish_CI_AS NOT NULL,
[CodigoPais] [char](2) COLLATE Modern_Spanish_CI_AS NOT NULL,
[Nombre] [varchar](50) COLLATE Modern_Spanish_CI_AS NOT NULL,
[Territorio] [smallint] NOT NULL,
 CONSTRAINT [PK_ProvinciasArgentinas] PRIMARY KEY CLUSTERED
(
[CodigoProvincia] ASC
)WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

INSERT INTO ProvinciasArgentinas (CodigoProvincia, CodigoPais, Nombre, Territorio)
VALUES ('SF','AR','Santa Fe',1)
INSERT INTO ProvinciasArgentinas (CodigoProvincia, CodigoPais, Nombre, Territorio)
VALUES ('MI','AR','Misiones',1)
INSERT INTO ProvinciasArgentinas (CodigoProvincia, CodigoPais, Nombre, Territorio)
VALUES ('ER','AR','Entre Rios',1)
```

6. INSERT INTO Person.StateProvince (StateProvinceCode, CountryRegionCode, Name, TerritoryId)
   SELECT CodigoProvincia, CodigoPais, Nombre, Territorio
   FROM ProvinciasArgentinas

7. SELECT * FROM Person.StateProvince
   [WHERE countryregioncode='AR'] -- Optativo

Ejercicio 3
Escribiendo Sentencias UPDATE

En este ejercicio escribirá varias sentencias UPDATE para modificar información en tablas de la base de datos **AdventureWorks**.

1. Escribir y ejecutar una sentencia UPDATE que modifique algunos valores de la tabla Sales.SalesReason. Modifique el campo Name como “Precio” cuando SalesReasonId = 1, modifique el campo Name como “En Promoción” y el campo ModifiedDate a la fecha de hoy cuando SalesReasonId = 2

2. Verificar que los valores fueron modificados en la tabla

3. Escribir y ejecutar una sentencia UPDATE que modifique el campo TerritoryId de la tabla Person.StateProvince por el valor 8 para todas las provincias que pertenecen al país cuyo código es ‘AR’.

4. Verificar que los valores fueron modificados en la tabla

5. Escribir y ejecutar una sentencia UPDATE similar a la anterior pero que el valor para TerritoryId sea 5. Trabaje como filtro el nombre del país ‘Argentina’ sin conocer cual es su código para generar la condición de modificación.

6. Verificar que los valores fueron modificados en la tabla

Respuestas

1.
```
UPDATE Sales.SalesReason SET Name='Precio'
WHERE SalesReasonId = 1

UPDATE Sales.SalesReason SET Name='En Promoción', ModifiedDate=GetDate()
WHERE SalesReasonId = 2
```

2.
```
SELECT * FROM Sales.SalesReason
```

3.
```
UPDATE Person.StateProvince SET TerritoryId=8
WHERE CountryregionCode='AR'
```

4.
```
SELECT * FROM Person.StateProvince
[WHERE countryregioncode='AR']        -- Optativo
```

5.
```
UPDATE Person.StateProvince SET TerritoryId = 5
FROM Person.StateProvince p INNER JOIN Person.CountryRegion c
ON p.CountryRegionCode = c.CountryRegionCode
WHERE c.Name='Argentina'
```

6.
```
SELECT * FROM Person.StateProvince
[WHERE countryregioncode='AR']        -- Optativo
```

Ejercicio 4
Escribiendo Sentencias DELETE

En este ejercicio escribirá varias sentencias DELETE para eliminar información en tablas de la base de datos **AdventureWorks**.

1. Escribir y ejecutar una sentencia DELETE que elimine todos los registros de la tabla ProvinciasArgentinas creada en el Ejercicio 2.

2. Verificar que los valores fueron eliminados de la tabla

3. Escribir y ejecutar una sentencia DELETE que elimine el registro correspondiente a la provincia de Buenos Aires (Código 'BA' el país 'AR') de la tabla Person.StateProvince.

4. Verificar que los valores fueron eliminados de la tabla

5. Escribir y ejecutar una sentencia DELETE que elimine todos los registros pertenecientes a Argentina de la tabla Person.StateProvince. Trabaje como filtro el nombre del país 'Argentina' sin conocer cual es su código para generar la condición de modificación.

6. Verificar que los valores fueron eliminados de la tabla

Respuestas

1. DELETE FROM ProvinciasArgentinas

2. SELECT * FROM ProvinciasArgentinas

3. DELETE FROM Person.StateProvince
   WHERE CountryRegionCode = 'AR' and StateProvinceCode = 'BA'

4. SELECT * FROM Person.StateProvince
   [WHERE countryregioncode='AR'] -- Optativo

5. DELETE FROM Person.StateProvince
   FROM Person.StateProvince p INNER JOIN Person.CountryRegion c
   ON p.CountryRegionCode = c.CountryRegionCode
   WHERE c.Name='Argentina'

6. SELECT * FROM Person.StateProvince
   [WHERE countryregioncode='AR'] -- Optativo

Módulo 4: Creación de Bases de Datos

Ejercicio 1
Creando una base de datos

En este ejercicio creará una base de datos y configurará su opciones.

1. Escribir y ejecutar una sentencia SQL que cree la siguiente base de datos:

Parámetro	Valor
Nombre de la base de datos	Prueba
Nombre lógico del archivo de datos	Prueba_Datos
Nombre físico del archivo de datos	Prueba.mdf
Tamaño Inicial del archivo de datos	20MB
Tamaño Máximo del archivos de datos	50MB
Incremento de Crecimiento del archivo de datos	3MB
Nombre lógico del archivo de registro	Prueba_Log
Nombre físico del archivo de registro	Prueba.ldf
Tamaño Inicial del archivo de registro	5MB
Tamaño Máximo del archivos de registro	15MB
Incremento de Crecimiento del archivo de registro	10%

2. Ejecutar el procedimiento almacenado ***sp_helpdb*** para ver la información de esta nueva base de datos.

3. Escribir y ejecutar una sentencia para eliminar la base de datos Prueba recién creada.

4. Generar nuevamente la base de datos pero esta vez usando las herramientas gráficas del SQL Server Management Studio.

5. Escribir y ejecutar una sentencia SQL que modifique el valor del Tamaño Máximo del archivo de datos y del registro de transacciones para que ambos sean irrestrictos.

6. Escribir y ejecutar sentencias SQL para Modificar las siguientes opciones de la base de datos Prueba: ***Recovery Model*** = Simple, ***Ansi Warnings***=ON.

7. Verificar los cambios usando el SQL Server Management Studio.

8. Ejecutar las siguientes vistas de catálogo para las bases de datos Prueba y AdventureWorks y analizar sus resultados:
   - sys.databases
   - sys.database_files
   - sys.tables

9. Ejecutar los siguientes procedimientos almacenados para las bases de datos Prueba y AdventureWorks y analizar sus resultados
   - sp_databases
   - sp_helpdb

Respuestas

1.
```
CREATE DATABASE [Prueba] ON  PRIMARY
(NAME = N'Prueba_Datos',
FILENAME = N'C:\Datos\Prueba.mdf',
SIZE = 20MB,
MAXSIZE = 50MB,
FILEGROWTH = 3MB)
 LOG ON
( NAME = N'Prueba_log',
FILENAME = N'C:\Datos\Prueba_log.ldf',
SIZE = 5MB,
MAXSIZE = 15MB,
FILEGROWTH = 10%)
GO
```

2.
```
EXEC sp_helpdb

EXEC sp_helpdb 'prueba'
```

3.
```
DROP DATABASE prueba
```

4. En el Explorador de Objetos del SQL Server Management Studio sobre la opción Base de Datos presionar el botón derecho del mouse y seleccionar Nueva Base de Datos. Ingresar los datos especificados sobre la solapa General y aceptar.

5.
```
ALTER DATABASE prueba
MODIFY FILE (Name='Prueba_Datos', MaxSize=UNLIMITED)
```

6.
```
ALTER DATABASE Prueba
SET RECOVERY Simple

ALTER DATABASE Prueba
SET ANSI_Warnings ON

EXEC sp_dboption 'Prueba', 'ANSI Warnings', 'TRUE'
```

7. En el Explorador de Objetos seleccionar el nombre de la base de datos, presionar el botón derecho del Mouse, elegir Propiedades. Solapa Opciones (Options).

8.
```
SELECT * FROM sys.databases

SELECT * FROM sys.databases WHERE Name='Prueba'

SELECT * FROM sys.databases WHERE Name='AdventureWorks'

SELECT * FROM sys.database_files

SELECT * FROM sys.Tables
```

9.
```
EXEC sp_helpdb
```

EXEC sp_helpdb 'Prueba'

EXEC sp_helpdb 'AdventureWorks'

Ejercicio 2
Agregando Grupos de Archivos

En este ejercicio se agregarán grupos de archivos a la base de datos **Prueba** creada en el ejercicio anterior.

1. Escribir y ejecutar una sentencia SQL que agregue un nuevo grupo de archivos llamado GrupoPrueba a la base de datos Prueba.

2. Verificar la creación del grupo de archivos usando el SQL Server Management Studio.

3. Agregar dos grupos de archivos más llamados GrupoPrueba2 y GrupoPrueba3 usando el SQL Server Management Studio.

4. Crear tres nuevos archivos asociados a cada uno de estos nuevos grupos de archivos. Crear el primero llamado Prueba1 asociado al grupo GrupoPrueba usando el SQL Server Management Studio. Los otros dos llamados Prueba2 y Prueba3 asociados a los grupos de archivos GrupoPrueba2 y GrupoPrueba3 respectivamente usando sentencias SQL.

5. Verificar la creación de estos archivos usando el SQL Server Management Studio.

6. Verificar la creación de estos archivos ejecutando el procedimiento almacenado ***sp_helpdb***.

Respuestas

1.
```
ALTER DATABASE Prueba
ADD FILEGROUP GrupoPrueba
```

2. En el Explorador de Objetos seleccionar el nombre de la base de datos, presionar el botón derecho del Mouse, elegir Propiedades. Dentro de Propiedades seleccionar la solapa Grupos de Archivos (FileGroups).

3. En la ventana de Propiedades abierta en el punto 2 presionar el botón Agregar (Add) e ingresar los datos de los grupos de archivos. Solo es necesario ingresar el nombre del mismo.

4. En el Explorador de Objetos seleccionar el nombre de la base de datos, presionar el botón derecho del Mouse, elegir Propiedades. Dentro de Propiedades seleccionar la solapa Archivos (Files). Presionar el botón Agregar (Add) e ingresar los datos de los archivos: Nombre, Grupo de Archivos y Ubicación.

```
ALTER DATABASE Prueba
ADD     FILE
(NAME = Prueba2,
FILENAME = N'C:\Datos\Prueba2.ndf')
TO FILEGROUP GrupoPrueba2

ALTER DATABASE Prueba
ADD     FILE
```

```
(NAME = Prueba3,
FILENAME = N'C:\Datos\Prueba3.ndf')
TO FILEGROUP GrupoPrueba3
```

5. En el Explorador de Objetos seleccionar el nombre de la base de datos, presionar el botón derecho del Mouse, elegir Propiedades. Dentro de Propiedades seleccionar la solapa Archivos (Files).

6. EXEC sp_helpdb 'Prueba'

Ejercicio 3
Creando Esquemas

En este ejercicio se agregarán esquemas a la base de datos **Prueba** creada en el ejercicio anterior.

1. Escribir y ejecutar una sentencia SQL que agregue un nuevo esquema llamado Compras a la base de datos Prueba.

2. Verificar la creación de este esquema usando el SQL Server Management Studio.

3. Crear otro esquema llamado Ventas usando el SQL Server Management Studio

4. Verificar su creación usando la vista de catálogo ***sys.schemas***

Respuestas

1. CREATE SCHEMA Compras

2. En el Explorador de Objetos expandir la solapa Seguridad (Security) de la base de datos. Dentro de Seguridad expandir la solapa Esquemas (Schemas)

3. En el Explorador de Objetos expandir la solapa Seguridad (Security) de la base de datos. Dentro de Seguridad expandir la solapa Esquemas (Schemas). Para agregar un nuevo esquema presionar el botón derecho del mouse y seleccionar Nuevo Esquema (New schemas)

4. SELECT * FROM sys.schemas

Ejercicio 4
Creando Instantáneas de Bases de Datos

En este ejercicio se creará una instantánea de datos para la base de datos **AdventureWorks**.

1. Escribir y ejecutar una sentencia SQL que cree una instantánea de datos de la base de datos AdventureWorks llamada SSAdvWorks

2. Verificar que la instantánea de datos se haya creado usando el SQL Server Management Studio.

3. Agregar un nuevo registro en la tabla Person.StateProvince de la base de datos AdventureWorks o cualquier otra modificación sobre los datos. Puede usar algunas de las sentencias del Laboratorio 2. Verificar ambas bases de datos. Que paso?

Respuestas

1. CREATE DATABASE SSAdvWorks ON
(NAME= AdventureWorks_Data, FILENAME = 'C:\Datos\SSAdvWorks.ss')
AS SNAPSHOT OF AdventureWorks

2. En el Explorador de Objetos expandir la solapa Instantáneas de Base de Datos (Database Snapshots)

3. El registro aparece sobre la base de datos AdventureWorks, pero no aparece en SSAdvWorks. Para verificar las cantidad de registros buscar la tabla en el Explorador de Objetos, solapa Tablas (Tables) de cada base de datos, presionar el botón derecho del mouse, elegir Propiedades. En la solapa General tiene la información sobre Cantidad de Registros (Row Count).

Módulo 5: Creación de Tipos de Datos y Tablas

Ejercicio 1
Creando Tipos de Datos

En este ejercicio creará nuevos tipos de datos para la base de datos **AdventureWorks**.

1. Crear un nuevo tipo de datos llamado CodigoPais de tipo char(2) que no permite nulos usando el SQL Server Management.

2. Escribir y ejecutar una sentencia SQL que cree un nuevo tipo de datos llamado DireccionMail de tipo varchar(50) que permite nulos

3. Verificar que ambos tipos de datos se hayan creado. Puede usar la vista de catálogo **sys.types** para esta verificación

Respuestas

1. En el Explorador de Objetos expandir la solapa Programación (Programmability) y dentro de Programación la solapa Tipos (Types). Presionar el botón derecho del mouse sobre la opción Tipos de Datos de Usuario (User-defined Data Types) y seleccionar nuevo tipo de dato. Ingresar los datos en la solapa General.

2. CREATE TYPE dbo.DireccionMail
   FROM varchar(50) NULL

3. SELECT * FROM sys.types

Ejercicio 2
Creando Tablas

En este ejercicio creará nuevas tablas para la base de datos **Prueba**.

1. Escribir y ejecutar una sentencia SQL que cree una tabla llamada Categorías que contenga las siguientes columnas:

Columna	Tipo de Dato	Observaciones
CodigoCategoria	Smallint	Identity – No soporta nulos
Descripcion	Varchar(50)	No soporta Nulos
Sueldo	Decimal(18,2)	No soporta Nulos
FhUltimaModif	smalldatetime	

2. Crear otra tabla llamada Empleados usando el SQL Server Management Studio. La tabla debe contener la siguiente estructura:

Columna	Tipo de Dato	Observaciones
Legajo	int	Identity – No soporta nulos
Nombres	Varchar(50)	No soporta Nulos
Apellidos	Varchar(50)	No soporta Nulos
NombreCompleto		Calculada: Apellido, Nombres
FhIngresoEmpresa	smalldatetime	
FhNacimiento	Smalldatetime	No soporta Nulos
Categoria	smallint	No soporta Nulos
CUIL	Char(13)	No soporta Nulos

3. Verificar que ambas tablas hayan sido creadas. Puede usar la vista de catálogo ***sys.tables*** o el Explorador de Objetos del SQL Server Management Studio.

Respuestas

1.
```
CREATE TABLE dbo.Categorias(
        CodigoCategoria smallint IDENTITY(1,1) NOT NULL,
        Descripcion varchar(50) NOT NULL,
        Sueldo decimal(18,2) NOT NULL,
        FhUltimaModif smalldatetime NULL
CONSTRAINT PK_Categorias PRIMARY KEY CLUSTERED (CodigoCategoria))
```

2. En el Explorador de Objetos sobre la solapa Tablas (Tables) de la base de datos presionar el botón derecho del Mouse, elegir Nueva tabla. Ingresar los datos.

3. SELECT * FROM sys.tables

Ejercicio 3
Creando Tablas con Particiones

En este ejercicio creará una nuevas tablas con particiones para la base de datos **Prueba**.

1. Generar una función de partición sobre un tipo de dato fecha de la siguiente manera:
   - Una partición para las fechas anteriores del 1/1/2002
   - Una partición para las fechas comprendidas entre el 1/1/2002 y el 31/12/2002
   - Una partición para las fechas comprendidas entre el 1/1/2003 y el 31/12/2003
   - Una partición para las fechas posteriores al 1/1/2004

2. Generar un esquema para la función del punto anterior de manera que las dos primeras particiones se asocien al grupo de archivos llamado GrupoPrueba creado en el laboratorio 4 y las dos últimas con los grupos de archivos llamados GrupoPrueba2 y GrupoPrueba3 (también creados en el laboratorio 4) respectivamente.

3. Verificar que ambas hayan sido creadas. Puede usar el Explorador de Objetos del SQL Server Management Studio o las vistas de catálogo ***sys.partition_functions*** y ***sys.partition_schemes***

4. Levantar el archivo <carpeta de instalación>\Modulo5***TablaVentas.sql*** dentro del SQL Server Management Studio. Este procedimiento crea una nueva tabla llamada SalesOrderHeader en la base de datos Prueba con el mismo formato que dicha tabla de la base de datos AdventureWorks. No ejecutar la sentencia.

5. Modificar el código del CREATE TABLE levantado en el punto 3 de manera que la creación sea la de una tabla con particiones asociado al esquema creado en el punto 2. La columna para la partición es OrderDate. Ejecutar la instrucción para crear la tabla

6. Verificar que la tabla haya sido creada con particiones usando la vista de catálogo ***sys.partitions***. Verificar también el tamaño de los archivos asociados a cada partición del esquema en el Explorador de Windows.

7. Levantar el archivo <carpeta de instalación>\Modulo5***CargarDatos.sql*** dentro del SQL Server Management Studio. Este procedimiento carga los datos en la tabla SalesOrderHeader de la base de datos Prueba tomando los datos de la misma tabla en la base de datos AdventureWorks. Ejecutar el código.

8. Verificar la distribución de los datos en cada partición usando la vista de catálogo ***sys.partitions***. Verificar también el tamaño de los archivos asociados a cada partición del esquema en el Explorador de Windows.

Respuestas

1.
```
CREATE PARTITION FUNCTION pf_FechaOrden (datetime)
AS RANGE RIGHT
FOR VALUES ('01/01/2000', '01/01/2003', '01/01/2004')
```

2.
```
CREATE PARTITION SCHEME ps_FechaOrden
AS PARTITION pf_FechaOrden
TO (GrupoPrueba, GrupoPrueba, GrupoPrueba2, GrupoPrueba3)
```

3. SELECT * FROM sys.partition_functions

   SELECT * FROM sys.partition_functions

   En el Explorador de Objetos expandir la solapa Almacenamiento (Storage) de la base de datos. Dentro tiene las opciones para verificar las funciones y esquemas de partición.

5. Código del archivo TablaVentas.sql. En negrita las modificaciones necesarias para crear la tabla con particiones.

```
USE [Prueba]
CREATE TABLE dbo.SalesOrderHeader(
SalesOrderID int IDENTITY(1,1) NOT NULL,
RevisionNumber tinyint NOT NULL ,
OrderDate datetime NOT NULL ,
DueDate datetime NOT NULL,
ShipDate datetime NULL,
Status tinyint NULL ,
OnlineOrderFlag bit NULL ,
SalesOrderNumber  varchar(50) NULL,
PurchaseOrderNumber varchar(25) NULL,
AccountNumber varchar(15) NULL,
CustomerID int NULL,
ContactID int NULL,
SalesPersonID int NULL,
TerritoryID int NULL,
BillToAddressID int NULL,
ShipToAddressID int NULL,
ShipMethodID int NULL,
CreditCardID int NULL,
CreditCardApprovalCode varchar(15),
CurrencyRateID int NULL,
SubTotal money NULL ,
TaxAmt money NULL ,
Freight money NULL,
Comment nvarchar(128) NULL,
rowguid uniqueidentifier ROWGUIDCOL NULL DEFAULT (newid()),
ModifiedDate datetime NULL ,
CONSTRAINT [PK_SalesOrderHeader_SalesOrderID] PRIMARY KEY CLUSTERED
(SalesOrderID, OrderDate) WITH (IGNORE_DUP_KEY = OFF)
ON ps_FechaOrden(OrderDate))
```

6. SELECT * FROM sys.partitions WHERE object_id=object_id('SalesOrderHeader')

7. Código del archivo CargarDatos.sql

```
INSERT INTO SalesOrderHeader ([RevisionNumber], [OrderDate], [DueDate],
[ShipDate], [Status], [OnlineOrderFlag], [SalesOrderNumber], [PurchaseOrderNumber],
[AccountNumber], [CustomerID], [ContactID], [SalesPersonID], [TerritoryID],
[BillToAddressID], [ShipToAddressID], [CreditCardID], [CreditCardApprovalCode],
[CurrencyRateID], [SubTotal], [TaxAmt], [Freight], [Comment], [ModifiedDate])
SELECT [RevisionNumber], [OrderDate], [DueDate], [ShipDate],
[Status], [OnlineOrderFlag], [SalesOrderNumber], [PurchaseOrderNumber],
[AccountNumber], [CustomerID], [ContactID], [SalesPersonID], [TerritoryID],
```

[BillToAddressID], [ShipToAddressID], [CreditCardID], [CreditCardApprovalCode], [CurrencyRateID], [SubTotal], [TaxAmt], [Freight], [Comment], [ModifiedDate] FROM AdventureWorks.Sales.SalesOrderHeader

8. SELECT * FROM sys.partitions WHERE object_id=object_id('SalesOrderHeader')

Módulo 6: Usando XML

Ejercicio 1
Creando sentencias SQL con la cláusula FOR XML

En este ejercicio creará sentencias SQL que devuelven datos en formato XML con información de la base de datos **AdventureWorks**.

1. Escribir y ejecutar una sentencia SELECT que devuelva las columnas CustomerId y CustomerType de la tabla Sales.Customer y la columna SalesOrderId de la tabla Sales.SalesOrderHeader de manera de mostrar todas las órdenes de venta con los datos del cliente asociado. Las columnas deben mostrarse con los nombres Codigo, TipoCliente y NumOrden respectivamente.

2. Modificar la sentencia del punto 1 para que devuelva los datos en formato XML de manera que aparezcan con un elemento llamado <row> para cada registro y la columnas en forma de atributo. Cambiar la forma de mostrar el resultado a Texto.

3. Modificar la sentencia del punto 2 de manera que los campos se muestren como elementos y el XML generado en vez de llamar <row> a cada registro lo llame <Orden>. El documento deberá estar bien formado.

4. Escribir y ejecutar una sentencia SELECT que devuelva las columnas ProductModelId y Name de la tabla Production.ProductModel para todos los registros cuyo nombre comienza con 'Road'. El conjunto de registros debe estar en formato XML, con un elemento por registro cuyo nombre sea el nombre de la tabla y las columnas en forma de elementos.

5. Escribir y ejecutar una sentencia SELECT que devuelva el siguiente documento XML:

```
<Clientes Codigo="1" Tipo="S">
        <Ordenes NumOrden="43860" Total="13216.0537"/>
        <Ordenes NumOrden="44501" Total="23646.0339"/>
        <Ordenes NumOrden="45283" Total="34066.1881"/>
        <Ordenes NumOrden="46042" Total="31423.5209"/>
</Clientes>
```

   Los datos de Clientes están tomados de la tabla Sales.Customer y son los campos Customerid y CustomertType renombradas como Codigo y Tipo respectivamente. Los datos de las órdenes están tomados de la tabla Sales.SalesOrderHeader y son las columnas SalesOrderId y SubTotal renombradas como NumOrden y Total respectivamente. Tomar solo los datos pertenecientes al cliente 1. Usar la cláusula FOR XML EXPLICIT.

6. Escribir y ejecutar una sentencia SELECT que devuelva un documento XML igual o similar al del punto 5 usando la cláusula FOR XML PATH

Respuestas

1.
```
SELECT c.CustomerID as Codigo, c.CustomerType as TipoCliente,
o.SalesOrderID as NumOrden
FROM Sales.Customer c INNER JOIN Sales.SalesOrderHeader o
```

```
ON c.CustomerID = o.CustomerID
ORDER BY c.CustomerID
```

2.
```
SELECT c.CustomerID as Codigo, c.CustomerType as TipoCliente,
o.SalesOrderID as NumOrden
FROM Sales.Customer c INNER JOIN Sales.SalesOrderHeader o
ON c.CustomerID = o.CustomerID
ORDER BY c.CustomerID
FOR XML RAW
```

3.
```
SELECT c.CustomerID as Codigo, c.CustomerType as TipoCliente,
o.SalesOrderID as NumOrden
FROM Sales.Customer c INNER JOIN Sales.SalesOrderHeader o
ON c.CustomerID = o.CustomerID
ORDER BY c.CustomerID
FOR XML RAW('Orden'), Elements, ROOT('OrdenesVentas')
```

4.
```
SELECT ProductModelID, Name
FROM Production.ProductModel
WHERE Name like 'Road%'
FOR XML AUTO, Elements;
```

5.
```
SELECT 1       as Tag,
NULL           as Parent,
CustomerID     as [Clientes!1!Codigo],
CustomerType as [Clientes!1!Tipo],
NULL           as [Ordenes!2!NumOrden],
NULL           as [Ordenes!2!Total]
FROM Sales.Customer as Clientes
WHERE Clientes.CustomerID = 1
UNION
SELECT 2 as tag,
        1 as parent,
        c.CustomerID,
        c.CustomerType,
        o.SalesOrderID,
        o.SubTotal
FROM Sales.Customer c INNER JOIN Sales.SalesOrderHeader o
ON c.CustomerID = o.CustomerID
WHERE c.CustomerID = 1
FOR XML EXPLICIT
```

6.
```
SELECT c.Customerid "@Codigo", c.CustomerType "@Tipo",
o.SalesOrderID "Ordenes/NumOrden",
o.SubTotal "Ordenes/Total"
FROM Sales.Customer c INNER JOIN Sales.SalesOrderHeader o
ON c.CustomerID = o.CustomerID
WHERE c.CustomerID = 1
FOR XML PATH('Clientes')

SELECT CustomerID as "@Codigo", CustomerType as "@Tipo",
        (SELECT SalesOrderID as "@NumOrden", SubTotal as "@Total"
        FROM Sales.SalesOrderHeader o
        WHERE o.CustomerID = c.CustomerID
```

```
        FOR XML PATH('Ordenes'), TYPE)
FROM Sales.Customer c
WHERE c.CustomerId=1
FOR XML PATH('Clientes')
```

Ejercicio 2
Creando sentencias SQL con OPENXML

En este ejercicio creará sentencias SQL que devuelven conjuntos de registros tomando información de un documento XML. Trabajar con la base de datos **AdventureWorks**

1. Levantar el archivo <carpeta de instalación>\Modulo6***Paises.xml***. Usar este documento xml para generar un conjunto de registros usando como base la tabla Person.CountryRegion de la base de datos AdventureWorks.

2. Modificar el ejemplo anterior cambiando el nombre de la tabla por una descripción de campos personalizada.

Respuestas:

1.
```
DECLARE @Doc varchar(max)
SET @Doc='<?xml version="1.0" standalone="yes"?>
<Paises>
<xs:schema id="Paises" xmlns=""
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-
microsoft-com:xml-msdata">
<xs:element name="Paises" msdata:IsDataSet="true"
msdata:MainDataTable="Pais" msdata:UseCurrentLocale="true">
<xs:complexType>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="Pais">
                <xs:complexType>
                        <xs:sequence>
                        <xs:element name="CountryRegionCode" type="xs:string"
                        minOccurs="0" />
                        <xs:element name="Name" type="xs:string" minOccurs="0" />
                        <xs:element name="ModifiedDate" type="xs:dateTime"
                        minOccurs="0" />
                        </xs:sequence>
                </xs:complexType>
                </xs:element>
        </xs:choice>
</xs:complexType>
</xs:element>
</xs:schema>
<Pais>
        <CountryRegionCode>EC</CountryRegionCode>
        <Name>Ecuador</Name>
        <ModifiedDate>01-06-2008</ModifiedDate>
</Pais>
<Pais>
        <CountryRegionCode>EE</CountryRegionCode>
        <Name>Estonia</Name>
        <ModifiedDate>01-06-2008</ModifiedDate>
</Pais>
<Pais>
```

```
        <CountryRegionCode>EG</CountryRegionCode>
        <Name>Egypt</Name>
        <ModifiedDate>01-06-2008</ModifiedDate>
</Pais>
<Pais>
        <CountryRegionCode>ER</CountryRegionCode>
        <Name>Eritrea</Name>
        <ModifiedDate>01-06-2008</ModifiedDate>
</Pais>
<Pais>
        <CountryRegionCode>ES</CountryRegionCode>
        <Name>Spain</Name>
        <ModifiedDate>01-06-2008</ModifiedDate>
</Pais>
<Pais>
        <CountryRegionCode>ET</CountryRegionCode>
        <Name>Ethiopia</Name>
        <ModifiedDate>01-06-2008</ModifiedDate>
</Pais>
</Paises>'

DECLARE @hdoc integer
EXEC sp_xml_preparedocument @hdoc OUTPUT, @doc

SELECT * FROM OPENXML(@hdoc, '/Paises/Pais', 2)
WITH Person.CountryRegion
```

2. Ídem Anterior, cambiar las últimas 2 líneas por alguna de estas dos opciones:

```
SELECT * FROM OPENXML(@hdoc, '/Paises/Pais', 2)
WITH (CountryRegionCode char(2), Name varchar(50), ModifiedDate datetime)

SELECT * FROM OPENXML(@hdoc, '/Paises/Pais', 2)
WITH (Codigo char(2) 'CountryRegionCode',
Nombre varchar(50) 'Name',
Fecha datetime 'ModifiedDate')
```

Ejercicio 3
Usando el tipo de dato XML

En este ejercicio usará el tipo de dato xml con tablas dentro de la base de datos **Prueba**.

1. Escribir y ejecutar una sentencia SQL que cree una tabla llamada Documentos que contenga las siguientes columnas. Puede usar el SQL Server Management Studio.

Columna	Tipo de Dato	Observaciones
CodigoDocumento	Int	Identity – No soporta nulos
Documento	Xml	No soporta Nulos

2. Levantar el archivo <carpeta de instalación>\Modulo6***Datos.xml***. Escribir y ejecutar tres sentencias SQL de tipo INSERT para agregar tres registros en la tabla Documentos con los xml contenidos en dicho archivo.

3. Verificar que los datos hayan sigo ingresados.

4. Escribir y ejecutar una sentencia SQL aplicando las sentencias FLOWR para que muestre la lista de países (solo el nombre) del primer registro de la tabla creada en el punto 1.

5. Escribir y ejecutar una sentencias SQL que muestre el tercer país que figura en el documento xml del primer registro de la misma tabla.

6. Escribir y ejecutar una sentencia SQL que verifique si el código de provincia “BA” existe en el segundo registro de la tabla creada en el punto 1.

7. Modificar la sentencia del punto 6 de manera que el código “BA” este contenido en una variable y usar la variable para controlar la existencia de ese código.

8. Escribir y ejecutar una sentencia SQL que devuelva un conjunto de registros basados en los datos xml del primer registro de la tabla Documentos

9. Escribir y ejecutar las mismas sentencias de los puntos 4, 5, 6 y 8 para el registro 3 de la tabla Documentos. **Nota**: dicho xml usa atributos y no elementos.

Respuestas:

1.
```
CREATE TABLE [dbo].[Documentos]
(CodigoDocumento int IDENTITY(1,1) NOT NULL,
Documento xml NOT NULL)
```

2.
```
INSERT INTO Documentos (Documento)
VALUES
('<Paises><Pais><Codigo>AR</Codigo><Nombre>Argentina</Nombre></Pais><Pais>
<Codigo>UR</Codigo>
<Nombre>Uruguay</Nombre></Pais><Pais><Codigo>PA</Codigo><Nombre>Paragua
y</Nombre></Pais><Pais>
```

<Codigo>CH</Codigo><Nombre>Chile</Nombre></Pais></Paises>')

INSERT INTO Documentos (Documento)
VALUES ('<Provincias><Provincia><Pais>AR</Pais><Codigo>BA</Codigo>
<Nombre>Buenos
Aires</Nombre></Provincia><Provincia><Pais>AR</Pais><Codigo>ME</Codigo>
<Nombre>Mendoza</Nombre></Provincia><Provincia><Pais>CH</Pais><Codigo>VL<
/Codigo><Nombre>Valparaiso</Nombre>
</Provincia><Provincia><Pais>CH</Pais><Codigo>SA</Codigo><Nombre>Santiago</
Nombre></Provincia></Provincias>')

INSERT INTO Documentos (Documento)
VALUES ('<Paises><Pais Codigo="AR" Nombre="Argentina" /><Pais Codigo="UR" Nombre="Uruguay" /><Pais Codigo="PA" Nombre="Paraguay" /><Pais Codigo="CH" Nombre="Chile" /></Paises>')

3. SELECT * FROM Documentos

4. SELECT documento.query('for $Nombre in /Paises/Pais/Nombre
return string($Nombre)')
FROM Documentos WHERE CodigoDocumento=1

5. SELECT Documento.value('(/Paises/Pais/Nombre)[3]', 'char(50)')

FROM Documentos WHERE CodigoDocumento=1

6. DECLARE @x xml
DECLARE @f bit
SELECT @x=Documento FROM Documentos WHERE CodigoDocumento=2
SET @f = @x.exist('/Provincias/Provincia[(Codigo="BA")]')
SELECT @f

7. DECLARE @Codigo char(2)
SET @Codigo='BA'
DECLARE @x xml
DECLARE @f bit
SELECT @x=Documento FROM Documentos WHERE CodigoDocumento=8
SET @f = @x.exist('/Provincias/Provincia[(Codigo=sql:variable("@Codigo"))]')
SELECT @f

8. DECLARE @xml xml
SELECT @xml=Documento FROM Documentos WHERE CodigoDocumento=1
SELECT miCol.value('Codigo[1]','char(2)') as Codigo,
miCol.value('Nombre[1]','varchar(50)') as Nombre
FROM @xml.nodes('/Paises/Pais') AS miTabla(miCol)

9. SELECT documento.query('for $Nombre in /Paises/Pais/@Nombre
return string($Nombre)')
FROM Documentos WHERE CodigoDocumento=3

SELECT Documento.value('(/Paises/Pais/@Nombre)[3]', 'char(50)')
FROM Documentos WHERE CodigoDocumento=3

```
DECLARE @x xml
DECLARE @f bit
SELECT @x=Documento FROM Documentos WHERE CodigoDocumento=3
SET @f = @x.exist('/Paises/Pais[(@Codigo="CH")]')
SELECT @f

DECLARE @xml xml
SELECT @xml=Documento FROM Documentos WHERE CodigoDocumento=9
SELECT miCol.value('@Codigo','char(2)') as Codigo,
miCol.value('@Nombre','varchar(50)') as Nombre
FROM @xml.nodes('/Paises/Pais') AS miTabla(miCol)
```

Módulo 7: Índices

Ejercicio 1
Creando índices

En este ejercicio creará índices para las tablas de la base de datos **AdventureWorks**.

1. Escribir y ejecutar una sentencia SQL que cree un índice no agrupado llamado IX_ContactID sobre el campo ContactID en la tabla Sales.SalesOrderHeader.

2. Verificar que el índice haya creado y revisar los otros índices de esta tabla usando el SQL Server Management Studio.

3. Verificar los índices de esa tabla usando la vista de catálogo ***sys.indexes***

4. Crear un nuevo índice no agrupado llamado IX_SpecialOffer sobre el campo SpecialOfferID en la tabla Sales.SalesOrderDetails. Use el SQL Server Management Studio para la creación de este índice

5. Escribir y ejecutar una sentencia SQL que cree un nuevo índice no agrupado llamado IX_Contact_NombreCompleto sobre los campos LastName, FirstName en la tabla Person.Contact incluyendo los campos Title, MiddleName y Suffix.

6. Escribir y ejecutar una sentencia SQL que cree un nuevo índice llamado IX_Descripcion sobre el campo Description en la tabla Production.ProductDescription dejando las páginas libres en un 40%. El índice deberá poder crearse on line.

Respuestas

1.
```
CREATE NONCLUSTERED INDEX IX_ContactID
ON Sales.SalesOrderHeader (ContactID)
```

2. En el Explorador de Objetos expandir la tabla Sales.SalesOrderHeader. Expandir la solapa Índices (indexes).

3.
```
SELECT * FROM sys.indexes

SELECT * FROM sys.indexes WHERE object_Id=object_id('Sales.SalesOrderHeader')
```

4. En el Explorador de Objetos seleccionar la tabla Sales.SalesOrderDetails, presionar el botón derecho de mouse y elegir Modificar (Modify). Dentro de la modificación de la tabla volver a presionar el botón derecho sobre la parte superior y elegir Índices/Claves. Dentro de esta opción presionar el botón Agregar (Add) para crear un nuevo índice. Ingresar los datos necesarios.

5.
```
CREATE NONCLUSTERED INDEX IX_Contact_NombreCompleto
ON Person.Contact (LastName ASC,FirstName ASC)
INCLUDE (Title,MiddleName,Suffix)
```

6.
```
CREATE NONCLUSTERED INDEX IX_Descripcion
ON Production.ProductDescription (Description)
WITH (FILLFACTOR = 60, ONLINE = ON)
```

Ejercicio 2
Optimizando índices

En este ejercicio buscará información sobre índices y ejecutará tareas que permitan mejorar el funcionamiento de los mismo. Se usa la base de datos **AdventureWorks**.

1. Escribir y ejecutar una sentencia SQL que permita analizar la desfragmentación de los índices de la tabla Sales.SalesOrderHeader. Genere una unión con ***sys.indexes*** para encontrar el nombre de los mismos.

2. Escribir y ejecutar una sentencia SQL que permita reorganizar el índice IX_SalesOrderDetail_ProductID de la tabla Sales.SalesOrderHeader.

3. Analizar el resultado obtenido sobre la fragmentación

4. Escribir y ejecutar una sentencia SQL que permita reconstruir el mismo índice que en el punto 2.

5. Analizar el resultado obtenido sobre la fragmentación. Comparar los resultados con los obtenidos después de la reorganización

6. Utilizar el Asistente para la optimización de motor de base de datos de Microsoft SQL Server (Database Engine Tunning Advisor) para evaluar la carga de trabajo del archivo <carpeta de instalación>\Modulo7***CargaTrabajo.xml***. Analizar los resultados.

Respuestas

1.
```
SELECT a.index_id, name, avg_fragmentation_in_percent
FROM sys.dm_db_index_physical_stats (DB_ID(N'AdventureWorks'),
OBJECT_ID(N'Sales.SalesOrderDetail'), NULL, NULL, NULL) AS a
JOIN sys.indexes AS b ON a.object_id = b.object_id AND a.index_id =
b.index_id;
```

2.
```
ALTER INDEX IX_SalesOrderDetail_ProductID
ON Sales.SalesOrderDetail REORGANIZE
```

3. Idem punto 1

4.
```
ALTER INDEX IX_SalesOrderDetail_ProductID
ON Sales.SalesOrderDetail REBUILD
```

5. Ídem punto 1

6.

- Para abrir el Asistente para la optimización de motor de base de datos de Microsoft SQL Server puede hacerlo desde el menú Inicio – Todos los Programas – Microsoft SQL Server 2005 – Rendimiento o desde el SQL Server Management Studio – Menú Herramientas.
- Dentro del Asistente seleccionar Archivo – Nueva Sesión. En la solapa General ingresar el nombre de la sesión (cualquier nombre), la opción archivo, asociar el archivo <carpeta de instalación>\Modulo7\CargaTrabajo.xml y seleccionar la base de datos AdventureWorks.

- Para ejecutar presionar “Comenzar Análisis”

Ejercicio 3
Índices XML

En este ejercicio creará índices sobre una columna xml.

1. Escribir y ejecutar una sentencia SQL que permita crear un índice principal en la tabla HumanResource.JobCandidate sobre el campo Resume de tipo de dato xml.

2. Escribir y ejecutar una sentencia SQL que permita crear un índice secundario de tipo PATH sobre el índice del punto 1

3. Verificar la creación de ambos índices.

Respuestas

1.
```
CREATE PRIMARY XML INDEX IXML_JobCandidate_Resume
ON HumanResources.JobCandidate (Resume)
```

2.
```
CREATE XML INDEX IXML_JobCandidate_Resume_Path
ON HumanResources.JobCandidate (Resume)
USING XML INDEX IXML_JobCandidate_Resume
FOR PATH
```

3. En el Explorador de Objetos expandir la tabla HumanResources.JobCandidate. Expandir la solapa Índices (indexes).

Módulo 8: Integridad de Datos

Ejercicio 1
Aplicando restricciones

En este ejercicio creará restricciones para las tablas de las base de datos **Prueba**.

1. Escribir y ejecutar una sentencia SQL que cree una clave principal sobre el campo Legajo en la tabla Empleados creada en el laboratorio 5.

2. Usar el SQL Server Management Studio para crear una clave principal sobre el campo CodigoCategoria en la tabla Categorías creada en el laboratorio 5.

3. Agregar un valor predeterminado 0 (cero) para el campo Sueldo de la tabla Categorías. Usar el SQL Server Management Studio.

4. Escribir y ejecutar una sentencia SQL que agregue como valor predeterminado la fecha del día para el campo FhIngresoEmpresa de la tabla Empresas.

5. Agregar una regla para el campo Sueldo de la tabla Categoría de manera que el valor ingresado no pueda ser menor a cero. Usar el SQL Server Management Studio.

6. Escribir y ejecutar una sentencia SQL que agregue una condición de manera que el campo FhNacimiento no pueda ser anterior al 1/1/1900.

7. Escribir y ejecutar una sentencia SQL que controle que el CUIL de la tabla Empleados sea único.

8. Escribir y ejecutar una sentencia SQL que genere una relación entre el campo CodigoCategoria de la tabla Categorías y el campo Categoría de la tabla Empleados. No permitir acciones en cascada.

9. Ingresar valores en ambas tablas y verificar que todas las restricciones están funcionando correctamente.

Respuestas

1. ALTER TABLE Empleados
   ADD CONSTRAINT PK_Empleados PRIMARY KEY CLUSTERED (Legajo)

2. En el Explorador de Objetos seleccionar la tabla Categorías, presionar el botón derecho de mouse y elegir Modificar (Modify). Seleccionar el campo Legajo y presionar el botón derecho del Mouse. Elegir la opción Establecer Clave Principal.

3. En el Explorador de Objetos seleccionar la tabla Categorías, presionar el botón derecho de mouse y elegir Modificar (Modify). Seleccionar el campo Sueldo y sobre la parte inferior del asistente ingrese el valor predeterminado.

4. ALTER TABLE Empleados
   ADD CONSTRAINT DF_FhIngreso

DEFAULT GetDate() FOR FhIngresoEmpresa

5. En el Explorador de Objetos seleccionar la tabla Categorías presionar el botón derecho de mouse y elegir Modificar (Modify). Sobre la parte superior del Asistente presionar el botón derecho del Mouse y seleccionar Restricciones Check (Check Constraint). Dentro de esta opción presionar el botón Agregar (Add) y en Expresión ingresar la condición: Sueldo > 0.

6.
```
ALTER TABLE Empleados WITH CHECK
ADD CONSTRAINT CK_FhNacimiento
CHECK (FhNacimiento >= '19000101')
```

7.
```
ALTER TABLE Empleados
ADD CONSTRAINT U_Cuil UNIQUE (CUIL)
```

8.
```
ALTER TABLE Empleados WITH CHECK
ADD CONSTRAINT Fk_categorias FOREIGN KEY(Categoria)
REFERENCES Categorias (CodigoCategoria)
```

Ejercicio 2
Creando Desencadenadores

En este ejercicio creará desencadenadores para generara controles de auditoría en las base de datos **Prueba**.

1. Usar el SQL Server Management Studio para crear una tabla llamada Auditoría que contenga las siguientes columnas:

Columna	Tipo de Dato	Observaciones
Movimiento	Int	Identity – No soporta nulos - PK
Tabla	Varchar(50)	No soporta Nulos
Operacion	Char(1)	No soporta Nulos
Codigo	Varchar(50)	No soporta Nulos
Fecha	smalldatetime	No soporta Nulos. Default GetDate()
Detalle	Varchar(200)	
Usuario	Varchar(50)	No soporta Nulos. Default suser_sname()

2. Escribir y ejecutar una sentencia SQL que cree un desencadenador para INSERT llamado t_AltaEmpleado sobre la tabla Empleados que agregue un registro en la tabla Auditoría. En el campo Tabla ingresar el nombre de la tabla (‘Empleados’), en el campo operación ingresar una ‘A’ (por Alta), en el campo Codigo el código que identifica al registro ingresado y en el campo Detalle ‘Alta del empleado: ‘ concatenado con el campo Nombre_Completo de la tabla Empleados. Los otros campos deben tomar su valor predeterminado.

3. Ingresar nuevos registros en la tabla Empleados y comprobar que generan nuevos registros en la tabla Auditoría

4. Escribir y ejecutar una sentencia SQL que cree un desencadenador para DELETE llamado t_BajaEmpleado sobre la tabla Empleados que agregue un registro en la tabla Auditoría. En el campo Tabla ingresar el nombre de la tabla (‘Empleados’), en el campo operación ingresar una ‘B’ (por Baja), en el campo Codigo el código que identifica al registro ingresado y en el campo Detalle ‘Baja del empleado: ‘ concatenado con el campo Nombre_Completo de la tabla Empleados. Los otros campos deben tomar su valor predeterminado.

5. Eliminar un registro de la tabla Empleados y comprobar que se genera un nuevo registro en la tabla Auditoría

6. Escribir y ejecutar una sentencia SQL que cree un desencadenador para UPDATE llamado t_ModifEmpleado sobre la tabla Empleados que agregue un registro en la tabla Auditoría. En el campo Tabla ingresar el nombre de la tabla (‘Empleados’), en el campo operación ingresar una ‘M’ (por modificación), en el campo Codigo el código que identifica al registro ingresado y en el campo Detalle si no se modificaron los campos Nombres y/o Apellidos ingresar ‘Modificación del empleado: ‘ concatenado con el campo Nombre_Completo de la tabla Empleados. Si se modificó alguno de estos campos guardar los datos anteriores y los datos actuales de ambos campos para poder comparar los cambios. Los otros campos deben tomar su valor predeterminado.

7. Modificar algunos registros de la tabla Empleados y comprobar que generan nuevos registros en la tabla Auditoría

8. Verificar la creación de estos desencadenadores usando la vista de catálogo ***sys.triggers***

Respuestas

1. En el Explorador de Objetos seleccionar la solapa Tablas (Tables) de la base de datos, presionar el botón derecho del Mouse, elegir Nueva tabla. Ingresar los datos.

2.
```
CREATE TRIGGER t_AltaEmpleados ON Empleados
FOR INSERT AS
BEGIN
INSERT INTO Auditoria (Tabla, Operacion, Codigo, Detalle)
SELECT 'Empleados','A',Legajo, 'Alta del Empleado ' + NombreCompleto
FROM inserted
END
```

4.
```
CREATE TRIGGER t_BajaEmpleados ON Empleados
FOR DELETE AS
BEGIN
INSERT INTO Auditoria (Tabla, Operacion, Codigo, Detalle)
SELECT 'Empleados','B',Legajo, 'Baja del Empleado ' + NombreCompleto
FROM deleted
END
```

6.
```
CREATE TRIGGER t_ModifEmpleados ON Empleados
FOR UPDATE AS
BEGIN
INSERT INTO Auditoria (Tabla, Operacion, Codigo, Detalle)
SELECT 'Empleados','M', i.Legajo,
CASE WHEN UPDATE(Nombres) or UPDATE(Apellidos) Then
'Modificación del Empleado ' + d.NombreCompleto + ' Ahora ' + i.NombreCompleto
ELSE
'Modificación del Empleado ' + d.NombreCompleto
END
FROM inserted i INNER JOIN  deleted d ON i.Legajo=d.Legajo
END
```

8.
```
SELECT * FROM sys.triggers
```

Ejercicio 3
Creando Esquemas XML

En este ejercicio creará nuevos esquemas xml sobre la base de datos **AdventureWorks**.

1. Levantar el archivo <carpeta de instalación>\Modulo8***EsquemaXML.txt***. Usar este documento xml para escribir y ejecutar una sentencia SQL que cree un nuevo esquema xml llamado esq_datosPersonales.

2. Crear una nueva tabla llamada Datos con las siguiente columnas. Puede usar el SQL Server Management Studio o la sentencia CREATE TABLE

Columna	Tipo de Dato	Observaciones
Codigo	Int	No soporta nulos - PK
Datos	Xml	No soporta Nulos. Asociar al esquema esq_datosPersonales

3. Ingresar dos nuevos registros en esta tabla tomando para el campo Datos los documentos xml contenidos en el archivo <carpeta de instalación>\Modulo8***Datos.txt***. Se pueden ingresar? Por que?

Respuestas

1.
```
CREATE XML SCHEMA COLLECTION esq_DatosPersonales
AS
'<xs:schema id="Datos" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
   <xs:element name="Datos" msdata:MainDataTable="Datos"
msdata:UseCurrentLocale="true">
<xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
         <xs:element name="DatosPersonales">
         <xs:complexType>
           <xs:sequence>
                  <xs:element name="NombreApellido" type="xs:string" minOccurs="0" />
                  <xs:element name="Domicilio" type="xs:string" minOccurs="0" />
                  <xs:element name="Empresa" type="xs:string" minOccurs="0" />
           </xs:sequence>
         </xs:complexType>
        </xs:element>
     </xs:choice>
</xs:complexType>
</xs:element>
</xs:schema>'
```

2.
```
CREATE TABLE Datos(
          Codigo int NOT NULL,
          Datos xml(CONTENT [dbo].[DatosPersonales]) NOT NULL,
CONSTRAINT [PK_Datos] PRIMARY KEY CLUSTERED
(Codigo ASC)
```

3. Este registro ingresa sin problemas ya que el xml cumple con el esquema asociado al campo.

```
INSERT INTO Datos (codigo, datos)
VALUES (1,'<Datos>
<DatosPersonales>
        <NombreApellido>Juan Perez</NombreApellido>
        <Domicilio>Libertador 234</Domicilio>
        <Empresa>Microsoft</Empresa>
</DatosPersonales>
</Datos>')
```

Este registro da un error ya que el xml no cumple con el esquema asociado. El elemento Nombre no existe, debería ser NombreApellido.

```
INSERT INTO Datos (codigo, datos)
VALUES (2,'<Datos>
<DatosPersonales>
        <Nombre>Juan Perez</Apellido>
        <Domicilio>Libertador 234</Domicilio>
        <Empresa>Microsoft</Empresa>
</DatosPersonales>
</Datos>')
```

Módulo 9: Vistas

Ejercicio 1
Creando Vistas

En este ejercicio creará vistas para las tablas de las base de datos **AdventureWorks**.

1. Escribir y ejecutar una sentencia SQL que cree una vista llamada v_Provincias que devuelva todas las provincias tomadas de la tabla Person.StateProvince en donde figure el código y el nombre del país asociado a cada provincia. Tomar los datos de los países de la tabla Person.CountryRegion.

2. Escribir y ejecutar una sentencia SQL que devuelva los datos de la vista solo para el país “US” ordenado por el nombre de la provincia.

3. Crear una nueva vista llamada v_ProductosMasCaros que liste los 10 productos más caros de la tabla Production.Product. Mostrar solo el código, nombre y precio del producto. Usar el SQL Server Management Studio para crear esta vista. Ejecutarla.

4. Ejecutar el procedimiento almacenado ***sp_HelpText*** para comprobar la sintaxis de la vista ProductosMasCaros.

5. Verificar la creación de las vistas usando la vista de catálogo ***sys.views***.

6. Escribir y ejecutar una sentencia SQL que cree una vista llamada v_ProductLocation que devuelta todos los datos de la tabla Production.Location cuando el campo CostRate es distinto de 0 (cero).

7. Ingresar un nuevo registro a través de la vista v_ProductLocation con CostRate = 0. Se pudo ingresar el producto?

8. Modificar la vista v_ProductLocation para agregarle la cláusula WITH CHECK OPTION.

9. Ingresar un nuevo registro a través de la vista v_ProductLocation con CostRate = 0. Se pudo ingresar el producto?

Respuestas

1.
```
CREATE VIEW v_Provincias
AS
SELECT p.StateProvinceId, p.StateProvinceCode, p.CountryRegionCode, p.Name as
NombreProvincia, c.Name as NombrePais
FROM Person.StateProvince p INNER JOIN Person.CountryRegion c ON
p.CountryRegionCode=c.CountryRegionCode
```

2.
```
SELECT * FROM v_Provincias
WHERE CountryRegionCode='US'
ORDER BY NombreProvincia
```

3. En el explorador de Objetos seleccionar Vistas (Views) y presionar el botón derecho del mouse, Nueva Vista. Se abre el asistente para la creación de vistas. En la ventana de Agregar Tablas seleccionar la tabla Production.Product y cerrarla. Sobre el asistente seleccionar lo campos necesarios sobre la tabla (ProductId, Name y ListPrice). En la segunda sección del asistente seleccionar Descendente sobre el registro de ListPrice en la columna Tipo de Ordenamiento. En la tercera sección se va escribiendo la sentencia SQL. Agregar la cláusula TOP 10. Si desea ejecutar la instrucción para probarla presionar el botón derecho del mouse sobre la primera sección. Ejecutar SQL. El resultado se muestra en la cuarta sección. Grabar usando el icono del disquette de la barra de herramientas. La sentencia SQL debería ser:

```
SELECT  TOP 10 ProductID, Name, ListPrice
FROM  Production.Product
ORDER BY ListPrice DESC

SELECT * FROM ProductosMasCaros
```

4. EXEC sp_helptext 'ProductosMasCaros'

5. SELECT * FROM sys.views

6.
```
CREATE VIEW v_ProductLocation
AS
SELECT * FROM Production.Location
WHERE CostRate <> 0
```

7.
```
INSERT INTO v_ProductLocation (Name, CostRate, Availability)
VALUES ('Prueba',0,0)
```

   El registro se ingresa en la tabla sin problemas

8.
```
ALTER VIEW v_ProductLocation
AS
SELECT * FROM Production.Location
WHERE CostRate <> 0
WITH CHECK OPTION
```

9.
```
INSERT INTO v_ProductLocation (Name, CostRate, Availability)
VALUES ('Otra Prueba',0,0)
```

   El registro no puede ingresarse por que no cumple con la cláusula WHERE de la vista.

Ejercicio 2
Optimizaciones a través de Vistas

En este ejercicio creará una vista indexada en la base de datos **AdventureWorks**.

1. Escribir y ejecutar una sentencia SQL para modificar la vista HumanResource.vEmployee predefinida en esta base de datos. Agregarle la cláusula SCHEMABINDING.

2. Escribir y ejecutar una sentencia SQL para crear un índice sobre los campos LastName, FirstName en la vista HumanResource.vEmployee

Respuestas

1.
```
ALTER VIEW HumanResources.vEmployee
WITH SCHEMABINDING
AS
SELECT e.EmployeeID, c.Title, c.FirstName, c.MiddleName, c.LastName, c.Suffix,
    e.Title AS JobTitle, c.Phone, c.EmailAddress, c.EmailPromotion, a.AddressLine1,
    a.AddressLine2, a.City, sp.Name AS StateProvinceName, a.PostalCode, cr.Name
   AS CountryRegionName,  c.AdditionalContactInfo
FROM HumanResources.Employee e
INNER JOIN Person.Contact c ON c.ContactID = e.ContactID
INNER JOIN HumanResources.EmployeeAddress ea
ON e.EmployeeID = ea.EmployeeID
INNER JOIN Person.Address a ON ea.AddressID = a.AddressID
INNER JOIN Person.StateProvince sp ON sp.StateProvinceID = a.StateProvinceID
INNER JOIN Person.CountryRegion cr
ON cr.CountryRegionCode = sp.CountryRegionCode
```

2.
```
CREATE UNIQUE CLUSTERED INDEX [IX_vEmployee]
ON HumanResources.vEmployee (LastName, FirstName)
```

Módulo 10: Procedimientos Almacenados

Ejercicio 1
Creando procedimientos almacenados

En este ejercicio creará procedimientos almacenados en las base de datos **AdventureWorks**.

1. Escribir una sentencia SQL que cree un procedimiento almacenado llamado TraerTarjetasCredito que devuelta todos los registros de la tabla Sales.CreditCard.

2. Ejecutar el procedimiento almacenado creado en el punto 1.

3. Verificar la sintaxis del mismo procedimiento usando el procedimiento almacenado de sistemas ***sp_helptext***.

4. Modificar el procedimiento de manera que el mismo quede cifrado. Verificar que esté cifrado. Ejecutarlo

5. Verificar todos los procedimiento creados en esta base de datos usando la vista de catálogo ***sys.procedures***

Respuestas

1.
```
CREATE PROC TraerTarjetasCredito
AS
SELECT * FROM Sales.CreditCard
```

2.
```
EXEC TraerTarjetasCredito
```

3.
```
EXEC sp_helptext 'TraerTarjetasCredito'
```

4.
```
ALTER PROC TraerTarjetasCredito
WITH ENCRYPTION
AS
SELECT * FROM Sales.CreditCard

EXEC sp_helptext 'TraerTarjetasCredito'

EXEC TraerTarjetasCredito
```

5.
```
SELECT * FROM sys.procedures
```

Ejercicio 2
Creando procedimientos almacenados con parámetros

En este ejercicio creará procedimientos almacenados con parámetros en las base de datos **AdventureWorks**.

1. Modificar el procedimiento almacenado TraerTarjetasCredito de manera que no esté cifrado. Agregarle un parámetro llamado Codigo de tipo int opcional con valor predeterminado en 0 (cero). Modificar el código de manera que si el parámetro es 0 (cero) devuelva toda la tabla y si el parámetro no es 0 (cero) devuelva el registro cuyo código coincide con el parámetro.

2. Ejecutar el procedimiento almacenado creado en el punto 1. Probar pasándole como parámetro 0 y 5. También ejecutar el procedimiento sin parámetro.

3. Escribir una sentencia SQL que cree un nuevo procedimiento almacenado llamado AgregarTarjetasCredito que permita ingresar un nuevo registro en la tabla Sales.CreditCard. El procedimiento debe recibir parámetros para todos los campos de la tabla que no tengan valores predeterminados.

4. Ejecutar el procedimiento creado en el punto 3 pasando los parámetros por posición

5. Ejecutar el procedimiento creado en el punto 3 pasando los parámetros por nombre

6. Escribir una sentencia SQL que cree un procedimiento almacenado llamado TraerNombreProducto que devuelva por parámetro el nombre de un producto de la tabla Production.Product asociado a un código que recibe también por parámetro.

7. Ejecutar el procedimiento creado en el punto anterior y mostrar el nombre del producto recibido.

8. Modificar el procedimiento almacenado AgregarTarjetasCredito de manera que devuelva el código identificatorio (valor Identity) de la tarjeta ingresada.

9. Ejecutar el procedimiento creado en el punto anterior y mostrar el código recibido.

Respuestas

1.
```
ALTER PROC TarjetasCredito @Codigo int=0
AS
IF @Codigo=0
        SELECT * FROM Sales.CreditCard
ELSE
        SELECT * FROM Sales.CreditCard WHERE CreditCardId=@Codigo
```

2.
```
EXEC TarjetasCredito 0

EXEC TarjetasCredito 5

EXEC TarjetasCredito
```

3. CREATE PROC AgregarTarjetasCredito @Tipo varchar(50), @Numero varchar(25), @Mes tinyint, @Anio smallint
AS
INSERT INTO Sales.CreditCard (CardType, CardNumber, ExpMonth, ExpYear)
VALUES (@Tipo, @Numero, @Mes, @Anio)

4. EXEC AgregarTarjetasCredito 'Visa', '132465435465421',1,2005

5. EXEC AgregarTarjetasCredito @Mes=10, @Anio=2005, @Tipo='Visa', @Numero='345465435465421'

6. CREATE PROC TraerNombreProducto @Codigo int, @Nombre varchar(50) OUTPUT
AS
SELECT @Nombre=Name FROM Production.Product WHERE Productid=@Codigo

7. DECLARE @Nom varchar(50)
EXEC TraerNombreProducto 3, @Nom OUTPUT
PRINT @Nom

8. ALTER PROC AgregarTarjetasCredito @Tipo varchar(50), @Numero varchar(25), @Mes tinyint, @Anio smallint
AS
INSERT INTO Sales.CreditCard (CardType, CardNumber, ExpMonth, ExpYear)
VALUES (@Tipo, @Numero, @Mes, @Anio)
RETURN @@Identity

9. DECLARE @Id int
EXEC @Id = AgregarTarjetasCredito 'American Express','138475093845',5,2008
PRINT @Id

Ejercicio 3
Manejo de Errores

En este ejercicio creará procedimientos almacenados con control de errores en las base de datos **AdventureWorks**.

1. Escribir una sentencia SQL que cree un procedimiento almacenado llamado EliminarProductos que recibe un parámetro de tipo int. La función de este procedimiento es eliminar de la tabla Production.Product el producto recibido en el parámetro. Programar control de errores usando Try-Catch. Si se produce un error devolver los siguientes datos del error: Número, Descripción, Procedimiento donde se produjo el error y el numero de línea.

2. Ejecutar el procedimiento anterior con el código 980.

3. Escribir una sentencia SQL que cree un procedimiento almacenado llamado AgregarPaises que reciba dos parámetros: Código varchar(3) y Nombre varchar(50), que permita agregar un nuevo registro en la tabla Person.StateProvince. Programar control de errores con los siguientes datos: Número, Descripción, Procedimiento donde se produjo el error y el numero de línea, severidad del error y estado.

4. Ejecutar el procedimiento anterior para el código 'AR'.

Respuestas

1.
```
CREATE PROC EliminarProductos @Prod int
AS
BEGIN TRY
        DELETE FROM Production.Product
        WHERE ProductID = @Prod;
END TRY
BEGIN CATCH
        SELECT  ERROR_NUMBER() as NumeroError, ERROR_MESSAGE() as
        Mensaje, ERROR_PROCEDURE() as Procedimiento, ERROR_LINE() as
        Linea
END CATCH;
```

2.
```
EXEC EliminarProductos 980
```

3.
```
CREATE PROC AgregarPaises @Codigo varchar(3), @Nombre varchar(50)
AS
BEGIN TRY
        INSERT INTO Person.CountryRegion (CountryRegionCode, Name)
        VALUES (@Codigo, @Nombre)
END TRY
BEGIN CATCH
        SELECT  ERROR_NUMBER() as NumeroError, ERROR_MESSAGE() as
        Mensaje, ERROR_PROCEDURE() as Procedimiento, ERROR_LINE() as
        Linea, ERROR_SEVERITY() as Severidad, ERROR_STATE() as Estado
END CATCH;
```

4. EXEC AgregarPaises 'AR','Argentina'

Módulo 11: Funciones

Ejercicio 1
Creando funciones

En este ejercicio creará funciones de usuario en las base de datos **AdventureWorks**.

1. Escribir una sentencia SQL que cree una función escalar llamada f_TipoResponsable que recibe un parámetro entero y que devuelve el nombre de los tipos de responsable según esta codificación:
   1- Responsable Inscripto
   2- Monotributista
   3- Exento
   4- No Responsable
   5- Consumidor Final
   Sino Código Incorrecto

2. Ejecutar la función creada en el punto 1.

3. Escribir una sentencia SQL que cree una función con valor de tabla en línea llamada f_ProvinciasxPais que devuelva todas las provincias asociadas a un país recibido por parámetro.

4. Ejecutar la función anterior creada en el punto 3 probando distintas opciones de uso.

5. Escribir una sentencia SQL que cree una función con valor de tabla con múltiples instrucciones que tome los campos TransactionID, ProductId, Name, TransactionDate, Quantity y ActualCost de las tablas Production.TransactionHistoryArchive o Production. TransactionHistory según corresponda a la fecha recibida por parámetro. La tabla Production.TransactionHistoryArchive contiene información hasta 31/8/2003, el resto de la información se encuentra en la otra tabla.

6. Ejecutar la función anterior creada en el punto 5 probando distintas opciones de uso.

7. Verificar la creación de estas funciones usando la vista de catálogo ***sys.sql_module.*** Como esta vista tiene información sobre varios objetos genere una relación con la vista ***sys.objects*** de manera de filtrar solo la funciones. Las funciones se identifican en la vista ***sys.objects*** por el campo Type igual a FN (función escalar), IF (función en línea), TF (función múltiples instrucciones).

Respuestas

1.
```
CREATE FUNCTION f_TipoResponsable (@Cod tinyint)
RETURNS varchar(50)
AS
BEGIN
DECLARE @Nombre varchar(50)
SET @Nombre= CASE WHEN @Cod = 1 THEN 'Responsable Inscripto'
                  WHEN @Cod = 2 THEN 'Monotributista'
                  WHEN @Cod = 3 THEN 'Exento'
                  WHEN @Cod = 4 THEN 'No Responsable'
```

```
                                        WHEN @Cod = 5 THEN 'Consumidor Final'
                                        ELSE 'Código Incorrecto' END
        RETURN @Nombre
END
```

2.
```
SELECT dbo.f_TipoResponsable(1)
```

3.
```
CREATE FUNCTION f_ProvinciasxPais (@Pais char(2))
RETURNS TABLE
AS
RETURN
(SELECT * FROM Person.StateProvince
WHERE CountryRegionCode=@Pais)
```

4.
```
SELECT * FROM dbo.f_ProvinciasxPais('US') ORDER BY Name

SELECT StateProvinceCode, Name FROM dbo.f_ProvinciasxPais('US')
ORDER BY Name
```

5.
```
CREATE FUNCTION f_HistorialTransacciones (@Fecha smalldatetime)
RETURNS @miTabla TABLE
(
        NumTransaccion int,
        NumProducto int,
        NombreProducto varchar(50),
        Fecha smalldatetime,
        Cantidad int,
        Costo money
)
AS
BEGIN
IF @Fecha < '20030901'
        INSERT INTO @miTabla (NumTransaccion, NumProducto, NombreProducto,
        Fecha, Cantidad, Costo)
        SELECT t.TransactionID, t.ProductId, p.Name, t.TransactionDate, t.Quantity,
        t.ActualCost
        FROM Production.TransactionHistoryArchive t
        INNER JOIN Production.Product p
        ON t.Productid=p.Productid
        WHERE t.TransactionDate=@Fecha
ELSE
        INSERT INTO @miTabla (NumTransaccion, NumProducto, NombreProducto,
        Fecha, Cantidad, Costo)
        SELECT t.TransactionID, t.ProductId, p.Name, t.TransactionDate, t.Quantity,
        t.ActualCost
        FROM Production.TransactionHistory t INNER JOIN Production.Product p
        ON t.Productid=p.Productid
        WHERE t.TransactionDate=@Fecha
RETURN
END
```

6.
```
SELECT * FROM dbo.f_HistorialTransacciones('20020201')
SELECT * FROM dbo.f_HistorialTransacciones('20020201')
WHERE NumProducto=771
SELECT * FROM dbo.f_HistorialTransacciones('20040201')
```

7.
```
SELECT m.*, o.Type_Desc FROM sys.sql_modules m
INNER JOIN sys.objects o
ON m.object_id = o.object_id AND type IN ('FN', 'IF', 'TF');
```

Ejercicio 2
Contexto de ejecución

En este ejercicio generará ejecuciones a nombre de otros usuarios en las base de datos **AdventureWorks**.

1. Utilizar la siguiente sentencia SQL para crear un inicio de sesión llamado Login1 con una contraseña 123456.
   CREATE LOGIN Login1 WITH PASSWORD = '123456'

2. Utilizar la siguiente sentencia SQL para crear un usuario dentro de la base de datos AdventureWorks llamado Usuario1 asociado al inicio de sesión creado en el punto 1
   CREATE USER Usuario1 FOR LOGIN Login1;

3. Escribir una sentencia SQL que cree un procedimiento almacenado llamado NombreUsuario que ejecute a nombre del dueño y que devuelva el nombre del usuario.

4. Ejecutar el procedimiento almacenado creado en el punto 3

5. Cambiar el contexto de ejecución y asociarlo al Usuario1 creado en el punto2

6. Volver a ejecutar el procedimiento almacenado creado en el punto 3.

7. Cancelar el contexto de ejecución generado en el punto 5

8. Volver a ejecutar el procedimiento almacenado creado en el punto 3.

Respuestas

3.
```
CREATE PROCEDURE NombreUsuario
WITH EXECUTE AS OWNER
AS
SELECT user_name();
```

4. EXEC NombreUsuario
   El procedimiento ejecuta perfectamente.

5. EXECUTE AS USER = 'Usuario1'

6. El procedimiento genera un error de permisos

7. REVERT

8. El procedimiento ejecuta perfectamente.

Módulo 12: Transacciones y Bloqueos

Ejercicio 1
Trabajando con transacciones y bloqueos

En este ejercicio trabajará con transacciones y bloqueos en las base de datos **AdventureWorks**.

1. Escribir una sentencia SQL que cree un procedimiento almacenado llamado NuevoPais que ingrese un país en la tabla Person.CountryRegion y que además ingrese la moneda correspondiente a ese país en la tabla Sales.CountryRegionCurrency. El procedimiento deberá tener tres parámetros: Código del País, Nombre del País y el Código de la moneda. Generar un ambiente de transacciones con control de errores.

2. Ejecutar el procedimiento almacenado en el punto 1 usando “PP” como código de país, “Prueba” para el nombre del país y EUR (por Euro) para la moneda.

3. Ejecutar el procedimiento almacenado en el punto 1 usando “AR” como código de país, “Argentina” para el nombre del país y EUR (por Euro) para la moneda.

4. Establecer el modo de transacción implícita en ON. Hacer alguna modificación en alguna tabla mediante el uso de una sentencia SQL en la misma ventana de consultas. Cerrar la ventana de consultas. El cambio efectuado se confirmo? Porque?

5. Establecer nuevamente el modo de transacción implícita en ON y volver a generar una modificación. Dejar la ventana de consultas abierta. Abrir una nueva ventana de consultar y verificar los bloqueos generados

6. Manteniendo la transacción abierta, abrir el Monitor de Actividades y analizar los bloqueos en curso.

Respuestas

1.
```
CREATE PROCEDURE NuevoPais @Pais varchar(3), @Nombre varchar(50),
@Moneda char(3)
AS
BEGIN TRANSACTION
BEGIN TRY
        INSERT INTO Person.CountryRegion (CountryRegionCode, Name)
        VALUES (@Pais, @Nombre)

        INSERT INTO Sales.CountryRegionCurrency (CountryRegionCode,
        CurrencyCode)
        VALUES (@Pais, @Moneda)
END TRY
BEGIN CATCH
        SELECT  ERROR_NUMBER() as NumeroError, ERROR_MESSAGE() as
        Mensaje, ERROR_PROCEDURE() as Procedimiento, ERROR_LINE() as
        Linea
```

```
        IF @@TranCount > 0
                ROLLBACK TRANSACTION
END CATCH;
IF @@TranCount > 0
        COMMIT TRANSACTION
```

2. EXEC NuevoPais 'PP','Prueba','EUR'

3. EXEC NuevoPais 'AR','Argentina','EUR'

4. SET IMPLICIT_TRANSACTIONS ON
   GO

   INSERT INTO Person.CountryRegion (CountryRegionCode, Name)
   VALUES ('ZZ','Otra Prueba')

   Si se establece el modo de transacción implícita en ON cualquier instrucción inicia una transacción. Si la misma no es confirmada explícitamente la transacción se revierte al cerrar la conexión.

5. SELECT * FROM sys.dm_tran_locks

   SELECT * FROM sys.dm_tran_database_transactions

6. En el Explorador de Objetos expandir Administración y seleccionar el Monitor de Actividades

Módulo 13: Código Manejado en la Base de Datos

Ejercicio 1
Trabajando con código manejado

En este ejercicio armará una dll para ser importada por la base de datos **AdventureWorks** y permitir crear objetos CLR

1. Habilitar para la base de datos AdventureWorks que pueda trabajar con código manejado usando el procedimiento almacenado de sistema ***sp_configure***.

2. En la <carpeta de instalación>\Modulo13 encontrará un proyecto .Net de tipo Biblioteca de Clases (Class Library) llamado CursoSQL. Este proyecto tiene una sola clase llamada StoredProcedures que a su vez contiene un único método PromedioPrecios. Este método calcula el precio promedio de los productos de la tabla Production.Product. A su vez en la misma carpeta se encuentra el archivo CursoSQL.dll que contiene la compilación de este proyecto.

3. Generar la importación de este proyecto para la base de datos AdventureWorks.

4. Escribir una sentencia SQL que cree un procedimiento almacenado que invoque al método PromedioPrecios de la clase StoredProcedures del ensamblado CursoSQL.

5. Ejecutar el procedimiento almacenado creado en el punto anterior y mostrar el valor promedio encontrado.

Respuestas

1.
```
sp_configure 'clr enabled', 1
GO
RECONFIGURE
GO
```

2. Puede abrir el proyecto presionando doble click sobre Curso.sln si tiene instalado el Microsoft Visual Studio 2005.

3.
```
CREATE ASSEMBLY CursoSQL
FROM <carpeta de instalación>\Modulo13\CursoSQL.dll'
WITH PERMISSION_SET = SAFE
```

4.
```
CREATE PROCEDURE PromedioPrecios (@Prom int OUTPUT)
AS EXTERNAL NAME CursoSQL.[CursoSQL.StoredProcedures].PromedioPrecios
```

5.
```
DECLARE @Prom int
EXEC PromedioPrecios @Prom OUTPUT
Print @Prom
```