CONSTRUCCIÓN DE GUIONES (SCRITS)

🕮 Introducción

Hasta ahora hemos estudiado el acceso a la base de datos de manera interactiva. Esta forma es necesaria para aprender las sentencias básicas del lenguaje SQL y para aquellos usuarios puntuales y expertos.
En general se realizará el ACCESO A BASES DE DATOS DESDE PROGRAMAS, en este caso tenemos dos opciones:
- Utilizar lenguajes de programación habituales (C++, Java,...). Esto supone incluir las sentencias SQL donde se necesiten en el programa, utilizando unas normas concretas. Al lenguaje en este caso se denomina **huésped o anfitrión** (host) y de las sentencias SQL se dice que están **embebidas**.
- Utilizar lenguajes propietarios que amplían el SQL e incluyen algunas funcionalidades típicas de los lenguajes de programación (PL/SQL de Oracle, **Transact SQL en SQL Server**, ...).

Transact SQL es el lenguaje de programación que proporciona SQL Server para ampliar SQL con los elementos característicos de los lenguajes de programación: variables, sentencias de control de flujo, bucles ...
Cuando se desea realizar una aplicación completa para el manejo de una base de datos relacional, resulta necesario utilizar alguna herramienta que soporte la capacidad de consulta del SQL y la versatilidad de los lenguajes de programación tradicionales. **Transact SQL** es el lenguaje de programación que proporciona **SQL Server** para extender el SQL estándar con otro tipo de instrucciones.

Un script de **Transact SQL** es un conjunto de sentencias de **Transact SQL** en formato de texto plano que se ejecutan en un servidor de **SQL Server**.
Un script está compuesto por uno o varios lotes. Un lote delimita el alcance de las variables y sentencias del script. Dentro de un mismo script se diferencian los diferentes lotes a través de las instrucción **GO.**

```
-- Este es el primer lote del script
SELECT * FROM HOSPITAL
GO -- GO es el separador de lotes
-- Este es el segundo lote del script
SELECT getdate()  -- getdate() es una función integrada que devuelve
                  -- la fecha
```

🕮 Elementos de la sintaxis de Transact-SQL.

Comentarios

Los comentarios son cadenas de texto que no se ejecutan. Se pueden utilizar de una de estas dos formas: en una línea de una instrucción o como un bloque (mediante dos guiones (--)).
Para crear bloques de varias líneas de comentarios, coloca un carácter de comentario (/*) al comienzo del texto del comentario, escribe tus anotaciones y, después, concluye el comentario con un carácter de cierre de comentario (*/). Ejemplo:

```
-- Esto es un comentario de linea simple

/*
Este es un comentario con varias líneas.
Conjunto de Lineas.
*/
declare @nombre varchar(50)-- declare declara una variable
                           -- @nombre es el identificador de la
                           -- variable de tipo varchar
```

Tipos de datos

Los tipos de datos limitan los tipos de valores que se pueden almacenar en una base de datos.

Variables

Las variables son elementos del lenguaje con valores asignados. Una variable local es una variable definida por el usuario en una instrucción DECLARE; se le asigna un valor inicial en una instrucción SET o SELECT y, después, se utiliza en la instrucción, programa por lotes o procedimiento en el que se declaró. Una variable local se identifica mediante un símbolo arroba (@) que precede a su nombre mientras que una variable global incluye dos símbolos arroba delante de su nombre.
Las variables locales sólo duran el tiempo correspondiente a un proceso por lotes, mientras que las variables globales tienen la misma duración que una sesión.

```
DECLARE @nombre VARCHAR(100)
-- La consulta debe devolver un único registro
SET @nombre =  (SELECT nombre
                FROM CLIENTES
                WHERE ID = 1)
-- Otra forma
SELECT @nombre = nombre
FROM CLIENTES
WHERE ID = 1
```

Operadores

Los tipos de operadores son los que se pueden utilizar en las sentencias estándar de SQL. Aritméticos. Comparación. Concatenación de cadenas. Lógicos.

Identificadores

- **Identificadores estándar**
- **Identificadores delimitados.** Deben encerrarse entre corchetes ([ ]) o dobles comillas (" ")

Expresiones

Las *expresiones* son combinaciones de símbolos y operadores que dan como resultado un único valor. Puede tratarse de expresiones simples, como una constante, variable, columna o valor escalar, o expresiones complejas que se crean mediante la conexión, por medio de operadores, de una o varias expresiones simples.

El tipo de datos del resultado depende de los elementos que forman la expresión. Durante la evaluación del resultado, se realizan con frecuencia conversiones implícitas de los tipos de datos de los elementos que componen la expresión.

Estructuras de control en Transact SQL

🕮 Estructura condicional IF

La estuctura condicional **IF** permite evaluar una expresión booleana (resultado SI - NO), y ejecutar las operaciones contenidas en el bloque formado por **BEGIN END**.

```
IF (<expresion>)
  BEGIN
        ...
  END
ELSE IF (<expresion>)
  BEGIN
        ...
  END
ELSE
  BEGIN
        ...
  END
```

Ejemplo:

```
DECLARE @coPais int,
        @descripcion varchar(255)
set @coPais = 5
set @descripcion = 'España'
IF EXISTS(SELECT * FROM PAISES
          WHERE CO_PAIS = @coPais)
  BEGIN
        ...
  END

ELSE
  BEGIN
        ...
  END
```

🕮 Estructura condicional CASE

La estructura condicional **CASE** permite evaluar una expresion y devolver un valor u otro.

La sintaxis general de case es:

```
CASE <expresion>
        WHEN <valor_expresion> THEN <valor_devuelto>
        WHEN <valor_expresion> THEN <valor_devuelto>
        ELSE <valor_devuelto>  -- Valor por defecto
END

--- otra forma

CASE
        WHEN <expresion> = <valor_expresion> THEN <valor_devuelto>
```

```
        WHEN <expresion> = <valor_expresion> THEN <valor_devuelto>
        ELSE <valor_devuelto> -- Valor por defecto
END
```

Ejemplo:

```
DECLARE @Web varchar(100),
            @diminutivo varchar(3)
    SET @diminutivo = 'DJK'

    SET @Web = (CASE
                    WHEN @diminutivo = 'DJK' THEN (SELECT  web
                                                   FROM WEBS
                                                   WHERE id=1)
                    WHEN @diminutivo = 'ALM' THEN (SELECT  web
                                                   FROM WEBS
                                                   WHERE id=2)
                    ELSE 'www.mysql.org'
                END)
```

🕮 Bucle WHILE

El bucle **WHILE** se repite mientras expresion se evalue como verdadero.
Es el único tipo de bucle del que dispone **Transact SQL**.

```
WHILE <expresion>
    BEGIN
       ...
    END
```

Ejemplo:

```
  DECLARE @coRecibo int
    WHILE  EXISTS (SELECT *
                   FROM RECIBOS
                   WHERE PENDIENTE = 'S')-- Ojo, la subconsulta se ejecuta
                                         -- una vez por cada iteracion
                                         -- del bucle!
    BEGIN
        SET @coRecibo = (SELECT TOP 1 CO_RECIBO
                         FROM RECIBOS WHERE PENDIENTE = 'S')
        ...
    END
```

🕮 Control de errores en Transact SQL

Uso de TRY CATCH

A partir de la versión 2005, **SQL Server** proporciona el control de errores a través de las instrucciones **TRY** y **CATCH**.
La sintaxis de **TRY CATCH** es la siguiente:

```
BEGIN TRY
        ...
END TRY
BEGIN CATCH
        ...
END CATCH
```

Funciones especiales de Error

Las funciones especiales de error, están disponibles únicamente en el bloque **CATCH** para la obtención de información detallada del error.

Son:

- **ERROR_NUMBER()**, devuelve el número de error.
- **ERROR_SEVERITY()**, devuelve la severidad del error.
- **ERROR_STATE()**, devuelve el estado del error.
- **ERROR_PROCEDURE()**, devuelve el nombre del procedimiento almacenado que ha provocado el error.
- **ERROR_LINE()**, devuelve el número de línea en el que se ha producido el error.
- **ERROR_MESSAGE()**, devuelve el mensaje de error.

La variable de sistema @@ERROR

En versiones anteriores a **SQL Server 2005**, no estaban disponibles las instrucciones **TRY CATCH**. En estas versiones se controlaban los errores utilizando la variable global de sistema @@ERROR, que almacena el número de error producido por la última sentencia **Transact SQL** ejecutada.

Ejemplo:

```
DECLARE @divisor   int ,
        @dividendo int ,
        @resultado int

 SET @dividendo = 100
 SET @divisor = 0
 -- Esta linea provoca un error de division por 0
 SET @resultado = @dividendo/@divisor

 IF @@ERROR = 0
     BEGIN
         ...
     END
 ELSE
     BEGIN
          ...
     END
```

Generar un error con RAISERROR

En ocasiones es necesario provocar voluntariamente un error, por ejemplo nos puede interesas que se genere un error cuando los datos incumplen una regla de negocio.
Podemos provocar un error en tiempo de ejecución a través de la función RAISERROR.

```
DECLARE @tipo int,
        @clasificacion int

 SET @tipo = 1
 SET @clasificacion = 3
 IF (@tipo = 1 AND @clasificacion = 3)
```

```
BEGIN
   RAISERROR ('El tipo no puede valer uno y la clasificacion 3',
              16, -- Severidad
              1   -- Estado
              )
END
```

La función RAISERROR recibe tres parámetros, el mensaje del error (o código de error predefinido), la severidad y el estado.
La severidad indica el grado de criticidad del error. Admite valores de 0 al 25, pero solo podemos asignar valores del 0 al 18. Los errores el 20 al 25 son considerados fatales por el sistema, y cerraran la conexión que ejecuta el comando **RAISERROR**. Para asignar valores del 19 al 25 necesitares ser miembros de la función de **SQL Server** sysadmin.
El estado es un valor para permitir que el programador identifique el mismo error desde diferentes partes del código. Admite valores entre 1 y 127, permite tratar .

🕮 SQL dinámico en Transact SQL

Transact SQL permite dos formas de ejecutar SQL dinámico (construir sentencias SQL dinámicamente para ejecutarlas en la base de datos):

- La instrucción **EXECUTE -** o simplemente **EXEC**
- El procedimiento almacenado **sp_executesql**

La directiva EXEC se usa también para ejecutar una función definida por el usuario, un procedimiento de sistema, un procedimiento almacenado definido por el usuario o un procedimiento almacenado extendido. También puede controlar la ejecución de una cadena de caracteres dentro de un lote de Transact-SQL.

La instrucción EXECUTE

La instrucción EXECUTE - o simplemente EXEC - permite ejecutar una cadena de caracteres que representa una sentencia SQL. La cadena de caracteres debe ser de tipo **nvarchar** .
El siguiente ejemplo muestra como ejecutar una cadena de caracteres con la instrucción **EXEC**.

```
DECLARE @sql nvarchar(1000)

SET @sql = 'SELECT
                COD_PAIS,
                NOMBRE_PAIS,
                ACTIVO,
                FX_ALTA
            FROM
                PAISES'
EXEC (@sql)
```

También con SQL dinámico podemos ejecutar sentencias de tipo DDL (Data Definition Languaje), como CREATE TABLE.

```
DECLARE @sql nvarchar(1000)
SET @sql='CREATE TABLE TEMPORAL
        ( ID int IDENTITY, DATO varchar(100))'
EXEC (@sql)

SET @sql = 'SELECT * FROM TEMPORAL'
```

```
EXEC (@sql)
```

El procedimiento almacenado sp_executesql

Para ejecutar sql dinamico, se recomienda utilizar el procedimiento almacenado **sp_executesql**, en lugar de una instrucción **EXECUTE**.

- **sp_executesql** admite la sustitución de parámetros
- **sp_executesql** es más seguro y versátil que **EXECUTE**
- **sp_executesql** genera planes de ejecución con más probabilidades de que SQL Server los vuelva a utilizar, es más eficaz que **EXECUTE**.

El siguiente ejemplo muestra el uso (muy simple) de **sp_executesql**.

```
DECLARE @sql nvarchar(1000),
        @paramDefinition nvarchar(255),
        @paramValue char(3)

SET @paramDefinition = '@codPais char(3)'
SET @paramValue = 'ESP'
SET @sql = 'SELECT
                COD_PAIS,
                NOMBRE_PAIS,
                ACTIVO,
                FX_ALTA
            FROM
                PAISES
            WHERE COD_PAIS = @codPais'

EXEC sp_executesql @sql, @paramDefinition, @paramValue
```

🕮 DICCIONARIO DE DATOS

Un diccionario de datos es un conjunto de metadatos que contiene las características lógicas de los datos que se van a utilizar en el sistema que se programa, incluyendo nombre, descripción, alias, contenido y organización.
Para comprender mejor el significado de un diccionario de datos, puede considerarse su contenido como "datos acerca de los datos"; es decir, descripciones de todos los demás objetos (archivos, programas, informes, sinónimos...) existentes en el sistema. Un diccionario de datos almacena la totalidad de los diversos esquemas y especificaciones de archivos, así como sus ubicaciones. Si es completo incluye también información acerca de qué programas utilizan qué datos, y qué usuarios están interesados en unos u otros informes. Por lo general, el diccionario de datos está integrado en el sistema de información que describe.
Durante la creación de la base de datos, el SGBD crea estructuras de datos adicionales junto con los data files. Este diccionario de datos es un conjunto de tablas de solo lectura y vistas que registran, verifican y proveen información. El diccionario de datos describe la base de datos y sus objetos.

DICCIONARIO DE DATOS EN SQL Server

Una vista de esquema de información (information schema view) es uno de los diversos métodos que proporciona SQL Server para obtener metadatos.

Esta asignación entre convenciones de nomenclaturas se aplica a las siguientes vistas de SQL Server compatibles con ISO.

CHECK_CONSTRAINTS	REFERENTIAL_CONSTRAINTS
COLUMN_DOMAIN_USAGE	ROUTINES
COLUMN_PRIVILEGES	ROUTINE_COLUMNS
COLUMNS	SCHEMATA
CONSTRAINT_COLUMN_USAGE	TABLE_CONSTRAINTS
CONSTRAINT_TABLE_USAGE	TABLE_PRIVILEGES
DOMAIN_CONSTRAINTS	TABLES
DOMAINS	VIEW_COLUMN_USAGE
KEY_COLUMN_USAGE	VIEW_TABLE_USAGE
PARAMETERS	VIEWS

Además, algunas vistas contienen referencias a diferentes clases de datos, como los datos de caracteres o los datos binarios.

Al hacer referencia a las vistas de esquema de información, debe utilizar un nombre completo que incluya el nombre del esquema INFORMATION_SCHEMA. Por ejemplo:

```
...
IF EXISTS (SELECT domain_name FROM INFORMATION_SCHEMA.DOMAINS
        WHERE DOMAIN_SCHEMA = 'dbo' AND DOMAIN_NAME = 'socio_no')
...
```

🕮 CREACIÓN DE UNA BASE DE DATOS UTIIZANDO FICHEROS SCRIPTS

ARCHIVOS DE UNA BASE DE DATOS EN SQL SERVER

Se pueden distinguir tres tipos de archivos en SQL Server:

Principal. Contiene la información de inicio de la base de datos, así como un registro del resto de los archivos de la base de datos. Este archivo también puede contener datos y objetos de usuario. La extensión de estos archivos es .MDF.

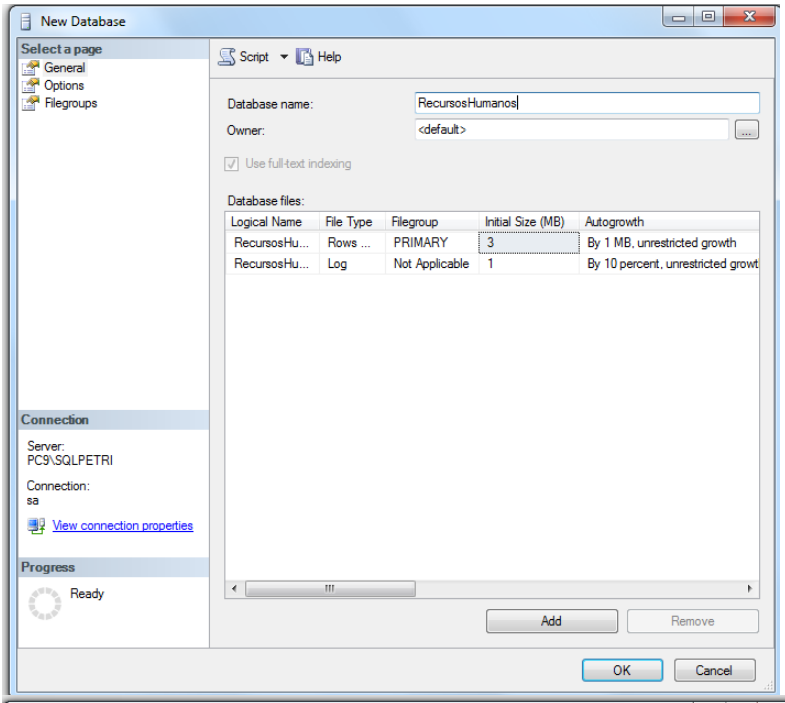
Secundario. Los archivos de datos secundarios solamente almacenan datos de usuario. Son opcionales y se pueden utilizar para distribuir datos en varios discos físicos, (o lógicos), colocando cada archivo en una unidad diferente. La extensión de estos archivos es .NDF.

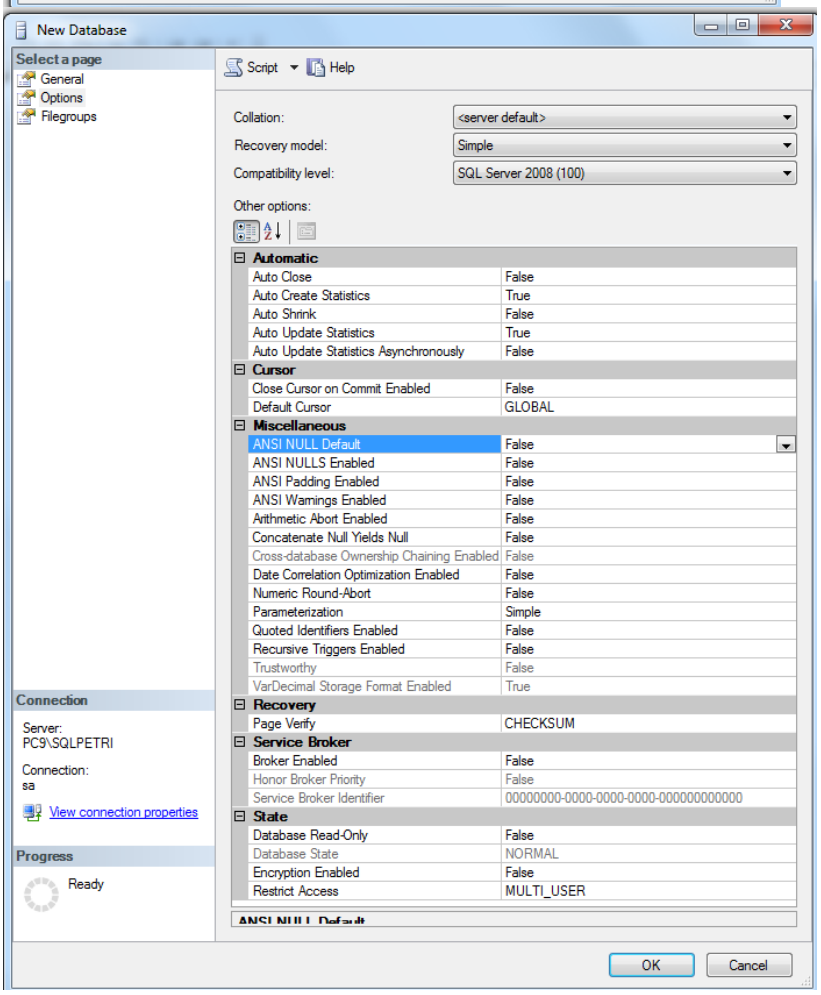
Registro de transacciones (ficheros LOG). Contienen la información que se utiliza para recuperar la base de datos. Cada base de datos debe tener como mínimo uno de estos archivos y su extensión es .LDF.

ELEMENTOS BÁSICOS PARA LA CREACIÓN DE BASES DE DATOS

FORMAS DE CREACIÓN. Para crear una base de datos hay dos posibles caminos:

(1) Utilizando herramientas básicas de SQL Server.





(2) Utilizando código Transact-SQL.

```
/*  Crea la base de datos  */
CREATE DATABASE Biblioteca ON PRIMARY
```

```
 (
  NAME=biblio_data,
  FILENAME='C:\Archivos de programa\Microsoft SQL Server\MSSQL\Data\biblio.mdf',
  SIZE=50MB,
  MAXSIZE=70MB,
  FILEGROWTH=1MB
  )
LOG ON
  (
  NAME=biblio_log,
  FILENAME='C:\Archivos de programa\Microsoft SQL Server\MSSQL\Data\biblio.ldf',
  SIZE=15MB,
  MAXSIZE=20MB,
  FILEGROWTH=1MB
  )

GO
```

En este caso, los parámetros de configuración del archivo (NAME, FILENAME, SIZE,….) no son los únicos a tener en cuenta. Utilizando la sentencia ALTER se podrán activar o desactivar los diferentes parámetros de configuración:

```
ALTER DATABASE [RecursosHumanos] SET COMPATIBILITY_LEVEL = 100
GO
ALTER DATABASE [RecursosHumanos] SET ANSI_NULL_DEFAULT OFF
GO
ALTER DATABASE [RecursosHumanos] SET ANSI_NULLS OFF
GO
ALTER DATABASE [RecursosHumanos] SET ANSI_PADDING OFF
GO
....
```

EJERCICIO EJEMPLO. FICHERO QUE CREA LA BASE DE DATOS "BIBLIOTECA" Y TODOS SUS OBJETOS.

Creación de la base de datos y de los tipos definidos por el usuario

```
Crea la base de datos Biblioteca. El tamaño se especifica en megabytes.
NOTA: esta secuencia de comandos está dentro del código asumiendo que la
ubicación de la base de datos es   'C:\Program Files (x86)\Microsoft SQL
Server\MSSQL.1\MSSQL\Data'.
Tendrá que cambiar esta ruta de acceso si desea crear la base de datos en una
unidad o ruta de acceso diferente.
*/

USE master

/*  Borra la base de datos Biblioteca si ya existe  */
IF DB_ID('Biblioteca') IS NOT NULL
BEGIN
    DROP DATABASE Biblioteca
END

/*  Crea la base de datos  */
CREATE DATABASE Biblioteca ON PRIMARY
 (
  NAME=biblioteca_data,
```

```
  FILENAME='C:\Program Files (x86)\Microsoft SQL
Server\MSSQL.1\MSSQL\Data\biblioteca.mdf',
  SIZE=50MB,
  MAXSIZE=70MB,
  FILEGROWTH=1MB
  )
LOG ON
  (
  NAME=biblioteca_log,
  FILENAME='C:\Program Files (x86)\Microsoft SQL
Server\MSSQL.1\MSSQL\Data\biblioteca.ldf',
  SIZE=15MB,
  MAXSIZE=20MB,
  FILEGROWTH=1MB
  )

GO
/*
Crea todos los tipos de datos definidos por el usuario para la base de datos
Biblioteca
*/
USE biblioteca

IF EXISTS (SELECT domain_name FROM information_schema.domains
           WHERE domain_schema = 'dbo' AND domain_name = 'socio_no')
    EXEC  sp_droptype  socio_no

IF EXISTS (SELECT domain_name FROM information_schema.domains
           WHERE domain_schema = 'dbo' AND domain_name = 'isbn')
     EXEC  sp_droptype  isbn

IF EXISTS (SELECT domain_name FROM information_schema.domains
           WHERE domain_schema = 'dbo' AND domain_name = 'titulo_no')
     EXEC  sp_droptype  titulo_no

IF EXISTS (SELECT domain_name FROM information_schema.domains
           WHERE domain_schema = 'dbo' AND domain_name = 'telefono')
    EXEC  sp_droptype  telefono

IF EXISTS (SELECT domain_name FROM information_schema.domains
           WHERE domain_schema = 'dbo' AND domain_name = 'yes_no')
     EXEC  sp_droptype  yes_no

IF EXISTS (SELECT domain_name FROM information_schema.domains
           WHERE domain_schema = 'dbo' AND domain_name = 'shortstring')
    EXEC  sp_droptype  shortstring

IF EXISTS (SELECT domain_name FROM information_schema.domains
           WHERE domain_schema = 'dbo' AND domain_name = 'normstring')
     EXEC  sp_droptype  normstring

IF EXISTS (SELECT domain_name FROM information_schema.domains
           WHERE domain_schema = 'dbo' AND domain_name = 'longstring')
     EXEC  sp_droptype  longstring

IF EXISTS (SELECT domain_name FROM information_schema.domains
           WHERE domain_schema = 'dbo' AND domain_name = 'remarks')
     EXEC  sp_droptype  remarks
GO

EXEC  sp_addtype  socio_no,    'smallint'
```

```
EXEC  sp_addtype  isbn,        'int'
EXEC  sp_addtype  titulo_no,   'int'
EXEC  sp_addtype  telefono,    'char(13)', 'NULL'
EXEC  sp_addtype  yes_no,      'char(1)'
EXEC  sp_addtype  shortstring, 'varchar(15)'
EXEC  sp_addtype  normstring,  'varchar(31)'
EXEC  sp_addtype  longstring,  'varchar(63)'
EXEC  sp_addtype  remarks,     'varchar(255)'
GO

/* Muestra el resultado */

SELECT domain_name
   FROM information_schema.domains
   ORDER BY domain_name
GO
```

CREAMOS LAS TABLAS

```
/*
Este archivo de secuencias de comandos crea todas las tablas para la base de
datos.
Hay un total de  5 tablas.
Primero se eliminan las tablas para volver a crearlas.
*/

USE biblioteca

IF OBJECT_ID('dbo.socios') IS NOT NULL
    DROP TABLE dbo.socios

IF OBJECT_ID('dbo.adultos') IS NOT NULL
    DROP TABLE dbo.adultos

IF OBJECT_ID('dbo.juvenil') IS NOT NULL
    DROP TABLE dbo.juvenil

IF OBJECT_ID('dbo.libros') IS NOT NULL
    DROP TABLE dbo.libros

IF OBJECT_ID('dbo.reservas') IS NOT NULL
    DROP TABLE dbo.reservas

GO

CREATE TABLE socios
  (
     socio_no         socio_no    NOT NULL
  ,  apellidos        shortstring  NOT NULL
  ,  nombre           shortstring  NOT NULL
  )

CREATE TABLE adultos
  (
     socio_no         socio_no     NOT NULL
  ,  calle            shortstring  NOT NULL
  ,  ciudad           shortstring  NOT NULL
  ,  telefono         varchar(10)      NULL
  ,  expr_fecha       datetime     NOT NULL
  )
```

```
CREATE TABLE juvenil
  (
     socio_no          socio_no    NOT NULL
  ,  adult_socio_no    socio_no    NOT NULL
  ,  cumpleaños        datetime     NOT NULL
  )

CREATE TABLE libros
  (
     isbn              isbn         NOT NULL
  ,  titulo_no         titulo_no    NOT NULL
  ,  titulo            longstring   NOT NULL
  ,  autor_prin        normstring   NOT NULL
  ,  synopsis          text             NULL
  )

CREATE TABLE reservas
  (
     isbn              isbn         NOT NULL
  ,  socio_no          socio_no    NOT NULL
  ,  log_fecha         datetime         NULL
  ,  remarks           remarks          NULL
  )

GO

/* Muestra el resultado */

SELECT table_name
  FROM information_schema.tables
  WHERE table_name IN (  'socios'
                       , 'adultos'
                       , 'juvenil'
                       , 'libros'
                       , 'reservas'
                    )
GO

BACKUP LOG biblioteca WITH TRUNCATE_ONLY
GO
```

CREAMOS LAS CLAVES PRIMARIAS

```
/*
Agrega una restricción a la clave principal para el resto de las tablas de la
base de datos biblioteca incluyendo el título y las tablas de elementos donde las
restricciones PK se agregaron anteriormente en la práctica.
Si se vuelve a ejecutar (y ya existe la restricción), la elimina.
Utiliza la base de datos biblioteca y establece NOCOUNT a ON para eliminar el
mensaje que indica el número de filas afectadas.
*/

USE biblioteca
SET NOCOUNT ON
GO

/* Comprueba si existen restricciones para la clave principal. Si hay las elimina
*/

IF EXISTS
   (SELECT * FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
```

```
    where CONSTRAINT_SCHEMA = 'dbo'
      AND CONSTRAINT_NAME = 'socio_ident'
      AND CONSTRAINT_TYPE = 'PRIMARY KEY')
  ALTER TABLE socios
      DROP CONSTRAINT socio_ident

IF EXISTS
   (SELECT * FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
    where CONSTRAINT_SCHEMA = 'dbo'
      AND CONSTRAINT_NAME = 'adulto_ident'
      AND CONSTRAINT_TYPE = 'PRIMARY KEY')
  ALTER TABLE adultos
     DROP CONSTRAINT adulto_ident

IF EXISTS
   (SELECT * FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
    where CONSTRAINT_SCHEMA = 'dbo'
      AND CONSTRAINT_NAME = 'juvenil_ident'
      AND CONSTRAINT_TYPE = 'PRIMARY KEY')
  ALTER TABLE juvenil
     DROP CONSTRAINT juvenil_ident

IF EXISTS
   (SELECT * FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
    where CONSTRAINT_SCHEMA = 'dbo'
      AND CONSTRAINT_NAME = 'isbn_ident'
      AND CONSTRAINT_TYPE = 'PRIMARY KEY')
  ALTER TABLE libros
     DROP CONSTRAINT isbn_ident

IF EXISTS
   (SELECT * FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
    where CONSTRAINT_SCHEMA = 'dbo'
      AND CONSTRAINT_NAME = 'reservas_ident'
      AND CONSTRAINT_TYPE = 'PRIMARY KEY')
  ALTER TABLE reservas
     DROP CONSTRAINT reservas_ident

GO

/* Crea todas las restricciones PK */

ALTER TABLE socios
   ADD CONSTRAINT socio_ident PRIMARY KEY NONCLUSTERED (socio_no)

ALTER TABLE adultos
   ADD CONSTRAINT adulto_ident      PRIMARY KEY  CLUSTERED (socio_no)

ALTER TABLE juvenil
   ADD CONSTRAINT juvenil_ident   PRIMARY KEY NONCLUSTERED (socio_no)

ALTER TABLE libros
   ADD CONSTRAINT isbn_ident        PRIMARY KEY NONCLUSTERED (isbn)

ALTER TABLE reservas
   ADD CONSTRAINT reserva_ident PRIMARY KEY NONCLUSTERED (socio_no, isbn)

GO

/* Restablece el valor de la opción NOCOUNT */
```

```
SET NOCOUNT OFF
GO

/* Selecciona todas las restricciones PK que existen en la base de datos biblio
*/

PRINT 'PRIMARY KEYS CREATED:'
SELECT TABLE_NAME, CONSTRAINT_NAME
  FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
  WHERE CONSTRAINT_TYPE = 'PRIMARY KEY'
```

CREAMOS LAS CLAVES EXTERNAS

```
/*
Agrega las restricciones de clave externa a la base de datos Biblioteca.
Si se vuelve a ejecutar (y ya existe la restricción), primero la elimina.
Para los propósitos de este ejercicio práctico, la opción WITH NOCHECK se utiliza
para mejorar el rendimiento de esta secuencia de comandos.
Establece la base de datos Biblioteca para que sólo la utilice el propietario de
la base de datos.
*/

USE master
GO

/*
Utiliza la base de datos Biblio y establece NOCOUNT a ON para eliminar
el mensaje que indica el número de filas afectadas.
*/

USE biblioteca
SET NOCOUNT ON
GO

IF EXISTS
   (SELECT * FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
    where CONSTRAINT_SCHEMA = 'dbo'
      AND CONSTRAINT_NAME = 'adulto_socio_link'
      AND CONSTRAINT_TYPE = 'FOREIGN KEY')
  ALTER TABLE adultos
     DROP CONSTRAINT adulto_socio_link
GO

IF EXISTS
   (SELECT * FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
    where CONSTRAINT_SCHEMA = 'dbo'
      AND CONSTRAINT_NAME = 'juvenil_socio_link'
      AND CONSTRAINT_TYPE = 'FOREIGN KEY')
  ALTER TABLE juvenil
     DROP CONSTRAINT juvenil_socio_link
GO

IF EXISTS
   (SELECT * FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
    where CONSTRAINT_SCHEMA = 'dbo'
      AND CONSTRAINT_NAME = 'juvenil_adulto_link'
      AND CONSTRAINT_TYPE = 'FOREIGN KEY')
  ALTER TABLE juvenil
     DROP CONSTRAINT juvenil_adulto_link
GO

IF EXISTS
```

```
   (SELECT * FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
    where CONSTRAINT_SCHEMA = 'dbo'
      AND CONSTRAINT_NAME = 'reserva_socio_link'
      AND CONSTRAINT_TYPE = 'FOREIGN KEY')
  ALTER TABLE reservas
     DROP CONSTRAINT reserva_socio_link
GO

IF EXISTS
   (SELECT * FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
    where CONSTRAINT_SCHEMA = 'dbo'
      AND CONSTRAINT_NAME = 'reserva_isbn_link'
      AND CONSTRAINT_TYPE = 'FOREIGN KEY')
  ALTER TABLE reservas
     DROP CONSTRAINT reserva_isbn_link
GO

/* Crea las restricciones de la clave externa para la tabla Adultos */

ALTER TABLE adultos WITH NOCHECK
   ADD CONSTRAINT adulto_socio_link FOREIGN KEY (socio_no) REFERENCES
socios(socio_no)
GO

/* Crea las restricciones de la clave externa para la tabla Juvenile */

ALTER TABLE juvenil WITH NOCHECK
   ADD CONSTRAINT juvenil_socio_link FOREIGN KEY (socio_no) REFERENCES
socios(socio_no)

ALTER TABLE juvenil WITH NOCHECK
   ADD CONSTRAINT juvenil_adulto_link  FOREIGN KEY (adult_socio_no) REFERENCES
adultos(socio_no)
GO

/* Crea las restricciones de la clave externa para la tabla Reservas */

ALTER TABLE reservas WITH NOCHECK
  ADD CONSTRAINT reserva_isbn_link   FOREIGN KEY (isbn) REFERENCES libros(isbn)

ALTER TABLE reservas WITH NOCHECK
  ADD CONSTRAINT reserva_socio_link FOREIGN KEY (socio_no) REFERENCES
socios(socio_no)
GO

/* Restablece la opción de la base de datos y NOCOUNT */
USE master
GO

USE Biblioteca
SET NOCOUNT ON
GO
/* Selecciona todas las restricciones FK que existen en la base de datos Library
*/

PRINT 'FOREIGN KEYS CREATED:'
SELECT TABLE_NAME, CONSTRAINT_NAME
  FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
  WHERE CONSTRAINT_TYPE = 'FOREIGN KEY'
```