|  |
| --- |
| **LOGO UTN FRC SISTEMAS**    **CÁTEDRA PROGRAMACIÓN DE APLICACIONES VISUALES I** |

## Programando transacciones con C#

El objetivo de este documento es que el lector diseñe y desarrolle una aplicación que inserta bajo transacción un registro y su histórico. Podemos mencionar entonces como objetivos específicos:

* Programar en C# usando transacciones sql.
* Desarrollar aplicaciones asegurando la consistencia de la base de datos.
* Identificar transacciones en un requerimiento.

## Incorporando el concepto de transacción.

Creamos un nuevo proyecto de tipo “Aplicación de Windows Forms”:

Una TRANSACCIÓN es una unidad de trabajo compuesta por diversas tareas, cuyo resultado final debe ser que se ejecuten todas juntas o ninguna de ellas. Las tareas son sentencias sql para manipular la base de datos modificando sus datos (insert, delete, update).

Ejemplos típicos de transacciones son:

Guardar una factura:

Insertar encabezado.

Obtener identificador de la fila de encabezado.

Insertar tantas filas de detalle como items contenga la factura.

Guardar un registro nuevo y generar un registro histórico.

Actualizar un registro y actualizar su histórico.

Transferir dinero entre una cuenta y otra:

Disminuir el saldo en la cuenta origen.

Aumentar el saldo en la cuenta destino.

## Sentencias slq para manejo de transacción

Una transacción es una unidad única de trabajo. Si una transacción tiene éxito, todas las modificaciones de los datos realizadas durante la transacción se confirman y se convierten en una parte permanente de la base de datos. Si una transacción encuentra errores y debe cancelarse o revertirse, se borran todas las modificaciones de los datos.

SQL Server funciona en los modos de transacción siguientes:

* Transacciones de confirmación automática
* Cada instrucción individual es una transacción.

Transacciones explícitas: Cada transacción se inicia explícitamente con la instrucción BEGIN TRANSACTION y se termina explícitamente con una instrucción COMMIT o ROLLBACK.

Transacciones implícitas: Se inicia implícitamente una nueva transacción cuando se ha completado la anterior, pero cada transacción se completa explícitamente con una instrucción COMMIT o ROLLBACK.

Una manera de engañar al motor de bases de datos es ordenarle ejecutar un conjunto de sentencias separadas por punto y coma. El motor interpreta que se trata de una sola orden y ejecuta todas las sentencias sql o ninguna.

Pero la forma correcta de trabajar con transacciones es mediante el uso de los comandos apropiados: BEGIN TRANSACTION, COMMIT, ROLLBACK.

Una Transacción está delimitada por instrucciones de inicio transacción y fin transacción (la transacción consiste en todas las operaciones que se ejecutan entre inicio transacción y fin transacción).

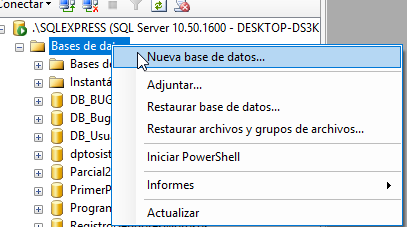
Este concepto vio la luz debido a que varias operaciones sucesivas pueden modificar el resultado de operaciones anteriores. En esos casos, si alguna operación produce un error, el estado resultante puede ser indeterminado.

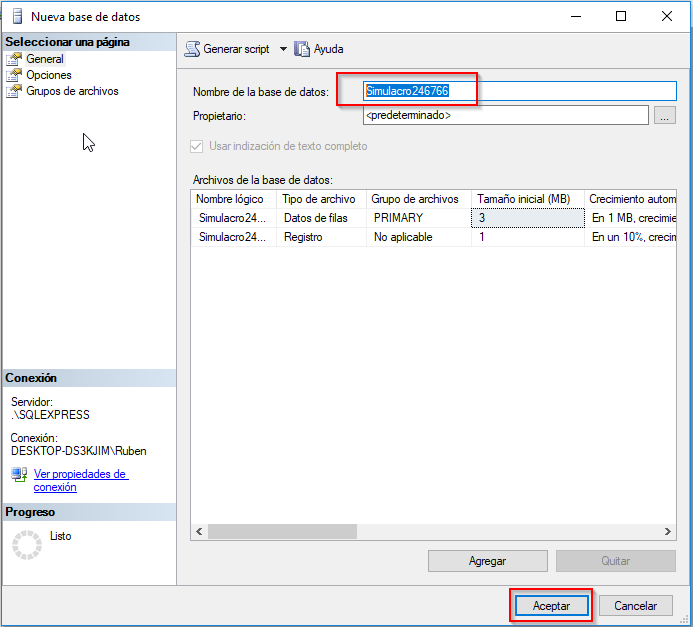
Para solucionar este problema, las transacciones agrupan una serie de operaciones de manera que es posible garantizar la integridad del resultado final. O todas las operaciones se ejecutan con éxito y se confirman (se escriben en la base de datos), o toda la transacción se considera no realizada. La acción de cancelar una transacción se denomina deshacer la transacción. Deshacer una transacción permite anular los cambios y recuperar el estado de la base de datos previo a la transacción.

Por ejemplo, en una transacción bancaria automatizada, si un banco transfiere dinero desde la cuenta A a la cuenta B, la retirada de fondos de A y el depósito en B deben producirse con éxito para procesar los fondos correctamente, de lo contrario la transacción entera debe cancelarse.

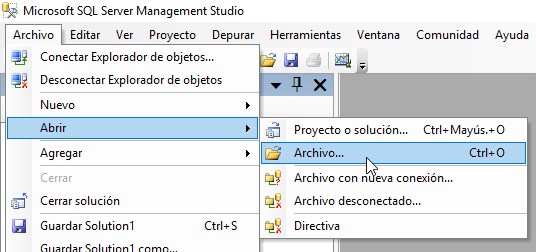
## Programando una transacción

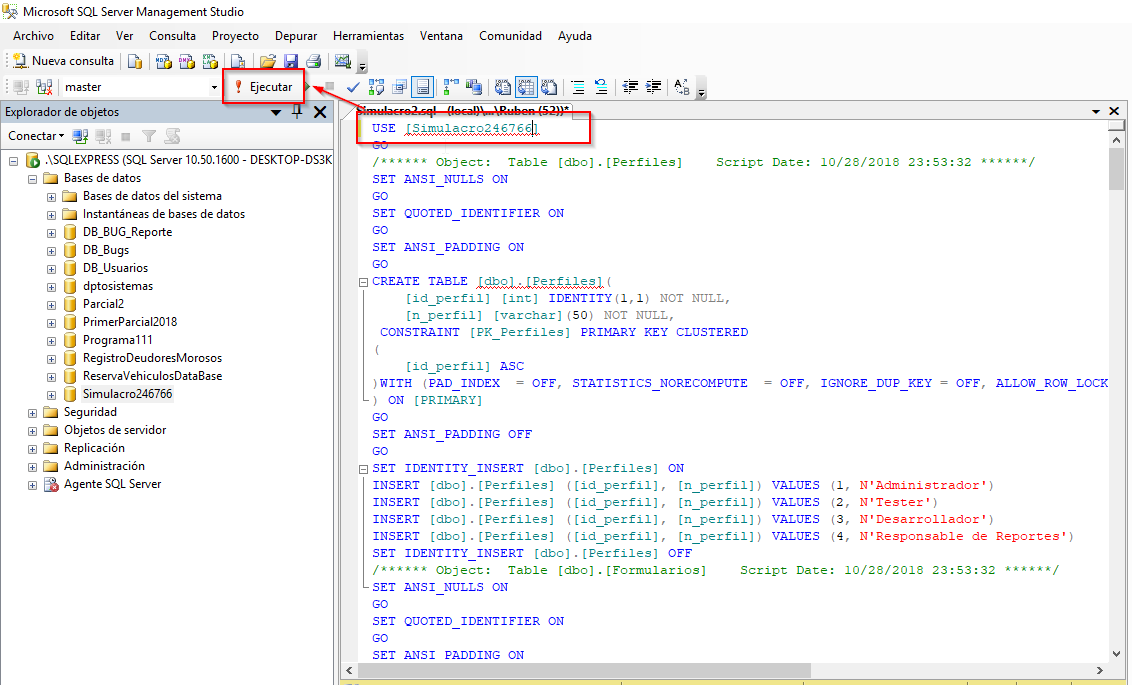
* 1. Crear base de datos Simulacro2+[legajo] (NO usar guiones).



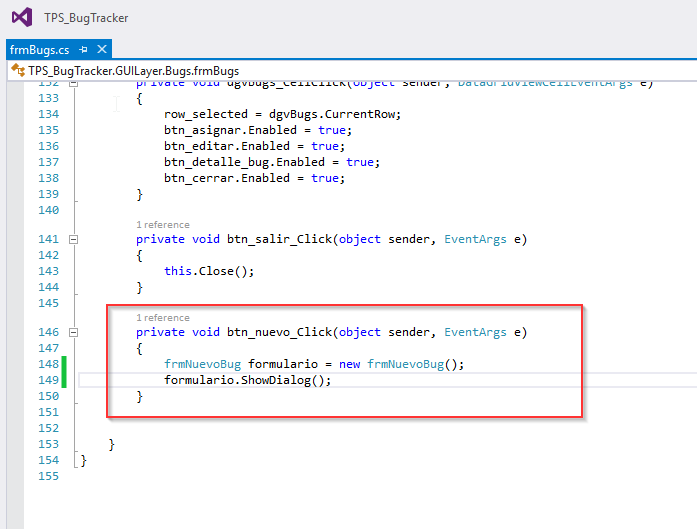


* 1. Ejecutar el script **Simulacro2.sql** para crear las tablas de la base de datos creada anteriormente (se encuentra en la carpeta dp), usar el nombre de la base de datos creada anteriormente:

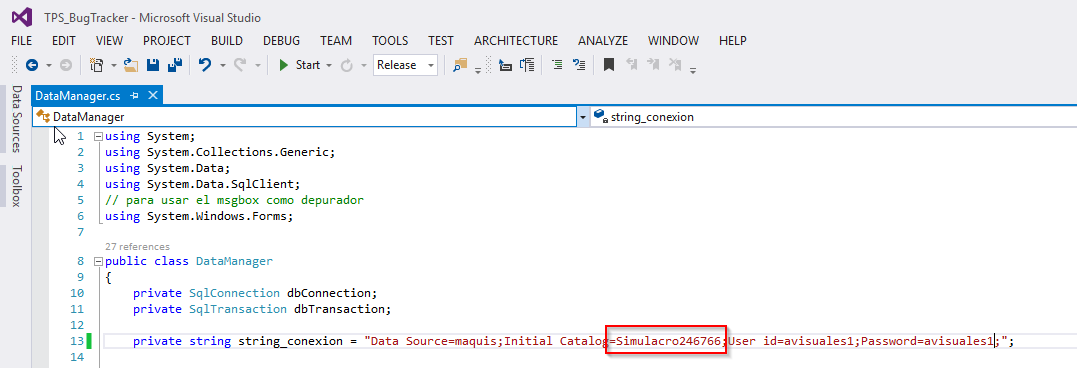




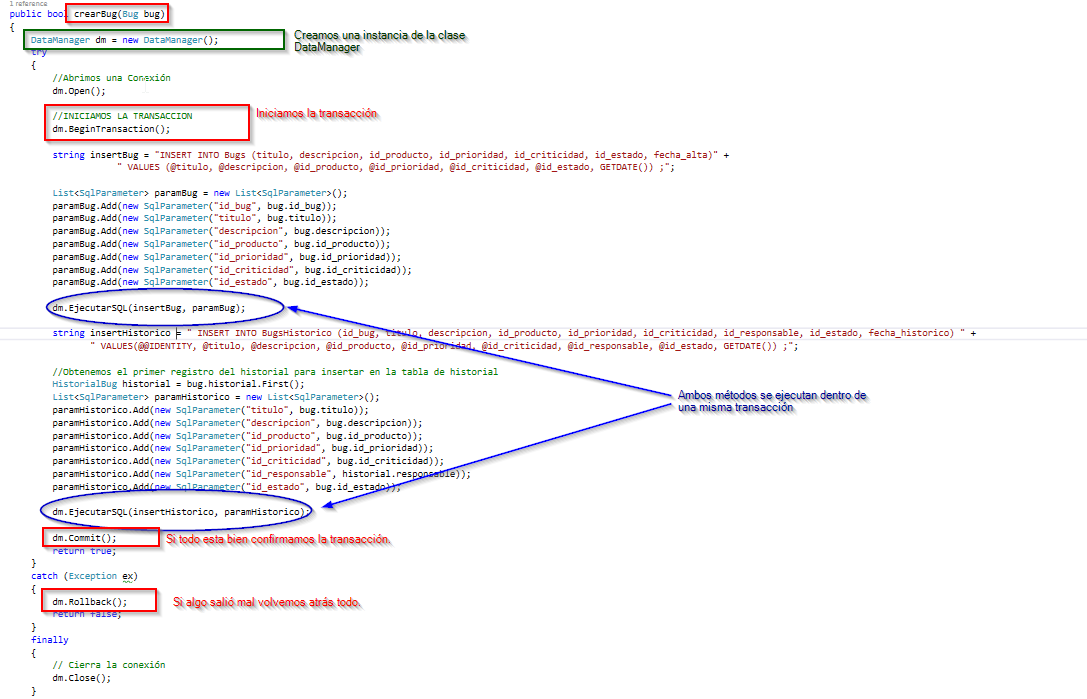
* 1. En el formulario **frmBugs** agregamos el evento click al botón Editar para que abra el formulario **frmNuevoBug**:



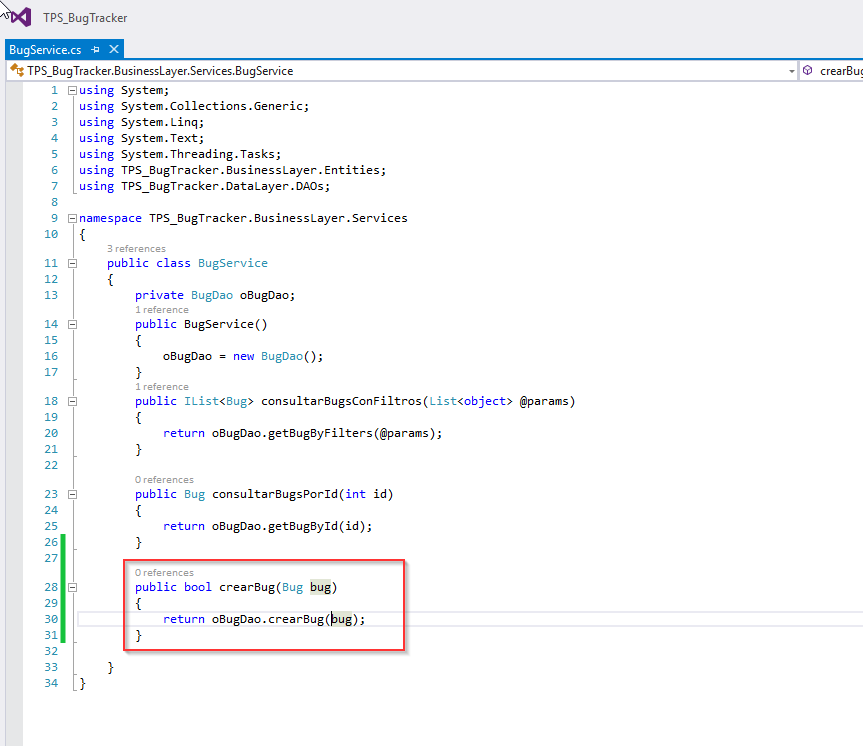
* 1. En clase **DataManager** modificar la variable **string\_conexion** con el nombre de la base de datos creada en el punto 1:



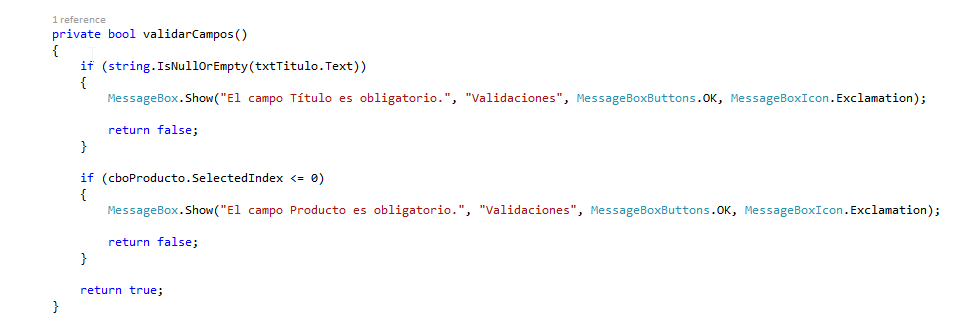
* 1. En la clase **BugDao** agregamos el método **crearBug** que ejecute el INSERT para dar de alta un Bug y un BugsHistorico a la base de datos:



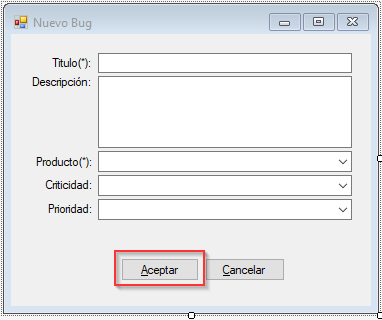
* 1. En la clase **BugService** agregamos el método **crearBug** que a su vez invoque al método BugDao.crearBug():

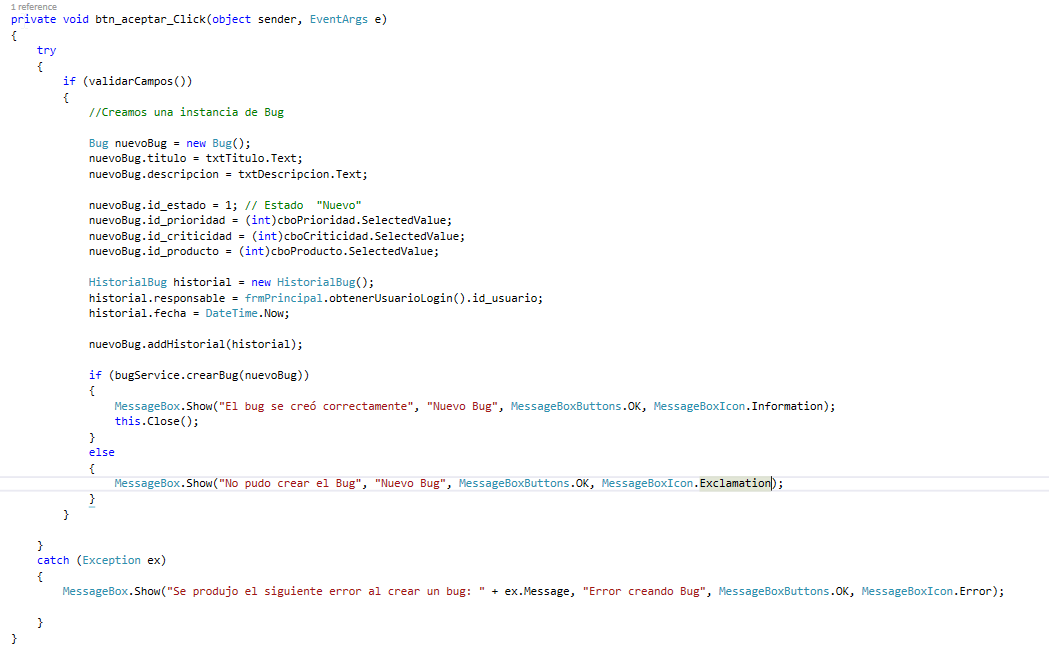


* 1. En el formulario **frmNuevoBug** programamos un método **validarCampos()** para validar los campos obligatorios del formulario:



* 1. En el formulario **frmNuevoBug** programamos el evento click del botón **Aceptar**, donde validamos los campos obligatorios, creamos una instancia de la clase **Bug** e invocamos el método BugService.crearBug():





* 1. En el formulario **frmNuevoBug** programamos el evento click del botón **Cancelar**, dónde debemos cerrar el formulario:

