

# Enunciado:

---

Tema: **Bloqueo optimista.**

Antes de encarar el desarrollo deberemos exponer un poco de teoría que le de sustento.

Existen dos tipos de bloqueo de entidades:

- **Bloqueo optimista**, donde una entidad se bloquea impidiendo su persistencia, sólo cuando se confirman cambios en la base de datos.
- **Bloqueo pesimista**, donde una entidad permanece bloqueada durante todo el tiempo que dure su edición, en otras palabras la entidad es editada en forma exclusiva impidiendo la existencia de otra edición simultánea.

Aquí trataremos el bloqueo **optimista**.

**Bloqueo optimista:** Es el tipo de bloqueo al que también se denomina control de concurrencia optimista, es un método de control de concurrencia utilizado en bases de datos relacionales que no utiliza bloqueo exclusivo de registros. El bloqueo optimista permite que varios usuarios intenten actualizar el mismo registro sin informar a los usuarios de que otros también están intentando actualizarlo. El permiso de persistir el registro es evaluado sólo cuando se intenta guardar los datos.

Existen varias técnicas para establecer si se ha producido una infracción de la concurrencia optimista. Una sería comparar el valor de cada campo y validar si alguno ha cambiado, y otra consiste en incluir un campo de marca de modificación del registro, esta última técnica será la que utilizaremos para este desarrollo, razón por la que se creó el campo *Producto.Bloqueo* cuya responsabilidad es almacenar el valor de la marca. Cada vez que un registro de la tabla "Producto" es modificado debe darse un nuevo valor al campo *Producto.Bloqueo*, valor dado mediante el siguiente código C#: `"Guid.NewGuid().ToString();"`

La **Figura 1** consta de cuatro columnas, la del tiempo que señala cronológicamente como suceden las acciones realizadas por los usuarios, la de las acciones que realiza el "Usuario 1", la que indica los valores de los campos del registro en la base de datos y por último la de las acciones que realiza el "Usuario 2".

La **Figura 1**, muestra en función del tiempo, como dos usuarios intentan modificar un mismo producto o entidad, logrando el "Usuario 1" persistir los cambios en forma exitosa, cosa que no logra hacer el "Usuario 2" ya que al intentar persistir sus cambios es bloqueado por el "Usuario 1".



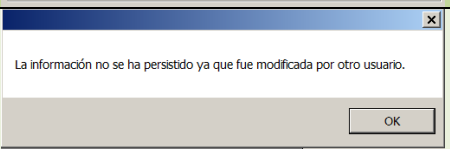
**T0:** Ambos usuarios editan un producto existente en la base de datos, de modo que los valores de los registros coinciden tanto en sus aplicaciones como en la base de datos, hasta el momento ninguno de los usuarios es bloqueado.

**T1:** El "Usuario 1" cambia el valor del *Producto.Precio* y persiste los datos en la base de datos. Observemos como ha cambiado el valor de la propiedad *Producto.Bloqueo* que es suministrado en

forma automática cada vez que se realiza un cambio de un registro en la base de datos, también notemos que el “Usuario 2” está viendo información del producto desactualizada ya que la misma ha cambiado sin que la aplicación le de aviso alguno.

**T2:** El “Usuario 2” cambia el valor del *Producto.Precio* editando el producto en su aplicación, hasta el momento no es bloqueado.

**T3:** El “Usuario 2” intenta persistir el cambio pero su aplicación bloquea esta acción impidiéndosela y mostrándole un mensaje de aviso que le indica que existió una modificación previa a la suya realizada por otro usuario. Evidentemente el “Usuario 2” no logra persistir el producto, teniendo que actualizar la información leyéndola de la base de datos y perdiendo de este modo los cambios que había realizado en la propiedad “*Producto.Precio*”.

Figura 1			
Tiempo	Usuario 1	Base de datos	Usuario 2
T0	Producto: Id: 577 Marca: Sprite Precio: 2222 Bloqueo: 444e46bd-4a8d-44fd-aeed-4e8e8ec13b32	Ídem Usuario 1 y Usuario 2	Producto: Id: 577 Marca: Sprite Precio: 2222 Bloqueo: 444e46bd-4a8d-44fd-aeed-4e8e8ec13b32
T1	Producto: Id: 577 Marca: Sprite Precio: 111 Bloqueo: 57b23a6f-9e06-4c74-a062-65e1f02e6be7 	Ídem Usuario 1	Producto: Id: 577 Marca: Sprite Precio: 2222 Bloqueo: 444e46bd-4a8d-44fd-aeed-4e8e8ec13b32
T2	Producto: Id: 577 Marca: Sprite Precio: 111 Bloqueo: 57b23a6f-9e06-4c74-a062-65e1f02e6be7	Ídem Usuario 1	Producto: Id: 577 Marca: Sprite Precio: 333 Bloqueo: 444e46bd-4a8d-44fd-aeed-4e8e8ec13b32 
T3	Producto: Id: 577 Marca: Sprite Precio: 111 Bloqueo: 57b23a6f-9e06-4c74-a062-65e1f02e6be7	Ídem Usuario 1	

Solo a los fines de aumentar la comprensión se ha dejado visible en la interfaz de usuario el valor de la propiedad “*Producto.Bloqueo*” cosa que en una aplicación productiva no pasaría y se manejaría este valor en memoria y en un perfecto desconocimiento del usuario, lo que se suele decir en la jerga es que: “*el valor de Producto.Bloqueo es transparente al usuario*”.

La **Figura 2**, muestra el método “`Persistidor.Update`”, podemos destacar que se evalúa una precondición del método, verificando la infracción por bloqueo en la línea (78) y lanzando una excepción en caso positivo, impidiendo que se ejecute el “Update real” definido por el método “`UpdateSinBloqueo`”, cuya invocación se encuentra en la línea (80).

Figura 2

```

76 private bool Update(Producto producto)
77 {
78     this.BloqueoPrecondition(producto);
79
80     bool success = this.UpdateSinBloqueo(producto);
81
82     return success;
83 }

```

En la **Figura 3**, podemos ver la precondición responsable de evaluar la infracción por bloqueo, es tan simple como comparar el valor actual de la marca de bloqueo que posee la aplicación en memoria contra el valor que se encuentra en la base de datos, si son distintos es porque otro usuario modificó el producto por lo cual se lanza una excepción del tipo “`BloqueoException`” impidiendo de esta forma la operación de persistencia, en caso en que las marcas de bloque sean iguales la excepción no es disparada y la persistencia es llevada a cabo con éxito.

Figura 3

```

89 private void BloqueoPrecondition(Producto producto)
90 {
91     Producto productoEnBaseDeDatos = this.GetByid(producto.Id.Value);
92     if (producto.Bloqueo != productoEnBaseDeDatos.Bloqueo)
93         throw new BloqueoException();
94 }

```

En la **Figura 4**, podemos ver la definición del tipo de excepción lanzada para indicar una infracción por bloqueo en la línea 93, del código definido en de la **Figura 3**.

Figura 4

```

/// <summary>
/// Representa una infracción por bloqueo de registro.
/// </summary>
public class BloqueoException : Exception { }

```