

**Instituto Tecnológico de Costa Rica
Campus Tecnológico Central Cartago
Escuela De Ingeniería En Computación
Bases de datos I
I-Semestre 2023
Jueves 27 de abril
Nombre: Leonardo Céspedes
Número de Carné: 2022080602
Nombre: Frankmin Feng
Número de Carné: 2022089248**

Utilizando los stored procedures existentes o bien creando nuevos pero que no sean forzados, si no que deben ser stored procedures que realmente van a ser necesarios para que el sistema funcione, proceda a averiguar bajo que situaciones particulares en el orden de ejecución de cuales quiera dos stored procedures transaccionales, podría producirse alguno de los siguientes problemas:

a) dirty read b) lost update c) phantom d) deadlock

Consideremos este stored procedure:

```
1 CREATE PROCEDURE [dbo].[registerSales]
2     @client INT,
3     @product INT,
4     @seller INT,
5     @totalPrice DECIMAL(12,2),
6     @paymentType INT,
7     @contract INT,
8     @quantity INT
9 AS
10 BEGIN
11     -- Escenario: Actualizaciones concurrentes
12     -- Si varias instancias de este procedimiento almacenado se ejecutan simultáneamente y actualizan el mismo @producto,
13     -- existe la posibilidad de que se pierdan actualizaciones cuando una actualización sobrescribe los cambios realizados por otra instancia.
14     DECLARE @sellerPercentage decimal(5,2);
15     DECLARE @producerPercentage decimal(5,2);
16     DECLARE @collectorPercentage decimal(5,2);
17     DECLARE @availableStock int;
18
19     set @availableStock = (Select top 1 inventoryProduct.quantity FROM inventoryProduct WHERE productId = @product);
20
21     IF(@availableStock >= @quantity)
22     BEGIN
23         UPDATE inventoryProduct SET quantity = quantity - @quantity WHERE productId = @product;
24
25         INSERT INTO [dbo].[sales]([clientId], [productId], [sellerId], [totalPrice], [posttime], [checksum], [paymentTypeId], [contractId], [quantity]) VALUES
26             (@client, @product, @seller, @totalPrice, GETDATE(), NULL, @paymentType, @contract, @quantity);
27
28
29         SET @sellerPercentage = (SELECT participantPercentage FROM contractParticipants
30             INNER JOIN contracts ON contractParticipants.contractId = contracts.contractId
31             WHERE contractParticipants.contractId = @contract AND
32                 contractParticipants.participantId = @seller);
33
34         -- Update seller's balance
35         UPDATE participants
36             SET participants.balance = participants.balance + @totalPrice * (@sellerPercentage/100)
37             where participants.participantId = @seller;
38
39         -- Update producers balance
40         UPDATE producers
41             SET producers.balance = producers.balance + @totalPrice * (contractProducers.producerPercentage / 100)
42             FROM contracts
43             INNER JOIN contractProducers ON contracts.contractId = contractProducers.contractId
44             INNER JOIN producers ON contractProducers.producerId = producers.producerId
45             WHERE contracts.contractId = @contract;
46
47         -- Update collectors balance
48         UPDATE collectors
49             SET collectors.balance = collectors.balance + @totalPrice * (contractCollectors.collectorPercentage / 100)
50             FROM contracts
51             INNER JOIN contractCollectors ON contracts.contractId = contractCollectors.contractId
52             INNER JOIN collectors ON contractCollectors.collectorId = collectors.collectorId
53             WHERE contracts.contractId = @contract;
54     END;
55 END;
```

Dirty Read: Un *dirty read* ocurre cuando una transacción lee datos no confirmados de otra transacción que aún no se ha confirmado.

¿Cuándo podría ocurrir?

Si otra transacción modifica la tabla *inventoryProduct* después de hacerle read al *@availableStock*, pero antes de realizar el *UPDATE inventoryProduct*.

Lost Update: Una *lost update* se produce cuando dos transacciones intentan modificar los mismos datos al mismo tiempo y una sobrescribe los cambios realizados por la otra, lo que provoca la pérdida de las modificaciones de la segunda transacción.

¿Cuándo podría ocurrir?

Si dos o más instancias del stored procedure se están ejecutando al mismo tiempo e intentan actualizar el *inventoryProduct* del mismo *@product*, existe la posibilidad de que se den lost updates.

Phantom: Se produce un *phantom* cuando una transacción lee un conjunto de filas que cumplen una determinada condición, pero otra transacción inserta o elimina filas que habrían cumplido la condición, lo que hace que la primera transacción vea un conjunto diferente de filas si tuviera que volver a ejecutar la transacción. misma consulta.

¿Cuándo podría ocurrir?

Si otra transacción inserta o elimina registros de *sales* entre el SELECT y el INSERT INTO, los registros retornados por el select pueden ser diferentes.

Deadlock: Un *deadlock* ocurre cuando dos o más transacciones esperan indefinidamente para liberar recursos, lo que hace que el sistema se detenga.

¿Cuándo podría ocurrir?

Este problema no se puede ver tan claramente en el script. Se podría dar cuando ocurren múltiples transacciones al mismo tiempo y existe una dependencia cíclica en el orden de *locking*.

Query mejorado para evitar Lost Updates

```
1 CREATE PROCEDURE [dbo].[registerSales]
2     @client INT,
3     @product INT,
4     @seller INT,
5     @totalPrice DECIMAL(12,2),
6     @paymentType INT,
7     @contract INT,
8     @quantity INT
9 AS
10 BEGIN
11     -- 'SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;' permite que las transacciones concurrentes se serialicen adecuadamente
12     SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
13     -- Se añade un BEGIN TRANSACTION y COMMIT para definir el inicio y fin de la transacción explícitamente y que el sp se trate como una sola unidad atómica en la ejecución
14     BEGIN TRANSACTION;
15
16     DECLARE @sellerPercentage decimal(5,2);
17     DECLARE @producerPercentage decimal(5,2);
18     DECLARE @collectorPercentage decimal(5,2);
19     DECLARE @availableStock int;
20
21     -- UPDLOCK añade un lock exclusivo a la fila que se está leyendo, lo cual evita que otras transacciones concurrentes puedan leerla o modificarla antes de que se termine la transacción actual
22     SET @availableStock = (SELECT TOP 1 inventoryProduct.quantity FROM inventoryProduct WITH (UPDLOCK) WHERE productId = @product);
23
24     IF(@availableStock >= @quantity)
25     BEGIN
26         UPDATE inventoryProduct SET quantity = quantity - @quantity WHERE productId = @product;
27
28         INSERT INTO [dbo].[sales]([clientId], [productId], [sellerId], [totalPrice], [posttime], [checksum], [paymentTypeId], [contractId], [quantity])
29         VALUES (@client, @product, @seller, @totalPrice, GETDATE(), NULL, @paymentType, @contract, @quantity);
30
31         SET @sellerPercentage = (SELECT participantPercentage FROM contractParticipants
32             INNER JOIN contracts ON contractParticipants.contractId = contracts.contractId
33             WHERE contractParticipants.contractId = @contract AND
34             contractParticipants.participantId = @seller);
35
36         -- Update seller's balance
37         UPDATE participants
38         SET participants.balance = participants.balance + @totalPrice * (@sellerPercentage/100)
39         WHERE participants.participantId = @seller;
40
41         -- Update producers balance
42         UPDATE producers
43         SET producers.balance = producers.balance + @totalPrice * (contractProducers.producerPercentage / 100)
44         FROM contracts
45         INNER JOIN contractProducers ON contracts.contractId = contractProducers.contractId
46         INNER JOIN producers ON contractProducers.producerId = producers.producerId
47         WHERE contracts.contractId = @contract;
48
49         -- Update collectors balance
50         UPDATE collectors
51         SET collectors.balance = collectors.balance + @totalPrice * (contractCollectors.collectorPercentage / 100)
52         FROM contracts
53         INNER JOIN contractCollectors ON contracts.contractId = contractCollectors.contractId
54         INNER JOIN collectors ON contractCollectors.collectorId = collectors.collectorId
55         WHERE contracts.contractId = @contract;
56     END;
57
58     COMMIT;
59 END;
```

Pruebas con el query original y el mejorado (Sales Lost Update)

Original:

	InventoryLogId	posttime	quantity	productId	operationTypeid
1	1	2023-05-01 10:00:00.000	20.00	1	1
2	2	2023-05-02 12:00:00.000	30.00	2	1
3	3	2023-05-03 15:30:00.000	40.00	3	1
4	4	2023-05-04 09:15:00.000	20.00	4	1
5	5	2023-05-05 14:45:00.000	60.00	5	1
6	6	2023-05-06 11:30:00.000	25.00	6	1
7	7	2023-05-07 13:20:00.000	35.00	7	1
8	8	2023-05-08 16:10:00.000	45.00	8	1
9	9	2023-05-09 09:45:00.000	55.00	9	1
10	10	2023-05-10 12:30:00.000	50.00	10	1

Mejorado:

	InventoryLogId	posttime	quantity	productId	operationTypeid
1	1	2023-05-01 10:00:00.000	50.00	1	1
2	2	2023-05-02 12:00:00.000	30.00	2	1
3	3	2023-05-03 15:30:00.000	40.00	3	1
4	4	2023-05-04 09:15:00.000	20.00	4	1
5	5	2023-05-05 14:45:00.000	60.00	5	1
6	6	2023-05-06 11:30:00.000	25.00	6	1
7	7	2023-05-07 13:20:00.000	35.00	7	1
8	8	2023-05-08 16:10:00.000	45.00	8	1
9	9	2023-05-09 09:45:00.000	55.00	9	1
10	10	2023-05-10 12:30:00.000	50.00	10	1

Query original Wastes

```
1 CREATE PROCEDURE GetWasteMovementsByProducer
2     @producerId INT
3 AS
4 BEGIN
5     -- Escenario: Phantom Read
6     -- Si durante la ejecución de este stored procedure se insertan, eliminan o modifican filas en la tabla wastes,
7     -- es posible que se produzcan lecturas fantasma.
8     SELECT
9         wm.posttime,
10        wm.quantity,
11        c.containerName,
12        w.wasteName,
13        wt.typeName,
14        p.producerName,
15        co.countryName
16    FROM
17        dbo.wasteMovements wm
18    INNER JOIN
19        dbo.wastes w ON wm.wasteId = w.wasteId
20    INNER JOIN
21        dbo.wasteTypes wt ON w.wasteType = wt.wasteTypeId
22    INNER JOIN
23        dbo.addresses a ON wm.addressId = a.addressId
24    INNER JOIN
25        dbo.countries co ON a.countryId = co.countryId
26    INNER JOIN
27        dbo.containers c ON wm.containerId = c.containerId
28    INNER JOIN
29        dbo.containerTypes ct ON c.containerTypeId = ct.containerTypeId
30    INNER JOIN
31        dbo.producersXmovements pxm ON wm.wasteMovementId = pxm.wasteMovementId
32    INNER JOIN
33        dbo.producers p ON pxm.producerId = p.producerId
34    WHERE
35        p.producerId = @producerId
36    ORDER BY
37        wm.posttime DESC;
38 END
```

Query mejorado para evitar Phantom

```
1 CREATE PROCEDURE GetWasteMovementsByProducer
2     @producerId INT
3 AS
4 BEGIN
5     SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
6     BEGIN TRANSACTION;
7
8     -- Escenario: Phantom Read
9     -- Si durante la ejecución de este stored procedure se insertan, eliminan o modifican filas en la tabla wastes,
10    -- es posible que se produzcan lecturas fantasma.
11    SELECT
12        wm.posttime,
13        wm.quantity,
14        c.containerName,
15        w.wasteName,
16        wt.typeName,
17        p.producerName,
18        co.countryName
19    FROM
20        dbo.wasteMovements wm WITH (HOLDLOCK) -- Acquire a shared lock on the table
21    INNER JOIN
22        dbo.wastes w ON wm.wasteId = w.wasteId
23    INNER JOIN
24        dbo.wasteTypes wt ON w.wasteType = wt.wasteTypeId
25    INNER JOIN
26        dbo.addresses a ON wm.addressId = a.addressId
27    INNER JOIN
28        dbo.countries co ON a.countryId = co.countryId
29    INNER JOIN
30        dbo.containers c ON wm.containerId = c.containerId
31    INNER JOIN
32        dbo.containerTypes ct ON c.containerTypeId = ct.containerTypeId
33    INNER JOIN
34        dbo.producersXmovements pxm WITH (HOLDLOCK) ON wm.wasteMovementId = pxm.wasteMovementId -- Acquire a shared lock on the table
35    INNER JOIN
36        dbo.producers p ON pxm.producerId = p.producerId
37    WHERE
38        p.producerId = @producerId
39    ORDER BY
40        wm.posttime DESC;
41
42    COMMIT;
43 END;
44
```

Pruebas con el query original y el mejorado (Wastes Phantom)

Original:

120 %

Results

Messages

	posttime	quantity	containerName	wasteName	typeName	producerName	countryName
8411	2023-01-01	999.90	Container A	Organic Waste	Paper	John Doe	United States
8412	2023-01-01	999.90	Container A	Organic Waste	Paper	John Doe	United States
8413	2023-01-01	999.90	Container A	Organic Waste	Paper	John Doe	United States
8414	2023-01-01	999.90	Container A	Organic Waste	Paper	John Doe	United States
8415	2023-01-01	999.90	Container A	Organic Waste	Paper	John Doe	United States
8416	2023-01-01	999.90	Container A	Organic Waste	Paper	John Doe	United States
8417	2023-01-01	1000....	Container A	Organic Waste	Paper	John Doe	United States
8418	2023-01-01	1000....	Container A	Organic Waste	Paper	John Doe	United States
8419	2023-01-01	1000....	Container A	Organic Waste	Paper	John Doe	United States

	posttime	quantity	containerName	wasteName	typeName	producerName	countryName
8412	2023-01-01	999.90	Container A	Organic Waste	Paper	John Doe	United States
8413	2023-01-01	999.90	Container A	Organic Waste	Paper	John Doe	United States
8414	2023-01-01	999.90	Container A	Organic Waste	Paper	John Doe	United States
8415	2023-01-01	999.90	Container A	Organic Waste	Paper	John Doe	United States
8416	2023-01-01	999.90	Container A	Organic Waste	Paper	John Doe	United States
8417	2023-01-01	1000....	Container A	Organic Waste	Paper	John Doe	United States
8418	2023-01-01	1000....	Container A	Organic Waste	Paper	John Doe	United States
8419	2023-01-01	1000....	Container A	Organic Waste	Paper	John Doe	United States
8420	2023-01-01	1000....	Container A	Organic Waste	Paper	John Doe	United States

Query executed successfully.

LEOC/SQLEXPRESS (16.0 RTM) LEOC/leona (32) caso3 00:00:06 16 839 rows

Mejorado:

Results

Messages

	posttime	quantity	containerName	wasteName	typeName	producerName	countryName
8412	2023-01-01	999.90	Container A	Organic Waste	Paper	John Doe	United States
8413	2023-01-01	999.90	Container A	Organic Waste	Paper	John Doe	United States
8414	2023-01-01	999.90	Container A	Organic Waste	Paper	John Doe	United States
8415	2023-01-01	999.90	Container A	Organic Waste	Paper	John Doe	United States
8416	2023-01-01	999.90	Container A	Organic Waste	Paper	John Doe	United States
8417	2023-01-01	1000....	Container A	Organic Waste	Paper	John Doe	United States
8418	2023-01-01	1000....	Container A	Organic Waste	Paper	John Doe	United States
8419	2023-01-01	1000....	Container A	Organic Waste	Paper	John Doe	United States
8420	2023-01-01	1000....	Container A	Organic Waste	Paper	John Doe	United States

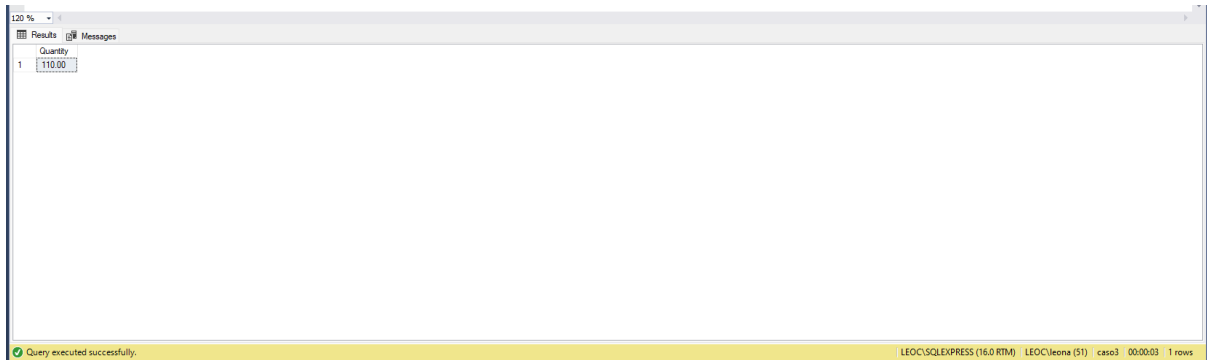
	posttime	quantity	containerName	wasteName	typeName	producerName	countryName
8412	2023-01-01	999.90	Container A	Organic Waste	Paper	John Doe	United States
8413	2023-01-01	999.90	Container A	Organic Waste	Paper	John Doe	United States
8414	2023-01-01	999.90	Container A	Organic Waste	Paper	John Doe	United States
8415	2023-01-01	999.90	Container A	Organic Waste	Paper	John Doe	United States
8416	2023-01-01	999.90	Container A	Organic Waste	Paper	John Doe	United States
8417	2023-01-01	1000....	Container A	Organic Waste	Paper	John Doe	United States
8418	2023-01-01	1000....	Container A	Organic Waste	Paper	John Doe	United States
8419	2023-01-01	1000....	Container A	Organic Waste	Paper	John Doe	United States
8420	2023-01-01	1000....	Container A	Organic Waste	Paper	John Doe	United States

Query executed successfully.

LEOC/SQLEXPRESS (16.0 RTM) LEOC/leona (32) caso3 00:00:05 8 420 rows

Pruebas con el query original y el mejorado (Dirty Read)

Original:



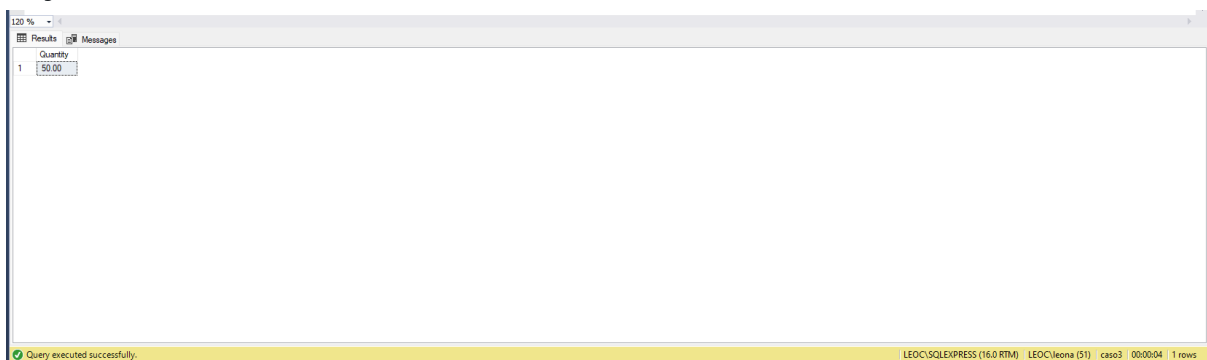
120 %

Results Messages

Quantity	
1	110.00

Query executed successfully. LEOC\SQLEXPRESS (16.0 RTM) LEOC\leona (51) caso3 00:00:03 1 rows

Mejorado:



120 %

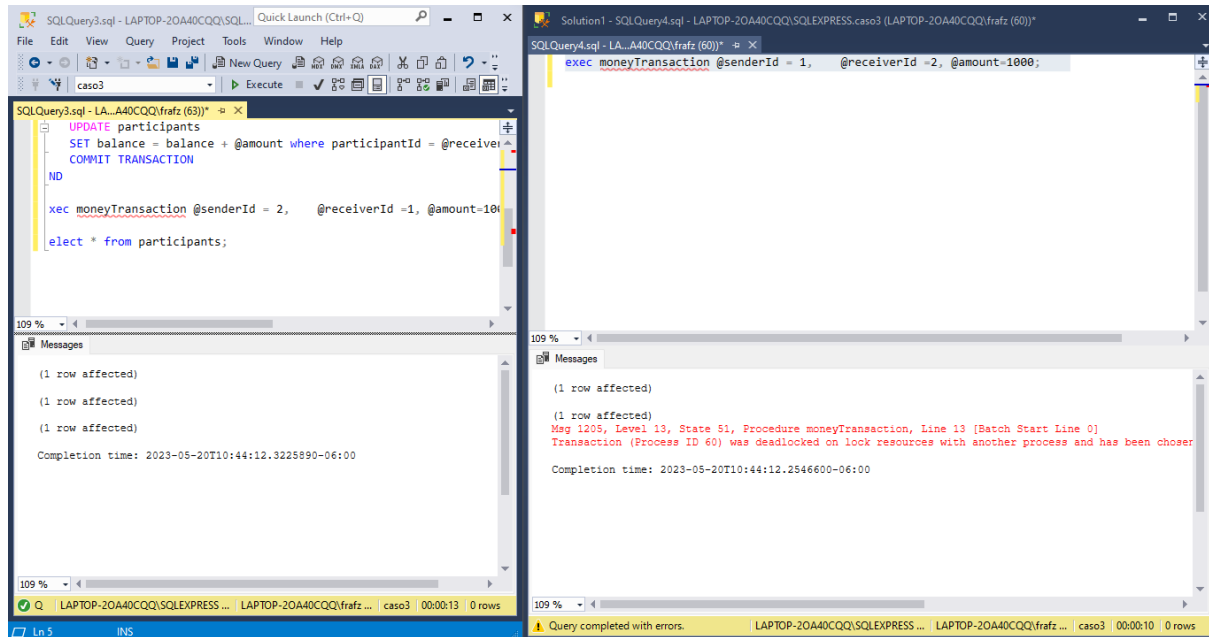
Results Messages

Quantity	
1	50.00

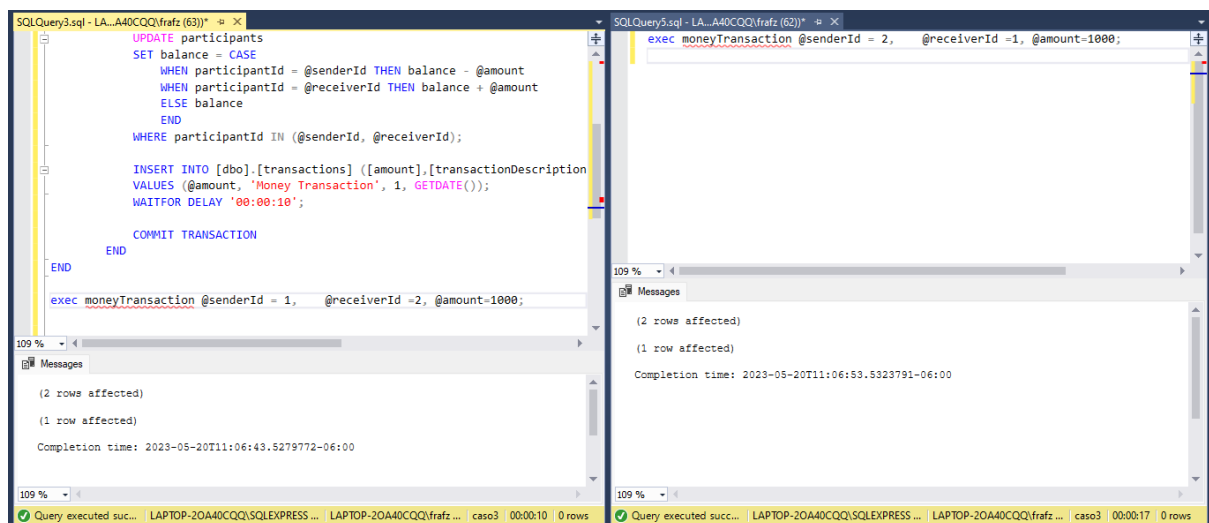
Query executed successfully. LEOC\SQLEXPRESS (16.0 RTM) LEOC\leona (51) caso3 00:00:04 1 rows

Pruebas con el query original y el mejorado (Deadlocks)

Original:



Mejorado:



Cambios:

1. Poner la transacción en SERIALIZABLE al comienzo del procedimiento. Esto asegura que las transacciones se ejecuten de manera serializada y reduce las posibilidades de deadlocks.
2. Tener un orden coherente y consistente para todas las transactions
3. Reducir la cantidad de veces que se accede a una tabla.

a) demuestre que es posible tener SP encriptados y que un atacante no va a poder ver el código del mismo

Código de sp con encriptación:

```
Alter PROCEDURE [dbo].[registerSales]
    @client INT,
    @product INT,
    @seller INT,
    @totalPrice DECIMAL(12,2),
    @paymentType INT,
    @contract INT,
    @quantity INT
    WITH ENCRYPTION
AS
BEGIN
    -- Escenario: Actualizaciones concurrentes
    -- Si varias instancias de este procedimiento almacenado se ejecutan
    simultáneamente y actualizan el mismo @producto,
    -- existe la posibilidad de que se pierdan actualizaciones cuando una
    actualización sobrescribe los cambios realizados por otra instancia.
    DECLARE @sellerPercentage decimal(5,2);
    DECLARE @producerPercentage decimal(5,2);
    DECLARE @collectorPercentage decimal(5,2);
    DECLARE @availableStock int;

    set @availableStock = (Select top 1 inventoryProduct.quantity FROM
    inventoryProduct WHERE productId = @product);

    IF(@availableStock >= @quantity)
    BEGIN
        UPDATE inventoryProduct SET quantity = quantity -
        @quantity WHERE productId = @product;

        INSERT INTO [dbo].[sales]([clientId], [productId], [sellerId], [totalPrice],
        [posttime], [checksum], [paymentTypeId], [contractId], [quantity]) VALUES
        (@client, @product, @seller, @totalPrice, GETDATE(), NULL,
        @paymentType, @contract, @quantity);

        SET @sellerPercentage = (SELECT participantPercentage FROM
        contractParticipants
        INNER JOIN contracts ON contractParticipants.contractId =
        contracts.contractId
        WHERE contractParticipants.contractId = @contract AND
        contractParticipants.participantId = @seller);

        -- Update seller's balance
```

```
UPDATE participants
SET participants.balance = participants.balance + @totalPrice *
(@sellerPercentage/100)
  where participants.participantId = @seller;
-- Update producers balance
UPDATE producers
SET producers.balance = producers.balance + @totalPrice *
(contractProducers.producerPercentage / 100)
FROM contracts
  INNER JOIN contractProducers ON contracts.contractId =
contractProducers.contractId
  INNER JOIN producers ON contractProducers.producerId =
producers.producerId
  WHERE contracts.contractId = @contract;
-- Update collectors balance
UPDATE collectors
SET collectors.balance = collectors.balance + @totalPrice *
(contractCollectors.collectorPercentage / 100)
FROM contracts
  INNER JOIN contractCollectors ON contracts.contractId =
contractCollectors.contractId
  INNER JOIN collectors ON contractCollectors.collectorId =
collectors.collectorId
  WHERE contracts.contractId = @contract;
END;
END;
```

Al correr “sp_HelpText [registerSales]” se puede verificar que la encriptación fue exitosa.



Código para revisar la descripción del SP.

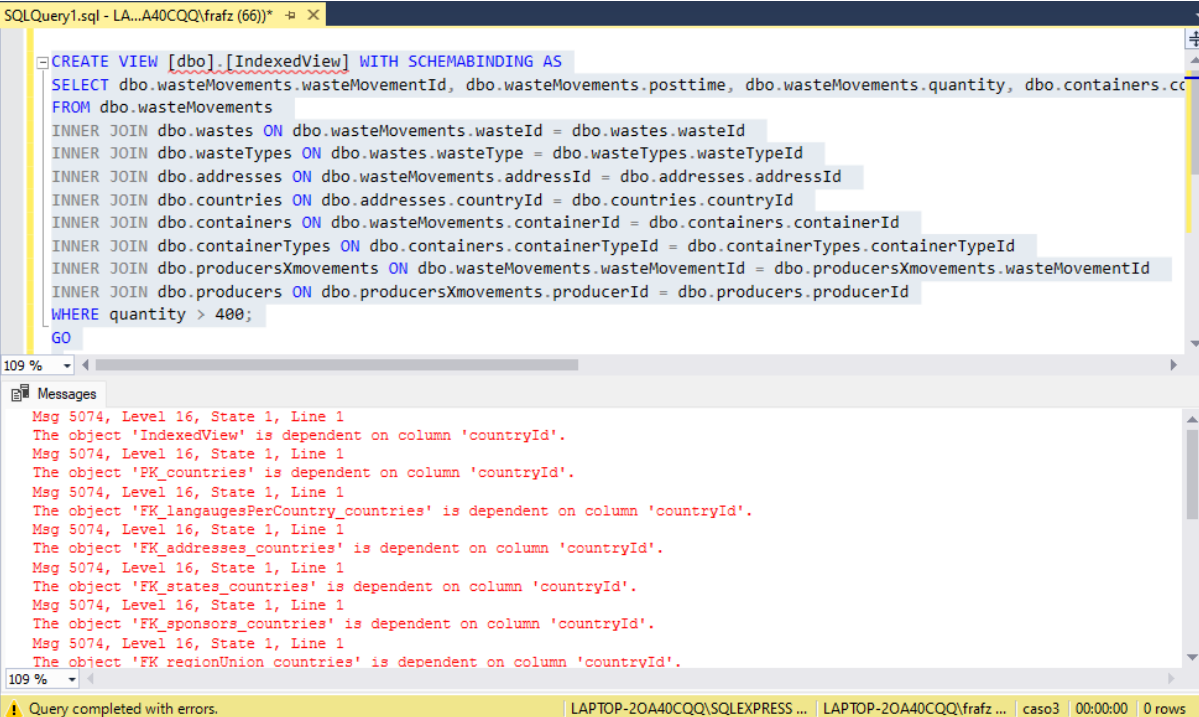
```
SELECT
  m.definition
FROM
  sys.sql_modules m
INNER JOIN
  sys.objects o ON m.object_id = o.object_id
WHERE
  o.type = 'P'
  AND o.name = 'registerSales';
```

b) por medio de un script asegúrese que un schemabinding nos proteja la lógica de negocios de la base de datos ante cambios estructurales en las tablas

```
CREATE VIEW [dbo].[IndexedView] WITH SCHEMABINDING AS
SELECT dbo.wasteMovements.wasteMovementId,
dbo.wasteMovements.posttime, dbo.wasteMovements.quantity,
dbo.containers.containerName, dbo.wastes.wasteName,
dbo.wasteTypes.typeName, dbo.producers.producerName,
dbo.countries.countryName
FROM dbo.wasteMovements
INNER JOIN dbo.wastes ON dbo.wasteMovements.wasteId = dbo.wastes.wasteId
INNER JOIN dbo.wasteTypes ON dbo.wastes.wasteType =
dbo.wasteTypes.wasteTypeId
INNER JOIN dbo.addresses ON dbo.wasteMovements.addressId =
dbo.addresses.addressId
INNER JOIN dbo.countries ON dbo.addresses.countryId = dbo.countries.countryId
INNER JOIN dbo.containers ON dbo.wasteMovements.containerId =
dbo.containers.containerId
INNER JOIN dbo.containerTypes ON dbo.containers.containerTypeId =
dbo.containerTypes.containerTypeId
INNER JOIN dbo.producersXmovements ON
dbo.wasteMovements.wasteMovementId =
dbo.producersXmovements.wasteMovementId
INNER JOIN dbo.producers ON dbo.producersXmovements.producerId =
dbo.producers.producerId
WHERE quantity > 400;
GO

ALTER TABLE dbo.countries
DROP COLUMN countryId;
```

Mensaje de error:



```
SQLQuery1.sql - LA...A40CQQ\frfz (66)* - X
CREATE VIEW [dbo].[IndexedView] WITH SCHEMABINDING AS
SELECT dbo.wasteMovements.wasteMovementId, dbo.wasteMovements.posttime, dbo.wasteMovements.quantity, dbo.containers.co
FROM dbo.wasteMovements
INNER JOIN dbo.wastes ON dbo.wasteMovements.wasteId = dbo.wastes.wasteId
INNER JOIN dbo.wasteTypes ON dbo.wastes.wasteType = dbo.wasteTypes.wasteTypeId
INNER JOIN dbo.addresses ON dbo.wasteMovements.addressId = dbo.addresses.addressId
INNER JOIN dbo.countries ON dbo.addresses.countryId = dbo.countries.countryId
INNER JOIN dbo.containers ON dbo.wasteMovements.containerId = dbo.containers.containerId
INNER JOIN dbo.containerTypes ON dbo.containers.containerTypeId = dbo.containerTypes.containerTypeId
INNER JOIN dbo.producersXmovements ON dbo.wasteMovements.wasteMovementId = dbo.producersXmovements.wasteMovementId
INNER JOIN dbo.producers ON dbo.producersXmovements.producerId = dbo.producers.producerId
WHERE quantity > 400;
GO

109 %
Messages
Msg 5074, Level 16, State 1, Line 1
The object 'IndexedView' is dependent on column 'countryId'.
Msg 5074, Level 16, State 1, Line 1
The object 'FK_countries' is dependent on column 'countryId'.
Msg 5074, Level 16, State 1, Line 1
The object 'FK_languagesPerCountry_countries' is dependent on column 'countryId'.
Msg 5074, Level 16, State 1, Line 1
The object 'FK_addresses_countries' is dependent on column 'countryId'.
Msg 5074, Level 16, State 1, Line 1
The object 'FK_states_countries' is dependent on column 'countryId'.
Msg 5074, Level 16, State 1, Line 1
The object 'FK_sponsors_countries' is dependent on column 'countryId'.
Msg 5074, Level 16, State 1, Line 1
The object 'FK regionUnion countries' is dependent on column 'countryId'.

109 %
Query completed with errors. LAPTOP-20A40CQQ\SQLXPRESS ... LAPTOP-20A40CQQ\frfz ... caso3 00:00:00 0 rows
```

c) cree dos jobs del sistema, uno que recompile todos los stored procedures, sacando la lista de los mismos de las vistas del sistema, y que eso lo haga 1 vez por semana. Finalmente, otro que se encargue de pasar los registros de la bitácora del sistema a una bitácora gemela en un linked server, garantizando la transferencia de los registros y con ello eliminando los registros pasados y así mantener la bitácora con un tamaño aceptable, dicha operación debe hacerse 2 veces por semana.

