



### 1.1 Explique y ejemplifique como implementa seguridad a través de distintos objetos de BD. El DBMS implementa seguridad por ejemplo a través de:

Los **TRIGGERS**, estos pueden estar atentos al acceso a una tabla en un horario restringido chequeando que el usuario que accedió a la tabla este autorizado. Si no estaba autorizado a entran en ese horario se registra en otra tabla quien fue el usuario ingresante.

Los **ROLES o PERFILES** limitan a los usuarios a poder realizar acciones con determinados objetos de la Base. Por ejemplo si un usuario tiene asignado el rol Ventas podríamos limitarlo a que solo pueda modificar datos en la tabla Ventas, pero que no pueda realizar ninguna acción con la tabla Compras.

Las **VISTAS**, estas nos permiten ocultar que es lo que realmente está consultando el usuario que consulta a la vista. Por Ejemplo si nosotros creamos una vista que se llame VentasPorMes el usuario que acceda a ella solo realiza `SELECT * FROM VentasPorMes`, pero no tiene la información de que a que tablas se consulta para generar la vista.

Las **FUNCIONES**, estas nos permiten ocultar procesos internos. Por ejemplo si tenemos una función que calcula que precio asignarle a cierto producto el usuario solo deberá llamar a la función pasándole por parámetro el producto, pero no sabe como es que se averigua el precio a asignar.

Los **SINONIMOS**, estos nos permiten ocultar la real ubicación de las tablas. Por ejemplo un usuario puede consultar una tabla Clientes (Sinónimo) pero esa tabla puede no estar en esa base, sino que se va a buscar a otra base.

---

Los **STORED PROCEDURES**, estos nos permiten hacer chequeos de permisos entre todos los usuarios. Por ejemplo se puede ejecutar un SP diariamente que chequee que cada usuario tenga asignado su rol correctamente

**1.2 Explique Lectura Sucia, lectura repetible y lectura fantasma. Relacione los conceptos con cada nivel de aislamiento.**

**Lectura sucia(Dirty Read)**

Datos actualizados en una transacción concurrente que luego se deshacen por un rollback.

Si una transacción T1 se encuentra actualizando datos de una tabla y otra transacción T2 lee estos datos actualizados antes de que se confirme T1, entonces T1 podría realizar un rollback y la T2 habría leído datos "sucios", por eso se lo llama *dirty read*.

**Lectura fantasma (Phantom Read)**

Registros o filas que aparecen durante la transacción que fueron insertadas en otra transacción concurrente.

Si una transacción T1 se encuentra insertando registros en una tabla y otra transacción T2 lee estos registros insertados antes de que se confirme T1, entonces T1 podría realizar un rollback y la T2 habría leído registros "fantasmas", por eso se lo llama *phantom read* o *phantom records*.

**Lectura repetible (Repeatable Read)**

En la misma transacción poder hacer dos veces o más la misma consulta en distintos momentos, asegurando siempre el mismo resultado de la consulta.

Si se realiza una misma consulta SELECT en dos momentos diferentes de la transacción debe arrojar como resultado los mismos registros exactamente en cantidad y estado.

## Niveles de aislamiento

### **Read Uncommitted**

Es el nivel de aislamiento más permisivo, permitiendo que puedan existir lecturas sucias y lecturas fantasmas, y NO asegura lecturas repetibles.

### **Read Committed**

El nivel de aislamiento Read Committed asegura que sólo leerá datos confirmados o commiteados por otra transacción, entonces NO realizará lecturas sucias.

Pero si bien el Read Committed ante una lectura de datos chequea la existencia de bloqueos exclusivos en la tabla a consultar, una vez que leyó los datos estos podrían ser bloqueados o modificados por otra transacción concurrente, entonces pueden existir lecturas fantasmas y no asegura lecturas repetibles.

### **Repeatable Read**

Este nivel de aislamiento asegura que NO existan lecturas sucias, NO evita las lecturas fantasmas y asegura que las lecturas sean repetibles solo para dirty read, pero NO para phantom record.

### **Serializable Read**

Es el nivel de aislamiento menos permisivo, lo que implica que asegura que no existan lecturas sucias y lecturas fantasmas, y asegura que las lecturas sean repetibles.

## 2. Query

Mostrar el código y descripción del Estado, código y descripción del Tipo de Producto y la cantidad de unidades vendidas de aquellos tipos de productos más vendidos (por cantidad) en cada Estado. Si hubiese más de un tipo de producto en un mismo estado con la misma cantidad de unidades vendidas máxima, informarlos a todos.

Mostrar el resultado ordenado por cantidad total vendida en orden descendente y, en caso que las cantidades de varios productos sean iguales, por la Descripción del Estado en forma ascendente, si los productos con cantidades iguales son del mismo estado, ordenar además por descripción del tipo de producto en forma ascendente.

SELECT

```
j.Cod_Estado AS Cod_Estado,  
j.Descripcion_Estado AS Descripcion_Estado,  
stock_num AS Cod_Tipo_Producto,  
description AS Descripcion_Tipo_Producto,  
j.Cant_Total_Vendida AS Cant_Total_Vendida
```

FROM product\_types pto

RIGHT JOIN

(

SELECT

```
st.state AS Cod_Estado,  
st.sname AS Descripcion_Estado,  
pt.stock_num AS Cod_Tipo_Producto,  
pt.description AS Descripcion_Tipo_Producto,  
SUM(i.quantity) AS Cant_Total_Vendida
```

FROM items i

```
JOIN product_types pt ON pt.stock_num = i.stock_num
```

```
JOIN orders o ON o.order_num = i.order_num
```

```
JOIN customer c ON c.customer_num = o.customer_num
```

```
JOIN state st ON st.state = c.state
```

GROUP BY st.state, st.sname, pt.stock\_num, pt.description

HAVING SUM(i.quantity) =

(SELECT TOP 1 SUM(i.quantity)

FROM items i

```
JOIN orders o ON o.order_num = i.order_num
```

---

```
JOIN customer c ON c.customer_num = o.customer_num

JOIN state st ON st.state = c.state

GROUP BY st.state, i.stock_num

ORDER BY 1 DESC)

) j ON j.Cod_Tipo_Producto = pto.stock_num AND j.Descripcion_Tipo_Producto =
pto.description

ORDER BY j.Cant_Total_Vendida, j.Descripcion_Estado, pto.description
```

### 3. Store Procedure

Desarrollar un stored procedure que realice la inserción o modificación de un producto determinado.

Parámetros de Entrada STOCK\_NUM, MANU\_CODE, UNIT\_PRICE, UNIT\_CODE, DESCRIPTION

Previamente a realizar alguna operación validar:

EXISTENCIA de MANU\_CODE en Tabla MANUFACT - Informando Error por Fabricante Inexistente.

EXISTENCIA del atributo UNIT\_CODE en la Tabla UNITS - Informando Error por Código de Unidad Inexistente

EXISTENCIA en Tabla PRODUCT\_TYPES –

Si no existe INSERTAR Registro en la tabla PRODUCT\_TYPES

Sino realizar el UPDATE del atributo 'description'.

Una vez validados los parámetros: Si el producto no existe, insertarlo. En caso que ya exista, actualizar los atributos no clave.

GO

```
CREATE PROCEDURE punto3 @stock_num SMALLINT, @manu_code CHAR(3), @unit_price DECIMAL,
@unit_code SMALLINT, @description VARCHAR(15)
```

AS

BEGIN

BEGIN TRY

BEGIN TRANSACTION

-- Previamente a realizar alguna operación validar:

-- EXISTENCIA de MANU\_CODE en Tabla MANUFACT - Informando Error por Fabricante Inexistente.

IF NOT EXISTS(SELECT 1 FROM manufact WHERE manu\_code = @manu\_code)

THROW 50000, 'Fabricante Inexistente' , 1

---

```
-- EXISTENCIA del atributo UNIT_CODE en la Tabla UNITS - Informando
Error por Código de Unidad Inexistente

IF NOT EXISTS(SELECT 1 FROM units WHERE unit_code = @unit_code)
    THROW 50001, 'Fabricante Inexistente' , 1

-- EXISTENCIA en Tabla PRODUCT_TYPES -
-- Si no existe INSERTAR Registro en la tabla PRODUCT_TYPES
-- Sino realizar el UPDATE del atributo 'description'.

IF NOT EXISTS(SELECT 1 FROM product_types WHERE stock_num =
@stock_num)

    INSERT INTO product_types(stock_num, description)
    VALUES(@stock_num, @description)

ELSE

    UPDATE product_types
        SET description = @description
        WHERE stock_num = @stock_num

-- Una vez validados los parámetros:
-- Si el producto no existe, insertarlo.
-- En caso que ya exista, actualizar los atributos no clave.

IF NOT EXISTS(SELECT 1 FROM products WHERE stock_num = @stock_num
AND manu_code = @manu_code)

    INSERT INTO products(stock_num, manu_code, unit_price,
unit_code)

    VALUES(@stock_num, @manu_code, @unit_price, @unit_code)

ELSE

    UPDATE products
        SET unit_price = @unit_price, unit_code = @unit_code
        WHERE stock_num = @stock_num AND manu_code = @manu_code

COMMIT TRANSACTION

END TRY

BEGIN CATCH
```

---

ROLLBACK TRANSACTION

END CATCH

END

#### 4. Trigger

Dada la vista:

CREATE VIEW OrdenItems AS

```
SELECT o.order_num, o.order_date, o.customer_num, o.paid_date,  
       i.item_num, i.stock_num, i.manu_code, i.quantity, i.unit_price  
FROM orders o JOIN items i ON o.order_num = i.order_num;
```

Se desea manejar las operaciones de ALTA sobre la vista anterior.

Los controles a realizar son los siguientes:

- No se permitirá que una Orden contenga ítems de fabricantes de más de dos estados en la misma orden.
- Por otro parte los Clientes del estado de ALASKA no podrán realizar compras a fabricantes fuera de ALASKA.

Notas:

- Las altas son de una misma Orden y de un mismo Cliente pero pueden venir varias líneas de ítems en una ORDEN.
- Ante el incumplimiento de una validación, deshacer TODA la transacción y finalizar la ejecución.

GO

CREATE VIEW OrdenItems AS

SELECT

```
o.order_num,  
o.order_date,  
o.customer_num,  
o.paid_date,  
i.item_num,  
i.stock_num,  
i.manu_code,  
i.quantity,  
i.unit_price
```

FROM orders o

```
JOIN items i ON o.order_num = i.order_num
```

GO

---

```
CREATE TRIGGER insert_OrdenItems ON OrdenItems

INSTEAD OF INSERT AS

BEGIN

    BEGIN TRY

        BEGIN TRANSACTION

        DECLARE @order_num SMALLINT, @order_date DATETIME, @customer_num SMALLINT,
@paid_date DATETIME, @item_num SMALLINT, @stock_num SMALLINT, @manu_code CHAR(3),
@quantity SMALLINT, @unit_price DECIMAL

        DECLARE cursor_inserted CURSOR FOR

        SELECT order_num, order_date, customer_num, paid_date, item_num, stock_num,
manu_code, quantity, unit_price

        FROM inserted

        OPEN cursor_inserted

        FETCH NEXT FROM cursor_inserted

        INTO @order_num, @order_date, @customer_num, @paid_date, @item_num,
@stock_num, @manu_code, @quantity, @unit_price

        WHILE @@FETCH_STATUS = 0
        BEGIN

            -- Los controles a realizar son los siguientes:

            -- a.No se permitirá que una Orden contenga ítems de fabricantes de
            más de dos estados en la misma orden.

            IF( (SELECT COUNT(DISTINCT m.state) FROM items i JOIN manufact m ON
m.manu_code = i.manu_code WHERE order_num = @order_num) > 2)

                THROW 50000, 'Una orden no puede contener ítems de fabricantes
de más de dos estados en la misma orden' , 1
```



```
-- b.Por otro parte los Clientes del estado de ALASKA no podrán
realizar compras a fabricantes fuera de ALASKA.

IF ( (SELECT state FROM customer WHERE customer_num = @customer_num)
= 'AK' AND EXISTS( SELECT 1 FROM manufact WHERE manu_code = @manu_code AND state != 'AK')
)

    THROW 50000, 'Los Clientes del estado de ALASKA no podrán
realizar compras a fabricantes fuera de ALASKA' , 1

INSERT INTO OrdenItems(order_num, order_date, customer_num,
paid_date, item_num, stock_num, manu_code, quantity, unit_price)

VALUES(@order_num, @order_date, @customer_num, @paid_date,
@item_num, @stock_num, @manu_code, @quantity, @unit_price)

FETCH NEXT FROM cursor_inserted

INTO @order_num, @order_date, @customer_num, @paid_date, @item_num,
@stock_num, @manu_code, @quantity, @unit_price

END

COMMIT TRANSACTION

END TRY

BEGIN CATCH

    ROLLBACK TRANSACTION

END CATCH

CLOSE cursor_inserted

DEALLOCATE cursor_inserted

END
```

---

Notas

1.1	1.2	2	3	4
12	13	30	23	22