

Asignatura	Datos del alumno	Fecha
Bases De Datos	Apellidos: De Mendoza	24/11/2024
	Nombre: Alejandro	

ACTIVIDAD LABORATORIO NO.2
PROGRAMACIÓN DE CURSORES, FUNCIONES, PROCEDIMIENTOS Y DISPARADORES
EN SQL

PRESENTADO POR:
ALEJANDRO DE MENDOZA

PRESENTADO AL PROFESOR:
ING JAVIER DIAZ DIAZ

FUNDACIÓN UNIVERSITARIA INTERNACIONAL DE LA RIOJA
BOGOTÁ D.C.
24 DE NOVIEMBRE
2024

Asignatura	Datos del alumno	Fecha
Bases De Datos	Apellidos: De Mendoza	24/11/2024
	Nombre: Alejandro	

TABLA DE CONTENIDO

INTRODUCCIÓN	3
DESARROLLO DEL LABORATORIO.....	3
CREACIÓN DE TABLAS Y CARGA DE DATOS	3
USO DE CURSORES PARA CALCULAR TOTAL DE VENTAS.....	4
CREACIÓN DE FUNCIÓN	6
USO DE PROCEDIMIENTOS PARA ACTUALIZAR STOCK	9
CREACIÓN DE UN DISPARADOR (TRIGGER)	13
CONCLUSIÓN	15
BIBLIOGRAFÍA	15

Asignatura	Datos del alumno	Fecha
Bases De Datos	Apellidos: De Mendoza	24/11/2024
	Nombre: Alejandro	

INTRODUCCIÓN

Como introducción y para la realización de este laboratorio se estudiarán los temas sobre SQL: *Creación Y Carga De Datos, Creación De Tablas, Realizar Inserción De Datos, Creación De Procedimientos, Funciones, Cursores Y Disparadores*. Y es por esto que, lo primero es que voy a crear es una tabla que se denomina “empleados” que tiene la siguiente estructura:

ID_PRODUCTO	NOMBRE	PRECIO	STOCK
1	Producto A	50	100
2	Producto B	75	80
3	Producto C	60	120
4	Producto D	90	60

Luego voy a crear una tabla adicional denominada “registro_ventas” con columnas para id_venta (número), id_producto (número), cantidad (número) y fecha_venta (fecha) para crear un procedimiento que utilice un cursor para calcular el total de ventas basado en la cantidad y precio de los productos vendidos. Después, voy a crear una función llamada “obtener_stock_producto” que reciba el ID de un producto como parámetro y devuelva la cantidad en stock de ese producto. Y se va a utilizar una consulta para seleccionar y devolver el stock del producto correspondiente al ID proporcionado. Luego ejecutare la creación de un procedimiento llamado “actualizar_stock” que reciba el ID de un producto y la cantidad vendida como parámetros. Para entonces actualizar el stock del producto restando la cantidad vendida. Finalmente, se va a crear de mi parte un disparador llamado “trg_actualizar_stock” que se active antes de insertar una fila en la tabla “registro_ventas”, y se va a utilizar este disparador para llamar al procedimiento “actualizar_stock” con los valores del nuevo producto y cantidad a insertar, permitiendo actualizar automáticamente el stock antes de registrar la venta. Es importante indicar que en este ejercicio voy a utilizar la herramienta **APEX de Oracle**, con el fin de ejecutar el desarrollo correcto y correspondiente al ejercicio de la actividad de laboratorio. Tener presente que de igual manera que toda la creación de los códigos, funciones, procedimientos, cursores y disparadores se van a adjuntar en un archivo de bloc de notas para facilidad a la hora de ejecución, en caso que sea requerido.

DESARROLLO DEL LABORATORIO

CREACIÓN DE TABLAS Y CARGA DE DATOS

Con base en el desarrollo de la actividad y como primer paso procedo a ejecutar la creación de las dos primeras tablas con base en la actividad 1 a desarrollar en este laboratorio de gran importancia:

Creación De Tabla “Productos”

Para este desarrollo lo primero es entonces crear la primera tabla en Apex de Oracle de nombre “productos”, el código es el siguiente:

```
create table productos (
  id_producto number,
  nombre varchar2(50),
  precio number,
  stock number
);
```

Luego entonces procedo a insertar los valores con el siguiente comando:

```
1 INSERT INTO productos VALUES (1, 'Producto A', 50, 100);
2 INSERT INTO productos VALUES (2, 'Producto B', 75, 80);
3 INSERT INTO productos VALUES (3, 'Producto C', 60, 120);
4 INSERT INTO productos VALUES (4, 'Producto D', 90, 60);
```

Es importante precisar que se debe ejecutar cada comando de forma independiente ya que si ejecuto todos los comandos al mismo tiempo se produce un error en la inserción. A continuación, la imagen de la tabla creada con sus datos:

Asignatura	Datos del alumno	Fecha
Bases De Datos	Apellidos: De Mendoza	24/11/2024
	Nombre: Alejandro	

PRODUCTOS				
	ID_PRODUCTO	NOMBRE	PRECIO	STOCK
	3	Producto C	60	120
	4	Producto D	90	60
	1	Producto A	50	100
	2	Producto B	75	80

Creación De Tabla "registro_ventas"

Ahora procedo a crear la tabla "registro_ventas" con columnas para id_venta (número), id_producto (número), cantidad (número) y fecha_venta (fecha), con el siguiente código

```
1 create table registro_ventas (
2
3 id_venta number,
4 id_producto number,
5 cantidad number,
6 fecha_venta date
7 );
```

Ahora si quiero ver la descripción de esta tabla entonces ejecuto el comando: **desc registro_ventas;** y me muestra en la consola la tabla:

Results	Explain	Describe	Saved SQL	History					
Object Type		TABLE	Object		REGISTRO_VENTAS				
Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
REGISTRO_VENTAS	ID_VENTA	NUMBER	22	-	-	-	✓	-	-
	ID_PRODUCTO	NUMBER	22	-	-	-	✓	-	-
	CANTIDAD	NUMBER	22	-	-	-	✓	-	-
	FECHA_VENTA	DATE	7	-	-	-	✓	-	-

USO DE CURSORES PARA CALCULAR TOTAL DE VENTAS

Procedo ahora a desarrollar el procedimiento que utilice un cursor para calcular el total de ventas basado en la cantidad y precio de los productos vendidos, entonces, utilizo el siguiente comando:

```
CREATE OR REPLACE PROCEDURE calcular_total_ventas IS
CURSOR ventas_cursor IS
SELECT rv.id_producto, rv.cantidad, p.precio
FROM registro_ventas rv
JOIN productos p ON rv.id_producto = p.id_producto;

v_total_ventas NUMBER := 0;
v_precio NUMBER;
v_cantidad NUMBER;
BEGIN
FOR venta IN ventas_cursor LOOP
v_precio := venta.precio;
v_cantidad := venta.cantidad;

v_total_ventas := v_total_ventas + (v_precio * v_cantidad);
END LOOP;

DBMS_OUTPUT.PUT_LINE('El total de ventas es: ' || v_total_ventas);
END calcular_total_ventas;
```

Asignatura	Datos del alumno	Fecha
Bases De Datos	Apellidos: De Mendoza	24/11/2024
	Nombre: Alejandro	

A continuación, la explicación del paso a paso del procedimiento: **CREATE OR REPLACE PROCEDURE calcular_total_ventas** IS: Donde, **CREATE OR REPLACE** indica si ya existe un procedimiento con el mismo nombre (calcular_total_ventas), se reemplazará con este procedimiento. Y **IS** marca el comienzo de la sección de declaración del procedimiento. **ventas_cursor**: El cursor selecciona los datos de dos tablas: registro_ventas y productos. **registro_ventas rv**: Se refiere a la tabla de ventas y el alias que le coloque es rv. **productos p**: Hago referencia a la tabla de productos y el alias que le indique es p. La consulta hace un JOIN entre registro_ventas y productos usando la columna id_producto en ambas tablas, y se obtienen tres columnas: id_producto, cantidad de registro_ventas, y precio de productos. **v_total_ventas**: Es una variable NUMBER que comienza en 0 y almacena el total acumulado de las ventas. **v_precio**: Es una variable de tipo NUMBER que almacenará el precio del producto de cada registro. **v_cantidad**: Variable de tipo NUMBER que almacena la cantidad de productos vendidos de cada registro. **BEGIN**: Marca el inicio del cuerpo del procedimiento. **FOR venta IN ventas_cursor LOOP**: Recorre todos los resultados del cursor ventas_cursor. Cada fila del cursor se asigna a la variable venta. Considero importante indicar que, en cada iteración del ciclo o bucle, se asignan los valores de precio y cantidad de la fila actual del cursor a las variables v_precio y v_cantidad, y se calcula el total de la venta multiplicando v_precio por v_cantidad, y el resultado se agrega al total acumulado de ventas en v_total_ventas. **DBMS_OUTPUT.PUT_LINE**: Imprimo el total acumulado de ventas calculado en el bucle, con el mensaje 'El total de ventas es: ' seguido del valor de v_total_ventas. **END calcular_total_ventas**: Marca el final del procedimiento calcular_total_ventas. Ahora con el fin de verificar procedo entonces a ejecutar, con el siguiente comando: EXEC calcular_total_ventas;

```

1 BEGIN
2   calcular_total_ventas;
3 END;

```

Results Explain Describe Saved SQL History

El total de ventas es: 0

Statement processed.

0.02 seconds

Como se denota el resultado es 0 ya que no se le han insertado datos a la tabla de “registro_ventas”, entonces voy a proceder a insertar datos para verificar que el procedimiento este correcto:

- **Producto A:**

ID_Venta = 1, ID_Producto = El mismo de la tabla de “Productos”, Cantidad = 10, Fecha Venta = Se coloco la fecha del día de hoy con el formato (“YYYY-MMM-DD”).

- **Producto B:**

ID_Venta = 2, ID_Producto = El mismo de la tabla de “Productos”, Cantidad = 5, Fecha Venta = Se coloco la fecha del día de hoy con el formato (“YYYY-MMM-DD”).

- **Producto C:**

ID_Venta = 3, ID_Producto = El mismo de la tabla de “Productos”, Cantidad = 8, Fecha Venta = Se coloco la fecha del día de hoy con el formato (“YYYY-MMM-DD”).

- **Producto D:**

Asignatura	Datos del alumno	Fecha
Bases De Datos	Apellidos: De Mendoza	24/11/2024
	Nombre: Alejandro	

ID_Venta = 4, ID_Producto = El mismo de la tabla de "Productos", Cantidad = 12, Fecha Venta = Se coloco la fecha del día de hoy con el formato ("YYYY-MM-DD"), que se puede modificar por la fecha que uno considere.

NOTA: Se aclara que no se toman los datos del Stock como las cantidades vendidas ya que Stock es diferente a estas, el Stock hace referencia al inventario de unidades y no a las cantidades vendidas.

```
INSERT INTO registro_ventas VALUES (1, 1, 10, TO_DATE('2024-11-24', 'YYYY-MM-DD'));
INSERT INTO registro_ventas VALUES (2, 2, 5, TO_DATE('2024-11-24', 'YYYY-MM-DD'));
INSERT INTO registro_ventas VALUES (3, 3, 8, TO_DATE('2024-11-24', 'YYYY-MM-DD'));
INSERT INTO registro_ventas VALUES (4, 4, 12, TO_DATE('2024-11-24', 'YYYY-MM-DD'));
```

Y la tabla queda entonces de la siguiente manera:

Results	Explain	Describe	Saved SQL	History
ID_VENTA	ID_PRODUCTO	CANTIDAD	FECHA_VENTA	
1	1	10	11/24/2024	
2	2	5	11/24/2024	
3	3	8	11/24/2024	
4	4	12	11/24/2024	

4 rows returned in 0.01 seconds [Download](#)

Ahora con los datos insertados procedo entonces a ejecutar el procedimiento con el comando:

```
1 BEGIN
2   calcular_total_ventas;
3 END;
```

Y el resultado es el siguiente:

Results	Explain	Describe	Saved SQL	History
El total de ventas es: 2435				
Statement processed.				
0.01 seconds				

Lo cual es correcto ya que si tomamos la suma de los **precios de los productos** por sus **cantidades vendidas** nos da:

$$\text{Total de Ventas} = (50 \cdot 10) + (75 \cdot 5) + (60 \cdot 8) + (90 \cdot 12) = 2435.$$

Por lo que el procedimiento está correcto.

CREACIÓN DE FUNCIÓN

A continuación, voy a proceder a crear una función llamada "obtener_stock_producto" que recibe el ID de un producto como parámetro y devuelve la cantidad en stock de ese producto. Y adicionalmente a generar una consulta para seleccionar y devolver el stock del producto correspondiente al ID proporcionado.

Creación De Función "Obtener_Stock_Producto"

Procedo entonces a crear una función llamada "obtener_stock_producto" que recibe el ID de un producto como parámetro y devuelve la cantidad en stock de ese producto. Entonces procedo a ejecutar:

Asignatura	Datos del alumno	Fecha
Bases De Datos	Apellidos: De Mendoza	24/11/2024
	Nombre: Alejandro	

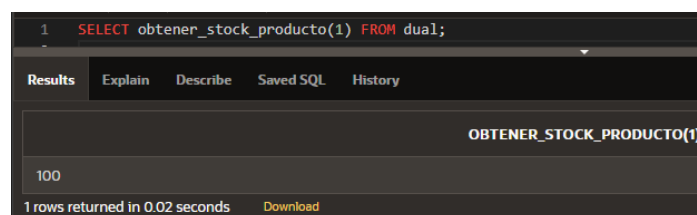
```

CREATE OR REPLACE FUNCTION obtener_stock_producto (
  p_id_producto IN NUMBER
) RETURN NUMBER IS
  v_stock NUMBER;
BEGIN
  SELECT stock
  INTO v_stock
  FROM productos
  WHERE id_producto = p_id_producto;

  RETURN v_stock;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RETURN 0;
  WHEN OTHERS THEN
    RETURN NULL;
END obtener_stock_producto;

```

A continuación, la explicación de la función: **CREATE OR REPLACE FUNCTION:** Crea o reemplaza la función llamada `obtener_stock_producto`. **p_id_producto IN NUMBER:** El parámetro de entrada `p_id_producto` es de tipo `NUMBER` y se usará para identificar el producto. **RETURN NUMBER:** La función devuelve un valor de tipo `NUMBER`, que será la cantidad en stock del producto. **v_stock:** Variable donde se almacenará el stock del producto. **SELECT stock INTO v_stock:** Esta consulta obtiene el valor del campo `stock` de la tabla `productos` y lo almacena en la variable `v_stock`. **WHERE id_producto = p_id_producto:** Filtra la consulta para obtener el stock solo del producto cuyo `id_producto` coincide con el parámetro de entrada `p_id_producto`. **RETURN v_stock:** La función devuelve el valor de la variable `v_stock`, que contiene la cantidad de stock del producto. En este caso considere efectuar manejo de excepciones en caso de que el producto no se encuentre por medio de su ID, por ejemplo, en dado caso que el producto no se encuentre en la base de datos, entonces la función lo que hace es devolver el valor de 0 (cero) como un valor por defecto. A continuación, su explicación en detalle: **WHEN NO_DATA_FOUND THEN:** Si no se encuentra un producto con el `id_producto` dado, como explicó la función devuelve 0 como valor por defecto. **WHEN OTHERS THEN:** Si ocurre cualquier otro error, la función devuelve `NULL`. Ahora con el fin de validar que la función ha sido desarrollada correctamente entonces procedo a llamarla con el comando: **SELECT obtener_stock_producto() FROM dual;** y lo que hace es devolverme el stock del producto con el id del producto y en dado caso que el producto no exista pues me devuelve el valor con un 0. Entonces acá simplemente ejecutamos la función colocando entre paréntesis el número del ID del producto, por ejemplo, para el producto A con ID = 1, entonces el comando y su respuesta quedan de la siguiente manera:



```

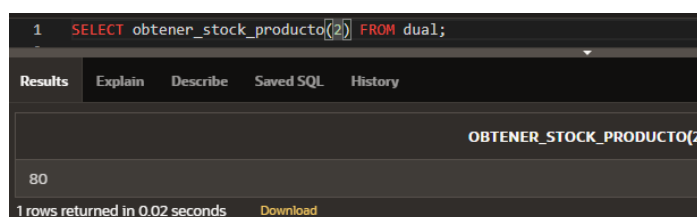
1 SELECT obtener_stock_producto(1) FROM dual;

```

OBTENER_STOCK_PRODUCTO(1)
100

1 rows returned in 0.02 seconds

Ahora para el producto B con ID = 2, entonces el comando y su respuesta quedan de la siguiente manera:



```

1 SELECT obtener_stock_producto(2) FROM dual;

```

OBTENER_STOCK_PRODUCTO(2)
80

1 rows returned in 0.02 seconds

Ahora para el producto C con ID = 3, entonces el comando y su respuesta quedan de la siguiente manera:

Asignatura	Datos del alumno	Fecha
Bases De Datos	Apellidos: De Mendoza	24/11/2024
	Nombre: Alejandro	

```
1 SELECT obtener_stock_producto(3) FROM dual;
```

Results Explain Describe Saved SQL History

OBTENER_STOCK_PRODUCTO(3)

120

1 rows returned in 0.00 seconds Download

Finalmente, para el producto D con ID = 4, entonces el comando y su respuesta quedan de la siguiente manera:

```
1 SELECT obtener_stock_producto (4) FROM dual;
```

Results Explain Describe Saved SQL History

OBTENER_STOCK_PRODUCTO(4)

60

1 rows returned in 0.01 seconds Download

Y por último al comparar con la tabla de productos, vemos que la información esta correcta:

ID_PRODUCTO	NOMBRE	PRECIO	STOCK
1	Producto A	50	100
2	Producto B	75	80
3	Producto C	60	120
4	Producto D	90	60

Consulta Para Seleccionar Y Devolver El Stock Del Producto Correspondiente Al ID Proporcionado

Esta consulta es de fácil desarrollo simplemente consiste en utilizar una consulta SELECT, a continuación, su desarrollo:

```
1 SELECT stock
2 FROM productos
3 WHERE id_producto = :id_producto;
```

Explicando el código entonces: **SELECT stock:** Selecciona la columna stock de la tabla productos, que es la que contiene la cantidad de unidades disponibles de cada producto. **FROM productos:** Especifica que la consulta se realiza sobre la tabla productos. **WHERE id_producto = :id_producto:** Filtra la consulta para obtener el stock del producto cuyo id_producto coincida con el valor proporcionado. Tener presente que el :id_producto es un parámetro que lo puedo reemplazar con el ID del producto deseado al ejecutar la consulta. Ahora al ejecutar la consulta nos lanza a la siguiente ventana:

Enter Bind Variables - Google Chrome

apex.oracle.com/pls/apex/f?p=4500:138:109838123301428::

Submit

Bind Variable	Value
:ID_PRODUCTO	1

Donde simplemente colocamos el ID del producto y nos lanza el stock, en este caso utilice el ID del Producto A que es el número 1, entonces ingreso el número 1 en el campo donde dice ID_PRODUCTO, y me trae la siguiente respuesta:

Results Explain Describe Saved SQL History

STOCK

100

1 rows returned in 0.01 seconds Download

Y como se puede denotar la consulta esta correcta. Ahora si se requiere **efectuar la consulta de manera directa** simplemente cambiamos el comando eliminando :id_producto por el número del ID del producto, entonces para el Producto B, el comando y su respuesta quedan entonces de la siguiente manera:

Asignatura	Datos del alumno	Fecha
Bases De Datos	Apellidos: De Mendoza	24/11/2024
	Nombre: Alejandro	

```

1 SELECT stock
2 FROM productos
3 WHERE id_producto = 2;

```

Results Explain Describe Saved SQL History

STOCK

80

1 rows returned in 0.01 seconds Download

Ahora para el producto C cuyo ID es 3 queda de la siguiente manera:

```

1 SELECT stock
2 FROM productos
3 WHERE id_producto = 3;

```

Results Explain Describe Saved SQL History

STOCK

120

1 rows returned in 0.01 seconds Download

Y por último para el producto D cuyo ID es 4 queda de la siguiente manera:

```

1 SELECT stock
2 FROM productos
3 WHERE id_producto = 4;

```

Results Explain Describe Saved SQL History

STOCK

60

1 rows returned in 0.01 seconds Download

Y como se puede ver al comparar con la tabla de productos las respuestas de la consulta están correctas:

ID_PRODUCTO	NOMBRE	PRECIO	STOCK
1	Producto A	50	100
2	Producto B	75	80
3	Producto C	60	120
4	Producto D	90	60

USO DE PROCEDIMIENTOS PARA ACTUALIZAR STOCK

A continuación, voy a crear un procedimiento llamado "actualizar_stock" que reciba el ID de un producto y la cantidad vendida como parámetros. Y adicionalmente, luego voy a actualizar el stock del producto restando la cantidad vendida.

Creación Del Procedimiento "actualizar_stock"

Entonces y con base en lo anterior, primero desarrollo un procedimiento llamado "actualizar_stock" que recibe el ID de un producto y la cantidad vendida como parámetros, y luego actualiza el stock de ese producto. Entonces procedo a escribir un procedimiento PL/SQL que hace lo siguiente: 1) El procedimiento primero debe recibir el ID del producto y la cantidad vendida. 2) El procedimiento debe verificar si hay suficiente stock. 3) Ahora si hay suficiente stock, el procedimiento actualiza el valor del stock en la tabla productos restando la cantidad vendida. 4) Y por último en dado caso que no haya suficiente stock, el procedimiento muestra un mensaje de advertencia. Entonces procediendo con el desarrollo del mismo lo primero es desarrollar el código de ejecución y queda de la siguiente manera:

Asignatura	Datos del alumno	Fecha
Bases De Datos	Apellidos: De Mendoza	24/11/2024
	Nombre: Alejandro	

```

1 CREATE OR REPLACE PROCEDURE actualizar_stock (
2     p_id_producto IN NUMBER,
3     p_cantidad_vendida IN NUMBER
4 ) IS
5     v_stock NUMBER;
6 BEGIN
7     SELECT stock
8     INTO v_stock
9     FROM productos
10    WHERE id_producto = p_id_producto;
11
12    IF v_stock >= p_cantidad_vendida THEN
13        UPDATE productos
14        SET stock = stock - p_cantidad_vendida
15        WHERE id_producto = p_id_producto;
16
17        DBMS_OUTPUT.PUT_LINE('El stock del producto ID ' || p_id_producto || ' ha sido actualizado.');
```

A continuación, la explicación del código: **p_id_producto IN NUMBER**: El ID del producto cuyo stock será actualizado. **p_cantidad_vendida IN NUMBER**: La cantidad vendida que se debe restar del stock del producto. **SELECT stock**: Indica que queremos seleccionar el valor de la columna stock de la tabla productos. Esta columna contiene la cantidad de unidades disponibles de cada producto en inventario. **INTO v_stock**: Este es el mecanismo en PL/SQL para asignar el valor seleccionado (de la columna stock) a una variable local del programa, en este caso, v_stock. Es preciso indicar de mi parte que **v_stock**: es una variable previamente declarada en el bloque PL/SQL donde se ejecuta esta consulta. Si el SELECT devuelve un valor, este se almacena en v_stock. **FROM productos**: Especifica que la columna stock se toma de la tabla productos. **WHERE id_producto = p_id_producto**: Es una condición que asegura que solo se seleccione el valor de stock para el producto cuyo ID coincida con el valor del parámetro p_id_producto. p_id_producto es un parámetro de entrada que se pasa al procedimiento o función PL/SQL. Representa el ID del producto que queremos consultar. **IF v_stock >= p_cantidad_vendida THEN**: Este código verifica si el stock disponible es suficiente para cubrir la cantidad vendida. **UPDATE productos**: Este comando me especifica que se realizará una actualización (modificación) en la tabla productos. Se modifica uno o más registros en esta tabla dependiendo de las condiciones establecidas en la cláusula WHERE. **SET stock = stock - p_cantidad_vendida**: Aquí se indica qué columna será modificada, entonces, **stock**: Es la columna que contiene el inventario actual del producto. **stock - p_cantidad_vendida**: En este comando se calcula el nuevo valor de stock restando la cantidad vendida (p_cantidad_vendida) al valor actual del stock, y el nuevo valor del stock reemplazará al valor existente en la tabla. **WHERE id_producto = p_id_producto**: Con este comando se especifica la condición que debe cumplir el registro para ser actualizado. Ahora el **id_producto**: es el identificador único del producto en la tabla. Por otra parte, el comando **p_id_producto**: este es un parámetro que se pasa al procedimiento o función. Representa el ID del producto que se desea actualizar. Esto asegura que solo se modifique el registro correspondiente al producto cuyo ID coincide con p_id_producto. **DBMS_OUTPUT.PUT_LINE ('El stock del producto ID ' || p_id_producto || ' ha sido actualizado.')**: Este mensaje se muestra cuando el stock de un producto ha sido actualizado correctamente. **||**: Es el operador de concatenación en SQL y PL/SQL. Combina cadenas de texto. **p_id_producto**: Es el parámetro que representa el ID del producto cuyo stock ha sido actualizado. **ELSE (cuando no hay suficiente stock)**: Si la cantidad vendida (p_cantidad_vendida) es mayor que el stock actual del producto (stock), no se realiza la actualización. En su lugar, se muestra un mensaje de advertencia. **DBMS_OUTPUT.PUT_LINE ('No hay suficiente**

Asignatura	Datos del alumno	Fecha
Bases De Datos	Apellidos: De Mendoza	24/11/2024
	Nombre: Alejandro	

stock para el producto ID ' || p_id_producto || ''); Genera un mensaje que advierte que no hay suficiente stock disponible para realizar la operación. **END IF:** Finaliza el bloque condicional que evalúa si hay suficiente stock. **NO_DATA_FOUND:** Si no se encuentra un producto con el id_producto proporcionado, muestra un mensaje indicando que no se encontró el producto. **OTHERS:** Cualquier otro error genera un mensaje indicando que ocurrió un error al actualizar el stock. Ahora procedo a ejecutar con el fin de verificar si quedo correctamente desarrollado, entonces para ejecutar el procedimiento, simplemente ejecuto el siguiente comando donde voy a restar las unidades vendidas a los productos, es preciso recordar que en la inserción de datos en la tabla **registro_ventas**, las ventas fueron las siguientes: **Producto A:** Cantidad = 10, **Producto B:** Cantidad = 5, **Producto C:** Cantidad = 8, **Producto D:** Cantidad = 12. A continuación, la tabla “productos”, con el fin de dar claridad al Stock antes de su actualización:

ID_PRODUCTO	NOMBRE	PRECIO	STOCK
1	Producto A	50	100
2	Producto B	75	80
3	Producto C	60	120
4	Producto D	90	60

A continuación, el código a ejecutar para actualizar el Stock de todos los productos y la respuesta de la herramienta:

- **Producto A:**

```

1 BEGIN
2   actualizar_stock(1, 10);
3 END;

```

Results Explain Describe Saved SQL History

El stock del producto ID 1 ha sido actualizado.

Statement processed.

0.03 seconds

Ahora como se denota el producto A ahora tiene 90 unidades en Stock:

<

- **Producto B:** Cantidad = 5

```

1 BEGIN
2   actualizar_stock(2, 5);
3 END;

```

Results Explain Describe Saved SQL History

El stock del producto ID 2 ha sido actualizado.

Statement processed.

0.01 seconds

Ahora como se denota el producto B ahora tiene 75 unidades en Stock:

Asignatura	Datos del alumno	Fecha
Bases De Datos	Apellidos: De Mendoza	24/11/2024
	Nombre: Alejandro	

PRODUCTOS				
Columns	Data	Indexes	Constraints	Grants
Statistics	Triggers	Dependencies	DDL	Sample Queries
+ Insert Row Columns... Filter... Count Rows Load Data Download Refresh				
	ID_PRODUCTO	NOMBRE	PRECIO	STOCK
	3	Producto C	60	120
	4	Producto D	90	60
	1	Producto A	50	90
	2	Producto B	75	75

- Producto C: Cantidad = 8

```

1 BEGIN
2   actualizar_stock(3, 8);
3 END;

```

Results Explain Describe Saved SQL History

El stock del producto ID 3 ha sido actualizado.

Statement processed.

0.00 seconds

Ahora como se denotar el producto C ahora tiene 112 unidades en Stock:

PRODUCTOS				
Columns	Data	Indexes	Constraints	Grants
Statistics	Triggers	Dependencies	DDL	Sample Queries
+ Insert Row Columns... Filter... Count Rows Load Data Download Refresh				
	ID_PRODUCTO	NOMBRE	PRECIO	STOCK
	3	Producto C	60	112
	4	Producto D	90	60
	1	Producto A	50	90
	2	Producto B	75	75

- Producto D: Cantidad = 12

```

1 BEGIN
2   actualizar_stock(4, 12);
3 END;

```

Results Explain Describe Saved SQL History

El stock del producto ID 4 ha sido actualizado.

Statement processed.

0.00 seconds

Ahora como se denotar el producto D ahora tiene 48 unidades en Stock:

Asignatura	Datos del alumno	Fecha
Bases De Datos	Apellidos: De Mendoza	24/11/2024
	Nombre: Alejandro	

PRODUCTOS				
Columns	Data	Indexes	Constraints	Grants
Statistics	Triggers	Dependencies	DDL	Sample Queries
+ Insert Row Columns... Filter... Count Rows Load Data Download Refresh				
	ID_PRODUCTO	NOMBRE	PRECIO	STOCK
	3	Producto C	60	112
	4	Producto D	90	48
	1	Producto A	50	90
	2	Producto B	75	75

Es por esto y como se puede ver, tanto el procedimiento como la ejecución del mismo quedo correcta.

CREACIÓN DE UN DISPARADOR (TRIGGER)

A continuación, creo el disparador llamado `trg_actualizar_stock`, que se activa antes de insertar una fila en la tabla `registro_ventas`. Es importante precisar que en este momento este disparador no realiza validaciones ni modificaciones adicionales, por esto, luego que cree este disparador entonces ahí si procedo a utilizar este disparador para llamar al procedimiento "actualizar_stock" con los valores del nuevo producto y cantidad a insertar, permitiendo actualizar automáticamente el stock antes de registrar la venta.

Creación Del Disparador (TRIGGER)

Entonces procedo a crear el disparador llamado "trg_actualizar_stock", que se activa antes de insertar una fila en la tabla "registro_ventas". Entonces el código es el siguiente:

```

1 CREATE OR REPLACE TRIGGER trg_actualizar_stock
2 BEFORE INSERT ON registro_ventas
3 FOR EACH ROW
4 BEGIN
5     NULL;
6 END;

```

Results Explain Describe Saved SQL History

Trigger created.

0.04 seconds

La explicación del código es la siguiente: **CREATE OR REPLACE TRIGGER trg_actualizar_stock**: Define el nombre del disparador como `trg_actualizar_stock` y lo reemplaza si ya existe. **BEFORE INSERT ON registro_ventas**: Especifica que el disparador se ejecutará antes de que se inserte una fila en la tabla `registro_ventas`. **FOR EACH ROW**: Indica que el disparador se ejecuta para cada fila que se intente insertar. **BEGIN ... END**: Es el bloque donde se definen las acciones del disparador. En este caso, no realiza ninguna acción concreta. Se utiliza `NULL`; para indicar que no hay lógica adicional.

Disparador Para Llamar Al Procedimiento "actualizar_stock" Con Valores Del Nuevo Producto Y Cantidad A Insertar

Ahora procedo a aplicar funcionalidad al disparador con el fin de llamar al procedimiento "actualizar_stock" con valores del nuevo producto y cantidad a insertar, entonces el código actualizado es el siguiente:

Asignatura	Datos del alumno	Fecha
Bases De Datos	Apellidos: De Mendoza	24/11/2024
	Nombre: Alejandro	

```

1 CREATE OR REPLACE TRIGGER trg_actualizar_stock
2 BEFORE INSERT ON registro_ventas
3 FOR EACH ROW
4 BEGIN
5     actualizar_stock(:NEW.id_producto, :NEW.cantidad);
6 END;
```

Results Explain Describe Saved SQL History

Trigger created.

0.07 seconds

La explicación del código es: **CREATE OR REPLACE TRIGGER trg_actualizar_stock**: Define el nombre del disparador como trg_actualizar_stock y lo reemplaza si ya existe. **BEFORE INSERT ON registro_ventas**: Especifica que el disparador se ejecutará antes de que se inserte una fila en la tabla registro_ventas. **FOR EACH ROW**: Aplica el disparador a cada fila que se intente insertar. **actualizar_stock (:NEW.id_producto, :NEW.cantidad)**: Llama al procedimiento actualizar_stock, pasando como parámetros **:NEW.id_producto**: El ID del producto de la nueva fila que se va a insertar. **:NEW.cantidad**: La cantidad de la nueva fila que se va a insertar. **END**: Finaliza el bloque del disparador. A continuación, un ejemplo para determinar su aplicabilidad correcta, y para esto voy a proceder con una inserción en la tabla “registro_ventas” con el siguiente código actualizando el Stock del producto A cuyo ID es el numero 1 con una cantidad vendida de 5 unidades:

```

1 INSERT INTO registro_ventas (id_venta, id_producto, cantidad, fecha_venta)
2 VALUES (1, 1, 5, SYSDATE);
```

Donde, lo primero que debe suceder es que el disparador se active automáticamente, luego Llama al procedimiento actualizar_stock con: p_id_producto = 1 y p_cantidad_vendida = 5, para que finalmente el procedimiento se ejecute y actualice el stock del producto con ID 1 en la tabla productos. A continuación, la imagen de la ejecución:

```

1 INSERT INTO registro_ventas (id_venta, id_producto, cantidad, fecha_venta)
2 VALUES (1, 1, 5, SYSDATE);
```

Results Explain Describe Saved SQL History

El stock del producto ID 1 ha sido actualizado.

1 row(s) inserted.

0.02 seconds

A continuación, la imagen de la tabla “registro_ventas”, con la información actualizada:

REGISTRO_VENTAS

Columns

Data

Indexes

Constraints

Grants

Statistics

Triggers

Dependencies

DDL

Sample Queries

+ Insert Row

Columns...

Filter...

Count Rows

Load Data

Download

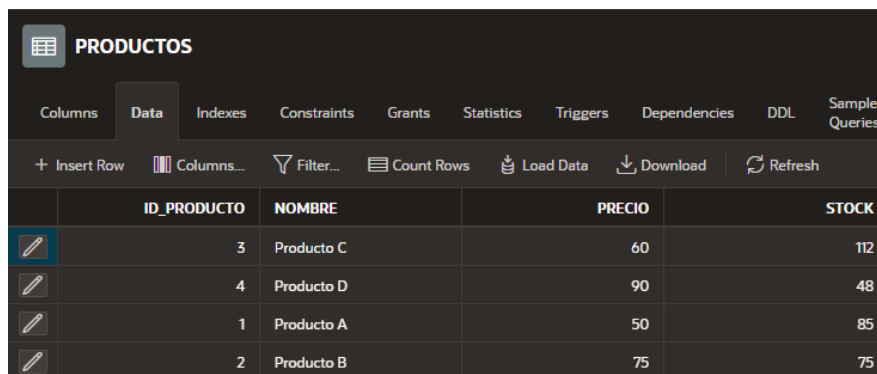
Refresh

	ID_VENTA	ID_PRODUCTO	CANTIDAD	FECHA_VENTA
	1	1	10	11/24/2024
	2	2	5	11/24/2024
	3	3	8	11/24/2024
	1	1	5	11/25/2024
	4	4	12	11/24/2024

Para finalizar y como se puede denotar se agregó una fila en la tabla “registro_ventas” arriba de las ventas del producto D con una fecha del 25/11/24 por un total de 5 unidades. Ahora para finalizar a continuación la imagen

Asignatura	Datos del alumno	Fecha
Bases De Datos	Apellidos: De Mendoza	24/11/2024
	Nombre: Alejandro	

de la tabla “productos” donde el Stock del producto A debió pasar de 90 unidades (recordar Creación Del Procedimiento “actualizar_stock” donde las unidades de Stock se actualizaron y quedaron en un total de 90 luego de una venta ejecutada de 10 unidades en la inserción de datos en la tabla registro_ventas) a 85 unidades con la venta de estas 5 unidades. A continuación, la imagen respectiva de la tabla “productos”:



	ID_PRODUCTO	NOMBRE	PRECIO	STOCK
	3	Producto C	60	112
	4	Producto D	90	48
	1	Producto A	50	85
	2	Producto B	75	75

Y como se puede denotar el Stock del producto A quedo en 85 unidades por lo que esta correcta la ejecución.

CONCLUSIÓN

A lo largo de esta actividad, he aplicado conceptos fundamentales de bases de datos y programación PL/SQL para gestionar datos relacionados con productos y ventas. Como pude determinar el objetivo principal de la actividad fue diseñar y automatizar procesos que optimicen el manejo del stock de productos mientras se registran las ventas, garantizando una base sólida para la integridad y la consistencia de los datos. Reforcé la estructura de tablas como productos y registro_ventas, estableciendo columnas clave para almacenar información relevante de productos, cantidades, y fechas de venta. Desarrolle la creación del procedimiento calcular_total_ventas para calcular automáticamente el total de ventas utilizando un cursor lo que me demuestra el poder de PL/SQL para manejar cálculos complejos directamente en la base de datos. Ahora, la función obtener_stock_producto me permitió obtener dinámicamente el stock actual de un producto según su ID, mostrando cómo PL/SQL puede generar consultas reutilizables y modulares. Frente a el procedimiento actualizar_stock lo utilicé para ajustar automáticamente el stock de productos con cada venta registrada lo cual considero es clave para mantener la precisión de los datos. Finalmente, Implemente el disparador trg_actualizar_stock para integrar automáticamente la lógica de actualización del stock antes de insertar una venta, lo que a mi parecer minimiza errores manuales y asegura la consistencia de los datos. Para concluir, debo indicar que los disparadores y procedimientos almacenados son herramientas esenciales para automatizar procesos críticos en bases de datos, reduciendo errores manuales, simplificando las operaciones, realizando operaciones como cálculos y actualizaciones directamente en la base de datos lo que reduce la carga en las aplicaciones externas y mejora el rendimiento general.

BIBLIOGRAFÍA

A continuación, la bibliografía implementada:

- ✓ Tema 7: Tema 7. SQL: Consultas complejas y vistas. Tema 8. SQL: Subrutinas y disparadores. Tema 9. Diseño de base de datos y el modelo entidad-relación.
- ✓ Clases virtuales con el profesor Ing. Javier Diaz Diaz.
- ✓ Material de estudio del aula.