Desarrollo de un Paquete de Administración Avanzada para Productos y Activos (Parte avanzada)

Contexto: (Mismo contexto que en la parte básica)

Se necesitan utilidades PL/SQL más sofisticadas para tareas de administración que involucran lógica de negocio compleja.

Nombre del Paquete:

PKG_ADMIN_PRODUCTOS_AVANZADO

Objetivo General:

Crear un paquete PL/SQL (PKG_ADMIN_PRODUCTOS_AVANZADO) que contenga funciones y procedimientos para realizar operaciones que *no* se pueden implementar directamente con una sola sentencia SQL, sino que requieren lógica de programación más elaborada.

Funciones Requeridas:

1. F_VALIDAR_PLAN_SUFICIENTE(p_cuenta_id IN CUENTA.ID%TYPE) RETURN VARCHAR2
    - **Parámetros de Entrada:**
        - p_cuenta_id: El ID de la cuenta.
    - **Tipo de Retorno:** VARCHAR2
    - **Descripción:** Esta función debe determinar si el plan actual de una cuenta es suficiente para los recursos que está utilizando. Debe realizar las siguientes comprobaciones *en orden*:
        - Obtener el plan actual de la cuenta (usar la función F_OBTENER_PLAN_CUENTA del paquete anterior, si está disponible; si no, reimplementar esa lógica *dentro* de esta función).
        - Contar la cantidad actual de productos, activos, categorías de producto, categorías de activos y relaciones que tiene la cuenta. *No* usar funciones del paquete anterior para estos conteos; calcularlos directamente dentro de esta función.
        - Comparar los conteos actuales con los límites permitidos por el plan (PRODUCTOS, ACTIVOS, CATEGORIASPRODUCTO, CATEGORIAS_ACTIVOS, RELACIONES en la tabla PLAN).
        - **Retornar:**
            - 'SUFICIENTE': Si todos los conteos están dentro de los límites del plan.
            - 'INSUFICIENTE: [RecursoExcedido]': Si *algún* conteo excede el límite del plan. [RecursoExcedido] debe ser reemplazado por el nombre del recurso que excede el límite (por ejemplo, 'INSUFICIENTE: PRODUCTOS', 'INSUFICIENTE: ACTIVOS'). Si

múltiples recursos exceden el límite, devolver el *primero* que se encuentre.

- **Manejo de Errores:**
  - Si la cuenta no existe, lanzar NO_DATA_FOUND.
  - Si la cuenta no tiene un plan asociado, lanzar una excepción personalizada EXCEPTION_PLAN_NO_ASIGNADO.
  - Otros errores: captura genérica, registro y relanzamiento.
- **Sugerencia:** Usa variables locales para almacenar los conteos y los límites del plan. Usa estructuras condicionales (IF-THEN-ELSIF-ELSE) para realizar las comparaciones.

2. F_LISTA_CATEGORIAS_PRODUCTO(p_producto_gtin IN PRODUCTO.GTIN%TYPE, p_cuenta_id IN PRODUCTO.CUENTA_ID%TYPE) RETURN VARCHAR2

- **Parámetros:**
- p_producto_gtin: El GTIN del producto.
- p_cuenta_id: El ID de la cuenta.
- **Tipo de retorno:** VARCHAR2
- **Descripción:** Esta función devuelve una cadena que representa la lista completa de categorías a las que pertenece un producto.
- **Lógica:**
- Verificar que el producto exista (para el GTIN y cuenta dados); lanzar NO_DATA_FOUND si no.
- Obtener *todas* las categorías a las que pertenece el producto (a través de la tabla PROD_CAT).
- Concatenar los nombres de las categorías en una sola cadena, separados por un delimitador (por ejemplo, ' ; '). El orden de las categorías en la cadena no es importante. Ejemplo: "Electrónica ; Móviles ; Regalo día de la Madre ".
- Si el producto no está en ninguna categoría, devolver una cadena vacía ('') o un mensaje como 'Sin categoría'.
- **Manejo de errores:**
- Si el producto no existe, lanzar NO_DATA_FOUND.
- Otras excepciones: captura genérica, registro y relanzamiento.
- **Sugerencia:** Usa un cursor explícito (o un FOR loop con una consulta) para obtener las categorías. Dentro del bucle, concatena los nombres.

Procedimientos Requeridos:

1. P_MIGRAR_PRODUCTOS_A_CATEGORIA(p_cuenta_id IN CUENTA.ID%TYPE, p_categoria_origen_id IN CATEGORIA.ID%TYPE, p_categoria_destino_id IN CATEGORIA.ID%TYPE)

- **Parámetros:**
  - p_cuenta_id: El ID de la cuenta.
  - p_categoria_origen_id: El ID de la categoría de origen.
  - p_categoria_destino_id: El ID de la categoría de destino.

- **Descripción:** Mueve *todos* los productos de una categoría de origen a una categoría de destino, *dentro de la misma cuenta*.
- **Lógica:**
  - Verificar que la cuenta, la categoría de origen y la categoría de destino existan. Lanzar NO_DATA_FOUND si alguna no existe. Verificar que las categorías de origen y destino pertenezcan a la cuenta dada.
  - Obtener todos los productos que están actualmente en la categoría de origen (usando PROD_CAT).
  - Para cada producto:
    - Modificar la categoría de origen por la de destino.
  - **Importante:** Realizar todas las operaciones dentro de una sola transacción. Si ocurre *cualquier* error, hacer un ROLLBACK.
- **Manejo de Errores:**
  - NO_DATA_FOUND si la cuenta, categoría de origen o destino no existen.
  - Captura genérica y relanzamiento de otras excepciones, después de hacer ROLLBACK.
- **Sugerencia:** Usa un cursor explícito (o un bucle FOR con una consulta) para recorrer los productos. Pensar en un cursor SELECT … FOR UPDATE …

2. P_REPLICAR_ATRIBUTOS(p_cuenta_id IN CUENTA.ID%TYPE, p_producto_gtin_origen IN PRODUCTO.GTIN%TYPE, p_producto_gtin_destino IN PRODUCTO.GTIN%TYPE)

- **Parámetros de Entrada:**
  - p_cuenta_id: El ID del atributo de origen.
  - p_producto_gtin_origen: El ID del producto de origen.
  - p_producto_gtin_destino: El ID del producto de destino.
- **Descripción:** Copia los atributos de un producto a otro producto, incluyendo sus valores.
- **Lógica:**
  - Verificar que ambos productos (origen y destino) existan. Lanzar NO_DATA_FOUND si alguno no existe.
  - Obtener todos los registros de la tabla ATRIBUTO_PRODUCTO donde PRODUCTO_ID sea igual a p_producto_gtin_origen.
  - Para cada uno de esos registros:
    - Verificar si ya existe un registro en ATRIBUTO_PRODUCTO para el *mismo* producto (mismo p_producto_gtin_destino y PRODUCTO_CUENTA_ID y Código de atributo).
    - Si *no* existe, insertar un nuevo registro en ATRIBUTO_PRODUCTO, pero con p_producto_gtin_destino.
    - Si *sí* existe, *actualizar* el registro existente en ATRIBUTO_PRODUCTO, estableciendo su VALOR al valor del registro de origen.
  - Usar una transacción para asegurar la consistencia.
- **Manejo de Errores:**

- NO_DATA_FOUND si alguno de los atributos no existe.
- Otras excepciones: captura genérica, registro (después de ROLLBACK) y relanzamiento.

- **Sugerencia**: Utiliza un cursor explícito (o un FOR loop con una consulta) para iterar sobre los registros del atributo origen. Dentro del bucle, usa una subconsulta para verificar la existencia del atributo destino, y luego un INSERT o UPDATE según corresponda.

Consideraciones Adicionales: (Mismas que antes)

- **Manejo de Excepciones:** Robusto y específico.
- **Buenas Prácticas:** Nombres descriptivos, comentarios, sangría, evitar SELECT *.
- **Pruebas:** Después de implementar el paquete, escribe casos de prueba exhaustivos para cada función y procedimiento. Estos casos de prueba deben ser más complejos ahora, cubriendo la lógica adicional y las interacciones entre tablas.

JOBS

1. J_LIMPIA_TRAZA**:** Limpia las entradas de la tabla TRAZA que tengan más de 1 año. Para probarlo se pueden hacer con las que tengan más de un minuto y luego modificarlo.
2. J_ACTUALIZA_PRODUCTOS. Actualiza desde la tabla de productos externos los productos de la tabla Productos para todas las cuentas de la base de datos llamando a P_ACTUALIZAR_PRODUCTOS