Problemática Propuesta: Sistema de Reportería y Fidelización de Clientes para "Chilegames"

Contexto de Negocio:

Tu empresa ficticia, "Chilegames", ha crecido rápidamente. La gerencia necesita entender mejor a sus clientes para lanzar un nuevo programa de fidelización. Actualmente, no tienen una forma automatizada de ver el historial de compras de un cliente ni de clasificarlo según su comportamiento de compra. Además, el control de stock es manual y propenso a errores.

Objetivo del Proyecto:

Desarrollar un sistema en la base de datos VideojuegosDB que permita generar un informe detallado de las compras de cualquier cliente y, al mismo tiempo, automatice la actualización del stock y la clasificación de los clientes para el nuevo programa de fidelización.

Guía de Solución Orientada a tu Evaluación

Aquí te detallo cómo abordar esta problemática cumpliendo con cada punto de la pauta de evaluación (BDY1103 Evaluación Parcial 1\_EE\_EP\_Estudiante.pdf).

1. Tipos de Datos Compuestos (RECORD y VARRAY)

El Problema a Resolver: Necesitas crear una estructura que pueda manejar la información de un cliente y una lista variable de todas las compras que ha realizado.

Cómo Orientarte:

TYPE para un Registro de Venta (RECORD):

Crea un tipo RECORD llamado rec\_detalle\_venta\_t que almacene los datos más importantes de una venta: el título del videojuego, la cantidad, el precio unitario y la fecha de la venta. Esto te permitirá manejar cada compra como un objeto único y ordenado.

TYPE para una Colección de Ventas (VARRAY):

Crea un tipo VARRAY llamado varray\_ventas\_t que sea una colección del RECORD que creaste antes. Este VARRAY te servirá para almacenar todas las compras de un cliente en una sola variable.

Bloque PL/SQL Anónimo (Para el Informe):

Crea un bloque anónimo donde declares una variable del tipo RECORD para guardar los datos del cliente (nombre, email, fecha de registro) y otra variable del tipo VARRAY para guardar su historial de compras. Esto cumple directamente con el indicador

IE1.1.1.

2. Cursores Explícitos Complejos y Loops Anidados

El Problema a Resolver: Necesitas recorrer la lista de clientes y, para cada uno, recorrer su historial de ventas específico para generar un reporte.

Cómo Orientarte:

Cursor Principal (Clientes):

Declara un cursor explícito llamado c\_clientes que seleccione el cliente\_id, nombre y apellido de todos los clientes en la tabla cliente.

Cursor Anidado con Parámetros (Ventas por Cliente):

Declara un segundo cursor explícito llamado c\_ventas\_por\_cliente que acepte un cliente\_id como parámetro. Este cursor seleccionará la información de las ventas (título del juego, cantidad, precio, fecha) uniendo las tablas venta, detalle\_venta y videojuego para ese cliente específico.

Implementación con Loops Anidados:

En tu bloque PL/SQL, abre un LOOP para recorrer el cursor c\_clientes.

Dentro de ese loop, abre un segundo LOOP anidado que recorra el cursor c\_ventas\_por\_cliente, pasándole el cliente\_id del cliente actual como parámetro.

Dentro del segundo loop, llena tu VARRAY con los datos de cada venta. Al final del loop anidado (cuando ya tengas todas las ventas de ese cliente), imprime la información del cliente y el contenido del VARRAY. Esto aborda directamente el indicador

IE1.2.1.

3. Control de Excepciones (Predefinidas y de Usuario)

El Problema a Resolver: Tu código debe ser robusto. ¿Qué pasa si se busca un cliente que no existe? ¿O si un cliente no tiene ventas registradas? ¿O si una venta supera un monto sospechoso?

Cómo Orientarte:

Excepción Predefinida (NO\_DATA\_FOUND):

Cuando intentes obtener los datos de un cliente específico usando un SELECT ... INTO, si el cliente\_id no existe, Oracle lanzará un error. Captura esta excepción en el bloque EXCEPTION y muestra un mensaje amigable como "El cliente con ID X no fue encontrado".

Excepción Definida por el Usuario:

Declara una excepción personalizada llamada e\_cliente\_sin\_ventas.

Después de consultar las ventas de un cliente, verifica si tu VARRAY de ventas está vacío. Si lo está, lanza tu excepción personalizada usando RAISE e\_cliente\_sin\_ventas;.

En el bloque EXCEPTION, captura esta excepción y muestra el mensaje "El cliente no tiene ventas registradas". Esto demuestra un control de lógica de negocio y cumple con el indicador

IE1.3.1.

4. Procedimientos, Funciones, Packages y Triggers

Esta es la parte donde construyes una solución integral y reutilizable.

Cómo Orientarte:

Función Almacenada: FN\_CATEGORIZAR\_CLIENTE

Propósito: Calcular y devolver la categoría de un cliente ("Bronce", "Plata", "Oro") basado en el total de sus compras.

Lógica: La función recibirá un cliente\_id, calculará la suma total de sus compras (total\_venta) y, usando IF/ELSIF, devolverá un VARCHAR2 con la categoría.

Uso: La podrás llamar desde otros procedimientos o consultas para obtener rápidamente la categoría de un cliente.

Procedimiento Almacenado: SP\_GENERAR\_INFORME\_CLIENTE

Propósito: Encapsular toda la lógica de los cursores anidados y la impresión del informe que hiciste en los pasos anteriores.

Lógica: Recibirá un cliente\_id como parámetro. Dentro, ejecutará los cursores, llenará el RECORD y el VARRAY, y usará DBMS\_OUTPUT.PUT\_LINE para mostrar el informe completo, incluyendo la categoría del cliente obtenida llamando a la función FN\_CATEGORIZAR\_CLIENTE.

Package: PKG\_CLIENTES

Propósito: Agrupar de forma lógica y ordenada todos los componentes relacionados con la gestión de clientes. Esto es una práctica profesional fundamental y clave en la evaluación.

Estructura:

Especificación del Package (PACKAGE SPEC): Declara las firmas de la función FN\_CATEGORIZAR\_CLIENTE y del procedimiento SP\_GENERAR\_INFORME\_CLIENTE. Esta es la "cara pública" de tu paquete.

Cuerpo del Package (PACKAGE BODY): Contiene la implementación completa (el código) de la función y el procedimiento.

Ventaja: Permite modularizar tu código, facilita el mantenimiento y mejora la organización, tal como se pide en los puntos de la pauta.

Trigger: TRG\_ACTUALIZAR\_STOCK

Propósito: Automatizar la actualización del inventario para que el stock sea siempre correcto.

Lógica: Crea un trigger que se dispare después (AFTER) de cada inserción (INSERT) en la tabla detalle\_venta.

Acción: Por cada fila insertada en detalle\_venta (FOR EACH ROW), el trigger debe ejecutar una sentencia UPDATE sobre la tabla videojuego, restando la cantidad vendida del stock del videojuego\_id correspondiente. Esto demuestra cómo automatizar respuestas a eventos en la base de datos, cumpliendo con el indicador

IE1.4.1.

Siguiendo esta guía, no solo resolverás una problemática de negocio coherente con tu base de datos, sino que también abordarás de manera directa y ordenada cada uno de los puntos exigidos en tu evaluación parcial.