Informe Evaluación Parcial N°2

“ChileGames”

**Sección:**

*003D*

**Integrantes:**  
*Francisco Jesús Aránguiz Inostroza*

*Vicente Benjamín Alarcón Gallardo*

*Carlos Ignacio Bittner Navea*

**Docente:**

***German Gabriel Barrientos Tereucan***

**Fecha de entrega:**

*26/10/2025*

1. ***Introducción***

*Descripción del Proyecto: “ChileGames”*

Este proyecto se centra en la base de datos de "Chilegames", una empresa ficticia de venta de videojuegos. Tras un rápido crecimiento, la gerencia identificó la necesidad de implementar un sistema de reportería y un nuevo programa de fidelización de clientes. El estado anterior del proyecto carecía de automatización para el control de stock y de una clasificación formal del comportamiento de compra de los clientes.

El objetivo de esta segunda fase del proyecto es evolucionar los scripts iniciales de PL/SQL hacia una solución integral, robusta y modularizada. Esto implica la implementación de **Packages** para encapsular la lógica de negocio, la creación de **Procedimientos** de procesamiento masivo y la implementación de **Triggers** avanzados para garantizar la integridad y seguridad de los datos.

*Alcance*

El alcance de este proyecto abarca la creación de una API interna en la base de datos (Package) para gestionar la lógica de clientes, la automatización de procesos de mantenimiento (Procedimientos masivos) y la implementación de reglas de negocio automáticas (Triggers). El objetivo final es proveer a la gerencia de "Chilegames" las herramientas necesarias para lanzar su programa de fidelización y asegurar la integridad de su inventario.

*Tecnologías Utilizadas*

SGBD: Oracle Database

Lenguaje: PL/SQL

IDE: SQL Developer

1. ***Desarrollo de Procedimientos y Funciones Almacenadas***

*Propósito y Definición*

Las **Funciones** y **Procedimientos** son bloques de código PL/SQL con nombre que se almacenan en la base de datos. Su propósito principal es la **reutilización de código**, permitiendo ejecutar lógica de negocio compleja sin necesidad de reescribirla.

* Una **Función** siempre debe retornar un valor y puede ser utilizada en sentencias SQL.
* Un **Procedimiento** ejecuta una serie de acciones y no necesariamente retorna un valor.

*Implementación de Funciones*

Para el proyecto "Chilegames", se implementaron funciones con y sin parámetros para procesar información:

* **FN\_CATEGORIZAR\_CLIENTE (Con Parámetros):** Esta función (mejorada de la Evaluación 1) es clave para el programa de fidelización. Recibe un p\_cliente\_id como parámetro, calcula el monto total gastado por ese cliente y retorna una categoría ('BRONCE', 'PLATA', 'ORO').
* **FN\_TOTAL\_VENTAS\_TIENDA (Sin Parámetros):** Esta función demuestra el procesamiento de información masiva. No recibe parámetros y calcula la suma total de todas las ventas registradas en la tienda, siendo útil para reportes gerenciales rápidos.

*Implementación de Procedimientos*

Se implementaron procedimientos para encapsular tareas transaccionales y de procesamiento masivo:

* **SP\_GENERAR\_INFORME\_CLIENTE (Con Parámetros):** Este procedimiento es el núcleo de la reportería. Recibe un p\_cliente\_id y ejecuta una serie de consultas para imprimir en consola un informe completo del cliente, incluyendo sus datos personales y su historial de ventas.
* **SP\_RECALCULAR\_TOTALES\_VENTA (Sin Parámetros):** Este procedimiento demuestra el procesamiento masivo. Recorre *todas* las ventas de la base de datos y recalcula su campo TOTAL\_VENTA basándose en la suma de sus detalles. Esto asegura la integridad de los datos ante posibles modificaciones manuales.

*Justificación y ventajas*

La implementación de estas funciones y procedimientos es fundamental por varias razones:

* **Modularidad:** El código se divide en bloques lógicos y reutilizables.
* **Mantenimiento:** Si la lógica de "Bronce/Plata/Oro" cambia, solo se modifica la función FN\_CATEGORIZAR\_CLIENTE en un lugar, y todos los programas que la usan se actualizan automáticamente.
* **Rendimiento:** El código PL/SQL almacenado se ejecuta en el servidor de la base de datos, reduciendo el tráfico de red.
* **Seguridad:** Se pueden otorgar permisos para ejecutar un procedimiento sin necesidad de dar acceso directo a las tablas subyacentes.

1. ***Desarrollo de Packages***

*Propósito y Definición*

Un **Package** (Paquete) es un objeto de esquema de Oracle que agrupa lógicamente tipos, variables, funciones y procedimientos relacionados. Es la herramienta fundamental de PL/SQL para lograr el **encapsulamiento** y el modularidad a gran escala.

Se componen de dos partes:

1. **Especificación (SPEC):** La "cara pública" o API. Declara los elementos que pueden ser llamados desde fuera del paquete.
2. **Cuerpo (BODY):** La "cara privada". Contiene la implementación real del código y puede incluir elementos (funciones/procedimientos) que son *privados*, es decir, solo pueden ser llamados por otro código dentro del mismo cuerpo del paquete.

*Implementación del Package*

Para profesionalizar la solución de "Chilegames", se migró toda la lógica de negocio relacionada con clientes al PKG\_CLIENTES:

* **Especificación (Pública):**
  + Se declararon las firmas de FUNCTION FN\_CATEGORIZAR\_CLIENTE y PROCEDURE SP\_GENERAR\_INFORME\_CLIENTE. Estos son los únicos dos elementos que el resto de la aplicación puede "ver" y utilizar.
* **Cuerpo (Privado y Público):**
  + Se implementó el código de las dos funciones públicas declaradas.
  + **Elemento Privado:** Se creó la función FN\_CALCULAR\_TOTAL\_COMPRADO dentro del *body* (y *no* en la *spec*). El propósito de esta función es calcular el total gastado, pero esta lógica queda **oculta** al exterior. La función pública FN\_CATEGORIZAR\_CLIENTE es la única que puede llamar a esta función privada.

*Justificación y Ventajas*

El uso de PKG\_CLIENTES es una mejora sustancial respecto para tener funciones y procedimientos "sueltos" (como en la Evaluación 1):

* **Organización:** Toda la lógica de "Clientes" está en un solo lugar.
* **Ocultamiento de la Complejidad (Encapsulamiento):** El usuario (u otro programa) solo necesita saber que existe FN\_CATEGORIZAR\_CLIENTE. No necesita saber *cómo* se calcula ese total; esa lógica está protegida y oculta en la función privada FN\_CALCULAR\_TOTAL\_COMPRADO.
* **Mantenimiento:** Si la forma de calcular el total cambia, solo se modifica el *body* del paquete. La *spec* (la "firma" pública) no cambia, por lo que ningún programa que la llame se verá afectado.
* **Gestión de Permisos:** Es mucho más sencillo conceder permiso de ejecución sobre un solo paquete (GRANT EXECUTE ON PKG\_CLIENTES TO...) que sobre múltiples objetos individuales.

1. ***Desarrollo de Triggers***

*Propósito y Definición*

Un **Trigger** (Disparador) es un bloque PL/SQL que se ejecuta automáticamente cuando ocurre un evento DML (INSERT, UPDATE, DELETE) en una tabla específica. Su propósito es automatizar reglas de negocio, auditoría o mantener la integridad de los datos.

Existen dos niveles principales:

1. **Nivel de Fila (Row-Level):** Se ejecuta *una vez por cada fila* afectada por la sentencia.
2. **Nivel de Sentencia (Statement-Level):** Se ejecuta *una sola vez por la sentencia*, sin importar cuántas filas afecte.

*Implementación de Triggers*

En el proyecto se implementaron ambos tipos de triggers para distintas necesidades:

* **TRG\_ACTUALIZAR\_STOCK (Nivel de Fila):**
  + Este trigger se dispara AFTER INSERT ON DETALLE\_VENTA y se define con FOR EACH ROW.
  + Su acción es tomar la cantidad vendida de la fila que se acaba de insertar (: NEW.CANTIDAD) y restarla del stock en la tabla VIDEOJUEGO.
* **TRG\_AUDIT\_HORARIO\_VIDEOJUEGO (Nivel de Sentencia):**
  + Este trigger se dispara BEFORE INSERT OR UPDATE OR DELETE ON VIDEOJUEGO.
  + **No** lleva la cláusula FOR EACH ROW.
  + Su acción es verificar la hora y día actuales. Si la operación se intenta realizar fuera de horario laboral (lunes a viernes, 9:00-18:00), el trigger lanza una excepción (RAISE\_APPLICATION\_ERROR) y detiene la transacción.

*Justificación y Ventajas*

La utilización de ambos niveles de trigger es crucial:

* **Justificación de TRG\_ACTUALIZAR\_STOCK (Nivel de Fila):** Es obligatorio que sea a nivel de fila porque necesita acceder a los datos específicos de *cada* producto vendido (el :NEW.CANTIDAD y :NEW.VIDEOJUEGO\_ID). Si fuera a nivel de sentencia, no sabría qué productos ni qué cantidades restar. Garantiza la **integridad de los datos** del inventario.
* **Justificación de TRG\_AUDIT\_HORARIO\_VIDEOJUEGO (Nivel de Sentencia):** Es más eficiente que sea a nivel de sentencia porque la regla de negocio (validar el horario) es la misma para todas las filas. Si se hiciera un UPDATE masivo que afectara a 1.000 videojuegos, este trigger se ejecuta **solo una vez**, en lugar de 1.000 veces. Garantiza la **seguridad y las reglas de negocio**.

**Desventaja:** El principal riesgo de los triggers es la complejidad oculta. Un DML puede fallar por un trigger y no ser obvio para el desarrollador, y un trigger mal diseñado puede causar problemas de rendimiento.

1. ***Conclusión***

*Resumen del Trabajo*

En esta segunda evaluación, el proyecto "Chilegames" evolucionó de un conjunto de scripts PL/SQL individuales a una solución de software robusta y modularizada. Se implementaron funciones y procedimientos con y sin parámetros para manejar la lógica de negocio y el procesamiento masivo. La lógica de cliente fue encapsulada exitosamente en el PKG\_CLIENTES, demostrando el uso de elementos públicos y privados. Finalmente, se implementaron triggers a nivel de fila y de sentencia para asegurar la integridad de los datos del inventario y las reglas de seguridad de la empresa.

*Impacto del Proyecto*

El impacto de estas mejoras es inmediato:

1. **Sistema de Fidelización:** La gerencia ahora puede usar PKG\_CLIENTES.FN\_CATEGORIZAR\_CLIENTE para clasificar a sus clientes y lanzar el programa de fidelización.
2. **Reportería:** El procedimiento PKG\_CLIENTES.SP\_GENERAR\_INFORME\_CLIENTE provee un informe estandarizado y rápido.
3. **Integridad de Datos:** El stock se actualiza automáticamente (TRG\_ACTUALIZAR\_STOCK) y los totales de venta pueden ser auditados (SP\_RECALCULAR\_TOTALES\_VENTA).
4. **Mantenibilidad:** El código está organizado, encapsulado y es mucho más fácil de mantener y escalar a futuro .

*Recomendaciones a Futuro*

Para la siguiente etapa del proyecto, se recomienda:

1. **Crear un PKG\_INVENTARIO:** De forma similar al paquete de clientes, crear un paquete que encapsule la lógica de videojuegos (ej. SP\_AGREGAR\_JUEGO, FN\_VERIFICAR\_STOCK, etc.).
2. **Triggers de Auditoría:** Implementar triggers adicionales que guarden los cambios de precio o stock en una tabla de auditoría (log) para un control histórico.
3. **Interfaz de Usuario:** Desarrollar una aplicación (Web o de escritorio) que consuma los paquetes PL/SQL creados en la base de datos.

***6. Evidencias***

\* **Todas las evidencias de código se encuentran en el repositorio que se tuvo que haber hecho llegar junto con este documento.**