Teoría de base de datos 1

Nombre del Trabajo

Documentación del proyecto

Proyecto

Sistema de gestión de presupuestos personal

Tecnologías principales

Oracle, Python, TKinter, Power BI

Nombres de los integrantes y números de cuenta

Jorge Discua - 22311096

Edgar Vásquez - 22211135

Docente

Ing. Elvin Deras

Fecha de entrega

15/12/25

1. Introducción

1.1 Contexto del problema

Hoy en día, gestionar las finanzas personales se ha convertido en un desafío constante para la mayoría de la población. Si bien la mayoría de las personas generan ingresos periódicos, existe una dificultad constante para mantener registros claros y estructurados de dónde se gasta el dinero, cuánto se ahorra y si las decisiones financieras que se toman son o serán sostenibles a mediano y largo plazo.

Gran parte de este problema se debe a la falta de herramientas adecuadas para registrar y analizar la información financiera personal. Muchas personas recurren a métodos informales, como anotaciones en papel, aplicaciones generales o incluso la memoria, lo que genera errores, omisiones y una visión incompleta de su situación económica real. Esta falta de control detallado generalmente resulta en gastos innecesarios, desequilibrio entre ingresos y gastos, incapacidad para cumplir con los objetivos financieros e incluso, en casos extremos, en endeudamiento continuo.

Por otro lado, aunque existen aplicaciones comerciales para la gestión de finanzas personales, estas suelen presentar varias limitaciones, como que algunas son demasiado complejas para el usuario promedio; otras requieren suscripciones pagadas, y la gran mayoría no permite personalización ad hoc según las necesidades específicas de cada usuario. Además, en contextos académicos o de aprendizaje, estas herramientas no permiten analizar ni comprender la lógica interna de cómo se gestionan los datos, lo que limita su utilidad como medio formativo.

En este contexto, surge la necesidad de desarrollar un Sistema de Gestión de Presupuestos Personales cuyo objetivo principal sea facilitar, de forma clara, estructurada y accesible, el registro, la organización y el análisis de los ingresos y gastos del usuario. Este proyecto busca simular un escenario real de gestión de finanzas personales donde el usuario pueda crear categorías de gastos, registrar movimientos financieros, establecer presupuestos y estudiar su comportamiento económico a lo largo del tiempo.

Desde una perspectiva académica, el presente proyecto responde a la necesidad de aplicar los conocimientos teóricos adquiridos en temas como bases de datos, modelado de información,

programación y análisis de datos, integrándolos en una solución práctica que represente un problema real. La gestión financiera personal es aún más relevante de desarrollar, ya que puede implicar datos sensibles, las relaciones entre entidades, la validación de la información y la creación de informes, lo que la convierte en un caso práctico ideal para el desarrollo de sistemas de información.

Al mismo tiempo, se apoya en la visualización de datos como herramienta clave en la toma de decisiones. No basta con almacenar información financiera; es necesario transformarlo en informes y gráficos a través de los cuales se puedan observar patrones, tendencias y áreas de mejora. En relación a esto, se incorpora el uso de herramientas de análisis y visualización como Power BI, que permite al usuario tener una interpretación más intuitiva de su comportamiento financiero. En pocas palabras, el proyecto nace como respuesta a una problemática real y cotidiana: la dificultad de llevar un control financiero personal eficiente. Al mismo tiempo, sirve como un ejercicio integral de desarrollo de software y análisis de datos, permitiendo aplicar conocimientos técnicos en un contexto práctico, comprensible y de alto valor tanto académico como personal.

1.2 Justificación del proyecto

La elaboración del Sistema de Gestión de Presupuesto Personal se justifica por la necesidad de contar con una herramienta que permita organizar y analizar la información financiera de forma clara y estructurada. Una gestión adecuada de los ingresos y gastos garantiza la estabilidad económica, pero no todas las familias cuentan con métodos adecuados para supervisar de cerca sus finanzas, lo que dificulta la toma de decisiones.

Este proyecto permite aplicar los conocimientos adquiridos en los campos de bases de datos, programación y análisis de información. El problema en cuestión representa un enfoque muy realista, que implica la gestión de datos, la elaboración de estructuras relacionales y la generación de informes, todo lo cual contribuye al desarrollo de habilidades técnicas esenciales en la formación profesional.

Además, el sistema mejora la comprensión del comportamiento financiero mediante la aplicación de informes y visualizaciones. Permite identificar patrones de gasto, comparar ingresos y gastos, y analizar el cumplimiento presupuestario. Esta capacidad analítica aporta

un valor significativo a este proyecto, ya que datos sencillos se convierten en información útil para la toma de decisiones.

Finalmente, el sistema se justifica como una solución didáctica y funcional que muestra cómo el manejo y visualización adecuada de los datos puede apoyar la gestión financiera personal, sirviendo al mismo tiempo como un ejercicio integral de desarrollo y análisis de sistemas de información en un entorno académico.

1.3 Objetivos del proyecto

1.3.1 Objetivo general

Desarrollar un sistema de gestión de presupuestos personales que permita registrar, organizar y analizar la información financiera de un usuario, facilitando el control de ingresos y gastos mediante el uso de estructuras de datos y herramientas de visualización que apoyen la toma de decisiones financieras.

1.3.2 Objetivos específicos

- Diseñar una estructura de datos adecuada para el almacenamiento de información financiera, incluyendo usuarios, categorías, presupuestos y movimientos financieros.

- Implementar mecanismos para el registro de ingresos y gastos, permitiendo su clasificación por categoría y periodo de tiempo.

- Establecer presupuestos planificados y comparar dichos valores con los gastos reales registrados, con el fin de evaluar el comportamiento financiero del usuario.

- Generar reportes y visualizaciones que representen de forma clara el estado financiero, incluyendo ingresos, gastos, balance y distribución por categorías.

- Facilitar el análisis del comportamiento financiero a lo largo del tiempo mediante la representación gráfica de los datos históricos.

- Aplicar conceptos teóricos de bases de datos y análisis de información en un proyecto práctico que simule un escenario real de gestión financiera personal.

2. Alcance del Proyecto

El objetivo del presente proyecto es profundizar en el diseño y desarrollo de un Sistema de Gestión de Presupuesto Personal, dirigido al registro, organización y análisis de los datos financieros de un usuario a nivel individual. El sistema está conceptualizado como uno que apoyaría la toma de decisiones en finanzas, permitiendo al usuario percibir claramente los ingresos, los gastos y el comportamiento económico a lo largo del tiempo.

Dentro del alcance funcional del sistema se cuenta con la administración de usuarios, permitiendo identificar a cada usuario como entidad independiente dentro del sistema. Para cada uno de ellos, el sistema contempla la posibilidad de registrar diferentes movimientos financieros, que pueden clasificarse como ingresos o gastos y asociarse a categorías previamente definidas. Estas categorías permiten agrupar los movimientos según su naturaleza, tales como alimentación, transporte, servicios, entretenimiento, entre otros, facilitando así el análisis posterior.

El proyecto abarcará también la fijación y gestión presupuestaria, entendida como las cantidades previstas para un período de tiempo determinado. Estos presupuestos permiten establecer el límite financiero y comparar el gasto real con lo planificado, proporcionando un punto de referencia claro del nivel de control financiero por parte del usuario. Si bien el sistema permite su almacenamiento y análisis, su propósito es principalmente analítico y de soporte, no la ejecución automática de restricciones de gasto.

En cuanto a la gestión de la información, el alcance de este proyecto incluye el diseño de un modelo de datos, la definición de las tablas, relaciones y estructuras necesarias para el almacenamiento coherente de la información financiera. Se consideran aspectos básicos de la integridad de los datos, como la correcta asociación de usuarios con categorías y movimientos, para que la simulación del entorno de la base de datos sea lo más realista posible.

Además, el sistema también permitirá la generación de informes y la visualización de datos mediante sus herramientas de análisis. Mediante el uso de conjuntos de datos estructurados o incluso visualizaciones gráficas, el sistema permitirá el mapeo adecuado del comportamiento financiero del usuario, mostrando la evolución del gasto por categoría, ingresos versus gastos, la evolución temporal de los movimientos y la distribución porcentual del gasto. Estas visualizaciones tendrán como objetivo facilitar la comprensión de la información y apoyar la toma de decisiones.

Este proyecto se centrará en un entorno académico y demostrativo. Por lo tanto, no se considerarán mecanismos avanzados de seguridad, como autenticación compleja o cifrado de datos, integración con servicios financieros reales, etc. El sistema no prevé la conexión directa con cuentas bancarias, medios de pago ni la automatización de ninguna forma de transacción financiera.

Asimismo, el desarrollo de una aplicación móvil o web para un entorno productivo, ni la implementación de funcionalidades avanzadas relacionadas con IA, previsiones financieras automáticas o sugerencias, no forman parte del alcance de este proyecto. El sistema se basa principalmente en la gestión, el análisis y la visualización adecuados de la información financiera, priorizando la claridad, la estructura y la comprensión de los datos. El alcance del proyecto se resume en el desarrollo de un sistema funcional para gestionar presupuestos y permitir el análisis estructurado de datos financieros personales; también debe centrarse en una herramienta de aprendizaje académico, modelar correctamente la información y permitir la creación de informes claros y comprensibles; esto implica excluir temas típicos de sistemas comerciales o de producción a gran escala.

3. Arquitectura del sistema

3.1 Arquitectura general

El Sistema de Gestión de Presupuestos Personales se diseñó siguiendo el modelo de arquitectura en tres capas, una estructura ampliamente utilizada en el desarrollo de sistemas de información por su claridad, escalabilidad y separación de responsabilidades. Esta arquitectura permite dividir el sistema en componentes bien definidos, facilitando el mantenimiento, la comprensión del flujo de datos y la evolución futura del sistema.

Las capas que conforman esta arquitectura son: capa de presentación, capa de lógica de negocio y capa de datos, cada una con funciones específicas y responsabilidades claramente delimitadas.

3.2 Capa de presentación (Tkinter)

La capa de presentación es la encargada de la interacción directa con el usuario. En este proyecto, dicha capa se implementa utilizando Tkinter, una biblioteca gráfica incluida en Python que permite desarrollar interfaces de usuario de escritorio de manera sencilla y funcional.

Esta capa tiene como objetivo principal capturar las acciones del usuario y mostrar la información de forma clara y comprensible. Entre sus responsabilidades se incluyen:

- Mostrar formularios para el registro de usuarios, categorías, presupuestos e ingresos/gastos.
- Permitir la navegación entre las diferentes funcionalidades del sistema.
- Presentar mensajes informativos, de confirmación y de error al usuario.
- Mostrar resultados obtenidos del sistema, como listados de movimientos o resúmenes básicos.

La capa de presentación no contiene lógica de negocio ni acceso directo a la base de datos. Su función se limita exclusivamente a la visualización de la información y a la captura de eventos, los cuales son enviados a la capa de lógica para su procesamiento. Esta separación evita dependencias innecesarias y mejora la organización del código.

3.3 Capa de lógica de negocio (Python)

La capa de lógica de negocio constituye el núcleo funcional del sistema y se encuentra implementada en Python. Esta capa es responsable de procesar la información recibida desde

la interfaz gráfica, aplicar las reglas del negocio y coordinar la comunicación entre la capa de presentación y la capa de datos.

Dentro de esta capa se realizan tareas como:

- Validación de datos ingresados por el usuario (formatos, valores permitidos, campos obligatorios).
- Clasificación de movimientos financieros como ingresos o gastos.
- Gestión de categorías y asociación de estas con los movimientos.
- Cálculo de totales, balances y comparaciones entre valores planificados y reales.
- Preparación de la información que será enviada a la capa de datos para su almacenamiento o consulta.

Además, esta capa actúa como intermediaria entre la interfaz y la base de datos, evitando que la capa de presentación tenga conocimiento directo de las estructuras de almacenamiento. De esta manera, cualquier cambio en la base de datos o en las reglas del negocio puede ser gestionado desde la lógica sin afectar directamente a la interfaz gráfica.

3.4 Capa de datos (Oracle)

La capa de datos es la encargada del almacenamiento persistente de la información del sistema y se implementa utilizando una base de datos Oracle. Esta capa se responsabiliza de mantener la integridad, consistencia y disponibilidad de los datos financieros registrados.

Entre sus funciones principales se incluyen:

- Almacenamiento de información relacionada con usuarios, categorías, presupuestos y movimientos financieros.
- Gestión de relaciones entre las entidades del sistema.

- Ejecución de consultas, procedimientos almacenados y funciones para la obtención y manipulación de datos.

- Garantizar la integridad referencial y la coherencia de la información almacenada.

El acceso a la base de datos se realiza exclusivamente a través de la capa de lógica de negocio, lo que permite controlar las operaciones de inserción, actualización y consulta de datos. Esta separación reduce riesgos, mejora la seguridad y facilita la administración del sistema.

3.5 Flujo general de funcionamiento

El flujo de funcionamiento del sistema sigue una secuencia clara entre las capas:

1. El usuario interactúa con la interfaz gráfica en la capa de presentación.

2. Las acciones del usuario son enviadas a la capa de lógica de negocio.

3. La lógica valida y procesa la información, aplicando las reglas correspondientes.

4. En caso necesario, la lógica se comunica con la capa de datos para almacenar o recuperar información.

5. Los resultados obtenidos regresan a la capa de lógica y posteriormente a la capa de presentación para ser mostrados al usuario.

Este enfoque garantiza un sistema modular, organizado y fácil de mantener.

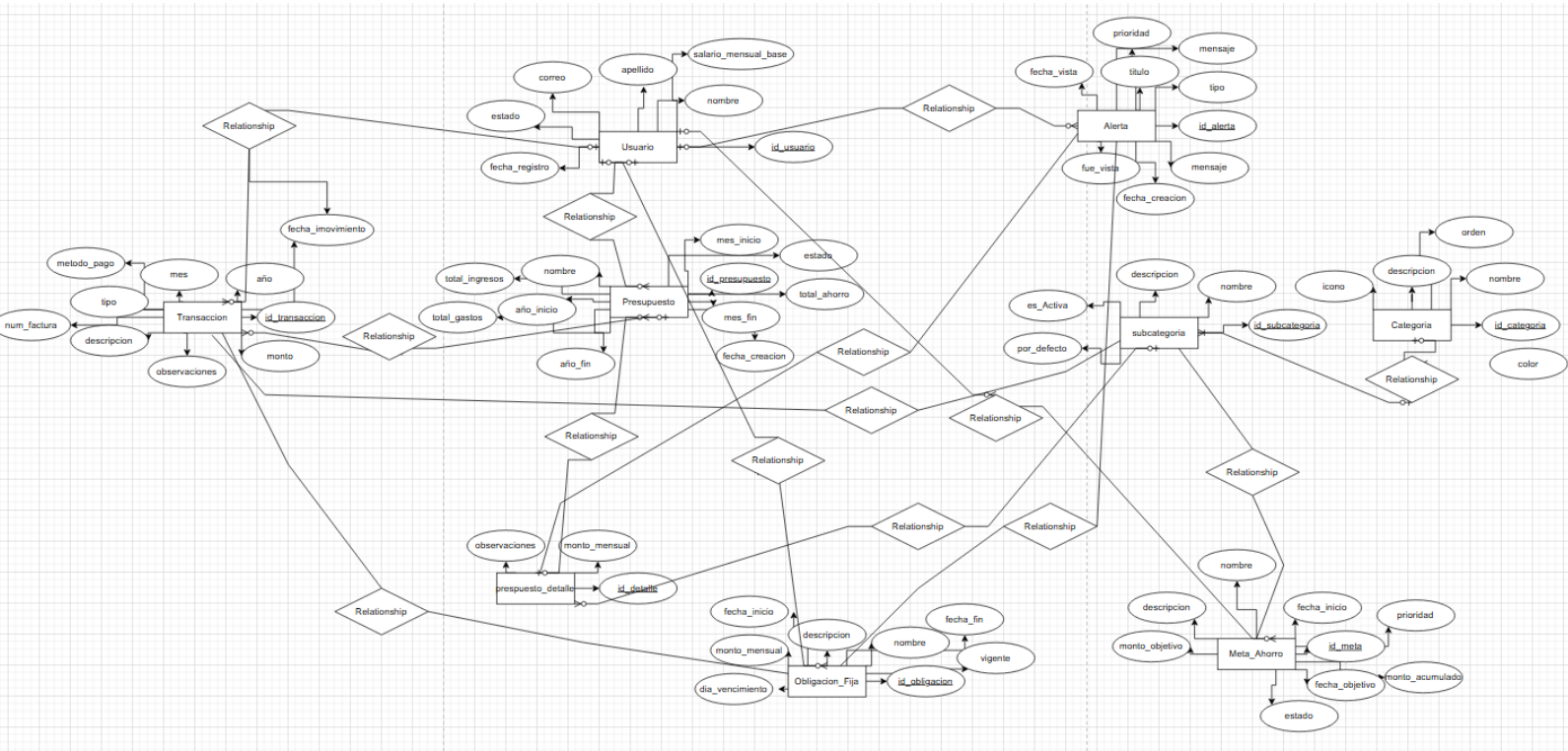
3.6 Ventajas de la arquitectura utilizada

La arquitectura en tres capas aporta múltiples beneficios al proyecto, entre los cuales destacan:

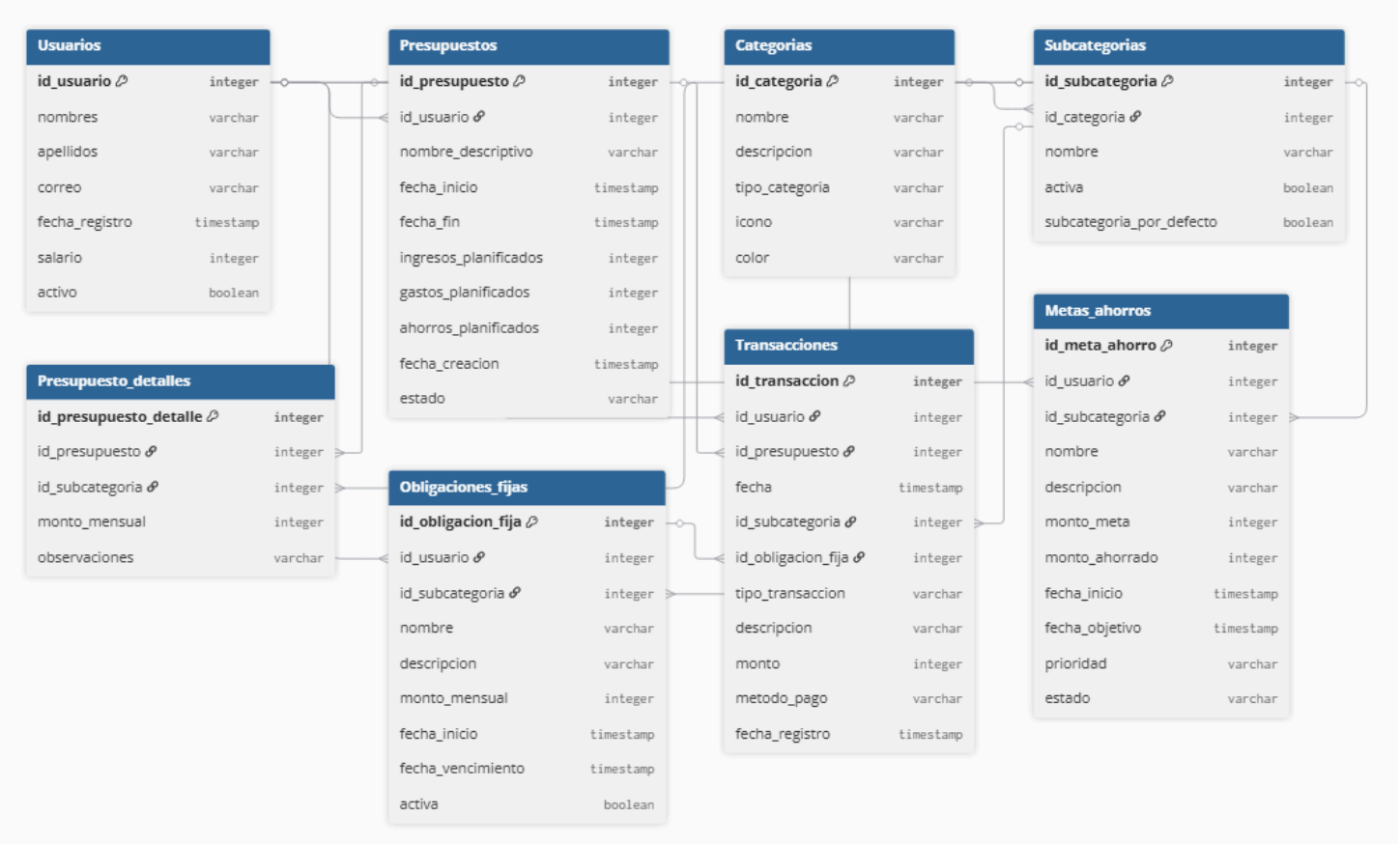
- Separación clara de responsabilidades entre interfaz, lógica y datos.
- Mayor facilidad de mantenimiento y escalabilidad.
- Posibilidad de reemplazar o mejorar una capa sin afectar significativamente a las demás.
- Mejora en la organización del código y comprensión del sistema.

4. Modelo de datos

4.1 Diagrama entidad-relación (ER)



4.2 Modelo relacional (RM)



4.3 Diccionario de datos

El presente diccionario de datos describe las entidades principales del Sistema de Gestión de Presupuestos Personales, detallando la estructura de cada tabla, sus atributos, tipos de datos y una breve descripción funcional. Este diccionario permite comprender de manera precisa cómo se almacena y organiza la información dentro del sistema, sirviendo como referencia técnica para el desarrollo, mantenimiento y análisis del mismo.

4.3.1 Tabla: USUARIOS

Descripción:
Almacena la información básica de los usuarios que utilizan el sistema. Cada usuario representa una entidad independiente con sus propios presupuestos y movimientos financieros.

Campo	Tipo de dato	Descripción
id_usuario	NUMBER	Identificador único del usuario.
nombres	VARCHAR2	Nombre(s) del usuario.
apellidos	VARCHAR2	Apellido(s) del usuario.
correo	VARCHAR2	Correo electrónico del usuario.
fecha_registro	DATE	Fecha en la que el usuario fue registrado en el sistema.
activo	NUMBER(1)	Indica si el usuario se encuentra activo (1) o inactivo (0).

4.3.2 Tabla: CATEGORIAS

Descripción:
Contiene categorías utilizadas para clasificar los movimientos financieros, permitiendo agrupar ingresos y gastos según su naturaleza.

Campo	Tipo de dato	Descripción
id_categoria	NUMBER	Identificador único de la categoría.
nombre	VARCHAR2	Nombre de la categoría (ej. Alimentación, Transporte).
descripcion	VARCHAR2	Descripción general de la categoría.
tipo_categoria	VARCHAR2	Indica si la categoría corresponde a un Ingreso o un Gasto.
icono	VARCHAR2	Identificador visual o icono asociado a la categoría.

4.3.3 Tabla: PRESUPUESTOS

Descripción:
Almacena los presupuestos planificados por cada usuario para un periodo determinado. Permite comparar los valores planificados con los gastos reales registrados.

Campo	Tipo de dato	Descripción
id_presupuesto	NUMBER	Identificador único del presupuesto.
id_usuario	NUMBER	Identificador del usuario asociado al presupuesto.
nombre_descriptivo	VARCHAR2	Nombre o descripción del presupuesto.

fecha_inicio	DATE	Fecha de inicio del periodo del presupuesto.
fecha_fin	DATE	Fecha de finalización del periodo del presupuesto.
ingresos_planificados	NUMBER	Monto total de ingresos esperados.
gastos_planificados	NUMBER	Monto total de gastos planificados.
ahorros_planificados	NUMBER	Monto destinado al ahorro.
fecha_creacion	DATE	Fecha de creación del presupuesto.
estado	VARCHAR2	Estado del presupuesto (Activo, Cerrado, etc.).

4.3.4 Tabla: SUBCATEGORIAS

Descripción:

Permite un mayor nivel de detalle en la clasificación de las transacciones, asociando subcategorías a una categoría principal.

Campo	Tipo de dato	Descripción
id_subcategoria	NUMBER	Identificador único de la subcategoría.
id_categoria	NUMBER	Categoría a la que pertenece la subcategoría.

nombre	VARCHAR2	Nombre de la subcategoría.
descripcion	VARCHAR2	Descripción de la subcategoría.

4.3.5 Tabla: PRESUPUESTO_DETALLES

Descripción:

Desglosa el presupuesto total en montos asignados por categoría o subcategoría, permitiendo un control más específico del gasto planificado.

Campo	Tipo de dato	Descripción
id_detalle	NUMBER	Identificador único del detalle del presupuesto.
id_presupuesto	NUMBER	Presupuesto al que pertenece el detalle.
id_categoria	NUMBER	Categoría asociada al monto planificado.
id_subcategoria	NUMBER	Subcategoría asociada (opcional).
monto_asignado	NUMBER	Monto planificado para la categoría o subcategoría.

4.3.6 Tabla: TRANSACCIONES

Descripción:

Registra todos los movimientos financieros realizados por los usuarios, incluyendo ingresos, gastos, pagos de obligaciones y aportes a metas de ahorro.

Campo	Tipo de dato	Descripción
id_transaccion	NUMBER	Identificador único de la transacción.
id_usuario	NUMBER	Usuario que realizó la transacción.
id_categoria	NUMBER	Categoría asociada a la transacción.
id_subcategoria	NUMBER	Subcategoría asociada (opcional).
fecha	DATE	Fecha de la transacción.
monto	NUMBER	Monto del ingreso o gasto.
tipo	VARCHAR2	Tipo de transacción (Ingreso o Gasto).
descripcion	VARCHAR2	Descripción opcional de la transacción.

4.3.7 Tabla: METAS_AHORROS

Descripción:

Permite definir metas de ahorro con un monto objetivo y dar seguimiento al progreso de ahorro del usuario.

Campo	Tipo de dato	Descripción

id_meta	NUMBER	Identificador único de la meta de ahorro.
id_usuario	NUMBER	Usuario propietario de la meta.
nombre	VARCHAR2	Nombre descriptivo de la meta.
monto_objetivo	NUMBER	Monto total que se desea ahorrar.
monto_actual	NUMBER	Monto acumulado hasta el momento.
fecha_inicio	DATE	Fecha de inicio de la meta.
fecha_fin	DATE	Fecha estimada de cumplimiento.
estado	VARCHAR2	Estado de la meta (Activa, Cumplida).

4.3.8 Tabla: OBLIGACIONES_FIJAS

Descripción:

Registra los gastos recurrentes u obligaciones financieras periódicas del usuario, como servicios, alquileres o pagos fijos.

Campo	Tipo de dato	Descripción
id_obligacion	NUMBER	Identificador único de la obligación.

id_usuario	NUMBER	Usuario al que pertenece la obligación.
nombre	VARCHAR2	Nombre de la obligación.
monto	NUMBER	Monto fijo de la obligación.
periodicidad	VARCHAR2	Frecuencia de pago (Mensual, Quincenal).
fecha_inicio	DATE	Fecha de inicio de la obligación.
estado	VARCHAR2	Estado de la obligación (Activa, Cancelada).

4.3.9 Observaciones finales

El diccionario de datos fue diseñado para reflejar un sistema financiero personal realista, permitiendo un alto nivel de detalle en la planificación, ejecución y análisis de las finanzas del usuario. La estructura facilita la generación de reportes y visualizaciones, así como la extensión futura del sistema sin comprometer su integridad.

5. Reglas de negocio implementadas

Las reglas de negocio del Sistema de Gestión de Presupuestos Personales definen el comportamiento, las restricciones y las validaciones que rigen el funcionamiento del sistema. Estas reglas garantizan la coherencia de la información financiera, el correcto control presupuestal y la correcta relación entre las distintas entidades del modelo de datos.

Las reglas descritas a continuación corresponden únicamente a las funcionalidades implementadas, excluyendo de manera explícita el módulo de alertas automáticas, el cual fue descartado según indicaciones del docente

5.1 Reglas de negocio relacionadas con usuarios

- Todo usuario debe contar con un identificador único dentro del sistema.

- El correo electrónico del usuario debe ser único y no puede repetirse entre distintos registros.

- Un usuario puede tener múltiples presupuestos a lo largo del tiempo, pero estos deben estar claramente delimitados por su periodo de vigencia.

- Un usuario puede registrar múltiples transacciones, metas de ahorro y obligaciones fijas asociadas a su perfil.

- Los usuarios inactivos no pueden registrar nuevas transacciones ni crear nuevos presupuestos.

5.2 Reglas de negocio relacionadas con categorías y subcategorías

- Toda categoría debe pertenecer a un único tipo: ingreso, gasto o ahorro.

- Toda categoría debe contar obligatoriamente con al menos una subcategoría asociada.

- Al momento de crear una categoría, el sistema crea automáticamente una subcategoría por defecto (por ejemplo, “General”), garantizando que la categoría siempre sea utilizable

- Las subcategorías heredan el tipo de su categoría padre y no pueden cambiarlo de manera independiente.

- No se permite registrar transacciones ni asignaciones presupuestarias directamente a una categoría; estas deben realizarse exclusivamente a nivel de subcategoría.

- Una subcategoría solo puede pertenecer a una única categoría.

- Toda subcategoría utilizada en transacciones o presupuestos debe encontrarse activa.

5.3 Reglas de negocio relacionadas con presupuestos

- Todo presupuesto debe estar asociado a un usuario específico.

- Cada presupuesto debe contar con un periodo de vigencia definido por año y mes de inicio y fin.

- El periodo de fin del presupuesto debe ser igual o posterior al periodo de inicio.

- Un usuario no puede tener más de un presupuesto activo que cubra el mismo periodo de tiempo.

- El presupuesto define los montos globales planificados para ingresos, gastos y ahorro, los cuales sirven como referencia para el análisis financiero.

- Un presupuesto puede encontrarse en diferentes estados, tales como activo o cerrado.

- No se permite registrar transacciones fuera del periodo de vigencia del presupuesto asociado.

5.4 Reglas de negocio relacionadas con los detalles del presupuesto

- Todo detalle de presupuesto debe estar asociado a una subcategoría específica.

- El monto asignado en el detalle de presupuesto corresponde a un monto mensual fijo, el cual se aplica a todos los meses dentro de la vigencia del presupuesto.

- No se permite que un detalle de presupuesto tenga valores nulos en la subcategoría o en el monto asignado.

- Si se desea asignar montos distintos para meses diferentes, se debe crear un nuevo presupuesto para ese periodo.

- El monto ejecutado de una subcategoría se calcula dinámicamente sumando las transacciones registradas para el mes correspondiente.

- El porcentaje de ejecución presupuestal se obtiene comparando el monto ejecutado frente al monto mensual asignado.

5.5 Reglas de negocio relacionadas con transacciones

- Toda transacción debe estar asociada a un usuario, un presupuesto y una subcategoría.

- El tipo de la transacción (ingreso, gasto o ahorro) debe coincidir con el tipo de la categoría padre de la subcategoría asociada.

- Toda transacción debe contar con un monto mayor a cero.

- Las transacciones incluyen campos de año y mes presupuestal, los cuales determinan a qué periodo se imputa el movimiento financiero.

- El año y mes presupuestal pueden ser distintos a la fecha real de la transacción, permitiendo flexibilidad contable, siempre que se encuentren dentro de la vigencia del presupuesto

- Toda transacción asociada a una obligación fija debe utilizar la misma subcategoría definida en dicha obligación.

- Las transacciones constituyen la fuente principal para el cálculo de gastos ejecutados, ingresos reales y avances en metas de ahorro.

5.6 Reglas de negocio relacionadas con obligaciones fijas

- Toda obligación fija debe estar asociada a una subcategoría de tipo gasto.
- Las obligaciones fijas representan compromisos financieros recurrentes con monto y periodicidad definidos.
- Una obligación puede tener una fecha de finalización o ser indefinida.
- La fecha de finalización, si existe, debe ser posterior a la fecha de inicio.
- Las obligaciones fijas pueden utilizarse como referencia para el registro de transacciones periódicas.
- Una obligación inactiva no puede ser utilizada para asociar nuevas transacciones.

5.7 Reglas de negocio relacionadas con metas de ahorro

- Toda meta de ahorro debe estar asociada a una subcategoría de tipo ahorro.
- Una subcategoría de ahorro solo puede estar asociada a una meta activa a la vez.
- El monto acumulado de una meta no puede superar el monto objetivo definido.
- El progreso de la meta se calcula como el porcentaje entre el monto acumulado y el monto objetivo.
- El monto acumulado se actualiza automáticamente al registrarse transacciones de ahorro asociadas a la subcategoría de la meta, garantizando consistencia en tiempo real

  .

- Una meta cambia automáticamente su estado a completado cuando alcanza o supera el 100% del monto objetivo.

- La fecha objetivo de una meta debe ser posterior a la fecha de inicio.

5.8 Exclusión del módulo de alertas

Aunque el diseño original del sistema contempla la generación de alertas automáticas relacionadas con ejecución presupuestal, vencimiento de obligaciones y avance de metas de ahorro, dicho módulo fue excluido de la implementación final siguiendo instrucciones directas del docente.
No obstante, el diseño del modelo de datos y de las reglas de negocio deja abierta la posibilidad de incorporar este módulo en futuras extensiones del sistema, sin necesidad de modificar la estructura base de la base de datos.

5.9 Consideraciones finales sobre las reglas de negocio

Las reglas de negocio implementadas permiten representar de manera fiel el comportamiento financiero personal, asegurando coherencia entre presupuestos, transacciones, obligaciones y metas de ahorro. Estas reglas constituyen el núcleo lógico del sistema y garantizan que la información almacenada pueda ser analizada de forma confiable mediante reportes y visualizaciones, cumpliendo los objetivos académicos del proyecto

6. Auditoría y seguridad de datos

La auditoría y la seguridad de los datos son aspectos fundamentales en cualquier sistema de información que gestione datos sensibles. En el Sistema de Gestión de Presupuestos Personales, estos aspectos se abordan mediante la implementación de campos de auditoría obligatorios en todas las tablas y mediante un control estructurado del acceso y la modificación de la información, garantizando la trazabilidad, integridad y confiabilidad de los datos almacenados.

6.1 Campos de auditoría

Con el objetivo de mantener un registro detallado de las operaciones realizadas sobre la base de datos, todas las tablas del sistema incluyen campos de auditoría obligatorios, los cuales permiten identificar quién creó o modificó un registro, así como el momento exacto en que dichas acciones ocurrieron.

Los campos de auditoría implementados son los siguientes:

Campo	Descripción
creado_por	Identifica el usuario o proceso que creó el registro.
creado_en	Fecha y hora exacta en la que se realizó la inserción del registro.
modificado_por	Identifica el usuario o proceso que realizó la última modificación del registro.
modificado_en	Fecha y hora exacta de la última modificación del registro.

Estos campos se encuentran presentes en todas las entidades principales del sistema, incluyendo usuarios, categorías, subcategorías, presupuestos, detalles de presupuesto, transacciones, metas de ahorro y obligaciones fijas.

La actualización de estos campos se realiza de manera automática mediante valores por defecto y lógica implementada en la base de datos, evitando que dependan de la intervención manual del usuario. Esto asegura que la información de auditoría sea consistente y confiable.

6.2 Justificación de los campos de auditoría

La inclusión de campos de auditoría se justifica principalmente por la necesidad de trazabilidad y control de cambios dentro del sistema. Dado que el sistema gestiona información financiera personal, resulta fundamental poder identificar el origen de cada registro y las modificaciones que ha sufrido a lo largo del tiempo.

Desde el punto de vista técnico, los campos de auditoría permiten:

- Rastrear errores o inconsistencias en los datos, identificando cuándo y por quién se realizó una modificación.

- Facilitar el mantenimiento y la depuración del sistema, especialmente durante pruebas y validaciones.

- Proporcionar soporte para análisis históricos y revisión de cambios en la información financiera.

- Simular buenas prácticas utilizadas en sistemas reales de gestión financiera y contable.

Desde el punto de vista académico, la implementación de auditoría demuestra la aplicación de buenas prácticas de diseño de bases de datos, alineadas con sistemas empresariales reales, donde la trazabilidad de la información es un requisito indispensable.

6.3 Seguridad de los datos

La seguridad de los datos en el sistema se gestiona principalmente a través de la **separación de responsabilidades** y el control del acceso a la base de datos. El diseño del sistema sigue el principio de que la mayor parte de la lógica de negocio reside en la base de datos, lo que reduce el riesgo de manipulaciones incorrectas desde la capa de presentación.

Las principales medidas de seguridad implementadas incluyen:

- Acceso controlado a la base de datos mediante credenciales específicas.

- Uso de procedimientos almacenados para realizar operaciones CRUD, evitando el acceso directo a las tablas.

- Validación de reglas de negocio en la base de datos, garantizando la integridad de la información.

- Restricciones de claves primarias y foráneas para mantener la coherencia entre las entidades.

- Control del estado de los registros (activos/inactivos) en lugar de eliminaciones físicas cuando corresponde.

Estas medidas permiten minimizar errores, prevenir inconsistencias y asegurar que los datos financieros sean manipulados únicamente bajo las reglas definidas por el sistema.

6.4 Consideraciones finales

La implementación de auditoría y seguridad de datos en el Sistema de Gestión de Presupuestos Personales refuerza la confiabilidad del sistema y su alineación con prácticas profesionales de desarrollo de sistemas de información. Aunque el proyecto se desarrolla en un entorno académico, las decisiones adoptadas reflejan escenarios reales de sistemas financieros, sentando una base sólida para futuras extensiones o adaptaciones a entornos productivos

7. Procesos almacenados

7.1 Tabla resumen de procedimientos almacenados

La siguiente tabla presenta un resumen de los procedimientos almacenados implementados en la base de datos del sistema, indicando la tabla principal que afectan y el tipo de operación que realizan. Esta clasificación permite identificar de forma clara la cobertura funcional del sistema a nivel de base de datos.

Procedimiento almacenado	Tabla(s) asociada(s)	Tipo
SP_INSERTAR_USUARIO	USUARIOS	Inserción
SP_ACTUALIZAR_USUARIO	USUARIOS	Actualización
SP_DESACTIVAR_USUARIO	USUARIOS	Lógica (desactivación)
SP_CONSULTAR_USUARIO	USUARIOS	Consulta
SP_LISTAR_USUARIOS	USUARIOS	Consulta
SP_VALIDAR_USUARIO	USUARIOS	Validación
SP_INSERTAR_CATEGORIA	CATEGORIAS	Inserción

SP_ACTUALIZAR_CATEGORIA	CATEGORIAS	Actualización
SP_ELIMINAR_CATEGORIA	CATEGORIAS	Lógica (desactivación)
SP_CONSULTAR_CATEGORIA	CATEGORIAS	Consulta
SP_LISTAR_CATEGORIAS	CATEGORIAS	Consulta
SP_INSERTAR_SUBCATEGORIA	SUBCATEGORIAS	Inserción
SP_ACTUALIZAR_SUBCATEGORIA	SUBCATEGORIAS	Actualización
SP_ELIMINAR_SUBCATEGORIA	SUBCATEGORIAS	Lógica (desactivación)
SP_CONSULTAR_SUBCATEGORIA	SUBCATEGORIAS	Consulta
SP_LISTAR_SUBCATEGORIAS_POR_CATEGORIA	SUBCATEGORIAS	Consulta
SP_INSERTAR_PRESUPUESTO	PRESUPUESTOS	Inserción
SP_CREAR_PRESUPUESTO_COMPLETO	PRESUPUESTOS	Orquestación
SP_ACTUALIZAR_PRESUPUESTO	PRESUPUESTOS	Actualización

SP_CERRAR_PRESUPUESTO	PRESUPUESTOS	Lógica (cierre)
SP_CONSULTAR_PRESUPUESTO	PRESUPUESTOS	Consulta
SP_LISTAR_PRESUPUESTOS_USUARIO	PRESUPUESTOS	Consulta
SP_RESUMEN_PRESUPUESTO	PRESUPUESTOS	Resumen
SP_INSERTAR_PRESUPUESTO_DETALLE	PRESUPUESTO_DETALLES	Inserción
SP_ACTUALIZAR_PRESUPUESTO_DETALLE	PRESUPUESTO_DETALLES	Actualización
SP_ELIMINAR_PRESUPUESTO_DETALLE	PRESUPUESTO_DETALLES	Eliminación
SP_CONSULTAR_PRESUPUESTO_DETALLE	PRESUPUESTO_DETALLES	Consulta
SP_LISTAR_DETALLES_PRESUPUESTO	PRESUPUESTO_DETALLES	Consulta
SP_INSERTAR_TRANSACCION	TRANSACCIONES	Inserción
SP_ACTUALIZAR_TRANSACCION	TRANSACCIONES	Actualización

SP_ELIMINAR_TRANSACCION	TRANSACCIONES	Eliminación
SP_CONSULTAR_TRANSACCION	TRANSACCIONES	Consulta
SP_LISTAR_TRANSACCIONES_PRESUPUESTO	TRANSACCIONES	Consulta
SP_INSERTAR_META_AHORRO	METAS_AHORROS	Inserción
SP_ACTUALIZAR_META_AHORRO	METAS_AHORROS	Actualización
SP_CANCELAR_META_AHORRO	METAS_AHORROS	Lógica (cancelación)
SP_CONSULTAR_META_AHORRO	METAS_AHORROS	Consulta
SP_LISTAR_METAS_USUARIO	METAS_AHORROS	Consulta
SP_ACTUALIZAR_METAS_POR_AHORRO	METAS_AHORROS	Lógica (cálculo)
SP_INSERTAR_OBLIGACION_FIJA	OBLIGACIONES_FIJAS	Inserción
SP_ACTUALIZAR_OBLIGACION_FIJA	OBLIGACIONES_FIJAS	Actualización
SP_DESACTIVAR_OBLIGACION_FIJA	OBLIGACIONES_FIJAS	Lógica (desactivación)

SP_CONSULTAR_OBLIGACION_FIJA	OBLIGACIONES_FIJAS	Consulta
SP_LISTAR_OBLIGACIONES_USUARIO	OBLIGACIONES_FIJAS	Consulta
SP_RESUMEN_CATEGORIA_PRESUPUESTO	CATEGORIAS / TRANSACCIONES	Resumen

7.2 Observaciones generales

- El sistema implementa procedimientos para todas las operaciones críticas, evitando el acceso directo a las tablas.

- No se realizan eliminaciones físicas en la mayoría de entidades; se privilegia la desactivación lógica, alineada con las reglas de negocio.

- Existen procedimientos de orquestación (como SP_CREAR_PRESUPUESTO_COMPLETO) que encapsulan múltiples operaciones, reduciendo errores desde la capa de aplicación.

- Los procedimientos de resumen concentran la lógica de cálculo financiero directamente en la base de datos, garantizando consistencia en los reportes.

7.3 Descripción detallada de los procedimientos

USUARIOS

1. Insertar usuario

Nombre: sp_insertar_usuario

Tipo: Procedimiento (CRUD)

Descripción: Registra un nuevo usuario al sistema

Parámetros de entrada:

Parámetro	Tipo	Descripción
p_nombres	**VARCHAR2**	**Nombres del usuario**
p_apellidos	**VARCHAR2**	**Apellidos del usuario**
p_correo	**VARCHAR2**	**Correo electrónico**
p_salario	**NUMBER**	**Salario mensual**
p_usuario_creacion	**VARCHAR2**	**Usuario que crea el registro**

Uso en SQL:

```
BEGIN
  sp_insertar_usuario(
    'Juan',
    'Pérez',
    'juan@mail.com',
    20000,
    'admin'
  );
END;
/
```

Uso en Python (front):

```
cursor.callproc(
    "sp_insertar_usuario",
    ["Juan", "Pérez", "juan@mail.com", 20000, "admin"]
)
connection.commit()
```

2. Actualizar usuario

Nombre: sp_actualizar_usuario

Tipo: Procedimiento (CRUD)

Descripción: un nuevo usuario al sistema

Parámetros de entrada:

Parámetro	Tipo	Descripción
p_id_usuario	NUMBER	ID del usuario
p_nombres	VARCHAR2	Nombres
p_apellidos	VARCHAR2	Apellidos
p_correo	VARCHAR2	Correo
p_salario	NUMBER	Salario
p_activo	CHAR	'Y' / 'N'
p_usuario_modificacion	VARCHAR2	Auditoría

Uso en SQL:

```
BEGIN
  sp_actualizar_usuario(
    1,
    'Juan',
    'Pérez',
    'juan@mail.com',
    22000,
    'Y',
    'admin'
  );
END;
```

Uso en Python (front):

```
cursor.callproc(
    "sp_actualizar_usuario",
    [1, "Juan", "Pérez", "juan@mail.com", 22000, "Y", "admin"]
)
connection.commit()
```

3. Listar usuarios

Nombre: sp_listar_usuarios

Tipo: Procedimiento (Consulta)

Descripción: Devuelve una lista de usuarios registrados

Parámetros de salida:

Parámetro	Tipo
p_resultado	SYS_REFCURSOR

Uso en Python (front):

```
out_cursor = cursor.var(cx_Oracle.CURSOR)
cursor.callproc("sp_listar_usuarios", [out_cursor])

for row in out_cursor.getvalue():
    print(row)
```

CATEGORIAS Y SUBCATEGORIAS

4. Insertar categoría

Nombre: sp_insertar_categoria

Tipo: Procedimiento (CRUD)

Descripción: Crea una categoría financiera. El sistema genera automática una subcategoria “general”

Parámetros de entrada:

Parámetro	Tipo
p_nombre	**VARCHAR2**
p_descripcion	**VARCHAR2**
p_tipo_categoria	**VARCHAR2 (INGRESO / GASTO / AHORRO)**
p_icono	**VARCHAR2**
p_color	**VARCHAR2**
p_usuario_creacion	**VARCHAR2**

Uso en SQL:

```
BEGIN
  sp_insertar_categoria(
    'Ahorros',
    'Categoría de ahorro',
    'AHORRO',
    'piggy-bank',
    '#2ECC71',
    'admin'
  );
END;
/
```

Uso en Python (front):

```
cursor.callproc(
    "sp_insertar_categoria",
    ["Ahorros", "Categoría de ahorro", "AHORRO", "piggy-bank", "#2ECC71", "admin"]
)
connection.commit()
```

5. Listar categorias

Nombre: sp_listar_categorias

Tipo: Procedimiento (Consulta)

Descripción: Devuelve todas las categorías del sistema.

Uso en Python (front):

```
out_cursor = cursor.var(cx_Oracle.CURSOR)
cursor.callproc("sp_listar_categorias", [out_cursor])

for row in out_cursor.getvalue():
    print(row)
```

6. Listar categorias de una categoría

Nombre: sp_listar_subcategorias_categoria

Tipo: Procedimiento (Consulta)

Descripción: Devuelve las subcategorías de una categoría.

Parámetros de entrada:

Parámetro	Tipo
p_id_categoria	NUMBER
p_resultado	SYS_REFCURSOR (OUT)

Uso en Python (front):

```
out_cursor = cursor.var(cx_Oracle.CURSOR)
cursor.callproc(
    "sp_listar_subcategorias_categoria",
    [1, out_cursor]
)
for row in out_cursor.getvalue():
    print(row)
```

PRESUPUESTOS

7. Insertar presupuesto

Nombre: sp_insertar_presupuesto

Tipo: Procedimiento (Negocio)

Descripción: Crea un presupuesto activo. Solo se permite uno activo por usuario.

Parámetros de entrada:

Parámetro	Tipo
p_id_usuario	NUMBER
p_nombre	VARCHAR2
p_fecha_inicio	DATE
p_fecha_fin	DATE
p_ingresos_planificados	NUMBER
p_gastos_planificados	NUMBER
p_ahorros_planificados	NUMBER
p_usuario_creacion	VARCHAR2

Uso en Python (front):

```
cursor.callproc(
    "sp_insertar_presupuesto",
    [1, "Presupuesto Enero", fecha_ini, fecha_fin, 20000, 15000, 3000, "admin"]
)
connection.commit()
```

8. Listar presupuestos

Nombre: sp_listar_presupuestos_usuario

Tipo: Procedimiento (Consulta)

Descripción: Lista los presupuestos de un usuario, opcionalmente filtrados por el estado.

Parámetros de entrada:

Parámetro	Tipo
p_id_usuario	NUMBER
p_estado	VARCHAR2 (opcional)
p_resultado	SYS_REFCURSOR

Uso en Python (front):

```
out_cursor = cursor.var(cx_Oracle.CURSOR)
cursor.callproc(
    "sp_listar_presupuestos_usuario",
    [1, None, out_cursor]
)
```

9. Consultar presupuesto

Nombre: sp_consultar_presupuesto

Tipo: Procedimiento (Consulta)

Descripción: Obtiene el detalle completo de un presupuesto.

Uso en Python (front):

```
out_cursor = cursor.var(cx_Oracle.CURSOR)
cursor.callproc(
    "sp_consultar_presupuesto",
    [1, out_cursor]
)
```

10. Cerrar presupuesto

Nombre: sp_cerrar_presupuesto

Tipo: Procedimiento (CRUD)

Descripción: Cierra un presupuesto activo.

Parámetros de entrada:

Parámetro	Tipo
p_id_presupuesto	**NUMBER**
p_usuario_modificacion	**VARCHAR2**

DETALLES DE PRESUPUESTO

11. Insertar presupuesto detalle

Nombre: sp_insertar_presupuesto_detalle

Tipo: Procedimiento (CRUD)

Descripción: Asigna un monto mensual a una subcategoría dentro del presupuesto.

Parámetros de entrada:

Parámetro	Tipo
p_id_presupuesto	**NUMBER**
p_id_subcategoria	**NUMBER**
p_monto_mensual	**NUMBER**
p_observaciones	**VARCHAR2**

12. Actualizar presupuesto detalle

Nombre: sp_actualizar_presupuesto_detalle

Tipo: Procedimiento (CRUD)

Descripción: Actualiza el monto asignado a una subcategoria del presupuesto.

Parámetros de entrada:

Parámetro	Tipo
p_id_presupuesto_detalle	NUMBER
p_monto_mensual	NUMBER
p_observaciones	VARCHAR2

13. Listar detalles de presupuesto

Nombre: sp_listar_detalles_presupuesto

Tipo: Procedimiento (Consulta)

Descripción: Lista los detalles de un presupuesto

Uso en Python:

```
out_cursor = cursor.var(cx_Oracle.CURSOR)
cursor.callproc(
    "sp_listar_detalles_presupuesto",
    [1, out_cursor]
)
```

14. Consultar presupuesto detalle

Nombre: sp_consultar_presupuesto_detalle

Tipo: Procedimiento (Consulta)

Descripción: Consulta un detalle específico del presupuesto.

OBLIGACIONES FIJAS

15. Insertar obligación fija

Nombre: sp_insertar_ogligacion_fija

Tipo: Procedimiento (Negocio)

Descripción: Registra una obligación mensual fija (ej. alquiler)

Parámetros de entrada:

Parámetro	Tipo
p_id_usuario	NUMBER
p_id_subcategoria	NUMBER (GASTO)
p_nombre	VARCHAR2
p_descripcion	VARCHAR2
p_monto_mensual	NUMBER
p_fecha_inicio	DATE
p_fecha_vencimiento	DATE
p_usuario_creacion	VARCHAR2

16. Actualizar obligación fija

Nombre: sp_actualizar_obligacion_fija

Tipo: Procedimiento (CRUD)

Descripción: Actualiza los datos de una obligación fija existente.

Uso en Python:

```python
cursor.callproc(
    "sp_actualizar_obligacion_fija",
    [
        1,                      # id_obligacion_fija
        "Alquiler casa",
        "Pago mensual actualizado",
        5200,
        None,
        "Y",
        "admin"
    ]
)
connection.commit()
```

17. Desactivar obligación fija

Nombre: sp_desactivar_obligacion_fija

Tipo: Procedimiento (Negocio)

Descripción: Desactiva (borrado lógico) una obligación fija.

Uso en Python:

```python
cursor.callproc(
    "sp_desactivar_obligacion_fija",
    [1, "admin"]
)
connection.commit()
```

18. Listar obligaciones de usuario

Nombre: sp_listar_obligaciones_usuario

Tipo: Procedimiento (Consulta)

Descripción: Lista las obligaciones fijas de un usuario.

Uso en Python:

```
out_cursor = cursor.var(cx_Oracle.CURSOR)

cursor.callproc(
    "sp_listar_obligaciones_usuario",
    [1, None, out_cursor]
)

for row in out_cursor.getvalue():
    print(row)
```

19. Consultar obligación fija

Nombre: sp_consultar_obligacion_fija

Tipo: Procedimiento (Consulta)

Descripción: Consulta el detalle de una obligación fija específica.

Uso en Python:

```
out_cursor = cursor.var(cx_Oracle.CURSOR)

cursor.callproc(
    "sp_consultar_obligacion_fija",
    [1, out_cursor]
)

for row in out_cursor.getvalue():
    print(row)
```

METAS DE AHORRO

20. Insertar meta ahorro

Nombre: sp_insertar_meta_ahorro

Tipo: Procedimiento (Negocio)

Descripción: Crea una meta de ahorro asociada a una subcategoría AHORRO

Parámetros de entrada:

Parámetro	Tipo
p_id_usuario	**NUMBER**
p_id_subcategoria	**NUMBER**
p_nombre	**VARCHAR2**
p_descripcion	**VARCHAR2**
p_monto_meta	**NUMBER**
p_fecha_inicio	**DATE**
p_fecha_objetivo	**DATE**
p_prioridad	**VARCHAR2**
p_usuario_creacion	**VARCHAR2**

Uso en Python:

```
cursor.callproc(
    "sp_insertar_meta_ahorro",
    [
        1,
        8,                      # subcategoría AHORRO
        "Fondo emergencia",
        "Para imprevistos",
        10000,
        fecha_inicio,
        fecha_objetivo,
        "ALTA",
        "admin"
    ]
)
connection.commit()
```

21. Cancelar meta ahorro

Nombre: sp_cancelar_meta_ahorro

Tipo: Procedimiento (Negocio)

Descripción: Cancela una meta de ahorro activa.

Uso en Python:

```
cursor.callproc(
    "sp_cancelar_meta_ahorro",
    [1, "admin"]
)
connection.commit()
```

22. Listar metas ahorro

Nombre: sp_listar_metas_usuario

Tipo: Procedimiento (Negocio)

Descripción: Muestra las metas de ahorro del usuario.

Uso en Python:

```
out_cursor = cursor.var(cx_Oracle.CURSOR)

cursor.callproc(
    "sp_listar_metas_usuario",
    [1, None, out_cursor]
)

for row in out_cursor.getvalue():
    print(row)
```

23. Consultar meta ahorro

Nombre: sp_consultar_meta_ahorro

Tipo: Procedimiento (Negocio)

Descripción: Consulta los datos de una meta de ahorro.

Uso en Python:

```python
out_cursor = cursor.var(cx_Oracle.CURSOR)

cursor.callproc(
    "sp_consultar_meta_ahorro",
    [1, out_cursor]
)

for row in out_cursor.getvalue():
    print(row)
```

TRANSACCIONES

24. Registrar transacción completa

Nombre: sp_registrar_transaccion_completa

Tipo: Procedimiento (Negocio - principal)

Descripción: Registra una transacción aplicando todas las reglas del sistema.

Parámetros de entrada:

Parámetro	Tipo
p_id_usuario	**NUMBER**
p_id_presupuesto	**NUMBER**
p_fecha	**DATE**
p_id_subcategoria	**NUMBER**
p_tipo_transaccion	**VARCHAR2**
p_monto	**NUMBER**
p_descripcion	**VARCHAR2**
p_metodo_pago	**VARCHAR2**
p_usuario_creacion	**VARCHAR2**

Uso en Python (front):

```
cursor.callproc(
    "sp_registrar_transaccion_completa",
    [1, 1, fecha, 8, "AHORRO", 500, "Ahorro mensual", "EFECTIVO", "admin"]
)
connection.commit()
```

25. Listar transacciones de un presupuesto

Nombre: sp_listar_transacciones_presupuesto

Tipo: Procedimiento (Consulta)

Descripción: Lista las transacciones de un presupuesto.

Uso en Python (front):

```
out_cursor = cursor.var(cx_Oracle.CURSOR)

cursor.callproc(
    "sp_listar_transacciones_presupuesto",
    [1, out_cursor]
)

for row in out_cursor.getvalue():
    print(row)
```

26. Consultar transacción

Nombre: sp_consultar_transaccion

Tipo: Procedimiento (Consulta)

Descripción: Muestra los detalles de una transacción.

Uso en Python (front):

```
out_cursor = cursor.var(cx_Oracle.CURSOR)

cursor.callproc(
    "sp_consultar_transaccion",
    [1, out_cursor]
)

for row in out_cursor.getvalue():
    print(row)
```

27. Eliminar transacción

Nombre: sp_eliminar_transaccion

Uso en Python (front):

```
cursor.callproc(
   "sp_eliminar_transaccion",
   [1]
)
connection.commit()
```

FUNCIONES DE CONSULTA (REPORTES)

28. Obtener balance del presupuesto

Nombre: fn_balance_presupuesto

Descripción: Devuelve el balance del presupuesto

Uso en Python (front):

```
balance = cursor.callfunc(
    "fn_balance_presupuesto",
    float,
    [1]
)
```

29. Obtener avance de meta ahorro

Nombre: fn_avance_meta_ahorro

Tipo: Función

Descripción: Devuelve el porcentaje de avance de una meta de ahorro.

Uso en Python (front):

```
avance = cursor.callfunc(
    "fn_avance_meta_ahorro",
    float,
    [1]
)
```

30. Total ingresos de un presupuesto

Nombre: fn_total_ingresos_presupuesto

Uso en Python (front):

```
total_ingresos = cursor.callfunc(
   "fn_total_ingresos_presupuesto",
   float,
   [1]
)
```

31. Obtener total de gastos de presupuesto

Nombre: fn_totalgastos_presupuesto

Uso en Python (front):

```
total_gastos = cursor.callfunc(
   "fn_total_gastos_presupuesto",
   float,
   [1]
)
```

32. Obtener total ahorros presupuesto

Nombre: fn_avance_meta_ahorro

Uso en Python (front):

```
total_ahorros = cursor.callfunc(
   "fn_total_ahorros_presupuesto",
   float,
   [1]
)
```

33. Porcentaje de ejecución de subcategoria

Nombre: fn_porcentaje_ejecucion_subcategoria

Uso en Python (front):

```
porcentaje = cursor.callfunc(
   "fn_porcentaje_ejecucion_subcategoria",
   float,
   [1, 5]
)
```

34. Total ejecutado categoría

Nombre: fn_total_ejecutado_categoria

Uso en Python (front):

```
total_categoria = cursor.callfunc(
   "fn_total_ejecutado_categoria",
   float,
   [1, 2]
)
```

8. Triggers

8.1 Tabla resumen de triggers

Trigger	Tabla asociada	Momento	Tipo	Descripción
TRG_CATEGORIA_SUBCAT_DEF	CATEGORIAS / SUBCATEGORIAS	AFTER INSERT	Regla de negocio	Crea automáticamente una subcategoría por defecto al insertar una nueva categoría.
TRG_NO_TRANSACCION_PRESUPUESTO_CERRADO	TRANSACCIONES / PRESUPUESTOS	BEFORE INSERT	Validación	Evita registrar transacciones en presupuestos que no se encuentren

				en estado activo.
TRG_TRANSACCION_AHORRO_META	TRANSACCIONES / METAS_AHORROS	AFTER INSERT	Lógica de negocio	Actualiza automáticamente el avance de las metas de ahorro al registrar transacciones de tipo ahorro.
TRG_TRANSACCIONES_AUDIT_UPD	TRANSACCIONES	BEFORE UPDATE	Auditoría	Actualiza automáticamente la fecha de modificación de una transacción.

8.2 Justificación del uso de triggers

El uso de triggers en el Sistema de Gestión de Presupuestos Personales se justifica por la necesidad de garantizar la integridad de los datos, la correcta aplicación de reglas de negocio críticas y la automatización de procesos directamente en la base de datos, independientemente de la capa de presentación o de la lógica de aplicación.

Dado que el sistema maneja información financiera sensible, resulta fundamental que ciertas validaciones y acciones no dependan exclusivamente de la aplicación cliente, sino que estén protegidas y controladas a nivel de base de datos. Los triggers permiten asegurar que estas

reglas se ejecuten siempre que ocurran eventos específicos sobre las tablas, evitando inconsistencias y errores que podrían comprometer la confiabilidad de la información.

En primer lugar, los triggers se utilizan para reforzar reglas de negocio que no pueden ser violadas bajo ninguna circunstancia, como la restricción de registrar transacciones en presupuestos cerrados. Este tipo de validación es crítica, ya que garantiza que los datos financieros respeten el estado lógico del presupuesto, incluso si una inserción se realiza por medios distintos a la interfaz principal del sistema.

Asimismo, los triggers permiten la automatización de comportamientos derivados, como la creación automática de una subcategoría por defecto al insertar una nueva categoría. Esta lógica evita estados inconsistentes en el modelo de datos y asegura que toda categoría sea inmediatamente utilizable sin requerir acciones adicionales por parte del usuario o del desarrollador.

Otro uso importante de los triggers en el sistema es la actualización automática de información relacionada, como el avance de las metas de ahorro al registrar transacciones de tipo ahorro. Este enfoque centraliza el cálculo y la actualización del progreso directamente en la base de datos, garantizando coherencia entre las transacciones registradas y el estado actual de las metas, sin necesidad de cálculos redundantes en la aplicación.

Desde el punto de vista de la auditoría, los triggers permiten mantener actualizados los campos de control, como la fecha de modificación de los registros, de manera transparente y confiable. Esto asegura que toda modificación quede registrada correctamente, fortaleciendo la trazabilidad de los datos y facilitando futuras tareas de mantenimiento, validación o análisis histórico.

Finalmente, el uso de triggers contribuye a una arquitectura más robusta y profesional, donde la base de datos no se limita únicamente al almacenamiento de información, sino que actúa como un componente activo en la aplicación de reglas y validaciones. Este enfoque es coherente con prácticas utilizadas en sistemas reales de gestión financiera y contable, y refuerza el carácter académico y técnico del proyecto.

En conclusión, los triggers implementados permiten garantizar la consistencia, integridad y confiabilidad de la información financiera del sistema, automatizando procesos críticos y

asegurando el cumplimiento de las reglas de negocio definidas, independientemente del origen de las operaciones realizadas sobre la base de datos.

9. Backend en Python

9.1 Descripción general del backend

El backend del Sistema de Gestión de Presupuestos Personales fue desarrollado en Python y constituye la capa de lógica de negocio dentro de la arquitectura en tres capas del sistema. Su función principal es actuar como intermediario entre la interfaz gráfica desarrollada en Tkinter y la base de datos Oracle, coordinando el flujo de información, aplicando validaciones básicas y ejecutando las operaciones correspondientes mediante procedimientos almacenados.

El backend no accede directamente a las tablas de la base de datos para operaciones críticas, sino que delega estas acciones a procedimientos almacenados, garantizando que las reglas de negocio y la integridad de los datos se mantengan centralizadas en la base de datos.

9.2 Conexión con la base de datos

La comunicación con la base de datos Oracle se realiza mediante la librería oracledb, utilizando una función centralizada de conexión:

```
def conectar_oracle():
```

Esta función encapsula la lógica de conexión, permitiendo reutilizarla en todas las operaciones del sistema. En caso de error, el backend captura la excepción y notifica al usuario mediante mensajes visuales, evitando que el sistema falle de forma abrupta.

Este enfoque permite:

- Centralizar la configuración de la conexión.
- Facilitar el mantenimiento y posibles cambios futuros.

- Garantizar un manejo controlado de errores de base de datos.

9.3 Gestión de sesión

El backend implementa un manejo básico de sesión mediante variables globales:

- SESSION_USUARIO_ID
- SESSION_USUARIO_NOMBRE
- SESSION_PRESUPUESTO_ID

Estas variables permiten mantener el contexto del usuario autenticado durante la ejecución del sistema, facilitando la asociación correcta de presupuestos, transacciones, metas de ahorro y obligaciones fijas con el usuario activo.

Este mecanismo, aunque simple, resulta adecuado para el alcance académico del proyecto y permite simular el comportamiento de un sistema con sesiones persistentes.

9.4 Gestión de usuarios

El backend incluye funciones para el registro y validación de usuarios, las cuales interactúan directamente con procedimientos almacenados en la base de datos:

- Registro de nuevos usuarios mediante sp_insertar_usuario
- Validación de usuarios activos mediante sp_validar_usuario

Antes de realizar cualquier operación, el backend ejecuta validaciones básicas sobre los datos ingresados, como campos obligatorios y formatos correctos. La lógica de validación crítica (existencia, estado activo) se delega a la base de datos.

Este enfoque garantiza:

- Separación clara entre validación básica y reglas de negocio.
- Mayor seguridad y consistencia de los datos.
- Simulación de un sistema real de autenticación.

9.5 Gestión de presupuestos

El backend permite la creación y gestión de presupuestos mediante funciones que:

- Reciben los datos ingresados por el usuario.
- Realizan conversiones seguras de tipos numéricos y fechas.
- Validan la coherencia de los periodos de inicio y fin.
- Ejecutan el procedimiento almacenado sp_insertar_presupuesto.

Además, el backend controla el estado del presupuesto antes de permitir el registro de transacciones, reforzando a nivel de aplicación las reglas que posteriormente son validadas también en la base de datos.

9.6 Gestión de transacciones

El registro de transacciones es una de las funcionalidades centrales del backend. Para ello, el sistema:

- Valida el formato de fechas.
- Verifica que el presupuesto asociado se encuentre activo.
- Envía la información al procedimiento almacenado sp_insertar_transaccion.

El backend no calcula saldos ni actualiza metas directamente, ya que estas responsabilidades se encuentran delegadas a triggers y procedimientos en la base de datos, garantizando consistencia global independientemente del origen de la transacción.

9.7 Gestión de categorías y subcategorías

El backend permite la creación y consulta de categorías mediante funciones que llaman a procedimientos almacenados como sp_insertar_categoria y sp_listar_categorias.

La creación automática de subcategorías por defecto no se gestiona en Python, sino que se realiza mediante triggers en la base de datos, reduciendo la complejidad de la capa de aplicación y reforzando la integridad del modelo.

9.8 Gestión de obligaciones fijas y metas de ahorro

El backend implementa funciones específicas para:

- Registrar obligaciones fijas, validando fechas y montos.
- Registrar metas de ahorro, verificando campos obligatorios y formatos.
- Delegar el cálculo del progreso de las metas a la base de datos.

Este diseño permite que la aplicación se limite a capturar y enviar información, mientras que la lógica financiera compleja se mantiene centralizada en la base de datos.

9.9 Generación de reportes

El backend incluye funcionalidades para la generación de reportes básicos mediante consultas SQL, tales como:

- Gastos por categoría.
- Resumen general de ingresos, gastos y ahorros.
- Listado de metas de ahorro activas.

Estos reportes se presentan en la interfaz gráfica y sirven como complemento visual al análisis financiero realizado posteriormente mediante herramientas externas como Power BI.

9.10 Consideraciones finales del backend

El backend desarrollado en Python cumple el rol de orquestador del sistema, coordinando la interacción entre la interfaz gráfica y la base de datos sin duplicar reglas de negocio ni comprometer la integridad de la información. Su diseño modular, basado en funciones claramente definidas, facilita la comprensión del sistema y su mantenimiento, al mismo tiempo que se alinea con prácticas utilizadas en sistemas reales de gestión financiera

10. Interfaz gráfica del sistema (Tkinter)

10..1 Descripción general de la interfaz gráfica

La interfaz gráfica del Sistema de Gestión de Presupuestos Personales fue desarrollada utilizando Tkinter, la biblioteca estándar de Python para la creación de interfaces gráficas de escritorio. Esta interfaz constituye la capa de presentación del sistema y es el medio principal mediante el cual el usuario interactúa con las funcionalidades del sistema.

El diseño de la interfaz se orienta a ofrecer una experiencia clara y estructurada, permitiendo al usuario navegar entre las distintas funcionalidades sin necesidad de conocimientos técnicos. La interfaz se comunica exclusivamente con la capa lógica del sistema (backend en Python), evitando el acceso directo a la base de datos y manteniendo una separación adecuada de responsabilidades.

10.2 Estructura general de ventanas

La interfaz gráfica se organiza en ventanas independientes, cada una asociada a un módulo funcional específico del sistema. Este enfoque facilita la navegación, mejora la legibilidad del código y permite aislar cada funcionalidad de forma clara.

Las principales ventanas implementadas son:

- Ventana de inicio
- Ventana de registro de usuarios
- Ventana de inicio de sesión
- Menú principal
- Ventana de gestión de presupuestos
- Ventana de transacciones
- Ventana de categorías
- Ventana de obligaciones fijas
- Ventana de metas de ahorro
- Ventana de reportes

Cada ventana se implementa como un objeto Toplevel, lo que permite mantener múltiples vistas abiertas sin cerrar la aplicación principal.

10..3 Ventana de inicio y autenticación

La ventana de inicio actúa como punto de entrada al sistema, ofreciendo al usuario las opciones de crear una cuenta, iniciar sesión o salir de la aplicación. Esta ventana tiene un diseño simple y directo, reduciendo la fricción inicial del usuario.

Las ventanas de registro e inicio de sesión permiten capturar los datos necesarios para autenticar al usuario. En estas ventanas se realizan validaciones básicas de campos obligatorios antes de enviar la información al backend, el cual se encarga de ejecutar los procedimientos almacenados correspondientes para el registro y validación del usuario.

10.4 Menú principal

El menú principal funciona como el núcleo de navegación del sistema. Desde esta ventana, el usuario puede acceder a todas las funcionalidades principales mediante botones claramente identificados.

Entre las opciones disponibles se encuentran:

- Gestión de presupuestos
- Registro de transacciones
- Administración de categorías
- Gestión de obligaciones fijas
- Gestión de metas de ahorro
- Visualización de reportes
- Cierre de sesión

El menú principal también muestra información contextual, como el nombre del usuario que ha iniciado sesión, reforzando la noción de sesión activa dentro del sistema.

10.5 Formularios de captura de información

La mayoría de las ventanas del sistema utilizan formularios para la captura de información financiera. Estos formularios están compuestos por:

- Etiquetas descriptivas (Label)
- Campos de entrada (Entry)
- Botones de acción (Button)

Antes de enviar los datos al backend, la interfaz realiza validaciones mínimas, como la verificación de campos obligatorios y formatos básicos de fecha o valores numéricos. Este enfoque mejora la experiencia del usuario y reduce errores comunes de entrada de datos. Jojojo creo que nadie se dará cuenta si pongo un pequeño easter egg en medio de todo el documento, o si? Bromas inge. En el grupo de la clase mencionaron que mientras más llena estaba la documentación más probabilidades de recibir un buen puntaje teniamos, asi que aqui esta esta documentación de más de 100 páginas y personalmente creo que demasiado completa, tanto como para que alguien normal la lea completa siendo sincero.

11.6 Gestión visual de datos

Para la visualización de información estructurada, como listas de categorías o reportes, la interfaz utiliza el componente Treeview de la librería ttk. Este componente permite mostrar información en formato tabular, facilitando la lectura y comprensión de los datos.

Además, se implementan barras de desplazamiento (Scrollbar) para manejar volúmenes de información mayores, asegurando que la interfaz se mantenga usable incluso con un número elevado de registros.

11.7 Mensajes y retroalimentación al usuario

La interfaz gráfica proporciona retroalimentación constante al usuario mediante cuadros de diálogo (messagebox), los cuales informan sobre:

- Operaciones exitosas
- Errores de validación
- Errores de conexión o base de datos
- Restricciones de negocio (por ejemplo, presupuestos cerrados)

Esta comunicación directa mejora la usabilidad del sistema y permite al usuario comprender el resultado de sus acciones sin ambigüedades.

10.8 Diseño visual y experiencia de usuario

El diseño visual de la interfaz utiliza una paleta de colores oscuros y contrastantes, lo que mejora la legibilidad y reduce la fatiga visual. Se emplean estilos consistentes para botones, etiquetas y formularios, reforzando una identidad visual uniforme en todo el sistema.

Aunque el diseño no pretende competir con interfaces comerciales, cumple adecuadamente su función académica y funcional, priorizando la claridad, la organización y la facilidad de uso.

10.9 Relación con la arquitectura del sistema

La interfaz gráfica se comunica exclusivamente con el backend en Python, el cual a su vez interactúa con la base de datos mediante procedimientos almacenados. Esta separación garantiza que la interfaz no contenga lógica de negocio ni validaciones críticas, alineándose con la arquitectura en tres capas definida para el sistema.

Cualquier cambio en la lógica del sistema o en la base de datos puede realizarse sin afectar directamente la interfaz, siempre que se mantengan los contratos de comunicación definidos.

10.10 Consideraciones finales de la interfaz gráfica

La interfaz gráfica desarrollada con Tkinter cumple de manera efectiva su rol como capa de presentación del Sistema de Gestión de Presupuestos Personales. Su diseño modular, su integración con el backend y su enfoque en la experiencia del usuario permiten una interacción clara y ordenada con el sistema, facilitando el registro, análisis y visualización de la información financiera.

11. Reportes y visualización de datos

11.1 Introducción a los reportes

El sistema de gestión de presupuestos personales no se limita únicamente al registro y almacenamiento de información financiera, sino que también contempla la visualización y análisis de los datos como un componente clave para la toma de decisiones.

Para este propósito, se utilizaron reportes generados con Power BI, los cuales permiten transformar grandes volúmenes de datos financieros en información clara, resumida y visualmente comprensible.

Los reportes fueron construidos a partir de datasets estructurados que representan transacciones, categorías y periodos de tiempo, permitiendo analizar el comportamiento financiero del usuario desde distintas perspectivas.

11.2 Herramienta utilizada: Power BI

Power BI fue seleccionado como herramienta de visualización por las siguientes razones:

- Permite una integración sencilla con datasets en formato estructurado (CSV).

- Facilita la creación de gráficos interactivos sin necesidad de programación avanzada.

- Es ampliamente utilizada en entornos profesionales para análisis de datos.

- Permite representar grandes volúmenes de información de manera resumida y clara.

El uso de Power BI complementa el sistema, ya que permite analizar la información financiera sin afectar el rendimiento ni la lógica interna del sistema principal.

11..3 Reporte 1: Gastos por categoría

Descripción del reporte

El primer reporte presenta la **distribución de los gastos del usuario por categoría**, permitiendo identificar en qué áreas se concentra el mayor consumo de recursos financieros.

Este reporte se construyó utilizando una gráfica de barras/dona, donde cada categoría representa una porción del gasto total. El reporte permite visualizar rápidamente cuáles categorías tienen mayor impacto en el presupuesto del usuario.

Objetivo del reporte

- Identificar las categorías con mayor gasto acumulado.

- Facilitar la detección de patrones de consumo.

- Apoyar la toma de decisiones relacionadas con la reducción o redistribución del gasto.

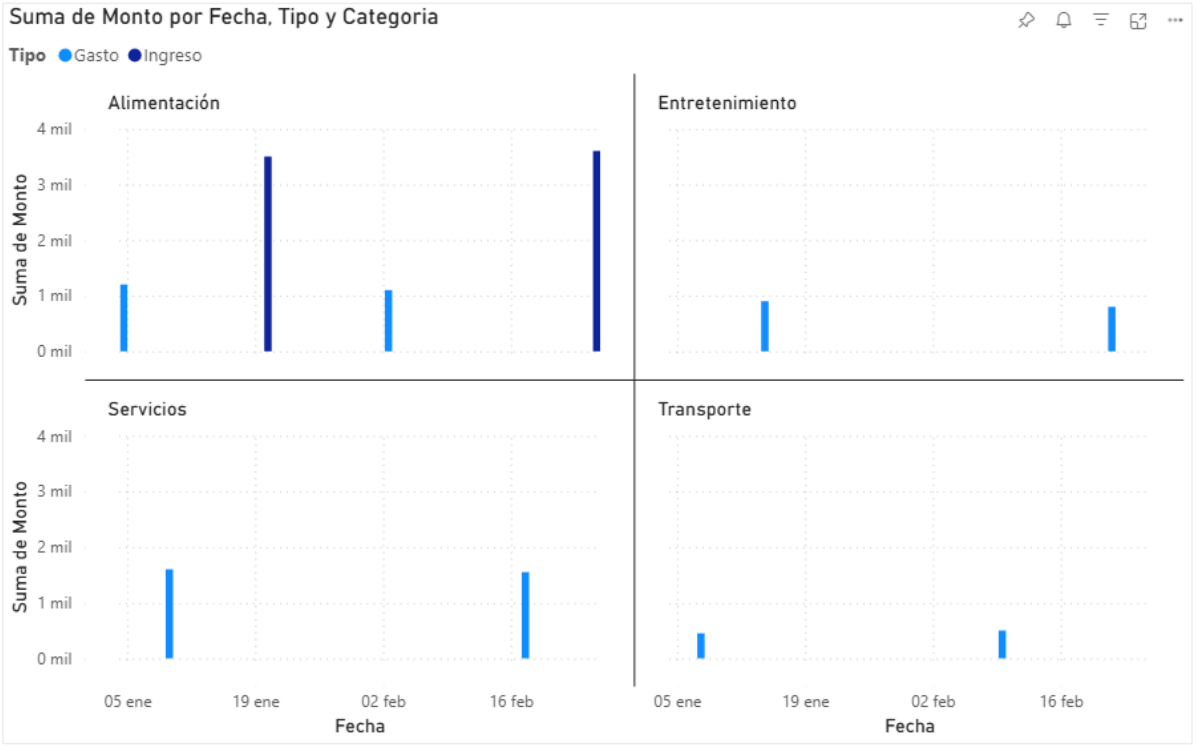
Información visualizada

- Categoría

- Monto total gastado

- Porcentaje del gasto total

📎 **Evidencia:**



Fuente: Elaboración propia mediante Power BI.

11.4 Reporte 2: Resumen financiero general

Descripción del reporte

El segundo reporte presenta un resumen financiero general, comparando los ingresos, gastos y ahorros del usuario en un periodo determinado. Este reporte ofrece una visión global del estado financiero y permite evaluar si el usuario mantiene un balance positivo.

El reporte se apoya en gráficos combinados y tarjetas de indicadores, mostrando totales acumulados y comparaciones claras entre los distintos tipos de movimientos financieros.

Objetivo del reporte

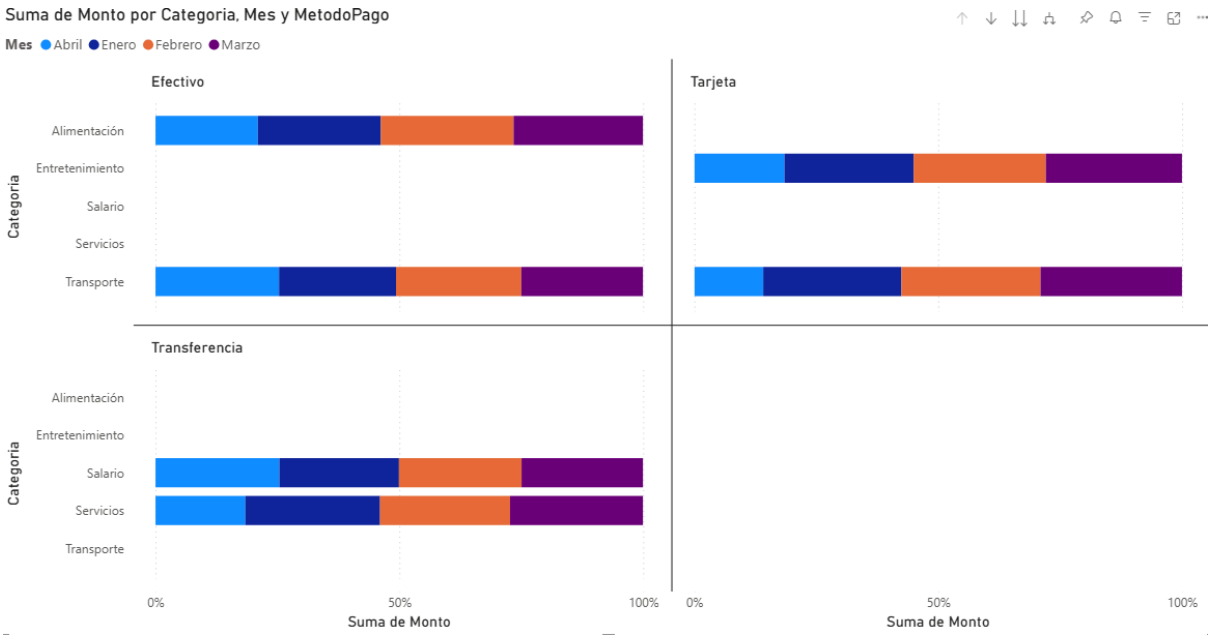
- Analizar la relación entre ingresos y gastos.

- Identificar si el usuario mantiene un balance financiero saludable.
- Evaluar el nivel de ahorro alcanzado en el periodo analizado.

Información visualizada

- Total de ingresos
- Total de gastos
- Total de ahorros
- Balance financiero general

📎 Evidencia:



Fuente: Elaboración propia mediante Power BI

11.5 Importancia de los reportes en el sistema

Los reportes generados permiten convertir datos financieros en **información útil**, facilitando la interpretación del comportamiento económico del usuario. A diferencia de los listados tradicionales, las visualizaciones gráficas permiten identificar tendencias, patrones y posibles áreas de mejora de forma rápida e intuitiva.

Además, el uso de Power BI demuestra la capacidad del sistema para integrarse con herramientas de análisis externas, reforzando su valor académico y su aplicabilidad en escenarios reales de análisis financiero.

11.6 Consideraciones finales sobre los reportes

Los reportes desarrollados cumplen una función complementaria al sistema principal, proporcionando una visión analítica de la información financiera almacenada. Aunque los datos utilizados pueden ser simulados con fines académicos, la estructura y el análisis realizados reflejan escenarios reales de gestión financiera personal.

La incorporación de Power BI como herramienta de visualización fortalece el proyecto, demostrando la importancia del análisis de datos como parte integral de un sistema de información moderno.

12. Manejo de errores

12.1 Introducción al manejo de errores

El manejo de errores es un componente esencial en el desarrollo de sistemas de información, especialmente cuando se trabaja con bases de datos relacionales que aplican restricciones de integridad y reglas de negocio. En el Sistema de Gestión de Presupuestos Personales, se implementó un manejo de errores orientado a detectar, controlar y comunicar adecuadamente los fallos que pueden ocurrir durante la ejecución de operaciones sobre la base de datos Oracle.

El sistema contempla tanto errores estándar de Oracle como errores personalizados, los cuales son capturados en la capa de backend y comunicados al usuario de manera comprensible a través de la interfaz gráfica.

12.2 Error ORA-02291 – Violación de clave foránea

Descripción del error

El error ORA-02291: integrity constraint violated - parent key not found ocurre cuando se intenta insertar o actualizar un registro que hace referencia a una clave foránea inexistente en la tabla padre.

En el contexto del sistema, este error puede presentarse, por ejemplo, al intentar registrar una transacción asociada a un usuario, presupuesto, categoría o subcategoría que no existe o que ha sido desactivada.

Manejo en el sistema

Este error es gestionado principalmente a nivel de base de datos mediante restricciones de integridad referencial. Cuando ocurre, el backend captura la excepción y muestra un mensaje claro al usuario, indicando que la información relacionada no existe o no es válida.

Este manejo evita:

- Registros huérfanos en la base de datos.
- Inconsistencias entre entidades relacionadas.
- Fallos silenciosos que comprometan la integridad del sistema.

12.3 Error ORA-01400 – Inserción de valores nulos

Descripción del error

El error ORA-01400: cannot insert NULL into se produce cuando se intenta insertar un valor nulo en una columna definida como obligatoria (NOT NULL).

En el sistema, este error puede presentarse si el usuario no completa campos obligatorios en formularios como:

- Registro de usuarios
- Creación de presupuestos
- Registro de transacciones
- Definición de metas de ahorro u obligaciones fijas

Manejo en el sistema

El sistema aplica un manejo preventivo y reactivo:

- **Preventivo:**
  La interfaz gráfica valida que los campos obligatorios estén completos antes de enviar la información al backend.

- **Reactivo:**
  En caso de que el error ocurra en la base de datos, el backend captura la excepción y muestra un mensaje informativo al usuario, indicando que existen campos obligatorios sin completar.

Este enfoque doble mejora la experiencia del usuario y refuerza la integridad de los datos.

12.4 Error ORA-20200 – Error personalizado de negocio

Descripción del error

El error ORA-20200 corresponde a un error personalizado, definido explícitamente en procedimientos almacenados o triggers mediante la instrucción RAISE_APPLICATION_ERROR. Este tipo de error se utiliza para comunicar violaciones a reglas de negocio que no pueden ser expresadas únicamente mediante restricciones estructurales.

En el sistema, este error se utiliza para situaciones como:

- Intentar registrar transacciones en presupuestos cerrados.
- Violaciones a reglas de vigencia de presupuestos.
- Inconsistencias en la relación entre transacciones, metas de ahorro u obligaciones fijas.

Manejo en el sistema

Cuando se produce un error ORA-20200, el backend captura el mensaje definido en la base de datos y lo presenta al usuario de forma clara, permitiendo comprender la razón exacta por la cual la operación fue rechazada.

Este enfoque permite:

- Centralizar las reglas de negocio en la base de datos.
- Proporcionar mensajes específicos y contextualizados.
- Evitar la duplicación de lógica en la capa de aplicación.

12.5 Comunicación de errores al usuario

Todos los errores capturados por el backend son comunicados al usuario mediante cuadros de diálogo en la interfaz gráfica. Los mensajes están diseñados para ser claros y comprensibles, evitando exponer detalles técnicos innecesarios, pero proporcionando suficiente información para corregir la acción realizada.

Este enfoque mejora la usabilidad del sistema y reduce la frustración del usuario ante errores comunes.

12.6 Importancia del manejo de errores en el sistema

El manejo adecuado de errores permite que el sistema sea más robusto, confiable y fácil de mantener. Al anticipar y controlar errores tanto estructurales como lógicos, el sistema garantiza que la información financiera almacenada sea consistente y que las reglas de negocio se respeten en todo momento.

Además, la inclusión de errores personalizados demuestra una comprensión avanzada del uso de Oracle como motor de base de datos y refuerza el carácter profesional y académico del proyecto.

13. Limitaciones del sistema

A pesar de que el Sistema de Gestión de Presupuestos Personales cumple con los objetivos planteados y ofrece una solución funcional para la administración y análisis de información financiera, es importante reconocer una serie de limitaciones propias de su alcance académico, las tecnologías utilizadas y las decisiones de diseño adoptadas durante su desarrollo.

13.1 Limitaciones tecnológicas

El sistema fue desarrollado como una aplicación de escritorio utilizando Tkinter, lo que implica que su ejecución está limitada a entornos locales. No se cuenta con acceso remoto ni

con una arquitectura web o cliente-servidor distribuida, lo que restringe su uso a un único equipo por sesión.

Asimismo, el uso de Oracle como motor de base de datos, aunque robusto y profesional, requiere una configuración previa y conocimientos técnicos para su instalación y administración, lo cual puede limitar su adopción en entornos no académicos o por usuarios sin experiencia técnica.

13.2 Limitaciones funcionales

El sistema está orientado a la gestión financiera de un usuario a la vez por sesión, por lo que no contempla escenarios de uso simultáneo por múltiples usuarios concurrentes.

Adicionalmente, el sistema no incluye funcionalidades avanzadas como:

- Predicción de gastos o ingresos futuros.
- Recomendaciones financieras automáticas.
- Análisis avanzado basado en inteligencia artificial.
- Integración con instituciones bancarias o plataformas de pago.

Estas funcionalidades fueron consideradas fuera del alcance del proyecto.

13.3 Limitaciones de seguridad

Aunque el sistema implementa controles básicos de seguridad y auditoría, no se incluyen mecanismos avanzados como:

- Encriptación de contraseñas.
- Autenticación multifactor.

- Gestión de roles y permisos detallados.
- Protección avanzada contra accesos no autorizados.

Estas medidas no fueron implementadas debido al enfoque académico del proyecto y a las restricciones de tiempo y alcance establecidas.

13.4 Limitaciones en la visualización de datos

Los reportes generados mediante **Power BI** se realizan utilizando datasets exportados y no están integrados de forma dinámica con la base de datos del sistema. Esto implica que los reportes no se actualizan en tiempo real y requieren la generación manual de los datos para su análisis.

Además, los reportes tienen un enfoque descriptivo y no predictivo, limitándose a representar información histórica.

13.5 Limitaciones de usabilidad

La interfaz gráfica, aunque funcional y clara, presenta limitaciones en cuanto a personalización y diseño visual avanzado. No se incluyen opciones como:

- Temas personalizables.
- Adaptación a distintos tamaños de pantalla.
- Accesibilidad avanzada para usuarios con discapacidades.

Estas características podrían mejorar la experiencia del usuario, pero no fueron prioritarias dentro del alcance del proyecto.

13.6 Consideraciones finales sobre las limitaciones

Las limitaciones identificadas no afectan el cumplimiento de los objetivos académicos del proyecto, sino que delimitan de manera clara el contexto en el que el sistema fue desarrollado. Reconocer estas limitaciones permite identificar oportunidades de mejora y posibles extensiones futuras, así como demostrar una comprensión realista del alcance y las capacidades del sistema.

14. Posibles mejoras futuras

El Sistema de Gestión de Presupuestos Personales establece una base funcional sólida para la administración y análisis de información financiera. No obstante, existen múltiples oportunidades de mejora que podrían implementarse en futuras versiones del sistema para ampliar sus capacidades, mejorar la experiencia del usuario y acercarlo a un entorno de producción real.

14.1 Evolución hacia una arquitectura web

Una de las principales mejoras futuras sería la migración del sistema hacia una arquitectura web, utilizando un backend basado en servicios y una interfaz accesible desde el navegador. Esto permitiría el acceso remoto al sistema, la gestión de múltiples usuarios concurrentes y una mayor escalabilidad.

La lógica de negocio actualmente centralizada en la base de datos y el backend facilita esta transición, ya que la separación de capas ya se encuentra definida.

14.2 Integración en tiempo real con herramientas de análisis

Actualmente, los reportes se generan a partir de datasets exportados. Una mejora futura sería la integración directa con herramientas de análisis, permitiendo la actualización automática de los reportes a partir de la base de datos, ya sea mediante conexiones en vivo o procesos de sincronización periódicos.

Esto permitiría análisis en tiempo real y una toma de decisiones más inmediata.

14.3 Implementación de alertas financieras

Aunque el módulo de alertas fue excluido de la implementación actual por indicaciones académicas, su incorporación futura permitiría notificar al usuario sobre:

- Excesos de gasto respecto al presupuesto.
- Vencimientos de obligaciones fijas.
- Avance o incumplimiento de metas de ahorro.

El diseño actual del sistema facilita esta mejora, ya que las reglas de negocio y la estructura de datos necesarias ya se encuentran definidas.

14.4 Mejora de la seguridad del sistema

En futuras versiones, se podrían implementar mecanismos de seguridad más avanzados, tales como:

- Encriptación de contraseñas.
- Autenticación basada en roles.
- Control de permisos por usuario.
- Registro detallado de accesos al sistema.

Estas mejoras permitirían elevar el sistema a un nivel más cercano a entornos productivos reales.

14.5 Automatización de transacciones recurrentes

Otra mejora relevante sería la automatización del registro de transacciones asociadas a obligaciones fijas, generando automáticamente los movimientos correspondientes según su periodicidad. Esto reduciría la intervención manual del usuario y aumentaría la precisión del registro financiero.

14.6 Análisis avanzado y predicción financiera

A largo plazo, el sistema podría incorporar funcionalidades de análisis avanzado, tales como:

- Proyección de gastos e ingresos futuros.
- Identificación de patrones de consumo.
- Recomendaciones financieras personalizadas.

Estas mejoras podrían apoyarse en técnicas de análisis estadístico o aprendizaje automático, utilizando los datos históricos almacenados en el sistema.

14.7 Mejoras en la experiencia de usuario

Finalmente, se podrían realizar mejoras orientadas a la experiencia del usuario, como:

- Diseño visual más moderno.
- Personalización de temas.
- Mejora en la navegación.
- Accesibilidad avanzada.

Estas mejoras permitirían una interacción más cómoda e intuitiva con el sistema.

14.8 Consideraciones finales

Las mejoras propuestas no afectan la funcionalidad actual del sistema, sino que representan posibles evoluciones que podrían desarrollarse en futuras iteraciones. La estructura del sistema y las decisiones de diseño adoptadas permiten que estas mejoras sean viables sin necesidad de reconstruir la solución desde cero.

15. Conclusiones

El desarrollo del Sistema de Gestión de Presupuesto Personal permitió la integración funcional y coherente de los conocimientos adquiridos en diferentes asignaturas de la carrera, especialmente los relacionados con bases de datos, programación y análisis de información. Durante la ejecución del proyecto, se logró construir una solución funcional a un problema real, como la gestión de las finanzas personales, aplicando los principios de diseño y las buenas prácticas propias de los sistemas de información.

Algunas de las conclusiones clave derivadas del proyecto se relacionan con la centralización de las reglas de negocio en la base de datos. El uso de procedimientos almacenados y disparadores permitió garantizar la precisión, consistencia y fiabilidad de los datos financieros, independientemente de la capa desde la que se realizaran las operaciones. Esto contribuyó a reducir las duplicaciones en la lógica de la aplicación y mejoró la robustez general del sistema.

Mediante la adopción de una arquitectura de tres niveles, se estableció una clara separación de responsabilidades entre la interfaz gráfica, la lógica de negocio y los niveles de datos. Esta estructura no solo mejoró la organización del código, sino que también facilitó la comprensión del flujo de información dentro del sistema y sentó las bases para futuras mejoras o ampliaciones.

Python como lenguaje backend, junto con Tkinter para la interfaz gráfica de usuario (GUI), resultó ser adecuado para un proyecto académico. Este desarrollo permitió crear una aplicación funcional y fácil de entender que podía interactuar con una base de datos robusta como Oracle sin comprometer la claridad del diseño ni la experiencia del usuario.

Por otro lado, la integración de informes y visualizaciones mediante Power BI demostró cómo el análisis de datos aportaba valor al registro de información. La capacidad de convertir datos financieros en representaciones visuales claras mejoró la visión del comportamiento financiero para el usuario y reforzó la utilidad del sistema para la toma de decisiones. Finalmente, el proyecto permitió comprender que el desarrollo de un sistema de información va más allá del desarrollo de funcionalidades visibles: comprender cómo tratar errores, auditar la información, definir reglas de negocio e identificar limitaciones son aspectos importantes para construir soluciones sólidas y sostenibles que se adapten a escenarios reales. El sistema desarrollado, en su conjunto, cumple con los objetivos propuestos y representa una experiencia integral de aprendizaje en el desarrollo de sistemas de información enfocados en la gestión financiera.

16. Anexos

16.1 Código main en Python:

```
import tkinter as tk

from tkinter import ttk, messagebox

from datetime import datetime

import oracledb

from datetime import date



import oracledb



# ---------------- SESIÓN ACTIVA ----------------

SESSION_USUARIO_ID = None

SESSION_USUARIO_NOMBRE = None

SESSION_PRESUPUESTO_ID = None




def conectar_oracle():

   try:
```

```
        connection = oracledb.connect(
            user="SisGestPresupuestos",
            password="1234",
            dsn="localhost/XEPDB1"
        )
        return connection
    except oracledb.DatabaseError as e:
        messagebox.showerror("Error", str(e))
        return None



def registrar_usuario(nombre, apellido, correo, salario):
    conn = conectar_oracle()
    if conn is None:
        return False


    try:
        cursor = conn.cursor()
```

```
        if not nombre or not apellido or not correo
or not salario:
            messagebox.showerror(
                "Error",
                "Nombre, apellido, correo y salario
son obligatorios."
            )
            return False


        cursor.callproc(
            "sp_insertar_usuario",
            [
                nombre,
                apellido,
                correo,
                float(salario),
                "admin"
            ]
```

```
        )

        conn.commit()
        messagebox.showinfo(
            "Registro exitoso",
                  f"Usuario  '{nombre}'  registrado
correctamente"
        )
        return True

    except oracledb.DatabaseError as e:
        messagebox.showerror(
            "Error",
            f"Error al registrar el usuario:\n{e}"
        )
        return False

    finally:
        cursor.close()
```

```
        conn.close()



# Ventana para registrar el usuario
def ventana_registro():
    reg = tk.Toplevel()
    reg.title("Registrar Usuario")
    reg.geometry("350x430")
    reg.configure(bg="#020617")
    tk.Label(
        reg, text="Registro de Usuario",
        font=("Segoe UI", 14, "bold"),
        fg="#e5e7eb",
        bg="#020617"
    ).pack(pady=10)
    form = tk.Frame(reg, bg="#020617")
    form.pack(pady=5)
    # Campos
```

```
    labels = ["Nombre", "Apellido", "Correo",
"Salario", "Contraseña"]

    entries = []

    for idx, texto in enumerate(labels):

        tk.Label(form, text=texto, font=("Segoe
UI", 9), fg="#e5e7eb",
bg="#020617").grid(row=idx*2, column=0,
sticky="w")

        entry = tk.Entry(form, show="*" if texto ==
"Contraseña" else "")

        entry.grid(row=idx*2 + 1, column=0,
pady=(0, 8), ipadx=60)

        entries.append(entry)



    def enviar_registro():

        nombre, apellido, correo, edad, salario,
password = [e.get() for e in entries]

        ok = registrar_usuario(nombre, apellido,
correo, edad, salario, password)

        if ok:

            ventana_menu_principal(nombre)
```

```
            reg.destroy()


    tk.Button(
        reg,
        text="Registrar",
        bg="#22c55e",
        fg="white",
        bd=0,
        font=("Segoe UI", 10, "bold"),
        activebackground="#16a34a",
        activeforeground="white",
        command=enviar_registro
    ).pack(pady=15)


def ingreso_usuario(correo, password):
                    global       SESSION_USUARIO_ID,
SESSION_USUARIO_NOMBRE


    conn = conectar_oracle()
```

```
    if conn is None:
        return None

    try:
        cursor = conn.cursor()

        nombre_out = cursor.var(str)
        activo_out = cursor.var(str)

        cursor.callproc(
            "sp_validar_usuario",
            [
                correo.strip().lower(),
                nombre_out,
                activo_out
            ]
        )

        nombre = nombre_out.getvalue()
```

```
        activo = activo_out.getvalue()


        if nombre is None:
                messagebox.showerror("Error",  "El
usuario no existe.")
            return None



        if activo is None or activo.strip().upper()
!= 'Y':
                messagebox.showerror("Error",  "El
usuario está inactivo.")
            return None


        return nombre


    except oracledb.DatabaseError as e:
        messagebox.showerror(
            "Error",
            f"Error al iniciar sesión:\n{e}"
```

```
        )
        return None

    finally:
        cursor.close()
        conn.close()




# Ventana de login para ingresar al sistema
def ventana_login():
    login = tk.Toplevel()
    login.title("Iniciar Sesión")
    login.geometry("350x260")
    login.configure(bg="#020617")
    tk.Label(
        login,
        text="Iniciar Sesión",
        font=("Segoe UI", 14, "bold"),
```

```
        fg="#e5e7eb",
        bg="#020617"
    ).pack(pady=10)
    form = tk.Frame(login, bg="#020617")
    form.pack(pady=5)
    tk.Label(form, text="Correo", font=("Segoe UI", 9),     fg="#e5e7eb",     bg="#020617").grid(row=0, column=0, sticky="w")
    entry_correo = tk.Entry(form)
      entry_correo.grid(row=1,  column=0,  pady=(0, 10), ipadx=60)
     tk.Label(form, text="Contraseña", font=("Segoe UI",  9),  fg="#e5e7eb",  bg="#020617").grid(row=2, column=0, sticky="w")
    entry_password = tk.Entry(form, show="*")
     entry_password.grid(row=3,  column=0,  pady=(0, 10), ipadx=60)


    def enviar_login():
```

```
                                        nombre      =
ingreso_usuario(entry_correo.get(),
entry_password.get())
        if nombre:
            ventana_menu_principal(nombre)
            login.destroy()


    tk.Button(
        login,
        text="Entrar",
        bg="#1d4ed8",
        fg="white",
        bd=0,
        font=("Segoe UI", 10, "bold"),
        activebackground="#1e40af",
        activeforeground="white",
        command=enviar_login
    ).pack(pady=15)
```

```
# Función para registrar un presupuesto
def registrarPresupuesto(
    nombre_presupuesto,
    anio_inicio,
    mes_inicio,
    anio_fin,
    mes_fin,
    ingresos,
    gastos,
    ahorro
):
    conn = conectar_oracle()
    if conn is None:
        return False

    try:
        cursor = conn.cursor()

        # Validaciones básicas
```

```
        if not nombre_presupuesto:
                messagebox.showerror("Error",  "El
nombre del presupuesto es obligatorio")
            return False


        # Conversión segura
        anio_inicio = int(anio_inicio)
        mes_inicio = int(mes_inicio)
        anio_fin = int(anio_fin)
        mes_fin = int(mes_fin)


        ingresos = float(ingresos)
        gastos = float(gastos)
        ahorro = float(ahorro)


                fecha_inicio  =  date(anio_inicio,
mes_inicio, 1)
        fecha_fin = date(anio_fin, mes_fin, 1)
```

```
        # ⚠️ ID DE USUARIO (TEMPORAL PARA DEFENSA)
           id_usuario = 1   # luego puedes hacerlo
dinámico
        SESSION_PRESUPUESTO_ID = cursor.lastrowid


        cursor.callproc(
            "sp_insertar_presupuesto",
            [
                id_usuario,
                nombre_presupuesto,
                fecha_inicio,
                fecha_fin,
                ingresos,
                gastos,
                ahorro,
                "admin"
            ]
        )
```

```
        conn.commit()
        return True

    except (ValueError, TypeError):
            messagebox.showerror("Error", "Datos
numéricos inválidos")
        return False

    except oracledb.DatabaseError as e:
        messagebox.showerror(
            "Error",
                        f"Error al registrar el
presupuesto:\n{e}"
        )
        return False

    finally:
        cursor.close()
        conn.close()
```

```
#Ventana Presupuesto
def ventana_presupuesto():
    win = tk.Toplevel()
    win.title("Registrar Presupuesto")
    win.geometry("650x500")
    win.configure(bg="#020617")

    form = tk.Frame(win, bg="#020617")
    form.pack(pady=5)

    # Campos de entrada
     labels = ["Nombre del presupuesto", "Año de inicio", "Mes de inicio (1-12)", "Año de fin", "Mes de fin (1-12)", "Total de ingresos", "Total de gastos", "Total de ahorro"]
    entries = []
    for idx, text in enumerate(labels):
```

```
            tk.Label(form, text=text, fg="white",
bg="#020617").grid(row=idx*2,            column=0,
sticky="w")

        entry = tk.Entry(form)

             entry.grid(row=idx*2 + 1, column=0,
pady=(0, 8), ipadx=60)

        entries.append(entry)


    def enviar_presupuesto():

           nombre_presupuesto, anio_ini, mes_ini,
anio_fin, mes_fin, ingresos, gastos, ahorro =
[entry.get() for entry in entries]

        if registrarPresupuesto(nombre_presupuesto,
anio_ini, mes_ini, anio_fin, mes_fin, ingresos,
gastos, ahorro):

                messagebox.showinfo("Presupuesto
registrado",        "Presupuesto        registrado
correctamente")

           for entry in entries:

               entry.delete(0, tk.END)
```

```
    # BOTÓN GUARDAR TRANSACCIÓN (ASEGÚRATE QUE ESTÉ AQUÍ)
    btn_guardar = tk.Button(
        win,
        text="Guardar transacción",
        bg="#22c55e",
        fg="white",
        bd=0,
        font=("Segoe UI", 10, "bold"),
        activebackground="#16a34a",
        activeforeground="white",
        width=22,
        command=enviar_transaccion
    )
    btn_guardar.pack(pady=10)


    # Función de regresar al menú principal
    def regresar_menu():
        win.destroy()
```

```
        ventana_menu_principal("Usuario")  # Cambia
"Usuario" por el nombre real del usuario


    tk.Button(
        win,
        text="Regresar al Menú Principal",
        bg="#2563eb",
        fg="white",
        bd=0,
        font=("Segoe UI", 10, "bold"),
        activebackground="#1e40af",
        activeforeground="white",
        width=22,
        command=regresar_menu
    ).pack(pady=10)


# Función para registrar una transacción
from datetime import date
```

```
from datetime import date


from datetime import date

def verificar_estado_presupuesto(presupuesto_id):

   conn = conectar_oracle()

   if conn is None:

       return False


   try:

       cursor = conn.cursor()


       cursor.execute("""

           SELECT estado

           FROM presupuestos

           WHERE id_presupuesto = :presupuesto_id

       """, {"presupuesto_id": presupuesto_id})


       resultado = cursor.fetchone()
```

```
        if resultado is None:
                messagebox.showerror("Error", "El presupuesto no existe.")
            return False

        if resultado[0] != 'ACTIVO':
                messagebox.showerror("Error", "No se pueden registrar transacciones en presupuestos cerrados.")
            return False

        return True

    except oracledb.DatabaseError as e:
            messagebox.showerror("Error", f"Error al verificar el estado del presupuesto:\n{e}")
        return False

    finally:
        cursor.close()
```

```
        conn.close()


from datetime import datetime


def registrar_transaccion(
    id_usuario,
    presupuesto_id,
    fecha_str,
    id_subcategoria,
    tipo,
    descripcion,
    monto,
    metodo_pago
):
    try:
            fecha  =  datetime.strptime(fecha_str,
"%Y-%m-%d")
    except ValueError:
        messagebox.showerror(
```

```
            "Error",
                "La fecha debe tener el formato
YYYY-MM-DD"
        )
        return False


    conn = conectar_oracle()
    cursor = conn.cursor()


    cursor.callproc(
        "sp_insertar_transaccion",
        [
            SESSION_USUARIO_ID,
            presupuesto_id,
            fecha,
            id_subcategoria,
            tipo,
            descripcion,
            monto,
```

```
            metodo_pago,
            "admin"  # usuario_creacion
        ]
    )

    conn.commit()
    cursor.close()
    conn.close()
    return True




def ventana_transacciones():
    print(">>> ventana_transacciones ejecutada")

    win = tk.Toplevel()
    win.title("Registrar Transacción")
    win.geometry("650x550")
```

```
    win.configure(bg="#020617")


    form = tk.Frame(win, bg="#020617")
    form.pack(pady=10)


    labels = [
        "Fecha (YYYY-MM-DD)",
        "ID Subcategoría",
        "Tipo (INGRESO/GASTO/AHORRO)",
        "Descripción",
        "Monto",
        "Método de pago"
    ]


    entries = []


    for i, text in enumerate(labels):
            tk.Label(form,  text=text,  fg="white",
bg="#020617")\
```

```
            .grid(row=i*2, column=0, sticky="w")
        e = tk.Entry(form, width=40)
        e.grid(row=i*2 + 1, column=0, pady=5)
        entries.append(e)


    def enviar_transaccion():
        print(">>> click guardar")


        fecha, id_subcat, tipo, desc, monto, metodo
= \
            [e.get() for e in entries]


        registrar_transaccion(
            SESSION_USUARIO_ID,
            1,  # presupuesto activo
            fecha,
            int(id_subcat),
            tipo.upper(),
            desc,
```

```
            float(monto),
            metodo
        )

    # 👇 BOTÓN GUARDA SIEMPRE
    tk.Button(
        win,
        text="Guardar Transacción",
        bg="#22c55e",
        fg="white",
        font=("Segoe UI", 10, "bold"),
        width=25,
        command=enviar_transaccion
    ).pack(pady=15)

    tk.Button(
        win,
        text="Regresar",
        bg="#2563eb",
```

```
        fg="white",
        width=25,
        command=win.destroy
    ).pack(pady=5)



#insertar categoria
def     insertar_categoria(nombre,     descripcion,
tipo_categoria, icono, color):
    """Insertar una nueva categoría en la base de
datos."""
    conn = conectar_oracle()
    if conn is None:
        return False

    try:
        cursor = conn.cursor()

        # Verificación de los campos
```

```
        if not nombre or not tipo_categoria:
            messagebox.showerror("Error", "Nombre y tipo de categoría son obligatorios.")
            return False

        # Llamar al procedimiento de base de datos para insertar la categoría
        cursor.callproc("sp_insertar_categoria", [nombre, descripcion, tipo_categoria, icono, color, "admin"])

        conn.commit()
        messagebox.showinfo("Categoría registrada", f"Categoría '{nombre}' registrada correctamente")
        return True

    except cx_Oracle.DatabaseError as e:
        messagebox.showerror("Error", f"Error al registrar la categoría: {e}")
        return False
    finally:
```

```
        cursor.close()
        conn.close()


def ventana_categoria():
    win = tk.Toplevel()
    win.title("Registrar Categoría")
    win.geometry("650x500")
    win.configure(bg="#020617")


    # Encabezado
    tk.Label(
        win,
        text="Crear Categoría",
        font=("Segoe UI", 14, "bold"),
        fg="#22c55e",
        bg="#020617"
    ).pack(pady=10)


    # Formulario de categoría
```

```
    form = tk.Frame(win, bg="#020617")
    form.pack(pady=5)

    # Campos de entrada
    tk.Label(form, text="Nombre de la categoría:", fg="white", bg="#020617").grid(row=0, column=0, sticky="w")
    entry_categoria = tk.Entry(form, width=30)
    entry_categoria.grid(row=1, column=0, pady=(0, 8))

        tk.Label(form, text="Tipo de categoría (ingreso/gasto):", fg="white", bg="#020617").grid(row=2, column=0, sticky="w")
    entry_tipo_categoria = tk.Entry(form, width=30)
        entry_tipo_categoria.grid(row=3, column=0, pady=(0, 8))

        tk.Label(form, text="Descripción de la categoría:", fg="white", bg="#020617").grid(row=4, column=0, sticky="w")
```

```
   entry_descripcion = tk.Entry(form, width=30)
         entry_descripcion.grid(row=5,   column=0,
pady=(0, 8))


    tk.Label(form, text="Color de la categoría:",
fg="white",   bg="#020617").grid(row=6,   column=0,
sticky="w")
   entry_color = tk.Entry(form, width=30)
   entry_color.grid(row=7, column=0, pady=(0, 8))


   def guardar_categoria():
       categoria = entry_categoria.get()
       tipo_categoria = entry_tipo_categoria.get()
       descripcion = entry_descripcion.get()
       color = entry_color.get()


       if categoria and tipo_categoria:
                  if  insertar_categoria(categoria,
descripcion, tipo_categoria, None, color):
```

```
                    messagebox.showinfo("Categoría
registrada", f"Categoría '{categoria}' registrada
correctamente.")
                entry_categoria.delete(0, tk.END)
                    entry_tipo_categoria.delete(0,
tk.END)
                entry_descripcion.delete(0, tk.END)
                entry_color.delete(0, tk.END)
                cargar_categorias()  # Recargar las
categorías después de guardar
        else:
                messagebox.showerror("Error", "El
nombre y el tipo de la categoría son
obligatorios")


    tk.Button(
        win,
        text="Guardar categoría",
        bg="#22c55e",
        fg="white",
        bd=0,
```

```
        font=("Segoe UI", 10, "bold"),
        activebackground="#16a34a",
        activeforeground="white",
        width=22,
        command=guardar_categoria
    ).pack(pady=5)

    # Tabla de Categorías
    tabla_frame = tk.Frame(win, bg="#020617")
          tabla_frame.pack(pady=10,   fill="both",
expand=True)

        columnas   =   ("ID",   "Nombre",   "Tipo",
"Descripción", "Color")

          tabla    =    ttk.Treeview(tabla_frame,
columns=columnas, show="headings", height=8)
    tabla.heading("ID", text="ID")
    tabla.heading("Nombre", text="Nombre")
    tabla.heading("Tipo", text="Tipo")
```

```
                    tabla.heading("Descripción",
text="Descripción")
    tabla.heading("Color", text="Color")


    tabla.column("ID", width=80)
    tabla.column("Nombre", width=150)
    tabla.column("Tipo", width=100)
    tabla.column("Descripción", width=150)
    tabla.column("Icono", width=100)


            tabla.pack(side="left",    fill="both",
expand=True)


        scrollbar   =   ttk.Scrollbar(tabla_frame,
orient="vertical", command=tabla.yview)
    scrollbar.pack(side="right", fill="y")
    tabla.configure(yscrollcommand=scrollbar.set)


    # Función para cargar las categorías
    def cargar_categorias():
```

```
        conn = conectar_oracle()
        if conn is None:
            return

        try:
            cursor = conn.cursor()

                # Cursor de salida correcto con
oracledb
                                    out_cursor    =
cursor.var(oracledb.CURSOR)

            cursor.callproc(
                "sp_listar_categorias",
                [
                    None,  # o 'INGRESO' / 'GASTO'
                    out_cursor
                ]
            )
```

```
            # Limpiar tabla
            for fila in tabla.get_children():
                tabla.delete(fila)

            # Insertar filas
            for categoria in out_cursor.getvalue():
                tabla.insert(
                    "",
                    "end",
                    values=(
                                categoria[0],   # id_categoria
                        categoria[1],  # nombre
                                categoria[2],   # tipo_categoria
                                categoria[3],   # descripcion
                        categoria[4]  # color
                    )
```

```
                )

        except oracledb.DatabaseError as e:
            messagebox.showerror(
                "Error",
                            f"Error al cargar las
categorías:\n{e}"
            )

        finally:
            cursor.close()
            conn.close()

    win.after(100, cargar_categorias)

    # Función de regresar al menú principal
    def regresar_menu():
        win.destroy()
        ventana_menu_principal("Usuario")
```

```
    tk.Button(
        win,
        text="Regresar al Menú Principal",
        bg="#2563eb",
        fg="white",
        bd=0,
        font=("Segoe UI", 10, "bold"),
        activebackground="#1e40af",
        activeforeground="white",
        width=22,
        command=regresar_menu
    ).pack(pady=10)


from datetime import datetime
import oracledb


def insertar_obligacion_fija(
    id_subcategoria,
```

```
    nombre,
    descripcion,
    monto_mensual,
    fecha_inicio_str,
    fecha_vencimiento_str
):
    conn = conectar_oracle()
    if conn is None:
        return False

    try:
        # Parsear fechas
        try:
                                    fecha_inicio   =
datetime.strptime(fecha_inicio_str, "%Y-%m-%d")
                                fecha_vencimiento   =
datetime.strptime(fecha_vencimiento_str,
"%Y-%m-%d")
        except ValueError:
            messagebox.showerror(
```

```
                "Error",
                 "Las fechas deben tener el formato
YYYY-MM-DD"
            )
            return False


        cursor = conn.cursor()


        cursor.callproc(
            "sp_insertar_obligacion_fija",
            [
                        SESSION_USUARIO_ID,          #
id_usuario
                id_subcategoria,
                nombre,
                descripcion,
                float(monto_mensual),
                fecha_inicio,
                fecha_vencimiento,
```

```
                    "admin"                                 #
usuario_creacion
                ]
            )


            conn.commit()
            messagebox.showinfo(
                "Obligación registrada",
                    f"Obligación '{nombre}' registrada
correctamente"
            )
            return True


        except oracledb.DatabaseError as e:
            messagebox.showerror(
                "Error",
                    f"Error al registrar la obligación
fija:\n{e}"
            )
            return False
```

```
    finally:
        cursor.close()
        conn.close()



def ventana_obligacion_fija():
    win = tk.Toplevel()
    win.title("Registrar Obligación Fija")
    win.geometry("650x500")
    win.configure(bg="#020617")


    # Encabezado
    tk.Label(
        win,
        text="Gestión de Obligaciones Fijas",
        font=("Segoe UI", 14, "bold"),
        fg="#22c55e",
        bg="#020617"
```

```
    ).pack(pady=10)


    # Formulario
    form = tk.Frame(win, bg="#020617")
    form.pack(pady=5)


    # Campos de entrada
        labels = ["Nombre", "Descripción", "Monto Mensual", "Fecha de inicio", "Fecha de vencimiento"]
    entries = []
    for idx, text in enumerate(labels):
            tk.Label(form, text=text, fg="white", bg="#020617").grid(row=idx*2, column=0, sticky="w")
        entry = tk.Entry(form)
            entry.grid(row=idx*2 + 1, column=0, pady=(0, 8), ipadx=60)
        entries.append(entry)
```

```
   def enviar_obligacion():

            nombre, descripcion, monto_mensual,
fecha_inicio, fecha_vencimiento = [entry.get() for
entry in entries]

          if insertar_obligacion_fija(1, nombre,
descripcion,      monto_mensual,      fecha_inicio,
fecha_vencimiento):  # Subcategoria ID de ejemplo:
1

                messagebox.showinfo("Obligación
registrada",    "Obligación    fija    registrada
correctamente")

          for entry in entries:

              entry.delete(0, tk.END)



   tk.Button(

       win,

       text="Guardar obligación fija",

       bg="#22c55e",

       fg="white",

       bd=0,

       font=("Segoe UI", 10, "bold"),
```

```
        activebackground="#16a34a",
        activeforeground="white",
        width=22,
        command=enviar_obligacion
    ).pack(pady=5)


    # Función de regresar al menú principal
    def regresar_menu():
        win.destroy()
        ventana_menu_principal("Usuario")


    tk.Button(
        win,
        text="Regresar al Menú Principal",
        bg="#2563eb",
        fg="white",
        bd=0,
        font=("Segoe UI", 10, "bold"),
        activebackground="#1e40af",
```

```
        activeforeground="white",
        width=22,
        command=regresar_menu
    ).pack(pady=10)


from datetime import datetime
import oracledb


def insertar_meta_ahorro(
    id_subcategoria,
    nombre,
    descripcion,
    monto_meta,
    fecha_inicio_str,
    fecha_objetivo_str,
    prioridad
):
    conn = conectar_oracle()
    if conn is None:
```

```
        return False


    try:
        # Validaciones mínimas
            if not nombre or not monto_meta or not
fecha_inicio_str or not fecha_objetivo_str or not
prioridad:
            messagebox.showerror(
                "Error",
                                "Todos los campos son
obligatorios."
            )
            return False


        # Parseo de fechas
        try:
                                    fecha_inicio   =
datetime.strptime(fecha_inicio_str, "%Y-%m-%d")
                                   fecha_objetivo   =
datetime.strptime(fecha_objetivo_str, "%Y-%m-%d")
```

```
        except ValueError:
            messagebox.showerror(
                "Error",
                 "Las fechas deben tener el formato YYYY-MM-DD"
            )
            return False

        cursor = conn.cursor()

        cursor.callproc(
            "sp_insertar_meta_ahorro",
            [
                    SESSION_USUARIO_ID,          # id_usuario
                id_subcategoria,
                nombre,
                descripcion,
                float(monto_meta),
```

```
                fecha_inicio,
                fecha_objetivo,
                prioridad.upper(),
                    "admin"                         #
usuario_creacion
            ]
        )

        conn.commit()

        messagebox.showinfo(
            "Meta de ahorro registrada",
             f"Meta de ahorro '{nombre}' registrada
correctamente"
        )
        return True

    except oracledb.DatabaseError as e:
        messagebox.showerror(
```

```
            "Error",
                f"Error al registrar la meta de
ahorro:\n{e}"
        )
        return False

    finally:
        cursor.close()
        conn.close()



def ventana_meta_ahorro():
    win = tk.Toplevel()
    win.title("Registrar Meta de Ahorro")
    win.geometry("650x500")
    win.configure(bg="#020617")

    # Encabezado
    tk.Label(
```

```
        win,
        text="Gestión de Metas de Ahorro",
        font=("Segoe UI", 14, "bold"),
        fg="#22c55e",
        bg="#020617"
    ).pack(pady=10)

    # Formulario
    form = tk.Frame(win, bg="#020617")
    form.pack(pady=5)

    # Campos de entrada
    labels = ["Nombre", "Descripción", "Monto Meta", "Fecha de inicio", "Fecha de objetivo", "Prioridad"]
    entries = []
    for idx, text in enumerate(labels):
        tk.Label(form, text=text, fg="white", bg="#020617").grid(row=idx*2, column=0, sticky="w")
```

```
        entry = tk.Entry(form)
            entry.grid(row=idx*2 + 1, column=0,
pady=(0, 8), ipadx=60)
        entries.append(entry)


    def enviar_meta():
            nombre, descripcion, monto_meta,
fecha_inicio, fecha_objetivo, prioridad = \
            [entry.get() for entry in entries]


        insertar_meta_ahorro(
            21,  # id_subcategoria (AHORRO)
            nombre,
            descripcion,
            monto_meta,
            fecha_inicio,
            fecha_objetivo,
            prioridad
        )
```

```
tk.Button(
    win,
    text="Guardar meta de ahorro",
    bg="#22c55e",
    fg="white",
    bd=0,
    font=("Segoe UI", 10, "bold"),
    activebackground="#16a34a",
    activeforeground="white",
    width=22,
    command=enviar_meta
).pack(pady=5)


# Función de regresar al menú principal
def regresar_menu():
    win.destroy()
    ventana_menu_principal("Usuario")
```

```
    tk.Button(
        win,
        text="Regresar al Menú Principal",
        bg="#2563eb",
        fg="white",
        bd=0,
        font=("Segoe UI", 10, "bold"),
        activebackground="#1e40af",
        activeforeground="white",
        width=22,
        command=regresar_menu
    ).pack(pady=10)



#     ----------------------     MENÚ     PRINCIPAL
---------------------- #
def ventana_menu_principal(nombre_usuario):
    menu = tk.Toplevel()
```

```
    menu.title("Sistema de Presupuesto - Menú Principal")
    menu.geometry("480x520")
    menu.configure(bg="#020617")
    # Encabezado
    topbar = tk.Frame(menu, bg="#020617")
    topbar.pack(fill="x", pady=10, padx=15)
    tk.Label(
        topbar,
        text="Sistema de Presupuesto",
        font=("Segoe UI", 16, "bold"),
        fg="#22c55e",
        bg="#020617"
    ).pack(side="left")
    tk.Label(
        topbar,
        text=f"Sesión de {nombre_usuario}",
        font=("Segoe UI", 9),
        fg="#9ca3af",
```

```
        bg="#020617"
    ).pack(side="right")
    main = tk.Frame(menu, bg="#020617")
    main.pack(pady=15)
    tk.Label(
        main,
        text="Menú principal",
        font=("Segoe UI", 12, "bold"),
        fg="#e5e7eb",
        bg="#020617"
    ).grid(row=0, column=0, columnspan=2, pady=(0,
15))


    def mk_btn(texto, row, col, command=None):
        btn = tk.Button(
            main,
            text=texto,
            width=22,
            height=2,
```

```
            bg="#111827",
            fg="#e5e7eb",
            bd=0,
            activebackground="#1f2937",
            activeforeground="white",
            font=("Segoe UI", 9, "bold"),
            command=command
        )
            btn.grid(row=row, column=col, padx=10,
pady=6)
        return btn
          mk_btn("Presupuesto",   1,   0,   lambda:
ventana_presupuesto())
         mk_btn("Transacciones",   1,   1,   lambda:
ventana_transacciones())
          mk_btn("Categorías",   2,   0,   lambda:
ventana_categoria())
     mk_btn("Obligaciones Fijas", 2, 1, lambda:
ventana_obligacion_fija())
      mk_btn("Metas de Ahorro", 3, 0, lambda:
ventana_meta_ahorro())
```

```
          mk_btn("Reportes",    3,    1,    lambda:
ventana_reportes())



   tk.Button(

       main,

       text="Cerrar sesión",

       width=22,

       height=2,

       bg="#b91c1c",

       fg="white",

       bd=0,

       activebackground="#7f1d1d",

       activeforeground="white",

       font=("Segoe UI", 9, "bold"),

       command=menu.destroy

   ).grid(row=4, column=1, padx=10, pady=15)

#    ----------------------    VENTANA    REGISTRO
---------------------- #

def ventana_registro():
```

```
    reg = tk.Toplevel()
    reg.title("Registrar Usuario")
    reg.geometry("350x500")
    reg.configure(bg="#020617")


    tk.Label(
        reg, text="Registro de Usuario",
        font=("Segoe UI", 14, "bold"),
        fg="#e5e7eb",
        bg="#020617"
    ).pack(pady=10)


    form = tk.Frame(reg, bg="#020617")
    form.pack(pady=5)


       labels = ["Nombre", "Apellido", "Correo",
"Edad", "Salario", "Contraseña"]
    entries = []
```

```
    for idx, texto in enumerate(labels):
        tk.Label(
            form,
            text=texto,
            font=("Segoe UI", 9),
            fg="#e5e7eb",
            bg="#020617"
        ).grid(row=idx*2, column=0, sticky="w")

        entry = tk.Entry(form, show="*" if texto ==
"Contraseña" else "")
            entry.grid(row=idx*2 + 1, column=0,
pady=(0, 8), ipadx=60)
        entries.append(entry)

    def enviar_registro():
        nombre = entries[0].get()
        apellido = entries[1].get()
        correo = entries[2].get()
```

```
        # entries[3] = edad → se ignora
        salario = entries[4].get()
        # entries[5] = contraseña → se ignora

          ok = registrar_usuario(nombre, apellido,
correo, salario)
        if ok:
            ventana_menu_principal(nombre)
            reg.destroy()

    tk.Button(
        reg,
        text="Registrar",
        bg="#22c55e",
        fg="white",
        bd=0,
        font=("Segoe UI", 10, "bold"),
        activebackground="#16a34a",
        activeforeground="white",
```

```
        command=enviar_registro
    ).pack(pady=15)


#-------------------ventana reportes son las 4:37 am dios mio-------------------
def ventana_reportes():
    win = tk.Toplevel()
    win.title("Reportes del Sistema")
    win.geometry("750x500")
    win.configure(bg="#020617")

    tk.Label(
        win,
        text="Reportes",
        font=("Segoe UI", 16, "bold"),
        fg="#22c55e",
        bg="#020617"
    ).pack(pady=10)
```

```
    tabla_frame = tk.Frame(win, bg="#020617")
        tabla_frame.pack(fill="both",  expand=True,
padx=10, pady=10)

    columnas = ("col1", "col2")
            tabla    =    ttk.Treeview(tabla_frame,
columns=columnas, show="headings")
    tabla.heading("col1", text="Descripción")
    tabla.heading("col2", text="Monto / Valor")

    tabla.column("col1", width=400)
    tabla.column("col2", width=200)

            tabla.pack(side="left",    fill="both",
expand=True)

        scrollbar   =   ttk.Scrollbar(tabla_frame,
orient="vertical", command=tabla.yview)
    scrollbar.pack(side="right", fill="y")
    tabla.configure(yscrollcommand=scrollbar.set)
```

```
    def limpiar_tabla():
        for fila in tabla.get_children():
            tabla.delete(fila)

    # ==============================
    # REPORTE 1: GASTOS POR CATEGORÍA
    # ==============================
    def reporte_gastos_categoria():
        limpiar_tabla()
        conn = conectar_oracle()
        cursor = conn.cursor()

        cursor.execute("""
            SELECT c.nombre, SUM(t.monto)
            FROM transacciones t
                        JOIN subcategorias s ON
s.id_subcategoria = t.id_subcategoria
```

```
            JOIN categorias c ON c.id_categoria =
s.id_categoria
            WHERE t.tipo_transaccion = 'GASTO'
            GROUP BY c.nombre
        """)

        for nombre, total in cursor.fetchall():
            tabla.insert("", "end", values=(nombre,
total))

        cursor.close()
        conn.close()

    # ==============================
    # REPORTE 2: RESUMEN PRESUPUESTO
    # ==============================
    def reporte_resumen_presupuesto():
        limpiar_tabla()
        conn = conectar_oracle()
```

```
        cursor = conn.cursor()


        cursor.execute("""
            SELECT
                SUM(CASE WHEN tipo_transaccion =
'INGRESO' THEN monto ELSE 0 END) AS ingresos,
                SUM(CASE WHEN tipo_transaccion =
'GASTO' THEN monto ELSE 0 END) AS gastos,
                SUM(CASE WHEN tipo_transaccion =
'AHORRO' THEN monto ELSE 0 END) AS ahorros
            FROM transacciones
        """)


                ingresos,  gastos,  ahorros  =
cursor.fetchone()


            tabla.insert("", "end", values=("Total
Ingresos", ingresos or 0))
            tabla.insert("", "end", values=("Total
Gastos", gastos or 0))
```

```
            tabla.insert("", "end", values=("Total
Ahorros", ahorros or 0))


        cursor.close()

        conn.close()


    # ==============================

    # REPORTE 3: METAS DE AHORRO

    # ==============================

    def reporte_metas_ahorro():

        limpiar_tabla()

        conn = conectar_oracle()

        cursor = conn.cursor()


        cursor.execute("""

            SELECT nombre, monto_meta

            FROM metas_ahorros

            WHERE estado = 'ACTIVA'

        """)
```

```
        for nombre, monto in cursor.fetchall():
            tabla.insert("", "end", values=(nombre,
monto))

        cursor.close()
        conn.close()

    botones = tk.Frame(win, bg="#020617")
    botones.pack(pady=10)

    tk.Button(
        botones,
        text="Gastos por Categoría",
        width=22,
        command=reporte_gastos_categoria
    ).grid(row=0, column=0, padx=5)

    tk.Button(
```

```
        botones,
        text="Resumen General",
        width=22,
        command=reporte_resumen_presupuesto
    ).grid(row=0, column=1, padx=5)

    tk.Button(
        botones,
        text="Metas de Ahorro",
        width=22,
        command=reporte_metas_ahorro
    ).grid(row=0, column=2, padx=5)

    tk.Button(
        win,
        text="Cerrar",
        bg="#b91c1c",
        fg="white",
        width=20,
```

```
        command=win.destroy
    ).pack(pady=10)



#     ----------------------     VENTANA     LOGIN
---------------------- #
def ventana_login():
    login = tk.Toplevel()
    login.title("Iniciar Sesión")
    login.geometry("350x260")
    login.configure(bg="#020617")
    tk.Label(
        login,
        text="Iniciar Sesión",
        font=("Segoe UI", 14, "bold"),
        fg="#e5e7eb",
        bg="#020617"
    ).pack(pady=10)
    form = tk.Frame(login, bg="#020617")
```

```
   form.pack(pady=5)

   tk.Label(form, text="Correo", font=("Segoe UI",
9),     fg="#e5e7eb",     bg="#020617").grid(row=0,
column=0, sticky="w")

   entry_correo = tk.Entry(form)

      entry_correo.grid(row=1,  column=0,  pady=(0,
10), ipadx=60)

    tk.Label(form, text="Contraseña", font=("Segoe
UI",  9),  fg="#e5e7eb",  bg="#020617").grid(row=2,
column=0, sticky="w")

   entry_password = tk.Entry(form, show="*")

    entry_password.grid(row=3,  column=0,  pady=(0,
10), ipadx=60)

   def enviar_login():

                                        nombre      =
ingreso_usuario(entry_correo.get(),
entry_password.get())

       if nombre:

           ventana_menu_principal(nombre)

           login.destroy()

   tk.Button(
```

```
        login,
        text="Entrar",
        bg="#1d4ed8",
        fg="white",
        bd=0,
        font=("Segoe UI", 10, "bold"),
        activebackground="#1e40af",
        activeforeground="white",
        command=enviar_login
    ).pack(pady=15)

#     ----------------------     VENTANA     INICIO
---------------------- #

def ventana_inicio():
    root = tk.Tk()
    root.title("Sistema de Presupuesto")
    root.geometry("420x320")
    root.configure(bg="#020617")
    header = tk.Frame(root, bg="#020617")
    header.pack(fill="x", pady=10)
```

```
    tk.Label(
        header,
        text="Sistema de Presupuesto",
        font=("Segoe UI", 18, "bold"),
        fg="#22c55e",
        bg="#020617"
    ).pack()
    card = tk.Frame(root, bg="#020617", bd=0)
    card.pack(pady=20)
    tk.Label(
        card,
        text="Acceso al sistema",
        font=("Segoe UI", 11, "bold"),
        fg="#e5e7eb",
        bg="#020617"
    ).grid(row=0, column=0, pady=(0, 15))
    tk.Button(
        card,
        text="Crear cuenta",
```

```
        width=20,
        height=2,
        bg="#22c55e",
        fg="white",
        bd=0,
        activebackground="#16a34a",
        activeforeground="white",
        font=("Segoe UI", 10, "bold"),
        command=ventana_registro
    ).grid(row=1, column=0, pady=5)
    tk.Button(
        card,
        text="Iniciar sesión",
        width=20,
        height=2,
        bg="#1d4ed8",
        fg="white",
        bd=0,
        activebackground="#1e40af",
```

```
        activeforeground="white",
        font=("Segoe UI", 10, "bold"),
        command=ventana_login
    ).grid(row=2, column=0, pady=5)
    tk.Button(
        card,
        text="Salir",
        width=20,
        height=2,
        bg="#b91c1c",
        fg="white",
        bd=0,
        activebackground="#7f1d1d",
        activeforeground="white",
        font=("Segoe UI", 10, "bold"),
        command=root.destroy
    ).grid(row=3, column=0, pady=(15, 0))


    root.mainloop()
```

```
#main ejecucion

if __name__ == "__main__":

   ventana_inicio()
```

16.2 Esquema de las tablas de la base de datos

```
--------------------------------------------------------
```

```
--  File created - Saturday-December-13-2025

--------------------------------------------------------

--------------------------------------------------------

--  DDL for Table USUARIOS

--------------------------------------------------------



  CREATE TABLE "SISGESTPRESUPUESTOS"."USUARIOS"

     (    "ID_USUARIO" NUMBER GENERATED ALWAYS AS IDENTITY MINVALUE 1
MAXVALUE 9999999999999999999999999999 INCREMENT BY 1 START WITH 1 CACHE
20 NOORDER  NOCYCLE  NOKEEP  NOSCALE ,

  "NOMBRES" VARCHAR2(100 BYTE),

  "APELLIDOS" VARCHAR2(100 BYTE),

  "CORREO" VARCHAR2(150 BYTE),

  "FECHA_REGISTRO" TIMESTAMP (6),

  "SALARIO" NUMBER,

  "ACTIVO" CHAR(1 BYTE),

  "FECHA_CREACION" TIMESTAMP (6) DEFAULT CURRENT_TIMESTAMP,

  "USUARIO_CREACION" VARCHAR2(50 BYTE),
```

```
  "FECHA_MODIFICACION" TIMESTAMP (6),
  "USUARIO_MODIFICACION" VARCHAR2(50 BYTE)
   ) SEGMENT CREATION IMMEDIATE
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
  TABLESPACE "USERS" ;
--------------------------------------------------------
--  DDL for Table TRANSACCIONES
--------------------------------------------------------

  CREATE TABLE "SISGESTPRESUPUESTOS"."TRANSACCIONES"
    (   "ID_TRANSACCION" NUMBER GENERATED ALWAYS AS IDENTITY MINVALUE 1
MAXVALUE 9999999999999999999999999999 INCREMENT BY 1 START WITH 1 CACHE
20 NOORDER  NOCYCLE  NOKEEP  NOSCALE ,
  "ID_USUARIO" NUMBER,
  "ID_PRESUPUESTO" NUMBER,
  "FECHA" TIMESTAMP (6),
  "ID_SUBCATEGORIA" NUMBER,
  "ID_OBLIGACION_FIJA" NUMBER,
  "TIPO_TRANSACCION" VARCHAR2(50 BYTE),
  "DESCRIPCION" VARCHAR2(200 BYTE),
  "MONTO" NUMBER,
  "METODO_PAGO" VARCHAR2(50 BYTE),
```

```
  "FECHA_REGISTRO" TIMESTAMP (6),
  "FECHA_CREACION" TIMESTAMP (6) DEFAULT CURRENT_TIMESTAMP,
  "USUARIO_CREACION" VARCHAR2(50 BYTE),
  "FECHA_MODIFICACION" TIMESTAMP (6),
  "USUARIO_MODIFICACION" VARCHAR2(50 BYTE)
   ) SEGMENT CREATION IMMEDIATE
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
  TABLESPACE "USERS" ;
--------------------------------------------------------
--  DDL for Table SUBCATEGORIAS
--------------------------------------------------------


  CREATE TABLE "SISGESTPRESUPUESTOS"."SUBCATEGORIAS"
   (  "ID_SUBCATEGORIA" NUMBER GENERATED ALWAYS AS IDENTITY MINVALUE 1
MAXVALUE 9999999999999999999999999999 INCREMENT BY 1 START WITH 1 CACHE
20 NOORDER  NOCYCLE  NOKEEP  NOSCALE ,
  "ID_CATEGORIA" NUMBER,
  "NOMBRE" VARCHAR2(100 BYTE),
  "ACTIVA" CHAR(1 BYTE),
  "SUBCATEGORIA_POR_DEFECTO" CHAR(1 BYTE)
   ) SEGMENT CREATION IMMEDIATE
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
```

```
 NOCOMPRESS LOGGING

  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645

  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1

  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)

  TABLESPACE "USERS" ;

--------------------------------------------------------

--  DDL for Table PRESUPUESTOS

--------------------------------------------------------



  CREATE TABLE "SISGESTPRESUPUESTOS"."PRESUPUESTOS"

    (   "ID_PRESUPUESTO" NUMBER GENERATED ALWAYS AS IDENTITY MINVALUE 1
MAXVALUE 9999999999999999999999999999 INCREMENT BY 1 START WITH 1 CACHE
20 NOORDER  NOCYCLE  NOKEEP  NOSCALE ,

  "ID_USUARIO" NUMBER,

  "NOMBRE_DESCRIPTIVO" VARCHAR2(150 BYTE),

  "FECHA_INICIO" TIMESTAMP (6),

  "FECHA_FIN" TIMESTAMP (6),

  "INGRESOS_PLANIFICADOS" NUMBER,

  "GASTOS_PLANIFICADOS" NUMBER,

  "AHORROS_PLANIFICADOS" NUMBER,

  "FECHA_CREACION" TIMESTAMP (6),

  "ESTADO" VARCHAR2(50 BYTE),

  "USUARIO_CREACION" VARCHAR2(50 BYTE),

  "FECHA_MODIFICACION" TIMESTAMP (6),

  "USUARIO_MODIFICACION" VARCHAR2(50 BYTE)

   ) SEGMENT CREATION IMMEDIATE
```

```
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
  TABLESPACE "USERS" ;
--------------------------------------------------------
--  DDL for Table PRESUPUESTO_DETALLES
--------------------------------------------------------


  CREATE TABLE "SISGESTPRESUPUESTOS"."PRESUPUESTO_DETALLES"
    (   "ID_PRESUPUESTO_DETALLE" NUMBER GENERATED ALWAYS AS IDENTITY
MINVALUE 1 MAXVALUE 9999999999999999999999999999 INCREMENT BY 1 START
WITH 1 CACHE 20 NOORDER  NOCYCLE  NOKEEP  NOSCALE ,
  "ID_PRESUPUESTO" NUMBER,
  "ID_SUBCATEGORIA" NUMBER,
  "MONTO_MENSUAL" NUMBER,
  "OBSERVACIONES" VARCHAR2(200 BYTE)
   ) SEGMENT CREATION IMMEDIATE
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
  TABLESPACE "USERS" ;
--------------------------------------------------------
```

```
--  DDL for Table OBLIGACIONES_FIJAS

--------------------------------------------------------



  CREATE TABLE "SISGESTPRESUPUESTOS"."OBLIGACIONES_FIJAS"

   (  "ID_OBLIGACION_FIJA" NUMBER GENERATED ALWAYS AS IDENTITY MINVALUE
1  MAXVALUE  99999999999999999999999999999  INCREMENT  BY  1  START  WITH  1
CACHE 20 NOORDER  NOCYCLE  NOKEEP  NOSCALE ,

  "ID_USUARIO" NUMBER,

  "ID_SUBCATEGORIA" NUMBER,

  "NOMBRE" VARCHAR2(100 BYTE),

  "DESCRIPCION" VARCHAR2(200 BYTE),

  "MONTO_MENSUAL" NUMBER,

  "FECHA_INICIO" TIMESTAMP (6),

  "FECHA_VENCIMIENTO" TIMESTAMP (6),

  "ACTIVA" CHAR(1 BYTE),

  "FECHA_CREACION" TIMESTAMP (6) DEFAULT CURRENT_TIMESTAMP,

  "USUARIO_CREACION" VARCHAR2(50 BYTE),

  "FECHA_MODIFICACION" TIMESTAMP (6),

  "USUARIO_MODIFICACION" VARCHAR2(50 BYTE)

   ) SEGMENT CREATION IMMEDIATE

  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255

 NOCOMPRESS LOGGING

  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645

  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1

  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)

  TABLESPACE "USERS" ;
```

```
--------------------------------------------------------

--  DDL for Table METAS_AHORROS

--------------------------------------------------------



  CREATE TABLE "SISGESTPRESUPUESTOS"."METAS_AHORROS"

    (   "ID_META_AHORRO" NUMBER GENERATED ALWAYS AS IDENTITY MINVALUE 1
MAXVALUE 9999999999999999999999999999 INCREMENT BY 1 START WITH 1 CACHE
20 NOORDER  NOCYCLE  NOKEEP  NOSCALE ,

  "ID_USUARIO" NUMBER,

  "ID_SUBCATEGORIA" NUMBER,

  "NOMBRE" VARCHAR2(100 BYTE),

  "DESCRIPCION" VARCHAR2(200 BYTE),

  "MONTO_META" NUMBER,

  "MONTO_AHORRADO" NUMBER,

  "FECHA_INICIO" TIMESTAMP (6),

  "FECHA_OBJETIVO" TIMESTAMP (6),

  "PRIORIDAD" VARCHAR2(50 BYTE),

  "ESTADO" VARCHAR2(50 BYTE),

  "FECHA_CREACION" TIMESTAMP (6) DEFAULT CURRENT_TIMESTAMP,

  "USUARIO_CREACION" VARCHAR2(50 BYTE),

  "FECHA_MODIFICACION" TIMESTAMP (6),

  "USUARIO_MODIFICACION" VARCHAR2(50 BYTE)

   ) SEGMENT CREATION IMMEDIATE

  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255

 NOCOMPRESS LOGGING

  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
```

```
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
  TABLESPACE "USERS" ;
--------------------------------------------------------
--  DDL for Table CATEGORIAS
--------------------------------------------------------


  CREATE TABLE "SISGESTPRESUPUESTOS"."CATEGORIAS"
    (   "ID_CATEGORIA" NUMBER GENERATED ALWAYS AS IDENTITY MINVALUE 1
MAXVALUE 9999999999999999999999999999 INCREMENT BY 1 START WITH 1 CACHE
20 NOORDER  NOCYCLE  NOKEEP  NOSCALE ,
  "NOMBRE" VARCHAR2(100 BYTE),
  "DESCRIPCION" VARCHAR2(150 BYTE),
  "TIPO_CATEGORIA" VARCHAR2(50 BYTE),
  "ICONO" VARCHAR2(50 BYTE),
  "COLOR" VARCHAR2(20 BYTE),
  "FECHA_CREACION" TIMESTAMP (6) DEFAULT CURRENT_TIMESTAMP,
  "USUARIO_CREACION" VARCHAR2(50 BYTE),
  "FECHA_MODIFICACION" TIMESTAMP (6),
  "USUARIO_MODIFICACION" VARCHAR2(50 BYTE)
   ) SEGMENT CREATION IMMEDIATE
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
```

```
  TABLESPACE "USERS" ;

--------------------------------------------------------

--  DDL for Index SYS_C008242

--------------------------------------------------------


      CREATE   UNIQUE   INDEX   "SISGESTPRESUPUESTOS"."SYS_C008242"   ON
"SISGESTPRESUPUESTOS"."USUARIOS" ("ID_USUARIO")

  PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS

  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645

  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1

  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)

  TABLESPACE "USERS" ;

--------------------------------------------------------

--  DDL for Index SYS_C008254

--------------------------------------------------------


      CREATE   UNIQUE   INDEX   "SISGESTPRESUPUESTOS"."SYS_C008254"   ON
"SISGESTPRESUPUESTOS"."TRANSACCIONES" ("ID_TRANSACCION")

  PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS

  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645

  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1

  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)

  TABLESPACE "USERS" ;

--------------------------------------------------------

--  DDL for Index SYS_C008246

--------------------------------------------------------
```

```
     CREATE   UNIQUE   INDEX   "SISGESTPRESUPUESTOS"."SYS_C008246"   ON
"SISGESTPRESUPUESTOS"."SUBCATEGORIAS" ("ID_SUBCATEGORIA")
  PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
  TABLESPACE "USERS" ;
--------------------------------------------------------
--  DDL for Index SYS_C008248
--------------------------------------------------------

     CREATE   UNIQUE   INDEX   "SISGESTPRESUPUESTOS"."SYS_C008248"   ON
"SISGESTPRESUPUESTOS"."PRESUPUESTOS" ("ID_PRESUPUESTO")
  PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
  TABLESPACE "USERS" ;
--------------------------------------------------------
--  DDL for Index SYS_C008250
--------------------------------------------------------

     CREATE   UNIQUE   INDEX   "SISGESTPRESUPUESTOS"."SYS_C008250"   ON
"SISGESTPRESUPUESTOS"."PRESUPUESTO_DETALLES" ("ID_PRESUPUESTO_DETALLE")
  PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
```

```
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
  TABLESPACE "USERS" ;
--------------------------------------------------------
--  DDL for Index SYS_C008252
--------------------------------------------------------


     CREATE   UNIQUE   INDEX   "SISGESTPRESUPUESTOS"."SYS_C008252"   ON
"SISGESTPRESUPUESTOS"."OBLIGACIONES_FIJAS" ("ID_OBLIGACION_FIJA")
  PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
  TABLESPACE "USERS" ;
--------------------------------------------------------
--  DDL for Index SYS_C008256
--------------------------------------------------------


     CREATE   UNIQUE   INDEX   "SISGESTPRESUPUESTOS"."SYS_C008256"   ON
"SISGESTPRESUPUESTOS"."METAS_AHORROS" ("ID_META_AHORRO")
  PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
  TABLESPACE "USERS" ;
```

```
--------------------------------------------------------

--  DDL for Index SYS_C008244

--------------------------------------------------------



     CREATE   UNIQUE   INDEX   "SISGESTPRESUPUESTOS"."SYS_C008244"   ON
"SISGESTPRESUPUESTOS"."CATEGORIAS" ("ID_CATEGORIA")

  PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS

  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645

  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1

  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)

  TABLESPACE "USERS" ;

--------------------------------------------------------

--  DDL for Trigger TRG_TRANSACCIONES_AUDIT_UPD

--------------------------------------------------------



            CREATE      OR      REPLACE      EDITIONABLE      TRIGGER
"SISGESTPRESUPUESTOS"."TRG_TRANSACCIONES_AUDIT_UPD"

BEFORE UPDATE ON TRANSACCIONES

FOR EACH ROW

BEGIN

    :NEW.FECHA_MODIFICACION := CURRENT_TIMESTAMP;

END;



/

ALTER    TRIGGER    "SISGESTPRESUPUESTOS"."TRG_TRANSACCIONES_AUDIT_UPD"
ENABLE;
```

```
--------------------------------------------------------

--  DDL for Trigger TRG_TRANSACCION_AHORRO_META

--------------------------------------------------------



            CREATE      OR      REPLACE      EDITIONABLE      TRIGGER
"SISGESTPRESUPUESTOS"."TRG_TRANSACCION_AHORRO_META"

AFTER INSERT ON TRANSACCIONES

FOR EACH ROW

BEGIN

    IF :NEW.TIPO_TRANSACCION = 'AHORRO' THEN

        sp_actualizar_metas_por_ahorro(

            :NEW.ID_USUARIO,

            :NEW.ID_SUBCATEGORIA,

            :NEW.MONTO

        );

    END IF;

END;



/

ALTER    TRIGGER    "SISGESTPRESUPUESTOS"."TRG_TRANSACCION_AHORRO_META"
ENABLE;

--------------------------------------------------------

--  DDL for Trigger TRG_NO_TRANSACCION_PRESUPUESTO_CERRADO

--------------------------------------------------------
```

```
            CREATE      OR      REPLACE      EDITIONABLE      TRIGGER
"SISGESTPRESUPUESTOS"."TRG_NO_TRANSACCION_PRESUPUESTO_CERRADO"

BEFORE INSERT ON TRANSACCIONES

FOR EACH ROW

DECLARE

    v_estado PRESUPUESTOS.ESTADO%TYPE;

BEGIN

    SELECT ESTADO

    INTO v_estado

    FROM PRESUPUESTOS

    WHERE ID_PRESUPUESTO = :NEW.ID_PRESUPUESTO;



    IF v_estado <> 'ACTIVO' THEN

        RAISE_APPLICATION_ERROR(

            -20200,

                'No se pueden registrar transacciones en presupuestos
cerrados'

        );

    END IF;

END;



/

ALTER                                                    TRIGGER
"SISGESTPRESUPUESTOS"."TRG_NO_TRANSACCION_PRESUPUESTO_CERRADO" ENABLE;

--------------------------------------------------------

--  DDL for Trigger TRG_CATEGORIA_SUBCAT_DEF
```

```
--------------------------------------------------------


            CREATE      OR      REPLACE      EDITIONABLE      TRIGGER
"SISGESTPRESUPUESTOS"."TRG_CATEGORIA_SUBCAT_DEF"
AFTER INSERT ON CATEGORIAS
FOR EACH ROW
BEGIN
    INSERT INTO SUBCATEGORIAS (
        ID_CATEGORIA,
        NOMBRE,
        ACTIVA,
        SUBCATEGORIA_POR_DEFECTO
    ) VALUES (
        :NEW.ID_CATEGORIA,
        'General',
        'Y',
        'Y'
    );
END;


/
ALTER TRIGGER "SISGESTPRESUPUESTOS"."TRG_CATEGORIA_SUBCAT_DEF" ENABLE;
--------------------------------------------------------
--  Constraints for Table USUARIOS
--------------------------------------------------------
```

```
  ALTER TABLE "SISGESTPRESUPUESTOS"."USUARIOS" MODIFY ("ID_USUARIO" NOT
NULL ENABLE);

    ALTER  TABLE  "SISGESTPRESUPUESTOS"."USUARIOS"  ADD  PRIMARY  KEY
("ID_USUARIO")

  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS

  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645

  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1

  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)

  TABLESPACE "USERS"  ENABLE;

  ALTER TABLE "SISGESTPRESUPUESTOS"."USUARIOS" MODIFY ("FECHA_CREACION"
NOT NULL ENABLE);

        ALTER    TABLE    "SISGESTPRESUPUESTOS"."USUARIOS"    MODIFY
("USUARIO_CREACION" NOT NULL ENABLE);

--------------------------------------------------------

--  Constraints for Table TRANSACCIONES

--------------------------------------------------------



      ALTER   TABLE   "SISGESTPRESUPUESTOS"."TRANSACCIONES"   MODIFY
("ID_TRANSACCION" NOT NULL ENABLE);

   ALTER  TABLE  "SISGESTPRESUPUESTOS"."TRANSACCIONES"  ADD  PRIMARY  KEY
("ID_TRANSACCION")

  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS

  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645

  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1

  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)

  TABLESPACE "USERS"  ENABLE;
```

```
       ALTER    TABLE    "SISGESTPRESUPUESTOS"."TRANSACCIONES"    MODIFY
("FECHA_CREACION" NOT NULL ENABLE);

       ALTER    TABLE    "SISGESTPRESUPUESTOS"."TRANSACCIONES"    MODIFY
("USUARIO_CREACION" NOT NULL ENABLE);

--------------------------------------------------------

--  Constraints for Table SUBCATEGORIAS

--------------------------------------------------------



       ALTER    TABLE    "SISGESTPRESUPUESTOS"."SUBCATEGORIAS"    MODIFY
("ID_SUBCATEGORIA" NOT NULL ENABLE);

   ALTER  TABLE  "SISGESTPRESUPUESTOS"."SUBCATEGORIAS"  ADD  PRIMARY  KEY
("ID_SUBCATEGORIA")

  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS

  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645

  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1

  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)

  TABLESPACE "USERS"  ENABLE;

--------------------------------------------------------

--  Constraints for Table PRESUPUESTOS

--------------------------------------------------------



       ALTER    TABLE    "SISGESTPRESUPUESTOS"."PRESUPUESTOS"    MODIFY
("ID_PRESUPUESTO" NOT NULL ENABLE);

   ALTER  TABLE  "SISGESTPRESUPUESTOS"."PRESUPUESTOS"  ADD  PRIMARY  KEY
("ID_PRESUPUESTO")

  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS

  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
```

```
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1

  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)

  TABLESPACE "USERS"  ENABLE;

       ALTER    TABLE    "SISGESTPRESUPUESTOS"."PRESUPUESTOS"    MODIFY
("USUARIO_CREACION" NOT NULL ENABLE);

--------------------------------------------------------

--  Constraints for Table PRESUPUESTO_DETALLES

--------------------------------------------------------



    ALTER  TABLE  "SISGESTPRESUPUESTOS"."PRESUPUESTO_DETALLES"  MODIFY
("ID_PRESUPUESTO_DETALLE" NOT NULL ENABLE);

  ALTER TABLE "SISGESTPRESUPUESTOS"."PRESUPUESTO_DETALLES" ADD PRIMARY
KEY ("ID_PRESUPUESTO_DETALLE")

  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS

  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645

  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1

  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)

  TABLESPACE "USERS"  ENABLE;

--------------------------------------------------------

--  Constraints for Table OBLIGACIONES_FIJAS

--------------------------------------------------------



    ALTER  TABLE  "SISGESTPRESUPUESTOS"."OBLIGACIONES_FIJAS"  MODIFY
("ID_OBLIGACION_FIJA" NOT NULL ENABLE);

  ALTER TABLE "SISGESTPRESUPUESTOS"."OBLIGACIONES_FIJAS" ADD PRIMARY
KEY ("ID_OBLIGACION_FIJA")

  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
```

```
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645

  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1

  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)

  TABLESPACE "USERS"  ENABLE;

     ALTER   TABLE   "SISGESTPRESUPUESTOS"."OBLIGACIONES_FIJAS"   MODIFY
("FECHA_CREACION" NOT NULL ENABLE);

     ALTER   TABLE   "SISGESTPRESUPUESTOS"."OBLIGACIONES_FIJAS"   MODIFY
("USUARIO_CREACION" NOT NULL ENABLE);

--------------------------------------------------------

--  Constraints for Table METAS_AHORROS

--------------------------------------------------------



       ALTER    TABLE    "SISGESTPRESUPUESTOS"."METAS_AHORROS"    MODIFY
("ID_META_AHORRO" NOT NULL ENABLE);

   ALTER  TABLE  "SISGESTPRESUPUESTOS"."METAS_AHORROS"  ADD  PRIMARY  KEY
("ID_META_AHORRO")

  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS

  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645

  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1

  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)

  TABLESPACE "USERS"  ENABLE;

       ALTER    TABLE    "SISGESTPRESUPUESTOS"."METAS_AHORROS"    MODIFY
("FECHA_CREACION" NOT NULL ENABLE);

       ALTER    TABLE    "SISGESTPRESUPUESTOS"."METAS_AHORROS"    MODIFY
("USUARIO_CREACION" NOT NULL ENABLE);

--------------------------------------------------------

--  Constraints for Table CATEGORIAS
```

```
--------------------------------------------------------


  ALTER TABLE "SISGESTPRESUPUESTOS"."CATEGORIAS" MODIFY ("ID_CATEGORIA"
NOT NULL ENABLE);

    ALTER  TABLE  "SISGESTPRESUPUESTOS"."CATEGORIAS"  ADD  PRIMARY  KEY
("ID_CATEGORIA")

  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS

  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645

  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1

  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)

  TABLESPACE "USERS"  ENABLE;

        ALTER    TABLE    "SISGESTPRESUPUESTOS"."CATEGORIAS"    MODIFY
("FECHA_CREACION" NOT NULL ENABLE);

        ALTER    TABLE    "SISGESTPRESUPUESTOS"."CATEGORIAS"    MODIFY
("USUARIO_CREACION" NOT NULL ENABLE);

--------------------------------------------------------

--  Ref Constraints for Table TRANSACCIONES

--------------------------------------------------------


    ALTER  TABLE  "SISGESTPRESUPUESTOS"."TRANSACCIONES"  ADD  CONSTRAINT
"FK_TRANS_USUARIO" FOREIGN KEY ("ID_USUARIO")

    REFERENCES "SISGESTPRESUPUESTOS"."USUARIOS" ("ID_USUARIO") ENABLE;

    ALTER  TABLE  "SISGESTPRESUPUESTOS"."TRANSACCIONES"  ADD  CONSTRAINT
"FK_TRANS_PRESUPUESTO" FOREIGN KEY ("ID_PRESUPUESTO")

     REFERENCES "SISGESTPRESUPUESTOS"."PRESUPUESTOS" ("ID_PRESUPUESTO")
ENABLE;

    ALTER  TABLE  "SISGESTPRESUPUESTOS"."TRANSACCIONES"  ADD  CONSTRAINT
"FK_TRANS_SUBCATEGORIA" FOREIGN KEY ("ID_SUBCATEGORIA")
```

```
                    REFERENCES     "SISGESTPRESUPUESTOS"."SUBCATEGORIAS"
("ID_SUBCATEGORIA") ENABLE;

    ALTER  TABLE  "SISGESTPRESUPUESTOS"."TRANSACCIONES"  ADD  CONSTRAINT
"FK_TRANS_OBLIGACION" FOREIGN KEY ("ID_OBLIGACION_FIJA")

                REFERENCES    "SISGESTPRESUPUESTOS"."OBLIGACIONES_FIJAS"
("ID_OBLIGACION_FIJA") ENABLE;

--------------------------------------------------------

--  Ref Constraints for Table SUBCATEGORIAS

--------------------------------------------------------



    ALTER  TABLE  "SISGESTPRESUPUESTOS"."SUBCATEGORIAS"  ADD  CONSTRAINT
"FK_SUBCATEGORIA_CATEGORIA" FOREIGN KEY ("ID_CATEGORIA")

        REFERENCES  "SISGESTPRESUPUESTOS"."CATEGORIAS"  ("ID_CATEGORIA")
ENABLE;

--------------------------------------------------------

--  Ref Constraints for Table PRESUPUESTOS

--------------------------------------------------------



    ALTER  TABLE  "SISGESTPRESUPUESTOS"."PRESUPUESTOS"  ADD  CONSTRAINT
"FK_PRESUPUESTO_USUARIO" FOREIGN KEY ("ID_USUARIO")

    REFERENCES "SISGESTPRESUPUESTOS"."USUARIOS" ("ID_USUARIO") ENABLE;

--------------------------------------------------------

--  Ref Constraints for Table PRESUPUESTO_DETALLES

--------------------------------------------------------



      ALTER   TABLE   "SISGESTPRESUPUESTOS"."PRESUPUESTO_DETALLES"   ADD
CONSTRAINT "FK_DET_PRESUPUESTO" FOREIGN KEY ("ID_PRESUPUESTO")
```

```
     REFERENCES "SISGESTPRESUPUESTOS"."PRESUPUESTOS" ("ID_PRESUPUESTO")
ENABLE;

     ALTER   TABLE   "SISGESTPRESUPUESTOS"."PRESUPUESTO_DETALLES"   ADD
CONSTRAINT "FK_DET_SUBCATEGORIA" FOREIGN KEY ("ID_SUBCATEGORIA")

                  REFERENCES     "SISGESTPRESUPUESTOS"."SUBCATEGORIAS"
("ID_SUBCATEGORIA") ENABLE;

--------------------------------------------------------

--  Ref Constraints for Table OBLIGACIONES_FIJAS

--------------------------------------------------------



  ALTER TABLE "SISGESTPRESUPUESTOS"."OBLIGACIONES_FIJAS" ADD CONSTRAINT
"FK_OBLIG_USUARIO" FOREIGN KEY ("ID_USUARIO")

    REFERENCES "SISGESTPRESUPUESTOS"."USUARIOS" ("ID_USUARIO") ENABLE;

  ALTER TABLE "SISGESTPRESUPUESTOS"."OBLIGACIONES_FIJAS" ADD CONSTRAINT
"FK_OBLIG_SUBCATEGORIA" FOREIGN KEY ("ID_SUBCATEGORIA")

                  REFERENCES     "SISGESTPRESUPUESTOS"."SUBCATEGORIAS"
("ID_SUBCATEGORIA") ENABLE;

--------------------------------------------------------

--  Ref Constraints for Table METAS_AHORROS

--------------------------------------------------------



    ALTER  TABLE  "SISGESTPRESUPUESTOS"."METAS_AHORROS"  ADD  CONSTRAINT
"FK_META_USUARIO" FOREIGN KEY ("ID_USUARIO")

    REFERENCES "SISGESTPRESUPUESTOS"."USUARIOS" ("ID_USUARIO") ENABLE;

    ALTER  TABLE  "SISGESTPRESUPUESTOS"."METAS_AHORROS"  ADD  CONSTRAINT
"FK_META_SUBCATEGORIA" FOREIGN KEY ("ID_SUBCATEGORIA")

                  REFERENCES     "SISGESTPRESUPUESTOS"."SUBCATEGORIAS"
("ID_SUBCATEGORIA") ENABLE;
```