**UNIVERSIDAD MAYOR REAL Y PONTIFICIA DE SAN FRANCISCO XAVIER DE CHUQUISACA**

**FACULTAD DE TECNOLOGÍA**



**Diseño e Implementación de una Base de Datos Relacional para la Gestión de Pedidos y Ventas de Librería**

**UNIVERSITARIOS:**

Aparicio Llanquipacha Gabriel Ing.Ciencias de la Computación.

Rua Echalar Juan Manuel Ing.Ciencias de la Computación.

Miguel Ángel Choque Garcia Ing. Ciencias de la Computación.

**MATERIA:** Bases de Datos III – SIS306.

**DOCENTE:** Ing. Espinoza Rodriguez Edgar Tito.

**SUCRE-BOLIVIA**

**Fase 1: Investigación y Análisis del Problema.**

1. **Antecedentes.**

Actualmente, la gestión de la información de los pedidos de libros se maneja a través de un registro plano (hoja de cálculo única) que carece de una estructura relacional adecuada. En este formato actual, se evidencia una mezcla desordenada de datos donde la información del cliente, los detalles del envío y los productos solicitados (libros) se encuentran en una sola fila.

Esta situación presenta problemas críticos de gestión de datos:

* **Grupos repetitivos y falta de atomicidad:** Existen celdas que contienen múltiples valores, como títulos de libros, autores y precios separados por comas, lo cual impide el procesamiento individual de cada artículo.
* **Redundancia e inconsistencia:** Los datos del cliente se repiten en cada pedido, lo que aumenta el riesgo de errores de escritura y anomalías de actualización.
* **Ausencia de historial:** No existe un mecanismo para rastrear cambios en los precios de los libros o la evolución del estado del pago ("Pendiente" a "Pagado") a lo largo del tiempo.

Por lo tanto, surge la necesidad de diseñar un sistema de base de datos que garantice la integridad, reduzca la redundancia y permita una administración eficiente de las ventas.

1. **Objetivos del Proyecto.**

**Objetivo General.** Diseñar e implementar una base de datos relacional robusta para la gestión de ventas y pedidos de una librería, aplicando metodologías de normalización y modelado de datos para garantizar la integridad y eficiencia en el acceso a la información.

**Objetivos Específicos**

* **Analizar y Normalizar:** Transformar la tabla original no normalizada aplicando las reglas de normalización (1FN, 2FN, 3FN y 4FN) para eliminar grupos repetitivos y dependencias parciales o transitivas.
* **Modelar Datos Históricos:** Diseñar estructuras para el manejo de **Catálogos e Históricos** (SCD Tipo 2), permitiendo registrar la variación de los precios de los libros en el tiempo y la trazabilidad de los estados de los pedidos.
* **Garantizar la Integridad Referencial:** Establecer restricciones de claves primarias y foráneas para asegurar la consistencia de los datos entre clientes, libros y pedidos.
* **Implementación Física:** Desarrollar el esquema físico en el SGBD ORACLE, incluyendo la creación de tablas, secuencias y *triggers* para la automatización de reglas de negocio.

1. **Requerimientos Funcionales.**

Basándonos en el análisis de la información actual y las necesidades operativas, el sistema debe cumplir con los siguientes requerimientos de datos y transacciones:

**Requerimientos de Gestión de Datos (Estáticos)** El sistema debe ser capaz de almacenar y administrar:

* **Catálogo de Libros:** Información detallada de cada título, incluyendo autor(es), género y precio actual.
* **Clientes:** Datos personales (nombre, correo) y múltiples direcciones de envío desglosadas (ciudad, calle, etc.) para evitar redundancias.
* **Pedidos:** Registro de la cabecera del pedido (fecha, cliente, monto total) y su detalle (libros específicos adquiridos y cantidad).
* **Histórico de Precios:** Registro de los cambios de precio de los libros para asegurar que los pedidos antiguos mantengan el precio vigente al momento de la compra.
* **Control de Estados:** Catálogo de estados del pago (Pagado, Pendiente) y el flujo histórico de cambios de estado de cada pedido.

**Requerimientos de Transacciones (Dinámicos)** El sistema debe permitir realizar las siguientes operaciones:

* Registrar nuevos pedidos validando la existencia del cliente y los libros.
* Actualizar el estado de un pedido (de "Pendiente" a "Pagado") manteniendo un registro de cuándo ocurrió el cambio.
* Consultar el historial de compras de un cliente específico.
* Calcular el monto total de un pedido basándose en el precio histórico del libro en la fecha de la compra, no en el precio actual.
* Gestionar la actualización de precios de libros sin afectar la integridad de las ventas pasadas.

1. **Requerimientos no Funcionales.**

Estos requisitos definen los criterios de calidad, restricciones técnicas y estándares que el sistema de base de datos debe cumplir para operar correctamente en el entorno de producción (Oracle).

**Eficiencia y Rendimiento**

* **Optimización de Consultas:** El sistema debe ser capaz de ejecutar consultas complejas (como el cálculo de montos históricos o reportes de ventas por género) en tiempos de respuesta aceptables.
* **Estrategia de Normalización y Desnormalización:** La base de datos se diseñará inicialmente cumpliendo estrictamente con la **4FN (Cuarta Forma Normal)** para garantizar la eliminación de redundancias y anomalías. No obstante, se permite la desnormalización controlada en tablas de reportes si las pruebas de rendimiento lo justifican.
* **Uso de Índices:** Se implementarán índices en las columnas de mayor frecuencia de búsqueda y en las claves foráneas para acelerar las operaciones de *JOIN* y filtrado.

**Integridad y Fiabilidad de los Datos**

* **Integridad Referencial Estricta:** El sistema no permitirá registros huérfanos. Toda venta debe estar asociada a un cliente y libros existentes; todo detalle de precio debe corresponder a un registro histórico válido.
* **Trazabilidad Histórica (Auditoría):** El sistema debe garantizar la persistencia de datos históricos. Los cambios en los precios de los libros o en los estados de los pedidos no deben sobrescribir la información pasada, asegurando que un pedido realizado ayer mantenga los valores vigentes en ese momento (Modelo SCD Tipo 2).

**Seguridad y Control de Acceso**

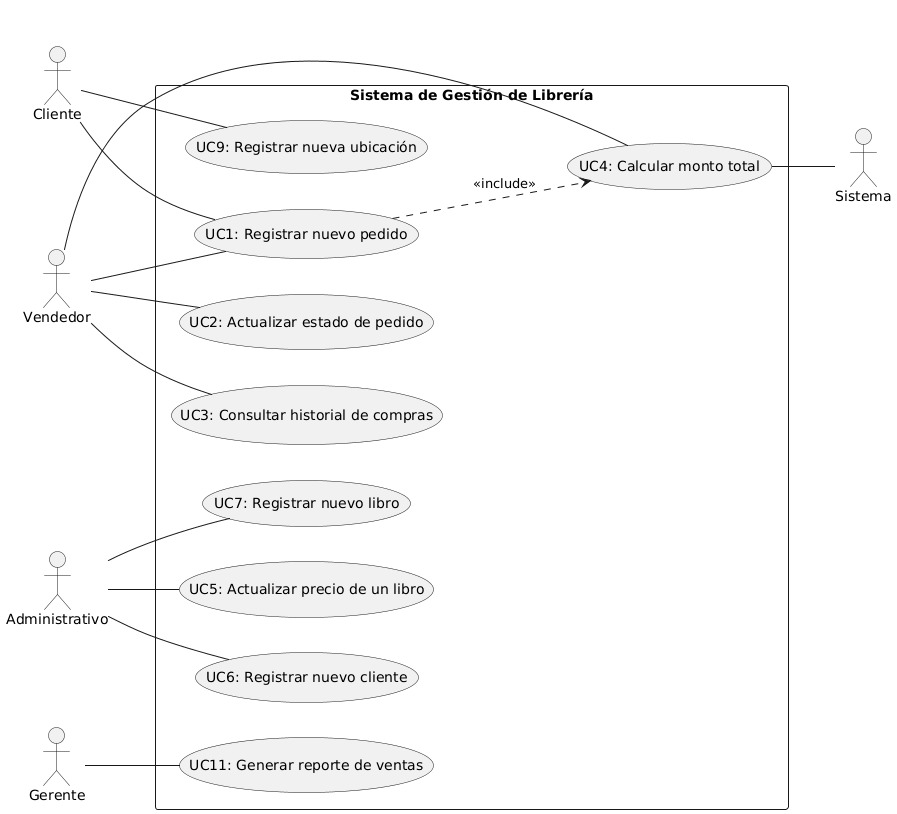
* **Gestión de Usuarios y Privilegios:** El sistema debe implementar un esquema de seguridad basado en roles, definiendo al menos tres perfiles con permisos diferenciados:
* *DBA:* Control total sobre la estructura y datos.
* *Gerente/Administrativo:* Permisos de lectura y generación de reportes financieros.
* *Vendedor/Operativo:* Permisos para registrar (INSERT) y actualizar (UPDATE) pedidos.

**Restricciones Físicas y Tecnológicas**

* **Plataforma SGBD:** La implementación debe realizarse obligatoriamente sobre el Sistema de Gestión de Bases de Datos **ORACLE**.
* **Almacenamiento:** Se debe realizar y documentar una estimación del espacio en disco requerido para las tablas y sus índices, previendo el crecimiento de los datos históricos a mediano plazo.

1. **Casos de uso**

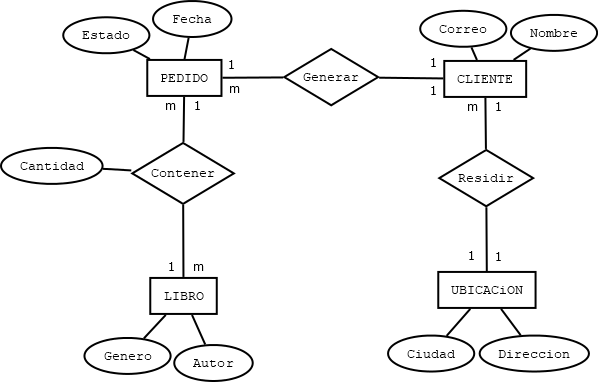
Figura 1. Diagrama de Casos de Uso



**Fase 2: Análisis del Sistema y Modelado Conceptual**

1. **Diagrama Entidad Relación.**

Figura 2. Diagrama Entidad Relación de la Base de Datos Inicial.



1. **Descripción del Problema.**

La organización (Librería) actualmente gestiona el registro de sus ventas y pedidos mediante un sistema de archivos planos (hojas de cálculo no estructuradas). En este sistema actual, toda la información relacionada con la transacción —datos del cliente, libros adquiridos, precios y estado del pago— se almacena en una única tabla general sin reglas de integridad.

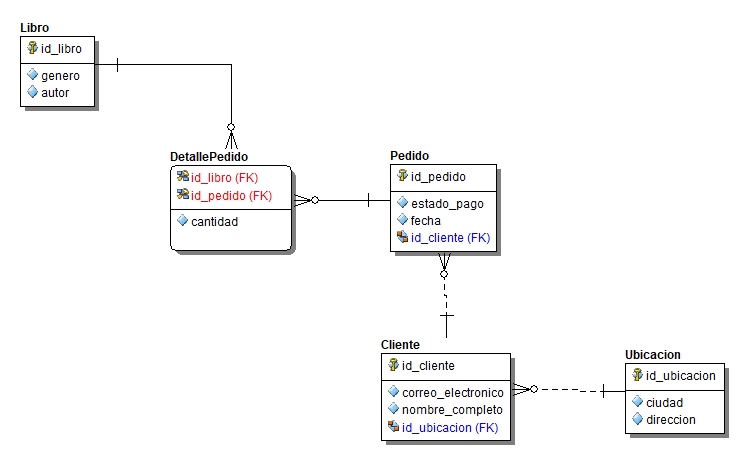
Al analizar la estructura de datos actual, se identifican deficiencias críticas en el manejo de la información:

* **Violación de Atomicidad:** Existen atributos multivaluados en una sola celda. Por ejemplo, la columna *"Titulo Libro"* contiene múltiples libros separados por comas (ej. *"SQL Básico, Modelado Datos"*), y sus respectivos precios y cantidades también están agrupados en cadenas de texto (ej. *"299.9, 399.9"* y *"2,1"*). Esto imposibilita realizar cálculos automáticos o búsquedas por producto específico.
* **Mezcla de Entidades:** La información del cliente (Nombre, Dirección, Correo) se repite en cada fila de pedido, mezclada con información de los productos y del envío.
* **Falta de Estandarización:** Direcciones compuestas (Calle, Ciudad, Código Postal) están aglomeradas en un solo campo de texto libre, dificultando la clasificación geográfica de las ventas.

**Fase 3: Diseño Lógico y Normalización.**

**Diseño Lógico:**

Figura 3. Modelo lógico después de normalizar a 4FN



**Tabla Original:**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pedido ID | Info Cliente | Correo Cliente | Titulo Libro | Autor Libro | Genero Libro | Precio Libro | Cantidad | Fecha Pedido | Monto Total | Estado Pago |
| 101 | Juan Pérez, Calle Luna 123, CDMX | juan@email.com | SQL Básico, Modelado Datos | A. Brown, B. White | Técnico, Técnico | 299.9, 399.9 | 2, 1 | 10/1/2025 | 999.7 | Pagado |
| 102 | Ana López, Av. Sol 456, Miami | ana@email.com | Guía Python | C. Green | Técnico | 499.9 | 1 | 12/1/2025 | 499.9 | Pendiente |
| 103 | Luis Gómez, Calle Estrella 789, Bogotá | luis@email.com | SQL Avanzado, Big Data | D. Black, E. Blue | Técnico, Técnico | 349.9, 599.9 | 1, 2 | 15/1/2025 | 1,549.70 | Pagado |
| 104 | María Ruiz, Av. Nube 101, CDMX | maria@email.com | Modelado Datos | B. White | Técnico | 399.9 | 3 | 18/1/2025 | 1,199.70 | Pagado |
| 105 | Carlos Díaz, Calle Sol 202, Lima | carlos@email.com | Guía Python, Big Data | C. Green, E. Blue | Técnico, Técnico | 499.9, 599.9 | 2, 1 | 20/1/2025 | 1,599.70 | Pendiente |
| 106 | Laura Vega, Av. Luna 303, Miami | laura@email.com | SQL Básico | A. Brown | Técnico | 299.9 | 1 | 22/1/2025 | 299.9 | Pagado |
| 107 | Pedro Mora, Calle Cielo 404, Bogotá | pedro@email.com | Big Data | E. Blue | Técnico | 599.9 | 2 | 25/1/2025 | 1,199.80 | Pagado |
| 108 | Sofía Castro, Av. Sol 505, CDMX | sofia@email.com | SQL Avanzado, Guía Python | D. Black, C. Green | Técnico, Técnico | 349.9, 499.9 | 1, 1 | 28/1/2025 | 849.8 | Pendiente |
| 109 | Diego León, Calle Luna 606, Lima | diego@email.com | Modelado Datos | B. White | Técnico | 399.9 | 2 | 30/1/2025 | 799.8 | Pagado |
| 110 | Elena Cruz, Av. Estrella 707, Miami | elena@email.com | SQL Básico, Big Data | A. Brown, E. Blue | Técnico, Técnico | 299.9, 599.9 | 1, 1 | 1/2/2025 | 899.8 | Pagado |

Se procederá a verificar si cumple cada Forma Normal empezando desde la primera

**1. Cumplimiento de la 1FN (Primera Forma Normal).**

No cumple con la 1FN, hay tuplas que no cumplen con la definición de atomicidad:

Para cumplir con la 1FN haremos que cada celda de cada tupla contenga un valor atómico:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pedido ID | Cliente | Dirección | Ciudad | Correo Cliente | Titulo Libro | Autor Libro | Genero Libro | Precio Libro | Cantidad | Fecha Pedido | Monto Total | Estado Pago |
| 101 | Juan Pérez | Calle Luna 123 | CDMX | juan@email.com | SQL Básico | A. Brown | Técnico | 299.9 | 2 | 10/1/2025 | 999.7 | Pagado |
| 101 | Juan Pérez | Calle Luna 123 | CDMX | juan@email.com | Modelado Datos | B. White | Técnico | 399.9 | 1 | 10/1/2025 | 999.7 | Pagado |
| 102 | Ana López | Av. Sol 456 | Miami | ana@email.com | Guía Python | C. Green | Técnico | 499.9 | 1 | 12/1/2025 | 499.9 | Pendiente |
| 103 | Luis Gómez | Calle Estrella 789 | Bogotá | luis@email.com | SQL Avanzado | D. Black | Técnico | 349.9 | 1 | 15/1/2025 | 1 | Pagado |
| 103 | Luis Gómez | Calle Estrella 789 | Bogotá | luis@email.com | Big Data | E. Blue | Técnico | 599.9 | 2 | 15/1/2025 | 549.70 | Pagado |
| 104 | María Ruiz | Av. Nube 101 | CDMX | maria@email.com | Modelado Datos | B. White | Técnico | 399.9 | 3 | 18/1/2025 | 1,199.70 | Pagado |
| 105 | Carlos Díaz | Calle Sol 202 | Lima | carlos@email.com | Guía Python | C. Green | Técnico | 499.9 | 2 | 20/1/2025 | 1,599.70 | Pendiente |
| 105 | Carlos Díaz | Calle Sol 202 | Lima | carlos@email.com | Big Data | E. Blue | Técnico | 599.9 | 1 | 20/1/2025 | 1,599.70 | Pendiente |
| 106 | Laura Vega | Av. Luna 303 | Miami | laura@email.com | SQL Básico | A. Brown | Técnico | 299.9 | 1 | 22/1/2025 | 299.9 | Pagado |
| 107 | Pedro Mora | Calle Cielo 404 | Bogotá | pedro@email.com | Big Data | E. Blue | Técnico | 599.9 | 2 | 25/1/2025 | 1,199.80 | Pagado |
| 108 | Sofía Castro | Av. Sol 505 | CDMX | sofia@email.com | SQL Avanzado | D. Black | Técnico | 349.9 | 1 | 28/1/2025 | 849.8 | Pendiente |
| 108 | Sofía Castro | Av. Sol 505 | CDMX | sofia@email.com | Guía Python | C. Green | Técnico | 499.9 | 1 | 28/1/2025 | 849.8 | Pendiente |
| 109 | Diego León | Calle Luna 606 | Lima | diego@email.com | Modelado Datos | B. White | Técnico | 399.9 | 2 | 30/1/2025 | 799.8 | Pagado |
| 110 | Elena Cruz | Av. Estrella 707 | Miami | elena@email.com | SQL Básico | A. Brown | Técnico | 299.9 | 1 | 1/2/2025 | 899.8 | Pagado |
| 110 | Elena Cruz | Av. Estrella 707 | Miami | elena@email.com | Big Data | E. Blue | Técnico | 599.9 | 1 | 1/2/2025 | 899.8 | Pagado |

**2. Identificación de las Claves Candidatas.**

Una vez en 1FN se procederá a identificar llaves candidatas, pero antes procederemos a renombrar cada columna:

Una vez renombrado se hará el análisis de dependencias funcionales, basado en la definición formal:

Se observa que el atributo A (Pedido ID) determina funcionalmente a toda la información general de la transacción y del cliente asociado en ese momento:

Sin embargo, A no puede determinar por sí solo los detalles del producto ni la cantidad específica:

El atributo F (Título Libro) determina las características inherentes al libro:

Dado que ni A ni F por separado logran identificar una fila única en la tabla universal, se establece que la Clave Candidata es compuesta:

**Identificación de Anomalías.**

Para avanzar hacia la 2FN y 3FN, se detectan las siguientes irregularidades que deben ser resueltas mediante descomposición:

1. **Dependencias Transitivas (Datos del Cliente):** Dentro de las dependencias de **A**, existe un subconjunto de datos que se definen entre sí. El **Correo (E)** actúa como un identificador fuerte para el Cliente, determinando su Nombre y Dirección.

A su vez, existe una dependencia geográfica interna, donde la Dirección física determina la Ciudad:

Esto genera la cadena transitiva:

1. **Atributos Derivados (Redundancia):** El atributo **M (Monto Total)** es un dato calculado resultante de la sumatoria de agrupada por pedido, es decir:

Almacenar este dato genera **Anomalías de Actualización**: si se modifica una cantidad o precio en el detalle, el total en la cabecera pierde integridad. Por tanto, se considera un atributo derivado a excluir del modelo físico base.

**Resumen de la identificación:**

****

**Nota:** Aunque en la instancia de datos actual la columna H (Género) presenta un valor constante ("Técnico"), lógica y funcionalmente es un atributo dependiente del libro (F), no del pedido ni del cliente.

**3. Cumplimiento de la 2FN (Segunda Forma Normal).**

Nuestra Base de Datos actualmente es:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pedido ID | Cliente | Dirección | Ciudad | Correo Cliente | Titulo Libro | Autor Libro | Genero Libro | Precio Libro | Cantidad | Fecha Pedido | Monto Total | Estado Pago |

No cumple con la 2FN. La relación original R posee una clave candidata compuesta . Para cumplir con la 2FN, eliminamos las Dependencias Parciales, es decir, atributos que dependen solo de una parte de la clave.

Para la tabla inicial tendremos:

|  |  |  |
| --- | --- | --- |
| Pedido ID | Titulo Libro | Cantidad |

Esto dada la dependencia funcional:

Se crea una nueva tabla para las columnas dependientes de A:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Pedido ID | Cliente | Dirección | Ciudad | Correo Cliente | Fecha Pedido | Monto Total | Estado Pago |

Y finalmente una nueva tabla para las columnas dependientes de F:

|  |  |  |  |
| --- | --- | --- | --- |
| Titulo Libro | Autor Libro | Genero Libro | Precio Libro |

Con estos pasos se han eliminado dependencias parciales.

**4. Cumplimiento de la 3FN (Tercera Forma Normal).**

No cumple con la 3FN, se detectan Dependencias Transitivas (un atributo no clave determina a otro no clave) en la tabla:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pedido ID | Cantidad | Fecha Pedido | Monto Total | Estado Pago | Correo Cliente |

Y para las columnas dependientes de E:

|  |  |  |  |
| --- | --- | --- | --- |
| Correo Cliente | Cliente | Dirección | Ciudad |

En esta tabla también existen dependencias transitivas , por lo que se procederá a separarse:

|  |  |  |
| --- | --- | --- |
| Correo Cliente | Cliente | Dirección |

Y para C:

|  |  |
| --- | --- |
| Dirección | Ciudad |

**5. Cumplimiento de la FNBC (Forma Normal Boyce Codd).**

Se procede a verificar si las relaciones resultantes de la 3FN cumplen con la Forma Normal de Boyce-Codd. La definición establece que una tabla está en FNBC si, para toda dependencia funcional no trivial , donde X es una superclave.

Analizando las tablas resultantes:

* **En :** La clave es A, F. La única dependencia es . El determinante es la clave completa. **Cumple.**
* **En :** La clave es F. La dependencia es . El determinante es la clave. **Cumple.**
* **En :** La clave es E. La dependencia es . El determinante es la clave. **Cumple.**
* **En :** La clave es C. La dependencia es . El determinante es la clave. **Cumple.**
* **En :** La clave es A. La dependencia es . El determinante es la clave. **Cumple.**

**Conclusión:** Dado que en todas las relaciones resultantes los únicos determinantes son las claves candidatas, el modelo se encuentra automáticamente en **FNBC**. No existen anomalías de claves superpuestas.

**6. Cumplimiento de la 4FN (Cuarta Forma Normal).**

Finalmente, se valida la ausencia de Dependencias Multivaluadas (DMV) independientes, definidas formalmente como donde Y y Z son independientes entre sí.

Análisis:

En nuestro modelo actual, no existen atributos multivaluados dentro de una misma relación que sean independientes.

* Ejemplo de violación (que no ocurre aquí): Si un libro tuviera múltiples autores y simultáneamente múltiples géneros no relacionados almacenados en la misma tabla, generando un producto cartesiano.
* En nuestro diseño, cada atributo tiene una relación directa y singular con su clave primaria o pero nunca incrustado.

**Conclusión:** Al no existir dependencias multivaluadas no triviales, el esquema cumple con la **4FN**.

**7. Resultado Final.**

|  |  |
| --- | --- |
| Dirección | Ciudad |

|  |  |  |
| --- | --- | --- |
| Correo Cliente | Cliente | Dirección |

|  |  |  |  |
| --- | --- | --- | --- |
| Titulo Libro | Autor Libro | Genero Libro | Precio Libro |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Pedido ID | Cantidad | Fecha Pedido | Monto Total | Estado Pago | Correo Cliente |

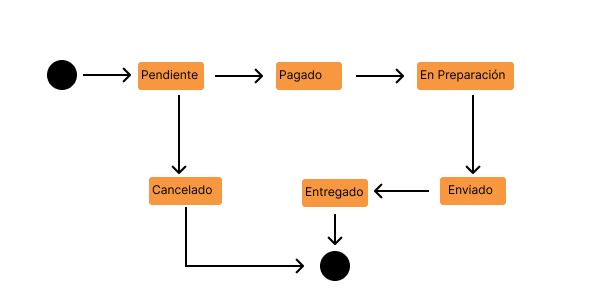
|  |  |  |
| --- | --- | --- |
| Pedido ID | Titulo Libro | Cantidad |

Este esquema final garantiza la integridad referencial y elimina las anomalías de inserción, actualización y borrado detectadas en el análisis inicial. Cumple estrictamente con las formas normales hasta 4FN y prepara el terreno para la implementación física en Oracle.

**Fase 4: Implementación Física en un SGBD**

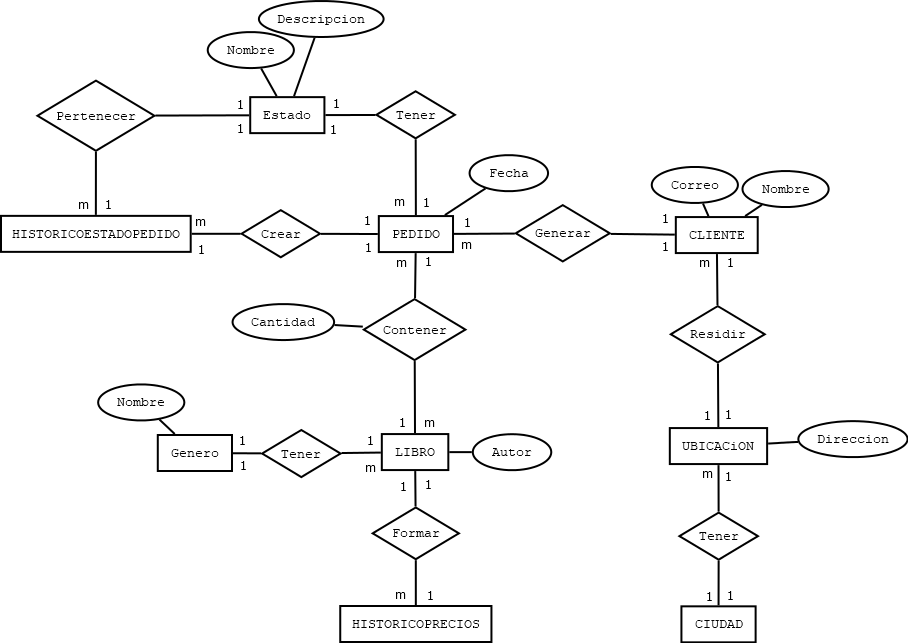
**Flujo de estados.**

Figura 4. Flujo de estados en las transacciones para el estad de un histórico de precios.



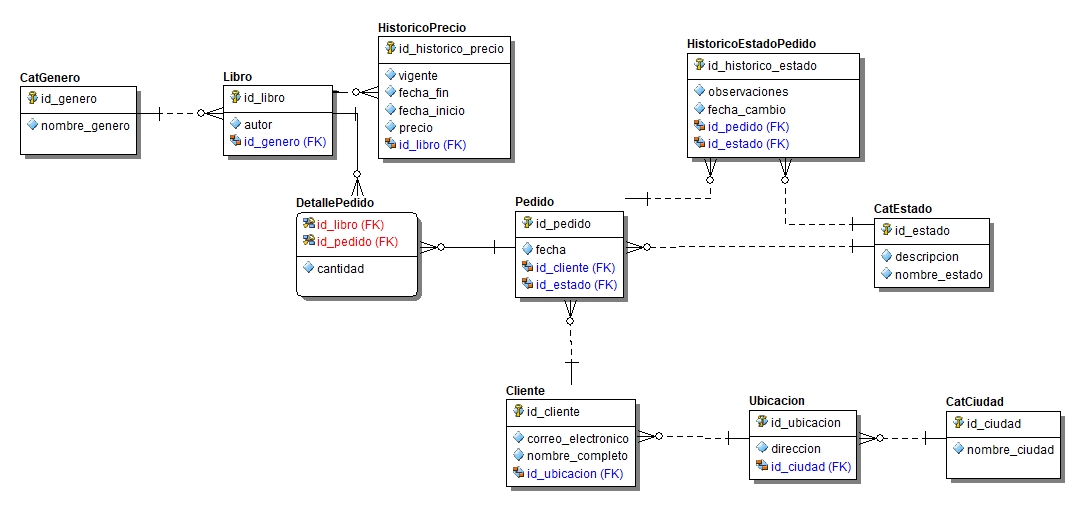
**Diagrama Entidad Relación.**

Figura 5. Modelo Entidad Relación luego de añadir catálogos e historial.



**Modelo lógico.**

Figura 6. Modelo lógico luego de añadir catálogos e historial.

**Creación de Tablas.**

*CreacionTablas.sql:*

-- A. CATÁLOGOS SIMPLES -----------------------

-- 1. Catálogo de Estados

**CREATE** **TABLE** CatEstado (

id\_estado NUMBER **GENERATED** **BY** **DEFAULT** **AS** **IDENTITY** **PRIMARY** **KEY**,

nombre\_estado VARCHAR2(**50**) **NOT** **NULL** **UNIQUE**,

descripcion VARCHAR2(**200**)

);

-- 2. Catálogo de Ciudades

**CREATE** **TABLE** CatCiudad (

id\_ciudad NUMBER **GENERATED** **BY** **DEFAULT** **AS** **IDENTITY** **PRIMARY** **KEY**,

nombre\_ciudad VARCHAR2(**100**) **NOT** **NULL** **UNIQUE**

);

-- 3. Catálogo de Géneros

**CREATE** **TABLE** CatGenero (

id\_genero NUMBER **GENERATED** **BY** **DEFAULT** **AS** **IDENTITY** **PRIMARY** **KEY**,

nombre\_genero VARCHAR2(**100**) **NOT** **NULL** **UNIQUE**

);

-- B. TABLAS PRINCIPALES

-- 4. Ubicación

**CREATE** **TABLE** Ubicacion (

id\_ubicacion NUMBER **GENERATED** **BY** **DEFAULT** **AS** **IDENTITY** **PRIMARY** **KEY**,

id\_ciudad NUMBER **NOT** **NULL**, -- FK

direccion VARCHAR2(**200**) **NOT** **NULL**,

**CONSTRAINT** fk\_ubicacion\_ciudad **FOREIGN** **KEY** (id\_ciudad)

**REFERENCES** CatCiudad(id\_ciudad)

);

-- 5. Cliente

**CREATE** **TABLE** Cliente (

id\_cliente NUMBER **GENERATED** **BY** **DEFAULT** **AS** **IDENTITY** **PRIMARY** **KEY**,

correo\_electronico VARCHAR2(**150**) **NOT** **NULL** **UNIQUE**,

nombre\_completo VARCHAR2(**200**) **NOT** **NULL**,

id\_ubicacion NUMBER **NOT** **NULL**,

**CONSTRAINT** fk\_cliente\_ubicacion **FOREIGN** **KEY** (id\_ubicacion)

**REFERENCES** Ubicacion(id\_ubicacion)

);

-- 6. Libro

**CREATE** **TABLE** Libro (

id\_libro NUMBER **GENERATED** **BY** **DEFAULT** **AS** **IDENTITY** **PRIMARY** **KEY**,

id\_genero NUMBER **NOT** **NULL**, -- FK

autor VARCHAR2(**100**),

titulo VARCHAR2(**100**),

**CONSTRAINT** fk\_libro\_genero **FOREIGN** **KEY** (id\_genero)

**REFERENCES** CatGenero(id\_genero)

);

-- 7. Histórico de Precios

**CREATE** **TABLE** HistoricoPrecio (

id\_historico\_precio NUMBER **GENERATED** **BY** **DEFAULT** **AS** **IDENTITY** **PRIMARY** **KEY**,

id\_libro NUMBER **NOT** **NULL**,

precio NUMBER(**12**,**2**) **NOT** **NULL** **CHECK** (precio > **0**),

fecha\_inicio DATE **NOT** **NULL**,

fecha\_fin DATE,

vigente CHAR(**1**) **DEFAULT** '1' **CHECK** (vigente **IN** ('0','1')),

**CONSTRAINT** fk\_hist\_precio\_libro **FOREIGN** **KEY** (id\_libro)

**REFERENCES** Libro(id\_libro),

**CONSTRAINT** chk\_fechas\_precio **CHECK** (fecha\_fin **IS** **NULL** **OR** fecha\_fin > fecha\_inicio)

);

-- 8. Pedido

**CREATE** **TABLE** Pedido (

id\_pedido NUMBER **GENERATED** **BY** **DEFAULT** **AS** **IDENTITY** **PRIMARY** **KEY**,

id\_cliente NUMBER **NOT** **NULL**,

fecha DATE **DEFAULT** SYSDATE **NOT** **NULL**,

id\_estado\_actual NUMBER **NOT** **NULL**,

fecha\_ult\_cambio DATE **DEFAULT** SYSDATE **NOT** **NULL**,

**CONSTRAINT** fk\_pedido\_cliente **FOREIGN** **KEY** (id\_cliente)

**REFERENCES** Cliente(id\_cliente),

**CONSTRAINT** fk\_pedido\_estado **FOREIGN** **KEY** (id\_estado\_actual)

**REFERENCES** CatEstado(id\_estado)

);

-- 9. Histórico de Estados

**CREATE** **TABLE** HistoricoEstadoPedido (

id\_historico\_estado NUMBER **GENERATED** **BY** **DEFAULT** **AS** **IDENTITY** **PRIMARY** **KEY**,

id\_pedido NUMBER **NOT** **NULL**,

id\_estado NUMBER **NOT** **NULL**,

fecha\_cambio DATE **DEFAULT** SYSDATE **NOT** **NULL**,

observaciones VARCHAR2(**500**),

**CONSTRAINT** fk\_hep\_pedido **FOREIGN** **KEY** (id\_pedido)

**REFERENCES** Pedido(id\_pedido),

**CONSTRAINT** fk\_hep\_estado **FOREIGN** **KEY** (id\_estado)

**REFERENCES** CatEstado(id\_estado)

);

-- 10. Detalle de Pedido

**CREATE** **TABLE** DetallePedido (

id\_detalle NUMBER **GENERATED** **BY** **DEFAULT** **AS** **IDENTITY** **PRIMARY** **KEY**,

id\_pedido NUMBER **NOT** **NULL**,

id\_libro NUMBER **NOT** **NULL**,

cantidad NUMBER(**6**) **NOT** **NULL** **CHECK** (cantidad > **0**),

precio\_unitario NUMBER(**12**,**2**) **NOT** **NULL** **CHECK** (precio\_unitario > **0**),

**CONSTRAINT** fk\_detalle\_pedido **FOREIGN** **KEY** (id\_pedido)

**REFERENCES** Pedido(id\_pedido) **ON** **DELETE** **CASCADE**,

**CONSTRAINT** fk\_detalle\_libro **FOREIGN** **KEY** (id\_libro)

**REFERENCES** Libro(id\_libro),

**CONSTRAINT** uq\_pedido\_libro **UNIQUE** (id\_pedido, id\_libro)

);

**Estimación de Espacio en Disco.**

Definiremos un Escenario de Negocio (Supuestos) para proyectar el crecimiento, ya que la base de datos está vacía actualmente.

Supuestos de Crecimiento (Escenario: Librería Mediana en Expansión):

* **Pedidos Diarios:** 50 pedidos al día.
* **Libros por Pedido:** Promedio de 3 libros (afecta a DetallePedido).
* **Cambios de Estado:** Promedio de 3 cambios por pedido (Creado -> Pagado -> Enviado/Entregado).
* **Días Operativos:** 365 días al año.
* **Criterio de Tamaño:**
* NUMBER: ~22 bytes.
* VARCHAR2(N): ~N bytes (Asumiremos llenado completo para estimación de "peor caso" o seguridad).
* DATE: ~7 bytes (Oracle estándar).

**Cálculo de Espacio en Tablas (Data Size)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tabla | Tamaño Aprox. Fila (Bytes) | Carga Inicial (Reg) | Crecimiento Diario (Reg) | Registros a 5 Años | Bytes Totales a 5 Años |
| CATÁLOGOS (Estado, Ciudad, Genero) | ~100 B (Promedio) | 100 | 0 (Estáticos) | 100 | 10,000 B |
| UBICACION | ID(22)+FK(22)+Dir(200) = **244 B** | 50 | 2 (Nuevas zonas) | 3,700 | 902,800 B |
| CLIENTE | ID(22)+Email(150)+Nom(200)+FK(22) = **394 B** | 50 | 2 (Nuevos clientes) | 3,700 | 1,457,800 B |
| LIBRO | ID(22)+FK(22)+Txt(200) = **244 B** | 1,000 | 1 (Nuevos libros) | 2,825 | 689,300 B |
| HIST. PRECIO | ID(22)+FK(22)+Pre(22)+Dates(14) = **80 B** | 1,000 | 1 (Cambios precio) | 2,825 | 226,000 B |
| PEDIDO (Transaccional) | ID(22)+FKs(44)+Dates(14) = **80 B** | 0 | 50 | 91,250 | 7,300,000 B |
| DETALLE PEDIDO (Transaccional) | ID(22)+FKs(44)+Nums(44) = **110 B** | 0 | 150 (3 items/pedido) | 273,750 | 30,112,500 B |
| HIST. ESTADO (Transaccional) | ID(22)+FKs(44)+Date(7)+Obs(500) = **573 B** | 0 | 150 (3 estados/pedido) | 273,750 | 156,858,750 B |
| TOTAL |  |  |  |  | **197,557,150 B** |

**Cálculo de Espacio en Índices.**

Asumimos un tamaño promedio de **15 bytes por entrada**. Consideramos:

* **PKs (Automáticas):** 1 por cada tabla.
* **FKs (Manuales):** Críticas para el rendimiento (Integridad referencial y Joins).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tabla | Índices Considerados | Registros a 5 Años | Entradas Totales de Índice | Bytes Totales (15 B/Reg) |
| CATÁLOGOS | 3 PKs | 100 | 300 | 4,500 B |
| UBICACION | 1 PK + 1 FK | 3,700 | 7,400 | 111,000 B |
| CLIENTE | 1 PK + 1 FK | 3,700 | 7,400 | 111,000 B |
| LIBRO | 1 PK + 1 FK | 2,825 | 5,650 | 84,750 B |
| HIST. PRECIO | 1 PK + 1 FK | 2,825 | 5,650 | 84,750 B |
| PEDIDO | 1 PK + 2 FKs (Cliente, Estado) | 91,250 | 273,750 | 4,106,250 B |
| DETALLE | 1 PK + 2 FKs (Pedido, Libro) | 273,750 | 821,250 | 12,318,750 B |
| HIST. ESTADO | 1 PK + 2 FKs (Pedido, Estado) | 273,750 | 821,250 | 12,318,750 B |
| TOTAL |  |  |  | **29,139,750 B** |

**Tabla Resumen y Proyección Final (Sizing).**

|  |  |  |
| --- | --- | --- |
| Componente de Espacio | Bytes a 5 Años | Megabytes (MB) a 5 Años |
| Espacio en Tablas | **197,557,150** | **188.40 MB** |
| Espacio en Índices | **29,139,750** | **27.79 MB** |
| SUBTOTAL REQUERIDO | **226,696,900** | **216.19 MB** |
| Espacio Adicional (20% Overhead) | **45,339,380** | **43.24 MB** |
| TOTAL PROYECTADO A 5 AÑOS | **272,036,280** | **~ 259.43 MB** |

**Fase 5: Creación de Consultas SQL y Procedimientos Almacenados.**

**Creación de Datos.**

*InsertarDatosOriginales.sql*

**SET** SERVEROUTPUT **ON**;

**DECLARE**

-- Variables para capturar IDs

v\_id\_ciudad NUMBER;

v\_id\_genero NUMBER;

v\_id\_estado\_p NUMBER; -- Pagado

v\_id\_estado\_pd NUMBER; -- Pendiente

v\_id\_cliente NUMBER;

v\_id\_ubicacion NUMBER;

v\_id\_pedido NUMBER;

-- Variables para IDs de Libros (Cache simple)

v\_libro\_sql\_bas NUMBER;

v\_libro\_mod\_dat NUMBER;

v\_libro\_python NUMBER;

v\_libro\_sql\_adv NUMBER;

v\_libro\_bigdata NUMBER;

-- Procedimiento Local: Gestionar Ciudad (Busca o Crea)

**PROCEDURE** get\_create\_ciudad(p\_nombre **IN** VARCHAR2, p\_id **OUT** NUMBER) **IS**

**BEGIN**

**SELECT** id\_ciudad **INTO** p\_id **FROM** CatCiudad **WHERE** nombre\_ciudad = p\_nombre;

**EXCEPTION**

**WHEN** NO\_DATA\_FOUND **THEN**

**INSERT** **INTO** CatCiudad (nombre\_ciudad) **VALUES** (p\_nombre) RETURNING id\_ciudad **INTO** p\_id;

**END**;

-- Procedimiento Local: Gestionar Libro y Precio (Busca o Crea)

**PROCEDURE** get\_create\_libro(

p\_titulo **IN** VARCHAR2, p\_autor **IN** VARCHAR2, p\_precio **IN** NUMBER, p\_id\_genero **IN** NUMBER, p\_id\_out **OUT** NUMBER

) **IS**

**BEGIN**

**SELECT** id\_libro **INTO** p\_id\_out **FROM** Libro **WHERE** titulo = p\_titulo;

**EXCEPTION**

**WHEN** NO\_DATA\_FOUND **THEN**

-- Insertar Libro

**INSERT** **INTO** Libro (id\_genero, autor, titulo)

**VALUES** (p\_id\_genero, p\_autor, p\_titulo)

RETURNING id\_libro **INTO** p\_id\_out;

-- Insertar Precio Inicial

**INSERT** **INTO** HistoricoPrecio (id\_libro, precio, fecha\_inicio, vigente)

**VALUES** (p\_id\_out, p\_precio, SYSDATE, '1');

**END**;

-- Procedimiento Local: Gestionar Cliente (Busca por correo o Crea con Ubicación)

**PROCEDURE** get\_create\_cliente(

p\_correo **IN** VARCHAR2, p\_nombre **IN** VARCHAR2, p\_direccion **IN** VARCHAR2, p\_id\_ciudad **IN** NUMBER, p\_id\_out **OUT** NUMBER

) **IS**

v\_id\_ubi\_local NUMBER;

**BEGIN**

**SELECT** id\_cliente **INTO** p\_id\_out **FROM** Cliente **WHERE** correo\_electronico = p\_correo;

**EXCEPTION**

**WHEN** NO\_DATA\_FOUND **THEN**

-- Crear Ubicación primero

**INSERT** **INTO** Ubicacion (id\_ciudad, direccion)

**VALUES** (p\_id\_ciudad, p\_direccion)

RETURNING id\_ubicacion **INTO** v\_id\_ubi\_local;

-- Crear Cliente

**INSERT** **INTO** Cliente (correo\_electronico, nombre\_completo, id\_ubicacion)

**VALUES** (p\_correo, p\_nombre, v\_id\_ubi\_local)

RETURNING id\_cliente **INTO** p\_id\_out;

**END**;

**BEGIN**

DBMS\_OUTPUT.PUT\_LINE('--- INICIANDO CARGA DE TABLA SOLICITADA ---');

-- 1. PREPARAR CATÁLOGOS BÁSICOS (Estados y Género)

-- Genero 'Técnico'

**BEGIN**

**SELECT** id\_genero **INTO** v\_id\_genero **FROM** CatGenero **WHERE** nombre\_genero = 'Técnico';

**EXCEPTION** **WHEN** NO\_DATA\_FOUND **THEN**

**INSERT** **INTO** CatGenero (nombre\_genero) **VALUES** ('Técnico') RETURNING id\_genero **INTO** v\_id\_genero;

**END**;

-- Estados 'Pagado' y 'Pendiente'

**BEGIN**

**SELECT** id\_estado **INTO** v\_id\_estado\_p **FROM** CatEstado **WHERE** nombre\_estado = 'Pagado';

**EXCEPTION** **WHEN** NO\_DATA\_FOUND **THEN**

**INSERT** **INTO** CatEstado (nombre\_estado, descripcion) **VALUES** ('Pagado', 'Pago confirmado') RETURNING id\_estado **INTO** v\_id\_estado\_p;

**END**;

**BEGIN**

**SELECT** id\_estado **INTO** v\_id\_estado\_pd **FROM** CatEstado **WHERE** nombre\_estado = 'Pendiente';

**EXCEPTION** **WHEN** NO\_DATA\_FOUND **THEN**

**INSERT** **INTO** CatEstado (nombre\_estado, descripcion) **VALUES** ('Pendiente', 'Esperando pago') RETURNING id\_estado **INTO** v\_id\_estado\_pd;

**END**;

-- 2. PREPARAR LIBROS (Insertamos los 5 libros únicos de la tabla para tener sus IDs)

get\_create\_libro('SQL Básico', 'A. Brown', **299**.**90**, v\_id\_genero, v\_libro\_sql\_bas);

get\_create\_libro('Modelado Datos', 'B. White', **399**.**90**, v\_id\_genero, v\_libro\_mod\_dat);

get\_create\_libro('Guía Python', 'C. Green', **499**.**90**, v\_id\_genero, v\_libro\_python);

get\_create\_libro('SQL Avanzado', 'D. Black', **349**.**90**, v\_id\_genero, v\_libro\_sql\_adv);

get\_create\_libro('Big Data', 'E. Blue', **599**.**90**, v\_id\_genero, v\_libro\_bigdata);

DBMS\_OUTPUT.PUT\_LINE('-> Catálogos y Libros preparados.');

-- 3. PROCESAMIENTO DE PEDIDOS (FILA POR FILA)

-- ==============================================================================

-- ID 101: Juan Pérez (CDMX) - 2 Libros - Pagado

-- ==============================================================================

get\_create\_ciudad('CDMX', v\_id\_ciudad);

get\_create\_cliente('juan@email.com', 'Juan Pérez', 'Calle Luna 123', v\_id\_ciudad, v\_id\_cliente);

**INSERT** **INTO** Pedido (id\_cliente, fecha, id\_estado\_actual)

**VALUES** (v\_id\_cliente, TO\_DATE('10/01/2025','DD/MM/YYYY'), v\_id\_estado\_p) RETURNING id\_pedido **INTO** v\_id\_pedido;

-- Histórico Estado

**INSERT** **INTO** HistoricoEstadoPedido (id\_pedido, id\_estado, fecha\_cambio, observaciones)

**VALUES** (v\_id\_pedido, v\_id\_estado\_p, TO\_DATE('10/01/2025','DD/MM/YYYY'), 'Pedido Creado Pagado');

-- Detalles (SQL Básico x2, Modelado x1)

**INSERT** **INTO** DetallePedido (id\_pedido, id\_libro, cantidad, precio\_unitario) **VALUES** (v\_id\_pedido, v\_libro\_sql\_bas, **2**, **299**.**90**);

**INSERT** **INTO** DetallePedido (id\_pedido, id\_libro, cantidad, precio\_unitario) **VALUES** (v\_id\_pedido, v\_libro\_mod\_dat, **1**, **399**.**90**);

-- ==============================================================================

-- ID 102: Ana López (Miami) - 1 Libro - Pendiente

-- ==============================================================================

get\_create\_ciudad('Miami', v\_id\_ciudad);

get\_create\_cliente('ana@email.com', 'Ana López', 'Av. Sol 456', v\_id\_ciudad, v\_id\_cliente);

**INSERT** **INTO** Pedido (id\_cliente, fecha, id\_estado\_actual)

**VALUES** (v\_id\_cliente, TO\_DATE('12/01/2025','DD/MM/YYYY'), v\_id\_estado\_pd) RETURNING id\_pedido **INTO** v\_id\_pedido;

**INSERT** **INTO** HistoricoEstadoPedido (id\_pedido, id\_estado, fecha\_cambio) **VALUES** (v\_id\_pedido, v\_id\_estado\_pd, TO\_DATE('12/01/2025','DD/MM/YYYY'));

-- Detalle (Guía Python x1)

**INSERT** **INTO** DetallePedido (id\_pedido, id\_libro, cantidad, precio\_unitario) **VALUES** (v\_id\_pedido, v\_libro\_python, **1**, **499**.**90**);

-- ==============================================================================

-- ID 103: Luis Gómez (Bogotá) - 2 Libros - Pagado

-- ==============================================================================

get\_create\_ciudad('Bogotá', v\_id\_ciudad);

get\_create\_cliente('luis@email.com', 'Luis Gómez', 'Calle Estrella 789', v\_id\_ciudad, v\_id\_cliente);

**INSERT** **INTO** Pedido (id\_cliente, fecha, id\_estado\_actual)

**VALUES** (v\_id\_cliente, TO\_DATE('15/01/2025','DD/MM/YYYY'), v\_id\_estado\_p) RETURNING id\_pedido **INTO** v\_id\_pedido;

**INSERT** **INTO** HistoricoEstadoPedido (id\_pedido, id\_estado, fecha\_cambio) **VALUES** (v\_id\_pedido, v\_id\_estado\_p, TO\_DATE('15/01/2025','DD/MM/YYYY'));

-- Detalles (SQL Avanzado x1, Big Data x2)

**INSERT** **INTO** DetallePedido (id\_pedido, id\_libro, cantidad, precio\_unitario) **VALUES** (v\_id\_pedido, v\_libro\_sql\_adv, **1**, **349**.**90**);

**INSERT** **INTO** DetallePedido (id\_pedido, id\_libro, cantidad, precio\_unitario) **VALUES** (v\_id\_pedido, v\_libro\_bigdata, **2**, **599**.**90**);

-- ==============================================================================

-- ID 104: María Ruiz (CDMX) - 1 Libro - Pagado

-- ==============================================================================

get\_create\_ciudad('CDMX', v\_id\_ciudad);

get\_create\_cliente('maria@email.com', 'María Ruiz', 'Av. Nube 101', v\_id\_ciudad, v\_id\_cliente);

**INSERT** **INTO** Pedido (id\_cliente, fecha, id\_estado\_actual)

**VALUES** (v\_id\_cliente, TO\_DATE('18/01/2025','DD/MM/YYYY'), v\_id\_estado\_p) RETURNING id\_pedido **INTO** v\_id\_pedido;

**INSERT** **INTO** HistoricoEstadoPedido (id\_pedido, id\_estado, fecha\_cambio) **VALUES** (v\_id\_pedido, v\_id\_estado\_p, TO\_DATE('18/01/2025','DD/MM/YYYY'));

-- Detalle (Modelado Datos x3)

**INSERT** **INTO** DetallePedido (id\_pedido, id\_libro, cantidad, precio\_unitario) **VALUES** (v\_id\_pedido, v\_libro\_mod\_dat, **3**, **399**.**90**);

-- ==============================================================================

-- ID 105: Carlos Díaz (Lima) - 2 Libros - Pendiente

-- ==============================================================================

get\_create\_ciudad('Lima', v\_id\_ciudad);

get\_create\_cliente('carlos@email.com', 'Carlos Díaz', 'Calle Sol 202', v\_id\_ciudad, v\_id\_cliente);

**INSERT** **INTO** Pedido (id\_cliente, fecha, id\_estado\_actual)

**VALUES** (v\_id\_cliente, TO\_DATE('20/01/2025','DD/MM/YYYY'), v\_id\_estado\_pd) RETURNING id\_pedido **INTO** v\_id\_pedido;

**INSERT** **INTO** HistoricoEstadoPedido (id\_pedido, id\_estado, fecha\_cambio) **VALUES** (v\_id\_pedido, v\_id\_estado\_pd, TO\_DATE('20/01/2025','DD/MM/YYYY'));

-- Detalles (Python x2, Big Data x1)

**INSERT** **INTO** DetallePedido (id\_pedido, id\_libro, cantidad, precio\_unitario) **VALUES** (v\_id\_pedido, v\_libro\_python, **2**, **499**.**90**);

**INSERT** **INTO** DetallePedido (id\_pedido, id\_libro, cantidad, precio\_unitario) **VALUES** (v\_id\_pedido, v\_libro\_bigdata, **1**, **599**.**90**);

-- ==============================================================================

-- ID 106: Laura Vega (Miami) - 1 Libro - Pagado

-- ==============================================================================

get\_create\_ciudad('Miami', v\_id\_ciudad);

get\_create\_cliente('laura@email.com', 'Laura Vega', 'Av. Luna 303', v\_id\_ciudad, v\_id\_cliente);

**INSERT** **INTO** Pedido (id\_cliente, fecha, id\_estado\_actual)

**VALUES** (v\_id\_cliente, TO\_DATE('22/01/2025','DD/MM/YYYY'), v\_id\_estado\_p) RETURNING id\_pedido **INTO** v\_id\_pedido;

**INSERT** **INTO** HistoricoEstadoPedido (id\_pedido, id\_estado, fecha\_cambio) **VALUES** (v\_id\_pedido, v\_id\_estado\_p, TO\_DATE('22/01/2025','DD/MM/YYYY'));

-- Detalle (SQL Básico x1)

**INSERT** **INTO** DetallePedido (id\_pedido, id\_libro, cantidad, precio\_unitario) **VALUES** (v\_id\_pedido, v\_libro\_sql\_bas, **1**, **299**.**90**);

-- ==============================================================================

-- ID 107: Pedro Mora (Bogotá) - 1 Libro - Pagado

-- ==============================================================================

get\_create\_ciudad('Bogotá', v\_id\_ciudad);

get\_create\_cliente('pedro@email.com', 'Pedro Mora', 'Calle Cielo 404', v\_id\_ciudad, v\_id\_cliente);

**INSERT** **INTO** Pedido (id\_cliente, fecha, id\_estado\_actual)

**VALUES** (v\_id\_cliente, TO\_DATE('25/01/2025','DD/MM/YYYY'), v\_id\_estado\_p) RETURNING id\_pedido **INTO** v\_id\_pedido;

**INSERT** **INTO** HistoricoEstadoPedido (id\_pedido, id\_estado, fecha\_cambio) **VALUES** (v\_id\_pedido, v\_id\_estado\_p, TO\_DATE('25/01/2025','DD/MM/YYYY'));

-- Detalle (Big Data x2)

**INSERT** **INTO** DetallePedido (id\_pedido, id\_libro, cantidad, precio\_unitario) **VALUES** (v\_id\_pedido, v\_libro\_bigdata, **2**, **599**.**90**);

-- ==============================================================================

-- ID 108: Sofía Castro (CDMX) - 2 Libros - Pendiente

-- ==============================================================================

get\_create\_ciudad('CDMX', v\_id\_ciudad);

get\_create\_cliente('sofia@email.com', 'Sofía Castro', 'Av. Sol 505', v\_id\_ciudad, v\_id\_cliente);

**INSERT** **INTO** Pedido (id\_cliente, fecha, id\_estado\_actual)

**VALUES** (v\_id\_cliente, TO\_DATE('28/01/2025','DD/MM/YYYY'), v\_id\_estado\_pd) RETURNING id\_pedido **INTO** v\_id\_pedido;

**INSERT** **INTO** HistoricoEstadoPedido (id\_pedido, id\_estado, fecha\_cambio) **VALUES** (v\_id\_pedido, v\_id\_estado\_pd, TO\_DATE('28/01/2025','DD/MM/YYYY'));

-- Detalles (SQL Avanzado x1, Python x1)

**INSERT** **INTO** DetallePedido (id\_pedido, id\_libro, cantidad, precio\_unitario) **VALUES** (v\_id\_pedido, v\_libro\_sql\_adv, **1**, **349**.**90**);

**INSERT** **INTO** DetallePedido (id\_pedido, id\_libro, cantidad, precio\_unitario) **VALUES** (v\_id\_pedido, v\_libro\_python, **1**, **499**.**90**);

-- ==============================================================================

-- ID 109: Diego León (Lima) - 1 Libro - Pagado

-- ==============================================================================

get\_create\_ciudad('Lima', v\_id\_ciudad);

get\_create\_cliente('diego@email.com', 'Diego León', 'Calle Luna 606', v\_id\_ciudad, v\_id\_cliente);

**INSERT** **INTO** Pedido (id\_cliente, fecha, id\_estado\_actual)

**VALUES** (v\_id\_cliente, TO\_DATE('30/01/2025','DD/MM/YYYY'), v\_id\_estado\_p) RETURNING id\_pedido **INTO** v\_id\_pedido;

**INSERT** **INTO** HistoricoEstadoPedido (id\_pedido, id\_estado, fecha\_cambio) **VALUES** (v\_id\_pedido, v\_id\_estado\_p, TO\_DATE('30/01/2025','DD/MM/YYYY'));

-- Detalle (Modelado Datos x2)

**INSERT** **INTO** DetallePedido (id\_pedido, id\_libro, cantidad, precio\_unitario) **VALUES** (v\_id\_pedido, v\_libro\_mod\_dat, **2**, **399**.**90**);

-- ==============================================================================

-- ID 110: Elena Cruz (Miami) - 2 Libros - Pagado

-- ==============================================================================

get\_create\_ciudad('Miami', v\_id\_ciudad);

get\_create\_cliente('elena@email.com', 'Elena Cruz', 'Av. Estrella 707', v\_id\_ciudad, v\_id\_cliente);

**INSERT** **INTO** Pedido (id\_cliente, fecha, id\_estado\_actual)

**VALUES** (v\_id\_cliente, TO\_DATE('01/02/2025','DD/MM/YYYY'), v\_id\_estado\_p) RETURNING id\_pedido **INTO** v\_id\_pedido;

**INSERT** **INTO** HistoricoEstadoPedido (id\_pedido, id\_estado, fecha\_cambio) **VALUES** (v\_id\_pedido, v\_id\_estado\_p, TO\_DATE('01/02/2025','DD/MM/YYYY'));

-- Detalles (SQL Básico x1, Big Data x1)

**INSERT** **INTO** DetallePedido (id\_pedido, id\_libro, cantidad, precio\_unitario) **VALUES** (v\_id\_pedido, v\_libro\_sql\_bas, **1**, **299**.**90**);

**INSERT** **INTO** DetallePedido (id\_pedido, id\_libro, cantidad, precio\_unitario) **VALUES** (v\_id\_pedido, v\_libro\_bigdata, **1**, **599**.**90**);

DBMS\_OUTPUT.PUT\_LINE('-> 10 Pedidos (ID 101 a 110) insertados correctamente con sus detalles.');

**COMMIT**;

DBMS\_OUTPUT.PUT\_LINE('--- CARGA DE TABLA COMPLETADA EXITOSAMENTE ---');

**EXCEPTION**

**WHEN** OTHERS **THEN**

**ROLLBACK**;

DBMS\_OUTPUT.PUT\_LINE('ERROR FATAL EN LA CARGA: ' || SQLERRM);

**END**;

/

**Creación de Índices.**

-- ÍNDICES DE RENDIMIENTO

**CREATE** **INDEX** idx\_cliente\_correo **ON** Cliente(correo\_electronico);

**CREATE** **INDEX** idx\_pedido\_fecha **ON** Pedido(fecha);

**CREATE** **INDEX** idx\_pedido\_estado **ON** Pedido(id\_estado\_actual);

**CREATE** **INDEX** idx\_detalle\_pedido **ON** DetallePedido(id\_pedido);

**CREATE** **INDEX** idx\_hist\_precio\_libro **ON** HistoricoPrecio(id\_libro);

**CREATE** **INDEX** idx\_hist\_precio\_vig **ON** HistoricoPrecio(id\_libro, vigente, fecha\_inicio);

**CREATE** **INDEX** idx\_hist\_estado\_pedido **ON** HistoricoEstadoPedido(id\_pedido);

**CREATE** **INDEX** idx\_hist\_estado\_fecha **ON** HistoricoEstadoPedido(id\_pedido, fecha\_cambio);

**Procedimientos Almacenados.**

-- A. Procedimiento insertar Ubicación (Recibe ID Ciudad)

**CREATE** **OR** **REPLACE** **PROCEDURE** sp\_insertar\_ubicacion (

p\_id\_ciudad **IN** NUMBER,

p\_direccion **IN** VARCHAR2,

p\_id\_ubicacion **OUT** NUMBER

) **AS**

**BEGIN**

**INSERT** **INTO** Ubicacion (id\_ciudad, direccion)

**VALUES** (p\_id\_ciudad, p\_direccion)

RETURNING id\_ubicacion **INTO** p\_id\_ubicacion;

**COMMIT**;

**EXCEPTION** **WHEN** OTHERS **THEN** **ROLLBACK**; RAISE\_APPLICATION\_ERROR(-**20001**, SQLERRM);

**END**;

/

-- B. Procedimiento insertar Cliente

**CREATE** **OR** **REPLACE** **PROCEDURE** sp\_insertar\_cliente (

p\_correo\_electronico **IN** VARCHAR2,

p\_nombre\_completo **IN** VARCHAR2,

p\_id\_ubicacion **IN** NUMBER,

p\_id\_cliente **OUT** NUMBER

) **AS**

**BEGIN**

**INSERT** **INTO** Cliente (correo\_electronico, nombre\_completo, id\_ubicacion)

**VALUES** (p\_correo\_electronico, p\_nombre\_completo, p\_id\_ubicacion)

RETURNING id\_cliente **INTO** p\_id\_cliente;

**COMMIT**;

**EXCEPTION** **WHEN** OTHERS **THEN** **ROLLBACK**; RAISE\_APPLICATION\_ERROR(-**20002**, SQLERRM);

**END**;

/

-- C. Procedimiento insertar Libro (Recibe ID Genero)

**CREATE** **OR** **REPLACE** **PROCEDURE** sp\_insertar\_libro (

p\_id\_genero **IN** NUMBER,

p\_autor **IN** VARCHAR2,

p\_titulo **IN** VARCHAR2, -- Nuevo parámetro

p\_precio **IN** NUMBER,

p\_id\_libro **OUT** NUMBER

) **AS**

**BEGIN**

-- Insertar libro incluyendo el Título

**INSERT** **INTO** Libro (id\_genero, autor, titulo)

**VALUES** (p\_id\_genero, p\_autor, p\_titulo)

RETURNING id\_libro **INTO** p\_id\_libro;

-- Insertar precio inicial

**INSERT** **INTO** HistoricoPrecio (id\_libro, precio, fecha\_inicio, fecha\_fin, vigente)

**VALUES** (p\_id\_libro, p\_precio, SYSDATE, **NULL**, '1');

**COMMIT**;

**EXCEPTION** **WHEN** OTHERS **THEN** **ROLLBACK**; RAISE\_APPLICATION\_ERROR(-**20003**, SQLERRM);

**END**;

/

-- D. Procedimiento Actualizar Precio

**CREATE** **OR** **REPLACE** **PROCEDURE** sp\_actualizar\_precio\_libro (

p\_id\_libro **IN** NUMBER,

p\_nuevo\_precio **IN** NUMBER

) **AS**

**BEGIN**

**UPDATE** HistoricoPrecio **SET** fecha\_fin = SYSDATE, vigente = '0'

**WHERE** id\_libro = p\_id\_libro **AND** vigente = '1';

**INSERT** **INTO** HistoricoPrecio (id\_libro, precio, fecha\_inicio, fecha\_fin, vigente)

**VALUES** (p\_id\_libro, p\_nuevo\_precio, SYSDATE, **NULL**, '1');

**COMMIT**;

**EXCEPTION** **WHEN** OTHERS **THEN** **ROLLBACK**; RAISE\_APPLICATION\_ERROR(-**20004**, SQLERRM);

**END**;

/

-- E. Procedimiento Crear Pedido

**CREATE** **OR** **REPLACE** **PROCEDURE** sp\_crear\_pedido (

p\_id\_cliente **IN** NUMBER,

p\_id\_pedido **OUT** NUMBER

) **AS**

v\_id\_estado\_pendiente NUMBER;

**BEGIN**

**SELECT** id\_estado **INTO** v\_id\_estado\_pendiente **FROM** CatEstado **WHERE** nombre\_estado = 'Pendiente';

**INSERT** **INTO** Pedido (id\_cliente, fecha, id\_estado\_actual, fecha\_ult\_cambio)

**VALUES** (p\_id\_cliente, SYSDATE, v\_id\_estado\_pendiente, SYSDATE)

RETURNING id\_pedido **INTO** p\_id\_pedido;

**COMMIT**;

**EXCEPTION** **WHEN** OTHERS **THEN** **ROLLBACK**; RAISE\_APPLICATION\_ERROR(-**20005**, SQLERRM);

**END**;

/

**CREATE** **OR** **REPLACE** **PROCEDURE** sp\_cambiar\_estado\_pedido (

p\_id\_pedido **IN** NUMBER,

p\_nombre\_estado **IN** VARCHAR2,

p\_observaciones **IN** VARCHAR2

) **AS**

v\_id\_nuevo\_estado NUMBER;

**BEGIN**

-- Obtener ID del estado basado en el nombre

**SELECT** id\_estado **INTO** v\_id\_nuevo\_estado

**FROM** CatEstado

**WHERE** nombre\_estado = p\_nombre\_estado;

-- Actualizar el pedido (El trigger se encargará de guardar el histórico)

**UPDATE** Pedido

**SET** id\_estado\_actual = v\_id\_nuevo\_estado,

fecha\_ult\_cambio = SYSDATE

**WHERE** id\_pedido = p\_id\_pedido;

**COMMIT**;

**EXCEPTION**

**WHEN** NO\_DATA\_FOUND **THEN**

RAISE\_APPLICATION\_ERROR(-**20015**, 'Estado no encontrado: ' || p\_nombre\_estado);

**WHEN** OTHERS **THEN**

**ROLLBACK**;

RAISE\_APPLICATION\_ERROR(-**20016**, SQLERRM);

**END**;

/

-- F. Procedimiento Detalle Pedido

**CREATE** **OR** **REPLACE** **PROCEDURE** sp\_agregar\_detalle\_pedido (

p\_id\_pedido **IN** NUMBER,

p\_id\_libro **IN** NUMBER,

p\_cantidad **IN** NUMBER

) **AS**

v\_precio\_actual NUMBER(**12**,**2**);

**BEGIN**

**SELECT** precio **INTO** v\_precio\_actual **FROM** HistoricoPrecio **WHERE** id\_libro = p\_id\_libro **AND** vigente = '1';

**INSERT** **INTO** DetallePedido (id\_pedido, id\_libro, cantidad, precio\_unitario)

**VALUES** (p\_id\_pedido, p\_id\_libro, p\_cantidad, v\_precio\_actual);

**COMMIT**;

**EXCEPTION** **WHEN** OTHERS **THEN** **ROLLBACK**; RAISE\_APPLICATION\_ERROR(-**20006**, SQLERRM);

**END**;

/

-- 2. PROCEDIMIENTO PARA BORRAR CLIENTE

-- Nota: Solo borrará si el cliente no tiene pedidos (por integridad referencial)

**CREATE** **OR** **REPLACE** **PROCEDURE** sp\_eliminar\_cliente (

p\_id\_cliente **IN** NUMBER

) **AS**

**BEGIN**

**DELETE** **FROM** Cliente **WHERE** id\_cliente = p\_id\_cliente;

**IF** **SQL**%ROWCOUNT = **0** **THEN**

RAISE\_APPLICATION\_ERROR(-**20010**, 'Cliente no encontrado');

**END** **IF**;

**COMMIT**;

**EXCEPTION**

**WHEN** OTHERS **THEN**

**ROLLBACK**;

-- Capturar error de llave foránea (Integridad)

**IF** **SQLCODE** = -**2292** **THEN**

RAISE\_APPLICATION\_ERROR(-**20011**, 'No se puede borrar: El cliente tiene pedidos asociados.');

**ELSE**

RAISE\_APPLICATION\_ERROR(-**20012**, 'Error al eliminar cliente: ' || SQLERRM);

**END** **IF**;

**END**;

/

**CREATE** **OR** **REPLACE** **PROCEDURE** sp\_obtener\_info\_pedido (

p\_id\_pedido **IN** NUMBER,

p\_cursor **OUT** SYS\_REFCURSOR

) **AS**

**BEGIN**

**OPEN** p\_cursor **FOR**

**SELECT**

-- 1. Información del Pedido y Estado

p.id\_pedido,

p.fecha **AS** fecha\_compra,

ce.nombre\_estado **AS** estado\_pedido,

-- 2. Información del Cliente y Ubicación

**c**.nombre\_completo **AS** cliente,

**c**.correo\_electronico,

u.direccion **AS** direccion\_entrega,

cc.nombre\_ciudad **AS** ciudad,

-- 3. Detalle del Libro (Producto) - CORREGIDO

l.id\_libro,

l.titulo **AS** libro\_titulo, -- Agregado

l.autor **AS** libro\_autor,

cg.nombre\_genero **AS** genero,

-- 4. Finanzas (Cálculos)

dp.cantidad,

dp.precio\_unitario,

(dp.cantidad \* dp.precio\_unitario) **AS** subtotal\_item,

-- 5. Monto Total del Pedido

**SUM**(dp.cantidad \* dp.precio\_unitario) OVER (PARTITION **BY** p.id\_pedido) **AS** monto\_total\_pedido

**FROM** Pedido p

**JOIN** Cliente **c** **ON** p.id\_cliente = **c**.id\_cliente

**JOIN** Ubicacion u **ON** **c**.id\_ubicacion = u.id\_ubicacion

**JOIN** CatCiudad cc **ON** u.id\_ciudad = cc.id\_ciudad

**JOIN** CatEstado ce **ON** p.id\_estado\_actual = ce.id\_estado

**JOIN** DetallePedido dp **ON** p.id\_pedido = dp.id\_pedido

**JOIN** Libro l **ON** dp.id\_libro = l.id\_libro

**JOIN** CatGenero cg **ON** l.id\_genero = cg.id\_genero

**WHERE** p.id\_pedido = p\_id\_pedido;

**EXCEPTION**

**WHEN** OTHERS **THEN**

**IF** p\_cursor%ISOPEN **THEN** **CLOSE** p\_cursor; **END** **IF**;

RAISE\_APPLICATION\_ERROR(-**20020**, 'Error al obtener info del pedido: ' || SQLERRM);

**END**;

/

**Funciones.**

-- G. FUNCIONES

-- Funcion 1: Total gastado por cliente

**CREATE** **OR** **REPLACE** **FUNCTION** fn\_total\_gastado\_cliente (

p\_id\_cliente **IN** NUMBER

) **RETURN** NUMBER **IS**

v\_total NUMBER(**12**,**2**);

**BEGIN**

**SELECT** NVL(**SUM**(dp.cantidad \* dp.precio\_unitario), **0**)

**INTO** v\_total

**FROM** Pedido p

**JOIN** DetallePedido dp **ON** p.id\_pedido = dp.id\_pedido

**JOIN** CatEstado ce **ON** p.id\_estado\_actual = ce.id\_estado

**WHERE** p.id\_cliente = p\_id\_cliente

**AND** ce.nombre\_estado **IN** ('Pagado', 'Enviado', 'Entregado'); -- Solo ventas reales

**RETURN** v\_total;

**EXCEPTION**

**WHEN** OTHERS **THEN** **RETURN** **0**;

**END**;

/

-- Función 2: Obtener Precio Actual de un Libro (Para evitar subconsultas repetitivas)

**CREATE** **OR** **REPLACE** **FUNCTION** fn\_obtener\_precio\_actual (

p\_id\_libro **IN** NUMBER

) **RETURN** NUMBER **IS**

v\_precio NUMBER(**12**,**2**);

**BEGIN**

**SELECT** precio **INTO** v\_precio

**FROM** HistoricoPrecio

**WHERE** id\_libro = p\_id\_libro **AND** vigente = '1';

**RETURN** v\_precio;

**EXCEPTION**

**WHEN** NO\_DATA\_FOUND **THEN** **RETURN** **0**;

**END**;

/

-- Función 3: Calcular Ventas Totales por Género (Analítica)

**CREATE** **OR** **REPLACE** **FUNCTION** fn\_ventas\_por\_genero (

p\_nombre\_genero **IN** VARCHAR2

) **RETURN** NUMBER **IS**

v\_total NUMBER(**12**,**2**);

**BEGIN**

**SELECT** NVL(**SUM**(dp.cantidad \* dp.precio\_unitario), **0**)

**INTO** v\_total

**FROM** DetallePedido dp

**JOIN** Libro l **ON** dp.id\_libro = l.id\_libro

**JOIN** CatGenero cg **ON** l.id\_genero = cg.id\_genero

**WHERE** cg.nombre\_genero = p\_nombre\_genero;

**RETURN** v\_total;

**END**;

/

-- Función 4: Verificar Stock/Disponibilidad (Simulación de regla de negocio)

-- Retorna 1 si se puede vender (ej. menos de 50 unidades vendidas hoy), 0 si no.

**CREATE** **OR** **REPLACE** **FUNCTION** fn\_verificar\_limite\_diario (

p\_id\_libro **IN** NUMBER

) **RETURN** NUMBER **IS**

v\_cantidad\_hoy NUMBER;

**BEGIN**

**SELECT** NVL(**SUM**(dp.cantidad), **0**) **INTO** v\_cantidad\_hoy

**FROM** DetallePedido dp

**JOIN** Pedido p **ON** dp.id\_pedido = p.id\_pedido

**WHERE** dp.id\_libro = p\_id\_libro

**AND** TRUNC(p.fecha) = TRUNC(SYSDATE);

**IF** v\_cantidad\_hoy >= **50** **THEN**

**RETURN** **0**; -- Límite diario alcanzado

**ELSE**

**RETURN** **1**; -- Disponible

**END** **IF**;

**END**;

/

**Triggers.**

-- 4. TRIGGERS (Lógica de negocio y logs)

**CREATE** **OR** **REPLACE** **TRIGGER** trg\_pedido\_insert\_historico

**AFTER** **INSERT** **ON** Pedido **FOR** **EACH** **ROW**

**BEGIN**

**INSERT** **INTO** HistoricoEstadoPedido (id\_pedido, id\_estado, fecha\_cambio, observaciones)

**VALUES** (:**NEW**.id\_pedido, :**NEW**.id\_estado\_actual, :**NEW**.fecha\_ult\_cambio, 'Estado inicial');

**END**;

/

**CREATE** **OR** **REPLACE** **TRIGGER** trg\_pedido\_update\_historico

**AFTER** **UPDATE** **OF** id\_estado\_actual **ON** Pedido **FOR** **EACH** **ROW**

**WHEN** (**NEW**.id\_estado\_actual != **OLD**.id\_estado\_actual)

**BEGIN**

**INSERT** **INTO** HistoricoEstadoPedido (id\_pedido, id\_estado, fecha\_cambio)

**VALUES** (:**NEW**.id\_pedido, :**NEW**.id\_estado\_actual, :**NEW**.fecha\_ult\_cambio);

**END**;

/

**CREATE** **OR** **REPLACE** **TRIGGER** trg\_validar\_estado\_pedido

**BEFORE** **UPDATE** **ON** Pedido **FOR** **EACH** **ROW**

**DECLARE** v\_nombre\_estado VARCHAR2(**50**);

**BEGIN**

**SELECT** nombre\_estado **INTO** v\_nombre\_estado **FROM** CatEstado **WHERE** id\_estado = :**OLD**.id\_estado\_actual;

**IF** v\_nombre\_estado **IN** ('Entregado', 'Cancelado') **THEN**

RAISE\_APPLICATION\_ERROR(-**20008**, 'Pedido finalizado, no se puede modificar.');

**END** **IF**;

**END**;

/

**CREATE** **OR** **REPLACE** **TRIGGER** trg\_validar\_detalle\_pedido

**BEFORE** **INSERT** **ON** DetallePedido **FOR** **EACH** **ROW**

**DECLARE** v\_nombre\_estado VARCHAR2(**50**);

**BEGIN**

**SELECT** ce.nombre\_estado **INTO** v\_nombre\_estado **FROM** Pedido p

**JOIN** CatEstado ce **ON** p.id\_estado\_actual = ce.id\_estado **WHERE** p.id\_pedido = :**NEW**.id\_pedido;

**IF** v\_nombre\_estado != 'Pendiente' **THEN**

RAISE\_APPLICATION\_ERROR(-**20009**, 'Solo se agregan items a pedidos Pendientes.');

**END** **IF**;

**END**;

/

**Vistas.**

-- VISTA HISTORIAL DE PRECIOS (Necesaria para el reporte)

**CREATE** **OR** **REPLACE** **VIEW** v\_historial\_precios **AS**

**SELECT**

l.id\_libro,

l.titulo **AS** titulo\_libro, -- Corregido: Usa la columna titulo real

l.autor, -- Agregado: Es útil ver también el autor

cg.nombre\_genero,

hp.precio,

hp.fecha\_inicio,

hp.fecha\_fin,

**CASE** **WHEN** hp.vigente = '1' **THEN** 'VIGENTE' **ELSE** 'HISTÓRICO' **END** **AS** estado\_precio

**FROM** Libro l

**JOIN** CatGenero cg **ON** l.id\_genero = cg.id\_genero

**JOIN** HistoricoPrecio hp **ON** l.id\_libro = hp.id\_libro

**ORDER** **BY** l.id\_libro, hp.fecha\_inicio **DESC**;

-- VISTA MAESTRA (Simulación de "Hoja de Cálculo" o Desnormalizada)

-- Esta vista reconstruye toda la información para reportes, equivalente a lo que tenías en SinFN.

**CREATE** **OR** **REPLACE** **VIEW** v\_reporte\_general\_ventas **AS**

**SELECT**

p.id\_pedido,

**c**.nombre\_completo **AS** cliente,

**c**.correo\_electronico,

u.direccion || ', ' || cc.nombre\_ciudad **AS** ubicacion\_entrega,

l.titulo **AS** libro,

l.autor,

cg.nombre\_genero,

dp.cantidad,

dp.precio\_unitario,

(dp.cantidad \* dp.precio\_unitario) **AS** subtotal,

p.fecha **AS** fecha\_pedido,

ce.nombre\_estado **AS** estado

**FROM** Pedido p

**JOIN** Cliente **c** **ON** p.id\_cliente = **c**.id\_cliente

**JOIN** Ubicacion u **ON** **c**.id\_ubicacion = u.id\_ubicacion

**JOIN** CatCiudad cc **ON** u.id\_ciudad = cc.id\_ciudad

**JOIN** DetallePedido dp **ON** p.id\_pedido = dp.id\_pedido

**JOIN** Libro l **ON** dp.id\_libro = l.id\_libro

**JOIN** CatGenero cg **ON** l.id\_genero = cg.id\_genero

**JOIN** CatEstado ce **ON** p.id\_estado\_actual = ce.id\_estado;

-- VISTA RESUMEN (Simulación 2FN - Cabeceras y Totales)

-- Muestra la información agrupada por pedido, útil para contabilidad.

**CREATE** **OR** **REPLACE** **VIEW** v\_resumen\_facturacion **AS**

**SELECT**

p.id\_pedido,

**c**.nombre\_completo,

p.fecha,

**COUNT**(dp.id\_libro) **AS** items\_unicos,

**SUM**(dp.cantidad) **AS** total\_libros,

**SUM**(dp.cantidad \* dp.precio\_unitario) **AS** monto\_total\_pedido,

ce.nombre\_estado

**FROM** Pedido p

**JOIN** Cliente **c** **ON** p.id\_cliente = **c**.id\_cliente

**JOIN** DetallePedido dp **ON** p.id\_pedido = dp.id\_pedido

**JOIN** CatEstado ce **ON** p.id\_estado\_actual = ce.id\_estado

**GROUP** **BY** p.id\_pedido, **c**.nombre\_completo, p.fecha, ce.nombre\_estado;

-- VISTA ANALÍTICA (KPIs de Negocio)

**CREATE** **OR** **REPLACE** **VIEW** v\_kpi\_rendimiento **AS**

**SELECT**

TO\_CHAR(p.fecha, 'YYYY-MM') **AS** mes,

cg.nombre\_genero,

**COUNT**(**DISTINCT** p.id\_pedido) **AS** total\_pedidos,

**SUM**(dp.cantidad \* dp.precio\_unitario) **AS** ingresos\_totales

**FROM** Pedido p

**JOIN** DetallePedido dp **ON** p.id\_pedido = dp.id\_pedido

**JOIN** Libro l **ON** dp.id\_libro = l.id\_libro

**JOIN** CatGenero cg **ON** l.id\_genero = cg.id\_genero

**GROUP** **BY** TO\_CHAR(p.fecha, 'YYYY-MM'), cg.nombre\_genero

**ORDER** **BY** mes **DESC**, ingresos\_totales **DESC**;

**Seguridad.**

-- 1. Crear ROL ADMINISTRADOR (DBA del Proyecto)

-- Tiene control total sobre las tablas y datos.

**CREATE** **ROLE** ROL\_ADMIN\_LIBRERIA;

**GRANT** **ALL** **PRIVILEGES** **ON** Cliente **TO** ROL\_ADMIN\_LIBRERIA;

**GRANT** **ALL** **PRIVILEGES** **ON** Pedido **TO** ROL\_ADMIN\_LIBRERIA;

**GRANT** **ALL** **PRIVILEGES** **ON** DetallePedido **TO** ROL\_ADMIN\_LIBRERIA;

**GRANT** **ALL** **PRIVILEGES** **ON** Libro **TO** ROL\_ADMIN\_LIBRERIA;

-- ... dar permisos al resto de tablas

-- 2. Crear ROL VENDEDOR (Operativo)

-- Puede registrar clientes, crear pedidos y consultar inventario.

-- NO puede borrar historiales ni alterar configuraciones de sistema.

**CREATE** **ROLE** ROL\_VENDEDOR;

**GRANT** **SELECT**, **INSERT**, **UPDATE** **ON** Cliente **TO** ROL\_VENDEDOR;

**GRANT** **SELECT**, **INSERT** **ON** Ubicacion **TO** ROL\_VENDEDOR;

**GRANT** **SELECT**, **INSERT**, **UPDATE** **ON** Pedido **TO** ROL\_VENDEDOR;

**GRANT** **SELECT**, **INSERT** **ON** DetallePedido **TO** ROL\_VENDEDOR;

**GRANT** **SELECT** **ON** v\_catalogo\_libros **TO** ROL\_VENDEDOR;

**GRANT** **EXECUTE** **ON** sp\_crear\_pedido **TO** ROL\_VENDEDOR;

**GRANT** **EXECUTE** **ON** sp\_agregar\_detalle\_pedido **TO** ROL\_VENDEDOR;

-- 3. Crear ROL AUDITOR (Solo Lectura)

-- Solo puede ver datos y reportes, no puede modificar nada.

**CREATE** **ROLE** ROL\_AUDITOR;

**GRANT** **SELECT** **ON** v\_reporte\_general\_ventas **TO** ROL\_AUDITOR;

**GRANT** **SELECT** **ON** v\_historial\_precios **TO** ROL\_AUDITOR;

**GRANT** **SELECT** **ON** HistoricoEstadoPedido **TO** ROL\_AUDITOR;

-- EJEMPLO DE ASIGNACIÓN (Simulado):

**CREATE** **USER** JuanVendedor IDENTIFIED **BY** Password123;

**GRANT** **CREATE** **SESSION** **TO** JuanVendedor;

**GRANT** ROL\_VENDEDOR **TO** JuanVendedor;

**Fase 6: Pruebas y Optimización.**

* **Pruebas Funcionales (Caja Negra):** Ejecución de las 15 consultas (Selects) para demostrar que el sistema recupera datos correctamente.
* **Pruebas de Integridad (Caja Blanca):** Intentar romper el sistema (triggers y constraints) para probar la robustez.
* **Optimización Nivel 1 (Índices y Estadísticas):** Recálculo de estadísticas para que el motor use los índices creados.
* **Optimización Nivel 2 (Vistas Materializadas):** Implementación de caché para reportes pesados.
* **Pruebas de Rendimiento (Explain Plan):** Evidencia visual de la mejora.