Universidad San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Sistemas de Bases de Datos 1 Sección
Primer Semestre 2025
Ing. Luis Espino
Ing. Álvaro Giovanni Longo
Aux. Rony Ormandy Ortíz Alvarez
Aux. Enrique Fernando Gaitán Ibarra

Enunciado

Proyecto #1

Objetivos

Objetivo General

- Diseñar una base de datos relacional eficiente y normalizada para la gestión de una empresa de ventas y distribución, aplicando normas de integridad y desarrollando consultas avanzadas que permitan generar informes específicos, empleando cargas masivas de datos desde archivos CSV.

Objetivos Específicos

- Diseñar una base de datos relacional óptima para la gestión de datos.
- Implementar procesos de normalización en una base de datos.
- Realizar cargas masivas de datos desde archivos CSV e inserción de datos desde una API a una base de datos.
- Desarrollar habilidades en SQL para formular consultas que generen información clave.

Descripción

En el contexto de los sistemas de comercio electrónico, la gestión eficiente de datos es fundamental para garantizar el correcto funcionamiento de las operaciones de compra, venta y distribución de productos. Este proyecto tiene como objetivo el diseño y desarrollo de una **base de datos relacional eficiente y normalizada** para un centro de ventas en línea de gran escala, similar a plataformas como **Amazon o Alibaba**.

La base de datos deberá estructurarse siguiendo los principios de **integridad referencial y normalización**, permitiendo el almacenamiento y procesamiento de grandes volúmenes de información de manera óptima. Se considerarán entidades clave como **usuarios, trabajadores, productos, órdenes de compra, pagos, envíos, devoluciones y traslados de productos**, asegurando una relación coherente y escalable entre los diferentes elementos del sistema.

Adicionalmente, el sistema deberá ser capaz de **incorporar datos de manera automatizada**, no solo mediante **cargas masivas desde archivos CSV**, sino también a través de una **API de integración** que permita la comunicación con plataformas externas y facilite la actualización en tiempo real de la información.

Se sabe que en un centro de ventas en línea es fundamental contar con una gestión eficiente de los usuarios, quienes interactúan en la plataforma con distintos roles. Un usuario puede ser un **cliente**, y para su identificación es necesario almacenar un identificador único, identificación nacional, nombre, apellido, correo electrónico, número de teléfono, si está activo, la fecha de registro en la plataforma y sí confirmó su correo. También debe incluir información adicional como **direcciones** y **métodos de pago** preferidos por motivos de personalización de la página.

Para el correcto funcionamiento del sistema, también es necesario registrar a los **trabajadores** encargados de la administración y logística de la plataforma. Cada trabajador debe contar con un identificador único, identificación nacional, nombre, apellido, cargo, **departamento** en el que labora, número de teléfono y correo institucional. Además, los trabajadores relacionados con la logística y almacenes deben estar vinculados a una **sede** o centro de distribución específico y si está activo.

El sistema debe gestionar una amplia variedad de **productos**, los cuales requieren un control detallado. Cada producto debe contar con un identificador único, sku del producto, nombre, descripción, precio, slug, **cantidad en inventario por sede/centro** y **categoría**. Además, se debe almacenar información sobre la disponibilidad del producto (activo o no) y una referencia a las **imágenes** asociadas para su visualización en la tienda en línea.

Por razones de control financiero y trazabilidad de las transacciones, se debe registrar la información de cada **orden de compra**. Una orden debe contar con un número de identificación único, la fecha de creación, el identificador del cliente que realizó la compra y la **lista de productos** adquiridos con sus respectivas cantidades y precios unitarios.

Los **pagos** son un componente esencial del sistema y deben ser gestionados de manera segura y eficiente. Cada pago debe estar vinculado a una orden de

compra y debe contar con un identificador único, la fecha de la transacción, el monto total, el método de pago utilizado (tarjeta de crédito, transferencia bancaria, billetera electrónica, entre otros) y el estado del pago, que puede ser "pendiente", "aprobado" o "rechazado".

Para garantizar la entrega eficiente de los productos adquiridos, se debe administrar la información de los **envíos**. Cada envío debe contar con un identificador único, la fecha de despacho, el número de orden de compra asociado, la dirección de entrega, la empresa de transporte encargada, el número de seguimiento y el estado del envío, que puede ser "en tránsito", "entregado" o "devuelto".

En el caso de productos que no cumplen con las expectativas del cliente o presentan defectos, es necesario gestionar las **devoluciones**. Cada devolución debe estar vinculada a una orden de compra y debe contar con un identificador único, la fecha de solicitud de devolución, el motivo de la devolución, el estado del proceso (en revisión, aprobada, rechazada) y el cproducto.

Para optimizar la distribución de productos entre las distintas sedes y centros de distribución, se debe administrar los **traslados internos de productos**. Cada traslado debe contar con un identificador único, la fecha de movimiento, el almacén de origen y el almacén de destino, **lista con el identificador del producto** trasladado, la cantidad transferida. Además, se debe registrar el estado del traslado y la fecha estimada de llegada al nuevo destino.

Todas las tablas que se creen tendrán que llevar los campos de fecha de creación (created_at), fecha de modificación (updated_at).

Documentación

Manual Técnico

- Esquema Conceptual
- Esquema Lógico
- Esquema Físico
- Fases del proceso de normalización, hasta forma normal 3
- Descripción de las tablas
- Descripción de la API
- Descripción de los endpoints utilizados

Manual de Usuario

El manual de usuario debe contener capturas y descripciones que le ayude al usuario a levantar y utilizar los endpoints, así como también de los resultados de cada endpoint.

Endpoints Funcionales de la API

1 Gestión de Usuarios

1.1 Crear Usuario (Registro) (POST)

- **Endpoint**: `/api/users`
- **Descripción**: Crea un nuevo usuario en la plataforma.
- **Request (JSON)**:

```
Unset
{
  "username": "jdoe",
  "email": "jdoe@example.com",
  "password": "secret123",
  "phone": "12345678"
}
```

- **Proceso Interno**:
  - Verificar que el usuario o correo no exista.
  - Hashear la contraseña con **bcrypt**.
  - Guardar en la BD (tabla `usuarios`, por ejemplo).
- **Response (JSON)**:

```
Unset
{
  "status": "success",
  "message": "User created successfully"
}
```

- **Códigos**:
  - 201 (Created): Usuario creado.
  - 400 (Bad Request): Datos incompletos.
  - 409 (Conflict): El `username` o `email` ya existe.

1.2 Iniciar Sesión (Login) (POST)

- **Endpoint**: `/api/users/login`
- **Descripción**: Autentica un usuario comparando la contraseña con el hash almacenado.
- **Request (JSON)**:

```
Unset
{
  "username": "jdoe",
  "password": "secret123"
}
```

- **Proceso Interno**:
  - Verificar la existencia de `username`.
  - Comparar la contraseña con bcrypt.
  - Establecer o devolver un **ID de sesión** (o cookie) si es válido.
- **Response (JSON)**:

```
Unset
{
  "status": "success",
  "message": "User authenticated",
  "sessionId": "abc123"  // si se maneja el ID de sesión
}
```

- **Códigos**:
  - 200 (OK): Login exitoso.
  - 401 (Unauthorized): Credenciales inválidas.
  - 400 (Bad Request): Falta algún campo.

1.3 Obtener Perfil de Usuario (GET)

- **Endpoint**: `/api/users/:id`
- **Descripción**: Obtiene la información de un usuario específico (datos básicos, sin exponer contraseña).
- **Request**:
  - Parámetro `:id` en la URL.
  - Cabecera de sesión (cookie) o similar, si se protege esta ruta.
- **Response (JSON)**:

```
Unset
{
  "id": 10,
  "username": "jdoe",
  "email": "jdoe@example.com",
  "phone": "12345678",
  "createdAt": "2025-02-01T10:00:00Z"
}
```

- **Códigos**:
  - 200 (OK): Perfil retornado.
  - 404 (Not Found): Usuario no existe.
  - 401 (Unauthorized): Falta autenticación (si se requiere).

1.4 Actualizar Usuario (PUT)

- **Endpoint**: `/api/users/:id`
- **Descripción**: Modifica los datos de un usuario (excepto la contraseña, que podría ser otro endpoint separado).
- **Request (JSON)** (campos a actualizar):

```
Unset
{
  "phone": "87654321",
  "email": "john@example.com"
}
```

- **Response (JSON)**:

```
Unset
{
  "status": "success",
  "message": "User updated successfully"
}
```

- **Códigos**:
  - 200 (OK): Actualización exitosa.
  - 400 (Bad Request): Datos inválidos.
  - 404 (Not Found): Usuario no existe.

1.5 Eliminar Usuario (DELETE)

- **Endpoint**: `/api/users/:id`
- **Descripción**: Elimina un usuario o lo marca como inactivo.
- **Response (JSON)**:

```
Unset
{
  "status": "success",
  "message": "User deleted/inactivated successfully"
}
```

- **Códigos**:
  - 200 (OK): Se eliminó/inactivó el usuario.
  - 404 (Not Found): No existe dicho usuario.

---

2 Gestión de Productos

2.1 Listar Productos (GET)

- **Endpoint**: `/api/products`
- **Descripción**: Retorna la lista de productos.
- **Response (JSON)** (ejemplo simplificado):

```
Unset
{
  "products": [
    {
      "id": 1,
      "name": "Laptop X",
      "price": 750.00,
      "stock": 10
    },
    {
      "id": 2,
      "name": "Smartphone Y",
      "price": 500.00,
      "stock": 20
    }
  ]
}
```

2.2 Detalle de Producto (GET)

- **Endpoint**: `/api/products/:id`
- **Descripción**: Obtiene el detalle de un producto específico.
- **Response (JSON)**:

```
Unset
{
  "id": 1,
  "name": "Laptop X",
  "description": "Laptop de alto rendimiento",
  "price": 750.00,
  "stock": 10,
  "category": "Electrónicos"
}
```

- **Códigos**:
  - 200 (OK): Producto encontrado.
  - 404 (Not Found): No existe el producto.

2.3 Crear Producto (POST)

- **Endpoint**: `/api/products`
- **Descripción**: Crea un nuevo producto (generalmente restringido a administradores o vendedores).
- **Request (JSON)**:

```
Unset
{
  "name": "Laptop X",
  "description": "Laptop de alto rendimiento",
  "price": 750.00,
  "stock": 10,
  "category": "Electrónicos"
}
```

- **Response (JSON)**:

```
Unset
{
  "status": "success",
  "message": "Product created successfully",
```

```
  "productId": 1
}
```

2.4 Actualizar Producto (PUT)

- **Endpoint**: `/api/products/:id`
- **Descripción**: Modifica la información de un producto existente.
- **Request (JSON)**:

```
Unset
{
  "price": 700.00,
  "stock": 15
}
```

- **Response (JSON)**:

```
Unset
{
  "status": "success",
  "message": "Product updated successfully"
}
```

2.5 Eliminar Producto (DELETE)

- **Endpoint**: `/api/products/:id`
- **Descripción**: Elimina o desactiva un producto de la base.
- **Response (JSON)**:

```
Unset
{
  "status": "success",
  "message": "Product deleted successfully"
}
```

---

3 Gestión de Órdenes

3.1 Crear Órden de Compra (POST)

- **Endpoint**: `/api/orders`
- **Descripción**: Crea una nueva órden asociada a un usuario.
- **Request (JSON)**:

```
Unset
{
  "userId": 10,
  "items": [
    {"productId": 1, "quantity": 2},
    {"productId": 2, "quantity": 1}
  ],
  "shippingAddress": "Calle 123, Ciudad",
  "paymentMethod": "credit_card"
}
```

- **Response (JSON)**:

```
Unset
{
  "status": "success",
  "message": "Order created successfully",
  "orderId": 101,
  "totalAmount": 1800.00,
  "orderStatus": "processing"
}
```

- **Códigos**:
  - 201 (Created): Orden creada.
  - 400 (Bad Request): Datos incompletos.
  - 404 (Not Found): Producto o usuario inexistente.
  - 401 (Unauthorized): No autenticado (si se requiere).

3.2 Listar Órdenes (GET)

- **Endpoint**: `/api/orders`
- **Descripción**: Retorna un listado de órdenes. Se pueden añadir filtros (por estado, fecha, etc.).
- **Response (JSON)** (ejemplo):

```
Unset
{
  "orders": [
    {
      "orderId": 101,
      "userId": 10,
      "totalAmount": 1800.00,
      "status": "processing",
      "createdAt": "2025-02-01"
    },
    {
      "orderId": 102,
      "userId": 11,
      "totalAmount": 500.00,
      "status": "shipped",
      "createdAt": "2025-02-02"
    }
  ]
}
```

3.3 Detalle de Órden (GET)

- **Endpoint**: `/api/orders/:id`
- **Descripción**: Muestra la información detallada de la órden, incluyendo ítems, cantidades, etc.
- **Response (JSON)**:

```
Unset
{
  "orderId": 101,
  "userId": 10,
  "items": [
    {"productId": 1, "quantity": 2, "price": 750.00},
    {"productId": 2, "quantity": 1, "price": 300.00}
  ],
  "totalAmount": 1800.00,
  "status": "processing",
  "createdAt": "2025-02-01"
}
```

3.4 Actualizar Estado de Órden (PUT)

- **Endpoint**: `/api/orders/:id`

- **Descripción**: Permite cambiar el estado (processing, shipped, delivered, cancelled).
- **Request (JSON)**:

```
Unset
{
  "status": "shipped"
}
```

- **Response (JSON)**:

```
Unset
{
  "status": "success",
  "message": "Order updated successfully"
}
```

---

4 Gestión de Pagos

4.1 Registrar Pago (POST)

- **Endpoint**: `/api/payments`
- **Descripción**: Registra un pago asociado a una órden.
- **Request (JSON)**:

```
JavaScript
{
  "orderId": 101,
  "amount": 1800.00,
  "method": "credit_card"
}
```

- **Response (JSON)**:

```
Unset
{
  "status": "success",
  "message": "Payment registered successfully",
```

```
  "paymentId": 501,
  "paymentStatus": "approved"
}
```

4.2 Consultar Pagos (GET)

- **Endpoint**: `/api/payments`
- **Descripción**: Lista los pagos realizados, con posibilidad de filtrar por orden, fecha, método, etc.
- **Response (JSON)**:

```
Unset
{
  "payments": [
    {
      "paymentId": 501,
      "orderId": 101,
      "amount": 1800.00,
      "method": "credit_card",
      "status": "approved",
      "createdAt": "2025-02-01T12:00:00Z"
    },
    ...
  ]
}
```

Entregables

- Código de la API
- Script de la base de datos
- Script de las consultas
- Documentación
    - Manual técnico
    - Manual de usuario

Se debe crear un repositorio en github con el siguiente nombre: **[SBD1]P1_#CARNET** e invitar a los auxiliares de sus respectivos laboratorios:

- Sección B: OrmandyRony

- Sección N: feribarra-usac

Restricciones

- El motor de base de datos a utilizar será ORACLE.
- La API puede ser desarrollada en Python, NodeJS o Golang.
- **El proyecto es individual.**
- **Las entregas tarde tendrán una penalización del 50% de la nota total.**
- La carga de datos se puede hacer a una tabla temporal y de ahí distribuirlos a las tablas correspondientes según el diseño propuesto. No está permitido que los datos se presenten tal cual como se comparten en el archivo csv para realizar las consultas.
- **Las copias totales o parciales tendrán nota de 0 puntos y serán reportadas a la escuela**.
- Modelo entidad relación podrá ser generado con la herramienta a su elección.
- Modelo relacional se podrá realizar en cualquier herramienta de modelado (Visio, DIA, ORACLE Data Modeler, etc.)

Recursos

Enlace de los archivos csv: CSVs

Fecha límite de entrega

- Domingo 9 de marzo

Semana de calificación

- Del 10 al 17 de marzo