Technical Report

Data base structure:

For the develop of the mercado free database I used the model relational that provide me the tools to develop the database with a lot advantages like the following:

- Data Integrity: Relational databases ensure referential integrity using primary and foreign keys. This is crucial in e-commerce to maintain consistency between related entities, such as products and orders.
- Advanced Queries: SQL, the standard query language for RDBMS, enables complex and powerful queries, essential for functions such as product search, category filtering, and sales reporting.
- ACID Transactions: They guarantee the Atomicity, Consistency, Isolation and Durability of transactions. This is critical for critical operations such as payment processing and inventory updating.
- Standardization: RDBMS follows well-defined standards, facilitating interoperability and maintenance.
- Security: They offer robust security features, including access control and encryption, important for protecting sensitive customer data.

And the relational model was the best option to develop mine DB because thanks to the model that used, I was able to build the structure of the database through the following tables that collected the data that could receive an e-commerce app.

- User Table: Stores customer information, their addresses, and credentials.
- Product Table: Contains the product catalog, with details such as name, description, price, manufacturer and stock.
- Order Table: Records each purchase made by customers, with references to the products purchased and their quantities.
- Shopping Cart Table: Maintains temporary information about the products selected by users.
- Ratings and Comments Table: Saves the opinions and ratings that customers leave about the products.
- Product Category Table: Facilitates the classification of products for better organization and search.

Data Dictionary

In this section I will explain how the data is composed in the tables by demonstrating the information using the concept of meta data.

Table "usuarios"

Column	Data type	Constraints	Default value	Description
id_usuarios	CHAR(36)	PRIMARY KEY, NOT NULL		Unique identifier for each user
Nombre	Varchar(50)	NOT NULL		Name of the user
contrasena	Varchar(50)	NOT NULL		Password of the user
Direccion	Varchar(50)	NOT NULL		Address of the user
Telefono	Varchar(50)	NOT NULL		Phone number of the user
Fecha_registro	Timestamp		CURRENT TIMESTAMP	Registration date and time
Email	Varchar(50)	NOT NULL		Email address of the user
Estado	Tinyint	Default 0	0	Status of the user (True = Seller, False=User)
Codigo_pais_fk	Int	FOREIGN KEY REFERENCES pais(codigo_pais)		Foreign key to the country code

Table Pais:

Column	Data type	Constraints	Default value	Description
Código Pais	**INT**	**PRIMARY KEY, not null**		**Unique identifier for each country, primary key.**
Nombre	**Varchar (50)**	**Not null**		Name of the country.

Table Cuenta Bancaria

Column	Data type	Constraints	Default value	Description
id_cuenta	CHAR(36)	PRIMARY KEY, NOT NULL		Unique identifier for each bank account, primary key.
Nombre_banco	Varchar(50)	NOT NULL		Name of the bank.
Numero_cuenta	Varchar(50)	NOT NULL, UNIQUE		Bank account number, must be unique.
Codigo_pais	INT	FOREIGN KEY		Foreign key referencing the codigo_pais in the pais table.
Id_usuario_fk	CHAR(36)	FOREIGN KEY		Foreign key referencing the id_usuarios in the usuarios table.

Table productos:

Column	Data type	Constraints	Default value	Description
id_productos	INT	PRIMARY KEY, NOT NULL		Unique identifier for each product, auto-incremented primary key.
Nombre	Varchar(50)	NOT NULL		Name of the product.
Description	Varchar(50)	NOT NULL		Description of the product.
precio	INT	NOT NULL	0	Price of the product, default value is 0.
fabricante	Varchar(50)			Manufacturer of the product.
stock	INT	NOT NULL	0	Stock quantity of the product, default value is 0.
fecha_creacion	Timestamp		CURRENT_TIMESTAMP	Creation date and time of the product record, default is current timestamp.

Table pedidos

Column	Data type	Constraints	Default value	Description
id_pedidos	INT	PRIMARY KEY, NOT NULL		Unique identifier for each pedido.
fecha_creacion	Timestamp		CURRENT_TIMESTAMP	Timestamp indicating the date and time when the pedido was placed.
estado	Varchar(50)			Describes the current state of the pedido.(Entregado,Activo, Cancelado)
metodo_pago	Varchar(50)			Specifies the payment method used for the pedido.(Efectivo,Tarjeta)
id_historial_fk	INT			Foreign key referencing the id_historial column in another table.
id_usuario_fk	CHAR(36)			Foreign key referencing the id_usuarios column in the usuarios table.

Table Lista_Prouctos

Column	Data type	Constraints	Default value	Description
id_lista_productos	INT	PRIMARY KEY, NOT NULL		Unique identifier for each lista_productos entry.
id_productos_fk	INT			Foreign key referencing the id_productos column in the productos table.
id_pedidos_fk	INT			Foreign key referencing the id_pedidos column in the pedidos table.
id_carrito_fk	INT			Foreign key referencing the id_carrito column in the carrito table.
cantidad	INT	NOT NULL	0	Represents the quantity of productos in the lista_productos entry.

Table Historial

Column	Data type	Constraints	Default value	Description
id_historial	INT	PRIMARY KEY, NOT NULL		Unique identifier for each historial entry.
fecha_historial	TimeStamp		CURRENT_TIMESTAMP	Timestamp indicating the date and time when the historial entry was created.
estado	Varchar(50)			Represents the state or status of the historial entry.(Completado,En transito,Cancelado ,pendiente)
id_usuario_fk	CHAR(36)			Foreign key referencing the id_usuarios column in the usuarios table.

Table Comentarios

Column	Data type	Constraints	Default value	Description
id_comentarios	INT	PRIMARY KEY, NOT NULL		Unique identifier for each Comment entry.
texto	varchar(300)	NOT NULL		The text content of the comment.
fecha_creacion_comentario	TimeStamp		CURRENT_TIMESTAMP	Timestamp indicating the date and time when the comment was created.
id_producto_fk	INT			Foreign key referencing the id_productos column in the productos table.
id_usuario_fk	char (36)			Foreign key referencing the id_usuarios column in the usuarios table.

Table Categoria_productos

Column	Data type	Constraints	Default value	Description
id_categorias_productos	INT	PRIMARY KEY, NOT NULL		Unique identifier for each category-product entry.
id_categoria_fk	TimeStamp		CURRENT_TIMESTAMP	Foreign key referencing the id_categoria column in the categoria table.
id_producto_fk	INT			Foreign key referencing the id_productos column in the productos table.

Table Categoria

Column	Data type	Constraints	Default value	Description
id_categoria	INT	PRIMARY KEY, NOT NULL		Unique identifier for each category entry.
nombre	Varchar(40)	NOT NULL		The name of the category.
descripcion	Varchar(40)	NOT NULL		The description of the category.

Table Carrito

Column	Data type	Constraints	Default value	Description
id_carrito	INT	PRIMARY KEY, NOT NULL		Unique identifier for each Carrito entry.
estado	bool	NOT NULL	False	The state of the cart, default is false.(False= desactivado, True= activado)
fecha_creacion_carrito	TimeStamp		CURRENT_TIMESTAMP	Timestamp indicating the date and time when the carrito was created.
id_usuario_fk	char (36)			Foreign key referencing the id_usuarios column in the usuarios table.

Table calificaciones

Column	Data type	Constraints	Default value	Description
id_calificaciones	INT	PRIMARY KEY, NOT NULL		Unique identifier for each calificacion entry.
puntuacion	INT	CHECK (puntuacion >= 0 AND puntuacion <= 5)	0	The state of the cart, default is false.(False= desactivado, True= activado)
id_producto_fk	INT			Foreign key referencing the id_productos column in the productos table.
id_usuario_fk	char (36)			Foreign key referencing the id_usuarios column in the usuarios table.

Web Services

These are the web services based on user histories that we think our users will do. The web services are Mounted on postman and using the language programming python for the connection between the databases and the Api “postman”

- **User Authentication:**
  - Endpoint: /login
  - Method: POST
  - Description: Allows registered users to log in to the platform using their email and password credentials.
- **User Registration:**
  - Endpoint: /register
  - Method: POST
  - Description: Allows users to register on the platform by providing personal and contact information to create a new account.
- **Product Search:**
  - Endpoint: /products/search
  - Method: GET
  - Description: Enables users to search for products by categories, keywords, or specific features to quickly find what they need.
- **Product Listing:**
  - Endpoint: /products
  - Method: GET
  - Description: Retrieves a list of products with images, detailed descriptions, prices, and ratings to assist users in making informed purchase decisions.
- **Shopping Cart Management:**
  - Endpoint: /cart
  - Method: POST (Add to Cart), DELETE (Remove from Cart)
  - Description: Allows users to add products to their shopping cart and remove them from the cart.
- **Checkout Process:**
  - Endpoint: /checkout
  - Method: POST
  - Description: Enables users to proceed with the checkout process to complete their purchase.

- **Product Management (Admin):**
  - Endpoint: /admin/products
  - Method: POST (Add Product), PUT (Edit Product), DELETE (Delete Product)
  - Description: Provides site administrators with the ability to add, edit, and delete products from the catalog to keep the information up-to-date.
- **User Account Management (Admin):**

    - Endpoint: /admin/users
    - Method: POST (Approve Registration), PUT (Reset Password), DELETE (Delete User)
    - Description: Allows site administrators to manage user accounts, including approving registrations, resetting passwords, and deleting users.
- **Business Analytics (Admin):**
    - Endpoint: /admin/analytics
    - Method: GET
    - Description: Provides site administrators with access to sales reports, performance metrics, and inventory data to make informed business decisions.
- **Product Reviews and Ratings:**
    - Endpoint: /products/{product_id}/reviews
    - Method: POST (Add Review), GET (Retrieve Reviews)
    - Description: Allows users to leave reviews and ratings on products they have purchased to assist other users in their purchasing decisions.

Tests carried out on web services

1. Health Check (`GET /`)

Prueba: Validar que el servicio web esté activo y responda correctamente.

- **Solicitud**: `GET /`
- **Resultado esperado**:

```
{
  "status": "ok"
}
```

2. Login (`POST /login`)

Prueba: Validar el proceso de inicio de sesión con credenciales correctas e incorrectas.

- **Solicitud correcta**:

```
{
  "email": "test@example.com",
  "contrasena": "correctpassword"
}
```

    - **Resultado esperado**: Datos del usuario.
- **Solicitud incorrecta**:

```
{
  "email": "test@example.com",
  "contrasena": "wrongpassword"
}
```

- **Resultado esperado**: Error 404, usuario no encontrado.

3. Registro de Usuario (`POST /register`)

Prueba: Validar el proceso de registro de un nuevo usuario.

- **Solicitud**:

```
json
Copiar código
{
  "nombre": "Nuevo Usuario",
  "contrasena": "password",
  "direccion": "123 Calle Falsa",
  "telefono": "123456789",
  "email": "newuser@example.com",
  "codigo_pais_fk": 1
}
```

  - **Resultado esperado**:

```
json
Copiar código
{
  "message": "Usuario registrado exitosamente"
}
```

4. Búsqueda de Productos (`POST /productos/search`)

Prueba: Validar la búsqueda de productos con diferentes filtros.

- **Solicitud**:

```
json
Copiar código
{
  "categoria_id": 1,
  "keyword": "producto",
  "fabricante": "fabricanteX"
}
```

  - **Resultado esperado**: Lista de productos que coincidan con los filtros.

5. Detalles de Productos (`GET /productos/detalles`)

Prueba: Validar la obtención de detalles de productos con el promedio de calificaciones.

- **Solicitud**: `GET /productos/detalles`
- **Resultado esperado**: Lista de productos con sus detalles y promedio de calificaciones.

6. Checkout del Carrito (`POST /carrito/checkout`)

Prueba: Validar el proceso de checkout de un carrito.

- **Solicitud**:

```
{
  "id_carrito": 1
}
```

  - **Resultado esperado**:

```
{
  "message": "Compra realizada"
}
```

7. Añadir Producto (Admin) (`POST /productos/admin/add`)

Prueba: Validar la adición de un nuevo producto por el administrador.

- **Solicitud**:

```
{
  "nombre": "Producto Nuevo",
  "descripcion": "Descripción del producto",
  "precio": 100,
  "fabricante": "FabricanteX",
  "stock": 50
}
```

  - **Resultado esperado**:

```
{
  "message": "Producto agregado"
}
```

8. Actualizar Producto (Admin) (`PUT /productos/admin/update/{id_productos}`)

Prueba: Validar la actualización de un producto existente por el administrador.

- **Solicitud**:

```
{
  "nombre": "Producto Actualizado",
  "descripcion": "Nueva descripción",
  "precio": 150,
  "fabricante": "FabricanteY",
  "stock": 30
}
```

- **Resultado esperado**:

```
json
Copiar código
{
  "message": "Producto actualizado"
}
```

9. Eliminar Producto (Admin) (`DELETE /productos/admin/delete/{id_productos}`)

Prueba: Validar la eliminación de un producto por el administrador.

- **Solicitud**: `DELETE /productos/admin/delete/1`
- **Resultado esperado**:

```
json
Copiar código
{
  "message": "Producto eliminado"
}
```

10. Aprobar Usuario (Admin) (`PUT /usuarios/admin/approve`)

Prueba: Validar la aprobación de un usuario por el administrador.

- **Solicitud**:

```
json
Copiar código
{
  "id_usuarios": 1
}
```

- **Resultado esperado**:

```
json
Copiar código
{
  "message": "Usuario aprobado"
}
```

11. Restablecer Contraseña (Admin) (`PUT /usuarios/admin/reset-password`)

Prueba: Validar el restablecimiento de la contraseña de un usuario por el administrador.

- **Solicitud**:

```
json
Copiar código
```

```
{
  "email": "user@example.com",
  "new_password": "newpassword"
}
```

- **Resultado esperado**:

```
{
  "message": "Contraseña actualizada"
}
```

12. Métricas de Rendimiento (`GET /informes/rendimiento`)

Prueba: Validar la obtención de métricas de rendimiento de pedidos.

- **Solicitud**: `GET /informes/rendimiento`
- **Resultado esperado**: Lista de métricas de rendimiento.

13. Datos de Inventario (`GET /informes/inventario`)

Prueba: Validar la obtención de datos de inventario de productos.

- **Solicitud**: `GET /informes/inventario`
- **Resultado esperado**: Lista de productos con su stock.

14. Ventas por Fabricante (`GET /informes/ventas/fabricantes`)

Prueba: Validar la obtención de datos de ventas por fabricante.

- **Solicitud**: `GET /informes/ventas/fabricantes`
- **Resultado esperado**: Lista de ventas por fabricante.

15. Eliminar Calificaciones de Productos (`DELETE /productos/eliminarCalificacion`)

Prueba: Validar la eliminación de calificaciones de productos según filtros.

- **Solicitud**:

```
{
  "keyword": "producto",
  "category": "categoríaX"
}
```

- **Resultado esperado**:

```
{
  "message": "Calificaciones eliminadas correctamente"
```

```
}
```

16. Eliminar Comentarios (`DELETE /eliminar/comentarios`)

Prueba: Validar la eliminación de comentarios basados en el id del producto y el id del usuario.

- **Solicitud**:

```
{
  "product_id": 1,
  "user_id": 1
}
```

  - **Resultado esperado**:

```
{
  "message": "Comentarios eliminados correctamente"
}
```

Strategy to fill the database with information

To fill out our database we use the following strategies and tools to make this process easier for us and we could develop it in the most efficient and quickest way possible.

1. **Mockaroo Data Generation:**
    - Utilized Mockaroo to create custom datasets with diverse data types and configurations.
    - Generated data for each database table, ensuring coverage of all relevant fields and scenarios.
    - Downloaded the generated data in formats like CSV or JSON for further use.
2. **Random Data Generation:**
    - Employed random data generation for non-critical fields such as product descriptions or user interests.
    - Used libraries like Faker in Python or Chance.js in JavaScript to create randomized yet realistic data.
    - Adjusted randomness parameters to control data distribution and characteristics.
3. **Integration Testing:**
    - Incorporated database population into integration tests to validate functionality and performance.
    - Leveraged testing frameworks like pytest, Mocha, or Jasmine to automate test execution involving the populated database.
    - Verified that populated data accurately reflected the expected application state and covered various edge cases and scenarios.

This approach ensured that the database was adequately populated with realistic and relevant data for application development and testing purposes.