**SQL Server Case Study ( Library Management System )**

**Problem Statement:** Online Store Inventory

**Background**

An emerging e-commerce business, "TechTrend Innovations," operates an online store selling electronics and accessories, such as laptops, headphones, and chargers. To streamline operations, the company requires a database to manage its product inventory, customer information, and order processing. The database should ensure data integrity, enforce business rules, and support basic reporting to track sales and inventory levels. As a database developer, your task is to design and implement a Microsoft SQL Server database that meets these needs while incorporating fundamental database concepts, tables, columns, and constraints.

**Objectives**

The goal is to create a robust database system that:

1. Stores information about products, customers, orders, and order details.
2. Enforces business rules using appropriate constraints (e.g., primary key, foreign key, unique, not null, check, default).
3. Supports basic queries to retrieve and update data, such as listing low-stock products, calculating customer spending, and adjusting inventory after orders.
4. Provides a foundation for beginner-to-intermediate SQL learners to practice database creation, data manipulation, and querying.

**Requirements**

**1. Database Structure**

* **Database Name**: StoreDB
* **Tables**:
  + **Products**: Stores details about items sold in the store.
  + **Customers**: Stores customer information for order processing.
  + **Orders**: Tracks customer orders with order date and total amount.
  + **OrderDetails**: Links orders to products, specifying quantities purchased.
* Each table must include appropriate columns with correct data types and constraints to enforce business rules.

**2. Business Rules**

The database must enforce the following rules using SQL Server constraints:

* **Products**:
  + Each product must have a unique identifier.
  + Product names and categories are mandatory.
  + Prices must be positive values (greater than 0).
  + Stock quantities default to 0 if not specified.
* **Customers**:
  + Each customer must have a unique identifier.
  + First and last names are mandatory.
  + Email addresses must be unique to prevent duplicate customer accounts.
  + Phone numbers are optional.
* **Orders**:
  + Each order must have a unique identifier.
  + Orders must be linked to a valid customer.
  + Order dates default to the current date if not provided.
  + Total order amounts must be non-negative.
* **OrderDetails**:
  + Each order detail record must have a unique identifier.
  + Must link to a valid order and product.
  + Quantities ordered must be positive (greater than 0).

**3. Functional Requirements**

* **Data Insertion**: Populate the database with sample data for at least:
  + 2 products (e.g., a laptop and headphones).
  + 2 customers with contact details.
  + 2 orders, each associated with a customer.
  + Order details linking products to orders with quantities.
* **Queries**:
  + Retrieve products with low stock (e.g., less than 20 units).
  + Calculate the total amount spent by each customer.
  + Update product stock quantities after orders are placed to reflect purchased items.
* **Constraints**:
  + Use **primary keys** to uniquely identify records in each table.
  + Use **foreign keys** to enforce relationships between tables (e.g., Orders to Customers, OrderDetails to Orders and Products).
  + Apply **not null** constraints for mandatory fields.
  + Use **unique** constraints to prevent duplicate emails and other unique fields.
  + Implement **check** constraints to ensure valid data (e.g., positive prices and quantities).
  + Use **default** constraints to set default values where applicable.

**4. Technical Requirements**

* **Platform**: Microsoft SQL Server (any recent version, e.g., SQL Server 2019 or 2022).
* **SQL Features**:
  + Use basic DDL (Data Definition Language) commands like CREATE DATABASE, CREATE TABLE.
  + Use DML (Data Manipulation Language) commands like INSERT, SELECT, UPDATE.
  + Implement joins to combine data from multiple tables.
  + Use aggregation (SUM, GROUP BY) for reporting.
  + Include subqueries where appropriate (e.g., updating stock based on order details).
* **Constraints**:
  + Primary Key: Ensure unique identification of records.
  + Foreign Key: Maintain referential integrity.
  + Not Null: Enforce mandatory fields.
  + Unique: Prevent duplicate entries in specific columns.
  + Check: Restrict data to valid ranges or values.
  + Default: Provide fallback values for optional fields.

**5. Deliverables**

* SQL scripts to:
  + Create the StoreDB database.
  + Define tables (Products, Customers, Orders, OrderDetails) with all specified columns and constraints.
  + Insert sample data to test the database.
  + Execute queries to demonstrate functionality (e.g., low-stock report, customer spending, stock updates).
* Documentation (optional for learners):
  + Brief explanation of each table’s purpose and constraints.
  + Description of how queries meet the business needs.

**6. Sample Use Cases**

* **Inventory Management**: The store manager needs to identify products with low stock to reorder them.
* **Customer Insights**: The sales team wants to know which customers are spending the most to offer loyalty discounts.
* **Order Processing**: When an order is placed, the system must update the stock quantity to reflect items sold.

**7. Constraints and Assumptions**

* The database is for a small-scale store with a limited number of products and customers.
* No advanced features like triggers, stored procedures, or views are required for this beginner-to-intermediate case study.
* All monetary values (e.g., Price, TotalAmount) are in a single currency (e.g., USD).
* Stock quantities are updated manually via SQL queries for this case study (no automated triggers).

**8. Learning Objectives**

* **Fundamentals**: Understand how to create a database, define tables, and use basic SQL commands.
* **Tables and Columns**: Design tables with appropriate data types (e.g., INT, VARCHAR, DECIMAL, DATE).
* **Constraints**: Apply and understand the role of primary key, foreign key, not null, unique, check, and default constraints.
* **Intermediate Skills**: Write queries involving joins, aggregations, and subqueries to solve business problems.

**9. Challenges for Learners**

* Ensure all constraints are correctly implemented to prevent invalid data (e.g., negative prices, duplicate emails).
* Write efficient queries to retrieve and update data without errors.
* Understand how foreign keys maintain relationships between tables.
* Test the database by attempting to insert invalid data (e.g., negative quantities) to observe constraint violations.

**10. Expected Output**

* A functional StoreDB database with four tables: Products, Customers, Orders, and OrderDetails.
* Sample data inserted into each table.
* Query results demonstrating:
  + Products with low stock.
  + Total spending per customer.
  + Updated stock quantities after orders.
* No errors when running the SQL scripts in SQL Server Management Studio (SSMS) or a similar tool.