Week 3 Lab

Due Date: Tuesday, 19 November 2024

Introduction

TechCorp, a rapidly growing tech company, relies on an old database system that stores all its data in a single, sprawling table. This table contains details on products, customers, orders, and inventory, leading to data redundancy, slow queries, and difficulty maintaining data integrity. To improve efficiency and scalability, TechCorp needs to redesign this system into a well-structured relational database.

Part 1: Analyzing TechCorp's Existing Table

Objectives:

By the end of this lab, you will be able to:

- Analyze and normalize an unstructured table to reduce redundancy.
- Design a relational schema from a single-table database.
- Implement ACID-compliant transactions to maintain data integrity.
- Apply concurrency control techniques to handle simultaneous transactions safely.

Given the following table "TechCorp_Data" which includes information on products, customers, orders and inventory

OrderID	Customer Name	CustomerEmail	ProductID	ProductName	Category	CustomerAddress	Supplier	OrderDate	Quantity	Price	Total
001	Jean Doe	jsmith@lmail.com	101	Laptop	Electronics	Ghana-Accra	CompuGhana	2023-11-01	1	1000	1000
002	Rebecca Yeboah	ryeb@lmail.com	102	Phone	Electronics	Rwanda-Kigali	RoboTech	2023-11-02	2	500	1000
003	Jean Doe	jdoe@lmail.com	103	Mouse	Electronics	Ghana-Accra	RapasatTech	2024-11-02	2	25	50
004	Bertina Ayuure	bayuure@lmail.com	104	Keyboard	Accessories	German-Bonn	T-Shop	2024-11-03	1	10	10

Tasks:

- Break down the data into separate tables that meet 3NF requirements.
- Explain your design choices for primary and foreign keys in each normalized table.
- Create an ERD to visualize the relationships of the new schema. (NB: Ensure primary and foreign keys are correctly defined, do well to label relationships).

Discussion:

- Discuss how normalization helps reduce redundancy and improve data integrity.
- Consider potential trade-offs and when denormalization might be necessary.

Part 2: Transaction Management with ACID Properties

Design a Transaction Scenario
Imagine a customer placing an order in TechCorp's online store. This order transaction should include:

- Inserting a new record in the Orders table.
- Adding each item from the order into the OrderDetails table.
- Decreasing the quantity in stock for each item in the Products table.

1. **Apply ACID Properties**
   - **Atomicity:** The entire order process should be treated as one transaction (if any part fails, the entire transaction should fail).
   - **Consistency:** The database should be updated only if all steps in the transaction succeed, leaving it in a consistent state.
   - **Isolation:** Ensure the transaction operates independently from other transactions.
   - **Durability:** Once completed, the transaction should be saved permanently.
2. **Write the Transaction Code (or Pseudocode)**
   Draft working SQL code that applies each ACID property to the order transaction.

Part 3: Preventing Concurrency Issues in a Multi-User Environment

TechCorp's new online store has gone live, and customers are now placing orders simultaneously. One popular product, the "TechCorp Smart Speaker," has only **10 units** left in stock. Two customers, Alex and Taylor, both try to order 5 units of the speaker at the same time.

Without proper concurrency control, this could lead to:

- **Lost Update:** Both orders succeed, allowing sales beyond available stock.
- **Dirty Read:** One order accesses data that another order hasn't fully committed yet.

Your task is to set up concurrency controls that prevent these issues and ensure accurate stock management.

Steps to Complete the Task

1. Set Up the Concurrency Scenario

1. Assume the following starting table setup for the Products table:

ProductID	ProductName	StockQuantity
101	TechCorp Smart Speaker	10

2. Assume two transactions are being processed at the same time for the same product:
    - Transaction 1 (Alex's order): Decreases stock by 5 units.
    - Transaction 2 (Taylor's order): Decreases stock by 5 units.

2. Implement Row-Level Locking to Prevent Conflicts

1. **Choose a Locking Mechanism:** Use a **row-level lock** on the Products table to prevent both transactions from updating the same row simultaneously.
2. **Write a SQL Script for the Locking Process**: Write code that:
    - Starts **Transaction 1** for Alex.
    - Places a **lock** on the row for ProductID 101.
    - Decreases StockQuantity by 5 for ProductID 101.
    - Commits Transaction 1 and releases the lock.
3. **Simulate Transaction 2 (Taylor's order)**:
    - Transaction 2 will now wait for Transaction 1 to complete and release the lock before it can proceed.
    - Once Transaction 1 is complete, Transaction 2 can begin, verify the remaining stock, decrease the stock by 5, and commit.

Part 4: Document Stores with MongoDB – Designing a Product Catalog

MongoDB is a document-oriented NoSQL database that allows you to store data in flexible, JSON-like documents. This structure is ideal for handling data with diverse attributes, such as product catalogs where each product may have unique characteristics.

Activity 1: Set Up a MongoDB Database

1. **Set Up MongoDB**
2. **Create a Database** named ProductCatalog.
3. **Create a Collection** named Products where each document represents a unique product.

Activity 2: Design Product Documents

Design documents to store different types of products, considering that each product type might have unique attributes. For example:

- **Electronics** (e.g., ProductID, Name, Brand, Price, Specifications).
- **Clothing** (e.g., ProductID, Name, Size, Color, Material, Price).
- **Books** (e.g., ProductID, Title, Author, Genre, ISBN, Price).

Each document should contain only relevant attributes to its product type, showcasing MongoDB's schema flexibility.

Activity 3: Insert Sample Documents

Use MongoDB queries to insert several sample product documents into the Products collection.

Part 5: Data Lakes and Lakehouses

- **Activity:** Design a data lake architecture for TechCorp's analytics platform.
- **Objective:** Understand the components and benefits of data lakes and lakehouses.
- **Task:** Create a high-level architecture diagram and outline the data ingestion, storage, and processing workflows.

Summary

This lab provides hands-on experience with various data storage technologies, enabling learners to design and implement solutions tailored to specific use cases. By the end of the lab, learners will have a comprehensive understanding of relational and NoSQL databases, as well as data lakes and lakehouses, preparing them for real-world data management challenges.