Module 14

What to learn

Trigger
DML Trigger
DDL Trigger
After Trigger
Instead of Trigger

Practice Exercise

Practice 1

Do the hands on the things provided in video and ppt.

Practice 2

Practice: Create a Basic Trigger

Create a trigger named 'log_insert_customer' that is triggered automatically after an INSERT operation on the 'CustomerDetails' table. The trigger should add an entry into a separate 'CustomerLog' table to track the ID and insertion time of each new customer. Begin by creating the 'CustomerLog' table with columns for LogID (auto-increment primary key), CustID, and InsertionTime.

Purpose

This practice will help you understand the basics of AFTER triggers and how they can be used to monitor and log changes in a table.

Practice 3

Practice: Create a DML Trigger for Update Logging

Write a DML trigger named 'log_update_customer' that tracks any UPDATE operation on the 'CustomerDetails' table. When a customer's address is updated, the trigger should insert a log into an 'AddressUpdateLog' table with columns for LogID (auto-increment primary key), CustID, OldAddress, NewAddress, and UpdateTime.

Table Creation

First, create the 'AddressUpdateLog' table, and then link the trigger to the 'CustomerDetails' table to capture updates.

Expected Result

Whenever a customer's address is updated, the changes should be logged with the old and new address values.

Practice 4

Practice: Implement an INSTEAD OF Trigger

Create an INSTEAD OF INSERT trigger named 'handle_booking_insert' for the 'BookingDetails' table. Assume there's a business rule that bookings cannot be made for dates older than today's date. The trigger should reject such invalid inserts by raising an error, while allowing valid bookings to proceed.

Focus

This exercise will help you understand how INSTEAD OF triggers can be used to enforce custom constraints and business rules during DML operations.

Practice 5

Practice: DDL Trigger to Audit Table Changes

Create a DDL trigger named 'audit_schema_changes' to monitor and log any changes to the schema of the 'CustomerDetails' table. The trigger should log events like creation, alteration, or dropping of the table into an 'SchemaAuditLog' table with columns: AuditID (auto-increment primary key), EventType, EventTime, and ModifiedBy.

Goal

This exercise emphasizes using DDL triggers to monitor and audit schema-level operations.

Practice 6

Practice: AFTER Trigger for Cascaded Updates

Design an AFTER trigger named 'update_booking_on_customer' on the 'CustomerDetails' table. When a customer's name or contact details are changed, the corresponding records in the 'BookingDetails' table should also be updated to maintain data consistency. Use the customer ID as the linking key between the two tables.

Output

After any update in 'CustomerDetails', changes should reflect seamlessly in the 'BookingDetails' table.

Assignment Exercise

Assignment 1

Student -> StudentID, StudentName, TotalFees, RemainingAmt Course -> CourseID, CourseName, TotalFees CourseEnrolled -> StudentID, CourseID FeePayment ->StudentID, AmountPaid, DateofPayment Create an insert trigger on

CourseEnrolled Table, record should be updated in TotalFees Field on the Student table for the respective student. Create an insert trigger on FeePayment, record should be updated in RemainingAmt Field of the Student Table for the respective student.

Assignment 2

Assignment: Real-World Booking and Transaction System

Implement a trigger-based system for a travel company's booking database that ensures data integrity, auditing, and rule enforcement. Use the following scenario: A travel booking company needs tighter management of its database comprising the tables BusRecords, CustomerDetails, BookingDetails, and TransactionDetails (structures provided below).

Functional Requirements

Create DML triggers to log all INSERT, UPDATE, and DELETE actions on the 'BookingDetails' table. For each operation, record the BookingId, OperationType (INSERT/UPDATE/DELETE), OperationTime, and UserName performing the action in a 'BookingAuditLog' table.

Set up a DDL trigger to monitor schema changes (CREATE/ALTER/DROP) on the 'TransactionDetails' table. Log these events to a table 'TransactionDDLLog' with columns: LogID, Event, TableName, EventTime, and PerformedBy.

Create a trigger that ensures all transactions in 'TransactionDetails' are checked for validity. For example, if a transaction amount exceeds the customer's previous total bookings amount by 200%, raise an error and block the transaction.

Design an AFTER trigger to update total booking amounts in 'CustomerDetails' whenever a new transaction is added in 'TransactionDetails' or when an existing transaction amount is updated.

Business Logic

All booking dates must lie in the future relative to the current date. Implement an INSTEAD OF trigger on the 'BookingDetails' table to enforce this check.

Data consistency must be maintained across tables. Use AFTER triggers to update the 'BookingDetails' and 'CustomerDetails' tables when relevant fields are modified.

All logs and audits must store their respective timestamps and the user responsible for the changes (default to SYSTEM for this exercise).

The transaction validity rule ensures no fraudulent or overly large transactions by comparing against historical data.

Deliverables

SQL scripts to create necessary tables and constraints for the system.
Triggers for the described business rules and auditing system.
A report summarizing how triggers were implemented, along with examples of trigger execution logs from test data.

Online Reference

No online Reference

Introduction to Relational Databases

Introduction to Select Statement

Filtering Results with WHERE Statements

Utilizing Joins

Executing Sub queries and Unions

Aggregating Data

Advanced Data Aggregations

Built in Functions

Query Optimization

Modifying Data

Advanced Data Modification

Stored Procedure

Transaction

Error handling

Designing Tables

triggers