# **Real-Time Audit Logging System in SQL Server**

Implementing a real-time audit logging system in SQL Server involves capturing changes made to critical tables, such as Employees and Salaries, and recording these changes in an AuditLogs table. This can be achieved using triggers that respond to INSERT, UPDATE, and DELETE operations. Below is a step-by-step guide to creating such a system.

AuditLoggingProject/

│

├── Tables/

│   └── Create\_AuditLogs\_Table.sql

│

├── Triggers/

│   ├── Employees/

│   │   ├── trg\_Employees\_Insert.sql

│   │   ├── trg\_Employees\_Update.sql

│   │   └── trg\_Employees\_Delete.sql

│   └── Salaries/

│       ├── trg\_Salaries\_Insert.sql

│       ├── trg\_Salaries\_Update.sql

│       └── trg\_Salaries\_Delete.sql

│

└── README.md

**1. Create the AuditLogs Table**

First, define the AuditLogs table to store the audit information. This table should include columns to capture relevant details about the changes, such as the operation performed, the user who made the change, and timestamps.

CREATE TABLE AuditLogs (

    AuditID INT IDENTITY(1,1) PRIMARY KEY,

    TableName NVARCHAR(128),

    Operation NVARCHAR(10),

    KeyValue NVARCHAR(1000),

    OldData NVARCHAR(MAX),

    NewData NVARCHAR(MAX),

    ChangedBy NVARCHAR(128),

    ChangedOn DATETIME DEFAULT GETDATE()

);

* AuditID: Unique identifier for each audit record.
* TableName: Name of the table where the change occurred.
* Operation: Type of operation (INSERT, UPDATE, or DELETE).
* KeyValue: Primary key value(s) of the affected row.
* OldData: JSON representation of the row data before the change.
* NewData: JSON representation of the row data after the change.
* ChangedBy: Username of the person who made the change.
* ChangedOn: Timestamp of when the change occurred.

**2. Create Triggers for the Employees Table**

Next, create triggers on the Employees table to log INSERT, UPDATE, and DELETE operations.

**a. INSERT Trigger**

CREATE TRIGGER trg\_Employees\_Insert

ON Employees

AFTER INSERT

AS

BEGIN

    INSERT INTO AuditLogs (TableName, Operation, KeyValue, NewData, ChangedBy)

    SELECT

        'Employees',

        'INSERT',

        CONCAT('EmployeeID=', CAST(i.EmployeeID AS NVARCHAR(100))),

        (SELECT \* FROM INSERTED i FOR JSON PATH),

        SUSER\_SNAME()

    FROM INSERTED i;

END;

**b. UPDATE Trigger**

CREATE TRIGGER trg\_Employees\_Update

ON Employees

AFTER UPDATE

AS

BEGIN

    INSERT INTO AuditLogs (TableName, Operation, KeyValue, OldData, NewData, ChangedBy)

    SELECT

        'Employees',

        'UPDATE',

        CONCAT('EmployeeID=', CAST(i.EmployeeID AS NVARCHAR(100))),

        (SELECT \* FROM DELETED d FOR JSON PATH),

        (SELECT \* FROM INSERTED i FOR JSON PATH),

        SUSER\_SNAME()

    FROM INSERTED i

    INNER JOIN DELETED d ON i.EmployeeID = d.EmployeeID;

END;

**c. DELETE Trigger**

CREATE TRIGGER trg\_Employees\_Delete

ON Employees

AFTER DELETE

AS

BEGIN

    INSERT INTO AuditLogs (TableName, Operation, KeyValue, OldData, ChangedBy)

    SELECT

        'Employees',

        'DELETE',

        CONCAT('EmployeeID=', CAST(d.EmployeeID AS NVARCHAR(100))),

        (SELECT \* FROM DELETED d FOR JSON PATH),

        SUSER\_SNAME()

    FROM DELETED d;

END;

**3. Create Triggers for the Salaries Table**

Similarly, create triggers for the Salaries table.

**a. INSERT Trigger**

CREATE TRIGGER trg\_Salaries\_Insert

ON Salaries

AFTER INSERT

AS

BEGIN

    INSERT INTO AuditLogs (TableName, Operation, KeyValue, NewData, ChangedBy)

    SELECT

        'Salaries',

        'INSERT',

        CONCAT('SalaryID=', CAST(i.SalaryID AS NVARCHAR(100))),

        (SELECT \* FROM INSERTED i FOR JSON PATH),

        SUSER\_SNAME()

    FROM INSERTED i;

END;

**b. UPDATE Trigger**

CREATE TRIGGER trg\_Salaries\_Update

ON Salaries

AFTER UPDATE

AS

BEGIN

    INSERT INTO AuditLogs (TableName, Operation, KeyValue, OldData, NewData, ChangedBy)

    SELECT

        'Salaries',

        'UPDATE',

        CONCAT('SalaryID=', CAST(i.SalaryID AS NVARCHAR(100))),

        (SELECT \* FROM DELETED d FOR JSON PATH),

        (SELECT \* FROM INSERTED i FOR JSON PATH),

        SUSER\_SNAME()

    FROM INSERTED i

    INNER JOIN DELETED d ON i.SalaryID = d.SalaryID;

END;

**c. DELETE Trigger**

CREATE TRIGGER trg\_Salaries\_Delete

ON Salaries

AFTER DELETE

AS

BEGIN

    INSERT INTO AuditLogs (TableName, Operation, KeyValue, OldData, ChangedBy)

    SELECT

        'Salaries',

        'DELETE',

        CONCAT('SalaryID=', CAST(d.SalaryID AS NVARCHAR(100))),

        (SELECT \* FROM DELETED d FOR JSON PATH),

        SUSER\_SNAME()

    FROM DELETED d;

END;

**4. Considerations**

* **Performance**: Implementing triggers can introduce overhead, especially on tables with high transaction volumes. It's essential to test the performance impact in a development environment before deploying to production.
* **Security**: Ensure that only authorized users have access to the AuditLogs table to maintain data integrity and confidentiality.
* **Scalability**: For large-scale applications, consider using SQL Server's built-in auditing features or third-party tools designed for auditing to handle more extensive auditing requirements efficiently.

By following these steps, you can establish a real-time audit logging system in SQL Server that captures and records changes to critical tables, providing a valuable trail of data modifications for security and compliance purposes.