Database Using SQL Server & Raw Data available over my GitHub Profile

MySQL

PostgreSQL

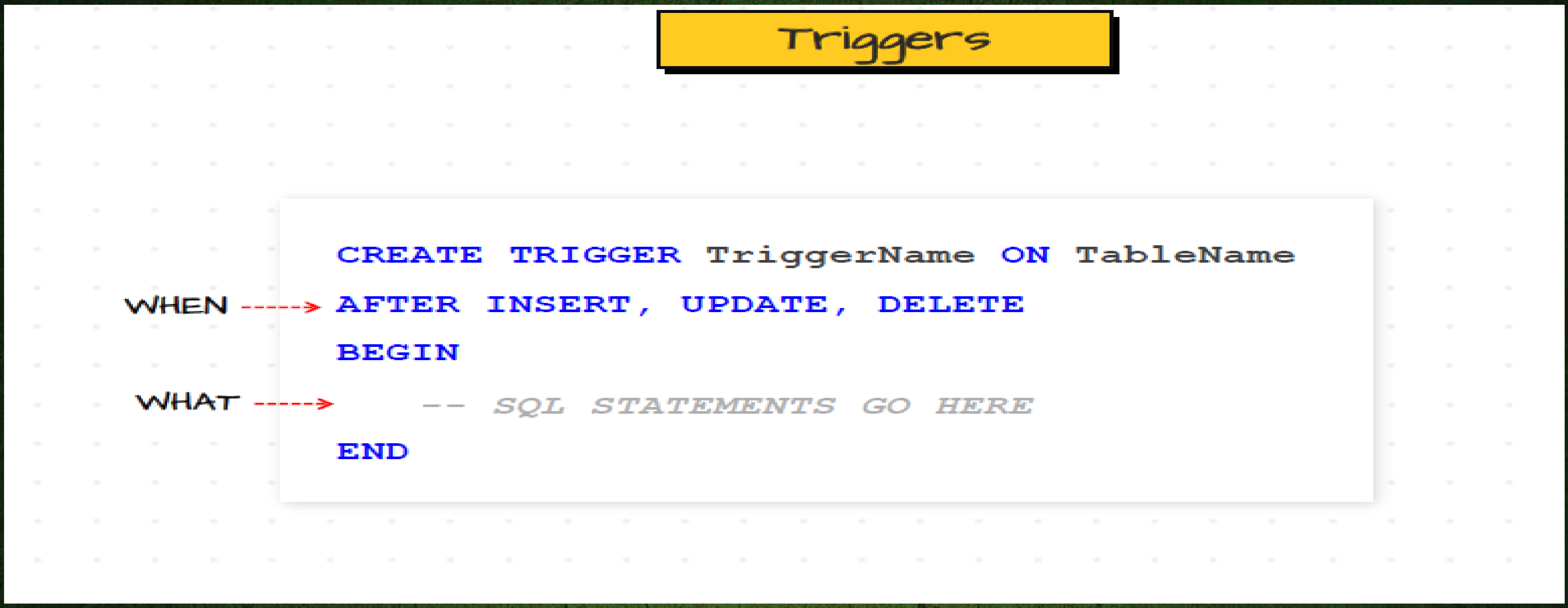
SQL Triggers With 12 Most Used Queries

AnshLibrary

What is a Trigger

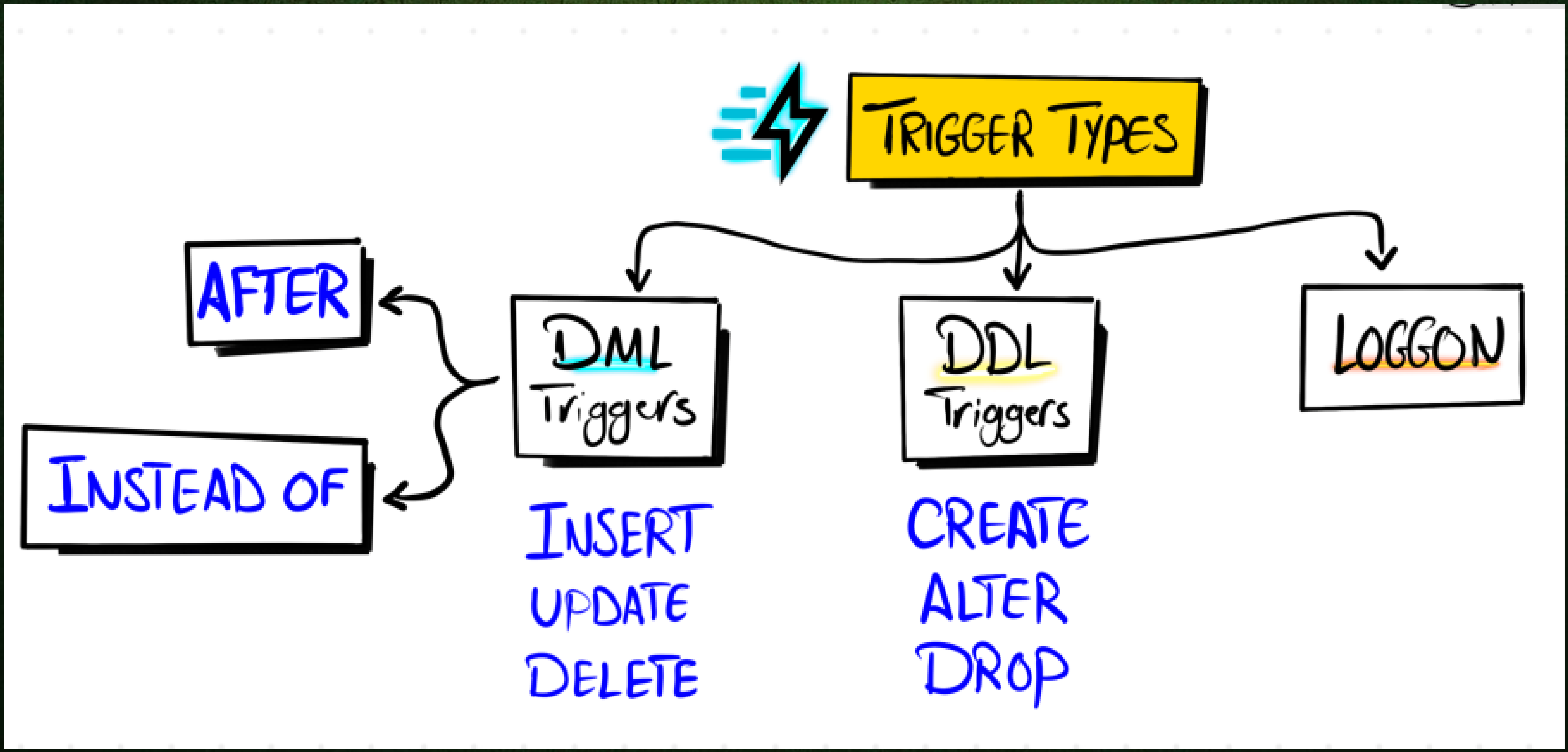
A trigger is a special kind of stored procedure that automatically executes when an event (INSERT, UPDATE, DELETE) occurs on a table or view.

Syntax :



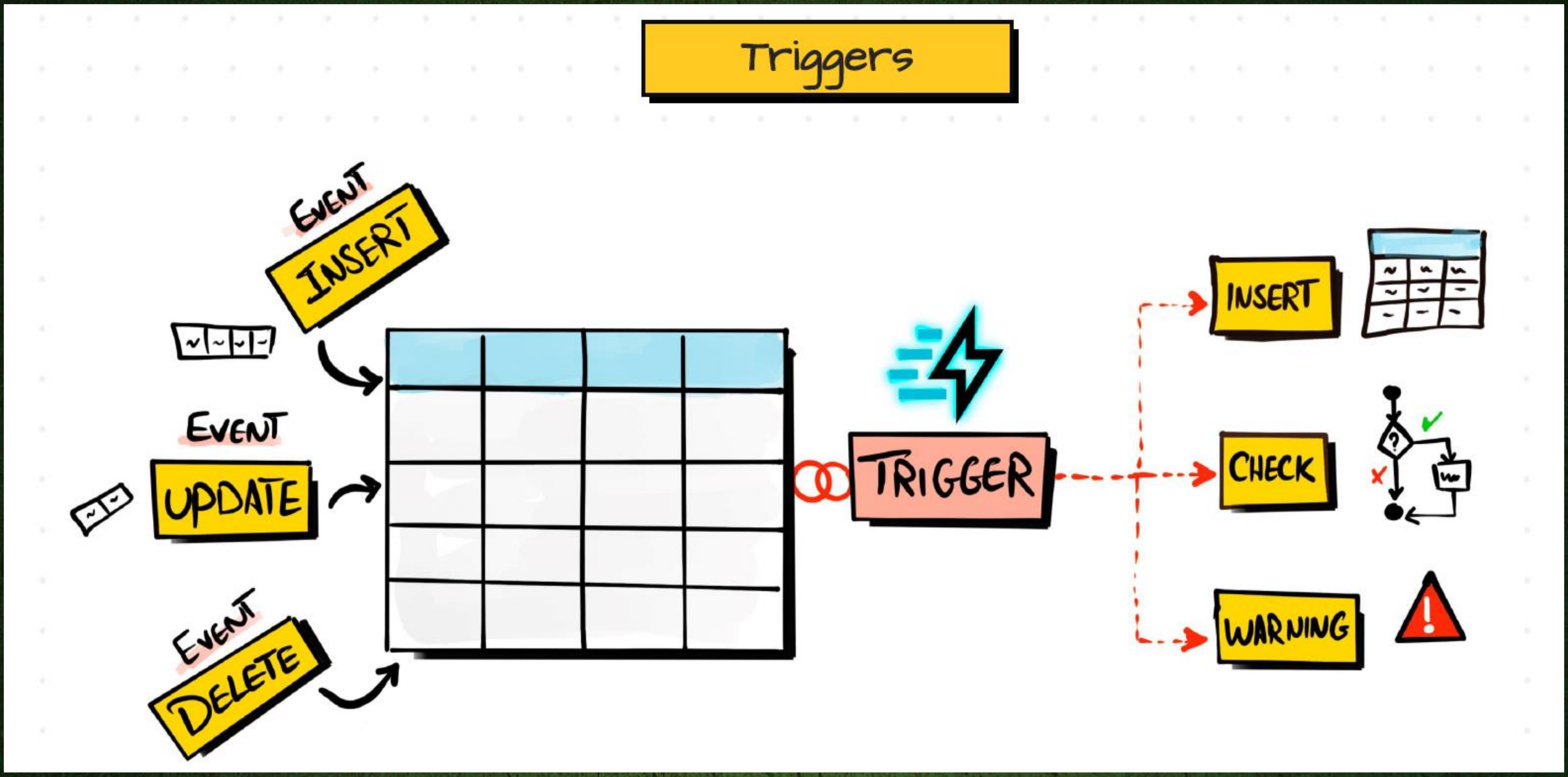
```
CREATE TRIGGER TriggerName ON TableName
AFTER INSERT, UPDATE, DELETE
BEGIN
    -- SQL STATEMENTS GO HERE
END
```

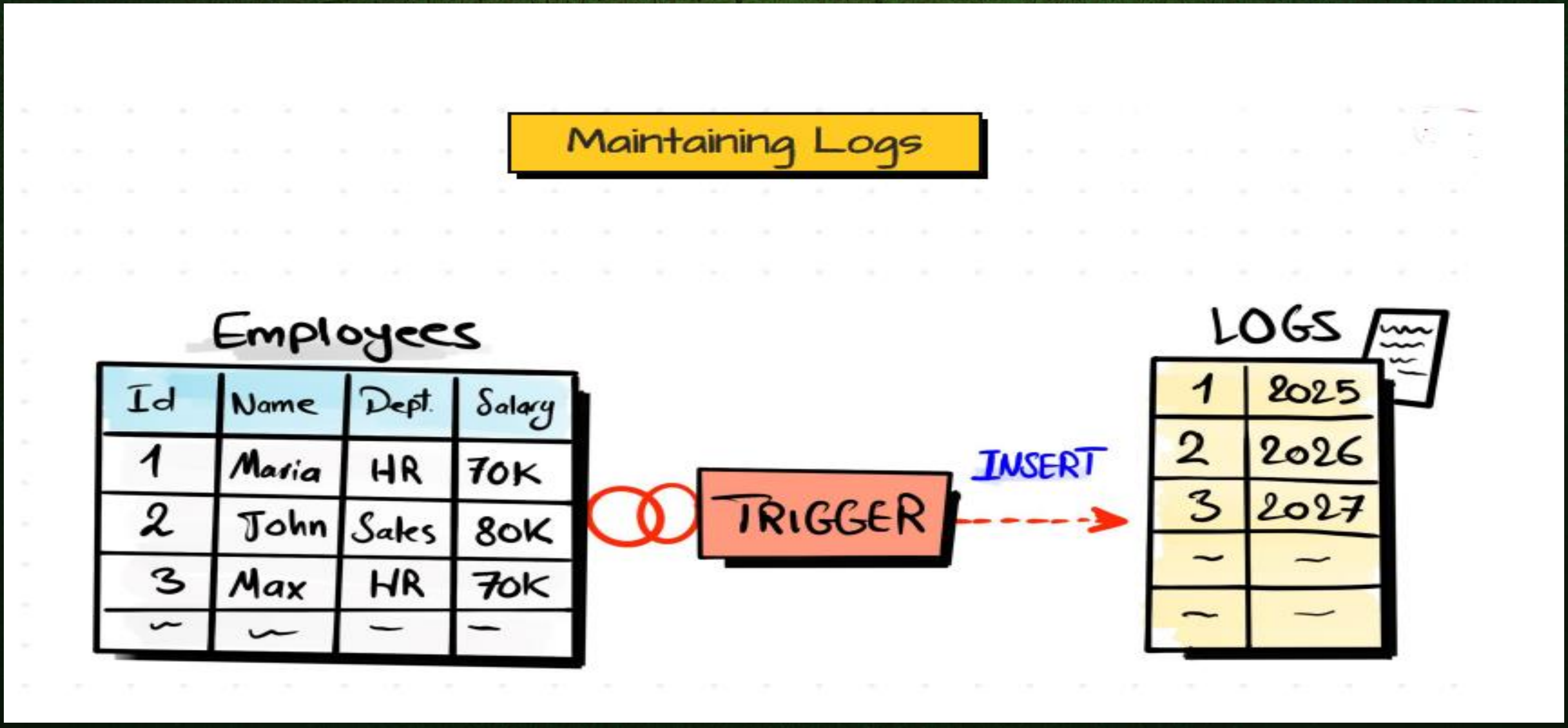
Trigger Types:



Why We Using :

- **Enforce business rules** automatically on data changes.
- **Log changes** for auditing purposes.
- **Validate data** before it's inserted or updated.
- **Send alerts or notifications** based on data events.
- **Automate repetitive tasks** tied to data changes.





Id	Name	Dept.	Salary
1	Maria	HR	70K
2	John	Sales	80K
3	Max	HR	70K
~	~	–	–

1	2025
2	2026
3	2027
~	~
~	~

01

Audit Log After New Order Insert

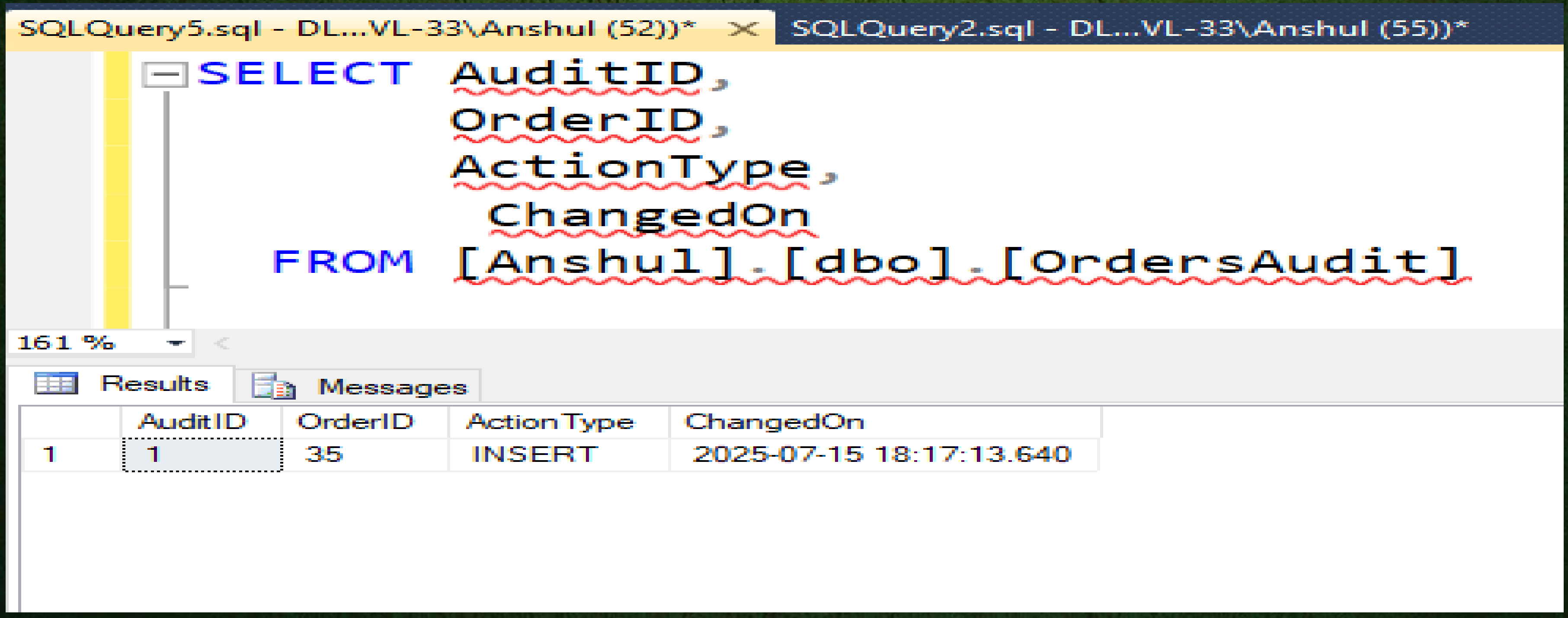
--Created Table First Like This

```
CREATE TABLE OrdersAudit (
    AuditID INT IDENTITY(1,1) PRIMARY KEY,
    OrderID INT,
    ActionType VARCHAR(10),
    ChangedOn DATETIME DEFAULT GETDATE()
);
```

--Create Trigger With This Select Query

```
CREATE TRIGGER trg_Audit_Order_Insert
ON Orders
AFTER INSERT
AS
BEGIN
    INSERT INTO OrdersAudit (OrderID, ActionType)
    SELECT OrderID, 'INSERT' FROM inserted;
END;
```

--Once you insert any values into Orders table then you able to find that records in this OrdersAudit table


SQLQuery5.sql - DL...VL-33\Anshul (52))*
SQLQuery2.sql - DL...VL-33\Anshul (55))*
161 %
Results
Messages

```
SELECT AuditID,
       OrderID,
       ActionType,
        ChangedOn
  FROM [Anshul].[dbo].[OrdersAudit]
```

```
   AuditID  OrderID  ActionType  ChangedOn
1  1        35       INSERT      2025-07-15 18:17:13.640
```

Use Case: Maintain audit log when new orders are added.

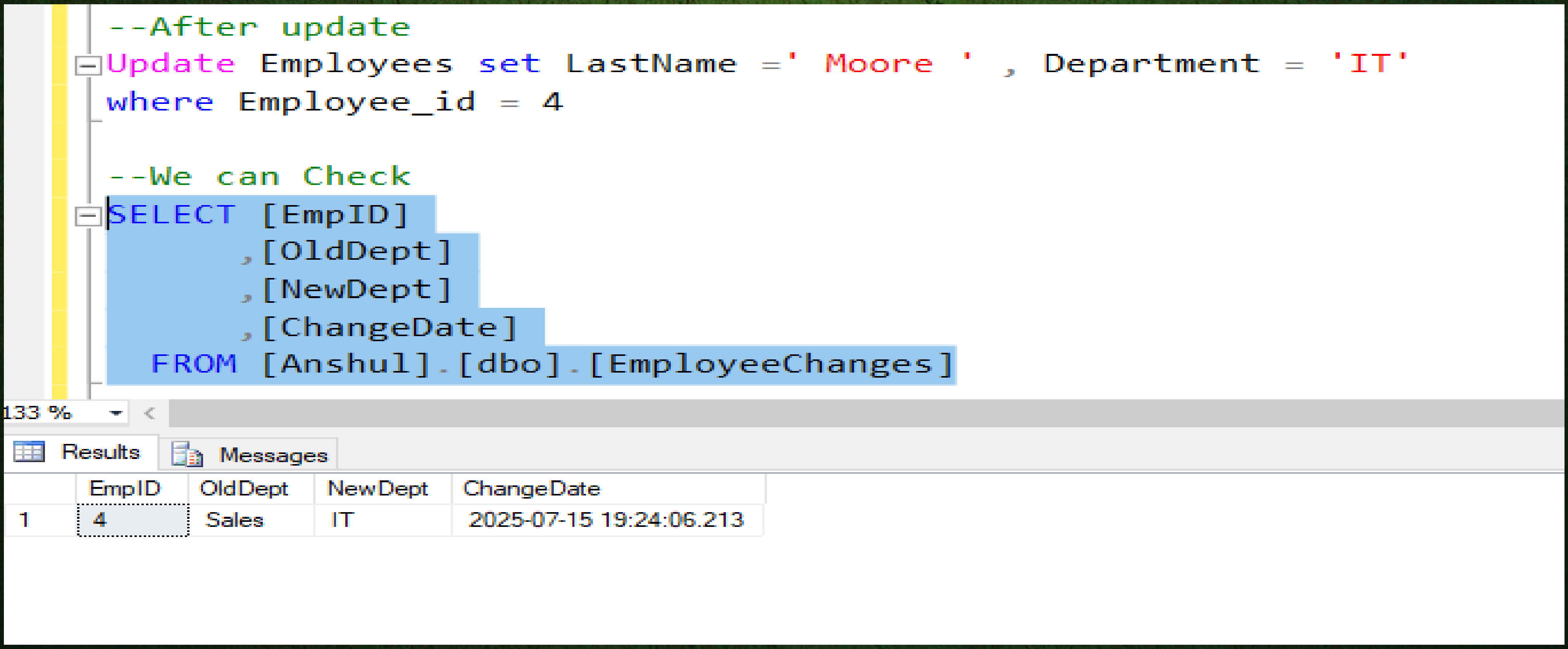
AnshLibrary

02

Track Employee Promotions (Department Changes)

```
CREATE TABLE EmployeeChanges (
    EmpID INT,
    OldDept VARCHAR(50),
    NewDept VARCHAR(50),
    ChangeDate DATETIME DEFAULT GETDATE()
);

CREATE TRIGGER trg_Employee_DeptChange
ON Employees
AFTER UPDATE
AS
BEGIN
    INSERT INTO EmployeeChanges (EmpID, OldDept, NewDept)
    SELECT d.Employee_id, d.Department, i.Department
    FROM deleted d
    JOIN inserted i ON d.Employee_id = i.Employee_id
    WHERE d.Department <> i.Department;
END;
```



```
--After update
Update Employees set LastName =' Moore ' , Department = 'IT'
where Employee_id = 4

--We can Check
SELECT [EmpID]
      ,[OldDept]
      ,[NewDept]
      ,[ChangeDate]
  FROM [Anshul].[dbo].[EmployeeChanges]
```

133 %

Results | Messages

	EmpID	OldDept	NewDept	ChangeDate
1	4	Sales	IT	2025-07-15 19:24:06.213

Use Case: Track internal movement in HR systems

AnshLibrary

03

Prevent Orders with NULL Shipping Address

```
CREATE TRIGGER trg_Prevent_Null_ShipAddress
ON Orders
AFTER INSERT, UPDATE
AS
BEGIN
    IF EXISTS (SELECT 1 FROM inserted WHERE
ShipAddress IS NULL)
    BEGIN
        RAISERROR ('ShipAddress cannot be NULL', 16,
1);
        ROLLBACK;
    END
END;
```

Use Case: Enforce data quality

AnshLibrary

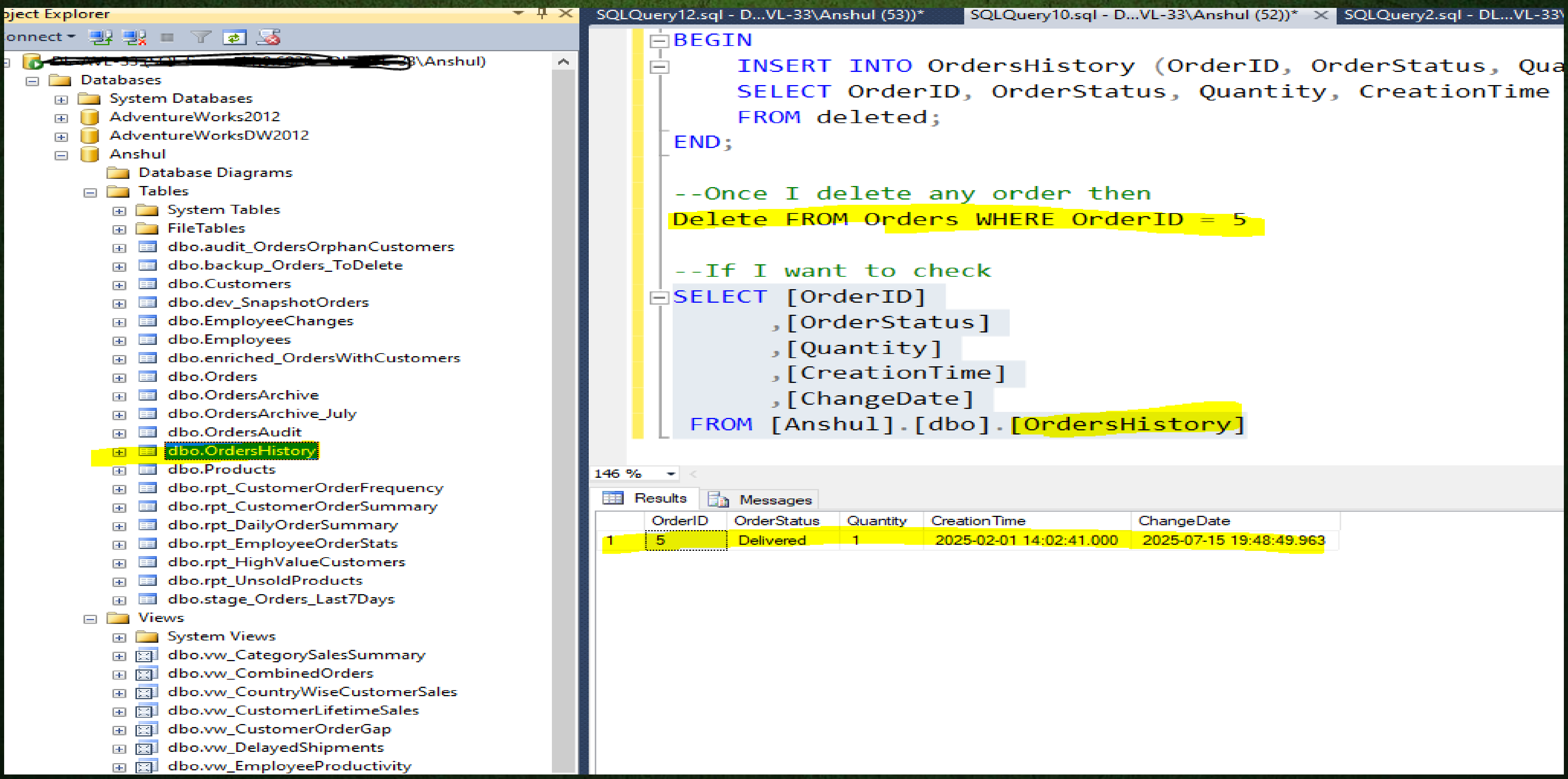
04

Auto-Archive Orders on DELETE

```sql
CREATE TABLE OrdersHistory (
    OrderID INT,
    OrderStatus VARCHAR(50),
    Quantity INT,
    CreationTime DATETIME,
    ChangeDate DATETIME DEFAULT GETDATE()
);

CREATE TRIGGER trg_Archive_Orders_OnDelete
ON Orders
AFTER DELETE
AS
BEGIN
    INSERT INTO OrdersHistory (OrderID, OrderStatus, Quantity, CreationTime)
    SELECT OrderID, OrderStatus, Quantity, CreationTime
    FROM deleted;
END;

--Once I delete any row
Delete FROM Orders WHERE OrderID = 5
```



```
[redacted]\Anshul)
  Databases
    System Databases
    AdventureWorks2012
    AdventureWorksDW2012
    Anshul
      Database Diagrams
      Tables
        System Tables
        FileTables
        dbo.audit_OrdersOrphanCustomers
        dbo.backup_Orders_ToDelete
        dbo.Customers
        dbo.dev_SnapshotOrders
        dbo.EmployeeChanges
        dbo.Employees
        dbo.enriched_OrdersWithCustomers
        dbo.Orders
        dbo.OrdersArchive
        dbo.OrdersArchive_July
        dbo.OrdersAudit
        dbo.OrdersHistory
        dbo.Products
        dbo.rpt_CustomerOrderFrequency
        dbo.rpt_CustomerOrderSummary
        dbo.rpt_DailyOrderSummary
        dbo.rpt_EmployeeOrderStats
        dbo.rpt_HighValueCustomers
        dbo.rpt_UnsoldProducts
        dbo.stage_Orders_Last7Days
      Views
        System Views
        dbo.vw_CategorySalesSummary
        dbo.vw_CombinedOrders
        dbo.vw_CountryWiseCustomerSales
        dbo.vw_CustomerLifetimeSales
        dbo.vw_CustomerOrderGap
        dbo.vw_DelayedShipments
        dbo.vw_EmployeeProductivity
```

```sql
BEGIN
    INSERT INTO OrdersHistory (OrderID, OrderStatus, Qua
    SELECT OrderID, OrderStatus, Quantity, CreationTime
    FROM deleted;
END;

--Once I delete any order then
Delete FROM Orders WHERE OrderID = 5

--If I want to check
SELECT [OrderID]
      ,[OrderStatus]
      ,[Quantity]
      ,[CreationTime]
      ,[ChangeDate]
 FROM [Anshul].[dbo].[OrdersHistory]
```

```
   OrderID  OrderStatus  Quantity  CreationTime             ChangeDate
1  5        Delivered    1         2025-02-01 14:02:41.000  2025-07-15 19:48:49.963
```

Use Case: Keep deleted orders archived for historical access.

AnshLibrary

05

Restrict Deleting Managers with Active Reportees

```
CREATE TRIGGER trg_Prevent_Manager_Delete
ON Employees
INSTEAD OF DELETE
AS
BEGIN
    IF EXISTS (
        SELECT 1 FROM Employees e
        JOIN deleted d ON e.ManagerID =
d.Employee_id
    )
    BEGIN
        RAISERROR ('Cannot delete manager with
active reportees.', 16, 1);
        RETURN;
    END
    ELSE
    BEGIN
        DELETE FROM Employees WHERE Employee_id IN
(SELECT Employee_id FROM deleted);
    END
END;
```

Used Case : Prevent accidental deletion of hierarchy.

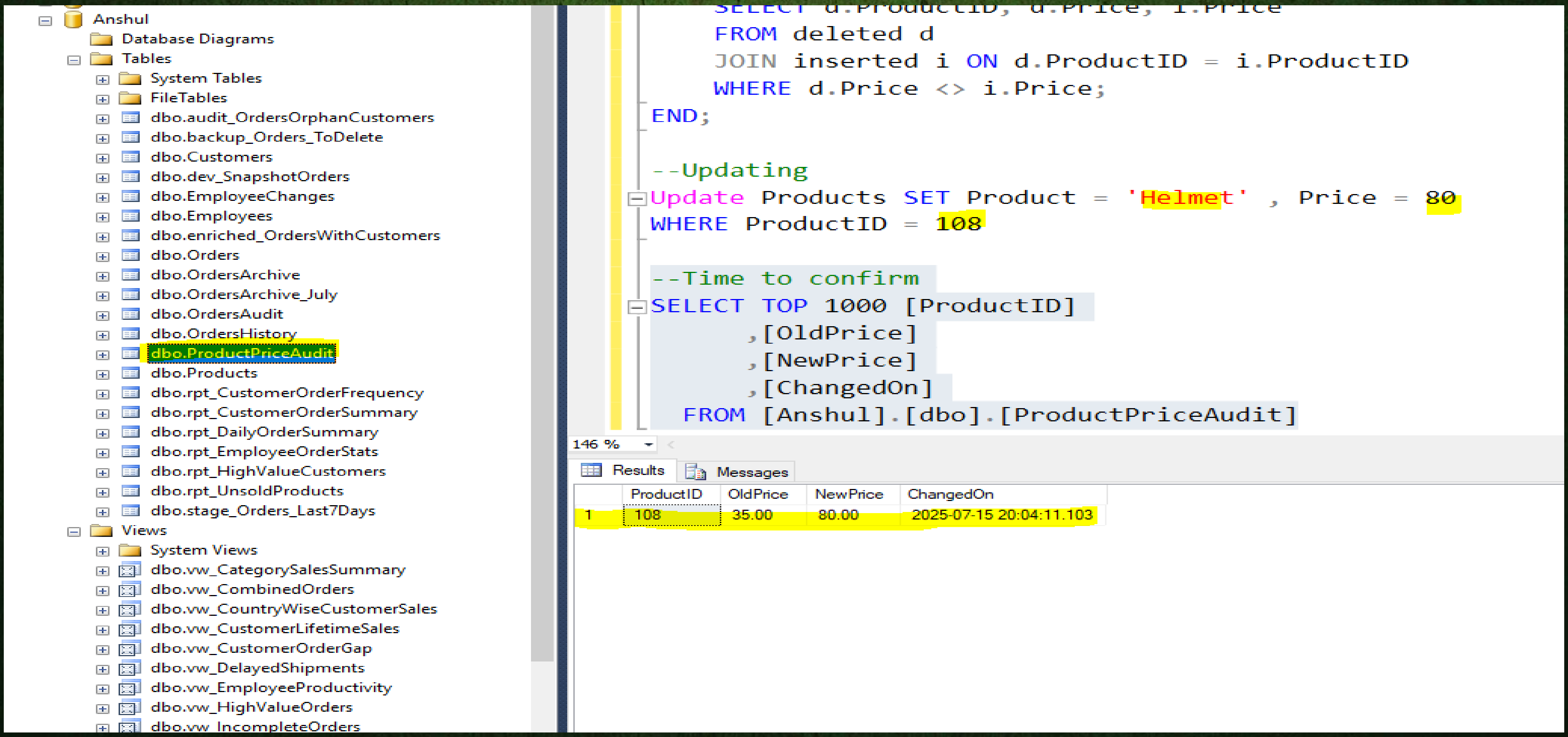
AnshLibrary

06

Log Product Price Changes

```sql
CREATE TABLE ProductPriceAudit (
    ProductID INT,
    OldPrice DECIMAL(10,2),
    NewPrice DECIMAL(10,2),
    ChangedOn DATETIME DEFAULT GETDATE()
);

CREATE TRIGGER trg_ProductPriceChange
ON Products
AFTER UPDATE
AS
BEGIN
    INSERT INTO ProductPriceAudit (ProductID, OldPrice, NewPrice)
    SELECT d.ProductID, d.Price, i.Price
    FROM deleted d
    JOIN inserted i ON d.ProductID = i.ProductID
    WHERE d.Price <> i.Price;
END;
```



```
Anshul
  Database Diagrams
  Tables
    System Tables
    FileTables
    dbo.audit_OrdersOrphanCustomers
    dbo.backup_Orders_ToDelete
    dbo.Customers
    dbo.dev_SnapshotOrders
    dbo.EmployeeChanges
    dbo.Employees
    dbo.enriched_OrdersWithCustomers
    dbo.Orders
    dbo.OrdersArchive
    dbo.OrdersArchive_July
    dbo.OrdersAudit
    dbo.OrdersHistory
    dbo.ProductPriceAudit
    dbo.Products
    dbo.rpt_CustomerOrderFrequency
    dbo.rpt_CustomerOrderSummary
    dbo.rpt_DailyOrderSummary
    dbo.rpt_EmployeeOrderStats
    dbo.rpt_HighValueCustomers
    dbo.rpt_UnsoldProducts
    dbo.stage_Orders_Last7Days
  Views
    System Views
    dbo.vw_CategorySalesSummary
    dbo.vw_CombinedOrders
    dbo.vw_CountryWiseCustomerSales
    dbo.vw_CustomerLifetimeSales
    dbo.vw_CustomerOrderGap
    dbo.vw_DelayedShipments
    dbo.vw_EmployeeProductivity
    dbo.vw_HighValueOrders
    dbo.vw_IncompleteOrders
```

```sql
    SELECT d.ProductID, d.Price, i.Price
    FROM deleted d
    JOIN inserted i ON d.ProductID = i.ProductID
    WHERE d.Price <> i.Price;
END;

--Updating
Update Products SET Product = 'Helmet' , Price = 80
WHERE ProductID = 108

--Time to confirm
SELECT TOP 1000 [ProductID]
      ,[OldPrice]
      ,[NewPrice]
      ,[ChangedOn]
  FROM [Anshul].[dbo].[ProductPriceAudit]
```

146 %

Results | Messages

	ProductID	OldPrice	NewPrice	ChangedOn
1	108	35.00	80.00	2025-07-15 20:04:11.103

Use Case: Track pricing history for audit.

AnshLibrary

07

Prevent Orders for Discontinued Products

```sql
CREATE TRIGGER trg_Discontinued_Product
ON Orders
AFTER INSERT
AS
BEGIN
    IF EXISTS (
        SELECT 1 FROM inserted i
        LEFT JOIN Products p
        ON i.ProductID = p.ProductID
        WHERE p.Product IS NULL
    )
    BEGIN
        RAISERROR ('Product does not exist or
is discontinued.', 16, 1);
        ROLLBACK;
    END
END;
```

Use Case: Block invalid SKUs..

AnshLibrary

08
Sync New Customers to CRM Table

```sql
CREATE TABLE CRM_Customers (
    CustomerID INT,
    FullName VARCHAR(100),
    Country VARCHAR(50),
    Score INT
);

CREATE TRIGGER trg_Sync_New_Customer
ON Customers
AFTER INSERT
AS
BEGIN
    INSERT INTO CRM_Customers (CustomerID,
FullName, Country, Score)
    SELECT Customer_id, First_Name + ' ' +
ISNULL(Last_Name, ''), Country, Score
    FROM inserted;
END;
```

Use Case: Real-time CRM sync on new customer creation.

AnshLibrary

Prevent Sales Over ANY AMOUNT in One Order

```
CREATE TRIGGER trg_Limit_High_Sales
ON Orders
AFTER INSERT
AS
BEGIN
    IF EXISTS (SELECT 1 FROM inserted WHERE
Sales > 150)
    BEGIN
        RAISERROR ('Sales exceeds allowed
transaction limit.', 16, 1);
        ROLLBACK;
    END
END;
```

Use Case: Fraud detection or compliance enforcement.

10

Log Failed Shipments

```
CREATE TABLE FailedShipments (
    OrderID INT,
    ShipStatus VARCHAR(50),
    LoggedOn DATETIME DEFAULT GETDATE()
);

CREATE TRIGGER trg_Log_Failed_Shipment
ON Orders
AFTER UPDATE
AS
BEGIN
    INSERT INTO FailedShipments (OrderID, 
ShipStatus)
    SELECT OrderID, OrderStatus FROM inserted
    WHERE OrderStatus = 'Failed';
END;
```

Use Case: Monitor logistics failure for escalation.

AnshLibrary

11

Track Customer Score Updates

```
CREATE TABLE CustomerScoreAudit (
    CustomerID INT,
    OldScore INT,
    NewScore INT,
    ChangedOn DATETIME DEFAULT GETDATE()
);

CREATE TRIGGER trg_CustomerScoreAudit
ON Customers
AFTER UPDATE
AS
BEGIN
    INSERT INTO CustomerScoreAudit (CustomerID,
OldScore, NewScore)
    SELECT d.Customer_id, d.Score, i.Score
    FROM deleted d
    JOIN inserted i
    ON d.Customer_id = i.Customer_id
    WHERE d.Score <> i.Score;
END;
```

Use Case: Record credit score history.

AnshLibrary

Auto Update OrderStatus on High Quantity

```
CREATE TRIGGER trg_HighQty_AutoHold
ON Orders
AFTER INSERT
AS
BEGIN
    UPDATE o
    SET OrderStatus = 'Hold'
    FROM Orders o
    JOIN inserted i
    ON o.OrderID = i.OrderID
    WHERE i.Quantity >= 10;
END;
```

Use Case: Flag bulk orders for manual review.

AnshLibrary