Development Templates

This document combines template development including stored procedures, functions, views, queries, error handling, and parameter documentation. It serves as a single master reference for projects.

**SQL Version Control**

a). Initialize a Git Repository

b). Create a folder for SQL scripts

c). mkdir TNSF

cd TNSF

git init

d). organize scripts

V1\_\_create\_users\_table.sql

V2\_\_add\_email\_column.sql

V3\_\_create\_orders\_table.sql

e). Track changes

git add “V1\_\_create\_users\_table.sql”

git commit -m “V1\_\_create\_users\_table.sql”

**CTE (Common Table Expression)** is a temporary

Makes queries **easier to read and maintain**.

Useful for breaking complex queries into smaller parts.

Can be **recursive** (used for hierarchies).

WITH SalesCTE AS (

SELECT CustomerID, SUM(Sales) AS TotalSales

FROM Orders

GROUP BY CustomerID

)

SELECT CustomerID, TotalSales

FROM SalesCTE

WHERE TotalSales > 5000;

**Naming Conventions**

**Columns Use singular, descriptive names like CustomerID, OrderID**

Object Type Convention

Database SalesDB

Tables OrderDetails

Views vw\_CustomerOrders

Stored Procedures sp\_GetCustomerSales

Functions fn\_CalcDiscount

Indexes IX\_Orders\_OrderDate

Primary Key PK\_Customers

Foreign Key FK\_Orders\_Customers

**Database & Schema**

CREATE DATABASE SalesDB;

CREATE SCHEMA Sales;

**Tables**

CREATE TABLE Sales.Customers (

CustomerID INT PRIMARY KEY,

FirstName VARCHAR(50) NOT NULL,

LastName VARCHAR(50) NOT NULL,

Email VARCHAR(100) UNIQUE,

CreatedDate DATETIME DEFAULT GETDATE()

);

**Data Manipulation (DML)**

INSERT INTO Sales.Customers (CustomerID, FirstName, LastName, Email)

VALUES (1, 'Apple', 'Green', 'green@gmail.com');

**Select & Joins**

SELECT o.OrderID, c.FirstName, o.TotalAmount

FROM Sales.Orders o

INNER JOIN Sales.Customers c ON o.CustomerID = c.CustomerID;

**Aggregation**

SELECT CustomerID, SUM(TotalAmount) AS TotalSales

FROM Sales.Orders

GROUP BY CustomerID;

**CTEs**

WITH OrderSummary AS (

SELECT CustomerID, SUM(TotalAmount) AS TotalSales

FROM Sales.Orders

GROUP BY CustomerID

)

SELECT \* FROM OrderSummary WHERE TotalSales > 5000;

# View Template

================================================

-- View Name : vw\_CustomerLatestOrder

-- Author : Name

-- Purpose : Get customers with their latest order

-- ================================================

CREATE VIEW vw\_CustomerLatestOrder

AS

WITH LatestOrder AS

(

SELECT

o.CustomerID,

MAX(o.OrderDate) AS LatestOrderDate

FROM Orders AS o

GROUP BY o.CustomerID

)

SELECT

c.CustomerID,

c.CustomerName,

lo.LatestOrderDate

FROM Customers AS c

LEFT JOIN LatestOrder AS lo

ON c.CustomerID = lo.CustomerID;

2) Top Product By Site View

-- ================================================

-- View Name : vw\_TopProductsBySite

-- Purpose : Get Top Products per SiteWise

-- ================================================

CREATE VIEW vw\_TopProductsBySite

AS

WITH RankedPrice AS

(

SELECT

e.ProductID,

e.ProductName,

e.SiteID,

e.Cost,

ROW\_NUMBER() OVER

(

PARTITION BY e.SiteID

ORDER BY e.Cost DESC

) AS RowNum

FROM Products AS e

)

SELECT

SiteID,

ProductID,

ProductName,

Cost

FROM RankedPrice

WHERE RowNum <= 2;

# 1. Stored Procedure Template

/\*=========================================================  
 Project Name : <Project Name>  
 Module : <Module Name>  
 File Name : <ProcedureName>.sql  
 Author : <Your Name>  
 Create Date : <yyyy-mm-dd>  
 Purpose : <Purpose of the procedure>  
=========================================================\*/  
  
CREATE PROCEDURE dbo.<ProcedureName>  
 @Param1 INT,  
 @Param2 VARCHAR(50),  
 @OutputParam INT OUTPUT  
AS  
BEGIN  
 SET NOCOUNT ON;  
  
 BEGIN TRY  
 BEGIN TRANSACTION;  
  
 -- Example Business Logic  
 -- INSERT INTO TableName (Col1, Col2) VALUES (@Param1, @Param2);  
  
 -- Return ID  
 -- SET @OutputParam = SCOPE\_IDENTITY();  
  
 COMMIT TRANSACTION;  
 END TRY  
 BEGIN CATCH  
 IF @@TRANCOUNT > 0 ROLLBACK TRANSACTION;  
  
 INSERT INTO ErrorLog (ErrorMessage, ProcedureName, ErrorDate)  
 VALUES (ERROR\_MESSAGE(), OBJECT\_NAME(@@PROCID), GETDATE());  
  
 THROW;  
 END CATCH;  
END;

# 2. Function Template

/\*=========================================================  
 Function Name : dbo.fn\_GetFullName  
 Author : <Your Name>  
 Create Date : <yyyy-mm-dd>  
 Purpose : Returns concatenated full name  
=========================================================\*/  
  
CREATE FUNCTION dbo.fn\_GetFullName  
(  
 @FirstName VARCHAR(50),  
 @LastName VARCHAR(50)  
)  
RETURNS VARCHAR(100)  
AS  
BEGIN  
 RETURN LTRIM(RTRIM(@FirstName + ' ' + @LastName));  
END;  
  
-- Example Inline TVF  
CREATE FUNCTION dbo.fn\_GetActiveCustomers()  
RETURNS TABLE  
AS  
RETURN  
(  
 SELECT CustomerID, CustomerName, Email  
 FROM Customers  
 WHERE IsActive = 1  
);

# 3. View Template

/\*=========================================================  
 View Name : vw\_FtDailySummary  
 Author : Name(Eg)  
 Date : <yyyy-mm-dd>  
 Purpose : Summarized sales view for reporting  
=========================================================\*/  
  
CREATE VIEW dbo.vw\_FtDailySummary  
AS  
SELECT   
 c.CustomerName,  
 SUM(o.TotalAmount) AS TotalSales,  
 COUNT(o.OrderID) AS OrdersCount  
FROM Orders o  
JOIN Customers c ON o.CustomerID = c.CustomerID  
GROUP BY c.CustomerName;

# Query Template

/\*=========================================================  
 Query Purpose : Fetch sales summary by region  
 Author : <Your Name>  
 Date : <yyyy-mm-dd>  
=========================================================\*/  
  
WITH SalesCTE AS  
(  
 SELECT   
 r.RegionName,  
 SUM(o.TotalAmount) AS TotalSales  
 FROM Orders o  
 JOIN Regions r ON o.RegionID = r.RegionID  
 WHERE o.OrderDate >= '2025-01-01'  
 GROUP BY r.RegionName  
)  
SELECT \* FROM SalesCTE  
ORDER BY TotalSales DESC;

# 5. Product Table

CREATE TABLE Product  
(  
 ProductID INT IDENTITY(1,1) PRIMARY KEY,  
 ProductMessage NVARCHAR(4000),  
 ProcedureName NVARCHAR(255),  
 ProductDate DATETIME DEFAULT GETDATE()  
);

# 6. Parameter Documentation

/\* Parameters:  
 @CustomerID INT (IN) -> Unique ID of the customer  
 @OrderDate DATETIME (IN) -> Date of the order  
 @OrderID INT OUTPUT (OUT) -> Newly generated order ID  
\*/

/\*=========================================================  
 Project Name : <Project Name>  
 Module : <Module Name>  
 File Name : <ProcedureName>.sql  
 Author : <Your Name>  
 Create Date : <yyyy-mm-dd>  
 Purpose : <Purpose of the procedure>  
=========================================================\*/

-- Select specific columns with filters

SELECT

EmployeeID,

FirstName,

LastName,

Department,

Salary

FROM dbo.Employee

WHERE Department = 'IT'

ORDER BY Salary DESC;

/\*=========================================================  
 Project Name : <Project Name>  
 Module : <Module Name>  
 File Name : <ProcedureName>.sql  
 Author : <Your Name>  
 Create Date : <yyyy-mm-dd>  
 Purpose : <Purpose of the procedure>  
=========================================================\*/

-- Department-wise total salary

SELECT

DepartmentID,

SUM(Salary) AS TotalSalary,

AVG(Salary) AS AvgSalary

FROM dbo.Employee

GROUP BY DepartmentID;

/\*=========================================================  
 Project Name : <Project Name>  
 Module : <Module Name>  
 File Name : <ProcedureName>.sql  
 Author : <Your Name>  
 Create Date : <yyyy-mm-dd>  
 Purpose : <Purpose of the procedure>  
=========================================================\*/

-- Employees with above average salary

SELECT

EmployeeID,

FirstName,

Salary

FROM

dbo.Employee

WHERE Salary > (

SELECT AVG(Salary) FROM dbo.Employee

);

/\*=========================================================  
 Project Name : <Project Name>  
 Module : <Module Name>  
 File Name : <ProcedureName>.sql  
 Author : <Your Name>  
 Create Date : <yyyy-mm-dd>  
 Purpose : <Purpose of the procedure>  
=========================================================\*/

-- Top 3 highest paid employees per department

WITH RankedSalaries AS (

SELECT

EmployeeID,

DepartmentID,

Salary,

RANK() OVER (PARTITION BY DepartmentID ORDER BY Salary DESC) AS rnk

FROM dbo.Employee

)

SELECT \*

FROM RankedSalaries

WHERE rnk <= 3;

/\*=========================================================  
 Project Name : <Project Name>  
 Module : <Module Name>  
 File Name : <ProcedureName>.sql  
 Author : <Your Name>  
 Create Date : <yyyy-mm-dd>  
 Purpose : <Purpose of the procedure>  
=========================================================\*/

-- Insert new employee

INSERT INTO dbo.Employee (FirstName, LastName, DepartmentID, Salary)

VALUES ('John', 'Doe', 2, 65000);

/\*=========================================================  
 Project Name : <Project Name>  
 Module : <Module Name>  
 File Name : <ProcedureName>.sql  
 Author : <Your Name>  
 Create Date : <yyyy-mm-dd>  
 Purpose : <Purpose of the procedure>  
=========================================================\*/-

SELECT A.Col1, B.Col2

FROM dbo.TableA AS A

INNER JOIN dbo.TableB AS B ON A.ID = B.ID;

/\*=========================================================  
 Project Name : <Project Name>  
 Module : <Module Name>  
 File Name : <ProcedureName>.sql  
 Author : <Your Name>  
 Create Date : <yyyy-mm-dd>  
 Purpose : <Purpose of the procedure>  
=========================================================\*/-- Non-Clustered Index on one column

-----------------------------------------------------

CREATE NONCLUSTERED INDEX IX\_Employees\_LastName

ON dbo.Employees (LastName);

/\*=========================================================  
 Project Name : <Project Name>  
 Module : <Module Name>  
 File Name : <ProcedureName>.sql  
 Author : <Your Name>  
 Create Date : <yyyy-mm-dd>  
 Purpose : <Purpose of the procedure>  
=========================================================\*/

-- Clustered Index (defines physical order of data)

-- Usually on Primary Key

-----------------------------------------------------

CREATE CLUSTERED INDEX IX\_Employees\_EmployeeID

ON dbo.Employees (EmployeeID);

-----------------------------------