**P5 - PSM Implementation**

Each team is required to submit the SQL code necessary to meet the database requirements. Teams are expected to create the following database objects:

* At least 3 stored procedures with input and output parameters, transaction management, and error handling (e.g., using TRY...CATCH blocks).
* At least 3 views, often used for reporting purposes.
* At least 3 user-defined functions
* At least 1 DML trigger.
* Column data encryption.
* At least 3 non-clustered indexes.
* Data visualization using Power BI or Tableau.
* A graphical user interface (GUI) for CRUD operations is a plus (optional).

1. At least 3 stored procedures with input and output parameters.

Create a New Order and Return the OrderID

CREATE PROCEDURE CreateOrder

@CustomerID INT,

@PaymentID INT,

@TotalAmount DECIMAL(10,2),

@Status VARCHAR(50),

@OrderDate DATE,

@DeliveryType VARCHAR(50),

@NewOrderID INT OUTPUT

AS

BEGIN

IF NOT EXISTS (SELECT 1 FROM Customer WHERE CustomerID = @CustomerID)

BEGIN

RAISERROR ('Invalid CustomerID: Customer does not exist.', 16, 1);

RETURN;

END

IF NOT EXISTS (SELECT 1 FROM PaymentMethod WHERE PaymentID = @PaymentID AND CustomerID = @CustomerID)

BEGIN

RAISERROR ('Invalid PaymentID: Payment method does not belong to this customer.', 16, 1);

RETURN;

END

INSERT INTO [Order] (CustomerID, PaymentID, TotalAmount, Status, OrderDate, DeliveryType)

VALUES (@CustomerID, @PaymentID, @TotalAmount, @Status, @OrderDate, @DeliveryType);

SET @NewOrderID = SCOPE\_IDENTITY();

END;

—-------------Execution Sample---------------------

DECLARE @OrderID INT;

EXEC CreateOrder

@CustomerID = 1,

@PaymentID = 2,

@TotalAmount = 45.50,

@Status = 'Pending',

@OrderDate = '2025-03-21',

@DeliveryType = 'Standard',

@NewOrderID = @OrderID OUTPUT;

PRINT 'New Order ID: ' + CAST(@OrderID AS VARCHAR);

UpdateOrderStatus

CREATE PROCEDURE UpdateOrderStatus

@OrderID INT,

@NewStatus VARCHAR(50),

@Success BIT OUTPUT

AS

BEGIN

SET @Success = 0;

IF NOT EXISTS (SELECT 1 FROM [Order] WHERE OrderID = @OrderID)

BEGIN

RETURN;

END

IF @NewStatus NOT IN ('Pending', 'Completed', 'Cancelled')

BEGIN

RETURN;

END

UPDATE [Order]

SET Status = @NewStatus

WHERE OrderID = @OrderID;

SET @Success = 1;

END;

—-------------Execution Sample-----------------

DECLARE @Result BIT;

EXEC UpdateOrderStatus

@OrderID = 3,

@NewStatus = 'Completed',

@Success = @Result OUTPUT;

PRINT 'Success? ' + CAST(@Result AS VARCHAR);

GetCustomerTotalSpent

CREATE PROCEDURE GetCustomerTotalSpent

@CustomerID INT,

@TotalAmountSpent DECIMAL(10,2) OUTPUT

AS

BEGIN

SELECT @TotalAmountSpent = ISNULL(SUM(TotalAmount), 0)

FROM [Order]

WHERE CustomerID = @CustomerID;

END;

—-------------Execution Sample-----------------

DECLARE @TotalSpent DECIMAL(10,2);

EXEC GetCustomerTotalSpent 1, @TotalSpent OUTPUT;

PRINT 'Total spent by customer 1: ' + CAST(@TotalSpent AS VARCHAR);

----------------------insert Order and OrderItem simultaneously

CREATE PROCEDURE CreateOrderWithItems

@CustomerID INT,

@PaymentID INT,

@TotalAmount DECIMAL(10,2),

@Status VARCHAR(50),

@OrderDate DATE,

@DeliveryType VARCHAR(50),

@OrderID INT OUTPUT

AS

BEGIN

BEGIN TRY

BEGIN TRANSACTION;

-- Step 1: Insert into [Order]

INSERT INTO [Order] (CustomerID, PaymentID, TotalAmount, Status, OrderDate, DeliveryType)

VALUES (@CustomerID, @PaymentID, @TotalAmount, @Status, @OrderDate, @DeliveryType);

SET @OrderID = SCOPE\_IDENTITY(); -- Get new OrderID

-- Step 2: insert multipel OrderItem

INSERT INTO OrderItem (RestaurantID, OrderID, MenuItemID, Quantity, UnitPrice)

VALUES

(1, @OrderID, 3, 2, 5.99),

(2, @OrderID, 5, 1, 9.99);

COMMIT TRANSACTION;

END TRY

BEGIN CATCH

ROLLBACK TRANSACTION;

-- cast error message

DECLARE @ErrMsg NVARCHAR(4000) = ERROR\_MESSAGE();

RAISERROR('Transaction failed: %s', 16, 1, @ErrMsg);

END CATCH

END;

--------------------Cancel Order

CREATE PROCEDURE CancelOrder

@OrderID INT

AS

BEGIN

BEGIN TRY

BEGIN TRANSACTION;

-- Step 1: Make sure the Order is existed or not

IF NOT EXISTS (SELECT 1 FROM [Order] WHERE OrderID = @OrderID)

BEGIN

RAISERROR('Order with ID %d does not exist.', 16, 1, @OrderID);

ROLLBACK TRANSACTION;

RETURN;

END

-- Step 2: Update Order status

UPDATE [Order]

SET Status = 'Cancelled'

WHERE OrderID = @OrderID;

-- Step 3: delete Deliver

DELETE FROM Delivery

WHERE OrderID = @OrderID;

COMMIT TRANSACTION;

END TRY

BEGIN CATCH

ROLLBACK TRANSACTION;

DECLARE @ErrMsg NVARCHAR(4000) = ERROR\_MESSAGE();

RAISERROR('Order cancelation failed %s', 16, 1, @ErrMsg);

END CATCH

END;

--------------------Ｕpdate Review

CREATE PROCEDURE UpdateReview

@ReviewID INT,

@NewRating INT,

@NewComment VARCHAR(255)

AS

BEGIN

BEGIN TRY

BEGIN TRANSACTION;

-- Check if this Review is existed or not

IF NOT EXISTS (SELECT 1 FROM Review WHERE ReviewID = @ReviewID)

BEGIN

RAISERROR('Review with ID %d does not exist.', 16, 1, @ReviewID);

ROLLBACK TRANSACTION;

RETURN;

END

-- Check if the Rating is legal(1~5)

IF @NewRating < 1 OR @NewRating > 5

BEGIN

RAISERROR('Rating must be between 1 and 5.', 16, 1);

ROLLBACK TRANSACTION;

RETURN;

END

-- Update

UPDATE Review

SET

Rating = @NewRating,

Comment = @NewComment,

ReviewDate = GETDATE()

WHERE ReviewID = @ReviewID;

COMMIT TRANSACTION;

END TRY

BEGIN CATCH

ROLLBACK TRANSACTION;

DECLARE @ErrMsg NVARCHAR(4000) = ERROR\_MESSAGE();

RAISERROR('Review Updating failed：%s', 16, 1, @ErrMsg);

END CATCH

END;

—------------CreateSimpleOrder----------------

CREATE PROCEDURE CreateSimpleOrder

@CustomerID INT,

@MenuItemID INT,

@Quantity INT,

@OrderID INT OUTPUT

AS

BEGIN

BEGIN TRY

BEGIN TRANSACTION;

-- 🔍 Step 1: Check if MenuItem exists

IF NOT EXISTS (SELECT 1 FROM MenuItem WHERE MenuItemID = @MenuItemID)

BEGIN

RAISERROR('Invalid MenuItemID', 16, 1);

ROLLBACK TRANSACTION;

RETURN;

END

-- 🔍 Step 2: Get first available PaymentID for the Customer

DECLARE @PaymentID INT;

SELECT TOP 1 @PaymentID = PaymentID FROM PaymentMethod WHERE CustomerID = @CustomerID;

IF @PaymentID IS NULL

BEGIN

RAISERROR('No payment method found for this customer.', 16, 1);

ROLLBACK TRANSACTION;

RETURN;

END

-- 💰 Step 3: Get Unit Price and calculate total

DECLARE @UnitPrice DECIMAL(10,2), @TotalAmount DECIMAL(10,2);

SELECT @UnitPrice = Price FROM MenuItem WHERE MenuItemID = @MenuItemID;

SET @TotalAmount = @UnitPrice \* @Quantity;

-- 🧾 Step 4: Insert into [Order]

INSERT INTO [Order] (CustomerID, PaymentID, TotalAmount, Status, OrderDate, DeliveryType)

VALUES (@CustomerID, @PaymentID, @TotalAmount, 'Pending', GETDATE(), 'Standard');

SET @OrderID = SCOPE\_IDENTITY();

-- 🍽️ Step 5: Insert into OrderItem

INSERT INTO OrderItem (RestaurantID, OrderID, MenuItemID, Quantity, UnitPrice)

VALUES (1, @OrderID, @MenuItemID, @Quantity, @UnitPrice); -- ← RestaurantID

COMMIT TRANSACTION;

END TRY

BEGIN CATCH

ROLLBACK TRANSACTION;

DECLARE @Err NVARCHAR(4000) = ERROR\_MESSAGE();

RAISERROR('CreateSimpleOrder failed: %s', 16, 1, @Err);

END CATCH

END;

—----------------------SAMPLE EXECUTION—-------------------

DECLARE @OrderID INT;

EXEC CreateSimpleOrder

@CustomerID = 1,

@MenuItemID = 5,

@Quantity = 2,

@OrderID = @OrderID OUTPUT;

SELECT 'New OrderID:' AS Info, @OrderID AS OrderID;

1. At least 3 views, often used for reporting purposes.

-- View 1: Customer Order History

CREATE OR ALTER VIEW CustomerOrderHistory AS

SELECT

c.CustomerID,

c.CustomerFirstName + ' ' + c.CustomerLastName AS FullName,

o.OrderID,

o.OrderDate,

o.Total\_Amount,

o.Status

FROM [Order] o

JOIN Customer c ON o.CustomerID = c.CustomerID;

GO

-- View 2: Restaurant Performance

CREATE OR ALTER VIEW RestaurantPerformance AS

SELECT

r.RestaurantID,

r.Name AS RestaurantName,

r.CuisineType,

COUNT(DISTINCT o.OrderID) AS TotalOrders,

SUM(o.Total\_Amount) AS TotalRevenue,

AVG(r2.Rating) AS AverageRating,

COUNT(DISTINCT r2.ReviewID) AS ReviewCount

FROM Restaurant r

LEFT JOIN OrderItem oi ON r.RestaurantID = oi.RestaurantID

LEFT JOIN [Order] o ON oi.OrderID = o.OrderID

LEFT JOIN Review r2 ON r.RestaurantID = r2.RestaurantID

GROUP BY r.RestaurantID, r.Name, r.CuisineType;

GO

-- View 3: Delivery Analytics

CREATE OR ALTER VIEW DeliveryAnalytics AS

SELECT

d.DriverID,

dr.DriverName,

COUNT(d.OrderID) AS TotalDeliveries,

AVG(DATEDIFF(MINUTE, d.PickupTime, d.ArrivalTime)) AS AvgDeliveryTimeMinutes,

SUM(CASE WHEN d.DeliveryStatus = 'Completed' THEN 1 ELSE 0 END) AS CompletedDeliveries,

SUM(CASE WHEN d.DeliveryStatus = 'Cancelled' THEN 1 ELSE 0 END) AS CancelledDeliveries,

SUM(o.Total\_Amount) AS TotalDeliveryRevenue

FROM Delivery d

JOIN Driver dr ON d.DriverID = dr.DriverID

JOIN [Order] o ON d.OrderID = o.OrderID

GROUP BY d.DriverID, dr.DriverName;

GO

1. At least 3 user-defined functions

CREATE FUNCTION GetTotalOrders(@CustomerID INT)

RETURNS INT

AS

BEGIN

DECLARE @TotalOrders INT;

SELECT @TotalOrders = COUNT(\*) FROM [Order] WHERE CustomerID = @CustomerID;

RETURN @TotalOrders;

END;



CREATE FUNCTION GetAverageRating(@RestaurantID INT)

RETURNS DECIMAL(3,2)

AS

BEGIN

DECLARE @AvgRating DECIMAL(3,2);

SELECT @AvgRating = AVG(CAST(Rating AS DECIMAL(3,2))) FROM Review

WHERE RestaurantID = @RestaurantID;

RETURN ISNULL(@AvgRating, 0);

END;

CREATE FUNCTION GetOrderTotalAmount()

RETURNS TABLE

AS

RETURN

(

SELECT o.OrderID, SUM(oi.Quantity \* oi.UnitPrice) AS TotalAmount

FROM [Order] o

JOIN OrderItem oi ON o.OrderID = oi.OrderID

GROUP BY o.OrderID

);

1. At least 1 DML trigger.

This trigger logs the order status changes into an audit table.

CREATE TABLE OrderAuditLog (

AuditID INT IDENTITY(1,1) PRIMARY KEY,

OrderID INT,

OldStatus VARCHAR(50),

NewStatus VARCHAR(50),

ChangedAt DATETIME DEFAULT GETDATE()

);

CREATE TRIGGER trg\_LogOrderStatusUpdate

ON [Order]

AFTER UPDATE

AS

BEGIN

IF UPDATE(Status)

BEGIN

INSERT INTO OrderAuditLog (OrderID, OldStatus, NewStatus)

SELECT

i.OrderID,

d.Status AS OldStatus,

i.Status AS NewStatus

FROM inserted i

JOIN deleted d ON i.OrderID = d.OrderID

WHERE i.Status <> d.Status;

END

END;

or

This trigger blocks changing a status from 'Completed' to any other.

CREATE TRIGGER trg\_PreventInvalidStatusChange

ON [Order]

INSTEAD OF UPDATE

AS

BEGIN

IF EXISTS (

SELECT 1

FROM inserted i

JOIN deleted d ON i.OrderID = d.OrderID

WHERE d.Status = 'Completed' AND i.Status <> 'Completed'

)

BEGIN

RAISERROR('Cannot change status from Completed to another status.', 16, 1);

ROLLBACK;

END

ELSE

BEGIN

UPDATE o

SET

o.TotalAmount = i.TotalAmount,

o.Status = i.Status,

o.OrderDate = i.OrderDate,

o.DeliveryType = i.DeliveryType,

o.CustomerID = i.CustomerID,

o.PaymentID = i.PaymentID

FROM [Order] o

JOIN inserted i ON o.OrderID = i.OrderID;

END

END;

1. Column data encryption.

IF NOT EXISTS (SELECT \* FROM sys.symmetric\_keys WHERE name = '##MS\_DatabaseMasterKey##')

BEGIN

CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'C0mpl3x\_M@ster\_Encryption\_Key\_DAMG2025!';

END

GO

IF NOT EXISTS (SELECT \* FROM sys.certificates WHERE name = 'FoodDeliveryCertificate')

BEGIN

CREATE CERTIFICATE FoodDeliveryCertificate

WITH SUBJECT = 'Food Delivery System Encryption Certificate',

EXPIRY\_DATE = '20340630';

END

GO

IF NOT EXISTS (SELECT \* FROM sys.symmetric\_keys WHERE name = 'FoodDeliverySymmetricKey')

BEGIN

CREATE SYMMETRIC KEY FoodDeliverySymmetricKey

WITH ALGORITHM = AES\_256

ENCRYPTION BY CERTIFICATE FoodDeliveryCertificate;

END

GO

ALTER TABLE Customer

ADD EncryptedPhoneNumber VARBINARY(MAX),

EncryptedEmail VARBINARY(MAX);

GO

ALTER TABLE PaymentMethod

ADD EncryptedCardNumber VARBINARY(MAX),

EncryptedExpDate VARBINARY(MAX);

GO

OPEN SYMMETRIC KEY FoodDeliverySymmetricKey

DECRYPTION BY CERTIFICATE FoodDeliveryCertificate;

UPDATE Customer

SET EncryptedPhoneNumber = EncryptByKey(Key\_GUID('FoodDeliverySymmetricKey'), PhoneNumber),

EncryptedEmail = EncryptByKey(Key\_GUID('FoodDeliverySymmetricKey'), Email);

UPDATE PaymentMethod

SET EncryptedCardNumber = EncryptByKey(Key\_GUID('FoodDeliverySymmetricKey'), CardNumber),

EncryptedExpDate = EncryptByKey(Key\_GUID('FoodDeliverySymmetricKey'), CONVERT(VARCHAR(10), ExpDate, 120));

CLOSE SYMMETRIC KEY FoodDeliverySymmetricKey;

GO

CREATE OR ALTER VIEW EncryptedCustomerView

AS

SELECT

CustomerID,

CustomerFirstName,

CustomerLastName,

EncryptedPhoneNumber,

EncryptedEmail,

Password

FROM Customer;

GO

CREATE OR ALTER VIEW EncryptedPaymentMethodView

AS

SELECT

PaymentID,

CustomerID,

EncryptedCardNumber,

EncryptedExpDate,

PaymentType

FROM PaymentMethod;

GO

CREATE OR ALTER PROCEDURE DecryptCustomerData

@CustomerID INT

AS

BEGIN

OPEN SYMMETRIC KEY FoodDeliverySymmetricKey

DECRYPTION BY CERTIFICATE FoodDeliveryCertificate;

SELECT

CustomerID,

CustomerFirstName,

CustomerLastName,

CONVERT(VARCHAR(15), DecryptByKey(EncryptedPhoneNumber)) AS DecryptedPhoneNumber,

CONVERT(VARCHAR(100), DecryptByKey(EncryptedEmail)) AS DecryptedEmail

FROM EncryptedCustomerView

WHERE CustomerID = @CustomerID;

CLOSE SYMMETRIC KEY FoodDeliverySymmetricKey;

END

GO

CREATE OR ALTER PROCEDURE DecryptPaymentMethodData

@PaymentID INT

AS

BEGIN

OPEN SYMMETRIC KEY FoodDeliverySymmetricKey

DECRYPTION BY CERTIFICATE FoodDeliveryCertificate;

SELECT

PaymentID,

CustomerID,

CONVERT(VARCHAR(16), DecryptByKey(EncryptedCardNumber)) AS DecryptedCardNumber,

CONVERT(DATE, CAST(DecryptByKey(EncryptedExpDate) AS VARCHAR(10))) AS DecryptedExpDate,

PaymentType

FROM EncryptedPaymentMethodView

WHERE PaymentID = @PaymentID;

CLOSE SYMMETRIC KEY FoodDeliverySymmetricKey;

END

GO

SELECT \* FROM EncryptedCustomerView;

SELECT \* FROM EncryptedPaymentMethodView;

-- Decrypt specific row

EXEC DecryptCustomerData @CustomerID = 4;

EXEC DecryptPaymentMethodData @PaymentID = 1;

1. At least 3 non-clustered indexes.
   1. IX\_Review\_ByRestaurant

CREATE NONCLUSTERED INDEX IX\_Review\_RestaurantID

ON Review(RestaurantID);

* 1. IX\_Order\_ByCustomer

CREATE NONCLUSTERED INDEX IX\_Order\_CustomerID

ON [Order](CustomerID);

* 1. IX\_Delivery\_ByDriver

CREATE NONCLUSTERED INDEX IX\_Delivery\_DriverID

ON Delivery(DriverID);

* 1. IX\_Delivery\_ByDriver

CREATE NONCLUSTERED INDEX IX\_Order\_ByStatus

ON [Order](Status);

1. Data visualization using Power BI or Tableau.
2. A graphical user interface (GUI) for CRUD operations is a plus (optional).