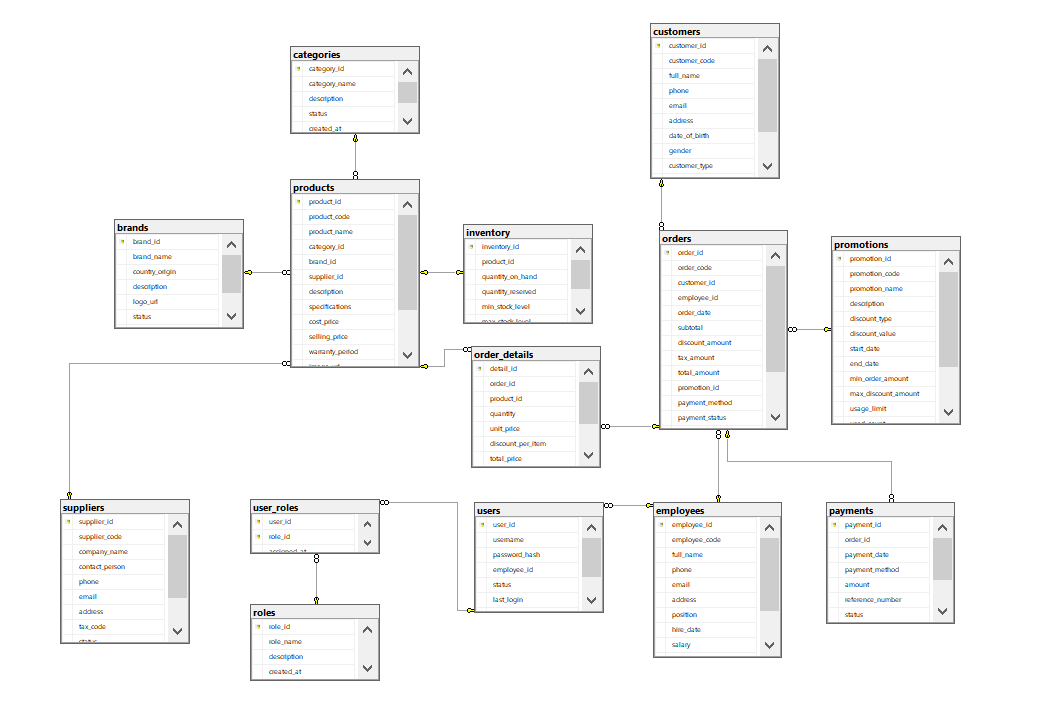
CHƯƠNG 1: TỔNG QUAN VỀ HỆ THỐNG

1. Đặc tả đề tài

1.1 Mô tả bài toán



CREATE PROCEDURE sp\_GetDashboardStats

AS

BEGIN

SET NOCOUNT ON;

DECLARE @today DATE = CAST(GETDATE() AS DATE);

DECLARE @month\_start DATE = DATEFROMPARTS(YEAR(GETDATE()), MONTH(GETDATE()), 1);

SELECT

-- Today's stats

COALESCE(SUM(CASE WHEN CAST(o.order\_date AS DATE) = @today AND o.order\_status = 'completed' THEN o.total\_amount END), 0) AS today\_revenue,

COALESCE(COUNT(CASE WHEN CAST(o.order\_date AS DATE) = @today AND o.order\_status = 'completed' THEN 1 END), 0) AS today\_orders,

-- Month stats

COALESCE(SUM(CASE WHEN CAST(o.order\_date AS DATE) >= @month\_start AND o.order\_status = 'completed' THEN o.total\_amount END), 0) AS month\_revenue,

COALESCE(COUNT(CASE WHEN CAST(o.order\_date AS DATE) >= @month\_start AND o.order\_status = 'completed' THEN 1 END), 0) AS month\_orders,

-- Other stats

COALESCE(AVG(CASE WHEN o.order\_status = 'completed' THEN o.total\_amount END), 0) AS avg\_order\_value,

(SELECT COUNT(\*) FROM customers) AS total\_customers,

(SELECT COUNT(\*) FROM inventory i WHERE i.quantity\_on\_hand <= i.min\_stock\_level) AS low\_stock\_products

FROM orders o;

END;

public DashboardStats GetDashboardStats()

{

string procedureName = "sp\_GetDashboardStats";

var dataTable = ExecuteQuery(procedureName, null, isStoredProcedure: true);

if (dataTable.Rows.Count > 0)

{

var row = dataTable.Rows[0];

return new DashboardStats

{

TodayRevenue = Convert.ToDecimal(row["today\_revenue"]),

TodayOrders = Convert.ToInt32(row["today\_orders"]),

MonthRevenue = Convert.ToDecimal(row["month\_revenue"]),

MonthOrders = Convert.ToInt32(row["month\_orders"]),

AverageOrderValue = Convert.ToDecimal(row["avg\_order\_value"]),

TotalCustomers = Convert.ToInt32(row["total\_customers"]),

LowStockProducts = Convert.ToInt32(row["low\_stock\_products"])

};

}

return new DashboardStats();

}

CREATE PROCEDURE sp\_GetCustomerByPhone

@phone NVARCHAR(20)

AS

BEGIN

SELECT c.\*,

COUNT(o.order\_id) AS total\_orders,

COALESCE(MAX(o.order\_date), c.created\_at) AS last\_order\_date

FROM customers c

LEFT JOIN orders o

ON c.customer\_id = o.customer\_id

AND o.order\_status = 'completed'

WHERE c.phone = @phone

GROUP BY

c.customer\_id, c.customer\_code, c.full\_name, c.phone, c.email,

c.address, c.date\_of\_birth, c.gender, c.customer\_type,

c.total\_spent, c.created\_at, c.updated\_at

END

public Customer GetCustomerByPhone(string phone)

{

var parameters = new SqlParameter[]

{

new SqlParameter("@phone", phone)

};

var dataTable = ExecuteQuery("sp\_GetCustomerByPhone", parameters, isStoredProcedure: true);

if (dataTable.Rows.Count > 0)

{

return MapRowToCustomer(dataTable.Rows[0]);

}

return null;

}

CREATE VIEW v\_AllActiveProducts

AS

SELECT

p.\*,

c.category\_name,

b.brand\_name,

COALESCE(i.quantity\_on\_hand, 0) AS quantity\_on\_hand

FROM products p

LEFT JOIN categories c ON p.category\_id = c.category\_id

LEFT JOIN brands b ON p.brand\_id = b.brand\_id

LEFT JOIN inventory i ON p.product\_id = i.product\_id

WHERE p.status = 'active';

public List<Product> GetAllProducts()

{

const string query = "SELECT \* FROM v\_AllActiveProducts";

var dataTable = ExecuteQuery(query);

var products = new List<Product>();

foreach (DataRow row in dataTable.Rows)

{

products.Add(MapRowToProduct(row));

}

return products;

}

CREATE PROCEDURE sp\_GetProductStock

@productId INT

AS

BEGIN

SELECT COALESCE(quantity\_on\_hand, 0)

FROM inventory

WHERE product\_id = @productId;

END

public int GetProductStock(int productId)

{

var parameters = new SqlParameter[]

{

new SqlParameter("@productId", productId)

};

var result = ExecuteScalar("sp\_GetProductStock", parameters, isStoredProcedure: true);

return result != null ? Convert.ToInt32(result) : 0;

}

CREATE PROCEDURE sp\_GetProductById

@productId INT

AS

BEGIN

SELECT

p.\*,

c.category\_name,

b.brand\_name,

COALESCE(i.quantity\_on\_hand, 0) AS quantity\_on\_hand

FROM products p

LEFT JOIN categories c ON p.category\_id = c.category\_id

LEFT JOIN brands b ON p.brand\_id = b.brand\_id

LEFT JOIN inventory i ON p.product\_id = i.product\_id

WHERE p.product\_id = @productId;

END

public Product GetProductById(int productId)

{

var parameters = new SqlParameter[]

{

new SqlParameter("@productId", productId)

};

var dataTable = ExecuteQuery("sp\_GetProductById", parameters, isStoredProcedure: true);

if (dataTable.Rows.Count > 0)

{

return MapRowToProduct(dataTable.Rows[0]);

}

return null;

}

CREATE FUNCTION fn\_GenerateOrderCode()

RETURNS NVARCHAR(50)

AS

BEGIN

DECLARE @prefix NVARCHAR(20) = 'ORD' + FORMAT(GETDATE(), 'yyyyMMdd');

DECLARE @nextNumber INT = (

SELECT COALESCE(MAX(CAST(RIGHT(order\_code, 4) AS INT)), 0) + 1

FROM orders

WHERE order\_code LIKE @prefix + '%'

);

DECLARE @code NVARCHAR(50) = @prefix + RIGHT('0000' + CAST(@nextNumber AS NVARCHAR(4)), 4);

RETURN @code;

END

public string GenerateOrderCode()

{

const string query = "SELECT dbo.fn\_GenerateOrderCode()";

var result = ExecuteScalar(query);

return result?.ToString() ?? "ORD" + DateTime.Now.ToString("yyyyMMdd") + "0001";

}

CREATE PROCEDURE sp\_InsertOrder

@order\_code NVARCHAR(50),

@customer\_id INT = NULL,

@employee\_id INT = NULL,

@order\_date DATETIME,

@subtotal DECIMAL(18,2),

@discount\_amount DECIMAL(18,2),

@tax\_amount DECIMAL(18,2),

@total\_amount DECIMAL(18,2),

@promotion\_id INT = NULL,

@payment\_method NVARCHAR(50),

@payment\_status NVARCHAR(50),

@order\_status NVARCHAR(50),

@notes NVARCHAR(MAX),

@order\_id INT OUTPUT

AS

BEGIN

SET NOCOUNT ON;

BEGIN TRY

BEGIN TRANSACTION;

INSERT INTO orders (

order\_code, customer\_id, employee\_id, order\_date,

subtotal, discount\_amount, tax\_amount, total\_amount,

promotion\_id, payment\_method, payment\_status, order\_status, notes

)

VALUES (

@order\_code, @customer\_id, @employee\_id, @order\_date,

@subtotal, @discount\_amount, @tax\_amount, @total\_amount,

@promotion\_id, @payment\_method, @payment\_status, @order\_status, @notes

);

SET @order\_id = SCOPE\_IDENTITY();

COMMIT;

END TRY

BEGIN CATCH

ROLLBACK;

THROW;

END CATCH

END

CREATE PROCEDURE sp\_InsertOrderDetail

@order\_id INT,

@product\_id INT,

@quantity INT,

@unit\_price DECIMAL(18,2),

@discount\_per\_item DECIMAL(18,2),

@total\_price DECIMAL(18,2)

AS

BEGIN

SET NOCOUNT ON;

BEGIN TRY

BEGIN TRANSACTION;

INSERT INTO order\_details (

order\_id, product\_id, quantity, unit\_price, discount\_per\_item, total\_price

)

VALUES (

@order\_id, @product\_id, @quantity, @unit\_price, @discount\_per\_item, @total\_price

);

COMMIT;

END TRY

BEGIN CATCH

ROLLBACK;

THROW;

END CATCH

END

public int InsertOrder(Order order, List<OrderDetail> orderDetails)

{

using (var conn = GetConnection())

{

conn.Open();

using (var transaction = conn.BeginTransaction())

{

try

{

// 1. Insert Order using stored procedure

using (var cmd = new SqlCommand("sp\_InsertOrder", conn, transaction))

{

cmd.CommandType = CommandType.StoredProcedure;

var orderIdParam = new SqlParameter("@order\_id", SqlDbType.Int)

{

Direction = ParameterDirection.Output

};

cmd.Parameters.AddRange(new SqlParameter[]

{

new SqlParameter("@order\_code", order.OrderCode),

new SqlParameter("@customer\_id", (object)order.CustomerId ?? DBNull.Value),

new SqlParameter("@employee\_id", (object)order.EmployeeId ?? DBNull.Value),

new SqlParameter("@order\_date", order.OrderDate),

new SqlParameter("@subtotal", order.Subtotal),

new SqlParameter("@discount\_amount", order.DiscountAmount),

new SqlParameter("@tax\_amount", order.TaxAmount),

new SqlParameter("@total\_amount", order.TotalAmount),

new SqlParameter("@promotion\_id", (object)order.PromotionId ?? DBNull.Value),

new SqlParameter("@payment\_method", order.PaymentMethod),

new SqlParameter("@payment\_status", order.PaymentStatus),

new SqlParameter("@order\_status", order.OrderStatus),

new SqlParameter("@notes", order.Notes ?? ""),

orderIdParam

});

cmd.ExecuteNonQuery();

int orderId = Convert.ToInt32(orderIdParam.Value);

// 2. Insert Order Details

foreach (var detail in orderDetails)

{

using (var detailCmd = new SqlCommand("sp\_InsertOrderDetail", conn, transaction))

{

detailCmd.CommandType = CommandType.StoredProcedure;

detailCmd.Parameters.AddRange(new SqlParameter[]

{

new SqlParameter("@order\_id", orderId),

new SqlParameter("@product\_id", detail.ProductId),

new SqlParameter("@quantity", detail.Quantity),

new SqlParameter("@unit\_price", detail.UnitPrice),

new SqlParameter("@discount\_per\_item", detail.DiscountPerItem),

new SqlParameter("@total\_price", detail.TotalPrice)

});

detailCmd.ExecuteNonQuery();

}

}

transaction.Commit();

return orderId;

}

}

catch

{

transaction.Rollback();

throw;

}

}

}

}

CREATE PROCEDURE sp\_GetOrderById

@orderId INT

AS

BEGIN

SELECT

o.\*,

c.full\_name AS customer\_name,

e.full\_name AS employee\_name

FROM orders o

LEFT JOIN customers c ON o.customer\_id = c.customer\_id

LEFT JOIN employees e ON o.employee\_id = e.employee\_id

WHERE o.order\_id = @orderId;

END

CREATE PROCEDURE sp\_GetOrderDetails

@orderId INT

AS

BEGIN

SELECT

od.\*,

p.product\_name,

p.product\_code

FROM order\_details od

INNER JOIN products p ON od.product\_id = p.product\_id

WHERE od.order\_id = @orderId

ORDER BY od.detail\_id;

END

public List<OrderDetail> GetOrderDetails(int orderId)

{

var parameters = new SqlParameter[]

{

new SqlParameter("@orderId", orderId)

};

var dataTable = ExecuteQuery("sp\_GetOrderDetails", parameters, isStoredProcedure: true);

var details = new List<OrderDetail>();

foreach (DataRow row in dataTable.Rows)

{

details.Add(MapRowToOrderDetail(row));

}

return details;

}

public Order GetOrderById(int orderId)

{

var parameters = new SqlParameter[]

{

new SqlParameter("@orderId", orderId)

};

var dataTable = ExecuteQuery("sp\_GetOrderById", parameters, isStoredProcedure: true);

if (dataTable.Rows.Count > 0)

{

var order = MapRowToOrder(dataTable.Rows[0]);

order.OrderDetails = GetOrderDetails(orderId);

return order;

}

return null;

}

CREATE FUNCTION fn\_GetPaymentSummaryByOrderId (@orderId INT)

RETURNS TABLE

AS

RETURN

(

SELECT

o.order\_id,

o.order\_code,

o.total\_amount AS total\_order\_amount,

COALESCE(SUM(p.amount), 0) AS total\_paid\_amount,

(o.total\_amount - COALESCE(SUM(p.amount), 0)) AS remaining\_amount,

o.payment\_status,

COUNT(p.payment\_id) AS payment\_count,

MAX(p.payment\_date) AS last\_payment\_date

FROM orders o

LEFT JOIN payments p ON o.order\_id = p.order\_id AND p.status = 'successful'

WHERE o.order\_id = @orderId

GROUP BY o.order\_id, o.order\_code, o.total\_amount, o.payment\_status

);

public PaymentSummary GetPaymentSummaryByOrderId(int orderId)

{

try

{

string query = "SELECT \* FROM fn\_GetPaymentSummaryByOrderId(@orderId)";

var parameters = new SqlParameter[]

{

new SqlParameter("@orderId", orderId)

};

var dataTable = ExecuteQuery(query, parameters);

if (dataTable.Rows.Count > 0)

{

return MapRowToPaymentSummary(dataTable.Rows[0]);

}

return null;

}

catch (Exception ex)

{

throw new Exception($"Lỗi lấy tóm tắt thanh toán: {ex.Message}");

}

}

CREATE PROCEDURE sp\_InsertPayment

@order\_id INT,

@payment\_date DATETIME,

@payment\_method NVARCHAR(50),

@amount DECIMAL(18,2),

@reference\_number NVARCHAR(100),

@status NVARCHAR(50),

@notes NVARCHAR(MAX),

@payment\_id INT OUTPUT

AS

BEGIN

BEGIN TRY

BEGIN TRANSACTION;

INSERT INTO payments (

order\_id, payment\_date, payment\_method, amount,

reference\_number, status, notes

)

VALUES (

@order\_id, @payment\_date, @payment\_method, @amount,

@reference\_number, @status, @notes

);

SET @payment\_id = SCOPE\_IDENTITY();

-- Cập nhật trạng thái thanh toán của đơn hàng

DECLARE @total\_paid DECIMAL(18,2) = (

SELECT SUM(amount)

FROM payments

WHERE order\_id = @order\_id AND status = 'successful'

);

DECLARE @order\_total DECIMAL(18,2) = (

SELECT total\_amount FROM orders WHERE order\_id = @order\_id

);

UPDATE orders

SET payment\_status =

CASE

WHEN @total\_paid >= @order\_total THEN 'paid'

WHEN @total\_paid > 0 THEN 'partial'

ELSE 'unpaid'

END

WHERE order\_id = @order\_id;

COMMIT;

END TRY

BEGIN CATCH

ROLLBACK;

THROW;

END CATCH

END

public int InsertPayment(Payment payment)

{

try

{

var parameters = new SqlParameter[]

{

new SqlParameter("@order\_id", payment.OrderId),

new SqlParameter("@payment\_date", payment.PaymentDate),

new SqlParameter("@payment\_method", payment.PaymentMethod),

new SqlParameter("@amount", payment.Amount),

new SqlParameter("@reference\_number", payment.ReferenceNumber ?? ""),

new SqlParameter("@status", payment.Status),

new SqlParameter("@notes", payment.Notes ?? ""),

new SqlParameter

{

ParameterName = "@payment\_id",

SqlDbType = SqlDbType.Int,

Direction = ParameterDirection.Output

}

};

ExecuteNonQuery("sp\_InsertPayment", parameters, isStoredProcedure: true);

return Convert.ToInt32(parameters.First(p => p.ParameterName == "@payment\_id").Value);

}

catch (Exception ex)

{

throw new Exception($"Lỗi thêm thanh toán: {ex.Message}");

}

}

CREATE FUNCTION fn\_GenerateCustomerCode()

RETURNS NVARCHAR(50)

AS

BEGIN

DECLARE @prefix NVARCHAR(20) = 'CUS' + FORMAT(GETDATE(), 'yyyyMMdd');

DECLARE @nextNumber INT = (

SELECT COALESCE(MAX(CAST(RIGHT(customer\_code, 4) AS INT)), 0) + 1

FROM customers

WHERE customer\_code LIKE @prefix + '%'

);

RETURN @prefix + RIGHT('0000' + CAST(@nextNumber AS NVARCHAR(4)), 4);

END

public string GenerateCustomerCode()

{

const string query = "SELECT dbo.fn\_GenerateCustomerCode()";

var result = ExecuteScalar(query);

return result?.ToString() ?? "CUS" + DateTime.Now.ToString("yyyyMMdd") + "0001";

}

CREATE PROCEDURE sp\_InsertCustomer

@customer\_code NVARCHAR(50),

@full\_name NVARCHAR(100),

@phone NVARCHAR(20),

@email NVARCHAR(100),

@address NVARCHAR(255),

@date\_of\_birth DATE = NULL,

@gender NVARCHAR(10),

@customer\_type NVARCHAR(50)

AS

BEGIN

BEGIN TRY

BEGIN TRANSACTION;

INSERT INTO customers (

customer\_code, full\_name, phone, email, address,

date\_of\_birth, gender, customer\_type

)

VALUES (

@customer\_code, @full\_name, @phone, @email, @address,

@date\_of\_birth, @gender, @customer\_type

);

COMMIT;

END TRY

BEGIN CATCH

ROLLBACK;

THROW;

END CATCH

END

public bool InsertCustomer(Customer customer)

{

customer.CustomerCode = GenerateCustomerCode();

var parameters = new SqlParameter[]

{

new SqlParameter("@customer\_code", customer.CustomerCode),

new SqlParameter("@full\_name", customer.FullName),

new SqlParameter("@phone", customer.Phone ?? ""),

new SqlParameter("@email", customer.Email ?? ""),

new SqlParameter("@address", customer.Address ?? ""),

new SqlParameter("@date\_of\_birth", (object)customer.DateOfBirth ?? DBNull.Value),

new SqlParameter("@gender", customer.Gender ?? ""),

new SqlParameter("@customer\_type", customer.CustomerType ?? "regular")

};

return ExecuteNonQuery("sp\_InsertCustomer", parameters, isStoredProcedure: true) > 0;

}

CREATE FUNCTION fn\_GetAllOrders

(

@fromDate DATE = NULL,

@toDate DATE = NULL

)

RETURNS TABLE

AS

RETURN

(

SELECT

o.\*,

c.full\_name AS customer\_name,

e.full\_name AS employee\_name,

COUNT(od.detail\_id) AS total\_items

FROM orders o

LEFT JOIN customers c ON o.customer\_id = c.customer\_id

LEFT JOIN employees e ON o.employee\_id = e.employee\_id

LEFT JOIN order\_details od ON o.order\_id = od.order\_id

WHERE

(@fromDate IS NULL OR CAST(o.order\_date AS DATE) >= @fromDate)

AND (@toDate IS NULL OR CAST(o.order\_date AS DATE) <= @toDate)

GROUP BY

o.order\_id, o.order\_code, o.customer\_id, o.employee\_id,

o.order\_date, o.subtotal, o.discount\_amount, o.tax\_amount,

o.total\_amount, o.promotion\_id, o.payment\_method, o.payment\_status,

o.order\_status, o.notes, o.created\_at, o.updated\_at,

c.full\_name, e.full\_name

);

public List<Order> GetAllOrders(DateTime? fromDate = null, DateTime? toDate = null)

{

var parameters = new List<SqlParameter>

{

new SqlParameter("@fromDate", (object)fromDate ?? DBNull.Value),

new SqlParameter("@toDate", (object)toDate ?? DBNull.Value)

};

string query = "SELECT \* FROM fn\_GetAllOrders(@fromDate, @toDate)";

var dataTable = ExecuteQuery(query, parameters.ToArray());

var orders = new List<Order>();

foreach (DataRow row in dataTable.Rows)

{

orders.Add(MapRowToOrder(row));

}

return orders;

}

CREATE PROCEDURE sp\_UpdateOrderStatus

@orderId INT,

@newStatus NVARCHAR(50)

AS

BEGIN

UPDATE orders

SET order\_status = @newStatus

WHERE order\_id = @orderId;

END

public bool UpdateOrderStatus(int orderId, string newStatus)

{

var parameters = new SqlParameter[]

{

new SqlParameter("@orderId", orderId),

new SqlParameter("@newStatus", newStatus)

};

return ExecuteNonQuery("sp\_UpdateOrderStatus", parameters, isStoredProcedure: true) > 0;

}

CREATE PROCEDURE sp\_UpdatePaymentStatus

@orderId INT,

@newStatus NVARCHAR(50)

AS

BEGIN

UPDATE orders

SET payment\_status = @newStatus

WHERE order\_id = @orderId;

END

public bool UpdatePaymentStatus(int orderId, string newStatus)

{

var parameters = new SqlParameter[]

{

new SqlParameter("@orderId", orderId),

new SqlParameter("@newStatus", newStatus)

};

return ExecuteNonQuery("sp\_UpdatePaymentStatus", parameters, isStoredProcedure: true) > 0;

}

CREATE FUNCTION fn\_GetDailySalesReport

(

@fromDate DATE,

@toDate DATE

)

RETURNS TABLE

AS

RETURN

(

SELECT

CAST(order\_date AS DATE) AS report\_date,

COUNT(\*) AS total\_orders,

SUM(total\_amount) AS total\_revenue,

AVG(total\_amount) AS avg\_order\_value

FROM orders

WHERE order\_status = 'completed'

AND CAST(order\_date AS DATE) BETWEEN @fromDate AND @toDate

GROUP BY CAST(order\_date AS DATE)

);

public List<DailySalesReport> GetDailySalesReport(DateTime fromDate, DateTime toDate)

{

string query = "SELECT \* FROM fn\_GetDailySalesReport(@fromDate, @toDate)";

var parameters = new SqlParameter[]

{

new SqlParameter("@fromDate", fromDate.Date),

new SqlParameter("@toDate", toDate.Date)

};

var dataTable = ExecuteQuery(query, parameters);

var reports = new List<DailySalesReport>();

foreach (DataRow row in dataTable.Rows)

{

reports.Add(new DailySalesReport

{

Date = Convert.ToDateTime(row["report\_date"]),

TotalOrders = Convert.ToInt32(row["total\_orders"]),

TotalRevenue = Convert.ToDecimal(row["total\_revenue"]),

AverageOrderValue = Convert.ToDecimal(row["avg\_order\_value"])

});

}

return reports;

}

CREATE PROCEDURE sp\_GetTopSellingProducts

@fromDate DATE,

@toDate DATE,

@topCount INT

AS

BEGIN

SET NOCOUNT ON;

SELECT TOP (@topCount)

p.product\_id,

p.product\_name,

b.brand\_name,

SUM(od.quantity) AS quantity\_sold,

SUM(od.total\_price) AS total\_revenue,

AVG(od.unit\_price) AS avg\_price,

COUNT(DISTINCT od.order\_id) AS total\_orders

FROM products p

INNER JOIN order\_details od ON p.product\_id = od.product\_id

INNER JOIN orders o ON od.order\_id = o.order\_id

LEFT JOIN brands b ON p.brand\_id = b.brand\_id

WHERE o.order\_status = 'completed'

AND CAST(o.order\_date AS DATE) BETWEEN @fromDate AND @toDate

GROUP BY p.product\_id, p.product\_name, b.brand\_name

ORDER BY quantity\_sold DESC;

END

public List<ProductSalesReport> GetTopSellingProducts(DateTime fromDate, DateTime toDate, int topCount = 10)

{

var parameters = new SqlParameter[]

{

new SqlParameter("@topCount", topCount),

new SqlParameter("@fromDate", fromDate.Date),

new SqlParameter("@toDate", toDate.Date)

};

var dataTable = ExecuteQuery("sp\_GetTopSellingProducts", parameters, isStoredProcedure: true);

var reports = new List<ProductSalesReport>();

foreach (DataRow row in dataTable.Rows)

{

reports.Add(new ProductSalesReport

{

ProductId = Convert.ToInt32(row["product\_id"]),

ProductName = row["product\_name"].ToString(),

BrandName = row["brand\_name"].ToString(),

QuantitySold = Convert.ToInt32(row["quantity\_sold"]),

TotalRevenue = Convert.ToDecimal(row["total\_revenue"]),

AveragePrice = Convert.ToDecimal(row["avg\_price"]),

TotalOrders = Convert.ToInt32(row["total\_orders"])

});

}

return reports;

CREATE PROCEDURE sp\_GetEmployeePerformanceReport

@fromDate DATE,

@toDate DATE

AS

BEGIN

SET NOCOUNT ON;

SELECT

e.employee\_id,

e.full\_name,

e.position,

COUNT(o.order\_id) AS total\_orders,

COALESCE(SUM(o.total\_amount), 0) AS total\_sales,

COALESCE(AVG(o.total\_amount), 0) AS avg\_order\_value,

COALESCE(MAX(o.order\_date), e.created\_at) AS last\_sale\_date

FROM employees e

LEFT JOIN orders o ON e.employee\_id = o.employee\_id

AND o.order\_status = 'completed'

AND CAST(o.order\_date AS DATE) BETWEEN @fromDate AND @toDate

WHERE e.status = 'active'

GROUP BY e.employee\_id, e.full\_name, e.position, e.created\_at

ORDER BY total\_sales DESC;

END

public List<EmployeePerformanceReport> GetEmployeePerformanceReport(DateTime fromDate, DateTime toDate)

{

var parameters = new SqlParameter[]

{

new SqlParameter("@fromDate", fromDate.Date),

new SqlParameter("@toDate", toDate.Date)

};

var dataTable = ExecuteQuery("sp\_GetEmployeePerformanceReport", parameters, isStoredProcedure: true);

var reports = new List<EmployeePerformanceReport>();

foreach (DataRow row in dataTable.Rows)

{

reports.Add(new EmployeePerformanceReport

{

EmployeeId = Convert.ToInt32(row["employee\_id"]),

EmployeeName = row["full\_name"].ToString(),

Position = row["position"].ToString(),

TotalOrders = Convert.ToInt32(row["total\_orders"]),

TotalSales = Convert.ToDecimal(row["total\_sales"]),

AverageOrderValue = Convert.ToDecimal(row["avg\_order\_value"]),

LastSaleDate = Convert.ToDateTime(row["last\_sale\_date"])

});

}

return reports;

}

CREATE VIEW v\_ActiveEmployees

AS

SELECT

employee\_id,

employee\_code,

full\_name,

position,

status

FROM employees

WHERE status = 'active';

public List<EmployeeListItem> GetEmployeesForComboBox(bool includeAll = true)

{

var items = new List<EmployeeListItem>();

if (includeAll)

{

items.Add(new EmployeeListItem

{

Value = -1,

Text = "Tất cả nhân viên",

Position = "",

Status = "all"

});

}

string query = "SELECT \* FROM v\_ActiveEmployees ORDER BY position, full\_name";

var dataTable = ExecuteQuery(query);

foreach (DataRow row in dataTable.Rows)

{

items.Add(new EmployeeListItem

{

Value = Convert.ToInt32(row["employee\_id"]),

Text = row["full\_name"].ToString(),

Position = row["position"].ToString(),

Status = row["status"].ToString()

});

}

return items;

}

CREATE VIEW v\_ActivePromotions

AS

SELECT \*

FROM promotions

WHERE

status = 'active'

AND CAST(GETDATE() AS DATE) BETWEEN start\_date AND end\_date

AND (usage\_limit IS NULL OR used\_count < usage\_limit);

public List<Promotion> GetActivePromotions()

{

string query = "SELECT \* FROM v\_ActivePromotions ORDER BY created\_at DESC";

var dataTable = ExecuteQuery(query);

var promotions = new List<Promotion>();

foreach (DataRow row in dataTable.Rows)

{

promotions.Add(MapRowToPromotion(row));

}

return promotions;

}

CREATE PROCEDURE sp\_GetPromotionByCode

@code NVARCHAR(50)

AS

BEGIN

SET NOCOUNT ON;

SELECT \*

FROM promotions

WHERE promotion\_code = @code

AND status = 'active'

AND CAST(GETDATE() AS DATE) BETWEEN start\_date AND end\_date

AND (usage\_limit IS NULL OR used\_count < usage\_limit);

END

public Promotion GetPromotionByCode(string promotionCode)

{

var parameters = new SqlParameter[]

{

new SqlParameter("@code", promotionCode)

};

var dataTable = ExecuteQuery("sp\_GetPromotionByCode", parameters, isStoredProcedure: true);

if (dataTable.Rows.Count > 0)

{

return MapRowToPromotion(dataTable.Rows[0]);

}

return null;

}

CREATE PROCEDURE sp\_InsertPromotion

@code NVARCHAR(50),

@name NVARCHAR(100),

@description NVARCHAR(MAX),

@discountType NVARCHAR(20),

@discountValue DECIMAL(18, 2),

@startDate DATE,

@endDate DATE,

@minOrderAmount DECIMAL(18, 2),

@maxDiscountAmount DECIMAL(18, 2),

@usageLimit INT = NULL,

@status NVARCHAR(20)

AS

BEGIN

SET NOCOUNT ON;

BEGIN TRY

BEGIN TRANSACTION;

INSERT INTO promotions (

promotion\_code, promotion\_name, description, discount\_type,

discount\_value, start\_date, end\_date, min\_order\_amount,

max\_discount\_amount, usage\_limit, status

)

VALUES (

@code, @name, @description, @discountType, @discountValue,

@startDate, @endDate, @minOrderAmount, @maxDiscountAmount, @usageLimit, @status

);

COMMIT;

END TRY

BEGIN CATCH

ROLLBACK;

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

THROW 50001, @err, 1;

END CATCH

END

public bool InsertPromotion(Promotion promotion)

{

try

{

var parameters = new SqlParameter[]

{

new SqlParameter("@code", promotion.PromotionCode),

new SqlParameter("@name", promotion.PromotionName),

new SqlParameter("@description", promotion.Description ?? ""),

new SqlParameter("@discountType", promotion.DiscountType),

new SqlParameter("@discountValue", promotion.DiscountValue),

new SqlParameter("@startDate", promotion.StartDate.Date),

new SqlParameter("@endDate", promotion.EndDate.Date),

new SqlParameter("@minOrderAmount", promotion.MinOrderAmount),

new SqlParameter("@maxDiscountAmount", promotion.MaxDiscountAmount),

new SqlParameter("@usageLimit", promotion.UsageLimit > 0 ? (object)promotion.UsageLimit : DBNull.Value),

new SqlParameter("@status", promotion.Status ?? "active")

};

ExecuteNonQuery("sp\_InsertPromotion", parameters, isStoredProcedure: true);

return true;

}

catch

{

return false;

}

}

CREATE PROCEDURE sp\_GetAllPromotions

AS

BEGIN

SET NOCOUNT ON;

SELECT \*

FROM promotions

ORDER BY created\_at DESC;

END

public List<Promotion> GetAllPromotions()

{

var dataTable = ExecuteQuery("sp\_GetAllPromotions", null, isStoredProcedure: true);

var promotions = new List<Promotion>();

foreach (DataRow row in dataTable.Rows)

{

promotions.Add(MapRowToPromotion(row));

}

return promotions;

}

CREATE PROCEDURE sp\_UpdatePromotionStatus

@promotionId INT,

@status NVARCHAR(20)

AS

BEGIN

SET NOCOUNT ON;

UPDATE promotions

SET status = @status,

updated\_at = GETDATE()

WHERE promotion\_id = @promotionId;

END

public bool UpdatePromotionStatus(int promotionId, string status)

{

try

{

var parameters = new SqlParameter[]

{

new SqlParameter("@status", status),

new SqlParameter("@promotionId", promotionId)

};

ExecuteNonQuery("sp\_UpdatePromotionStatus", parameters, isStoredProcedure: true);

return true;

}

catch

{

return false;

}

}

CREATE PROCEDURE sp\_DeletePromotion

@promotionId INT

AS

BEGIN

SET NOCOUNT ON;

BEGIN TRY

BEGIN TRANSACTION;

DELETE FROM promotions

WHERE promotion\_id = @promotionId;

COMMIT;

END TRY

BEGIN CATCH

ROLLBACK;

THROW; -- ném lỗi rõ ràng lên ứng dụng

END CATCH

END

public bool DeletePromotion(int promotionId)

{

try

{

var parameters = new SqlParameter[]

{

new SqlParameter("@promotionId", promotionId)

};

ExecuteNonQuery("sp\_DeletePromotion", parameters, isStoredProcedure: true);

return true;

}

catch

{

return false;

}

}

ALTER PROCEDURE sp\_UpdatePromotion

@promotionId INT,

@name NVARCHAR(100),

@description NVARCHAR(MAX),

@discountType NVARCHAR(20),

@discountValue DECIMAL(18, 2),

@startDate DATE,

@endDate DATE,

@minOrderAmount DECIMAL(18, 2),

@maxDiscountAmount DECIMAL(18, 2),

@usageLimit INT = NULL,

@status NVARCHAR(20)

AS

BEGIN

SET NOCOUNT ON;

UPDATE promotions

SET promotion\_name = @name,

description = @description,

discount\_type = @discountType,

discount\_value = @discountValue,

start\_date = @startDate,

end\_date = @endDate,

min\_order\_amount = @minOrderAmount,

max\_discount\_amount = @maxDiscountAmount,

usage\_limit = @usageLimit,

status = @status,

updated\_at = GETDATE()

WHERE promotion\_id = @promotionId;

END

public bool UpdatePromotion(Promotion promotion)

{

try

{

var parameters = new SqlParameter[]

{

new SqlParameter("@promotionId", promotion.PromotionId),

new SqlParameter("@name", promotion.PromotionName),

new SqlParameter("@description", promotion.Description ?? ""),

new SqlParameter("@discountType", promotion.DiscountType),

new SqlParameter("@discountValue", promotion.DiscountValue),

new SqlParameter("@startDate", promotion.StartDate.Date),

new SqlParameter("@endDate", promotion.EndDate.Date),

new SqlParameter("@minOrderAmount", promotion.MinOrderAmount),

new SqlParameter("@maxDiscountAmount", promotion.MaxDiscountAmount),

new SqlParameter("@usageLimit", promotion.UsageLimit > 0 ? (object)promotion.UsageLimit : DBNull.Value),

new SqlParameter("@status", promotion.Status ?? "active")

};

ExecuteNonQuery("sp\_UpdatePromotion", parameters, isStoredProcedure: true);

return true;

}

catch

{

return false;

}

}