GAP Retail Database System - Project Overview

> **AI Assistant Instructions**:
> When helping with this project:
> - Reference ALL files in the project folder to maintain consistency
> - Suggest updates to any relevant files when necessary
> - Understand how changes to one file might impact others
> - Keep track of progress through the project phases
> - Provide an updated version of this context document at the end of each session

Project Description

This project involves developing a comprehensive retail database system for GAP, a global clothing and accessories retailer. The database supports all aspects of retail operations including inventory management, order processing, customer management, store operations, warehouse management, and analytics. This SQL-based system is designed to handle GAP's omnichannel retail environment (online, in-store, and pickup services).

Database Schema Overview

The database follows a relational model with properly normalized tables and appropriate relationships. The schema is designed to support the complete retail lifecycle from inventory management to sales and returns processing. The system handles both B2C (direct to consumer) and internal operations (warehouse management, supplier orders).

Current Progress

- Completed database schema design with all tables, relationships, and constraints
- Populated base independent tables (categories, warehouses, stores, suppliers)
- Populated dependent tables (customers, addresses, loyalty_accounts, products, promotions, shopping_cart)

Next Steps

1. Populate child/junction tables (product_variants, product_categories, inventory_levels, shopping_cart_items)
2. Generate transaction data (orders, order_items, payments, shipments)
3. Create operational data (returns, supplier_orders, warehouse transfers)

4. Develop business query examples for common operations
5. Implement reporting and analytics features

Technical Details

Database Architecture

- MySQL InnoDB database with comprehensive schema
- Includes 30+ interconnected tables with proper foreign key relationships
- Designed for scalability with appropriate indexes and constraints
- Features timestamp tracking for record creation and updates

Key Tables

- **Independent entities**: categories, warehouses, stores, suppliers, payment_methods, payment_status
- **Dependent entities**: customers, addresses, loyalty_accounts, products, promotions, shopping_cart
- **Child/junction entities**: product_variants, inventory_levels, shopping_cart_items, product_categories
- **Transaction entities**: orders, order_items, payments, shipments, shipment_items
- **Operational entities**: returns, return_items, supplier_orders, warehouse_stock_transfer
- **Analytics entities**: sales_analytics, order_tracking

Key Use Cases

1. **Customer Online Ordering (UC-1)**: Supports full e-commerce flow from product browsing to checkout
2. **Customer In-Store Purchase (UC-2)**: Handles point-of-sale transactions in physical stores
3. **Inventory Management (UC-3)**: Manages product lifecycle and stock levels across locations
4. **Warehouse Fulfillment (UC-4)**: Processes order fulfillment, shipping, and tracking
5. **Returns Management (UC-5)**: Handles product returns and refunds across channels
6. **Promotions Application (UC-6)**: Applies eligible promotions to shopping carts and orders
7. **Payment Processing (UC-7)**: Processes various payment methods and tracks payment status

Data Generation Challenges & Solutions

- **Volume Challenge**: Need to generate thousands of realistic records
  - **Solution**: Using cross joins and temporary tables for efficient generation
- **Consistency Challenge**: Maintaining referential integrity across related tables
  - **Solution**: Using transactions and foreign key constraints

- **Variety Challenge**: Creating diverse, realistic data
  - **Solution**: Using randomization with controlled distributions
- **Performance Challenge**: Efficient insertion of large datasets
  - **Solution**: Batch inserts, temporarily disabling indexes during load

Common SQL Patterns Used

- Batch inserts with multiple rows per INSERT statement
- Temporary tables for reference data
- Transaction blocks for data integrity
- CROSS JOIN operations for combinatorial data generation
- Randomization functions (RAND()) with constraints
- JSON generation for flexible attribute storage

Technology Stack

- MySQL/MariaDB with InnoDB engine
- SQL for all database operations and queries
- Potential integration with retail applications via standard connectors

Project Timeline

- **Phase 1** (Completed): Database schema design and base table creation
- **Phase 2** (In Progress): Data population for all tables
- **Phase 3** (Upcoming): Implementation of complex queries and stored procedures
- **Phase 4** (Upcoming): Development of business reports and analytics
- **Phase 5** (Upcoming): Performance optimization and indexing strategies

Project Files Organization

- **Schema Definition**: gap-revised-schema.sql
- **Base Tables Population**: 02_p1.sql (categories, warehouses, stores), 02_p2.sql (suppliers)
- **Dependent Tables**: 03_dependent_tables.sql (customers, addresses, loyalty, products, promotions)
- **Future Files**: 04_child_tables.sql, 05_transaction_tables.sql, 06_operational_tables.sql, etc.

Notable Features

- Support for multi-channel shopping (online, in-store, pickup)

- Comprehensive inventory tracking across warehouses and stores
- Flexible promotion system supporting various discount types
- Detailed customer profiles with loyalty program integration
- Order lifecycle tracking from creation to fulfillment
- Return processing with inventory adjustments
- Supplier order management and warehouse transfer capabilities

Design Considerations

- Normalized structure with appropriate denormalization for performance
- Comprehensive foreign key relationships for data integrity
- JSON support for flexible data structures (e.g., marketing preferences)
- Timestamp tracking for audit purposes
- Status fields using ENUMs for better data validation
- CHECK constraints for business rule enforcement

Instructions for AI Assistants

This is a living document that represents the current state of the GAP Retail Database project. When working on this project:

1. **REFERENCE ALL FILES**: Always examine and reference ALL files in the project folder to ensure consistency in your recommendations.
2. **SUGGEST FILE UPDATES**: Proactively suggest updates to any relevant SQL files when improvements are possible or when changes in one file necessitate updates to others.
3. **TRACK DEPENDENCIES**: Be aware of how changes to one file might impact others due to table dependencies and referential integrity requirements.
4. **MONITOR PROJECT PROGRESS**: Track progress through the project phases and align your assistance with the current phase.
5. **UPDATE THIS DOCUMENT**: At the end of each session, provide an updated version of this context document that incorporates all progress made, ensuring it accurately reflects the current state of the project.

These instructions are critical for maintaining project continuity across multiple sessions.

Key Resources

- Database Schema: gap-revised-schema.sql

- Base Data Population: 02_p1.sql, 02_p2.sql
- Dependent Tables Population: 03_dependent_tables.sql
- Use Case Documentation: Contains detailed descriptions of the 7 primary use cases (UC-1 through UC-7)
- Next Steps Plan: Outlines the sequence for completing the remaining data population scripts

Glossary

- **SKU**: Stock Keeping Unit, unique identifier for product variants
- **MSRP**: Manufacturer's Suggested Retail Price
- **UC**: Use Case (e.g., UC-1 refers to Use Case 1 - Customer Online Ordering)
- **POS**: Point of Sale system used in physical stores
- **WMS**: Warehouse Management System