CSC 540 - Group W:

High-Level Design Decisions:
Throughout the development of our application programs, we made several key design decisions that helped us maintain consistency, enforce integrity, and ensure the system was both functional and realistic.

1. Managing transactions manually using JDBC: One of the first things we agreed on was to take control of transaction management in our Java programs. We disabled autocommit and used commit() and rollback() explicitly. This gave us the flexibility to group related operations—like restocking inventory or handling returns—into atomic transactions. It helped us avoid partial updates and made error handling much more reliable.

2. Structuring staff roles through separate tables: We decided early on to create separate tables for different staff roles, like 'Billing', 'Cashier', 'Registration Officer', 'Warehouse Worker', 'Manager', 'Assistant Manager', 'Other', instead of relying on just one general StaffMember table. All of these reference the main staff table, which helped us reduce redundancy while keeping our role-specific logic organized and easier to manage.

3. Enforcing data integrity at multiple levels: Maintaining data integrity was something we prioritized throughout. On the schema side, we enforced it using NOT NULL, UNIQUE, and FOREIGN KEY constraints. But we also added validation logic in our Java code to prevent things like negative inventory quantities or duplicate entries. This dual-layer protection made our system more robust and less error-prone.

4. Building reports directly with SQL queries: Rather than creating a separate layer for analytics, we implemented all reporting features using direct SQL queries. These include customer activity reports, sales summaries, and inventory stock levels. This approach made use of SQL’s strengths and allowed us to get meaningful insights with minimal overhead.

5. Letting our assumptions shape the design: Many of our design choices were guided by the assumptions we documented, like each transaction involving a single customer, or reward checks being generated only for platinum members at year-end. These assumptions helped us simplify logic and avoid unnecessary complexity while staying aligned with our use-case.