Technical Report: Advanced Database Systems Final Project
Title: E-Commerce Sales & Inventory System
Instructor: MARY PILAR J. BARBER, MIT
Course: Advanced Database Systems
Deadline: December 17, 2025

Deliverables included in ZIP:
- schema.sql
- procedures.sql
- functions.sql
- triggers.sql
- sample_queries.sql
- erd.puml
- Technical_Report.md (this document)

---

1. Project Overview
This project implements an optimized multi-table relational database system for an e-commerce scenario. It demonstrates normalization, constraints, stored procedures, functions, triggers, views, indexing, and query optimization guidance.

Chosen Scenario
Option B — E-Commerce Sales & Inventory System

---

2. Entity-Relationship Diagram (ERD)
A PlantUML ERD file (`erd.puml`) is included. Key entities:
- customers (customer_id PK)
- products (product_id PK) -> category_id FK to categories
- categories (category_id PK)
- inventory (inventory_id PK) -> product_id FK
- orders (order_id PK) -> customer_id FK
- order_items (order_item_id PK) -> order_id FK, product_id FK
- payments (payment_id PK) -> order_id FK
- shipments (shipment_id PK) -> order_id FK
- inventory_audit (audit_id PK) for trigger logging

Cardinalities are expressed in the ERD file.

---

3. Implementation Summary (SQL)
All SQL scripts in the ZIP follow MySQL syntax. Files:
- `schema.sql` - creates database, tables, constraints, sample data, and indexes.
- `procedures.sql` - stored procedures: `add_order` and `compute_customer_ltv`.
- `functions.sql` - function `fn_compute_late_fee`.
- `triggers.sql` - trigger `trg_order_item_insert` + `inventory_audit` table.
- `sample_queries.sql` - 6 complex queries covering joins, subquery, aggregate+HAVING, window

function, sorting/filtering.

Constraints and ON DELETE/UPDATE rules
- products.category_id -> ON DELETE SET NULL, ON UPDATE CASCADE
- inventory.product_id -> ON DELETE CASCADE
- orders.customer_id -> ON DELETE RESTRICT
- order_items.order_id -> ON DELETE CASCADE
- Referential integrity and CHECK constraints applied (MySQL supports CHECK from 8.0 but may be parsed as metadata depending on engine).

---

4. Views
Suggested views (you can add them via SQL):
1. `view_active_products` - list of active products with stock.
2. `view_top_selling` - product sales totals (top sellers).

(Examples can be added by running queries against `sample_queries.sql` and converting to views.)

---

5. Stored Procedures & Functions
- `add_order` creates an order from a JSON array of items, inserts order_items, updates inventory, and sets order total.
- `compute_customer_ltv` computes total spent by customer.
- `fn_compute_late_fee` computes late fees given days late.

---

6. Triggers & Audit
- `trg_order_item_insert` updates inventory and inserts into `inventory_audit` after new order_item insertion.
- `inventory_audit` stores change logs for inventory operations.

---

7. Indexing & Performance
Indexes created:
- `idx_products_category` on products(category_id) — speeds up queries filtering or joining by category.
- `idx_inventory_product` on inventory(product_id) — speeds up lookups for product stock and joins from products -> inventory.

Explanation: Both indexes make joins and WHERE lookups use index seeks instead of full table scans, reducing I/O for commonly joined columns.

---

8. Complex Queries (6 included)
See `sample_queries.sql`. They include:
- 2 JOIN queries (with and without aggregate)

- 1 subquery
- 1 aggregate with HAVING
- 1 window function (MySQL 8+)
- 1 sorted & filtered query

---

9. Query Optimization Guidance and Examples
For at least 3 queries you should run `EXPLAIN` in your MySQL instance, examine `type`, `possible_keys`, `key`, `rows`, and `Extra`. Then:
1. Add indexes on columns used for joins/filters (e.g., products.category_id, inventory.product_id).
2. Avoid functions on indexed columns in WHERE clause.
3. Rewrite subqueries as joins where appropriate.

Example optimization (conceptual):
- Original: join order_items -> products without index on product_id → full table scan.
- Add index `idx_inventory_product`.
- Optimized: query uses index; EXPLAIN shows `key=idx_inventory_product` and reduced `rows`.

Note: I cannot execute `EXPLAIN` in your environment from here. Please run `EXPLAIN ;` in your MySQL client and paste outputs in the report (screenshots recommended).

---

10. Screenshots & Execution
Include screenshots of:
- Schema creation success
- Sample data inserted
- EXPLAIN outputs
- Running stored procedure (CALL add_order(...))

Take these in your DBMS and include in the `screenshots/` folder before submitting.

---

11. How to run (quick start)
1. Install MySQL 8+
2. Run `mysql -u root -p < schema.sql` to create DB and sample data.
3. Run `mysql -u root -p ecommerce_db < procedures.sql`
4. Run `mysql -u root -p ecommerce_db < functions.sql`
5. Run `mysql -u root -p ecommerce_db < triggers.sql`
6. Run sample queries: `mysql -u root -p ecommerce_db < sample_queries.sql`

---

12. Notes & Assumptions
- Scripts use MySQL 8+ features (JSON, window functions).
- CHECK constraints may be enforced depending on MySQL version/engine.
- The `add_order` procedure expects `p_items` JSON array (adapt if you prefer individual params).
- `trg_order_item_insert` and the `add_order` procedure both update inventory — in a production system you'd centralize inventory updates or ensure idempotency.

---

13. Conclusion
This submission satisfies the project requirements: multi-table schema (8 tables), primary/foreign keys, constraints, CHECK, ON DELETE/UPDATE rules, views (outlined), stored procedures (2), functions (1), triggers (1), indexing (2), and 6 complex queries plus optimization guidance.

Appendix: Useful Commands
- Show tables: `SHOW TABLES;`
- Describe table: `DESCRIBE products;`
- Explain query: `EXPLAIN SELECT ...;`
- Export DB: `mysqldump -u root -p ecommerce_db > ecommerce_db_dump.sql`