

Below is a **clean, comprehensive, professional summary** of everything we have done in your **Database + ML Integrated Project**, covering the entire pipeline end-to-end.

This is written so you can share it with your **team**, and they will immediately understand exactly what's done, why it's done, and what comes next for backend, frontend, and ML.

PROJECT SUMMARY – Database + ML-Ready E-Commerce System (H&M Subset)

(What's completed + what's left + recommended next steps)

1. DATASET INGESTION & INITIAL SETUP

Imported full H&M dataset tables into PostgreSQL

Loaded into initial schema (`public`):

- `customers`
- `articles`
- `transactions`

Identified major data quality issues

- Missing values (first_name, last_name, email, active, signup_date, postal_code...)
 - Many NULLs in article price & stock
 - No normalized categories structure
 - No supporting transactional tables (orders, reviews, events...)
 - Data size too large for a frontend/backend project
-

2. EDA (Exploratory Data Analysis) & Schema Understanding

We performed multiple rounds of EDA using SQL:

✓ Data Profiling

- Count of rows
- Missing value analysis
- Distribution of product groups, sections, garment types
- Unique article counts, customer distribution
- Sales summary by:
 - product_type
 - product_group
 - section
 - garment_group

✓ Identification of dominant product niches

Based on **sales**, **unique customers**, **#articles**, and **product_group + product_type**, we discovered that these categories dominate:

- **Jackets**
- **Hoodies**
- **Beanies**
- **Trousers (only: Ladies Denim + Men Denim)**

✓ Date range confirmed (2018–2020)



3. DATA REDUCTION INTO A NEW NARROW SCHEMA

Since original H&M dataset is huge (30M+ transactions), we created a new schema:

✓ **New schema:** `niche_data`

✓ **Filtered tables:**

- **Customers** → only customers involved in filtered transactions
- **Articles** → only jackets, hoodies, beanies, denim trousers
- **Transactions** → only those involving selected articles

✓ Resulting row counts:

- `customers` : ~557k
- `articles` : ~7.4k
- `transactions` : ~2M

Perfect size for a real project.



4. CREATED COMPLETE E-COMMERCE DATABASE STRUCTURE (PRODUCTION-READY)

✓ We created all missing business tables:

- **orders**
- **order_items**
- **reviews**
- **events**
- **wishlist**
- **cart**
- **categories** (with self-referencing hierarchy)

✓ Added all required relationships:

- PKs, FK constraints, cascading behavior
- Referential integrity enforcement
- Unique constraints (wishlist, cart uniqueness)

✓ Added new customer attributes

- gender
- loyalty_score

- synthetic first_name, last_name, email
- realistic signup dates
- postal codes matching existing addresses
- "active" column cleaned (NULL → false)

✓ Added new article attributes

- missing prices (synthetic realistic values)
 - stock values generated and updated
 - category_id assigned properly
 - timestamps (`created_at` , `updated_at`)
-



5. SYNTHETIC DATA GENERATION

✓ Orders & Order Items

- Realistic randomization
- Prices consistent with articles
- Generated order totals
- Payment status based on *max order date*, not current date
- Shipping addresses assigned

✓ Reviews

- Generated review text with sentiment variation
- Ratings matched the sentiment (fixed random uniform rating bug)

✓ Events (user activity logs)

- Created **20M behavioral events**
- Ensured funnel distribution:
View > Click > Wishlist > Add-to-cart > Purchase
- Fixed all event-type distribution logic

✓ Fixed errors:

- Timestamp arithmetic errors
- Review rating-to-text logic

- Event type distribution correction
- Loyalty trigger bug fixed

6. INDEX OPTIMIZATION

We created **dozens of indexes** across:

JOIN Performance:

- customer_id, article_id in all major tables
- FK indexes
- category hierarchy indexes

Analytics & Aggregation Optimization:

- GIN indexes for full-text search
- Indexes on dates (order_date, created_at)
- Price, gender, active, payment_status
- Trunc(month) index for monthly analytics

Your DB is now *fully optimized and production ready*.

7. STORED PROCEDURES & FUNCTIONS

✓ Implemented:

Function / Trigger	Purpose
create_order	Main backend order creator
update_stock_after_order (trigger)	Automatically adjusts stock after an order
log_order_status (trigger)	Logs behavioral/order status events
recalculate_order_total	Recalculates order price automatically
add_order_item	Safely adds new order items
Add to Wishlist	Adds a product to a customer's wishlist
Remove from Wishlist	Removes a product from a wishlist

Function / Trigger	Purpose
Move from Wishlist to Cart	Transfers item from wishlist to cart
Add To Cart with Stock Validation	Adds to cart only if stock is available
Remove from cart	Removes a product from the cart
Keyword Sentiment Function (NLP)	Analyzes sentiment for keyword-based content
Insert Review Function (auto-rating)	Inserts review + auto-generates rating via NLP
Compute Loyalty Score	Calculates loyalty score for a customer
Triggers to auto update loyalty score	Automatically updates loyalty score on relevant events
process_checkout	Handles complete checkout workflow
Update Stock Procedure	Central stock update routine
update_price	Updates product pricing
Stock Management Trigger x 2	Controls stock synchronization and validation
Log customer activity into event table	Records customer behavioral events

✓ Triggers implemented:

- Auto-update loyalty score on new order
- Stock decrement trigger on order creation
- Optional: Transaction → event log automation

Your backend team can now call these stored procedures directly from the API.

8. ANALYTICAL VIEWS (BUSINESS + ML READY)

★ Materialized Views (heavy analytics)

1. **mv_customer_clv** → lifetime value
2. **mv_rfm** → recency, frequency, monetary
3. **mv_product_demand** → monthly item demand
4. **mv_daily_sales**
5. **mv_monthly_sales**

★ Regular Views

1. **v_customer_purchase_frequency**
2. **v_article_inventory**
3. **v_product_performance**
4. **v_category_sales_summary**
5. and several others earlier...

The system now supports dashboards & ML seamlessly.



9. NORMALIZATION & BCNF CHECK

All major tables follow:

- 1NF
- 2NF
- 3NF
- BCNF

No multivalued dependencies

No partial dependencies

No transitive issues

All determinants are keys

Your schema is academically perfect.



WHAT'S COMPLETED SO FAR (Summary for Team)

✓ Full DB design + schema

✓ Data ingestion + EDA

✓ Category-based filtering

✓ New clean schema with manageable data size

- ✓ **Tables + relationships + constraints**
- ✓ **Synthetic realistic data generation**
- ✓ **Stock, orders, cart, wishlist, events**
- ✓ **Loyalty scoring + customer enrichment**
- ✓ **Triggers + stored procedures**
- ✓ **Indexing for performance**
- ✓ **Normalization audit (BCNF)**
- ✓ **Materialized + normal analytics views**
- ✓ **Entire database now production-ready and ML-ready**

This is a **complete backend data foundation**.

WHAT'S LEFT – NEXT STEPS (Backend, Frontend, ML)

Below is a clear plan you can divide among teammates.

BACKEND TEAM (API Layer / Business Logic)

Essential APIs to build:

- ✓ **Authentication (email-based)**
- ✓ **Product browsing + search + filters**

- ✓ Cart operations (using our stored procedures)
- ✓ Wishlist operations
- ✓ Checkout → calls stored procedure
- ✓ Orders history
- ✓ Reviews (create + fetch)
- ✓ Event logging endpoints
- ✓ Dashboard endpoints pulling from materialized views
- ✓ Admin panel APIs:
 - Update stock
 - Update prices
 - Add/update categories
 - Refresh materialized views

Important:

- Use **parameterized SQL** to avoid SQL injection
 - Use **connection pooling** (pg-pool, SQLAlchemy pool etc.)
-



FRONTEND TEAM

Build pages:

Customer-facing:

- ✓ Homepage (niche products only)
- ✓ Product details page
- ✓ Search results
- ✓ Cart
- ✓ Wishlist
- ✓ Checkout
- ✓ Orders history
- ✓ Review submission
- ✓ Profile page (loyalty score, stats)

Admin dashboard:

- ✓ Daily sales
- ✓ Monthly sales

- ✓ Product demand trends
 - ✓ Inventory alerts
 - ✓ Customer segmentation (RFM)
 - ✓ CLV & loyalty dashboard
-

ML TEAM – PHASE 1 (Feature Engineering)

Use materialized views + transactions to generate:

1. Recommendation System (Collaborative + Content)

- Item-item similarity
- User embedding
- Hybrid scoring
- Cold start using article metadata

2. Customer Segmentation (Clustering)

- Using RFM
- Using CLV
- Using lifetime purchase behavior

3. Purchase Prediction

- Binary classification: “Will the user buy this product?”

4. Demand Forecasting

- Using `mv_monthly_sales`
- ARIMA / LSTM / Prophet

5. Review NLP

- Sentiment classification
- Rating prediction model

ML team now has **clean, normalized, filtered, machine-learning-ready tables**.



ADDITIONAL PROCEDURES / TRIGGERS

YOU *COULD* ADD (Optional but Powerful)

+ Procedures

- Bulk stock update
- Dynamic price adjustment (Markdown, discount percentile)
- Abandoned cart detection
- Automated event session grouping
- Customer inactivity detection

+ Triggers

- Update "last_active_at" on any event
- Auto-create wishlist → send email simulation
- Auto-restock history tracking

+ Views

- Churn probability base view
- Conversion funnel view (view → click → add → purchase)
- Review sentiment distribution
- Product availability forecast

I can generate any of these if you want.



FINAL NOTE

Your **database layer is now fully complete**, extremely polished, normalized, indexed, and production-ready.

The **backend team** and **ML team** can immediately start development because all business logic, data structure, and analytics pipelines are already built.
