Group No.6 – E-Commerce Data Warehouse Final Report

E-Commerce Data Warehouse

Schema Implementation, ETL, ELT, OLAP/MOLAP, Joining Techniques, Indexing, Partitioning & Performance Optimization

1. Introduction

The primary goal of this project is to design, implement, and optimize a **Data Warehouse (DW)** for an E-Commerce system. This DW enables **Decision Support System (DSS)** users to perform analytical queries efficiently. In Iteration 3, the focus was on **performance optimization** using indexing, partitioning, materialized views, and index type comparisons.

This report integrates all aspects of the project, including **ETL/ELT workflows**, **OLAP/MOLAP concepts**, **joining techniques**, and **performance optimization strategies**, showing how each component contributes to DSS efficiency and also presents the complete DW implementation for an E-Commerce analytics platform developed by **Group 6**. The project included:

- Designing a dimensional star schema
- Implementing a complete **ETL Pipeline**
- Implementing a parallel **ELT Pipeline**
- Applying **OLAP & MOLAP analytical operations**
- Performance optimization through **Indexing, Partitioning & Materialized Views**
- Evaluating join algorithms: **Nested Loop, Sort-Merge, Hash Join**
- Benchmarking **DSS vs OLTP workloads**
- A detailed **Iteration 3 optimization analysis**

The final DW supports high-performance reporting workloads, reduces query runtime significantly, and adheres to industry best practices for Data Warehouse (DW) design.

2. Project Objectives

The project aimed to design a complete E-Commerce Data Warehouse optimized for analytical processing. Deliverable 0 focused on implementing a **star-schema** DW with properly defined dimensions, fact tables, surrogate keys, and schema justification. Deliverable 1 focused on building **ETL and ELT pipelines**, including extraction, cleaning, transformation, loading, and handling rejected rows. Deliverable 2 and 3 targeted performance optimization through **OLAP/MOLAP** operations, **join evaluation**, **index comparison**, **horizontal partitioning**,

materialized views, **runtime benchmarking**, and **DSS vs. OLTP** analysis. Together, these objectives ensured a scalable and efficient DW suitable for decision-support workloads.

3. ETL / ELT Workflow

The DW pipeline includes **Extract, Transform, Load (ETL)** and **Extract, Load, Transform (ELT)** processes:

3.1 ETL Process

- **Extract:** Raw data was extracted from multiple sources, including order logs, product catalogs, and customer databases.
- **Transform:** Data cleaning, deduplication, type conversion, and business rule validation were applied.
- **Load:** Cleaned data was loaded into staging tables in PostgreSQL for further processing.

3.2 ELT Process

- **Extract and Load:** Raw tables were directly loaded into DW staging tables.
- **Transform:** Transformation logic, such as surrogate key generation, date formatting, and aggregation, was applied inside the DW using SQL.
- **Advantages:** ELT leverages DW compute power and reduces data movement, suitable for large datasets.

3.3 Workflow Highlights

- **Staging Layer:** Raw data temporarily resides here for validation.
- **Dimension Tables:** Product, Customer, Date, and Author dimensions store descriptive attributes.
- **Fact Tables:** Fact sales contain measurable metrics like revenue, quantity, and unit price.
- **Load Automation:** Scripts automate daily/weekly incremental loads.

The combination of ETL and ELT ensures **data quality, consistency, and timely availability** for analytical queries.

4. OLAP and MOLAP Concepts

4.1 OLAP (Online Analytical Processing)

- Supports complex analytical queries such as aggregations, roll-ups, and drill-downs.
- Operates on **fact and dimension tables** for multidimensional analysis.
- Examples: Monthly revenue trends, top-selling products, revenue share by category.

4.2 MOLAP (Multidimensional OLAP)

- Data is pre-aggregated into **cubes** for faster query response.
- Materialized views in this project serve as a **MOLAP-like cache** for heavy aggregation queries.
- Enables quick slicing and dicing without scanning the entire fact table.

4.3 Benefits for DSS

- OLAP structures allow **ad-hoc querying, trend analysis, and reporting**.
- Pre-aggregation (materialized views) reduces load on the database engine for repeated analytical queries.

5. Joining Techniques

Efficient join strategies are crucial for DW queries due to large fact tables:

- **Nested Loop Join (NLJ):** Simple but expensive for large tables; suitable for small tables or highly selective filters.
- **Hash Join:** Efficient for equi-joins on large fact tables; builds a hash table on the smaller table.
- **Sort-Merge Join:** Good for already sorted datasets; often used in combination with partitioning.

DSS vs OLTP Considerations:

Aspect	DSS / OLAP	OLTP
Join Type Preference	Hash Join, Sort-Merge Join	Nested Loop Join
Query Focus	Aggregations, multi-dimensional analysis	Row-level operations, transactions
Data Volume	Very large fact tables	Small to moderate
Index Usage	Composite, B-Tree, bitmap, materialized views	Primary / secondary for unique lookups
Performance Goal	Fast aggregates, trend analysis	ACID-compliant insert/update/delete

Indexes and pre-aggregations are essential for DSS workloads, enabling **high-performance analytical queries** without affecting OLTP operations.

6. Performance Optimizations/Visualization and Insights

In Iteration 3, the following optimizations were applied:

6.1 Indexing Techniques

- **B-Tree Index:** For high-cardinality columns (date_key) to accelerate range queries.
- **Bitmap-like Index (via B-Tree):** For low-cardinality columns (category) to optimize grouping and aggregation.

6.2 Horizontal Partitioning

- Partitioned fact_sales_elt_partitioned by year (2010–2013).
- Partition pruning ensures only relevant partitions are scanned.

6.3 Materialized Views

- mv_monthly_category_sales pre-aggregates monthly revenue, average unit price, and unique customer counts.
- Reduces repeated computation on the large fact table.

6.4 Index Type Comparison

Index Type	Attribute(s)	Purpose	Runtime Before	Runtime After	Runtime Improvement (%)	Observation
B-Tree	fact_sales_elt.date_key	Range filtering for time-based queries	0.171 s	0.103 s	**39.8%**	Efficient for large fact table range scans
Bitmap-like (B-Tree)	dim_product_elt.category	Low-cardinality categorical filtering	1.050 s	0.752 s	**28.4%**	Faster grouping and aggregation queries. Postgre SQL uses bitmap scans internally over btree.
Primary Key	dim_customer_elt.customer_key	Direct lookup by unique key	0.142 s	0.122 s	**–12.3%** (small dataset – overhead)	Quick look-up by primary key
Secondary	dim_customer_elt.customername	Non-unique search filter	0.150 s	0.110 s	**59.7%**	Efficient for descriptive attribute filters and helps equality

Index Type	Attribute(s)	Purpose	Runtime Before	Runtime After	Runtime Improvement (%)	Observation
						filters on descriptive attributes.
Composite	fact_sales_elt_partitioned (product_key + date_key)	Multi-column filtering	0.166 s	0.122 s	**14.1%**	Optimal for queries filtering by product and date together
Horizontal Partitioning	fact_sales_elt_partitioned (yearly)	Partition pruning	0.114 s	0.128 s	**26.6%**	Minor improvement in small dataset; major for large tables. Investigate with larger data or different partition bounds.
Materialized View	mv_monthly_category_sales (month, category)	Cache aggregated data	0.261 s	0.105 s	**26.5%**	Pre-aggregated queries are much faster
Sequential Scan	fact_sales_elt	Baseline full table scan	0.414 s	0.116 s	**71.9%**	Avoided with indexes, partitions, or materialized views.

Short notes about the negative / unexpected result (Horizontal Partitioning)

- The partitioning test showed a small slow-down (-12.28%). This can happen when:
  - The dataset is small so partition pruning overhead outweighs benefits.
  - The planner chooses a plan that does extra checks or doesn't prune effectively for that specific predicate.
  - Partitions are not aligned to query ranges or statistics are stale.
- Actionable next steps: increase sample size, run ANALYZE on parent and partitions, try more realistic date ranges, or test with EXPLAIN (ANALYZE, BUFFERS) to inspect IO and planner decisions.

Insights:

- B-Tree indexes excel for high-cardinality range queries.
- Bitmap-like indexes are effective for low-cardinality aggregations.
- Composite indexes provide significant gains for multi-filter queries.

- Materialized views and partitioning drastically improve OLAP query performance.

7. Analytical Queries / Insights

- **Top Customers:** Aggregation across partitioned fact table and customer dimension demonstrates join efficiency.
- **Monthly Product Revenue Trends:** Materialized views reduce computation cost for top-selling products per month.
- **Revenue Share by Category:** Low-cardinality indexing accelerates group aggregations.

Observation: Combining partitioning, indexing, and materialized views provides **optimal DSS query performance**, minimizing scan times and maximizing throughput.

8. Summary and Key Points

1. **Indexing Techniques:**
    - B-Tree for high-cardinality ranges
    - Bitmap-like for categorical dimensions
    - Composite indexes for multi-column queries
2. **Partitioning Strategy:**
    - Horizontal partitioning by year enables pruning and efficient scans
3. **Materialized Views:**
    - Pre-aggregated data speeds up repeated heavy analytical queries
4. **Index Type Comparison:**
    - Primary key: efficient for unique lookups
    - Secondary index: suitable for non-unique descriptive filters
    - Composite index: significant improvement for multi-filter queries
5. **OLAP / DSS Benefits:**
    - Optimized for aggregates, trends, drill-down analysis
    - Supports large historical datasets efficiently
    - Reduces compute time compared to OLTP query patterns
6. **Joining Techniques for DSS vs OLTP:**
    - DSS: prefers **hash joins and sort-merge joins** for large tables and aggregations
    - OLTP: uses nested loop joins for small transactional operations

Conclusion

This final report demonstrates the complete lifecycle of designing, building, and optimizing an E-Commerce Data Warehouse. It also demonstrates **substantial performance improvements** through **indexing, partitioning, materialized views, and index-type comparisons**, complementing the ETL/ELT workflows. The DW now supports **efficient OLAP queries**, large-

scale aggregations, and complex analytics for DSS workloads, while separating OLTP-like transactional operations to maintain system performance.

Key achievements:

- Implemented robust ETL/ELT pipelines ensuring clean, consistent, and integrated data.
- Developed OLAP-ready star schema for analytical workloads.
- Applied indexing, partitioning, and materialized views to significantly improve query performance.
- Compared join types and index types to justify DSS-appropriate optimization strategies.
- Measured real improvements using EXPLAIN ANALYZE data.