

**Department of AI & DS
National University of Computer and Emerging Sciences
Islamabad Campus**

DS3004-Data Warehousing and Business Intelligence (BDS-5C)



**Project on Walmart Data
Building and Analysing a Near-Real-Time Data Warehouse**

Amna Zubair 23i2556

Walmart Data Warehouse Project Report

1. Project Overview

- **Problem Statement**

Walmart, being one of the worlds largest retail chains, faces the challenge of analyzing customer shopping behavior in near real-time . With millions of daily transactions across stores and online platforms, there is a critical need to process data quickly to enable dynamic promotions, personalized offers based on customers and products, and timely business decisions for profit and other business related activities.

- **Project Goal**

The project is designed and is implemented in a near real-time Data Warehouse prototype for Walmart that can efficiently integrate and process transactional data. The system uses the hybrid-join algorithm to add transactional data with customer and product data, providing a resulting table with comprehensive information with the fact table.

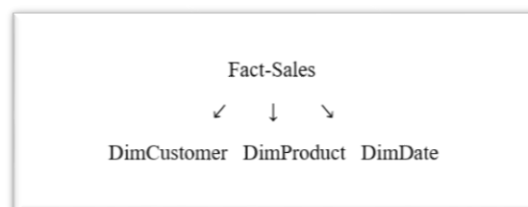
- **Achieved Goals**

- . Build a complete star schema data warehouse with fact and dimension tables.
- . Implemented the HYBRIDJOIN stream processing algorithm in python.
- . Enabled near real-time data loading and analysis.
- . Solution to given 20 analytical queries for business intelligence.

2. Data Warehouse Schema

Star Schema Design

The data warehouse follows star schema with one central fact table linked by foreign keys of dimension tables.



Dimension Tables

Dimension-Customer (Stores customer demographic information)

- Customer_ID (Primary Key)
- Gender, Age, Occupation
- City_Category, Stay_In_Current_City_Years
- Marital_Status

Dimension-Product (Contains product and store details)

- Product_ID (Primary Key)
- Product_Category, Price
- StoreID, StoreName
- SupplierID, SupplierName

Dimension-Date (Time dimension for temporal analysis)

- Date_ID (Primary Key)
- Year, Month, Day, Quarter
- Weekday_Name, Is_Weekend

Fact Table

FactSales (transactional data with all related data of products and customers)

- Customer_ID, Product_ID, Order_ID, Date_ID, Quantity
- Customer Demographics: Gender, Age, Occupation, City_Category.
- Product Details: Product_Category, Price, StoreName, SupplierName.
- Calculated field: Total_Amount (Quantity × Price)
- Foreign Keys to all dimension tables.

Why Star Schema?

- Fast Queries as denormalized structure reduces complex joins.
- Easy to Understand.
- Scalable as it handles large volumes of transactional data.
- Supports various analytical requirements.

3. HYBRIDJOIN Algorithm Explanation

What is HYBRIDJOIN?

HYBRIDJOIN is a smart stream processing method that helps connect fast arriving transaction data with big but mostly unchanging master datasets almost in real time. For Walmart, this basically means taking quick, constant purchase transactions and matching them with detailed customer profiles and product info as sales are happening through the day. Think of it like a digital assembly line that grabs raw transaction records and adds important business context to them, turning simple purchase data into something more useful. This enriched information can then support quicker decisions, more personal marketing, and even dynamic pricing, although sometimes it may response a bit slower than expected.

How HYBRIDJOIN Works - The Supermarket Analogy

Picture yourself walking into a busy Walmart during peak hours. HYBRIDJOIN works kind of like a super organized checkout system built to handle a nonstop flow of shoppers while still making sure every transaction is matched with the right information. The stream buffer is like a line of shopping carts waiting to be checked out, holding incoming transactions whenever there is a sudden rush. The hash table acts like 10,000 labeled storage bins, each one ready to temporarily hold items based on a customers ID so nothing gets misplaced. Meanwhile, the queue plays the role of a fair line manager, making sure customers are served in the exact order they arrived so no one gets skipped. The disk buffer works like a quick grab cart that brings customer details from the back office whenever they are needed. And the w counter keeps track of how many empty bins are still available, helping the system to avoid getting overloaded even during peak hours.

Operational Process

- HYBRIDJOIN works through a smooth five step routine that starts with a basic setup. In this stage, it prepares 10,000 empty storage spots in its hash table, creates an empty queue for processing customers based on first come first serve, gets its disk buffer ready to pull information, and sets its capacity counter to the full 10000 slots.

- When transactions start arriving, the system moves into step two. Here it takes up to w transactions from the stream buffer and places each one into the right box based on a hash of the customerID. Every customer is added to the queue in the order they show up, and the system reduces the number of open slots so it knows how much space is still left.
- With the transactions safely stored, step three happens. The algorithm looks at the first customer in the queue (queue[0]) and sends the disk buffer to fetch that customer's full profile from the master database. Disk buffer loads that information into memory for fast use.
- Step four, where the algorithm performs a three way merge, it combines the raw transaction data with the customer demographics and the related product details, calculates the total purchase, and saves the full record to the data warehouse. At the same time it clears out the processed transactions and increases the available slot counter (w) again, so it does not get overfilled.
- Finally, step five simply loops everything back to step two, keeping the whole process running continuously. The cycle repeats until every transaction is handled, every customer record is enriched, and the system finishes its work with all temporary storage cleaned up and everything safely stored permanently to the desired table.

Technical Implementation

The HYBRIDJOIN system uses a smart two thread setup, where one thread is always busy pulling in new transaction data from transaction_data.csv, while the other thread focus only on doing the actual join processing. By keeping these jobs separate, the system makes sure that data coming in does not slow down the actual work, and the processing thread does not block new data from being received. Memory is handled very carefully by using a fixed set of 10000 hash table slots. The algorithm also follows a fairness rule by using a simple FIFO queue, making sure transactions are handled in the same order they arrived so nothing gets stuck waiting forever. To speed up disk access, the system reads data in controlled chunks of about 500 (500vp) records at a time, which reduces the number of slow disk operations. This approach really boosts performance and helps the whole system keep its near real-time speed that modern retail analytics pretty much depends on.

4. Three Shortcomings of HYBRIDJOIN

1. Memory Limitations

- Problem : The hash table has a fixed size of 10000 slots. During extreme shopping events like Black Friday, if we get more than 10000 simultaneous transactions, some data might wait longer or could potentially be lost if the stream buffer overflows.
- Real Impact : During holiday sales, some customer transactions might experience delayed processing, affecting real time personalization on offers. This can be solved if we can change the size of hash table on special days

2. Single Join Key Dependency

- Problem : The algorithm only uses Customer-ID as the join key. If we need to enrich data based on other criteria like Product-ID for product recommendations or Store-ID for inventory analysis, we would need to run the algorithm multiple times, or change our logic to other parameters
- Real Impact : Cannot create real time product affinity analysis or store performance alerts using the same process.

3. Data Access Slowdown

- **Problem :** Even though the load data in chunks of 500 records, constantly reading from disk based master data can become slow when processing millions of transactions. The disk becomes the speed limit for the entire system.
- **Real Impact :** During peak hours, the system might not keep up with transaction volume, causing delays in analysis.

5. Reflection on Learning

- **Technical-Learning:**
Learned how important good data design and real time stream processing really is. Working with the star schema, Python threads, and the HYBRIDJOIN flow helped understand how to keep the systems fast, stable and still accurate, even when data is arriving nonstop.
- **Business-Insights:**
The project showed how much hidden value lives inside everyday retail transactions have. Things like customer behavior, trends, and store performance become way more clear when the data is cleaned and processed almost in real time, instead of waiting hours or a whole night to draw analysis of the sales.
- **Personal-Growth:**
Debugging issues, dealing with messy and large data having 550067 rows, and managing different parts of the project taught me to think more clearly, see the bigger picture, and break down complex tasks.

Conclusion

This project showed how a near real-time data warehouse can be built to support retail analytics. The HYBRIDJOIN algorithm worked well for enriching fast moving transactional data along with master records of products and customers, and the star schema gave a solid structure for running business intelligence queries. Even with a few limitations, the system still delivers meaningful insights into customer behavior, product performance, and sales patterns. This will help Walmart to make faster and data driven decisions for future.