## 1. Assessment Task

The student is required to design, implement, and analyze a **near-real-time Data Warehouse** (DW) prototype for **Walmart**. This task involves the creation of a system capable of integrating and processing transactional data efficiently to enable timely business insights and decision-making.

## 2. Project Overview

**Walmart**, one of the world's largest retail chains, serves millions of customers daily across its stores and online platforms. To optimize sales and enhance customer satisfaction, Walmart needs to analyze shopping behavior in **near real-time**, enabling dynamic promotions and personalized offers. An abstract level overview of Wallmart DW is shown in Figure 1.

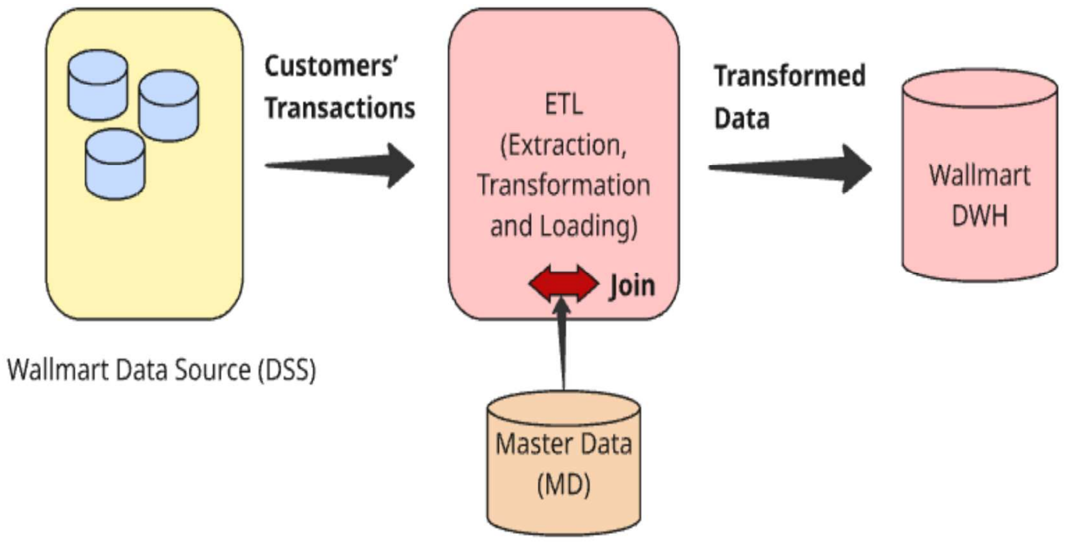

Figure 1: An overview of Wallmart DW

To achieve this near-real-time functionality, the system must include **ETL (Extraction, Transformation, and Loading) tools** capable of processing data streams dynamically. Since data generated by customers is often **incomplete** for DW requirements, it must be **enriched during the transformation phase** of ETL.
For example, additional details such as customer demographics or product attributes can be integrated from Master Data (MD) as illustrated in *Figure 2*.

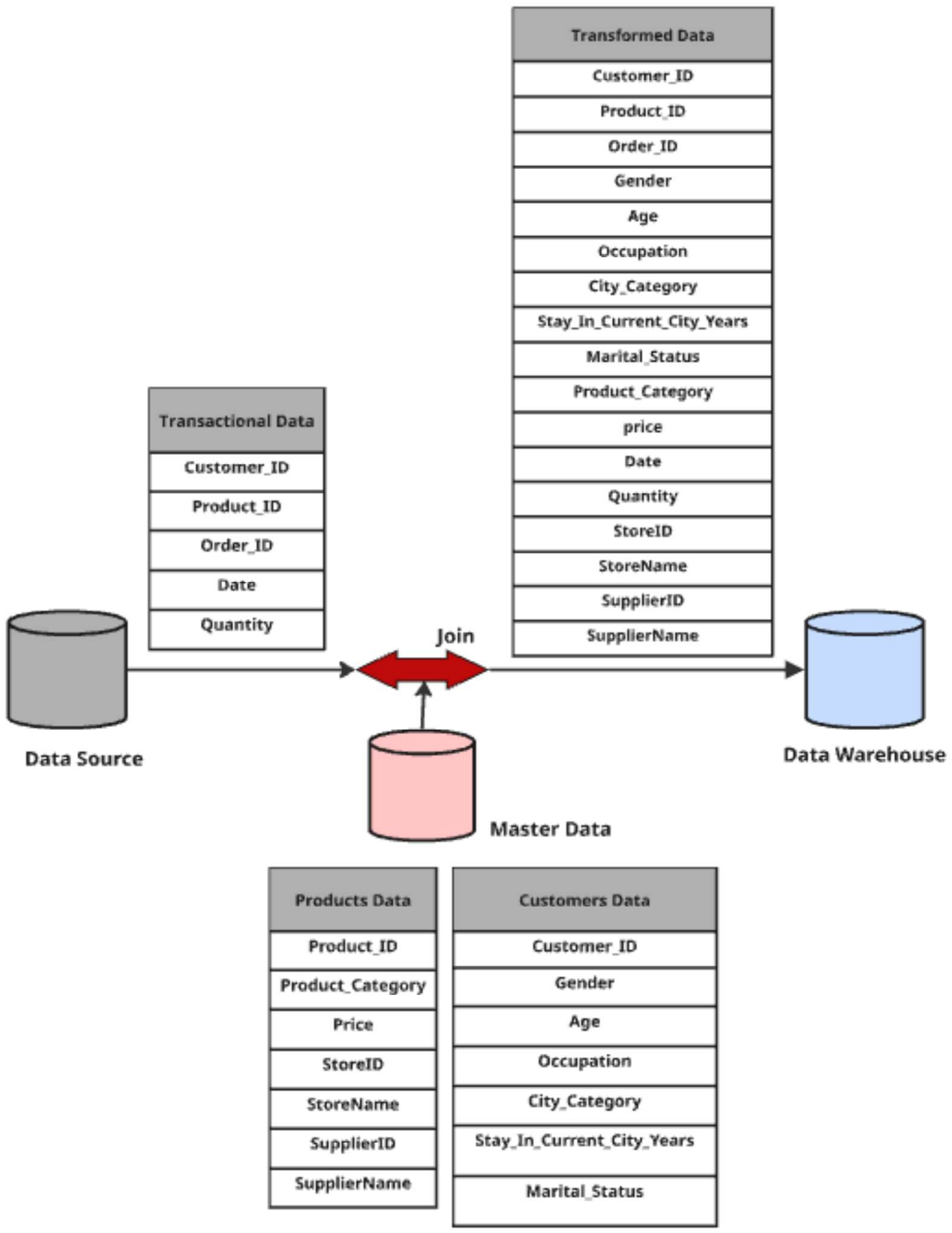

Figure 2: Data Enrichment Process

To implement this **data enrichment** feature, a **Stream-Relation Join Operator** is required in the transformation phase**.** Several algorithms exist to perform this type of join operation;
In this project, you will **implement HYBRIDJOIN (Hybrid Join) algorithm** in **Python**, enhancing its functionality to support near-real-time DW operations efficiently**.**

## 3. HYBRIDJOIN

HYBRIDJOIN is a stream-based join algorithm designed for scenarios like near-real-time data warehousing, where you need to join a fast-arriving, potentially bursty data stream (S) with a large, disk-based relation.

## Main Components:

**Stream Buffer**: A small buffer to temporarily hold incoming stream tuples if the algorithm can't process them immediately. This prevents loss of data in bursty scenarios.

**Hash Table (H)**: A multi-map (allows multiple entries per key) that stores stream tuples. Each entry also includes a pointer (address) to a corresponding node in the queue. The hash table has a fixed number of slots (hS), limiting its memory usage. In this project you will allocate 10,000 slots to accommodate the stream tuples.

**Queue**: A doubly-linked list (for efficient random deletions) that stores the join attribute values (keys) from the stream tuples in FIFO order (oldest at the front). Each node also has pointers to its neighbors. The queue tracks the order of arrival for fairness in processing.

**Disk Buffer**: A memory buffer that holds a loaded partition p of size vP from R. This is part of the "Join Window." Each disk partition size will be 500 tuples.

**Stream Input (w):** The number of available slots in the hash table which are equal to the number of free up spaces in the previous iteration.

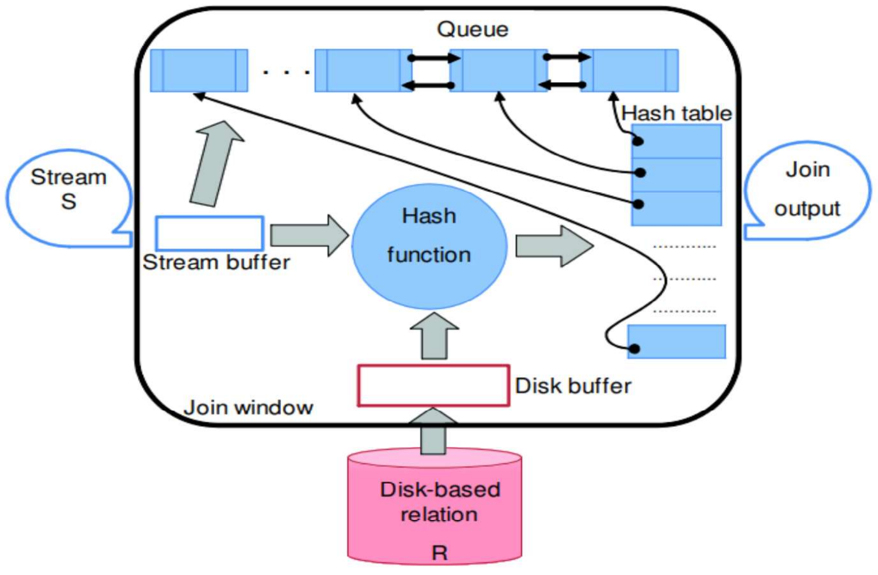

Figure 3: Data Structures used in HYBRID JOIN

## Working Principle

HYBRIDJOIN is a stream-based join algorithm designed to efficiently combine a continuous, potentially bursty data stream (S) with a large, disk-based relation (R.

It begins by initializing memory for its key components: a hash table with hS slots, a queue (a doubly-linked list for random deletions), a disk buffer, and a stream buffer.
The process runs in an endless outer loop where it first checks the stream buffer for incoming tuples, loading up to w (the number of available hash table slots) tuples into the hash table and adding their join attribute values to the queue. Each tuple is hashed using a function that maps its join key (e.g., Customer_ID) to a slot, allowing multiple matches in a multi-map structure. After loading stream data, the algorithm uses the oldest key from the queue to index into R, loading a relevant partition (size vP) into the disk buffer, leveraging R’s indexed and sorted nature to minimize I/O.

In the inner loop, the algorithm probes the hash table with each tuple from the loaded disk partition, generating join outputs for matches, deleting matched stream tuples from the hash table and their corresponding queue nodes, and incrementing w by the number of vacated slots. Unmatched tuples persist in the hash table and queue. The queue shrinks or grows based on join successes or new stream arrivals.

**Note: The stream buffer will continuously get transactional data as the HYBRIDJOIN is for the near real time data. A thread will be implemented which will continuously get data from the transactional_data.csv provided into the stream buffer independent of the join operation. Then there will be another thread which will implement the HYBRIDJOIN algorithm.**

## 4. Star Schema

The star schema (which you will use in this project) is a data modelling technique that is used to map multidimensional decision support data into a relational database. Star-schema yields an easily implemented model for multidimensional data analysis while still preserving the relational structures on which the operational database is built.

The star schema represents aggregated data for specific business activities. Using the schema, one can create multiple aggregated data sources that will represent different aspects of business operations. For example, the aggregation may involve total sales by selected time periods, by products, by stores, and so on. Aggregated totals can be total product sold, total sales values by products, etc. The basic star schema has three main components: facts, dimensions, attributes, and classification levels. Figure 2 can help you to determine the right components for your star schema.

## 5. Implementation of HYBRIDJOIN:

To implement the **HYBRIDJOIN** algorithm, you will perform the following steps:

1. The algorithm sets up a few tools: a hash table (like a box with hS compartments), a queue (a list), a disk buffer (a small memory space), and a stream buffer (to catch extra data). It starts with w (free compartments) set to hS.
2. It checks the stream buffer for new data (e.g., purchase records). It takes up to w pieces of data, puts them in the hash table (using a key like Customer_ID to pick compartments), and adds the keys to the queue. Extra data waits in the stream buffer. Then, w becomes 0.
3. It looks at the oldest key in the queue and uses it to grab a small chunk (vP size) of disk data (R) into the disk buffer.
4. It checks each disk chunk item against the hash table. If there's a match, it makes a combined result and loads it into the DW, removes the matched data from the hash table and queue, and adds the freed spaces to w.
5. It loops back to Step 2, taking new data and repeating the process with the next key, continuing as long as the stream runs or until the hash table is empty.

## 6. DW Analysis

**Q1. Top Revenue-Generating Products on Weekdays and Weekends with Monthly Drill-Down** Identifies the top 5 products by revenue, split by weekdays and weekends, with monthly breakdowns for a year.

**Q2. Customer Demographics by Purchase Amount with City Category Breakdown**
Analyzes total purchase amounts by gender and age, detailed by city category.

**Q3. Product Category Sales by Occupation**
Examines total sales for each product category based on customer occupation.

**Q4. Total Purchases by Gender and Age Group with Quarterly Trend**
Tracks purchase amounts by gender and age across quarterly periods for the current year.

**Q5. Top Occupations by Product Category Sales**
Highlights the top 5 occupations driving sales within each product category.

**Q6. City Category Performance by Marital Status with Monthly Breakdown**
Assesses purchase amounts by city category and marital status over the past 6 months.

**Q7. Average Purchase Amount by Stay Duration and Gender**
Calculates the average purchase amount based on years stayed in the city and gender.

**Q8. Top 5 Revenue-Generating Cities by Product Category**
Ranks the top 5 city categories by revenue, grouped by product category.

**Q9. Monthly Sales Growth by Product Category**
Measures month-over-month sales growth percentage for each product category in the current year.

**Q10. Weekend vs. Weekday Sales by Age Group**
Compares total sales by age group for weekends versus weekdays in the current year.

**Q11. Top Revenue-Generating Products on Weekdays and Weekends with Monthly Drill-Down**
Find the top 5 products that generated the highest revenue, separated by weekday and weekend
sales, with results grouped by month for a specified year.

**Q12. Trend Analysis of Store Revenue Growth Rate Quarterly for 2017**
Calculate the revenue growth rate for each store on a quarterly basis for 2017.

**Q13. Detailed Supplier Sales Contribution by Store and Product Name**
For each store, show the total sales contribution of each supplier broken down by product name. The output should group results by store, then supplier, and then product name under each supplier.

**Q14. Seasonal Analysis of Product Sales Using Dynamic Drill-Down**
Present total sales for each product, drilled down by seasonal periods (Spring, Summer, Fall, Winter). This can help understand product performance across seasonal periods.

**Q15. Store-Wise and Supplier-Wise Monthly Revenue Volatility**
Calculate the month-to-month revenue volatility for each store and supplier pair. Volatility can be defined as the percentage change in revenue from one month to the next, helping identify stores or suppliers with highly fluctuating sales.

**Q16. Top 5 Products Purchased Together Across Multiple Orders (Product Affinity Analysis)**
Identify the top 5 products frequently bought together within a set of orders (i.e., multiple products purchased in the same transaction). This product affinity analysis could inform potential
product bundling strategies.

**Q17. Yearly Revenue Trends by Store, Supplier, and Product with ROLLUP**
Use the ROLLUP operation to aggregate yearly revenue data by store, supplier, and product,

enabling a comprehensive overview from individual product-level details up to total revenue per store. This query should provide an overview of cumulative and hierarchical sales figures.

**Q18. Revenue and Volume-Based Sales Analysis for Each Product for H1 and H2**
For each product, calculate the total revenue and quantity sold in the first and second halves of the year, along with yearly totals. This split-by-time-period analysis can reveal changes in product
popularity or demand over the year.

**Q19. Identify High Revenue Spikes in Product Sales and Highlight Outliers**
Calculate daily average sales for each product and flag days where the sales exceed twice the daily
average by product as potential outliers or spikes. Explain any identified anomalies in the report, as these may indicate unusual demand events.

**Q20. Create a View STORE_QUARTERLY_SALES for Optimized Sales Analysis**
Create a view named STORE_QUAsweRTERLY_SALES that aggregates total quarterly sales by store,
ordered by store name. This view allows quick retrieval of store-specific trends across quarters, significantly improving query performance for regular sales analysis.

## 7. Tasks Break-Up

The following is the list of tasks that you need to complete for this project:

1. Identify appropriate dimension tables, fact table, and their attributes for the sales scenario presented in *Figure 2*. Based on that, create a star-schema for the Data Warehouse (DW) with appropriate Primary and Foreign Keys.
2. Implement the HYBRIDJOIN algorithm (using the steps described in *Section 5*) in python for successfully loading transactional data into the DW after joining it with the Master Data (MD).
3. Apply different analyses (as described in *Section 6*) on the DW using slicing, dicing, drill-down, and materialized view concepts.
4. Write a project report that includes the project overview, the DW schema, an explanation of the HYBRIDJOIN algorithm, any three of its shortcomings, and a reflection on what you learned from the project.

## 8. Files Provided

1. customer_master_data.csv
2. product_master_data.csv
3. transactional_data.csv

## 9. What to Submit

Each student must submit the following files:

1. Create-DW **–** An SQL script file to create the star schema for the DW.

   **Note:** Your script should remove any pre-existing schema before creating new tables.

2. Hybrid-Join **–** An Python project that implements the HYBRIDJOIN algorithm.

   **Note**: Your program should take the database credentials from the user at execution time.

3. Project-Report **–** A document file containing all the contents described in point 4 under the “Tasks Break-Up” section.
4. ReadMe **–** A text file describing step-by-step instructions to operate your project. Note: All the above files must be submitted in a zipped folder named using your name and student ID (e.g., `xyz_22i11111_Project`).

## 10. When to Submit

**Due Date: Monday, 17 Nov 2025, midnight.**
**Late Penalty:** Maximum late submission time is **24 hours** after the due date. In this case, a **15% deduction** will be applied.

## 11. Where to Submit

The project must be submitted through **Google Classroom**.
**Note:** Each student must complete the project **individually**. All project source files and report materials must be **unique** and prepared **independently**. Submissions will be checked using the **Turnitin** system, and any duplication or identical content will result in a score of **zero (0) marks**.

## Marking Guidelines

| Component | Description | Marks |
|---|---|---|
| **Create-DW** | SQL script file to create a star-schema for the Data Warehouse (DW). The script must drop existing tables, create all dimensions and fact tables, and apply all Primary and Foreign Keys. | /15 |
| **Hybrid-Join** | A  project that implements the HYBRIDJOIN algorithm. The Python file should implement all three ETL phases: extract from the TRANSACTIONS table, transform using the Master Data (MD), and load the records into the DW. | /30 |
| **Queries-DW** | SQL script file containing all the OLAP queries for the tasks outlined in Section 6 of the project brief. | /30 |
| **Project-Report** | A document file containing a project overview, the DW schema, an explanation of the HYBRIDJOIN algorithm, any three of its shortcomings, and a summary of lessons learned from the project. | /20 |
| **Read-Me** | A text file providing step-by-step instructions to operate the project. **(Note: 5 marks will be deducted if this file is missing.)** | /5 |
| | **TOTAL MARKS** | **/100** |
| | **Late Submission Penalty** | **-15%** |