Data Warehousing Course Project

Active Warehouse using MESHJOIN

Submitted by:
Muhammad Farjad Ilyas
18I-0436

Submitted to:
Dr. Asif Naeem

The National University of Computer and Emerging Sciences

Project Overview

Problem Domain

Applications which require a high extent of consistency between operational data and analytics require an active warehouse that can reflect changes in the operational data at shorter time intervals. This project aims to implement an ETL process that is appropriate for an Active Warehouse. It targets a common scenario of a fast stream of new operational data and a collection of Master Data on disk. The stream of new data must be enriched with the stored Master Data in order to include necessary information for analysis with each stream record.

The scenario for this project is a chain of stores, with different outlets and a variety of products for sale in each store. The operational data provided is transactional data. The major details typical in a store scenario are provided and can be seen in Figure 1.

TRANSACTIONS								
Attributes	TRANSACTION_ID	PRODUCT_ID	CUSTOMER_ID	CUSTOMER_NAME	STORE_ID	STORE_NAME	T_DATE	QUANTITY
Data type and size	VARCHAR2(8)	VARCHAR2(6)	VARCHAR2(4)	VARCHAR2(30)	VARCHAR2(3)	VARCHAR2(20)	DATE	NUMBER(3,0)

MASTERDATA					
Attributes	PRODUCT_ID	PRODUCT_NAME	SUPPLIER_ID	SUPPLIER_NAME	PRICE
Data type and size	VARCHAR2(6)	VARCHAR2(30)	VARCHAR2(5)	VARCHAR2(30)	NUMBER(5,2) DEFAULT 0.0

Figure 1: Operational Database Schema

Key Issues

A significant problem in the scenario mentioned above is the gap between the access speeds for data arriving as part of a stream, and data stored in the Master Data collection on disk. Accessing data from the stream is much faster than the Disk I/O required to access Master Data. This is the bottleneck the MESHJOIN algorithm aims to solve.

Data Warehouse Schema

Every Data Warehouse, regardless of whether it is a conventional or Active Warehouse, and the algorithms being used requires a Data Warehouse schema to hold aggregated data used for analytical queries. The schema for the scenario mentioned earlier is depicted in Figure 2.

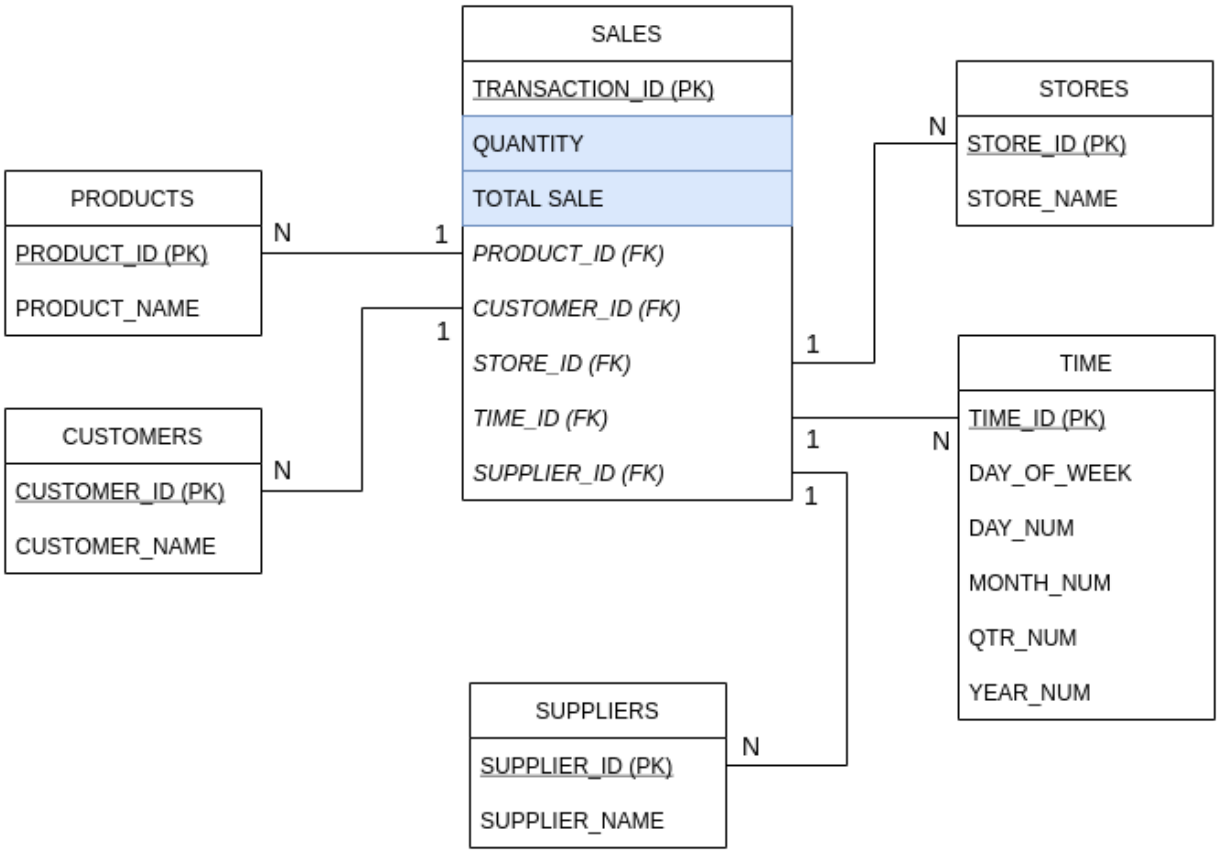

Figure 2: Star Schema for this project's Data Warehouse

MESHJOIN

Approach

The MESHJOIN algorithm aims to amortize the cost of accessing Master Data for the purpose of joining it with the data arriving in a stream. It achieves this by leveraging the following two concepts:

1. The Disk I/O cost is divided over multiple sets of records arriving from the stream, i.e, one-to-many relationship between the number of passes over the Master Data and the number of joins carried out with the streaming data

2. Disk I/O cost is minimized by avoiding direct access to Master Data. Master Data is read sequentially when it needs to be loaded to execute joins. The fact that sequential reads are faster is taken advantage of.

Another important consideration for the MESHJOIN algorithm is limited memory. The algorithm makes the realistic assumption that Master Data is often very large in size. Hence, the algorithm assumes a Master Data buffer, a fixed block of memory which is allocated for keeping a batch of Master Data in memory for the purpose of executing a join. The algorithm also includes a Stream Buffer. However, this is typical for stream processing.

Algorithm details

The algorithm keeps a large chunk of streaming data in memory compared to Master Data. Hence, reading one batch of Master Data as part of a slow operation yields many more joined tuples. However, it is important that MESHJOIN is correct and doesn't 'miss' any joins that must be executed.

In order to consolidate both of this, MESHJOIN uses a queue to simulate a sliding window over the streaming data. At each time step, the window slides over the streaming data, moving one step closer to the source of the stream, as seen in Figure 3.

Figure 3: A snapshot of the state of MESHJOIN's queue and Master Data Buffer for each time step

Join Operation

At each time step ‘t’, a new batch of Master Data is loaded into the buffer in a cyclic manner. In tandem, a new stream partition is inserted into the queue and the Hash Table. Each stream partition is then joined with this new batch of Master Data. This is where the Hash Table pays dividends, since it stores the transactions present in the queue in buckets, based on the product id they reference. Hence, for all tuples stored in the Master Data, the set of transactions corresponding to the product id referenced in each Master Data tuple can be retrieved from the Hash Table in effectively constant time. This means that the join operation itself occurs in linear time.

Removing joined records from memory

Each stream partition contains pointers to the stream data. If the size of the queue, sliding window, and the number of partitions of Master Data are the same, and equal to N. Then, after N steps, all the stream records in S1 have been joined with the entire Master Data. Hence, these records can be discarded. The pointers to stream tuples present in S1 can be used to remove the tuples from the hash table, then S1 itself can be removed from the queue.

Shortcomings

1. The **coupling** between the size of the Master Data and the size of the disk buffer. If the size of the master data increases, the size of the disk buffer must increase as well since the number of partitions are fixed. This is another parameter in the cost model proposed by the authors of MESHJOIN and it is one that prevents optimal distribution of memory available to MESHJOIN. R-MESHJOIN (Reduced MESHJOIN) is a development of MESHJOIN which removes this dependency and enables better distribution of memory between components.

2. MESHJOIN’s performance, measured by its service rate drops when the streaming data is **skewed**, i.e, the streaming data is not distributed evenly with respect to the join attribute. Which results in the join being carried out having low selectivity. In our case study, this translated to a product ID being referenced in many more transactions relative to other product IDs. Hence, the list corresponding to the more frequent product ID value will be big in the hash table and the join will be less efficient.

3. MESHJOIN doesn’t make use of a cache, which can be used to supplement both the queue and the Master Data buffer so that streaming records which hit the cache can be joined straight away (like the SSIJ algorithm)

Takeaways

Summary: The MESHJOIN algorithm operates similarly to the working of a Field Programmable Gate Array. Instead of slowing down the data stream to be processed, MESHJOIN adds more ‘processors’ in the form of the different stream partitions in the sliding window / queue. Since one Master Data batch joins with multiple stream partitions the cost of Disk I/O is divided amongst the relatively large set of stream records which are being ‘processed’ in parallel. This parallelism is enabled by the Hash Table,

which efficiently yields all matching transactions for each Master Data tuple, regardless of its arrival time. The queue is used to enforce memory restrictions on the algorithm.

Effect on Warehouses: MESHJOIN enhances the capability of a Data Warehouse by offering low memory consumption in relation to the streaming rate the algorithm has to service.