Report on Near-Real-Time Data Warehouse for METRO Shopping Store

Project Overview

The project aimed to design, implement, and analyze a near-real-time Data Warehouse (DW) prototype for METRO shopping stores in Pakistan. This DW is intended to facilitate the real-time analysis of customer shopping behaviors, enabling METRO to optimize its sales strategies by providing dynamic insights, such as identifying top-selling products or tracking revenue trends.

To achieve this, a near-real-time ETL (Extraction, Transformation, Loading) pipeline was developed to ensure customer transactions were promptly integrated into the DW. The transformation phase of ETL involved enriching incomplete transactional data using master data. The prototype also incorporated the implementation of the MESHJOIN algorithm to perform the Stream-Relation join operation, which is pivotal for combining real-time streams with static master datasets.

Data Warehouse Schema

The project adopted a **star schema** for the DW design due to its simplicity and efficiency in querying multidimensional data. The schema comprised:

```
drop schema if exists `metroDB` ;
create schema `metroDB` ;
use metroDB;
drop table if exists `customers_data`;
create table customers_data (
   CustomerID int primary key,
   CustomerName VARCHAR(100) not null,
   Gender CHAR(1) CHECK (Gender IN ('M', 'F'))
);
drop table if exists `products_data`;
create table products_data (
   ProductID int primary key,
   ProductName VARCHAR(100) not null,
   ProductPrice decimal(10, 2) not null,
   SupplierID int not null,
         SupplierName varchar(100) not null,
   StoreID int not null,
         StoreName varchar(100) not null
);
drop table if exists `Suppliers_data`;
create table Suppliers_data (
    SupplierID int primary key,
    SupplierName VARCHAR(100) not null
);
drop table if exists `Stores_data`;
create table Stores_data (
    StoreID int primary key,
    StoreName VARCHAR(100) not null

);
drop table if exists `Dates_data`;
create table Dates_data (
            DateID INT AUTO_INCREMENT PRIMARY KEY,
    FullDate DATE NOT NULL,
    Year INT NOT NULL,
    Quarter INT NOT NULL,
    Month INT NOT NULL,
    MonthName VARCHAR(10) NOT NULL,
    Day INT NOT NULL,
    DayOfWeek INT NOT NULL,
    DayName VARCHAR(10) NOT NULL,
    IsWeekend BOOLEAN NOT NULL,
    Season VARCHAR(10) NOT NULL
);
drop table if exists `Sales_data`;
create table Sales_data (
    order_id int primary key,
    DateID int not null,
    ProductID int not null,
    CustomerID int not null,
    Quantity int not null,
    Sale DECIMAL(10, 2) not null,
    StoreID int not null,
    SupplierID int not null,
    FOREIGN key (DateID) references Dates_data(DateID),
    FOREIGN key (ProductID) references products_data(ProductID),
    FOREIGN key (CustomerID) references customers_data(CustomerID),
    FOREIGN key (StoreID) references Stores_data(StoreID),
    FOREIGN key (SupplierID) references Suppliers_data(SupplierID)
);
```

MESHJOIN Algorithm

The MESHJOIN algorithm, introduced by Polyzotis in 2008, is critical for performing a Stream-Relation join operation in the ETL transformation phase. The algorithm integrates streaming data (customer transactions) with static master data by leveraging memory and disk efficiently. Its key components include:

1. **Disk Buffer:** Cyclically loads partitions of the large master data into memory.
2. **Hash Table:** Stores streaming tuples for matching with master data.
3. **Queue:** Organizes streaming tuples based on arrival time and ensures each is processed with all master data partitions.
4. **Stream Buffer:** Temporarily holds incoming streaming data for processing.

The staggered processing pattern ensures overlapping residence times for streaming data, optimizing memory usage and join efficiency.

Three Shortcomings of MESHJOIN

1. **Memory Constraints:** The algorithm requires a careful balance of memory allocation for the hash table, disk buffer, and queue. Insufficient memory can degrade performance or cause data loss.
2. **Latency:** While MESHJOIN is efficient, its cyclic processing of master data partitions can introduce delays, especially when the master dataset is significantly large.
3. **Static Master Data Dependency:** MESHJOIN assumes that master data does not change frequently. Real-world scenarios involving dynamic updates to master data require additional handling, complicating the process.

Key Learnings

This project offered valuable insights into the design and implementation of a near-real-time data processing system. Key takeaways include:

1. **Understanding Star Schema:** The simplicity and efficiency of the star schema for querying multidimensional data make it an ideal choice for data warehousing.
2. **Importance of Efficient Joins:** Implementing MESHJOIN underscored the challenges of real-time data integration, including memory management and balancing speed with accuracy.
3. **Data Enrichment in ETL:** The transformation phase's role in enhancing raw transactional data with master data is crucial for meaningful analytics.
4. **OLAP Querying:** Writing and executing OLAP queries provided practical experience in extracting business insights from the DW.