Analytics Workflow in AWS

Background

One of their leading applications wishes to allow XYZ's clients to evaluate and understand the effectiveness of their supply chains. They want to derive analytics from their data

Core capabilities of their solution are: 1) data fidelity, 2) flexibility of data ingestion process and 3) optimal storage footprint.

Goal

You are tasked with using two different methods to process ingest, transform and provide analytics to customer XYZ and follow the AWS best practices
Method 1. Using Redshift for serving analytics
Method 2. Using EMR - Spark core for transforming and S3 for serving analytics reports.

Analytics Metrics.

Feel free to define up to 5 analytics metrics or reports based on the data provided and use Method 1 and Method 2 and assume things if required

Data

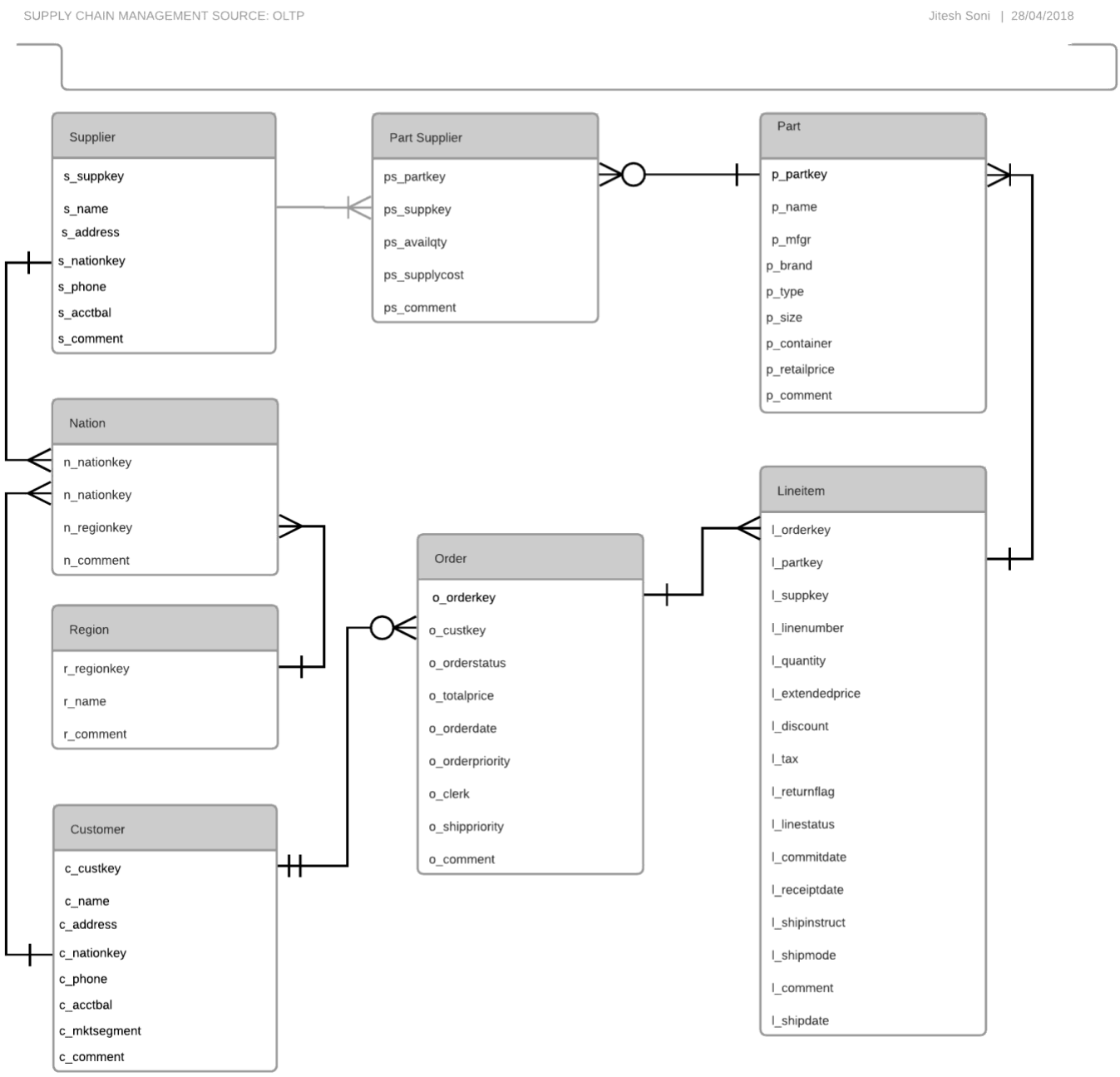
Sample data set is available in: s3://dory-public/tpch/1000/supplier/ in the AWS region of us-east-1.

Output

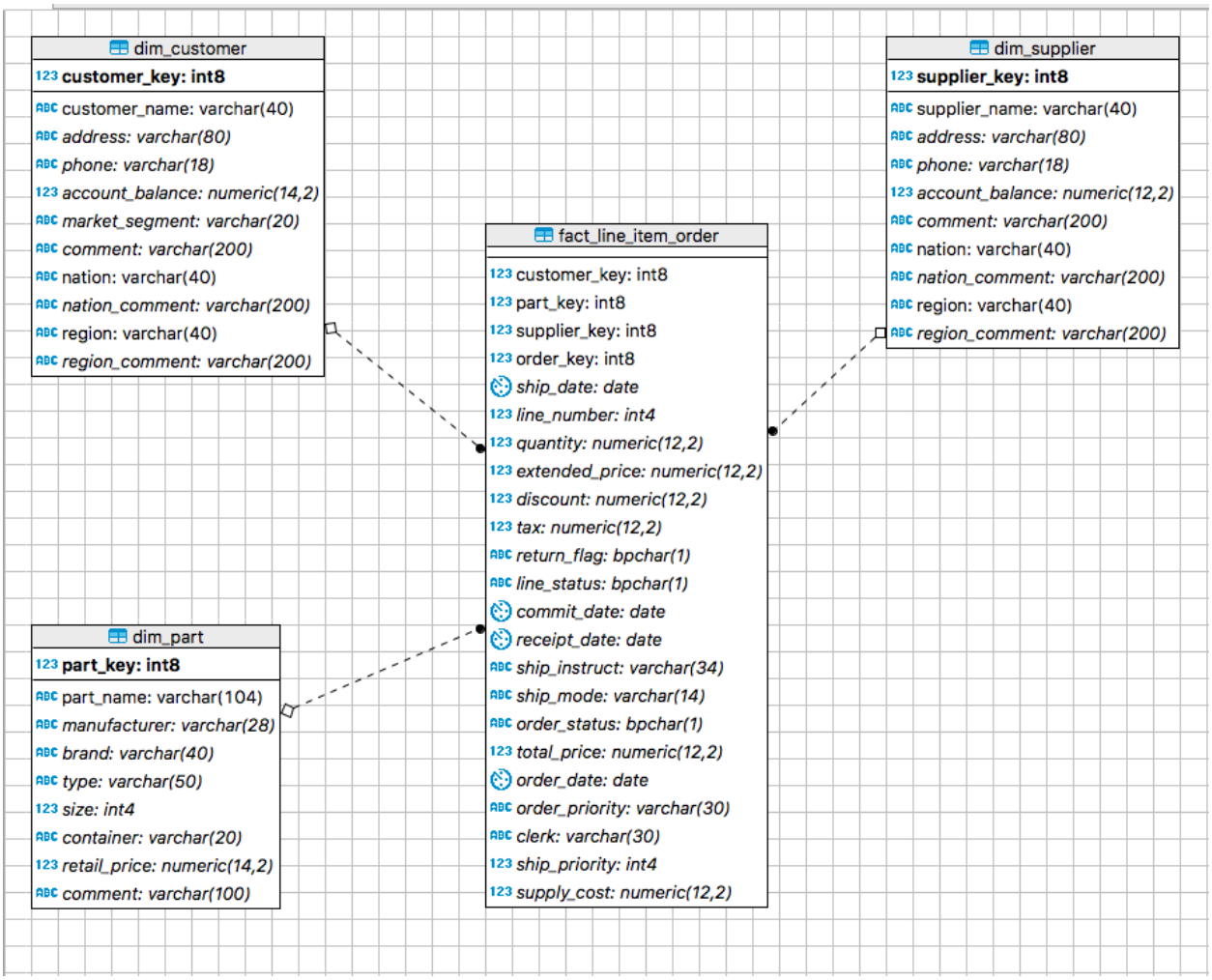
Please return a PDF document detailing your architecture, thought process and any code samples on code.amazon.com or git. In your architecture, please highlight how you are meeting XYZ's core capabilities.

Solution:

The source data provided is stored in parquet format and is represented in the ER diagram below.



The above schema design is not suited to meet the need of analytics; we would need to de-normalize the above schema. We are proposing a Star Schema as it's the best suited from an analytics perspective. Having a Star schema would improve client's experience along with providing data fidelity. The proposed Star schema does not include any semi-additive or non-additive attributes; thus, this schema could be directly exposed to customers via any data visualization tool like QuickSight, Tableau, etc. without the need of modifications.



Design consideration for Star Schema (OLAP):

- In a supply chain, most of the business questions would revolve around: Customer, Supplier, and Part. Thus, these 3 would be the dimension tables for our Star Schema.

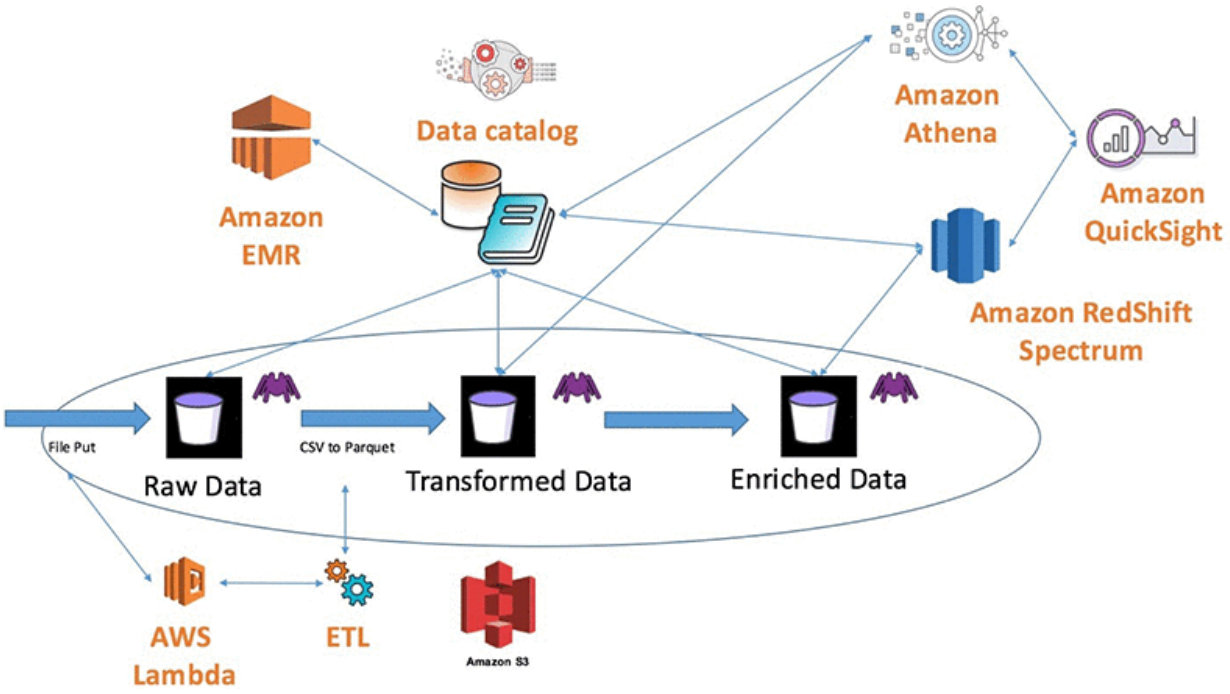
- One of the common design mistakes visible in the current schema design is header/line pattern. Thus, we would merge information available under
  Order and LineItem tables; into a single fact table called: 'fact_line_item_order'.

Ref: https://www.kimballgroup.com/2007/10/design-tip-95-patterns-to-avoid-when-modeling-headerline-item-transactions/

- Region and Nation dataset could be merged and stored together. Customer, and Supplier dataset reference Nation dataset in themselves, thus to avoid snowflaking we will merge the combined dataset of Nation and Region.

Method 1. Using Redshift for serving analytics:

Source data is in parquet; thus it cannot be directly ingested into Redshift as parquet format is not supported natively. However, Redshift Spectrum natively supports parquet format. We could crawl through the dataset with Glue and maintain table information with Glue catalog. Once tables were registered in Glue catalog, we could create external schema and tables inside Redshift and run our analytical queries. The architecture would be similar to:



Ref: https://aws.amazon.com/blogs/big-data/build-a-data-lake-foundation-with-aws-glue-and-amazon-s3/

Since Glue is not available in Canada region, thus we will solve it with a combination of S3, EMR (Apache Spark) and Redshift. We will use Apache Spark on EMR to read source Parquet files, transform data and will write data back to S3 in any of the formats supported by

Redshift. Pyspark code for ETL, would be discussed under method 2 of the document.

Ref: https://docs.aws.amazon.com/redshift/latest/dg/copy-parameters-data-format.html

Redshift Table Design: Distribution Key and Sort Key choice

Supplier Dimension:
This table has a row count of about ~10 million and is 10 times smaller in comparison to Customer and Part dimension. Although we are dealing with static data even for a real-world setting, the row count would grow slowly. Since this dataset is not suspect to huge peaks and given the current row count of the table DISTSTYLE ALL would be a great choice for the table.

Our customers would include supply chain managers who are generally responsible for specific geographic locations. Thus, having a Sort key on the column:'nation' would be a great choice. Furthermore, since this table would be joined with a Fact table on the column:'supplier_key', we would add it to the sort key as well. We will choose compound sort keys as they can also improve the performance of joins, group by and order by statements, while interleaved sort keys do not.

It's always recommended to sort on low cardinality column first followed by higher cardinality. Thus, our distribution and sorting would be:

' DISTSTYLE all
compound (nation, supplier_key)'

Part Dimension:
This table has about ~200 million rows and the fact table with sort key on the column:'part_key'. In the real world, you would expect the same, some common parts would be used much more often than others. Thus, we cannot distribute fact table on the column: 'part_key' hence it's not useful to distribute dimension table:'Part' on it either.

Given the size of this table, it would be preferable to distribute it evenly across the nodes. We would compound sort this table on brand, as brand level analysis is generally common in supply chain optimization. This is being designed keeping user queries pattern in mind and how teams or business divisions are structured. Again, keeping column:'part_key' as a sort key to boost up join performances.

'diststyle even
sortkey (brand, part_key);'

Customer Dimension:
This table has about ~150 million rows and does not show any signs of skew in the fact table. Thus, this table and the fact table would be ideal candidates for distribution with the column: 'customer_key'. This will help us with data locality when these tables are joined there will be no inter-node traffic; providing us with a superior and ideal join performance.

We would compound sort this table on the columns: 'nation' as we expect user queries to most likely to use this as a filter condition and column:'customer_key' to boost join performance.

'distkey (customer_key)
sortkey (nation, customer_key)'

Fact table:
As discussed earlier, we will distribute this table on 'customer_key'. Based on the sorting of our dimension table, column:'part_key' would be a good choice. Furthermore, we will sort this data on the column:'ship_date' as it will help in: help store our data in order and help with range filtering. Sorting on a date column is also helpful in lowering run times operations of vacuuming and sorting.

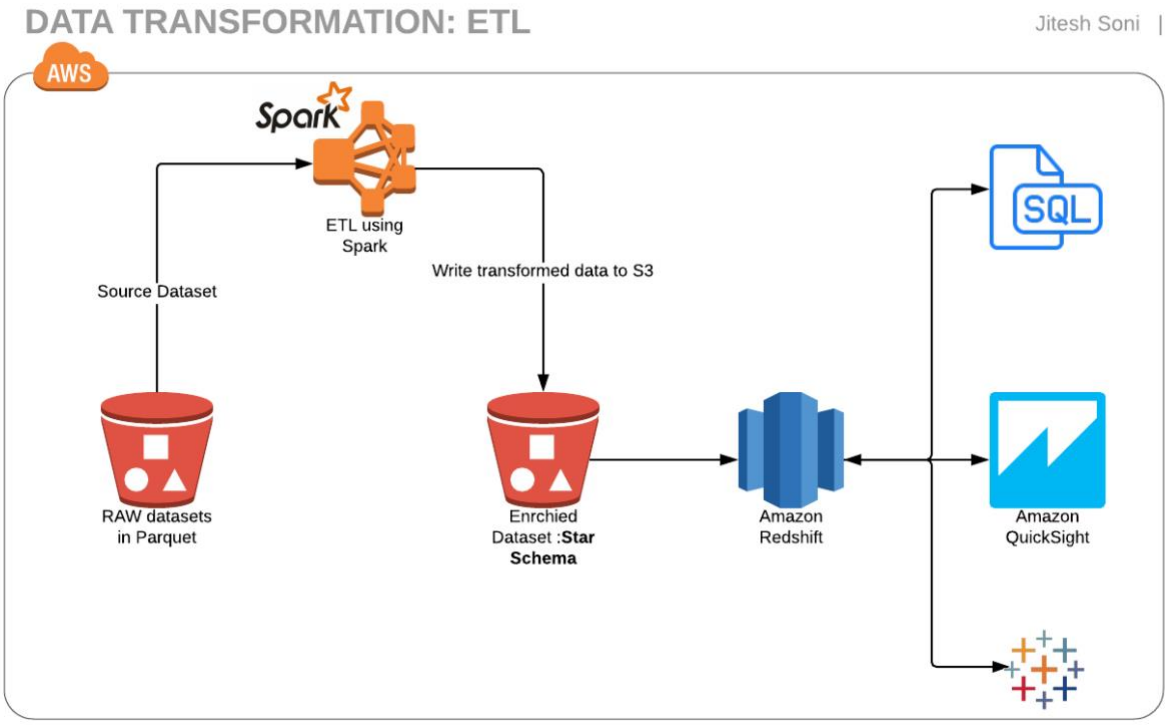
'distkey (customer_key)
sortkey (ship_date, part_key)'

DDL statements of the schema:
https://github.com/jiteshsoni/supply_chain_management/blob/master/sql-script/DDL.sql

Ref: How to optimize Star Schema on Redshift https://aws.amazon.com/blogs/big-data/optimizing-for-Star -schemas-and-interleaved-sorting-on-amazon-redshift/

Method 2. Using EMR - Spark core for transforming and S3 for serving analytics reports.

Architecture:



We would be using Apache Spark on EMR cluster, to transform our dataset into a Star schema. The transformed dataset could be copied to Redshift cluster for Method 1 and dataset could be also stored in S3 in formats like Parquet, JSON, CSV, etc.

After result set is stored in S3, customers could use any SQL Clients, BI tools and products like Athena, Redshift Spectrum, etc. to run analytics.

Pyspark ETL script for data processing:
https://github.com/jiteshsoni/supply_chain_management/blob/master/etl/etl.py

Using a combination of data frames, Star schema and parquet as a storage format provides:

1. Columnar compression
2. Having single joins between fact and dimension ensures that data is not misinterpreted by customers

3. Addition of new columns upstream will not impact our code as by using dataframe api, we only use the required columns.
4. Storing data in appropriate S3 paths: say YYYY/MMM/DD/hh/mm/ to take care of future design

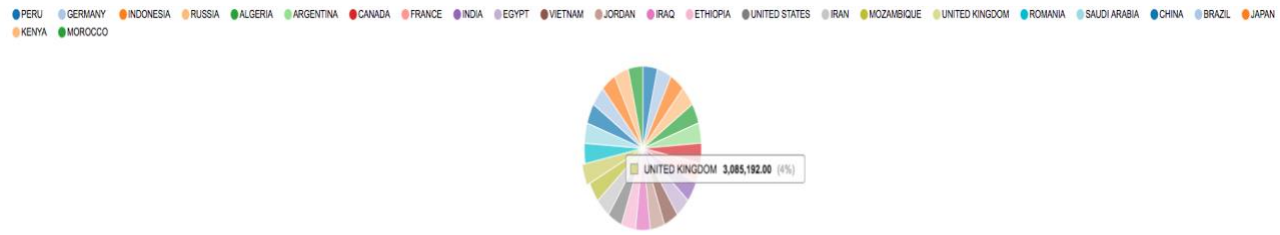
Analytical Reports:

After our transformed datasets are stored in S3, we have the freedom to choose between multiple technologies like Athena, Quicksight, Redshift, Tableau, etc. To check the effectiveness of dimensional modeling, we would define some useful analytical queries or reports and check for performance.

Report 1: Overall summary of supply chain performance: It calculates totals of Customers, Suppliers, Orders, Shipping Cost and Parts Shipped at Nation level.

https://github.com/jiteshsoni/supply_chain_management/blob/master/sql-script/Report1.sql

It's interesting to note that, supply chain performance in all nations/countries are performing very similar. It also proves that tool used to create this dataset uses a very good randomization algorithm.




```
XN HashAggregate  (cost=399044775.00..399044777.00 rows=200 width=182)
 ->  XN Subquery Scan volt_dt_0  (cost=392150475.00..396172150.00 rows=229810000 width=182)
     ->  XN HashAggregate  (cost=392150475.00..393874050.00 rows=229810000 width=53)
         ->  XN Hash Join DS_DIST_NONE  (cost=2048000.00..389277850.00 rows=229810000 width=53)
             Hash Cond: ("outer".customer_key = "inner".customer_key)
             ->  XN Hash Join DS_DIST_ALL_NONE  (cost=128000.00..30577825.00 rows=229810000 width=50)
                 Hash Cond: ("outer".supplier_key = "inner".supplier_key)
                 ->  XN Seq Scan on fact_line_item_order fact  (cost=0.00..2298100.00 rows=229810000 width=58)
                 ->  XN Hash  (cost=102400.00..102400.00 rows=10240000 width=8)
                     ->  XN Seq Scan on dim_supplier supp  (cost=0.00..102400.00 rows=10240000 width=8)
             ->  XN Hash  (cost=1536000.00..1536000.00 rows=153600000 width=19)
                 ->  XN Seq Scan on dim_customer cust  (cost=0.00..1536000.00 rows=153600000 width=19)
```

The above explain plan proves that data distribution is in an ideal state: Hash Join DS_DIST_NONE and Hash Join DS_DIST_ALL_NONE. No redistribution is required, because corresponding slices are co-located on the compute nodes'

Ref: Evaluating the Query plan
https://docs.aws.amazon.com/redshift/latest/dg/c_data_redistribution.html

Report 2: Top 5 customers for each market segment ordered on total orders
This report is to help us identify our winners (top customers)

https://github.com/jiteshsoni/supply_chain_management/blob/master/sql-script/Report2.sql

Explain plan:

```
XN Subquery Scan customer_rank_tab  (cost=1000383363755.08..1000387587755.08 rows=51200000 width=142)
 Filter: (customer_rank <= 5)
 ->  XN Window  (cost=1000383363755.08..1000385667755.08 rows=153600000 width=42)
       Partition: cust.market_segment
       Order: count(DISTINCT fact.order_key)
       ->  XN Sort  (cost=1000383363755.08..1000383747755.08 rows=153600000 width=42)
             Sort Key: cust.market_segment, count(DISTINCT fact.order_key)
             ->  XN Network  (cost=362094300.00..362478300.00 rows=153600000 width=42)
                   Distribute
                   ->  XN HashAggregate  (cost=362094300.00..362478300.00 rows=153600000 width=42)
                         ->  XN Hash Join DS_DIST_NONE  (cost=2872625.00..360370725.00 rows=229810000 width=42)
                               Hash Cond: ("outer".customer_key = "inner".customer_key)
                               ->  XN Seq Scan on dim_customer cust  (cost=0.00..1536000.00 rows=153600000 width=42)
                               ->  XN Hash  (cost=2298100.00..2298100.00 rows=229810000 width=16)
                                     ->  XN Seq Scan on fact_line_item_order fact  (cost=0.00..2298100.00 rows=229810000 width=16)
```

As, we can see the query gives out a very efficient plan with join as 'DS_DIST_NONE'

Report 3: Helps us with identifying growth areas by identifying poorly performing supplier:
The below query is trying to identify suppliers whose parts gets delayed most often.

https://github.com/jiteshsoni/supply_chain_management/blob/master/sql-script/Report3.sql

```
XN HashAggregate  (cost=12959058.42..13035858.42 rows=10240000 width=38)
 ->  XN Hash Join DS_DIST_ALL_NONE  (cost=128000.00..12384533.41 rows=76603334 width=38)
       Hash Cond: ("outer".supplier_key = "inner".supplier_key)
       ->  XN Seq Scan on fact_line_item_order fact  (cost=0.00..2872625.00 rows=76603334 width=24)
             Filter: (commit_date < receipt_date)
       ->  XN Hash  (cost=102400.00..102400.00 rows=10240000 width=30)
             ->  XN Seq Scan on dim_supplier supp  (cost=0.00..102400.00 rows=10240000 width=30)
```

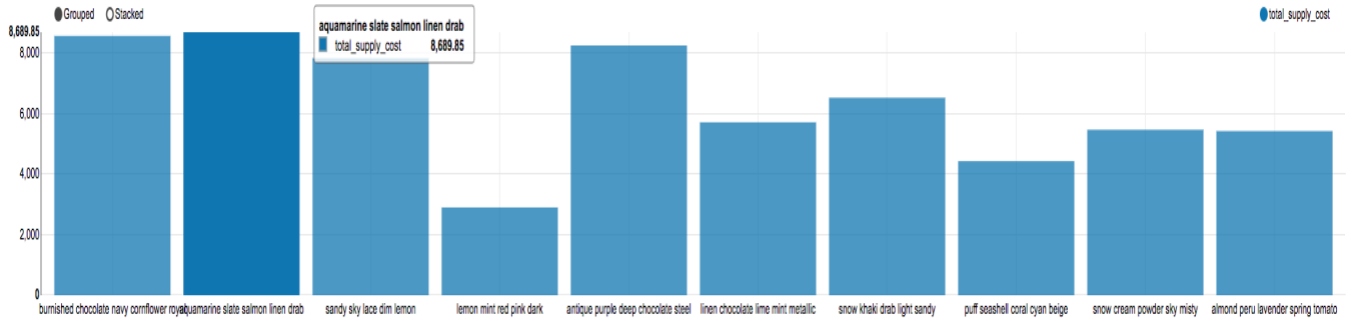
Report 4: Individual Supplier performance: Top 10 parts supplied with the total supply cost, number of parts shipped, average shipping time, average delivery time, average delay time

This schema design works even well for Supplier query pattern if the suppliers were allowed to access their performance data. The same schema would work without requiring any change except they would have to be given limited access.

https://github.com/jiteshsoni/supply_chain_management/blob/master/sql-script/Report4.sql

```
XN Limit  (cost=1000025960627.17..1000025960627.19 rows=10 width=73)
 ->  XN Merge  (cost=1000025960627.17..1000025960627.24 rows=30 width=73)
     Merge Key: count(fact.quantity)
     ->  XN Network  (cost=1000025960627.17..1000025960627.24 rows=30 width=73)
         Send to leader
         ->  XN Sort  (cost=1000025960627.17..1000025960627.24 rows=30 width=73)
             Sort Key: count(fact.quantity)
             ->  XN HashAggregate  (cost=25960625.83..25960626.43 rows=30 width=73)
                 ->  XN Hash Join DS_BCAST_INNER  (cost=2872625.08..25960625.38 rows=30 width=73)
                     Hash Cond: ("outer".part_key = "inner".part_key)
                     ->  XN Seq Scan on dim_part part  (cost=0.00..2048000.00 rows=204800000 width=44)
                     ->  XN Hash  (cost=2872625.00..2872625.00 rows=33 width=45)
                         ->  XN Seq Scan on fact_line_item_order fact  (cost=0.00..2872625.00 rows=33 width=45)
                             Filter: (supplier_key = 7045232)
```

DS_BCAST_INNER is ideal in this scenario, as we are only broadcasting only 30 rows.



Report 5: Individual Supplier performance: Show top 3 parts which are delayed most often for every order priority, along with total parts delayed and average days delayed

https://github.com/jiteshsoni/supply_chain_management/blob/master/sql-script/Report5.sql

```
XN Merge  (cost=2000013175150.81..2000013175150.82 rows=4 width=147)
 Merge Key: fact_rank.order_priority, fact_rank.product_rank
 ->  XN Network  (cost=2000013175150.81..2000013175150.82 rows=4 width=147)
     Send to leader
     ->  XN Sort  (cost=2000013175150.81..2000013175150.82 rows=4 width=147)
         Sort Key: fact_rank.order_priority, fact_rank.product_rank
         ->  XN Hash Join DS_BCAST_INNER  (cost=1000003447150.72..1000013175150.77 rows=4 width=147)
             Hash Cond: ("outer".part_key = "inner".part_key)
             ->  XN Seq Scan on dim_part part  (cost=0.00..2048000.00 rows=204800000 width=44)
             ->  XN Hash  (cost=1000003447150.71..1000003447150.71 rows=4 width=119)
```

```
-> XN Subquery Scan fact_rank  (cost=1000003447150.38..1000003447150.71 rows=4 width=119)
    Filter: (product_rank <= 3)
    -> XN Window  (cost=1000003447150.38..1000003447150.58 rows=11 width=38)
        Partition: order_priority
        Order: sum(quantity)
        -> XN Sort  (cost=1000003447150.38..1000003447150.41 rows=11 width=38)
            Sort Key: order_priority, sum(quantity)
            -> XN Network  (cost=3447150.11..3447150.19 rows=11 width=38)
                Distribute
                -> XN HashAggregate  (cost=3447150.11..3447150.19 rows=11 width=38)
                    -> XN Seq Scan on fact_line_item_order  (cost=0.00..3447150.00 rows=11 width=38)
                        Filter: ((supplier_key = 7045232) AND (commit_date < receipt_date))
```