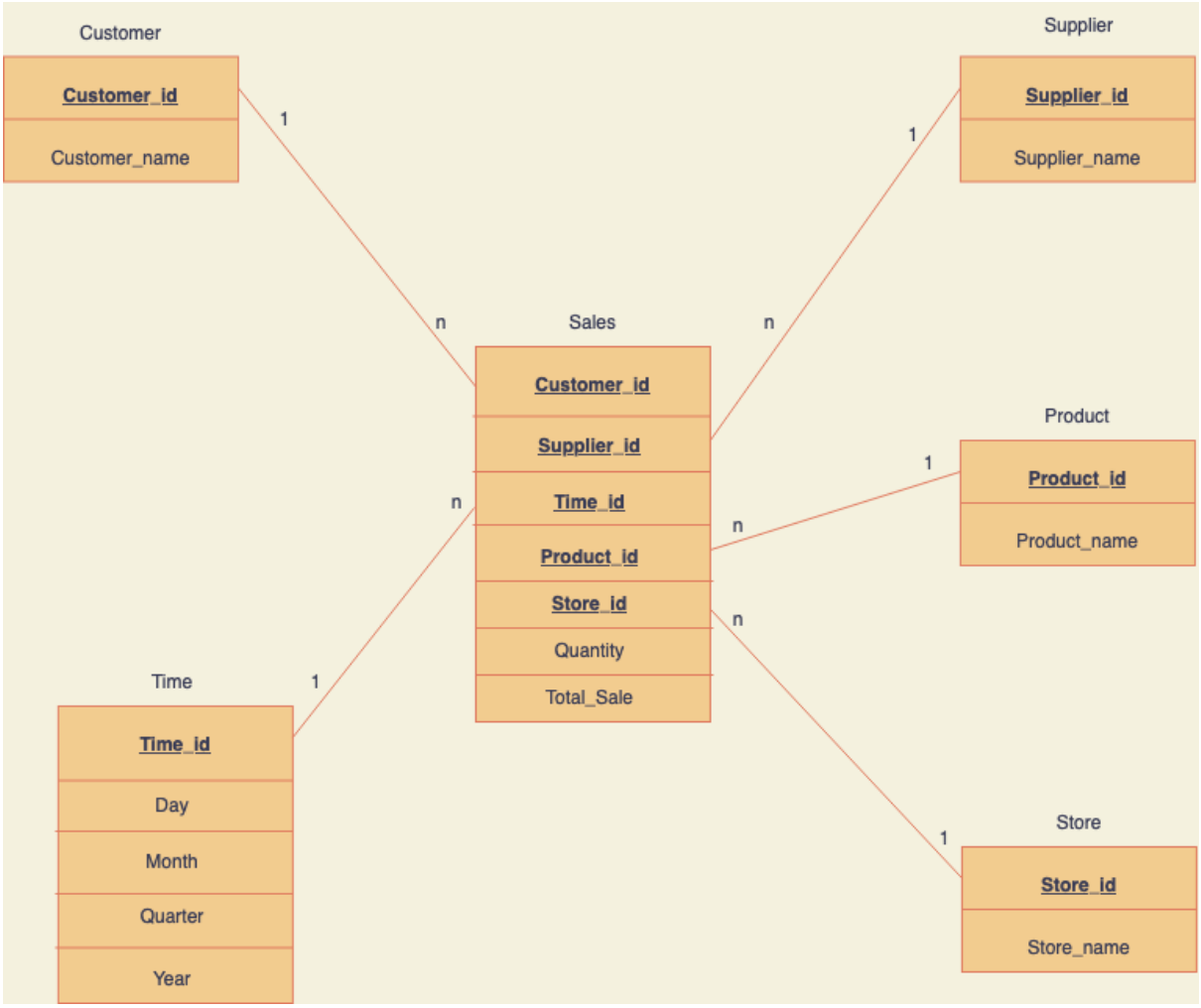
CS4033 DATA WAREHOUSING PROJECT

Building and Analyzing Data Warehouse Prototype for METRO Shopping Store

Project Overview

The aim of the project was to design, implement, and analyze a Data Warehouse (DW) prototype for METRO shopping store in Pakistan so that we make analysis of shopping behavior, optimize selling techniques etc. We had to build the warehouse with the data source which we were given. For this we implemented a real time ETL (Extraction, Transformation, and Loading) because the warehouse schema is different from database. We implemented the Mesh Join algorithm for integrating the transactional data with master data before loading into warehouse. After building the warehouse I analyzed the DW by applying OLAP queries.

Schema for Data Warehouse



Mesh join Algorithm:

Start

While (reading_data or Queue_not_empty)

```
stream_buffer  =  load partition from Transactional table
Queue = insert(50) , hash_table = insert (key,tuple)
disk_buffer = load next partition from Master table

for i in  disk_buffer
    if (hash_table contains)
        DW_tuple = join(i,hash_table(key))
        load_into_DW(DW_tuple)

if (Queue.size == partition.size)
    Queue.poll()
    hash_table.remove()
```

end

OLAP QUERIES:

-- 1ST QUERY

```
SELECT SUPPLIER_ID, QUARTER, MONTH, ROUND(SUM(TOTAL_SALE),2) AS TOTAL_SALES
FROM SALES
INNER JOIN TIME
USING (TIME_ID)
GROUP BY SUPPLIER_ID, QUARTER, MONTH
ORDER BY SUPPLIER_ID;
```

SUPPLIER_ID	QUARTER	MONTH	TOTAL_SALES
SP-1	1	1	2987.05
SP-1	1	2	2732.58
SP-1	1	3	3794.3
SP-1	2	4	3096.08
SP-1	2	5	3466.14
SP-1	2	6	3306.2
SP-1	3	7	3355.13
SP-1	3	8	3038.01
SP-1	3	9	3216.66
SP-1	4	10	3484.45
SP-1	4	11	4232.29
SP-1	4	12	3484.05
SP-10	1	1	2848.35
SP-10	1	2	2357.57
SP-10	1	3	2604.02
SP-10	2	4	2716.34
SP-10	2	5	2235.55
SP-10	2	6	2951.6
SP-10	3	7	2872.16

-- 2ND QUERY

```
SELECT STORE_ID, PRODUCT_ID, ROUND(SUM(TOTAL_SALE),2) AS TOTAL_SALES
FROM SALES
GROUP BY STORE_ID, PRODUCT_ID
ORDER BY STORE_ID;
```

STORE_ID	PRODUCT_ID	TOTAL_SALES
S-1	P-1001	540.9
S-1	P-1002	164.4
S-1	P-1003	448.76
S-1	P-1004	250.2
S-1	P-1005	1318.68
S-1	P-1006	378.9
S-1	P-1007	817.32
S-1	P-1008	439.56
S-1	P-1009	932.34
S-1	P-1010	489.84
S-1	P-1011	105.57
S-1	P-1012	422.69
S-1	P-1013	368.88
S-1	P-1014	102.03
S-1	P-1015	125.12
S-1	P-1016	389.84
S-1	P-1017	89.95
S-1	P-1018	109.48

-- 3RD QUERY

```
SELECT PRODUCT_ID, SUM(QUANTITY) AS QUANTITY
FROM SALES
INNER JOIN TIME
USING (TIME_ID)
WHERE dayname(TIME_ID) like 'Saturday' or dayname(TIME_ID) like 'Sunday'
GROUP BY PRODUCT_ID
ORDER BY QUANTITY DESC
LIMIT 5;
```

PRODUCT_ID	QUANTITY
P-1086	228
P-1091	226
P-1015	224
P-1034	221
P-1011	216

-- 4TH QUERY

```
SELECT PRODUCT_ID,
ROUND (SUM (CASE WHEN QUARTER =1 THEN TOTAL_SALE END),2) AS 1st_QUARTER,
ROUND (SUM (CASE WHEN QUARTER =2 THEN TOTAL_SALE END),2) AS 2nd_QUARTER,
ROUND (SUM (CASE WHEN QUARTER =3 THEN TOTAL_SALE END),2) AS 3rd_QUARTER,
ROUND (SUM (CASE WHEN QUARTER =4 THEN TOTAL_SALE END),2) AS 4th_QUARTER,
ROUND (SUM (CASE WHEN QUARTER <=4 THEN TOTAL_SALE END),2) AS TOTAL_SALES
FROM SALES
INNER JOIN TIME
USING (TIME_ID)
WHERE YEAR =2016
GROUP BY PRODUCT_ID
ORDER BY PRODUCT_ID;
```

PRODUCT_ID	1st_QUARTER	2nd_QUARTER	3rd_QUARTER	4th_QUARTER	TOTAL_SALES
P-1000	612.75	840.75	997.5	869.25	3320.25
P-1001	2740.56	2866.77	2452.08	3425.7	11485.11
P-1002	1106.96	328.8	586.36	630.2	2652.32
P-1003	1726	2554.48	1570.66	2847.9	8699.04
P-1004	3327.66	3277.62	4003.2	3427.74	14036.22
P-1005	4102.56	3980.46	5543.34	2661.78	16288.14
P-1006	1524.02	1288.26	1481.92	1220.9	5515.1
P-1007	2451.96	3230.36	2082.22	3308.2	11072.74
P-1008	2091.24	2024.64	2131.2	2504.16	8751.24
P-1009	2615.1	2592.36	2546.88	4661.7	12416.04
P-1010	1708.16	2110.08	1896.56	2097.52	7812.32
P-1011	835.38	817.02	500.31	472.77	2625.48
P-1012	1169.77	1621.95	1651.44	1612.12	6055.28
P-1013	2397.72	1367.93	1537	1859.77	7162.42
P-1014	248.81	270.29	186.16	216.59	921.85
P-1015	1177.6	1096.64	920	1354.24	4548.48
P-1016	1479.62	1045.48	1532.78	992.32	5050.2

-- 5TH QUERY

```
SELECT PRODUCT_ID, ROUND (SUM (CASE WHEN QUARTER <=2 THEN TOTAL_SALE END),2)
AS FIRST_HALF_SALE,
ROUND (SUM (CASE WHEN QUARTER >2 THEN TOTAL_SALE END),2) AS
SECOND_HALF_SALE,
ROUND (SUM (CASE WHEN QUARTER <=4 THEN TOTAL_SALE END),2) AS YEARLY_SALE
FROM SALES
INNER JOIN TIME
USING (TIME_ID)
WHERE YEAR =2016
GROUP BY PRODUCT_ID
ORDER BY PRODUCT_ID;
```

PRODUCT_ID	FIRST_HALF_SALE	SECOND_HALF_SALE	YEARLY_SALE
P-1000	1453.5	1866.75	3320.25
P-1001	5607.33	5877.78	11485.11
P-1002	1435.76	1216.56	2652.32
P-1003	4280.48	4418.56	8699.04
P-1004	6605.28	7430.94	14036.22
P-1005	8083.02	8205.12	16288.14
P-1006	2812.28	2702.82	5515.1
P-1007	5682.32	5390.42	11072.74
P-1008	4115.88	4635.36	8751.24
P-1009	5207.46	7208.58	12416.04
P-1010	3818.24	3994.08	7812.32
P-1011	1652.4	973.08	2625.48
P-1012	2791.72	3263.56	6055.28
P-1013	3765.65	3396.77	7162.42
P-1014	519.1	402.75	921.85
P-1015	2274.24	2274.24	4548.48
P-1016	2525.1	2525.1	5050.2

-- 6TH QUERY

```
SELECT DISTINCT(PRODUCT_ID), PRODUCT_NAME, PRICE, SUPPLIER_ID
FROM TRANSACTIONS
inner join MASTERDATA
using (PRODUCT_ID)
where PRODUCT_NAME= 'Tomatoes';
```

	PROD...	PRODUCT_NAME	PRICE	SUPPLIER_ID
▶	P-1014	Tomatoes	1.79	SP-4
	P-1088	Tomatoes	19.40	SP-9

Anomaly:

The product “Tomatoes” has two different suppliers with different price so we have two different product ids for the same product which makes “Tomatoes” difficult to analyze.

-- 7TH QUERY

```
CREATE OR REPLACE VIEW STOREANALYSIS_MV AS
SELECT STORE_ID, PRODUCT_ID, ROUND(SUM(TOTAL_SALE),2) AS STORE_TOTAL
FROM SALES
GROUP BY STORE_ID, PRODUCT_ID
ORDER BY STORE_ID;
```

STORE_ID	PRODUCT_ID	STORE_TOTAL
S-1	P-1001	540.9
S-1	P-1002	164.4
S-1	P-1003	448.76
S-1	P-1004	250.2
S-1	P-1005	1318.68
S-1	P-1006	378.9
S-1	P-1007	817.32
S-1	P-1008	439.56
S-1	P-1009	932.34
S-1	P-1010	489.84
S-1	P-1011	105.57
S-1	P-1012	422.69
S-1	P-1013	368.88
S-1	P-1014	102.03

Shortcomings in Mesh Join:

1. Stream buffer can create a bottleneck as its size is fixed. It can overflow if we consider a stream of data coming.

2. When multiple partition of queue has same join id in hash table it becomes a challenge to differentiate between there tuples in the hash table when we are removing the tuple.

3. The no of partitions of master data should be equal to the partitions of the queue which has to be predefined which creates an overhead.

4. The time for one partition of transactional data to join with all of the master data is large.