

Assessment 2 –Data Warehousing Project: Building and Analysing a DW for NatureFresh Stores in NZ

Udit Sharma
Department of Computer Science
Auckland University of Technology
Auckland, New Zealand
Student Id-19066381
kby5902@autuni.ac.nz

Contents

1	Project Overview	1
2	Data Source	2
3	Schema for Data Warehouse	2
4	Index Nested Loop Join	5
5	Datawarehouse Analysis/OLAP Queries	8
5.1	Top 5 products in Dec 2019	8
5.2	Store with most sales in the whole year	9
5.3	Top 3 products for Dec 2019 and 2 months before it	9
5.4	Creating the Materialized View "STOREANALYSIS"	10
5.5	Information that can retrived using CUBE and ROLLUP con- cepts in STOREANALYSIS Materialized View	11
5.5.1	Cube	11
5.5.2	Using ROLLUP Function	12
5.5.3	Suggestion of Choice for Management	12
6	Learning Summary	13

1 Project Overview

The aim of this project is to design, implement and analyse a Data warehouse for a supermarket chain in New Zealand called NatureFresh. NatureFresh is spread

all across the country and is considered to be one of the biggest fresh food market chains. Due to sheer amount of NatureFresh customers it is important that they can analyse the behavior of their customers and this in turn can help they maximize their profits. To achieve this goal there is a need for daily data integration between the Data sources and Data warehouse. So, for this task an Index Nested Loop Join has been designed in this Project using PL/SQL procedures which is further expanded in Section 3 of this paper.

The Data warehouse in this project has been designed as a Star schema which contains one fact table and 4 dimensional tables as is discussed in Section 2 of this paper. To analyse the data warehouse multiple Online Analytical Processing (OLAP) queries have been explained in Section 4 of this project. Section 5 of this paper discusses the different learning objectives of this project. This project has been built using SQL developer for writing SQL queries/creating the PL/SQL procedure and and Oracle 10g server for creating and storing the Datawarehouse.

2 Data Source

The data source is the raw data is being generated by NatureFresh. For this Project the Data Source is two tables:- TRANSACTIONS Table and MASTERDATA Table which contain the raw data for the Year 2019. The Transactions Table shown in Figure 1a contains 10,000 records and Masterdata Table shown in Figure 1b contains 100 records.

KBY5902.TRANSACTIONS		
P	*	TRANSACTIONS_ID NUMBER (6)
	*	PRODUCT_ID VARCHAR2 (6 BYTE)
	*	CUSTOMER_ID VARCHAR2 (4 BYTE)
	*	CUSTOMER_NAME VARCHAR2 (30 BYTE)
	*	STORE_ID VARCHAR2 (4 BYTE)
	*	STORE_NAME VARCHAR2 (20 BYTE)
	*	T_DATE DATE
	*	QUANTITY NUMBER (3)
		TRANSACTIONS_PK (TRANSACTIONS_ID)
		TRANSACTIONS_PK (TRANSACTIONS_ID)

(a) TRANSACTIONS Table

KBY5902.MASTERDATA		
P	*	PRODUCT_ID VARCHAR2 (6 BYTE)
	*	PRODUCT_NAME VARCHAR2 (30 BYTE)
	*	SUPPLIER_ID VARCHAR2 (6 BYTE)
	*	SUPPLIER_NAME VARCHAR2 (30 BYTE)
	*	PRICE NUMBER (5,2)
		MASTERDATA_PK (PRODUCT_ID)
		MASTERDATA_PK (PRODUCT_ID)

(b) MASTERDATA Table

Figure 1: Data Source Tables

3 Schema for Data Warehouse

The Data warehouse in this paper has been built using a Star Schema. The Star Schema has a central fact table which is connected to multiple dimension tables. The major advantage of star schema is that it reduces code complexities as it does not require complex join queries to bridge data from multiple tables [1]. The fact table in the schema contains multiple data items and foreign keys that

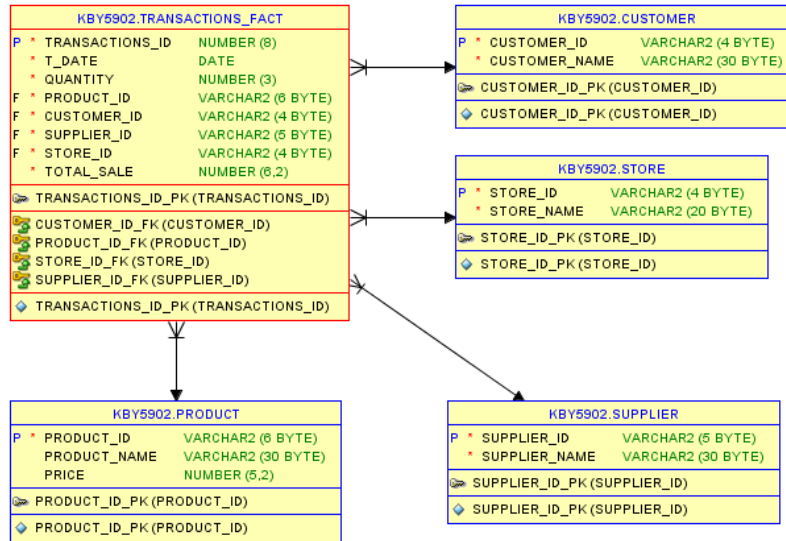


Figure 2: Star Schema

help to define its relationship with the dimension table. The Star Schema for this project is shown in Figure 2. This Star schema contains 5 tables:-1 fact table and 4 dimension tables. Table TRANSACTIONS_FACT is the fact table of this schema and tables PRODUCT,SUPPLIER,CUSTOMER and STORE are the dimension tables. The description of these tables can be found below:-

TRANSACTIONS_FACT TABLE

1. **Data Information:** Contains the information regarding transactions and sale
2. **Number of Columns:** 8
3. **Number of Records:** 10000
4. **Primary Keys:** TRANSACTIONS_ID
5. **Foreign Keys:**
 - PRODUCT_ID reference to PRODUCT TABLE
 - CUSTOMER_ID reference to CUSTOMER TABLE
 - STORE_ID reference to STORE TABLE
 - SUPPLIER_ID reference to SUPPLIER TABLE

```

DROP TABLE TRANSACTIONS_FACT;
DROP TABLE PRODUCT;
DROP TABLE SUPPLIER;
DROP TABLE CUSTOMER;
DROP TABLE STORE;

CREATE TABLE PRODUCT(
PRODUCT_ID VARCHAR2(6),
PRODUCT_NAME VARCHAR2(30),
PRICE NUMBER(5,2) DEFAULT 0.0,
CONSTRAINT PRODUCT_ID_PK PRIMARY KEY (PRODUCT_ID)
);

CREATE TABLE SUPPLIER(
SUPPLIER_ID VARCHAR2(5),
SUPPLIER_NAME VARCHAR2(30) NOT NULL,
CONSTRAINT SUPPLIER_ID_PK PRIMARY KEY(SUPPLIER_ID));

CREATE TABLE CUSTOMER(
CUSTOMER_ID VARCHAR2(4),
CUSTOMER_NAME VARCHAR2(30) NOT NULL,
CONSTRAINT CUSTOMER_ID_PK PRIMARY KEY(CUSTOMER_ID));

CREATE TABLE STORE(
STORE_ID VARCHAR2(4),
STORE_NAME VARCHAR2(20) NOT NULL,
CONSTRAINT STORE_ID_PK PRIMARY KEY(STORE_ID));

```

Figure 3: SQL code for Dimensions Tables

```

CREATE TABLE TRANSACTIONS_FACT(
TRANSACTIONS_ID NUMBER(8,0),
T_DATE DATE NOT NULL,
QUANTITY NUMBER(3,0) NOT NULL,
PRODUCT_ID VARCHAR2(6) NOT NULL,
CUSTOMER_ID VARCHAR2(4) NOT NULL,
SUPPLIER_ID VARCHAR2(5) NOT NULL,
STORE_ID VARCHAR2(4) NOT NULL,
TOTAL_SALE NUMBER(6,2) NOT NULL,
CONSTRAINT TRANSACTIONS_ID_PK PRIMARY KEY(TRANSACTIONS_ID),
CONSTRAINT PRODUCT_ID_FK FOREIGN KEY (PRODUCT_ID) REFERENCES PRODUCT(PRODUCT_ID),
CONSTRAINT CUSTOMER_ID_FK FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMER(CUSTOMER_ID),
CONSTRAINT SUPPLIER_ID_FK FOREIGN KEY (SUPPLIER_ID) REFERENCES SUPPLIER(SUPPLIER_ID),
CONSTRAINT STORE_ID_FK FOREIGN KEY (STORE_ID) REFERENCES STORE(STORE_ID));

```

Figure 4: SQL code for Fact Table

PRODUCT TABLE

1. **Data Information:** Contains information about the products of NatureFresh.
2. **Number of Columns:** 3
3. **Number of Records:** 100
4. **Primary Keys:** PRODUCT_ID

CUSTOMER TABLE

1. **Data Information:** Contains information of NatureFresh customers.
2. **Number of Columns:** 2
3. **Number of Records:** 50
4. **Primary Keys:** CUSTOMER_ID

SUPPLIER TABLE

1. **Data Information:** Contains information of different product suppliers of NatureFresh.
2. **Number of Columns:** 2
3. **Number of Records:** 20
4. **Primary Keys:** SUPPLIER_ID

STORE TABLE

1. **Data Information:** Contains information of NatureFresh store across the country;
2. **Number of Columns:** 2
3. **Number of Records:** 10
4. **Primary Keys:** STORE_ID

4 Index Nested Loop Join

To implement data integration between the Data source and the Data warehouse we need to use ETL (Extraction, Transformation, and Loading) tools. The ETL tools are used for extraction from multiple and heterogeneous data sources then transform the data into a suitable format such that it can be loaded into the data warehouse [2]. For this project an Index Nested Loop Join (INLJ) has been selected. The index nested loop join is an extension of nested loop join where the join attribute is based on an index join [3]. The INLJ algorithm has been built using a procedure written in PL/SQL using SQL developer. The Procedure INLJ shown in figure 5.

```

create or replace PROCEDURE INLJ AS
  t_id TRANSACTIONS.TRANSACTIONS_ID*type;
  p_id TRANSACTIONS.PRODUCT_ID*type;
  c_id TRANSACTIONS.CUSTOMER_ID*type;
  c_name TRANSACTIONS.CUSTOMER_NAME*type;
  s_id TRANSACTIONS.STORE_ID*type;
  s_name TRANSACTIONS.STORE_NAME*type;
  tt_date TRANSACTIONS.T_DATE*type;
  t_quant TRANSACTIONS.QUANTITY*type;
  type transact_type IS varray(50) of TRANSACTIONS%ROWTYPE;
  transact transact_type;
  md_p_id MASTERDATA.PRODUCT_ID*type;
  p_name MASTERDATA.PRODUCT_NAME*type;
  sup_id MASTERDATA.SUPPLIER_ID*type;
  sup_name MASTERDATA.SUPPLIER_NAME*type;
  md_price MASTERDATA.PRICE*type;
  check_id NUMBER;
  counter NUMBER;
  loopnum NUMBER;
  CURSOR transactions_fetch is
  SELECT * FROM transactions;

```

```

BEGIN
  OPEN transactions_fetch;
  LOOP
    loopnum:=loopnum+1;
    FETCH transactions_fetch BULK COLLECT INTO transact LIMIT 50;
    EXIT WHEN transactions_fetch%notfound;
    FOR counter in 1..50
      LOOP
        t_id:=transact(counter).transactions_id;
        p_id:=transact(counter).product_id;
        c_id:=transact(counter).customer_id;
        c_name:=transact(counter).customer_name;
        s_id:=transact(counter).store_id;
        s_name:=transact(counter).store_name;
        tt_date:=transact(counter).t_date;
        t_quant:=transact(counter).quantity;
        SELECT product_id,product_name,supplier_id,supplier_name,price
        INTO md_p_id,p_name,sup_id,sup_name,md_price
        FROM MASTERDATA
        WHERE product_id=p_id;

```

```

--ADDING DATA IN DIMENSION TABLES
SELECT COUNT(*) INTO check_id FROM product WHERE product_id=p_id;
IF (check_id=0) THEN
INSERT INTO product VALUES(p_id,p_name,md_price);
END IF;
SELECT COUNT(*) INTO check_id FROM customer WHERE customer_id=c_id;
IF (check_id=0) THEN
INSERT INTO customer VALUES(c_id,c_name);
END IF;
SELECT COUNT(*) INTO check_id FROM supplier WHERE supplier_id=sup_id;
IF (check_id=0) THEN
INSERT INTO supplier VALUES(sup_id,sup_name);
END IF;
SELECT count(*) INTO check_id FROM store WHERE store_id=s_id;
IF (check_id=0) THEN
INSERT INTO store VALUES(s_id,s_name);
END IF;
SELECT count(*) INTO check_id FROM transactions_fact WHERE transactions_id=t_id;
IF (check_id=0) THEN
INSERT INTO transactions_fact VALUES(t_id,tt_date,t_quant,p_id,c_id,sup_id,s_id,t_quant*md_price);
END IF;
END LOOP;
END LOOP;
COMMIT;
Exception
WHEN OTHERS THEN
dbms_output.put_line(counter||loopnum);
dbms_output.put_line(dbms_utility.format_call_stack());
END INLJ;

```

Figure 5: INLJ Code

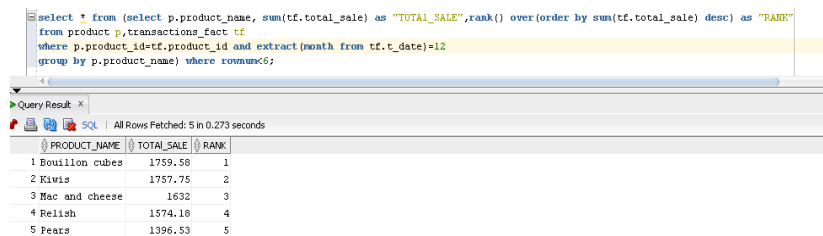
The algorithm for INLJ is described as follows:-

1. Declare the Variables such that they correspond the variable data types to right table columns.
2. Declare the Cursor to read data from Transactions table (Data source).
3. Open the Cursor
4. Start the Loop
5. Fetch 50 records from the Cursor into the variables
6. Read the records tuple by tuple and read the data from Masterdata table (Data source) using PRODUCT_ID as the joining condition.
7. Check if the record exist in dimension tables.
 - If the data exist in Dimension Table add the record in only the Transactions_FACT table.
 - If the data doesn't exist in any Dimension Table add the data in that Dimension table only. Then add the record in Facts Table.
8. Repeat from Step 5 till all the Records have been fetched. Meanings the Exit condition should be Cursor Not Found.
9. Commit

5 Datawarehouse Analysis/OLAP Queries

Online Analytical Processing queries are used to analyse multi-dimensional data [7]. These OLAP queries have been used to analyse different aspects of NatureFresh Business and are discussed in the follow:-

5.1 Top 5 products in Dec 2019



```
select * from (select p.product_name, sum(tf.total_sale) as "TOTAL_SALE",rank() over(order by sum(tf.total_sale) desc) as "RANK"
from product p,transactions_fact tf
where p.product_id=tf.product_id and extract(month from tf.t_date)=12
group by p.product_name) where rownum<6;
```

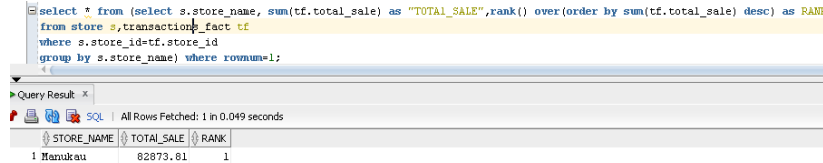
	PRODUCT_NAME	TOTAL_SALE	RANK
1	Bouillon cubes	1759.58	1
2	Kiwis	1757.75	2
3	Mac and cheese	1632	3
4	Relish	1574.18	4
5	Pears	1396.53	5

Figure 6: SQL CODE AND OUTPUT TASK 1

The query shown in figure 6 is used to find the Top 5 products that were sold in December 2019. The query displays the Product names, their total sale

value and Ranking. The query extracts data from transactions_fact and product table using the product_id as the joining condition and checking condition for month=2 where month is extracted using extract function.

5.2 Store with most sales in the whole year



```
select * from (select s.store_name, sum(tf.total_sale) as "TOTAL_SALE",rank() over(order by sum(tf.total_sale) desc) as RANK
from store s,transactions_fact tf
where s.store_id=tf.store_id
group by s.store_name) where rownum=1;
```

STORE_NAME	TOTAL_SALE	RANK
1 Manukau	82873.81	1

Figure 7: SQL CODE AND OUTPUT TASK 2

The query shown in figure 7 is to find the store with the highest total sale in the whole year. The query extracts data from transactions_fact and store table using the store_id as the joining condition.

5.3 Top 3 products for Dec 2019 and 2 months before it

```
select * from (select p.product_name, sum(tf.total_sale) as "TOTAL_SALE",
'Rank:- '||rank() over(order by sum(tf.total_sale) desc)||' Month '||extract(month from tf.t_date) as "RANK"
from product p,transactions_fact tf
where p.product_id=tf.product_id and extract(month from tf.t_date)=12
group by p.product_name,extract(month from tf.t_date)
) where rownum<4
union
select * from (select p.product_name, sum(tf.total_sale) as "TOTAL_SALE",
'Rank:- '||rank() over(order by sum(tf.total_sale) desc)||' Month '||extract(month from tf.t_date) as "RANK"
from product p,transactions_fact tf
where p.product_id=tf.product_id and extract(month from tf.t_date)=11
group by p.product_name,extract(month from tf.t_date)
) where rownum<4
union
select * from (select p.product_name, sum(tf.total_sale) as "TOTAL_SALE",
'Rank:- '||rank() over(order by sum(tf.total_sale) desc)||' Month '||extract(month from tf.t_date) as "RANK"
from product p,transactions_fact tf
where p.product_id=tf.product_id and extract(month from tf.t_date)=10
group by p.product_name,extract(month from tf.t_date)
) where rownum<4;
```

Figure 8: SQL CODE TASK 3

The query shown in figure 8 is to find the top 3 products for December 2019 and 2 months before that. The query uses Union operator to join the results from 3 individual queries that get the top 3 products for each month. The query fetches data from product and transactions_fact table using product_id as a joining condition and month checks for each month. The figure 9 shows the output of the query.

PRODUCT_NAME	TOTAL_SALE	RANK
1 Bouillon cubes	1759.58	Rank:- 1 Month 12
2 Broccoli	1514.52	Rank:- 3 Month 11
3 Kiwis	1757.75	Rank:- 2 Month 12
4 Mac and cheese	1632	Rank:- 3 Month 12
5 Onions	2296.74	Rank:- 1 Month 11
6 Oregano	1476.8	Rank:- 3 Month 10
7 Paprika	1692.6	Rank:- 1 Month 10
8 Pizza / Pizza Rolls	1505	Rank:- 2 Month 10
9 Relish	1751.91	Rank:- 2 Month 11

Figure 9: OUTPUT TASK 3

5.4 Creating the Materialized View "STOREANALYSIS"

The code shown in 10 is to create a Materialized View called "STOREANALYSIS" that contains the sale of each product for each store. The Materialized view is created using tables transactions_fact, product and store and uses store_id and product_id to create the join statement. The final materialized view is shown in figure 11

```
CREATE MATERIALIZED VIEW STOREANALYSIS
as select s.store_id,p.product_id,sum(tf.total_sale)
from product p,store s,transactions_fact tf
where tf.store_id=s.store_id and tf.product_id=p.product_id
group by s.store_id,p.product_id
order by s.store_id,p.product_id;
```

Figure 10: SQL CODE TASK 4

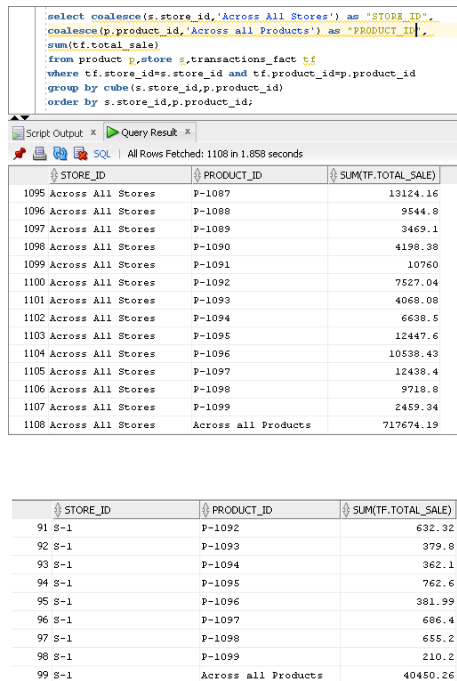
STORE_ID	PRODUCT_ID	SUM(TF.TOTAL_SALE)
1 S-1	P-1001	546.9
2 S-1	P-1002	164.4
3 S-1	P-1003	446.76
4 S-1	P-1004	250.2
5 S-1	P-1005	1318.68
6 S-1	P-1006	378.9
7 S-1	P-1007	617.32
8 S-1	P-1008	439.56
9 S-1	P-1009	932.34
10 S-1	P-1010	469.84
11 S-1	P-1011	105.57
12 S-1	P-1012	422.69
13 S-1	P-1013	368.88
14 S-1	P-1014	102.03
15 S-1	P-1015	125.12

Figure 11: MATERIALIZED VIEW TASK 4

5.5 Information that can retrived using CUBE and ROLLUP concepts in STOREANALYSIS Materialized View

5.5.1 Cube

Cube function is an extension to select and group-by clause. CUBE function is used when we need to enable a Query to fetch all possible combinations for a group of dimensions [9]. CUBE can be used when querying in situations that require cross tabular report or requires or when multiple dimensions are needed with their all possible combinations. The sample cube query and its output for this project is shown in figure. 12 For this Project Cube function can be used



```

select coalesce(s.store_id,'Across All Stores') as "STORE_ID",...
coalesce(p.product_id,'Across all Products') as "PRODUCT_ID",...
sum(tf.total_sale)
from product p,store s,transactions_fact tf
where tf.store_id=s.store_id and tf.product_id=p.product_id
group by cube(s.store_id,p.product_id)
order by s.store_id,p.product_id;

```

STORE_ID	PRODUCT_ID	SUM(TF.TOTAL_SALE)
1095 Across All Stores	P-1087	13124.16
1096 Across All Stores	P-1088	9544.8
1097 Across All Stores	P-1089	3469.1
1098 Across All Stores	P-1090	4198.38
1099 Across All Stores	P-1091	10760
1100 Across All Stores	P-1092	7527.04
1101 Across All Stores	P-1093	4068.08
1102 Across All Stores	P-1094	6638.5
1103 Across All Stores	P-1095	12447.6
1104 Across All Stores	P-1096	10538.43
1105 Across All Stores	P-1097	12438.4
1106 Across All Stores	P-1098	9718.8
1107 Across All Stores	P-1099	2459.34
1108 Across All Stores	Across all Products	717674.19

STORE_ID	PRODUCT_ID	SUM(TF.TOTAL_SALE)
91 S-1	P-1092	632.32
92 S-1	P-1093	379.8
93 S-1	P-1094	362.1
94 S-1	P-1095	762.6
95 S-1	P-1096	381.99
96 S-1	P-1097	686.4
97 S-1	P-1098	655.2
98 S-1	P-1099	210.2
99 S-1	Across all Products	40450.26

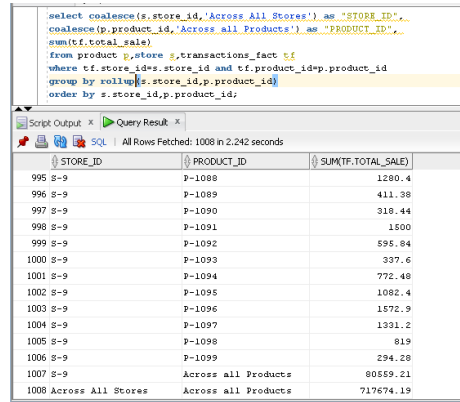
Figure 12: Using Cube Function

to 4 different possible set of values. The Cube function show in figure 12 takes product_id and store_id. The set of outputs generated by it are :-

1. To calculate sale of each product across each store
2. To calculate sale of all products across each store
3. To calculate sale of each product across all store
4. To calculate grand total sale of NatureFresh. Meaning sum of sale of all products across all stores.

5.5.2 Using ROLLUP Function

RollUP function is another extension of select and group by clause. Roll up functions like cube can be used to create sub totals and grand totals [9]. The difference in rollup and cube is that rollup produces only a subset of results what cube can produce. For example if dimensions (c1,c2) are given roll up produces sets:- [(c1,c2),(c1),()] where as cube produces [(c1,c2),(c1),(c2),()].



```

select coalesce(s.store_id,'Across All Stores') as "STORE_ID",
       coalesce(p.product_id,'Across all Products') as "PRODUCT_ID",
       sum(tf.total_sale)
from product p,store s,transactions_fact tf
where tf.store_id=s.store_id and tf.product_id=p.product_id
group by rollup(s.store_id,p.product_id)
order by s.store_id,p.product_id;

```

STORE_ID	PRODUCT_ID	SUM(TF.TOTAL_SALE)
995 S-9	P-1088	1280.4
996 S-9	P-1089	411.38
997 S-9	P-1090	310.44
998 S-9	P-1091	1500
999 S-9	P-1092	595.84
1000 S-9	P-1093	397.6
1001 S-9	P-1094	772.48
1002 S-9	P-1095	1082.4
1003 S-9	P-1096	1572.9
1004 S-9	P-1097	1391.2
1005 S-9	P-1098	819
1006 S-9	P-1099	294.28
1007 S-9	Across all Products	80559.21
1008 Across All Stores	Across all Products	717674.19

Figure 13: Using the Rollup Function

For this project rollup can be used to calculate sub and grand total of Sale. The possible values that can be calculate using rollup is as follows:-

1. To calculate sale of each product across each store
2. To calculate total sale of each store.
3. To calculate total sale by NatureFresh.

5.5.3 Suggestion of Choice for Management

The rollup and cube provide very similar functions. Both can be used to calculate different combination of Sale for our project. Cube produces a more detailed report than rollup function. The use of both totally depends on the needs and priorities of the management but it is my recommendation to use the Cube function due to the fact that more elaborate materialized view can be created using Cube. The cube can help management analyse better by providing the following analysis:-

- Which stores are performing well?
- What products are in more demand and generating more sales?
- What products are popular across each store or region?
- What is the total profit generated across all the stores?

When a materialized view is created using Cube it would be taking up more memory in the data warehouse than a materialized view generated by rollup would. But as data warehouses are huge and are able to store excess amounts of data, Cube can provide better and more comprehensive analysis to management.

6 Learning Summary

This project uses a wide variety of concepts. The concepts learnt in during this project are:-

1. **Designing Star Schema for a Data warehouse demonstrated in section 3.** In this project I have designed a Star Schema that has one 1 fact table and 4 dimension tables. I have learnt how to define a relationship in Star schema databases. Star schema contains a central fact table that is connected to all the dimension tables and it helps to make the querying process easier by providing a central table to perform all join queries with.
2. **ETL process and using Index Nested Join Loop to perform ETL demonstrated in Section 4.** I have used INLJ algorithm to perform extraction, transformation and loading the data in a data warehouse. I learnt how to read data from a raw data source how to transform it so that it can be loaded in different tables inside the data warehouse. ETL is an important pre-processing step as there is a huge amount of raw data and that needs to be transformed and loaded in the data warehouse.
3. **Creation of PL/SQL procedures and functions demonstrated in Section 4.** I have learnt how we can create and write PL/SQL procedure and functions in SQL developer. I have also learnt how to display data inside the procedure, debug them and how to run them. I also learnt how to declare variables using type and rowtype. PL/SQL procedures/functions are very useful and use various features that help to make the database more accessible and create a fixed named entity for repetitive queries.
4. **Exception handling in PL/SQL demonstrated in Section 4.** To debug and catch different errors that can occur in PL/SQL procedure I have also learnt the concept of exception handling in PL/SQL. Exception handling is a very important part of logging that can help to understand where and when the error has occurred and help to fix it.
5. **Use of Cursor, using fetch(including Bulk Collect and Limit uses) to fetch data values in a cursor demonstrated in Section 4.** I have learnt how to declare cursors, how to open a cursor and how to Fetch data in them. I also learnt how to use Bulk Collect and Limit to limit the number of rows fetched inside a cursor. Cursor are important as they can fetch table results and help in processing and analysing them.
6. **Use of If conditions,Loops and varray in PL/SQL demonstrated in Section 4.** I have learnt how to declare array/varray in PL/SQL,

how we can use conditional or IF statements and how looping can be used in PL/SQL using loop command,for loop command and exit when statements. Conditional statements are important as they can be used in providing conditions to dynamic data and loops are useful as they help to repeat a process over large amounts of data.

7. **Using Insert and Select demonstrated in Section 4.** I have learnt how to select data from various tables and data source. As part of ETL process I have also learnt how we can use Insert statement to insert data in a table.
8. **Performing analysis using OLAP queries demonstrated in Section 5.** In section 5 I have used various queries to perform different analysis for NatureFresh. I have learnt how to use various conditions and how they can be used in industry to actually analyse the data present inside a table. OLAP queries are built to provide in depth analysis and this can prove very important while making decisions.
9. **Use of group by and order by Clause in Select query demonstrated in Section 5.** Throughout section 5 I have used group by and order by clause to sort data in different orders and to group it to get the desired results and perform analysis like how much sale occurred for different stores and products. This helped me understand how you can extract different conclusions from data.
10. **Limiting the Number of results using top, rownum and limit. Using extract function. Demonstrated in Section 5.** I have used Limit,Top N and Rownum limit the queries so that data is more presentable and easy to read and this helps to only get the required data and remove the not wanted data.
11. **How to create materialized views, what are there advantages and how to use them demonstrated in Section 5.4.** I have created a materialized view named as "STOREANALYSIS". This has helped me to understand how we can create materialized views, how they can be used to retrieve data faster and improve query performance. Materialized views save the results of a query for faster access.
12. **How CUBE and ROLLUP concepts work and how they can applied to analysis of data demonstrated in Section 5.5.** I have demonstrated how rollup and cube produce different results and learnt how they can be useful in different kinds of analysis. Cube helps to create all different combinations of given dimensions that can help to get all possible sub and grand total's whereas rollup can be used for the same but focuses on combinations considering the first dimension as shown in 5.5.2.

References

- [1] Krippendorf, M., Song, I. Y. (1997, September). The translation of star schema into entity-relationship diagrams. In Database and Expert Systems Applications. 8th International Conference, DEXA'97. Proceedings (pp. 390-395). IEEE.
- [2] Hanlin, Q., Xianzhen, J., Xianrong, Z. (2012, October). Research on extract, transform and load (ETL) in land and resources star schema data warehouse. In 2012 Fifth International Symposium on Computational Intelligence and Design (Vol. 1, pp. 120-123). IEEE.
- [3] Gunadhi, H., Segev, A. (1991, April). Query processing algorithms for temporal intersection joins. In [1991] Proceedings. Seventh International Conference on Data Engineering (pp. 336-344). IEEE.
- [4] Nguyen, A., Edahiro, M., Kato, S. (2018, July). GPU-Accelerated VoltDB: A Case for Indexed Nested Loop Join. In 2018 International Conference on High Performance Computing Simulation (HPCS) (pp. 204-212). IEEE.
- [5] Hanlin, Q., Xianzhen, J., Xianrong, Z. (2012, October). Research on extract, transform and load (ETL) in land and resources star schema data warehouse. In 2012 Fifth International Symposium on Computational Intelligence and Design (Vol. 1, pp. 120-123). IEEE.
- [6] Oracle Docs "Processing OLAP Queries" Link:- <https://docs.oracle.com/cd/E57185-01/HIRUG/ch06s05.html>
- [7] IBM document-" Writing Efficient OLAP Queries" link:- <https://www.ibm.com/developerworks/data/library/cognos/page128.html>
- [8] Oracle docs:- "Querying OLAP Cubes" link:- https://www.oracle.com/webfolder/technetwork/tutorials/obe/db/12c/r1/olap/olap_cube/querycubes.htm
- [9] Oracle Docs:- "Analyzing Data with ROLLUP, CUBE, AND TOP-N QUERIES" link:- https://docs.oracle.com/cd/F49540-01/DOC/server.815/a68003/rollup_c.htm