Data Warehouse For Nature Fresh - Project Report

Siddharth Tripathi

July 9, 2020

Contents

1 Project Overview

Data Warehouses are basically very large databases. The size of data warehouses might vary from few hundred GBs to several TBs. DWs are used in data analysis and reporting. It not only provides business information to managers and users(Devlin and Cote, 1996).

This report, contains a overview of the assessment task which is performed for design, implement and analysis of data warehouse for Nature Fresh - a fresh food store chain in NZ. ETL tools perform data extraction from different sources, cleaning the data and its customization and insertion into data warehouse (Vassiliadis et al., 2002).

The join operation is implemented using Index Nested Loop Join (INLJ) algorithm, which is discussed further in this report. This algorithm is implemented for Extraction, Transformation and Loading of data in DW.

2 Data Source and Master Data

The assessment task sheet provided to us contains a script file named - Transaction_and_MasterData_Generator.sql. The execution of script results in the two things -

1. Creation of tables -Transaction and Masterdata.
2. Insertion of Data into both the tables created.

Information about the tables such as attributes, constraints, data types etc is as follows:

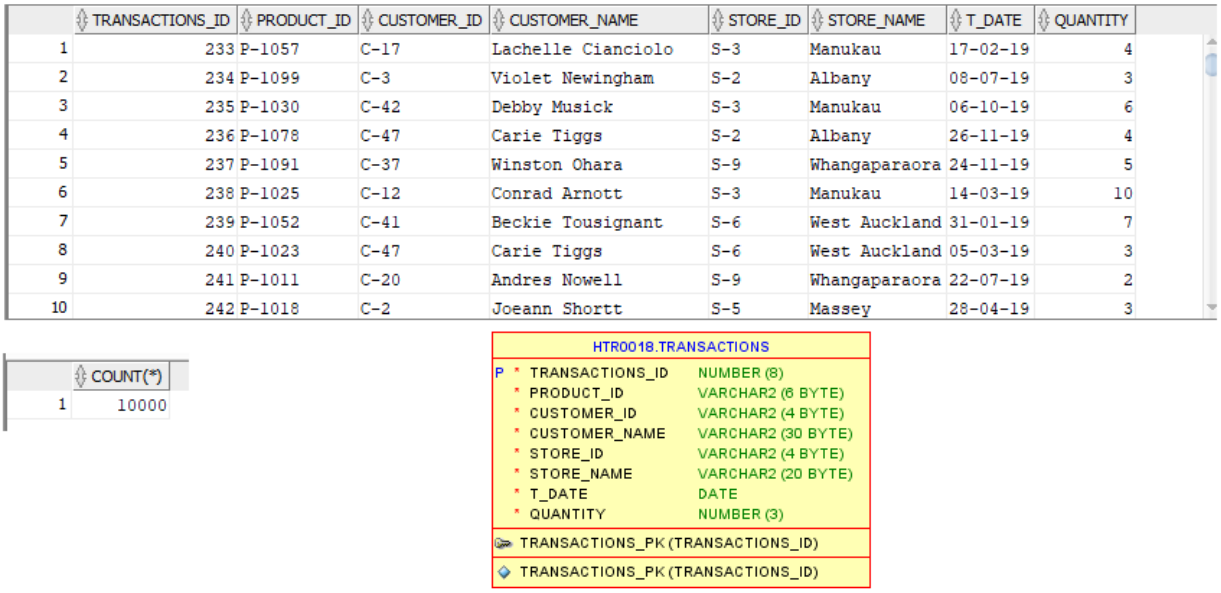
2.1 Transactions Table

1. Number of Records: 10000
2. Number of Columns: 8
3. Primary Key: Transactions_ID

Fig 1 shows an overview of the table, its attributes(columns) in tabular output format.The count output shows total records. Model overview(highlighted in yellow) shows columns as well as their data types and fields whose value cannot be null(marked by red dot).

2.2 Master Data

1. Number of Records: 100
2. Number of Columns: 5
3. Primary Key: Product_ID

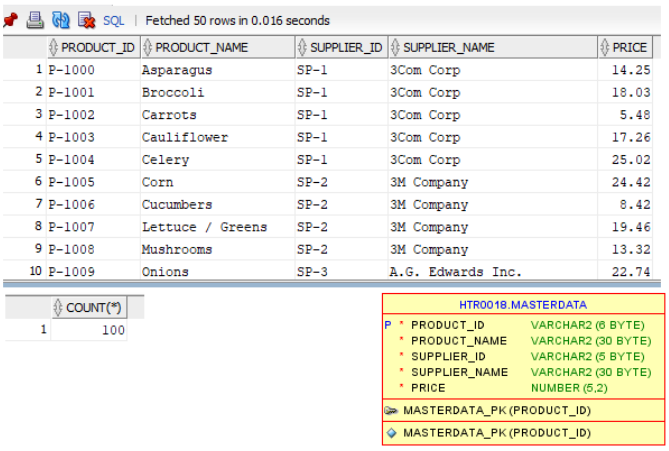


	TRANSACTIONS_ID	PRODUCT_ID	CUSTOMER_ID	CUSTOMER_NAME	STORE_ID	STORE_NAME	T_DATE	QUANTITY
1	233	P-1057	C-17	Lachelle Cianciolo	S-3	Manukau	17-02-19	4
2	234	P-1099	C-3	Violet Newingham	S-2	Albany	08-07-19	3
3	235	P-1030	C-42	Debby Musick	S-3	Manukau	06-10-19	6
4	236	P-1078	C-47	Carie Tiggs	S-2	Albany	26-11-19	4
5	237	P-1091	C-37	Winston Ohara	S-9	Whangaparaora	24-11-19	5
6	238	P-1025	C-12	Conrad Arnott	S-3	Manukau	14-03-19	10
7	239	P-1052	C-41	Beckie Tousignant	S-6	West Auckland	31-01-19	7
8	240	P-1023	C-47	Carie Tiggs	S-6	West Auckland	05-03-19	3
9	241	P-1011	C-20	Andres Nowell	S-9	Whangaparaora	22-07-19	2
10	242	P-1018	C-2	Joeann Shortt	S-5	Massey	28-04-19	3

	COUNT(*)
1	10000

Figure 1: Transactions Table Overview

Fig 2 shows an overview of the table, its attributes(columns) in tabular output format.The count output shows total records. Model overview(highlighted in yellow) shows columns as well as their data types and fields whose value cannot be null(marked by red dot).



SQL | Fetched 50 rows in 0.016 seconds

	PRODUCT_ID	PRODUCT_NAME	SUPPLIER_ID	SUPPLIER_NAME	PRICE
1	P-1000	Asparagus	SP-1	3Com Corp	14.25
2	P-1001	Broccoli	SP-1	3Com Corp	18.03
3	P-1002	Carrots	SP-1	3Com Corp	5.48
4	P-1003	Cauliflower	SP-1	3Com Corp	17.26
5	P-1004	Celery	SP-1	3Com Corp	25.02
6	P-1005	Corn	SP-2	3M Company	24.42
7	P-1006	Cucumbers	SP-2	3M Company	8.42
8	P-1007	Lettuce / Greens	SP-2	3M Company	19.46
9	P-1008	Mushrooms	SP-2	3M Company	13.32
10	P-1009	Onions	SP-3	A.G. Edwards Inc.	22.74

	COUNT(*)
1	100

Figure 2: Master Data table overview

3 Star-Schema

The Star-Schema is one of the most widely used approach to develop data warehouses and it is a special case of Snowflake schema. In this schema, a large table is present at center known as fact table and multiples smaller tables known as dimension tables around it connected in such a pattern that the entity relationship diagram of star schema resembles with star, hence called star schema (Moody and Kortink, 2000).

Why Star-Schema? In Star-Schema, the number of tables are reduced in database which also reduce the number of relationships between them. Lesser relationships means lesser joins in queries, which reduces the complexity (Levene and Loizou, 2003). Also, using this schema, aggregated data from various sources can be stored which represents different features of business operations.

Therefore, in our assessment, the data modelling technique used is Star-Schema. For this, we have created a sql script file named - createDW.sql. The execution of script results in creation of dimension tables and fact table as in star-schema which are as follows:

3.1 Dimension Tables

3.1.1 Product Table

We use create statement for creating product table. Below script (Fig 3) creates table product.

1. Primary Key :product_id
2. No. Of Columns: 3

```
--SQL statement for creating Dimension table Product
--PK-->product_id
create table product(
product_id varchar2(6),
product_name varchar2(30) not null,
price number(5,2) default 0.0,
constraint product_id_pk Primary key (product_id));
```

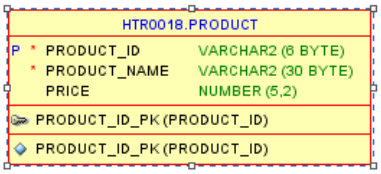


Figure 3: Script - Create Product

3.1.2 Supplier Table

We use create statement for creating supplier table. Below script (Fig 4) creates table Supplier.

1. Primary Key :supplier_id
2. No. Of Columns: 2

```sql
--SQL statement for creating Dimension table Supplier
--PK-->supplier_id
create table supplier(
supplier_id varchar2(5),
supplier_name varchar2(30) not null,
constraint supplier_id_pk primary key (supplier_id)
);
```

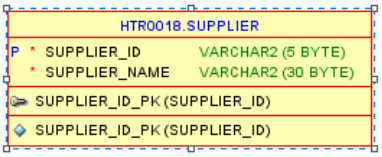


Figure 4: Script - Create Supplier

3.1.3 Customer Table

We use create statement for creating customer table. Below script (Fig 5) creates table Customer.

1. Primary Key :customer_id
2. No. Of Columns: 2

```sql
--SQL statement for creating Dimension table Customer
--PK-->customer_id
create table customer(
customer_id varchar2(4),
customer_name varchar2(30) not null,
constraint customer_id_pk primary key (customer_id)
);
```

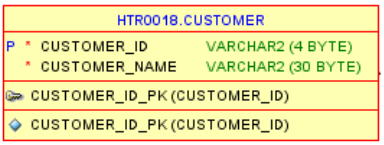


Figure 5: Script - Create Customer

3.1.4 Store Table

We use create statement for creating store table. Below script (Fig 6) creates table Store.

1. Primary Key :store_id
2. No. Of Columns: 2

```sql
--SQL statement for creating Dimension table Store
--PK-->store_id
create table store(
store_id varchar2(4),
store_name varchar2(20) not null,
constraint store_id_pk primary key (store_id)
);
```

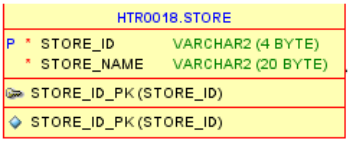


Figure 6: Script - Create Store

3.1.5 Date Table

We use create statement for creating Date_Table table. Below script (Fig 7) creates table Date_table.

1. Primary Key :date_id
2. No. Of Columns: 7

```
--SQL statement for creating Dimension table date_table
--PK-->date_id
Create table date_table(
date_id NUMBER,
day NUMBER,
MONTH varchar(10),
month_num number,
t_date date,
year number(4),
quart number,
constraint date_id_pk primary key (date_id));
```

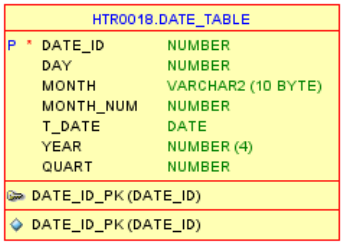


Figure 7: Script - Create Date

3.2 Fact Table - Transaction_Fact

The execution of this script creates a central fact table named - TRANSACTION_FACT (fig 8)

1. Primary Key :transaction_id
2. No. Of Columns: 8
3. Foreign Keys :product_id, customer_id, store_id,supplier_id, date_id

```
create table transaction_fact(
transaction_id number(8,0),
date_id NUMBER,
--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 transaction_id_pk primary key (transaction_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),
constraint date_id_fk FOREIGN key (date_id)references date_table(date_id)
);
```

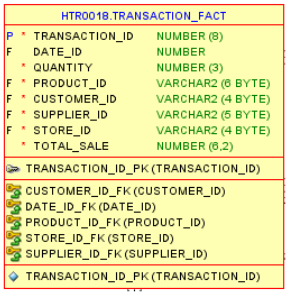


Figure 8: Script - Create Transaction_Fact

The Foreign Key defined in fact table - Transaction_fact table creates the star-schema(Fig 9) as discussed in starting of this section.

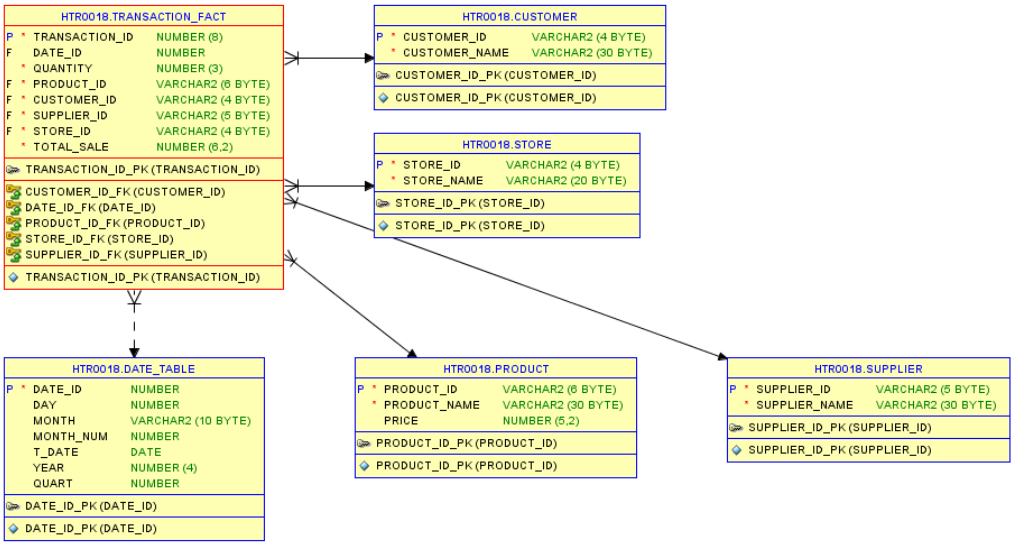


Figure 9: Star Schema

4 Index Nested Loop Join (INLJ)

Extract-Transform-Load the data from database to data warehouse. Index Nested Loop Join Algorithm is also one of the three fundamental algorithms available for performing content enrichment, that is to append additional attributes in data(Naeem et al., 2008). The INLJ perfroms JOIN operation between the Data Source and MASTERDATA tables and does extraction transformation and loading of data.

4.1 Implementation of INLJ in Assessment Task

The INLJ algorithm is implemented for data integration i.e JOIN operation between Data Sources and Master Data. Basically, in INLJ, chunks of tuples are read from Data Source and then they are joined with the relevant tuples in master data using index. For example in the scenario of Nature Fresh, PRODUCT_ID is index to join relevant tuples in master data and data source.

4.1.1 INLJ Algorithm

Below are steps explaining how INLJ is implemented for the present scenario using Procedure.

1. Create a procedure (INLJ here) and declare variable using %TYPE attribute so that they are of same data type as corresponding columns.

2. Create a cursor and fetch data from TRANSACTION table, 50 records at a time. For loading data into date_table dimension table, another procedure named create_date(Fig 11) is created and called inside cursor of procedure INLJ.

3. The PRODUCT_ID is used as index and cursor is read tuple by tuple and corresponding tuple from Master Data is fetched using index.

4. Then this data is loaded into Dimension tables (PRODUCT, SUPPLIER, STORE,CUSTOMER) with a check condition(id_check==0) - if information already exists or not.

5. The steps 2-4 are repeated with help of loop, till all the records are uploaded or say Cursor is empty.

The procedure INLJ has been created using SQL to perform above steps and load data(Fig 10). The procedure CREATE_DATE is used to load data in date_table(Fig 11)

```
create or replace PROCEDURE INLJ AS

trans_id TRANSACTIONS.TRANSACTIONS_ID%type;
prod_id TRANSACTIONS.PRODUCT_ID%type;
cust_id TRANSACTIONS.CUSTOMER_ID%type;
cust_name TRANSACTIONS.CUSTOMER_NAME%type;
st_id TRANSACTIONS.STORE_ID%type;
st_name TRANSACTIONS.STORE_NAME%type;
trans_date TRANSACTIONS.T_DATE%type;
trans_quant TRANSACTIONS.QUANTITY%type;
md_prod_id MASTERDATA.PRODUCT_ID%type;
prod_name MASTERDATA.PRODUCT_NAME%type;
supp_id MASTERDATA.SUPPLIER_ID%type;
supp_name MASTERDATA.SUPPLIER_NAME%type;
md_price MASTERDATA.PRICE%type;
t_date_id date_table.date_id%type;


--Variables for cursor and loops for fetching 50 records at time
type trans_type IS varray(50) of TRANSACTIONS%ROWTYPE;
transact trans_type;
id_check NUMBER;
countr NUMBER;
loop_numb NUMBER;

--Step 2
CURSOR trans_fetch is
SELECT * FROM transactions;
BEGIN
OPEN trans_fetch;

CREATE_DATE_TABLE(
    YEAR_NUM => 2019
  );
LOOP
loop_numb:=loop_numb+1;
FETCH trans_fetch BULK COLLECT INTO transact LIMIT 50;
EXIT WHEN trans_fetch%notfound;
```

```
FOR countr in 1..50
--step 3
LOOP
trans_id:=transact(countr).transactions_id;
prod_id:=transact(countr).product_id;
cust_id:=transact(countr).customer_id;
cust_name:=transact(countr).customer_name;
st_id:=transact(countr).store_id;
st_name:=transact(countr).store_name;
trans_date:=transact(countr).t_date;
trans_quant:=transact(countr).quantity;
SELECT product_id,product_name,supplier_id,supplier_name,price
INTO md_prod_id,prod_name,supp_id,supp_name,md_price
FROM MASTERDATA
WHERE product_id=prod_id;

SELECT date_id INTO t_date_id FROM date_table WHERE t_date=trans_date;

SELECT COUNT(*) INTO id_check FROM product WHERE product_id=prod_id;
IF(id_check=0) THEN
INSERT INTO product VALUES(prod_id,prod_name,md_price);
END IF;
SELECT COUNT(*) INTO id_check FROM customer WHERE customer_id=cust_id;
IF(id_check=0) THEN
INSERT INTO customer VALUES(cust_id,cust_name);
END IF;
SELECT COUNT(*) INTO id_check FROM supplier WHERE supplier_id=supp_id;
IF(id_check=0) THEN
INSERT INTO supplier VALUES(supp_id,supp_name);
END IF;
SELECT count(*) INTO id_check FROM store WHERE store_id=st_id;
IF(id_check=0) THEN
INSERT INTO store VALUES(st_id,st_name);
END IF;
SELECT count(*) INTO id_check FROM transaction_fact WHERE transaction_id=trans_id;
IF(id_check=0) THEN
INSERT INTO transaction_fact
VALUES(trans_id,t_date_id,trans_quant,prod_id,cust_id,supp_id,st_id,trans_quant*md_price);
END IF;
END LOOP;

END LOOP;
```

Figure 10: INLJ -PROCEDURE

```
create or replace PROCEDURE CREATE_DATE(year_num NUMBER) AS


leap_num NUMBER;
type monthseq IS varray(12) of NUMBER;
month_seq monthseq;
type trans_type IS varray(50) of TRANSACTIONS%ROWTYPE;
start_date Date;
quart number;


BEGIN
start_date:=TO_DATE('01-JAN-'||TO_CHAR(year_num));
leap_num:=365;
IF(mod(year_num,4)=0 ) THEN
leap_num:=366;
ELSIF (mod(year_num,4)=0 and mod(year_num,100)=0) THEN
leap_num:=365;
ELSIF (mod(year_num,4)=0)THEN
leap_num:=366;
END IF;

--Loops for making Quarter column
FOR i in 1..leap_num
LOOP
IF(extract(month from start_date) IN (1,2,3)) THEN
quart:=1;
ELSIF(extract(month from start_date) IN (4,5,6)) THEN
quart:=2;
ELSIF(extract(month from start_date) IN (7,8,9)) THEN
quart:=3;
ELSIF(extract(month from start_date) IN (10,11,12)) THEN
quart:=4;
END IF;

--data insertion
insert into date_table values(DATE_ID_SEQ.nextval,
extract(day from start_date),to_char(start_date,'MONTH'),extract(month from start_date),start_date,year_num,quart);

start_date:=start_date+1;
END LOOP;

commit;
EXCEPTION
WHEN others then
```

Figure 11: Create Date Procedure

5 Online Analytical Processing - OLAP Queries

Online Analytical Processing is a process of analysing multi-dimensional data. OLAP is a subset of data mining, business intelligence and relational databases. OLAP provide fast solution to business problems, data warehousing problems and data mining tasks which help establish a competitive advantage (Berson and Smith, 1997). The OLAPs are an important element of decision support, and are opposite of OLTP or online transaction processing. The three basic operations in OLAP are - drill down, roll-up, slicing and dicing(Chaudhuri and Dayal, 1997). In this assessment, we analyse the data warehouse using OLAP queries as follows:

5.1 Task-1 Query

The query determines the top 5 products in terms of sales for the month of December in year 2019(Fig 12) The sub-query fetches the records in terms of sales for December 2019 while the outer query fetches top 5 records by ROWNUM (Fig 13).

```
-----Task 1--------

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,transaction_fact tf, date_table dt
   where p.product_id = tf.product_id and tf.date_id=dt.date_id and dt.month_num=12
   group by p.product_name
   )
where rownum<6;
```

Figure 12: Top 5 Products - Query

	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 13: Top 5 Products - Results

5.2 Task-2 Query

This query(Fig 14) determines the highest sale by a store in whole year. The sub-query fetches the records in terms of sales for whole year while the outer query fetches top record by row-num (Fig 15).

```
----Task 2------
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, transaction_fact tf
  where s.store_id = tf.store_id
  group by s.store_name
 )
where rownum<=1;
```

Figure 14: Highest Sale by store - Query

	STORE_NAME	Total Sale	Rank
1	Manukau	82873.81	1

Figure 15: Highest Sale by store - Results

5.3 Task-3 Query

This query determines the top 3 products for given month and 2 months before it. The query takes month as input from user in alphabetical form(Fig 16).

```
select * from
     (
      select p.product_name,sum(tf.total_sale)as Total_Sale,
      dt.month,rank() over
        (
         partition by dt.month
         order by sum(tf.total_sale) desc
         )as rank
      from product p,transaction_fact tf,date_table dt
      where p.product_id = tf.product_id and tf.date_id = dt.date_id and
      dt.month_num in
           (
            select * from
             (
               select unique(month_num)
               from date_table
               where month_num<=&month
               order by month_num desc
               )
             where rownum<=3
            )
      group by dt.month,p.product_name
      order by total_sale desc
      )
rslt where rslt.rank<=3 order by rslt.month, rslt.rank;
```

Figure 16: Top 3 Product sales for month - Query

Fetches top 3 product in terms of sale for that month and two months consecutive previous months.For Eg. If user enters 'December', then the results will be for December, November and October.The results are in descending order along with the rank column for each product in a particular month. Top 3 ranks are provided(Fig 17).

	PRODUCT_NAME	TOTAL_SALE	MONTH	RANK
1	Bouillon cubes	1759.58	DECEMBER	1
2	Kiwis	1757.75	DECEMBER	2
3	Mac and cheese	1632	DECEMBER	3
4	Onions	2296.74	NOVEMBER	1
5	Relish	1751.91	NOVEMBER	2
6	Broccoli	1514.52	NOVEMBER	3
7	Paprika	1692.6	OCTOBER	1
8	Pizza / Pizza Rolls	1505	OCTOBER	2
9	Oregano	1476.8	OCTOBER	3

Figure 17: Top 3 Product sales for month - Query

5.4 Task-4 Query

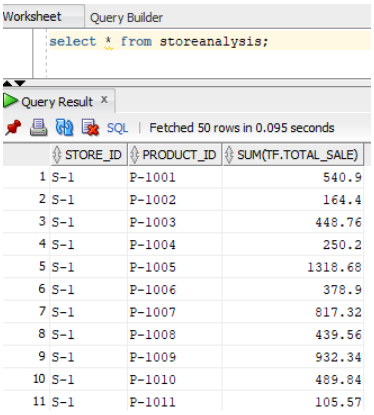
The query in task-4 create a materialised view called "STOREANALYSIS" that presents the product-wise sales analysis for each store (Fig 18).

```
create MATERIALIZED View

storeanalysis as select s.store_id, p.product_id,sum(tf.total_sale)
from product p, store s, transaction_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 18: Creating Materialized View Store Analysis - Query

If we run this view, we will get below output results which are ordered by store id and then by product id (Fig 19).



	STORE_ID	PRODUCT_ID	SUM(TF.TOTAL_SALE)
1	S-1	P-1001	540.9
2	S-1	P-1002	164.4
3	S-1	P-1003	448.76
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	817.32
8	S-1	P-1008	439.56
9	S-1	P-1009	932.34
10	S-1	P-1010	489.84
11	S-1	P-1011	105.57

Figure 19: Creating Materialized View Store Analysis - Query

5.5 Task-5 Query

This query implements the ROLLUP and CUBE. These are subclauses of GROUP BY clause. Here, ROLLUP and CUBE are implemented on the Materialized view to extract some useful information for management.

5.5.1 CUBE

The implementation of CUBE is done through below query (Fig 20).

```
---**************-------
-------Task 5----------
---**************-------


---CUBE  USING Materialized View---

select
NVL(store_id,'All Stores') as "Stores",
NVL(product_id,'All Products') as "Products",
sum("Total")
from storeanalysis
group by cube(store_id,product_id );
```

Figure 20: CUBE implementation on STOREANALYSIS - Query

The CUBE gives following useful information -

1. Total Sale of one product by one store

	Stores	Products	SUM("TOTAL")
1	S-1	P-1001	540.9
2	S-1	P-1002	164.4

Figure 21: CUBE INFO 1

2. Total Sale for all product in one store.

	Stores	Products	SUM("TOTAL")
907	S-9	All Products	80559.21

Figure 22: CUBE INFO 2

3. Total Sale for one product in all stores

	Stores	Products	SUM("TOTAL")
2	All Stores	P-1000	3320.25
3	All Stores	P-1001	11485.11

Figure 23: CUBE INFO 3

4. Total Sale for All products in All stores

	Stores	Products	SUM("TOTAL")
1	All Stores	All Products	717674.19

Figure 24: CUBE INFO 4

5.5.2 ROLLUP

The implementation of CUBE is done through below query (Fig 25)

```
--ROLLUP USING Materialized View-----

select
NVL(store_id,'All Stores') as "Stores",
NVL(product_id,'All Products') as "Products",
sum("Total")
from storeanalysis
group by ROLLUP(store_id,product_id );
```

Figure 25: ROLLUP implementation on STOREANALYSIS

The ROLLUP gives following useful information

1. Total Sale of one product by one store

	Stores	Products	SUM("TOTAL")
1	S-1	P-1001	540.9
2	S-1	P-1002	164.4

Figure 26: ROLLUP INFO 1

2. Total Sale for all product in one store.

	Stores	Products	SUM("TOTAL")
907	S-9	All Products	80559.21

Figure 27: ROLLUP INFO 2

3. Total Sale for All products in All stores

	Stores	Products	SUM("TOTAL")
1	All Stores	All Products	717674.19

Figure 28: ROLLUP INFO 3

6 Summary

This section provides overall summary of the assessment, tasks completed and different Big data and PL/SQL concepts learnt while completing those tasks.

- The aim of the assessment is to reflect transactions from customers in the Data Warehouse of Nature Fresh - a fresh food chain store. Data Warehouses are basically very large databases whose size may vary from Gigabytes to Terabytes.
  The ETL tools perform the extraction, transformation and loading of data in the data warehouse from different data sources and enrichment of data from master data.
  LEARNING - Data Warehouse, ETL tools

- Firstly, data source(Transactions table) and master data(MASTERDATA table) tables are created and data is inserted in it using sql script Transaction_and_MasterData_Generator.sql. The transactions table contain 10000 records and primary key is Transactions_ID while MASTERDATA table contains 100 records and has primary key-Product_ID

- We have implemented star-schema for our data warehouse. The schema contains a center fact table surrounded by dimension tables in such way that it forms star like structure. The star-schema reduces the number of tables and relationships which helps in reducing complexity.

- For start schema, we create TRANSACTION_FACT table (center) and dimension table - PRODUCT, CUSTOMER,STORE,SUPPLIER and DATE_TABLE. We use 'CREATE' sql statement for creating all these tables. 'CONSTRAINT' defines the rules for data in tables and is used here to define primary keys and foreign keys for the tables created.
  LEARNING - CREATE STATEMENT, CONSTRAINTS

- INDEX NESTED LOOP JOIN algorithm is implemented for data integration i.e JOIN operation between Data Sources and Master Data and loading data in the data warehouse. The index in INLJ helps in joining data from data source with relevant data from Masterdata while the loops help to implement chunk loading.

  A procedure is created named INLJ for implementing the above. When this procedure is run, the data is loaded into our star schema data warehouse from TRANSACTIONS and MASTERDATA tables.
  LEARNING - PROCEDURE, CURSORS, %TYPE, LOOPS, CONDITIONAL OPERATOR (IF)

- We use OLAP queries to analyse data which has been loaded by INLJ in our data warehouse and determine various information such as -

  - Top 5 products in terms of sale for a month
  - Highest sale produced by a store in year
  - Top 3 products by sale for 3 consecutive months
  - creating a materialized view
  - Using Rollup and Cube for extraction of information useful for management.

  LEARNING - ROWNUM, RANK, OVER, PARTITION, ROLLUP, CUBE, GROUP BY, ORDER BY

- The ROLLUP and CUBE are sub-clause of Group by clause. Group by is used for grouping up rows having same values. The ROLLUP and CUBE are similar except that CUBE creates all possible groups while ROLLUP does not create all possible groupings. Eg: If C1, C2, C3 are 3 columns, and CUBE(C1,C2,C3) is implemented then groups created will be - (C1,C2,C3), (C1,C2), (C2,C3),(C1,C3),(C1),(C2),(C3),()
  On the other hand, ROLLUP(C1,C2,C3) will create following groups - (C1,C2,C3),(C1,C2),(C1),(). That's why, in task-5 Query, the CUBE implementation gives extra information compared to ROLLUP.

References

Devlin, B., & Cote, L. D. (1996). *Data warehouse: From architecture to implementation*. Addison-Wesley Longman Publishing Co., Inc.

Vassiliadis, P., Simitsis, A., & Skiadopoulos, S. (2002). Conceptual modeling for etl processes, In *Proceedings of the 5th acm international workshop on data warehousing and olap*.

Moody, D. L., & Kortink, M. A. (2000). From enterprise models to dimensional models: A methodology for data warehouse and data mart design., In *Dmdw*.

Levene, M., & Loizou, G. (2003). Why is the snowflake schema a good data warehouse design? *Information Systems*, *28*(3), 225–240.

Naeem, M. A., Dobbie, G., & Webber, G. (2008). An event-based near real-time data integration architecture, In *2008 12th enterprise distributed object computing conference workshops*. IEEE.

Berson, A., & Smith, S. J. (1997). *Data warehousing, data mining, and olap*. McGraw-Hill, Inc.

Chaudhuri, S., & Dayal, U. (1997). An overview of data warehousing and olap technology. *ACM Sigmod record*, *26*(1), 65–74.