Assignment 6.1 – Executive Memo

Group 2

Theodore Fitch, Jesse Ford, Michael Goddard, and Somayah Eltoweissy

University of Maryland Global Campus

DATA 620

Dr. Majed Al-Ghandour

February 23, 2021

Executive Summary

In this document, our team will demonstrate the successful extract, transform, and load of data into a format that can be easily accessible for further analysis or storage in a data mart. Our team will provide the methods used to create a functional database that can aggregate the data fed into a system that ensures data integrity. Our team will also demonstrate the database's ability to answer business-specific inquiries such as summing orders in a specific area or tracking yearly growth. Finally, our team will discuss the importance of employing the appropriate data granularity and layout to ensure a database establishes sustainable attributes that provide useful information.

ETL Process

ETL is the process of Extracting, Transforming, and Loading data from a data source to a data warehouse. For our example, we will transform source data for order transactions from 2012 to 2014 into a cohesive data set that can be readily displayed and loaded into a flat-file for further review. Please see Appendix A – ETL Process Documentation for directions on generating the output flat-file using SQL code and MySQL Workbench.

The first step in the ETL process is extraction. In this case, we created a database using MySQL, which provides the necessary infrastructure to house and organize the data. We initially created a "week 6" database and ran the given script, which established the business units and product tables with associated data (Appendix A, Figure 1). Next, we used the "import" button on the MySQL GUI interface to upload the three flat files into the database, which provided the ordering data for 2012, 2013, and 2014. Our team followed an extraction process so that the data can be placed in a single location where it can be readily manipulated and analyzed.

The next step in the ETL process is transformation. The transformation process cleans the source data to produce data that conforms to a uniform format while ensuring data integrity and completeness (El-Sappagh, Hendawi, & El Bastawissy, 2011). Transformation of data may involve many intermediate steps that can link the data provided to unique identifiers or provide a consistent method to present, analyze, and export data for review. For example, our initial step to transforming the data was creating a "products" table to provide each product with a unique identifier (Appendix A, Figure 2). As a result, each product name will have a unique number that one can use to link a specific product to a specific order. Next, our team created a "working_orders" table to combine the order data for 2012, 2013, and 2014. Note that we included records with a quantity of zero or an order total of zero since knowing which products or time periods had no sales is meaningful information to retain. Our team had to perform some minor data scrubbing before continuing the process to ensure each variable had the same data type and description. For example, our team removed a space in the "order total" variable from the 2012 flat file, and two orders were totaled then multiplied by the unit price to produce the "order total" variable from the 2013 flat file. Our team subtracted the quantity discount from the order subtotal to create the "order total" variable from the 2014 flat file (Appendix A, Figure 3). As a result, the newly created working_orders table produced data for all the desired variables, such as product, region, quantity, order total, month, and year to the corresponding input file. Next, the working_orders table was altered and given a primary key of "orders_id" to ensure each order had a unique identifier. Product_id was assigned, which connects the working_orders table and the product table. (Appendix A, Figure 4). Finally, our team created the "prod_bu" table to normalize the data by creating a foreign key for both the business_unit table and products table (Appendix A, Figure 5). As a result, the prod_bu table links both product and

business_unit table, enabling the data to be joined together uniformly and loaded for further review.

The final stage of the ETL process is to load the data into a data mart. The load process may involve multiple steps that require creating an intermediate dimensional structure that the end-user can easily access (El-Sappagh, Hendawi, & El Bastawissy, 2011). As a result, one can join data tables or create new tables to aggregate the data into a condensed and readable format. For example, our team created an "all_data" table that joined together the orders, product, and prod_bu tables, which brought forward all variables from each table while excluding business units in decline (Appendix A, Figure 6). Our team also created an "aggregate_table" to quickly sum the quality ordered and order total for each bu_designation, bu_name, product_name, region, year, and month variable (Appendix A, Figure 7). As a result, our team can export each table into a flat-file for further analysis.

ERD Diagram

An Entity Relationship Diagram (ERD) is a visual representation of a relational database that illustrates the relationship among different entities within the system (Lucidchart, 2021). The week 6 database was reverse engineered to create an ERD to display the relationships between the business_unit, prod_bu, product, and orders entities present in the database (Diagram 1). As a result, one can draw the relationship between entities to establish the entities' interconnectedness using a defined set of symbols. For example, all entities in the ERD have a one-to-many relationship with another entity that may exist. A one-to-many relationship exists when the primary key from one entity table connects to many attributes of another entity table (Lucidchart, 2021). The dash mark from one entity table followed by a dotted line leading to a "crows' foot" connected to another entity table depicts this relationship. As shown in Diagram 1,

one BU_ID may be related to many Prod_BU_IDs in the pro_bu table. One product_id may be related to many order_ids in the orders table. One product_id may be related to many Prod_BUs in the prod_bu table. As a result, the ERD depicts all database entities and their relationship with each other.

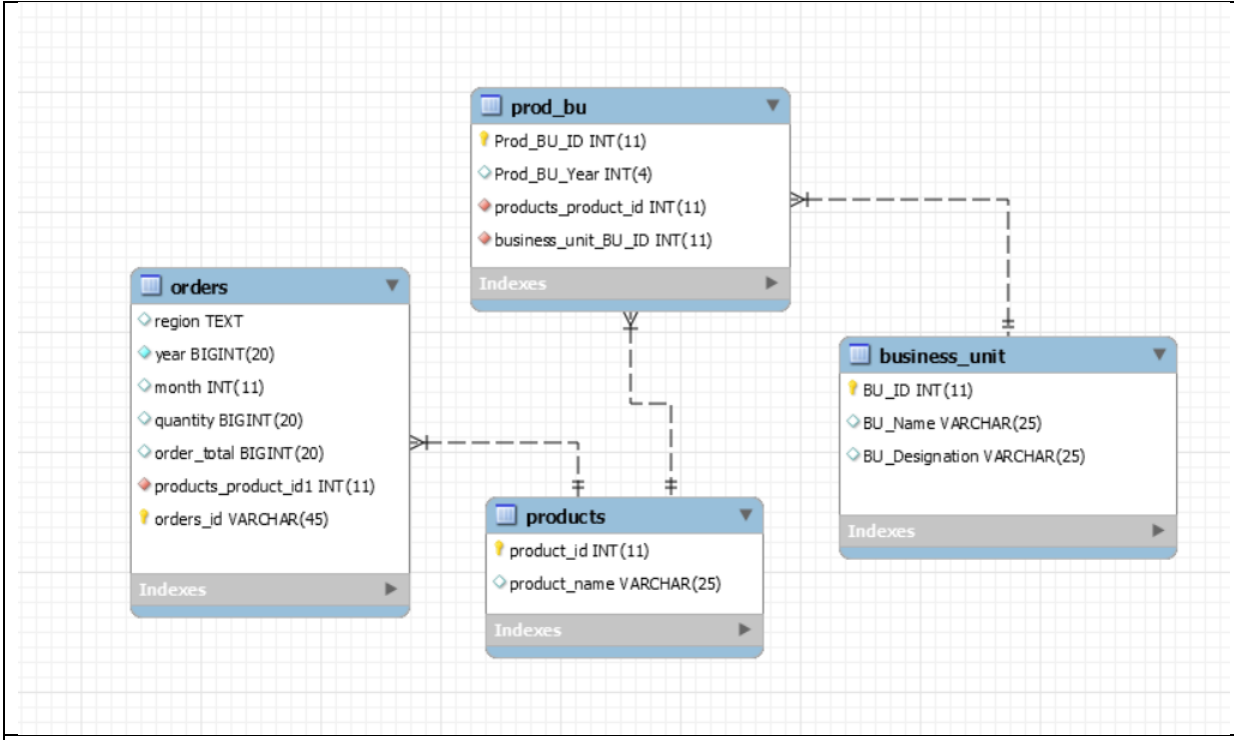


Diagram 1, Entity Relationship Diagram for Week 6 Database

Next, our team addressed missing metadata resulting from being provided a SQL script from the last data management team that contained no metadata file. An essential aspect of data mart design and support is the establishment and maintenance of metadata. In simple terms, metadata is data that describes data (Oracle, 2020). There are two distinct categories of metadata: business metadata and technical metadata (Oracle, 2020).

Business metadata is essential for the departments who use our data mart because it helps them understand what information in our data mart, how to access the information, and which

group is responsible for metadata documentation (Oracle, 2020). Our company could use technical metadata to describe data in the data warehouse, where the data originated, how one transforms the data for load into the data mart, and how often one should refresh the data (Oracle, 2020).

Our team reviewed the company's data warehouse and developed clear and comprehensive technical metadata documentation in Appendix B - Metadata Documentation. This documentation will assist other data engineers in future ETL processes and data mart maintenance tasks.

Levels of Granularity (Question 1)

Next, our team evaluated your question concerning the level of granularity for our data mart. To answer this question, we must understand which departments plan to utilize the data mart and which Key Performance Indicators (KPIs) each department is interested in tracking (Inmon & Linstedt, 2014). For example, the sales department may want to understand what impact volume discounts are having on sales. In 2014, our company started offering a 10% discount if a customer ordered more than 36 units of any of our products. The challenge when viewing the sum of quantity in the data mart is that in large orders (e.g., 72 units), one cannot tell if 72 units represent one order times 72 packets or two orders times 36 packets. This ambiguity makes it impossible to know if a volume discount was applied when only viewing data using the final layout. Our current data mart only reflects the sum of quantity value for each business unit designation, business unit name, region, and product for an entire month and year. With that said, our company cannot determine if a volume discount was applied to large orders. In other words, the coarseness of data does not enable the sales department to develop a KPI to measure the impact of *volume discounts* on the sum of order total for each month. The sales department may

have to request a change to the data mart, a separate data mart, or access to our data warehouse to measure the impact of volume discounts on sales.

As another example, consider a scenario where the marketing department has developed a KPI that measures the impact of advertising spend on sales revenue during certain times of the year, such as holidays or special sporting events like the Super Bowl or the World Series. The marketing department wants to determine how television and radio ads impact revenue during specific ***weeks*** or even ***days*** of the year. For example, the marketing department wants to run a 30-second commercial during the Super Bowl and then review the data mart later to determine how sales increased on the day the commercial aired. The marketing department wants to repeat this process for other campaigns throughout the year and then report to management how well their marketing strategy improved revenue. The marketing department cannot achieve this result using the data mart in its current form since there is no time dimension as granular as the day of the week.

Our team would like to mention that our current data mart ***does*** have the granularity necessary to support KPIs such as year over year, quarterly, or monthly revenue growth for each product line. A savvy data analyst could use the data mart to readily create visualizations depicting revenue growth or produce a report for management on short notice. Revenue growth is a crucial barometer of company performance, so it is paramount that our data mart supports this fundamental measurement. Notably, our data mart does not currently support KPIs such as profit margins. Referring to our metadata documentation in Appendix B – Metadata Documentation, there are currently no tables in our data warehouse that track this data, such as the costs of raw materials used to produce our products. If this data were available, we could incorporate it into our data mart to better estimate profit margins and add value to our company

stakeholders and investors. The main takeaway from this last point is that data warehouses and data marts are flexible and can be modified to meet the business's changing needs. KPIs will change over time depending on business trends, leadership styles, and stakeholder needs. (Inmon & Linstedt, 2014). Our team is prepared to adapt the data warehouse and data mart to serve the company.

In summary, whether the data mart has the granularity we need comes primarily down to which departments will use the data mart and what KPIs they use. The data mart has the granularity necessary to measure revenue growth. Still, it does not have the granularity to support more specialized KPIs such as the impact of advertising spend or volume discounts on revenue. Fortunately, one advantage of data marts is that they are flexible and adjusted to meet the business's changing needs.

Measure Yearly Growth (Question 2)

One crucial aspect of managing our business is the ability to measure sales performance over time. As noted in Appendix B – Metadata Documentation, it is apparent that our company stores data related to orders, order totals, and product business unit designations such as our "Mature" and "Growth" segments. The orders, order totals, and product business unit designation are the values that Ramon needs to calculate year-over-year growth for both quantity sold and order totals for individual segments. Ramon can execute queries against the flat file we produced as one of our project deliverables to answer sales performance questions. The output data file we created will easily support Ramon in this sort of analysis. As a proof-of-concept, we developed the following workflow to demonstrate a few of the calculations that one can perform.

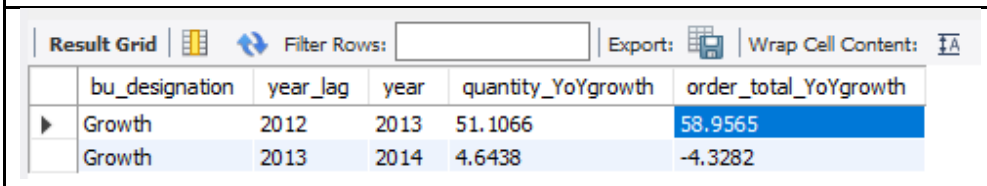
First, Ramon can import the data file into MySQL or any other program that supports the Structured Query Language (SQL) after he creates a new database using the SQL software. The imported table is named data_aggregate and contains all the fields present in the output data file.

Second, Ramon can create a table in the SQL database named year_aggregate that aggregates orders by business unit designation and year based on the data_aggregate table (Appendix C, Figure 12). The purpose of this step is to simplify the problem by selecting the distinct business unit designation, year, the sum of quantity sold, and the sum of order totals from the aggregated data.

Third, Ramon can utilize his expert analysis skills to execute a sophisticated SQL query that will select distinct rows from the aggregated year table using the business unit designation, prior year, and current year as column headers. Ramon can use a SQL join command to select quantity totals and order totals from both the current year and the prior year, along with a formula to calculate the percentage growth year over year (Appendix C, Figure 13). Our team used the Year-over-Year growth formula to calculate the percent growth.

Fourth, Ramon executes the SQL query to produce the output shown in Figure 14 below, reflecting year-over-year growth for both quantity and order total. The results below validate that the year-over-year growth for both quantity and order total can be determined using our team's output file. According to the data below, we can tell that from 2012 to 2013, the company experienced growth exceeding 10% for both quantity sold and order total. However, from 2013 to 2014, our company did not experience the level of growth we expected. This analysis provides a result where a data mart – with the appropriate variables and layout - can ensure the company meets or falls short of its revenue projections.

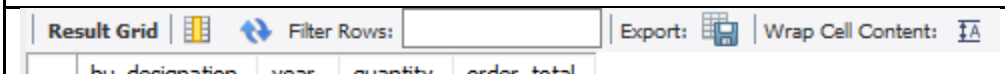
Diagram 2 – Query Results Indicating Year Over Year Growth for Quantity and Order Total



bu_designation	year_lag	year	quantity_YoYgrowth	order_total_YoYgrowth
Growth	2012	2013	51.1066	58.9565
Growth	2013	2014	4.6438	-4.3282

Fifth, Ramon executes a SQL query to determine if sales remained relatively flat year over year for our Mature segment (Appendix C, Figure 14). He selects all rows from the aggregated year table where the Business Unit Designation is Mature. The output shown in Diagram 3 below validates that the quantity and order total for each year's Mature segment can readily be displayed.

Diagram 3 - Query Results Indicating Quantity and Order Totals for Mature Segment



bu_designation	year	quantity	order_total
Mature	2012	1912	389920
Mature	2013	4140	1092598
Mature	2014	4577	1340486

Data Mart Layout (Question 3)

It is clear each of these layouts has its pros and cons, but we strongly recommend staying with the existing layout. While this is primarily a data trending issue, there is also an architectural reason to keep things as they are. First, we will discuss data trending. The current layout is so robust because each row represents how a product performed for a given year. However, a data analyst can also parse each row by monthly timepoints meaning monthly,

quarterly, and seasonal trending can be performed (as we previously mentioned when addressing data granularity). Losing this functionality would be unacceptable. Our company must know when product sales are performing or underperforming throughout a year. Sales performance measurements are essential for production, supply chain, distribution, and adequately serving our customers. For example, if our company determined that most "growth" products were all selling during the summer months, this would be critical to understanding the markets buying the product and driving this trend. Alternatively, suppose multiple "mature" product sales were steadfast throughout the year, but one product had increased sales around the holiday season (Nov-Dec). This sales uptick would drive the manufacturing schedule and everything downstream.

Some supply chains (like pharmaceuticals) are incredibly complicated and are still relying on old methods of supply chain forecasting (based on Excel spreadsheets)(Merkuryeva et al., 2019). These may be enough to "get by" but are unimpressive and ineffective for a competitive market. If anything, we should move towards more granularity in our data instead of less. Boone et al. (2019) have an extensive review showing a significant industry trend towards real-time data. While many of the markets they mention demand real-time granularity (e.g., finance, theater tickets, tourism, etc.), it would behoove our company to move swiftly towards leveraging real-time data practices. These real-time data practices will give us an edge in online markets where real-time data becomes non-negotiable (Boone et al., 2019). Thus, from the data trending perspective, we should stick with the current format to add more granularity (including the day of the month and time purchased). A real-time data approach would not be possible with the proposed model.

Next, we will discuss data architecture. We are primarily concerned with the layout for the data mart, which will temporarily store data so that analysts can perform trending. In other words, we are concerned with how the table can accommodate changes and how one can utilize the table. One can immediately see the preceding years of sales using the proposed layout and data format. There are fewer rows but more columns. While it helps visualize three years side by side using one table, there are a few issues with this layout. My first question is, how will this look the following year? Is the report meant to pull the current year and two previous years' worth of data? Or would the intent be to add 2015 as a new column? Bobby needs to specify how this report will work. If we will be adding a new column for 2015, this is quite problematic. One would need to amend any new code written for yearly trending on an annual basis to derive any benefit from the latest data.

On the other hand, if a data mart customer is only interested in the most recent three years' data, the code would still require edits (because the column header names would change yearly). While these are minor edits, this causes inefficiency and steals valuable time away from other projects. The current format we use would require no changes to the yearly trending code due to data mart design. We should choose a more streamlined approach.

The proposed format has the benefit of showing the three years' worth of data immediately in the table (which is genuinely helpful when examining the raw table in the data mart). However, this same table could easily be recreated from the current format using code that would remain the same year over year. The proposed data mart format comes at the unacceptable cost of losing the ability to trend over monthly, quarterly, and seasonal time points. Furthermore, this model entails the loss of the ability for real-time data. Thus, we recommend staying with the current ETL layout.

Conclusion

We have elaborated on the ETL process and data mart creation using the 2012 through 2014 sales data. We have created, established, and discussed the ERD of this process, delineating how each table relates to the other tables. Furthermore, we addressed the metadata for each table used - what each table and column represents. Lastly, we answered management's questions regarding the data's granularity, using the data to measure yearly growth, and whether the proposed layout would be appropriate. If there are any further questions, please feel free to reach out to our team.

Appendix A – ETL Process Documentation

There are 3 SQL script files:

1. G2_1_extractandtransform.sql
2. G2_2_load.sql
3. G2_3_proofofconcept.sql

Each script contains step-wise comments to follow along.

The **first** script (G2_1_extractandtransform) includes SQL commands interspersed with MySQL GUI Commands. It extracts, transforms, and aggregates the data as prescribed.

1. Complete Steps 1a and 1b in the G2_1_extractandtransform.sql script
2. Complete Step 2 in the G2_1_extractandtransform.sql script
3. Complete Step 3a through 3c in the G2_1_extractandtransform.sql script

   NOTE:

   Steps 3a through 3c assume that the individual executing the steps knows how to import CSV files into a MySQL database using the Data Table Import Wizard. Our team used default settings during the import of each CSV. These tables are created using the same names as the CSV files (2012_product_data_students, 2013_product_data_students, 2014_product_data_students).
4. Complete Steps 4 through 8 in the G2_1_extractandtransform.sql script

The **second** script (G2_2_load.sql) may be run through the CLI pipe. This script formats and exports the data to a CSV named G2_output_final.csv

The **third** and final script (G2_3_proofofconcept.sql) was used to illustrate the proof of concept to answer the business question regarding the Growth/Mature segments described in the memo.

Figure 1, Create database week 6

```
drop database if exists week6;
Create database week6;
```

Figure 2, Create products Table and Add Primary Key

```
create table products as select distinct product_name from product_bu;
ALTER TABLE products ADD product_id INT PRIMARY KEY AUTO_INCREMENT FIRST;
```

Figure 3, Create orders_working Table

```
create table orders_working as
select product,region,2012 as year,month,quantity, `order total` as order_total
     from 2012_product_data_students
union all
select product,region,2013 as year,month,(quantity_1+quantity_2) as quantity,
        (quantity_1+quantity_2)*`Per-Unit Price` as order_total
     from 2013_product_data_students
union all
select product,region,2014 as year,month,quantity,(`order subtotal`-`quantity discount`) as order_total
     from 2014_product_data_students
order by year, month;
```

Figure 4, Alter orders_working Table and Add Primary and Foreign Keys

```
ALTER TABLE orders_working ADD orders_id INT PRIMARY KEY AUTO_INCREMENT FIRST;
create table orders as
select b.product_id as products_product_id, a.region, a.year, a.month, a.quantity, a.order_total
from orders_working a
left join products b
on a.product=b.product_name;
```

Figure 5, Create prod_bu Table and Establish Foreign Keys

```
create table prod_bu as
select a.Prod_BU_ID, a.Prod_BU_Year,
        b.product_id as products_product_id,c.BU_Id as business_unit_BU_ID
from product_bu a
left join products b
on a.product_name=b.product_name
left join business_unit c on a.BU_name=c.BU_name;
```

Figure 6, Create data_all Table

```
create table data_all as
select business_unit.bu_name,business_unit.bu_designation,products.product_name,region, year,month,quantity,order_total
from orders
left join products on orders.products_product_id=products.product_id
join prod_bu on prod_bu.products_product_id=products.product_id and prod_bu.prod_bu_year=orders.year
join business_unit on prod_bu.business_unit_BU_ID=business_unit.BU_ID
where BU_designation <>'Decline';
```

Figure 7, Create data_aggregate Table

```
create table data_aggregate as
select distinct bu_designation,bu_name,product_name,region, year,month,
            sum(quantity) as `Sum of Quantity`,sum(order_total) as `Sum of Order Total`
from data_all
group by bu_designation, bu_name, product_name, region, year, month
order by bu_designation, bu_name, product_name, region, year, month,`Sum of Quantity`,`Sum of Order Total`;
```

Appendix B – Metadata Documentation

Below is information describing each table present in our Entity Relationship Diagram.

Figure 8 – Orders Table

The orders table is a table that stores data related to each order placed by a customer, with one entry for each order. Note that the products_product_id is a foreign key used to reference the product_id in the products table.

Column	Data type	Null	Key	Description
orders_id	int(11)	No	Primary	Unique identifier for each order, auto-incrementing
product	varchar(25)	Yes		The name of a packaged food product
region	varchar(25)	Yes		The regions within the country
year	int(4)	No		The year the order was placed
month	int(2)	Yes		The month the order was placed
quantity	bigint(20)	Yes		How many items were in a particular order

order_total	bigint(20)	Yes		The per-unit price times the quantity
products_product_id	int(11)	No	Foreign	Used to reference the product_id in the products table

Figure 9 – Products Table

The products table is a table that stores data related to each product manufactured by the company.

Column	Data type	Null	Key	Description
product_id	int(11)	No	Primary	Unique identifier for each product, auto-incrementing
product_name	varchar(25)	Yes		The name of a packaged food product

Figure 10 – Business Unit Table

The business unit table is a table that stores a master list of the company's business units, with one entry for each business unit.

Column	Data type	Null	Key	Description
BU_ID	int(11)	No	Primary	Unique identifier for each business unit, auto-incrementing
BU_Name	varchar(25)	Yes		The business unit name (snack, on the go, energy, etc.)
BU_Designation	varchar(25)	Yes		The business unit designation (growth, mature, etc.)

Figure 11 – Product Business Unit Table

The product business unit table is a table that stores a master list of the company's product business units, with one entry for each product business unit. Note that the products_product_id is a foreign key used to reference the product_id in the products table. Note that the business_unit_BU_ID is a foreign key used to reference the BU_ID in the Business Units table.

Column	Data type	Null	Key	Description
Prod_BU_ID	int(11)	No	Primary	Unique identifier for each product business unit

Prod_BU_Year	int(4)	Yes		The product business unit designation for a year
products_product_id	int(11)	No	Foreign	Used to reference the product_id in the products table
business_unit_BU_ID	int(11)	No	Foreign	Used to reference the BU_ID in the Business Units table

Appendix C – SQL Query Supporting Screen Shots

Figure 12 - Create year_aggregate table

```
#Aggregated orders data by bu_designation, year (rolled up bu_name,product_name,region,month)
create table year_aggregate as
select distinct bu_designation, year,
            sum(`Sum of Quantity`) as quantity ,sum(`Sum of Order Total`) as order_total
from data_aggregate
group by bu_designation, year
order by bu_designation, year;
```

Figure 13 - Calculated Year over Year Growth for Growth Segment

```
#Calculated Year over Year growth for bu_designation='Growth'
# by joining the table on itself to obtain the prior year's data
#You can uncomment the rows below to see the source data and do the calculations yourself
select distinct now.bu_designation, prior.year as year_lag, now.year,
    #   prior.quantity as prior_q, now.quantity as now_q,
((now.quantity-prior.quantity)/prior.quantity)*100 as quantity_YoYgrowth,
    # prior.order_total as prior_o, now.order_total as now_o,
((now.order_total-prior.order_total)/prior.order_total)*100 as order_total_YoYgrowth
#, ordertotal_YoYgrowth
from year_aggregate now
left join year_aggregate prior
on now.bu_designation=prior.bu_designation and now.year=prior.year+1
where now.bu_designation='Growth' and prior.year is not null;
```

Figure 14 - Display Quantity and Order Total for All Years for Mature Segment

```
#B- Mature segment should remain pretty much the same in terms of quantity and order totals
#Look at the Mature segment, previously aggregated by bu_designation and year
select * from year_aggregate where bu_designation='Mature';
```

References

Boone, T., Ganeshan, R., Jain, A., & Sanders, N. R. (2019). Forecasting sales in the supply chain: Consumer analytics in the big data era. International Journal of Forecasting, 35(1), 170-180.

El-Sappagh, S., Hendawi, A., El Bastawissy, A. (2011). A proposed model for data warehouse ETL processes. *Journal of King Saud University – Computer and Information Sciences*, 23, 91-104.

Lucidchart. (2021). *What is an Entity Relationship Diagram (ERD)?* Retrieved from https://www.lucidchart.com/pages/er-diagrams

Merkuryeva, G., Valberga, A., & Smirnov, A. (2019). Demand forecasting in pharmaceutical supply chains: A case study. Procedia Computer Science, 149, 3-10.

Oracle. (2020). *Design the Data Mart: User Manual*. Retrieved from https://docs.oracle.com/cd/E10352_01/doc/bi.1013/e10312/dm_design.htm

W.H. Inmon, & Daniel Linstedt. (2014). *Data Architecture: A Primer for the Data Scientist: Big Data, Data Warehouse and Data Vault.* Morgan Kaufmann.