DAT701
ENTERPRISE DATABASE SOLUTIONS

Assignment Two | Submission Date: 15.11.21
Dale Stephenson | Student ID: 13502967

Contents

Executive Summary

This report outlines the steps and methods deployed to develop a data warehouse capable of answering the business questions outlined in Assignment 1. The report is submitted in conjunction with the SQL queries that are made available in the SQL Server instance, and the data visualisation's produced in the Windows 10 instance of the virtual environment.

The requirements outlined for each question are answered in this document, including all reporting queries, the evidence and descriptions of query performance, and query design. Queries are also available on the author's GitHub repository labelled d-stephenson/DAT701. Queries have been created using t-SQL and formatted for readability following the provided style guidelines.

Part One: Data Warehouse Design

M1.1. Dimensional Model

The developed data warehouse model is designed to meet the business requirements highlighted in Assignment 1 and utilises the methodology outlined by Moody et al. (2000) for the transforming of entity relational databases to dimensional data warehouses. The steps taken are described by Moody et al. (2000), which includes the classification of entities, the identification of hierarchies, and the aggregation of transactional data to form the dimensional model. Two design schemas have been considered and explored, a *star* schema and a *fact constellation* schema.

The benefit of deploying a dimensional schema for an enterprise data warehouse is the ability for end-users, particularly non-technical users, to produce business-level analytics and reporting (Moody et al., 2000). This is commonly deployed with user-friendly querying and business intelligence tools such as Power BI, these systems require no knowledge of SQL or other technical coding languages. Dimensional model databases are considered read-only and designed for data retrieval for analytics. Updates are normally performed with a batch extract, transform and load (ETL) (Moody et al., 2000). ETL is best performed during system downtime, preferably overnight so they do not affect query performance. Entities are classified into three categories as outlined in Table 1.

Transaction	Component	Classification
Transaction entities are records relating to particular events that occur, such as sales orders, travel bookings, salary payments etc. These are the events that will be analysed by stakeholders to gain greater business insight. These entities are known as measures or quantities and will be identified by dollar figures or weights, for example. These entities are contained within the fact tables. Consideration should be made to those transaction entities that are capable of answering the business questions.	Component entities enable transaction entities to be described through direct one-to-many relationships. These answer the *who, what, when, where, how,* and *why*. Dimension tables are formed based on the component entities and can include the salesperson (*who*), the product (*what)*, the location (*where*), and the period (*when*).	Classification entities are directly or transitively functionally dependent on component entities through one-to-many relationships. Classification entities may be collapsed into component entities to form dimension tables through their embedded hierarchy. FinanceDB contains classification entities such as *Segment* and *Country* that can be collapsed into the component, *Region* entity. Likewise, *ProductCost* is a classification of *Product* that can

		be collapsed into a DimProduct table.
FinanceDB Classifications		
Transaction	**Component**	**Classification**
SalesOrderLineItem SalesOrder	Promotion Product SalesRegion SalesPerson	ProductCost Region Country Segment SalesKPI

Table 1 *Entity Classifications for the FinanceDB database*

The classifications form a precedence hierarchy with transaction entities forming the highest precedence, through to classification entities, and ending with component entities as the lowest precedence. Entities are capable of being classified as either component or classification, where this is the case, the entity should be classified in the classification category. When entities do not fit into the hierarchical structure they are not capable of being included in the schema.

The following is the process deployed to identify the hierarchies:

- Product > Promotion > SalesOrderLineItem
  - Product is the parent of Promotion
  - Promotion is the child of Product
  - SalesOrderLineItem and Promotion are descendants of Product
  - Product and Promotion are ancestors of SalesOrderLineItem
- Product > SalesOrderLineItem
- Product > Product Cost
- Country > Product Cost
- Country > Region > SalesRegion > SalesOrder > SalesOrderLineItem
  - Country is the parent of Region
  - Region is the child of Country
  - SalesOrderLineItem, SalesOrder, SalesRegion and Region are descendants of Country
  - Country, Region, SalesRegion and SalesOrder are ancestors of SalesOrderLineItem
- Segment > Region > SalesRegion > SalesOrder > SalesOrderLineItem
  - Country is the parent of Segment
  - Segment is the child of Country
  - SalesOrderLineItem, SalesOrder, SalesRegion and Region are descendants of Segment
  - Segment, Region, SalesRegion and SalesOrder are ancestors of SalesOrderLineItem
- SalesPerson > SalesKPI
- SalesPerson > SalesRegion > SalesOrder > SalesOrderLineItem
  - SalesPerson is the parent of SalesRegion
  - SalesRegion is the child of SalesPerson
  - SalesOrderLineItem, SalesOrder, SalesRegion are descendants of SalesPerson
  - SalesPerson, SalesRegion and SalesOrder are ancestors of SalesOrderLineItem
- SalesPerson > SalesOrder > SalesOrderLineItem

- SalesPerson is the parent of SalesOrder
- SalesOrder is the child of SalesPerson
- SalesOrderLineItem and SalesOrder are descendants of SalesPerson
- SalesPerson and SalesOrder are ancestors of SalesOrderLineItem

Entities at the bottom of the hierarchy are referred to as *minimal*, entities at the top of the hierarchy are referred to as *maximal*. The FinanceDB OLTP database contains four maximal entities and three minimal entities that are listed in Table 2.

Maximal Entities	Quantity	Minimal Entities	Quantity
Product	3	SalesKPI	1
Country	2	ProductCost	2
Segment	1	SalesOrderLineItem	6
SalesPerson	3		

Table 2 *Maximal and Minimal Entities for FinanceDB database*

The hierarchical structure can be used to collapse lower-level entities into higher-level entities. Lower-level entities in the FinanceDB relational tables can be collapsed as follows:

- ProductCost into Product
- Promotion into Product
- Country Into Region
- Segment into Region
- SalesKPI into SalesPerson
- SalesOrderLineItem into SalesOrder

The higher-level entity receives the attributes of the collapsed table resulting in transitive dependency redundancy. This is a form of denormalisation that violates Codd's third normal form concerning database schema design.

Additional entities can be created that contain summarised data in the schema design. The new entity stores aggregate attributes selected from the entity source, which is then grouped using a further subset of attributes. In the FinanceDB OLTP database, an entity could be created that summarises product sales by region. The table will total the sales by region, allowing for the average quantity and price to be aggregated for sales in each given region for a given time, using the date dimension. It is important to note that the data cannot be reconstructed, meaning information is lost as a result of the aggregation.

M1.2. Dimensional Schema Design

The initial design consideration was a dimensional model that consisted of a *star* schema, a single fact table surrounded by the dimension tables. The key of the fact table is formed through the combination of foreign keys created through the relationships with the component entities. Initially, all dimensions were included in the schema to create a model that was future-proofed, with *Promotion* acting as a classification entity and not a component entity. Consideration was made as to the category type of the *Promotion* table, and certainly, an argument can be made either way given the ambiguity. However, the model states that should an entity be capable of being classified as either a classification entity or a component entity, then the classification entity will prevail.

Ultimately, a *fact constellation* table was chosen that was better able to support the sales performance business process and the sales order business process, and the corresponding granularity level required to drill down and provide the detail necessary to answer the business questions.

- **DimDate** New table generated specifically for the dimensional schema
- **DimSalesPerson** [SalesYear, SalesKPI] *SalesPersonID, FirstName, LastName, Gender, HireDate, DayOfBirth, DaysOfLeave, DaysOfSickLeave*
- **DimSalesLocation** [Country/Segment > Region] *RegionID, CountryID, SegmentID, Country, Segment*
- **DimProduct** [Product > Product Cost] *ProductID, Product Name, PromotionYear, Discount, ManufacturingPrice, RRP*
- **FactSalePerformance** [SalesKPI > SalesOrder > SalesOrderLineItem]
- **FactSaleOrder** [SalesOrder > SalesOrderLineItem] *SalesOrderID, UnitsSold, SalePrice*

Figure 1 shows the initial *star* schema design. The developed *fact constellation* schema version 3.2 is shown in Figure 2 in its late-stage iteration. Primary keys have not been enforced on the dimensional schema. The primary keys have been taken from the OLTP database for the dimension tables and act as foreign key references in the fact tables. The reason for this decision is that multiple entries of the same entity are present as a result of the hierarchical structure, collapsing lower classification entities into their component entity.

Two paths have been identified for the DimDate table, the first was to treat this like the other dimension tables and insert the dates directly from the OLTP database. The second option was to research options for a more comprehensive DimDate table that included a finer grain of date related information. Several options were tested, with the final solution offering a suitable trade-off for the data set in FinanceDB, whilst giving consideration to future-proofing of the data warehouse. The date range of this table has been set from the first indicated data entry into FinanceDB and finishes on the last day of 2030. The greater level of detail in the date table allows for increased flexibility for a non-technical user of the business intelligence system.

M1.3. Schema – Draft Dimensional Model

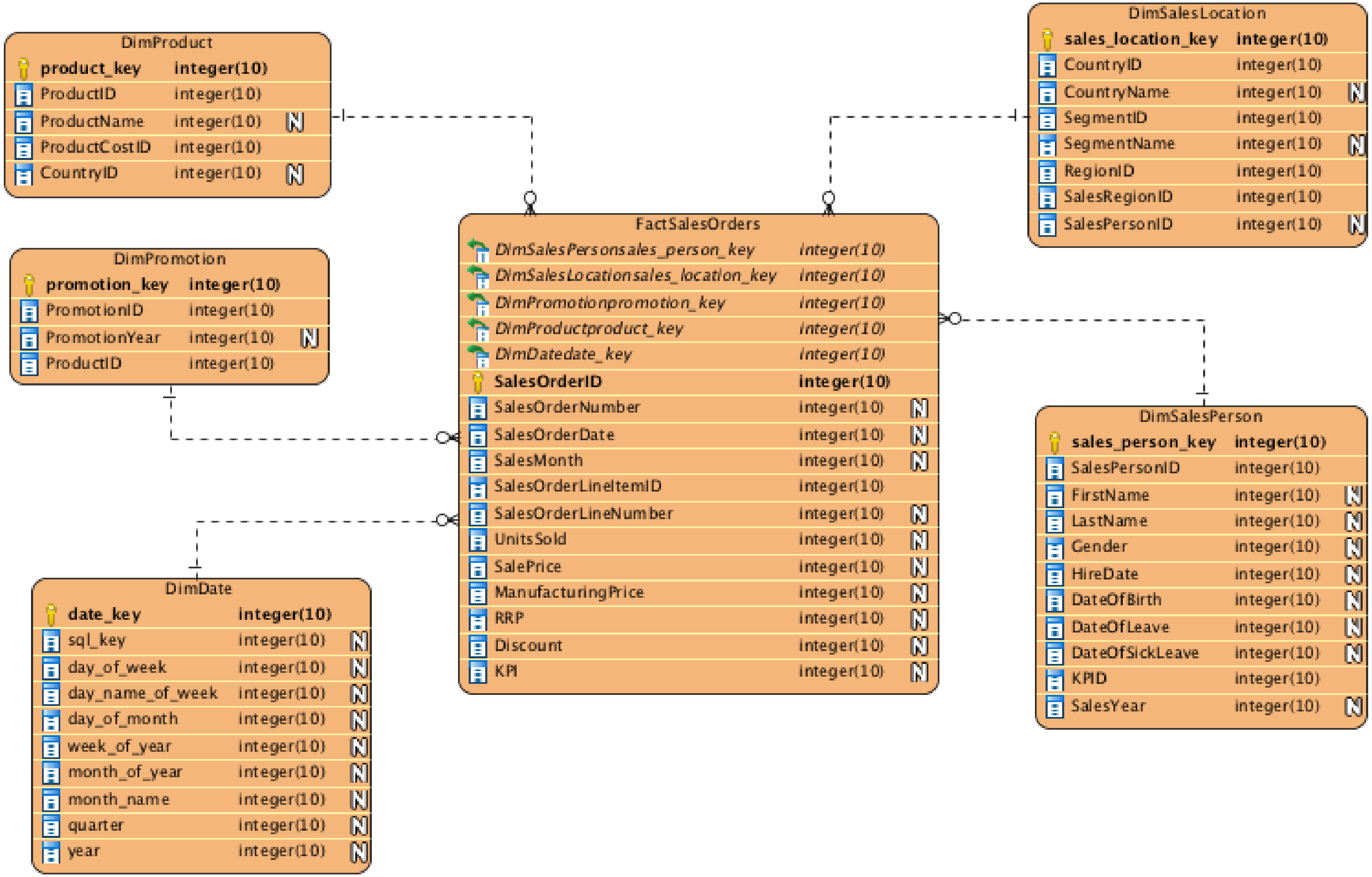
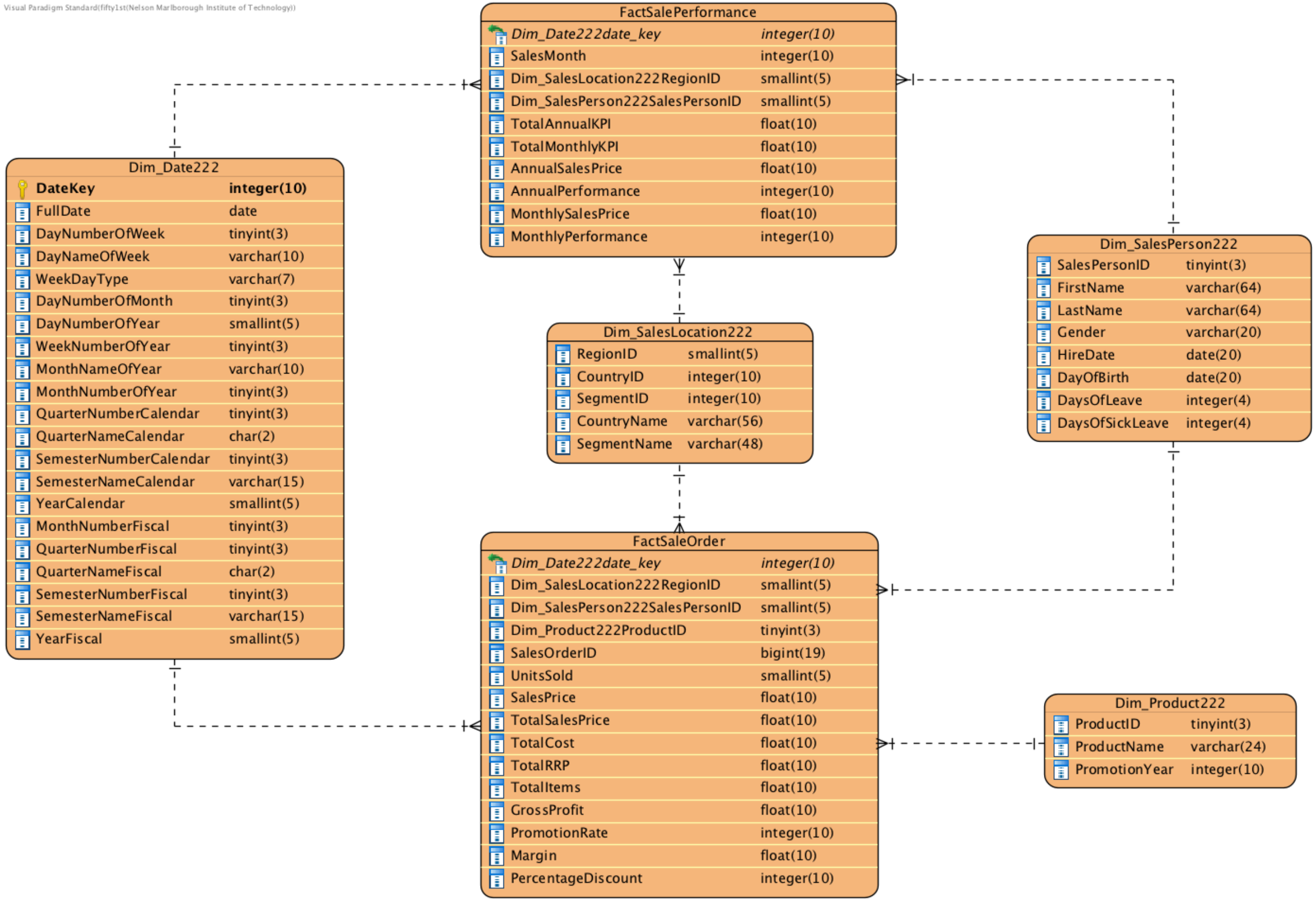

DimProduct
product_key integer(10)
ProductID integer(10)
ProductName integer(10) N
ProductCostID integer(10)
CountryID integer(10) N
DimPromotion
promotion_key integer(10)
PromotionID integer(10)
PromotionYear integer(10) N
ProductID integer(10)
DimDate
date_key integer(10)
sql_key integer(10) N
day_of_week integer(10) N
day_name_of_week integer(10) N
day_of_month integer(10) N
week_of_year integer(10) N
month_of_year integer(10) N
month_name integer(10) N
quarter integer(10) N
year integer(10) N
FactSalesOrders
DimSalesPersonsales_person_key integer(10)
DimSalesLocationsales_location_key integer(10)
DimPromotionpromotion_key integer(10)
DimProductproduct_key integer(10)
DimDatedate_key integer(10)
SalesOrderID integer(10)
SalesOrderNumber integer(10) N
SalesOrderDate integer(10) N
SalesMonth integer(10) N
SalesOrderLineItemID integer(10)
SalesOrderLineNumber integer(10) N
UnitsSold integer(10) N
SalePrice integer(10) N
ManufacturingPrice integer(10) N
RRP integer(10) N
Discount integer(10) N
KPI integer(10) N
DimSalesLocation
sales_location_key integer(10)
CountryID integer(10)
CountryName integer(10) N
SegmentID integer(10)
SegmentName integer(10) N
RegionID integer(10)
SalesRegionID integer(10)
SalesPersonID integer(10) N
DimSalesPerson
sales_person_key integer(10)
SalesPersonID integer(10)
FirstName integer(10) N
LastName integer(10) N
Gender integer(10) N
HireDate integer(10) N
DateOfBirth integer(10) N
DateOfLeave integer(10) N
DateOfSickLeave integer(10) N
KPID integer(10)
SalesYear integer(10) N

***Figure 1** First Iteration – Investigating Draft Star Schema Implementation*

M1.4. Fact Constellation Schema – Draft Dimensional Model



FactSalePerformance	
Dim_Date222date_key	*integer(10)*
SalesMonth	integer(10)
Dim_SalesLocation222RegionID	smallint(5)
Dim_SalesPerson222SalesPersonID	smallint(5)
TotalAnnualKPI	float(10)
TotalMonthlyKPI	float(10)
AnnualSalesPrice	float(10)
AnnualPerformance	integer(10)
MonthlySalesPrice	float(10)
MonthlyPerformance	integer(10)

Dim_Date222	
DateKey	**integer(10)**
FullDate	date
DayNumberOfWeek	tinyint(3)
DayNameOfWeek	varchar(10)
WeekDayType	varchar(7)
DayNumberOfMonth	tinyint(3)
DayNumberOfYear	smallint(5)
WeekNumberOfYear	tinyint(3)
MonthNameOfYear	varchar(10)
MonthNumberOfYear	tinyint(3)
QuarterNumberCalendar	tinyint(3)
QuarterNameCalendar	char(2)
SemesterNumberCalendar	tinyint(3)
SemesterNameCalendar	varchar(15)
YearCalendar	smallint(5)
MonthNumberFiscal	tinyint(3)
QuarterNumberFiscal	tinyint(3)
QuarterNameFiscal	char(2)
SemesterNumberFiscal	tinyint(3)
SemesterNameFiscal	varchar(15)
YearFiscal	smallint(5)

Dim_SalesPerson222	
SalesPersonID	tinyint(3)
FirstName	varchar(64)
LastName	varchar(64)
Gender	varchar(20)
HireDate	date(20)
DayOfBirth	date(20)
DaysOfLeave	integer(4)
DaysOfSickLeave	integer(4)

Dim_SalesLocation222	
RegionID	smallint(5)
CountryID	integer(10)
SegmentID	integer(10)
CountryName	varchar(56)
SegmentName	varchar(48)

FactSaleOrder	
Dim_Date222date_key	*integer(10)*
Dim_SalesLocation222RegionID	smallint(5)
Dim_SalesPerson222SalesPersonID	smallint(5)
Dim_Product222ProductID	tinyint(3)
SalesOrderID	bigint(19)
UnitsSold	smallint(5)
SalesPrice	float(10)
TotalSalesPrice	float(10)
TotalCost	float(10)
TotalRRP	float(10)
TotalItems	float(10)
GrossProfit	float(10)
PromotionRate	integer(10)
Margin	float(10)
PercentageDiscount	integer(10)

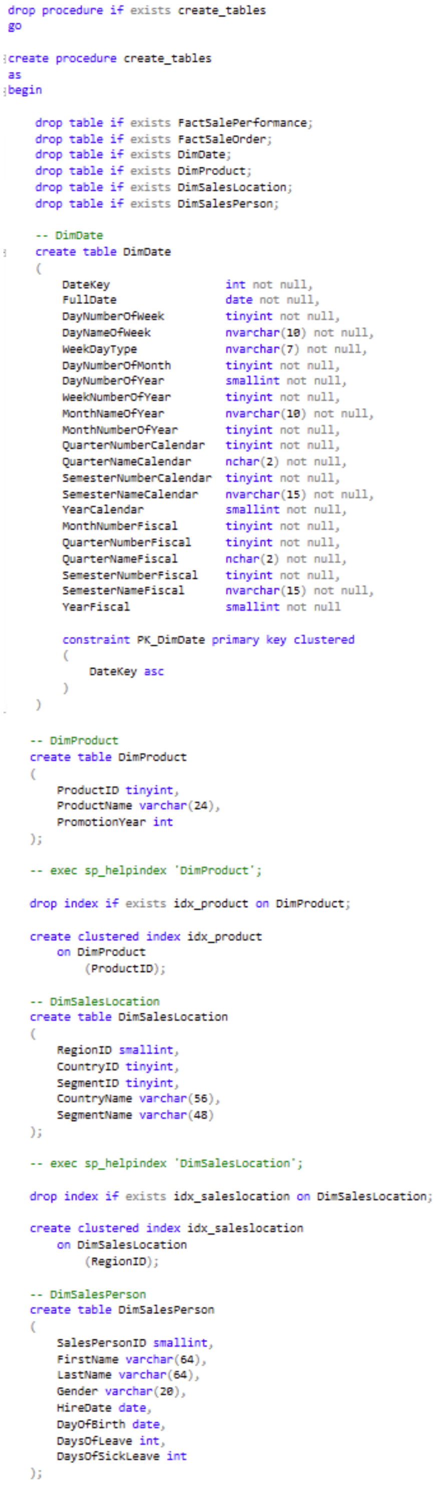
Dim_Product222	
ProductID	tinyint(3)
ProductName	varchar(24)
PromotionYear	integer(10)

Figure 2 *Late Iteration Version 3.2 – Fact Constellation Schema Implementation*

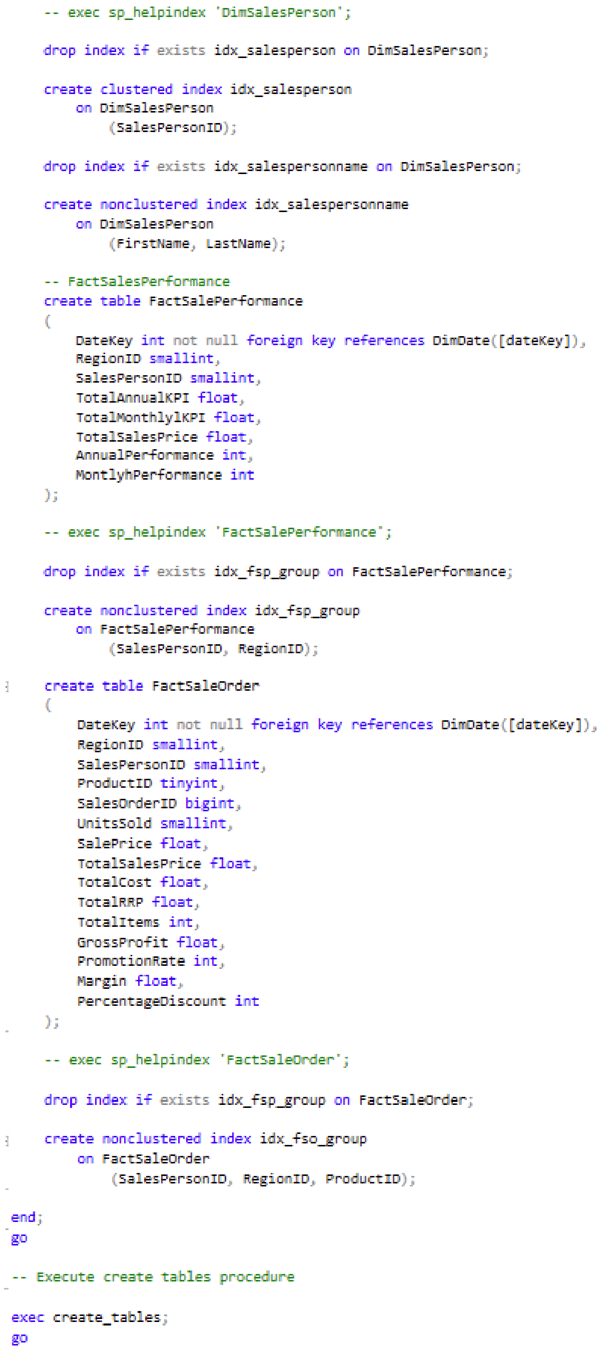
Part Two: Data Warehouse DDL

M2.1. Data Definition Language DDL

Figure 3 is a view of the data definition language (DDL) used to meet the schema design. Column data types match the attributes in the FinanceDB database. A primary key constraint has been created against the date key in the DimDate table. Primary key constraints have not been created on the other dimension tables because the fact table granularity is such that the relationship references will replicated in these tables, instead a clustered index has been created against these references.



```sql
drop procedure if exists create_tables
go

create procedure create_tables
as
begin

    drop table if exists FactSalePerformance;
    drop table if exists FactSaleOrder;
    drop table if exists DimDate;
    drop table if exists DimProduct;
    drop table if exists DimSalesLocation;
    drop table if exists DimSalesPerson;

    -- DimDate
    create table DimDate
    (
        DateKey                 int not null,
        FullDate                date not null,
        DayNumberOfWeek         tinyint not null,
        DayNameOfWeek           nvarchar(10) not null,
        WeekDayType             nvarchar(7) not null,
        DayNumberOfMonth        tinyint not null,
        DayNumberOfYear         smallint not null,
        WeekNumberOfYear        tinyint not null,
        MonthNameOfYear         nvarchar(10) not null,
        MonthNumberOfYear       tinyint not null,
        QuarterNumberCalendar   tinyint not null,
        QuarterNameCalendar     nchar(2) not null,
        SemesterNumberCalendar  tinyint not null,
        SemesterNameCalendar    nvarchar(15) not null,
        YearCalendar            smallint not null,
        MonthNumberFiscal       tinyint not null,
        QuarterNumberFiscal     tinyint not null,
        QuarterNameFiscal       nchar(2) not null,
        SemesterNumberFiscal    tinyint not null,
        SemesterNameFiscal      nvarchar(15) not null,
        YearFiscal              smallint not null

        constraint PK_DimDate primary key clustered
        (
            DateKey asc
        )
    )

   -- DimProduct
   create table DimProduct
   (
       ProductID tinyint,
       ProductName varchar(24),
       PromotionYear int
   );

   -- exec sp_helpindex 'DimProduct';

   drop index if exists idx_product on DimProduct;

   create clustered index idx_product
       on DimProduct
           (ProductID);

   -- DimSalesLocation
   create table DimSalesLocation
   (
       RegionID smallint,
       CountryID tinyint,
       SegmentID tinyint,
       CountryName varchar(56),
       SegmentName varchar(48)
   );

   -- exec sp_helpindex 'DimSalesLocation';

   drop index if exists idx_saleslocation on DimSalesLocation;

   create clustered index idx_saleslocation
       on DimSalesLocation
           (RegionID);

   -- DimSalesPerson
   create table DimSalesPerson
   (
       SalesPersonID smallint,
       FirstName varchar(64),
       LastName varchar(64),
       Gender varchar(20),
       HireDate date,
       DayOfBirth date,
       DaysOfLeave int,
       DaysOfSickLeave int
   );
```



```
    -- exec sp_helpindex 'DimSalesPerson';

    drop index if exists idx_salesperson on DimSalesPerson;

    create clustered index idx_salesperson
        on DimSalesPerson
            (SalesPersonID);

    drop index if exists idx_salespersonname on DimSalesPerson;

    create nonclustered index idx_salespersonname
        on DimSalesPerson
            (FirstName, LastName);

    -- FactSalesPerformance
    create table FactSalePerformance
    (
        DateKey int not null foreign key references DimDate([dateKey]),
        RegionID smallint,
        SalesPersonID smallint,
        TotalAnnualKPI float,
        TotalMonthlylKPI float,
        TotalSalesPrice float,
        AnnualPerformance int,
        MontlyhPerformance int
    );

    -- exec sp_helpindex 'FactSalePerformance';

    drop index if exists idx_fsp_group on FactSalePerformance;

    create nonclustered index idx_fsp_group
        on FactSalePerformance
            (SalesPersonID, RegionID);

    create table FactSaleOrder
    (
        DateKey int not null foreign key references DimDate([dateKey]),
        RegionID smallint,
        SalesPersonID smallint,
        ProductID tinyint,
        SalesOrderID bigint,
        UnitsSold smallint,
        SalePrice float,
        TotalSalesPrice float,
        TotalCost float,
        TotalRRP float,
        TotalItems int,
        GrossProfit float,
        PromotionRate int,
        Margin float,
        PercentageDiscount int
    );

    -- exec sp_helpindex 'FactSaleOrder';

    drop index if exists idx_fsp_group on FactSaleOrder;

    create nonclustered index idx_fso_group
        on FactSaleOrder
            (SalesPersonID, RegionID, ProductID);

end;
go

-- Execute create tables procedure

exec create_tables;
go
```

Figure 3 *DDL in relation to v.3.2 Fact Constellation Dimensional Model Schema*

M2.2. Indexes

The DDL includes clustered and non-clustered indexes that have been created with consideration made to the results produced in Assignment 1. The DimDate table contains a clustered index against the date key. Currently, there is no requirement to create non-clustered indexes for this table. However, with the data loaded into the warehouse, performance tests can be run to determine if a non-clustered index would be beneficial against the *MonthNumberOfYear,* given the granularity of the data month is likely to be the most commonly used column to perform queries that are capable of answering the business questions.

Clustered indexes have been created for the remaining dimension tables against the relevant primary keys from the OLTP database. Due to the size of the data set concerning these tables, it is anticipated that there will be no further requirement to create further non-clustered indexes. However, as sales representatives *First* and *Last* names will be selected in the performance queries, to support query performance a non-clustered index has been created against these to future proof against a growing data set.

The fact tables contain clustered indexes that have been created in conjunction with table partitioning, this is detailed in the following sub-section M2.3. Non-clustered indexes have been created that include all relationship IDs that act as foreign key references to the dimension tables, with *FactSaleOrder* containing a further non-clustered index on *SalesOrderID*, which effectively acts as the key identifier for the orders contained in this fact table.

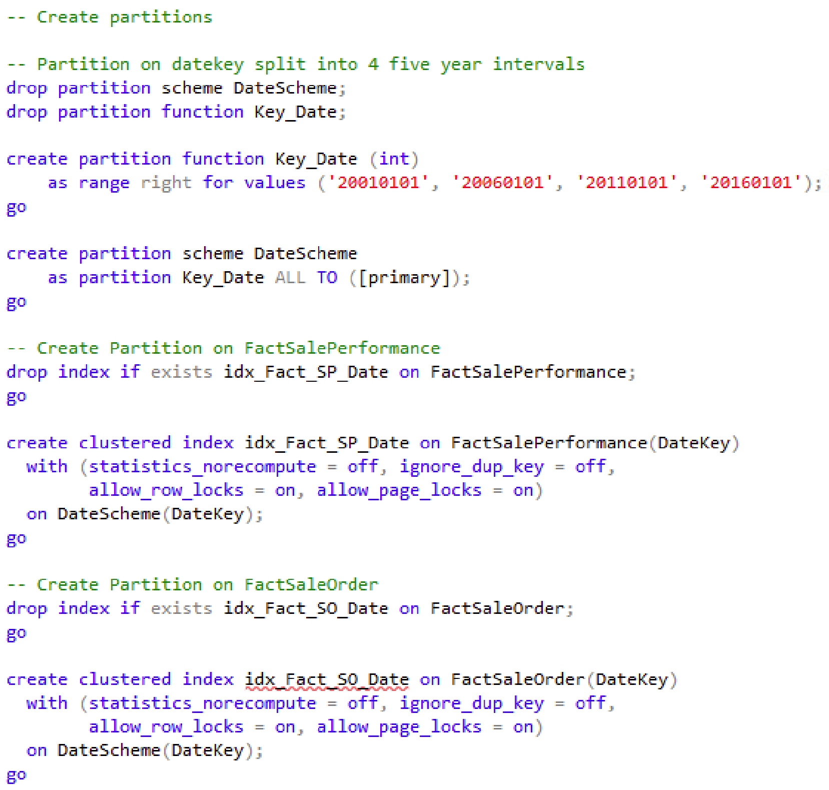
M2.3. Table Partitioning

Table partitioning has been carefully applied against columns that are anticipated to be used most often for the analytical reporting, in this instance partitioning is performed against the date column included in both fact tables. The date column is an ideal candidate for table partitioning because it is low cardinality, having low distinct values concerning the number of rows contained in the column.

Two approaches to table partitioning have been considered, the first was partitioning using the built-in wizard available in SQL Server, and the second is performing table partitioning in conjunction with the index clustering on the fact tables. The latter has been performed as outlined in Figure 4a, with the result observed in Figures 4b, 4c, and 4d.

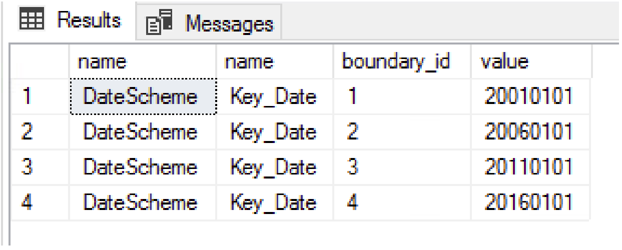
The partition has been created using a partition function and a partition scheme that can be applied against both fact tables on the date key. The scheme must be created before the function is created. Once both are created the table can be partitioned. As FinanceDW is a new data warehouse schema, there was no necessity to drop any clustered indexes against the date key, meaning the index can be created directly against these columns as shown in Figure 4a. The vertical partitioning against the date column has been split into four sections as groups of five years covering the total period directly relating to the data set.

In addition to the partitions and clustered indexes against the date dimension *DateKey* column in the two fact tables, additional non-clustered indexes have been created against the fact tables and the sales representative dimension table. The fact tables include non-clustered indexes against the relationship columns that are linked to the dimension tables, which would ordinarily be classified as foreign key relations. In the sales representative dimension table, a non-clustered index has been created against the first and last names of the sales representatives, as these are most likely to be searched for querying and data analytics. However, it is recognised that given the relatively small size of this table, the effect this is likely to have on overall performance is expected to be minimal.



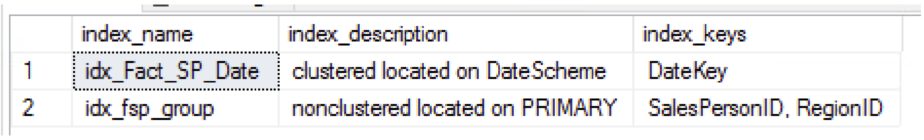
```
-- Create partitions

-- Partition on datekey split into 4 five year intervals
drop partition scheme DateScheme;
drop partition function Key_Date;

create partition function Key_Date (int)
    as range right for values ('20010101', '20060101', '20110101', '20160101');
go

create partition scheme DateScheme
    as partition Key_Date ALL TO ([primary]);
go

-- Create Partition on FactSalePerformance
drop index if exists idx_Fact_SP_Date on FactSalePerformance;
go

create clustered index idx_Fact_SP_Date on FactSalePerformance(DateKey)
  with (statistics_norecompute = off, ignore_dup_key = off,
        allow_row_locks = on, allow_page_locks = on)
  on DateScheme(DateKey);
go

-- Create Partition on FactSaleOrder
drop index if exists idx_Fact_SO_Date on FactSaleOrder;
go

create clustered index idx_Fact_SO_Date on FactSaleOrder(DateKey)
  with (statistics_norecompute = off, ignore_dup_key = off,
        allow_row_locks = on, allow_page_locks = on)
  on DateScheme(DateKey);
go
```

Figure 4a *Partition code to create table partitioning*



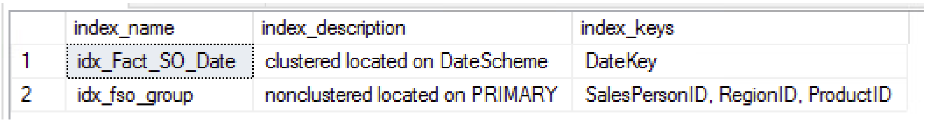
	name	name	boundary_id	value
1	DateScheme	Key_Date	1	20010101
2	DateScheme	Key_Date	2	20060101
3	DateScheme	Key_Date	3	20110101
4	DateScheme	Key_Date	4	20160101

Figure 4b *Defined partition structure*



	index_name	index_description	index_keys
1	idx_Fact_SP_Date	clustered located on DateScheme	DateKey
2	idx_fsp_group	nonclustered located on PRIMARY	SalesPersonID, RegionID

Figure 4c *Partition applied to FactSalePerformance on the clustered index*



	index_name	index_description	index_keys
1	idx_Fact_SO_Date	clustered located on DateScheme	DateKey
2	idx_fso_group	nonclustered located on PRIMARY	SalesPersonID, RegionID, ProductID

Figure 4d *Partition applied to FactSaleOrder on the clustered index*

M2.4. Recovery Model

Recovery models in SQL Server control how transactions are logged, the type of available restore operations, and whether a transaction log is allowed to be backed up or needs to be backed up. There are three recovery models available:

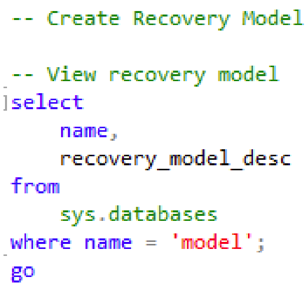
1. Simple
2. Full
3. Bulk-logged

The FinanceDW data warehouse uses the *full* recovery model by default. Should this not meet the technical requirements the data warehouse can be switched to an alternate recovery model using a `alter database` query, for example setting `model set recovery simple`.

A review of the three recovery models allowed a determination to be made, indicating that the default *full* model was suitable for the FinanceDW data warehouse. The *simple* model was deemed to be insufficient as it prioritises disk space at the expense of log backups, meaning changes since the last backup are exposed to loss. The *full* model in comparison allows recovery to a point in time resulting in limited exposure to loss.

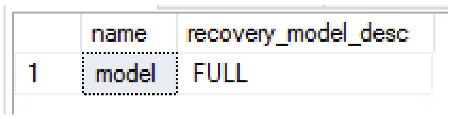
The *bulk-logged* model is an adjunct of the *full* model, however, this model reduces log space by limiting logs against bulk operations. If damage occurs to the log since the most recent backup there will be exposure to loss. Additionally, recovery can only be performed to the end of a performed backup, the model does not support point in time recovery.

The SQL statement used to confirm the *full* recovery model in the FinanceDW data warehouse can be viewed in Figure 5a, the result of which is shown in Figure 5b.



```
-- Create Recovery Model

-- View recovery model
select
    name,
    recovery_model_desc
from
    sys.databases
where name = 'model';
go
```

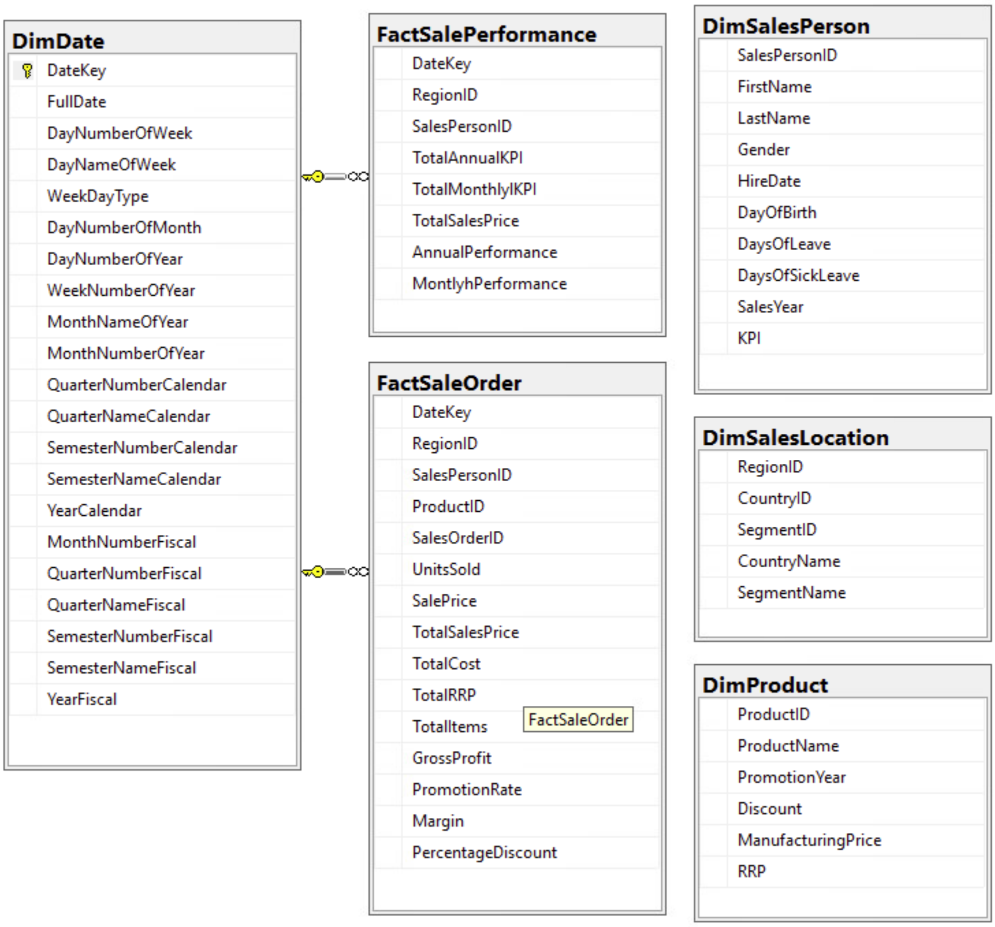
Figure 5a *Query used to confirm default recovery model*



	name	recovery_model_desc
1	model	FULL

Figure 5b *Results of recovery model query*

M2.5. FinanceDW Physical Diagram



DimDate

- DateKey
- FullDate
- DayNumberOfWeek
- DayNameOfWeek
- WeekDayType
- DayNumberOfMonth
- DayNumberOfYear
- WeekNumberOfYear
- MonthNameOfYear
- MonthNumberOfYear
- QuarterNumberCalendar
- QuarterNameCalendar
- SemesterNumberCalendar
- SemesterNameCalendar
- YearCalendar
- MonthNumberFiscal
- QuarterNumberFiscal
- QuarterNameFiscal
- SemesterNumberFiscal
- SemesterNameFiscal
- YearFiscal

FactSalePerformance

- DateKey
- RegionID
- SalesPersonID
- TotalAnnualKPI
- TotalMonthlyKPI
- TotalSalesPrice
- AnnualPerformance
- MontlyhPerformance

FactSaleOrder

- DateKey
- RegionID
- SalesPersonID
- ProductID
- SalesOrderID
- UnitsSold
- SalePrice
- TotalSalesPrice
- TotalCost
- TotalRRP
- TotalItems
- GrossProfit
- PromotionRate
- Margin
- PercentageDiscount

DimSalesPerson

- SalesPersonID
- FirstName
- LastName
- Gender
- HireDate
- DayOfBirth
- DaysOfLeave
- DaysOfSickLeave
- SalesYear
- KPI

DimSalesLocation

- RegionID
- CountryID
- SegmentID
- CountryName
- SegmentName

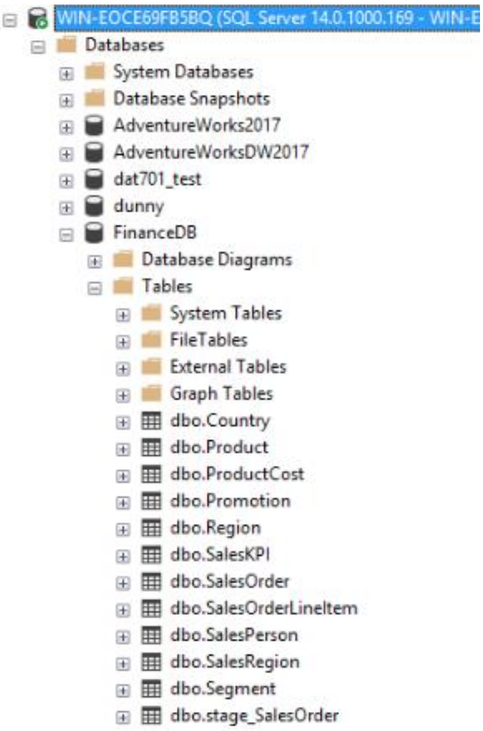
DimProduct

- ProductID
- ProductName
- PromotionYear
- Discount
- ManufacturingPrice
- RRP

Figure 6 *Physical Diagram generated in SQL Server*

M2.6. FinanceDB Installation

Figure 7 has been produced as evidence of the FinanceDB backup installation in virtual machine -*SS*, as outlined in Assignment 1.



- WIN-EOCE69FB5BQ (SQL Server 14.0.1000.169 - WIN-E
  - Databases
    - System Databases
    - Database Snapshots
    - AdventureWorks2017
    - AdventureWorksDW2017
    - dat701_test
    - dunny
    - FinanceDB
      - Database Diagrams
      - Tables
        - System Tables
        - FileTables
        - External Tables
        - Graph Tables
        - dbo.Country
        - dbo.Product
        - dbo.ProductCost
        - dbo.Promotion
        - dbo.Region
        - dbo.SalesKPI
        - dbo.SalesOrder
        - dbo.SalesOrderLineItem
        - dbo.SalesPerson
        - dbo.SalesRegion
        - dbo.Segment
        - dbo.stage_SalesOrder

Figure 7 *Backup FinanceDB installation*

Part Three: ETL

M3.1. Business Process

A business process in dimensional modelling is identified and described through the classification of a set of related business activities that have the highest importance to stakeholders. Business processes will not necessarily relate to business departments, and instead might revolve around customers, sales, profit, and products. For example, an organisation might have a sales department and marketing department, both will be interested in order data as the recognised business process. Dimensional models should be constructed around the business process to limit duplication or data redundancy, which can lead to problems and inconsistencies with data quality. In the case of FinanceDB, two business processes have been identified:

1. Sales Performance
2. Sales Orders

After the business processes have been identified they must be prioritised. To clarify the priority level several criteria must be considered:

- Strategic significance and scope of the business process
- Availability and quality of data in the source system
- Feasibility of the process and its complexity
- Other factors specified as important to the business process

Defining business processes allows for the high-level entities that are common across the processes to be identified and grouped to create shared dimensions. In the case of FinanceDB, these have been determined as:

1. Date
2. Sales Representative
3. Region
4. Product

It is important to identify these common entities early in the development lifecycle, to allow business stakeholders to agree on the definitions to ensure consistency across the organisation. This is crucial to mitigate the risk of entities changing once the data warehouse is in production, which can impact the ETL process and applications that rely on the definitions, such as the business intelligence tools.

M3.2. Granularity

The level of granularity that will be required in the fact tables can be determined from the business processes, and the requirements gathering conducted as part of the systems development. Documents collected in the requirements gathering process will often contain information that can be used to define the grain of measurement stores in the fact tables. The level of the grain should convey the exact contents of a measurement record stored in the fact table. The greater the level of detail included, the lower the level of granularity, conversely, if there is less detail included in the record, the granularity level will be higher.

The level of detail available in a dimensional schema is the *grain*, and each fact and dimension table in the schema will have separate granularity. The dimensional schema's grain is defined as the finest level of detail that can be realised by joining the fact and dimension tables. The grain defined in the FinanceDW model consists of dimensions *Date*, *Sales Representative*, *Product*, and *Region,* meaning the granularity obtained extends to the *date* a *product* was sold by *region* and *sales representative*.

The grain of the fact tables is based on the business questions listed in Assignment 1. In a business scenario, additional information may be produced such as invoices, order forms, and any access granted to business operation applications. These documents and applications can also point to entities such as customer or product that can form the dimension tables. High-level or preliminary measures that are easy to identify should be considered first to define the grain, these measures can include unit price, manufacturing price, and cost. These measures may not be the final set of measurements included in the fact tables, however, they can support formal measures for the fact tables.

The FinanceDW data warehouse contains two business processes that form the two fact tables. Several preliminary measurements have been incorporated in the *FactSaleOrder* table to support future proofing and to be able to quickly answer business questions that may not have been identified in the requirements gathering process. The measurements in the fact tables extend to the dimension tables, which includes data that allows performance data to be aggregated annually and monthly, and further related to both the sales representative and region tables. The sales order table aggregates measurements to total sales relating to an order, with granularity to the product level. The fact table measurements constitute multiple granularities within the single fact, the columns have been named to specifically identify the granularity level of the records represented by the column, such as *TotalAnnualKPI* and *TotalMonthlyKPI*.

1. Sale Performance (Fact)
    a. By Date (Granularity to the Month) (Dim)
    b. By Region (Granularity to the Country and Segment) (Dim)
    c. By Sales Representative (Granularity to the Sales Representative Name) (Dim)
2. Sale Order (Granularity to the Sales Order ID) (Fact)
    a. By Date (Granularity to the Day) (Dim)
    b. By Region (Granularity to the Country and Segment) (Dim)
    c. By Sales Representative (Granularity to the Sales Representative Name) (Dim)
    d. By Product (Granularity to the Product Name) (Dim)

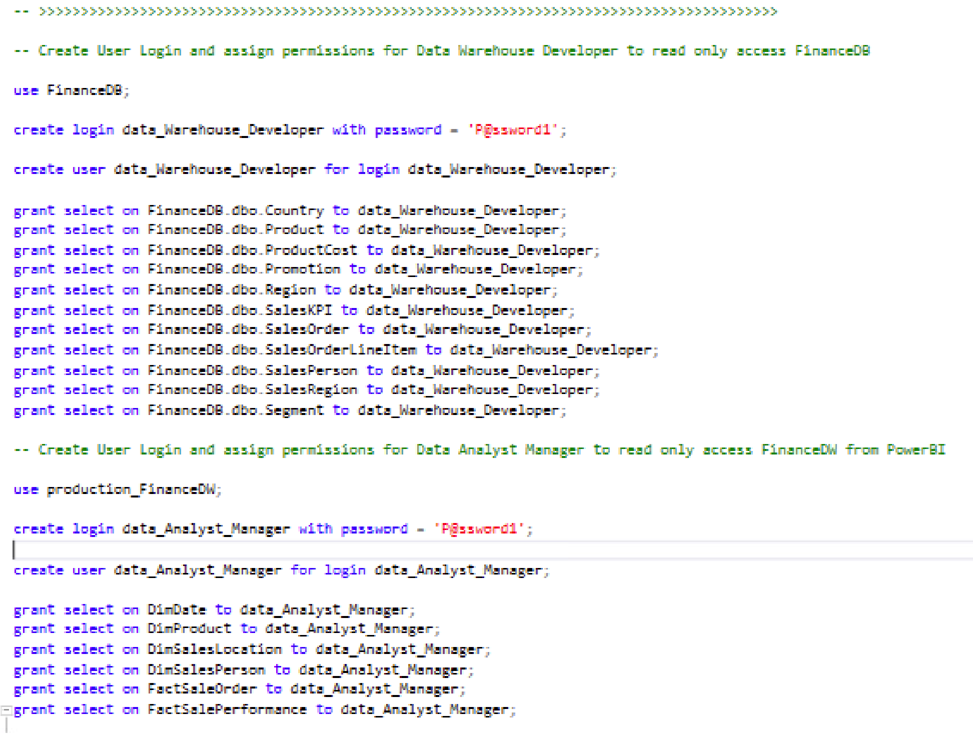
M3.3. User Access & Security

To ensure the operational integrity of the OLTP FinanceDB database, access permissions have been granted to the Data Warehouse Developer. The developer does not require permissions that extend beyond READ, therefore `select` has been granted to this user. The permissions granted are a set of governing rules that determine the access level given to a user. Further permissions can be applied such as `grant`, `revoke`, and `deny`. Two further users have been created in the FinanceDW data warehouse for the data analysts to READ the data from the business intelligence tool.

Users have been set up using two available processes in SQL Server:

1. T-SQL statement
2. SQL Server Management Studio Wizard

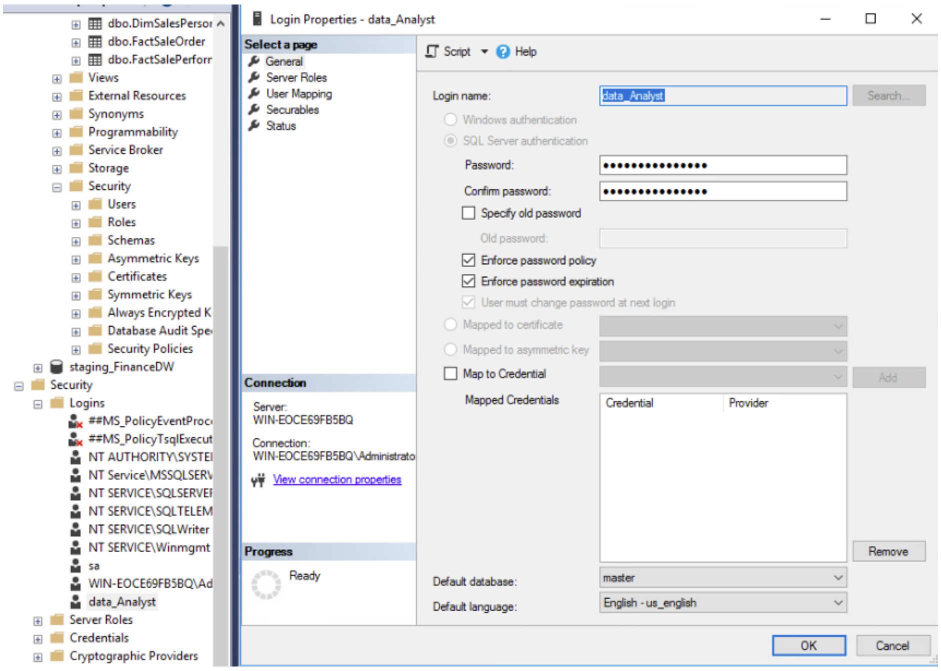
These methods have been used to create the necessary login, the users, and the permissions associated with those users. The processes have been recorded in Figure 8, Figure 9, Figure 10, and Figure 11.



```
-- >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

-- Create User Login and assign permissions for Data Warehouse Developer to read only access FinanceDB

use FinanceDB;

create login data_Warehouse_Developer with password = 'P@ssword1';

create user data_Warehouse_Developer for login data_Warehouse_Developer;

grant select on FinanceDB.dbo.Country to data_Warehouse_Developer;
grant select on FinanceDB.dbo.Product to data_Warehouse_Developer;
grant select on FinanceDB.dbo.ProductCost to data_Warehouse_Developer;
grant select on FinanceDB.dbo.Promotion to data_Warehouse_Developer;
grant select on FinanceDB.dbo.Region to data_Warehouse_Developer;
grant select on FinanceDB.dbo.SalesKPI to data_Warehouse_Developer;
grant select on FinanceDB.dbo.SalesOrder to data_Warehouse_Developer;
grant select on FinanceDB.dbo.SalesOrderLineItem to data_Warehouse_Developer;
grant select on FinanceDB.dbo.SalesPerson to data_Warehouse_Developer;
grant select on FinanceDB.dbo.SalesRegion to data_Warehouse_Developer;
grant select on FinanceDB.dbo.Segment to data_Warehouse_Developer;

-- Create User Login and assign permissions for Data Analyst Manager to read only access FinanceDW from PowerBI

use production_FinanceDW;

create login data_Analyst_Manager with password = 'P@ssword1';

create user data_Analyst_Manager for login data_Analyst_Manager;

grant select on DimDate to data_Analyst_Manager;
grant select on DimProduct to data_Analyst_Manager;
grant select on DimSalesLocation to data_Analyst_Manager;
grant select on DimSalesPerson to data_Analyst_Manager;
grant select on FactSaleOrder to data_Analyst_Manager;
grant select on FactSalePerformance to data_Analyst_Manager;
```

Figure 8 *Login, User, and Permissions assigned to Data Warehouse Developer on FinanceDB and Data Analyst on FinanceDW*

The following figures show the same process represented in Figure 8, a further data analyst having been created in the data warehouse using the built-in *Login Properties* and *Database User* wizards.



Login name: data_Analyst

○ Windows authentication

● SQL Server authentication

Password: •••••••••••••••

Confirm password: •••••••••••••••

□ Specify old password

Old password:

☑ Enforce password policy

☑ Enforce password expiration

☑ User must change password at next login

○ Mapped to certificate

○ Mapped to asymmetric key

□ Map to Credential

Mapped Credentials

Credential	Provider

Default database: master

Default language: English - us_english

Connection

Server:
WIN-EOCE69FB5BQ

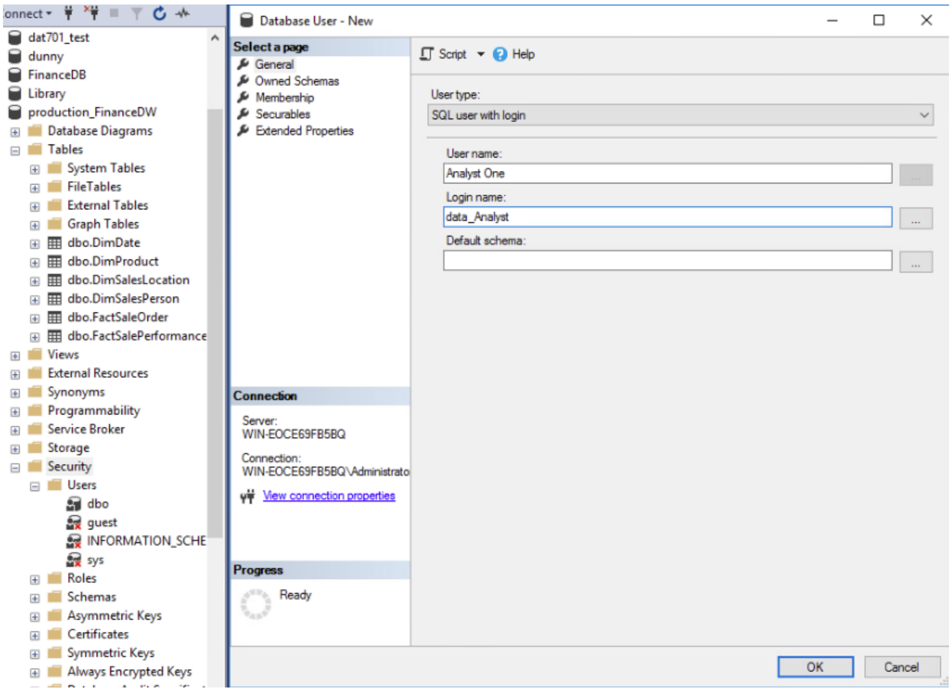
Connection:
WIN-EOCE69FB5BQ\Administrato

View connection properties

Progress

Ready

Figure 9 *Login Wizard*



User type:
SQL user with login

User name:
Analyst One

Login name:
data_Analyst

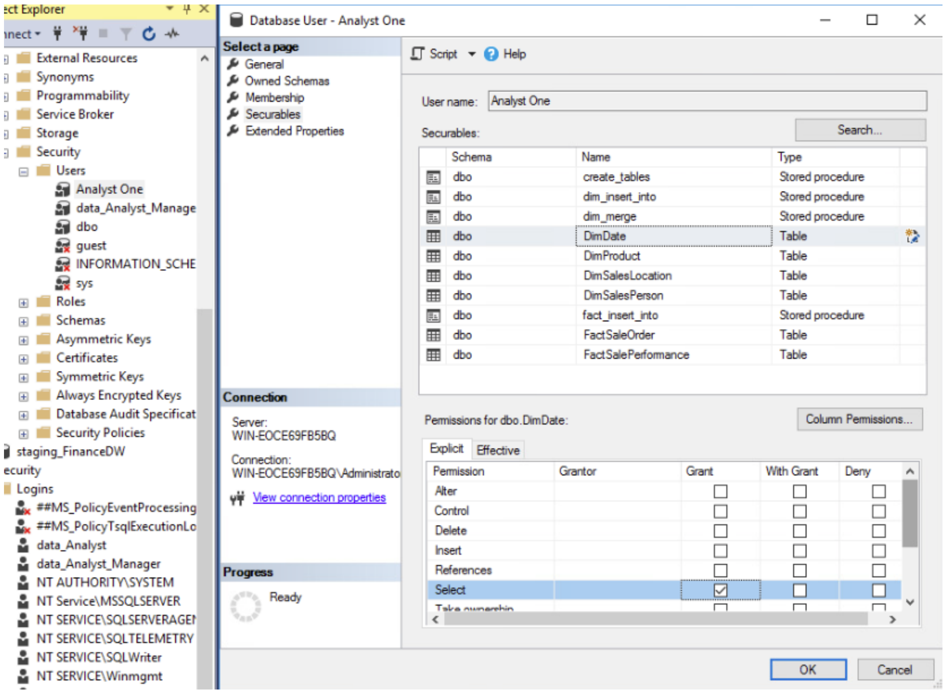
Default schema:

Server:
WIN-EOCE69FB5BQ

Connection:
WIN-EOCE69FB5BQ\Administrato

Ready

Figure 10 *User Wizard*



User name: Analyst One

Securables:

Schema	Name	Type
dbo	create_tables	Stored procedure
dbo	dim_insert_into	Stored procedure
dbo	dim_merge	Stored procedure
dbo	DimDate	Table
dbo	DimProduct	Table
dbo	DimSalesLocation	Table
dbo	DimSalesPerson	Table
dbo	fact_insert_into	Stored procedure
dbo	FactSaleOrder	Table
dbo	FactSalePerformance	Table

Permissions for dbo.DimDate:

Permission	Grantor	Grant	With Grant	Deny
Alter		□	□	□
Control		□	□	□
Delete		□	□	□
Insert		□	□	□
References		□	□	□
Select		☑	□	□
Take ownership		□	□	□

Server:
WIN-EOCE69FB5BQ

Connection:
WIN-EOCE69FB5BQ\Administrato

Ready

Figure 11 *Assign Permissions Wizard*

M3.4. Dimensions and Facts

ETL of the operational data has been performed through the use of two procedures and two views. The first procedure performs a complete bulk update of all the data ingested into the data warehouse as an initial ingestion of data, this is provided in the TALOS virtual machine, the GitHub repository listed in this report, and has been included in Appendix A. A further method has been created that is also available in the TALOS virtual machine, the GitHub repository identified in this report and has been included in Appendix B. This method deploys a variation of UPSERT in SQL Server called `MERGE`, which aims to update column records in the data warehouse for tuples that already exist in the fact or dimension tables, insert data where new information has been added to the OLTP database, or delete data that has been removed. UPSERT is an acronym for `UPDATE` and `INSERT` in T-SQL.

Merge is an operation that runs `INSERT`, `UPDATE`, or `DELETE` on a target table defined in the statement, in this case, FinanceDW, and joins it with the source database, FinanceDB. The merge statement effectively synchronises the two tables based on the differences that have been identified. This is accomplished through the use of the following terms:

- `when matched then` – when this is satisfied by the search condition the tuples are either updated or deleted. Merge statements can have two `when matched` clauses, with the second clause applied only if the first has not been applied.
- `when not match then` – specifies that the row will be inserted into the target table. Merge statements can only contain one `when not match` clause.

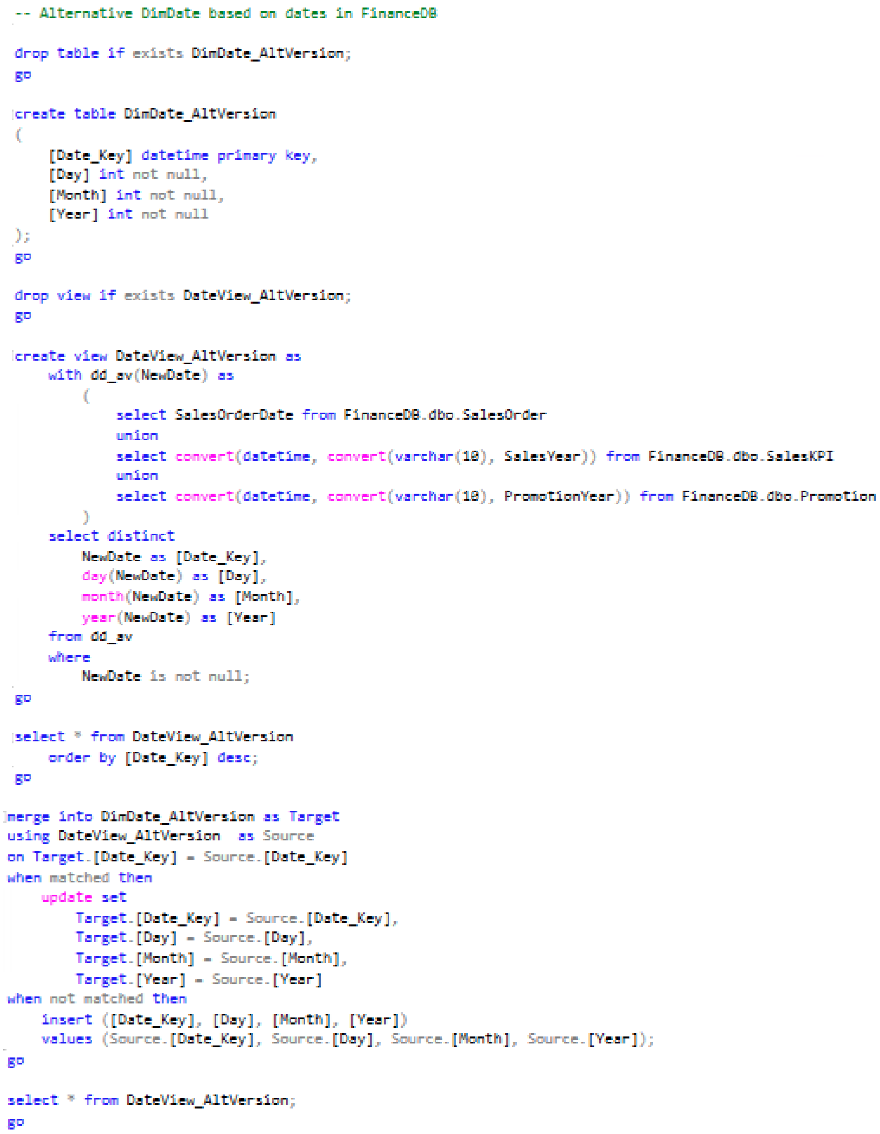
Common table expressions have been deployed for the dimension tables, acting as the temporary name for the result set, the merge statement references the common table expression naming structure. The fact tables are more complex due to the granularity of the measures and the relationships to the dimension tables, these require multiple common table expressions that are not capable of being referenced by the merge function. This required the fact table queries to be wrapped inside a view, these views can then be referenced by the merge statements to perform the UPSERT event.

To test the merge statements, data was changed in the FinanceDB database after the bulk insert procedure was run and the data set was successfully stored in the FinanceDW data warehouse. The merge statements showed that the changed data was updated as expected in the data warehouse. One such test has been provided in the TALOS virtual machine that can be tested and reviewed for this assignment, the statement is currently commented out for ease of performing the primary procedures as required.

M3.5. Data ETL for Dimensional Tables and Fact Tables

The two ETL procedures created for the FinanceDW data warehouse are wrapped in stored procedures that can be run in the TALOS virtual machine. The completed procedures including the views created for the merge statements can be found in Appendix A and Appendix B.

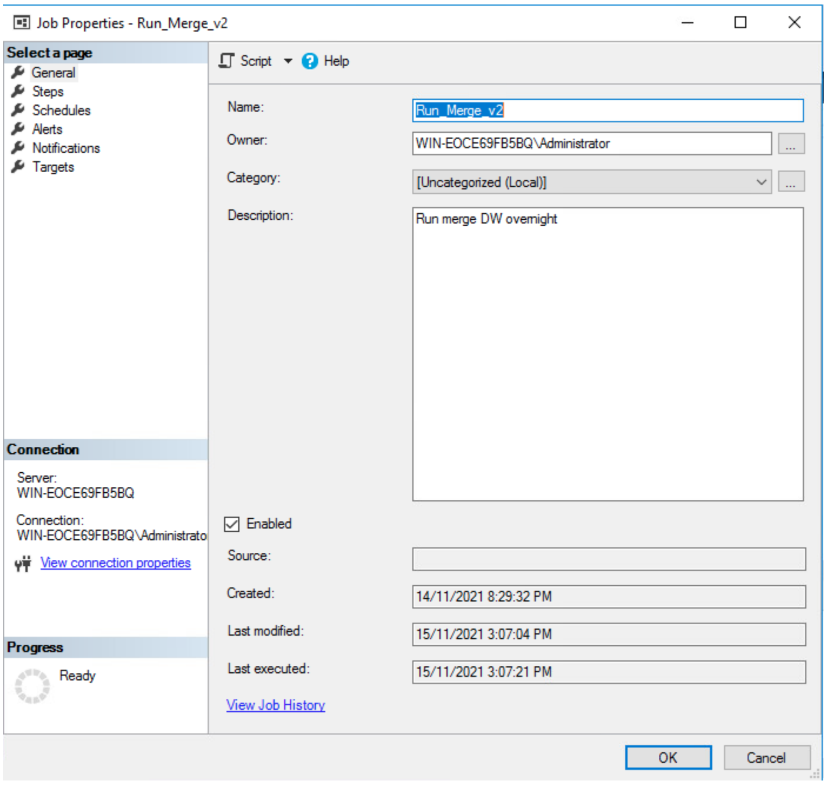
An alternative DimDate table has been considered and created that uses the dates contained across the tables in the FinanceDB database. The table takes the sales order date as the date key and includes the sales KPI year and promotion year as dates through the union select statement using the convert function. The day, month, and year have been extrapolated as integers from the collection of distinct dates. Figure 12 shows the process used to create this table.



```sql
-- Alternative DimDate based on dates in FinanceDB

drop table if exists DimDate_AltVersion;
go

create table DimDate_AltVersion
(
    [Date_Key] datetime primary key,
    [Day] int not null,
    [Month] int not null,
    [Year] int not null
);
go

drop view if exists DateView_AltVersion;
go

create view DateView_AltVersion as
    with dd_av(NewDate) as
        (
            select SalesOrderDate from FinanceDB.dbo.SalesOrder
            union
            select convert(datetime, convert(varchar(10), SalesYear)) from FinanceDB.dbo.SalesKPI
            union
            select convert(datetime, convert(varchar(10), PromotionYear)) from FinanceDB.dbo.Promotion
        )
    select distinct
        NewDate as [Date_Key],
        day(NewDate) as [Day],
        month(NewDate) as [Month],
        year(NewDate) as [Year]
    from dd_av
    where
        NewDate is not null;
go

select * from DateView_AltVersion
    order by [Date_Key] desc;
go

merge into DimDate_AltVersion as Target
using DateView_AltVersion  as Source
on Target.[Date_Key] = Source.[Date_Key]
when matched then
    update set
        Target.[Date_Key] = Source.[Date_Key],
        Target.[Day] = Source.[Day],
        Target.[Month] = Source.[Month],
        Target.[Year] = Source.[Year]
when not matched then
    insert ([Date_Key], [Day], [Month], [Year])
    values (Source.[Date_Key], Source.[Day], Source.[Month], Source.[Year]);
go

select * from DateView_AltVersion;
go
```

Figure 12 *Alternative DimDate solution explored for the FinanceDW data warehouse*

Research was conducted to find a solution that would allow the ETL merge procedure to be performed on a schedule when access is likely to be limited resulting in a low demand on system resources, such as overnight. SQL Server Agent was used to create a job that would run the procedure automatically, the process to establish the run merge procedure can be viewed in Figures 13, 14, 15, 16, and 17.



Name: Run_Merge_v2

Owner: WIN-EOCE69FB5BQ\Administrator

Category: [Uncategorized (Local)]

Description: Run merge DW overnight

☑ Enabled

Source:

Created: 14/11/2021 8:29:32 PM

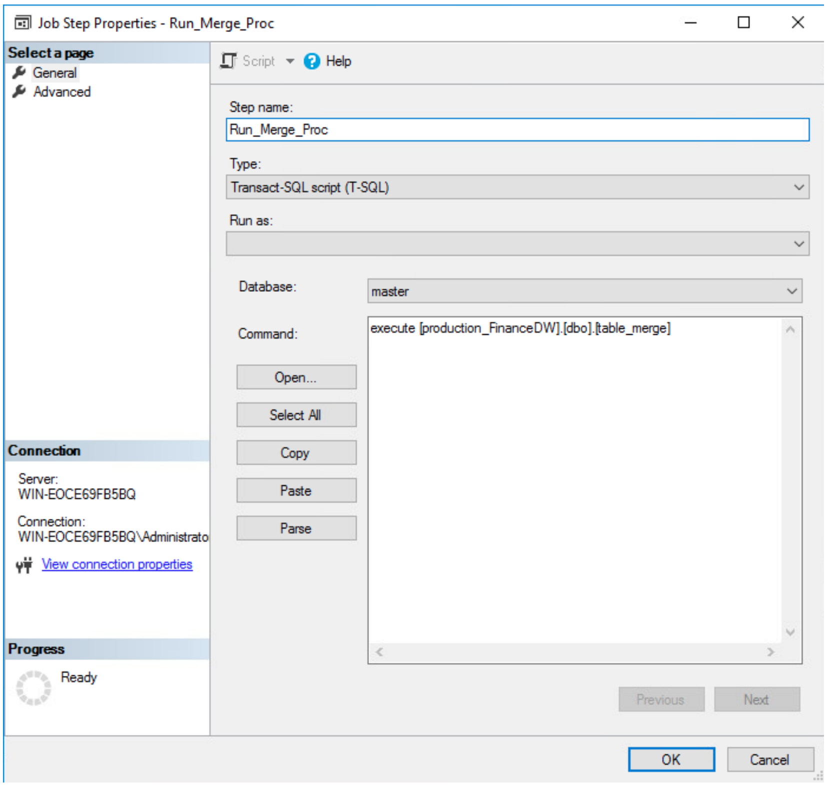
Last modified: 15/11/2021 3:07:04 PM

Last executed: 15/11/2021 3:07:21 PM

Server: WIN-EOCE69FB5BQ

Connection: WIN-EOCE69FB5BQ\Administrato

Figure 13 *Create new agent job to perform procedure automation*



Step name: Run_Merge_Proc

Type: Transact-SQL script (T-SQL)

Run as:

Database: master

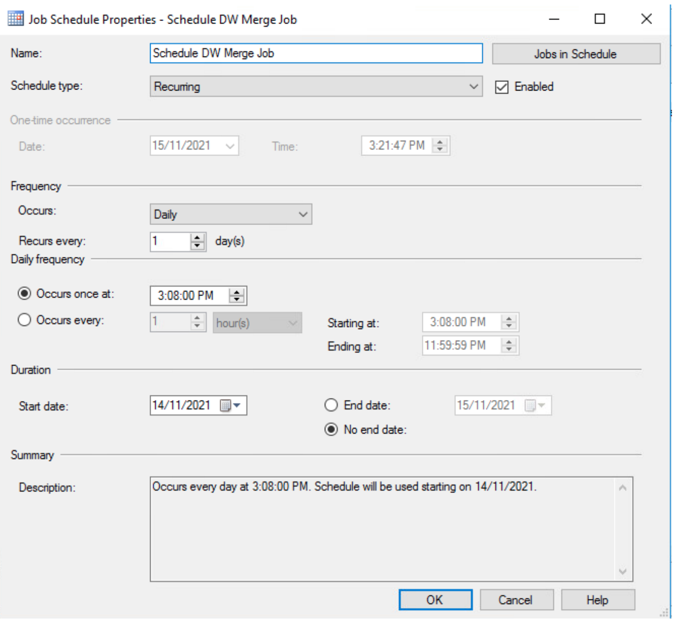
Command:

```
execute [production_FinanceDW].[dbo].[table_merge]
```

Server: WIN-EOCE69FB5BQ

Connection: WIN-EOCE69FB5BQ\Administrato

Figure 14 *Create T-SQL script to execute the procedure against the production_FinanceDW data warehouse*



Job Schedule Properties - Schedule DW Merge Job

Name: Schedule DW Merge Job

Schedule type: Recurring ☑ Enabled

One-time occurrence

Date: 15/11/2021 Time: 3:21:47 PM

Frequency

Occurs: Daily

Recurs every: 1 day(s)

Daily frequency

● Occurs once at: 3:08:00 PM

○ Occurs every: 1 hour(s) Starting at: 3:08:00 PM Ending at: 11:59:59 PM

Duration

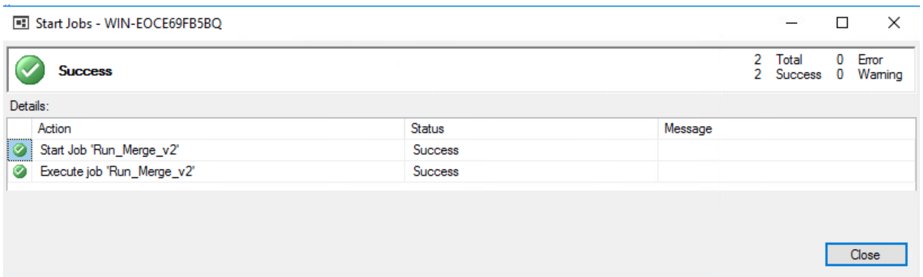
Start date: 14/11/2021 ○ End date: 15/11/2021

● No end date:

Summary

Description: Occurs every day at 3:08:00 PM. Schedule will be used starting on 14/11/2021.

Figure 15 *Assign a schedule to the agent job*



Start Jobs - WIN-EOCE69FB5BQ

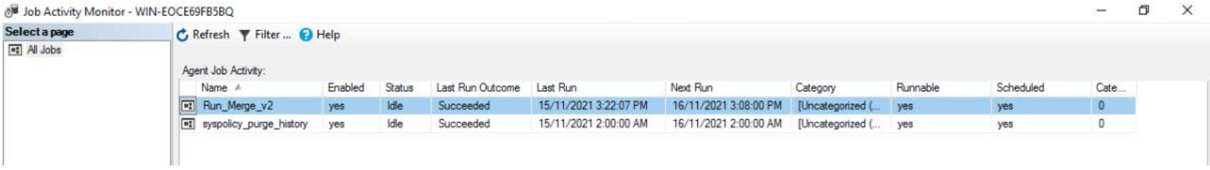
Success

2	Total	0	Error
2	Success	0	Warning

Details:

Action	Status	Message
✓ Start Job 'Run_Merge_v2'	Success	
✓ Execute job 'Run_Merge_v2'	Success	

Figure 16 *Run the job*



Job Activity Monitor - WIN-EOCE69FB5BQ

Agent Job Activity:

Name	Enabled	Status	Last Run Outcome	Last Run	Next Run	Category	Runnable	Scheduled	Cate...
Run_Merge_v2	yes	Idle	Succeeded	15/11/2021 3:22:07 PM	16/11/2021 3:08:00 PM	[Uncategorized (...	yes	yes	0
syspolicy_purge_history	yes	Idle	Succeeded	15/11/2021 2:00:00 AM	16/11/2021 2:00:00 AM	[Uncategorized (...	yes	yes	0

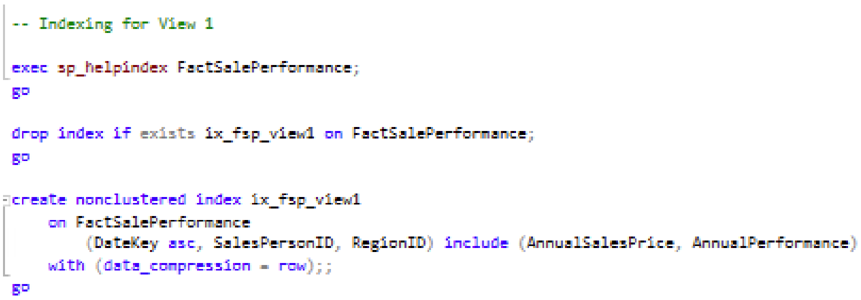
Figure 17 *Review the job logs to confirm the procedure was run successfully and the next run is scheduled*

Part Four: Indexing and Views

M4.1. Indexes

To support the views that have been created to answer the business questions, further indexes have been considered in addition to those created in the DDL. These non-clustered indexes are specifically designed to target columns required by the views.

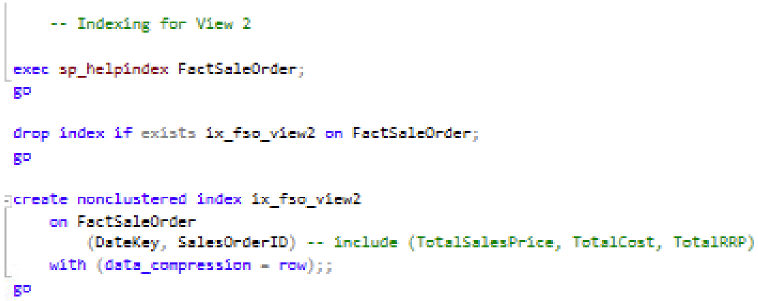
The view relating to the fact table *sales performance* is performed in zero seconds, meaning that addition non-clustered indexing is not necessary at this stage. However, to support scalability a non-clustered index was created on the table to support this view in recognition of business growth over time. The non-clustered index groups the *DateKey, SalesPersonID,* and *RegionID,* and uses `include` in the statement for the *AnnualSalesPrice* and *AnnualPerformance.* The index had no impact on the load time of the select statement. Should the records in the table grow over time, further testing should take place, particularly if the performance decreases.



```
-- Indexing for View 1

exec sp_helpindex FactSalePerformance;
go

drop index if exists ix_fsp_view1 on FactSalePerformance;
go

create nonclustered index ix_fsp_view1
    on FactSalePerformance
        (DateKey asc, SalesPersonID, RegionID) include (AnnualSalesPrice, AnnualPerformance)
    with (data_compression = row);;
go
```

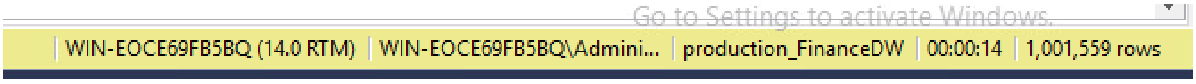
Figure 18 *Indexing created for view 1*

The view against the fact table *sale order* is more taxing on the system resources, requiring 20 seconds to complete the query and generating over 1 million records as observed in Figure 20. A non-clustered index was created in addition to the clustered and non-clustered index performed in the DDL, this grouped *DateKey* and *SalesOrderID,* and was tested with several columns in the index and utilising `include` for the columns. The results were less than impressive, with most attempts producing the same 14-second result, or increasing the load time to as much as 16-seconds.



```
    -- Indexing for View 2

exec sp_helpindex FactSaleOrder;
go

drop index if exists ix_fso_view2 on FactSaleOrder;
go

create nonclustered index ix_fso_view2
    on FactSaleOrder
        (DateKey, SalesOrderID) -- include (TotalSalesPrice, TotalCost, TotalRRP)
    with (data_compression = row);;
go
```

Figure 19 *Indexing created for view 2*



WIN-EOCE69FB5BQ (14.0 RTM) | WIN-EOCE69FB5BQ\Admini... | production_FinanceDW | 00:00:14 | 1,001,559 rows

Figure 20 *Execution time against view 2*

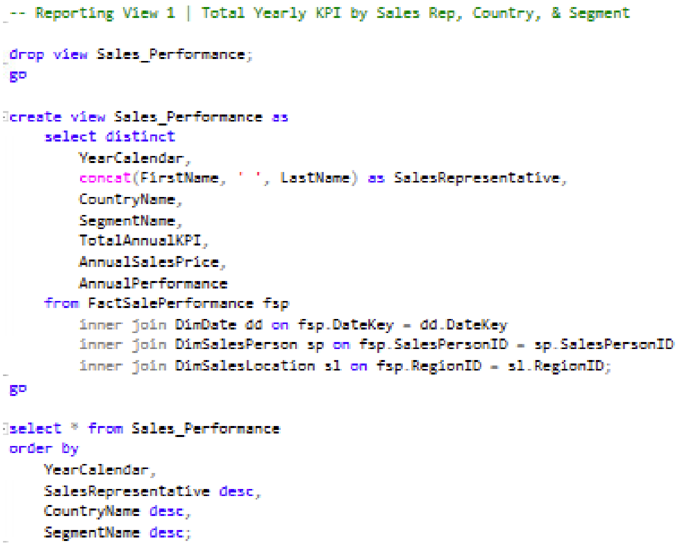
The results experienced in this test were not unexpected, given that the index strategy deployed in the DDL is sufficient given the size of the data set. The index strategy that has been deployed surrounds the creation of clustered indexes on the single-column primary keys in the dimension tables, which work in conjunction with the table partitioning meaning that given the tables are not considered very large by data warehousing standards, > 100k rows or even large, ~100k rows, a performance improvement has not been realised.

The fact tables can be classified as very large tables, therefore the strategy follows the recognised practice that includes the creation of a clustered index on the *DateKey*, and a non-clustered index on the relationship keys that act as foreign keys to the dimension tables, these have previously been defined in the DDL. Based on the testing of these earlier indexes, in conjunction with the table partitioning on the *DateKey*, indications are that these are sufficient to produce the query run times currently experienced.

M4.2. Views

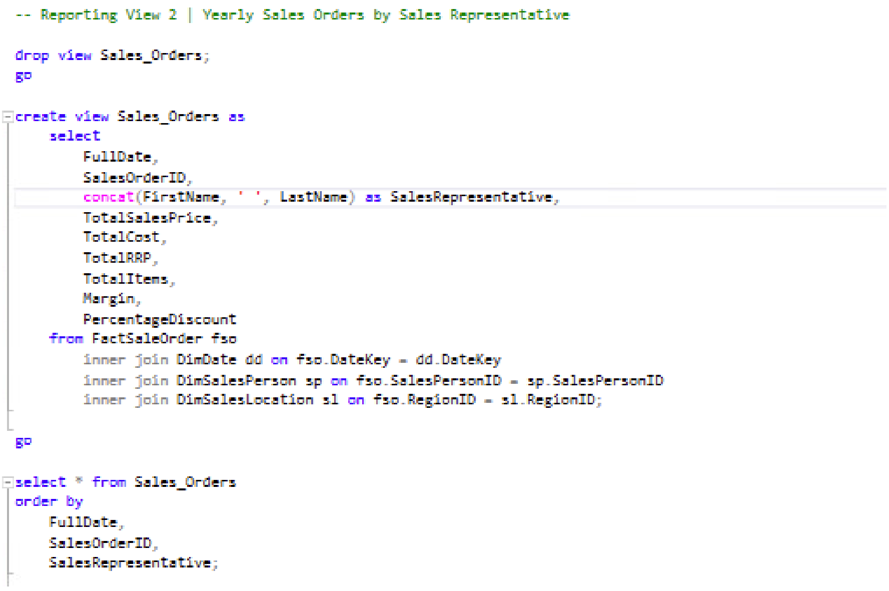
The two views created in the data warehouse have been specifically chosen to produce different information relating to separate business processes, the first view relating to sales performance and the second to sales orders. The view relating to sales orders contains all the necessary information to produce a dashboard in Power BI that satisfies the query produced in section C of Assignment 1.

These views will be used to produce data visualisations by the data analyst in Power BI. By restricting the data available to these views, the data warehouse developer can provide a greater level of security by restricting access to the other data contained in the tables.



```
-- Reporting View 1 | Total Yearly KPI by Sales Rep, Country, & Segment

drop view Sales_Performance;
go

create view Sales_Performance as
    select distinct
        YearCalendar,
        concat(FirstName, ' ', LastName) as SalesRepresentative,
        CountryName,
        SegmentName,
        TotalAnnualKPI,
        AnnualSalesPrice,
        AnnualPerformance
    from FactSalePerformance fsp
        inner join DimDate dd on fsp.DateKey = dd.DateKey
        inner join DimSalesPerson sp on fsp.SalesPersonID = sp.SalesPersonID
        inner join DimSalesLocation sl on fsp.RegionID = sl.RegionID;
go

select * from Sales_Performance
order by
    YearCalendar,
    SalesRepresentative desc,
    CountryName desc,
    SegmentName desc;
```

Figure 21 *View 1 for total yearly KPI by Sales Representative, Country, and Segment on Fact Sale Performance*



```
-- Reporting View 2 | Yearly Sales Orders by Sales Representative

drop view Sales_Orders;
go

create view Sales_Orders as
    select
        FullDate,
        SalesOrderID,
        concat(FirstName, ' ', LastName) as SalesRepresentative,
        TotalSalesPrice,
        TotalCost,
        TotalRRP,
        TotalItems,
        Margin,
        PercentageDiscount
    from FactSaleOrder fso
        inner join DimDate dd on fso.DateKey = dd.DateKey
        inner join DimSalesPerson sp on fso.SalesPersonID = sp.SalesPersonID
        inner join DimSalesLocation sl on fso.RegionID = sl.RegionID;

go

select * from Sales_Orders
order by
    FullDate,
    SalesOrderID,
    SalesRepresentative;
```

Figure 22 *View 2 for yearly sales orders on Fact Sale Order*

The execution plan was considered against both fact table views throughout the development process. A trade-off was necessary between the need to perform the aggregations across the tables in the OLTP database and the level of granularity required in the fact tables, with the desire to ensure optimum performance. The highest cost areas of the execution plan for the fact performance tables can be observed in Figure 23 and Figure 24. The highest cost areas of the execution plan for the fact performance tables can be observed in Figure 25 and Figure 26. The higher cost areas of the queries were expected, being with the aggregations and the inner joins. The queries and the common table expression have been designed to mitigate the effect on performance and run times.

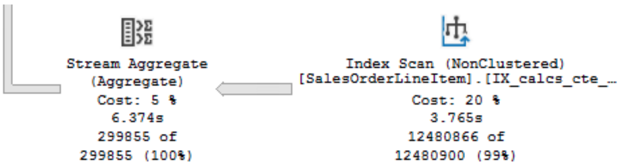


Figure 23 *Fact Sale Performance execution plan – Index scan and aggregation cost*

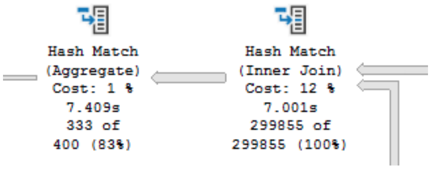


Figure 24 *Fact Sale Performance execution plan – Inner join cost*

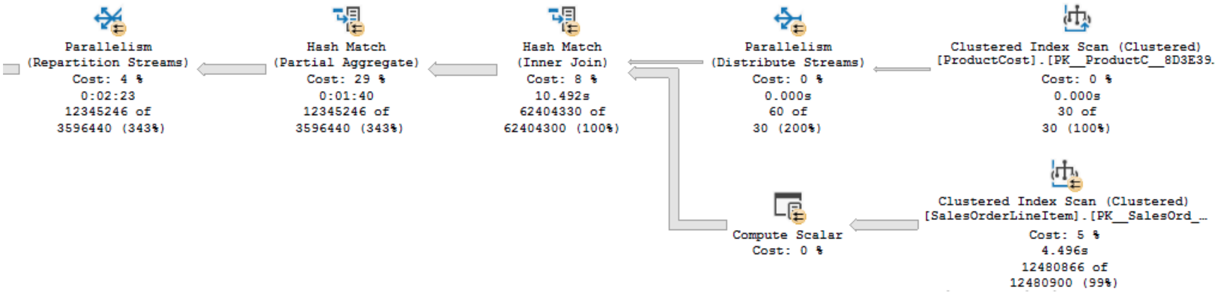


***Figure 25** Fact Sale Order execution plan – Clustered Index scan, inner join and aggregation hash match cost*

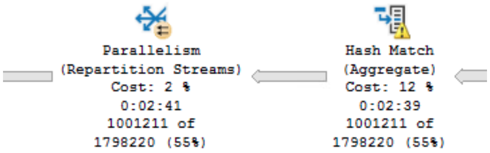
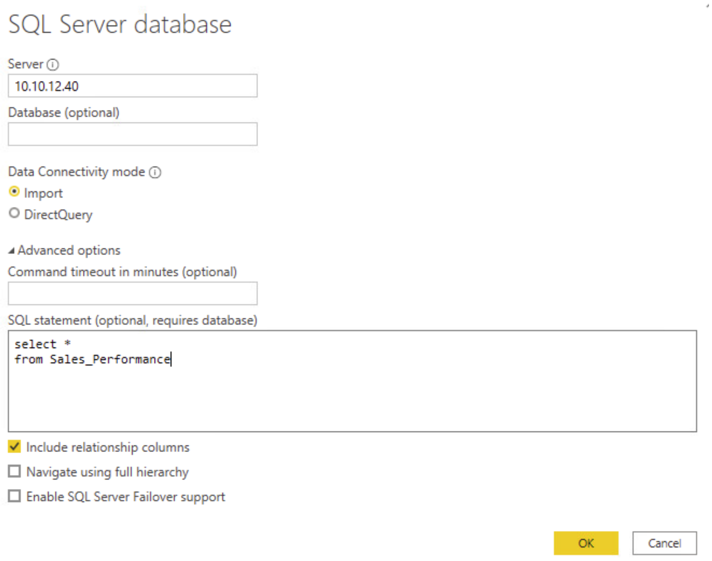


***Figure 26** Fact Sale Order execution plan – Aggregation hash match cost*

Milestone Five: PowerBI

M5.1. Power BI Dashboard

The dashboards in Power BI are constructed with the information from the two views created in the data warehouse, Figure 27 displays the set-up of these views during the connection between Power BI to the SQL Server instance. In the assignment documentation, this task would be split between the data warehouse developer and business analyst. The developer, having relevant permissions would create the view in SQL server, allowing the business analysts to make the connection in Power BI. Depending on the technical skill level of the business analyst, the data warehouse developer may need to make this connection on the business analysts behalf, before handing it over to the analyst to generate meaningful business insights.



SQL Server database

Server ⓘ
10.10.12.40

Database (optional)

Data Connectivity mode ⓘ
● Import
○ DirectQuery

◢ Advanced options
Command timeout in minutes (optional)

SQL statement (optional, requires database)

```
select *
from Sales_Performance
```

☑ Include relationship columns
☐ Navigate using full hierarchy
☐ Enable SQL Server Failover support

Figure 27 *Connection made to Sales_Performance view in FinanceDW*

The dashboards make use of charts, colours, and headings that allow a stakeholder to easily digest the information that has been made available. Best practices and standardisation have been deployed across all the dashboard views. High level and more important information are grouped in the top left of the dashboard, which is where the eye naturally looks. Colour is used to theme the dashboards and make the visualisations easier to view and understand. Headings are in grey so they can be used as a reference point, without distracting from the business information that is displayed that uses a prominent blue. If possible, an interview of stakeholders would be conducted to understand what information in the dashboard is of key importance, it may be the case that a particular region, segment, or sales team are under particular scrutiny or marked for anticipated growth, these key areas could be highlighted in blue, with the other information in various shades of grey so that the key areas of importance stand out and can be found at a glance.

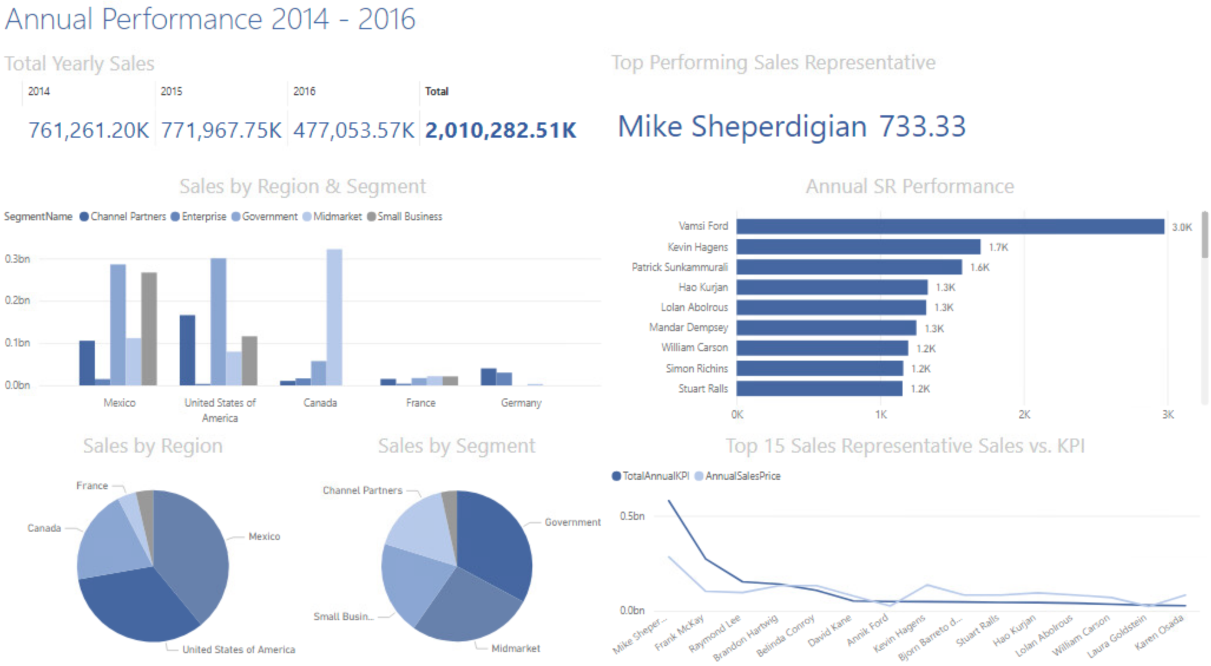
M5.2. Sales Performance against KPI

View 1 has been used to create the dashboard displayed in Figure 28, this relates to the sales performance section of Assignment 1 and is set to the year granularity level. A view could also be generated to provide monthly granularity, however, a determination has been made that this would not provide reliable results to business stakeholders given that the KPIs are set for sales representatives annually. This would lead to misleading figures, for example, if a monthly granularity level was produced it would be the *annual KPI divided by 12 months*, evenly split, with no consideration to monthly variances experienced by all businesses. For example, in the hospitality industry, revenue increases dramatically in the summer months and around Christmas time, but are far lower in the late autumn and new year, comparing this with an even split KPI would not only be unfair to the sales representatives but would be misleading for business stakeholders.

To create a valuable dashboard for business stakeholders given the quality of the data available, a comparative set of analysis visualisation has been produced. This includes total yearly sales for covering the current financial year, and the previous two financial years, which includes the top-performing sales representative as high-level business information. Additional charts are available that provide a finer grain, allowing stakeholders to view a greater level of detail, including the:

- grouping of sales by region and segment combined
- grouping of sales by region and segment as splits in pie chart format to easily view where sales are generated
- sales representative (SR) performance
- sales against KPIs by sales representative

Tooltips are available to perform a deeper dive into the detail in the charts, showing the colour key and additional information.



Annual Performance 2014 - 2016

Total Yearly Sales

2014	2015	2016	Total
761,261.20K	771,967.75K	477,053.57K	**2,010,282.51K**

Top Performing Sales Representative

Mike Sheperdigian 733.33

Figure 28 *Power BI dashboard relating to Sales Performance against KPI*

M5.3. Sales Orders

View 2 has been used to create the dashboard in Figure 29 relating to sales orders covering the current and preceding two financial years, with granularity to the month level. Figure 29 collates the annual information for a top-level view of the total sales and gross profit, items sold, and sales orders. The collated monthly figures cover each of the three financial years. The dashboard includes a filter that allows users to drill down the years to select an individual year. Tooltips are available to provide specific information on certain areas of the chart. Figure 30 uses the same view and relates to the most recent months total margin and total percentage.

The main purpose of the view is to provide the most valuable sales order data for stakeholders as *cards* across the top of the dashboard, these headline figures are easier to read and digest. The bold text differentiated with the colourway makes the figures stand out, this data can be viewed quickly and on the fly, the charts provide a more detailed overview of the company's orders where required.

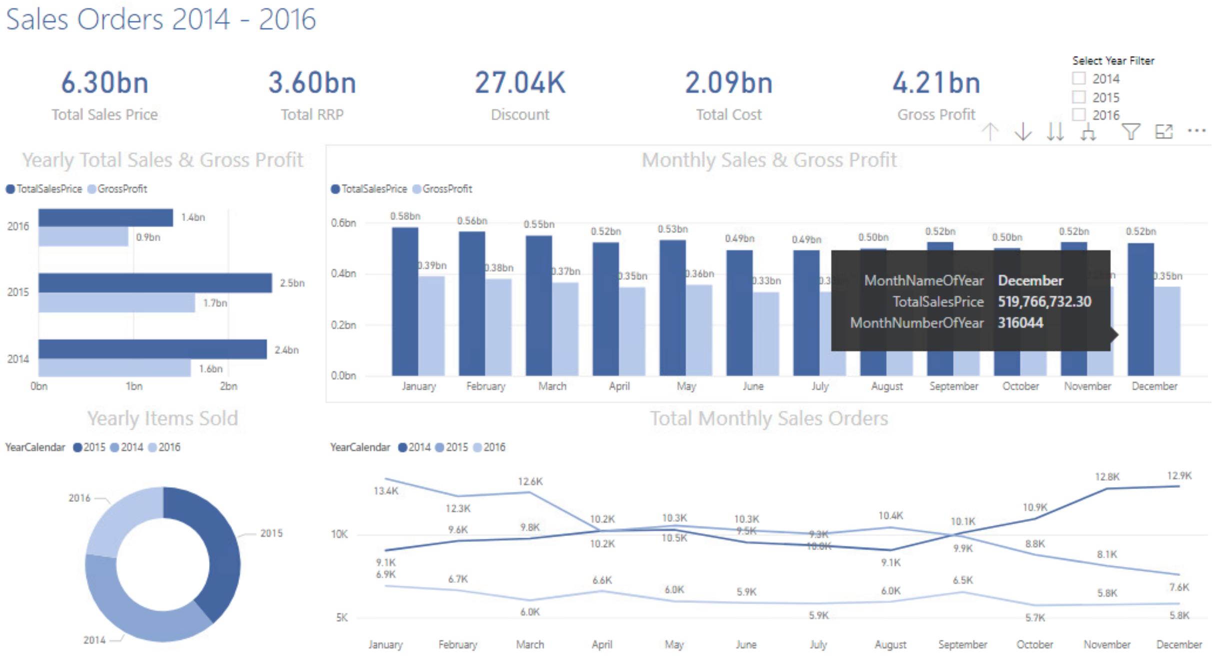
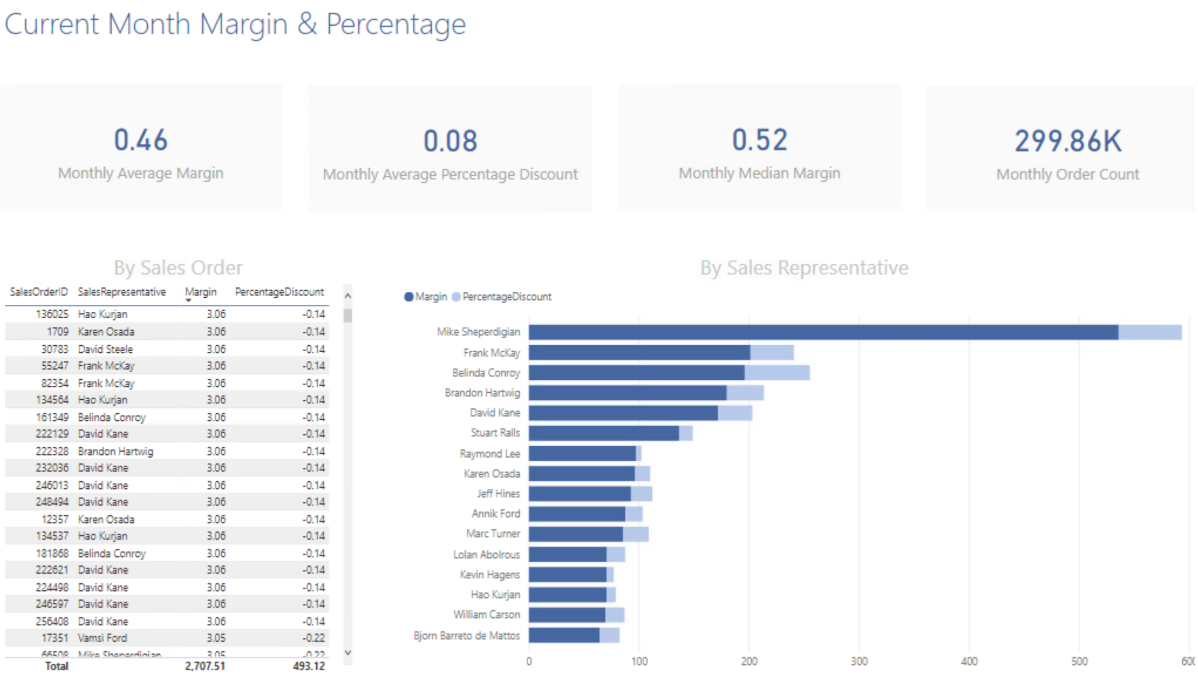


Figure 29 *Power BI dashboard relating to Sales Orders*

M5.4. Monthly Margin & Percentage

Figure 30 has been produced to meet the business question asked in section C of Assignment 1. The dashboard shows the average margin, percentage discount, and median margin for the most recent month in the data set, being December 2016. Figures are produced for high-level stakeholders that need quick and easy access to the data. Additionally, the dashboard provides a detailed list of sales orders for the month, the sales representative attached to the order, and the margin and percentage discount for that order. A more user-friendly bar chart has been provided showing the margin produced by sales representative, and the percentage discount applied for the month across sales orders generated.



Current Month Margin & Percentage

0.46
Monthly Average Margin

0.08
Monthly Average Percentage Discount

0.52
Monthly Median Margin

299.86K
Monthly Order Count

By Sales Order

SalesOrderID	SalesRepresentative	Margin	PercentageDiscount
136025	Hao Kurjan	3.06	-0.14
1709	Karen Osada	3.06	-0.14
30783	David Steele	3.06	-0.14
55247	Frank McKay	3.06	-0.14
82354	Frank McKay	3.06	-0.14
134564	Hao Kurjan	3.06	-0.14
161349	Belinda Conroy	3.06	-0.14
222129	David Kane	3.06	-0.14
222328	Brandon Hartwig	3.06	-0.14
232036	David Kane	3.06	-0.14
246013	David Kane	3.06	-0.14
248494	David Kane	3.06	-0.14
12357	Karen Osada	3.06	-0.14
134537	Hao Kurjan	3.06	-0.14
181868	Belinda Conroy	3.06	-0.14
222621	David Kane	3.06	-0.14
224498	David Kane	3.06	-0.14
246597	David Kane	3.06	-0.14
256408	David Kane	3.06	-0.14
17351	Vamsi Ford	3.05	-0.22
66508	Mike Sheperdigian	3.05	-0.22
Total		2,707.51	493.12

By Sales Representative

Figure 30 *Power BI dashboard relating to Margin & Percentage*

Milestone Six: Reflection

M6.1. What did you learn?

The most significant learning outcome has been the very different concept of OLAP data warehousing design and implementation, with a focus on data analytics and how this requires a paradigm shift in comparison to the deployment of OLTP databases, designed for operational use within an organisation. It was interesting to learn how the underlying technology need not change to produce a data warehouse environment, in instances where the existing data sources are integrated into the same system, in this case, SQL Server. Much of the change instead comes from the design of the dimensional schema and the attributes that make up the dimension and fact tables.

The paradigm that has been gained from an understanding of relational database design to third normal form and beyond, shifting to a design that breaks these rules which introduce duplication and redundancy to the data set took time to process. The benefit of processing speed to produce data analytics have, however, become obvious. The ETL process from relational to dimensional is a valuable solution to achieve response times required for business intelligence, without impacting the performance of the operational database. Observing the performance benefits through the query execution plan, in addition to the speed at which queries can be performed through the design changes is impressive when performing a direct back to back comparison with the OLTP database used in Assignment 1.

Table partitioning was a particularly interesting topic. The benefits, when combined with clustered and non-clustered indexing, have made querying against the data warehouse incredibly fast and efficient when compared with those same queries performed in the OLTP database. As a result of much of the querying being performed against dates, the most sensible approach was to apply partitioning on date columns and this was backed up by the research performed and referenced in this report.

M6.2. How did you apply it?

The dimensional model schema has been produced through an understanding of the business processes. These processes are specifically designed to cross-business department boundaries in an attempt to produce analytics that can serve the needs of the whole organisation. Visual paradigm was used to create the schema design, with tables created to ensure that the business questions from Assignment 1 can be answered from the new dimensional model after the completion of the ETL process.

Attributes in the dimension tables required little transformation in the ETL process, the DDL statement was designed to match the field types from the OLTP FinanceDB database. Fact tables required a significant amount more work and revision as a result of several factors, firstly, granularity caused some issues, and secondly, careful consideration of the design of the CTE statements and the joins within the select statements was needed to ensure that the figures were correct. Initially, a bulk insert procedure was created to test the CTE statements to ensure the correct outcome of the dataset, these could be queried against for testing and to ensure that the data warehouse could act as a single source of truth for the business.

Consideration was made to the future inserting and updating of data, and how this could be achieved without disrupting the existing data in the warehouse, UPSERT was researched and considered, with MERGE being deployed to perform inserts for new tuples identified in the OLTP database, whilst also recognising changes that might occur to existing data. This was tested against the product dimension table before being deployed across the data warehouse. An update was performed in the OLTP database on one of the products, once the change was made the merge statement was run and checked to ensure it identified this change and updated the attributes in the tuple. Again, more complexity was involved with the MERGE statements for the fact tables, requiring a view to be created that contained the multiple CTE statement necessary to produce an accurate data set. A MERGE statement is not capable of being run against a CTE statement containing multiple CTEs, necessitating the need for the views.

To ensure the security of the data warehouse and data source, read-only access was granted to the data warehouse developer against the OLTP database, likewise, read-only access was granted to the data analyst against the OLAP data warehouse. Two methods were used to perform the action, the built-in wizard, and the use of SQL statements. Using SQL statements is the preferred method being capable of acting as a record of the user and account creation, and the access granted. The record can be stored in a GitHub repository and version controlled, which can be used to support the documentation of the data warehouse process and data pipeline environment.

The Power BI dashboards were created from the viewpoint of a data analyst, it is often the case that these users have little or no experience with SQL. Power BI allows analytics to be produced from the data set as a whole, or views created by the data warehouse developer. Once the analyst has this information in Power BI, analytics can be produced quickly. However, it is important to note that features that add complexity to a dashboard, such as automation, can slow development progress. Power BI has an excellent user interface that is intuitive and easy to use, there is also a great deal of online community support that can assist with generating views and reviewing dashboard design standards.

M6.3. Other Technologies

Stitch

Stitch is a platform that performs the ETL function of a data pipeline. Stitch Import API is a REST API that is sent data that can be processed and pushed to a data warehouse. The import API accepts JSON or Transit and returns JSON for all methods using HTTP verb for example GET and POST. Data is processed using UPSERT ensuring that updates are captured plus any new data to be inserted, this prevents duplication occurring in the data set.

Stitch's extensibility allows organisations to connect to disparate SaaS data sources deployed across departments, sources include:

- Salesforce
- Trello
- Facebook
- MailChimp
- SQL Server

Stitch provide visibility and control over the data pipeline as data is transmitted from data sources to the destination. The service allows data engineers to implement data replication scheduling as often as required to meet specific needs, this can range from every minute to once a day. Further advanced scheduling can be configured including specific granular extraction start times, or the whitelisting of set hours to perform the extract.

Extraction logs and loading reports are produced that allow data engineers to monitor the replication process and quickly identify results that are identified as unexpected or incorrect. This provides a faster route to remedy problems that arise. Error handling is integrated into the service, Stitch detects any errors that arise in the pipeline and where possible will make any necessary corrections automatically. Where input is needed by a data engineer a notification is sent that details the issue. These notifications can be integrated with external applications such as Slack, Datadog, and PagerDuty.

Stitch markets the platform as a highly available infrastructure that allows businesses to process billions of records every day, whilst being flexible enough to scale up or down the volume as business requirements change. This is achievable as a result of the service level agreements that guarantee system uptime and support response times, with no need for a business to concern itself with hardware provision of workload management.

The transformation and data quality solution conjunction with the performance attributes allow data teams to deliver data analytics in near real-time. Transformation can be performed through a vast array of connectors and components such as sort, aggregate, join, map, and more. Data quality can be improved on any size of format, including unstructured data through parsing technology integrated into the service.

Consideration is made to the privacy and security of the data transmitted during the ETL process. Sensitive data can be masked to protect the data privacy of individuals, supporting organisational efforts to meet regulatory frameworks. This is further supported by Stitch's compliance with the EU's GDPR, including independent audit accreditation against SOC-2 security, availability and confidentiality principles. The security features provided by Stitch include:

- HSTS encrypted communication for web browsers
- Configurable minimum permissions that allow read-only access to necessary data, or subsets of data through replication
- SSH tunnelling, SSL/TLS, and IP whitelisting for secure connections to data sources
- Direct access to logs from data source integrations for auditing purposes
- Data retention only as long as necessary to ensure successful load to destination sources

Stitch produces a range of open-source documentation that supports users when learning to deploy the system. Available documentation ranges from destination sources such as data lakes, data warehouses, and storage platforms, integration with databases and SaaS applications, data extract and replication settings and loading and structuring data.

DBT

dbt operates within the data warehouse or data lake environment to transform data sets to production-grade data pipelines in SQL, capable of providing data visualisations to business users. SQL models with SELECT statements are deployed to impose order on diverse data sources, handle the chore of dependency and build dimensional model tables and data views. dbt performs data transformation, it does not perform extract or load functions within the data pipeline.

Data modelling in dbt allows meaningful insights that are valuable to stakeholders can be created more easily and faster. The use of SQL means there is no requirement for analysts to learn new tools or coding languages which aid collaboration and understanding of the transformed data. dbt can be deployed using dev environments with version control in GitHub, further enabling collaboration and allowing data engineers to return to previous states. The environment architecture allows models to be tested before production.

SQL queries can be run to perform database management tasks, these include:

- Creation of user-defined functions
- Granting table privileges

Hooks and operations are used in dbt to execute these tasks. Hooks are snippets of SQL that can be executed at different times, these include:

- `pre-hook`: executed before a model, seed or snapshot is built.
- `post-hook`: executed after a model, seed or snapshot is built.
- `on-run-start`: executed at the start of dbt run, dbt seed or dbt snapshot
- `on-run-end`: executed at the end of dbt run, dbt seed or dbt snapshot

By way of example, on-end-run hooks can be used to grant usage on a target schema to a specific role, likewise, post-hook can grant `select` on models to a specific role.

Operations allow for macros to be invoked without the need to run a model and are triggered in the CLI using the dbt run operation command. Where operations differ from hooks is the need to explicitly execute the query within a macro. Macros can perform similar actions to webhooks, for example granting schema usage to a role or granting `select` to a table.

Data Warehousing in the Cloud

A cloud approach allows companies to benefit from efficient solutions and the latest innovative technologies without high capital expenditure, allowing the cost can be moved to operational expenditure. A comparison between on-premises data warehousing solutions and cloud-based solutions highlighting the benefits of cloud adoption are outlined in Table 3.

On-Premises Solution	Cloud Solution
Longer to set up and deploy	Significantly faster deployment time
High cost of storage and computing	More affordable, as much as one-tenth of the cost
Limited flexibility, balancing, and tuning	Elastic, flexible, and more scalable
Possibility of delays and downtime	Almost no delays or downtime
Higher costs relating to security and discovery	Comparatively, far lower costs

Table 3 *Comparative Analysis between On-Premises and Cloud Data Warehouse Solutions*

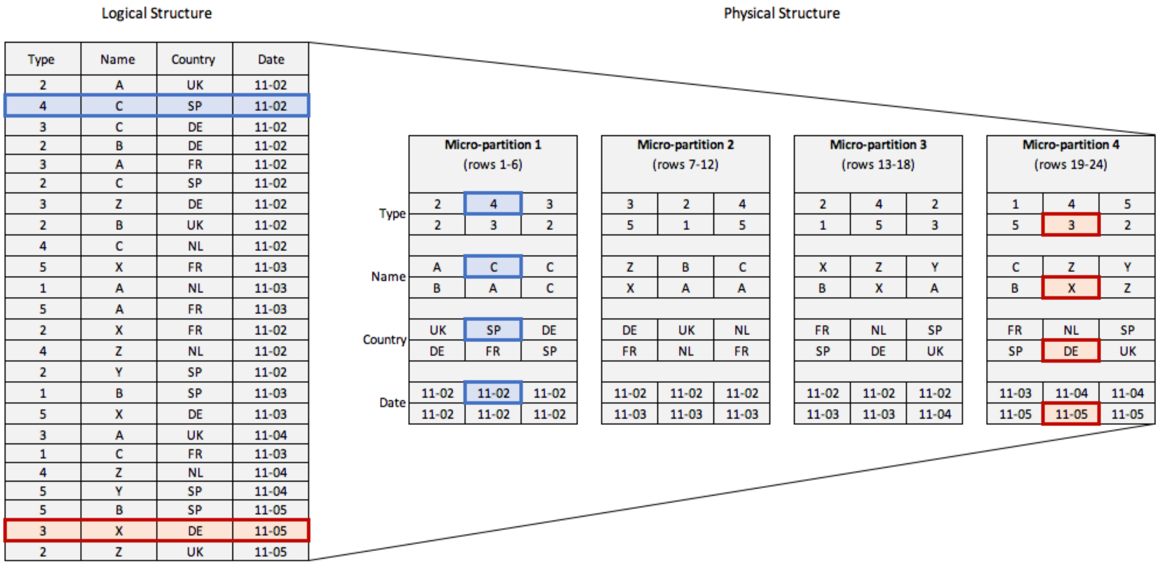
In any organisation, time is money, reducing the implementation time through the deployment of a cloud solution makes it an attractive option, whilst limiting exposure to market downturns without access to the data necessary to adapt. The reduced planning and implementation time of a cloud data warehouse can lead to it quickly becoming a crucial supporting technology. The speed of markets and the necessity for companies to keep up drives them to limit their exposure to capital expenditure that is supported through the use of cloud solutions. In contrast, on-premises solutions require servers, storage devices, high-speed network implementations, licensing fees, and increased staffing levels to be budgeted, in addition to forecasted events such as upgrades, ongoing maintenance, security protocols, firewall protection, and the monitoring and identification of vulnerabilities and threats. The advantages of cloud data warehouses should also be considered from a technical perspective. For instance, cloud providers are better positioned to handle peak usage, can offer seemingly unlimited storage and compute, and provide scalability to meet workload demands without affecting system performance. Cloud data warehouse architectures are, therefore, capable of producing more effective data pipelines and efficient query run times.

Snowflake

Snowflake allows the organisation to bring together users, data and workloads into a single solution deployed completely on cloud infrastructure, breaking down the technological barriers experienced with other solutions and supports both Amazon Web Services (AWS) and Microsoft Azure, taking advantage of both platforms. In addition to the infrastructure advantages, users benefit from Snowflake's use of SQL. Furthermore, Snowflake supports common data formats including JSON, Parquet, and XML, creating a single source data warehouse from petabytes of both structured and unstructured data. The separation of computing and storage allows for more flexible scalability, which is necessary to quickly scale up or down. System management is performed by the data engineers in a separate application side user experience to that of the data analysts extracting business insights. This process separation supports the management of concurrent bottlenecks during periods of high demand.

The benefit of Snowflake's Data Platform is its use of micro-partitioning to automatically divide tables into small contiguous units of uncompressed data storage. Snowflake automatically determines a compression algorithm that offers the most efficient method to reduce the actual data size. Rows are mapped into individual micro-partitions that allows for extremely granular pruning of large tables. In contrast, traditional data warehouses rely on static partitioning that increases maintenance overhead and data skew, which results in disproportionately sized partitions. Micro-partitioning has a beneficial impact on performing SQL Data Manipulation Language (DML) operations such as deleting rows from a table. The underlying micro-partition metadata facilities and simplifies the maintenance of tables, making them metadata only operations. The metadata maintained by Snowflake enables precise column pruning in micro-partitions, including columns that store semi-structured data. For example, a table containing historical and uniformed data would allow a query to target and scan only those micro-partitions that meet the filter predicate on the range of defined values, then scan the micro-partitions that contain the data. Snowflake uses columnar scanning that significantly improves efficiency and performance, if the query filter is by one column, then only the filtered column is scanned.

Clustering is another important factor in query performance. To improve efficiency, Snowflake orders data when it is stored in tables. Snowflake collects clustering metadata and records it for each micro-partition when data is loaded. The metadata is leveraged during query runs to avoid scanning irrelevant micro-partitions, offering accelerated performance. Clustering metadata is maintained by Snowflake and includes the total number of micro-partitions for a table, the total number that contain overlapping values with one another, and the depth of the overlapping. Clustering depth is an indicator of clustering health, which can diminish over time as a result of DML being performed on the table. Figure 31 illustrates the data clustering of a table consisting of 24 rows, stored by columns, equally across 4 micro-partitions. Snowflake can prune the micro-partitions that are not necessary for the query, and then prune the remaining partitions by column. If tables become very large and the data is not suitably ordered, query performance can become noticeably diminished, clustering keys can also be defined to offer overall query improvement.



Logical Structure

Type	Name	Country	Date
2	A	UK	11-02
4	C	SP	11-02
3	C	DE	11-02
2	B	DE	11-02
3	A	FR	11-02
2	C	SP	11-02
3	Z	DE	11-02
2	B	UK	11-02
4	C	NL	11-02
5	X	FR	11-03
1	A	NL	11-03
5	A	FR	11-03
2	X	FR	11-02
4	Z	NL	11-02
2	Y	SP	11-02
1	B	SP	11-03
5	X	DE	11-03
3	A	UK	11-04
1	C	FR	11-03
4	Z	NL	11-04
5	Y	SP	11-04
5	B	SP	11-05
3	X	DE	11-05
2	Z	UK	11-05

Physical Structure

	Micro-partition 1 (rows 1-6)			Micro-partition 2 (rows 7-12)			Micro-partition 3 (rows 13-18)			Micro-partition 4 (rows 19-24)		
Type	2	4	3	3	2	4	2	4	2	1	4	5
	2	3	2	5	1	5	1	5	3	5	3	2
Name	A	C	C	Z	B	C	X	Z	Y	C	Z	Y
	B	A	C	X	A	A	B	X	A	B	X	Z
Country	UK	SP	DE	DE	UK	NL	FR	NL	SP	FR	NL	SP
	DE	FR	SP	FR	NL	FR	SP	DE	UK	SP	DE	UK
Date	11-02	11-02	11-02	11-02	11-02	11-02	11-02	11-02	11-02	11-03	11-04	11-04
	11-02	11-02	11-02	11-03	11-03	11-03	11-03	11-03	11-04	11-05	11-05	11-05

Figure 31 *Logical and physical representation of micro-clustering in Snowflake data warehousing*

Clustering keys are explicitly designated table columns that can be leveraged to co-locate data in the table within the same micro-partitions, this can also be useful if the clustering depth is large. Clustering keys offer efficiency improvements when scanning by avoiding data that does not match filtering predicates. Furthermore, clustering keys provide improved table compression and require no additional maintenance by users once deployed, instead Snowflake automatically ensure optimal clustering performance. It is important to consider the use of clustering keys carefully as they will consume higher compute resources meaning they should only be created where there is a substantial benefit. Common SQL where queries sort on the table clustering key with an order by, group by, and some join operations. Figure 32 shows a proposed data pipeline solution.

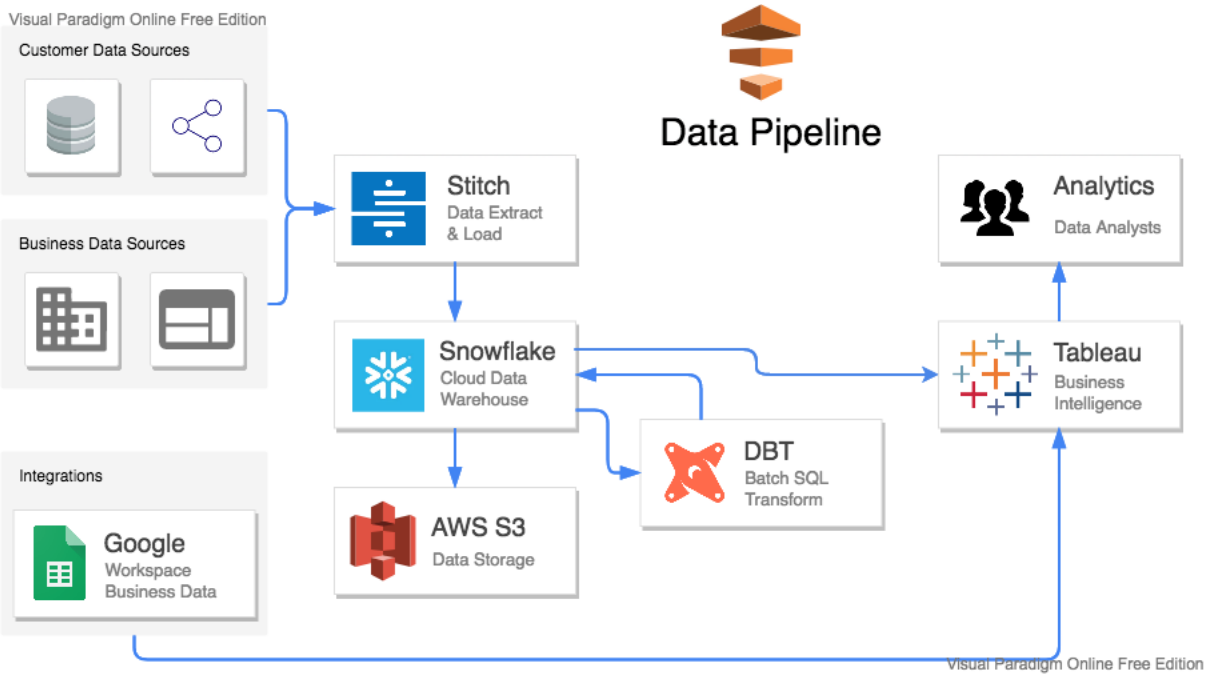


Figure 32 *Proposal – data pipeline solution*

Tableau

A variety of data analytic tools have been tested in addition to Power BI, included Tableau, Metabase, and Qlik. A preliminary comparative analysis was conducted of these four industry-leading tools followed by extensive testing. Research of the services included an investigation that included the following metrics:

- Ease of integration with company deployed SQL databases and applications
- Set-up time and time to deployment
- The ability to create meaningful data using SQL and replication using the GUI environment
- Automation of charts and figures that reference monthly, quarterly, and annual results
- Alerts and notifications for triggers identified as important to business stakeholders
- Dashboard and result visualisation, both within a secure environment and for publication online through a web interface, is compatible with mobile devices, large office monitors, and for insertion into presentation documents
- Quantity and quality of the features available, particularly surrounding the ability to visualise data for non-technical business users
- Consideration to cost that includes user profiling for license requirements
- Security and privacy features with specific consideration to GDPR and CCPA
- Company branding, including colour schemes and fonts for organisational consistency

The research conducted provided a sufficient overview of the offerings that narrowed the list for testing, which began with the removal of Power BI. This was a technical decision aimed at limiting the exposure to Microsoft products as a result of the corporation's perceived reputation for forcing users to stay within their technology ecosystem. Testing was conducted on Tableau, Metabase, and Qlik.

Several factors became obvious as a result of the systems testing. Firstly, the two cloud-based solutions, Metabase and Qlik, were simpler to set up and integrate with the wider technology infrastructure. Tableau, in comparison, required the installation of a desktop application followed by an ODBC driver. The solution will require installation support for non-technical users, which, when combined with a system that is stored locally, reduces the desirability of the product that effectively increases the attack surface area from a security perspective. Despite the drawbacks, Tableau does allow for the creation of data visualisations in the cloud. However, many key features are only available in the desktop instance, with the dashboard being uploaded to the cloud. Secondly, Tableau benefits from a GUI environment that is similar to Power BI, being user friendly and simple to operate for non-technical users. Thirdly, with Tableau and Metabase, visualisations can be created with SQL queries, meaning charts to be developed more quickly by those skilled in the language.

Metabase and Qlik take very different approaches to Tableau and Power BI. Metabase have an interface with reduced complexity and feature clutter. Metabase deploys SQL structure as part of the user interface in the form of *tick boxes* that run queries against the data, despite this integrated GUI feature, knowledge of SQL would still be required, making it redundant. Ultimately, the simple design proved to be a result of missing features that were desired by the team, limiting the ability to future proof the data pipeline. Qlik proved to be a different approach to data analytics, deploying an *associative engine* that seeks to do the heavy lifting through the deployment of machine learning to understand the data and automatically generate data visuals, requiring little or no input from the user. Qlik is an impressive application with a unique pricing structure that is based on the compute time of the users actively engaged with the product.

Figure 32 is a representation of a dashboard created in Tableau.

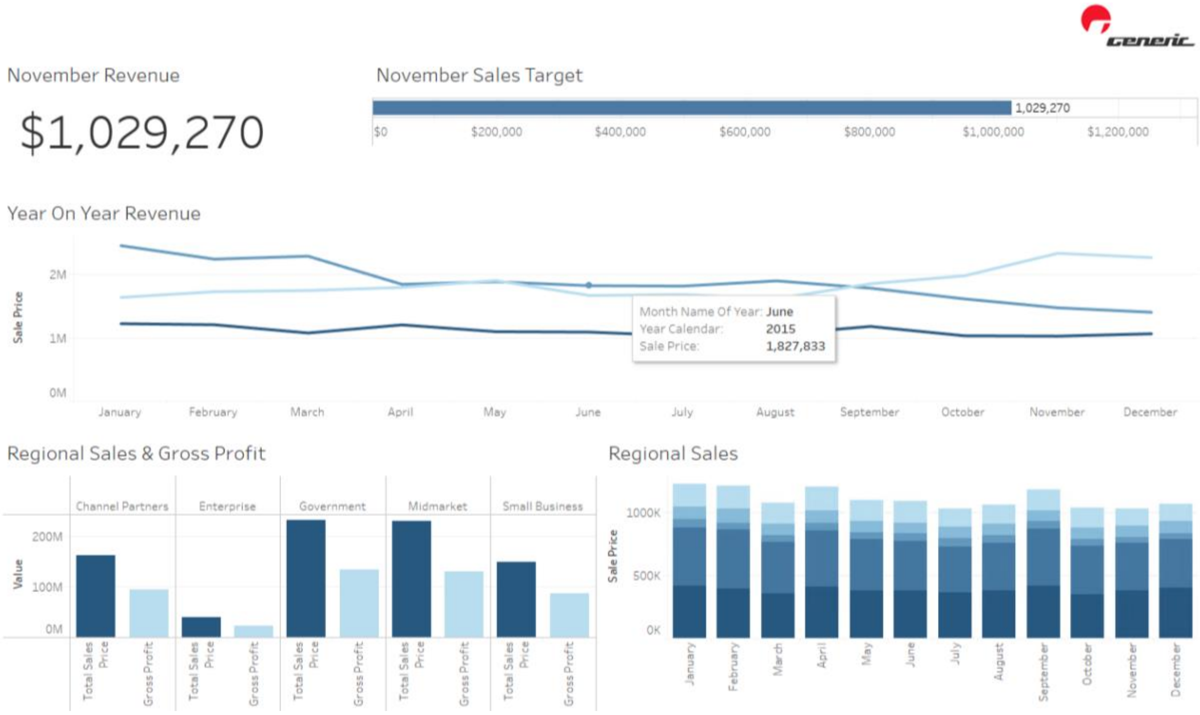


Figure 32 *Revenue dashboard created in Tableau using open-source data from a Snowflake data warehouse*

Files

The SQL file is available directly on the virtual machine's desktop labelled as ‘Assignment_2.sql’. The SQL file can also be found on the author's GitHub repository named DAT701 using the following link: https://github.com/d-stephenson/DAT701

References

Blaha, M. (2016, June 15). Data Warehouses Should Stage Source Data. *DATAVERSITY*. https://www.dataversity.net/data-warehouses-stage-source-data/

cawrites. (n.d.-a). *Recovery Models (SQL Server)—SQL Server*. Retrieved October 22, 2021, from https://docs.microsoft.com/en-us/sql/relational-databases/backup-restore/recovery-models-sql-server

cawrites. (n.d.-b). *Set database recovery model—SQL Server*. Retrieved October 22, 2021, from https://docs.microsoft.com/en-us/sql/relational-databases/backup-restore/view-or-change-the-recovery-model-of-a-database-sql-server

Convert datetime to integer. (n.d.). Retrieved October 17, 2021, from https://social.msdn.microsoft.com/Forums/en-US/04f2098e-9802-4fea-a936-3f988981f8ee/convert-datetime-to-integer?forum=transactsql

Converting dates to integer. (n.d.). Kimballgroup.Forumotion.Net. Retrieved October 17, 2021, from https://kimballgroup.forumotion.net/t720-converting-dates-to-integer

Create a Table Partition in SQL Server 2012—Concurrency. (n.d.). Retrieved October 22, 2021, from https://www.concurrency.com/blog/december-2016/create-a-table-partition-in-sql-server-2012

Data Transformation and Data Quality. (n.d.). Stitch. Retrieved November 10, 2021, from https://www.stitchdata.com/platform/datatransformation/

data warehouse—Loading Fact Tables with SQL Server. (n.d.). Stack Overflow. Retrieved October 17, 2021, from https://stackoverflow.com/questions/10053002/loading-fact-tables-with-sql-server

Dbt Explained | Blog. (2021, July 12). Fivetran. https://fivetran.com/blog/dbt-explained

Dbt for Data Transformation – Hands-on Tutorial. (n.d.). *KDnuggets*. Retrieved November 10, 2021, from https://www.kdnuggets.com/dbt-for-data-transformation-hands-on-tutorial.html/

Execute Stored Proc Using SQL Job. (n.d.). Retrieved November 14, 2021, from https://www.c-sharpcorner.com/article/execute-stored-proc-using-sql-job/

Hooks & Operations | dbt Docs. (n.d.). Retrieved November 10, 2021, from https://docs.getdbt.com/docs/building-a-dbt-project/hooks-operations

How to Partition an existing SQL Server Table. (n.d.). Retrieved October 22, 2021, from https://www.mssqltips.com/sqlservertip/2888/how-to-partition-an-existing-sql-server-table/

How to Run a SQL Server UPSERT using Merge. (2020, August 14). *HackDeploy*. https://hackdeploy.com/how-to-run-a-sql-server-upsert-using-merge/

IBM Docs. (2021a, March 8). https://prod.ibmdocs-production-dal-6099123ce774e592a519d7c33db8265e-0000.us-south.containers.appdomain.cloud/docs/en/ida/9.1?topic=phase-step-identify-business-process-requirements

IBM Docs. (2021b, March 8). https://prod.ibmdocs-production-dal-6099123ce774e592a519d7c33db8265e-0000.us-south.containers.appdomain.cloud/docs/en/ida/9.1?topic=phase-step-identify-business-process-requirements

maggiesMSFT. (n.d.). *Create a Power BI dashboard from a report—Power BI*. Retrieved November 6, 2021, from https://docs.microsoft.com/en-us/power-bi/create-reports/service-dashboard-create

markingmyname. (n.d.). *sp_procoption (Transact-SQL)—SQL Server*. Retrieved November 14, 2021, from https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/sp-procoption-transact-sql

Moody, D. L., & Kortink, M. A. R. (2000). *From Enterprise Models to Dimensional Models: A Methodology for Data Warehouse and Data Mart Design*. *Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW'2000)*.

MS SQL Server—How to get Date only from the datetime value? (2018, September 1). TablePlus. https://tableplus.com/blog/2018/09/ms-sql-server-how-to-get-date-only-from-datetime-value.html

mssaperla. (n.d.). *Best practices: Delta Lake - Azure Databricks*. Retrieved October 22, 2021, from https://docs.microsoft.com/en-us/azure/databricks/delta/best-practices

Multidimensional Warehouse (MDW). (n.d.). [Pstopic]. Retrieved October 17, 2021, from https://docs.oracle.com/cd/E41507_01/epm91pbr3/eng/epm/penw/concept_MultidimensionalWarehouseMDW-9912e0.html

Peterson, R. (2020, March 16). *How to Create Login, User & Assign Permissions in SQL Server*. https://www.guru99.com/sql-server-create-user.html

Populating Fact Tables. (2015, February 9). *Data Warehousing and Machine Learning*. https://dwbi1.wordpress.com/2015/02/09/populating-fact-tables/

Snowflake Data Cloud | Enable the Most Critical Workloads. (n.d.). Snowflake. Retrieved November 10, 2021, from https://www.snowflake.com/

Sql server—How to automatically run a stored procedure on scheduler basis? - Stack Overflow. (n.d.). Retrieved November 14, 2021, from https://stackoverflow.com/questions/12158158/how-to-automatically-run-a-stored-procedure-on-scheduler-basis

sql—MySQL - How to join two subqueries. (n.d.). Stack Overflow. Retrieved October 14, 2021, from https://stackoverflow.com/questions/47561248/mysql-how-to-join-two-subqueries

sql—Select the first N digits of an integer. (n.d.). Stack Overflow. Retrieved October 17, 2021, from https://stackoverflow.com/questions/30887327/select-the-first-n-digits-of-an-integer

Stitch Platform: Enterprise-grade security and compliance. (n.d.-a). Stitch. Retrieved November 10, 2021, from https://www.stitchdata.com/platform/security/

Stitch Platform: Extensibility with Singer and the Import API. (n.d.-b). Stitch. Retrieved November 10, 2021, from https://www.stitchdata.com/platform/extensibility/

Stitch Platform: Orchestration. (n.d.-c). Stitch. Retrieved November 10, 2021, from https://www.stitchdata.com/platform/orchestration/

Stitch Platform: Performance and Reliability. (n.d.-d). Stitch. Retrieved November 10, 2021, from https://www.stitchdata.com/platform/performance/

Surrogate Key in Data Warehouse, What, When and Why. (n.d.). *Data Integration Solutions*. Retrieved October 17, 2021, from http://www.disoln.org/2013/11/Surrogate-Key-in-Data-Warehouse-What-When-Why-and-Why-Not.html

Systems, C. (n.d.). *SQL Server: Data Warehouse Indexing Strategy*. Catapult Systems. Retrieved November 6, 2021, from https://www.catapultsystems.com/blogs/sql-server-data-warehouse-indexing-strategy/

tsql - How do I get the month and day with leading 0's in SQL? (E.g. 9 => 09). (n.d.). Stack Overflow. Retrieved November 4, 2021, from https://stackoverflow.com/questions/13804096/how-do-i-get-the-month-and-day-with-leading-0s-in-sql-e-g-9-09

UPSERT Functionality in SQL Server 2008—DatabaseJournal.com. (n.d.). Retrieved November 6, 2021, from https://www.databasejournal.com/features/mssql/article.php/3739131/UPSERT-Functionality-in-SQL-Server-2008.htm

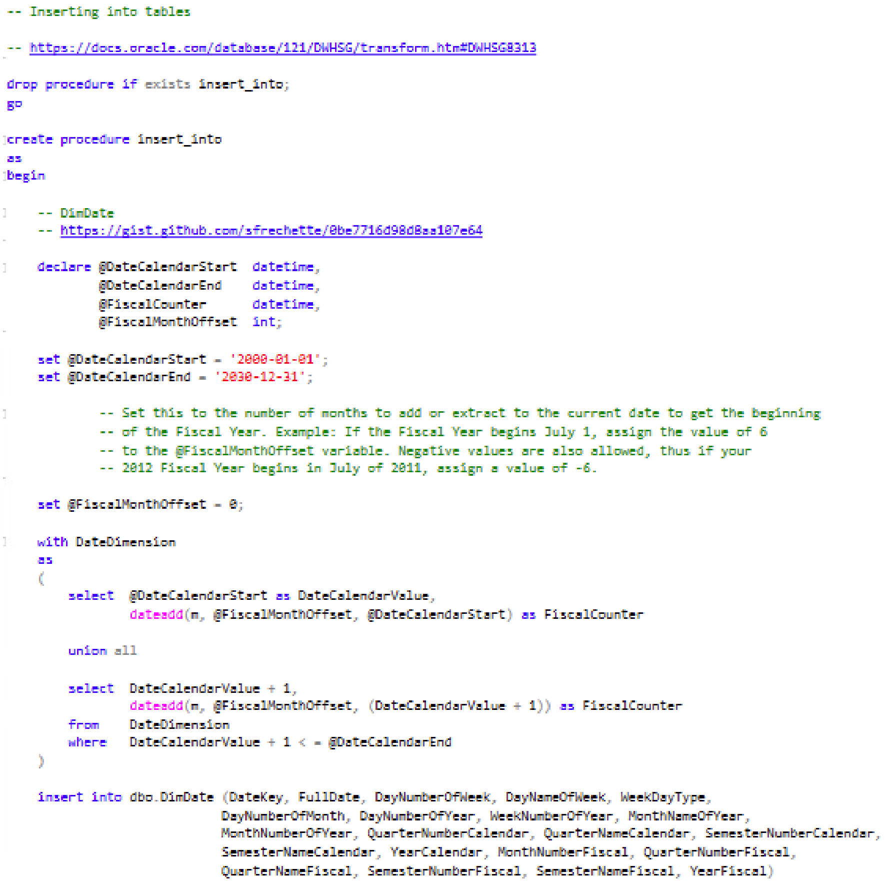
Welcome to Stitch Documentation | Stitch Documentation. (n.d.). Stitch Docs. Retrieved November 10, 2021, from https://www.stitchdata.com/docs

What is dbt? (n.d.). Transform Data in Your Warehouse. Retrieved November 10, 2021, from https://www.getdbt.com/product/what-is-dbt/

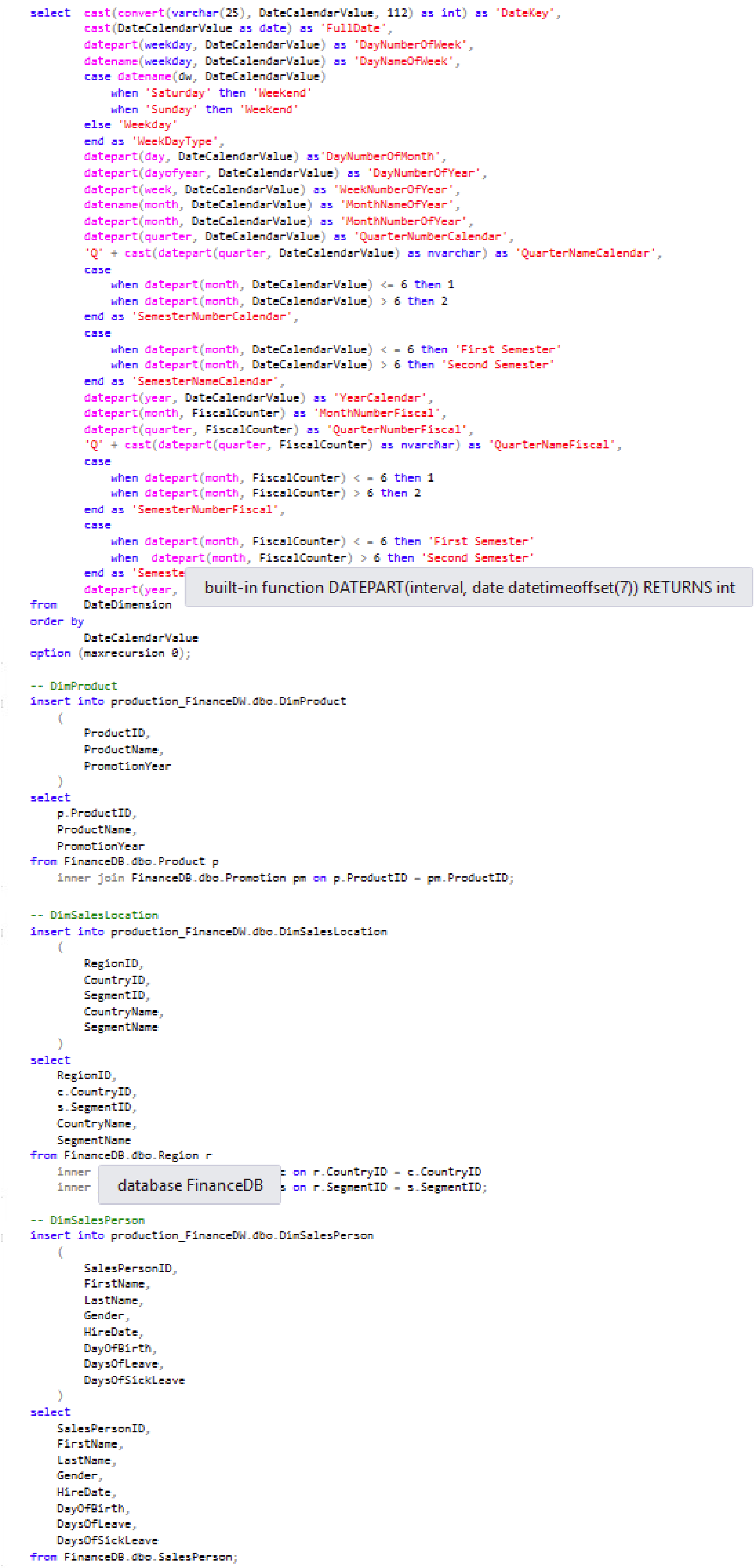
XiaoyuMSFT. (n.d.). *MERGE (Transact-SQL)—SQL Server*. Retrieved November 6, 2021, from https://docs.microsoft.com/en-us/sql/t-sql/statements/merge-transact-sql

Appendices

Appendix A

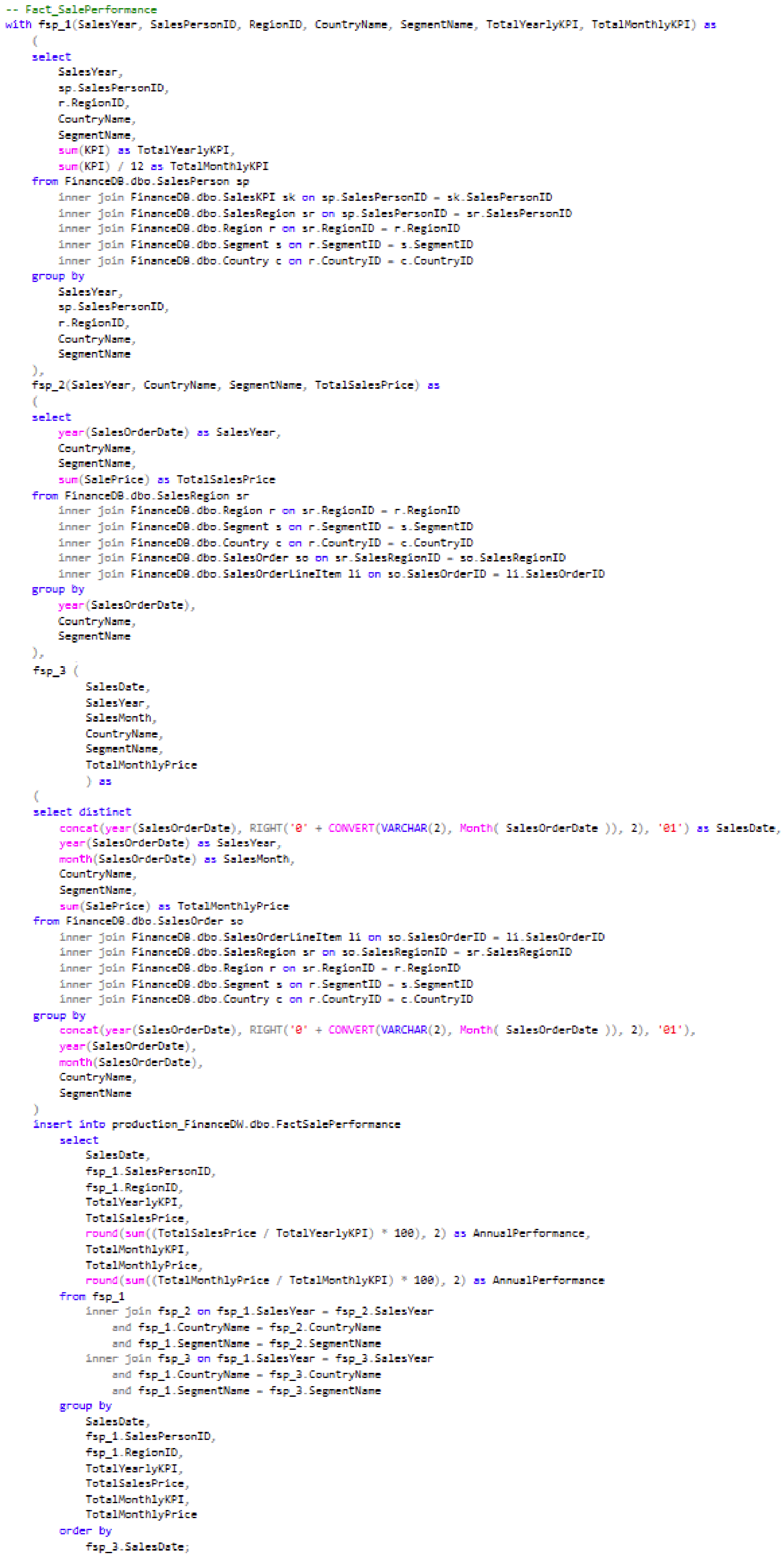


```
-- Inserting into tables

-- https://docs.oracle.com/database/121/DWHSG/transform.htm#DWHSG8313

drop procedure if exists insert_into;
go

create procedure insert_into
as
begin

    -- DimDate
    -- https://gist.github.com/sfrechette/0be7716d98d8aa107e64

    declare @DateCalendarStart  datetime,
            @DateCalendarEnd    datetime,
            @FiscalCounter      datetime,
            @FiscalMonthOffset  int;

    set @DateCalendarStart = '2000-01-01';
    set @DateCalendarEnd = '2030-12-31';

            -- Set this to the number of months to add or extract to the current date to get the beginning
            -- of the Fiscal Year. Example: If the Fiscal Year begins July 1, assign the value of 6
            -- to the @FiscalMonthOffset variable. Negative values are also allowed, thus if your
            -- 2012 Fiscal Year begins in July of 2011, assign a value of -6.

    set @FiscalMonthOffset = 0;

    with DateDimension
    as
    (
        select  @DateCalendarStart as DateCalendarValue,
                dateadd(m, @FiscalMonthOffset, @DateCalendarStart) as FiscalCounter

        union all

        select  DateCalendarValue + 1,
                dateadd(m, @FiscalMonthOffset, (DateCalendarValue + 1)) as FiscalCounter
        from    DateDimension
        where   DateCalendarValue + 1 < = @DateCalendarEnd
    )

    insert into dbo.DimDate (DateKey, FullDate, DayNumberOfWeek, DayNameOfWeek, WeekDayType,
                            DayNumberOfMonth, DayNumberOfYear, WeekNumberOfYear, MonthNameOfYear,
                            MonthNumberOfYear, QuarterNumberCalendar, QuarterNameCalendar, SemesterNumberCalendar,
                            SemesterNameCalendar, YearCalendar, MonthNumberFiscal, QuarterNumberFiscal,
                            QuarterNameFiscal, SemesterNumberFiscal, SemesterNameFiscal, YearFiscal)
```

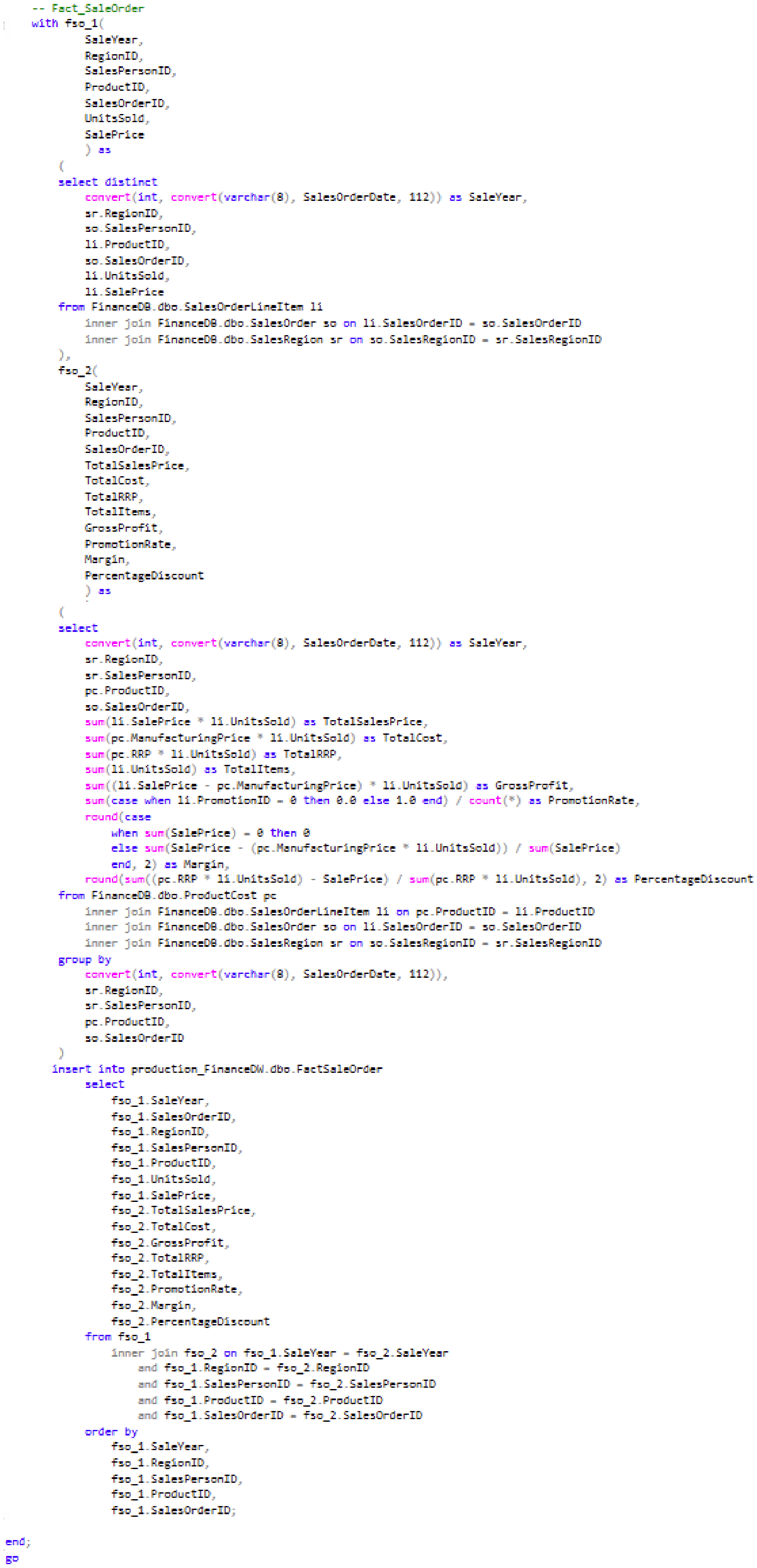


built-in function DATEPART(interval, date datetimeoffset(7)) RETURNS int
database FinanceDB

```sql
select  cast(convert(varchar(25), DateCalendarValue, 112) as int) as 'DateKey',
        cast(DateCalendarValue as date) as 'FullDate',
        datepart(weekday, DateCalendarValue) as 'DayNumberOfWeek',
        datename(weekday, DateCalendarValue) as 'DayNameOfWeek',
        case datename(dw, DateCalendarValue)
            when 'Saturday' then 'Weekend'
            when 'Sunday' then 'Weekend'
        else 'Weekday'
        end as 'WeekDayType',
        datepart(day, DateCalendarValue) as'DayNumberOfMonth',
        datepart(dayofyear, DateCalendarValue) as 'DayNumberOfYear',
        datepart(week, DateCalendarValue) as 'WeekNumberOfYear',
        datename(month, DateCalendarValue) as 'MonthNameOfYear',
        datepart(month, DateCalendarValue) as 'MonthNumberOfYear',
        datepart(quarter, DateCalendarValue) as 'QuarterNumberCalendar',
        'Q' + cast(datepart(quarter, DateCalendarValue) as nvarchar) as 'QuarterNameCalendar',
        case
            when datepart(month, DateCalendarValue) <= 6 then 1
            when datepart(month, DateCalendarValue) > 6 then 2
        end as 'SemesterNumberCalendar',
        case
            when datepart(month, DateCalendarValue) < = 6 then 'First Semester'
            when datepart(month, DateCalendarValue) > 6 then 'Second Semester'
        end as 'SemesterNameCalendar',
        datepart(year, DateCalendarValue) as 'YearCalendar',
        datepart(month, FiscalCounter) as 'MonthNumberFiscal',
        datepart(quarter, FiscalCounter) as 'QuarterNumberFiscal',
        'Q' + cast(datepart(quarter, FiscalCounter) as nvarchar) as 'QuarterNameFiscal',
        case
            when datepart(month, FiscalCounter) < = 6 then 1
            when datepart(month, FiscalCounter) > 6 then 2
        end as 'SemesterNumberFiscal',
        case
            when datepart(month, FiscalCounter) < = 6 then 'First Semester'
            when  datepart(month, FiscalCounter) > 6 then 'Second Semester'
        end as 'Semeste
        datepart(year,
from    DateDimension
order by
        DateCalendarValue
option (maxrecursion 0);

-- DimProduct
insert into production_FinanceDW.dbo.DimProduct
    (
        ProductID,
        ProductName,
        PromotionYear
    )
select
    p.ProductID,
    ProductName,
    PromotionYear
from FinanceDB.dbo.Product p
    inner join FinanceDB.dbo.Promotion pm on p.ProductID = pm.ProductID;

-- DimSalesLocation
insert into production_FinanceDW.dbo.DimSalesLocation
    (
        RegionID,
        CountryID,
        SegmentID,
        CountryName,
        SegmentName
    )
select
    RegionID,
    c.CountryID,
    s.SegmentID,
    CountryName,
    SegmentName
from FinanceDB.dbo.Region r
    inner                          c on r.CountryID = c.CountryID
    inner                          s on r.SegmentID = s.SegmentID;

-- DimSalesPerson
insert into production_FinanceDW.dbo.DimSalesPerson
    (
        SalesPersonID,
        FirstName,
        LastName,
        Gender,
        HireDate,
        DayOfBirth,
        DaysOfLeave,
        DaysOfSickLeave
    )
select
    SalesPersonID,
    FirstName,
    LastName,
    Gender,
    HireDate,
    DayOfBirth,
    DaysOfLeave,
    DaysOfSickLeave
from FinanceDB.dbo.SalesPerson;
```

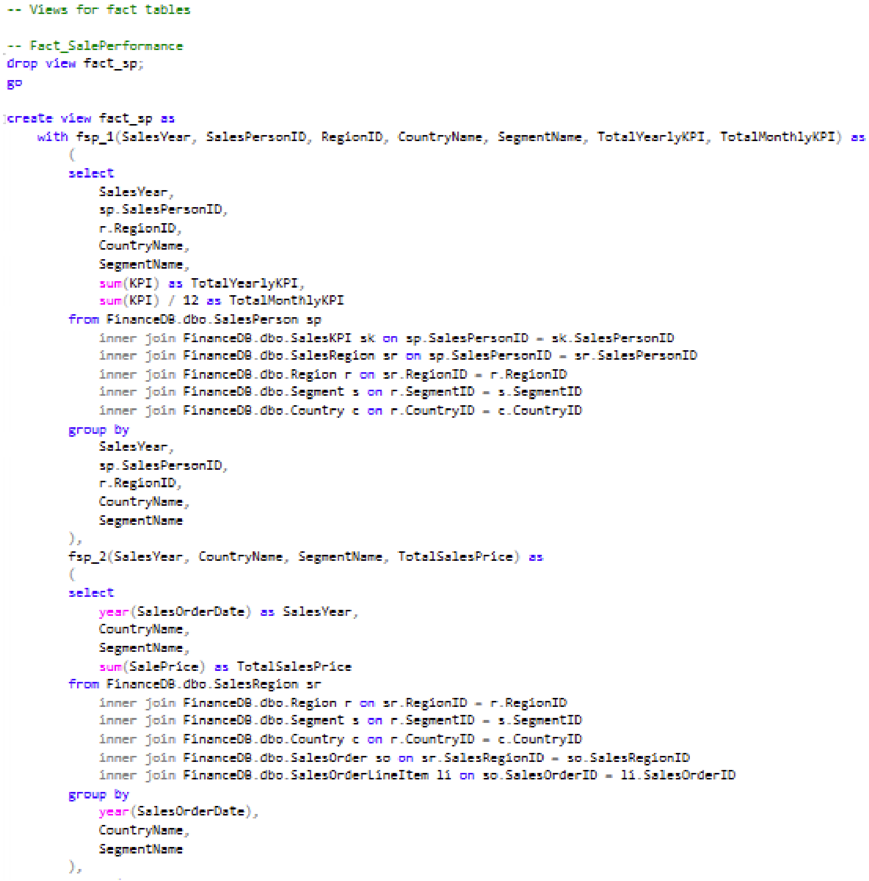


```sql
-- Fact_SalePerformance
with fsp_1(SalesYear, SalesPersonID, RegionID, CountryName, SegmentName, TotalYearlyKPI, TotalMonthlyKPI) as
    (
    select
        SalesYear,
        sp.SalesPersonID,
        r.RegionID,
        CountryName,
        SegmentName,
        sum(KPI) as TotalYearlyKPI,
        sum(KPI) / 12 as TotalMonthlyKPI
    from FinanceDB.dbo.SalesPerson sp
        inner join FinanceDB.dbo.SalesKPI sk on sp.SalesPersonID = sk.SalesPersonID
        inner join FinanceDB.dbo.SalesRegion sr on sp.SalesPersonID = sr.SalesPersonID
        inner join FinanceDB.dbo.Region r on sr.RegionID = r.RegionID
        inner join FinanceDB.dbo.Segment s on r.SegmentID = s.SegmentID
        inner join FinanceDB.dbo.Country c on r.CountryID = c.CountryID
    group by
        SalesYear,
        sp.SalesPersonID,
        r.RegionID,
        CountryName,
        SegmentName
    ),
    fsp_2(SalesYear, CountryName, SegmentName, TotalSalesPrice) as
    (
    select
        year(SalesOrderDate) as SalesYear,
        CountryName,
        SegmentName,
        sum(SalePrice) as TotalSalesPrice
    from FinanceDB.dbo.SalesRegion sr
        inner join FinanceDB.dbo.Region r on sr.RegionID = r.RegionID
        inner join FinanceDB.dbo.Segment s on r.SegmentID = s.SegmentID
        inner join FinanceDB.dbo.Country c on r.CountryID = c.CountryID
        inner join FinanceDB.dbo.SalesOrder so on sr.SalesRegionID = so.SalesRegionID
        inner join FinanceDB.dbo.SalesOrderLineItem li on so.SalesOrderID = li.SalesOrderID
    group by
        year(SalesOrderDate),
        CountryName,
        SegmentName
    ),
    fsp_3 (
            SalesDate,
            SalesYear,
            SalesMonth,
            CountryName,
            SegmentName,
            TotalMonthlyPrice
            ) as
    (
    select distinct
        concat(year(SalesOrderDate), RIGHT('0' + CONVERT(VARCHAR(2), Month( SalesOrderDate )), 2), '01') as SalesDate,
        year(SalesOrderDate) as SalesYear,
        month(SalesOrderDate) as SalesMonth,
        CountryName,
        SegmentName,
        sum(SalePrice) as TotalMonthlyPrice
    from FinanceDB.dbo.SalesOrder so
        inner join FinanceDB.dbo.SalesOrderLineItem li on so.SalesOrderID = li.SalesOrderID
        inner join FinanceDB.dbo.SalesRegion sr on so.SalesRegionID = sr.SalesRegionID
        inner join FinanceDB.dbo.Region r on sr.RegionID = r.RegionID
        inner join FinanceDB.dbo.Segment s on r.SegmentID = s.SegmentID
        inner join FinanceDB.dbo.Country c on r.CountryID = c.CountryID
    group by
        concat(year(SalesOrderDate), RIGHT('0' + CONVERT(VARCHAR(2), Month( SalesOrderDate )), 2), '01'),
        year(SalesOrderDate),
        month(SalesOrderDate),
        CountryName,
        SegmentName
    )
    insert into production_FinanceDW.dbo.FactSalePerformance
        select
            SalesDate,
            fsp_1.SalesPersonID,
            fsp_1.RegionID,
            TotalYearlyKPI,
            TotalSalesPrice,
            round(sum((TotalSalesPrice / TotalYearlyKPI) * 100), 2) as AnnualPerformance,
            TotalMonthlyKPI,
            TotalMonthlyPrice,
            round(sum((TotalMonthlyPrice / TotalMonthlyKPI) * 100), 2) as AnnualPerformance
        from fsp_1
            inner join fsp_2 on fsp_1.SalesYear = fsp_2.SalesYear
                and fsp_1.CountryName = fsp_2.CountryName
                and fsp_1.SegmentName = fsp_2.SegmentName
            inner join fsp_3 on fsp_1.SalesYear = fsp_3.SalesYear
                and fsp_1.CountryName = fsp_3.CountryName
                and fsp_1.SegmentName = fsp_3.SegmentName
        group by
            SalesDate,
            fsp_1.SalesPersonID,
            fsp_1.RegionID,
            TotalYearlyKPI,
            TotalSalesPrice,
            TotalMonthlyKPI,
            TotalMonthlyPrice
        order by
            fsp_3.SalesDate;
```

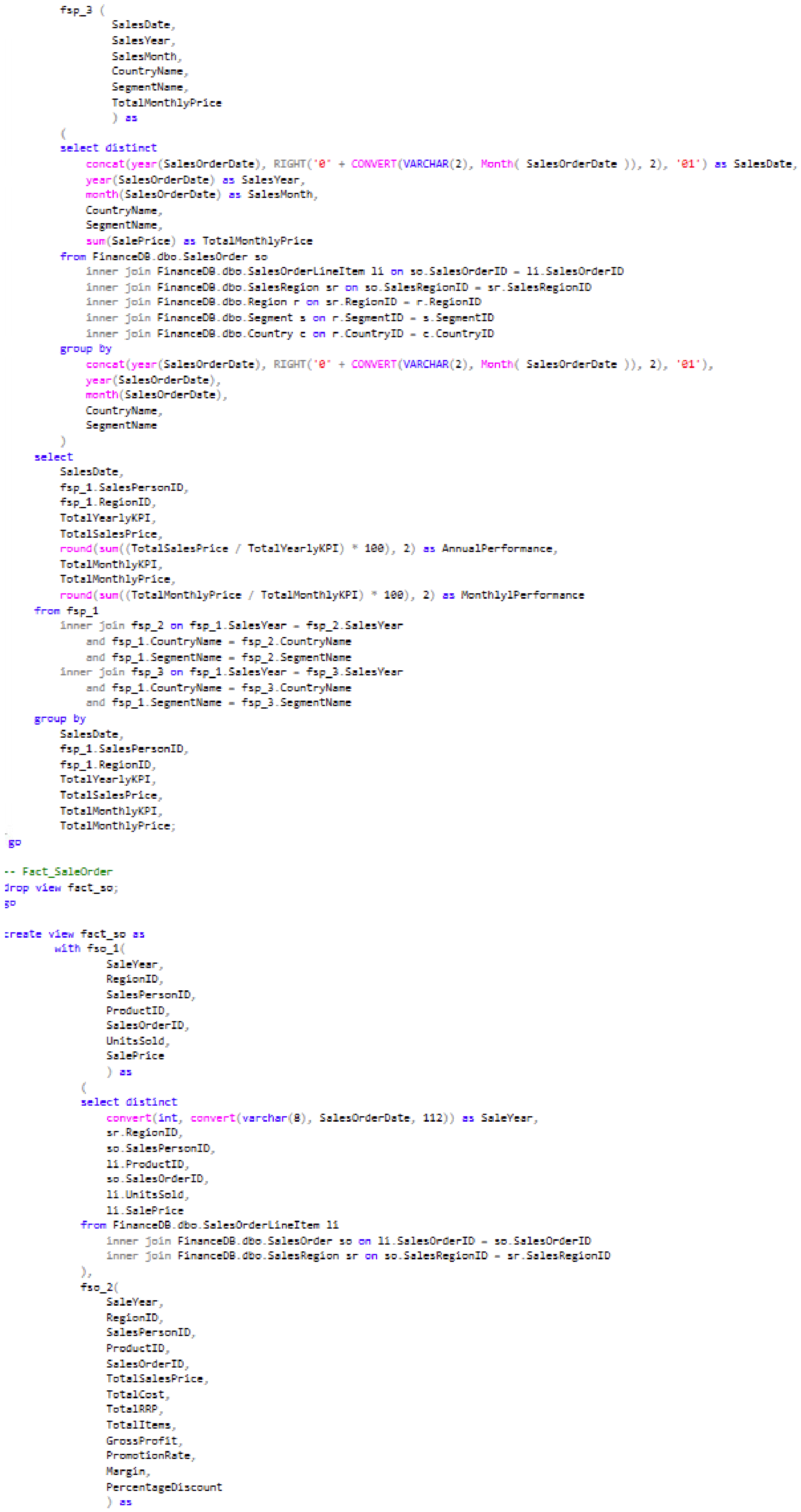


```sql
    -- Fact_SaleOrder
    with fso_1(
            SaleYear,
            RegionID,
            SalesPersonID,
            ProductID,
            SalesOrderID,
            UnitsSold,
            SalePrice
            ) as
        (
        select distinct
            convert(int, convert(varchar(8), SalesOrderDate, 112)) as SaleYear,
            sr.RegionID,
            so.SalesPersonID,
            li.ProductID,
            so.SalesOrderID,
            li.UnitsSold,
            li.SalePrice
        from FinanceDB.dbo.SalesOrderLineItem li
            inner join FinanceDB.dbo.SalesOrder so on li.SalesOrderID = so.SalesOrderID
            inner join FinanceDB.dbo.SalesRegion sr on so.SalesRegionID = sr.SalesRegionID
        ),
        fso_2(
            SaleYear,
            RegionID,
            SalesPersonID,
            ProductID,
            SalesOrderID,
            TotalSalesPrice,
            TotalCost,
            TotalRRP,
            TotalItems,
            GrossProfit,
            PromotionRate,
            Margin,
            PercentageDiscount
            ) as
        (
        select
            convert(int, convert(varchar(8), SalesOrderDate, 112)) as SaleYear,
            sr.RegionID,
            sr.SalesPersonID,
            pc.ProductID,
            so.SalesOrderID,
            sum(li.SalePrice * li.UnitsSold) as TotalSalesPrice,
            sum(pc.ManufacturingPrice * li.UnitsSold) as TotalCost,
            sum(pc.RRP * li.UnitsSold) as TotalRRP,
            sum(li.UnitsSold) as TotalItems,
            sum((li.SalePrice - pc.ManufacturingPrice) * li.UnitsSold) as GrossProfit,
            sum(case when li.PromotionID = 0 then 0.0 else 1.0 end) / count(*) as PromotionRate,
            round(case
                when sum(SalePrice) = 0 then 0
                else sum(SalePrice - (pc.ManufacturingPrice * li.UnitsSold)) / sum(SalePrice)
                end, 2) as Margin,
            round(sum((pc.RRP * li.UnitsSold) - SalePrice) / sum(pc.RRP * li.UnitsSold), 2) as PercentageDiscount
        from FinanceDB.dbo.ProductCost pc
            inner join FinanceDB.dbo.SalesOrderLineItem li on pc.ProductID = li.ProductID
            inner join FinanceDB.dbo.SalesOrder so on li.SalesOrderID = so.SalesOrderID
            inner join FinanceDB.dbo.SalesRegion sr on so.SalesRegionID = sr.SalesRegionID
        group by
            convert(int, convert(varchar(8), SalesOrderDate, 112)),
            sr.RegionID,
            sr.SalesPersonID,
            pc.ProductID,
            so.SalesOrderID
        )
       insert into production_FinanceDW.dbo.FactSaleOrder
            select
                fso_1.SaleYear,
                fso_1.SalesOrderID,
                fso_1.RegionID,
                fso_1.SalesPersonID,
                fso_1.ProductID,
                fso_1.UnitsSold,
                fso_1.SalePrice,
                fso_2.TotalSalesPrice,
                fso_2.TotalCost,
                fso_2.GrossProfit,
                fso_2.TotalRRP,
                fso_2.TotalItems,
                fso_2.PromotionRate,
                fso_2.Margin,
                fso_2.PercentageDiscount
            from fso_1
                inner join fso_2 on fso_1.SaleYear = fso_2.SaleYear
                    and fso_1.RegionID = fso_2.RegionID
                    and fso_1.SalesPersonID = fso_2.SalesPersonID
                    and fso_1.ProductID = fso_2.ProductID
                    and fso_1.SalesOrderID = fso_2.SalesOrderID
            order by
                fso_1.SaleYear,
                fso_1.RegionID,
                fso_1.SalesPersonID,
                fso_1.ProductID,
                fso_1.SalesOrderID;

end;
go
```

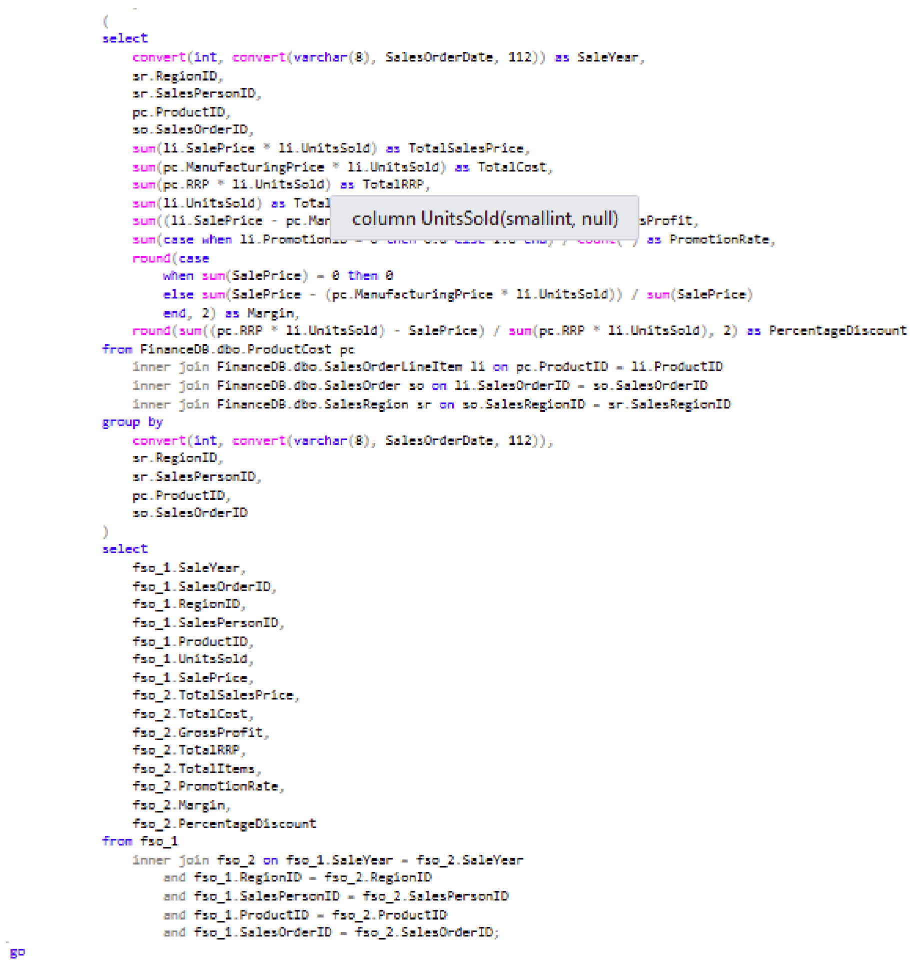
Appendix B



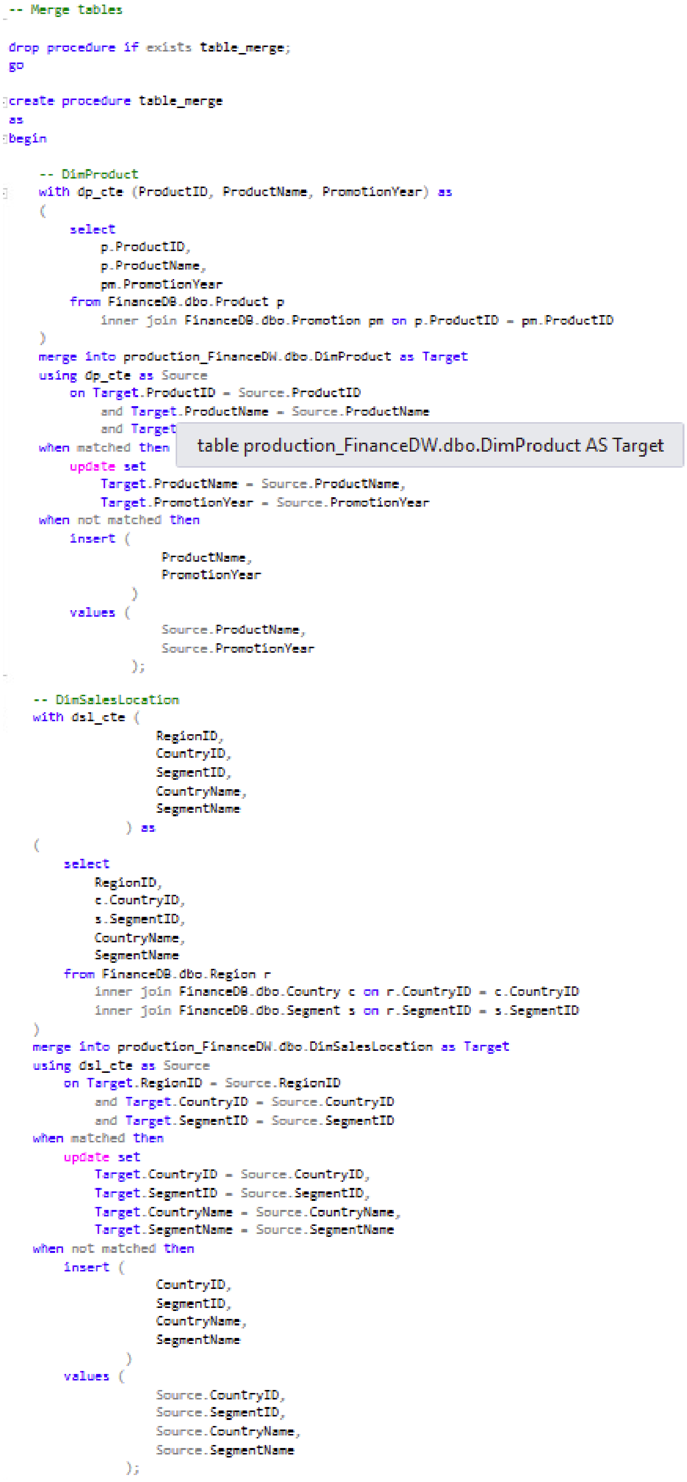
```sql
-- Views for fact tables

-- Fact_SalePerformance
drop view fact_sp;
go

create view fact_sp as
    with fsp_1(SalesYear, SalesPersonID, RegionID, CountryName, SegmentName, TotalYearlyKPI, TotalMonthlyKPI) as
        (
        select
            SalesYear,
            sp.SalesPersonID,
            r.RegionID,
            CountryName,
            SegmentName,
            sum(KPI) as TotalYearlyKPI,
            sum(KPI) / 12 as TotalMonthlyKPI
        from FinanceDB.dbo.SalesPerson sp
            inner join FinanceDB.dbo.SalesKPI sk on sp.SalesPersonID = sk.SalesPersonID
            inner join FinanceDB.dbo.SalesRegion sr on sp.SalesPersonID = sr.SalesPersonID
            inner join FinanceDB.dbo.Region r on sr.RegionID = r.RegionID
            inner join FinanceDB.dbo.Segment s on r.SegmentID = s.SegmentID
            inner join FinanceDB.dbo.Country c on r.CountryID = c.CountryID
        group by
            SalesYear,
            sp.SalesPersonID,
            r.RegionID,
            CountryName,
            SegmentName
        ),
        fsp_2(SalesYear, CountryName, SegmentName, TotalSalesPrice) as
        (
        select
            year(SalesOrderDate) as SalesYear,
            CountryName,
            SegmentName,
            sum(SalePrice) as TotalSalesPrice
        from FinanceDB.dbo.SalesRegion sr
            inner join FinanceDB.dbo.Region r on sr.RegionID = r.RegionID
            inner join FinanceDB.dbo.Segment s on r.SegmentID = s.SegmentID
            inner join FinanceDB.dbo.Country c on r.CountryID = c.CountryID
            inner join FinanceDB.dbo.SalesOrder so on sr.SalesRegionID = so.SalesRegionID
            inner join FinanceDB.dbo.SalesOrderLineItem li on so.SalesOrderID = li.SalesOrderID
        group by
            year(SalesOrderDate),
            CountryName,
            SegmentName
        ),
```



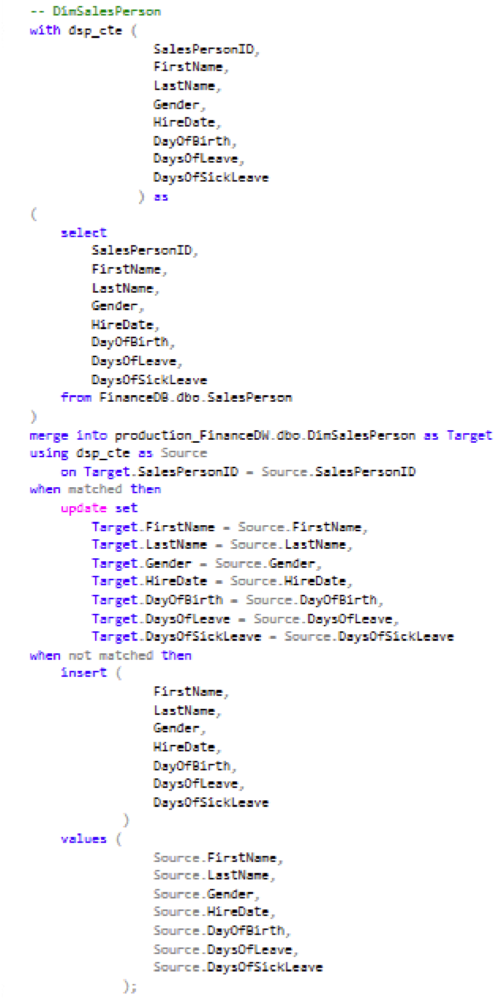
```
        fsp_3 (
                SalesDate,
                SalesYear,
                SalesMonth,
                CountryName,
                SegmentName,
                TotalMonthlyPrice
                ) as
        (
        select distinct
            concat(year(SalesOrderDate), RIGHT('0' + CONVERT(VARCHAR(2), Month( SalesOrderDate )), 2), '01') as SalesDate,
            year(SalesOrderDate) as SalesYear,
            month(SalesOrderDate) as SalesMonth,
            CountryName,
            SegmentName,
            sum(SalePrice) as TotalMonthlyPrice
        from FinanceDB.dbo.SalesOrder so
            inner join FinanceDB.dbo.SalesOrderLineItem li on so.SalesOrderID = li.SalesOrderID
            inner join FinanceDB.dbo.SalesRegion sr on so.SalesRegionID = sr.SalesRegionID
            inner join FinanceDB.dbo.Region r on sr.RegionID = r.RegionID
            inner join FinanceDB.dbo.Segment s on r.SegmentID = s.SegmentID
            inner join FinanceDB.dbo.Country c on r.CountryID = c.CountryID
        group by
            concat(year(SalesOrderDate), RIGHT('0' + CONVERT(VARCHAR(2), Month( SalesOrderDate )), 2), '01'),
            year(SalesOrderDate),
            month(SalesOrderDate),
            CountryName,
            SegmentName
        )
    select
        SalesDate,
        fsp_1.SalesPersonID,
        fsp_1.RegionID,
        TotalYearlyKPI,
        TotalSalesPrice,
        round(sum((TotalSalesPrice / TotalYearlyKPI) * 100), 2) as AnnualPerformance,
        TotalMonthlyKPI,
        TotalMonthlyPrice,
        round(sum((TotalMonthlyPrice / TotalMonthlyKPI) * 100), 2) as Monthly1Performance
    from fsp_1
        inner join fsp_2 on fsp_1.SalesYear = fsp_2.SalesYear
            and fsp_1.CountryName = fsp_2.CountryName
            and fsp_1.SegmentName = fsp_2.SegmentName
        inner join fsp_3 on fsp_1.SalesYear = fsp_3.SalesYear
            and fsp_1.CountryName = fsp_3.CountryName
            and fsp_1.SegmentName = fsp_3.SegmentName
    group by
        SalesDate,
        fsp_1.SalesPersonID,
        fsp_1.RegionID,
        TotalYearlyKPI,
        TotalSalesPrice,
        TotalMonthlyKPI,
        TotalMonthlyPrice;
go

-- Fact_SaleOrder
drop view fact_so;
go

create view fact_so as
        with fso_1(
                SaleYear,
                RegionID,
                SalesPersonID,
                ProductID,
                SalesOrderID,
                UnitsSold,
                SalePrice
                ) as
            (
            select distinct
                convert(int, convert(varchar(8), SalesOrderDate, 112)) as SaleYear,
                sr.RegionID,
                so.SalesPersonID,
                li.ProductID,
                so.SalesOrderID,
                li.UnitsSold,
                li.SalePrice
            from FinanceDB.dbo.SalesOrderLineItem li
                inner join FinanceDB.dbo.SalesOrder so on li.SalesOrderID = so.SalesOrderID
                inner join FinanceDB.dbo.SalesRegion sr on so.SalesRegionID = sr.SalesRegionID
            ),
            fso_2(
                SaleYear,
                RegionID,
                SalesPersonID,
                ProductID,
                SalesOrderID,
                TotalSalesPrice,
                TotalCost,
                TotalRRP,
                TotalItems,
                GrossProfit,
                PromotionRate,
                Margin,
                PercentageDiscount
                ) as
```



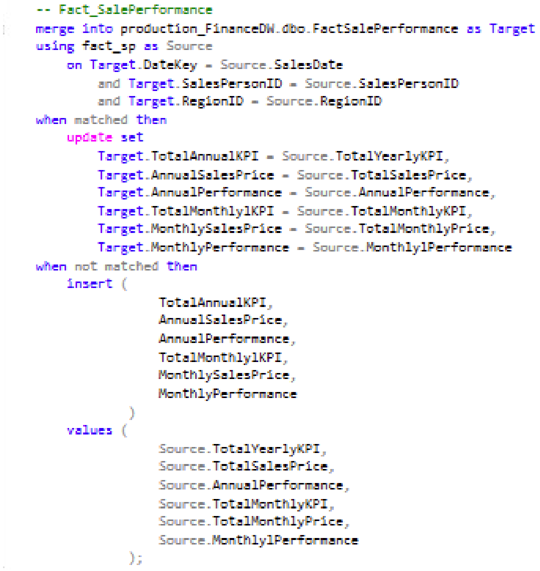
```
(
select
    convert(int, convert(varchar(8), SalesOrderDate, 112)) as SaleYear,
    sr.RegionID,
    sr.SalesPersonID,
    pc.ProductID,
    so.SalesOrderID,
    sum(li.SalePrice * li.UnitsSold) as TotalSalesPrice,
    sum(pc.ManufacturingPrice * li.UnitsSold) as TotalCost,
    sum(pc.RRP * li.UnitsSold) as TotalRRP,
    sum(li.UnitsSold) as Total
    sum((li.SalePrice - pc.Man                                  sProfit,
    sum(case when li.PromotionID = 0 then 0.0 else 1.0 end) / count(*) as PromotionRate,
    round(case
        when sum(SalePrice) = 0 then 0
        else sum(SalePrice - (pc.ManufacturingPrice * li.UnitsSold)) / sum(SalePrice)
        end, 2) as Margin,
    round(sum((pc.RRP * li.UnitsSold) - SalePrice) / sum(pc.RRP * li.UnitsSold), 2) as PercentageDiscount
from FinanceDB.dbo.ProductCost pc
    inner join FinanceDB.dbo.SalesOrderLineItem li on pc.ProductID = li.ProductID
    inner join FinanceDB.dbo.SalesOrder so on li.SalesOrderID = so.SalesOrderID
    inner join FinanceDB.dbo.SalesRegion sr on so.SalesRegionID = sr.SalesRegionID
group by
    convert(int, convert(varchar(8), SalesOrderDate, 112)),
    sr.RegionID,
    sr.SalesPersonID,
    pc.ProductID,
    so.SalesOrderID
)
select
    fso_1.SaleYear,
    fso_1.SalesOrderID,
    fso_1.RegionID,
    fso_1.SalesPersonID,
    fso_1.ProductID,
    fso_1.UnitsSold,
    fso_1.SalePrice,
    fso_2.TotalSalesPrice,
    fso_2.TotalCost,
    fso_2.GrossProfit,
    fso_2.TotalRRP,
    fso_2.TotalItems,
    fso_2.PromotionRate,
    fso_2.Margin,
    fso_2.PercentageDiscount
from fso_1
    inner join fso_2 on fso_1.SaleYear = fso_2.SaleYear
        and fso_1.RegionID = fso_2.RegionID
        and fso_1.SalesPersonID = fso_2.SalesPersonID
        and fso_1.ProductID = fso_2.ProductID
        and fso_1.SalesOrderID = fso_2.SalesOrderID;
go
```



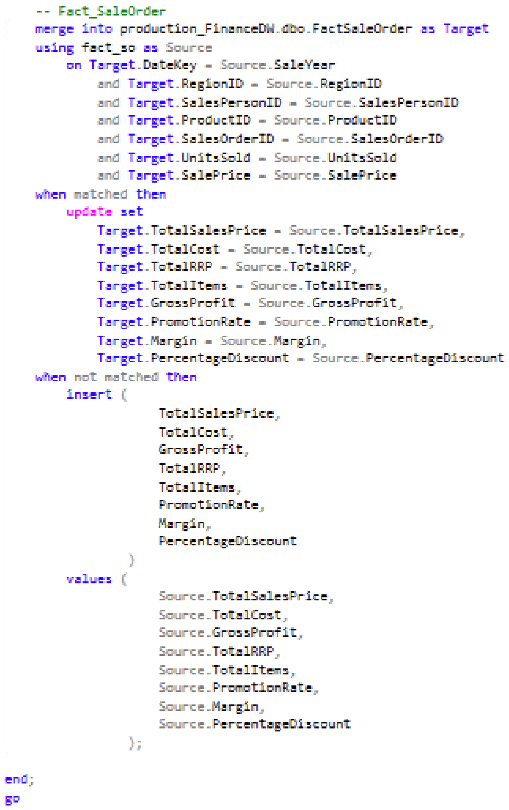
```sql
-- Merge tables

drop procedure if exists table_merge;
go

create procedure table_merge
as
begin

    -- DimProduct
    with dp_cte (ProductID, ProductName, PromotionYear) as
    (
        select
            p.ProductID,
            p.ProductName,
            pm.PromotionYear
        from FinanceDB.dbo.Product p
            inner join FinanceDB.dbo.Promotion pm on p.ProductID = pm.ProductID
    )
    merge into production_FinanceDW.dbo.DimProduct as Target
    using dp_cte as Source
        on Target.ProductID = Source.ProductID
            and Target.ProductName = Source.ProductName
            and Target
    when matched then
        update set
            Target.ProductName = Source.ProductName,
            Target.PromotionYear = Source.PromotionYear
    when not matched then
        insert (
                    ProductName,
                    PromotionYear
                )
        values (
                    Source.ProductName,
                    Source.PromotionYear
                );

    -- DimSalesLocation
    with dsl_cte (
                    RegionID,
                    CountryID,
                    SegmentID,
                    CountryName,
                    SegmentName
                ) as
    (
        select
            RegionID,
            c.CountryID,
            s.SegmentID,
            CountryName,
            SegmentName
        from FinanceDB.dbo.Region r
            inner join FinanceDB.dbo.Country c on r.CountryID = c.CountryID
            inner join FinanceDB.dbo.Segment s on r.SegmentID = s.SegmentID
    )
    merge into production_FinanceDW.dbo.DimSalesLocation as Target
    using dsl_cte as Source
        on Target.RegionID = Source.RegionID
            and Target.CountryID = Source.CountryID
            and Target.SegmentID = Source.SegmentID
    when matched then
        update set
            Target.CountryID = Source.CountryID,
            Target.SegmentID = Source.SegmentID,
            Target.CountryName = Source.CountryName,
            Target.SegmentName = Source.SegmentName
    when not matched then
        insert (
                    CountryID,
                    SegmentID,
                    CountryName,
                    SegmentName
                )
        values (
                    Source.CountryID,
                    Source.SegmentID,
                    Source.CountryName,
                    Source.SegmentName
                );
```



```sql
-- DimSalesPerson
with dsp_cte (
                SalesPersonID,
                FirstName,
                LastName,
                Gender,
                HireDate,
                DayOfBirth,
                DaysOfLeave,
                DaysOfSickLeave
              ) as
(
    select
        SalesPersonID,
        FirstName,
        LastName,
        Gender,
        HireDate,
        DayOfBirth,
        DaysOfLeave,
        DaysOfSickLeave
    from FinanceDB.dbo.SalesPerson
)
merge into production_FinanceDW.dbo.DimSalesPerson as Target
using dsp_cte as Source
    on Target.SalesPersonID = Source.SalesPersonID
when matched then
    update set
        Target.FirstName = Source.FirstName,
        Target.LastName = Source.LastName,
        Target.Gender = Source.Gender,
        Target.HireDate = Source.HireDate,
        Target.DayOfBirth = Source.DayOfBirth,
        Target.DaysOfLeave = Source.DaysOfLeave,
        Target.DaysOfSickLeave = Source.DaysOfSickLeave
when not matched then
    insert (
                FirstName,
                LastName,
                Gender,
                HireDate,
                DayOfBirth,
                DaysOfLeave,
                DaysOfSickLeave
            )
    values (
                Source.FirstName,
                Source.LastName,
                Source.Gender,
                Source.HireDate,
                Source.DayOfBirth,
                Source.DaysOfLeave,
                Source.DaysOfSickLeave
            );
```



```sql
-- Fact_SalePerformance
merge into production_FinanceDW.dbo.FactSalePerformance as Target
using fact_sp as Source
    on Target.DateKey = Source.SalesDate
        and Target.SalesPersonID = Source.SalesPersonID
        and Target.RegionID = Source.RegionID
when matched then
    update set
        Target.TotalAnnualKPI = Source.TotalYearlyKPI,
        Target.AnnualSalesPrice = Source.TotalSalesPrice,
        Target.AnnualPerformance = Source.AnnualPerformance,
        Target.TotalMonthly1KPI = Source.TotalMonthlyKPI,
        Target.MonthlySalesPrice = Source.TotalMonthlyPrice,
        Target.MonthlyPerformance = Source.Monthly1Performance
when not matched then
    insert (
                TotalAnnualKPI,
                AnnualSalesPrice,
                AnnualPerformance,
                TotalMonthly1KPI,
                MonthlySalesPrice,
                MonthlyPerformance
            )
    values (
                Source.TotalYearlyKPI,
                Source.TotalSalesPrice,
                Source.AnnualPerformance,
                Source.TotalMonthlyKPI,
                Source.TotalMonthlyPrice,
                Source.Monthly1Performance
            );
```



```
    -- Fact_SaleOrder
    merge into production_FinanceDW.dbo.FactSaleOrder as Target
    using fact_so as Source
        on Target.DateKey = Source.SaleYear
            and Target.RegionID = Source.RegionID
            and Target.SalesPersonID = Source.SalesPersonID
            and Target.ProductID = Source.ProductID
            and Target.SalesOrderID = Source.SalesOrderID
            and Target.UnitsSold = Source.UnitsSold
            and Target.SalePrice = Source.SalePrice
    when matched then
        update set
            Target.TotalSalesPrice = Source.TotalSalesPrice,
            Target.TotalCost = Source.TotalCost,
            Target.TotalRRP = Source.TotalRRP,
            Target.TotalItems = Source.TotalItems,
            Target.GrossProfit = Source.GrossProfit,
            Target.PromotionRate = Source.PromotionRate,
            Target.Margin = Source.Margin,
            Target.PercentageDiscount = Source.PercentageDiscount
    when not matched then
        insert (
                    TotalSalesPrice,
                    TotalCost,
                    GrossProfit,
                    TotalRRP,
                    TotalItems,
                    PromotionRate,
                    Margin,
                    PercentageDiscount
                )
        values (
                    Source.TotalSalesPrice,
                    Source.TotalCost,
                    Source.GrossProfit,
                    Source.TotalRRP,
                    Source.TotalItems,
                    Source.PromotionRate,
                    Source.Margin,
                    Source.PercentageDiscount
                );

end;
go
```