Name	Berkay Berber
Class:	
Date:	6/14/2024

MINI-PROJECT 2 – FINAL ANALYSIS

TASK 1.1 – DIMENSIONAL MODEL – DETAILS:

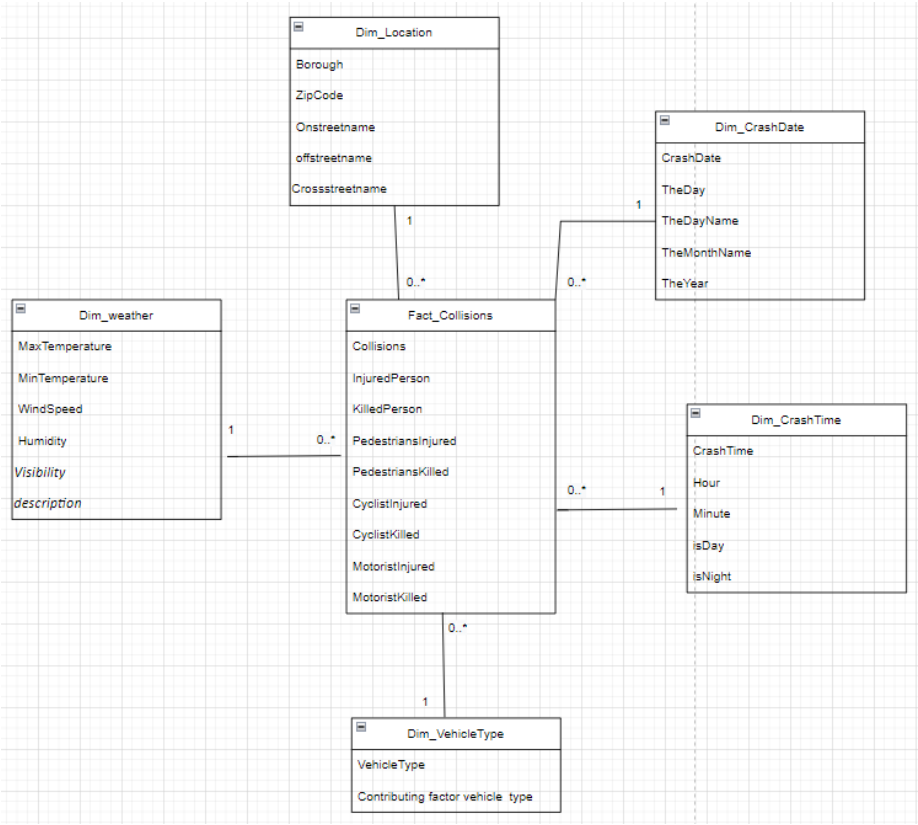
Please prepare the dimensional model for the selected dataset and specified user requirements, in accordance with the below specification and discuss the solution with the lecturer.

TASK 1.1 – DIMENSIONAL MODEL – SOLUTIONS:

1.1 DIMENSIONAL SCHEMA – DIAGRAM

Since Weather dataset columns are also a measure which needs to be created as a second fact table to follow the standards, our intention was decrease the complexity by store them in dimension table. Moreover, rather than turning diagram to snowflake by adding another dimension table called Weather_datetime, I intent to populate the necessary columns in the ETL process to extract relevant part of the datetime column. There is below representation of the final version of the discussed schema creation->

Below is the final version of Logical Diagram (Star schema):

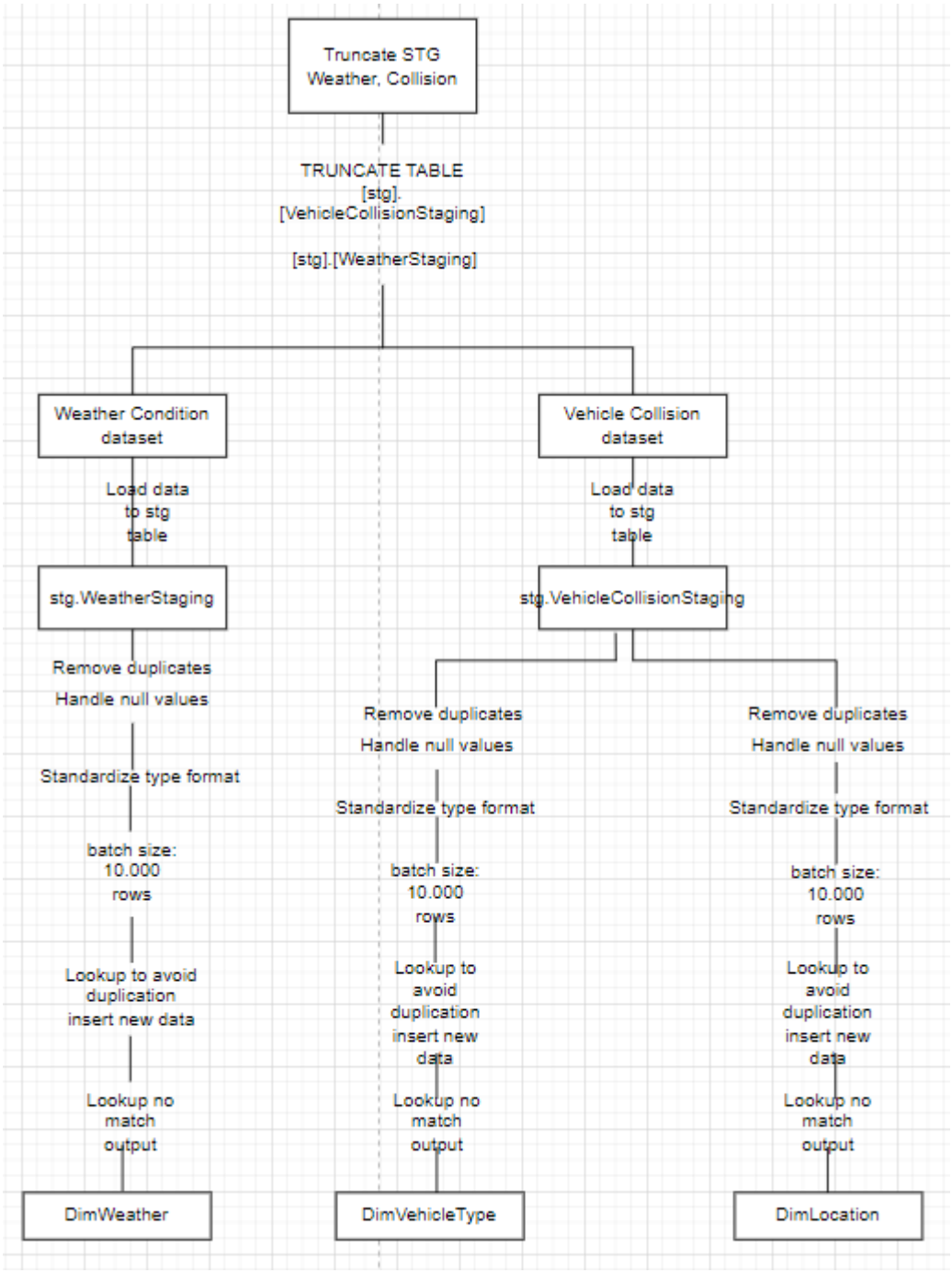


TASK 1.2 – ETL MODEL – DETAILS:

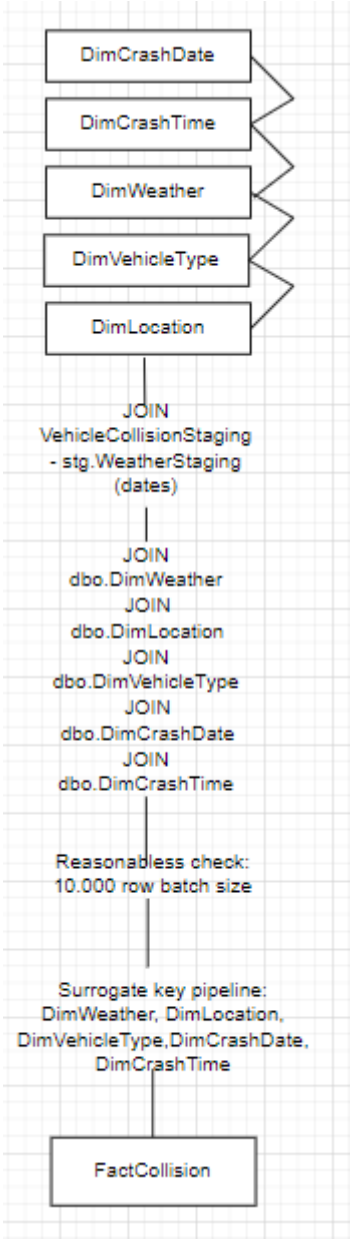
Please prepare the model of the ETL process, in accordance with the below specification and discuss the solution with the lecturer.

TASK 1.2 – ETL MODEL – SOLUTIONS:

1.2 ETL MAP



The above Etl map is set up the make sure we have a clean and organized process for loading and transforming the Weather and Collision data. Starting with clearing the staging tables, we make sure there is no leftover data messings. And steps like, removing duplicates, handling missing values, and standardizing data formats help keep the data clean and reliable. Using lookups to find new and existing records prevents that there will be no duplicates and keeps the data accurate over the time. The dimension tables for weather, vehicle types, locations, crash dates, crash times provide a foundation us to have a chance for detailed analysis and efficient.



I started with staging tables and transform that data into dimension tables for dimension tables. These dimensions are then linked with the staging data making sure all the detailed are captured and mapped correctly. As the given requirements, I process the data into batches checking for any issues and assigning unique keys to keep everything in order. Moreover, I load the data into FactCollision table setting up a robust system for analyzing both datasets over time.

MINI-PROJECT 2 – IMPLEMENTATION

TASK 2.1 – ETL PROCESS – DETAILS:

TASK 2.1 – ETL SYSTEM – SOLUTIONS:

ETL implementation – <u>using a tool of your choice</u> (suggested SSIS and MS SQL Server, remember about staging the data):

1. Divide your data into three batches (first batch can be, and should be, significantly larger)
2. Historical load of data – with the first batch
    a. Define target data structures – required dimension tables and a fact table
    b. Perform extraction of needed data from the source system. Please consider limiting the extraction to small portions of data (like 10000 rows), you'll that this is a required approach for larger datasets (required for **>85%**).
    c. Perform basic anomaly detection and resolution transformations. Define a basic set of data-quality screens and implement them (and further run against extracted data). Please note that a simple data-quality screen can check for null values and mark the rows as requiring attention. If you are using SSIS, please note that more

advanced screens can be implemented using external scripts. In short, define an approach and clean the identified anomalies

d. Perform needed data transformation - creation of assumed multi-dimensional scheme - star, snowflake, or fact constellation:
    i. Create dimension tables. Remember to integrate and standardize the needed information, to make names and values verbose, to introduce surrogate keys, to handle SCDs, please also include basic sanity checks against the completeness and correctness of data, e.g., row counts. Consider implementing automated tests (required for **>85%**).
    ii. Create fact table. Remember to integrate and standardize the needed information. Please also include basic sanity checks against the completeness and correctness of data, e.g., row counts. Consider implementing automated tests (required for **>85%**).
e. Finally load the prepared data into target tables.
    i. Define proper relations between dimensions and fact tables.
f. Use proper logging mechanisms and proper error handling (required for **>85%**). Remember to capture all erroneous data.

3. Incremental load of data – with the second and third batch
    a. Identify required extraction approach and implement it – remember to utilise a mechanism to capture only new/modified data (required for **>65%**). Perform extraction of needed data from the source system
    b. Perform basic anomaly detection and resolution transformations. Define a basic set of data-quality screens and implement them (and further run against extracted data). Please note that a simple data-quality screen can check for null values and mark the rows as requiring attention. If you are using SSIS, please note that more advanced screens can be implemented using external scripts. Consider implementing automated quality tests (required for >**50%)**. In short, define an approach and clean the identified anomalies.
    c. Perform needed data transformation - creation of assumed multi-dimensional scheme - star, snowflake, or fact constellation. Create dimension and fact tables. Remember Implement a mechanism to update only new/modified data (required for **>65%**)
    d. Finally load the prepared data into target tables.
    e. Use proper logging mechanisms and proper error handling (required for **>85%**). Remember to capture all erroneous data.

As the final solution, please upload your source files. In the final report provide a concise description of your implementation in accordance with the following points:

2.1.1 TARGET DATABASE CREATION SCRIPT

Before starting to SSIS, I import the data from the csv file into created crash database than create the tables and insert the data to see everything as it supposed to be in SSMS. Below is the target database creation script.

```
--STG Tables

CREATE TABLE stg.WeatherStaging
(
    name      VARCHAR(512),
    datetime  VARCHAR(512),
    tempmax   DECIMAL(10,2),
    tempmin   DECIMAL(10,2),
    humidity  INT,
    windspeed DECIMAL(10,2),
    visibility       INT,
    description      VARCHAR(512)
);

CREATE TABLE stg.VehicleCollisionStaging
(
    CRASHDATE VARCHAR(512),
    CRASHTIME VARCHAR(512),
    BOROUGH   VARCHAR(512),
    ZIPCODE   INT,
    ONSTREETNAME     VARCHAR(512),
    CROSSSTREETNAME  VARCHAR(512),
    OFFSTREETNAME    VARCHAR(512),
    NUMBEROFPERSONSINJURED  INT,
```

```
        NUMBEROFPERSONSKILLED       INT,
        NUMBEROFPEDESTRIANSINJURED  INT,
        NUMBEROFPEDESTRIANSKILLED   INT,
        NUMBEROFCYCLISTINJURED      INT,
        NUMBEROFCYCLISTKILLED       INT,
        NUMBEROFMOTORISTINJURED     INT,
        NUMBEROFMOTORISTKILLED      INT,
        CONTRIBUTINGFACTORVEHICLE1  VARCHAR(512),
        VEHICLETYPECODE1     VARCHAR(512)
);


--Dimension

CREATE TABLE dbo.DimWeather (
    weatherID INT IDENTITY(1,1) PRIMARY KEY,
    MaxTemperature NUMERIC(10,2) NOT NULL,
    MinTemperature NUMERIC(10,2) NOT NULL,
    humidity NUMERIC(10,2) NOT NULL,
    windspeed NUMERIC(10,2) NOT NULL,
    visibility NUMERIC(10,2) NOT NULL,
    [description] VARCHAR(255) NOT NULL
);

CREATE TABLE DimLocation (
    LocationID INT IDENTITY(1,1) PRIMARY KEY,
    borough VARCHAR(50) NOT NULL,
    zipcode VARCHAR(10) NOT NULL,
    onstreetname VARCHAR(100) NOT NULL,
    offstreetname VARCHAR(100) NOT NULL,
    crossstreetname VARCHAR(100) NOT NULL
);


CREATE TABLE dimVehicleType (
    VehicleTypeID INT IDENTITY(1,1) PRIMARY KEY,
    VehicleType VARCHAR(50) NOT NULL,
    ContributingFactortype VARCHAR(50) NOT NULL
)

CREATE TABLE dbo.FactCollisions (
    CollisionID INT IDENTITY(1,1) PRIMARY KEY,
    CrashDateID INT NOT NULL,
    CrashTimeID INT NOT NULL,
    LocationID INT NOT NULL,
    VehicleTypeID INT NOT NULL,
    WeatherID INT NOT NULL,
    InjuredPerson SMALLINT NOT NULL,
    KilledPerson SMALLINT NOT NULL,
    PedestriansInjured SMALLINT NOT NULL,
    PedestriansKilled SMALLINT NOT NULL,
    CyclistInjured SMALLINT NOT NULL,
    CyclistKilled SMALLINT NOT NULL,
    MotoristInjured SMALLINT NOT NULL,
    MotoristKilled SMALLINT NOT NULL,
    FOREIGN KEY (CrashDateID) REFERENCES dimCrashDate(DateID),
    FOREIGN KEY (CrashTimeID) REFERENCES dimCrashTime([TimeID]),
    FOREIGN KEY (LocationID) REFERENCES dimLocation(LocationID),
    FOREIGN KEY (VehicleTypeID) REFERENCES dimVehicleType(VehicleTypeID),
    FOREIGN KEY (WeatherID) REFERENCES DimWeather(weatherID)
);
```

DimCrashDate and dimCrashtime tables creation were presented in the dimension description.

2.1.2 ETL PROCESSES

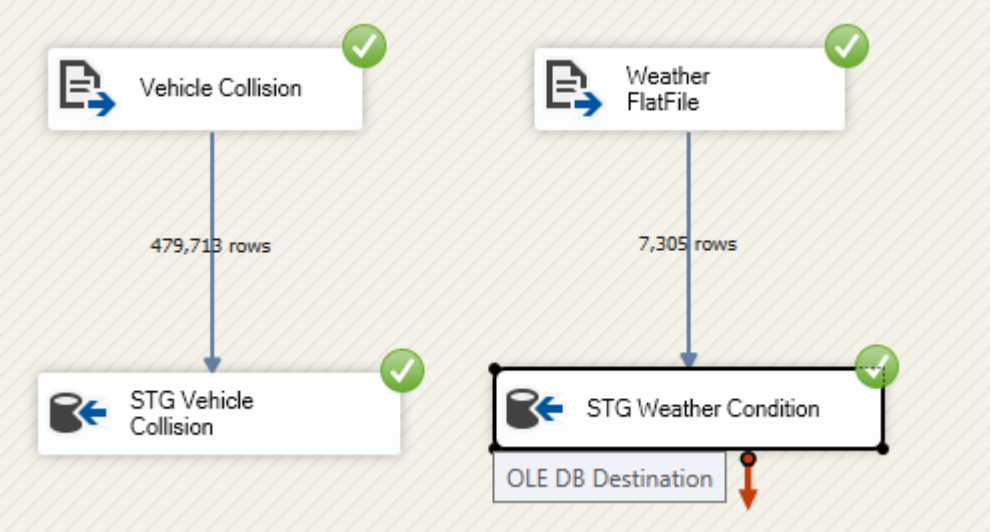
Please specify stages of your implemented approach and provided detailed description of each stage, provide details for each target table. Please specify tools that you have used and highlight their purpose in the process

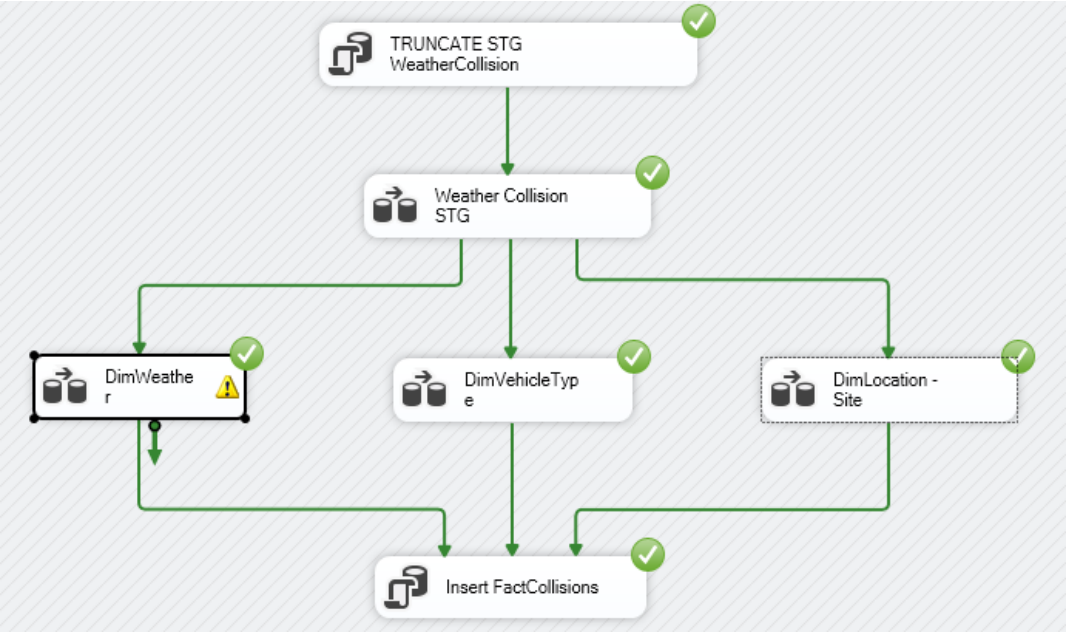
2.1.2.1 GENERAL APPROACH

Since the source website of the weather file was limited for user to download only 1 file for 2 years I needed to download them partially and then merge them and change lower case to upper case for name of the state names using python. And then using pandas library I cleaned the data based on my request. (.py file has attached to zipfile (motorvehicle, weather)). Changed to utf-8 etc. The reason is that I summarized my data based on my requirements is that I didn't need these unnecessary column in my database to keep the rest of the column for providing user needs. Moreover, I had very inconsistent data with lots of null values, together with the datetime and name columns which blocks the analysis part. In order to avoid the time complexity, I re-format my data.

In general, I have created a database in mssql because I have 2 time related dimensions which are crash date and crash time that not going to be imported from any data file. These time related dimensions are 1 time created for historical data within a range is set up once and doesn't require update frequently. So through this way we wont need the edit anything when the new data found to improve the analysis. The range of years is 4 years between 2016 – 2019 and time is each possible combination of time that there is in a day etc. Control flow of the task is presented in the image2 below. First process begins with truncation of Weather and Collision staging tables to ensure that the staging tables are cleared of any previous data and preparing it for the new data load. After truncation, the 'Weather Collision STG' step is created to load the new data into staging table. And than Dimension table updates starts for the DimWeather, DimVehicleType and DimLocation tables. And after all that finally populating the fact table with integrated and transformed data. The dimension tables were updated, the 'Insert FactCollision' step aggregates and inserts the processes data into the fact table in intention of involving transformation of staged data combining with the dimensions to create a comprehensive dataset for analysis.

The first image below is Vehicle Collision and Weather Condition datas loaded into a staging table. I extract data from the source files. As we can see a 479.718 and 7.305 rows are processed to each stg table.

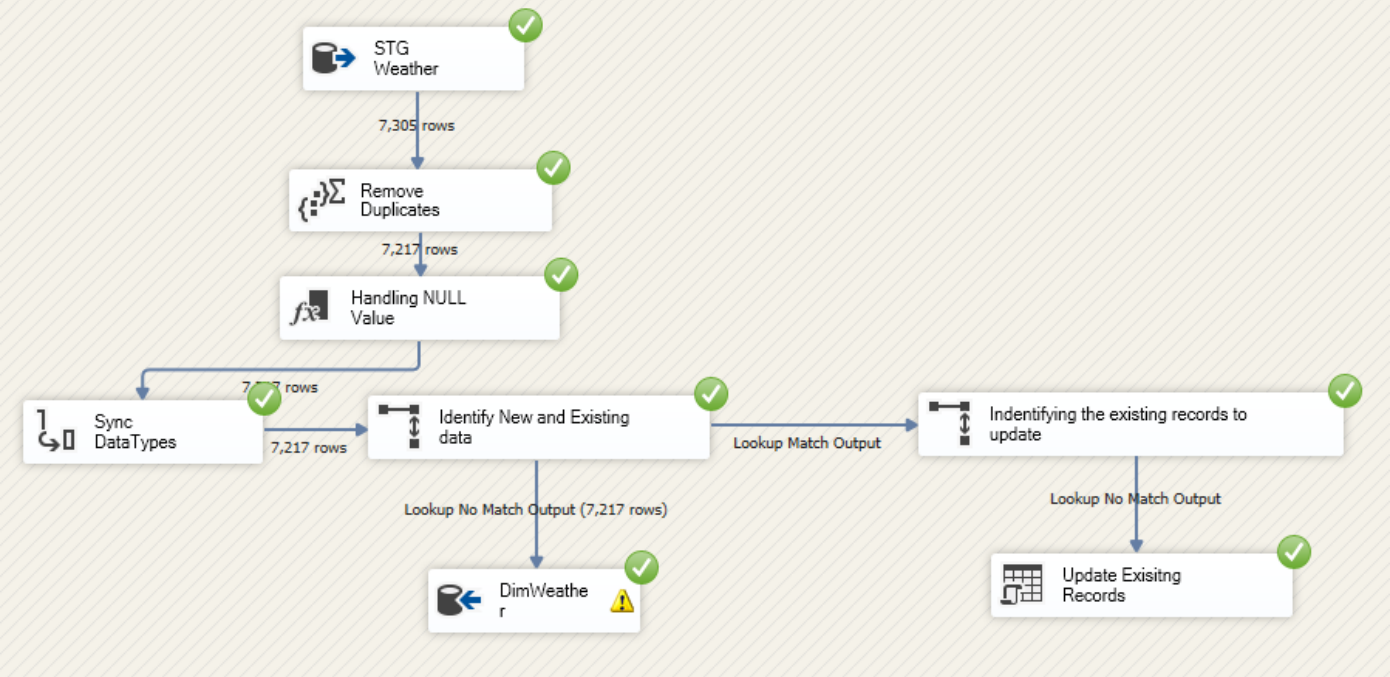




2.1.2.2 DIMENSION [X]

For each dimension [X], please specify stages of your approach and provided detailed description of each stage.

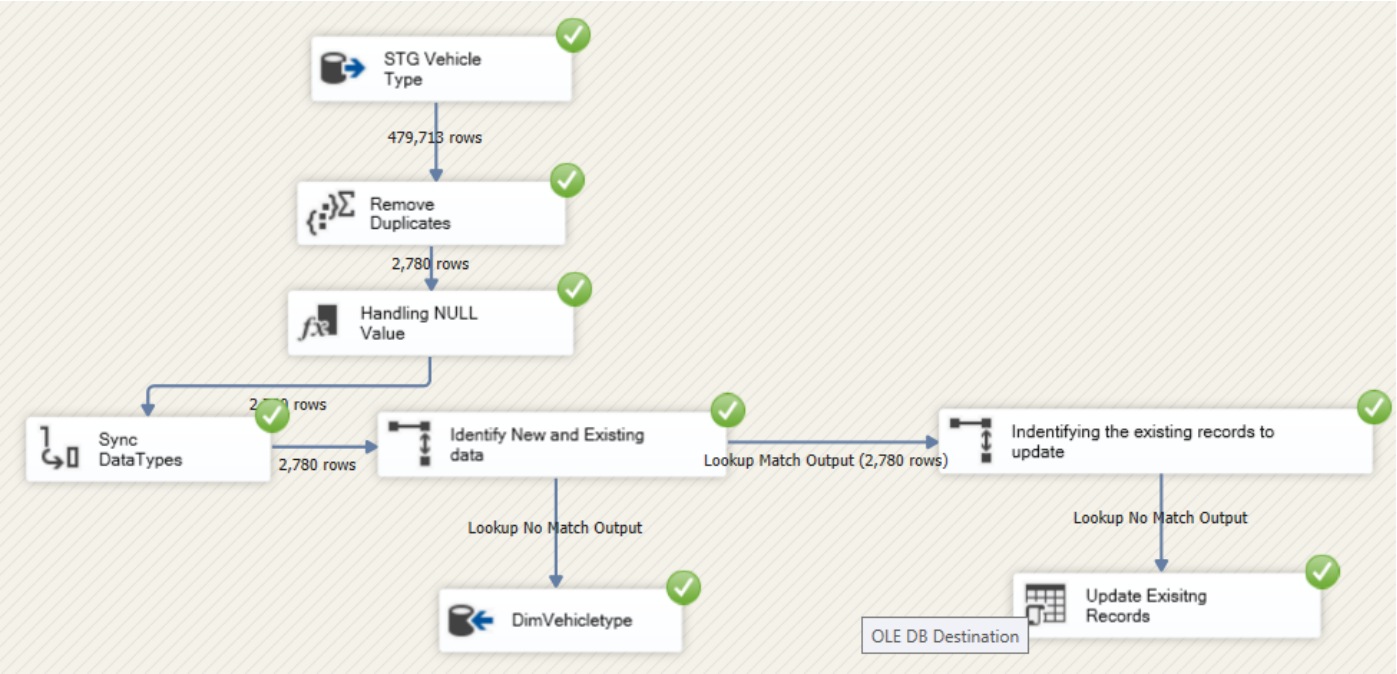
DimWeather Table



The above image is representation of dimWeather dimension within the ETL process for data warehouse population. It begins with extracting the weather data from staging area, compromising 7.305 rows.

The initial step is removing the duplicates from the data to be sure each weather record is unique and contributes accurately to analysis. After that although I made the data cleaning using py, to be ensure handling the null values step added. Than syncing data types across the dataset standardizes formats and be sure to compatibility with target schema. The process than identifies new weather records and determines existing ones through a lookup transformation. So new records are identifies from the staging area are directed to insert paths in the dimWeather dimension ensuring all data is captures and stored. For existing records, a lookup transformation identifies matches based on keys, that allows efficient updates without duplication for later.

DimVehicleType Table

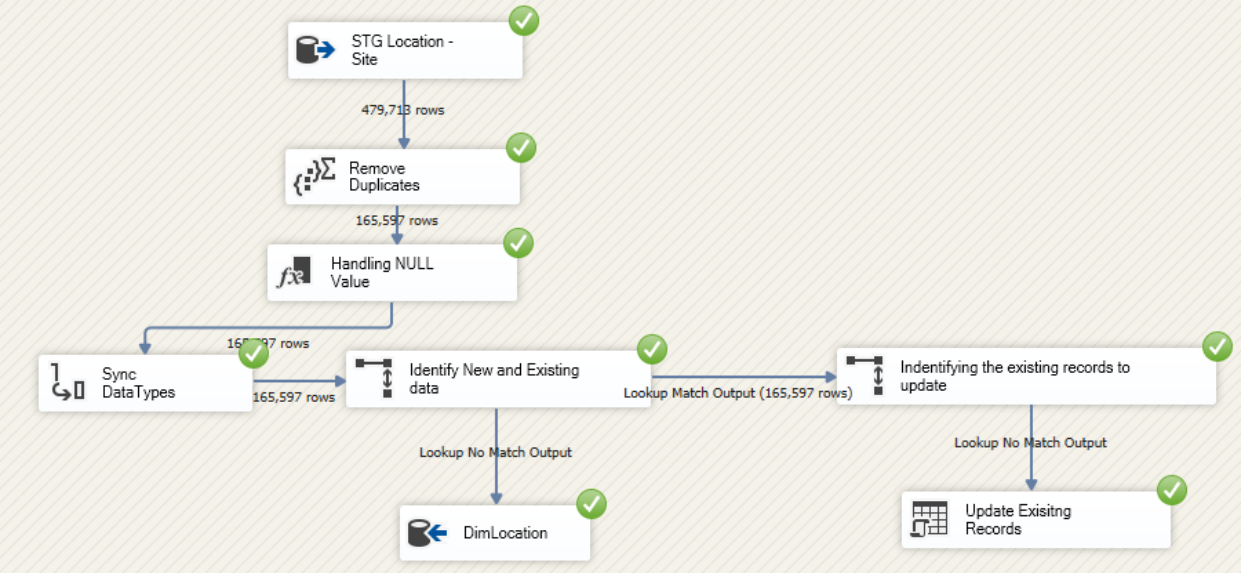


The above image represents the creation of the DimVehicleType dimension within the ETL process for data warehouse population. It begins with extracting vehicle type data from the staging area (STgVehicleTypes), comprising 479,713 rows.

The initial step is removing the duplicates from the data to be sure each Vehicle record is unique and contributes accurately to analysis resulting in 2.780 rows. Following this, handling null values, Syncing data types across the dataset standardizes formats, ensuring compatibility with the target schema, which facilitates seamless integration with other dimensions and fact tables.

The process then identifies new vehicle type records and determines existing ones through a lookup transformation. New records identified from the staging area are directed to insert paths in the DimVehicleType dimension, ensuring all relevant data is captured and stored. For existing records, a second lookup transformation identifies matches based on predefined keys, allowing for efficient updates without duplication for later.

DimLocation Table



Since the processes for these 3 dimension are about the same, Very similar approach for creation of these dimensions are followed. The above image represents the creation of the DimLocation dimension within the ETL process for data warehouse population. It begins with extracting location site data from the staging area (STgLocationSite), comprising 479,713 rows.

The initial step is removing the duplicates from the data to be sure each location record is unique and accurately contributes to the analysis. After duplicate removal, handling null values is critical, reducing the dataset to 165,597 rows. After that Syncing data types across the dataset standardizes the formats, ensuring compatibility with the target schema.The process then identifies new location records and determines existing ones through a lookup transformation. New records identified from the staging area are directed to insert paths in the DimLocation dimension, ensuring all relevant data is captured and stored.For existing records, a second lookup transformation identifies matches based on predefined keys, allowing for efficient updates without duplication. This step optimizes the ETL process by selectively updating existing records in the dimension and maintaining historical accuracy.

CrashDate and CrashTime tables

dimCrashDate and dimCrashTime dimensions were created to enable detailed temporal analysis of crash data. The 'dimCrashDate' dimension captures each date from 2016 to 2019, including like day, month and year. The 'dimCrashTime' includes every minute of the day with flags for day or night. These tables are populated once and reused because time dimension doesn't change over time. Once the data for all possible times in a day is populated than it remains constant. it's ensures the effecienty and comprehensive time based reporting and analysis. This one-time setup simplifies for us to maintenance and enhances the ability to perform granular data analysis.

Below tables are for creation of both dimCrashDate and dimCrashTime.

```
--use VehicleandWeather;
-- Create the table
DROP TABLE IF EXISTS dbo.dimCrashTime
CREATE TABLE dbo.dimCrashTime (
    TimeID INT Identity(1,1)PRIMARY KEY,
    CrashTime VARCHAR(5) NOT NULL,
    [Hour] INT NOT NULL,
    [Minute] INT NOT NULL,
    isDay TINYINT NOT NULL,
    isNight TINYINT NOT NULL
);

-- Declare the variables for looping through hours and minutes
DECLARE @Hour INT = 0;
DECLARE @Minute INT = 0;
DECLARE @TimeID INT;
DECLARE @CrashTime VARCHAR(5);

-- Loop through each hour of the day
WHILE @Hour < 24
BEGIN
    -- Loop through each minute of the hour
    WHILE @Minute < 60
    BEGIN
        -- Format TimeID as HHMM with leading zeros
        SET @TimeID = CAST(RIGHT('0' + CAST(@Hour AS VARCHAR(2)), 2) + RIGHT('0' + CAST(@Minute AS
VARCHAR(2)), 2) AS INT);

        -- Format CrashTime as HH:MM with leading zeros
        SET @CrashTime = RIGHT('0' + CAST(@Hour AS VARCHAR(2)), 2) + ':' + RIGHT('0' + CAST(@Minute
AS VARCHAR(2)), 2);

        -- Determine if it is day (6 AM to 6 PM) or night (6 PM to 6 AM)
        IF @Hour >= 6 AND @Hour < 18
        BEGIN
            INSERT INTO dbo.dimCrashTime ( CrashTime, [Hour], [Minute], isDay, isNight)
            VALUES ( @CrashTime, @Hour, @Minute, 1, 0);
        END
        ELSE
        BEGIN
```

```
            INSERT INTO dbo.dimCrashTime ( CrashTime, [Hour], [Minute], isDay, isNight)
            VALUES ( @CrashTime, @Hour, @Minute, 0, 1);
        END

        -- Increment the minute
        SET @Minute = @Minute + 1;
    END

    -- Reset the minute and increment the hour
    SET @Minute = 0;
    SET @Hour = @Hour + 1;
END;




--crashdate

DROP TABLE IF EXISTS dbo.dimCrashDate
CREATE TABLE dbo.dimCrashDate (
    DateID INT PRIMARY KEY,
    CrashDate DATE,
    TheDay INT,
    TheDayName NVARCHAR(50),
    TheMonthName NVARCHAR(50),
    TheYear INT
);
-- Declare the variables for looping through the dates
DECLARE @StartDate DATE = '2016-01-01';
DECLARE @EndDate DATE = '2019-12-31';
DECLARE @CurrentDate DATE = @StartDate;

-- Loop through each date from the start date to the end date
WHILE @CurrentDate <= @EndDate
BEGIN
    INSERT INTO dbo.dimCrashDate (
        DateID,
        CrashDate,
        TheDay,
        TheDayName,
        TheMonthName,
        TheYear
    )
    VALUES (
        CAST(CONVERT(VARCHAR, @CurrentDate, 112) AS INT), -- Convert date to YYYYMMDD format
        @CurrentDate,
        DAY(@CurrentDate),
        DATENAME(WEEKDAY, @CurrentDate),
        DATENAME(MONTH, @CurrentDate),
        YEAR(@CurrentDate)
    );

    -- Increment the date
    SET @CurrentDate = DATEADD(DAY, 1, @CurrentDate);
END;
```

2.1.2.3 FACT [X]

The fact table centralizes information from various dimensions; Weather conditions, vehicletypes, locations, crash dates and times providing dataset for collision analysis.

Since I didn't create a WeatherDatetime dimension from the weather file, but asked to populate it in ETL process, the procedure designed to integrate weather data with collision records although I don't have datetime dimension in the schema. By joining the

weather attributes with dimensions like dimWeather and DimLocation, it enables analysis of collisions patterns influenced by specific date of weather conditions in these cities.

As requested in the requirements part, to enhance the performance and manage large datasets efficiently, data is loaded into the staging area and subsequently into the VehicleCollisionStaging table in batches of 10.000 rows. Indexes are created on temporary tables like VehicleCollisionStaging to improve query performance during data processing. The stored procedure dbo.spInsertFactCollision coordinates this process. It first retrieves vehiclecollision records into VehicleCollisionstaging, ensuring only new data is processed. And than it populates 'Collisions' by joining staged data with dimension tables to map and aggreagate attributes. Unique id are generated for each collision event.

To populate FactCollisions, only records absent from the existing table are selected into InsertCollision, using LEFT JOIN operations to filter duplicates. Via this filtering we are able to ensure that only unique collision events are inserted into final FactCollisions.

Finally batches of 10.000 rows from InsertCollision are then inserted into FactCollisions, a process designed to uphold both performance efficiency and transactional consistency throughout the data loading phase.

spInserFactCollision query shared below.

```
use VehicleandWeather;
GO


CREATE PROCEDURE dbo.spInsertFactCollision
AS

BEGIN


	--Getting all vehicle collisions records from stage into #VehicleCollisionStaging


	IF OBJECT_ID ('tempdb..#VehicleCollisionStaging','U') IS NOT NULL
		DROP TABLE #VehicleCollisionStaging

	SELECT *
	INTO   #VehicleCollisionStaging
	FROM   [stg].[VehicleCollisionStaging]

	CREATE ColumnStore INDEX TempCollision
    ON #VehicleCollisionStaging
    ([CRASHDATE],[CRASHTIME], [BOROUGH],[ZIPCODE], [ONSTREETNAME],[CROSSSTREETNAME],
[OFFSTREETNAME],
     [NUMBEROFPERSONSINJURED],[NUMBEROFPEDESTRIANSINJURED], [NUMBEROFPEDESTRIANSKILLED],
[NUMBEROFCYCLISTINJURED],
     [NUMBEROFCYCLISTKILLED], [NUMBEROFMOTORISTINJURED], [NUMBEROFMOTORISTKILLED],
     [CONTRIBUTINGFACTORVEHICLE1],[VEHICLETYPECODE1]
    )



	--Getting all vehicle collisions records to populate  FactCollision


	IF OBJECT_ID ('tempdb..#COllisions','U') IS NOT NULL
		DROP TABLE #COllisions

	SELECT   NEWID() UNID,
		      CD.[DateID]
		     ,CT.[TimeID]
		     ,DL.[LocationID]
		     ,DV.[VehicleTypeID]
		     ,DW.[weatherID]
```

```
                    ,C.[NUMBEROFPERSONSINJURED]    AS [InjuredPerson]
                    ,C.[NUMBEROFPERSONSKILLED]     AS [KilledPerson]
                    ,C.[NUMBEROFPEDESTRIANSINJURED]AS [PedestriansInjured]
                    ,C.[NUMBEROFPEDESTRIANSKILLED] AS [PedestriansKilled]
                    ,C.[NUMBEROFCYCLISTINJURED]    AS [CyclistInjured]
                    ,C.[NUMBEROFCYCLISTKILLED]     AS [CyclistKilled]
                    ,C.[NUMBEROFMOTORISTINJURED]   AS [MotoristInjured]
                    ,C.[NUMBEROFMOTORISTKILLED]    AS [MotoristKilled]
       INTO #COllisions
       FROM #VehicleCollisionStaging C
       JOIN [stg].[WeatherStaging] W WITH(NOLOCK) ON CONVERT(DATE,C.Crashdate)= CASE      WHEN
CHARINDEX('-', Datetime) = 5 -- yyyy-MM-dd format

                                                    THEN CAST(Datetime AS DATE)

                                                WHEN CHARINDEX('-', Datetime) = 3 -- dd-MM-yyyy
format

                                                    THEN CAST(SUBSTRING(Datetime, 7, 4) + '-' +

                                                                SUBSTRING(Datetime, 4,
2) + '-' +

                                                                SUBSTRING(Datetime, 1,
2) AS DATE) END
       JOIN dbo.DimWeather DW WITH(NOLOCK) ON DW.[MaxTemperature]           = W.[tempmax]
                                                                       AND
DW.[MinTemperature] = W.[tempmin]
                                                                       AND DW.[humidity]
             = W.[humidity]
                                                                       AND
DW.[windspeed]              = W.[windspeed]
                                                                       AND
DW.[visibility]             = W.[visibility]
                                                                       AND
DW.[description]     = W.[description]

       JOIN dbo.DimLocation DL WITH(NOLOCK) ON  CONVERT(VARCHAR(50),DL.[borough])=
CONVERT(VARCHAR(50),C.[BOROUGH])
                                                                  AND
CONVERT(VARCHAR(50),DL.[zipcode])= CONVERT(VARCHAR(50),C.[ZIPCODE])
                                                                  AND (REPLACE
(Dl.[ONSTREETNAME]+ DL.[CROSSSTREETNAME]+DL.[OFFSTREETNAME] ,'Unknown',''')=
                                                                     REPLACE
(C.[ONSTREETNAME]+ C.[CROSSSTREETNAME]+C.[OFFSTREETNAME] ,'Unknown','')
                                                                     )
       JOIN [dbo].[DimVehicleType] DV WITH(NOLOCK) ON  DV.[VehicleType]                =
C.[VEHICLETYPECODE1]
                                                                       AND
DV.[ContributingFactortype] = C.[CONTRIBUTINGFACTORVEHICLE1]
       JOIN dbo.DImCrashdate             CD WITH(NOLOCK) ON  CONVERT(DATE,CD.Crashdate)  =
CONVERT(DATE,C.Crashdate)
       JOIN dbo.DimCrashtime             CT WITH(NOLOCK) ON  CONVERT(TIME(0),CT.CrashTime)
       = CONVERT(TIME(0),C.[CRASHTIME])

       GROUP BY  CD.[DateID]
                    ,CT.[TimeID]
                    ,DL.[LocationID]
                    ,DV.[VehicleTypeID]
                    ,DW.[WeatherID]
                     ,C.[NUMBEROFPERSONSINJURED]
                     ,C.[NUMBEROFPERSONSKILLED]
                     ,C.[NUMBEROFPEDESTRIANSINJURED]
                     ,C.[NUMBEROFPEDESTRIANSKILLED]
                     ,C.[NUMBEROFCYCLISTINJURED]
                     ,C.[NUMBEROFCYCLISTKILLED]
```

```
                        ,C.[NUMBEROFMOTORISTINJURED]
                        ,C.[NUMBEROFMOTORISTKILLED]



        --Getting only those records that are not present already in FactCollision


        IF OBJECT_ID ('tempdb..#INsertCollision','U') IS NOT NULL
                DROP TABLE #INsertCollision


        SELECT          C.UNID,
                                C.[DateID],C.[TimeID],C.[LocationID],C.[VehicleTypeID],C.[WeatherID],

        C.[InjuredPerson],C.[KilledPerson],C.[PedestriansInjured],C.[PedestriansKilled],

        C.[CyclistInjured],C.[CyclistKilled],C.[MotoristInjured],C.[MotoristKilled]
        INTO #INsertCollision
        FROM #Collisions C
        LEFT JOIN dbo.FactCollisions  FC ON FC. [CrashDateID]                   = C.[DateID]
                                                                        AND FC.[CrashTimeID]        =
C.[TimeID]
                                                                        AND FC.[LocationID]
        = C.[LocationID]
                                                                        AND FC.[VehicleTypeID]
        = C.[VehicleTypeID]
                                                                        AND FC.[WeatherID]
        = C.[WeatherID]
                                                                        AND FC.[InjuredPerson]
        = C.[InjuredPerson]
                                                                        AND FC.[KilledPerson]
        = C.[KilledPerson]
                                                                        AND FC.[PedestriansInjured] =
C.[PedestriansInjured]
                                                                        AND FC.[PedestriansKilled]  =
C.[PedestriansKilled]
                                                                        AND FC.[CyclistInjured]
        = C.[CyclistInjured]
                                                                        AND FC.[CyclistKilled]
        = C.[CyclistKilled]
                                                                        AND FC.[MotoristInjured]    =
C.[MotoristInjured]
                                                                        AND FC.[MotoristKilled]
        = C.[MotoristKilled]

        WHERE FC.[CrashDateID] IS NULL

        CREATE COLUMNSTORE INDEX CollisionCus ON
#Collisions([DateID],[TimeID],[LocationID],[VehicleTypeID],[WeatherID],
                                [InjuredPerson],[KilledPerson],[PedestriansInjured],[PedestriansKilled],
                                [CyclistInjured],[CyclistKilled],[MotoristInjured],[MotoristKilled]  )
        CREATE CLUSTERED INDEX Cltr ON  #Collisions(UNID)

        --INSERT in batch of 10000 FactCollision


        --SELECT COUNT(*) FROM #Collisions
        DECLARE @i INT = 0;
    DECLARE @batchSize INT = 10000;
    DECLARE @totalCount INT;
        SELECT  @totalCount = COUNT(1)FROM #Collisions

        WHILE (@i < @totalCount)
        BEGIN
```

```
            Print 'Batch Starts'

            INSERT INTO [dbo].[FactCollisions]([CrashDateID], [CrashTimeID], [LocationID],
[VehicleTypeID], [WeatherID]
                                                                        , [InjuredPerson],
[KilledPerson], [PedestriansInjured]
                                                                        , [PedestriansKilled],
[CyclistInjured]
                                                                        , [CyclistKilled],
[MotoristInjured], [MotoristKilled])
            SELECT
                        C.[DateID],C.[TimeID],C.[LocationID],C.[VehicleTypeID],C.[WeatherID],

      C.[InjuredPerson],C.[KilledPerson],C.[PedestriansInjured],C.[PedestriansKilled],

      C.[CyclistInjured],C.[CyclistKilled],C.[MotoristInjured],C.[MotoristKilled]
            FROM #INsertCollision C
            ORDER BY UNID
            OFFSET @i ROWS
            FETCH NEXT @batchSize ROWS ONLY

       SET @i = @i + @batchSize;
            --Print'Batch End '
            --PRINT @i PRINT @batchSize

      END --End of While loop

END --End of procedure
```

TASK 2.2 – CUBE MODEL – DETAILS:

Please prepare the model of the ETL process, in accordance with the below specification and discuss the solution with the lecturer.

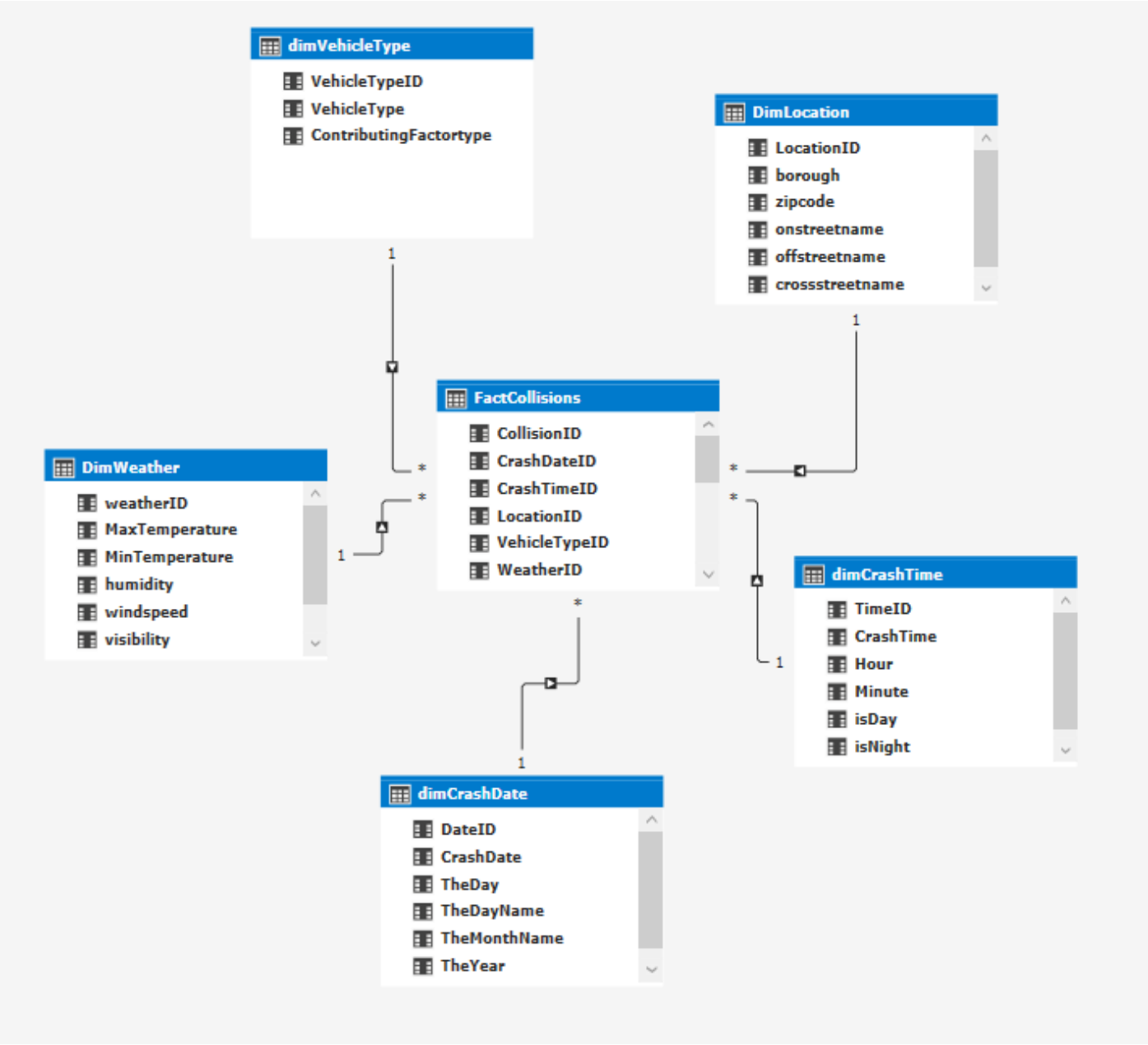
General note: ***The complexity and completeness of the created cube affects the grade. During ongoing lab assignments and during lecture we have tackled different properties and settings related to proper OLAP cube definition and creation – try to use them and try to use them wisely.***

TASK 2.2 – CUBE IN SSAS – SOLUTIONS:

1. Prepare a cube based on prepared data from the previous task.
    a. Remember to define needed dimensions, attributes, attribute relations, hierarchies, etc. Remember about proper ordering of members within the dimensions – e.g., January comes before February
    b. Define needed measures and aggregation functions, try including at least one calculated measure
    c. Additionally define (*required for 85%*): at least a single perspective, at least one KPI and proper aggregations (cube materialisations)
2. Process and deploy your cube in your local SSAS instance
    a. Be able to show and document your results in cube's browser

As the final solution, please upload your source files. In the final report provide a concise description of your implementation in accordance with the following points:

2.2.1 FINAL CUBE STRUCTURE



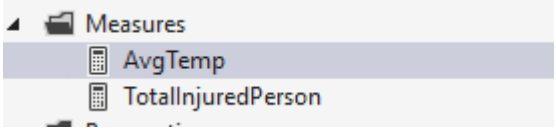
2.2.2 MEASURES

Prepare a short description and a screenshot for all measures (include information about formatting string and aggregation function).

AvgTemp: Calculating the average of the maximum temperatures recorded. It uses the aggregation function AVERAGE and is formatted as a number with two decimal places.

SumInjuredPerson: Computes the total number of injured persons. It uses the aggregation function SUM and is formatted as an integer without decimal places.

2 example measure is created from the DimWeather and FactCollisions tables below. Together with the formula is shared below.



Properties

TotalInjuredPerson Measure

Advanced	
Display Folder	
Basic	
Decimal places	2
Description	
Format	Decimal Number
Formula	(SUM(FactCollisions[InjuredPerson]))
Measure Name	TotalInjuredPerson
Show Thousand Separator	False
Misc	
Id	TotalInjuredPerson:FactCollisions

Properties

AvgTemp Measure

Advanced	
Display Folder	
Basic	
Description	
Format	Decimal Number
Formula	AVERAGE(DimWeather[MaxTemperature])
Measure Name	AvgTemp
Misc	
Id	AvgTemp:DimWeather

```
AvgTemp:=AVERAGE(DimWeather[MaxTemperature])
```

AvgTemp:
20.7570320077595

```
TotalInjuredPerson:=(SUM(FactCollisions[InjuredPerson]))
```

TotalInjuredPerson
774659

2.2.3 DIMENSIONS

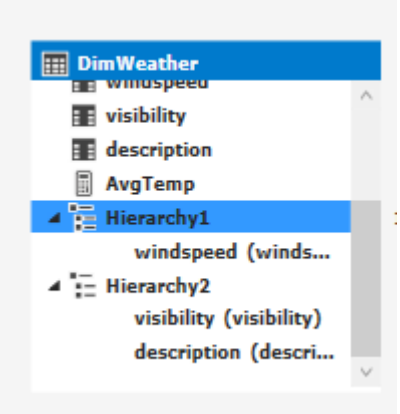
DimWeather: Captures weather conditions, organized by temperature metrics and descriptions.

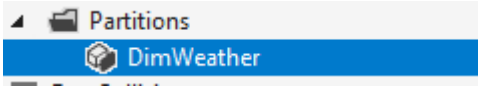
DimWeather Table

Basic	
Connection Name	SqlServer DESKTOP-JJHQLTC\MSSQLSERVER_LAB VehicleandWeather
Formula	
Hidden	False
Partitions	(Click to edit)
Source Data	(Click to edit)
Table Description	
Table Name	DimWeather
Misc	
Id	DimWeather
Reporting Properties	
Table Behavior	(Click to edit)

[MaxTemperature]

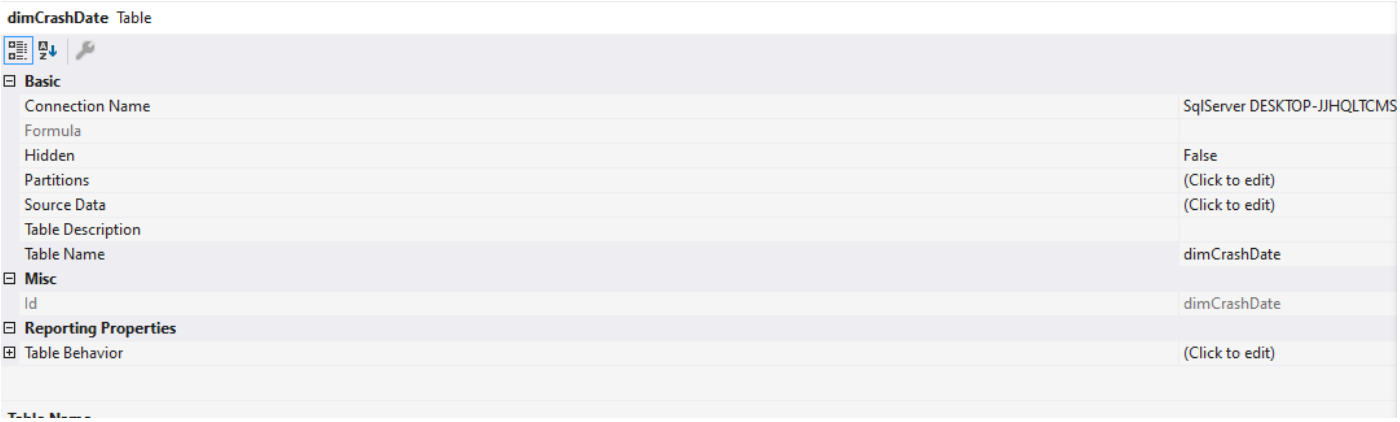
	weat...	MaxTemperature	MinTemperature	humidity	windspeed	visibility	description
1	51	12.8	3.6	50	26.6	16	Partly cloudy...
2	164	2.6	-2.4	50	36.4	16	Partly cloudy...
3	171	24	17.7	50	18.3	16	Partly cloudy...
4	290	27.8	20.1	50	20.1	16	Partly cloudy...
5	378	4.4	-0.8	50	17.4	16	Partly cloudy...
6	391	6.6	0	50	34.4	16	Partly cloudy...

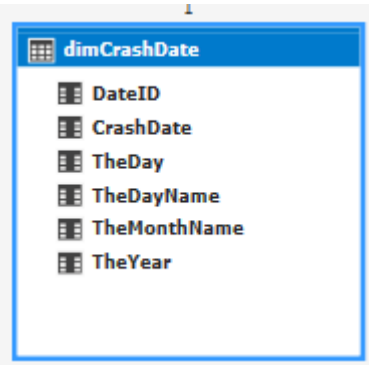




Created hierarchies also presented in above table.

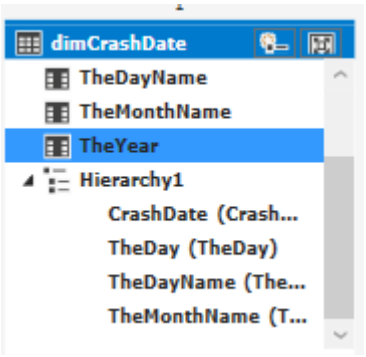
DimCrashDate: Includes hierarchical levels such as Year, Month, Day, and Weekday. Attribute relations ensure proper sorting and grouping of dates.



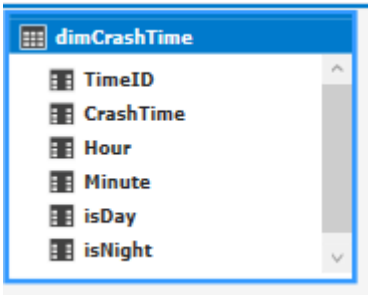


[CrashDate] | 1/5/2016 12:00:00 AM

	DateID	CrashDate	TheDay	TheDayName	TheMonthName	TheYear
1	20160101	1/1/2016 12...	1	Friday	January	2016
2	20160102	1/2/2016 12...	2	Saturday	January	2016
3	20160103	1/3/2016 12...	3	Sunday	January	2016
4	20160104	1/4/2016 12...	4	Monday	January	2016
5	20160105	1/5/2016 12...	5	Tuesday	January	2016
6	20160106	1/6/2016 12...	6	Wednesday	January	2016
7	20160107	1/7/2016 12...	7	Thursday	January	2016
8	20160108	1/8/2016 12...	8	Friday	January	2016
9	20160109	1/9/2016 12...	9	Saturday	January	2016
10	20160110	1/10/2016 1...	10	Sunday	January	2016



DimCrashTime: Organized by hours and minutes, facilitating time-based analysis of collisions.



Data Warehouses - Report MP - 2

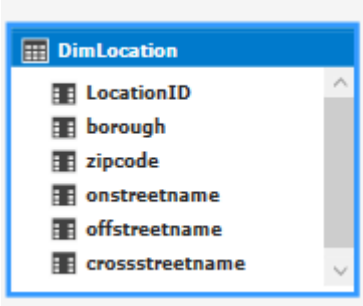
[TimeID]

	T...	CrashTime	Hour	Minute	isDay	isNight	Ad
1	361	06:00	6	0	1	0	
2	362	06:01	6	1	1	0	
3	363	06:02	6	2	1	0	
4	364	06:03	6	3	1	0	
5	365	06:04	6	4	1	0	
6	366	06:05	6	5	1	0	

DimLocation: Contains geographical data, hierarchically structured by Borough, ZIP code, and specific address.

DimLocation Table

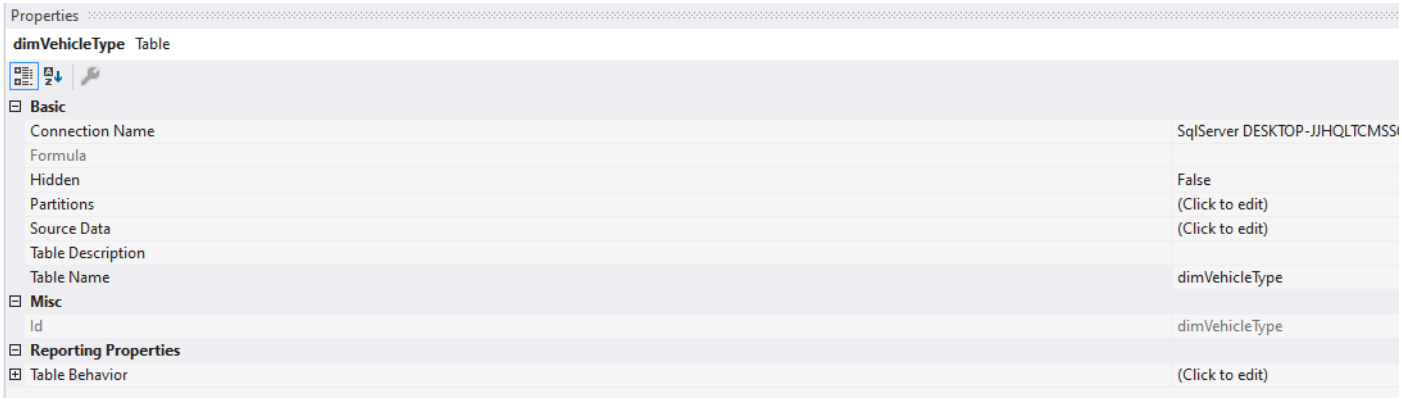
Basic	
Connection Name	SqlServer DESKTOP-JJHQLT
Formula	
Hidden	False
Partitions	(Click to edit)
Source Data	(Click to edit)
Table Description	
Table Name	DimLocation
Misc	
Id	DimLocation
Reporting Properties	
Table Behavior	(Click to edit)

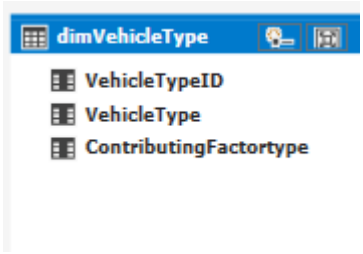


[LocationID]

	Loca...	borough	zipcode	onstreetname	offstreetname	crossstreetname	Add
1	50	BROOKLYN	11207	Unknown	2094 PITKIN ...	Unknown	
2	247	BROOKLYN	11207	Unknown	559 MILLER ...	Unknown	
3	298	BROOKLYN	11207	Unknown	442 HEGEM...	Unknown	
4	382	BROOKLYN	11207	Unknown	419 BRADF...	Unknown	
5	427	BROOKLYN	11207	Unknown	625 LIVONI...	Unknown	
6	577	BROOKLYN	11207	Unknown	901 ASHFO...	Unknown	
7	731	BROOKLYN	11207	Unknown	752 NEW JE...	Unknown	
8	880	BROOKLYN	11207	Unknown	310 HIGHLA...	Unknown	
9	963	BROOKLYN	11207	Unknown	153 SCHENC...	Unknown	
10	1031	BROOKLYN	11207	Unknown	601 NEW LO...	Unknown	

dimVehicleType: Categorized by different types of vehicles involved in collisions.



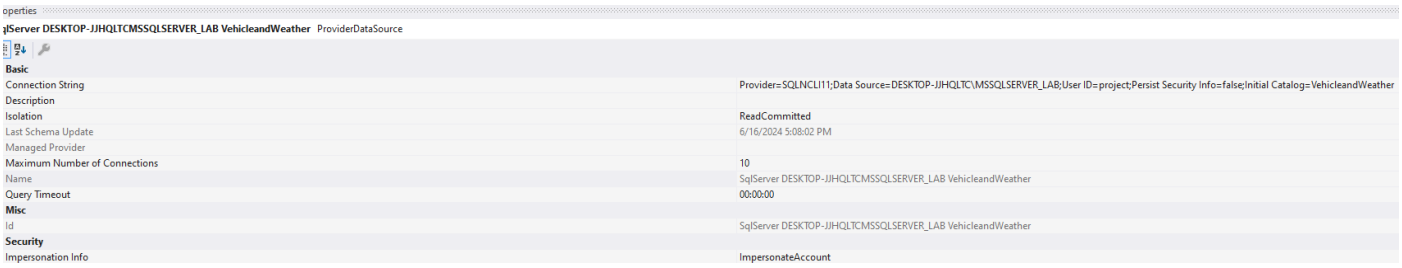


[VehicleTypeID]

	VehicleTypeID	VehicleType	ContributingFactortype
1	5	Fire	Unknown
2	25	Minicycle	Unknown
3	35	1	Unknown
4	50	TRANS	Unknown
5	51	SELF	Unknown
6	55	Tow t	Unknown
7	56	FIRE TRUCK	Unknown
8	58	FEDEX	Unknown
9	69	Van Camper	Unknown
10	70	Tanker	Unknown

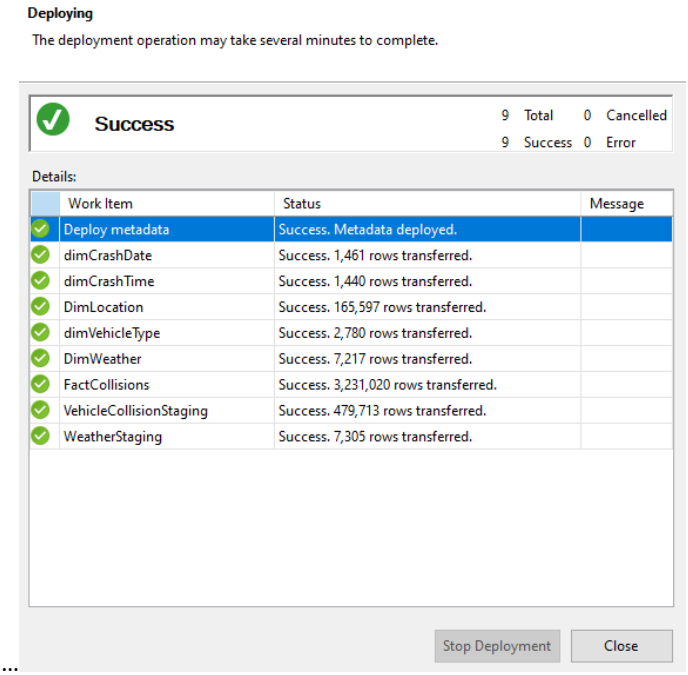
2.2.4 CUBE DETAILS

Prepare a short description and a screenshot for all additional mechanisms utilised, e.g., calculations, KPIs, aggregations, partitions, perspectives.

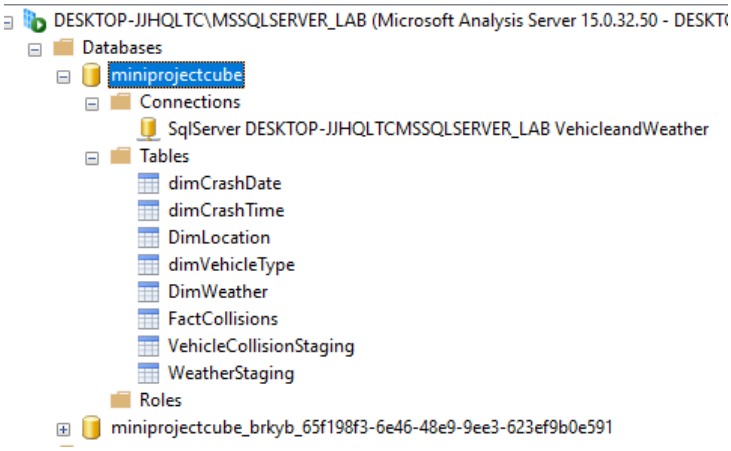


2.2.5 PROCESS RESULTS

Prepare a screenshot from successful completion of cube processing and deployment process, and a screenshot from cube browser (with an exemplar query of your choice – make sure to capture cube's structure).



...



GENERAL CONCLUSIONS:

Use this section to provide your general conclusions:

ETL process in Visual Studio for the person who doesn't have experience about, is one of the time taking and problematic tasks in my opinion. Before I started to stage2 documentation I thought finding relevant data would be the hardest part till I faced with the issues designing the ETL. My collision data were already not in good quality to be able to work on it. A lot of null values for the important columns and unnecessary column existed. Since I am not familiar with this tool, debugging and reading errors were complicated than I expected. However, at the end I believe that project successfully implemented for ETL process and the schema for relevant analyzing vehicle collisions and weather data. By leveraging SSIS for ETL and creating a star schema with dimension and fact tables, the solution handles the historical and incremental data loads.