In our project, **Spar Nord Bank** is trying to observe the withdrawal behavior and the corresponding dependent factors to optimally manage the refill frequency. Apart from this, other insights also have to be drawn from the data.

Coming to the analysis part, you will be tasked to carry out the calculations to perform the following analytical queries:

- Top 10 ATMs where most transactions are in the ’inactive’ state
- Number of ATM failures corresponding to the different weather conditions recorded at the time of the transactions
- Top 10 ATMs with the most number of transactions throughout the year
- Number of overall ATM transactions going inactive per month for each month
- Top 10 ATMs with the highest total amount withdrawn throughout the year
- Number of failed ATM transactions across various card types
- Top 10 records with the number of transactions ordered by the ATM_number, ATM_manufacturer, location, weekend_flag and then total_transaction_count, on weekdays and on weekends throughout the year
- Most active day in each ATMs from location "Vejgaard"

Your overall task in this project will be to build a batch ETL pipeline to read transactional data from RDS, transform and load it into target dimensions and facts on Redshift **Data Mart(Schema)**.

Please note that the source data and target schema details are provided to better understand the source and targets, which would help design the ETL pipeline. Once the data is loaded into Redshift, you would have to write the analytical queries discussed above.

We have data from more than 100 ATMs across Denmark. Data is captured for every transaction including, card type, location, date, time, ATM type, etc.

Also, the transaction amount field in the data set was added separately using a random function for the analysis purpose.

Spar Nord Bank has also built a dimensional model datastore (**ATM Data Mart**) on this ATM transaction data to understand the ATM usage pattern. This exact schema(target schema) of this Data Mart will be provided to you for the sake of this project. Basically, this will be the schema according to which you will be creating tables in Redshift.

Broadly you will be performing the following task-

- Extracting the transactional data from a given MySQL RDS server to HDFS(EC2) instance using Sqoop.

- Transforming the transactional data according to the given target schema using PySpark.
- This transformed data is to be loaded to an S3 bucket.
- Creating the Redshift tables according to the given schema.
- Loading the data from Amazon S3 to Redshift tables.
- Performing the analysis queries.

In the next segment, you will learn more about the data set, which you will be dealing with in this project.

Additional Content

- If you want to get clarity on the concepts of **Dimension Model** and **Data Mart,** you can refer to the relevant segments in the **Data Warehousing and ETL** module. Anyways the Dimensional Model(target schema) to be used in this project will be provided to you.
- Data Marts - A data mart is a simple form of a data warehouse that is focused on a single subject (or functional area), such as Sales or Finance or Marketing.
- Dimensional modelling - Dimensional modelling includes a set of methods, techniques and concepts for use in data warehouse design.

Data Set

This dataset comprises around **2.5 million records** of withdrawal data along with weather information at the time of the transactions from around **113 ATMs from the year 2017**.

The actual data set is divided into **two part files**, both amounting to about **503** MB in total.

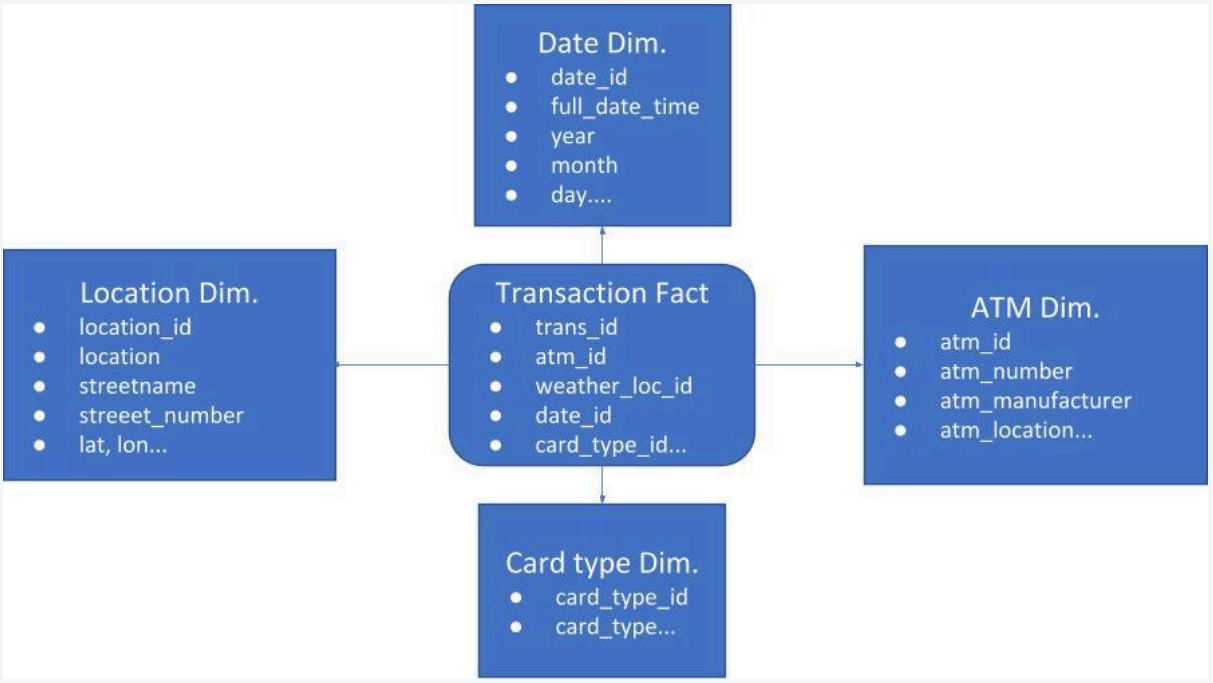
There is also a data dictionary present along with the data set, which defines all of the **33 columns** present in the data set. The data dictionary is given below.

This data set contains various types of transactional data as well as the weather data at the time of the transaction, such as:

- **Transaction Date and Time**: Year, month, day, weekday, hour
- **Status of the ATM**: Active or inactive
- **Details of the ATM**: ATM ID, manufacturer name along with location details such as longitude, latitude, street name, street number and zip code
- **The weather of the area near the ATM during the transaction**: Location of measurement such as longitude, latitude, city name along with the type of weather, temperature, pressure, wind speed, cloud and so on
- **Transaction details:** Card type, currency, transaction/service type, transaction amount and error message (if any)

Target Dimension Model

Now, in this segment, you will get to know about the **target schema(dimension model)** according to which you will have to transform the data set using Spark as well as create and load it accordingly in RedShift. Simply put this will be the schema according to which you'll create the tables in Redshift.
As discussed in the video, you will be following the provided target dimension model schema for this ETL project.



Target Dimension Model

For this project, you will need four dimension tables and one fact table. They are as follows:

- **ATM dimension** - This dimension will have the data related to the various ATMs present in the dataset along with the ATM number(ATM ID in the original dataset), ATM manufacturer and a reference to the ATM location

and is very important for solving analytical queries related where ATM data will be used.

- **Location dimension** - This is a very important dimension containing all the location data including location name, street name, street number, zip code and even the latitude and longitude. This information will be very important for solving problems related to the particular location at which a transaction took place and can help banks in things like pinpointing locations where ATMs where demand is higher as compared to other locations. Combined with weather data in the transaction table, this can be used to further do analysis such as how weather affects the demand at ATMs at a particular location.
- **Date dimension** - This is another very important dimension which is almost always present where data such as transactional data is being dealt with. This dimension includes fields such as the full date and time timestamp, year, month, day, hour as well as the weekday for a transaction. This all can help in analysing the transaction behaviour with respect to the time at which the transaction took place and also how the transaction activity varies between weekdays and weekends.
- **Card type dimension** - This dimension has the information about the particular card type with which a particular transaction took place. This can help in performing analysis on how the number of transactions varies with respect to each different card type.

- **Transaction fact** - This is the actual fact table for the data set which contains all of the numerical data such as the currency of the transaction, service, transaction amount, message code and text as well as weather info such as description, weather id etc.

PySpark

- **Reading the data from the files in HDFS by a specific schema using PySpark**
    - Command to create an input schema using **StructType**(We recommend you to create a custom schema using the StructType class of PySpark, to avoid any data type mismatch.)
    - Commands to read the data using the input schema created and verifying the data using the count function
- **Creation of dimension tables using PySpark**
    - Command to create a data frame for the dimension according to the target schema(dimension model) provided
    - Commands to clean and transform the data:
        - Making sure that duplicate records are cleaned(Avoid duplicate info especially in the dimension tables.
        - Making sure that appropriate primary keys are present for the dimensions(You need to generate a primary key for each dimension table. For example for the '**Date**' dimension one way

to generate the primary key can be by adding "date" as the prefix to the row number i.e. 'date1', 'date2' and so on.)

        - Rearranging the fields if necessary(According to the target schema)

Note: Here, the tasks given above have to be done for all four dimension tables.

- **Creation of transaction fact table using PySpark**
    - Commands to set proper alias for the various **PySpark DataFrames** before proceeding with creating the fact table (**optional**)
    - Commands for various stages where the original data frame is appropriately joined with the dimension tables created above
    - Commands to clean and transform the data:
        - Making sure that the appropriate primary key is present for the fact table
        - Rearranging the fields if necessary
- **Loading the dimension and fact tables into Amazon S3 bucket**
    - Write the DataFrames containing the dimensions and fact table directly to an S3 bucket folder. [You should create different folders on your S3 bucket for different dimensions and fact table.]

Redshift

- **Creation of a Redshift Cluster**
    - You need to create a Redshift cluster in the same way as it was done in the **Amazon Redshift** module.
- **Setting up a database in the Redshift cluster and running queries to create the dimension and fact tables**
    - Queries to create the various dimension and fact tables with appropriate primary and foreign keys
- **Loading data into a Redshift cluster from Amazon S3 bucket**
    - Queries to **copy** the data from S3 buckets to the Redshift cluster in the appropriate tables
- **Using queries on a Redshift cluster to find the solution to the following analytical queries.**
    - Top 10 ATMs where most transactions are in the ’inactive’ state
    - Number of ATM failures corresponding to the different weather conditions recorded at the time of the transactions
    - Top 10 ATMs with the most number of transactions throughout the year
    - Number of overall ATM transactions going inactive per month for each month
    - Top 10 ATMs with the highest total amount withdrawn throughout the year
    - Number of failed ATM transactions across various card types

- Top 10 records with the number of transactions ordered by the ATM_number, ATM_manufacturer, location, weekend_flag and then total_transaction_count, on weekdays and on weekends throughout the year
- Most active day in each ATMs from location "Vejgaard"

Additional Resources

In this segment, you will learn some concepts in Spark SQL, such as Window function, Partition and row_number, and you will understand how to use them. These concepts will help you while working on the project.

row_number() - Spark SQL

During the creation of Dimension tables, you will have to add a row number/index to each table so that a primary key can be established.

Spark SQL provides a function called **row_number()** as part of the **window** functions group, which can be used to assign a sequential integer number to each row in a DataFrame according to the partition decided.

Let's assume that you have a dataset of employee information of a certain company. You have a dataframe called **emp_df** having the following data:

emp_name	dept	salary
Abhishek	Sales	1000
Kumar	Legal	2000
Saif	Finance	1200
Singh	HR	1500

You can use the following code to add another column, having sequential numbers as values, to the DataFrame:

```
from pyspark.sql.types import *

from pyspark.sql.window import Window

import pyspark.sql.functions as F

from pyspark.sql.functions import row_number
```

```
res_df =
emp_df.select("emp_name","dept","salary",F.row_number().over(Window.partitionBy().orderBy(res_df['emp_name'])).alias("index"))
```

This code snippet will essentially add another column at the end of the table called **index**, which will have sequential integer values. The final DataFrame would look as follows:

emp_name	dept	salary	index
Abhishek	Sales	1000	1
Kumar	Legal	2000	2
Saif	Finance	1200	3
Singh	HR	1500	4

Using this method, you will be able to create a unique value for each row of your dimension tables.