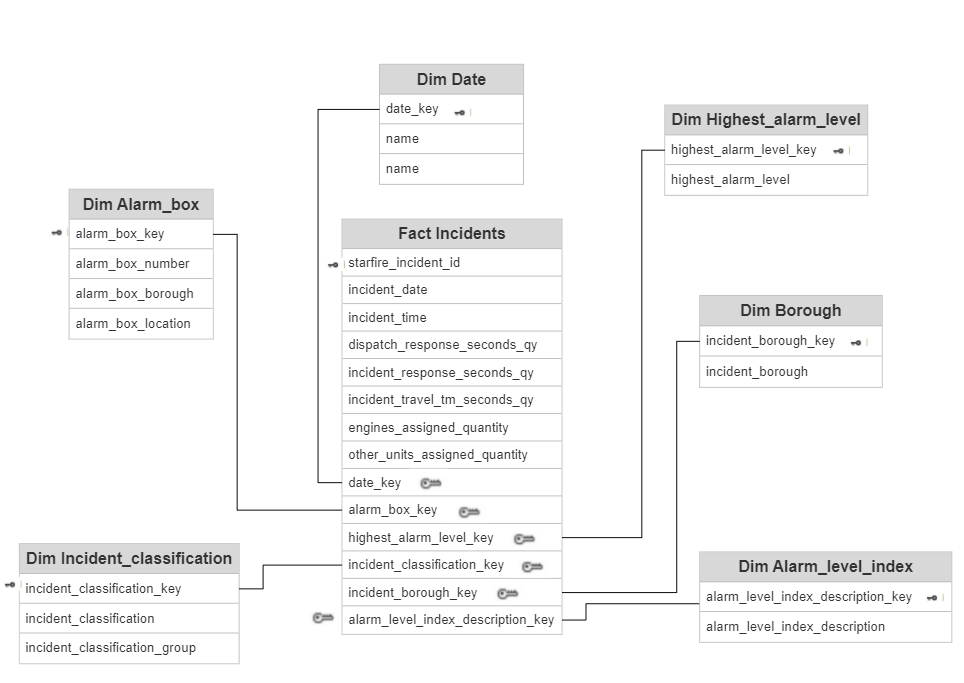
Data model document:

ERD & architecture:

The data set provided was very wide (more than 20 columns) and unnormalized (e.g., Incident borough was   
repeating as well as Highest\_alarm\_level and so on).  
Given that a few normalization (NF) rules were not considered, I decided to break down  
the table to several dim tables and a fact table that will focus on incident times.  
Hence, I decided to follow this ERD architecture, which, in my opinion, could help minimize the complexity  
of the fact table and could apply efficient querying and analyzing down the road.

Dim & Fact tables:

Worth mentioning that beside the dim tables that were extracted out the raw data set,

I decided to add a dim date table, script can be found in "Dim Date" file.

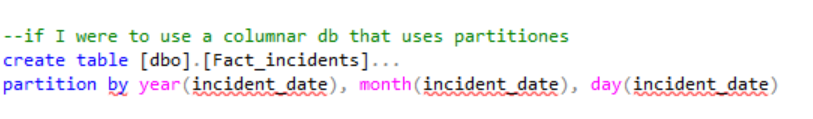
PK are blacked facing left and FK are greyed focusing right.  
I choose SQL Server as my DB for the mission and the queries attached were written in SSMS.

Dim tables were extracted from the raw table and I decided to create a surrogate key per each   
dim table as can be seen in the file "Creating dim tables".

The fact table was created similarly, by using an SQL query to extract additive fields from the raw table and   
joining the key fields with the dim tables.   
I was able to get the keys from each dim table as can be seen in "Creating fact table".

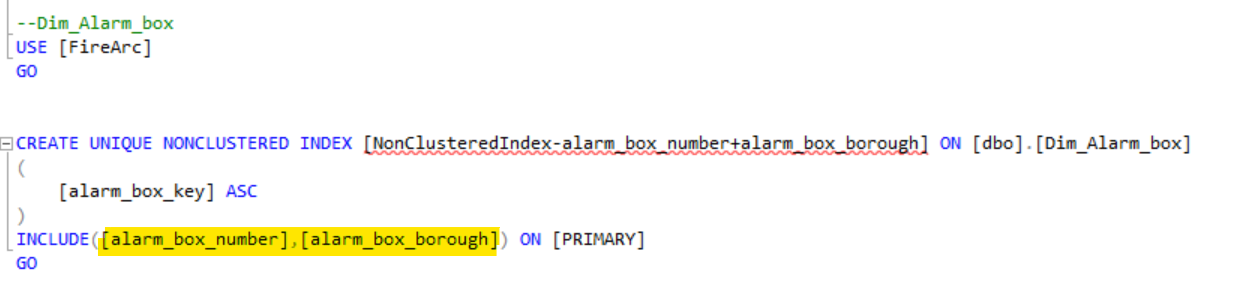
Each dim table received an identity int key to serve as a surrogate key,   
which is then used as a foreign key (FK) in the fact table, as can be shown in the ERD above.

Indexes & Partitions:

Since I used SQL Server applying partition was quite a problem as it requires knowledge in  
figuring out files and functions that looks out of the scope of this project, hence I assumed it  
is possible to assume that I could show what I would do for partition wise if I had columnar DB:  
I decided to apply partitioning to the fact table based on the date field since it's likely to be   
heavily filtered and frequently used in queries.

Regarding indexes, before applying any indexes, I cloned all of the tables (Dim + Fact) in order to be able  
to test efficiency for clustered tables and non-clustered.  
SQL Server automatically applied clustered indexes when I created the primary key (PK) in the create clause,   
so I didn't have to worry about them.  
All of the tables (Dim + Fact) holds clustered index on the surrogate key which is a PK,   
so best practice is applied as I found useful in my opinion upon on-line research.

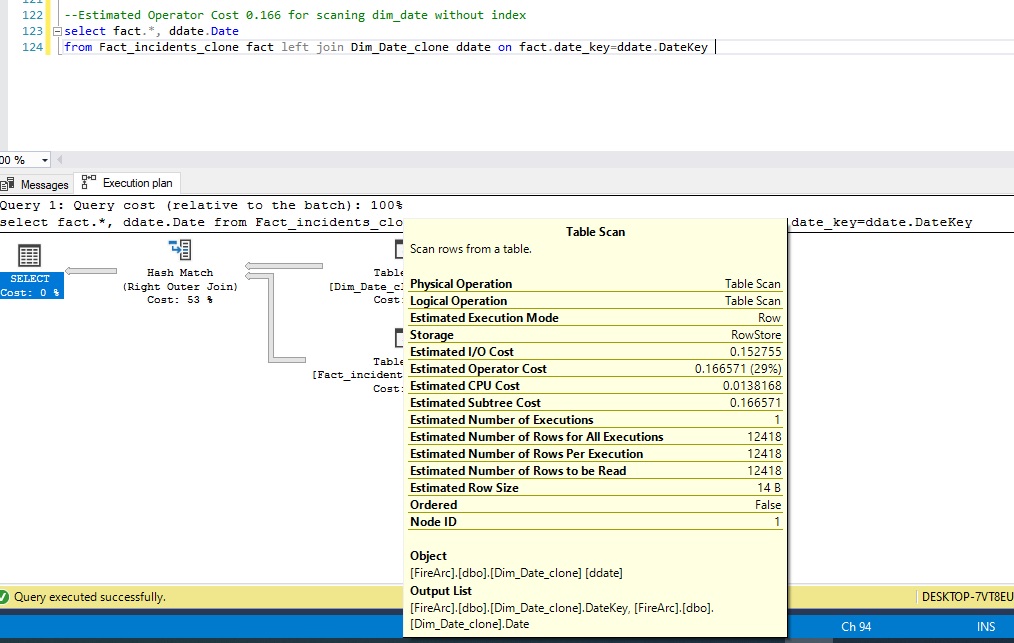
To follow that research, I applied non clustered index on the business keys in each dim table,   
for example in table [Dim\_Date], Date field is now clustered (non-clustered index).  
In some cases, two fields needed to be includedas non-clustered index because only the   
combination of both make uniqueness.



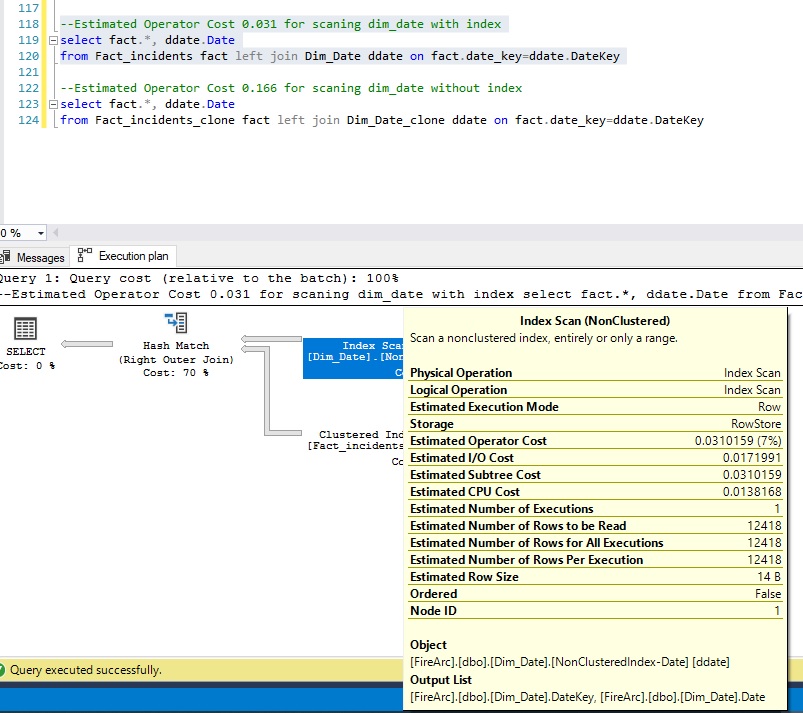
2 Fields for the index

When come to demonstrate the efficiency of the index's I applied some comparison between the  
indexed tables and the cloned table, all can be seen on "Creating and testing index's"  
The main character that was decided to demonstrate the efficiency is   
Operator Cost of SQL server execution plan.

For example:



No index- clone table



index- original table