**SQL STRUCTURE:**

Till höger har vi våran dimensionella modell som ett snowflake-diagram. Vi har här 6 dimensioner. Time, location, weather\_description, unit, precipitation\_category och cloud\_cover\_description, med weather\_data som vår fact-table.

I mitten syns en bild på databasens tabeller. Den enda som inte kan refereras till via ID är unit, som istället måste refereras till direkt om man vill ha den med.

Jag försökte till vänster göra en tydlig hierarkisk modell över hur vår sql\_functions.py fungerar.

Funktionen som ankallas via dag är **write\_sql**, som kallar på tre funktioner. **Initialize\_db** som initialiserar databasen och skapar strukturen och alla tabeller innan vi sätter in värden i våran fact-table.   
**Create\_main\_dataframe** som skapar en dataframe från alla filer, tar bort units från vår dataframe, konverterar tidsformatet, vilket krävdes då json, pandas och sqlalchemy hanterar datetime-objekt annorlunda.  
**date\_table** som skapar en dataframe för vår date-table.

**INITIALIZE DB**

Vi kör en funktion för att initialisera databasen. **I initialize\_db** kontrollerar funktionen om tabellerna finns och om de inte finns ännu så skapas de.

**Date\_table** är ett exempel på hur en av dessa funktioner kan se ut, även om den aldrig behöver initialiseras.

Vi förbereder i prepare dataframes-exemplet dictionaries som en lista av id:n och värden för att enkelt kunna skapa dataframes att skicka till databasen.

Vi deklarerar även konstanter högst upp i filen, saker som används regelbundet är SQLPATH, CLEANEDPATH och ROOTPATH som är dynamiskt skrivna för att programmet skall fungera var det än ligger.

**WRITE SQL**

På **write\_sql** ligger funktionerna write\_sql och create\_main\_dataframe.

Write\_sql initialiserar databasen och skickar in värden i den.

Create\_main\_dataframe ser alla json-filer i cleansed-mappen och kontrollerar om de har fått en insert ännu genom en loggfil och hoppar i så fall över dem. Sedan gör den datahanteringen som krävs innan den returnerar en dataframe.

**AIRFLOW DAG & DOCKER**

Till höger högst upp syns endast en liten del utav docker-compose filen för att köra projektet. I detta fall är det postgres-instansen vi använder som vårat data-warehouse. Den öppnar port 5325 utåt, och exponerar 5432 till andra containers.

**Initialize-project.sh** strax under Skapar alla undermappar, sätter rättigheter, environment variables och kör docker-compose.

I vår DAG-fil importeras först alla dependencies, här är geocode en extra del som tillsattes efter projektet var avslutat. Eftersom att vi har utvecklat hela projektet i en undermapp till /plugins som är en pythonpath satt i airflow-processen så blir det väldigt enkelt att importera alla funktioner.

Nedanför följer ett enkelt exempel, vi startar vår dag kl. 4 varje morgon och visar hur vi använder vår python-operator.

Och under detta deklareras följden vi kör funktionerna i.