Spotify End-to-end ETL Pipeline med Airflow och Snowflake

Namn	Steven Lomon Lennartsson
Utbildning	Pythonutvecklare med inriktning AI
Datum	5/31/2024

Abstract

This is a project within Data Engineering and my master thesis that was completed in just under three weeks. I have not done this for a company but it is rather completely my own personal project. This report aims to thoroughly describe the entire process, but also to explain all relevant terms and more specifically the 5 biggest and most encompassing concepts so that a 15-year-old can get a high-level understanding. The project covers the entire data life cycle from data extraction, data modeling, data transformations, and finally the loading of data into a data repository for further analysis and use within data visualization and Machine Learning. Some details were changed from the initial project description, but the result still shows how modern tools and techniques can be used to build a scalable and efficient ETL pipeline.

Sammanfattning

Det här är ett projekt inom Data Engineering och mitt examensarbete som utförts på just under tre veckor. Jag har inte gjort detta för ett företag utan det är helt mitt egna personliga projekt. Denna rapport syftar till att både genomförligt beskriva hela processen, men även förklara alla relevanta termer och specifikt de 5 största och övergripande koncepten så att en femtonåring ska ha en överblickande förståelse. Projektet omfattar hela livslängden för data; från dataextraktion, datamodellering, datatransformationer, och slutligen laddningen av data in i ett dataarkiv för vidare analys och användning inom datavisualisering och Machine Learning. Vissa detaljer ändrades från den initiella projektbeskrivningen men resultatet visar ändå hur man kan använda moderna verktyg och tekniker för att bygga en skalbar och effektiv ETL-pipeline.

Innehållsförteckning

1. Inledning

1.1 Bakgrund

Det här projektet föddes efter att jag under min andra LiA-period bestämde mig att jag vill bli Data Engineer och inte AI Engineer. Jag blev förvånad när jag sa det högt men fann mer och mer sanning i det ju mer jag satt i det; jag tycker att dataanalys, datavisualisering, att göra ML-modeller… ärligt talat är ganska tråkigt. Jag vill vara the provider of data istället för den som gör användbara och coola saker med datan. Jag vill ha konversationer med Data Scientists, Data Analysts och ML Engineers om hur de behöver sin data och bygga system som är underhållbara och skalbara så att de kan komma åt denna data snabbt och smidigt. Jag vill se till att dagens AI blir given data av hög kvalitet. Att ta chansen att göra mitt examensarbete inom Data Engineering blev därför ett ganska logiskt steg som dessutom ger mig något att visa i min portfolio när jag jobbar på egen hand mot att bli Data Engineer.

Jag älskar musik, jag älskar Spotify och förutom att använda det varje dag tar jag även varje dag ansiktliga steg mot att få ett jobb där. Med det här projektet får jag chansen och nöjet av att använda deras data och dessutom använda flera teknologier för första gången; PostgreSQL, Amazon S3, Apache Airflow (har använt ytterst ytterst lite) och Snowflake. Förutom detta har det även gett mig chansen att solidifiera grunderna jag lärt mig på Teknikhögskolan om datamodellering och databaser i ett praktiskt projekt.

1.2 Syfte

Detta projekt har ej gjorts som beställning för ett företag utan är endast för lärande syfte och samtidigt något praktiskt att ha i portfolion. Syftet med denna rapport är främst att dokumentera och genomgående beskriva hela projektprocessen men även att förklara de 5 mest relevanta termer så att en 15-åring skulle kunna förstå dem. Detta är dels för att jag skriver denna rapport och själv inte pluggar Data Engineering vilket betyder att folk som inte har en fot inom Data Engineering kommer att läsa det men dels tvingar detta även mig att förstå alla koncept och teknologier på en lägre nivå och solidifierar min förståelse ytterligare. Om det kan få en ung person att bli intresserad av tech är det en bonus! Förklaring för alla termer finns i 8.1: Terminologi.

1.3 Mål

Planen är omfattande och innehåller mer än vad man faktiskt gör på en arbetsplats men det är för att lära så mycket som möjligt. De första stegen är enbart för att skapa en miljö som liknar “den riktiga världen”. I sin helhet är det följande:

1. Samla in data. Planen är antingen helt från deras API eller en blandning av deras API och dataset från t.ex. Kaggle
2. Datamodellering. Skapa tabeller med primary keys och se hur allt hänger ihop med foreign keys. Här är planen att Lucidchart används

3. Ladda in datan i en databas i PostgreSQL
4. Skapa en dimension model. Reducera antalet tabeller från datamodelleringen och ge mer visualisering och dokumentation
5. ETL med Apache Airflow
   a. Extrahera en del av datan i Postgre till Amazon S3
   b. Transformera datan
   c. Ladda datan till Amazon Redshift

När pipeline:n är helt klar skulle den t.ex. kunna användas för att extrahera alla nya låtar som laddats upp på Spotify en fredag för att sedan t.ex. användas inom Machine Learning eller visualiseras på ett visuellt stimulerande sätt.

Här ska det läggas till att planen från början var att ladda in datan i en data warehouse i Amazon Redshift men detta ändrades under projektets gång till Snowflake istället. Detta är den enda ändringen i teknologier som skedde under projektets gång. Alla ändringar som skedde under projektets gång kommer att tas upp under rapportens gång.

1.4 Utvecklingsmiljö

I sin helhet har följande teknologier använts i detta projekt:

- Amazon EC2
- PostgreSQL
- Apache Airflow
- Amazon S3
- Snowflake

Min tanke med att under projektets gång byta från Amazon Redshift till Snowflake var följande: för detta allmänt lärande syfte resonerades det att det är mycket lättare och mer intuitivt att lära sig Snowflake. Redshift känns mer niche och nödvändigt att lära sig ifall man ska jobba på ett företag som använder Redshift och då får man specifik träning från företaget för de specifika områden och specifikationer som finns på företaget. Snowflake har omfattande dokumentation och kändes mer nybörjarvänligt. Det känns även mycket lättare att lära sig Redshift om man har erfarnhet från att arbeta med ett annat data warehouse som t.ex. Snowflake i sitt verktygsbälte.

Google Colab har använts över Jupyter Notebook helt av personlig preferens.

En beskrivning av alla dessa teknologier och även de verktyg som används under projektets gång finns i 8.2: Terminologi.

2. Kravlista

I och med att detta är ett helt personligt projekt finns ingen färdig kravlista. Istället togs följande krav fram:

- Autentisering och dataextration från Spotify's API
- Laddning av data, rå och transformerad, till Amazon S3
- Datamodellering och laddning av data till en PostgreSQL-databas
- Skapa en dimensional model på format star schema av datan som laddats till Postgre med facts tables och dimension tables
- Ladda datan från Amazon S3 till Snowflake med Snowpipe
- Automatisera hela ETL-processen med Apache Airflow som ska köras på en Amazon EC2 instance

3. Use cases

Att kunna extrahera, transformera och ladda stora mängder av data från en eller flera datakällor är en fundamental del av jobbet som en Data Engineer. Denna pipeline är byggd med principer inom Data Engineering i åtanke vilket betyder att pipeline:n som ska byggas lätt kan justeras för att ändra behov från en viss specifikation och principerna som använts kan även användas för att bygga en helt ny pipeline från grunden. Detta kan antingen vara genom att skala upp eller ner pipeline:n vilket görs möjligt med Airflow, S3 och Snowflake, eller genom att byta ut en eller flera komponenter helt.

I detta specifika projekt tar vi data om låtar vilket kanske inte har en särskilt use case inom Spotify som företag. Men om det instället vore användardata som extraherades skulle ett specifikt use case kunna vara att använda den data som lagras i rapporter och dashboards för att se till att beslut inom Spotify är datadrivna och baserat på den senaste informationen.

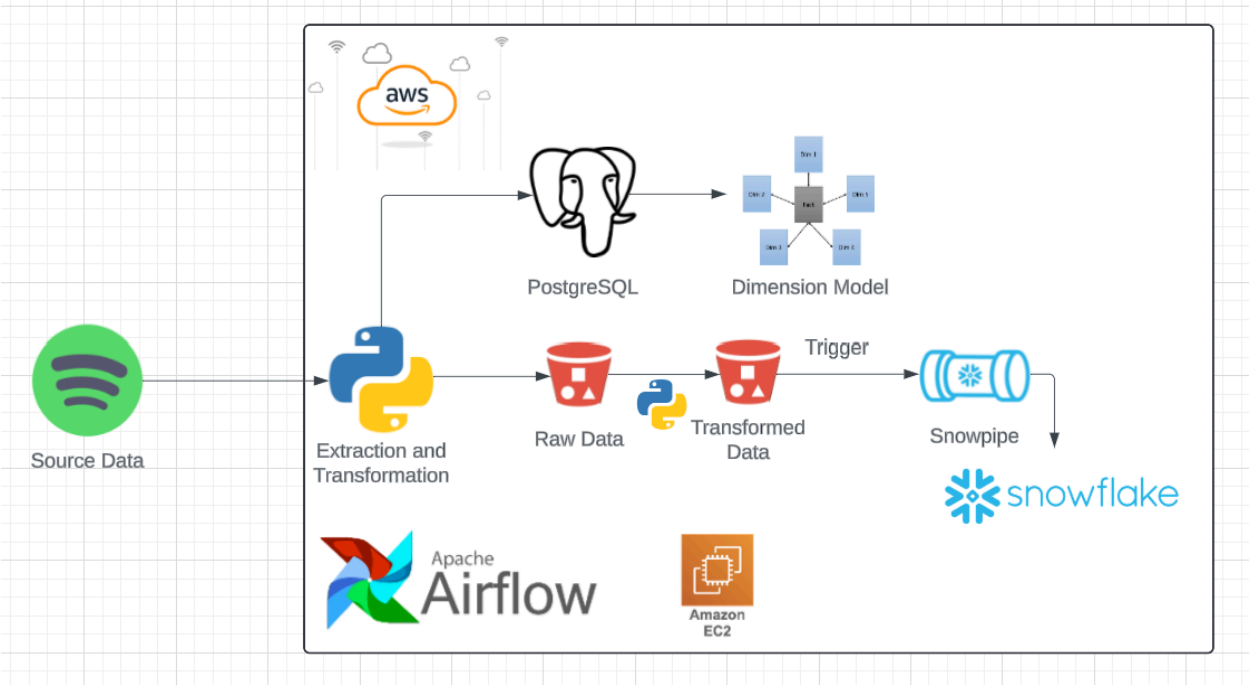
4. Tidsplanering

Detta projekt kommer att ge första kontakt med majoriteten av teknologier som kommer att användas och därav resoneras 20 dagar som en rimlig tidsplan för detta arbete. Planen är att jobba iterativt i sprintar nånting i följande stil:
Sprint 1: Datainsamling | 6 - 8 maj
Sprint 2: Datamodellering | 9 - 11 maj
Sprint 3: PostgreSQL | 12 - 14 maj
Sprint 4: Dimension Model | 15 - 18 maj
Sprint 5: ETL och sätta allt samman | 19 - 26 maj

5. Teknisk beskrivning

5.1 ETL-arkitektur

Projektet följer följande data-arkitektur:



figur 1 - Full ETL-arkitektur

Hela ETL-pipeline:n är automatiserad med Apache Airflow med hjälp av DAGs. Nu i det färdiga projektet kan den ställas in att extrahera data från Spotifys API varje dag. Mer om vilken data som extraheras från API:t i 6.1. Airflow körs på en virtuell maskin via Amazon EC2 instance för att få tillräckligt med beräkningskraft. Det andra rimliga valet som ett alternativ för Data orchestration istället för Airflow skulle vara Luigi. I och med att Spotify faktiskt har använt sig av Luigi skulle detta kunna vara ett logiskt val men Airflow är mycket mer använt och ses både som en guldstandard och ett bra första verktyg att använda för att gå vidare och testa på andra därifrån.

Datan som extraheras har två slutdestinationer vilket kommer tillbaka till det lärande syfte detta projekt har. Datan som laddas in i PostgreSQL och sedan modelleras till en Dimensional Model har ingenting med Airflow ETL:an att göra, detta är endast i lärande syfte och för att testa på saker man faktiskt ändå kan få i uppgift att göra för ett företag.

Valet att använda PostgreSQL över andra databaser har väldigt liknande resonering som Airflow. PostgreSQL är en guldstandard och en bra databas att kunna för att gå vidare och bli familjär med andra. Spotify använder i dagsläget Apache Cassandra. Precis som med Snowflake över Redshift: PostgreSQL är bättre att lära sig i lärande syfte som en allmän

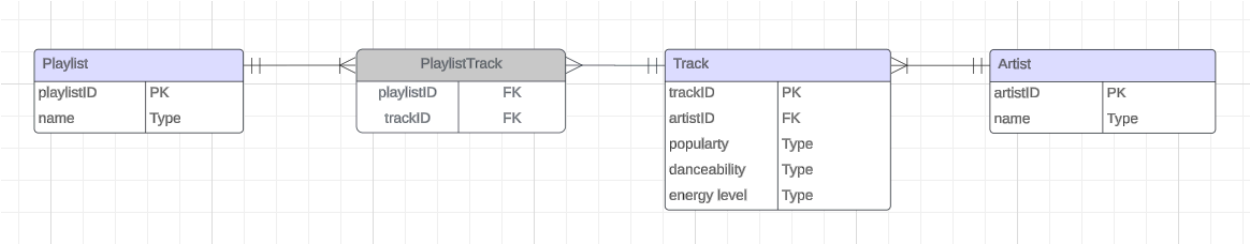
databas för att sedan dyka in i mer nischade databaser som Apache Cassandra ifall man kommer behöva använda det på ett jobb som Spotify.

I Airflow ETL:n läggs den extraherade datan från Spotifys API i två S3 buckets: en med rå data och en med transformerad data. När ny transformerad data i form av en csv-fil hamnar i sin S3 bucket aktiveras en trigger med hjälp av Snowpipe som kopierar datan med SQL och laddar in det i en data warehouse i Snowflake. I Snowflake skapades även en task för att se till att inga dubletter finns.

5.2 Datamodellering

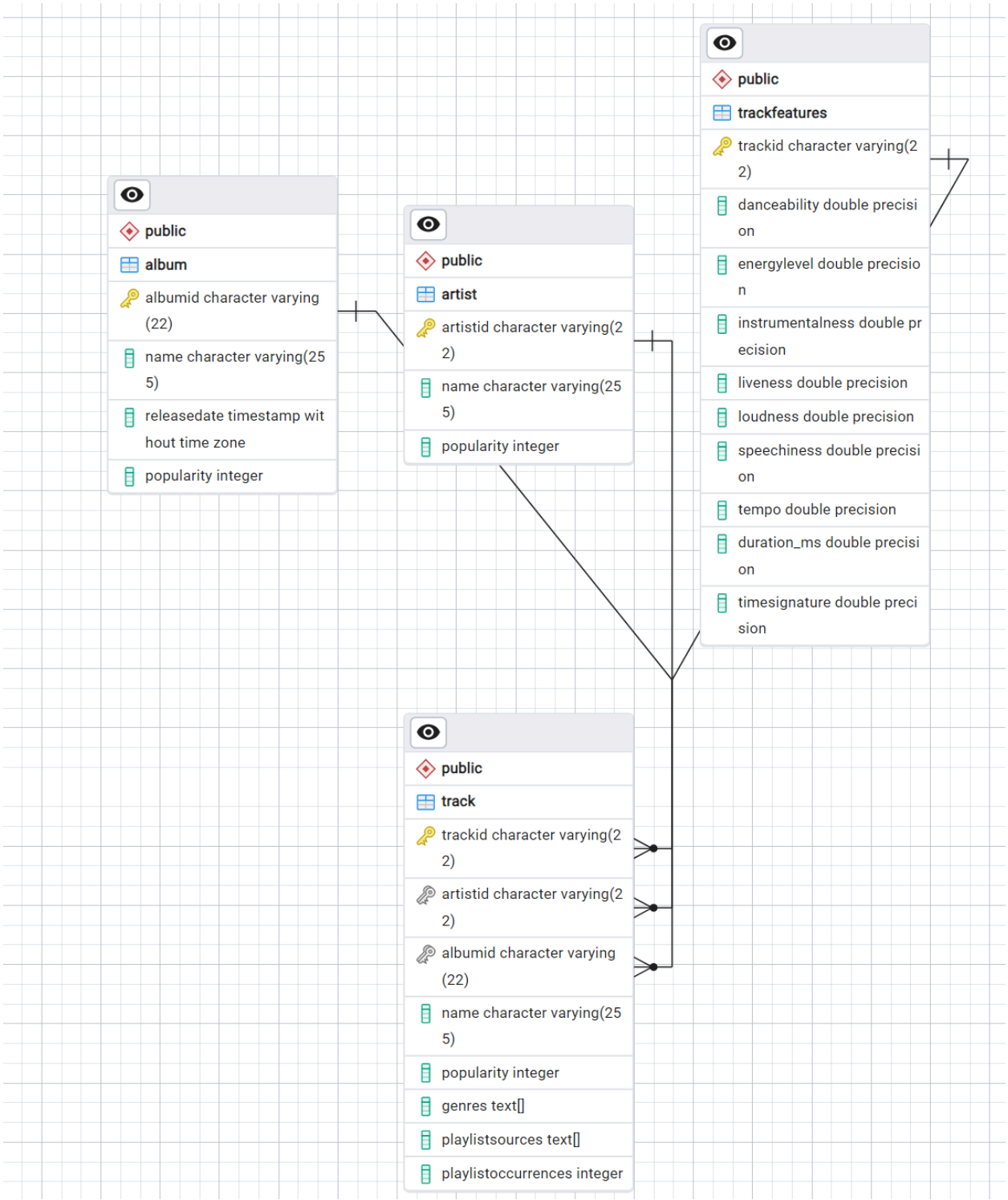
All datamodellering skedde i Lucidchart. Både standard och dimension model gick igenom flera iterationer under projektets gång. Först var tanken att tre tables skulle användas: Track, Artist och Playlist. En central fråga i början av projektet var “Är det Spotify eller min databas som modelleras?” Svaret är min databas och därifrån blev kardinaliteterna relativt självklara.

Iteration kom allt eftersom datan extraherades och transformerades. Mer avancerad extraktion och transformation gav upphov till mer avancerad datamodellering och tvärtom. Senare i projektets gång lades t.e.x. playlist sources och playlist occurrences till som kolumner vilket inte är med alls i första iterationen av datamodelleringen.



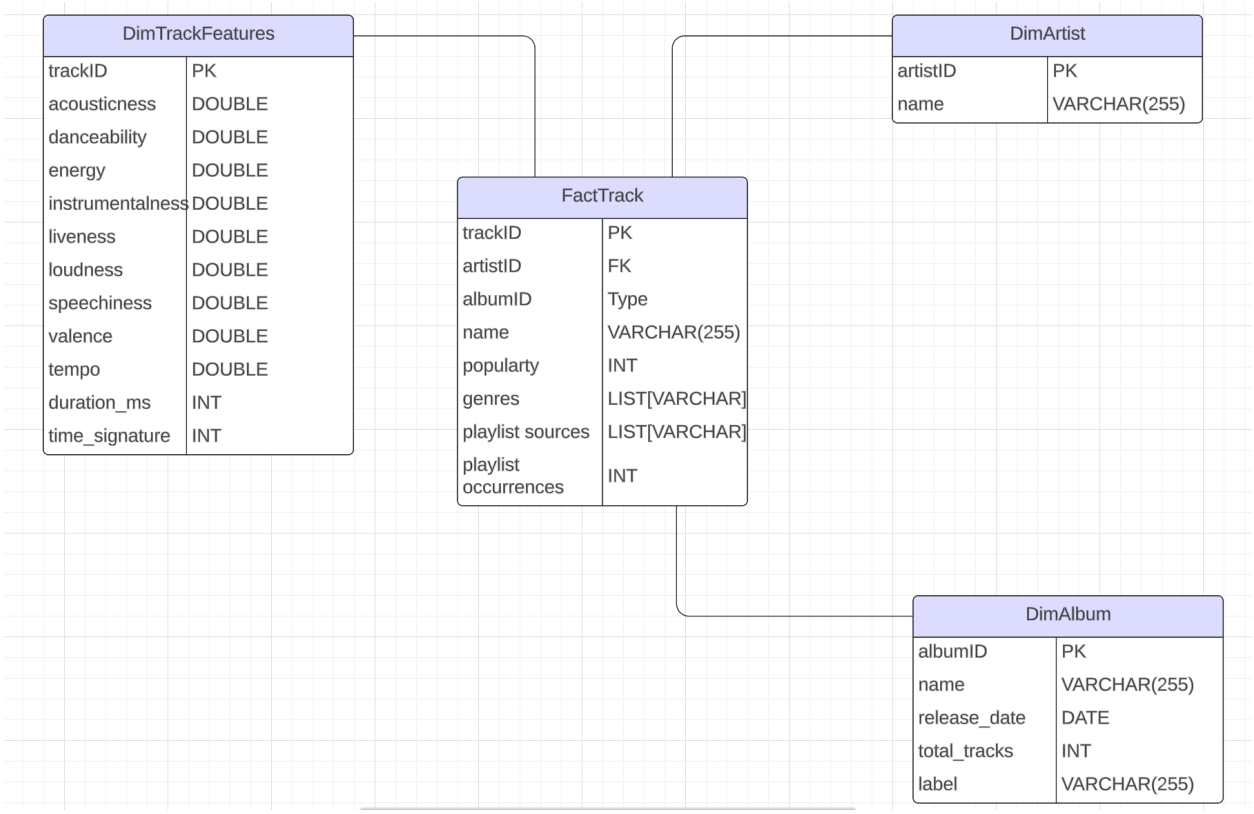
figur 2.1 - Den allra första iterationen av datamodellering för Spotify-datan. De två tabellerna till vänster användes till slut inte alls

Iterationen ledde till att det skapades två distinkta dataset; ett mer komplext och ett simplare. Mer om dessa i 6.1. Hur datamodelleringen för det komplexa datasetet översattes i PostgreSQL kan ses nedan:



figur 2.2 - Slutgiltiga ERD:n för databasen i PostgreSQL

När det var dags att modellera en dimension model efter det komplexare datasetet beslutades det att det fanns för få dimension tables vilket återigen påverkade hur datan extraherades och transformerades.



figur 2.3. - dimension model för den slutgiltiga kompexa datan i PostgreSQL

6. Implementering

6.1 Dataextraktion

Kaggle har inte använts alls utan data har endast extraherats från Spotifys API. Detta innebar initiellt att skapa en app i deras dashboard och att gå igenom en autentiseringprocess.

I början av projektet var tanken klar att hämta alla låtar från de 50 största spellistorna i Sverige och skapa ett dataset på ca 5000 låtar. Ytterligare kolumner med data skulle även extraheras från Spotifys audio feature API endpoint. Denna idé itererades på under projektets gång. Från början tog endast ‘energy level’ och ‘danceability’ men det slutade med att datan som laddades in i Postgre tog all data som går att extrahera från audio features för att ge mer data till den dimension table som sedan skapades.

Det allra största problemet som uppstod vid detta stadie var just audio features och dess rate limit vilket var mycket tydligare än andra endpoints. Denna jobbades omkring genom att bl.a. spara intermediate data i en JSON-fil för att bygga upp datasettet iterativt och fortsätta bygga upp pandas DataFrame:n från en specifik punkt. Detta utfördes dock endast framgånsrikt i Google Colab och inte i Airflow då det är två distinkta miljöer. Därav skapades behovet av det

andra simplare datasetet som används i Airflow. Så förutom ett komplext dataset med låtar från de 50 största spellistorna i Sverige skapades även ett simplare dataset där endast de 50 låtarna från Spotifys Global Top 50 extraherades. Det andra problemet som uppkom under projektets gång vilket ledde till behovet av ett simplare dataset skedde i skedet av datatransformation:

6.2 Datatransformation

Alla datatransformation utfördes med hjälp av Python-biblioteket pandas i Google Colab. Med två distinkta dataset kom två distinka kedjor av datatransformationer. Tidigt i projektet bestämdes det att dra nytta av att PostgreSQL kan hantera listor som datatyp genom att för det komplexa datasetet skapa två kolumner av denna typ: playlist sources och genres. Dessa är alltså listor med strängar.

Ett problem här är dock att när en DataFrame sparas som en csv konverteras all data till text vilket betyder att listor förlorar alla deras egenskaper och funktionalitet. Lösningen till detta blev att använda parquet-filformatet. Genom att använda denna behålls komplexa datatyper såsom listor även när de exporteras från en Notebook.

Det andra stora problemet under projektet var dock att parquet-filer inte visade sig vara kompatibla med Airflow. Detta blev den andra anledningen till att simplare dataset var tvungen att tas fram för att användas med Airflow. I och med att mycket tid hade lagts på det komplexa datasetet slängdes inte det utan det fick fortsätta ligga lagrad i PostgreSQL helt separat från Airflow.

```python
# Group by the unique track ID
df_aggregated = df.groupby('id').agg({
    'artist id': 'first', # We can safely use 'first' since 'id' guarantees uniqueness
    'album id': 'first',
    'name': 'first',
    'popularity': 'first',
    'genres': 'first',
    'artist': 'first',
    'artist popularity': 'first',
    'album': 'first',
    'album release date': 'first',
    'album popularity': 'first',
    'playlist': lambda x:list(set(x)),  # Compile all playlist names into a list
    'danceability': 'mean',  # Average track features if there's variation
    'energy level': 'mean',
    'instrumentalness': 'mean',
    'liveness': 'mean',
    'loudness': 'mean',
    'speechiness': 'mean',
    'tempo': 'mean',
    'duration_ms': 'mean',
    'time signature': 'mean'
}).reset_index()

# Rename the aggregated playlist column for clarity
df_aggregated.rename(columns={'playlist': 'playlist sources'}, inplace=True)

# Column with occurance count
df_aggregated['playlist occurrences'] = df_aggregated['playlist sources'].apply(len)
```

figur 3.1 - Avancerade transformationer med pandas för att aggregera data i det komplexa datasetet. df_aggregated sparades sedan som en parquet-fil.

```python
def transform_raw_playlist_data(raw_playlist_data):
    refined_playlist_tracks = []
    for item in raw_playlist_data['tracks']['items']:
        track_data = item['track']
        refined_playlist_tracks.append({'id': track_data['id'], 'name': track_data['name'],
                                        'popularity': track_data['popularity'], 'explicit': track_data['explicit'],
                                        'duration_ms': track_data['duration_ms'], 'external_url': track_data['external_urls']['spotify'],
                                        'artist': track_data['artists'][0]['name'], 'artist_id': track_data['artists'][0]['id'],
                                        'album': track_data['album']['name'], 'album_id': track_data['album']['id'],
                                        'album_release_date': track_data['album']['release_date'],})

    df = pd.json_normalize(refined_playlist_tracks)
    return df
```

figur 3.2 - Funktion i spotify_etl.py som används i spotify_dag.py för att transformera det simplare datasetet

6.3 Dataladdning

Dataladdning sker i detta projekt i fyra separata ställen i arkitekturen:

1. Den komplexa datan som ska till PostgreSQL extraheras, transformeras och sparas i en parquet-fil. Denna parquet-fil läses in i en DataFrame i ett Python-skript och laddas in i Postgre genom att använda biblioteket psycopg2:

```python
# Prepare and insert data into the track table
track_data_to_insert = [
    (
        row['id'],
        row['artist id'],
        row['album id'],
        row['name'],
        int(row['popularity']),
        row['genres'],
        row['playlist sources'],
        int(row['playlist occurrences'])
    )
    for index, row in df.iterrows()
]

extras.execute_batch(cur, """
                     INSERT INTO track (trackID, artistID, albumID, name, popularity, genres,
                                        playlistSources, playlistOccurrences) VALUES
                     (%s, %s, %s, %s, %s, %s, %s, %s)
                     ON CONFLICT (trackID) DO NOTHING
                     """, track_data_to_insert)
conn.commit()  # Commit after inserting into track

# Clean up
cur.close()
conn.close()
```

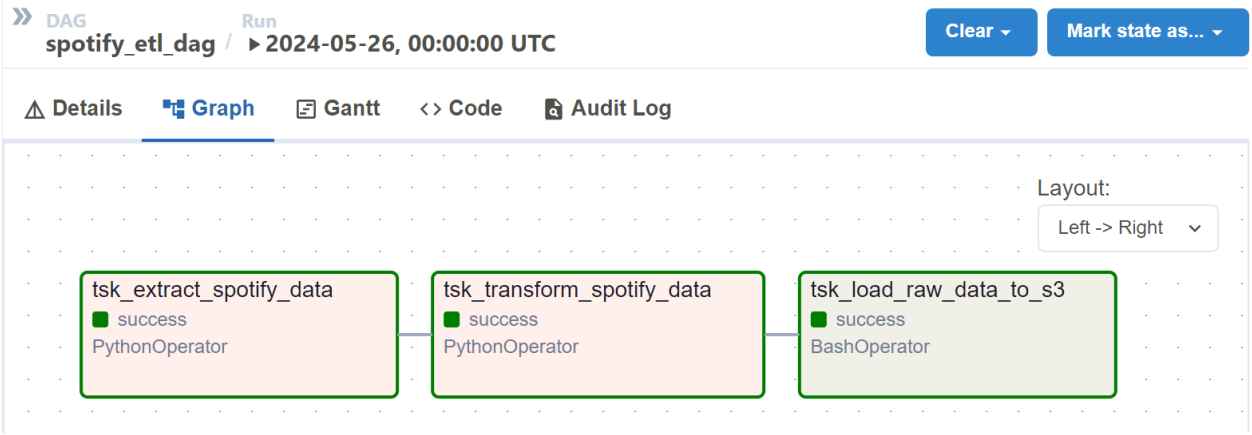
figur 4.1 - Sista raderna i db.py för att ladda in data i PostgreSQL med hjälp av Python och psycopg2

2. Den simpla datan som ska till Snowflake extraheras, transformeras laddas in i en specifik S3 bucket.

3. Den råa datan som senare blir det simpla datasetet extraheras och laddas in i en annan distinkt S3 bucket.

```
with DAG('spotify_etl_dag',
        default_args=default_args,
        schedule_interval = '@daily',
        catchup=False) as dag:

    extract_spotify_data = PythonOperator(
        task_id= 'tsk_extract_spotify_data',
        python_callable=extract_data
    )

    transform_spotify_data = PythonOperator(
        task_id= 'tsk_transform_spotify_data',
        python_callable=transform_data
    )

    load_raw_data_to_s3 = BashOperator(
        task_id= 'tsk_load_raw_data_to_s3',
        bash_command= 'aws s3 mv {{ ti.xcom_pull("tsk_extract_spotify_data")[0] }} s3://spotify-airflow-etl-bucket-raw'
    )

    extract_spotify_data >> transform_spotify_data >> load_raw_data_to_s3
```

figur 4.2 - Sista raderna i spotify_dag.py där vi definierar vår DAG och den råa datan laddas i en S3 bucket. Den transformerade datan laddas i en S3 bucket i funktionen transform_data några rader ovan



figur 4.3 - DAG:en som skapades i Airflow

4. När den transformerade datan hamnar i sin S3 bucket sätts en trigger igång med hjälp av Snowpipe som automatiskt kopierar datan från csv-filen in i en tabell i Snowflake. Denna dataladdning är inte del av Airflow.

```sql
-- Description to fetch the arn for notification updates in our S3 bucket
DESC PIPE spotify_database.snowpipe_schema.spotify_snowpipe;

-- Create a stream to track changes in the staging table
CREATE OR REPLACE STREAM spotify_schema.global50_tracks_stage_stream ON TABLE
spotify_schema.global50_tracks_stage;

-- Create a task to merge data from the staging table into the main table
CREATE OR REPLACE TASK spotify_schema.merge_global50_tracks_task
WAREHOUSE = spotify_warehouse
SCHEDULE = 'USING CRON 0 * * * * Europe/Stockholm'
AS
MERGE INTO spotify_schema.global50_tracks AS target
USING spotify_schema.global50_tracks_stage_stream AS source
ON target.id = source.id
WHEN MATCHED THEN UPDATE SET
    name = source.name,
    popularity = source.popularity,
    explicit = source.explicit,
    duration_ms = source.duration_ms,
    external_url = source.external_url,
    artist = source.artist,
    artist_id = source.artist_id,
    album = source.album,
    album_id = source.album_id,
    album_release_date = source.album_release_date
WHEN NOT MATCHED THEN INSERT (
    id, name, popularity, explicit, duration_ms, external_url, artist, artist_id, album, album_id, album_release_date
) VALUES (
    source.id, source.name, source.popularity, source.explicit, source.duration_ms, source.external_url, source.artist, source.artist_id, source.album, source.album_id,
source.album_release_date
);
```

figur 4.4 - De sista raderna i SQL Worksheet:en som användes för att skapa och ladda data i en data warehouse. Som standard laddas data in även om det leder till dubletter vilket merge_global50_tracks_task förebygger

Datan som laddas in i Snowflake är egentligen så simpel att en databas kan göra jobbet men i och med att det är data som inte har transaktionssyfte utan ska ligga där under en längre period bestämdes det för att skapa en simpel dimension table och lagra datan på star schema format med fact tables och dimension tables igen. Detta även för att det är standarden för hur man ska lagra data i en data warehouse.

Results | Chart

	TRACK_ID	NAME	ARTIST_ID	ALBUM_ID	POPULARITY	E
1	2qSkIjg1o9h3YT9RAgYN75	Espresso	74KM79TiuVKeVCqs8QtB0B	5quMTd5zel9yW5UDua8wS4	100	TI
2	7fzHQizxTqy8wTXwlrgPQQ	MILLION DOLLAR BABY	1WaFQSHVGZQJTbf0BdxdNo	52TwRwdTUMtkpglbOE5IRz	99	FA
3	6AI3ezQ4o3HUoP6Dhudph3	Not Like Us	2YZyLoL8N0Wb9xBt1NhZWg	5JjnoGJyOxfSZUZtk2rRwZ	98	TI
4	6XjDF6nds4DE2BBbagZol6	Gata Only	7CvTknweLr9feJtRGrpDBy	5tSQtQGkrCJx3hoQxmLgfM	98	TI
5	2GxrNKugF82CnoRFbQfzPf	i like the way you kiss me	0PCCGZ0wGLizHt2KZ7hhA2	5HIWDdg3g9CTOtnevKDl1z	98	FA
6	7221xlgOnuakPdLqT0F3nP	I Had Some Help (Feat. Morgan Wallen)	246dkjvS1zLTtiykXe5h60	1woYXxyyxTQJ0E0AhZE6mj	96	TI
7	2FQrifJ1N335Ljm3TjTVVf	A Bar Song (Tipsy)	3y2clKLjiOlp1Np37WiUdH	6egBeCLeGlTzGSo5VyRjwZ	96	TI
8	629DixmZGHc7ILtEntuiWE	LUNCH	6qqNVTkY8uBg9cP3Jd7DAH	7aJuG4TFXa2hmE4z1yxc3n	96	FA
9	2OzhQISqBEmt7hmkYxfT6m	Fortnight (feat. Post Malone)	06HL4z0CvFAxyc27GXpf02	1Mo4aZ8pdj6L1jx8zSwJnt	95	FA
10	6dOtVTDdiauQNBQEDOtlAB	BIRDS OF A FEATHER	6qqNVTkY8uBg9cP3Jd7DAH	7aJuG4TFXa2hmE4z1yxc3n	94	FA
11	7BRD7×5pt8Lqa1eGYC4dzj	CHIHIRO	6qqNVTkY8uBg9cP3Jd7DAH	7aJuG4TFXa2hmE4z1yxc3n	94	FA
12	5uQ7de4EWjb3rkcFxyEOpu	Belong Together	11p2E654TTU8e0nZWBR4AL	3PlCMwyq6nuQYtoYproV1o	93	FA
13	17phhZDn6oGtzMe56NuWvj	Lose Control	33qOK5uJ8AR2xuQQAhHump	7nacKlk586eLRBSlsrk9DB	92	FA
14	2uqYupMHANxnwgeiXTZXzd	Austin (Boots Stop Workin')	7Ez6lTtSMjMf2YSYpukP1I	40HsqPqeSR9Xe3lyAJWr6e	92	TI
15	5Z0UnEtpLDQyYlWwgi8m9C	Too Sweet	2FXC3k01G6Gw61bmprjgqS	3P4SQqMMgjqfqVxgLwtYRk	91	FA
16	3xkHsmpQCBMytMJNiDf3li	Beautiful Things	22wbnEMDvgVIAGdFeek6ET	168CdR21lfn0TTyw1Pkdcm	91	FA

5TPuPsRCf4FN29mAYqTToD 2
5quMTd5zel9yW5UDua8wS4 1
+ 44 more

POPULARITY
57 100

EXPLICIT 0
FALSE 34
TRUE 16

DURATION_MS
120000 334053

EXTERNAL_URL A
100% filled

figur 4.5 - De första 15 raderna av data i Snowflake-tabellen fact_tracks sorterad på popularity med metadata till höger

7. Avslutning

7.1 Slutsatser

Det slutgiltiga resultatet skiljer sig en del från projektbeskrivningen och min initiella vision. Detta dels för att det är svårt att helt förutspå hur ett projekt kommer gå från ruta noll vilket gör det viktigt att utföra projektet med iteration in mind. Resultatet nu i mållinjen är att jag har mer komplex data i Postgre. Allt detta är helt separat från Airflow pipeline:n som har skapats. Denna tar simplare data från API:t, transformerar denna och laddar den in i en data warehouse i Snowflake. Till både datan i PostgreSQL och i Snowflake har det skapats en star schema för att lagra datan i fact tables och dimension tables.

Bristerna är både tekniska som i fallet med Spotifys rate limit och att jag inte lyckades få Airflow att samarbeta med parquet-filer, men även teoretiska. Ändringarna i min dimension model kommer helt från att jag inte har en stadig grund och djup förståelse för star schemas och den typen av modellering med fact tables och dimension tables än. Jag ska vara helt ärlig och säga att jag fick en superytlig förståelse för de för att sedan fylla i med lite mer förståelse nu när terminologi skrivs där denna förståelse ska bli ännu djupare när jag nu efter detta fortsätter att studera på egen hand. Det har varit väldigt väldigt många bumps in the road men att använda AI som ett verktyg när man kodar gör allting ändå väldigt genomförbart med lite beslutsamhet och uthållighet!

Jag skulle dock kunna ha varit mer strategisk med min tid. Att ta beslutet att fokusera på simplare data i Airflow ETL:n tidigare skulle minska stressen kring projektet signifikant. Trots att DAG:en jag designade i Airflow skilde sig från det jag tänkte i början av projektet är jag ändå nöjd med att jag hanterat alla dessa nya tekniker med genuin nyfikenhet och vilja att lära och att jag ända designat en helt automatiserad end-to-end ETL pipeline från API till data warehouse. Det har varit givande att se vissa av the principer jag läst om Data Engineering i praktiken.

Lärdomar till framtiden: planering. Planering, planering, planering. Men detta blir lättare och lättare ju mer iteration och erfarenhet man plockar på sig med varje projekt man tar från början till slut! Överlag har detta projekt varit superkul och superlärorikt!

7.2. Framtida arbete

Det finns utvecklingsmöjligheter som jag inte lyckats hantera. En helt automatiserad pipeline som faktiskt tar data från de 50 största spellistorna med de mer avancerade transformationerna som gjordes innan datan laddades in i Postgre genom att på något sätt hitta sig runt API:ts rate limit är en av de största utvecklingsmöjligheterna. En del av feedback som jag fick när jag presenterade projektet var att AWS Glue skulle kunna användas och simplifiera saker betydligt. Annars är det en end-to-end pipeline som kan skalas utan större problem och där de individuella delarna kan bytas ut allt eftersom behov. En sak jag vill testa snarast är att extrahera real-time data från en sensor eller liknande med Apache Kafka.

När det kommer till datan som nu ligger i både Postgre och Snowflake kan den användas till datavisualisering och dataanalys och när den växer tillräckligt stor användas till Machine Learning-modeller. Att ta pipeline:n ett steg längre och automatiskt visualisera datan som laddas in i Snowflake i en dashboard eller liknande är ett till steg man skulle kunna ta tillsammans med en Data Scientist. Coola saker som jag är mer än villig att göra möjligt som Data Engineer!

8. Bilagor

8.1 Terminologi

ETL: ETL står för Extract Transform Load. Tänk dig att du har en låda med LEGO och en kompis som vill ha en bil byggd av dessa LEGO-bitar. Du skulle då extrahera LEGO-bitar genom att ta de från lådan, transformera de genom att bygga bilen utifrån din kompis beskrivning och vision, och sedan ladda genom att antingen ge bilen till din kompis direkt eller lägga den på ett säkert ställe som du vet att din kompis har tillgång till.

På samma sätt vill vi som Data Engineers 1. Extrahera data från en eller flera datakällor. Detta kan t.ex. vara databaser, appar och system eller API. 2. Transformera datan genom att städa, organisera och ibland sammanfatta den för att se till att den passar behovet av Data Analysts, Data Scientists, Machine Learning Engineers och andra som ska använda datan. Vi ser till att datan är på ett användbart format efter de specifikationer vi får och genom att ha möten för att diskutera. 3. När data är ren och redo att användas laddas den i någon typ av dataarkiv där den kan ligga säkert och lättillgängligt tills det att den t.ex. används för vidare analys, visualisering eller Machine Learning-modeller.

(Data) pipeline: När man säger pipeline i Data Engineering-kontext är det underförstått att man menar en data pipeline. Simplifierat kan man säga att genom att bygga en pipeline gör man det möjligt att automatisera flödet av data och att ETL är en process som är en del av en pipeline. Men en pipeline kan omfatta mer än bara ETL genom att t.e.x. automatiskt visualisera den laddade datan i ett program som PowerBI.

För att spinna vidare på LEGO-analogin: tänk dig att istället för LEGO-bilar gäller det riktiga bilar i en bilverkstad och att du skulle automatisera din process att ta de korrekta delarna, bygga en bil, och göra den tillgänglig till en kund istället för en kompis. För att ta dig själv helt ut ur bilden och ändå se till att bilen byggs och görs tillgänglig kl 8 varje morgon skulle du antingen behöva se till en av dina arbetskollegor finns på plats för att bygga bilen eller bygga en robot som är specialist på att bygga denna typ av bilar. Man kan säga att med en pipeline bygger vi en robot som gör det möjligt att automatisera hela flödet av data.

Data warehouse: Tänk dig att du har ett stort skolprojekt och du behöver göra research på flera ämnen såsom Fysik, Historia och Idrott. Istället för att ha all information spritt över ditt rum, din ryggsäck, ditt skåp etc. kan du istället lagra all information i ett stort arkivskåp. Här i finns all information prydligt organiserat och sorterat och därför snabbt tillgängligt när du behöver det. Ett data warehouse kan ses som detta arkivskåp fast för ett företag. Alla data finns prydligt organiserat och sorterat så att när det behövs för att göra rapporter, se trender och för att göra företagsbeslut finns det snabbt och enkelt tillgängligt!

En data warehouse och en databas kan till synes upplevas som samma sak. Båda samlar data på ett ställe. Skillnaden är att databaser är designade för att hantera transaktioner och

operationer som att lägga till, ändra, och ta bort data. De lagrar och hanterar aktuell data som potentiellt ändras varje dag medan en data warehouse både lagrar aktuell data men främst även historisk data. Fokuset i en data warehouse ligger inte på transaktioner, utan rapportering och analys, och datan är formatterad och strukturerad på ett sätt som gör detta smidigare. [1] Man kan likna en databas till en butiks kassasystem och ett data warehouse som butikens företagskontor där data från flera butiker lagras för att analyseras.

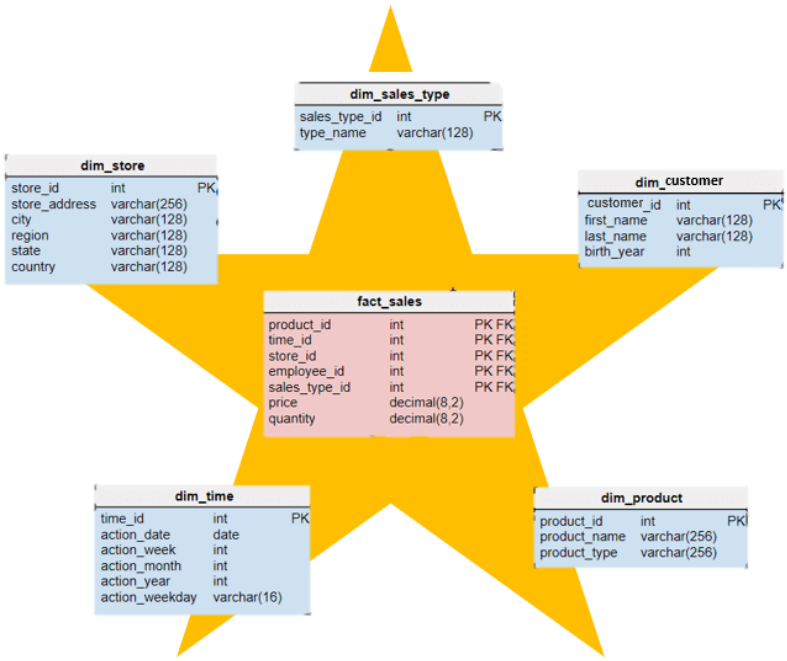
Data warehouse är bland de äldsta och mest väletablerade data-arkitekturerna. Bill Inmon, skaparen av det vi kallar data warehouse, beskrev de själv 1989 så här: “a subject-oriented, integrated, nonvolatile, and time-variant collection of data in support of management’s decisions.” [2]

Datamodellering och Star schemas: Innan data laddas in i en databas är det standard att modellera datan. Vi använder skolanalogin igen: tänk dig att du ska göra en ritning för en superorganiserad pärm med alla dina skolämnen. Alla ämnen - matte, biologi, fysik etc. - får varsin sektion i din pärm. I varje ämne har vi olika typer av anteckingar - lektionsanteckningar, läxor, prov etc. - som alla får varsin sektion i pärmen och blir tylidgt märkta. Precis som med en data warehouse är syftet att all information och all data ska hållas prydligt organiserat.

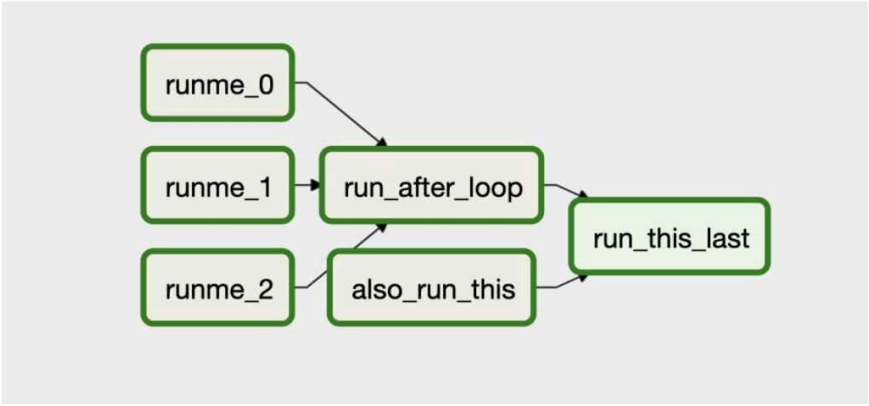
Man gör datamodellering för att få en mental bild av datan; hur den kan delas upp i tabeller och vilka rader som finns i dessa tabeller. Man måste även beskriva *kardinalitet;* det numeriska sambandet mellan tabeller. En artist i Spotify t.ex. kan ha noll eller flera låtar (zero-to-many). Men för att en låt ska existera måste den ha en och endast en huvudartist (one-to-one). Dessa kardinaliteter ritas ut med speciella pilar i datamodelleringen.

En mer komplicerad typ av datamodelleringen är Dimension Modeling där man modellerar data efter det som kallas Kimball Star schema. Här delas data upp i två specifika tabeller; fact tables och dimension tables. Fact tables beskriver de mest fundamentala datan som mäts i ett företag; de väljs för att reflektera de viktigaste affärsprioriteringarna i ett företag. Vår fact table omgivs av dimension tables vilket ger kontext till fact table:n och sätt att gruppera data för insikter.[3] I den star schema som finns nedan kollar vi på sales vilket blir den centrala fact table:n fact_sales. Denna kan användas i query tillsammans med en eller flera av de omgivna dimension tables:

[1] Joe Reis & Matt Housley, “Fundamentals of Data Engineering”, 2022, s. 98
[2] Joe Reis & Matt Housley, “Fundamentals of Data Engineering”, 2022, s. 98
[3] Kimball Group (2008). Fact Tables. https://www.kimballgroup.com/2008/11/fact-tables/ [2024-05-31]



DAG: Apache Airflow är uppbyggd av tasks och dependencies som styrs av såkallade DAGs: Directed Acyclic Graph. Dessa beskrivs bäst med en bild:



Vi kan först bryta ner varje ord i namnet. Directed betyder att alla pilar endast pekar i en riktning. Acyclic betyder att det finns en klar startposition och en klar slutposition och att det aldrig kommer att finnas loopar i en DAG. ‘a-’ eller ‘an-’ som prefix indikterar negativ. Så icke-cyklisk. Graph är helt enkelt formatet vi beskriver våra tasks i; med noder och pilar.

Det är vi som har full kontroll och fullt ansvar att sätta upp dessa dependencies, dvs. i vilken ordning våra tasks ska köras. “DAG:en själv bryr sig inte om *vad* som händer i dessa tasks; den bryr sig endast om *hur* vi ska köra dem - ordningen de ska köras i, hur många gånger vi ska försöka igen ifall något går del, vad vi gör om det blir timeouts, osv.” [4]

[4] The Apache Software Foundation (2024). DAGs. https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/dags.html# [2024-05-30]

8.2 Referenser

Spotify API-dokumentaion. Finns tillgänglig på:
https://developer.spotify.com/documentation/web-api

Apache Airflow dokumentation. Finns tillgänglig på:
https://airflow.apache.org/docs/

Snowflake dokumentation. Finns tillgänglig på:
https://docs.snowflake.com/

The Apache Software Foundation (2024). DAGs.
https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/dags.html# [2024-05-30]

Kimball Group (2008). Fact Tables. https://www.kimballgroup.com/2008/11/fact-tables/
[2024-05-31]

Reis, Joe & Housley, Matt (2022). *Fundamentals of data engineering: plan and build robust data systems.* Sebastopol, CA: O'Reilly