airflow\dags\lastfm_brainz_dag.py

```python
from airflow import DAG
from airflow.operators.empty import EmptyOperator
from airflow.providers.postgres.operators.postgres import PostgresOperator
from airflow.operators.python_operator import PythonOperator, BranchPythonOperator
from airflow.models import Variable
from airflow.providers.postgres.hooks.postgres import PostgresHook
import pandas as pd
import requests
import time
import json
from datetime import datetime, timedelta
import os
import numpy as np
import unicodedata
import uuid


default_args = {
    'owner': 'group6',
    'depends_on_past': False,
    'start_date': datetime(2025, 1, 1),
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': None
}


user_agent = Variable.get("LASTFM_USER_AGENT")
api_key = Variable.get("LASTFM_API_KEY")
base_url = Variable.get("LASTFM_BASE_URL")

MAX_TAGS = 50
TRACKS_PER_TAG = 100
HEADERS = {"User-Agent": user_agent}

def get_top_tags():
    print(f"Getting top {MAX_TAGS} tags from LastFM")

    # Get postgres connection
    pg_hook = PostgresHook(postgres_conn_id='pg_group6')

    # Set up params for lastfm api
    params = {
        "method": "tag.getTopTags",
        "api_key": api_key,
        "format": "json",
    }

    # Call lastfm api
    top_tags_r = requests.get(base_url, params=params, headers=HEADERS)
    top_tags_data = top_tags_r.json()
    df_top_tags = pd.DataFrame(top_tags_data['toptags']['tag'][:MAX_TAGS])

    # Write data to postgres
    with pg_hook.get_conn() as conn:
        with conn.cursor() as cur:
            for index, row in df_top_tags.iterrows():
                cur.execute("""INSERT INTO lastfm_top_tags (tag_name, tag_count, tag_reach)
                            VALUES (%s, %s, %s)""", (row['name'], row['count'],
row['reach']))
        conn.commit()
    print("Saved tags to database")
    return

# Gets basic track info from tag.getTopTracks endpoint
def get_track_basics():
    print(f"Getting basics for {TRACKS_PER_TAG} tracks for each tag")

    # Get api key and base url from airflow variables
    api_key = Variable.get("LASTFM_API_KEY")
    base_url = Variable.get("LASTFM_BASE_URL")

    # Get postgres connection
    pg_hook = PostgresHook(postgres_conn_id='pg_group6')



    # Get top tags from db
    with pg_hook.get_conn() as conn:
        with conn.cursor() as cur:
            cur.execute("SELECT tag_name FROM lastfm_top_tags")
            top_tags = [tag[0] for tag in cur.fetchall()]

    # List to store all tracks
    all_tracks_initial = []
    track_data = []

    # Get tracks for each tag
    for tag in top_tags:
        print(f"Getting tracks for tag: {tag}")
        tag_tracks = []

        # Need to paginate
        page = 1
        while len(tag_tracks) < TRACKS_PER_TAG:
            time.sleep(0.2)
            params = {
                "method": "tag.getTopTracks",
                "tag": tag,
                "api_key": api_key,
                "format": "json",
                "limit": TRACKS_PER_TAG,
                "page": page
            }

            response = requests.get(base_url, params=params, headers=HEADERS).json()
            tracks = response.get("tracks", {}).get("track", [])
            if not tracks:
                break

            # Add tag to each track
            for track in tracks:
                track["tag"] = tag

            tag_tracks.extend(tracks)
            page += 1
        tag_tracks = tag_tracks[:TRACKS_PER_TAG]
        all_tracks_initial.extend(tag_tracks)

    print("Total number of tracks retrieved from LastFM:", len(all_tracks_initial))

    # If duplicate mbid, consolidate tag rank
    consolidated_tracks = {}
    for track in all_tracks_initial:
        mbid = track['mbid']
        if not mbid:
            continue
        tag = track.get('tag', None)
        rank = track.get('@attr', {}).get('rank', None)
        rank = int(rank) if rank is not None else None
        if mbid in consolidated_tracks:
            consolidated_tracks[mbid]['tag_ranks'][tag] = rank
        else:
            track_basics = {
                'mbid': mbid,
                'tag_ranks': {tag: rank}
            }
            consolidated_tracks[mbid] = track_basics

    # Convert to dataframe
    track_data = list(consolidated_tracks.values())
    df_lastfm_initial = pd.DataFrame(track_data)
    initial_count_raw = len(all_tracks_initial)
    final_count = len(df_lastfm_initial)
    print(f"Retrieved basic info for {initial_count_raw} raw tracks")
    print(f"After consolidating tracks by 'mbid', we have {final_count} unique tracks")
    print(f"Removed {initial_count_raw - final_count} duplicate tracks")

```

```python
    return df_lastfm_initial

# Gets detailed track info from track.getInfo endpoint
def get_track_details(df_initial):
    request_count = 0
    all_tracks_details = []

    for index, row in df_initial.iterrows():
        mbid = row['mbid']

        params = {
            "method": "track.getInfo",
            "mbid": mbid,
            "api_key": api_key,
            "format": "json"
        }

        request_count += 1
        try:
            response = requests.get(base_url, params=params, headers=HEADERS)
            if response.status_code == 200:
                response_data = response.json()
                if 'track' in response_data:
                    track_data = response_data['track']
                    track_data['tag_ranks'] = row['tag_ranks']
                    all_tracks_details.append(track_data)
                if request_count % 10 == 0:
                    print(f"Retrieved last.fm track info for {request_count} tracks...")
            else:
                print(f"Failed to retrieve track info for {mbid}. Status code: 
{response.status_code}")
        except Exception as e:
            print(f"Error retrieving track info for {mbid}: {e}")

        # Rate limit
        # Last.fm doesn't publish their limits, but 0.2s was mentioned in some forums
        time.sleep(0.2)

    return all_tracks_details


def process_lastfm_tracks(all_tracks_details):
    print("Processing last.fm tracks...")
    lastfm_tracks = []
    for track in all_tracks_details:
        track_details = {}
        track_details['artist'] = track.get('artist', {}).get('name', None)
        track_details['song_name'] = track.get('name', None)
        track_details['duration'] = track.get('duration', None)
        track_details['listeners'] = track.get('listeners', None)
        track_details['playcount'] = track.get('playcount', None)
        track_details['mbid'] = track.get('mbid', None)
        track_details['album_name'] = track.get('album', {}).get('title', None)
        track_details['url'] = track.get('url', None)
        track_details['tag_ranks'] = json.dumps(track.get('tag_ranks', None))
        toptags = track.get('toptags', {}).get('tag', [])
        if toptags:
            tag_names = [tag.get('name') for tag in toptags if tag.get('name')]
            track_details['toptags'] = tag_names if tag_names else None
        else:
            track_details['toptags'] = None
        track_details['wiki_summary'] = track.get('wiki', {}).get('summary', None)

        lastfm_tracks.append(track_details)

    df_lastfm_tracks = pd.DataFrame(lastfm_tracks)

    df_lastfm_tracks.replace('', None, inplace=True)

    # Clean a few columns
    cols_to_clean = ['artist', 'song_name', 'album_name']
    for col in cols_to_clean:
        if col in df_lastfm_tracks.columns:
            df_lastfm_tracks[col] = df_lastfm_tracks[col].str.lower().str.strip()

    # Make sure integer columns are ints
    int_columns = ['duration', 'listeners', 'playcount']
    for col in int_columns:
        df_lastfm_tracks[col] = pd.to_numeric(df_lastfm_tracks[col], errors='coerce')
    df_lastfm_tracks[int_columns] = 
df_lastfm_tracks[int_columns].where(pd.notnull(df_lastfm_tracks[int_columns]), None)
    for col in int_columns:
        df_lastfm_tracks[col] = df_lastfm_tracks[col].astype('Int64')

    print(f"Processed and cleaned {len(df_lastfm_tracks)} tracks")

    # Make a timestamped csv for record
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    data_dir = os.path.join(os.getcwd(), 'data')
    os.makedirs(data_dir, exist_ok=True)
    lastfm_clean_file_name = f"lastfm_clean_{timestamp}.csv"
    lastfm_clean_file_path = os.path.join(data_dir, lastfm_clean_file_name)
    df_lastfm_tracks.to_csv(lastfm_clean_file_path, index=False, encoding='utf-8-sig')
    print(f"Successfully processed Last.fm tracks...")
    print(f"Successfully wrote record to {lastfm_clean_file_name}\n")

    return df_lastfm_tracks

# Calls the other functions to get last.fm tracks
# Then loads into postgres
```

```python
def get_and_load_lastfm_tracks():
    print("Getting and loading last.fm tracks...")
    df_initial = get_track_basics()
    all_tracks_details = get_track_details(df_initial)
    df_lastfm_tracks = process_lastfm_tracks(all_tracks_details)

    # Get postgres connection
    pg_hook = PostgresHook(postgres_conn_id='pg_group6')


    # Write data to postgres
    with pg_hook.get_conn() as conn:
        with conn.cursor() as cur:
            for index, row in df_lastfm_tracks.iterrows():

                # Generate group6_id
                group6_id = str(uuid.uuid5(uuid.NAMESPACE_DNS, str(row['song_name'].strip() + row['artist'].strip())))

                cur.execute("""INSERT INTO lastfm_tracks (mbid, group6_id, artist, song_name, duration, listeners,
                            playcount, album_name, url, tag_ranks, toptags, wiki_summary)
                            VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
                            ON CONFLICT DO NOTHING""",
                            (row['mbid'], group6_id,
                             row['artist'], row['song_name'], row['duration'], row['listeners'],
                             row['playcount'], row['album_name'], row['url'],
                             row['tag_ranks'], row['toptags'], row['wiki_summary']))
            conn.commit()
            print(f"Saved {len(df_lastfm_tracks)} tracks to postgres")

    return


# Gets acousticbrainz features
# For all tracks with mbids currently in the lastfm_tracks table
def get_ab_features():
    print("Getting acousticbrainz features")

    # Get base url from airflow variables
    base_url = Variable.get("AB_BASE_URL")
    endpoint = base_url + "/api/v1/high-level"

    # Get postgres connection
    pg_hook = PostgresHook(postgres_conn_id='pg_group6')

    # Get all mbid from lastfm_tracks
    with pg_hook.get_conn() as conn:
        with conn.cursor() as cur:
            # Select both mbid and group6_id
            cur.execute("SELECT mbid, group6_id FROM lastfm_tracks WHERE mbid IS NOT NULL and mbid != ''")
            mbid_results = cur.fetchall()

            # Create dict mapping from mbid to group6_id
            mbid_to_group6id = {row[0]: row[1] for row in mbid_results}

            # List of just mbids for the API calls
            mbid_list = list(mbid_to_group6id.keys())

    batch_size = 25
    print('Starting to get features for', len(mbid_list), f'tracks using batch_size = {batch_size}')

    results = []

    # Get features for each batch of mbid
    for i in range(0, len(mbid_list), batch_size):
        batch = mbid_list[i:i + batch_size]
        params = {'recording_ids': ';'.join(batch)}

        response = requests.get(endpoint, params=params, headers=HEADERS)
        response.raise_for_status()  # Raise exception for HTTP errors

        response_data = response.json()

        # Process each mbid in the response
        for mbid, data in response_data.items():
            if mbid not in batch or '0' not in data:
                continue

            group6_id = mbid_to_group6id.get(mbid)
            if not group6_id:
                print(f"Warning: No group6_id found for mbid: {mbid}")
                continue

            metadata = data['0']['metadata']
            tags = metadata.get('tags', {})
            highlevel = data['0']['highlevel']

            # Simple, direct field extraction
            features = {
                'mbid': mbid,
                'group6_id': group6_id,
                'artist': json.dumps(tags.get('artist', None)),
                'song_name': json.dumps(tags.get('title', None)),
                'album': json.dumps(tags.get('album', None)),
                'date': json.dumps(tags.get('date', None)),
                'isrcs': json.dumps(tags.get('isrc', None)),
                'bpm': json.dumps(tags.get('bpm', None)),
                'initialkey': tags.get('initialkey', None),
                'musicbrainz_albumid': tags.get('musicbrainz_albumid', None),
                'musicbrainz_artistid': tags.get('musicbrainz_artistid', None),
                'mood': json.dumps(tags.get('mood', None)),

                'danceability_danceable': highlevel.get('danceability', {}).get('all', {}).get('danceable', None),
                'danceability_not_danceable': highlevel.get('danceability', {}).get('all', {}).get('not_danceable', None),
                'danceability_max_class': highlevel.get('danceability', {}).get('value', None),

                'gender_female': highlevel.get('gender', {}).get('all', {}).get('female', None),
                'gender_male': highlevel.get('gender', {}).get('all', {}).get('male', None),
                'gender_max_class': highlevel.get('gender', {}).get('value', None),

                'genre_alternative': highlevel.get('genre_dortmund', {}).get('all', {}).get('alternative', None),
                'genre_blues': highlevel.get('genre_dortmund', {}).get('all', {}).get('blues', None),
                'genre_electronic': highlevel.get('genre_dortmund', {}).get('all', {}).get('electronic', None),
                'genre_folkcountry': highlevel.get('genre_dortmund', {}).get('all', {}).get('folkcountry', None),
                'genre_funksoulrnb': highlevel.get('genre_dortmund', {}).get('all', {}).get('funksoulrnb', None),
                'genre_jazz': highlevel.get('genre_dortmund', {}).get('all', {}).get('jazz', None),
                'genre_pop': highlevel.get('genre_dortmund', {}).get('all', {}).get('pop', None),
                'genre_raphiphop': highlevel.get('genre_dortmund', {}).get('all', {}).get('raphiphop', None),
                'genre_rock': highlevel.get('genre_dortmund', {}).get('all', {}).get('rock', None),
                'genre_dortmund_max_class': highlevel.get('genre_dortmund', {}).get('value', None),

                'voice_instrumental_instrumental': highlevel.get('voice_instrumental', {}).get('all', {}).get('instrumental', None),
                'voice_instrumental_voice': highlevel.get('voice_instrumental', {}).get('all', {}).get('voice', None),
                'voice_instrumental_max_class': highlevel.get('voice_instrumental', {}).get('value', None),
                'genre': json.dumps(tags.get('genre', None))
            }


            results.append(features)


```

```
        print('Batch', i // batch_size + 1, 'of', (len(mbid_list) - 1) // batch_size + 1, 
'complete')
        time.sleep(0.2)

    if results:
        results_df = pd.DataFrame(results)

        results_df.replace('', None, inplace=True)

        # Convert empty strings to None for all numeric fields
        numeric_fields = ['danceability_danceable', 'danceability_not_danceable', 
                        'gender_female', 'gender_male',
                        'genre_alternative', 'genre_blues', 'genre_electronic',
                        'genre_folkcountry', 'genre_funksoulrnb', 'genre_jazz',
                        'genre_pop', 'genre_raphiphop', 'genre_rock',
                        'voice_instrumental_instrumental', 'voice_instrumental_voice']

        # Convert numeric fields that need to be floats
        for field in numeric_fields:
            if field in results_df.columns:
                results_df[field] = pd.to_numeric(results_df[field], errors='coerce')

        # Replace NaN with None
        results_df = results_df.where(pd.notnull(results_df), None)

        timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
        data_dir = os.path.join(os.getcwd(), 'data')
        filename = f"acousticbrainz_features_{timestamp}.csv"
        file_path = os.path.join(data_dir, filename)
        results_df.to_csv(file_path, index=False, encoding='utf-8-sig')
        print(f"Saved {len(results_df)} records to {filename}")
    else:
        print("No features were retrieved")


    # Now open a new database connection to insert the data
    with pg_hook.get_conn() as conn:
        with conn.cursor() as cur:
            for track in results:

                if 'bpm' in track and (track['bpm'] == '' or track['bpm'] == []):
                    track['bpm'] = None

                # Directly replace all empty strings in all features
                for key in track:
                    if track[key] == '':
                        track[key] = None

                cur.execute("""INSERT INTO acousticbrainz_features 
                            (mbid, group6_id, artist, song_name, album, date, isrcs, 
```

```python
                            bpm, initialkey, musicbrainz_albumid, musicbrainz_artistid, mood,
                            danceability_danceable, danceability_not_danceable, danceability_max_class,
                            gender_female, gender_male, gender_max_class,
                            genre_alternative, genre_blues, genre_electronic, genre_folkcountry, genre_funksoulrnb,
                            genre_jazz, genre_pop, genre_raphiphop, genre_rock, genre_dortmund_maxclass,
                            voice_instrumental_instrumental, voice_instrumental_voice, voice_instrumental_max_class, genre)
                            VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s,
                                %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)""",
                            (track['mbid'],
                            track['group6_id'],
                            track['artist'],
                            track['song_name'],
                            track['album'],
                            track['date'],
                            track['isrcs'],
                            track['bpm'],
                            track['initialkey'],
                            track['musicbrainz_albumid'],
                            track['musicbrainz_artistid'],
                            track['mood'],
                            track['danceability_danceable'],
                            track['danceability_not_danceable'],
                            track['danceability_max_class'],
                            track['gender_female'],
                            track['gender_male'],
                            track['gender_max_class'],
                            track['genre_alternative'],
                            track['genre_blues'],
                            track['genre_electronic'],
                            track['genre_folkcountry'],
                            track['genre_funksoulrnb'],
                            track['genre_jazz'],
                            track['genre_pop'],
                            track['genre_raphiphop'],
                            track['genre_rock'],
                            track['genre_dortmund_max_class'],
                            track['voice_instrumental_instrumental'],
                            track['voice_instrumental_voice'],
                            track['voice_instrumental_max_class'],
                            track['genre']))
                conn.commit()
            print(f"Inserted {len(results)} tracks into database")


with DAG(
        'lastfm_brainz_dag',
        default_args=default_args,
        description='Dag for lastfm and AcousticBrainz data',
        schedule_interval=None,
        catchup=False
    ) as lastfm_dag:

    start_task = EmptyOperator(task_id='start')

    create_tables = PostgresOperator(
        task_id='create_tables',
        postgres_conn_id='pg_group6',
        sql='sql/lastfm_brainz_create_z.sql'
    )

    load_top_tags = PythonOperator(
        task_id='load_top_tags',
        python_callable=get_top_tags
    )

    load_lastfm_tracks = PythonOperator(
        task_id='load_lastfm_tracks',
        python_callable=get_and_load_lastfm_tracks
    )

    load_ab_features = PythonOperator(
        task_id='load_ab_features',
        python_callable=get_ab_features
    )

    end_task = EmptyOperator(task_id='end')

    start_task >> create_tables >> load_top_tags >> load_lastfm_tracks >> load_ab_features >> end_task


```