test

November 23, 2022

```
In [ ]: %load_ext sql
```

```
In [ ]: %sql postgresql://student:student@127.0.0.1/sparkifydb
```

```
In [ ]: %sql SELECT * FROM songplays LIMIT 5;
```

```
In [ ]: %sql SELECT count(*) FROM songplays
```

```
In [ ]: %sql SELECT * FROM users LIMIT 5;
```

```
In [ ]: %sql SELECT count(*) FROM users
```

```
In [ ]: %sql SELECT * FROM songs LIMIT 5;
```

```
In [ ]: %sql SELECT count(*) FROM songs
```

```
In [ ]: %sql SELECT * FROM artists LIMIT 5;
```

```
In [ ]: %sql SELECT count(*) FROM artists
```

```
In [ ]: %sql SELECT * FROM time LIMIT 5;
```

```
In [ ]: %sql SELECT count(*) FROM time
```

0.1 REMEMBER: Restart this notebook to close connection to `sparkifydb`

Each time you run the cells above, remember to restart this notebook to close the connection to your database. Otherwise, you won't be able to run your code in `create_tables.py`, `etl.py`, or `etl.ipynb` files since you can't make multiple connections to the same database (in this case, sparkifydb).

1 Sanity Tests

Execute the cells below once you are ready to submit the project. Some basic sanity testing will be performed to esnure that your work does NOT contain any commonly found issues.

Run each cell and if a cell produces an warning message is orange, you should make appropriate changes to your code before submitting. If all test in a cell pass,no warnings will be printed.

The test cases assume that you are using certain column names in your tables. If you get a `IndexError: single positional indexer is out-of-bounds` you may need to change the column names being used by the test cases. Instructions for doing this appear right boefore cell that may require these changes.

The tests below are only meant to help you make your work foolproof. The submission will still be graded by a human grader against the project rubric.

1.1 Grab Table Names for Testing

```
In [ ]: import sql_queries as sqlq
```

```
In [ ]: %%sql _tablenames <<
        SELECT tablename
        FROM pg_catalog.pg_tables
        WHERE schemaname != 'pg_catalog' AND schemaname != 'information_schema' AND tableowner =
```

```
In [ ]: tablenames = _tablenames.DataFrame()
```

```
In [ ]: user_table = [name for name in list(tablenames.tablename) if name in sqlq.user_table_cre
        song_table = [name for name in list(tablenames.tablename) if name in sqlq.song_table_cre
        artists_table = [name for name in list(tablenames.tablename) if name in sqlq.artist_tabl
        songplay_table = [name for name in list(tablenames.tablename) if name in sqlq.songplay_t
```

1.2 Run Primary Key Tests

```
In [ ]: %sql _output << SELECT a.attname, format_type(a.atttypid, a.atttypmod) AS data_type, a.a
        FROM   pg_index i \
        JOIN   pg_attribute a ON a.attrelid = i.indrelid \
                             AND a.attnum = ANY(i.indkey) \
        WHERE  i.indrelid = '{user_table}'::regclass
```

```
In [ ]: if not _output:
            print('\033[93m'+'[WARNING] '+ f"The {user_table} table does not have a primary key!
```

```
In [ ]: %sql _output << SELECT a.attname, format_type(a.atttypid, a.atttypmod) AS data_type, a.a
        FROM   pg_index i \
        JOIN   pg_attribute a ON a.attrelid = i.indrelid \
                             AND a.attnum = ANY(i.indkey) \
        WHERE  i.indrelid = '{artists_table}'::regclass
```

```
In [ ]: if not _output:
            print('\033[93m'+'[WARNING] '+ f"The {artists_table} table does not have a primary k
```

```
In [ ]: %sql _output << SELECT a.attname, format_type(a.atttypid, a.atttypmod) AS data_type, a.a
        FROM   pg_index i \
        JOIN   pg_attribute a ON a.attrelid = i.indrelid \
                             AND a.attnum = ANY(i.indkey) \
        WHERE  i.indrelid = '{songplay_table}'::regclass
```

```
In [ ]: if not _output:
            print('\033[93m'+'[WARNING] '+ f"The {songplay_table} table does not have a primary
```

```
In [ ]: %sql _output << SELECT a.attname, format_type(a.atttypid, a.atttypmod) AS data_type, a.a
        FROM   pg_index i \
        JOIN   pg_attribute a ON a.attrelid = i.indrelid \
                             AND a.attnum = ANY(i.indkey) \
        WHERE  i.indrelid = '{song_table}'::regclass
```

```
In [ ]: if not _output:
            print('\033[93m'+'[WARNING] '+ f"The {song_table} table does not have a primary key!
```

1.3 Run Data Type and Constraints Check

```
In [ ]: %sql _output << SELECT * FROM information_schema.columns where table_name='{user_table}'
```

Check the column `user_id` for correct data type. If you get a `IndexError: single positional indexer is out-of-bounds` error, you may be using a different column name. Change the column name below and run the cell again.

```
In [ ]: output = _output.DataFrame()
        _dtype = output[output.column_name == 'user_id'].data_type.iloc[0]
        if _dtype not in ['integer', 'bigint']:
            print('\033[93m'+'[WARNING] '+ f"Type {_dtype} may not be an appropriate data type f

In [ ]: %sql _output << SELECT * FROM information_schema.columns where table_name='{song_table}'
```

Check the column `year` for correct data type. Check columns `title` and `duration` for not-NULL constraints.

If you get a `IndexError: single positional indexer is out-of-bounds` error, you may be using different column names. Change the column name(s) below and run the cell again.

```
In [ ]: output = _output.DataFrame()

        _dtype = output[output.column_name == 'year'].data_type.iloc[0]
        if _dtype not in ['integer']:
            print('\033[93m'+'[WARNING] '+ f"Type '{_dtype}' may not be an appropriate data type

        _nullable_title = output[output.column_name == 'title'].is_nullable.iloc[0]
        _nullable_duration = output[output.column_name == 'duration'].is_nullable.iloc[0]
        if (_nullable_duration != 'NO') or (_nullable_title != 'NO'):
            print('\033[93m'+'[WARNING] '+ f"You may want to add appropriate NOT NULL constraint

In [ ]: %sql _output << SELECT * FROM information_schema.columns where table_name='{artists_tabl
```

Check the columns `latitude` and `longitude` for correct data type. Check column `name` for not-NULL constraint.

If you get a `IndexError: single positional indexer is out-of-bounds` error, you may be using different column names. Change the column name(s) below and run the cell again.

```
In [ ]: output = _output.DataFrame()

        _dtype_latitude = output[output.column_name == 'latitude'].data_type.iloc[0]
        if _dtype_latitude not in ['double precision']:
            print('\033[93m'+'[WARNING] '+ f"Type '{_dtype_latitude}' may not be an appropriate

        _dtype_longitude = output[output.column_name == 'longitude'].data_type.iloc[0]
        if _dtype_longitude not in ['double precision']:
```

```
        print('\033[93m'+'[WARNING] '+ f"Type '{_dtype_longitude}' may not be an appropriate


    _nullable_name = output[output.column_name == 'name'].is_nullable.iloc[0]
    if _nullable_name != 'NO':
        print('\033[93m'+'[WARNING] '+ f"You may want to add appropriate NOT NULL constraint
```

```
In [ ]: %sql _output << SELECT * FROM information_schema.columns where table_name='{songplay_tab
```

Check the columns `start_time` and `user_id` for correct data type. Check columns `start_time` and `user_id` for not-NULL constraint.

If you get a `IndexError: single positional indexer is out-of-bounds` error, you may be using different column names. Change the column name(s) below and run the cell again.

```
In [ ]: output = _output.DataFrame()
```

```
    _dtype_start_time = output[output.column_name == 'start_time'].data_type.iloc[0]
    if 'timestamp' not in _dtype_start_time:
        print('\033[93m'+'[WARNING] '+ f"Type '{_dtype_start_time}' may not be an appropriat

    _dtype_user_id = output[output.column_name == 'user_id'].data_type.iloc[0]
    if _dtype_user_id not in ['integer', 'bigint']:
        print('\033[93m'+'[WARNING] '+ f"Type '{_dtype_user_id}' may not be an appropriate d


    _nullable_time = output[output.column_name == 'start_time'].is_nullable.iloc[0]
    _nullable_uid = output[output.column_name == 'user_id'].is_nullable.iloc[0]

    if (_nullable_time != 'NO') or (_nullable_uid != 'NO'):
        print('\033[93m'+'[WARNING] '+ f"You may want to add appropriate NOT NULL constraint
```

1.4 Run Tests for Upsertion Check

```
In [ ]: import re
```

```
In [ ]: if not re.search('ON\s+CONFLICT',sqlq.songplay_table_insert,re.IGNORECASE) or \
           not re.search('ON\s+CONFLICT',sqlq.user_table_insert,re.IGNORECASE) or \
           not re.search('ON\s+CONFLICT',sqlq.song_table_insert,re.IGNORECASE) or \
           not re.search('ON\s+CONFLICT',sqlq.artist_table_insert,re.IGNORECASE):
            print('\033[93m'+'[WARNING]Some of your insert queries may need an "ON CONFLICT" cla
            print('\033[93m'+'         You can either skip conflicting insertions with with "ON
            print('\033[93m'+'         OR use "ON CONFLICT DO UPDATE SET"')
            print('\033[93m'+'         Check this link for more details: https://www.postgresqlt
```