# Pair Programming Joins and Views

HD Sheets, February 6, 2025 DSE5002 MODULE 5 PETER GYORDA APRIL 16, 2025

Sources

https://www.sqlitetutorial.net/sqlite-join/

Beaulieau, Chapter 5, Chapter 10,

In [3]:
```
# Set Up and Connect
```

In [5]:
```
# Libaries

import sqlalchemy

# we will want Pandas for the data frame structure

import pandas as pd
```

In [9]:
```
# Connect to the database
# Alter this to reflect your username and password,   this is for postgres on the same machine

engine=sqlalchemy.create_engine('postgresql://todd:password@localhost:5432/chinook')
```

In [11]:
```
# really just testing the connection

pd.read_sql_query("SELECT table_name  FROM information_schema.tables LIMIT 15",engine)
```

Out[11]:

|  | table_name |
|---|---|
| **0** | artist |
| **1** | album |
| **2** | employee |
| **3** | customer |
| **4** | invoice |
| **5** | invoice_line |
| **6** | track |
| **7** | playlist |
| **8** | playlist_track |
| **9** | genre |
| **10** | pg_statistic |
| **11** | pg_type |
| **12** | media_type |
| **13** | pg_foreign_table |
| **14** | pg_authid |

## Finding the artist for each album

Suppose we want a list of the artists for each album,

the album titles are in album, the artist names are in artist.

in album, we have album.artist_id which is the same artist id number as in artist, where it is artist.artist_id, we can use these in the Join

This is ordered by title

```
In [13]: pd.read_sql_query("""SELECT
                        title,name
                    FROM album
                        INNER JOIN artist ON artist.artist_id =album.artist_id
                ORDER BY title;
                   """
                     ,engine)
```

Out[13]:

| | title | name |
|---|---|---|
| **0** | ...And Justice For All | Metallica |
| **1** | [1997] Black Light Syndrome | Terry Bozzio, Tony Levin & Steve Stevens |
| **2** | 20th Century Masters - The Millennium Collecti... | Scorpions |
| **3** | A-Sides | Soundgarden |
| **4** | A Copland Celebration, Vol. I | Aaron Copland & London Symphony Orchestra |
| **...** | ... | ... |
| **342** | War | U2 |
| **343** | Warner 25 Anos | Antônio Carlos Jobim |
| **344** | Weill: The Seven Deadly Sins | Kent Nagano and Orchestre de l'Opéra de Lyon |
| **345** | Worlds | Aaron Goldberg |
| **346** | Zooropa | U2 |

347 rows × 2 columns

```
In [17]: # LEFT JOIN
         # We could also do this with a LEFT JOIN, since every album has an associated artist,
         # we get the same result as we did with the inner join

         # If this is meant to be actual SQL code, it should be in a string:
         sql_query = """
```

```
SELECT *
FROM albums
LEFT JOIN artists ON albums.artist_id = artists.id
"""
```

In [19]:
```
pd.read_sql_query("""SELECT
                         title,name
                     FROM album
                         LEFT JOIN artist ON artist.artist_id =album.artist_id
                     ORDER BY title
                    """
                      ,engine)
```

Out[19]:

| | title | name |
|---|---|---|
| **0** | ...And Justice For All | Metallica |
| **1** | [1997] Black Light Syndrome | Terry Bozzio, Tony Levin & Steve Stevens |
| **2** | 20th Century Masters - The Millennium Collecti... | Scorpions |
| **3** | A-Sides | Soundgarden |
| **4** | A Copland Celebration, Vol. I | Aaron Copland & London Symphony Orchestra |
| **...** | ... | ... |
| **342** | War | U2 |
| **343** | Warner 25 Anos | Antônio Carlos Jobim |
| **344** | Weill: The Seven Deadly Sins | Kent Nagano and Orchestre de l'Opéra de Lyon |
| **345** | Worlds | Aaron Goldberg |
| **346** | Zooropa | U2 |

347 rows × 2 columns

## RIGHT JOIN

If we do the same join with a RIGHT JOIN, I would expect will cause some problems since each artist may have multiple albums

```
In [21]: pd.read_sql_query("""SELECT
                              title,name
                          FROM album
                              RIGHT JOIN artist ON artist.artist_id =album.artist_id
                          ORDER BY title
                         """
                           ,engine)
```

Out[21]:

| | title | name |
|---|---|---|
| **0** | ...And Justice For All | Metallica |
| **1** | [1997] Black Light Syndrome | Terry Bozzio, Tony Levin & Steve Stevens |
| **2** | 20th Century Masters - The Millennium Collecti... | Scorpions |
| **3** | A-Sides | Soundgarden |
| **4** | A Copland Celebration, Vol. I | Aaron Copland & London Symphony Orchestra |
| **...** | ... | ... |
| **413** | None | Jaguares |
| **414** | None | Barão Vermelho |
| **415** | None | João Gilberto |
| **416** | None | Los Lonely Boys |
| **417** | None | Jorge Vercilo |

418 rows × 2 columns

## CROSS JOIN

creates all possible combinations, also called a "Cartesian Join"

In the SELECT before we get the first name of each employee, with each possible media type after the employee's name

They can be useful for creating large and varied test sets for use in development

It might be helpful to generate a "grid" of all permutations for calculating over all possible combinations, for example 4 sales categories over each of 12 months

In [23]:
```
pd.read_sql_query("""SELECT employee.first_name, media_type.name mt_name FROM employee
                     CROSS JOIN media_type""",engine)
```

Out[23]:

| | first_name | mt_name |
|---|---|---|
| **0** | Andrew | MPEG audio file |
| **1** | Nancy | MPEG audio file |
| **2** | Jane | MPEG audio file |
| **3** | Margaret | MPEG audio file |
| **4** | Steve | MPEG audio file |
| **5** | Michael | MPEG audio file |
| **6** | Robert | MPEG audio file |
| **7** | Laura | MPEG audio file |
| **8** | Andrew | Protected AAC audio file |
| **9** | Nancy | Protected AAC audio file |
| **10** | Jane | Protected AAC audio file |
| **11** | Margaret | Protected AAC audio file |
| **12** | Steve | Protected AAC audio file |
| **13** | Michael | Protected AAC audio file |
| **14** | Robert | Protected AAC audio file |
| **15** | Laura | Protected AAC audio file |
| **16** | Andrew | Protected MPEG-4 video file |
| **17** | Nancy | Protected MPEG-4 video file |
| **18** | Jane | Protected MPEG-4 video file |
| **19** | Margaret | Protected MPEG-4 video file |
| **20** | Steve | Protected MPEG-4 video file |
| **21** | Michael | Protected MPEG-4 video file |

| | first_name | mt_name |
|---|---|---|
| **22** | Robert | Protected MPEG-4 video file |
| **23** | Laura | Protected MPEG-4 video file |
| **24** | Andrew | Purchased AAC audio file |
| **25** | Nancy | Purchased AAC audio file |
| **26** | Jane | Purchased AAC audio file |
| **27** | Margaret | Purchased AAC audio file |
| **28** | Steve | Purchased AAC audio file |
| **29** | Michael | Purchased AAC audio file |
| **30** | Robert | Purchased AAC audio file |
| **31** | Laura | Purchased AAC audio file |
| **32** | Andrew | AAC audio file |
| **33** | Nancy | AAC audio file |
| **34** | Jane | AAC audio file |
| **35** | Margaret | AAC audio file |
| **36** | Steve | AAC audio file |
| **37** | Michael | AAC audio file |
| **38** | Robert | AAC audio file |
| **39** | Laura | AAC audio file |

# Views

A View is the stored output of a query

I haven't figured out how to create a View using SQL Alchemy, that seems to be an issue

We can do it through the postgress command window

1.) Start the postgres command window and log in as the superuser postgres

2.) Connect to the chinook database

```
    \connect chinook
```

3.) Creat a view

```
    CREATE VIEW enames AS SELECT first_name, last_name FROM employee;
```

4.) Use \dv to see all the viewers, and verify it works

5.) Grant your user access to the view

```
   GRANT SELECT ON ALL TABLES IN SCHEMA public TO bob;

   my user is bob,  you may have a different username

   Note: when we set up bob as a user, we granted him SELECT privileges, but when we create new
tables or views
   we have to grant it again.    There is a way to change this default setting in postgres, but
finding that could be
   a bit of work
```

6.) We can now treat the View (enames) as though it was a table. This can be very helpful if we have a large database and really complex queries to carry out. The View can simplify this

In [31]:
```
# First, check if we're connected to the database
try:
    # This query should work on any PostgreSQL database
    pd.read_sql_query("SELECT 1;", engine)
    print("Database connection is working")

    # List all tables in the database
    tables = pd.read_sql_query("""
        SELECT table_name
        FROM information_schema.tables
        WHERE table_schema = 'public';
    """, engine)

    print("Available tables in the database:")
    print(tables)

    # After seeing the available tables, use the correct table name
    # For example, if you see a table named 'employee' (not 'employees'):
    # pd.read_sql_query("SELECT * FROM employee;", engine)

except Exception as e:
    print(f"Error connecting to database: {e}")
```

```
Database connection is working
Available tables in the database:
        table_name
0           artist
1            album
2         employee
3         customer
4          invoice
5     invoice_line
6            track
7         playlist
8   playlist_track
9            genre
10      media_type
```

In [ ]:

```
```

In [33]:

```
# Query to get first and last names from the employee table as separate columns
enames = pd.read_sql_query("""
    SELECT first_name, last_name
    FROM employee;
""", engine)
```

```
# Display the result
print("Employee names:")
enames.head()
```

```
Employee names:
```

Out[33]:

| | first_name | last_name |
|---|---|---|
| **0** | Andrew | Adams |
| **1** | Nancy | Edwards |
| **2** | Jane | Peacock |
| **3** | Margaret | Park |
| **4** | Steve | Johnson |

In [35]:
```
engine.dispose()
```

In [ ]:
```
```