Pair Programming Joins and Views

HD Sheets, February 6, 2025 checked 6/27/2025

Sources

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

Beaulieau, Chapter 5, Chapter 10,

Set Up and Connect

In [1]:
```
# Libaries

import sqlalchemy

# we will want Pandas for the data frame structure

import pandas as pd
```

In [11]:
```
# Connect to the database
# Alter this to reflect your username and password,   this is for postgres c

engine=sqlalchemy.create_engine('postgresql://Alt-Evelyn:superuser@localhost
```

In [12]:
```
# really just testing the connection, by reading some tables to be sure the

# this isn't strictly necessary, but I'd rather know now if the connection i

pd.read_sql_query("SELECT table_name  FROM information_schema.tables LIMIT 1
```

Out[12]:

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

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_
                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 Copland Celebration, Vol. I	Aaron Copland & London Symphony Orchestra
4	A Matter of Life and Death	Iron Maiden
...	...	...
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

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

In [14]:
```
pd.read_sql_query("""SELECT
                        title,name
                    FROM album
                        LEFT JOIN artist ON artist.artist_id =album.artist_i
                    ORDER BY title
                   """
                     ,engine)
```

Out[14]:

	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 Copland Celebration, Vol. I	Aaron Copland & London Symphony Orchestra
4	A Matter of Life and Death	Iron Maiden
...	...	...
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 [15]:
```
pd.read_sql_query("""SELECT
                        title,name
                    FROM album
                        RIGHT JOIN artist ON artist.artist_id =album.artist_
                    ORDER BY title
                   """
                     ,engine)
```

Out[15]:

	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 Copland Celebration, Vol. I	Aaron Copland & London Symphony Orchestra
4	A Matter of Life and Death	Iron Maiden
...	...	...
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 [16]:
```
pd.read_sql_query("""SELECT employee.first_name, media_type.name mt_name FRO
                     CROSS JOIN media_type""",engine)
```

Out[16]:

	first_name	mt_name
0	Andrew	MPEG audio file
1	Andrew	Protected AAC audio file
2	Andrew	Protected MPEG-4 video file
3	Andrew	Purchased AAC audio file
4	Andrew	AAC audio file
5	Nancy	MPEG audio file
6	Nancy	Protected AAC audio file
7	Nancy	Protected MPEG-4 video file
8	Nancy	Purchased AAC audio file
9	Nancy	AAC audio file
10	Jane	MPEG audio file
11	Jane	Protected AAC audio file
12	Jane	Protected MPEG-4 video file
13	Jane	Purchased AAC audio file
14	Jane	AAC audio file
15	Margaret	MPEG audio file
16	Margaret	Protected AAC audio file
17	Margaret	Protected MPEG-4 video file
18	Margaret	Purchased AAC audio file
19	Margaret	AAC audio file
20	Steve	MPEG audio file
21	Steve	Protected AAC audio file
22	Steve	Protected MPEG-4 video file
23	Steve	Purchased AAC audio file
24	Steve	AAC audio file
25	Michael	MPEG audio file
26	Michael	Protected AAC audio file
27	Michael	Protected MPEG-4 video file
28	Michael	Purchased AAC audio file
29	Michael	AAC audio file
30	Robert	MPEG audio file
31	Robert	Protected AAC audio file

	first_name	mt_name
32	Robert	Protected MPEG-4 video file
33	Robert	Purchased AAC audio file
34	Robert	AAC audio file
35	Laura	MPEG audio file
36	Laura	Protected AAC audio file
37	Laura	Protected MPEG-4 video file
38	Laura	Purchased 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 [17]:
```
pd.read_sql_query("""SELECT *
                     FROM enames1;""",engine)
```

Out[17]:

	first_name	last_name
0	Andrew	Adams
1	Nancy	Edwards
2	Jane	Peacock
3	Margaret	Park
4	Steve	Johnson
5	Michael	Mitchell
6	Robert	King
7	Laura	Callahan

In [18]:
```
engine.dispose()
```

In [ ]:
```

```