SQL Queries Part 2

Pair Programming Exercise DSE5002

Peter Gyorda, April 10, 2025

More details on SQL Queries

IN

subqueries

GROUP BY HAVING

regex in postgres

Source material

"Learning SQL", Beaulieu, O'Reilly 2005

https://www.sqlitetutorial.net/ - explains queries using the chinook database, albeit in the SQLite database system. The SELECT system used for queries is pretty standard for most SQL databases, the other aspects and commands seem to be a bit more variable from one server program to another.

That said, there are minor differences in variable names between the chinook database in postgres and the tutorial for SQlite, watch for underscores and pluralization (track vs tracks, etc). I have fixed all the examples shown here.

In [3]:
```
# Libaries

import sqlalchemy

# we will want Pandas for the data frame structure
```

```
import pandas as pd
```

In [5]:
```
# 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 [7]:
```
# really just testing the connection

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

Out[7]:

	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

IN

Used to find if a variable is within a range of value

In [9]:
```
# Look for tracks with media types 1 or 2

pd.read_sql_query("""SELECT
                        track_id,
                        name,
                        media_type_id
                    FROM
                        track
                    WHERE
                        media_type_id IN (1, 2)
                    ORDER BY
                        name ASC
                    LIMIT 20;"""
                     ,engine)
```

Out[9]:

	track_id	name	media_type_id
0	602	'Round Midnight	1
1	3027	"40"	1
2	3412	"Eine Kleine Nachtmusik" Serenade In G, K. 525...	2
3	109	#1 Zero	1
4	3254	#9 Dream	2
5	1833	(Anesthesia) Pulling Teeth	1
6	570	(Da Le) Yaleo	1
7	3045	(I Can't Help) Falling In Love With You	1
8	3057	(Oh) Pretty Woman	1
9	3471	(There Is) No Greater Love (Teo Licks)	2
10	1947	(We Are) The Road Crew	1
11	2595	(White Man) In Hammersmith Palais	1
12	709	(Wish I Could) Hideaway	1
13	1894	...And Justice For All	1
14	3273	[Just Like] Starting Over	2
15	2505	[Untitled]	1
16	1268	01 - Prowler	1
17	1269	02 - Sanctuary	1
18	1270	03 - Remember Tomorrow	1
19	1271	04 - Running Free	1

Subquery

A subquery is one query nested within another.

Here is a starting query that retrieves all the album ids for a given artist

In [11]:
```
pd.read_sql_query("""SELECT
                        album_id
                    FROM
                        album
                    WHERE
                        artist_id=12;"""
                     ,engine)
```

Out[11]:

	album_id
0	16
1	17

Subqueries and the IN operation

we can now use this query as a subquerry within another query to look up all the tracks by this artist, on whatever albums they have out'

Notice how the query we saw above is used within the IN operation

Notice that this subquery uses no values from the outer loop, so it is not a "correlated subquery".

In [13]:
```
pd.read_sql_query("""SELECT
                        track_id, name, album_id
                    FROM
                        track
                    WHERE
                        album_id IN(
                            SELECT
                                album_id
                            FROM
                                album
```

```
                                WHERE
                                    artist_id=12);"""
                            ,engine)
```

Out[13]:

	track_id	name	album_id
0	149	Black Sabbath	16
1	150	The Wizard	16
2	151	Behind The Wall Of Sleep	16
3	152	N.I.B.	16
4	153	Evil Woman	16
5	154	Sleeping Village	16
6	155	Warning	16
7	156	Wheels Of Confusion / The Straightener	17
8	157	Tomorrow's Dream	17
9	158	Changes	17
10	159	FX	17
11	160	Supernaut	17
12	161	Snowblind	17
13	162	Cornucopia	17
14	163	Laguna Sunrise	17
15	164	St. Vitus Dance	17
16	165	Under The Sun/Every Day Comes and Goes	17

Subqueries in a WHERE, another example

We can find all the invoices with total greater than 1.5 times the average invoice price

The subquery here finds 1.5 times the average invoice total for us

The subquery here is (SELECT AVG(total) FROM invoice)

In [15]:
```
pd.read_sql_query("""SELECT
                        invoice_id, customer_id, total
                    FROM
                        invoice
                    WHERE
                        total>1.5*(SELECT AVG(total) FROM invoice)
                    ORDER BY
                        total;"""
                     ,engine)
```

Out[15]:

	invoice_id	customer_id	total
0	207	54	8.91
1	410	35	8.91
2	11	52	8.91
3	403	56	8.91
4	18	31	8.91
...	...	...	...
115	201	25	18.86
116	96	45	21.86
117	194	46	21.86
118	299	26	23.86
119	404	6	25.86

120 rows × 3 columns

In [17]:
```
pd.read_sql_query("""SELECT
                        *
```

```
                    FROM
                        invoice
                    LIMIT 10;
                    """
                     ,engine)
```

Out[17]:

	invoice_id	customer_id	invoice_date	billing_address	billing_city	billing_state	billing_country	billing_postal_code	total
0	1	2	2021-01-01	Theodor-Heuss-Straße 34	Stuttgart	None	Germany	70174	1.98
1	2	4	2021-01-02	Ullevålsveien 14	Oslo	None	Norway	0171	3.96
2	3	8	2021-01-03	Grétrystraat 63	Brussels	None	Belgium	1000	5.94
3	4	14	2021-01-06	8210 111 ST NW	Edmonton	AB	Canada	T6G 2C7	8.91
4	5	23	2021-01-11	69 Salem Street	Boston	MA	USA	2113	13.86
5	6	37	2021-01-19	Berger Straße 10	Frankfurt	None	Germany	60316	0.99
6	7	38	2021-02-01	Barbarossastraße 19	Berlin	None	Germany	10779	1.98
7	8	40	2021-02-01	8, Rue Hanovre	Paris	None	France	75002	1.98
8	9	42	2021-02-02	9, Place Louis Barthou	Bordeaux	None	France	33000	3.96
9	10	46	2021-02-03	3 Chatham Street	Dublin	Dublin	Ireland	None	5.94

Regular Expressions

We can use a set of expression to match text strings to regular expressions, allowing use of regular expression based operations within a query

In postgress we have

~ case sensitive match to a regular expression

~* non case senstive match to a regular expression

!~ not a match, case sensitive

!~*- not a match, not case sensitive

Example let's find if Philip Glass is in our composer list

I am using the regular expression '(.+)glas(.+)' which means any number of any character follwed by glas and then any number of characters leaving off the last s in glass means that (.+) can find the last s. Other more effective regular expressions ar e possible

In [19]:
```
pd.read_sql_query("""SELECT
                        *
                    FROM
                        track
                    WHERE composer ~* '(.+)glas(.+)';"""
                     ,engine)
```

Out[19]:

	track_id	name	album_id	media_type_id	genre_id	composer	milliseconds	bytes	unit_price
0	560	Unidos Da Tijuca	45	1	7	Douglas/Neves, Vicente Das/Silva, Gilmar L./To...	338834	11440689	0.99
1	3503	Koyaanisqatsi	347	2	10	Philip Glass	206005	3305164	0.99

GROUP BY

Group data in some way, typically computing an aggregate function for the group

In this example we are grouping the counts of tracks from albums, grouping by album_id

In [21]:
```
pd.read_sql_query("""SELECT
                        album_id,
                        COUNT(track_id)
                    FROM
                        track
                    GROUP BY
```

```
    album_id
LIMIT 10
;"""
 ,engine)
```

Out[21]:

	album_id	count
0	1	10
1	2	1
2	3	3
3	4	8
4	5	15
5	6	13
6	7	12
7	8	14
8	9	8
9	10	14

In [23]:

```
#Add an order

pd.read_sql_query("""SELECT
    album_id,
    COUNT(track_id)
FROM
    track
GROUP BY
    album_id
ORDER BY COUNT(track_id) DESC
LIMIT 10
;"""
 ,engine)
```

Out[23]:

	album_id	count
0	141	57
1	23	34
2	73	30
3	229	26
4	230	25
5	251	25
6	83	24
7	231	24
8	253	24
9	228	23

HAVING

We can filter grouping results using a HAVING operation

In [25]:

```
pd.read_sql_query("""SELECT
                        album_id,
                        COUNT(track_id)
                    FROM
                        track
                    GROUP BY
                        album_id
                    HAVING COUNT(track_id)>20
                    ORDER BY COUNT(track_id) DESC
                    ;"""
                     ,engine)
```

Out[25]:

	album_id	count
0	141	57
1	23	34
2	73	30
3	229	26
4	251	25
5	230	25
6	83	24
7	231	24
8	253	24
9	228	23
10	255	23
11	24	23
12	51	22
13	250	22
14	224	22
15	167	21
16	39	21

In [21]:

```
# Using SUM() in a Group By

# note the use of aliasing with the SUM() operations, so sum of milliseconds is length
# and sum of bytes is size

pd.read_sql_query("""SELECT
                        album_id,
                        SUM(milliseconds) length,
```

```
                    SUM(bytes) size
                FROM
                    track
                GROUP BY
                    album_id
                ;"""
                 ,engine)
```

Out[21]:

	album_id	length	size
0	184	2967110	97909484
1	116	3327211	111603549
2	87	915904	30732325
3	273	501503	8285941
4	51	4637011	151386329
...	...	...	...
342	55	4499818	147920569
343	148	3759224	122464832
344	130	2555478	84008603
345	270	3292399	54019835
346	23	7875643	261227821

347 rows × 3 columns

In [27]:

```
# Another HAVING example

pd.read_sql_query("""SELECT
                    album_id,
                    COUNT(track_id)
                FROM
                    track
                GROUP BY
                    album_id
```

```
HAVING album_id < 10
;"""
 ,engine)
```

Out[27]:

	album_id	count
0	1	10
1	2	1
2	3	3
3	4	8
4	5	15
5	6	13
6	7	12
7	8	14
8	9	8

In [ ]: