Generating Total Revenue for a City Using Rollup

The data is a three year record of revenue generated from 144 stores in 37 cities. The business owners want to generate the total revenue from each city and the total revenue from the individual stores in that city in one report.

This notebook shows how to generate this report by using a query with a 'rollup' in the group by clause.

In [ ]:
```
#import libraries for the notebook
import os
from sqlalchemy import create_engine
import pandas as pd
```

In [ ]:
```
# parameters for connecting to the database
host = os.getenv('HOST')
database = os.getenv('SQL_DATABASE')
user = os.getenv('SQL_USER')
password = os.getenv('SQL_PASSWORD')
```

In [ ]:
```
connection_string = f"postgresql://{user}:{password}@{host}/{database}"
```

In [ ]:
```
engine = create_engine(connection_string)
```

Queries

Total Revenue By Store and By City

The revenue generated from the sale of products can be grouped by stores. This will generate the total revenue for the 144 stores in the database. These stores are in 37 cities and when we group these stores by cities, we can generate the total revenue for each city. Ordinarily this will require two separate queries and a merge operation. Other ways could be exporting the result of grouping th erevenue by stores and doing the other part of the analysis on excel or a similar program.

This query applies an extension of the 'group by' clause which allows extra rows representing subtotals in addition to the grand total in line with the columns included in the 'group by' clause.

In [ ]:
```
city_tot_rev = pd.read_sql(
    '''SELECT sales.store_id, city_id, CAST(SUM(revenue) AS DECIMAL
    (10,2)) FROM sales INNER JOIN store_cities ON sales.store_id=
    store_cities.store_id GROUP BY rollup(city_id, sales.store_id)
    order by city_id, sales.store_id
    ''',
    engine
)
```

In [ ]:

```
city_tot_rev.head(10)
```

Out[ ]:

	store_id	city_id	sum
0	S0005	C001	108092.49
1	S0036	C001	115044.76
2	None	C001	223137.25
3	S0104	C002	680987.33
4	None	C002	680987.33
5	S0068	C003	61471.00
6	S0086	C003	41781.45
7	None	C003	103252.45
8	S0038	C004	1027630.08
9	None	C004	1027630.08

The result of the query shows that after each set of stores in a city, an extra row is added showing the total revenue for that city. On these rows, the store_id column is set to 'None'(NULL). After the total for all the cities have been displayed, another extra row that displays the total revenue for all stores and all cities. The store_id and products_id columns in this row are set to 'None'(NULL) as shown in the output below.

In [ ]:

```
city_tot_rev.tail(5)
```

Out[ ]:

	store_id	city_id	sum
177	S0090	C036	145177.28
178	None	C036	1947336.62
179	S0130	C037	37790.84
180	None	C037	37790.84
181	None	None	37700236.56

Total Revenue By City

Following the query above, we can apply a partial rollup to be able to produce the report requested by the business owners. The partial rollup query applies the rollup to only the store_id column. In this query, there is no grand total revenue.

In [ ]:

```
city_rev = pd.read_sql(
    '''select sales.store_id, city_id, cast(sum(revenue) as decimal
    (10,2)) from sales inner join store_cities on sales.store_id=
    store_cities.store_id group by city_id, rollup(sales.store_id)
    order by city_id, sales.store_id
    ''',
    engine
)
```

In [ ]:
```
city_rev
```

Out[ ]:

	store_id	city_id	sum
0	S0005	C001	108092.49
1	S0036	C001	115044.76
2	None	C001	223137.25
3	S0104	C002	680987.33
4	None	C002	680987.33
...	...	...	...
176	S0078	C036	156648.62
177	S0090	C036	145177.28
178	None	C036	1947336.62
179	S0130	C037	37790.84
180	None	C037	37790.84

181 rows × 3 columns

We can apply a filter to the query above to limit its result to a particular store. An additional step we can include in the filtered result is to replace the 'None' value with a 'total_store_revenue' value.

Let's start off with the total revenue for city C003

In [ ]:
```
city003_tot_rev = pd.read_sql(
    '''select *
    from( select coalesce(sales.store_id, 'total_city_revenue') store_id,
    city_id, cast(sum(revenue) as decimal
    (10,2)) from sales inner join store_cities on sales.store_id=
    store_cities.store_id group by city_id, rollup(sales.store_id))result
    where result.city_id = 'C003'
    ''',
    engine
)
```

In [ ]:
```
city003_tot_rev
```

Out[ ]:

	store_id	city_id	sum
0	S0086	C003	41781.45
1	S0068	C003	61471.00
2	total_city_revenue	C003	103252.45

Next, we can generate the total revenue for city C030

In [ ]:
```
city030_tot_rev = pd.read_sql(
    '''select *
    from( select coalesce(sales.store_id, 'total_city_revenue') store_id,
    city_id, cast(sum(revenue) as decimal
    (10,2)) from sales inner join store_cities on sales.store_id=
    store_cities.store_id group by city_id, rollup(sales.store_id))result
    where result.city_id = 'C030'
    ''',
    engine
)
```

In [ ]:
```
city030_tot_rev
```

Out[ ]:

	store_id	city_id	sum
0	S0100	C030	59941.53
1	S0019	C030	56689.20
2	S0070	C030	196766.92
3	S0124	C030	93258.39
4	S0017	C030	120575.53
5	total_city_revenue	C030	527231.57

We can do the same for the city with the maximum revenue. This city has been identified from a previous query (not shown on this notebook) as city C014.

In [ ]:
```
city014_tot_rev = pd.read_sql(
    '''select *
    from( select coalesce(sales.store_id, 'total_city_revenue') store_id,
    city_id, cast(sum(revenue) as decimal
    (10,2)) from sales inner join store_cities on sales.store_id=
    store_cities.store_id group by city_id, rollup(sales.store_id))result
    where result.city_id = 'C014'
    ''',
    engine
)
```

In [ ]:
```
city014_tot_rev
```

Out[ ]:

	store_id	city_id	sum
0	S0007	C014	26003.36
1	S0107	C014	163289.94
2	S0095	C014	1008681.52
3	S0014	C014	149459.32
4	S0089	C014	48882.70
5	S0052	C014	171398.04
6	S0010	C014	360954.11
7	S0120	C014	52746.63
8	S0050	C014	257590.68
9	S0097	C014	1488854.26
10	S0076	C014	76478.26
11	S0080	C014	212949.41
12	S0062	C014	1299166.32
13	S0067	C014	231282.88
14	S0016	C014	93782.65
15	S0022	C014	270016.85
16	S0073	C014	136912.28
17	S0072	C014	327770.84
18	S0055	C014	201183.64
19	S0143	C014	57265.58
20	S0059	C014	92031.65
21	S0039	C014	80900.86
22	S0003	C014	132146.69
23	S0026	C014	1317370.03
24	S0126	C014	429800.43
25	S0015	C014	366800.78
26	S0020	C014	1052297.55
27	S0077	C014	64744.05
28	S0085	C014	2156037.86
29	S0071	C014	130608.09
30	S0058	C014	238352.02
31	S0099	C014	64680.28
32	total_city_revenue	C014	12760439.56

This notebook shows a simple query that can generate total revenue for each store in a selected city as well as the total revenue for the city. While there are other queries or combinations of queries to achieve the same result, using 'rollup' in the broup by clause provides a simplistic approach to generating the same result.