L1 E2 - 0 - OLAP Cubes - Solution

March 10, 2022

1 Exercise 02 - OLAP Cubes - Solution

All the databases table in this demo are based on public database samples and transformations - Sakila is a sample database created my MySql Link - The postgresql version of it is called Pagila Link - The facts and dimension tables design is based on O'Reilly's public dimensional modelling tutorial schema Link

```
In [1]: !PGPASSWORD=student createdb -h 127.0.0.1 -U student pagila_star
        !PGPASSWORD=student psql -q -h 127.0.0.1 -U student -d pagila_star -f Data/pagila-star.s
```

```
 set_config
------------

(1 row)

 setval
--------
    200
(1 row)

 setval
--------
    605
(1 row)

 setval
--------
     16
(1 row)

 setval
--------
    600
(1 row)

 setval
--------
    109
```

```
(1 row)

 setval
--------
    599
(1 row)

 setval
--------
      1
(1 row)

 setval
--------
      1
(1 row)

 setval
--------
      1
(1 row)

 setval
--------
      1
(1 row)

 setval
--------
  16049
(1 row)

 setval
--------
   1000
(1 row)

 setval
--------
   4581
(1 row)

 setval
--------
      6
(1 row)

 setval
```

```
--------
  32098
(1 row)

 setval
--------
  16049
(1 row)

 setval
--------
      2
(1 row)

 setval
--------
      2
(1 row)
```

```
In [2]: %load_ext sql
        import sql
```

2 STEP1 : Connect to the local database where Pagila is loaded

```
In [3]: DB_ENDPOINT = "127.0.0.1"
        DB = 'pagila'
        DB_USER = 'student'
        DB_PASSWORD = 'student'
        DB_PORT = '5432'

        # postgresql://username:password@host:port/database
        conn_string = "postgresql://{}:{}@{}:{}/{}" \
                                .format(DB_USER, DB_PASSWORD, DB_ENDPOINT, DB_PORT, DB)

        print(conn_string)
```

```
postgresql://student:student@127.0.0.1:5432/pagila
```

```
In [4]: %sql $conn_string
```

```
Out[4]: 'Connected: student@pagila'
```

3 STEP2 : Star Schema

4 Start by a simple cube

```
In [5]: %%time
        %%sql
        SELECT dimDate.day,dimMovie.rating, dimCustomer.city, sum(sales_amount) as revenue
        FROM factSales
        JOIN dimMovie     on (dimMovie.movie_key         = factSales.movie_key)
        JOIN dimDate      on (dimDate.date_key         = factSales.date_key)
        JOIN dimCustomer  on (dimCustomer.customer_key = factSales.customer_key)
        group by (dimDate.day, dimMovie.rating, dimCustomer.city)
        order by revenue desc
        limit  20;
```

```
 * postgresql://student:***@127.0.0.1:5432/pagila
0 rows affected.
CPU times: user 4.69 ms, sys: 156 ţs, total: 4.84 ms
Wall time: 10.1 ms
```

```
Out[5]: []
```

4.1 Slicing

- Slicing is the reduction of the dimensionality of a cube by 1 e.g. 3 dimensions to 2, fixing one of the dimensions to a single value
- In the following example we have a 3-deminensional cube on day, rating, and country
- In the example below `rating` is fixed and to "PG-13" which reduces the dimensionality

```
In [6]: %%time
        %%sql
        SELECT dimDate.day,dimMovie.rating, dimCustomer.city, sum(sales_amount) as revenue
        FROM factSales
        JOIN dimMovie     on (dimMovie.movie_key         = factSales.movie_key)
        JOIN dimDate     on (dimDate.date_key         = factSales.date_key)
        JOIN dimCustomer on (dimCustomer.customer_key = factSales.customer_key)
        WHERE dimMovie.rating = 'PG-13'
        GROUP by (dimDate.day, dimCustomer.city, dimMovie.rating)
        ORDER by revenue desc
        LIMIT  20;
```

```
 * postgresql://student:***@127.0.0.1:5432/pagila
0 rows affected.
CPU times: user 842 ţs, sys: 4.12 ms, total: 4.96 ms
Wall time: 7.77 ms
```

```
Out[6]: []
```

4.2 Dicing

- Creating a subcube, same dimensionality, less values for 2 or more dimensions
- e.g. PG-13

```
In [7]: %%time
        %%sql
        SELECT dimDate.day,dimMovie.rating, dimCustomer.city, sum(sales_amount) as revenue
        FROM factSales
        JOIN dimMovie     on (dimMovie.movie_key         = factSales.movie_key)
        JOIN dimDate      on (dimDate.date_key           = factSales.date_key)
        JOIN dimCustomer  on (dimCustomer.customer_key = factSales.customer_key)
        WHERE dimMovie.rating in ('PG-13', 'PG')
        AND dimCustomer.city in ('Bellevue', 'Lancaster')
        AND dimDate.day in ('1', '15', '30')
        GROUP by (dimDate.day, dimCustomer.city, dimMovie.rating)
        ORDER by revenue desc
        LIMIT  20;
```

```
 * postgresql://student:***@127.0.0.1:5432/pagila
0 rows affected.
CPU times: user 5.25 ms, sys: 264 ţs, total: 5.51 ms
Wall time: 7.48 ms
```

```
Out[7]: []
```

4.3 Roll-up

- Stepping up the level of aggregation to a large grouping
- e.g.`city` is summed as `country`

```
In [8]: %%time
        %%sql
        SELECT dimDate.day,dimMovie.rating, dimCustomer.country, sum(sales_amount) as revenue
        FROM factSales
        JOIN dimMovie     on (dimMovie.movie_key         = factSales.movie_key)
        JOIN dimDate      on (dimDate.date_key           = factSales.date_key)
        JOIN dimCustomer  on (dimCustomer.customer_key = factSales.customer_key)
        GROUP by (dimDate.day,  dimMovie.rating, dimCustomer.country)
        ORDER by revenue desc
        LIMIT  20;
```

```
 * postgresql://student:***@127.0.0.1:5432/pagila
0 rows affected.
CPU times: user 3.84 ms, sys: 678 ţs, total: 4.52 ms
Wall time: 6.59 ms
```

```
Out[8]: []
```

4.4 Drill-down

- Breaking up one of the dimensions to a lower level.
- e.g.city is broken up to districts

```
In [9]: %%time
        %%sql
        SELECT dimDate.day,dimMovie.rating, dimCustomer.district, sum(sales_amount) as revenue
        FROM factSales
        JOIN dimMovie     on (dimMovie.movie_key         = factSales.movie_key)
        JOIN dimDate     on (dimDate.date_key         = factSales.date_key)
        JOIN dimCustomer on (dimCustomer.customer_key = factSales.customer_key)
        GROUP by (dimDate.day, dimCustomer.district, dimMovie.rating)
        ORDER by revenue desc
        LIMIT  20;
```

```
 * postgresql://student:***@127.0.0.1:5432/pagila
0 rows affected.
CPU times: user 5.57 ms, sys: 0 ns, total: 5.57 ms
Wall time: 10.1 ms
```

```
Out[9]: []
```

5 Grouping Sets

- It happens a lot that for a 3 dimensions, you want to aggregate a fact:
  - by nothing (total)
  - then by the 1st dimension
  - then by the 2nd
  - then by the 3rd
  - then by the 1st and 2nd
  - then by the 2nd and 3rd
  - then by the 1st and 3rd
  - then by the 1st and 2nd and 3rd
- Since this is very common, and in all cases, we are iterating through all the fact table anyhow, there is a move clever way to do that using the SQL grouping statement "GROUPING SETS"

5.1 total revenue

```
In [10]: %%sql
         SELECT sum(sales_amount) as revenue
         FROM factSales
```

```
 * postgresql://student:***@127.0.0.1:5432/pagila
1 rows affected.
```

```
Out[10]: [(None,)]
```

5.2 revenue by country

```
In [11]: %%sql
         SELECT dimStore.country,sum(sales_amount) as revenue
         FROM factSales
         JOIN dimStore on (dimStore.store_key = factSales.store_key)
         GROUP by  dimStore.country
         order by dimStore.country, revenue desc;
```

```
 * postgresql://student:***@127.0.0.1:5432/pagila
0 rows affected.
```

```
Out[11]: []
```

5.3 revenue by month

```
In [12]: %%sql
         SELECT dimDate.month,sum(sales_amount) as revenue
         FROM factSales
         JOIN dimDate     on (dimDate.date_key         = factSales.date_key)
         GROUP by dimDate.month
         order by dimDate.month, revenue desc;
```

```
 * postgresql://student:***@127.0.0.1:5432/pagila
0 rows affected.
```

```
Out[12]: []
```

5.4 revenue by month & country

```
In [13]: %%sql
         SELECT dimDate.month,dimStore.country,sum(sales_amount) as revenue
         FROM factSales
         JOIN dimDate     on (dimDate.date_key         = factSales.date_key)
         JOIN dimStore on (dimStore.store_key = factSales.store_key)
         GROUP by (dimDate.month, dimStore.country)
         order by dimDate.month, dimStore.country, revenue desc;
```

```
 * postgresql://student:***@127.0.0.1:5432/pagila
0 rows affected.
```

```
Out[13]: []
```

5.5 revenue total, by month, by country, by month & country All in one shot

- watch the nones

```
In [14]: %%time
         %%sql
         SELECT dimDate.month,dimStore.country,sum(sales_amount) as revenue
         FROM factSales
         JOIN dimDate  on (dimDate.date_key  = factSales.date_key)
         JOIN dimStore on (dimStore.store_key = factSales.store_key)
         GROUP by grouping sets ((), dimDate.month,  dimStore.country, (dimDate.month,  dimStore
```

```
 * postgresql://student:***@127.0.0.1:5432/pagila
1 rows affected.
CPU times: user 1.39 ms, sys: 3.44 ms, total: 4.83 ms
Wall time: 9.87 ms
```

```
Out[14]: [(None, None, None)]
```

6 CUBE

- Group by CUBE (dim1, dim2, ..) , produces all combinations of different lenghts in one go.
- This view could be materialized in a view and queried which would save lots repetitive aggregations

```
SELECT dimDate.month,dimStore.country,sum(sales_amount) as revenue
FROM factSales
JOIN dimDate  on (dimDate.date_key   = factSales.date_key)
JOIN dimStore on (dimStore.store_key = factSales.store_key)
GROUP by cube(dimDate.month,  dimStore.country);
```

```
In [15]: %%time
         %%sql
         SELECT dimDate.month,dimStore.country,sum(sales_amount) as revenue
         FROM factSales
         JOIN dimDate     on (dimDate.date_key         = factSales.date_key)
         JOIN dimStore on (dimStore.store_key = factSales.store_key)
         GROUP by cube(dimDate.month,  dimStore.country);
```

```
 * postgresql://student:***@127.0.0.1:5432/pagila
1 rows affected.
CPU times: user 3.92 ms, sys: 0 ns, total: 3.92 ms
Wall time: 8.09 ms
```

```
Out[15]: [(None, None, None)]
```

6.1 revenue total, by month, by country, by month & country All in one shot, NAIVE way

```
In [16]: %%time
         %%sql
```

```
SELECT  NULL as month, NULL as country, sum(sales_amount) as revenue
FROM factSales
    UNION all
SELECT NULL, dimStore.country,sum(sales_amount) as revenue
FROM factSales
JOIN dimStore on (dimStore.store_key = factSales.store_key)
GROUP by  dimStore.country
    UNION all
SELECT cast(dimDate.month as text) , NULL, sum(sales_amount) as revenue
FROM factSales
JOIN dimDate on (dimDate.date_key = factSales.date_key)
GROUP by dimDate.month
    UNION all
SELECT cast(dimDate.month as text),dimStore.country,sum(sales_amount) as revenue
FROM factSales
JOIN dimDate     on (dimDate.date_key         = factSales.date_key)
JOIN dimStore on (dimStore.store_key = factSales.store_key)
GROUP by (dimDate.month, dimStore.country)
```

```
 * postgresql://student:***@127.0.0.1:5432/pagila
1 rows affected.
CPU times: user 6.94 ms, sys: 0 ns, total: 6.94 ms
Wall time: 15.7 ms
```

```
Out[16]: [(None, None, None)]
```