OLAP: CUBE operator

DATA-DRIVEN DECISION MAKING IN SQL

Irene Ortner
Data Scientist at Applied Statistics

Introduction to OLAP

- OLAP: on-line analytical processing
- Aggregate data for a better overview
  - Count number of rentings for each customer.
  - Average rating of movies for each genre and each country.
- Produce pivot tables to present aggregation results

Table rentings_extended

renting_id	country	genre	rating
2	Belgium	Drama	10
32	Belgium	Drama	10
203	Austria	Drama	6
292	Austria	Comedy	8
363	Belgium	Drama	7
.....	.....	.....	.....

Pivot table - number of movie rentals

	Austria	Belgium	Total
Comedy	2	1	3
Drama	4	15	19
Total	6	16	22

Pivot table and SQL output

	Austria	Belgium	Total
Comedy	2	1	3
Drama	4	15	19
Total	6	16	22

country	genre	count
Austria	Comedy	2
Austria	Drama	4
Belgium	Comedy	1
Belgium	Drama	15
Austria	null	6
Belgium	null	16
null	Comedy	3
null	Drama	19
null	null	22

GROUP BY CUBE

```
SELECT country,
       genre,
       COUNT(*)
FROM renting_extended
GROUP BY CUBE (country, genre);
```

country	genre	count
Austria	Comedy	2
Belgium	Drama	15
Austria	Drama	4
Belgium	Comedy	1
Belgium	null	16
Austria	null	6
null	Comedy	3
null	Drama	19
null	null	22

Number of ratings

```
SELECT country,
       genre,
       COUNT(rating)
FROM renting_extended
GROUP BY CUBE (country, genre);
```

```
| country  | genre   | count |
|----------|---------|-------|
| Austria  | Comedy | 1     |
| Belgium  | Drama  | 6     |
| Austria  | Drama  | 2     |
| Belgium  | Comedy | 0     |
| Belgium  | null   | 6     |
| Austria  | null   | 3     |
| null     | Comedy | 1     |
| null     | Drama  | 8     |
| null     | null   | 9     |
```

Now it's your turn to GROUP BY CUBE!

DATA-DRIVEN DECISION MAKING IN SQL

ROLLUP

DATA-DRIVEN DECISION MAKING IN SQL

Bart Baesens
Professor Data Science and Analytics

datacamp

Table renting_extended

The first few rows of the table `renting_extended`:

renting_id	country	genre	rating
2	Belgium	Drama	10
32	Belgium	Drama	10
203	Austria	Drama	6
292	Austria	Comedy	8
363	Belgium	Drama	7
.....	.....	.....	.....

Query with ROLLUP

```
SELECT country,
       genre,
       COUNT(*)
FROM renting_extended
GROUP BY ROLLUP (country, genre);
```

- Levels of aggregation
  - Aggregation of each combination of country and genre
  - Aggregation of country alone
  - Total aggregation

Query with ROLLUP

```
SELECT country,
       genre,
       COUNT(*)
FROM renting_extended
GROUP BY ROLLUP (country, genre);
```

country	genre	count
null	null	22
Austria	Comedy	2
Belgium	Drama	15
Austria	Drama	4
Belgium	Comedy	1
Belgium	null	16
Austria	null	6

Order in ROLLUP

```
SELECT country,
       genre,
       COUNT(*)
FROM renting_extended
GROUP BY ROLLUP (genre, country);
```

```
| country | genre   | count |
|---------|---------|-------|
| null    | null    | 22    |
| Austria | Comedy  | 2     |
| Belgium | Drama   | 15    |
| Austria | Drama   | 4     |
| Belgium | Comedy  | 1     |
| null    | Comedy  | 3     |
| null    | Drama   | 19    |
```

Summary ROLLUP

- Returns aggregates for a hierarchy of values, e.g. `ROLLUP (country, genre)`
  - Movie rentals for each country and each genre
  - Movie rentals for each country
  - Total number of movie rentals
- In each step, one level of detail is dropped
- Order of column names is important for ROLLUP

Number of rentals and ratings

```
SELECT country,
       genre,
       COUNT(*) AS n_rentals,
       COUNT(rating) AS n_ratings
FROM renting_extended
GROUP BY ROLLUP (genre, country);
```

country	genre	n_rentals	n_ratings
null	null	22	9
Belgium	Drama	15	6
Austria	Comedy	2	1
Belgium	Comedy	1	0
Austria	Drama	4	2
null	Comedy	3	1
null	Drama	19	8

Let's practice!

DATA-DRIVEN DECISION MAKING IN SQL

OLAP operations: GROUPING SETS

DATA-DRIVEN DECISION MAKING IN SQL

Irene Ortner
Data Scientist at Applied Statistics

datacamp

Overview of OLAP operators in SQL

Extensions in SQL to facilitate OLAP operations

- `GROUP BY CUBE`
- `GROUP BY ROLLUP`
- `GROUP BY GROUPING SETS`

Table renting_extended

The first few rows of the table `renting_extended`:

renting_id	country	genre	rating
2	Belgium	Drama	10
32	Belgium	Drama	10
203	Austria	Drama	6
292	Austria	Comedy	8
363	Belgium	Drama	7
.....	.....	.....	.....

GROUP BY GROUPING SETS

Example of a query with `GROUPING SETS` operator:

```
SELECT country,
       genre,
       COUNT(*)
FROM rentings_extended
GROUP BY GROUPING SETS ((country, genre), (country), (genre), ());
```

- Column names surrounded by parentheses represent one level of aggregation.
- `GROUP BY GROUPING SETS` returns a `UNION` over several `GROUP BY` queries.

GROUPING SETS and GROUP BY queries

```
SELECT country,
       genre,
       COUNT(*)
FROM renting_extended
GROUP BY GROUPING SETS (country, genre);
```

```
SELECT country,
       genre,
       COUNT(*)
FROM renting_extended
GROUP BY country, genre;
```

- Count movie rentals for each unique combination of country and genre.
- Expression in `GROUPING SETS` : `(country, genre)`

```
| country | genre  | count  |
|---------|--------|--------|
| Austria | Comedy | 2      |
| Belgium | Drama  | 15     |
| Austria | Drama  | 4      |
| Belgium | Comedy | 1      |
```

GROUPING SETS and GROUP BY queries

```
SELECT country, COUNT(*)
FROM renting_extended
GROUP BY GROUPING SETS (country);
```

```
SELECT country, COUNT(*)
FROM renting_extended
GROUP BY country;
```

- Count movie rentals for each country.
- Expression in `GROUPING SETS` : `(country)`

```
| country | count |
|---------|-------|
| Austria | 16    |
| Belgium | 6     |
```

GROUPING SETS and GROUP BY queries

```
SELECT genre, COUNT(*)
FROM renting_extended
GROUP BY GROUPING SETS (genre);
```

```
SELECT genre, COUNT(*)
FROM renting_extended
GROUP BY genre;
```

- Count movie rentals for each genre.
- Expression in `GROUPING SETS` : `(genre)`

```
| country | count |
|---------|-------|
| Comedy  | 3     |
| Drama   | 19    |
```

GROUPING SETS and GROUP BY queries

```
SELECT COUNT(*)
FROM renting_extended
GROUP BY GROUPING SETS ();
```

- Total aggregation - count all movie rentals.
- Expression in `GROUPING SETS` : `()`

```
SELECT COUNT(*)
FROM renting_extended;
```

```
| count |
|-------|
| 22    |
```

Notation for GROUP BY GROUPING SETS

- `GROUP BY GROUPING SETS (...)`

```
SELECT country, genre, COUNT(*)
FROM renting_extended
GROUP BY GROUPING SETS ((country, genre), (country), (genre), ());
```

- `UNION` over 4 previous queries.
- Combine all information of a pivot table in one query.
- This query is equivalent to `GROUP BY CUBE (country, genre)`.

Result with GROUPING SETS operator

```
SELECT country, genre, COUNT(*)
FROM renting_extended
GROUP BY GROUPING SETS ((country, genre), (country), (genre), ());
```

```
| country | genre  |count|
|---------|--------|-----|
| NULL    | NULL   | 22  |
| Austria | Comedy | 2   |
| Belgium | Drama  | 15  |
| Austria | Drama  | 4   |
| Belgium | Comedy | 1   |
| Belgium | NULL   | 16  |
| Austria | NULL   | 6   |
| NULL    | Comedy | 3   |
| NULL    | Drama  | 19  |
```

Calculate number of rentals and average rating

- Combine only selected aggregations:
  - country and genre
  - genre
- Use the number of movie rentals and the average ratings for aggregation.

```
SELECT country,
       genre,
       COUNT(*),
       AVG(rating) AS avg_rating
FROM renting_extended
GROUP BY GROUPING SETS ((country, genre), (genre));
```

Calculate number of rentals and average rating

```
SELECT country, genre, COUNT(*), AVG(rating) AS avg_rating
FROM renting_extended
GROUP BY GROUPING SETS ((country, genre), (genre));
```

```
| country | genre   | count| avg_rating |
|---------|---------|------|------------|
| Austria | Comedy | 2    | 8.00       |
| Belgium | Drama   | 15   | 9.17       |
| Austria | Drama   | 4    | 6.00       |
| Belgium | Comedy | 1    | NULL       |
| NULL    | Comedy | 3    | 8.00       |
| NULL    | Drama   | 19   | 8.38       |
```

Let's practice!

DATA-DRIVEN DECISION MAKING IN SQL

Final example

DATA-DRIVEN DECISION MAKING IN SQL

Tim Verdonck
Professor Statistics and Data Science

datacamp

Business Case

- MovieNow considers to invest money in new movies.
- It is more expensive for MovieNow to make movies available which were recently produced than older ones.
- First step of data analysis:
  - Do customers give better ratings to movies which were recently produced than to older ones?
  - Is there a difference across countries?

1. Join data

- Information needed:
  - `renting` records of movie rentals with ratings
  - `customers` information about country of the customer
  - `movies` year of release of the movie

```
SELECT *
FROM renting AS r
LEFT JOIN customers AS c
ON c.customer_id = r.customer_id
LEFT JOIN movies AS m
ON m.movie_id = r.movie_id;
```

2. Select relevant records

- Use only records of movies with at least 4 ratings
- Use only records of movie rentals since 2018-04-01

```
SELECT *
FROM renting AS r
LEFT JOIN customers AS c
ON c.customer_id = r.customer_id
LEFT JOIN movies AS m
ON m.movie_id = r.movie_id
WHERE r.movie_id IN (
    SELECT movie_id
    FROM renting
    GROUP BY movie_id
    HAVING COUNT(rating) >= 4)
AND r.date_renting >= '2018-04-01';
```

3. Aggregation

Type of aggregation:

- Count the number of movie rentals
- Count the number of different movies
- Calculate the average rating

Levels of aggregation:

- Total aggregation
- For movies by year of release
- For movies by year of release separately for the country of the customers

3. Aggregation

```
SELECT c.country,
       m.year_of_release,
       COUNT(*) AS n_rentals,
       COUNT(DISTINCT r.movie_id) AS n_movies,
       AVG(rating) AS avg_rating
FROM renting AS r
LEFT JOIN customers AS c
ON c.customer_id = r.customer_id
LEFT JOIN movies AS m
ON m.movie_id = r.movie_id
WHERE r.movie_id IN (
    SELECT movie_id
    FROM renting
    GROUP BY movie_id
    HAVING COUNT(rating) >= 4)
AND r.date_renting >= '2018-04-01'
GROUP BY ROLLUP (m.year_of_release, c.country)
ORDER BY c.country, m.year_of_release;
```

Resulting table

year_of_release	country	n_rentals	n_movies	avg_rating
2009	null	10	1	8.750000000000000
2010	null	41	5	7.962962962962630
2011	null	14	2	8.222222222222222
2012	null	28	5	8.111111111111111
2013	null	10	2	7.600000000000000
2014	null	5	1	8.000000000000000
null	null	333	50	7.902439024390243

Let's practice!

DATA-DRIVEN DECISION MAKING IN SQL