P2 Assignment: Analytic SQL

ISM 6208 Data Warehousing

Dalton Anderson, Kevin Hitt

### Query 1: Aggregations with CUBE and ROLLUP

The use of ROLLUP and CUBE here allows us to analyze crime data at different levels of aggregation. ROLLUP provides subtotals and grand totals for crimes by type and year, while CUBE goes further by considering all possible combinations. By utilizing ROLLUP, we might be able to see patterns/trends otherwise unnoticeable without the aggregation. This provides a useful tool for understanding the relationships between crime types and their occurrences over different years.

-- Use **ROLLUP** to get aggregate of crimes by type by year:

SELECT YEAR\_ONLY, CRIMETYPE, COUNT(\*) as TOTAL\_CRIMES

FROM PEARL\_CHICAGO

GROUP BY ROLLUP (YEAR\_ONLY, CRIMETYPE);

A screenshot of a computer

Description automatically generated with medium confidence

-- USE **CUBE** to get crimes per year per arrest status,

-- as well as the total number of crimes per location description

SELECT YEAR\_ONLY, ARREST, LOCATION\_DESCRIPTION, COUNT(\*) as TOTAL\_CRIMES

FROM PEARL\_CHICAGO

GROUP BY CUBE (YEAR\_ONLY, ARREST), LOCATION\_DESCRIPTION

ORDER BY TOTAL\_CRIMES DESC;

A screenshot of a computer

Description automatically generated with low confidence

-- Use both ROLLUP and CUBE to get the total crimes per district,

-- per description of the crime, and across all descriptions and districts:

SELECT DISTRICT, DESCRIPTION, COUNT(\*) as TOTAL\_CRIMES

FROM PEARL\_CHICAGO

GROUP BY ROLLUP (DISTRICT), CUBE (DESCRIPTION)

ORDER BY TOTAL\_CRIMES DESC;

A screenshot of a computer

Description automatically generated with low confidence

### Query 2: Computing RANKs

While the sparseness of the Chicago Crime data doesn’t lend itself to an obvious comparison of RANK v. DENSE\_RANK, here we will consider a common student assessment example. We can compute the RANK of each student based on their scores. The student with the highest score will have a RANK of 1; the next highest will have a RANK of 2, and so on.

SELECT student\_name, score,

RANK() OVER (ORDER BY score DESC) AS rank,

DENSE\_RANK() OVER (ORDER BY score DESC) AS dense\_rank

FROM student\_scores;

student\_name | score | rank | dense\_rank

-------------\*-------\*------\*-----------

John | 95 | 1 | 1

Emily | 90 | 2 | 2

David | 88 | 3 | 3

Sophia | 88 | 3 | 3

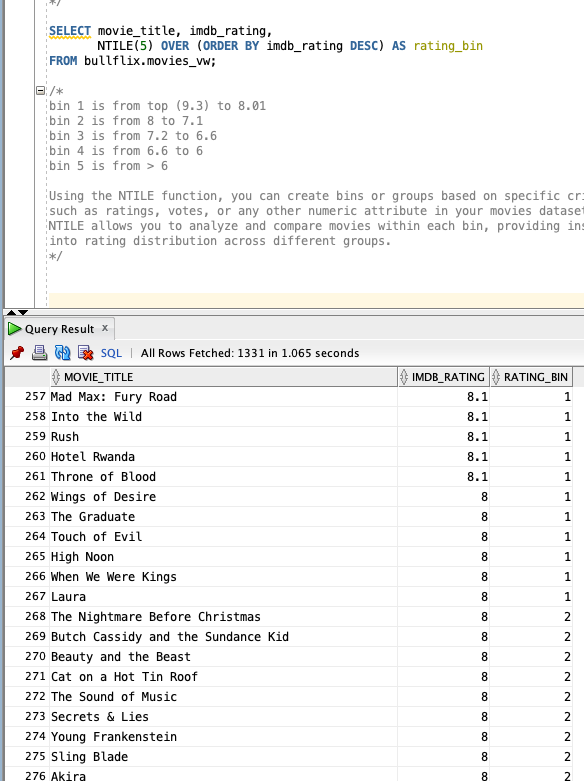
Michael | 85 | 5 | 4

### The main difference here between RANK() and DENSE\_RANK() is how they handle ties (cases where multiple rows have the same value). The RANK() function leaves gaps in the ranking sequence when there are ties [Sophia -> Michael], while the DENSE\_RANK() function assigns consecutive ranks to tied rows without any gaps [Sophia -> Michael].

### Query 3: Creating Bins with NTILE

Creating bins with NTILE is a technique in SQL that allows you to divide a dataset into equal-sized groups or compartments based on a specified criterion. It is beneficial for analyzing data distribution and performing data segmentation. For example, let's consider a table of products with their prices. We can use NTILE to create bins or groups of products based on their price range. This can help identify price segments or categories for further analysis or marketing strategies.

Here's an example query that creates bins using NTILE:



/\*In this query, the NTILE function is applied to the imdb\_rating column

The NTILE(5) specifies that the ratings will be divided into 5 equal-sized bins.

The ORDER BY imdb\_rating DESC orders the movies based on their IMDb ratings in descending order.

\*/

SELECT movie\_title, imdb\_rating,

NTILE(5) OVER (ORDER BY imdb\_rating DESC) AS rating\_bin

FROM bullflix.movies\_vw;

/\*

bin 1 is from top (9.3) to 8.01

bin 2 is from 8 to 7.1

bin 3 is from 7.2 to 6.6

bin 4 is from 6.6 to 6

bin 5 is from > 6

Using the NTILE function, you can create bins or groups based on specific criteria,

such as ratings, votes, or any other numeric attribute in your dataset.

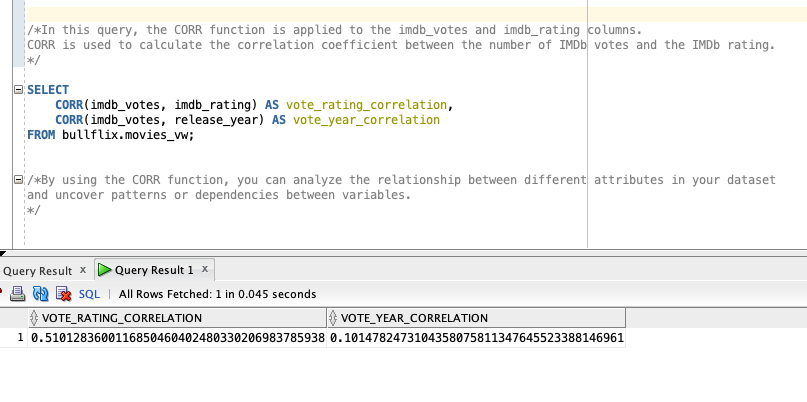
NTILE allows you to analyze and compare movies within each bin, providing insights across different groups.

\*/

### Query 4: Correlations (CORR)

Correlations (CORR) in SQL is a powerful statistical function that allows you to measure the strength and direction of the linear relationship between two numeric variables in a dataset. It provides valuable insights into the dependency and association between variables. For example, consider a table with data on housing prices and square footage. Using the CORR function, we can determine the correlation coefficient between these two variables, which indicates how closely they are related.

Here's an example query that calculates the correlation coefficient:



/\*In this query, the CORR function is applied to the imdb\_votes and imdb\_rating columns.

CORR is used to calculate the correlation coefficient between the number of IMDb votes and the IMDb rating.

\*/

SELECT

CORR(imdb\_votes, imdb\_rating) AS vote\_rating\_correlation,

CORR(imdb\_votes, release\_year) AS vote\_year\_correlation

FROM bullflix.movies\_vw;

/\*By using the CORR function, you can analyze the relationship between different attributes in your dataset

and uncover patterns or dependencies between variables.

\*/

The results align with expectations where the year is a limited influence as movies are timeless. In comparison, the amount of votes a movie receives is moderately correlated with the rating outcome of the film. For a studio, an important question is whether X person is worth the money. A studio can look at the correlation between Quentin Tarantino and his movies.