**SQL HACKATHON EXTRA QUESTIONS OCT 2023(50 Queries)**

1. Can you find patients from the demographics table whose last name contains ‘an’ Provide their patient IDs and the position where ‘an’ appears in their last name. What insights can be derived from this information?

**Function Used : STRPOS()**

**This SQL query identifies the position of the substring 'an' within the 'lastname' column for patients whose last names contain 'an'. It uses the STRPOS() function to perform this task efficiently.**

**QUERY**

| **SELECT**  **patientid,**  **STRPOS(lastname, 'an') AS mc\_position *-- Calculate the position of 'an' in the last name***  **FROM**  **demographics**  **WHERE**  **STRPOS(lastname, 'an') > 0 ; *-- Filter records where 'an' is found in the last name*** |
| --- |

**OUTPUT**

**A screenshot of a computer

Description automatically generated**

1. Can you generate a report that includes a list of patients from the demographics table with their full names, where each name is formatted as "Last Name, First Name," and the patient ID is left-padded with zeros to a total length of 8 characters? Additionally, calculate the average age of these patients. How can this report be used for patient identification and analysis?

**Function Used : LPAD()**

**The LPAD() function is used to left-pad patient IDs with zeros to ensure a consistent 8-character length in the report.**

**QUERY:**

| ***-- This SQL query generates a report with patient information, excluding rows with NULL values.***  **SELECT**  **LPAD(patientid::text, 8, '0') AS padded\_patientid,**  **CONCAT\_WS(', ', lastname, firstname) AS full\_name,**  **ROUND(AVG(EXTRACT(YEAR FROM CURRENT\_DATE) - EXTRACT(YEAR FROM dob))) AS average\_age**  **FROM**  **demographics**  **WHERE**  **patientid IS NOT NULL *-- Exclude rows where 'patientid' is NULL***  **AND lastname IS NOT NULL *-- Exclude rows where 'lastname' is NULL***  **AND firstname IS NOT NULL *-- Exclude rows where 'firstname' is NULL***  **AND dob IS NOT NULL *-- Exclude rows where 'dob' is NULL***  **GROUP BY**  **padded\_patientid, full\_name;** |
| --- |

**OUTPUT**

A screenshot of a computer

Description automatically generated

1. Can you retrieve a list of food items from the foodlog table, where the item's name is right-padded with underscores to a minimum length of 20 characters? Additionally, count the occurrences of each unique padded food item in the dataset. How can this information help in analyzing food preferences?

**Function Used: RPAD()**

**The RPAD() function is used to pad the 'logged\_food' column with underscores, ensuring a consistent length for comparison.This query is useful for analyzing the frequency of different food items logged in the 'foodlog' table.**

**QUERY:**

| ***-- Analyzes food log data, counts food item occurrences***  ***-- Pads 'logged\_food' with underscores for uniformity***  **SELECT**  **RPAD(logged\_food, 20, '\_') AS padded\_food\_item, *-- Pad food items with underscores***  **COUNT(\*) AS item\_count *-- Count occurrences of each padded food item***  **FROM**  **foodlog**  **GROUP BY**  **padded\_food\_item *-- Group results by padded food item***  **ORDER BY**  **item\_count DESC; *-- Sort results by item count in descending order*** |
| --- |

**OUTPUT**

A screenshot of a computer

Description automatically generated

1. Can you find all patients from the demographics table whose first names contain any of the specified substrings ('John', 'David', 'Sarah')? Display their full names, first names, and the matched substring.

**Function Used: CONCAT\_WS() and POSITION()**

**This SQL query uses the CONCAT\_WS() function to concatenate the 'firstname' and 'lastname' columns from the 'demographics' table, separated by a space (' '), resulting in a 'full\_name' column. Additionally, it utilizes the POSITION() function to check for the presence of specific first names ('John', 'David', or 'Sarah') within the 'firstname' column.**

**QUERY:**

| *-- Select full names and first names where the first name contains 'John', 'David', or 'Sarah'.*  **SELECT**  CONCAT\_WS(' ', d.firstname, d.lastname) **AS** full\_name,  d.firstname  **FROM**  demographics **AS** d  **WHERE**  **POSITION**('John' **IN** d.firstname) > 0  **OR** **POSITION**('David' **IN** d.firstname) > 0  **OR** **POSITION**('Sarah' **IN** d.firstname) > 0; |
| --- |

**OUTPUT**

A screenshot of a chat

Description automatically generated

1. Can you analyze the relationship between the last two digits of patients' birth years (DOB) and their mean heart rate (mean\_hr) from the "demographics" and "hr" tables? Specifically, we want to know if there's any correlation between the last two digits of birth years and mean heart rates. Additionally, can you identify any patterns or trends in the data?

**Function Used: RIGHT() and CORR()**

**The RIGHT() function is employed to extract the last two digits of birth years from the 'dob' (date of birth) column in the 'demographics' table. It ensures that only the relevant portion of the birth year is considered for analysis.**

**The CORR() function calculates the Pearson correlation coefficient, measuring the linear relationship between the last two digits of birth years and mean heart rates. This coefficient helps assess whether there is any significant correlation between these two variables in the dataset.**

**QUERY:**

| **-- Calculate the last two digits of birth years and retrieve mean heart rates**  **-- This query computes statistics related to patient birth years and mean heart rates.**  **WITH birth\_year\_last\_two\_digits AS (**  **SELECT**  **patientid,**  **RIGHT(EXTRACT(YEAR FROM dob)::TEXT, 2) AS last\_two\_digits\_of\_birth\_year**  **FROM public.demographics**  **),**  **mean\_heart\_rates AS (**  **SELECT**  **h.patientid,**  **AVG(h.mean\_hr) AS avg\_mean\_hr**  **FROM public.hr AS h**  **GROUP BY h.patientid**  **)**  **SELECT**  **byl2d.last\_two\_digits\_of\_birth\_year,**  **COUNT(\*) AS num\_patients,**  **ROUND(AVG(mhr.avg\_mean\_hr)::NUMERIC, 2) AS avg\_mean\_hr,**  **MAX(mhr.avg\_mean\_hr) AS max\_mean\_hr,**  **MIN(mhr.avg\_mean\_hr) AS min\_mean\_hr**  **FROM birth\_year\_last\_two\_digits AS byl2d**  **LEFT JOIN mean\_heart\_rates AS mhr ON byl2d.patientid = mhr.patientid**  **GROUP BY byl2d.last\_two\_digits\_of\_birth\_year**  **ORDER BY byl2d.last\_two\_digits\_of\_birth\_year;**  ***-- Calculate the Pearson correlation coefficient***  ***-- This query calculates the Pearson correlation coefficient between***  ***-- last two digits of birth years and mean heart rates.***  **WITH birth\_year\_last\_two\_digits AS (**  **SELECT**  **patientid,**  **RIGHT(EXTRACT(YEAR FROM dob)::TEXT, 2) AS last\_two\_digits\_of\_birth\_year**  **FROM public.demographics**  **),**  **mean\_heart\_rates AS (**  **SELECT**  **h.patientid,**  **AVG(h.mean\_hr) AS avg\_mean\_hr**  **FROM public.hr AS h**  **GROUP BY h.patientid**  **)**  **SELECT**  **CORR(**  **byl2d.last\_two\_digits\_of\_birth\_year::DOUBLE PRECISION,**  **mhr.avg\_mean\_hr::DOUBLE PRECISION**  **) AS pearson\_correlation\_coefficient**  **FROM birth\_year\_last\_two\_digits AS byl2d**  **LEFT JOIN mean\_heart\_rates AS mhr ON byl2d.patientid = mhr.patientid;** |
| --- |

**OUTPUT**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. Can you generate a report that shows the distribution of logged foods (logged\_food) from the "foodlog" table, where each food item is categorized by its first letter (A-Z) and the number of times each category appears in the data?

**Function Used: TRANSLATE() and UPPER()**

**TRANSLATE standardizes food item categories by converting them to uppercase, while UPPER() ensures uniformity in letter case for accurate categorization and counting of food items in the dataset.**

**QUERY:**

| ***-- Analyze food log data to categorize and count food items***  ***-- This query groups food items into categories by converting them to uppercase***  ***-- and then counts the occurrences of each category.***  **WITH food\_categories AS (**  **SELECT**  **TRANSLATE(UPPER(logged\_food), 'ABCDEFGHIJKLMNOPQRSTUVWXYZ', 'ABCDEFGHIJKLMNOPQRSTUVWXYZ') AS category,**  **COUNT(\*) AS category\_count**  **FROM public.foodlog**  **GROUP BY category**  **)**  **SELECT**  **category,**  **SUM(category\_count) AS total\_count *-- Sum the counts within each category***  **FROM food\_categories**  **GROUP BY category *-- Group results by category***  **ORDER BY category; *-- Sort results alphabetically by category*** |
| --- |

**OUTPUT**

A screenshot of a computer

Description automatically generated

1. How can we identify significant increases in heart rate for patients

**Function Used: LAG()**

**LAG window function used for tracking previous heart rates and employs a CASE statement to identify significant heart rate increases in patients' data.**

**QUERY:**

| ***-- Identify and track significant increases in heart rate for patients***  ***-- This query calculates fluctuations in mean heart rates, marks significant increases,***  ***-- and presents a structured view of patients' heart rate data.***  **WITH HeartRateFluctuations AS (**  **SELECT**  **patientid,**  **datestamp,**  **mean\_hr,**  **LAG(mean\_hr) OVER (PARTITION BY patientid ORDER BY datestamp) AS previous\_hr**  **FROM**  **public.hr**  **),**  **IncreasedHeartRateDays AS (**  **SELECT**  **patientid,**  **datestamp,**  **mean\_hr,**  **previous\_hr,**  **CASE**  **WHEN previous\_hr IS NOT NULL AND mean\_hr > 1.1 \* previous\_hr THEN 1**  **ELSE 0**  **END AS increased\_heart\_rate**  **FROM**  **HeartRateFluctuations**  **)**  **SELECT**  **patientid,**  **datestamp,**  **mean\_hr,**  **previous\_hr,**  **CASE**  **WHEN increased\_heart\_rate = 1 THEN 'Yes'**  **ELSE 'No'**  **END AS significant\_increase**  **FROM**  **IncreasedHeartRateDays**  **WHERE**  **increased\_heart\_rate = 1**  **ORDER BY**  **patientid, datestamp;** |
| --- |

**OUTPUT**

A screenshot of a computer

Description automatically generated

1. Can you calculate the average heart rate (mean\_hr) for each gender from the "demographics" and "hr" tables, and round it up to the nearest whole number.

**Function Used: CEIL()**

**The CEIL function is used to round up the average heart rates to whole numbers, improving the readability of gender-specific heart rate statistics.**

**QUERY**

| ***-- Calculate and round up the average heart rate for each gender***  ***-- This query computes the average heart rate for each gender by joining demographics and heart rate data,***  ***-- and then rounds up the result to the nearest whole number.***  **WITH avg\_heart\_rates AS (**  **SELECT d.gender, ceil(AVG(h.mean\_hr)) AS average\_heart\_rate**  **FROM public.demographics AS d**  **JOIN public.hr AS h ON d.patientid = h.patientid**  **GROUP BY d.gender**  **)**  **SELECT gender, average\_heart\_rate**  **FROM avg\_heart\_rates;** |
| --- |

**OUTPUT**

A screenshot of a chat

Description automatically generated

1. Concatenate logged food items for each patient into a single array

**Function Used: ARRAY\_AGG() AND UNNEST()**

**The ARRAY\_AGG function groups food log entries into arrays by patient, simplifying data organization. Meanwhile, the UNNEST function flattens these arrays, enabling the concatenation of individual food items into a single array, facilitating food consumption pattern analysis.**

**QUERY:**

| ***-- Concatenate logged food items for each patient into a single array***  ***-- This query aggregates individual food log entries into arrays for each patient***  ***-- and then concatenates those arrays into a single array, simplifying food item analysis.***  **WITH food\_arrays AS (**  **SELECT patientid, array\_agg(logged\_food) AS food\_items\_array**  **FROM public.foodlog**  **GROUP BY patientid**  **ORDER BY patientid**  **)**  **SELECT patientid, array\_agg(food\_items) AS concatenated\_food\_items**  **FROM (**  **SELECT patientid, unnest(food\_items\_array) AS food\_items**  **FROM food\_arrays**  **) subquery**  **GROUP BY patientid**  **ORDER BY patientid;** |
| --- |

**OUTPUT**

A screenshot of a computer

Description automatically generated

1. How can we efficiently process and present food log data, including both concatenating food items into comma-separated strings and converting them back into arrays for further analysis.

**Function Used: ARRAY\_TO\_STRING() and STRING\_TO\_ARRAY()**

**The ARRAY\_TO\_STRING function simplifies the presentation of food consumption data by converting arrays of logged food items into comma-separated strings, while the STRING\_TO\_ARRAY function reverses this process, allowing for more versatile analysis and manipulation of the data.**

**QUERY:**

| ***-- Retrieve logged food items as comma-separated strings for each patient***  ***-- This query compiles food log entries for each patient into comma-separated strings,***  ***-- simplifying the presentation of food consumption data.***  **SELECT patientid, array\_to\_string(array\_agg(logged\_food), ',') AS food\_items\_csv**  **FROM public.foodlog**  **GROUP BY patientid**  **ORDER BY patientid;**  ***-- Convert comma-separated strings back into arrays***  ***-- This query reverses the process, converting previously concatenated strings***  ***-- back into arrays for further analysis or manipulation.***  **WITH food\_arrays AS (**  **SELECT patientid, string\_to\_array(food\_items\_csv, ',') AS food\_items\_array**  **FROM (**  **SELECT patientid, array\_to\_string(array\_agg(logged\_food), ',') AS food\_items\_csv**  **FROM public.foodlog**  **GROUP BY patientid**  **) subquery**  **)**  **SELECT patientid, food\_items\_array**  **FROM food\_arrays**  **ORDER BY patientid;** |
| --- |

**Array\_to\_string() output:**

A screenshot of a computer

Description automatically generated

**String\_to\_array() output:**

A screenshot of a computer

Description automatically generated

1. **ANALYSIS QUESTION**: "How can we ensure reproducibility when selecting a random sample of 10 patients from the 'demographics' table, while also excluding records with NULL values in the 'patientid,' 'firstname,' and 'lastname' columns?"

**Function Used: SETSEED()**

**The SETSEED function ensures reproducibility by setting a specific seed value (0.42) for random number generation.**

**In the second query, the RANDOM function orders records randomly to create a sample of 10 patients, excluding records with NULL values in key columns for data integrity.**

**QUERY:**

| ***-- Set the seed for reproducibility***  ***-- This query initializes a seed value of 0.42 for random number generation, ensuring that subsequent random operations are reproducible.***  **SELECT setseed(0.42);**  ***-- Generate a random sample of 10 patients from the "demographics" table, excluding NULL values***  ***-- This query selects a random sample of 10 patients from the "demographics" table,***  ***-- ensuring that records with NULL values in the 'patientid,' 'firstname,' or 'lastname' columns are excluded.***  **SELECT patientid, firstname, lastname**  **FROM public.demographics**  **WHERE patientid IS NOT NULL**  **AND firstname IS NOT NULL**  **AND lastname IS NOT NULL**  **ORDER BY random()**  **LIMIT 10;** |
| --- |

**OUTPUT**

A screenshot of a computer

Description automatically generated

1. "What is the patient's glucose level (glucose\_value\_mgdl) at the 75th percentile (3rd quartile) in the provided medical dataset, and how does it vary based on gender and age group?"

**Function Used: PERCENTILE\_DISC and CASE statement**

**The CASE statement efficiently categorizes patients into age groups. The PERCENTILE\_DISC function calculates the 75th percentile of glucose values, aiding glucose level analysis by age group and gender.**

**QUERY:**

| ***-- Categorize patients into age groups, calculate 75th percentile of glucose values***  ***-- This query categorizes patients into age groups (0-20, 21-30, 31-40, and 41+),***  ***-- and calculates the 75th percentile of glucose values for each age group and gender.***  **SELECT**  **CASE**  **WHEN EXTRACT(YEAR FROM age(dob))::integer <= 20 THEN '0-20'**  **WHEN EXTRACT(YEAR FROM age(dob))::integer <= 30 THEN '21-30'**  **WHEN EXTRACT(YEAR FROM age(dob))::integer <= 40 THEN '31-40'**  **ELSE '41+'**  **END AS age\_group,**  **gender,**  **PERCENTILE\_DISC(0.75) WITHIN GROUP (ORDER BY glucose\_value\_mgdl) AS percentile\_75th**  **FROM**  **demographics**  **JOIN**  **dexcom ON demographics.patientid = dexcom.patientid**  **GROUP BY**  **age\_group, gender**  **ORDER BY**  **age\_group, gender;** |
| --- |

**OUTPUT**

A screenshot of a chat

Description automatically generated

1. What is the relationship between the variability in glucose levels (sample variance) and the average glucose values among different age groups and genders in the provided medical dataset, and how can this information be leveraged to improve healthcare outcomes?

**Function Used: VAR\_SAMP()**

**The VAR\_SAMP function computes the sample variance of glucose values within each age group and gender, while the AVG function calculates the average glucose value.**

**QUERY:**

| ***-- Calculate glucose statistics by age group and gender***  ***-- This query computes glucose statistics, including sample variance and average values, for patients grouped by age and gender.***  **WITH glucose\_stats AS (**  **SELECT**  **CASE**  **WHEN EXTRACT(YEAR FROM age(demographics.dob)) <= 20 THEN '0-20'**  **WHEN EXTRACT(YEAR FROM age(demographics.dob)) <= 30 THEN '21-30'**  **WHEN EXTRACT(YEAR FROM age(demographics.dob)) <= 40 THEN '31-40'**  **ELSE '41+'**  **END AS age\_group,**  **demographics.gender,**  **VAR\_SAMP(dexcom.glucose\_value\_mgdl) AS sample\_glucose\_variance,**  **AVG(dexcom.glucose\_value\_mgdl) AS average\_glucose\_value**  **FROM**  **demographics**  **JOIN**  **dexcom ON demographics.patientid = dexcom.patientid**  **WHERE**  **demographics.dob IS NOT NULL**  **GROUP BY**  **age\_group, demographics.gender**  **ORDER BY**  **age\_group, demographics.gender**  **)**  **SELECT**  **age\_group,**  **gender,**  **sample\_glucose\_variance,**  **average\_glucose\_value**  **FROM**  **glucose\_stats;** |
| --- |

**OUTPUT**

A screenshot of a computer

Description automatically generated

1. Is there a significant difference in the population variances of glucose levels (glucose\_value\_mgdl) between male and female patients in the provided medical dataset? We want to determine if there is a statistically significant difference in glucose level variability between these gender groups

**Function Used: VAR\_POP()**

**The VAR\_POP function calculates the population variance of glucose values within each gender, providing a measure of glucose level variability for analysis.**

**QUERY:**

| ***-- Calculate population variance of glucose values by gender***  ***-- This query computes the population variance of glucose values for each gender.***  **WITH gender\_glucose\_variance AS (**  **SELECT**  **gender,**  **VAR\_POP(glucose\_value\_mgdl) AS population\_variance**  **FROM**  **demographics**  **JOIN**  **dexcom ON demographics.patientid = dexcom.patientid**  **GROUP BY**  **gender**  **)**  **SELECT**  **gender,**  **population\_variance**  **FROM**  **gender\_glucose\_variance**  **WHERE**  **population\_variance IS NOT NULL; *-- Exclude records with NULL variance values*** |
| --- |

**OUTPUT**

A screenshot of a computer

Description automatically generated

1. Is there a significant relationship between a patient's mean heart rate (mean\_hr) and their recorded blood glucose levels (glucose\_value\_mgdl) in the provided medical dataset?

**Function Used: COVAR\_POP()**

**The COVAR\_POP function quantifies the population-level covariance between mean heart rates and glucose values, providing insights into the degree and direction of their statistical association across patients. COVAR\_POP IS POSITIVE HENCE HAS A POSITIVE LINEAR RELATIONSHIP**

**QUERY:**

| ***-- Calculate the population covariance between mean heart rates and glucose values***  ***-- This query computes the population covariance, measuring the relationship between mean heart rates and glucose values for patients.***  **SELECT**  **COVAR\_POP(mean\_hr, glucose\_value\_mgdl) AS population\_covariance**  **FROM**  **hr**  **JOIN**  **dexcom**  **ON**  **hr.patientid = dexcom.patientid;** |
| --- |

**OUTPUT**

A screen shot of a message

Description automatically generated

1. What are the most common starting substrings (e.g., the first three characters) of food item names logged by patients in the foodlog dataset, and how frequently do they appear

**Function Used: SUBSTRING()**

**The SUBSTRING function extracts the initial three characters of food items, allowing the analysis of common food categories based on their starting characters, thus revealing potential patterns in food choices.**

**QUERY:**

| ***-- Analyze food log data by examining the starting three characters of food items***  ***-- This query extracts the first three characters of food items and calculates their frequency,***  ***-- providing insights into common food item categories based on their initial characters.***  **SELECT**  **SUBSTRING(logged\_food FROM 1 FOR 3) AS starting\_substring,**  **COUNT(\*) AS frequency**  **FROM foodlog**  **GROUP BY starting\_substring**  **ORDER BY frequency DESC;** |
| --- |

**OUTPUT**

A screenshot of a computer

Description automatically generated

1. How can we efficiently replace spaces with underscores in logged food item names for patients.

**Function Used: REPLACE()**

**The REPLACE function efficiently substitutes spaces with underscores in food item names, ensuring uniformity and improving data legibility.**

**QUERY:**

| ***-- Replace spaces with underscores in logged food item names for patients***  ***-- This query transforms food item names by replacing spaces with underscores, enhancing data consistency and readability.***  **SELECT**  **patientid,**  **REPLACE(logged\_food, ' ', '\_') AS food\_with\_underscore**  **FROM foodlog;** |
| --- |

**OUTPUT**

A screenshot of a computer

Description automatically generated

1. Find the total number of logged food items for each patient

**Function Used: ARRAY\_LENGTH()**

**The ARRAY\_LENGTH function efficiently counts the total number of food items logged for each patient by aggregating food items into an array and determining the array's length, providing a concise summary of patient food log data.**

**QUERY:**

| ***-- Calculate the total number of food items logged for each patient***  ***-- This query aggregates and counts the total number of food items logged for each patient.***  **SELECT**  **patientid,**  **ARRAY\_LENGTH(ARRAY\_AGG(logged\_food), 1) AS total\_food\_items\_logged**  **FROM foodlog**  **GROUP BY patientid**  **ORDER BY patientid;** |
| --- |

**OUTPUT**

A screenshot of a chat

Description automatically generated

1. Calculate the number of days since the last recorded event for each patient in the dexcom table.

**Function Used: NOW()**

**The NOW() function retrieves the current date and time, allowing the calculation of the maximum number of days elapsed since the last event date for each patient in the "dexcom" dataset, which provides insights into patient activity or event frequency.**

**QUERY:**

| ***-- Calculate the days since the last event for each patient in the "dexcom" dataset***  ***-- This query finds the maximum days elapsed since the last event date for each patient.***  **SELECT**  **d.patientid,**  **EXTRACT(DAY FROM NOW() - MAX(d.last\_event\_date))::INTEGER AS days\_since\_last\_event**  **FROM**  **(SELECT**  **patientid,**  **MAX(datestamp) AS last\_event\_date**  **FROM**  **dexcom**  **GROUP BY**  **patientid) AS d**  **GROUP BY**  **d.patientid;** |
| --- |

**OUTPUT**

A screenshot of a computer

Description automatically generated

1. What is the most common event type recorded for each patient, and how does it correlate with their HbA1c levels.

**Function Used: MODE() AND CORR()**

**The query calculates the most common event type for each patient and the correlation between their HbA1c levels and event counts. It uses the mode() function to find the most common event type within each patient's data and the CORR() function to calculate the correlation between HbA1c levels and event counts.**

**QUERY:**

| ***-- Common Table Expression (CTE) to calculate event counts for each patient***  **WITH EventCounts AS (**  **SELECT**  **d.patientid,**  **COUNT(e.event\_type) AS event\_count *-- Count the events for each patient***  **FROM**  **demographics d**  **LEFT JOIN**  **dexcom dx ON d.patientid = dx.patientid *-- Join demographics and dexcom tables***  **LEFT JOIN**  **eventtype e ON dx.eventid = e.id *-- Join with the eventtype table to get event types***  **GROUP BY**  **d.patientid, d.hba1c *-- Group by patient and their HbA1c levels***  **)**  **SELECT**  **ec.patientid,**  **mode() WITHIN GROUP (ORDER BY e.event\_type) AS most\_common\_event\_type, *-- Calculate the mode (most common event type)***  **CORR(d.hba1c, ec.event\_count::numeric) AS correlation\_hba1c\_event\_count *-- Calculate the correlation between HbA1c and event counts***  **FROM**  **EventCounts ec**  **LEFT JOIN**  **demographics d ON ec.patientid = d.patientid *-- Join the CTE with demographics to get patient details***  **LEFT JOIN**  **dexcom dx ON ec.patientid = dx.patientid *-- Join with dexcom to get event details***  **LEFT JOIN**  **eventtype e ON dx.eventid = e.id *-- Join with eventtype to get event types***  **GROUP BY**  **ec.patientid, d.hba1c; *-- Group by patient and their HbA1c levels*** |
| --- |

**OUTPUT**

A screenshot of a computer

Description automatically generated

There is no correlation here.

1. How many patients have their birthdays in each month, and what is the total count of patients born in each month?

**Function Used: TO\_CHAR()**

**This query calculates and presents the count of patients born in each month, utilizing the TO\_CHAR() function to extract both the full month name and the month number from the dob column. It excludes records with NULL birth dates and orders the results by the month number for clarity.**

**QUERY:**

| ***-- Calculate the number of patients born in each month***  **SELECT**  **TO\_CHAR(dob, 'Month') AS birth\_month, *-- Full month name***  **TO\_CHAR(dob, 'MM') AS month\_number, *-- Month number***  **COUNT(\*) AS patient\_count *-- Count of patients***  **FROM**  **public.demographics**  **WHERE**  **dob IS NOT NULL *-- Exclude NULL birth dates***  **GROUP BY**  **birth\_month, month\_number *-- Group by month name and number***  **ORDER BY**  **month\_number; *-- Order by month number*** |
| --- |

**OUTPUT**

A screenshot of a computer

Description automatically generated

1. With respect to foodlog table, We need to perform several operations on this data using regular expressions and string functions. Please explain how we can achieve the following tasks using SQL queries:

* Find all rows where the logged\_food column contains the word 'cheese' in a case-insensitive manner. Additionally, list all matches of 'cheese' within each food item.
* Replace all occurrences of the word 'son' with 'daughter' in the logged\_food column while preserving case sensitivity.
* Extract the first word from the logged\_food column.
* Determine the position of the substring 'cheese' in the logged\_food column (case-insensitive

**Function Used: REGEXP\_MACHES(), REGEXP\_REPLACE, REGEXP\_SUBSTR(),POSITION()**

**SET RETURNING FUNCTIONS IN POSTGRES –**

**REGEXP\_MATCHES(logged\_food, 'cheese', 'i'):**

**Finds all case-insensitive occurrences of 'cheese' in logged\_food.**

**REGEXP\_REPLACE(logged\_food, 'son', 'daughter', 'gi'):**

**Replaces 'son' with 'daughter' (case-insensitive) in logged\_food.**

**REGEXP\_SUBSTR(logged\_food, '\w+'):**

**Extracts the first word from logged\_food.**

**POSITION('cheese' IN LOWER(logged\_food)):**

**Locates the position of 'cheese' (case-insensitive) in logged\_food**

**QUERY:**

| **WITH ComplexRegexQuery AS (**  **SELECT**  **patientid,**  **logged\_food,**  ***-- Use REGEXP\_MATCHES to find all food items containing 'cheese' (case-insensitive)***  **REGEXP\_MATCHES(logged\_food, 'cheese', 'i') AS cheese\_matches,**  ***-- Use REGEXP\_REPLACE to replace 'son' with 'daughter' in logged\_food***  **REGEXP\_REPLACE(logged\_food, 'son', 'daughter', 'gi') AS modified\_logged\_food,**  ***-- Use REGEXP\_SUBSTR to extract the first word from logged\_food***  **REGEXP\_SUBSTR(logged\_food, '\w+') AS first\_word,**  ***-- Use POSITION to find the position of 'cheese' in logged\_food***  **POSITION('cheese' IN LOWER(logged\_food)) AS cheese\_position**  **FROM**  **public.foodlog**  **WHERE**  ***-- Filter rows where 'cheese' (case-insensitive) is present in logged\_food***  **EXISTS (SELECT 1 FROM unnest(REGEXP\_MATCHES(logged\_food, 'cheese', 'i')) AS m WHERE m ~\* 'cheese')**  **)**  **SELECT**  **patientid,**  **logged\_food,**  **cheese\_matches,**  **modified\_logged\_food,**  **first\_word,**  **cheese\_position**  **FROM**  **ComplexRegexQuery;** |
| --- |

**OUTPUT**

A screenshot of a computer

Description automatically generated

1. Analyze the foodlog data to determine the length of logged\_food entries for each patient while also identifying if there are any leading or trailing spaces in those entries?"

**Function Used: TRIM(), RTRIM(), LTRIM(),LENGTH()**

**-TRIM(logged\_food): Removes leading and trailing spaces from the logged\_food column.**

**-RTRIM(logged\_food): Removes trailing spaces from the logged\_food column.**

**-LTRIM(logged\_food): Removes leading spaces from the logged\_food column.**

**-LENGTH(logged\_food): Calculates the length of the logged\_food entry.**

**The query also includes conditions to detect leading and trailing spaces based on the comparison of entry length with trimmed, rtrimmed, and ltrimmed lengths.**

**QUERY:**

| **WITH FoodlogEntryAnalysis AS (**  **SELECT**  **patientid,**  **logged\_food,**  **LENGTH(logged\_food) AS entry\_length,**  **LENGTH(TRIM(logged\_food)) AS trimmed\_length,**  **LENGTH(RTRIM(logged\_food)) AS rtrimmed\_length,**  **LENGTH(LTRIM(logged\_food)) AS ltrimmed\_length,**  **CASE**  **WHEN LENGTH(logged\_food) > LENGTH(TRIM(logged\_food)) THEN 'Leading spaces detected'**  **ELSE 'No leading spaces'**  **END AS leading\_space\_status,**  **CASE**  **WHEN LENGTH(logged\_food) > LENGTH(RTRIM(logged\_food)) THEN 'Trailing spaces detected'**  **ELSE 'No trailing spaces'**  **END AS trailing\_space\_status**  **FROM**  **public.foodlog**  **)**  **SELECT**  **patientid,**  **logged\_food,**  **entry\_length,**  **trimmed\_length,**  **rtrimmed\_length,**  **ltrimmed\_length,**  **leading\_space\_status,**  **trailing\_space\_status**  **FROM**  **FoodlogEntryAnalysis;** |
| --- |

**OUTPUT**

A screenshot of a computer

Description automatically generated

1. Analyze the foodlog data to extract and format the food item names in a reversed order, and then split them to identify specific components, while also calculating the average length of food item names

**Function Used: REVERSE() and SPLIT\_PART()**

**REVERSE(logged\_food): Reverses the characters in the logged\_food column.**

**SPLIT\_PART(REVERSE(logged\_food), ',', N): Extracts the last component of the reversed logged\_food column using a comma (',') as the delimiter. LENGTH(logged\_food): Calculates the length of the logged\_food column.**

**QUERY:**

| ***-- Create a common table expression (CTE) for food item analysis.***  **WITH FoodItemAnalysis AS (**  **SELECT**  **patientid,**  **logged\_food,**  **REVERSE(logged\_food) AS reversed\_food,**  **SPLIT\_PART(REVERSE(logged\_food), ',', 1) AS last\_component,**  **SPLIT\_PART(REVERSE(logged\_food), ',', 2) AS second\_last\_component,**  **SPLIT\_PART(REVERSE(logged\_food), ',', 3) AS third\_last\_component,**  **LENGTH(logged\_food) AS food\_length**  **FROM**  **public.foodlog**  **)**  **SELECT**  **patientid,**  **logged\_food,**  **reversed\_food,**  **last\_component,**  **second\_last\_component,**  **third\_last\_component,**  **AVG(food\_length) AS average\_food\_length**  **FROM**  **FoodItemAnalysis**  **WHERE**  **last\_component IS NOT NULL**  **AND second\_last\_component IS NOT NULL**  **GROUP BY**  **patientid,**  **logged\_food,**  **reversed\_food,**  **last\_component,**  **second\_last\_component,**  **third\_last\_component;** |
| --- |

**OUTPUT**

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

1. Find the top 5 patients with the highest cumulative distribution of glucose values in the public.dexcom table, along with their demographics

**Function Used: CUME\_DIST()**

**CUME\_DIST() window function USED TO CALCULATE CUMULATIVE FREQUENCY OF GLUCOSE VALUES.**

**QUERY:**

| ***-- Calculate the maximum glucose value and cumulative distribution for each patient***  **WITH GlucoseCumulativeDistribution AS (**  **SELECT**  **d.patientid,**  **MAX(d.glucose\_value\_mgdl) AS max\_glucose\_value,**  ***-- Calculate the cumulative distribution based on max glucose values***  **CUME\_DIST() OVER (ORDER BY MAX(d.glucose\_value\_mgdl) DESC) AS cumulative\_distribution**  **FROM**  **public.dexcom AS d**  **GROUP BY**  **d.patientid**  **)**  ***-- Select patient demographics for the top 5 patients with the highest cumulative distribution***  **SELECT**  **g.patientid,**  **g.cumulative\_distribution,**  **dm.firstname,**  **dm.lastname,**  **dm.gender,**  **dm.dob**  **FROM**  **GlucoseCumulativeDistribution AS g**  ***-- Join with demographics table to get patient information***  **JOIN**  **public.demographics AS dm ON g.patientid = dm.patientid**  ***-- Filter for patients with cumulative distribution less than or equal to 0.2***  **WHERE**  **g.cumulative\_distribution <= 0.2**  ***-- Order the results by cumulative distribution in descending order***  **ORDER BY**  **g.cumulative\_distribution DESC**  ***-- Limit the results to the top 5 patients***  **LIMIT**  **5;** |
| --- |

**OUTPUT**

A screenshot of a computer

Description automatically generated

1. Retrieve patient glucose data and calculate the difference in glucose values between consecutive readings for each patient.

**Function Used: LEAD()**

**The LEAD() function is used to access the value of the next row within the same result set, allowing for calculations and comparisons between consecutive rows.**

**QUERY:**

| **WITH GlucoseValueDifference AS (**  **SELECT**  **d.patientid,**  **d.glucose\_value\_mgdl,**  **d.datestamp AS current\_timestamp,**  **LEAD(d.glucose\_value\_mgdl) OVER (PARTITION BY d.patientid ORDER BY d.datestamp) AS next\_glucose\_value,**  **d.datestamp AS next\_timestamp,**  **LEAD(d.datestamp) OVER (PARTITION BY d.patientid ORDER BY d.datestamp) AS next\_reading\_timestamp,**  **d.glucose\_value\_mgdl - LEAD(d.glucose\_value\_mgdl) OVER (PARTITION BY d.patientid ORDER BY d.datestamp) AS glucose\_difference**  **FROM**  **public.dexcom AS d**  **)**  ***-- Select patient glucose data and the difference in glucose values between consecutive readings***  **SELECT**  **patientid,**  **glucose\_value\_mgdl,**  **current\_timestamp,**  **next\_timestamp,**  **next\_glucose\_value,**  **next\_reading\_timestamp,**  **glucose\_difference**  **FROM**  **GlucoseValueDifference**  **ORDER BY**  **patientid,**  **current\_timestamp;** |
| --- |

**OUTPUT**

A screenshot of a computer

Description automatically generated

1. Based on the linear regression analysis between Interbeat Interval (rmssd\_ms) and glucose values (glucose\_value\_mgdl), can we conclude that Interbeat Interval is a statistically significant predictor of glucose levels? How well does the linear regression model explain the variability in glucose values

**Function Used: REGR\_SLOPE(), REGR\_INTERCEPT(),REGR\_R2()**

**REGR\_Slope: A minimal positive slope (8.767e-05) suggests a weak positive relationship between Interbeat Interval (rmssd\_ms) and glucose values (glucose\_value\_mgdl).**

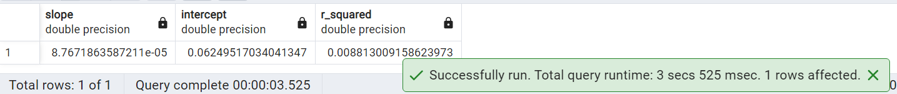
**REGR\_Intercept: The intercept (0.0625) lacks practical interpretation.**

**REGR\_R2 R-squared: A low R-squared (0.0088) indicates the model's limited explanatory power for glucose variability based on Interbeat Interval.**

**QUERY:**

| ***-- Calculate the slope (coefficient), intercept, and R-squared value for the linear regression analysis***  **SELECT**  **REGR\_SLOPE(rmssd\_ms, glucose\_value\_mgdl) AS slope, *-- Calculate the slope of the regression lin***  **REGR\_INTERCEPT(rmssd\_ms, glucose\_value\_mgdl) AS intercept, *-- Calculate the intercept of the regression line***  **REGR\_R2(rmssd\_ms, glucose\_value\_mgdl) AS r\_squared *-- Calculate the coefficient of determination (R-squared)***  **FROM**  **ibi *-- Select data from the 'ibi' table***  **JOIN**  **dexcom ON ibi.patientid = dexcom.patientid *-- Join with the 'dexcom' table based on the 'patientid' column***  **WHERE**  **rmssd\_ms IS NOT NULL *-- Filter out rows where 'rmssd\_ms' is not null***  **AND glucose\_value\_mgdl IS NOT NULL; *-- Filter out rows where 'glucose\_value\_mgdl' is not null*** |
| --- |

**OUTPUT**



1. In the 'foodlog' table, find the top 5 patients who have logged the highest number of food items.

**Function Used: ARRAY\_UPPER()**

**array\_upper function to determine the number of food items for each patient**

**QUERY:**

| ***-- Find the top 5 patients with the highest number of food items logged***  **SELECT**  **patientid,**  **(array\_upper(string\_to\_array(logged\_food, ','), 1)) AS food\_item\_count**  **FROM**  **foodlog**  **ORDER BY**  **food\_item\_count DESC**  **LIMIT 5;** |
| --- |

**OUTPUT**

A screenshot of a computer

Description automatically generated

1. Find out the lowercase first names of all patients along with the total number of food items logged by each patient.

**Function Used: LOWER(), COUNT()**

**LOWER() Function: Converts a string to lowercase characters.**

**COUNT() Function: Counts the number of rows in a result set or the occurrences of a specified expression in a set of rows.**

**QUERY:**

| ***-- Retrieve patient first names in lowercase and total food items logged***  **SELECT**  **d.patientid,**  **LOWER(d.firstname) AS lowercase\_firstname,**  **COUNT(f.patientid) AS total\_food\_items\_logged**  **FROM**  **demographics d**  **LEFT JOIN**  **foodlog f ON d.patientid = f.patientid**  **GROUP BY**  **d.patientid, d.firstname**  **ORDER BY**  **d.patientid;** |
| --- |

**OUTPUT**

A screenshot of a computer

Description automatically generated

1. Can you retrieve the patient IDs, first names, and last names for patients whose mean heart rate is greater than 80, and if the first name or last name is missing, replace it with 'Unknown.

**Function Used: COALSECE()**

**COALESCE: It returns the first non-null value in a list, effectively replacing null values with a specified default.**

**QUERY:**

| **SELECT**  **demographics.patientid,**  **COALESCE(demographics.firstname, 'Unknown') AS patient\_firstname,**  **COALESCE(demographics.lastname, 'Unknown') AS patient\_lastname**  **FROM**  **demographics**  **WHERE**  **demographics.patientid IN (**  **SELECT DISTINCT patientid**  **FROM hr**  **WHERE mean\_hr > 80**  **);** |
| --- |

**OUTPUT**

A screenshot of a computer

Description automatically generated

1. Retrieve the patient IDs, first names, and last names for patients whose root mean square of successive differences is less than 50, and replace the last name 'Doe' with NULL if it exists.

**Function Used: NULLIF()**

**NULLIF: It returns null if two expressions are equal, allowing you to suppress specific values.**

**QUERY:**

| ***-- Select patient information from the "demographics" table.***  **SELECT**  **demographics.patientid, *-- Patient ID***  **demographics.firstname, *-- First Name***  ***-- Replace the last name with null for patients whose last name is 'Doe'.***  **NULLIF(demographics.lastname, 'Doe') AS patient\_lastname**  **FROM**  **demographics**  **WHERE**  ***-- Filter patients based on their IDs.***  **demographics.patientid IN (**  ***-- Subquery to retrieve distinct patient IDs from the "ibi" table***  ***-- where the value in the "rmssd\_ms" column is less than 50.***  **SELECT DISTINCT patientid**  **FROM ibi**  **WHERE rmssd\_ms < 50**  **);** |
| --- |

**OUTPUT**

A screenshot of a computer

Description automatically generated

1. For each patient, can you provide a concatenated list of logged foods along with their total calorie count?

**Function Used: STRING\_AGG()**

**STRING\_AGG function to concatenate the logged\_food values along with their total calorie counts in a formatted manner**

**QUERY:**

| **WITH FoodCalories AS (**  **SELECT**  **patientid,**  **logged\_food,**  **SUM(calorie) AS total\_calories**  **FROM**  **foodlog**  **GROUP BY**  **patientid, logged\_food**  **)**  **SELECT**  **fc.patientid,**  **STRING\_AGG(fc.logged\_food || ' (' || fc.total\_calories || ' calories)', ', ') AS food\_with\_calories**  **FROM**  **FoodCalories fc**  **GROUP BY**  **fc.patientid;** |
| --- |

**OUTPUT**

A screenshot of a menu

Description automatically generated

1. Identify patients who have logged their meals more than five times and categorizes their most common dietary preference.

**Function Used: REGEXP\_SPLIT\_TO\_ARRAY()**

**It uses common table expressions (CTEs) to split, categorize, and count food items logged by patients. The main function used here is REGEXP\_SPLIT\_TO\_ARRAY, which splits logged food items into arrays based on regular expressions.**

**QUERY:**

| ***-- Step 1: Split the logged food text into an array of food items.***  **WITH FoodArrays AS (**  **SELECT**  **f.patientid,**  **REGEXP\_SPLIT\_TO\_ARRAY(f.logged\_food, E'\\s\*,\\s\*') AS food\_array *-- Splitting the logged\_food into an array***  **FROM**  **foodlog f**  **),**  ***-- Step 2: Explode the food arrays into individual food items using REGEXP\_SPLIT\_TO\_TABLE.***  **ExplodedFoods AS (**  **SELECT**  **patientid,**  **UNNEST(food\_array) AS food\_item *-- Unnesting the array to individual food items***  **FROM**  **FoodArrays**  **),**  ***-- Step 3: Categorize food items by food groups (e.g., 'Fruit', 'Vegetable', 'Protein', 'Carb').***  **CategorizedFoods AS (**  **SELECT**  **ef.patientid,**  **ef.food\_item,**  **CASE**  **WHEN food\_item ~\* E'\\b(apple|banana|orange|grape|strawberry|...\\b)' THEN 'Fruit' *-- Categorizing food as 'Fruit'***  **WHEN food\_item ~\* E'\\b(broccoli|spinach|carrot|kale|lettuce|...\\b)' THEN 'Vegetable' *-- Categorizing food as 'Vegetable'***  **WHEN food\_item ~\* E'\\b(chicken|beef|salmon|tofu|...\\b)' THEN 'Protein' *-- Categorizing food as 'Protein'***  **WHEN food\_item ~\* E'\\b(rice|pasta|bread|potato|...\\b)' THEN 'Carb' *-- Categorizing food as 'Carb'***  **ELSE 'Other' *-- Categorizing food as 'Other' if it doesn't match any category***  **END AS food\_group**  **FROM**  **ExplodedFoods ef**  **),**  ***-- Step 4: Calculate the frequency of each food group for each patient.***  **FoodGroupFrequencies AS (**  **SELECT**  **patientid,**  **food\_group,**  **COUNT(\*) AS frequency *-- Counting the frequency of each food group***  **FROM**  **CategorizedFoods**  **GROUP BY**  **patientid, food\_group**  **)**  ***-- Step 5: Find the most common dietary preference for each patient.***  **SELECT**  **f.patientid,**  **d.firstname,**  **d.lastname,**  **(SELECT food\_group FROM FoodGroupFrequencies WHERE patientid = f.patientid ORDER BY frequency DESC LIMIT 1) AS most\_common\_diet *-- Finding the most common dietary preference***  **FROM**  **foodlog f**  **JOIN**  **demographics d ON f.patientid = d.patientid *-- Joining with demographics table***  **GROUP BY**  **f.patientid, d.firstname, d.lastname**  **HAVING**  **COUNT(\*) > 5; *-- Selecting patients who have logged their meals more than five times*** |
| --- |

**OUTPUT**

A screenshot of a computer

Description automatically generated

1. For each patient in the 'demographics' table, can you provide a breakdown of their logged meals by food item, including the number of times each food item has been logged?

**Function Used: REGEXP\_SPLIT\_TO\_TABLE()**

**The REGEXP\_SPLIT\_TO\_TABLE() and unnest functions are used to split the 'logged\_food' column into individual food items and convert them into rows, respectively, allowing for meal count analysis.**

**QUERY:**

| ***-- Use REGEXP\_SPLIT\_TO\_TABLE to extract food items from the 'logged\_food' column***  **WITH FoodItems AS (**  **SELECT**  **f.patientid,**  **TRIM(food\_item) AS food\_item**  **FROM**  **foodlog f**  **CROSS JOIN LATERAL REGEXP\_SPLIT\_TO\_TABLE(f.logged\_food, E'\\s\*,\\s\*') AS food\_item**  **)**  ***-- Count the number of meals logged for each food item and patient***  **SELECT**  **d.patientid,**  **d.firstname,**  **d.lastname,**  **fi.food\_item,**  **COUNT(\*) AS meal\_count**  **FROM**  **demographics d**  **JOIN**  **FoodItems fi ON d.patientid = fi.patientid**  **GROUP BY**  **d.patientid, d.firstname, d.lastname, fi.food\_item**  **ORDER BY**  **d.patientid, meal\_count DESC;** |
| --- |

**OUTPUT**

A screenshot of a computer

Description automatically generated

1. For each patient in the 'demographics' table, can you provide a summary of their dietary information, including the total dietary fiber, sugar, total fat, protein, calorie, and total carbohydrate intake?

**Function Used: ARRAY\_DIMS,ARRAY\_AGG**

**ARRAY\_AGG function to aggregate dietary information (dietary fiber, sugar, total fat, protein, calorie, and total carbohydrate) for each patient into arrays.**

**ARRAY\_DIMS function to each of these arrays to get the dimensions of the array, effectively providing a summary of dietary intake for each patient.**

**QUERY:**

| **SELECT**  **d.patientid,**  **d.firstname,**  **d.lastname,**  **ARRAY\_DIMS(array\_agg(f.dietary\_fiber)) AS dietary\_fiber\_summary,**  **ARRAY\_DIMS(array\_agg(f.sugar)) AS sugar\_summary,**  **ARRAY\_DIMS(array\_agg(f.total\_fat)) AS total\_fat\_summary,**  **ARRAY\_DIMS(array\_agg(f.protein)) AS protein\_summary,**  **ARRAY\_DIMS(array\_agg(f.calorie)) AS calorie\_summary,**  **ARRAY\_DIMS(array\_agg(f.total\_carb)) AS total\_carb\_summary**  **FROM**  **demographics d**  **LEFT JOIN**  **foodlog f ON d.patientid = f.patientid**  **GROUP BY**  **d.patientid, d.firstname, d.lastname**  **ORDER BY**  **d.patientid;** |
| --- |

**OUTPUT**

A screenshot of a computer

Description automatically generated

1. Can you provide a summary of dietary information for each patient in JSON format, including their dietary fiber, sugar, total fat, protein, calorie, and total carbohydrate intake?

**Function Used: ARRAY\_TO\_JSON()**

**ARRAY\_TO\_JSON function to convert an array containing dietary information (dietary fiber, sugar, total fat, protein, calorie, and total carbohydrate) into JSON format.**

**The ARRAY function is used to create an array containing the dietary information values**

**QUERY:**

| ***-- Select the patient ID, first name, last name, and dietary information in JSON format.***  **SELECT**  **f.patientid, *-- Select the patient ID***  **d.firstname, *-- Select the patient's first name***  **d.lastname, *-- Select the patient's last name***  **ARRAY\_TO\_JSON(ARRAY[f.dietary\_fiber, f.sugar, f.total\_fat, f.protein, f.calorie, f.total\_carb]) AS dietary\_info\_json**  ***-- Use ARRAY\_TO\_JSON to convert an array containing dietary information to JSON format.***  **FROM**  **foodlog f**  ***-- Join the 'foodlog' table (aliased as 'f') to access dietary information.***  **JOIN**  **demographics d ON f.patientid = d.patientid**  ***-- Join the 'demographics' table (aliased as 'd') based on patient ID.***  **ORDER BY**  **f.patientid;**  ***-- Order the results by patient ID for readability.*** |
| --- |

**OUTPUT**

A screenshot of a computer

Description automatically generated

1. Which patients have the highest and lowest recorded glucose values (blood sugar levels)?

**Function Used: GREATEST(), LEAST()**

**QUERY:**

| **SELECT**  **patientid,**  **GREATEST(MAX(glucose\_value\_mgdl), MAX(glucose\_value\_mgdl)) AS highest\_glucose,**  **LEAST(MIN(glucose\_value\_mgdl), MIN(glucose\_value\_mgdl)) AS lowest\_glucose**  **FROM**  **dexcom**  **GROUP BY**  **patientid;** |
| --- |

**OUTPUT**

A screenshot of a computer

Description automatically generated

1. Find the average sample covariance between 'rmssd\_ms' and 'mean\_hr' for patients in the 'ibi' and 'hr' tables

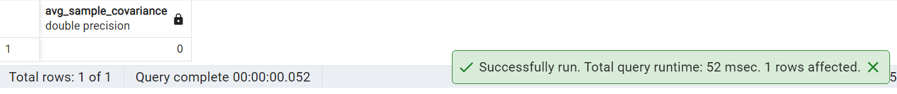
**Function Used: COVAR\_SAMP()**

**The query calculates the average of the sample covariances between 'rmssd\_ms' and 'mean\_hr' for individual patients in the 'ibi' and 'hr' tables using the COVAR\_SAMP() function and then computes the overall average of these sample covariances.**

**QUERY:**

| ***-- Calculate the average of sample covariances between 'rmssd\_ms' and 'mean\_hr' for individual patients.***  **SELECT AVG(sample\_covariance) AS avg\_sample\_covariance**  **FROM (**  ***-- Subquery: Calculate sample covariances for each patient and their heart rate data.***  **SELECT i.patientid,**  **COVAR\_SAMP(i.rmssd\_ms, h.mean\_hr) AS sample\_covariance**  **FROM ibi i**  **JOIN hr h ON i.patientid = h.patientid**  **GROUP BY i.patientid**  **) AS subquery;** |
| --- |

**OUTPUT**



If the sample covariance between two variables is calculated to be zero, it indicates that there is little to no linear relationship or correlation between those variables in the sample data.

1. Calculate both the sample standard deviation and variance of 'glucose\_value\_mgdl' for male patients (gender = 'Male') in the 'dexcom' table.

**Function Used: STDDEV\_SAMP(), VARIANCE()**

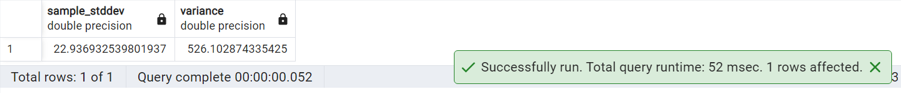
**STDDEV\_SAMP(glucose\_value\_mgdl) AS sample\_stddev: This computes the sample standard deviation of the 'glucose\_value\_mgdl' values for male patients. The sample standard deviation measures the amount of variation or dispersion in the glucose values. A higher standard deviation indicates greater variability in glucose levels.**

**VARIANCE(glucose\_value\_mgdl) AS variance: This calculates the variance of the 'glucose\_value\_mgdl' data for male patients. Variance quantifies how data points differ from the mean. A higher variance means that glucose values are more spread out from the mean.**

**QUERY:**

| **SELECT STDDEV\_SAMP(glucose\_value\_mgdl) AS sample\_stddev,**  **VARIANCE(glucose\_value\_mgdl) AS variance**  **FROM dexcom**  **WHERE patientid IN (**  **SELECT patientid**  **FROM demographics**  **WHERE gender = 'Male'**  **);** |
| --- |

**OUTPUT**



**OUTPUT ANALYSIS :**

Sample Standard Deviation (sample\_stddev): This value of approximately 22.937 indicates that there is a moderate amount of variability or fluctuation in glucose levels among male patients. It represents the average amount by which individual glucose values differ from the sample mean.

Variance: The variance value of approximately 526.103 indicates that glucose values for male patients have a relatively wide spread around the mean. In other words, there is a significant amount of variation in glucose levels among male patients.

1. Retrieve the patient IDs and the total number of unique characters in their last names. Order the results by the number of unique characters in descending order.

**Function Used: CHAR\_LENGTH(),REGEXP\_REPLACE()**

**CHAR\_LENGTH() function - to find the length of the last name.**

**REGEXP\_REPLACE() function - used to remove consecutive duplicate characters in the last name using a regular expression enabling the calculation of the count of unique characters.**

**This query is designed is to calculate and order patients based on the count of unique characters in their last names. This can help identify patients with more diverse or less common last names within the dataset.**

**QUERY:**

| ***-- Calculate and order patients by the count of unique characters in their last names.***  **SELECT**  **patientid,**  **CHAR\_LENGTH(lastname) - CHAR\_LENGTH(REGEXP\_REPLACE(lastname, '(.).\*?\\1', '\\1', 'g')) AS unique\_char\_count**  ***-- Calculate the unique character count by subtracting the length after removing duplicate characters.***  **FROM demographics *-- From the demographics table***  **ORDER BY unique\_char\_count DESC; *-- Order the results in descending order of unique character count.*** |
| --- |

**OUTPUT:**

A screenshot of a computer

Description automatically generated

1. Retrieve a list of patients along with their patient IDs and the DENSE RANK of their average glucose levels from the Dexcom table. Order the results by patient ID and rank in ascending order.

**Function Used: DENSE\_RANK(),AVG()**

**This query is to provide a ranking of patients based on their average glucose values, which can be valuable for medical analysis and identifying patients with different glucose level profiles. DENSE\_RANK() is used here to assign a rank to each patient while handling ties (patients with the same average glucose) by assigning the same rank to tied values. This helps in producing a meaningful ranking when multiple patients have identical average glucose levels.**

**QUERY:**

| ***-- Rank patients by their average glucose values using DENSE\_RANK.***    ***-- Create a Common Table Expression (CTE) to calculate the average glucose values and rank patients.***  **WITH RankedPatients AS (**  **SELECT**  **d.patientid,**  **AVG(dex.glucose\_value\_mgdl) AS avg\_glucose,**  **DENSE\_RANK() OVER (ORDER BY AVG(dex.glucose\_value\_mgdl)) AS dense\_rank *-- Rank patients based on average glucose***  **FROM demographics d**  **LEFT JOIN dexcom dex ON d.patientid = dex.patientid**  **GROUP BY d.patientid**  **)**    ***-- Select patient ID, average glucose, and dense rank from the CTE.***  **SELECT**  **Patientid,**  **avg\_glucose,**  **dense\_rank *-- Dense rank based on average glucose***  **FROM RankedPatients**  **ORDER BY patientid, dense\_rank; *-- Order the results by patient ID and dense rank.*** |
| --- |

**OUTPUT:**

A screenshot of a computer

Description automatically generated

1. Calculate the percent rank of each patient's maximum EDA value within their gender group

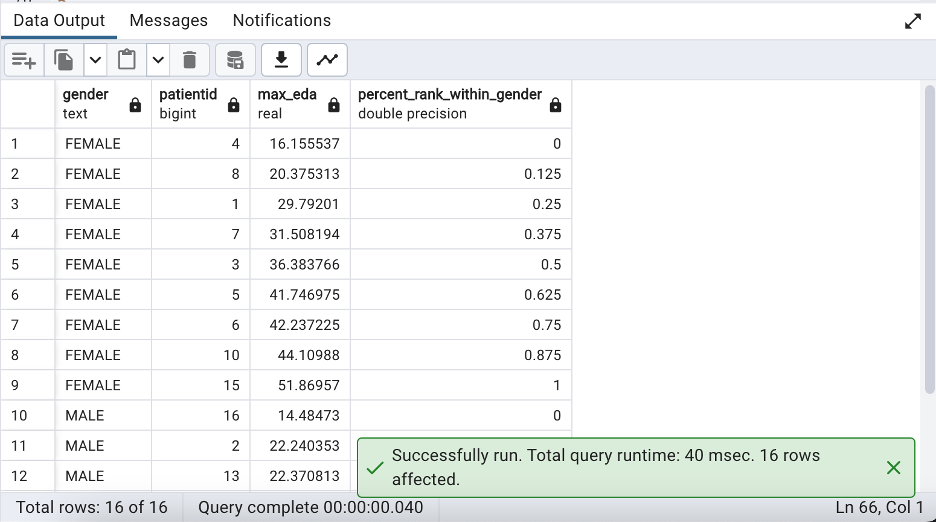
**Function Used: PERCENT\_RANK**

**PERCENT\_RANK() window function to calculate the percent rank of each patient's max EDA within their gender group. The PARTITION BY clause ensures that the percent rank is calculated separately for each gender.**

**QUERY:**

| ***-- Create a Common Table Expression (CTE) named GenderMaxEDA***  ***-- This CTE calculates the maximum electrodermal activity (EDA) for each patient by gender***  **WITH GenderMaxEDA AS (**  **SELECT**  **d.gender,**  **e.patientid,**  **MAX(e.max\_eda) AS max\_eda *-- Calculate the maximum EDA for each patient***  **FROM**  **demographics d**  **JOIN**  **eda e**  **ON**  **d.patientid = e.patientid *-- Join the tables on patient ID***  **GROUP BY**  **d.gender, e.patientid *-- Group the results by gender and patient ID***  **)**    ***-- Select and analyze the data from the GenderMaxEDA CTE***  **SELECT**  **gender,**  **patientid,**  **max\_eda,**  **PERCENT\_RANK() OVER (PARTITION BY gender ORDER BY max\_eda) AS percent\_rank\_within\_gender**  ***-- Calculate the percent rank of each patient's max EDA within their gender group***  **FROM**  **GenderMaxEDA**  **ORDER BY**  **gender, percent\_rank\_within\_gender; *-- Order the results by gender and percent rank*** |
| --- |

**OUTPUT:**



1. Group patients by gender and aggregate maximum EDA values into a JSON array.

**Function Used: JSON\_AGG()**

**JSON\_AGG() - The primary function used in this query is json\_agg(), which aggregates values into a JSON array.This query allows for the summarization and representation of maximum EDA values in a structured JSON format for each gender, making it easier to analyze EDA data in a gender-specific context.**

**QUERY:**

| ***-- Group patients by gender and aggregate maximum EDA values into a JSON array.***    ***-- Select gender and JSON array of maximum EDA values.***  **SELECT**  **gender, *-- Gender***  **json\_agg(max\_eda) AS max\_eda\_values *-- Aggregate maximum EDA values into a JSON array***  **FROM (**  ***-- Subquery to calculate maximum EDA values for each patient within each gender.***  **SELECT**  **d.gender, *-- Gender***  **e.patientid,**  **ROUND(MAX(e.max\_eda)::numeric,2) AS max\_eda *-- Maximum EDA value***  **FROM demographics d**  **JOIN eda e ON d.patientid = e.patientid**  **GROUP BY d.gender, e.patientid**  **) AS gender\_max\_eda *-- Alias for the subquery***  **GROUP BY gender; *-- Group the results by gender*** |
| --- |

**OUTPUT:**

A screenshot of a computer

Description automatically generated

1. Display the count patients in  diabetic,prediabetic and normal category based on HbA1c value using a Bar Chart.We'll categorize patients based on their HbA1c values into diabetic (HbA1c >= 6.5), prediabetic (HbA1c between 5.7 and 6.4), and normal (HbA1c < 5.7).

**Function Used: COUNT(\*)**

**QUERY**

| **SELECT**  **CASE**  **WHEN hba1c >= 6.5 THEN 'Diabetic'**  **WHEN hba1c >= 5.7 AND hba1c < 6.5 THEN 'Prediabetic'**  **ELSE 'Normal'**  **END AS hba1c\_category,**  **COUNT(\*) AS patient\_count**  **FROM**  **demographics**  **GROUP BY**  **hba1c\_category; *-- No Diabetic Patients*** |
| --- |

**OUTPUT**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

1. Display each patients HRV and Mean glucose value  as  3 different groups.  (uses the NTILE() function to distribute rows into 3 buckets)

**Function Used: NTILE()**

**The PostgreSQL NTILE() function allows you to divide ordered rows in the partition into a specified number of ranked groups as equal size as possible. These ranked groups are called buckets**

**INPUT**

| **SELECT**  **patientid,**  **hrv,**  **glucose\_value,**  **NTILE(3) OVER (ORDER BY hrv) AS hrv\_group,**  **NTILE(3) OVER (ORDER BY glucose\_value) AS glucose\_group**  **FROM**  **hrv\_bloodsugar**  **ORDER BY patientid;** |
| --- |

**OUTPUT**

A screenshot of a computer

Description automatically generated

1. Using ROW\_NUMBER() function to get the patient demographucs deatils with 3rd highest Hba1c value.

**Function Used: ROW\_NUMBER()**

**INPUT**

| ***-- The following SQL query is used to rank rows based on the 'hba1c' column in the 'demographics' table***  ***-- and retrieve the rows with a rank of 3.***  ***-- Step 1: With the Common Table Expression (CTE) 'ranked\_hba1c', we are creating a temporary result set.***  **WITH ranked\_hba1c AS (**  ***-- Inside the CTE, we select all columns ('\*') from the 'demographics' table and add a new column 'rank'.***  **SELECT**  **\*,**  **ROW\_NUMBER() OVER (ORDER BY hba1c DESC) AS rank**  **FROM**  **demographics d**  **)**  ***-- Step 2: After defining the CTE, we use a SELECT statement to query the CTE.***  ***-- We want to retrieve all columns ('\*') from the 'ranked\_hba1c' CTE where the 'rank' column equals 3.***  **SELECT \* FROM ranked\_hba1c**  **WHERE rank = 3;** |
| --- |

**OUTPUT**

**A screenshot of a computer

Description automatically generated**

1. Using cumulative distribution function calculate the Blood Sugar for each patient partitioned by gender.

**Function Used: CUME\_DIST()**

**QUERY**

| ***-- The following SQL query calculates the average blood sugar level for each patient***  ***-- and also calculates the cumulative distribution of those averages based on gender.***  ***-- Step 1: In the SELECT clause, we specify the columns we want to include in the result:***  **SELECT**  ***-- We include the 'patientid' column from the 'demographics' table.***  **d.patientid,**    ***-- We calculate the average of 'glucose\_value\_mgdl' for each patient and round it to 2 decimal places.***  **ROUND(AVG(dc.glucose\_value\_mgdl)::numeric, 2)::double precision AS Blood\_Sugar,**    ***-- We include the 'gender' column from the 'demographics' table.***  **d.gender,**    ***-- We calculate the cumulative distribution of average blood sugar values for each gender.***  **ROUND(cume\_dist() OVER (PARTITION BY gender ORDER BY AVG(dc.glucose\_value\_mgdl))::numeric, 2)::double precision AS cumulative\_distribution**    ***-- Step 2: In the FROM clause, we specify the tables and join conditions.***  **FROM**  ***-- We join the 'demographics' table 'd' with the 'dexcom' table 'dc' using the 'patientid' column.***  **demographics d JOIN**  **dexcom dc ON d.patientid = dc.patientid**  ***-- Step 3: We group the results by 'patientid' to calculate the average blood sugar for each patient.***  **GROUP BY d.patientid;** |
| --- |

**OUTPUT**

**A screenshot of a computer

Description automatically generated**

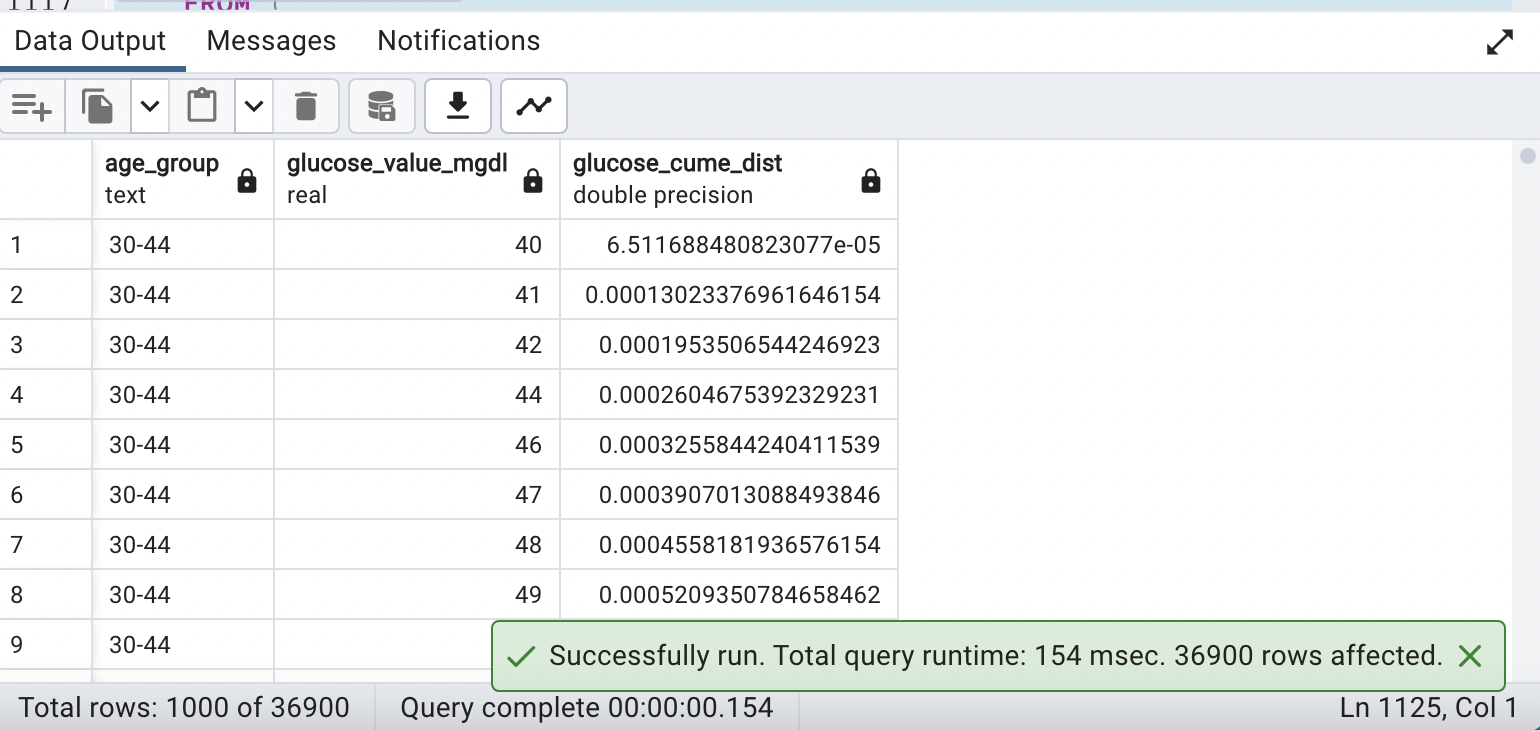
1. Categorize patients into age groups and calculate cumulative distribution of glucose values.

**Function Used: CUME\_DIST()**

**QUERY**

| ***-- Select the age group, glucose value, and cumulative distribution.***  **SELECT**  **age\_group,**  **glucose\_value\_mgdl,**  **cume\_dist() OVER (PARTITION BY age\_group ORDER BY glucose\_value\_mgdl) AS glucose\_cume\_dist *-- Calculate cumulative distribution***  **FROM (**  ***-- Subquery to calculate dynamically calculated age group and glucose values from the dexcom and demographics tables.***  **SELECT**  **CASE**  **WHEN age >= 18 AND age < 30 THEN '18-29' *-- Age group 18-29***  **WHEN age >= 30 AND age < 45 THEN '30-44' *-- Age group 30-44***  **ELSE '45+' *-- Age group 45+***  **END AS age\_group, *-- Calculate age group based on age***  **glucose\_value\_mgdl**  **FROM (**  **SELECT**  **EXTRACT(YEAR FROM AGE(dob)) AS age, *-- Calculate age from date of birth (dob)***  **glucose\_value\_mgdl**  **FROM dexcom**  **JOIN demographics ON dexcom.patientid = demographics.patientid**  **) AS age\_glucose**  **) AS age\_group\_glucose; *-- Alias for the subquery*** |
| --- |

**OUTPUT**



1. Calculate the Median RMSSD Value from IBI Table per Patient:

**Function Used: PERCENTILE\_CONT()**

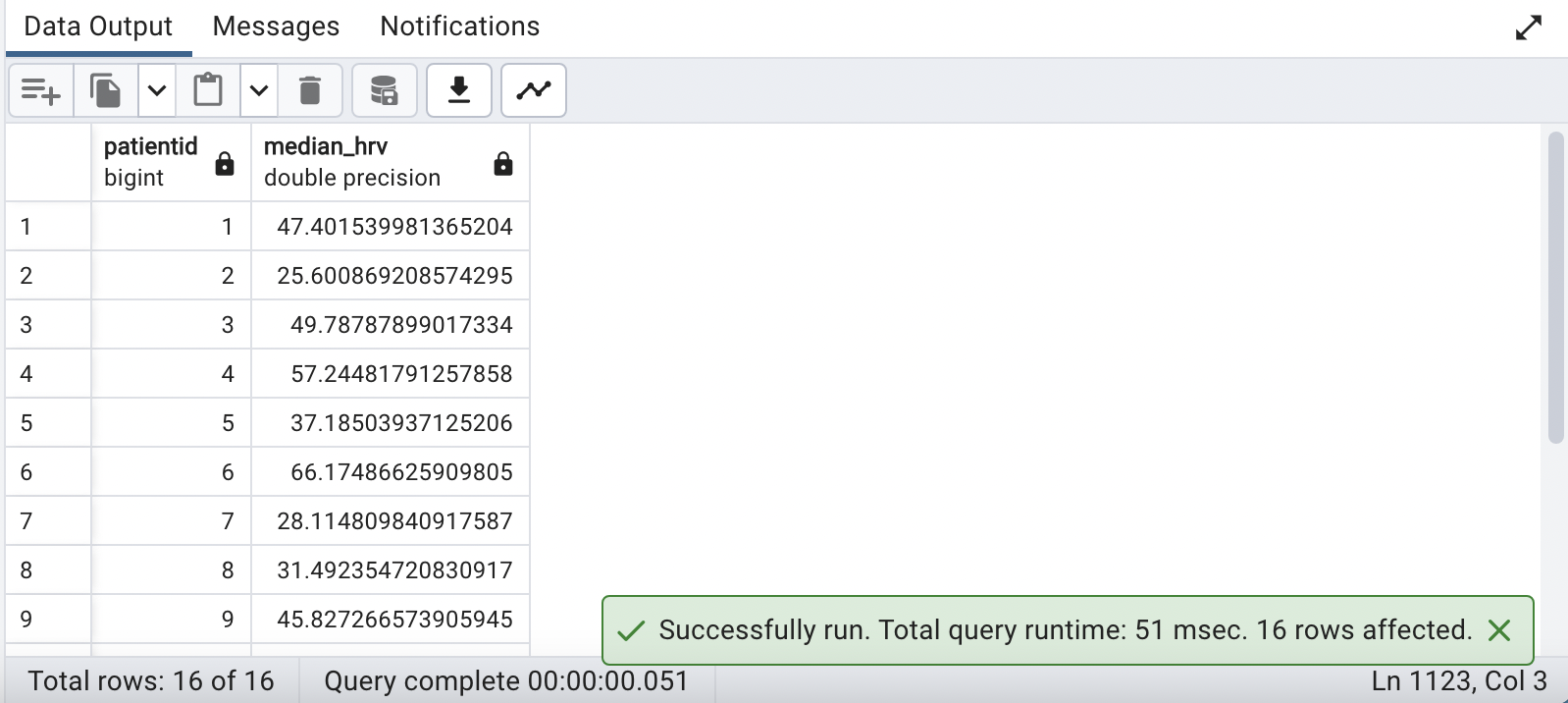
**PERCENTILE\_CONT : It is an inverse distribution function that assumes a continuous distribution model. It takes a percentile value and a sort specification, and returns an interpolated value that would fall into that percentile value with respect to the sort specification. Nulls are ignored in the calculation**

**The purpose of this query is to provide insights into the median HRV for each patient based on their "ibi" (Inter-Beat Interval) data. A higher HRV is generally associated with better cardiovascular health and lower stress levels. Calculating the median HRV for each patient helps in understanding their heart rate variability and overall health status.**

**QUERY**

| ***-- This query calculates the median HRV (Heart Rate Variability) for each unique patient.***  ***-- HRV is an important metric for assessing heart health and stress levels.***  **SELECT**  **patientid, *-- Select the patient ID for identification in the result.***  **percentile\_cont(0.5) WITHIN GROUP (ORDER BY rmssd\_ms \* 600) AS median\_hrv**  ***-- Calculate the 50th percentile (median) of HRV within each patient's data.***  ***-- The ORDER BY clause sorts the data by HRV values multiplied by 600 for ordering.***  **FROM**  **ibi**  ***-- This query operates on the "ibi" table, which contains data related to heart rate variability.***  **GROUP BY**  **patientid; *-- Group the results by patient ID to calculate median HRV for each patient.*** |
| --- |

**OUTPUT**

****

1. Identify the patients with overlapping date ranges of EDA measurements

**Function Used: RANGE\_INTERSECT\_AGG()**

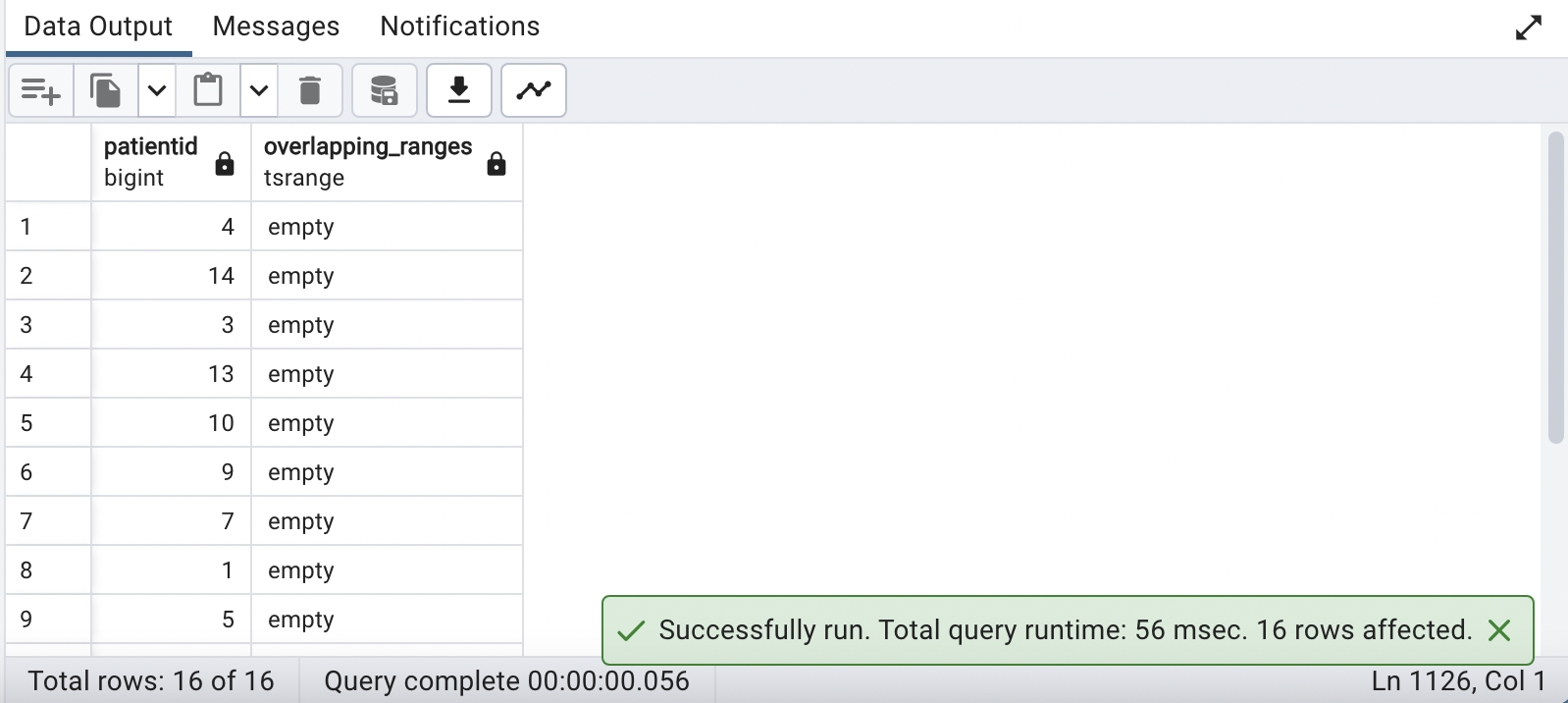
**RANGE\_INTERSECT\_AGG() - function is an aggregate function that returns the intersection of all non-NULL input range values ​​in a group.**

**This query helps identify patients who have multiple EDA measurements recorded on overlapping dates, which may be useful for further analysis or investigation.**

**QUERY**

| ***-- Find patients with overlapping date ranges of EDA measurements.***  ***WITH OverlappingEda AS (***  ***SELECT***  ***patientid,***  ***range\_intersect\_agg(date\_range) AS overlapping\_ranges***  ***FROM (***  ***SELECT***  ***patientid,***  ***tsrange(datestamp, datestamp + interval '1 day', '[]') AS date\_range***  ***FROM***  ***eda***  ***) AS date\_ranges***  ***GROUP BY***  ***patientid***  ***HAVING***  ***count(\*) > 1***  ***)***  ***SELECT***  ***o.patientid,***  ***o.overlapping\_ranges***  ***FROM***  ***OverlappingEda o;*** |
| --- |

**OUTPUT**

****