CREATE OR REPLACE FUNCTION db\_23\_0853\_ancestry\_2023.sp\_check\_n\_consecutive\_consonant(

in\_batch\_name character varying,

in\_field\_name character varying,

in\_n int)

RETURNS refcursor

LANGUAGE plpgsql

SET search\_path TO 'db\_23\_0853\_ancestry\_2023'

AS $function$

DECLARE

var\_condition varchar;

var\_sql varchar;

cursor\_result refcursor := 'cursor\_result';

BEGIN

/\*

in\_batch\_name: tên batch, empty mean all

in\_field\_name: tên field

in\_n: số lượng phụ âm cần lọc

\*/

var\_regex varchar := '[b-df-hj-np-tv-z]{' || in\_n || '}';

var\_sql := '

SELECT data\_value as ' || quote\_ident(in\_field\_name) || ', amount

FROM sp\_split\_single\_word\_by\_field($1,$2)

WHERE data\_value ~\* ''' || var\_regex || '''

ORDER BY 2 DESC, 1

';

OPEN cursor\_result FOR

EXECUTE

(var\_sql) USING in\_batch\_name,

in\_field\_name;

RETURN cursor\_result;

END;

$function$

Giải thích function (1)

Đây là một đoạn code PL/pgSQL để tạo một hàm có tên db\_23\_0853\_ancestry\_2023.sp\_check\_n\_consecutive\_consonant. Hàm này có chức năng lọc các từ trong một trường dữ liệu cụ thể, trong đó xuất hiện liên tiếp có n phụ âm và trả về dữ liệu qua một con trỏ tham chiếu.

**Giải thích từng dòng lệnh trong hàm như sau:**

1. CREATE OR REPLACE FUNCTION db\_23\_0853\_ancestry\_2023.sp\_check\_n\_consecutive\_consonant(in\_batch\_name character varying, in\_field\_name character varying, in\_n int) RETURNS refcursor LANGUAGE plpgsql SET search\_path TO 'db\_23\_0853\_ancestry\_2023' AS $function$ - Khai báo hàm tạo ra một hàm PL/pgSQL. Các tham số đầu vào của hàm bao gồm tên của batch, tên của trường và số lượng phụ âm cần lọc. Hàm trả về một con trỏ tham chiếu và được định nghĩa để sử dụng trong schema db\_23\_0853\_ancestry\_2023.

2. DECLARE var\_condition varchar; var\_sql varchar; cursor\_result refcursor := 'cursor\_result'; - Định nghĩa các biến cục bộ cho hàm, bao gồm một chuỗi điều kiện và một chuỗi SQL để thực thi. Cuối cùng, một con trỏ tham chiếu cursor\_result được khởi tạo.

3. var\_regex varchar := '[b-df-hj-np-tv-z]{' || in\_n || '}'; - Định nghĩa một biểu thức chính quy được sử dụng để lọc các từ trong trường dữ liệu. Biểu thức này chỉ liên quan đến những phụ âm xuất hiện liên tiếp với số lượng phụ âm được chỉ định bởi in\_n.

4. var\_sql := 'SELECT data\_value as ' || quote\_ident(in\_field\_name) || ', amount FROM sp\_split\_single\_word\_by\_field($1,$2) WHERE data\_value ~\* ''' || var\_regex || ''' ORDER BY 2 DESC, 1'; - Định nghĩa một câu lệnh SQL được sử dụng để truy vấn cơ sở dữ liệu. Câu lệnh này sử dụng hàm sp\_split\_single\_word\_by\_field để phân tách các từ trong trường dữ liệu và lọc ra các từ có liên tiếp số lượng phụ âm được chỉ định bởi in\_n. Câu lệnh trả về hai cột: data\_value - giá trị từ đã lọc và amount - số lần xuất hiện của từ trong trường dữ liệu.

5. OPEN cursor\_result FOR EXECUTE(var\_sql) USING in\_batch\_name, in\_field\_name; - Mở con trỏ tham chiếu cursor\_result và thực thi câu lệnh SQL được định nghĩa trước đó. Các tham số in\_batch\_name và in\_field\_name được sử dụng trong câu lệnh SQL thông qua cấu trúc USING.

6. RETURN cursor\_result; - Trả về con trỏ tham chiếu cursor\_result.

Giải thích function (2)

Đây là một function trong PostgreSQL để tìm kiếm các từ có số lượng phụ âm liên tiếp lớn hơn hoặc bằng giá trị được xác định.

**Các tham số đầu vào:**

- in\_batch\_name: Tên batch, nếu để trống có nghĩa là tìm trong tất cả các batch.

- in\_field\_name: Tên của trường chứa dữ liệu cần tìm kiếm.

- in\_n: Số lượng phụ âm liên tiếp cần tìm kiếm.

**Các hàm được sử dụng:**

1. SET search\_path TO 'db\_23\_0853\_ancestry\_2023': Thiết lập schema mặc định cho function.

2. quote\_ident(): Hàm này nhận vào một chuỗi và trả về chuỗi đã được quote để sử dụng như một identifier hợp lệ trong cú pháp SQL. Trong function này, nó được sử dụng để quote tên trường (in\_field\_name).

3. sp\_split\_single\_word\_by\_field(): Đây là một stored procedure (lưu trữ) khác trong cùng schema với function hiện tại. Nó được sử dụng để phân tách các từ trong trường dữ liệu thành các record riêng biệt, mỗi record chứa một từ. Stored procedure này nhận hai tham số đầu vào: tên batch và tên trường, và trả về một bảng với các cột sau:

- batch\_name: Tên batch.

- field\_name: Tên trường.

- data\_value: Giá trị của từng từ đã được phân tách.

- row\_number: Số thứ tự của record tương ứng.

4. OPEN cursor\_result FOR EXECUTE(var\_sql) USING in\_batch\_name, in\_field\_name;: Hàm này dùng để mở một cursor và lấy dữ liệu từ câu lệnh SQL. Câu lệnh SQL được lưu trong biến var\_sql, và giá trị của in\_batch\_name và in\_field\_name được truyền vào sử dụng USING.

**Giải thích:**

Trong function này, trước khi thực hiện câu lệnh SQL để tìm kiếm các từ có số lượng phụ âm liên tiếp lớn hơn hoặc bằng giá trị được xác định (in\_n), ta thiết lập schema mặc định cho function và sử dụng hàm quote\_ident() để quote tên trường (in\_field\_name). Sau đó, ta sử dụng stored procedure sp\_split\_single\_word\_by\_field() để phân tách các từ trong trường dữ liệu thành các record riêng biệt. Kết quả trả về từ stored procedure này được sử dụng để thực hiện câu lệnh SQL để tìm kiếm các từ có số lượng phụ âm liên tiếp lớn hơn hoặc bằng giá trị được xác định (in\_n). Cuối cùng, ta sử dụng hàm OPEN cursor\_result FOR EXECUTE(var\_sql) USING in\_batch\_name, in\_field\_name; để mở một cursor và lấy dữ liệu từ câu lệnh SQL. Kết quả cuối cùng của function là một cursor.

Để linh động thay thế số lượng nguyên âm/phụ âm cần lọc, chúng ta có thể sửa câu lệnh WHERE theo yêu cầu. Thay vì sử dụng một biểu thức chính quy cứng nhắc [aeiou]{2} để tìm kiếm chuỗi có 2 nguyên âm liên tiếp, chúng ta có thể thay thế nó bằng một biến vowel\_count để lọc dựa trên số lượng nguyên âm.

Dưới đây là function đã được điều chỉnh:

CREATE OR REPLACE FUNCTION db\_23\_0853\_ancestry\_2023.sp\_check\_consecutive\_vowel(

batch\_name character varying,

fieldname character varying,

vowel\_count integer

) RETURNS refcursor LANGUAGE plpgsql

SET search\_path TO 'db\_23\_0853\_ancestry\_2023' AS $function$

DECLARE

var\_sql varchar;

cursor\_result refcursor := 'cursor\_result';

BEGIN

/\*

$1: batch name, empty mean all

$2: fieldname

$3: vowel count

\*/

var\_sql := '

SELECT data\_value as ' || quote\_ident(fieldname) || ', amount

FROM sp\_split\_single\_word\_by\_field($1,$2)

WHERE data\_value ~\* ''([aeiou]{' || vowel\_count || ',})''

ORDER BY 2 DESC, 1

';

OPEN cursor\_result FOR EXECUTE var\_sql USING batch\_name, fieldname;

RETURN cursor\_result;

END;

$function$

Chúng ta đã thêm một tham số vowel\_count mới vào function để quy định số lượng nguyên âm cần tìm kiếm. Trong phần WHERE của câu lệnh SQL, chúng ta đã sử dụng biến vowel\_count để xác định số lượng nguyên âm cần tìm kiếm và sử dụng nó trong biểu thức chính quy ([aeiou]{ + vowel\_count + ,}) để tìm kiếm chuỗi có số lượng nguyên âm liên tiếp bằng với giá trị của biến vowel\_count.

Ví dụ: Nếu bạn muốn tìm kiếm các chuỗi có 3 nguyên âm liên tiếp, bạn chỉ cần gọi function như sau:

SELECT \* FROM db\_23\_0853\_ancestry\_2023.sp\_check\_consecutive\_vowel('my\_batch', 'my\_field', 3);

Nếu bạn muốn tìm kiếm các chuỗi có 4 phụ âm liên tiếp, bạn chỉ cần thay đổi vowel\_count thành 0:

SELECT \* FROM db\_23\_0853\_ancestry\_2023.sp\_check\_consecutive\_vowel('my\_batch', 'my\_field', 0);

Function db\_23\_0853\_ancestry\_2023.sp\_check\_consecutive\_vowel trả về một con trỏ điều khiển (refcursor) chứa kết quả của câu lệnh SELECT được thực hiện trong hàm. Để thực thi function, bạn sử dụng cú pháp sau:

SELECT \* FROM db\_23\_0853\_ancestry\_2023.sp\_check\_consecutive\_vowel(batch\_name, fieldname, vowel\_count);

Trong đó:

- batch\_name: Tên loạt dữ liệu để tìm kiếm.

- fieldname: Tên trường dữ liệu để tìm kiếm.

- vowel\_count: Số lượng nguyên âm cần tìm kiếm liên tiếp. Nếu giá trị của biến này là 0, function sẽ tìm kiếm các chuỗi có số lượng phụ âm liên tiếp.

Sau đó, function sẽ thực hiện các công việc sau:

1. Khai báo biến: var\_sql để lưu trữ câu lệnh SQL, cursor\_result để lưu trữ con trỏ điều khiển và trả về kết quả cuối cùng.

2. Xây dựng câu lệnh SQL: Biến var\_sql sẽ được xây dựng để chứa câu lệnh SELECT để tìm kiếm chuỗi có số lượng nguyên âm hoặc phụ âm liên tiếp tùy thuộc vào giá trị của biến vowel\_count.

3. Mở con trỏ điều khiển: Function sẽ mở con trỏ điều khiển và thực thi câu lệnh SELECT được xây dựng trong bước 2.

4. Trả về kết quả: Cuối cùng, function sẽ trả về con trỏ điều khiển chứa kết quả của câu lệnh SELECT.

Tóm lại, function db\_23\_0853\_ancestry\_2023.sp\_check\_consecutive\_vowel có chức năng tìm kiếm các chuỗi có số lượng nguyên âm hoặc phụ âm liên tiếp tùy thuộc vào giá trị của biến vowel\_count, trong một trường dữ liệu cụ thể và trả về kết quả thông qua một con trỏ điều khiển.

Để linh hoạt thay đổi số lượng nguyên âm/phụ âm cần lọc, ta có thể thêm hai tham số mới vào hàm và sử dụng chúng để tạo biểu thức chính quy trong câu lệnh SQL. Sau đây là phiên bản đã được điều chỉnh của hàm này:

CREATE OR REPLACE FUNCTION db\_23\_0853\_ancestry\_2023.sp\_check\_2\_consecutive\_vowel(

batch\_name character varying,

field\_name character varying,

num\_vowels integer,

num\_consonants integer

) RETURNS refcursor LANGUAGE plpgsql

AS $function$

DECLARE

var\_sql varchar;

cursor\_result refcursor := 'cursor\_result';

BEGIN

/\*

\* $1: batch name, empty mean all

\* $2: fieldname

\* $3: number of consecutive vowels

\* $4: number of consecutive consonants

\*/

var\_sql := format($$

SELECT data\_value AS %1$I, amount

FROM sp\_split\_single\_word\_by\_field('%2$s', '%1$I')

WHERE data\_value ~\*

'[aeiou]{%3$d}' || -- number of consecutive vowels

'[bcdfghjklmnpqrstvwxyz]{%4$d}' || -- number of consecutive consonants

'[^aeiou]\*$' -- optional trailing characters that are not vowels

ORDER BY 2 DESC, 1

$$, field\_name, batch\_name, num\_vowels, num\_consonants);

OPEN cursor\_result FOR EXECUTE var\_sql;

RETURN cursor\_result;

END;

$function$;

Trong phiên bản mới này, ta đã thêm hai tham số num\_vowels và num\_consonants để lưu trữ số lượng nguyên âm/phụ âm cần lọc. Câu lệnh SQL được tạo ra bằng hàm format() và sử dụng các giá trị của các tham số để tạo biểu thức chính quy. Biểu thức chính quy này là [aeiou]{num\_vowels}[bcdfghjklmnpqrstvwxyz]{num\_consonants}[^aeiou]\*$, có nghĩa là tìm kiếm các từ có num\_vowels nguyên âm liên tiếp, theo sau bởi num\_consonants phụ âm liên tiếp, và có thể có các ký tự không phải nguyên âm ở cuối từ.

Ví dụ: Nếu bạn muốn tìm kiếm các từ có 3 nguyên âm liên tiếp và 2 phụ âm liên tiếp trong trường description của bảng product, bạn có thể sử dụng lệnh sau:

SELECT \* FROM db\_23\_0853\_ancestry\_2023.sp\_check\_2\_consecutive\_vowel('product', 'description', 3, 2);

Giải thích đoạn code:

var\_sql := format($$

SELECT data\_value AS %1$I, amount

FROM sp\_split\_single\_word\_by\_field('%2$s', '%1$I')

WHERE data\_value ~\*

'[aeiou]{%3$d}' || -- number of consecutive vowels

'[bcdfghjklmnpqrstvwxyz]{%4$d}' || -- number of consecutive consonants

'[^aeiou]\*$' -- optional trailing characters that are not vowels

ORDER BY 2 DESC, 1

$$, field\_name, batch\_name, num\_vowels, num\_consonants);

Đoạn code này tạo ra một câu lệnh SQL để truy vấn dữ liệu từ bảng sp\_split\_single\_word\_by\_field. Câu lệnh này sẽ tìm kiếm các từ trong trường data\_value của bảng kết quả trả về từ hàm sp\_split\_single\_word\_by\_field, có số lượng nguyên âm liên tiếp là num\_vowels và số lượng phụ âm liên tiếp là num\_consonants.

Câu lệnh SQL được tạo ra bằng cách sử dụng hàm format() để chèn giá trị của các biến vào trong chuỗi SQL. Một số điểm cần lưu ý:

Ký tự %1$I trong chuỗi SQL sẽ được thay thế bằng giá trị của biến field\_name.

Ký tự %2$s trong chuỗi SQL sẽ được thay thế bằng giá trị của biến batch\_name.

Ký tự %3$d trong biểu thức chính quy [aeiou]{%3$d} sẽ được thay thế bằng giá trị của biến num\_vowels.

Ký tự %4$d trong biểu thức chính quy [bcdfghjklmnpqrstvwxyz]{%4$d} sẽ được thay thế bằng giá trị của biến num\_consonants.

Sau khi các giá trị được chèn vào trong chuỗi SQL, biểu thức chính quy được tạo ra như sau:

[aeiou]{num\_vowels}[bcdfghjklmnpqrstvwxyz]{num\_consonants}[^aeiou]\*$

Biểu thức này sẽ tìm kiếm các từ có num\_vowels nguyên âm liên tiếp, theo sau bởi num\_consonants phụ âm liên tiếp, và có thể có các ký tự không phải nguyên âm ở cuối từ. Kết quả truy vấn sẽ được sắp xếp theo hai tiêu chí: đầu tiên là số lượng từ tìm thấy (amount), sau đó là giá trị của từ đó (data\_value). Cuối cùng, câu lệnh EXECUTE var\_sql được sử dụng để thực thi câu lệnh SQL đã được tạo ra.

Thực thi function vừa điều chỉnh:

Để thực thi hàm sp\_check\_2\_consecutive\_vowel, bạn có thể sử dụng câu lệnh SQL sau:

SELECT \* FROM db\_23\_0853\_ancestry\_2023.sp\_check\_2\_consecutive\_vowel('batch\_name', 'field\_name', num\_vowels, num\_consonants);

Trong đó:

'batch\_name' là tên của bảng cần truy vấn, hoặc rỗng nếu muốn áp dụng trên toàn bộ bảng.

'field\_name' là tên của trường trong bảng cần truy vấn.

num\_vowels là số lượng nguyên âm liên tiếp cần tìm kiếm.

num\_consonants là số lượng phụ âm liên tiếp cần tìm kiếm.

Sau khi thực thi câu lệnh SQL này, kết quả truy vấn sẽ được trả về dưới dạng một con trỏ (refcursor) có tên cursor\_result. Bạn có thể sử dụng các câu lệnh SQL khác để xử lý kết quả này, ví dụ như sử dụng lệnh FETCH để lấy các hàng dữ liệu từ con trỏ.

Nháp:

là một chuyên gia bạn hãy giải thích chức năng chính và chi tiết từng câu lệnh trong function dưới đây:

CREATE

OR

REPLACE

FUNCTION db\_23\_0853\_ancestry\_2023.sp\_check\_2\_consecutive\_vowel(character varying, character varying) RETURNS refcursor LANGUAGE plpgsql

SET

search\_path TO 'db\_23\_0853\_ancestry\_2023' AS $function $DECLARE var\_sql varchar;

cursor\_result refcursor := 'cursor\_result';

BEGIN

/\*

$1: batch name, empty mean all

$2: fieldname

\*/

var\_sql := '

SELECT data\_value as ' || quote\_ident($2) || ', amount

FROM sp\_split\_single\_word\_by\_field($1,$2)

WHERE data\_value ~\* ''[aeiou]{2}''

ORDER BY 2 DESC, 1

';

OPEN cursor\_result FOR

EXECUTE

(var\_sql) USING $1,

$2;

RETURN

cursor\_result;

END;

$function $

**Mục đích của Function:**

Function db\_23\_0853\_ancestry\_2023.sp\_check\_2\_consecutive\_vowel được tạo ra với mục đích kiểm tra xem có từ nào trong field name chứa 2 nguyên âm liền nhau và không có phụ âm nào ngay sau đó.

**Các tham số của Function:**

- batch\_name (kiểu character varying): tên lô, trường hợp giá trị rỗng sẽ áp dụng cho tất cả các lô.

- field\_name (kiểu character varying): Tên trường cần kiểm tra.

- num\_vowels (kiểu integer): Số lượng nguyên âm cần xuất hiện liền nhau, ví dụ: 2.

- num\_consonants (kiểu integer): Số lượng phụ âm không được xuất hiện ngay sau các nguyên âm liền nhau, ví dụ: 0.

**Giải thích từng phần của Function:**

1. Khai báo biến SQL để chứa câu lệnh SELECT.

var\_sql varchar;

2. Khởi tạo con trỏ kết quả.

cursor\_result refcursor := 'cursor\_result';

3. Bắt đầu khối chương trình của Function.

BEGIN

4. Sử dụng hàm format() để xây dựng câu lệnh SELECT với giá trị nhập vào.

var\_sql := format($$

SELECT data\_value AS %1$I, amount

FROM sp\_split\_single\_word\_by\_field('%2$s', '%1$I')

WHERE data\_value ~\*

'[aeiou]{%3$d}' ||

'[bcdfghjklmnpqrstvwxyz]{%4$d}' ||

'[^aeiou]\*$'

ORDER BY 2 DESC, 1

$$, field\_name, batch\_name, num\_vowels, num\_consonants);

Giải thích từng phần của var\_sql:

- data\_value và amount là các trường dữ liệu (column) mà câu lệnh SELECT sẽ tìm kiếm.

- sp\_split\_single\_word\_by\_field() được sử dụng để tách các từ trong trường field\_name.

- Trong cùng một điều kiện WHERE, chúng ta kiểm tra rằng chuỗi phải bắt đầu với số nguyên âm có độ dài num\_vowels và tiếp theo là không quá num\_consonants phụ âm. Kết thúc chuỗi có thể có hoặc không có ký tự không phải nguyên âm.

- ORDER BY sắp xếp kết quả trả về theo thứ tự giảm dần của cột amount, sau đó theo thứ tự tăng

dần của cột data\_value.

5. Mở con trỏ kết quả và thực thi câu lệnh SQL.

OPEN cursor\_result FOR EXECUTE var\_sql;

6. Trả về con trỏ kết quả.

RETURN cursor\_result;

7. Kết thúc khối chương trình của Function.

END;

Dưới đây là phiên bản function đã được chỉnh sửa để cho phép thay đổi số lượng nguyên âm/phụ âm cần lọc:

CREATE OR REPLACE FUNCTION db\_23\_0853\_ancestry\_2023.sp\_check\_consecutive\_vowel\_or\_consonant(character varying, character varying, **integer**) RETURNS refcursor LANGUAGE plpgsql

SET search\_path TO 'db\_23\_0853\_ancestry\_2023' AS $function$

DECLARE

var\_sql varchar;

cursor\_result refcursor := 'cursor\_result';

BEGIN

/\*

$1: batch name, empty mean all

$2: fieldname

**$3: consecutive count**

\*/

var\_sql := '

SELECT data\_value as ' || quote\_ident($2) || ', amount

FROM sp\_split\_single\_word\_by\_field($1,$2)

WHERE data\_value ~\* ''' || repeat('[aeiou]', **$3**) || ''' **OR**

**data\_value ~\* ''' || repeat('[bcdfghjklmnpqrstvwxyz]', $3) || '''**

ORDER BY 2 DESC, 1

';

OPEN cursor\_result FOR EXECUTE (var\_sql) USING $1, $2;

RETURN cursor\_result;

END;

$function$

Chúng ta đã thêm một tham số mới vào function để truyền số lượng liên tiếp của các nguyên âm hoặc phụ âm cần lọc. Nó được truyền vào qua tham số thứ ba trong function.

Thay vì tìm kiếm các từ có hai nguyên âm liền kề, câu lệnh WHERE của truy vấn SQL hiện đã được chỉnh sửa để sử dụng hàm repeat để tạo ra một chuỗi regex đại diện cho số lượng nguyên âm hoặc phụ âm cần tìm kiếm liên tiếp.

Bây giờ function có thể được sử dụng để truy xuất các từ có số lượng nguyên âm hoặc phụ âm liên tiếp bất kỳ.

Giải thích Function ở trên:

function này được thiết kế để truy xuất các từ trong một trường dữ liệu có chứa số lượng nguyên âm hoặc phụ âm liên tiếp cần tìm kiếm, và trả về một refcursor.

Dưới đây là giải thích chi tiết của từng câu lệnh:

- CREATE OR REPLACE: Đây là câu lệnh để tạo hoặc thay đổi một function trong cơ sở dữ liệu nếu nó đã tồn tại.

- FUNCTION db\_23\_0853\_ancestry\_2023.sp\_check\_consecutive\_vowel\_or\_consonant(character varying, character varying, integer) RETURNS refcursor LANGUAGE plpgsql: Đây là khai báo function với tên, các tham số đầu vào và kiểu dữ liệu trả về của nó. Ngôn ngữ được sử dụng ở đây là PL/PGSQL.

- SET search\_path TO 'db\_23\_0853\_ancestry\_2023': Câu lệnh này thiết lập biến môi trường "search\_path" trong cơ sở dữ liệu. Nó chỉ định schema trong database mà truy vấn sẽ được thực hiện.

- AS $function$: Đây là cú pháp để bắt đầu khai báo thân của function.

- DECLARE var\_sql varchar; cursor\_result refcursor := 'cursor\_result'; BEGIN: Đây là phần bắt đầu của thân function. Biến var\_sql được khai báo để lưu trữ chuỗi SQL được tạo ra sau đó để thực thi. Biến cursor\_result là một refcursor được khởi tạo bằng 'cursor\_result'.

- var\_sql := 'SELECT data\_value as ' || quote\_ident($2) || ', amount FROM sp\_split\_single\_word\_by\_field($1,$2) WHERE data\_value ~\* ''' || repeat('[aeiou]', $3) || ''' OR data\_value ~\* ''' || repeat('[bcdfghjklmnpqrstvwxyz]', $3) || ''' ORDER BY 2 DESC, 1': Đây là câu lệnh gán cho biến var\_sql. Nó chứa một câu lệnh SQL để truy vấn các từ có số lượng nguyên âm hoặc phụ âm liên tiếp cần tìm kiếm trong trường dữ liệu được chỉ định bởi tham số thứ hai (fieldname). Truy vấn này sử dụng hàm repeat để tạo một chuỗi regex đại diện cho số lượng nguyên âm hoặc phụ âm cần tìm kiếm liên tiếp. Câu lệnh WHERE sử dụng toán tử OR để kết hợp hai điều kiện với số lượng nguyên âm và phụ âm liên tiếp cần tìm kiếm.

- OPEN cursor\_result FOR EXECUTE (var\_sql) USING $1, $2;: Câu lệnh này mở refcursor cursor\_result và thực thi chuỗi SQL được lưu trữ trong biến var\_sql. Tham số đầu vào của truy vấn được chỉ định bằng cách sử dụng cú pháp "USING $1, $2" để tham chiếu đến các giá trị được truyền vào qua tham số của function.

- RETURN cursor\_result;: Cuối cùng, function trả về refcursor cursor\_result để truy cập kết quả truy vấn được lưu trữ trong đó.

- $function$: Đây là cú pháp để kết thúc khai báo function.

This is a PL/pgSQL function named ‘sp\_check\_consecutive\_vowel\_or\_consonant’ which takes in 3 parameters:

- ‘batch name’: A string representing the name of the batch. If an empty string is passed, it will return results for all batches.

- ‘fieldname’: A string representing the name of the field on which to perform the check.

- ‘consecutive count’: An integer representing the number of consecutive vowels or consonants to be checked.

The function first declares two variables: ‘var\_sql’, which will hold the SQL query to be executed, and ‘cursor\_result’, which will hold the reference cursor to be returned.

The SQL query consists of selecting the ‘data\_value’ and the count of occurrences of its consecutive vowels or consonants based on the provided ‘fieldname’ and ‘batch name’ parameters. It uses the custom split function named ‘sp\_split\_single\_word\_by\_field()’ to split the words and filter them according to the regex pattern that matches the specified number of consecutive vowels or consonants.

Finally, the function opens the ‘cursor\_result’ using the ‘OPEN’ statement with the SQL query as its argument and returns the ‘cursor\_result’.

Note that this function assumes that the ‘sp\_split\_single\_word\_by\_field()’ function has already been defined and is available in the same schema as the ‘sp\_check\_consecutive\_vowel\_or\_consonant()’ function.

Đây là một hàm PL/pgSQL được đặt tên sp\_check\_consecutive\_vowel\_or\_consonantcó 3 tham số:

batch name: Một chuỗi đại diện cho tên của lô. Nếu một chuỗi trống được truyền vào, nó sẽ trả về kết quả cho tất cả các đợt.

fieldname: Một chuỗi đại diện cho tên của trường để thực hiện kiểm tra.

consecutive count: Là số nguyên thể hiện số nguyên âm hoặc phụ âm liên tiếp cần kiểm tra.

Đầu tiên, hàm khai báo hai biến: var\_sql, sẽ giữ truy vấn SQL được thực thi và cursor\_result, sẽ giữ con trỏ tham chiếu được trả về.

Truy vấn SQL bao gồm việc chọn data\_valuevà số lần xuất hiện của các nguyên âm hoặc phụ âm liên tiếp của nó dựa trên các tham số fieldnamevà batch nametham số được cung cấp. Nó sử dụng chức năng phân tách tùy chỉnh có tên sp\_split\_single\_word\_by\_field()để phân tách các từ và lọc chúng theo mẫu biểu thức chính quy khớp với số nguyên âm hoặc phụ âm liên tiếp đã chỉ định.

Cuối cùng, hàm mở câu lệnh cursor\_resultsử dụng OPENvới truy vấn SQL làm đối số của nó và trả về tệp cursor\_result.

Lưu ý rằng chức năng này giả định rằng sp\_split\_single\_word\_by\_field()chức năng đã được xác định và có sẵn trong cùng một giản đồ như sp\_check\_consecutive\_vowel\_or\_consonant()chức năng.

Để thay đổi số lượng nguyên âm/phụ âm cần lọc trong function db\_23\_0853\_ancestry\_2023.sp\_check\_2\_consecutive\_vowel, bạn có thể thêm một tham số consecutive\_count vào hàm để cho phép người dùng truyền giá trị tùy ý.

Dưới đây là phiên bản mới của hàm với thêm tham số consecutive\_count:

CREATE OR REPLACE FUNCTION db\_23\_0853\_ancestry\_2023.sp\_check\_consecutive\_vowel\_or\_consonant(

batch\_name character varying,

field\_name character varying,

consecutive\_count integer

) RETURNS refcursor LANGUAGE plpgsql

SET search\_path TO 'db\_23\_0853\_ancestry\_2023' AS $function$

DECLARE

var\_sql varchar;

cursor\_result refcursor := 'cursor\_result';

BEGIN

/\*

batch\_name: the name of the batch, empty means all batches

field\_name: the name of the field to check

consecutive\_count: number of consecutive vowels or consonants to filter

\*/

var\_sql := format('

SELECT data\_value as %I, amount

FROM sp\_split\_single\_word\_by\_field($1, $2)

WHERE data\_value ~\* %L OR

data\_value ~\* %L

ORDER BY 2 DESC, 1

', field\_name, repeat('[aeiou]', consecutive\_count), repeat('[bcdfghjklmnpqrstvwxyz]', consecutive\_count));

OPEN cursor\_result FOR EXECUTE var\_sql USING batch\_name, field\_name;

RETURN cursor\_result;

END;

$function$

Như bạn có thể thấy, chúng tôi đã thêm một tham số consecutive\_count vào hàm và sử dụng hàm format() để tạo câu lệnh SQL với giá trị tham số được truyền vào. Hàm repeat() được sử dụng để tạo chuỗi ký tự của nguyên âm hoặc phụ âm theo số lượng cần lọc. Cuối cùng, chúng tôi thay thế các giá trị tương ứng trong biểu thức điều kiện của câu lệnh SQL.

Function db\_23\_0853\_ancestry\_2023.sp\_check\_consecutive\_vowel\_or\_consonant() được tạo ra để tìm kiếm các từ trong một trường dữ liệu cụ thể có số lượng nguyên âm hoặc phụ âm liên tiếp theo yêu cầu của người dùng. Dưới đây là giải thích chi tiết về các phần của hàm:

1. Tên và tham số của hàm:

CREATE OR REPLACE FUNCTION db\_23\_0853\_ancestry\_2023.sp\_check\_consecutive\_vowel\_or\_consonant(

batch\_name character varying,

field\_name character varying,

consecutive\_count integer

) RETURNS refcursor LANGUAGE plpgsql

SET search\_path TO 'db\_23\_0853\_ancestry\_2023' AS $function$

Đầu tiên, chúng ta khai báo tên của hàm và các tham số của nó. Hàm này nhận vào ba tham số:

- batch\_name: Tên của lô dữ liệu cần tìm kiếm. Nếu tham số này bị rỗng, hàm sẽ trả về kết quả cho tất cả các lô.

- field\_name: Tên trường dữ liệu cần tìm kiếm.

- consecutive\_count: Số lượng nguyên âm hoặc phụ âm liên tiếp cần tìm kiếm.

Hàm sử dụng kiểu trả về refcursor để trả về kết quả.

2. Khai báo biến và câu lệnh SQL:

DECLARE

var\_sql varchar;

cursor\_result refcursor := 'cursor\_result';

BEGIN

/\*

batch\_name: the name of the batch, empty means all batches

field\_name: the name of the field to check

consecutive\_count: number of consecutive vowels or consonants to filter

\*/

var\_sql := format('

SELECT data\_value as %I, amount

FROM sp\_split\_single\_word\_by\_field($1, $2)

WHERE data\_value ~\* %L OR

data\_value ~\* %L

ORDER BY 2 DESC, 1

', field\_name, repeat('[aeiou]', consecutive\_count), repeat('[bcdfghjklmnpqrstvwxyz]', consecutive\_count));

Sau đó, chúng ta khai báo hai biến: var\_sql để lưu trữ câu lệnh SQL và cursor\_result để lưu trữ kết quả tìm kiếm. Câu lệnh SQL được tạo ra bằng cách sử dụng hàm format() để thêm các giá trị của các tham số vào trong câu lệnh SQL. Câu lệnh này sử dụng hàm sp\_split\_single\_word\_by\_field() để tách từ trong trường dữ liệu cần tìm kiếm. Sau đó, nó sử dụng một biểu thức điều kiện để lọc các từ có số lượng nguyên âm hoặc phụ âm liên tiếp theo yêu cầu của người dùng. Cuối cùng, kết quả được sắp xếp theo thứ tự giảm dần của số lượng nguyên âm hoặc phụ âm và tăng dần của từ.

3. Mở con trỏ và trả về kết quả:

OPEN cursor\_result FOR EXECUTE var\_sql USING batch\_name, field\_name;

RETURN cursor\_result;

END;

$function$

Cuối cùng, chúng ta mở con trỏ và thực thi câu lệnh SQL bằng cách sử dụng EXECUTE trong hàm OPEN. Kết quả tìm kiếm được gán cho biến cursor\_result và sau đó trả về bằng cách sử dụng lệnh RETURN.