bimanu
Business Intelligence Manufaktur

« Blog

```
CREATE OR REPLACE PROCEDURE fill_numbers_2()
RETURNS VARCHAR LANGUAGE JAVASCRIPT
AS
$$
 var sql_command = "INSERT INTO NUMBERS (ARABIC_NUMBER) VALUES ";
 var args = [];
 for(var i = 1; i <= 2995; i++) args.push('(' + String(i) + ')');
 snowflake.execute ({sqlText: sql_command + args.join(',')});
$$;
```

Erfahren Sie welche Möglichkeiten die neuen Stored Procedures in Snowflake bieten

13.05.2019 · Snowflake, Stored Procedures, JavaScript

Jon Nedelmann

Seit Kurzem gibt es in snowflake die Möglichkeit, Stored Procedures zu erstellen und natürlich auch auszuführen.

Analog zu den UDFs (User Defined Functions), die dem Entwickler schon seit längerer Zeit zur Verfügung stehen, wird als Basis-Programmiersprache JavaScript verwendet. Das mag dem einen oder anderen, der zum Beispiel die prozeduralen Erweiterungen von Oracle (PL/SQL) oder von Microsofts SQL Server (Transact-SQL) kennt, verwundern: Ist doch JavaScript eine Sprache, die eher von der Webanwendungsentwicklung bekannt ist, und bisher kaum mit nativer Datenbankprogrammierung in Verbindung gebracht wurde.

In diesem Blog möchte ich an einigen einfachen Beispielprozeduren zeigen, wie klassische prozedurale Themen in snowflake behandelt werden können, an anderen Beispielen aber auch, welche weiteren Möglichkeiten mit JavaScript zur Verfügung stehen.

Zur Vorbereitung lege ich zunächst zwei Tabellen an, die in einer Spalte ganze positive Zahlen und in einer zweiten Spalte das Pendant in römischer Schrift speichern:

> CREATE TABLE NUMBERS(ARABIC_NUMBER INT, ROMAN_NUMBER VARCHAR);
>
> CREATE TABLE NUMBERS_OF_DAY(ARABIC_NUMBER INT, ROMAN_NUMBER VARCHAR);

bimanu
Business Intelligence Manufaktur

```
CREATE OR REPLACE PROCEDURE fill_numbers()
RETURNS VARCHAR LANGUAGE JAVASCRIPT
AS
$$
 for(var i = 1; i <= 2995; i++) {
  var sql_command = "INSERT INTO NUMBERS (ARABIC_NUMBER) VALUES (" +    String(i) + ")";
  snowflake.execute ({sqlText: sql_command});
 }
$$;
```

Eine Prozedur wird also in einem Rahmen

```
CREATE OR REPLACE PROCEDURE <procedure_name>(<parameter>)
 RETURNS VARCHAR LANGUAGE JAVASCRIPT
 AS
 $$
 …
 $$;
```

erzeugt. Die Zeile RETURNS VARCHAR… ist Teil der snowflake-Syntax für die Prozedur und sollte nicht so gelesen werden, dass diese Prozedur ein Ergebnis vom Typ VARCHAR zurückgibt (Prozeduren können aber tatsächlich Werte zurückgeben, die snowflake-Dokumentation gibt mehrere Beispiele).

Die Prozedur kann durch ein einfaches CALL-Statement aufgerufen werden:

call fill_numbers();

Der Block $$ … $$; umfasst den JavaScript-Code. In diesem Block wird 2.995 mal ein INSERT-Statement erzeugt, das dann auch innerhalb der Schleife ausgeführt wird. Das funktioniert, ist aber sehr, sehr langsam. Mit einem XS Warehouse lief die Prozedur 21 Minuten und 38 Sekunden. Das hat folgenden Grund: Der Aufruf

snowflake.execute ({sqlText: sql_command});

führt das angegebene Statement in einer eigenen gekapselten Transaktion aus (deshalb wird auch kein COMMIT benötigt). Das sequentielle Abarbeiten von diesen vielen Transaktionen braucht halt seine Zeit.

II. Bulk-Load Statement in einer FOR-Schleife erzeugen

Bevor nun aber das Warehouse nach oben skaliert wird, um die Performance zu steigern, gibt es einen anderen Weg. Die folgende Prozedur kommt zu demselben Ergebnis, aber auf eine andere Weise: in der FOR-Schleife wird ein INSERT-Statement für den BULK-Load zusammengestellt und im Anschluss ausgeführt. Die Prozedur benötigt in diesem Fall weniger als 2 Sekunden.

bimanu
Business Intelligence Manufaktur

```
  var sql_command = "INSERT INTO NUMBERS (ARABIC_NUMBER) VALUES ";
  var args = [];
  for(var i = 1; i <= 2995; i++) args.push('(' + String(i) + ')');
  snowflake.execute ({sqlText: sql_command + args.join(',')});
 $$;
```

III. JSON-Objekte als Parameter

Da JSON-Objekte durch den Datentyp VARIANT Bürger erster Klasse sind, kann auch eine Liste übergeben werden, und aus dieser direkt das Statement ergänzt werden. Eine Variante der vorherigen Prozedur sieht dann so aus:

```
CREATE OR REPLACE PROCEDURE fill_numbers_3(moreNumbers VARIANT)
 RETURNS VARCHAR LANGUAGE JAVASCRIPT
 AS
 $$
  var sql_command = "INSERT INTO NUMBERS (ARABIC_NUMBER) VALUES " + MORENUMBERS.map(n => '(' + String(n) + ')').join(',');
  snowflake.execute ({sqlText: sql_command});
 $$;
```

und kann so aufgerufen werden:

call fill_numbers_3(PARSE_JSON('[2996, 2997, 2999, 3000]'));

Der Parameter war übrigens in der in JavaScript üblichen camel case Notation als moreNumbers deklariert. Innerhalb der Prozedur muss die Variable komplett in Großbuchstaben als MORENUMBERS geschrieben werden; more_numbers wäre also der bessere Parameternamen gewesen.

IV. Daten mit einem Cursor auslesen und zeilenweise verarbeiten

Eine typische Verarbeitung in klassischen prozeduralen Erweiterungen von SQL ist es, einen Cursor zu definieren und die mit diesem Cursor erhaltenen Daten zeilenweise zu verarbeiten. Eine Prozedur könnte dann ungefähr so aussehen:

```
CREATE OR REPLACE PROCEDURE WORK_WITH_CURSOR()
IS
 CURSOR c is ....

BEGIN

  FOR x in c(...)
  LOOP
   -- Verarbeite einzelnen Datensatz
  END LOOP;

END;
```

Auch das ist im Prinzip in snowflake möglich, wird aber ein wenig anders gehandhabt. Zunächst schreibe ich eine Prozedur, welche die Zahlen, die wir gerade ergänzt haben, wieder ausliest:

bimanu
Business Intelligence Manufaktur

```
$$
 var selectArabic = "SELECT ARABIC_NUMBER FROM NUMBERS";
 var result = snowflake.execute({sqlText: selectArabic});

 while(result.next()) {
  var arabicNumber = result.getColumnValue(1);
  // mach was mit der Zahl
 }
$$;
```

Das Ergebnis der Abfrage wird hier in die Variable *result* geschrieben, über *result.next()* kann jeweils der nächste Datensatz gelesen und über *result.getColumnValue(n)* der Wert der n-ten Spalte ausgelesen werden – die klassische Cursor-Verarbeitung. Nun soll zu jeder Zahl die römische Zahldarstellung ermittelt werden. Wie kann nun an welchen Stellen der passende JavaScript-Code ergänzt werden? Die snowflake-Dokumentation gibt dazu folgende Auskunft:

The JavaScript code must define a single literal JavaScript object for the stored procedure to be valid.

Der Prozedurcode kann wie ein einziges Objekt betrachtet werden. Das bedeutet unter anderem, dass nach Belieben Funktionalitäten in separaten Funktionen ergänzt werden können. Zur Berechnung der römischen Ziffern ergänze ich also zwei Hilfsfunktionen *times* und *parts* und die Funktion *to_roman*, welche dann die eigentliche Umrechnung durchführt (zumindest bis 3.000). In der Schleife ergänze ich dann einfach einen Aufruf an *to_roman* und erstelle mit dem Ergebnis ein UPDATE-Statement. Der fertige Code sieht dann folgendermaßen aus:

bimanu
Business Intelligence Manufaktur

```
function times(n, digit) {
 if(n <= 1) return digit;
 else return digit + times(n-1, digit);
}

function parts(digit, oneSymbol, fiveSymbol, tenSymbol) {
 if(digit == 9) return oneSymbol + tenSymbol;
 else if (digit > 4) return fiveSymbol + times(digit - 5, oneSymbol);
 else if (digit == 4) return oneSymbol + fiveSymbol;
 else return times(digit, oneSymbol);
}

function toRoman(n) {
 if(n > 3000 || n < 1) return "NN";
 else {
  var result = "";

  if(n > 999) {
   result = times(Math.floor(n / 1000), 'M');
   n = n % 1000;
  }

  if(n > 99) {
   result = result + parts(Math.floor(n / 100), 'C', 'D', 'M');
   n = n % 100;
  }

  if(n > 9) {
   result = result + parts(Math.floor(n / 10), 'X', 'L', 'C');
   n = n % 10;
  }

  result = result + parts(n, 'I', 'V', 'X');
 }

 return result;
}


var selectArabic = "SELECT ARABIC_NUMBER FROM NUMBERS";
var result = snowflake.execute({sqlText: selectArabic});


while(result.next()) {
 var arabicNumber = result.getColumnValue(1);
 var updateStatement = "UPDATE NUMBERS SET ROMAN_NUMBER = '" + toRoman(arabicNumber) + "' WHERE ARABIC_NUMBER = " + arabicNumber;
 snowflake.execute ({sqlText: updateStatement});
}
$$;
```

Ein Blick in die Tabelle NUMBERS zeigt, dass wir das gewünschte Ergebnis erhalten haben. Die Performance dieser Prozedur ist aber wieder alles andere als akzeptabel gewesen: alle 3.000 UPDATE-Statements werden separat durchgeführt, das braucht seine Zeit. Hier sehen wir die aktuellen Begrenzungen der CURSOR-Verarbeitung in snowflake: Innerhalb der Schleife sollten keine DML-Statements ausgeführt werden. In unserem Fall könnten wird die Prozedur schnell umschreiben, dass arabische und römische Ziffern in einem einzigen BULK-INSERT geladen werden.

V. Dynamisches SQL ausführen

Eine weitere typische Anwendung von Datenbank-Prozeduren besteht darin, dynamisch eine SQL-Anweisung zusammenzustellen und sie dann innerhalb der Prozedur

bimanu
Business Intelligence Manufaktur

snowflake.execute ({sqlText: sql_command});

Dieser Befehl hat noch weitere Optionen. Wir können in dem SQL-Befehl für noch nicht bekannte Parameter ein Fragezeichen setzen und dann bei der Befehlsausführung diesen Parameter binden. Die folgende Prozedur zeigt das Vorgehen. Sie kopiert montags alle Zahlen von der Tabelle NUMBERS in die Tabelle NUMBERS_OF_DAY, dienstags alle durch 2 teilbaren Zahlen, mittwochs alle durch 3...

```
CREATE OR REPLACE PROCEDURE copy_by_day()
RETURNS VARCHAR LANGUAGE JAVASCRIPT
AS
$$
 var dayOfWeek;
 var result = snowflake.execute({sqlText: "select dayofweek(current_date) from dual"});
 if(result.next()) dayOfWeek = result.getColumnValue(1);

 var copyStatement = "INSERT INTO NUMBERS_OF_DAY SELECT * FROM NUMBERS where ARABIC_NUMBER % ? = 0";
 snowflake.execute ({sqlText: copyStatement, binds:[dayOfWeek]});

$$;
```

VI. Fazit

Mit der Einführung der stored procedures hat snowflake eine große Lücke geschlossen. Für Entwickler, die aus der Oracle oder SQL Server-Ecke kommen, ist ein wenig Umdenken gefragt, um sich auf JavaScript als Programmiersprache einzulassen. Es lohnt sich aber, denn mit wenigen Zeilen können dann elegante Prozeduren erstellt werden. Hilfreich bei diesem Umstieg ist die wirklich gute snowflake-Dokumentation zu diesem Thema.

Schwachstelle ist bisher noch, dass jeder Aufruf eines DML-Statements in der Prozedur - sowie jeder Prozeduraufruf selbst – als eine Transaktion behandelt wird, und dann keine gute Performance zu erwarten ist. Die „klassische Cursor-Verarbeitung“ sollte dann anders gestaltet werden. Aber noch sind Prozeduren ja ein ganz neues Thema für snowflake, und es wird sich sicherlich in der nächsten Zeit weiterentwickeln.

First Name*

Swen

Last Name*

Göllner

Email*

bimanu
Business Intelligence Manufaktur

geschützt durch reCAPTCHA
Datenschutzerklärung -
Nutzungsbedingungen

Submit Comment

SAP Silver Partner
SAP® Certified Associate
snowflake
bimanu Business Intelligence Manufaktur snowflake CERTIFIED Associate

bimanu
Business Intelligence Manufaktur

Kontakt:

Bickenbachstraße 38

41462 Neuss
Deutschland

+49 (0) 2131 / 74 211 74
info@bimanu.de

Weiteres:

LinkedIn
Xing
Facebook
Instagram

bimanu
Business Intelligence Manufaktur

Startseite
bimanu Cloud