Zunächst Präsentation der Möglichkeiten (SQL Funktionen), mittels **IDAA** (IBM Db2 Analytics Accelerator) reguläre Ausdrücke im Db2 für z/OS zu benutzten.

Anschließend wird eine UDF (user-defined function) vorgestellt, mit der man reguläre Ausdrücke auch dann benutzen kann, wenn kein IDAA zur Verfügung steht.

Reguläre Ausdrücke

- Beispiel:

Zeige alle Produkt-IDs an,
deren Format ungültig ist, also nicht die **Form 'nnn-nnn-nn'** (mit n=0,..,9) hat:

```
SELECT pid
  FROM product
 WHERE NOT
       REGEXP_LIKE(pid, '[0-9]{3}-[0-9]{3}-[0-9]{2}' ) = 1
```

neu ab V12R1M**504**

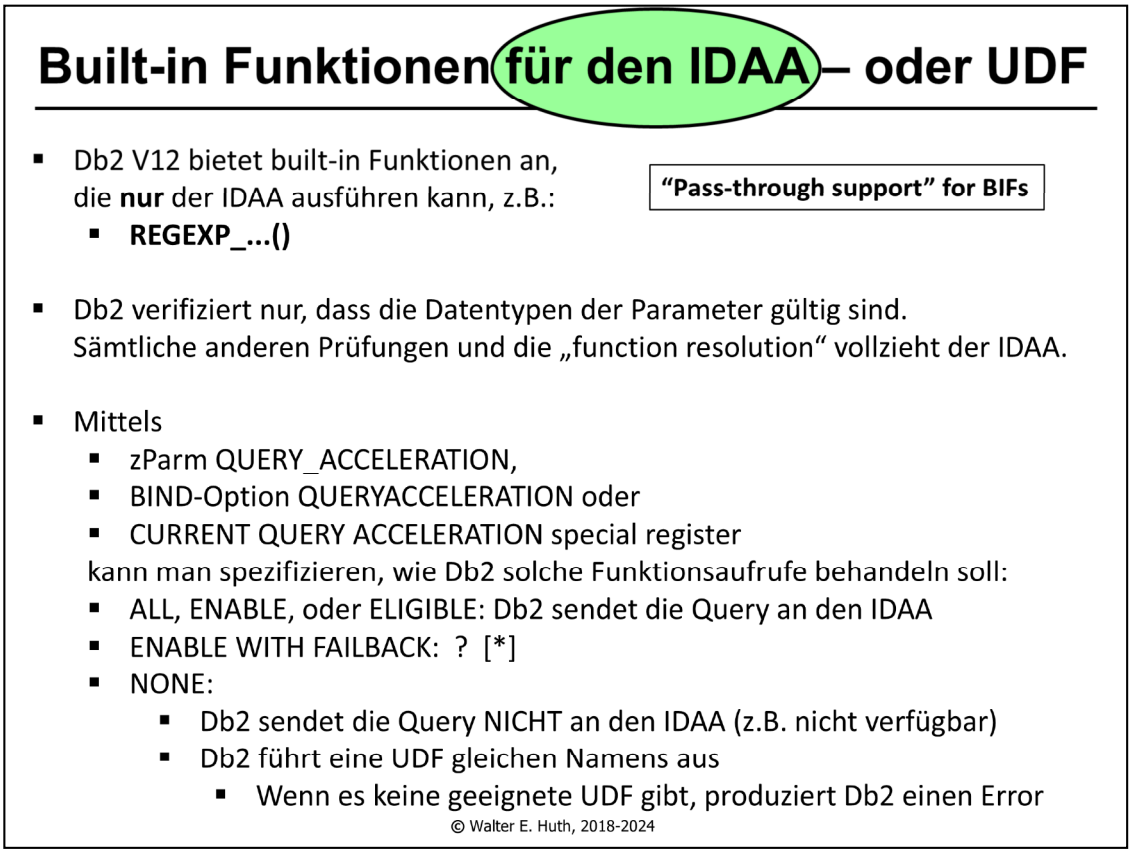
- Erläuterungen zu regulären Ausdrücken (regular expressions): siehe Web
- Im Folgenden:

1. Reguläre Ausdrücke im Db2 für z/OS mit Hilfe des IDAA

2. Reguläre Ausdrücke im Db2 für z/OS mit eigener Funktion (ohne IDAA)

IDAA: IBM Db2 Analytics Accelerator for z/OS

Eine Abfrage wie im obigen Beispiel wäre mit den bisherigen Db2-Funktionen äußerst aufwendig. Die Möglichkeit des Einsatzes von Regulären Ausdrücken ist also sehr vorteilhaft.



„Pass-through support" für built-in Funktionen:
IDAA basiert auf einem DBMS (Db2 Warehouse), das Funktionen ausführen kann, die Db2 für z/OS (noch) nicht ausführen kann. Wenn dem Db2 ein IDAA zur Verfügung steht und wenn eine solche Funktion aufgerufen wird, erfolgt im Db2 eine Prüfung der Signatur und u.U. ein „Query Rewrite", sodann sendet Db2 die Query zum IDAA zwecks Ausführung.

„function resolution" (Funktionsauswahl): Auswahl der aufzurufenden Funktion aufgrund des Funktionsnamens, der Parameter etc.

[*] Hier widersprechen sich die IBM-Dokumentationen:
https://www.ibm.com/docs/en/db2-for-zos/12?topic=zos-how-db2-determines-whether-accelerate-eligible-queries#accel-passthrough-only
https://www.ibm.com/docs/en/daafz/7.5.5?topic=accelerator-configuring-db2-zos-acceleration-dynamic-sql-queries

Im Folgenden wird eine UDF definiert, mittels derer man mit regulären Ausdrücken suchen kann. Man könnte diese UDF auch so nennen wie eine der built-in Funktionen, um letztere zu ersetzen. Dies wird hier/in dieser Präsentation aber nicht gezeigt.

Reguläre Ausdrücke in SQL – ab FL 504

- Fünf Skalarfunktionen, die reguläre Ausdrücke in SQL-Statements unterstützen:

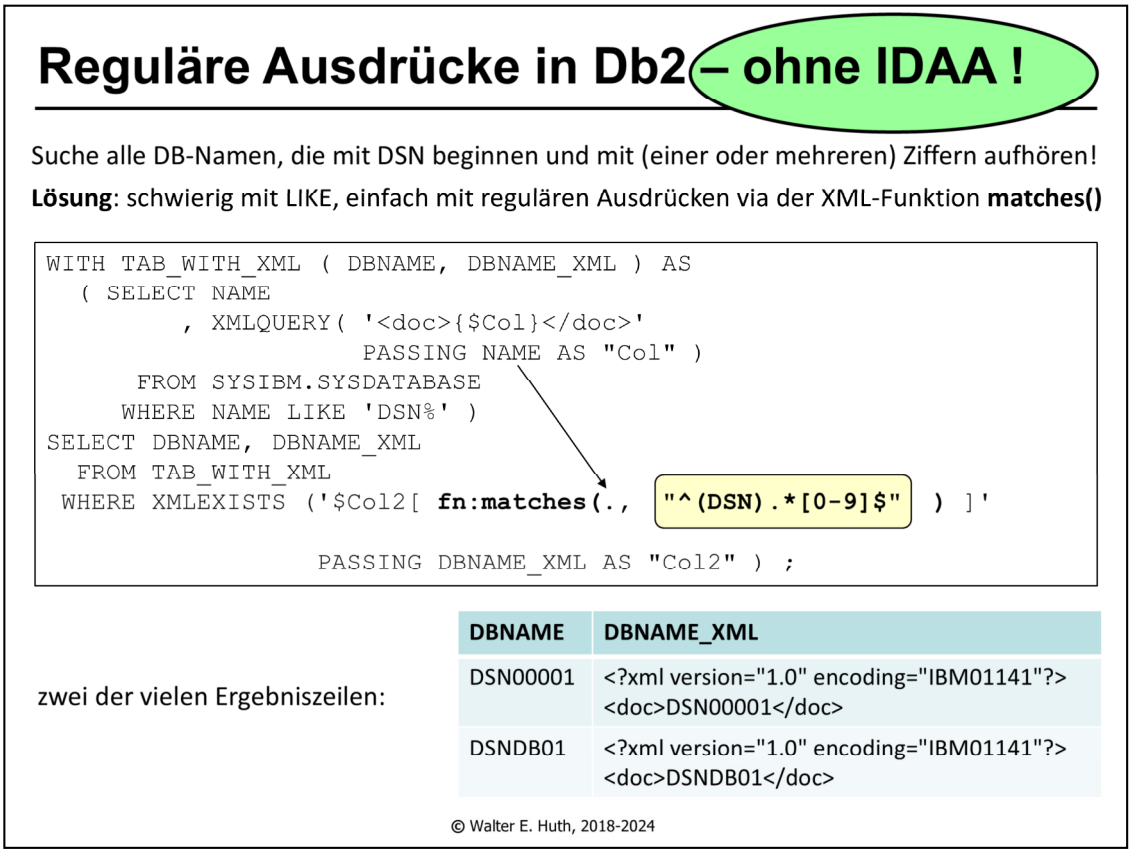
```
REGEXP_LIKE()
REGEXP_COUNT()
REGEXP_INSTR()
REGEXP_REPLACE()
REGEXP_SUBSTR()
```

neu ab V12R1M**504**

- Details zu den fünf Funktionen in der SQL Ref.

Die Funktion **REGEXP_LIKE()**
- liefert 1 (=wahr) oder 0 (=falsch)
- kann noch mehr Input-Parameter haben,
  z.B. das Flag '**i**' für case-**i**nsensitive String-Vergleiche.

REGEXP_COUNT() zeigt an, wie oft Strings (z.B. die Werte einer Spalte) einem vorgegebenem Muster (eines regulären Ausdrucks) entsprechen.
REGEXP_INSTR() zeigt die Position eines vorgegebenen Musters (eines regulären Ausdrucks) innerhalb zu untersuchender Strings an.
REGEXP_REPLACE() verändert zu untersuchende Strings dergestalt, dass Substrings, die einem zutreffenden Muster (eines regulären Ausdrucks) entsprechen, durch einen anderen String ersetzt werden.
REGEXP_SUBSTR() gibt einen Substring aus, der einem Muster (eines regulären Ausdrucks) entspricht.



Wenn IDAA nicht zur Verfügung steht, kann man dennoch mit regulären Ausdrücken arbeiten – und zwar mittels der integralen Komponente pureXML des Db2 (verfügbar seit Db2 V9), die u.a. XPath unterstützt.

Erläuterung der obigen Query: Die XPath-Funktion fn:matches() kann man nur auf XML-Dokumente anwenden, daher generiert die Common Table Expression (CTE) „TAB_WITH_XML“ mittels der Funktion XMLQUERY() zunächst eine Spalte, die den DB-Namen als XML-Dokument enthält. [Das Label („tag“), hier „doc“, ist frei wählbar.] Daraufhin kann man dann die Funktion fn:matches() anwenden, die die Suche mit Regulären Ausdrücken unterstützt, die ein Muster für Strings vorgeben, hier das Muster „^(DSN).*[0-9]$“. Es bedeutet, der String muss mit „DSN“ beginnen („^“) und mit einer Ziffer von 0 bis 9 enden(„$“). Dazwischen können beliebige Zeichen („.“) beliebig oft („*“) vorkommen. Die Funktion fn:matches() wird im Prädikat XMLEXISTS() verwendet, um nur die zutreffenden Reihen auszugeben.

Anmerkungen:

1. Die Idee, matches() zu verwenden, basiert auf einem Artikel von Jane Man (IBM) im Web, der inzwischen aber nicht mehr verfügbar ist.
2. Die Bedingung NAME LIKE ‚DSN%‘ in der CTE ist optional, sie bewirkt nur eine Vorauswahl zwecks Performance.
3. Im verwendeten Test-Db2 [auch sonst?] haben DSNDB04 und DSNDB06 am Ende ein Blank im Namen – diese Datenbank-Namen werden also nicht aufgeführt.

UDF für reguläre Ausdrücke – ohne IDAA (1)

- Ziel: Benutzung von regulären Ausdrücken – ohne XML/XPath kennen zu müssen
- Wunsch: Eine UDF **islike_regexpr()**, die 1 (sonst 0) ergibt,
  wenn 1.Parameter einem regulären Ausdruck entspricht, den man als 2.Paramter eingibt
- Einsatz:

```
SELECT islike_regexpr(deptname, '.*[0-9].*' ) AS match
     , deptname
  FROM DSN81210.DEPT
```

MATCH	DEPTNAME
0	SPIFFY COMPUTER SERVICE DIV.
etc.	
1	BRANCH OFFICE I2
1	BRANCH OFFICE J2

```
SELECT deptname
  FROM DSN81210.DEPT
 WHERE islike_regexpr(deptname, '.*[0-9].*' )
       = 1
```

DEPTNAME
BRANCH OFFICE F2
BRANCH OFFICE G2
BRANCH OFFICE H2
BRANCH OFFICE I2
BRANCH OFFICE J2

Nehmen wir also an, dass kein IDAA zur Verfügung steht.

Im Folgenden wird eine UDF erstellt, die es dennoch ermöglicht, im Db2 reguläre Ausdrücke bequem – d.h. ohne Kenntnisse von XPath-Funktionen – zu benutzen.

Hier sieht man zunächst die Verwendung dieser UDF, einmal in der SELECT-Klausel, einmal in der WHERE-Klausel (Suche nach Department-Namen, die eine Ziffer enthalten).

UDF für reguläre Ausdrücke – ohne IDAA (2)

- Wunsch: Die UDF **islike_regexpr()**, die 1 (sonst 0) ergibt,
  wenn 1.Parameter einem regulären Ausdruck entspricht, den man als 2.Paramter eingibt
- Einsatz:

```
SELECT deptname
  FROM DSN81210.DEPT
 WHERE islike_regexpr(deptname, '.*[0-9].*' )
       = 1
```

DEPTNAME
BRANCH OFFICE F2
BRANCH OFFICE G2
etc.

- Lösungsidee:
  Die UDF erhält für den 1. Parameter (deptname) bei der ersten DEPT-Reihe
  den Wert 'SPIFFY COMPUTER SERVICE DIV.'
  Hiermit und mit dem 2. „konstanten“ Parameter '.*[0-9].*'
  sollte die UDF für diese DEPT-Reihe folg. Statement generieren und ausführen:

```
WITH temp1 (parm1_as_xml_col) AS
( SELECT XMLQUERY ('<doc>{$p1}</doc>'
                PASSING 'SPIFFY COMPUTER SERVICE DIV.' AS "p1" )
    FROM SYSIBM.SYSDUMMY1 )
SELECT SMALLINT(COUNT(*))
  FROM temp1
 WHERE XMLEXISTS ('$px1[fn:matches(.,".*[0-9].*" ) ] '
             PASSING parm1_as_xml_col AS "px1" )
```


Zum vorgestellten Statement: Die CTE temp1 enthält nur eine Zeile; der Punkt hinter matches ist ein Platzhalter für den aktuellen Wert; somit lautet das Statement „im Wesentlichen“:

```
SELECT COUNT(*) ..   -- ergibt 1 oder 0
WHERE XMLEXISTS [ matches ('SPIFFY COMPUTER SERVICE DIV.', ".*[0-9].*" )  ]
```

Auf der folgenden Seite die Definition dieser UDF.

UDF für reguläre Ausdrücke – ohne IDAA (3)

```
CREATE FUNCTION islike_regexpr
( parm1 VARCHAR(256), regexpr VARCHAR(256) )
RETURNS SMALLINT
LANGUAGE SQL
READS SQL DATA
BEGIN
DECLARE result SMALLINT ;
DECLARE stmt_str VARCHAR(1000) ;
DECLARE s1 STATEMENT ;
DECLARE c1 CURSOR FOR s1 ;
SET parm1 = REPLACE(parm1,'''',''''''') ;
SET stmt_str =
'WITH temp1 (parm1_as_xml_col) AS '            CONCAT
' ( SELECT XMLQUERY (''<doc>{$p1}</doc>'' '    CONCAT
' PASSING '''                                  CONCAT
parm1                                          CONCAT
''' AS "p1" )'                                 CONCAT
' FROM SYSIBM.SYSDUMMY1 ) '                    CONCAT
' SELECT SMALLINT(COUNT(*)) FROM temp1 '       CONCAT
' WHERE XMLEXISTS (''$px1[fn:matches(.,"''     CONCAT
regexpr                                        CONCAT
'" ) ] '' PASSING parm1_as_xml_col AS "px1" ) ' ;
```

```
.. Fortsetzung ..

PREPARE s1 FROM stmt_str ;
OPEN c1 ;
FETCH c1 INTO result ;
CLOSE c1 ;
RETURN result ;
END
```

Zum Ersatz der built-in Funktion REGEXP_LIKE() müsste die Funktion genauso heißen !

Diese UDF verwendet dynamisches SQL, was seit V12 möglich ist.

Für die erste DEPT-Reihe ergibt der Statement-String (stmt_str) mit den zwei Parametern

a) deptname = 'SPIFFY COMPUTER SERVICE DIV.'

b) dem regulären Ausdruck: `'.*[0-9].*'`

das Statement:

```
WITH temp1 (parm1_as_xml_col) AS
( SELECT XMLQUERY ('<doc>{$p1}</doc>'
                PASSING 'SPIFFY COMPUTER SERVICE DIV.' AS "p1" )
    FROM SYSIBM.SYSDUMMY1 )
SELECT SMALLINT(COUNT(*))
  FROM temp1
 WHERE XMLEXISTS ('$px1[fn:matches(.,".*[0-9].*" ) ] '
             PASSING parm1_as_xml_col AS "px1" )
```

Für den Fall, dass der 1. Parameter selbst Apostrophs enthält, wird der vorhergehende Befehl benötigt (Genaueres folgt unter „Zusatz: Erläuterung zu REPLACE “) :

```
SET parm1 = REPLACE ...
```

1. Zusatz: Test des Statement-Strings

Wie kann man testen/sicherstellen,
dass der in der UDF generierte Statement-String
tatsächlich zu der gewünschten Query wird?

Indem man die UDF das generierte Statement anzeigen lässt,
und zwar durch drei Änderungen in der UDF:

1. Auf Kommentar-Setzen von: RETURNS SMALLINT ;
   und stattdessen Einfügen von: RETURNS VARCHAR(1000) ;
2. Auf Kommentar-Setzen der vier Zeilen:
   PREPARE s1 FROM stmt_str ;
   OPEN c1 ;
   FETCH c1 INTO result ;
   CLOSE c1 ;
3. Auf Kommentar-Setzen von: RETURN result ;
   und stattdessen Einfügen von: RETURN stmt_str ;

Wenn man dann die UDF in der SELCT-Klausel aufruft, wird sie “stmt_str” ausgeben.

Diese Kommentieren/Auskommentieren mag helfen,
wenn man das SELECT Statement verändern und diese Änderung testen will.

2. Zusatz: Erläuterung zu REPLACE (1/2)

Wenn man die Spalte LASTNAME (von EMP) statt der Spalte DEPTNAME (von DEPT) untersucht,
und zwar mit dem simpleren regulären Ausdruck '^(H).*'
- dann führt die UDF zu einem Fehler:

```
SELECT lastname
FROM DSN81210.EMP
WHERE islike_regexpr(lastname, '^(H).*' ) = 1
---------+---------+---------+---------+---------+---------+---------+-------
LASTNAME
---------+---------+---------+---------+---------+---------+---------+-------
HAAS
HENDERSON
DSNE610I NUMBER OF ROWS DISPLAYED IS 2
DSNT408I SQLCODE = -104, ERROR: ILLEGAL SYMBOL "' AS parm1_as_col FROM
SYSIBM.SYSDUMMY1 ) , temp2 (parm1_as_xml_col) AS ( SELECT XMLQUERY
('". SOME SYMBOLS THAT MIGHT BE LEGAL ARE: , FROM INTO
```

Problem/Grund: Ein Nachname ist “O’Connel” – und der enthält ein Apostroph.
Lösungsidee: Ein Apostroph im Input-Parameter vorher verdoppeln!
“Medizin”: Folgendes SET-Statement zwischen “DECLARE ..” und “SET stmt-str ..” hinzufügen:

```
DECLARE c1 CURSOR FOR s1 ;
SET parm1 = REPLACE(parm1,'''','''''') ;
SET stmt_str = ..
```

2. Zusatz: Erläuterung zu REPLACE (2/2)

Damit wird der Statement-String der UDF zu (beachte die 2 Apostrophs in O''Connel):

```
WITH temp1 (parm1_as_xml_col) AS
( SELECT XMLQUERY ('<doc>{$p1}</doc>'
          PASSING 'O''CONNELL' AS "p1" )
   FROM SYSIBM.SYSDUMMY1 )
SELECT SMALLINT(COUNT(*))
   FROM temp1
  WHERE XMLEXISTS ('$px1[fn:matches(.,"^(H).*" ) ] '
                PASSING parm1_as_xml_col AS "px1" )
```

Und die obige Query produziert nun das folgende Resultat (fehlerlos):

```
---------+---------+---------+---------+---------+---------+---------+-------
LASTNAME
---------+---------+---------+---------+---------+---------+---------+-------
HAAS
HENDERSON
HEMMINGER
DSNE610I NUMBER OF ROWS DISPLAYED IS 3
---------+---------+---------+---------+---------+---------+---------+-------
```


Abgebildet ist der Statement-String der UDF, wenn die EMP-Reihe mit dem Namen „O‘Connel“ verarbeitet wird.

3. Zusatz: Performance-Überlegungen

Bei großen Tabellen sollte man – wenn möglich –
die zu untersuchenden Zeilen (z.B. mittels LIKE-Prädikaten) zuvor reduzieren,
statt die vorgeschlagene UDF auf dem Gesamt-Bestand arbeiten zu lassen.

Ungelöste Frage:
Ist es möglich, einen Parameter-Marker "?" statt des "parm1" im Statement-String zu verwenden?
Dies wäre – hinsichtlich des DSC – vorteilhaft
(zusammen mit der UDF-Option CONCENTRATE STATEMENTS WITH LITERALS).

Problem:
Wenn man "parm-2" durch "?" ersetzt, erhält man:

```
-418 A STATEMENT STRING TO BE PREPARED CONTAINS AN INVALID USE OF PARAMETER MARKERS
Explanation [verkürzt]:
The statement cannot be executed because a parameter marker has been used in an invalid way.
Examples of places where parameter markers cannot be used:
- As an argument to an XMLEXISTS predicate.
```

Es ist zu befürchten, dass dasselbe auch für XMLQUERY gilt.
Daher keine weiteren Versuche/Nachforschungen meinerseits.
Evtl. haben Sie eine Lösungsidee?!