CCDUG 2024

Functioning in Db2

Chris Crone – cjc@broadcom.com

Broadcom

Session Code: <Insert Code Here> | Platform: <Insert Platform(s) here>

Agenda

- Overview of Functions
- Digging Deeper
  - Aggregate Functions
  - Scalar Functions
  - Table Functions
  - ROW – UNPACK (unpacks a row built by PACK function)
  - OLAP Specifications
  - User Defined Functions
  - XML Functions – Not covered
- SQL Data Insights
- Conclusion

Overview

- Functions in Db2 are a simple way to enhance your SQL statements with capability that enables you to aggregate or modify the data that is returned to the application or user.
- Db2 Supports a wide variety of functions and we will discuss some of the more interesting and useful functions in this session.
- Db2 also supports User Defined functions that can be written in host languages, as well as SQL PL

Aggregate Functions

Aggregate Functions

- ARRAY_AGG
- AVG
- CORR or CORRELATION
- COUNT
- COUNT_BIG
- COVAR or COVARIANCE or COVAR
- COVAR_SAMP or COVARIANCE_SAMP
- CUME_DIST
- GROUPING
- LISTAGG
- MAX
- MEDIAN
- MIN
- PERCENTILE_CONT
- PERCENTILE_DISC
- PERCENT_RANK
- Regression functions (REGR_*)
- STDDEV_POP or STDDEV
- STDDEV_SAMP
- SUM
- VAR_POP or VARIANCE or VAR
- VAR_SAMP or VARIANCE_SAMP
- XMLAGG

Underscored functions are IDAA only

Statistical Aggregate Functions

- ARRAY_AGG
- AVG
- CORR or CORRELATION
- COUNT
- COUNT_BIG
- COVAR or COVARIANCE or COVAR
- COVAR_SAMP or COVARIANCE_SAMP
- CUME_DIST
- GROUPING
- LISTAGG
- MAX
- MEDIAN
- MIN
- PERCENTILE_CONT
- PERCENTILE_DISC
- PERCENT_RANK
- Regression functions (REGR_*)
- STDDEV_POP or STDDEV
- STDDEV_SAMP
- SUM
- VAR_POP or VARIANCE or VAR
- VAR_SAMP or VARIANCE_SAMP
- XMLAGG

Underscored functions are IDAA only

Db2 12 New Built-In Aggregate Functions

MEDIAN

```
SELECT MAX(COLCOUNT) AS MAX,
       MIN(COLCOUNT) AS MIN,
       AVG(COLCOUNT) AS AVG,
       MEDIAN(COLCOUNT) AS MEDIAN
FROM "SYSIBM".SYSTABLES;
```

MAX	MIN	AVG	MEDIAN
750	0	8	3

PERCENTILE_CONT and PERCENTILE_DISC

(PERCENTILE_DISC is always a value that appeared in the input set.)

```
SELECT MEDIAN(SALARY) AS MEDIAN,
       PERCENTILE_CONT(.50) WITHIN GROUP
        (ORDER BY SALARY) AS PCT_CONT,
       PERCENTILE_DISC(.50) WITHIN GROUP
        (ORDER BY SALARY) AS PCT_DISC
FROM EMP ;
```

MEDIAN	PCT_CONT	PCT_DISC
49,545	49,545	49,250

Associative Array Example (V12)

```
-- Manipulate an Associative Array
CREATE OR REPLACE PROCEDURE MY_PROC (OUT SHOVELS ASSOC_ARRAY_VC)
 BEGIN
   -- Build up Array
   SELECT ARRAY_AGG(NAME, PRICE) INTO SHOVELS FROM PRODUCT;

   -- Remove Ice Scraper
   SET SHOVELS = ARRAY_DELETE(SHOVELS,'Ice Scraper, Windshield 4 inch');

END#
```

PRODUCT TABLE

NAME	PRICE
Snow Shovel, Basic 22 inch	9.99
Snow Shovel, Deluxe 24 inch	19.99
Snow Shovel, Super Deluxe 26 inch	49.99
Ice Scraper, Windshield 4 inch	3.99

INITIAL SHOVELS ARRAY

NAME	PRICE
Snow Shovel, Basic 22 inch	9.99
Snow Shovel, Deluxe 24 inch	19.99
Snow Shovel, Super Deluxe 26 inch	49.99
Ice Scraper, Windshield 4 inch	3.99

FINAL SHOVELS ARRAY

NAME	PRICE
Snow Shovel, Basic 22 inch	9.99
Snow Shovel, Deluxe 24 inch	19.99
Snow Shovel, Super Deluxe 26 inch	49.99

Treating an Array Like a Table

Treat an array like a table i.e. fetch data from the array (using UNNEST specification for a collection-derived-table) just like you would from a table

from-clause

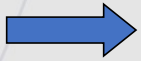
collection-derived-table

A *collection-derived-table* can be used to convert the elements of one or more arrays into column values in separate rows of an intermediate result table

Associative Array Example
SELECT T.State, T.Population **FROM UNNEST(myAssocIntArray) AS T(State,Population);**

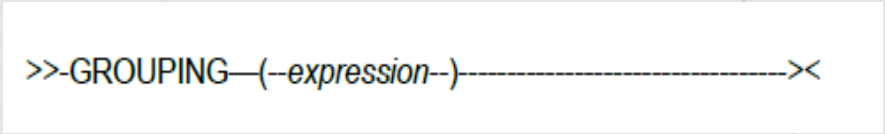
Deconstruct an Array

myAssocIntArray	
Index	Value
California	**38965193**
Nevada	**1978379**



State	Population
California	**38965193**
Nevada	**1978379**

Grouping Aggregate Function



Used to indicate whether or not the row was returned as a result of a grouping set

- i.e. the GROUPING aggregate function tells you the result of the grouping set was NULL versus the source row of the grouping set was NULL
- 1 - The NULL value was generated by a super-group
- 0 - The NULL value is from the source row

```
SELECT SALES_DATE, SALES_PERSON,
   SUM(SALES) AS UNITS_SOLD,
   GROUPING(SALES_DATE) AS DG,
   GROUPING(SALES_PERSON) AS SG
FROM SALES
GROUP BY CUBE (SALES_DATE, SALES_PERSON)
ORDER BY SALES_DATE, SALES_PERSON;
```

SALES_DATE	SALES_PERSON	UNITS_SOLD	DG	SG
12/31/1995	GOUNOT	1	0	0
12/31/1995	LEE	6	0	0
12/31/1995	LUCCHESSI	1	0	0
12/31/1995	-	8	0	1
03/29/1996	GOUNOT	11	0	0
03/29/1996	LEE	12	0	0
03/29/1996	LUCCHESSI	4	0	0
03/29/1996	-	27	0	1
03/30/1996	GOUNOT	21	0	0
03/30/1996	LEE	21	0	0
03/30/1996	LUCCHESSI	4	0	0
03/30/1996	-	46	0	1
03/31/1996	GOUNOT	3	0	0
03/31/1996	LEE	27	0	0
03/31/1996	LUCCHESSI	1	0	0
03/31/1996	-	31	0	1
04/01/1996	GOUNOT	14	0	0
04/01/1996	LEE	25	0	0
04/01/1996	LUCCHESSI	4	0	0
04/01/1996	-	43	0	1
-	GOUNOT	50	1	0
-	LEE	91	1	0
-	LUCCHESSI	14	1	0
-	-	155	1	1

LISTAGG APPLCOMPAT V12R1M500

LISTAGG results can be (re)ordered with an ORDER BY clause

APPLCOMPAT V13R1M504


```
SELECT workdept, LISTAGG(DISTINCT RTRIM(job),';  ')
WITHIN GROUP(ORDER BY RTRIM(job)) AS POSITION
FROM emp
GROUP BY workdept;
```

WORKDEPT	POSITION
A00	CLERK, PRES, SALESREP
B01	MANAGER
C01	ANALYST, MANAGER
D11	DESIGNER, MANAGER
D21	CLERK, MANAGER
E01	MANAGER
E11	MANAGER, OPERATOR
E21	FIELDREP, MANAGER

```
SELECT workdept, LISTAGG(DISTINCT RTRIM(job),';  ')
WITHIN GROUP(ORDER BY RTRIM(job)) AS POSITION
FROM emp
GROUP BY workdept
ORDER BY workdept DESC;
```

WORKDEPT	POSITION
E21	FIELDREP, MANAGER
E11	MANAGER, OPERATOR
E01	MANAGER
D21	CLERK, MANAGER
D11	DESIGNER, MANAGER
C01	ANALYST, MANAGER
B01	MANAGER
A00	CLERK, PRES, SALESREP

Scalar Functions

Scalar Functions

- Literally 100s of Scalar Functions that can be grouped into Categories
  - DATE/TIME/TIMESTAMP
  - Character Manipulation
    - BIT manipulation/HASH
    - String Manipulation
    - Character Based Functions
  - Trig and Math Related Functions
    - DECFLOAT
  - ENCRYPT/DECRYPT
  - CAST
  - ARRAY Manipulation/Traversal
- It is impossible to cover these all - Let’s explore some interesting Functions

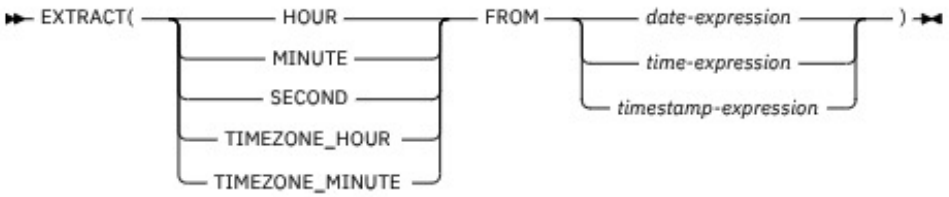
EXTRACT

EXTRACT - returns a portion of a date or timestamp based on its arguments.

Extract date values:

Extract time values:

Extract time zone values:




```
SELECT
  EXTRACT(YEAR FROM CURRENT DATE) AS YEAR,
  EXTRACT( SECOND FROM CURRENT TIMESTAMP(12)) AS SECOND,
  EXTRACT(TIMEZONE_HOUR FROM CURRENT TIMESTAMP WITH TIME ZONE )
    AS TZH
FROM SYSIBM.SYSDUMMYU;
```

YEAR	SECOND	TZH
2,024	15.2996935	-4

Integer Decimal Integer

TIMESTAMP_FORMAT

TIMESTAMP_FORMAT gives the user extreme flexibility to allow Db2 to support virtually any format of DATE/TIME/TIMESTAMP String

```
SELECT TIMESTAMP_FORMAT('Jan 20,2002 8:08 AM','MONTH DD,YYYY HH:MI AM' ) AS
 TIMESTAMP FROM "SYSIBM".SYSDUMMYU;

  TIMESTAMP
  --------------------------
  2002-01-20 08:08:00.000
```

23 Format String options

```
SELECT TIMESTAMP_FORMAT('143,2004 8.47.30','DDD,YYYY HH24.MI.SS' ) AS
TIMESTAMP FROM "SYSIBM".SYSDUMMYU;

  TIMESTAMP
  --------------------------
  2004-05-22 08:47:30.000
```

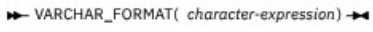
VARCHAR_FORMAT

SELECT VARCHAR_FORMAT('aBc') AS STR FROM "SYSIBM".SYSDUMMYU ;

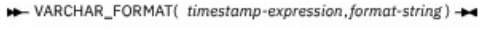
```
STR
---
aBc
```

The value of the result is the same as the value of character-expression

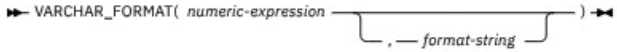
Character to VARCHAR



Timestamp to VARCHAR



Numeric to VARCHAR



SELECT VARCHAR_FORMAT(1234.56,'99999.999') AS NUM FROM "SYSIBM".SYSDUMMYU;

```
NUM
--------
1234.560
```

SELECT VARCHAR_FORMAT(CURRENT TIMESTAMP,'MONTH DD,YYYY HH:MI:SS AM') AS TS_STRING FROM "SYSIBM".SYSDUMMYU;

```
TS_STRING
------------------------
APRIL 03,2024 05:17:22 PM
```

ROUND_TIMESTAMP & TRUNC_TIMESTAMP

- SELECT TRUNC_TIMESTAMP(CURRENT TIMESTAMP, 'CC') AS CENTURY FROM SYSIBM.SYSDUMMYU;

```
CENTURY
--------------------------
2001-01-01 00:00:00.000
```

- SELECT TRUNC_TIMESTAMP(CURRENT TIMESTAMP, 'YYYY') AS YEAR FROM SYSIBM.SYSDUMMYU;

```
YEAR
--------------------------
2024-1-01-01 00:00:00.000
```

- SELECT TRUNC_TIMESTAMP(CURRENT TIMESTAMP, 'Q') AS QUARTER FROM SYSIBM.SYSDUMMYU;

```
QUARTER
--------------------------
2024-04-01 00:00:00.000
```

- SELECT TRUNC_TIMESTAMP(CURRENT TIMESTAMP, 'DD') AS T_DAY,
  ROUND_TIMESTAMP(CURRNT TIMESTAMP,'DD) AS R_DAY FROM SYSIBM.SYSDUMMYU;

```
T_DAY                      R_DAY
-------------------------- --------------------------
2024-04-04 00:00:00.000 2024-04-05 00:00:00.000
```

Days, Weeks, Months, Quarters, and Years…

Days

- DAY
  - If the argument is a date, timestamp, the result is between 1 and 31
  - If the argument is a date duration or timestamp duration, the result is the **result is the day part** of the value which is between -99 and 99*
  - If the argument contains a time zone, the result is the day part of the value expressed in UTC.
- DAYOFMONTH
- DAYOFWEEK
- DAYOFWEEK_ISO
- DAYOFYEAR
- DAYS
  - The result is 1 more than the number of days from January 1, 0001 to the Date
- DAYS_BETWEEN
- NEXT_DAY
  - The result is a TIMESTAMP or DATE of the following day that is specified as the second argument

Weeks

- WEEK
- WEEK_ISO

Months

- MONTH
- MONTHS_BETWEEN
- NEXT_MONTH

Quarters

- QUARTER

Years

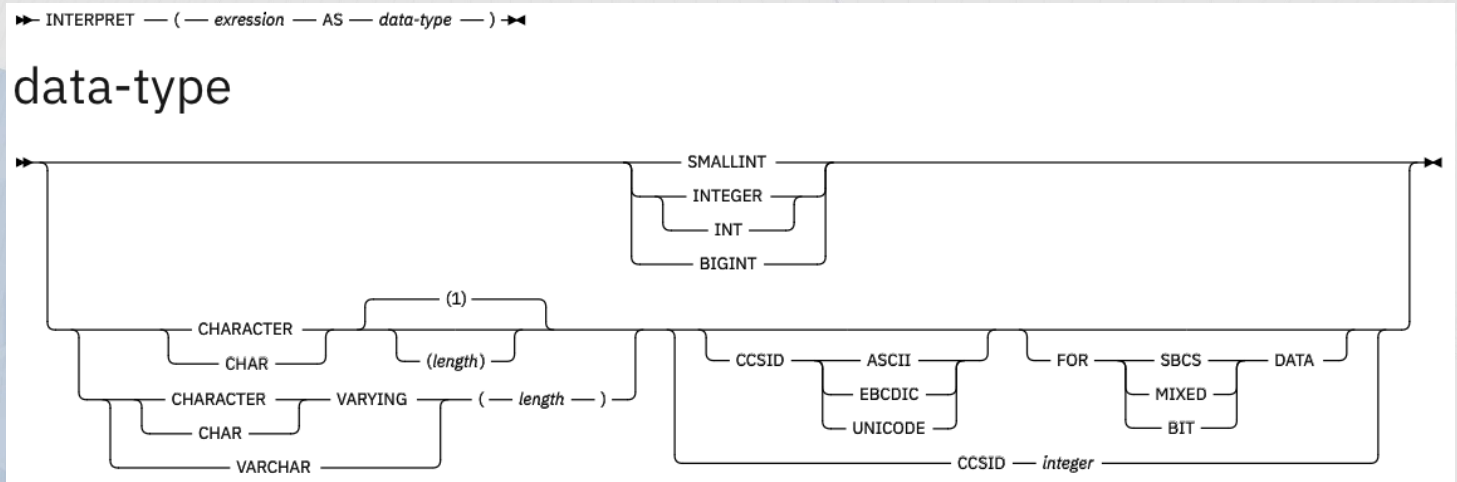
- YEAR

- MIDNIGHT_SECONDS
  - The result is an Integer in the range 0 - 86400, that represents the number of seconds between midnight and the time that is specified in the argument.

Bits and Bytes

- GENERATE_UNIQUE_BINARY
  - 16 Byte STCKE built to Scale Beyond 256 Processors
  - Can get a Timestamp from the value
- PACK -
  - Creates a VARBINARY string from an input expression – Consumed by UNPACK function
- VARCHAR_BIT_FORMAT
  - Takes a string representation of a hex value and returns a bit string representation of that value
- HASH Functions
  - HASH_CRC32, HASH_MD5, HASH_SHA1, HASH_SHA256
- BIT FUNCTIONS
  - BITAND, BITANDNOT, BITOR, BITXOR, BITNOT
- BTRIM
  - TRIMs a string using a binary representation of a character (default trim str is “blank”)

INTERPRET (APPLCOMPAT V13R1M505)



The schema is SYSIBM.

Function invocation	Result value
INTERPRET(BX'00000011' AS INTEGER)	17
INTERPRET(BX'0000000000B0370D' AS BIGINT)	11548429
INTERPRET(BX'616263' AS CHAR(3) CCSID 37)	/ÃÄ
INTERPRET(BX'616263' AS CHAR(3) CCSID 1208)	abc
INTERPRET(BX'0005C1C2C3C4C5' AS VARCHAR(5))	ABCDE
INTERPRET(BX'0003C1C2C3C4C5' AS VARCHAR(5))	ABC
INTERPRET(BX'0007C1C2C3C4C5' AS VARCHAR(7))	Error

String Manipulation

- CONCAT
- LOCATE, LOCATE_IN_STRING, STRPOS
- SOUNDEX
- TRIM, STRIP, LTRIM, RTRIM, LEFT(STLEFT), RIGHT(STRIGHT)
  - BTRIM
- EXTRACT
- LCASE(LOWER)/UCASE(UPPER)
- LPAD/RPAD/SPAE
- MAX/MIN
- REPEAT
- REPLACE
- REGEXP_COUNT, REGEXP_INSTR, REGEXP_LIKE, REGEXP_REPLACE, REGEXP_SUBSTR
- TRANSLATE

Casting Between Datatypes

Cast from data type – / To data type[1]	SMALLINT	INTEGER	BIGINT	DECIMAL	DECFLOAT	REAL	DOUBLE	CHAR	VARCHAR	CLOB	GRAPHIC	VARGRAPHIC	DBCLOB	BINARY	VARBINARY	BLOB	DATE	TIME	TIMESTAMP WITHOUT TIME ZONE	TIMESTAMP WITH TIME ZONE	ROWID	XML
SMALLINT	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y[4]	Y[4]										
INTEGER	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y[4]	Y[4]										
BIGINT	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y[4]	Y[4]										
DECIMAL	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y[4]	Y[4]										
DECFLOAT	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y[4]	Y[4]										
REAL	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y[4]	Y[4]										
DOUBLE	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y[4]	Y[4]										
CHAR	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	
VARCHAR	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	
CLOB								Y	Y	Y	Y	Y	Y	Y	Y	Y						
GRAPHIC	Y	Y	Y	Y	Y	Y	Y	Y[2]	Y[2]	Y[2]	Y	Y	Y	Y	Y	Y	Y[3]	Y[3]	Y[3]	Y[3]		
VARGRAPHIC	Y	Y	Y	Y	Y	Y	Y	Y[2]	Y[2]	Y[2]	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y[3]		
DBCLOB								Y[2]	Y[2]	Y[2]	Y	Y	Y	Y	Y	Y						
BINARY														Y	Y	Y						
VARBINARY														Y	Y	Y						
BLOB														Y	Y	Y						
DATE								Y	Y								Y		Y			
TIME								Y	Y									Y				
TIMESTAMP WITHOUT TIME ZONE								Y	Y								Y	Y	Y	Y		
TIMESTAMP WITH TIME ZONE								Y	Y								Y	Y	Y	Y		
ROWID								Y	Y					Y	Y	Y					Y	
XML																						Y

CAST Specification or Cast Function?

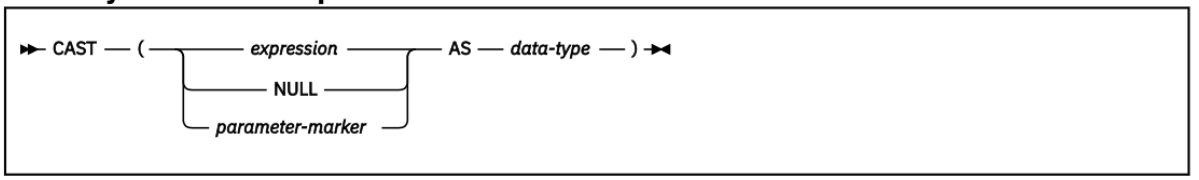
In general, a CAST Function or CASTing using the CAST Specification are EQUAL

- CHAR9 and VARCHAR9 (non-Standard behavior)
- Wordy Syntax

CAST specification

The CAST specification returns the first operand (the cast operand) converted to the data type that is specified by *data-type*.

Syntax for CAST specification



data-type:

Character Based Functions

- CHARACTER_LENGTH
- CHAR
- CLOB
- DBCLOB
- GRAPHIC
- INSERT
- INSTR
- LEFT
- LOCATE
- LOCATE_IN_STRING
- OVERLAY
- POSITION
- REGXP_* (IDAA Passthru)
- RIGHT
- STRLEFT
- STRRIGHT
- SUBSTRING
- VARCHAR
- VARGRAPIC
- CAST Specification (Code Units can be specified)

Character Based Function Examples

Function	Returns	
SUBSTR('Hegelstraße',1,10)	'Hegelstra?'	– x'486567656C73747261E1'
SUBSTRING('Hegelstraße',1,10,OCTETS)	'Hegelstra '	– x'486567656C7374726120'
SUBSTRING('Hegelstraße',1,11,OCTETS)	'Hegelstra '	– x'486567656C737472612020'
SUBSTRING('Hegelstraße',1,11,CODEUNITS16)	'Hegelstraße'	– x'486567656C73747261E1BA9E65'
SUBSTRING('Hegelstraße',1,11,CODEUNITS32)	'Hegelstraße'	– x'486567656C73747261E1BA9E65'

Assume that T1 is a Unicode table with column **C1 VARCHAR(10)** with one row with the value of the mathematical bold capital **𝐀** (X'F09D9080') – it looks like a bold "**A**", but it's not.
The following similar queries return different answers:

```
SELECT CHARACTER_LENGTH(C1,CODEUNITS32) FROM T1;      -- Returns 1
SELECT CHARACTER_LENGTH(C1,CODEUNITS16) FROM T1;      -- Returns 2
SELECT CHARACTER_LENGTH(C1,OCTETS) FROM T1;           -- Returns 4
```

The following similar queries return different answers:

```
SELECT HEX(SUBSTRING(C1,1,1,CODEUNITS32)) FROM T1;    -- Returns X'F09D9080'
SELECT HEX(SUBSTRING(C1,1,1,CODEUNITS16)) FROM T1;    -- Returns X'20'
SELECT HEX(SUBSTRING(C1,1,2,CODEUNITS16)) FROM T1;    -- Returns X'F09D9080'
SELECT HEX(SUBSTRING(C1,1,1,OCTETS)) FROM T1;         -- Returns X'20' (blank)
SELECT HEX(SUBSTR(C1,1,1)) FROM T1;                   -- Returns X'F0'
```

ASCII_STR and EBCDIC_STR

Enable Unicode data to be returned without substitution in ASCII or EBCDIC. Data not directly convertible is “escaped”

Assuming T1.C1 contains Unicode string – "Hi, my name is Андрей”

```
SELECT C1 FROM T1;
 Returns (to 3270 CCSID 37 screen)
       ‘Hi, my name is ......’
SELECT ASCII_STR(C1)FROM T1;  -- Note EBCDIC_STR would return the same result
 Returns
       ‘Hi, my name is \0410\043D\0434\0440\0435\0439’  -- Escaped UTF-16
```

UNISTR or UNICODE_STR

Accepts “escaped” Unicode input and converts it to UTF-8 or UTF-16.

Assuming T1.C1 contains “Андрей”

```
SELECT ASCII_STR(C1)FROM SYSIBM.SYSDUMMYE;
 Returns '\0410\043D\0434\0440\0435\0439' – Escaped UTF-16


SELECT HEX(UNISTR(ASCII_STR(C1))) FROM SYSIBM.SYSDUMMYE;
 Returns 'D090D0BDD0B4D180D0B5D0B9' – HEX of UTF-8

SELECT HEX(UNISTR(ASCII_STR(C1),UTF16)) FROM SYSIBM.SYSDUMMYE;
 Returns '0410043D0434044004350439' – HEX of UTF-16
```

COLLATION_KEY

```
CREATE TABLE T1 (C1 VARCHAR(6) ) CCSID UNICODE;
INSERT INTO T1 VALUES('Cat’);
INSERT INTO T1 VALUES('cat’);
```

```
SELECT C1 FROM T1
  ORDER BY C1 ;
```

Cat
cat

```
SELECT C1 FROM T1
  ORDER BY
COLLATION_KEY(C1,'UCA410_LEL_CL’);
```

cat
Cat

COLLATION_KEY with Locales

```
CREATE TABLE T1 (C1 VARCHAR(6) ) CCSID UNICODE;
INSERT INTO T1 VALUES('cote',1);
INSERT INTO T1 VALUES('côté',2);
INSERT INTO T1 VALUES('côte',3);
INSERT INTO T1 VALUES('coté',4);
```

```
SELECT C1 FROM T1
  ORDER BY C1;
```

cote
coté
côte
côté

```
SELECT C1 FROM T1
  ORDER BY COLLATION_KEY(C1,'UCA410_LFR_FO’);
```

cote
côte
coté
côté

Index on Expression

EXPLAIN ALL SET QUERYNO = **110** FOR SELECT C1 FROM T1 ORDER BY COLLATION_KEY(C1,'UCA410_LFR_FO');

CREATE INDEX I1 ON T1 (COLLATION_KEY(C1,'UCA410_LFR_FO'));

EXPLAIN ALL SET QUERYNO = **210** FOR SELECT C1 FROM T1 ORDER BY COLLATION_KEY(C1,'UCA410_LFR_FO');

SELECT * FROM PLAN_TABLE;

	QUERYNO	QBLOCKNO	PROGNAME	PLANNO	METHOD	CREATOR	TNAME	TABNO	ACCESSTYPE
1	110	1	DSNTEP2	1	0	ADMF001	T1	1	R
2	110	1	DSNTEP2	2	3			0	
3	210	1	DSNTEP2	1	0	ADMF001	T1	1	I

	MATCHCOLS	ACCESSNAME	INDEXONLY	SORTN_UNIQ	SORTN_JOIN	SORTN_ORDERBY
1	0		N	N	N	N
2	0		N	N	N	N
3	0	I1	N	N	N	N

	SORTN_GROUPBY	SORTC_UNIQ	SORTC_JOIN	SORTC_ORDERBY	SORTC_GROUPBY	PREFETCH
1	N	N	N	N	N	S
2	N	N	N	Y	N	
3	N	N	N	N	N	

UPPER/LOWER with Locale

```
CREATE TABLE T1 (C1 VARCHAR(15)) VOLATILE CCSID UNICODE;
INSERT INTO T1 VALUES('Hegelstraße');
```

```
SELECT C1 FROM T1;
```

C1
Hegelstraße

```
SELECT UPPER(C1)AS C1 FROM T1;
```

C1
HEGELSTRAßE

```
SELECT UPPER(C1,'UNI') AS C1 FROM T1 ;
```

C1
HEGELSTRASSE

NORMALIZE_STRING

The NORMALIZE_STRING function can convert two strings that look the same (such as Å, which can be encoded in UTF-16 as X'00C5' and as X'00410307') but might not be encoded using the same Unicode code point, to a normalized form that can be compared

- ```
  SELECT NORMALIZE_STRING('åbc',NFD) AS C1,
     HEX(NORMALIZE_STRING('åbc',NFD)) AS HEX_NFD_C1
  FROM SYSIBM.SYSDUMMYU;
  ```

C1	HEX_NFD_C1
åbc	61CC8A6263

- ```
  SELECT NORMALIZE_STRING('åbc',NFC) AS C1,
     HEX(NORMALIZE_STRING('åbc',NFC)) AS HEX_NFC_C1
  FROM SYSIBM.SYSDUMMYU;
  ```

C1	HEX_NFC_C1
åbc	C3A56263

SOUNDEX & DIFFERENCE Phonetic Algorithms

- SOUNDEX - The SOUNDEX function returns a 4 character code that represents the sound of the words in the argument.
- DIFFERENCE - The DIFFERENCE function returns a value from 0 to 4 that represents the difference between the sounds of two strings based on applying the SOUNDEX function to the strings. A value of 4 is the best possible sound match.

```
SELECT EMPNO, LASTNAME, SOUNDEX(LASTNAME) AS SOUNDEX_LN,
SOUNDEX('Loucesy') AS SOUNDEX_STR, DIFFERENCE(LASTNAME,'Loucesy')
AS DIFFERENCE FROM DSN81210.EMP
WHERE SOUNDEX(LASTNAME) = SOUNDEX('Loucesy');
```

EMPNO	LASTNAME	SOUNDEX_LN	SOUNDEX_STR	DIFFERENCE
110	LUCCHESI	L220	L220	4

DECFLOAT

- DECFLOAT – Cast Function
- COMPARE_DECFLOAT
- DECFLOAT_SORTKEY
- NORMALIZE_DECFLOAT
- QUANTIZE
- TOTLORDER
- DECFLOAT_FORMAT

COMPARE_DECFLOAT

- Returns a smallint that indicates how the two values compare - One of the following values will be the result:
  - 0 The arguments are exactly equal
  - 1 decfloat-expression1 is less than decfloat-expression2
  - 2 decfloat-expression1 is greater than decfloat-expression2
  - 3 The arguments are unordered

```
COMPARE_DECFLOAT(DECFLOAT(2.17), DECFLOAT(2.17))     = 0
COMPARE_DECFLOAT(DECFLOAT(2.17), DECFLOAT(2.170))    = 2
COMPARE_DECFLOAT(DECFLOAT(2.170), DECFLOAT(2.17))    = 1
COMPARE_DECFLOAT(DECFLOAT(2.17), DECFLOAT(0.0))      = 2
COMPARE_DECFLOAT(INFINITY,INFINITY)                  = 0
COMPARE_DECFLOAT(INFINITY,-INFINITY)                 = 2
COMPARE_DECFLOAT(DECFLOAT(-2),INFINITY)              = 1
COMPARE_DECFLOAT(NAN,NAN)                            = 3
COMPARE_DECFLOAT(DECFLOAT(-0.1),SNAN)                = 3
```

DECFLOAT_SORTKEY

- Sort DECFLOAT data “Correctly”

```
CREATE TABLE T1(D1 DECFLOAT(16));
INSERT INTO T1 VALUES (2.100);
INSERT INTO T1 VALUES (2.1);
INSERT INTO T1 VALUES (2.1000);
INSERT INTO T1 VALUES (2.10);
```

```
SELECT D1 FROM T1
ORDER BY D1;

D1
------
2.10
2.1000
2.1
2.100
```

```
SELECT D1 FROM T1 ORDER BY
DECFLOAT_SORTKEY(D1);

D1
------
2.1000
2.100
2.10
2.1
```

NORMALIZE_DECFLOAT

- Normalizes a DECFLOAT ARGUMENT to it's Simplest Form

```
NORMALIZE_DECFLOAT(DECFLOAT(2.1))        = 2.1
NORMALIZE_DECFLOAT(DECFLOAT(-2.0))       = -2
NORMALIZE_DECFLOAT(DECFLOAT(1.200))      = 1.2
NORMALIZE_DECFLOAT(DECFLOAT(1.2000))     = 1.2
NORMALIZE_DECFLOAT(DECFLOAT(-120))       = -1.2E+2
NORMALIZE_DECFLOAT(DECFLOAT(120.00))     = 1.2E+2
NORMALIZE_DECFLOAT(DECFLOAT(0.00))       = 0
NORMALIZE_DECFLOAT(-NAN)                 = -NAN
```

QUANTIZE

- Formats DECFLOAT Using a Pattern

```
QUANTIZE(2.17, DECFLOAT(0.001))          = 2.170
QUANTIZE(2.17, DECFLOAT(0.01))           = 2.17
QUANTIZE(2.17, DECFLOAT(0.1))            = 2.2
QUANTIZE(2.17, DECFLOAT('1E+0'))         = 2
QUANTIZE(2.17, DECFLOAT('1E+1'))         = 0E+1
QUANTIZE(-0.1, DECFLOAT(1) )             = 0
QUANTIZE(0, DECFLOAT('1E+5'))            = 0E+5
```

TOTALORDER

Compares DECFLOAT data using the following IEEE 754R base order

- -1 if decfloat-expression1 is lower in order compared to decfloat-expression2
- 0 if both decfloat-expression1 and decfloat-expression2 have the same order
- 1 if decfloat-expression1 is higher in order compared to decfloat-expression2

-NAN<-SNAN<-INFINITY<-0.10<-0.100<-0<0<0.100<0.10<INFINITY<SNAN<NAN

```
TOTALORDER(-INFINITY, -INFINITY)                = 0
TOTALORDER(DECFLOAT(-1.0), DECFLOAT(-1.0))      = 0
TOTALORDER(DECFLOAT(-1.0), DECFLOAT(-1.00))     = -1
TOTALORDER(DECFLOAT(-1.0), DECFLOAT(-0.5))      = -1
TOTALORDER(DECFLOAT(-1.0), INFINITY)            = -1
TOTALORDER(DECFLOAT(-1.0), NAN)                 = -1
TOTALORDER(NAN, DECFLOAT(-1.0))                 = 1
TOTALORDER(-SNAN, -SNAN)                        = 0
TOTALORDER(NAN, NAN)                            = 0
```

DECFLOAT_FORMAT

Example	Result
DECFLOAT_FORMAT('123.45’)	123.45
DECFLOAT_FORMAT('-123456.78’)	-123456.78
DECFLOAT_FORMAT('+123456.78’)	123456.78
DECFLOAT_FORMAT('1.23E4’)	12300
DECFLOAT_FORMAT('123.4', '9999.99’)	123.40
DECFLOAT_FORMAT('001,234', '000,000')	1234
DECFLOAT_FORMAT('1234 ', '9999MI’)	1234
DECFLOAT_FORMAT('1234-', '9999MI’)	-1234
DECFLOAT_FORMAT('+1234', 'S9999’)	1234
DECFLOAT_FORMAT('-1234', 'S9999’)	-1234
DECFLOAT_FORMAT(' 1234 ', '9999PR’)	1234
DECFLOAT_FORMAT('<1234>', '9999PR’)	-1234
DECFLOAT_FORMAT('$1,234.56', '$9,999.99')	1234.56

ENCRYPT_DATAKEY, DECRYPT_DATAKEY_xxx


```
CREATE TABLE T1
  (NAME CHAR(20), PHONE_NUMBER VARBINARY(95));

INSERT INTO T1 VALUES
  ('Chris',ENCRYPT_DATAKEY('408-867-5309','MYKEYLABEL',AES256D));

SELECT
  NAME, DECRYPT_DATAKEY_VARCHAR(PHONE_NUMBER)
FROM SAMPLE_TABLE;
```

News from the Lab Blog on ENCRYPT_DATAKEY - https://ibm.co/39M55UW

IDENTITY_VAL_LOCAL

CREATE TABLE TID (CHAR_COL CHAR(5), IDENT_COL INTEGER GENERATED ALWAYS AS IDENTITY);

INSERT INTO TID (CHAR_COL) VALUES ('ROW1');

SELECT * FROM TID;

```
CHAR_COL    IDENT_COL
--------    ---------
ROW1                1
```

SELECT * FROM FINAL TABLE (INSERT INTO TID (CHAR_COL) VALUES (‘ROW2’));

```
CHAR_COL    IDENT_COL
--------    ---------
ROW2                2
```

SELECT * FROM TID;

```
CHAR_COL    IDENT_COL
--------    ---------
ROW1                1
ROW2                2
```

Table Functions

Table Functions

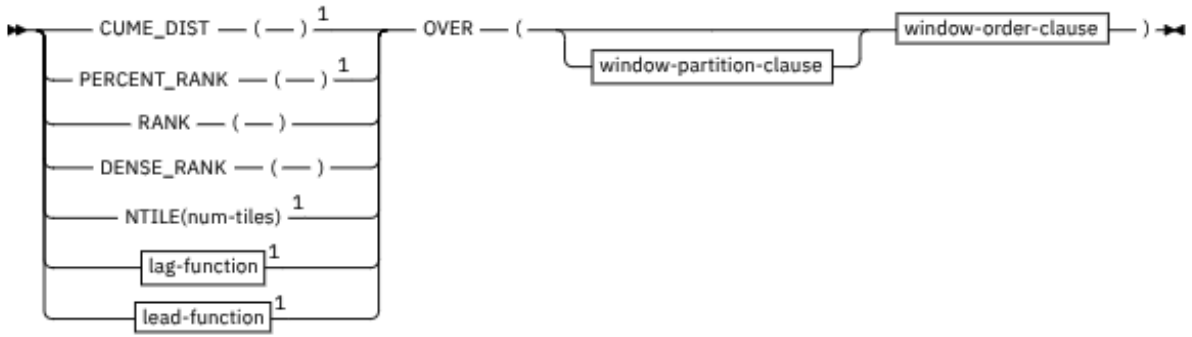
- ADMIN_TASK_* Functions
  - The ADMIN_TASK_LIST - Returns a table with one row for each of the tasks that are defined in the administrative task scheduler task list.
  - ADMIN_TASK_OUTPUT – Returns the output parameter values and result sets, if available.
  - ADMIN_TASK_STATUS - Returns a table with one row for each task that is defined in the administrative task scheduler task list.
- BLOCKING_THREADS
  - Returns a table that contains one row for each lock or claim that threads hold against the databases that are specified in the input parameter.
  - The same information is essentially provided via the –DISPLAY BLOCKERS Command
- MQ* Functions
  - MQREADALL/MQREADALLCLOB
    - Returns a table that contains the messages and message metadata from a specified IBM MQ location without removing the messages from the queue.
  - MQRECEIVEALL/MQRECEIVEALLCLOB
    - Returns a table that contains the messages and message metadata from a specified IBM MQ location and removes the messages from the queue.
- XMLTABLE
  - Returns a result table from the evaluation of XQuery expressions.

OLAP Specifications

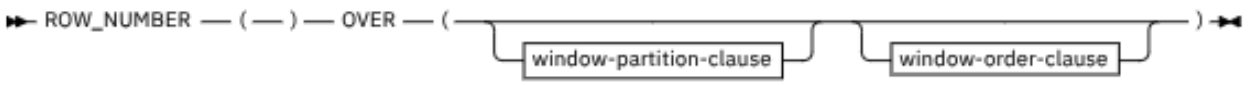
OLAP specifications

- CUME_DIST
- PERCENT_RANK
- RANK
- DENSE_RANK
- NTILE(num-tiles)
- LAG
- LEAD
- ROW_NUMBER

ordered-OLAP-specification



numbering-specification



RANK

```
SELECT EMPNO, LASTNAME, SALARY+BONUS AS TOTAL_SALARY,
       RANK() OVER(ORDER BY SALARY+BONUS DESC) AS RANK_SALARY
FROM DSN81210.EMP
WHERE SALARY+BONUS > 30000
ORDER BY RANK_SALARY
```

EMPNO	LASTNAME	TOTAL_SALARY	RANK_SALARY
000010	HAAS	53,750	1
200010	HEMMINGER	47,500	2
000110	LUCCHESI	47,400	3
000020	THOMPSON	42,050	4
000050	GEYER	40,975	5
000030	KWAN	39,050	6
000070	PULASKI	36,870	7
000060	STERN	32,850	8
000220	LUTZ	30,440	9
200220	JOHN	30,440	9
000090	HENDERSON	30,350	11

DENSE_RANK

```
SELECT EMPNO, LASTNAME, SALARY+BONUS AS TOTAL_SALARY,
  DENSE_RANK() OVER(ORDER BY SALARY+BONUS DESC) AS RANK_SALARY
FROM DSN81210.EMP
WHERE SALARY+BONUS > 30000
ORDER BY RANK_SALARY
```

EMPNO	LASTNAME	TOTAL_SALARY	RANK_SALARY
000010	HAAS	53,750	1
200010	HEMMINGER	47,500	2
000110	LUCCHESI	47,400	3
000020	THOMPSON	42,050	4
000050	GEYER	40,975	5
000030	KWAN	39,050	6
000070	PULASKI	36,870	7
000060	STERN	32,850	8
000220	LUTZ	30,440	9
200220	JOHN	30,440	9
000090	HENDERSON	30,350	10

ROW_NUMBER

```
SELECT EMPNO, LASTNAME, SALARY+BONUS AS TOTAL_SALARY,
  ROW_NUMBER() OVER(ORDER BY SALARY+BONUS DESC) AS ROW_NUMBER
FROM DSN81210.EMP
WHERE SALARY+BONUS > 30000
ORDER BY ROW_NUMBER
```

EMPNO	LASTNAME	TOTAL_SALARY	ROW_NUMBER
000010	HAAS	53,750	1
200010	HEMMINGER	47,500	2
000110	LUCCHESI	47,400	3
000020	THOMPSON	42,050	4
000050	GEYER	40,975	5
000030	KWAN	39,050	6
000070	PULASKI	36,870	7
000060	STERN	32,850	8
200220	JOHN	30,440	9
000220	LUTZ	30,440	10
000090	HENDERSON	30,350	11

SQL Scalar Functions

User Defined Functions or UDFs

- External Scalar
  - Written in a host language such as COBOL, C, or PL/I, Java, HLASM
- External Table
  - A Table Function written in a host Language such as COBOL, C, PL/I, HLASM
- Sourced
  - A function that is implemented by an existing Scalar or Aggregate Function
  - Often used with Distinct Types
- SQL Scalar
  - A function that is implemented in SQL PL and is either inlined or compiled
    - Inlined SQL scalar functions contain a SINGLE RETURN statement which returns the value of a simple expression
    - Compiled SQL scalar functions support full SQL PL syntax and can contain more than one RETURN statement, in additional to other complex logic such as loops or IF THEN ELSE type syntax

In-lined SQL Function Example

```
CREATE FUNCTION TAN (X DOUBLE) RETURNS DOUBLE
  LANGUAGE SQL
  CONTAINS SQL
  NO EXTERNAL ACTION DETERMINISTIC
RETURN SIN(X)/COS(X);
```

In-lined SQL Function to fix Oracle SQL Syntax

```
SELECT CURRENT TIMESTAMP + 1 DAY AS CURRENT_TIMESTAMP FROM SYSIBM.SYSDUMMY1;
```

CURRENT_TIMESTAMP
2024-04-03 16:57:03.981

Oracle does not require ”DAY”

```
SELECT CURRENT TIMESTAMP + 1    AS CURRENT_TIMESTAMP FROM SYSIBM.SYSDUMMY1;
```

SQL Error [42815]: THE DATA TYPE, LENGTH, OR VALUE OF ARGUMENT 2 OF + IS INVALID.
SQLCODE=-171, SQLSTATE=42815, DRIVER=4.32.28

```
CREATE FUNCTION ADD_TS (TS TIMESTAMP, NUM_DAYS INTEGER) RETURNS TIMESTAMP LANGUAGE SQL
CONTAINS SQL NO EXTERNAL ACTION DETERMINISTIC RETURN TS + NUM_DAYS DAYS;

CREATE FUNCTION "+" (TIMESTAMP,INTEGER) RETURNS TIMESTAMP SOURCE ADD_TS(TIMESTAMP,INTEGER);

SELECT CURRENT TIMESTAMP + 1    AS CURRENT_TIMESTAMP FROM SYSIBM.SYSDUMMY1;
```

CURRENT_TIMESTAMP
2024-04-03 16:57:03.981

Using a Function to deal with variable data issues

```
CREATE TABLE TD1 (C1 VARCHAR(40));
INSERT INTO TD1 VALUES (SPACE(5) || CHAR(CURRENT TIMESTAMP));
INSERT INTO TD1 VALUES (SPACE(4) || CHAR(CURRENT TIMESTAMP));
INSERT INTO TD1 VALUES (CHAR(CURRENT TIMESTAMP) || SPACE(5));
SELECT C1 AS TS_DATA FROM TD1;
```

TS_DATA
2024-04-02-18.54.01.894882
2024-04-02-18.54.02.702539
2024-04-02-18.54.03.530089

```
SELECT TIMESTAMP(C1) AS TS_DATA FROM TD1;
```

SQL Error [22007]: THE DATE, TIME, OR TIMESTAMP VALUE *N IS INVALID. SQLCODE=-180, SQLSTATE=22007, DRIVER=4.32.28

Using a Function to deal with variable data issues (cont)

```
SELECT STRIP(C1,BOTH) AS TS_DATA FROM TD1;
```

TS_DATA
2024-04-02-18.54.01.894882
2024-04-02-18.54.02.702539
2024-04-02-18.54.03.530089

```
SET CURRENT SCHEMA = 'TEST';

CREATE FUNCTION TIMESTAMP (VC VARCHAR(40)) RETURNS TIMESTAMP LANGUAGE SQL CONTAINS SQL
 NO EXTERNAL ACTION DETERMINISTIC  RETURN TIMESTAMP(STRIP(VC,BOTH));

SELECT TEST.TIMESTAMP(C1) AS TS_DATA FROM TD1;
```

TS_DATA
2024-04-02-18.54.01.894882
2024-04-02-18.54.02.702539
2024-04-02-18.54.03.530089

```
SELECT TIMESTAMP(C1) AS TS_DATA FROM TD1;
```

TS_DATA
2024-04-02-18.54.01.894882
2024-04-02-18.54.02.702539
2024-04-02-18.54.03.530089

SQL PL Functions

```
CREATE FUNCTION DYN_QUERY(SCH VARCHAR(128), TB VARCHAR(128))
  RETURNS BIGINT
DETERMINISTIC NO EXTERNAL ACTION PARAMETER CCSID UNICODE READS SQL DATA

BEGIN

 DECLARE CNT BIGINT;
 DECLARE STMT_STR VARCHAR(256);
 DECLARE S1 STATEMENT;
 DECLARE C1 CURSOR FOR S1;

 -- Set up statement that will query the table that we want to Count

 SET STMT_STR = 'SELECT COUNT(*) FROM' || SCH || '.' || TB;

 PREPARE S1 FROM STMT_STR;

 OPEN C1;
 FETCH C1 INTO CNT;
 CLOSE C1;

 RETURN CNT;

END
```

SQL Data Insights

SQL Data Insights (V13R1M500 and M504)

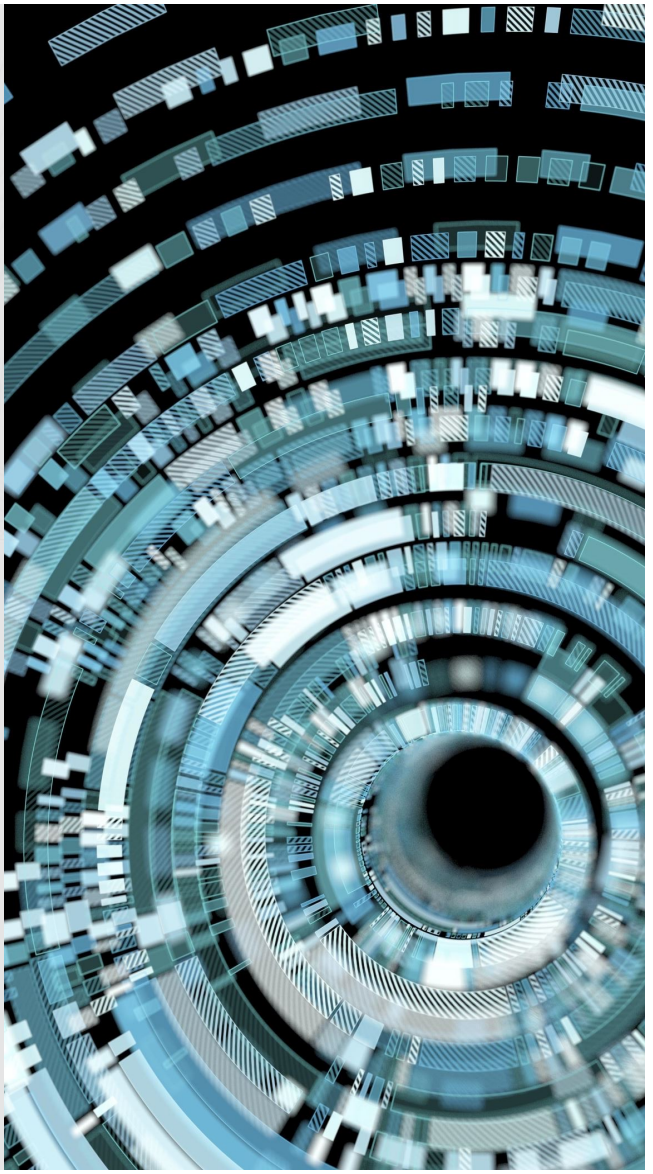
SELECT to find 10 most similar customers to customer “3668-QPYBK’:

```
SELECT AI_SIMILARITY(X.customerID,'3668-QPYBK') AS SimilarityScore, X.*
  FROM DSNAIDB.CHURN X
WHERE X.customerID <> '3668-QPYBK’
ORDER BY SimilarityScore DESC
FETCH FIRST 10 ROWS ONLY
```

SELECT to find the top five outliers of the model column CUSTOMERID:

```
SELECT AI_COMMONALITY(CUSTOMERID) AS SCORE, C.*
  FROM CHURN C ORDER BY SCORE ASC
  FETCH FIRST 5 ROWS ONLY;
```

Db2 Built-in AI Semantic Function	Semantic Query Type
AI_SIMILARITY	Similarity Query
AI_SIMILARITY	Dissimilarity Query
AI_SEMANTIC_CLUSTER	Inductive Reasoning Semantic Clustering Query
AI_ANALOGY	Inductive Reasoning Analogy Query
AI_COMMONALITY[M504]	Similarity Query based on expression value and Centroid Value



Conclusion

- Functions enhance SQL in a wide variety of ways
- Db2 supports numerous types of functions
  - Aggregate Functions
  - Scalar Functions
  - Table Functions
  - ROW – UNPACK (unpacks a row built by PACK function)
  - OLAP Specifications
  - User Defined Functions
    - Host Language
    - SQL Scalar
  - AI – SQL Data Insights
- All these extend application capability and aid productivity

Broadcom Mainframe Technical Exchanges

- ✓ European in-person event in Prague: April 16-18
- North American in-person event in Plano, TX: September 10-12 (registration open)
- Global virtual event: October 8-10 (save the date)

Make plans to attend

- Network with peers and Mainframe technical experts
- Technical education, product update, how-to and roundtable sessions
- No registration fee! Open to all Broadcom customers
- Learn more: https://bit.ly/MainframeTechEx


THANK YOU
Chris Crone
cjc@broadcom.com
BROADCOM