DB2 9 new datatypes and SQL functions: blessing or curse?

Peter Vanroose
ABIS Training & Consulting

Dutch DB2 User Group Meeting,
IBM Forum Amsterdam, 26 May 2009
Scope

- this presentation is aimed at:
  - all users of DB2:
    - writers of SQL
    - application developers
    - DBAs
    - database designers
  - all platforms, i.e.: z/OS and L/U/W
  - already at, or soon going to version 9
Goal

- overview of DB2 9 new possibilities
  - datatypes: BIGINT, DECFLOAT, BINARY
  - SQL scalar functions
  - evolutions: DW; XML; application programming

- some thoughts on standardisation
  - Edgar Codd and domains
  - SQL ISO/ANSI standards
  - IEEE standards
  - DB2 cross-platform uniformisation
  - compatibility: Oracle, MySQL, SQLServer...
Agenda

- datatypes and standardisation
- DB2 cross-platform uniformisation
- text: VARCHAR, BINARY, UNICODE
- numeric datatypes: BIGINT
- numeric datatypes: DECFLOAT
- floating-point: the IEEE-754 standard
- scalar functions for DECFLOAT
- scalar functions for text data
- more (new) functions
thoughts on datatypes

- Edgar Codd's heritage:
  - relational model ≠ physical implementation
  - domains: more than just datatypes:
    - semantics of a column; referential integrity
    - range and precision (granularity, quantisation)
  - “date”: example of non-elementary datatype
    - combination of 3 numbers (day, month, year)
    - Interface ('2009-05-26') ≠ internal representation
  - even text is non-trivial:
    - CCSIDs: mapping of byte value to character set
    - Unicode, esp. UTF-8
    - VARCHAR: length as prefix ↔ nul-terminated
SQL standardisation and datatypes

- **SQL:1999**
  - CHAR(n), VARCHAR(n), CLOB(n), BLOB(n)
  - INT, SMALLINT, BIGINT, DECIMAL(n,p)
  - REAL, FLOAT(n), DOUBLE
  - DATE, TIME, TIMESTAMP, INTERVAL
- **SQL:2003**
  - BOOLEAN
- **SQL:2008**
  - BINARY, VARBINARY; BOOLEAN dropped
- for a next release?
  - DECFLOAT
Agenda

- datatypes and standardisation
- DB2 cross-platform uniformisation
- text: VARCHAR, BINARY; UNICODE
- numeric datatypes: BIGINT
- numeric datatypes: DECFLOAT
- floating-point: the IEEE-754 standard
- scalar functions for DECFLOAT
- scalar functions for text data
- more (new) functions
Evolutions in DB2

• why?
  - helps application development
  - “education” - also DBA (cross-platform)

• how?
  - bring SQL syntactically closer to each other

• why historic differences z/OS - LUW?
  - different programming languages:
    • COBOL, PL/I: DECIMAL, CHAR(n)
      INT, SMALLINT
    • C, Java: VARCHAR
      INT, SMALLINT, BIGINT
    REAL, DOUBLE (“FLOAT”)
Importance of data representation

- (mis)matching data representation database ↔ program
  - (un)efficient communication
  - (no) unexpected surprises:
    - numeric precision; rounding effects
    - overflow; underflow
    - truncation; trailing blanks
    - CCSID interpretation
  - education; standardisation
- cf. misunderstanding of NULL
Agenda

- datatypes and standardisation
- DB2 cross-platform uniformisation
- text: VARCHAR, BINARY; UNICODE
- numeric datatypes: BIGINT
- numeric datatypes: DECFLOAT
- floating-point: the IEEE-754 standard
- scalar functions for DECFLOAT
- scalar functions for text data
- more (new) functions
not so new ... but still ...
- data representation:
  - DB2: 2-byte length + actual data
  - C: actual data, ended with '\0'
- DB2 v8 allows “alter table alter column”
  - CHAR(64) → VARCHAR(1024)
  - consequences?
- DB2 9 for z/OS: RRF (reordered row format)

getting the feel for:
- (mis)matching data representation
- strict separation: “concept” vs “physical implem.”
VARCHAR

• CHAR(n) vs VARCHAR(n):
  - not transparent to application!
    • data in DB2 is VARCHAR, host variable is CHAR:
      ==> must RTRIM before INSERT/UPDATE
    • data in DB2 is CHAR, host variable is VARCHAR:
      ==> must RTRIM during SELECT
      (could be made transparent through a view)
  - advantage of VARCHAR:
    • less average storage ==> more rows per I/O page
  - disadvantage of VARCHAR:
    • runtime computation of start of next column (→ RRF)
• PADDED vs NOT PADDED indexes (z/OS)
BINARY and VARBINARY

- new datatypes in DB2 9 for z/OS
  - alias “CHAR(n) FOR BIT DATA”
  - better (?) name: “BYTE”
- guarantee of non-interpreted bytes
  - CHAR will auto-convert CCSIDs (since v8)
  - cf. BLOB
- usage: “code” field; pictures; x'FF'; TINYINT
- most elementary datatype!
  - e.g.: INT == BINARY(4) with interpretation
  - CHAR(30) = BINARY(30) w. interpretation
  - note: CHAR(30) ≠ 30 characters!
Unicode

- Avoids CCSID conversion problems
  - CCSIDs 87 and 500 (EBCDIC) have no š š Ł ő
  - ISO 8859-2 has no à è ê û æ å ô
  - CCSIDs 87 and 500: inconsistently map |!¢¬[]^
- More than 256 characters => 1 char ≠ 1 byte
- Code points vs encoding (UTF-8, UTF-16)
- UTF-8:
  - 1-byte chars: digits, non-accented chars, punct
  - 2-byte chars: most “short” alphabets
  - 3-byte chars: e.g. €, Japanese, Chinese (simpl)
  - 4-byte chars: e.g. Gothic, Chinese (full)
Unicode: caveats

- DB2 v8 for z/OS: catalog in Unicode
- DB2 v8 for LUW: Unicode database
  - application declares its character encoding
  - bind time: all SQL is interpreted in this CCSID
  - runtime: host vars interpreted in this CCSID
  - DB2 converts where necessary
- ORDER BY: order of the table encoding
  - what about “virtual” tables / views?
  - application needs knowledge of data encoding
- v8 feature becomes potential nightmare!
  - e.g. LENGTH(s) → byte length or # characters?
Agenda

- datatypes and standardisation
- DB2 cross-platform uniformisation
- text: VARCHAR, BINARY; UNICODE
- numeric datatypes: BIGINT
- numeric datatypes: DECFLOAT
- floating-point: the IEEE-754 standard
- scalar functions for DECFLOAT
- scalar functions for text data
- more (new) functions
BIGINT

- 64-bit (8-byte) signed integer
  - hardware-supported on 64-bit processors
  - return type of RID(tbl)
  - maps to “long int” / “bigint” in host languages
- ranges from $-9 \times 10^{18}$ to $9 \times 10^{18}$
  - $(2^{63} = 9.223.372.036.854.775.808)$
- already in DB2 v8 for LUW
datatypes and standardisation
DB2 cross-platform uniformisation
text: VARCHAR, BINARY; UNICODE
numeric datatypes: BIGINT
numeric datatypes: DECFLOAT
floating-point: the IEEE-754 standard
scalar functions for DECFLOAT
scalar functions for text data
more (new) functions
DECFLOAT

- non-integral numbers:
  - DECIMAL(n,p): fixed precision n, fixed scale p
  - FLOAT(n): REAL (32 bit) & DOUBLE (64 bit)
  - since v9: DECIMAL FLOAT (DECFLOAT)

- DECIMAL: advantages:
  - integer based arithmetic
  - decimal representation ==> no rounding errors

- FLOAT: advantages:
  - each number stores its decimal point position
    ==> larger range than DEC, still fixed precision

- DECFLOAT: combines these advantages
floating point: representation

- general form: \((-1)^s \times c \times b^{1-p} \times b^q\)
  - \(b\): base (10 or 2 or ...), fixed; \(p\): precision, fixed
  - \(s\): sign; \(c\): \(p\)-digit “coefficient”; \(q\): exponent (int)
- examples: (\(b=10\))
  - 123.40  ==>  \((-1)^0 \times 1.2340 \times 10^2\)
  - -999e6   ==>  \((-1)^1 \times 9.99 \times 10^8 = -999000000\)
  - 88.77e-6 ==>  \((-1)^0 \times 8.877 \times 10^{-5} = 0.00008877\)
- exponent: \(q\) between -emax & emax (=\(2^w-1\))
- storage needs 3 parts:
  - \(s\) (1bit), \(c\) (\(p\) digits), \(q\) (\(w+1\) bits).
  ==> typically > 75% for \(c\), < 20% for \(q\).
floating point: IEEE-754

- IEEE standard from 1985 for b=2, p=23 or 52
  - supported by compilers and hardware (FPU)
  - p=23, emax=127: needs 1+23+1+7=32 bits
  - p=52, emax=1023: needs 1+52+1+10=64 bits
  - these are the datatypes REAL and DOUBLE
- advantage of b=2: fast multiply; compact
- disadvantage of b=2: I/O conversion dec/bin
- largest possible REAL (based on this):
  \[ 1.111111111111111111111111 \times 10^{11111111} \text{ (binary)} \]
  \[ = (2^{24}-1) \times 2^{127-22} = 1.68 \times 10^7 \times 4.05 \times 10^{31} = 6.806 \times 10^{38} \]
  with precision of 6.92 decimal digits (23 bits)
floating point: IEEE-754

- IEEE-754 also defines +Inf, -Inf, NaN, sNaN:
  - $-\infty = -\text{Inf} < \text{any finite number} < +\text{Inf} = +\infty$
  - $+x / 0 = +\text{Inf}; -x / 0 = -\text{Inf}; 0 / 0 = \text{NaN}$
  - $+\text{Inf} + x = +\text{Inf}; x - +\text{Inf} = -\text{Inf}, ...$
  - $x / \text{Inf} = 0; \text{Inf} / \text{Inf} = \text{NaN}$
  - all operations with NaN return NaN

- Overflow / underflow
  - due to limited emax (independent of precision p):
    - example (p=5, emax=7):
      - $-1.1111\times10^4 \times 2.0000\times10^4 = -\text{Inf} (\text{with overflow})$
      - $1.1111\times10^{-5} / 1000 = 0 \ (\text{with underflow})$
**floating point: IEEE-754**

- IEEE-754 was extended in aug. 2008:
  - $b=2$, $p=112$, $emax=16383$ $\Rightarrow$ “long double”
  - $b=10$, $p=16$, $emax=384$ $\Rightarrow$ DECFLOAT(16)
  - $b=10$, $p=34$, $emax=6144$ $\Rightarrow$ DECFLOAT(34)
- “long double” has been in use for a while
- “DECIMAL FLOAT” is relatively new
  - see e.g. [http://speleotrove.com/decimal/](http://speleotrove.com/decimal/)
- HW support: z9, z10, Power6 (IBM)
- SW support: gcc 4.2; IBM compilers; DB2 9
DECFLOAT

- **Datatype DECFLOAT(16):**
  - 8 bytes of storage
  - 16 digits precision
  - largest: $9.999\ldots9 \times 10^{384}$
  - smallest positive: $10^{-383}$
  - ==> 53.1 bits needed
  - ==> exp needs 9.5 bits

- **Datatype DECFLOAT(34):**
  - 16 bytes of storage
  - 34 digits precision
  - largest: $9.999\ldots9 \times 10^{6144}$
  - smallest positive: $10^{-6143}$
  - ==> 113 bits needed
  - ==> exp needs 14 bits
DECFLOAT: constructors

Table with DECFLOAT(34) column:

```sql
CREATE TABLE t ( p INT NOT NULL, d DECFLOAT ) ;
INSERT INTO t(p,d) VALUES (1, 123.45) ;
INSERT INTO t(p,d) VALUES (2, -1000) ;
INSERT INTO t(p,d) VALUES (3, decfloat('111e99')) ;
INSERT INTO t(p,d) VALUES (4, CAST('-111e-99' AS decfloat)) ;
INSERT INTO t(p,d) VALUES (5, decfloat('-Inf')) ;
SELECT * FROM t ORDER BY d ;
```

<table>
<thead>
<tr>
<th>P</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>-Infinity</td>
</tr>
<tr>
<td>2</td>
<td>-1000</td>
</tr>
<tr>
<td>4</td>
<td>-1.11E-97</td>
</tr>
<tr>
<td>1</td>
<td>123.45</td>
</tr>
<tr>
<td>3</td>
<td>1.11E+101</td>
</tr>
</tbody>
</table>
DECFLOAT: computations

- (notice truncation and rounding effects)

```
SELECT * FROM t WHERE d < -1111 ;
SELECT p, d + 1.23 FROM t ;
SELECT p, d * decfloat('1.000e-2') FROM t WHERE d >= -1000 ;
```

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>-Infinity</td>
</tr>
<tr>
<td>1</td>
<td>124.68</td>
</tr>
<tr>
<td>2</td>
<td>-998.77</td>
</tr>
<tr>
<td>3</td>
<td>1.110000000000000000000000000000000E+101</td>
</tr>
<tr>
<td>4</td>
<td>1.230000000000000000000000000000000</td>
</tr>
<tr>
<td>5</td>
<td>-Infinity</td>
</tr>
<tr>
<td>1</td>
<td>1.2345000</td>
</tr>
<tr>
<td>2</td>
<td>-10.0000</td>
</tr>
<tr>
<td>3</td>
<td>1.11000E+99</td>
</tr>
<tr>
<td>4</td>
<td>-1.11000E-99</td>
</tr>
</tbody>
</table>
DECFLOAT: rounding modes

- **CURRENT DECFLOAT Rounding Mode** (special register)

  - **ROUND_HALF_UP**: “default”
    
    \[
    \begin{array}{llll}
    1.432 & \rightarrow & 1.4 & 1.750 \rightarrow 1.8 & -1.750 \rightarrow -1.8 \\
    \end{array}
    \]

  - **ROUND_DOWN**: truncation
    
    \[
    \begin{array}{llll}
    1.432 & \rightarrow & 1.4 & 1.750 \rightarrow 1.7 & -1.750 \rightarrow -1.7 \\
    \end{array}
    \]

  - **ROUND_FLOOR**: “towards -∞”
    
    \[
    \begin{array}{llll}
    1.432 & \rightarrow & 1.4 & 1.750 \rightarrow 1.7 & -1.750 \rightarrow -1.8 \\
    \end{array}
    \]

  - **ROUND_CEILING**: “towards +∞”
    
    \[
    \begin{array}{llll}
    1.432 & \rightarrow & 1.5 & 1.750 \rightarrow 1.8 & -1.750 \rightarrow -1.7 \\
    \end{array}
    \]

  - **ROUND_HALF_EVEN**: last digit even
    
    \[
    \begin{array}{llll}
    1.432 & \rightarrow & 1.4 & 1.750 \rightarrow 1.8 & 1.85 \rightarrow 1.8 \\
    \end{array}
    \]
**DECFLOAT: child diseases**

- **Several APARs related to DECFLOAT**
  
  DB2 9.5 for LUW - 5 may 2008
  IZ09711: POSSIBLE STACK CORRUPTION CONVERTING DECFLOAT(16) TO DOUBLE
  The conditions under which this problem can arise are fairly specific:
  * only conversions from DECFLOAT(16) to double are affected;
  * one must be carrying out an affected operation on a DECFLOAT(16) value;
  * the DECFLOAT(16) value being converted must contain 16 digits; and
  * the DECFLOAT(16) value, x, being converted must be in the range 1E-6 < ABS(x) < 1.
  
  Problem conclusion
  First fixed in DB2 UDB Version 9.5, FixPak 1

  IZ12232: NEW DECFLOAT COLUMN IN 9.5 NOT IN RESULTSETS FOR CLI FUNCTIONS
  When a table is created with the new DECFLOAT column in 9.5, the column information is not in the result set from SQLColumns, although columns values can still be selected.

  ...  

- **Still some misunderstanding → education**
DECFLOAT: current use

- XML
  - DB2 9 has new datatype “XML”
  - DB2 9 can extract fragments from XML documents (by using Xpath)
  - XML data is textual
    even for numeric data: '123.45' '1.2345e2'
  - Conversion errors are unacceptable
    real('0.2') → 0.1999996 (binary 1.100110011...e11)
    decfloat('0.2') → 0.2000000
- “float”-like programming interface in SQL
Agenda

- datatypes and standardisation
- DB2 cross-platform uniformisation
- text: VARCHAR, BINARY; UNICODE
- numeric datatypes: BIGINT
- numeric datatypes: DECFLOAT
- floating-point: the IEEE-754 standard
- scalar functions for DECFLOAT
- scalar functions for text data
- more (new) functions
functions: DECFLOAT-related

- **NORMALIZE_DECFLOAT(f)**
  - returns a decfloat in “normalized” form
    
    e.g. `normalize_decfloat('56.7800e+8') → '5.678e+9'

- **QUANTIZE(f, q)**
  - returns truncated/rounded or denormalized form
  
    e.g. `quantize('56.7800e+8','100e7') → '568e+7'
    
    e.g. `quantize('56.7800e+8','100.00000e7') → '567.80000e+7'
  
  - rounding mode is obeyed!
  
  - returns a DECFLOAT(34)
    
    unless both arguments are DECFLOAT(16)
functions: DECFLOAT-related

- **COMPARE_DECFLOAT(f1, f2)**
  - returns:
    - 0 if f1 & f2 are “physically” equal
    - 1 if f1 < f2
    - 2 if f1 > f2
    - 3 if f1 & f2 are not comparable (f1 or f2 is NaN)
    
      e.g. `compare_decfloat('Inf', 3.141592) → 2`

- **TOTALORDER(f1, f2)** → used by ORDER BY & WHERE
  - returns:
    - 0 if f1 & f2 are “logically” equal
    - -1 if f1 < f2 (where 'Inf' < 'NaN')
    - 1 if f1 > f2
    
      e.g. `totalorder(decfloat('3.1415920'), 3.141592) → -1`
Agenda

- datatypes and standardisation
- DB2 cross-platform uniformisation
- text: VARCHAR, BINARY; UNICODE
- numeric datatypes: BIGINT
- numeric datatypes: DECFLOAT
- floating-point: the IEEE-754 standard
- scalar functions for DECFLOAT
- scalar functions for text data
- more (new) functions
functions: string handling

“units”: OCTETS or CODEUNITS32
(meaning: in bytes or in characters)
no units allowed when s is BINARY

• CHARACTER_LENGTH(s, units)
  - cf. LENGTH(s): in bytes!
• LOCATE_IN_STRING(s, patt [,pos[,n], units])
  - cf. POSSTR(s, patt), POSITION(patt, s[, units]),
    LOCATE(patt, s [, startpos] [, units])
• SUBSTRING(s, startpos [, len], units)
  - cf. SUBSTR(s, startpos [, len])
functions: string handling

- **LPAD(s, n [, pad-char])**
  - prepend s with n spaces
- **RPAD(s, n [, pad-char])**
  - append s with n spaces
- **INSERT(s, pos, len, repl [, units])** [already in v8]
  - at start position pos, replace len bytes by repl
- **OVERLAY(s, repl, pos[, len], units)**
  - idem
functions: encoding-related

- **UNICODE(c)**
  - returns the Unicode “code point” (int) of char c

- **ASCII(c)** [already in DB2 v8 for LUW]
  - returns the ordinal position of char c in ASCII

- **ASCII_CHR(n)  CHR(n)**
  - returns the character at position n in ASCII

- **EBCDIC_CHR(n)**
  - returns the character at position n in EBCDIC

- **UNICODE_STR(s)**
  - returns the Unicode “translation” of string s

- **ASCII_STR(s), EBCDIC_STR(s)**
functions: encoding-related

- **COLLATION_KEY**(s, collation_name)
  - to be used for “cultural sort”
  - return value only useful in mutual compare
  - example:

  ```
  SELECT name 
  FROM   clients 
  WHERE  COLLATION_KEY(name,'UCA400R1_AS_LNL_S1_NO') 
         BETWEEN COLLATION_KEY('VAAA','UCA400R1_AS_LNL_S1_NO') 
            AND COLLATION_KEY('VZZZ','UCA400R1_AS_LNL_S1_NO') 
  ORDER BY COLLATION_KEY(name,'UCA400R1_AS_LNL_S1_NO')
  ```

  A: punctuation; L: locale; S: case&accents; N: normalisation

- UCA collation_name: encodes elements like
  case-(in)sensitive; ignore-whitespace; ignore-accents; ignore-punctuation; country-specific alphabet; ... 

Agenda

- datatypes and standardisation
- DB2 cross-platform uniformisation
- text: VARCHAR, BINARY; UNICODE
- numeric datatypes: BIGINT
- numeric datatypes: DECFLOAT
- floating-point: the IEEE-754 standard
- scalar functions for DECFLOAT
- scalar functions for text data
- more (new) functions
SOUNDEX and related

- **SOUNDEX(s)**
  - “sounds as”
  - returns a 4-character code (1 letter, 3 digits)
  - equal soundex-code: similar pronunciation

- **DIFFERENCE(s1,s2)**
  - similarity of soundex-codes (0, 1, 2, 3, or 4)
    - 4: very similar
    - 0: very dissimilar

[already in DB2 v8 for LUW]
Unicode related

• NORMALIZE_STRING(s, form)
  - s: Unicode string
  - form: one of NFC, NFD, NFKC, NFKD
    NFD: canonical decomposition
      example: é → e + ´
    NFC: canonical decomposition + composition
      example: é → e + ´ → é
    NFKD: compatibility decomposition
    NFKC: compat. decomposition + composition
Date & time related

- **EXTRACT**(year FROM dt) [already in DB2 v8]
  - this is the new SQL standard
  - also: “month”, “day”, “hour”, “minute”, “second”
- **TIMESTAMPADD**(interval-unit, n, ts)
  - interval-unit: 1: µs; 2: s; 4: min; 8: h; ...; 256: y
  - cf. **TIMESTAMPDIFF**(interval-unit, ts1-ts2)
- **VARCHAR_FORMAT**(ts, format)
  - Format: e.g. 'DD/MM/YYYY HH24:MI:SS'
  - Alias (LUW only): TO_CHAR()
- **TIMESTAMP_FORMAT**(string, format)
  - Alias (LUW only): TO_DATE()
aggregate functions

- COUNT, SUM, AVG, MIN, MAX
  - Already from day 1...
- STDDEV(col), STDDEV_SAMP(col)
  - standard deviation (already in v8)
- VARIANCE, VARIANCE_SAMP
  - square or stddev (already in v8)
- CORRELATION(col1, col2)
  - between -1 and 1
- COVARIANCE, COVARIANCE_SAMP
- XMLAGG(xmlexpr ORDER BY expr)
  - aggregate concat! (already in v8)
Conclusions

• new datatypes: blessing or curse?
  ==> **opportunity**!
• text: CHAR / VARCHAR / BINARY
  ==> make the right choice
  ==> careful with encoding
  ==> Unicode
• numeric: consider using DECFLOAT
  ==> where appropriate
  ==> INT / DECIMAL / FLOAT
• new functions: blessing or curse?
  ==> **standardisation**!
Q & A

...
DB2 9 new datatypes and SQL functions: blessing or curse?

Peter Vanroose
pvanroose@abis.be

ABIS Training & Consulting
Leuven / Woerden
http://www.abis.be/

Dutch DB2 User Group Meeting,
IBM Forum Amsterdam, 26 May 2009