Session F16

DB2 V8 Unicode Support

Christopher J. Crone

Presented By

Tapio Kirjonen / IBM Finland

IBM DB2 Information Management Technical Conference

Sept. 20-24, 2004

Las Vegas, NV

Presentation Topics

DB2 Character Conversion Fundamentals, how and why

Unicode - What is it?

DB2 V8 Unicode Support

- How is Unicode Used inside DB2 for z/OS V8
- New Character Based Functions
- Multiple CCSID Support
- Utility Support

Gotchas!!!

Terminology

For the purposes of this presentation

- ASCII - all ASCII CCSIDs that DB2 currently supports
- EBCDIC - all EBCDIC CCSIDs that DB2 currently supports
- UNICODE - UTF-8 or UTF-16
- Encoding Scheme
  - ASCII, EBCDIC or Unicode
- CCSID – Coded Character Set Identifier **(used by DB2 to tag string data)**
  - Two-byte, unsigned binary integer identifying a specific set of encoding scheme and one or more pairs of character sets (CS) and code pages (CP)
- CCSID set
  - The single byte CCSID value (SBCS), mixed CCSID value and double byte CCSID value (DBCS) associated with a particular encoding scheme
- Multiple CCSID Sets
  - When two or more CCSID sets contain different CCSID values for one or more values in a set (SBCS, Mixed or DBCS).

IBM Software Group

DB2 Character Conversion Fundamentals – How and Why

DB2 Information Management Software

business on demand software

Why do we convert?

HEX DIGITS 1ST → 2ND ↓	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0	(SP) SP010000	& SM030000	- SP100000	ø LO610000	Ø LO620000	° SM190000	µ SM170000	^ SD150000	{ SM110000	} SM140000	\ SM070000	0 ND100000
-1	(RSP) SP300000	é LE110000	/ SP120000	É LE120000	a LA010000	j LJ010000	~ SD190000	£ SC020000	A LA020000	J LJ020000	÷ SA060000	1 ND010000
-2	â LA150000	ê LE150000	Â LA160000	Ê LE160000	b LB010000	k LK010000	s LS010000	¥ SC050000	B LB020000	K LK020000	S LS020000	2 ND020000
-3	ä LA170000	ë LE170000	Ä LA180000	Ë LE180000	c LC010000	l LL010000	t LT010000	· SD630000	C LC020000	L LL020000	T LT020000	3 ND030000
-4	à LA130000	è LE130000	À LA140000	È LE140000	d LD010000	m LM010000	u LU010000	© SM520000	D LD020000	M LM020000	U LU020000	4 ND040000
-5	á LA110000	í LI110000	Á LA120000	Í LI120000	e LE010000	n LN010000	v LV010000	§ SM240000	E LE020000	N LN020000	V LV020000	5 ND050000
-6	ã LA190000	î LI150000	Ã LA200000	Î LI160000	f LF010000	o LO010000	w LW010000	¶ SM250000	F LF020000	O LO020000	W LW020000	6 ND060000
-7	å LA270000	ï LI170000	Å LA280000	Ï LI180000	g LG010000	p LP010000	x LX010000	¼ NF040000	G LG020000	P LP020000	X LX020000	7 ND070000
-8	ç LC410000	ì LI130000	Ç LC420000	Ì LI140000	h LH010000	q LQ010000	y LY010000	½ NF010000	H LH020000	Q LQ020000	Y LY020000	8 ND080000
-9	ñ LN190000	ß LS610000	Ñ LN200000	` SD130000	i LI010000	r LR010000	z LZ010000	¾ NF050000	I LI020000	R LR020000	Z LZ020000	9 ND090000
-A	¢ SC040000	! SP020000	¦ SM650000	: SP130000	« SP170000	ª SM210000	¡ SP030000	[SM060000	(SHY) SP320000	1 ND011000	2 ND021000	3 ND031000
-B	. SP110000	$ SC030000	, SP080000	# SM010000	» SP180000	º SM200000	¿ SP160000	] SM080000	ô LO150000	û LU150000	Ô LO160000	Û LU160000
-C	< SA030000	* SM040000	% SM020000	@ SM050000	ð LD630000	æ LA510000	Ð LD620000	¯ SM150000	ö LO170000	ü LU170000	Ö LO180000	Ü LU180000
-D	(SP060000	) SP070000	_ SP090000	' SP050000	ý LY110000	¸ SD410000	Ý LY120000	" SD170000	ò LO130000	ù LU130000	Ò LO140000	Ù LU140000
-E	+ SA010000	; SP140000	> SA050000	= SA040000	þ LT630000	Æ LA520000	Þ LT640000	´ SD110000	ó LO110000	ú LU110000	Ó LO120000	Ú LU120000
-F	| SM130000	¬ SM660000	? SP150000	" SP040000	± SA020000	¤ SC010000	® SM530000	× SA070000	õ LO190000	ÿ LY170000	Õ LO200000	(EO)

Code Page 00037

HEX DIGITS 1ST → 2ND ↓	4-	5-	6-	7-	8-	9-	A-	B-	C-	D-	E-	F-
-0	(SP) SP010000	& SM030000	- SP100000	ø LO610000	Ø LO620000	° SM190000	µ SM170000	¢ SC040000	{ SM110000	} SM140000	\ SM070000	0 ND100000
-1	(RSP) SP300000	é LE110000	/ SP120000	É LE120000	a LA010000	j LJ010000	~ SD190000	£ SC020000	A LA020000	J LJ020000	÷ SA060000	1 ND010000
-2	â LA150000	ê LE150000	Â LA160000	Ê LE160000	b LB010000	k LK010000	s LS010000	¥ SC050000	B LB020000	K LK020000	S LS020000	2 ND020000
-3	ä LA170000	ë LE170000	Ä LA180000	Ë LE180000	c LC010000	l LL010000	t LT010000	· SD630000	C LC020000	L LL020000	T LT020000	3 ND030000
-4	à LA130000	è LE130000	À LA140000	È LE140000	d LD010000	m LM010000	u LU010000	© SM520000	D LD020000	M LM020000	U LU020000	4 ND040000
-5	á LA110000	í LI110000	Á LA120000	Í LI120000	e LE010000	n LN010000	v LV010000	§ SM240000	E LE020000	N LN020000	V LV020000	5 ND050000
-6	ã LA190000	î LI150000	Ã LA200000	Î LI160000	f LF010000	o LO010000	w LW010000	¶ SM250000	F LF020000	O LO020000	W LW020000	6 ND060000
-7	å LA270000	ï LI170000	Å LA280000	Ï LI180000	g LG010000	p LP010000	x LX010000	¼ NF040000	G LG020000	P LP020000	X LX020000	7 ND070000
-8	ç LC410000	ì LI130000	Ç LC420000	Ì LI140000	h LH010000	q LQ010000	y LY010000	½ NF010000	H LH020000	Q LQ020000	Y LY020000	8 ND080000
-9	ñ LN190000	ß LS610000	Ñ LN200000	` SD130000	i LI010000	r LR010000	z LZ010000	¾ NF050000	I LI020000	R LR020000	Z LZ020000	9 ND090000
-A	[SM060000	] SM080000	¦ SM650000	: SP130000	« SP170000	ª SM210000	¡ SP030000	¬ SM660000	(SHY) SP320000	1 ND011000	2 ND021000	3 ND031000
-B	. SP110000	$ SC030000	, SP080000	# SM010000	» SP180000	º SM200000	¿ SP160000	| SM130000	ô LO150000	û LU150000	Ô LO160000	Û LU160000
-C	< SA030000	* SM040000	% SM020000	@ SM050000	ð LD630000	æ LA510000	Ð LD620000	¯ SM150000	ö LO170000	ü LU170000	Ö LO180000	Ü LU180000
-D	(SP060000	) SP070000	_ SP090000	' SP050000	ý LY110000	¸ SD410000	Ý LY120000	" SD170000	ò LO130000	ù LU130000	Ò LO140000	Ù LU140000
-E	+ SA010000	; SP140000	> SA050000	= SA040000	þ LT630000	Æ LA520000	Þ LT640000	´ SD110000	ó LO110000	ú LU110000	Ó LO120000	Ú LU120000
-F	! SP020000	^ SD150000	? SP150000	" SP040000	± SA020000	¤ SC010000	® SM530000	× SA070000	õ LO190000	ÿ LY170000	Õ LO200000	(EO)

Code Page 00500

What are CCSIDs used for?

DB2 uses CCSIDs to describe data stored in the DB2 subsystem

DB2 supports specification of CCSIDs at a subsystem level

Starting with V7, DB2 supports 3 encoding schemes

- ASCII
- EBCDIC
- UNICODE

Conversion methods

Native DB2

SYSIBM.SYSSTRINGS (V2.3)

OS Conversion services

ICONV (Requires OS/390 V2R9 and above)

Uses LE base services (V6 & V7 only)

OS/390 V2 R8/R9/R10 Conversion Services (V7)

31 Bit only

z/OS support for Unicode (V8)

31 and 64 bit capable (after z/OS V1R3).

Conversion services

- Central repository for OS/390 system
  - Used by
    - ODBC Driver
    - COBOL
    - DB2
- High performance
  - Uses HW instructions available in zSeries 800, 900, and 990
  - Uses page fixed tables in a data space
- Conversion image built by off-line utility
  - CUNMIUTL - see sample in hlq.SCUNJCL (CUNJIUTL)
- Administered via OS/390 Console
  - SET UNI
  - DISPLAY UNI
- Default specified by PARMLIB member (CUNUNIxx)

Conversion services configuration

Which Conversions should be configured

- CCSID 367 (7-bit ASCII) <-> ASCII & EBCDIC System CCSID(s)
- CCSID 1208 (UTF-8) <-> ASCII & EBCDIC System CCSID(s)
- CCSID 1200 (UTF-16) <-> ASCII & EBCDIC System CCSID(s)
- Client CCSID(s) <-> Unicode CCSIDs (367, 1208, 1200)
- CCSID 37,500, and 1047 <-> Unicode CCSIDs (367, 1208, 1200)
- ASCII or EBCDIC Conversions not included in SYSSTRINGS
- Other
  - Conversions needed to LOAD/UNLOAD Data
  - Conversions needed to support application encoding bind option, DECLARE VARIABLE, or CCSID overrides

See example in Reference section in back of presentation

Round Trip - vs - Enforced Subset

Round Trip (RT) Conversions

- Preserves codepoints that are not representable in both codepages
- Work well in a two-tier environment

Enforced Subset (ES) Conversions

- Codepoints that are not representable are converted to SUB character
- Works well in an heterogeneous environment

DB2 Uses a combination of RT and ES conversions

- Trend is toward ES conversions
- Continue to use RT conversions in some cases for compatibility reasons

Unicode and RT/ES Conversions

- ASCII/EBCDIC -> Unicode conversions are RT
- Unicode -> ASCII/EBCDIC conversions are ES

When does conversion occur?

- Local
  - Generally, conversion does not occur for local applications
  - When dealing with ASCII/Unicode tables
  - When specified by application
    - CCSID Override in SQLDA (V2.3)
    - Declare Variable (V7)
    - Application Encoding Bind Option (V7)
    - Current Application Encoding Special Register (V7)
  - ODBC/JDBC/SQLJ
- Remote
  - Automatically when needed
    - DRDA Receiver Makes Right

IBM Software Group

Unicode – What is it?

DB2 Information Management Software

e business on demand software

Why Unicode?

Unicode is a single character set that encodes all of the worlds scripts **(sort of)**

The Unicode standard provides a cross platform, cross vendor method of encoding data that enbles lossless representation and manipulation

Before Unicode

- Many Standards
  - ANSI, JIS, TISI
- Provided by various vendors
  - IBM (ASCII and EBCDIC), HP, Microsoft…

Foundation for globalization of data

- Customers want to use DB2 to manage data from around the world
- No pre-Unicode code page handles all characters
- Handles most current and historical languages
- Handles scientific and mathematical symbols
- Handles other symbols: windings, dingbats, . . .
- Required by many modern standards: XML, Java, LDAP, CORBA

Required by many VARs

- SAP, PeopleSoft, Siebel… all require DBMSs that support Unicode

Unicode fundamentals

Three forms of Unicode

UTF-8

Unicode Transformation Format in 8 bits

UTF-16

Unicode Transformation Format in 16 bits

UTF-32

Unicode Transformation Format in 32 bits

Introduced with Unicode Technical Report # 19 to replace UCS-4

Character examples

Character	ASCII	UTF-8	UTF-16 (Big Endian format)	UTF-32 (Big Endian format)
A	'41'x	'41'x	'0041'x	'00000041'x
a	'61'x	'61'x	'0061'x	'00000061'x
9	'39'x	'39'x	'0039'x	'00000039'x
Å (The character A with Ring accent)	'C5'x	'C385'x Note: 'C5'x becomes double byte in UTF-8	'00C5'x	'000000C5'x
顕 U+9860	'CDDB'x (CCSID 939)	'E9A1A0'x	'9860'x	'00009860'x
𠀑 U+200D0	N/A	'F0A08390'x	'D840DCD0'x	'000200D0'x

Note: UCS-2/UTF-16 and UCS-4/UTF-32 are using a technique called Zero Extension

Endianess

Big Endian

- pSeries, zSeries, iSeries, Sun, HP
- Most significant byte is leftmost
  - For a 4 byte word - Byte order 0,1,2,3

Little Endian

- Intel based machines including xSeries
- Least significant byte is leftmost
  - For a 4 byte word - Byte order 3,2,1,0

UTF-8 - not affected by endianess issues

UTF-16 and UTF-32 are effected by endianess issues

- Big Endian
  - 'A' = x'0041' for UTF-16 or x'00000041' for UTF-32
- Little Endian
  - 'A' = x'4100' for UTF-16 or x'41000000' for UTF-32

Note: A BYTE is always ordered as leftmost most significant bit to rightmost least significant bit. Bit order within a byte is always 7,6,5,4,3,2,1,0

String length issues

Conversions can cause the length of a string to change

Expanding Conversions

When data converted from one CCSID to another expands

For Example Å - 'C5'x in CCSID 819 -> 'C385'x in CCSID 1208

Contracting Conversions

When data converted from one CCSID to another contracts

For Example Å - '00C5'x in CCSID 1200 -> 'C5'x in CCSID 819

Combining Characters

Å can be represented as

'00C5'x for UTF-16 (or 'C385'x for UTF-8)

'0041030A'x for UTF-16 (or '41CC8A'x for UTF-8)

Allocate columns for storage length, not display length

IBM Software Group

DB2 UDB for z/OS
V8 Unicode Support

DB2 Information Management Software

e business on demand software

DB2 Internal Processing

Most internal Processing is done in Unicode

- Parsing – Literals in parse tree are in Unicode
- Utility parsing – Option to parse in Unicode
- Precompiler – Source converted to Unicode, parsed in Unicode
- DBRM – Unicode in NFM (New Function Mode)
- Bind – Statement text converted to Unicode if necessary
- Optimization – Catalog look up in Unicode (even in COMPAT Mode)
- Authorization – Unicode internal, EBCDIC external (e.g. RACF)
- Internal names – PLAN Name, Package Name
- Special Registers – Stored in Unicode
- Tracing – Option to output trace data in Unicode (certain fields)

Specifying Unicode as the encoding scheme

System level Unicode CCSIDs during installation

Database

CREATE DATABASE mydb CCSID UNICODE

Table space

CREATE TABLESPACE myts IN mydb CCSID UNICODE

Table

CREATE TABLE t1 (c1 CHAR(10)) CCSID UNICODE

Other objects, for example:

CREATE PROCEDURE

mysp (in in_parm1 char(10) ccsid unicode) . . .

How is Unicode data stored?

Storage of Unicode Data

- Char/VarChar/CLOB FOR SBCS DATA
  - (7-bit) ASCII this is a subset of UTF-8 CCSID 367
- Char/VarChar/CLOB [FOR MIXED DATA]
  - UTF-8 CCSID 1208
- Graphic/VarGraphic/DBCLOB
  - UTF-16 CCSID 1200

Parsing Statements

DB2 V8 Parsing is in Unicode UTF-8

- Conversion to UTF-8 (CCSID 1208) will occur before parsing
- Literal values will be preserved
- Application Encoding CCSID used to “interpret” literals

Catalog Encoding

DB2 V8 Catalog is encoded in Unicode UTF-8

- Static statements are stored in the catalog in Unicode
- View definitions are stored in the catalog in Unicode

Any name which must be passed to z/OS must be convertible to EBCDIC. For example:

Database Name	(dataset qualifier)
Table space/Index space Name	(dataset qualifier)
DBRM Name	(PDS member name)
(UDF, SP, Exits, Fieldproc...)	(PDS member name)

Identifiers may not be generateable from all clients so should really be limited to common subset that is representable on all clients.

Accessing the Unicode catalog

Automatic conversion to the CCSID of the application, specified by:

- Application encoding scheme
- Host variable declaration

Predicates

- Equal predicates get converted from the application encoding scheme, so should work the same
- Range predicates are where differences may occur . . .

EBCDIC	hex value		Unicode (ASCII)	hex value
space	'40'x		space	'20'x
lower case	'81-89'x '91-99'x 'A1-A9'x		numerals	'30-39'x
upper case	'C1-C9'x 'D1-D9'x 'E1-E9'x		upper case	'40-4F'x '50-5A'x
numerals	'F0-F9'x		lower case	'61-6F'x '70-7A'x

Literals

Character literals may be used for all string data

```
INSERT INTO T1 (C1) VALUES ('Å');
INSERT INTO T1 (G1) VALUES ('abc'); -- converted to UTF-16
```

Graphic literals should only be used for Graphic data

```
INSERT INTO T1 (G1) VALUES (G'阿唎'); -- U+80E2 U+8137
```

Hex literals should only be used for character data

```
INSERT INTO T1 (C1) VALUES (X'3132' );
```

UX (can be used for UTF-8 or UTF-16) and GX constants added

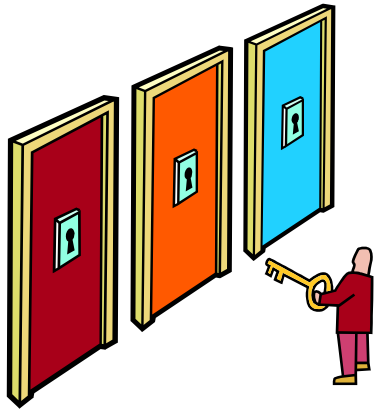
```
INSERT INTO T1 (C1) VALUES (UX'80E28137');
INSERT INTO T1 (G1) VALUES (GX'42C142C2');
```

GX encoding is determined by Application Encoding Scheme

Host Variables and Parameter Markers

Host variable / parameter type:

- ASCII / EBCDIC / Unicode
  - Char
  - Graphic
- Unicode
  - UTF-8
  - UTF-16



DB2 data storage:

- Unicode
  - UTF-8
  - UTF-16
- ASCII / EBCDIC / Unicode
  - CHAR
  - GRAPHIC

Applications don't need to change just because the back end data store changes

Controlling Encoding

DECLARE VARIABLE statement

New way to allow CCSID to be specified for host variables

Example

```
EXEC SQL DECLARE :hv1 CCSID UNICODE;
EXEC SQL DECLARE :hv2 CCSID 37;
```

Precompiler directive to treat hostvar as a specific CCSID

Useful for PREPARE / EXECUTE IMMEDIATE statement text

```
EXEC SQL PREPARE S1 FROM :hv2;
```

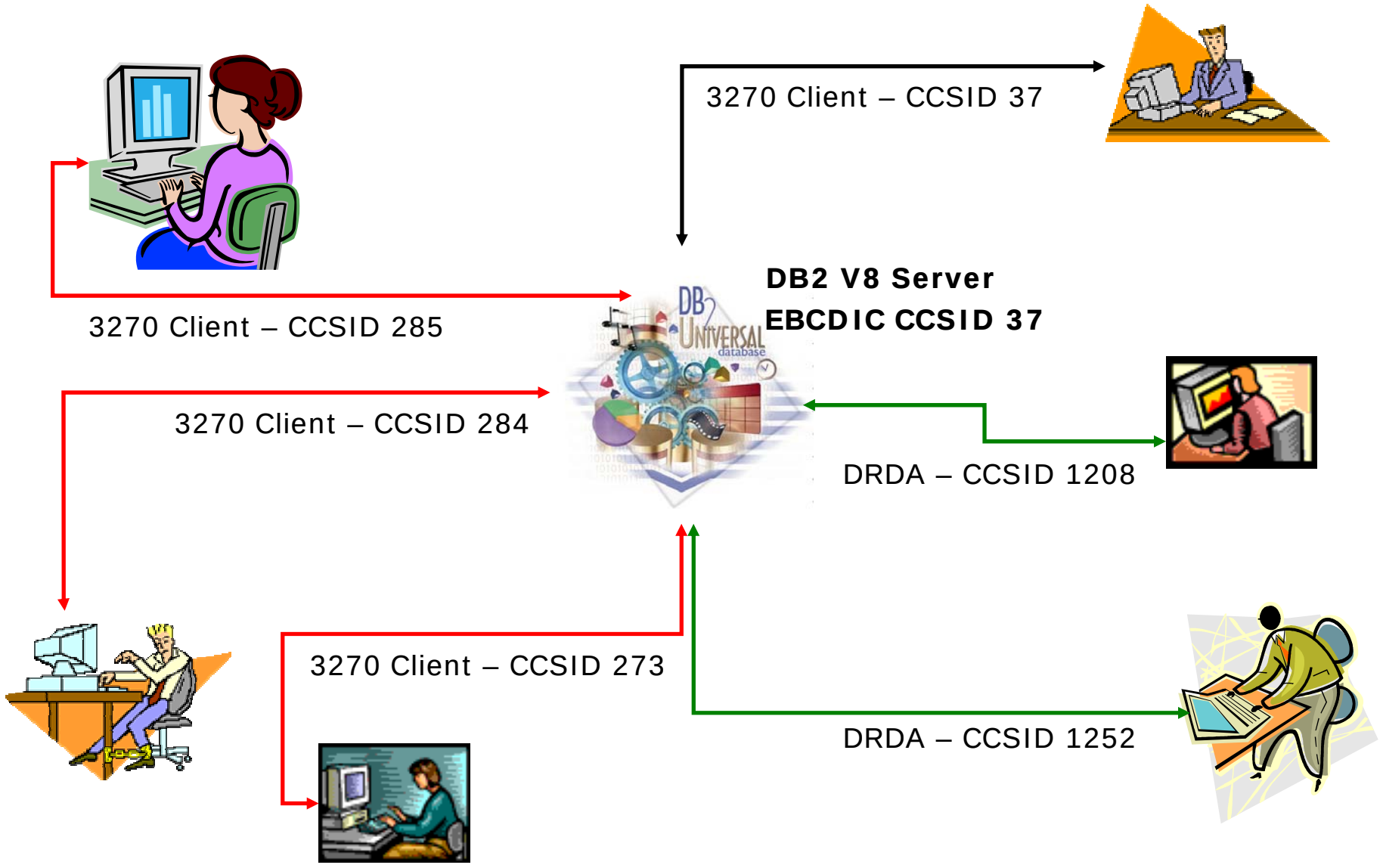
May be used with any string host variable on input or output

Application Encoding Scheme

New Application Encoding Scheme

- System Default
  - Determines Encoding Scheme when none is explicitly specified
- Bind Option
  - Allows explicit specification of ES at an application level. Affects Static SQL - Provides default for dynamic
  - System Default used if bind option not specified
- Special Register
  - Allows explicit specification of ES at the application level. Affects Dynamic SQL
  - Initialized with Bind Option
- OPTION is ignored (for CCSID info) when packages are executed remotely
  - DRDA specified Input CCSID, Data flows as it to client

Application Encoding Example



ODBC/SQLJ/JDBC

V7 ODBC Support

Support for Wide Character API's (UCS2/UTF-16)

See ODBC Guide and Reference (SC26-9941-01)

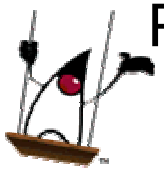
```
SQLRETURN
   SQLPrepare (
    SQLHSTMT hstmt,
    SQLCHAR *szSqlStr,
    SQLINTEGER cbSqlStr );
```

```
SQLRETURN
   SQLPrepareW (
    SQLHSTMT hstmt,
    SQLWCHAR *szSqlStr,
    SQLINTEGER cbSqlStr );
```

V8 ODBC Support

CURRENTABBENSCH ini file setting

SQLJ/JDBC Support



Remove current support for converting to EBCDIC before calling engine. Let DB2 engine determine where conversion is necessary

COBOL

Enterprise COBOL for z/OS and OS/390 V3R1 Supports Unicode

- NATIONAL is used to declare UTF-16 variables
  - MY-UNISTR pic N(10). -- declares a UTF-16 Variable
- N and NX Literals
  - N'123'
  - NX'003100320033'
- Conversions
  - NATIONAL-OF Converts to UTF-16
  - DISPLAY-OF Converts to specific CCSID

```
Greek-EBCDIC pic X(10) value "ΞΣΦΛΘΖΔΓΩ".
UTF16STR pic N(10).
UTF8STR pic X(20).
Move Function National-of(Greek-EBCDIC, 00875) to UTF16STR.
Move Function Display-of(UTF16STR, 01208) to UTF8STR.
```

Functions & Routines

Functions – are by default not character based

LENGTH, SUBSTR, POSSTR, LOCATE

Byte Oriented for SBCS and Mixed (UTF-8)

Double-Byte Character Oriented for DBCS (UTF-16)

Cast Functions

UTF-16/UTF-8 accepted anywhere char is accepted (char, date, integer...)

SELECT DATE(graphic column) FROM T1;

SELECT INTEGER(graphic column) FROM T1;

UTF-8 is result data type/CCSID 1208 for character functions

SELECT CHAR(float_col) FROM T1;

Routines

UDFs, UDTFs, and SPs will all be enabled to allow Unicode parameters

Parameters will be converted as necessary between char (UTF-8) and graphic (UTF-16)

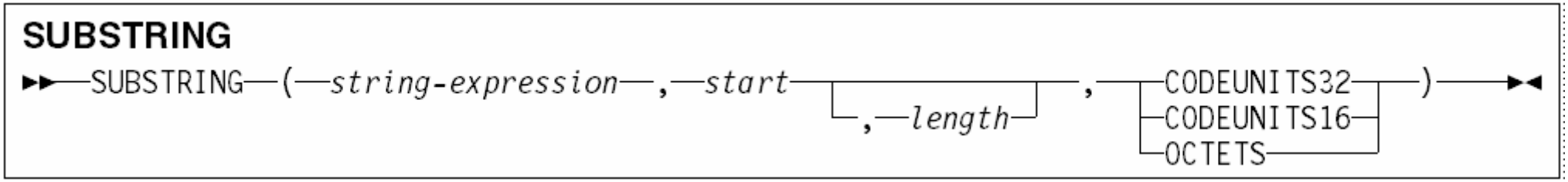
Date/Time/Timestamp passed as UTF-8 (ISO Format)

Character Based Functions (PQ88784)

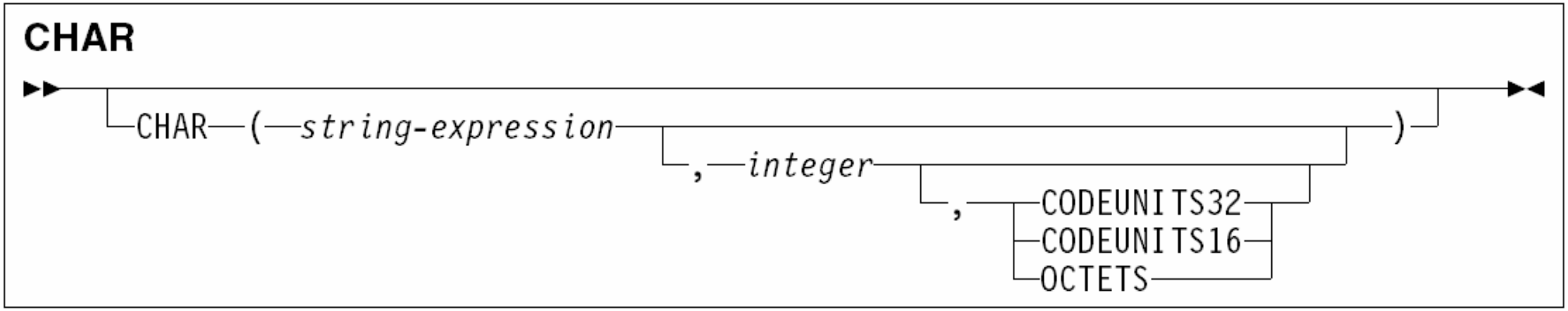
- New functions
  - CHARACTER_LENGTH
  - POSITION
  - SUBSTRING
- Updated Functions
  - CHAR
  - CLOB
  - DBCLOB
  - GRAPHIC
  - INSERT
  - LEFT
  - LOCATE
  - RIGHT
  - VARCHAR
  - VARGRAPHIC
- CAST Specification
  - Changes to enable specification of Code Units

Character Based Functions (sample syntax)

SUBSTRING



CHAR



Character Based Functions - Example

Assume that NAME is a VARCHAR(128) column, encoded in Unicode UTF-8, that contains 'Jürgen'. The following query:

```
SELECT CHARACTER_LENGTH(NAME,CODEUNITS32)
FROM T1 WHERE NAME = 'Jürgen';
```

or

```
SELECT CHARACTER_LENGTH(NAME,CODEUNITS16)
FROM T1 WHERE NAME = 'Jürgen';
```

returns the value 6. A similar query:

```
SELECT CHARACTER_LENGTH(NAME,OCTETS)
FROM T1 WHERE NAME = 'Jürgen';
```

or

```
SELECT LENGTH(NAME)
FROM T1 WHERE NAME = 'Jürgen';
```

returns the value 7.

Name	UTF-8 Representation	UTF-16 Representation	UTF-32 Representation
Jürgen	x'4AC3BC7267656E'	x'004A00FC007200670065006E'	x'0000004A000000FC00000072000000670000006500000006E'

Multiple CCSID Sets - Example 1:

```
SELECT  a.name, a.creator, b.charcol, 'ABC',
                        :hvchar, X'C1C2C3'
FROM sysibm.systables a,
            ebcdictable b
WHERE a.name = b.name AND
              b.name > 'B' AND
              a.creator = 'SYSADM'
ORDER BY b.name;
```

In the above example, since both tables have the same system EBCDIC CCSID set, the comparisons are done in EBCDIC and the result data is EBCDIC.

Multiple CCSID Sets - Example 1 (continued)

```
SELECT  a.name, a.creator, b.charcol, 'ABC',
                    :hvchar, X'C1C2C3'
FROM sysibm.systables a,
          ebcdictable b
WHERE a.name = b.name AND
          b.name > 'B' AND
          a.creator = 'SYSADM'
ORDER BY b.name;
```

Result or Evaluated:

EBCDIC

Unicode

Application Encoding Scheme

Assuming a Unicode catalog, the result will contain multiple CCSIDs and the comparisons and ordering will be dependent on the context.

SQL statements with multiple CCSID sets

Comparison and resulting data types for multiple CCSID sets...

If an expression or comparison involves two strings which contain columns with different CCSID sets,

Drive to Unicode if necessary

WHERE T1.C1 = T2.C1

If an expression or comparison involves two strings with different CCSID sets where only one of them contains a column,

Drive to the column's CCSID set

WHERE T1.C1 = X'C1C2'

If an expression or comparison involves two strings with different CCSID sets and neither contains a column,

Drive to Unicode

WHERE GX'42C142C2' = X'C1C2'

String constants and special registers in a context by themselves use the application encoding scheme

SELECT ‘ABC‘ FROM T1 . . .

Example of CAST to influence ordering

Given table names: TA, TB, T1, T2

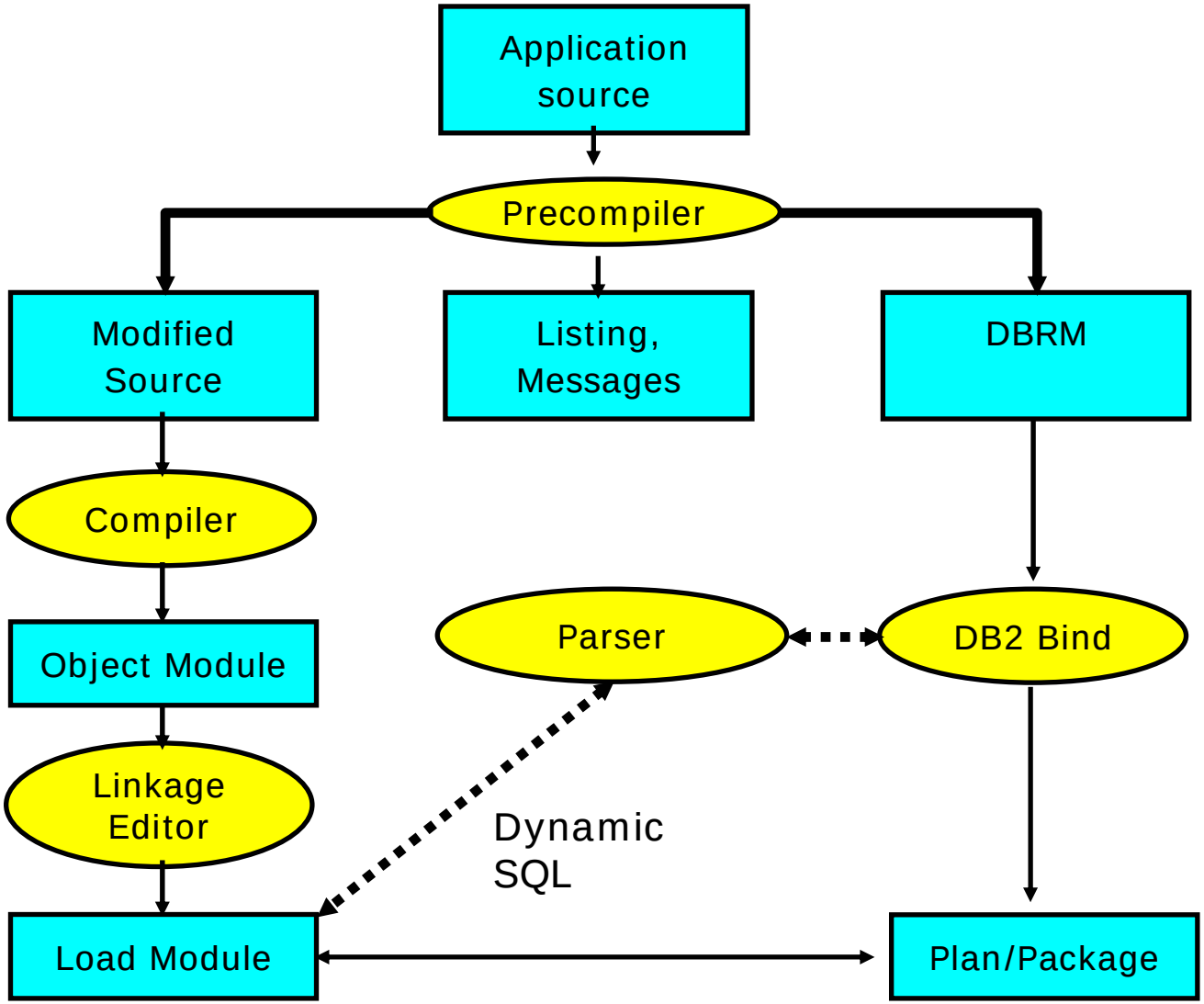
```
SELECT NAME
FROM SYSIBM.SYSTABLES
WHERE NAME LIKE 'T%'
ORDER BY NAME
```

In EBCDIC (V7), returns:
TA, TB, T1, T2
In Unicode (V8), returns:
T1, T2, TA, TB

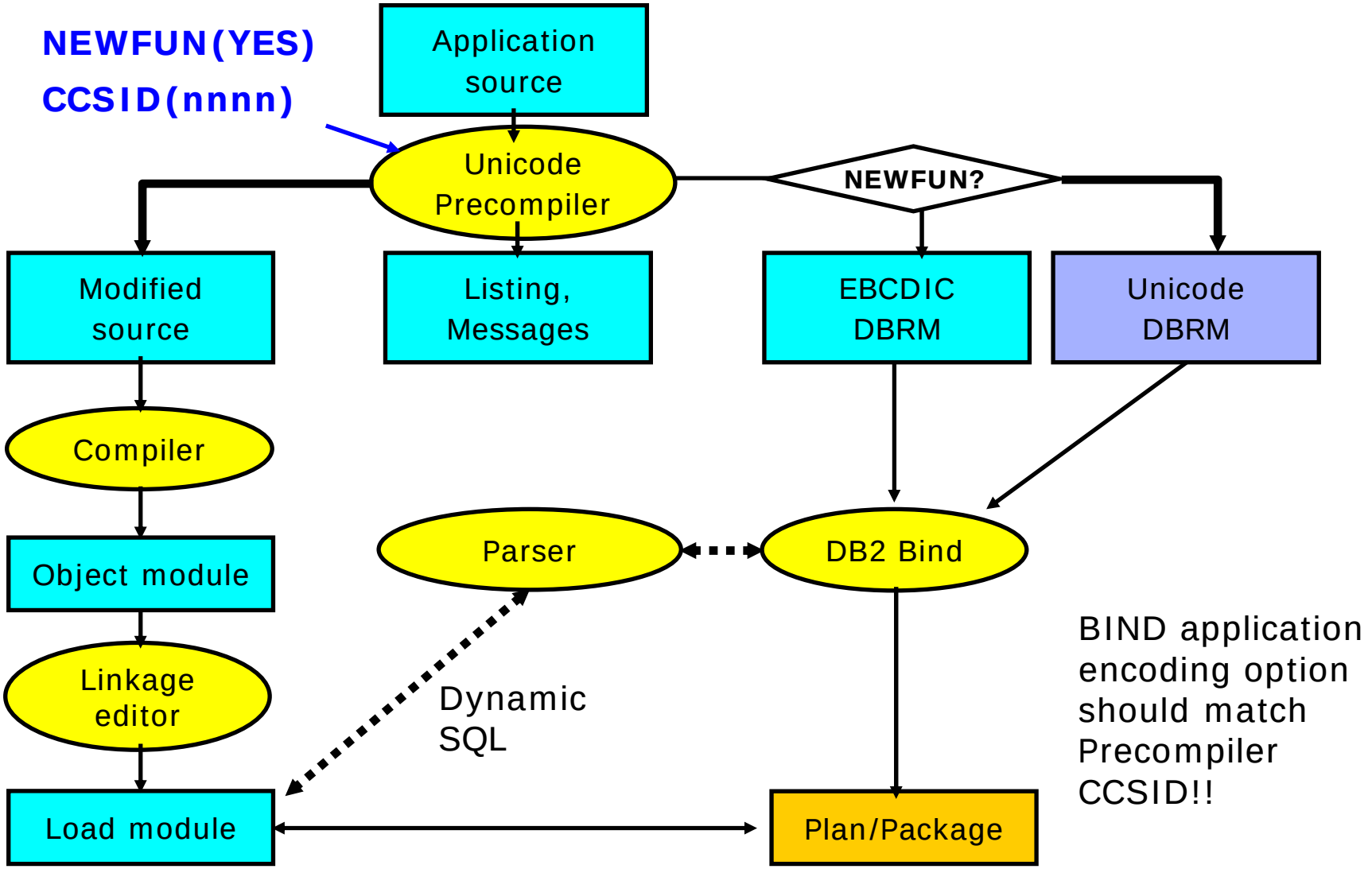
```
SELECT
 CAST (NAME AS CCSID
 EBCDIC)
          AS E_NAME
FROM SYSIBM.SYSTABLES
WHERE NAME LIKE 'T%'
ORDER BY E_NAME
```

Returns:
TA, TB, T1, T2

Program preparation - V7



Program Preparation V8



Utility Unicode statements

Utility control statements may be specified in Unicode or EBCDIC

- DB2 detects which encoding scheme is being used
- Must be all UTF-8 or EBCDIC - no mixing!
- Object names in messages will be in EBCDIC

New utility stored procedure interface DSNUTILU for Unicode

- Identical to DSNUTILS except:
  - Inputs are in UNICODE
  - utility_name parameter dropped
  - Data set DYNALLOC keywords dropped
    - use TEMPLATE for all data sets

Utility control statements

EBCDIC:

```
//SYSIN      DD    *
   QUIESCE TABLESPACE A.B
   COPY TABLESPACE A.B
/*
```

UNICODE:

```
//SYSIN    DD *
"éíñáëääá"è â<áë& ää" "â"""""""""""""""""""
"ä|&ß"è â<áë& ää" "â"""""""""""""""""""""""
/*
```

DSNUTILS –vs– DSNUTILU

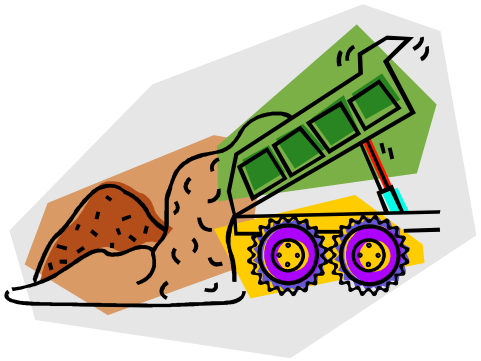
```
CREATE PROCEDURE DSNUTILS
  ( IN UTILITY_ID VARCHAR(16)
  , IN RESTART VARCHAR(8)
  , IN UTSTMT VARCHAR(32704)
  , OUT RETCODE INTEGER
  , IN UTILITY_NAME VARCHAR(20)
  , IN RECDSN VARCHAR(54)
  , IN RECDEVT CHAR(8)
  , IN RECSPACE SMALLINT
  , IN DISCDSN VARCHAR(54)
  , IN DISCDEVT CHAR(8)
  , IN DISCSPACE SMALLINT
  , IN PNCHDSN VARCHAR(54)
  , IN PNCHDEVT CHAR(8)
  , IN PNCHSPACE SMALLINT
  , IN COPYDSN1 VARCHAR(54)
  , IN COPYDEVT1 CHAR(8)
  , IN COPYSPACE1 SMALLINT   . . .
```

```
CREATE PROCEDURE DSNUTILU
  ( IN UTILITY_ID VARCHAR(16) CCSID UNICODE
  , IN RESTART VARCHAR(8) CCSID UNICODE
  , IN UTSTMT VARCHAR(32704) CCSID UNICODE
  , OUT RETCODE INTEGER)
  EXTERNAL NAME DSNUTILU
  LANGUAGE ASSEMBLE
  WLM ENVIRONMENT WLMENV1
  NO COLLID
  RUN OPTIONS 'TRAP(OFF)'
  PROGRAM TYPE MAIN
  MODIFIES SQL DATA
  ASUTIME NO LIMIT
  STAY RESIDENT NO
  COMMIT ON RETURN NO
  PARAMETER STYLE GENERAL
  RESULT SETS 1
  EXTERNAL SECURITY USER;
```

Conversion support in Load and Unload

- LOAD Utility
  - UTF-16 <-> UTF-8
    - SBCS/MIXED -> DBCS
    - DBCS -> SBCS/MIXED
  - ASCII/EBCDIC <-> UNICODE

- UNLOAD Utility
  - ASCII/EBCDIC <-> UNICODE
    - No support for
      - SBCS/MIXED -> DBCS
      - DBCS -> SBCS/MIXED



Gotchas!!!

DB2 Information Management Software

SPUFI, DSNTEP2, and Hex constants

UTF-16 and SPUFI or DSNTEP2

- SPUFI and DSNTEP2 really aren't UTF-16 aware
  - In most cases, you should use CHAR(graphic column) when selecting data.
  - For example, use:
    - SELECT CHAR(g1) FROM T1
  - Not
    - SELECT g1 FROM T1

Hex constants are character based

- INSERT INTO T1 (g1) VALUES(x'0041'); -- will result in x'00000041' not x'0041' as you might expect. Because hex constants are character based, DB2 will convert from UTF-8 to UTF-16 for you.
  - x'00' -> x'0000' and x'41' -> x'0041'
- In V8, use GX or UX hex constants to avoid this problem

SYSSTMT and SYSPACKSTMT

Statement text in SYSSTMT and SYSPACKSTMT

- EBCDIC for
  - Applications Precompiled Prior To DB2 V8
  - Applications Precompiled in NEWFUN(NO) mode
- Unicode for
  - Applications Precompiled in NEWFUN(YES) mode

String constants have new maximum lengths

Checking the length of a string constant

- DB2 uses the Unicode representation of the string constant
  - Stored length might differ from the length that you entered
    - e.g., if the string constant is in a CCSID other than UTF-8
- A string that was valid in Version 7 might be flagged as too long in Version 8

How can this occur?

- If the string contains one or more characters whose Unicode representations require more bytes than their original representations
- Expansion can cause the string to grow beyond the maximum allowed length
- V7 maximum length was 255; V8 maximum length is 32704
- Incompatability can exist in all modes of DB2 or disappear on entering NFM
  - Might be flagged as too long in any mode
    - **ALTER INDEX** or **CREATE INDEX**, **VALUES(constant)**
  - Might be too long in CM, but not in NFM
    - Expression in a **WHERE clause**

IBM Software Group

Reference Material

DB2 Information Management Software

e business on demand software

References

DB2 UDB for z/OS Version 8:
Everything You Ever Wanted to Know , ... and More - SG24-6079
DB2 UDB for z/OS Internationalization Guide
http://www.ibm.com/software/data/db2/zos/pdf/ccmstr.pdf
DB2 Universal Database Administration Guide - SC09-2946
Appendix E - National Language Support
The Unicode Standard Version 4.0
The Unicode Consortium - Addison-Wesley - www.unicode.org
Character Data Representation Architecture: Reference & Registry
SC09-2190
National Language Design Guide Volume 2 - SE09-8002
eBusiness Globalization Solution Design Guide, Getting Started
SG24-6851-00

Appendix – z/OS Support for Unicode

z/OS support for Unicode (V7 & V8) - Conversion Services

- Available in OS/390 as download - Included in base code for z/OS 1.2+
- Conversion Services had major changes between z/OS 1.2 and 1.3, conversion images must be rebuilt (i.e., re-run job to define conversions)
- Documentation :
  - Manual: *z/OS: Support for Unicode(TM): Using Conversion Services* (SC33-7050)
  - Additional configuration in information APARs II13048, II13049, and II13277
  - Requires OS/390 V2R8 and above + APAR OW44581
  - code and program directory
    - http://www6.software.ibm.com/dl/os390/unicodespt-p
  - documentation
    - http://publibfp.boulder.ibm.com/pubs/pdfs/os390/cunpde00.pdf
    - http://publibfp.boulder.ibm.com/pubs/pdfs/os390/cunuge00.pdf
    - Information APAR II13048 and II03049

z/OS Conversion Services (64 Bit enabled) (V8)

- Requires z/OS V1R2 and above + OW56703 and OW56704
- Documentation:
  - Pointers to new documentation contained in OW56703 and OW56704

Appendix - zSeries Unicode Support

The UTF-8 <-> UTF-16 instructions are used when DB2 converts from char <-> graphic. These instructions are used on G5, G6, and zSeries 800,900, 890, and 990:

- CUUTF - Convert UTF-16 to UTF-8
- CUTFU - Convert UTF-8 to UTF-16

The following two instructions are similar to CLCLE and MVCLE. DB2 will use these instructions to perform comparison and padding on UTF-16 data. These instructions are used on zSeries 800, 900, 890, and 990:

- CLCLU - Compare logical long UNICODE
- MVCLU - Move logical long UNICODE

These instructions pack/unpack ASCII (also UNICODE UTF-8) and UNICODE (UTF-16) data. These instructions are used on zSeries 800, 900, 890, and 990:

- PKU - Pack Unicode
- PKA - Pack ASCII
- UNPKU - Unpack Unicode
- UNPKA - Unpack ASCII

These instructions are all used when DB2 performs conversion. DB2 indirectly uses these instructions via the Conversion System Services:

- TRTT - Translate Two to Two
- TRTO - Translate Two to One
- TROT - Translate One to Two
- TROO - Translate One to One

Conversion Services Example

```
//CUNMIUTL EXEC PGM=CUNMIUTL
//SYSPRINT DD   SYSOUT=*
//TABIN    DD   DISP=SHR,DSN=hlq.SCUNTBL
//SYSIMG   DD   DSN=hlq.IMAGES(CUNIMG00),DISP=SHR
//SYSIN    DD   *
/*** INPUT STATEMENTS FOR THE IMAGE GENERATOR ***/
CONVERSION 0037,1200,ER;  /*EBCDIC  037 -> UTF-16      */
CONVERSION 0037,1208,ER;  /*EBCDIC  037 -> UTF-8       */
CONVERSION 0037,0367,ER;  /*EBCDIC  037 -> ASCII 367   */
CONVERSION 1200,0037,ER;  /*UTF-16      -> EBCDIC  037 */
CONVERSION 1208,0037,ER;  /*UTF-8       -> EBCDIC  037 */
CONVERSION 0367,0037,ER;  /*ASCII 367   -> EBCDIC  037 */
CONVERSION 1252,1200,ER;  /*ASCII  1252 -> UTF-16      */
CONVERSION 1252,1208,ER;  /*ASCII  1252 -> UTF-8       */
CONVERSION 1252,0367,ER;  /*ASCII  1252 -> ASCII 367   */
CONVERSION 1200,1252,ER;  /*UTF-16      -> ASCII  1252 */
CONVERSION 1208,1252,ER;  /*UTF-8       -> ASCII  1252 */
CONVERSION 0367,1252,ER;  /*ASCII 367   -> ASCII  1252 */
CONVERSION 1208,1200,ER;  /*UTF-8       -> UTF-16      */
CONVERSION 0367,1200,ER;  /*ASCII 367   -> UTF-16      */
CONVERSION 1200,1208,ER;  /*UTF-16       -> UTF-8      */
CONVERSION 0367,1208,ER;  /*ASCII 367    -> UTF-8      */
CONVERSION 1200,0367,ER;  /*UTF-16       -> ASCII 367  */
CONVERSION 1208,0367,ER;  /*UTF-8        -> ASCII 367  */
/*
```

Conversion Services Example Display

```
14.34.14          d uni,all
14.34.15          CUN3000I 14.34.14 UNI DISPLAY 097
   ENVIRONMENT: CREATED        12/11/2002 AT 09.13.53
                MODIFIED       12/11/2002 AT 09.13.53
                IMAGE CREATED 12/06/2002 AT 17.10.01
       SERVICE: CUNMCNV   CUNMCASE
       STORAGE: ACTIVE        50 PAGES
                LIMIT    524287 PAGES
     CASECONV: NONE
    CONVERSION: 00037-00367-ER                00037-01208-ER
                00037-01200(13488)-ER         00367-00037-ER
                00367-01208-ER                00367-01200(13488)-ER
                00367-01252-ER                01200(13488)-00037-ER
                01200(13488)-00367-ER         01200-01208-ER
                01200(13488)-01252-ER         01208-00037-ER
                01208-00367-ER                01208-01200-ER
                01208-01252-ER                01252-00367-ER
                01252-01200(13488)-ER         01252-01208-ER
```

Where Is Encoding Information stored?

CCSIDs are stored in the following places

- SYSIBM.SYSDATABASE (V5)
- SYSIBM.SYSCOLUMNS (V8)
- SYSIBM.SYSPACKAGE (V7)
- SYSIBM.SYSPARMS (V6)
- SYSIBM.SYSPLAN (V7)
- SYSIMB.SYSROUTINES (V8)
- SYSIBM.SYSTABLES (V8)
- SYSIBM.SYSTABLESPACE (V5)
- SYSIBM.SYSVTREE (V5)
- Plans and Packages (SCT02 and SPT01)
- Directory (DSNDB01) (V5)
- DECP (V2.3)

In ENCODING_SCHEME column of - Stored as 'A', 'E', 'U', or blank (default)

- SYSIBM.SYSDATATYPES
- SYSIBM.SYSDATABASE
- SYSIBM.SYSPARMS
- SYSIBM.SYSTABLESPACE
- SYSIBM.SYSTABLES

Predicates

DB2 V7 Problem

Predicates limited to 255 bytes (except like – 4000 byte pattern)

DB2 V8 Solution

Predicates extended to 32K (except like – 4000 byte pattern)

Mixing of UTF-8 and UTF-16 allowed

Basic Predicate

SELECT … WHERE C1 = :HG1

(where C1 is UTF-8 and :HG1 is UTF-16)

Like predicate

SELECT … WHERE C1 LIKE :HG1 ESCAPE :HG2;

(where C1 is UTF-8 and :HG1 and :HG2 are UTF-16)

In Predicate

SELECT … WHERE C1 in (:HG1, :HV1);

(where C1 is UTF-8 and :HG1 is UTF-16 and HV1 is character)

Multiple CCSID Sets per SQL Statement

While there are no syntax changes to allow multiple CCSID sets, the following SQL statements may be affected.

ALTER TABLE & ALTER TABLE ADD (materialized query table)
CREATE TABLE (materialized query table)
CREATE TABLE LIKE view-table
CREATE GLOBAL TEMPORARY TABLE LIKE view-table
CREATE VIEW
DECLARE GLOBAL TEMPORARY TABLE AS (fullselect) DEFINITION ONLY
DECLARE GLOBAL TEMPORARY TABLE LIKE view-table
DELETE
INSERT
SELECT
SELECT INTO
UPDATE
Scalar fullselect expression

New EXPLAIN tables / columns - PLAN_TABLE

Column Name and Type	Description
TABLE_ENCODE CHAR(1)	Indicates the encoding scheme of the statement. If the statement represents a single CCSID set, then the column will contain 'E' for EBCDIC, 'A' for ASCII, or 'U' for Unicode. If the statement is a multiple CCSID set statement, then the column will be set to 'M' for multiple CCSID sets.
TABLE_SCCSID FIXED(16)	The SBCS CCSID value of the table or zero if the TABLE_ENCODE column is 'M'
TABLE_MCCSID FIXED(16)	The Mixed CCSID value of the table or zero if the TABLE_ENCODE column is 'M'
TABLE_DCCSID FIXED(16)	The DBCS CCSID value of the table or zero if the TABLE_ENCODE column is 'M'

New EXPLAIN tables / columns -- DSN_STATEMNT_TABLE

Column Name and Type	Description
STMT_ENCODE CHAR(1)	Indicates the encoding scheme of the statement. If the statement represents a single CCSID set, then the column will contain 'E' for EBCDIC, 'A' for ASCII, or 'U' for Unicode. If the statement is a multiple CCSID set statement, then the column will be set to 'M' for multiple CCSID sets.

UTF-8 (CCSID 1208)

ASCII Safe UNICODE (maps to 7-Bit ASCII)

- Bytes '00'x - '7F'x = 7-Bit ASCII

Bytes '00'x - '7F'x represented by single byte chars

Chars above '80'x are encoded by 2-6 byte chars

- Most characters take 2-3 bytes
  - Most Japanese, Chinese, and Korean characters take 3 bytes
  - Most Extended Latin characters take 2 bytes
- Surrogates take 4 bytes

UCS-2 (CCSID 13488, 17584)

Basic Multilingual Plane - BMP(0)

Pure Double Byte Characters

64K characters in Repertoire

'0000'x - '00FF'x Represent 8 bit ASCII

'00'x appended to 8 Bit ASCII characters

'00FF'x - 'FFFF'x Represent additional characters

Greek -> '0370'x - '03FF'x

Cyrillic -> '0400'x - '04FF'

...

UTF-16 (CCSID 1200)

UCS-2 with Surrogate Support

- Uses two two-byte characters to represent additional characters

~1 Million characters in repertoire

- BMP1-BMP16 (additional 16 planes).
  - Supplementary Multilingual Plane (SMP) - Plane 1
    - U+10000..U+1FFFF
  - Supplementary Ideographic Plane (SIP) - Plane 2
    - U+20000..U+2FFFF
  - Supplementary Special Purpose Plane (SSP) - Plane 14
    - U+E0000..U+EFFFF
  - BMP15 and BMP16 are reserved for private use

UTF-32

Each Character is 4 bytes

- Range is restricted to values '00000000'x - '0010FFFF'x
- Represents the same repertoire as UTF-16

UCS-4 Implemented by SUN Solaris and HP/UX as base Unicode data type

XPG/4 standard requires fixed width character format

- z/Series, p/Series looking at UTF-32 implementations to support surrogate characters in C/C++ applications

ODBC Application variables & encoding schemes

Format of String Data	ODBC symbolic C datatype[1]	ODBC Application Variable Type[2]	C base datatype
UTF-8	SQL_C_CHAR	SQLCHAR	char
UCS-2	SQL_C_WCHAR*	SQLWCHAR*	wchar_t
single/mixed-byte ASCII	SQL_C_CHAR	SQLCHAR	char
double-byte ASCII	SQL_C_DBCHAR	SQLDBCHAR	wchar_t
single/mixed-byte EBCDIC	SQL_C_CHAR	SQLCHAR	char or
double-byte EBCDIC	SQL_C_DBCHAR	SQLDBCHAR	wchar_t

* New in V8

1 the datatype of the C buffer used to store data in the application

2 used for declaring variables in the ODBC application

Query to get statement text from SYSSTMT

```
SELECT A.NAME, B.STMTNO, B.STMTNOI,
   CASE WHEN A.IBMREQD < 'L' OR
             A.IBMREQD='N' OR A.IBMREQD='Y' THEN
  B.TEXT
       ELSE
        CAST(
          CAST(B.TEXT AS VARCHAR(3500) CCSID 1208)
          AS VARCHAR(3500) CCSID EBCDIC)
       END
 FROM SYSIBM.SYSDBRM A,SYSIBM.SYSSTMT B
 WHERE A.NAME = B.NAME AND
      A.PLNAME = B.PLNAME AND
      NOT (B.STMTNO=0 AND B.SEQNO=0 AND
  B.SECTNO=0)
 ORDER BY A.NAME, A.PLNAME,B.STMTNO, B.STMTNOI;
```

Query to get statement text from SYSPACKSTMT

```
SELECT A.LOCATION, A.COLLID, A.NAME, A.CONTOKEN,
  A.VERSION,
       B.STMTNO, B.STMTNOI,
  CASE WHEN A.IBMREQD < 'L' OR
            A.IBMREQD='N' OR A.IBMREQD='Y' THEN B.STMT
       ELSE
        CAST(
          CAST(B.STMT AS VARCHAR(3500) CCSID 1208)
          AS VARCHAR(3500) CCSID EBCDIC)
       END
FROM SYSIBM.SYSPACKAGE A,SYSIBM.SYSPACKSTMT B
WHERE A.LOCATION = B.LOCATION AND
      A.COLLID   = B.COLLID   AND
      A.NAME     = B.NAME     AND
      A.CONTOKEN = B.CONTOKEN AND
      NOT (B.STMTNO=0 AND B.SEQNO=0 AND B.SECTNO=0)
ORDER BY A.LOCATION, A.COLLID, A.NAME,
  A.CONTOKEN,B.STMTNO,B.STMTNOI;
```