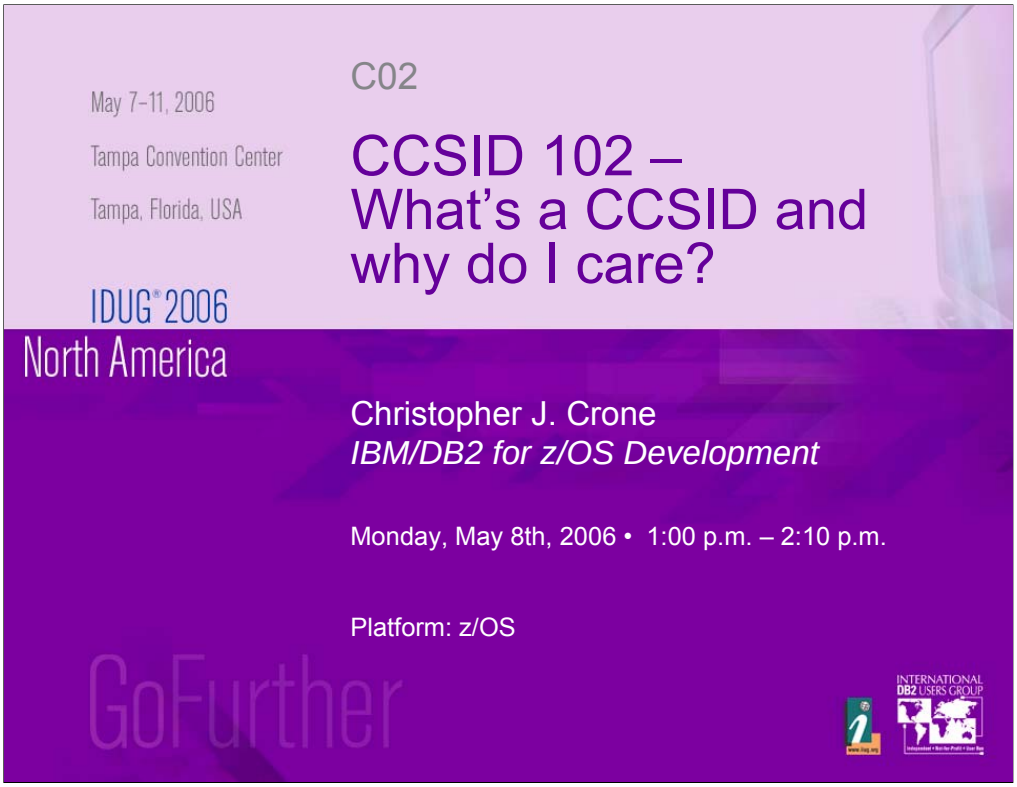
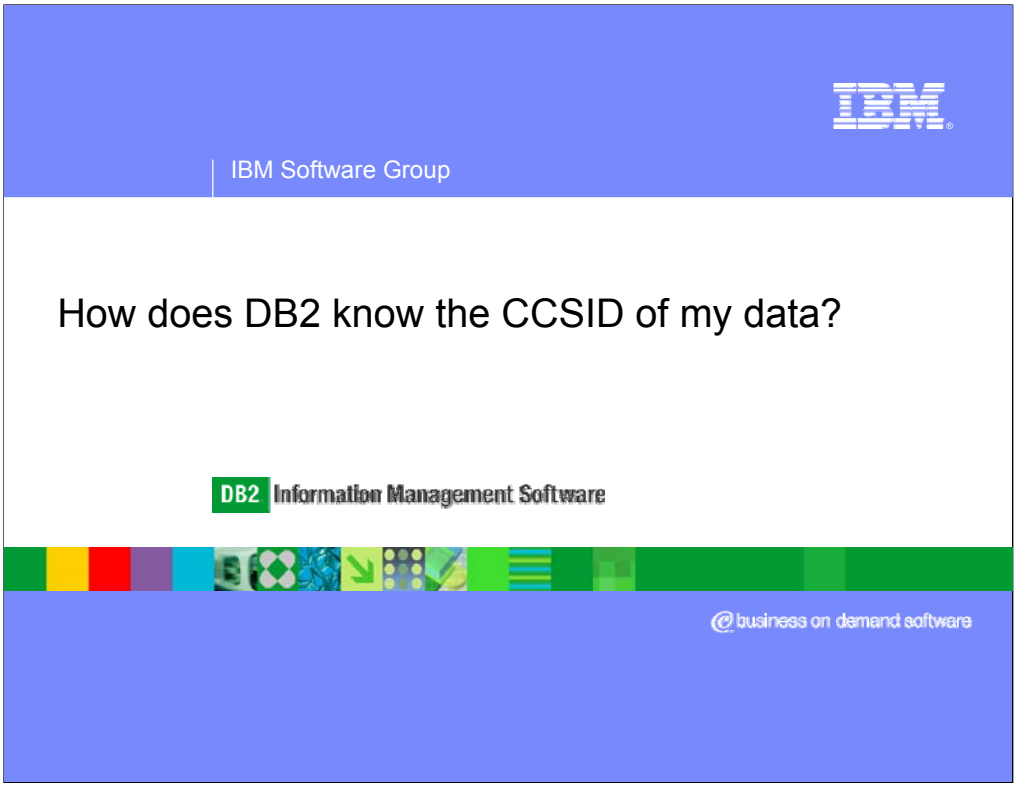

May 7–11, 2006
Tampa Convention Center
Tampa, Florida, USA
IDUG® 2006
North America
C02
CCSID 102 –
What's a CCSID and
why do I care?
Christopher J. Crone
IBM/DB2 for z/OS Development
Monday, May 8th, 2006 • 1:00 p.m. – 2:10 p.m.
Platform: z/OS
GoFurther
INTERNATIONAL
DB2 USERS GROUP

Presentation Topics

- How does DB2 know the CCSID of My data
- When does DB2 convert
- Character Based Functions
- Queries with more than one Encoding
- Putting it all together – Examples
  - DDL
  - SQL
  - Utilities

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
How does DB2 know the CCSID of my data?
DB2 Information Management Software
business on demand software

Installation (continued)

- Information from DSNTIPF ends up in Job DSNTIJUZ

Mixed System

```
DSNHDECM ASCCSID=1088,
         AMCCSID=949,
         AGCCSID=951,
         SCCSID=833,
         MCCSID=933,
         GCCSID=834,
         USCCSID=367,
         UMCCSID=1208,
         UGCCSID=1200,
         ENSCHEME=EBCDIC,
         APPENSCH=EBCDIC,
         MIXED=YES
END
```

Non-Mixed System

```
DSNHDECM ASCCSID=819,
         AMCCSID=65534,
         AGCCSID=65534,
         SCCSID=37,
         MCCSID=65534,
         GCCSID=65534,
         USCCSID=367,
         UMCCSID=1208,
         UGCCSID=1200,
         ENSCHEME=EBCDIC,
         APPENSCH=EBCDIC,
         MIXED=NO
END
```


The information from panel DSNTIPF flows to Job DSNTIJUZ.
In the case on the Left, we have a Mixed = Yes system that is set up to support Korea. The ASCII and EBCDIC system CCSIDs that actually would have been specified on panel DSNTIPF, to result in this specification, would have been 949 and 833.
For mixed systems, and for the Unicode CCSID, the Mixed CCSID is specified on install panel DSNTIPF and DB2 will pick the corresponding Single byte and Graphic (Double Byte) CCSIDs.
In the case on the Right, we have a Mixed = No system that is set up to support US English.
Note that the user specified 819 and 37 for the ASCII and EBCDIC Single byte CCSIDs and that DB2 used the value 65534 for the ASCII and EBCDIC Mixed and Graphic (Double byte) CCSIDs. 65534 is a reserved value that means no CCSID.
Also note that the Default Encoding and Default Application Encoding also flow to this job.

Note there is a bug in DSNTIJUZ and DSNHDECM - These ship with CCSID 500 as default.

Installation

Specification of CCSIDs is performed at installation via install Panel DSNTIPF

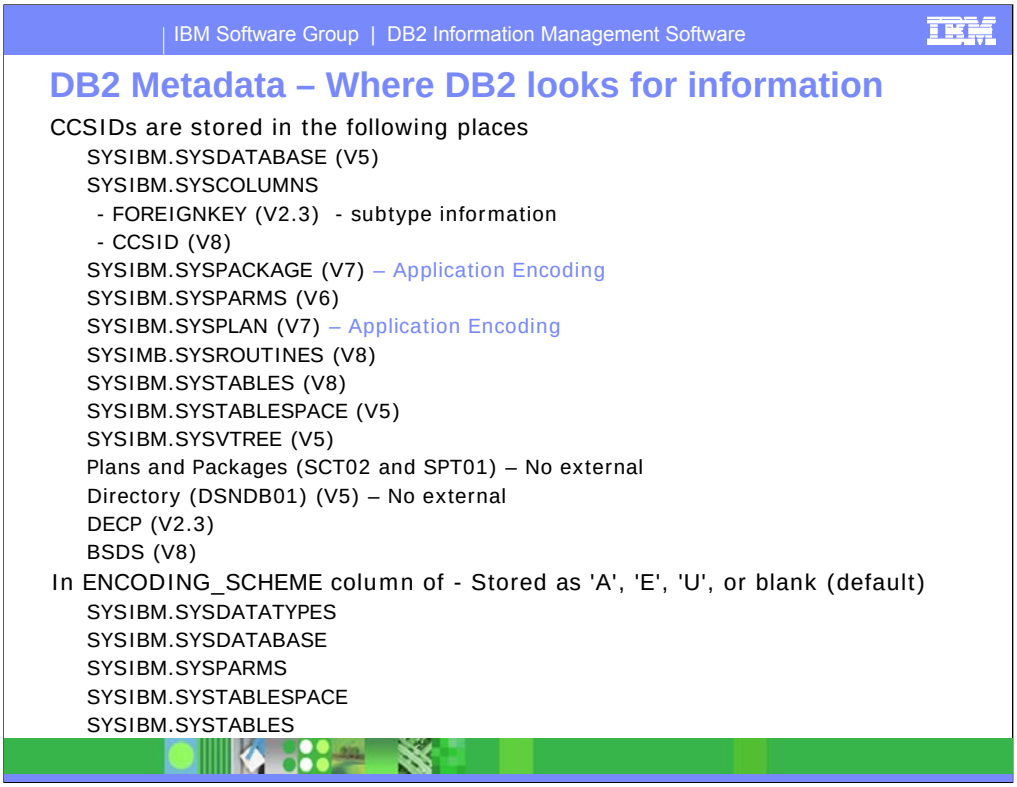
```
DSNTIPF INSTALL DB2 -APPLICATION PROGRAMMING DEFAULTS PANEL 1
===>_
Enter data below:
1  LANGUAGE DEFAULT      ===> IBMCOB   ASM,C,CPP,COBOL,COB2,IBMCOB,FORTRAN,PLI
2  DECIMAL POINT IS      ===> .        .or ,
3  MINIMUM DIVIDE SCALE  ===> NO       NO or YES for a minimum of 3 digits
   to right of decimal after division
4  STRING DELIMITER      ===> DEFAULT  DEFAULT,"or '(COBOL or COB2 only)
5  SQL STRING DELIMITER  ===> DEFAULT  DEFAULT,"or '
6  DIST SQL STR DELIMTR  ===> '        'or "
7  MIXED DATA            ===> NO       NO or YES for mixed DBCS data
8  EBCDIC CCSID          ===> 0        CCSID of your SBCS or MIXED DATA
9  ASCII CCSID           ===> 0        CCSID of SBCS or mixed data.
10 Unicode CCSID         ===> 1208     CCSID of Unicode UTF-8 data.
11 DEF ENCODING SCHEME   ===> EBCDIC   EBCDIC, ASCII, or UNICODE
12 LOCALE LC_CTYPE       ===>
13 APPLICATION ENCODING  ===> EBCDIC   EBCDIC, ASCII, UNICODE ccsid (1-65533)
14 DECIMAL ARITHMETIC    ===> DEC15    DEC15,DEC31,15,31
15 USE FOR DYNAMICRULES  ===> YES      YES or NO
16 DESCRIBE FOR STATIC   ===> NO       Allow DESCRIBE for STATIC SQL.NO or YES.
```


Install panel DSNTIPF is used to specify CCSID information
Options 8,9, and 10 are where the CCSIDs for the three encoding schemes are specified.

Notice that ASCII and EBCDIC CCSIDs are initialized to 0 and the Unicode CCSID is initialized to 1208
The ASCII and EBCDIC CCSIDs are not pre-filled, these values needs to be set by the customer.
The EBCDIC should be set to the CCSID that the customer's 3270 emulators, CICS, and IMS transactions use.

The ASCII value should be set to the CCSID that is most commonly used by workstations in the customer shop (1252 for example).
The Unicode value is pre-filled with 1208 cannot be changed.
This value specifies the mixed CCSID for Unicode tables.
Other things to note on this page
Option 11 - This specifies the default encoding scheme for Objects created in the DB2 subsystem.

Option 13 - This option specifies the default application encoding. Changing this valuse should be done with great care.



Once we've specified CCSIDs for our system, what does DB2 do with them?
DB2 stores CCSIDs in the Catalog, the directory, in bound statements, the directory, and of course the DECP
The value stored in these areas depends on what release of DB2 was used to create the object, and the value in the DECP at the time the object was created.
If a value is 0, it is assumed that the object is EBCDIC.
In general, DB2 does not support changing of a CCSID once it is specified in a DECP
The exceptions are
Changing from 0 to a valid value

Changing from a CCSID that does not support the EURO symbol to a CCSID that supports the EURO symbol (37 -> 1140 for instance).
Note that this sort of change requires special, disruptive, changes and should be undertaken only after the documentation has been read and the process is thoroughly understood.
Encoding enformation is also stored in some catalog tables


IBM Software Group
When Does DB2 Convert?
DB2 Information Management Software
business on demand software

When does conversion occur?

Local

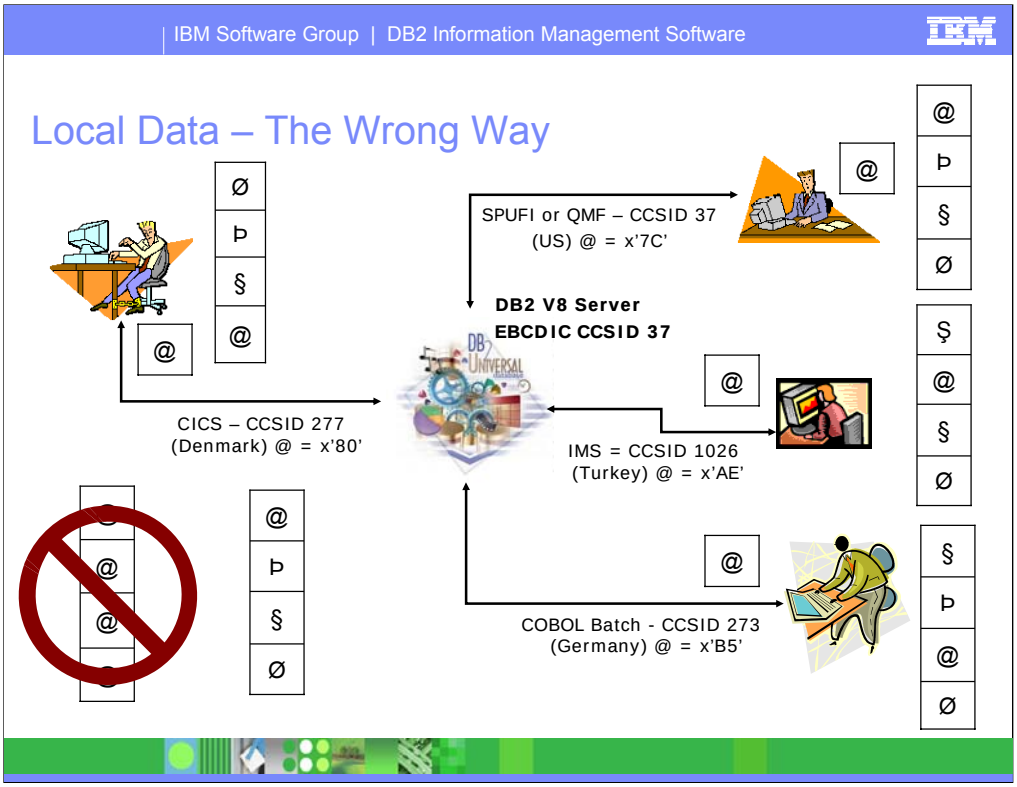
- Generally, conversion does not occur for local applications
- When dealing with ASCII/Unicode tables
- When specified by application
  - CCSID Override in SQLDA (V2.3)
  - Declare Variable (V7)
  - Application Encoding Bind Option (V7)
  - Current Application Encoding Special Register (V7)
- ODBC/JDBC/SQLJ

Remote

- Automatically when needed
  - DRDA Receiver Makes Right

There are no hard and fast rules as to when a conversion occurs. The short answer is that conversion occurs when necessary.

Some of the cases when we do conversion are listed here.



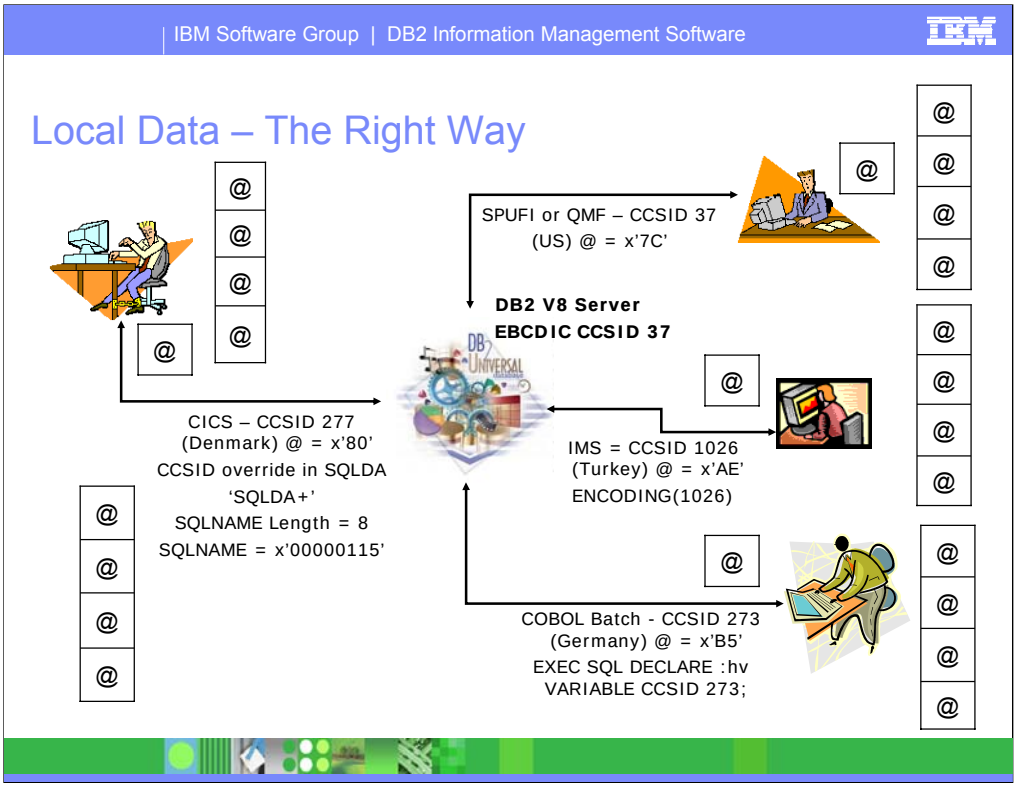
In this example, assume that all the clients are referencing an EBCDIC table (CCSID 37 in this case). Each of the local applications is attempting to insert an "@" into a DB2 table. In general, the results will be incorrect unless DB2 "knows" that the data coming from the CICS, IMS, and COBOL applications is not CCSID 37. There are various ways of the application to tell DB2 the CCSID of the data that they are giving DB2 (in input host variables) or they want to receive from DB2 (in output host variables). These are:

-CCSID overrides in the SQLDA – when using a descriptor

-DECLARE VARIABE – precompiler directive

-ENCODING bind option

-APPLICATION ENCODNG special register

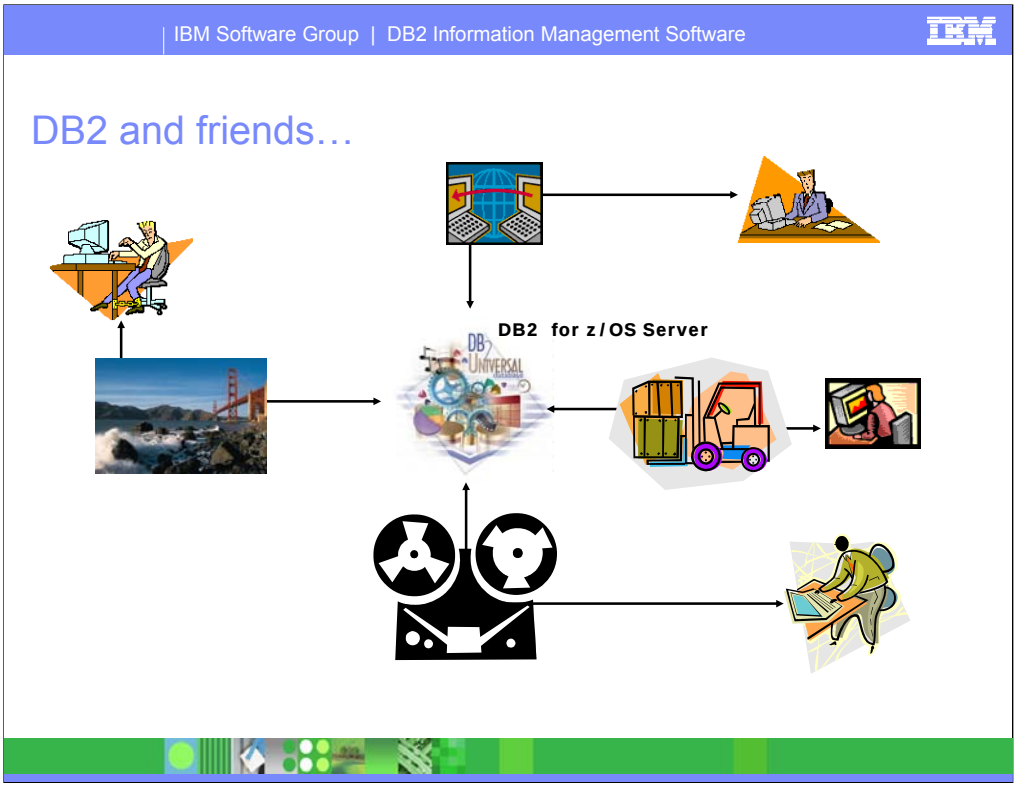


In this example, assume that all the clients are referencing an EBCDIC table (CCSID 37 in this case). Each of the local applications is attempting to insert an "@" into a DB2 table. In general, the results will be incorrect unless DB2 "knows" that the data coming from the CICS, IMS, and COBOL applications is not CCSID 37.

In this case:

- The ENCODING bind option is used to let DB2 know the CCSID of the application inserting the data from Turkey.
- DECLARE VARIABLE is used by the batch job inserting data from Germany
- CCSID overrides are used in the CICS application inserting data from Denmark

The result is that the data is inserted correctly, and when each application fetches the data that was just inserted, it is represented correctly (each client sees the correct data).



This picture is showing that data does not have to come into DB2 through a terminal emulator to cause problems.

All character data has a CCSID. That CCSID may be implicit or it may be explicit, but there is a CCSID for the data.

In this case, you might have data coming through a (golden) gateway, a tape, electronic computer to computer transfer (such as FTP), or LOAD. In every case there is a CCSID associated with the data and if DB2 does not know what the CCSID is, it may treat the data incorrectly.

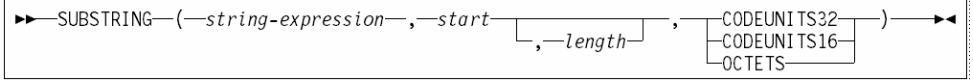

IBM Software Group
Character Based Functions
DB2 Information Management Software
business on demand software

Character Based Functions (PQ88784)

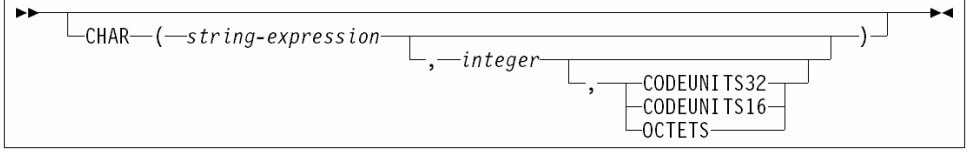
- New functions
  - CHARACTER_LENGTH
  - POSITION
  - SUBSTRING
- Updated Functions
  - CHAR
  - CLOB
  - DBCLOB
  - GRAPHIC
  - INSERT
  - LEFT
  - LOCATE
  - RIGHT
  - VARCHAR
  - VARGRAPHIC
- CAST Specification
  - Changes to enable specification of Code Units

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:

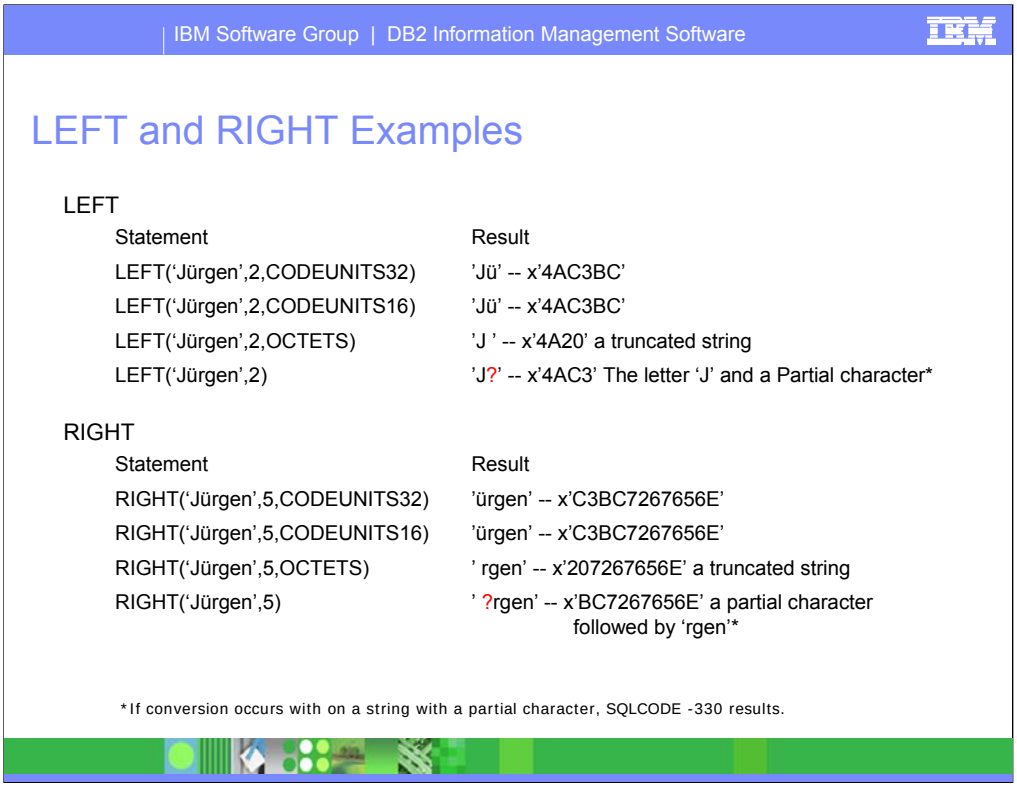
```
SELECT CHARACTER_LENGTH(NAME,OCTETS)
FROM T1 WHERE NAME = 'Jürgen';
```

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'

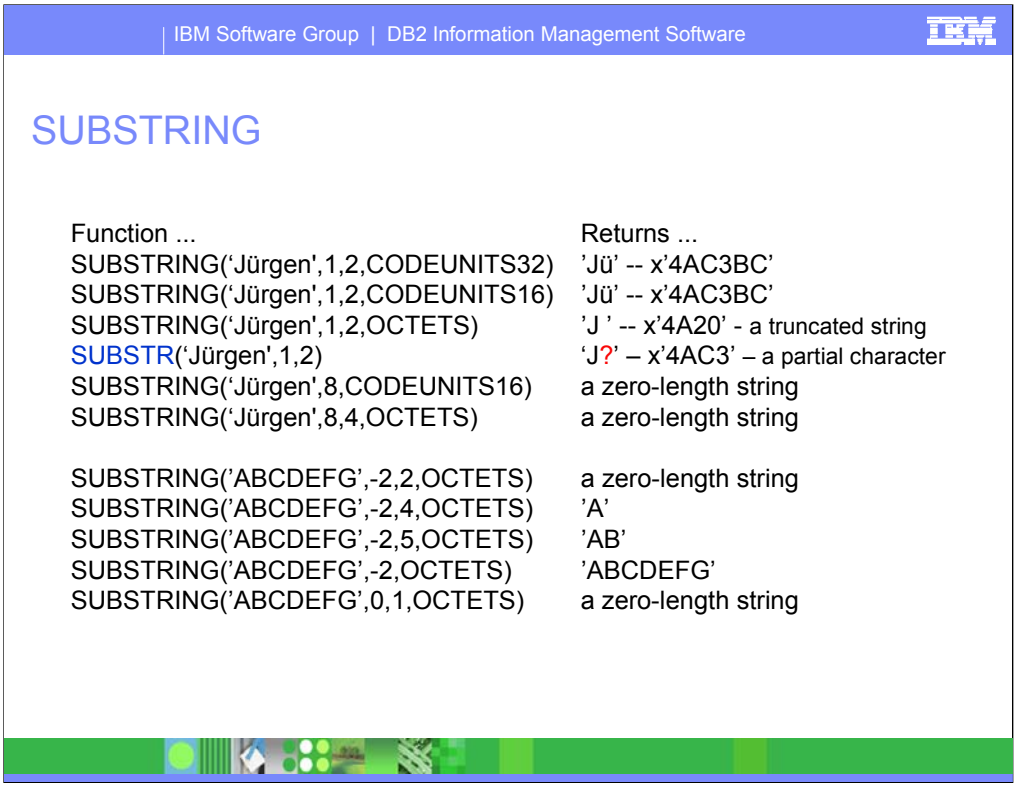


LEFT and RIGHT operate in a similar manner.

In these examples, CODEUNITS32 and CODEUNITS16 return the same results.

When OCTETS is specified, an a partial character would result, the PAD character is used.

When no units are specified, the behavior is similar to that of OCTETS (byte based), but partial characters can result.



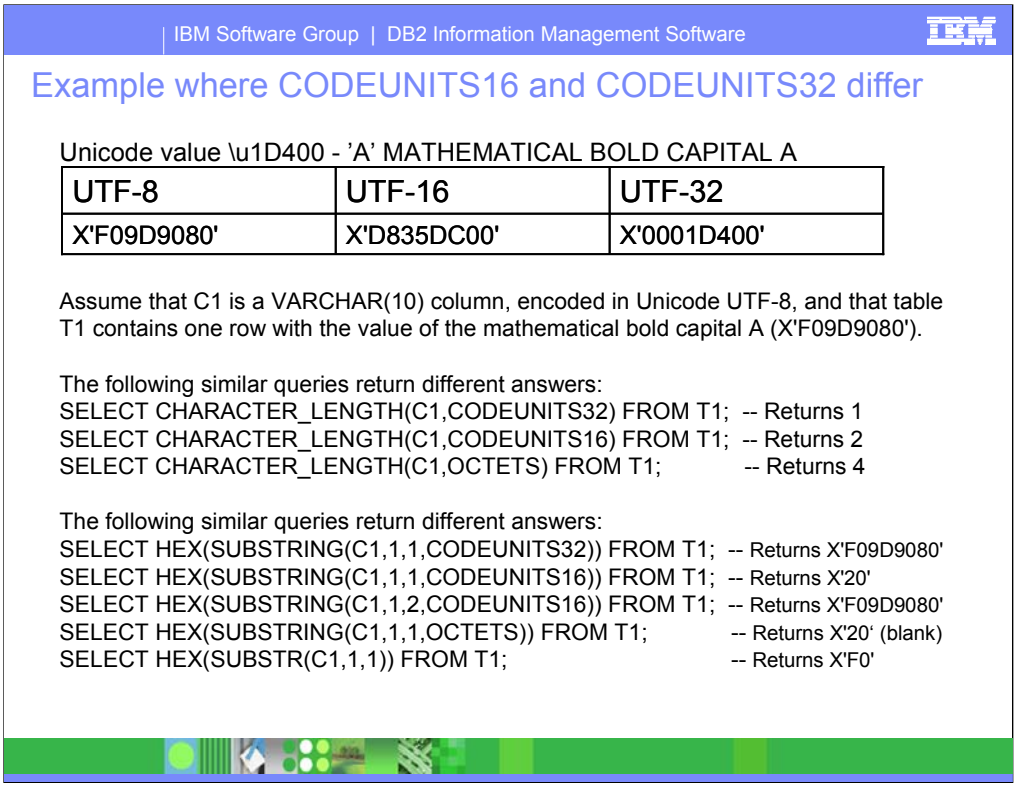
Let C be the value of the *string-expression*, let LC be the length in characters of C, and let S be the value of the *start*. v If *length* is

The first and second examples return the same value because even though 'ü' is a multi-byte character in UTF-8, it is one CODEUNITS32 or CODEUNITS16 codepoint.

In the third example OCTETS is specified and since a partial character would have resulted, truncation occurs on the partial character and a pad character is returned in place of the partial character.

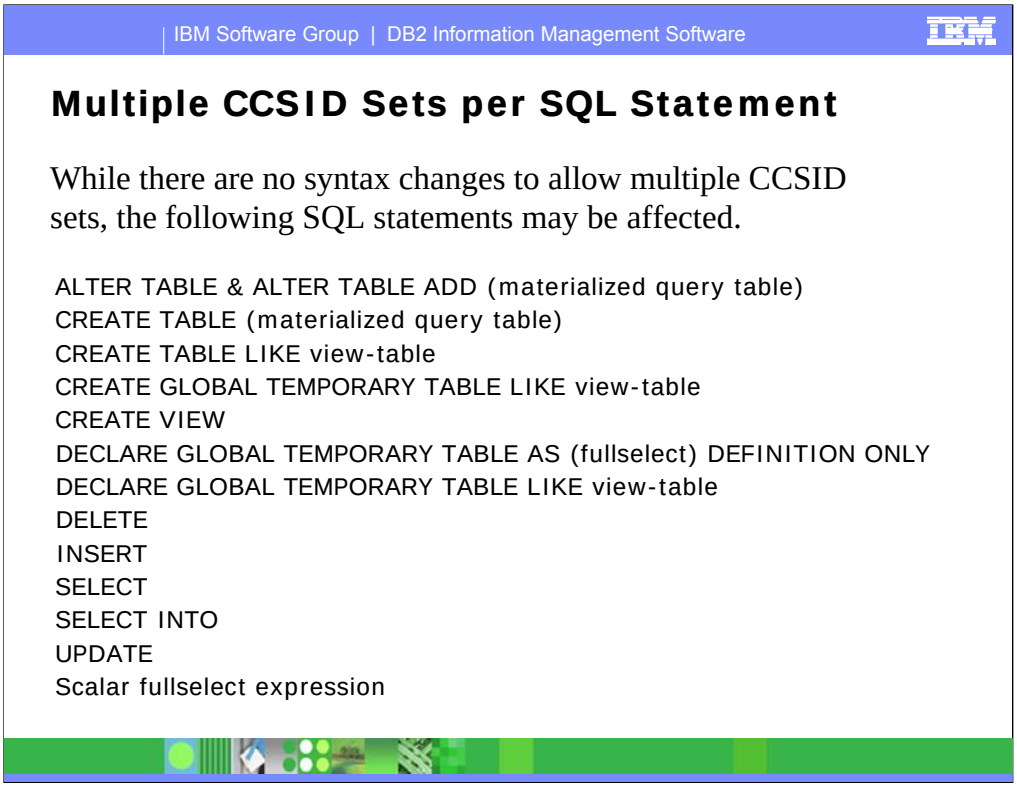
In the fourth example, we are using SUBSTR, not SUBSTRING. SUBSTR is byte based and will return partial characters. As mentioned previously if conversion occurs on a string with a partial character, -330 will result.

specified, let L be the value of <string length> and let E be S+L. Otherwise, let E be the larger of LC + 1 and S. v If either C, S, or L is the null value, the result of the function is the null value. v If E is less than S, an exception condition is raised: data exception — substring error. v Otherwise: – If S is greater than LC or if E is less than 1 (one), the result of the function is a zero-length string. – Otherwise: - Let S1 be the larger of S and 1 (one). Let E1 be the smaller of E and LC+1. Let L1 be E1–S1. - The result of the function is a character string that contains the L1 characters of C starting at character number S1 in the same order that the characters appear in C.

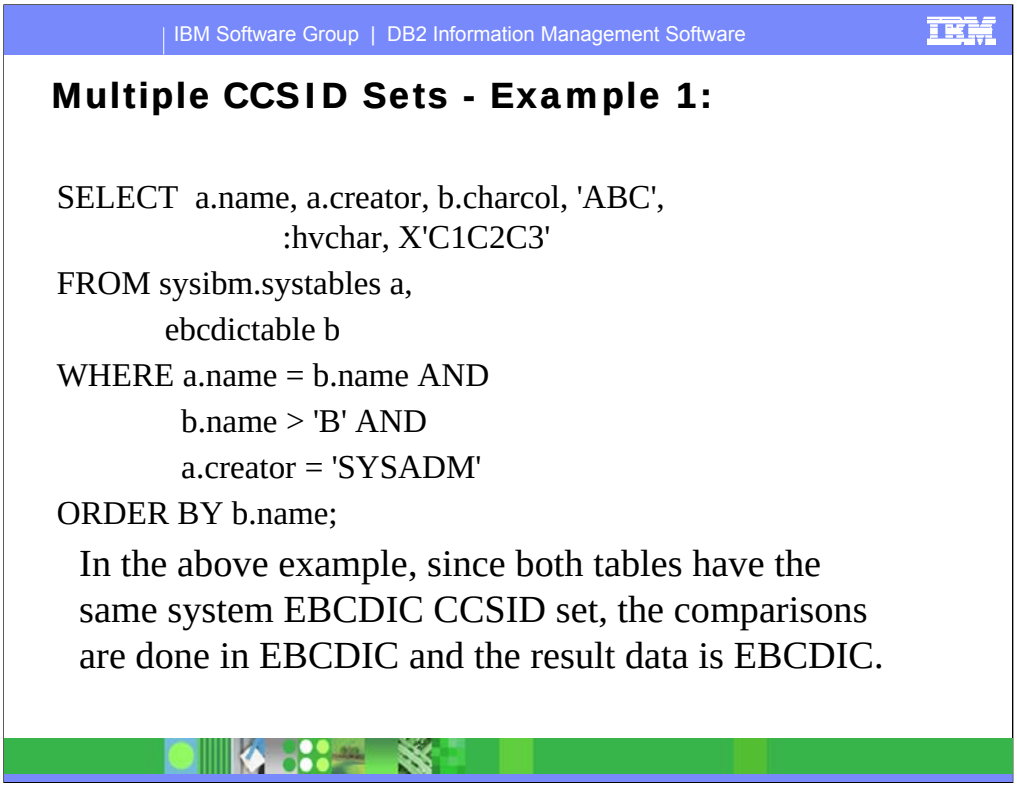


For many characters, specifying CODEUNITS16 and CODEUNITS32 on a BIF does not matter. However, when supplementary characters (sometimes also called surrogate characters) are used, the results can be different.


IBM Software Group
Queries with more than one encoding
DB2 Information Management Software
business on demand software



The statements listed above may have their behavior affected when data from more than one encoding is processed as part of the execution of the statement



To maintain compatibility with previous releases, statements that do not reference objects with more than one CCSID set, will continue to use the old rules.

- For compatibility with prior releases:
  - IF an SQL statement which references table objects with **only one CCSID set**
  - THEN **the results** will continue to have the same **result encoding scheme (CCSID set) as the table objects** and basic semantics.
- All string objects and special registers in the SQL statement are converted to this result encoding scheme (CCSID set) vs. using the application encoding scheme like multiple CCSID set SQL statements.

In this example, all the tables reference a single encoding scheme.

The results of this query will be EBCDIC.

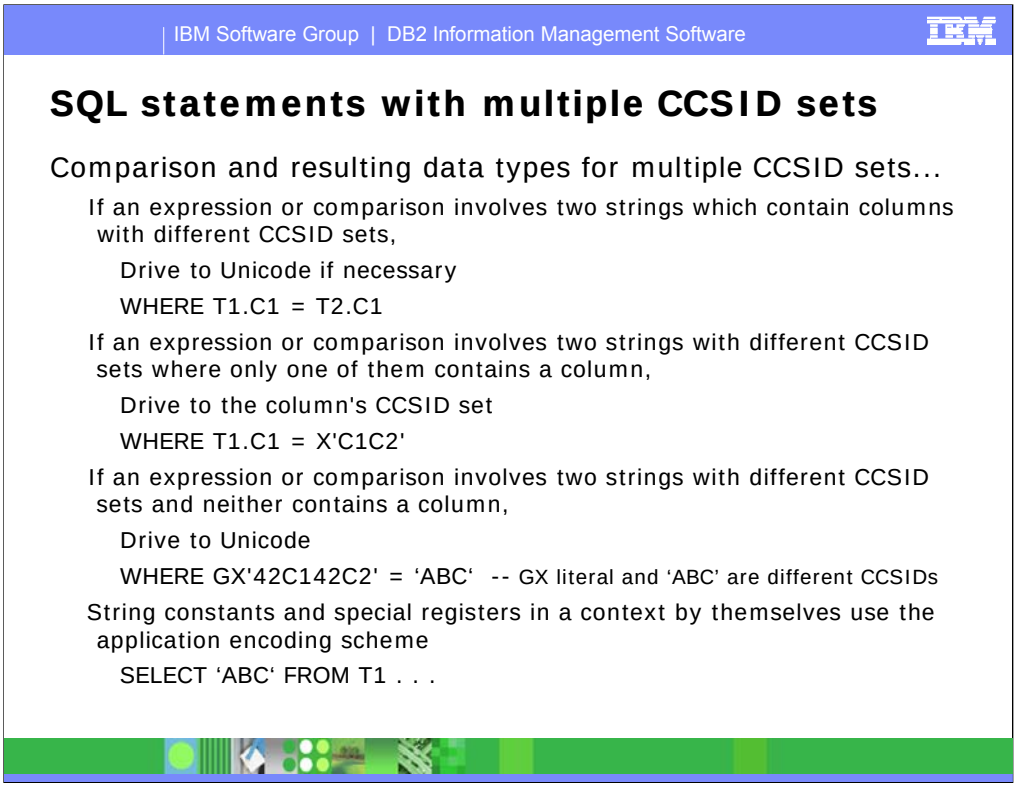
Multiple CCSID Sets - Example 1 (continued)

```
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.

In this example, where we have a Unicode catalog, some columns are EBCDIC and others are Unicode. Literal values are interpreted using the Application Encoding option in effect.
Queries that cross encoding schemes are allowed once ENFM (Enabling New Function Mode) is entered. This is to ensure that applications that access the catalog continue to function.



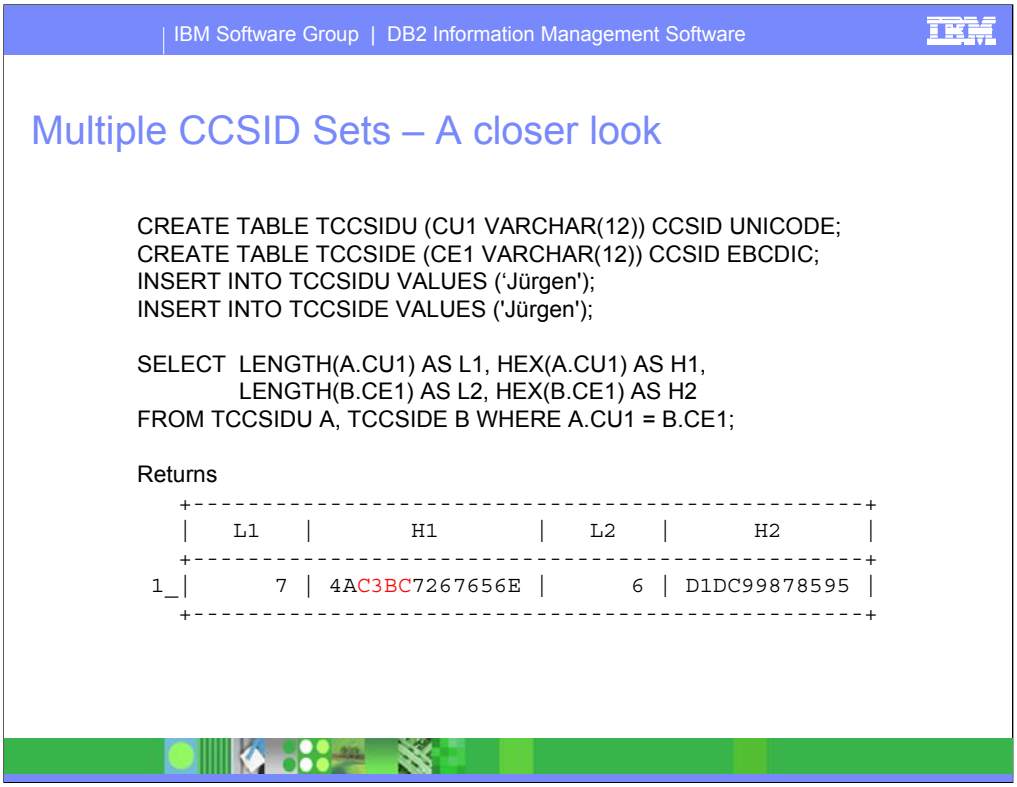
Here are some examples of how the new DB2 rules for multiple CCSID sets work

-In the first case we have two columns being compared – in this case, drive to Unicode if they are not already Unicode

-In the second case, we drive to the column CCSID set

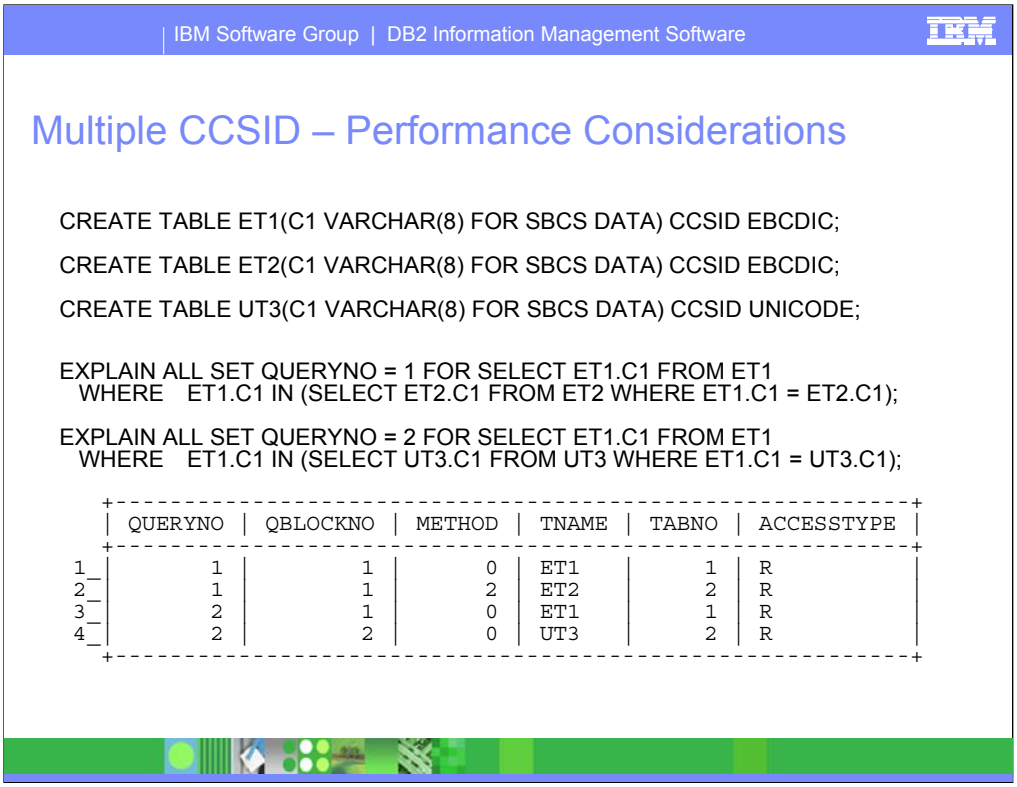
-In the third case, neither side is a column, so drive to Unicode

-In the fourth case we use the APPLICATION ENCODING SCHEME – the thought here is that the application is providing the string, we should interpret the string in the same encoding as the application.



In this example, you can see that up until the data is returned to the application, the data from TCCSIDU is Unicode, and the data from TCCSIDE is EBCDIC. At the point the data is returned to the application (copied from DB2 to the applications host variables), conversion will be performed to the CCSID specified by the application (either implicitly or explicitly). In this case, since we are returning the HEX representation of the data, we really don’t convert the data, we convert the hex representation of the data (which will always be composed of characters between 0-9 and A-F).

The result is the same for all Join methods. For Sort Merge Join (SMJ), we convert the TCCSIDE.CE1 to Unicode before sorting. In the sort workfile, we carry both the EBCIDC version of the string (to return to the application), and the Unicode version of the string (for comparison processing in the Join). As a result, the size of workfiles can increase when EBCDIC and Unicode tables are joined (as compared to joining two EBCDIC or two Unicode tables). Additionally, it is possible that SQLCODE -136 or -670 may be issued.



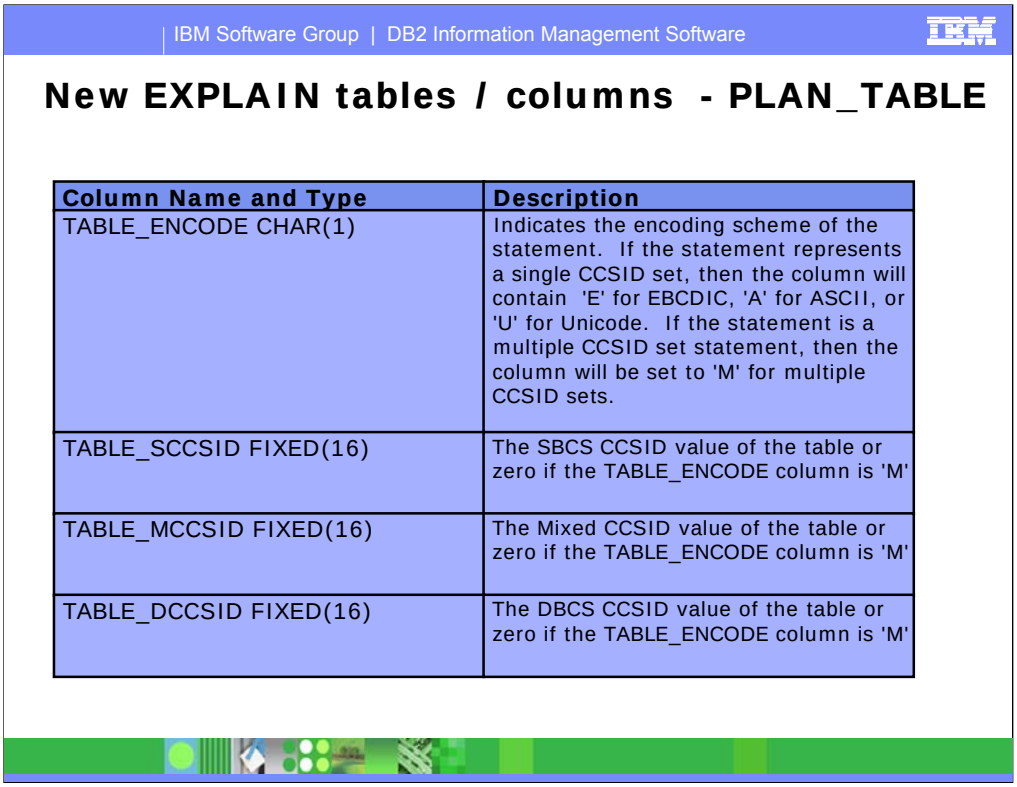
In this example, we have three tables ET1 and ET2 are EBCDIC tables, and UT3 is a Unicode table.

The two queries are identical, except the second query has UT3 where the first query has ET2.

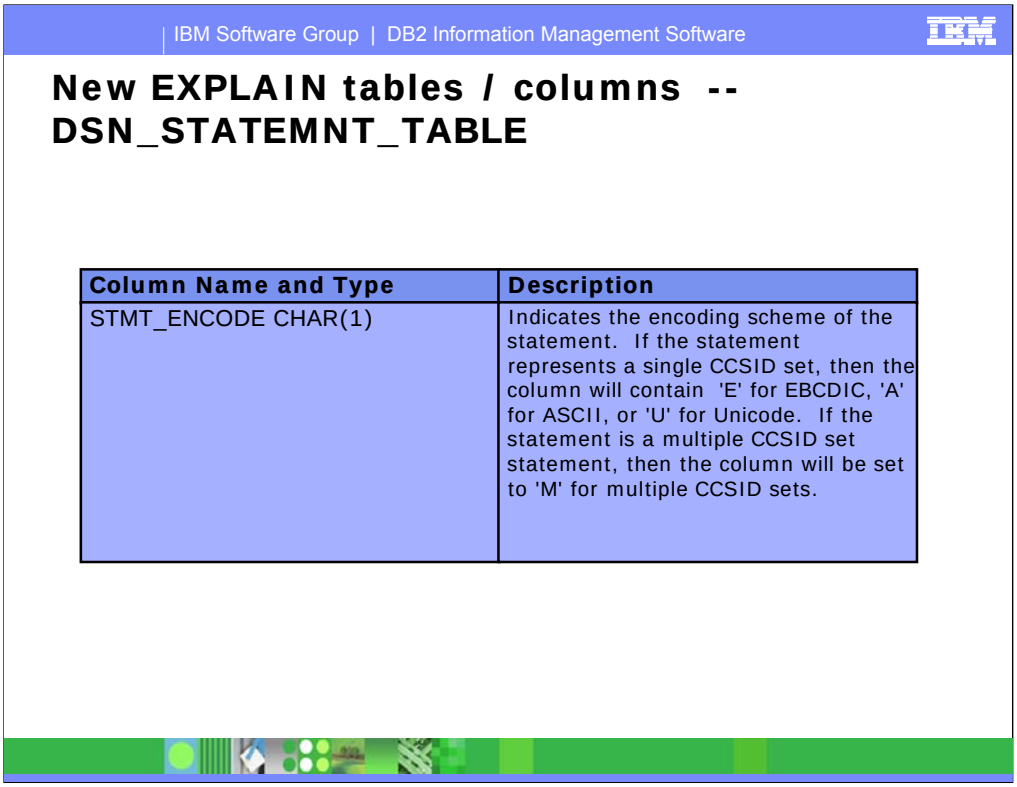
Note that table UT3 isn't really even using Unicode data (it is defined as "FOR SBCS DATA" which means that the data will be 7-bit ASCII).

When two EBCDIC tables are involved, as in QUERYNO=1, we are able to transform the query into a join. In the second example, the transformation to a join did not take place, as a result the second query may perform slower than the first query.

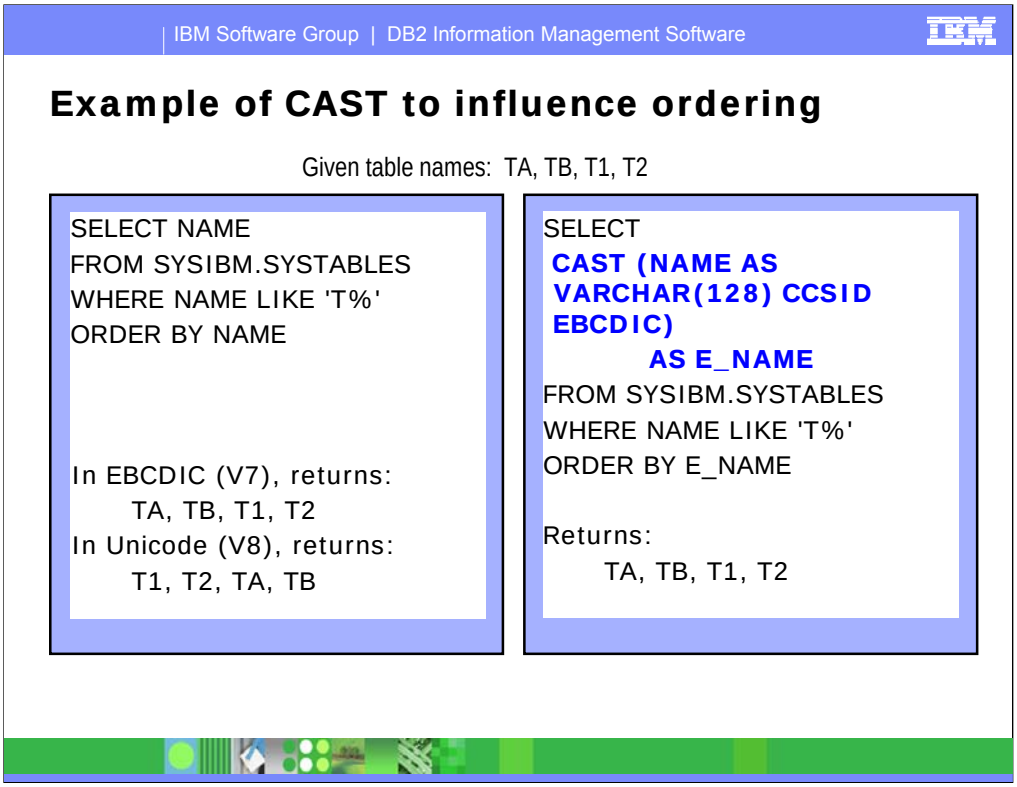
This is a very simple example that demonstrates the ramifications to performance of multiple CCSID statements.



The table above shows the changes to the PLAN_TABLE to support multiple CCSIDs in a statement

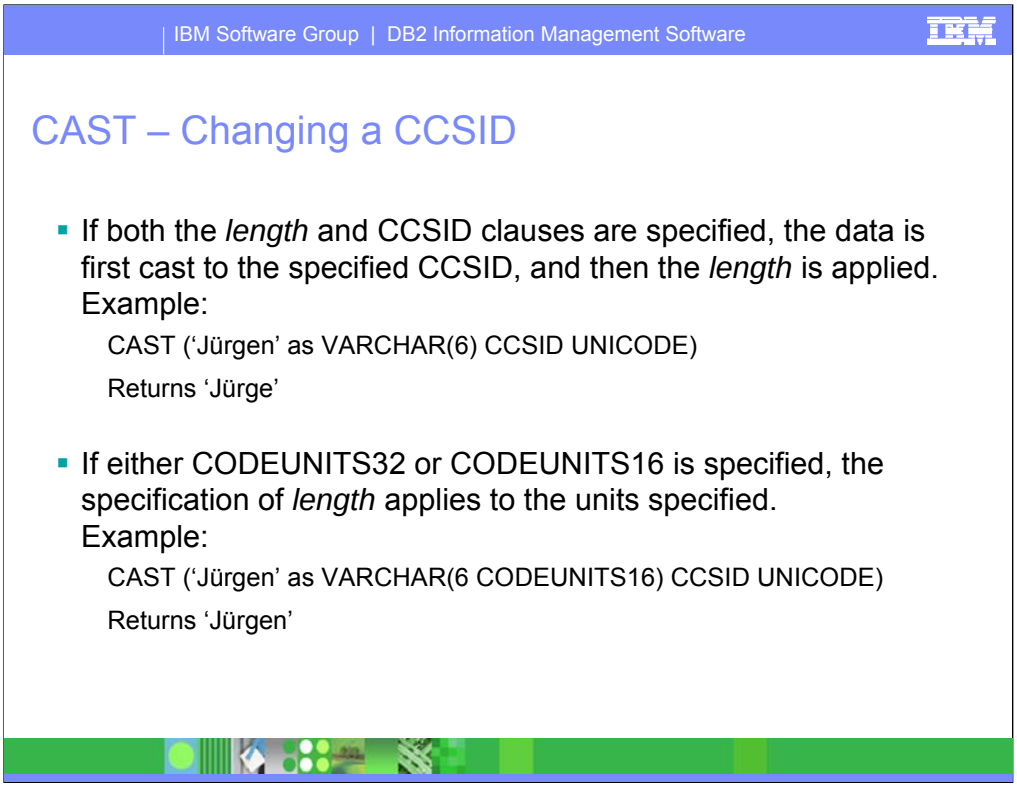


The table above shows the changes to the DSN_STATEMNT_TABLE to support multiple CCSIDs in a statement



The new CAST specification can be used to influence ordering.

The above example shows how SQL could be recoded to cause data to be returned in the same order as DB2 V7 would.



When a length is specified as part of a CAST that also specifies a CCSID, the length is applied after the data has been converted to the target CCSID, unless CODEUNITS16 or CODEUNITS32 is specified. If CODEUNITS16 or CODEUNITES32 is specified, then the length applies in the specified units, then the data is cast to the final CCSID.


IBM Software Group
Putting it all together - Examples
DB2 Information Management Software
business on demand software

Sample Schema - EBCDIC

```
CREATE TABLE DSN8XX0.EMP
 (EMPNO     CHAR(6)                      NOT NULL,
    FIRSTNME  VARCHAR(12)                 NOT NULL,
    MIDINIT   CHAR(1)                     NOT NULL,
    LASTNAME  VARCHAR(15)                 NOT NULL,
    WORKDEPT  CHAR(3)                                ,
    PHONENO   CHAR(4) CONSTRAINT NUMBER CHECK
     (PHONENO >= '0000' AND PHONENO <= '9999')   ,
    HIREDATE  DATE                                   ,
    JOB       CHAR(8)                                ,
    EDLEVEL   SMALLINT                               ,
    SEX       CHAR(1)                                ,
    BIRTHDATE DATE                                   ,
    SALARY    DECIMAL(9, 2)                          ,
    BONUS     DECIMAL(9, 2)                          ,
    COMM      DECIMAL(9, 2)                          ,
    PRIMARY KEY(EMPNO))
  CCSID EBCDIC;
```

Sample Schema – Unicode UTF-8 and SBCS

```
CREATE TABLE DSN8YY0.EMP
 (EMPNO     CHAR(6) FOR SBCS DATA             NOT NULL,
    FIRSTNME  VARCHAR(36)                      NOT NULL,
    MIDINIT   VARCHAR(3)                       NOT NULL,
    LASTNAME  VARCHAR(45)                      NOT NULL,
    WORKDEPT  CHAR(3) FOR SBCS DATA                    ,
    PHONENO   CHAR(4) FOR SBCS DATA CONSTRAINT NUMBER CHECK
     (PHONENO >= '0000' AND PHONENO <= '9999')         ,
    HIREDATE  DATE                                     ,
    JOB       CHAR(9)FOR SBCS DATA                     ,
    EDLEVEL   SMALLINT                                 ,
    SEX       CHAR(1)FOR SBCS DATA                     ,
    BIRTHDATE DATE                                     ,
    SALARY    DECIMAL(9, 2)                            ,
    BONUS     DECIMAL(9, 2)                            ,
    COMM      DECIMAL(9, 2)                            ,
    PRIMARY KEY(EMPNO))
CCSID UNICODE;
```

Sample Schema – Unicode UTF-16

```
CREATE TABLE DSN8ZZ0.EMP
  (EMPNO     GRAPHIC(6)                          NOT NULL,
     FIRSTNME  VARGRAPHIC(12)                     NOT NULL,
     MIDINIT   GRAPHIC(1)                         NOT NULL,
     LASTNAME  VARGRAPHIC(15)                     NOT NULL,
     WORKDEPT  VARGRAPHIC(3)                                ,
     PHONENO   VARGRAPHIC(4) CONSTRAINT NUMBER CHECK
      (PHONENO >= '0000' AND PHONENO <= '9999')           ,
     HIREDATE  DATE                                       ,
     JOB       VARGRAPHIC(8)                              ,
     EDLEVEL   SMALLINT                                   ,
     SEX       VARGRAPHIC(1)                              ,
     BIRTHDATE DATE                                       ,
     SALARY    DECIMAL(9, 2)                              ,
     BONUS     DECIMAL(9, 2)                              ,
     COMM      DECIMAL(9, 2)                              ,
     PRIMARY KEY(EMPNO))
   CCSID UNICODE;
```

Sample Schema – Unicode UTF16 and SBCS

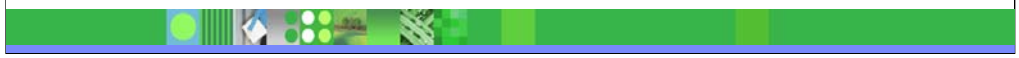
```
CREATE TABLE DSN8AA0.EMP
  (EMPNO     CHAR(6) FOR SBCS DATA              NOT NULL,
     FIRSTNME  VARGRAPHIC(12)                    NOT NULL,
     MIDINIT   VARGRAPHIC(1)                     NOT NULL,
     LASTNAME  VARGRAPHIC(15)                    NOT NULL,
     WORKDEPT  CHAR(3) FOR SBCS DATA                     ,
     PHONENO   CHAR(4) FOR SBCS DATA CONSTRAINT NUMBER CHECK
      (PHONENO >= '0000' AND PHONENO <= '9999')          ,
     HIREDATE  DATE                                      ,
     JOB       CHAR(8) FOR SBCS DATA                     ,
     EDLEVEL   SMALLINT                                  ,
     SEX       CHAR(1) FOR SBCS DATA                     ,
     BIRTHDATE DATE                                      ,
     SALARY    DECIMAL(9, 2)                             ,
     BONUS     DECIMAL(9, 2)                             ,
     COMM      DECIMAL(9, 2)                             ,
     PRIMARY KEY(EMPNO))
   CCSID UNICODE;
```

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

```
DECLARE 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.
```



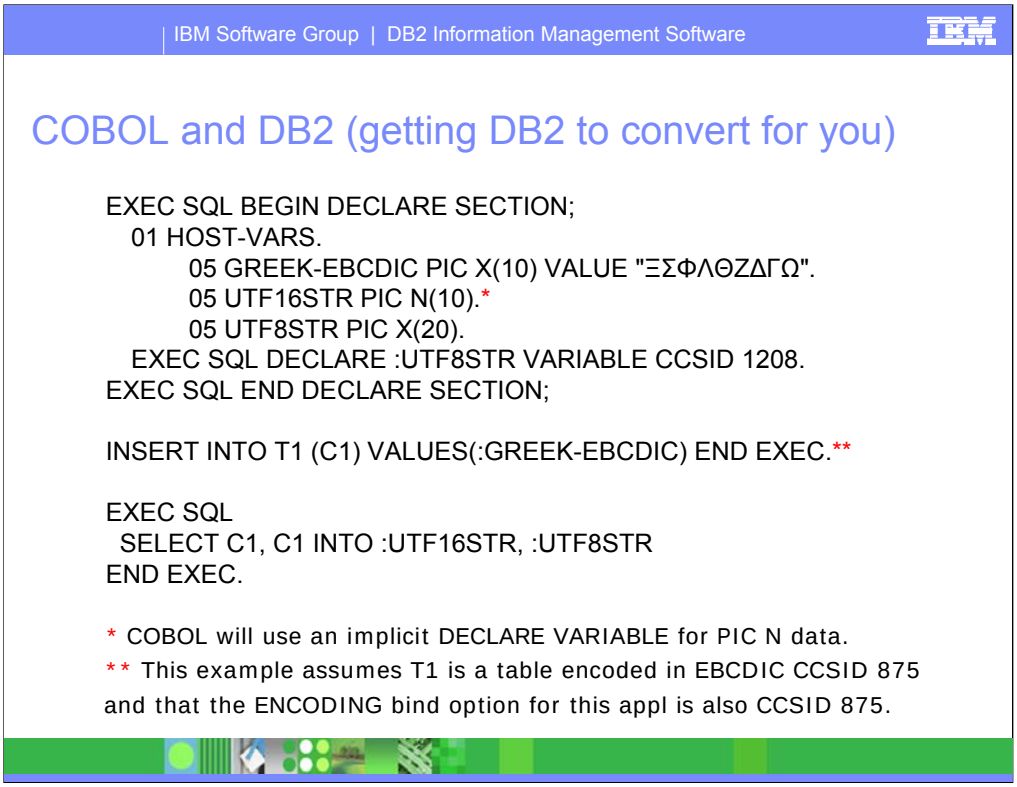
Cobol has recently added support for Unicode characters

Included in this support

-New NATIONAL data type

-N and NX literals

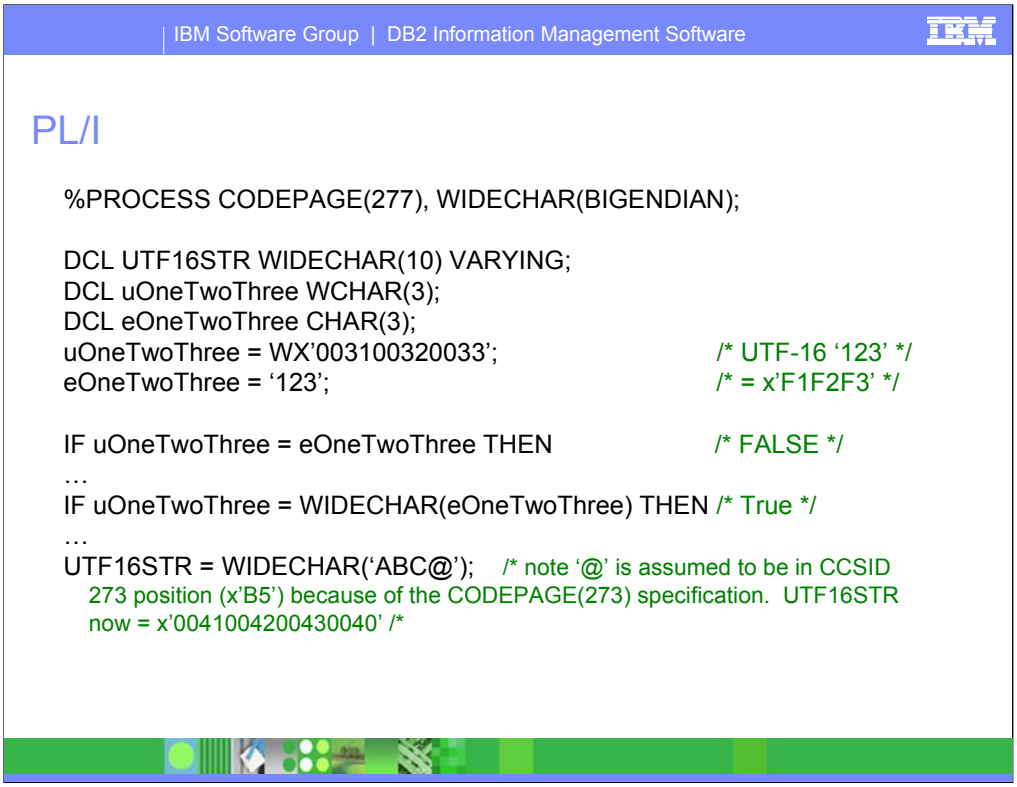
-Conversion operations



Using the previous example as a starting point, it is also possible to get DB2 to perform the needed conversions for you.

In this example, the EBCDIC 875 data is INSERTed into an EBCDIC CCSID 875 table. Then the COBOL application fetches the data into the UTF-16 and UTF-8 host variables.

Note that only one DECLARE VARIABLE statement is used needed because PIC N data is implicitly UTF-16, and the PIC X data is assumed to be in the CCSID specified by the ENCODING bind option.



PL/I has some support for Unicode UTF-16 data.

PL/I uses the WIDECHAR datatype to support UTF-16.

PL/I supports UTF-16 data as either BIG or LITTLE ENDIAN. For compatibility with DB2, BIGENDIAN data should be used.

PL/I has basic conversion support via the WIDECHAR function. This function does not use the z/OS conversion services at this time.

PL/I does not have any support for a UTF-16 literal, but does offer a WX literal which can be used to specify a UTF-16 codepoint (like the UX constant would be used in DB2).

PL/I Example with USING DESCRIPTOR

```
…
   DCL STMT1 CHAR(100) VARYING INIT('INSERT INTO T1 VALUES (?,?) ');

   DCL DA1 CHAR(16+(2*44));     /* ALLOCATE SPACE FOR 2 SQLDA ENTRIES */

   EXEC SQL  INCLUDE SQLDA;
   SQLDAID = 'SQLDA+  ';                          /* Note the "+"          */
   SQLN = 2;                                      /* Allocated SQLVARS     */
   SQLD = 2;                                      /* Used SQLVARS          */
   SQLVAR(1).SQLTYPE = 468;                       /* Graphic – not null    */
   SQLVAR(1).SQLLEN1 = 3;                         /* Length = 6 bytes      */
   SQLVAR(1).SQLDATA = ADDR(uOneTwoThree);        /* Address of host var   */
   SQLVAR(1).SQLNAME = '0000048000000000'X;       /* CCSID 1200            */
   SQLVAR(2).SQLTYPE = 452;                       /* Character – not null  */
   SQLVAR(2).SQLLEN1 = 3;                         /* Length = 3 bytes      */
   SQLVAR(2).SQLDATA = ADDR(eOneTwoThree);        /* Address of host var   */
   SQLVAR(2).SQLNAME = '0000011100000000'X;       /* CCSID 273             */

   EXEC SQL PREPARE S1 FROM :STMT1;               /* Prepare Statement     */

   EXEC SQL EXECUTE S1 USING DESCRIPTOR :SQLDA;  /* Execute Stmt          */
…
```

This code snippet shows the code needed to use a descriptor (SQLDA) with a dynamic INSERT statement.

This is one example of how to perform this sort of programming and is intended to demonstrate the techniques associated with using a descriptor and setting CCSID values with a dynamic INSERT statement.

Conversion Considerations – Application –vs- DB2

- In many cases, conversion can be handled by the application or DB2 (as shown in previous slides).
- Conversion by DB2
  - Pro
    - Less application code
  - Con
    - Performance impact may be greater than just conversion cost
- Conversion by Application
  - Pro
    - May perform better in some cases
  - Con
    - Requires more application code

ODBC/SQLJ/JDBC

V7 ODBC Support

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

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

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

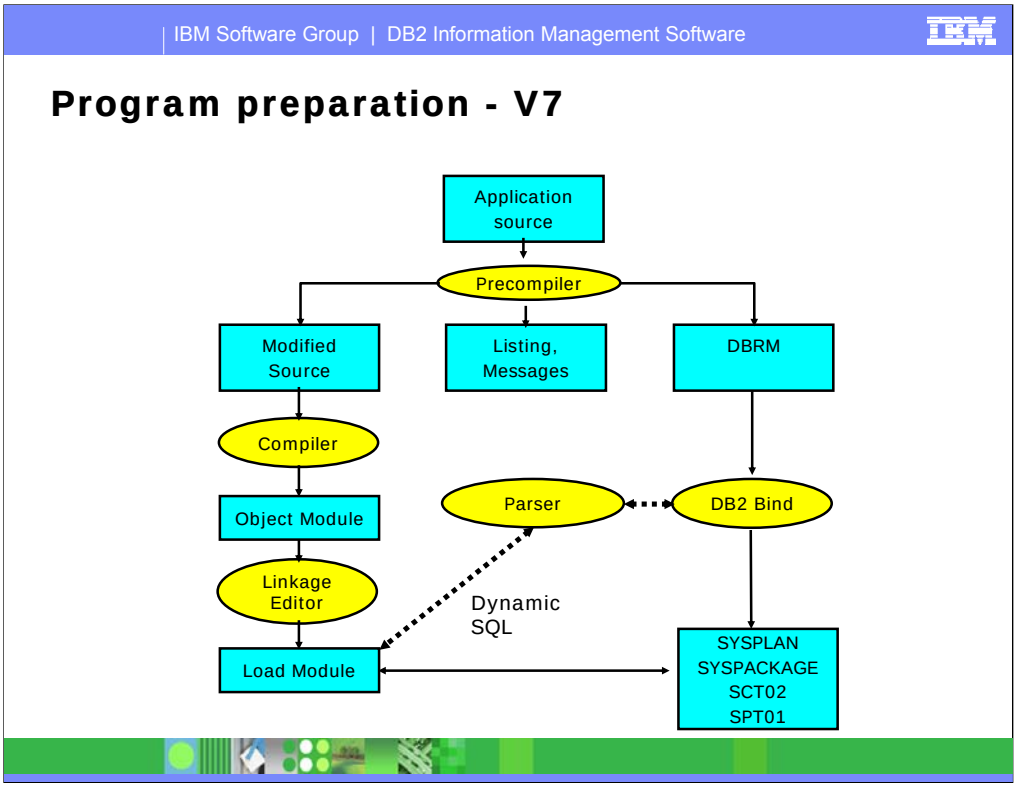
ODBC support for Unicode is included as part of the effort to support ODBC 3.0
In V7, ODBC added support for Wide Character APIs. These Wide Character APIs support Unicode input, but the data is converted to EBCDIC before being passed to DB2
In V8, ODBC added support for specification of processing using the CURRENTAPPENSCH ini file setting Additionally, in V8, the ODBC Wide Character APIs pass data to DB2 in Unicode.

SQLJ and JDBC already support Unicode, but changes have been made to exploit Unicode support in DB2 V7 and DB2 V8.

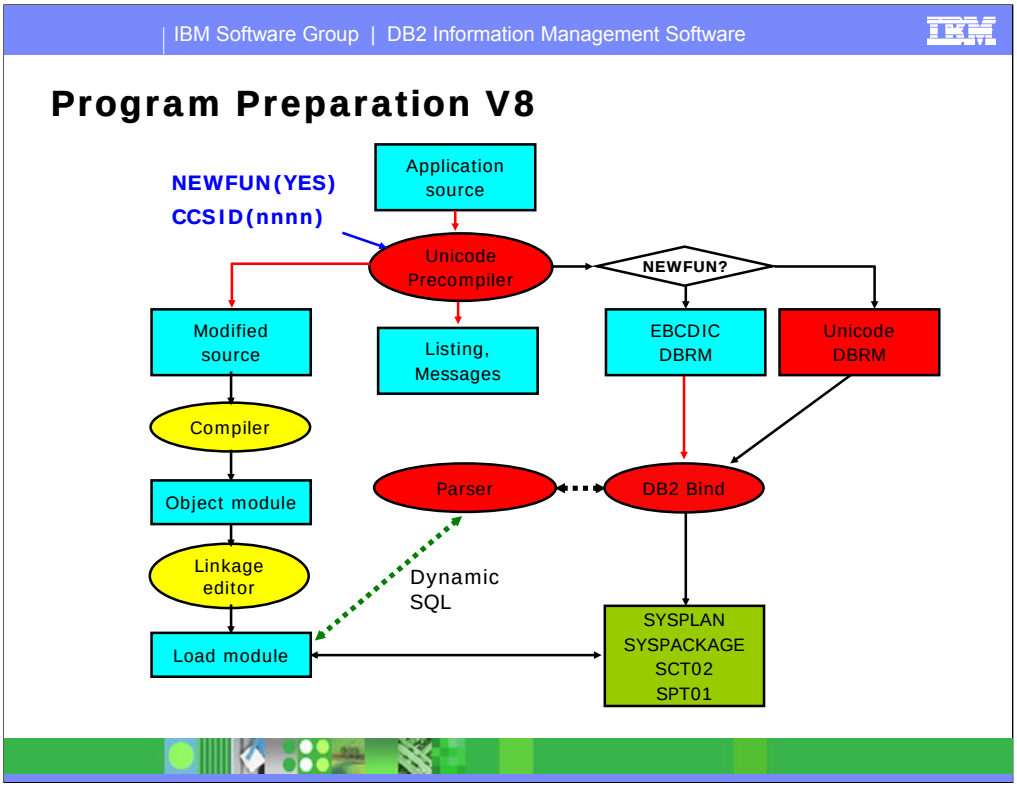
These changes are aimed at reducing the number of conversions that take place, and avoiding the potential for data loss.

SPUFI, DSNTEP2/4, DSNTIAUL (PQ98170 & PK06076)

- Have been modified to support UTF-16 better. The SQLDA from DESCRIBE is changed to accommodate the data type of each graphic column having CCSID 1200 as follows:
  - GRAPHIC -> CHAR
  - VARGRAPHIC and LONG VARGRAPHIC -> VARCHAR - DBCLOB -> CLOB
- During FETCH, DB2 converts all UTF-16 characters that can be mapped to the EBCDIC SBCS CCSID, and otherwise sets SQLCODE 0 with SQLWARNING flag 8 (a character that could not be converted was replaced with a substitute character).



Program preparation in DB2 V7 and below assumes all data is in the SYSTEM EBCDIC CCSID

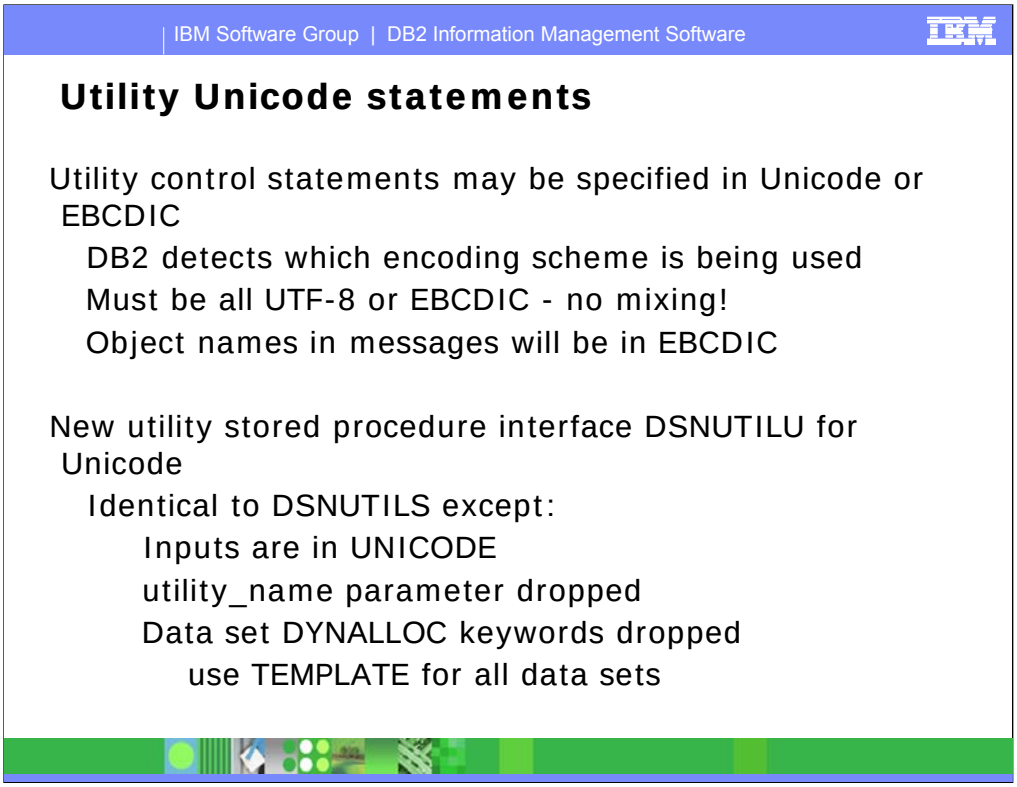


CCSID(nnnn) input parameter to the Unicode Precompiler specifies the CCSID of the application source to ensure proper conversion to unicode for processing. The default value of the CCSID option is the EBCDIC system CCSID as specified on the panel DSNTIPF during installation.

The modified source program (an output of precompilation) remains in its original CCSID. If the DBRM is later bound to a server that does not support UTF-8, the SQL statements are then converted from CCSID 1208 (UTF-8) and sent in the EBCDIC system CCSID.

- NEWFUN(YES)
  - Accept V8 new syntax
  - Unicode DBRM
- NEWFUN(NO)
  - Reject V8 new syntax
  - EBCDIC DBRM
- CCSID(nnnn)
  - It is important to note that this CCSID applies to the source of the application program. It does not necessarily apply to host variables that are used by this application.


IBM Software Group
Utility Support
DB2 Information Management Software
business on demand software



DB2 V8 adds support for Utility control statements in Unicode.

Additionally, a new Unicode Stored Procedure, DSNUTILU is added to compliment DSNUTILS

DSNUTILS is an updated version of DSNTILU that

-Has Unicode parameters

-Has fewer required parameters

-Allocates datasets based on TEMPLATEs

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""""""""""""""""""""
/*
```

If you see what appears to be garbage where a utility statement should be, don't panic. It could be Unicode UTF-8. In V8, Utility control statements can now be in Unicode.

Control statements (in EBCDIC or Unicode) also allow long names (V8)

ISPF

- DISPLAY CCSID ccsid_number [LINE start_line end_line] [COLS start_col end_col]
  - CCSID
    - n
    - UTF8
    - UTF16
    - UTF32
    - UCS2
    - EBCDIC
    - UNICODE
- FIND C'xxxxxx' UTF8
- OA07685 (OA08496 fixes a problem with FIND)
- z/OS 1.4, 1.5, and 1.6

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;
```

Items in green are parms which are not in DSNUTILU. Notice that DSNUTILU specifies CCSID UNICODE for it's character parameters.

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

The load utility has been extended to support conversion to and from Unicode.

Additionally, the load utility will support conversion between character and graphic as long as conversion exists.
Character in load dataset -> Graphic column

Graphic in load dataset -> character column
The unload utility, new for V7, supports conversion to/from Unicode, but does not support conversion between character and graphic

LOAD Example

CREATE TABLE TCCSIDU (CU1 VARCHAR(11)) CCSID UNICODE;

LOAD DATA INDDN SYSREC LOG YES CCSID(0037) INTO TABLE
"USER1"."TCCSIDU" ("CU1" POSITION(00001:00011) CHAR(00011))

Data is
Jürgen
Hegelstraße

```
DSNU000I    DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = LOAD65710
DSNU1044I   DSNUGTIS - PROCESSING SYSIN AS EBCDIC
DSNU050I    DSNUGUTC -  LOAD DATA INDDN SYSREC LOG YES CCSID(37)
DSNU650I  ) DSNURWI -  INTO TABLE "ADMF001"."TCCSIDU"
DSNU650I  ) DSNURWI -   ("CU1" POSITION(1:11) CHAR(11))
DSNU334I  ) DSNURCON - INPUT FIELD 'CU1' INVALID FOR `USER1.TCCSIDU',
  ERROR CODE '19'
DSNU302I  ) DSNURWBF - (RE)LOAD PHASE STATISTICS - NUMBER OF INPUT
  RECORDS PROCESSED=1
DSNU017I    DSNUGBAC - UTILITY DATA BASE SERVICES MEMORY EXECUTION
  ABENDED, REASON=X'00E40323'
```

To prevent this error table should be declared as follows:
CREATE TABLE TCCSIDU (CU1 VARCHAR(33)) CCSID UNICODE;*

*Recommendation based on Table 25 (*Result length of CCSID conversion) in the SQL Ref*

UNLOAD Example

```
CREATE TABLE TCCSIDE (CE1 VARCHAR(11)) CCSID EBCDIC;
INSERT INTO TCCSIDE VALUES ('Jürgen');
INSERT INTO TCCSIDE VALUES ('Hegelstraße');
```

```
UNLOAD CCSID(1208) FROM TABLE USER1.TCCSIDE
```

```
DSNU000I    DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = AOAD65710
DSNU1044I   DSNUGTIS - PROCESSING SYSIN AS EBCDIC
DSNU050I    DSNUGUTC -  UNLOAD CCSID(1208)
DSNU650I  ) DSNUUGMS -    FROM TABLE ADMF001.TCCSIDE
DSNU1233I ) DSNUULVA - DATA IS TOO LONG FOR FIELD CE1, TABLE ADMF001.TCCSIDE
DSNU1219I ) DSNUULVA - THE NUMBER OF RECORDS IN ERROR REACHED THE LIMIT 1
DSNU253I    DSNUUNLD - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=1
  FOR TABLE USER1.TCCSIDE
DSNU252I    DSNUUNLD - UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=1
  FOR TABLESPACE DSNDB04.TCCSIDE
DSNU250I    DSNUUNLD - UNLOAD PHASE COMPLETE, ELAPSED TIME=00:00:00
DSNU012I    DSNUGBAC - UTILITY EXECUTION TERMINATED, HIGHEST RETURN CODE=8
```

To prevent truncation, the UNLOAD statement should be modified as follows:

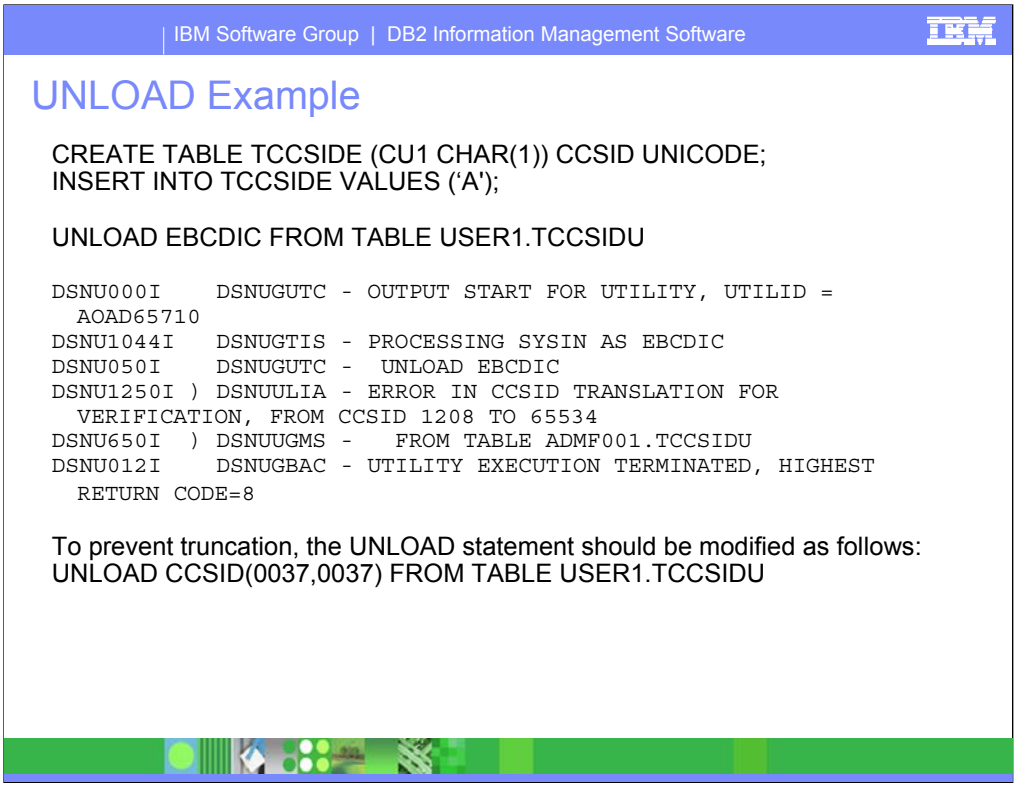
```
UNLOAD CCSID(1208) FROM TABLE USER.TCCSIDE (CE1 POSITION(*) VARCHAR(33))*
```

*Recommendation based on Table 25 (*Result length of CCSID conversion) in the SQL Ref*

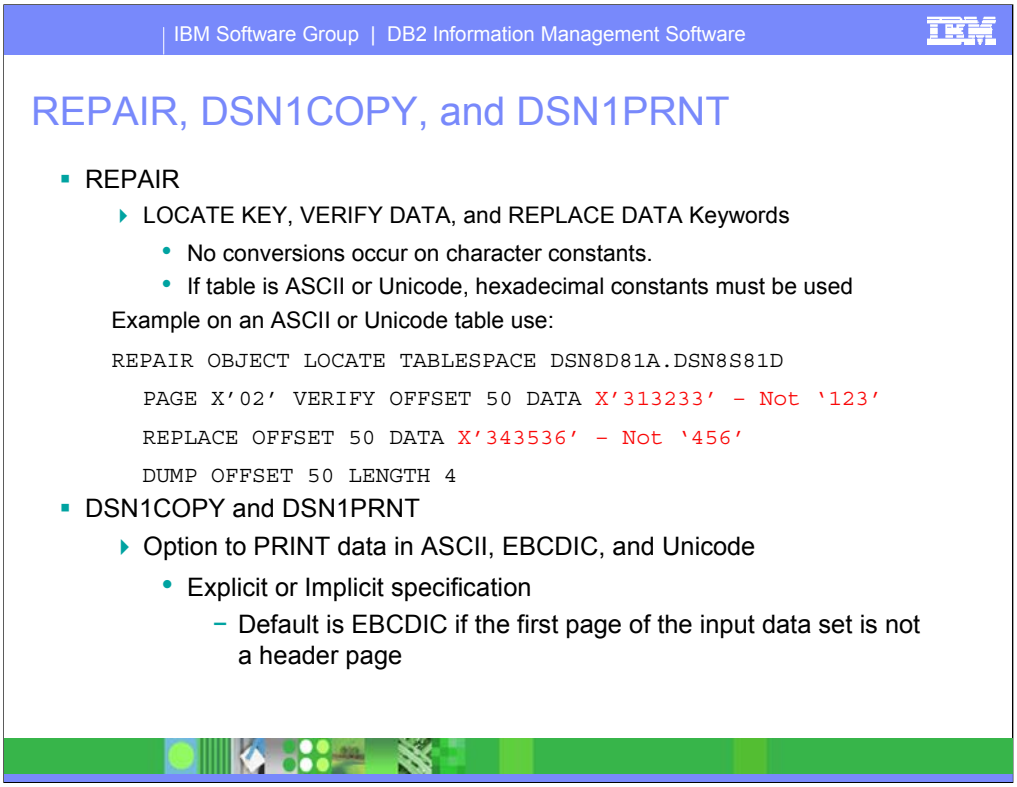
When unloading data, if conversion is involved, Table 25 in the V8 SQL reference should be consulted and lengths adjusted accordingly to ensure that truncation problems do not occur.

In this case, 'Hegelstraße' expands from 11 bytes in EBCDIC to 12 bytes in Unicode UTF-8.

Also note that had 'Jürgen' been inserted as 'Jürgen ' (with 5 pad characters), that the unload of that row would also have failed. Unlike in SQL, where trailing blanks are insignificant, with the UNLOAD utility, trailing blanks are significant. The UNLOAD utility can remove trailing blanks if you use the STRIP clause.



In this example, the UNLOAD utility is trying to convert UTF-8 data to the MIXED EBCDIC CCSID. Since this system is a MIXED=NO system, there isn't a MIXED EBCDIC CCSID, so the reserved value of 65534 is used. To prevent this issue, explicit specification of CCSIDs should be used. If UTF-16 data is also involved, a similar issue could involve, and the UNLOAD statement should be changed to specify CCSID(0037,0037,0037).



REPAIR can repair ASCII and Unicode data, as well as EBCDIC data, however, you must specify the data using hexadecimal constants, not literal constants.

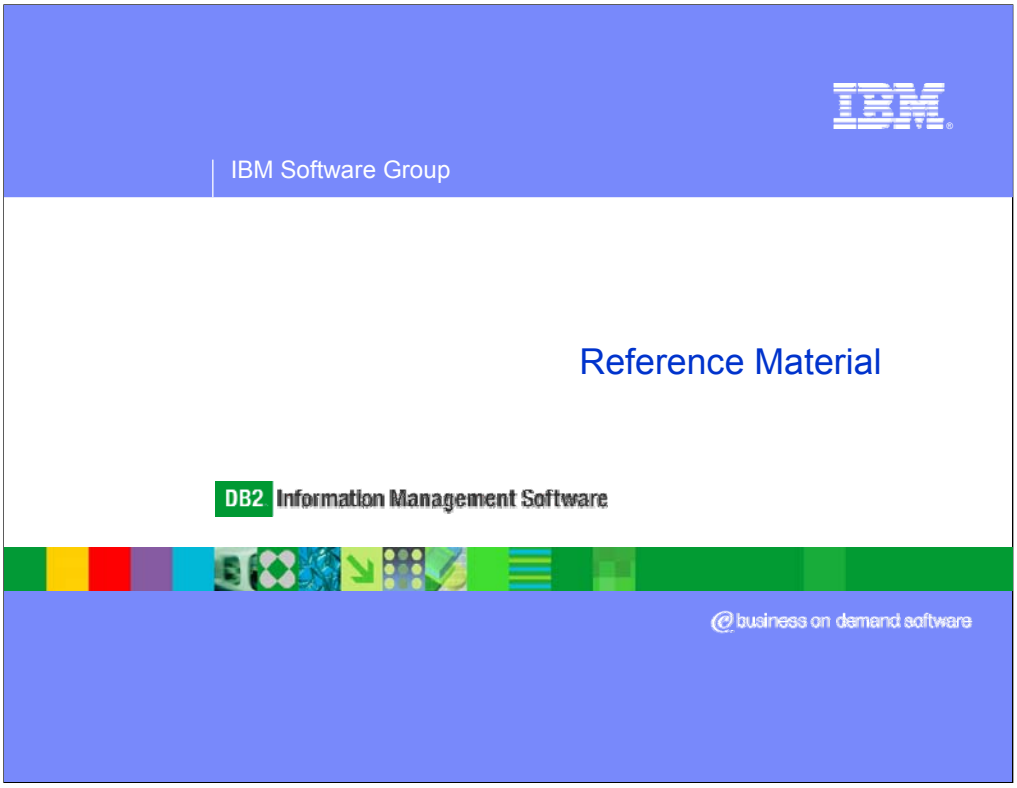
Session C02

CCSID 201 – What's a CCSID and why do I care?

Christopher J. Crone

IBM/DB2 for z/OS Development

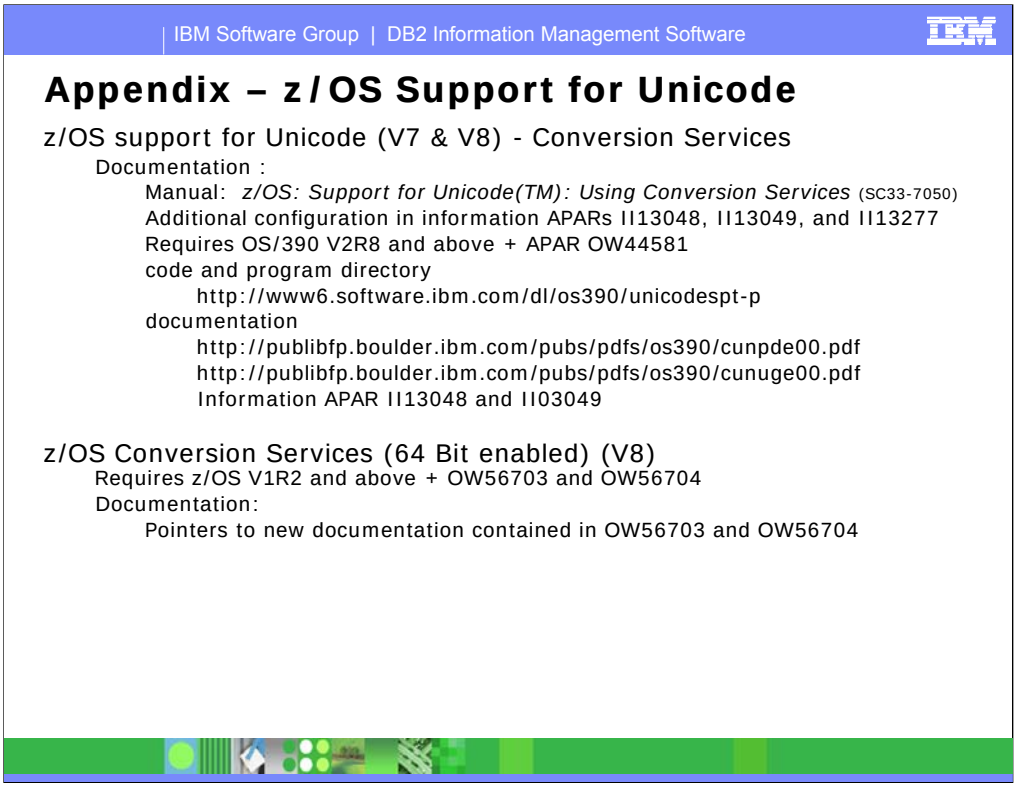
cjc@us.ibm.com


IBM Software Group
Reference Material
DB2 Information Management Software
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

In appendix E of the DB2 UDB Admin Guide, there is a discussion of NLS issues and how to set/override the codepage at the client using the codepage keyword on NT, and LANG variable on AIX.



The z/OS support for Unicode now offers

-Conversion

-Normalization

-Casing

These services are used by DB2 and are also available to Assembler and C/C++ applications.