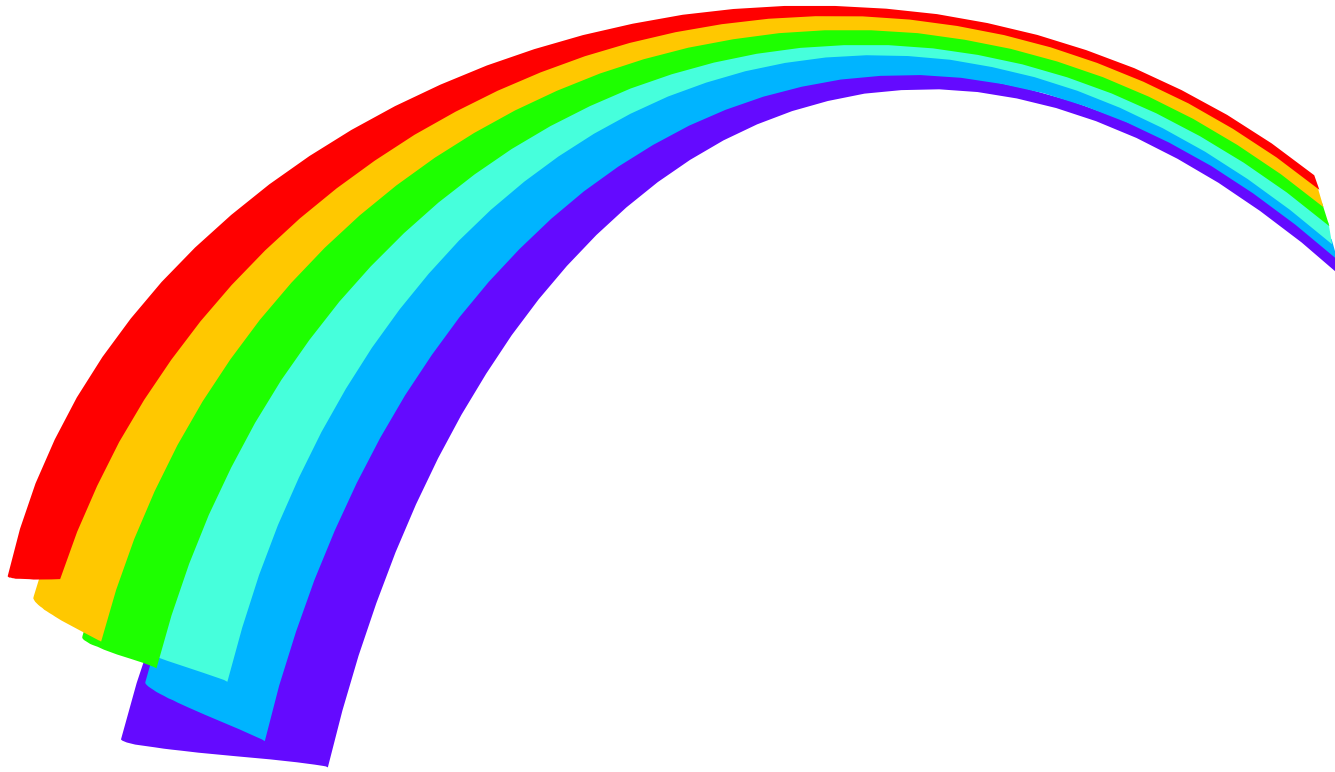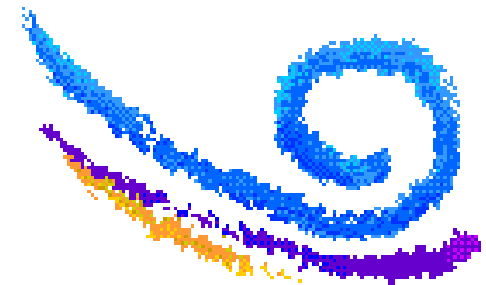# The Arc of DB2

Using DB2 to work the World Wide Web

Steve Comstock for The Trainer's Friend, Inc.

Each new release of DB2 introduces new features

▶ Many are clearly beneficial, but others may not seem relevant to your shop

- So you may be in no hurry to keep current
  - Who needs the extra hassle and work?

▶ Today I would like to explore three of the areas that may seem esoteric but actually have great potential

▶ In addition, we will look briefly at some more general features new with DB2 V9.1
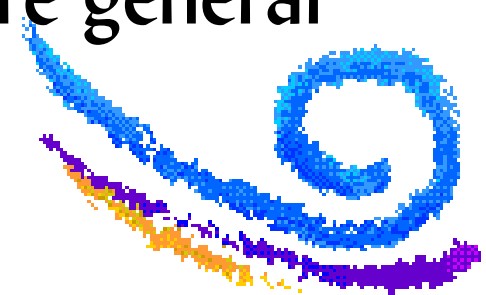
There are some common threads

► Provide richer data to the Web from DB2, for example:

- ■ LOBs (Large OBjects) - store and present media and documents

- ■ Unicode - because the Web is really the World Wide Web and it speaks in many languages

- ■ XML (eXtensible Markup Language) - the standards-based technology for interchanging self-describing data
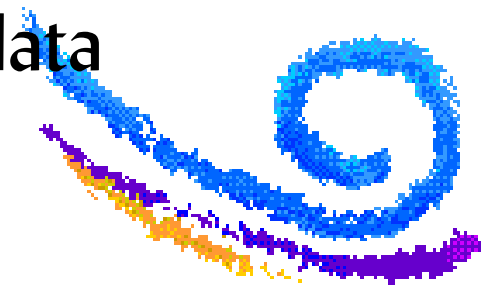
BLOBs, CLOBs, and DBCLOBs

► Large OBjects (LOBs) come in three flavors, but they are all ways to store unstructured data

- BLOBs - Binary Large OBjects include photos, movies, sound, signatures

- CLOBs - Character Large OBjects include documents such as policies, contracts, specifications

- DBCLOBs - Double Byte Character Large OBjects are CLOBs composed of CJK (Chinese, Japanese, Korean) characters encoded in DBCS or UTF-16

4
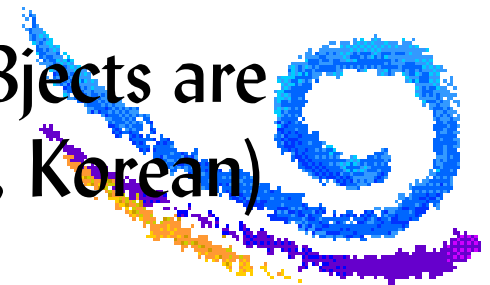
# But what <u>use</u> are LOBs?

► If DB2 is your backend data server to the Web, you can store product images in BLOBs to produce online catalogs including pictures, for example

► You can store document chunks in CLOBs, then combine them to build tailored contracts and policies

► Data items > 32K in size <u>must</u> be stored in LOBs, although you can store smaller items; 2GB is the maximum size per LOB
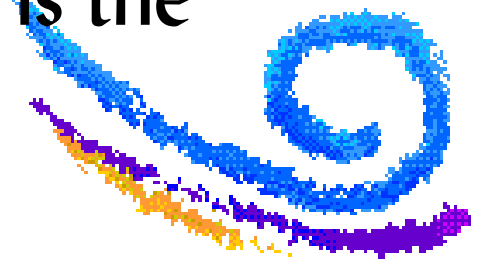
Using LOBs - an Example

► Suppose we have an inventory database that contains the following fields (plus more, but we'll ignore them):

- PartNo - a nine-byte field of the form "Part*nnnnn*"

- Description - a 30 byte field

- QOH (Quantity on Hand) - an integer

- Picture - an image of the item, for our online catalog
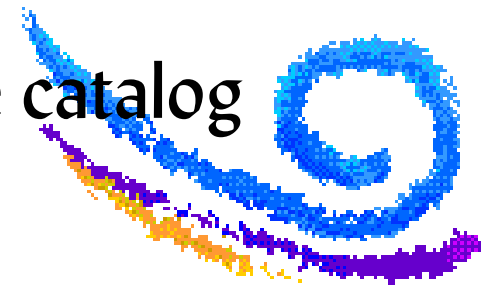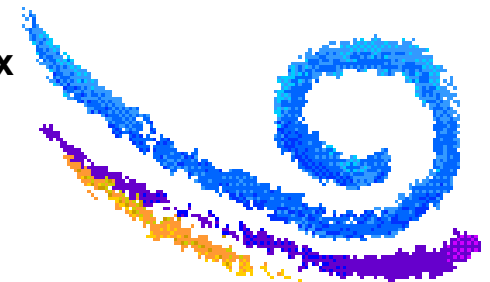
Using LOBs - an Example, 2

► The first thing we have to do is define the table, and it might be something like this (using SPUFI):

```
create table Items
  (PartNo        char(9)   not null,
   Description   char(30)  not null,
   QOH           int       not null,
   ItemRowID     rowid     not null,
   Picture       blob(4m),
constraint EnsurePartNo primary key (PartNo)  );


create unique index PartNoIndex on Items(PartNo);
```

Notes:
* A table with any kind of LOB must have a ROWID column
* Every table should have a primary key, and this requires a unique index

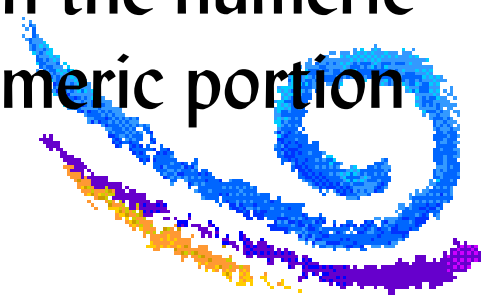# Using LOBs - an Example, 3

► Each LOB column requires an auxiliary tablespace, LOB table, and index

► DB2 Version 9.1 helps: if you issue CREATE TABLE that includes one or more LOB columns, and do not specify a tablespace, DB2 automatically creates the auxiliary database, tablespace, LOB table, and index

  ■ Perhaps something DBAs would rather control, but at least it's easy to try things
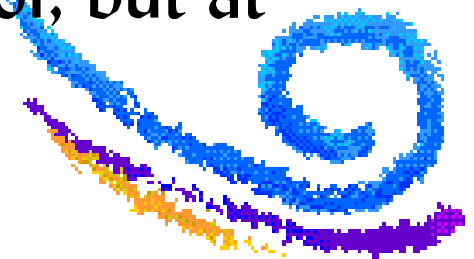
Using LOBs - an Example, 4

► Of course, we need to get our images into our database LOB column in a standard graphics format such as .bmp, .jpg, .png, .gif, and so on

- Perhaps simply a digital camera shot, perhaps something more professional, but it needs to end up as an image file on our workstation

- We named the images "part*nnnnn*.bmp", with the numeric portion of the name corresponding to the numeric portion of the item's part number
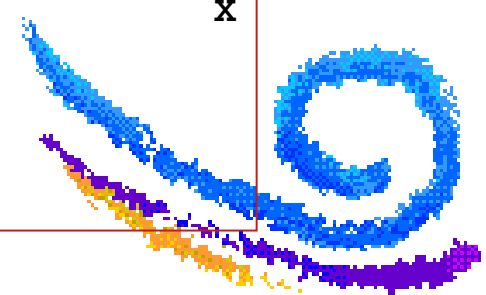
Using LOBs - an Example, 5

► Next, upload the image files to our z/OS system, preferably into our HFS or zFS

  ▪ We created a directory /u/scomsto/blobs and uploaded all the image files into that directory (as binary, of course)
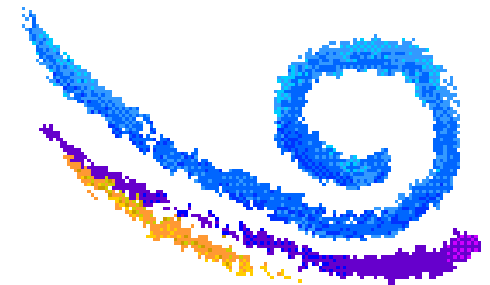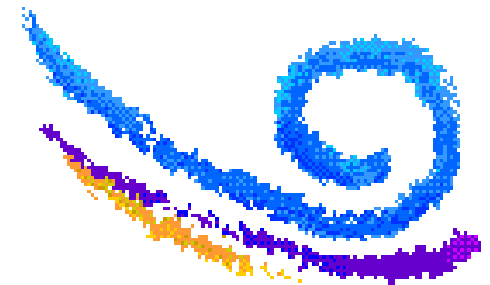
Using LOBs - an Example, 6

► Next, we have to populate our table
  - We used the LOAD DB2 utility, something like this:

```
//SCOMSTO  JOB ...
//STEP1 EXEC DSNUPROC,UID=SCOMSTO.LOADLOB
//SYSUT1    DD  UNIT=SYSALLDA,SPACE=(TRK,(20,20))
//SORTOUT   DD  UNIT=SYSALLDA,SPACE=(TRK,(20,20))
//SYSIN     DD  *
LOAD DATA RESUME NO REPLACE CONTINUEIF(72:72)='X' INTO TABLE ITEMS
    (PARTNO        POSITION (01) CHAR(9),
     DESCRIPTION   POSITION (10) CHAR(30),
     QOH           POSITION (42) INTEGER EXTERNAL(2),
     PICTURE       POSITION (80) CHAR(30) BLOBF )
//SYSREC DD *
Part00105Keys to the Kingdome              001075003483                    X
/u/scomsto/blobs/part00105.bmp
Part00240Overtime Hours                    001750000288                    X
/u/scomsto/blobs/part00240.bmp
.
.
.
```

Using LOBs - an Example, 7

► DB2 Version 9.1 helps: the LOAD utility can now use HFS file names (can also be PDS/E member names) to hold names of files containing BLOB data

- ▪ (See the BLOBF data type in the statements on the previous page)

- ▪ Also works for CLOBs and DBCLOBs (that is, CLOBF and DBCLOBF can be used)

Using LOBs - an Example, 8

► At this point, we wanted to prepare for writing code against this table, so we used DCLGEN, like this:

```
Enter table name for which declarations are required:
 1   SOURCE TABLE NAME ===> items
 2   TABLE OWNER ..... ===> SCOMSTO
 3   AT LOCATION ..... ===>                          (Optional)
Enter destination data set:           (Can be sequential or partitioned)
 4   DATA SET NAME ... ===> TR.COBOL(ITEMS)
 5   DATA SET PASSWORD ===>             (If password protected)
Enter options as desired:
 6   ACTION .......... ===> REPLACE    (ADD new or REPLACE old declaration)
 7   COLUMN LABEL .... ===> NO         (Enter YES for column label)
 8   STRUCTURE NAME .. ===>                            (Optional)
 9   FIELD NAME PREFIX ===> host-                      (Optional)
10   DELIMIT DBCS .... ===> YES        (Enter YES to delimit DBCS identifiers)
11   COLUMN SUFFIX ... ===> YES        (Enter YES to append column name)
12   INDICATOR VARS .. ===> NO         (Enter YES for indicator variables)
13   RIGHT MARGIN .... ===> 72         (Enter 72 or 80)
```
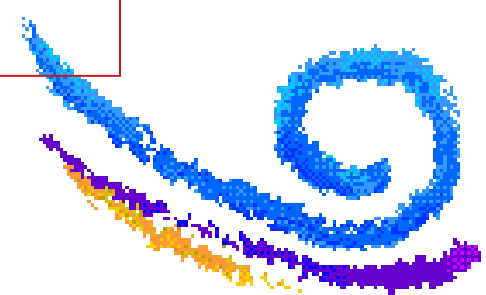
Using LOBs - an Example, 9

► The resulting statements, after a little tweaking:

```
      EXEC SQL DECLARE SCOMSTO.ITEMS TABLE
      ( PARTNO                          CHAR(9) NOT NULL,
        DESCRIPTION                     CHAR(30) NOT NULL,
        QOH                             INTEGER NOT NULL,
        ITEMROWID                       ROWID NOT NULL,
        PICTURE                         BLOB(4194304)
      ) END-EXEC.
**************************************************************
* COBOL DECLARATION FOR TABLE SCOMSTO.ITEMS                 *
**************************************************************
 01   DCLITEMS.
      10 HOST-PARTNO          PIC X(9).
      10 HOST-DESCRIPTION     PIC X(30).
      10 HOST-QOH             PIC S9(9) USAGE COMP.
      10 HOST-ITEMROWID       USAGE SQL TYPE IS ROWID.
      10 HOST-PICTURE         USAGE SQL TYPE IS BLOB-LOCATOR.
```
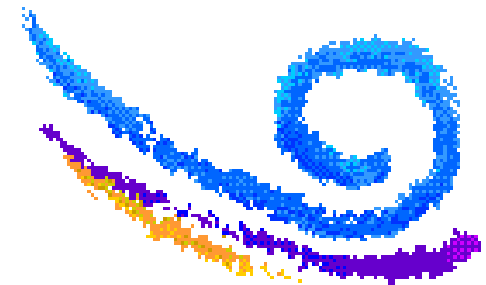
Using LOBs - an Example, 10

► Our goal was to serve web pages that included these BLOBs on demand; this required:

- Designing and coding a web page where the user could request information about an item

- Coding a COBOL program to run as a CGI that would accept the request and return a web page containing the image of the requested item

Using LOBs - an Example, 11

► The page we designed looked like this:
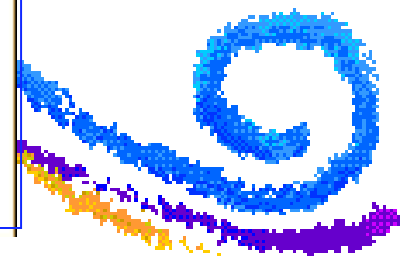
Using LOBs - an Example, 12

► The HTML to create that page is available in the supplementary materials

- As are the bmp files, the stylesheet, and the other supporting code and control data

- This is not the place to explore HTML (different course, more money)

Using LOBs - an Example, 13

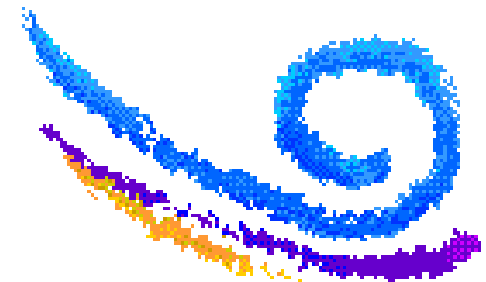► However, here are few notes regarding how we set up the files we provide in our environment - you need to make your own adjustments:

- Our HTTP configuration sets "public_html" as our starter directory for each user's HTML; we placed our HTML (and stylesheet and animated gif) in the subdirectory /u/scomsto/public_html/db2work, to keep other work separate

- The animated GIF logo we found at http://www.animationlibrary.com/

- In our example, we hardcoded the entries in the list box; in a real world example, you would generate this page through another CGI, so you always create a current list

Using LOBs - an Example, 14

► More notes on the supplementary files:

- Our HTTP configuration maps /SCOMSTO/* to /u/scomsto/CGI/*, where I place my own CGIs

- The logic is: whenever an item in the drop down list is clicked, the variable "ano" is set to contain the corresponding part number; when the "submit" button is clicked, the CGI named "coblob1" in my CGI directory will get invoked, being passed the string: ano=Part*nnnnn*

- Program "coblob1" is a COBOL program that runs as a CGI, accepts the value in ano as the WHERE value in a SELECT statement, and locates the corresponding row; then the program generates an HTML page to display the information, including the BLOB that is the picture of the corresponding item
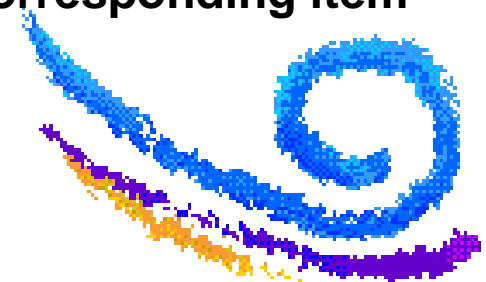
Using LOBs - an Example, 15

► Some highlights from the COBOL CGI (just looking at the DB2 part, especially handling the BLOBs):

```
      EXEC SQL INCLUDE SQLCA END-EXEC.
      EXEC SQL DECLARE SCOMSTO.ITEMS TABLE
      ( PARTNO                          CHAR(9) NOT NULL,
        DESCRIPTION                     CHAR(30) NOT NULL,
        QOH                             INTEGER NOT NULL,
        ITEMROWID                       ROWID NOT NULL,
        PICTURE                         BLOB(4194304)
      ) END-EXEC.
***************************************************************
 01   DCLITEMS.
      10 HOST-PARTNO            PIC X(9).
      10 HOST-DESCRIPTION       PIC X(30).
      10 HOST-QOH               PIC S9(9) USAGE COMP.
      10 HOST-ITEMROWID         USAGE SQL TYPE IS ROWID.
      10 HOST-PICTURE           USAGE SQL TYPE IS BLOB-LOCATOR.
***************************************************************
 01  blob-file-var usage is sql type is blob-file.
```
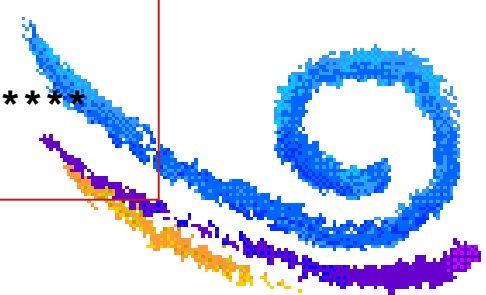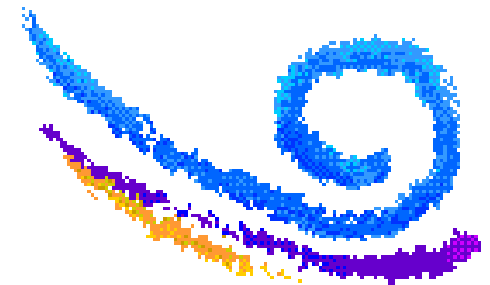
Using LOBs - an Example, 16

▶ DB2 Version 9.1 helps: the ability to reference LOBs using <u>file reference variables</u> is new:

- In COBOL, specify a USAGE as SQL TYPE BLOB-FILE (or CLOB-FILE or DBCLOB-FILE)

- If you use file reference variables for retrieval of a LOB, DB2 copies the data in the LOB table into the filename you specify; on update or insert, the contents of a named file will be copied into the database LOB table
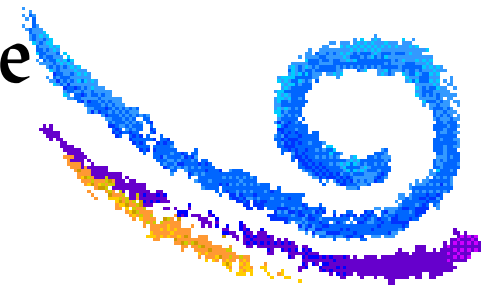
Using LOBs - an Example, 17

► Use file reference variables when you simply want to retrieve or replace the value in a LOB column

- Since BLOB data is unstructured (but probably well-defined), you certainly don't want to change <u>part</u> of the value!
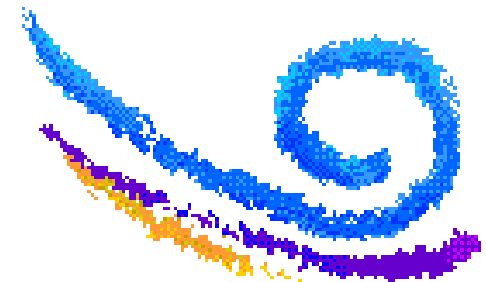
Using LOBs - an Example, 18

▶ When you specify a file reference variable, as we did with:

```
01  blob-file-var usage is sql type is blob-file.
```

■ ... the pre-processor or co-processor generates four sub fields, using a well-defined naming convention; in our case we get:

```
01   BLOB-FILE-VAR.
     49 BLOB-FILE-VAR-NAME-LENGTH PIC S9(9) COMP-5 SYNC.
     49 BLOB-FILE-VAR-DATA-LENGTH PIC S9(9) COMP-5.
     49 BLOB-FILE-VAR-FILE-OPTION PIC S9(9) COMP-5.
     49 BLOB-FILE-VAR-NAME PIC X(255).
```
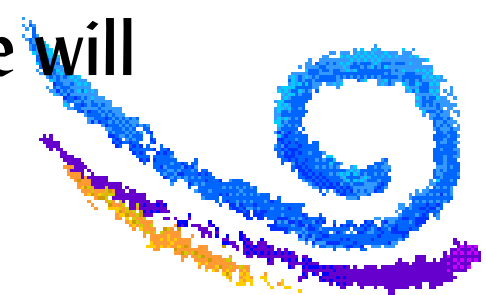
Using LOBs - an Example, 19

► Working with file reference variables requires you to:

- Set a value into blob-file-var-file-option to indicate if you intend to:
  - 2 - read an existing file
  - 8 - create a new file (and if it exists that is an error)
  - 16 - overwrite an existing file (and if it doesn't exist, create it)
  - 32 - append to an existing file (and if it doesn't exist, create it)
- Data items are generated with these values, if you care to use them

Using LOBs - an Example, 20

▶ Working with file reference variables requires you to:

- Build the file name in blob-file-var-name

- Put the length of the file name in blob-file-var-name-length

- If you are inserting or updating a LOB, you must put the size of the file into blob-file-var-data-length

- If you are retrieving a LOB, the SELECT service will populate blob-file-var-data-length for you
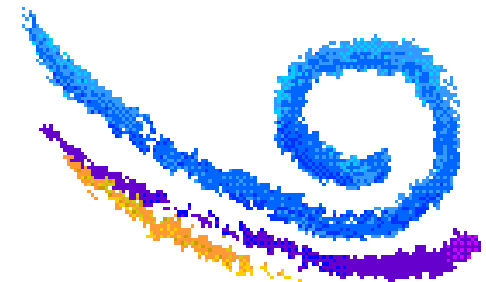
Using LOBs - an Example, 21

► Here is the code we used to retrieve our BLOB:

```
      move sql-file-overwrite to blob-file-var-file-option
      move 1 to blob-file-var-name-length
      string '/u/scomsto/public_html/db2work/'
              trans-partno '.bmp' delimited by size
          into blob-file-var-name
          with pointer blob-file-var-name-length
      subtract 1 from blob-file-var-name-length
      exec sql
          select description, qoh,  picture
          into  :host-description, :host-qoh, :blob-file-var
          from scomsto.items
          where  partno = :trans-partno
      end-exec
```

Notes:
  * SQL-FILE-OVERWRITE is the name of the generated field with value 16
  * The other generated fields are:
      SQL-FILE-READ with value 2
   SQL-FILE-CREATE with value 8
      SQL-FILE-APPEND with value 32

Using LOBs - an Example, 22

► Given that we know the part number we are after, we can construct the name for the blob file by appending .bmp and placing this in an available directory

  ■ We ended up with /u/scomsto/public_html/db2work/Part*nnnnn*.bmp
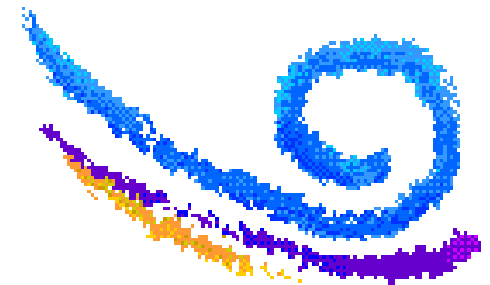
  Note:
     * You can have indicator variables for LOB columns, for managing NULL values when retrieving / updating these columns; in our application we didn't need them, since only valid part numbers can be requested and all rows have non-NULL values for the PICTURE column
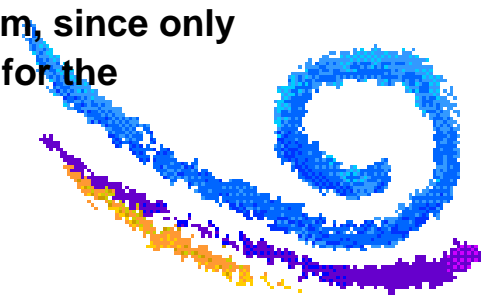
Using LOBs - an Example, 23

► If the SELECT is successful, we have retrieved the information we need

  ▪ We can now build an HTML page to send back to the requestor

    ● In COBOL, you do this with a series of "display" statements

► Then we're done!

Using LOBs - an Example, 24

►Here is a sample response from our CGI:

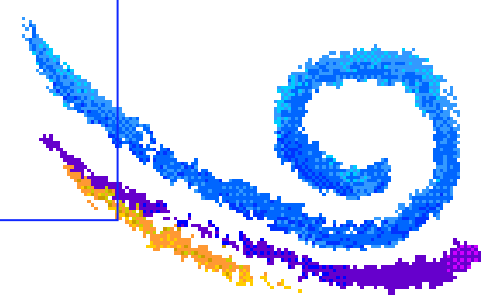

Welcome to The Trainer's Fiend Novelty Products Division

"We provide the finest products you can't find anywhere else!"

Here is the item called User Friendly Toys
Quantity on hand: 38

LOBs - a Summary

► BLOBs, CLOBs, and DBCLOBs provide unique opportunities to handle nontraditional data

► DB2 V9.1 provides new facilities to make it easier to store and work with documents and multimedia as LOBs in your database tables:

- Automatic create of supporting infrastructure
- File reference variables, usable in utilities such as LOAD as well as from programming languages
- FETCH CONTINUE to access parts of LOBs in small chunks (not discussed here)
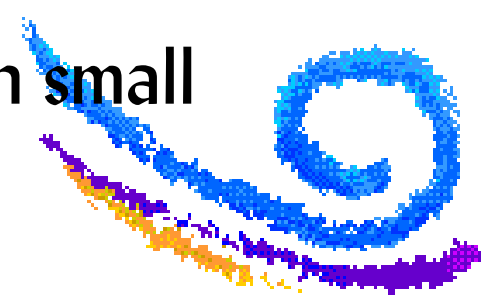
Unicode - working with ßµßO.com
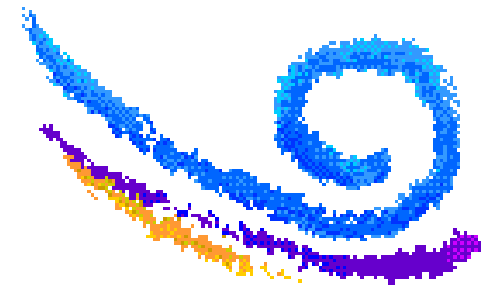
► Every business today needs to have a website

- A presence that represents the company and its products and services to customers and prospects, as a minimum

- A place to sell and fulfill orders

- A place to provide support to customers

31

Unicode - working with ßµßO.com

► Even if you only do business locally, once you are on the Web your company is connected to the world

- If you sell to an international clientele, you need to be aware of differences in language and culture

- But even if you only sell in the US, you want to come across as web-savvy

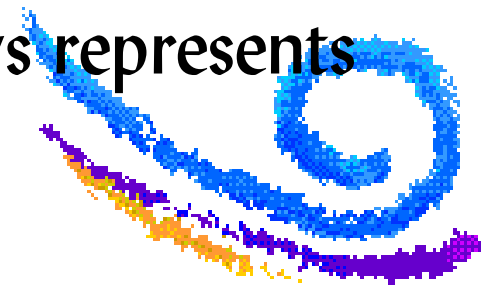# Unicode - working with ßµßO.com

► The Web has been deliberately designed to support all languages around the world

  ▪ But classic EBCDIC (the character encoding of IBM mainframes and AS/400 machines) and classic ASCII (the character encoding for everyone else) both fall short

    ● 8 bits (EBCDIC) and 7 bits (ASCII) don't contain enough patterns to uniquely assign a bit pattern to each character you need
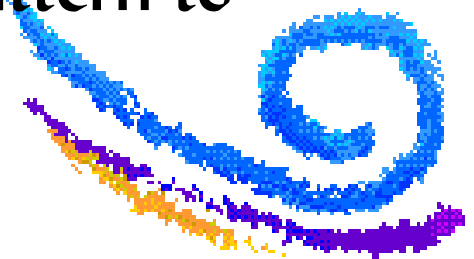
Unicode - working with ßµßO.com

► Unicode - the Universal Code Page - is a standard supported by a large number of the major players in the IT industry

- The Unicode Consortium was founded in 1991 to devise a standard for character encoding that would be:

  - <u>Universal</u> (encompass all characters likely to be needed)
  - <u>Efficient</u> (no escape sequences required, just characters)
  - <u>Unambiguous</u> (any given code point always represents the same Unicode character)

Unicode - working with ßµßO.com

► Industry support - a sampling of the list of members of the Unicode Consortium:

| Full Members	Associate Members	Liaison Members
Adobe Systems	AOL	CCID, Beijing
Apple Computer	Booz, Allen, Hamilton	High Council of Informatics, Iran
Google, Inc.	Church of LDS	ICTA, Sri Lanka
H-P	Columbia University	IETF
IBM	Linotype Library	ISO/IEC JTC1/SC2
Microsoft	NCR	Linguistic Society of America
Oracle	Nokia	Object Management Group
Peoplesoft	SAS Institute	Standard Norge
SAP AG	Sony Ericsson	Swedish Standards Institute
Sun	Symbian, Ltd	TCVN/TCI, Hanoi, Viet Nam
Sybase	Talis Information	World Wide Web Consoritium
Verisign	Vernacular Information	(W3C) 118N Core Working Group
Yahoo!	Society Project	
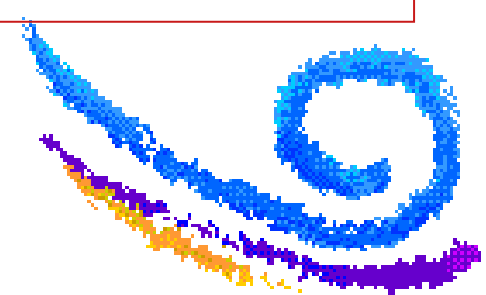
Unicode - working with ßµßO.com

▶ So there are heavy hitters here, and they have made significant commitments to Unicode:

- Java (from Sun) - all strings are Unicode

- .NET languages (from Microsoft) - all strings are Unicode

- Most browsers support Unicode

- The various internet standards groups have declared that XML, HTML 4.1 and later, XHTML, and so on, should all be encoded in Unicode

Mainframe support for Unicode includes:

▶ Hardware instructions to convert between Unicode and other code pages, to search in Unicode strings, compare Unicode strings, and so on

▶ Suite of callable services

▶ Support in Assembler, PL/I, COBOL, C, etc.

▶ DB2
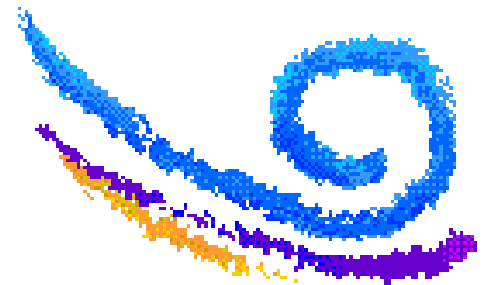
Unicode details

► Current standard is 5.0, with 98,884 graphic characters assigned code points (72% (71,226) of them are CJK - Chinese, Japanese, Korean - characters)

► Each code point is an integer called the <u>Unicode scalar value</u>, in the range of 0 to 1,114,111 (plenty of room for growth)

  ■ As a binary number, this is a 21-bit unsigned integer
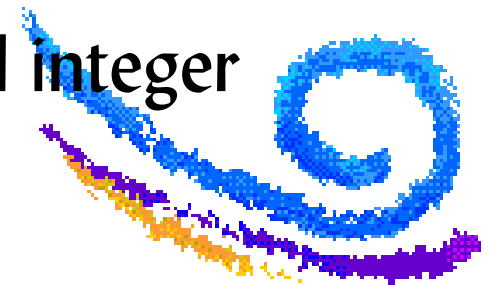
Unicode details, continued
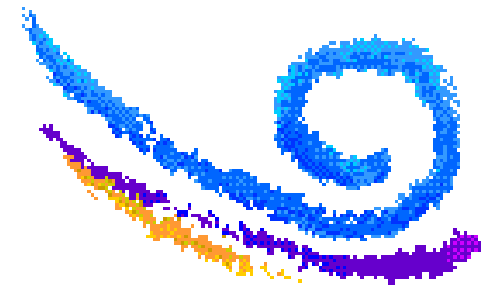
► Unicode strings are stored in one of three formats (called UTF - Unicode Transformation Format):

- **UTF-32** - 4 bytes per character; leftmost 11 bits always 0

- **UTF-16** - 2 bytes for most characters; some require 4 bytes, as a pair of surrogate characters

- **UTF-8** - each character is 1 byte or 2 bytes or 3 bytes or 4 bytes [the 1 byte characters are the same as 7-bit ASCII]
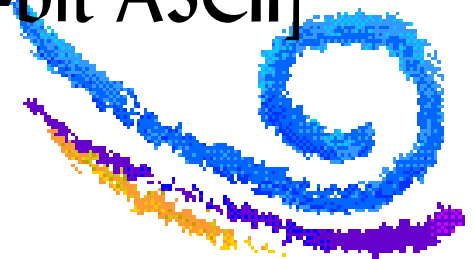
# Unicode details, continued

► The fact that the 1-byte characters of UTF-8 are the same as 7-bit ASCII has helped acceptance

- But you have to be careful when working with data outside the range that fit into one byte: strings could be longer than you expect

- UTF-16 seems to be the most common format used, although there does seem to be some interest in moving to UTF-32

DB2 and Unicode

► Although there was some earlier support of Unicode, Version 8 of DB2 was really <u>the</u> Unicode release:

- The catalog code was re-written, and character data in the catalog is now encoded in UTF-8

- You can specify an encoding of EBCDIC, ASCII, or UNICODE in CREATE statements for data bases, parameters in user defined functions and stored procedures, tables, table spaces, user defined types, and even host variables in a program
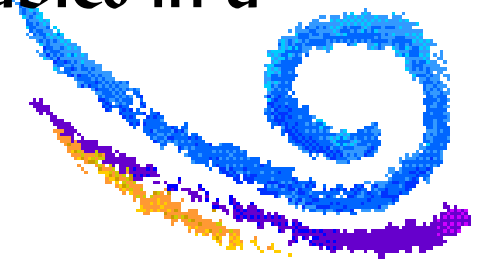
DB2 and Unicode, continued

► The good news was (and still is): you don't need to convert your existing EBCDIC <u>data</u> to Unicode

► Some existing queries had different results under Version 8 than they did in Version 7

  ■ For example, ORDER BY and WHERE clauses that relied on collating sequence might produce different results for queries against SYSIBM. tables (catalog tables), since the collating sequence for Unicode is different than that of EBCDIC
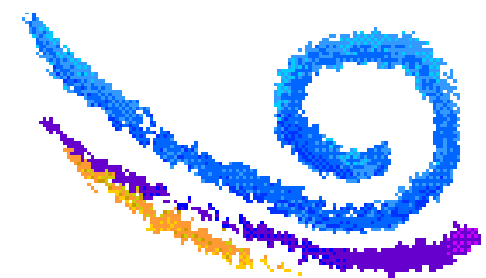
DB2 and Unicode, continued

► More good news is that you <u>can</u> store data in Unicode

- Providing a way to store customer names, product descriptions, domain names, and other text, in any language

- Note that CGIs written in Assembler, PL/I, COBOL, or C can write HTML pages in Unicode, saving some translation overhead

- Prediction: most character data will stay EBCDIC for a long time; some early experimentation is needed first
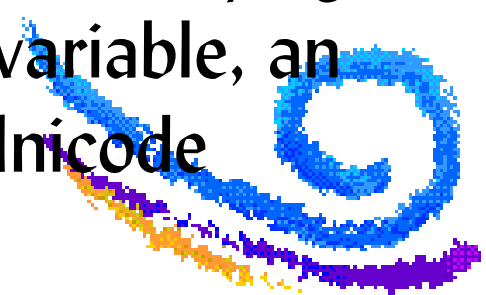
DB2 and Unicode, continued

► Here are some lines from a sample report using Unicode data:

```
00234   Mike's Mystical Magic Shop

00235   Heidi's Secret Dream Palace

00241   City Center Bag of Tricks

00243   花火と花屋

00246   Phoney Baloney and Other Stuff

00374   Nick's Novelties and Pawn Shop

00383   Steve's Slight Sleights

00384   Hunter's Hang Out

00390   King Korn Komedy Shop

00392   Kitty's Central Spot

00395   The Miracle Smile

00401   John's Emporium

00407   圣训之友

00413   On The Edge

00414   Central Periphery
```

DB2 and Unicode, continued

▶ DB 2 Version 9.1 Unicode Support Additions

▶ The main addition to Unicode support in DB2 V9 is a number of SQL Unicode-related functions:

- ASCII_CHR(n) - with $0 <= n <= 255$; returns the ASCII character at codepoint n

- ASCII_STR(*string_exp*) - converts *string_exp* to a varying length ASCII string; *string_exp* can be a host variable, an expression, or a literal in ASCII, EBCDIC, or Unicode

DB2 and Unicode, continued

▶ DB 2 Version 9.1 Unicode Support Additions, continued

▶ The main addition to Unicode support in DB2 V9 is a number of SQL Unicode-related functions, continued:

- EBCDIC_CHR(n) - with $0 <= n <= 255$; returns the EBCDIC character at codepoint n

- EBCDIC_STR(*string_exp*) - converts *string_exp* to a varying length EBCDIC string; *string_exp* can be a host variable, an expression, or a literal in ASCII, EBCDIC, or Unicode
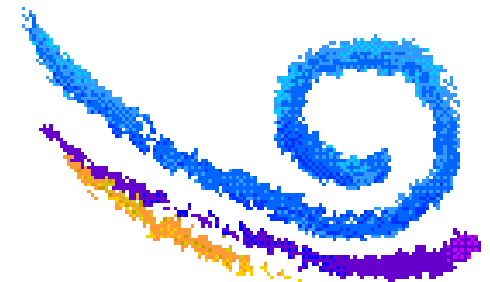
DB2 and Unicode, continued

▶ DB 2 Version 9.1 Unicode Support Additions, continued

▶ The main addition to Unicode support in DB2 V9 is a number of SQL Unicode-related functions, continued:

- UNICODE(*string_exp*) - returns the UTF-16 code value of the leftmost character of *string_exp*

- UNICODE_STR(*string_exp*[,UTF8|UTF16]) - returns a varying length string encoded in UTF8 (VARCHAR) or UTF16 (VARGRAPHIC); *string_exp* can be a host variable, an expression, or a literal in ASCII, EBCDIC, or Unicode

DB2 and Unicode, continued

▶ DB 2 Version 9.1 Unicode Support Additions, continued

▶ The main addition to Unicode support in DB2 V9 is a number of SQL Unicode-related functions, continued:

- CHARACTER_LENGTH(*expr*,{CODEUNITS16 | CODEUNITS32 | OCTETS}) - returns the length of *expr* in number of UTF16 characters, number of UTF32 characters, or number of bytes
- Previous functions LOCATE and POSITION have been enhanced to accept an optional extra parameter of {CODEUNITS16 | CODEUNITS32 | OCTETS}

DB2 and Unicode, continued

▶ DB 2 Version 9.1 Unicode Support Additions, continued

▶ The main addition to Unicode support in DB2 V9 is a number of SQL Unicode-related functions, continued:

  ■ COLLATION_KEY(*expr*, *coll_name* [,*n*]) - returns a varying length binary string that is the collation key of *expr* in the collation name *coll_name*; *n* is the desired length of the returned string
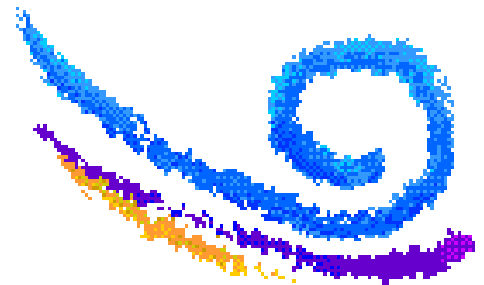
DB2 and Unicode, continued

► Digression: collation

 ▪ Which character should sort first?:

四　　五

...trick question - it depends

DB2 and Unicode, continued

► Digression: collation

- The first argument of the COLLATION_KEY function is a character or graphic string that is not a LOB

- The second argument, the collation name, is constructed as described in the manual on Unicode services (non-trivial)

- The third argument is the length to reserve for the key; the default is 12 times the length of the first argument; if the result is longer you must specify a longer value (might take some trial and error)
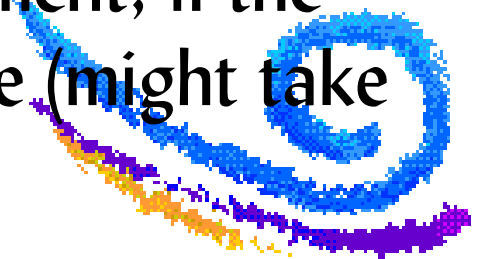
DB2 and Unicode, continued

► Digression: collation

  ■ We ran these tests:

```
select collation_key(ux'56DB','uca410_lja',84) from sysibm.sysdummy1;

select collation_key(ux'4E94','uca410_lja',84) from sysibm.sysdummy1;
```

► UX'56DB' is the hex value for the first Japanese character we showed (which is the symbol for '4'), and UX'4E94' is for the second (which is '5')

► 'UCA410_LJA' says to use the algorithm for Unicode version 4.1 (the latest version that the Unicode utilities support) for Japanese
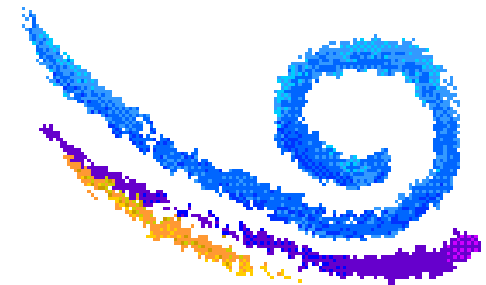
DB2 and Unicode, continued

► Digression: collation

- ▪ We ran these tests:

```
select collation_key(ux'56DB','uca410_lja',84) from sysibm.sysdummy1;

select collation_key(ux'4E94','uca410_lja',84) from sysibm.sysdummy1;
```

► The third operand, the length, was trial and error; we needed more than the default so we over-specified (the returned result was 32 bytes for each test):

```
---------+---------+---------+--
5FF62F00000000003D3D000000000300


---------+---------+---------+--
5FF41D00000000003D3D000000000300
```

DB2 and Unicode, continued

▶ DB 2 Version 9.1 Unicode Support Additions, continued

▶ The main addition to Unicode support in DB2 V9 is a number of SQL Unicode-related functions, continued:

■ LOCATE_IN_STRING(*haystack*, *needle* [,*start* [,*instance*]] [,{CODEUNITS16 | CODEUNITS32 | OCTETS}]) - returns the displacement of *instance*th occurrence of *needle* in the *haystack*, beginning at location *start*, with the search using two-byte, four-byte, or single-byte chunks
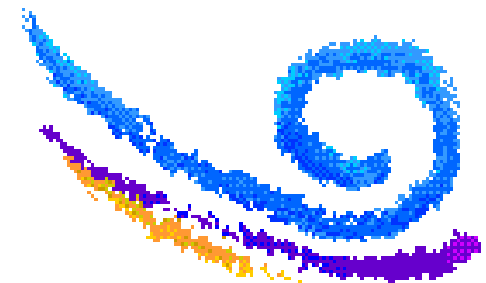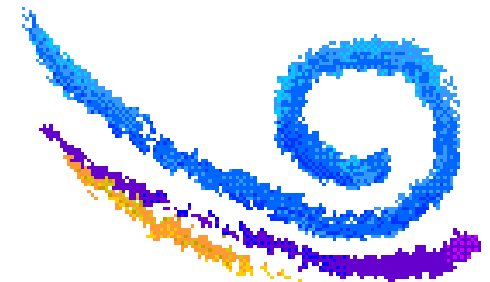
DB2 and Unicode, continued

► DB 2 Version 9.1 Unicode Support Additions, continued

► The main addition to Unicode support in DB2 V9 is a number of SQL Unicode-related functions, continued:

  ■ Existing functions LOWER and UPPER have been extended to include a locale specification and result length (for Unicode data, the result can be longer than the original string)
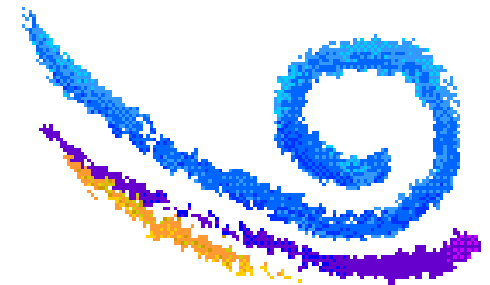
DB2 and Unicode, continued

▶ DB 2 Version 9.1 Unicode Support Additions, continued

▶ The main addition to Unicode support in DB2 V9 is a number of SQL Unicode-related functions, continued:

- NORMALIZE_STRING($expr$,{NFC|NFD|NFKC|NFKD}[,n]) - given a Unicode string, $expr$, returns the normalized form of the string, using one of four normalization algorithms, with a length of n characters
  - Normalization has to do, primarily, with representation of characters with accents and such in a consistent fashion

DB2 and Unicode, continued

▶ DB 2 Version 9.1 Unicode Support Additions, continued

▶ The main addition to Unicode support in DB2 V9 is a number of SQL Unicode-related functions, continued:

▪ OVERLAY(*base*,*new*,*displ*,*len*,{CODEUNITS16|CODEUNITS32|OCTETS}) - returns a string that is the *base* string overlayed by the *new* string beginning at displacement *displ* in the *base* string, for a length of *len*, where *displ* and *len* are in units of 2-bytes, 4-bytes, or 1-byte

Unicode - a summary

► Unicode is a character encoding designed to support all written characters in a single code page

  ■ It's use is mandated as the encoding for an increasing number of standards-based technologies such as XML, SOAP, Web Services, and more

► DB 2 Version 9.1 extends DB2's support of Unicode through the addition of new functions and the extension of previously existing functions
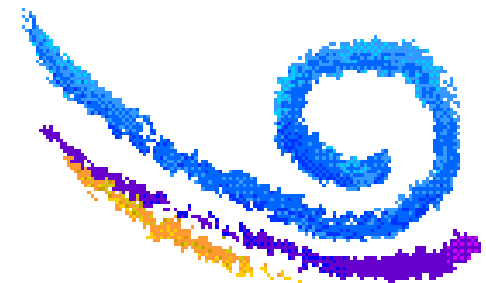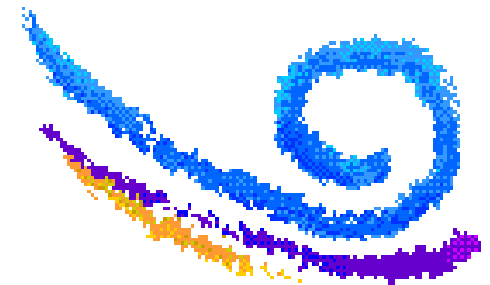
XML - the eXtensible Markup Language

► XML is a standard that describes how to describe data so it can be transmitted and understood on every computer platform

- To accomplish this, data formatted according to the XML standard contains both the data and the description of the data

  - To accomplish this, XML breaks down every piece of data into an <u>element</u>, strings the elements together in one long stream, and encodes the stream in Unicode
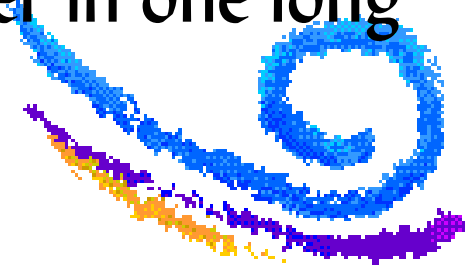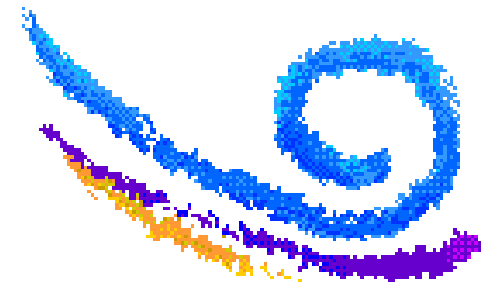
XML - a Brief Tutorial

► Since DB2 Version 9.1 has its biggest changes in the arena of XML, we really need to spend some time describing the major tenets and vocabulary of XML

► XML files are also called XML <u>documents</u>

► XML documents have a well-defined structure - we show an example on the next page...

XML - a Brief Tutorial, 2

► Here is a small XML document and some descrptive comments:

```
<?xml version="1.0" encoding="ibm037" ?>   <= xml declaration
<!-- sample inventory item -->             <= xml comment
<inventory>                        <= inventory element start tag (root element)
<rec_type>I</rec_type>               <= rec_type element: start tag, content, end tag
<part_no>PART00213</part_no>         <= part_no element: start tag, content end tag
<description supplier="SL1023">      <= description element: start tag and attribute
Green ketchup squirt bottles         <= description element: content
</description>                        <= description element: end tag
<qoh>55</qoh>                        <= qoh element: start tag, content, end tag
<unit_price currency="USD">4.75</unit_price>  <= unit_price element: start tag
                                               with attribute, content, end tag
<re-order_level>45</re-order_level>   <= re-order_level element
</inventory>                        <= inventory element end tag (root element)
```

XML - a Brief Tutorial, 3

► Although there are no pre-defined markup tags (which is why the language is extensible), there are some general rules that describe the composition of XML documents

► XML documents are composed of:

- A <u>prolog</u>, comprised of

  - An optional XML declaration statement, identifying the document to be an XML document

  - An optional document type declaration providing information regarding what markup is allowed, for example, or how to interpret particular tags or symbols

XML - a Brief Tutorial, 4

▶ XML documents are composed of (continued):

A prolog (optional xml declaration, optional document type declaration (DTD)), as discussed on the previous page

Note: instead of document type declaration, you may have a schema declaration, such as `<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">`

Schema's play the same role as DTD's: a definition of what's valid in a document

Any number of comments (format: start with: `<!--` end with: `-->` may cross lines)

Any number of elements (start tag with optional attribute(s), content, end tag)

Any number of Processing Instructions (PIs)

* A string beginning with `<?` followed immediately by a name (also called a PI target) followed by the content ending with `?>`

* Typically used to pass information to specific applications and ignored by other applications
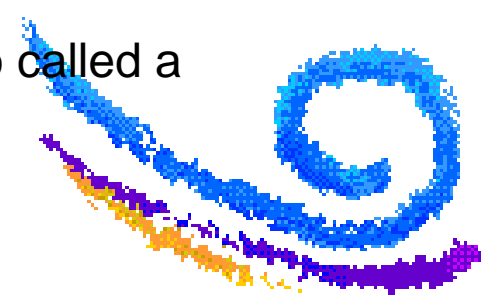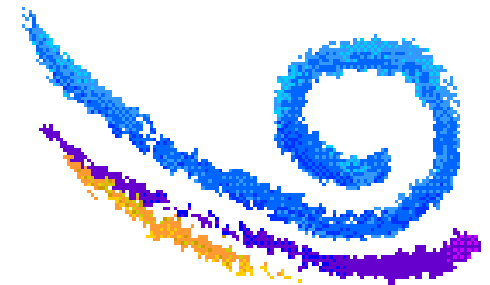
XML - a Brief Tutorial, 5

► Elements are the important pieces of XML documents

- An element is identified with a start tag:

  <*name*>

  where *name* is the user-chosen name of the element

- After the start tag comes the text that is the content of the element

- After the content is the end tag: </*name*> where *name* must be the same (including case) as in the end tag
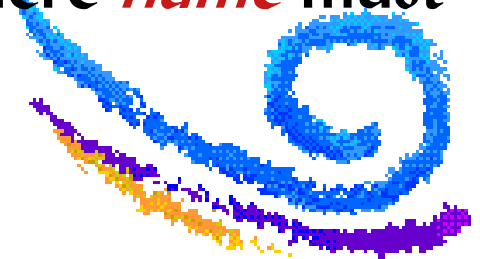
XML - a Brief Tutorial, 6

► The start tag for an element may contain one or more attributes - descriptive information that is intended to be useful to at least one of the application programs that will process (the literature says "consume") this document

► The format of an attribute is:

attribute_name="*attribute_value*"

or

attribute_name='*attribute_value*'

XML - a Brief Tutorial, 7

► Attribute <u>names</u> are case sensitive; case sensitivity of attribute <u>values</u> depends on the application program(s) using the data, but the values must be [double- or single-] quoted

► We had some examples earlier:

```
<description supplier="SL1023">
Green ketchup squirt bottles
</description>


<unit_price currency="USD">4.75</unit_price>
```

► Here's an example with multiple attributes

```
<object data="report1.html" type='text/html'>
This is the most recent report.</object>
```
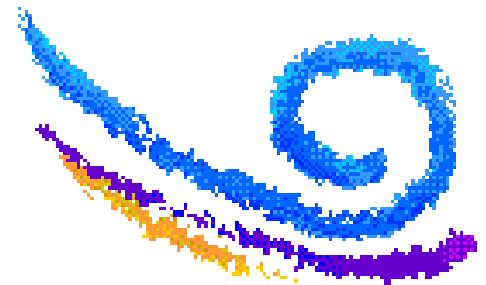
XML - a Brief Tutorial, 8

► Rules for names of elements and attributes:

- Names are case sensitive, of arbitrary length, composed of letters, numbers, hyphens, periods and underscores
  - "letters" encompass Latin, accented Roman characters, and letters from alphabets such as Cyrillic, Greek, Hebrew, Arabic, Thai, Hiragana, Katakana, Devanagari, even ideograms from Chinese, Japanese, and Korean

- Names begin with a letter or an underscore, colon, hyphen, or period (and some others - you don't want to know)

- Names beginning with 'xml', in any case, are reserved

XML - a Brief Tutorial, 9

►Named character references

- Because of the special meaning / use of the five characters
  < > ' " &

  when element content requires a literal of one of these, you
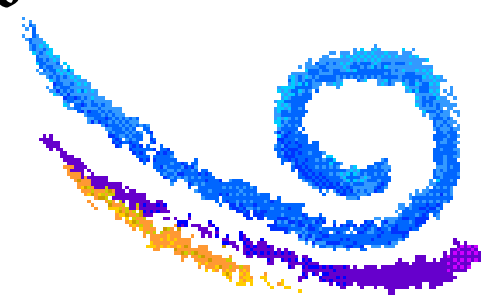 must specify it using special notation:

  <   for <
  >   for >
  &   for &
  '   for '   - inside attribute values
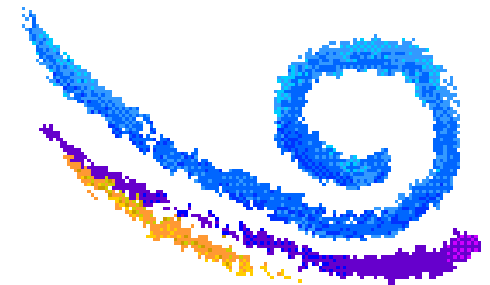 "   for "   - inside attribute values

XML - a Brief Tutorial, 10

► Named character references, continued

- ■ So, for example:

  <company>AT&T</company>   -incorrect

  <company>AT&amp;T</company>   -correct

XML - a Brief Tutorial, 11

► Empty elements

- An element may have no content, for example:
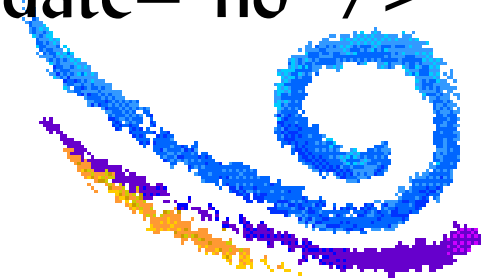
  `<found_on_hand></found_on_hand>`

- In such cases, you can simply have the end indicator inside the start tag itself (and this is more common):

  `<found_on_hand />`

- An empty element may still have attributes:

  `<process_status start="yes" parse="yes" validate="no" />`

XML - a Brief Tutorial, 12

► Element nesting

 ▪ The first element in a document is the <u>root</u> element

 ▪ All subsequent elements are <u>nested</u> inside the root - that is, the entire element is contained in the content of another element (terms: parent, child); for example

```
<inventory>
  <rec_type>I</rec_type>
  <part_no>PART00213</part_no>
  <description supplier="SL1023">
    Green ketchup squirt bottles
    <unitPrice currency="USD">13.50</unitPrice>
  </description>
</inventory>

In this example, inventory is the root with
children rec_type, part_no, and description;
description has a child element named unitPrice
```
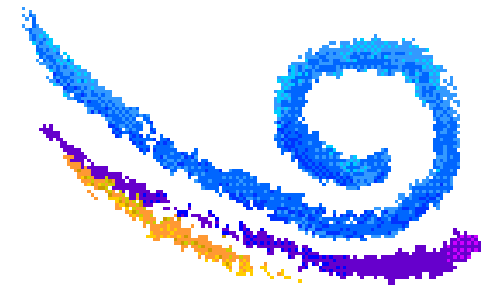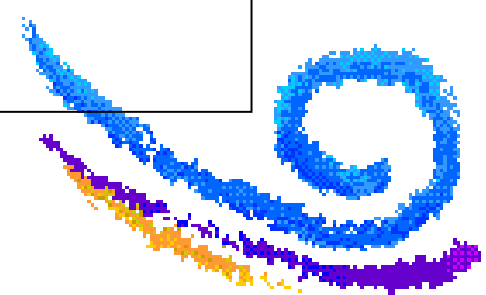
XML - a Brief Tutorial, 13

► We typically show elements and sub-elements on separate lines with indentation

► But in actuality, XML ignores white space (blanks, new lines, tabs, and the like) between elements and attributes and usually works with a document as a long uninterrupted string, called a <u>serialized string format</u>:

```
<inventory><rec_type>I</rec_type><part_no>PART00213</part_no><description
supplier="SL1023">Green ketchup squirt bottles<unitPrice
currency="USD">13.50</unitPrice></description></inventory>

Line breaks shown here are simply for typographical reasons
```
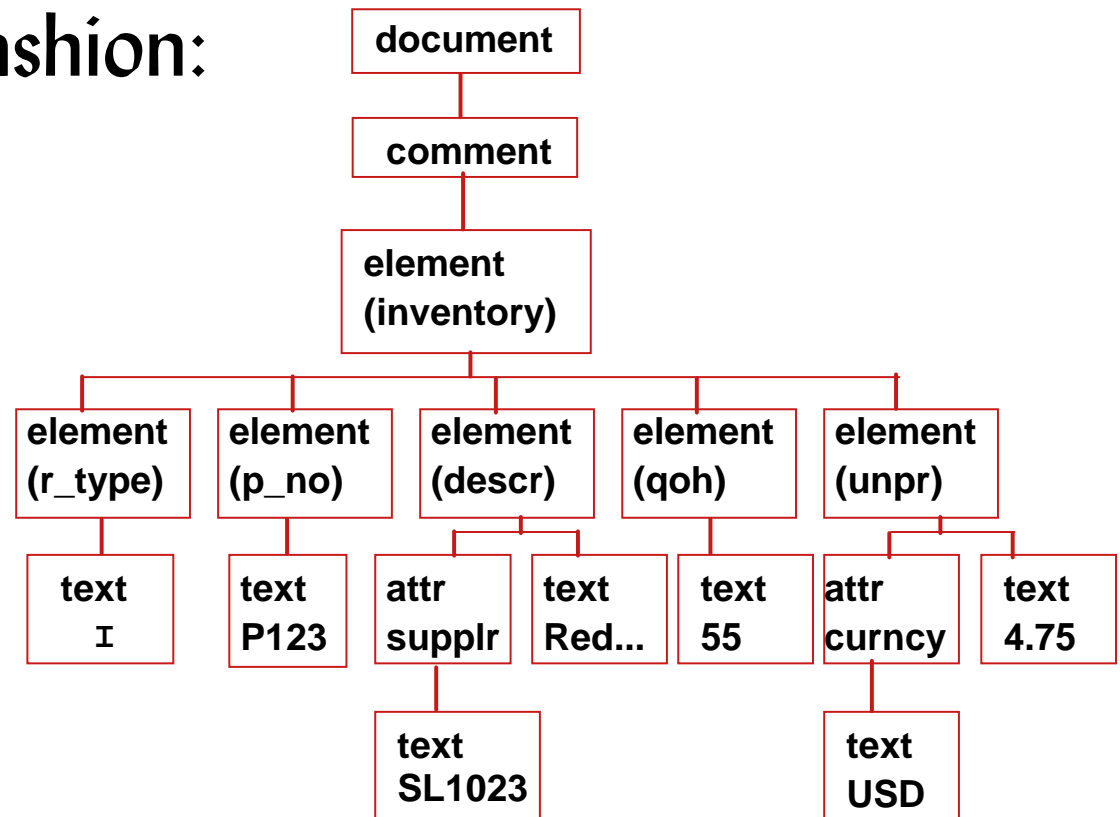
XML - a Brief Tutorial, 14

► XML documents are also often represented as a hierarchy of <u>nodes</u>, in a visual fashion:

```
<?xml version="1.0" ?>
<!-- sample inventory item -->
<inventory>
<r_type>I</r_type>
<p_no>P123</p_no>
<descr supplr="SL1023">
Red ketchup squirt bottles
</descr>
<qoh>55</qoh>
<unpr curncy="USD">4.75</unpr>
</inventory>
```
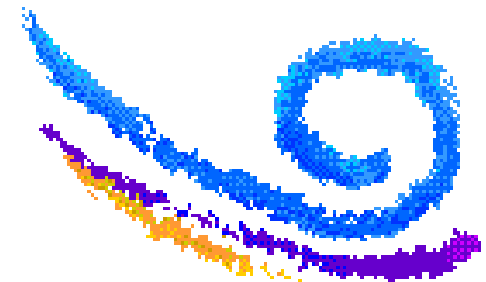
document

comment

element
(inventory)

element
(r_type)

element
(p_no)

element
(descr)

element
(qoh)

element
(unpr)

text
I

text
P123

attr
supplr

text
Red...

text
55

attr
curncy

text
4.75

text
SL1023

text
USD

Notes:

* The example has been simplified for space reasons
* The diagram shows document, comment, element, attr (attribute) and text nodes
* Other nodes are also recognized (PI, for example) but not discussed here
* Consider these terms: parent, child, string value
* A document <u>fragment</u> is a connected collection of nodes

73
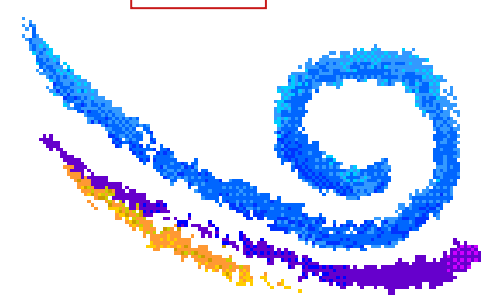
# XML - a Brief Tutorial, 15

► <u>XPath</u> is a language that provides expressions for locating a particular node in a document

- ■ It is used in a number of DB2 SQL statements
- ■ For example, to access a particular node (start tag, attributes, content,end tag) specify the name as: */root/node1/node2/.../node*
- ■ For instance: /inventory/p_no
- ■ To access the <u>value</u> of a text node, add text(), so: /inventory/p_no/text()

XML in DB2

► With DB2 V9.1, XML data is stored as a native data type

► When you create a table, include one or more columns of type XML, for example

```
CREATE TABLE INSTOCK
    (WAREHOUSE_NAME CHAR(60) NOT NULL,
     WAREHOUSE_ID CHAR(15) NOT NULL,
     ITEMS XML);
```
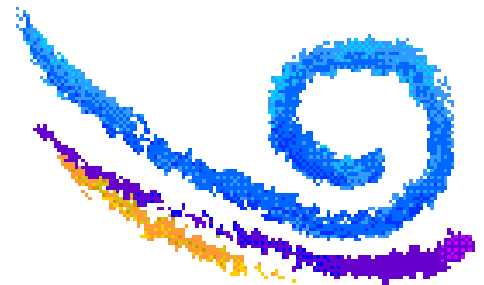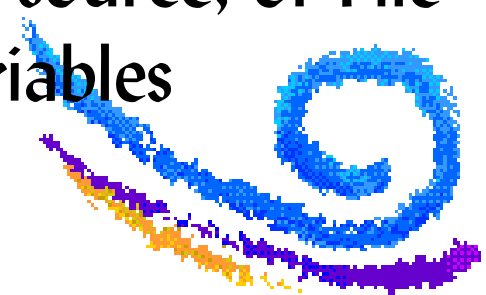
XML in DB2, 2

► XML columns, like LOB columns, require auxiliary objects (a hidden BIGINT column in the base table, a unique index on the hidden column, an XML table space, an XML table, and an index on the XML table)

► Unlike LOBs, however, these auxiliary objects are always created for you automatically: you couldn't define them manually if you wanted
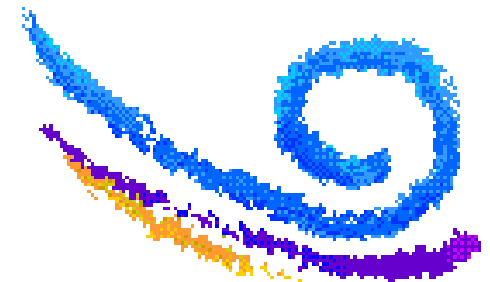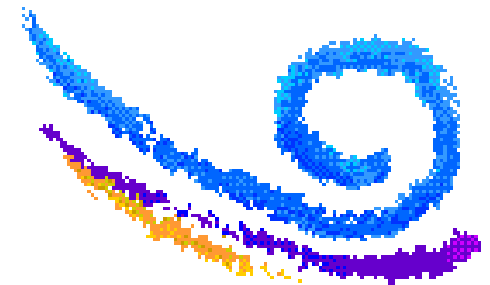
XML in DB2, 3

► Data in XML columns is stored in an internal format, and text in that format is stored in UTF-8

► An XML column stores a complete XML document

► Populate an XML column using SQL INSERT statements as for any column in any row

  ▪ Can populate from BLOB, CLOB, or DBCLOB source, or File Reference Variables (as with LOBs) or host variables
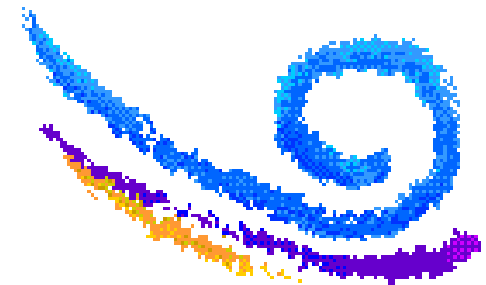
XML in DB2, 4

► Data in XML columns may be changed using UPDATE SQL statements

  ▪ However, you can only replace the entire XML document, not any of the individual components

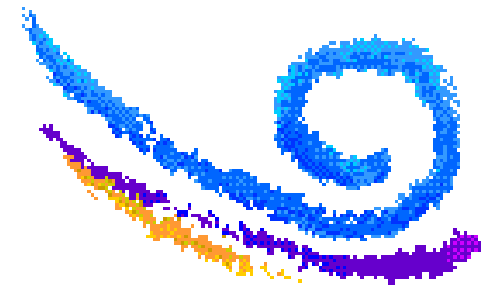► Most of the work done with XML columns requires using specialized built-in functions ...

XML in DB2, 5

► XMLPARSE(DOCUMENT *source* [{<u>STRIP</u>|PRESERVE} WHITESPACE]) - convert an XML document to an XML column value (internal format with text stored as UTF-8)

► XMLSerialize(*non-attribute-node-XML-expression*, AS [CLOB | DBCLOB | BLOB [{<u>1M</u> | *n*{K|M|G} }]] [{<u>EXCLUDING</u> | INCLUDING} XMLDECLARATION]) - returns a string represenation of the input expression

- Both of these are new with DB2 V9.1

XML in DB2, 6

► XMLQUERY(*XPath-expression* PASSING *column-name*) - returns an XML value found as a result of checking *column-name* for a value that matches *XPath-expression*

  ▪ New with DB2 V9.1

► Usually nest XMLQUERY within an XMLSerialize function

► Some examples on the following pages …

XML in DB2, 7

► To extract values from an element in an XML column from all rows:

- SELECT XMLSERIALIZE(XMLQUERY ('//descr/text()' PASSING ITEMS) AS CLOB(1k)) as Description FROM INSTOCK;
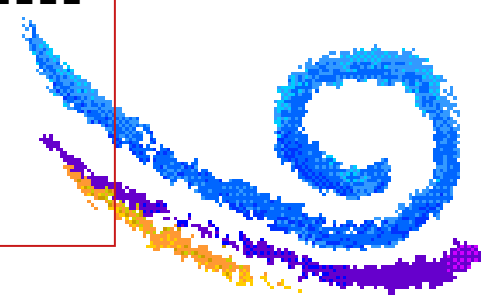
- Might return something like this:

```
Description
-------------------------------------------------
Condiment containers
Red ketchup squirt bottles
Buns and rolls
...
```
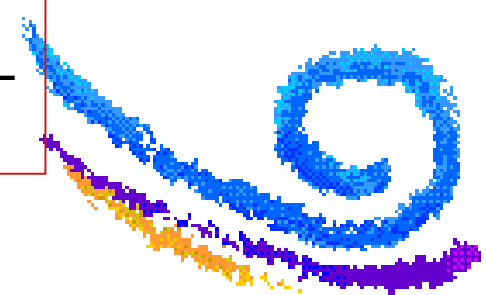
XML in DB2, 8

▶ To extract values from an element in an XML column from only those rows where the documents pass a test:

- SELECT XMLSERIALIZE(XMLQUERY ('//descr/text()' PASSING ITEMS) AS CLOB(1k)) as Description FROM INSTOCK WHERE XMLEXISTS('//qoh[.=55]' PASSING ITEMS);
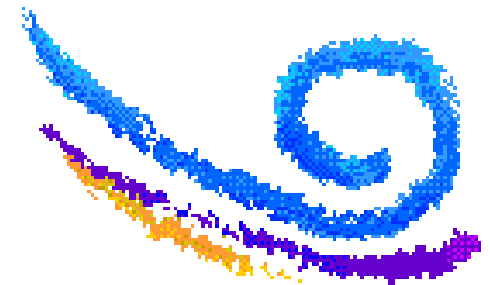
- Might return something like this:

```
Description

-------------------------------------------------

Red ketchup squirt bottles
```

XML in DB2, 9

► When using an XML column in a WHERE predicate, the only comparison allowed is XMLEXISTS

   ▪ The argument to XMLEXISTS is composed of an XPath expression and an <u>XQuery</u> test, as shown on the previous page

► XPath and XQuery support are new with DB2 V9

XML in DB2 - Why use it?

► XML documents have a hierarchical structure that lends itself to sophisticated content search within the context of SQL statements

► XML provides a rigorous way to structure and present data useable in many environments (self-defining, Unicode encoded)

► XML is a widely accepted standard used in a growing collection of contexts (XHTML, SOAP, WSDL, configuration files for products such as Eclipse and WebSphere Developer for z Series, etc.)

The Arc of DB2 - a Summary

► We have seen that the evolution of DB2 has followed a path that is leading to better integration of data in DB2 databases with web servers:

- <u>LOBs</u>, especially BLOBs, allow for various binary and media storage and serving
- <u>Unicode</u> supports encoding for web technologies
- <u>XML</u> provides the framework for communication to the web and on through to customer browsers

► DB2 Version 9.1 represents more progress along that curve - but that's not all …
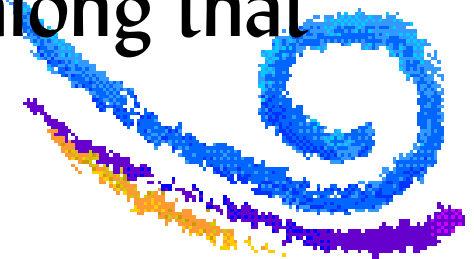
DB2 Version 9.1 - Other Goodies

► In addition to the enhancements we've noted in the three features under consideration, DB2 Version 9.1 provides these new or improved facilities:

► Universal Table Spaces (Simple Table Spaces are deprecated)

- Partition-by-growth: start with one partition and DB2 adds partitions as necessary

- Range-partitioned: partitioned by user-specified key ranges
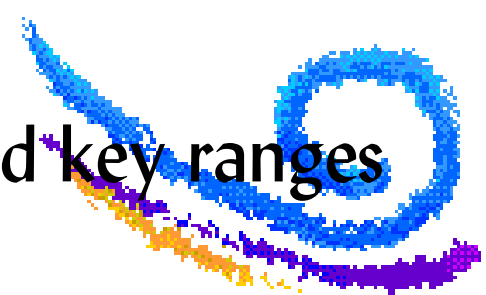
DB2 Version 9.1 - Other Goodies, 2

► zIIP exploitation (z9 and z10 machines)

► New column data types: BIGINT, BINARY, DECFLOAT, VARBINARY

► Implicit creation of supporting objects (table spaces, auxiliary tables, auxiliary indexes, primary key, unique key indexes)
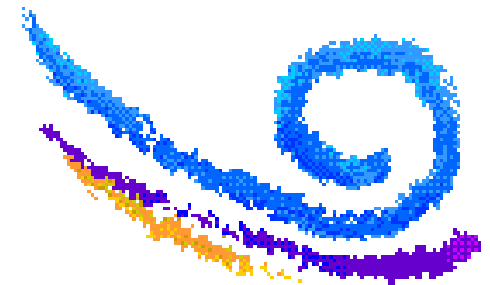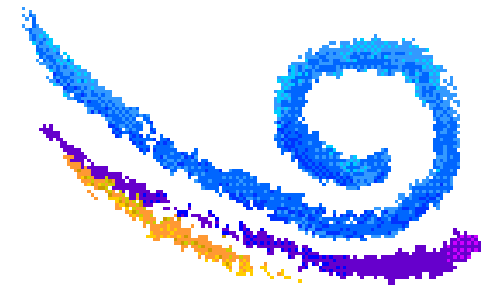
DB2 Version 9.1 - Other Goodies, 3

► New SQL statements / features

- TRUNCATE TABLE - delete all rows without activating delete triggers

- MERGE - if a transaction references an existing row, update it; if it references a non-existing row, insert it - all with one trip to DB2

- EXCEPT and INTERSECT - alternative ways to combine two (or more) queries
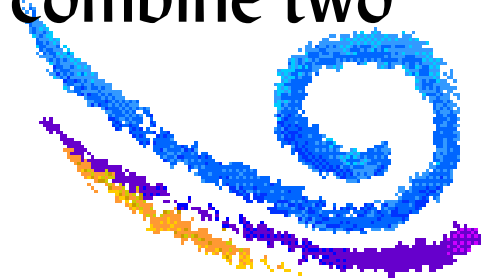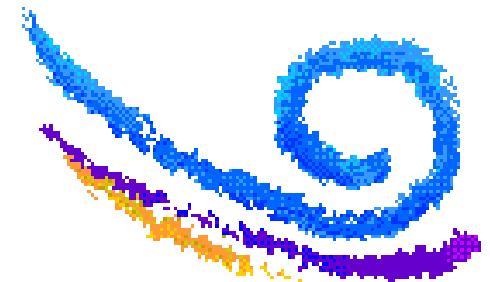
DB2 Version 9.1 - Other Goodies, 4

► New SQL statements / features

- SELECT ... FROM INSERT - INCLUDE additional columns
- SELECT ... FROM INSERT - ORDER BY INPUT SEQUENCE
- SELECT ... FROM UPDATE - select new value or old value or both
- SELECT ... FROM DELETE - select value(s) from deleted row(s)
- SELECT ... FROM MERGE - variations on the above

DB2 Version 9.1 - Other Goodies, 5

►New SQL statements / features

- INSTEAD OF triggers - definable on views, allows you to INSERT, UPDATE, or DELETE rows in the underlying table(s)

- FETCH WITH CONTINUE to access LOBs a buffer at a time

- SKIP LOCKED DATA allows skipping currently locked data

DB2 Version 9.1 - Other Goodies, 6

► New SQL Built-in Functions (a small sampling):

- EXTRACT - grab the YEAR, MONTH, DAY, HOUR, MINUTE, or SECOND from a date expression

- MONTHS_BETWEEN - two date expressions

- TIMESTAMPADD - timestamp result of adding *n* microseconds, seconds, minutes, hours, days, weeks, months, quarters, or years to a timestamp expression

- TIMESTAMPDIFF - return number of microseconds, seconds, minutes, hours, days, weeks, months, quarters, or years between two timestamp expressions

DB2 Version 9.1 - Other Goodies, 7

► New SQL Built-in Functions (a small sampling):

- TIMESTAMP_FORMAT - return a timestamp given a string expression and a format string that says how to interpret the expression

- TIMESTAMP_ISO - return a timestamp based on interpreting a timestamp, date, time, character string or graphic string

- SOUNDEX - return a 4 character code that represents the sound of the words in number, character string, or graphic string (!)

DB2 Version 9.1 - Other Goodies, 8

► Native SQL Stored Procedures

- Simplify coding and deploying stored procedures
  - No pre-compile to create a C program
  - No pre-compile, compile, link of resulting C program
  - No explicit BIND of the DBRM
  - No CREATE PROCEDURE naming the resulting load module
  - No defining a WLM application environment
  - No creating a cataloged procedure for the WLM AS
- Just CREATE PROCEDURE and define the procedure in SQL

93

DB2 Version 9.1 - Other Goodies, 9

► Native SQL Stored Procedures

- FOR statement - combine cursor definition with looping syntax

- Nested compound statements

- Supporting changes in catalog, DB2 commands, DSN subcommands
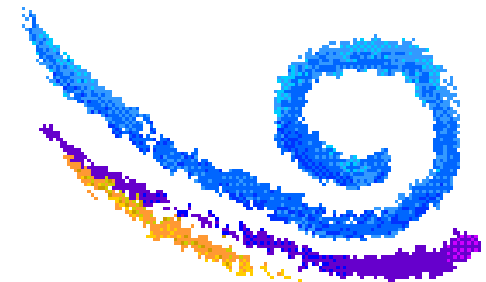
DB2 Version 9.1 - Other Goodies, 10

► Reordered Row Format - rows with variable length columns are now stored with all fixed length columns first, followed by offsets to beginning of each variable length column

  ◆ First REORG or LOAD REPLACE run under Version 9 for such a table will automatically include a reformating to this new style

► New column type: ROW CHANGE TIMESTAMP

  ▪ Updated by DB2 every time a row is INSERTed or UPDATEd

DB2 Version 9.1 - Getting There

► As with all major releases of DB2, Version 9.1 has a multi-step migration process that lets you move forward then fall back at various points in time

► Requirements: start from

  ▪ z/OS 1.7 or later

  ▪ DB V8 in New Function Mode (NFM)
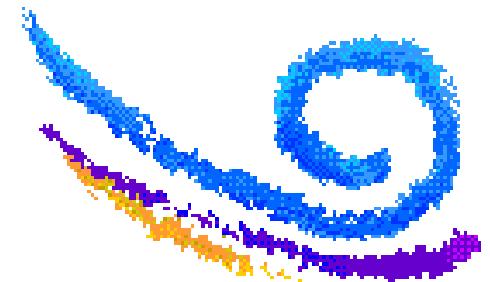
DB2 Version 9.1 - Getting There, 2

► Big Picture of the steps:

- Migrate to V9, Compatibility Mode (CM), test for some period of time (typically 1-3 months)

- Migrate to V9, Enabling New Function Mode (ENFM), test for a short time (typically a few minutes)
  - Can fallback to V9 partial CM if need to

- Migrate to V9, New Function Mode (NFM)
  - Fallback to V9 partial ENFM or V9 partial CM is possible, but cannot fallback to V8

The Arc of DB2 - References

DB2 V9.1 IBM manuals found at:
  http://www-03.ibm.com/systems/z/os/zos/bkserv/zswpdf/#db2v91

IBM Redbooks on DB2:
  http://publib-b.boulder.ibm.com/abstracts/sg246826.html?Open
  http://publib-b.boulder.ibm.com/abstracts/sg247427.html?Open
  http://publib-b.boulder.ibm.com/abstracts/sg247330.html?Open

Unicode standard
  http://www.unicode.org/

World Wide Web Consorition (W3C)
  http://www.w3.org/ (follow links here to XML, HTML, etc.)

Free technical papers from The Trainer's Friend
  http://www.trainersfriend.com/General_content/Book_site.htm
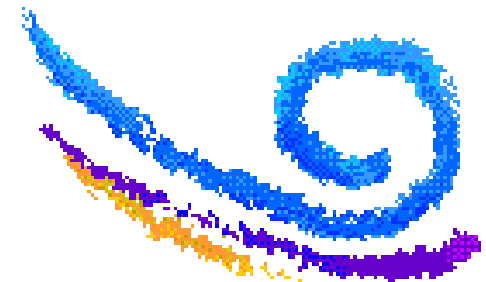Includes "Introduction to Unicode"
        "Introduction to XSLT"
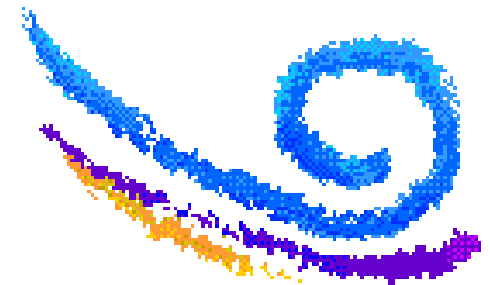   "Hosting a Web Site on z/OS"
and more

Training on DB2 and other technologies discussed here
  http://www.trainersfriend.com

The Arc of DB2 - Conclusion

► DB2 Version 9.1 is a significant release of this flagship product, with lots of useful enhancements, that clearly continues the arc of DB2 to support enterprise-quality relational database in z/OS and beyond

► Questions, time permitting

The Trainer's Friend
INCORPORATED

6790 East Cedar Avenue, Suite 201
Denver, Colorado 80224
USA

http://www.trainersfriend.com
303.393.8716
Sales: kitty@trainersfriend.com
Technical: steve@trainersfriend.com