



# Hardcore Universal Database for iSeries

Jarek Miszczyk

PartnerWorld for Developers, iSeries

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Trademarks

IBM @server iSeries

Copyright International Business Machines Corporation 1999

References in this document to IBM products or services do not imply that IBM intends to make them available in every country. The following terms are trademarks or registered trademarks of the IBM Corporation in the United States or other countries or both:

ADSTAR	DataGuide	NetFinity
Advanced Function Printing	DataPropagator	NetView
AFP	DB2	OfficeVision
AIX	IBM	OS/2
AnyNet	IBM Network Station	OS/400
Application Development	Information Warehouse	PowerPC
APPN	Integrated Language Environment	PowerPC AS
AS/400	Intelligent Printer Data Stream	Print Services Facility
AT	IPDS	PSF
BrioQuery	JustMail	SanFrancisco
BRMS	Net.Commerce	SmoothStart
Client Series	Net.Data	SystemView

cc:Mail, Lotus, Lotus Notes, Lotus Domino, Domino.Action, and Domino.Merchant are trademarks or registered trademarks of Lotus Development Corporation in the United States or other countries or both.

Microsoft, Windows, Windows NT, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Java and all Java-related trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

IBM's VisualAge products and services are not associated with or sponsored by Visual Edge Software, Ltd.

Intel and Pentium are trademarks of Intel Corporation in the United States and other countries.

Tivoli is a registered trademark of Tivoli Systems Inc. in the United States or other countries or both.

Other company, product, and service names may be trademarks or service marks of others.

Information in this presentation concerning non-IBM products was obtained from a supplier of these products, published announcement material, or other publicly available sources. Sources for non-IBM list prices and performance numbers are taken from publicly available information, including vendor announcements and vendor worldwide homepages. IBM has not tested these products and cannot confirm the accuracy of performance, capability, or any other claims related to non-IBM products. Questions on the capability of non-IBM products should be addressed to the supplier of those products.

All statements regarding IBM future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. Contact your local IBM office or IBM authorized reseller for the full text of the specific statement of direction.

Any performance data contained in this document was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements quoted in this presentation may have been made on development level systems. There is no guarantee these measurements will be the same on generally available systems. Some measurements quoted in this presentation may have been estimated through extrapolation. Actual results may vary. Users of this presentation should verify the applicable data for their specific environment. Customer examples cited are examples of how the referenced customers use IBM and other products. Results vary by environment and may not be realized in all situations.

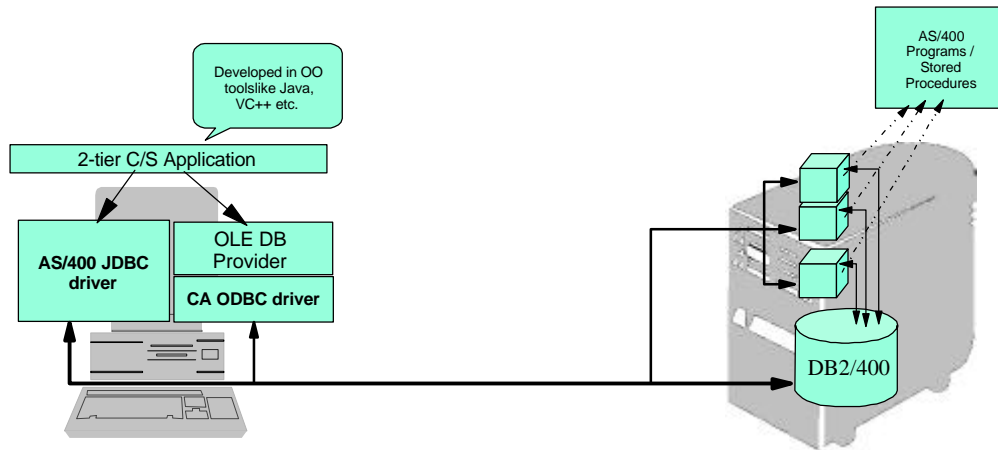
THIS MATERIAL IS PROVIDED 'AS IS' WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. IN NO EVENT WILL IBM BE LIABLE TO ANY PARTY FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES FOR ANY USE OF THIS MATERIAL INCLUDING, WITHOUT LIMITATION, ANY LOST PROFITS, BUSINESS INTERRUPTION, LOSS OF PROGRAMS OR OTHER DATA ON YOUR INFORMATION HANDLING SYSTEM OR OTHERWISE, EVEN IF WE ARE EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Functions and availability dates may change after this presentation was completed.

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## 2-tier Application Architecture

IBM @server iSeries



IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

IBM @server iSeries

### Advantages

Better application performance due to fewer layers

Processing logic split between client and server tier, with server doing all database tasks.

Extremely good performance with record level access and most of all commitment control works just like on the AS/400.

### Disadvantages

Complexity of the application design increases as considerable code needs to be written at both the tiers for a good performing concurrent user application.

Managing such an application may become cumbersome on the long run.

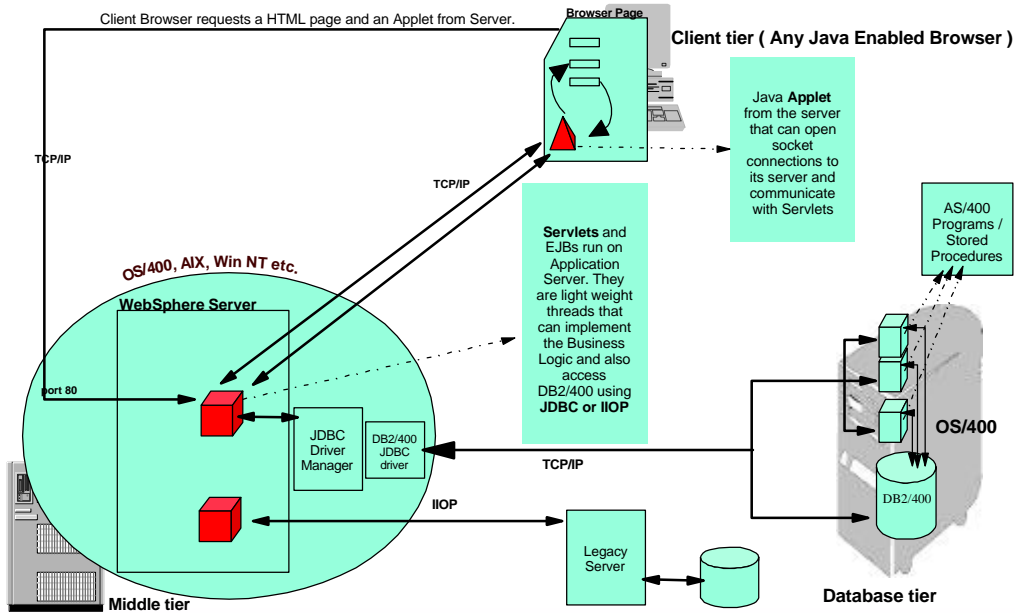
If lot of application logic is placed on the client, efficient DB2/400 features can not be used.

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

# 3-tier application using Java

IBM @server iSeries



IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

IBM @server iSeries

The first tier of such an application could use any number of Java enabled browsers, including ones running on Network Computers (NCs) as well as ones built on more traditional platforms such as personal computers or workstations. Complex user interface tasks would be handled by Java applets downloaded from the second tier servers; simpler tasks could be handled using standard HTML forms.

The second tier of such a system would consist of servlets which encapsulate the particular business rules and logic of the application at hand, perhaps arranging to deliver a new shipment of parts. Such rules could include application-specific access controls for sensitive (but unclassified) corporate data.

Servlets may be used to connect the second tier of an application to the first tier. A variety of client technologies may be used to connect to other tiers.

The third tier of such systems consists of data repositories. This tier might be accessed using relational database interfaces such as JDBC, or other interfaces supported by legacy data. This includes RPC-like protocols such as ONC RPC, DCE RPC, and CORBA/IIOIP.

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Need for Large Objects (LOBs)

IBM @server iSeries

- Provide support for Web-based multimedia applications
- Provide a built-in data type that can store large amount of data

SOLD	ONHAND	RATING	ARTIST	TITLE	COVER	VIDEO	MUSIC	SCRIPT
234	59	PG-13	Arnold	The Exterminator				
13	45	R	Kevin	Dancing with Bulls				
1295	209	G	Glenn	101 Doll Imitations				
379	112	G	Buzz	Toy Glory				

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

IBM @server iSeries

With LOB support, the AS/400 Database is able to store and manipulate data fields that are much larger than the current limits. Whereas in the previous versions each record in a database file could hold up to 32K bytes, an AS/400 LOB field is able to hold up to 15 Megabytes of data. It is through the LOB support that the database establishes a tight tie between traditional character/numeric data and data that occupies large amounts of storage. Today's multimedia applications depend on storage of many types of large data objects, such as scanned documents, medical images, and audio messages.

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Average size of LOB Objects

IBM @server iSeries

OBJECT	FROM	TO
Bank Check	45K	-
Text	30KB per page	40KB per page
Small Image	30KB	40KB
Large image	200KB	3MB
Colour Image	20MB	40MB
Radiology Image	40MB	60MB
Video	1GB per hour	-
Feature Length Movie	2GB	-
High Resolution Video	3GB per hour	-
High Resolution Movie	5GB	6GB
High Defination TV	200MB per second	-

IBM @server. For the next generation of e-business.

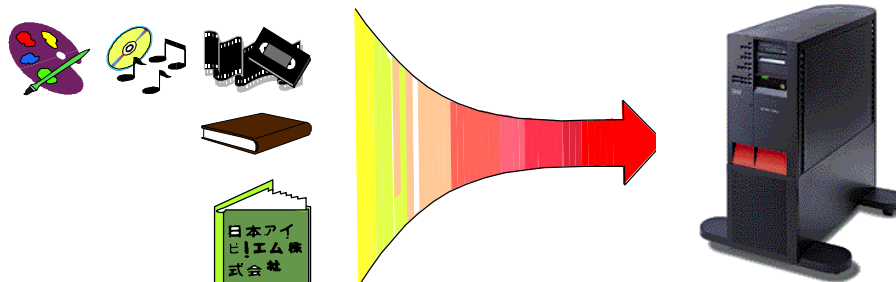
© 2000 IBM Corporation

## LOB Types

IBM @server iSeries

### Three different types of data

- **BLOB (Binary Large Object)**
  - For Scanned documents, digital images, etc.
- **CLOB (Character Large Object)**
  - Large character string data type
- **DBCLOB (Double Byte Character Large Object)**
  - Large double-byte character string data type



IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

IBM  server iSeries

A LOB field holds a string of ordered bytes and can vary from zero to 2 Gigabytes in length. They are an extension of the current character data types. There are three kinds of LOB fields:

- BLOB - Binary Large Object - A LOB field whose CCSID is set to 65535 and consequently requires no CCSID conversions at I/O time.

Other character types in the same 'family' as BLOBs are:

- VARCHAR -- FOR BIT DATA
- CHAR -- FOR BIT DATA

- CLOB - Character Large Object - A LOB field that is tagged with a single byte or mixed CCSID value.

Other character types in the same 'family' as CLOBs are:

- VARCHAR -- FOR SBCS DATA
- VARCHAR -- FOR MIXED DATA
- CHAR -- FOR SBCS DATA
- CHAR -- FOR MIXED DATA

- DBCLOB - Double Byte Character Large Object - A LOB field that is tagged with a double byte or UCS2 CCSID.

Other character types in the same 'family' as DBCLOBs are:

- VARGRAPHIC
- GRAPHIC

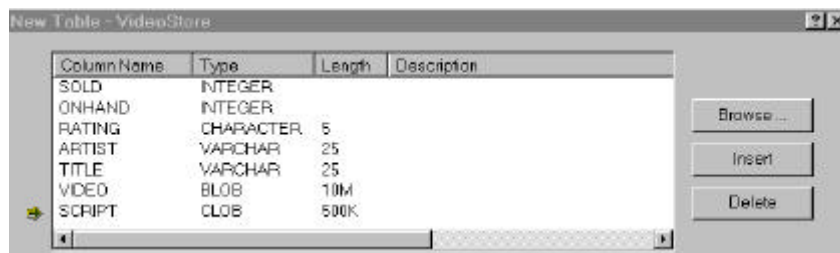
**Note:** Any operation that combines a LOB type along with any of the other character types will always return a result that is a LOB.

IBM  server. For the next generation of e-business.

© 2000 IBM Corporation

## Using LOB Columns

IBM  server iSeries



Column Name	Type	Length	Description
SOLD	INTEGER		
ONHAND	INTEGER		
RATING	CHARACTER	5	
ARTIST	VARCHAR	25	
TITLE	VARCHAR	25	
VIDEO	BLOB	10M	
SCRIPT	CLOB	500K	

- Size must fall in the range of 1 to 2 Gigabytes

**BLOB (15728640)**

**CLOB (10000)**

- Shorthand provided with K & M notation

K = size value \* 1024 (number of kilobytes)

M = size value \* 1048576 (number of megabytes)

G = size value \* 1073741824 (number of Gigabytes)

**BLOB (10M)**

**DBCLOB (300K) {# of DB characters}**

- ALLOCATE can be used similar to VARCHAR columns

- NO DDS support available

IBM  server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

IBM @server iSeries

Although DB2 UDB, and nearly every other database, provides the ability to declare LOB columns with a maximum size of 2 Gig, we provide the ability to declare LOB columns with a maximum size of 15 Meg. This magnitude difference in size is mitigated by the recognition that most LOB users store small LOBs rather than large LOBs. For example: signatures, bank checks, a page of text, and both small and large images all occupy between 30K and 5 Meg.

ALLOCATE specifies for the LOB types the space to be reserved for the column in each row. Column values with lengths less than or equal to the allocated value are stored in the fixed-length portion of the row. Column values with lengths greater than the allocated value are stored in the variable-length portion of the row and require additional input/output operations to retrieve. The allocated value may range from 1 to maximum length of the string, subject to the maximum record buffer size limit.

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## SQL Functions Supporting LOBs

IBM @server iSeries

### ● Basic Predicates (=, <, >, ^=, <>, <=, ^>, >=, ^<)

- Create a dummy table to test predicates

```
create table dbteamxx/dummy (col1 int);
insert into dummy values( 1 );
```

- Comparing CLOB values

```
/* TRUE - trailing spaces are removed */
select case when count(*) = 1 then 'TRUE' else 'FALSE' end case
from dummy where clob('abc') = clob('abc ');
```

```
/* FALSE - leading spaces are not removed */
select case when count(*) = 1 then 'TRUE' else 'FALSE' end case
from dummy where clob('abc') = clob(' abc ');
```

```
/* TRUE - trim removes unwanted spaces */
select case when count(*) = 1 then 'TRUE' else 'FALSE' end case
from dummy where clob('abc') = trim(clob(' abc '));
```

- Comparing BLOB values

```
/* FALSE - the length is not equal */
select case when count(*) = 1 then 'TRUE' else 'FALSE' end case
from dummy where blob(X'123456') > blob(X'1234');
```

```
/* TRUE - the BLOB values are different */
select case when count(*) = 1 then 'TRUE' else 'FALSE' end case
from dummy where blob(X'123456') <> blob(X'1234');
```

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

---

IBM @server iSeries

When comparisons are made between CLOB data, the database runtime removes the trailing spaces first and then compares the character values. Note that the leading spaces are not trimmed.

When comparisons are made between BLOB data types, the first comparison is of their length. Once the lengths are found to be equal, a comparison of the binary contents is made. If it is not equal, no further comparison is made. It applies the predicate to test their binary content only if the database finds out that the length of the objects is equal. For example, the following expression is evaluated as FALSE:

```
blob(X'123456') > blob(X'1234')
```

Because the lengths of the two BLOB objects are different, no further comparison is made. In other words, blob(X'123456') is neither greater, nor equal, nor smaller than blob ('1234'). Because it's different, only the '<>' predicate is evaluated as TRUE.

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## SQL Functions Supporting LOBs

---

IBM @server iSeries

- Column Functions (COUNT(), COUNT ALL())  

```
select count(*) as Count from CUSTOMERHUS  
where House_Picture = blob(X'1234');
```
- Existing Scalar Functions (CHAR(), CONCAT(), LENGTH(), MIN(), MAX(), SUBSTR(), TRIM() and more ... )  

```
select max(clob('abcd'), clob('abc')) from dummy;  
select length(clob('abcd')) from dummy;  
select posstr(clob('DB2 UDB for AS/400'), 'UDB') from dummy;  
select substr(clob('DB2 UDB for AS/400'), 5, 3) from dummy;  
/* The result is of type CLOB */  
select concat(clob('DB2 UDB '), 'for AS/400') from dummy;
```

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## New SQL Functions for LOBs

---

IBM  server iSeries

### New Scalar Functions:

- **BLOB (string-expression)**

returns a BLOB representation of a string of any type

```
CREATE TABLE custtext (custid INT, sign BLOB(20K), cv CLOB(5K) CCSID 37);
```

```
INSERT INTO custtext VALUES(1, BLOB('text'), 'some more text');
```

```
SELECT * FROM custtext WHERE sign = blob('text');
```

- **CLOB (string-expression)**

returns a CLOB string representation of a character or graphic string

```
SELECT * FROM custtext WHERE cv = clob('some more text',DEFAULT ,37);
```

- **DBCLOB (graphic expression)**

returns a DBCLOB representation of graphic string

- **LOWER/LCASE (expression)**

returns a character string representation of input character string where all A-Z converted to a-z

IBM  server. For the next generation of e-business.

© 2000 IBM Corporation

## NOTE:

---

IBM  server iSeries

LIKE can only be used with a constant.  
IN does not work at this time.

IBM  server. For the next generation of e-business.

© 2000 IBM Corporation

## LOB Column Considerations

IBM @server iSeries

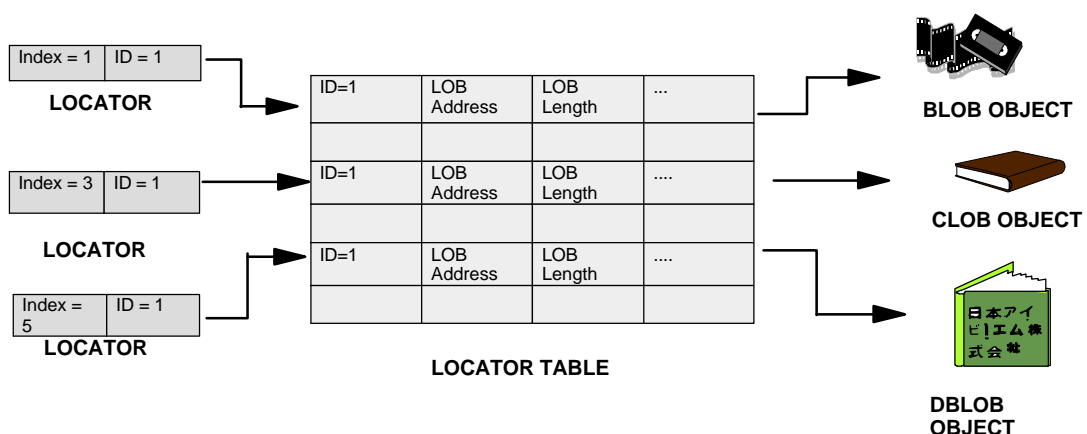
- 15 Megabyte limit for the total size of all columns in a single row (BLOB, CLOB, & DBCLOB)
  - Max size for "fixed" portion is still 32K
  - ALLOCATE clause provides better performance, if the average size for LOB fits in "fixed" 32K portion  
`CREATE TABLE custtext (custid INT, pict BLOB(1M) ALLOCATE(20000) );`
- Columns defined with CLOB, DBCLOB, or BLOB not allowed:
  - As key field in an index  
`/* Expect SQLCODE = -350 */  
CREATE INDEX custidx ON custtext (pict);`
  - In the GROUP BY or ORDER BY clause
  - As UNIQUE, PRIMARY KEY, and FOREIGN KEY  
`/* Expect SQLCODE = -350 */  
ALTER TABLE custtext ADD COLUMN cv CLOB(20K) PRIMARY KEY;`

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Large Objects - Locators

IBM @server iSeries



- Reduce data transfer
- Allow deferring the evaluation of LOB expression for performance

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

---

IBM @server iSeries

Because LOB fields may occupy magnitudes of storage more than our current maximum record lengths, we need a mechanism for referencing the data in these fields without moving the actual bytes until absolutely necessary. For example, say a user wants to read a LOB value from one file, and update a second file with that value. A poorer performing implementation would copy the LOB value into a separate space at read time, and then update the second file using this copy as our update image. A better performing implementation would be to defer data movement until the update operation itself. A LOB Locator - sometimes called a 'handle' - is intended to refer to the data we are manipulating without actually moving the data until the results are assigned to another field or variable. A LOB Locator gives read only access to the data it addresses - a user will not be able to change a field addressed via a LOB Locator by an operation on the locator itself. In our example, if the user were to select the LOB value into a LOB Locator rather than buffer area, we would set the locator so it would reference the actual data in the file, rather than copying the data from the file into a buffer area.

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Power of Locators

---

IBM @server iSeries

### LOB Locators characteristic

- Represent a constant values
  - User cannot change the underlying LOB data
  - Address LOB columns or host variables
- **Allowed wherever a character string expression can be used**
  - Built-in functions (POSSTR, SUBSTR, ...)
  - User-Defined Functions
  - Host variables
- **Can be 'combined' with other character types**
  - Resulting data type will always be a LOB type
- **Are not a database type**
  - Database field of type LOB Locator not allowed

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

IBM  server iSeries

The following are characteristics of LOB Locators:

- They give access to LOB data, but a user cannot change the LOB data by issuing an operation against the LOB Locator itself.
- At fetch time, a LOB Locator can be set to address a LOB field's data. However, unlike a variable of any other type, when a LOB Locator is set equal to a field, the data itself is not moved into a buffer area - instead, the locator is given 'addressability' to the data. Still, just as a variable of any other data type provides the user access to a value after being set, regardless of whether or not the cursor has been repositioned, so the LOB Locator should give the user access to the LOB data even after the cursor which was originally read from has been repositioned, or even closed.
- A LOB Locator can be used wherever a character string expression can be used, such as in SQL host variables, built-in functions, or User Defined Functions.
- They can 'address' Large Object columns from a table, or Large Object variables from within an SQL program.
- They can also be 'combined with' any of the other character types: CHAR, VARCHAR, LONG VARCHAR, etc. However, the resulting type will always be a LOB type.
- They are not a Database type - we cannot create a field in a file that is of type LOB Locator.
- A FREE LOCATOR statement will release the locator from addressing the string it was referring to. After the FREE statement, the locator can no longer view the data that it was referencing.
- A LOB Locator can reference a Null LOB field. The locator itself will not be null; instead an indicator variable will tell that the field being referenced is null.

IBM  server. For the next generation of e-business.

© 2000 IBM Corporation

## LOB Locators - Coding

IBM  server iSeries

### ● Locator declaration

```
EXEC SQL BEGIN DECLARE SECTION;  
SQL TYPE IS CLOB(20K) prod01;  
SQL TYPE IS CLOB(20K) strpro;  
SQL TYPE IS CLOB_LOCATOR clob1;  
SQL TYPE IS CLOB_LOCATOR clob2;  
long posbeg;  
long posutil;  
long len;
```

```
EXEC SQL END DECLARE SECTION;
```

### ● Read a CLOB value into a locator

```
EXEC SQL SELECT House_Description INTO :clob1  
FROM CUSTOMERHUS  
WHERE Customer_Number = '00002';
```

### ● Manipulate the CLOB using the locator

```
EXEC SQL VALUES posstr(:clob1, 'Garden:') INTO :posbeg;  
EXEC SQL VALUES char_length(:clob1) INTO :len;  
EXEC SQL VALUES posstr(:clob1, 'Description') INTO :posutil;
```

IBM  server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

IBM @server iSeries

A locator variable is a variable that is declared in the SQL DECLARE Section of an application program to hold a locator. In each host programming language, there is a datatype designated for locator variables. For example, in C, the type type of a locator is long. However, locator variables must be distinguished from long variables used for manipulation of Integer data. A special is used to notify the precompiler of the intended use of a locator variable.

The VALUE variable statement produces a result table consisting of at most one row and assigns the values in that row to host variables. This statement can only be embedded in an application program. It is an executable statement that cannot be dynamically prepared.

You learn how to use the locator with the SQL functions POSSTR and CHAR\_LENGTH. The locator is used to manipulate the CLOB data without materializing the object in the application buffer. Only a small subset of the input CLOB is assigned to the new CLOB locator

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## LOB Locators - Coding

IBM @server iSeries

- Cast the locator to the CLOB when concatenating

EXEC SQL

```
VALUES concat('Swimming Pool:X. ',  
substr(cast(:clob1 as clob(200k)), :posutil, :len - 105))  
INTO :prod01;
```

- Materialize the manipulated CLOBs into a new CLOB object

EXEC SQL

```
VALUES concat (cast(:strpro as clob(200k)), cast(:prod01 as clob (200k)))  
INTO :clob2;
```

- Save the new CLOB object in the table

```
EXEC SQL INSERT INTO CUSTOMERHUS VALUES ('77777', blob(X'7777'),  
:clob2);
```

- Commit to release locks held

```
EXEC SQL COMMIT WORK;
```

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

---

IBM @server iSeries

The highlighted cast is required since the database manager does not know the size of the locator result when it is validating the concat( ..., substr(..)) operation. First, the database manager tries to determine the result size of the substring. Because there is no size associated with any LOB Locator, and because the substring in this program uses host variables rather than constants, the database cannot assess the size of the locator operand at validation time. Consequently, it chooses the maximum LOB size. After choosing the max LOB size, the concatenation of even a single byte literal along with the substring result will result in exceeding the maximum result size limit. One way of getting around this is to cast the CLOB Locator to a CLOB of a defined length. By doing this, the validation code is able to use 200 KB as the maximum size of the result rather than 15 MB.

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## LOB Locators - Compiling

---

IBM @server iSeries

- How to compile

`CRTSQLCI OBJ(DBTEAMXX/LOBLOCLB2) COMMIT(*ALL)`

`CRTPGM PGM(DBTEAMXX/LOBLOCLB2) MODULE(DBTEAMXX/LOBLOCLB2)`

- Isolation Level Considerations

- \*NONE not allowed

- \*ALL recommended for performance reasons

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

---

IBM @server iSeries

The commit level of \*NONE is not allowed for programs using LOB locators, because DB2 UDB for AS/400 implementation requires the commitment control to cleanup the internal structures used to keep track of the locators. We recommend that you use the commit level of \*ALL for programs using LOB locators if you want to achieve best performance. DB2 UDB for AS/400 doesn't have to create a copy of the LOB data when running under this isolation level. However, the down side of using this setting is a more restricted concurrent access to the underlying tables.

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

---

IBM @server iSeries

## User-defined Distinct Types (UDTs)

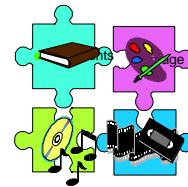
IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

# Need for User-Defined Types

IBM @server iSeries

- Provide means of defining data in business terms:
  - Extend types provided by DB2 for AS/400
  - Encapsulate business semantics into database
- In combination with User-Defined Functions, establish consistent behavior of data types
  - Strong typing
  - Casting
- Highly integrated into database manager for performance
  - Internally represented as built-in types
  - Share same efficient code used to implement built-in functions, comparison operators, indexes, etc. for built-in data types.
- Provides foundation for object-oriented extensions
  - Complex data types still to come
- Compatible with the ANSI SQL-99 Standard



IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

IBM @server iSeries

DB2 for AS/400 provides a range of built-in datatypes, which include the basic datatypes like INTEGER, DECIMAL, CHAR plus the large object datatypes discussed in the previous section of this presentation (BLOB, CLOB, DBLOB). Your database design may, however, require that you use one of the built-in datatypes in a specialized way. You may use for example DECIMAL(11,2) datatype to represent amounts of money. From the database semantic point of view it makes sense to add and subtract two amounts of money but it probably makes no sense to multiply two amounts of money.

DB2 for AS/400 provides a way to declare such specialized usages of datatypes and the rules that go with them in a form of *Distinct Type*. Distinct data types are user defined types that are derived from existing types (e.g. predefined types like INTEGER, DECIMAL, CLOB). They share the same representation with the types they are derived from, but because they are incompatible types, they can have quite different semantics.

The most important characteristics of User-Defined Types include:

- Strong typing

Strong typing insures that your UDTs will behave appropriately. It guarantees that only functions defined on your UDT can be applied to instances of the UDT.

- Casting

Casting from a distinct type to its source type and vice versa is allowed.

- Performance.

Distinct types are highly integrated into the database manager. Because distinct types are internally represented the same way as built-in data types, they share the same efficient code used to implement built-in functions, comparison operators, indexes, etc. for built-in data types.

- Foundation for object-oriented extensions.

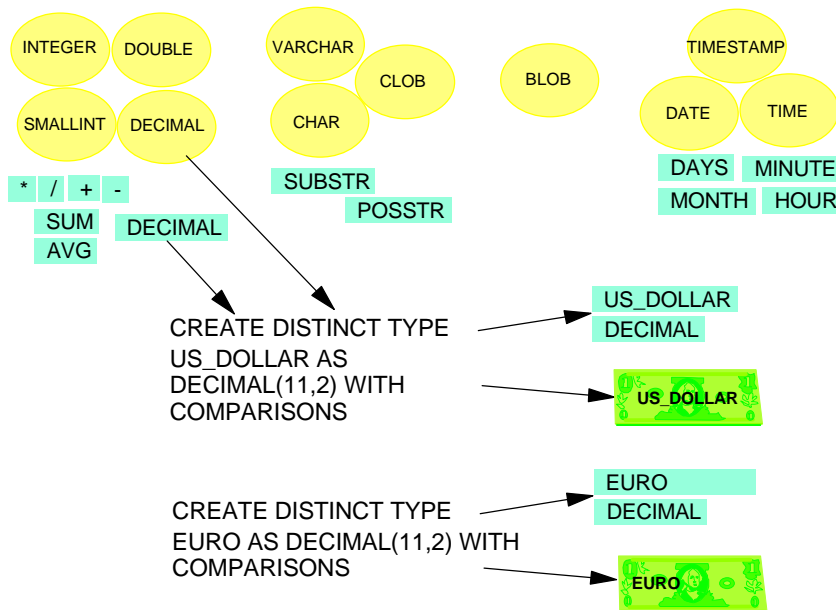
UDTs are the foundation for most object-oriented features. They represent the most important step towards object-oriented extensions.

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

# User-Defined Types - Example

IBM @server iSeries



IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

IBM @server iSeries

Once created, a distinct type can be used for the definition of columns in the CREATE TABLE statement or in the ALTER TABLE statement or for the definition of parameters of user-defined functions.

Only instances of the same distinct type can be compared with each other. Following the same line, operations and functions defined on the source type cannot be applied to the distinct type. Note that distinct types cannot be used as arguments of built-in functions and operators like +, -, \*, / etc.

Default functions to support the relational operators are automatically generated by the system whenever a distinct type is created. For example, the following statement:

```
CREATE DISTINCT TYPE euro AS DECIMAL(11,2) WITH COMPARISONS
```

creates the distinct type euro and functions to support all the relational operators (=, <>, >, <, <=, >=) on two instances of euro.

IBM @server. For the next generation of e-business.

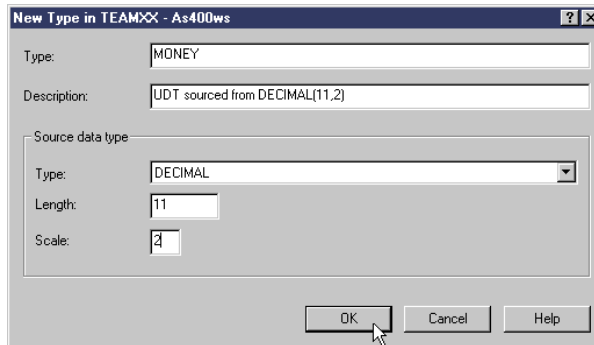
© 2000 IBM Corporation

# Creating User-defined Distinct Types

IBM @server iSeries

## Operations Navigator

- Select **New -> Type** from Library context menu
- New Type dialog
  - Type name
  - Description
  - Source type



## SQL

- CREATE [DISTINCT] TYPE distinct-type-name AS source-data-type [WITH COMPARISONS]
  - Type name
  - Source type
  - WITH COMPARISONS default
- COMMENT ON DISTINCT TYPE type-name IS string-constant
  - Description

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

IBM @server iSeries

Note that the WITH COMPARISONS option is the default so the system will generate comparison operators whether or not it is specified. It should be specified for compatibility with other DB2 products.

IBM @server. For the next generation of e-business.

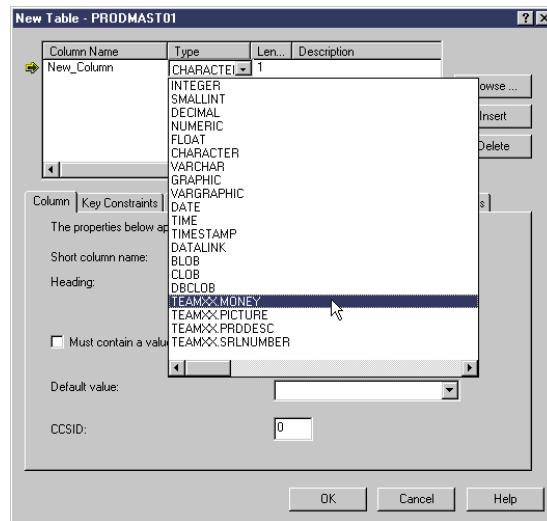
© 2000 IBM Corporation

## Using Distinct Types in a Table

IBM @server iSeries

### Operations Navigator

- Select **New -> Table** from Library context menu
- Select distinct type from Column type list



### SQL

- CREATE TABLE table-name column-name FOR COLUMN system-column-name data-type ...
- CREATE TABLE PRODMAST01 PRODUCT\_PRICE FOR COLUMN PMPRIC MONEY ...

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Casting for Distinct Types

IBM @server iSeries

- Casting functions are automatically defined when distinct type is defined
  - Enable converting from sourced type to distinct type and vice-versa
  - Implemented with user-defined functions
- CREATE DISTINCT TYPE money AS DECIMAL(11,2) WITH COMPARISONS;  
*Three casting functions also created with new type:*  
FUNCTION money( decimal ) RETURNS money  
FUNCTION decimal( money ) RETURNS decimal  
FUNCTION numeric( money ) RETURNS numeric
- Explicit casting
  - Using casting functions
- Implicit casting
  - Allows database manager to perform cast in certain situations
- Promotion of data types
  - Allows implicit casting of additional data types

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

IBM @server iSeries

Casting functions registered by database manager when creating a new type

- QSYS2/SYSROUTINES for function details
- QSYS2/SYSPARMS for function parameter details

Casting from a distinct type to its source type and vice versa is allowed. Cast functions are automatically generated whenever a distinct type is defined. Using these casting functions, you can freely convert a distinct type value to its source type and vice versa.

Casting can occur implicitly or explicitly. Casting occurs explicitly when you use casting functions to cast a data type. Casting occurs implicitly when the database manager recognizes that an automatic cast is allowed in certain situations.

Promotion of data types allows the database manager to consider additional data type when performing implicit casts, based on the precedence of data types.

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Explicit Casting

IBM @server iSeries

### Explicit casting must be done for functions, operations, and comparisons

#### Use casting function to convert distinct type to the source type

```
select decimal(product_price) * 1.1 as "AFTER_TAX_PRICE" from prodmast01;  
select product_price from prodmast01 where product_price < money(500.00);  
select product_number, product_description from prodmast01 where  
posstr(clob(product_description), 'moguls') <> 0;
```

#### Define a user defined function that supports the distinct type as an argument or operand (sourced UDFs)

```
CREATE FUNCTION avg (money) RETURNS money  
SOURCE "qsys2".avg (DECIMAL(11,2))
```

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

IBM @server iSeries

Strong typing requires that distinct types be explicitly cast when using:

- built-in functions and operators;
- basic predicate comparisons, such as "=", "<", ">", involving different types;
- other predicate comparisons, such as BETWEEN, IN, LIKE.

Although casting functions are automatically defined between each distinct type and its source type, you must invoke these casting functions explicitly if you wish to compare a distinct type value with a source type value. Please note that constants are always considered to have built-in types. For example, 100.00 is considered to be decimal. To compare a distinct value to a constant, you must perform an explicit cast:

money > 100.00 is a type error  
money > money(100.00) is correct

The built-in datatypes come with a collection of built-in functions that operate on them. These built-in functions include arithmetic operators on numeric datatypes (+, -, \*, /) and column functions such as sum and avg. After creating a distinct type, you can specify that the distinct type inherits some or all of the functions that operate on the source type. This is done by creating new functions, so called source functions, that operate on the distinct type.

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Implicit Casting

IBM @server iSeries

- Implicit casting allows some distinct type assignments without exact type match
  - Assignments involving constants and castable data types
  - Host variable assignments in non-SQL languages
- Implicit casting on output:
  - A distinct type column can be assigned to a *castable* type, or same distinct type (*castable* = using casting function)
  - A distinct type column can be assigned into any host variable that is compatible with its source type
- Implicit casting on input:
  - A castable type or same distinct type can be assigned to distinct type column
  - A host variable that is compatible with the distinct type source type can be assigned to distinct type column

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

IBM @server iSeries

Implicit casting allows some distinct type assignments without exact type matching. Implicit casting allows:

- castable constant and other type values to be assigned to distinct types;
- castable distinct type values to be assigned to other types;
- host variable assignments for non-SQL language access to distinct types.

Cast functions are also used when passing distinct types to host variables (and also back to the database), whose types correspond to the SQL predefined types (e.g. integer, float). Whenever an assignment to a host variable is being performed, the system automatically invokes the corresponding cast function in order to enable the assignment of an instance of a distinct type to a host variable whose type corresponds to the source type of the distinct type instance being assigned. This is necessary because there is not a type corresponding to the distinct type in the type system of the host programming language. This is referred to as implicit casting on output data. The following query is an example of the implicit casting:

```
dcl h_age int
select age into h_age from emp      (where age is of type t_age)
```

On assignments from host variables to columns, no automatic CAST is performed by the system. The user needs to explicitly invoke the corresponding cast function. Thus, explicit casting is required, as in the following example:

```
update emp set age = age(:h_age)
```

Explicit casting is also required when comparing a distinct type to a constant. For example the following statement is incorrect:

```
select * from emp where age > 21
```

A correct way to specify the above statement is:

```
select * from emp where int(age) > 21
```

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Implicit Casting with Promotion

IBM @server iSeries

### Promotion of source types allows additional source types to be assigned to the target distinct type

- Based on precedence order of data types
- Lower precedence types can be promoted to higher types
- Higher precedence type cannot be promoted to lower types

### Example

- MONEY sourced from DECIMAL(11,2)
  - can assign INTEGER value to MONEY value as INTEGER is lower than DECIMAL in precedence order
  - cannot assign MONEY value to INTEGER value as DECIMAL is higher than INTEGER in precedence order

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Precedence of Data Types

IBM  server iSeries

Data Type	Data Type Precedence List (in best-to-worst order)
CHAR or GRAPHIC	CHAR or GRAPHIC, VARCHAR or VARGRAPHIC, CLOB or DBCLOB
VARCHAR or VARGRAPHIC	VARCHAR or VARGRAPHIC, CLOB or DBCLOB
CLOB or DBCLOB	CLOB or DBCLOB
BLOB	BLOB
SMALLINT	SMALLINT, INTEGER, DECIMAL or NUMERIC, REAL, DOUBLE
INTEGER	INTEGER, DECIMAL or NUMERIC, REAL, DOUBLE
DECIMAL or NUMERIC	DECIMAL or NUMERIC, REAL, DOUBLE
REAL	REAL, DOUBLE
DOUBLE	DOUBLE
DATE	DATE
TIME	TIME
TIMESTAMP	TIMESTAMP
DATALINK	DATALINK
A distinct type	The same distinct type

IBM  server. For the next generation of e-business.

© 2000 IBM Corporation

## UDTs - DB2 for AS/400 Implementation

IBM  server iSeries

- New object \*SQLUDT
- Stored in a new catalog SYSTYPES (with built-in types)
- Part of DB Recovery Procedure
  - \*SQLUDT objects automatically saved and restored with library
  - A table with distinct types can be restored if the distinct type does not exist
    - Distinct type columns in table will not be accessible
  - A distinct type cannot be dropped if it is used by a table
    - Delete the dependent table first
- Supported by system utilities and CL commands
  - DSPFFD displays both distinct data type and the source data type
  - WRKOBJ
- Logical files can be created over tables with distinct types
  - UDT fields cannot be specified as part of LF definition
  - UDT fields can be cast to source type in SQL view
  - Native I/O restricted

IBM  server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

---

IBM  server iSeries

A new object type, \*SQLUDT created for distinct types. There is one object for each distinct type. The object contains all of the information for that distinct type.

Distinct types (as well as predefined types) are stored in a new catalog SYSTYPES. Types are also referred to in catalogs SYSCOLUMNS, SYSROUTINES, SYSFUNCS, and SYSPARMS.

A table with distinct types, can be restored if the distinct types does not exist on the system with the same source type.

Types themselves can be saved by saving the \*SQLUDT object or the library. The new object in the library for types will be used to do the restore for types in that library. A restore handler will be invoked at restore time for the new object. The object will contain the information to update the catalogs to do the restore.

Restore must invoke the restore handler before any physical files are restored. That is because the files may be dependent on the types. However, there is no restriction on types being in the same library as the files that use them. The types must be restored to allow access to columns in the dependent tables.

SQL view casting UDT to source type allows native I/O read-only access to UDT column, write access to non-UDT columns.

IBM  server. For the next generation of e-business.

© 2000 IBM Corporation

## UDT - Restrictions

---

IBM  server iSeries

- Native I/O on tables with distinct types not be allowed.
- Distinct types cannot be defined over other distinct types.
- Distinct types cannot be created in QTEMP.
- Distinct types are not allowed to be referenced in a logical file, except for SQL Indexes and SQL Views.

IBM  server. For the next generation of e-business.

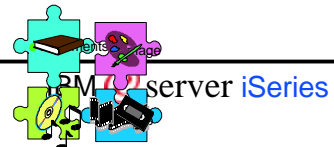
© 2000 IBM Corporation

## User Defined Functions

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

### User-Defined Functions (UDFs)



- Provides the ability to add application logic to the database
  - Function can be applied inside or outside the database
- Functions are called through SQL and are therefore reusable by all authorized database users, including programmers and end users
- Provide enhanced performance by:
  - Executing on the server rather than at the client
  - Being tightly integrated with database optimization and execution
- In combination with LOBs and UDTs, can provide processing methods for business objects tailored to specific business needs:
  - Advanced datatype support (e.g. multimedia, arrays, etc.)
  - Additional search technologies
    - contextual search, e.g. find all documents with the words DB2 and UDF
    - video scan e.g. find the scene starting with a sunset
- Use of standards allows functions to be easily added to the database

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

---

IBM @server iSeries

A user-defined function is a mechanism with which you can write your own extensions to SQL. The built-in functions supplied with DB2 are a useful set of functions, but they may not satisfy all of your requirements. Thus, you may need to extend SQL .

In certain cases, invoking the UDF directly from the database engine instead of from your application can have a considerable performance advantage. You will notice this advantage when the function may be used in the qualification of data for further processing. These cases occur when the function is used in record selection processing.

You can implement the behavior of a user-defined distinct type (UDT), also called distinct type, using a UDF. When you create a distinct type, you are automatically provided cast functions between the distinct type and its source type. You may also be provided comparison operators such as =, >, <, and so on, depending on the source type. You have to provide any additional behavior yourself. It is best to keep the behavior of a distinct type in the database where all of the users of the distinct type can easily access it. You can use UDFs, therefore, as the implementation mechanism.

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## UDF Types

---

IBM @server iSeries

- External
  - Similar to external stored procedures
  - Use AS/400 high-level language to implement
- Sourced
  - Instead of defining from scratch, redefine an existing built-in function or UDF
  - UDT Strong Typing requires their own basic functions
- SQL
  - All the function logic coded with SQL
  - Utilizes procedural SQL introduced in V4R2

**NOTE: Sourced & SQL UDFs require the ILE C & SQL Products to be installed during creation**

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

IBM  server iSeries

There are three types of UDFs: sourced, external, and SQL. The implementation of each type is considerably different.

- Sourced UDFs.

These are simply functions registered to the database that themselves reference another function. They, in effect, map the sourced function. As such, nothing more is required in implementing these functions than registering them to the database using the CREATE FUNCTION statement.

- External functions.

These are references to programs and service programs written in a high level language such as C, COBOL, or RPG. Once the function is registered to the database, the database will invoke the program or service program whenever the function is referenced in a DML statement. As such, external UDFs require that the UDF writer, besides knowing the high level language and how to develop code in it, must understand the interface between the program and the database.

- SQL UDFs.

SQL UDFs are functions written entirely in the SQL language. Their 'code' is actually SQL statements embedded within the CREATE FUNCTION statement itself. SQL UDFs provide several advantages:

- They are written in SQL, making them quite portable.
- Defining the interface between the database and the function is by use of SQL declares, with no need to worry about details of actual parameter passing.
- They allow the passing of large objects, datalinks, and UDTs as parameters, and subsequent manipulation of them in the function itself.

IBM  server. For the next generation of e-business.

© 2000 IBM Corporation

## Sourced UDFs

IBM  server iSeries

- All the system built-in functions reside in QSYS2
- Can source the following system operators only if one of the parameters is a distinct data type
  - /, -, \*, +, ||, CONCAT
  - conditional operators **NOT** supported
- Small service program created for the sourced function

```
CREATE DISTINCT TYPE cv AS CLOB(32K)
```

```
CREATE FUNCTION SUBSTR(cv,INT,INT) RETURNS cv SOURCE SUBSTR(CLOB)
```

```
CREATE FUNCTION LENGTH(cv) RETURNS INT SOURCE LENGTH(CLOB)
```

```
CREATE FUNCTION CONCAT(cv,cv) RETURNS cv SOURCE CONCAT(CLOB)
```

IBM  server. For the next generation of e-business.

© 2000 IBM Corporation

## SQL UDFs

IBM @server iSeries

- "Pure" SQL implementation
  - Defined, written, and registered using CREATE FUNCTION statement
- Easier to write than External Functions
  - Parameter passing rules defined by the database support
- Require ILE C and SQL on the development system
- No SQL debugger
  - Debugging can be challenging

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## SQL UDF - coding example

IBM @server iSeries

```
create function GETDESCRIPTION( chs_ProductNumber CHAR(5) )
returns VARCHAR(1024)
language SQL
specific GTDESC0001
is deterministic
reads SQL DATA
no external action
BEGIN
    DECLARE chs_Description CLOB(50K);
    DECLARE chs_ReturnValue VARCHAR(1024);

    select product_description into chs_Description
    from prodmast04
    where product_number = chs_ProductNumber;

    set chs_ReturnValue = VARCHAR( CLOB( SUBSTRING( chs_Description,
        1,
        ( LOCATE('Sizes', chs_Description, 1) - 1 ) ) ),
        1024 );
    return chs_ReturnValue;
END
```

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## External UDFs

IBM @server iSeries

- Supported languages
  - **C, C++, ILE RPG, ILE COBOL**
  - CL, RPG, COBOL
- Cannot be a column function
- Creation process updates program object so that UDF will be registered automatically when program restored to another system
  - Must be ILE language and contain at least one SQL statement

```
CREATE FUNCTION SQUARE(INT) RETURNS INT CAST FROM FLOAT
LANGUAGE C
EXTERNAL NAME 'MYLIB/MATH(SQUARE)'
DETERMINISTIC
NO SQL
NO EXTERNAL ACTION
PARAMETER STYLE SQL
ALLOW PARALLEL
```

To create the external service program so it can be debugged:  
CRTCMOD MODULE(mylib/square) DBGVIEW(\*SOURCE)  
CRTSRVPGM SRVPGM(mylib/math) MODULE(mylib/square)  
EXPORT(\*ALL) ACTGRP(\*CALLER)

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Resolving UDFs

IBM @server iSeries

- Finding the right UDF to call can be challenging for DB2 when you consider that:
  - UDFs support function overloading
    - UDFs can be sourced from existing functions
    - UDFs can have the same name as built-in functions
  - Unqualified UDF invocations allowed
- Three determining resolution factors are:
  - Function Signature
  - Function Path
  - Parameter Matching

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

---

IBM @server iSeries

Resolving to the correct function to use for an operation is more complicated than other resolution operations since multiple possible functions could exist (with the same name). Since a function is defined by its signature, a function resolution process is needed.

A user may define a function with the same name as a built-in. For example, SUBSTR is a built-in function, but the user may define his own SUBSTR function that takes slightly different parameters. Therefore, even resolving to a supposedly built-in function still requires that function resolution be performed.

Function resolution uses the library list to help determine which function is a 'best fit'. Normally, the QSYS2 library (the library where the built-in and shipped functions are defined) is the first in the library list. However, using CHGSYSLIBL or the SET PATH statement, it is possible for the user to move his or her own library ahead of the built-in ones, forcing the function resolution code to consider the user's functions before the system's. This can be very powerful for the user, but it also dangerous. The user needs to be aware of this case and should not normally do it.

Also, QSYS2 can be completely removed from the library list. This will make the built-in and shipped functions unusable. This should definitely be avoided by the user (although it is not disallowed).

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Function Signature

---

IBM @server iSeries

- Made up of Collection, Name, and Parameter Type(s)
  - Allows for multiple functions to have the same name in the same collection/library/schema
  - Return value NOT part of the signature
  - Parameter length, precision & scale are NOT used
- Signature must be unique within a collection
- Signature examples:
  - mylib.myUDF(integer, char(5)) & mylib.myUDF(integer,integer) can exist in the same collection
  - xyz(integer,char(50)) & xyz(integer,char(5)) CANNOT exist in the same collection

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

---

IBM @server iSeries

A function is uniquely defined by its name, library, number and type of parameters (note that the return value of the function is not part of the signature). This is called a function signature. This means that it is possible (and likely) to have multiple functions in the same library with the same name; their difference lies in their number of parameters or data types of parameters. This process allows overloading of functions and is useful for having the same function handle multiple different data types.

When a function signature is determined, the parameters are compared without regard to length, precision or scale.

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Function Path

---

IBM @server iSeries

- Ordered list of collection names
  - Used when searching for unqualified data type references and function & procedure invocations
  - For System Naming (\*SYS), default path is \*LIBL
  - For SQL Naming (\*SQL) default path is: QSYS, QSYS2, <user-id>
- Path can be changed with SET PATH statement
- QSYS & QSYS2 assumed to be first in path unless explicitly put in another position
  - SYSTEM PATH is equivalent to QSYS, QSYS2
- Examples:
  - SET PATH myfunc, SYSTEM PATH  
(Result Path: myfunc, QSYS, QSYS2)
  - SET PATH myfunc, morefunc  
(Result Path: QSYS, QSYS2, myfunc, morefunc)

IBM @server. For the next generation of e-business.

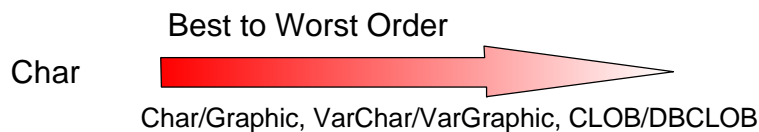
© 2000 IBM Corporation

# Parameter Matching

IBM @server iSeries

- Done by parameter type only
- Exact match is best, DB2 chooses the next best promoted argument

```
SELECT UDF(charcolumn), col2 FROM Table1
```



Function Signatures: ✓ Fred.UDF( varchar )  
Joshua.UDF( clob )  
Jenna.UDF( integer )

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

IBM @server iSeries

A function is considered applicable to a given call if its function name matches the call and if the arguments of the call are promotable to the parameters of the function. This means that the data type of each function parameter either must match the data type of the corresponding call argument or must be found on the right hand side of the argument on the promotion path. Following table illustrates the available promotion paths:

Data Type	Type can be Promoted To: (listed best to worst)
char or graphic	char or graphic, varchar or vargraphic, clob or dbclob
varchar or vargraphic	varchar or vargraphic, clob or dbclob
integer	integer, smallint, decimal or numeric, real, double
smallint	smallint, integer, decimal or numeric, real, double
decimal or numeric	decimal or numeric, real, double
real	real, double
double	double, real

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

# CREATE TABLE with FILE LINK CONTROL

IBM @server iSeries

## Operations Navigator

## SQL Equivalent

```
CREATE TABLE ProdMast02
(Product_Number      TEAMXX/SRLNUMBER,
 Product_Name        CHAR(25),
 Product_Description  TEAMXX/PRDESC,
 Product_Price        TEAMXX/MONEY,
 Product_Picture_Link DATALINK(200)
 LINKTYPE URL
 FILE LINK CONTROL
 INTEGRITY ALL
 READ PERMISSION DB
 WRITE PERMISSION BLOCKED
 RECOVERY NO
 ON UNLINK RESTORE)
```

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

# CREATE TABLE with FILE LINK CONTROL

IBM @server iSeries

```
CREATE TABLE ProdMast03
(Product_Number      TEAMXX/SRLNUMBER,
 Product_Name        CHAR(25),
 Product_Description  TEAMXX/PRDESC,
 Product_Price        TEAMXX/MONEY,
 Product_Picture_Link DATALINK(200)
 LINKTYPE URL
 FILE LINK CONTROL
 INTEGRITY ALL
 READ PERMISSION FS
 WRITE PERMISSION FS
 RECOVERY NO)
```

```
CREATE TABLE ProdMast03
(Product_Number      TEAMXX/SRLNUMBER,
 Product_Name        CHAR(25),
 Product_Description  TEAMXX/PRDESC,
 Product_Price        TEAMXX/MONEY,
 Product_Picture_Link DATALINK(200)
 LINKTYPE URL
 FILE LINK CONTROL
 INTEGRITY ALL
 MODE DB2OPTIONS)
```

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

IBM @server iSeries

The DataLinks architecture defines the following attributes:

**Linktype:** The only link type currently defined is the URL.

**Link Control:** This is the basic attribute that defines whether file system objects will be linked to a DataLink row in an RDBMS table.

- *No Link Control:* When rows are inserted into the table there would be no links established to the file objects referenced in the DataLink column. No check is made to verify that the file server can be accessed or that the file object being referenced even exists. However, the syntax of the URL is validated. While the 'No Link Control' option still provides value in terms of new application potential it does not enable you to benefit from the management and integrity control provided by the 'File Link Control' option.
- *File Link Control:* When a row is inserted into the table the DLFM immediately attempts to establish a link to the referenced file object. The file server must be accessible and the file object must exist. Once the link has been established the DLFM will maintain control of the link through its metadata. A file object may only be linked to one table row. However, a table row may contain multiple DataLink columns so long as each is linked to a different file object. Once a file has been linked it may not be moved, deleted or renamed. Deleting the table row will unlink the associated file. Updating the DataLink value in the table row will cause the originally referenced file to be unlinked while the newly referenced file is linked.

**Integrity:** This attribute controls the level of referential integrity between the database server and the file server.

- *Integrity All:* Any linked file object referenced by a DataLink column is considered to be under the control of the RDBMS and attempts to rename, delete or move the file object from a file system interface are rejected.
- *Integrity Selective:* Any linked file object referenced by a DataLink column is considered to be under the control of the RDBMS only if the file server has the DataLinks Filter installed. This option is **not supported by V4R4** of DB2 Universal Database for AS/400.

**Read Permission:** This defines where file object read access is controlled.

*Read Permission FS:* The file system will control whether a user has the necessary authority to perform a read operation on a linked file system object. No prior access to the associated RDBMS table is required.

- *Read Permission DB:* The RDBMS will control whether a user may perform a read operation on a linked file system object. Assuming the file system object has been given no public access authority it can only be read by first accessing the DataLink value in the database table and retrieving an access control token.

**Write Permission:** This defines whether a user can write to the file object.

- *Write Permission FS:* The file system will control whether a user has the necessary authority to perform a write operation to a linked file system object. No prior access to the associated RDBMS table is required. V4R4 of DB2 Universal Database for AS/400 enforces this option if Read Permission FS has been selected.
- *Write Permission Blocked:* A file system object cannot be written to through any interface because it is owned by the DLFM. V4R4 of DB2 Universal Database for AS/400 enforces this option if Read Permission DB has been selected.

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

IBM @server iSeries

**Recovery:** This attribute specifies whether point-in-time recovery of linked files will be supported.

- *Recovery Yes:* Point-in-time recovery is achieved by the RDBMS ensuring that backup copies of the linked files are made as needed. It is only valid when Integrity All and Write Permission Blocked are also specified. This option is **not supported by V4R4** of DB2 Universal Database for AS/400.
- *Recovery No:* Point-in-time recovery is not supported.

**On Unlink:** This attribute determines the action to be taken when the RDBMS controls write operations to a linked file (Write Permission Blocked) and the file is unlinked either through the DataLink value in the associated table row being updated or the row being deleted. Note that updating a row's DataLink value effectively deletes the current file link and replaces it with a new file link. The option is not applicable when write operations are controlled by the file system (Write Permission FS).

- *On Unlink Restore:* When a file is unlinked this option will ensure that the file's ownership and permissions are restored to their state at the time that the file was linked. If the owner no longer exists in the file system then a default owner may be established but this action will be dependent upon the particular file system involved. Apart from only being a valid option when Write Permission Blocked is also specified, Integrity All is also a prerequisite.
- *On Unlink Delete:* When a file is unlinked it will automatically be deleted. This option is only valid when Read Permission DB and Write Permission Blocked are also specified.

The diagram shows an alternative, shorthand SQL definition. MODE DB2OPTIONS is used to define a default set of options and is functionally equivalent to:

```
INTEGRITY ALL  
READ PERMISSION FS  
WRITE PERMISSIN FS  
RECOVERY NO
```

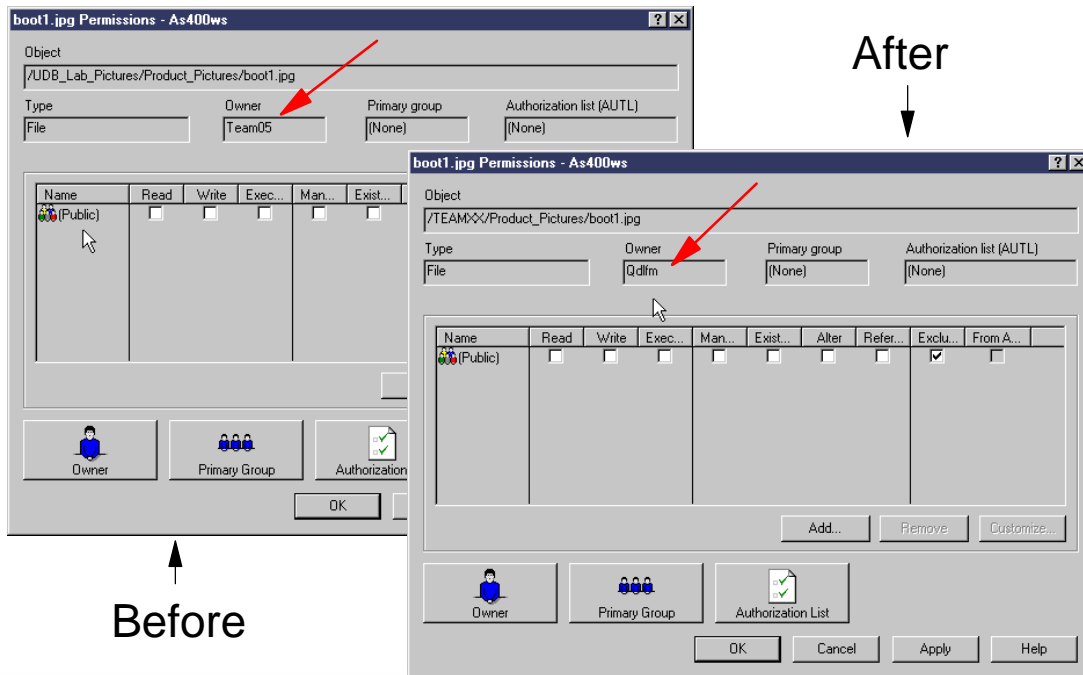
This notation is currently the only mode option that has been defined and is provided by DB2 Universal Database for AS/400 for compatibility with the other DataLink capable platforms.

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Database Permissions - File Ownership

IBM @server iSeries



IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

IBM @server iSeries

READ PERMISSION DB/WRITE PERMISSION BLOCKED means that DB2 Universal Database for AS/400 controls access to the associated file objects. This is achieved by transferring ownership of each file object to the DLFM (user profile QDLFM) at the time that a table row is inserted and the link is established. However, attempts to move, delete or rename the file while it is linked are always denied because referential integrity is enforced. Attempts by a program to perform write operations on the file are always rejected regardless of the permissions in place for the file. Attempts to perform read operations directly on the file will be honoured if the user has sufficient permissions in place for that file. However, as the intention of this link option is to have the database control read access to the file objects you should always ensure that the files to be linked have no public access permissions defined. Then, read operations will only be successful if the program first obtains an access control token from the database by reading the associated table row.

You can display the file permissions by using Operations Navigator to display the IFS files.

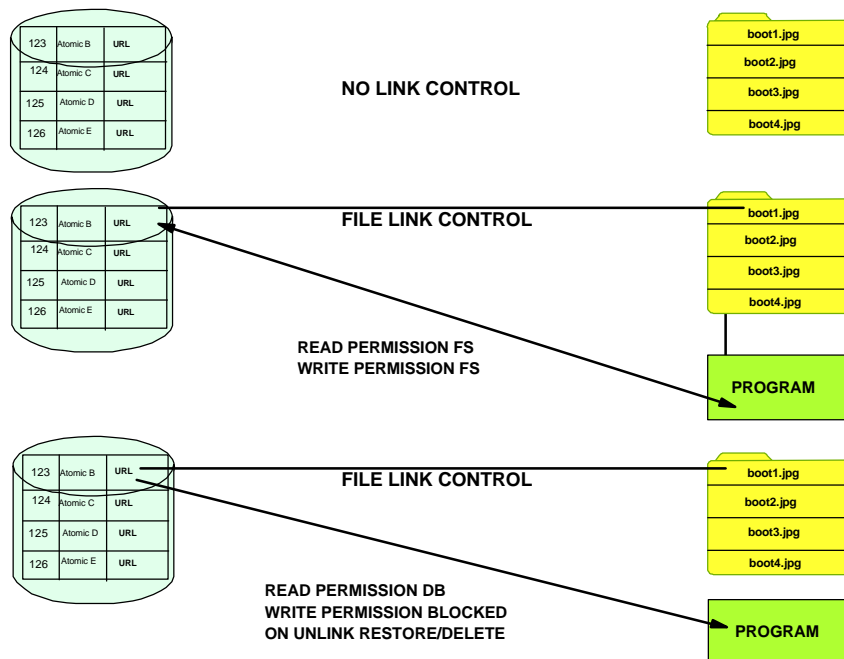
'Before' shows the ownership of a file in the AS/400 IFS before it has been linked. You will see that it is owned by user TEAM05. 'After' then shows the ownership of the same file after it has been linked to a table row where the table was created with the option Read Permission DB. You will see that the owner of the file is now QDLFM, the user profile of the DataLinks File Manager. However when ownership is changed to QDLFM details about the previous ownership are saved

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## AS/400 Link Control Options - Summary

IBM @server iSeries



IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## JDBC

IBM @server iSeries

### JDBC provides standard Java interfaces to access databases

- It is based on ODBC and supports a subset of what ODBC supports
- Database definition and manipulation is done using SQL statements

### JDBC is useful because it implements a standard interface

- It can theoretically work for accessing any database

### A JDBC driver provides a database specific connection

- AS/400 Toolbox for Java
  - com.ibm.as400.access.AS400JDBCdriver
- AS/400 Native Driver
  - com.ibm.db2.jdbc.app.DB2Driver

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

IBM @server iSeries

The JDBC driver implements standard Java interfaces (java.sql package) to provide access to the AS/400 database. It is based on ODBC and supports a subset of what ODBC supports. Database definition and manipulation is done using SQL statements.

- JDBC is useful because it implements a standard interface. An application or applet developer can write a program using JDBC and it can theoretically work for accessing any database. The only requirement is that the user has the JDBC driver for the specific database installed. The correct JDBC driver is selected automatically by the JDBC DriverManager. This decision is based on the URL used to name the database. AS/400 database URLs are in the format jdbc:as400://DBName.

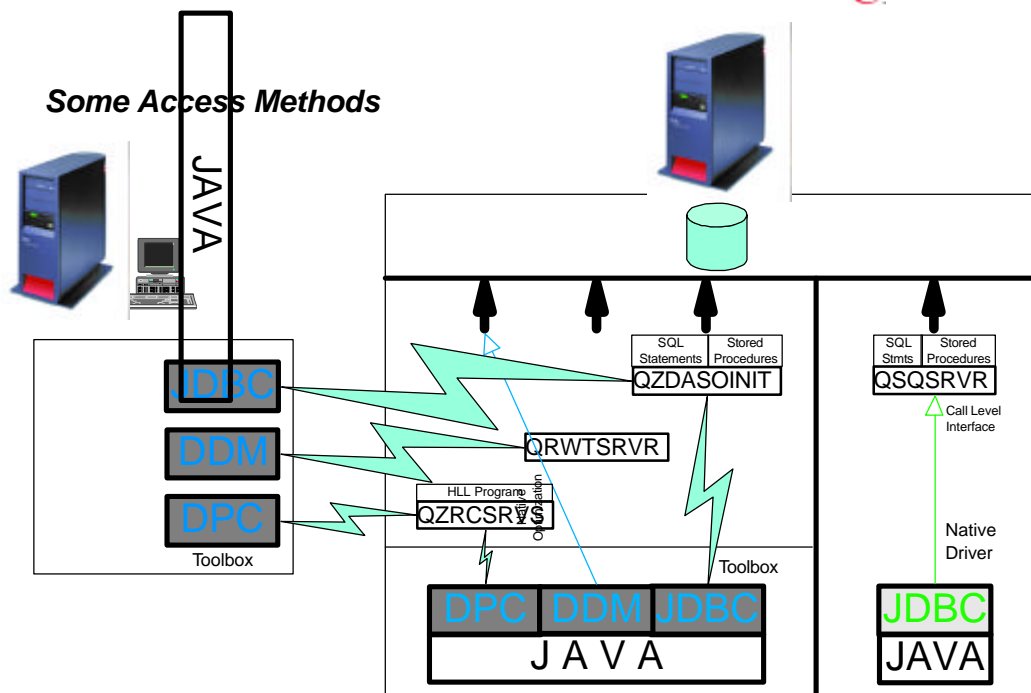
- In general, any level of user is exposed to JDBC. The reason is that applets and applications will be written to the standard JDBC interface. Users will plug the AS/400 JDBC driver in and expect it to work.

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Toolbox JDBC Driver VS. Native Driver

IBM @server iSeries



IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

IBM @server iSeries

In this figure, the left portion represents a Java application running on a client system and using the AS/400 Toolbox for Java classes to access the AS/400 system. Since the AS/400 Toolbox for Java classes are 100% pure Java, this application is portable to any compliant JVM. However, the server that it accesses has to be an AS/400 server.

The middle portion represents a Java application running on the AS/400 system using various ways of accessing system services. These include:

The middle portion represents a Java application running on the AS/400 system using various ways of accessing system services. These include:

- The DPC model is designed for client-server work so the Java application communicates through the QZRCSRVS server job to access and AS/400 program.
- The DDM or record-level access model has the AS/400 Toolbox for Java classes "optimized for native." This means that if the AS/400 Toolbox for Java classes determine that access is being done locally, there is no need to go through a server job or through sockets. The Java application accesses the database directly.
- This JDBC approach uses the AS/400 Toolbox for Java JDBC driver.

The right portion in the figure shows a server Java application using the "native" JDBC driver.

Regardless of which driver is used, JDBC can access DB2 through SQL statements or through stored procedures. The AS/400 system is unique in the industry because stored procedures do not have to contain embedded SQL statements. Even traditional 3GL programs using native I/O operations can be used as stored procedures.

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Toolbox JDBC Driver VS. Native Driver

IBM @server iSeries

### AS/400 Native JDBC Driver

- Uses the QSQSRVR server job
  - Runs inside the QSYSWRK subsystem
- Better performance when running in the AS/400 JVM
- Server side only
  - Can use DRDA to access other systems
  - Cannot pass result sets via DRDA with stored procedures
- Type 2 Driver

### AS/400 Toolbox JDBC Driver

- Uses the QZDASOINIT server job (like ODBC)
  - Runs inside the QSERVER subsystem
- Client/Server or Server
- Type 4 driver

### Drivers are functionally equivalent

- Both implement the java.sql.\* APIs

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

IBM  server iSeries

There are two JDBC drivers available when running a Java application on the AS/400 system. One is the native driver or the AS/400 Developer Kit for Java driver, `com.ibm.db2.jdbc.app.DB2Driver`. The other is the AS/400 Toolbox for Java driver, `com.ibm.as400.access.AS400JDBCdriver`. Some of the properties may vary, but the SQL functionality is mostly similar. The AS/400 Toolbox for Java driver is a type 4 driver and provides network access for clients that are physically separated from the database. The AS/400 Developer Kit for Java driver is a type 2 driver that provides direct access.

### AS/400 Developer Kit for Java Driver Performance

The AS/400 Developer Kit for Java driver, on the other hand, was designed to process SQL requests from a server Java program to access DB2/400. It uses the Call Level Interface (CLI) standard to submit SQL requests. CLI is like ODBC running natively on the AS/400 system, with essentially the same APIs. It produces dynamic SQL statements, as do the ODBC or Toolbox JDBC environments. The difference is in the way that SQL statements are cached.

Past experience with dynamic SQL in various computing platforms brings back memories of poor performance. Most of the problem was due to statement parsing and syntax checking. However, starting with OS/400 V4R2, an internal system-wide SQL statement cache was implemented, which significantly improves the execution of dynamic SQL. This cache stores dynamic SQL statements for much improved subsequent execution.

Currently, there is a maximum of 500 concurrent statement handles per connection. In the unlikely situation that more than 500 are required, another connection can be established.

### AS/400 Toolbox for Java Driver Performance

The AS/400 Toolbox for Java driver was originally designed to let a client access DB2/400 in a client/server environment through a network. It uses a sockets connection between the client and the server. The server job to which the JDBC requests are submitted is named QZDASOINIT. This is the same server job name used to process 32-bit client requests in the ODBC environment.

The AS/400 Toolbox for Java driver allows SQL Extended Dynamic support, which allows SQL statements and their access plans to be stored in objects of type SQL Package (\*SQLPKG) on the AS/400 system. This provides a significant performance benefit for repeatable SQL statements because of the reduced parsing and syntax checking overhead. Note that one performance difference between the AS/400 Toolbox for Java JDBC environment and ODBC is that in the latter, Extended Dynamic mode is false, by default.

However, when used in a server Java application within the same AS/400 system that contains the data, the current implementation does not "short circuit" the sockets implementation. This means that requests from a server Java program go through the sockets interface before accessing DB2/400. The result is additional latency as well as additional processing overhead.

IBM  server. For the next generation of e-business.

© 2000 IBM Corporation

## Application Flow

IBM  server iSeries

### Establish a connection

### Prepare SQL statement(s)

### Begin optional loop:

- Set values of input parameters (prepared and callable statements only)
- Execute SQL statement
- Process the results

### Close result set

### Close statements

### Close the connection

IBM  server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

---

IBM @server iSeries

The general flow for any application that uses JDBC is similar.

- Establish a connection
- Prepare SQL statement(s)
- Set values of input parameters (prepared and callable statements only)
- Execute SQL statement
- Process the results
- Close result set
- Close the connection

The JDBC classes of the Java core API (java.sql package) allow us to do all these actions. We will show examples of how the classes are used to implement an application using JDBC.

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Establish a connection

---

IBM @server iSeries

```
String system = "AS400WS";
Connection connection = null;
try {
    // Load the AS/400 Toolbox for Java JDBC driver.
    DriverManager.registerDriver(new
    com.ibm.as400.access.AS400JDBCDriver());
    // Get a connection to the database. Since we do not
    // provide a user id or password, a prompt will appear.
    connection = DriverManager.getConnection ("jdbc:as400://" + system);
    DatabaseMetaData dmd = connection.getMetaData ();
    ....
}
```

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

IBM @server iSeries

We need to register the JDBC driver that we want to use. This loads the JDBC driver into the JVM. The following code registers the AS/400 Toolbox JDBC Driver:

```
DriverManager.registerDriver( new com.ibm.as400.access.AS400JDBCDriver());
```

We want to be able to run this example on either a client workstation or an AS/400 system, so we keep the name of the driver to use in the properties file:

```
DriverManager.registerDriver(dr);
```

Next, we need to contain a connection to the database. We use the getConnection method to do this.

The getConnection method expects parameters for URL, DBUserId, and DBPasswd.

- When using the AS/400 Toolbox for Java driver, use the following syntax for the url:

```
"jdbc:as400://systemName/defaultLibrary".
```

```
conn1 = DriverManager.getConnection("jdbc:as400://" + dbName + "/apilib;naming=sql;errors=full; date format=iso",  
dbUserId, dbPasswd);
```

- When using the AS/400 Native JDBC driver, use the following syntax for the url:

```
"jdbc:db2://sysname/defaultLibrary"
```

Again, because we want to be able to run this example on either a client workstation or an AS/400 system, so we keep the url value to use in the properties file. Also notice how we just pass in the properties file to set the properties that we want to use.

```
conn = DriverManager.getConnection(dbURL, props);
```

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Executing statements

IBM @server iSeries

### prepareStatement(String)

- Method in java.sql.Connection interface

**Returns a PreparedStatement object**

### setXXX()

- Methods in java.sql.PreparedStatement interface

### executeQuery()

- Method in java.sql.PreparedStatement interface
- Executes an SQL statement that returns a single ResultSet

**Returns a ResultSet**

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

IBM @server iSeries

### prepareStatement:

public PreparedStatement prepareStatement(String sql) throws SQLException

Creates a PreparedStatement object for sending parameterized SQL statements to the database. A SQL statement with or without IN parameters can be pre-compiled and stored in a PreparedStatement object. This object can then be used to efficiently execute this statement multiple times.

Note: This method is optimized for handling parametric SQL statements that benefit from precompilation. If the driver supports precompilation, the method prepareStatement will send the statement to the database for precompilation.

#### Returns:

a new PreparedStatement object

#### Throws: SQLException

if a database-access error occurs.

### executeQuery

public abstract ResultSet executeQuery(String sql) throws SQLException

Execute a SQL statement that returns a single ResultSet.

#### Parameters:

sql - typically this is a static SQL SELECT statement

#### Returns:

a ResultSet that contains the data produced by the query; never null

#### Throws: SQLException

if a database-access error occurs.

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Executing Statements

IBM @server iSeries

```
PreparedStatement stmt = connection.prepareStatement(  
"SELECT product_picture " +  
"FROM dbteamxx.prodmast01 " +  
"WHERE PRODUCT_NUMBER = CAST (? AS SRLNUMBER)");  
stmt.setString(1, productNumber);  
ResultSet rs = stmt.executeQuery();
```

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Creating and meterializing BLOB

---

IBM @server iSeries

```
while (rs.next()) {
    Blob pictblob = rs.getBlob(1);
    long length = pictblob.length();
    ImageIcon imageicon = new ImageIcon(pictblob.getBytes(0, (int)
length));
    image = imageicon.getImage();
}
```

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

---

IBM @server iSeries

An SQL BLOB is mapped by the JDBC driver into a Java Blob object. You can access values of type Blob in the same way that you access traditional SQL92 built-in types. The interfaces ResultSet, CallableStatement, and PreparedStatement support methods getBlob and setBlob for a BLOB value. You can use these methods in the same way that you use getString and setString to manipulate a CHAR or VARCHAR value. The JDBC 2.0 specification defines Blob as an interface. The JDBC 2.0 driver provides a database specific class, which implements this interface. In case of the AS/400 Toolbox for Java driver, this class is called com.ibm.as400.access.AS400JDBCBlob.

The setString method of the PreparedStatement class is used to set the parameter to the Product\_Number passed by the invoking process.

The Blob object is created. At this time, the variable pictblob contains a logical pointer to the BLOB value stored in the Product\_Picture column. Note that the UDT Picture was implicitly cast to its source type BLOB(1M) on the I/O operation. Therefore, no explicit casting is needed, and we can use getBlob method on the rs object.

We need to materialize the BLOB data before we can display it on the workstation. We use the getBytes method on the Blob object for this purpose. The imageicon object now contains a copy of all of the bytes in the BLOB value.

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Close and Disconnect

---

IBM @server iSeries

```
public void dispose() {  
  
    try{  
        // close Statement, Result Set, Connection  
        rs.close();  
        stmt.close();  
        connection.close();  
  
    }  
    catch (Exception e){  
        System.out.println("Error while closing...");  
        System.out.println(e);  
    }  
  
}
```

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

---

IBM @server iSeries

When the application is done, we use the close method for the statement objects and the connection objects. This will end the host server jobs and close any open cursors and sockets connections.

**ps.close();**

Releases the PreparedStatement's database and JDBC resources immediately. This will also close the current ResultSet.

**con1.close();**

Releases the connection

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## The LoadPicture class

---

IBM @server iSeries

```
public static void main (String[] args)
{
  ...
  try {
    File file = new File(args[0]);
    int fileLength = (int)file.length();
    InputStream fin = new FileInputStream(file);
    ...
  }
}
```

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

---

IBM @server iSeries

The following code example shows how to load Blob data into a AS/400 table using the `setBinaryStream` method. This approach is useful if you have to construct the BLOB object in your application and then upload it to the database for persistent storage. The LoadPicture program accepts two parameters: name of a file on the workstation that contains the product picture, and the product number for the given picture. The program reads the content of the file and stores it as a Blob object in the AS/400 database.

We use the instance of `FileInputStream` to obtain the content of the picture file located in the workstation's file system.

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Upload Blobs with Java Input Streams

---

IBM @server iSeries

```
PreparedStatement stmt = connection.prepareStatement(
"UPDATE " + collectionName + dmd.getCatalogSeparator() + tableName +
" SET PRODUCT_PICTURE = ? " +
"WHERE PRODUCT_NUMBER = CAST( ? AS SRLNUMBER)");
// Set the first parameter marker to a binary input stream
stmt.setBinaryStream(1, fin, fileLength);
// Set the second parameter marker to a String
stmt.setString(2, args[1]);
// Execute the SQL statement
stmt.executeUpdate();
```

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation

## Notes:

---

IBM @server iSeries

The Blob parameter marker is set to input stream.  
At the SQL statement execution, the JDBC driver repeatedly calls to the `fin` input stream to transmit the Blob content to the database.

IBM @server. For the next generation of e-business.

© 2000 IBM Corporation