Mastering Advanced SQL Data Types in ILE RPG - From LOBs to XML

Common Europe 2026 - Lyon

Antonio Salcedo
www.cptserv.com
asalcedo@cptserv.com

About me

- Antonio Salcedo.
- IBM i instructor, consultor and programmer for the las 26 years.
- IBM Champion 2024 and 2025.
- WebSphere Application Server, IBM Security Directory Server, IBM Verify Access, IBM Security Identity Manager, QRadar SIEM.

Agenda (1/2)

- Introduction: Beyond VARCHAR(32766)
- Main LOB Types:
  - CLOB
  - DBCLOB
  - BLOB
- Efficient Handling: LOB Locators
- IFS Integration: LOB File References
- Demo 1

Agenda (2/2)

- XML_CLOB_FILE
  - XMLPARSE
  - XMLGROUP
- Demo 2
- Q&A

Introduction: The Modern Challenge

IBM i is no longer isolated.

- We need to consume and serve REST APIs (which speak JSON/XML).
- We need to store documents (PDFs, Word files).
- We store images, application logs, and multimedia data.
- Today's data easily exceeds the 32KB limit of a VARCHAR. (https://www.ibm.com/docs/en/i/7.6.0?topic=reference-sql-limits)

LOBs (Large Objects) are the answer from Db2 for i.

What is a LOB?

"**Large OBject**"

- A data type designed to store massive amounts of information (up to 2GB).
- The VARCHAR, VARGRAPHIC, and VARBINARY data types have a storage limit of 32KB. When large texts need to be stored, this size is insufficient. Additionally, it may be necessary to store audio, video, images, etc., making LOBs the appropriate data types.
- If programming language variables are used, the maximum amount of information will be limited to the maximum size that variable can hold.
- If you need to store more information than the maximum size supported by the host variable, you must use LOB Locators.

LOB Types

The 3 LOB types we will work with are:

- **CLOB**: For single-byte characters.
  - JSON, XML, Logs, CSV…
- **DBCLOB**: For double-byte characters.
  - Text in languages such as Chinese, Japanese, or Korean
  - Data in UTF-16
- **BLOB**: For binary data (unformatted).
  - Images, documents (PDF, DOCX), multimedia (MP3, MP4), *STMF…

CLOB in ILE RPG

- The length must be in the range of 1 to 16,773,100 bytes.
- The declaration:

```
dcl-s db2Clob sqltype(clob:5000);
```

generates

```
DCL-DS DB2CLOB;
   DB2CLOB_LEN UNS(10);
   DB2CLOB_DATA CHAR(5000) CCSID(*JOBRUNMIX);
END-DS DB2CLOB
```

CLOB in ILE RPG - Example

SQL

```
-- DDL for an orders table with JSON
create table orders (
    id_order    int not null primary key,
    customer    varchar(100),
    date        timestamp,
    -- a 5 megabyte clob for the json
    json_data   clob(5m) ccsid 1208
);
```

RPG

```
**free
dcl-s myJsonSQL sqltype(clob:5000000) ccsid(1208);
dcl-s myJsonRPG varchar(5000000) ccsid(1208);

// We fetch 5 megs of data into a SQL variable...
exec sql
  select json_data into :myJsonSQL
    from orders where id_order = 1;

// ...and then we copy the text portion into a varchar
myJsonRPG = %subst(myJsonSQL_data:1:myJsonSQL_len);
```

DBCLOB in ILE RPG

- The length must be in the range of 1 to 8.386.550 bytes.
- The declaration:

```
dcl-s db2DbClob sqltype(dbclob:1000);
```

generates

```
DCL-DS DB2DBCLOB;
   DB2DBCLOB_LEN UNS(10);
   DB2DBCLOB_DATA GRAPH(1000);
END-DS DB2DBCLOB
```

DBCLOB in ILE RPG - Example

SQL

```
-- DDL for global customers
create table global_customers (
    id_customer   int       not null primary key,
    country       char(2),
    -- a dbclob for names/addresses in
    -- any language (utf-16)
    legal_name    dbclob(1m) ccsid 1200;
);
```

RPG

```
**free
dcl-s legalName sqltype(dbclob:1000000) ccsid(1200);

// We fetch 1 meg of data into a SQL variable...
exec sql
  select legal_name into :legalName
  from global_customers where country = 1;

// Reminder:
//   legalName_data contains the data
//   legalName_len  contains the length of that data
```

BLOB in ILE RPG

- The length must be in the range of 1 to 16.773.100 bytes.
- The declaration:

```
dcl-s db2Blob sqltype(blob:1000000);
```

generates

```
DCL-DS DB2BLOB;
   DB2BLOB_LEN UNS(10);
   DB2BLOB_DATA CHAR(1000000) CCSID(*HEX);
END-DS DB2BLOB
```

BLOB in ILE RPG - Example

SQL

```
-- DDL for a document repository
create table attached_documents (
    id_attachment    int   not null generated always as identity,
    id_case          int   not null,
    file_name        varchar(255),
    mime_type        varchar(100),
    -- a 5 megabyte blob for the file
    binary_file      blob(5m)
);
```

RPG

```
**free
dcl-s myPdf_sql  sqltype(blob:5000000); // 5 Megs

// we fetch 5mb into the sql variable...
exec sql
  select binary_file into :myPdf_sql
  from attached_documents where id_attachment = 1;
```

What do we do with the PDF? Save it to a file in the IFS?

Efficient Handling: LOB Locators

- **The Problem:** We don't want to bring large amounts of data into the RPG program just to read 100 bytes.
- **The Solution: Locators**
  - CLOB LOCATOR
  - DBCLOB LOCATOR
  - BLOB LOCATOR
- What are they? They are RPG variables (of type SQLTYPE) that act as a "pointer" or "handle" to the LOB within the database.
- They do not contain the data, only a reference.

LOB Locators: The Concept

Without Locator:

1. SELECT … INTO :myClob
2. Db2 sends 5MB of JSON to the RPG program.
3. RPG stores it in a variable.
4. RPG processes 1KB.
5. 4.999MB are discarded.

With Locator:

1. SELECT ... INTO :myLocator
2. Db2 sends a 16-bytes "handle" to the RPG program.
3. RPG uses SQL functions (e.g. SUBSTR) on the Locator.
4. Db2 processes the LOB inside the DB and returns only 1KB.

LOB Locators: Handling

You can use many SQL functions with Locators:

- **LENGTH**(:myLocator): Returns the actual size of the LOB.
- **SUBSTR**(:myLocator, start, len): Extracts a portion of the string.
- **POSITION**('text' IN :myLocator): Searches for a string.
- **CONCAT**(:myLocator1, :myLocator2): Concatenates LOBs.

All of this happens on the DB server!!!

LOB Locators: Important! Free Resources

- A Locator is a resource. It consumes memory on the server.
- When you are done using it, you must release it.
- If you don't, you can cause memory leaks.
- To do so, use the SQL statement **FREE LOCATOR**.

LOB Locators in ILE RPG - Example

```
-- DDL for an orders table with JSON
create table orders (
    id_order    int not null primary key,
    customer    varchar(100),
    date        timestamp,
    -- a 5 megabyte clob for the json
    json_data   clob(5m) ccsid 1208
);
```

```
**free
dcl-s myJsonLocator sqltype(clob_locator);
dcl-s jsonChunk     varchar(1000);
dcl-s lenJson       packed(3);
dcl-s pos           packed(2);

// We get the "pointer" (locator)
exec sql
  select json_data into :myJsonLocator
  from orders where id_order = 1;

// We request a substr of the lob using the locator
exec sql
  set :jsonChunk = substr(:myJsonLocator, 1, 5);

// Now :jsonChunk contains the first 5 bytes.
// The 5mb lob never left the database.

// Calculating the length of the clob data
exec sql
  set :lenJson = length(:myJsonLocator);

// Calculating the position of the string "json" in the clob
// Assuming it contains "Data from a json" pos = 13
exec sql
  set :pos = position('json' in :myJsonLocator);
```

IFS Integration: File References

The problem:

- **Reading:** How do I insert a 5MB PDF located in the IFS (/home/Antonio/docs/my.pdf) into a BLOB field?
- **Writing:** How do I save a BLOB from the DB to a file in the IFS?

The Inefficient (and complicated) Solution:

1. Using APIs to read the file from the IFS into a variable.
2. Performing an INSERT from RPG with that variable.

The Efficient Solution: File References

LOB File References: The Types

- LOB files are used to transfer data to and from IFS files.
- They represent a file, they do not contain it.
- Database queries, updates, and inserts can be used to store or retrieve LOB values without having to use routines (APIs) to read and write files.
- LOB files are allowed in host data structures and cannot be initialized.
- LOB File variables can be of type CLOB, DBCLOB, or BLOB.
- The SQL precompiler will generate a data structure with sub-fields related to the file using suffixes.

LOB File References: Data Structure (1/3)

The precompiler generates a structure with the RPG variable name and the following suffixes:

- **_NL:** File name length. Must be specified in the program.
- **_DL:** Data length. Not used during input. During output, the application sets the value to the length of the new data to be written to the file.
- **_FO:** File operation. Must be specified in the program from a fixed list of operations (see next slide).
- **_NAME:** File name in the IFS. A relative name can be specified but it is preferable to use the full path.

LOB File References: Data Structure (2/3)

It also generates the following constants for the open mode:

- **DCL-C SQFRD CONST(2):** Open for reading.
- **DCL-C SQFCRT CONST(8):** A new file will be created. If it already exists, an error is returned.
- **DCL-C SQFOVR CONST(16):** The file will be overwritten if it exists; if not, a new one is created.
- **DCL-C SQFAPP CONST(32):** Data will be appended if the file exists; if not, a new one is created.

LOB File References: Data Structure (3/3)

```
**free
dcl-s ifsFile1 sqltype(clob_file);
// The compiler will generate this:
//   DCL-DS IFSFILE1;
//        IFSFILE1_NL UNS(10);
//        IFSFILE1_DL UNS(10);
//        IFSFILE1_FO UNS(10);
//        IFSFILE1_NAME CHAR(255) CCSID(*JOBRUNMIX);
//   END-DS;
//
// Where:
// _name = File name
// _nl   = File name length
// _fo   = Operation to perform on the file (based on the constants below)
// _dl   = Not used
//
// The compiler adds the following constants:
//   SQFRD  = 2  = Reads the file
//   SQFCRT = 4  = Creates the file
//   SQFOVR = 8  = Overwrites the file
//   SQFAPP = 16 = Appends to the end of the file
```

Demo Time!


TIME FOR THE DEMO!
LOBS & LOB FILE REFERENCES in ACTION
JSON/XML (CLOB)
PDFs/Images
(BLOB)
IFS Files
(LOB FILE)
(LOB FILE)
DB2
for i
ILE RPG

XML_CLOB_FILE

XML_CLOB_FILE is another SQL data type that can be used in ILE RPG.

- It contains valid XML.
- Writing is done directly from SQL, avoiding APIs to write to the IFS.
- It can be populated from a variable of type varchar, xml_clob, xml_dbclob, or directly from a SQL statement selecting only the records we want.
- The data structure generated by the compiler is exactly the same as the one generated with clob_file.
- If the XML is in a character variable, it must be parsed with the SQL function XMLPARSE to obtain a valid XML document.

XML_CLOB_FILE: Data Structure

```
**free
dcl-s ifsFile1 sqltype(xml_clob_file);
// The compiler will generate this:
//   DCL-DS IFSFILE1;
//        IFSFILE1_NL UNS(10);
//        IFSFILE1_DL UNS(10);
//        IFSFILE1_FO UNS(10);
//        IFSFILE1_NAME CHAR(255) CCSID(*JOBRUNMIX);
//   END-DS;
//
// Where:
// _name = File name
// _nl   = File name length
// _fo   = Operation to perform on the file (based on the constants below)
// _dl   = Not used
//
// The compiler adds the following constants:
//   SQFRD  = 2  = Reads the file
//   SQFCRT = 4  = Creates the file
//   SQFOVR = 8  = Overwrites the file
//   SQFAPP = 16 = Appends to the end of the file
```

XMLPARSE (1/3)

Converts a text string with XML content into a **structured XML** data type, ready to be processed or queried. Its main features are:

- **Input:** Character string (CLOB, VARCHAR) with well-formed XML.
- **Output:** Returns a value of the XML data type, which the database can store, validate, and query efficiently.
- **Errors:** If the XML is not well-formed, it generates a parsing error.
- **Whitespace:** Allows deciding whether whitespace is preserved or stripped during parsing.
- **DTD and schemas:** Verifies that the XML is well-formed.

XMLPARSE (2/3)

Some common use cases:

XML Data Ingestion: When XML data is received from external sources (e.g., web services or files) as a text string, XMLPARSE is the first step to convert it into a manipulable XML data type within the database.

XML Storage: It is used before storing XML strings in columns defined with the XML data type.

XML Querying: Once converted to the XML data type, functions such as **XMLQUERY**, **XMLTABLE**, or **XMLCAST** can be used to extract information, transform data, or combine XML with relational data.

XMLPARSE (3/3)

The syntax is:

```
XMLPARSE(DOCUMENT argument {STRIP WHITESPACE | PRESERVE WHITESPACE})
```

- **DOCUMENT** indicates that the input string (argument) represents a complete XML document (and not just a content fragment).
- **STRIP WHITESPACE** removes additional whitespace between XML tags.
- **PRESERVE WHITESPACE** keeps the string as-is, preserving any whitespace between XML tags if present.

The input must be a valid XML document; otherwise, the function will generate an error.

XMLGROUP (1/3)

The SQL function **XMLGROUP** in IBM i is used to generate an XML document from a set of rows.

It is an **aggregation function**, meaning it operates on a group of rows and returns a single XML value.

This function is especially useful when relational data needs to be transformed into a hierarchical XML structure.

Together with XMLPARSE, it allows us to retrieve data from a table and generate a well-formed XML document that can be saved to a file in the IFS.

XMLGROUP (2/3)

Main features and limitations of XMLGROUP:

- **Empty result set:** If the SELECT query returns no rows, the XMLGROUP function returns NULL instead of an empty XML structure. This can cause issues if at least the root element is expected.

- **Rows with all NULL values:** If a row contains only NULL values, it will be completely omitted from the generated XML.

- **Element name restrictions:** It is possible to rename elements using **AS**, but only within the **SELECT** clause or in **subqueries**, not directly inside **XMLROW** or **XMLATTRIBUTES** when used with XMLGROUP.

XMLGROUP (3/3)

The syntax is:

```
XMLGROUP (
   expression [ AS elementName ] [, expression [ AS elementName ] ... ]
   [ ORDER BY sortExpress [, sortExpress ... ] ]
   [ OPTION ROW "rowName" ]
   [ OPTION ROOT "rootName" ]
   [ OPTION AS ATTRIBUTES ]
)
```

- **Expression:** Any SQL expression is valid as the element value. If the expression is not a column, it must be given a name with AS elementName.
- **ORDER BY:** If you want to sort the rows in the XML, use the ORDER BY clause inside the XMLGROUP function itself, not outside the query.
- **OPTION ROW:** If not specified, the name of each row will be <row>.
- **OPTION ROOT:** If not specified, the name of the root element will be <rowset>.
- **OPTION AS ATTRIBUTES:** All values in each row will be written as attributes within the <row> tag instead of as nested elements.

Demo Time!


⚡ LIVE DEMO! ⚡
XML_CLOB_FILE, XMLPARSE & XMLGROUP
IFS Folder Data
XML CLOB FILE
XML XML DATA
IFS
XMLGROUP
XML
XMLPARSE
(LOB FILE)
XML_CLOB_FILE,
ILE RPG
DB2 for i

Thank You!!!

Antonio Salcedo
www.cptserv.com
asalcedo@cptserv.com