Table of Contents

SUMMARY

1. LOB data construction
   1.1 Fundamental issues to store LOB
   1.2 LOB datatypes
   1.3 LOB implementation
   1.4 LOB storage
   1.5 LOB OBID issue
   1.6 Base - LOB association
   1.7 Inline-LOB’s

2. Utilities on LOB
   2.1 Unload - Load
   2.2 Runstats
   2.3 Reorg
   2.4 Copy – Quiesce – Recovery
   2.5 Check data
   2.6 Repair
   2.7 DSN1COPY

3. LOB system fundamentals
   3.1 Database commands
   3.2 Page format
   3.3 Catalog information
   3.4 Bufferpools
   3.5 Logging
   3.6 Locking
   3.7 Sub system parameters
   3.8 LOB – system level significance
SUMMARY

The following document is for IBM DB2 for z/OS, Topic is LOB data administration. As a DB2 DBA administrator every one should have strong knowledge on how to create the objects and how to maintain data by running utilities.

When that object involves LOB data then complexity increases, since creating objects for LOB data and maintaining them by utilities is different when compared to that of for pure RDMS objects.

This paper will give clear idea on how to create objects for LOB data, what issues may arise while creating, what are the considerations should take, how to maintain them with utilities, especially backup and recovery, and some system fundamentals.

This paper will mainly applicable to DB2 versions 10 and 11.

Author
Venkata Rama Rajesh
DB2 DBA for Z/OS
vmallina@in.ibm.com

Any comments or corrections should be forwarded to their attention.

Company: IBM
1. LOB Data Construction

The motive of this chapter is to provide clear knowledge on how to create DB2 objects to store LOB data and also why to create them in that standard manner. While creating objects with LOB data, there may be SQL errors with SQLCODES -171, -670, -747, -904, -764, -766, -109, -356. This chapter also provides a detailed description on these errors.

1.1 Fundamentals issues to store LOB

LOB means Large object. Large text document, audio, video, drawings, mixed text, graphics, are referred as Large OBjects (LOB). Examples for Large objects are

<table>
<thead>
<tr>
<th>Object</th>
<th>Memory size (Approx.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bank cheques, Signatures</td>
<td>30 KB – 40 KB</td>
</tr>
<tr>
<td>Images</td>
<td>30 KB – 40 MB</td>
</tr>
<tr>
<td>Color images</td>
<td>20 MB – 40 MB</td>
</tr>
<tr>
<td>Radiology images</td>
<td>40 MB – 60 MB</td>
</tr>
<tr>
<td>Video</td>
<td>0.5 GB/hour</td>
</tr>
<tr>
<td>Feature length movie</td>
<td>1 GB/hour</td>
</tr>
<tr>
<td>High resolution video</td>
<td>3 GB/hour</td>
</tr>
<tr>
<td>High resolution movie</td>
<td>5 GB/hour</td>
</tr>
<tr>
<td>High definition TV</td>
<td>720 GB/hour</td>
</tr>
</tbody>
</table>

DB2 is a Relational database (RDMS). In recent years it was extended to support non-relational structures like LOB and XML. DB2 as an RDMS faces majorly 2 issues to store LOB data.
Datatype capability

Page size

Datatype Capability

The VARCHAR, VARGRAPHIC, and VARBINARY DB2 data types have a size limit of 32 KB. Although this can be sufficient for small to medium size data, large text document such as audio, video, drawings, mixed text, graphics and images may not fit in 32KB storage. To store such large objects in a column, DB2 has some additional data types called LOB data types.

Page Size

In DB2 base table, the record size should not exceed page size. Even 32K data cannot fit in 32K page, since page also include control information like 20 byte page header, 6 byte row header, 512 byte tail. So, even if string datatypes can support upto 32 K data, DB2 table will not support 32K record. SQL error is received as shown below in such cases.

SQL CODE -670

THE RECORD LENGTH OF THE TABLE EXCEEDS THE PAGE SIZE LIMIT

Explanation

The row length for a table cannot exceed the page size of the table space in which that table resides (or is to reside).
1.2 LOB Datatypes

LOB datatypes are datatypes for LOB columns. LOB datatypes can store data up to 2 GB. There are 3 types of LOB datatypes as follows.

➢ Binary Large Objects (BLOBs)
➢ Character Large Objects (CLOBs)
➢ Double-Byte Character Large Objects (DBCLOBs)

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Description</th>
<th>Maximum limit (n)</th>
<th>Default size</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLOB(n)</td>
<td>To store data such as pictures, movies, signature, voice...</td>
<td>2147483647 Bytes</td>
<td>1M</td>
</tr>
<tr>
<td>CLOB(n)</td>
<td>Varying-length SBCS data to store text documents</td>
<td>2147483647 Bytes</td>
<td>1M</td>
</tr>
<tr>
<td>DBCLOB(n)</td>
<td>Varying length string of double-byte characters</td>
<td>1073741824 Double bytes (2147483648÷2)</td>
<td>1M</td>
</tr>
</tbody>
</table>

• In above table n is the maximum size of lob data in terms of bytes/double bytes that can be stored in column.
• For CLOBs, you can also specify the parameters FOR SCBS, MIXED, or BIT DATA. For BLOBs and DBCLOBs, this is not supported, because BLOBs contain binary data and DBCLOBs have a graphic CCSID.
• Although BLOB strings and FOR BIT DATA character strings might be used for similar purposes, the two data types are incompatible.
• The only supported default value for a LOB column is NULL.
1.3 LOB Implementation

There are three data objects to be created to store and access LOB. They are

- LOB tablespace
- Auxiliary table
- Auxiliary Index

What is LOB tablespace

The auxiliary table needs to reside in a separate tablespace, where actual LOB data is stored, called the LOB tablespace.

What is Auxiliary table

LOB data and relational data cannot be physically stored in same table. If a base table has LOB columns, separate tables have to be created to store that LOB data. Those tables are called auxiliary tables.

What is Auxiliary Index

Every auxiliary table needs an index that completes the table definition, called auxiliary Index. Tablespace scan on LOB tablespace will result in poor performance. To improve the performance, auxiliary index is mandatory.

LOB Column should not have

UNIQUE, Primary or Foreign keys, INDEX, FIELDPROCs, EDITPROCs, VALIDPROCs, and check constraints.

Why is LOB data stored in separate tablespace

An important reason for the separation of data for LOB columns is performance. Assuming table space scans on the base table, LOB values do not have to be processed during these scans. Probably most of the scanning time would be spent in scanning LOB columns if the LOB data resided in the same table as the non-LOB data.
LOB - COMPRESS

Base table can be compressed, but LOB tablespace cannot be compressed. This is decided in design phase to avoid complexity for DB2 in doing compression and decompression of large data, which was also a time consuming process.

```sql
ALTER TABLESPACE MALLIV.T1GKX030
    COMPRESS YES;
```

DSNT408I SQLCODE = -109, ERROR: COMPRESS CLAUSE IS NOT PERMITTED
DSNT418I SQLSTATE  = 42601 SQLSTATE RETURN CODE

SQL CODE -109

clause-type CLAUSE IS NOT PERMITTED

Explanation

A clause was used where it is not permitted in SQL statement.

LOB implementation scenarios

Let us consider some scenarios

➢ Segmented tablespace having table with single LOB column
➢ Segmented tablespace having table with multiple LOB columns
➢ Partitioned tablespace having table with multiple LOB columns

Segmented tablespace having table with single LOB column

The Lob column values will not be in base tablespace along with Non-Lob columns, they will be stored in separate table called auxiliary table which will be present in LOB tablespace as shown in below example.
Below syntax shows creating base table with LOB column.

```sql
CREATE TABLE VX$B5B1.EMP_DATA
    (ID INT
    ,NAME CHARACTER(45)
    ,DEPT CHARACTER(1)
    ,PHOTO BLOB(50K)
    ) IN MALLIV.TSEMP
DSNT400I SQLCODE = 000, SUCCESSFUL EXECUTION
```

When you create a table with LOB column or alter a table to add LOB column then the table should include a ROWID column as well. Defining a ROWID column is optional. If you do not define a ROWID column, DB2 defines a hidden ROWID column for you. Define only one ROWID column, even if multiple LOB columns are in the table.
The above DDL specifies ID, NAME, DEPT, PHOTO columns. An extra ROWID column is implicitly created by DB2. Below snapshot shows implicitly created ROWID column (DB2_GENERATED_ROWID_FOR_LOBS).

- Now we have to create LOB tablespace to store LOB data (PHOTO) physically. CREATE LOB TABLESPACE <tablespace name> is the basic syntax for LOB tablespace. There will be no partitions, no segments in LOB tablespace. LOB tablespace is a single table tablespace.

```sql
CREATE LOB TABLESPACE TSXPHOTO
    IN MALLIV
    DSSIZE 4G
    LOG NO
DSNT400I SQLCODE = 000, SUCCESSFUL EXECUTION
```

- Now we need to create auxiliary table in that LOB tablespace. The basic syntax to create auxiliary table shown below

Syntax: CREATE AUXILIARY TABLE < auxiliary table name>

    IN < LOB tablespace name>

    STORES <base table name>

    COLUMN <LOB column name>

```sql
CREATE AUXILIARY TABLE VX$B5B1.EMP_DATA_AUX
    IN MALLIV.TSXPHOTO
    STORES VX$B5B1.EMP_DATA
    COLUMN PHOTO;
DSNT400I SQLCODE = 000, SUCCESSFUL EXECUTION
```

Create LOB tablespace is for LOB storage, creating auxiliary table is for RDBMS access. But this does not complete base table definition. Creating auxiliary index on auxiliary table will complete the table definition. Creating auxiliary index is mandatory.
We should not mention column name in auxiliary index creation. The basic syntax for creating auxiliary index is

```
CREATE INDEX <index name>
ON < auxiliary table name>
```

Below is the example for creating auxiliary index.

```
CREATE INDEX VX$B5B1.IX
ON VX$B5B1.EMP_DATA_AUX
DSNT400I SQLCODE = 000, SUCCESSFUL EXECUTION
```

Now the base table definition is complete. In our examples

<table>
<thead>
<tr>
<th>Base segmented tablespace</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base table</td>
<td>-</td>
</tr>
<tr>
<td>Lob tablespace</td>
<td>-</td>
</tr>
<tr>
<td>Auxiliary table</td>
<td>-</td>
</tr>
<tr>
<td>Auxiliary index</td>
<td>-</td>
</tr>
</tbody>
</table>

© Copyright IBM Corporation, 2013
Segmented tablespace having table with multiple LOB columns

Each LOB column in a base table requires its own LOB table space, each LOB table space will have auxiliary table that will contain the values of an LOB column, each auxiliary table will have an auxiliary index as shown in below example.

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Movie</th>
<th>Photo</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Raju</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Phani</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Partitioned tablespace having table with multiple LOB columns

If the base table is partitioned, every LOB column in each partition has its own LOB tablespace with Auxiliary table and Auxiliary Index. If table has m columns and n partitions then m*n LOB tablespaces with auxiliary tables and indexes are
If tablespace is PBG then we have to create the auxiliary objects only for first
partition. DB2 automatically create the LOB tablespace, auxiliary table and auxiliary index for the new partitions. Following example shows creating auxiliary table for the first partition.

```sql
CREATE AUXILIARY TABLE VX$B5B1.GVRN_FILE NY3A8YYJ IN MALLIV.LNY3ZX@@ STORES VX$B5B1.GVRN_FILETEMP APPEND NO COLUMN FILE_OBJ PART 1;
```

Why shouldn't we have single LOB tablespace for all partitions?
Consider that the LOB data for all partitions are in a single LOB tablespace. The SQLs will be scanning the LOB data belonging to other partitions also, which will affect the performance of query accessing a particular partition.

To Summarize, here are the steps for completing the definition of a table with LOB columns

1. Define a column of the appropriate LOB type.
2. Implicit or explicit ROWID column is needed in the base table.
3. Create an LOB table space and AUX table for each LOB column and for each partition. If tablespace is PBG, creating for first partition would suffice.
4. Create AUX index on each auxiliary table.

What will happen if we not followed above steps? SQLCODE -747.
If we do not properly follow the above procedure to complete base table definition we will get this sqlcode. The causes can be

- an auxiliary table for storing LOB columns has not be created.
- an index has not been created for an auxiliary table.
- there is not an auxiliary table for each partition of the table space.
- auxiliary table for partition does not have part clause.
**SQL CODE -747**

TABLE table-name IS NOT AVAILABLE UNTIL THE AUXILIARY TABLES AND INDEXES FOR ITS EXTERNALLY STORED COLUMNS HAVE BEEN CREATED

Note:

If tablespace is implicitly created for base table containing LOB columns then LOB tablespace, auxiliary table and auxiliary index will also be implicitly created.

**SQL error -766**

If we perform the below operations on auxiliary table we will get SQLCODE -766

- Auxiliary table attributes cannot be ALTERed, except APPEND attribute
- CREATE ALIAS or CREATE FUNCTION or CREATE SYNONYM or CREATE VIEW is not allowed on auxiliary table
- DELETE / INSERT / MERGE / SELECT / TRUNCATE / UPDATE (any DML activities) cannot be performed on auxiliary table.
- DESCRIBE TABLE SQL cannot be performed on auxiliary table.

Example:

```sql
SELECT AUXID,AUXVER,AUXVALUE FROM VX$B5B1.GVRN_FILE_AUX;
```

**SQL CODE -766**

THE OBJECT OF A STATEMENT IS A TABLE FOR WHICH THE REQUESTED OPERATION IS NOT PERMITTED
1.4 LOB – OBID Issue

The LOB tablespaces belonging to a base table has to be in the same database where the base table space resides. Otherwise we will get -764 sql error.

**SQL CODE -764**

A LOB TABLE SPACE AND ITS ASSOCIATED BASE TABLE SPACE MUST BE IN THE SAME DATABASE

A single database can hold a maximum of 65,535 objects, or to be more specific, X'FFFF' object identifiers (OBIDs). Let us consider a database having a partitioned tablespace with a table having n LOB columns and p partitions. The below table describes how many OBID's / PSID's are utilized for that setup.

<table>
<thead>
<tr>
<th>Partitioned tablespace with base table</th>
<th>For Partitioned tablespace with base table there will be 3 ID's utilized</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSID, OBID</td>
<td>1. Base Tablespace PSID</td>
</tr>
<tr>
<td></td>
<td>2. Base Tablespace OBID</td>
</tr>
<tr>
<td></td>
<td>3. Base table OBID</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n LOB columns</th>
<th>For one LOB relation at one partition level, there will be 5 ID's utilized</th>
</tr>
</thead>
<tbody>
<tr>
<td>p Partitions</td>
<td>1. LOB Tablespace PSID</td>
</tr>
<tr>
<td></td>
<td>2. LOB Tablespace OBID</td>
</tr>
<tr>
<td></td>
<td>3. Auxiliary table OBID</td>
</tr>
<tr>
<td></td>
<td>4. Auxiliary Index PSID</td>
</tr>
<tr>
<td></td>
<td>5. Auxiliary Index OBID</td>
</tr>
<tr>
<td></td>
<td>So 5<em>n</em>p ID's will utilized here</td>
</tr>
</tbody>
</table>

| n LOB columns                        | There will be one OBID for each LOB column (LOB relation). So for n LOB columns n OBID's will utilized. (Refer: AUXRELOBID in SYSIBM.SYSAUXRELS) |

© Copyright IBM Corporation, 2013
3 + n + 5np ID’s will be uses for table with LOB data which should be less then the 65535 – I, where i = PSID’S & OBID’s used by other objects of same database Therefore 3 + n + 5np <= 65535 – i, with n as the number of LOB columns in your base table and p as the number of partitions. The formula gives you the number of partitions and LOB columns that can reside inside one database. Below table gives an example with numerical values (If i = 0, it means only one base table present in database).

<table>
<thead>
<tr>
<th>Partitions</th>
<th>Max. number of LOB columns</th>
<th>Total ID's</th>
</tr>
</thead>
<tbody>
<tr>
<td>4,096</td>
<td>3</td>
<td>61446</td>
</tr>
</tbody>
</table>

If the base table is in a PBG, there may be a situation where OBID’s are exhausted for new partitions.

1.5 LOB – Storage

For an LOB tablespace maximum 254 LDS datasets can be created. DSSIZE determines maximum size of each dataset. If you omit the DSSIZE, default is 4G.

CREATE LOB TABLESPACE " ....... ” →

Maximum LOB data in LOB tablespace = 4 GB x 254

= 1024 GB

= 1 TB (Approx.)
The maximum size of a LOB tablespace is 16 TB which can be obtained at DSSIZE 64 GB.

CREATE LOB TABLESPACE ...... DSSIZE 64 GB →

```
64 GB  64 GB  64 GB  64 GB  64 GB
  1     2     3     4     254
```

Maximum LOB data in LOB tablespace = 64 GB x 254
= 16,256 GB
= 16 TB

In DB2 V10 and V11, DSSIZE can be 128 GB or 256 GB. Even in this scenario also, maximum LOB tablesapce size is 16 TB only (number of datasets will be managed automatically)

Maximum LOB data that fit in single LOB column
- If base table is in segmented tablespace, 16 TB is the maximum limit, since one LOB column will have only one LOB tablespace.

Maximum LOB data in single LOB column = 16 TB

- If base table is in partitioned or universal tablespace, then 16 TB x n is the maximum limit, since one LOB column can have n LOB tablespaces (one LOB tablespace for one partition).

Maximum LOB data in single LOB columns = 16 TB x n
= 16 TB x 4096 (at n=n\text{max})
= 65,536 TB
= 64 PB (Approx.)
1.6 Base – LOB association

What is LOB Indicator

We know that LOB data will not be stored in base table. Then, what will be stored in base table? LOB column in base table will not have any pointer which refers to the LOB data in LOB tablespace. It will have LOB Indicators. Below diagram shows DB2 catalog information of a base table with two LOB columns (Book_Text, Book_Cover).

*The LOB indicators are stored as VARCHAR (4) columns, resulting in a total of 6 bytes, including the length prefix (which will be always 4).*

<table>
<thead>
<tr>
<th>Content of LOB indicator columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length [Hex]</td>
</tr>
<tr>
<td>00 04</td>
</tr>
</tbody>
</table>

• After length, next 2 bytes are called flags, which has the information about NULL. This column provides information on whether LOB column has data or not. If it has data, only then LOB tablespace will be accessed.
• Last 2-bytes contains the current version number of the LOB. Every LOB in LOB tablespace has a version number.

© Copyright IBM Corporation, 2013
What is LOB version number

Initially all LOB's in the LOB tablespace will have version number of X'0001'. When any LOB data is changed by update SQL, the version number of the updated LOB data is X'0002'. The LOB data before the update will not be removed from the LOB tablespace, but the updated data will be added with a new version number. For DB2 to access the active LOB data, i.e., the LOB data after update, the version number has to be used. DB2 stores the active version number in LOB column of the base table.

Example:

For ID=1, Photo column has photo with name raju.jpeg, raju.jpeg will have version number X'0001'. Now you are changed that LOB raju.jpeg to ahan.jpeg using update SQL. Then ID=1 will have ahan.jpeg at photo column. The version number of LOB ahan.jpeg is X'0002'. Now the current active version is X'0002'. If you change ahan.jpeg to raju.jpeg by recover utility, then the current active version will become X'0001' which is version number of raju.jpeg.

Row ID Column

- Row ID is a 17 byte column and will have data in internal format. In readable format (hex-decimal format), it is 40 bytes. (Refer LENGTHTH2 column of SYSIBM.SYSCOLUMNS)
- This column is not included in the result set of a SELECT * from base table, but by selecting the column explicitly by name, you can retrieve its content.
• If you omit an explicit ROWID definition when a LOB column is present in your CREATE TABLE statement, DB2 creates a ROWID column using GENERATED ALWAYS method for you.

• When implicitly created, DB2 creates the column with a name of **DB2_GENERATED_ROWID_FOR _ LOBSnn**. DB2 appends `nn` only if another column with the same name already exists in the table, replacing `nn` with `00` and increment by 1 until the name is unique in the table.

• When using GENERATED ALWAYS keyword, DB2 always generates a ROWID when inserting a row. Applications and users are not allowed to insert a ROWID.

• If you use GENERATED BY DEFAULT, users and applications can supply a value for the ROWID column.

• It is a NOT NULL column.

• You have to create a unique index on the ROWID column when you specify GENERATED BY DEFAULT.

**Auxiliary table:**

• We defined only one column for the auxiliary table, but DB2 implicitly added two more columns for the table as shown below

<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>CREATOR</th>
<th>COLUMN_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMP_DATA_AUX</td>
<td>VX$B5B1</td>
<td>AUXID</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AUXVER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AUXVALUE</td>
</tr>
</tbody>
</table>

Below diagram shows catalog information for these columns

```
Catalog description

<table>
<thead>
<tr>
<th>AUXID</th>
<th>AUXVER</th>
<th>AUXVALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR (17)</td>
<td>SMALLINT (2)</td>
<td>BLOB (4)</td>
</tr>
</tbody>
</table>

LENGTH = 1,648,578
```
<table>
<thead>
<tr>
<th>S. No</th>
<th>Column name</th>
<th>Datatype</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AUXID</td>
<td>VARCHAR</td>
<td>17</td>
</tr>
<tr>
<td>2</td>
<td>AUXVER</td>
<td>SMALLINT</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>AUXVALUE</td>
<td>BLOB</td>
<td>4</td>
</tr>
</tbody>
</table>

✔ **AUXID**: ROWIDs of corresponding row in base table.

✔ **AUXVER**: Current version number of the LOB.

✔ **AUXVALUE**: Here 4 is not actual length of LOB. Refer LENGTH2 column of SYSIBM.SYSCOLUMNS for actual length. This AUXVALUE refers actual LOB data.

**Auxiliary Index**

An index defined on an auxiliary table is automatically defined as a unique index. Internally DB2 will build the INDEX on AUXID & AUXVER columns, for unique identification of LOB data.

<table>
<thead>
<tr>
<th>INDEX_NAME</th>
<th>COLUMN_NAME</th>
<th>COLSEQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>116KX030</td>
<td>AUXID</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>AUXVER</td>
<td>2</td>
</tr>
</tbody>
</table>

The ROWID from base table will be matched with AUXID of auxiliary table to retrieve data from auxiliary table. If an LOB has more than one version in LOB tablespace, then ROWID will map with many LOB versions in auxiliary table. This is the reason why version number is kept in base table as well. Rowid and version number together provides one to one mapping between data record in base table and LOB data in auxiliary table, it also provides uniqueness to the data.
Consider below example. Suppose in base table Id=1 had an update in LOB column. Then there will be two LOB’s for same data record (before and after update). Row id and version number together provides unique identification of LOB data for the data record.

**Base table**

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Dept</th>
<th>Photo</th>
<th>Row ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Raju</td>
<td>XXX</td>
<td>(4, null_info, 2)</td>
<td>---------</td>
</tr>
<tr>
<td>2</td>
<td>Phani</td>
<td>YYY</td>
<td>(4, null_info, 1)</td>
<td>---------</td>
</tr>
</tbody>
</table>

**Auxiliary table**

<table>
<thead>
<tr>
<th>AUXID</th>
<th>AUXVER</th>
<th>AUXVALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
1.7 In-line Lobs

- An inline LOB allows a portion of LOB data to reside in the base tablespace along with non-LOB columns data. Inline LOB option is only available from V10, that too only for universal tablespaces.

- For LOBs of size less than or equal to the specified inline length, DB2 stores the complete LOB data in the base table space. DB2 does not need to access the LOB table space or auxiliary indexes for processes that access the LOB data.

- In such cases, DB2 can access the data at similar cost, in terms of CPU time and elapsed time as non-LOB data types.

```sql
create table mytab1 (a int, b char(5), c clob inline length 1000)
```

- For LOBs of size greater than the specified inline length, the inline portion of the LOB resides in the base table space and DB2 stores the remaining part of LOB in the LOB table space. In this case, any process that accesses the LOB data must access both the base table space and the LOB table space.

- INLINE LENGTH cannot be specified if FOR SBCS DATA or FOR MIXED DATA is also specified. The acceptable values for inline length is 0 to 32680 for CLOB and BLOB, 0 to 16340 for DBCLOB. Since 32K is the limit.

- But practically while creating base table we cannot specify Max limit for Inline length, since DB2 will not support record size gather than or equal to 32K. Below example shows SQL error for giving more inline length which is equal to: 

```sql
create table mytab1 (a int, b char(5), c clob inline length 32750)
```
• Below is the example for creating base table with in line LOB.

```
CREATE TABLE VX$B5B1.EMP_DATA_IN
  (ID INTEGER
   WITH DEFAULT NULL
   , "NAME" CHARACTER(45) FOR SBCS DATA
   WITH DEFAULT NULL
   , DEPT CHARACTER(1) FOR SBCS DATA
   WITH DEFAULT NULL
   , PHOTO BLOB(50K) INLINE LENGTH 32680
   WITH DEFAULT NULL
  )
  IN DATABASE MALLIV
APPEND NO
NOT VOLATILE CARDINALITY
DATA CAPTURE NONE
AUDIT NONE
CCSID EBCDIC;
```

```
DSNT400I SQLCODE = -670, ERROR: THE RECORD LENGTH OF THE TABLE EXCEEDS THE PAGE SIZE LIMIT
```

• For inline LOB, in base table LOB column length is Inline_length + 4. See the below BLOB length, inline_length defined as 30,000.

```
<table>
<thead>
<tr>
<th>COLTYPE</th>
<th>LENGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>CHAR</td>
<td>45</td>
</tr>
<tr>
<td>CHAR</td>
<td>1</td>
</tr>
<tr>
<td>BLOB</td>
<td>30,004</td>
</tr>
<tr>
<td>ROWID</td>
<td>17</td>
</tr>
</tbody>
</table>
```
LOB_INLINE_LENGTH Subsystem Parameter

LOB_INLINE_LENGTH is the subsystem parameter which can provide default inline length. The default value for this parameter is 0.

LOB to Inline LOB conversion

We cannot change LOB datatype length. But we can change inline length of LOB columns using ALTER TABLE statement. This is the way to convert normal LOB’s to inline LOB’s. Reorg is needed after ALTER.

Index on LOB Column

We cannot create index on LOB columns. But we can create expression based index on Inline LOB columns. Only SUBSTR function is allowed in the expression.

Consider the below example

Base table creation with inline LOB:

```sql
CREATE TABLE MYTAB (CLOBCOL CLOB(1M) INLINE LENGTH 10);
```

Expression based index creation on in-line LOB column:

```sql
CREATE UNIQUE INDEX MYINDEX1 ON MYTAB (VARCHAR (SUBSTR(CLOBCOL,1,10))) ;
```

And you can issue the statement:

```sql
SELECT CLOBCOL From MYTAB WHERE VARCHAR(SUBSTR(CLOBCOL,1,10)) = 'ABCDEFGHIK'
```

Below snapshot is for Expression based index on inline LOB column.

![Expression based index on inline LOB column]

We cannot create expression based index on inline BLOB column. It will show SQL error as below. Since SUBSTR is a string function.

```
CREATE UNIQUE INDEX VX$B5B1.MYINDEX1 ON VX$B5B1.EMP_DATA_OUT (VARCHAR (SUBSTR(RESUME,1,10))) ;
DSNT400I SQLCODE = 000, SUCCESSFUL EXECUTION
```
Below diagram shows SQL error while trying to create expression based index using left function on in-line LOB column.

**SQL CODE -171**

THE DATA TYPE, LENGTH, OR VALUE OF ARGUMENT argument-position OF function-name IS INVALID

**Explanation**

The data type, the length, or the value of an argument of a scalar function is incorrect.

Below diagram shows SQL error while trying to create expression based index using left function on in-line LOB column.

**SQL CODE -356**

KEY EXPRESSION expression-number IS NOT VALID, REASON CODE = rc

**Explanation**

For rc=12, The built-in function is allowed to reference the inline portion of a LOB in the specified context. In addition, the START and LENGTH arguments of the SUBSTR function must be constants.

It shows SUBSTR is the only function which can be used in expression based index for in-line LOB column.
Alters available for INLINE LOBs

Alter Add INLINE LENGTH to existing LOB column

✔ Space left in AREO status after alter
✔ Cannot create index on expression until after a reorg

Alter increase existing INLINE LENGTH

✔ Space left in AREO status after alter
✔ Newly inserted rows will have new inline length
✔ Cannot increment to length greater than LOB column length

Alter decrease existing INLINE LENGTH

✔ Space left in REORP status after alter
✔ Cannot decrement if space has index on expression
✔ Cannot decrement if new inline length is less than default

Note: REORG UNLOAD ONLY and REORG UNLOAD EXTERNAL are not available for tables with INLINE LOB

Inline LOB Impact

There are some drawbacks of using inline LOB's as shown below

• Large size Image copies
• Big Log records
• Bufferpool Hit ratio will reduce for base table
• Inline LOB's can compress, but there is no tools which will predict how LOB's are compressed.
• Impact on maximum non-relational data stored in base tablespace pageset.
2. Utilities on LOB

2.1 Unload – Load

If a table contains LOB columns then we cannot unload data into PS dataset with fixed block (FB) or Fixed(F) record format. Unload utility will issue following error.

```sql
OUTPUT START FOR UTILITY, UTILID = TEMP
PROCESSING SYSIN AS EBCDIC
UNLOAD TABLESPACE DXGBK030.T1GBK030
   - LOGICAL RECORD LENGTH OF OUTPUT RECORD EXCEEDED THE LIMIT FOR TABLE

UTILITY EXECUTION TERMINATED, HIGHEST RETURN CODE=8
******************************************************************************
```

There are two ways to use the unload utility with LOB data

1. UNLOAD using SPANNED YES option
2. UNLOAD LOB data using file reference variables

**UNLOAD using SPANNED YES option**

- SPANNED YES option is used to unload data into a VBS data set (spanned record format). This will be used when table has LOB or XML columns.
- If SPANNED YES is mentioned in the UNLOAD control card and RECFM is not specified for new SYSREC dataset, then by default VBS dataset is created. Even if RECFM = FB is mentioned it will be overridden to create VBS dataset only.
- You cannot browse the SYSREC. If you try to open SYSREC you will get Invalid Record length or Invalid Blocksize. We can see VBS or question mark (?) in the record format field of dataset information.

```
Organization . . . : PS
Record format . . . : VBS
```

- Below snapshot is sample JOB for UNLOAD with SPANNED YES option.
As shown in above diagram, for Spanned YES option we should specify all columns names and its datatypes in control card.

We should not specify LOB column size in unload card, otherwise job will abend with RC 08.

In unload card LOB columns should be specified at the end, irrespective of the order of columns in the table. Otherwise job will abend with RC 08.

SPANNED YES will be ignored if you are performing unload from image copy.

Do not specify DELIMITED with SPANNED YES.

This file can reside on DASD and can span multiple volumes.

Following diagram shows the SYSPUNCH created for above job, which can be used to perform load without any issues.
Spanned record format:

The following figure shows a conceptual example of a spanned record that was unloaded where C1 to C8 are non LOB column name and CLOB1, CLOB2, CLOB3 are LOB column names. Here, after non-LOB data LOB data will stored for each record.

```
LOAD DATA INDDN SYSREC LOG NO RESUME YES
EBCDIC CCSID(00037,00000,00000)
FORMAT SPANNED YES
INTO TABLE "VXS$5B1"."GVRN_FILE"
WHEN(00001:00002) = X'0017'
NUMRECS 116
IGNOREFIELDS YES
("TRNSM_SRS"
  POSITION( 00003:00022) CHAR(00020)
  , "ENG_MDL_DESC"
  POSITION( 00023:00067) CHAR(00045)
  , "LD_IND"
  POSITION( 00068:00068) CHAR(00001)
  , "ENG_RPM"
  POSITION( 00069:00071) DECIMAL PACKED
  , "FRQ_CD"
  POSITION( 00072:00073) CHAR(00002)
  , "SW_TO_IND"
  POSITION( 00074:00074) CHAR(00001)
  , "FILE_NM"
  POSITION( 00075:00114) CHAR(00040)
  , "DSN_ROWID"
  POSITION( 00115) ROWID
  , "DSN_NULL_IND_00000" POSITION( *) CHAR(1)
  , "FILE_DESC"
  POSITION( *) VARCHAR NULLIF(DSN_NULL_IND_00000)=X'FF'
  , "FILE_OBJ"
  POSITION( *) BLOB )
```
UNLOAD LOB data using file reference variables

This is the method you probably use most when unloading tables containing LOB columns. With this method, the LOB values are unloaded to a different file than the normal unload file. DB2 creates or uses a different output file for each LOB value to be unloaded. The output file can be one of the following.

- Member of a partitioned data set (PDS)
- Member of a partitioned data set extended (PDSE)
- Hierarchical File System (HFS) file on a HFS directory

It cannot be a simple sequential data set (this was not implemented because of performance reasons: allocate, open, write, close, and deallocate of a sequential file for each LOB value would have been a very costly operation). Below job is an example for this concepts where LOB’s are stored as PDS members. This PDS should be allocated using template before UNLOAD card.

```
//UNLOB001 EXEC DSNUPROC,SYSTEM=Z1U2,UID='',UTPROC=''
//DSNUPROC.SYSPRINT DD SYSOUT=* 
//SYSPUNCH DD DSN=VX$B5B1.Z1U2.UNLLOB01.SYSPUNCH,
// DISP=(NEW,CATLG,DELETE),
// SPACE=(TRK,(5,5),RLSE)
//DSNUPROC.SYSREC DD DSN=VX$B5B1.Z1U2.UNLLOB01.SYSREC,
// DISP=(NEW,CATLG,DELETE),
// UNIT=SYSDA,SPACE=(CYL,(200,200),RLSE)
//DSNUPROC.SYSIN DD *

TEMPLATE TSYSLOBA DSN 'VX$B5B1 &DB .&TS .LOBDATA'
DSNTYPE(PDS) UNIT(SYSDA)
DISP(NEW,CATLG,CATLG)
SPACE(200,200) CYL

UNLOAD DATA
FROM TABLE DXGK001$.GVRN_FILE
(TRNSM_SRS CHAR(20))
,ENG_MDL_DESC CHAR(45)
,LD_IND CHAR(1)
,ENG_RPM DECIMAL(4, 0)
,FRQ_CD CHAR(2)
,SW_TO_IND CHAR(1)
,FILE_NM CHAR(40)
,BLOB_COL_ID ROWID
,FILE_OBJ CHAR(53) BLOBF TSYSLOBA
,FILE_DESC VARCHAR(300) )
SHRLEVEL CHANGE ISOLATION UR

© Copyright IBM Corporation, 2013
• In this method also we should specify all columns. But LOB columns may not be at the end.

• In Control card at LOB column we should specify below syntax.
  
  \[
  \text{<LOB Column>} \quad \text{CHAR(n)} \quad \text{<BLOBF/CLOBF/DBCLOBF>} \quad \text{<template name>}
  \]

• The description for BLOBF, CLOBF, DBCLOBF keywords is given below.
  
  BLOBF: The output field contains the name of a file to store a BLOB field.

  CLOBF: The output field contains the name of a file to store a CLOB field.

  DBCLOBF: The output field contains the name of a file to store a DBCLOB field.

• Here \text{<template name>} refers to template name of PDS or PDSE defined in the above UNLOAD card for LOB data.

• In addition, we will require a PS file which stores non-LOB data.

• In this format, you can browse the PDS members for LOB data.

• In UNLOAD card, CHAR(53) at LOB column does not mean datatype. It means the maximum length occupied by the PDS name with member name.

• If you see the PS SYSREC dataset, instead of LOB column values, dataset and member names will be present (where LOB is stored) as shown below.

  \[
  \begin{align*}
  &\text{VX$BSB1.DXGB0030.T1G6X030.L0BDATA(FPM0BA5K)} \quad \text{LCG2} \\
  &\text{VX$BSB1.DXGB0030.T1G6X030.L0BDATA(FPM0BA6G)} \quad \text{LCG2} \\
  &\text{VX$BSB1.DXGB0030.T1G6X030.L0BDATA(FPM0BA6K)} \quad \text{LCG1} \\
  &\text{VX$BSB1.DXGB0030.T1G6X030.L0BDATA(FPM0BA6D)} \quad \text{LCG1} \\
  &\text{VX$BSB1.DXGB0030.T1G6X030.L0BDATA(FPM0BA6W)} \quad \text{LCG2}
  \end{align*}
  \]

• File Reference Variables (FRV) limitations
  
  ▪ PDS/PDSE is limited to one volume
  
  ▪ PDS is limited to 64K tracks
  
  ▪ PDSE is limited to 512K members
  
  ▪ Load will take relatively more time with FRV

• Below diagram shows the SYSPUNCH created for above method. We can use these Load card for LOAD, but this Load will take relatively more time when compared to VBS method, since it has to read data for PS and PDS synchronously.

© Copyright IBM Corporation, 2013
Beware of following tasks during LOAD

LOAD LOG YES for NOT LOGGED TABLESPACE

For NOT LOGGED tablespace, LOAD with LOG YES will give message DSNU1153I which shows nothing is logged. The base table space is put in INFORMATIONAL COPY PENDING state and the LOB table space has no pending states after the LOAD.

Auxiliary Indexes are not built at LOAD-build phase

Auxiliary indexes are not built during the BUILD phase. LOB values are inserted (loaded) into auxiliary tables during the RELOAD phase one by one. or every LOB data insertion , LOAD utility inserts keys into an auxiliary index, free space within the index might be consumed and index page splits might occur. Consider reorganizing an index on the auxiliary table after LOAD completes, to introduce free space into the index for future INSERT's and LOAD's. See below diagram (LOAD job sysprint), there is no build phase.

© Copyright IBM Corporation, 2013
In-line image copy, In-line stats during Load

If we specify COPYDDN, STATISTICS keywords in LOAD card then inline copy and inline stats will be taken only for base table. Inline stats and inline copy are not supported on auxiliary table during load. We can notice this from below snapshots taken from LOAD job sysprint.

DSNU068I - DSNURWI - FUNCTION INLINE COPY IS NOT SUPPORTED FOR LOB OR XML TABLE SPACES

DSNU068I - DSNURWI - FUNCTION INLINE STATISTICS IS NOT SUPPORTED FOR LOB OR XML TABLE SPACES

Cross Loader control card:

• To transfer data from one location to another location or from one table to another table at the same location, we can use a cursor. This method of loading data is called the cross-loader function.
• For this method, the sum of the lengths of the non-LOB columns plus the sum of 8 bytes per LOB column cannot exceed 32 KB. DB2 uses a separate buffer for LOB data.

• If the SELECT statement in the cursor definition contains the ROWID column of the source table, the table to be loaded must have a ROWID column with the same name and defined as GENERATED BY DEFAULT.

• In the below example ARE_EXTRACT is the LOB column

```sql
EXEC SQL
DECLARE C1 CURSOR FOR
  SELECT
    ARE_SETCODE
  , ARE_SUBSETCODE
  , ARE_MAJOR
  , ARE_MINOR
  , ARE_REVISION
  , ARE_TYPE
  , ARE_ENABLED
  , ARE_EXTRACT
  , ARE_DATE
  FROM UKLGDB2N.CNBS.AURA_RULE_EXTRACT
ENDEXEC
LOAD DATA LOG YES REPLACE
INCURSOR C1
INTO TABLE CNBS.AURA_RULE_EXTRACT
```

Restart Load Job

• If you restart a LOAD job with the RESUME YES and SORTKEYS NO options for a table that has LOB columns, you must use RESTART(CURRENT).

• If you use RESTART(PHASE) to restart a LOAD job that specified RESUME NO, the LOB table spaces and auxiliary indexes are reset.

• For a table that has LOB columns, you cannot restart a LOAD job that uses the INCURSOR option.
2.2 Runstats

Runstats should be run on base tablespace and LOB tablespace separately. There is no option to run Runstats on base table and dependent auxiliary table together. Below picture shows an example for runstats on LOB tablespace.

```
//UTILITY EXEC DSNUPROC, SYSTEM='Z1U2', UID='TEMP', UTPROC=''
//SYSIN DD *
RUNSTATS TABLESPACE MALLIV.T1GKX030
INDEX(ALL)
UPDATE ALL
REPORT YES
```

For Runstats on LOB tablespace, **TABLE(..)** is an invalid keyword. It will throw RC 08 with message DSNU076I in sysprint.

```
RUNSTATS TABLESPACE MALLIV.T1GKX030 TABLE(ALL) REPORT YES
L - KEYWORD=TABLE INVALID FOR LOB TABLE SPACE=MALLIV.T1GKX030
UTILITY EXECUTION TERMINATED, HIGHEST RETURN CODE=8
```

SYSIBM.SYSCOLUMNS will be updated by default.

```
43 DSNUSUCO - SYSCOLUMNS CATALOG STATISTICS FOR AUXID
COLCARD = -2
COLCARDF = -2.00E+00
HIGH2KEY = X'404040404040404040'
LOW2KEY = X'404040404040404040'
43 DSNUSUCO - SYSCOLUMNS CATALOG UPDATE FOR VX$S5B1.T1GKX030 SUCCESS
```

Apart from SYSTABLESPACE, SYSTABLEPART, SYSTABLES, SYSINDEXES, SYSINDEXPART, SYSCOLUMNS etc, one more table will updated for LOB tablespace i.e **SYSLOBSTATS**. Where LOB statistics is stored.

```
05:48:02.04 DSNUSLOB - SYSLOBSTATS CATALOG
AVGSIZE = 30984
FREESPACE = 0
ORGRTATIO = 100.00
```
2.3 Reorg

Offline Reorg

REORG SHRLEVEL NONE on base tablespace containing LOB columns can be done. But we cannot perform offline reorg for LOB tablespace. If we try to do, then job will execute successfully with return code 0. It doesn't mean offline reorg is done. It will show warning message as shown below.

```
- OUTPUT START FOR UTILITY, UTILID = TEMP
- PROCESSING SYSPRINT AS EBCDIC
- REORG TABLESPACE MALLIV.T1GKX030 SHRLEVEL NONE

FIT - REORG SHRLEVEL NONE ON LOB TABLE SPACE IS NO LONGER SUPPORTED
- UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=0
```

REORG LOB SHRLEVEL NONE will work before DB2 V10. But it does not reclaim fragmented space, it will ‘re-chunk’ the LOBs.

Online Reorg (SHRLEVEL REFERENCE or SHRLEVEL CHANGE)

- Below screen shots shows Reorg with shrlevel reference (base tablespace & LOB tablespace).

```
//DSNUPROC.SYSIN DD *
  REORG TABLESPACE MALLIV.T1GKB030 SHRLEVEL REFERENCE
```

But these reorgs will fail. Because online reorg of base tablespace containing LOB columns requires inline image copy. Below screen shots shows failing reorg due to lack of image copies. To avoid this error add SYSCOPY DD statement in the job.

```
OUTPUT START FOR UTILITY, UTILID = TEMP
PROCESSING SYSPRINT AS EBCDIC
  REORG TABLESPACE MALLIV.T1GKB030 SHRLEVEL REFERENCE
IT - A REQUIRED DD CARD OR TEMPLATE IS MISSING. NAME=SYSCOPY
UTILITY EXECUTION TERMINATED, HIGHEST RETURN CODE=8
```
AUX Keyword

- During REORG of PBG, data movement may happen across partitions. But data cannot move across partitions when an LOB exists, since corresponding LOB data movement across LOB tablespaces is also required.
- There will be similar issue for REORG REBALANCE for classic partitioned tablespace and Range partitioned tablespace.
- As of DB2 V10, the new parameter AUX can be specified to permit data movement across partitions. The AUX keyword allows you to specify whether DB2 reorganizes associated auxiliary LOB table spaces along with the base table space.
- Specifying AUX keyword enables data movement across partitions during PBG REORG, REORG REBALANCE of classic partitioned or PBR.

Below statements are sample control cards with AUX YES for reorg of classic partitioned tablespace.

```plaintext
// DSNUPROC.SYSIN  DD  *
REORG TABLESPACE MALLIV.T1KKBLB1
SHRLEVEL CHANGE
MAPPINGTABLE VX$B5B1.MAPPING
AUX YES
```
- REORG AUX YES is very costly and time consuming process. Because it will do reorg of base tablespace and all dependent LOB tablespace parallely as shown below.

```
REORG TABLESPACE MALLIV.T1KKBLB1 SHRLEVEL CHANGE MAPPINGTABLE
T - AUXILIARY TABLESPACE MALLIV.TRKXXLB1 WILL BE REORGANIZED IN PARALLEL WITH
T - AUXILIARY TABLESPACE MALLIV.TPKXXLB1 WILL BE REORGANIZED IN PARALLEL WITH
UTILITY PERFORMS DYNAMIC ALLOCATION OF SORT DISK SPACE
UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=0 FOR TABLESPACE
UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=0 FOR TABLESPACE
UNLOAD PHASE STATISTICS - NUMBER OF RECORDS UNLOADED=0 FOR TABLESPACE
UNLOAD PHASE COMPLETE, ELAPSED TIME=00:00:00
- (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=0 FOR TABLE VX$B5B1.SVC_IMG
- (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=0 FOR TABLE VX$B5B1.SVC_IMG
- (RE)LOAD PHASE STATISTICS - NUMBER OF RECORDS=0 FOR TABLE VX$B5B1.SVC_IMG
(RE)LOAD PHASE STATISTICS - NUMBER OF INPUT RECORDS PROCESSED=0
(RE)LOAD PHASE COMPLETE, ELAPSED TIME=00:00:01
LOG PHASE STATISTICS. NUMBER OF ITERATIONS = 1, NUMBER OF LOG RECORDS = 0
LOG PHASE COMPLETE, ELAPSED TIME = 00:00:01
```

- For segmented tablespace online reorg, if AUX YES keyword is specified, it will be ignored, since there will be no significant advantage of using keyword AUX functionality for segmented tablespace.

```
REORG TABLESPACE MALLIV.T1GKB030 SHRLEVEL CHANGE MAPPINGTABLE
- AUX YES SPECIFICATION IS IGNORED AND AUX NO IS IN EFFECT FOR CURRENT UTILITY
```

- Furthermore, it can be specified only when using SHRLEVEL REFERENCE / CHANGE. So we can say SHRLEVEL NONE and AUX (YES/NO) keywords are mutually exclusive. If we try to specify both, then job with abend with RC 08.

```
- REORG TABLESPACE MALLIV.T1GKB030 SHRLEVEL NONE AUX NO KEYWORD OR OPERAND 'AUX' INVALID WITH 'SHRLEVEL NONE'
- UTILITY EXECUTION TERMINATED, HIGHEST RETURN CODE=8

- REORG TABLESPACE MALLIV.T1GKB030 SHRLEVEL NONE AUX YES KEYWORD OR OPERAND 'AUX' INVALID WITH 'SHRLEVEL NONE'
- UTILITY EXECUTION TERMINATED, HIGHEST RETURN CODE=8
```
Default value for AUX Keyword

If the AUX keyword is omitted, in most of the cases AUX NO is default. But in some cases, AUX YES is default as follows:

1. **REORG TABLESPACE** of a partition-by-growth base table space with one or more LOB columns, where the table space has MAXPARTITIONS value that is greater than one.

2. **REORG TABLESPACE SHRLEVEL REFERENCE REBALANCE** of a partitioned base table space with one or more LOB columns.

3. **REORG TABLESPACE** is against directory table space SPT01, and SPT01 is in the REORP or AREOR state.

4. **REORG TABLESPACE** of a partitioned base table space with one or more LOB columns where one or more partition ranges are in REORG pending state because an ALTER TABLE PARTITION command has been issued to change the partition key boundaries.

5. **REORG TABLESPACE DISCARD** of a table in a partitioned table space with one or more LOB columns.

Execution phases of **REORG TABLESPACE** on an LOB table space

```
 UTILINIT
  ↓
REORGLOB
  ↓
SWITCH
  ↓
UTILTERM
```
**UTILINIT**
Performs initialization and setup.

**REORGLOB**
For SHRLEVEL REFERENCE, the utility unloads LOBs to a shadow data set. If any error is encountered during this phase, it will leave the original data set intact, RECOVER-pending is not set on the LOB table space. You cannot restart REORG TABLESPACE on a LOB tablespace in the REORGLOB phase.

**SWITCH**
Switches access to shadow copy of table space or partition.

**UTILTERM**
Performs cleanup.

For an LOB table space, REORG TABLESPACE performs these actions:

- Removes embedded free space
- Attempts to make LOB pages contiguous

**Note:**
- If you observe above REORG LOB phases, there is no SORT and BUILD phases. The reason is same as we discussed in LOAD utility.
- There are so many Reorg keywords which is not valid for REORG LOB, some of them shown below.

  - STATISTICS (no inline stats in LOB REORG)
  - NEAROFFPOSF
  - FAROFFPOSF
  - UNLOAD
  - SAMPLE
  - PUNCHDDN
  - SORTDEVT
  - PART
2.4 Copy – Quiesce – Recovery

We have to take copy for base tablespace and LOB tablespace separately. There is no option to take the copies together in one dataset. Below are the sample statements for taking image copies for Base and LOB tablespaces

```
//DSNUPROC.SYSCOPY DD DSN=VX$B5B1.MALLIV.T1GBK030.COPY1,
//     DISP=(MOD,CATLG),
//     SPACE=(16384,(20,20),,ROUND),
//     UNIT=SYSDA
//DSNUPROC.SYSCOPY DD DSN=VX$B5B1.MALLIV.T1GBK030.COPY1,
//     DISP=(MOD,CATLG),
//     SPACE=(16384,(20,20),,ROUND),
//     UNIT=SYSDA
//DSNUPROC.SYSIN     DD *
//COPY TABLESPACE MALLIV.T1GBK030
```

Now recover the base table to last FULL IMAGE COPY after some DML activities.

```
//UTIL EXEC DSNUPROC,SYSTEM=Z1U2,UID='TEMP',UTPROC=''
//DSNUPROC.SYSCOPY DD *
//RECOVER TABLESPACE MALLIV.T1GBK030 TOLASTFULLCOPY
```

The job ends with RC 04 as AUX CHECK Pending status is set on base table as shown in below snapshot. Since Recovery did not include LOB tablespace. LOB part of records are not recovered.

```
12:03 DSNUCARS - INDEX VX$B5B1.T1GBK030 IS IN REBUILD PENDING
12:03 DSNUCARS - ALL INDEXES OF MALLIV.T1GBK030 ARE IN REBUILD PENDING
12:03 DSNUCATM - TABLE SPACE MALLIV.T1GBK030 IS IN AUX CHECK PENDING STATE
DSNUCDBR - RECOVERY COMPLETE, ELAPSED TIME=00:00:01
DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=4
```

AUX CHECK Pending status will not be removed even if you run Recovery on LOB Tablespace. Since this is not a common point of recovery, DB2 cannot ensure data integrity.
Why data may not be consistent

Copy for base table space may run fast when compared to LOB tablespaces, if we run COPY utility on base and LOB tablespaces one by one. Some changes may have be included in auxiliary table image copy and which is not included in base table image copy or vice-verse.

CONCURRENT copies:

• If we use CONCURRENT option in copy utility, you might be able to gain improved availability by using the concurrent copy function of the DFSMS.

• If we use CONCURRENT option, it is recommended to take a Quiesce Write Yes before the copy. Since copy is taken by storage sub-system, the data present in bufferpool may not be available in the copy, otherwise.

Recommended process for image copy recovery

1. Start the DB2 objects that are being backed up for read-only access by issuing the command:
   
   -START DB(database-name) SPACENAM(table space name) ACCESS(RO)

2. Run QUIESCE with the WRITE(YES) option to Quiesce all DB2 objects that are being backed up.

3. Run concurrent image copies on base and LOB tablespaces with SHRLEVEL REFERENCE Option

4. Issue the following command to allow transactions to access the data
   
   -START DATABASE(database-name) SPACENAM(table space name)

How to take common recovery point? QUIESCE

Below statements show how to take common recoverable point (Quiesce point) for both LOB and non-LOB data.

```
//DSNUPROC.SYSIN   DD *
QUIESCE TABLESPACESET MALIY.T1GKB030
```
QUIESCE job sysprint shows log point for recovery.

Below statements shows how to recover to the log point.

```sql
//UTIL EXEC DSNUPROC,SYSTEM=Z1U2,UID='TEMP',UTPROC='' 
//DSNUPROC.SYSIN DD *
LISTDEF MYLIST INCLUDE TABLESPACES TABLESPACE MALLIV.T1GBK030
INCLUDE TABLESPACES TABLESPACE MALLIV.T1GKX030
RECOVER LIST MYLIST
TOLOGPOINT X'6DF0B902631'
PARALLEL
```

2.5 Check Data

**AUX CHECK Pending status**

When the auxiliary check-pending status is set on a base table space, that base table space is unavailable for DML's. We will get AUX CHECK Pending status on base tablespace when we run Recover Base tablespace or recover LOB tablespace or both to prior point (Not to current point) which are not having common recovery point. If base tablespace is in AUX CHECK Pending, it means DB2 needs to check data consistency through CHECK DATA Utility

```sql
SELECT TRNSM_SRS,ENG_MDL_DESC,LD_IND,
ENG_RPM,FRQ_CD,SW_TO_IND,FILE_NM,BLOB_COL_ID,
FILE_OBJ,FILE_DESC
FROM VX$SB5B1.GVRN_FILE
;
DSN408I SQLCODE = -904, ERROR: UNSUCCESSFUL EXECUTION CAUSED BY AN UNAVAILABLE RESOURCE. REASON 00C900C5, TYPE OF RESOURCE 00000200, AND RESOURCE NAME MALLIV .T1GBK030
```

CHECK DATA can be run in two modes:

- SHRLEVEL REFERENCE
- SHRLEVEL CHANGE
CHECK DATA SHRLEVEL CHANGE never sets or resets the ACHKP or AUXW states. We can run CHECK DATA on Base table only, not on LOB tablespace.

- Check data will fail in following cases
  - LOB table space is in check pending (CHKP) status
  - Base or LOB tablespace are in Recovery pending (RECP) status
  - Auxiliary index is in the rebuild pending (RBDP) status.

- **Check Data SCOPE AUXONLY**
  Check data will also check data consistency regarding referential and check constraints. If we specify SCOPE AUXONLY, then LOB & XML data consistency only will be checked (Default is ALL).

```
+-----------------------+-
| -SCOPE-- PENDING------+-
| +-----------------------+
| ' -SCOPE-- +-AUXONLY---+-'
| +-ALL- ---------------+
| +-REFONLY- -----------+
| ' -XMLSCHEMAONLY-'    +
```

- **Exception Tables will not work here**
  Exception tables can only be used to remove rows with referential integrity or table constraint errors, not for rows with “bad” LOBs.

- **Auxiliary warning state**
  - The base table space remains available for applications, but not LOB.
  - We can perform any sql on base table with out including LOB Column name.
  - We may get this state after execution of CHECK DATA with AUXERROR INVALIDATE option.
What check data will report?

Check data will report following kind of LOB errors

✓ Missing LOBs
✓ Orphan LOBs
✓ Out-of-sync LOBs
✓ Invalid LOBs

Missing LOBs

LOB referenced by the base table space, but the LOB entry is not in the auxiliary index. We can get missing LOB’s when recovering base table space to a point prior to the DELETE operation OR recovering LOB table space to a point prior to the INSERT Operation OR recovering the base table space to the point when LOB column is not in the base table.

Orphan LOBs

An orphan LOB column is an LOB entry found in the auxiliary index but not referenced by the base table space. An orphan can result if you recover the base table space to a point in time prior to the INSERT operation OR recover LOB table space to a point prior to the DELETE Operation.

Out-of-sync LOBs

An out-of-sync LOB error occurs when DB2 detects an LOB that is found in both the base table and the auxiliary index. But the LOB entry in the auxiliary index is at a different version. An out-of-sync LOB can occur recovering base table space or LOB table space prior to the UPDATE Operation.

Invalid LOBs

An invalid LOB is an uncorrected LOB error set on missing LOB’s by the execution of CHECK DATA AUXERROR INVALIDATE, which will set Auxiliary Warning state on base table.
Lets have some test scenarios to understand different types of LOB errors

**Test 1 (Consistent data test):**

Take Image copies for base tablespace & LOB tablespace one by one and ensure that there is no DML activities during execution of COPY jobs. After some time, recover both tablespaces. Then the base tablespace will be kept in ACHKP. Even if data is consistent here, but how will DB2 ensure that.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSNT360I -Z1U2</td>
<td>DISPLAY DATABASE SUMMARY</td>
</tr>
<tr>
<td>DSNT361I -Z1U2</td>
<td>DISPLAY DATABASE SUMMARY - GLOBAL</td>
</tr>
<tr>
<td>DSNT362I -Z1U2</td>
<td>DATABASE = MALLIV STATUS = RW DBD LENGTH = 28256</td>
</tr>
<tr>
<td>DSNT397I -Z1U2</td>
<td>NAME TYPE PART STATUS PHYERRLO PHYERRHI CATALOG PIECE</td>
</tr>
<tr>
<td><strong>T1GBK030</strong></td>
<td>TS RW,ACHKP</td>
</tr>
<tr>
<td><strong>DISPLAY OF DATABASE MALLIV ENDED</strong></td>
<td></td>
</tr>
<tr>
<td>DSN9022I -Z1U2</td>
<td>DSNTDDIS 'DISPLAY DATABASE' NORMAL COMPLETION</td>
</tr>
</tbody>
</table>

DB2 can ensure consistency when we run check data.

```
//DSNUPROC.SYSPIN DD *
CHECK DATA TABLESPACE MALLIV.T1GBK030
SCOPE AUXONLY
```

See Job Sysprint. If job executes with RC 00, then only we can say data is consistent. Since data is consistent, ACHKP status has been reset on base tablespace.
Test 2 (Missing LOB error test / Invalid LOB error test):

Take Image copies for base tablespace & LOB tablespace one by one and delete some records from base table (let's consider below example)

```
DELETE FROM IDB4SADM.LOBTEST
WHERE ID='1593';
```

DSNE615I NUMBER OF ROWS AFFECTED IS 1
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0

Now recover base tablespace to last full image copy, and rebuild dependent indexes.

```
//DSNUPROC.SYSIN     DD *
RECOVER TABLESPACE KINSSD.LOBTEST TOLASTFULLCOPY
//
INDEX IDB4SADM.LOB_TESTA_H_KN2 IS IN REBUILD PENDING
ALL INDEXES OF KINSSD.LOBTEST ARE IN REBUILD PENDING
TABLE SPACE KINSSD.LOBTEST IS IN AUX CHECK PENDING STATE
```

Base tablespace will be set to Aux check pending status. Now run CHECK DATA. Aux check pending status will not be removed. And it will show information about inconsistent data in sysprint.

```
DSNU809I 174 07:43:52.92 DSNUKERK -
TABLE=IDB4SADM.LOBTEST
COLUMN=ARGUMENTTYPE IS MISSING IN INDEX IDB4SADM.LOBTESTARGUMND03Z9F
ROWID=X'AE29A8446B70B4E81401E7AF6001000000000000201'
VERSION=X'0001'
```

```
DSNU749I 174 08:04:04 DSNUK001 - CHECK DATA COMPLETE. ELAPSED TIME=00:00:0
DSNU010I 174 08:04:05 DSNUGBAC - UTILITY EXECUTION COMPLETE. HIGHEST RETURN CODE=4
```

Even if we forcibly remove the ACHKP using the START DATABASE <database_name> ACCESS (FORCE), we can perform SQL on the consistent data only (base table data), we can’t touch the inconstant data (LOB column). It will throw error as shown below, which is not recommended.
**Reason:**
The deleted records are recovered in base table, but corresponding LOB data was not recovered. So there are some records in base table without having LOB references to LOB table. These records are called Missing LOB’s.

**Resolution:**
We can resolve the issue in 2 ways
1. Providing LOB’s for missing LOB records (which can be done by recovery utility on LOB tablespace)
2. Deleting Missing LOB records in base table using SQL.

**How delete SQL can be performed on object in ACHKP status?**
There is an option called AUXERROR REPORT / INVALIDATE in CHECK DATA. If you run CHECK DATA AUXERR INVALIDATE, this utility will change auxiliary check pending status to auxiliary warning state. If there are any missing LOB’s then the base table LOB column is set to an invalid status. Now we can perform SQL on base table without including LOB column.

```
//DSNPROC.SYIN DD *
CHECK DATA TABLESPACE KINSSD.LOBTEST
SCOPE AUXONLY
AUXERROR INVALIDATE
```

Sysprint of above check data job shown below

```sql
DSNU809I 174 08:04:03.93 DSNUKERK - TABLE=IDB4SADM.LOBTEST COLUMN=ARGUMENTTYPE IS MISSING IN INDEX
IDB4SADM.LOBTEARGUMNDO0Z9F
  ROWID=x'AE29A8446B70B643E81401E7AF600100000000021001'
  VERSION=x'0001'
DSNU739I 174 08:04:03.93 DSNUKDAT - CHECK TABLE IDB4SADM.LOBTEST COMPLETE,
ELAPSED TIME=00:00:00
DSNU806I /IDB4 174 08:04:03.98 DSNUKRON - TABLE=IDB4SADM.LOBTEST COLUMN=
  =ARGUMENTTYPE WAS SET INVALID
  ROWID=x'AE29A8446B70B643E81401E7AF600100000000021001'
  VERSION=x'0001'
DSNU810I /IDB4 174 08:04:04.04 DSNUGRSRX - TABLESPACE KINSSD.LOBTEST IS IN AUX WARNING STATE
DSNU749I 174 08:04:04.04 DSNUK001 - CHECK DATA COMPLETE,ELAPSED TIME=00:00:0
DSNU0010I 174 08:04:04.05 DSNUGBAC - UTILITY EXECUTION COMPLETE. HIGHEST RETURN CODE=4
```
What SQL to perform to delete the inconsistent data?

If we know SQL to delete missing LOB records, then we can execute that SQL as shown below. Since this is our test case we know the SQL. But in real-time how to handle this scenario.

```sql
DELETE FROM IDB4SADM.LOBTEST
WHERE ID = '1593';
```

DSNE615I NUMBER OF ROWS AFFECTED IS 1
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0

From CHECK DATA sysprint, we can get rowid's of missing LOB's, we can delete missing LOB records using that rowid's.

```sql
DELETE FROM IDB4SADM.LOBTEST
WHERE ROWID =
ROWID(X'AE29A8446B70B4C4E81401E7AF6001000000000000201');
```

DSNE615I NUMBER OF ROWS AFFECTED IS 1
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0

Finally run Check data which will check data consistency and it will remove ACHKP / AUXW.

Test 3 (Orphan LOB error test):

Take Image copies for base tablespace & Lob tablespace one by one and delete some records from base table (let us consider below example)

```sql
DELETE FROM IDB4SADM.LOBTEST
WHERE ID = '1593';
```

DSNE615I NUMBER OF ROWS AFFECTED IS 1
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0

Now recover LOB tablespace to last full image copy, and rebuild dependent indexes. Base tablespace will be set to Aux check pending status. Now run CHECK DATA. Aux check pending status will be removed. But there will be inconsistent data in LOB tablespace as shown in sysprint.
Reason:
The deleted records are recovered in LOB table, but corresponding data records was not recovered in base tablespace. There are some LOB’s in LOB table which are not referred by base table. These LOB’s are called Orphan LOB’s.

Effect:
These Orphan LOB’s are cause lot of storage wastage. Theoretically, there is no performance issue due to Orphan LOBs.

Resolution:
We can resolve the issue in 2 ways
1. Providing non-LOB part for missing LOB records (which can be done by recovery utility on base tablespace)
2. Deleting Orphan LOB records using repair utility.

2.6 Repair
CHECK DATA cannot delete bad LOBs. Using Repair utility we can delete or dump bad LOBs (orphan LOB’s, out-of-sync LOBs). When you specify LOCATE ROWID and VERSION for a LOB table space, with the DELETE option, the entire LOB specified is deleted with its index entry. All pages occupied by that LOB are converted to free space.
Instead of DELETE keyword we can use DUMP keyword in above job, which will produces a hexadecimal dump of LOB data.

2.7 DSN1COPY

The PARM field LOB must be specified. Below is the sample DSN1COPY:

```plaintext
//COPY0 EXEC PGM=DSN1COPY,COND=(4,LT),
//    PARM='FULLCOPY,LOB,OBJIDXLAT,RESET'
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DSN=DB2BU.IDB1COP1.GEMS5D.GEMSSFXN.D1604,
//    DISP=SHR
//SYSUT2 DD DSN=MAPMSSU.DSNDBD.GEMS5D.GEMSSFXN.I0001.A001
//    DISP=OLD
//SYSXLAT DD *
317,293
385,276
387,278
/*
//
```
3. LOB System Fundamentals

3.1 Database commands

We can perform START DB and STOP DB commands on a LOB tablespaces.

Even if we stop LOB tablespace, we can access Non-LOB columns of base table.

In display database output, LS indicates LOB tablespace.
3.2 Page format

LOB table spaces have a completely different format compared to other table spaces. Because an LOB entry in an LOB data can span pages, LOB data pages have to be chunked together. A (full) chunk is referred to as 16 pages of contiguous space acquired in a LOB table space.

Depending on your size of data, a certain number of full chunks and partial chunk are allocated for a single LOB value.

For example, if a LOB occupies 36 Pages, 2 chunks plus 4 page contiguous space is provided for that LOB. This 4 page contiguous space is called partial chunk.
LOB Page set structure

<table>
<thead>
<tr>
<th>Page</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Page 0</td>
<td>4 Pages</td>
</tr>
<tr>
<td>4 Pages</td>
<td>16 Pages</td>
</tr>
<tr>
<td>16 Pages</td>
<td>3072 Pages</td>
</tr>
<tr>
<td>3072 Pages</td>
<td>16 Pages</td>
</tr>
</tbody>
</table>

Header Page
The header page contains control information used by DB2 to manage and maintain the table space. For example, the PSID, OBID and DBID of the table space, LOGGED or NOT LOGGED attributes, etc..

High Level Space Map (HLSM) pages, Low Level Space Map (LLSM) pages
LOB space map pages are structured like a multi-level index, which is an index containing several levels of non leaf pages and leaf pages are LOB map pages.

**LOB Map Pages**

- The chunk information itself is stored in LOB map pages. The LOB map entries point to the page number where LOB data exists for corresponding row ID and version ID, as well as information about the length of the LOB data.
- There will be at least one LOB map page for every version of every single LOB value in your LOB table space.
- LOB map page may also contains LOB data after LOB map information, as shown below.

![Diagram of LOB Map Page]

**LOB data pages**

- The LOB data page contains the actual LOB data.
• One page will have one LOB. But one lob can occupy many pages.

• We should not think that 32KB bufferpool (page size) is always better for LOB. It will depend upon LOB size.

• For example, If LOB size is around 33 KB to 35 KB and page size is 32 KB, then for every LOB there will wastage of space 31 KB to 29 KB.

• Recommended Pagesize for LOB’s based on LOB size are shown below

<table>
<thead>
<tr>
<th>LOB size (y)</th>
<th>Suggested page size</th>
</tr>
</thead>
<tbody>
<tr>
<td>y ≤ 4 KB</td>
<td>4 KB</td>
</tr>
<tr>
<td>4 KB &lt; y ≤ 8 KB</td>
<td>8 KB</td>
</tr>
<tr>
<td>8 KB &lt; y ≤ 12 KB</td>
<td>4 KB</td>
</tr>
<tr>
<td>12 KB &lt; y ≤ 16 KB</td>
<td>16 KB</td>
</tr>
<tr>
<td>16 KB &lt; y ≤ 24 KB</td>
<td>8 KB</td>
</tr>
<tr>
<td>24 KB &lt; y ≤ 32 KB</td>
<td>32 KB</td>
</tr>
<tr>
<td>32 KB &lt; y ≤ 48 KB</td>
<td>16 KB</td>
</tr>
<tr>
<td>48 KB &lt; y</td>
<td>32 KB</td>
</tr>
</tbody>
</table>
3.3 Catalog Information

• **SYSIBM.SYSTABLESPACE** (TYPE: O → LOB Tablespace)

• **SYSIBM.SYSTABLES** (TYPE: X → AUX Table)

• **SYSIBM.SYSAUXRELS**

This table contains one row for each auxiliary table created for a LOB column.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBOWNER</td>
<td>The schema of the base table</td>
</tr>
<tr>
<td>TBNAME</td>
<td>Base table name</td>
</tr>
<tr>
<td>COLNAME</td>
<td>LOB column name</td>
</tr>
<tr>
<td>PARTITION</td>
<td>Partition number if the base table space is partitioned. Otherwise, the value is 0</td>
</tr>
<tr>
<td>AUXTBOWNER</td>
<td>The schema of the auxiliary table.</td>
</tr>
<tr>
<td>AUXTBNAME</td>
<td>Name of the auxiliary table.</td>
</tr>
<tr>
<td>AUXRELOBID</td>
<td>Internal identifier of the relationship between the base table and the auxiliary table.</td>
</tr>
</tbody>
</table>

• **SYSIBM.SYSLOBSTATS**, This table will have LOB statistics

<table>
<thead>
<tr>
<th>Column name</th>
<th>Column description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATSTIME</td>
<td>Timestamp when last invocation of RUNSTATS</td>
</tr>
<tr>
<td>AVGSIZE</td>
<td>Average size of a LOB, measured in bytes</td>
</tr>
<tr>
<td>FREESPACE</td>
<td>Number of kilobytes of available space</td>
</tr>
<tr>
<td>ORGRATIO</td>
<td>The percentage of organization in the LOB table space. A value of '100' indicates perfect organization.</td>
</tr>
<tr>
<td>DBNAME</td>
<td>Database name that contains the LOB table space.</td>
</tr>
<tr>
<td>NAME</td>
<td>Name of the LOB table space.</td>
</tr>
</tbody>
</table>

• **SYSIBM.SYSCOLUMNS**

(AUXID, AUXVER, AUXVALUE are fixed column names for Auxiliary table)
INLINE_LENGTH column of SYSIBM.SYSDATATYPES

If there is no INLINE LENGTH then value is -1. Value greater than or equal to zero refers inline length for LOB column.

TTYPE column in SYSIBM.SYSCOPY

For Alter INLINE LENGTH activity, column ICTYPE=A (ALTER) and column STYPE=I (inline length)

TTYPE=I (increase)       REORG INCREASE INLINE LENGTH
TTYPE=D (decrease)       REORG DECREASE INLINE LENGTH

CHECKFLAG column of SYSIBM.SYSTABLEPART

D - inline length of LOB column was decremented with an alter
I – inline length of LOB column was incremented with an alter

LOGGED column in SYSIBM.SYSCOPY

Y - to indicate the logging attribute is LOGGED.
N - to indicate the logging attribute is NOT LOGGED
Blank - to indicate that the row was inserted prior to DB2 V9

The possible combination are

<table>
<thead>
<tr>
<th>Action</th>
<th>LOGGED</th>
<th>ICTYPE</th>
<th>STYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE TABLESPACE</td>
<td>Y</td>
<td>C</td>
<td>L</td>
</tr>
<tr>
<td>CREATE TABLESPACE</td>
<td>N</td>
<td>C</td>
<td>O</td>
</tr>
<tr>
<td>ALTER TABLESPACE</td>
<td>Y</td>
<td>A</td>
<td>L</td>
</tr>
<tr>
<td>ALTER TABLESPACE</td>
<td>N</td>
<td>A</td>
<td>O</td>
</tr>
</tbody>
</table>

ICTYPE :  
C → CREATE
A → ALTER

STYPE :  
L → Logging attribute of the table space was altered to LOGGED
O → Logging attribute of the table space was altered to NOT LOGGED
Some Important queries on catalog regarding LOB

Below query will filter all Base tables which contains LOB columns (Whether base
table contains LOB columns are not, we cannot get directly from base table).

```
SELECT DISTINCT TBOWNER, TBNAME
FROM SYSIBM.SYSAUXRELS
WITH UR;
```

Below query is used to get all base table which contain LOB columns in particular
database (Here intersection with SYSTABLES is required since there is no
DBNAME column in syauxrels).

```
SELECT CREATOR, NAME
FROM SYSIBM.SYSTABLES
WHERE DBNAME = 'MALLIV'
AND TYPE = 'T'
INTERSECT
SELECT DISTINCT TBOWNER, TBNAME
FROM SYSIBM.SYSAUXRELS
WITH UR;
```

3.4 Bufferpools

LOB data is not recommended to share bufferpools with base table data. It
is better to use separate set of Bufferpools and different thresholds to get
performance.

If you use same set of bufferpools for both LOB and Base tablespace, it
will cause performance issues, since bufferpools may cache more LOB data and
may not be able to cache more relational data.
3.5 Logging

When logging LOBs will cause lot of issues like wasting log capacity, using more log datasets, and using more CPU. Before DB2 V9 even if LOG YES is mentioned logging was disabled for LOBs exceeding 1 GB. In DB2 V9 two major changes happened.

- LOGGED / NOT LOGGED was introduced instead of LOG YES / NO
- Logging for all LOB sizes upto 2GB-1 byte

LOGGED and NOT LOGGED attributes

In DB2 V9, the CREATE TABLESPACE syntax is changed from LOG YES/NO to LOGGED/NOT LOGGED. The old syntax is still supported for LOB tables spaces for compatibility reasons.
**LOGGED**

- Specifying LOGGED for an LOB table space tells DB2 to log almost all data manipulations on LOB columns, except **DELETE** operations. When you delete an LOB value, no LOB data is written to the log, only LOB system pages are written to the log data sets, because an LOB delete internally is translated into only deallocation of the pages where the LOB is stored (which is referred as old versions). The space occupied by old versions can be reused once you have taken a backup (image copy).
- When you **INSERT** a LOB, DB2 writes the entire LOB value to the log.
- When you **UPDATE** a LOB, it will work like a DELETE plus INSERT operations. Log activity also works as DELETE plus INSERT.
- LOGGED is the default value when you create your LOB table space.

**NOT LOGGED**

- To prevent your system from the possible overhead caused by logging large amounts of LOB data, DB2 allows you to turn off logging.
- For LOB tablespaces, header pages, space map pages, and the new LOB map pages are logged even though NOT LOGGED is specified (so recovery and rollback is possible for even NOT LOGGED table space).
- NOT LOGGED LOB values are written into disk at COMMIT.

**Logging combination of base table space and LOB table spaces**

- It is a requirement that if the base table space has the NOT LOGGED logging attribute, all associated LOB table spaces must also have the NOT LOGGED logging attribute. Even if you put LOGGED it will overwrite to NOT LOGGED.
- If the base table space has the LOGGED attribute, the logging attribute of the LOB table space continues to be independent of the base table space. In this case, the LOB table space can have either a LOGGED or a NOT LOGGED.
Note:
COPY SHRLEVEL CHANGE / REORG SHRLEVEL CHANGE is not allowed for a table space that is defined as NOT LOGGED unless it is an LOB table space

Altering LOGGED - NOT LOGGED

• When a base table space logging attribute is altered from LOGGED to NOT LOGGED, all associated LOB table spaces with the LOGGED logging attribute are also implicitly altered to force their logging attribute to NOT LOGGED to match the base table space.
• When the base table space has a NOT LOGGED attribute, the LOB table space attribute should not be altered to LOGGED (DB2 issues SQLCODE -763, SQLSTATE 560A1)

DB2 rollback for LOB tablespace without undo log records

• Because new pages are allocated while inserting an LOB, they are simply deallocated if DB2 does a rollback.
• If an LOB is deleted (pages are deallocated), they are simply reallocated if DB2 does a rollback.
• This mechanism is known as Shadow Copy Recovery.
• Because of allocation and deallocation of data pages, no UNDO log records are written for LOBs, regardless of which LOG parameter you use.
• How will this work for NOT LOGGED? Reason is Header pages, space map pages, and the new LOB map pages are logged even though NOT LOGGED is specified.
3.6 Locking
Locking techniques for LOB data is not exactly same as non-LOB data, there are some differences.

What is LOB Lock
A single LOB value can span over a lot of pages, so in order to ensure data integrity a new lock is introduced which can lock an LOB value in an LOB tablespace, called an LOB lock. Choosing LOCKSIZE ANY for LOB tablespace implies the use of LOCKSIZE LOB.

Types of LOB Locks
There are 2 types of LOB locks

- Shared LOB lock (S-LOB lock)
- Exclusive LOB lock (X-LOB lock)

Shared LOB lock (S-LOB lock)
The lock owner and any concurrent processes can read, update, or delete the locked LOB. Concurrent processes can acquire an S lock on the LOB.

Exclusive LOB lock (X-LOB lock)
The lock owner can read or change the locked LOB. Concurrent processes cannot access the LOB.

There is no U – LOB lock
DB2 uses two types of locks for ensuring an LOB’s integrity: the S-LOB and the X-LOB locks. They are very similar to the common S- and X-Locks. There are no U-LOB locks since here update is equivalent to delete-insert operation.

LOB locking techniques
LOB locking is significantly changed in V9; therefore, the LOB’s locking information before DB2 V9 and from DB2 V9 are mentioned in 2 separate sections.
LOB Locking Pre-V9

– Insert: X LOB lock, held until commit
– Delete: S LOB lock, held until commit
– Select: S LOB lock, held until commit
– Even for ISO(UR)

LOB Locking from V9 onwards

– Insert: X LOB lock, release after insert
– Delete & normal select: No LOB lock
– Select with ISO(UR): S LOB lock, released immediately.

Lock escalation

If LOCKSIZE is LOB and number of LOB locks reached LOCKMAX then lock is escalated to table space
3.7 Sub-system parameters

<table>
<thead>
<tr>
<th>Parm</th>
<th>Macro</th>
<th>Panel</th>
<th>Acceptable values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOB_INLINE_LENGTH</td>
<td>DSN6SYSP</td>
<td>DSNTIPD</td>
<td>0 to 32680</td>
<td>0</td>
</tr>
<tr>
<td>LOBVALA</td>
<td>DSN6SYSP</td>
<td>DSNTIPD</td>
<td>1 to 2097152</td>
<td>10240</td>
</tr>
<tr>
<td>LOBVALS</td>
<td>DSN6SYSP</td>
<td>DSNTIPD</td>
<td>1 to 51200</td>
<td>2048</td>
</tr>
<tr>
<td>MAXOFILR</td>
<td>DSN6SYSP</td>
<td>DSNTIPD</td>
<td>0 - value in MAX USERS field</td>
<td>100</td>
</tr>
<tr>
<td>CHECK_SETCHKP</td>
<td>DSN6SYSP</td>
<td>DSNTIPD</td>
<td>YES / NO</td>
<td>NO</td>
</tr>
</tbody>
</table>

LOB_INLINE_LENGTH
The LOB_INLINE_LENGTH subsystem parameter specifies the default length (in bytes) that is to be used for inline LOB columns.

USER LOB VALUE STG field (LOBVALA)
The LOBVALA subsystem parameter controls the maximum amount of storage, in KB, that each user is to have for storing LOB values.

SYSTEM LOB VAL STG field (LOBVALS)
The LOBVALS subsystem parameter sets an upper limit, in MB, for the amount of memory per system that is to be used for storing LOB values.

MAX OPEN FILE REFS field (MAXOFILR)
The MAXOFILR subsystem parameter specifies the maximum number of data sets that can be opened concurrently for the processing of LOB file references.

SET CHECK PENDING field (CHECK_SETCHKP)
Specifies whether the CHECK DATA and CHECK LOB utilities are to place inconsistent objects in CHECK PENDING status. If it is NO means do not place
objects in CHECK PENDING status if an inconsistency is detected. If the object was already in CHECK PENDING status, it remains in that status.

**Note:** Except LOB_INLINE_LENGTH remaining all zparm parameters will apply in DB2 V9, DB2 V10 and DB2 V11. LOB_INLINE_LENGTH is applicable from V10.

## 3.8 LOB – System level Significance

As we know all the catalog information is stored in directory in internal format. How hundreds of catalog tables data are fitted into few directory tables. There are two reasons behind this, one reason is that the data is in internal format. Another reason it that most of the data is stored in LOB tablespaces. If we see the below directory tables we know that most of the DB2 data objects information is stored in DBD01 directory table. Actually here the data is stored in its auxiliary table SYSDBD_DATA.

<table>
<thead>
<tr>
<th>Description</th>
<th>Tablespace (DSNDB01.xx)</th>
<th>Table (SYSIBM.XXX)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Descriptor Table</td>
<td>DBD01</td>
<td>DBDR</td>
</tr>
<tr>
<td>Skeleton Package Table</td>
<td>SPT01</td>
<td>SPTR</td>
</tr>
<tr>
<td>Skeleton Cursor Table</td>
<td>SCT02</td>
<td>SCTR</td>
</tr>
<tr>
<td>Log range Table</td>
<td>SYSLGRNX</td>
<td>SYSLGRNX</td>
</tr>
<tr>
<td>Utility job processing Table</td>
<td>SYSUTILX</td>
<td>SYSUTIL</td>
</tr>
<tr>
<td>SYSUTIL Extended Table</td>
<td></td>
<td>SYSUTILX</td>
</tr>
<tr>
<td>Auxiliary Table for DBDR</td>
<td>SYSDBDXA</td>
<td>SYSDBD_DATA</td>
</tr>
<tr>
<td>Auxiliary Table for SPTR</td>
<td>SYSSPUXA</td>
<td>SYSSPTSEC_DATA</td>
</tr>
<tr>
<td>Auxiliary Table for Static packages in SPTR</td>
<td>SYSSPUXB</td>
<td>SYSSPTSEC_EXPL</td>
</tr>
</tbody>
</table>

© Copyright IBM Corporation, 2013
Improvements in DB2 for LOB from version to version.

**DB2 8**
- ZPARMS to limit storage allocation for LOB(LOBVALA,LOBVALS)
- Auto generation of ROWID column for base table
- XMB2CLOB function convert XML value into a CLOB

**DB2 9**
- Implicit DDL for LOBs
- LOB lock avoidance
- Faster preformatting
- LOB APPEND
- FETCH CONTINUE
- REORG SHRLEVEL REFERENCE
- CHECK LOB SHRLEVEL CHANGE
- CHECK DATA SHRLEVEL CHANGE
- MAXOFILR ZPARAM
- Load/Unload FRV performance
- SQL support for FRV (File reference variables)
- Below diagram shows performance improvement for UNLOAD-FRV in V8 & V9 versions.
DB2 10

- Inline LOBs
- Alter AUX tablespace and index page size
- DEFINE NO on AUX tablespace and index
- REORG SHRLEVEL CHANGE
- REORG AUX YES
- Faster LOAD of FRV
- LOAD/UNLOAD support for RECFM=VBS
- LOB materialization avoidance
- Faster LOAD REPLACE from V9 to V10

DB2 11

- Cross loader support for FETCH CONTINUE (28% CPU reduction)
Bibliography

Websites:

• IBM Knowledge center is good source for almost all DB2 related concepts.
  
  https://www.ibm.com/support/knowledgecenter/search/

• toadworld.com is one of the sites where we can get knowledge about IBM DB2
  

Manuals:

• “LOBs with DB2 for z/OS: Stronger and Faster”, Red book about DB2 Z/OS - LOB's
  

• IDUG presentation about Inline LOB's
  
  http://www.neodbug.org/201208/InLineLOBS.pdf

• “ Large Objects in DB2 for z/OS “ by Francis Desiron
  

• “DB2 for z/OS LOBs Experiences & Best Practices” by Haakon Roberts
  

© Copyright IBM Corporation, 2013