Handout - DB2 LOB's

1. Large Objects (LOBs) - IBM DB2 LUW

The term large object and the generic acronym LOB refer to the BLOB, CLOB, or DBCLOB data type. LOB values are subject to restrictions that apply to LONG VARCHAR values. These restrictions apply even if the length attribute of the LOB string is 254 bytes or less.

LOB values can be very large, and the transfer of these values from the database server to client application program host variables can be time consuming. Because application programs typically process LOB values one piece at a time, rather than as a whole, applications can reference a LOB value by using a large object locator.

A large object locator, or LOB locator, is a host variable whose value represents a single LOB value on the database server.

An application program can select a LOB value into a LOB locator. Then, using the LOB locator, the application program can request database operations on the LOB value (such as applying the scalar functions SUBSTR, CONCAT, VALUE, or LENGTH; performing an assignment; searching the LOB with LIKE or POSSTR; or applying user-defined functions against the LOB) by supplying the locator value as input. The resulting output (data assigned to a client host variable) would typically be a small subset of the input LOB value.

LOB locators can represent more than just base values; they can also represent the value associated with a LOB expression. For example, a LOB locator might represent the value associated with:
SUBSTR(<lob 1> CONCAT <lob 2> CONCAT <lob 3>, <start>, <length>)

When a null value is selected into a normal host variable, the indicator variable is set to -1, signifying that the value is null. In the case of LOB locators, however, the meaning of indicator variables is slightly different. Because a locator host variable can itself never be null, a negative indicator variable value indicates that the LOB value represented by the LOB locator is null. The null information is kept local to the client by virtue of the indicator variable value — the server does not track null values with valid locators.

It is important to understand that a LOB locator represents a value, not a row or a location in the database. Once a value is selected into a locator, there is no operation that one can perform on the original row or table that will affect the value which is referenced by the locator. The value associated with a locator is valid until the transaction ends, or until the locator is explicitly freed, whichever comes first. Locators do not force extra copies of the data to provide this function. Instead, the locator mechanism stores a description of the base LOB value. The materialization of the LOB value (or expression, as shown above) is deferred until it is actually assigned to some location — either a user buffer in the form of a host variable, or another record in the database. A LOB locator is only a mechanism used to refer to a LOB value during a transaction; it does not persist beyond the transaction in which it was created. It is not a database type; it is never stored in the database and, as a result, cannot participate in views or check constraints. However, because a LOB locator is a client representation of a LOB type, there are SQLTYPEs for LOB locators so that they can be described within an SQLDA structure used by FETCH, OPEN, or EXECUTE statements.

2. Large object table spaces

Large object (LOB) table spaces (also known as auxiliary table spaces) hold large object data, such as graphics, video, or large text strings. If your data does not fit entirely within a data page, you can define one or more columns as LOB columns.

LOB objects can do more than store large object data. If you define your LOB columns for infrequently accessed data, a table space scan on the remaining data in the base table is potentially faster because the scan generally includes fewer pages.

A LOB table space always has a direct relationship with the table space that contains the logical LOB column values. The table space that contains the table with the LOB columns is, in this context, the base table space. LOB data is logically associated with the base table, but it is physically stored in an auxiliary table that resides in a LOB table space. Only one auxiliary table can exist in a large object table space. A LOB value can span several pages. However, only one LOB value is stored per page.

You must have a LOB table space for each LOB column that exists in a table. For example, if your table has LOB columns for both resumes and photographs, you need one LOB table space (and one auxiliary table) for each of those columns. If the base table space is a partitioned table space, you need one LOB table space for each LOB in each partition.

If the base table space is not a partitioned table space, each LOB table space is associated with one LOB column in the base table. If the base table space is a partitioned table space, each partition of the base table space is associated with a LOB table space. Therefore, if the base table space is a partitioned table space, you can store more LOB data for each LOB column.

The following table shows the approximate amount of LOB data that you can store for a LOB column in each of the different types of base table spaces.

Table 1. Base table space types and approximate maximum size of LOB data for a LOB column

Base table space type	Maximum (approximate) LOB data for each column
Segmented	16 TB
Partitioned, with NUMPARTS up to 64	1000 TB
Partitioned with DSSIZE, NUMPARTS up to 254	4000 TB
Partitioned with DSSIZE, NUMPARTS up to 4096	64000 TB

Recommendations:

- Consider defining long string columns as LOB columns when a row does not fit in a 32 KB page. Use the following guidelines to determine if a LOB column is a good choice:
  - Defining a long string column as a LOB column might be better if the following conditions are true:
    - Table space scans are normally run on the table.
    - The long string column is not referenced often.
    - Removing the long string column from the base table is likely to improve the performance of table space scans.
  - LOBs are physically stored in another table space. Therefore, performance for inserting, updating, and retrieving long strings might be better for non-LOB strings than for LOB strings.

3. Creation of large objects

Defining large objects to DB2® is different than defining other types of data and objects.

These are the basic steps for defining LOBs and moving the data into DB2:

1. Define a column of the appropriate LOB type.

   When you create a table with a LOB column, or alter a table to add a LOB column, 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 LOB column holds information about the LOB, not the LOB data itself. The table that contains the LOB information is called the base table, which is different from the common base table. DB2 uses the ROWID column to locate your LOB data. You can define the LOB column and the ROWID column in a CREATE TABLE or ALTER TABLE statement. If you are adding a LOB column and a ROWID column to an existing table, you must use two ALTER TABLE statements. If you add the ROWID after you add the LOB column, the table has two ROWIDs; a hidden one and the one that you created. DB2 ensures that the values of the two ROWIDs are always the same.

2. Create a table space and table to hold the LOB data.

   For LOB data, the table space is called a LOB table space, and a table is called an auxiliary table. If your base table is nonpartitioned, you must create one LOB table space and one auxiliary table for each LOB column. If your base table is partitioned, you must create one LOB table space and one auxiliary table for each LOB column in each partition. For example, you must create three LOB table spaces and three auxiliary tables for each LOB column if your base table has three partitions. Create these objects by using the CREATE LOB TABLESPACE and CREATE AUXILIARY TABLE statements.

3. Create an index on the auxiliary table.

   Each auxiliary table must have exactly one index in which each index entry refers to a LOB. Use the CREATE INDEX statement for this task.

4. Put the LOB data into DB2.

   If the total length of a LOB column and the base table row is less than 32 KB, you can use the LOAD utility to put the data in DB2. Otherwise, you must use one of the SQL statements that change data. Even though the data resides in the auxiliary table, the LOAD utility statement or SQL statement that changes data specifies the base table. Using INSERT or MERGE statements can be difficult because your application needs enough storage to hold the entire value that goes into the LOB column.

Example: Assume that you must define a LOB table space and an auxiliary table to hold employee resumes. You must also define an index on the auxiliary table. You must define the LOB table space in the same database as the associated base table. Assume that EMP_PHOTO_RESUME is a base table. This base table has a LOB column named EMP_RESUME. You can use statements like this to define the LOB table space, the auxiliary table space, and the index:

```
CREATE LOB TABLESPACE RESUMETS
  IN MYDB
  LOG NO;
COMMIT;
CREATE AUXILIARY TABLE EMP_RESUME_TAB
  IN MYDB.RESUMETS
  STORES EMP_PHOTO_RESUME
  COLUMN EMP_RESUME;
```

```
CREATE UNIQUE INDEX XEMP_RESUME
  ON EMP_RESUME_TAB;
COMMIT;
```