HOWTO - Work with LOB Data Types

by Torsten Schlabach

1. Using Oracle LOB Data Types with OJB

1.1. Introduction

In a lot of applications there is a need to store large text or binary objects into the database. The definition of large usually means that the object's size is beyond the maximum length of a character field. In Oracle this means the objects to be stored can grow to > 4 KB each.

Depending on the application you are developing your "large objects" may either be in the range of some Kilobytes (for example when storing text-only E-Mails or regular XML documents), but they may also extend to several Megabytes (thinking of any binary data such as larger graphics, PDFs, videos, etc.).

In practice, the interface between your application and the database used for fetching and storing of your "large objects" needs to be different depending on the expected size. While it is probably perfectly acceptable to handle XML documents or E-Mails in memory as a string and always completely retrieve or store them in one chunk this will hardly be a good idea for video files for example.

This HOWTO will explain:

1. Why you would want to store large objects in the database
2. Oracle LARGE versus LOB data types
3. LOB handling in OJB using JDBC LOB types

This tutorial presumes you are familiar with the basics of OJB.

2. Backgrounder: Large objects in databases

This section is meant to fill in non-DBA people on some of the topics you need to understand when discussing large objects in databases.

2.1. Your database: The most expensive file system?

Depending on background some people tend to store anything in a database while others are biased against that. As databases use a file system for physical storage anyway, why would it make sense to store pictures, videos and the like as a large object in a database rather that just create a set of folders and store them right into the database.

When listening to Oracle's marketing campaingns one might get the impression that there is no need to have plain filesystems anymore and that they all will vanish and be replaced by Oracle database servers. If that happened this would definitely boast Oracle's revenues, but at the same time make IT cost in companies explode.

But there are applications where it in fact makes sense to have the database take care of unstructured data that you would otherwise just store in a file. The most common criteria for storing non-relational data in the database instead of storing it directly into the file system is whenever there is a strong link between this non-relatinal and some relational data.

Typical examples for that would be:

1. Pictures or videos of houses in a real estate agent's offer database

2. E-Mails related to a customer's order

If you are not storing these objects into the database you would need to create a link between the relational and the non-relational data by saving filenames in the database. This means that you application is responsible for managing this weak link in any respect. In order to make sure your application will be robust you need to make sure in your own code that

1. When creating a new record you create a valid and unique filename for storing the object.
2. When deleting a record you delete the corresponding file as well
3. When accessing the file referred to in the record you double-check the file is there and no locked

(There might be other, more subtle implications.)

All this is done for you by the database in case you choose to store your objects there. In addition to that, when discussing text data, a database might come with an option to automatically index the stored text documents for easy retrievel. This would allow you to perform an SQL seach such as "give me all customers that ever referred to the project foo in an e-mail". (In Oracle you need to install the InterMedia option, aka Oracle Text in order to get this extra functionality. Several vendors have also worked on technologies that allowed to seach rich content such as PDFs files, pictures or even sound or video stored in a database from SQL.)

2.2. Oracle LARGE versus LOB datatypes

Some people are worried about the efficiency of storing large objects in databases and the implications on performance. They are not necessarily entirely wrong in fearing that storing large objects in databases might be problematic the best or might require a lot of tweaks to parameters in order to be able to handle the large objects. It all depends on how a database implements storing large objects.

Oracle comes with two completely different mechanisms for that:

1. LARGE objects
2. LOB objects

When comparing the LARGE datatypes such as (*fixme*) to the LOB datatypes such as CLOB, BLOB, NCLOB (*fixme*) they don't read that different at first. But there is a huge difference in how they are handled both internally inside the database as well when storing and retrieving from the database client.

LARGE fields are embedded directly into the table row. This has some consequences you should be aware of:

1. If your record is made up of 5 VARCHAR fields with a maximum length of 40 bytes each and one LONGVARCHAR and you store 10 MB into the LONGVARCHAR column, your database row will extent to 10.000.200 bytes or roughly 10 MB.
2. The database always reads or writes the entire row. So if you do a SELECT on the VARCHAR fields in order to display their content in a user interface as a basis for the user to decide if he or she will need the content of the LONGVARCHAR at all the database will already have fetched all the 10 MB. If you SELECT and display 25 records each with a 10 MB object in it this will mean about 250 MB of I/O.
3. When storing or fetching such a row you need to make sure your fetch buffer is sized appropriately.

In practice this cannot be efficient. It might work as long as you stay in the KB range, but you will most likely run into trouble as soon as it gets into the MBs per record. Additionally, there are more limitations to the concept of LONG datatypes such as limiting the number of them you can have in one row and how you can index them. This is probably why Oracle decided to deprecate LONG datatypes in favor of LOB columns.

A lot of non-Oracle-DBA people believe that LOB means "large OBject" because some other vendors have used the term BLOB for "Binary Large OBject" in their products. This is not only wrong but - even worse - misleading, because people are asking: "What's the difference between large and long?" (Bear with all non native English speakers here, please!)

Instead, LOB stands for Locator OBject which exactly describes what is is. It is a pointer to the place where the actual data itself is stored. This locator will need only occupy some bytes in the row thus not harming row size at all. So all the issues discussed above vanish immediatelly. For the sake of simplicity think of a LOB as a pointer to a file on the databases internal file system that stores the actual content of that field. (Oracle might use plain files or different mechanisms in their

implementation, we don't have to care.)

But as there is always a trade-off while LOBs are exstremely handy inside a row, they are more complex to store and retrieve. As opposed to all other column types their actual content stays where it is even if you transfer the row from the database to the client. All that goes over the wire in that case will be a token representing the actual LOB column content.

In order to read the content or to write LOB content it needs to open a separate stream connection over the network that can be read from or written to similar to a file on a network file system. JDBC (starting at version *fixme*) comes with special objects such as java.sql.Blob and java.sql.Clob to access the content of LOBs that do not represent character arrays or strings but streams!

3. Large Objects in OJB

After having skipped the above Backgrounder (in case you do Oracle administration for a living) of having read and understood it (hopefully applies to the rest of us) now that you've most likely decided to go for LOBs and forget about LONGs how is this handled with OJB?

3.1. Strategy 1: Using streams for LOB I/O

########## to be written #########

3.2. Strategy 2: Embedding OJB content in Java objects

########## to be written #########

3.3. Querying CLOB content

########## to be written #########