LOB – Das Unbekannte Wesen

Dierk Lenz
Herrmann & Lenz Services GmbH
Burscheid

Schlüsselworte
Oracle Datenbanken, Large Objects, CLOB, BLOB

Einleitung

Large Objects (LOBs) gibt es seit Oracle8i - also eigentlich schon immer. Sie sollten den Datentyp LONG ablösen. (Was sie bis heute nicht geschafft haben, insbesondere im Oracle Data Dictionary.) Die Verwendung von LOBs ist mittlerweile in vielen Datenmodellen angekommen. Meist werden sie als Ablageort für "Alles, was groß ist" angesehen und ansonsten nicht weiter beachtet. Außer vielleicht vom DBA, der irgendwann feststellt, dass die Datenbank quasi unkontrolliert wächst, und vom Anwender, der sich beschwert, dass die Anwendung sich irgendwie zäh anfühlt. Im Vortrag geht es um einige Dinge, die einem das Leben mit LOBs einfacher machen können. DBAs, Datenmodellierer und Entwickler sind angesprochen.

Warum LOBs?

Bevor mit Oracle8i LOBs eingeführt wurden, hatte man mit LONG (bzw. LONG RAW) bereits einen Datentyp für „längere“ Inhalte. LONG hatte einige sehr restriktive Eigenschaften: Pro Tabelle darf es nur eine LONG-Spalte geben. Der komplette LONG-Inhalt wird zusammen mit den anderen Attributen im gleichen Block gespeichert; eventuell als *chained row*.

Auch LOBs haben gegenüber den skalaren Datentypen einige Einschränkungen, aber man kann z.B. mehrere LOBs pro Tabelle definieren. Die LOB-Inhalte werden in eigenen LOB-Segmenten gespeichert, so dass die Tabellen selbst kompakter werden. Das macht Zugriffe auf die Tabellen, bei denen die LOBs nicht betrachtet werden, erheblich schneller.

Hier werden nun einige wichtige Eigenschaften von LOBs behandelt. Dabei werden hauptsächlich die Datentypen BLOB (Binary LOB), CLOB (Character LOB) und NCLOB (National Character LOB) betrachtet; BFILE (extern gespeicherte LOBs) werden nicht betrachtet.

Implementierung von LOBs

Die wichtigste Idee bei der Implementierung von LOBs ist die Trennung der LOB-Daten von den – im Vergleich dazu kurzen – anderen Attributdaten der Tabelle.[1] Damit wir eine ganze Palette von Möglichkeiten eröffnet: LOBs können in einem eigenen Tablespace abgelegt werden, der z.B. eine optimierte Blockgröße haben kann. Hierdurch können LOB-Blöcke in einen eigenen Cache-Bereich gelegt werden, so dass sie die Tabellen- und Indexblöcke nicht behindern. Das manchmal unkontrolliert erscheinende Wachstum von LOBs kann in einem eigenen Tablespace besser kontrolliert werden.

---

[1] Eine Ausnahme: Wird für ein LOB-Feld die Option ENABLE STORAGE IN ROW gesetzt, so werden kleinere LOB-Inhalte bei den anderen Attributen gespeichert.

Zum hohen Speicherplatzverbrauch von LOBs trägt sicher die Tatsache bei, dass die Speicherung anderen Mechanismen gehorcht als “normale” Datensätze. In einem LOB-Segment werden NIE Inhalte verschiedener LOBs in einem Block gemischt. Die Rege ist hier, dass LOBs in sog. Chunks gleicher Größe abgelegt werden; die minimale Größe eines Chunks ist die Blockgröße.

Im Datensatz wird für den LOB lediglich ein sog. Locator gespeichert. Dieser ist lediglich ein Verweis auf die LOB-Daten im LOB-Segment. Neben dem LOB-Segment für die LOB-Daten ist ein zusätzlicher LOB-Index vorhanden, der die Speicherung der LOBs abbildet.

BasicFile LOBs und SecureFile LOBs

Die erste Implementierung von LOBs erfolgte wie bereits erwähnt mit Oracle8i. Viele der heute in der Datenbank vorhandenen Features wie Komprimierung und Verschlüsselung waren zu dem Zeitpunkt noch nicht vorhanden. Mit Oracle 11g wurden LOBs praktisch komplett neu implementiert. Die neue LOB-Variante wurde SecureFile LOBs getauft, die alte im Nachhinein BasicFile LOBs.

Nur für SecureFile LOBs wurden Komprimierung und Verschlüsselung ermöglicht. Nun hört man oft das folgende Argument: „Ich habe keine Enterprise Edtion bzw. nicht die Advanced Compression Option bzw. die Advanced Security Option – dann kann ich ja bei BasicFile LOBs bleiben.“ Das ist prinzipiell richtig, allerdings haben SecureFile LOBs noch einen weiteren Vorteil: die Performance. Generell haben SecureFile LOBs erhebliche Performance-Vorteile!

LOB Caching

Als LOBs erstmalig implementiert wurden, waren Hauptspeicher (und demnach SGAs) erheblich kleiner als heute. Damit ist klar, dass LOBs so implementiert wurden, dass sie im Cache kein großes Unheil anrichten konnten. Im Standard ist für LOBs NOCACHE eingestellt, d.h. werde beim Lesen noch beim Schreiben wird der Buffer Cache genutzt. Alle logischen Lese- und Schreiboperationen werden synchron physisch ausgeführt. Wenn man in den System-Events einer Datenbank hohe Werte für „direct path read“ bzw. „direct path write“ sieht, kann man sehr gut auf LOB-Aktivitäten mit NOCACHE-LOBs wetten.

Stellt man einen LOB auf CACHE ein, so ändert sich das Zeitverhalten dramatisch. Es gibt eine weitere Variante CACHE READS, bei der lesende Vorgänge im Cache landen, schreibende nicht. Hierbei ist zu beachten, dass je nach der verarbeiteten Datenmenge von LOBs der Buffer Cache stark beansprucht wird. D.h. CACHE ist um so mehr zu empfehlen, je größer der Buffer Cache ist. Zusätzlich kann hier auch mit speziellen Caches gearbeitet werden, z.B. wenn der LOB-Tablespace eine Nicht-Standard-Blockgröße hat.

Das nicht vorhandene LOB-Caching ist auch beim Export und insbesondere beim Import für den schlechten Ruf von LOBs verantwortlich.

CLOBs und Unicode

Viele Datenbanken werden aktuell in einem Unicode-Zeichensatz in UTF-8-Kodierung angelegt.[2] Dieser ist z.B. für deutsche Texte und Namen sehr gut bezüglich des Platzbedarfs: ASCII-Zeichen belegen lediglich ein Byte, Umlaute zwei Bytes und das €-Symbol drei Bytes. Da Umlaute und €-Symbole typischerweise nur einen kleinen Anteil an Texten haben, macht sich die UTF-8-Kodierung im Gegensatz zu einer Single-Byte-Kodierung kaum bemerkbar.

[2] Oracle-Zeichensatz AL32UTF8

Allerdings gilt diese Feststellung ausschließlich für die Datentypen CHAR und VARCHAR2. CLOBs werden in Unicode-Datenbanken grundsätzlich in einem UCS2-kompatiblen Format gespeichert. Dieses Format ist jedoch im Gegensatz zur UTF-8-Kodierung nicht variabel lang sondern hat eine feste Länge von zwei Bytes pro Zeichen.

Damit haben CLOB-Felder in Unicode-Datenbanken einen erhöhten Platzbedarf – die Feldinhalte benötigen exakt doppelt so viel Speicher wie in der Single-Byte-Kodierung. Zudem enthalten aktuelle UTF-8-Kodierungen erheblich mehr Zeichen als eine Zwei-Byte-Kodierung fassen kann. Die neuesten Emojis sind also in CLOBs nicht verfügbar.

Große VARCHAR2-Felder

Mit Oracle 12c ist es möglich, die Beschränkung von 4000 Bytes pro VARCHAR2-Feld auf 32 KB anzuheben. Dazu muss der Datenbankparameter max_string_size von STANDARD auf EXTENDED umgestellt werden.[3]

Dies betrifft ja VARCHAR2-Felder – also warum kommt das in einem Vortrag über LOBs vor? Ganz einfach: Sobald der Inhalt eines VARCHAR2-Feldes dann tatsächlich größer als 4000 Bytes ist, wird die LOB-Technologie zum Speichern verwendet. Somit bekommt man über 32 KB-Felder so etwas wie „kleine LOBs“ in die Datenbank.

Die Erwähnung der großen VARCHAR2-Felder kommt allerdings nicht zufällig direkt hinter der Unicode-Problematik bei den CLOBs. Die großen VARCHAR2-Felder sind allerdings grundsätzlich im Datenbankzeichensatz kodiert, also z.B. mit AL32UTF-Zeichensatz in der „platzsparenden“ UTF-8-Kodierung.

Programmierung

Zugriffe auf LOBs sind mit SQL möglich; einige Programmierumgebungen bieten zusätzlich spezielle APIs an. In PL/SQL ist das Package DBMS_LOB verfügbar. Das Oracle Call Interface (OCI), Oracle C++ Call Interface (OCCI) sowie die PRO*-Compiler besitzen dedizierte LOB APIs. Auch der Oracle Data Provider for .NET bietet LOB-spezifische Klassen an.

Die APIs zum Zugriff auf LOBs können größtenteils genauso benutzt werden, wie „normale“ String-APIs. Das scheint den meisten Programmierern zu reichen.

Wenn man allerdings mit sehr großen LOBs zu tun hat und nicht immer komplette LOB-Inhalte lesen oder schreiben möchte, sondern nur Teile des LOBs, dann sollte man sich die speziellen LOB-APIs (s.a. Locator Interface) ansehen. Diese sind oft zunächst mit mehr Aufwand verbunden, bieten jedoch ein hohes Maß an Flexibilität sowie Performance. Dies wird z.B. mit Prefetching im Oracle Call Interface erreicht.

Fazit

LOBs werden in der Oracle Datenbank anders behandelt als „normale“ Daten. Das heißt zunächst, dass man sich der Konsequenzen bewusst werden sollte. Einfacher und schneller ist i.a. die Verwendung der „kleineren“ Datentypen, insbesondere VARCHAR2. Setzt man LOBs ein, so sollte

---

[3] Dies ist für eine Datenbank eine nicht-reversible Aktion: einmal umgestellt kann man STANDARD nicht mehr einstellen. Neben der Einstellung des Parameters ist die Ausführung des Skripts $ORACLE_HOME/rdbms/admin/utl32k.sql notwendig.

man die Speicherung sowie einige Parameter (insbesondere LOB Caching) prüfen, um unter dem Strich eine gut funktionierende Lösung zu haben.

Kontaktadresse:
Dierk Lenz
Herrmann & Lenz Services GmbH
Am Ziegelfeld 28
51399 Burscheid

Telefon: +49 (2174) 30710 - 11
E-Mail dierk.lenz@hl-services.de
Internet: www.hl-services.de