Datenbanken II

Speicherung und Verarbeitung großer Objekte (Large Objects)

Jörg Kohlsdorf

05IND-T

Hochschule für Technik, Wirtschaft und Kultur (FH) Leipzig

Fachbereich Informatik, Mathematik und Naturwissenschaften

27. Juni 2008

Inhaltsverzeichnis

Listings

1 Einführung

1.1 Was sind Large Objects?

Large Objects (LOBs) sind eine Klasse von Datentypen, die dafür geschaffen wurden, große Datenmengen aufzunehmen. So können sie je nach Konfiguration des Datenbanksystems zwischen 1 und 128 Terabyte an Daten aufnehmen.
Daten in Large Objects zu speichern ermöglicht effiziente Zugriffe sowie Handhabung dieser innerhalb von Anwendungen.
Large Objects unterliegen zudem dem Transaktionsprinzip von Datenbanken (**A**tomarität, **C**onsistenz, **I**ntegrität, **D**auerhaftigkeit). Im Fehlerfall (Absturz des DBMS oder ähnliches) kann ein ROLLBACK auf diesen durchgeführt werden um einen gültigen Zustand wiederherzustellen.

1.2 Warum sollten LOBs genutzt werden?

Kapazität

Die Kapazität der Datentypen LONG und LONG RAW ist in Oracle auf 2GB limitiert.

Anzahl LOB-Spalten pro Tabelle

In Oracle kann eine Datenbanktabelle lediglich eine LONG- bzw. LONG RAW-Spalte enthalten, sie kann jedoch mehrere LOB-Spalten enthalten

wahlfreier Zugriff

Die LONG Datentypen unterstützen lediglich sequentiellen Zugriff. Um Daten an einer bestimmten, bekannten Position innerhalb eines solchen Datensatzes zu lesen ist es deshalb nötig, den gesamten Datensatz sequentiell zu lesen. Large Object Datentypen bieten im Gegensatz dazu Unterstützung für wahlfreien Zugriff.

Operationen

Für die Large Objects wurden eine Reihe von Operationen innerhalb des DBMS implementiert, so dass diese nicht in der Anwendung selbst ausgeführt werden müssen.

Tablespace

Die LOB-Spalten können jeweils in einen separaten Tablespace (dies kann auch ein anderer physikalischen Speicherort sein) ausgegliedert werden.

1.3 LOBs im SQL-99 (SQL3) Standard

Im nachfolgenden wird der SQL-99 Standard betrachtet da dieser der neueste ist, der fast vollständig von den bekanntesten Datenbanksystemen unterstützt wird. Die neueren Standards SQL:2003 und SQL:2006 werden teilweise von den Datenbanksystemen unterstützt, wobei SQL:2006 bisher nur zu sehr geringen Teilen implementiert wurde. Es muss jedoch angemerkt werden, dass sich in diesen neuen Standards kaum etwas an den LOB-Datentypen geändert hat.

Der SQL-99 Standard definiert zwei LOB Datentypen: Den *BLOB* Typ sowie den *CLOB* Typ.

BLOB - Binary Large Object

Geeignet für binäre Daten wie Bilder, Audio, Video.

CLOB - Character Large Object

Geeignet für Textdaten.

Es gilt zu beachten, dass der Standard keine Vorgaben für die Verarbeitung (Speichern, Füllen, Lesen) der Large Object Datentypen macht. Dies kann deshalb in jedem Datenbanksystem anders sein.

1.3.1 Deklaration

Deklaration: **<Attributname> BLOB | CLOB [(<Länge>)]**

Wobei die Länge die maximale Größe des Inhalts der Large Object Spalte in Bytes angibt. Zur Vereinfachung kann optional **K**, **M** oder **G** für die Längenangabe benutzt werden:

```
1 CREATE TABLE Buecher (
2 BUCH_ID INTEGER,
3 TITEL VARCHAR(255),
4 ZUSAMMENFASSUNG CLOB(64K),
5 EBOOK_TEXT CLOB(50M),
6 FILM BLOB(5G));
```

Listing 1: *Anlegen einer Tabelle mit LOB-Spalten*

1.3.2 Benutzung der LOBs

Large Objects werden abgerufen, eingefügt und aktualisiert wie jeder andere Datentyp. Es müssen dazu in der Applikation Puffer allokiert werden, die groß genug sind um den Inhalt der LOBs zu fassen:

```
1 EXEC SQL
2         SELECT ZUSAMMENFASSUNG, EBOOK_TEXT, FILM
3         INTO :puffer, :grosserpuffer, :riesigerpuffer
4         FROM Buecher
5         WHERE TITEL="SQL-99 Standard";
```

Listing 2: *SELECT von LOB-Spalten beispielhaft mit Embedded SQL*

1.3.3 Operationen

Der Standard definiert folgende Operationen für den Umgang mit LOB-Datentypen:

- Verkettung - **string1 || string2**
- **SUBSTRING(string FROM anfang FOR laenge)**
- **LENGTH(ausdruck)**
- **POSITION(nadel IN heuhaufen)**
- **NULLIF**
- **COALESCE**
- **TRIM**
- **OVERLAY**
- **BIT_LENGTH**, **CHAR_LENGTH**, **OCTET_LENGTH**
- das **LIKE**-Prädikat
- Casting
- Userdefinierte Funktionen

1.3.4 Lokatoren

SQL-99 stellt sogenannte Lokatoren bereit um den Zugriff auf LOBs handhabbar zu machen. Ein solcher Lokator ist ein 4-Byte Wert welcher in einer Variable gespeichert wird, die ein Programm nutzen kann um damit auf den aktuellen Inhalt eines LOBs zu referenzieren. Ein Lokator darf überall genutzt werden, wo der Inhalt eines LOBs genutzt werden kann.

```
1  EXEC SQL BEGIN DECLARES SECTION:
2           SQL TYPE IS BLOB_LOCATOR
3                   ebooklokator;
4  EXEC SQL END DECLARE SECTION;
5
6  EXEC SQL
7           SELECT EBOOK_TEXT
8           INTO :ebooklokator
9           FROM Buecher
10          WHERE TITEL="SQL-99 Standard";
```

Listing 3: *Benutzung von Lokatoren beispielhaft mit Embedded SQL*

HOLD LOCATOR

Behält den Inhalt eines LOBs nach COMMIT einer Transaktion.

```
1 EXEC SQL
2         SELECT ZUSAMMENFASSUNG
3         INTO :zusammenfassungslokator
4         FROM Buecher
5         WHERE TITEL="SQL-99 Standard";
6
7 HOLD LOCATOR :zusammenfassungslokator;
8
9 COMMIT;
```

Listing 4: *Benutzung des HOLD Lokators beispielhaft mit Embedded SQL*

FREE LOCATOR

Gibt einen Lokator und den Inhalt des zugeordneten LOBs frei.

```
1 EXEC SQL
2         INSERT INTO
3                 VALUES (...,:lokator,...);
4
5 FREE LOCATOR :lokator;
```

Listing 5: *Benutzung von FREE LOCATOR beispielhaft mit Embedded SQL*

1.4 Einschränkungen bei der Verwendung von LOBs

Large Object dürfen nicht Teil eines *Index* oder *Primärschlüssels* sein. Ebenfalls dürfen sie nicht in der *GROUP BY* oder *ORDER BY* - Klausel bzw. in Aggregationen vorkommen.

2 Large Objects in Oracle 10g Release 2

In Oracle werden 2 Typen von Large Objects unterschieden:

Interne LOB-Typen

- Standardtypen aus SQL-99: **CLOB**, **BLOB**
- **NCLOB** (National Character LOB)

Externe LOB-Typen

- BFILE (Binary File)

2.1 Interne LOBs

Bei den internen Large Objects werden die Inhalte der LOBs im Tablespace der Datenbank gespeichert. Ist die Länge des Inhalts kleiner als 3964 Bytes, so kann der Datensatz innerhalb der Tabelle gespeichert werden. Ansonsten enthält der Lokator einen Verweis auf den physischen Speicherort.
Für die Verarbeitung von LOB-Werten bietet Oracle die beiden Datentypen **TEMPORARY BLOB** und **TEMPORARY CLOB** sowie weitere Funktionen, zusätzlich zu den im SQL-99 Standard geforderten an.
Die Initialisierung von LOBs erfolgt in Oracle mit **EMPTY_BLOB()** bzw. **EMPTY_CLOB()**, die Spalte enthält dann einen Lokator auf einen leeren LOB der Länge Null. Eine Ausnahme ist das Setzen eines LOBs auf NULL, in der Spalte wird dann, wie bei allen anderen Datentypen auch, NULL statt eines Lokators gespeichert.

```
1 CREATE TABLE LOBTable (
2         key NUMBER,
3         image BLOB DEFAULT EMPTY_BLOB(),
4         imageFile BFILE,
5         text CLOB DEFAULT EMPTY_CLOB());
```

Listing 6: *Definition einer Tabelle mit verschiedenen LOB-Datentypen in Oracle 10g*

2.2 Externe LOBs

Externe LOBs sind eine Oracle-spezifische Erweiterung um Large Objects außerhalb des Datenbanksystems zu speichern. Die Lokatoren verweisen bei diesen Typen auf Betriebssystemdateien (dies kann zum Beispiel auch eine DVD sein) außerhalb der Datenbank. Deshalb gilt für diesen LOB-Typ das Transaktionsprinzip nicht!
BFILE-LOBs sind READ ONLY, es kann lediglich der Lokator geändert werden um auf eine andere Betriebssystemdatei zu zeigen bzw. NULL zu enthalten. Beim Löschen einer Zeile in der Datenbank, die ein BFILE-LOB enthält, bleibt die zugeordnete Betriebssystemdatei erhalten.
In der Datenbank muss für den Zugriff auf die Betriebssystemdateien als BFILE-LOBs ein virtuelles Verzeichnis angelegt werden welches auf den physikalischen Speicherort der Dateien verweist. Dieses Verzeichnis und dessen Inhalt muss für das Datenbanksystem lesbar sein.

```
1 CREATE DIRECTORY IMAGES_DIR AS 'C:\Bilder';
```

Listing 7: *Anlegen eines virtuellen Verzeichnises für die Benutzung als BFILE-Verzeichnis*

```
1 INSERT INTO LOBTable
2         VALUES (
3                 1,
```

```
4                   EMPTY_BLOB(),
5                   BFILENAME('IMAGES_DIR', bild.gif),
6                   EMPTY_CLOB());
```

Listing 8: *Einfügen eines Beispieldatensatzes mit BFILE-LOB*

3 Tuningansätze für LOBs in Oracle

Von Oracle werden folgende, grundlegende Tipps zur Optimierung von LOB-Zugriffen gegeben:

Freigabe nicht mehr benötigter temporärer LOBs

Temporäre LOBs die nicht weiter benötigt werden, sollten von der Applikation so früh wie möglich freigegeben werden. Diese bleiben ansonsten bis zur Terminierung der Datenverbindung im Speicher. Durch die mögliche Datengröße ist dies bei LOBs besonders wichtig.

Tablespace Optimierung

LOBs sollten in einen anderen Tablespace gespeichert werden, als die eigentliche Datenbanktabelle, welche die LOB-Spalte enthält. Dies soll vor Fragmentierung des Tablespace durch die LOB-Daten schützen. Bei mehreren LOB-Spalten in einer Tabelle wird empfohlen mehrere Tablespaces zu nutzen und diese, wenn möglich, über mehrere Speichermedien zu verteilen.

Kleine LOBs

Wenn bekannt ist, daß eine LOB-Spalte nur kleine Datensätze (z.B. < 8KB) enthält, wird dazu geraten, die STORAGE-Klausel so anzupassen, daß diese innerhalb der Datenbanktabelle gespeichert werden.

3.1 Anpassung der Storage Klausel

Von Oracle werden mehrere Optimierungsmöglichkeiten für die STORAGE-Klausel einer Tabelle mit LOB-Spalten vorgeschlagen.

```
1 CREATE TABLE LOBTable (n NUMBER, c CLOB)
2 lob (c) STORE AS SEGNAME (
3         TABLESPACE lobts1 CHUNK x
4         PCTVERSION y/RETENTION
5         CACHE/NOCACHE/CACHE READS LOGGING/NOLOGGING
6         ENABLE/DISABLE STORAGE IN ROW
7         STORAGE (MAXEXTENTS 5));
```

Listing 9: *Einfügen eines Beispieldatensatzes mit BFILE-LOB*

Im Folgenden werden die einzelnen Parameter erläutert:

CHUNK

Größe der Oracle Blocks (maximal 32K, standardmäßig 4K) in denen die Daten gespeichert werden. In großen Chunks ist der Datenzugriff effizienter, werden allerdings auch kleine Daten gespeichert geht so Speicherplatz verloren.

PCTVERSION

Wenn ein LOB verändert wird, wird eine neue Version der LOB Seite angelegt um konsistenten Zugriff auf alte Versionen des LOB Inhalts zu gewähren.
PCTVERSION ist der Prozentsatz des gesamt benutzten LOB-Speicherplatzes, der für alte Versionen benutzt werden kann. Sobald alte Versionen mehr Speicherplatz als PCTVERSION benötigen, versucht Oracle diesen zurückzugewinnen.
Der Standardwert von PCTVERSION ist 10 (%).
Wenn viele Änderungen an den LOB-Daten gemacht werden, wird empfohlen diesen Wert zu erhöhen. Falls Updates kaum bis gar nicht vorkommen, wird empfohlen diesen Wert zu senken. Wenn existierente LOBs nie geändert werden, kann PCTVERSION sogar auf 0 gesetzt werden.

RETENTION

Anstelle von PCTVERSION kann alternativ RETENTION gesetzt werden. Dadurch werden alte Versionen für eine bestimmte Zeitspanne, anstelle von einem Prozentsatz des Tablespaces, gespeichert. Diese Zeitspanne kann über den UNDO_RETENTION Parameter angepasst werden.

CACHE / NOCACHE / CACHE READS

Einstellung des Cache-Verhaltens.

- **CACHE:** Oracle speichert LOB Seiten in den Puffer um schnelleren Zugriff zu ermöglichen. Dies empfiehlt sich, wenn oft gelesen und oft geschrieben wird.
- **NOCACHE:** Dies ist der Standardwert, bei dem LOBs nicht in den Puffer gelassen werden. Dies empfiehlt sich, wenn nie geschrieben und sehr selten gelesen wird.
- **CACHE READS:** LOB Daten werden nur zum lesen, aber nicht beim schreiben in den Puffer geholt. Dies empfiehlt sich, wenn selten geschrieben, aber oft gelesen wird.

Es gilt zu beachten, dass bei Benutzung der CACHE-Option andere Nicht-LOB Seiten möglicherweise vorzeitig aus dem Cache verdrängt werden.

[NO]LOGGING

Wenn CACHE gesetzt ist muss LOGGING ebenfalls gesetzt sein. NOLOGGING ist immer dann sinnvoll, wenn viele Daten eingefügt werden (Bulk Loading)

ENABLE / DISABLE STORAGE IN ROW

Das aktivieren dieser Option veranlasst die Datenbank den Inhalt des LOB-Feldes innerhalb der Spalte zu speichern. Dies funktioniert nur für solche Daten, die klein genug sind (< 4000 Bytes inklusive der nötigen Kontrollinformationen). Sobald die Daten diese Größe überschreiten, werden sie außerhalb der der Datenbankzeile gespeichert wobei die Kontrollinformationen in der LOB-Spalte verbleiben.
Manchmal ist es jedoch besser die Daten außerhalb der Datenbankzeile zu speichern, vor allem da dadurch die Datenbankzeile nicht anwächst. Für Operationen wie Full Table Scans, Range Scans oder viele UPDATEs/SELECTs auf die Nicht-LOB Spalten kann dies zu höherer Performanz führen. Diese Optionen können nur beim Anlegen der Tabelle gesetzt werden und nicht nachträglich geändert werden.

3.2 Indexierung von LOBs

Weitere Performancesteigerungen können durch die Verwendung von Indexen erzielt werden.
Dabei muss jedoch beachtet werden, daß B-Baum Index, funktionsbasierter Index, sowie Bitmap Indexe auf LOB-Spalten nicht möglich sind.

Text Index

Je nach Inhalt des betroffenen LOBs ist es möglich die bekannten Oracle Text Indexe für zum Beispiel CLOBs zu verwenden.

4 Verarbeitung von LOBs mit Oracle 10g Release 2

4.1 API-Zugriffe

LOBs können mit den folgenden APIs verarbeitet werden:

- **PL/SQL** mit dem **DBMS_LOB**-Paket
- **OCI** (Oracle Call Interface) eine Low-Level API für C
- **JDBC**

4.2 Erweiterte LOB-Funktionen

Das **DBMS_LOB**-Paket stellt neben den Funktionen aus dem SQL-99 Standard weitere Funktionen zur Be- und Verarbeitung von LOB-Werten bereit.
Folgende Funktionen dienen zur *Veränderung* von BLOB-, CLOB- und NCLOB-Werten:

- **APPEND()** - hängt einen LOB-Wert an einen anderen
- **COPY()** - kopiert einen LOB-Wert ganz oder teilweise

- **ERASE()** - löscht Teil eines LOBs ab einem bestimmten Offset
- **LOADCLOBFROMFILE()** - lädt Character-Daten aus einer Datei in einen internen LOB
- **LOADBLOBFROMFILE()** - lädt Binär-Daten aus einer Datei in einen internen LOB
- **WRITE()** - schreibt Daten ab einem Offset in ein LOB
- **WRITEAPPEND()** - schreibt Daten an das Ende eines LOBs

Folgende Funktionen dienen zur *Analyse* von internen oder externen LOB-Werten:

- **COMPARE()** - vergleicht die Inhalte zweier LOBs
- **GETCHUNKSIZE()** - ermittelt die Chunkgröße
- **GETLENGTH()** - ermittelt die Länge eines LOBs
- **INSTR()** - ermittelt die Position eines Suchmusters
- **READ()** - liest Daten ab einem gegebenen Offset
- **SUBSTR()** - ermittelt Teil eines LOB-Wertes ab einem Offset

Weitere wichtige Funktionen:

- **OPEN()** - lädt den Lokator eines externen LOB
- **CLOSE()** - schließte Lokator und gibt Ressourcen frei
- **CREATETEMPORARY()** - erzeugt ein temporäres LOB
- **FREETEMPORARY()** - löscht temporäres LOB
- **FILEEXISTS()** - prüft, ob eine Datei sich auf dem Server befindet
- **FILEGETNAME()** - ermittelt Datei- und Verzeichnisname

Quellenverzeichnis

[Bil91] BILIRIS, Alexander: *The EOS Large Object Manager*. April 1991

[Bil92] BILIRIS, Alexander: An Efficient Database Storage Structure for Large Dynamic Objects. In: *Proceedings, IEEE Data Engineering Conference*. Phoenix, Arizona : IEEE Press, 1992, S. 301–308

[Böt03] BÖTTGER, Christina: *Oberseminar Moderne Datenbanken - Objektrelationale Datenbanken*. `http://www.gismo81.de/Studium/download/ordbs.pdf`. Version: November 2003

[DG00] DIEKER, Stefan ; GUTING, Ralf H.: Efficient Handling of Tuples with Embedded Large Objects. In: *Data Knowledge Engineering* 32 (2000), Nr. 3, S. 247–269

[Eis03] EISENTRAUT, Peter: *PostgreSQL: Das offizielle Handbuch*. 1. mitp-Verlag Bonn, 2003. – ISBN 3–8266–1337–6

[Han02] HAN, Ki-Joon: *SQL3 Standardization*. `http://db.konkuk.ac.kr/upload_file/research/2002-03-23%2013:18:06SQL3.pdf`. Version: März 2002

[MP05] MELNICK, Jack ; PAAPANEN, Eric: *Oracle Database Application Developer's Guide - Large Objects 10g Release 2*. Juni 2005