Datenbanken II
Speicherung und Verarbeitung großer Objekte (Large Objects [LOBs])

Jörg Kohlsdorf

Hochschule für Technik, Wirtschaft und Kultur Leipzig

06.06.2008

Inhalt der Präsentation

Was sind Large Objects (LOBs)?

- LOBs sind eine Klasse von Datentypen die dafür konstruiert sind, große Datenmengen aufzunehmen
- Ein LOB kann je nach Konfiguration des Datenbanksystems zwischen 1 und 128 TB an Daten aufnehmen
- Daten in LOBs zu speichern ermöglicht effiziente Zugriffe und Handhabung dieser Daten innerhalb der Applikationen, die diese nutzen
- LOBs unterliegen ebenfalls dem Transaktionsprinzip (ACID) - sie können mit einem ROLLBACK rückgängig gemacht werden

Was sind Large Objects (LOBs)?

- LOBs sind eine Klasse von Datentypen die dafür konstruiert sind, große Datenmengen aufzunehmen
- Ein LOB kann je nach Konfiguration des Datenbanksystems zwischen 1 und 128 TB an Daten aufnehmen
- Daten in LOBs zu speichern ermöglicht effiziente Zugriffe und Handhabung dieser Daten innerhalb der Applikationen, die diese nutzen
- LOBs unterliegen ebenfalls dem Transaktionsprinzip (ACID) - sie können mit einem ROLLBACK rückgängig gemacht werden

Was sind Large Objects (LOBs)?

- LOBs sind eine Klasse von Datentypen die dafür konstruiert sind, große Datenmengen aufzunehmen
- Ein LOB kann je nach Konfiguration des Datenbanksystems zwischen 1 und 128 TB an Daten aufnehmen
- Daten in LOBs zu speichern ermöglicht effiziente Zugriffe und Handhabung dieser Daten innerhalb der Applikationen, die diese nutzen
- LOBs unterliegen ebenfalls dem Transaktionsprinzip (ACID) - sie können mit einem ROLLBACK rückgängig gemacht werden

Was sind Large Objects (LOBs)?

- LOBs sind eine Klasse von Datentypen die dafür konstruiert sind, große Datenmengen aufzunehmen
- Ein LOB kann je nach Konfiguration des Datenbanksystems zwischen 1 und 128 TB an Daten aufnehmen
- Daten in LOBs zu speichern ermöglicht effiziente Zugriffe und Handhabung dieser Daten innerhalb der Applikationen, die diese nutzen
- LOBs unterliegen ebenfalls dem Transaktionsprinzip (ACID) - sie können mit einem ROLLBACK rückgängig gemacht werden

Motivation (Warum Daten nicht in LONGs speichern?)

LOB Kapazität

Kapazität der LONG und LONG RAW Datentypen sind in Oracle auf 2GB limitiert

Anzahl LOB-Spalten pro Tabelle

Eine Tabelle kann mehrere LOB-Spalten haben, aber nur eine LONG- oder LONG RAW-Spalte (Oracle)

wahlfreier Zugriff

LOBs unterstützen wahlfreien Zugriff auf die Daten
LONGs unterstützen nur sequentiellen Zugriff

Motivation (Warum Daten nicht in LONGs speichern?)

LOB Kapazität

Kapazität der LONG und LONG RAW Datentypen sind in Oracle auf 2GB limitiert

Anzahl LOB-Spalten pro Tabelle

Eine Tabelle kann mehrere LOB-Spalten haben, aber nur eine LONG- oder LONG RAW-Spalte (Oracle)

wahlfreier Zugriff

LOBs unterstützen wahlfreien Zugriff auf die Daten
LONGs unterstützen nur sequentiellen Zugriff

Motivation (Warum Daten nicht in LONGs speichern?)

LOB Kapazität

Kapazität der LONG und LONG RAW Datentypen sind in Oracle auf 2GB limitiert

Anzahl LOB-Spalten pro Tabelle

Eine Tabelle kann mehrere LOB-Spalten haben, aber nur eine LONG- oder LONG RAW-Spalte (Oracle)

wahlfreier Zugriff

LOBs unterstützen wahlfreien Zugriff auf die Daten
LONGs unterstützen nur sequentiellen Zugriff

SQL Standard

Der SQL-99 (SQL3) Standard definiert für LOBs:

- Datentypen: **BLOB**, **CLOB**
- Deklaration: **<Attributname> BLOB|CLOB [(<Länge>)]**
- Länge: Länge des LOB-Inhalts in Bytes, optional: K, M, G

Der Standard gibt keine Vorgaben für die Verarbeitung (Speichern, Füllen, Auslesen) von LOBs.

Beispiel

```
CREATE TABLE LOBTable ( key INTEGER, image BLOB (5M),
                        text CLOB(50K) );
```

Operationen

substring(), overlay(), trim(), position(), bit_length(), char_length(), octet_length()

SQL Standard

Der SQL-99 (SQL3) Standard definiert für LOBs:

- Datentypen: **BLOB**, **CLOB**
- Deklaration: **<Attributname> BLOB|CLOB [(<Länge>)]**
- Länge: Länge des LOB-Inhalts in Bytes, optional: K, M, G

Der Standard gibt keine Vorgaben für die Verarbeitung (Speichern, Füllen, Auslesen) von LOBs.

Beispiel

```
CREATE TABLE LOBTable ( key INTEGER, image BLOB (5M),
                        text CLOB(50K) );
```

Operationen

substring(), overlay(), trim(), position(), bit_length(), char_length(), octet_length()

SQL Standard

Der SQL-99 (SQL3) Standard definiert für LOBs:

- Datentypen: **BLOB**, **CLOB**
- Deklaration: **<Attributname> BLOB|CLOB [(<Länge>)]**
- Länge: Länge des LOB-Inhalts in Bytes, optional: K, M, G

Der Standard gibt keine Vorgaben für die Verarbeitung (Speichern, Füllen, Auslesen) von LOBs.

Beispiel

```
CREATE TABLE LOBTable ( key INTEGER, image BLOB (5M),
                        text CLOB(50K) );
```

Operationen

substring(), overlay(), trim(), position(), bit_length(), char_length(), octet_length()

Einschränkungen bei Verwendung von LOBs

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

Einschränkungen bei Verwendung von LOBs

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

Einschränkungen bei Verwendung von LOBs

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

Oracle 10g Release 2

In Oracle 10g werden 2 LOB-Typen unterschieden:

interne LOB-Typen

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

externe LOB-Typen

BFILE (Binary File)

Oracle 10g Release 2

In Oracle 10g werden 2 LOB-Typen unterschieden:

interne LOB-Typen

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

externe LOB-Typen

BFILE (Binary File)

Interne LOBs

- Transaktionskonzept - ACID (ROLLBACK möglich)
- Werte werden im Tablespace der DB gespeichert
- Länge < 3964 Bytes, so kann Datensatz in Tabelle gespeichert werden
- Länge > 3964 Bytes, wird Datensatz außerhalb der Tabelle gespeichert, Datensatzes enthält Lokator, der auf physischen Speicherort verweist
- Für Verarbeitung von LOB-Werten, die nicht in der DB gespeichert werden sollen, bietet Oracle die Datentypen **TEMPORARY BLOB** / **CLOB** an
- Initialisierung von LOB-Attributen erfolgt mit **EMPTY_BLOB()** bzw. **EMPTY_CLOB()** - dies erzeugt einen leeren Lokator (auch nötig, wenn LOB NULL enthält)

Interne LOBs

- Transaktionskonzept - ACID (ROLLBACK möglich)
- Werte werden im Tablespace der DB gespeichert
- Länge < 3964 Bytes, so kann Datensatz in Tabelle gespeichert werden
- Länge > 3964 Bytes, wird Datensatz außerhalb der Tabelle gespeichert, Datensatzes enthält Lokator, der auf physischen Speicherort verweist
- Für Verarbeitung von LOB-Werten, die nicht in der DB gespeichert werden sollen, bietet Oracle die Datentypen **TEMPORARY BLOB** / **CLOB** an
- Initialisierung von LOB-Attributen erfolgt mit **EMPTY_BLOB()** bzw. **EMPTY_CLOB()** - dies erzeugt einen leeren Lokator (auch nötig, wenn LOB NULL enthält)

Interne LOBs

- Transaktionskonzept - ACID (ROLLBACK möglich)
- Werte werden im Tablespace der DB gespeichert
- Länge < 3964 Bytes, so kann Datensatz in Tabelle gespeichert werden
- Länge > 3964 Bytes, wird Datensatz außerhalb der Tabelle gespeichert, Datensatzes enthält Lokator, der auf physischen Speicherort verweist
- Für Verarbeitung von LOB-Werten, die nicht in der DB gespeichert werden sollen, bietet Oracle die Datentypen **TEMPORARY BLOB** / **CLOB** an
- Initialisierung von LOB-Attributen erfolgt mit **EMPTY_BLOB()** bzw. **EMPTY_CLOB()** - dies erzeugt einen leeren Lokator (auch nötig, wenn LOB NULL enthält)

Interne LOBs

- Transaktionskonzept - ACID (ROLLBACK möglich)
- Werte werden im Tablespace der DB gespeichert
- Länge < 3964 Bytes, so kann Datensatz in Tabelle gespeichert werden
- Länge > 3964 Bytes, wird Datensatz außerhalb der Tabelle gespeichert, Datensatzes enthält Lokator, der auf physischen Speicherort verweist
- Für Verarbeitung von LOB-Werten, die nicht in der DB gespeichert werden sollen, bietet Oracle die Datentypen **TEMPORARY BLOB** / **CLOB** an
- Initialisierung von LOB-Attributen erfolgt mit **EMPTY_BLOB()** bzw. **EMPTY_CLOB()** - dies erzeugt einen leeren Lokator (auch nötig, wenn LOB NULL enthält)

Interne LOBs

- Transaktionskonzept - ACID (ROLLBACK möglich)
- Werte werden im Tablespace der DB gespeichert
- Länge < 3964 Bytes, so kann Datensatz in Tabelle gespeichert werden
- Länge > 3964 Bytes, wird Datensatz außerhalb der Tabelle gespeichert, Datensatzes enthält Lokator, der auf physischen Speicherort verweist
- Für Verarbeitung von LOB-Werten, die nicht in der DB gespeichert werden sollen, bietet Oracle die Datentypen **TEMPORARY BLOB** / **CLOB** an
- Initialisierung von LOB-Attributen erfolgt mit **EMPTY_BLOB()** bzw. **EMPTY_CLOB()** - dies erzeugt einen leeren Lokator (auch nötig, wenn LOB NULL enthält)

Interne LOBs

- Transaktionskonzept - ACID (ROLLBACK möglich)
- Werte werden im Tablespace der DB gespeichert
- Länge < 3964 Bytes, so kann Datensatz in Tabelle gespeichert werden
- Länge > 3964 Bytes, wird Datensatz außerhalb der Tabelle gespeichert, Datensatzes enthält Lokator, der auf physischen Speicherort verweist
- Für Verarbeitung von LOB-Werten, die nicht in der DB gespeichert werden sollen, bietet Oracle die Datentypen **TEMPORARY BLOB** / **CLOB** an
- Initialisierung von LOB-Attributen erfolgt mit **EMPTY_BLOB()** bzw. **EMPTY_CLOB()** - dies erzeugt einen leeren Lokator (auch nötig, wenn LOB NULL enthält)

Externe LOBs

- BFILE-LOBs sind vom Transaktionsmechanismus ausgenommen
- Lokator verweist auf Betriebssystemdatei ausserhalb der DB
- in der DB muss ein virtuelles Verzeichnis angelegt werden, welches auf Betriebssystemverzeichnis zeigt, in dem die Dateien liegen
- dieses Verzeichnis muss für die DB lesbar sein
- BFILE-LOBs sind READ ONLY!

Externe LOBs

- BFILE-LOBs sind vom Transaktionsmechanismus ausgenommen
- Lokator verweist auf Betriebssystemdatei ausserhalb der DB
- in der DB muss ein virtuelles Verzeichnis angelegt werden, welches auf Betriebssystemverzeichnis zeigt, in dem die Dateien liegen
- dieses Verzeichnis muss für die DB lesbar sein
- BFILE-LOBs sind READ ONLY!

Externe LOBs

- BFILE-LOBs sind vom Transaktionsmechanismus ausgenommen
- Lokator verweist auf Betriebssystemdatei ausserhalb der DB
- in der DB muss ein virtuelles Verzeichnis angelegt werden, welches auf Betriebssystemverzeichnis zeigt, in dem die Dateien liegen
- dieses Verzeichnis muss für die DB lesbar sein
- BFILE-LOBs sind READ ONLY!

Externe LOBs

- BFILE-LOBs sind vom Transaktionsmechanismus ausgenommen
- Lokator verweist auf Betriebssystemdatei ausserhalb der DB
- in der DB muss ein virtuelles Verzeichnis angelegt werden, welches auf Betriebssystemverzeichnis zeigt, in dem die Dateien liegen
- dieses Verzeichnis muss für die DB lesbar sein
- BFILE-LOBs sind READ ONLY!

Externe LOBs

- BFILE-LOBs sind vom Transaktionsmechanismus ausgenommen
- Lokator verweist auf Betriebssystemdatei ausserhalb der DB
- in der DB muss ein virtuelles Verzeichnis angelegt werden, welches auf Betriebssystemverzeichnis zeigt, in dem die Dateien liegen
- dieses Verzeichnis muss für die DB lesbar sein
- BFILE-LOBs sind READ ONLY!

Externe LOBs

- BFILE-LOBs sind vom Transaktionsmechanismus ausgenommen
- Lokator verweist auf Betriebssystemdatei ausserhalb der DB
- in der DB muss ein virtuelles Verzeichnis angelegt werden, welches auf Betriebssystemverzeichnis zeigt, in dem die Dateien liegen
- dieses Verzeichnis muss für die DB lesbar sein
- BFILE-LOBs sind READ ONLY!

Beispiel

Definition einer Tabelle mit LOB-Datentypen:

```
CREATE TABLE LOBTable ( key NUMBER,
                        image BLOB DEFAULT EMPTY_BLOB(),
                        imageFile BFILE,
                        text CLOB );
```

Beispiel

Betriebssystemverzeichnis für BFILE-Datentyp festlegen:

```
CREATE DIRECTORY IMAGES_DIR AS 'C:\Images\';
```

Einfügen eines Beispieldatensatzes:

```
INSERT INTO LOBTable (
VALUES ( 1,
         EMPTY_BLOB(),
         BFILENAME('IMAGES_DIR', bild.gif),
         EMPTY_CLOB()
);
```

Beispiel

Betriebssystemverzeichnis für BFILE-Datentyp festlegen:

```
CREATE DIRECTORY IMAGES_DIR AS 'C:\Images\';
```

Einfügen eines Beispieldatensatzes:

```
INSERT INTO LOBTable (
VALUES ( 1,
         EMPTY_BLOB(),
         BFILENAME('IMAGES_DIR', bild.gif),
         EMPTY_CLOB()
);
```

Tuningansätze

Tuningansätze

- Modifikation der STORAGE-Klausel
- sofortiges Freigeben nicht mehr benötigter temporärer LOBs, diese bleiben sonst bis zur Terminierung der Datenbankverbindung im Speicher
- LOB in anderen Tablespace als den, der den LOB enthält speichern - bei mehreren LOBs in einer Tabelle evt. mehrere Tablespaces nutzen (auf mehrere Platten verteilen)
- für kleine LOBs (z.B. <8K) Storage-Klausel so anpassen, daß diese in der Tabelle gespeichert werden

Tuningansätze

Tuningansätze

- Modifikation der STORAGE-Klausel
- sofortiges Freigeben nicht mehr benötigter temporärer LOBs, diese bleiben sonst bis zur Terminierung der Datenbankverbindung im Speicher
- LOB in anderen Tablespace als den, der den LOB enthält speichern - bei mehreren LOBs in einer Tabelle evt. mehrere Tablespaces nutzen (auf mehrere Platten verteilen)
- für kleine LOBs (z.B. <8K) Storage-Klausel so anpassen, daß diese in der Tabelle gespeichert werden

Tuningansätze

Tuningansätze

- Modifikation der STORAGE-Klausel
- sofortiges Freigeben nicht mehr benötigter temporärer LOBs, diese bleiben sonst bis zur Terminierung der Datenbankverbindung im Speicher
- LOB in anderen Tablespace als den, der den LOB enthält speichern - bei mehreren LOBs in einer Tabelle evt. mehrere Tablespaces nutzen (auf mehrere Platten verteilen)
- für kleine LOBs (z.B. <8K) Storage-Klausel so anpassen, daß diese in der Tabelle gespeichert werden

Tuningansätze

Tuningansätze

- Modifikation der STORAGE-Klausel
- sofortiges Freigeben nicht mehr benötigter temporärer LOBs, diese bleiben sonst bis zur Terminierung der Datenbankverbindung im Speicher
- LOB in anderen Tablespace als den, der den LOB enthält speichern - bei mehreren LOBs in einer Tabelle evt. mehrere Tablespaces nutzen (auf mehrere Platten verteilen)
- für kleine LOBs (z.B. <8K) Storage-Klausel so anpassen, daß diese in der Tabelle gespeichert werden

Oracle 10g Release 2

STORAGE-Klausel

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

CHUNK

- Anzahl Oracle Blocks (max: 32K) - Zugriff in großen Chunks effizienter, verschwendet allerdings Speicherplatz bei kleineren Daten

Oracle 10g Release 2

STORAGE-Klausel

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

CHUNK

- Anzahl Oracle Blocks (max: 32K) - Zugriff in großen Chunks effizienter, verschwendet allerdings Speicherplatz bei kleineren Daten

Oracle 10g Release 2

STORAGE-Klausel

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

PCTVERSION

- (Default: 10) - Prozentsatz vom LOB-Speicherplatz der für alte Versionen benutzt werden kann, bei seltenen Updates kann Wert verkleinert werden

Oracle 10g Release 2

STORAGE-Klausel

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

RETENTION

- alte Versionen werden für eine bestimmte Zeit behalten (einstellbar durch UNDO_RETENTION Parameter)

Oracle 10g Release 2

STORAGE-Klausel

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

CACHE READS

- viel lesen, selten schreiben

Oracle 10g Release 2

STORAGE-Klausel

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

CACHE

- viel lesen, viel schreiben

Oracle 10g Release 2

STORAGE-Klausel

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

NOCACHE (DEFAULT)

- selten lesen, nie schreiben

Oracle 10g Release 2

STORAGE-Klausel

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

NOLOGGING

- nicht möglich mit CACHE, praktisch für Bulk Load

Oracle 10g Release 2

STORAGE-Klausel

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

STORAGE IN ROW

- erzwingt die Daten innerhalb der Tabelle zu speichern (falls sie hineinpassen)

Fragen?

Vielen Dank für die Aufmerksamkeit.

Quellenangabe

- *Oracle 10g Application Developer's Guide - Large Objects*, Juni 2005
- Alexander Biliris, *The EOS Large Object Manager*, AT&T Bell Laboratories, December 1992
- Stefan Dieker, Ralf Hartmut Güting, *Efficient Handling of Tuples with Embedded Large Objects*, FernUniversität Hagen
- Peter Eisentraut, *PostgreSQL - Das Offizielle Handbuch*, mitp-Verlag, 1. Auflage 2003
- Christina Böttger, *Oberseminar - Moderne Datenbanken*, HTWK Leipzig, November 2003