

第11章 大对象的历史

本章要点：

历史

Oracle 7与Oracle 8结构的对比

LOB

BFILE

使用LOB和BFILE的例子

DBMS_LOB包

更多的一些例子

最后的一些问题

11.1 历史

关系型数据库管理系统（RDBMS）一开始对大对象（LOB）的存储和查询的尝试是粗糙的。有趣的是尽管这些开始的尝试是粗糙的，但是他们的确在一些初步的级别上完成了这项工作，这可能解释为什么这些早期的方法在今天仍然对于不同厂商以不同的名字存在。

大部分RDBMS厂商做的早期尝试是外部存储数据类型方法，通常在数据库世界中称之为二进制大对象（BLOB），不要与本章后面将要讨论的内部 Oracle 8二进制大对象数据类型（BLOB）相混淆。这是一种管理大的、未被结构化数据的常规方法。大的意思是什么？这实际上对于不同的RDBMS厂商是不同的，但是通常它的意思是比厂商能支持的最大的变长字符要大。例如在Oracle 7中，最大的变量是varchar2 (2000)。Rdb/VMS、Sybase和Oracle在二进制大对象上都有一些变化。Oracle 7有LONG和LONG RAW，它们在本章后面部分“Oracle 7与Oracle 8结构的对比”中与Oracle 8新的数据类型做比较。未结构化的意思是什么？一般来讲，它的意思是数据可能代表平凡（字母数字）信息，如注释、标记、记事簿、摘要、对调查问题的开放结尾回答等等。那些熟悉dBASE的用户应使用过dBASE术语MEMO域。但它也有多媒体数据的意思，包括图形、工程图、图像、视频、声音等等。

换句话说，BLOB被创建以存储不规则结构的数据，但是能可靠地分解到一张表的每一行的非空列上。实际上，如果数据在某种程度上依附在一个规范化表的右边，由于在每一行中有不同长度的数据，这个表看起来将是不规则的。对于熟悉C/C++的用户，他们将很快认识到这是一个“不规则数组”。从本质上说，对大多数关系数据库设计者而言，将不会这么做：在数据库内部存储数据，依附到一个特定的表并破坏规范化的设计。在今天所有的RDBMS厂商仍然使用的快速补救方法是BLOB，BLOB将存储在数据库外部的操作系统文件中。存储在表中的将是列名和在每一行的那列中指向相关的外部文件的指针。这样，表好像被“正规化”了，因为指针自身具有相同的数据类型（是完全专用的和对特定的RDBMS厂商的实现特定的）。

使用专有实现的缺点

与大多数厂商对关系模型的扩展一样，在RDBMS中的专有数据类型不是（现在仍然不是）一个好主意。为什么？因为这将完全或部分地牺牲可移植性。有这种情况：一个公司的应用完全用Sybase的TEXT列（一种Sybase的BLOB类型）开发；然后，由于这样或那样的原因必须抛弃全部的Sybase环境，公司不得不重新在Oracle中实现每一件事。猜一猜怎么了？是的，所有的Sybase专有的BLOB（TEXT列）不得不用Oracle专列BLOB（LONG列）替代。

除了专有数据类型自身之外存在的其他不利因素是什么？第一，正如你可能预期的，专有数据类型意味着非标准，这意味着非ANSI SQL，反过来意味着没有标准的语言支持数据类型。因此，只要有专有数据类型，你必须伴随着它使用专有语言存储、查询并用不同的方法操作它。换句话说，它需要自己的DDL和DML操作集，实际上，所有厂商都有这种情况。因此那些必须从Sybase转移到Oracle的公司不仅必须替换数据类型，而且必须重新编写代码，这通常是在任何转移操作中的最后一种选择手段。

除了数据类型自身和它的相关操作是否还有其他缺点？肯定有。其他特定厂商的环境和操作的行为模式通常要求你或者用“BLOB使能”模式工作，或者不用这种模式工作。你经常会发现厂商（包括Oracle在内）在许多日常操作中不允许BLOB操作或访问。在Oracle 7中用LONG型工作的人员能够很好地理解这一点。

用BOLB最根本的问题是它们不是数据库“头等公民”。这意思是，尽管它们被外围存储和查询，但是它们并不与数据库中的其他数据或部分以任何形式发生交互。一种考虑这种情况的方法是BLOB实际上仅代表RDBMS具有附加的简单操作文件系统的能力，再也没有其他的了。到关系型模型的分支也不能被忽略。一般来说，这些早期BLOB是很少有或根本没有并发控制的，这意味着锁完全不是相对于数据库页或行的锁。另外，优化器不“了解”这些BLOB，不能产生任何形式的包括它们的查询计划。在一个相关点上，如索引和普通的字符集数据子串操作不能用于BLOB上。你作为DBA或开发者可能想去做的所有其他事情，或者希望RDBMS处理的事情能不用BLOB。因此，使用Oracle 8，早期的Oracle 7的BLOB已经转生为另一种形式，但是Oracle的确朝着使BLOB成为关系模型实现中更加有用的列的方面取得了一些进步。

11.2 Oracle 7与Oracle 8结构的对比

在Oracle 7中，LONG和LONG RAW数据类型是内部BLOB。除了它们是内部的之外，它们也具有BLOB的坏特性。在Oracle 8中，大对象（LOB）可以是内部的或外部的。表11-1概括了Oracle 7 LONG与Oracle 8 LOB之间的不同。

表11-1 Oracle 7与Oracle 8大对象的对比

Oracle版本	7 (LONG)	8 (LOB)
最大尺寸	2GB	4GB
内部或外部	内部	皆可
访问	顺序	随机
SELECT返回值	数据	指针
每个表的列	一个	多个

将旧的LONG与新的LOB数据类型进行对比是非常有益的。用Oracle 7的LONG，仅有两个基本LONG类型：LONG字符和LONG RAW二进制。Oracle 7 LONG在它自己的“表中”存

储最多2GB的数据，只能按顺序进行访问，每一个表只能存储一行，SELECT语句直接返回实际的数据。

用Oracle 8 LOB存储数据能够达到4GB，既可以存储在表的内部又可存储在表的外部，能够被随机（直接）访问，每一个表能够存储多于一行的LOB，SELECT语句返回一个指向数据（不管是在存储在表内部还是外部）的指针（定位器）。

11.3 LOB

在Oracle 8中，有四种类型的LOB：BLOB、CLOB、NCLOB和BFILE。它们概括如下：

BLOB——内部二进制大对象。

CLOB——内部字符大对象。

NCLOB——内部定长多字节字符大对象。

BFILE——外部二进制文件。

正如你可能注意到的，Oracle 8 BFILE是早期RDBMS BLOB（再一次指出，不要将其与Oracle 8 BLOB混淆）的直接继承，它作为数据库指针存储在数据库内部，指向数据库的外部操作系统文件。

LOB由两部分组成：数据（值）和指向数据的指针（定位器）。尽管值与表自身一起存储，但是一个LOB列并不包含值，仅有它的定位指针。更进一步，为了使用大对象，程序必须声明定位器类型的本地变量。你将在本章后面见到PL/SQL接口使用定位器如同操作系统的文件处理一样。当LOB（除了BFILE）被创建时，定位器被存放在列中，值被存放在LOB段中，LOB段是在数据库内部表的一部分。当BFILE被创建时，定位器如同平常一样存储在列中，但是它的值被存储在数据库之外的操作系统文件中。

正如你已学过的，内部LOB（BLOB、CLOB和NCLOB，但不包括BFILE）可以是一个表的列、一个SQL变量或者一个PL/SQL变量。最重要的是，考虑忠实于关系型模型的实现，内部LOB主要是修改并发控制、重做日志和备份/恢复机制，这代表了在早期的RDBMS BLOB基础上的进步。Oracle 8的BLOB在功能上与Oracle 7 LONG RAW相似，CLOB的功能与LONG相似。但是NCLOB是新出现的。它们被创建用来处理国家性字符（NC）集。最后，在这些不同的内部LOB类型之间没有隐式转换。

11.3.1 使用LOB创建表

正如你通常使用的创建表的语句一样，使用CREATE TABLE语句可以创建包含一些LOB列的表。在INSERT操作时，定位器和空值被创建。PL/SQL的DBMS_LOB包或者Oracle Call Interface（OCI）能够用来装载LOB。删除包含LOB的行，不仅删除定位器，而且还删除相关的值（正如所希望的）。UPDATE语句能更新全部的LOB值，但是不能正好更新它之中的一块（例如在字符类型中子集的操作）。还有，当在INSERT创建LOB时，LOB索引段和LOB（值）段会自动地创建，这个索引段索引LOB值的片断以便随机访问。因此，每一个LOB创建时，创建两个段：一个LOB（值）段和一个LOB索引段。

11.3.2 LOB存储管理

内部LOB存储是如何管理的？存储和并发粒度称为一大块（chunk），它是由一个或多个

Oracle块（由db_block设置）组成的。大块的缺省大小是一个 Oracle块。实际上，一个大块是一串块（链表）。在一个大块中的块在逻辑上是连续的，尽管在给定的链表数据结构下不要求连续。这些知识可以用来协调数据库的其他方面，如设置 db_file_multiblock_read_count等于大块的大小能使性能增强。LOB索引维护大块的链表以使它们可以被直接访问；没有 LOB索引，它们只能被顺序访问。LOB索引是一个大块值到大块的地址的转换列表。版本化是使相同LOB的各种版本能够占有不同比例的 LOB段的机制。存储参数 pctversion表明LOB段空间的多少比例用于存储除了当前版本外的所有（老的）版本。如果当前 LOB的老版本超过 pctversion的值，那么 Oracle尽力从最老版本中申请空间，直到老版本小于或等于 pctversion的值。版本化发生在大块级。

另外两个存储参数 cache/nocache和logging/nologging分别控制I/O是否使用系统全局区数据库缓冲区和重做日志机制。缺省组合 nocache和nologging，对一般的性能是最好的。正如通常一样，当预期重复再使用 LOB时，使用cache；当对LOB的关键性改变必须不能丢失时，使用loggin。这与通常的对象没有特别不同之处。

什么是内部LOB并发管理？结果是与块一样，大块并不只是存储单元（和 I/O）和版本化，而且也是并发的单元。对读一致性的处理与对其他普通的 Oracle数据类型非常相似。使用 LOB的最主要的区别在于粒度是大块而不是块，读一致性 LOB数据（老的大块代表老版本）被存储在LOB段内，而不是存储在回滚段或数据库缓冲区中。一般来说，如果对 LOB有很多更新活动，在相同的时间框架中还具有高选择性，那么 pctversion应该尽可能相对高一些，以使能更多的非中断并发。什么是中断？就是当使用读一致性时，用户可能遇到 ORA-1555错误“快照太老”。其基本意思是你需要更多的回滚空间（多个单个段存储或多个段）。使用LOB段，你能增加存储总量或者如果 pctversion充足，仅仅简单地增加pctversion。与此相对，如果低的更新的活动或高的更新活动与高的查询活动不在同一时间发生，pctversion值不是关键的，可以设置得低一些。

11.4 BFILE

文件系统和第三代语言形成了前一代关系型数据库管理系统的基础，甚至是前数据库管理系统、信息存储和检索系统（或者简称为信息系统）的基础。但是，他们的缺点正是朝着 DBMS系统和最终朝着 RDBMS方向发展的驱动力。但是，尽管缺少主要的现代的 RDBMS子系统，基于文件的信息系统仍在某种程度上在一些领域内很广泛采用，特别是，某种基于文件的信息系统可能受益于旧的 BLOB外部存储和检索（在 Oracle 8中的BFILE）方法。最可能的候选者应该不是面向事务的，不是主要依赖于数据的完整性的，而是可能形成于一些松散的结构化的或非结构化的历史应用的基础上，如服务在因特网上的记录本或简历归档。

Oracle 8能够创建、相关和安全化 BFILE对象。然而对 BFILE所做的所有其他事必须通过 RDBMS_LOB包或OCI来做。BFILE是只读的。因此，服务于前述历史应用的 BFILE很适合存储在光盘（CDROM）中。数据必须已经作为操作系统的文件存在，Oracle必须有读该文件（当然还有目录访问、到达文件）的特权。在删除表空间时，数据文件作为操作系统文件仍留在操作系统中；同样，删除 BFILE对象时没有删除相关的操作系统文件。作为 Oracle的DBA要清楚这一点，必须亲自删除它们，或让操作系统管理员来做这项工作。

正如所提到的，当 BFILE被创建时，它的定位器存储在一些表的某一列中，而它的值是

相关的外部操作系统文件自身。BFILE定位器存储的信息不仅仅是文件名，还包括目录和其他有关的状态信息。在“规范化”的表中，每行和列交叉处能包含一个唯一的值（尽管它也能包含在外键中以前被使用的值）。同样，每个BFILE列的每一行能够指向不同的文件。正如同其他内部LOB一样，DBMS_LOB包和Oracle 8 OCI提供访问BFILE的I/O流。不同于发生在普通对象（例如表）和SGA正常块之间的I/O，BFILE I/O（通过DBMS_LOB或OCI）是直接的，类似于用SQL*loader直接的路径装载。换句话说。它绕过SGA。

数据库子系统如并发控制、重做日志和备份/恢复机制在Oracle 8中用BFILE都无法获得。另外安全性是有疑问的。因为BFILE在Oracle 8数据库内的安全性主要依靠定义它的操作系统的安全性。例如，为了BFILE的安全，SA或DBA必须设置目录和文件访问控制（对Oracle读允许），使能Oracle之外的应用访问权限，为初始存储分配足够的存储空间和文件预期的增长空间，并考虑到操作系统或硬件地址的限制，如每一个物理磁盘的最大文件尺寸、单个逻辑卷的容量或绝对的（无条件的）最大值。举一个例子，Solaris 2.2可以访问（通过它的I/O接口）的文件到达2GB，如同许多早期的UNIX实现的一样。但是用Solaris 2.3，“大”文件和文件系统被使能，其尺寸大于2GB（例如4GB或9GB，它和当时可获得的最大单个磁盘的尺寸相一致）。

为了帮助管理员在Oracle 8中访问BFILE，目录数据库对象被创建。这个目录对象是操作系统目录的一个别名或同义词，因此，目录对象是虚拟的。这些内部Oracle目录能被保证安全，通过Oracle内部机制（与操作系统安全相对），提供唯一真正安全的方式对BFILE进行访问。用户能被授予访问这些目录的特权，并且它们能够使用于PL/SQL或OCI。一个DBA或用户在Oracle 8上有CREATE ANY DIRECTORY特权，能够创建一个目录。然后，与任何其他普通数据库对象一样，特权被授予和收回。

在运行检查对BFILE的访问，可能发生的运行错误是以下几种：

- 不充足的用户特权。
- 目录不存在。
- 文件不存在。

当BFILE或目录被创建时，不检查正确的特权、目录名和文件名是否存在（在编译时）。

11.5 使用LOB和BFILE的例子

看一些LOB和BFILE的例子。

清单 11-1 用一个BLOB(photo)和一个CLOB(resume) 创建一个表。CREATE语句产生了五个段：表自身，另两个LOB各有两个段。然后，一个很简单的PL/SQL块声明一个本地变量CLOB(r)和一个本地变量BLOB(p)，用来查询在数据库中相关的LOB对象的定位指针。

清单 11-1 用CLOB和BLOB创建一个表

```
SQL> create table candidates (  
2 pk number(9),  
3 name varchar2(40),  
4 address varchar2(80),  
5 resume CLOB,  
6 photo BLOB );  
SQL> declare  
2 r CLOB;  
3 p BLOB;
```

```

4 Begin
5   SELECT resume into r from candidates where pk = 12345;
6   SELECT photo into p from candidates where pk = 12345;
7 End;

```

清单 11-2 用三个 LOB、两个 BLOB 和一个 CLOB 创建一个表。大块的大小被设置为 32 个块（你也可能想在 init_ora 初始化参数文件中设置 db_file_multiblock_read_count 为 32）。pctversion 被设置为 30%。这样老的 LOB 版本能占有总的 LOB 段的 30%。缺省的 nocache 和 nologging 在这里进行特别重申。随后是对 LOB 的索引语句，指定的范围大小比 LOB 实际使用的尺寸要小些（大约十分之一）。

清单 11-2 用 LOB、CLOB 和 BLOB 创建一个表

```

SQL> create table landscape_plans (
2   front_yard BLOB,
3   back_yard BLOB,
4   contract CLOB,
5   plan_name varchar2(20),
6   plan_budget number(9)
7   LOB (front_yard, back_yard, contract)
8   store as (
9   storage (initial 1M next 1M pctincrease 0)
10  chunk 32
11  pctversion 30
12  nocache /* default */
13  nologging /* default */
14  index (
15  storage (initial 128K next 128K)
16  ) /* end LOB clause */
17 ); /* end create table */

```

清单 11-3 创建有两个 BFILE 列的表，一个用于存储员工的照片（photo），另一个存储指纹（thumbprint）。注意，你实际没有（或需要）任何附加的存储参数，因为 BFILE 实际上更多是在操作系统域中存储。

清单 11-3 用 BFILE 创建表

```

SQL> create table background_check (
2   employee_id number(5),
3   hire_date date,
4   photo BFILE,
5   thumbprint BFILE );

```

清单 11-4 说明了怎样在操作系统级和在 Oracle 8 内为 thumbprint 和 photo 创建目录。

清单 11-4 为 LOB 创建目录

```

# mkdir /user/data/bfiles/thumbprints /* UNIX or DOS */
# mkdir /user/data/bfiles/photos

SQL> create directory thumbprints as '/user/data/bfiles/thumbprints';

SQL> create directory photos as
'/user/data/bfiles/photos';

```

11.6 DBMS_LOB 包

下面概括了 DBMS_LOB 过程和函数以及调用形式并对它们的功能进行描述。

1. APPEND (仅用于LOB)

示例语法：APPEND (d,s)

注释：在d后附加s。

2. COMPARE

示例语法：result:=COMPARE(b1 , b2 , n , o1 , o2)

注释：比较b1与b2 (包括BFILE) 的n个字节，从b1偏移o1、b2偏移o2处开始。结果是0表示相等；如果b1小于b2，结果是-1；b1大于b2，结果是1。

3. COPY

示例语法：COPY (d,s,n,do,so)

注释：从d复制n个字节到s，从d偏移do、s偏移so处开始。

4. ERASE (仅用于LOB)

示例语法：ERASE (b,n,o)

注释：从b中擦除几个字节，从偏移量o处开始。

5. FILECLOSE (仅用于BFILE)

示例语法：FILECLOSE (b)

注释：关闭一个打开的BFILE b。

6. FILECLOSEALL (仅用于BFILE)

示例语法：FILECLOSEALL

注释：关闭当前所有打开的BFILE。

7. FILEEXISTS (仅用于BFILE)

示例语法：result:=FILEEXISTS(b)

注释：如果在操作系统中实际文件BFILE b存在返回1，否则返回0。

8. FILEGETNAME (仅用于BFILE)

示例语法：FILEGETNAME (b,d,f)

注释：给定一个BFILE b，返回目录d和文件名f。

9. FILEISOPEN (仅用于BFILE)

示例语法：FILEISOPEN (b)

注释：确定BFILE是否打开。

10. FILEOPEN (仅用于BFILE)

示例语法：FILEOPEN(b,m)

注释：以模式m (例如，file_readonly) 打开文件BFILE b。

11. LOADFROMFILE

示例语法：LOADFROMFILE (b,l,n,bo,lo)

注释：从BFILE b复制 n个字节到LOB l，从b偏移bo、l偏移lo处开始。

12. GETLENGTH

示例语法：length:=GETLENGTH(l)

注释：返回LOB l的长度。如果这个LOB是空的，返回长度将是0。

13. INSTR

示例语法：position:=INSTR(l,p,o,n)

注释：返回LOB l中p的位置，从偏移量o处开始查找p的第n次出现。

14. READ

示例语法：READ (l, n, o, b)

注释：从LOB l中读取n个字节的数据，从偏移量o处开始，返回读入缓冲区b中的字节数。当前返回缓冲区b是RAW或VARCHAR 2类型，仅能处理32K长度。

15. SUBSTR

示例语法：s := SUBSTR (l, n, o)

注释：返回从偏移o处开始LOB l中n个字节的子串s。与READ一样，返回值是RAW或VARCHAR2型，仅能达到32K。

16. TRIM (仅用于LOB)

示例语法：TRIM (l, n)

注释：从右边TRIM (减少) LOB l到长度n。

17. WRITE

示例语法：WRITE (l, n, o, b)

注释：从缓冲区b中读n个字节到LOB l中，从偏移量o处开始，与READ和SUBSTR相同，WRITE仅能从RAW或VARCHAR 2缓冲区b中写最多32K。

另外，除了这些，还有“构造器”类型函数创建空的LOB。正如你想象的那样，EMPTY_CLOB ()和EMPTY_BLOB ()分别建立空的CLOB和空的BLOB。

11.7 更多的一些例子

清单11-5是插入一个空的BLOB和用一个已存在的BLOB值副本更新该BLOB列的例子。

清单11-5 插入和更新一个BLOB列

```
SQL> create table inventory (  
2 ID number(9),  
3 cost number(6),  
4 picture BLOB );  
  
SQL> insert into inventory values (  
2 123456789,  
3 5900, /* $59.00 */  
4 empty_blob() );  
  
SQL> update inventory  
2 set picture = (  
3 select picture from inventory  
4 where ID=999999999 )  
5 where ID=123456789;  
/* This copies the picture belonging  
to item 999999999 to our new item 123456789. */
```

注意这里没有要求任何特定的锁来完成这个复制。这是因为这里做的是全规模复制。但是使用DBMS_LOB例程来移动数据时，你需要用SELECT FOR UPDATE语句锁定你的行，正如清单11-6所示。清单11-7展示如何使用空LOB函数从一行中分离LOB的值。

清单11-6 使用LOB移动数据

```
SQL> create table memo_archive (  
2 ID number(9),
```

```
3 memo CLOB );

SQL> declare
buffer varchar2(80);
3 local_memo CLOB;
4 begin
5 buffer:="For Official Use Only";
6 select memo into local_memo
7 from memo_archive where ID=873561495
8 for update;
9 dbms_lob.write(local_memo,length(buffer),65536,buffer);
10 commit;
11 end;
12 /
```

清单 11-7 使用空LOB函数

```
SQL> update memo_archive
2 set memo=empty_clob()
3 where ID=371620603;
```

这使LOB定位器原封不动，但简单地使LOB为空（设置它为“空值字符串”）。

11.8 最后的一些问题

BLOB和BFILE是以字节为单位衡量的，而CLOB和NCLOB则以字符为单位衡量。仅对CLOB来说，一个字节等于一个字符，但对于多字节字符集和NCLOB这就不成立了，对于传给DBMS_LOB过程和函数的偏移量，这一点很重要。缺省的偏移量被设置为1，意思是BLOB或BFILE的第一个字节和CLON或NCLOB的第一个字符。

有些问题来自于处理导出/导入。导入将CLOB和NCLOB从导出传储文件的字符集转换为目标数据库字符集（或国家字符集）。要确保你的NLS参数设置正确。BFILE文件不被EXPORT导出，但能导出定位器和目录。导入之后，手动移动操作系统文件并重新将BFILE文件关联到新的目录。