

SecureFiles: новые большие объекты (SecureFiles: The New LOBs, by Arup Nanda)

Аруп Нанда,
член-директор  Oracle ACE

Источник: сайт корпорации Oracle, серия статей «Oracle Database 11g: The Top New Features for DBAs and Developers»,
<http://www.oracle.com/technology/pub/articles/oracle-database-11g-top-features/11g-securefiles.html>

Рассматривается использование следующего поколения LOB-объектов: SecureFiles (обеспеченные файлы), которые объединяют лучшее качества внешних файлов (external files) и LOB-объектов базы данных для хранения неструктурированных данных, допускает также шифрование, сжатие, однократное хранение нескольких одинаковых объектов (deduplication – дедупликация) и т.д.

Внутренние двоичные объекты базы данных (BLOB) и файлы операционной системы

Что находится в базе данных Oracle? Обычно, это данные, хранимые в реляционном виде, применяемые для простого преобразования в некий вид заданного шаблона и определенного типа данных: имена пользователей, остатки на счетах, коды состояний и т.п. Но со столь же большой долей вероятности может понадобиться хранение данных в неструктурированном или полуструктурированном виде. Например, изображения, текстовые документы, таблицы, XML-файлы и т.п. Как хранятся эти типы данных?

Обычно используется два подхода: такие данные хранятся в базе данных как LOB-поля (BLOB для двоичных и CLOB для символьных данных) или в виде файлов операционной системы, а в базе данных хранятся на них ссылки.

Каждый подход имеет достоинства и недостатки. Файлы операционной системы могут кэшироваться и журналироваться средствами операционной системы, что ускоряет их восстановление после сбоев. К тому же они обычно занимают меньше места, чем данные в базе данных, поскольку могут быть сжаты.

Существуют так же средства, которые позволяют распознавать шаблоны в файлах и удалять повторяющиеся элементы для более эффективного хранения. Но они не являются частью базы данных и не имеют ее свойств. Резервное копирование таких файлов не выполняется, не доступны тонкости политики безопасности, эти файлы не являются частью транзакций, то есть целостность данных, первичная концепция базы данных Oracle, на них не распространяется.

Как же воспользоваться преимуществами каждого подхода? В Oracle Database 11g имеется ответ – SecureFiles, совершенно новая инфраструктура в базе данных,

которая объединяет лучшие особенности LOB-объектов базы данных и файлов операционной системы. Давайте рассмотрим, как они работают. (Кстати, традиционные LOB-типы до сих пор доступны в виде так называемых *BasicFiles*).

Практический пример

Возможно, лучше всего представить концепцию SecureFiles с помощью простого примера. Предположим, вы разрабатываете систему документооборота, в которой хотите поместить в таблице копии договоров. Обычно отсканированные документы представляют собой не текстовые, а PDF-файлы. Некоторые - документы MS Word или даже отсканированные рисунки. Это отличный пример для использования BLOB-объектов, так как такой столбец должен поддерживать двоичные данные.

Традиционное, до Oracle Database 11g, определение таблицы было бы следующим:

```
create table contracts_basic
(
  contract_id number(12),
  contract_name varchar2(80),
  file_size number,
  orig_file blob
)
tablespace users
lob (orig_file)
(
  tablespace users
  enable storage in row
  chunk 4096
  pctversion 20
  nocache
  nologging
);
/
```

Реальные файлы хранятся в двоичном формате в столбце ORIG_FILE. Другие параметры показывают, что LOB-объект не должен кэшироваться и журналироваться в операциях, он хранится в строке таблицы, имеет размер

порции (chunk) 4 КБ, таблица находится в табличном пространстве USERS. Поскольку это не определено, LOB-объекты хранятся в Oracle Database 11g в общепринятом формате (BasicFiles).

Если надо хранить LOB-объект в виде SecureFile, то все, что необходимо сделать, это при создании таблицы записать фразу store as securefile, как показано ниже:

```
create table contracts_sec
(
  contract_id number(12),
  contract_name varchar2(80),
  file_size number,
  orig_file blob
)
tablespace users
lob (orig_file)
store as securefile
(
  tablespace users
  enable storage in row
  chunk 4096
  pctversion 20
  nocache
  nologging
)
/
```

Для того чтобы создать LOB-объект в виде SecureFile, необходимо выполнить два условия, причем оба выполняются по умолчанию (так что можно быть спокойным).

- Параметр инициализации **db_securefile** должен быть установлен в **permitted** (значение по умолчанию). Я объясню, что это за параметр позднее.

- Созданное табличное пространство, где размещаются обеспеченные (securefile) файлы, должно быть **Automatic Segment Space Management (ASSM)**. ASSM - режим по умолчанию при создании табличного пространства в Oracle Database 11g, поэтому он уже установлен для табличного пространства. Если это не так, тогда необходимо размещать SecureFiles на новом табличном ASSM-пространстве.

После того, как таблица создана, можно загружать данные тем же способом, как и обычные (до-11g) LOB-объекты (BasicFile). Не надо изменять приложения и не надо запоминать какой-то специальный синтаксис.

Вот пример небольшой программы, которая загружает данные в таблицу:

```
declare
  l_size number;
  l_file_ptr bfile;
  l_blob blob;
begin
  l_file_ptr := bfilename('SECFILE', 'contract.pdf');
  dbms_lob.fileopen(l_file_ptr);
  l_size := dbms_lob.getlength(l_file_ptr);
  for ctr in 1 .. 100 loop
```

```
insert into contracts_sec
(
  contract_id,
  contract_name,
  file_size,
  orig_file
)
values
(
  ctr,
  'Contract '||ctr,
  null,
  empty_blob()
)
returning orig_file into l_blob;
dbms_lob.loadfromfile(l_blob, l_file_ptr, l_size);
end loop;
commit;
dbms_lob.close(l_file_ptr);
end;
/
```

Эта программа 100 раз загружает файл contract.pdf в 100 строк таблицы. Заранее надо иметь определенный объект типа **directory**, названный SECFILE, для директории операционной системы, где расположен файл contract.pdf. Ниже приводится пример, где файл contract.pdf расположен в директории /opt/oracle.

```
SQL> create directory secfile for '/opt/oracle';
```

Один раз сохранив LOB-объект как SecureFile, вы получаете множество возможностей для выполнения оптимальных действий. Приведем несколько примеров этих весьма полезных возможностей.

Однократное хранение нескольких одинаковых объектов (deduplication)

Дедупликация, вероятно, самая яркая возможность SecureFiles, поскольку из всех преимуществ файлов операционной системы перед внутренними BLOB-объектами она наиболее востребована. Допустим, в таблице хранится пять записей, каждая с BLOB-объектом. Три из них одинаковы. Если была бы возможность однократного хранения BLOB-объекта и размещения ссылки на него в других двух записях, это существенно бы уменьшило занимаемое место. Это возможно для файлов операционной системы, но не для LOB-объектов в Oracle Database 10g. Для SecureFile это легко реализуется с помощью свойства дедупликации. Его можно определить при создании таблицы или позднее:

```
SQL> alter table contracts_sec
2 modify lob(orig_file)
3 (deduplicate)
4 /
```

Table altered.

В процессе дедупликации СУБД хеширует значения столбцов в каждой строке и сравнивает хеш-значения друг с другом. Если хеш-значения совпадают, то сохраняются они, а не исходный BLOB-объект. Когда добавляется новая запись, то вычисляется хеш, и если он совпадает со значением в другой строке, то сохраняется хеш-значение, в противном случае сохраняется реальное значение.

Теперь давайте определим объем пространства, сохраненного в результате дедупликации. Определение занимаемого пространства LOB-сегментом можно выполнить с помощью пакета DBMS_SPACE. Ниже демонстрируется пример программы, показывающей занимаемое место:

```

declare
  l_segment_name  varchar2(30);
  l_segment_size_blocks  number;
  l_segment_size_bytes  number;
  l_used_blocks      number;
  l_used_bytes       number;
  l_expired_blocks   number;
  l_expired_bytes    number;
  l_unexpired_blocks number;
  l_unexpired_bytes  number;

begin
  select segment_name
  into l_segment_name
  from dba_lobs
  where table_name = 'CONTRACTS_SEC';
  dbms_output.put_line('Segment Name='|| l_segment_name);

  dbms_space.space_usage(
    segment_owner  => 'ARUP',
    segment_name   => l_segment_name,
    segment_type   => 'LOB',
    partition_name => NULL,
    segment_size_blocks  => l_segment_size_blocks,
    segment_size_bytes  => l_segment_size_bytes,
    used_blocks         => l_used_blocks,
    used_bytes          => l_used_bytes,
    expired_blocks      => l_expired_blocks,
    expired_bytes       => l_expired_bytes,
    unexpired_blocks    => l_unexpired_blocks,
    unexpired_bytes     => l_unexpired_bytes
  );

  dbms_output.put_line('segment_size_blocks  => '|| l_seg-
ment_size_blocks);
  dbms_output.put_line('segment_size_bytes   => '|| l_seg-
ment_size_bytes);
  dbms_output.put_line('used_blocks         => '|| l_used_
blocks);
  dbms_output.put_line('used_bytes          => '|| l_used_bytes);
  dbms_output.put_line('expired_blocks      => '|| l_expired_
blocks);
  dbms_output.put_line('expired_bytes       => '|| l_expired_
bytes);
  dbms_output.put_line('unexpired_blocks    => '|| l_unex-
pired_blocks);
  dbms_output.put_line('unexpired_bytes     => '|| l_unex-
pired_bytes);
end;
/

```

Этот скрипт показывает различные значения занимаемого LOB-объектами пространства. Вот его результат до процесса дедупликации:

```

Segment Name=SYS_LOB0000070763C00004$$
segment_size_blocks  => 1072
segment_size_bytes   => 8781824
used_blocks          => 601
used_bytes           => 4923392
expired_blocks       => 448
expired_bytes        => 3670016
unexpired_blocks     => 0
unexpired_bytes      => 0

```

и после дедупликации:

```

Segment Name=SYS_LOB0000070763C00004$$
segment_size_blocks  => 1456
segment_size_bytes   => 11927552
used_blocks          => 7
used_bytes           => 57344
expired_blocks       => 127
expired_bytes        => 1040384
unexpired_blocks     => 1296
unexpired_bytes      => 10616832

```

Из приведенных достаточно изучить только одну метрику used_bytes, которая показывает точное количество байтов, занимаемых LOB-столбцом. До дедупликации он занимал 4,923,392 байтов, или около 5 Мбайт, а уменьшился до 57,344 байт, что около 57 Кбайт, то есть всего лишь около одного процента от исходного размера. Так получилось потому, что процесс дедупликации 100 раз нашел строки с одним и тем же значением (помните, мы в LOB-столбце всех строки поместили одно и тоже значение) и сохранил его только в одной строке, а в остальных – только указатели.

Можно обратить результат дедупликации:

```

SQL> alter table contracts_sec
  2 modify lob(orig_file)
  3 (keep_duplicates)
  4 /

```

Table altered.

Посмотрим после этого снова на занимаемое место:

```

Segment Name=SYS_LOB0000070763C00004$$
segment_size_blocks  => 1456
segment_size_bytes   => 11927552
used_blocks          => 601
used_bytes           => 4923392
expired_blocks       => 0
expired_bytes        => 0
unexpired_blocks     => 829
unexpired_bytes      => 6791168

```

Мы видим, что значение USED_BYTES выросло до исходной величины около 5 Мбайт.

Сжатие

Другой возможностью SecureFiles является сжатие. Можно сжимать значения, сохраненные в LOB-объектах, используя следующий SQL:

```
SQL> alter table contracts_sec
2 modify lob(orig_file)
3 (compress high)
4 /
```

Table altered.

Сейчас, если запустить блок PL/SQL, вычисляющий объем:

```
Segment Name=SYS_LOB0000070763C00004$$
segment_size_blocks => 1456
segment_size_bytes => 11927552
used_blocks => 201
used_bytes => 1646592
expired_blocks => 0
expired_bytes => 0
unexpired_blocks => 1229
unexpired_bytes => 10067968
```

то увидим, что значение used_bytes сейчас 1,646,592, или около 1,5 MB, что существенно меньше 5 MB.

Сжатие отличается от дедупликации. Сжатие происходит внутри LOB- столбца, в строке – каждый LOB-объект сжимается независимо. При дедупликации проверяются все строки, и повторные значения удаляются и заменяются указателями. Если есть две существенно отличающиеся строки, дедупликация не уменьшит занимаемый размер, а сжатие может оптимизировать место занимаемое LOB-объектами. Можно как сжимать, так и дедуплицировать данные.

Сжатие требует работы CPU, поэтому в зависимости от количества сжимаемых данных, сжатие может терять смысл. Например, имеется много изображений в формате JPEG, которые уже сжаты, поэтому дальнейшее сжатие не уменьшит занимаемое место. С другой стороны, если CLOB-объектом является XML-документ, то сжатие может получиться существенным. Процесс SecureFiles-сжатия автоматически определяет, сжимаются ли данные или только расходуется процессорное время.

Индексы Oracle Text могут быть созданы как LOB-ы SecureFiles. Это главное преимущество хранения неструктурированных данных в базе данных Oracle по сравнению сжатыми файлами файловой системы.

Так же отметим, что сжатие LOB-объектов не зависит от сжатия таблиц. Если применяется таблицы CONTRACTS_SEC, LOB-объекты не сжимаются. Сжатие LOB-объектов будет происходить только при использовании приведенного SQL.

Шифрование

Для столбца с SecureFiles, как для любого столбца, можно использовать механизм прозрачного шифрования (Transparent Database Encryption). Ниже показано, как зашифровать LOB-столбец orig_file с использованием 128-битного AES-шифрования.

```
alter table contracts_sec
modify lob(orig_file)
(encrypt using 'AES128')
```

/

До шифрования необходимо установить крипто-блокнот (encryption wallet). (Полное описание encryption wallet можно найти в Oracle Magazine в моей статье «Transparent Data Encryption» (<http://www.oracle.com/technology/ora-mag/oracle/05-sep/o55security.html>)). Кратко перечислим основные действия:

1. Установить параметр в sqlnet.ora, если еще не установлен, определяющего расположение крипто-блокнота:

```
ENCRYPTION_WALLET_LOCATION=
(SOURCE=
(METHOD=FILE)
(METHOD_DATA=
(DIRECTORY= /opt/oracle/orawall)
)
)
```

Директория /opt/oracle/orawall уже должна существовать, в противном случае ее необходимо создать.

2. Создать крипто-блокнот:

```
alter system set encryption key authenticated by "mypass"
```

Это предложение создает крипто-блокнот с паролем mypass и открывает его.

3. После того, как крипто-блокнот создан и открыт, он остается открытым, пока работает база данных (до тех пор, пока она полностью не завершит работу). Если база данных перезапускается, необходимо открыть wallet предложением:

```
alter system set encryption wallet open identified by "mypass"
```

Когда шифруется LOB-столбец о SecureFile, то шифруются все элементы столбца в этой таблице. После шифрования нельзя использовать обычный (Conventional) экспорт или импорт таблицы, а необходимо использовать Data Pump.

Можно посмотреть представление dba_encrypted_columns, чтобы увидеть, какие и как столбцы зашифрованы.

```
SQL> select table_name, column_name, encryption_alg
2 from dba_encrypted_columns
3 /
```

TABLE_NAME	COLUMN_NAME	ENCRYPTION_ALG
CONTRACTS_SEC	ORIG_FILE	AES 128 bits key

Кэширование

Возможность кэширования – одно из преимуществ хранения неструктурированных данных в файлах операционной системы, а не во внутренних объектах базы данных. Файлы могут кэшироваться в файловых буферах операционной системы. Внутренние объекты базы данных могут кэшироваться базой данных. Тем не менее, в некоторых случаях кэширование может снизить производительность. LOB-объекты обычно очень велики (отсюда и название Large OBjects) и, если они попадают в кэш, то большинство количество других блоков будут оттуда

вытеснены, чтобы дать место поступившему LOB-объекту. LOB-объект, возможно, и не будет потом использоваться, но при его записи в кэш используемые блоки оттуда будут удалены. Поэтому в большинстве случаев следует, вероятно, отключить кэширование LOB-объектов.

В приведенном примере для CONTRACTS_SEC использовалась фраза noscache для отключения кэширования. Для включения кэширования следует изменить таблицу:

```
alter table contracts_sec
modify lob(orig_file)
(cache)
/
```

Это действие включает кэширование LOB-объектов. Отметим, что это кэширование относится только к LOB-объектам. Оставшаяся часть таблицы кэшируется по тем же правилам, что и любая другая таблица, вне зависимости от того, установлено ли кэширование LOB-объектов таблицы, или нет.

Преимущества от кэширования сильно зависят от приложений. В приложении, работающем с мелкими рисунками, производительность, возможно, при кэшировании увеличится. Тем не менее, для больших документов или рисунков кэширование лучше отключить. Securefiles позволяют это контролировать.

Журналирование

Журналирование определяет, как изменения данных в LOB-объектах записываются в поток redo-журнала. По умолчанию [для них] установлено полное журналирование, как для всех остальных данных. Но, поскольку данные в LOB-объектах обычно велики, вероятно, захочется отключить журналирование в некоторых случаях. Фраза NOLOGING, использованная в предыдущем примере, делает именно это.

Для SecureFiles, как показано ниже, имеется еще одно значение для этой фразы - filesystem_like_logging:

```
create table contracts_sec_fs
(
  contract_id number(12),
  contract_name varchar2(80),
  file_size number,
  orig_file blob
)
tablespace users
lob (orig_file)
store as securefile
(
  tablespace users
  enable storage in row
  chunk 4096
  pctversion 20
  nocache
  filesystem_like_logging
)
```

Отметим, что выделенная жирным шрифтом строка, включает ведение redo-журнала для метаданных LOB-объектов, но не для LOB-объектов. Это похоже на

файловую систему. Метаданные файлов записываются в журналы файловой системы. Эта возможность управляет восстановлением после сбоев.

Администрирование

Представление DBA_LOBS словаря данных показывает свойства LOB-объектов базы данных, включая Secure-Files. Вот столбцы этого представления:

Column Name	Description
OWNER	Владелец таблицы
TABLE_NAME	Имя таблицы
COLUMN_NAME	Имя столбца, содержащего LOB-объекты
SEGMENT_NAME	LOB-объекты хранятся в отдельном сегменте, поименованном пользователем или по умолчанию как SYS_LOB...
TABLESPACE_NAME	Имя табличного пространства
INDEX_NAME	Имя LOB-индекса
CHUNK	Размер порции LOB-объекта
PCTVERSION	Не используется для Secure-Files
RETENTION	Если обновляется LOB типа SecureFile, то предыдущий образ, как любой другой блок базы данных, сохраняется в сегменте отката; но в отличие от блоков базы данных, можно установить, как долго хранить предыдущие образы (сохранность).
FREEPOOLS	Не используется SecureFiles
CACHE	Кэшируется или нет LOB типа SecureFile в буферном пуле (Yes/No).
LOGGING	Журналируются или нет изменения LOB типа Secure-File (Yes/No).
ENCRYPT	Шифруется или нет LOB типа SecureFile (Yes/No).
COMPRESSION	Сжимается или нет LOB типа SecureFile (Yes/No).
DEDUPLICATION	Используется или нет дедупликация LOB типа SecureFile (Yes/No).
IN_ROW	Хранятся ли LOB-объекты в строке (inline) таблицы.
FORMAT	LOB зависит от порядка байтов, принятого для платформы
PARTITIONED	Входит ли LOB-столбец в секционированную таблицу
SECUREFILE	Являются ли LOB-объект SECUREFILE (Yes or No) или BASICFILE

В случае секционированных таблиц информация о LOB-ах хранится в представлении DBA_LOB_PARTITIONS.

Миграция LOB в SecureFiles

Теперь, увидев как полезны SecureFiles, вы, возможно, захотите преобразовать уже существующие таблицы. Простейший способ сделать это – создать новую таблицу, загрузить туда данные из старой и переименовать ее. (Конечно, требуется на время сделать обе таблицы недоступными для операций.) Другой способ – использовать пакет `dbms_redefinition` для переопределения таблиц на лету, не ограничивая к ним доступность.

Давайте посмотрим на примере, как это делается. Предположим, вы хотите преобразовать исходную таблицу `CONTRACTS_BASIC`, чтобы хранить в ней SecureFiles. Это делается следующим образом:

1. Убедитесь в наличие первичного ключа. Если его нет, то создайте его.

```
alter table contracts_basic
add constraint pk_contacts
primary key (contract_id)
/
```

2. Создайте новую таблицу

```
create table contracts_new
(
contract_id number(12),
contract_name varchar2(80),
file_size number,
orig_file BLOB
)
lob (orig_file)
store as securefile
(nocache nologging)
/
```

3. Установите связи со столбцами новой таблицы

```
declare
l_col_mapping varchar2(1000);
begin
l_col_mapping :=
'contract_id contract_id , '||
'contract_name contract_name , '||
'file_size file_size , '||
'orig_file orig_file';
dbms_redefinition.start_redef_table
('ARUP', 'CONTRACTS_BASIC',
'CONTRACTS_NEW', l_col_mapping);
end;
/
```

4. Запустите процесс преобразования

```
declare
l_error_count pls_integer := 0;
begin
dbms_redefinition.copy_table_dependents
(
'ARUP', 'CONTRACTS_BASIC',
'CONTRACTS_NEW',
```

```
l, TRUE, TRUE, TRUE, FALSE, l_error_count
);
dbms_output.put_line('Errors Occurred := ' ||
to_char(l_error_count));
end;
/
```

Все строки `CONTRACTS_BASIC` копируются в `CONTRACTS_NEW`. В зависимости от их количества этот процесс может занять много времени.

5. Завершите процесс переименования:

```
begin
dbms_redefinition.finish_redef_table
('ARUP', 'CONTRACTS_BASIC',
'CONTRACTS_NEW');
end;
/
```

6. Убедитесь, что таблица преобразована.

```
select securefile
from dba_lobs
where table_name = 'CONTRACTS_BASIC'
/
```

```
SEC
```

```
---
```

```
YES
```

Значение YES, что означает, что столбец преобразован в SecureFiles.

7. Удалите промежуточную таблицу `CONTRACTS_NEW`.

```
SQL> drop table contracts_new;
Table dropped.
```

Чтобы ускорить процесс копирования, можно попытаться включить параллельное выполнение этой DML-операции. Включение возможности параллельного DML делается так:

```
alter session force parallel dml;
```

Параметр инициализации

Параметр инициализации `db_securefile` определяет использование SecureFiles в базе данных. Ниже приводятся его возможные значения и описания.

Значение	Описание
PERMITTED	Значение по умолчанию; показывает, что в базе данных могут быть созданы LOB-объекты SecureFiles
ALWAYS	Теперь, когда показано, насколько удобны SecureFiles, вам, вероятно, захочется, чтобы в дальнейшем все LOB-объекты были только SecureFiles, а не принятыми по умолчанию BasicFiles, даже если пользователь не определит securefile. Это значение параметра гарантирует, что по умолчанию все создаваемые LOB-ы будут SecureFiles. Помните, что SecureFiles требуют наличия качества ASSM для табличной области (это значение принято по умолчанию в 11g). Поэтому попытка создания LOB-объектов в табличном пространстве с выключенным ASSM приведет к ошибке.
NEVER	Это значение прямо противоположно ALWAYS. По каким-то причинам не требуется применение SecureFiles, и надо не допустить их создания в базе данных. При этом значении параметра db_securefile LOB-объекты будут создаваться только как BasicFiles, даже если есть фраза SecureFiles. В этом случае пользователю не посылается сообщение об ошибке, но LOB-объект создается как BasicFiles.
IGNORE	Фраза securefile вместе с параметрами хранения игнорируется.

Заключение

SecureFiles – это не просто новое поколение LOB-объектов; они обладают гораздо большими возможностями, особенно теми, которые раньше были прерогативой файловых систем. SecureFiles могут быть зашифрованы в целях безопасности, дедуплицированы и сжаты для более эффективного хранения, кэшированы (или нет) для более быстрого доступа (или для экономии пространства буферного пула), и журналированы на нескольких уровнях, чтобы уменьшить среднее время восстановления после аварии. Применяя SecureFiles можно хранить в базе данных большее число неструктурированных документов без издержек и потерь каких-либо важных функциональных возможностей, которые предоставляются файловыми системами ОС.