



Дискретные и автономные транзакции

(начиная с версии Oracle8i)

Дискретные транзакции

Дискретные транзакции (DISCRETE TRANSACTION) – это транзакции, для которых не генерируется информация отката (rollback segment).

Они работают в условиях некоторых ограничений, а именно:

- не должны модифицировать один блок более одного раза за одну транзакцию;
- модифицируют небольшое количество блоков базы данных;
- не модифицируют данные, участвующие в долго идущих запросах;
- не опрашивают новых значений данных, после того как обновили эти данные;
- не участвуют в распределенных транзакциях;
- не могут выполнять вставки и обновления сразу в обе таблицы, вовлеченные в ограничение ссылочной целостности;
- не модифицируют таблиц, содержащих значения типа LONG.

Дискретные транзакции могут использоваться одновременно со стандартными транзакциями.

Как работают дискретные транзакции

1. В течение дискретной транзакции все изменения, которые она вносит в любые данные, откладываются до момента завершения этой транзакции.
2. Информация повторения генерируется, но сохраняется в отдельной области памяти.
3. Когда такая транзакция выдает запрос `commit`, ее информация повторения переписывается в файл журнала транзакций (вместе с другими групповыми операциями `commit`), а изменения в блоке базы данных применяются непосредственно к блоку.
4. Модифицированный блок записывается в файл базы данных обычным порядком.
5. Управление возвращается в приложение после завершения операции `commit`.

Синтаксис создания дискретной транзакции

Дискретная транзакция запускается с помощью процедуры `BEGIN_DISCRETE_TRANSACTION`.

Процедура `BEGIN_DISCRETE_TRANSACTION` должна быть вызвана перед первым предложением в транзакции. Вызов этой процедуры действует лишь на протяжении одной транзакции.

При этом параметр инициализации `DISCRETE_TRANSACTIONS_ENABLED` должен быть установлен в `TRUE`.

Если этот параметр установлен в `FALSE`, то все обращения к процедуре `BEGIN_DISCRETE_TRANSACTION` игнорируются, и транзакции, запрашивающие эту службу, обрабатываются как обычные транзакции.

Любые ошибки, встреченные во время обработки дискретной транзакции, вызывают возбуждение predefined исключения `DISCRETE_TRANSACTION_FAILED`.

Пример дискретной транзакции

Пример транзакции, использующей `BEGIN_DISCRETE_TRANSACTION`, дает приложение, которое выдает библиотечные книги. Это приложение вызывает приведенную ниже процедуру, используя в качестве аргумента номер книги. Эта процедура проверяет, не заказана ли данная книга кем-либо другим, прежде чем выдавать ее. Если заказано (зарезервировано) больше копий книги, чем имеется в наличии, то библиотечному приложению возвращается статус `RES`, и приложение может, если угодно, заказать эту книгу, обратившись к другой процедуре. В противном случае книга выдается, и комплект наличных книг соответственно обновляется.

```
CREATE PROCEDURE checkout (bookno IN NUMBER (10), status OUT
VARCHAR(5)) AS
  DECLARE tot_books NUMBER(3);
  checked_out NUMBER(3);
  res NUMBER(3);
```

Продолжение примера

```
BEGIN
  dbms_transaction.begin_discrete_transaction;
  FOR i IN 1 .. 2 LOOP    -- вторая транзакция не будет дискретной
  BEGIN
    SELECT total, num_out, num_res INTO tot_books, checked_out, res
      FROM books
      WHERE book_num = bookno FOR UPDATE;
    IF res >= (tot_books - checked_out) THEN status := 'RES';
    ELSE UPDATE books SET num_out = checked_out + 1
      WHERE book_num = bookno;
      status := 'AVAIL';
    ENDIF;
    COMMIT;
    EXIT;
  EXCEPTION WHEN dbms_transaction.discrete_transaction_failed
    THEN ROLLBACK;
  END;
  END LOOP;
END;
```

Замечания по использованию дискретных транзакций

1. Хотя дискретные транзакции не могут видеть своих собственных изменений, можно получить старое значение и заблокировать строку, используя фразу FOR UPDATE предложения SELECT, прежде чем обновлять это значение.
2. Дискретные транзакции не могут выполнять вставки и обновления сразу в обе таблицы, вовлеченные в ограничение ссылочной целостности.
3. Так как дискретные транзакции могут обновлять каждый блок базы данных лишь один раз, некоторые комбинации предложений манипулирования данными по одной и той же таблице лучше подходят для таких транзакций, чем иные комбинации:
 - INSERT + UPDATE, INSERT + DELETE – подходит;
 - INSERT + INSERT – не подходит.
 - Операции DML, выполняемые на разных таблицах, не затрагивают одних и тех же блоков базы данных, если эти таблицы не кластеризованы.

Автономные транзакции

Автономные транзакции (AUTONOMOUS_TRANSACTION) позволяют создавать новые подтранзакции (subtransaction), которые можно сохранять или отменять изменения вне зависимости от родительской транзакции.

До версии Oracle8i поддерживались внутренние автономные транзакции – рекурсивные SQL-операции, такие как:

- Выбор из некешируемой последовательности. При этом выполняется рекурсивная транзакция для немедленного увеличения последовательности. Это обновление последовательности сразу же сохраняется, и становится видимым для других транзакций, при этом для всей транзакции сохранение еще не выполнялось. Откат транзакции не отменит увеличение последовательности.
- Управление памятью и другие внутренние операции выполняются аналогичным рекурсивным способом.

Пример 1. Автономная транзакция

```
create table t ( x int );
create or replace procedure insert_into_t
as
pragma autonomous_transaction;
begin
insert into t values ( 1 );
commit;
end;
/
begin
insert into t values ( -1 );
insert_into_t;
rollback;
end;
/
select * from t;
X
-----
1
```

Пример 2. Аудит, который нельзя откатить

-- Создание таблицы, в которую будет записываться аудит

```
create table audit_tab (  
    uname varchar2(30),  
    dt      date,  
    msg     varchar2(4000));  
create or replace trigger emp_trigger  
before update of SAL on emp  
for each row  
declare  
pragma autonomous_transaction;  
l_cnt number;  
begin
```

/*Следующий запрос проверяет, действительно ли работник, данные о котором меняются, подчиняется сотруднику, выполняющему обновление. Для построения иерархии удобно использовать конструкцию connect by. Поскольку предложение where обрабатывается после того, как иерархия построена, здесь можно использовать exists */

Продолжение примера 2.

```
select count(*) into l_cnt
from dual
where exists ( select empno from emp
               where empno = :new.empno
               start with mgr = (select empno from emp where ename=USER)
               connect by prior empno = mgr );
/* Если exists ничего не возвращает, значит происходит попытка
обновить данные о работнике, который не является подчинённым */
if ( l_cnt = 0 ) then
    insert into audit_tab values ( user, sysdate,
    'Попытка обновления зарплаты ' ||
    :new.ename || '-' || :new.empno);
commit;
raise_application_error( -20001, 'Вы пытаетесь сделать то, что вы не
имеете права делать, и мы знаем об этом');
end if;
end;
/
```

Пример. Выполнение DDL в триггерах

Постановка задачи: триггер фиксирует изменения, произведенные в основной таблице, причем эти изменения должны каждый день фиксироваться в новой таблице с именем archive_date, где date – это текущая дата в формате 'yyyymmdd'.

```
create or replace trigger arch
  before delete or update on tab
  for each row
  declare
    pragma autonomous_transaction;
    tablename char(16);
    str_ins varchar(100);
    str_crt varchar(100);
begin
  tablename := 'archive_' || to_char(sysdate, 'yyyymmdd');
  str_ins := 'insert into ' || tablename || ' values(:f1, :f2, :f3)';
```

Продолжение примера

```
begin
  execute immediate str_ins USING :old.f1, :old.f2, :old.f3;
  exit;
exception when OTHERS then
  str_crt := 'create table ' || tablename || ' (f1 number(6),
    f2 varchar(50), f3 number(8,2));';
  execute immediate str USING :old.f1, :old.f2, :old.f3;

end;
execute immediate str_ins USING :old.f1, :old.f2, :old.f3;

end;
/
```

Регистрация команды SELECT

- Выполнение команды SELECT регистрируется в специальной таблице.
- Функция выполняет регистрацию в автономной транзакции.

```
create or replace function f_aud return integer as
  pragma autonomous_transaction;
begin
  insert into ref values(10,substr(user,1,20), sysdate);
  commit;
  return 1;
end;
/
```

- Доступ к таблице осуществляется ТОЛЬКО через представление.

```
create or replace view ve as select c.*
  from clients c, (select f_aud f from dual) d
  where rownum<11 and f>0;
```

Регистрируется выполнение команды независимо от того, сколько строк она затронула.