“Fire and Forget”: When to Use Autonomous Transactions

Michael Rosenblum
Dulcian, Inc.
www.dulcian.com
Autonomous Transactions

**Definition:**
- Autonomous transactions are independent transactions that can be called from within another transaction.

**Syntax:**

```plaintext
declare
    Pragma autonomous_transaction;
Begin
    ....
    commit;
End
```
Specifications

An autonomous transaction allows you to:

- Leave the context of the calling transaction (parent)
- Perform SQL operations
- Commit or rollback those operations
- Return to the calling transaction's context
- Continue with the parent transaction
Syntax

Can be used:

- Top-level anonymous blocks
- Local, standalone or packaged functions and procedures
- Methods of object types
- Database triggers
Syntax

Cannot be used:

- Outside of a declaration section
- Within the declaration section of a nested block (a block within a block)
- In a package specification
- In a package body outside of a procedure or function definition
- In a type body outside of a method definition
Possible uses

- **Security**
  - Audit of querying
  - Audit of failed activities

- **Modular code**
  - Environmental consistency
  - Structural optimization

- **Non-standard PL/SQL cases**
  - DDL in triggers
  - SELECT-only environment
  - Avoid self-mutation
procedure f_log_update
(v_user varchar2,
 v_table varchar2,
 v_field varchar2)
Is
pragma autonomous_transaction;
Begin
insert into audit_empvalues (v_user, sysdate, 'Update');
commit;
insert into audit_table values (v_table, sysdate, v_field);
commit;
End;

Simple example of autonomous transaction

Begin

f_log_update (user, 'EMP','SAL');
Savepoint A;

update emp
 set sal = sal*1.5;
Exception
 when others
 then
 rollback to
     Savepoint A;
 raise;
End;
Part 1

Autonomous Vs. Nested transactions
Autonomous vs. Nested Transactions

- Autonomous transaction does not share transactional resources (such as locks) with the main transaction.
- Autonomous transaction does not depend on the main transaction.
- Non-committed changes of parent transaction are not visible to autonomous transactions immediately, but visible for nested ones.
- Changes made by autonomous transaction may or may not be visible by parent one depending upon the isolation level. Changes made by nested transactions are always visible by parent one.
- Exceptions raised in an autonomous transaction cause a transaction-level rollback, not a statement-level rollback.
Scope of Transaction

**Scope** – The ability to see values of various things within the database.

- Variables
- Session settings/parameters
- Data changes
- Locks
Object of interest:
- Transactional resources (Locks)

Rule:
- Autonomous transaction does not share transactional resources (such as locks) with the main transaction.
Autonomous vs. Nested #1
Example Locks

Session: scott/tiger@ora817

```sql
declare
    v varchar2(2000);
begin
    select ename into v
    from emp
    where ename = 'SCOTT'
    for update;

    lock_test;

    commit;
End;
```

```sql
procedure lock_test is
    v varchar2(2000);
begin
    select ename into v
    from emp
    where ename = 'SCOTT'
    for update;
end;
```

```sql
procedure lock_test is
    v varchar2(2000);
    pragma autonomous_transaction;
begin
    select ename into v
    from emp
    where ename = 'SCOTT'
    for update;
    commit;
end;
```
Object of interest:
- Session resources (Variables)

Rule:
- Autonomous transaction does not depend on the main transaction. It belongs to the same session.
package var_test
As
  global_nr number := 0;
end;

procedure p_var_test
(v_nr number) is
  pragma
  autonomous_transaction;
Begin
  dbms_output.put_line(' Before Auto value: '|| var_test.global_nr);
  var_test.global_tx := v_nr;
  dbms_output.put_line(' After auto value: '|| var_test.global_nr);
End;

Session: scott/tiger@ora817
Object of interest:
- Data changes in parent session

Rule:
- Non-committed changes of parent transaction are not visible to autonomous transactions immediately, but are visible for nested ones.
Autonomous vs. Nested #3
Parent Data Changes

```
declare
  v_nr number;
begin
  select count(1) into v_nr
  from audit_emp;
  insert into audit_emp
  values (user, sysdate, 'Test');
  dbms_output.put_line
  ('Count#1='||v_nr);
  data_change_test;
  data_change_auto_test;
end;
```

```
procedure data_change_test is
  v_nr number;
begin
  select count(1) into v_nr
  from audit_emp
  dbms_output.put_line
  ('Count#2='||v_nr);
end;
```

```
procedure data_change_test is
  v_nr number;
pragma autonomous_transaction;
begin
  select count(1) into v_nr
  from audit_emp
  dbms_output.put_line
  ('Count#3='||v_nr);
  commit;
end;
```
Object of interest:
- Data changes in child session

Rule:
- Changes made by autonomous transactions may or may not be visible to the parent one depending upon the isolation level.
- Changes made by nested transactions are always visible by the parent one.
declare
    v_nr number;
Begin
    set transaction isolation level serializable;
    insert into audit_emp values (user, sysdate, 'Test', 1);
    commit_test;
    select max(log_id) into v_nr from audit_emp;
    dbms_output.put_line ('Maximum=' || v_nr);
end;

procedure commit_test
is
    pragma autonomous_transaction;
begin
    insert into audit_emp values (user, sysdate, 'Test', 2);
    commit;
end;
Object of interest:

Exception

Rule:

Exceptions raised in an autonomous transaction cause a transaction-level rollback, not a statement-level rollback.
Autonomous vs. Nested #5: Exceptions

procedure rollback_test is
begin
    insert into audit_emp values (user, sysdate, 'Test', 1);
    insert into audit_emp values (user, sysdate, 'Test', 0);
    commit;
end;

1

procedure rollback_test is
    v_nr number;
begin
    rollback_test;
    Exception
    when others
    then
        select count(1) into v_nr
        from audit_emp;
        dbms_output.put_line ('Count=' || v_nr);
    end;
end;

2

procedure rollback_test is
    pragma autonomous_transaction;
begin
    insert into audit_emp values (user, sysdate, 'Test', 2);
    insert into audit_emp values (user, sysdate, 'Test', 'Wrong datatype');
end;

procedure rollback_test is
    pragma autonomous_transaction;
begin
    insert into audit_emp values (user, sysdate, 'Test', 1);
    insert into audit_emp values (user, sysdate, 'Test', 'Wrong datatype');
    commit;
end;

Declare
    v_nr number;
Begin
    rollback_test;
Exception
when others
then
    select count(1) into v_nr
    from audit_emp;
    dbms_output.put_line ('Count=' || v_nr);
end;

Autonomous Vs Nested:
Mission critical

- Transactional resources
- Session-level resources
- Data changes of parent transaction
- Data changes of autonomous transaction
- Exceptions
Part 2

How to use autonomous transaction?
Usage #1: Audit of querying

Object of interest:
◆ Audit of querying

Business rule:
◆ Each user requesting a view of the Salary column should be recorded.
Create or replace view v_emp
As
Select empno,
    ename,
    audit.record(empno, 'sal', sal) sal
From emp

package body audit as
  function record (v_nr number, 
                   v_tx varchar2, 
                   v_value_nr number)
  return number is
    pragma 
    autonomous_transaction;
    begin
      insert into audit_emp 
      values (user, sysdate, 
              v_nr||':'||v_tx 
              ||'='||v_value_nr, 
              log_seq.nextval);
      commit;
      return v_value_nr;
    end;
End;
Usage #1a: Extended query

Object of interest:
- Expended audit of querying

Business rule:
- User can query specific data only once per session
- Temporary data is created each time a session begins
Usage #1a: Example

Create or replace view v_emp
As
Select empno, ename,
clean.record(empno, sal)
sal
From temp_emp

package body clean as

function record
(v_id number, v_nr number)
return number
is
pragma
autonomous_transaction;
begin
delete from temp_emp
where empno = v_id;
commit;
return v_nr;
end;
End;

Select empno, ename, sal
From v_emp

Delete from emp_emp

Select ... From temp_emp
Usage #2: Audit of activity

Object of interest:
- Audit of activities

Business rule:
- User-executed update on any salary should be recorded, even if the update failed
Usage #2: Example

Trigger emp_audit
Before update on emp
For each row
Declare
  pragma autonomous_transaction;
Begin
  if (check_privileges
      (user,:new.empno))
  then
    audit.f_log (user,'Update #' ||:new.empno,'OK');
    commit;
  else
    audit.f_log (user,'Update #' ||:ew.empno,'FAILED');
    commit;
    raise_application_error (-2001, 'Access denied!');
  end if;
End;

Function check_privileges
  (v_tx varchar2,
   v_empno_nr number)
Return boolean is
  pragma autonomous_transaction;
  v_nr number := 0;
Begin
  select count(1) into v_flag_nr from dual
  where exists (select 1
               from emp
               where empno = v_empno_nr
               and mgr = (select empno
                           from emp
                           where ename = v_tx)
               from emp
               where ename = v_tx);
  commit;
  if v_nr = 0 then return FALSE;
  else return TRUE;
  end if;
End;
Usage #3: Structural optimization

Object of interest:
- Error handling in the complex environment

Business rule:
- Failure of defined subprograms should not stop execution of the whole script.
Usage #3: Example

```sql
Declare
    v_mail MailList;
Begin
    v_mail := GetList(sysdate);
    create_msg(v_mail);
    ... ... ...
End;
```

```sql
Procedure create_msg (v_in MailList)
Return varchar2 is
    pragma autonomous_transaction;
Begin
    for i in v_in.First..v_in.Last loop
        add_message (v_in(i).Address);
    end loop;
    commit;
Exception
    when others then rollback;
End;
```

```sql
Declare
    v_mail MailList;
Begin
    v_mail := GetList(sysdate);
    begin
        Savepoint A;
        create_msg(v_mail);
        exception
            when others
                Rollback to Savepoint A;
    end;
End;
```

```sql
Procedure create_msg (v_in MailList)
Return varchar2 is
Begin
    for i in v_in.First..v_in.Last loop
        add_message (v_in(i).Address);
    end loop;
End;
```
Usage #4: Consistency of environment

Object of interest:
◆ Actions forced by commit

Business rule:
◆ Commit of changes in subroutine should not force any activity in other routines.
Create table A
    (a number primary key);

Create table B (a number,
    b number);

Alter table B
    add constraint a_fk
    foreign key (a)
    references A(a) deferrable
    initially deferred;

Procedure copy_a (v_dt date)
Is
    pragma autonomous_transaction;
Begin
    execute immediate
        'create table a_copy_
           ||to_char (sysdate,'ddmmmyyy')
           ||' as select * from a@link';
    commit;
End;
Usage #5: DDL in triggers

Object of interest:
- DDL in triggers

Business rule:
- Insertion of record in the view causes creation of the new column in other table.
Create or replace trigger u.uml_attrib
Instead of Insert on uml_attrib
For each row
Declare
pragma autonomous_transaction;
Begin
if check(:new.attrib_cd)='Y'
then
execute immediate
  ' alter table '||:new.class_cd
|| ' add column '||:new.attrib_cd
|| ' '||:new.datatype;
commit;
End;
Usage #6: SELECT-only environment

Object of interest:
◆ Extended activity
  in SELECT-only environment

Business rule:
◆ User needs to execute some code while tools allow only SELECT.
Create or replace view v_log
As
Select start_session (sysdate, user) flag
From dual

function start_session (sysdate date,
user varchar2)
return varchar2
is
pragma autonomous_transaction;
Begin
log_user (user, sysdate);
set_system_defaults;
populate_temp(sysdate, user);
commit;
return 'Y'
Exception
when others return 'N';
End;

Select *
From v_log

Log user
Create and populate temporary tables
Set default environment
Usage #7: Avoid self-mutation

Object of interest:
- Self-mutation

Business rule:
- Rule for UPDATE is based on the same column that is updated:
  “Average salary of employees cannot be less than half of the maximum salary in their department.”
create type emp_t as object
  (empno number,
   deptno number,
   old_sal number,
   new_sal number);

create type emp_tt as table of emp_t;

create package obj
as
  emp_temp emp_tt := emp_tt();
end;

Create or replace trigger BU_EMP
before update on EMP
begin
  obj.emp_temp.delete;
end;

Create or replace trigger BU_EMP_ROW
before update on EMP
for each row
Begin
  obj.emp_temp.extend;
  obj.emp_temp(obj.emp_temp.last) := emp_t (:new.empno,
                                           :new.deptno,
                                           :old.sal,
                                           :new.sal);
End;
Usage #7: Example (page 2)

Create or replace trigger AU_EMP
After update on EMP
    pragma autonomous_transaction;

cursor cDept
is
select t.deptno,
    sum(t.new_sal) -
    sum(t.old_sal) DeptDif,
    max(new_sal) MaxDif
from table (cast (obj.emp_temp as emp_tt) ) t
    group by t.deptno;

v_max number;
v_avg number;
v_count number;

Begin
    for cD in cDept
        loop
            select max(sal), avg(sal), count(1) into v_max, v_avg, v_count
                from emp
                where deptno = cd.Deptno;

            if (greatest (v_max, cd.MaxDif)/2) > ((v_avg*v_count +
                cd.DeptDif)/v_count)
                then
                raise_application_error
                    (-20001, 'Rule Violated!');
                end if;
        end loop;
    commit;
End;
Autonomous transactions are:

- A powerful tool that allows us to solve old problems in a new way.
- A complex tool requiring a high level of familiarity with the Oracle DBMS.
- A sensitive tool that may cause unexpected (even catastrophic) results if used improperly.
Contact Information

Michael Rosenblum
mrosenblum@dulcian.com
www.dulcian.com
(732) 744-1116