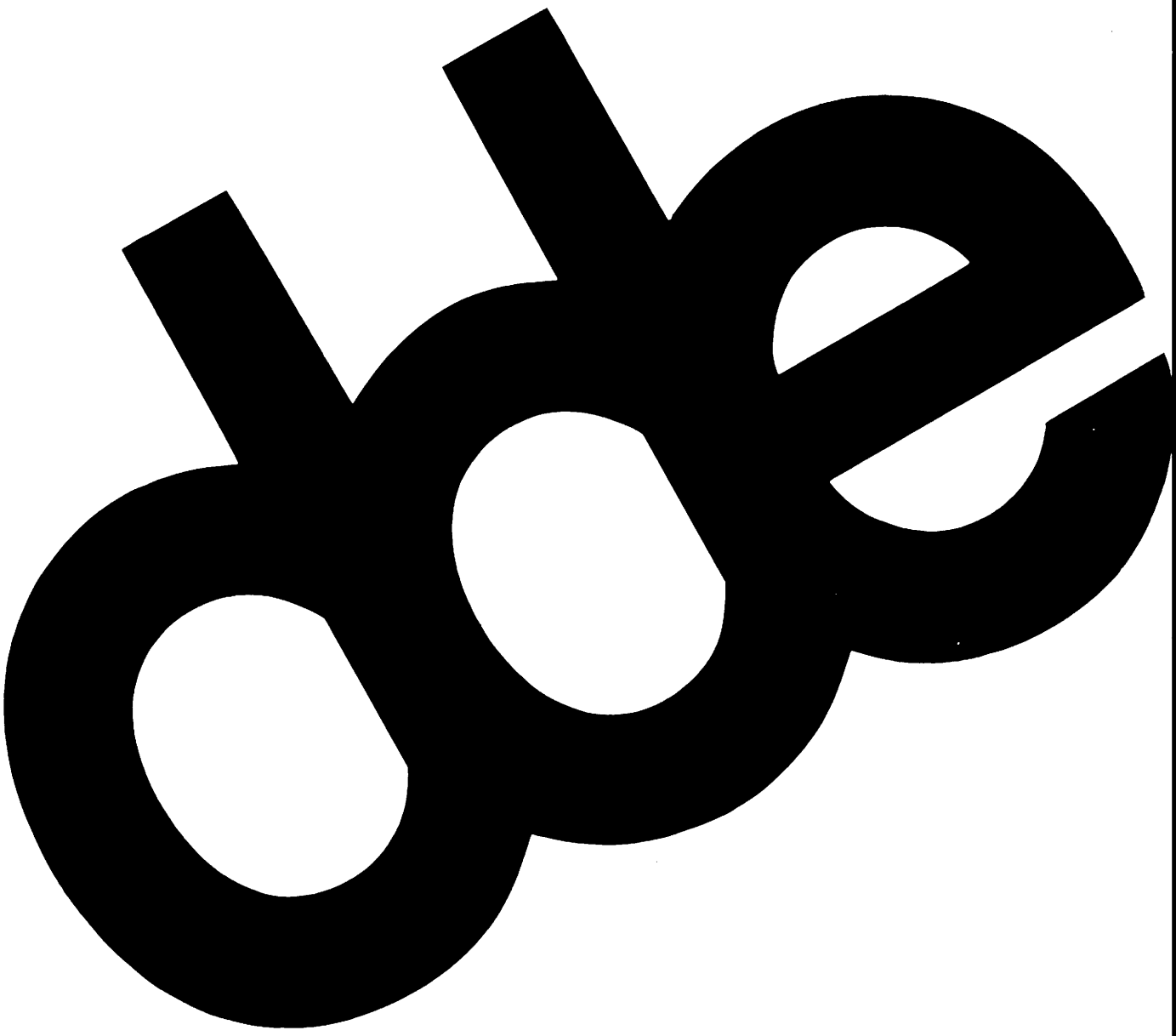
Oracle 7 Hints



22/5 96

Development Environments Using Oracle7

The present paper is written as a contribution to the debate about which ingredients are applicable to a development environment in order to develop good and robust bug fixed programs.

A vertical bar (|) in the margin shows that information has changed (or is added) since the last major revision (1.1.50.1).

A minor advice in Alerts are suplues in version 1.1.52. And some observations on the date 29. Feb. 2000 are stated. And spelling errors corrected in 1.1.53 to 1.1.55.

In version 1.1.56 a minor hint on ORA_NLS environment is given. And some information on the Object oriented directions of the Oracle rdbms.

In version 1.1.58 a hint on sorting character columns with respect to national characters are presented, and comments on the Replication Option from version 7.2 is given.

In version 1.1.59 small corrections are inserted. Getting ORACLE_SID from the database. A trick on how redundancy on interval information may be managed. And more on space usage from 7.2.

In version 1.1.60 a little enhancement on the interval example has been made. A hint on User defined SQL Functions with side-effect using local memory.

1. Objective

The objective is to make as good use of the features in Oracle7 as possible, and prevent developers from falling into the same traps as others did before them.

- Oracle, Oracle7, PL/SQL, SQL*Forms, SQL*ReportWriter, ORACLE Case, Oracle Forms and Oracle Report are registered trademarks of Oracle Corporation.

2. Programmatic Advice

The following list of advice is by no means complete, and skilled developers will be able to find cases, where the advice is not very applicable. Still, I suggest the advice should be discussed among Oracle7 developers.

2.1 Procedures and Triggers

2.1.1 Use Packages Instead of writing complex triggers, develop some generic procedures in packages, and call these from the triggers. This will increase the possibility of reusing the code, saving execution time, storage and development effort.

2.1.2 Use before Triggers Since snapshots require the use of *after row* triggers on all the three actions you should prefer *before* triggers.

2.1.3 Use Integrity Constraints - If Possible Using Integrity constraints to make sure that foreign keys are valid is indeed a feature which should be utilized.

Each table should have a *primary key* constraint, and possibly some *unique* constraints defined. Note that it is possible to ask for a primary key based on columns not declared as mandatory, but the definition of the primary key constraint will change these columns to *not null.*

The following constraint shows how to protect the *emp.deptno* column from getting values not pointing to any legal key in the *dept* table:

```
alter table emp add
 (constraint emp_foreign_key
  foreign key ( deptno ) references dept)
```

You may in fact instruct the kernel to automatically delete all employees from a department, when that department is deleted. This may be accomplished by adding *on delete cascade* to the constraint.

The following example will delete all employees with the corresponding manager being deleted recursively:

```
alter table emp add
 (constraint emp_self_key
  foreign key ( mgr ) references emp on delete cascade )
```

It is, however still not possible to specify *on update cascade*, if we want all the corresponding employees to change their department numbers, when changing a department number. This must be done through triggers, although the construct is defined in the SQL2 standard.

Suppose we want to *help* the kernel to tell how we want an update to be implemented:

```
create trigger upd_dept
 before
 update of deptno on dept
 for each row
 begin
  update emp
   set deptno = :new.deptno
   where deptno = :old.deptno;
 end
```

We would see the following messages when trying to modify a department number from the *dept* table:

```
ORA-04091: table SCOTT.EMP is mutating, trigger may not read or modify it
ORA-06512: at line 2
ORA-04088: error during execution of trigger 'SCOTT.UPD_DEPT'
```

Because the trigger and the integrity constraint are in conflict.

We could in fact drop the constraint and implement a trigger instead, but then a *truncate* of the table could violate data integrity.

The first shut could look like this:

```
create trigger ref_dept_after
 after
 update of deptno or delete on dept
 for each row
 begin
  if updating then
    update emp
      set deptno = :new.deptno
      where deptno = :old.deptno;
  end if;
  if deleting then
    delete from emp
    where deptno = :old.deptno;
  end if;
 end;

create trigger ref_emp_before
 before
 update of deptno or insert on emp
 for each row
 when (new.deptno is not null)
 declare
  my_dept dept.deptno%type;

  cursor get_deptno (dn Number) is
    select deptno from dept
    where deptno = dn
    for update of deptno;

 begin
  open get_deptno (:new.deptno);
  fetch get_deptno into my_dept;
  if get_deptno%NOTFOUND then
    Raise_Application_Error( -20291, 'Reference violated' );
  end if;
  close get_deptno;
 end;
```

Now deleting goes fine, but still updating is troublesome:

```
ORA-04091: table SCOTT.DEPT is mutating, trigger may not read or modify it
ORA-06512: at line 4
ORA-06512: at line 8
ORA-04088: error during execution of trigger 'SCOTT.REF_EMP_BEFORE'
ORA-06512: at line 3
ORA-04088: error during execution of trigger 'SCOTT.REF_DEPT_BEFORE
```

The two triggers conflict, although they should not, because updating the emp table through the dept-trigger will automatically preserve the consistancy. So there is no need to fire the emp trigger, when the change is happening from the *friend*. It could be implemented like this:

```
create package dept_emp_management as

 procedure come_from_friend;
 procedure not_from_friend;

 procedure dept_emp_cascade(
  upd in Boolean, old_key in Number, new_key in Number);

 function dept_for_key_check(new_key in Number) return Boolean;
end dept_emp_management;

create or replace package body dept_emp_management as

 from_friend Boolean := FALSE;

 procedure come_from_friend is
 begin
  from_friend := TRUE;
 end come_from_friend;

 procedure not_from_friend is
 begin
  from_friend := FALSE;
 end not_from_friend;

 procedure dept_emp_cascade(
  upd in Boolean, old_key in Number, new_key in Number) is
 begin
  if upd then
   update emp
    set deptno = new_key
    where deptno = old_key;
  else
   delete from emp
   where deptno = old_key;
  end if;
 end dept_emp_cascade;

 function dept_for_key_check(new_key in Number) return Boolean is
  my_dept dept.deptno%type;

  cursor get_deptno (dn Number) is
   select deptno from dept
   where deptno = dn
   for update of deptno;

 begin
  if dept_emp_management.from_friend then return TRUE; end if;
  open get_deptno (new_key);
  fetch get_deptno into my_dept;
  if get_deptno%NOTFOUND then
   return FALSE;
```

```
    end if;
    close get_deptno;
    return TRUE;
  end dept_for_key_check;

end dept_emp_management;

create trigger ref_dept_after
  after
  update of deptno or delete on dept
  for each row
  begin
    dept_emp_management.dept_emp_cascade( updating, :old.deptno, :new.deptno );
  end;

create trigger ref_emp_before
  before
  update of deptno or insert on emp
  for each row
  when (new.deptno is not null)
  begin
    if not dept_emp_management.dept_for_key_check(:new.deptno) then
      Raise_Application_Error( -20291, 'Reference Violation' );
    end if;
  end;

create trigger ref_dept_start
  before
  update of deptno on dept
  begin
    dept_emp_management.come_from_friend;
  end;

create trigger ref_dept_stop
  after
  update of deptno on dept
  begin
    dept_emp_management.not_from_friend;
  end;
```

I have not been able to implement the *on update cascade* feature on a self referencing reference yet, due to mutating conflicts.

2.1.4 Multiple Triggers of the same kind From version 7.1.3 of Oracle7, multiple triggers of the same type, on the same table, and acting on the same events are allowed. Provided you did specify that you wanted this feature by adding the parameter *compatible = "7.1.3"* to the parameter file.

This feature is indeed great, since it allows different modules to add their own triggers to the system, regardless of when triggers on the same table, type and kind already exist. (Hereby snapshorts and after rowlevel triggers may now be used together).

There are, however a number of issues to consider when using this feature.

Triggers of the same type and kind on a table are still executed in random order, so one should carefully analyze if the whole sequence delivers the same result regardless of the sequence. Different triggers that manipulate the same rows in foreign tables should therefore be melted into one trigger.

Also the database administrator should remember to add the *compatible* parameter to any other database if a total import is to be done. Or else only the first trigger will be known to the database kernel.

The following sql-statement will show if a database (prior to total export) utilizes this feature:

```
select decode(t.type,1,'Row level','Statement level'),
    decode(t.update$,1,'for update'),
    max(u.name)||'.'||max(o.name), count(*)
from sys.trigger$ t, sys.obj$ o, sys.user$ u
where t.baseobject = o.obj#
 and o.owner# = u.user#
 and t.update$ = 1 and t.enabled = 1
group by t.type, t.update$, t.baseobject
 having count(*) > 1
union
select decode(t.type,1,'Row level','Statement level'),
    decode(t.insert$,1,'for insert'),
    max(u.name)||'.'||max(o.name), count(*)
from sys.trigger$ t, sys.obj$ o, sys.user$ u
where t.baseobject = o.obj#
 and o.owner# = u.user#
 and t.insert$ = 1 and t.enabled = 1
group by t.type, t.insert$, t.baseobject
 having count(*) > 1
union
select decode(t.type,1,'Row level','Statement level'),
    decode(t.delete$,1,'for delete'),
    max(u.name)||'.'||max(o.name), count(*)
from sys.trigger$ t, sys.obj$ o, sys.user$ u
where t.baseobject = o.obj#
 and o.owner# = u.user#
 and t.delete$ = 1 and t.enabled = 1
group by t.type, t.delete$, t.baseobject
 having count(*) > 1
```

Figure 1. The *multtrg.sql* Statement

2.1.5 Protecting Source in the RDBMS Server The ability to maintain procedures, functions and packages in the Oracle7 kernel raises a security problem, because a privileged user may change these procedures, thus

changing the very core of the application, without making sure that these changes are known to the distributor of the original functions. This will make error tracking and support rather difficult.

First, let us agree that this problem has been with us since the early days of using relational technology, because table definitions, views, indexes, etc, may have been changed likewise.

Second, the procedure source cannot be encrypted or locked, so we cannot prevent a skilled customer from changing the source.

We can, however save information about the objects in the data dictionary, in such a way that we can easily check if the objects have been changed since they where shipped. Note that the creation date cannot be used, since the customer may reimport the whole lot. We have to log information based on the source directly.

The proposed method works in tree phases: Phase 1 extracts information from the data dictionary about the objects in question (*chksum.sql*). Phase 2 will compute a checksum for each object (*chksum.pl*) and prepare a script to insert the information in the database. Phase 3 may run the generated script in order to make the comparison more easy.

The two mentioned scripts should not generally be available at the customer site.

```
set heading off
set verify off
set recsep off
set pagesize 9999
set linesize 255
set feedback off
set long 50000
column secondcol newline
break on objid skip 1
select 'Owner: '||user||
    ', Type: '||Rtrim( type )||
    ', Name: '||Rtrim( name )||
    ', Date: '||To_Char( sysdate,'DD-MON-YYYY HH24.MI.SS' ) objid,
    Rtrim( text )
from user_source
where type in ('PACKAGE', 'PROCEDURE', 'PACKAGE BODY', 'FUNCTION')
order by type, name, line;

select 'Owner: '||user||
    ', Type: TRIGGER-'||Translate( FBrtrim(trigger_type),' ','_' )||
    ', Name: '||Rtrim( table_name )||'.'||Rtrim( trigger_name )||
    ', Date: '||To_char( sysdate,'DD-MON-YYYY HH24.MI.SS' ) objid,
trigger_body
from user_triggers
order by table_name, trigger_name, trigger_type;

select 'Owner: '||user||
    ', Type: VIEW'||
    ', Name: '||Rtrim( view_name )||
    ', Date: '||To_Char( sysdate,'DD-MON-YYYY HH24.MI.SS' ) objid,
text
from user_views
order by view_name;

select 'Owner: '||user||
    ', Type: PRIMARY_KEY'||
    ', Name: '||Rtrim( t.table_name )||'.'||Rtrim( t.constraint_name )||
    ', Date: '||To_Char( sysdate,'DD-MON-YYYY HH24.MI.SS' ) objid,
c.column_name secondcol
from user_constraints t, user_cons_columns c
where t.owner = c.owner
 and t.constraint_name = c.constraint_name
 and t.constraint_type = 'P'
order by t.table_name, t.constraint_name;

select 'Owner: '||user||
    ', Type: REFERENCE'||
    ', Name: '||Rtrim( t.table_name )||'.'||Rtrim( t.constraint_name )||
    ', Date: '||To_Char( sysdate,'DD-MON-YYYY HH24.MI.SS' ) objid,
t.status, t.delete_rule, c.column_name
from user_constraints t, user_cons_columns c
where t.owner = c.owner
 and t.constraint_name = c.constraint_name
 and t.constraint_type = 'R'
```

```
order by t.table_name, t.constraint_name;

select  'Owner: '||user||
     ', Type: CHECK'||
     ', Name: '||Rtrim( t.table_name )||'.'||Rtrim( t.constraint_name )||
     ', Date: '||To_Char( sysdate,'DD-MON-YYYY HH24.MI.SS' ) objid,
c.column_name secondcol, t.search_condition
from user_constraints t, user_cons_columns c
where t.owner = c.owner
  and t.constraint_name = c.constraint_name
  and t.constraint_type = 'C'
order by t.table_name, t.constraint_name;

select 'Owner: '||user||
     ', Type: TABLE'||
     ', Name: '||Rtrim( t.table_name )||
     ', Date: '||To_Char( sysdate,'DD-MON-YYYY HH24.MI.SS' ) objid,
t.tablespace_name secondcol, t.cluster_name,
c.column_name, c.data_type, c.data_length, c.data_precision, c.data_scale,
c.nullable, c.column_id
from user_tables t, user_tab_columns c
where t.table_name = c.table_name
order by t.table_name;

select 'Owner: '||user||
     ', Type: INDEX'||
     ', Name: '||Rtrim( i.table_name )||'.'||Rtrim( i.index_name )||
     ', Date: '||To_Char( sysdate,'DD-MON-YYYY HH24.MI.SS' ) objid,
i.tablespace_name secondcol, i.uniqueness, c.column_name, c.column_position
from user_indexes i, user_ind_columns c
where i.index_name = c.index_name
  and i.table_name = c.table_name
order by i.table_name, i.index_name;

exit
```

Figure 2. *chksum.sql* SQL*Plus Script

```
#! perl
eval "exec /usr/local/bin/perl -S $0 $*"
   if $running_under_some_shell;
##############################################
# chksum.pl 9. jan. 94   MJ DDE
# Usage: chksum.pl list
# VERS="@(#) oracle7.mm 1.1.60.1 5/22/96"

$tmpfile="/tmp/chksum$$";
$sumcom="sum $tmpfile";
printf( "drop table check_sum;\n\n ");
printf( "create table check_sum (owner varchar2(30), obj_type varchar2(61),\n ");
printf( "  obj_name varchar2(61), info_date date, obj_check number);\n\n" );
$state='first';
while (<>) {
 if (/^Owner: /) {
   if ($state eq 'first') {
     $state = 'not first';
   } else {
     close( SRC );
     $check = `$sumcom`;
     ($check, @rest) = split( ' ', $check );
     printf( "insert into check_sum (owner, obj_type, obj_name, info_date, obj_check) values\n" );
     printf( "  ('%s', '%s', '%s',\n", $owner, $type, $name );
     printf( "    to_date( '%s',' DD-MON-YYYY HH24.MI.SS' ), %s);\n", $date, $check );
   }
   ($owner, $type, $name, $date, @rest) = split( ',' );
   chop( $date );
   ($temp1, $owner, @rest) = split( ':', $owner );
   ($owner, @rest) = split( ' ', $owner );
   ($temp1, $type , @rest) = split( ':', $type );
   ($type , @rest) = split( ' ', $type );
   ($temp1, $name , @rest) = split( ':', $name );
   ($name , @rest) = split( ' ', $name );
   ($temp1, $date , @rest) = split( ':', $date );
   open( SRC, ">$tmpfile" );
 } else {
   s/ *$//;
   print( SRC $_ );
 }
}
close( SRC );
$check = `$sumcom`;
($check, @rest) = split( ' ', $check );
printf( "insert into check_sum (owner, obj_type, obj_name, info_date, obj_check) values\n ");
printf( "  ('%s', '%s', '%s', to_date('%s',' DD-MON-YYYY HH24.MI.SS'), %s);\n",
  $owner, $type, $name, $date, $check );
printf( "\ncommit;\n" );
system( 'rm -r $tmpfile' );
```

Figure 3. *chksum.pl* Perl Script

2.1.6 Table Logging It is often desirable to keep a log on changes of important tables in order to be able to track more historical information on who changed what when.

Often the application developer should not be involved in this process, since different applications may address the same set of tables. It is thus an advantage if the logging procedures can be stored and maintained within the database system close to the tables in question.

This is in many cases possible to achieve through triggers as the following example will suggest.

Assume that we have an important table **ATable** with the columns **aColumnA** and **aColumnB**.

We could then specify the following trigger:

```
create trigger ATableLogTrg
 before
 delete or insert or update
 on ATable
 for each row
 declare
  act Varchar(10);
 begin
  if inserting then
   act := 'INSERT';
  end if;
  if updating then
   act := 'UPDATE';
  end if;
  if deleting then
   act := 'DELETE';
  end if;
  insert into ATableLog (theColumnA, theColumnB, newColumnA, newColumnB,
     whenEvent, byWho, action)
   values (:old.aColumnA, :old.aColumnB, :new.aColumnA, :new.aColumnB,
    sysdate, user, act);
 end;
```

A possible look at the table **ATableLog** could show:

```
A   B N NB WHENVEN   BYWHO   ACTION
- -- - -- --------- ------- ----------
      a  1 12-APR-94 OPS$MJ  INSERT
      b  2 12-APR-94 OPS$MJ  INSERT
      x 11 12-APR-94 OPS$MJ  INSERT
      x 12 12-APR-94 OPS$MJ  INSERT
x 11  u 11 12-APR-94 OPS$MJ  UPDATE
b  2       12-APR-94 OPS$MJ  DELETE
```

Note that you could add a statement trigger to insert a row in a central historical table telling that changes in the **ATable** did take place.

2.1.7 Unique Index is also a kind of Constraint Let us examine the following senario. First we create a table with a *primary key* constraint:

```
create table t1 (
 a number,
 b varchar2(10),
 constraint key_ix primary key (a));
```

Examine from the dictionary that the constraint is known:

```
CONSTRAINT_NAME TABLE_NAME C COLUMN_NAME
--------------- ---------- - -----------
KEY_IX          T1         P A
```

Now insert 2 identical rows, and you will get the following error message:

```
ORA-00001: unique constraint (SCOTT.KEY_IX) violated
```

Now drop the constraint and create a unique index instead:

```
alter table t1 drop constraint KEY_IX;
create unique index ix on t1(a);
```

Now when two identical rows are inserted the following message appears:

```
ORA-00001: unique constraint (SCOTT.IX) violated
```

Looking for constraints on **t1** shows none at all. The conclusion is therefore that unique constraints are either declared explicitly (and found in the constraint views), or declared implicitly (and found in the index tables).

2.1.8 Remember Constraint Info Enabling a Constraint Assume the following table is created, and that the exception table **exceptions** does exist (use *@/rdbms/admin/utlexcpt*). Note that the pctfree parameter is specified for the underlying index.

```
create table t1 (
 a number,
 b varchar2(10),
 constraint key_ix primary key (a) using index pctfree 88
  exceptions into exceptions,
 constraint a_check check (a > 10)
  exceptions into exceptions)
```

Now try to insert some rows violating the constraints.

```
SQL> insert into t1 values (1, 'test');
ORA-02290: check constraint (SCOTT.A_CHECK) violated

SQL> insert into t1 values (11, 'test');

SQL> insert into t1 values (11, 'test2');
ORA-00001: unique constraint (SCOTT.KEY_IX) violated
```

The exceptions table is empty though! Disable the two constraints and insert the 3 rows again, and try now to enable the constraints:

```
SQL> alter table t1 enable constraint key_ix;
ORA-02299: cannot add or enable constraint (SCOTT.KEY_IX)- duplicate keys found

SQL> alter table t1 enable constraint a_check;
ORA-02296: cannot enable constraint (SCOTT.A_CHECK) - found noncomplying values
```

Still nothing is found in the exceptions table! The *exceptions into exceptions* clause has to be issued again before the exceptions table may be inspected:

```
ROW_ID             OWNER TABLE_NAME CONSTRAINT
------------------ ----- ---------- ----------
00000625.0000.0004 SCOTT T1         KEY_IX
00000625.0002.0004 SCOTT T1         KEY_IX
00000625.0001.0004 SCOTT T1         A_CHECK
```

Now delete the rows and enable the constraints again with the exceptions clause, and look for indexes on the T1 table:

SQL> **select** * **from** user_indexes **where** table_name = 'T1';

```
INDEX_NAME                     TABLE_OWNER
------------------------------ ------------------------------
TABLE_NAME                     TABLE_TYPE  UNIQUENES
------------------------------ ----------- ---------
TABLESPACE_NAME                 INI_TRANS  MAX_TRANS INITIAL_EXTENT NEXT_EXTENT
------------------------------ ---------- ---------- -------------- -----------
MIN_EXTENTS MAX_EXTENTS PCT_INCREASE   PCT_FREE     BLEVEL LEAF_BLOCKS
----------- ----------- ------------ ---------- ---------- -----------
DISTINCT_KEYS AVG_LEAF_BLOCKS_PER_KEY AVG_DATA_BLOCKS_PER_KEY CLUSTERING_FACTOR
------------- ----------------------- ----------------------- -----------------
STATUS
-----------
KEY_IX                         SCOTT
T1                             TABLE       UNIQUE
TOOLS                                   2        255          20480       20480
          1         249           50         10

VALID
```

The *pct_free* parameter was *forgotten* and storage information should thus be specified again at enable time.

2.1.9 Embeded SQL from PL/SQL procedures From version 7.1 of Oracle7 it is possible to run dynamic sql-statements from the general package called *dbms_sql*. This may be used to implement more general procedures and packages, and will make life more easy for those who wish to store total (or partly) sql-statements in tables.

The following small function example returns the number of rows on a table given as argument:

```
create or replace function count_rows( s_table in varchar2 ) return integer is
 -- This procedure counts rows from a given table
 count_cursor integer;
 rows integer;
 fetched_rows integer;
 rows_processed integer;

begin
 -- prepare a cursor to select number of rows from the table
 count_cursor := dbms_sql.open_cursor;
 dbms_sql.parse( count_cursor,
          'select count(*) from ' || s_table, dbms_sql.native );
 dbms_sql.define_column( count_cursor, 1, rows );
 rows_processed := dbms_sql.execute( count_cursor );
 fetched_rows := dbms_sql.fetch_rows( count_cursor );

 -- get column values of the row
 dbms_sql.column_value( count_cursor, 1, rows );
 dbms_sql.close_cursor( count_cursor );

 -- return rows;
 return( rows );

exception
 when others then
  if dbms_sql.is_open( count_cursor ) then
   dbms_sql.close_cursor( count_cursor );
  end if;
  return( -1 );
end;
```

Figure 4. The *count_rows* Function

2.1.10 User defined SQL Functions From version 7.1 of the Oracle kernel it is possible to define functions in PL/SQL, and to use these functions directly in the SQL-statements. The following example validates if a Danish social number could pass a modulus 11 check.

```
create function is_cpr( cpr in char ) return integer as
begin
 if length( cpr ) != 11 then return 1; end if;
 if substr( cpr,7,1 ) != '-' then return 1; end if;
 if rtrim( ltrim( translate( cpr,'0123456789','          '))) != '-' then
  return 1;
 end if;
 if ( to_number( substr( cpr,1,1 ) ) * 4 +
    to_number( substr( cpr,2,1 ) ) * 3 +
    to_number( substr( cpr,3,1 ) ) * 2 +
    to_number( substr( cpr,4,1 ) ) * 7 +
    to_number( substr( cpr,5,1 ) ) * 6 +
    to_number( substr( cpr,6,1 ) ) * 5 +
    to_number( substr( cpr,8,1 ) ) * 4 +
    to_number( substr( cpr,9,1 ) ) * 3 +
    to_number( substr( cpr,10,1 ) ) * 2 +
    to_number( substr( cpr,11,1 ) ) ) mod 11 = 0 then
  return 0;
 else
  return 1;
 end if;
end;

select c from cpr_test
where is_cpr( c ) = 0;

create view cpr as
 select c from cpr_test
 where is_cpr( c ) = 0;
```

If you try to redefine a standard function, you might find it hard to call the user defined version without prefixing with the schema name.

```
create function trunc( n in number, m in number ) return number as
begin
-- This is a silly reimplementation of trunc - only used as an example.
 if m = 0 then
-- return trunc( n, 0 ); would a recursive call to the function itself
-- stoppable with ctrl-c
   return n - mod( n, 1 );
 else
   return n;
 end if;
end;

select trunc( 3.1234, 0 ), trunc( 3.1234, 1 ),
 trunc( 3.1234, 2 ) from dual;

TRUNC(3.1234,0) TRUNC(3.1234,1) TRUNC(3.1234,2)
--------------- --------------- ---------------
              3             3.1            3.12

select scott.trunc( 3.1234, 0 ), scott.trunc( 3.1234, 1 ),
 scott.trunc( 3.1234, 2 ) from dual;

SCOTT.TRUNC(3.1234,0) SCOTT.TRUNC(3.1234,1) SCOTT.TRUNC(3.1234,2)
--------------------- --------------------- ---------------------
                    3                3.1234                3.1234
```

Even with a public synonym, the built-in functions are preferred.

2.1.11 User Defined SQL Functions with Memory The new version 7.1 feature allowing users to define their own SQL functions may (as stated earlier) be used to create extra functions on the server-side accessible to all SQL based clients.

There are however restriction on what these functions may do. They must not change the state of any database object, since there is no explicit commit to be expected.

There is however a way to inform Oracle, that a function is safe. Here the new *restrict_references* pragma may be of use.

The following example shows how a function to calculate a running total may be implemented, using this technique, where the function has its own memory of previous values or states.

```
create or replace package total_values as
 pragma restrict_references( total_values, wnds, rnds, wnps, rnps );

 function total (rno in Number, val in Number ) return number;
 pragma restrict_references( total, wnds, rnds );

 procedure reset_total;
end total_values;

create or replace package body total_values as
 the_total Number;

 function total (rno in Number, val in Number ) return Number is
 begin
  if rno = 1 then
   the_total := 0;
  end if;
  the_total := the_total + val;
  return the_total;
 end;

 procedure reset_total is
 begin
  the_total := 0;
 end;
end total_values;
```

Now the following select will give a running total for each row in the select.

```
select ename, sal, total_values.total(rownum, sal) from emp
```

If the pragma was not used, the following error would have been given:

```
ORA-06571: Function RUNNING_TOTAL does not guarantee not to update database
```

Note that the example is not preaty as the rownumber is used to reset the internal counter, but if *order by* is used by the select statement and no indexes are in use, the first row returned is not guarentied to have *rownum* one, therefor one should use the reset procedure.

2.2 Snapshots

Snapshots are easy to create. This one should be updated by Oracle every day at 1 pm:

```
create snapshot scott.dept_sal
 refresh fast next Trunc( sysdate ) + 13/24 as
  select d.deptno, Sum( e.sal ) sum_sal
  from scott.dept d, scott.emp e
  where d.deptno = e.deptno(+)
  group by d.deptno;
```

Note that the *next* expressions are rather limited. If you want to say every 10 seconds you would probably specify:

```
Trunc( sysdate )+( To_Number( To_Char( sysdate,'SSSSS' ) )+100)/86400
```

But then you would get the error:

```
ORA-01480: trailing null missing from STR bind value
```

Even snapshots based on complex views may be queried using rowid, since Oracle7 builds a read-only table for the snapshot.

```
select rowid, deptno, sum_sal from dept_sal;

ROWID                  DEPTNO  SUM_SAL
------------------     ------  -------
000016F5.0000.0001     10      27048
000016F5.0001.0001     20      39520
000016F5.0002.0001     30      13100
000016F5.0003.0001     40
```

Note that the snapshot is made of two views and one read-only table:

```
select * from tab;

DEPT_SAL                VIEW
MVIEW$_DEPT_SAL         VIEW
SNAP$_DEPT_SAL          TABLE
```

You may select from the view *user_snapshots* to obtain more information on the snapshot.

Please, note that you could use snapshots to collect aggregate data locally or remotely on a regular basis, but Oracle7 does not guarantee the contents of the snapshot to be consistent with the original tables at any time. Neither does Oracle7 guarantee two table snapshots from two different tables to be consistent, since one transaction updating the two original tables will not propagate the changes to the snapshots as one transaction. Oracle has promised to support transaction autonomy across replication later in version 7. This version will not use the snapshot facility, but underlying system packages for Remote Oracle Procedure Calls as well as some sort of job control.

2.3 Evolution of the Oracle RDBMS

The kernel rdbms has always been the basic product of Oracle, and up to version 6 Oracle has included many features to deal with performance and multi user issues.

From *Oracle7* a long line of features are introduced enabling the user to pull more and more semantic and algorithmic information into the rdbms itself making it appear more like an traditional operating system. Some of these facilities covered in Oracle7 are:

- Procedures and functioner can have eperate execute priviledges, and may take *default* parameter specifications.
- Overloading of procedures and functioner in the same package.
- Package and package body including session persistent variables.
- Userwritten functioner as SQL functioner from version 7.1. Unfortunately there is no easy way to redefine the standard functions.
- DDL and embeded SQL from the *DBMS_SQL* package from PL/SQL 2.1.
- More of the same kind of triggers per tabel per event in version 7.1.
- Simple user defined types in PL/SQL version 2.1.
- Compilering of PL/SQL version 2.2 code protecting the source.
- Widening the SQL syntax, introducing *Inline views* from the SQL3 draft.
- Extending the notion of tables in PL/SQL to cover records instead of just simple values. Expected in PL/SQL version 2.3.
- In PL/SQL *Tabel* types based on simple types are offered, as thin arrays without narow limits, suitable for returning more komplex struktures. (Partly in PL/SQL version 2.2!)
- Pasing an already parsed and opened curser as parameter to other procedures or functioner. (Not quite in PL/SQL version 2.2)

2.3.1 Some Eksempel from Oracle 7.2 The following statements are executed on the Oracle rdbms:

```
Oracle7 Server Release 7.2.2.3.0 - Production Release
With the distributed, replication and parallel query options
PL/SQL Release 2.2.2.3.0 - Production
```

First a simple small user defined function.

```
create or replace function short_id( n in Varchar2 ) return Varchar2 is
 r Varchar2( 2000 );
begin
 r := replace( translate( upper( n ), 'AEIOUYFXE', 'AAAAAAAAA' ), 'A', '');
 return substr( r, 1, 3 );
end short_id;
--
-- Could be used like this:
-- select * from emp where short_id( ename ) = 'BLK';
```

Example of different procedures and functions.

```
create or replace package dept_emp_management as

 type Emps_in_dept_type is table of emp.ename%type index by binary_integer;

 last_emp emp.ename%type;

 function incr_sal( p_emp in Varchar2, p_incr in Number Default 200 ) return Number;
 function incr_sal( p_emp in Number, p_incr in Number) return Number;

 function get_dept_emps( p_dept in Number, p_count out Number ) return Emps_in_dept_type;

 procedure name_test( p_a in Number );
 procedure name_test( p_a in Varchar2 );

end dept_emp_management;

-- Cluld be used like this:
-- sal := incr_sal( 'FORD', -10 );
-- sal := incr_sal( 'FORD' );
-- sal := incr_sal( 'FORD', p_incr => -100 );
-- sal := incr_sal( 7902, 10 );
```

```
create or replace package body dept_emp_management as
--
 function incr_sal( p_emp in Number, p_incr in Number default 200 ) return Number is
   count_val Number;
--
   cursor get_emp ( en Number ) is
    select rowid, ename, sal from emp
    where empno = en
--   where current of get_emp;
--   Will give Oracle error.
    for update of sal;
   emp_rec   get_emp%rowtype;
--
 begin
   open get_emp ( p_emp );
   fetch get_emp into emp_rec;
   if get_emp%notfound then return null; end if;
   close get_emp;
   update emp
    set sal = emp_rec.sal + p_incr
    where empno = p_emp;
   commit;
   last_emp := emp_rec.ename;
   return emp_rec.sal + p_incr;
 end incr_sal;
--
 function get_dept_emps( p_dept in Number, p_count out Number ) return Emps_in_dept_type is
   emps Emps_in_dept_type;
   counter Number := 0;
--
   cursor get_emps ( dn Number ) is
    select rowid, empno, ename from emp
    where deptno = dn;
--
 begin
   for emp_rec in get_emps( p_dept ) loop
    counter := counter + 1;
    emps( counter ) := emp_rec.ename;
   end loop;
   p_count := counter;
   return emps;
 end get_dept_emps;
--
end dept_emp_management;
```

A small example of a select statement using an *inline view*:

```
select *
from (select job, count(*) no from emp group by job)
where no > 1
order by no
```

2.3.2 In later Versions of the Oracle RDBMS In later Versions of the Oracle RDBMS an actual objekt vission is introduced, making it possible to define abstract datatypes (ADT) including methods for these, embeded tables and actual objekt identifikation and definition.

The following small example shows part of what we may expect on the ADT front.

```
create type period (
  start_time date,
  end_time date,
  status varchar2(3),
  member function in_period return integer,
  member function days_left return number );

create type note (
  name varchar2(30),
  kind varchar2(10),
  format varchar2(6),
  info text,
  intervals table (periods period) );

create type document under note (
  auther varchar2(30) );

create extent table intervals (
  p period );
```

2.4 Management

2.4.1 Version Control Using procedures and functions on the database server side instead of in the application on the client side, requires a lot more discipline by the developers.

If e.g. some central procedures are used by different applications, how do we make changes to them without having to inspect all the applications immediately?

You may from the very beginning add a version parameter to the procedure, in order at be able to add new versions to the implementation of the procedures, like this:

```
create package emp_mgmt as
 type emps_type is table of emp.empno%type index by binary_integer;

 function max_sal (version in char, default_sal in number) return number;
 procedure sal_raise (version in char, pct in number);
 procedure get_dept (version in char, deptno_par in number, emps out emps_type);
end emp_mgmt;

create package body emp_mgmt as
 version_not_ready exception;
 pragma exception_init(version_not_ready, -1096);
-- We are here using the Oracle error 1096 with the following generic text:
-- program version (%s) incompatible with instance (%s)
-- Although the parameters are not filled yet.

 function max_sal (version in char, default_sal in number) return number is
  max_sal_var emp.sal%type;
 begin
  if version = '1.0' then
   select max(nvl(sal,default_sal) + nvl(comm,0))
   into max_sal_var
   from emp;
   return(max_sal_var);
  else
   raise version_not_ready;
  end if;
 end;

 procedure sal_raise (version in char, pct in number) is
 begin
  if version = '1.0' then
   update emp
    set sal = sal + sal*pct/100;
  elsif version = '1.1' then
   update emp
    set sal = nvl(sal + sal*pct/100, 100);
  else
   raise version_not_ready;
  end if;
 end;

 procedure get_dept (version in char,
                     deptno_par in number,
                     emps out emps_type) is
  i binary_integer := 0;
 begin
  if version = '1.0' then
   for emprec in (select empno from emp where deptno = deptno_par) loop
    i := i + 1;
    emps(i) := emprec.empno;
   end loop;
  else
   raise version_not_ready;
  end if;
```

```
 end;

end emp_mgmt;
```

2.4.2 Adapt a Naming Convention In order to distinguish local PL/SQL variables from database object names, always prefix the local PL/SQL variables with something. (like *Local, p_, ...*)

2.4.3 Tracking dependencies As more types of objects are handled by the Oracle kernel, there is a stronger need to be able to check which objects depend on others.

The following series of action shows how this may accomplished.

```
connect scott/tiger

create table base_table (
 key number, data varchar2( 200 ) );

create trigger base_before
 before
 update or insert on base_table
 for each row
 begin
  if :new.key = 0 then
     :new.key := 999;
  end if;
 end;

create function count_base_rows return number is
 rows number;
begin
 rows := 0;
 select count( * ) into rows from base_table;
 return rows;
end;

create view base_view as
 select * from base_table;

grant select on base_view to system;

connect system

create synonym foreign_syn for scott.base_view;

create view foreign_view as
 select * from foreign_syn;
```

First create a package to handle this, either from *rdbms/admin/utldtree*, or the following from the system user account.

```
create table audit_dependencies (
 object_id number not null );

create package audit_dep as
 procedure clear_dependencies;
 procedure new_dependency(
  fp_object_type in varchar2,
  fp_object_name in varchar2,
  fp_schema_name in varchar2);
end audit_dep;

create or replace package body audit_dep as

 procedure clear_dependencies is
 begin
  delete from audit_dependencies;
  commit;
 end;

 procedure new_dependency(
  fp_object_type in varchar2,
  fp_object_name in varchar2,
  fp_schema_name in varchar2) is
  l_object_id number;
 begin
  select object_id into l_object_id from all_objects
   where owner       = upper( fp_schema_name )
   and   object_name = upper( fp_object_name )
   and   object_type = upper( fp_object_type );
  insert into audit_dependencies ( object_id ) values ( l_object_id );
  commit;
  insert into audit_dependencies ( object_id )
   select object_id
    from public_dependency
    connect by prior object_id = referenced_object_id
    start with referenced_object_id = l_object_id;
  commit;
 exception
  when no_data_found then
   raise_application_error(-20000, 'ORU-10013: '||
    fp_object_type||' '||fp_schema_name||'.'||fp_object_name||
    ' was not found.');
 end;

end audit_dep;
```

If we want to see which objects depend on e.g. base_table owned by scott, the following statement may be executed.

```
execute audit_dep.new_dependency( 'table', 'base_table', 'scott' );

select * from audit_dependencies;
    4649
    4650
    4652
    4671

select distinct u.user_id, d.object_id,
    '('||o.owner||'.'||o.object_name||') ('||o.object_type||')' object
from audit_dependencies d, dba_objects o, dba_users u
where o.object_id = d.object_id
 and u.username = o.owner

   USER_ID  OBJECT_ID OBJECT
---------- ---------- --------------------
         8       4649 (SCOTT.BASE_TABLE) (TABLE)
         8       4650 (SCOTT.BASE_VIEW) (VIEW)
         8       4671 (SCOTT.COUNT_BASE_ROWS) (FUNCTION)
         5       4652 (SYSTEM.FOREIGN_VIEW) (VIEW)
```

2.4.4 Space Usage in Tables, Indexes and Clusters From version 7.2 an onwards it is possible to see how much space allocated to an object is actually in use by that object.

```
rem name: space.sql
rem Usage: sqlplus <account> @space object_name object_type
rem
rem on 7.2 or later, will tell about space usage for an object.
rem
set verify off
set serveroutput on
declare
 cursor get_extent_info ( fp_name Varchar2, fp_type Varchar2 ) is
  select count( * ) extents, sum( blocks ) blocks,
      sum( bytes ) bytes, max( tablespace_name ) ts_name
  from user_extents
  where segment_name = fp_name
   and segment_type = fp_type;
--
 cursor get_freelists ( fp_name Varchar2, fp_type Varchar2 ) is
  select freelists from user_tables
  where table_name = fp_name and 'TABLE' = fp_type
  union
  select freelists from user_indexes
  where index_name = fp_name and 'INDEX' = fp_type
  union
  select freelists from user_clusters
  where cluster_name = fp_name and 'CLUSTER' = fp_type;
--
 ts_name Varchar2( 30 );
 l_object_name  Varchar2( 30 );
 l_object_type  Varchar2( 30 );
 l_total_blocks          Number;
```

```
  l_total_bytes              Number;
  l_unused_blocks            Number;
  l_unused_bytes             Number;
  l_last_used_extent_file_id     Number;
  l_last_used_extent_block_id    Number;
  l_last_used_block          Number;
  l_freelist_group_id        Number;
  l_free_blks                Number;
  l_freelists                        Number;
--
begin
  l_object_name := upper( '&1' );
  l_object_type := upper( '&2' );
--
  dbms_output.put_line( 'Extent info for '||l_object_type||' '||l_object_name||':' );
  for extent_rec in get_extent_info ( l_object_name, l_object_type ) loop
    dbms_output.put_line( '  Extents    : '||to_char( extent_rec.extents ));
    dbms_output.put_line( '  Blocks     : '||to_char( extent_rec.blocks ));
    dbms_output.put_line( '  Bytes      : '||to_char( extent_rec.bytes ));
    dbms_output.put_line( '  Tablespace : '||extent_rec.ts_name );
  end loop;
--
  dbms_output.put_line( '--' );
  dbms_space.unused_space( user, l_object_name, l_object_type, l_total_blocks,
    l_total_bytes, l_unused_blocks, l_unused_bytes, l_last_used_extent_file_id,
    l_last_used_extent_block_id, l_last_used_block );
--
  dbms_output.put_line( 'Unused space for '||l_object_type||' '||l_object_name||'.' );
  dbms_output.put_line( '  total_blocks          : '||to_char( l_total_blocks ));
  dbms_output.put_line( '  total_bytes           : '||to_char( l_total_bytes ));
  dbms_output.put_line( '  unused_blocks         : '||to_char( l_unused_blocks ));
  dbms_output.put_line( '  unused_bytes          : '||to_char( l_unused_bytes ));
  dbms_output.put_line( '  last_used_extent_file_id : '||to_char( l_last_used_extent_file_id ));
  dbms_output.put_line( '  last_used_extent_block_id: '||to_char( l_last_used_extent_block_id ));
  dbms_output.put_line( '  last_used_block       : '||to_char( l_last_used_block ));
--
  dbms_output.put_line( '--' );
  for freelist_rec in get_freelists ( l_object_name, l_object_type ) loop
    l_freelists := freelist_rec.freelists;
  end loop;
  dbms_output.put_line( 'No of Freelists for '||l_object_type||' '||
    l_object_name||': '||to_char( l_freelists ));
  l_freelist_group_id := 0;
  loop
    dbms_space.free_blocks( user, l_object_name, l_object_type,
      l_freelist_group_id, l_free_blks );
    dbms_output.put_line( 'Free blocks for '||l_object_type||' '||
      l_object_name||' group '||to_char( l_freelist_group_id )||': '||
      to_char( l_free_blks ));
    l_freelist_group_id := l_freelist_group_id + 1;
    exit when l_freelist_group_id = l_freelists;
  end loop;
end;
```

2.5 Tuning

2.5.1 Use Explicit Cursors If you know that a certain select does not return more than one row, use an explicit cursor, because an implicit cursor will always execute two fetches in order to get the *NO_MORE_ROWS* flag back.

2.5.2 Cache the SQL-Statements The *shared_pool* with a size that is set by the init parameter **shared_pool_size**, now holds a cache of already parsed SQL statements. If these entries are flushed too often, the pool should be larger.

Because an exact literal match is required for two SQL statements to use the same slot, it is advisable to put the SQL-statements in a library or include file to maximize the reusability of the parse information.

The package *dbms_shared_pool* may help you in getting a list of big elements, as well as keeping often used elements from being invalidated.

To get a list of elements of more than 30K each:

```
set serveroutput on size 20000
execute dbms_shared_pool.sizes(30);

SIZE(K) KEPT   NAME
-------        ----    ------------------------------------------------------------------
178                    SYS.DBMS_SQL              (PACKAGE BODY)
143                    SYS.STANDARD              (PACKAGE)
48                     SYS.DBMS_SQL              (PACKAGE)
37                     SELECT UPDATE_PRIV FROM TABLE_PRIVILEGES WHERE
                       TABLE_NAME=:b1 AAND UPDATE_PRIV='A'
                       (20171020,2100549302)    (CURSOR)
37                     SELECT TO_CHAR(SHARABLE_MEM / 1000 ,'999999') SZ,
                       DECODE(KEPT_VERSIONS,0,'    ',RPAD('YES(' ||
                       TO_CHAR(KEPT_VERSIONS)|| ')' ,6)) KEEPED,
                       RAWTOHEX(ADDRESS) || ',' ||TO_CHAR(HASH_VALUE) NAME,
                       SUBSTR(SQL_TEXT,1,354) EXTRA  FROM V$SQLAREA
                       WHERE SHARABLE_MEM > :b1 * 1000  UNION SELECT
                       TO_CHAR(SHARABLE_MEM / 1000 ,'999999') SZ,
                       DECODE(KEPT,'YES','YES(20587D3C,-474114044) (CURSOR)
31                     SYS.STANDARD              (PACKAGE BODY)
```

Note, the view *v$plsarea* to look at the program unit cache cannot be found. (In version 7.0.16.2)

2.5.3 Hints to the optimizer Not only is it possible to ask the optimizer to use a cust based approch instead of the rule based. It is also possible in specific SQL-statements, to give a direct hint to the optimizer on the purpose of the statement. It is however also possible to tell the optimizer which index should drive the main select, but this facility should be used with great care, since one never knows when the specified index will be removed or called something else.

2.6 Transactions

2.6.1 Package Variables Variables in packages do have values owned by the sessions using the packages. It is thus not possible directly to have one session change a variable to a value readable by another session.

Take the following small package as an example:

```
create package var_test as

 procedure add (a in number);

end var_test;
/

create package body var_test as

 pkg_var number := 0;

 procedure add (a in number) is
 begin
   dbms_output.put_line('add before: '||to_char(pkg_var));
   pkg_var := pkg_var + a;
   dbms_output.put_line('add after : '||to_char(pkg_var));
 end;

end var_test;
/
```

Using this package from *sqlplus* will look like this no matter if other sessions are using the same package, and no matter if they commit or rollback:

```
set serveroutput on
SQL> execute var_test.add(4);
add before: 0
add after : 4

PL/SQL procedure successfully completed.

SQL> execute var_test.add(4);
add before: 4
add after : 8
```

2.6.2 Discrite Transactions Discrete transactions are only possible if the parameter *discrete_transactions_enabled* is TRUE in the init.ora file. And the procedure *dbms_transaction.begin_discrete_transaction* has to be called as the first call to the database in order for every transaction to be discrete.

Please note however, that the call does not fail if the session parameter was not TRUE, so you should check this parameter in the application itself:

```
EXEC SQL select value into :val from V$PARAMETER
     where name = 'discrete_transactions_enabled';

if (!strcmp(val,"TRUE") {
 EXEC SQL EXECUTE
     begin
       dbms_transaction.begin_discrete_transaction;
     end;
 END-EXEC;
}
```

Unfortunately LONG values cannot be manipulated in discrete transactions.

2.7 Migration

Remember that the goal of having only one source master makes it very difficult to migrate the application to Oracle7, since they should in one way or other still be able to function well connected to Oracle 6.

As the following sections will indicate, some adjustments of the SQL-statements might be nessesary in order to migrate an application to Oracle7, although most applications would see no immediate changes at all.

2.7.1 Which Version In order to exercise new features in Oracle7, and still be able to run the same application against Oracle version 6, it is desirable to be able to identify the kind of database currently connected to.

One might want to see if some new views exist, such as *v$version*, but this might give a misleading answer, since some Oracle7 accounts may not have access to these views at all.

It is thus more safe to test if one of the new *all_* views exists. Use e.g. *all_errors* or *all_source*, since they are probably not present in the Oracle version 6 account.

Check if the parallel server option is used (the instance has been started in shared mode), by calling the routine *dbms_utility.is_parallel_server* returning TRUE or FALSE.

2.7.2 Char's and Long's In Oracle7 the *char* datatype means fixed-length strings, whereas *varchar* has similar capabilities as *char* from Oracle 6. So be careful if you are using the *char* datatype in Oracle7, you should probably be using *varchar* or more likely **varchar2** instead. And by the way, *varchar*

columns may now consist of up to 2000 characters.

In Oracle7 the *long* may store up to 2Gb of data, so do not think of having the total long column value in memory, unless you know a reasonable upper limit to the size of the structures.

2.7.3 Outer-Joins versus OR / IN Oracle7 is more restrictive on certain combinations of outer joins and OR or IN operations as the following small example shows:

Suppose we want to have a list of all departments, and if the given department has managers or the president, let us have their names as well. This could in Oracle 6 be expressed in the following way:

```
select dname, ename, job
from emp, dept
where dept.deptno = emp.deptno (+)
 and emp.job(+) in ('MANAGER','PRESIDENT')
```

Producing the following result:

```
DNAME          ENAME  JOB
-----------    -----  ---------
ACCOUNTING     CLARK  MANAGER
ACCOUNTING     KING   PRESIDENT
RESEARCH       JONES  MANAGER
SALES
OPERATIONS
```

But in Oracle7 the following error message appears:

```
ORA-01719: outer join operator (+) not allowed in operand of OR or IN
```

Here the statement could be expressed in the following way:

```
select dname, ename, job
from emp, dept
where dept.deptno = emp.deptno
 and emp.job in ('MANAGER','PRESIDENT')
union all
select dname, null, null
from dept
where dept.deptno not in (
   select deptno from emp
   where job in ('MANAGER','PRESIDENT'))
```

If the same statement should be used in Oracle 6 and in Oracle7 do not use the *all* clause, and see if there are other ways to express the query since *union* would initiate an internal sort.

2.7.4 Week numbers As the Americans and Europeans do not agree on how week numbers should be calculated, we used to use the *WW* format in Oracle 6 in combination with the *LANGUAGE* parameter. In Oracle7 however, the *WW* format always means the American way, whereas *IW* follows the *nls_territory* parameter.

Likewise you should use *IYYY, IYY, IY* and *I* instead of *YYYY, YYY, YY* and *Y*, if you want the local way of calculating the year instead of the American way.

In Oracle 6 without any NLS setting in the init.ora file:

select to_char(to_date('01-JAN-93'),'WW YYYY') **from** dual;

```
TO_CHAR(TO_DATE('01-JAN-93'),'WWYYYY')
--------------------------------------
01 1993
```

In Oracle7 with *nls_territory* = Denmark:

select to_char(to_date('01-JAN-93','DD-MON-YY'),'WW YYYY / IW IYYY') **from** dual

```
TO_CHAR(TO_DATE('01-JAN-93','DD-MON-YY'),'WWYYYY/IWIYYY')
---------------------------------------------------------
01 1993 / 53 1992
```

Please note that the example also indicates that Oracle7 is more restrictive regarding type conversion as '01-JAN-93' is not considered a date any more but a string. The corresponding date may be specified like this: *to_date('01-JAN-93','DD-MON-YY').*

2.7.5 Error Codes A number of error codes have been added to version 7.0 and 7.1, but some error codes have also been eliminated. Naturally this is to be considered when migrating applications into newer Oracle RDBMS versions. The following list of error codes are known to be eliminated in version 7.1.4 or before.

2.7.5.1 4040 Removing 4040.

```
04040, 00000, "new timestamp is not greater than existing one"
// *Cause:  The given timestamp is not greater than the current timestamp of
//          the existing object.
// *Action: Specify a greater timestamp.
```

2.7.5.2 **1930** Removing 1930.

01930, 00000, "no privileges to REVOKE"
// *Cause: "ALL" was specified when there were no privileges to revoke.
// *Action: Don't try revoking from that user.

2.7.5.3 **960, 962** Removing 110 and adding 960, 962.

00960, 00000, "ambiguous column naming in select list"
// *Cause: A column name in the order-by list matches more than one select
// list columns.
// *Action: Remove duplicate column naming in select list.

00962, 00000, "too many group-by / order-by expressions"
// *Cause: The group-by or order-by column list contain more than 255
// expressions.
// *Action: Use 255 or less expressions in the group-by or order-by list.

2.7.5.4 **1547** Adding 1650, 1651, 1652, 1653, 1654 and removing 1547.

```
01547, 00000, "failed to allocate extent of size %s in tablespace '%s'"
// *Cause: Tablespace indicated is out of space
// *Action: Use ALTER TABLESPACE ADD DATAFILE statement to add one or more
//        files to the tablespace indicated or create the object in other
//        tablespace if this happens during a CREATE statement

01650, 00000, "unable to extend rollback segment %s by %s in tablespace %s"
// *Cause:  Failed to allocate an extent for rollback segment in tablespace.
// *Action: Use ALTER TABLESPACE ADD DATAFILE statement to add one or more
//        files to the tablespace indicated.

01651, 00000, "unable to extend save undo segment by %s for tablespace %s"
// *Cause:  Failed to allocate an extent for saving undo entries for
//        the indicated offline tablespace.
// *Action: Check the storage parameters for the SYSTEM tablespace. The
//        tablespace needs to be brought back online so the undo can be
//        applied.

01652, 00000, "unable to extend temp segment by %s in tablespace %s"
// *Cause:  Failed to allocate an extent for temp segment in tablespace.
// *Action: Use ALTER TABLESPACE ADD DATAFILE statement to add one or more
//        files to the tablespace indicated or create the object in other
//        tablespace.

01653, 00000, "unable to extend table %s.%s by %s in tablespace %s"
// *Cause:  Failed to allocate an extent for table segment in tablespace.
// *Action: Use ALTER TABLESPACE ADD DATAFILE statement to add one or more
//        files to the tablespace indicated.

01654, 00000, "unable to extend index %s.%s by %s in tablespace %s"
// *Cause:  Failed to allocate an extent for index segment in tablespace.
// *Action: Use ALTER TABLESPACE ADD DATAFILE statement to add one or more
//        files to the tablespace indicated.

01655, 00000, "unable to extend cluster %s.%s by %s in tablespace %s"
// *Cause:  Failed to allocate an extent for cluster segment in tablespace.
// *Action: Use ALTER TABLESPACE ADD DATAFILE statement to add one or more
//        files to the tablespace indicated.
```

2.7.5.5 1596 Removing 1596.

```
01596, 00000, "fail to coalesce extents because sort area size is too small"
// MERGE: 1591 RENUMBERED TO 1596
// *Cause:  there are too many entries of free extents to sort it in the
//        in-memory sort area
// *Action: increase the sort area size or reduce the fragmentation in
//        tablespace by doing a full export followed by an import.
```

2.7.5.6 481, 482 Removing 481 and 482.

```
00481, 00000, "SMON process posting itself"
// *Cause: This is trapped internally
// *Action: None

00482, 00000, "SMON shut, shutdown abort required"
// *Cause: Instance or Transaction recovery was required after the SMON
//       process was stopped in anticipation of shutdown. This is
//       a rare condition that can occur only in multi-instance operation.
// *Action: Use shutdown abort. This instance will be recovered by surviving
//       instance(s) or by warm-start recovery.
```

2.7.5.7 983 Removing 983.

```
00983, 00000, "cannot define ROWID column - no corresponding SQL datatype"
// *Cause:  Attempt to create a table with a rowid column as in
//       "CREATE TABLE ... AS SELECT ROWID ... FROM ..." which is illegal
//       because there is no corresponding SQL datatype for rowid.
// *Action:  Use the function ROWIDTOCHAR to convert rowid to character as in
//       "CREATE TABLE ... AS SELECT ROWIDTOCHAR(ROWID) ... FROM ...".
```

2.7.5.8 1635, 1636 Removing 1635 and 1636.

```
01635, 00000, "rollback segment #%s specified not available"
// *Cause: (same as 1545)
// *Action: (same as 1545)

01636, 00000, "rollback segment '%s' is already online"
// *Cause: The instance is trying to online an already online RS
// *Action:
```

2.7.5.9 4093 Removing 4093.

```
04093, 00000, "references to columns of type LONG are not allowed in triggers"
// *Cause: A trigger attempted to reference a long column in the triggering
//       table.
// *Action: Do not reference the long column.
```

2.7.6 In future According to the ISO standard of SQL *null* should be different from the empty string(''). So even if the two different states cannot be distinguished today in Oracle7, it is advisable to use them accordingly in the applications.

2.7.7 Trailing NULLs Despite the fact that Oracle 6.0 manuals claimed that trailing NULLs were suppressed, the following example shows otherwise. This is important here since the same example shows that Oracle7 does suppress trailing NULLs, and that an export / import therefore may take up less space if big tables are involved and many trailing columns are NULL.

```
create table space_test (a date, b char(10), c char(11), d number(6),
    e number(7), f number(6,2), g number(6,3));

insert into space_test values ('02-JAN-94', 'GTTGTTGTTG', 'GTTGTTGTTGT',
    123456, 1234567, 1234.56, 123.456);

insert into space_test values ('03-JAN-94', NULL, 'GTT2', NULL, NULL, NULL,
    NULL);

insert into space_test values ('04-JAN-94', 'GTT3', 'GTT4', NULL,
    123456, NULL, NULL)

commit;

select * from space_test;
```

Gives the following result:

```
A         B          C                    D          E          F          G
--------- ---------- ---------- ---------- ---------- ---------- ----------
02-JAN-94 GTTGTTGTTG GTTGTTGTTGT     123456    1234567    1234.56    123.456
03-JAN-94            GTT2
04-JAN-94 GTT3       GTT4                      123456
```

Looking at the actual datablock shows:

```
          0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f   0123456789abcdef
2d65f90   00 00 2c 00 07 07 77 c2 01 04 01 01 01 04 47 54   ..,...wB......GT
2d65fa0   54 33 04 47 54 54 34 ff 04 c3 0d 23 39 ff ff 2c   T3.GTT4..C.#9..,
2d65fb0   00 07 07 77 c2 01 03 01 01 01 ff 04 47 54 54 32   ...wB.......GTT2
2d65fc0   ff ff ff ff 2c 00 07 07 77 c2 01 02 01 01 01 0a   ....,...wB......
2d65fd0   47 54 54 47 54 54 47 54 54 47 0b 47 54 54 47 54   GTTGTTGTTG.GTTGT
2d65fe0   54 47 54 54 47 54 04 c3 0d 23 39 05 c4 02 18 2e   TGTTGT.C.#9.D...
2d65ff0   44 04 c2 0d 23 39 05 c2 02 18 2e 3d f3 c1 00 08   D.B.#9.B...=sA..
          0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f   0123456789abcdef
```

On Oracle7 7.0.16 the similar datablock looks as follows:

```
          0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f   0123456789abcdef
c4df90    00 00 00 00 00 00 00 00 2c 00 05 07 77 c2 01 04   ........,...wB..
c4dfa0    01 01 01 04 47 54 54 33 04 47 54 54 34 ff 04 c3   ....GTT3.GTT4..C
c4dfb0    0d 23 39 2c 00 03 07 77 c2 01 03 01 01 01 ff 04   .#9,...wB.......
c4dfc0    47 54 54 32 2c 00 07 07 77 c2 01 02 01 01 01 0a   GTT2,...wB......
c4dfd0    47 54 54 47 54 54 47 54 54 47 0b 47 54 54 47 54   GTTGTTGTTG.GTTGT
c4dfe0    54 47 54 54 47 54 04 c3 0d 23 39 05 c4 02 18 2e   TGTTGT.C.#9.D...
c4dff0    44 04 c2 0d 23 39 05 c2 02 18 2e 3d 16 97 00 07   D.B.#9.B...=....
```

2.8 Tricks

2.8.1 Using NLS Here we shall try to give an idea about how to utilize the NLS features of Oracle7, as well as warn against some of the traps in this area.

NLS parameters may be set in the init.ora file, changed for each session, and controlled by the *NLS_LANG* environment variable. Anyway, a session may read its active NLS values in *v$nls_parameters*.

If none of the parameters are specified anywhere,' they default to *American*.

The environment variable *NLS_LANG* takes a value in the following format: *<language>[_<territory>[.<character set>]]*.

When the *NLS_LANG* variable is specified for a process, an implicit *alter session* is performed in order to change the nls parameters from the default setting. Note, that *sqlplus* only alters the session at the first login. Additional connects will not reflect these settings.

A select from *v$nls_parameters* may return the following values where the *nls_characterset* was specified during database creation:

```
PARAMETER            Default value NLS_LANG=danish
-------------------- ------------- ---------------
NLS_LANGUAGE         AMERICAN      DANISH
NLS_TERRITORY        AMERICA       DENMARK
NLS_CURRENCY         $             Kr
NLS_ISO_CURRENCY     AMERICA       DENMARK
NLS_NUMERIC_CHARACTERS .,          ,.
NLS_DATE_FORMAT      DD-MON-YY     YY-MM-DD
NLS_DATE_LANGUAGE    AMERICAN      DANISH
NLS_CHARACTERSET     WE8ISO8859P9  WE8ISO8859P9
NLS_SORT             BINARY        DANISH
```

Observe that the *nls_date_format, nls_numeric_characters, nls_currency* and *nls_iso_currency* take a default value from the territory, whereas *nls_date_language* and *nls_sort* take the default value from the language.

If the following statement is executed: *alter session set nls_date_format='DD/MM-YYYY'* then all nls parameters take the default value except for the date_format, which takes the specified one. Actually this statement may be specified in the **login.sql** file, allowing sqlplus to use this format through the session.

Now, the *to_date* function will use the default date_format, if nothing is specified, so in an American environment the following statement runs like this:

```
select to_date('01-MAY-94') from dual;

TO_DATE('
---------
01-MAY-94
```

But in a Danish environment the result is this:

```
ERROR:
ORA-01858: a letter was found in a date where a number was expected
```

And supplying the date format does not solve the problem:

```
select to_date('01-MAY-94', 'DD-MON-YY') from dual
ERROR:
ORA-01843: not a valid month
```

The correct way of doing this, no matter how the nls values are set, is:

```
select to_date('01-MAY-94', 'DD-MON-YY', 'nls_date_language = American')
from dual

TO_DATE(
--------
94-05-01
```

It is now also possible to display amounts according to the local standards, as this example shows:

```
select to_char(
 3429, 'L99G999D99','nls_numeric_characters=''.'' nls_currency= Kr.')
from dual

TO_CHAR(3429,'L99G99
----------------
     Kr.3.429,00
```

Remember to set the character set to an 8-bit character set, and to set the NLS_LANG to e.g. *danish* in order to be able to store special national characters.

Observe that the *Server Manager* and oracle7 rdbms from version 7.1.6 requires the *ORA_NLS* environment variable to be set to something like: *$(ORACLE_HOME)/ocommon/nls/admin/data.* Make sure that the scripts *oraenv* and *dbstart* does set this variable.

2.8.2 Sorting char columns The character set used on the Supermax is the standard ISO/DIS 8859/1 8-bit character set. If the sequence of your alphabeth agrees with this standard, you may simply use the **order by** clause.

Some languages however uses another sequence. To sort in these languages the **translate** function may be used. Take Danish as an example: In Danish the national letters 'f,x,e' are to be sorted in the specified sequence. But the character numbers in the standard are:

Char	Hex
F	0xC6
X	0xD8
E	0xC5
f	0xE6
x	0xF8
e	0xE5

In order to sort properly we have to move 'E' to a position after 'X':

```
order by
  translate(<col>, 'eEzZ', 'zZeE')
```

If you want upper-case and lower-case letters together the clause may look like this:

```
order by
  translate(upper(<col>),
    'eEzZ', 'ZZEE'), <col>
```

You may also choose to set the *nls-sort = true* line in the **init.ora** file. In that case '**order by** <col>' can be used directly.

2.8.3 On V$.. Views The changed *v$session* view now makes it possible for a running Oracle process to get more information about itself.

The new *audsid* column corresponds to userenv('sessionid'), so it is now easy to get say OS user name, program, attatched terminal, ...

```
select osuser, program, terminal
from v$session
where audsid = userenv('sessionid');

OSUSER  PROGRAM                          TERMINAL
------  -------------------------------  ---------
mj      sqlplus@Arwen (Pipe Two-Task)    ttyq0
```

Some views are by the way difficult to select from, since reserved words are used as column names. (In version 7.0.16.2). They are:

In view *v$archive* the column *current* cannot be selected.

In view *v$recover* the column *online* cannot be selected.

In view *v$type_size* the column *size* cannot be selected.

2.8.4 Using Pipes It is possible to use the pipe package between processes logged into the same Oracle7 instance. Note that the transfer is made instantly, no matter if the transaction is committed or not. Take the following example:

```
set serveroutput on
declare
  status  number;
begin
  dbms_pipe.pack_message('This is a piped message (fxeFXE)');
  status := dbms_pipe.send_message('demo_pipe');
  dbms_output.put_line('message send, res: '||to_char(status));
end;
```

Figure 5. Process A

```
set serveroutput on
declare
  message varchar2(2000);
  status  number;
begin
  status := dbms_pipe.receive_message('demo_pipe');
  dbms_output.put_line('message received, res: '||to_char(status));
  if status = 0 then
    dbms_pipe.unpack_message(message);
    dbms_output.put_line('got >'||message||'<');
  end if;
end;
```

Figure 6. Process B

When process A does not use *NLS_LANG*, the following text is returned by B:

```
message received, res: 0
got >This is a piped message (fxeFXE)<
```

When process B does not use *NLS_LANG*, but process A does (danish_Denmark), the following text is returned by B:

```
message received, res: 0
got >This is a piped message (??????)<
```

When both processes use *NLS_LANG* (danish_Denmark), the following text is returned by B:

```
message received, res: 0
got >This is a piped message (fxeFXE)<
```

2.8.5 Using Alerts The Oracle7 RDBMS allows the programmer to use alerts through procedures in the *dbsm_alert* package.

In brief, it is possible for a process to register interest of one or more alerts, and to wait for one or all of them to happen. Another process may then signal an alert associated with a message either directly or indirectly through database changes. These changes may then trigger alerts to the waiting process.

Please note that signals are passed on to the processes that show interest, when the signalling process does a commit. Also note that signals work across all instances on the same shared database.

It is possible for the database administrator to see part of the alert traffic by monitoring the **dbms_alert_info** table.

An example:

Process A may show interest and wait:

```
execute dbms_alert.register('emp_table_alert');
execute dbms_alert.register('an_other_one');

set serveroutput on
declare
 name    varchar2(2000);
 message varchar2(2000);
 status  number;
begin
 dbms_alert.waitany(name, message, status);
 dbms_output.put_line(name||' got '||message||' taken status: '||to_char(status));
end;
/
```

While process B may directly do the signal:

```
execute dbms_alert.signal('emp_table_alert', 'This is a message');
commit;
```

You might want to set up a demon process showing interest on all alerts, in order to empty the alert_info tables and to be able to trace and debug the handling of alerts.

2.8.6 Handeling LONGs As the upper limit on LONG's (and LONG RAWs) has been raised to 2GB, a number of considerations have to be made.

New external datatypes such as LONG VARCHAR and LONG VARRAW have been introduced in order to cope with the possibility to specify (and get) the actual length of a LONG field. When LONG VARCHAR / LONG VARRAW is used, the first 4 bytes of the buffer may be interpreted as the length. The new PRO*C may in fact be used now in most situations when the LONG values do not get too big.

It is however not easy to store big LONG values into Oracle, because the total LONG value must be known to the system when handed over to Oracle. Do not try to allocate a 2GB chunk of memory on most platforms, or you will (at least) get a very painful swapping / paging problem. In fact, trying to import LONG values will fail if the value requires more storage than specified for the import process (*array fetch buffer size*).

From version 7.3 of the rdbms it is however possible also to do piecewise inserts of long values.

```
IMP-00020: long column too large for column buffer size (30654)
Import terminated successfully with warnings.
```

So you may want to know the 'longest' LONG in the database.

Modifying LONG values is in fact particularly painful, because the whole value has to be stored by the Oracle kernel itself, not only in the redolog, but also in rollback segments. So do use more small LONG values rather than a few big ones. Also observe that the new *discrete* transaction type does not accept LONG management.

It is in fact more easy to fetch big LONG values, since a new OCI routine *oflng* has been introduced to get the LONG value chunk by chunk. And this way we do not have to allocate space for the total value.

Here is an example program to find the length of LONG fields in a LONG column.

```
/* maxlong.c
 * maxlong takes arguments - table_name column_name used/passwd
 * this column is assumed to be LONG or LONG RAW,
 * and are scanned to find the maximum value.
 * The bytes used in the column will be calculated in total.
 */

#include <stdio.h>

#define FALSE 0
#define TRUE  1
#define NON_EXISTENT -942

#define CHUNK_SIZE   30004

typedef unsigned char chunk[CHUNK_SIZE];

char     username[20];                /* null-terminated strings */
char     password[20];

char     rowid[20];

chunk    cbuf;

long     long_get, long_indp, pos, totlen, res;
short    rlen;

short  lda[32], cur1[32], hda[256];

void getdoc();               /* generates an employee doc     */
void cleardoc();             /* generates an employee doc     */
void showdoc();              /* renders doc buffers on screen */

void signoff();                /* handles normal termination      */
void sqlerror();               /* handles unrecoverable errors    */

main(ac,av)
  int ac;
  char **av;
{
    char    table_name[30], column_name[30];
    char    sqlstate[100];
    char    max_rowid[20];
    int         max_totlen, sum_totlen, long_tmp;
    short   dbtype;

    char    reply[10];

    /* Log on to ORACLE. */

    strcpy(table_name, av[1]);
    strcpy(column_name, av[2]);
    strcpy(username, av[3]);
    strcpy(password, "");
```

```
    if (ac > 4) strcpy(password, av[4]);
    sprintf(sqlstate, "select rowid, %s from %s", column_name, table_name);
    strcpy(max_rowid, "");
    max_totlen = sum_totlen = 0;

    if (orlon(&lda[0], &hda[0], username, -1, password, -1, 0))
      goto sql_error;

    if (oopen(cur1, lda, 0, 0, 0, 0, 0))
      goto sql_error;

    if (oparse(cur1, sqlstate, -1, TRUE, 1))
      goto sql_error;

    if (odescr(cur1, 2, &long_tmp, &dbtype, 0, 0, 0, 0, 0, 0))
      goto sql_error;
    if (dbtype == 8 ) dbtype = 94;
    if (dbtype == 24) dbtype = 95;
    if (odefin(cur1, 1, rowid, sizeof(rowid), 1, -1, 0, 0, 0, -1, 0, 0))
      goto sql_error;
    if (odefin(cur1, 2, cbuf, CHUNK_SIZE, dbtype, -1, &long_indp, 0, 0, -1, &rlen,
      goto sql_error;

    if (oexfet(cur1, 1, FALSE, FALSE)) goto sql_error;
    for (;;) {

        totlen = rlen;
        for (res=0, pos=rlen; long_indp!=0;) {
          res = oflng(cur1, 2, cbuf, CHUNK_SIZE, dbtype, &long_get, pos);
          pos += long_get;
          totlen += long_get;
          if (long_get < CHUNK_SIZE -4) break;
          if (res != 0) break;
        }

      printf("\005Nc>Row: %s, Length: %d\n", rowid, totlen);
        sum_totlen += totlen;
        if (totlen > max_totlen) {
        max_totlen = totlen;
          strcpy(max_rowid, rowid);
        }
        if (ofetch(cur1)) break;
    }
    oclose(cur1);
    printf("\nLongest %s.%s is %s, with length: %d, in total: %d\n",
      table_name, column_name, max_rowid, max_totlen, sum_totlen);

notfound:
    ologof(lda);

sql_error:
    sqlerror();
}
```

```
void sqlerror()
{
    if ((lda[12] != 0) || (cur1[12] != 12)) {
      printf("\nORACLE error detected:\n");
      printf("LDA return: %d, Cursor Return: %d\n", lda[12], cur1[12]);
      ologof(lda);
      exit(1);
    }
}
```

Figure 7. maxlong

2.8.7 Leap Year The date *29th of February year 2000* seems to be a special date, since Oracle does not permit a *to_date* to be used on that date:

select to_date(to_date('29-FEB-2000','DD-MON-YYYY')) **from** dual;
ERROR:
ORA-01839: date not valid for month specified.

no rows selected

Whereas the day before works ok:

select to_date(to_date('28-FEB-2000','DD-MON-YYYY')) **from** dual

TO_DATE(

00-02-28

As well as only one *to_date* of that leap year.

select to_date('29-FEB-2000','DD-MON-YYYY') **from** dual

TO_DATE(

00-02-29

2.8.8 Get ORACLE_SID from the Database The x$ table *x$ksqdn* - only accessable from the *sys* account, may be used to give information on the *ORACLE_SID*:

select ksqdngdn **from** sys.x$ksqdn;

If you are going to use it, access the table through a view, and grant the users acces to the view. But note that since this is not documented by Oracle it might be changed in later versions.

2.8.9 Automatic Interval Handling Often start and end dates or times are stored on a per row basis in tables to tell of periods of legality for other information in the records. If it is needed to have an unproken period of time recorded by a number of smaller intervals spanning the whole period.

In fact one should only have start dates (or times), and then let the sequence of records describe the intervals in the period, but redundance here is often handy in order to do fast querying. As always we would like to cover this redundancy from the application, asking the database itself to maintain this redundancy.

Assume we have the following table *interval_test*, where *id* points to an object of a kind, some data, and an interval as a *from_date* and a *to_date*. The primary key would be (id, from_date).

```
create table interval_test (
 id Number not null,
 data Varchar2( 10 ),
 from_date Date not null,
 to_date Date);

alter table interval_test add (
 constraint interval_pk primary key ( id, from_date ));

insert into interval_test values ( 2, 'testing', '01-JAN-96', '01-MAY-96' );
insert into interval_test values ( 2, 'testing', '01-MAY-96', null );
commit;
```

It is now esay to define an insert trigger to update other rows in the table.

```
create or replace trigger interval_mng
before insert on interval_test
for each row
begin
 update interval_test
  set to_date = :new.from_date
  where to_date is null
   and id = :new.id;
end;
```

So when the application dows the following insert, the last record is updated automaticly.

```
insert into interval_test values ( 2, 'testing', '05-MAY-96', null );
commit;

select * from interval_test order by id, from_date;

     ID DATA      FROM_DATE TO_DATE
---------- ---------- --------- ---------
      2 testing   01-JAN-96 01-MAY-96
      2 testing   01-MAY-96 05-MAY-96
      2 testing   05-MAY-96
```

Now an update trigger may be implemented as well, but it will not work since it selects (internally) from the table causing the trigger to fire. (I do not understand why the previously stated insert trigger did not also tried to mutate the table!)

```
create or replace trigger interval_upd
before update on interval_test
for each row
begin
 if ( :old.from_date < :new.from_date ) then
  insert into interval_test ( id, data, from_date, to_date )
    values ( :old.id, :old.data, :old.from_date, :new.from_date );
 end if;
end;

update interval_test set from_date = '03-MAY-96'
 where id = 2 and from_date = '01-MAY-96';

ERROR at line 1:
ORA-04091: table MJ.INTERVAL_TEST is mutating, trigger/function may not see it
ORA-06512: at line 3
ORA-04088: error during execution of trigger 'MJ.INTERVAL_UPD'
```

Now drop the two triggers, and observe that it is the *for each row* trigger that causes the problem as it tries to manipulate the table from where it is called. The trick is here to store this information elsewhere and to do the update in an *after statement level* trigger. It is prefered to store the information needed in PL/SQL package variables since these values are only accessible from the actual session, and that any cleanup activities are done automaticly.

If however arrays are needed to store information of more rows and your current PL/SQL version is lower than 2.2, then the same mechanism may be implemented using an extra database table qualifying each row with the actual session number.

```
create or replace package interval_values as
  type trans_array is table of Varchar2( 1 ) index by Binary_Integer;
  type id_array is table of interval_test.id%type index by Binary_Integer;
  type data_array is table of interval_test.data%type index by Binary_Integer;
  type date_array is table of interval_test.to_date%type
   index by Binary_Integer;

  inx      Number;
  trans    trans_array;
  id       id_array;
  data     data_array;
  from_date date_array;
  to_date   date_array;
end interval_values;

create or replace trigger interval_variable_clear
before insert or update on interval_test
begin
  interval_values.inx := 0;
end interval_variable_clear;

create or replace trigger interval_value_collect
after insert or update on interval_test
for each row
begin
-- Yes I know that validation here is not propor, but ...
  if updating then
    interval_values.inx := interval_values.inx + 1;
    interval_values.trans( interval_values.inx )   := 'I';
    interval_values.id( interval_values.inx )      := :new.id;
    interval_values.data( interval_values.inx )    := :new.data;
    interval_values.from_date( interval_values.inx ):= :old.from_date;
    interval_values.to_date( interval_values.inx ) := :new.from_date;
  end if;
end interval_value_collect;

create or replace trigger interval_post_change
after insert or update on interval_test
begin
  for i in 1 .. interval_values.inx loop
    if interval_values.trans( i ) = 'I' then
      insert into interval_test (id, data, from_date, to_date)
        values (interval_values.id( i ), interval_values.data( i ),
              interval_values.from_date( i ), interval_values.to_date( i ));
    end if;
  end loop;
end interval_post_change;
```

Now we got rid of the mutation error, but the mechanism still does not work!

```
update interval_test set from_date='03-MAY-96'
 where id = 2 and from_date = '01-MAY-96';

ERROR at line 1:
ORA-01000: maximum open cursors exceeded
ORA-06512: at line 3
ORA-04088: error during execution of trigger 'MJ.INTERVAL_POST_CHANGE'
ORA-06512: at line 3
ORA-04088: error during execution of trigger 'MJ.INTERVAL_POST_CHANGE'
ORA-06512: at line 3
```

: : : : : : : : : :

The problem is that the after statement trigger does insert information to the table, which would fire the triggers again, which would insert information to the table, which would ...

Somehow we need to tell the main *interval_value_collect* trigger that the changes from the *interval_post_change* trigger are already accounted for and that this trigger *is a friend.*

```
create or replace package interval_values as
 type trans_array is table of Varchar2( 1 ) index by Binary_Integer;
 type id_array is table of interval_test.id%type index by Binary_Integer;
 type data_array is table of interval_test.data%type index by Binary_Integer;
 type date_array is table of interval_test.to_date%type
  index by Binary_Integer;

 from_friend Boolean := FALSE;
 inx      Number;
 trans    trans_array;
 id       id_array;
 data     data_array;
 from_date date_array;
 to_date   date_array;
end interval_values;

create or replace trigger interval_variable_clear
before insert or update on interval_test
begin
 interval_values.inx := 0;
 interval_values.from_friend := FALSE;
end interval_variable_clear;

create or replace trigger interval_value_collect
after insert or update on interval_test
for each row
begin
 if not interval_values.from_friend then
  if inserting then
    interval_values.inx := interval_values.inx + 1;
    interval_values.trans( interval_values.inx )   := 'U';
    interval_values.id( interval_values.inx )      := :new.id;
```

```
      interval_values.data( interval_values.inx )    := :new.data;
      interval_values.from_date( interval_values.inx ):= :new.from_date;
      interval_values.to_date( interval_values.inx )  := :new.to_date;
    end if;
    if updating then
     if (:old.id = :new.id) and (:old.data = :new.data) and
       (:old.from_date <= :new.from_date) and
       ((:old.to_date >= :new.to_date) or
        (:old.to_date is null) or (:new.to_date is null)) then
       interval_values.inx := interval_values.inx + 1;
       interval_values.trans( interval_values.inx )   := 'I';
       interval_values.id( interval_values.inx )      := :new.id;
       interval_values.data( interval_values.inx )    := :new.data;
       if (:old.from_date < :new.from_date) then
        interval_values.from_date( interval_values.inx ):= :old.from_date;
        interval_values.to_date( interval_values.inx )  := :new.from_date;
       elsif (:old.to_date > :new.to_date) then
        interval_values.from_date( interval_values.inx ):= :new.to_date;
        interval_values.to_date( interval_values.inx )  := :old.to_date;
       end if;
--   else
--    Raise failure
     end if;
    end if;
  end if;
end interval_value_collect;

create or replace trigger interval_post_change
after insert or update on interval_test
begin
 interval_values.from_friend := TRUE;
 for i in 1 .. interval_values.inx loop
  if interval_values.trans( i ) = 'I' then
   insert into interval_test (id, data, from_date, to_date)
    values (interval_values.id( i ), interval_values.data( i ),
         interval_values.from_date( i ), interval_values.to_date( i ));
  end if;
-- if interval_values.trans( i ) = 'U' then
--  update ...
-- end if;
 end loop;
 interval_values.from_friend := FALSE;
end interval_post_change;

update interval_test set from_date='03-MAY-96'
 where id = 2 and from_date = '01-MAY-96';

select * from interval_test order by from_date;

      2 testing   01-Jan-96 01-May-96
      2 testing   01-May-96 03-May-96
      2 testing   03-May-96 05-May-96
      2 testing   05-May-96
```

Note that this mechanism is safe, even if the user sets the *from_friend* variable as the variable is also set in the *before statement level* trigger.

3. Database Administration

3.1 Get a Database Generate Script

In Oracle7 it is possible to generate a script, from which it is possible to recreate the database control files. This script may be changed to form the create script of the database.

alter database backup controlfile to trace;

In the trace file of the process the following statement is found:

```
# The following commands will create a new control file and use it
# to open the database.
# No data other than log history will be lost. Additional logs may
# be required for media recovery of offline data files. Use this
# only if the current version of all online logs are available.
STARTUP NOMOUNT
CREATE CONTROLFILE REUSE DATABASE V70 NORESETLOGS NOARCHIVELOG
    MAXLOGFILES 16
    MAXLOGMEMBERS 2
    MAXDATAFILES 999
    MAXINSTANCES 1
    MAXLOGHISTORY 100
LOGFILE
  GROUP 1 '/wd/sql12/v70/dbs/logV70_1.dbf'  SIZE 500K,
  GROUP 2 '/wd/sql12/v70/dbs/logV70_2.dbf'  SIZE 500K
DATAFILE
  '/wd/sql12/v70/dbs/dbV70_1.dbf' SIZE 15M
;
# Recovery is required if any of the datafiles are restored backups,
# or if the last shutdown was not normal or immediate.
RECOVER DATABASE
# Database can now be opened normally.
ALTER DATABASE OPEN;
```

Now you just have to remove the data files that do not belong to the *SYSTEM* tablespace.

3.2 Total Export / Import

In order to regenerate the database, it is often wise to let the system itself generate the new database, as *all* files, sizes, etc would then be reflected.

The basis of the create script may be generated with the following command:

alter database backup controlfile to trace;

Remenber to get the appropriate character set from the old database before it is too late:

```
select value from v$nls_parameters
 where parameter = 'NLS_CHARACTERSET'
```

Also the rollback segment optimal size is not stored in the export file, and must therefore also be selected from the existing database in order to generate a new one with the same architecture.

On the old database you could therefore execute a script like this to a spool file (*altroll.sql*):

```
select 'alter rollback segment "'||name||
     '" storage (optimal '||to_char(optsize)||
     ');'
from v$rollname n, v$rollstat s
where n.usn = s.usn
  and s.optsize is not null
```

And execute *altroll.sql* when the new database and rollback segments are created.

3.3 Oracle 7 Backup and Archiving

The following section has been partly donated by Oliver Felix Fox from Dansk Data Elektronik A/S. It demonstrates a practical example of how Oracle7 (and as it turns out also Oracle 6) may be managed backupwise, using compression of the archive files in order to reduce time and space on the backup tapes.

Note that the example is written for a database implemented on UNIX-files, and must be modified if raw devices are used. Normally you would inspect the *cpio* calls.

If the database requires 24-hour uptime, the database is run with archiving and backup of the individual tablespaces.

The directory **$ORACLE_ARCHIVE_DIR/arch**, is used by Oracle to receive the archive-files. The files are named: $ORACLE_SID<threadno>.<sequenceno>.arc e.g. DE1.159.arc.

The directory **$ORACLE_ARCHIVE_DIR/bu_arch** contains compressed archive-files. These are ready for backup. The compress utility is called

compress. Moving the archived files to this directory and compressing them is done by the script *mvarch.sh*. The names are <archive_name>.Z .

The script *butables.sh* is used for backing up a given tablespace, called with the tablespace name as argument and optionally as second argument the keyword *rewind*. The script marks the tablespace for backup and creates a copy of the control file, then it copies the file(s) of the tablespace, the backed-up control-file and all compressed archive files in the directory: **$ORACLE_ARCHIVE_DIR/bu_arch** to tape (*cpio*). If the keyword *rewind* is present then the tape is rewound. The archive files are then deleted, and the tablespace is marked as *backup ended*. The idea is to call the script from another, listing the tablespaces that are to be backed up. In the calling script the last call of butables.sh must use the rewind parameter. Look at *bu_first.sh* and *bu_second.sh* for an example. *butables.sh* writes to a log-file called: **$ORACLE_ARCHIVE_DIR/bu_arch/bu_tspc.log**. Herein is listed the name of the tablespace to backup and the output from *cpio*, which should contain the files of the tablespace, a control file and possibly some compressed archive-files. You should once in a while check that the filenames are correct, and that *all* tablespace names are taken into account. Note, that if a new file is added to a tablespace, it will automatically be backed-up by the script.

```
# butables.sh <tablespace_name> (rewind/norewind)
# find the filenames of the given tablespace and mark start of backup
# if rewind is present then rewind the tape. Run script as Oracle
# By off 5-aug-93
SYSTEM_PASS=${SYSTEM_PASS:-system/manager}
SYSTEM_USER=${SYSTEM_USER:-$SYSTEM_PASS}
logfile=$ORACLE_ARCHIVE_DIR/bu_arch/bu_tspc.log

# make a list of the archives which are compressed (and not under compression)
cd $ORACLE_ARCHIVE_DIR/bu_arch
if [ -n "`ls ${ORACLE_SID}*arc.Z 2>/dev/null | tail -1`" ]; then
  for archfile in ${ORACLE_SID}*arc.Z; do
    # orgfile will contain the uncompressed filename
    orgfile=`expr $archfile : ').Z'`
    if [ ! -r $orgfile ]; then # its not under compression
      arch_to_tape="$arch_to_tape $archfile"
    fi
  done #2>/dev/null
fi
# Take a copy of the controlfile, then mark tablespace for backup
ORAENV_ASK=NO . oraenv
sqlplus -s $SYSTEM_USER <<EOF >>$logfile
whenever sqlerror exit 1
set heading off
set feedback off
alter database backup controlfile to '/tmp/${ORACLE_SID}ctrl_bu' reuse;
alter tablespace $1 begin backup;
spool /tmp/bu_$1.files
select file_name
from dba_data_files
where tablespace_name = upper('$1');
spool off
exit 0
EOF
rc=$?
if [ $rc = 1 ]; then
  echo "`date`: Cannot begin backup of $1: CONTINUING" >>$logfile
fi
if [ ! -s /tmp/bu_$1.files ]; then
  echo "`date`: No files in tablespace $1: NO BACKUP" >>$logfile
  exit 1;
fi
# copy the files of tablespace to tape; also copy the controlfile backup
echo "`date`: ---------- Backup $1 ----------" >>$logfile
if [ x$2 = "xrewind" ]; then
  ls $arch_to_tape `cat /tmp/bu_$1.files` /tmp/${ORACLE_SID}ctrl_bu | \
    cpio -ovcBL > $TAPE_DRIVE 2>>$logfile
else
  ls `cat /tmp/bu_$1.files` /tmp/${ORACLE_SID}ctrl_bu | \
    cpio -ovcBL > $TAPE_DRIVE 2>>$logfile
fi
rc=$?
echo "`date`: ---------- Ended $1 ----------" >>$logfile
if [ $rc != 0 ]; then
```

```
  echo "`date`: Cpio of $1 failes: CHECK BACKUP" >>$logfile
  exit 1;
fi
# Remove the compressed archives older than 'ctime' which should be on tape
if [ -n "$arch_to_tape" -a x$2 = "xrewind" ]; then
  #rm $arch_to_tape
  find $arch_to_tape -ctime +2 -exec rm {} \;
fi
# Mark end of backup
sqlplus -s $SYSTEM_PASS <<EOF >>$logfile
whenever sqlerror exit 1
set heading off
set feedback off
alter tablespace $1 end backup;
exit 0
EOF
rc=$?
if [ $rc != 0 ]; then
  echo "`date`: Cannot end backup of $1: CHECK BACKUP" >>$logfile
  exit 1;
fi
rm -f /tmp/bu_$1.files /tmp/${ORACLE_SID}ctrl_bu
```

Figure 8. butables.sh

The script *mvarch.sh* is called by *cron* (root); checks periodically for files in the directory: **$ORACLE_ARCHIVE_DIR/arch**; these files will be moved to the directory: **$ORACLE_ARCHIVE_DIR/bu_arch** and then compressed. The compression reduces the size of the archive files from e.g. 40 Mb to about 5 - 15 Mb. *mvarch.sh* logs its work in the file **$ORACLE_ARCHIVE_DIR/bu_arch/mv_arch.log**.

```
user=`id | cut -f2 -d"(" | cut -f1 -d")"`
bu_dir=$ORACLE_ARCHIVE_DIR/bu_arch
if [ x$user != xroot ]; then
  echo "You must be superuser(root) to run $0!"
  exit 1
fi
cd $ORACLE_ARCHIVE_DIR/arch
if [ ! -r ${ORACLE_SID}*arc ]; then exit
fi
#ulimit 409600
for archfile in ${ORACLE_SID}*.arc; do
 if [ ! -r $archfile ]; then continue
 fi
 set -- `/etc/fuser $archfile 2>/dev/null`
 COUNT=$#
 if [ $COUNT -lt 1 ]; then
   mv $archfile $bu_dir
   echo `date`\\c
   echo ": $archfile moved"\\c
   compress $bu_dir/$archfile
   echo ", compressed"
 fi
done
```

Figure 9. mvarch.sh

The scripts *bu_first.sh* & *bu_second.sh* are called by *cron* (Oracle), alternately every day in a week, except for one day, which is reserved for file-backup. See the section on backup. These two scripts together form the tablespace-backup of the database. If a new tablespace is added to the database, then adjustments must be made to one or both of the scripts. Remember the last call to *butables.sh* must use *rewind*.

```
butablespc.sh SYSTEM
butablespc.sh INDEX_TS1
butablespc.sh ROBAS_TS
butablespc.sh TEMP_TS
butablespc.sh STATIC_TS rewind
#check read tape
rm -f /tmp/bu_second.lis
for i in 1 2 3 4 5; do
 cpio -itvBL < $TAPE_DRIVE >> /tmp/bu_second.lis 2>&1
 rc=$?
 if [ "$rc" != "0" ]; then
  echo "CHECK read failed see file /tmp/bu_second.lis" \
    >>$ORACLE_ARCHIVE_DIR/bu_arch/bu_tspc.log
 fi
done
mt -f $TAPE_DRIVE rewind
```

Figure 10. bu_second.sh

3.3.1 Backup of a given database The archiving-files and tablespaces are backed up using the procedures described in the section about archiving.

6 days a week the tape station $TAPE_DRIVE used for tablespace/archiving backup, giving 3 total cycles of backup. The remaining day is used for a file-system backup. All files in the database server are saved to tape.

The script *filsyst.cpio*, run via *cron* (root), takes the file system backup. The tape is check-read after it is written.

For the server there are currently 31-tapes to be used for backups. These are numbered and a given tape number corresponds to the day of the month it is inserted in the tape-drive. A logbook of the tape-usage resides in the computer room. So, if a given file is to be restored it is necessary to combine the information in the crontab of the users root and Oracle (use the *crontab -l* command as these users) with the log-messages in the file **$ORACLE_ARCHIVE_DIR/bu_arch/bu_tspc.log** to find the date (=tapenumber) when the file was backed up.

Restoring of normal Unix-files is done with the *cpio* utility, for restoring Oracle tablespaces and archives (which are used for recovering) see the section on recovery.

3.3.2 Oracle 7 crashes There isn't currently any system which notifies the system-operators if Oracle has gone down. You may however check if the *PMON* and *SMON* processes are alive for that instance.

Whenever it is detected that Oracle is down, You should do the following, before starting Oracle again:

1. change directory to Oracle log-directory:

cd $ORACLE_HOME/rdbms/log

2. tail the alert-log:

tail -NO_OF_LINES alert_${ORACLE_SID}.log

to check what the error was. In the log there is probably some references to other files (trace-files) in the same directory eg. pmon_937.trc; check these files as well.

3. If an error code of ORA-00600 (internal error) is found or a similar error code, then you ought to contact Oracle support and report the error; they will normally ask some questions about the log/trace-files, so keep them ready.

Oracle support service will tell you if it is ok to start Oracle. If you have any problems contacting Oracle support, then make sure you have at copy of the relevant trace/log-files: from the Oracle log-directory,use:

ls -lrt # reverse time-stamped list of files

copy the alert-log together with the files listed on the last part of the screen, eg: alert_DE.log dbwr_938.trc ora_8752.trc reco_942.trc arch_939.trc lgwr_940.trc pmon_937.trc smon_941.trc # the numbers will be different make sure to at least copy the files which are referenced in the alert_DE.log

4. Start Oracle:

su oracle
sqldba
connect internal
startup

If Oracle won't start, you will have to issue, still within *sqldba*, a: *shutdown* and if that didn't help then: *shutdown immediate* or even a: *shutdown abort* to be able to use the startup-command. Anyway you must use judgement here!

3.3.3 Oracle 7 recovery On the database it is possible to perform a media recovery, due to the fact that the database runs in archive mode and backups of the tablespaces have been made.

Different reasons for recovering, could be:

- Corrupted objects in the database,
- Power failure (if file system files are used for data- and redo files.
- File of tablespace ruined or lost.

In any case Oracle will tell you if tablespaces need recovery, these messages will be in the Oracle logs.

Whenever it is unavoidable to recover a single tablespace or more you must do the following:

1. It is beneficial to run at least two unix-sessions on the unix server.
2. Become Oracle user: *su oracle*

3. Enter the sqldba: *sqldba [lmode=y]*
4. Try to bring the tablespace(s) in question offline:

   alter tablespace <name> *offline*;

   you could also use the menu-driven interface for the above action and several of the following; use whichever method you prefer.
5. Now find the newest backup of the tablespace(s) and copy them in (from tape). Remember that a tablespace can consist of several files, but usually it is on a single file. You must check the backup log to find the correct tape. When found and inserted in the tape-drive, forward skip to the cpio-file using, from another session:

   mt -f $TAPE_DRIVE fsf (NO)

   where NO is the number of files to skip. If the wanted files are in the first cpio-file then just insert the tape, and use:

   cpio <$TAPE_DRIVE (file pattern) # see unix doc.
6. Now issue, in the sqldba:

   recover tablespace <name>;

   sqldba responds with some messages like:

   ```
   ORA-00279: Change 6347.....
   ORA-00289: Suggestion : <archive name>
   ORA-00280: Change 6347.....
   Specify log: {<RET>= suggested | filename | AUTO | FROM .. | CANCEL}
   ```
7. Now find the suggested archive name (from tape) if it is not already present - it could be either:
   - in the directory: $ORACLE_ARCHIVE_DIR
   - on tape (compressed)
   - in the directory: $ORACLE_ARCHIVE_DIR/bu_arch (compressed)

   If need, copy the archive name to the suggested location and if compressed uncompress by:

   uncompress <archive name>.Z

Then press return to indicate the suggested filename, and Oracle will start applying the changes from the archive file to the database. This step is repeated with all the suggested archive names until sqldba responds:

Media recovery complete

If Oracle is not running then start it, see section on crashes.

8. Bring the tablespaces online again:

alter tablespace <name> *online*;

To speed things up a little you should expect that all archive-files from the first suggestion and to the last existing will be needed by Oracle.

To see, at any point in time, which archive files will be needed to perform a recovery of all tablespaces, i.e the whole database, you could issue within sqlplus or sqldba:

```
select thread#,sequence#
from v$log_history
where high_change# >=
 (select min(change#) from v$backup);
```

The thread# and sequence# correspond directly to an archive file, thus showing the possibly needed files.

4. Operating an Oracle7 based Application

4.1 Batch jobs on Oracle7

It is well known, that on-line applications (such as forms), require other kinds of system resources than those needed for executing batch oriented programs (such as reports and loaders), and that the different kinds of activities do not function very well together.

4.1.1 On-line Application Characteristics Very often, an on-line application will wait for user inputs on an already prepared screen, on an already logged-on Oracle7 instance, where often used SQL-statements are already parsed by the kernel, and where the SQL-statements are often tuned to use very few system resources.

On the other side, it is often a requirement that these on-line applications deliver results in seconds, no matter how other users behave.

So the best way to support on-line applications, from the underlying systems point of view, is to make sure that every process ready to run will run within milliseconds, as it may often complete the job without using the whole time slice. Futhermore, the SGA should be large enough to, at least, contain the upper blocks of all important indexes in order to minimize disk I/O.

The amount of simultaneous users are not important here, as only a few of them will be ready to run at a given time.

4.1.2 Batch Application Characteristics A Batch-oriented application will often get its parameters and, without waiting for futher user interactions:

1. spawn a new Oracle7 server,
2. logs on to the database,
3. parses one or more complex SQL-statements,
4. retrieves more blocks from the disk, in order to extract the rows in question,
5. and, eventually, it will logs off the database.

Time is not so critical here and will usually be from minutes to hours, and some reports may even be submitted between the heavy on-line periods.

So the best way to support report applications from the underlying systems point of view is to make sure that when a process is ready to run, it should get as much cpu power as possible, and the complex SQL-statements should be well-tuned (one complex statement is often better than many small fast statements).

Futhermore, as sequential reads are often performed in tables or indexes, a bigger block size is preferable to lower the number of block requests. Generally, more blocks in the buffer cache do not help much, since the likelihood of accessing the same data block in a large database is rather small.

But more simultaneous reports will fight at the central buffer allocation mechanism in the SGA.

4.1.3 Conclusion We see, thus, that report-oriented applications require other system resources than those needed when running on-line applications.

In an ideal situation, one should not run any report or batch process in the on-line period, because any hardware platform may loose its steam, when more reports are executed simultaneously. In fact the number of simultaneous reports that a fast computer can support without loosing on-line response time is rather small - from 1 to 3 depending on the configuration.

The world is indeed not ideal. Modern systems require, of course, the ability to run report and batch jobs in the on-line period, without loosing too much on-line response time.

4.2 Advise

- Start producing periodic resource logs of the central resources of the system and include, if possible, the number of active batch and report processes and the run-queue length.
- At the first glance, one would suggest to run all reports at a lower priority in order to get the on-line activity through the system at all costs. Unfortunately, this action may have the opposite effect, since all Oracle7 processes will be fighting against central resources in the SGA and OS, no matter the priority.
- You may try to get stronger and more powerful hardware platforms, it may buy you some time, but it will not solve the real problem, as one or two simultaneous reports more will degrade that system as well.
- You may set up a parallel server for all your reports and, thus run the on-line system on a separate Oracle7 instance on a seperate CPU. This is a better approach, since too many simultaneous reports will not harm the on-line applications too much. On the other hand, it may be hard to get even small reports through the system, since they will be fighting against the longer reports. And, eventually the on-line users will notice anyway, as disk I/O will get more intense.
- We do advise you to consider if any of the small reports are in fact used as a kind of on-line forms oriented applications. If this is the case, rewriting of these fast reports in forms will, in general reduce the overall resources such as logging on to the database, parsing SQL-statements, etc.
- We do also strongly advise you to impose on the system a job scheduler to ensure an upper limit of simultaneous report and batch jobs are

enshured.

5. Replication

Replication means to support that copies or replica of chosen data elements are automatic kept updated across databases and systems in a network.

5.1 Reasons

There may be many reasons why one would like to have replication support. Some of them are:

- Backup service, as a back office node in the network could automatic get all changes to the databases.
- Statistical service, at one could demand for periodic load of aggregated data for statistical use for management.
- Performance, as search and updates could be performed locally, and changes distributed asynchroniously in periods outside peak hours.
- Consolidating and distributing information across a number of different sites, who needs only part of the total data.

5.2 Mechanisms

A number of different mechanisms may be used in order to achieve the above benefits. But each mechanism has its own characteristics with up and down sides.

- Export / Import of the whole (or part of the) logical database from one database to another. Obviously this only works with a limited amount of data, and only if data does not need to be updated in longer periods.
- Image Backup of the total database, may be faster but more space consuming, but suffers from the same kind of limitations as the export / import approach.
- Using archive logs, may be much faster, as only the changes to a database, can be moved to another site and loaded there. The drawback is that it require strong synchronisation between the actual setup of the databases, and the database being updated cannot be in operation meanwhile. This feature is in version 7.3 labeled *Stand by Database*.

- Parallel server, may support the notion of more access possibilities to the same one and only physical database. The drawback is that the hardware does put strong limits on the actual distance between the instances for that database. Also all database accesses does go to the same disks anyway.
- Read-only snapshots, may provide easy readonly copies of data (even aggregating data on the fly), but may unfortunately not collect any changes.
- Triggers, may be written to actually propagate changes of a table from one site to other sites, as part of the actual online transaction.
- Applications may of course be written to do a sophisticated user oriented replication with proper support for the actual need. But this is usually a very time consuming and expensive task to perform.

Any way, as the application developer would like to see replication as a database service, rather than some facilities they would have to implement and maintain together with the rest of the application, the Oracle Replication Option has been developed for those systems, to deal with most of the drawbacks of the other mechanisms.

It is however important to note that replication is usually not a discipline to go for just because it is exciting, new, trendy or smart, but from distress and because alternatives are worse.

5.3 Oracle Replication Option

The Oracle Replication Option as of version 7.2 includes 2 different techniques to deal with replication: Master to Master and Updatable Snapshots, and these 2 techniques may be mixed.

5.3.1 Master to Master Replication Master to Master Replication does in fact automatic create a number of triggers and packages for all involved tables to extract any change to the tables in question. These changes are inserted to a local deferred queue (being a normal table). Then at proper intervals these change instructions in the deferred queue are transported to the other replication sites using two phase commits. And here these changes are executed to the actual tables with proper and flexible conflict detection and handling.

Master to Master Replication used in fact tree existing Oracle7 features:

- DBMS_JOB to execute the transport of records in the deferred queue at certain periods.
- Two Phase Commit to be sure that change records are inserted in the remote queue as they are deleted from the local one as one transaction.
- Deferred Transactions, the ability to asyncroniously transmit elements of a transaction, not violating the internal sequences in which execution is to be performed
- Database links to connect the different sites together.

Master to Master Replication has a number of advantages and disadvantages:

Master to Master Replication preserves the transaction sequences, and replicates all tables in a group with no horizontal or vertical partitioning. Master to Master Replication does support N-way replication where N is to be kept as a relative small number. And Master to Master Replication puts only a small number of constraints to the application. But Master to Master Replication is hard to administer and maintain.

Master to Master Replication offers automatic conflict detection, and automatic and manual conflict resolution, as the designers and implementers may write special conflict resolution procedures to minimize the manual work to be done. Note here that handling 3 or more Master replication sites mares it more difficult to handle conflicts properly.

5.3.2 Updatable Snapshots Updatable Snapshots works half the way as a Master to Master Replication and the other half as an ordinary read-only snapshot.

When an updatable snapshot is updated, a number of triggers will catch the changes and send them to the master site in the same way it is done with Master to Master Replication (using DBMS_JOB, Two Phase Commit and Deferred Transactions). Note that the change will also appear in the snapshot even it is not validated by the master.

After the master has updates the table being the base of the snapshot, any change will be propagated back onto the local snapshot in a normal read-only snapshot fashion.

Note that read-only snapshots may aggregate data and may select fewer columns and rows, whereas updatable snapshots may only select fewer rows.

Updatable Snapshots has a number of advantages and disadvantages:

Updatable Snapshots does allow for many more sites and does allow for horizontal partitioning, but with more constraints to the application. Updatable Snapshots maintains the transaction towards the master, but not back again to the snapshot. Note however that constraints are not supported on the snapshots or the underlying tables.

5.4 Limitations

Only the replication of tables are automatic supported across replication sites. Other database objects like Sequences and Clusters must be distributed using replication procedures.

There is no LONG or LONG RAW support by either technique, as this datatype is not supported by PL/SQL.

DDL changes can only be distributed when the total system is quiessed, which is a administration intensive job.

Deletes are difficult to handle conflict wise, which is why Oracle recomment that rows are marked deleted rather than actually deleted by the transaction. These rows may later on be deleted by a batch procedure - also replicated accross.

All constraints should be properly named and database objects should be identified within the first 24 characters rather than 30 characters.

Integration with Trusted Oracle7 will come later.

5.5 Challenges

5.5.1 On Development All constraints should have proper names (including 'not null's), which Designer/2000 cannot generate today.

As Sequence generators cannot be replicated, the datamodel should be carefully inspected on where sequences are used to identify objects.

As the time-issue is rather vague in a distributed environment the usage of timestamps should carefully be inspected.

It must be decided which combination of Symmetric replication and updatable snapshots should be used in order to know how much validation is

needed in the client.

As deletes from the database should be performed more gently, a number of views, triggers and procedures may have to be developed to support this.

The customer should agree not to use LONG's, and should also not be asking for Trusted Oracle7 right away.

5.5.2 On Implementation A number of conflict resolution routines does already exist, but a number of specific resolution routines must be written to support the datamodel.

All nodes kept updated with updatable snapshots must be maintained separately at referential integrity and server oriented validation will be weak.

5.5.3 Etc. Tools / scripts for monitoring the running system must be prepared.

Backup procedures must be prepared.

Restore procedures must be prepared.

Procedures for changing the datamodel must be prepared.

5.6 Settings

Some of the Oracle resources would have to be adjusted in order to run Oracle Replication Option. Some are:

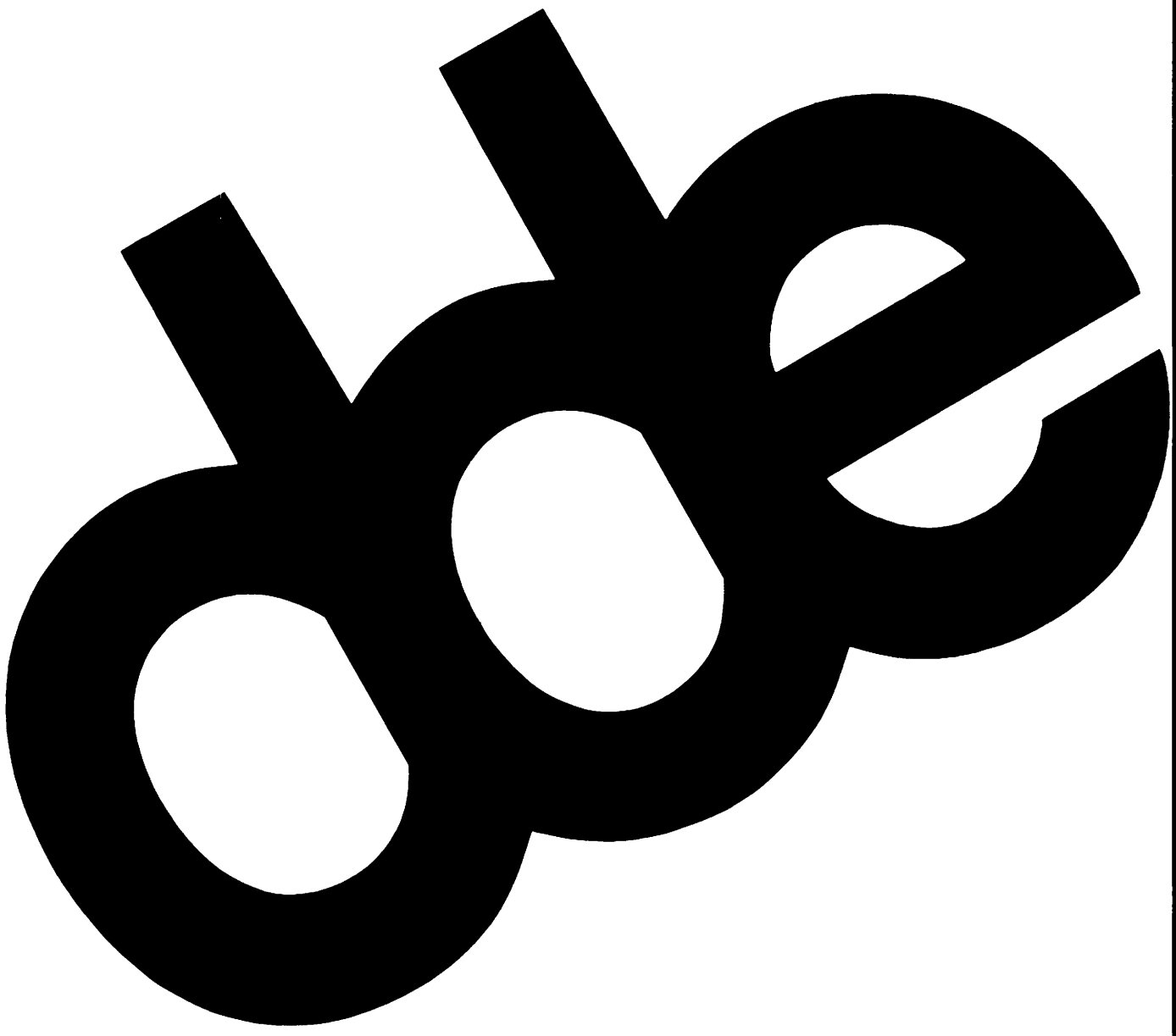
- *compatible* should be set to the version of the Oracle instance ("7.2", "7.3" or whatever)
- *job_queue_processes* at least 2, or the actual number of replication sites.
- *job_queue_interval* around 10, being the number of seconds the demons would suspend, if there is no activity.
- *shared_pool_size* add 15 to 25 Mb to account for large replication packages and a higher number of triggers.
- *distributed_lock_timeout* to 300.
- *distributed_transactions* to 10 to 20.
- *global_names* to true.
- *open_links* to 5.

- *remote_os_authent* to true.

CONTENTS

Dansk Data Elektronik A/S
Herlev Hovedgade 199
DK 2730 Herlev
Tel.: (+ 45) 42 84 50 11
Fax (+ 45) 42 84 52 20

Oracle 7 Hints



© Dansk Data Elektronik A/S
April 1995

Table of contents

Development Environments Using Oracle7

The present paper is written as a contribution to the debate about which ingredients are applicable to a development environment in order to develop good and robust bug fixed programs.

1. Objective

The objective is to make as good use of the features in Oracle7 as possible, and prevent developers from falling into the same traps as others did before them.

2. Programmatic Advice

The following list of advice is by no means complete, and skilled developers will be able to find cases, where the advice is not very applicable. Still, I suggest the advice should be discussed among Oracle7 developers.

2.1 Procedures and Triggers

2.1.1 Use Packages

Instead of writing complex triggers, develop some generic procedures in packages, and call these from the triggers. This will increase the possibility of reusing the code, saving execution time, storage and development effort.

Oracle, Oracle7, PL/SQL, SQL*Forms, SQL*ReportWriter, ORACLE Case, Oracle Forms, and Oracle Report are registered trademarks of Oracle Corporation.

2.1.2 Use before Triggers

Since snapshots require the use of *after row* triggers on all the three actions you should prefer *before* triggers.

2.1.3 Use Integrity Constraints - If Possible

Using Integrity constraints to make sure that foreign keys are valid is indeed a feature which should be utilized.

The following constraint shows how to protect the *emp.deptno* column from getting values not pointing to any legal key in the *dept* table:

```
alter table emp add
  (constraint emp_foreign_key
   foreign key ( deptno ) references dept)
```

You may in fact instruct the kernel to automatically delete all employees from a department, when that department is deleted. This may be accomplished by adding *on delete cascade* to the constraint.

The following example will delete all employees with the corresponding manager being deleted recursively:

```
alter table emp add
  (constraint emp_self_key
   foreign key ( mgr ) references emp on delete cascade )
```

It is, however still not possible to specify *on update cascade*, if we want all the corresponding employees to change their department numbers, when changing a department number. This must be done through triggers, although the construct is defined in the SQL2 standard.

Suppose we want to *help* the kernel to tell how we want an update to be implemented:

```
create trigger upd_dept
 before
 update of deptno on dept
 for each row
 begin
  update emp
   set deptno = :new.deptno
   where deptno = :old.deptno;
end
```

We would see the following messages when trying to modify a department number from the *dept* table:

```
ORA-04091:  table SCOTT.EMP is mutating, trigger may not read or modify it
ORA-06512:  at line 2
ORA-04088:  error during execution of trigger 'SCOTT.UPD_DEPT'
```

Because the trigger and the integrity constraint are in conflict.

We could in fact drop the constraint and implement a trigger instead, but then a *truncate* of the table could violate data integrity.

The first shot could look like this:

```
create trigger ref_dept_after
 after
 update of deptno or delete on dept
 for each row
 begin
  if updating then
   update emp
    set deptno = :new.deptno
    where deptno = :old.deptno;
  end if;
  if deleting then
   delete from emp
   where deptno = :old.deptno;
  end if;
end;
```

```
create trigger ref_emp_before
 before
 update of deptno or insert on emp
 for each row
 when (new.deptno is not null)
 declare
  my_dept dept.deptno%type;

  cursor get_deptno (dn Number) is
   select deptno from dept
   where deptno = dn
   for update of deptno;
begin
 open get_deptno (:new.deptno);
 fetch get_deptno into my_dept;
 if get_deptno%NOTFOUND then
  Raise_Application_Error( -20291, 'Reference violated' );
 end if;
 close get_deptno;
end;
```

Now deleting goes fine, but still updating is troublesome:

```
ORA-04091: table SCOTT.DEPT is mutating, trigger may not read or modify it
ORA-06512: at line 4
ORA-06512: at line 8
ORA-04088: error during execution of trigger 'SCOTT.REF_EMP_BEFORE'
ORA-06512: at line 3
ORA-04088: error during execution of trigger 'SCOTT.REF_DEPT_BEFORE
```

The two triggers conflict, although they should not, because updating the emp table through the dept-trigger will automatically preserve the consistancy. So there is no need to fire the emp trigger, when the change is happening from the *friend*. It could be implemented like this:

```
create package dept_emp_management as

 procedure come_from_friend;
 procedure not_from_friend;

 procedure dept_emp_cascade(
  upd in Boolean, old_key in Number, new_key in Number);

 function dept_for_key_check(new_key in Number) return
 Boolean;
end dept_emp_management;

create or replace package body dept_emp_management as

  from_friend Boolean := FALSE;

 procedure come_from_friend is
 begin
  from_friend := TRUE;
 end come_from_friend;

 procedure not_from_friend is
 begin
  from_friend := FALSE;
 end not_from_friend;

 procedure dept_emp_cascade(
  upd in Boolean, old_key in Number, new_key in Number) is
 begin
  if upd then
   update emp
    set deptno = new_key
    where deptno = old_key;
  else
   delete from emp
   where deptno = old_key;
  end if;
 end dept_emp_cascade;
 function dept_for_key_check(new_key in Number) return Boolean is
  my_dept dept.deptno%type;

  cursor get_deptno (dn Number) is
   select deptno from dept
   where deptno = dn
   for update of deptno;
```

```
 begin
  if dept_emp_management.from_friend then return TRUE; end if;
  open get_deptno (new_key);
  fetch get_deptno into my_dept;
  if get_deptno%NOTFOUND then
   return FALSE;
  end if;
  close get_deptno;
  return TRUE;
 end dept_for_key_check;

end dept_emp_management;

create trigger ref_dept_after
 after
 update of deptno or delete on dept
 for each row
 begin
  dept_emp_management.dept_emp_cascade(updating,:old.deptno,:new.dept);
 end;

create trigger ref_emp_before
 before
 update of deptno or insert on emp
 for each row
 when (new.deptno is not null)
 begin
  if not dept_emp_management.dept_for_key_check
  (:new.deptno) then
   Raise_Application_Error(-20291, 'Reference Violation');
  end if;
 end;

create trigger ref_dept_start
 before
 update of deptno on dept
 begin
  dept_emp_management.come_from_friend;
 end;

create trigger ref_dept_stop
 after
 update of deptno on dept
 begin
  dept_emp_management.not_from_friend;
 end;
```

I have not been able to implement the *on update cascade* feature on a self referencing reference yet, due to mutating conflicts.

2.1.4 Multiple Triggers of the same kind

From version 7.1.3 of Oracle7, multiple triggers of the same type, on the same table, and acting on the same events are allowed. Provided you did specify that you wanted this feature by adding the parameter compatible = '7.1.3' to the parameter file.

This feature is indeed great, since it allows different modules to add their own triggers to the system, regardless of when triggers on the same table, type and kind already exist. (Hereby snapshorts and after rowlevel triggers may now be used together).

There are, however a number of issues to consider when using this feature.

Triggers of the same type and kind on a table are still executed in random order, so one should carefully analyse if the whole sequence delivers the same result regardless of the sequence. Different triggers that manipulate the same rows in foreign tables should therefore be melted into one trigger.

Also the database administrator should remember to add the compatible parameter to any other database if a total import is to be done. Or else only the first trigger will be known to the database kernel.

The following sql-statement will show if a database (prior to total export) utilizes this feature:

```
select decode(t.type,1,'Row level','Statement level'),
  decode(t.update$,1,'for update'),
  max(u.name)||'.'||max(o.name), count(*)
from sys.trigger$ t, sys.obj$ o, sys.user$ u
where t.baseobject = o.obj#
 and o.owner# = u.user#
 and t.update$ = 1 and t.enabled = 1
group by t.type, t.update$, t.baseobject
 having count(*) > 1
union
select decode(t.type,1,'Row level','Statement level'),
  decode(t.insert$,1,'for insert'),
  max(u.name)||'.'||max(o.name), count(*)
from sys.trigger$ t, sys.obj$ o, sys.user$ u
where t.baseobject = o.obj#
 and o.owner# = u.user#
 and t.insert$ = 1 and t.enabled = 1
group by t.type, t.insert$, t.baseobject
 having count(*) > 1
union
select decode(t.type,1,'Row level','Statement level'),
  decode(t.delete$,1,'for delete'),
  max(u.name)||'.'||max(o.name), count(*)
from sys.trigger$ t, sys.obj$ o, sys.user$ u
where t.baseobject = o.obj#
 and o.owner# = u.user#
 and t.delete$ = 1 and t.enabled = 1
group by t.type, t.delete$, t.baseobject
 having count(*) > 1
```

Figure 1. *The multtrg.sql Statement*

2.1.5 Protecting Source in the RDBMS Server

The ability to maintain procedures, functions and packages in the Oracle7 kernel raises a security problem, because a privileged user may change these procedures, thus changing the very core of the application, without making sure that these changes are known to the distributor of the original functions. This will make error tracking and support rather difficult.

First, let us agree that this problem has been with us since the early days of using relational technology, because table

definitions, views, indexes, etc, may have been changed likewise.

Second, the procedure source cannot be encrypted or locked, so we cannot prevent a skilled customer from changing the source.

We can, however save information about the objects in the data dictionary, in such a way that we can easily check if the objects have been changed since they where shipped. Note that the creation date cannot be used, since the customer may reimport the whole lot. We have to log information based on the source directly.

The proposed method works in tree phases: Phase 1 extracts information from the data dictionary about the objects in question (chksum.sql). Phase 2 will compute a checksum for each object (chksum.pl) and prepare a script to insert the information in the database. Phase 3 may run the generated script in order to make the comparison more easy.

The two mentioned scripts should not generally be available at the customer site.

```
set heading off
set verify off
set recsep off
set pagesize 9999
set linesize 255
set feedback off
set long 50000
column secondcol newline
break on objid skip 1
select 'Owner: '||user||
    ', Type: '||Rtrim( type )||
    ', Name: '||Rtrim( name )||
    ', Date: '||To_Char( sysdate,'DD-MON-YYYY HH24.MI.SS' ) objid,
    Rtrim( text )
from user_source
where type in ('PACKAGE', 'PROCEDURE', 'PACKAGE BODY',
'FUNCTION')
order by type, name, line;
```

```
select 'Owner: '||user||
    ', Type: TRIGGER-'||Translate( \Fbrtrim
    (trigger_type),' ','_' )||
    ', Name: '||Rtrim( table_name )||'.'||Rtrim
    ( trigger_name )||
    ', Date: '||To_char( sysdate,
    'DD-MON-YYYY HH24.MI.SS' ) objid, trigger_body
from user_triggers
order by table_name, trigger_name, trigger_type;

select 'Owner: '||user||
    ', Type: VIEW'||
    ', Name: '||Rtrim( view_name )||
    ', Date: '||To_Char( sysdate,
    'DD-MON-YYYY HH24.MI.SS' ) objid, text
from user_views
order by view_name;

select 'Owner: '||user||
    ', Type: PRIMARY_KEY'||
    ', Name: '||Rtrim( t.table_name )||'.'||
    Rtrim( t.constraint_name )||
    ', Date: '||To_Char( sysdate,
    'DD-MON-YYYY HH24.MI.SS' ) objid,
    c.column_name secondcol
from user_constraints t, user_cons_columns c
where t.owner = c.owner
 and t.constraint_name = c.constraint_name
 and t.constraint_type = 'P'
order by t.table_name, t.constraint_name;

select 'Owner: '||user||
    ', Type: REFERENCE'||
    ', Name: '||Rtrim( t.table_name )||'.'|| Rtrim
    ( t.constraint_name )||
    ', Date: '||To_Char( sysdate,
    'DD-MON-YYYY HH24.MI.SS' ) objid,
    t.status, t.delete_rule, c.column_name
from user_constraints t, user_cons_columns c
where t.owner = c.owner
 and t.constraint_name = c.constraint_name
 and t.constraint_type = 'R'
order by t.table_name, t.constraint_name;
```

```
select 'Owner: '||user||
    ', Type: CHECK'||
    ', Name: '||Rtrim( t.table_name )||'.'|| Rtrim
    ( t.constraint_name )||
    ', Date: '||To_Char( sysdate,
    'DD-MON-YYYY HH24.MI.SS' ) objid,
    c.column_name secondcol, t.search_condition
from user_constraints t, user_cons_columns c
where t.owner = c.owner
 and t.constraint_name = c.constraint_name
 and t.constraint_type = 'C'
order by t.table_name, t.constraint_name;

select 'Owner: '||user||
    ', Type: TABLE'||
    ', Name: '||Rtrim( t.table_name )||
    ', Date: '||To_Char( sysdate,
    'DD-MON-YYYY HH24.MI.SS' ) objid,
    t.tablespace_name secondcol, t.cluster_name,
    c.column_name,c.data_type,c.data_length,
    c.data_precision, c.data_scale,
    c.nullable, c.column_id
from user_tables t, user_tab_columns c
where t.table_name = c.table_name
order by t.table_name;

select 'Owner: '||user||
    ', Type: INDEX'||
    ', Name: '||Rtrim( i.table_name )||'.'||
Rtrim( i.index_name )||
    ', Date: '||To_Char( sysdate,
    'DD-MON-YYYY HH24.MI.SS' ) objid,
    i.tablespace_name secondcol, i.uniqueness,
    c.column_name, c.column_position
from user_indexes i, user_ind_columns c
where i.index_name = c.index_name
 and i.table_name = c.table_name
order by i.table_name, i.index_name;

exit
```

Figure 2. *chksum.sql* SQL*Plus Script

```
#! perl
eval "exec /usr/local/bin/perl -S $0 $*"
  if $running_under_some_shell;
##############################################
# chksum.pl 9. jan. 94   MJ DDE
# Usage: chksum.pl list
# VERS="@(#) oracle7.mm 1.1.50.1 4/10/95"
$tmpfile="/tmp/chksum$$";
$sumcom="sum $tmpfile";
printf( "drop table check_sum;\n\n ");
printf( "create table check_sum (owner varchar2(30),
 obj_type varchar2(61),\n ");
printf( "  obj_name varchar2(61), info_date date,
 obj_check number);\n\n" );
$state='first';
while (<>) {
 if (/^Owner: /) {
  if ($state eq 'first') {
   $state = 'not first';
  } else {
    close( SRC );
    $check = `$sumcom`;
    ($check, @rest) = split( ' ', $check );
    printf( "insert into check_sum (owner,
     obj_type, obj_name, info_date, obj_check) values\n" );
    printf( "  ('%s', '%s', '%s',\n", $owner, $type,
     $name );
    printf( "    to_date( '%s',' DD-MON-YYYY HH24.MI.SS' ),
     %s);\n", $date, $check );
    }
   ($owner, $type, $name, $date, @rest) = split( ',' );
   chop( $date );
   ($temp1, $owner, @rest) = split( ':', $owner );
   ($owner, @rest) = split( ' ', $owner );
   ($temp1, $type , @rest) = split( ':', $type );
   ($type , @rest) = split( ' ', $type );
   ($temp1, $name , @rest) = split( ':', $name );
   ($name , @rest) = split( ' ', $name );
   ($temp1, $date , @rest) = split( ':', $date );
   open( SRC, ">$tmpfile" );
 } else {
  s/ *$//;
  print( SRC $_ );
 }
}
```

```
close( SRC );
$check = `$sumcom`;
($check, @rest) = split( ' ', $check );
printf( "insert into check_sum (owner, obj_type, obj_name,
 info_date, obj_check) values\n ");
printf( "  ('%s', '%s', '%s',
 to_date('%s',' DD-MON-YYYY HH24.MI.SS'),%s);\n",
 $owner, $type, $name, $date, $check );
printf( "\ncommit;\n" );
system( 'rm -r $tmpfile' );
```

Figure 3. *chksum.pl* Perl Script

2.1.6 Table Logging

It is often desirable to keep a log on changes of important tables in order to be able to track more historical information on who changed what when.

Often the application developer should not be involved in this process, since different applications may address the same set of tables. It is thus an advantage if the logging procedures can be stored and maintained within the database system close to the tables in question.

This is in many cases possible to achieve through triggers as the following example will suggest.

Assume that we have an important table *ATable* with the columns *aColumnA* and *aColumnB*.

We could then specify the following trigger:

```
create trigger ATableLogTrg
 before
 delete or insert or update
 on ATable
 for each row
 declare
  act Varchar(10);
 begin
  if inserting then
   act := 'INSERT';
  end if;
  if updating then
   act := 'UPDATE';
  end if;
  if deleting then
   act := 'DELETE';
  end if;
  insert into ATableLog (theColumnA, theColumnB,
  newColumnA, newColumnB,
    whenEvent, byWho, action)
   values (:old.aColumnA, :old.aColumnB,
   :new.aColumnA, :new.aColumnB,
    sysdate, user, act);
end;
```

A possible look at the table **ATableLog** could show:

```
A     B N NB WHENVEN   BYWHO   ACTION
- ---- - -- --------- ------- -------
      a    1 12-APR-94 OPS$MJ  INSERT
      b    2 12-APR-94 OPS$MJ  INSERT
      x   11 12-APR-94 OPS$MJ  INSERT
      x   12 12-APR-94 OPS$MJ  INSERT
x 11  u   11 12-APR-94 OPS$MJ  UPDATE
b  2          12-APR-94 OPS$MJ  DELETE
```

Note that you could add a statement trigger to insert a row in a central historical table telling that changes in the **ATable** did take place.

2.1.7 Unique Index is also a kind of Constraint

Let us examine the following senario. First we create a table with a primary key constraint:

```
create table t1 (
 a number,
 b varchar2(10),
 constraint key_ix primary key (a));
```

Examine from the dictionary that the constraint is known:

```
CONSTRAINT_NAME TABLE_NAME C COLUMN_NAME
--------------- ---------- - -----------
KEY_IX          T1         P A
```

Now insert 2 identical rows, and you will get the following error message:

```
ORA-00001: unique constraint (SCOTT.KEY_IX) violated
```

Now drop the constraint and create a unique index instead:

```
alter table t1 drop constraint KEY_IX;
create unique index ix on t1(a);
```

Now when two identical rows are inserted the following message appears:

```
ORA-00001: unique constraint (SCOTT.IX) violated
```

Looking for constraints on **t1** shows none at all. The conclusion is therefore that unique constraints are either declared explicitly (and found in the constraint views), or declared implicitly (and found in the index tables).

2.1.8 Remember Constraint Info Enabling a Constraint

Assume the following table is created, and that the exception table **exceptions** does exist (use @/rdbms/admin/utlexcpt). Note that the pctfree parameter is specified for the underlying index.

```
create table t1 (
 a number,
 b varchar2(10),
 constraint key_ix primary key (a) using index pctfree 88
  exceptions into exceptions,
 constraint a_check check (a > 10)
  exceptions into exceptions)
```

Now try to insert some rows violating the constraints.

```
SQL> insert into t1 values (1, 'test');
ORA-02290: check constraint (SCOTT.A_CHECK) violated

SQL> insert into t1 values (11, 'test');

SQL> insert into t1 values (11, 'test2');
ORA-00001: unique constraint (SCOTT.KEY_IX) violated
```

The exceptions table is empty though! Disable the two constraints and insert the 3 rows again, and try now to enable the constraints:

```
SQL> alter table t1 enable constraint key ix;
ORA-02299: cannot add or enable constraint (SCOTT.KEY_IX)-
           duplicate keys found
```

```
SQL> alter table t1 enable constraint a_check;
ORA-02296: cannot enable constraint (SCOTT.A_CHECK) -
           found noncomplying values
```

Still nothing is found in the exceptions table! The exceptions into exceptions

```
ROW_ID              OWNER TABLE_NAME CONSTRAINT
------------------- ----- ---------- ----------
00000625.0000.0004 SCOTT T1          KEY_IX
00000625.0002.0004 SCOTT T1          KEY_IX
00000625.0001.0004 SCOTT T1          A_CHECK
```

Now delete the rows and enable the constraints again with the exceptions clause, and look for indexes on the **T1** table:

```
SQL> select * from user_indexes where table_name = 'T1';

INDEX_NAME                     TABLE_OWNER
------------------------------ ---------------------
TABLE_NAME                     TABLE_TYPE  UNIQUENES
------------------------------ ----------- ---------
TABLESPACE_NAME                INI_TRANS   MAX_TRANS INITIAL_EXTENT NEXT_EXTENT
------------------------------ ----------- --------- -------------- -----------
MIN_EXTENTS MAX_EXTENTS PCT_INCREASE PCT_FREE BLEVEL LEAF_BLOCKS
----------- ----------- ------------ -------- ------ -----------
DISTINCT_KEYS AVG_LEAF_BLOCKS_PER_KEY AVG_DATA_BLOCKS_PER_KEY CLUSTERING_FACTOR
------------- ---------------------- ---------------------- -----------------
STATUS
------
KEY_IX                         SCOTT
T1                             TABLE       UNIQUE
TOOLS                                    2         255          20480       20480
          1         249           50         10

VALID
```

The pct_free parameter was forgotten and storage information should thus be specified again at enable time.

2.1.9 Embedded SQL from PL/SQL procedures

From version 7.1 of Oracle7 it is possible to run dynamic sql-statements from the general package called dbms_sql. This may be used to implement more general procedures and packages, and will make life more easy for those who wish to store total (or partly) sql-statements in tables.

The following small function example returns the number of rows on a table given as argument:

```
create or replace function count_rows(s_table in varchar2)
return integer is

 -- This procedure counts rows from a given table
 count_cursor integer;
 rows integer;
 fetched_rows integer;
 rows_processed integer;

begin
 -- prepare a cursor to select number of rows from the
 -- table
 count_cursor := dbms_sql.open_cursor;
 dbms_sql.parse( count_cursor,
          'select count(*) from ' || s_table,
          dbms_sql.native );
 dbms_sql.define_column( count_cursor, 1, rows );
 rows_processed := dbms_sql.execute( count_cursor );
 fetched_rows := dbms_sql.fetch_rows( count_cursor );

 -- get column values of the row
 dbms_sql.column_value( count_cursor, 1, rows );
 dbms_sql.close_cursor( count_cursor );
```

```
 -- return rows;
 return( rows );
exception
 when others then
  if dbms_sql.is_open( count_cursor ) then
   dbms_sql.close_cursor( count_cursor );
  end if;
  return( -1 );
end;
```

Figure 4. The *count_rows* Function

2.1.10 User defined SQL Functions

From version 7.1 of the Oracle kernel it is possible to define functions in PL/SQL, and to use these functions directly in the SQL-statements. The following example validates if a danish social number could pass a modulus 11 check.

```
create function is_cpr( cpr in char ) return integer as
begin
 if length( cpr ) != 11 then return 1; end if;
 if substr( cpr,7,1 ) != '-' then return 1; end if;
 if rtrim( ltrim( translate( cpr,'0123456789','
          '))) != '-' then
  return 1;
 end if;
 if ( to_number( substr( cpr,1,1 ) ) * 4 +
     to_number( substr( cpr,2,1 ) ) * 3 +
     to_number( substr( cpr,3,1 ) ) * 2 +
     to_number( substr( cpr,4,1 ) ) * 7 +
     to_number( substr( cpr,5,1 ) ) * 6 +
     to_number( substr( cpr,6,1 ) ) * 5 +
     to_number( substr( cpr,8,1 ) ) * 4 +
     to_number( substr( cpr,9,1 ) ) * 3 +
     to_number( substr( cpr,10,1 ) ) * 2 +
     to_number( substr( cpr,11,1 ) ) ) mod 11 = 0 then
  return 0;
 else
  return 1;
 end if;
end;
```

```
select c from cpr_test
where is_cpr( c ) = 0;

create view cpr as
 select c from cpr_test
 where is_cpr( c ) = 0;
```

If you try to redefine a standard function, you might find it hard to call the user defined version without prefixing with the schema name.

```
create function trunc( n in number, m in number ) return
number as
begin
-- This is a silly reimplementation of trunc - only used as -
-- an example.
 if m = 0 then
-- return trunc( n, 0 ); would a recursive call to the --
-- function itself
-- stoppable with ctrl-c
   return n - mod( n, 1 );
 else
   return n;
 end if;
end;

select trunc( 3.1234, 0 ), trunc( 3.1234, 1 ),
 trunc( 3.1234, 2 ) from dual;


TRUNC(3.1234,0) TRUNC(3.1234,1) TRUNC(3.1234,2)
--------------- --------------- ---------------
              3             3.1            3.12


select scott.trunc( 3.1234, 0 ), scott.trunc( 3.1234, 1 ),
 scott.trunc( 3.1234, 2 ) from dual;

SCOTT.TRUNC(3.1234,0) SCOTT.TRUNC(3.1234,1) SCOTT.TRUNC(3.1234,2)
--------------------- --------------------- ---------------------
                    3                3.1234                3.1234
```

Even with a public synonym, the built-in functions are preferred.

2.2 Snapshots

Snapshots are easy to create. This one should be updated by Oracle every day at 1 pm:

```
create snapshot scott.dept_sal
 refresh fast next Trunc( sysdate ) + 13/24 as
  select d.deptno, Sum( e.sal ) sum_sal
  from scott.dept d, scott.emp e
  where d.deptno = e.deptno(+)
  group by d.deptno;
```

Note that the next expressions are rather limited. If you want to say every 10 seconds you would probably specify:

```
Trunc( sysdate )+( To_Number( To_Char( sysdate,'SSSSS' ) )+100)/86400
```

But then you would get the error:

```
ORA-01480: trailing null missing from STR bind value
```

Even snapshots based on complex views may be queried using rowid, since Oracle7 builds a read-only table for the snapshot.

```
select rowid, deptno, sum_sal from dept_sal;

ROWID                  DEPTNO  SUM_SAL
------------------     ------  -------
000016F5.0000.0001     10      27048
000016F5.0001.0001     20      39520
000016F5.0002.0001     30      13100
000016F5.0003.0001     40
```

Note that the snapshot is made of two views and one read-only table:

```
select * from tab;

DEPT_SAL                    VIEW
MVIEW$_DEPT_SAL             VIEW
SNAP$_DEPT_SAL              TABLE
```

You may select from the view user_snapshots to obtain more information on the snapshot.

Please, note that you could use snapshots to collect aggregate data locally or remotely on a regular basis, but Oracle7 does not guarantee the contents of the snapshot to be consistent with the original tables at any time. Neither does Oracle7 guarantee two table snapshots from two different tables to be consistent, since one transaction updating the two original tables will not propagate the changes to the snapshots as one transaction. Oracle has promised to support transaction autonomy across replication later in version 7. This version will not use the snapshot facility, but underlying system packages for Remote Oracle Procedure Calls as well as some sort of job control.

2.3 Management

2.3.1 Version Control

Using procedures and functions on the database server side instead of in the application on the client side, requires a lot more discipline by the developers.

If e.g. some central procedures are used by different applications, how do we make changes to them without having to inspect all the applications immediately?

You may from the very beginning add a version parameter to the procedure, in order at be able to add new versions to the implementation of the procedures, like this:

```
create package emp_mgmt as
 type emps_type is table of emp.empno%type index by binary_integer;

 function max_sal (version in char, default_sal in number) return number;
 procedure sal_raise (version in char, pct in number);
 procedure get_dept (version in char, deptno_par in number,
  emps out emps_type);
end emp_mgmt;

create package body emp_mgmt as
 version_not_ready exception;
 pragma exception_init(version_not_ready, -1096);
-- We are here using the Oracle error 1096 with the
-- following generic text:
-- program version (%s) incompatible with instance (%s)
-- Although the parameters are not filled yet.
```

```
 function max_sal (version in char, default_sal in number)
 return number is
  max_sal_var emp.sal%type;
 begin
  if version = '1.0' then
   select max(nvl(sal,default_sal) + nvl(comm,0))
   into max_sal_var
   from emp;
   return(max_sal_var);
  else
   raise version_not_ready;
  end if;
 end;

 procedure sal_raise (version in char, pct in number) is
 begin
  if version = '1.0' then
   update emp
    set sal = sal + sal*pct/100;
  elsif version = '1.1' then
   update emp
    set sal = nvl(sal + sal*pct/100, 100);
  else
   raise version_not_ready;
  end if;
 end;

 procedure get_dept (version in char,
             deptno_par in number,
             emps out emps_type) is
  i binary_integer := 0;

 begin
  if version = '1.0' then
    for emprec in (select empno from emp where deptno = deptno_par)
     loop
      i := i + 1;
      emps(i) := emprec.empno;
    end loop;
   else
   raise version_not_ready;
  end if;
 end;

end emp_mgmt;
```

2.3.2 Adapt a Naming Convention

In order to distinguish local PL/SQL variables from database object names, always prefix the local PL/SQL variables with something. (like *Local, p_, ...*)

2.3.3 Tracking dependencies

As more types of objects are handled by the Oracle kernel, there is a stronger need to be able to check which objects depend on others.

The following series of action shows how this may accomplished.

```
connect scott/tiger

create table base_table (
 key number, data varchar2( 200 ) );

create trigger base_before
 before
 update or insert on base_table
 for each row
 begin
  if :new.key = 0 then
     :new.key := 999;
  end if;
 end;

create function count_base_rows return number is
 rows number;
begin
 rows := 0;
 select count( * ) into rows from base_table;
 return rows;
end;

create view base_view as
 select * from base_table;

grant select on base_view to system;

connect system
```

```
create synonym foreign_syn for scott.base_view;

create view foreign_view as
 select * from foreign_syn;
```

First create a package to handle this, either from *rdbms/admin/utldtree*, or the following from the system user account.

```
create table audit_dependencies (
 object_id number not null );

create package audit_dep as
 procedure clear_dependencies;
 procedure new_dependency(
  fp_object_type in varchar2,
  fp_object_name in varchar2,
  fp_schema_name in varchar2);
end audit_dep;

create or replace package body audit_dep as

 procedure clear_dependencies is
 begin
  delete from audit_dependencies;
  commit;
end;

procedure new_dependency(
 fp_object_type in varchar2,
 fp_object_name in varchar2,
 fp_schema_name in varchar2) is
 l_object_id number;
```

```
begin
 select object_id into l_object_id from all_objects
  where owner         = upper( fp_schema_name )
  and   object_name   = upper( fp_object_name )
  and   object_type   = upper( fp_object_type );
 insert into audit_dependencies ( object_id ) values ( l_object_id );
 commit;
 insert into audit_dependencies ( object_id )
  select object_id
   from public_dependency
   connect by prior object_id = referenced_object_id
   start with referenced_object_id = l_object_id;
 commit;
exception
 when no_data_found then
  raise_application_error(-20000, 'ORU-10013:
  '||fp_object_type||'
  '||fp_schema_name||'.'||fp_object_name||
   ' was not found.');
end;

end audit_dep;
```

If we want to see which objects depend on e.g. base_table owned by scott, the following statement may be executed.

```
execute audit_dep.new_dependency( 'table', 'base_table',
'scott' );

select * from audit_dependencies;
    4649
    4650
    4652
    4671

select distinct u.user_id, d.object_id,
    '('||o.owner||'.'||o.object_name||') ('||o.object_type||')' object
from audit_dependencies d, dba_objects o, dba_users u
where o.object_id = d.object_id
 and u.username = o.owner
```

USER_ID	OBJECT_ID	OBJECT
8	4649	(SCOTT.BASE_TABLE) (TABLE)
8	4650	(SCOTT.BASE_VIEW) (VIEW)
8	4671	(SCOTT.COUNT_BASE_ROWS) (FUNCTION)
5	4652	(SYSTEM.FOREIGN_VIEW) (VIEW)

2.4 Tuning

2.4.1 Use Explicit Cursors

If you know that a certain select does not return more than one row, use an explicit cursor, because an implicit cursor will always execute two fetches in order to get the *NO_MORE_ROWS* flag back.

2.4.2 Cache the SQL-Statements

The *shared_pool* with a size that is set by the init parameter *shared_pool_size*, now holds a cache of already parsed SQL statements. If these entries are flushed too often, the pool should be larger.

Because an exact literal match is required for two SQL statements to use the same slot, it is advisable to put the SQL-statements in a library or include file to maximize the reusability of the parse information.

The package *dbms_shared_pool* may help you in getting a list of big elements, as well as keeping often used elements from being invalidated.

To get a list of elements of more than 30K each:

```
set serveroutput on size 20000
execute dbms_shared_pool.sizes(30);
```

```
SIZE(K) KEPT    NAME
-------         ----    ------------------------------------------------
178                     SYS.DBMS_SQL                    (PACKAGE BODY)
143                     SYS.STANDARD                    (PACKAGE)
48                      SYS.DBMS_SQL                    (PACKAGE)
37                      SELECT UPDATE_PRIV FROM TABLE_PRIVILEGES WHERE
                        TABLE_NAME=:b1AAND UPDATE_PRIV='A'
                        (20171020,2100549302)     (CURSOR)
37                      SELECT TO_CHAR(SHARABLE_MEM / 1000 ,'999999') SZ,
                        DECODE(KEPT_VERSIONS,0,'      ',RPAD('YES(' ||
                        TO_CHAR(KEPT_VERSIONS)|| ')' ,6)) KEEPED,
                        RAWTOHEX(ADDRESS) || ',' ||TO_CHAR(HASH_VALUE) NAME,
                        SUBSTR(SQL_TEXT,1,354) EXTRA FROM V$SQLAREA
                        WHERE SHARABLE_MEM > :b1 * 1000   UNION SELECT
                        TO_CHAR(SHARABLE_MEM / 1000 ,'999999') SZ,
                        DECODE(KEPT,'YES','YES(20587D3C,-474114044) (CURSOR)
31                      SYS.STANDARD                    (PACKAGE BODY)
```

Note, the view *v$plsarea* to look at the program unit cache cannot be found. (In version 7.0.16.2)

2.4.3 Hints to the optimizer

Not only is it possible to ask the optimizer to use a cust based approch instead of the rule based. It is also possible in specific SQL-statements, to give a direct hint to the optimizer on the purpose of the statement. It is however also possible to tell the optimizer which index should drive the main select, but this facility should be used with great care, since one never knows when the specified index will be removed or called something else.

2.5 Transactions

2.5.1 Package Variables

Variables in packages do have values owned by the sessions using the packages. It is thus not possible directly to have one session change a variable to a value readable by another session.

Take the following small package as an example:

```
create package var_test as

 procedure add (a in number);

end var_test;
/

create package body var_test as

 pkg_var number := 0;

 procedure add (a in number) is
 begin
   dbms_output.put_line('add before: '||to_char(pkg_var));
   pkg_var := pkg_var + a;
   dbms_output.put_line('add after : '||to_char(pkg_var));
 end;

end var_test;
/
```

Using this package from *sqlplus* will look like this no matter if other sessions are using the same package, and no matter if they commit or rollback:

```
set serveroutput on
SQL> execute var_test.add(4);
add before: 0
add after : 4
```

```
PL/SQL procedure successfully completed.

SQL> execute var_test.add(4);
add before: 4
add after : 8
```

2.5.2 Discrete Transactions

Discrete transactions are only possible if the parameter *discrete_transactions-_enabled* is TRUE in the init.ora file. And the procedure *dbms_transaction .begin_discrete_transaction* has to be called as the first call to the database in order for every transaction to be discrete.

Please note, however, that the call does not fail if the session parameter was not TRUE, so you should check this parameter in the application itself:

```
EXEC SQL select value into :val from V$PARAMETER
   where name = 'discrete_transactions_enabled';

if (!strcmp(val,"TRUE") {
  EXEC SQL EXECUTE
   begin
      dbms_transaction.begin_discrete_transaction;
   end;
  END-EXEC;
}
```

Unfortunately LONG values cannot be manipulated in discrete transactions.

2.6 Migration

Remember that the goal of having only one source master makes it very difficult to migrate the application to Oracle7, since they should in one way or other still be able to function well connected to Oracle 6.

As the following sections will indicate, some adjustments of the SQL-statements might be nessesary in order to migrate an application to Oracle7, although most applications would see no immediate changes at all.

2.6.1 Which Version

In order to exercise new features in Oracle7, and still be able to run the same application against Oracle version 6, it is desirable to be able to identify the kind of database currently connected to.

One might want to see if some new views exist, such as *v$version*, but this might give a misleading answer, since some Oracle7 accounts may not have access to these views at all.

It is thus more safe to test if one of the new *all_views* exists. Use e.g. *all_errors or all_source*, since they are probably not present in the Oracle version 6 account.

Check if the parallel server option is used (the instance has been started in shared mode), by calling the routine *dbms_utility.is_parallel_server* returning TRUE or FALSE.

2.6.2 Char's and Long's

In Oracle7 the *char* datatype means fixed-length strings, whereas varchar has similar capabilities as *char* from Oracle 6. So be careful if you are using the *char* datatype in Oracle7, you should probably be using *varchar* or more likely varchar2

instead. And by the way, *varchar* columns may now consist of up to 2000 characters.

In Oracle7 the *long* may store up to 2Gb of data, so do not think of having the total long column value in memory, unless you know a reasonable upper limit to the size of the structures.

2.6.3 Outer-Joins versus OR / IN

Oracle7 is more restrictive on certain combinations of outer joins and OR or IN operations as the following small example shows:

Suppose we want to have a list of all departments, and if the given department has managers or the president, let us have their names as well. This could in Oracle 6 be expressed in the following way:

```
select dname, ename, job
from emp, dept
where dept.deptno = emp.deptno (+)
 and emp.job(+) in ('MANAGER','PRESIDENT')
```

Producing the following result:

```
DNAME          ENAME  JOB
-----------    -----  ---------
ACCOUNTING     CLARK  MANAGER
ACCOUNTING     KING   PRESIDENT
RESEARCH       JONES  MANAGER
SALES
OPERATIONS
```

But in Oracle7 the following error message appears:

```
ORA-01719: outer join operator (+) not allowed in operand
of OR or IN
```

Here the statement could be expressed in the following way:

```
select dname, ename, job
from emp, dept
where dept.deptno = emp.deptno
 and emp.job in ('MANAGER','PRESIDENT')
union all
select dname, null, null
from dept
where dept.deptno not in (
  select deptno from emp
  where job in ('MANAGER','PRESIDENT'))
```

If the same statement should be used in Oracle 6 and in Oracle7 do not use the *all* clause, and see if there are other ways to express the query since *union* would initiate an internal sort.

2.6.4 Week numbers

As the Americans and Europeans do not agree on how week numbers should be calculated, we used to use the *WW* format in Oracle 6 in combination with the *LANGUAGE* parameter. In Oracle7 however, the *WW* format always means the American way, whereas *IW* follows the *nls_territory* parameter.

Likewise you should use *IYYY, IYY, IY* and I instead of *YYYY, YYY, YY* and *Y*, if you want the local way of calculating the year instead of the American way.

In Oracle 6 without any NLS setting in the init.ora file:

```
select to_char(to_date('01-JAN-93'),'WW YYYY')
from dual;

TO_CHAR(TO_DATE('01-JAN-93'),'WWYYYY')
--------------------------------------
01 1993
```

In Oracle7 with *nls_territory* = Denmark:

```
select to_char(to_date('01-JAN-93','DD-MON-YY'),'WW YYYY / IW IYYY')
from dual

TO_CHAR(TO_DATE('01-JAN-93','DD-MON-YY'),'WWYYYY/IWIYYY')
---------------------------------------------------------
01 1993 / 53 1992
```

Please note that the example also indicates that Oracle7 is more restrictive regarding type conversion as '01-JAN-93' is not considered a date any more but a string. The corresponding date may be specified like this: *to_date('01-JAN-93','DD-MON-YY').*

2.6.5 Error Codes

A number of error codes have been added to version 7.0 and 7.1, but some error codes have also been eliminated. Naturally this is to be considered when migrating applications into newer Oracle RDBMS versions. The following list of error codes are known to be eliminated in version 7.1.4 or before.

2.6.5.1 4040 Removing 4040.

```
04040, 00000, "new timestamp is not greater than existing one"
// *Cause:  The given timestamp is not greater than the
//      current timestamp of the existing object.
// *Action: Specify a greater timestamp.
```

2.6.5.2 1930 Removing 1930.

```
01930, 00000, "no privileges to REVOKE"
// *Cause:  "ALL" was specified when there were no
//      privileges to revoke.
// *Action: Don't try revoking from that user.
```

2.6.5.3 960, 962 Removing 110 and adding 960, 962.

```
00960, 00000, "ambiguous column naming in select list"
// *Cause: A column name in the order-by list matches more
//      than one select list columns.
// *Action: Remove duplicate column naming in select list.

00962, 00000, "too many group-by / order-by expressions"
// *Cause: The group-by or order-by column list contain
//       more than 255 expressions.
// *Action: Use 255 or less expressions in the group-by or
//      order-by list.
```

2.6.5.4 1547 Adding 1650, 1651, 1652, 1653, 1654 and removing 1547.

```
01547, 00000, "failed to allocate extent of size %s in
tablespace '%s'"
// *Cause: Tablespace indicated is out of space
// *Action: Use ALTER TABLESPACE ADD DATAFILE statement to
//      add one or more files to the tablespace indicated
//      or create the object in other tablespace if this
//      happens during a CREATE statement

01650, 00000, "unable to extend rollback segment %s by %s
in tablespace %s"
// *Cause:  Failed to allocate an extent for rollback
//      segment in tablespace.
// *Action: Use ALTER TABLESPACE ADD DATAFILE statement to
//      add one or more files to the tablespace indicated.

01651, 00000, "unable to extend save undo segment by %s
for tablespace %s"
// *Cause:  Failed to allocate an extent for saving undo
//      entries for the indicated offline tablespace.
// *Action: Check the storage parameters for the SYSTEM
//      tablespace. The tablespace needs to be brought
//      back online so the undo can be applied.
```

```
01652, 00000, "unable to extend temp segment by %s in
tablespace %s"
// *Cause:  Failed to allocate an extent for temp segment
//      in tablespace.
// *Action: Use ALTER TABLESPACE ADD DATAFILE statement to
//      add one or more files to the tablespace indicated
//      or create the object in other tablespace.

01653, 00000, "unable to extend table %s.%s by %s in
tablespace %s"
// *Cause:  Failed to allocate an extent for table segment
//      in tablespace.
// *Action: Use ALTER TABLESPACE ADD DATAFILE statement to
//      add one or more files to the tablespace indicated.

01654, 00000, "unable to extend index %s.%s by %s in
tablespace %s"
// *Cause:  Failed to allocate an extent for index segment
//      in tablespace.
// *Action: Use ALTER TABLESPACE ADD DATAFILE statement to
//      add one or more files to the tablespace indicated.

01655, 00000, "unable to extend cluster %s.%s by %s in
tablespace %s"
// *Cause:  Failed to allocate an extent for cluster
//      segment in tablespace.
// *Action: Use ALTER TABLESPACE ADD DATAFILE statement to
//      add one or more files to the tablespace indicated.
```

2.6.5.5 1596 Removing 1596.

```
01596, 00000, "fail to coalesce extents because sort area
size is too small"
// MERGE: 1591 RENUMBERED TO 1596
// *Cause:  there are too many entries of free extents to
//      sort it in the in-memory sort area
// *Action: increase the sort area size or reduce the
//      fragmentation in tablespace by doing a full export
//      followed by an import.
```

2.6.5.6 481, 482 Removing 481 and 482.

```
00481, 00000, "SMON process posting itself"
// *Cause: This is trapped internally
// *Action: None

00482, 00000, "SMON shut, shutdown abort required"
// *Cause: Instance or Transaction recovery was required
//      after the SMON process was stopped in anticipation
//      of shutdown.  This is a rare condition that can
//      occur only in multi-instance operation.
// *Action: Use shutdown abort.  This instance will be
//      recovered by surviving instance(s) or by warm-
//      start recovery.
```

2.6.5.7 983 Removing 983.

```
00983, 00000, "cannot define ROWID column - no
corresponding SQL datatype"
// *Cause:  Attempt to create a table with a rowid column
//      as in "CREATE TABLE ... AS SELECT ROWID ... FROM
//      ..." which is illegal because there is no
//      corresponding SQL datatype for rowid.
// *Action:  Use the function ROWIDTOCHAR to convert rowid
//      to character as in "CREATE TABLE ... AS SELECT
//      ROWIDTOCHAR(ROWID) ... FROM ...".
```

2.6.5.8 1635, 1636 Removing 1635 and 1636.

```
01635, 00000, "rollback segment #%s specified not
available"
// *Cause: (same as 1545)
// *Action: (same as 1545)

01636, 00000, "rollback segment '%s' is already online"
// *Cause: The instance is trying to online an already
//      online RS
// *Action:
```

2.6.5.9 4093 Removing 4093.

```
04093, 00000, "references to columns of type LONG are not
allowed in triggers"
// *Cause: A trigger attempted to reference a long column
//      in the triggering table.
// *Action: Do not reference the long column.
```

2.6.6 In future

According to the ISO standard of SQL *null* should be different from the empty string(''). So even if the two different states cannot be distinguished today in Oracle7, it is advisable to use them accordingly in the applications.

2.6.7 Trailing NULLs

Despite the fact that Oracle 6.0 manuals claimed that trailing NULLs were suppressed, the following example shows otherwise. This is important here since the same example shows that Oracle7 does suppress trailing NULLs, and that an export / import therefore may take up less space if big tables are involved and many trailing columns are NULL.

```
create table space_test (a date, b char(10), c char(11), d number(6),
      e number(7), f number(6,2), g number(6,3));

insert into space_test values ('02-JAN-94', 'GTTGTTGTTG','GTTGTTGTTGT',
      123456, 1234567, 1234.56, 123.456);

insert into space_test values ('03-JAN-94', NULL, 'GTT2',
      NULL, NULL, NULL, NULL);

insert into space_test values ('04-JAN-94', 'GTT3',
      'GTT4', NULL, 123456, NULL, NULL)

commit;

select * from space_test;
```

Gives the following result:

```
A         B          C                    D          E          F          G
--------- ---------- ---------- ---------- ---------- ---------- ----------
02-JAN-94 GTTGTTGTTG GTTGTTGTTGT     123456    1234567    1234.56    123.456
03-JAN-94            GTT2
04-JAN-94 GTT3       GTT4                      123456
```

Looking at the actual datablock shows:

```
          0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f   0123456789abcdef
 2d65f90  00 00 2c 00 07 07 77 c2 01 04 01 01 01 04 47 54   ..,...wÂ......GT
 2d65fa0  54 33 04 47 54 54 34 ff 04 c3 0d 23 39 ff ff 2c   T3.GTT4..Ã.#9..,
 2d65fb0  00 07 07 77 c2 01 03 01 01 01 ff 04 47 54 54 32   ...wÂ.......GTT2
 2d65fc0  ff ff ff ff 2c 00 07 07 77 c2 01 02 01 01 01 0a   ....,...wÂ......
 2d65fd0  47 54 54 47 54 54 47 54 54 47 0b 47 54 54 47 54   GTTGTTGTTG.GTTGT
 2d65fe0  54 47 54 54 47 54 04 c3 0d 23 39 05 c4 02 18 2e   TGTTGT.Ã.#9.Ä...
 2d65ff0  44 04 c2 0d 23 39 05 c2 02 18 2e 3d f3 c1 00 08   D.Â.#9.Â...=óÁ..
          0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f   0123456789abcdef
```

On Oracle7 7.0.16 the similar datablock looks as follows:

```
          0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f   0123456789abcdef
 c4df90   00 00 00 00 00 00 00 00 2c 00 05 07 77 c2 01 04   ........,...wÂ..
 c4dfa0   01 01 01 04 47 54 54 33 04 47 54 54 34 ff 04 c3   ....GTT3.GTT4..Ã
 c4dfb0   0d 23 39 2c 00 03 07 77 c2 01 03 01 01 01 ff 04   .#9,...wÂ.......
 c4dfc0   47 54 54 32 2c 00 07 07 77 c2 01 02 01 01 01 0a   GTT2,...wÂ......
 c4dfd0   47 54 54 47 54 54 47 54 54 47 0b 47 54 54 47 54   GTTGTTGTTG.GTTGT
 c4dfe0   54 47 54 54 47 54 04 c3 0d 23 39 05 c4 02 18 2e   TGTTGT.Ã.#9.Ä...
 c4dff0   44 04 c2 0d 23 39 05 c2 02 18 2e 3d 16 97 00 07   D.Â.#9.Â...=....
```

2.7 Tricks

2.7.1 Using NLS

Here we shall try to give an idea about how to utilize the NLS features of Oracle7, as well as warn against some of the traps in this area.

NLS parameters may be set in the init.ora file, changed for each session, and controlled by the *NLS_LANG* environment variable. Anyway, a session may read its active NLS values in *v$nls_parameters.*

If none of the parameters are specified anywhere,' they default to *American.*

The environment variable *NLS_LANG* takes a value in the following format: *<language>[_<territory>[.<character set>]].*

When the NLS_LANG variable is specified for a process, an implicit *alter session* is performed in order to change the nls parameters from the default setting. Note, that *sqlplus* only alters the session at the first login. Additional connects will not reflect these settings.

A select from *v$nls_parameters* may return the following values where the *nls_characterset* was specified during database creation:

```
PARAMETER              Default value NLS_LANG=danish
---------------------- ------------- --------------
NLS_LANGUAGE           AMERICAN      DANISH
NLS_TERRITORY          AMERICA       DENMARK
NLS_CURRENCY           $             Kr
NLS_ISO_CURRENCY       AMERICA       DENMARK
NLS_NUMERIC_CHARACTERS .,            ,.
NLS_DATE_FORMAT        DD-MON-YY     YY-MM-DD
NLS_DATE_LANGUAGE      AMERICAN      DANISH
NLS_CHARACTERSET       WE8ISO8859P9  WE8ISO8859P9
NLS_SORT               BINARY        DANISH
```

Observe that the *nls_date_format, nls_numeric_characters, nls_currency* and *nls_iso_currency* take a default value from the territory, whereas *nls_date_-language* and *nls_sort* take the default value from the language.

If the following statement is executed: *alter session set nls_date_format='DD/-MM-YYYY'* then all nls parameters take the default value except for the date_format, which takes the specified one. Actually this statement may be specified in the **login.sql** file, allowing sqlplus to use this format through the session.

Now, the *to_date* function will use the default date_format, if nothing is specified, so in an American environment the following statement runs like this:

```
select to_date('01-MAY-94') from dual;

TO_DATE('
---------
01-MAY-94
```

But in a Danish environment the result is this:

```
FEJL:
ORA-01858: et ikke-numerisk tegn blev fundet, hvor et
numerisk tegn var forventet
```

And supplying the date format does not solve the problem:

```
select to_date('01-MAY-94', 'DD-MON-YY') from dual
FEJL:
ORA-01843: ugyldig måned
```

The correct way of doing this, no matter how the nls values are set, is:

```
select to_date('01-MAY-94', 'DD-MON-YY',
'nls_date_language = American') from dual

TO_DATE(
--------
94-05-01
```

It is now also possible to display amounts according to the local standards, as this example shows:

```
select to_char(
 3429, 'L99G999D99','nls_numeric_characters='',.''
 nls_currency= Kr.')
from dual

TO_CHAR(3429,'L99G99
--------------------
    Kr.3.429,00
```

Remember to set the character set to an 8-bit character set, and to set the NLS_LANG to *e.g. Danish* in order to be able to store special national characters.

2.7.2 On V$.. Views

The changed *v$session* view now makes it possible for a running Oracle process to get more information about itself.

The new *audsid* column corresponds to userenv('sessionid'), so it is now easy to get say OS user name, program, attached terminal, ...

```
select osuser, program, terminal
from v$session
where audsid = userenv('sessionid');

OSUSER  PROGRAM                          TERMINAL
------  -------------------------------  ---------
mj      sqlplus@Arwen (Pipe Two-Task)    ttyq0
```

Some views are by the way difficult to select from, since reserved words are used as column names. (In version 7.0.16.2). They are:

In view *v$archive* the column *current* cannot be selected.

In view *v$recover* the column *online* cannot be selected.

In view *v$type_size* the column *size* cannot be selected.

2.7.3 Using Pipes

It is possible to use the pipe package between processes logged into the same Oracle7 instance. Note that the transfer is made instantly, no matter if the transaction is committed or not. Take the following example:

```
set serveroutput on
declare
 status  number;
begin
 dbms_pipe.pack_message('This is a piped message (æøåÆØÅ)');
 status := dbms_pipe.send_message('demo_pipe');
 dbms_output.put_line('message send, res:  '||to_char(status));
end;
```

Figure 5. Process A

```
set serveroutput on
declare
 message varchar2(2000);
 status  number;
begin
 status := dbms_pipe.receive_message('demo_pipe');
 dbms_output.put_line('message received, res: '||to_char(status));
 if status = 0 then
  dbms_pipe.unpack_message(message);
  dbms_output.put_line('got >'||message||'<');
 end if;
end;
```

Figure 6. Process B

When process A does not use *NLS_LANG*, the following text is returned by B:

```
message received, res: 0
got >This is a piped message (fxeFXE)<
```

When process B does not use *NLS_LANG*, but process A does (danish_Denmark), the following text is returned by B:

```
message received, res: 0
got >This is a piped message (??????)<
```

When both processes use *NLS_LANG* (danish_Denmark), the following text is returned by B:

```
message received, res: 0
got >This is a piped message (æøåÆØÅ)<
```

2.7.4 Using Alerts

The Oracle7 RDBMS allows the programmer to use alerts through procedures in the *dbsm_alert* package.

In brief, it is possible for a process to register interest of one or more alerts, and to wait for one or all of them to happen. Another process may then signal an alert associated with a message either directly or indirectly through database changes. These changes may then trigger alerts to the waiting process.

Please note that signals are passed on to the processes that show interest, when the signaling process does a commit. Also note that signals work across all instances on the same shared database.

It is possible for the database administrator to see part of the alert trafic by monitoring the **dbms_alert_info** table.

An example:

Process A may show interest and wait:

```
execute dbms_alert.register('emp_table_alert');
execute dbms_alert.register('an_other_one');

set serveroutput on
declare
 name    varchar2(2000);
 message varchar2(2000);
 status  number;
begin
 dbms_alert.waitany(name, message, status);
 dbms_output.put_line(name||' got '
 ||message||' taken status: '||to_char(status));
end;
/
```

While process B may directly do the signal:

```
execute dbms_alert.signal('emp_table_alert', 'This is a message');
commit;
```

2.7.5 Handeling LONGs

As the upper limit on LONG's (and LONG RAWs) have been raised to 2GB, a number of considerations have to be made.

New external datatypes such as LONG VARCHAR and LONG VARRAW have been introduced in order to cope with the possibility to specify (and get) the actual length of a LONG field. When LONG VARCHAR / LONG VARRAW is used, the first 4 bytes of the buffer may be interpreted as the length. The new PRO*C may in fact be used now in most situations when the LONG values do not get too big.

It is however not easy to store big LONG values into Oracle, because the total LONG value must be known to the system when handed over to Oracle. Do not try to allocate a 2GB chunk of memory on most platforms, or you will (at least) get a very painful swapping / paging problem. In fact, trying to import LONG values will fail if the value requires more storage than specified for the import process *(array fetch buffer size).*

```
IMP-00020: long column too large for column buffer size (30654)
Import terminated successfully with warnings.
```

So you may want to know the 'longest' LONG in the database.

Modifying LONG values is in fact particularly painful, because the whole value has to be stored by the Oracle kernel itself, not only in the redolog, but also in rollback segments. So do use more small LONG values rather than a few big ones. Also observe that the new *discrete* transaction type does not accept LONG management.

It is in fact more easy to fetch big LONG values, since a new OCI routine *oflng* has been introduced to get the LONG value chunk by chunk. And this way we do not have to allocate space for the total value.

Here is an example program to find the length of LONG fields in a LONG column.

```
/* maxlong.c
 * maxlong takes arguments - table_name column_name used/passwd
 * this column is assumed to be LONG or LONG RAW,
 * and are scanned to find the maximum value.
 * The bytes used in the column will be calculated in
   total.
 */

#include <stdio.h>

#define FALSE 0
#define TRUE  1
#define NON_EXISTENT -942

#define CHUNK_SIZE   30004
typedef unsigned char chunk[CHUNK_SIZE];

char     username[20]; /* null-terminated strings */
char     password[20];

char     rowid[20];

chunk    cbuf;

long     long_get, long_indp, pos, totlen, res;
short    rlen;

short   lda[32], cur1[32], hda[256];

void getdoc();          /* generates an employee doc */
void cleardoc();        /* generates an employee doc */
void showdoc();         /* renders doc buffers on screen */

void signoff();         /* handles normal termination */
void sqlerror();        /* handles unrecoverable errors */

main(ac,av)
  int ac;
  char **av;
{
    char    table_name[30], column_name[30];
    char    sqlstate[100];
    char    max_rowid[20];
    int          max_totlen, sum_totlen, long_tmp;
    short   dbtype;

    char    reply[10];

    /* Log on to ORACLE. */
```

```
strcpy(table_name, av[1]);
strcpy(column_name, av[2]);
strcpy(username, av[3]);
strcpy(password, "");
if (ac > 4) strcpy(password, av[4]);
sprintf(sqlstate, "select rowid, %s from %s", column_name,
 table_name);
strcpy(max_rowid, "");
max_totlen = sum_totlen = 0;

if (orlon(&lda[0], &hda[0], username, -1, password, -1, 0))
  goto sql_error;

if (oopen(cur1, lda, 0, 0, 0, 0, 0))
  goto sql_error;

if (oparse(cur1, sqlstate, -1, TRUE, 1))
  goto sql_error;

if (odescr(cur1, 2, &long_tmp, &dbtype, 0, 0, 0, 0, 0, 0))
  goto sql_error;
if (dbtype == 8 ) dbtype = 94;
if (dbtype == 24) dbtype = 95;
if (odefin(cur1, 1, rowid, sizeof(rowid), 1, -1, 0, 0, -1, 0, 0))
  goto sql_error;
if (odefin(cur1, 2, cbuf, CHUNK_SIZE, dbtype, -1, &long_indp,
0, 0, -1, &rlen, 0))
  goto sql_error;

if (oexfet(cur1, 1, FALSE, FALSE)) goto sql_error;
for (;;) {

    totlen = rlen;
    for (res=0, pos=rlen; long_indp!=0;) {
      res = oflng(cur1, 2, cbuf, CHUNK_SIZE, dbtype,
      &long_get, pos);
      pos += long_get;
      totlen += long_get;
      if (long_get < CHUNK_SIZE -4) break;
      if (res != 0) break;
    }
```

```
 printf("\005Nc>Row: %s, Length: %d\n", rowid,
     totlen);
  sum_totlen += totlen;
  if (totlen > max_totlen) {
  max_totlen = totlen;
    strcpy(max_rowid, rowid);
        }
        if (ofetch(cur1)) break;
    }
    oclose(cur1);
    printf("\nLongest %s.%s is %s, with length: %d,
      in total: %d\n",
      table_name, column_name, max_rowid, max_totlen,
      sum_totlen);

notfound:
    ologof(lda);

sql_error:
    sqlerror();
}

void sqlerror()
{
    if ((lda[12] != 0) || (cur1[12] != 12)) {
      printf("\nORACLE error detected:\n");
      printf("LDA return: %d, Cursor Return: %d\n",
       lda[12], cur1[12]);
      ologof(lda);
      exit(1);
    }
}
```

Figure 7. maxlong

3 Database Administration

3.1 Get a Database Generate Script

In Oracle7 it is possible to generate a script, from which it is possible to recreate the database control files. This script may be changed to form the create script of the database.

```
alter database backup controlfile to trace;
```

In the trace file of the process the following statement is found:

```
# The following commands will create a new control file
# and use it to open the database.
# No data other than log history will be lost. Additional
# logs may be required for media recovery of offline data
# files.  Use this only if the current version of all on-
# line logs are available.
STARTUP NOMOUNT
CREATE CONTROLFILE REUSE DATABASE V70 NORESETLOGS NOARCHIVELOG
  MAXLOGFILES 16
  MAXLOGMEMBERS 2
  MAXDATAFILES 999
  MAXINSTANCES 1
  MAXLOGHISTORY 100
LOGFILE
 GROUP 1 '/wd/sql12/v70/dbs/logV70_1.dbf'  SIZE 500K,
 GROUP 2 '/wd/sql12/v70/dbs/logV70_2.dbf'  SIZE 500K
DATAFILE
 '/wd/sql12/v70/dbs/dbV70_1.dbf' SIZE 15M
;
# Recovery is required if any of the datafiles are
# restored backups,
# or if the last shutdown was not normal or immediate.
RECOVER DATABASE
# Database can now be opened normally.
ALTER DATABASE OPEN;
```

Now you just have to remove the data files that do not belong to the *SYSTEM* tablespace.

3.2 Total Export / Import

In order to regenerate the database, it is often wise to let the system itself generate the new database, as *all* files, sizes, etc would then be reflected.

The basis of the create script may be generated with the following command:

```
alter database backup controlfile to trace;
```

Remember to get the appropriate character set from the old database before it is too late:

```
select value from v$nls_parameters
where parameter = 'NLS_CHARACTERSET'
```

Also the rollback segment optimal size is not stored in the export file, and must, therefore, be selected from the existing database in order to generate a new one with the same architecture.

On the old database you could therefore execute a script like this to a spool file *(altroll.sql):*

```
select 'alter rollback segment "'||name||
  '" storage (optimal '||to_char(optsize)||');'
from v$rollname n, v$rollstat s
where n.usn = s.usn
 and s.optsize is not null
```

And execute *altroll.sql* when the new database and rollback segments are created.

3.3 Oracle 7 Backup and Archiving

The following section has been partly donated by Oliver Felix Fox from Dansk Data Elektronik A/S. It demonstrates a practical example of how Oracle7 (and as it turns out also Oracle 6) may be managed backupwise, using compression of the archive files in order to reduce time and space on the backup tapes.

Note that the example is written for at database implemented on UNIX-files, and must be modified if raw devices are used. Normally you would inspect the *cpio* calls.

If the database requires 24-hour uptime, the database is run with archiving and backup of the individual tablespaces.

The directory **$ORACLE_ARCHIVE_DIR/arch**, is used by Oracle to receive the archive-files. The files are named:

$ORACLE_SID<threadno>.<sequenceno>.arc e.g. DE1.159.arc.

The directory **$ORACLE_ARCHIVE_DIR/bu_arch** contains compressed archive-files. These are ready for backup. The compress utility is called *compress*. Moving the archived files to this directory and compressing them is done by the script *mvarch.sh*. The names are <archive_name>.Z .

The script *butables.sh* is used for backing up a given tablespace, called with the tablespace name as argument and optionally as second argument the keyword *rewind*. The script marks the tablespace for backup and creates a copy of the control file, then it copies the file(s) of the tablespace, the backed-up control-file and all compressed archive files in the directory: **$ORACLE_ARCHIVE_DIR/bu_arch** to tape *(cpio).* If the keyword *rewind* is present then the tape is rewound. The archive files are then deleted, and the tablespace is marked as *backup ended.*

The idea is to call the script from another, listing the tablespaces that are to be backed up. In the calling script the last call of butables.sh must use the rewind parameter. Look at *bu_first.sh* and *bu_second.sh* for an example. *butables.sh* writes to a log-file called:

$ORACLE_ARCHIVE_DIR/bu_arch/bu_tspc.log. Herein is listed the name of the tablespace to backup and the output from *cpio*, which should contain the files of the tablespace, a control file and possibly some compressed archive-files. You should once in a while check that the filenames are correct, and that *all* tablespace names are taken into account. Note, that if a new file is added to a tablespace, it will automatically be backed-up by the script.

```
# butables.sh <tablespace_name> (rewind/norewind)
# find the filenames of the given tablespace and mark
# start of backup
# if rewind is present then rewind the tape. Run script as
# Oracle
# By off 5-aug-93
SYSTEM_PASS=${SYSTEM_PASS:-system/manager}
SYSTEM_USER=${SYSTEM_USER:-$SYSTEM_PASS}
logfile=$ORACLE_ARCHIVE_DIR/bu_arch/bu_tspc.log

# make a list of the archives which are compressed (and
# not under compression)
cd $ORACLE_ARCHIVE_DIR/bu_arch
if [ -n "`ls ${ORACLE_SID}*arc.Z 2>/dev/null | tail -1`" ]; then
  for archfile in ${ORACLE_SID}*arc.Z; do
   # orgfile will contain the uncompressed filename
   orgfile=`expr $archfile : '\(.*\).Z'`
   if [ ! -r $orgfile ]; then # its not under compression
    arch_to_tape="$arch_to_tape $archfile"
   fi
  done #2>/dev/null
fi
```

```
# Take a copy of the controlfile, then mark tablespace for backup
ORAENV_ASK=NO . oraenv
sqlplus -s $SYSTEM_USER <<EOF >>$logfile
whenever sqlerror exit 1
set heading off
set feedback off
alter database backup controlfile to '/tmp/${ORACLE_SID}ctrl_bu' reuse;
alter tablespace $1 begin backup;
spool /tmp/bu_$1.files
select file_name
from dba_data_files
where tablespace_name = upper('$1');
spool off
exit 0
EOF
rc=$?
if [ $rc = 1 ]; then
 echo "`date`: Cannot begin backup of $1: CONTINUING"  >>$logfile
fi
if [ ! -s /tmp/bu_$1.files ]; then
  echo "`date`: No files in tablespace $1: NO BACKUP"  >>$logfile
  exit 1;
fi
# copy the files of tablespace to tape; also copy the
# controlfile backup
echo "`date`: ------------ Backup $1 ------------" >>$logfile
if [ x$2 = "xrewind" ]; then
  ls $arch_to_tape `cat /tmp/bu_$1.files`   /tmp/${ORACLE_SID}ctrl_bu | \
    cpio -ovcBL > $TAPE_DRIVE 2>>$logfile
else
  ls `cat /tmp/bu_$1.files` /tmp/${ORACLE_SID}ctrl_bu | \
    cpio -ovcBL > $TAPE_DRIVE 2>>$logfile
fi
rc=$?
echo "`date`: ------------ Ended $1 ------------" >>$logfile
if [ $rc != 0 ]; then
  echo "`date`: Cpio of $1 failes: CHECK BACKUP"   >>$logfile
  exit 1;
fi
# Remove the compressed archives older than 'ctime' which
#  should be on tape
if [ -n "$arch_to_tape" -a x$2 = "xrewind" ]; then
  #rm $arch_to_tape
  find $arch_to_tape -ctime +2 -exec rm {} \;
fi
```

```
# Mark end of backup
sqlplus -s $SYSTEM_PASS <<EOF >>$logfile
whenever sqlerror exit 1
set heading off
set feedback off
alter tablespace $1 end backup;
exit 0
EOF
rc=$?
if [ $rc != 0 ]; then
  echo "`date`: Cannot end backup of $1: CHECK BACKUP"   >>$logfile
  exit 1;
fi
rm -f /tmp/bu_$1.files /tmp/${ORACLE_SID}ctrl_bu
```

Figure 8. butables.sh

The script *mvarch.sh* is called by *cron* (root); checks periodically for files in the directory:

$ORACLE_ARCHIVE_DIR/arch; these files will be moved to the directory: **$ORACLE_ARCHIVE_DIR/bu_arch** and then compressed. The compression reduces the size of the archive files from e.g. 40 Mb to about 5 - 15 Mb. *mvarch.sh* logs its work in the file **$ORACLE_ARCHIVE_DIR/bu_arch/mv_arch.log**.

```
user=`id | cut -f2 -d"(" | cut -f1 -d")"`
bu_dir=$ORACLE_ARCHIVE_DIR/bu_arch
if [ x$user != xroot ]; then
   echo "You must be superuser(root) to run $0!"
   exit 1
fi
cd $ORACLE_ARCHIVE_DIR/arch
if [ ! -r ${ORACLE_SID}*arc ]; then exit
fi
#ulimit 409600
for archfile in ${ORACLE_SID}*.arc; do
 if [ ! -r $archfile ]; then continue
 fi
 set -- `/etc/fuser $archfile 2>/dev/null`
 COUNT=$#
```

```
if [ $COUNT -lt 1 ]; then
    mv $archfile $bu_dir
    echo `date`\\c
    echo ": $archfile moved"\\c
    compress $bu_dir/$archfile
    echo ", compressed"
 fi
done
```

Figure 9. *mvarch.sh*

The scripts *bu_first.sh* & *bu_second.sh* are called by *cron* (Oracle), alternately every day in a week, except for one day, which is reserved for file-backup. See the section on backup. These two scripts together form the tablespace-backup of the database. If a new tablespace is added to the database, then adjustments must be made to one or both of the scripts. Remember the last call to *butables.sh* must use *rewind*.

```
butablespc.sh SYSTEM
butablespc.sh INDEX_TS1
butablespc.sh ROBAS_TS
butablespc.sh TEMP_TS
butablespc.sh STATIC_TS rewind

#check read tape
rm -f /tmp/bu_second.lis
for i in 1 2 3 4 5; do
 cpio -itvBL < $TAPE_DRIVE >> /tmp/bu_second.lis 2>&1
 rc=$?
 if [ "$rc" != "0" ]; then
  echo "CHECK read failed see file /tmp/bu_second.lis" \
   >>$ORACLE_ARCHIVE_DIR/bu_arch/bu_tspc.log
 fi
done
mt -f $TAPE_DRIVE rewind
```

Figure 10. bu_second.sh

3.3.1 Backup of a given database

The archiving-files and tablespaces are backed up using the procedures described in the section about archiving.

6 days a week the tape station $TAPE_DRIVE used for tablespace/archiving backup, giving 3 total cycles of backup. The remaining day is used for a file-system backup. All files in the database server are saved to tape.

The script *filsyst.cpio,* run via *cron* (root), takes the file system backup. The tape is check-read after it is written.

For the server there are currently 31-tapes to be used for backups. These are numbered and a given tape number corresponds to the day of the month it is inserted in the tape-drive. A logbook of the tape-usage resides in the computer room. So, if a given file is to be restored it is necessary to combine the information in the crontab of the users root and Oracle (use the *crontab -l* command as these users) with the log-messages in the file

$ORACLE_ARCHIVE_DIR/bu_arch/bu_tspc.log to find the date (=tape-number) when the file was backed up.

Restoring of normal Unix-files is done with the *cpio* utility, for restoring Oracle tablespaces and archives (which are used for recovering) see the section on recovery.

3.3.2 Oracle 7 crashes

There isn't currently any system which notifies the system-operators if Oracle has gone down. You may however check if the *PMON* and *SMON* processes are alive for that instance.

Whenever it is detected that Oracle is down, You should do the following, before starting Oracle again:

1. change directory to Oracle log-directory:

```
cd $ORACLE_HOME/rdbms/log
```

2. tail the alert-log:

```
tail -NO_OF_LINES alert_${ORACLE_SID}.log
```

to check what the error was. In the log there is probably some references to other files (trace-files) in the same directory eg. pmon_937.trc; check these files as well.

3. If an error code of ORA-00600 (internal error) is found or a similar error code, then you ought to contact Oracle support and report the error; they will normally ask some questions about the log/trace-files, so keep them ready. Oracle support service will tell you if it is ok to start Oracle. If you have any problems contacting Oracle support, then make sure you have at copy of the relevant trace/log-files: from the Oracle log-directory,use:

```
ls -lrt # reverse time-stamped list of files
```

copy the alert-log together with the files listed on the last part of the screen, eg: alert_DE.log dbwr_938.trc ora_8752.trc reco_942.trc arch_939.trc lgwr_940.trc pmon_937.trc smon_941.trc # the numbers will be different make sure to at least copy the files which are referenced in the alert_DE.log

4. Start Oracle:

```
su oracle
sqldba
connect internal
startup
```

If Oracle won't start, you will have to issue, still within *sqldba*, a: *shutdown* and if that didn't help then: *shutdown immediate* or even a: *shutdown abort* to be able to use the startup-command. Anyway you must use judgement here!

3.3.3 Oracle 7 recovery

On the database it is possible to perform a media recovery, due to the fact that the database runs in archive mode and backups of the tablespaces have been made.

Different reasons for recovering, could be:

- Corrupted objects in the database,
- Power failure (if file system files are used for data- and redo files.
- File of tablespace ruined or lost.

In any case Oracle will tell you if tablespaces need recovery, these messages will be in the Oracle logs.

Whenever it is unavoidable to recover a single tablespace or more you must do the following:

1. It is beneficial to run at least two unix-sessions on the unix server.
2. Become Oracle user: su oracle
3. Enter the sqldba: sqldba [lmode=y]
4. Try to bring the tablespace(s) in question offline:

```
alter tablespace <name> offline;
```

you could also use the menu-driven interface for the above action and several of the following; use whichever method you prefer.

5. Now find the newest backup of the tablespace(s) and copy them in (from tape). Remember that a tablespace can consist of several files, but usually it is on a single file. You must check the backup log to find the correct tape. When found and inserted in the tape-drive, forward skip to the cpio-file using, from another session:

```
mt -f $TAPE_DRIVE fsf (NO)
```

   where NO is the number of files to skip. If the wanted files are in the first cpio-file then just insert the tape, and use:

```
cpio <$TAPE_DRIVE (file pattern) # see unix doc.
```

6. Now issue, in the sqldba:

```
recover tablespace <name>;
```

   sqldba responds with some messages like:

```
ORA-00279: Change 6347.........
ORA-00289: Suggestion: <archive name>
ORA-00280: Change 6347.........
Specify log: {<RET> : suggested | filename | AUTO |   FROM.. | CANCEL}
```

7. Now find the suggested archive name (from tape) if it is not already present - it could be either:

   - in the directory: $ORACLE_ARCHIVE_DIR
   - on tape (compressed)
   - in the directory: $ORACLE_ARCHIVE_DIR/bu_arch (compressed).

   If need, copy the archive name to the suggested location and if compressed uncompress by

   ```
   uncompress <archive name>.Z
   ```

   Then press return to indicate the suggested filename, and Oracle will start applying the changes from the archive file to the database. This step is repeated with all the suggested archive names until sqldba responds:

   ```
   Media recovery complete
   ```

   If Oracle is not running then start it, see section on crashes.

8. Bring the tablespaces online again:

   ```
   alter tablespace <name> online;
   ```

   To speed things up a little you should expect that all archive-files from the first suggestion and to the last existing will be needed by Oracle.

To see, at any point in time, which archive files will be needed to perform a recovery of all tablespaces, i.e the whole database, you could issue within *sqlplus or sqldba:*

```
select thread#,sequence#
from v$log_history
where high_change# >=
(select min(change#) from v$backup);
```

The thread# and sequence# correspond directly to an archive file, thus showing the possibly needed files.

Dansk Data Elektronik A/S
Herlev Hovedgade 199
DK 2730 Herlev
Tel: + 45: 42 84 60 11
Fax: + 45: 42 84 62 20