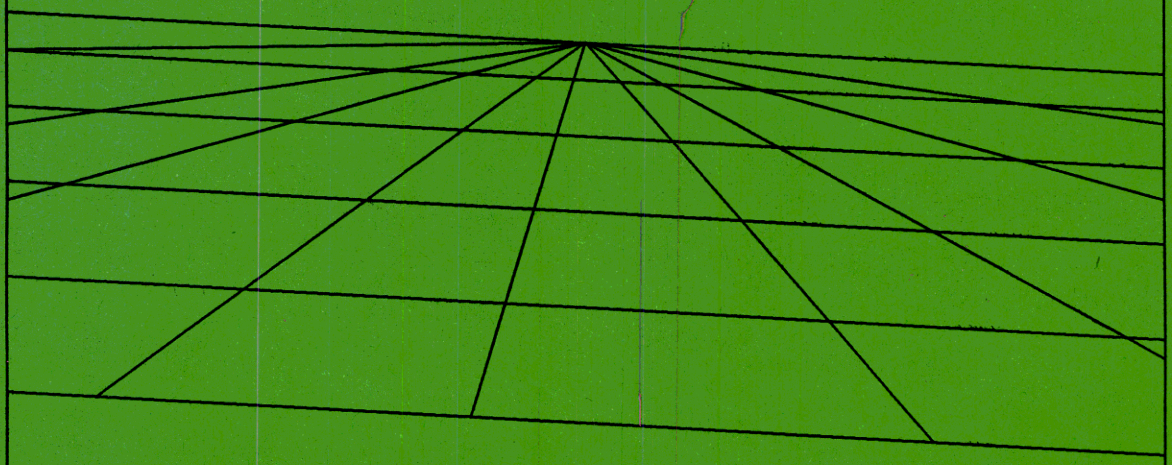
ORACLE®



Supermax ORACLE 6.0 Hints

Dansk Data Elektronik A/S

Stock no: 94003351

Issue.Version: 1.8 - Date: Nov, 07, 1991

CONTENTS

LIST OF FIGURES

Supermax ORACLE 6.0 Hints

Author: DDE, Oracle group

Purpose

The purpose of this document is to clarify some of the more complicated issues in Oracle. When more customers ask the same questions on Oracle a new chapter is likely to appear in this document.

Audience

This manual is written for persons who maintain and develop using the ORACLE Relational Database Management System (RDBMS) on DDE Supermax computers running the UNIX System V operating system. These persons may or may not also be assigned as ORACLE database administrator(s) for the site.

The reader is assumed to have fundamental knowledge of the operating system under which the ORACLE RDBMS will be running.

Changes to this Hint

The following chapters have been added:

- 3.17 - Index Inspection.
- 12.9 - Get Current Date or Time.

The following chapters have been modified:

ORACLE is a registered trademark of Oracle Corporation.

SQL*Plus is a trademark of Oracle Corporation.

SUPERMAX® is a trademark of Dansk Data Elektronik A/S.

UNIX® is a registered trademark of AT&T Bell Labs in the USA and other contries.

- The migration chapter of export 5 patching, has been updated with a new program.

1. Hints on application programming

Due to increasing customer interest in designing and creation of user friendly application programs, making use of the features of Oracle, we have felt the need for more specific guidelines, recommendations and hints on using Oracle for this purpose. Please note that the programmer is free to use our advices.

The designers are advised to use the SQL*forms for application development as long as possible. If, however, the programmer should select a 3rd generation language (C), and PRO*SQL or PRO*C, we think these notes should be taken into account.

We assume that the reader is familiar with the normal behaviour of user oriented data manipulation applications as well as the basic concepts of Oracle.

We see some basic goals to the application, apart from the problem to be solved :

1. The application must be reliant, the values shown must correspond to online information stored in the database. The user may use the information for important decision making.
2. The applications must be able to be started concurrently by many users. This means easy administration to the users.
3. The application should be informative, telling the user of the progress of the transaction, and if any problems is likely to arise, warn the user. Of course only validated data should be inserted and committed to the database.
4. The application should not lock any resource in the database, when the application is waiting for user responses. Otherwise a user could lock part of the system infinitely, preventing colleagues from using these parts of the system.
5. The application must create transactions large enough to cover the integrity of the information in the database, but not bigger. Transactions too large will result in unnecessary resource allocations, where too small transactions results in difficult administration overhead for the application.
6. The application should react wisely to every possible Oracle returncode, e.g. deadlock detected, context area too small and so on. We shall discuss this later.
7. The application should also be as fast as possible (except for special applications where a similar responce time is more desirable that dissimilar faster responces).

As you see, it is quite a challenge to create what we call good applications. Let us study how these goals can be achieved.

1.1 Simultaneous access to the same set of tables

A resource in the database system is usually a lock on a table or on a row. Please note that whenever an insert, delete or update operation is performed on a table, the whole table is locked for that user until a commit transaction call is issued.

As many applications should be able to work on the same set of tables, some strategy has to be followed to assure that the goals above is met. Let us thus discuss the general outline of a standard online application.

NOTE: If you are using the *transaction processing option* no table locks are required, highly reducing the contention on central locks.

a. Present a certain layout to the user.
b. Ask the user for search information.
c. Get current information from the database.
d. Present these values to the user.
e. If the user wants to do some modifications :
f. Validate these modifications.
g. Make the changes in the database.
h. Proceed at state a.

There are of course many different ways to design an application like this one. Let us however point out some of the more sophisticated points.

State **b** and **e** are *human waiting* spots, or states where the application is waiting for a user action. Because the user may decide to make a telephonecall, go to lunch, or even go shopping, the application may be suspended for a rather long period of time. This has the following impact on the way the application should acces the database.

- At **a**: Possible normal read operations from the database. No resource should be locked here, as we will wait for a user responce at **b**. You may however select ROWID.
- At **c**: Information are selected from the database, giving the user a more detailed view (e.g. seat reservations, stock informations, account contents, employee information). **No resource** should be locked here as well, because of the user responce at **e**. ROWID should be used.
- At **f**: If the user feels the need to change information, the application has to re-read the old information from the database (using ROWID), to check if the data has been changed by other users in the meantime. This read should be **SELECT FOR UPDATE** to lock the rows - this is not critical since only machine processing time will delay the responce, which is considerably shorter than the human waiting time. If no changes has occured, the user specified updates can be performed followed by a commit. If other users have changed the date, the locks should be released (commit) and the application should proceed to state **a**, telling the user about the event.

```
WriteScreenLayout; (* a *)
repeat
  GetSearchCriteria; (* b *)
  (* Human waiting time *)
  select rowid, ... into oldrowid, olddata (* c *)
         from ... where ... order by ...
  WriteInformationToScreen; (* d *)
  (* Human waiting time *)
  IfUserModifications then begin (* e *)
    select ... into checkdata
         from ... where rowid = oldrowid
         for update of ...
    if olddata = checkdata then begin
         ValidateModifications; (* f *)
         update ... set ...
           where rowid = oldrowid
      commit
    end
    else begin
         commit;
         InformUserAboutEvent
    end
  end
until NoModifications or ModificationsMadeOk;
```

As you can see this strategy requires three distinct sets of variables : *olddata, checkdata and a set to store possible user modifications.*

1.2 Preventing deadlocks detection returncodes

A deadlock situation arises if the following sequence of events happens :

User 1 locks resource a.	
	user 2 locks resource b.
User 1 locks resource b.	
	user 2 locks resource a.

Oracle will detect that this situation has occured and will rollback the transaction started by user 1 or 2, letting the other transaction pass through. An action to take if you receive this return code, is to start the same changes in a new transaction (this is another good reason to store the data for the whole transaction), or you may of course again ask the user to select possible changes. If you do not want to have this returncode, you must make sure, that the sequence in which you update your tables is the same in all of the applications.

1.3 Locking mechanism's at SELECT FOR UPDATE

Oracle 6 has introduced a new set of lock modes. The locking mechanism's and transaction management are described in the "ORACLE Database Administrator's Guide" chapter 11 and 12. There are however some changes in the method of locking when SELECT FOR UPDATE are done, that should be especially noted when applications are moved from Oracle 5 to Oracle 6.

Select for update in Oracle 6 uses an *"optimistic locking"* method. *"Optimistic locking"* means that rows satisfying the WHERE clause are locked at exec (before the first row is fetched). While Version 5 row-locks are placed when rows are fetched. There are two aspects to be aware of with this change:

1. In Version 5 you can continue to FETCH from a SELECT FOR UPDATE even after COMMIT, since row-locks are set at fetch time. In Version 6, COMMIT will release all locks, and a try to fetch will result in error. A new exec of the select statement is needed, and you will fetch from beginning again.
2. *"Optimistic lock"* means that all rows selected for update is locked at exec time. Once you have locked a row, other users cannot lock or update it until you free it with a COMMIT or ROLLBACK statement. In Version 5 you will have a period of time where the SELECT FOR UPDATE statement has been exec'ed, and where not yet fetched rows can be updated or locked by other users. This means that a transaction can hang at a state where a number of rows has been updated, when another user in between has locked a row in the middle of the first transaction. In Version 6 you can be sure that when the statement is exec'ed, the rows that satisfy the WHERE clause is locked and cannot be updated or locked before COMMIT/ ROLLBACK.

1.4 Sorting char columns

The character set used on the Supermax is the standard ISO/DIS 8859/1 8-bit character set[1]. 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 'æ,ø,å' are to be sorted in the specified sequence. But the character numbers in the standard are:

Char	Hex
Æ	0xC6
Ø	0xD8
Å	0xC5
æ	0xE6
ø	0xF8
å	0xE5

In order to sort properly we have to move 'Å' to a position after 'Ø':

```
order by
  translate(<col>, 'åÅúÚ', 'úÚåÅ')
```

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

```
order by
  translate(upper(<col>),
    'åÅúÚ', 'ÚÚÅÅ'), <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.

1. Described in *Supermax Virtual Terminal Guide*

If your language has more complex sorting requirements like in German or Spain, you may use the *straorder* routine found in the libpa.a library.

1.5 Week numbers

As seen in the SQL reference manual the *WW* returns the week number in which the actual date is located on a year basis. Unfortunately the algorithm used to calculate this number is not the one used in Denmark and other countries. Calculating the week number Oracle assume that no week will overlap the year, or to put it differently:

The 7. January in any year will belong to week 1.
WW = round((*DDD* + 7) / 7).

On the other hand the weekly day number will always turn a Sunday to week-day no 1.

In Denmark the week number calculation uses a more complex algorithm. The week starts on a Monday (Oracle week-day no 2). The Thursday in the week overlapping the newyear is the turnkey to the calculation. The whole week belongs to the same year as this thursday. F.ex the 1. January of 88 belongs to week no 53 of 87.

This following SQL-expression will for a given date, return a week number, following the Danish definition. If 0 is returned the week belongs to the previous year, and if the week is 53, you would have to check if the week really belongs to the next year. The complexity of this SQL-statement suggest that a C-routine should be considered.

```
trunc(
 (
  to_number(    /* get day number of year */
   to_char(to_date(date),'DDD')
  ) +
  decode(
   to_number(
    to_char(    /* get day number of week of 1. January */
     to_date('1-JAN-'||to_char(to_date(date),'YY')),
     'D'
    )
   ),   /* assign an offset according to start of 1. JAN */
   1,5,2,6,3,11,4,9,5,2,6,3,7,4
  )
 ) / 7
)
```

There will of course be other ways to form an SQL-expression to retrieve the week number.

You may of course just select *to_char(<date>, 'WW')* if the *territory* is set proporly in the *LANGUAGE* parameter. (See National Language Support)

1.6 Parameters in Views

Views is a nice SQL-macro like facility, where complex SQL-statements may be stored in the data dictionary. But sometimes the specification of the view is very difficult, because views does not directly handle parameters.

Let us as a simple example illustrate this difficulty by asking for the average salary for employees depending on their department.

```
select deptno, avg(sal+nvl(comm,0)) salary
from emp
group by deptno
```

Sometimes you would like to restrict the select to the clerks only!

```
select deptno, avg(sal+nvl(comm,0)) salary
from emp
where job = 'CLERK'
group by deptno
```

How do we specify a views to select the stuff, without specifying the actual job at creation time?

Create a new table:

```
create table parameter (id number(10),
                        arg1 char(30));
```

and create the view like this:

```
create view sal_dept1 as
  select deptno, avg(sal+nvl(comm,0)) salary
  from emp, parameter
  where job like parameter.arg1
  group by deptno
```

Now to run a select against the view you just have to insert and commit a proper row to the parameter table.

There is however still one problem to solve. How can we make sure that this solution works for multible users accessing the view simultaniously?

When the row is inserted in the parameter table, insert a userid as well, to allow for many different parameter sets:

```
insert into parameter values
     (userenv('sessionid'), 'CLERK')
```

and create the view like this instead:

```
create view sal_dept2 as
  select deptno, avg(sal+nvl(comm,0)) salary
  from emp, parameter
  where parameter.id = userenv('sessionid')
   and job like parameter.arg1
  group by deptno
```

1.7 Turning a table 90°

Creating nice matrix-like reports or forms applications are not trivial, especially if the underlying tables have a different structure.

The *budget* table looks like this:

Department	Month	Expense
Sales	JAN	5000
Sales	FEB	4500
:	:	
Sales	DEC	6700
Development	JAN	12000
:	:	
Development	DEC	10450
:	:	

and we want a table like this to be selected:

Department	JAN	FEB	...	DEC
Sales	5000	4500	...	6700
Development	12000	...	...	10450

The following use of *decode* and *max* will solve the problem:

```
select department,
     max(decode(month,'JAN',expense,0)) JAN,
     max(decode(month,'FEB',expense,0)) FEB,
        :          :          :
     max(decode(month,'DEC',expense,0)) DEC
from budget
group by department
```

This framework may be further extended to cover the situation where all the different fields in the matrix should be selected in one row. Form applications may need this extra facility.

```
select department,
     max(decode(department||month,'SalesJAN',expense,0)) SAL_JAN,
     max(decode(department||month,'SalesFEB',expense,0)) SAL_FEB,
        :          :          :
     max(decode(department||month,'SalesDEC',expense,0)) SAL_DEC,
     max(decode(department||month,'DevelopmentJAN',expense,0)) DEV_JAN,
     max(decode(department||month,'DevelopmentFEB',expense,0)) DEV_FEB,
        :          :          :
     max(decode(department||month,'DevelopmentDEC',expense,0)) DEV_DEC,
        :          :          :
from budget
```

You may of cause join the budget table with itself 12 times, but we think this statement is faster.

1.8 Don't use Sequence with the Nocache Option

The following SQL operations returns an error :

```
SQL> create sequence dept_seq increment by 5 start with 50 nocache order;

Sequence created.

SQL> insert into  dept values (dept_seq.nextval,'DDE','HERLEV');

1 record created.

SQL> update emp set deptno = dept_seq.currval
  2  where ename = 'MILLER';
update emp set deptno = dept_seq.currval
       *
ERROR at line 1:
ORA-08002: DEPT_SEQ.CURRVAL is not yet defined in this session
```

When using Sequence created with the nocache option the generated sequence number is literally not cached - it is thrown away immediately. If you want to be able to refer to CURRVAL in subsequent statements within a session you must always specify at least **cache 2** in the create sequence command.

1.9 Using ROWID as a datatype

ROWID as a datatype is described in the "ORACLE Database Administrator Guide". It is especially mentioned that ROWID not always can be treated like columns because, a ROWID is not guaranteed to be a constant for any row over time (The physical location of a row may change when the row is updated or exported and re-imported.) and ROWID is not stored in the database. Though it can be referenced like ordinary data, it is not another column of data. Thus it would not make sense to UPDATE, INSERT or DELETE a ROWID.

In addition one should be aware that though, it is technically possible to use ROWID as a datatype - you can create a table with a column of type ROWID or create a view where a column is defined as *select rowid* from a table - it is still not describing a real column of data.

Especially you should **note** that the import utility raises a warning when such tables are imported.

We strongly recomment you not to use ROWID as a datatype when creating tables and views.

1.10 Use of Memory in the Oracle Kernel.

The Oracle kernel uses memory for the following purposes:

To store program code being executed.
To store System Global Area (SGA).
To store Program Global Area (PGA).
To store Context area.

The first three areas are fixed in size in the way that the size are determined either at installation time, at instance startup or at connect time; after that it does not change. Together the Program Global Area and the Context Area make the process specific part of memory use. The context area however start at a given size and grow dynamically if more space is needed. The amount of space initially allocated for a context area is set by the INIT.ORA parameter CONTEXT_AREA, or it can be specified by user-written precompiled programs when opening cursors. When the context area is extended, memory is allocated in predefined sizes set by the INIT.ORA parameter CONTEXT_INCR. And the number of cursors (= context area) is set by OPEN_CURSORS. So, You can tune the use of memory space by tuning the parameters:

OPEN_CURSORS
CONTEXT_AREA
CONTEXT_INCR

The application programmers can determine the number of cursors to use, and be aware of closing unneeded cursors, to minimize the total number of open cursors.

You may analyze memory usage for "fine tuning" your application. To help doing so we can give some guidelines for how some specific SQL statements uses context area.

A Context area start at a given size as extension of the Program Global Area and grow dynamically if more space is needed. The context area contains the text of the SQL statement, one row of the result and some intermediate information, plus some status information and control information for possible internal sorts. So the cost of memory depends on the size of the statement and it depends on the size of a row (the number and size of the columns,) and it depends on the complexity of the SQL statement.

The main types of SQL statements are:

Simple select statement.
Select of many columns.
Select with ORDER BY.
Complex queries with use of DISTINCT and SET operators.

In the following, specifications of context size will include the size of Program Global Area.

Logon.

Costs 110592 bytes.

Simple select statements.

118784 bytes. Cost only the size of the statement as it is represented internally.

Select of many columns

The use of memory is direct proportional to the number of columns selected. For every 100 columns memory grows with 16384 bytes. And memory grows as well direct proportional to the size of the columns.

Select with ORDER BY.

The important object here is if the ORDER BY column has an index and if this index is unique. If there is no unique index on the column the RDBMS will have to sort. Even with a non-unique index sort can be needed! Sorting needs some internal memory available in context area. The size of this area can be set by the INIT.ORA parameter SORT_AREA_SIZE. Default is 0x10000 bytes (65536) which means a demand for this size in the context area.

Complex SQL statements.

In general we can say that it costs sort memory when RDBMS have to use sort-merge. And it costs sort memory when SQL statements using functions where statements have to be processed in parallel (ex. set operators, nested selects).

DISTINCT will have to sort as well, it costs the sort area size for the context area.

The set operators UNION, INTERSECT and MINUS combines queries to return distinct rows from every individual query and will therefor use temporary tables for each set of query combined (that is a sort area size for each query).

1.11 An SQL*Plus Deamon

In many application systems, it is nessesary to be able to run small SQL-statements in an asynchronous manner. The application wants to initiate a little update script, but does not want to wait for the response.

Normally the application process will spawn small background tasks to implement its needs. But if many simultaneous processes are doing this, the amount of small background tasks will have an significant impact on the performance for the on-line users. Instead you might consider starting an *sqldeamon* at boot time (through a script in the */etc/rc.d* directory), ready for handling the application needs.

```
## sqldeamon
## DDE 25. OCT 1990, MJ
## Usage : /bin/prod sqldeamon > > log 2>&1
## Set SYSTEM_PASS to "username/passwd" before the call
## The result from the deamon are written to std-err
##
## The deamon may be used by writing commands to /tmp/sqldeamon
##

DEAMON=`basename $0`

SQLFIL=/tmp/$$.sql
COMFIL=/tmp/$DEAMON
export SQLFIL COMFIL LOGIN DEAMON

## Check that the deamon is not running allready
if [ -p $COMFIL ]; then
  echo "The deamon >$DEAMON< is allready running" >&2
  exit 1
fi

trap "" 1 2 3 15

## Set default SYSTEM_PASS
set ${SYSTEM_PASS:=system/manager}
export SYSTEM_PASS

## create the pipes
/etc/mknod $SQLFIL p
/etc/mknod $COMFIL p
chmod 777 $COMFIL

## Make sure the background job can be started, and does not wait on the
## pipe opening!
sleep 60 >/dev/null <$SQLFIL &

## Start the sqlplus program in background
echo "The $DEAMON deamon is being started, date: `date`" >&2
SQL=`/bin/backgr $ORACLE_HOME/bin/sqlplus $SYSTEM_PASS 6>$SQLFIL <$SQLFIL 3>&2`
if test $?; then

## setup sqlplus
  echo "set heading off"   > $SQLFIL
  echo "set autocommit on" > $SQLFIL
  echo "set time on"       > $SQLFIL

## get input and process:
  while read LINE <$COMFIL 6>$COMFIL
  do
    case $LINE in
      exit*)
        echo $LINE > $SQLFIL;
        break ;;
      *)
        echo $LINE > $SQLFIL ;;
    esac
  done

## Wait for the termination of sqlplus
  SQLRES=`wait $SQL`
fi
echo "The $DEAMON deamon has stopped, date: `date`" >&2

## Show possible errors on std-err
echo "SQL-result: $SQLRES" >&2
```

```
## Remove work-files
rm -f $SQLFIL $COMFIL
```

The *backgr* program is used instead of & to make shure the sqlplus runs on the same priority as the shell itself. See the *RPT RPF through Pipes* chapter for a listing of *backgr*.

1.12 Using Row Locking

Let us investigate the *row level locking* feature from the *Transaction Processing Option* of version 6.0.30.

First suppose we have a unique index om emp(empno):

```
SQL> create unique index emp_inx on emp(empno);

Index created.
```

And inserts a new row with the id 8888 - without committing from session A.

```
SQLA> insert into emp (empno, ename)
      values (8888, 'SESSION A');

1 row created.

SQLA> select empno, ename from emp
      where empno = 8888;

     EMPNO ENAME
---------- ----------
      8888 SESSION A
```

Now start annother session from another window - session B:

```
SQLB> select * from emp
      where empno=8888;

no rows selected

SQLB> insert into emp (empno, ename)
      values (8888, 'SESSION B');
```

Session B now waits until session A releases the lock on the new 8888 item - not the row.

```
SQLA> commit;

Commit complete.
```

```
insert into emp (empno, ename)
values (8888, 'SESSION B')
                *
ERROR at line 1:
ORA-00001: duplicate key in index
```

So - as you see - an insert may wait on a resource hold by another session.

Of course - if session A did an rollback instead, session B would succeed the insert.

On earlier Oracle 6 versions , session B did not wait but returned instantly. We welcome this change, and think this is more correct - if possible.

The next example shows that the rows are not only checked against the search criteria in the execute fase, but also when lockes are released and the rows are about to be fetched.

Consider this table (**dept**).

```
    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

Now suppose session A changes BOSTON to BERN - without committing.

```
SQLA> update dept set LOC = 'BERN'
      where deptno = 40;

1 row updated.

SQLA> select * from dept
      where deptno = 40;

    DEPTNO DNAME          LOC
---------- -------------- ----
        40 OPERATIONS     BERN
```

From another session B we select for update all from BOSTON:

```
SQLB> select * from dept;

    DEPTNO DNAME          LOC
---------- -------------- --------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON

SQLB> select deptno, loc from dept
  2  where loc = 'BOSTON'
  3  for update of dname;
```

Here session B is waiting for session A to decide on commit or rollback, which will have an inpact on the rows returned to session B.

```
SQLA> commit;

Commit complete.
```

After session A commits, there is no rows for session B.

```
no rows selected
```

From Session A, Now insert a new row identifying BERN.

```
SQLA> insert into dept
      values (50, 'ADMIN', 'BERN');

1 row created.
```

```
SQLB> select deptno, loc from dept
  2  where loc = 'BERN'
  3  for update of dname;

    DEPTNO LOC
---------- -------------
        40 BERN

SQLB> commit;

Commit complete.
```

From Session A, Now change no 40 to lock it.

```
SQLA> update dept set dname = 'OPER'
      where deptno = 40;

1 row updated.

SQLA> select * from dept
      where loc = 'BERN';

    DEPTNO DNAME          LOC
---------- -------------- -----
        40 OPER           BERN
        50 ADMIN          BERN
```

```
SQLB> select * from dept
      where loc = 'BERN';

    DEPTNO DNAME          LOC
---------- -------------- -----
        40 OPERATIONS     BERN

SQLB> select deptno, loc from dept
  2  where loc = 'BERN'
  3  for update of dname;
```

Here we wait again for session A to commit of rollback.

```
SQLA> commit;

Commit complete.
```

```
   DEPTNO LOC
--------- -------------
       40 BERN
       50 BERN
```

The nice thing here is to observe that we got both the rows back (not only no 40). This feature was not working on previous Oracle 6 versions either.

1.13 Userenv('terminal')

The *userenv('terminal')* feature is in general used to determine from which terminal the userprogram is running.

```
select userenv('terminal') from dual;

USERE
-----
tty53
```

If you are using this feature, be prepared to handle other values.

In a window on an X-terminal the value might be *sp05*.

If the userprogram is started in background, or has redirected the input from the terminal, expect the value *?*.

2. Hints on tuning and optimizing

When developing an application on top of the Oracle database system you should always include an optimizing and tuning phase in your overall planning. Some basic programming considerations is described in *Hints to the Oracle Application programmer* - this paper will discuss some basic hints on how to make your application programs run faster. By the way, the Oracle manual: *ORACLE RDBMS Performance Tuning Guide* is excelent, and should be studied carefully.

2.1 Swapping

If your application is slow (very slow), the first thing you should check is whether the Oracle processes are swapping. If this is the case, then the only solution to the problem is to allocate some extra space on the Oracle CPU, or to use less. The latter could be done by reducing the size and number of your cursors, or you may reduce the size of the SGA. This can be done by reducing the number of database buffers (determined by the init.ora parameter db_block_buffers) - but be careful because this might increase the io to the database files. No other course of action will have any effect.

2.2 Storage Definitions

Creating sensible storage definitions for you database tables, can spare Oracle for some disk readings, thus optimizing the system. This is particularly true when dealing with :

1. *Small tables*, where the default start block of 12 kb. and the extents of 12 kb. each, could be made smaller. Much free space makes update faster, but degrades retrieval.
2. *Fixed tables* - tables that does not grow and contains rather stable information has no use for any free space (the PCTFREE can be set to 1 and PCTUSED to 95 or even 99).
3. *Big tables* - allocating an appropriate initial block will make the retrieval of data faster - when scanning sequentially, more extents means more jumps so the Oracle kernel process may be less effective. This overhead leads to the general statement "Fewer larger extents are better".
4. *Dynamic tables* - tables who has a large number of updates should have a reasonable PCTFREE to let the rows grow without allocating overflow blocks as they will degrade the system. Tables who has a large number of deletes and inserts should have a reasonable PCTUSED. The higher PCTUSED the harder Oracle works to keep rows in as few blocks as possible. There are no specific tool to count the number of overflow blocks in a specific table, but if the average size of the row is known, you may run the following SQL-statement to get an idea about the number of rows per block:

```
select substr(rowid,15,4) tbspace,
     substr(rowid,1,8) block,
     count(*)
from <table>
group by substr(rowid,15,4), substr(rowid,1,8)
having count(*) > <average rows per block>
order by count(*) desc
```

Space usage of existing tables is shown in the dba_extents table - if you want to go into more details you could make an export/import of your table, to assure that the rows of the

table is laid out sequentially, before looking at the dba_extents table. Besides you could study the minimum and maximum ROWID's within each extent. Creating a new table with a new storage definition for an existing table is done in the following manner :

- export the table
- drop the table
- recreate the table using the new storage definition
- import the table
- recreate indexes

2.3 Indexes

Oracle uses a structure known as B*-trees for organizing its indexes. This structure is fixed in the sense that no more than 3 or 4 disk readings is ever needed in order to retrieve a certain entity of data, regardless of the table size. This is because each index block will contain between 50 and 100 references to blocks on the next level, hence 4 levels with only 50 entries can handle table sizes up to more than 6 million rows.

Using indexes on larger tables is a must, if there is a need to speed up the response time. If you know the normal search paths through your tables, you should create appropriate indexes for optimizing these paths. The Oracle query optimizer will check the columns for indexes and it will make use of these indexes whenever possible.

Indexes will increase the performance of queries, but will do so at the cost of additional disk space usage and overhead on insert,update and delete operations. This is due to the internal housekeeping connected with the maintenance of indexes. Therefore you are advised to be carefull using more than 3 or 4 indexes on one single table.

Another use of indexes is for data validation. Creating a *unique* index on a table key will cause Oracle to reject records during insert, that uses an already existing key. Always create the index *unique* if possible.

2.4 Multiuser systems and the Oracle Cache

In a multiuser environment where more people are working on the same table(s), the response time will be faster. The currently used table(s) will be located in the SGA *cache*, thus minimizing the need for physical disk readings. If the users are working on different tables, the *hit-rate* in the cache will fall and respons time will grow. Consequently : increasing the size of the cache is a way of optimizing the system. This is done by changing the db_block_buffers parameter in the init.ora file (but remember: never increase it so much that the system is forced to swap).

2.5 Optimizing SQL-statements

When a user sends an SQL statement to the Oracle kernel, it is processed in the following manner :

1. the SQL sentence is parsed
2. and then optimized : For the columns mentioned in the WHERE predicates, Oracle has a scheme for determining which column or columns *drive* or act as a starting

point for database searches. A clause like **rowid=constant** will get a high score whereas a clause like **column like '%C%'** will not. For more information on this you should look in the *Administrator's Guide* under *Oracle system query optimizer.*

Since The Oracle Cooperation is still expanding the intelligence of the strategies, there might be some changes of algorithms in different versions of ORACLE DBMS. From a programmers point of view, the thing worth noticing is that some predicates are more expensive than others, especially those that forces a full table scan.

The Oracle system does not keep any record of the size of a table. When joining two tables it has no way of determining which table is the smaller one. If there is an index on both of the tables Oracle will perform a full table scan, starting with the last table in the FROM clause : If your join operation is slow then try to reverse the order of the tables mentioned in the FROM clause.

2.6 The TRACE function

When optimizing an SQL statement you should take it out of your application program and test it separately in SQL*plus. This can be done by setting a traceflag in the database :

```
                    /* in 6.0.26: */
alter session set events '10035 trace name context forever';
alter session set events '10046 trace name context forever';
                    /* in 6.0.27 and later: */
alter session set sql_trace true;
```

This will enable a TRACE function that keeps track of how the system handles your SQL statement, and writes this information into a file : oraSXX.trc where 'S' is your ORACLE_SID and number. This file will be located in the directory specified by the user dumps parameter in the init.ora file - the default value is '$ORACLE_HOME/dbs/ '.

To end the TRACE function you should write :

```
                    /* in 6.0.26: */
                          /* We do not know!!! */
                    /* in 6.0.27 and later: */
alter session set sql_trace false;
```

The trace file will contain information on tables and views that SQL*plus has been working on and the search paths it has choosen (the use of indexes, full table scans etc.). The TRACE function is a very good tool for checking which tables are picked first, if there is performed any full table scans, if there is a need for more indexes, and so on.

2.7 I/O Operations

The Monitor function of SQL*DBA will give you an idea about the number of logical and physical read operations to perform when an SQL sentence is executed in SQL*plus. You may adjust the buffer parameter in the init.ora file to minimize the number of physical i/o operations. If you are testing SQL's this way and want to assure uniform testing conditions, you will have to restart Oracle each time to clear the Oracle cache.
It might not be possible to use the Monitor io function to tell how heavy the load of the

Oracle disks are. This function will only give you the load caused by Oracle not by other users. If you know that your Oracle database files and/or log files are placed on physical disks used by others as well, always use another statistic tool to show the load on the physical disk (on Supermax the program sysdisp can be used).

2.8 Rewriting your SQL-statements

A general and effective way of optimizing an application, is to join a number of small SQL selects into fewer and more powerfull SQL statements - parsing these bigger SQL's will take longer but they will execute much faster because of the smaller amount of data exchange involved.

2.9 Two more brutal ways of optimizing

If you have run out of methods and ideas on how to make your application go faster, you could start to denormalize you tables by ways of including smaller tables in the bigger ones, even though this is certainly not neat.

Another thing you could do is to change your program, so that it recieves more data from Oracle than actually needed, e.g.: if a select sentence involves a function like DISTINCT, a temporary table will be created. This will effect performance in a negativ way. Instead you could leave out the DISTINCT function and let your application program take care of the data evaluation, especially if it has to do some sorting of the data anyway.

2.10 Unique key checking

Consider the following situation:

table a (key, dataa) (small one)
table b (key, datab) (big one)

Suppose you want a list of keys from table **a** not found in table **b**, because you are going to merge data. You might come up with this suggestion:

```
select key from a
where key not in
    (select key from b)
```

Figure 1. Statement 1

This statement will however perform rather slow. Try instead an outer join on the two tables:

```
select a.key, b.key
from a, b
where a.key = b.key (+)
 and b.key is null
```

Figure 2. Statement 2

Try to swap the table specification:

```
select a.key, b.key
from b, a
where a.key = b.key (+)
  and b.key is null
```

Figure 3. Statement 3

In an example where table a contained 1300 rows and table b about 25000 rows, the following relations was found:

Statement	No inx	Inx on a	Inx on b	Inx on both
1	42	42	42	42
2	4	7	1	1
3	3	3	1	1

The numbers represents the relative response time. The best result was set to one.

Joining tables, always keep an index on the smaller tabel. Tests showed that clusters did not improve the results further.

2.11 A concluding remark

Tuning is in its essence a question of deciding where it is worth the effort to optimize an application. If your process is spending most of its time with disk accesses, this is where to set in and not anywhere else. If it is choking on some forever lasting SQL construct, this is the thing to worry about. Look for the tight spots in the application and concern yourself with optimizing these parts.

3. Oracle Database Administration

3.1 Oracle Backup procedures

There are basicly two different backup methods you can use when running Oracle on a Supermax machine. The first is to apply one of the available administration programs (*tar*, *btar*, *dskback*). The second is to use the dedicated Oracle program exp.

3.1.1 Administration program

tar - is used for backup of altered files out on diskettes, often on a daily basis :

tar cfv /dev/flop <filename(s)>

Used in conjunction with the Oracle exp program, tar could be used for making backups of single Oracle tables or users and saving them on diskette.

btar - this program is especially well suited for copying files out on a streamer, as it writes out its data in big chunks. Given the R option it will also copy raw disks :

btar cvRf /dev/stream <filename(s)>

The <filenames> could be the names of the databases and redologs raw disk files, which would then be copied out on the same streamer tape.

When copying Unix files, one would often prefer to use the program *cpio*, which reads a number of filenames from standard input (in the example : filea,fileb,filec,filed) and copies these to a specified device :

ls filea fileb filec filed ¦ cpio -ov ¦ streamdrv > /dev/stream

The program for copying logical disks is called *dskback* and has the following call format:

dskback source_disk destination_disk

e.g. : dskback -c /dev/dsk/dbs_disk /dev/stream

3.1.2 Important !

Before using *btar* or *dskback* for making backups of Oracle raw disks (or files), the database must be shut down. *Always* copy out **DBS** (database), **REDOLOG** (redolog) *and* **CONTROL** (control) files.

If at least one of the **REDOLOG** files of a backup is lost or damaged, one may use a earlier backup.

3.1.3 Oracle administration programs

exp - The Oracle exp program is suited for exporting single tables or users out into unixfiles, disks or streamers.

The Oracle administration programs backups the database on a condensed format, meaning that the backup files created are much smaller than they would be using *btar* or *dskback* - for instance: the index definitions are exported instead of the actual indexes, thus saving a lot of space.

In addition, the exported tables are reorganized when reimported, assuring that the data is laid out sequentially. This could result in better Oracle performance after an export/import.

If the Oracle system is heavy loaded with user transactions, the export will not guarantee a consistant backup. The reason is that export only selects one table at a time, maintaining a consistant picture of a single table. But if transactions are updating more tables there is a little chance that your backup will only reflect part of the transaction.

3.1.4 The MAXEXTENTS Parameter of Space Definitions

The maximum number of incremental extents Oracle can allocate on a 4k block system (our default) for a table or index is the minimum of 249 and MAXEXTENTS. Even if the MAXEXTENTS parameter is set to the default of 9999 the operating system specific limit of 249 applies.

3.1.5 Export Space Definitions

If you are selecting *compressed* (the default), then the export program, will calculate a special Space Definition for yor table, where the new *Initial pages* can contain all data for your table. This choice is fine in most cases. For very large tables however, where it is esential, that you have divided your table in a big initial chunk as well as a number of big extents, this scheme is bad too, and may abort your import procedure as well, if the database do not hold sufficient room for the table.

We therefore recomment, that you should use export with *compress = no* or create your tables with the prefered storage parameters before importing.

3.1.6 Backup Strategy

1. *btar* or *dskback* provides a fast way of backing up the database files. The Oracle system has to be shut down, and copies of both the DBS, REDOLOG and Control files must be made. The data is copied exactly as found.
2. *totexp* is slower but takes up less space and insures an optimized layout of the database when data is reimported with *totimp*. The Oracle system is up while these programs are running, but no heavy and ramified transactions must take place.

   A usefull strategy when running on a Supermax system with general backup procedures, is to make a total export of the database into a UNIX file, and letting the generel backup routines take care of the actual backup of this new file, for instance during the night.
3. If you a running your database with archiving on, and ofcause are saving your redolog-files you may choose to backup one or more tablespace at a time using any raw copy utility while the database is still running. The only requirement is that the *Alter tablespace <name> start backup* is issued. You must remember to *Alter tablespace <name> stop backup* when the backup is finished, or Oracle will try to *Recover* the tablespace nect time it is warmstarted.

There are unfortunately no way to see if a tablespace are put in a *start backup* state.

NOTE: If you set a tablespace offline, and then copy the tablespace, you cannot use this copy to recover your tablespace since the other tablespaces will depend on the values in the tablespace. If you are not using the archive facility, you may either take a total raw backup, or use the total export / import strategy.

3.2 Full Database Backup / Restore

Regardless of your choice of strategy for the daily database backup, it is highly recommended to regularly back up the entire database. The backup must comprise all database files, redolog files and control files. New versions of the tape management utilities enable you to back up the entire database to one single tape.

3.2.1 Backup to tape

The following is an outline shell-script to back up one database to tape. The database must be shut down and the script must be run as super-user.

```
# backup database and redo log disks and control files to tape.
# Requires Basic Utilities 2.70 or newer
#
# syntax:   backup < tapedev >
#
if [ $# -lt 1 ]
then
      echo "Usage: backup < tapedev >"; exit 1
fi
TAPEDEV=$1
ORACLE_SID=H ORAENV_ASK=NO . oraenv # Insert ORACLE_SID for your site
if orastat -s >/dev/null
then
      echo "Can't back up a running database, shut down first"; exit 2
fi
# Get file names
SPECFILES=`dbfiles | grep block | awk '{ print $1 }'`
#NORMFILES=`dbfiles | grep file | awk '{ print $1 }'`
dbfiles | grep file | awk '{ print $1 }' >.NFILE$$
# Copy raw disks first
dskback -B $SPECFILES $TAPEDEV
# Position tape at end of dskback
mt -f $TAPEDEV append
# Copy control files
cat .NFILE$$ | cpio -ova | streamdrv -R >$TAPEDEV
rm -f .NFILE$$
```

This is a physical rather than a logical backup, so no compression will take place. Make sure your backup medium is large enough to store all the files.

3.2.2 Restore from tape

This shell-script will restore a database from a tape that was made as described in the previous section.

NOTE: This procedure will restore the entire database, including redo logs and control files. If you are running in *ARCHIVELOG mode* (see later this chapter) and want to restore only database disks, do not use this shell script.

```
# restore database disk, redolog and control files from tape.
# Requires Basic Utilities 2.70 or newer
#
# syntax:   restore < tapedev >
#
if [ $# -lt 1 ]
then
      echo "Usage: restore < tapedev >"; exit 1
fi
DEV=$1
ORACLE_SID=H ORAENV_ASK=NO . oraenv
if orastat -s >/dev/null
then
      echo "Can't restore running database, shut down first"; exit 2
fi
# Restore files with their original names
RESTFILES=`/etc/dskback -T $DEV | while read line; do
            awk '{ printf "%s:RESTORE ", $1 }'
      done`
NUMDISK=`echo $RESTFILES | wc -w`
dskback -R $DEV $RESTFILES
# Reset tape to beginning of cpio archive
mt -f $DEV fsf $NUMDISK
# Copy control files into their original names, overwrite if necessary
streamdrv -R <$DEV | cpio -idmuv
```

This will restore the entire database to its state at the time of the backup. The script will work provided the database and redolog disks have the same names and sizes as at the time of the backup.

Before starting the database, check if the control-files have been renamed since the time of the backup. The restored control files must have the names expected in the current init.ora file. If these conditions are met, you should be able to start and work on the database immediately upon running the script.

3.2.3 Recovering a single disk.

If you are running in *ARCHIVELOG mode* you might want to restore only part of a full database backup. This requires that all archived redo logs from the time of the backup to the moment of recovery are available. Use *dskback -T* to get a table of the disks copied on your tape. Then restore the database disks that you are going to recover using *dskback -R*.

3.3 Export / Import Remarks

In general it should be possible to running export (*exp*) to a file, and later on possible to apply the total or partial information on the export-file to the database again running import (*imp*). There are however cases where such a procedure will produce errors, or missing information in the database, and because the time used for an import of a large database is considerable, you may want to prevent problems running import.

3.3.1 Tablespace Information

Information on Tablespaces and Files belonging to different tablespaces will be present in the export file. This is ok, since it is possible to reclaim files by dropping a tablespace.

NOTE: Remember to set the tablespace offline before dropping the tablespace.

If you have more tablespaces, you must have at least two rollback segments. Unfortunately, it is impossible to insert data in a table of a non SYSTEM tablespace if the system has not been brought down and up since the creation of the second rollback segment.

In other words, if a total export covering more tablespaces are to be imported, you will have to:

1. Creating tablespace and rollback segments.
2. Restart the database.
3. And then importing the tables and rows.

This scheem is covered in the *totimp* script.

3.3.2 Acceptable Errors

The database may include invalid view definitions, because a view is not deleted when a table is altered, droped or renamed. This is fine if you want to recreate the table, and like your views to work again automatic. But if you do not recreate your table - or recreate the table with other column named or types - you will see the following happen.

The export program will form view create statements in the export file. But when the file is imported, the views are not created, because they would produce error messages.

3.3.3 Parallel Import

If you are running a large database, it may take many hours to import an entire export file. You may observe the following trick to speed up the import.

Instead of just having an entire export file, you may ask for the entire export with no records, and for each user an export of all thier tables and rows. The total export may then be used to create all the users, as well as grants of facilities from one user to an other.

Having applied the total export, the user imports may be performed in parallel.

3.4 Recovery using Oracle in ARCHIVELOG mode

This hint is an introduction to Oracle's recovery scheme using ARCHIVELOG mode, including saving the archived redologs automaticly. You must consult the following DBA manual for detailed information:

- ORACLE, "Database Administrator's Guide", version 6.0, (At least chap. 14 and 15).
- SUPERMAX, "System Administrator's Guide", UNIX System V Realease 3.1, version 4.0, (At least chap. P11 "Backup management Procedures" and 11 "Backup Administration").

Oracle may recover from an **instance failure** (power break, system crash or "shutdown abort") to the point of the last committed transaction:

> Oracle performs instance recovery automatically after an instace failure when a *startup* is issued from *sqldba*. This brings the database up to the state of the latest commit before the failure.
>
> Oracle scans the redolog from the last checkpoint and inforces all changes to the database (**Note** that small redologs and frequent checkpoints will reduce the time needed to recover). At this point all transactions are *rolled forward*, next all uncommitted transactions are undone based on the information in the rollbacksegments.

In case of a **media failure** (disk crash), the disk must be checked, repaired or replaced, *you must have a plan for recovery and have taken backup actions accordingly - prior to the failure*. The choice is to install the backup (import or diskback) to the state of the backup time, or recontruct the database (diskback and archived redologs) the time of the last *commit* before the media failure. A summery of the later recontructing recovery scheme is:

> Before a crash occurs you must take regular diskbackups and run Oracle in ARCHIVELOG mode. Note: Consistent disk backups of all databasefiles and file backups of controlfiles must be taken before and after changing files configuration (ex. alter (add)/drop tablespace).
>
> If the SYSTEM tablespace is damaged install all files form latest diskcopy (NOT the controlfiles). Issue "RECOVER DATABASE" from *sqldba*.
>
> If an other tablespace than SYSTEM is damaged bring the tablespace offline and install the damaged files form latest diskcopy (NOT the controlfiles). Issue "RECOVER TABLESPACE name" from *sqldba*.
>
> Hereafter *sqldba* will prompt for all needed archived redologs to be inpected to reinforce all committed transactions made after the time of the backups. If the archived redologs have been saved on tape they must be reinstalled.

The archiving mode is switched using the following *sqldba* script *archive*. You should backup all database disks immediatly after a switch to ARCHIVELOG mode! You may call *archive list* to inspect the archiving state, to initiate archiving call *archive true* and *archive false* to cease archiving. Note: The script alters the **init@.ora** file.

This insures automatic archiving each time Oracle is warm started but not necessarelly at creation time, ex. recreating a database without archiving and later enabling arciving with *arcive true* and theire by setting the init.ora parameter *log_archive_start=true*.

```
# archive < list ¦ true ¦ false >          Note!  Alters init@.ora file
case $1 in
  false) cmd='echo "alter database noarchivelog;"' break;;
  true)  cmd='echo "alter database   archivelog;\narchive log start;"' break;;
  list) sqldba < < EOF
        connect internal
        archive log list
EOF
          exit; break;;
  *)    echo "usage: $0 [list ¦ true ¦ false]"; exit; break;;
esac
sqldba <<EOF
shutdown
startup nomount
connect internal
alter database $ORACLE_SID mount;
$cmd
archive log list;
alter database open;
EOF
grep -v log_archive_start $ORACLE_HOME/dbs/init$ORACLE_SID.ora >  /tmp/init$$
echo   log_archive_start = $1                                  >> /tmp/init$$
cp     /tmp/init$$      $ORACLE_HOME/dbs/init$ORACLE_SID.ora
```

If Oracle should run in ARCHIVELOG mode form the start you must specify the init.ora parameter *log_archive_start=true* in the **init.ora** file and specify "CREATE DATABASE ... ARCHIVELOG;" when the database is created. This does not invalidate later use of the *archive* script. On the other hand this strategy may generate a substantial amount of archived redologs, ex. initiating a database with a large import in ARCHIVELOG mode.

Running Oracle with automatic archiving will systematicly fill up the archive directory, default **$ORACLE_HOME/dbs**, with files named **sysar@_*n*.dbf**.

The archives are infact on version 6.0.27 by default called: **arch_*n*.dbf**. Because the *sid* has been removed from the filename, you should definitly avoid running multible databases on the same *ORACLE_HOME* using the default archive name and directory. When the archive directory fills up Oracle well stop, and avoiding this situation therefore becomes a main concern for the database administrator. The solution is automatic backup and thereafter deleting the files. The following procedure works for version 6.0.26, where on version 6.0.27 the *arch* process may write the archivefiles directly on tape in *cpio* format.

The Supermax backup system included in a new version of *sysadm* offers an automated backup service ready to be exploited. The strategy is to move all archived redologs not in use to a subdirectory and specify this directory to the backup system. After the backup has been performed the arcived redologs are deleted. The actions to be taken are:

1. Define a logical dataset name to the backup system using *sysadm setconf*, ex. add the dataset name *ORA6ARCHIVE* and a subdirectory to the archive directory **/usr/oracle6/dbs/ARCHIVE**. This subdirectory must be created manually by *mkdir ARCHIVE*.
2. Create the following pre-backup script **/etc/backup.d/pre.d/ORA6ARCHIVE** to move all archived redologs (not in use) to the archive subdirectory **ARCHIVE**:

```
# Pre-backup script moving Oracle's archived redologs (not used) for backup
#
ORACLE_SID=L ORAENV_ASK=NO . oraenv
cd $ORACLE_HOME/dbs
ulimit 204800
test -r sysar$ORACLE_SID*.dbf &&
for redofile in 'ls -tr sysar$ORACLE_SID*.dbf 2>/dev/null'
# default name for 6.0.27 is arch*.dbf!
do
        /etc/fuser $redofile > /dev/null 2>&1 ||
        mv      $redofile ARCHIVE &&
        echo    "$redofile moved to subdir ARCHIVE"
done
echo "***0** Moved sysar* to $ORACLE_HOME/dbs/ARCHIVE0**"
```

3. Create the following post-backup script **/etc/backup.d/post.d/ORA6ARCHIVE** to delete all the backup'ed archived redologs. The post-backup script is only invoked if a backup succeeds:

```
# Post-backup script deleting backuped archived redologs from Oracle
#
ORACLE_SID=L ORAENV_ASK=NO . oraenv
rm   $ORACLE_HOME/dbs/ARCHIVE/* &&
echo "***0** Cleared $ORACLE_HOME/dbs/ARCHIVE0**"
```

4. You must label the media (tape,video) for the backup system to performe the actual backup, use *sysadm tapemgmt* and invoke *putlabel* with the label, ex. *ORA6ARCHIVE*.
5. Now you may initiate a manual backup with *sysadm totalback*, just to check the setting. When this seems to be working the specify automatic backup with *sysadm setplan*.

The list below is a rough checklist of the main issues in running Oracle in ARCHIVELOG mode:

- Turn on ARCHIVELOG mode using *archive true* with *dba* privileges.
- Make a total image backup of all disk. May use *sysadm dskback*.
- Start automatic backup of archived redologs as indicated above (specify *totalback* method). May use *sysadm totalback* for manual backup check.
- Make sure it works - try to recover in practice, regularly.

3.5 Is Oracle up?

It is sometimes required by software packages to have a running Oracle system to rely on. The following shell script shows how to check this. Unfortunately the *orastat* program does not exist from the generic proting kit from Oracle, so we have tried to emulate it as a shell script. The user should therefor be able to use the same script from Orcale 5 to check whether the database is up or not.

```
# Check if ORACLE_SID is defined
if [ "x$ORACLE_SID" = "x" ]
then
  echo "\$ORACLE_SID is not set"
  exit 0
fi
#
# Check if Oracle is running:
#
ORASTAT=`orastat -s`
if test $ORASTAT = 0
then
  echo "Oracle System $ORACLE_SID is running"
elif test $ORASTAT = 1
then
  echo "Oracle System $ORACLE_SID is not running"
else
  echo "Oracle System $ORACLE_SID is not running, "
  echo "probably 'shutdown abort', 'startup' is nessesary"
fi
```

The script may also be used in the profile of Oracle users. Not all users are familiar with an error message like this:

ERROR: ORA-01034: ORACLE not available
ORA-07318: smsget: open error when opening sgadef.dbf file.
Supermax SMOS V Error: 2: No such file or directory

3.6 Setting the ORACLE_HOME environment?

This script sets **ORACLE_HOME** and **ORACLE_SID** in the most flexible way possible.

```
#
# Check if the ORACLE_SID environment is set
#
ORAENV_ASK=NO
if [ "x$ORACLE_SID" != "x" ]
then
  . oraenv
else
#
# Look for the /etc/oratab file
#
  if test ! -r /etc/oratab
  then
    echo "*** ERROR   *** The file /etc/oratab is not readable,"
    echo "              the file is used by oraenv, dbstart, dbstop"
    exit 1
  fi

#
# Se how many entries /etc/oratab contains
#
  NO='grep '^[^#]' /etc/oratab | wc -l'
  TEST='expr $NO > 1'
  if [ "$TEST" = "1" ]
  then
    LINE='grep '^[^#]' /etc/oratab'
    X='echo $LINE | cut -c1'
    ORACLE_SID=$X . oraenv
  else
    LINE='grep '^[^#]' /etc/oratab | cat | cut -c1-2 | sed '
    s+:+,+' '
    X='echo $LINE | sed '
    s+,$+ +"
    echo "Enter ORACLE_SID [ $X ]: \c"
    ORACLE_SID='line'
    . oraenv
  fi
fi
```

If **ORACLE_SID** is set it is used. Else if only one entry in the **/etc/oratab** is found, this entry is used. Otherwise the user is asked to enter **ORACLE_SID**. On this basis *oraenv* is called, setting **ORACLE_HOME**.

3.6.1 Changing the user environment between V5 and V6

The environment for a user is normally set up using the *oraenv* script, typically

```
ORACLE_SID=DEFA . oraenv
```

The oraenv distributed with Oracle V6 is able to correctly set the environment for a V5 database as well. Using the V5 scripts for a V6 database, however, will not set up the environment correctly. It follows that the V6 version should always be used.

The oraenv script is copied to the local *bin* directory during installation, typically /usr/bin. The same is true about *dbhome* which is called from oraenv.

To make sure that users can correctly change their environment from V6 to V5 as well as from V5 to V6, the following should be observed:

1. The newest V6 version of the programs should be found in the local bin directory. This will normally always be the case unless you update your V5 software after installation of V6. In this case you must rerun the *root.install* script from your V6 distribution or copy the files manually.

2. Make an entry for every V5 database in /etc/oratab as well as /etc/ORACLE_HOME. Please observe that the V5 lines in /etc/oratab should not have 'y' or 'n' appended, or those databases will be attempted started or stopped twice during boot and shutdown.
3. Make sure that the user's PATH environment is set to search the local bin directory before the Oracle bin directory. This will be the default behaviour of the oraenv distributed with version 6.0.30 of Oracle.

3.7 Running ORACLE Batch Programs

There is a high need for batch programs in an information processing system. Wellknown examples of such programs are large reports, major updates like merging transactions etc.. In a flexible operating system like UNIX, these batch programs may be started at any time, and the operating system will of course do its best to schedule its resources so that the batch processes get their share. But in a large Oracle environment with many online users, user responce time will increase if more batch programs are started, and users will tend to be frustrated if they do not know what causes the increase.

The standard way to cope with this problem is to make sure that the batch processes gets lower priority, and therefore less resources by the operating system. This is done by using the *nice(1)* program or by starting programs in shell background (&).

There is one major problem doing so however. If the batch process requires any Oracle locks or latches, and the online applications requires the same locks or latches, then a low priority batch process may block a high priority user process. Using the priority mechanism to solve the problem should therefore be done with the extreme care, and as a general rule **only consider low priority on batch processes reading information from the database.**

Any system trying to schedule say 10 batch reports at the same time as more online users are trying to get treir things done, will run out of steam - especially in Oracle, because Oracle uses its own block cache beeing therefore cpu bound instead of disk bound - wich is nice in most situations. The keypoint here is that there should be sufficient power present to drive the online processes and a small number of batch processes in parallel. The *batch* program can unfortunately not be used for this purpose, because it may start many batch operations in parallel, but a simple shell may solve the problem:

```
# Create a fifo-file, beeing our batch mailbox
mknod batch p
# Setup the batch demon process.
sh <batch 4>batch &
```

Any time a user wants to spawn a batch process, the following script may be called:

```
# batch command file:
# $1 is the command to execute
echo "echo start $1 \`id\` \`date\` >> batch.out 2>>batch.out; \
    $1 >> batch.out 2>>batch.out; \
    echo stop $1 \`id\` \`date\` >> batch.out 2>>batch.out" > batch
```

You might want to put the *start* and *stop* handling in the shell demon instead.

The above mentioned method does not provide any queue administration features. If that is nessesary you might consider to create a spooler service, which instead of printing the document on a lineprinter, will execute the program in a sequential fasion.

3.8 Removing Duplicate Rows

First you may avoid these problems by creating a **unique** index on the key column(s).

This is however of no comfort if you have the problem already - say you imported some tables twice.

We see two different ways removing Duplicate rows from a table.

1. The easiest way is to use **distinct**:

```
create table tmp as
  select distinct * from <table>;

commit;

drop table <table>;

rename tmp to <table>;

/* Recreate indexes */
```

   But because the distinct operation will use an internal temporary table, you may not have room enough to do the operation.

2. An other approch is to scann the table for duplicates:

```
delete from <table> a
 where exists (
  select <column> from <table> b
  where a.<column 1> = b.<column 1>
   and a.<column 2> = b.<column 2>
    :     :     :     :     :
   and a.rowid < b.rowid
 );

commit;
```

   If a column may take **null** status, the **and** clause should be protected by the *nvl* function:

```
and nvl(a.<column>, 'This does not apear in the database') =
    nvl(b.<column>, 'This does not apear in the database')
```

   If all columns are protected by *nvl*, or no indexes exists, then the statement will require as many full table scanns as there are rows in the table. So be carefull!

3.9 Constraints

Oracle 6 has an enhanged *Create Table* command that supports the syntax of column and table constraints. This is explained and exemplified in the *SQL Language Reference guide* under the *Create Table* command and the *Constraint clause.*

Only the syntax is supported - there is no semantic support for these constraints, but the contents of these constraints are actually stored in tables in the database.

The default values for a column can be seen in the *Defaultvalue* column of the *Col* view.

The information on table and column constraints is stored in the CONSTRAINT_DEFS and the CONSTRAINT_COLUMNS tables. The SQL statement below selects from the USER_CONSTRAINTS and USER_CONS_COLUMNS views on these tables :

```
SELECT      c.owner owner, c.constraint_name constraint,
          ucc.table_name table_name, ucc.column_name column_name,
          constraint_type type, search_condition, position pos,
          r_constraint_name r_con
FROM user_constraints c, user_cons_columns ucc
WHERE ucc.table_name LIKE UPPER('&Table_name')
AND ucc.constraint_name = c.constraint_name
ORDER BY ucc.table_name, ucc.column_name, position
```

The tables used in the following example are called *Newemp* and *Newdept*. These tables where created using the following definitions :

```
CREATE TABLE newdept(
       deptno NUMBER(2) PRIMARY KEY,
       dname CHAR(12) UNIQUE,
       loc CHAR(13));

CREATE TABLE newemp (
       empno NUMBER NOT NULL ,
       ename CHAR(10) NOT NULL CHECK (ename= UPPER(ename)),
       job   CHAR(9)
       CHECK ( job IN ('CLERK','MANAGER','SALESMAN','ANALYST','PRESIDENT')),
       mgr   NUMBER REFERENCES newemp(empno),
       hiredate DATE CHECK (hiredate >=sysdate),
       comm  NUMBER(9,0) DEFAULT NULL,
       deptno NUMBER(2) NOT NULL REFERENCES newdept(deptno),
       PRIMARY KEY (empno)
       )
       PCTFREE 5 PCTUSED 75;
```

Here below you see a spool of the information stored in the database about these two tables (Using the select statement shown above) :

```
OWNER CONSTRAINT TABLE_NAME COLUMN_NAM T SEARCH_CONDITION      POS R_CONSTRAINT
----- ---------- ---------- ---------- - -------------------- ---- ----------
CFJ   SYS_C001378 NEWEMP    DEPTNO     R                         1 SYS_C001368
CFJ   SYS_C001372 NEWEMP    DEPTNO     C DEPTNO IS NOT NULL
CFJ   SYS_C001376 NEWEMP    EMPNO      P                         1
CFJ   SYS_C001370 NEWEMP    EMPNO      C EMPNO IS NOT NULL
CFJ   SYS_C001371 NEWEMP    ENAME      C ENAME IS NOT NULL
CFJ   SYS_C001373 NEWEMP    ENAME      C ename= UPPER(ename)
CFJ   SYS_C001375 NEWEMP    HIREDATE   C hiredate >=sysdate
CFJ   SYS_C001374 NEWEMP    JOB        C  job IN ('CLERK','MA
                                         NAGER','SALESMAN','A
                                         NALYST','PRESIDENT')
CFJ   SYS_C001377 NEWEMP    MGR        R                         1 SYS_C001376


OWNER CONSTRAINT TABLE_NAME COLUMN_NAM T SEARCH_CONDITION      POS R_CON
----- ---------- ---------- ---------- - -------------------- ---- ----------
CFJ   SYS_C001368 NEWDEPT   DEPTNO     P                         1
CFJ   SYS_C001367 NEWDEPT   DEPTNO     C DEPTNO IS NOT NULL
CFJ   SYS_C001369 NEWDEPT   DNAME      U                         1
```

The columns are :

Owner

The owner of the object.

Constraint The name of the constraint - in this case the name have been generated by the system.

Table_name The name of the table.

Column_name The name of the column.

Type The type of the constraint :

C means a *Column Constraint*,

P means a *Primary Key Constraint*,

U means a *Unique Constraint*,

R means a *Referential Constraint*,

R means a *Referential Constraint*,

Search_condition

If the type is *Column Constraint* then text of the constraint is found in this column.

Position A *Unique, Referential* or *Primary constraint* can span more columns - e.g. a primary key can be defined as the concatenation of two or more columns in the table. The *Position* shown here is the ordinal number of the constraint column in the constraint.

R_constraint_name

When dealing with *Referential constraints* this column will hold information about the constraint name of the column referenced, e.g :

The *Referential constraint* of the DEPTNO column of the NEWEMP table points to the SYS_C001368 constraint, which is the name of the *Primary Key* constraint on the DEPT table.

The manual on constrains for the 6.0 kernel, tells that although the syntax is checked and information is stored in the dictionary tables, no side effects will disturb the user tables. This hint describes an exception to this rule.

Assume we have two tables:

```
create table tab1
  (pk number, d char(100));
create table tab2
  (pk number, fk number);
```

Assume we have two constrains on these tables. A *primary key* constraint on table **tab1**, and a *foreign key* constraint from table **tab2** to table **tab1**:

```
alter table tab1 add
  (primary key (pk) constraint pri_key);
alter table tab2 add
  (foreign key (pk) references tab1 (pk) constraint for_key);
```

Suppose you want to drop table **tab1**, you will get the following error:

```
drop table tab1
           *
ERROR at line 1:
ORA-02273: this unique/primary key is referenced by some foreign keys
```

You will have to drop the *foreign key* constraint from table **tab2** in order to drop table **tab1**.

```
alter table tab2 drop constraint for_key;
```

It might be difficult in a big system with lots of tables and constrains, to find, remove and reinstall constrains, preventing a table drop.

The following SQL-statement will find the constraints, referencing the primary constraint on the table **tab1**:

```
select c.owner owner, c.constraint_name constraint,
      ucc.table_name table_name, ucc.column_name column_name,
      c.constraint_type type, c.search_condition, ucc.position pos,
      c.r_constraint_name r_con
from user_constraints c, user_cons_columns ucc,
    user_constraints sc, user_cons_columns succ
where ucc.constraint_name = c.constraint_name
 and succ.constraint_name = sc.constraint_name
 and c.r_constraint_name = sc.constraint_name
 and succ.table_name = 'TAB1'
 and sc.constraint_type = 'P'
```

3.10 Multiple Rollback segments

Rollback segments are portions of the database which records actions whichs should be undone under certain circumstances. An initial Rollback segment called SYSTEM is created when a database is created. The initial segment is created in the SYSTEM tablespace.

Multiple rollback segments are usually preferred to one rollback segment:
- If a database has multiple tablespaces it *must* have two or more rollback segments.
- And if the database will be accessed by many simultaneous users.

Rollback segments can be created as public or private for the instance. On Supermax (and other Unix platforms) it is not possible to run several instances, therefor it is not importen whether a rollback segment is public or private. But a private rollback segment is much simpler to drop than a public, because the private rollback segment must be listed in the

init.ora file to be used. If you want to drop a (private) rollback segment then just remove it from the rollback segment list in the *init.ora* file, start the database and drop it.

Further more rollback segments can be created either in specific tablespaces or as default in tablespace SYSTEM.

We recommend that You normally create rollback segments in tablespace SYSTEM. The reason to do so, is that the main part of the rollback segments normally (if the rollback segment are small enough or the buffer pool large enough) will be cashed in the main memory. In that case there is no real advantage in having rollback segments in several tablespaces.

NOTE: If you, for some reason, *do want* to place rollback segment in other tablespaces than SYSTEM, certain rools must be observed:

1. At least the first rollback segment that is created besides the SYSTEM rollback segment **have** to be created in tablespace SYSTEM, and then the following rollback segments can be created in any tablespace, but first after having shut down and restarted the database again.

   This means that you CANNOT have exactly two rollback segments placed at two table spaces: the SYSTEM and a "Tablespace-two".

2. The next point is, import of a total export covering more tablespaces **and** rollback segments in other segment than SYSTEM, then you will have to:
   1. Create tablespace and rollback segments tablespace SYSTEM.
   2. Restart the database.
   3. Create rollback segment in non-SYSTEM-tablespaces.
   4. Restart the database.
   5. And then import the tables and rows.

In Technical Bulletin Volumn 2, Number 7 from Oracle Corporation Technical Support, there is a little select, that will show the processes activ on the different rollback segments.

```
select trunc(l.id1/65536) rs_id, l.pid, p.spid, p.username, p.terminal
 from v$lock l, v$process p
where l.pid = p.pid
  and l.type = 'TX'
order by l.id1
```

```
    RS_ID        PID SPID       USERNAME   TERMINAL
---------- ---------- ---------- ---------- ----------
         0          5 26855      mj         tty53
```

3.10.1 The SYSTEM Rollback Segment

At the time of database creation only one rollback segment, *SYSTEM*, is created. If no other rollback segments are created in the database, all rollback information will be written to this SYSTEM rollback segment. The default storage parameters of SYSTEM will allow it to grow dynamically when more space is needed. It follows that if you use very large transactions or transactions of very long duration of time, the SYSTEM rollback segment may grow very large as well.

Remember that Oracle will choose rollback segments to be used by a transaction at random

with one exception: If more than the SYSTEM rollback segments can be used, then the SYSTEM rollback segment will NOT be used for user rollback information. By creating additional rollback segments you prevent the SYSTEM rollback segment from growing very large.

3.10.2 Re-sizing a Too Large Rollback Segment

If a rollback segment has grown too large, this is normally an indication that the DBA should re-consider the need for rollback segments, their number and size. The only way of diminshing a rollback segment is by dropping it and re-creating it with a smaller size. This is easily done with private rollback segments created by the DBA.

3.10.3 Creating a Smaller SYSTEM Rollback Segment

The SYSTEM rollback segment however cannot be dropped, as this rollback segment is used internally by the database. If the SYSTEM rollback segment has grown too large and you need to re-create it with a smaller size, follow these steps:

1. Calculate the need for rollback segments, their number and size, for your database. Refer to the *Oracle RDBMS DBA Guide* for further information on this issue.
2. Make a full database export of the database. Shut down the database and take a full backup of all database disks, redolog files, and control files as well for data security.
3. Run the *sqldba* program to re-create the database. This step includes the automatic creation of a default SYSTEM rollback segment.
4. Create additional rollback segments (possibly also additional tablespaces) according to your calculation in step 1.
5. Import your data.

You may wish to change the storage parameters of the SYSTEM rollback segment in order to prevent it from growing very big again. In that case you should make sure that you provide the necessary space in other rollback segments. Otherwise the result will be various error messages from Oracle, if the necessary rollback extents cannot be allocated when it is needed.

3.10.4 Considerations for Rollback Segment STORAGE

The Oracle documentation gives extensive guidance concerning *number* and *size* of rollback segments. It does not, however, give any guidance for the number and size of the *extents* of a rollback segment.

In it's *Technical Bulletin, Volume 2, Number 5*, Oracle Corporation Worldwide Technical Support has given the following suggestions in order to allow the maximum number of transactions to use a given rollback segment:

"The Moral

Create each rollback segment with several extents of equal size.

Increasing the number of extents in a rollback segment beyond 20 will have very little impact on the size of transactions able to use a rollback segment.

Page 20-22 in the ORACLE RDBMS Database Administrator's Guide Version 6.0 discusses reasons for allocating each rollback segment in the database the same amount of space. In the light of the above discussion, the rollback segments of a database should have nearly equivalent *Ne*. (Many sites benefit from an exception to this general rule. If desired, a rollback segment that has *Ne* several times larger than the *Ne* of other rollback segments

may be created. For special periods of processing where transactions with large *Nt* must be run, the database may be brought up with only the large rollback segment in use. During normal processing, the other rollback segments are brought online as well.)

Variables

Ne	The 'effective number of bytes' in rollback segment *s*. Maximizing this statistic for a rollback segment is the point of this bulletin.
Nt	Number of bytes written to rollback segment *s* by all transactions during the running of transaction *t*.
s	Rollback Segment
t	A Transaction"

NOTE: These suggestions are not part of the official Oracle Corp. documentation and are not examined in detail by DDE.

3.11 Regeneration of Tables

There is a number of reasons why a database administrator would like to be able to regenerate a table exactly as it was before. One reason could be to restore a single table correctly from an export file, including the recreation of grants, indexes, and other objectes depending on the table. Here we will present a number of small but nontrivial SQL-scripts creating parts of the information needed in order to recreate the table in question.

You may run these scripts to produce a total creation script. You might want to do some changes in order to adapt the scripts to your environment.

3.11.1 Generating the Table Drop Statement

In order to form a propor script, the table must be dropped before it is generated. The following script has this effect:

```
rem Generate drop statements
rem MJ - DDE 7. September 1989
rem run: start gendrop <table name>
rem Change the script if the table name is more than 7 characters
rem
set termout off
set heading off
set verify off
set echo off
set pause off
set pagesize 1000
spool &1.drp.sql
select 'drop table "&1";' from dual where rownum < 2;
spool off
set heading on
set termout on
set echo on
```

Figure 4. The gendrop.sql Script

The *rownum < 2* clause is present to avoid complications, should the dual consist of more rows.

3.11.2 Creating the Table Create Statement

The table create statement is generated from the *user_tab_columns* view, in order to guarentee that the sequence of columns corresponds exactly to the one in the export file:

```
rem Generate table create statements
rem MJ - DDE 7. September 1989
rem run: start gentab <table name>
rem Change the script if the table name is more than 7 characters
rem clusters and constrains are not handled yet
rem
set termout off
set heading off
set verify off
set echo off
set pause off
set pagesize 1000
set linesize 200
column a format a50
column b format a40
column c format a15
column d format a10
column e format a40
spool &1.tab.sql
break on a skip 0
select 'create table "'||max(x.table_name)||'" (' a,
decode(max(x.data_type),
      'NUMBER','('||max(x.data_precision)||','||max(x.data_scale)||')',
      'CHAR','('||max(x.data_length)||')',NULL) c,
decode(max(x.nullable),'Y','null','not null') d,
decode(x.column_id, max(y.column_id), ') ', ',') e
from user_tab_columns x, user_tab_columns y
where x.table_name = y.table_name and x.column_id < = y.column_id
  and x.table_name = '&1'
  and exists (select table_name from user_tables
         where table_name = '&1' and cluster_name is null)
group by x.table_name, x.column_id
order by x.column_id;
rem get space information
select 'pctfree '||pct_free f,
      'pctused '||pct_used g,
      'initrans '||ini_trans h,
      'maxtrans '||max_trans i,
      'tablespace "'||tablespace_name||'"' j,
      'storage (initial '||initial_extent||
            ' next '||next_extent k,
            '       minextents '||min_extents||
            ' maxextents '||max_extents||
            ' pctincrease '||pct_increase||');' l
from user_tables
where table_name = '&1';
spool off
set heading on
set termout on
set echo on
```

Figure 5. The gentab.sql Script

3.11.3 Create the Comment Create Statements

There may be comments on teh table and on its columns. This script will generate the creation script of the comments.

```
rem Generate comment statements
rem MJ - DDE 7. September 1989
rem run: start gencom <table name>
rem Change the script if the table name is more than 7 characters
rem
set termout off
set heading off
set verify off
set echo off
set pause off
set pagesize 1000
spool &1.com.sql
select 'comment on table "'||table_name||'" is '''||comments||''';'
from user_tab_comments
where table_name = '&1' and comments is not null;
select 'comment on column "'||table_name||'"."'||column_name||
     '" is '''||comments||''';'
from user_col_comments
where table_name = '&1' and comments is not null;
spool off
set heading on
set termout on
set echo on
```

Figure 6. The gencom.sql Script

3.11.4 Generate the Index Create Statements

```
rem Generate index create statements
rem MJ - DDE 7. September 1989
rem run: start geninx <table name>
rem Change the script if the table name is more than 7 characters
rem
set termout off
set heading off
set verify off
set echo off
set pause off
set pagesize 1000
set linesize 200
break on a skip 0
column a format a50
column b format a50
column c format a20
spool &1.inx.sql
select 'create '||max(i.uniqueness)||' index "'||max(i.index_name)||
      '" on "'||max(i.table_name)||'" (' a,
      '"'||max(c.column_name)||'"' b,
      decode(c.column_position, max(y.column_position),
        ' ) initrans '||max(i.ini_trans), ',') c,
      decode(c.column_position, max(y.column_position),
        ' maxtrans '||max(i.max_trans), null) d,
      decode(c.column_position, max(y.column_position),
        ' tablespace '||max(i.tablespace_name), null) e,
      decode(c.column_position, max(y.column_position),
        ' storage (initial '||max(i.initial_extent), null) f,
      decode(c.column_position, max(y.column_position),
        ' next '||max(i.next_extent), null) g,
      decode(c.column_position, max(y.column_position),
        ' minextents '||max(i.min_extents), null) h,
      decode(c.column_position, max(y.column_position),
        ' maxextents '||max(i.max_extents), null) i,
      decode(c.column_position, max(y.column_position),
        ' pctincrease '||max(i.pct_increase)||');', null) j
from user_indexes i, user_ind_columns c, user_ind_columns y
where i.index_name = c.index_name
  and c.index_name = y.index_name
  and c.column_position < = y.column_position
  and i.table_name = '&1'
group by c.index_name, c.column_position
order by c.index_name, c.column_position;
spool off
set heading on
set termout on
set echo on
```

Figure 7. The geninx.sql Script

Unfortunately the *pctfree* parametes in the Create index statement are not stored in the Data Dictionary accessible to the user, so this parameter can not be used in this automated procedure. It is however stored in the Data Dictionary table *IND$*. Because the *asc/desc* parameters are not implemented in this version of oracle, the information is not stored in the Data Dictionary, so do not expect this information to be pressent in the generated sql script.

3.11.5 Generate Autorisation Statements

This script will create a script that reflects the grants on the table for different users:

```
rem Generate table grant statements
rem MJ - DDE 7. September 1989
rem run: start genaut <table name>
rem Change the script if the table name is more than 7 characters
rem
set termout off
set heading off
set verify off
set feedback off
set echo off
set pause off
set pagesize 1000
column a format a50
column b format a20
spool &1.aut.sql
select 'grant index on "'||table_name||'" to "'||grantee||'"' a,
     decode(index_priv,'G',' with grant option', null)||';' b
from user_tab_grants_made
where table_name = '&1' and not index_priv = 'N';
select 'grant alter on "'||table_name||'" to "'||grantee||'"' a,
     decode(alter_priv,'G',' with grant option', null)||';' b
from user_tab_grants_made
where table_name = '&1' and not alter_priv = 'N';
select 'grant references on "'||table_name||'" to "'||grantee||'"' a,
     decode(references_priv,'G',' with grant option', null)||';' b
from user_tab_grants_made
where table_name = '&1' and not references_priv = 'N';
select 'grant update on "'||table_name||'" to "'||grantee||'"' a,
     decode(update_priv,'G',' with grant option', null)||';' b
from user_tab_grants_made
where table_name = '&1' and not update_priv = 'N';
select 'grant delete on "'||table_name||'" to "'||grantee||'"' a,
     decode(delete_priv,'G',' with grant option', null)||';' b
from user_tab_grants_made
where table_name = '&1' and not delete_priv = 'N';
select 'grant insert on "'||table_name||'" to "'||grantee||'"' a,
     decode(insert_priv,'G',' with grant option', null)||';' b
from user_tab_grants_made
where table_name = '&1' and not insert_priv = 'N';
select 'grant select on "'||table_name||'" to "'||grantee||'"' a,
     decode(select_priv,'G',' with grant option', null)||';' b
from user_tab_grants_made
where table_name = '&1' and not select_priv = 'N';
spool off
set heading on
set termout on
set echo on
```

Figure 8. The genaut.sql Script

3.11.6 Generate the View Create Statements

```
rem Generate view create statements
rem MJ - DDE 7. September 1989
rem run: start genview <table name>
rem Change the script if the table name is more than 7 characters
rem
set termout off
set heading off
set verify off
set echo off
set pause off
set pagesize 1000
spool &1.vie.sql
select 'create view "'||r.table_name||'" as ' a, text, ';'
from user_cross_refs r, user_views v
where r.table_name = v.view_name
  and r.table_type = 'VIEW'
  and r.ref_table_name = '&1';
spool off
set heading on
set termout on
set echo on
```

Figure 9. The genview.sql Script

3.11.7 The Result

The different spool files are concatenated into one file, and this file are processed by a streem editor, to remove empty lines and spaces:

```
# Shell script to remove empty lines
sed 's/[  ]*$//
    s/[ ][  ]*/ /g
    /^$/d'
```

Figure 10. The remline Script

The brackets contail a space and a tabulator character.

The result may look like this:

```
drop table "TESTTABLE";

create table "TESTTABLE" ("A" CHAR (100) not null,
"B" NUMBER (10,0) null,
"C" DATE null,
"D" LONG null)
pctfree 10 pctused 40 initrans 1 maxtrans 255
tablespace "SYSTEM" storage (initial 12288 next 12288
minextents 1 maxextents 99 pctincrease 50);

comment on table "TESTTABLE" is 'This is a comment on table testtable';
comment on column "TESTTABLE"."A" is 'This is a comment on column testtable.a';

create UNIQUE index "INX_A" on "TESTTABLE" ( "A" ) initrans 2 maxtrans 255
tablespace SYSTEM storage (initial 12288 next 12288
minextents 1 maxextents 99 pctincrease 50);

create NONUNIQUE index "INX_B" on "TESTTABLE" ( "B" ,
"A" ) initrans 2 maxtrans 255
tablespace SYSTEM storage (initial 12288 next 12288
minextents 1 maxextents 99 pctincrease 50);

grant index on "TESTTABLE" to "SCOTT" with grant option;
grant alter on "TESTTABLE" to "SCOTT" with grant option;
grant insert on "TESTTABLE" to "SCOTT" ;

create view "TESTVIEW" as
select * from testtable with check option;
```

Figure 11. Example Table generation Script

3.12 Maximum allowed Memory

The *Maximum allowed Menory* parameter known from the *config -m* command may have an impact on the behaviour of Oracle. The Oracle kernel will now and then try to allocate memory to do its job proporly - if it cannot - malfunction might happen. It is therefor highly recommented that the Maximum allowed Memory parameter is set to cover at least the sum of the Oracle kernel text seqment (1.1 M), the data area for open cursors (say 0.5M), and the size of the SGA (from 0.3 to 8.0 M).

So why not set it to 10.0 M right away?

3.13 Tablespace Status

One of the nice features of Oracle 6 is that a tablespace may be created and droped as the rest of the database is online. There are however some minor difficulties in examining the status of the used datafiles after a *drop tablespace* has been issued.

It is easy enough to create a new tablespace:

```
create tablespace <tablespace_name>
  datafile '<new_file_name>' size <size>;
```

Just remember to have at least two rollback segments online to use it!

You may drop the tablespace again:

```
alter tablespace <tablespace_name> offline;
drop tablespace <tablespace_name>;
```

If you now select from the *sys.dba_tablespaces* view, you will see that the *STATUS* for the dropped tablespace is *INVALID*. Now that does not hurt us, but what is more serious is that the *monitor* option *File IO* does not work any more - you get the error DBA-0333:

```
333, 0, "db file(s) added since Monitor invocation"
// *Cause: Since invoking MONITOR FILEIO a database file has been
// added, invalidating the screen.
// *Action: Reinvoke MONITOR FILEIO.
```

The error is obviously that even though the view *v$dbfile* does not show the dropped tablespace any more, the status is still in the *v$filestat* view. This error has been fixed in version 6.0.27.

You should also note that even though a tablespace may consist of more data files, an extent of an object can only be allocated in one file. So if you have more small files instead of few large ones as your tablespace You may waist diskspace if the extentsizes are big.

3.14 Rearrange Redolog and Data files

A normal duty of a database administrator is also to be able to rearrange the filestructure of the database, whenever new disks are installed and the space needed for Oracle information is grohing.

Let us first study how to change size and location of the *redolog* files. The *SQL statement reference* suggests that this could be done with a the following statement:

```
alter database <database_name>
  rename file '<old_file_name>' to '<new-file_name>';
```

This does not work. You will have to do something like this:

```
alter database <database_name>
  add logfile '<new-file_name>' size <size>;
alter database <database_name>
  drop logfile '<old-file_name>';
```

This will however only work if the logfile is not currently in use, otherwise you will get ORA-01515:

```
01515, 00000, "cannot drop log at this time; log in use"
// *Cause: ALTER DATABASE is attempting to drop a log file which
//         is currently in use.
// *Action: Wait until the log file is no longer in use.
```

Anyway, *drop* will not physical remove the file - So you should remove the file from the filesystem if you are **sure** the *drop* command succeded.

It is esier to rename a datafile belonging to a tablespace (not from the *system* tablespace).

```
alter tablespace <tablespace_name> offline;
host cp <old_file_name> <new_file_name>
alter tablespace <tablespace_name>
  rename datafile '<old_file_name>' to '<new_file_name>';
alter tablespace <tablespace_name> online;
rem remember that the <old_file_name> is now obsolete
```

If one has to rename files from the *system* tablespace, or to make a tablespace consist of one big file instead of many smaller ones, this can be done on a mounted but not opened database.

3.15 Extra V$... views

The *ORACLE RDBMS Database Administrator's Guide* Appendix E lists a number of very interesting *v$...* views. The *ORACLE RDBMS Performance Tuning Guide* adds a number of *v$...* views to the list.

Here is a list of all *v$...* views found on the version 6.0.27.8 RDBMS, not mentioned in the *Administrator's Guide*.

3.15.1 v$parameter

Lists all the parameters - and their values - specified through init.ora parameters.

```
SQL> descr v$parameter
 Name                           Null?    Type
 ------------------------------ -------- ----
 NUM                                     NUMBER
 NAME                                    CHAR(64)
 TYPE                                    NUMBER
 VALUE                                   CHAR(50)
```

3.15.2 v$sga

The size of the main areas of the SGA.

```
SQL> descr v$sga
 Name                           Null?    Type
 ------------------------------ -------- ----
 NAME                                    CHAR(20)
 VALUE                                   NUMBER

SQL> select * from v$sga;

NAME                      VALUE
-------------------- ----------
Fixed Size                22688
Variable Size            281292
Database Buffers         819200
Redo Buffers              32768
```

3.15.3 v$logfile

The redologfiles currently associated with the RDBMS.

```
SQL> descr v$logfile
 Name                           Null?    Type
 ------------------------------ -------- ----
 FILE#                                   NUMBER
 NAME                                    CHAR(255)
```

3.15.4 v$rollname

The Rollback seqments in use.

```
SQL> descr v$rollname
 Name                           Null?    Type
 ------------------------------ -------- ----
 USN                                     NUMBER
 NAME                           NOT NULL CHAR(30)

SQL> select * from v$rollname;

       USN NAME
---------- ------------------------------
         0 SYSTEM
```

3.15.5 v$filestat

This view shows some file i/o activity.

```
SQL> descr v$filestat
 Name                            Null?    Type
 ------------------------------- -------- ----
 PHYWRTS                                  NUMBER
 PHYRDS                                   NUMBER
 PHYBLKWRT                                NUMBER
 PHYBLKRD                                 NUMBER
 READTIM                                  NUMBER
 WRITETIM                                 NUMBER

SQL> select * from v$filestat;

   PHYWRTS      PHYRDS  PHYBLKWRT   PHYBLKRD    READTIM   WRITETIM
---------- ---------- ---------- ---------- ---------- ----------
       533         85       2459         85          0          0
       270         51       1102         51          0          0
```

3.15.6 v$rollstat

This view shows the activity om the rollback seqments.

```
SQL> descr v$rollstat
 Name                            Null?    Type
 ------------------------------- -------- ----
 EXTENTS                                  NUMBER
 RSSIZE                                   NUMBER
 WRITES                                   NUMBER
 XACTS                                    NUMBER
 GETS                                     NUMBER
 WAITS                                    NUMBER

SQL> select * from v$rollstat;

   EXTENTS      RSSIZE      WRITES       XACTS        GETS       WAITS
---------- ---------- ---------- ---------- ---------- ----------
         5     487440       28042          0        437          0
```

3.15.7 v$rowcache

This view gives information on the usage of the Data Dictionary Cache Elements.

```
SQL> descr v$rowcache
 Name                           Null?    Type
 ------------------------------ -------- ----
 CACHE#                                  NUMBER
 TYPE                                    CHAR(11)
 SUBORDINATE#                            NUMBER
 PARAMETER                               CHAR(32)
 COUNT                                   NUMBER
 USAGE                                   NUMBER
 FIXED                                   NUMBER
 GETS                                    NUMBER
 GETMISSES                               NUMBER
 SCANS                                   NUMBER
 SCANMISSES                              NUMBER
 SCANCOMPLETES                           NUMBER
 MODIFICATIONS                           NUMBER
 FLUSHES                                 NUMBER
```

3.15.8 v$waitstat

This view contains information about memory contention.

```
SQL> descr v$waitstat
 Name                           Null?    Type
 ------------------------------ -------- ----
 OPERATION                               CHAR(20)
 CLASS                                   CHAR(19)
 RANGE                                   CHAR(9)
 COUNT                                   NUMBER
 TIME                                    NUMBER
```

3.15.9 x$ksqdn

This x$ table - only accessable from the *sys* account, may be used to give information on the *ORACLE_SID*:

select ksqdngdn from x$ksqdn;

If you are going to use it, access the table through a view, and grant the users acces to the view.

3.16 Extent sizes

Estimating the effect of different storage parameters on the exact sizes on numbers of extents is not trivial. The following script donated from K. Kalsbøl - DDE, helps the administrator to sort this out:

```
if [ x"$1" != x"-p" -a x"$1" != x ];then
   echo "Usage: $0 [ -p : print to
   exit
fi
if [ "$1" = "-p" ];then
   OUTLIST="$0.lst"
else
   OUTLIST="/dev/null"
fi
echo "Indtast initial . . . . . .  (default 12k): read initial_ext
initial_ext=${initial_ext:-12k}
echo $initial_ext >> $OUTLIST
echo "Indtast next  . . . . . .  (default 12k): read next_incr
next_incr=${next_incr:-12k}
echo $next_incr >> $OUTLIST
echo "Indtast pctincrease . . . . . .  (default 50 ): \c" | tee -a $OUTLIST
read pct_incr
pct_incr=${pct_incr:-50}
echo $pct_incr >> $OUTLIST
echo "Indtast estimeret diskforbrug i MB (default 20 ): \c" | tee -a $OUTLIST
read disk_mb
disk_mb=${disk_mb:-20}
echo $disk_mb >> $OUTLIST
echo $initial_ext $next_incr $pct_incr $disk_mb | awk '
BEGIN { max_ext=249; block_size=4096 }
{
  initial_ext=$1;
  next_incr=$2;
  pct_incr=$3;
  disk_mb=$4;
  if (initial_ext ~ /(k|K)/) {
      initial_ext=sprintf("%d",initial_ext*1024);
  } else {
      initial_ext=sprintf("%d",initial_ext);
  }
  if (next_incr ~ /(k|K)/) {
      next_incr=sprintf("%d",next_incr*1024);
  } else {
      next_incr=sprintf("%d",next_incr);
  }
  initial_ext=orablock(initial_ext);
  next_incr=orablock(next_incr);
  tot_size=initial_ext;
  next_ext=next_incr;
  diskuse=disk_mb * 1048576;
  printf"\n ext_id=%3d,  ext_size=%8d,  blocks= %4d\n",
         0, tot_size, tot_size/block_size;
  for (ext_id=1;ext_id<max_ext;ext_id++) {
      tot_size+=next_ext;
      printf" ext_id=%3d,  ext_size=%8d,  blocks= %4d\n",
             ext_id, next_ext, next_ext/block_size;
      if (tot_size > diskuse) { ext_id++; break }
      next_ext=ora_incr(next_ext);
  }
}
function ora_incr(x) {
  return(int((x+(x*pct_incr/100)+(block_size-1))/block_size)*block_size);
}
function orablock(x) {
  if (x<1) x=1;
  return(int((x+(block_size-1))/block_size)*block_size);
}
END{
```

Extent sizes MJ

```
printf"\nextents=%3d,  tot_size=%0.3lfMB,  blocks= %d   %s\n",
       ext_id,tot_size/1048576,tot_size/block_size,
       (ext_id == max_ext ? "(NB: max.exts. exceeded!)" : "");
printf"\ninitial: %d bytes, next: %d bytes, pct_incr: %d, est.fbr.: %dMB\n",
         initial_ext,next_incr,pct_incr,disk_mb;
}' | tee -a $OUTLIST
```

If you f.eks. run the script with the parameters in default (12k, 12k, 50, 20M), you will se the result:

```
Indtast initial . . . . . .  (default 12k): 12k
Indtast next  . . . . . .  (default 48k): 12k
Indtast pctincrease . . . . . .  (default  1 ): 50
Indtast estimeret diskforbrug i MB (default 20 ): 20

 ext_id=  0,  ext_size=   12288,  blocks=    3
 ext_id=  1,  ext_size=   12288,  blocks=    3
 ext_id=  2,  ext_size=   20480,  blocks=    5
 ext_id=  3,  ext_size=   32768,  blocks=    8
 ext_id=  4,  ext_size=   49152,  blocks=   12
 ext_id=  5,  ext_size=   73728,  blocks=   18
 ext_id=  6,  ext_size=  110592,  blocks=   27
 ext_id=  7,  ext_size=  167936,  blocks=   41
 ext_id=  8,  ext_size=  253952,  blocks=   62
 ext_id=  9,  ext_size=  380928,  blocks=   93
 ext_id= 10,  ext_size=  573440,  blocks=  140
 ext_id= 11,  ext_size=  860160,  blocks=  210
 ext_id= 12,  ext_size= 1290240,  blocks=  315
 ext_id= 13,  ext_size= 1937408,  blocks=  473
 ext_id= 14,  ext_size= 2908160,  blocks=  710
 ext_id= 15,  ext_size= 4362240,  blocks= 1065
 ext_id= 16,  ext_size= 6545408,  blocks= 1598
 ext_id= 17,  ext_size= 9818112,  blocks= 2397

extents= 18,  tot_size=28.047MB,  blocks= 7180

initial: 12288 bytes, next: 12288 bytes, pct_incr: 50, est.fbr.: 20MB
```

Whereas if you f.eks. run the script with PCTINCREASE set to only 1, you will se the result:

```
Indtast initial . . . . . .  (default 12k): 12k
Indtast next  . . . . . .  (default 48k): 12k
Indtast pctincrease . . . . . .  (default  1 ): 1
Indtast estimeret diskforbrug i MB (default 20 ): 20

 ext_id=  0,  ext_size=   12288,  blocks=    3
 ext_id=  1,  ext_size=   12288,  blocks=    3
 ext_id=  2,  ext_size=   16384,  blocks=    4
 ext_id=  3,  ext_size=   20480,  blocks=    5
 ext_id=  4,  ext_size=   24576,  blocks=    6
 ext_id=  5,  ext_size=   28672,  blocks=    7
 ext_id=  6,  ext_size=   32768,  blocks=    8
 ext_id=  7,  ext_size=   36864,  blocks=    9
 ext_id=  8,  ext_size=   40960,  blocks=   10
 ext_id=  9,  ext_size=   45056,  blocks=   11
 ext_id= 10,  ext_size=   49152,  blocks=   12
 ext_id= 11,  ext_size=   53248,  blocks=   13
 ext_id= 12,  ext_size=   57344,  blocks=   14
 ext_id= 13,  ext_size=   61440,  blocks=   15
 ext_id= 14,  ext_size=   65536,  blocks=   16
 ext_id= 15,  ext_size=   69632,  blocks=   17
 ext_id= 16,  ext_size=   73728,  blocks=   18
 ext_id= 17,  ext_size=   77824,  blocks=   19
 ext_id= 18,  ext_size=   81920,  blocks=   20
 ext_id= 19,  ext_size=   86016,  blocks=   21
 ext_id= 20,  ext_size=   90112,  blocks=   22
 ext_id= 21,  ext_size=   94208,  blocks=   23
 ext_id= 22,  ext_size=   98304,  blocks=   24
 ext_id= 23,  ext_size=  102400,  blocks=   25
 ext_id= 24,  ext_size=  106496,  blocks=   26
 ext_id= 25,  ext_size=  110592,  blocks=   27
 ext_id= 26,  ext_size=  114688,  blocks=   28
 ext_id= 27,  ext_size=  118784,  blocks=   29
 ext_id= 28,  ext_size=  122880,  blocks=   30
 ext_id= 29,  ext_size=  126976,  blocks=   31
 ext_id= 30,  ext_size=  131072,  blocks=   32
 ext_id= 31,  ext_size=  135168,  blocks=   33
 ext_id= 32,  ext_size=  139264,  blocks=   34
 ext_id= 33,  ext_size=  143360,  blocks=   35
 ext_id= 34,  ext_size=  147456,  blocks=   36
 ext_id= 35,  ext_size=  151552,  blocks=   37
 ext_id= 36,  ext_size=  155648,  blocks=   38
 ext_id= 37,  ext_size=  159744,  blocks=   39
 ext_id= 38,  ext_size=  163840,  blocks=   40
 ext_id= 39,  ext_size=  167936,  blocks=   41
 ext_id= 40,  ext_size=  172032,  blocks=   42
 ext_id= 41,  ext_size=  176128,  blocks=   43
 ext_id= 42,  ext_size=  180224,  blocks=   44
 ext_id= 43,  ext_size=  184320,  blocks=   45
 ext_id= 44,  ext_size=  188416,  blocks=   46
 ext_id= 45,  ext_size=  192512,  blocks=   47
 ext_id= 46,  ext_size=  196608,  blocks=   48
 ext_id= 47,  ext_size=  200704,  blocks=   49
 ext_id= 48,  ext_size=  204800,  blocks=   50
 ext_id= 49,  ext_size=  208896,  blocks=   51
 ext_id= 50,  ext_size=  212992,  blocks=   52
 ext_id= 51,  ext_size=  217088,  blocks=   53
 ext_id= 52,  ext_size=  221184,  blocks=   54
 ext_id= 53,  ext_size=  225280,  blocks=   55
 ext_id= 54,  ext_size=  229376,  blocks=   56
 ext_id= 55,  ext_size=  233472,  blocks=   57
 ext_id= 56,  ext_size=  237568,  blocks=   58
 ext_id= 57,  ext_size=  241664,  blocks=   59
 ext_id= 58,  ext_size=  245760,  blocks=   60
 ext_id= 59,  ext_size=  249856,  blocks=   61
```

```
ext_id= 60,  ext_size=  253952,  blocks=   62
ext_id= 61,  ext_size=  258048,  blocks=   63
ext_id= 62,  ext_size=  262144,  blocks=   64
ext_id= 63,  ext_size=  266240,  blocks=   65
ext_id= 64,  ext_size=  270336,  blocks=   66
ext_id= 65,  ext_size=  274432,  blocks=   67
ext_id= 66,  ext_size=  278528,  blocks=   68
ext_id= 67,  ext_size=  282624,  blocks=   69
ext_id= 68,  ext_size=  286720,  blocks=   70
ext_id= 69,  ext_size=  290816,  blocks=   71
ext_id= 70,  ext_size=  294912,  blocks=   72
ext_id= 71,  ext_size=  299008,  blocks=   73
ext_id= 72,  ext_size=  303104,  blocks=   74
ext_id= 73,  ext_size=  307200,  blocks=   75
ext_id= 74,  ext_size=  311296,  blocks=   76
ext_id= 75,  ext_size=  315392,  blocks=   77
ext_id= 76,  ext_size=  319488,  blocks=   78
ext_id= 77,  ext_size=  323584,  blocks=   79
ext_id= 78,  ext_size=  327680,  blocks=   80
ext_id= 79,  ext_size=  331776,  blocks=   81
ext_id= 80,  ext_size=  335872,  blocks=   82
ext_id= 81,  ext_size=  339968,  blocks=   83
ext_id= 82,  ext_size=  344064,  blocks=   84
ext_id= 83,  ext_size=  348160,  blocks=   85
ext_id= 84,  ext_size=  352256,  blocks=   86
ext_id= 85,  ext_size=  356352,  blocks=   87
ext_id= 86,  ext_size=  360448,  blocks=   88
ext_id= 87,  ext_size=  364544,  blocks=   89
ext_id= 88,  ext_size=  368640,  blocks=   90
ext_id= 89,  ext_size=  372736,  blocks=   91
ext_id= 90,  ext_size=  376832,  blocks=   92
ext_id= 91,  ext_size=  380928,  blocks=   93
ext_id= 92,  ext_size=  385024,  blocks=   94
ext_id= 93,  ext_size=  389120,  blocks=   95
ext_id= 94,  ext_size=  393216,  blocks=   96
ext_id= 95,  ext_size=  397312,  blocks=   97
ext_id= 96,  ext_size=  401408,  blocks=   98
ext_id= 97,  ext_size=  405504,  blocks=   99
ext_id= 98,  ext_size=  409600,  blocks=  100
ext_id= 99,  ext_size=  413696,  blocks=  101

extents=100,  tot_size=20.121MB,  blocks= 5151

initial: 12288 bytes, next: 12288 bytes, pct_incr: 1, est.fbr.: 20MB
```

As you see even PCTINCREASE 1, will make room for tables in most cases.

3.17 Index Inspection

Oracle version 6.0 contains mechanisms to check the actual data usage of the indexes. By inspecting this information, one may decide if an index should become *unique*, if the index should be reorganized, or even dropped.

First run the *validate index* command on a given index, and afterwards inspect the *index_stats* view for information on that index.

A B*tree consists of three kinds of blocks. The root block, the leaves, and the branch blocks. Oracle may allocate extra blocks to the index in order to control splitting and other tasks.

An empty B*Tree consists of tree blocks. The root block, one leaf block, and a temporary block. The temporary block will not be allocated, if the index is not empty upon creation.

The usable space in a leaf block in a 4K block system is 3924 bytes.

Running the *validate index <IndexName>* command may issue the ORA-08100 error (index is not valid - see trace file for diagnostics") when a heavy update of the index is taking place - do not be alarmed.

The **index_stats** table - or view as it is, is actually created in the *?/rdbms/admin/catalog.sql* file, and has the following columns:

height	Height of the b-tree. The number of levels excluding the root. The number is between 1 and 3. 4 and 5 indicates that very big indexes are at hand.
blocks	Blocks allocated to the segment. The block size may be determined by the following select: **select** value **from** v$parameter **where** name = 'db_block_size'
name	Name of the index. The user and table names are absent here.
lf_rows	Number of leaf rows (values in the index). Note that since the null states are not stored in the index, this number reflects the number of not null rows being allocated in the index.
lf_blks	Number of leaf blocks in the b-tree.
lf_rows_len	Sum of the lengths of all the leaf rows.
lf_blk_len	Useable space in a leaf block.
br_rows	Number of branch rows. The number of branch rows will be one less the number of leaf blocks.
br_blks	Number of branch blocks in the b-tree.
br_rows_len	Sum of the lengths of all the branch blocks in the b-tree.
br_blk_len	Useable space in a branch block.
del_lf_rows	Number of deleted leaf rows in the index. The number of entries used in the index is therefor *lf_rows - del_lf_rows*.
del_lf_rows_len	Total length of all deleted rows in the index.
distinct_keys	Number of distinct keys in the index. If the number of distinct keys is very low compared with the total number of keys, the use of the index should be avoided, and the index dropped.
most_repeated_key	How many times the most repeated key is repeated. If this number is 1, this index is definitely a candidate for a *unique* index.
btree_space	Total space currently allocated in the b-tree.
used_space	Total space that is currently being used in the b-tree. A regenerated index will use the space: *used_space - del_lf_rows_len*.

pct_used	Percent of space allocated in the b-tree that is being used.
rows_per_key	Average number of rows per distinct key.
blks_gets_per_access	Expected number of consistent mode block gets per row. This assumes that a row chosen at random from the table is being searched for using the index.

If the index is empty, the select from index_stats view will result in an ORA-01476 error (divisor is equal to zero), because the two last columns *rows_per_key* and *blks_gets_per_access* are calculated using a division by the number of entries.

3.17.1 Index_Stats_New

Here we will suggest a new adjusted **index_stats** view (*index_stats_new*) to treat the empty indexes correct. Null will be used instead of division by zero, the user name for the index owner will be prefixed the index name in the name column, and the actual table name on which the index is working is supplied.

```
drop view INDEX_STATS_NEW;

create view INDEX_STATS_NEW as
  select kdxstrot+1 height,
    s.blocks,
    u.name||'.'||o.name index_name,
    tab.name  table_name,
    kdxstlrw lf_rows,
    kdxstlbk lf_blks,
    kdxstlln lf_rows_len,
    kdxstlub lf_blk_len,
    kdxstbrw br_rows,
    kdxstbbk br_blks,
    kdxstbln br_rows_len,
    kdxstbub br_blk_len,
    kdxstdrw del_lf_rows,
    kdxstdln del_lf_rows_len,
    kdxstdis distinct_keys,
    kdxstmrl most_repeated_key,
    kdxstlbk*kdxstlub+kdxstbbk*kdxstbub btree_space,
    kdxstlln+kdxstbln used_space,
    decode(kdxstlbk+kdxstbbk,0,null,
      ceil(((kdxstlln+kdxstbln)*100)/(kdxstlbk*kdxstlub+kdxstbbk*kdxstbub))) pct_used,
    decode(kdxstdis,0,null,kdxstlrw/kdxstdis) rows_per_key,
    decode(kdxstdis,0,null,kdxstrot+1+(kdxstlrw+kdxstdis)/(kdxstdis*2)) blks_gets_per_access
  from obj$ o, ind$ i, seg$ s, user$ u, obj$ tab, x$kdxst
  where kdxstfil = s.file#
    and kdxstblk = s.block#
    and s.file# = i.file#
    and s.block# = i.block#
    and i.obj# = o.obj#
    and o.owner# = u.user#
    and i.bo# = tab.obj#;

drop public synonym INDEX_STATS_NEW;

create public synonym INDEX_STATS_NEW for INDEX_STATS_NEW;

grant select on INDEX_STATS_NEW to public;
```

Figure 12. Create the Index_Stats_New View

3.17.2 *Inspecting All Indexes*

The following script will validate all indexes in the system (except those owned by *sys*), and will insert values in the **system.all_index_stats_new** table about all these indexes. Note that we assume all indexes to validate ok, but the script may also be used simply to validate all the indexes.

```
rem table.sql
rem MJ 2. July 1991
rem Inspect indexes of the database.
rem
set heading off
set linesize 300
set pagesize 1000
set feedback off
set arraysize 1
set termout off
set tab on
set verify off
spool /tmp/map.ud

select 'Indexes, Dato: ', to_char(sysdate,'dd. mon yyyy hh24:mi:ss')
from dual;

drop table all_index_stats_new;

create table all_index_stats_new as select * from index_stats_new;
delete from all_index_stats_new;
commit;
spool off
host sed 's/ *$//g' < /tmp/map.ud

spool tab.sql
select 'start inxstat '||owner||'.'||index_name||';' a
from sys.dba_indexes
where owner != 'SYS';
spool off
host sed 's/ *$//g' < tab.sql | sed '/^ *$/d' >/tmp/tab.sql
host mv /tmp/tab.sql tab.sql
spool /tmp/map.ud
start tab.sql
spool off
host sed 's/ *$//g' < /tmp/map.ud
host rm /tmp/map.ud
rem host rm tab.sql
set termout on
exit
```

Figure 13. Total Index Inspection, index.sql

```
rem  inxstat.sql
rem  (Called from the generated tab.sql script)
rem  MJ / DDE 23. AUG. 91

define I = &1

prompt validate index &I;
validate index &I;

insert into all_index_stats_new select * from index_stats_new;
commit;
```

Figure 14. Index Inspection, inxstat.sql

The scripts may be called like this:

```
$ sqlplus -s $SYSTEM_PASS @index > index.out
```

3.17.3 Selecting Indexes to Inspect

The **all_index_stats_new** table is queried for indexes to look at.

First let us get an overview of the indexes of the system, ordered by the number of bytes allocated.

```
select i.name||' on table '||i.table_name "Index Name",
    i.blocks*p.value "Bytes Allocated"
from all_index_stats_new i, v$parameter p
where p.name = 'db_block_size'
order by i.blocks desc
```

Next, let us see which indexes are heigh (more than 2 levels).

```
select i.name||' on table '||i.table_name "Index Name",
    i.height "Index Heigth",
    i.br_blks "Branch Blocks"
from all_index_stats_new i
where i.height > = 3
order by i.height desc, i.name
```

If there are indexes, where the deleted space is more than 1/10 the used space, you could save space be regenerating them.

```
select i.name||' on table '||i.table_name "Index Name",
    i.used_space "Total used Space",
    i.del_lf_rows_len "Deletes Space"
from all_index_stats_new i
where i.used_space/(i.del_lf_rows_len+1) < 10
order by i.used_space
```

The list of empty indexes, can easily be obtaind. From a performance standpoint it is advisable to load a table prior to creating the indexes.

```
select i.name||' on table '||i.table_name "Index Name",
     i.used_space "Total used Space"
from all_index_stats_new i
where i.lf_rows - i.del_lf_rows = 0
order by i.used_space
```

If the number of distinct keys are less than 1/10 the number of entries, the index has in our terms, a bad key distribution. You may reconsider the use of the index.

```
select i.name||' on table '||i.table_name "Index Name",
     i.lf_rows "Number of Entries",
     i.distinct_keys "Number of Distinct Entries"
from all_index_stats_new i
where i.lf_rows/i.distinct_keys > = 10
  and i.lf_rows > 100
order by i.lf_rows
```

Here we select indexes defined as non-unique, which could actually be defined as unique if desired. The query optimizer will rank unique indexes higher than non-unique.

```
select i.name||' on table '||i.table_name "Index Name"
from all_index_stats_new i, sys.dba_indexes d
where i.name = d.owner||'.'||d.index_name
  and i.table_name = d.table_name
  and d.uniqueness = 'NONUNIQUE'
  and i.most_repeated_key = 1
order by i.name
```

3.17.4 Inspecting Columns for relevant for indexing

There is no general automatic procedure to find the proper columns in a table to index, but here is some vague guidelines.

- Each table should have a unique index on the primary key columns.
- Large tables should have an index on their foreign key columns.
- Extra indexes could be created if other columns are frequently accesed in a sorted order.

If you have specified the primary, unique and foreign key referential constraints to the rdbms, you may select the tables and columns to check:

```
rem
rem conssel.sql
rem MJ - DDE 28. Aug 91
rem This script will select all the users constraints, and list
rem the tables and columns, that could be subject to an index inspection.
rem
break on head1
select 'Table: '||max(def.table_name)||', Constraint: '||
     max(def.constraint_type)||' - '||max(def.constraint_name) head1,
     'Column '||colx.position||': '||max(colx.column_name)
from constraint_defs def, constraint_columns colx, constraint_columns coly
where def.constraint_name = colx.constraint_name
  and def.owner = colx.owner
  and def.constraint_name = coly.constraint_name
  and def.owner = coly.owner
  and colx.position < = coly.position
  and def.constraint_type != 'C'
group by colx.constraint_name, colx.position
order by 1, colx.constraint_name, colx.position;
```

Figure 15. The conssel.sql Script

Possible output from the script could look like this:

```
Table: CUSTOMER, Constraint: P - CUSTOMER_PRIMARY_KEY
Column 1: CUSTID

Table: ORD, Constraint: P - ORD_PRIMARY_KEY
Column 1: ORDID

Table: ORD, Constraint: R - ORD_FOREIGN_KEY
Column 1: CUSTID

Table: T1, Constraint: P - PK
Column 1: PK

Table: X, Constraint: P - SYS_C00416
Column 1: B
Column 2: A
```

The rdbms 6.0 distribution contains the following scripts in the dictionary **?/rdbms/admin**: **idxstat.sql, oneidxs.sql** and **dispidx,sql**. These scripts are described in **?/rdbms/doc/idxstat.doc**.

You may check all the tables / columns having a non-unique index on a per user basis from *sqlplus* with:

```
SQL> start idxstat % %
```

And then see the results with:

```
SQL> start dispidxs % %
```

Remember to set the environment variable *SQLPATH* to ':?/rdbms/admin', before running

the scripts with *sqlplus*.

Note that these scripts are limited to work only for single columns.

Here is a script (**idxs.sql**), that will check multi column keys, for their 'badness':

```
rem
Rem Copyright (c) 1989 by Oracle Corporation and 1991 Dansk Data Elektronik A/S
Rem NAME
Rem  IDXS.SQL
Rem FUNCTION
Rem   See below
Rem NOTES
Rem   See below
Rem MODIFIED
Rem  Porter   09/23/89 - Change to 8-character filenames
Rem  Porter   04/04/89 - Commenting, cleanup
Rem  Porter   03/27/89 - Creation
Rem  MJ - DDE  28. Aug 91 - Taken from oneidxs.sql, and modified to
Rem                  take care of multi column inspection.
Rem
Rem-----------------------------------------------------------------
Rem                      IDXS.SQL
Rem
Rem   Use this procedure to find out information about how
Rem   selective columns are. Use it to:
Rem
Rem     1. Identify prospective columns for new indexes
Rem     2. Determine how selective a current index is
Rem     3. Determine whether a current index is useful or not
Rem
Rem   SQL> START IDXS TABLE_NAME COLUMN_NAME COLUMN_NAME2 ... COLUMN_NAME5
Rem
Rem   If only 2 columns are to be inspected supply the values 'null'
Rem   to the column_names form 3, 4 and 5.
Rem
Rem   eg.  START IDXS EMP ENAME JOB NULL NULL NULL
Rem
Rem   NOTE: This procedure requires SQLPLUS version 3.0.3.1 or greater,
Rem       in order for the NEW_VALUE statement to be implemented
Rem       correctly.
Rem-----------------------------------------------------------------

Rem
Rem *** Set up variables ***
Rem
Set Heading Off
Set Verify Off
Set Feedback Off
column table_name new_value one_table_name
column column_name new_value one_column_name
column column_name2 new_value two_column_name
column column_name3 new_value tree_column_name
column column_name4 new_value four_column_name
column column_name5 new_value five_column_name
select upper('&1') table_name,
     upper('&2') column_name,
     upper('&3') column_name2,
     upper('&4') column_name3,
     upper('&5') column_name4,
     upper('&6') column_name5
 from dual;
```

```
column column_names new_value column_name_list
select '&one_column_name'||
     decode('&two_column_name','null',null,',&two_column_name')||
     decode('&tree_column_name','null',null,',&tree_column_name')||
     decode('&four_column_name','null',null,',&four_column_name')||
     decode('&five_column_name','null',null,',&five_column_name') column_names
 from dual;

Set Heading On
Set Verify On
Set Feedback On

SET TERMOUT OFF

Rem
Rem    Table statistics:
Rem
Rem    If an indexed column has nulls in it, then care must be used
Rem    in analyzing performance of that index:
Rem      1. A query for a null value will do a table-scan, and
Rem         will never use an index.
Rem      2. A query for a non-null value will only have to search
Rem         an index that has no null entries in it.  Thus, no
Rem         performance will be lost between a table with 100 not-null
Rem         entries and 100 not-null and 1000 null entries, as long
Rem         as queries for not-null values are made.
Rem


Rem
Rem *** Create user statistics tables, if not already created ***
Rem
create table index$index_stats (
  table_name char(30) not null,
  column_name char(155) not null,
  stat_name char(30) not null,
  stat_value number not null);

create table index$badness_stats
  (table_name char(30) not null,
   column_name char(155) not null,
   badness_factor number(9),
   keys_with_badness number(9),
   row_percent number(9,3),
   row_blk_fail number(9),
   row_blk_succ number(9),
   key_percent number(9,3));

Rem
Rem *** Get rid of any current lines in the tables ***
Rem
delete from index$index_stats
  where table_name = '&one_table_name' and column_name = '&column_name_list';
delete from index$badness_stats
  where table_name = '&one_table_name' and column_name = '&column_name_list';

Rem
Rem *** GET STATISTICS ***
Rem

column records new_value tot_rows;
select count(*) records
  from &one_table_name;
insert into index$index_stats values('&one_table_name','&column_name_list',
                        'Rows - Total', &tot_rows);
```

```
column nulls new_value tot_nulls;
select count(*) nulls
 from &one_table_name
 where &one_column_name is null
   and &two_column_name is null
   and &tree_column_name is null
   and &four_column_name is null
   and &five_column_name is null;
insert into index$index_stats values('&one_table_name','&column_name_list',
                        'Rows - Null', &tot_nulls);

drop view index$badness;
create view index$badness as (
 select count(*) repeat_count
   from &one_table_name
   group by &column_name_list);

column distinct_keys new_value tot_distinct_keys
select count(*) distinct_keys
 from index$badness;
insert into index$index_stats values('&one_table_name','&column_name_list',
                        'Total Distinct Keys', &tot_distinct_keys);

column not_nulls new_value tot_not_nulls;
select &tot_rows - &tot_nullS not_nulls
 from dual;
insert into index$index_stats values ('&one_table_name','&column_name_list',
                        'Rows - Not null', &tot_not_nulls);

Rem
Rem    Following are statistics for the distribution of the keys,
Rem    without null values, since null values can perturb the
Rem    statistics, and indexes don't use nulls.
Rem
Rem    Of particular interest here is the average number of
Rem    rows per key, since this is a general measure of the
Rem    selectivity of the column.
Rem

column average  new_value average
column minimum  new_value minimum
column maximum  new_value maximum
column std_dev  new_value std_dev
select nvl(avg(count(*)),0) average,
       nvl(min(count(*)),0) minimum,
       nvl(max(count(*)),0) maximum ,
       nvl(stddev(count(*)),0) std_dev
 from &one_table_name
 where &one_column_name is not null
   or &two_column_name is not null
   or &tree_column_name is not null
   or &four_column_name is not null
   or &five_column_name is not null
 group by &column_name_list;
insert into index$index_stats values('&one_table_name','&column_name_list',
                          'Rows per key - avg',&average);
insert into index$index_stats values('&one_table_name','&column_name_list',
                          'Rows per key - min',&minimum);
insert into index$index_stats values('&one_table_name','&column_name_list',
                          'Rows per key - max',&maximum);
insert into index$index_stats values('&one_table_name','&column_name_list',
                          'Rows per key - dev',&std_dev);

Rem
Rem    The following table is a histogram of key selectivity in the
```

```
Rem     columns.  This can be used to determine what the overall
Rem     'badness' (a technical term) of the column is.  NOTE: This
Rem     key selectivity is measured WITHOUT NULLS, since nulls don't
Rem     use indexes.
Rem
Rem     The 'badness_factor' column tells how many times a key was
Rem     repeated.
Rem     The 'keys_with_badness' column tells how many keys were
Rem     repeated 'badness_factor' times.
Rem
Rem     Higher badness factors are detrimental to the selectivity
Rem     of the column.
Rem
Rem     This table takes on different meanings depending upon
Rem     which access method is assumed:
Rem     1. Access distributed equally across rows
Rem        In this case, the user should look at the 'ROW_PERC'
Rem        column to determine what percentage of the queries will
Rem        have high badness factors.
Rem     2. Access distributed equally across key values
Rem        In this case, the user should look at the 'KEY_PERC'
Rem        column to determine what percentage of the queries will
Rem        have high badness factors.
Rem

insert into index$badness_stats (table_name,column_name,
   badness_factor, keys_with_badness, row_percent,
   row_blk_fail, row_blk_succ, key_percent)
 select '&one_table_name' table_name,'&column_name_list' column_name,
   repeat_count badness_factor,
   count(repeat_count) keys_with_badness,
   (count(repeat_count)*repeat_count/&tot_not_nulls)*100 row_percent,
   (count(repeat_count)*repeat_count*repeat_count) row_blk_fail,
   (count(repeat_count)*repeat_count*ceil(repeat_count/2)) row_blk_succ,
   (count(repeat_count)/&tot_distinct_keys)*100 key_percent
 from index$badness
 where repeat_count < > 0
 group by repeat_count;

Rem
Rem     The following two statistics attempt to determine
Rem     how expensive it is to access keys in this table.
Rem     A 'miss' will have to scan all the rows for a key value,
Rem     and a 1-row hit will, on the average, only have to scan
Rem     half of them.
Rem
Rem     These gets are the gets to read data from the table.
Rem     Gets needed to access branch and leaf nodes of the index
Rem     are not counted here.
Rem

column gets_per_miss_by_row new_value miss_gets
column gets_per_1_row_hit_by_row new_value hit_gets
select nvl((sum(row_blk_fail)/&tot_not_nulls),0) gets_per_miss_by_row,
     nvl((sum(row_blk_succ)/&tot_not_nulls),0) gets_per_1_row_hit_by_row
 from index$badness_stats
 where table_name like '&one_table_name'
   and column_name like '&column_name_list';
insert into index$index_stats values('&one_table_name','&column_name_list',
                          'db_gets_per_key_miss',&miss_gets);
insert into index$index_stats values('&one_table_name','&column_name_list',
                          'db_gets_per_key_hit',&hit_gets);

drop view index$badness;
SET TERMOUT ON
```

```
Rem
Rem Clean up
Rem
undefine 1
undefine 2
undefine one_table_name
undefine column_name_list
undefine one_column_name
undefine two_column_name
undefine tree_column_name
undefine four_column_name
undefine five_column_name
```

Figure 16. The idxs.sql Script

4. SQL*Forms 2.3 Hints

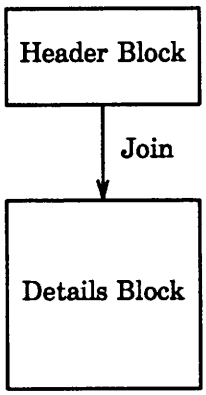
4.1 Referencing between blocks

If you consider an application developed by SQL*Forms working on two tables, where each record in the *Header* table refers to one or more rows in the *Details Table*, that is : There is a one to many relation between these two tables.

The example form used here is an *Order Entry Form* where each Order Entry will have a heading and several items. Building the application heading and corresponding items will be put into seperate blocks.

This hints shows you how coordinate the blocks, so that a query in first block will also fetch the corresponding rows in the second block - this is called *Query-Coordination*.

This example assumes that the *Unique Key* of each Header record is generated automatically, as described in the hint *Generating Sequence Numbers* just below.



Having heading and items in separate blocks it is needed to ensure correct referencing between heading and items connected to the specific heading.

Considering an application build for handling orderforms, this paper describes how SQL*Forms may handle the general problem of referencing between heading and items.

The approach used in these recommendations is to leave the operator in charge of committing. However to ensure correct referencing it is necessary to ask the user if he wants to commit. This is done by applying the *clrblk* macro.

The Foreign Key field(s) in the items block should be non-enterable, non-mandatory and their default values should be taken as copies of the Primary Key in the header block. This is done using the *Copy Field Value From* in the *Validation* window for these fields.

```
Field: *id  Number: Displayed 4 Stored 44
Page: 1 Line: 2 Col: 65  Query 44
Primary key Default: :head.id Non-enterable
```

The corresponding fields in the head block should only be enterable.

The referencing is ensured when displaying by making a where clause for the items block like

```
where id = :head.id
```

In order to show items, whenever the operator wants access to a new heading and clear the items when the heading is cleared, the following triggers are recommended at application-level:

```
TRIGGER showitem: 1
    #exemacro nofail nxtblk;
    case head.id is
      when '' then clrblk;
      when others then exeqry;
    end case; prvblk;
    *
    If step NOT successful:
    Global error showitem
    Error-stop;
TRIGGER clearitem: 1
    #exemacro nxtblk; clrblk; prvblk;
    *
    If step NOT successful:
    Global error clearitem
    Error-stop;
```

These triggers will be used througout the application as shown below.

The following head triggers are all at heading block-level using *showitem* and *clearitem*.

```
TRIGGER KEY-CLRBLK: 1
    #exemacro exetrg clearitem; clrblk;
    *
    If step NOT successful:
    Head error KEY-CLRBLK
    Error-stop;

TRIGGER KEY-CLRREC: 1
    #exemacro exetrg clearitem; clrrec;
    *
    If step NOT successful:
    Head error KEY-CLRREC
    Error-stop;

TRIGGER KEY-CREREC: 1
    #exemacro exetrg clearitem; crerec;
    *
    If step NOT successful:
    Head error KEY-CREREC
    Error-stop;

TRIGGER KEY-DELREC: 1
    #exemacro delrec; exetrg showitem;
    *
    If step NOT successful:
    Head error KEY-DELREC
    Error-stop;

TRIGGER KEY-EXEQRY: 1
    #exemacro exetrg clearitem; exeqry; exetrg showitem;
    *
    If step NOT successful:
    Head error KEY-ENTQRY
    Error-stop;
TRIGGER KEY-ENTQRY: 1
    #exemacro exetrg clearitem; entqry; exetrg showitem;
    *
    If step NOT successful:
    Head error KEY-ENTQRY
    Error-stop;

TRIGGER KEY-NXTREC: 1
    #exemacro nxtrec; exetrg showitem;
    *
    If step NOT successful:
    Head error KEY-NXTREC
    Error-stop;

TRIGGER KEY-NXTSET: 1
    #exemacro nxtset; exetrg showitem;
    *
    If step NOT successful:
    Head error KEY-NXTSET
    Error-stop;
```

```
TRIGGER KEY-PRVREC: 1
    #exemacro prvrec; exetrg showitem;
    *
    If step NOT successful:
    Head error KEY-PRVREC
    Error-stop;

TRIGGER KEY-UP: 1
    #exemacro up; exetrg showitem;
    *
    If step NOT successful:
    Head error KEY-UP
    Error-stop;

TRIGGER KEY-DOWN: 1
    #exemacro down; exetrg showitem;
    *
    If step NOT successful:
    Head error KEY-DOWN
    Error-stop;
```

In this application if deleting an order you want all records connected with this order to be deleted. In other cases you would like to check for consistency before deleting. For this purpose you would have to make changes in the KEY-DELREC trigger listed above.

```
TRIGGER PRE-DELETE: 1
    delete from orderline
    where id = :head.id
    *
    If step NOT successful: (No Message)
    Continue;
```

The POST-DELETE trigger has to be built in order to serve your applications needs. It has to be specified which records are to be deleted to ensure correct referencing.

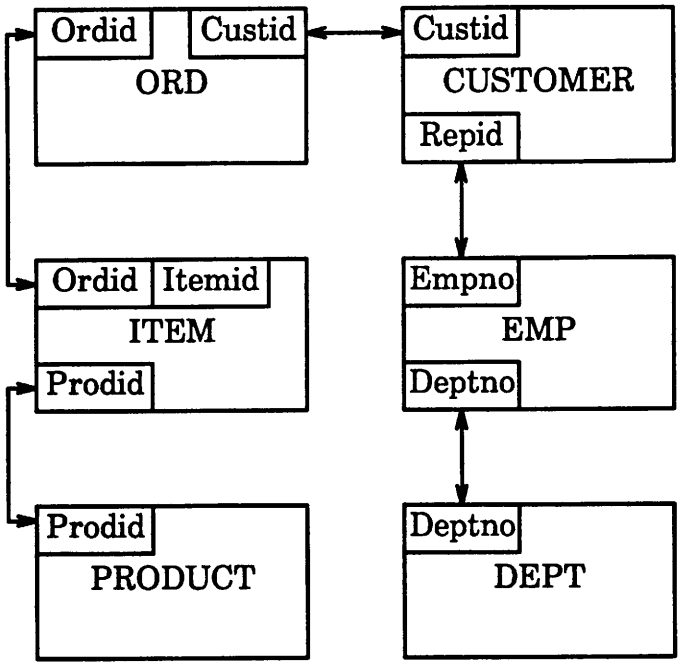
An application build as proposed in this paper has the following disadvantage. If an error occurs in one of the commit triggers PRE- INSERT, PRE-UPDATE, PRE-DELETE, POST-INSERT, POST-UPDATE, POST-DELETE or in the COMMIT itself an error message shows and a rollback is performed. In this situation the operator may see information in the heading and items not corresponding to each other. However regardless of the inconsistency shown the database is consistent.

4.2 Smart Query Coordination

The generel concept of query coordination is explained in the hint about *Referencing between blocks*.

This hint shows you how to program one set of application level triggers that allows you to coordinate any number of blocks with the simple addition of a usernamed trigger.

Let's assume we are working on the set of tables listed below :



The *Primary Key* is shown in the upper left corner of each table. The different *Foreign Keys* are connected to the other tables using arrows.

The heart of this query coordination is a single trigger :

Application-level Triggers

TRIGGER koor: 1

```
#exemacro nofail
        goblk &global.d_blk;
        case global.m_key is
        when '' then clrblk;
        when others then exeqry;
        end case;
        exetrg &system.cursor_block;
*
If step NOT successful: (No Message)
Error-stop;
```

This trigger goes to the block named in the global variabel **global.d_blk** (d_blk ~ details block). Depending on whether the variable **global.m_key** (m_key ~ contens of the foreign key in the master block) evaluates to empty of something else the details block is cleared or an execute query is performed. In order to make this query work the foreign key of the master block must be copied into the appropriate field(s) in the details block, using the *copy field value from* option in the Forms Validation Window - e.g :

Field: CUSTID in the ORD block Number: Displayed 8
Page: 1 Line: 22 Col: 60
Copied from: CUST.CUSTID
Help Message: Enter value for : CUSTID

The *koor* trigger terminates by calling a usernamed trigger bearing the name of the current block - the block where the cursor is placed which must be the *details* block.

For each block in the form an application level trigger must be created. This usernamed trigger must have the same name as the block. The example below shows the koordination triggers for the *customer* table, the block for this table is called *cust*, and the *ord* and *item* blocks :

Application-level Triggers

TRIGGER cust: 1

```
#exemacro
        copy cust.custid into global.m_key;
        copy 'ord' into global.d_blk;
        exetrg koor;
        goblk cust;
*
If step NOT successful: (No Message)
Error-stop;
```

TRIGGER ord: 1

```
#exemacro
        copy ord.ordid into global.m_key;
        copy 'item' into global.d_blk;
        exetrg koor;
        goblk ord;
*
If step NOT successful: (No Message)
Error-stop;
```

TRIGGER item: 1

```
#exemacro null;
*
If step NOT successful: (No Message)
Error-stop;
```

The KEY-ENTQRY trigger should look like this :

Application-level Triggers

TRIGGER KEY-ENTQRY: 1

```
#exemacro clrblk;
        exetrg &system.cursor_block;
        entqry;
        exetrg &system.cursor_block;
*
If step NOT successful: (No Message)
Error-stop;
```

This trigger will call the usernamed trigger that corresponds to the block in which the cursor is currently positioned. If the cursor is placed in the *cust* the usernamed trigger *cust* will be called.

This trigger will :

1. clear the *cust* block
2. copy the contens of the foreign key into the global variable globel.m_key
3. copy the name of the *details* block into global.d_key.
4. call the *koor* trigger

The first call to the *koor* trigger will clear the *ord* block since the global.m_key variable is empty.

The *koor* trigger calls the usernamed trigger *ord* just before it exits, which in turn will call the *koor* trigger again, this time to clear the *item* block. The *koor* trigger will terminate by calling the username trigger called *item*. This trigger only contains a dummy *null* operation, so the whole sequence of calls exits and the *cust* trigger ends by returning the cursor to the *cust* block.

The *KEY-ENTQRY* trigger reassumes and the *entqry* macro is performed. Now the trigger is suspended until the user presses [Execute Query].

The trigger continues by calling the *cust* trigger once again this time to query all the block, if the foreign key of the *cust* block is not empty.

If the foreign key in the master block is more than one field, the most important field, that is the field that must exist in order to do the join, could be copied into the global.m_key variable.

You could also choose to copy more than one field into the global.m_key variable :

```
copy cust.field1 ¦¦ cust.field2 into global.m_key
```

To program the coordination of the *cust* block and the *emp* and *dept* block, you copy the neccessary keys into the blocks, as described above. You modify the the *cust* trigger to

include the *emp* block and you create two new usernamed triggers, *emp* and *dept*, to coordinate the *emp* and *dept* block :

Application-level Triggers

```
TRIGGER cust: 1
_______ _______
      #exemacro
            copy cust.custid into global.m_key;
            copy 'ord' into global.d_blk;
            exetrg koor;
            copy cust.repid into global.m_key;
            copy 'emp' into global.d_blk;
            exetrg koor;
            goblk cust;
      *
      If step NOT successful: (No Message)
      Error-stop;

TRIGGER emp: 1
_______ _______
      #exemacro
            copy emp.deptno into global.m_key;
            copy 'dept' into global.d_blk;
            exetrg koor;
            goblk emp;
      *
      If step NOT successful: (No Message)
      Error-stop;

TRIGGER dept: 1
_______ _______
      #exemacro null;
      *
      If step NOT successful: (No Message)
      Error-stop;
```

To complete the coordination of the blocks, you need a number of additional application level key triggers :

Application-level Triggers

TRIGGER KEY-CLRBLK: 1

```
#exemacro clrblk; exetrg &system.cursor_block;
*
If step NOT successful: (No Message)
Error-stop;
```

TRIGGER KEY-CLRREC: 1

```
#exemacro clrrec; exetrg &system.cursor_block;
*
If step NOT successful: (No Message)
Error-stop;
```

TRIGGER KEY-CREREC: 1

```
#exemacro crerec; exetrg &system.cursor_block;
*
If step NOT successful: (No Message)
Error-stop;
```

TRIGGER KEY-DELREC: 1

```
#exemacro delrec; exetrg &system.current_block;
*
If step NOT successful: (No Message)
Error-stop;
```

TRIGGER KEY-DOWN: 1

```
#exemacro down;exetrg &system.cursor_block;
*
If step NOT successful: (No Message)
Error-stop;
```

TRIGGER KEY-EXEQRY: 1

```
#exemacro exeqry; exetrg &system.cursor_block;
*
If step NOT successful: (No Message)
Error-stop;
```

TRIGGER KEY-NXTREC: 1

```
#exemacro nxtrec; exetrg &system.cursor_block;
*
If step NOT successful: (No Message)
Error-stop;
```

TRIGGER KEY-NXTSET: 1

```
#exemacro nxtset; exetrg &system.cursor_block;
*
If step NOT successful: (No Message)
Error-stop;

TRIGGER KEY-PRVREC: 1

#exemacro prvrec;exetrg &system.cursor_block;
*
If step NOT successful: (No Message)
Error-stop;

TRIGGER KEY-UP: 1

#exemacro up;exetrg &system.cursor_block;
*
If step NOT successful: (No Message)
Error-stop;
```

Query in the detail blocks will be restricted by the foreign keys copied down from the master blocks and this is what you would normally prefer to avoid inconsistency between the blocks on the screen. However, if the master block is empty the user could go into the details block and perform a search without any arguments. To avoid this you could use a *WHERE* clause in the details block - e.g. for the *ord* block :

```
1 record Block: ord/ord
Table: ORD
Default: WHERE NVL(:cust.custid,-1) = ord.custid
```

Here the '-1' could be any value as long as it is not part of the set of values stored in this column.

4.3 Cursor Movement Between Blocks and Records.

If the application has a number of blocks you would normally try to hide this block concept from the end-user. This involves the *Query Coordination* described above but also that the cursor moves in a natural way, from one field to the next regardless of blocks and records.

We would like the cursor to move from the last field in a block to the first field in the next block when the user presses KEY-NXTFLD. This is done using the following KEY-trigger in the last field of the block :

```
KEY-NXTFLD
   Description: NONE    Hide: NO

   ---( Step 1 )---  Label:
   *Abort trigger when step fails
    Reverse return code
    Return success on abort      Success Label:
    Separate cursor data area    Failure Label:
   Failure Message:
   Trigger Step Text:
   #exemacro nxtblk;
```

The same type of trigger should be used in the last field of a Multi Record Blockain order to go to the next record, only this time the exemacro command should state :

#exemacro nxtrec;

When positioned in the first field in a Multi Record Block and you press KEY-PRVFLD you would like the cursor to either go to the last field in the privious record, or if positioned in the first record of the block, to go to the last field or the previous block :

```
KEY-PRVFLD
   Description: NONE    Hide: NO

   ---( Step 1 )---  Label:
   *Abort trigger when step fails
    Reverse return code
    Return success on abort      Success Label: ok
    Separate cursor data area    Failure Label: fail
   Failure Message:
   Trigger Step Text:

   #exemacro prvrec;

   ---( Step 2 )---  Label: fail
   *Abort trigger when step fails
    Reverse return code
    Return success on abort      Success Label:
    Separate cursor data area    Failure Label:
   Failure Message:
   Trigger Step Text:

   #exemacro prvblk;

   ---( Step 3 )---  Label: ok
   *Abort trigger when step fails
    Reverse return code
    Return success on abort      Success Label:
    Separate cursor data area    Failure Label:
   Failure Message:
   Trigger Step Text:

   #exemacro prvfld;
```

4.4 Generating Sequence Numbers.

Unique Keys can be generated using a *Sequence* when running under Oracle version 6. The

numbers are generated just before the record is inserted and the user should not be allowed to change this number. The procedure is as follows :

First create a *SQL Sequence* in SQL*Plus :

```
CREATE SEQUENCE seqno
START WITH 1
CACHE 5
```

Refer to the *SQL Language Reference Manual* of Oracle 6 for more information on Sequences.

Now, create a PRE-INSERT trigger in the Header Block of your application :

```
TRIGGER PRE-INSERT: 1
    SELECT seqno.nextval
    INTO header.id
    FROM dual
    *
    If step NOT successful:
    ERROR in PRE-INSERT
```

4.5 Using Trace in Forms 2.3.

To enable Oracle version 6 Trace in SQL*Forms 2.3 you need a few triggers. This hint shows you how to do this.

First you assign the startup of the trace function to a specific key. Here the KEY_DUBFLD is used so that the trace fundtion can be started manually, but you could use a KEY-STARTUP trigger instead :

```
KEY-DUPFLD
   Description: NONE    Hide: NO

   ---( Step 1 )---  Label:
   *Abort trigger when step fails
    Reverse return code
    Return success on abort      Success Label:
    Separate cursor data area    Failure Label:
   Failure Message:
   Trigger Step Text:

   #exemacro clrfrm;
   copy '0' into dept.deptno;
   commit; delrec; commit;
   message 'Trace Enabled..';
```

This key trigger inserts a dummy value in the *Primary Key Field* of the DEPT table, issues a commit and later deletes the the dummy record. All this is done to activate a the following PRE-INSERT trigger step :

```
PRE-INSERT
   Description: NONE    Hide: NO

   ---( Step 1 )---  Label:
   *Abort trigger when step fails
    Reverse return code
    Return success on abort      Success Label:
    Separate cursor data area    Failure Label:
   Failure Message:
   Trigger Step Text:

   #exemacro case dept.deptno is
   when '0' then exetrg trace;
     when others then null;
   end case;
```

This trigger states that if the commit was issued by the dummy insert operation then call the user named trigger *trace* :

trace
Description: NONE Hide: NO

---(Step 1)--- Label:
*Abort trigger when step fails
Reverse return code
Return success on abort Success Label:
Separate cursor data area Failure Label:
Failure Message:
Trigger Step Text:

alter session set events '10035 trace name context forever'

---(Step 2)--- Label:
*Abort trigger when step fails
Reverse return code
Return success on abort Success Label:
Separate cursor data area Failure Label:
Failure Message:
Trigger Step Text:

alter session set events '10046 trace name context forever'

Note: In version 6.0.27 *alter session set sql_trace true* may be used instead.

Remember to remove these triggers from the application before making the production version of the application.

4.6 Porting a SQL*Forms application

A number of minor details must be considered when a SQL*Form application is moved from one system to another. Especially if the underlying hardware in the two systems comes from different vendors. Here let us assume that the application *appl* should be moved from a Supermax to a system xx.

The application transport format is the *appl.inp* file. But because the application may include some graphical characters in the screen map layout, crt information is needed as well.

General speaking the same crt definition should be used porting the application on the two systems. The crt definition file *VTXXX.crt*, is f.x. available on most ports.

The following steps may be observed:

- Generate .inp file: *iac -c VTXXX appl appl user/passwd*
- Move the appl.inp file over to system xx.
- Maybe some character conversion is needed.
- Insert the application in the database again: *iac -c VTXXX -i appl appl user/passwd*
- Here you may generate your application again.

4.7 Various

- It is recommended to use capital letters for application names in SQL*Forms, because the *LIST* option for some reason converts all names to upper case.
- Usage of the *commit* macro is not recommended as it does not present a returncode, whether the commit has failed or succeded. It is better to leave the committing for the user to decide. When using the macro's *exit, clrfrm, clrblk, entqry,* and *exeqry* the user is asked whether he wants to commit, and using *call* the user has to commit. Using *clrblk, entqry* and *exeqry* the user is asked to commit only if changes has been made in the present block.

4.8 Optimizing Forms

Before reading this paper, you should study the **tuning and optimizing** article and get acquainted with the normal optimizing technics.

A general point is that you should analyse if use of the **dual** table is realy neaded. As we shall see a **case** statement may often be used instead. The main reason for avoiding too many selects from the **dual** table, is that:

- More cursers are in use.
- Errors occure if the dual table has more than one row.
- Responce time increases if the application is remotely logged on to a host database.

By the way, if you are using **sysdate** to retrive actual year number or day number, please consider to get this information by an **KEY-STARTUP** trigger to an SQL*Forms variable. Hereby a number of SQL statements can be redused providing better performance and a decrising space requirement.

4.9 Optimizing SQL*Forms

When writing and/or optimizing triggers in SQL*Forms you should watch out for certain constructions that are often used but unnecessary expensive in space and performance.

In Forms applications you will se statements like :

```
SELECT :field1, 'HALLO'
INTO field3, field4
FROM dual     /* Dual is the system dummy table */
```

This single cursor (SQL statement) will occupy at least 4 kb of memory and does nothing that you could not do better and faster with the **copy** command. SQL*Forms commands, like the copy command, are internal to SQL*Forms, so no SQL statements need to be parsed and no space has to be allocated on the Oracle MCU :

```
Step 1.  #COPY field1 field3
Step 2.  #COPY 'HALLO' field4
```

The point made here is that fewer cursors normally makes your application run faster and takes up less space.

4.9.1 The use of CASE

In all KEY triggers and all TYPES triggers in display fields (e.g. POST-CHANGE) and COMMIT triggers you could transform SQL statements into **case** statements.

Example :

```
SELECT 'x' FROM dual        /* never use the * operator */
WHERE :somefield IS NOT NULL

error message : somefield must contain a value
/
SELECT 'x' FROM dual
WHERE :somefield IN (val1,val2,val3,val4)

error message : somefield must be either val1 val2 val3 or val4
```

These two statements produces an error if they fail and writes out an error message in the status line.

This kind of 'SELECT 'x' FROM dual' statements should be changed into CASE statements :

```
#EXEMACRO CASE somefield IS
        WHEN '' THEN ENDTRIG FAIL MESSAGE 'Somefield must contain a value';
        WHEN OTHERS THEN NULL;
        END CASE;

#EXEMACRO CASE somefield IS
        WHEN val1,val2, val3, val4 THEN null;
        WHEN OTHERS THEN ENDTRIG FAIL;
        END CASE;

error message : somefield must be either val1 val2 val3 or val4
```

In TYPE triggers you can use all SQL*Forms macro's that does not affect the movement of the cursor.

4.9.2 How to disable function keys.

Function keys should not be disabled with a 'SELECT 'x' FROM dual' and reverse return code, but rather with :

```
#EXEMACRO NULL;
Reverse return code

error message : The key is disabled !
```

Or you could use :

```
#EXEMACRO NOOP;
```

The first one allows you to control the error message, the other one simply display an 'Unknown function' message and could be used together with the KEY-OTHERS key to disable all keys that are not redefined elsewhere.

4.9.3 How to parameterize SQL statements.

When a number of different checks is done against the same table, you could always consider the possibility of joining a number of trigger steps into a single trigger - that is if you can find a more general error message.

The join of a number of trigger steps could also be done using a parameterized WHERE clause :

```
User-named trigger 'validate'

SELECT 'x' FROM table
WHERE (:test = 1 AND condition .....)
OR    (:test = 2 AND condition .....)
OR    (:test = 3 AND condition .....)
OR    (:test = 4 AND condition .....)
OR    (:test = 5 AND condition .....)
```

This WHERE clause will do different validations depending on the value put into the :test field. If this SELECT was placed in a user-named trigger, it could be executed explicitly with an EXETRG whenever you need to check something :

```
Step 1. #COPY 3 test
Step 2. #EXEMACRO EXETRG validate;
```

The only drawback of this construction is that it should not be used in TYPES triggers in database fields (fields that are members of the underlying table of the block). Because the #EXEMACRO will execute even during a query, this might prove to be expensive.

4.9.4 You should rewrite slow SQL statements.

If a specific SQL statement involving a number of tables and conditions is slow, you should always try to rethink the statement.

Example :

Find the employees who have the same superior as JONES, a later hiredate and a smaller salary :

solution 1 This first solution is properly quite slow due to the 3 subqueries and should not be used :

```
select a.ename,a.sal,a.hiredate from emp a
where a.mgr =(select mgr from emp where ename='JONES')
  and a.hiredate >(select hiredate from emp where ename='JONES')
  and a.sal<(select sal from emp where ename ='JONES')
```

solution 2 This solution uses the Oracle SIGN operator, reducing the 3 subqueries to a single one :

```
select a.ename,a.sal,a.hiredate,a.mgr from emp a
where (0,1,1) = (select sign(mgr-a.mgr),
        sign(a.hiredate-hiredate),
        sign(sal-a.sal)
         from emp2 where ename='JONES')
```

solution 3 This last solution is a simple join.

```
select a.ename,a.sal,a.hiredate from emp a,emp b
where a.mgr =b.mgr
  and a.hiredate >b.hiredate
  and a.sal<b.sal
  and b.ename ='JONES'
```

This last solution, a simple join, will normally run faster than the other solutions, but you should always time and trace the different statements in sqlplus, to test the performance.

Do not store intermediate results in SQL*Forms fields where it is not necessary, e.g. :

```
Select val1 * 1.2 into field1
from .....
where .....
/
select (:field1 + :field2) / 2
into field3
from dual
where ...
```

Use a single select statement instead.

4.9.5 Search arguments in RUNFORM.

You should be aware, that using the % sign as the first character in search arguments, when querying big tables, corrupts the use of indexes on that column. Unless another and more specific search criterion is given, this might result in a devastating long search, especially if the rows are to be sorted as well.

You could redefine the KEY-EXEQRY in your application to check the search criteria, before actually executing the query, or you could warn against the use of '%' in first position (or no search arguments at all) in your user manual.

4.10 A simple Date-Conversion

The Oracle display format for dates in SQL*Forms is not an acceptable representation in Europe and elsewhere outside the USA. Here is an example showing how to use the **edate** data type in a display field and still store the date in a column in the database defined as **date**.

The **edate** field type has the format 'DD/MM/YY' and could be used by the operator when updating and inserting on the screen, the point here being that dates written in an edate field are checked, so that illegal dates will provoke a Runform error message.

To handle this situation, two fields must be created:

1. A **date** field : Dates should be stored in a database column of the type **date**. This field (dato) is placed as a not-displayed field on the screen.
2. An **edate** field : A displayed non-database field (ddato) is used for screen input.

Now you provide two POST-CHANGE triggers:

1. The first trigger should convert the Oracle date format found in the date field (dato) into edate format and place this in the edate field (ddato). The trigger attribute **return succes when trigger fails** should be set along with the normal **abort trigger when step fails**.
2. The second trigger is placed in the edate field and should convert user input/updates into Oracle date format and place the result in the date field (dato).

```
Field: DATO                          Date: Not Displayed 9
    Page: 1 Line: 1 Col: 31
    Non-enterable

     TRIGGER POST-CHANGE: 1
          select to_char(to_date(:dato,'DD-MON-YY'),'J')
          into ddato
          from dual
          where :ddato is null
          •
          If step NOT successful: (No Message)
          Check-stop;


Field: ddato                         Edate: Displayed 9
    Page: 1 Line: 1 Col: 13                 Query 99
    Help Message: None

     TRIGGER POST-CHANGE: 1
          select to_date(:ddato,'J')
          into dato
          from dual
          •
          If step NOT successful: (No Message)
          Error-stop;
```

Please note that queries with arguments (like >= ...) in the **edate** field cannot be used, and that the edate field on the screen cannot be referenced as a character string - it has to be converted into a Julian date first.

4.11 Block with no Underlying Table

A block in SQL*Forms with no underlying table is a block which contains no database fields :

- All the fields in such a block have the field attribute *Database Field* switched off. They may or may not be *Displayed*.
- The *Table Name* field in the **Define Block** window has been cleared to remove any reference to an underlying table for that block.

Two types of blocks with no underlying table are often used in SQL*Forms.

Control Block The *control block* which is used to store hidden varables containing general information. The user is normally prevented from entering the *control block* and the block is often hidden.

Display Block The *display block* is used to display information on the screen. The user may or may not be allowed into this block.

4.11.1 Creating A Control Block

If you have only a few variables within your form, these can be put in hidden fields in some single record block of the application. They can be placed on the page of that block or put in the *Zero Page* (page 0).

However - the application might not contain a single record block or you might prefer to put all variables in a special control block. This block should be hidden from the user and should not contain any enterable fields. This poses a small problem since SQL*Forms demands that each block must have at least one enterable field :

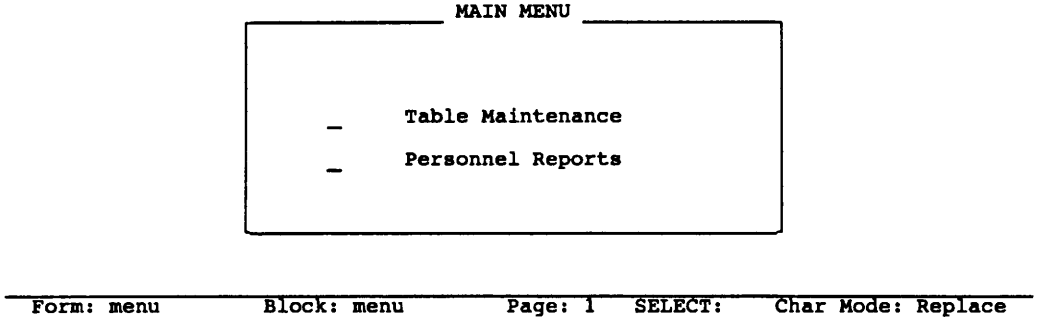
- Create a block with no underlying table as described above. Put this block on a separate page.
- Specify at least one field in the block with the attributes *Displayed* and *Input Allowed* on.
- Re-define KEY-NXTBLK in the preceding block and KEY-PRVBLK in the following block to skip over the *Control Block*. In this way we allow the user to enter a field in the *Control Block*, as SQL*Forms specifies that we must, but we prevent him from ever entering the block. Since the *Control Block* was put on a page of it's own, the whole block is now hidden.

4.11.2 Creating a Display only Block

A block containing non database fields for display only is created in the same way as a *Control Block*, except that the block is placed somewhere in the form, where it can be seen.

4.11.3 Creating a Menu System

- A Number of Menu Blocks with no underlying tables are created. Each block is placed on a separate page :

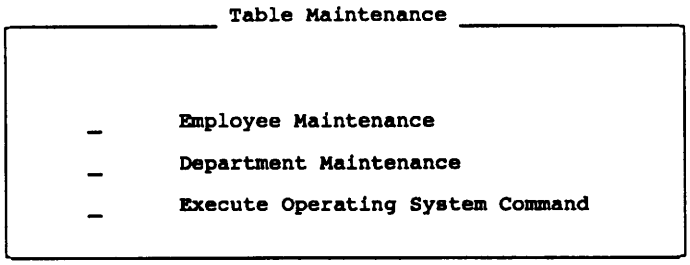


This is how the main menu looks in the screen painter. It has two 1 character fields, **m1** and **m2**, with the attributes *Displayed* and *input allowed* set, followed by the text of each entry.

The Commit key has been redefined on **block level**, so that it can be used to select items and the menu :

```
TRIGGER KEY-COMMIT: 1

#exemacro case system.current_field is
          when 'M1' then goblk submenu1;
          when 'M2' then goblk submenu2;
          end case;
*
If step NOT successful: (No Message)
Error-stop;
```

The first submenu is found on the second page :




```
Form: menu      Block: submenu1    Page: 2   SELECT:    Char Mode: Replace
```

The Commit trigger for this block is :

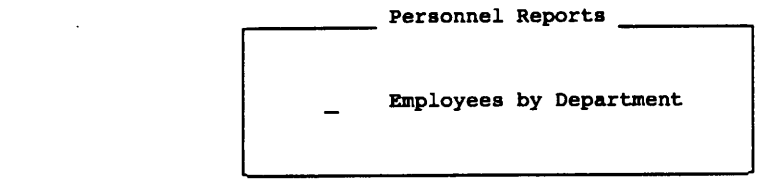
```
TRIGGER KEY-COMMIT: 1

#exemacro case SYSTEM.CURRENT_FIELD is
          when 'S1_M1' then call emp;
          when 'S1_M2' then call dept;
          when 'S1_M3' then gofld cmdline;
          end case;
*
If step NOT successful: (No Message)
Error-stop;
```

The two first items on this menu uses the *Call* command to start up SQL*Forms applications. In this way the user only logs on once - when he starts the menu - and this logon is reused for the rest of that session.

The third item demontrates a dynamic use of the *#host* command. The user is allowed to write any Unix commend in the *cmdline* field found on a separate page. In this field the KEY-NXTFLD has been redefined :

```
TRIGGER KEY-NXTFLD: 1

#exemacro case cmdline is
          when '' then nxtfld;
          when others then exetrg startshell;
          end case;
*
If step NOT successful: (No Message)
Error-stop;


TRIGGER startshell: 1

#host cmdline
*
If step NOT successful: (No Message)
Error-stop;

          startshell: 2

#exemacro nxtfld;
*
If step NOT successful: (No Message)
Error-stop;
```

A more static use of the *#host* is shown in the second submenu :




```
Form: menu      Block: submenu2    Page: 4    SELECT:    Char Mode: Replace
```

The trigger for this block is :

```
TRIGGER KEY-COMMIT: 1

#host 'sqlplus -s @report'
*
If step NOT successful: (No Message)
Error-stop;
```

To disable all the function keys in SQL*Forms, that you do not need in an menu application, you could write a few triggers on application level :

```
TRIGGER KEY-EXIT: 1

_______ ____________
        #exemacro case SYSTEM.CURRENT_BLOCK is
                  when 'MENU' then exit;
                  when others then goblk menu;
                  end case;
        *
        If step NOT successful: (No Message)
        Error-stop;

TRIGGER KEY-NXTREC: 1

_______ _____________
        #exemacro nxtfld;
        *
        If step NOT successful: (No Message)
        Error-stop;

TRIGGER KEY-OTHERS: 1

_______ _____________
        #exemacro NULL;
        *
        If step NOT successful: (No Message)
        Error-stop;

TRIGGER KEY-PRVREC: 1

_______ _____________
        #exemacro prvfld;
        *
        If step NOT successful: (No Message)
        Error-stop;
```

The *Automatic Help* attribute together with a *Help text* could be used to display additional text for each menu item in the status line.

4.12 Access Control in SQL*Forms

Some Oracle users may be allowed to use a specific SQL*Forms application while others may not, and some users may be allowed access to certain blocks of a form but not into others.

Security in Oracle is managed via the Oracle USER names.

Checking User names in a SQL*Forms application is relatively simple. It can be done using a **KEY-STARTUP** trigger on *form level* when controlling access to the form, or on *block level* when controlling access to individual blocks :

```
TRIGGER KEY-STARTUP: 1

        ------- ----------------
                select 'x' from system.dual
                where USER in ('ORA','CFJ')
                *
                If step NOT successful:
                You are NOT allowed to use this block....
                Go to step2    ELSE go to step3

    step2 KEY-STARTUP: 2

        ------ ----------------
                #exemacro exit;
                *
                If step NOT successful: (No Message)
                Error-stop;

    step3 KEY-STARTUP: 3

        ------ ----------------
                #exemacro null;
                *
                If step NOT successful: (No Message)
                Error-stop;
```

Step 1 in this trigger writes out an error message if it fails and then exits the form. The check of the *USER* names in this example is static, but could just as well be done using the *EXISTS* argument together with a subquery selecting from a control table.

The only dis-advantage of this approach is that the user sees the block before being rejected. This can be avoided if you create a special check block on a separate page and put the *KEY-STARTUP* trigger on this block :

```
+-------------------------------+
|  _ Checking your access rights |
+-------------------------------+

Control Block: check/check
Starts on Page: 1

                        Block-level Triggers

  TRIGGER KEY-STARTUP: 1

        ------- ----------------
                select 'x' from system.dual
                where USER in ('ORA','CFJ')
                *
                If step NOT successful:
                You are NOT allowed to use this block....
                Go to step2    ELSE go to step3

      step2 KEY-STARTUP: 2

          ------ ----------------
                #exemacro exit_or_whatever;
                *
                If step NOT successful: (No Message)
                Error-stop;

      step3 KEY-STARTUP: 3

          ------ ----------------
                #exemacro goblk the_block_in_question;
                *
                If step NOT successful: (No Message)
                Error-stop;


Field: check                                  Character: Displayed 1
       Page: 1 Line: 8 Col: 23                           Case unchanged
      Non-base-table Non-updatable Non-queryable
      Help Message: None
```

How to create a control block, that is a *block with no underlying table*, is explained in the article on that subject.

4.13 Oracle Corporate Support Hints

The following hints originates from the Oracle Corporate Support *Customer Support Newsletter* Oct/Nov 1989.

4.13.1 Fixing cursor Usage Problems in SQL*Forms

```
Oracle corporate support
Bulletin
Bulletin category : SQL*FORMS
Bulletin Number : 38366.060
Bulletin Topic : CURSOR USAGE
```

This document explains how SQL*Forms uses cursors, and how to diagnose and correct problems with cursor usage. This information pertains to Oracle version s.l and SQL*Forms 2.O, 2.2, or 2.3.

4.13.2 HOW SQL*FORMS USES CURSORS

What is a cursor?

A cursor is an area of memory used by Oracle to execute a SQL statement. A user process " opens" a cursor for each SQL statement it needs to execute. The SQL statement must be parsed when it is first submitted to Oracle. This parsing operation is relatively slow. Since the statement is only parsed the first time it is executed, subsequent executions are faster. When the user process is finished with the SQL statement, it can " close" the cursor, which releases the cursor's memory. To re-execute the SQL statement, the cursor must be opened and re-parsed. The memory used by cursors comes from the user process's memory.

How does SQL*Forms use cursors?

By default, SQL*Forms opens one cursor for each separate SQL statement it executes, and closes these cursors when the user exits the form. The designer can override this default operation, as discussed below. How many cursors will a form use? Running a form requires a minimum of four cursors. A simple form may use l5 to 30, and a complex form may use hundreds. This is how cursors are opened:

- General rule: Each SQL statement gets one cursor.
- cursors are opened when the SQL statement is executed. For example, SQL*Forms opens one cursor when a block is queried, or when a trigger step containing a SQL statement fires. Cursors are opened only for SQL statements that are executed; that is, if a trigger step is not executed, a cursor is not opened for it.
- A SQL statement gets only one cursor, no matter how many times it is executed. AS above, the statement will execute faster after the first time, because it does not have to be parsed again.
- Every SQL statement in the form is independent, even if it is identical to another SQL statement. So, if five trigger steps all do SELECT 'x, FROM DUAL, they use five separate cursors. operations that don't use SQL statements, such as #COPY, don't use cursors.
- Oracle will sometimes open additional cursors, if needed, to evaluate a cursor requested by a process. These are called "implicit" cursors, as opposed to the "explicit" cursors a process requests. Implicit cursors are recycled and closed by Oracle, and SQL*Forms designers usually do not need to be concerned with them.

This is how specific SQL statements use cursors:

```
Operation                  No. Of cursors/SQL statement
-------------------------------------------------------------------
start form:                      4      (overhead)
Query block, first time:         1      (SELECT)
First time a record is
marked for update or
deletion, in a block:            1      (SELECT FOR UPDATE)
First commit, in a block:        2-4    (LOCK TABLE IN SHARE
                                        UPDATE MODE,
                                        INSERT/UPDATE/DELETE)
Execute 1 SQL statement in
a trigger step:                  1
Repeat any of above:             0
```

How does SQL*Forms use and close cursors?

After being opened, this is how cursors are used:

- By default, a cursor is kept open and ready for re-execution until the form is exited.
- However, the form designer may specify that some or all cursors be "de-optimized". In that case, the cursor's memory may be reused by another cursor. Before the first cursor can be executed again, it must be given its own memory and reparsed.

This is how cursors are closed:

- All of a form's cursors are closed when the form is exited. If form A CALLs form B, form A's cursors stay open while B is running. Upon exiting form B, B's cursors are closed.
- NEWFRM closes the old form's cursors.

How can I monitor cursor usage?

There are two ways to monitor cursor usage in SQL*Forms:

- Forms statistics: the Forms statistics feature displays the number of cursors open at end of running the form. Turn statistics on by selecting it from the SQLFORMS Design Runtime options window, or by invoking RUNFORM with the "-s" switch.
- Under Oracle 5 ODS displays a user process's cursor use in real-time. This is good for monitoring cursor usage in detail. Invoke ODS, set cycle to 1 second, and invokd the User screen for the specific process you want to watch.

The "User cursors" value will increment as SQL statements are executed. (The "Oracle Cursors" value will usually stay near 0; this value tracks the "implicit" cursors in use.)

*4.13.3 DIAGNOSING CURSOR PROBLEMS IN SQL*FORMS*

All SQL*Forms errors regarding cursors mean essentially the same thing; that SQL*Forms tried to open a cursor and was unable to do so. The solution is almost always a) get more resources to open more cursors, or b) redesign the form to use fewer cursors.

Here are some cursor-related error messages:

```
ORA-1000: max open cursors exceeded
ORA-1046: cannot acquire space to extend context area
ORA-105o: can't acquire space to open context area
ORA-1051: maximum context area extents exceeded
```

(The cursor error messages usually use the term "context area", which for most purposes may be thought of as synonymous with a "cursor".) In all cases, you should refer to the Error codes Manual or the DBA Guide for more Information regarding the error.

To troubleshoot and fix cursor problems, follow these steps:

1) Reproduce the error

cursor errors may seem sporadic, but they should be possible to reproduce by running the form and using all possible blocks and triggers. Watch the process in ODS (or version 6 MONITOR) if needed. (For errors 1046 and 1050, see below.)

You may want to create a keystroke file to save the keystrokes you use to reproduce the error. To do this, use RUNFORM with the "-e" (to create the file) and "-r" (to read from the file) options.

2) was the error ORA-1051?

You are unlikely to get this error in a form. This error indicates that the SQL statement is larger than the size Oracle was set up to handle.

Action: Usually, increase INIT.ORA parameter CONTEXT_INCR.

3) was the error ORA-1000?

This is the most common SQL*Forms cursor error. It means that the form requested more cursors than the database is configured to give a user process. This value is set by the INIT.ORA value OPEN CURSORS, which ranges between lo and 255.

Actions: Note the number of cursors opened by the form when the error occurred. If less than 255 cursors where opened, have your system administrator raise the OPEN_CURSORS parameter in INIT.ORA. If near 255, see "Tuning a Form" below.

4) was the error 1046 or 1050?

These errors indicate that your process could not get enough memory from the operating system to open the needed cursors.

Actions: If appropriate for your o/s, confirm that the process does not have anything unnecessary in memory. Request more memory for your process from your system administrator. see also the next section on "Tuning a Form" .

4.13.4 TUNING A FORM TO USE FEWER CURSORS

At this point, you have determined that there is no way to give the form the cursors it needs. The only solution is to reduce the number of cursors the form needs. Do each of these in order. You may stop when you have achieved the desired results.

A. Change triggers to use fewer cursors:

   These steps are easy, and will actually enable the form to run faster, in addition to reducing cursor use.

   - Use #COPY instead of SELECT...INTO.. , wherever possible.
   - "Modularize" your triggers. That is, wherever more than one trigger performs the same operation, create a usernamed trigger at the block or form level, and call that trigger from the other triggers.
   - Group SELECT...INTO statements. "SELECT a, b, c INTO aa, bb, cc FROM..." only uses a single cursor.

Note: It is sometimes recommended to "fully qualify" object names in SQL statements, for example "SELECT EMP.ENAME FROM SCOTT.EMP" as opposed to "SELECT ENAME FROM EMP" . In the past, this was sometimes necessary for doing some joins, due to a bug in the way cursors were handled. This bug was fixed in all versions later than Oracle version 5.1.8. Fully qualifying object names may speed processing, but is no longer need to conserve cursor usage.

B. Change the form to close cursors:

You can usually break up a large application into several smaller forms. This lets SQL*Forms close cursors for parts of the application not currently being used.

- Use CALL or CALLQRY to link the forms making up the application.
- Design the application as a "menu" . For example, if the application consists of four forms that can all CALL each other, a user may end up with cursors for all four open at once. But, if the four are only accessable from a fifth "menu" form, ony two forms will ever be active at the same time.
- Use NEWFRM instead of CALL whenever possible. Unlike CALL, NEWFRM releases the previous form's cursors.
- As a last resort, you can use #HOST instead of CALL. Since #HOST spawns an entirely new process, this form will have the maximum number of cursors avàilable. However, #HOST is slower than CALL, CALLQRY, or NEWFRM.

C. "De-optimize" the form to use fewer cursors

As above, SQL*Forms by default keeps all cursors open until the form is exited. However, the designer has the option to "de-optimize" the form, so that some or all cursors are reused by new SQL statements as needed, rather than opening new cursors. De-optimizing the form will usually be your last resort, since it will slow the form down

To de-optimize a form:

1. Note the current level of cursor usage.
2. Run the form in de-optimize mode by either a) deselecting the "optimize SQL Processing" option in the Designer's Runtime options window, or b) using RUNFORM with the "-o" switch.

   Now all trigger SQL statements will share a one single cursor, instead of each statement getting its own cursor. If the form's speed is still satisfactory, simply have users always run with the "-o" switch.
3. If the form is now unacceptably slow, try reserving cursors for certain trigger steps:
   - Pick the trigger steps that are executed most often.
   - For those trigger steps, select the "separate cursor Data Area" attribute in the Trigger step Attributes window.
   - Watch ODS (or version 6 MONITOR) as the form runs, and note the form's speed and number of cursors used.
   - Repeat for as many trigger steps as you need to speed the form up, while leaving a margin of safety in cursor use.

Note: SQL*Forms also allows you to de-optimize the INSERT, UPDATE, and DELETE statements (which SQL*Forms calls "Transaction Processing" operations). However, these operations use relatively few cursors. REFERENCES

- "SQL*Forms Designer's Reference" , v2.O (Oracle Part No. 3304-v2.O)
- "Oracle Database Administrator's Guide" , v5.1 (Oracle Part No. #3601-v5.1)
- " Error Nessages and codes Nanual" , v5.1 (Oracle Part No. #3605-v5.1)

*4.13.5 SQL*FORMS v2.O and v2.3 - The Handling of Text*

```
Oracle corporate support
Bulletin category : SQL*FORMS
Bulletin Number : 98526.455
Bulletin Topic . V2.O & 2.3 - HANDLING TEXT IN FORMS
```

Have you ever wondered how lines of text are kept in their original order in version 2 of SQL*Forms? It is often necessary to store and retrieve lines of text in the order in which they were originally entered. Furthermore, lines of text may be inserted between existing lines and these lines also need to be properly ordered.

The basic idea is to utilize sequence numbers to maintain the order. Suppose you have three lines of text and you want to insert a new line of text between lines 1 and 2. After inserting this new line, when you query it back up, you want to see it between lines l and 2. The trick is to generate a unique sequence number that will ensure that the line will remain where you inserted it. The number 1.5 is calculated when you insert a new line between say, lines 1 and 2. Now, the sequence numbers will be 1, 1.5 and 2. what if you decide to insert several lines between lines 1 and 2? The same algorithm is applied to generate the unique sequence numbers to keep the lines of text in order and intact.

The following algorithm is implemented on our On-Line Support (OLS) and can be used to serve a variety of application needs. Let's consider, for example, the Order Entry application that is used in the SQL*Forms 2.O Designer's Tutorial. The example shows a master/detail relationship between the order block and the line items block. we will demonstrate how to implement a similar master/detail relationship for the text block.

Each order has a unique orderid number in the order table. For each line item in the line item table, there is a unique order id plus the line item number forming the concatenated primary key. Likewise with the text block, there will be a order id and a sequence number that will form the concatenated primary key.

Assume the following table definition:

```
Table ORDER_TEXT

Name                 Not Null?    Type
-------------      ---------      -------
ORDER_ID             NOT NULL     NUMBER
SEQ_NO               NOT NULL     NUMBER
TEXT                              CHAR(80)
```

With the above definition of the ORDER_TEXT table, follow these steps to implement this algorithm.

1. Create a control block (no base table) with two non-db fields called HI and LO

2. Create a block based on your text table i.e. , ORDER_TEXT.
3. Define all of the fields in your ORDER_TEXT block to be database fields. It is your choice whether to have the ORDER_ID and the SEQ_NO fields be displayable. It is not necessary.
4. Define the ORDER_ID field to COPY its value from the master block via the VALIDATION window.
5. Define a PRE-INSERT trigger for the ORDER_TEXT block with the following steps:

```
STEP 1:
SELECT 'X'
FROM DUAL
WHERE :SEQ_NO IS NULL

STEP 1 ATTRIBUTES:
*Abort trigger when step fails.
*Return success when aborting trigger.

STEP 2:
SELECT NVL(MAX(SEQ_NO),0) + 1
INTO SEQ_NO
FROM ORDER_TEXT
WHERE :ORDER_TEXT.ORDER ID = ORDER_ID

STEP 2 ATTRIBUTES:
*Abort trigger when trigger step fails.
```

6. Define a KEY-CREREC trigger for the ORDER_TEXT block with the following steps:

```
STEP 1:
#COPY :SEQ_NO LO

STEP 2:
#EXEMACRO NXTREC;

STEP 3:
#COPY :SEQ_NO HI

STEP 4:
#EXEMACRO PRVREC; CREREC;

STEP 5:
SELECT (:LO + :HI) / 2
INTO :SEQ_NO
FROM DUAL
```

7. Define the default ORDER_BY on the ORDER_TEXT block to order on the SEQ_NO field.

LIMITATIONS: You are limited in the number of inserts you can do between lines. For example, you have lines 1 and 2 already in the database and you query them up. Now you want to insert new lines between them. On VMS, you are limited to 15 insertions before losing accuracy. On Unix, this number is 26. You may need to do some simple

test cases to determine what your operating system will allow.

NOTE: This bulletin does not apply to SQL*Forms v3.O. In that version, the designer can take advantage of the sequence number generator offered by the v6.O RDBMS.

4.13.6 *How to Use #HOST, CALL, CALLQRY, and NEWFRM in SQL*Forms*

```
Bulletin category : SQL*FORMS
Bulletin Number : 82710.039
Bulletin Topic : USING #HOST, CALL, CALLQRY, AND NEWFRM
```

This document will describe and give examples of how to issue operating system commands and to call other forms from within SQL*Forms applications. Included will be information on the #HOST command, and the CALL, CALLQRY and NEWFRM macros.

1) #HOST command.

```
Basic syntax: #HOST 'command to be executed by the operating
                    system'
Alternate syntax  : #HOST :block.field
Alternate syntax  : #HOST &global.reference

Examples:

a) To run a SQL script called REPORT.SQL from the SCOTT/TIGER
   account:

            #HOST 'SQLPLUS SCOTT/TIGER @REPORT'

b) To run a SQL script whose name is contained in the field
   :BLOCK1.REPORT_NAME:

   i. Create a non-database character field somewhere in the
      form, called, in this example, BLOcX2.conNAND LINE.
   ii. Have your trigger do the following:

STEP 1: SELECT 'SQLPLUS SCOTT/TIGER @' || :BLOCK1.REPORT_NAME
      INTO :BLOCK2.COMMAND_LINE
      FROM DUAL

STEP 2: #HOST :BLOCK2.COMMAND_LINE

c) To run a SQL script whose name is contained in :BLOCK1.REPORT_NAME
and which requires two parameters, contained in fields
:BLOCK1.PARAM1, and :BLOCK1.PARAM2

STEP 1: SELECT 'SQLPLUS SCOTT/TIGER @' || :BLOCK1.REPORT_NAME
      || ' ' || :BLOCK1.PARAM1 || ' ' || :BLOCK1.PARAM2
```

```
      INTO :BLOCK2.COMMAND_LINE
      FROM DUAL

STEP 2: #HOST :BLOCK2.COMMAND_LINE
```

d) To execute an operating system command passed into the form in a global variable called GLOBAL.OS coMMD:

```
      #HOST &GLOBAL.OS_COMMAND
```

2) CALL, CALLQRY and NEWFRM macros.

Basic syntax	: #EXEMACRO CALL formname;
Alternate syntax	: #EXEMACRO CALL &global.reference;
Alternate syntax	: #EXEMACRO CALL &block.fieldname;
Basic syntax	: #EXEMACRO CALLQRY formname;
Alternate syntax	: #EXEMACRO CALLQRY &global.reference;
Alternate syntax	: #EXEMACRO CALLQRY &block.fieldname;
Basic syntax	: #EXEMACRO NEWFRM formname;
Alternate syntax	: #EXEMACRO NEWFRM &global.reference;
Alternate syntax	: #EXEMACRO NEWFRM &block.fieldname;

Examples: The use of arguments with CALL, CALLQRY, and NEWFRM is identical. Therefore each example will be presented only once, with the understanding that CALLQRY or NEWFRM may be freely substituted for CALL.

a) To call a form called COOLFORM:

```
#EXEMACRO CALL COOLFORM;
```

b) To call a form whose name is contained in the field :BLOCK1.FORMNAME

```
#EXEMACRO CALL &BLOCK1.FORMNAME;
```

c) To call a form that is in another directory:

STEP 1: #COPY '/Full/Path/and/Formname' GLOBAL.ANY_VARIABLE;

STEP 2: #EXEMACRO CALL &GLOBAL.ANY_VARIABLE;

4.13.7 Passing Query Criteria Between Applications via Global variables

Bulletin category : SQL*FORMS
Bulletin Number : 82470.040
Bulletin Topic : PASSING QUERY CONDITIONS BETWEEN FORMS

SQL*Forms application designers often desire to pass values between two forms via global variables and use these values as search criteria. This may be accomplished by use of the pre-query trigger.

A pre-query trigger is fired when the operator presses (Execute Query or (count Query Hits, or when an EXEQRY macro is executed, but before SQL*Forms retrieves any records. Pre-query triggers have a unique aspect--values placed into database fields of the current block are used as search criteria for the query being executed. All other triggers either modify existing records or initialize new records when they place values into database fields.

To pass search criteria to another form, use the following method:

Suppose we have two forms, FormA and FormB, where Form calls formB.

1) On some trigger in FormA, the following steps would be needed:

step 1: #COPY <value> GLOBAL.variablename;

step 2: #EXEMACRO CALL FormB;

where <value> is a constant, field reference, or variable reference.

2) In FormB, a pre-query trigger would do the following:

#COPY GLOBAL.variablename :blockname.fieldname;

whenever a query is executed in the specified block, the value stored in the global variable would serve as a search criterion, in conjunction with any other search criteria that the operator may have specified.

In SQL*Forms v2.3, a default value can be specified for a global variable. A form, like FormB above, which is normally called by another form can then be run standalone as well. If the global variable is not initialized with some value, the #COPY in 2) above would result in an error. To prevent this, default a value into the global variable. The command is:

#EXEMACRO DEFAULT <value> INTO GLOBAL.variablename;

The global variable only takes on the specified value if the variable has not been initialized prior to the execution of the DEFAULT macro.

See Chapter 5 of the SQL*Forms Documentation Addendum v 2.3 for a description of the DEFAULT macro.

*4.13.8 Date and Time support in SQL*Forms Version 2*

Bulletin category : SQL*FORMS
Bulletin Number : 13057.010
Bulletin Topic : DATE AND TIME SUPPORT

The DATE datatype in the Oracle RDBNS includes century, year, month, day, hour, minute, and second. However, the date_field type in SQL*Forms supports only year, month, and day in the default format DD-MON-YY, . (see IV-A below for SQL*Forms 2.3 exception.) Additional field types (JDATE, EDATE, TIME) are supplied, but since they are internally represented as NUMBER, they cannot be used for database fields corresponding TO_DATE

columns.

Following is a detailed explanation of the triggers needed to manipulate date and time values in SQL*Forms. Block-mode designers should be aware that the validation for this implementation is done at the FIELD level.

Keep in mind that regardless of field type, ALL field values are character strings and should be handled as such. Specifically, the TO_NUMBER function may be required on NUMBER fields and the TO_DATE function on date_fields to achieve the desired results (see IV-B below for SQL*Forms 2.3 exception.)

I. How to implement a Jdate_field

Despite its displayed format of 'MM/DD/YY', JDATE is actually the Julian day (the number of days since December 31, 4713 BC).

A. The date_field that corresponds to the DATE column in the base table is Database, Nondisplayed. The Jdate_field is Nondatabase, Displayed.

B. PRE-QUERY trigger:

```
SELECT TO_DATE( :jdate_field, 'J')
INTO :date_field
FROM DUAL
```

C. POST-QUERY trigger:

```
SELECT TO_CHAR(TO_DATE( :date_field) , 'J' )
INTO :jdate_field
FROM DUAL
```

D. POST-FIELD trigger on the JDATE field:

```
SELECT TO_DATE( :jdate_field, 'J' )
INTO :date_field
FROM DUAL
WHERE NVL(TO_CHAR(TO_DATE(:date_field)),0) !=
NVL(TO_CHAR(TO_DATE( :jdate_field, 'J')),0)
```

NOTES:

1. Be sure to turn off " Abort trigger when step fails" .
2. POST-FIELD is used rather than POST-CHANGE because POST-CHANGE will not fire in the case where the field is changed to null.

E. KEY-EXIT trigger on the Jdate_field:

```
#EXEMACRO GOFLD jdate_field; EXIT;
```

NOTE: This trigger is needed to fire the POST-FIELD trigger (see D above) in the case where the JDATE is the only field updated, then [Exit] is hit while the cursor is still in the Jdate_field. This trigger is not necessary in SQL*Forms 2.3.

II. How to implement an Edate_field

EDATE is displayed as 'DD/MM/YY', but like JDATE, it is actually a number representing the Julian day.

A. The date_field that corresponds to the DATE column in the base table is Database, Nondisplayed. The EDATE field is Nondatabase, Displayed.

B. PRE-QUERY trigger:

```
SELECT TO_DATE( :edate_field, 'J' )
INTO :date_field
FROM DUAL
```

C. POST-QUERY trigger:

```
SELECT TO_CHAR(TO_DATE( :date_field) , 'J' )
INTO :edate_field
FROM DUAL
```

D. POST-FIELD trigger on the EDATE field:

```
SELECT TO_DATE( :edate_field, 'J' )
INTO :date_field
FROM DUAL
WHERE NVL(TO_CHAR(TO_DATE( :date_field) ) ,0) !=
NVL(TO_CHAR(TO_DATE( :edate_field, 'J' ) ) ,0)
```

NOTES:

1. Be sure to turn off " Abort trigger when step fails" .
2. POST-FIELD is used rather than POST-CHANGE because POST-CHANGE will not fire in the case where the field is changed to null.

E. KEY-EXIT trigger on the EDATE field:

```
#EXEMACRO GOFLD edate_field; EXIT;
```

NOTE: This trigger is needed to fire the POST-FIELD trigger (see D above) in the case where the EDATE is the only field updated, then (Exit, is hit while the cursor is still in the EDATE field. This trigger is not necessary in SQL*Forms 2.3.

III. How to implement a TIME field

TIME is displayed as 'HH:MI:SS' ; however, its value is actually the total number of seconds ('HH'*3600 + 'MI'*60 + 'SS').

A. The DATE field is Database, Displayed. The TIME field is Nondatabase, Displayed. Unlike JDATE and EDATE, the TIME cannot be displayed merely by reformatting the DATE field. Rather, it must be retrieved directly from the base table. For this implementation, another TIME field that is Nondatabase, Nondisplayed (hereafter referred to as :hidden_time) is needed to determine if the user has updated the displayed time.

B. DEFAULT WHERE/ORDER BY:

```
WHERE (TO_CHAR(date_column, 'SSSSS' ) = :time_field
OR :time_field IS NULL)
```

C. POST-QUERY trigger:

```
SELECT TO_CHAR(date_column, 'SSSSS' ) ,
TO_CHAR(date_column, 'SSSSS' )
INTO :time_field, :hidden_time
FROM base_table
WHERE ROWID = :block_name.ROWID
```

D. POST-INSERT trigger:

```
UPDATE base_table
SET date_column = TO_DATE(:date_field || :time_field,
        'DD-MON-YYSSSSS' )
WHERE primary_key = :primary_key
```

E. POST-UPDATE trigger:

```
UPDATE base_table
SET date_column = TO_DATE( :date_field || :time field,
        'DD-MON-YYSSSSS' )
WHERE ROWID = :block_name.ROWID
```

F. POST-FIELD trigger on the TIME field:

```
SELECT :database_field, :time_field
INTO :database_field, :hidden-time
FROM DUAL
WHERE NVL(:time_field,0) != NVL( :hidden_time,0)
```

NOTES:

1. Be sure to turn off " Abort trigger when step fails" .
2. POST-FIELD is used rather than POST-CHANGE because POST-CHANGE will not fire in the case where the field is changed to null.
3. This trigger is needed to flag an update (by changing a database field) in the case where the TIME is the only field being updated.

G. KEY-EXIT trigger on the TIME field:

```
#EXEMACRO GOFLD time_field; EXIT;
```

NOTE: This trigger is needed to fire the POST-FIELD trigger (see F above) in the case where the TIME is the only field updated, then [Exit] is hit while the cursor ls still in the TIME field. This trigger is not necessary in SQL*Forms 2.3.

IV. Exceptions in SQL*Forms 2.3

In SQL*Forms version 2.3, the DATE field type can be extended to include a four-digit year ('DD-MON-YYYY') simply be increasing the width of the field from 9 to 11.

Note that this new feature does not apply to the JDATE or EDATE field types.

A. In SQL*Forms 2.3, the 11-character wide DATE field type ('DD-NON-YYYY') is NOT a character string. It is of Oracle DATE datatype, so the TO_CHAR function may be required to achieve the desired results.

For example, trigger III-E above would be:

```
UPDATE base_table
SET date_column = TO_DATE(TO_CHAR(:date_field,
'DD-MON-YYYY' ) || NVL( :time_field,0) ,
'DD-MON-YYYYSSSSS' )
WHERE ROWID = :block_name.ROWID
```

5. SQL*Report Hints

5.1 RPT Tuning.

Try to avoid **.execute** and **.if..then..else** within the body loop of a RPT whenever possible. These operations are expensive and placed in the body loop of a report they will be executed once for each row found in the database.

5.1.1 How to parameterize the ORDER BY clause.

Sometimes you might need the same select statement to run with a number of different ORDER BY clauses, enabling you to sort the same data in a number of ways.

This can be done using the DECODE function, as shown in the **rpt** example below :

```
.REM ************************************************************************
.REM                                DECLARE VARIABLES
.REM
.declare EMPNO 9999
.declare ENAME a10
.declare JOB a9
.declare MGR B9999
.declare HIREDATE DATE
.declare SAL B99999.99
.declare DEPTNO 99
.declare TJEK 99
.REM ************************************************************************
.REM                                DEFINE MACROSELECT
.REM
.DEFINE select
     SELECT EMP.EMPNO, EMP.ENAME, EMP.JOB, EMP.MGR,
          TO_CHAR(EMP.HIREDATE,'J'), EMP.SAL, EMP.DEPTNO
     INTO   EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, DEPTNO
     FROM   EMP
     ORDER BY DECODE(&TJEK,1,EMPNO,3,SAL,5,DEPTNO,1),
               DECODE(&TJEK,2,ENAME,4,JOB,'1'),
                    DECODE(&TJEK,6,HIREDATE,sysdate),
                    DECODE(&TJEK,7,JOB,'1'),
                    DECODE(&TJEK,7,SAL,1)
..
.
.
.
.TELL "Sorted output of the EMP table :"
.TELL "  "
.TELL "  1 : EMPNO"
.TELL "  2 : ENAME"
.TELL "  3 : SAL"
.TELL "  4 : JOB"
.TELL "  5 : DEPTNO"
.TELL "  6 : HIREDATE"
.TELL "  7 : JOB, SAL"
.TELL "  "
.ASK "Write the number of the sort criterion "TJEK
```

The ORDER BY clause will vary dependent on the value of the variable TJEK. The only restriction here is that the columns mentioned in each DECODE function must be of the same data type - this is the reason for the two DECODE's used for sort criterion number 7.

5.1.2 RPT IF-Statements.

If using IF-Statements in *rpt*, the efficiency of the report will highly depend on the complexity of that statement (as noted in the SQL*Report Release Notes Version 1.0.10). To verify if a given report has a potential performance problem with respect to IF-statements, you could define two trace macros:

```
.define trace_35
        alter session set events '10035 trace name context forever'
..
.define trace_46
        alter session set events '10046 trace name context forever'
..
```

Note: In version 6.0.27 *alter session set sql_trace true* may be used instead.

The very first thing to do (after the definitions) would then be to invoke the SQL-statement trace:

```
.execute trace35
.execute trace46
```

Having executed the report, you may look at the generated trace file (default directory is $ORACLE_HOME/dbs), under the name <Oracle-process-number>_<OS-process-number>_$ORACLE_SID.trc (eg. 6_2614_H.trc).

Run the tkprof as described in Appendix I in the *Installation and User's Guide*.

```
tkprof <trace-file> /dev/tty
```

If you find SQL-statements selecting from the dual table with where clauses coresponding to your IF-Statements, you may try to rewrite those statements.

The trace file will also tell, how many cursers are used, and thus how much memory is allocated by the corresponging Oracle server process. If you find the number of used cursers too high, you may use the -N# option on RPT to reduce this number.

A little report example is here shown to clarify the circumstances in which the *rpt* program will transform the IF-Statement to a select statement.

```
1: .DATABASE IF_TEST
2: .DECLARE x 99
3: .DEFINE trace_a
4: alter session set events '10035 trace name context forever'
5: ..
6: .DEFINE trace_b
7: alter session set events '10046 trace name context forever'
8: ..
9: .DEFINE macro1
10:         .&label1
11:         Loop Counter =
12:         .PRINT x
13:         .ADD x x 1
14:         .IF "&x < = 4 + 1" THEN label1
15:         End of Loop
16: ..
17: .EXECUTE trace_a
18: .EXECUTE trace_b
19: .macro1
```

The result of the report is:

```
Loop Counter =

Loop Counter =
 1
Loop Counter =
 2
Loop Counter =
 3
Loop Counter =
 4
Loop Counter =
 5
End of Loop
```

The tkprof output will look like this:

```
count   = number of times OPI procedure was executed
cpu     = cpu time executing in hundreths of seconds
elap    = elapsed time executing in hundreths of secs
phys    = number of physical reads of buffers (from disk)
cr      = number of buffers gotten for consistent read
cur     = number of buffers gotten in current mode (usually for update)
rows    = number of rows processed by the OPI call

********************************************************************************
select 1 from dual where 1 <= 4 + 1

            count      cpu     elap     phys       cr      cur     rows
Parse:          1        0        0        0        0        0
Execute:        1        0        0        0        0        2        0
Fetch:          1        0        0        0        1        0        1
********************************************************************************
select 1 from dual where 2 <= 4 + 1

            count      cpu     elap     phys       cr      cur     rows
Parse:          1        0        0        0        0        0
Execute:        1        0        0        0        0        2        0
Fetch:          1        0        0        0        1        0        1
********************************************************************************
select 1 from dual where 3 <= 4 + 1

            count      cpu     elap     phys       cr      cur     rows
Parse:          1        0        0        0        0        0
Execute:        1        0        0        0        0        2        0
Fetch:          1        0        0        0        1        0        1
********************************************************************************
select 1 from dual where 4 <= 4 + 1

            count      cpu     elap     phys       cr      cur     rows
Parse:          1        0        0        0        0        0
Execute:        1        0        0        0        0        2        0
Fetch:          1        0        0        0        1        0        1
********************************************************************************
select 1 from dual where 5 <= 4 + 1

            count      cpu     elap     phys       cr      cur     rows
Parse:          1        0        0        0        0        0
Execute:        1        0        0        0        0        2        0
Fetch:          1        0        0        0        1        0        1
********************************************************************************
select 1 from dual where 6 <= 4 + 1

            count      cpu     elap     phys       cr      cur     rows
Parse:          1        0        0        0        0        0
Execute:        1        0        0        0        0        0        0
Fetch:          1        0        0        0        0        0        0
********************************************************************************
alter session set events '10035 trace name context forever'

            count      cpu     elap     phys       cr      cur     rows
Parse:          1        0        0        0        0        0
Execute:        0        0        0        0        0        0        0
Fetch:          0        0        0        0        0        0        0
********************************************************************************
alter session set events '10046 trace name context forever'

            count      cpu     elap     phys       cr      cur     rows
Parse:          1        0        0        0        0        0
Execute:        1        0        0        0        0        0        0
Fetch:          0        0        0        0        0        0        0
```

```
********************************************
OVERALL TOTALS FOR ALL STATEMENTS

             count      cpu     elap     phys       cr      cur     rows
Parse:           8        0        0        0        0        0
Execute:         7        0        0        0        0       10        0
Fetch:           6        0        0        0        5        0        5

Total number of cursors: 8
```

Even if the line 11 of the report is changed to the following, select statements are still issued.

```
.IF "&x <= '5' " THEN label1 - or
.IF "&x <= &y + 1 " THEN label1 - where y is set to 4.
```

The only way to avoid the select statements is to change the IF-Statement, so that both sides have the same datatype, and beeing only very simple expressions:

```
.IF "&x <= 5 " THEN label1 - or
.IF "&x <= &y " THEN label1 - where y has the value 5.
```

Then the tkprof output will look like this:

```
count    = number of times OPI procedure was executed
cpu      = cpu time executing in hundreths of seconds
elap     = elapsed time executing in hundreths of secs
phys     = number of physical reads of buffers (from disk)
cr       = number of buffers gotten for consistent read
cur      = number of buffers gotten in current mode (usually for update)
rows     = number of rows processed by the OPI call

********************************************************************************
alter session set events '10035 trace name context forever'

             count      cpu     elap     phys       cr      cur     rows
Parse:           1        0        0        0        0        0
Execute:         0        0        0        0        0        0        0
Fetch:           0        0        0        0        0        0        0
********************************************************************************
alter session set events '10046 trace name context forever'

             count      cpu     elap     phys       cr      cur     rows
Parse:           1        0        0        0        0        0
Execute:         1        0        0        0        0        0        0
Fetch:           0        0        0        0        0        0        0


********************************************
OVERALL TOTALS FOR ALL STATEMENTS

             count      cpu     elap     phys       cr      cur     rows
Parse:           2        0        0        0        0        0
Execute:         1        0        0        0        0        0        0
Fetch:           0        0        0        0        0        0        0

Total number of cursors: 2
```

5.2 Modify the database in RPT.

It is possible to issue insert, delete and update SQL-statements in the report, but we avise you to be careful here. The execution of the report may be done in batch, so special care

must be observed with regard to locking.

- The update statements can only be used in *.execute* macroes.
- Remember to issue a *commit* to release the locks as soon as possible after the update.
- Do not lower the priority of the report, since it may prevent on-line applications from modifying the same tables for a long periode.

5.3 RPT Error Messages and Codes.

Received from the OLS System Bulletin (no 64738.039)

Note: Page references are for the SQL*Report User's Guide 1.0 (part# 3603-V1.0).

RPT-000: too few arguments

One of the RPT commands does not have enough arguments, or contains a carriage return. For example, a macro with no name:

```
.DEFINE SELECT empno, ename
INTO empno, ename
FROM emp
```

RPT-0001: argument overflow

RPT-0002: illegal variable or macro name

This error indicates that a variable has been declared or a macro has been defined with an illegal name. The rules for naming variables and macros are on p.59.

RPT-0003: illegal label name

This error indicates taht a label has been used which has an illegal name. The rules for naming statement labels are on p.59.

RPT-0004: formatting: no room to insert commas

RPT-0005: '..' command while not in macro

This error will occur if two periods (..) are found on a line by themselves, outside of a macro. This indicates the completion of a macro definition. (p.101-2)

RPT-0006: *** No digits or decimals specified

RPT-0007: *** Too many suffixes

RPT-0008: label multiply defined in macro

This error will occur if the report uses the same label name more than once in the same procedural macro. Since label definitions are local, the same label name may be used in different macroes. Also, only the first 30 characters on an identifier name are significant. (p.59,115)

RPT-0009: macro has already been defined

This error will occur if the report uses the same macro name in more than one DEFINE statement. Also, only the first 30 characters of an identifier name are significant. (p.59)

RPT-0010: macro call nesting level exceeded

This error will occur if, in a REPORT or EXECUTE statement, the same SQL macro is called recursively.

RPT-0011: nested macro definition

RPT-0012: *** ignoring following source lines (up to '..'):

This message has obvious meaning.

RPT-0013: variable multiply declared -- this declaration ignored

More than one variable with the same name have been declared in the report. Also, only the first 30 characters of an identifier name are significant. (p.59)

RPT-0014: meinline RPT program contains no executable lines

RPT-0015: *** <x> in illegal position

RPT-0016: divide by zero (<statement>)

This error will occur if a DIV statement attempts to divide a number by zero, which is undefined. (p.123)

RPT-0017: formatting: 2nd decimal point is illegal in <x>

RPT-0018: illegal display format <format>

Legal display formats for DECLARE statements are found on p.96-99. Check for typos.

RPT-0019: illegal display format <format>: out of range

Legal display formats for DECLARE statements are found on p.96-99. Check that alphanumeric variables are not declared as more than 255 characters.

RPT-0020: label <label> is undefined

This may occur if the report has an IF statement where the expression contains blank characters, but is not enclosed within double quotation marks. For example '.if $counter = 1 then label1 (p.117)

RPT-0021: number too large for printing or assignment

RPT-0022: variable <name> is not declared

The statement refers to a variable that has not been declared correctly (using a DECLARE statement) within the report, or you have used a literal value where a variable was expected. For example: .EQUAL counter 1 (the SET command should be used to do this). Also, this error will occur if a label is used outside of a procedual macro. Be sure to check for typos. (p.95-9, 115)

RPT-0023: conversion error: <error>

This error, along with RPT-0073, will occur if the report uses the SET statement incorrectly. For example: '.SET variable1 variable2.' The Set statement sets the value of a variable equal to a literal value, To set the value of a variable equal to the value of another variable, use the EQUAL command. Also, see cause errors RPT-0011 and RPT-0035 to occur. (p.101-2)

RPT-0024: macro <name> not proporly ended

This error occurs if a macro definition is not ended by two periods (..) on a line by themselves. This may also cause errors RPT-0011 and RPT-0035 to occur. (p.101-2)

RPT-0025: <x> to string variable. result truncated

RPT-0026: cannet execute if expression: *** <statement>

The syntax of the IF statement is incorrect. Make shure that all program variable names are preceded with an ampersand. This will also cause the error ORA-0704: invalid column name. Also, an IF statement cannot be more than 132 characters long. To get around this problem, shorten variable names, or break up into more than one IF statement. (p.117-8)

RPT-0027: if: numeric variable <name> has non-numeric value

RPT-0028: macro <name> is not declared

RPT-0029: cannot execute SQL statement; *** <statement>

This could be caused by a number of errors in the SQL statement of a select macro. Try running the SQL statement in SQL*Plus to determine what is causing the error. Also, check that the select macro only contains the text of the SQL query and nothing else (for example, a Table (#T) command). If you accidentally include a semicolon at the end of the select statement, this error will occur along with the error ORA-0911: invalig character. (p.102-4)

RPT-0030: SQL macro <name> cannot be executed

This error occurs if an EXECUTE statement attempts to execute a macro that is not defined in the report. Check for typos. (p.105-6)

RPT-0031: too many variable references: variable <name> ignored

RPT-0032: SQL statement in macro <name> is too long

Try breaking up the SQL statement. The limit in version 1.0 is 2000 characters and the limit in version 1.1 is 10,000 characters.

RPT-0033: SELECT macro <name> has no FROM clause

This error will occur if the select macro is missing an INTO clause. (p.102-4) For select macros using set operations (Union, Intersect, or Minus) refer to p.60 for the correct usage of an INTO clause. Also, in version 1.1.5, this error will occur if a column in the select list has the word 'FROM' as part of its name.

RPT-0034: missing report name: <name>

RPT-0035: body macro <name> unspecified or undeclared

This error will occur if a REPORT statement contains a body macro which has not been defined in the report.

RPT-0036: too many parallel reports

This error will occur if the report is disjunctive (where two or more select macros are specified on one REPORT statement joined by ANDs or ORs) and more than 4 select macros have been specified. (p.112)

RPT-0037: found <text> where 'AND' or 'OR' was expected

This error will occur if the report is disjunctive (the <SELECT macros> argument to the REPORT statement is enclosed within double quotes), but is missing an AND or an OR. (p.112, 114)

RPT-0038: user errors prevent execution of report

There are a number of causes for this error. Look for typos, such as references to macros with different spellings.

RPT-0039: stop.

Indicates that a STOP statement was encounted in the report.

RPT-0040: first argument not numeric or others not date types

RPT-0041: *** *** ERROR at line <n>: RPT-<error>: <text>

RPT-0042: *** Source: <line> ***

These messages indicate where the error probably occured.

RPT-0043: RPT internal error: ufree(<x> - bad address

RPT-0044: ualloc: out of heap space (needed <x>)

RPT-0045: insstr: destination too small

See RPT-0026.

RPT-0046: *** ORACLE error: <error>

RPT-0047: *** Operation : <operation>

RPT-0048: *** Function : <function>

These messages indicate that an Oracle error has occured.

RPT-0049: Month must be between 1 and 12

The following error will occur: conversion error: Month must be between 1 and 12 if a date variable is 'asked' for, but the month is not between 1 and 12.

RPT-0050: Year must be between 00 and 99

The following error will occur: conversion error: Year must be between 00 and 99 if a date variable is 'asked' for, but the year is not between 00 and 99.

RPT-0051: Day must be between 1 and the last of month

The following error will occur: conversion error: Day must be between 1 and last of month if a date variable is 'asked' for, but the day is not between 1 and the last of month.

RPT-0052: Usage is: RPT inputfile outputfile user/password options

RPT-0053: Options are:

RPT-0054: -n Number of open cursors (e.g., -n10 for 10 cursors)

RPT-0055: -? Display RPT version

These messages will appear, along with RPT-0066, RPT-0074, and RPT-0075, if the syntax on the RPT command line is incorrect. (p.57)

RPT-0056: Connected to <database>

This message tells you the version of the ORACLE database that the report is using.

RPT-0057: can't open <file> for input

The input file specified on the RPT command line does not exist or cannot be opened. Check for typos. (p.57)

RPT-0058: can't create <file> for output

RPT-0059: line cannot be executed due to errors reported earlier

RPT-0060: label not allowed on this command

RPT-0061: <n> lines written to output file

RPT-0062: <n> errors encounted and written to error file

RPT-0063: <n> lines read from input file

These messages have obvious meaning. Normally, you do not need to worry about the number of lines in the interim file.

RPT-0064: Unknown format type

RPT-0065: Report aborted

Indicates that the report program has been terminated. because errors have occured.

RPT-0066: -x eXecute report even if syntax errors are found

This message will appear if the syntax on the RPT command line is incorrect. (p.57)

RPT-0067: Errors during ROLLBACK

RPT-0068: Errors during COMMIT

RPT-0069: You are not authorized to use SQL*Report

RPT-0070: WARNING: Your SQL*Report license will expire soon

RPT-0071: Maximum open cursors exceeded

Try increasing the number of open cursors. See p.32 and p.231 of the ORACLE Database Administrator's Guide 5.1.

RPT-0072: Date format must match date type

The following error will occure: conversion error: Date format must match date type if a date variable is 'asked' for, but the format of the input is not MM/DD/YY. (p.126)

RPT-0073: Illegal number

See RPT-0023.

RPT-0074: -a Number or rows per array_fetch (e.g., -a20 for 20 rows)

RPT-0075: -o format numbers using Old RPT formatting rules

These messages will appear if the syntax on the RPT command line is incorrect. (p.57)

RPT-0076: SELECT macro <name> has no INTO clause

This error will occur, in version 1.1.8, if the select macro is missing an INTO clause. (p.102-4)

RPT-0077: Value entered or set for <variablename> is greater than declared -- value truncated

This error will occur, in version 1.1.9.4, if the data of the variable in a .set is greater than the format width defined for the column it is to be printed. The data will be truncated.

5.4 RPF Error Messages and Codes.

Received from the OLS System Bulletin (no 64738.040)

Note: Page references are for the SQL*Report User's Guide 1.0 (part# 3603-V1.0).

RPF-0001: table width (<n>) exceeds containing column width (<n>)

This error may occur if the report uses an Indent (#I) command and does not close it using a Table End ($TE) command. Also, this error can be caused by trying to reopen a table that is already open. For example, if the report uses the Indent command (#I) and then closes it using the Table End command (#TE), it does not need to reopen the provious table with the Table command (#T).

RPF-0002: token width (<n>) exceeds containing column width (<n>)

This error occurs when a token is wider than the containing column. A token is a string of numbers or letters which does not contain any spaces, so RPF in unable to wrap the extra characters to the next line.

RPF-0003: column literal width (<n>) exceeds column width (<n>)

This error indicates that the length of each line of a column literal must not exceed the length of the current column. (p.35)

RPF-0004: ident value (<n>) exceed column width (<n>)

This error occurs if the <number of spaces> argument to the Indent (#I) command exceeds the current column width. (p.39)

RPF-0005: centered text too long for column width (<n>)

This error will occur if the text to be centered using the Center (#CEN) command is wider than the current column. This may also cause error RPF-0002 to occur. (p.34)

RPF-0006: maximum table number 29 exceeded

The <table id> argument to a Define Table (#DT) command must not be grater than 29, or this error will occur. (p.37)

RPF-0007: maximum number of columns (255) exceeded

The maximum number of columns per table is 255.

RPF-0008: column definition exceeds maximum position (<n>)

This error will occur if the column definitions in a Define Table (#DT) command exceed the width of the column i which the table is invoked. This may also cause errors RPF-0012 and RPF-0029 to occur. (p.37)

RPF-0009: column <n> overlaps its predecessor; or end precedes start

This error will occur if the starting position of a column is not at least one greater than the ending position of the previous column in a Define Table command (#DT). This amy also cause errors RPF-0012 and RPF-0029 to occur. (p.37)

RPF-0010: column width (<n>) too small for paragraph indent

This error occurs if the report uses a Paragraph (#P) command within a column that is less than 6 character wide.

RPF-0011: table <n> is multiply defined

The report uses the same table id for more than one table in the report. This is only a warning message, since specifying a previously used table id will cause the new definition to replace the old definition. (p.37)

RPF-0012: illegal Table (#T) command - table <n> is undefined

The report refers to a table that is not defined within the report. This may also cause error RPF-0029 to occur. (p.46)

RPF-0013: illegal argument to Alternate Page Number (#APN) command

This error occurs when a non-numeric argument is used in the Alternate Page Number (#APN) command. (p.34)

RPF-0014: internal error: text buffer string truncated

RPF-0015: missing terminator (#) for Column Literal (#CL)

This error indicates that a Column Literal (#L) command was not terminated by a single period (.) or a pound sign (£) in the first column of an otherwise empty line. (p.35)

RPF-0016: Illegal (non-numeric) argumnet in Column Skip (#CS) command

This error occurs when a non-numeric argument is used in the Column Skip (#CS) command. (p.36)

RPF-0017: Define Table (#DT) must have table number and at least 1 column

This error occur if the report has a DT command with a <table id> argument, but no defined columns. For example: '#DT 2#' At least one column must be defined in a table. This error will also occur if you use a table id which is non-numeric. (p.37)

RPF-0018: Illegal argument in Figure (#F) command

This error occurs when a non-numeric argument is used in the Figure (#F) command. (p.38)

RPF-0019: illegal (non-numeric) argument in Horisontal Space (#SH) command

This error occurs when a non-numeric argument is used in the Horizontal Space (#HS) command. (p.39)

RPF-0020: illegal argument in Indent (#F) command

This error actually refers to the Indent (#I) command. It occurs when a non-numeric argument is used. (p.39)

RPF-0021: missing terminator (#) for literal (#L)

This error indicates that a Literal (#L) command was not terminated by a single period (.) or pound sign (£) in the first column of an otherwise empty line. (p.40)

RPF-0022: odd number of column definitions

This error will occur if a Define Table (#DT) command does not use the correct syntax. For example, a DT command having an even number of arguments: '#DT 1 12 14 65 #' Remember taht the first argument is the table id, and NOT the starting position of the first column. This may also cause errors RPF-0012 and RPF-0029 to occur. Also, this error may occur if the entire DF command does fit on one line, or there is not a space before the closing '#'. (p.37)

RPF-0023: illegal argument in Page Number (#SPN) command

This error actually refers to the Start Page Numbering (#SPN) command. It occurs when a non-numeric argument is used. (p.45)

RPF-0024: illegal page number type: must be in range [1..4]

The <type> argument given to the Start Page Numbering (#SPN) command must be an integer from 1 to 4 only. (p.45)

RPF-0025: illegal page number position: must be in range [1..250]

The <pos> argument given ti the Start Numbering (#SPN) command must be an integer from 1 to 255 only. (p.45)

RPF-0026: skip lines too large

RPF-0027: illegal argument to space (#S) command

This error actually refers to the Skip (#S) command. It occurs when a non-numeric argument is used. (p.44)

RPF-0028: illegal value in Spacing (#SP) command: require number 1 to 4

This error actually refers to the Space (#SP) command. The argument given to this command must be an integer from 1 to 4 only. (p.44)

RPF-0029: unmatched Table End (#TE) command

There is a Table End (#TE) command in the report, with no matching Table (#T) command. Be careful that there is not a Table End (#TE) command in the head, body, or foot macro of the select macro that did not return any rows. (p.46)

RPF-0030: illegal Table (#T) command - missing table number

RPF-0031: illegal (non-numeric) argument - missing table number

This error occurs when a non-numeric argument is used in the Vertical Space (#VS) command. (p.48)

RPF-0032: error closing input file

Indicates that your system has run out of disk space.

RPF-0033: error closing output file

Indicates that your system has run out of disk space.

RPF-0034: command line error: cannot open file <filename> for input

The input file specified on the RPF command line does not exist or cannot be opened. Check for typos. (p.49)

RPF-0035: command line error: cannot open file <filename> for output

This indicates that there is not room on the file for the output file.

RPF-0036: token <token> is too long

The largest input token allowed is 255.

RPF-0037: <n> lines read from <filename>

RPF-0038: <n> lines written to <filename>

RPF-0039: <n> lines written to error file

These messages have obvious meaning. Normally, you do not need to worry about the number of lines in the interim file. The error file is the terminal screen.

RPF-0040: menory exhausted -- needed <n> bytes

RPF-0041: *** ERROR at line <n>: RPF-<error>: <text>

RPF-0042: *** (last RPF command recognized was 'line' at line <n> ***

These messages indicate where the error probably occured.

RPF-0043: Usage is RPF input_file output_file -switches

RPF-0044: Switches:

RPF-0045: a - All bold face m - dynamic memory usage

RPF-0046: b - Bold face underlines p:n:m - Page #'s N to M

RPF-0047: d:d - Device is Diablo r - Reverse underlining order

RPF-0048: d:v - Device is VT100 s - Spool to line printer

RPF-0049: f - Form feed for page eject u - Upper case output

RPF-0050: i - Initial page eject w - enable Wait (pause), if device = d or v

These messages will appear if the syntax on the RPF command line is incorrect. If you are using more than one switch, then they must be combined. For example: 'rpf report report -wd:v'. (p.49-51)

RPF-0051: Illegal argumnet in Page (#PAGE) command

RPF-0052: Dynamic menory statistics:

RPF-0053: <n> bytes allocated

RPF-0054: <n> bytes freed

RPF-0055: <n> bytes for RPF context area

RPF-0056: <n> bytes for <n> text buffers

RPF-0057: <n> bytes for <n> colnodes

RPF-0058: <n> bytes (at least) for <n> table definitions

RPF-0059: <n> bytes for <n> tblnodes

RPF-0060: <n> bytes for <n> tblrows

RPF-0061: <n> bytes for <n> wcbs

RPF-0062: Diablo device is not supported (yet)

RPF-0063: <x> has been spooled for printing

RPF-0064: <n> is an illegal title width

Titles may be up to 255 characters long.

RPF-0065: title is longer than specified width

RPF-0066: start page number exceeds stop page number

RPF-0067: You arenot authorized to use SQL*Report

RPF-0068: WARNING: Your SQL*Report license will expire soon

5.5 RPT/RPF through Pipes.

The *rpt* and *rpf* programs are not build to reflect the dynamic way of handling standard-input and standard-output known in Unix. This is however of interest, since the possibility of running *rpt/rpf* without storing tempory files on disk, may save diskspace, administration, and increase response time.

As an example of how this may be achieved, look at the following *rptrpf* Shell-script:

```
## rptrpf in $ORACLE_HOME/bin
## DDE 1. OCT 1990, MJ, TPO, STR
## Usage rpt-file-name "USERNAME/PASSWD" ["RPTOPTIONS"] ["RPFOPTIONS"]
## Answers for the rpt are read from std-input, and the report
## is generated on std-out. stdout and error from rpt and rpf is
## placed in std-err.

RPTFIL=$1
LOGIN=$2
RPTOPTIONS=$3
RPFOPTIONS=$4

RPFFIL=/tmp/$$.rpf
LISFIL=/tmp/$$.lis
ERRFIL=/tmp/$$.err

## Check that the number of parameters are ok.
if [ -z "$2" ]; then
  echo Usage: rptrpf rpt-file-name \"USERNAME/PASSWD\" [\"RPTOPTIONS\"] \
       [\"RPFOPTIONS\"] >&2
  exit 1
fi

## Check the readability of the rpt-file.
if test ! -r $RPTFIL.rpt; then
  if test ! -r $RPTFIL; then
    echo "The file $RPTFIL(.rpt) cannot be read" >&2
    exit 1
  fi
fi

trap "rm -f $RPFFIL $LISFIL $ERR2FIL; echo $0 of $1 aborted >&2; exit 1" \
       1 2 3 15

## create the pipes
/etc/mknod $RPFFIL p
/etc/mknod $LISFIL p

## Start the rpt brogram in background
RPT='/bin/backgr $ORACLE_HOME/bin/rpt $RPTFIL $RPFFIL $LOGIN $RPTOPTIONS \
       3>&2'
if test $?; then

## Start the rpf brogram in background
   RPF='/bin/backgr $ORACLE_HOME/bin/rpf $RPFFIL $LISFIL $RPFOPTIONS \
       3>> $ERRFIL 2>&3'

## open and close the rpf-pipe, to make the rpf program ternimate if the rpt
## program for some reason did not start - or did stop very soon after start
   /bin/backgr sleep 20 >/dev/null 3>$RPFFIL

## Show the result on std-out
   cat $LISFIL

## Wait for the termination of rpt and rpf
   RPTRES='wait $RPT'
   RPFRES='wait $RPF'
fi

## Show possible errors on std-err
echo "RPT-result: $RPTRES, RPF-result: $RPFRES" >&2
cat $ERRFIL >&2

## Remove work-files
rm -f $RPFFIL $LISFIL $ERRFIL
```

Note that we did not use the '&' character to start the processes in background. This is due to the fact that all Oracle processes should be running on the same priority, and '&' will put them on 4 points lower. Instead we use a little C-program *backgr* donated by Thomas P. Olesen - DDE:

```
/*************************************************************************/
/* Module:      backgr.c                                                 */
/*                                                                       */
/* Produces a process                                                    */
/*                                                                       */
/* Written by: tpo                          Date: 01.10.1990             */
/* Copyright (c) 1984 by Dansk Data Elektronik A/S                       */
/*************************************************************************/
/* Modification list:                                                    */
/*************************************************************************/

static char *_version = "@(#) backgr version 3.1";

#include    <std.h>
#include    <stdio.h>
#include    <sys/signal.h>

main(argc, argv)
int argc;
char **argv;
{
    int i,hand,pid;

    if (argc < 2) {
        fputs("usage: backgr command arg ...\n", stderr);
        exit(2);
    }

    argv[argc] = 0;

    if (pid=fork()) { /* parent */
       if (pid < 0) {
           exit(-1);
       } else {
           printf("%d\n",pid);
           exit(0);
       }
    }

    /* child */
    close(1);
    if (dup(3)!=1)  {
       fprintf(stderr,"Error when dupping filedesc 3, use desc 3 as stderr.\n");
       exit(-1);
    }
    close(3);

    execvp(argv[1], &argv[1]);
    fputs(argv[1],stderr); fputs(": cannot be executed\n",stderr);
    testerr(-1);
}
```

6. Pro* Hints

6.1 Scope of variables and functions

The following PCC example results in a bus-error :

```
#include <stdio.h>
#include <std.h>

EXEC SQL BEGIN DECLARE SECTION;
VARCHAR uid[3];
VARCHAR pwd[3];
VARCHAR stat[2];

EXEC SQL END DECLARE SECTION;

EXEC SQL INCLUDE SQLCA;
main()
 {
        strcpy(uid.arr,"cfj");
        strcpy(pwd.arr,"cfj");
        uid.len = (pwd.len = 3);
        printf("connecting\n");
        EXEC SQL CONNECT :uid IDENTIFIED BY :pwd;
        printf("connected\n");
        EXEC SQL COMMIT WORK;
}
```

The VARCHAR variable **stat** is also the name of a c-function in the operating system which Oracle calls during logon. This function however, is never linked to the program. The linker mistakes the structure having the same name for the function and assumes that the function is already available and that the address of this function is that of the structure. This is because functions and **global** variables of any kind in a C program shares the same name-space - that is to say that there is no typechecking of function names versus global variable names in C at compile time.

The problem with the VARCHAR **stat** variable is not a PCC problem and it is not a bug in the linker - it is a characteristic of C and the way the C language is defined :

> Do not use global variables having the same name as a function in your program!

Another way to get around the problem is to declare global variables as **static** so their scope is limited to the file in which they are declared.

6.2 Calling PCC

When calling pcc do *not* specify a filename longer than 13 characters for iname (the input file). If a filename of 14 characters is given the file will be removed (!) due to an error in pcc.

6.3 Use symbolic constants in the declare section

The Pro*C precompiler does not allow symbolic constants in the declare section of a Pro*C program.

Symbolic constants are, however, extremely usefull in the declare section as size specification for array and VARCHAR variables. We therefore recommend that symbolic constant are defined and expanded by a macro processor such as m4 or cpp *before* running the Pro*C precompiler.

6.4 Array operations

Pro*C accepts host variables declared as arrays in INSERT, UPDATE, SELECT and FETCH statements. When processing multiple rows it is in general faster to apply an array operation than processing individual rows one by one.

When a FOR clause is used the number of array elements specified must be different from 0. If 0 elements are specified the Oracle error 1009 "missing mandatory parameter" is raised.

6.5 Layout of VARCHAR variables

The size of the array part of a VARCHAR variable may be longer than the declared length.

As an example consider the following declarations:

```
VARCHAR x[15];
VARCHAR y[100][15];
```

Here sizeof(x.arr) equals 15 as expected, but sizeof(y[0].arr) equals 18. When an array of VARCHAR variable is declared the Pro*C precompiler rounds up the length of the arr field such that the size of each VARCHAR in the array (ie. sizeof(y[0])) is a multiple of 4 bytes.

The point to be made here is: Do *not* use sizeof(...) to determine the declared size of a VARCHAR variable.

6.6 Generel programming hints

When the Precompiler command **WHENEVER xxx** is written somewhere in your program, it remains active in the rest of the programfile unless it is overwritten by another similar WHENEVER xxx command. This provides the programmer with an easy way of checking for errors and warnings programwide. However, using WHENEVER statements in conjunction with GOTO LABEL statements can lead to undesired results, as demonstrated by the following example :

```
function1()
{
WHENEVER SQLERROR GOTO errorhandling;
EXEC SQL SELECT .....;
errorhandling:
        printf(" error in ....");
}

function2()
{
EXEC SQL INSERT INTO .....;

}
```

A program like the one above will precompile normally, but will result in a symbol referencing error at compile time. This is because the **WHENEVER SQLERROR GOTO errorhandling** statement is still active in function2(). The C compiler will attempt to find a goto label called errorhandling, but will fail to do so since it cannot handle goto's to a label in another function.

This means that you should always include labels named in WHENEVER statements in every procedure where this statement is still active. This will make your program a bit unstructured.

Another way of dealing with this problem is to do your own errorhandling, checking the **sqlcode** in the SQLCA structure each time an SQL statement is executed and calling an errorhandling procedure when an error is detected rather than jumping to a label.

6.7 ASCII Nulls

Using PRO*C (or OCI) it is possible to insert ASCII Null characters in character fields. This is *not* recommented, since these values are difficult to handle.

Suppose we have a table like:

create table nulltest (a **char**(10))

And in our PRO*C program has the following sequence of statements:

```
EXEC SQL BEGIN DECLARE SECTION;
VARCHAR field[10];
EXEC SQL END DECLARE SECTION;

main(argc,argv)
int argc;
char **argv;
{
   /* Log on to the database */
   strcpy(field.arr, "hej");
   field.len = strlen(field.arr) + 2;
   field.arr[4] = field.arr[5] = 0;
   EXEC SQL INSERT INTO NULLTEST ( a ) VALUES ( :field );
   EXEC SQL COMMIT WORK RELEASE;
}
```

Where the *field* variable has some extra ascii nulls.

When the table is queried:

```
SQL> select a, length(a), dump(a) from nulltest;

A          LENGTH(A) DUMP(A)
---------- --------- ----------------------------------------------------
hej                5 Typ=1 Len=5: 104,101,106,0,0

SQL> select a from nulltest where a = 'hej';

no records selected

SQL> select a from nulltest where a like 'hej%';

A
----------
hej
```

So be carefull dealing with ascii null characters.

6.8 Do not use Simple Char

If one of the PRO*C variables are declared like this:

```
EXEC SQL BEGIN DECLARE SECTION;
  char foo;
EXEC SQL END DECLARE SECTION;
```

Then *pcc* will generate wrong c code. Instead the following declaration should be used:

```
EXEC SQL BEGIN DECLARE SECTION;
  char foo[1];
EXEC SQL END DECLARE SECTION;
```

This Hint was received from the *OLS System Bulletin 98489.889*.

7. SQL * Star - The Oracle Networking Concept

This is a brief introduction to the Oracle network concept, including some rough recommendations on choosing an appropriate strategy for Oracle applications in a network environment. **NOTE** The asunchronous driver is currently not supported.

Detailed information may be found in the SQL * Net manuals:

- Supermax ORACLE 5.0, "Installation and User's Guide" (Chapter 6: Alias administration, TWO_TASK environment. Chapter 11: Linking SQL * Net drivers.).
- "SQL*Net User's Guide" This manual is instructive, but will not provide a overview - reading this manual is not a must)
- SQl*Net TCP/IP Driver, "Installation and User's Guide" (Parameter setting, Server start)

The basics of the Oracle network concept is the SQL * Star architecture. This architecture allows Oracle applications to communicate in a heterogeneous network environment comprising various hosts, operating systems, network protocols and database systems.

7.1 SQL * Star architecture

The SQL * Star architecture is an integrated part of the Oracle product, and contains three main issues:

SQL * Net — This part of the architecture handles the basic communication between the user process and the Oracle process. SQL * Net uses various protocols drivers (Pipe, Fast, TCP/IP) to exchange information between processes.

SQL * Net insures reliable communication between various host computers with different operating systems. This makes it possible to build at network consisting of both local- and wide-area connections using a combination of the implemented protocols.

SQL * Connect — This feature offers the possibility of accessing other relational databases such as SQL/DS and DB2. SQL * Connect may be used in combination with SQL * Plus and SQL * Report. SQL * Forms, SQL * Calc and all PRO based systems will be able to use SQL * Connect later.

DBS Links — Database Links may be created between any database throughout the network, as long as the two involved hosts are in fact connected. Using the link identifier it is possible to query tables throughout the network.

The SQL * Star architecture offers distributed query access to all databases on hosts connected in the network. This includes updating the currently connected database - the client database - with data selected from other server databases in the network. SQL * Star does not however offer facilities to distribute updates yet.

In order for all hosts to interact the network they must be connected in a star formation, because hosts cannot act as forwarding nodes in the network if updating is to be performed. Queries may be forwarded using database links.

7.2 SQL * Net Protocol Drivers

The actual possibilities for the individual hosts to interconnect are defined by the available protocol drivers. As new protocols are implemented, they are inserted as drivers in SQL * Net, hereby extending the usage of the total system.

*7.2.1 SQL * Net Pipe Driver*

The pipe protocol is part of the normal two-task interface between the user process and the Oracle RDBMS process. This scheme protects Oracle RDBMS processes from any fatal errors made by user processes, thus making Oracle reliable in an multi user environment.

The pipe protocol may be used between more databases residing on the same host. Normally the environment variables are set to a default database, and the pipe protocol is used when accessing an alternative database on the same host, using a database link or the remote logon facility.

*7.2.2 SQL * Net Fast Driver*

The Fast driver provides no advantage to the Pipe driver and is not recommended.

*7.2.3 SQL * Net TCP/IP Driver*

The TCP/IP driver has proven to be a fast communication connection between different machines. It uses the Ethernet/Cheapernet local area network and has few parameters to configurate.

This protocol has been widely implemented by various computer vendors and is recommended for professional use.

7.3 Connecting Computer Hosts With SQL * Star

There are basically three ways of interconnecting computers with SQL * Star, including remote login, database links and connecting to multiple databases. All types of connections may be established in a heterogeneous network environment with a mix of hosts, operating systems, database systems and protocols.

Remote logon

This facility permits a process to performs a login on a remote database. The user process residing on the client host will handle all terminal and file i/o as well as any other non SQL computing. Whereas the SQL-processing will be handled by the Oracle RDBMS process on the remote server host.

A substantial processing load is hereby taken over by the local system offloading the host. Response time is therefore improved.

Database Link

When specifying a database link, tables from different server databases may be accessed in the same query, and the query processing may be divided between the involved hosts. This facility is normally used to retrieve statistical reports covering a number of local branches. It may also be used to make a query access to an Oracle 5 database on the same machine, thous providing a smoother conversion to Oracle 6 (only with TPC/IP driver).

Multiple Logins

When using Oracle's programmatic interface PRO * C, it is possible to logon to more than one database at a time. In this way programs may update tables distributed on different databases.

NOTE: The transactions concept *commit* offered by Oracle does only allow commit on updates of one database at a time. Therefore it is up to the programmer to insure consistency between the various databases.

7.4 Suggested Strategy Using SQL * Star

When choosing an appropriate strategy for an application in a network environment, one must carefully consider the nature of the application.

It is important to identify where data is captured, modified and processed. Concepts as processing power, availability, backup procedures and security must also be considered.

We advise you to start your design and implementation on a centralized database concept, and if and when response time degrades, applications may be installed locally using the remote logon facility, still operating on the central database.

If better response time is still required, some rather static tables may be distributed to allow faster local data validation, using the database link facility to access the remote host. This scheme does not allow updating the remote database and can only be used for queries. Multiple logons from Pro*C would however handle the job.

7.5 The TCP/IP Driver

The use of the TCP/IP driver demands the version 1.7 of SupermaxTCP software. Early hints on MCU locking are no longer applicable.

NOTE: The driver can be tested on one machine alone, acting as both client and server.

7.6 Integrating SQL * Star in software products

This hint decribes the most obvius way of integrating SQL * Star in software products, using the remote logon facility. This hint is mainly a highlight of chapter 15 in Supermax ORACLE 5.1.17, Installation and User's Guide. The topics are of special interest:

Dependence — As a precondition multi user applications must depend only on shared resources from the database. Files and environment variables may be accessed freely on the local host as long as they do not constitute a common resource. Normally this does not cause problems. Note that date and time may be different on different machines (se hint on SQL * Forms).

Database — The actual access of a remote database is indicated in combination with giving the userid and password to Oracle. A speciale environment variable **TWO_TASK** may be set on the command line when calling the program:

```
TWO_TASK=boss /alib/lib/kalender
```

Makefile — Changes must be made to the disribution due to the fact that the custemer may bey different combinations of SQL * Net drivers, and new releases will come. The distribution of the software products must be altered to include archives (or obejct modules) and makefiles. A makefile for an Oracle product using OCI on version 1.2 may look like this:

```
# Makefile.kalender  Used for linking kalender system with new drivers.
#

LIBS=lib.a ../datolib/datolib.a

OBJ=kalender.o month_number.o holiday.o set_date.o ....

kalender:
                cc -o kalender $(OBJ) $(LIBS) \
                        /lib/libpa.a /lib/libpext.a /lib/libm.a \
                  pbndrvst.o \
                  $(ORACLE_HOME)/pascall/libp4ora.a \
                  $(ORACLE_HOME)/net/osntab.o 'netlibs' \
                  $(ORACLE_HOME)/oci/libhlic.a \
                  $(ORACLE_HOME)/oci/libhli.a
```

Note the inclusion of **osntab.o**, the driver definition module (ref. Installations and User's Guide, Chapter 15). The inclusion of **'netlibs'** will return the archives used by drivers installed on each Supermax.

newpkg

No changes should be made to the standard intallation of the linked product, wich may be installed and used as normal if no drivers are requered.

But, the software must be able to be relinked at the custemers site in order to include installed or drivers baught in the near future.

This includes distributing archives (or obejct modules) and inserting a call for the products makefiles in **ORACLE_HOME**/*net/applmake*, a central Oracle application generation script. The installation script *INSTALL* used when generating the *newpkg* files may look something like:

```
#
# Enter product makefile in "applmake"
#
# !! $ORACLE_HOME should be set !!
# !! Hints: Capter 3. Oracle Database Administration !!
#
if test -s $ORACLE_HOME/net/applmake
then
  grep -v Makefile.kalender $ORACLE_HOME/net/applmake > remove
  echo "(cd $KALENDER;make -i -f Makefile.kalender kalender) # kalender" > > remove
  mv remove $ORACLE_HOME/net/applmake
fi
echo ""
echo "Do you want to execute Makefile.kalender"
echo "to include possible installed SQL*Net Dirvers ? [N]: \c"
read a;
if [ x$a = x ]
then
 a=N
fi
if [ "$a" = "Y" -o "$a" = "y" ]
then
 sh make -f Makefile.kalender
fi
```

7.7 SQL * Star Administration

A database link will only allow public or private connection to **one** user account on the remote database system. To make the maintenance and administration reasonable, we would like to suggest a special net user (say *netuser*) created on the remote database.

The database administrator on the remote system may then control the capabilities of the external users by controlling password and grants for the netuser.

7.8 Query to an Oracle 5 database with database links

While migrating to an Oracle 6 database you may want to acces data from an Oracle 5 database. In this case tables that are only accessed for query purpose may be left in the Oracle 5 database and accessed using database links (only with the TCP/IP driver).

The example below creates synomyms for tables residing on an Oracle 5 database using a database link to the special net user, *netuser*. The two databases reside on the same machine, using the *loopback* host name.

```
create database link oracle5
connect to netuser
identified by netpswd
using 'T:loopback:D'

create public synonym emp  for scott.emp@oracle5
create public synonym dept for scott.emp@oracle5
```

Queries may now be issued for the tables *emp* and *dept* using the existing applications.

7.9 Tuning use of database links

Using the database link option, you may form a query like this in say Copenhagen:

```
select info
from atab@Antwerp a, btab@Basel b, ctab c
where c.no = a.no
  and c.no = b.no
```

This select will force Oracle to transmit a substantial amount of information to Copenhagen. Maybe a pre join of the tables from Antwerp and Basel would reduce the amount of information with a magnitude, in this case you should consider to create a view in Antwerp like this:

```
create view pre_join as
select no, info
from atab a, btab@Basel b
where a.no = b.no
```

and then to use this view from Copenhagen:

```
select info
from pre_join@Antwerp a, ctab c
where a.no = c.no
```

When using database links Oracle will try to reduce the amount of communication cannels to one per database referenced in a select statement. Selecting from several tables on a remote database will only result in one connection. This makes it possible to do complex selects from remote database using database links and having only one remote connection.

There are however limitations in using views defined on the remote database that reference back tables on the local database, consider the following view defined in Caracas:

```
create view loopback as
select no, info
from ctab@Copenhagen
```

and then to use this view from Copenhagen:

```
select info
from atab_local a, loopback@Caracas l
where a.no = l.no
```

This will force Oracle to establish two connections between Copenhagen and Caracas, one as a result of the initial select and one as a result of using the view.

It is highly recommentet that the use of @ *<location>* in table expressions in SQL-statements of production application should be avoided. Instead a synonym should be formed to adress the remote table, hereby incresing the portability of the application.

8. SQL*Menu

This is a brief description of some special features in the current version of Sql*Menu.

8.1 FunctionKeys.

The function key F12 described as "Change special window" in the Help window, is described in the SQL*Menu Users Guide Version 4.1 as "WHERE Option: Finding Menu Location".

8.2 InvokingSQL*Plus.

SQL*plus can be invoked by command type 5 without a new database logon and without loading SQL*PLus in memory. When running SQL*Plus in batch with a command file like:

```
SQLPLUS &UN/&PW @JOB
```

you shold be aware that in SQL*Plus Version 3.0 EXIT allows you to specify a return code. This code is read by SQL*Menu, and any specified code will make Sqlmenu to pause until press of RETURN from the user.

9. SQL*ReportWriter 1.0

9.1 Parameters in SRW

Parameters in SRW (SQL*ReportWriter) should be given the datatype of the the corresponding search column - otherwise Oracle is unable to use an index on that column.

SRW has no build in functions for checking user input such as *Parameters*. The datatype and format of a parameter, as specified in the *Parameter Screen* of SRW, provides only a limited check of the user input. A more effective method is suggested in this hint.

9.1.1 Parameter Checking i SRW

The first query in this report is written solely for the purpose of checking the user input parameter. The user is supposed to supply a *department number* and the report then select's the department name and location (from the **dept** table) and in a third query, the employess working in that department (from the **emp** table).

```
 Action   Query   Group   Field   Summary   Text   Report   Parameter   Help
------------------------------ Query Settings ------------------------------
--------------------------------------------------------------------------------
| Query Name: q_0                                                Query  1 of 5  |
---------------------------- SELECT Statement -----------------------------------
|^| SELECT ' ' tjek                                                              |
|-| FROM dual                                                                    |
| | WHERE EXISTS                                                                 |
| | (SELECT 'x' FROM emp WHERE deptno=:Department)                               |
|-|                                                                              |
|v|                                                                              |
--------------------------------------------------------------------------------
```

The **q_1** query, which select's from the **dept** table, is written in a pseudo join with the first query - the join column is a constant, namely the **tjek** field selected in both queries. The purpose of this construction is to make sure that the **q_1** query will not be executed if **q_0** fails :

```
 Action   Query   Group   Field   Summary   Text   Report   Parameter   Help
------------------------------ Query Settings ------------------------------
--------------------------------------------------------------------------------
| Query Name: q_1                                                Query  2 of 5  |
---------------------------- SELECT Statement -----------------------------------
|^| SELECT dname, loc, deptno, ' ' tjek                                          |
|-| FROM dept                                                                    |
| | WHERE deptno=:Department                                                     |
|-|                                                                              |
|v|                                                                              |
--------------------------------------------------------------------------------
------------------------- Parent-Child Relationships ---------------------------
| Parent Query 1: q_0                  Parent Query 2:                           |
--------------------------------------------------------------------------------
| |     Child Columns      |    Parent 1 Columns    |    Parent 2 Columns     |
--------------------------------------------------------------------------------
|^| TJEK                   | TJEK                   |                         |
|v|                        |                        |                         |
--------------------------------------------------------------------------------
```

The **q_2** query is a normal join of the **emp** table with the **dept** of the **q_1** query. This query

will not be executed if **q_1** fails :

```
 Action   Query   Group   Field   Summary   Text   Report   Parameter   Help
------------------------------ Query Settings -------------------------------
--------------------------------------------------------------------------------
| Query Name: q_2                                                Query  3 of 5  |
------------------------------ SELECT Statement ------------------------------
|^| SELECT ename,sal,job,hiredate,deptno FROM emp                              |
|-|                                                                            |
| |                                                                            |
|-|                                                                            |
|v|                                                                            |
--------------------------------------------------------------------------------
------------------------- Parent-Child Relationships ---------------------------
| Parent Query 1: q_1                    Parent Query 2:                       |
--------------------------------------------------------------------------------
| |     Child Columns      |    Parent 1 Columns     |    Parent 2 Columns     |
--------------------------------------------------------------------------------
|^| DEPTNO                 | DEPTNO                  |                         |
|v|                        |                         |                         |
--------------------------------------------------------------------------------
```

Another part of the report might contain queries, that are not directly related to the **dept** and **emp** tables, such as the **q_3** query below, which selects a list of customers. To avoid the printout of this query if the others fail, this query is also joined to the *q_0* query using the **tjek** pseudo column.

```
 Action   Query   Group   Field   Summary   Text   Report   Parameter   Help
------------------------------ Query Settings -------------------------------
--------------------------------------------------------------------------------
| Query Name: q_3                                                Query  4 of 5  |
------------------------------ SELECT Statement ------------------------------
|^| SELECT name, address, city, '' tjek                                        |
|-| FROM customer                                                              |
| |                                                                            |
|-|                                                                            |
|v|                                                                            |
--------------------------------------------------------------------------------
------------------------- Parent-Child Relationships ---------------------------
| Parent Query 1: q_0                    Parent Query 2:                       |
--------------------------------------------------------------------------------
| |     Child Columns      |    Parent 1 Columns     |    Parent 2 Columns     |
--------------------------------------------------------------------------------
|^| TJEK                   | TJEK                    |                         |
|v|                        |                         |                         |
--------------------------------------------------------------------------------
```

The last query in the report writes out an error message if the report fails - that is if the **q_0** parameter check fails. This query is not joined to any other query since it should always be executed. If it fails it will not print anything.

```
 Action   Query   Group   Field   Summary   Text   Report   Parameter   Help
------------------------------ Query Settings -------------------------------
------------------------------------------------------------------------------
| Query Name: q_FAIL                                              Query  5 of 5 |
------------------------------ SELECT Statement ------------------------------
|^| SELECT 'Department ' || :Department ||                                     |
|-|  ' does not have any employees.' errmessage                                |
|  | FROM dual                                                                  |
|  | WHERE NOT EXISTS (SELECT 'x' FROM emp WHERE deptno = :Department)          |
|  |                                                                            |
|  |                                                                            |
|-|                                                                            |
|v|                                                                            |
------------------------------------------------------------------------------
```

If the report fails due to a bad department number supplied by the user the error message :

Report Error : Department 40 does not have any employees.

will be printed as the only output from the report.

The *Group Settings* for should be as follows :

```
 Action   Query   Group   Field   Summary   Text   Report   Parameter   Help
------------------------------ Group Settings ------------------------- 2 of 3

-------------------------------------------------------------------------------
| | |                    ||Relative | Lines  | Spaces |      Spacing      | Fields |
| | | Group Name         ||Position | Before | Before | Record   Field    | Across |
-------------------------------------------------------------------------------
|^| G_0                  ||         |        |        |                   |        |
|-| G_1                  || Below   |        |        |                   |        |
|  | G_2                  || Below   |   2    |        |                   |        |
|  | G_3                  || Below   |   2    |        |                   |        |
|  | G_FAIL               || Below   |        |        |                   |        |
|-|                      ||         |        |        |                   |        |
|v|                      ||         |        |        |                   |        |
-------------------------------------------------------------------------------
```

The *Relative Position* of group **G_1** must be *Below* to avoid printout of the **G_1** column headings when **q_0** fails.

Note that the *Label Position* and *G_FAIL* should be *left* as shown below.

```
 Action   Query   Group   Field   Summary   Text   Report   Parameter   Help
------------------------------ Group Settings ------------------------- 3 of 3

-------------------------------------------------------------------------------
| | |                    ||Multi- | Label    |          Highlight              |
| | | Group Name         ||Panel  | Position |   Field            Label        |
-------------------------------------------------------------------------------
|^| G_0                  ||       |          |                 |                 |
|-| G_1                  ||       |          |                 |                 |
|  | G_2                  ||       |          |                 |                 |
|  | G_3                  ||       |          |                 |                 |
|  | G_FAIL               ||       | Left     |                 |                 |
|-|                      ||       |          |                 |                 |
|v|                      ||       |          |                 |                 |
-------------------------------------------------------------------------------
```

The *field settings* are shown below - note that the label of the field **tjek** has been removed.

Action Query Group ***Field*** Summary Text Report Parameter Help
---------- Field Settings ---------- 1 of 3

	Field Name	Source Column	Group	Label
^	TJEK	q_0.TJEK	G_0	
-	DNAME	DNAME	G_1	Dname
	LOC	LOC	G_1	Loc
	DEPTNO	q_1.DEPTNO	G_1	Deptno
	TJEK2	q_1.TJEK	G_1	Tjek
	ENAME	ENAME	G_2	Ename
	SAL	SAL	G_2	Sal
	JOB	JOB	G_2	Job
	HIREDATE	HIREDATE	G_2	Hiredate
	DEPTNO2	q_2.DEPTNO	G_2	Deptno
	NAME	NAME	G_3	Name
	ADDRESS	ADDRESS	G_3	Address
	CITY	CITY	G_3	City
	TJEK3	q_3.TJEK	G_3	Tjek
	ERRMESSAGE	ERRMESSAGE	G_FAIL	***Report Error :***
-				
v				

Action Query Group Field Summary Text Report Parameter Help
---------- Field Settings ---------- 2 of 3

	Field Name	Data Type	Field Width	Display Format	Relative Position	Lines Before	Spaces Before
^	TJEK	CHAR	*1*				
-	DNAME	CHAR	14				
	LOC	CHAR	13				
	DEPTNO	NUM	2				
	TJEK2	CHAR	*1*				
	ENAME	CHAR	10				
	SAL	NUM	7				
	JOB	CHAR	9				
	HIREDATE	DATE	9				
	DEPTNO2	NUM	2				
	NAME	CHAR	*15*				
	ADDRESS	CHAR	*15*				
	CITY	CHAR	*15*				
	TJEK3	CHAR	*1*				
	ERRMESSAGE	CHAR	*60*				
-							
v							

```
 Action   Query   Group   Field   Summary   Text   Report   Parameter   Help
-------------------------------  Field Settings  ------------------------ 3 of 3

-------------------------------------------------------------------------------
|  |  Field Name          ||Align | Skip | Repeat | Computed Value            |
|  |                      ||      |      |        | Function      Reset Group |
-------------------------------------------------------------------------------
|^ | TJEK                 ||      |      |        |                           |
|- | DNAME                ||      |      |        |                           |
|  | LOC                  ||      |      |        |                           |
|  | DEPTNO               ||      |      |        |                           |
|  | TJEK2                ||      |  X   |        |                           |
|  | ENAME                ||      |      |        |                           |
|  | SAL                  ||      |      |        |                           |
|  | JOB                  ||      |      |        |                           |
|  | HIREDATE             ||      |      |        |                           |
|  | DEPTNO2              ||      |  X   |        |                           |
|  | NAME                 ||      |      |        |                           |
|  | ADDRESS              ||      |      |        |                           |
|  | CITY                 ||      |      |        |                           |
|  | TJEK3                ||      |  X   |        |                           |
|  | ERRMESSAGE           ||      |      |        |                           |
|- |                      ||      |      |        |                           |
|v |                      ||      |      |        |                           |
-------------------------------------------------------------------------------
```

9.1.2 SRW Parameters using Tables

Another solution to the problem of parameter checking in SQL*ReportWriter is to use parameter tables, thus totally avoiding the parameters of SRW.

If reports are started from within a SQL*Forms application, all the necessary checking of parameters can be done using triggers. When the parameters have been validated they can be stored in a *parameter* table.

Information on how to use such a parameter table from within SQL*ReportWriter can be found elsewhere in these hints, in the article about *Parameters in Views*.

To skip the runtime parameter screen of SQL*ReportWriter when executing the report the following syntax should be used :

runrep reportname uid/pwd PARAMFORM=NO

9.2 Suppressing Column Headings in SRW

When joining two queries in SRW , e.g. the **dept** and **emp** table, to create a Master/Detail report, the column headings will appear once for each master record. So the column headings for the detail query will appear even when no detail records are found :

```
Deptno 30                          Ename      Job          Sal
Dname SALES                        ---------- ---------  -------
Loc CHICAGO                        ALLEN      SALESMAN      1600
                                   WARD       SALESMAN      1250
                                   MARTIN     SALESMAN      1250
                                   BLAKE      MANAGER       2850
                                   TURNER     SALESMAN      1500
                                   JAMES      CLERK          950

Deptno 40                          Ename      Job          Sal
Dname OPERATIONS                   ---------- ---------  -------
Loc BOSTON
```

The topic of this hint is to demonstrate how to work around this situation.

First you enter the following two queries :

```
 Action   Query   Group   Field   Summary   Text   Report   Parameter   Help
------------------------------  Query Settings  -------------------------------
--------------------------------------------------------------------------------
| Query Name: Q_DEPT                                              Query  1 of 2  |
---------------------------------- SELECT Statement -----------------------------
|^|  SELECT deptno, dname, loc                                                   |
|-|  FROM  dept                                                                 |
|  |  ORDER BY deptno                                                          |
|-|                                                                             |
|v|                                                                             |
--------------------------------------------------------------------------------
```

```
 Action   Query   Group   Field   Summary   Text   Report   Parameter   Help
------------------------------  Query Settings  -------------------------------
--------------------------------------------------------------------------------
| Query Name: Q_EMP                                               Query  2 of 2  |
---------------------------------- SELECT Statement -----------------------------
|^|  SELECT NULL, deptno, ename, job, sal                                        |
|-|  FROM  emp                                                                  |
|  |                                                                            |
|-|                                                                             |
|v|                                                                             |
--------------------------------------------------------------------------------
--------------------------- Parent-Child Relationships --------------------------
| Parent Query 1: Q_DEPT                  Parent Query 2:                        |
--------------------------------------------------------------------------------
| |     Child Columns      |    Parent 1 Columns     |    Parent 2 Columns     |
--------------------------------------------------------------------------------
|^|  DEPTNO                |  DEPTNO                 |                         |
|v|                        |                         |                         |
--------------------------------------------------------------------------------
```

A new Group called **G_NULL** is created and the proper *Group Settings* for the **G_DEPT** is specified:

```
Action    Query    Group    Field    Summary    Text    Report    Parameter    Help
------------------------------ Group Settings ------------------------ 1 of 3
```

	Group Name	Query	Print Direction	Matrix Group	Page Break
^	G_DEPT	Q_DEPT	Down		
-	*G_NULL*	*Q_EMP*	Down		
	G_EMP	Q_EMP	Down		
- v					

```
Action    Query    Group    Field    Summary    Text    Report    Parameter    Help
------------------------------ Group Settings ------------------------ 2 of 3
```

	Group Name	Relative Position	Lines Before	Spaces Before	Spacing Record	Spacing Field	Fields Across
^	G_DEPT				*2*		*1*
-	G_NULL						
	G_EMP						
- v							

```
Action    Query    Group    Field    Summary    Text    Report    Parameter    Help
------------------------------ Group Settings ------------------------ 3 of 3
```

	Group Name	Multi-Panel	Label Position	Highlight Field	Highlight Label
^	G_DEPT		*Left*		
-	G_NULL				
	G_EMP				
- v					

The *Field Settings* should be as shown below. The **NULL** field is associated with the **G_NULL** group and the *Label* for this field is deleted.

```
Action    Query    Group    Field    Summary    Text    Report    Parameter    Help
------------------------------ Field Settings ------------------------ 1 of 3
```

	Field Name	Source Column	Group	Label
^	DEPTNO	Q_DEPT.DEPTNO	G_DEPT	Deptno
-	DNAME	DNAME	G_DEPT	Dname
	LOC	LOC	G_DEPT	Loc
	NULL	NULL	*G_NULL*	
	DEPTNO2	Q_EMP.DEPTNO	G_EMP	Deptno
	ENAME	ENAME	G_EMP	Ename
	JOB	JOB	G_EMP	Job
	SAL	SAL	G_EMP	Sal
- v				

Suppress printing of the **DEPTNO2** field :

```
 Action   Query   Group   Field   Summary   Text   Report   Parameter   Help
------------------------------- Field Settings ------------------------ 3 of 3

-------------------------------------------------------------------------------
| |                   ||      |       |          |       Computed Value       |
| | Field Name        ||Align | Skip  | Repeat   | Function      Reset Group  |
-------------------------------------------------------------------------------
|^| DEPTNO            ||      |       |          |          |                 |
|-| DNAME             ||      |       |          |          |                 |
| | LOC               ||      |       |          |          |                 |
| | NULL              ||      |       |          |          |                 |
| | DEPTNO2           ||      |   X   |          |          |                 |
| | ENAME             ||      |       |          |          |                 |
| | JOB               ||      |       |          |          |                 |
| | SAL               ||      |       |          |          |                 |
|-|                   ||      |       |          |          |                 |
|v|                   ||      |       |          |          |                 |
-------------------------------------------------------------------------------
```

The last thing to do is to wpecify the *Frequency* of the **G_EMP Column Heading** so that it is printed whenever the **G_NULL** group is printed :

```
 Action   Query   Group   Field   Summary   Text   Report   Parameter   Help
------------------------------- Text Settings -------------------------------

-------------------------------------------------------------------------------
| Object:  G_EMP                    Type:  Column Heading  | Status:  Default |
-------------------------------------------------------------------------------
| Relative Position:              | Repeat On Page Overflow:  X               |
|      Lines Before:  0           |          Justification:  Left             |
|     Spaces Before:  0           |              Frequency:  G_NULL           |
-------------------------------------------------------------------------------

----------------------------------- Text ------------------------------------
| Panel Number:  1                                      Panels Defined:  1   |
-------------------------------------------------------------------------------
|^| Ename        Job          Sal                                            |
|-| ----------   ---------   -------                                         |
| |                                                                          |
|-|                                                                          |
|v|                                                                          |
-------------------------------------------------------------------------------
```

If no record are selected by the **Q_EMP** query, printing the **G_NULL** will fail. Since the printing of the column headings for the rest of the **Q_EMP** fields has been associated with the success or failure of the **G_NULL** group, this will prevent them from being printed.

```
:
:
:

Deptno 30                     Ename        Job          Sal
Dname SALES                   ----------   ---------  -------
Loc CHICAGO                   ALLEN        SALESMAN     1600
                              WARD         SALESMAN     1250
                              MARTIN       SALESMAN     1250
                              BLAKE        MANAGER      2850
                              TURNER       SALESMAN     1500
                              JAMES        CLERK         950


Deptno 40
Dname OPERATIONS
Loc BOSTON
```

The space between the **G_DEPT** group and the **G_EMP** group in the finished report, could be narrowed down by changing the *Field Width* of the **NULL** field.

9.3 Database Values in Page Headings

You cannot reference any selected fields in *Page Headers* or *Page Footers*. If you need to do this anyway you have to create your own headers and footers, ignoring the standard ones supplied by SQL*ReportWriter.

In this example we want a list of orders issued by different customers and we want the customer number, name, address and city printed on top of each page in the report.

The query for this report is shown below :

```
 Action   Query   Group   Field   Summary   Text   Report   Parameter   Help
------------------------------ Query Settings ------------------------------
--------------------------------------------------------------------------------
| Query Name: Q_CUST                                              Query  1 of 1  |
------------------------------ SELECT Statement -------------------------------
|^| SELECT customer.custid, name, address, city,                               |
|-|        ord.ordid, itemid, item.prodid, descrip,                            |
|  |        actualprice, qty, itemtot                                           |
|  | FROM  customer, ord, item, product                                         |
|  | WHERE customer.custid = ord.custid                                         |
|  | AND   ord.ordid = item.ordid                                               |
|-| AND   item.prodid = product.prodid                                         |
|v| ORDER BY customer.custid, ord.ordid, itemid                                 |
--------------------------------------------------------------------------------
```

Next, a new group is created above the **G_CUST** group associated with the query. You should specify *Page Break* **Always** for this group and *Label Position* **Left**. The *Relative Position* of the **G_CUST** is set to **Below** and *Lines Before* to **2**.

```
 Action   Query   Group   Field   Summary   Text   Report   Parameter   Help
------------------------------ Group Settings ------------------------- 1 of 3

--------------------------------------------------------------------------------
| |  Group Name  |  Query  |  Print Direction  |  Matrix Group  |  Page Break  |
--------------------------------------------------------------------------------
|^| G_HEAD       | Q_CUST  | Down              |                | Always       |
|-| G_CUST       | Q_CUST  | Down              |                |              |
|  |              |         |                   |                |              |
|-|              |         |                   |                |              |
|v|              |         |                   |                |              |
--------------------------------------------------------------------------------
```

```
Action   Query   Group   Field   Summary   Text   Report   Parameter   Help
------------------------------  Group Settings  ------------------------ 2 of 3

---------------------------------------------------------------------------
| | Group Name | Relative Position | Lines Before | Spaces Before | Spacing Record | Spacing Field | Fields Across |
---------------------------------------------------------------------------
|^| G_HEAD     |                   |              |               |                |               |               |
|-| G_CUST     | Below             | 2            |               |                |               |               |
| |            |                   |              |               |                |               |               |
|-|            |                   |              |               |                |               |               |
|v|            |                   |              |               |                |               |               |
---------------------------------------------------------------------------
```

```
Action   Query   Group   Field   Summary   Text   Report   Parameter   Help
------------------------------  Group Settings  ------------------------ 3 of 3

-------------------------------------------------------------------
| | Group Name | Multi-Panel | Label Position | Highlight Field | Highlight Label |
-------------------------------------------------------------------
|^| G_HEAD     |             | Left           |                 |                 |
|-| G_CUST     |             |                |                 |                 |
| |            |             |                |                 |                 |
|-|            |             |                |                 |                 |
|v|            |             |                |                 |                 |
-------------------------------------------------------------------
```

In the *Field Settings* screen, the *custid, name, address* and *city* fields should be associted with the **G_HEAD** group. Also, the sizes of some of the fields are adjusted to make the report fit within the current pagewidth (80 characters).

```
Action   Query   Group   Field   Summary   Text   Report   Parameter   Help
------------------------------  Field Settings  ------------------------ 1 of 3

-----------------------------------------------------------------------
| | Field Name | Source Column | Group | Label |
-----------------------------------------------------------------------
|^| CUSTID     | CUSTID        | G_HEAD | Custid |
|-| NAME       | NAME          | G_HEAD | Name |
| | ADDRESS    | ADDRESS       | G_HEAD | Address |
| | CITY       | CITY          | G_HEAD | City |
| | ORDID      | ORDID         | G_CUST | Ordid |
| | ITEMID     | ITEMID        | G_CUST | Itemid |
| | PRODID     | PRODID        | G_CUST | Prodid |
| | DESCRIP    | DESCRIP       | G_CUST | Descrip |
| | ACTUALPRICE| ACTUALPRICE   | G_CUST | Actualprice |
| | QTY        | QTY           | G_CUST | Qty |
| | ITEMTOT    | ITEMTOT       | G_CUST | Itemtot |
|-|            |               |        |         |
|v|            |               |        |         |
-----------------------------------------------------------------------
```

```
 Action   Query   Group   Field   Summary   Text   Report   Parameter   Help
------------------------------ Field Settings ------------------------ 2 of 3

-------------------------------------------------------------------------------
| | |  Field Name  | Data | Field | Display | Relative | Lines | Spaces |
| | |              | Type | Width | Format  | Position | Before| Before |
-------------------------------------------------------------------------------
|^| | CUSTID       | NUM  |  4    |         |          |       |        |
|-| | NAME         | CHAR |  15   |         |          |       |        |
| | | ADDRESS      | CHAR |  15   |         |          |       |        |
| | | CITY         | CHAR |  15   |         |          |       |        |
| | | ORDID        | NUM  |  3    |         |          |       |        |
| | | ITEMID       | NUM  |  3    |         |          |       |        |
| | | PRODID       | NUM  |  6    |         |          |       |        |
| | | DESCRIP      | CHAR |  20   |         |          |       |        |
| | | ACTUALPRICE  | NUM  |  8    |         |          |       |        |
| | | QTY          | NUM  |  4    |         |          |       |        |
| | | ITEMTOT      | NUM  |  8    |         |          |       |        |
|-| |              |      |       |         |          |       |        |
|v| |              |      |       |         |          |       |        |
-------------------------------------------------------------------------------
```

When the report is executed, the **G_HEAD** group will print as a header on each page. Setting the *Page Hight* to **15** will demonstrate this. The **G_HEAD** body text will print on every page because of the default *Repeat on Page overflow :* **X** for this object :

Page 1:

```
Custid 102  Name VOLLYRITE        Address 9722 HAMILTON    City BURLINGAME

  Ordid  Itemid  Prodid  Descrip              Actualprice   Qty   Itemtot
  -----  ------  ------  -------------------  -----------  ----  --------
    602       1  100870  ACE TENNIS BALLS-3 P          2.8    20        56
    603       2  100860  ACE TENNIS RACKET I            56     4       224
    611       1  100861  ACE TENNIS RACKET II           45     1        45
    614       1  100860  ACE TENNIS RACKET I            35   444     15540
    614       2  100870  ACE TENNIS BALLS-3 P          2.8  1000      2800
    614       3  100871  ACE TENNIS BALLS-6 P          5.6  1000      5600
```

Page 2:

```
Custid 102  Name VOLLYRITE        Address 9722 HAMILTON    City BURLINGAME

  Ordid  Itemid  Prodid  Descrip              Actualprice   Qty   Itemtot
  -----  ------  ------  -------------------  -----------  ----  --------
    618       1  100860  ACE TENNIS RACKET I            35    23       805
    618       2  100861  ACE TENNIS RACKET II        45.11    50    2255.5
    618       3  100870  ACE TENNIS BALLS-3 P           45    10       450
```

10. Hints on Migration

Migration, the process of transforming your running Oracle version 5 database into a similar version 6 system, is described in this hint as a number of activities, that the Database Administrator could perform :

- Pre-Migration Activities.
- Exporting Oracle 5 Data.
- Creation of the Oracle 6 Database.
- Preparing for the Oracle 6 Import.
- Migrating SQL Scripts.
- Migrating Pro*C Programs.
- Migrating SQL*Forms.
- Changes in Rpt/Rpf.

10.1 Pre-Migration Activities

10.1.1 Migrating Users

All users on Oracle 5 has to be recreated on Oracle 6. The command file shown below will generate *grants* for all users on Oracle 5, with a password that is equal to the username :

```
set heading off
set feedback off
set pagesize 1000
set pause off
spool user.lst
column disk_space newline
column alter_space newline
SELECT 'GRANT '
       || DECODE(usr$connect,'Y', 'CONNECT, ',NULL)
       || DECODE(usr$dba,'Y','DBA, ',NULL)
       || DECODE(usr$resource,'Y','RESOURCE ',NULL)
       || 'TO ' || usr$name
       || ' IDENTIFIED BY ' || usr$name || ';',
      DECODE(usr$resource,'Y',
      'REM GRANT RESOURCE (x_bytes) ON tablespace TO '
      || usr$name || ';',NULL) disk_space,
      'REM ALTER USER ' || usr$name ||
      ' DEFAULT TABLESPACE tablespace ' alter_space ,
      'REM TEMPORARY TABLESPACE tablespace; ' alter_space
FROM sys.userauth
WHERE usr$name NOT IN ('SYS','SYSTEM','PUBLIC')
AND   usr$name IS NOT NULL
ORDER BY usr$name;
spool off
HOST sed '/, TO /s// TO /g' user.lst > user.sql
set heading on
set feedback on
set pagesize 18
set pause on
```

The command file is put in the Unix file *user.sql*.

10.1.2 New Version 6.0 Naming-Conventions

All *User Objects* - tables, clusters, synonyms and indexes, must have distinct names under Oracle 6. In version 5.1 it is possible to create indexes with names already used for other objects. The SQL script below will, when executed by the SYSTEM user, check all users on the database, and find these collisions of names. The script will generate a command file, *change.sql*, that DROP's these indexes and recreates them with a different name :

```
SET PAUSE OFF
SET HEADING OFF
SET PAGESIZE 80
SET TERMOUT OFF
SET FEEDBACK OFF
COLUMN idx$compress  NOPRINT OLD_VALUE comp
COLUMN usr$name  NOPRINT
COLUMN idx$name  NOPRINT
COLUMN command FORMAT a79
COLUMN ccmd FORMAT a79
COLUMN dcmd FORMAT a79
COLUMN cnames FORMAT a32 NEWLINE
TTITLE OFF
BTITLE LEFT ') 'comp'  ;'
BREAK ON idx$name SKIP PAGE ON ccmd ON dcmd ON command
SPOOL idx.lst
SELECT usr$name, idx$name , idx$compress, 'CONNECT ' ||usr$name ccmd,
       'DROP INDEX ' || idx$name ||';' dcmd, 'CREATE ' ||
       DECODE(idx$unique,'NON UNIQUE',NULL,idx$unique) ||
       ' INDEX  ' || SUBSTR(idx$name,1,25) || '_6_NEW ON ' ||
       b.tab$name || '( ' command , DECODE(idx$colseq,1,' ',',') ||
       idx$column || ' ' || idx$order  cnames
FROM   sys.indexes, sys.userauth, sys.tables a, sys.tables b
WHERE  idx$owner = usr$uid   AND    idx$name = a.tab$name
AND    a.tab$owner = usr$uid AND    b.tab$owner = usr$uid
AND    idx$pid=b.tab$pid     AND    idx$rba=b.tab$rba
AND    idx$tbl=b.tab$tbl     AND    usr$name NOT IN ('SYS')
ORDER BY b.tab$owner,b.tab$name,idx$name,idx$colseq;
SPOOL OFF
HOST sed -e '/^$/d' -e '/^ *$/d' idx.lst > change.sql
SET PAUSE ON
SET HEADING ON
SET PAGESIZE 20
SET FEEDBACK ON
SET TERMOUT ON
```

10.1.3 Version 6.0 Reserved Words and ANSI Keywords

There are no new *Reserved Words* in Oracle 6.0.

There are a number of new ANSI *Keywords*, which are not currently *Reserved Words*, but are likely to become so in later versions.

You do not have to rename identifiers that are using version 6.0 ANSI Keywords, but you could do so in order to prepare for later Oracle versions.

Both the Oracle *Reserved Words* and the ANSI *Keywords* are listed in chapter 3 of the *Oracle SQL Language Guide*.

10.1.4 Oracle 5 Backup

If the running version 5.1 database can co-exist with the new version 6 database, this is to be preferred. In this way data and systems can be migrated gradually.

Before starting the migration from version 5 to version 6 you should backup the running database using the **exp** program (Entire export) and taking raw copies of the database and before-image disks with :

```
dskback -c /dev/dsk/dbs_disk /dev/stream
                    :
                    :
dskback -c /dev/dsk/bi_disk  /dev/stream
```

10.2 Exporting the Oracle 5 Database

Moving data from version 5 to version 6 is done using the export program on a USER basis, exporting the data for later import into the Oracle 6 database.

10.2.1 Create list of Users

```
# Create User list
sqlplus system/$PASSWD << STOP
set heading off
set feedback off
set pagesize 1000
set pause off
spool users.lst
SELECT usr$name
FROM sys.userauth
WHERE usr$name NOT IN ('SYS','SYSTEM','PUBLIC')
  AND usr$name IS NOT NULL
ORDER BY usr$name ;
spool off
host sed -e '/^$/d' -e 's/  *$//' users.lst > Users.sql
set heading on
set feedback on
set pagesize 18
set pause on
exit

STOP
```

10.2.2 Creating Grant Command File per User

```
# Creating grant command file per user
mkdir grantdir
for USER in 'cat Users.sql | sed 's/\.//''
do
echo $USER
sqlplus system/$PASSWD 1>/dev/null 2>&1 <<STOP
start grants
$USER
$USER
$USER
$USER
$USER
$USER
STOP
FIL='echo $USER | sed 's/\\$/-/''
tr -d "\012" < Grants.sql | sed 's/,  / /' > grantdir/$FIL.sql
echo "exit" >> grantdir/$FIL.sql
done
```

10.2.3 The Get Grants Script

```
# The grants.sql Script
set termout off
set heading off
set verify off
set feedback off
set echo off
set pause off
set pagesize 150
column a format a90
column b format a30
spool Grants
SELECT 'grant alter on "' || tau$tname || '" to "'|| tau$grantee ||'"' a,
       DECODE(tau$alter,'G',' with grant option ',null)||';' b
FROM sys.tabauth
WHERE tau$grantor NOT IN ('SYS','SYSTEM')
  AND tau$grantee NOT IN ('SYS','SYSTEM')
  AND tau$grantor != tau$grantee
  AND tau$grantor = '&1' AND tau$alter != ' ' ;
SELECT 'grant delete on "' || tau$tname || '" to "'|| tau$grantee ||'"' a,
       DECODE(tau$delete,'G',' with grant option ',null)||';' b
FROM sys.tabauth
WHERE tau$grantor NOT IN ('SYS','SYSTEM')
  AND tau$grantee NOT IN ('SYS','SYSTEM')
  AND tau$grantor != tau$grantee
  AND tau$grantor = '&1' AND tau$delete != ' ' ;
SELECT 'grant index on "' || tau$tname || '" to "'|| tau$grantee ||'"' a,
       DECODE(tau$index,'G',' with grant option ',null)||';' b
FROM sys.tabauth
WHERE tau$grantor NOT IN ('SYS','SYSTEM')
  AND tau$grantee NOT IN ('SYS','SYSTEM')
```

```
  AND tau$grantor != tau$grantee
  AND tau$grantor = '&1' AND tau$index != ' ' ;
SELECT 'grant insert on "' || tau$tname || '" to "'|| tau$grantee ||'"' a,
       DECODE(tau$insert,'G',' with grant option ',null)||';' b
FROM sys.tabauth
WHERE tau$grantor NOT IN ('SYS','SYSTEM')
  AND tau$grantee NOT IN ('SYS','SYSTEM')
  AND tau$grantor != tau$grantee
  AND tau$grantor = '&1' AND tau$insert != ' ' ;
SELECT 'grant select on "' || tau$tname || '" to "'|| tau$grantee ||'"' a,
       DECODE(tau$select,'G',' with grant option ',null)||';' b
FROM sys.tabauth
WHERE tau$grantor NOT IN ('SYS','SYSTEM')
  AND tau$grantee NOT IN ('SYS','SYSTEM')
  AND tau$grantor != tau$grantee
  AND tau$grantor = '&1' AND tau$select != ' ' ;
SELECT 'grant update on "' || tau$tname || '" to "'|| tau$grantee ||'"' a,
       DECODE(tau$update,'G',' with grant option ',null)||';' b
FROM sys.tabauth
WHERE tau$grantor NOT IN ('SYS','SYSTEM')
  AND tau$grantee NOT IN ('SYS','SYSTEM')
  AND tau$grantor != tau$grantee
  AND tau$grantor = '&1' AND tau$update != ' ' ;
spool off
host sed -e '/^$/d' -e '/^ *$/d' Grants.lst > Grants.stl
host sed -e '/Enter val/d' -e 's/  */ /g' Grants.stl > Grants.sql
set heading on
set echo on
set pause on
set pagesize 20
set termout on
set feedback on
```

10.2.4 Export The Users

Each of the users from the userlist are now to be exported without grants.

```
Export: Version 5.1.17.4 - Production on Wed Nov 22 15:45:33 1989

Copyright (c) 1986, Oracle Corporation, California, USA.  All rights reserved.
             Prepared by Dansk Data Elektronik A/S for SUPEBMAX

Connected to: ORACLE V5.1.17.4 - Production

Enter array fetch buffer size(default is 20000)>
Export file: expdat.dmp > [Return]
U(sers), or T(ables): U > [Return]
Export Grants (Y/N): N > N
Export the rows (Y/N): Y > [Return]
Compress extents (Y/N): Y > [Return]
Exporting SCOTT
```

All users with the exception of *SYS* and *SYSTEM* are exported.

10.2.5 Grant's and Synonyms

Only *First-level grants* are exported. This could create problems if the users imported on version 6.0 are not imported in the correct sequence - this problem only arises when privileges have been granted in *chains*.

Grants of ALTER and INDEX on views are not permitted in version 6 and could cause problems during import.

Since the *SYSTEM* user of version 5 is **not** exported, the *public synonyms* that the Database Administrator have created under this user will not be exported. The following SQL-script will extract these *public synonyms* from the SYSTEM user :

```
set heading off
set feedback off
set pagesize 1000
set pause off
spool psyn.lst
SELECT 'CREATE PUBLIC SYNONYM ' || sname || ' FOR ' ||
        creator ||'.' || tname || ';'
FROM publicsyn
WHERE NOT EXISTS
(SELECT 'x' FROM dtab WHERE dtab.tname = publicsyn.sname)
AND sname NOT LIKE 'IAP%' AND sname != 'DUAL';
spool off
set heading on
set feedback on
set pagesize 18
set pause on
```

10.3 Creation of the Oracle 6.0 Database

Review the Oracle 6 documentation :

1. The Oracle Database Administrator's Guide.
2. the Oracle Utilities User's Guide (includes export/import)
3. The Supermax Oracle 6.0 Installation and User's Guide.
4. The Supermax Oracle 6.0 Release Bulletin.

10.3.1 Estimate the Space Usage

Check the partitions you have currently on the version 5 database and find the size of these partitions :

```
BREAK ON REPORT SKIP 2
COMPUTE SUM OF psize ON REPORT
SELECT par$name, SUM(fil$length) psize
FROM sys.partitions,sys.files
WHERE par$id = fil$pid
GROUP BY par$name;
```

The 6.0 database will use at least the same amount of space plus the necessary space for the first *Rollback Segments*.

Oracle 6.0 requires a minimum of 5 megabytes.

10.3.2 Plan the 6.0 Database

Plan the 6.0 database files :

- The database files. For each *Partition* in the version 5 database you could plan a *Tablespace* in version 6 of at least the same size.
- Redo log Files.
- Control Files.

Decide what logical disks and disk-drives these files should be placed on.

Choose the number and size of the Redo Log Files.

Choose the Block Size for the 6.0 database (the default is 4Kb).

adjust the Oracle 6.0 init.ora parameters.

10.3.3 Create the Database

Start an Oracle Instance.

Create the new Database.

Refer to the Installation Guide and the Database Administrator's Guide for information on how to do this.

10.4 Prepare Oracle 6 Import

Oracle 6.0 can be created using any of a number of different block sizes, 1Kb, 2Kb, 4Kb or 8Kb.

If the 6.0 base is created using a block size of 2Kb, which equals the version 5 blocksize, then the *Space Definitions* found in the export file could be used when importing the data on Oracle 6.

If you have a number of *Create Table Commands Containing Space Definitions* that you use in your normal day to day backup strategy, these should serve as the basis for your Oracle 6 Import.

If the Oracle 6.0 base is created with a blocksize of 4Kb it might be necessary to adjust the Oracle 5 export.

There are two way to prepare for a version 5 export on a Oracle 6 4Kb block database :

- Manual preparation of the import.
- Changing of the export-file using a program.

10.4.1 Manual Preparation of the Import

Estimating the space usage for a given users tables can be done with :

SELECT * FROM storage;

Since Oracle does not de-allocate blocks when the rows of a table is deleted, the storage currently allocated for a table might not be a true picture of the space actually needed to hold the data and indexes for that table.

Before migrating to Oracle 6 you could consider checking the actual space usage of the tables on your Oracle 5 system. You could follow this up by creating tables with correct STORAGE clauses in Oracle 6 for those tables in Oracle 5 that have incorrect *Space Definitions*. This must be done by hand. This procedure ensures an optimal use of the space available on your new Oracle 6 database.

To get a rough estimate of the space usage on your Oracle 5 database, try running the following SQL-script :

```
REM          This script originates from IOUG (International Oracle Users Group)
REM          It will give you a rough estimate of the space used to
REM          store each of a users TABLES.
REM          Please NOTE :
REM          It will not work for tables with a column of datatype LONG.
REM          The resulting block count for tables in CLUSTER's will be wrong.
REM          2 Blocks of table overhead should be added to each block count.
REM          NO FREEPCT per block is added      - this should be considered
REM        when the actual block usage is calculated.
REM
REM          .....DDE 3-11-89 cfj
REM
REM          This script   includes a   handy SQL*Plus   command file to
REM          automatically produce SQL statements  to determine  the amount of
REM          ORACLE blocks  actually occupied  by data.  The ORACLE dictionary
REM          only indicates the number of blocks  allocated to  the datapages.
REM          This command  file will  tell how many of those blocks are really
REM          occupied - including chained blocks - excluding the  2 blocks of
REM          table  overhead.  NOTE:  This  procedure will not work for tables
REM          with a column of datatype LONG.
REM
REM          The parameter, "block_size", must be set for the operating system
REM          you  are  running  ORACLE.  "Block_size" is currently set for the
REM          SUPERMAX operating system. MSDOS  is 1024,  and IBM/VMS  or CMS is
REM          4096.
REM
REM         Before executing  the command  file, you must first create a view
REM          which will output the maximum column number for each table.
REM
DROP VIEW TAB_STAT;
CREATE VIEW TAB_STAT AS
SELECT TNAME, MAX(COLNO) MAX_COLNO
FROM COL
GROUP BY TNAME;

rem **** beginning of command file ******
rem
define rec_overhead = 4
define col_overhead = 2
define block_size = 2048
define block_overhead = 76
set verify off
set arraysize 2
set maxdata 10000
set feedback off
set heading off
set termout off
set echo off
set pagesize 9999
set linesize 300
set trim on
column x_out format a78 trunc
spool temp.sql
```

```
select decode (colno,1,'select ''Estimated Space Usage for Table '||col.tname||
'decode(colno, max_colno,'+    &col_overhead,0)+&rec_overhead )/'||
col.tname||' group by substr(rowid,1,8)   ;',   '+ &col_overhead,0) +')
from col,tab_stat
where col.tname=tab_stat.tname
and col.tname in (select tname from tab where tabtype = 'TABLE')
order by col.tname,col.colno
/
spool off
set termout on
@temp
rem   ****** end of command file *********
```

10.5 Changing the Oracle 5 Export Files

If you are planing to migrate Oracle 5 data to an Oracle 6 database with the default block size of 4 kb, you should read this.

When exporting data from Oracle 5, using the **exp** program with a **U** option, a *Space Definition* is exported for each table. The number of block's specified in these *Space Definitions* refer to the 2 Kb blocks of the Oracle 5 database.

The Oracle 6 import program, when reading an Oracle 5 export file, knows how to read the Oracle 5 *Space Definitions* and translate them into useful Oracle 6 *Storage Clauses*. This translation however, does not take into account that the recommended block size on Oracle 6 is 4kb. So if you are using 4 Kb blocks on Oracle 6, an Oracle 5 INITIAL allocation of 5 * 2 Kb translates into an Oracle 6 INITIAL allocation of 5 * 4 Kb, thus producing a space usage overhead on Oracle 6.

One way of working around this problem is to change the INITIAL and INCREMENT values found in the Oracle 5 export file. Unfortunately this file is not in clear text, so an editor cannot be used.

The following two small programs should do the trick. To cope with input characters with the value '\0' you need a small C program. The output of this C program is then piped into a **lex** program that changes the INITIAL and INCREMENT values :

```
/* ch.c : program for converting 0 bytes */
#include <stdio.h>
#include <ctype.h>

main()
{
int c;
while ((c=getchar()) != EOF)
{
if (c == 0)  printf("_NULLBYTE_");
else printf("%c",c);
/* if the string _NULLBYTE_ is not present in the export */
}
}
```

The name of the lex program is patch.l :

```
%START INIT
      int k, b;
%%
INITIAL[^0-9] |
INCREMENT[^0-9]  { printf("%s",yytext); BEGIN INIT; };
<INIT>[0-9]* { k=atoi(yytext);
               b=k/2;
               printf("%d", b);
               BEGIN 0;
             }
_NULLBYTE_       { printf("%c",'\0');};
\n |
.            { printf("%s", yytext);};
%%

main()
{
      yylex();
}

yywrap(s)
char *s;
{
      printf("\n%s\n", s);
      exit(0);
}
```

The programs are compiled :

make ch
touch patch.l
make patch.c patch

and then executed :

ch <oracle_5.exp ¦ patch > oracle_5.patch

This file *,oracle_5.patch*, can now be imported into Oracle 6, creating nearly the same space allocations as on Oracle 5. The *patch* program does a division by 2 (6->3, 4->2, 5->2). A minimum of 2 blocks must be allocated for each Oracle 6 table.

The two programs *ch* and *patch* may be put in one (*PATCH*), which will also overcome the problem using INITIAL or INCREMENT as parts of table or kolumn names.

```
/*
* PATCH.C  v1.3
* Used to patch Oracle 5 space definition paramenters INITIAL and
* INCREMENT by dividing values by two.
* Usage:
*     mknod FIFO.dmp p
*     imp .... FILE=FIFO &
*     sleep 5
*     patch <expdat.dmp >FIFO.dmp
*
* SJM/DDE, August 1991.
*/

#include <stdio.h>
#include <string.h>

#define iswhite(c)(c==' ' || c=='\t' || c=='\n' || c=='\r')

int process_space(void);

static char *magic = "SPACE DEFINITION \"";

main()
{
  register int  c;
  register char *p;

  while ((c = getchar()) != EOF)
    if (c == *magic) {
      putchar(c);
      p = magic+1;
      while (*p != '\0' && (c = getchar()) != EOF && *p == c)
      putchar(c);
      if (c == EOF)
      break;
      if (*p == '\0') {
      if (process_space()) {
        fputs("unexpected EOF", stderr);
        break;
      }
      } else
      putchar(c);
```

```
    } else
      putchar(c);
  return(0);
}

process_space()
{
  register int  c, i;
  register char *p;
  int           num;
  char          buf[256],
           numbuf[16];

  while ((c = getchar()) != EOF && c != '(')     /* Find first '(' */
    putchar(c);
  putchar(c);
  if (c == EOF)
    return(1);
  i = 0;
  while ((c = getchar()) != EOF && c != ')')     /* Read "datapages" */
    buf[i++] = c;
  if (c == EOF)
    return(1);
  buf[i++] = ')';
  while ((c = getchar()) != EOF && c != ')')     /* Read "indexpages" */
    buf[i++] = c;
  if (c == EOF)
    return(1);
  buf[i++] = ')';
  buf[i] = '\0';
  for (p = buf; *p != '\0'; p++)   /* Now alter INITIAL and INCREMENT */
    if (*p == 'I')
      if (!strncmp(p, "INITIAL", 7) || !strncmp(p, "INCREMENT", 9)) {
      while (*p != '\0' && !iswhite(*p))
        p++;
      while (*p != '\0' && iswhite(*p))
        p++;
      sscanf(p, "%d", &num);
      num = num / 2 + 1;
        if (num < 2)
          num = 2;
      sprintf(numbuf, "%d", num);
      for (i = 0; i < strlen(numbuf); i++)
        *p++ = numbuf[i];
        while (*p >= '0' && *p <= '9')
          *p++ = ' ';
      }
  fwrite(buf, strlen(buf), 1, stdout);
  return(0);
}
```

10.6 SQL*Plus Scripts

- **Commit** and **Rollback** are SQL commands in Oracle 6. This means that you need to suffix them with ;(semicolon). To execute an Oracle 5 script under version 6 in sqlplus, use the command :

  SET COMPATIBILITY 5

- Script referencing Oracle 5 *Data Dictionary Tables* will not work in version 6. The version 5 System/Manager views on the Data Dictionary can be installed in version 6 :

  sqlplus @$ORACLE_HOME/dbs/catalog5.ora

- The following references are no longer legal in version 6 :

  - `CREATE PARTITION/SPACE`
  - `ALTER PARTITION/SPACE`
  - `DROP SPACE (PARTITION?)`
  - `"ON tablename" part of DROP INDEX/VALIDATE INDEX`

10.7 Pro*C

10.7.1 ROLLBACK on Statement Level

Oracle version performs ROLLBACK on statement level. This means that programs that receives the error "duplicate value in index" must handle the transaction ROLLBACK within the program.

Example :

```
WHENEVER SQLERROR GOTO error;
CREATE TABLE demotab(col1 NUMBER);
CREATE UNIQUE INDEX col1_inx ON demotab;
INSERT INTO DEMOTAB(COL1) VALUES (1)
INSERT INTO DEMOTAB(COL1) VALUES (1) -- error

error:
      COMMIT
```

In version 5 both INSERT's will be rolled back. In version 6 only the last INSERT is rolled back and the others are preserved.

The error handling in version 6 must look like the example below, if the whole transaction should be rolled back :

```
error:
        ROLLBACK
        COMMIT
```

10.7.2 Row Level Locking

When using SELECT FOR UPDATE version 5 locks the rows when FETCH'ing them, while version 6 locks when the EXEC call is issued (OPEN in Pro*C).

Since COMMIT releases locks it is no longer allowed to FETCH after a COMMIT when FETCH'ing from a SELECT FOR UPDATE cursor.

The handling of locks in version 6 stresses the need to be as restrictive as possible in the WHERE clause of a SELECT FOR UPDATE statement.

10.7.3 Other Changes

- Version 6 sorting returns NULL values sorted high - as opposed to version 5. If it is a problem it can be solved using constructions like :

  ORDER BY NVL(sort_col,' ')

- The command *INSERT INTO table SELECT ...* returns an error code if no rows are found.

- *Subqueries* returns NULL if no rows are found.

10.7.4 Childdeath, signalhandlers and system calls: signal(SIGCLD, ..)

In Oracle 6.0.26 Pro*C programs will have a signalhandler set for the signal **SIGCLD**. This may interfere with the system call *wait*, system calls using wait or other functionalities dependend on **SIGCLD**. The system call *system()*, as an example will return -1, set *errno* = 4 and *smoserr* = 118, although it performs any desired shell function.

This phenomenon may be handled in two ways. First the Pro*C program may choose to ignore the return value or identify this special case (*errno* = 4 and *smoserr* = 118). An other solution is to force the **SIGCLD** signalhandler to **SIG_DFL** before the critical system call and then reset the original handler after the system call:

```
  ...
#include <signal.h>
extern int smoserr, errno;
  ...
int res;
char *old_sig;
  ...
old_sig = (char *) signal(SIGCLD, SIG_DFL);
res=system("$SHELL -c 'echo Hello world: procid = $$0");
printf("%d=system(..., errno = %d, smoserr = %d0, res, errno, smoserr);
signal(SIGCLD, old_sig);
  ...
```

10.7.5 Delete where ROWID...

A small difference between Oracle version 5 and version 6, has been detected regarding the result of deleting a row identified by rowid, if the row does not exist.

Take the following steps:

```
select rowid
from dept
where deptno = 40
for update of deptno;

Result:
ROWID
------------------
00000374.0003.0001

delete from dept
where rowid = '374.3.1';

Result:
1 record deleted.

delete from dept
where rowid = '374.3.1';

Result:
delete from dept where rowid = '374.3.1'
                  *
ERROR at line 1:
ORA-08005: specified row does not exist
```

host oerr ora 8005

Result:
08005, 00000, "specified row does not exist"
// *Cause: A row with the given rowid does not exist in any of
the tables given
// *Action: check the query for misspellings of table names and
the rowid

In Oracle version 5 the second *delete* would have resulted in *0 record deleted.*

This changed behaviour has unfortunately not been found in the Oracle documantation.

10.7.6 Differences in the DECODE finction

The nature of the *decode* function has changed a little from version 5.1 to version 6. As snown below, all results must be of the same type - or more presise: all results must be convertable to the type of the first result.

In Oracle 5.1 this is checked at runtime (where you might be lucky not to hit the failing case), where as on Oracle 6.0 the check is done at parse time.

Suppose the **dual** table has one column *dummy*, and one row with the value 'X'.

```
SQL> select decode(dummy, 'A', 1, 'X', 'x', '*') from dual;
Oracle 5.1:
ERROR: ORA-1722:  invalid number

no records selected

Oracle 6.0:
select decode(dummy, 'A', 1, 'X', 'x', '*') from dual
                                 *
ERROR at line 1:
ORA-01722: invalid number
```

Almost the same, but on Oracle 5.1 we seem to get the error on execute, and in Oracle 6.0 at parse time.

```
SQL> select decode(dummy, 'A', 1, 'X', 2, '*') from dual;
Oracle 5.1:
DECODE(DUMMY,'A',1,'X',2,'*')
------------------------------
                             2

Oracle 6.0:
select decode(dummy, 'A', 1, 'X', 2, '*') from dual
                                    *
ERROR at line 1:
ORA-01722: invalid number
```

Since we on Oracle 5.1 does not hit the default case, no problem is reported. On Oracle 6.0 however a syntax error is reported.

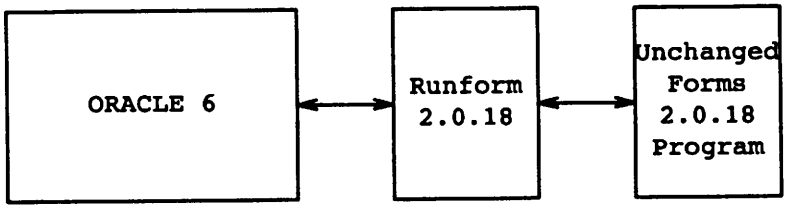
```
SQL> select decode(dummy, 'A', 'a', 'X', 2, '*') from dual;
Oracle 5.1:
DECODE(DUMMY,'A','A','X',2,'*')
---------------------------------------
2
Oracle 6.0:
DECODE(DUMMY,'A','A','X',2,'*')
---------------------------------------
2
```

No problem since a number can be converted to a character string. The same is true if dates are used instead of the number 2, because they also may be converted to character strings.

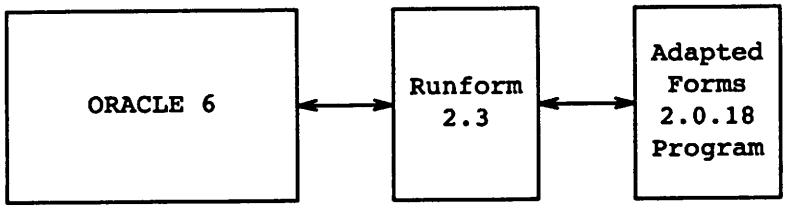
10.8 SQL*Forms Migration

It is difficult to have SQL*Forms 2.0 and SQL*Forms 2.3 available on the same Oracle instance. If you some reason want to mix anyway, you will have to study the advises given in this chapter. SQL*Forms applications developed in Forms 2.0.18 can be executed on Oracle 6 in two different ways :

Forms 2.0 under Oracle 6.0



Forms 2.3 under Oracle 6.0



*10.8.1 Running SQL*Forms 2.0 under Oracle 6*

It is possible to run SQL*Forms programs version 2.0.18 generated on Oracle version 5.1.17 on Oracle 6.0, if the following steps are followed.

- Copy the runform program to the Oracle 6 home

```
cp $ORACLE5_HOME/bin/runform $ORACLE6_HOME/bin/runform20
```

- The *National Language Support Mechanism* has changed, therefor the **language** dictionary should be copied from the Oracle 5 home to Oracle 6.

```
cd $ORACLE5_HOME
find language -print | cpio -padmv $ORACLE6_HOME
```

- The *crt*-definitions has changed as well, therefor you should copy the one you use from 5 to 6 in an other name, and use that one.

```
cp $ORACLE5_HOME/dbs/default.crt $ORACLE6_HOME/dbs/ora5crt.crt
```

  Run the *runform20* with the option *-c ora5crt*.

10.8.2 Forms 2.0 Applications under Forms 2.3

Forms 2.0 applications can be executed with the Runform 2.3 program if the application is adapted to run under the new version. Moving the applications is done using *.inp* files.

The macro-concept has some added functionality in Forms 2.3 - here the major enhancement is that macro's now returns *failure* when they cannot execute properly. When migrating form 2.0 applications you have to disable this new feature by inserting a **NOFAIL** command after each #EXEMACRO :

#EXEMACRO NOFAIL NXTBLK;

The way to do this is :

1. Load your applications into the SQL*Forms tables using the *LOAD* command or the *iac* program :

   iac -i inp_file form_name username/password

2. Log on *sqlplus* and run the script :

   start ?/forms/admin/macnfail

   Note: If the *NOFAIL* string makes the inp line too long, there will probably be problems in the application generation afterwards.

3. Recreate the *.inp* files :

   iac inp_file_name form_name username/password

4. and recreate the *.frm* files :

   iag inp_file_name -t

An alternative approach is to apply the changes directly to the *.inp* files, e.g. using a **sed** or **awk** command, and then re-generate the *.frm* files using the **iag** syntax shown above.

10.8.3 New Forms 2.3 Function-Keys

In Forms 2.3 has two new KEY-macro's **UP** and **DOWN** which are mapped to the function keys ↑ and ↓.

If the Forms 2.0 application includes a standard *Query Coordination* involving these arrow keys mapped to KEY-PRVREC and KEY-NXTREC, then the macro's defined for these two key should be copied to KEY-UP and KEY-DOWN in Forms 2.3, to give you the same functionality in the the new Form.

10.9 Changes in Rpt/Rpf

When using a **#CL**, Column Literal, command in Rpf the text following this command will be copied exactly as it is written. This text must then be terminated by a # written in the **first column**, as ducumented in the Rpt/Rpf Manual.

The older version of Rpf, the one used with Oracle 5, allowed for this terminator to appear elsewhere, even though the documentation stated otherwise.

This means that you might have reports where this # should be moved to the first column before executing under Rpt/Rpf version .

In RPT 1.1 numbers declared as *9* will not show the value *0*. If you want the same behaviour as in RPT 1.0, use the *-o* option, or declare the variable as *0* instead.

```
.declare number0 0
.declare number9 9
.set number0 0
.set number9 0
.print number0
.print number9
.add number0 number0 1
.add number9 number9 1
.print number0
.print number9
```

RPT 1.1 output:

```
0

1
1
```

RPT 1.1 -o output:

```
0
0
1
1
```

RPT 1.0 output:

```
0
0
1
1
```

RPT 1.1 will issue warnings when too large values are loaded into variables. No sign of warnings was issued on RPT 1.0.

MJ *Changes in RPT/RPF*

```
.declare var a4
.set var "abcde"
.print var
```

RPT 1.1 output: abcd

```
***
*** ERROR at line 2: RPT-0077: Value entered or set for var is greater than
*** declared -- Value truncated

*** Source: .set var "abcde"
***
```

RPT 1.1 -o output: abcd

```
***
*** ERROR at line 2: RPT-0077: Value entered or set for var is greater than
*** declared -- Value truncated

*** Source: .set var "abcde"
***
```

RPT 1.0 output: abcd

11. Hints on CASE products

11.1 CASE*Dictionary from a PC

A short comment on running CASE*Dictionary Programs on a PC with the Dictionary tables placed on the Supermax.

LONG strings cannot be send across SQL*Net and since one of the tables in Case Dictionary contains a LONG (the table is SDW_VIRTUAL_OBJECTS), the execution of the programs against a centralized database via SQL*Net could prove difficult.

12. SQL*Forms 3.0 Hints

12.1 The Data Model

SQL*Forms 30 stores information on the forms in 12 different tables in the database. These tables are documented in the manual *SQL*Forms Designer's Reference Version 3.0* chapter 12.

As these tables contains all the information from the forms generated as well as beeing designed, these tables may be used as f.ex. source for any cross-referencing tool on the forms.

In order to understand how these tables may be referenced, a number of different constraints may be issued.

```
rem MJ - 4 May 1991
rem create constraints on the forms30 tables.
rem

rem The form_app table
alter table form_app_ add (
  primary key (appowner, appname) constraint app_prim_key);

rem The form_sqltxt table
alter table form_sqltxt_ add (
  primary key (sqtapowner, sqtapname, sqtno, sqtline) constraint sqt_prim_key,
  unique (sqtapowner, sqtapname, sqtno) constraint sqt_unique_key,
  foreign key (sqtapowner, sqtapname)
    references form_app_ /* (appowner, appname) */ constraint sqt_for_key);

rem The form_blk table
alter table form_blk_ add (
  primary key (blkapowner, blkapname, blkname) constraint blk_prim_key,
  foreign key (blkapowner, blkapname)
    references form_app_ /* (appowner, appname) */ constraint blk_for_key,
  foreign key (blkapowner, blkapname, blkobysql)
    references form_sqltxt_ (sqtapowner, sqtapname, sqtno)
    constraint blk_for2_key,
  check (blkhide in ('N', 'Y')) constraint blk_hide_val,
  check (blkunqkey in ('N', 'Y')) constraint blk_unqkey_val,
  check (blkctrl in ('N', 'Y')) constraint blk_ctrl_val,
  check (blkcolsec in ('N', 'Y')) constraint blk_colsec_val);

rem The form_fld table
alter table form_fld_ add (
  primary key (fldapowner, fldapname, fldblk, fldname) constraint fld_prim_key,
  foreign key (fldapowner, fldapname, fldblk)
    references form_blk_ /* (blkapowner, blkapname) */ constraint fld_for_key,
  foreign key (fldapowner, fldapname, fldlovsql)
    references form_sqltxt_ (sqtapowner, sqtapname, sqtno)
    constraint fld_for2_key,
  check (fldbtab in ('N', 'Y')) constraint fld_btab_val,
  check (fldkey in ('N', 'Y')) constraint fld_key_val,
  check (fldprabov in ('N', 'Y')) constraint fld_prabov_val,
  check (fldprrpt in ('N', 'Y')) constraint fld_prrpt_val,
  check (fldenter in ('N', 'Y')) constraint fld_enter_val,
  check (fldquery in ('N', 'Y')) constraint fld_query_val,
  check (fldupdate in ('N', 'Y')) constraint fld_update_val,
  check (fldupdnul in ('N', 'Y')) constraint fld_updnul_val,
```

```
  check (fldmand in ('N', 'Y')) constraint fld_mand_val,
  check (fldfixed in ('N', 'Y')) constraint fld_fixed_val,
  check (fldskip in ('N', 'Y')) constraint fld_skip_val,
  check (fldhide in ('N', 'Y')) constraint fld_hide_val,
  check (fldautohlp in ('N', 'Y')) constraint fld_autohlp_val,
  check (fldupper in ('N', 'Y')) constraint fld_upper_val,
  check (fldedtwrp in ('N', 'Y')) constraint fld_edtwrp_val);

rem The form_trigger table
alter table form_trigger_ add (
  primary key (trigapowner, trigapname, trigblk, trigfld, trigtype)
    constraint trig_prim_key,
  /* Some of the columns of this primary key may contain null,        */
  /* on the other hand it would be misleding to restrict the primary key */
  foreign key (trigapowner, trigapname, trigblk, trigfld)
    references form_fld_ /* (fldapowner, fldapname, fldblk, fldname) */
    constraint trig_for_key,
  check (trighide in ('N', 'Y')) constraint trig_hide_val,
  check (trigplsql in ('N', 'Y')) constraint trig_plsql_val);

rem The form_trg table
alter table form_trg_ add (
  primary key (trgapowner, trgapname, trgblk, trgfld, trgtype, trgseq)
    constraint trg_prim_key,
  /* Some of the columns of this primary key may contain null,        */
  /* on the other hand it would be misleding to restrict the primary key */
  foreign key (trgapowner, trgapname, trgblk, trgfld, trgtype)
    references form_trigger_
    /* (trigapowner, trigapname, trigblk, trigfld, trigname) */
    constraint trg_for_key,
  foreign key (trgapowner, trgapname, trgsql)
    references form_sqltxt_ (sqtapowner, sqtapname, sqtno)
    constraint trg_for2_key);

rem The form_page table
alter table form_page_ add (
  primary key (pagapowner, pagapname, pagnum) constraint pag_prim_key,
  foreign key (pagapowner, pagapname)
    references form_app_ /* (appowner, appname) */ constraint pag_for_key);

rem The form_map table
alter table form_map_ add (
  primary key (mapapowner, mapapname, mappage, mapline, mapgrph)
    constraint map_prim_key,
  foreign key (mapapowner, mapapname, mappage)
    references form_page_ /* (pagapowner, pagapname, pagnum) */
    constraint map_for_key);

rem The form_comment table
alter table form_comment_ add (
  primary key (cmtapowner, cmtapname, cmtproc, cmtblk, cmtfld, cmttrgtyp,
          cmttrgseq, cmtline)
    constraint cmt_prim_key,
  /* Some of the columns of this primary key may contain null,        */
  /* on the other hand it would be misleding to restrict the primary key */
  foreign key (cmtapowner, cmtapname)
    references form_app_ /* (appowner, appname) */
    constraint cmt_for_key);

rem The form_reference table
alter table form_reference_ add (
  primary key (refapowner, refapname, refblk, reffld, reftrgtyp, refproc)
    constraint ref_prim_key,
  /* Some of the columns of this primary key may contain null,        */
  /* on the other hand it would be misleding to restrict the primary key */
```

```
  foreign key (refapowner, refapname)
    references form_app_ /* (appowner, appname) */
    constraint ref_for_key);

rem The form_procedure table
alter table form_procedure_ add (
  primary key (procapowner, procapname, procname)
    constraint proc_prim_key,
  foreign key (procapowner, procapname)
    references form_app_ /* (appowner, appname) */
    constraint proc_for_key,
  foreign key (procapowner, procapname, proctext)
    references form_sqltxt_ (sqtapowner, sqtapname, sqtno)
    constraint proc_for2_key);

rem The form_user table
alter table form_user_ add (
  primary key (usrapowner, usrapname, usruser)
    constraint usr_prim_key,
  foreign key (usrapowner, usrapname)
    references form_app_ /* (appowner, appname) */
    constraint usr_for_key);
```

If you need to drop or adjust the constrains, you may run the script below.

```
rem MJ - 4 May 1991
rem drop constraints on the forms30 tables.
rem

rem The form_user table
alter table form_user_ drop constraint usr_prim_key;
alter table form_user_ drop constraint usr_for_key;

rem The form_procedure table
alter table form_procedure_ drop constraint proc_prim_key;
alter table form_procedure_ drop constraint proc_for_key;
alter table form_procedure_ drop constraint proc_for2_key;

rem The form_reference table
alter table form_reference_ drop constraint ref_prim_key;
alter table form_reference_ drop constraint ref_for_key;

rem The form_comment table
alter table form_comment_ drop constraint cmt_prim_key;
alter table form_comment_ drop constraint cmt_for_key;

rem The form_map table
alter table form_map_ drop constraint map_prim_key;
alter table form_map_ drop constraint map_for_key;
alter table form_map_ drop constraint map_grph_val;

rem The form_page table
alter table form_page_ drop constraint pag_prim_key;
alter table form_page_ drop constraint pag_for_key;
alter table form_page_ drop constraint pag_popup_val;
alter table form_page_ drop constraint pag_border_val;
alter table form_page_ drop constraint pag_dislv_val;
alter table form_page_ drop constraint pag_hscrl_val;
alter table form_page_ drop constraint pag_vscrl_val;

rem The form_trg table
alter table form_trg_ drop constraint trg_prim_key;
alter table form_trg_ drop constraint trg_for_key;
alter table form_trg_ drop constraint trg_for2_key;
```

```
alter table form_trg_ drop constraint trg_curs_val;
alter table form_trg_ drop constraint trg_mve_val;
alter table form_trg_ drop constraint trg_inv_val;
alter table form_trg_ drop constraint trg_roll_val;

rem The form_trigger table
alter table form_trigger_ drop constraint trig_prim_key;
alter table form_trigger_ drop constraint trig_for_key;
alter table form_trigger_ drop constraint trig_hide_val;
alter table form_trigger_ drop constraint trig_plsql_val;

rem The form_fld table
alter table form_fld_ drop constraint fld_prim_key;
alter table form_fld_ drop constraint fld_for_key;
alter table form_fld_ drop constraint fld_for2_key;
alter table form_fld_ drop constraint fld_btab_val;
alter table form_fld_ drop constraint fld_key_val;
alter table form_fld_ drop constraint fld_prabov_val;
alter table form_fld_ drop constraint fld_prrpt_val;
alter table form_fld_ drop constraint fld_enter_val;
alter table form_fld_ drop constraint fld_query_val;
alter table form_fld_ drop constraint fld_update_val;
alter table form_fld_ drop constraint fld_updnul_val;
alter table form_fld_ drop constraint fld_mand_val;
alter table form_fld_ drop constraint fld_fixed_val;
alter table form_fld_ drop constraint fld_skip_val;
alter table form_fld_ drop constraint fld_hide_val;
alter table form_fld_ drop constraint fld_autohlp_val;
alter table form_fld_ drop constraint fld_upper_val;
alter table form_fld_ drop constraint fld_edtwrp_val;

rem The form_blk table
alter table form_blk_ drop constraint blk_prim_key;
alter table form_blk_ drop constraint blk_for_key;
alter table form_blk_ drop constraint blk_for2_key;
alter table form_blk_ drop constraint blk_hide_val;
alter table form_blk_ drop constraint blk_unqkey_val;
alter table form_blk_ drop constraint blk_ctrl_val;
alter table form_blk_ drop constraint blk_colsec_val;

rem The form_sqltxt table
alter table form_sqltxt_ drop constraint sqt_prim_key;
alter table form_sqltxt_ drop constraint sqt_unique_key;
alter table form_sqltxt_ drop constraint sqt_for_key;

rem The form_app table
alter table form_app_ drop constraint app_prim_key;
```

Having installed appropriate constrains on the forms30 tables, you may now easily create forms applications on the forms30 tables to inspect the forms.

Please note: **Do not change data in the forms30 tables, unless you are 200% sure.** There is no guarentee that your forms will work.

12.2 Comments on Referenced Objects

When referencing an object from one form to another, every attribute of that object, including lower level objects will be referenced.

The exception to this rule is comments for the referenced objects. Comments will not be referenced, so in the referencing form, the designer is free to type in her own comments.

If you did type in some good comments on the objects in the referenced form, you would like the designers of the referencing forms to copy those comments over to their forms as well, before they adjust them.

The following SQL-script will for at given referenced form create a script, which will copy comments on referenced objects to the referencing forms.

Please note that the forms must be pressent in the database, and that no SQL*Forms30 designer should hold the forms in memory, when the scripts are executed.

```
set termout off
rem creacom - create a command SQL-script to transfer comments for
rem           referenced objects from one form to another.
rem           1. argument is the username,
rem           2. argument is the form name holding the comments
rem MJ / DDE 2. May 91.
rem

set echo off
set heading off
set feedback off
set verify off
set linesize 200

spool comments.sql

rem Take care of the Triggers.
select 'insert into form_comment (cmtapowner, cmtapname, cmtproc, cmtblk, ' ||
     'cmtfld, cmttrgtyp, cmttrgseq, cmtline, cmttext) ' ||
     'select ''' || refapowner || ''', ''' || refapname || ''', ' a,
     'cmtproc, cmtblk, cmtfld, cmttrgtyp' c,
     ', cmttrgseq, cmtline, cmttext from form_comment ' ||
     'where cmtapowner = upper(''&1'') and cmtapname = ''&2''' d,
     ' and cmttrgtyp = ''' || reftrgtyp || ''';'
from form_reference
where reffowner = lower('&1')
  and reffapp = '&2'
  and reftrgtyp is not null;

rem Take care of the Procedures.
select 'insert into form_comment (cmtapowner, cmtapname, cmtproc, cmtblk, ' ||
     'cmtfld, cmttrgtyp, cmttrgseq, cmtline, cmttext) ' ||
     'select ''' || refapowner || ''', ''' || refapname || ''', ' a,
     'cmtproc, cmtblk, cmtfld, cmttrgtyp' c,
     ', cmttrgseq, cmtline, cmttext from form_comment ' ||
     'where cmtapowner = upper(''&1'') and cmtapname = ''&2''' d,
     ' and cmtproc = ''' || refproc || ''';'
from form_reference
where reffowner = lower('&1')
  and reffapp = '&2'
  and refproc is not null;

rem Take care of the Blocks.
select 'insert into form_comment (cmtapowner, cmtapname, cmtproc, cmtblk, ' ||
     'cmtfld, cmttrgtyp, cmttrgseq, cmtline, cmttext) ' ||
     'select ''' || refapowner || ''', ''' || refapname || ''', ' a,
     'cmtproc, cmtblk, cmtfld, cmttrgtyp' c,
     ', cmttrgseq, cmtline, cmttext from form_comment ' ||
     'where cmtapowner = upper(''&1'') and cmtapname = ''&2''' d,
     ' and cmtblk = ''' || refblk || ''';'
from form_reference
```

```
where reffowner = lower('&1')
  and reffapp = '&2'
  and refblk is not null;

rem Take care of the Fields.
select 'insert into form_comment (cmtapowner, cmtapname, cmtproc, cmtblk, ' ||
      'cmtfld, cmttrgtyp, cmttrgseq, cmtline, cmttext) ' ||
      'select ''' || refapowner || ''', ''' || refapname || ''', ' a,
      'cmtproc, cmtblk, cmtfld, cmttrgtyp' c,
      ', cmttrgseq, cmtline, cmttext from form_comment ' ||
      'where cmtapowner = upper(''&1'') and cmtapname = ''&2''' d,
      ' and cmtfld = ''' || reffld || ''';'
from form_reference
where reffowner = lower('&1')
  and reffapp = '&2'
  and reffld is not null;

select 'commit;' from dual;

spool off
set heading on
set feadback on
set verify on
set termout on
prompt --- Edit the script 'comments.sql' - and run it.
set echo on
```

12.3 NLS in Forms

The ability to generate forms for different languages has evolved quite far already. The kernel and runform messages does already correspond to the language environment.

Still the forms designer has to maintain different copies of a form, if she wants to make a german, an american, and a danish version of the form. This is due to the fact that help texts, prompt texts, literals and masks all has to be specified according to the language in question.

You may however rely on the fact that the **inp** file is a text file, and use a macro processer to patch in the correct part for your language.

Let us assume that you did your basis form (say **sample.inp**), and you want to make language divertions from this form.

Edit your **sample.inp** file like this:

```
/* Copyright (c) 1988 by the Oracle Corporation */

SQL*FORMS_VERSION = 03.00.15.03.01

DEFINE FORM

  NAME = _NLS_NAME
:       :       :       :       :
  DEFINE BLOCK

    NAME = orders
:       :       :       :       :
    DEFINE FIELD

      NAME = ORDID
:       :       :       :       :
      HELP = _NLS_ORDID_HELP
:       :       :       :       :
    DEFINE FIELD

      NAME = ORDERDATE
      DATATYPE = DATE
      LENGTH = _NLS_DATE_LENGTH
      DISPLAY_LENGTH = _NLS_DATE_LENGTH
      QUERY_LENGTH = _NLS_DATE_LENGTH
:       :       :       :       :
      INPUT_MASK = _NLS_DATE_MASK
      OUTPUT_MASK = _NLS_DATE_MASK
:       :       :       :       :
      MODE = TEXT
      LINE = 3
      BOILER = <<<
      _NLS_BOILER1
      >>>
:       :       :       :       :
ENDDEFINE FORM
```

And create an include file **sample.h** like this (assuming the *cpp* preprocessor will be used):

```
#ifdef DK
  /* Tekster er på dansk! */
#define _NLS_LANG DK
#define _NLS_NAME SAMPLEDK
#else
  /* Texts are in american */
#define _NLS_LANG US
#define _NLS_NAME SAMPLEUS
#endif

#ifdef DK
#define _NLS_DATE_LENGTH 8
#define _NLS_DATE_MASK DD/MM-YY
#else
#define _NLS_DATE_LENGTH 9
#define _NLS_DATE_MASK
#endif

#ifdef DK
#define _NLS_ORDID_HELP Tast værdi ind for ORDID
#define _NLS_BOILER1 O R D R E R
#else
#define _NLS_ORDID_HELP Enter value for : ORDID
#define _NLS_BOILER1 O R D E R S
#endif
```

Now generate the language specific **inp** files:

```
$ echo "#include <sample.h>" > remove
$ cat sample.inp >> remove
$ for L in DK US D
$ do
$   /lib/cpp -P -C -D$L -I. remove >sample$L.inp
$ done
$ rm remove
```

You may check if all the symbols has been solved by *cpp*:

```
$ grep _NLS_ sample?*.inp
```

Now generate the **frm** files:

```
$ for L in DK US D
$ do
$   generate30 -bt sample$L scott/tiger
$ done
```

Restore the basis form (**sample.inp**) to the database, if only text is redefined:

```
$ convert30 -d sample sample scott/tiger
$ convert30 -i sample sample scott/tiger
```

Assume your shell variable *LANG* has the value 'DK', 'US' or 'D', your form may be called:

```
$ runform30 sample$LANG scott/tiger
```

You may procede adjusting your basis form through *sqlforms30*, as long as you run *cpp* and *generate30* afterwards as described.

As shown in the example not only text can be modified, also masks, length, etc. Be carefull when using *$$DATE$$*, as the format here is 'DD-MON-YY', and do not have language literals in your user exits.

12.4 Debug Facilities

First you may utillize the SQL*Forms 30 Debugger by inserting the *break;* command in your triggers and use the debug option when the application is executed.

The SQL*Forms 30 Debugger has the nice capability of being able to show you the contents of system-, global- and control variables.

A long trigger may look like this:

```
:GLOBAL.debug1 := 'Trigger start';
break;
/* First action */
:GLOBAL.debug1 := 'After first action';
break;
/* Last action */
:GLOBAL.debug1 := 'After last action';
break;
```

If you tend to redefine most keys with key-triggers in your application, you may also redefine the *KEY-OTHERS* trigger in order to trap keys you did forget to redefine.

You may alse inspect the SQL-statements passed from runform30 to the kernel. This is easily done by issuing the -s swich when executing the form. The disadvantage here is that *all* the SQL-statements for the whole session are examined. If you want to be able to control - in the middle of an runform session - when to start and stop the trace facilitty, you may use the following scheme.

First observe that the *alter session* ... statement are not legal in an PL*SQL block.

Next, observe that if SQL*Menu 50 and SQL*Plus are linked together with your SQL*Forms 30, they will be able to work on your *current* Oracle session.

Make two new user key triggers in your application:

```
KEY-F2: replace_menu('debugmenu', BAR, 'debug');
                /* To start the debugmenu at debug */

KEY-F2: replace_menu('default');
                /* To set the menu back to the default */
```

The *debugmenu* may look like this:

```
DEBUG

+-------------------------------------------------------------------+
|                              Debug                                |
|                               ...                                 |
| 1  Call linked SQL*Plus                                           |
| 2  Set trace on                                                   |
| 3  Set trace off                                                  |
|                                                                   |
| Purpose:                                                          |
|    Testing the trace facility from forms                          |
+-------------------------------------------------------------------+

Item Number:    1
  Short Name:    Sqlplus
  Command Type:  SQL*Plus.
  Command Line:
  =============
  sqlplus

Item Number:    2
  Short Name:    traceOn
  Command Type:  SQL*Plus.
  Command Line:
  =============
  sqlplus -s / @traceon

Item Number:    3
  Short Name:    traceoFf
  Command Type:  SQL*Plus.
  Command Line:
  =============
  sqlplus -s / @traceoff
```

The SQL*Plus scripts *traceon* and *traceoff* look like this:

```
Rem Traceon
set termout off
alter session set sql_trace true;
exit

Rem Traceoff
set termout off
alter session set sql_trace false;
exit
```

Please note that as a side effect, all pending changes made to the database will be committes by the *alter session* command.

12.5 The LONG Datatype

The support for LONG datatypes in SQL*Forms 30 has improved quite a lot, because the editor may now handle these fields. There are however some minor aspects still to be considered beside the ones mentioned in *ORACLE RDBMS Database Administrator's Giode Version 6.0 - LONG Datatype*.

- The maximum number of characters stored in a LONG field is 65534.
- If the length of the LONG field i SQL*Forms 30 is less than the current lenght of a LONG value from the database, the error *FRM-40831* is issued, and long values to follow may be absent in the form.

  ERROR FRM-40831: Truncation occurred: Value too long for field %s.
- Do not try to make a form on a table where a LONG RAW column is used. Runform30 will not be able to select or insert values to the LONG RAW column.
- It may take more than a minute (which it a very long time to the operator) to bring the forms editor up on a long (more than 60000 characters) LONG field.

12.6 Migration to SQL*Forms V3.0

As an example we will try to move the demo form **demog** Version 2.3 from the demo directry **$ORACLE_HOME/forms/demo** to SQL*Forms 3.0.

We will do it in two steps: First converting to Version 3.0 with a minimun of changes. Second change the application to utillize some of the new features of SQL*Forms 3.0.

12.6.1 From 2.3 to 3.0

Before the *convert30* program is executed the, *inp* file should be copied, or you may lose the old form.

```
cp demog.inp demog3.inp
convert30 -v23 demog3 demog3
```

The fiels **TWO.EMPNO** will automatically be moved to page 0, which is nice since it makes it possible to see the value directly from the Forms Debugger.

The five old V2 triggeres will be there, working correctly.

Note that user exits must be moved and linked into SQL*Forms 3.0 in this step, as well as potential problems with data types must be taken care of.

12.6.2 Converting V2 triggers to V3

Because we want to compare the different forms afterwards, we make a new copy of the form, before inserting the form into the database.

```
cp demog3.inp demog3o.inp
convert30 -i demog3o demog3o personnel/john
```

The SQL*Forms Design manual makes some comments on how to convert the triggers (Chapter B 2). That scheme is followed here, only a bit more compressed.

Please note, that when the individual trigger is analyzed, the V2 text is not shown in the text window of the trigger. You will have to *zoom in* on the trigger to view the trigger steps, or you may simply look in the **inp** file directly, and just typing the V3 text into the text window of the trigger.

The *PRE-DELETE* trigger for block *ONE* in V2 format look like this:

```
DEFINE STEP
  LABEL =
  TEXT = < < <
  select empno from assignments
  where empno = &one.empno
  > > >
  SUCCESS_STEP =
  FAILURE_STEP =
  FAIL_MESSAGE = Delete project assignments for this employee before deleting employee!
  ABORT = ON
  REVERSE = ON
  NEW_CURSOR = ON
  IGNORE = OFF
```

The *PRE-DELETE* trigger for block *ONE* in V3 format may look like this:

```
declare
  dummy_into_buffer char(240);
  cursor dummy_select_into_cursor is
    select empno from assignments
    where empno = :one.empno;
begin
  open dummy_select_into_cursor;
  fetch dummy_select_into_cursor into dummy_into_buffer;
  if dummy_select_into_cursor%notfound then
    raise no_data_found;
  end if;
  message('Delete project assignments for this employee before deleting employee.');
  raise form_trigger_failure;
exception
  when no_data_found then null;
end;
```

The *POST-CHANGE* trigger for field *ONE.CITY*, in V2 format look like this:

```
DEFINE STEP
  LABEL =
  TEXT = < < <
  select state into one.state
  from cities where city = &one.city
  > > >
  SUCCESS_STEP =
  FAILURE_STEP =
  FAIL_MESSAGE = City not found in state table
  ABORT = OFF
  REVERSE = OFF
  NEW_CURSOR = ON
  IGNORE = OFF
```

Since the trigger is used to validate the field rather than getting more information from the database, you may save execution time by making it a *ON-VALIDATE-FIELD* trigger. The *ON-VALIDATE-FIELD* trigger for field *ONE.CITY*, may in V3 format look like this:

```
declare
  cursor dummy_select_into_cursor is
    select state from cities
    where city = :one.city;
begin
  open dummy_select_into_cursor;
  fetch dummy_select_into_cursor into :one.state;
  if dummy_select_into_cursor%notfound then
    raise no_data_found;
  end if;
exception
  when no_data_found then
    message('City not found in state table');
    raise form_trigger_failure;
end;
```

The *POST-CHANGE* trigger for field *ONE.POSITION*, in V2 format look like this:

```
OLD:
DEFINE STEP
  LABEL =
  TEXT = < < <
  select * from employees
  where position = &position
  > > >
  SUCCESS_STEP =
  FAILURE_STEP =
  FAIL_MESSAGE = Warning: No one in the database now has that job . .. check spelling!
  ABORT = OFF
  REVERSE = OFF
  NEW_CURSOR = ON
  IGNORE = OFF
```

Since the trigger is used to validate the field rather than getting more information from the database, you may save execution time by making it a *ON-VALIDATE-FIELD* trigger. The *ON-VALIDATE-FIELD* trigger for field *ONE.POSITION*, may in V3 format look like this:

```
declare
  dummy_into_buffer char(240);
  cursor dummy_select_into_cursor is
    select empno from employees
    where position = :position;
begin
  open dummy_select_into_cursor;
  fetch dummy_select_into_cursor into dummy_into_buffer;
  if dummy_select_into_cursor%notfound then
    raise no_data_found;
  end if;
exception
  when no_data_found then
    message('Warning: No one in the database now has that job ... check spelling!');
end;
```

The *List of Values* facility for the field *ONE.DEPTNO* looks like this:

```
LOV_TEXT = <<<
departments.deptno
>>>
LOV_TITLE =
LOV_X =
LOV_Y =
```

And may be changed to:

```
LOV_TEXT = <<<
select deptno, dname into :one.deptno, :one.dname
from departments order by deptno
>>>
LOV_TITLE = Departments
LOV_X = 20
LOV_Y = 5
```

The *POST-CHANGE* trigger for field *ONE.DEPTNO*, in V2 format look like this:

```
DEFINE STEP
  LABEL =
  TEXT = <<<
  select dname into dname
  from departments where deptno = &deptno
  >>>
  SUCCESS_STEP =
  FAILURE_STEP =
  FAIL_MESSAGE = Invalid department number...Try using "<esc> v" for list of valid values.
  ABORT = ON
  REVERSE = OFF
  NEW_CURSOR = ON
  IGNORE = OFF
```

The *POST-CHANGE* trigger for field *ONE.DEPTNO*, may in V3 format look like this:

```
declare
  cursor dummy_select_into_cursor is
    select dname from departments
    where deptno = :deptno;
begin
  open dummy_select_into_cursor;
  fetch dummy_select_into_cursor into :dname;
  if dummy_select_into_cursor%notfound then
    raise no_data_found;
  end if;
exception
  when no_data_found then
    message('Invalid department number. Use <esc>-v to list valid values.');
    raise form_trigger_failure;
end;
```

The *POST-CHANGE* trigger for field *TWO.PROJNO*, in V2 format look like this:

```
DEFINE STEP
  LABEL =
  TEXT = <<<
  select pname into pname
  from projects where projno = &projno
  >>>
  SUCCESS_STEP =
  FAILURE_STEP =
  FAIL_MESSAGE = Invalid project number; try <esc-v> for list of valid values
  ABORT = ON
  REVERSE = OFF
  NEW_CURSOR = ON
  IGNORE = OFF
```

The *POST-CHANGE* trigger for field *TWO.PROJNO*, may in V3 format look like this:

```
declare
  cursor dummy_select_into_cursor is
    select pname from projects
    where projno = :projno;
begin
  open dummy_select_into_cursor;
  fetch dummy_select_into_cursor into :pname;
  if dummy_select_into_cursor%notfound then
    raise no_data_found;
  end if;
exception
  when no_data_found then
    message('Invalid project number; try <esc-v> for list of valid values');
    raise form_trigger_failure;
end;
```

Triggers composed of more steps should be reconsidered when converted into PL/SQL.

12.6.3 Form sizes

Here we check the sizes of the **inp** and **frm** files, as well as the data segment sizes of the *runform* and *Oracle* processes.

	inp-file (k bytes)	frm-file (k bytes)	runform-data (k bytes)	Oracle-data (k bytes)
demog	12	4	130	250
demog3	33	4	220	250
demog3o	34	7	230	250

Since the form *demog3* only uses V2 triggers, you may actually save the PL/SQL code from the *runform30* text. This is however not recommented since your development possibilities will be limited.

The *runform30* text sizes and initial data are (without user exits):

	text (k bytes)	data (k bytes)
+plsql,+menu5	870	107
+plsql,-menu5	648	74
-plsql,-menu5	489	43

12.7 Get Shell Variable - a User-exit

Here we shall demonstrate how to make an user-exit in C, to copy a *shell* variable into a forms variable. And how the userexit might be used from a form.

First create the source of the user-exit **getenvr.pc**:

```
/* ************************************************************************
GETENVR will get the value of the <envr-name> and store it in the
        <field-name> with the maximum length of <max-length>

GETENVR is called with the following syntax for the parameter string p:

USAGE (in forms 3.0): USER-EXIT('GETENVR envr-name field-name max-length')

13. May 91 - MJ - DDE
************************************************************************ */

#include <stdio.h>
#include <usrxit.h>

#define MAXARGS             128
#define ARBUFSIZ            512

EXEC SQL BEGIN DECLARE SECTION;
                VARCHAR formname [30]; /* form variable name */
                VARCHAR varvalue [128];/* Variable to store value */
EXEC SQL END DECLARE SECTION;
EXEC SQL INCLUDE SQLCA.H;

char *wordb[MAXARGS];                   /* Holds pointers to the blank separated  */
                                        /* tokens in the string passed to GETENVR */
int getenvr(p, paramlen, erm, ermlen, query)
  register char *p;                     /* Parameter string */
  int *paramlen;                        /* Ptr to param string length */
  char *erm;                            /* Error message if doesnt match */
  int *ermlen;                          /* Ptr to error message length */
  int *query;                           /* Ptr to query status flag */
{
  register int listsiz;                 /* Number of values */
  register int i;                       /* Temp counter */
  char arbuf[ARBUFSIZ];                 /* To hold string that is passed */
  int maxlen;                           /* The max length to copy */
  char *cpoint;                         /* Temp counter */

  EXEC SQL WHENEVER SQLERROR GOTO sqlerr;

  remblank( p );                        /* Remove leading, trailing, and
                                           and double spaces */

  strncpy(arbuf, p, ARBUFSIZ-1 );
  listsiz = countargs( arbuf );         /* get envrname, fieldname and maxlen
                                           in wordb[] */

  if ( listsiz != 4 )                   /* Check for 4 arguments */
    return IAPFTL;

  strncpy(formname.arr, wordb[2], 30);  /* Store the form-name */
  formname.len = strlen(formname.arr);

  sscanf(wordb[3], "%d", &maxlen);      /* get the maxlen */

  strncpy(varvalue.arr, getenv(wordb[1]), maxlen);/* Get and store env */
  if (strlen(varvalue.arr) == 0) return IAPFAIL;
  varvalue.len= strlen(varvalue.arr);

  EXEC IAF PUT :formname values ( :varvalue );/* Store value in the form */

  return IAPSUCC;

sqlerr:                                 /* General error exception */
  sqlca.sqlerrm.sqlerrmc[sqlca.sqlerrm.sqlerrml] = ' ';
```

```
  sqliem( sqlca.sqlerrm.sqlerrmc, &sqlca.sqlerrm.sqlerrml );
  return IAPFAIL;
}

/* countargs - counts arguments in string and sets pointers in wordb[] */
/*             to point to individual, null-terminated tokens.  Relies */
/*             on string having no leading, trailing or double blanks. */

int countargs(textp)
char *textp;
{
  int numargs;
  char *sp;
  extf char *strchr();

  for ( numargs = 0, sp = textp; numargs < MAXARGS && sp; numargs++ ) {
    wordb[numargs] = sp;
    sp = strchr( textp, ' ');

    if ( sp ) {
      *sp = ' ';
      textp = ++sp;
    }
  }
  return(numargs);
}

/* Remove leading, trailing, and and double spaces */

int remblank (textp)
char *textp;
{
  register char *pin = textp;
  register char *pout = pin;
  register char c;

  while ( *pin == ' ' ) pin++;
  for (; c = *pin; ++pin) {
    if (c == ' ') {
      *pout++ = ' ';
       for (++pin; (c = *pin) && c == ' '; ++pin) ;
       if ( !c ) pout--;
    }
    if (c) {
      if (c != ' ')
        *pout++ = c;
      else
        --pin;
    } else
      break;
  }
  *pout = ' ';
}
```

Precompile and compile the file to **getenvr.o** by:

```
$ pcc include=$ORACLE_HOME/c/lib ireclen=132 oreclen=132 \
      sqlcheck=none iname=getenvr.pc
$ cc -c -I. -I$ORACLE_HOME/forms30/lib -O getenvr.c
```

Append an entry in the **iapxtb** table, by calling the form **genxtb** in the **$ORACLE_HOME/forms30/lib** directory.

MJ *Get Shell Variable - a User-exit*

```
********** GENXTB **********

User Exit  Type    Remarks                                    Creation  Modify
getenvr    C       get Unix environment to forms 30 variabe 13-MAY-91
```

If the **iapxtb** table has not been created for your account, run:

```
$ genxtb <user-name>/<password>
```

Generate the user-exit entry module:

```
$ genxtb <user-name>/<password> iapxtb.c
```

The **iapxtb.c** file will look like this:

```
#include <stdio.h>
#include <usrxit.h>

/* Define the user exits for IAP. */

extf          int       getenvr();

externdef     exitr     iapxtb[] = {
              "GETENVR",          getenvr,2,
              (char *)0,          0,0
              };

/* End of iapxtb. */
```

Relink *sqlforms30x* and *runform30x* with the new user-exit:

OTHERXIT = getenvr.o make -f sqlforms30.mk sqlforms30x

The following trigger will use the userexit to get the environment name stored in the field *ENVR*, and store the environment value in the field *VAL*. If it succede, the trigger step will validate the *VAL* field:

```
user_exit('getenvr '||name_in('ENVR')||' VAL  '||
      field_characteristic('VAL',field_length));
if form_fatal then
 message('GETENVR USAGE: env_var form_var max_len');
elsif form_failure then
 message('Envr >'||name_in('ENVR')||'< does not exist');
else
 go_field('VAL');
 next_field;
end if;
```

Note that all litteral text in english has beem moved from the user-exit to the calling trigger, to make it easier to translate. You may even copy values into fields other than character fileds, just remember to validate.

Get Shell Variable - a User-exit MJ

12.8 Ask Question in a Window

Here we shall demonstrate how to make an user-exit in C, to paint a window on top of your form using the rutines described in the *SQL*Forms Designer's Reference Version 3.0* manual chapter 19.

We will use such a window to ask the user a simple question, returning the answer back to the calling trigger. After the example we will comment on various errors concerning the window management facilities from user exits. Note that we did build in a timeout, inorder to prevent possible locking problems, if the user exit is called in a trigger fired during the commit phase.

You may argue that the Oracle*Tool Version 2.0 Alert facility would do the job easily, but 2.0 is not available yet, and the example will anyway demonstrate the use of the window management facility.

```
/* ***************************************************************************
WIND_ASK popup an alert window, and ask the user to type in an boolean answer.
         <Return> wille select the default value.
         A timeout of 10 sec. will automatical select the default answer.

WIND_ASK is called with the following syntax for the parameter string p:

USAGE (in forms 3.0): USER-EXIT('WIND_ASK x y v h title default other message')

The x,y is the position of the upper left corner of the window, v and h
    is the width and hight.
The title (one word) will be showed in top of the window.
The default and the other words are the two possibilities to choose from,
    the first letters of the two words must not be equal.
The message is the string to pressent the choise.

16. May 91 - MJ - DDE
*************************************************************************** */

#include <ctype.h>
#include <signal.h>
#include <iapuxw.h>
#include <usrxit.h>

#define TRUE                1
#define FALSE               0
#define TIMEOUT             10
#define MAXARGS             128
#define ARBUFSIZ            512

static int IAPSUCC = 0;
static int IAPFAIL = 1403;
static int IAPFTL  = 535;

char *wordb[MAXARGS];           /* Holds pointers to the blank separated  */
                                /* tokens in the string passed to GETENVR */

int alarm_cught;

int *alarm_handle(signo, info, extra)   /* Rutine called when timeout */
  int signo;                            /* from the alarm expiors     */
  struct businfo *info;
  int extra;
```

```
{ alarm_cught = TRUE; }

char *on_damag(a_window, x, y, h, v, a_param)
  uxwind a_window;
  int x, y, h, v;
  char *a_param;
{ /* Not nessesary since the ask window will be on top. */ }

int wind_ask(p, paramlen, erm, ermlen, query)
  char *p, *erm;
  int  *paramlen, *ermlen, *query;
{
  uxwind ask_window;                    /* The ask window handle */
  char title[20], deftxt[20], other[20], mess[60];
  register int listsiz;                 /* Number of values */
  register int i;                       /* Temp counter */
  char arbuf[ARBUFSIZ];                 /* To hold string that is passed */
  int x, y, h, v, rcode;
  int def_x, def_y;                     /* Position on default string */
  char mesg[80], ch;
  int (*oldsig)();                      /* The previous signal handler
                                           for SIGALRM */

  remblank( p );                        /* Remove leading, trailing, and
                                           and double spaces */

  strncpy(arbuf, p, ARBUFSIZ-1 );
  listsiz = countargs( arbuf );         /* get fieldname and value list
                                           in wordb[] */

  if ( listsiz < 9 )                    /* Check for at least 9 arguments */
    return IAPFTL;

  sscanf(wordb[1], "%d", &x);           /* Move parameters over */
  sscanf(wordb[2], "%d", &y);
  sscanf(wordb[3], "%d", &v);
  sscanf(wordb[4], "%d", &h);
  strncpy(title, wordb[5], 20);
  strncpy(deftxt, wordb[6], 20);
  strncpy(other, wordb[7], 20);
  strcpy(mess, wordb[8]);
  for (i=9; i<listsiz; i++) {
    strcat(mess, " ");
    strcat(mess, wordb[i]);
  }

  ask_window = uxwadd(x, y, v, h,       /* Create the ask window */
                      TRUE, on_damag, 0);

  uxwsho(ask_window);                   /* Show the ask window */
  uxwact(ask_window);                   /* Activate the ask window */
  if (strlen(title) + 12 < v)
    sprintf(mesg, "===== %s =====", title);
  else
    strcpy(mesg, title);
  uxwpus(ask_window, mesg,              /* print title string */
         (v - strlen(mesg))/2, 0, strlen(mesg));
  sprintf(mesg, mess, deftxt, other);
  uxwpus(ask_window, mesg, 1, 2, strlen(mesg));

  def_x = (v - strlen(deftxt)) / 2;
  def_y = 4 < h-2 ? 4 : h-2;
  uxwpus(ask_window, deftxt, def_x, def_y, strlen(deftxt));
  rcode = IAPSUCC;
```

Ask Question in a Window MJ

```
  alarm_cught = FALSE;
  oldsig = signal(SIGALRM, alarm_handle);/* Set the alarm signal handle */
                                         /* Save the old one           */

  for (;;) {
    uxwgxy(ask_window, def_x, def_y);
    alarm(TIMEOUT);                      /* Set the alarm clock to TIMEOUT sec */
    ch = toupper(getchar());
    if (alarm_cught) break;
    else if (ch == (char)13) break;
    else if (ch == toupper(deftxt[0])) break;
    else if (ch == toupper(other[0])) {
      rcode = IAPFAIL;
      break;
    } else {
      if (isprint(ch))
        sprintf(mesg, "Input %c is not recognised!", ch);
      else
        sprintf(mesg, "Input character is not recognised!");
      uxwpus(ask_window, mesg, def_x, def_y, strlen(mesg));
    }
  }
  alarm(0);                              /* Stop the alarm clock */
  signal(SIGALRM, oldsig);               /* Restore the old signal for SIGALRM */

  uxwrem(ask_window);                    /* Remove the ask window */

  return rcode;
}
```

A trigger using this user exit may look like this:

```
user_exit('WIND_ASK 20 7 40 7 Ask-Alert Yes No  Please enter %s or %s:');
if form_fatal then
 message('WIND_ASK USAGE: x y v h title default other mask');
elsif form_failure then
 message('User ansered No');
else
 message('User ansered Yes');
end if;
```

During the development of the user exit we found the following errors in the window management facility rutines from chapter 19: (Using Oracle*Tool 1.0.18.1 and SQL*Forms 3.0.15.3)

1. The *uxwadd* rutine takes the following parameters:

   ```
   uxwind uxwadd(x, y, v, h, border, damage_pointer, user_parameter)
     int x, y, v, h;
     bool border;
     void *damage_pointer();
     char *user_parameter;
   ```

   The user_parameter may be used to pass information to the *on_damage* procedure.

2. The *uxwpuc* rutine takes the following parameters:

```
int uxwpuc(window_pointer, window_char, x, y)
  uxwind window_pointer;
  char window_char;
  int x, y;
```

The x parameter seems to be replaced by the value 2 internally.

The example user exit at the end of chapter 19 will work if you adjust it like this (It cannot be a program since the declaration of *xokenv* and *xokcall* is missing):

```
#include <iapuxw.h>
#include <usrxit.h>

#define TRUE                1
#define FALSE               0

void a_pause()
{ getchar(); }

char *on_damag(a_window, x, y, h, v, a_param)
  uxwind a_window;
  int x, y, h, v;
  char *a_param;
{
  char tempstr1[10];
  char tempstr2[10];

  sprintf(tempstr1, "This is");
  sprintf(tempstr2, "window %s", a_param);
  uxwpus(a_window, tempstr1, 1, 1, 7);
  uxwpus(a_window, tempstr2, 1, 2, 8);
}

int win_exam()
{
  uxwind window_1, window_2;

  window_1 = uxwadd(5, 5, 12, 10, TRUE, on_damag, "1");
  window_2 = uxwadd(8, 8, 12, 10, TRUE, on_damag, "2");
  uxwsho(window_1);
  a_pause();

  uxwsho(window_2);
  a_pause();

  uxwpop(window_1);
  a_pause();

  uxwact(window_1);
  a_pause();

  uxwhid(window_1);
  a_pause();

  uxwhid(window_2);
  a_pause();
  return 0;
}
```

12.9 Get Current Date or Time

The standard way of getting the acutal date or time in a form is by assigning the value of *sysdate* to a field - or variable.

This will force the forms process to request the kernel for the actual date and time. If the actual date or time is used as input to database operations this is exactly the way you want it to happen.

If the date or time information is ment to be presented on the screen in non-database fields, the information is fetched more inexpensively by a local user exit - and the value will be local!

The userexit could look like this:

```
/* ************************************************************************
GETTIME wille get the actual local date and time, and store it in the
        <field-name> with the maximum length of <max-length>

GETTIME is called with the following syntax for the parameter string p:

USAGE (in forms 3.0): USER-EXIT('GETTIME field-name max-length')

20. Sep 91 - MJ - DDE
************************************************************************ */

#include <stdio.h>
#include <usrxit.h>
#include <time.h>

#define          MAXARGS                128
#define ARBUFSIZ 512

EXEC SQL BEGIN DECLARE SECTION;
                 VARCHAR formname [30]; /* form variable name */
                 VARCHAR varvalue [128];/* Variable to store value */
EXEC SQL END DECLARE SECTION;
EXEC SQL INCLUDE SQLCA.H;

char *wordb[MAXARGS];                   /* Holds pointers to the blank separated  */
                                        /* tokens in the string passed to GETENVR */
int gettime(p, paramlen, erm, ermlen, query)
  register char *p;                     /* Parameter string */
  int *paramlen;            /* Ptr to param string length */
  char *erm;                            /* Error message if doesnt match */
  int *ermlen;                          /* Ptr to error message length */
  int *query;                           /* Ptr to query status flag */
{
  register int listsiz;                 /* Number of values */
  register int i;                       /* Temp counter */
  char arbuf[ARBUFSIZ];                 /* To hold string that is passed */
  int maxlen, len;
  struct tm *tmptr;
  long   longtime;
  char monthnames[37];


  EXEC SQL WHENEVER SQLERROR GOTO sqlerr;
```

```
                                        and double spaces */

  strncpy(arbuf, p, ARBUFSIZ-1 );
  listsiz = countargs( arbuf );         /* get fieldname and value list
                                           in wordb[] */

  if ( listsiz != 3 )                   /* Check for 3 arguments */
    return IAPFTL;

  strncpy(formname.arr, wordb[1], 30);
  formname.len = strlen(formname.arr);

  sscanf(wordb[2], "%d", &maxlen);

  strcpy(monthnames,"JANFEBMARAPRMAYJUNJULAUGSEPOCTNOVDEC");
  longtime = time((long *)0);
  tmptr = localtime(&longtime);
  sprintf(varvalue.arr, "Date is: %.2d-%.3s-%.2d %.2d:%.2d:%.2d0,
    tmptr->tm_mday, monthnames+(tmptr->tm_mon)*3, tmptr->tm_year,
    tmptr->tm_hour, tmptr->tm_min, tmptr->tm_sec);

  varvalue.len= strlen(varvalue.arr);

  EXEC IAF PUT :formname values ( :varvalue );

  return IAPSUCC;

sqlerr:
  sqlca.sqlerrm.sqlerrmc[sqlca.sqlerrm.sqlerrml] = ' ';
  sqliem( sqlca.sqlerrm.sqlerrmc, &sqlca.sqlerrm.sqlerrml );
  return IAPFAIL;
}
```

Figure 17. The gettime.pc User Exit

13. Checklist for Error Documentation

13.1 Documentation of Oracle problems.

Whenever an error occurs you need to carefully investigate and describe the situation. This is true, whether you try to solve the problem in your own organisation or report it to your DDE Support Representative.

Our experience has shown that even in very different kinds of problems we often request very similar kinds of information to document the case. This chapter is an attempt to make a list of those main points of information and to outline a practical method for collecting this information - regardless of your being an expert or a novice to Oracle.

In many cases just producing the documentation will help you to solve the problem. Troubleshooting as such is the next step and is not covered in this chapter. Information on troubleshooting Oracle can be found in the *Supermax Oracle 6.0 Installation and User's Guide* appendices.

13.2 Checklist

- The error and the situation in which it occured
- Generel system information
- Oracle specific information
- Log files and Trace files
- Error codes and messages
- SQL*Net problems
- Information collection utilities
- Fill in and file an error report

13.2.1 The Error and the Situation in Which it Occured

An error situation arises when a user attempts to do some kind of operation that fails. For investigation you need to know

- which program was running (e.g. **SQL*Forms**, an **OCI** program, **sqldba**)?
- which operation was attempted (e.g. an **SQL INSERT** command, an **olon** call, the **startup** command)?
- what went wrong, what happened?
- which error messages did the user see?
- did other irregular incidents happen at the time of the error?

13.2.2 General System Information

Background information is necessary such as

- Possible contention problems: **CPU, Memory**, and **Disk I/O**
- Swap problems
- Operating system configuration: check **/usr/lib/errlog/log**
- Hard errors: check **/usr/lib/errlog/log**

Part of this information has to be collected using operating system utilities such as **sysdisp** and **sysstat**. For a further description, please refer to your *Supermax System V Reference Manual.*

13.2.2.1 Automatical Information Collection. Part of the information can be collected automatically by the script **sysinfo**, shipped with Oracle V6.0.30 or newer. Please refer to the *Supermax Oracle 6.0 Installation and User's Guide* for information on this and other administrative scripts.

13.2.3 Oracle Specific Information, Diagnose

13.2.3.1 Configuration Problems. Configuration problems are focused on issues such as

- file ownerships and permissions
- datafile, logfile and controlfiles
- ulimit, Maximum allowed Memory per proces

13.2.3.2 Diagnose. It is recommended to run the **diagnose** script, which can be invoked from the **sysadm packagemgmt** menu or as a stand-alone procedure. This script is likely to reveal configuration problems that affect the Oracle system as well as problems reported in the **errlog**.

13.2.3.3 The perms Utility In the **install** directory for each Oracle product you will find a so-called **perm** file. This file can be used by the standard **perms** utility to set a reasonable set of protections for the files shipped with that product.

Note: The **perms** utility is automatically executed for all Oracle products when running **diagnose**.

13.2.4 Log Files and Trace Files

13.2.4.1 Oracle Trace Files (Dump Files) Whenever an Oracle kernel proces discovers some abnormal condition, a trace file is produced. The standard location for these files is the **rdbms/log** directory (**dbs** for Oracle V5.1 and V6.0.26). Note however that the destination can be changed by entering a different pathname for the **init.ora** parameter **user_dumps_dest**.

The detached processes (**dbwr, lgwr, pmon, smon**, and **arch**) will each write a trace file at the time of instance startup. Abnormal termination of any of these processes will also be reported in a trace file. The destination of these trace files can be changed by entering a different pathname for the **init.ora** parameter **detached_dumps_dest**.

For a further description of trace files, please refer to the appendix on troubleshooting in the *Supermax Oracle 6.0 Installation and User's Guide.*

13.2.4.2 Collecting Trace Files. If some unusual condition occurs, the Oracle kernel will write a trace file containing a stack trace and possible error messages. Sometimes you can read error messages in the trace file which were never printed at the user's terminal. If you decide to file the situation as an error report to DDE, the trace files are an indispensable documentation.

For this reason we have developed another script, *ora6info*, which, among other things, will collect relevant trace files. This script can be invoked through the **collect** entry in the **sysadm packagemgmt** for Oracle 6 or as a stand-alone procedure.

13.2.4.3 Alert file and Audit file. All changes in the states of the oracle instance (mount, open, etc.) are reported in the file **rdbms/log/alert_*sid*.log**.

Logging of the actions performed by the **dbstop** script is done into the file **rdbms/log/audit_*sid*.log**.

13.2.5 Error Codes and Messages

Every error from an Oracle kernel proces or client program is followed by an error code and error message. The error messages is normally the main information on the error and carefully recorded error messages are the core of any error investigation or report. Looking up the error messages in the *ORACLE Error Messages and Codes* manual will often give you a useful suggestion on how to solve the problem.

13.2.5.1 Cascading Error Messages. Often one erroneous operation will result in other errors, and you will receive two or three error messages, one after the other. Such error "cascades" should be read bottom-up. This means, that the last error message printed is the error that has initiated all the other error messages in the cascade.

If the Oracle error is the result of a failed system call, the last line in the cascade is likely to contain the operating system **errno**. In this case you need to consult the "UNIX System Call Error Messages" chapter in the *Supermax System Administrator's Guide*.

*13.2.6 SQL*Net Problems*

Often problems with establishing a net connection on a TCP/IP network, are due to one of the following causes:

- The network is down
- The **orasrv** program is not running
- All available network channels are in use, either on the server or on the client side
- The server program has not been linked correctly
- The **Oracle** kernel program has not been linked correctly.

13.2.6.1 Which Modules to Link? Please notice that in order to get a SQL*Net TCP/IP connection working, you must relink your user program (such as SQL*Plus or a Pro*C program) as well as the Oracle kernel. The following should be included on both the client and the server side:

- **rdbms/lib/osntab.o** containing the driver entry routine called **osnttt**
- **rdbms/lib/libsqlnet.a** which must be merged with **tcp/lib/libtcp.a**

– the *socket* library, on the Supermax called **libstcp.a** and indicated to the linker by the **-lstcp** option.

This will normally all work correctly and automatically, provided you

1. Install Supermax TCP/IP correctly
2. Run the automatic installation **oracle.install** with you network driver product
3. Link your Oracle products using the shipped makefiles.

The latter can be done for the whole product range using either the **net/applmake** script or the centralized makefile **link.mk**. Refer to the *Supermax Oracle 6.0 Installation and User's Guide* for details on linking and installation.

13.2.7 Information Collection Utilities

For your convenience we have supplied some scripts with your Oracle V6.0 distribution to make information collection and troubleshooting easier. These scripts can be invoked from the **collect** entry in the **sysadm packagemgmt** menu for Oracle V6.0.

13.2.8 Fill in and File an Error Report

In case you decide to file an error report to the DDE Support Group, please include all the information you have collected with the error report form. Remember to allways give information on versions of the operating system, the Oracle kernel and the user program.

Files collected by the **collect** utilites can be included on a floppy or tape, and should be followed by a filled-in copy of the form at the back of the *Supermax Oracle 6.0 Installation and User's Guide.*

I. Index

!

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

W

Reader's comment form

`Supermax ORACLE 6.0 Hints`

Dansk Data Elektronik welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revisions.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvements, please indicate the topic, chapter and page number below:

Please send your comments to:

Oracle Group
Dansk Data Elektronik A/S
Herlev Hovedgade 199
DK-2730 Herlev
Denmark

phone: +45 42 84 50 11

Subscription Request form

for

Supermax ORACLE 6.0 Hints

Fill in this form if you want to receive new versions of this document automatically. Dansk Data Elektronik will charge you standard subscription fees, covering copying and mailing costs, if other conditions have not been agreed upon.

Company:

Address:

Contact Person:

We would like to receive ____ Copies of *The Supermax ORACLE 6.0 Hints* (Stock number 94003351), when ever updates to the document are made ready.

Date:

Signiture:

Please send this form to:

Support Group
Dansk Data Elektronik A/S
Herlev Hovedgade 199
DK-2730 Herlev
Denmark

phone: +45 42 84 50 11