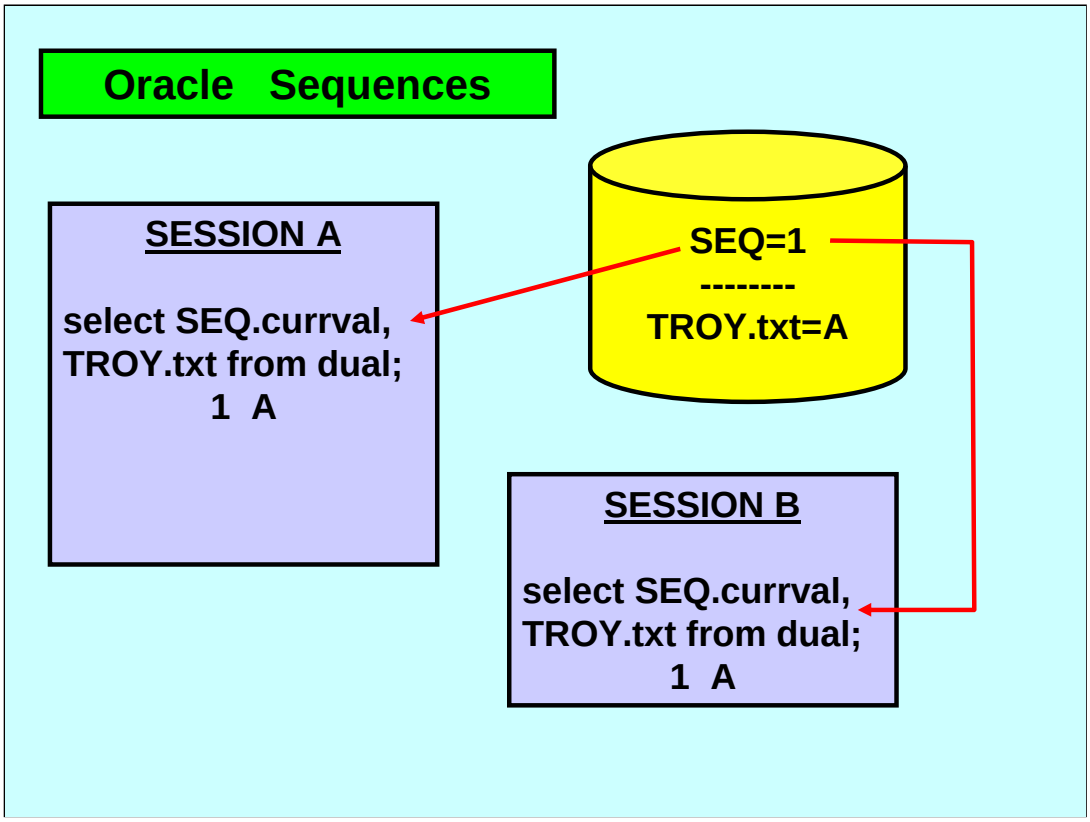

SUNCOAST
ORACLE
USERS' GROUP
Autonomous Transactions and Other Useful Tidbits
Troy Ligon
Ligon Solutions

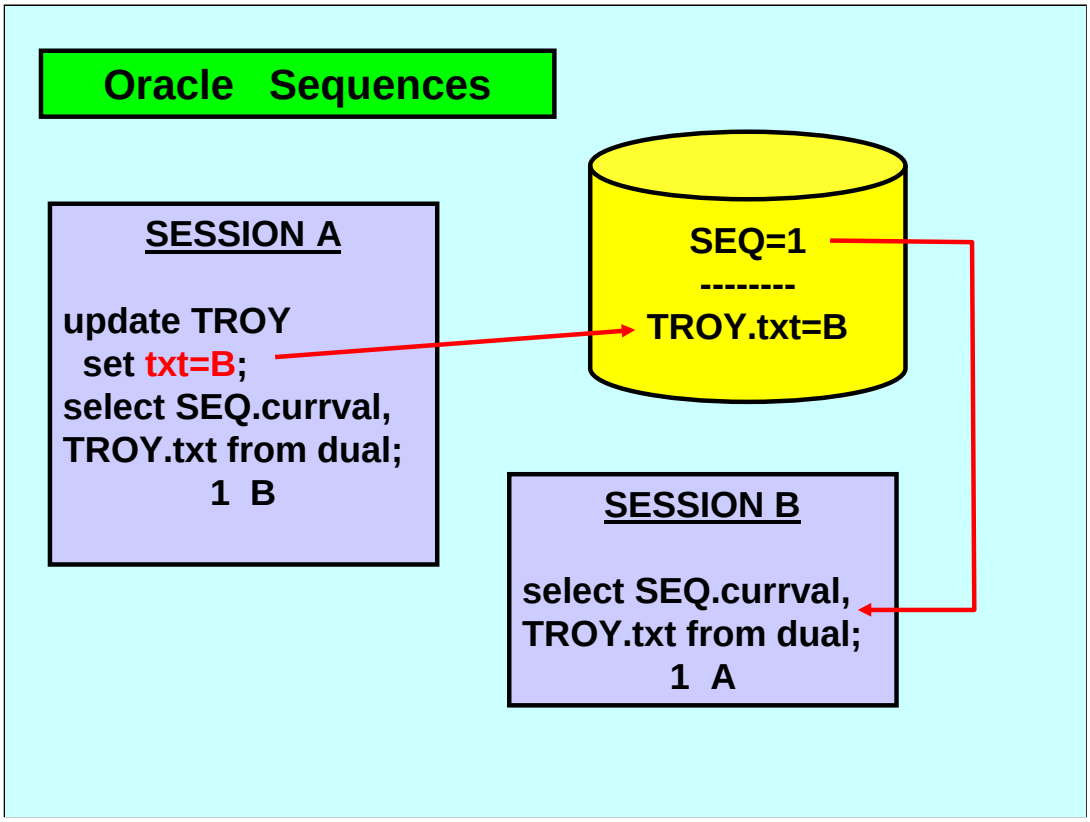
What is an **AUTONOMOUS TRANSACTION** and Why would I want to use one?

Have you ever found yourself in the situation where you want to commit some statements in a given transaction while rolling back other statements in that same transaction? Or you want to commit one particular statement and you're not sure yet whether you want to commit or roll back the rest of the statements in that transaction?

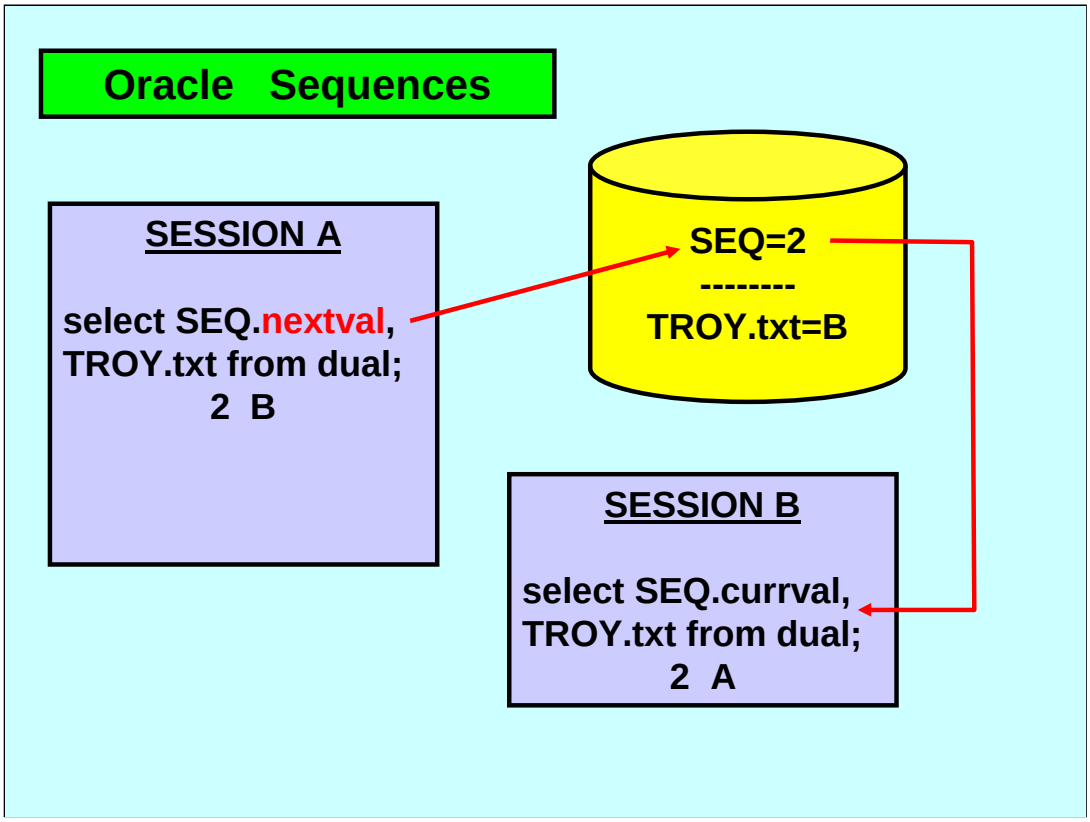
Well, you can think of Autonomous Transactions as this sort of "selective commit".



A good example of this is found when we take a look at Oracle Sequences.
Here we see a simple database with sequence SEQ = 1 and
table Troy column Txt = A
Both Session A and Session B see SEQ=1 and TXT=A



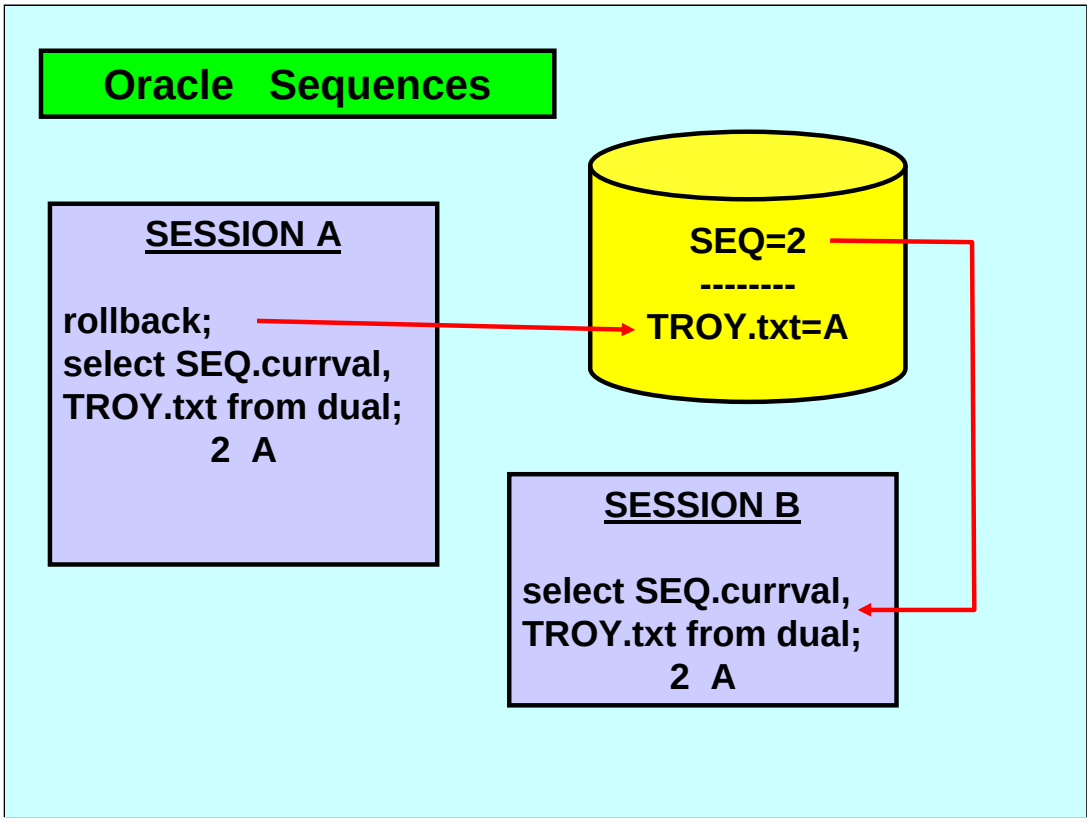
If Session A sets Troy.TXT = B, Session A will see the change, but Session B, of course, won't because the change hasn't been committed yet.



Anytime one session selects nextval from a sequence, the incremented value needs to be committed immediately so any other sessions will get the incremented value.

We see here Session A selecting nextval from the sequence, and both Session A and Session B immediately see SEQ=2

Note that Session B still doesn't see the update to TROY.txt because the sequence update was an autonomous transaction.

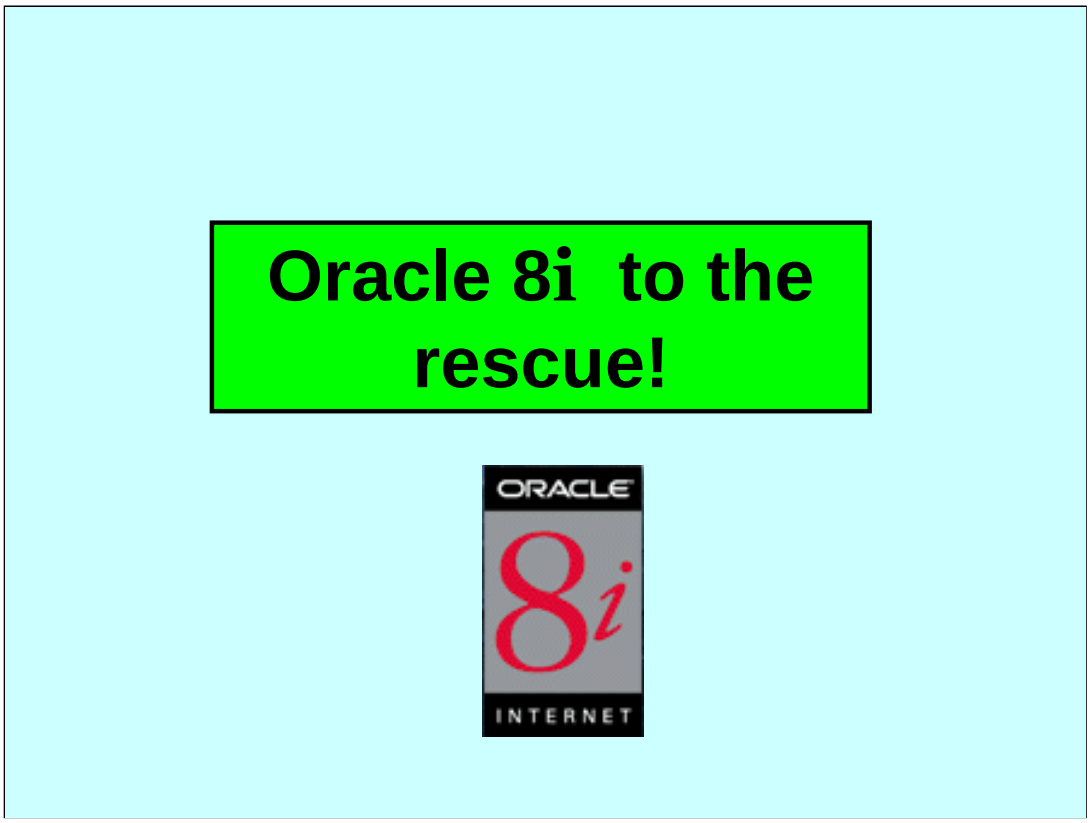


In fact, Session B might never see that change to TROY.TXT, because Session A might issue a ROLLBACK.

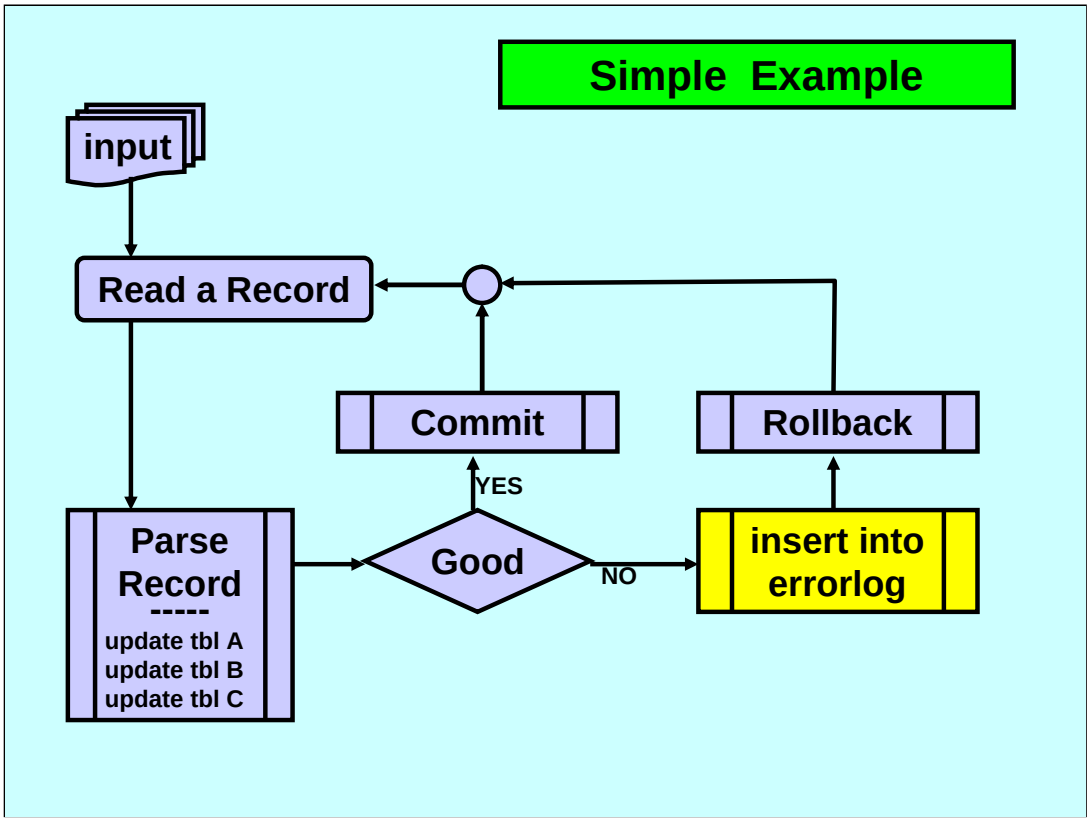
But as you can see here, the ROLLBACK doesn't effect the sequence. In fact, for many years, Oracle has used a lot of this type of functionality internally to update system resources and the like. You've probably seen them referred to as "recursive transactions" but they never really gave you a way to create your own recursive transactions.

How does this Help me ?

I know what you're thinking…"That's an interesting bit of trivia, Troy, but how does this help me?"



With the introduction of Autonomous Transactions in Oracle 8i, this type of functionality was finally opened up for our use. While it has been around for a long time now, it is such a useful, yet little known trick, that I thought I would present it to you here today. At the end of this presentation, I won't say that you'll be autonomous transaction experts, but you will have a concrete example that you should be able to take back with you and make immediate use of.



Here's a simple example that demonstrates the use of Autonomous Transactions. It's a typical transaction processing scenario:

Records are sitting in an INPUT table.

We need to parse each input record and update several other tables (A, B, & C)

As we process each input record, we check for errors.

If we find an error, we want to log that to our error table, rollback all of the changes for that record, and move on to the next INPUT record.

This insert into errorlog is what we need to be autonomous so the following rollback won't get rid of the errorlog record that we just inserted.

Simple Example

```
DECLARE
        CURSOR input_cur IS SELECT * FROM input_tbl;
        input_row   input_cur%ROWTYPE;

        PROCEDURE write_errorlog(message varchar2) IS
                PRAGMA AUTONOMOUS_TRANSACTION;
        BEGIN
                INSERT INTO errorlog_tbl VALUES(err_seq.nextval,message);
                COMMIT;
        END;
BEGIN
        FOR input_row IN input_cur LOOP
                update_table_A(input_row);
                update_table_B(input_row);
                update_table_C(input_row);
                IF input_is_good() THEN
                        COMMIT;
                ELSE
                        write_errorlog('bad record='||input_row.key);
                        ROLLBACK;
                END IF;
        END LOOP;
END;
```

The code for previous example would be something like the following…and I hope you'll forgive me if you can immediately see other ways to accomplish this…

this is admittedly a contrived example just to show the Autonomous Transaction functionality.

Simple Example

```
DECLARE
        CURSOR input_cur IS SELECT * FROM input_tbl;
        input_row   input_cur%ROWTYPE;

        PROCEDURE write_errorlog(message varchar2) IS
                PRAGMA AUTONOMOUS_TRANSACTION;
        BEGIN
                INSERT INTO errorlog_tbl VALUES(err_seq.nextval,message);
                COMMIT;
        END;
BEGIN
        FOR input_row IN input_cur LOOP
                update_table_A(input_row);
                update_table_B(input_row);
                update_table_C(input_row);
                IF input_is_good() THEN
                        COMMIT;
                ELSE
                        write_errorlog('bad record='||input_row.key);
                        ROLLBACK;
                END IF;
        END LOOP;
END;
```

We start off by declaring a cursor to read the input table and a corresponding variable to hold the current row.

Simple Example

```
DECLARE
        CURSOR input_cur IS SELECT * FROM input_tbl;
        input_row   input_cur%ROWTYPE;

        PROCEDURE write_errorlog(message varchar2) IS
                PRAGMA AUTONOMOUS_TRANSACTION;
        BEGIN
                INSERT INTO errorlog_tbl VALUES(err_seq.nextval,message);
                COMMIT;
        END;
BEGIN
        FOR input_row IN input_cur LOOP
                update_table_A(input_row);
                update_table_B(input_row);
                update_table_C(input_row);
                IF input_is_good() THEN
                        COMMIT;
                ELSE
                        write_errorlog('bad record='||input_row.key);
                        ROLLBACK;
                END IF;
        END LOOP;
END;
```

I've used a cursor FOR loop to step thru the input cursor.

Simple Example

```
DECLARE
        CURSOR input_cur IS SELECT * FROM input_tbl;
        input_row   input_cur%ROWTYPE;

        PROCEDURE write_errorlog(message varchar2) IS
                PRAGMA AUTONOMOUS_TRANSACTION;
        BEGIN
                INSERT INTO errorlog_tbl VALUES(err_seq.nextval,message);
                COMMIT;
        END;
BEGIN
        FOR input_row IN input_cur LOOP
                update_table_A(input_row);
                update_table_B(input_row);
                update_table_C(input_row);
                IF input_is_good() THEN
                        COMMIT;
                ELSE
                        write_errorlog('bad record='||input_row.key);
                        ROLLBACK;
                END IF;
        END LOOP;
END;
```

For each input record, I call update procedures to perform whatever database updates are needed.

I've omitted the details of these update procedures as well as the following input_is_bad() function as they're not relevant to this discussion.

Simple Example

```
DECLARE
        CURSOR input_cur IS SELECT * FROM input_tbl;
        input_row   input_cur%ROWTYPE;

        PROCEDURE write_errorlog(message varchar2) IS
                PRAGMA AUTONOMOUS_TRANSACTION;
        BEGIN
                INSERT INTO errorlog_tbl VALUES(err_seq.nextval,message);
                COMMIT;
        END;
BEGIN
        FOR input_row IN input_cur LOOP
                update_table_A(input_row);
                update_table_B(input_row);
                update_table_C(input_row);
                IF input_is_good() THEN
                        COMMIT;
                ELSE
                        write_errorlog('bad record='||input_row.key);
                        ROLLBACK;
                END IF;
        END LOOP;
END;
```

If the function decides the record is good then a commit is issued.

Simple Example

```
DECLARE
        CURSOR input_cur IS SELECT * FROM input_tbl;
        input_row   input_cur%ROWTYPE;

        PROCEDURE write_errorlog(message varchar2) IS
                PRAGMA AUTONOMOUS_TRANSACTION;
        BEGIN
                INSERT INTO errorlog_tbl VALUES(err_seq.nextval,message);
                COMMIT;
        END;
BEGIN
        FOR input_row IN input_cur LOOP
                update_table_A(input_row);
                update_table_B(input_row);
                update_table_C(input_row);
                IF input_is_good() THEN
                        COMMIT;
                ELSE
                        write_errorlog('bad record='||input_row.key);
                        ROLLBACK;
                END IF;
        END LOOP;
END;
```

Otherwise the write_errorlog procedure is called.

Simple Example

```
DECLARE
        CURSOR input_cur IS SELECT * FROM input_tbl;
        input_row   input_cur%ROWTYPE;

        PROCEDURE write_errorlog(message varchar2) IS
                PRAGMA AUTONOMOUS_TRANSACTION;
        BEGIN
                INSERT INTO errorlog_tbl VALUES(err_seq.nextval,message);
                COMMIT;
        END;
BEGIN
        FOR input_row IN input_cur LOOP
                update_table_A(input_row);
                update_table_B(input_row);
                update_table_C(input_row);
                IF input_is_good() THEN
                        COMMIT;
                ELSE
                        write_errorlog('bad record='||input_row.key);
                        ROLLBACK;
                END IF;
        END LOOP;
END;
```

The trick is all in the “PRAGMA AUTONOMOUS_TRANSACTION” declaration on the procedure.

When Oracle sees this pragma, it suspends the previous transaction and starts a new transaction for this procedure.

Simple Example

```
DECLARE
        CURSOR input_cur IS SELECT * FROM input_tbl;
        input_row   input_cur%ROWTYPE;

        PROCEDURE write_errorlog(message varchar2) IS
                PRAGMA AUTONOMOUS_TRANSACTION;
        BEGIN
                INSERT INTO errorlog_tbl VALUES(err_seq.nextval,message);
                COMMIT;
        END;
BEGIN
        FOR input_row IN input_cur LOOP
                update_table_A(input_row);
                update_table_B(input_row);
                update_table_C(input_row);
                IF input_is_good() THEN
                        COMMIT;
                ELSE
                        write_errorlog('bad record='||input_row.key);
                        ROLLBACK;
                END IF;
        END LOOP;
END;
```

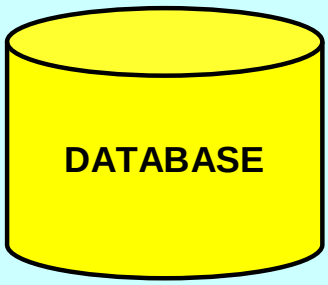
When this procedure ends, it returns us to the previous transaction, which I then rollback.
But since the insert into errorlog was autonomous, it survives while all other changes are rolled back.

You can review your errorlog at this point with a simple “select * from errorlog_tbl order by 1;”.
By issuing a “truncate errorlog_tbl;” you will be ready for the next test run of your procedure.

Do you ever get those calls from your customer where they say:

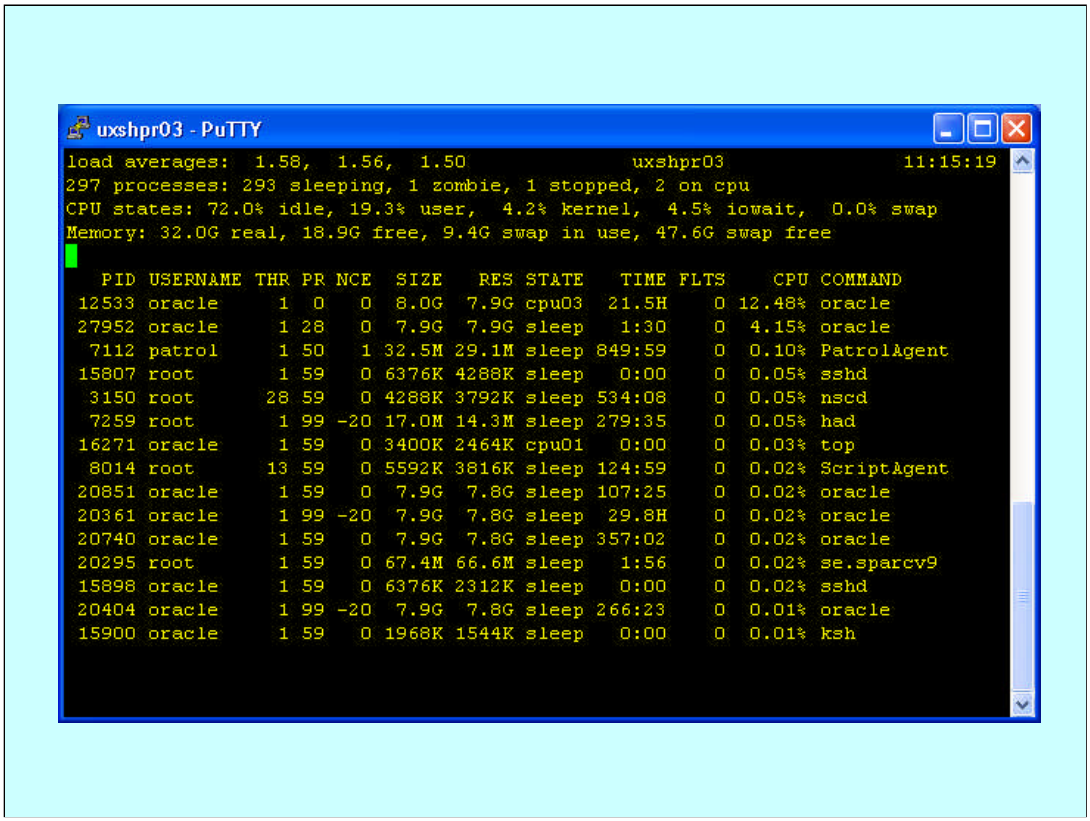
"How's the database looking ?"

“Round and kinda can-like on my screen, how’s it looking on yours?”



I like to respond with something along these lines, just to set the tone. You give me a silly question, you’re likely to get a silly answer. Of course what they are really asking is “Does it look like the database is performing properly or is it having any problems?” As in performance problems most often.

If I really want to know for sure, I fire up my trusty Quest Spotlight....but this takes a lot of time to get connected and up and running. Often times this is overkill. It's Oracle, so the database is never the problem, right? So what I really need is just a quick perk check.



So a quicker option is to fire up a putty session to login to the Unix box and do a “top” command. This is great for what it is....lots of information on what’s going on right now. But it doesn’t address the question “is this normal for this environment?” If you’re managing one or two boxes, maybe you’ll know, but if you have hundreds of boxes, it’s a much harder question. What you really need is some trend data.

How do I

Monitor Unix Servers?

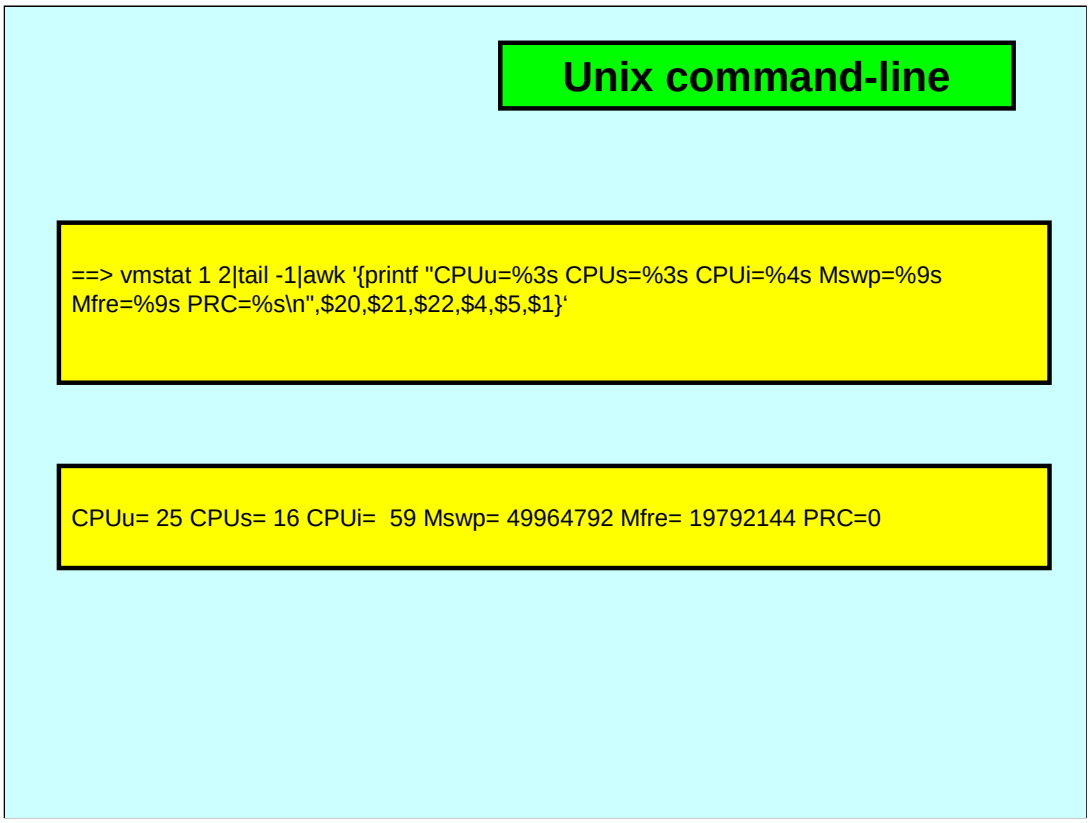
What Percent CPU & Memory?
How many DB Sessions?

So here’s the bigger question

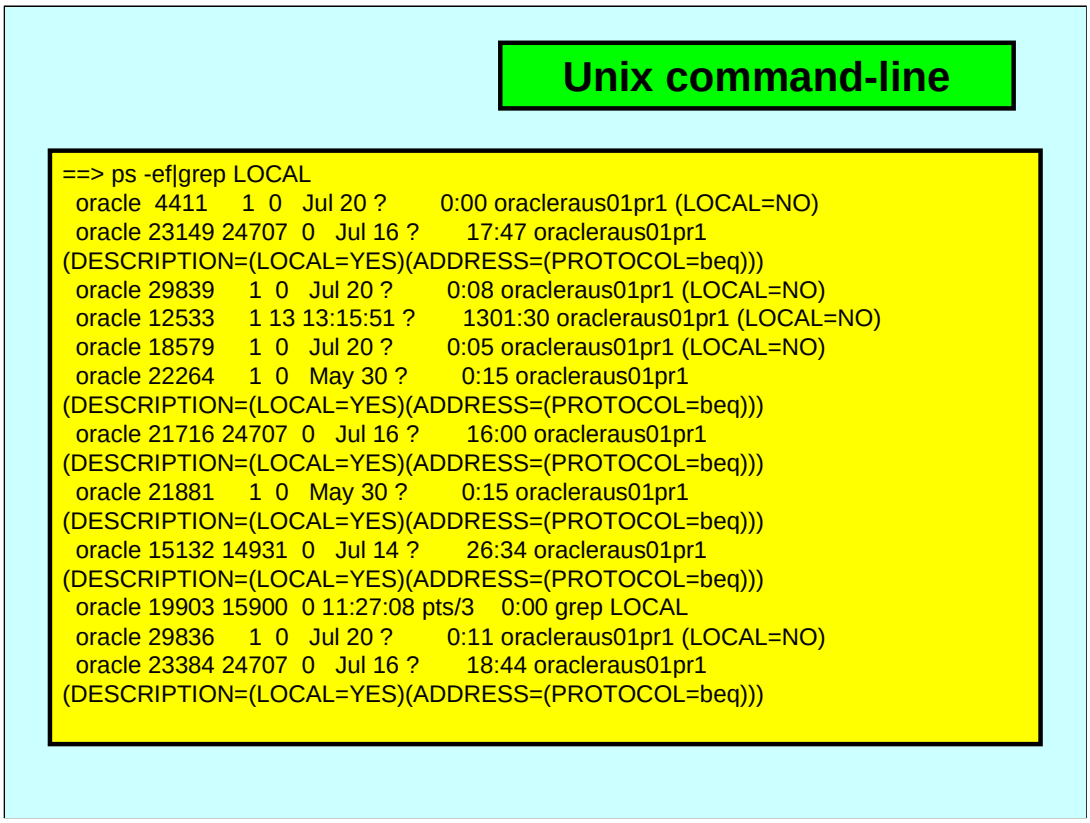
Unix command-line

```
==> vmstat 1 10
 kthr      memory            page            disk          faults      cpu
 r b w   swap  free  re  mf pi po fr de sr m0 m1 m2 m1   in   sy   cs us sy id
 0 1 0 49808048 19958704 464 663 48 4 3 0 0 0  0  0  0 1406 2287 2458 10  6 85
 0 0 0 49984544 19837576 1918 17242 0 0 0 0 0 0 0 0  0 2066 34780 4795 16 12 72
 0 0 0 49984544 19837576 1541 11336 16 0 0 0 0 0 0 0 0 2755 36918 5316 15 7 78
 0 0 0 49984688 19837576 643 4050 8 0 0 0 0 0  0  0  0 1457 12475 3206 14 2 84
 0 0 0 49984624 19837552 1670 6965 8 0 0 0 0 0 0  0  0 1345 18347 3075 17 3 80
 0 0 0 49982136 19833632 1665 5956 8 0 0 0 0 0 0  0  0 2122 21883 4028 18 2 80
 0 0 0 49984624 19837552 299 2598 8 0 0 0 0 0  0  0  0 1255 8268 2387 13  2 86
 0 0 0 49984624 19837552 307 2787 8 0 0 0 0 0  0  0  0 2308 18512 2930 16 5 79
 0 0 0 49980272 19836488 302 2727 8 0 0 0 0 0  0  0  0 2015 20505 3101 20 4 76
 0 1 0 49977400 19835472 364 3195 8 8 0 0 0 0  0  0  0 1857 27979 3541 22 2 76
```

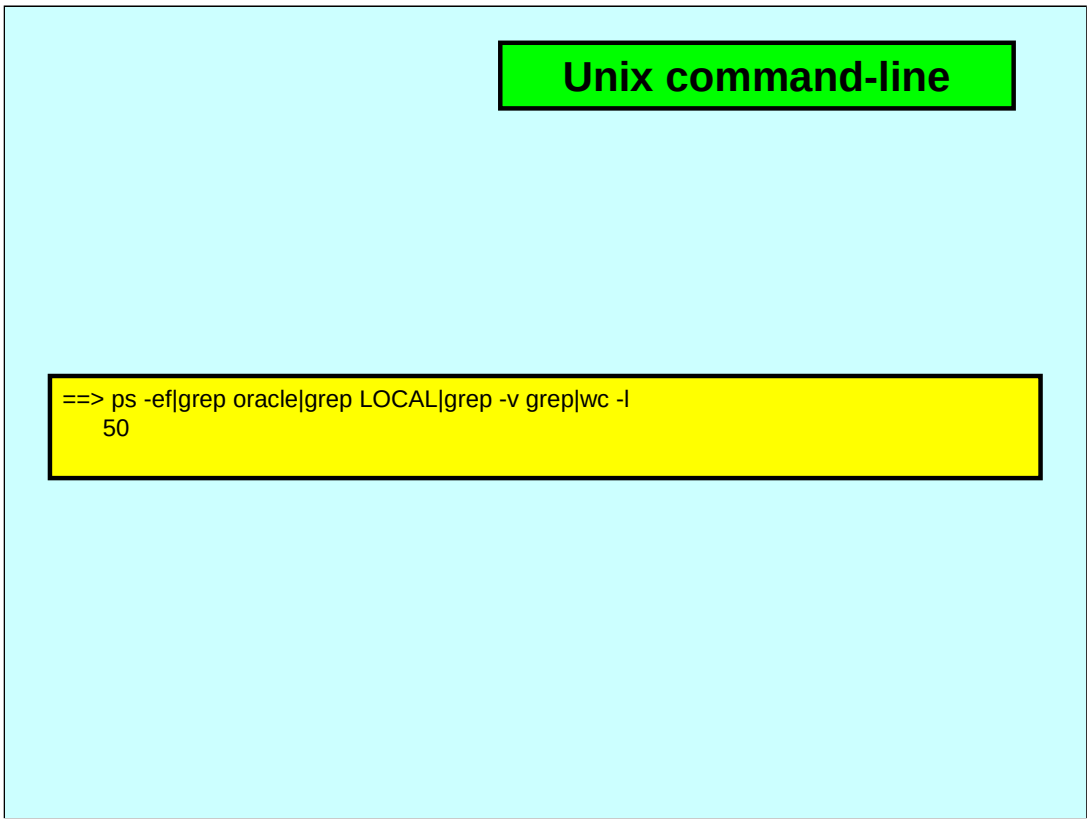
vmstat gives cpu and memory...but it’s kind of tedious to decode....the columns aren’t lined up with the headers, etc.



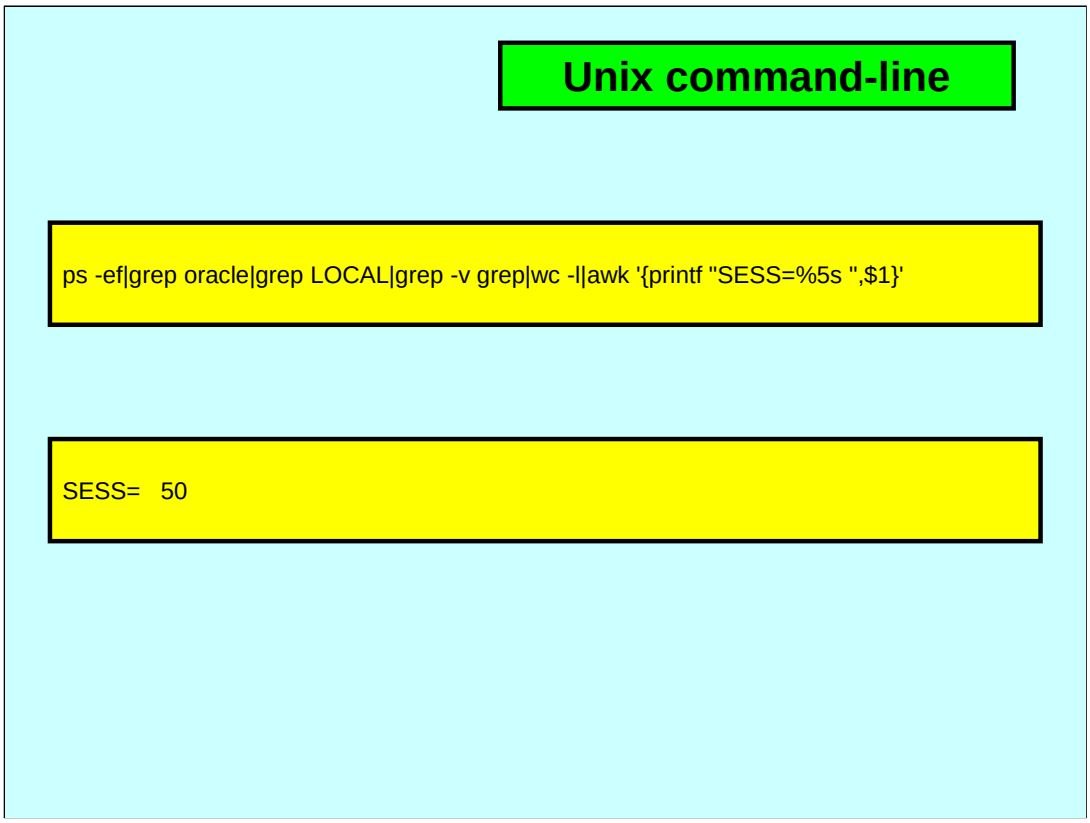
throw in a bit of formatting via AWK and we have pretty passable output



Now for the session count...the process status command lists all the processes and grep LOCAL will filter for just the oracle session processes ... but you were really just looking for the count so you pipe this to WC –L to count the lines of output...and note you also got a line for your GREP so you'd want to either subtract one or GREP –V to get rid of that.



so rolling all that together, something like this is what you're really after.....add a bit of AWK formatting and you get:



all of this is very doable, but if I'm doing it by hand, it's a lot of typing...and a lot of typo's if you're like me...and we started out wanting a quick and easy solution that is faster than firing up Spotlight....or available to those of you who haven't talked their company's finance officer into springing for the cost of Spotlight....

twl_monitor.sh

```
date '+%H:%M'|awk '{printf "%s ",$1}' ;ps -ef|grep
oracle|grep LOCAL|grep -v grep|wc -l|awk '{printf
"SESS=%5s ",$1}' ;vmstat 1 2|tail -1|awk '{printf
"CPUu=%3s CPUs=%3s CPUi=%4s Mswp=%9s Mfre=%9s
PRC=%s\n",$20,$21,$22,$4,$5,$1}'
```

```
12:50 SESS=   50 CPUu= 17 CPUs= 10 CPUi=  73 Mswp= 49985320 Mfre= 19795416 PRC=0
```

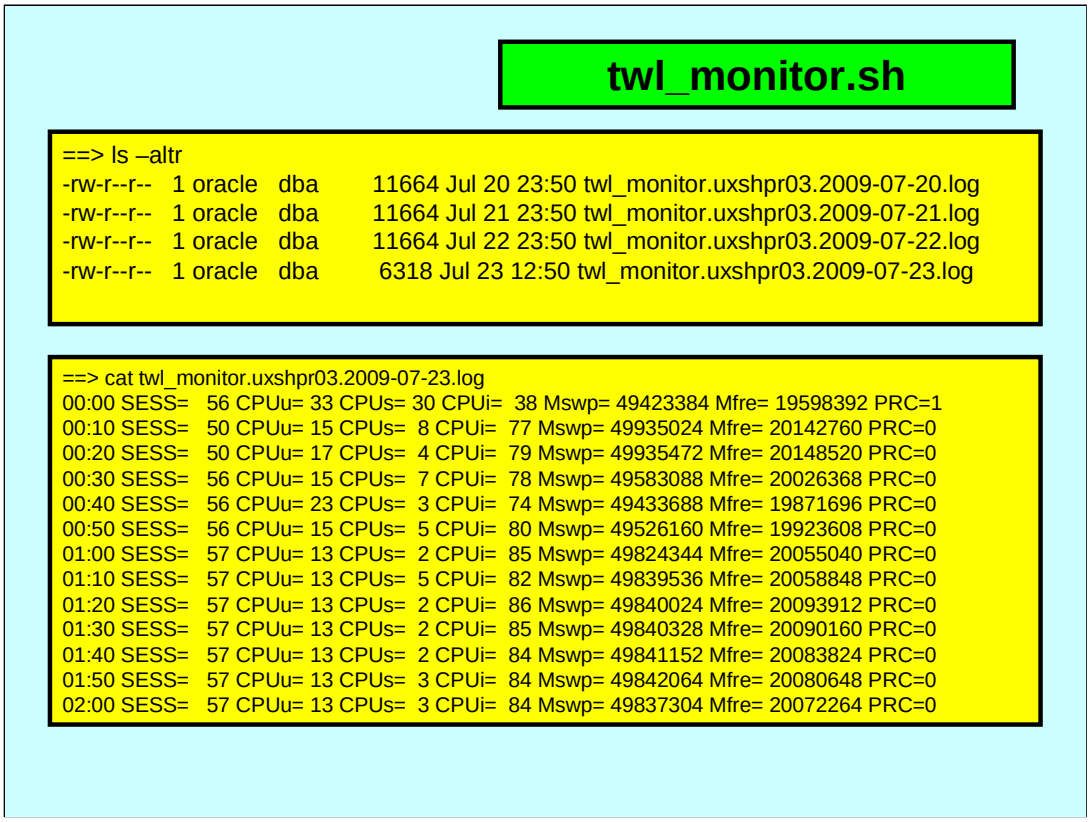
Enter twl_monitor.sh.... I found myself doing this over and over so I just put all of it in a little one line script so all I needed to do was login to the box and run it. I added a timestamp as well, because a common question was “when was this?”.....very handy, but it still didn’t answer the question “is this normal for this box?”

twl_monitor.sh

```
PATH=/usr/ccs/bin:/usr/sbin:/opt/bin:/usr/bin/:/var:/usr/local/bin:/usr/ucb
export PATH
cd `dirname $0`
host=`hostname`
dat=`date +%Y-%m-%d`
date '+%H:%M'|awk '{printf "%s ",$1}' >>
/opt/app/oracle/logs/twl_monitor.$host.$dat.log
ps -ef|grep oracle|grep LOCAL|grep -v grep|wc -l|awk '{printf "SESS=%5s ",$1}'
>> /opt/app/oracle/logs/twl_monitor.$host.$dat.log
vmstat 1 2|tail -1|awk '{printf "CPUu=%3s CPUs=%3s CPUi=%4s Mswp=%9s
Mfre=%9s PRC=%s\n",$20,$21,$22,$4,$5,$1}' >>
/opt/app/oracle/logs/twl_monitor.$host.$dat.log
```

```
#
# monitoring:
00,10,20,30,40,50 * * * * /opt/app/oracle/scripts/twl_monitor.sh 2>&1 >/dev/null
```

So I added a bit of code to redirect the output to a file and just append to that every time I ran it. That way I could look at the file and see what the previous runs had said and know whether this was normal or not. It then occurred to me that I could just put it in the cron to run every 10 minutes and I'd have a full dataset. By including the date in the filename, it would automagically give me one file per day with 24hours of data in each one.



This was great. Now I had a directory on the server where I could go and just cat the latest file to see what today's trend was. Or any of the older files to see what it was like last week, last month, whenever.

twl_monitor_mail.sh

```
# command-line parameter is email address to receive report
#
PATH=/usr/ccs/bin:/usr/sbin:/opt/bin:/usr/bin/:/var:/usr/local/bin:/usr/ucb
export PATH
cd `dirname $0`
host=`hostname`
dat=`date +%Y-%m-%d`
# wait till after midnight so file is no longer being updated
sleep 180
cat /opt/app/oracle/logs/twl_monitor.$host.$dat.log | /usr/bin/mailx -v -s
"twl_monitor Stats for $host on $dat" $1
```

```
#
# email monitoring logs
58 23 * * * /opt/app/oracle/scripts/twl_monitor_mail_attach.sh troy@ligonweb.com  2>&1 >/dev/null
```

Next I got too lazy to even login to the Unix box, so I made it email it to me each night. Date math is tricky in a shell script os I used a clever trick of doing it 2 minutes before midnight to capture the DATE and sleep 3 minutes so you get a full file without having to calculate yesterday's date. Note $1 is the email address on the command-line. Even lazier, I can include the customer's email (comma separated) and the phone call doesn't even occur. It becomes a customer self-service activity. And the only thing they need to be able to do is read their email....

twl_monitor_mail_attach.sh

```
# command-line parameter is email address to receive report
#
PATH=/usr/ccs/bin:/usr/sbin:/opt/bin:/usr/bin/:/var:/usr/local/bin:/usr/ucb
export PATH
cd `dirname $0`
host=`hostname`
dat=`date +%Y-%m-%d`
# wait till after midnight so file is no longer being updated
sleep 180
/usr/bin/mailx -v -s "twl_monitor Stats for $host on $dat" $1 <<EOF
~<!uuencode /opt/app/oracle/logs/twl_monitor.$host.$dat.log
/opt/app/oracle/logs/twl_monitor.$host.$dat.txt
~.
EOF
```

Here’s another version that makes it a text attachment instead of being just the body of the email.

What is the Most Useful Item I've recently stumbled across in the Data Dictionary?

DBA_HIST_SYSMETRIC_SUMMARY

I was working on an enhancement to a script I've written that drives off of the AWR data and was poking around in the Data Dictionary looking for a specific piece of data. In my browsing, I happened to take a look at this view and was floored.

dba_hist_sysmetric_summary

```
Name                                      Null?    Type
----------------------------------------- -------- ------------
SNAP_ID                                   NOT NULL NUMBER
DBID                                      NOT NULL NUMBER
INSTANCE_NUMBER                           NOT NULL NUMBER
BEGIN_TIME                                NOT NULL DATE
END_TIME                                  NOT NULL DATE
INTSIZE                                   NOT NULL NUMBER
GROUP_ID                                  NOT NULL NUMBER
METRIC_ID                                 NOT NULL NUMBER
METRIC_NAME                               NOT NULL VARCHAR2(64)
METRIC_UNIT                               NOT NULL VARCHAR2(64)
NUM_INTERVAL                              NOT NULL NUMBER
MINVAL                                    NOT NULL NUMBER
MAXVAL                                    NOT NULL NUMBER
AVERAGE                                   NOT NULL NUMBER
STANDARD_DEVIATION                        NOT NULL NUMBER
```

This view provides aggregated data for a variety of system and database metrics. It has pre-calculated Minimum, Maximum, and Average values and even includes the Standard Deviation and number of samples

dba_hist_sysmetric_summary

```
select count(distinct metric_name) from dba_hist_sysmetric_summary order by 1;

COUNT(DISTINCTMETRIC_NAME)
--------------------------
                       135
```

Current Logons Count
Host CPU Utilization (%)
Database CPU Time Ratio
Database Wait Time Ratio
Process Limit %
Logical Reads Per Sec
Network Traffic Volume Per Sec
Physical Read Total Bytes Per Sec
Physical Read Total IO Requests Per Sec
Physical Reads Per Sec
Physical Write Total Bytes Per Sec
Physical Write Total IO Requests Per Sec
Physical Writes Per Sec

It has 135 different metrics summarized for you in 10.2. Here are some of the ones most interesting to me.

dba_hist_sysmetric_summary

```
set pagesize 9999
set linesize 1000 trimsp on
col dbname          format a10     heading 'DBNAME'
col timestamp       format a16
col snap_id         format 999999  heading 'SNAPID'
col instance_number format 99      heading 'NODE'
col ses1            format 9999
col ses2            format 9999
col ses3            format 9999
col ses4            format 9999
col cpu1            format 9999
col cpu2            format 9999
col cpu3            format 9999
col cpu4            format 9999
col dbwait1         format 9999
col dbwait2         format 9999
col dbwait3         format 9999
col dbwait4         format 9999
col dbcpu1          format 9999
col dbcpu2          format 9999
col dbcpu3          format 9999
col dbcpu4          format 9999
```

set up column formatting

dba_hist_sysmetric_summary

```
select distinct b.value dbname, to_char(a.begin_time,'YYYY-MM-DD HH24:MI') timestamp,a.snap_id
  ,(select average from dba_hist_sysmetric_summary where instance_number=1 and
metric_name='Current Logons Count'      and snap_id=a.snap_id and dbid=a.dbid) ses1
,(select average from dba_hist_sysmetric_summary where instance_number=1 and
metric_name='Host CPU Utilization (%)' and snap_id=a.snap_id and dbid=a.dbid) cpu1
,(select average from dba_hist_sysmetric_summary where instance_number=1 and
metric_name='Database Wait Time Ratio' and snap_id=a.snap_id and dbid=a.dbid) dbwait1
,(select average from dba_hist_sysmetric_summary where instance_number=1 and
metric_name='Database CPU Time Ratio'  and snap_id=a.snap_id and dbid=a.dbid) dbcpu1
from dba_hist_sysmetric_summary a,dba_hist_parameter b
where a.instance_number=1
and a.snap_id=b.snap_id
and a.dbid=b.dbid
and b.parameter_name='db_name'
and a.snap_id > ((select max(snap_id) from dba_hist_snapshot where dbid=a.dbid)-(24*30))
order by a.snap_id;
```

give me the last 30 days sessions counts, cpu, dbwaits, and dbcpu from node 1 of the RAC

dba_hist_sysmetric_summary

DBNAME	TIMESTAMP	SNAPID	SES1	CPU1	DBWAIT1	DBCPU1
raus01pr	2009-06-23 13:02	24777	118	8	22	78
raus01pr	2009-06-23 14:01	24778	114	11	48	52
raus01pr	2009-06-23 15:01	24779	122	13	51	49
raus01pr	2009-06-23 16:01	24780	122	24	48	52
raus01pr	2009-06-23 17:01	24781	116	21	28	72
raus01pr	2009-06-23 18:02	24782	109	14	28	72
raus01pr	2009-06-23 19:01	24783	107	14	25	75
raus01pr	2009-06-23 20:01	24784	114	12	26	74
raus01pr	2009-06-23 21:02	24785	114	9	24	76
raus01pr	2009-06-23 22:02	24786	112	9	19	81
raus01pr	2009-06-23 23:01	24787	112	13	69	31
raus01pr	2009-06-24 00:01	24788	118	12	37	63
raus01pr	2009-06-24 01:02	24789	119	12	68	32
raus01pr	2009-06-24 02:01	24790	109	13	69	31
raus01pr	2009-06-24 03:02	24791	117	12	40	60
raus01pr	2009-06-24 04:01	24792	121	11	46	54
raus01pr	2009-06-24 05:01	24793	123	8	33	67
.....						

give me the last 30 days sessions counts, cpu, dbwaits, and dbcpu from node 1 of the RAC

dba_hist_sysmetric_summary

```
set pagesize 9999
set linesize 1000 trimsp on
col dbname          format a10     heading 'DBNAME'
col timestamp       format a16
col snap_id         format 999999  heading 'SNAPID'
col instance_number format 99      heading 'NODE'
col ses1            format 9999
col ses2            format 9999
col ses3            format 9999
col ses4            format 9999
col cpu1            format 9999
col cpu2            format 9999
col cpu3            format 9999
col cpu4            format 9999
col dbwait1         format 9999
col dbwait2         format 9999
col dbwait3         format 9999
col dbwait4         format 9999
col dbcpu1          format 9999
col dbcpu2          format 9999
col dbcpu3          format 9999
col dbcpu4          format 9999
```

so I’m sure some of you are wondering why I set up column formatting four times for each column....because I have a 4 node RAC

dba_hist_sysmetric_summary

```
select distinct b.value dbname, to_char(a.begin_time,'YYYY-MM-DD HH24:MI') timestamp,a.snap_id
  ,(select average from dba_hist_sysmetric_summary where instance_number=1 and
metric_name='Current Logons Count'     and snap_id=a.snap_id and dbid=a.dbid) ses1
  ,(select average from dba_hist_sysmetric_summary where instance_number=2 and
metric_name='Current Logons Count'     and snap_id=a.snap_id and dbid=a.dbid) ses2
  ,(select average from dba_hist_sysmetric_summary where instance_number=3 and
metric_name='Current Logons Count'     and snap_id=a.snap_id and dbid=a.dbid) ses3
  ,(select average from dba_hist_sysmetric_summary where instance_number=4 and
metric_name='Current Logons Count'     and snap_id=a.snap_id and dbid=a.dbid) ses4
  ,(select average from dba_hist_sysmetric_summary where instance_number=1 and
metric_name='Host CPU Utilization (%)' and snap_id=a.snap_id and dbid=a.dbid) cpu1
  ,(select average from dba_hist_sysmetric_summary where instance_number=2 and
metric_name='Host CPU Utilization (%)' and snap_id=a.snap_id and dbid=a.dbid) cpu2
  ,(select average from dba_hist_sysmetric_summary where instance_number=3 and
metric_name='Host CPU Utilization (%)' and snap_id=a.snap_id and dbid=a.dbid) cpu3
  ,(select average from dba_hist_sysmetric_summary where instance_number=4 and
metric_name='Host CPU Utilization (%)' and snap_id=a.snap_id and dbid=a.dbid) cpu4
```

Here I came up with a very scalable format for the sql statement. This makes it easy to add/remove/reorder various metrics totally independent of the rest of the sql.

dba_hist_sysmetric_summary

```
,(select average from dba_hist_sysmetric_summary where instance_number=1 and
metric_name='Database Wait Time Ratio' and snap_id=a.snap_id and dbid=a.dbid) dbwait1
  ,(select average from dba_hist_sysmetric_summary where instance_number=2 and
metric_name='Database Wait Time Ratio' and snap_id=a.snap_id and dbid=a.dbid) dbwait2
  ,(select average from dba_hist_sysmetric_summary where instance_number=3 and
metric_name='Database Wait Time Ratio' and snap_id=a.snap_id and dbid=a.dbid) dbwait3
  ,(select average from dba_hist_sysmetric_summary where instance_number=4 and
metric_name='Database Wait Time Ratio' and snap_id=a.snap_id and dbid=a.dbid) dbwait4
  ,(select average from dba_hist_sysmetric_summary where instance_number=1 and
metric_name='Database CPU Time Ratio'  and snap_id=a.snap_id and dbid=a.dbid) dbcpu1
  ,(select average from dba_hist_sysmetric_summary where instance_number=2 and
metric_name='Database CPU Time Ratio'  and snap_id=a.snap_id and dbid=a.dbid) dbcpu2
  ,(select average from dba_hist_sysmetric_summary where instance_number=3 and
metric_name='Database CPU Time Ratio'  and snap_id=a.snap_id and dbid=a.dbid) dbcpu3
  ,(select average from dba_hist_sysmetric_summary where instance_number=4 and
metric_name='Database CPU Time Ratio'  and snap_id=a.snap_id and dbid=a.dbid) dbcpu4
from dba_hist_sysmetric_summary a,dba_hist_parameter b
where a.instance_number=1
and a.snap_id=b.snap_id
and a.dbid=b.dbid
and b.parameter_name='db_name'
and a.snap_id > ((select max(snap_id) from dba_hist_snapshot where dbid=a.dbid)-(24*30))
order by a.snap_id;
```

dba_hist_sysmetric_summary

```
DBNAME     TIMESTAMP         SNAPID  SES1  SES2  SES3  SES4  CPU1  CPU2  CPU3  CPU4
DBWAIT1 DBWAIT2 DBWAIT3 DBWAIT4 DBCPU1 DBCPU2 DBCPU3 DBCPU4
---------- ---------------- -------- ----- ----- ----- ----- ----- ----- ----- ----- --
----- ------- -------- -------- ------ ------ ------ ------
raus01pr   2009-06-23 13:02   24777   118    46    90    55     8    20     5    17
22       0       22       30     78    100     78     70
raus01pr   2009-06-23 14:01   24778   114    47    97    56    11    20    11    18
48       5       39       33     52     95     61     67
raus01pr   2009-06-23 15:01   24779   122    63   117    74    13    26    20    24
51      65       83       77     49     35     17     23
raus01pr   2009-06-23 16:01   24780   122    55   104    67    24    24    19    24
48      34       58       55     52     66     42     45
raus01pr   2009-06-23 17:01   24781   116    46   144    57    21    20    13    22
28       3       45       32     72     97     55     68
raus01pr   2009-06-23 18:02   24782   109    46    82    55    14    20    12    22
28       1       27       32     72     99     73     68
raus01pr   2009-06-23 19:01   24783   107    45    81    54    14    20    11    21
25       0       32       32     75    100     68     68
raus01pr   2009-06-23 20:01   24784   114    48    91    56    12    22     9    19
26       5       32       31     74     95     68     69
raus01pr   2009-06-23 21:02   24785   114    46    84    55     9    24    10    17
24       4       31       31     76     96     69     69
raus01pr   2009-06-23 22:02   24786   112    46    77    56     9    11    14    17
19      38       30       30     81     62     70     70
raus01pr   2009-06-23 23:01   24787   112    47    80    52    13     8     6    18
69      28       34       30     31     72     66     70
.....
```

give me session counts, cpu, dbwaits, and dbcpu from all four nodes of the RAC, all on one line to facilitate graphing via Excel

Autonomous Transactions are as simple as that. I'd be happy to try and answer any questions that you might have. If there is something that I can't answer, I'd recommend that font of all Oracle knowledge, TechNet.