Creating Tests

Jonathan Lewis

jonathanlewis.wordpress.com

www.jlcomp.demon.co.uk

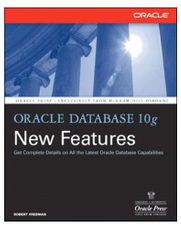
Who am I ?

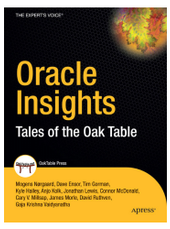
Independent Consultant

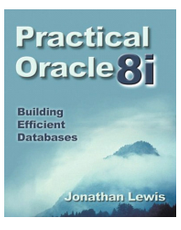
28+ years in IT
24+ using Oracle

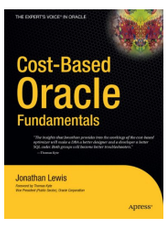
Strategy, Design, Review, Briefings, Educational, Trouble-shooting

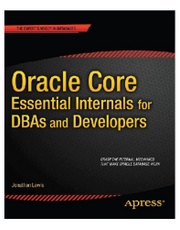
Member of the Oak Table Network
Oracle *ACE Director*
Oracle author of the year 2006
Select Editor's choice 2007
UKOUG Inspiring Presenter 2011
ODTUG 2012 Best Presenter (d/b)
O1 visa for USA











Requirements

- Volume of data
- Loading the data
- Patterns in data
- Checking data
- Indexing issues
- Concurrency issues
- Referential Integrity

Data Volume - ancient

```
select rownum id
from   all_objects   -- where rownum <= 3000
;
```

Minimal install on 8i ca. 4,000 rows
Full install on 11.2 ca. 80,000

```
select
        rownum  id
from    (select /*+ no_merge */ rownum id from all_objects) v1
        (select /*+ no_merge */ rownum id from all_objects) v2
where
        rownum <= 10000000     -- 10M
;
```

Data Volume - modern

```
select rownum n
from   dual
connect by
       level <= 10000
;
```

```
select rownum n
from   dual
connect by
       rownum <= 10000
;
```

```
with v1 as (select rownum n from dual connect by level <= 10000)
select
       rownum  id
from
       v1, v1
where
       rownum <= 100000000
;
```


Data Volume - comparison

```
select  rownum  n
from    dual
connect by
       level <= 1e6
;
```

9.38 seconds
PGA memory max 300 **MB**

```
with v1 as (select rownum n from dual connect by level <= 10000)
select
       rownum  id
from
       v1, v1
where
       rownum <= 1e6
;
```

4.16 seconds
PGA memory max 450 **KB**

Loading Data

```
create table t1 as
with v1 as (select rownum n from dual connect by level <= 10000)
select  rownum  id
from    v1, v1
where   rownum <= 1e6
;
```

(Noarchivelog mode)
4.16 seconds
Redo size 236 **KB**

```
create table t1 (id number);

insert into t1
with v1 as (select rownum n from dual connect by level <= 10000)
select  rownum  id
from    v1, v1
where   rownum <= 1e6
;
```

4.90 seconds
Redo size 160 **MB**
(Use append to speed it up)

Loading Data

```
create table t1 (id number);
begin
   for c1 in (
        with v1 as (
                select rownum n from dual
                connect by level <= 10000
        )
        select  rownum  id
        from    v1, v1
        where   rownum <= 1e6
   ) loop
   insert into t1 values(c1.id)
   -- commit /* write immediate wait */ ;
   end loop;
end;
/
```

1M rows:	30 seconds	2.5 minutes	10 minutes.	
Redo	235 MB	480 MB	532 MB	

Patterns (a)

```
select
       mod(rownum-1, 10),
       0,1,2,3,4,5,6,7,8,9,0,1,2,3,4,5,6,7,8,9,...

select
       trunc((rownum-1)/3),
       0,0,0,1,1,1,2,2,2,3,3,3,4,4,4,5,5,5 ...

select
       mod(trunc((rownum-1)/3),5)
       0,0,0,1,1,1,2,2,2,3,3,3,4,4,4,0,0,0,1,1,1 ...

select
       trunc(sysdate) - 180 + trunc((rownum-1)/3)/86400,
       -- three rows per second starting 6 months ago
```


Patterns (b)

```
execute dbms_random.seed(0)
   --   dbms_random.seed(binary_integer)
   --   dbms_random.seed(string)


function dbms_random.value
   -- 0.0 <= value < 1.0, 38 digit precision

function dbms_random.value(low, high)
   -- low <= value < high, 38 digit precision

function dbms_random.normal
   -- normal distribution, mean=0, standard deviation = 1

function dbms_random.string (format, length)
   -- upper alpha, lower alpha, mixed alpha
   -- mixed alphanumeric, any printable
```

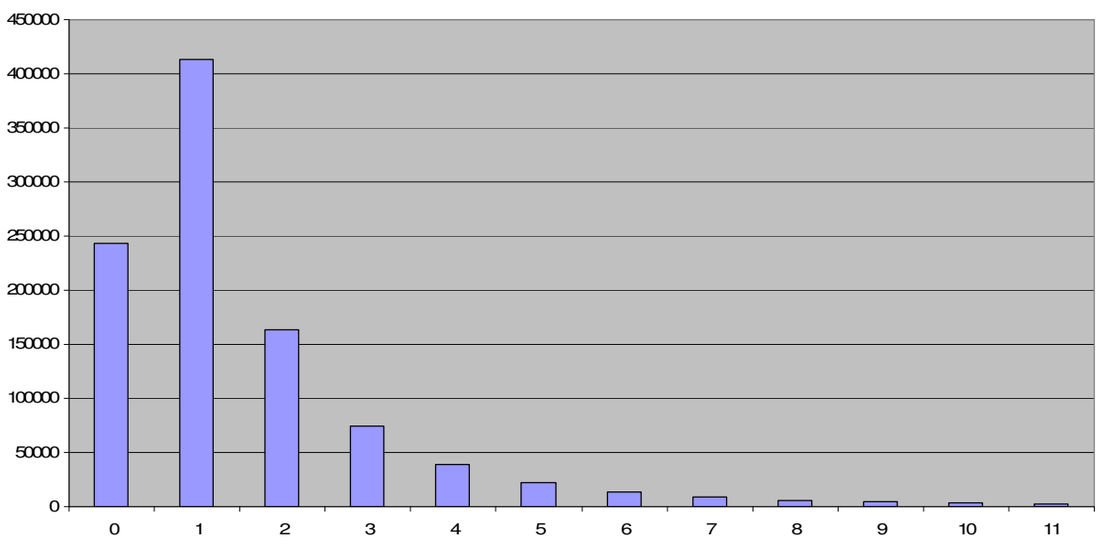
Patterns (c)

```
select
   trunc(dbms_random.value(100,200)), -- uniform random 100 - 199,

   trunc(dbms_random.value(1,100),2), -- cash values $1.00 to $99.99,

   dbms_random.string('U',dbms_random.value(2,7)),
                      -- random upper case string, 2 to 6 characters


select
   trunc(sysdate) - 180 + trunc((rownum-1)/3)/86400  date_created,
   trunc(sysdate) - 180 + trunc((rownum-1)/3)/86400
        + dbms_random.value(1800, 86400)/86400       date_completed,
```

Three rows per second for the last six months, where rows have been "processed" within 24 hours, but none faster than 30 minutes

```
select   round(100 + 15 * dbms_random.normal)          IQ
```


Patterns (d1)



Patterns (d2)

```
select
        case
                when lognorm_round > 11 then 1 else lognorm_round
        end             as lognorm_hacked,
        count(*)
from
        (
        with v1 as (
                select rownum id from dual connect by level <= 10000
        )
        select  round(exp(dbms_random.normal))        lognorm_round
        from    v1,v1
        where   rownum <= 1e6
        )
group by
        case
                when lognorm_round > 11 then 1 else lognorm_round
        end
order by        lognorm_hacked
;
```


Patterns (e1)

```
create table t1
as
with v1 as (
   select rownum id from dual connect by level <= 10000
)
select
   case
        when mod(rownum,100000) = 0 then cast('RARE'        as varchar2(12))
        when mod(rownum,10000)  = 0 then cast('FAIRLY RARE' as varchar2(12))
        when mod(rownum,1000)   = 0 then cast('NOT RARE'    as varchar2(12))
        when mod(rownum,100)    = 0 then cast('COMMON'      as varchar2(12))
                                    else cast('THE REST'    as varchar2(12))
   end  state
from
   v1, v1
where
   rownum <= 10000000
;
```

Patterns (e2)

```
select
        state, count(*)
from    t1
group by
        state
order by
        count(*)
;
```

```
STATE          COUNT(*)
-------------- ----------
RARE                  100
FAIRLY RARE           900
NOT RARE             9000
COMMON              90000
THE REST          9900000
```

Getting the right sort of quantities for a "shortlist" is easy

What about physical location ?

Patterns (f1)

Status values (in order of processing):		Typical Volume
NEW	new row from loader	1,000
PRP	prepared (cleaned)	5,000
FKC	foreign keys checked	3,000
LDD	loading to master table	1,000
COM	completed	9,990,000
ERR	error	100

Historical data is mostly "completed"

Recent data is in the other states

Errors are randomly scattered

Patterns (f2)

```
create table t1
as
with v1 as (
   select rownum id from dual connect by level <= 10000
)
select
   case
        when mod(rownum,100000) = 7             then 'ERR'
        when rownum <= 9990000                  then 'COM'
        when mod(rownum,10)     = 0             then 'NEW'
        when mod(rownum,10)     between 1 and 5 then 'PRP'
        when mod(rownum,10)     between 6 and 8 then 'FKC'
        when mod(rownum,10)     = 9             then 'LDD'
   end  status
from
   v1, v1
where
   rownum <= 1e6
;
```

STA	COUNT(*)
ERR	100
COM	9989900
NEW	1000
PRP	5000
FKC	3000
LDD	1000

Patterns (f3)

```
   case
        when mod(rownum,100000) = 7
               then 'ERR'
        when rownum <= 9990000
               then 'COM'
               else
                      case trunc(dbms_random.value(1,11))
                             when 1 then 'NEW'
                             when 2 then 'LDD'
                             when 3 then 'FKC'
                             when 4 then 'FKC'
                             when 5 then 'FKC'
                                    else 'PRP'
                      end
   end         status
```

STA	COUNT(*)
ERR	100
LDD	1046
NEW	1047
FKC	2994
PRP	4913
COM	9989900

Diagnostics (a)

```
select
        status,
        count(*)                                        row_ct,
        count(distinct substr(rowid,1,15))              blocks
from
        t1
group by
        status
order by
        count(*)
/
```

STA	ROW_CT	BLOCKS
ERR	100	100
LDD	1,046	16
NEW	1,047	16
FKC	2,994	16
PRP	4,913	16
COM	9,989,900	15,183

Diagnostics (b)

```
select
        dbms_rowid.rowid_to_absolute_fno(
                t1.rowid,'TEST_USER','T1'
        ) file_no,
        dbms_rowid.rowid_block_number(
                t1.rowid
        ) block_no,
        count(*)
from    test_user.t1
where   status = 'LDD'
group by
        dbms_rowid.rowid_to_absolute_fno(
                t1.rowid,'TEST_USER','T1'
        ),
        dbms_rowid.rowid_block_number(
                t1.rowid
        )
order by
        1,2
;
```

FILE_NO	BLOCK_NO	COUNT(*)
5	15192	56
5	15193	78
5	15194	66
5	15195	61
5	15196	65
5	15197	85
5	15198	63
5	15199	65
5	15200	69
5	15201	69
5	15202	81
5	15203	58
5	15204	64
5	15205	61
5	15206	69
5	15207	36

Diagnostics (c)

```
with extents as(
    select /*+ materialize */
           ext.extent_id, ext.file_id, ext.block_id, ext.blocks
    from   dba_extents ext
    where  ext.owner = 'TEST_USER'
    and    ext.segment_name = 'T1'
),
rowdata as (
    select/*+ materialize */
           dbms_rowid.rowid_to_absolute_fno(t1.rowid,'TEST_USER','T1') as file_no,
           dbms_rowid.rowid_block_number(t1.rowid)                     as block_no,
           status
    from   test_user.t1
    where  status = 'LDD'
)
select    /*+ leading(ext rowdata) use_merge(rowdata) */
          ext.extent_id, rowdata.file_no, rowdata.block_no, count(*)
from
          extents   ext,
          rowdata
where
          rowdata.file_no = ext.file_id
and       rowdata.block_no between ext.block_id and ext.block_id + blocks -1
group by  ext.extent_id, rowdata.file_no, rowdata.block_no
order by  ext.extent_id, rowdata.file_no, rowdata.block_no
;
```


Index Effects (a)

```
create table as
with (  ...  )
select
        trunc(dbms_random.value(1,1001))
from    v1, v1
where   rownum <= 1e6
```

Note: 1,000 rows per key

Option 1: create table as select, create index
Option 2: create table(), create index, insert as select
Option 2: create table(), create index, insert /*+ append */ as select
Option 4: create table(), create index, pl/sql loop with commit

How much difference can it make ?

Index Effects (b)

	CTAS Create index	Create table Create index Insert	Create table Create index Append	Create table Create index pl/sql loop
Height	3	3	3	3
Leafs	2077	3104	1863	3104
Branches	5	9	5	9
Space	16,656,160	24,904,288	14,944,160	24,904,288
Used space	14,923,306	14,923,306	14,920,033	14,937,054
Pct_used	90	60	100	60

8 seconds / 16 seconds / 16 seconds / 4 minutes 12 seconds
The spare space is a side effect of large number of rows per key

SQL vs. PL/SQL (a)

test.sql

```
-- setup
@start_100.sql
@start_100.sql
@start_100.sql
@start_100.sql
@start_100.sql
@start_100.sql
@start_100.sql
@start_100.sql
@start_100.sql
@start_100.sql
@start_100.sql
--report
```

start_100.sql

```
@start_10.sql
@start_10.sql
@start_10.sql
@start_10.sql
@start_10.sql
@start_10.sql
@start_10.sql
@start_10.sql
@start_10.sql
@start_10.sql
```

start_10.sql

```
@start_1.sql
@start_1.sql
@start_1.sql
@start_1.sql
@start_1.sql
@start_1.sql
@start_1.sql
@start_1.sql
@start_1.sql
@start_1.sql
```

start_1.sql has the critical SQL statement in it

SQL vs. PL/SQL (b)

test.sql

```
execute dbms_random.seed(0)
set feedback off
select to_char(sysdate,'dd hh24:mi:ss') from dual;

@start_1000000

select to_char(sysdate,'dd hh24:mi:ss') from dual;
set feedback on
```

start_1.sql

```
insert into t1 values(
        trunc(dbms_random.value(1,1001))
);
commit;
```

Storage Results:
as for the pl/sql loop,
but 42 minutes to run.

Concurrency (a)

How do you get N processes to start simultaneously?
How can you resynchronise N processes when they get out of step?

```
declare
        n1              number;
begin
        dbms_lock.allocate_unique(
                lockname        => 'Running',
                lockhandle      => :m_handle   -- incoming bind value
        );
        n1 := dbms_lock.request(
                lockhandle              => :m_handle,
                lockmode                => dbms_lock.x_mode,
                timeout                 => dbms_lock.maxwait,
                release_on_commit       => true
        );
end;
/
```

Note - ***v$lock.type*** is 'UL', ***v$lock.id1*** is dbms_lock_allocated.lockid

Concurrency (b)

Starting together:

Controller gets "Running" in exclusive mode.

Runners request same lock in shared mode - and wait.

Controller releases the lock (may simply commit if release_on_commit=> true)

Synchronisation points

Use two locks -

"Running" lock -- release_on_commit => false

"Synch" lock -- release_on_commit => true

At synchronisation point pseudo-code should be:

Try to get synch lock in ***exclusive*** mode ***nowait***
if failed
    get synch lock in ***share*** mode, ***willing to wait***
else
    poll v$lock until "synch lock waiters" = "running lock holders" - 1
end if
release synch lock

Concurrency (c1)

PL/SQL Loop to insert data (see slide 8 for code, slide 21 for results)
1M rows from a single process: Elapsed time 3:01
250,000 rows from 4 processes: Elapsed time 2:14
Approx. 1,000 rows per key:

Index stats:

```
Single run                          Parallel run
LF_BLKS             : 3100          LF_BLKS             : 3107
LF_BLK_LEN          : 8000          LF_BLK_LEN          : 7976
BR_BLKS             : 9             BR_BLKS             : 9
BTREE_SPACE         : 24872288      BTREE_SPACE         : 24853720
USED_SPACE          : 14936778      USED_SPACE          : 14936697
PCT_USED            : 61            PCT_USED            : 61
```

Concurrency (c2)

```
create sequence s1 cache 10000;
for r in 1..1000000 loop
        insert into t1 values(s1.nextval);
        commit;
end loop;
```

Index stats: (10.2.0.3)

Single run		**Parallel run**	
LF_BLKS	: ***1999***	LF_BLKS	: ***5423***
LF_BLK_LEN	: 8000	LF_BLK_LEN	: 4688
BR_BLKS	: 4	BR_BLKS	: 16
BTREE_SPACE	: 16024128	BTREE_SPACE	: 25551536
USED_SPACE	: 16003760	USED_SPACE	: 16044538
PCT_USED	: 100	PCT_USED	: 63

Concurrency (c3)

```
create sequence s1 cache 10000;
for r in 1..1000000 loop
        insert into t1 values(s1.nextval);
        commit;
end loop;
```

Index stats: (11.2.0.3)

Single run		**Parallel run**	
LF_BLKS	: **1999**	LF_BLKS	: **6943**
LF_BLK_LEN	: 7996	LF_BLK_LEN	: 3988
BR_BLKS	: 4	BR_BLKS	: 20
BTREE_SPACE	: 16016116	BTREE_SPACE	: 27849244
USED_SPACE	: 16003760	USED_SPACE	: 16062686
PCT_USED	: 100	PCT_USED	: 58

Concurrency (c4)

```
Block header dump:  0x0100258a
 Object id on Block? Y
 seg/obj: 0x13e6c  csc: 0x00.2300db0  itc: 169  flg: E  typ: 2 - INDEX
     brn: 0  bdba: 0x1002580 ver: 0x01 opc: 0
     inc: 0  exflg: 0

Itl           Xid                  Uba         Flag  Lck        Scn/Fsc
0x01   0x0003.001.00002fda  0x00c1ee7c.0248.01  CB--    0  scn 0x0000.02300daf
0x02   0x0000.000.00000000  0x00000000.0000.00  ----    0  fsc 0x0000.00000000
0x03   0x0000.000.00000000  0x00000000.0000.00  ----    0  fsc 0x0000.00000000
...
0xa6   0x0000.000.00000000  0x00000000.0000.00  ----    0  fsc 0x0000.00000000
0xa7   0x0000.000.00000000  0x00000000.0000.00  ----    0  fsc 0x0000.00000000
0xa8   0x0000.000.00000000  0x00000000.0000.00  ----    0  fsc 0x0000.00000000
0xa9   0x0000.000.00000000  0x00000000.0000.00  ----    0  fsc 0x0000.00000000
Leaf block dump
```


Concurrency (d)

```
for r in 1..1000000 loop
        dbms_lock.sleep(dbms_random.value);
        insert into t1 values(s1.nextval);
        commit;
end loop;
```

http://radino.eu/2008/12/25/how-to-implement-sleeping/ (Radoslav Golian)

```
create or replace procedure sleep(x_millis in number) as
language java
name 'java.lang.Thread.sleep(int)';

execute sleep(dbms_random.value(1,1001))
```

Parent/Child (a1)

Simple strategy: create child first, aggregate to parent

```
create table order_lines
as
with v1 as (
        select rownum id from dual connect by level <= 10000
)
Select
        trunc((rownum-1)/5)               id_order,
        1+ mod(rownum - 1,5)              line_id,          -- 1 to 5
        trunc(dbms_random.value(10001,20001))      id_product,
        trunc(dbms_random.value(50001,55001))      id_customer,
        lpad(rownum,8)                    v1,
        rpad('x',100,'x')                 padding
from
        v1, v1
where
        rownum <= 5000000         -- 5M order lines, 1M orders
;       -- 74 seconds

alter table order_lines
add constraint orl_pk primary key(id_order, line_id);       -- 34 seconds
```


Parent/Child (a2)

```
create table orders as
select
        ol.id_order                                 id,
        sysdate - ((1000000 - rownum)/864)          date_ordered,
        sysdate - ((1000000 - rownum)/864) +
                dbms_random.value(3,7)              date_delivered,
        lpad(rownum,10)                             v1,
        rpad('x',100,'x')                           padding
from
        (       select   /*+ no_use_hash_aggregation */
                         distinct id_order
                from order_lines
        ) ol
;       -- 13.64 seconds (1M rows)

update orders
set     date_delivered = null
where   date_delivered > sysdate
;       -- 2.00 seconds (4,300 rows)

alter table orders
add constraint ord_pk primary key(id)
;       -- 5.31 seconds
```

Parent/Child (b)

Assume we have already created the orders table.
We will allow an order to have several order lines (between 1 and 10).
For realism, order lines for an order have to be physically adjacent

```
create table order_lines as
with gen as (select rownum rn  from dual connect by level <= 10)
select
        /*+ leading(ord gen) use_nl(gen) */
        ord.id   order_id,
        gen.rn   line_id,
        {other bits}
from    (
        select   orders.id,
                 trunc(dbms_random.value(1, 11)) line_count
        from     orders
        )        ord,
        gen
where
        gen.rn <= ord.line_count
/* order by ord.id, gen.rn */     -- technically required, not actually needed
;
```


Reversing the join order and a hash join (building with ***gen***) may be faster:
/+ leading(gen ord) use_hash(ord) no_swap_join_inputs(ord) */*

Summary

- Generating the volume - quick and easy
- Loading the data - how
- Patterns in data - easy
- Diagnostic tools - how much and where
- Indexing issues - loading effects
- Concurrency issues - loading effects
- Referential Integrity - which comes first