April 23-27, 2017
Raleigh, NC, USA

C12
Informix Query Performance Tuning Basics

Mike Walker

Advanced DataTools

mike@advancedatatools.com

Performance Tuning Basics

- Identifying long running queries
- Explain plans – Query Tuning
- Optimizer directives
- Monitoring the buffer pool usage
- Finding busy tables
- Checking statistics

Long Running Queries – What's Running?

- How to tell if a session is doing anything
- Start with "onstat -u"
- First position in the Flags column indicates what's going on

  B - Waiting for a buffer

  C - Waiting for a checkpoint

  G - Waiting for a write of the logical-log buffer

  L - Waiting for a lock

  S - Waiting for mutex

  T - Waiting for a transaction

  Y - Waiting for condition

  X - Waiting for a transaction cleanup (rollback)

Interested in anything that is NOT a "Y" – and last position is a "-"

Long Running Queries – What's Running?

onstat -u

```
Userthreads
address          flags    sessid   user     tty      wait             tout locks nreads   nwrites
70000174751a028  ---P--D 1        informix -        0                0    0     47702    3024353
70000174751a850  ---P--F 0        informix -        0                0    0     0        5744623
70000174751b078  ---P--F 0        informix -        0                0    0     0        2744394
.
.
70000174a2615c8  Y--P--- 240522   admin    DX-ALCV  7000018cc1dc7c0  0    3     149935   0
70000174a261df0  ---PR-- 1555094  webuser  -        0                0    3     4989784  512
70000174a262e40  Y--P--- 1565579  webuser  -        700001855e069b8  0    2     0        0
700001753360ca8  Y--P--- 1567350  webuser  -        70000185997dd18  0    0     3905     0
7000017533614d0  --BPX-- 1567353  webuser  -        0                0    5     38       128
7000017533645c0  Y--P--- 1562970  webuser  -        700001856de09b8  0    3     13332    192
700001753365e38  Y--P--- 869782   bob      PROD-SRV 700001762fb5628  0    1     0        0
700001753366660  ---PR-- 1543869  webuser  -        0                0    3     11186388 834
700001753366e88  Y--P--- 1496985  webuser  -        700001787828898  0    3     39872    0
.
.
```

Long Running Queries – How Long?

- Use “onstat -g ntt” to find out when the connection was established and when the last SQL was submitted

```
          netscb thread name     sid      open      read     write
700000703247b10 sqlexec       951571 12:14:08 13:51:19 13:51:19
700000733af0960 sqlexec       951553 12:03:16 13:51:16 13:51:16
700000702ec3458 sqlexec       951546 12:03:16 13:51:16 13:51:16
7000007030e7cd0 sqlexec       951521 11:59:18 13:52:05 13:52:05
70000070284e600 sqlexec       951510 11:57:23 13:52:05 13:52:05
```

Session ID

Connection Established

Time of last SQL

Long Running Queries

- Use onstat –u to tell you what is running
- Use onstat –g ntt to tell you the last time SQL was submitted
- Use the results of both to see what's running *and* for how long
  - How long = difference between last SQL time and current time

Long Running Queries

Use a simple script to combine info together, and show locks/reads/writes, flags and current time, which repeats at regular intervals:

```
63867891    11:24:30 14:19:21 14:19:21 [14:19:24] <       3 142216977  423644 -        ---PR--
63957715    17:54:30 14:19:21 14:19:21 [14:19:24] <       3 29900439      256 -        ---PR--
64157342    04:49:23 14:19:22 14:19:22 [14:19:24] <       3 36750125       64 -        ---PR--
55507969    21:03:13 14:19:23 14:19:23 [14:19:24] <       3     2180    16192 -        ---PR--
....
63867891    11:24:30 14:19:21 14:19:21 [14:19:25] <       3 142302481  423644 -        ---PR--
63957715    17:54:30 14:19:21 14:19:21 [14:19:25] <       3 29912222      256 -        ---P---
64157342    04:49:23 14:19:22 14:19:22 [14:19:25] <       3 36842541       64 -        ---PR--
64158966    09:54:25 14:19:25 14:19:25 [14:19:25] <       3 28081252       64 -        ---P---
....
63867891    11:24:30 14:19:21 14:19:21 [14:19:27] <       3 142387729  423644 -        ---PR--
64157342    04:49:23 14:19:22 14:19:22 [14:19:27] <       3 36928045       64 -        ---PR--
....
63867891    11:24:30 14:19:21 14:19:21 [14:19:28] <       3 142472977  423644 -        ---PR--
64157342    04:49:23 14:19:22 14:19:22 [14:19:28] <       3 37013549       64 -        ---PR--
55507969    21:03:13 14:19:26 14:19:26 [14:19:28] <       3     2180    16192 -        ---P---
```

Long Running Queries

For OLTP, would like to see the SQL time change every second or so

SQL time is changing for most sessions approximately every second

```
59022843    16:37:54 16:37:54 16:37:54 [16:57:33] <       1  1093756        0 -        ---PR--
59024816    16:47:42 16:47:45 16:47:45 [16:57:33] <       2 50401217        0 -        ---PR--
58980487    13:54:36 16:57:32 16:57:32 [16:57:33] <       4  5801210   101632 -        ---PR--
58980607    13:54:49 16:57:32 16:57:32 [16:57:33] <       4  6394606   122688 -        ---PR--
59026950    16:57:33 16:57:33 16:57:33 [16:57:33] <       2      798       64 -        ---PR--
59021449    16:31:16 16:57:33 16:57:33 [16:57:33] <       2    34261      256 LT-USER1 ---P---
58891988    08:49:49 16:57:33 16:57:33 [16:57:33] <       3 409885058   10460 -        ---PR--
59026776    16:56:38 16:57:33 16:57:33 [16:57:33] <       4     5200     7360 -        ---P---
....
59022843    16:37:54 16:37:54 16:37:54 [16:57:34] <       1  1094017        0 -        ---PR--
59024816    16:47:42 16:47:45 16:47:45 [16:57:34] <       2 50425264        0 -        ---PR--
55507969    21:03:13 16:57:34 16:57:34 [16:57:34] <       3     1338     8320 -        ---PR--
58980487    13:54:36 16:57:34 16:57:34 [16:57:34] <       4  5801892   101632 -        ---PR--
58980607    13:54:49 16:57:34 16:57:34 [16:57:34] <       4  6394606   122688 -        ---PR--
59026776    16:56:38 16:57:35 16:57:35 [16:57:34] <       4     5502     7488 -        ---P---
....
59022843    16:37:54 16:37:54 16:37:54 [16:57:36] <       1  1094246        0 -        ---PR--
59024816    16:47:42 16:47:45 16:47:45 [16:57:36] <       2 50447952        0 -        ---PR--
55507969    21:03:13 16:57:35 16:57:35 [16:57:36] <       3     1338     8320 -        ---P---
59026776    16:56:38 16:57:36 16:57:36 [16:57:36] <       4     5548     7552 -        ---P---
58980487    13:54:36 16:57:36 16:57:36 [16:57:36] <       4  5802380   101632 -        ---P---
58980607    13:54:49 16:57:36 16:57:36 [16:57:36] <       4  6395033   122688 -        ---PR--
```

Sessions 59022843 and 59024816 have been running for 20 minutes and 10 minutes respectively

Long Running Queries

- For a session where the SQL is changing regularly, view the SQL at regular intervals:

  `onstat -g sql <SID> -r 1` ← Repeat every 1 second

- If the SQL is changing, then will soon get a good idea of what is going on:
  - Lots of different statements?
  - Repeated statements with different literal values?
- If different statements are executed, but see one SQL repeated frequently, then this may be the first statement to begin investigating

Long Running Queries

- Use the same approach for a background process that periodically checks how long all SQLs have been running and dump session information to a file when exceeds a threshold

---

- Also use **Informix SQL Trace** to record queries and then retrieve those that ran the longest
- SQL Trace is covered extensively elsewhere

Reviewing Query Plans

- Now we have found some slow SQL, find out what it's doing by obtaining a Query Plan

- Turn on Dynamic Explain to get plan for a session:

```
onmode -Y <sid> <0|1|2> [filename]
   0=off
   1=plan + statistics on
   2=only plan on
```

Dynamic Query Plans

```
onmode -Y 10563 1
```

Set Dynamic Explain for Session 10563

```
onstat -g ses

IBM Informix Dynamic Server Version 12.10.FC5AEE -- On-Line -- Up 1 days 12:01:36 --
2947104 Kbytes

session                                          #RSAM    total      used       dynamic
id       user     tty      pid      hostname    threads  memory     memory     explain
10657    informix -        0        -           0        16384      12480      off
10653    informix -        0        -           0        16384      12480      off
10563    informix 2        4243     apollo      1        73728      64480      on
10028    informix -        0        apollo      1        335872     321728     off
10011    informix -        0        apollo      1        241664     100072     off
44       informix -        0        -           1        626688     472280     off
43       informix -        0        -           1        626688     471576     off
42       informix -        0        -           1        618496     494080     off
41       informix -        0        -           1        102400     86784      off
```

Dynamic Query Plans

```
onstat -g sql

IBM Informix Dynamic Server Version 12.10.FC5AEE -- On-Line -- Up 1 days 12:03:11 --
2947104 Kbytes


Sess       SQL            Current            Iso Lock       SQL   ISAM F.E.
Id         Stmt type      Database           Lvl Mode       ERR   ERR  Vers  Explain
10563      -              adtc_monitor       CR  Not Wait    0     0    9.24  Dyn. stat
10028      -              adtc_monitor       CR  Not Wait    -255  0    9.28  Off
10011      -              adtc_monitor       CR  Not Wait    -255  0    9.28  Off
44                        sysadmin           DR  Wait 5      0     0    -     Off
43                        sysadmin           DR  Wait 5      0     0    -     Off
42                        sysadmin           DR  Wait 5      0     0    -     Off
41                        sysadmin           CR  Not Wait    0     0    -     Off
```

Dynamic Query Plans

Explain plan written to a file with the SID in the name:

```
-rw-rw-rw- 1 informix informix     573 Apr  7 11:17 sqexplain.out.10563

cat sqexplain.out.10563

QUERY: (OPTIMIZATION TIMESTAMP: 04-07-2017 11:17:33)
------
select * from snapshot

Estimated Cost: 79971
Estimated # of Rows Returned: 1199409

  1) informix.snapshot: SEQUENTIAL SCAN
```

Query & Query Plan

```
Query statistics:
-----------------

  Table map :
  ----------------------------
  Internal name     Table name
  ----------------------------
  t1                snapshot

  type     table  rows_prod  est_rows  rows_scan  time       est_cost
  ---------------------------------------------------------------------
  scan     t1     1203641    1199409   1203641    00:00.00   79971
```

Query Statistics shown because used: onmode –Y *<sid>* 1

Query Statistics

Dynamic Query Plans

- Using “onmode -Y” will not produce anything until the *next* statement runs – so no good for getting the explain plan for a single, long running statement
- Limited value if prepared SQL is being executed

- Capture the SQL to a file instead, and get the explain plan for that…

SET EXPLAIN

- Use the “SET EXPLAIN” SQL statement to start/stop the output of explain plans:
  - **SET EXPLAIN ON**: write explain plans to a file for SQL statements that follow
  - **SET EXPLAIN OFF**: turn off explain plans
  - **SET EXPLAIN ON AVOID_EXECUTE**: Produce explain plan *without* running the SQL
  - **SET EXPLAIN FILE TO *<filename>***: Write explain file to specified file

SET EXPLAIN

- SET EXPLAIN ON / SET EXPLAIN OFF:

```
SET EXPLAIN ON;
SELECT * FROM x WHERE y = 10;
SET EXPLAIN OFF;
```

- By default, the query plan is written to the file: sqexplain.out
- File is created in the current directory (UNIX)
- If use client app, the file will be in home directory of the user that SQL was executed as
- File will be appended to each time more SQL is executed

SET EXPLAIN

slow1.sql:

```
set explain file to "slow1.exp";
set explain on;

output to /dev/null
select c.customer_num, o.order_num
from customer c, orders o
where c.customer_num = o.customer_num
  and c.company = "Play Ball!"
order by 2;
```

```
timex dbaccess -e stores_demo slow1.sql > slow1.out 2>&1 &

-rw-rw-rw- 1 informix informix 2167 Apr  9 07:50 slow1.exp
```

SET EXPLAIN

slow1.exp

```
QUERY: (OPTIMIZATION TIMESTAMP: 04-09-2017 07:50:47)
------
select c.customer_num, o.order_num
from customer c, orders o
where c.customer_num = o.customer_num
  and c.company = "Play Ball!"
order by 2

Estimated Cost: 6
Estimated # of Rows Returned: 2
Temporary Files Required For: Order By

  1) informix.c: SEQUENTIAL SCAN

        Filters: informix.c.company = 'Play Ball!'

  2) informix.o: INDEX PATH

    (1) Index Name: informix. 102_4
        Index Keys: customer_num   (Serial, fragments: ALL)
        Lower Index Filter: informix.c.customer_num = informix.o.customer_num
NESTED LOOP JOIN
```

SET EXPLAIN

```
Query statistics:
-----------------

  Table map :
  ----------------------------
  Internal name     Table name
  ----------------------------
  t1                c
  t2                o

  type     table  rows_prod  est_rows  rows_scan  time       est_cost
  ------------------------------------------------------------------
  scan     t1     1          3         28         00:00.00   4

  type     table  rows_prod  est_rows  rows_scan  time       est_cost
  ------------------------------------------------------------------
  scan     t2     4          23        4          00:00.00   0

  type     rows_prod  est_rows  time       est_cost
  ---------------------------------------------------
  nljoin   4          3         00:00.00   6

  type     rows_sort  est_rows  rows_cons  time       est_cost
  ---------------------------------------------------------------
  sort     4          3         4          00:00.00   0
```

As long as query is allowed to complete, Query Statistics will be shown at the end of the plan [EXPLAIN_STAT=1 in ONCONFIG]

SET EXPLAIN

For long running SQL or for Insert, Update or Delete operations, use “AVOID_EXECUTE” to get the explain plan *without* running the SQL:

slow2.sql:

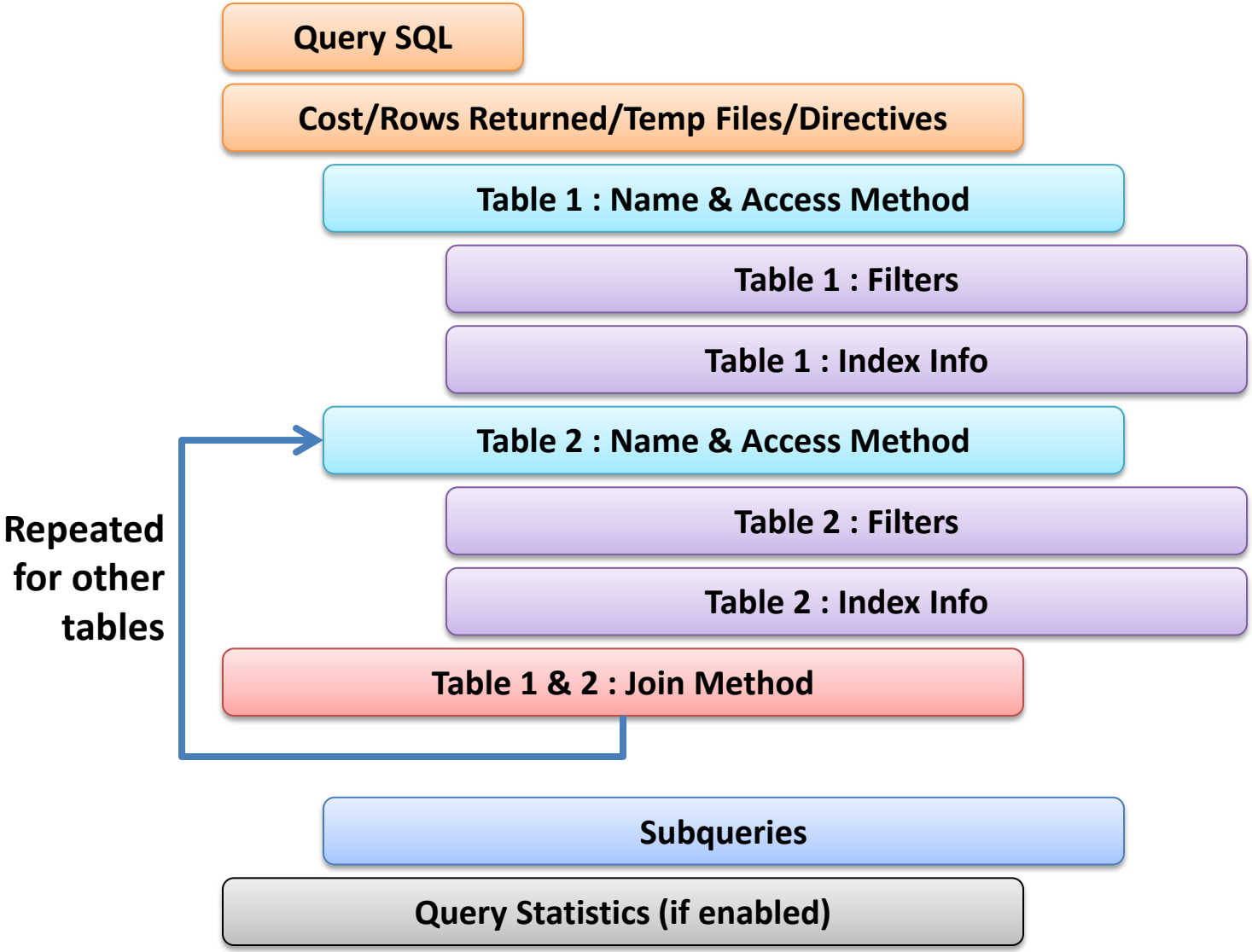
```
set explain file to "slow2.exp";
set explain on avoid_execute;

update orders
set ship_instruct = null
where customer_num = 104;
```

SET EXPLAIN

```
dbaccess -e stores_demo slow2.sql

Database selected.

set explain file to "slow2.exp";
Explain set.

set explain on avoid_execute;
Explain set.

update orders
set ship_instruct = null
where customer_num = 104;
0 row(s) updated.

Warning! avoid_execute has been set

Database closed.
```

If use AVOID_EXECUTE will NOT see the Query Statistics in the Explain Plan

Anatomy of a Query Plan



Query Plans

```
QUERY: (OPTIMIZATION TIMESTAMP: 04-09-2017 07:50:47)
------
select c.customer_num, o.order_num
from customer c, orders o
where c.customer_num = o.customer_num
  and c.company = "Play Ball!"
order by 2

Estimated Cost: 6
Estimated # of Rows Returned: 2
Temporary Files Required For: Order By

  1) informix.c: SEQUENTIAL SCAN

        Filters: informix.c.company = 'Play Ball!'

  2) informix.o: INDEX PATH

    (1) Index Name: informix. 102_4
        Index Keys: customer_num   (Serial, fragments: ALL)
        Lower Index Filter: informix.c.customer_num =
informix.o.customer_num
NESTED LOOP JOIN
```

Query SQL

Cost/Rows Returned/Temp Files/Directives

Table 1 : Name & Access Method

Table 1 : Filters

Table 2 : Name & Access Method

Table 2 : Index Info

Table 1 & 2 : Join Method

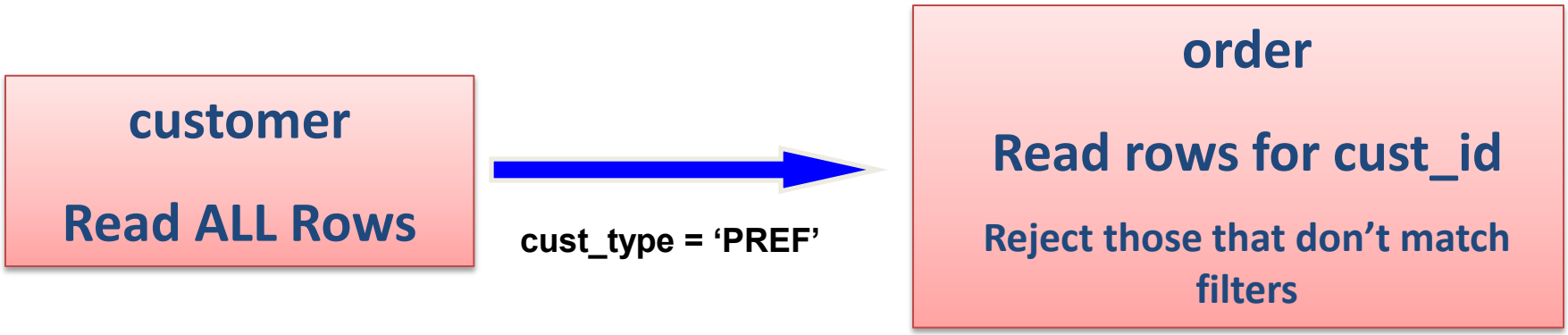
Query Plans

```
select c.cust_id, c.cust_name, o.order_id
from customer c, order o
where c.cust_id = o.cust_id
and c.cust_type = 'PREF'
and o.pay_type != 'CREDCARD'
and o.ship_method = 'FEDEX'


Estimated Cost: 51207
Estimated # of Rows Returned: 9000

  1) informix.c: SEQUENTIAL SCAN

        Filters: informix.c.cust_type = 'PREF'

  2) informix.o: INDEX PATH

        Filters: (informix.o.ship_method = 'FEDEX' AND
informix.o.pay_type != 'CREDCARD' )

    (1) Index Name: informix.order_ix2
        Index Keys: cust_id   (Serial, fragments: ALL)
        Lower Index Filter: informix.c.cust_id = informix.o.cust_id
NESTED LOOP JOIN
```

Estimate – if very wrong, then stats may be off

Query Plans – Breaking it Down

```
 1) informix.c: SEQUENTIAL SCAN

        Filters: informix.c.cust_type = 'PREF'

  2) informix.o: INDEX PATH

        Filters: (informix.o.ship_method = 'FEDEX' AND
informix.o.pay_type != 'CREDCARD' )

    (1) Index Name: informix.order_ix2
        Index Keys: cust_id   (Serial, fragments: ALL)
        Lower Index Filter: informix.c.cust_id = informix.o.cust_id
NESTED LOOP JOIN
```



Query Plans – Breaking it Down

customer

Read ALL Rows

cust_type = 'PREF'

order

Read rows for cust_id

Reject those that don't match filters

```
select count(*)
from customer;
```

```
(count(*))
    100000
```

```
select count(*)
from customer
where cust_type = 'PREF';
```

```
(count(*))
     25000
```

```
select count(*)
from customer c, order o
where c.cust_id = o.cust_id
and c.cust_type = 'PREF';
```

```
(count(*))
    250000
```

Results of query…

```
6040 row(s) retrieved.
```

Query Plans

customer (all rows)	: 100000 rows
customer.cust_type = 'PREF'	: 25000 rows
Joins to order table	: 250000 rows
Rows returned from query	: 6040 rows

Number of Rows Read

vs

Number of Rows Returned

Lots of rows read…and then discarded!

Try and make the number of rows read as close as possible to those that are needed

Query Statistics

Review the "Filters"
Including the filter columns in the index will reduce the rows scanned and the rows discarded

```
Query statistics:
-----------------

  Table map :
  ----------------------------
  Internal name      Tab
  ----------------------------
  t1                 c
  t2                 o

  type      table    rows_prod   est_rows   rows_scan   time        est_cost
  ------------------------------------------------------------------------
  scan      t1       25000       10000      100000      00:00.41    15501

  type      table    rows_prod   est_rows   rows_scan   time        est_cost
  ------------------------------------------------------------------------
  scan      t2       6040        90000      250000      00:14.76    4

  type      rows_prod   est_rows   time        est_cost
  ---------------------------------------------------
  nljoin    6040        9000       00:15.24    51207
```

Rows Needed (after filtering)

Rows Read

Table Scan but fast!

This is where the time was spent

Query Statistics

New index includes ship_method
create index order_ix3 on order(cust_id, **ship_method**)

```
Query statistics:
-----------------

  Table map :
  ----------------------------
  Internal name      Table name
  ----------------------------
  t1                 c
  t2                 o

  type      table   rows_prod   est_rows   rows_scan   time        est_cost
  ------------------------------------------------------------------------
  scan      t1      25000       10000      100000      00:00.40    15501

  type      table   rows_prod   est_rows   rows_scan   time        est_cost
  ------------------------------------------------------------------------
  scan      t2      6040        128571     6100        00:01.15    1

  type      rows_prod   est_rows   time        est_cost
  ---------------------------------------------------
  nljoin    6040        12858      00:01.61    25312
```

Rows Read is much closer to the Rows Needed – fewer rows discarded

Faster!

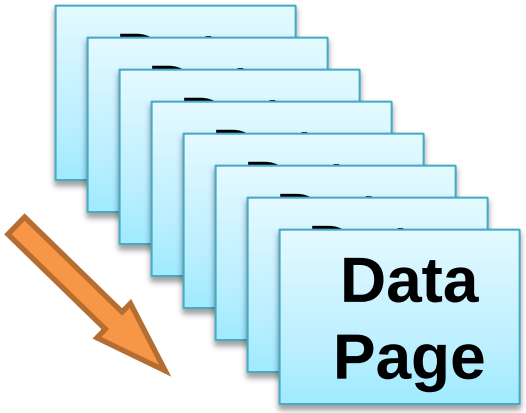
Sequential Scans

- If a Query Plan contains a Sequential Scan, all rows of the table are read (before any filter is applied)
- Don't freak out!
  - If most of the rows read from the table are needed, then it may be okay
  - Consider that many indexed reads of data can be costly because of the read of the index, *plus* the read of the data page

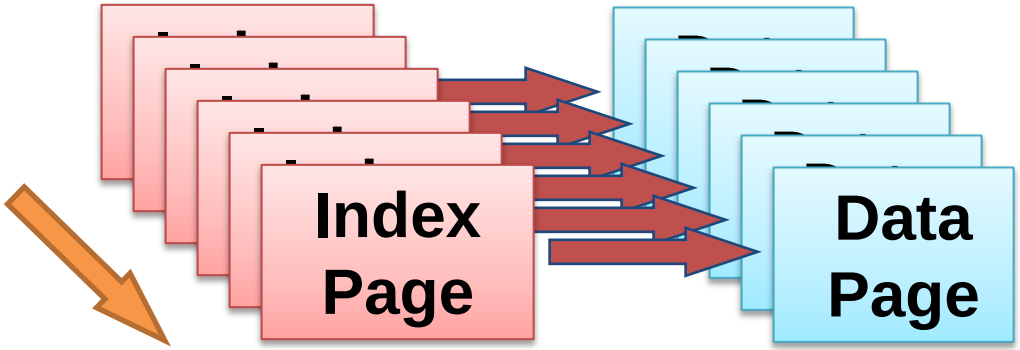
Sequential Scans

A Scan of all Data Pages *may* be faster than lots of Indexed Reads

Sequential Scan



Indexed Reads



But it depends on how many rows are actually needed

A scan of a large table can trash the cache

Sequential Scans

```
1) informix.order: SEQUENTIAL SCAN

        Filters: (informix.order.ship_method = 'FEDEX' AND
  informix.order.pay_type = 'CREDCARD' )

type      table   rows_prod   est_rows   rows_scan   time        est_cost
-----------------------------------------------------------------------
scan      t1      1000000     71429      1000000     00:04.55    280001
```

Rows Read

```
1) informix.order: INDEX PATH

        Filters: informix.order.ship_method = 'FEDEX'

    (1) Index Name: informix.order_ix4
        Index Keys: pay_type order_id   (Serial, fragments: ALL)
        Lower Index Filter: informix.order.pay_type = 'CREDCARD'

type      table   rows_prod   est_rows   rows_scan   time        est_cost
-----------------------------------------------------------------------
scan      t1      12200       71429      500000      00:06.16    2304336
```

Fewer Rows

Slower!

Query Tuning Example

```
SELECT inventory.week_nr, <snip>, inventory.quantity
FROM product, inventory
WHERE inventory.prod_num = product.prod_num
  AND inventory.year_num = 2016
  AND inventory.quantity > 0
  AND TRIM(product.department || '-' || product.prod_type) IN ('A-1')
  AND inventory.store IN (201)

Estimated Cost: 6828412
Estimated # of Rows Returned: 2349350

  1) informix.product: SEQUENTIAL SCAN

        Filters: TRIM ( BOTH ' ' FROM ((informix.product.department || '-' )||
informix.product.prod_type )) = 'A-1'

  2) informix.inventory: INDEX PATH

        Filters: informix.inventory.quantity > 0

    (1) Index Name: informix.inventory_idx3
        Index Keys: prod_num store week_nr year_num   (Key-First)  (Serial,
fragments: ALL)
        Lower Index Filter: (informix.inventory.prod_num =
informix.product.prod_num AND informix.inventory.store = prod_type )
        Index Key Filters:  (informix.inventory.year_num = 2016 )
NESTED LOOP JOIN
```

Query Tuning Example

```
Query statistics:
-----------------

  Table map :
  ----------------------------
  Internal name      Table name
  ----------------------------
  t1                 product
  t2                 inventory

  type      table   rows_prod   est_rows   rows_scan   time        est_cost
  ------------------------------------------------------------------------
  scan      t1      737         67311      673240      00:01.61    49460

  type      table   rows_prod   est_rows   rows_scan   time        est_cost
  ------------------------------------------------------------------------
  scan      t2      242321      23493496   977563      02:26.49    101

  type      rows_prod   est_rows   time        est_cost
  ---------------------------------------------------
  nljoin    242321      2349350    02:28.19    6828412
```

Despite the ugly scan of the product table, it doesn't take long

Time is spent reading inventory

75% of the rows read are discarded

Query Tuning Example

```
 2) informix.inventory: INDEX PATH

        Filters: informix.inventory.quantity > 0

    (1) Index Name: informix.inventory_idx3
        Index Keys: prod_num store week_nr year_num    (Key-First)
(Serial, fragments: ALL)
        Lower Index Filter: (informix.inventory.prod_num =
informix.product.prod_num AND informix.inventory.store = 201 )
        Index Key Filters:  (informix.inventory.year_num = 2016 )
NESTED LOOP JOIN
```

- prod_num is supplied from table 1 (*good*)
- store has a literal value (*good*)
- year_num has a literal value, BUT it's position in the index is after another column (week_nr), so it is used as a filter, but not for drilling into the index (Key-First). Index pages will be read and discarded (*bad*)
- quantity is not in the index. A jump to the data page is needed to read the value to apply the filter. Rows will be read and discarded (*bad*)

Query Tuning Example

- New Index:

  inventory(prod_num, store, year_num, quantity, week_nr)

- ***year_num*** can now be used for drilling down in the index
- ***quantity*** is now in the index. Query uses a “>” operator, but can be used for scanning the index leaf nodes, PLUS there is no need to check the data page
- ***week_nr*** is not needed to filter records, but is used in the select clause. Now no need to go to the data page at all!

Query Tuning Example – Key-Only

```
.
.
2) informix.inventory: INDEX PATH

    (1) Index Name: informix.inventory_idx5
        Index Keys: prod_num store year_num quantity week_nr
(Serial, fragments: ALL)
        Lower Index Filter: (((informix.inventory.prod_num =
informix.product.prod_num AND informix.inventory.store = 201 ) AND
informix.inventory.year_num = 2016 ) AND informix.inventory.quantity > 0 )
.
.
```

No filter anymore

(Key-Only)

Key-Only

With a Key-only read, all the columns needed to satisfy the query are in the index. There is no need to read the data page

Index must also include columns used by the select clause and order by to get a key-only

Key-only reads are very fast!

Query Tuning Example – Key-Only

```
Query statistics:
-----------------

  Table map :
  ----------------------------
  Internal name      Table name
  ----------------------------
  t1                 product
  t2                 inventory

  type      table   rows_prod   est_rows   rows_scan   time        est_cost
  ------------------------------------------------------------------------
  scan      t1      737         67311      673240      00:01.66    49460

  type      table   rows_prod   est_rows   rows_scan   time        est_cost
  ------------------------------------------------------------------------
  scan      t2      242321      23496424   242321      00:00.82    2

  type      rows_prod   est_rows   time        est_cost
  ---------------------------------------------------
  nljoin    242321      2349643    00:02.54    199938
```

With the new index, the query execution dropped from 2.5 minutes to 2.5 seconds

Previously 2m 26s

All rows read are rows that are needed

Correlated Sub-Queries

Correlated

```
select c.*
from customer c
where exists (
  select "X"
  from order o
  where o.custid = c.custid
    and o.stat = "OPEN" )
```

Outer query referenced in Inner query…
Inner query must be repeated for each row returned by the outer query

Non-Correlated

```
select unique c.*
from customer c,
      order o
where c.custid = o.custid
  and o.stat = "OPEN"

select c.*
from customer c
where custid in (
  select custid
  from order o
  where o.stat = "OPEN" )
```

Correlated Sub-Queries

```
... AND NOT EXISTS
(SELECT C.cust_key FROM customer C
 WHERE C.flag_1 = 0 AND C.flag_2 = 0
   AND C.cust_key = ST.cust_key)
```

Join between C and ST table makes this a Correlated Subquery

```
 1) informix.st: INDEX PATH

        Filters: (informix.st.cust_status IN ('F' , 'Q' , 'H')
AND NOT EXISTS <subquery> )

    (1) Index Name: informix.site_x02
        Index Keys: ssn site   (Key-First)  (Serial, fragments: ALL)
        Lower Index Filter: informix.st.ssn > 'A'
        Index Key Filters:  (informix.st.site = 210 )

<snip>

    Subquery:
    ---------
    Estimated Cost: 9236
    Estimated # of Rows Returned: 33

      1) informix.sm: SEQUENTIAL SCAN

            Filters: ((informix.c.cust_key = informix.st.cust_key AND
informix.c.flag_2 = 0 ) AND informix.c.flag_1 = 0 )
```

Subquery executed for every row meeting criteria

Scan of table executed repeatedly

Correlated Sub-Queries

```
  type      table   rows_prod   est_rows   rows_scan   time        est_cost
  ---------------------------------------------------------------------------
  scan      t1      1           85         3265281     04:05.61    816703

.
.
.

Subquery statistics:
--------------------

  Table map :
  ----------------------------
  Internal name      Table name
  ----------------------------
  t1                 c

  type      table   rows_prod   est_rows   rows_scan   time        est_cost
  ---------------------------------------------------------------------------
  scan      t1      8186        33         270015447   04:04.08    9236
```

Cumulative values

Correlated Sub-Queries

```
... AND NOT EXISTS
(SELECT C.cust_key FROM customer C
 WHERE C.flag_1 = 0 AND C.flag_2 = 0
   AND C.cust_key = ST.cust_key)
```

Change EXISTS to a NOT IN and remove the join

```
AND ST.cust_key NOT IN
(SELECT C.cust_key FROM customer C
 WHERE C.flag_1 = 0 AND C.flag_2 = 0)
```

No longer a Correlated Subquery

Subquery will execute one time only

Correlated Sub-Queries

```
  type      table   rows_prod   est_rows   rows_scan   time        est_cost
  ---------------------------------------------------------------------------
  scan      t1      2           171        3265356     00:01.79    36899

.
.
.

Subquery statistics:
--------------------

  Table map :
  ----------------------------
  Internal name      Table name
  ----------------------------
  t1                 c

  type      table   rows_prod   est_rows   rows_scan   time        est_cost
  ---------------------------------------------------------------------------
  scan      t1      8193        330        32981       00:00.04    9236
```

Fraction of records read & much faster!

With the change, the query execution dropped from 4m 5s to less than 2 seconds

Correlated Sub-Queries

- Sometimes a CSQ can be a good thing
- Adding a join to a subquery can reduce the data set returned
- Efficiencies made to a subquery will be compounded when executed repeatedly

Optimizer Directives

- Change the generated query plan by removing paths from consideration
  - Cannot be ignored (as long as they are valid)
  - Negative directives (*Don't* do something)
- Great tool for tuning queries

Optimizer Directives: Syntax

Include the directive as a comment in the SQL, followed by a "+":

SELECT **--+ *directive text***

SELECT **{+ *directive text* }**

SELECT **/*+ directive text*/**

Optimizer Directives – 4GL

Can be used in 4GL, but must PREPARE the SQL with the Directives so that it is submitted to the database

```
let sql_string =
  'select {+ USE_HASH(o) } c.state, max(order_date) ',
  'from order o, cust c ',
  'where o.cust_id = c.cust_id ',
  'group by 1 ',
  'order by 1'

 prepare sql_do_unl from sql_string
 declare curs_do_unl cursor for sql_do_unl
```

Optimizer Directives

Can also use the following syntax:

```
declare curs_do_unl cursor for
SQL
  select {+USE_HASH(o)} c.state,
         max(order_date)
    from order o, cust c
    where o.cust_id = c.cust_id
    group by 1
    order by 1
END SQL
```

Types of Directives

- Access Methods
- Join Order
- Join Methods
- Star Join
- Optimization Goal
- Explain Plan
- Statement Cache

Not all directives are available in all engine versions
The following are for 12.10

Types of Directives: Access Methods

FULL	Performs a full-table scan
INDEX	Uses the index specified to access the table
INDEX_ALL *or* MULTI_INDEX	Access the table using the specified indexes (Multi-index scan)
INDEX_SJ	Use the specified index to scan the table in an index self-join path.
AVOID_FULL	No full-table scan on the listed table
AVOID_INDEX	Does not use any of the specified indexes
AVOID_INDEX_SJ	Does not use an index self-join path for the specified indexes
AVOID_MULTI_INDEX	Does not use a multi-index scan path for the specified table

Types of Directives: Join Order

ORDERED Join tables or views in the order in which they are referenced in the FROM clause of the query

Types of Directives: Join Methods

USE_NL	Forces nested loop join on specified tables
USE_HASH	Forces hash join on specified tables
AVOID_NL	Avoids nested loop join on specified tables
AVOID_HASH	Avoids hash join on specified tables

Types of Directives: Star Join

AVOID_FACT	At least one table must be specified. Do not use the table (or any table in the list of tables) as a fact table in star-join optimization.
AVOID_STAR_JOIN	The optimizer does not consider a star-join execution plan.
FACT	Exactly one table must be specified. Only consider the specified table as a fact table in the star-join execution plan.
STAR_JOIN	Favor a star-join plan, if one is possible.

Types of Directives: Optimization Goal

FIRST_ROWS *(N)* Tells the optimizer to choose a plan optimized to return the first *N* rows of the result set

ALL_ROWS Tells the optimizer to choose a plan optimized to return all of the results

"Query level" equivalent of:

- OPT_GOAL configuration parameter (instance level)
  - 0=First Rows, -1=All Rows (default)
- OPT_GOAL environment variable (environment level)
- SET OPTIMIZATION statement (session level)
  - FIRST_ROWS, ALL_ROWS

Types of Directives: Explain Plan

EXPLAIN

Turns SET EXPLAIN ON for the specified query

EXPLAIN AVOID_EXECUTE

Prevents the data manipulation statement from executing; instead, the query plan is printed to the explain output file

Types of Directives: Statement Cache

AVOID_STMT_CACHE

Prevent the statement from being stored in the statement cache. Forces the optimizer to reoptimize the statement every time that the statement is run.

Directives Examples: ORDERED

```
select /*+ ORDERED */
customer.lname, orders.order_num, items.total_price
from customer, orders, items
where customer.customer_num = orders.customer_num
  and orders.order_num = items.order_num
  and items.stock_num = 6 and items.manu_code = "SMT"

DIRECTIVES FOLLOWED:
ORDERED
DIRECTIVES NOT FOLLOWED:

Estimated Cost: 15
Estimated # of Rows Returned: 1

  1) informix.customer: SEQUENTIAL SCAN

  2) informix.orders: INDEX PATH

    (1) Index Name: informix. 102_4
        Index Keys: customer_num   (Serial, fragments: ALL)
        Lower Index Filter: informix.customer.customer_num = informix.orders.customer_num
NESTED LOOP JOIN

  3) informix.items: INDEX PATH

    (1) Index Name: informix. 105_12
        Index Keys: stock_num manu_code   (Serial, fragments: ALL)
        Lower Index Filter: (informix.items.manu_code = 'SMT' AND informix.items.stock_num = 6 )

DYNAMIC HASH JOIN
    Dynamic Hash Filters: informix.orders.order_num = informix.items.order_num
```

Tables are accessed in the order they are listed in the FROM clause

Directives Examples : Combine Directives

```
select /*+ ordered index(customer, zip_ix)
  avoid_index(orders," 102_4") */
c.lname, o.order_num, i.total_price
from customer c, orders o, items i
where c.customer_num = o.customer_num
  and o.order_num = i.order_num
  and stock_num = 6
  and manu_code = "SMT"
```

Directives Examples : Combine Directives

```
DIRECTIVES FOLLOWED:
ORDERED
INDEX ( customer zip_ix )
AVOID_INDEX ( orders  102_4 )
DIRECTIVES NOT FOLLOWED:

Estimated Cost: 22
Estimated # of Rows Returned: 1

  1) informix.c: INDEX PATH

    (1) Index Name: informix.zip_ix
        Index Keys: zipcode   (Serial, fragments: ALL)

  2) informix.o: SEQUENTIAL SCAN

DYNAMIC HASH JOIN
    Dynamic Hash Filters: informix.c.customer_num = informix.o.customer_num

  3) informix.i: INDEX PATH

    (1) Index Name: informix. 105_12
        Index Keys: stock_num manu_code   (Serial, fragments: ALL)
        Lower Index Filter: (informix.i.manu_code = 'SMT' AND informix.i.stock_num =
   6 )

DYNAMIC HASH JOIN
    Dynamic Hash Filters: informix.o.order_num = informix.i.order_num
```

Directives Examples : Errors

Check the Explain Plan to make sure that the directives were followed as expected:

```
.
.
DIRECTIVES FOLLOWED:
ORDERED
INDEX ( customer zip_ix )
DIRECTIVES NOT FOLLOWED:
AVOID_INDEX ( orders  101_4 ) Invalid Index Name Specified.
.
.
```

The query will still be executed even with invalid directives

Optimizer Directives: Pros & Cons

Pros:

- Forces the engine to execute the SQL the way that *we* want
- Sometimes *we* know better!!
- Great for testing different plans. *What if..?*

Cons:

- Forces the engine to execute the SQL the way that *we* want
- Sometimes *the engine* knows better!!
- If new indexes are added, number of rows changes significantly, or data distributions change…then a better execution plan may be available

Bufferpool Usage

- Obvious performance benefit when queries are satisfied from the cache instead of disk
- Monitoring the bufferpool can give insight into proper sizing and direct you to problem tables, problem queries and missing indexes

Bufferpool Turnover

- Measure the turnover of pages in the bufferpool
  - Art Kagel's calculation of Buffer Turnover

    Buffer Turnover Ratio = ((bufwrits + pagreads) / number of buffers) / time_since_profile_reset
  - Aim for below 10 - Lower is better
  - See ratios script on IIUG site for this and other performance ratios

Bufferpool Turnover

BTR Calculation:

```
select  bufsize,nbuffs,
        round(((( pagreads + bufwrites )
/nbuffs ) / (
   select (ROUND (((
   sh_curtime - sh_pfclrtime)/60)/60) )
   from sysshmvals ) ),1) BTR
from sysbufpool;
```

Bufferpool Turnover

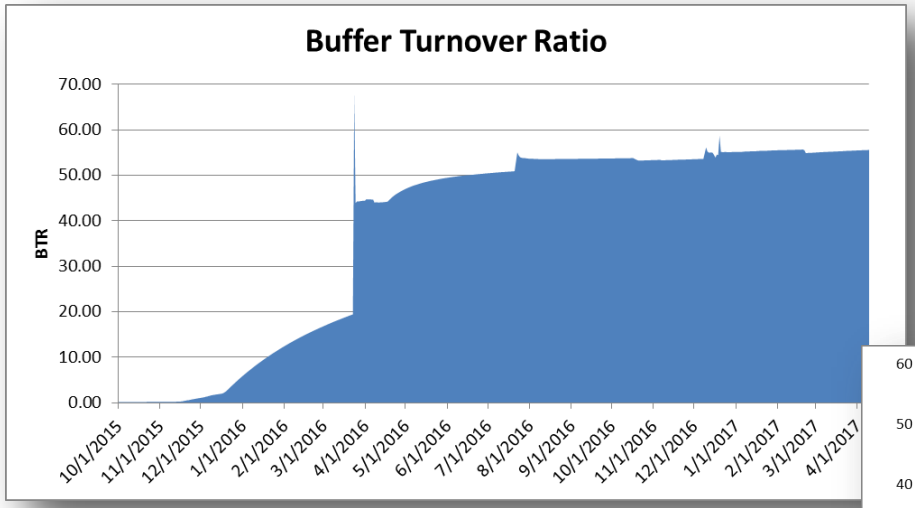
bufsize	nbuffs	btr
2048	250000	4.9
16384	10000	0.9

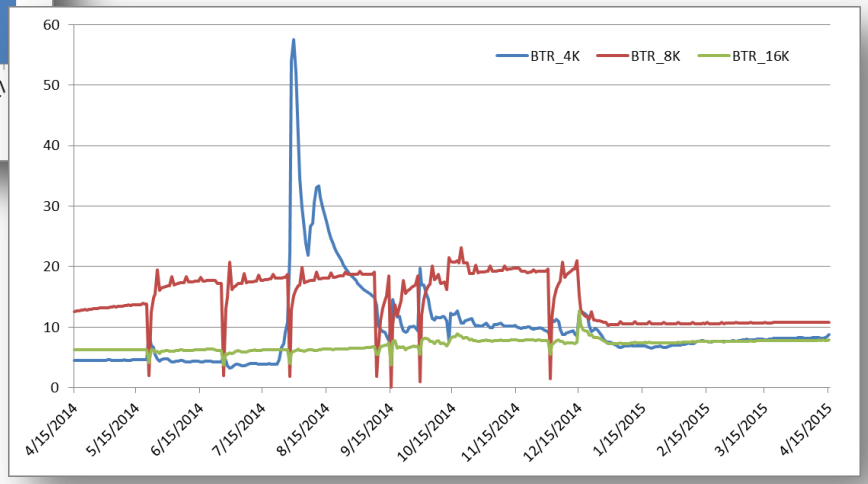
bufsize	nbuffs	btr
4096	50000	820.6

Way too high!

Bufferpool Turnover

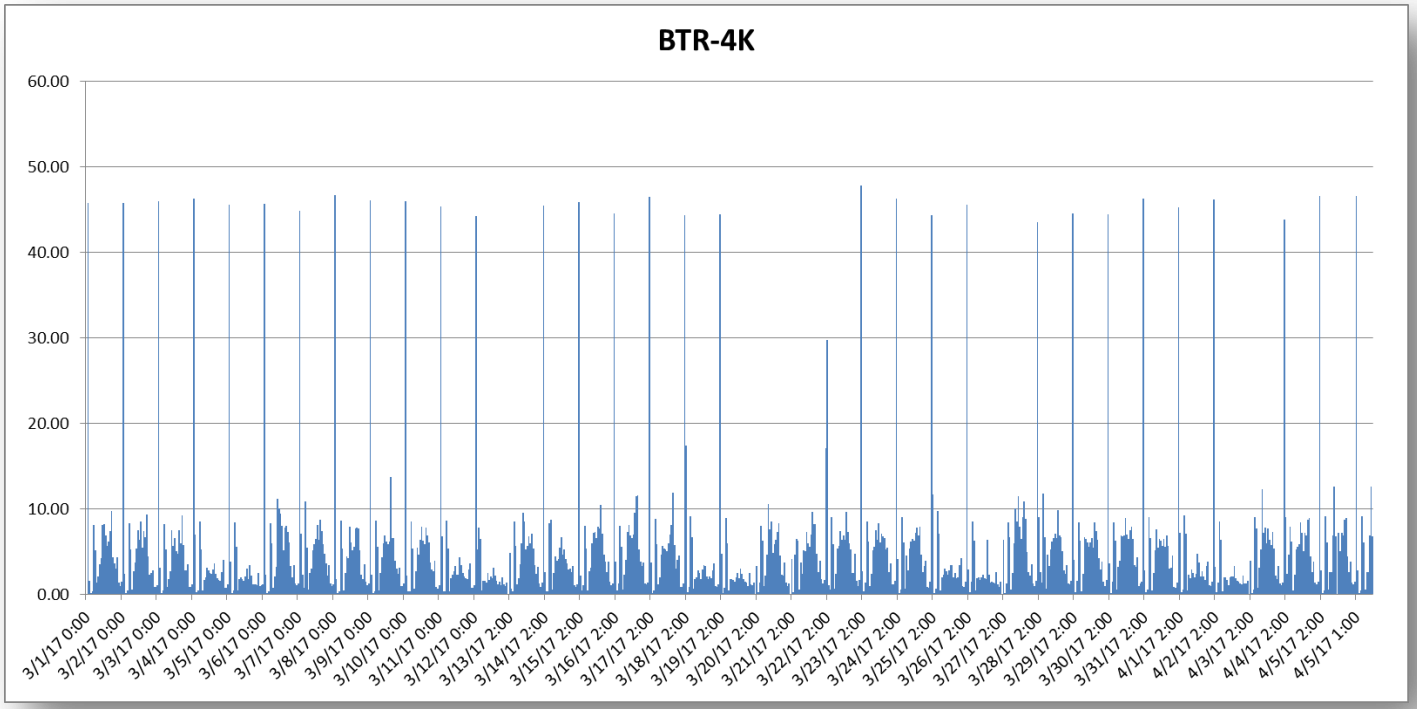
Useful to measure BTR ***over time*** to identify peak periods or look for changes





Bufferpool Turnover

Measure the turnover hourly using the delta of values to see when periods of higher turnover happen



The above showed that the bufferpool turnover spikes in the early morning hours during report generation

Bufferpool Use

- Find out what objects are using the bufferpool
- Look for tables/indexes that dominate the bufferpool
- Watch over time to see what swaps in/swaps out
- Identify the troublemakers

Don't assume that increasing the size of the bufferpool is the answer

onstat -P : Print partition buffer summary

```
Buffer pool page size: 2048
partnum   total    btree    data     other    dirty
0         28       0        12       16       11
1048577   600      0        586      14       6
1048578   3        1        1        1        0
1048579   1        0        0        1        0
.
.
.

51380228  2        0        1        1        0
51380229  7        0        6        1        0

Totals:   28674    915      3280     24479    83

Percentages:
Data   11.44
Btree  3.19
Other  85.37
```

Section for each pool

Table/Idx

How many pages of this table/index is in the buffer pool

Total pages used in this buffer pool

Bufferpool Use

- onstat -P : Find the largest consumers of each bufferpool – sort by “total” column for each pool
- Use a script to map the partnum to the database table or index and DBSpace (for fragmented tables)

```
select trim(dbsname)||":"||trim(tabname),
            dbinfo("DBSPACE", partnum)
     from systabnames
     where partnum = "${PARTNUM}"
```

Bufferpool Use

Buff Size: 2048 Num Buffers: 28673

total	btree	data	other		Database:Table/Idx	DBSpace
535	0	0	535	1% <-	sales:sales_fact	salesfact_2010
590	0	0	590	2% <-	sales:customers_idx1	salesidx2
600	0	586	14	2% <-	rootdbs:TBLSpace	rootdbs
638	0	0	638	2% <-	sales:sales_fact	salesfact_2003
701	0	0	701	2% <-	sales:sales_fact	salesfact_2002
715	0	0	715	2% <-	sales:sales_fact	salesfact_2011
785	0	0	785	2% <-	sales:sales_fact	salesfact_2004
832	0	0	832	2% <-	sales:sales_fact	salesfact_2006
921	0	0	921	3% <-	sales:sales_fact	salesfact_2005
1034	0	0	1034	3% <-	sales:customers	salesdbs1
1034	0	0	1034	3% <-	sales:customers	salesdbs2
1034	0	0	1034	3% <-	sales:customers	salesdbs3
1034	0	0	1034	3% <-	sales:customers	salesdbs4
1034	0	0	1034	3% <-	sales:customers	salesdbs5
1246	0	0	1246	4% <-	sales:sales_fact_store_idx	salesfact_idx1
1246	0	0	1246	4% <-	sales:sales_fact_promotion_idx	salesfact_idx5
1247	0	0	1247	4% <-	sales:sales_fact_time_idx	salesfact_idx3
1497	0	0	1497	5% <-	sales:sales_fact_product_idx	salesfact_idx4
2263	0	0	2263	7% <-	sales:sales_fact_customer_idx	salesfact_idx2
2412	0	0	2412	8% <-	sales:sales_fact_transaction_idx	salesfact_idx6

Bufferpool Use

Once have identified the tables that frequently consume a large part of the cache, or appear and disappear from the top-10, then can start tuning:

- Size the bufferpool to accommodate the commonly used tables to keep them in the cache
- Can tables be isolated to a different bufferpool?
- For tables that frequently trash the cache, identify queries that may be performing scans of the table
- Look at table level counters to get a better picture of what is happening

Bufferpool Use

Find those queries that are causing tables to consume a large amount of bufferpool pages by reviewing executing SQL and looking for references to the tables in question:

- **onstat -g sql 0 -r 1 >> sql.out 2>&1**
- **Loop through *running* sessions:**
  onstat -g sql > tempfile
  grep -i <*tablename*> tempfile
  Output tempfile for matches
- **onstat -g act for active threads**

Simple, but ugly - will include SQL that's no longer running

- Tables used in views can complicate the process
- Monitor table counters during capture period to find correlation with executing queries
- Get explain plans for likely candidates

Table Counters

- Get information at a table fragment/index level
- sysmaster:sysptprof
  (view between systabnames & sysptntab)
- Useful way to identify tables that may be performance bottlenecks
- Reset counters with onstat -z or restart instance
- Need TBLSPACE_STATS 1 set in ONCONFIG

Table Counters - sysmaster:sysptprof

```
dbsname      adtc_monitor
tabname      snapshot
partnum      9437506
lockreqs     197049665
lockwts      0
deadlks      0
lktouts      0
isreads      180714794
iswrites     9271
isrewrites   4087
isdeletes    17
bufreads     268432796
bufwrites    8661
seqscans     11459
pagreads     1273469
pagwrites    709
```

Identification

Locks

I/O

Table Counters – sysptprof – I/O

bufreads	number of pages from this table/index read from the bufferpool
bufwrites	number of pages from this table/index written to the bufferpool
pagreads	number of pages of this table/index that were read from disk
pagwrites	number of pages of this table/index written to disk

Table Counters – sysptprof – I/O

- Sort the results by pagreads/pagwrites to find those tables which result in high disk i/o
  - Use dbinfo("DBSPACE", partnum) to get the dbspace. Useful for fragmented tables.

```
select dbsname,
       tabname,
       bufreads,
       bufwrites,
       pagreads,
       pagwrites,
       dbinfo("DBSPACE", partnum) dbspace
from sysptprof
order by (pagreads+pagwrites) desc;
```

Table Counters – sysptprof – I/O

```
dbsname     adtc_monitor
tabname     snapshot
bufreads    273225824
bufwrites   8844
pagreads    1273593
pagwrites   722
dbspace     monitor_dbs1

dbsname     adtc_monitor
tabname     tabextent_idx1
bufreads    19237835
bufwrites   1986086
pagreads    1131902
pagwrites   58826
dbspace     monitor_dbs1
```

Table Counters – Sequential Scans

seqscans number of scans performed against this table

- Sorting by the number of scans alone will also include those small tables for which a scan is more efficient
- Combine with the ***size*** of the table to identify scans of large tables. Divide by the length of time since the counters were reset to get KB scanned/hour
  - Use systabinfo to get the size of a table - does not rely on update statistics

Sequential Scans per Hour

```
select first 5
       p.dbsname, p.tabname,
       p.seqscans,
       i.ti_nrows,
       round((i.ti_npused * i.ti_pagesize/1024),0) size_kb,
       round(p.seqscans /
          (select (((sh_curtime - sh_pfclrtime)/60)/60)
           from sysshmvals),0) scans_hr,
       round(p.seqscans * i.ti_nrows /
          (select (((sh_curtime - sh_pfclrtime)/60)/60)
           from sysshmvals),0) rows_scn_hr,
       round(p.seqscans * (i.ti_npused * i.ti_pagesize/1024) /
          (select (((sh_curtime - sh_pfclrtime)/60)/60)
           from sysshmvals),0) kb_scn_hr,
       dbinfo("DBSPACE", partnum) dbspace,
       (select ROUND ((((sh_curtime - sh_pfclrtime)/60)/60),2)
        from sysshmvals) hr_since_reset
from sysptprof p, systabinfo i
where p.partnum = i.ti_partnum
order by kb_scn_hr desc
```

Sequential Scans per Hour

```
dbsname          adtc_monitor
tabname          snapshot
seqscans         12552
ti_nrows         365324
size_kb          26846
scans_hr         101
rows_scn_hr      36970113
kb_scn_hr        2716766
dbspace          monitor_dbs1
hr_since_reset   124.03

dbsname          adtc_monitor
tabname          ratios_data
seqscans         165
ti_nrows         28655
size_kb          3186
scans_hr         1
rows_scn_hr      38119
kb_scn_hr        4238
dbspace          monitor_dbs1
hr_since_reset   124.03
```

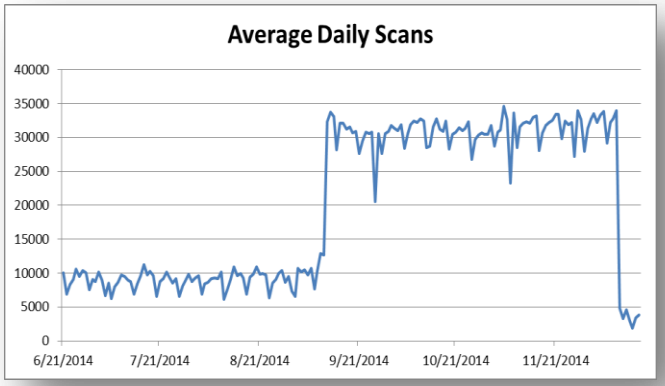
Scanning an average of 37 million rows/hour

Scanning an average of 2.6 GB/hour

Sequential Scans - History

As with other metrics, keeping a history can help diagnose problems and determine if something is still a problem

date	rows	scans
01/03/2015	210970	625
01/04/2015	211192	625
01/05/2015	211218	625
01/06/2015	220488	918
01/07/2015	230141	1110
01/08/2015	239930	1438
01/09/2015	249377	38353
01/10/2015	258954	251260
01/11/2015	259301	251261
01/12/2015	259451	251261
01/13/2015	269028	251512
01/14/2015	278812	251682
01/15/2015	287936	251939



Sudden jump, then very few scans

Sort Table Metrics

Sort sysptprof by the different counters to identify the worst offenders in each category

e.g. find the table/idx with the highest page reads…

```
./find_active pagreads

Database      Table/Idx              LockReqs lkwt d  l     isreads  iswrites isrewrt isdel   bufreads   bufwr  pagreads   pagwr    scan
-------------------------------------------------------------------------------------------------------------------------------------------
adtc_monitor  snapshot              845310135    0 0  0   770882949     40042   18150   250 1130772571   38903   5733642    3192   51117
adtc_monitor  tablayout_idx2         19591945    0 0  0    56975813   8048065       0 88941  111724402 8450563   3733561 3087704       0
adtc_monitor   1816849_949           13668136    0 0  0    47578769   6825398       0     0  154291923 6928766   2624054   17916       0
fastest_dba14 sales                 160000006    0 0  0    40952387  4429117740000000     0  17591752182463036   2505027 1231097       7
adtc_monitor  tabinfo_idx            13943013    0 0  0    45271160   6830893       0     0  124442435 6933681   2295151   17942       0
adtc_monitor  tablayout_idx1         24984506    0 0  0           0   8048065       0 88941  100392935 9882187   2005475  296124       0
adtc_monitor  partition_io_stats_pk  13890486    0 0  0           0   6825398       0     0   84820003 8259478   1811998  242031       0
adtc_monitor  partition_io_stats     13473799    0 0  0    36725259   6825398       0     0   81097885 7372452   1807371  178298       0
adtc_monitor  tabextent_idx1         20368931    0 0  0           0   6620720       0     0   71684437 7656056   1157531  198043       0
adtc_monitor  tabinfo                13422136    0 0  0    18373241   6830893
```

High number of scans may indicate a tuning opportunity

Table/Idx	isreads	bufreads	bufwr	**pagreads**	pagwr	scan
snapshot	770882949	1130772571	38903	**5733642**	3192	51117
tablayout_idx2	56975813	111724402	8450563	**3733561**	3087704	0
1816849_949	47578769	154291923	6928766	**2624054**	17916	0
sales	40952387	175917521	82463036	**2505027**	1231097	7

Table Counters – sysptprof – I/O

Use sysptprof to review counters for a specific table/index, while monitoring or capturing SQL to see values changing over time

```
Table/Idx    isreads  iswrites   isrewrt  isdel   bufreads    bufwr  pagreads   pagwr    scan
-----------------------------------------------------------------------------------------------
contract      889739      1940       289    176 1218475899     1129    152563     964  869301
...
Table/Idx    isreads  iswrites   isrewrt  isdel   bufreads    bufwr  pagreads   pagwr    scan
-----------------------------------------------------------------------------------------------
contract      889741      1940       289    176 1218478853     1129    152563     964  869303
...
Table/Idx    isreads  iswrites   isrewrt  isdel   bufreads    bufwr  pagreads   pagwr    scan
-----------------------------------------------------------------------------------------------
contract      889743      1940       289    176 1218481807     1129    152563     964  869305
```

Read operations are increasing, reads are satisfied from buffers and are the result of table scans

Update Statistics

- The Database Optimizer uses statistics gathered on the tables, their columns and indexes to determine which is the best query plan to use
- If the statistics don't exist or are inaccurate, then Informix may choose a poor query plan
- Make sure that statistics are rerun each time substantial changes are made to a table – including the number of rows and values in a column
  - This includes populating temp tables inside of applications

Update Statistics

- The basic (and most important) statistic is the number of rows in a table
  - Update statistics LOW for the table to update this statistic (stored in systables.nrows)
- Data Distributions denotes how unique or how common particular values are within a field
  - Update statistics MEDIUM performs a sampling of the records to determine distributions
  - Update statistics HIGH scans all records to determine distributions

Update Statistics Recommended Strategy

- Run UPDATE STATISTICS MEDIUM … WITH DISTRIBUTIONS ONLY for all columns in a table that do not head an index…or whole table
- Run UPDATE STATISTICS HIGH for all columns that head an index
- For each multicolumn index, execute UPDATE STATISTICS LOW for all of its columns
- For small tables, run UPDATE STATISTICS HIGH

You should be running update statistics regularly
Consider using Art's "dostats" or AUS

How Fresh are your Stats?

- Basic stat of how many rows in a table is stored in <*database*>:systables.nrows
- The time that LOW stats were last updated is now recorded in the column ustlowts

How Fresh are your Stats?

```
select
    tabname[1,20],
    ustlowts,
    round(nrows) nrows
from systables
where tabtype = "T";
```

tabname	ustlowts	nrows
systables	2017-04-09 01:12:08.00000	91
syscolumns	2017-04-09 01:12:08.00000	629
sysindices	2017-04-09 01:12:28.00000	130
systabauth	2017-04-09 01:12:08.00000	91
syscolauth	2017-04-09 01:12:09.00000	44
sysviews	2017-04-09 01:12:09.00000	5
sysusers	2017-04-09 01:12:26.00000	1
sysdepend	2017-04-09 01:12:09.00000	2
syssynonyms	2017-04-09 01:12:09.00000	0
syssyntable	2017-04-09 01:12:09.00000	0

How Accurate are your (LOW) Stats?

Compare the ***estimated*** row counts in systables.nrows with the ***actual*** row counts

```
select first 25 dbsname[1,12], tabname[1,20], est_nrows::int est, act_nrows::int
act,
   case
      when est_nrows = 0 and act_nrows > 0 then 100::smallint
      when est_nrows = act_nrows then 0
      else round(abs(act_nrows - est_nrows)/est_nrows*100)
   end diff
from ((
   select n.dbsname, t.tabname, round(t.nrows) est_nrows, sum(i.ti_nrows)
act_nrows
   from systables t, sysmaster:systabnames n, sysmaster:systabinfo i
   where t.tabname = n.tabname
     and n.dbsname = "<database name>"
     and n.partnum = i.ti_partnum
     and t.tabtype = "T"
   group by 1,2,3))
order by diff desc, act desc;
```

If actual and estimates are significantly different, then update low stats (at least)

dbsname	tabname	est	act	diff
stores_demo	sysattrtypes	5	16	220
stores_demo	sysxtddesc	3	4	33
stores_demo	sysxtdtypes	24	25	4
stores_demo	sysprocbody	3763	3763	0

How Fresh are your Stats?

Checking the column distributions (Simple)

```
dbschema -hd all -d <database>| egrep
"^Distribution|^Constructed | Mode,"
```

```
Distribution for informix.items.stock_num
Constructed on 2017-03-18 01:11:42.00000
High Mode, 0.500000 Resolution
Distribution for informix.items.manu_code
Constructed on 2017-03-18 01:11:42.00000
Medium Mode, 2.000000 Resolution, 0.950000 Confidence
```

How Fresh are your Stats?

Checking the column distributions

```
select t.tabname[1,18], c.colname[1,18],
 d.constr_time::datetime year to minute constructed,
 d.mode,
 d.resolution::decimal(5,2) res  ,
d.confidence::decimal(5,2) conf
from sysdistrib d, systables t, syscolumns c
where d.tabid = t.tabid
and t.tabid = c.tabid
and d.colno = c.colno
and d.seqno = 1
order by t.tabid, c.colno;
```

tabname	colname	constructed	mode	res	conf
items	item_num	2017-03-18 01:11	H	0.50	0.00
items	order_num	2017-03-18 01:11	H	0.50	0.00
items	stock_num	2017-03-18 01:11	H	0.50	0.00
items	manu_code	2017-03-18 01:11	M	2.00	0.95

Informix Best Practices Webcasts

Getting Started with Informix	***Replay Available***
Informix Configuration Part 1	***Replay Available***
Disks & Database Space Layout	***Replay Available***
Backups & High Availability	***Replay Available***
Informix Configuration Part 2	**May 18, 2017 2 PM EDT**
Informix Connection Manager	**Jun 29, 2017 2 PM EDT**
Informix Auditing	**Jul 27, 2017 2 PM EDT**

- Replays available at: http://advancedatatools.com/Informix/Webcasts.html
- Please register for each webcast here at: http://advancedatatools.com/Informix/NextWebcast.html

Informix Training in 2017

Advanced Informix Performance Tuning

July 10-13, 2017

Informix for Database Administrators

September 18-21, 2017

- All courses can be taken online on the web from your desk or at our training center in Virginia.
- We guarantee to *NEVER* cancel a course and will teach a course as long as one student is registered!

Questions?

Send follow-up questions to
mike@advancedatatools.com

Advanced DataTools

Informix Support and Training from the Informix Champions!

Advanced DataTools is an Advanced Level IBM Informix Data Management Partner, and has been an authorized Informix partner since 1993. We have a long-term relationship with IBM, we have priority access to high-level support staff, technical information, and Beta programs. Our team has been working with Informix since its inception, and includes 8 Senior Informix Database Consultants, 4 IBM Champions, 2 IIUG Director's Award winners, and an IBM Gold Consultant. We have Informix specialists Lester Knutsen and Art Kagel available to support your Informix performance tuning and monitoring requirements!

- ***Informix Remote DBA Support Monitoring***
- ***Informix Performance Tuning***
- ***Informix Training***
- ***Informix Consulting***
- ***Informix Development***

Free Informix Performance Tuning Webcast replays at:
http://advancedatatools.com/Informix/Webcasts.html

Call: (800) 807-6732 x101 or Email: info@advancedatatools.com
Web: http://www.advancedatatools.com

IBM Business Partner

Int Informix Users Group

April 23-27, 2017
Raleigh, NC, USA

C12
Informix Query Performance Tuning Basics

Mike Walker

Advanced DataTools Corporation

mike@advancedatatools.com