C12
Informix Query Performance Tuning Basics
Mike Walker
Advanced DataTools
mike@advanceddatatools.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

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

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

<table>
<thead>
<tr>
<th>netscb</th>
<th>thread name</th>
<th>sid</th>
<th>open</th>
<th>read</th>
<th>write</th>
</tr>
</thead>
<tbody>
<tr>
<td>7000007030e7cd0</td>
<td>sqlexec</td>
<td>951521</td>
<td>11:59:18</td>
<td>13:52:05</td>
<td>13:52:05</td>
</tr>
<tr>
<td>70000070284e600</td>
<td>sqlexec</td>
<td>951510</td>
<td>11:57:23</td>
<td>13:52:05</td>
<td>13:52:05</td>
</tr>
</tbody>
</table>

- 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:

<table>
<thead>
<tr>
<th>Process ID</th>
<th>Start Time</th>
<th>Query Time</th>
<th>Lock Time</th>
<th>Lock Duration</th>
<th>Flags</th>
<th>Current Time</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>63867891</td>
<td>11:24:30</td>
<td>14:19:21</td>
<td></td>
<td></td>
<td>&lt;</td>
<td>3 142216977</td>
<td>423644</td>
</tr>
<tr>
<td>63957715</td>
<td>17:54:30</td>
<td>14:19:21</td>
<td></td>
<td></td>
<td>&lt;</td>
<td>3 29900439</td>
<td>256</td>
</tr>
<tr>
<td>64157342</td>
<td>04:49:23</td>
<td>14:19:22</td>
<td></td>
<td></td>
<td>&lt;</td>
<td>3 36750125</td>
<td>64</td>
</tr>
<tr>
<td>55507969</td>
<td>21:03:13</td>
<td>14:19:23</td>
<td></td>
<td></td>
<td>&lt;</td>
<td>3 2180</td>
<td>16192</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>63867891</td>
<td>11:24:30</td>
<td>14:19:21</td>
<td></td>
<td></td>
<td>&lt;</td>
<td>3 142302481</td>
<td>423644</td>
</tr>
<tr>
<td>63957715</td>
<td>17:54:30</td>
<td>14:19:21</td>
<td></td>
<td></td>
<td>&lt;</td>
<td>3 29912222</td>
<td>256</td>
</tr>
<tr>
<td>64157342</td>
<td>04:49:23</td>
<td>14:19:22</td>
<td></td>
<td></td>
<td>&lt;</td>
<td>3 36842541</td>
<td>64</td>
</tr>
<tr>
<td>64158966</td>
<td>09:54:25</td>
<td>14:19:25</td>
<td></td>
<td></td>
<td>&lt;</td>
<td>3 28081252</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>63867891</td>
<td>11:24:30</td>
<td>14:19:21</td>
<td></td>
<td></td>
<td>&lt;</td>
<td>3 142387729</td>
<td>423644</td>
</tr>
<tr>
<td>64157342</td>
<td>04:49:23</td>
<td>14:19:22</td>
<td></td>
<td></td>
<td>&lt;</td>
<td>3 36928045</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>63867891</td>
<td>11:24:30</td>
<td>14:19:21</td>
<td></td>
<td></td>
<td>&lt;</td>
<td>3 142472977</td>
<td>423644</td>
</tr>
<tr>
<td>64157342</td>
<td>04:49:23</td>
<td>14:19:22</td>
<td></td>
<td></td>
<td>&lt;</td>
<td>3 37013549</td>
<td>64</td>
</tr>
<tr>
<td>55507969</td>
<td>21:03:13</td>
<td>14:19:26</td>
<td></td>
<td></td>
<td>&lt;</td>
<td>3 2180</td>
<td>16192</td>
</tr>
</tbody>
</table>
For OLTP, would like to see the SQL time change every second or so

<table>
<thead>
<tr>
<th>Session</th>
<th>Start Time</th>
<th>End Time</th>
<th>Duration</th>
<th>SQL Time</th>
<th>CPU Time</th>
</tr>
</thead>
</table>

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

• 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

<table>
<thead>
<tr>
<th>session</th>
<th>user</th>
<th>tty</th>
<th>pid</th>
<th>hostname</th>
<th>threads</th>
<th>memory</th>
<th>used</th>
<th>#RSAM</th>
<th>total</th>
<th>memory</th>
<th>used</th>
<th>dynamic explain</th>
</tr>
</thead>
<tbody>
<tr>
<td>10657</td>
<td>informix</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>0</td>
<td>16384</td>
<td>12480</td>
<td>off</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10653</td>
<td>informix</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>0</td>
<td>16384</td>
<td>12480</td>
<td>off</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10563</td>
<td>informix</td>
<td>2</td>
<td>4243</td>
<td>apollo</td>
<td>1</td>
<td>73728</td>
<td>64480</td>
<td>on</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10028</td>
<td>informix</td>
<td>-</td>
<td>0</td>
<td>apollo</td>
<td>1</td>
<td>335872</td>
<td>321728</td>
<td>off</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10011</td>
<td>informix</td>
<td>-</td>
<td>0</td>
<td>apollo</td>
<td>1</td>
<td>241664</td>
<td>100072</td>
<td>off</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>informix</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>1</td>
<td>626688</td>
<td>472280</td>
<td>off</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>informix</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>1</td>
<td>626688</td>
<td>471576</td>
<td>off</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>informix</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>1</td>
<td>618496</td>
<td>494080</td>
<td>off</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>informix</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>1</td>
<td>102400</td>
<td>86784</td>
<td>off</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Dynamic Query Plans

onstat -g sql

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

<table>
<thead>
<tr>
<th>Sess</th>
<th>SQL</th>
<th>Current</th>
<th>Iso Lock</th>
<th>SQL</th>
<th>ISAM</th>
<th>F.E.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Stmt type</td>
<td>Database</td>
<td>Lvl</td>
<td>Mode</td>
<td>ERR</td>
<td>ERR</td>
</tr>
<tr>
<td>10563</td>
<td>-</td>
<td>adtc_monitor</td>
<td>CR</td>
<td>Not Wait</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10028</td>
<td>-</td>
<td>adtc_monitor</td>
<td>CR</td>
<td>Not Wait</td>
<td>-255</td>
<td>0</td>
</tr>
<tr>
<td>10011</td>
<td>-</td>
<td>adtc_monitor</td>
<td>CR</td>
<td>Not Wait</td>
<td>-255</td>
<td>0</td>
</tr>
<tr>
<td>44</td>
<td>-</td>
<td>sysadmin</td>
<td>DR</td>
<td>Wait 5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>43</td>
<td>-</td>
<td>sysadmin</td>
<td>DR</td>
<td>Wait 5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>42</td>
<td>-</td>
<td>sysadmin</td>
<td>DR</td>
<td>Wait 5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>41</td>
<td>-</td>
<td>sysadmin</td>
<td>CR</td>
<td>Not Wait</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
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 shown because used: onmode –Y <sid> 1

Query Statistics

<table>
<thead>
<tr>
<th>Table map</th>
<th>Internal name</th>
<th>Table name</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>snapshot</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>type</th>
<th>table</th>
<th>rows_prod</th>
<th>est_rows</th>
<th>rows_scan</th>
<th>time</th>
<th>est_cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>scan</td>
<td>t1</td>
<td>1203641</td>
<td>1199409</td>
<td>1203641</td>
<td>00:00.00</td>
<td>79971</td>
</tr>
</tbody>
</table>
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 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 &
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:
------------------

<table>
<thead>
<tr>
<th>Internal name</th>
<th>Table name</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>c</td>
</tr>
<tr>
<td>t2</td>
<td>o</td>
</tr>
</tbody>
</table>

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

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

```sql
set explain file to "slow2.exp";
set explain on avoid_execute;
update orders
set ship_instruct = null
where customer_num = 104;
```
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 SQL

Cost/Rows Returned/Temp Files/Directives

Table 1 : Name & Access Method

Table 1 : Filters

Table 1 : Index Info

Table 2 : Name & Access Method

Table 2 : Filters

Table 2 : Index Info

Table 1 & 2 : Join Method

Subqueries

Query Statistics (if enabled)

Repeated for other tables
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
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
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

customer

Read ALL Rows

order

Read rows for cust_id
Reject those that don’t match filters

cust_type = ‘PREF’
Query Plans – Breaking it Down

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**

Table map:

<table>
<thead>
<tr>
<th>Internal name</th>
<th>Table name</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>c</td>
</tr>
<tr>
<td>t2</td>
<td>o</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>type</th>
<th>table</th>
<th>rows_prod</th>
<th>est_rows</th>
<th>rows_scan</th>
<th>time</th>
<th>est_cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>scan</td>
<td>t1</td>
<td>25000</td>
<td>10000</td>
<td>100000</td>
<td>00:00.41</td>
<td>15501</td>
</tr>
<tr>
<td>scan</td>
<td>t2</td>
<td>6040</td>
<td>90000</td>
<td>250000</td>
<td>00:14.76</td>
<td>4</td>
</tr>
</tbody>
</table>

This is where the time was spent

**Review the “Filters”**

Including the filter columns in the index will reduce the rows scanned and the rows discarded.
Query Statistics

Table map:

<table>
<thead>
<tr>
<th>Internal name</th>
<th>Table name</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>c</td>
</tr>
<tr>
<td>t2</td>
<td>o</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>type</th>
<th>table</th>
<th>rows_prod</th>
<th>est_rows</th>
<th>rows_scan</th>
<th>time</th>
<th>est_cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>scan</td>
<td>t1</td>
<td>25000</td>
<td>10000</td>
<td>100000</td>
<td>00:00.40</td>
<td>15501</td>
</tr>
<tr>
<td>scan</td>
<td>t2</td>
<td>6040</td>
<td>128571</td>
<td>6100</td>
<td>00:01.15</td>
<td>1</td>
</tr>
</tbody>
</table>

Faster!

New index includes `ship_method`

```
cREATE INDEX order_ix3 ON order (cust_id, ship_method)
```

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

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')`

<table>
<thead>
<tr>
<th>type</th>
<th>table</th>
<th>rows_prod</th>
<th>est_rows</th>
<th>rows_scan</th>
<th>time</th>
<th>est_cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>scan</td>
<td>t1</td>
<td>1000000</td>
<td>71429</td>
<td>1000000</td>
<td>00:04.55</td>
<td>280001</td>
</tr>
</tbody>
</table>

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`

<table>
<thead>
<tr>
<th>type</th>
<th>table</th>
<th>rows_prod</th>
<th>est_rows</th>
<th>rows_scan</th>
<th>time</th>
<th>est_cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>scan</td>
<td>t1</td>
<td>12200</td>
<td>71429</td>
<td>500000</td>
<td>00:06.16</td>
<td>2304336</td>
</tr>
</tbody>
</table>

- **Rows Read**
- **Fewer Rows**
- **Slower!**
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:**

<table>
<thead>
<tr>
<th>Internal name</th>
<th>Table name</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>product</td>
</tr>
<tr>
<td>t2</td>
<td>inventory</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>type</th>
<th>table</th>
<th>rows_prod</th>
<th>est_rows</th>
<th>rows_scan</th>
<th>time</th>
<th>est_cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>scan</td>
<td>t1</td>
<td>737</td>
<td>67311</td>
<td>673240</td>
<td>00:01.61</td>
<td>49460</td>
</tr>
<tr>
<td>scan</td>
<td>t2</td>
<td>242321</td>
<td>23493496</td>
<td>977563</td>
<td>02:26.49</td>
<td>101</td>
</tr>
</tbody>
</table>

- 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!
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 statistics:

```
<table>
<thead>
<tr>
<th>type</th>
<th>table</th>
<th>rows_prod</th>
<th>est_rows</th>
<th>rows_scan</th>
<th>time</th>
<th>est_cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>scan</td>
<td>t1</td>
<td>737</td>
<td>67311</td>
<td>673240</td>
<td>00:01.66</td>
<td>49460</td>
</tr>
<tr>
<td>scan</td>
<td>t2</td>
<td>242321</td>
<td>23496424</td>
<td>242321</td>
<td>00:00.82</td>
<td>2</td>
</tr>
<tr>
<td>nljoin</td>
<td></td>
<td>242321</td>
<td>23496424</td>
<td></td>
<td>00:02.54</td>
<td>199938</td>
</tr>
</tbody>
</table>
```

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**

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

**Non-Correlated**

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

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

Subquery executed for every row meeting criteria

Scan of table executed repeatedly

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 )
### Correlated Sub-Queries

<table>
<thead>
<tr>
<th>type</th>
<th>table</th>
<th>rows_prod</th>
<th>est_rows</th>
<th>rows_scan</th>
<th>time</th>
<th>est_cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>scan</td>
<td>t1</td>
<td>1</td>
<td>85</td>
<td>3265281</td>
<td>04:05:61</td>
<td>816703</td>
</tr>
</tbody>
</table>

Subquery statistics:

<table>
<thead>
<tr>
<th>type</th>
<th>table</th>
<th>rows_prod</th>
<th>est_rows</th>
<th>rows_scan</th>
<th>time</th>
<th>est_cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>scan</td>
<td>t1</td>
<td>8186</td>
<td>33</td>
<td>270015447</td>
<td>04:04:08</td>
<td>9236</td>
</tr>
</tbody>
</table>

Table map:

<table>
<thead>
<tr>
<th>Internal name</th>
<th>Table name</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>c</td>
</tr>
</tbody>
</table>

**Cumulative values**

- `rows_scan`: 270015447
- `time`: 04:04:08
- `est_cost`: 9236
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)

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

Change EXISTS to a NOT IN and remove the join

No longer a Correlated Subquery
Subquery will execute one time only
### Correlated Sub-Queries

<table>
<thead>
<tr>
<th>type</th>
<th>table</th>
<th>rows_prod</th>
<th>est_rows</th>
<th>rows_scan</th>
<th>time</th>
<th>est_cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>scan</td>
<td>t1</td>
<td>2</td>
<td>171</td>
<td>3265356</td>
<td>00:01.79</td>
<td>36899</td>
</tr>
</tbody>
</table>

Subquery statistics:

**Table map:**

<table>
<thead>
<tr>
<th>Internal name</th>
<th>Table name</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>c</td>
</tr>
</tbody>
</table>

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

```sql
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
```
Can also use the following syntax:

```sql
declare curs_do_unl1 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
<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FULL</td>
<td>Performs a full-table scan</td>
</tr>
<tr>
<td>INDEX</td>
<td>Uses the index specified to access the table</td>
</tr>
<tr>
<td>INDEX_ALL or</td>
<td>Access the table using the specified indexes</td>
</tr>
<tr>
<td>MULTI_INDEX</td>
<td>(Multi-index scan)</td>
</tr>
<tr>
<td>INDEX_SJ</td>
<td>Use the specified index to scan the table in an index self-join path.</td>
</tr>
<tr>
<td>AVOID_FULL</td>
<td>No full-table scan on the listed table</td>
</tr>
<tr>
<td>AVOID_INDEX</td>
<td>Does not use any of the specified indexes</td>
</tr>
<tr>
<td>AVOID_INDEX_SJ</td>
<td>Does not use an index self-join path for the specified indexes</td>
</tr>
<tr>
<td>AVOID_MULTI_INDEX</td>
<td>Does not use a multi-index scan path for the specified table</td>
</tr>
</tbody>
</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

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AVOID_FACT</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>AVOID_STAR_JOIN</strong></td>
<td>The optimizer does not consider a star-join execution plan.</td>
</tr>
<tr>
<td><strong>FACT</strong></td>
<td>Exactly one table must be specified. Only consider the specified table as a fact table in the star-join execution plan.</td>
</tr>
<tr>
<td><strong>STAR_JOIN</strong></td>
<td>Favor a star-join plan, if one is possible.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPLAIN</td>
<td>Turns SET EXPLAIN ON for the specified query</td>
</tr>
<tr>
<td>EXPLAIN AVOID_EXECUTE</td>
<td>Prevents the data manipulation statement from executing; instead, the query plan is printed to the explain output file</td>
</tr>
</tbody>
</table>
AvoidStmtCache

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

```sql
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 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
• Measure the turnover of pages in the bufferpool
  – Art Kagel’s calculation of Buffer Turnover
    \[
    \text{Buffer Turnover Ratio} = \frac{(\text{bufwrits} + \text{pagreads}) / \text{number of buffers}}{\text{time\_since\_profile\_reset}}
    \]
  – Aim for below 10 - Lower is better
  – See ratios script on IIUG site for this and other performance ratios
Bufferpool Turnover

BTR Calculation:

```sql
select bufsize,nbuffs,
    round(((pagreads + bufwrites) / nbuffs) / (select (ROUND (((sh_curtime - sh_pfclrt ime)/60)/60) ) from sysshmval s ) ),1) BTR
from sysbufpool;
```
### Bufferpool Turnover

<table>
<thead>
<tr>
<th>bufsize</th>
<th>nbuffs</th>
<th>btr</th>
</tr>
</thead>
<tbody>
<tr>
<td>2048</td>
<td>250000</td>
<td>4.9</td>
</tr>
<tr>
<td>16384</td>
<td>10000</td>
<td>0.9</td>
</tr>
<tr>
<td>4096</td>
<td>50000</td>
<td>820.6</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>partnum</th>
<th>total</th>
<th>btree</th>
<th>data</th>
<th>other</th>
<th>dirty</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>28</td>
<td>0</td>
<td>12</td>
<td>16</td>
<td>11</td>
</tr>
<tr>
<td>1048577</td>
<td>600</td>
<td>0</td>
<td>586</td>
<td>14</td>
<td>6</td>
</tr>
<tr>
<td>1048578</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1048579</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>51380228</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>51380229</td>
<td>7</td>
<td>0</td>
<td>6</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Totals: 28674 915 3280 24479 83

Percentages:
Data 11.44
Btree 3.19
Other 85.37
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)

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

**Buff Size:** 2048  
**Num Buffers:** 28673

<table>
<thead>
<tr>
<th>Database:Table/Idx</th>
<th>DBSpace</th>
</tr>
</thead>
<tbody>
<tr>
<td>sales:customers_idx1</td>
<td>salesidx2</td>
</tr>
<tr>
<td>rootdbs:TBLSpace</td>
<td>rootdbs</td>
</tr>
<tr>
<td>sales:customers</td>
<td>salesdbs1</td>
</tr>
<tr>
<td>sales:customers_idx1</td>
<td>salesidx2</td>
</tr>
<tr>
<td>sales:customers_idx2</td>
<td>salesdbs2</td>
</tr>
<tr>
<td>sales:customers_idx3</td>
<td>salesdbs3</td>
</tr>
<tr>
<td>sales:customers_idx4</td>
<td>salesdbs4</td>
</tr>
<tr>
<td>sales:customers_idx5</td>
<td>salesdbs5</td>
</tr>
<tr>
<td>sales:customers_transaction_idx</td>
<td>salesidx6</td>
</tr>
<tr>
<td>sales:customers_transaction_idx</td>
<td>salesidx6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>total</th>
<th>btree</th>
<th>data</th>
<th>other</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>535</td>
<td>0</td>
<td>0</td>
<td>535</td>
<td>1%</td>
<td>sales:sales_fact</td>
</tr>
<tr>
<td>590</td>
<td>0</td>
<td>0</td>
<td>590</td>
<td>2%</td>
<td>sales:customers_idx1</td>
</tr>
<tr>
<td>600</td>
<td>0</td>
<td>586</td>
<td>14</td>
<td>2%</td>
<td>rootdbs:TBLSpace</td>
</tr>
<tr>
<td>638</td>
<td>0</td>
<td>0</td>
<td>638</td>
<td>2%</td>
<td>sales:sales_fact</td>
</tr>
<tr>
<td>701</td>
<td>0</td>
<td>0</td>
<td>701</td>
<td>2%</td>
<td>sales:sales_fact</td>
</tr>
<tr>
<td>715</td>
<td>0</td>
<td>0</td>
<td>715</td>
<td>2%</td>
<td>sales:sales_fact</td>
</tr>
<tr>
<td>785</td>
<td>0</td>
<td>0</td>
<td>785</td>
<td>2%</td>
<td>sales:sales_fact</td>
</tr>
<tr>
<td>832</td>
<td>0</td>
<td>0</td>
<td>832</td>
<td>2%</td>
<td>sales:sales_fact</td>
</tr>
<tr>
<td>921</td>
<td>0</td>
<td>0</td>
<td>921</td>
<td>3%</td>
<td>sales:sales_fact</td>
</tr>
<tr>
<td>1034</td>
<td>0</td>
<td>0</td>
<td>1034</td>
<td>3%</td>
<td>sales:customers</td>
</tr>
<tr>
<td>1034</td>
<td>0</td>
<td>0</td>
<td>1034</td>
<td>3%</td>
<td>sales:customers</td>
</tr>
<tr>
<td>1034</td>
<td>0</td>
<td>0</td>
<td>1034</td>
<td>3%</td>
<td>sales:customers</td>
</tr>
<tr>
<td>1034</td>
<td>0</td>
<td>0</td>
<td>1034</td>
<td>3%</td>
<td>sales:customers</td>
</tr>
<tr>
<td>1034</td>
<td>0</td>
<td>0</td>
<td>1034</td>
<td>3%</td>
<td>sales:customers</td>
</tr>
<tr>
<td>1246</td>
<td>0</td>
<td>0</td>
<td>1246</td>
<td>4%</td>
<td>sales:sales_fact_store_idx</td>
</tr>
<tr>
<td>1246</td>
<td>0</td>
<td>0</td>
<td>1246</td>
<td>4%</td>
<td>sales:sales_fact_promotion_idx</td>
</tr>
<tr>
<td>1247</td>
<td>0</td>
<td>0</td>
<td>1247</td>
<td>4%</td>
<td>sales:sales_fact_time_idx</td>
</tr>
<tr>
<td>1497</td>
<td>0</td>
<td>0</td>
<td>1497</td>
<td>5%</td>
<td>sales:sales_fact_product_idx</td>
</tr>
<tr>
<td>2263</td>
<td>0</td>
<td>0</td>
<td>2263</td>
<td>7%</td>
<td>sales:sales_fact_customer_idx</td>
</tr>
<tr>
<td>2412</td>
<td>0</td>
<td>0</td>
<td>2412</td>
<td>8%</td>
<td>sales:sales_fact_transaction_idx</td>
</tr>
</tbody>
</table>
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

- 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

Simple, but ugly - will include SQL that’s no longer running
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>
<thead>
<tr>
<th>Database Name</th>
<th>dbuname</th>
<th>adtc_monitor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table Name</td>
<td>tabname</td>
<td>snapshot</td>
</tr>
<tr>
<td>Part Number</td>
<td>partnum</td>
<td>9437506</td>
</tr>
<tr>
<td>Lock Requests</td>
<td>lockreqs</td>
<td>197049665</td>
</tr>
<tr>
<td>Lock Waits</td>
<td>lockwts</td>
<td>0</td>
</tr>
<tr>
<td>Deadlocks</td>
<td>deadlks</td>
<td>0</td>
</tr>
<tr>
<td>Lock Timeouts</td>
<td>lktouts</td>
<td>0</td>
</tr>
<tr>
<td>Insert Reads</td>
<td>isreads</td>
<td>180714794</td>
</tr>
<tr>
<td>Insert Writes</td>
<td>iswrites</td>
<td>9271</td>
</tr>
<tr>
<td>Insert Rewrites</td>
<td>isrewrites</td>
<td>4087</td>
</tr>
<tr>
<td>Insert Deletes</td>
<td>isdeletes</td>
<td>17</td>
</tr>
<tr>
<td>Buffer Reads</td>
<td>bufreads</td>
<td>268432796</td>
</tr>
<tr>
<td>Buffer Writes</td>
<td>bufwrites</td>
<td>8661</td>
</tr>
<tr>
<td>Sequence Scans</td>
<td>seqscans</td>
<td>11459</td>
</tr>
<tr>
<td>Page Reads</td>
<td>pagreads</td>
<td>1273469</td>
</tr>
<tr>
<td>Page Writes</td>
<td>pagwrites</td>
<td>709</td>
</tr>
</tbody>
</table>
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.

```sql
select dbsname,
       tabname,
       bufreads,
       bufwrites,
       pagreads,
       pagwrites,
       dbinfo("DBSPACE", partnum) dbspace
from sysptprof
order by (pagreads+pagwrites) desc;
```
### Table Counters – sysptprof – I/O

<table>
<thead>
<tr>
<th>dbsname</th>
<th>adtc_monitor</th>
</tr>
</thead>
<tbody>
<tr>
<td>tabname</td>
<td>snapshot</td>
</tr>
<tr>
<td>bufreads</td>
<td>273225824</td>
</tr>
<tr>
<td>bufwrites</td>
<td>8844</td>
</tr>
<tr>
<td>pagreads</td>
<td>1273593</td>
</tr>
<tr>
<td>pagwrites</td>
<td>722</td>
</tr>
<tr>
<td>dbspace</td>
<td>monitor_dbs1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dbsname</th>
<th>adtc_monitor</th>
</tr>
</thead>
<tbody>
<tr>
<td>tabname</td>
<td>tabextent_idx1</td>
</tr>
<tr>
<td>bufreads</td>
<td>19237835</td>
</tr>
<tr>
<td>bufwrites</td>
<td>1986086</td>
</tr>
<tr>
<td>pagreads</td>
<td>1131902</td>
</tr>
<tr>
<td>pagwrites</td>
<td>58826</td>
</tr>
<tr>
<td>dbspace</td>
<td>monitor_dbs1</td>
</tr>
</tbody>
</table>
**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
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

<table>
<thead>
<tr>
<th>Database</th>
<th>Tablename</th>
<th>Seq Scans</th>
<th>Ti Nrows</th>
<th>Size KB</th>
<th>Scans/Hr</th>
<th>Rows Scn/Hr</th>
<th>KB Scn/Hr</th>
<th>DB Space</th>
<th>HR Since Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td>adtc_monitor</td>
<td>snapshot</td>
<td>12552</td>
<td>365324</td>
<td>26846</td>
<td>101</td>
<td>36970113</td>
<td>2716766</td>
<td>monitor_dbs1</td>
<td>124.03</td>
</tr>
<tr>
<td>adtc_monitor</td>
<td>ratios_data</td>
<td>165</td>
<td>28655</td>
<td>3186</td>
<td>1</td>
<td>38119</td>
<td>4238</td>
<td>monitor_dbs1</td>
<td>124.03</td>
</tr>
</tbody>
</table>

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.

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

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...

<table>
<thead>
<tr>
<th>Database</th>
<th>Table/Idx</th>
<th>LockReqs</th>
<th>lkwt</th>
<th>d</th>
<th>l</th>
<th>isreads</th>
<th>iswrites</th>
<th>isrewrt</th>
<th>isdel</th>
<th>bufreads</th>
<th>bufwr</th>
<th>pagreads</th>
<th>pagwr</th>
<th>scan</th>
</tr>
</thead>
<tbody>
<tr>
<td>adtc_monitor</td>
<td>snapshot</td>
<td>845310135</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>770882949</td>
<td>40042</td>
<td>18150</td>
<td>250</td>
<td>1130772571</td>
<td>38903</td>
<td>5733642</td>
<td>3192</td>
<td>51117</td>
</tr>
<tr>
<td>adtc_monitor</td>
<td>tablayout_idx2</td>
<td>19591945</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>56975813</td>
<td>8048065</td>
<td>0</td>
<td>88941</td>
<td>111724402</td>
<td>8450563</td>
<td>3733561</td>
<td>3087704</td>
<td>0</td>
</tr>
<tr>
<td>adtc_monitor</td>
<td>1816849_949</td>
<td>13668136</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>47578769</td>
<td>6825398</td>
<td>0</td>
<td>0</td>
<td>154291923</td>
<td>6928766</td>
<td>2624054</td>
<td>17916</td>
<td>0</td>
</tr>
<tr>
<td>fastest_dba14</td>
<td>sales</td>
<td>160000006</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>40952387</td>
<td>4429117740000000</td>
<td>0</td>
<td>17591752182462036</td>
<td>2505027</td>
<td>1231097</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>adtc_monitor</td>
<td>tabinfo_idx</td>
<td>13943013</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>45271160</td>
<td>6830893</td>
<td>0</td>
<td>0</td>
<td>124442435</td>
<td>6933681</td>
<td>2624054</td>
<td>17916</td>
<td>0</td>
</tr>
<tr>
<td>adtc_monitor</td>
<td>tablayout_idx1</td>
<td>24984506</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8048065</td>
<td>88941</td>
<td>0</td>
<td>100339235</td>
<td>9882187</td>
<td>2005475</td>
<td>296124</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>adtc_monitor</td>
<td>partition_io_stats_pk</td>
<td>13890486</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6825398</td>
<td>0</td>
<td>0</td>
<td>84820003</td>
<td>8259478</td>
<td>1811998</td>
<td>242031</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>adtc_monitor</td>
<td>partition_io_stats</td>
<td>13473799</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>36725259</td>
<td>6825398</td>
<td>0</td>
<td>0</td>
<td>81097885</td>
<td>7372452</td>
<td>1807371</td>
<td>178298</td>
<td>0</td>
</tr>
<tr>
<td>adtc_monitor</td>
<td>tabextent_idx1</td>
<td>20368931</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6620720</td>
<td>0</td>
<td>0</td>
<td>71684437</td>
<td>7656056</td>
<td>1157531</td>
<td>198043</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>adtc_monitor</td>
<td>tabinfo</td>
<td>13422136</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>18373241</td>
<td>6830893</td>
<td>0</td>
<td>0</td>
<td>5733642</td>
<td>3192</td>
<td>51117</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

High number of scans may indicate a tuning opportunity
Use sysptprof to review counters for a specific table/index, while monitoring or capturing SQL to see values changing over time.

<table>
<thead>
<tr>
<th>Table/Idx</th>
<th>isreads</th>
<th>iswrites</th>
<th>isrewrt</th>
<th>isdel</th>
<th>bufreads</th>
<th>bufwr</th>
<th>pagreads</th>
<th>pagwr</th>
<th>scan</th>
</tr>
</thead>
<tbody>
<tr>
<td>contract</td>
<td>889739</td>
<td>1940</td>
<td>289</td>
<td>176</td>
<td>1218475899</td>
<td>1129</td>
<td>152563</td>
<td>964</td>
<td>869301</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Table/Idx</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>contract</td>
<td>889741</td>
<td>1940</td>
<td>289</td>
<td>176</td>
<td>1218478853</td>
<td>1129</td>
<td>152563</td>
<td>964</td>
<td>869303</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Table/Idx</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>contract</td>
<td>889743</td>
<td>1940</td>
<td>289</td>
<td>176</td>
<td>1218481807</td>
<td>1129</td>
<td>152563</td>
<td>964</td>
<td>869305</td>
</tr>
</tbody>
</table>

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?

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

<table>
<thead>
<tr>
<th>tabname</th>
<th>ustlowts</th>
<th>nrows</th>
</tr>
</thead>
<tbody>
<tr>
<td>systables</td>
<td>2017-04-09 01:12:08.000000</td>
<td>91</td>
</tr>
<tr>
<td>syscolumnns</td>
<td>2017-04-09 01:12:08.000000</td>
<td>629</td>
</tr>
<tr>
<td>sysindices</td>
<td>2017-04-09 01:12:28.000000</td>
<td>130</td>
</tr>
<tr>
<td>systabauth</td>
<td>2017-04-09 01:12:08.000000</td>
<td>91</td>
</tr>
<tr>
<td>syscolauth</td>
<td>2017-04-09 01:12:09.000000</td>
<td>44</td>
</tr>
<tr>
<td>sysviews</td>
<td>2017-04-09 01:12:09.000000</td>
<td>5</td>
</tr>
<tr>
<td>sysusers</td>
<td>2017-04-09 01:12:26.000000</td>
<td>1</td>
</tr>
<tr>
<td>sysdepend</td>
<td>2017-04-09 01:12:09.000000</td>
<td>2</td>
</tr>
<tr>
<td>syssynonymys</td>
<td>2017-04-09 01:12:09.000000</td>
<td>0</td>
</tr>
<tr>
<td>syssyntable</td>
<td>2017-04-09 01:12:09.000000</td>
<td>0</td>
</tr>
</tbody>
</table>
How Accurate are your (LOW) Stats?

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

```sql
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)

<table>
<thead>
<tr>
<th>dbname</th>
<th>tabname</th>
<th>est</th>
<th>act</th>
<th>diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>stores_demo</td>
<td>sysattrtatypes</td>
<td>5</td>
<td>16</td>
<td>220</td>
</tr>
<tr>
<td>stores_demo</td>
<td>sysxtddesc</td>
<td>3</td>
<td>4</td>
<td>33</td>
</tr>
<tr>
<td>stores_demo</td>
<td>sysxtdtypes</td>
<td>24</td>
<td>25</td>
<td>4</td>
</tr>
<tr>
<td>stores_demo</td>
<td>sysprocbody</td>
<td>3763</td>
<td>3763</td>
<td>0</td>
</tr>
</tbody>
</table>
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

```sql
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;
```

<table>
<thead>
<tr>
<th>tabname</th>
<th>colname</th>
<th>constructed</th>
<th>mode</th>
<th>res</th>
<th>conf</th>
</tr>
</thead>
<tbody>
<tr>
<td>items</td>
<td>item_num</td>
<td>2017-03-18 01:11 H</td>
<td></td>
<td>0.50</td>
<td>0.00</td>
</tr>
<tr>
<td>items</td>
<td>order_num</td>
<td>2017-03-18 01:11 H</td>
<td></td>
<td>0.50</td>
<td>0.00</td>
</tr>
<tr>
<td>items</td>
<td>stock_num</td>
<td>2017-03-18 01:11 H</td>
<td></td>
<td>0.50</td>
<td>0.00</td>
</tr>
<tr>
<td>items</td>
<td>manu_code</td>
<td>2017-03-18 01:11 M</td>
<td></td>
<td>2.00</td>
<td>0.95</td>
</tr>
</tbody>
</table>
## Informix Best Practices Webcasts

<table>
<thead>
<tr>
<th>Topic</th>
<th>Date/Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Getting Started with Informix</td>
<td>Replay Available</td>
</tr>
<tr>
<td>Informix Configuration Part 1</td>
<td>Replay Available</td>
</tr>
<tr>
<td>Disks &amp; Database Space Layout</td>
<td>Replay Available</td>
</tr>
<tr>
<td>Backups &amp; High Availability</td>
<td>Replay Available</td>
</tr>
<tr>
<td>Informix Configuration Part 2</td>
<td>May 18, 2017 2 PM EDT</td>
</tr>
<tr>
<td>Informix Connection Manager</td>
<td>Jun 29, 2017 2 PM EDT</td>
</tr>
<tr>
<td>Informix Auditing</td>
<td>Jul 27, 2017 2 PM EDT</td>
</tr>
</tbody>
</table>

- Please register for each webcast here at: [http://advanceddatatools.com/Informix/NextWebcast.html](http://advanceddatatools.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@advanceddatatools.com
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://advanceddatatools.com/Informix/Webcasts.html

Call: (800) 807-6732 x101 or Email: info@advanceddatatools.com
Web: http://www.advanceddatatools.com
C12
Informix Query Performance Tuning Basics

Mike Walker
Advanced DataTools Corporation
mike@advanceddatatools.com