IDS Performance and Troubleshooting: Tips and Tricks

Tom Girsch
Hilton Hotels

A15
Wednesday, April 30, 2008 • 1:00 p.m. – 2:00 p.m.

2008 IIUG Informix Conference

www.iiug.org

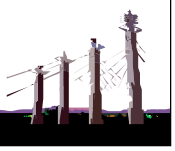
Introduction

- What We'll Cover
  - Using Configurable Page Sizes (CPS)
  - Finding CPU Hogs
  - Configuring Btree Scanners for Optimal Performance
  - Finding Out Who's Using A Table

Configurable Page Sizes (CPS)

- Introduced in IDS 10
- Control with `-k` option of `onspaces` command
- Page size must be a multiple of system default page size (which is 2K on most platforms, 4K on AIX)
- Maximum size is 16K
- Example:

```
onspaces -c -d dbs01 -p /links/chk01 -o 0 -s 8388608 -k 16
```

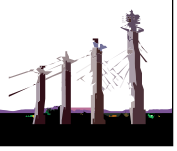


BUFFERPOOL Config Parameter

- Replaces old parameters BUFFERS, LRUS, LRU_MIN_DIRTY, and LRU_MAX_DIRTY – these parameters are now all set for each pool, rather than instance-wide
- Example:

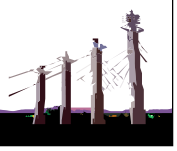
```
BUFFERPOOL size=8k,buffers=75000,lrus=128,lru_min_dirty=1,lru_max_dirty=2
```

- If no BUFFERPOOL value for a given page size, settings for "size=default" are used
- **Caution:** If a BUFFERPOOL setting is listed for *n*K buffers, that pool is initialized on startup, *whether or not* there are actually dbspaces of that size



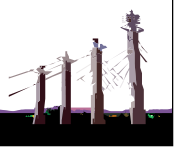
Benefits of CPS (1 of 3)

- Private buffer pools allow separation of high-access tables from other database tables (help avoid the “second tier” table problem)
- Checkpoint durations can be reduced, because LRU settings are done on a per-pool basis; Instead of 128 LRUs instance-wide, you can now have 128 LRUs *per buffer pool* (Note, however, that CLEANERS is still capped at 128 instance-wide)



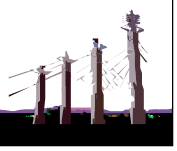
Benefits of CPS (2 of 3)

- Large row sizes can now be accommodated without having to resort to remainder pages
- Can optimize page size based on row size, to maximize rows per page
- Can isolate temp space buffers from all other buffers, if temp space is heavily used
- Can isolate heavily-scanned report tables from others



Benefits of CPS (3 of 3)

- Use 16K Pages for Indices:
  - Using a larger page size means more rows per page, which means fewer pages and, more importantly, fewer levels
  - Index buffers are isolated from data buffers (if tables are not also in 16K pages), allowing you to tune separately
  - **Caution:** Keep an eye on virtual shared memory allocation – indices use 2.5K * pagesize *each*



Configurable Page Size: Summary

- Use to Improve Checkpoint Times
- Use to Minimize Remainder Pages
- Use to Isolate High-Use Tables
- Use to Improve Index Efficiency
- Properly Configure BUFFERPOOL Settings
- Watch Out for Index Memory (up to 40K each)
- Don’t Get Carried Away!

Configurable Page Size Q&A

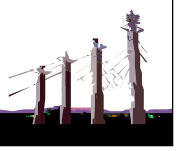
- Questions?

Finding CPU Hogs

- The Problem: Oninit processes are hoarding the CPU
- The Question: What session(s) is/are using the most CPU?
- The Answer: Repeated runs of onstat –g act
  - Run five times, a second apart
  - Sessions which appear multiple times are the most likely culprits

Procedure sp_show_hogs (1 of 2)

```
CREATE PROCEDURE sp_show_hogs() RETURNING CHAR(80);

DEFINE v_user   CHAR(8);
DEFINE v_host   LIKE sysmaster:syssessions.hostname;
DEFINE v_tid    INTEGER;
DEFINE v_sid    INTEGER;
DEFINE v_pid    INTEGER;
DEFINE v_cnt    INTEGER;

LET v_cnt = 0;

SELECT us_tid, us_sid
  FROM sysmaster:sysuserthreads AS u, sysmaster:systhreads AS t
 WHERE u.us_tid = t.th_id
   AND t.th_state = 0
   AND us_sid >= 100
  INTO TEMP sphogstmp WITH NO LOG;

LET v_cnt = v_cnt + 1;

WHILE (v_cnt < 5)
        SYSTEM("sleep 1");
        INSERT INTO sphogstmp
         SELECT us_tid, us_sid
           FROM sysmaster:sysuserthreads AS u, sysmaster:systhreads AS t
          WHERE u.us_tid = t.th_id
            AND t.th_state = 0
            AND us_sid >= 100;
        LET v_cnt = v_cnt + 1;
END WHILE;
```



Procedure sp_show_hogs (2 of 2)

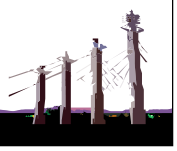
```
FOREACH SELECT us_tid, COUNT(*)
          INTO v_tid, v_cnt
          FROM sphogstmp
         GROUP BY us_tid
        HAVING COUNT(*) > 1
         ORDER BY 2 DESC

        FOREACH SELECT DISTINCT us_sid, username, pid, hostname
                  INTO v_sid, v_user, v_pid, v_host
                  FROM sphogstmp AS h, sysmaster:syssessions AS s
                 WHERE h.us_sid != DBINFO('sessionid')
                   AND h.us_tid = v_tid
                   AND h.us_sid = s.sid
                 ORDER BY 1
                RETURN "SID " || v_sid || " (" || v_cnt ||
                       "/5,User='" || TRIM(v_user) || "',Host='" ||
                       TRIM(v_host) || "',PID=" || v_pid || ")"
                       WITH RESUME;
        END FOREACH;

END FOREACH;

DROP TABLE sphogstmp;

END PROCEDURE;
```

CPU Hogs Q&A

- Questions?

BTSCANNERs

- What They Do
  - Keep indices clean and efficient
  - Clean up after DELETEs
  - Prioritize most DELETE-intensive indices first
- Configure with BTSCANNER parameter in $ONCONFIG
- Monitor with onstat –C (capital-C)
- Modify on-line with onmode –C (capital C)



Important Terms

- **Dirty Hit:** When an index scan incurred extra work because it encountered a committed deleted row
- **Hot List:** A working list, used by the BTSCANNER, of indices that have encountered a high number of dirty hits
- **Leaf Scan:** All of the leaf pages of an index are read into the buffer pool for scanning/cleaning
- **Range Scan:** A more efficient method of scanning larger indices for cleaning, using light scans to reduce the number of leaf pages read into the buffer pool
- **ALICE:** Adaptive Linear Index Cleaning (introduced in 9.40.xC6 and 10.00.xC5)

BTSCANNER Parameters

- **num:** Number of BTSCANNER threads to run (default = 1)
- **priority:**
  - high: Scanner threads have same priority as any other user thread in the system
  - low: Scanner threads only run when there are no user threads on the ready queue (default)
- **threshold:** How many "dirty hits" are required before an index is placed on the "hot list" (default = 500)
- **rangesize:** Minimum size (in pages) of index fragments to use range scans (default = -1, off)
- **alice:** Sparsely documented; default = 0 (off), but can be set to between 1 and 10 or 1 and 12 depending on where you look Higher number seems to mean more aggressive use of ALICE

Sample BTSCANNER Line

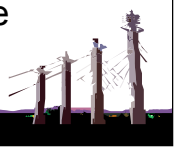
- A sample of what the BTSCANNER line might look like in the ${ONCONFIG} file:

```
BTSCANNER       num=2,priority=low,threshold=1000,rangesize=200
```

- Configures 2 low priority scanners, with a “dirty hit” threshold of 1000 and a range scan minimum index size of 200 pages

Montioring With `onstat -C`

- `onstat -C prof` (same as just "`onstat -C`")
- `onstat -C hot` – Displays "hot list"
- `onstat -C part` – Displays per-index stats
- `onstat -C clean` – Displays per-index cleaning stats
- `onstat -C range` – Displays range cleaning stats
- `onstat -C alice` – Displays ALICE stats
- `onstat -C all` – Displays all of the above

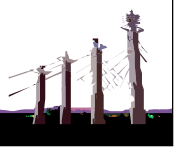


Output of `onstat -C [prof]`

```
$ onstat -C

Btree Cleaner Info
BT scanner profile Information
==============================
Active Threads                              4
Global Commands                       2000000   Building hot list
Number of partition scans                4207
Main Block                         0x00000001f57f7c50
BTC Admin                          0x00000001f6a19780


BTS info             id   Prio    Partnum      Key     Cmd
0x1f595ff68           0   High   0x00000000     0  2000000  Building hot list
    Number of leaves pages scanned                  101936981
    Number of leaves with deleted items                461925
    Time spent cleaning (sec)                           13003
    Number of index compresses                           9087
    Number of deleted items                            755482
    Number of index range scans                           273
    Number of index leaf scans                             18
    Number of index alice scans                             0
```

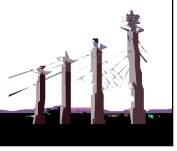
The "Hot List"

- Display with `onstat -C hot`
- Use this onstat to determine how well your BTSCANNERs are keeping up
- Automatically generated periodically (configurable)
- Shows which indices have the most "dirty hits"
- Lists partnums (in HEX) of indices on the list
- Asterisk indicates done cleaning
- Sample output follows



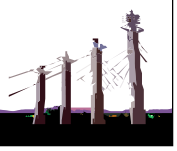
Output of `onstat -C hot`

```
$ onstat -C hot
Btree Cleaner Info

Index Hot List
==============
    Current Item           4     List Created          15:55:40
    List Size             98     List expires in            284 sec
    Hit Threshold       1000     Range Scan Threshold       100


Partnum         Key            Hits
0x07900002        1            4916 *
0x04E00003        1            4890 *
0x0E800004        1            4798 *
0x11800006        1            4759 *
0x001001B8        1            4652
0x10800002        1            4652
0x0E800005        1            4649
0x0FB00005        1            4557
0x11600005        1            4555
```



Partition Statistics

- Display partition statistics for all indices using `onstat -C part`
- Indices listed in partnum order
- Displays per-index stats on positions (number of times index was scanned), compresses (number of times index pages were merged), and splits (number of times an index had to split)
- Useful for determining busiest indices (sort on positions)
- Output sample follows



Output of `onstat -C part`

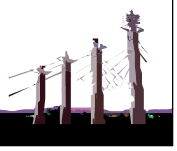
```
$ onstat -C part
Btree Cleaner Info


Index Statistics
================
 Partnum  Key      Positions   Compress      Split
0x00100002  1       10034091          0          0
0x00100021  1            277          0          0
0x00100021  3            153          0          0
0x00100022  1             44          0          0
0x00100022  3           1323          0          0
0x00100023  3            696          0          0
0x00100024  2              1          0          0
0x0010007d  1              0          0          0
0x0010007e  1            414          0          0
0x0010007e  2              0          0          0
0x0010007f  2              0          0          0
0x00100080  2              0          0          0
```

Cleaning Statistics

- Display cleaning statistics with `onstat -C clean`
- Stats are given for all indices that have been cleaned since engine was started
- Indices are listed in partnum order
- Displays per-index stats for pages examined, records deleted, and time spent cleaning
- Output sample follows

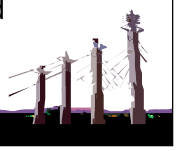
Output of `onstat -C clean`

```
$ onstat -C clean
Btree Cleaner Info

Index Cleaned Statistics
========================
 Partnum  Key       Dirty Hits  Clean Time  Pg Examined  Items Del  Pages/Sec
0x00100002  1             157           0            3         10       3.00
0x00100022  3             963           0            0          0       0.00
0x00100088  1              86           0            0          0       0.00
0x00100089  1             910           0            0         53       0.00
0x00100089  2             728           0            0          6       0.00
0x00100094  2            1804           0            0          0       0.00
0x00100095  2             276           0            0          0       0.00
0x00100096  1             208           0            0          0       0.00
0x001000a4  1               0           0            1         39       1.00
0x001000a7  1              12           0            0          0       0.00
0x001000a7  2              12           0            0          0       0.00
0x001000b7  1               0           0            1         39       1.00
0x001000b9  1               4           0            0          0       0.00
0x00100105  1             460           0            0          0       0.00
```



Range Scans

- A more efficient way of cleaning large indices
  - The whole index is first scanned using a light scan (bypassing the buffer pool) to determine the lowest and highest pages of the index that need cleaning (i.e., contains deleted rows)
  - Only that subset ("range") of the index is read into the buffer pool for cleaning
- Display range scan statistics using `onstat -C range`
- Lists buffer savings that would be achieved by using a range scan, whether or not a range scan was actually used



Output of `onstat -C range`

```
$ onstat -C range
Btree Cleaner Info

Cleaning Range Statistics
=========================
 Partnum  Key              Low          High          Size        Saving
0x00100002  1               5             5             8       100.0 %
0x00100088  1               1             1             8       100.0 %
0x00100089  2              10            67            72        20.8 %
0x00100095  2              17            35            40        55.0 %
0x00100096  1              10            17            32        78.1 %
0x001000a7  1               6            10            24        83.3 %
0x001000a7  2              14            19            24        79.2 %
0x001000b9  1               1             1             8       100.0 %
0x00100105  1               5             8            16        81.2 %
0x00100105  2               2             2            16       100.0 %
0x0010015f  1               4            21            32        46.9 %
0x0010015f  2              22            23            32        96.9 %
0x001001a0  1               7            14            40        82.5 %
0x001001a0  2              20            21            40        97.5 %
```

ALICE Scans

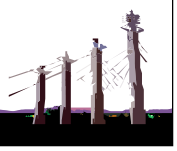
- This Slide Intentionally Left Blank
- Demand better documentation from IBM!

Using `onmode -C` (1 of 3)

- BTSCANNER configuration can be modified on-line using `onmode -C`
  - `onmode -C start [num]` – starts *num* additional BTSCANNER threads
    - Limit 128 at a time
    - Default 1 if unspecified
  - `onmode -C stop [num]` – stops *num* additional BTSCANNER threads
    - Limit 128 at a time
    - Default 1 if unspecified
    - If *num* exceeds the number actually running, all BTSCANNER threads are stopped, and *no index cleaning takes place*. Be *very* careful with this!

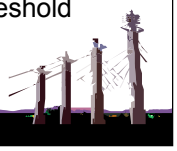
Using `onmode -C` (2 of 3)

- `onmode -C threshold` *`num`* – Sets "dirty hit" threshold to *`num`*
  - Default is 500 if not specified in $ONCONFIG or by previous `onmode -C` command
  - If *`num`*=0, every index in the system with a committed deleted row is cleaned, after which the threshold resets to 500, *no matter what* the $ONCONFIG `BTSCANNER` parameter says
  - If *`num`*=-1, every index in the system is cleaned, whether or not it has a committed deleted row (not sure why you'd ever do this). Same reset-to-500 warning applies.



Using `onmode -C` (3 of 3)

- `onmode -C rangesize` *`num`* – Sets minimum index size for range scanning to *`num`* pages
  - Default is -1 (off) if not specified in $ONCONFIG or by previous `onmode -C` command
- `Onmode -C duration` *`secs`* – Sets the expiration time for the "hot list" to *`secs`* seconds
  - Default is 300 seconds (5 minutes)
  - When the duration has expired, a new hot list is generated by the next available BTSCANNER thread, even if there are still items remaining to be worked on the existing hot list (that way, the dirtiest indices always get a high priority)
  - If the hot list is finished before the duration is up, and indices exist with a dirty hit rate larger than the threshold specified, a new hot list is immediately generated

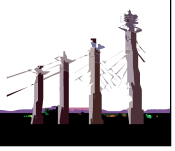


Tuning BTSCANNERs

- Start by monitoring hot list
  - How big is the list?
  - How many dirty hits per index?
- Adjust number of scanners to bring list size down
- Adjust threshold upward to spend more time on higher-priority indices

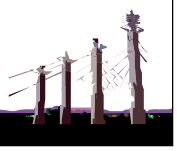
BTSCANNER Recommendations

- Run scanners in “high” priority mode (even if you don’t the engine will likely change them to high anyway)
- Run multiple scanner threads whenever possible, but don’t run more than (# CPUVPs – 1) on multi-CPUVP instances
- Turn range scanning on, and use `onstat –C range` to monitor effectiveness; adjust as necessary
- The default threshold of 500 is a good starting point, but monitor the hot list with `onstat –C hot`; if the list is too long or there are too many heavy-hitters, adjust upward
- Enabling ALICE may help, if IBM ever documents this
- YMMV!



Getting Useful Info From `onstat -C`

- Write a script to replace HEX numbers with meaningful values
- Sample shell script `onstat_C.sh` follows
- **DISCLAIMER:** My scripts use the “brute force” method; there is doubtless a more elegant way to do this



Script: `onstat_C.sh` (1 of 3)

```
#!/bin/ksh
#
# onstat_C.sh:  Replace hex values with database/table names in onstat -C output
#
# Created:      13 Mar 2008 by tjg
#
USAGE='usage: ${0} [ alice | clean | hot | part | range ]'
if [ ${#} -ne 1 ]
then
        echo "${0}: incorrect # of arguments (${#})" 1>&2
        echo ${USAGE} 1>&2
        exit 1
fi
if [ "${1}" != "alice" -a "${1}" != "clean" -a "${1}" != "hot" -a "${1}" != "part" -a "${1}" != "ran
ge" ]
then
        echo "${0}: bad argument (${1})" 1>&2
        echo ${USAGE} 1>&2
        exit 1
fi
```

onstat_C.sh (2 of 3)

```
echo "Gathering hex data..."
onstat -C ${1} | sed -f /tmp/fixhex$$.sed > /tmp/onstatC$$.tmp
rm -f fixhex$$.sed
grep "^0x" /tmp/onstatC$$.tmp | cut -c1-10 > /tmp/idx$$.tmp
if [ "${1}" = "hot" ]
then
        echo "s/^Partnum /Database:Table                        /g" > /tmp/idx$$.sed
else
        echo "s/^ Partnum /Database:Table                         /g" > /tmp/idx$$.sed
fi
cat /dev/null > /tmp/idx$$.sql
split -900 /tmp/idx$$.tmp /tmp/idx$$
rm -f /tmp/idx$$.tmp
for FILE in /tmp/idx$$??
do
        cat 2>/dev/null >> /tmp/idx$$.sql << EOF
OUTPUT TO PIPE "cat"
 WITHOUT HEADINGS
SELECT "s/" || LOWER(HEX(partnum)) || "/" || TRIM(TRIM(dbsname) || ":" || tabname)::CHAR(40)
    || "/g"
  FROM systabnames
 WHERE partnum IN (
EOF
```

onstat_C.sh (3 of 3)

```
        for IDX in `cat ${FILE}`
        do
                echo "\t\"${IDX}\"::INTEGER," >> /tmp/idx$$.sql
        done
        cat >> /tmp/idx$$.sql << EOF
        "0xFFFFFFFF"::INTEGER
        );
EOF
done
rm -f /tmp/idx$$??
dbaccess sysmaster /tmp/idx$$.sql 2>/dev/null | grep -v "^$" >> /tmp/idx$$.sed
rm -f /tmp/idx$$.sql
split -900 /tmp/idx$$.sed /tmp/sedidx$$
rm -f /tmp/idx$$.sed
for FILE in /tmp/sedidx$$??
do
        cat /tmp/onstatC$$.tmp | sed -f ${FILE} > /tmp/onstatC$$new.tmp
        mv /tmp/onstatC$$new.tmp /tmp/onstatC$$.tmp
        rm -f ${FILE}
done
cat /tmp/onstatC$$.tmp
rm -f /tmp/onstatC$$.tmp
```

onstat_C.sh Output Sample

```
$onstat_C.sh hot
Gathering hex data...

IBM Informix Dynamic Server Version 10.00.FC8      -- On-Line -- Up 6 days
    16:16:11 -- 5419008 Kbytes

Btree Cleaner Info

Index Hot List
==============
    Current Item            3     List Created          14:47:11
    List Size               3     List expires in            287 sec
    Hit Threshold        1000     Range Scan Threshold       100


Database:Table                                Key            Hits
db1:ix_table001_00                              1            1418 *
db1:ix_big_table_0002_00                        1            1095 *
db1:ix_big_table_0002_02                        1            1011 *
```

BTSCANNERs Q&A

- Questions?

Finding Out Who's Using A Table

- Have You Ever Fought This Error?

```
$ dbaccess mydb << EOF
> ALTER TABLE foo ADD col3 DATE;
> EOF

Database selected.


  242: Could not open database table (informix.foo).

  106: ISAM error:  non-exclusive access.
Error in line 1
Near character position 28


Database closed.
```

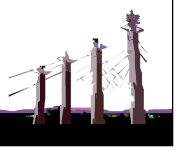
- The Solution Lies With `onstat -g opn`

Output of `onstat -g opn`

```
$ onstat -g opn

IBM Informix Dynamic Server Version 10.00.FC8     -- On-Line -- Up 5 days 04:20:28 -- 2316288 Kbytes
tid  rstcb              isfd  op_mode    op_flags   partnum    ucount ocount lockmode
3077 0x000000018470cbb0 0     0x00000400 0x00000397 0x00100080 2      2      0
3077 0x000000018470cbb0 1     0x00000002 0x00000003 0x00100080 2      2      0
3077 0x000000018470cbb0 2     0x00000400 0x00000017 0x00100079 1      1      0
```

Who's Using A Table – Manual

- Get a list of the partnum values for the table you're trying to modify (sysmaster:systables)
- Run `onstat -g opn` and look for those partnums
- Make note of the corresponding rstcb values associated with those partnums in `onstat -g opn`
- Run onstat –u and look for those rstcb values, tying them to the session IDs
- Or, write a script to do all this!

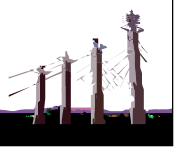


who_is_using.sh (1 of 3)

```
#!/bin/ksh
#
# who_is_using.sh:  Tell me who's using a table
#
# Created 31 Mar 2008 by tjg
#
# Usage: who_is_using.sh db table

if [ ${#} -ne 2 ]
then
        echo "${0}: incorrect number of arguments (${#})" 1>&2
        echo "usage: ${0} db table" 1>&2
        exit 1
fi
DB=${1}
TABLE=${2}
THISHOST=`uname -n | cut -c1-8`
```

who_is_using.sh (2 of 3)

```
echo "Getting partition list..."
(
dbaccess sysmaster 2>/dev/null << EOF
OUTPUT TO PIPE "cat"
 WITHOUT HEADINGS
 SELECT LOWER(HEX(partnum))
   FROM systabnames
  WHERE dbsname = "${DB}"
    AND tabname = "${TABLE}";
EOF
) | sed "s/  *$//g" | sed "s/^  *//g" | sed "s/0x00*//g" > /tmp/wiu_$$.tmp
>/tmp/wiu_$$b.tmp
for RSTCB in `cat /tmp/wiu_$$.tmp`
do
        echo "${RSTCB}|\c" >> /tmp/wiu_$$b.tmp
done
echo "Getting userthread list (onstat -u)..."
onstat -u > /tmp/wiu_onstat_u_$$.tmp
echo "Getting session list (syssessions)..."
dbaccess sysmaster > /tmp/wiu_sesinfo_$$.tmp 2> /dev/null << EOF
SELECT sid, username, pid, hostname
  FROM syssessions;
EOF
```



who_is_using.sh (3 of 3)

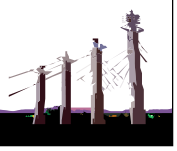
```
echo "Getting open tblspace list (onstat -g opn)..."
onstat -g opn | egrep "`cat /tmp/wiu_$$b.tmp`" | while read A RSTCB C D E F G H I
do
        echo ${RSTCB} | sed "s/^0x0*//g"
done | sort -u | while read RSTCB
do
        if [ -z "`grep ${RSTCB} /tmp/wiu_onstat_u_$$.tmp`" ]
        then
                echo "Skipping ${RSTCB}"
                continue
        fi
        grep ${RSTCB} /tmp/wiu_onstat_u_$$.tmp | while read A B SES D E F G H I J
        do
                grep "^${SES} " /tmp/wiu_sesinfo_$$.tmp | read A B C PID HOST F G H I
                if [ "${HOST}" = "${THISHOST}" ]
                then
                        echo "Session ${SES} (PID=${PID}) is using ${TABLE}"
                         ps -ef | grep "${PID}" | grep -v grep
                else
                        echo "Remote session ${SES} (PID=${PID}, HOST=${HOST}, USER=${USER}) is usin
g ${TABLE}"
                fi
        done
done
rm -f /tmp/wiu_$$.tmp /tmp/wiu_$$b.tmp /tmp/wiu_onstat_u_$$.tmp /tmp/wiu_sesinfo_$$.tmp
```

Output of `who_is_using.sh`

```
$who_is_using.sh mydb mytab
Getting partition list...
Getting userthread list (onstat -u)...
Getting session list (onstat -g ses)...
Getting open tblspace list (onstat -g opn)...
Remote session 1477993 (PID=25996, HOST=rmthost4.mydom.co) is using mytab
Session 1454418 (PID=26886) is using mytab
     mydb 26886 28441  0 10:35:58 ?       27:33 /local/apache/bin/httpd -D FOREGROUND
    -f /mydb/apps/webservices/conf/httpd
Session 155 (PID=28589) is using mytab
     mydb 28589     1  0   Mar 20 ?       55:23 /local/bin/perl -w /mydb/bin/KeyUpd -
    I keyupd-prod -C dbi:Informix:g
Remote session 1464942 (PID=23064, HOST=rmthost4.mydom.co) is using mytab
```

Wait For Offending User To Quit

- Undocumented Variable IFMX_DIRTY_WAIT
  - Set to *n* where *n* is the number of seconds to wait for dirty readers to go away
  - When all dirty readers have dropped off, Informix grabs the exclusive lock and makes the table change
  - **Careful!** Setting this variable locks out all new users from the table while the ALTER is pending!



"Who's Using A Table" Q&A

- Questions?

Session A15

IDS Performance and Troubleshooting: Tips and Tricks

Tom Girsch
Hilton Hotels
tom.girsch@hilton.com
INFORMIXDEV_DBA@hilton.com