Informix DB Daily Maintenance

Manually Re-load data files

Data loader programs on this Insight system are run by crontab automatically. You should understand and know where the data files are, just move or copy the data files to the right directory where the data-file-loader programs search for the data file:

Data loader program	**Data file and directory**
runner.10.ksh runner.all.ksh (all Qs except 71) runner.71.ksh	/dmqjtmp/rcp/*.vax /dmqjtmp/dmqvax/token/*.vax (Note: the actual data files are on /dmqjtmp/rcp, and you should touch the token files with the same filename of data files on dmqjtmp/dmqvax/token, then the data loader program will process these data files)
StartInsightBillingUpload.ksh	/dmqjtmp/rcp/*.recv
StartInsightUpload.ksh	/insight/local/scripts/iccdataupload/in/*.txt
StartInsightSetExpiryDates.ksh	/insight/local/scripts/ICCSetExpiryDates/*.txt

/var/adm/wtmp (who temp file) too large

#cp /var/adm/wtmp /recyclebox/lchen
#cat /dev/null > /var/adm/wtmp

There are no subheaders, line, or recaps in the informix database for this transaction number. Please reload it.

The wip appears to be from Jan 2014. The wip will need to be restored from the funnel file then we need to recreate the transaction. The history is below. It was completed, but no recaps went over. No errors showed up.

The programmers have a utility that allows them to recreate recaps from headers. We need to pull the funnel file from archives and then have them recreate recaps. Terry Bolton knows how. Everest knows where old funnel files are located. I think it's something like dsa3031.

hs_duty_rate table refresh

1. stopped data processor (runner) before new Q32 data files come(say 2:00PM)
2. new Q32 data files came at, say, 16:30PM
3. check you have enough free space (chunk files) on dbspace: datedbs1, this is the default dbspacse of database ip_0p, on which table **hs_duty_rate and hs_uom** is created
4. Purged **hs_duty_rate and hs_uom** at ,say, 19:30PM, and start runner at about 20:00PM

```
[lchen@ifx01 /home/lchen/tools/db/informix/etc] $ cat unload_HS.sql
CONNECT TO 'ip_0p@ipdb' USER 'lchen' USING 'admin12';
-- CONNECT TO 'ip_0p@ipdb' USER 'informix' USING 'infxrmvb';

UNLOAD TO "/recyclebox/lchen/hs_duty_rate.20150110" SELECT * FROM hs_duty_rate;
UNLOAD TO "/recyclebox/lchen/hs_uom.20150110" SELECT * FROM hs_uom;

-- TRUNCATE hs_duty_rate;
-- TRUNCATE hs_uom;
```

Reload HS tables to rollback if you find anything wrong after purging

```
[lchen@ifx01 /home/lchen/tools/db/informix/etc] $ cat load_HS.sql
CONNECT TO 'ip_0p@ipdb' USER 'informix' USING passwd

-- SET CONSTRAINTS,INDEXES,TRIGGERS FOR hs_duty_rate DISABLED;

-- TRUNCATE hs_duty_rate;

-- DROP INDEX 58858_619280;
-- ALTER TABLE hs_duty_rate DROP CONSTRAINT u58858_619280;

-- ALTER TABLE hs_duty_rate TYPE(RAW);

LOAD FROM "/recyclebox/lchen/hs_duty_rate.20150110" INSERT INTO hs_duty_rate;

-- ALTER TABLE hs_duty_rate TYPE(STANDARD);

-- ALTER TABLE hs_duty_rate ADD CONSTRAINT primary key (hsno,hstarifftrtmnt,effdate);
-- CREATE UNIQUE INDEX ON hs_duty_rate (hsno,hstarifftrtmnt,effdate);

-- SET CONSTRAINTS,INDEXES,TRIGGERS FOR hs_duty_rate ENABLED;

-- DISCONNECT CURRENT;
```

5. Q32 date filed completed loading at 23:40PM
6. /insight/local/scripts/ICCSetExpiryDates/StartInsightSetExpiryDates.ksh will update Table **hs_duty_rate**

Data processing LOGS

Find Data File process log in /usr/apps/dmq/beta,if you find some non zero-size file like ierr010.xxxx, it must mean some data process errors there. For history LOGS, /dmqjtmp/archiveBetaLog/beta

Storage consideration and Add Database storage space(chunck)

root@ifx01:/ # lspv

```
hdisk2          00ca32fde4198d51                    livedbvg        active
hdisk3          00ca32fde4198fc0                    archdbvg        active
hdisk4          00ca32fde41a128f                    appsvg          active
hdisk0          00ca32fd35a97b39                    rootvg          active
```

```
hdisk1          00ca32fd35a97d46                    rootvg          active
hdisk5          00ca32fdae1bdd5b                    archdbvg        active
hdisk6          00ca32fdae1d5a2a                    archdbvg        active
```

root@ifx01:/ # mpio_get_config -Av

```
Warning:  Unable to open message catalog.
Frame id 0:
    Storage Subsystem worldwide name: 60ab800264d8a0000456b76c6
    Controller count: 2
    Partition count: 1
    Partition 0:
    Storage Subsystem Name = 'LOIS_Imaging'
        hdisk#           LUN #   Ownership          User Label
        hdisk2               0   A (preferred)      lun067
        hdisk3               1   B (preferred)      lun068
        hdisk4               2   A (preferred)      lun069
        hdisk5               3   B (preferred)      lun106
        hdisk6               4   A (preferred)      lun107
```

We add 4G data files (chunk) to datadbs1 and 4G data files (chunk) to datadbs2

```
# touch /ach_dat1/ach_dat1.45

# touch /ach_dat2/ach_dat2.49

# chmod 660 /ach_dat1/ach_dat1.45

# chmod 660 /ach_dat2/ach_dat2.49

# chown informix:informix /ach_dat1/ach_dat1.45

# chown informix:informix /ach_dat2/ach_dat2.49
```

su – informix

For example: onspaces -a datadbs1 -p /ix_dat/ix_dat.2 -o 0 -s 1000000

```
$ onspaces -a datadbs1 -p /ach_dat1/ach_dat1.45 -o 0 -s 1000000

$ onspaces -a datadbs2 -p /ach_dat2/ach_dat2.49 -o 0 -s 1000000
```

All addition chunks to /ix_dat4

```
# chfs -a size=+1G /ix_dat4

$ onspaces -a datadbs1 -p /ix_dat4/ix_dat4.1 -o 0 -s 1000000
```

[lchen@ifx01 /home/lchen] $ echo "select count(*) from tariff"|dbaccess ip_0p

28274471 (2016-11-17)
29513860 (2017-02-06)

Database closed.

Reclaiming Unused Space Within an Extent

tblspace itself is the sum of allocated extents, not a single, contiguous allocation of space. The database server tracks tblspaces independently of the database.

Once the database server allocates disk space to a tblspace as part of an extent, that space remains dedicated to the tblspace. Even if all extent pages become empty after you delete data, the disk space remains unavailable for use by other tables.

Important:
When you delete rows in a table, the database server reuses that space to insert new rows into the same table. This section describes procedures to reclaim unused space for use by other tables.

You might want to resize a table that does not require the entire amount of space that was originally allocated to it. You can reallocate a smaller dbspace and release the unneeded space for other tables to use.

As the database server administrator, you can reclaim the disk space in empty extents and make it available to other users by rebuilding the table. To rebuild the table, use any of the following SQL statements:

- ALTER INDEX
- UNLOAD and LOAD
- ALTER FRAGMENT

Reclaiming Space in an Empty Extent with ALTER INDEX

If the table with the empty extents includes an index, you can execute the ALTER INDEX statement with the TO CLUSTER clause. Clustering an index rebuilds the table in a different location within the dbspace. All the extents associated with the previous version of the table are released. Also, the newly built version of the table has no empty extents.

Example:
```
//lchen@ipdev:/login/infown > cat alterindex.sql
       database sysadmin;
       alter index ix_ph_run_01 to cluster;
       close database;
```

For more information about the syntax of the ALTER INDEX statement, see the *IBM Informix Guide to SQL: Syntax*. For more information about clustering, see Clustering.

rootdbs full: Reclaiming Space in an Empty Extent with the UNLOAD and LOAD Statements or the onunload and onload Utilities

If the table does not include an index, you can unload the table, re-create the table (either in the same dbspace or in another one), and reload the data with the UNLOAD and LOAD statements or the **onunload** and **onload** utilities.

Example:
```
//lchen@ipdev:/login/infown > cat reclaimspace.sql

       database sysadmin;
       unload to "/recyclebox/lchen/ph_run" select * from ph_run;
       truncate ph_run;

       -- load from "/recyclebox/lchen/ph_run" insert into ph_run;

       close database;


Please check online log
$onstat  -rm
```

```
it will ask you to:
$oncheck -cDI sysadmin:"informix".ph_run

reads all pages, except for blobpages and sbpages, from the tblspace for the specified
database, table, fragment, or fragmants, and checks each pages for consistency, This
command compares entries in the bitmap page to the pages to verify mapping.
```

The BEGIN WORK statement is valid only in a database that supports transaction logging. This statement is not valid in an ANSI-compliant database. Each row that an UPDATE, DELETE, INSERT, or MERGE statement affects during a transaction is locked and remains locked throughout the transaction. A transaction that contains many such statements or that contains statements that affect many rows can exceed the limits that your operating system or the database server configuration imposes on the number of simultaneous locks. If no other user is accessing the table, you can avoid locking limits and reduce locking overhead by locking the table with the LOCK TABLE statement after you begin the transaction. Like other locks, this table lock is released when the transaction terminates. The example of a transaction on “Example of BEGIN WORK” on page 2-75 includes a LOCK TABLE statement. Important: Issue the BEGIN WORK statement only if a transaction is not in progress. If you issue a BEGIN WORK statement while you are in a transaction, the database server returns an error.

```
//lchen@ipdev:/login/infown > vi reclaimspace.sql

       "reclaimspace.sql" 8 lines, 194 characters
       database sysadmin;
       BEGIN WORK;
       lock table ph_run;
       -- unload to "/recyclebox/lchen/ph_run" select * from ph_run;
       -- truncate ph_run;
       load from "/recyclebox/lchen/ph_run" insert into ph_run;
       COMMIT WORK;
       close database;
```

For further information about selecting the correct utility or statement, see the *IBM Informix Migration Guide*. For more information about the syntax of the UNLOAD and LOAD statements, see the *IBM Informix Guide to SQL: Syntax*.

Releasing Space in an Empty Extent with ALTER FRAGMENT

You can use the ALTER FRAGMENT statement to rebuild a table, which releases space within the extents that were allocated to that table. For more information about the syntax of the ALTER FRAGMENT statement, see the *IBM Informix Guide to SQL: Syntax*.

There is an environment variable `IFX_DIRTY_WAIT` that is used to define to the number of seconds a DDL statement will wait for existing dirty readers to finish their access to the target table. When set, the variable also prevents new dirty readers from accessing the table.

The variable can be set in the Informix® Dynamic Server™ environment (before the database is started) or in the client environment. Setting it on the client side will override the setting on the server side.

Example:

`IFX_DIRTY_WAIT=`*`n`*

n
is a positive integer representing the number of seconds that your session will wait for a given dirty reader to finish accessing the target table. Ifthis amount of time is not enough time, the session returns the same error as it would without the variable being set.

Example:

Using the UNIX Korn shell and setting the timeout value to 300 seconds, you would set the environment variable as follows:

```
$ export IFX_DIRTY_WAIT=300

$ dbaccess sysadmin alterindex
```

Monitoring locks

You can analyze information about locks and monitor locks by viewing information in the internal lock table that contains stored locks.

View the lock table with **onstat -k**. Figure 1 shows sample output for **onstat -k**.

Figure 1. onstat -k output

```
Locks
address  wtlist   owner    lklist   type     tblsnum  rowid    key#/bsiz
300b77d0 0        40074140 0        HDR+S    10002    106        0
300b7828 0        40074140 300b77d0 HDR+S    10197    123        0
300b7854 0        40074140 300b7828 HDR+IX   101e4    0          0
300b78d8 0        40074140 300b7854 HDR+X    101e4    102        0
 4 active, 5000 total, 8192 hash buckets
```

In this example, a user is inserting one row in a table. The user holds the following locks (described in the order shown):

- A shared lock on the database
- A shared lock on a row in the **systables** system catalog table
- An intent-exclusive lock on the table
- An exclusive lock on the row

To determine the table to which the lock applies, execute the following SQL statement. For *tblsnum*, substitute the value shown in the **tblsnum** field in the **onstat -k** output.

```
SELECT *
   FROM SYSTABLES
   WHERE HEX(PARTNUM) = "tblsnum";
```

Where tblsnum is the modified value that **onstat -k** returns. For example, if **onstat -k** returns `10027f`, tbslnum is `0x0010027F`.

You can also query the **syslocks** table in the **sysmaster** database to obtain information about each active lock. The **syslocks** table contains the following columns.

Column	Description
dbsname	Database on which the lock is held
tabname	Name of the table on which the lock is held
rowidlk	ID of the row on which the lock is held (0 indicates a table lock.)
keynum	The key number for the row
type	Type of lock
owner	Session ID of the lock owner
waiter	Session ID of the first waiter on the lock

Monitoring lock waits and lock errors: You can view information about sessions, lock usage, and lock waits.

Summary: onstat -k will show you the locks. The owner column of onstat -k has the same value as the onstat -u column address. The onstat -u output should help you identify the owner of the locks in onstat -k. The owner's username will be listed in the user column of onstat -u The onstat - u output also has a sessid column. You can use the sessid value to find out more about the session that holds the lock. Run onstat -g ses <sessid value>.

If the application executes SET LOCK MODE TO WAIT, the database server waits for a lock to be released instead of returning an error. An unusually long wait for a lock can give users the impression that the application is hanging.

In Figure 1, the **onstat -u** output shows that session ID 84 is waiting for a lock (`L` in the first column of the **Flags** field). To find out the owner of the lock, use the **onstat -k** command.

Figure 1. onstat -u output that shows lock usage

```
onstat -u

Userthreads
address  flags   sessid user     tty    wait    tout locks nreads nwrites
40072010 ---P--D 7      informix -          0       0     0     35       75
400723c0 ---P--- 0      informix -          0       0     0      0        0
40072770 ---P--- 1      informix -          0       0     0      0        0
40072b20 ---P--- 2      informix -          0       0     0      0        0
40072ed0 ---P--F 0      informix -          0       0     0      0        0
40073280 ---P--B 8      informix -          0       0     0      0        0
40073630 ---P--- 9      informix -          0       0     0      0        0
400739e0 ---P--D 0      informix -          0       0     0      0        0
40073d90 ---P--- 0      informix -          0       0     0      0        0
 40074140 Y-BP--- 81    lsuto    4  50205788       0     4    106      221
400744f0 --BP--- 83     jsmit    -          0       0     4      0        0
400753b0 ---P--- 86     worth    -          0       0     2      0        0
40075760 L--PR-- 84      jones    3   300b78d8    -1     2      0        0
 13 active, 128 total, 16 maximum concurrent

onstat -k

Locks
address  wtlist   owner    lklist   type    tblsum rowid  key#/bsiz
300b77d0 0        40074140 0        HDR+S   10002  106     0
300b7828 0        40074140 300b77d0 HDR+S   10197  122     0
300b7854 0        40074140 300b7828 HDR+IX  101e4  0       0
 300b78d8 40075760 40074140 300b7854 HDR+X  101e4  100     0
300b7904 0        40075760 0             S   10002  106     0
300b7930 0        40075760 300b7904      S   10197  122     0
 6 active, 5000 total, 8192 hash buckets
```

To find out the owner of the lock for which session ID 84 is waiting:

1. Obtain the address of the lock in the **wait** field (`300b78d8`) of the **onstat -u** output.
2. Find this address (`300b78d8`) in the **Locks address** field of the **onstat -k** output.

The **owner** field of this row in the **onstat -k** output contains the address of the user thread (`40074140`).

3. Find this address (`40074140`) in the **Userthreads** field of the **onstat -u** output.

The **sessid** field of this row in the **onstat -u** output contains the session ID (`81`) that owns the lock.

To eliminate the contention problem, you can have the user exit the application gracefully. If this solution is not possible, you can stop the application process or remove the session with **onmode -z**.

Release the log files

The logical log contains a record of changes made to a database server instance. The logical-log records are used to roll back transactions, recover from system failures, and so on. The following parameters affect logical logging.

Configuration parameter	Description
DYNAMIC_LOGS	Determines whether the database server allocates new logical-log files automatically. For more information, see Logical log.
LOGBUFF	Determines the amount of shared memory reserved for the buffers that hold the logical-log records until they are flushed to disk. For information about how to tune the logical-log buffer, see Logical-log buffer.
LOGFILES	Specifies the number of logical-log files used to store logical-log records until they are backed up on disk. For more information, see Estimate the size and number of log files.
LOGSIZE	Specifies the size of each logical-log file.
LTXHWM	Specifies the percentage of the available logical-logspace that, when filled, triggers the database server to check for a long transaction. For more information, see Set high-watermarks for rolling back long transactions.
LTXEHWM	Specifies the point at which the long transaction being rolled back is given exclusive access to the logical log.
TEMPTAB_NOLOG	Disables logging on temporary tables.

Senario: database server suspends all processing. If the database server attempts to switch to the next online logical-log file but finds that the next log file in sequence is still in use, the database server immediately suspends all processing (Processing stops to protect the data in log files).

All following criteria must be satisfied before the database server frees a logical-log file for reuse:

1. The log file is backed up.
2. The logical-log file does not contain the oldest update not yet flushed to disk.
3. No records within the logical-log file are associated with open transactions.

So, the first thing to do is to Backup log files when filled

```
$ ontape -a
```

Secondary, The database server always forces a checkpoint when it switches to the last available log, if the previous checkpoint record or oldest update that is not yet flushed to disk is located in the log that follows the last available log, force a checkpoint:

```
$ onmode -c
```

switch the current log file to the next available log file manually:

```
$ onmode -l
```

Add log files manually, if no free space in logic log dbspace, you have to add chucks to logic log dbspace manually first.

```
$ onparams
```

Tips: If you do not want to wait until the transactions complete, take the database server to quiescent mode, then all the transactions will be roll back.

```
$ onmode -s
```

To quickly load a large, existing standard table

1. Drop indexes, referential constraints, and unique constraints.

2. Change the table to nonlogging.
   The following sample SQL statement changes a STANDARD table to nonlogging:
   ```
   ALTER TABLE largetab TYPE(RAW);
   ```

3. Load the table using a load utility such as **dbexport** or the High-Performance Loader (HPL).

   For more information on **dbexport** and **dbload**, see the *IBM Informix: Migration Guide*. For more information on HPL, see the *IBM Informix: High-Performance Loader User's Guide*.

4. Perform a level-0 backup of the nonlogging table.

   You must make a level-0 backup of any nonlogging table that has been modified before you convert it to STANDARD type. The level-0 backup provides a starting point from which to restore the data.

5. Change the nonlogging table to a logging table before you use it in a transaction.

   The following sample SQL statement changes a raw table to a standard table:
   ```
   ALTER TABLE largetab TYPE(STANDARD);
   ```

   Warning:
   It is recommended that you not use nonlogging tables within a transaction where multiple users can modify the data. If you need to use a nonlogging table within a transaction, either set Repeatable Read isolation level or lock the table in exclusive mode to prevent concurrency problems.

   For more information on standard tables, see the previous section, Advantages of Logging Tables.

6. Re-create indexes, referential constraints, and unique constraints

   Example:
   [lchen@ifx01 /home/lchen/tools/db/informix/etc] $ cat loadTable.sql

```
-- CONNECT TO 'ip_systest@systestdb' USER 'informix' USING 'passwd'
CONNECT TO 'ip_0p@ipdb' USER 'informix' USING 'passwd'
-- CONNECT TO 'sysadmin@systestdb';
-- SET CONSTRAINTS,INDEXES,TRIGGERS FOR hs_duty_rate DISABLED;
-- TRUNCATE hs_duty_rate;
-- DROP INDEX 58858_619251;
-- ALTER TABLE hs_duty_rate DROP CONSTRAINT u58858_619251;
-- ALTER TABLE hs_duty_rate TYPE(RAW);
-- LOAD FROM "/recyclebox/lchen/hs_duty_rate.20141004" INSERT INTO hs_duty_rate;
-- ALTER TABLE hs_duty_rate TYPE(STANDARD);
-- ALTER TABLE hs_duty_rate ADD CONSTRAINT primary key
(hsno,hstarifftrtmnt,effdate);
```

```
-- CREATE UNIQUE INDEX ON hs_duty_rate (hsno,hstarifftrtmnt,effdate);
-- SET CONSTRAINTS,INDEXES,TRIGGERS FOR hs_duty_rate ENABLED;
-- DISCONNECT CURRENT;
```

To quickly load a new, large table

1. Create a nonlogging table in a logged database.

   The following sample SQL statements creates a nonlogging table:

   ```
   CREATE DATABASE history WITH LOG;
   CONNECT TO DATABASE history;
   CREATE RAW TABLE history (...
   );
   ```

2. Load the table using a load utility such as **dbexport** or the High-Performance Loader (HPL).

   For more information on **dbexport** and **dbload**, see the *IBM Informix: Migration Guide*. For more information on HPL, see the *IBM Informix: High-Performance Loader User's Guide*.

3. Perform a level-0 backup of the nonlogging table.

   You must make a level-0 backup of any nonlogging table that has been modified before you convert it to STANDARD type. The level-0 backup provides a starting point from which to restore the data.

4. Change the nonlogging table to a logging table before you use it in a transaction.

   The following sample SQL statement changes a raw table to a standard table:

   ```
   ALTER TABLE largetab TYPE(STANDARD);
   ```

   Warning:
   It is recommended that you not use nonlogging tables within a transaction where multiple users can modify the data. If you need to use a nonlogging table within a transaction, either set Repeatable Read isolation level or lock the table in exclusive mode to prevent concurrency problems.

   For more information on standard tables, see the previous section, Advantages of Logging Tables.

5. Create indexes on columns most often used in query filters.
6. Create any referential constraints and unique constraints, if needed.

What happens between a client and server when a TCP/IP connection is opened

When a TCP/IP connection is opened, the following information is read on the client side:

- INFORMIXSERVER
- hosts file information (INFORMIXSQLHOSTS, $INFORMIXDIR/etc/sqlhosts, the registry entry on Windows NT) and services file information
- Other environment variables
- Resource files

The following information is read on the server side:

- DBSERVERNAME
- DBSERVERALIASES
- Server environment variables and configuration parameters, including any NETTYPE configuration parameter setting that manage TCP/IP connections

Strategy for estimating the size of the physical log

The size of the physical log depends on two factors: the rate at which transactions generate physical log activity and whether you set the RTO_SERVER_RESTART configuration parameter

The rate at which transactions generate physical log activity can affect checkpoint performance. During checkpoint processing, if the physical log starts getting too full as transactions continue to generate physical log data, the database server blocks transactions to allow the checkpoint to complete and to avoid a physical log overflow.

To avoid transaction blocking, the database server must have enough physical log space to contain all of the transaction activity that occurs during checkpoint processing. Checkpoints are triggered whenever the physical log becomes 75 percent full. When the physical log becomes 75 percent full, checkpoint processing must complete before the remaining 25 percent of the physical log is used. Transaction blocking occurs as soon as the system detects a potential for a physical log overflow, because every active transaction might generate physical log activity.

For example, suppose you have a one gigabyte physical log and 1000 active transactions. 1000 active transactions have the potential to generate approximately 80 megabytes of physical log activity if every transaction is in a critical section simultaneously. When 750 megabytes of the physical log fills, the database server triggers a checkpoint. If the checkpoint has not completed by the time the 920 megabytes of the physical log are used, transaction blocking occurs until the checkpoint completes. If transaction blocking takes place, the server automatically triggers more frequent checkpoints to avoid transaction blocking. You can disable the generation of automatic checkpoints.

The server might also trigger checkpoints if many dirty partitions exist, even if the physical log is not 75 percent full, because flushing the modified partition data to disk requires physical log space. When the server checks if the Physical Log is 75 percent full, the server also checks if the following condition is true:

```
(Physical Log Pages Used + Number of Dirty Partitions) >=
(Physical Log Size * 9) /10)
```

For more information about checkpoint processing and automatic checkpoints, see Checkpoints.

The second factor to consider when estimating the size of the physical log depends on your use of the RTO_SERVER_RESTART configuration parameter to specify a target amount of time for fast recovery. If you are not required to consider fast recovery time, you are not requires to enable the RTO_SERVER_RESTART configuration parameter. If you specify a value for the RTO_SERVER_RESTART configuration parameter, transaction activity generates additional physical log activity.

Typically, this additional physical log activity has little or no affect on transaction performance. The extra logging is used to assist the buffer pool during fast recovery, so that log replay performs optimally. If the physical log is considerably larger than the combined sizes of all buffer pools, page flushing and page faulting occur during fast recovery. The page flushing and page faulting substantially reduce fast recovery performance, and the database server cannot maintain the RTO_SERVER_RESTART policy.

For systems with less that four gigabytes of buffer pool space, the physical log can be sized at 110 percent of the combined size of all the buffer pools. For larger buffer pools, start with four gigabytes of physical log space and then monitor checkpoint activity. If checkpoints occur too frequently and seem to affect performance, increase the physical log size.

A rare condition, called a physical-log overflow, can occur when the database server is configured with a small physical log and has many users. Following the previously described size guidelines helps avoid physical-log overflow. The database server generates performance warnings to the message log whenever it detects suboptimal configurations.

You can use the onstat -g ckp command to display configuration recommendations if a suboptimal configuration is detected.

To change the size and location of the physical log, run the following command after you bring the database server to quiescent or administration mode:

```
onparams -p -s size -d dbspace -y
```

size

The new size of the physical log in KB

dbspace

Specifies the dbspace where the physical log is to be located

The following example changes the size and location of the physical log. The new physical-log size is 400 KB, and the log is located in the **dbspace6** dbspace:

```
onparams -p -s 400 -d dbspace6 -y
```

The onstat -g rea command

Use the **onstat -g rea** option to monitor the number of threads in the ready queue. If the number of threads in the ready queue is growing for a class of virtual processors (for example, the CPU class), you might be required to add more virtual processors to your configuration.

[lchen@ifx01 /home/lchen] $ onstat -g rea

IBM Informix Dynamic Server Version 11.50.UC3W2 -- On-Line -- Up 255 days 13:43:43 -- 2051200 Kbytes

Ready threads:

tid tcb rstcb prty status vp-class name

The onstat -g ioq command

Use the **onstat -g ioq** option to determine whether you must allocate additional virtual processors. The command **onstat -g ioq** displays the length and other statistics about I/O queues. If the length of the I/O queue is growing, I/O requests are accumulating faster
than the AIO virtual processors can process them. If the length of the I/O queue continues to show that I/O requests are accumulating, consider adding AIO virtual processors.

Create Intermdiate data to hold WIPs recycled b3iid record on IPDEV Database ip_arch

1. Create chunks , create dbspace on these new created chunks, create database ip_ardb using schema from archive ip_arch05 on dbspace ip_ardb

```
$ touch /ix_dat2/ix_ardb.0
$ touch /ix_dat2/ix_ardb.0
$ touch /ix_dat2/ix_ardb.0
$ touch /ix_dat2/ix_ardb.0
$ touch /ix_dat2/ix_ardb.0

$ onspaces -c -d ip_ardb -p /ix_dat2/ix_ardb.0 -o 0 -s 1000000
$ onspaces -a ip_ardb -p /ix_dat2/ix_ardb.1 -o 0 -s 1000000
$ onspaces -a ip_ardb -p /ix_dat2/ix_ardb.2 -o 0 -s 1000000
```

```
$ onspaces -a ip_ardb -p /ix_dat2/ix_ardb.3 -o 0 -s 1000000
$ onspaces -a ip_ardb -p /ix_dat2/ix_ardb.4 -o 0 -s 1000000
```

To change dbspace name, disconnect all session and change dbserver's status to quiescent mode, after rename completed, set dbserver online again:

```
$ onmode -u
$ onspaces -ren ip_archdb -n iparchdbs
$ onmode -m
```

modify ip_arch05.sql,

```
CREATE DATABASE ip_arch IN iparchdbs WITH LOG;
CONNECT TO 'ip_arch@systestdb';
```

TIPS: there are remote database server/database definition in this sql script procedure, like ip_0p@ipdb:informix.b3, it will take very long time to try to connect this database if this database cannot be accessed due to network issues or any other definition errors in sqlhosts/services files.

```
[lchen@ifx01 /home/lchen] $ echo "select count(*) from
ip_arch@systestdb:informix.b3"|dbaccess ip_0p

Database selected.



      (count(*))

               0

1 row(s) retrieved.



Database closed.

[lchen@ifx01 /home/lchen] $ echo "insert into ip_arch@systestdb:informix.usport_exit
select * from usport_exit"|dbaccess ip_0p

Database selected.


437 row(s) inserted.



Database closed.

[lchen@ifx01 /home/lchen] $ echo "insert into ip_arch@systestdb:informix.canct_off
select * from canct_off"|dbaccess ip_0p

Database selected.


323 row(s) inserted.



Database closed.
```

```
[lchen@ifx01 /home/lchen] $ echo "insert into ip_arch@systestdb:informix.ctry_code
select * from ctry_code"|dbaccess ip_0p

Database selected.


623 row(s) inserted.



Database closed.

[lchen@ifx01 /home/lchen] $ echo "insert into ip_arch@systestdb:informix.stringtable
select * from stringtable"|dbaccess ip_0p

Database selected.


39 row(s) inserted.



Database closed.

[lchen@ifx01 /home/lchen] $ echo "insert into ip_arch@systestdb:informix.transp_mode
select * from transp_mode"|dbaccess ip_0p

Database selected.


7 row(s) inserted.



Database closed.
```