IBM Security Identity Manager Performance Scripts Documentation

Updated: 2014/06/10

Introduction

This document discusses some of the tuning scripts and files included with the IBM Security Identity Manager Performance and Tuning Guide. Most of the scripts are generic enough to be used with other products that make use of IBM Security Directory Server and/or IBM DB2. Many of the scripts and files include comments and other information at the top of them.

Task Reference

The following sections is a cross reference to files or tools that will help with a specific task.

Enabling system monitoring

IBM Security Directory Server

- ids_auditEnable_v[5|6].ldif
- ids_auditDisable_v[5|6].ldif

IBM DB2

- perftune_enable_monitoring.[bat|sh]

Sun ONE Directory Server

- sunone_accessEnable.ldif
- sunone_accessDisable.ldif

Finding performance bottlenecks

IBM Security Identity Manager Adapter

- perfanalyze_adapter.pl

IBM Security Directory Server

- collectITDSData.[sh|bat]
- perfanalyze_audit.pl
- perfanalyze_indexes.pl
- perfcheck_tunings.pl

IBM DB2

- collectDB2Data.[sh|bat]
- perfanalyze_database.pl
- perfanalyze_dynamicsql.pl
- perfanalyze_eventmonitor.pl
- perfanalyze_tables.pl
- perfcheck_tunings.pl
- perftune_getallsnapshots.sh

Sun ONE Directory Server

- perfanalyze_audit.pl

Oracle

- oracle_dumpDBCfg.sql
- oracle_dumpDBStats.sql
- oracle_dumpSQLStats.sql
- oracle_exportDDL.sql

Tuning system performance

IBM DB2

- perftune_compress_tables.pl
- perftune_reorg.[bat|sh]
- perftune_runstats.[bat|sh]
- dynsqlsumm2advisworkload.pl

Data Collection Scripts

collectDB2Data.[sh|bat]

Product: DB2 (and thus ISDS since it uses DB2)

This script collects a commonly-requested set of data from a DB2 instance that is used by Support to diagnose problems. The script is designed to be lightweight and attempts to sanity-check the data collected and notify the user if something seems to be amiss.

Usage:
collectDB2Data.sh database [USER username USING password]
Options:
database – the name of the database to collect information on
username – user to log into the database as, if necessary
password – the password to log into the datbase with, if necessary

The script should be run as the database owner in a directory with write access. The script will save off database information to a temporary directory. The unix version will try to tar up the results and clean up after itself if possible.

collectITDSData.[sh|bat]

Product: ISDS

This script collects a commonly-requested set of data from an ISDS instance that is used by Support to diagnose problems. The script is designed to be

lightweight and attempts to sanity-check the data collected and notify the user if something seems to be amiss.

Usage:
collectITDSData.sh bindDN bindPassword [tdsEtcDirectory]
Options:
 bindDN – the root DN (usually cn=root)
 bindPassword – the root DN password
 tdsEtcDirectory – the directory containing the ibmslapd.conf file for the instance
 (only required if the script can't determine it for you)

The script should be run as the ISDS instance owner in a directory with write access. The script will save off database information to a temporary directory within the current directory. The unix version will try to tar up the results and clean up after itself if possible.

Note that if it is often useful to have the collectDB2Data.sh output in addition to the collectTDSData.sh output when diagnosing TDS problems.

oracle_dumpDBCfg.sql

Product: Oracle

This script outputs the Oracle database configuration parameters.

Usage:
Log into SQL*Plus as the ISIM User and import the file using:
@oracle_dumpDBCfg.sql

oracle_dumpDBStats.sql

Product: Oracle

This script outputs the Oracle database statistics information such as memory allocation in the SGA.

Usage:
Log into SQL*Plus as the ISIM User and import the file using:
@oracle_dumpDBStats.sql

oracle_dumpSQLStats.sql

Product: Oracle

This script outputs Oracle SQL statistics information in a format that can be parsed with the perfanalyze_dynamicsql.pl script.

Usage:
Log into SQL*Plus as the ISIM User and import the file using:
@oracle_dumpSQLStats.sql

oracle_exportDDL.sql

Product: Oracle

This script outputs a DDL file with all tables, indexes, and views for all tables in the database.

Usage:
Log into SQL*Plus as the ISIM User and import the file using:
@oracle_exportDDL.sql

Data Analysis Scripts

perfcheck_tunings.pl

Product: ISIM and ISAM via ISDS and DB2
Input: output from collectDB2Data.sh and collectITDSData.sh

This script does some high-level checking of common ISDS and DB2 tunings within ISIM and ISAM environments.

Usage (from the -h flag):

```
perfcheck_tunings.pl -i [ inputFile | inputDir ] [ -t dataType ]
Input Options
  -i - file or directory for processing
  -t - type of data file:
       db2 - default
       tds

Other options:
  -h - displays this help information
  -k - keep the temporary extracted data directory instead of removing it
```

For example, assume you have a itimldap_db2Data.tar file. To check for common missed tunings, do:

```
perfcheck_tunings.pl -i itimldap_db2Data.tar
```

You can optionally run it on the extracted directory from the tar file:
perfcheck_tunings.pl -i db2Data/

To check the output of a collectITDSData.sh file, use the -t flag:
perfcheck_tunings.pl -i tdsData.tar -t tds

perfanalyze_adapter.pl

Product: ISIM
Input: ISIM Adapter logs

This script shows adapter response times. It is useful for determining if a bottleneck is with the adapter. Full debug logging should be enabled to obtain worthwhile data.

Usage (from the –h flag):

```
perfanalyze_adapter.pl [ -i inputFile ] [ -c <cutoff> | -r | -g | -p ]
Output options:
  -s - show Request timing summary
  -r - show Request timings on a per-thread basis
  -g - show time gap analysis - where time is spent
  -p - show processed time gap analysis
  -l - processing level
        none - don't try to standardize line data
         min - remove DSML response lines (default)
         mid - normalize common log lines
         max - break out the big guns
  -c - cutoff for processed time gap analysis (default: 5)

Other options:
  -h - displays this help information
  -i - file containing log for processing

If no inputFile is given, the program will read input from STDIN.
```

perfanalyze_audit.pl

Product: ISDS or SunONE Directory Server
Input: ISDS audit.log or SunONE access log

This script is designed to pull five sets of information from either an ISDS audit log or an SunONE Directory Server access log:

- **Filter Timings**
  This report sorts searches by their individual execution time. This is useful for finding out which queries are taking a long time to process and need to be optimized.

- **Filter Distribution Timings**
  This report shows queries grouped into different response-time categories. This is useful for getting an idea of where queries fall across a spectrum of response times.
- **Transaction Summary**
  This report looks at all audit log entries, not just searches, and reports how many were seen and what percentage each operation is of the whole file.
- **Time Interval Distribution**
  This report shows how many queries were executed in the time interval specified (month, day, hour, minute, second).
- **Search Filter Frequencies**
  This report prints out the filters grouped by the frequency they occurred in the file. This is useful for finding which queries are executed the most frequently.

The four reports above can be used with the following three filter settings:

- **All**
  This option does no filter aggregation, each query is analyzed individually.
  Example: (objectclass=inetorgperson)
- **Full**
  This option aggregates identical queries with respect to both the attributes and the attribute values being queried.
  Example: (objectclass=inetorgperson)
- **Fuzzy**
  This option does an aggressive aggregation of queries by ignoring the attribute values being queried and using only the attributes.
  Example: (objectclass=)

By using a combination of the above reports and filter settings it is possible to get a better idea of what kind of transactions and searches the system is performing.

Use this script to determine which LDAP queries are taking the longest, possibly from missing indexes. Use this in conjunction with the perfanalyze_indexes.pl script to locate and index attributes used in long-running LDAP queries.

Usage (from the –h flag):

```
perfanalyze_audit.pl [ -i inputFile ] [ -o outputFile ] [ -f filterMethod ]
  [ -t [ -c cutOff ] | -g | -s | -d | -m timeframe ]
```

Filter options:
  -f - filter method, the following options are valid:
        all - all filters, don't collect similar filters
       fuzzy - use fuzzy filters (ie: no attribute values), default
        full - use full filters

Output options
  -t - show search filter timings
  -d - show search distribution timings
  -s - show transaction summary
  -g - show search filter frequencies
  -m - show time-interval stats, timeFrame is one of: second, minute, hour, day, month
  -c - statements longer than this time are not included in timings report, default is 0.1

Other options:
  -h - displays this help information
  -i - file containing log for processing
  -o - file to put the processed results in, default is STDOUT

If no inputFile is given, the program will read input from STDIN.

Examples:
To see what queries are taking a long time to run regardless of the values in the filter (ie: fuzzy):
perfanlayze_audit.pl –i audit.log –t –f fuzzy

To see what queries are taking a long time to run including the values in the filter (ie: full):
perfanlayze_audit.pl –i audit.log –t –f full

To see the frequency of queries in the log:
perfanlayze_audit.pl –i audit.log –g

To get a time interval report listing how many queries were excuted each minute:
perfanlayze_audit.pl –i audit.log –m minute

To pull all reports from an audit log:
perfanlayze_audit.pl –i audit.log –f –d –s –g –m second

perfanalyze_audit_grep.pl

Product: ISDS
Input: ISDS audit.log

This script acts much like the 'grep' command, except it returns a whole stanza instead of just one line. It is useful for pulling out specific entries that are then ran through perfanalyze_audit.pl for more detailed analysis.

Usage (from the –h flag):
perfanalyze_audit_grep.pl [-i inputFile] [-o outputFile] [-h] [-v | -n] searchString

Search hints:
 * Bind DN: search on the bind DN using: "bindDN: <DN>"
 * Operations: search on operations using one of the following:
   Search, Bind, Unbind, Add, Modify, Delete
 * Client IP: search on the client IP by using: "client: <ip:port>"
 * Connection: search on the connection ID by using: "connectionID: <num>"
 * Time: search on a time in the format: 2006-06-14-04:12:24.952-06:00DST
   Hint: to limit times to completed times, prefix with "--":
     --2006-06-14-04:12:24.952-06:00DST
   Hint: to limit to received times, prefix with "received:":
     received: 2006-06-14-04:12:24.952-06:00DST
 * Status: search on result status using one of the following:
   Success, No such object

Search options:
 -n - case insensitive search
 -v - negative search, find stanzas that do not match the criteria

Other options:
 -h - displays this help information
 -i - file containing log for processing
 -o - file to put the processed results in, default is STDOUT

If no inputFile is given, the program will read input from STDIN.

Examples:
To find all stanzas that were not bound as cn=root:
perfanlayze_audit_grep.pl –i audit.log -v cn=root

To find stanzas coming from a specific IP (say 192.168.1.1):
perfanlayze_audit_grep.pl –i audit.log 192.168.1.1

perfanalyze_audit_replay.pl

Product: ISDS
Input: ISDS audit.log

This script will take an ISDS audit log and replay the searches by either contacting the server directly or creating a script with ldapsearches. This is useful for validating tunings done to the database since the first audit log was captured.

It is suggested that you run the audit log through some of the other analyze scripts prior to replay to ensure that queries will not negatively impact the performance of the box as the script does not know (or care) how slow the query is to execute, how much server resources are tied up with a query, or how many results will be read back from the query.

Queries can either be run either anonymously, by a specific user, or by the original user. Binds will be done anonymously unless a username and password are specified (-D and -w) or a bindFile (-b) is given. Specifying a specific username and password will cause all queries to use that username and password regardless of the original caller. If you want the original caller to replay the query, create a bindFile that contains the usernames and passwords of the original callers. The file should be in the following format:

```
cn=user1:password1
cn=user2:password2
```

The script will try to replay the queries directly against the server unless a script file (-s) is specified.

Usage (from the –h flag):

```
perfanalyze_audit_replay.pl  [ -r remoteHost | -p portNum | -s scriptFile ] [ -i inputFile ]
  [ -o outputFile ] [ -b bindFile | -D username | -w password ]
Binding options:
  Unless specified, the search will be performed anonymously. Use -D/-w to
  set a default username to bind as. If you pass a bindFile in with -b it
  will try to use those passwords when it finds matching bind IDs in the file.
  -b - file with username:password pairs for binding
  -D - bind username
  -w - bind password
  -r - remote host to run against directly (or host for ldapsearch string)
     Default: localhost
  -p - port to connect to (or port for ldapsearch string)
     Default: 389

Output options:
  -s - file to output ldapsearch script in

Other options:
  -i - file containing dynamic sql statements or audit log for processing
  -o - file to put the processed results in, default is STDOUT

If no arguments are given, the program will read input from STDIN.

Specifying the -s will create a file instead of contacting the server
directly.
```

Examples:

To create a replay script against a specific server and port:

```
perfanlayze_audit_replay.pl –i audit.log –r ldapserver –p 389 –s relayScript.sh
```

To replay queries against a server directly:

```
perfanlayze_audit_replay.pl –i audit.log –r ldapserver
```

perfanalyze_audit_repldelay.pl

Product: ISDS
Input: ISDS audit.log

This script takes two ISDS audit logs from either a master/replica or peer-master configuration and attempts to determine the replication delay between the two machines.

By default the script assumes that the times on the two machines are exact (which is fairly safe to assume if they are both running ntpd). If the machine times are not in sync, use the -t option to pass in the time difference in seconds. In addition, the script assumes that both audit logs start with close to the same transaction, that is the first update entry in the master log matches one of the first replicated entries in the replica log. If this is not the case the timing data reported maybe highly inaccurate. You may try trimming the top of the audit logs such that this criteria holds true to avoid this problem.

Usage (from the –h flag):

```
perfanalyze_audit_repldelay.pl [ -o outputFile ] [ -t timeSkew ] [ -u ] [ -h ] -m inputFile1 -r inputFile2
Files:
  -m - first ITDS audit log (generally the master)
  -r - second ITDS audit log (generally the replica)

Processing options:
  -b - processing block size lower number will use less memory (default: 100)
  -t - time skew between the two machines in seconds (default: 0)
      Note: can include fractions of a second: 1.459

Display options:
  -u - show all unmatched DNs

Other options:
  -h - displays this help information
  -o - file to put the processed results in, default is STDOUT
```

Examples:
Basic run with the time of the two servers in sync:

```
perfanlayze_audit_repldelay.pl -m master_audit.log -r replica_audit.log
```

Run where the two servers are 5 seconds apart:

```
perfanlayze_audit_repldelay.pl -t 5 -m master_audit.log -r replica_audit.log
```

perfanalyze_audit_replrequirement.pl

Product: ISDS
Input: ISDS audit.log

This script takes a single ISDS audit log and attempts to see how tight replication would need to be for the same traffic to work spread across multiple peer-masters. Consider an application that does an update and within 2 seconds reads the same record back. For this traffic to work load-balanced across peer-master machines, the replication delay would need to be less than 2 seconds to avoid reading stale data if the update and search went to different nodes.

This script looks for updates with subsequent reads or modifications (adds, updates, deletes) on the same objects and prints out the time difference between the two activities.

Usage (from the –h flag):

```
perfanalyze_audit_replrequirement.pl [ -i inputFile ] [ -o outputFile ] [ -h | -s scope ]
Output options:
  -s - scope resolution, determines if the consistency check is done
       on base, one, and/or subtree searches. scope can have the following
       values:
        base - most reliable (default)
        one  - includes base, next reliable
        sub  - includes one and base, least reliable

Other options:
  -h - displays this help information
  -i - file containing log for processing
  -o - file to put the processed results in, default is STDOUT

If no inputFile is given, the program will read input from STDIN.
```

Example:

```
perfanlayze_audit_replrequirement.pl –i audit.log
```

perfanalyze_database.pl

Product: DB2 (and thus ISDS since it uses DB2)
Input: DB2 database and/or bufferpool snapshot

This script will take a database and/or bufferpool snapshot and report the following information:

- Bufferpools:
  - Hit ratio
  - Sync to async write ratio
  - Number of direct reads and writes
  - Number of database files closed
- Database (only available from a database snapshot):
  - Connections

- Transactions
- Sorts:
  - Total
  - Average sort time
- Locks:
  - Current number
  - Number of deadlocks
  - Number of escalations

The script then tries to make some generic recommendations about possible tuning parameters based on some of these values. That said, the primary goal of the script is to present processed information to the user, not as a tuning suggestion tool.

Usage (from the –h flag):

```
perfanalyze_database.pl [ -i inputFile | [ -d databaseName | -s ] ] [ -o outputFile ]
  -i - file containing snapshot for processing
  -d - database name to get snapshot from directly
  -s - if given the temporary file with the snapshot will be saved
  -o - file to put the processed results in, default is STDOUT

If no arguments are given, the program will read input from STDIN.
```

Example:
To get a list of queries sorted by execution time:

```
perfanlayze_database.pl –i idsdb2.snapshot
```

perfanalyze_dynamicsql.pl

Product: DB2 (and thus ISDS since it uses DB2)
Input: DB2 dynamic sql snapshot

This script is designed to locate slow queries from a dynamic sql database snapshot. The dynamic sql snapshot shows how long a query takes to execute in aggregate form. This is less meaningful from a response-time perspective than how long each individual query took to execute. This script provides a list of queries sorted by the execution time and the number of times those queries were executed. Long-running queries that are seen very infrequently are generally less of a problem than longish queries that are seen frequently.

Use this script to determine which LDAP queries are taking the longest, possibly from missing indexes. Use this in conjunction with the perfanalyze_indexes.pl script to locate and index attributes used in long-running LDAP queries.

Because SQL queries can be long, the printed report is truncated for the screen. To view the entire query (or to change the length of the SQL reported), use the –t option. A zero –t value will result in the query being printed in full.

By default only the queries taking longer than 0.1 seconds will be shown. To change the cut-off value, use the –c option. A zero –c value will result in all queries being printed.

If the database server is on unix machine and the script is ran from there it can pull the dynamic sql snapshot itself using the –d flag. If you want to retain this pulled snapshot, use the –s flag as well.

Usage (from the –h flag):

```
perfanalyze_dynamicsql.pl [ -i inputFile | [ -d databaseName | -s ] ] [ -o outputFile ]
  [ -c cutOff | -t truncLine ] [ -w | -n ]
Output options:
  -t - length to truncate statement at, default is 90 characters. 0 = don't truncate.
  -c - time cutoff; statements longer than this time are not included, default is 0.1
  -r - column to sort by, default is secPerExec
  -n - include queries that have no statistics information
  -w - include the number of rows written

Other options:
  -i - file containing dynamic sql statements for processing
  -d - database name to get dynamic sql statements from directly
  -s - if given the temporary file with the dynamic sql will be saved
  -o - file to put the processed results in, default is STDOUT


If no arguments are given, the program will read input from STDIN.
```

Examples:

To get a list of queries sorted by execution time:

```
perfanlayze_dynamicsql.pl –i idsdb2.snapshot
```

To view all queries regardless of execution time:

```
perfanlayze_dynamicsql.pl –i idsdb2.snapshot –c0
```

To view the full SQL for queries taking longer than 0.01 seconds:

```
perfanlayze_dynamicsql.pl –i idsdb2.snapshot –c0.01 –t0
```

perfanalyze_eventmonitor.pl

Product: DB2 (and thus ISDS since it uses DB2)
Input: DB2 event monitor output processed by db2evfmt

Much like the perfanalyze_dyanmicsql.pl script, this script reads in a DB2 event monitor file and prints a list of queries, how long they took to execute, and how many times they were executed.

Usage (from the –h flag):

```
perfanalyze_eventmonitor.pl [ -i inputFile ] [ -o outputFile ] [ -c cutOff | -t truncLine ] [ -w | -n ]
Filter options:
  -f - filter method, the following options are valid:
       fuzzy - use fuzzy filters (ie: no attribute values), default
        full - use full filters

Output options:
  -t - length to truncate statement at, default is 80 characters. 0 = don't truncate.
  -c - time cutoff; statements longer than this time are not included, default is 0.1
  -r - column to sort by, default is secPerExec
  -n - include queries that have no statistics information
  -w - include the number of rows written

Other options:
  -i - file containing dynamic sql statements for processing
  -o - file to put the processed results in, default is STDOUT


If no arguments are given, the program will read input from STDIN.
```

perfanalyze_indexes.pl

Product: ISDS
Input: ISDS audit.log or DB2 dynamic sql snapshot

This script is designed to determine if searches against ISDS are using filters against attributes that do not have indexes. It does this by the following:

- Locating the LDAP schema files by reading them from the ibmslapd.conf file, looking for the default schema files from a specified directory, reading the schema files to process from the command line, reading the schema directly from the LDAP server itself, or a combination of the above.
- Processing the LDAP schema to read all attributes and determine those that have indexes.
- Processing either an ISDS audit log or a dynamic sql snapshot from the ISDS database instance to determine all attributes being used in searches. Alternatively the script will accept a file containing a list of attributes, one per line.
- Comparing the searched attributes against the indexed attributes.
- Printing a report of the attributes search on and their index status.

The easiest way for the script to process the schema is to have it read it directly from the server. This requires the Net::LDAP perl module. If this module is not available you can use one of the other methods. If you are unable to have the schema read from the server, it is easiest to run the script from the ISDS server itself and pass it the fully qualified path to the ibmslapd.conf file. If this is not possible, or not desirable, the user can specify which directory on the local machine contains the schema files.

If attributes are encountered in the input but were not found in the LDAP schema (usually encountered when the audit log doesn't match the server it is being compared to or if the script was unable to find all ISDS schema files) the report will mention this fact and will be unable to determine the index status of these attributes.

Not every searched attribute needs an index. Indexes create additional work for DB2 when inserting or modifying entries with indexed attributes. Attributes that are searched on very rarely may not benefit from being indexed given the overhead required to maintain them. The script attempts to include the number of times the attribute was searched on if available to aid in the evaluation if an index is necessary.

To facilitate adding missing indexes, the script can optionally create an LDIF which can be used to add the indexes directly. The resulting LDIF should be viewed to confirm you really want to create indexes for all the specified attributes before importing it.

Usage (from the –h flag):

```
perfanalyze_indexes.pl [ -c confFile | -d schemaDir | -s schemaFiles | -r remoteHost
  [ -u username | -p password ] ] [ -i inputFile ] [ -o outputFile ] [ -l ldifFile ]
  [ -a ]
Schema options:
  -c - fully qualified path name to ibmslapd.conf file
  -d - directory where schema files are located
  -s - colon separated list of schema files (no spaces)
  -r - remote host to pull schema from
  -u - username (if not specified, bind will be anonymous)
  -p – password

Output options:
  -a - print all attributes searched on, not just unindexed ones
  -l - file to export an LDIF file for use in indexing any unindexed attributes

Other options:
  -i - file containing dynamic sql statements or audit log for processing
  -o - file to put the processed results in, default is STDOUT

Note: use unix-style slashes for all files and directories, even if on Windows
If no arguments are given, the program will read input from STDIN.
```

Examples:
To analyze indexes from an audit log in the current directory based on the schema pulled from the ibmslapd.conf file on a Windows platform. The –a option will show all attributes, not just unindexed ones:
perfanalayze_indexes.pl –i audit.log –c "c:/program files/ibm/ldap/etc/ibmslapd.conf" –a

To analyze indexes from a dynamic sql snapshot with the default schema files in a given directory but include a custom file as well:
perfanalayze_indexes.pl –i idsdb2.snapshot –d /usr/ldap/etc –s /usr/ldap/etc/custom.schema

To analyze indexes from an audit log based on a schema from a remote machine:
perfanalayze_indexes.pl –i audit.log –r ldapserver -u cn=root -p password

To create an LDIF to assist in indexing unindexed attributes:
perfanalayze_indexes.pl –i audit.log –r ldapserver –l idsIndexes.ldif

If you have a list of attributes you'd like to create indexes for, the script can generate an LDIF to index them. Create a file that includes the attributes you'd like to index, one per line. For instance, the file attributeList.txt may contain:

```
mail
st
roomnumber
```

Then run it through the script and have it create an LDIF:
perfanalayze_indexes.pl –i attributeList.txt –r ldapserver -u cn=root -p password
–l idsIndexes.ldif

perfanalyze_queries.pl

Product: DB2 (and thus ISDS since it uses DB2)
Input: DB2 event monitor output processed by db2evfmt

This is a script to take either multiple perfanalyze_dyanmicsql.pl outputs or simply a list of SQL statements and print a list of how many times the SQL was executed. This script is most useful when you have SQL statements in a format that none of the other scripts will read and yet you'd like to know how frequently queries are executed.

Usage (from the –h flag):
perfanalyze_queries.pl [-i inputFile] [-o outputFile] [-r <string>] [-t <num>]
-i - file containing dynamic sql statements for processing
-o - file to put the processed results in, default is STDOUT
-r - column to sort by, default is secPerExec
-t - length to truncate statement at, default is 80 characters. 0 = don't truncate.

If no arguments are given, the program will read input from STDIN.

perfanalyze_tables.pl

Product: DB2 (and thus ISDS since it uses DB2)
Input: DB2 table snapshot

This script will process a DB2 table snapshot and print the number of rows read from and written to a table, sorted by rows read. This can be useful for finding tables that are being tablescaned.

Note: Enabling the table monitor (DFT_MON_TABLE) can be a performance hit. It is not recommended that you leave this monitor switch enabled on production systems.

Usage (from the –h flag):

```
perfanalyze_tables.pl [ -i inputFile | [ -d databaseName | -s ] ] [ -o outputFile ] [ -c cutOff | -r sortColumn ]
Output options:
  -r - column to sort by, default is rowsRead
  -c - tables where number(sortColumn) < cutOff are not included, default is 100

Other options:
  -i - file containing snapshot for processing
  -d - database name to get snapshot from database directly
  -s - if given the temporary file with the snapshot will be saved
  -o - file to put the processed results in, default is STDOUT

If no arguments are given, the program will read input from STDIN.
```

perfanalyze_workflow.pl

Product: ISIM
Input: ISIM workflow XML

This script takes an ISIM workflow XML blob and shows all possible paths through the workflow. The script can either query the ISIM LDAP for a workflow (if the Net::LDAP perl module is installed) or one can be specified on the command line. The script can provide a list of workflows if one was not provided but LDAP connection information was given.

If the Easy::Graph perl module is installed the script can provide a graph showing the flow between the different nodes.

Usage (from the –h flag):

perfanalyze_workflow.pl [-a][-g] [-i inputFile | [-r host [-u username -p password] -b itimBase][-f filter]] [-o outputFile]
LDAP options:
 -r - remote LDAP server to query
 -b - ITIM base (eg: ou=acme,dc=com)
 -u - username (if not specified, bind will be anonymous)
 -p - password
 -f - erGlobalID or workflow name to analyze
    (if blank, will list all workflows)

Output options:
 -g - print out an ASCII graph (requires Graph::Easy)
 -a - print out all possible workflow paths

Other options:
 -i - file containing the workflow for processing
 -o - file to put the processed results in, default is STDOUT

Note: use unix-style slashes for all files and directories, even if on Windows

dynsqlsumm2advisworkload.pl

Product: DB2 (and thus ISDS since it uses DB2)
Input: output from perfanalyze_dynamicsql.pl

This script generates a workload file that can be used by db2advis. The purpose being to create a workload file that DB2 can use to suggest indexes. The workload file created includes frequency statements to give DB2 more data about your actual usage patterns.

Usage:
perfanalyze_dynamicsql.pl -i snapshot | dynsumm2advisworkload.pl > workload.sql

explainSQL.[bat|sh]

Product: DB2 (and thus ISDS since it uses DB2)
Input: SQL statement

This script generates a DB2 explain plan for a SQL statement. To use the script, place the SQL statement to explain into a separate file all on one line without a trailing semicolon (;). These files commonly have an .sql extension for clarity although this is not required. Run the script with the SQL file as the only argument – this will create an output file with the same name as the input file with an .exfmt extension.

Generating an explain plan requires that the explain tables exist before hand. Trying to generate an explain without these tables will result in an error. The

explain tables can be created by importing the EXPLAIN.DDL file located in the sqllib/misc directory for your install. If you cannot locate these files, the script package includes versions for DB2 v7 and DB2 v8.

Unix usage:
explainSQL.sh [DATABASE] [SCHEMA] query.sql

Windows usage:
Edit explainSQL.bat in a text editor such as notepad and set the DATABASE variable appropriately.
explainSQL.bat query.sql

do_statement_monitoring.sh

Product: DB2 (and thus ISDS since it uses DB2)
Input: statement monitor

This script will collect information about all the DB2 statements being executed for a short period of time while a workload is run. It will need to be customized for the database that is being monitored, by editing the 14th line:
"db2 connect to itim"
to reflect the correct database name.

This is similar to the perfanalyze_dynamicsql.sh script, but while that script reports statistical information for all statements that have been run prior to its execution, this script only collect information on statements that are run while the script is running.

After it is started, it will do any necessary preparation and then prompt with:
"Do experiment to be monitored now. Press enter to stop monitoring."
At this time you would execute your workload, then press enter when it completes.

The script will generate three files:
dstate.out - all information about each statement that was executed in order of execution
mon.out - summarized information, with one line per statement in order of execution
mon.sorted - summarized information, sorted with the longest running statements at the end

The summarized information for each statement includes the time, execution time, number of rows read, and the statement text.

Usage:
do_statement_monitoring.sh

Tuning Scripts

perftune_compress_tables.pl

Product: IBM DB2 under IBM Security Directory Server and IBM Security Identity Manager

This script automates the steps required to enable row-level compression for DB2 tables on DB2 V9 or later. It is designed specifically for the tables under ISDS and ISIM. Enabling row-level compression involves setting the 'compress' attribute for the table via an 'alter table' command and then reorganizing the table.

Enabling table compression should be done in a maintenance window when any application accessing the database is down due to the table reorgs. The reorgs can take hours or more depending on how much data is in the system.

Usage:

```
perftune_compress_tables.sh DBTYPE [DATABASE [SCHEMA]]
    DBTYPE = database type: ITDS or ITIM
  DATABASE = database to run against (optional - default ldapdb2)
    SCHEMA = schema to run against (optional - default LDAPDB2)
```

perftune_enable_monitoring.[bat|sh]

Product: IBM DB2

This script simply enables the DB2 monitoring flags. The flags are required for pulling useful database snapshots and using several of the DB2-based perfanalyze_* scripts. After enabling the monitors the database should be stopped and restarted.

Unix usage:

```
su - dbinstance_owner
perftune_enablemonitoring.sh
db2stop
db2start
```

Windows usage:

```
db2cmd
perftune_enablemonitoring.bat
db2stop
db2start
```

perftune_getallsnapshots.sh

Product: IBM DB2

This script simply pulls a composite database snapshot. The resulting snapshot can be used as input for perfanalyze_[database|dynamicsql|indexes|tables].pl.

Unix:
```
su - dbinstance_owner
perfanalyze_getallsnapshots.sh DBNAME
```

perftune_queues_*

Product: IBM WebSphere MQ under IBM Security Identity Manager 4.5.1 and 4.6

These scripts are used to help automate clearing or getting information on the MQ queues used by IBM WebSphere V5.x. These scripts will not work and should not be used for IBM WebSphere V6.x JMS queues.

perftune_reorg.[bat|sh]

Product: IBM DB2

These scripts are designed to run a DB2 reorg for all tables in a database for a specific schema. This is often useful for test systems when a lot of data has been added and then removed from the database. This script should be ran when the database is not in use.

Unix usage:
```
perftune_reorg.sh [DATABASE [SCHEMA [TABLES]]]
   DATABASE = database to run against (optional - default ldapdb2)
    SCHEMA = schema to run against (optional - default LDAPDB2)
    TABLES = list of tables to run against, if none are given, all
         tables are processed
```

Windows usage:
Edit perftune_reorg.bat using a text editor such as notepad and set the DATABASE and SCHEMA variables before running the script.

perftune_runstats.[bat|sh]

Product: IBM DB2

This scripts will execute runstats on all tables in a specific database. It auto-detects the version of DB2 being used and attempts to auto-detect the type of database being used (ISIM vs ISDS) and use the correct runstats options accordingly. In addition, it does some manual cardinality tunings based on the type of database that was auto-detected.

Unix usage:

```
perftune_runstats.sh [DATABASE [SCHEMA [TABLES]]]
   DATABASE = database to run against (optional - default ldapdb2)
    SCHEMA = schema to run against (optional - default LDAPDB2)
    TABLES = list of tables to run against, if none are given, all
         tables are processed
```

Windows usage:
Edit perftune_runstats.bat using a text editor such as notepad and set the DATABASE, SCHEMA, and SCHEMATYPE variables before running the script.

Support Files

DB2v[7|8|9]-EXPLAIN.DDL

These are copies of the EXPLAIN.DDL file that shipped with DB2 v7, v8, and v9 that is used to create the explain tables. The explain tables are necessary to generate explain plans. It is recommended that you use the copy of the file that came with your version of DB2 if it is available (see the sqllib/misc/ directory on Unix systems and SQLLIB\misc\ on Windows systems). Instructions for how to load the files are included in the top of the files themselves.

ids_auditEnable_v[5|6].ldif

This LDIF will enable auditing of search filters for ISDS. Two variants of the file are included, one for server version 5.x and one for server versions 6.x.

The file contains comments for how to enable auditing of other areas such as binds, unbinds, adds, modifies, and deletes as well.

ids_auditDisable_v[5|6].ldif

This LDIF will disable all auditing for ISDS. Two variants of the file are included, one for server version 5.x and one for server versions 6.x.

sunone_accessEnable.ldif

This LDIF will enable access logging for SunONE Directory Server at level 256.

sunone_accessDisable.ldif

This LDIF will disable access logging for SunONE Directory Server.