NAME

`mrskew` – create skew profiles for Oracle SQL trace files

SYNOPSIS

`mrskew` [`--alldepths`] [`--commas`] [`--csv`] [`--dashes`] [`--debug=`*n*] [`--depmin=`*n*] [`--foot`] [`--format=`*string*] [`--group=`*expr* | `--g=`*expr*] [`--group-label=`*expr* | `--gl=`*expr*] [`--group-width=`*n*] [`--head`] [`--help` | `-?`] [`--histogram`] [`--init=`*string*] [`--initrc`] [`--key`] [`--listrc`] [`--man`] [`--name=`*pattern*] [`--pfact=`*integer*] [`--pform=`*string*] [`--plabel=`*string* | `--pl=`*string*] [`--precision=`*n* | `--pre=`*n*] [`--rc=`*file*] [`--scanmax=`*n*] [`--select=`*numexpr* | `--s=`*numexpr*] [`--select-label=`*expr* | `--sl=`*expr*] [`--separator=`*string* | `--sep=`*string*] [`--sort=`*o1,o2,...,oN* | `--sort=n` | `--sort=no` | `--sort=none`] [`--thinktime=`*float* | `--z=`*float*] [`--top=`*string*] [`--verbose` | `--verbose=`*level*] [`--version`] [`--where=`*expr* | `--w=`*expr* | `--where0=`*expr* | `--w0=`*expr*] [`--where1=`*expr* | `--w1=`*expr*] [*file* ...]

DESCRIPTION

Oracle extended SQL trace files contain performance data, arranged so that a single line of trace data describes a single database call ("dbcall") or a single operating system call ("syscall" or "oscall"). Dbcalls are reported upon by trace lines beginning with tokens like `PARSE`, `EXEC`, or `FETCH`. Syscalls are reported upon by trace lines beginning with the token `WAIT`. The collection of dbcalls and syscalls taken together are called Oracle *calls*.

`mrskew` prints a report revealing skew information for Oracle calls (dbcalls or oscalls) that you can specify with a `--name` pattern. You may group your results with a `--group` expression, and you may filter your results with a `--where` expression. The first column of `mrskew` output is the grouping column. The second column is, by default, the duration column, which reports upon the accounted-for duration of the specified calls. However, you may select a different result for the second column with a `--select` numeric expression.

If no input *file* is specified, `mrskew` takes its input from the standard input device. If more than one *file* is named on the command line, `mrskew` concatenates the input files. This permits you to analyze data skew across multiple trace files in a single step.

When `mrskew` takes its input from a non-regular file (e.g., a pipe), it will not be able to calculate the `$depmin` value. To compensate for this deficiency (a speed-vs-accuracy trade-off), you can use the `--depmin` option to specify the values for `mrskew` to use. The default is `--depmin=0`.

OPTIONS

Some of the options require careful quoting on the command line. For more information, see OS Dependencies. Examples in this manual page use Unix-style quoting syntax.

`--alldepths` Shorthand for `--where1=1`, which matches calls at all recursive depths.

`--commas` Print commas in numerical output. Use `--nocommas` to supress commas. The default is `--commas`.

`--csv` Use comma-separated value output format. Using `--csv` is shorthand for `--nocommas --nodashes --format=csv --pfact=1 --pform=%.6f plabel=PCT separator=,`. If you wish to use different values for these options, then specify those options separately on the command line (or in a `--rc` file) after the `--csv` option. For example, `--csv --dashes` will do the expected thing, but listing the options in the reverse order would cause the `--csv` option to override the `--dashes` option.

`--dashes` Print dashes between the header and the body, and between the body and the footer. Use `--nodashes` to suppress the dashes. The default is `--dashes`. Note that using `--csv` automatically sets`--nodashes`.

`--debug=`*`n`* Print extra diagnostic information. Higher *n* values result in more information. The default value is `--debug=0`. The level is optional. If `--debug` is given (without a level), then the level is 1. Use `--debug=1` to see a trace of `.rc` file processing.

`--depmin=`*`n`* Set the `$depmin` variable value to *n*. The default behavior is that `mrskew` will calculate the `$depmin` value for your file. See `--scanmax` for details.

`--foot` Print a total at the end of the report. Use `--nofoot` to suppress the total. The default value is `--foot`. The parenthetical number in the footer label (e.g., `TOTAL (42)`) is the total number of rows (both displayed and aggregated with `%d other`) in the report.

`--format=`*`string`* Set output format type defined by *string*. Valid values for *string* are `tab` (tabular) or `csv` (comma-separated values). The default value is `--format=tab`.

`--group=`*`expr`*, `--g=`*`expr`* Group data by *expr*, which can be any valid Perl expression except for the empty (zero-length) string. The expression can refer to any variable described in Variables and Functions below, and it can use any Perl operators or functions. For example, the option `--group='"$file::$line"'` will direct `mrskew` to group output by distinct values of the concatenation of the file name, the string `::`, and the line number within that file. You could accomplish the same grouping with `--group='$file."::".$line'` or even

`--group='join("::",$file,$line)'`, since you can use any Perl expression (including function calls) within *expr*.

The default is `--group='$nam'`, which groups calls by their names.

`--group-label=`*`expr`*, `--gl=`*`expr`* Print the value of *expr* (which can be any valid Perl expression except for the empty string) as the label on the *group* column. The default value is `CALL-NAME` if `--group='$nam'` (the default `--group` value). Otherwise, the default value is *expr* from `--group=`*expr*, expressed as a string literal.

`--group-width=`*`n`*, `--gw=`*`n`* Set width of *group* column (the first column in the output) to no more than *n* characters. The default is `--group-width=0`, which means not to limit the width. `mrskew` will use a large enough value of *n* to render the complete *group* column header label (see `--group-label` and `--head`) and the "TOTAL" label (see `--foot`), even if you specify a value of *n* that is too small to render them. It will limit your elision to at the smallest, the width of "a…", which is the smallest elided value that it makes any sense to render.

`--head` Print a header at the beginning of the report. Use `--nohead` to suppress the header. The default is `--head`.

`--help`, `-?` Print usage information and exit.

`--histogram` Print call duration histogram data with each group value. Use `--nohistogram` if want to see each grouping value but no performance data associated with it. The default is `--histogram`.

`--init=`*`string`* Execute *string*, which must contain syntactically correct Perl code, before beginning file processing. The default is `--init=''`.

`--initrc` Use `--noinitrc` to ignore the default .rc files in your home directory and current working directory. The default is `--initrc`. Use `--noinitrc` if you don't want to run the default `.rc` files.

`--key` Print license key information and exit.

`--listrc` List the full pathnames of the `.rc` file names that would be used by this execution and exit. The default is `--nolistrc`. The options `--listrc --rc=my.rc` will show you where your `my.rc` file is being found.

`--man` Print the manual page and exit.

`--name=`*`pattern`* Choose Oracle timed events whose names match *pattern*. The *pattern* is a Perl regular expression used to match the dbcalls and syscalls named in your trace data. The matching is done case insensitively. For example:

`--name=parse` matches `PARSE` dbcall, etc.

`--name=read` matches `LOBREAD` dbcall, `db file scattered read` syscall, etc.

`--name='(read|write)'` matches `LOBREAD` dbcall, `db file scattered read` syscall, `direct path write temp` syscall, etc.

`--name='db.*read'` matches `db file scattered read` syscall, `db file sequential read` syscall, etc.

You may use the following special patterns:

`:all` matches any trace file line.

`:call` matches any dbcall or syscall.

`:dbcall` matches any dbcall.

`:syscall` **or** `:oscall` matches any syscall.

If you wish to express a literal character that is a regular expression operator such as the characters in the set `[()*+^$.]`, you must escape it by preceding it with a backslash. For example, to specify a pattern matching only the string `SQL*Net message to client`, you would use `--name='SQL\*Net message to client'`.

Note that the only pattern that allows you to match non-call lines is the special pattern `:all`. See Examples for a use case.

The default is `--name='.+'`, which will match any timed event name.

To understand how the `--name` option interacts with the `--where` option, see the description of `--where` below.

`--pfact=`***integer*** Multiply values in the *percentage* column by *integer*. The default value is `100`. Use `--pfact=1 --pform=%.6f --plabel=P` to express proportions as unitless floating point numbers.

`--pform=`***string*** Format proportions in the *percentage* column as *string*. The *string* value is the format string of a Perl `sprintf` function. The default value is `%.1f%%`. Use `--pfact=1 --pform=%.6f --plabel=P` to express proportions as unitless floating point numbers.

`--plabel=`***string*****,** `--pl=`***string*** Print *string* as the label on the *percentage* column (the third column in the output). The default value is `%`.

`--precision=`***n*****,** `--pre=`***n*** Round *select*, MEAN, MIN, and MAX values to *n* digits to the right of the decimal point. The default value is `6`.

`--rc=`***file*** Process lines in *file* as if they were listed on the command line. See .RC FILES for more information.

`--scanmax=`***n*** Read *n* lines looking for Oracle trace file meta data information, such as the start time of the task represented within the file and the minimum recursive depth. The default value is `250`. Using `0` means never give up.

--select=*numexpr*, --s=*numexpr* Select the specified numeric expression for the value of the second column of `mrskew` output. The default is `--select='$af'`, which produces a histogram of accounted-for durations (see Variables and Functions) for specified calls.

--separator=*string*, --sep=*string* Use *string* as the output column separator. The default is `--separator=' '` (two spaces).

--select-label=*expr*, --sl=*expr* Print the value of *expr* (which can be any valid Perl expression except for the empty string) as the label on the *select* column. The default value is `DURATION` if `--select='$af'` (the default `--select` value). Otherwise, the default value is *numexpr* from `--select=`*numexpr*, expressed as a string literal.

--sort=$s_1, s_2, ..., s_N$, --sort=n, --sort=no, --sort=none Sort by the columns denoted by strings $s_1, s_2, ..., s_N$. The first string, s_1, defines the primary sort order, s_2 defines the secondary sort order, and so on. Each s_i is a string matching the regular expression `/[1-7][ns]?[ad]?/`:

`[1-7]` The first character is a numeral denoting which column to order by.

`[ns]?` The second character is an optional `n` (numeric) or `s` (string), specifying the sort semantics. If neither an `n` nor `s` is specified, `n` is used.

`[ad]?` The third character is an optional `a` (ascending) or `d` (descending), specifying the sort order. If neither an `a` nor `d` is specified, `d` is used.

The default is `--sort=2nd,4nd,1sa`.

Using the `--sort` value of `n`, `no`, or `none` will result in no sort being performed, which saves time and CPU consumption, but may make it difficult to find what you're looking for, especially if you're using `--top`.

--thinktime=*float*, --z=*float* Set the think-time threshold for `$island_id` calculation to *float* seconds. For example, the default `--thinktime=1` means to regard any `SQL*Net message from client` call whose duration is 1.0 seconds or more to be an "ocean" of think time, which then defines "islands" of activity prior and subsequent to the `SQL*Net message from client` call.

--top=*n*, --top=all, --top=none Print only the first n rows in the output data, and then summarize the remaining entries in a single row. The default value is `10`. To print all rows, use `--top=0` or `--top=all`. To print no rows, use `--top=none`.

--verbose, --verbose=*level* Print information about option values, examined files, and matched call names. The default is `--verbose=1`, which is the same as using `--verbose` with no argument. Using `--verbose=2` yields more information, and `--verbose=0` produces compact table-only output. Note that if a file you're trying to analyze doesn't show up in the examined

files list, it is because either `mrskew` couldn't open it, or `mrskew` couldn't determine what Oracle version created it.

`--version` Print the version number and exit.

`--where=`***expr***`, --w=`***expr***`, --where0=`***expr***`, --w0=`***expr*** Filter output data by *expr*; that is, include in the output only information from raw trace lines for which *expr* is true. An *expr* is any valid Perl expression. The expression can refer to any variable described in Variables and Functions below, and it can use any Perl operators or functions. For example, the setting `--where='$p3 gt 1 or $p1 == 48'` will direct `mrskew` to include in the output only values for which the Oracle `p3` field value is greater than 1 or for which the Oracle `p1` field value equals 48.

The default is `--where=1`. This default, coupled with the default `--where1='$dep==$depmin'`, sensibly matches the `--name='.+' --group='$name'` defaults, which together create an accurate profile for the time accounted for within the input trace file(s). (Without restricting the `$dep` value, such a profile would double-count recursive dbcalls.) Use `--where=1 --where1=1` if you want no filtering whatsoever. Note that the expression values of `--where`, `--where1`, and sometimes `--name` are and-ed together to create the overall `mrskew` where-clause predicate. Thus, using `--where=A --where1=B --name=C` creates the filter `((A) and (B) and ($name=~/C/i))` to be applied to the input. Note that the `and ($name=~/C/i)` part of the predicate is applied only when you do not use one of the special ":" keywords as the `--name` option value.

`--where1=`***expr***`, --w1=`***expr*** Filter output data by *expr*; that is, include in the output only information from raw trace lines for which *expr* is true. An *expr* is any valid Perl expression. The expression can refer to any variable described in Variables and Functions below, and it can use any Perl operators or functions. See `--where` for examples.

The `--where1` option allows you to specify `--where` option values without having to continually remember to include the `$dep==$depmin and` part of the predicate. While you could refer to `$dep` values in `--where` expressions and non-`$dep` values in `--where1` expressions, we recommend that you use `--where1` to filter with `$dep`-related predicates and `--where` to filter with all other predicates.

EXPRESSIONS

Wherever `mrskew` requires an expression (such as in `--group` and `--where` option arguments), you may use any valid Perl expression consisting of:

Perl operators Operators such as `or xor and not , = ?: || && | ^ < > <= >= lt gt le ge == != <=> eq ne cmp + - . * / % =~ !~ ! ~ **` and so on. See http://perldoc.perl.org/perlop.html for details.

Perl builtin functions Functions such as `lc` `uc` `s///` `int` `join` `split` `sprintf` `substr` and so on. See http://perldoc.perl.org/perlfunc.html for details. Referring to an undefined function will result in an error.

`mrskew` variables and functions Any of the variable or function names described in Variables and Functions. The value of an undefined variable is the empty string for every line in the input trace file. Reference to an undefined function will produce a runtime error.

For more information about Perl expression syntax, see Perl's Tutorials.

VARIABLES AND FUNCTIONS

In grouping and filter expressions, you can reference any of the following variable names:

`$action_id` For a dbcall, remote procedure call (RPC), or syscall, `$action_id` is the trace file line number of the most recent ACTION line. The `$action_id` value is 0 if there is no preceding ACTION line. The `$action_id` value is always 0 for a MODULE line.

`$action_name` For a dbcall, remote procedure call (RPC), or syscall, `$action_name` is the Oracle `ACTION NAME` in context for the call.

`$ad` For a dbcall or a syscall, `$ad` is the Oracle cursor address in context for the call.

For a remote procedure call (RPC), `$ad` is 0.

`$af` For a dbcall or a remote procedure call (RPC), `$af` is the value of its `c` field, in seconds.

For a syscall, `$af` is the value of its *ela* field, in seconds.

`$af` is the call's accounted-for time. `mrskew` assigns `$af` this way so that specifying the following options (the defaults) will create a subroutine call profile:

```
--name='.+' --select='$af' --group='$name'
--where=1 --where1='$dep==$depmin'
```

`$base_name, $basename, $base` The basename of the file from which the line of text has been obtained.

`basename($fullname,@suffixlist)` The last level of a filepath. See http://perldoc.perl.org/File/Basename.html for details.

`@bind, $bind[`*i*`]` For a dbcall or syscall, `@bind` is an array containing the list of placeholder bind values in context for the call. To refer to the `Bind#0` value (the first placeholder value in that list), use the syntax `$bind[0]`. To refer to the `Bind#4` value, use `$bind[4]`. It is convenient to use the

expression `join(",",@bind)` to create a string showing all the values in the list, separated by commas. String data type values are presented with enclosing double quotes. Null values are presented as the empty string.

`$call` An alias for `$nam`.

`$call_id` An alias for `$line_number`.

`%calls, $calls{`*`key`*`}` Some distributed `.rc` files use the `%calls` variable to initialize histogram buckets so that `mrskew` will not print sparse output. Don't use it for any other purpose.

`$client_driver` For a dbcall, remote procedure call (RPC), or syscall, `$client_driver` is the Oracle `CLIENT DRIVER` in context for the call.

`$client_id` For a dbcall, remote procedure call (RPC), or syscall, `$client_id` is the Oracle `CLIENT ID` in context for the call.

`$client_ip` For a dbcall, remote procedure call (RPC), or syscall, `$client_ip` is the Oracle `CLIENT IP` in context for the call.

`$connection_id` For a dbcall, remote procedure call (RPC), or syscall, `$connection_id` is the Oracle `CONNECTION ID` in context for the call.

`$container_id` For a dbcall or remote procedure call (RPC) or syscall, `$container_id` is the Oracle `CONTAINER ID` in context for the call.

`$cpu, $c` For a dbcall or remote procedure call (RPC), `$cpu` is the value of the `c` field, in seconds. This is the CPU time consumed by a given dbcall, accurate on most operating systems to within only $\pm 10,000$ microseconds.

For a syscall, `$cpu` is 0.

`$cr` For a dbcall, `$cr` is the value of the `cr` field. `cr` is the number of database buffer cache accesses in consistent mode. See also the `$cu` and `$lio` variables.

For a syscall or remote procedure call (RPC), `$cr` is 0.

`$cu` For a dbcall, `$cu` is the value of the `cu` field. `cu` is the number of database buffer cache accesses in current mode. See also the `$cr` and `$lio` variables.

For a syscall or remote procedure call (RPC), `$cu` is 0.

`$cursor_id` For a dbcall or a syscall, `$cursor_id` is the value of the cursor ID field, which is the number that follows the symbol `#` in your trace data.

For a remote procedure call (RPC), `$cursor_id` is -1.

`$dep` For a dbcall, `$dep` is the value of the `dep` field.

For a syscall, `$dep` is set to the value of `$depmin` for the file.

For a remote procedure call (RPC), `$dep` is 0.

$depmin For any call, $depmin is the minimum recursive depth (dep value) found in the trace file. It is the level of top-most call stack depth in the file.

$dir_name, $dirname, $dir The dirname of the file from which the line of text has been obtained.

dirname($fullname) All but the last level of a filepath. See http://perldoc.perl.org/File/Basename.html for details.

$dur For a dbcall or remote procedure call (RPC), $dur is the value of the e field, in seconds.

For a syscall, $dur is the value of the ela field, in seconds.

$e For a dbcall or remote procedure call (RPC), $e is the value of its e field, in seconds.

For a syscall, $e is 0.

$ela For a dbcall or remote procedure call (RPC), $ela is 0.

For a syscall, $ela is the value of its ela field, in seconds.

$experience_id For a dbcall, remote procedure call (RPC), or a syscall, $experience_id is the Oracle EXPERIENCE ID in context for the call.

$exec_id For a dbcall, remote procedure call (RPC), or a syscall, $exec_id is the trace file line number of the most recent EXEC call that has the same $cursor_id as the current call. The $exec_id value is 0 if there is no preceding EXEC call for the current call's cursor. The $exec_id value is always 0 for a PARSE call.

$file_name, $filename, $file, $f The name of the file from which the line of text has been obtained. This is especially useful when using mrskew to process several input files in one run, so that you can learn the identities of the files that satisfy your query.

fileparse($fullname,@suffixlist) A three-element list containing the file's name, path, and suffix. See http://perldoc.perl.org/File/Basename.html for details.

$hv For a dbcall or a syscall, $hv is Oracle hash value that corresponds to the *cursorid* field of the current line of text. If there is no preceding PARSING IN CURSOR section for the cursor ID in context, then mrskew uses #*cursorid*:*file* as the hash value. When *cursorid* is zero, mrskew uses #0 as the hash value.

For a remote procedure call (RPC), $hv is the first 13 digits of the MD5checksum of the SQL text found in the most recent RPC CALL line.

$is_dbcall For a dbcall, BINDS, or remote procedure call (RPC), $is_dbcall is true.

Otherwise, it is false.

$island_id For each line in the trace file, $island_id is the line number of the line immediately following the most recent SQL*Net message from client call whose duration is the value of --thinktime or greater (or the line number of the first call in the file, if thre is no such recent SQL*Net message from client call). The $island_id of such a SQL*Net message from client call itself will be −1 times its line number.

The $island_id value is especially useful to group segments of a trace file created by a connection pooling application into chunks of trace data, each relating to a separate end-user experience. See Examples for a use case.

$is_oscall For a syscall, $is_oscall is true.

Otherwise, it is false.

$len For a dbcall or syscall, $len is the value of the len field in context for the call.

$lid For a dbcall or syscall, $lid is the value of the lid field in context for the call.

$lio For a dbcall, $lio is $cr + $cu.

For a syscall or remote procedure call (RPC), $lio is 0.

$line_number, $line, $call_id, $l, $NR The line number of a line within the trace data. With this variable, you can find out what lines of trace data are responsible for your greatest time consumptions for a given timed event. Grouping by an expression containing both $file_name and $line_number enables you to pinpoint the exact line within a directory of trace files that satisfies your query.

$mis For a dbcall, $mis is the value of the mis field, which is the number of misses upon the library cache encountered for that dbcall.

For a syscall or remote procedure call (RPC), $mis is 0.

$module_id For a dbcall, remote procedure call (RPC), or syscall, $module_id is the trace file line number of the most recent MODULE line. The $module_id value is 0 if there is no preceding MODULE line.

$module_name For a dbcall, remote procedure call (RPC), or a syscall, $module_name is the Oracle MODULE NAME in context for the call.

mrtime($tim) The ISO 8601 date and time string that corresponds to a tim value.

$nam, $name, $call, $call_name For a dbcall or remote procedure call (RPC), $nam is the call name (e.g., PARSE, EXEC, or RPC EXEC).

For a syscall, $nam is the value of the nam field, which is the Oracle-given name for the syscall.

$obj For a dbcall or remote procedure call (RPC), $obj is 0.

For a syscall, $obj is the value of the obj# field, which is the Oracle object ID of an object being manipulated. For syscalls reported by Oracle Database versions prior to 10.2, $obj is 0.

$oct For a dbcall or syscall, $oct is the value of the oct field in context for the call.

$oerr For an ERROR call, $oerr is the value of the err field for the call. For other calls, $oerr is 0.

$oracle_release For a dbcall, remote procedure call (RPC), or a syscall, $oracle_release is the Oracle Database software release in context for the call or ? if no Oracle Database release banner has been encountered or found within the limits of --scanmax. See --scanmax for details.

$os The name of the operating system recorded in the trace file.

$p1; $p2; $p3 For a dbcall or remote procedure call (RPC), $p1, $p2, $p3 are all 0.

For a syscall, $p1, $p2, $p3 are the values of the three parameter fields that Oracle emits for each syscall. In Oracle versions prior to 10.2, these parameters were named p1, p2, and p3, respectively. From Oracle version 10.2 onward, these three fields have more descriptive names like file # and block #. However, there are still only three such attributes per WAIT line, and mrskew still uses the names $p1, $p2, and $p3, regardless of what those attributes are called in the trace file. Oracle publishes definitions of these fields in the V$EVENT_NAME view.

Use the following Oracle SQL query in to view the functional Oracle definition of the data contained in the context-dependent p1, p2, and p3 fields:

```
select name, parameter1, parameter2, parameter3
from v$event_name where name='your nam value goes here'
```

$parse_id For a dbcall, remote procedure call (RPC), or a syscall, $parse_id is the trace file line number of the most recent PARSE call that has the same $cursor_id as the current call. The value is 0 if there is no preceding PARSE call for the current call's cursor.

$pio, $p For a dbcall, $pio is the value of the p field, which is the number of Oracle blocks obtained by OS read calls.

For a syscall or remote procedure call (RPC), $pio is 0.

$plh For a dbcall, $plh is the value of the plh field, which is the execution plan hash value for the cursor being closed.

For a remote procedure call (RPC) or syscall, $plh is 0.

$rd_only For an XCTEND dbcall, $rd_only is the value of the rd_only field, which is 1 if the transaction was read-only, or 0 if it was read-write.

For all other calls, $rd_only is 0.

$rlbk For an XCTEND dbcall, $rlbk is the value of the rlbk field, which is 1 if the transaction was rolled back, or 0 if it was committed.

For all other calls, $rlbk is 0.

$row, $r For a dbcall, $row is the value of the r field, which is the number of rows returned by a given dbcall.

For a remote procedure call (RPC) or syscall, $row is 0.

$serial_number, $serial For a dbcall or remote procedure call (RPC) or syscall, $serial_number is the Oracle session serial number in context (from the SESSION ID line) for the call.

$service_name For a dbcall or remote procedure call (RPC) or syscall, $service_name is the Oracle SERVICE NAME in context for the call.

$session_id, $sid For a dbcall or remote procedure call (RPC) or syscall, $session_id is the Oracle SESSION ID in context for the call.

$sql For a dbcall or remote procedure call (RPC) or syscall, $sql is the SQL text in context for the call, normalized by replacing sequences of whitespace (blanks, tabs, newlines, formfeeds, etc.) by a single space character. To use only a substring of the SQL text, use substr($sql,*offset*,*length*). For example, use substr($sql,0,50) to refer to the first 50 characters of a statement's text; use substr($sql,20,50) to refer to the 21st through the 70th characters; or use substr($sql,-50) to refer to the last 50 characters.

$sqlid For a dbcall or a syscall, $sqlid is the Oracle SQL ID value that corresponds to the *cursorid* field of the current line of text. If there is no *sqlid* field in the PARSING IN CURSOR section for the cursor ID in context, then mrskew uses hv=*hv* as the SQL ID value. If there is no preceding PARSING IN CURSOR section for the cursor ID in context, then mrskew uses #*cursorid*:*file* as the SQL ID value. When *cursorid* is zero, mrskew uses #0 as the SQL ID value.

For a remote procedure call (RPC), $sqlid is the first 13 digits of the MD5 checksum of the SQL text found in the most recent RPC CALL line.

$ssql For a dbcall or remote procedure call (RPC) or syscall, you can think of $ssql as the shareable version of a SQL or PL/SQL statement. $ssql is the SQL text in context for the call, normalized by (1) replacing sequences of whitespace (blanks, tabs, newlines, formfeeds, etc.) by a single space character, (2) replacing literal values by placeholder variables, (3) replacing each comment with an empty string, (4) replacing each IN and NOT IN list

with an empty list, and (5) replacing any suffix of two or more digits on an object name with an empty string.

$ssqlid For a dbcall or a syscall, $ssqlid is a synthesized Oracle SQL ID-like value for the $ssql text that corresponds to the *cursorid* field of the current line of text. If there is no preceding PARSING IN CURSOR section for the cursor ID in context, then mrskew will set $ssqlid to the same value as $sqlid. When *cursorid* is zero, mrskew uses #0 as the $ssqlid value. You can think of $ssqlid as the SQL ID of the shareable version of a SQL or PL/SQL statement.

$text The entire line of raw trace data. You can use this variable to create your own Perl expression to process each input line any way you like.

$tim For a dbcall or a syscall, $tim is the value of the tim field, in seconds. tim is the time at which the call concluded. For syscalls reported by Oracle Database versions prior to 10.2, $tim is 0. For a remote procedure call (RPC), $tim is 0.

$tim0 For a dbcall or a syscall, $tim0 is the time value at which the call began, in seconds. $tim0 is equal to $tim1 - $af. For syscalls reported by Oracle Database versions prior to 10.2, $tim0 is -$ela. For a remote procedure call (RPC), $tim0 is 0.

$tim1 $tim1 is an alias for $tim.

$tim1prior For a dbcall or remote procedure call (RPC) or syscall, $tim1prior is the time value at which the prior dbcall ended, in seconds. Note that syscall lines do not change the value of $tim1prior.

$time0 $time0 is the ISO 8601 date and time string that corresponds to the line's $tim0 value. $time0 is equal to both mrtime($tim0) and mrtime($tim1 - $af) but faster.

$time1 $time1 is the ISO 8601 date and time string that corresponds to the line's $tim1 value. $time1 is equal to both mrtime($tim1) and mrtime($tim0 + $af) but faster.

$type For a CLOSE dbcall, $type is the value of the type field.

For any other dbcall or a syscall, $type is 0.

$uafbc For a dbcall, $uafbc is $tim0 - $tim1prior, except for the first dbcall in the file. For the first dbcall in the file, $uafbc is 0. For a syscall, $uafbc is 0.

For syscalls reported by Oracle Database versions prior to 10.2 and for remote procedure calls (RPC), $uafbc is -$ela - $tim1prior.

$uafwc For a dbcall or remote procedure call (RPC) or syscall, $uafwc is the value of $e - ($c + $ela).

`$uafwc` is a dbcall's `unaccounted-for within dbcalls` time. If you group by call names, the `$uafwc` value will be negative for each syscall (`$e - ($c + $ela)` = `-$ela` for a syscall). But if you use, for example, `--group='$sqlid'` or `--group='"$module_name/$action_name"'`, you'll get a useful summary of the duration that is unaccounted-for within your trace data.

`$uid` For a dbcall or syscall, `$uid` is the value of the `uid` field in context for the call.

.RC FILES

A `.rc` file allows you to change the default behavior of a Method R Workbench utility. Method R ships several packaged `.rc` files, and you can customize or write your own.

Packaged `.rc` Files

`mrskew` comes with the following prepackaged `.rc` files, any of which can be used with `mrskew` with the `--rc=`*file* command line option:

`all.rc` A shorthand for `--name=:all --where1=1 --top=0 --sort=1na --nohistogram`. We commonly use these options in conjunction with a `--group` expression that includes `$line` to show `mrskew` variable states in a line-by-line narrative. For example, the following command shows the value of `$parse_id` and `$exec_id` for each line of the input trace file:

```
mrskew ora_1492.trc \
    --group='sprintf("%8d %8d %8d %s", $line, $parse_id, $exec_id, $text)' \
       --gl='sprintf("%8s %8s %8s %s", "LINE#", "PARSE_ID", "EXEC_ID", "TEXT")' \
    --rc=all.rc
```

`calls.rc` List database calls in chronological order.

`disk.rc` Group calls by duration into buckets suitable for analysis of calls on traditional spinning-disk storage devices.

`distinct-texts.rc` Count SQL texts that should have been shared, grouping by the `mrskew` synthesized shared SQL ID.

`first-read-only.rc` Group `db file sequential read` calls by data bock address `'$p:$p2'`, but show how much time would have been consumed by each block if the block had been read only its first time. Use this for deeper insight into how much time is being wasted by re-reading the same block more than once into the database buffer cache. `first-read-only.rc` will work properly only for single-block read calls (that is, when `$p3 == 1`). When you use `first-read-only.rc`, the DURATION-1ST-READ-ONLY column will omit information about each block's second and subsequent

reads. The CALLS, MEAN, MIN, and MAX columns will continue to represent *all* of the `db file sequential read` calls executed upon a block (not just the first read for each block).

`island.rc` Group calls by "`$file:$island_id`", which gives the file name and line number that begins an "island" of activity in a trace file between "oceans" of long-duration `SQL*Net message from client` calls. `island.rc` groups response time by this island ID, which creates a report of response times of end-user experiences.

`p10.rc` Group calls by duration into buckets with power-of-ten partitions 1us, 10us, 100us, 1000us, etc.

`shareable-texts.rc` Group `PARSE` calls by texts of statements that should have been shared.

`ssd.rc` Group calls by duration into buckets suitable for analysis of calls on solid state storage devices.

Some of these `.rc` files contain multi-byte Unicode characters that will not display properly on Microsoft Windows command shells. Each file contains instructions for how to replace those Unicode characters with ASCII strings that will render properly (albeit less beautifully).

Custom `.rc` Files

You can also write or customize your own `.rc` files. For example, the following `.rc` file will change your personal default value of `--top` to 12:

```
$ cat ~/.mrskew.rc
--top=12
```

You can also use a `.rc` file to store a complicated options list that you code carefully one time and then reuse:

```
$ cat mrskew-buckets.rc
# Bucket calls by duration.
--gl='"BUCKET"'
--group='$dur < .000010 ? " 0us <= dur < 10us"
       : $dur < .000020 ? "10us <= dur < 20us"
       : $dur < .000030 ? "20us <= dur < 30us"
       : $dur < .000040 ? "30us <= dur < 40us"
       :                  "40us <= dur       "'
```

Such a `.rc` file would be called into use like this:

```
$ mrskew --rc=mrskew-buckets.rc --name='.*read.*' ora_1492.trc
```

A `.rc` file line beginning with the `#` character (in the first column of the line) is a comment. A comment must be on a line by itself; do not append a comment to a line containing other content. There is no need to use a line continuation

character to specify a multi-line option. Do not use the following options inside a `.rc` file:

```
--debug
--initrc   --noinitrc
--listrc   --nolistrc
--regress  --noregress
```

A `.rc` file isn't limited to just command line *options*; you can specify command line *arguments* as well. For example:

```
$ cat myfiles.txt
"path/bingham_ora_12896.trc"
"path/bingham_ora_33427.trc"
```

Then using `--rc=myfiles.txt` would insert the two named files into the argument list. This is especially useful when you have so many file names that they cannot be listed on the command line without exceeding your system's command line length limit. (The quotation marks are a good habit to get into in case your path has spaces in it.)

Auto-Executed `.rc` Files

By default, a Method R Workbench utility will interpret the options listed in two special `.rc` files, in the following order, before it interprets the options you list on your command line:

```
~/.mrskew.rc    # the file named .mrskew.rc in your home directory
./.mrskew.rc    # the file named .mrskew.rc in your current working directory
```

All Method R Workbench utilities use the same `.rc` file naming convention; for example, `mrskew` opens `~/.mrskew.rc` and `./.mrskew.rc`, `mrkey` opens `~/.mrkey.rc` and `./.mrkey.rc`, and so on. If you want for a program not to interpret the options in these files automatically, then specify `--noinitrc` on the command line.

Upon encountering the `--rc=`*file* option, `mrskew` will execute the options listed within the named file, as if those options were specified on the command line right where the `--rc` option was used. For example, imagine that you have the following `.rc` files on your system, with `MRTOOLS_RCPATH` set to include `/d`:

```
$ cat ~/.mrskew.rc
--gl='"HOME"'

$ cat ./.mrskew.rc
--gl='"CWD"'

$ cat /d/a.rc
--gl='"A1"'
--rc=b.rc
```

```
--gl='"A2"'

$ cat /d/b.rc
--gl='"B"'
```

Then running this command:

```
mrskew myfile.trc --rc=a.rc
```

. . . would result in the following sequence of command line option assignments:

```
mrskew myfile.trc --gl='"HOME"' --gl='"CWD"' --gl='"A1"' --gl='"B"' --gl='"A2"'
```

The result is the setting `--gl='"A2"'`; the final `--gl` setting overrides the others.

To trace the execution of `.rc` file contents as they are processed, use `--debug` or `--debug=`*level* with *level* ≥ 1.

Executing `.rc` files in this order makes it easy for you to create "layers" of default values. For example, the product default value for the `mrskew --top` option is 20. You could set your personal default value to 10 by using `--top=10` in your `~/.mrskew.rc` file. Imagine that you are writing an analysis about files in directory `/d`, and for only this analysis, you want to use `--top=5` for all your `mrskew` reports. You could specify `--top=5` on the command line, but that would be tedious. You could edit `~/.mrskew.rc`, but then you would have to remember to set it back when your analysis is finished. The best solution is to create the file `/d/.mrskew.rc` and list `--top=5` within it. Then any `mrskew` command run from within the directory `/d` would use the default value of 5 and your default value in other directories would remain 10. You could, of course, override this default value on any `mrskew` command by specifying a new `--top` value on the command line.

See the description of the MRTOOLS_RCPATH environment variable for details on where the `--rc` option searches for files.

OS DEPENDENCIES

For some option values, your operating system may require quotation marks around the value. For example, passing the option value `$p1` into a tool on a Linux or macOS system requires the use of single quotes to prevent the command shell from interpreting `$p1` as the value of a shell variable. Different command shells have different quoting rules, but most fall into one of two categories: shells that behave like the Unix Bourne shell, or shells that behave like the DOS shell. Linux, macOS, and the Method R Workbench application all use shells that behave like Unix shells. Here are some examples of how to quote option values in each type of shell:

Unix	DOS
`--option='$p1'`	`--option=$p1`
`--option='"$p1.$p2"'`	`--option="""$p1.$p2"""`
`--option='join("::",$file,$line)'`	`--option="join(\"::\",$file,$line)"`
`--option='db.*read'`	`--option=db.*read`
`--option='latch free'`	`--option="latch free"`

This manual page uses the Unix-style line continuation character `\`.

```
# Unix
mrskew \
--name=exec \
*.trc
```

On DOS-style systems, the line continuation character is `^`.

```
REM DOS
mrskew ^
--name=exec ^
*.trc
```

Some `.rc` files contain multi-byte Unicode characters that do not display properly on some systems. Each `.rc` file containing a Unicode character contains detailed instructions about how to replace the Unicode character with ASCII text having equivalent meaning.

EXAMPLES

This command will emit a profile of calls from both input files, grouping those calls by their call names:

```
mrskew ora_1492.trc ora_1493.trc
```

The next command will emit a skew histogram for all `db file sequential read` and `db file scattered read` calls (and any other Oracle timed event calls matching the Perl regular expression `/db.*read/i`), grouped by elapsed duration per call:

```
mrskew --name='db.*read' --rc=p10.rc ora_1492.trc
```

The next command will print a skew histogram for the same calls, except this time they'll be grouped by the Oracle `p3` parameter for the calls, which in this case is the number of blocks obtained in each read call (see the Oracle `v$event_name` fixed view for more information). Note that in Unix-style shells, the single quotes in `'$p3'` are necessary for the reasons mentioned previously. If you forget this, you'll get a warning like "Option group requires an argument".

```
mrskew --name='db.*read' --group='$p3' ora_1492.trc
```

The next command will produce the same output as the one before, except that the output will have a nicer column header for the group-by column:

```
mrskew --name='db.*read' --group='$p3' --gl='"BLKS/CALL"' \
    ora_1492.trc
```

You can group on expression values and format your own group header with an expression:

```
mrskew --name='db.*read' --group='sprintf("%8d %8d", $p1, $p2)' \
    --gl='sprintf("%8s %8s", "FILE", "BLOCK")'
    ora_1492.trc
```

The next command will print a skew histogram for the same calls, but this time they'll be grouped by file ID and block ID. Note that by default, `mrskew` shows only the first 20 lines of the histogram, with a single additional line summarizing all the additional lines. If you want to see all the lines, you can specify `--top=0`, but be warned: doing that can produce a *lot* of output. Also note that grouping by a high-cardinality expression like `'$p1.$p2'` increases the likelihood that `mrskew` will use a lot of memory on your computer.

```
mrskew --name='db.*read' --group='"$p1.$p2"' ora_1492.trc
```

This command will show Oracle `/db.*read/i` calls with elapsed durations between .01 seconds (inclusive) and .1 seconds (exclusive), grouped by the Oracle `p3` parameter value (the number of Oracle blocks obtained in each read call) at the minimum recursive depth in the file:

```
mrskew --name='db.*read' \
    --where='.01<=$ela and $ela<.1' \
    --group='$p3' \
    ora_1492.trc
```

To show the file name and line number and the call name for each of the five longest Oracle call durations in all the `.trc` files within your current working directory:

```
mrskew --group='"$file:$line $name"' --gl='"FILE:LINE CALL-NAME"' \
    --where1=1 --top=5 *.trc
```

To show which trace files contain evidence of Oracle declarative parallel execution operations:

```
mrskew --name='PX.*' --group='$file' --gl='"FILE"' --where1=1 *.trc
```

To show which files have the most time spent waiting for Oracle timed events with the string `latch` in their names:

```
mrskew --name='.*latch.*' --group='$file' --gl='"FILE"' *.trc
```

To show a response time profile for all the timed events found in the input file, the following two commands are equivalent:

```
mrskew --name='.+' --group='$name' --gl='"SUBROUTINE CALL"' ora_1492.trc
mrskew --gl='"SUBROUTINE CALL"' ora_1492.trc
```

To show the hash values of the most time-consuming SQL statements in all the `.trc` files within your current working directory:

```
mrskew --group='$hv' --where1=1 *.trc
```

To show the same thing with SQL IDs instead of hash values:

```
mrskew --group='$sqlid' --where1=1 *.trc
```

To show the same thing with SQL text (the first 50 characters) instead of SQL IDs:

```
mrskew --group='substr($sql,0,50)' --where1=1 *.trc
```

To show the SQL ID, but only in files that are Oracle version 11 or above:

```
mrskew --group='$sqlid' --where='$oracle_release>=11' *.trc
```

To show which files in the current working directory have the most time contributed by the SQL text with hash value 2343063137:

```
mrskew --group='$file' --where='$hv eq 2343063137' --where1=1 *.trc
```

To show time consumption for all operating system calls executed by Oracle for the given file(s):

```
mrskew --name=:oscall ora_1492.trc
```

To show the names of all files in the current working directory that have references to cursor `#(nil)`:

```
mrskew --group='$file' --where='$cursor_id eq "(nil)"' *.trc
```

To show the service, module, and action names that consumed the most time:

```
mrskew --group='"$service_name.$module_name.$action_name"' ora_1492.trc
```

To show the line number and the Oracle session ID and serial number of the most time-consuming calls:

```
mrskew --where1=1 --group='"$line:$sid.$serial"' ora_1492.trc
```

To show how the module and action values track line-by-line through a trace file:

```
mrskew --name=:all --where1=1 --top=0 --nohistogram --sort=1a \
    --group='sprintf("%8d %4d %-35.35s %-.45s", \
    $line, $sid, $module_name.":".$action_name, $text)' \
    ora_1492.trc
```

To show how hash values track line-by-line through a trace file:

```
mrskew --name=:all --where1=1 --top=0 --nohistogram --sort=1a \
    --group='sprintf("%8d %10s %-.85s", $line, $hv, $text)' \
    ora_1492.trc
```

To show at what time each call began and ended (note that the value of `mrtime($tim1)` is identical to the value of `$time1`):

```
mrskew --group='sprintf("%5d %27s %27s %s", \
    $line, $time0, mrtime($tim1), $text)' --rc=all.rc \
    ora_1492.trc
```

To show how much time was consumed by calls associated with each parse call in the trace file:

```
mrskew --group='$parse_id' --gl='"PARSE-ID"' ora_1492.trc
```

To show how much time was consumed by calls associated with each execute call in the trace file:

```
mrskew --group='$exec_id' --gl='"EXEC-ID"' ora_1492.trc
```

To show which set of bind values accounts for the most time spent executing and fetching for a given SQL ID:

```
mrskew --group='join(",",@bind)' \
    --where='$sqlid eq "53saa2zkr6wc3" and ($call =~ /EXEC|FETCH/)' \
    ora_1492.trc
```

To show which value of the "three"-th placecholder variable (that is, "Bind#3") accounts for the most response time spent executing and fetching for a given SQL ID:

```
mrskew --group='$bind[3]' \
    --where='$sqlid eq "53saa2zkr6wc3" and ($call =~ /EXEC|FETCH/)' \
    ora_1492.trc
```

To understand how `mrskew` sets the `$island_id` variable for each line of its input:

```
mrskew --group='sprintf("%5d %5d %s", $line, $island_id, $text)' \
    --rc=all.rc ora_1492.trc
```

To show a list of user experience durations in a trace file created by a connection pooled application, use `--rc=island.rc`:

```
mrskew --rc=island.rc ora_1492.trc
```

To drill into an individual user experience with a `$island_id` value of 141281:

```
mrskew --where='$island_id == 141281' ora_1492.trc
```

`mrskew` is a tool that lets you stretch your imagination. This `mrskew` command identifies every EXEC dbcall in the current working directory, at the file's minimum recursive depth, that has one or more library cache misses:

```
mrskew \
    --name=exec \
    --group='sprintf "%s:%d %d", $f, $l, $mis' \
```

```
    --where='$mis > 0' \
    --gl='"FILE:LINE MISSES"' \
    --nohistogram \
    *.trc
```

And this `mrskew` command shows every FETCH dbcall in the current working directory, at the file's minimum recursive depth, that has a buffer cache hit ratio value (if you're interested in such things) of less than 40%:

```
mrskew \
    --name=fetch \
    --group='sprintf "%30s:%7d %9d %9d %7.3f", \
      $file, $line, $lio, $pio, ($lio-$pio)/$lio' \
    --where='$lio > 0 and ($lio-$pio)/$lio < .4' \
    --gl='"FILE:LINE LIO PIO HIT_RATIO"' \
    --nohistogram \
    *.trc
```

SECURITY

`mrskew` executes Perl code that the `mrskew` user types into a command-line argument. It is thus possible for a user to vandalize a system by running `mrskew`. This is not generally a problem, because a `mrskew` user typically has access to other operating system commands, like `rm(1)` or `perl(1)`, that could do just as much damage with far less effort. The probability of harming a system *by accident* with `mrskew` is vanishingly small; using `mrskew` as an implement of vandalism would require directed effort. But if your system is one of those in which certain users with command-line access are prohibited from executing specific operating system commands (like `rm(1)` or `perl(1)`), then those users should be prohibited from executing `mrskew` as well.

EXIT STATUS

Exit status is 0 on successful completion, and > 0 if an error occurs.

ENVIRONMENT

MRTOOLS_RCPATH

The `MRTOOLS_RCPATH` environment variable contains a list of directories that each `--rc=`*file* option will search for *file*. Format is identical to your platform's `PATH` environment format (e.g., `.:a:a/b` in Linux, `.;c:a;c:a\b` in Windows). If *file* begins with `/`, `.`, or `~`, then `--rc` looks for the file in the location you have specified. Otherwise, `--rc` will search each directory named in the `MRTOOLS_RCPATH` list for *file*, using only the first readable file that it finds.

.mrskew.rc

By default, `mrskew` will execute the options listed in the following files, in the following order, before the options you actually list on your command line:

```
~/.mrskew.rc
./.mrskew.rc
```

If you do not wish to execute the options in these files, then specify `--noinitrc` on the command line.

PERL5LIB

`mrskew` searches the following paths for the Perl modules it needs:

```
$PERL5LIB
$HOME/.method-r/workbench/26.1.5/perlmods
install-dir/perlmods
```

You can thus customize the Perl modules you want `mrskew` to use.

`PERL5LIB` First, `mrskew` will search the directories listed in the value of the `PERL5LIB` environment variable. If you want to add or change a Perl module, you can store it in one of the directories that are listed in the `PERL5LIB` value, or you can add a directory to the `PERL5LIB` value and put the module there. Note that for the Method R Workbench application to have access to the `PERL5LIB` environment variable, you'll have to set `PERL5LIB` globally.

`$HOME/.method-r/workbench/26.1.5/perlmods` After searching the directories listed in `PERL5LIB`, `mrskew` will search the versioned Method R Workbench `perlmods` directory. Add or change modules here that you want to be used only by Method R Workbench version 26.1.5.

***install-dir*`/perlmods`** This is the location of the Perl modules that we distribute with Method R Workbench (where *install-dir* is the name of the parent directory of the Method R Workbench `bin` directory). Do not add or change files in this directory.

BUGS AND DEFICIENCIES

The definitions of `$uafbc` and `$uafwc` are mathematically sensible given the design of `mrskew` as a call processor. But they're not going to give the values that you're usually looking for when you are analyzing unaccounted-for time. The best tool to use for analyzing unaccounted-for time is `mrprof --trace=ut`.

AUTHORS

Cary Millsap, Jeff Holt

VERSION

`mrskew` 26.1.5

COPYRIGHT AND LICENSE

Copyright 2008, 2026 Method R, LLC. All rights reserved.

This is commercially licensed software, governed by the Method R End User License Agreement. Please confirm with your software license administrator that you are licensed to use this product. Write license@method-r.com for more information.

Method R Workbench includes a partial distribution of Perl. The entire distribution is available at StrawberryPerl.com for Microsoft Windows, or CPAN.org for Linux and macOS.