Revision	Description
7/21/2010	Original

SQL-Hero Tracing

Introduction

Let's start by asking why you might want to do SQL tracing in the first place. As it turns out, this can be an extremely useful activity for debugging. For example, if you're not familiar with what a particular piece of logic is doing in the database, a trace can reveal exactly what's going on. Even for code that you are already familiar with, capturing the exact parameters used when a specific stored procedure is called can be very helpful. It's also incredibly valuable for diagnosing performance problems in SQL logic.

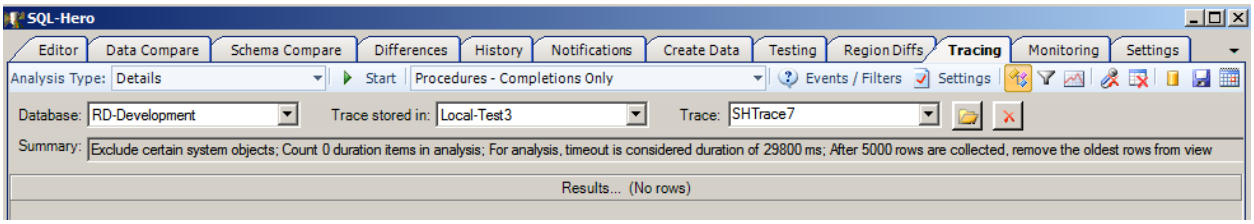
The tracing tool in SQL-Hero tries to deal with the following weaknesses found in the out-of-the-box SQL Profiler tool:

- Should be able to manipulate the trace grid (sort, filter, search with more options, use coloring, etc.), and extract information from it easily (e.g. copy information into e-mails, identify "call sequences", etc.)
- Should be able to analyze the data in the grid, in place (e.g. average execution times, etc.)
- Should be able to save the data in the grid to a repository where can then later do comparative analysis over time, search for previously collected data, and have the ability to deliver flexible analysis reports via e-mail
- Ability to keep the contents of the grid "relevant" (e.g. keep most recent rows in view and provide advanced filtering)
- Provide active notifications when conditions of choice are detected
- Provide a view of trace data that is anything other than a standard grid – SQL-Hero offers two special views of trace data that are graphical and convey important information

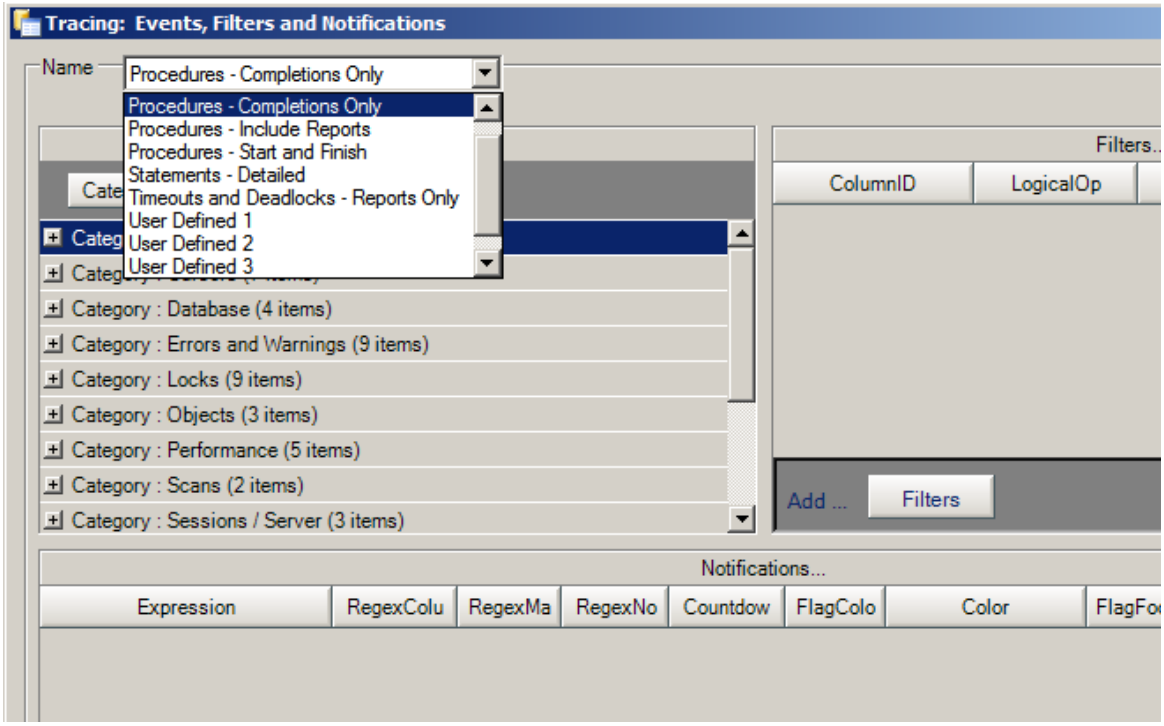
Another side goal is to make the ability to start tracing easy and accessible, such as is offered with Visual Studio integration. Furthermore, the integration of the various tools allows you to take collected trace records and turn them directly into SQL that can be executed, with 2 mouse clicks.

Tracing Tool

Let's take a detailed look at the Tracing tool:

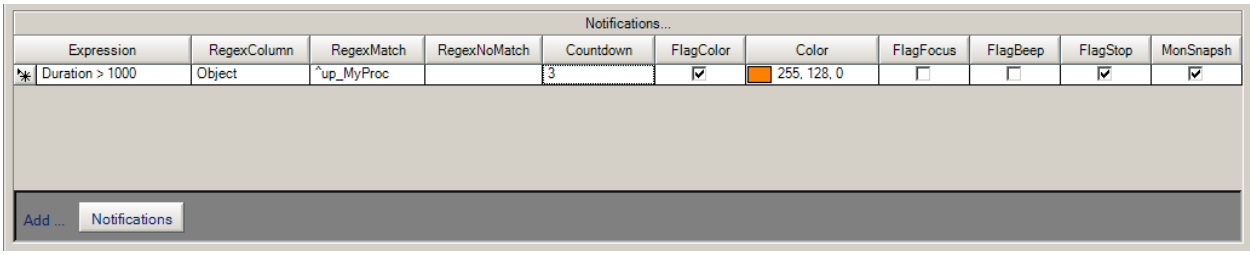


Here we see that the "Analysis Type" is set to "Details". This means that the grid will show each individual trace record, as opposed to a summary report style, which we'll look at later. "Procedures – Completions Only" in this case refers to the type of trace we wish to run. There are other types of trace which capture more detail, but "Procedures – Completions Only" is generally a good choice for capturing stored procedure and SQL batch completions. "Events / Filters" (Events / Filters) invokes a pop-up that lets you specify advanced filters and establish criteria that can be used to work as alerts:

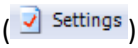


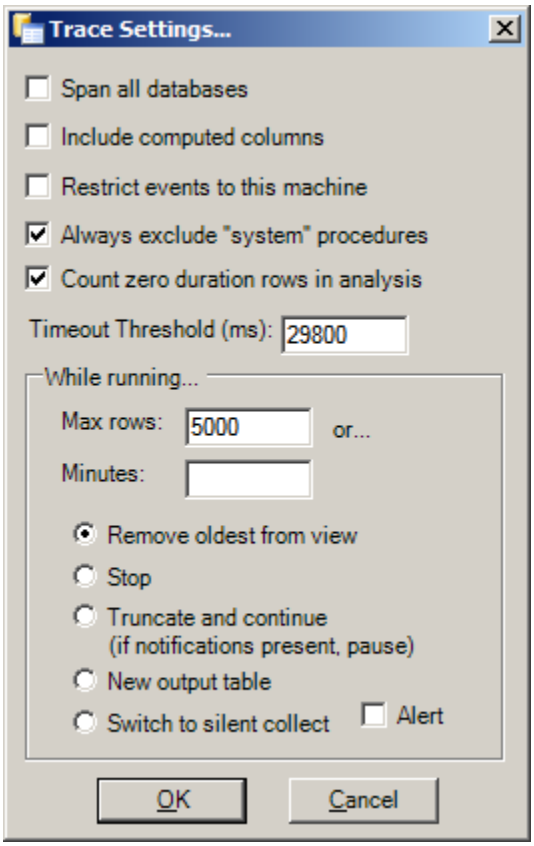
Notice that there are a number of "canned" trace types, but you can customize your own using the user defined types. (These let you pick the events to be captured, although you can set Filters and Notifications on *any* type.)

In terms of Notifications, here is one example of what you could do:



Once a trace is running, if a trace record is collected where the Duration is greater than 1000 (milliseconds), the object name starts with "up_MyProc", and it's the third time this appears, then the collected row will be shown in orange, the trace will be stopped, and if the monitoring tool is running, a snapshot will be taken on it. (See details on the monitoring tool in a different whitepaper.)

Back to the main toolbar, the "Settings" command () lets you control certain behaviors about how tracing will work:



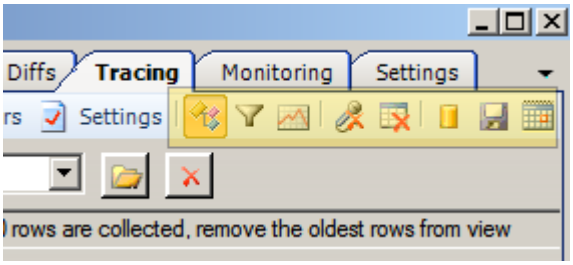
Here we can control:

- Whether we want the trace to span all databases (the default is to target a specific database of interest)
- Include some additional trace columns which are not normally part of SQL traces, out-of-the-box

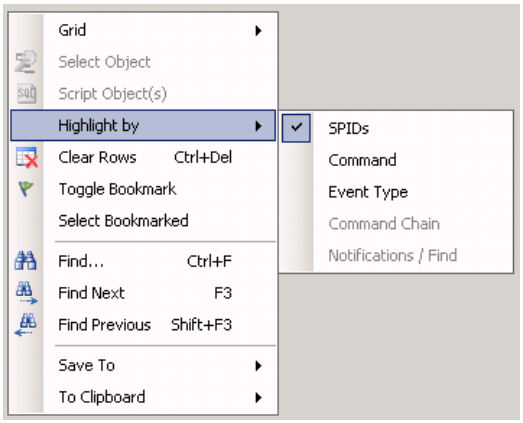
- Apply an event filter to only show events that originate from the current machine
- Exclude some common system calls which can "muddy" your traces with unnecessary detail
- Whether we will count zero duration records as part of some analysis reports
- Establish the number of milliseconds that will tell the trace tool that a particular call may have hit a timeout (used in some kinds of analysis reports)

Furthermore, to limit the quantity of data in the grid for clarity (and performance) you can specify limits and behaviors related to the number of trace records collected. In the settings shown above (default), only the last 5000 trace records are shown in the grid, although *all* records are still persisted into your trace table.

Returning back to the main tool window:



The first button ( ) after "Settings" is a state button that indicates whether you wish to color rows in the grid or not. (This only applies against stopped traces.) There are different ways to apply coloring, available from the grid's context menu (right-click):



The filter button ( ) invokes the global object filter screen. This lets you apply a wide range of criteria to limit what will be captured during an active trace, or loaded from saved trace data.

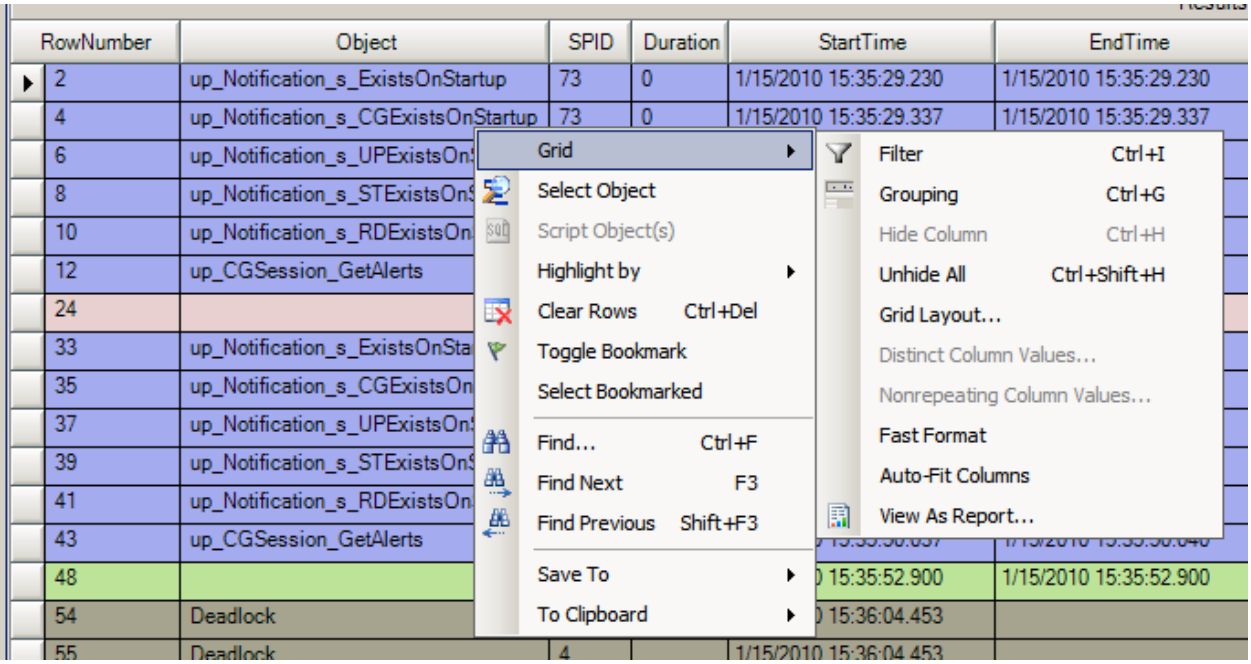
The silent button ( ), when toggled to the "on" state causes events to continue to be collected while a trace is running, but they will not be shown in the grid until the trace stops. This is useful when you're collecting a large number of events to reduce load on your system and load on the event source SQL

Server instance. In fact, one option described above lets you switch to silent mode automatically when a certain number of events are captured.

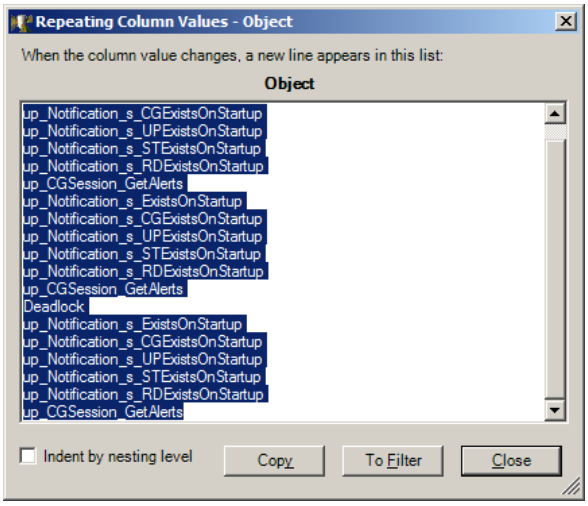
The clear results button ( ) removes the current contents of the grid. This does *not* delete the rows that may exist in your trace table.

To actually start a trace, you need to provide some basic information. You can pick the database you will be tracing, pick a database that you will store the collected trace records in, and name a trace table that will hold the trace records. Unlike SQL Profiler, you *must* store trace results in a table here, in this version of SQL-Hero. (Note that you can *load* traces from .trc files.)

To actually start your trace, use the "Start" (Start) command; you can then Stop or Pause a running trace. Once you have collected trace records, this grid context menu offers a number of options.



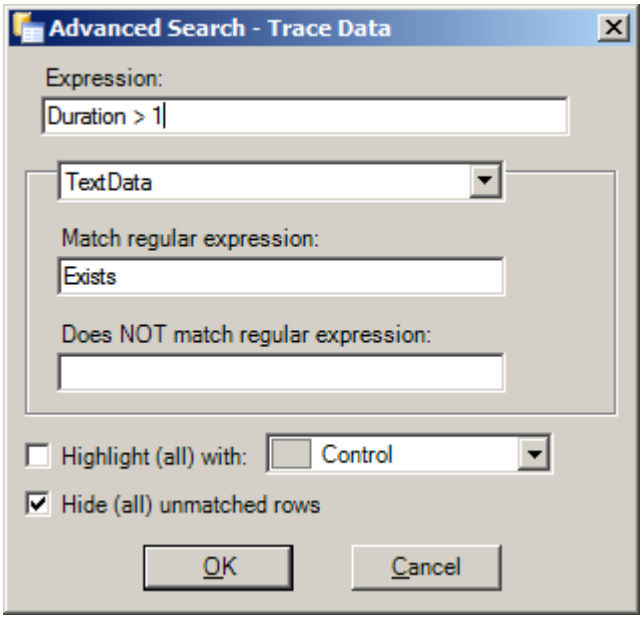
Grid actions are relatively standard throughout the application, but "Nonrepeating Column Values" takes on special significance for tracing. With this option you can build a "pasteable" list - typically of object names - showing call sequence:



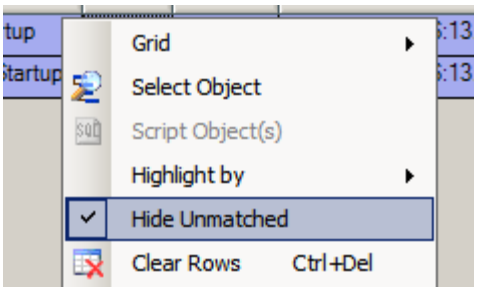
The "Select Object" command ( ) is available when an object name is selected and causes a switch to the Editor tool where the named object is searched for and selected. (The database is assumed to be the database named in the "Database:" drop down.)

The "Script Object(s)" command ( ) takes one or more captured SQL commands, concatenates them, and places them in a new Editor tool window, ready for execution if you wanted to run them (again). (You can select multiple rows in the grid by holding Ctrl as you click on row selectors.)

The "Find" command ( ) offers some searching capabilities that are especially helpful for analyzing traces. In the example below, we'll locate records with Duration > 1 and the TextData column containing "Exists". Rows that do not meet these criteria are hidden from view.

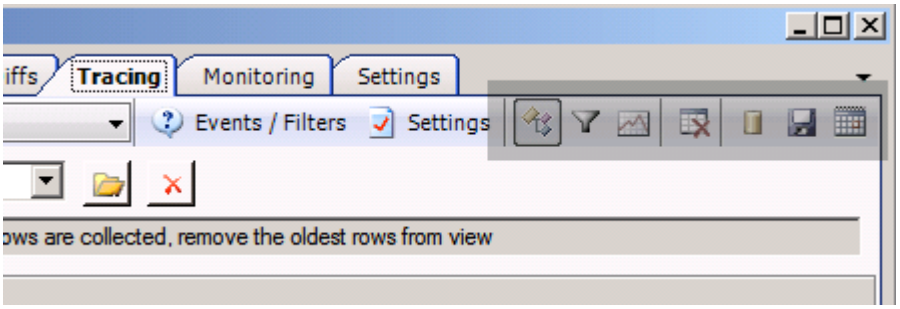


To bring all rows back into view, uncheck the "Hide Unmatched" option:

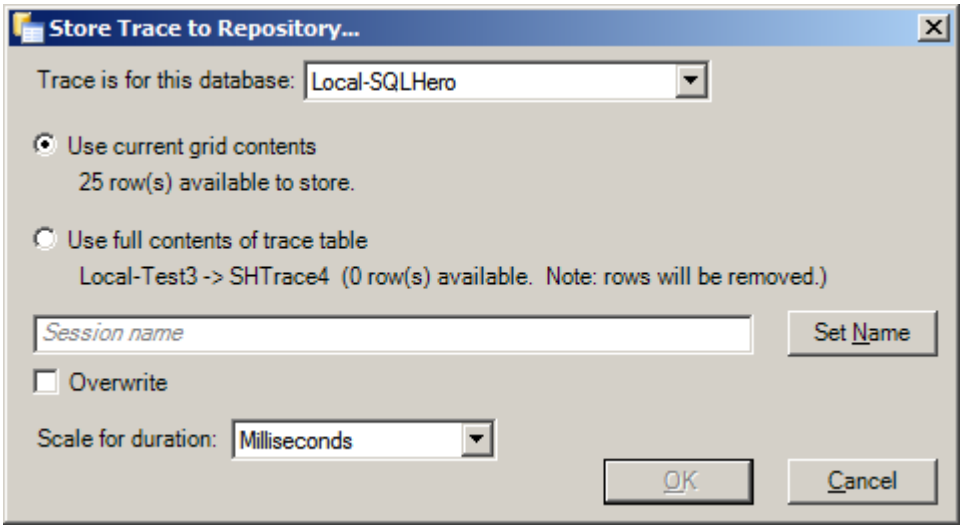


Trace Repository and Reports

Now that you have collected trace data, you can upload this into the SQL-Hero repository, if you have one installed and have your client pointed at it. (For details on this, see the "Installing SQL-Hero" whitepaper.) Back to the toolbar:

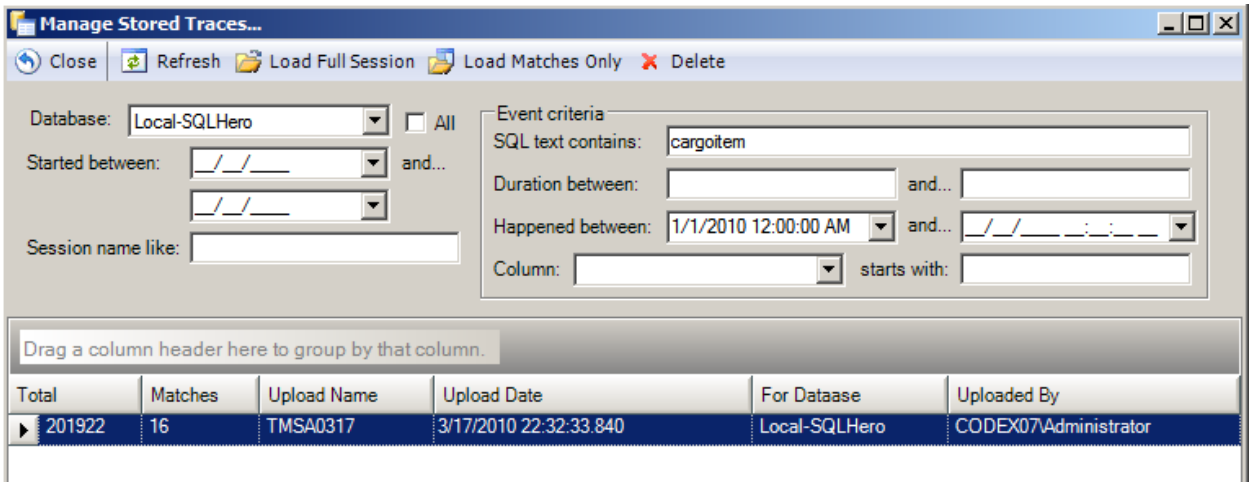


The "Save To Archive" command ( ) lets you save data to the repository either based on what is currently present in the grid, or directly from a trace table where trace data is stored:

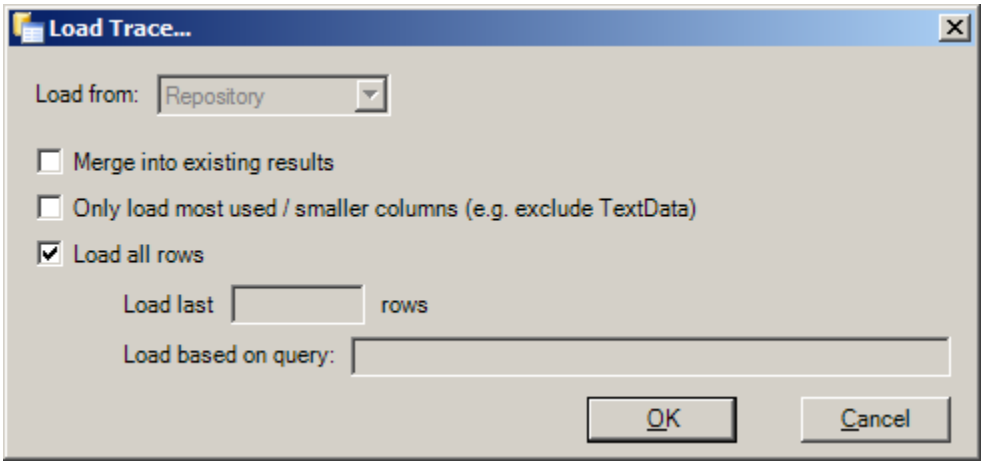


If you load from a trace table, its contents will be emptied as the rows are moved into the repository. Data loaded into the repository must have a label which is the upload session name. If the durations being uploaded are in microseconds, you should specify that here as well.

With data now in the repository, the Manage Archives command ( ) invokes a search screen which can be used to query the repository. You can use any of the provided search criteria to locate individual commands that were recorded (based on the "Event criteria" panel), or sessions themselves. In the example below, we've searched for stored trace records that were for the Local-SQLHero database, where the TextData contains the word "cargoitem" (case insensitive), and the commands' end date was 1/1/2010 or later. After issuing the search (Refresh), 16 commands met this criteria in one session that contains a total of 201,922 commands.



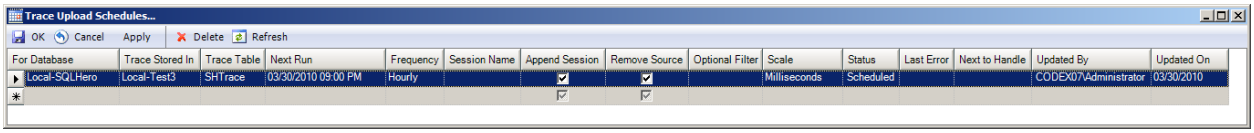
Clicking on "Load Full Session" would bring back all 201,922 rows into the trace grid. Clicking on "Load Matches Only" would bring back the 16 rows. In fact, after electing to load data from the repository, the Load Trace dialog lets you fine tune what's loaded even further:



In this case, checking "Merge into existing results" would allow you to accumulate trace rows into the grid – if unchecked, the loaded data replaces the current grid contents.

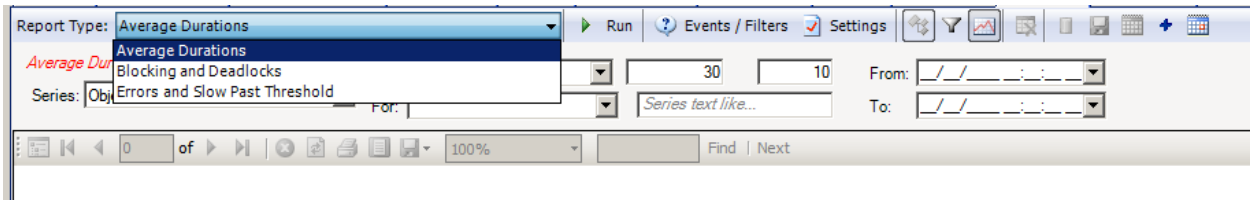
Instead of forcing you to load traces into the repository manually, SQL-Hero offers the option of automatically uploading trace data, on a schedule. The "Manage Upload Schedules" command ( ) lets you configure this. The typical approach here would be to configure a SQL Trace to run on an on-going

basis, perhaps recording long running commands, commands that result in errors, timeouts, deadlocks, etc. – captured into a trace table. On the upload schedules screen, we spell out where to find that trace table, the upload frequency, and other details about the upload:

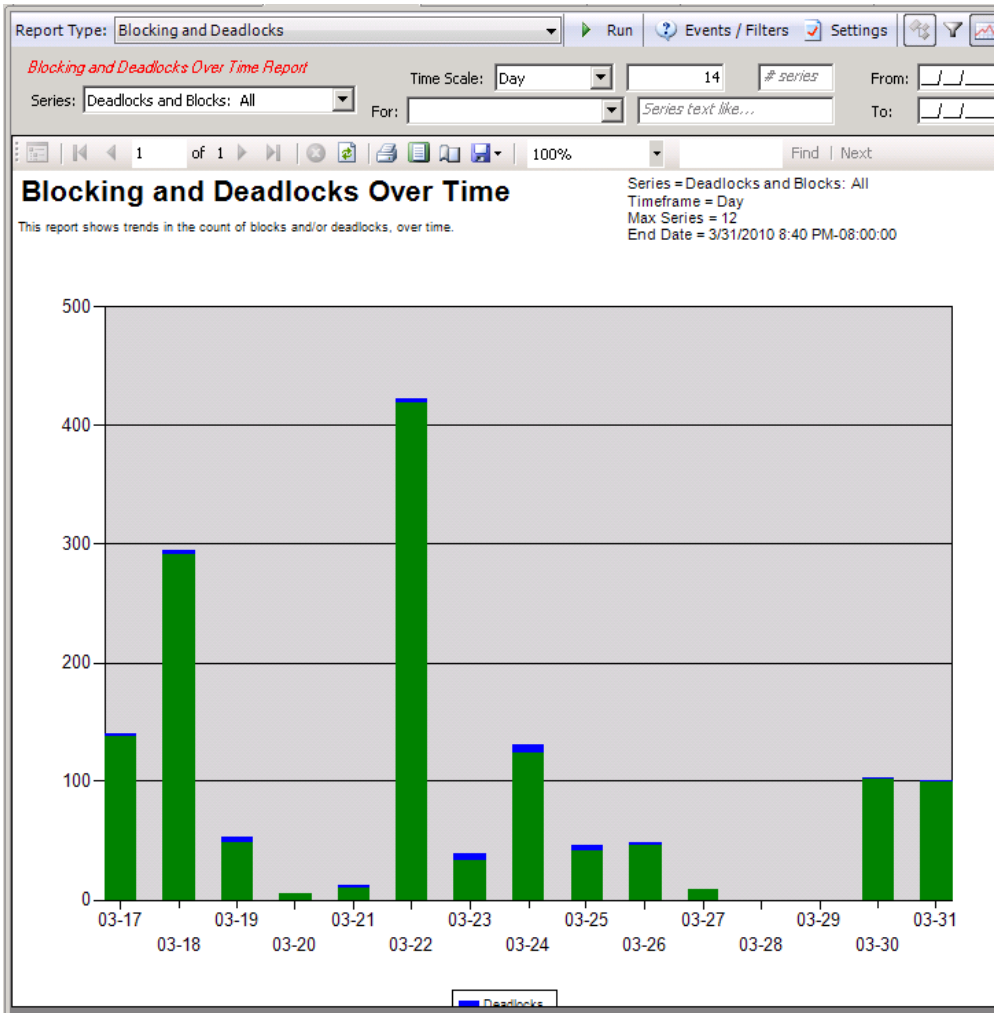


In this example, we've recorded events for the Local-SQLHero database in the database Local-Test3, in a trace table called SHTrace. The SQL-Hero service will on an hourly basis look for new events to move into the repository. "Append Session" is checked, so a single session will grow continuously as events are added. "Remove Source" is checked so rows in SHTrace will be deleted as they are moved into the repository. Leaving "Remove Source" unchecked would mean that trace data continues to reside in the SHTrace table, in addition to the repository. An optional filter can be applied - e.g. "Duration > 1000" would only upload events where the Duration is greater than 1000.

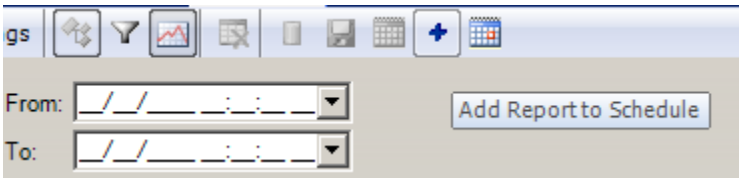
Now with trace data available in the repository, some reporting options are possible. Switching to the "Report View" () changes the available toolbar options:



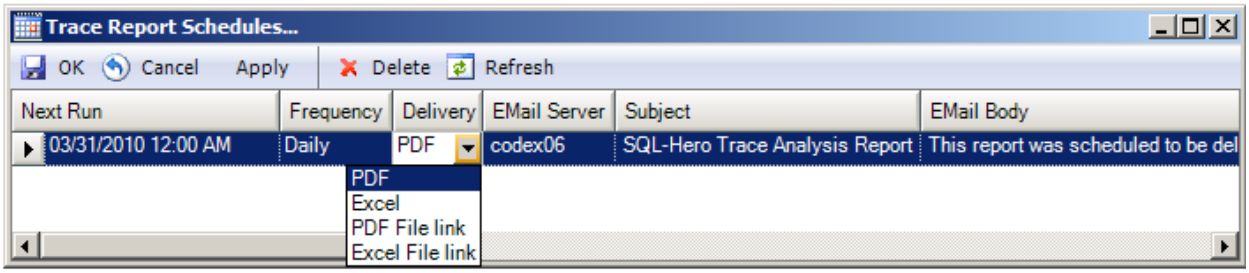
The report type choices here change the available report parameters. In this example, we've chosen to run a report that will show captured Block and Deadlock counts over the last 14 days, across all databases for which there is data in the repository:



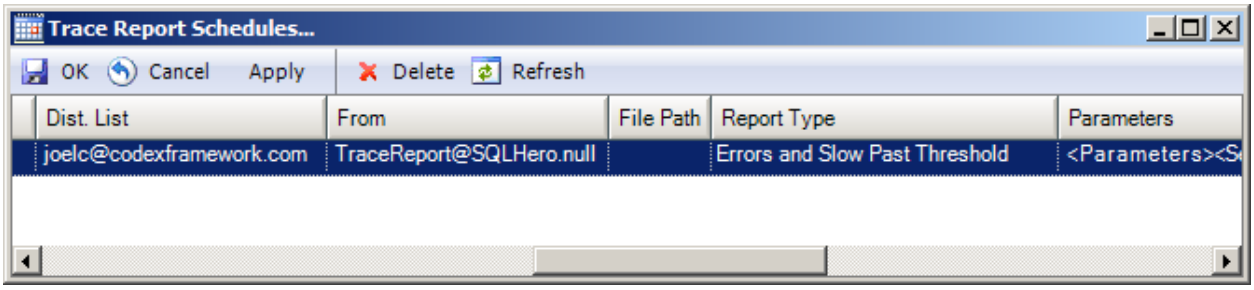
Let's say now that you want to deliver this report to a distribution list. You can schedule analysis reports to run using the "Add Report to Schedule" command ( ):



Adding a report implies you're scheduling the same report that would run if you clicked the "Run Report" button (Run). This brings up the report schedule window:



You can pick a run time of day, a frequency, a delivery type, and so on. Delivery types can be a PDF or Excel file attachment, or the PDF / Excel file can be stored on a server with your generated email containing a link to them. You must specify an email server (SMTP), an email subject, and an email body. If the body appears to be HTML, the email will be sent in HTML format.



The distribution list is a list of email addresses which can be ";" delimited. The "File Path" is only needed if you wish to distribute the report via a link. You may include the following substitution strings in your email body:

<!DateTime/>	Current Date (UTC)
<!FullFileName/>	The full local file path of the generated Excel/PDF file, on the machine the SQL-Hero Windows service is running.
<!FileName/>	Just the filename portion of the generated Excel/PDF file.

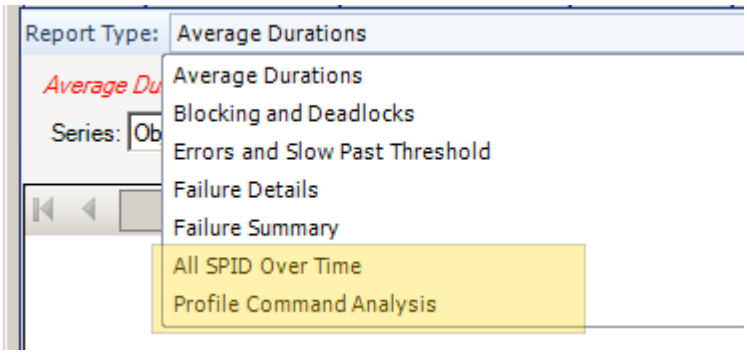
As such, you could for example use the following for an email body:

```
This report was scheduled to be delivered to you and has data current as of <!DateTime/> GMT.  <A HREF="\\codex06\reportshare\<!FileName/>">Click here to view report.</A>
```

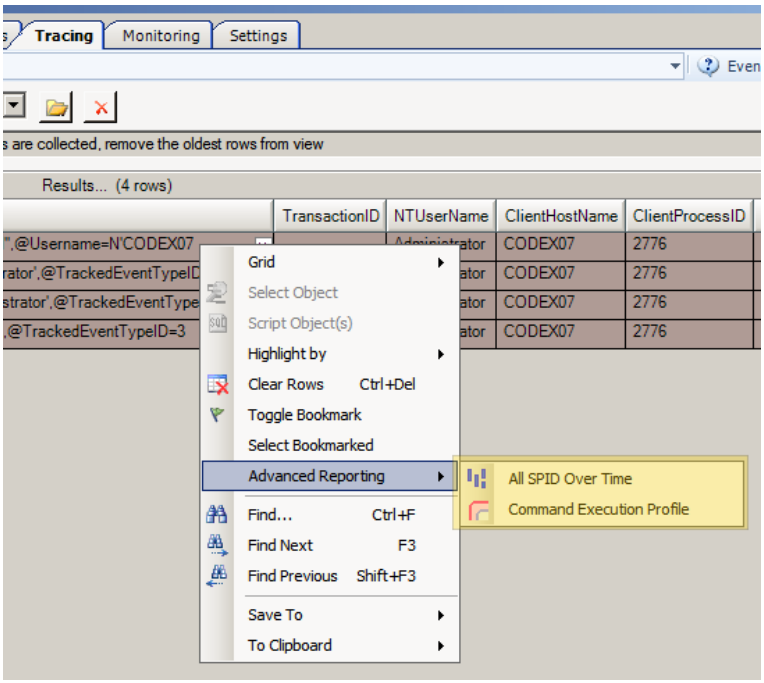
If your schedule's file path is set to "c:\reportpath", this assumes you've shared that directory as "reportshare" on the machine "codex06".

Advanced Reporting

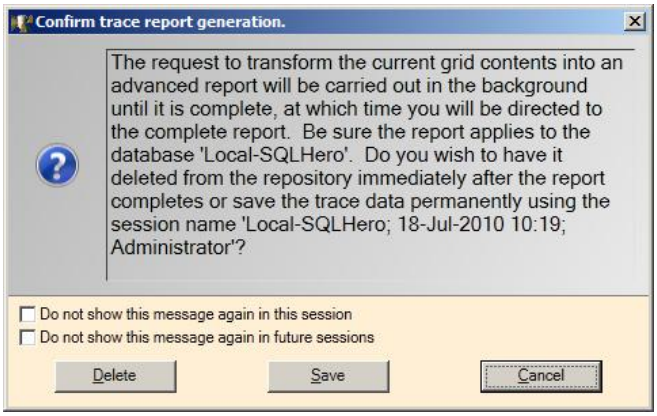
There are two "special" types of trace reports available in SQL-Hero currently. These are found under the report type list:



These reports cannot be scheduled for delivery by email currently, and work exclusively with trace data saved in the repository. We've made it easy, however, to get to both of these reports in as few as two mouse clicks, from trace data that's sitting in the detail grid. To do this, use the context menu on the trace grid and pick Advanced Reporting:



By trying to run these reports using this approach, you'll be asked whether you want to permanently save the grid data in the repository (Save) or whether you want to save it to the repository only long enough to run the report (Delete).

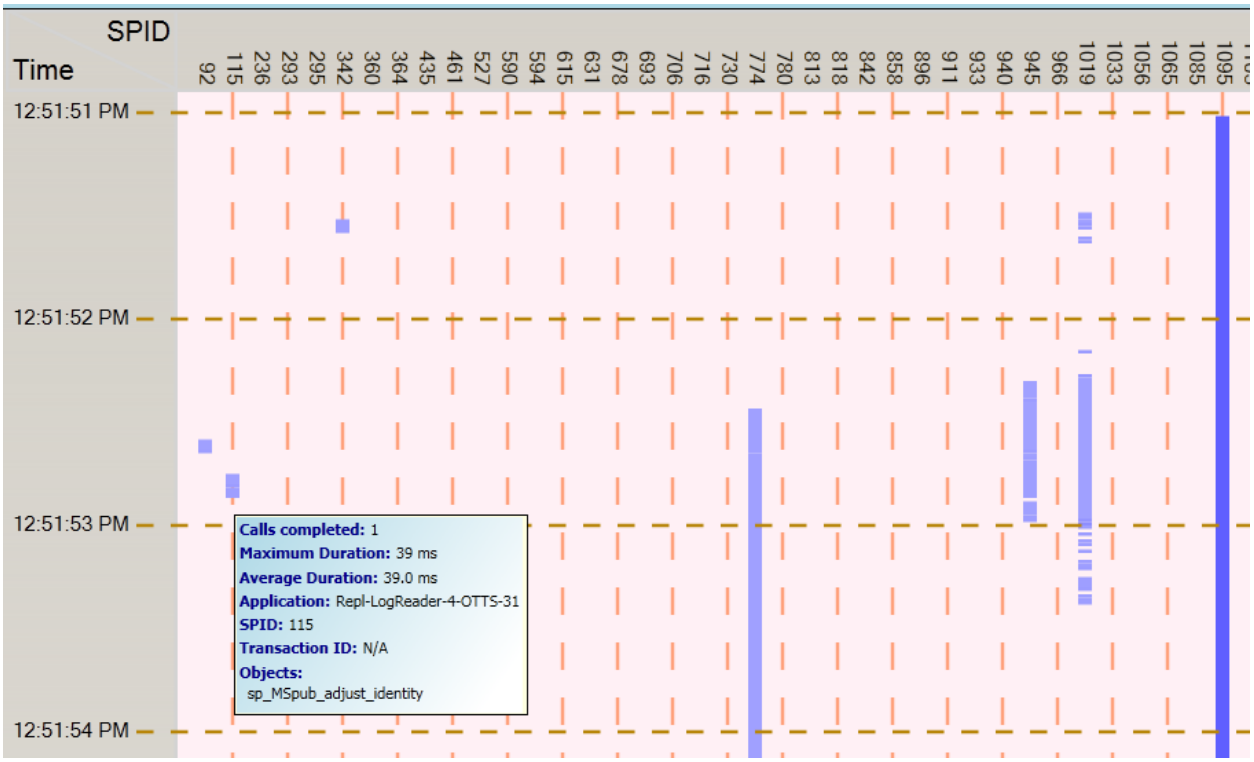


Regardless of whether you run the report using the grid's context menu or pick it from the report list, you will see a set of parameters, much as with all other report types. The following table explains the meaning behind each parameter field:

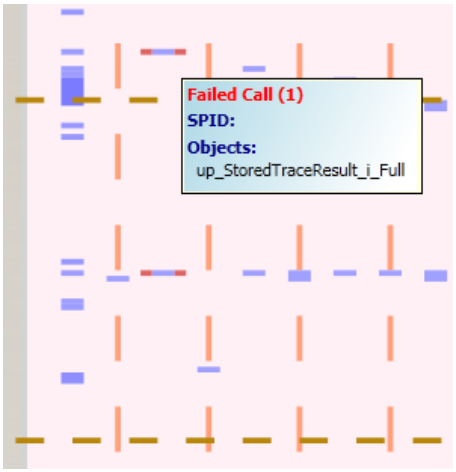
Report	Parameter	Description
All SPID Over Time	Type (Time frame)	Affects the length of the report by showing seconds, minutes or hours at fixed intervals
	Type (Coloring)	Determines how the color is selected to portray activity
	For (Database)	Filter events to those from a specific database
	Max Length	Effectively controls the height of the report down the screen (acts as a limit to prevent run-away rendering)
	From / To	Filter events to only those that occurred within a specific timeframe
	Session Name	Filter events to those which were captured in a specific named trace upload
	Exclude SPID List	This can be a comma separated list of SPID numbers that should be *excluded* from the report
Profile Command Analysis	Type (Level)	Controls the level of summary. "Procedure Summary" shows each procedure name listed once each where the width of bars reflects percentage of overall duration. "Procedure By Order" shows each procedure as many times as they are called, in the order they are called, where the width of bars reflects each call's percentage of the overall duration. "Statement Summary" shows the nesting of calls into calls such as triggers, UDF's, etc. but no further detail within calls; rectangle widths reflect relative duration with respect to the parent nesting level. "Statement Detail" shows the nesting of calls, and every statement is called out as individual rectangles. The "Statement" types require that you've captured a trace that includes statement level events.

	Type (Coloring)	Determines how the color is selected to portray activity
	Statement Spacing	Effectively controls the height of the report down the screen
	Session Name	Filter events to those which were captured in a specific named trace upload
	SPID	Filter events to one specific SPID number
	From / To	Filter events to only those that occurred within a specific timeframe
	Omit Whitespace	When checked, time that was *not* spent within database calls is *not* shown in the report; conversely, when not checked, this time is shown as the separation of rectangles in the report. This option is only applicable when the type is “Statement Summary” or “Statement Detail.”

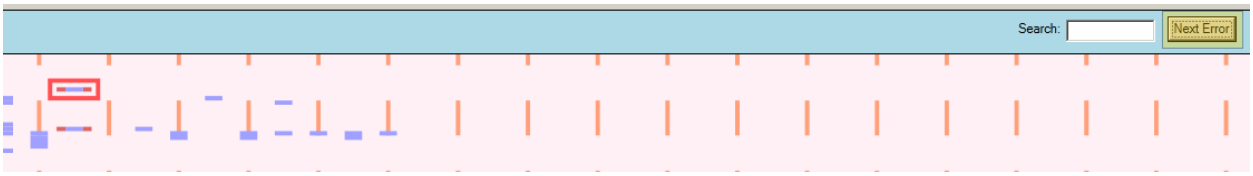
To understand what you see when you run these reports, let’s consider this example:



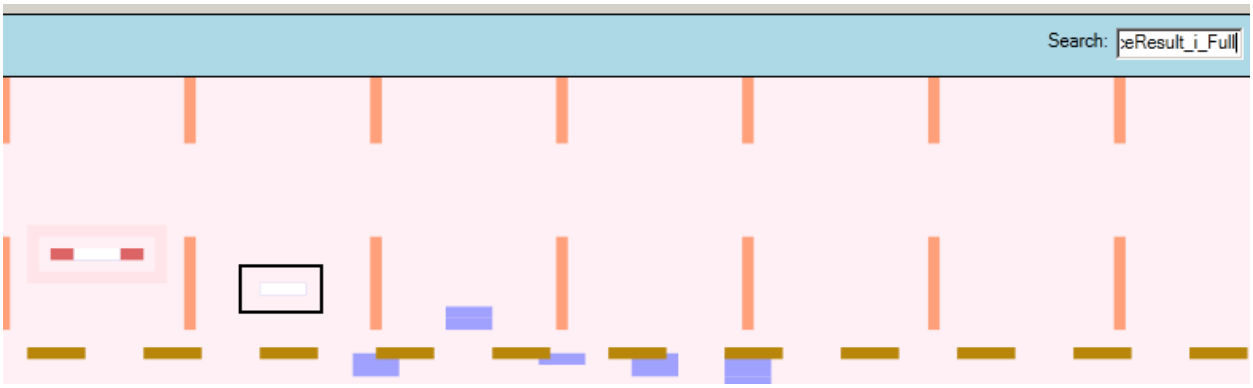
We’re seeing that at 12:51:53 PM, three different connections have activity going on concurrently. The SPID numbers are provided both at the top, and on the tool-tips provided for each activity bar. The coloring option used here is showing that on SPID 1095, a lot of calls are completing whereas on SPID 774, commands aren’t completing as frequently. If there are errors, blocks or deadlocks recorded in the same trace data as that being displayed, these issues are highlighted as red bands that surround the main activity bars, as illustrated here:



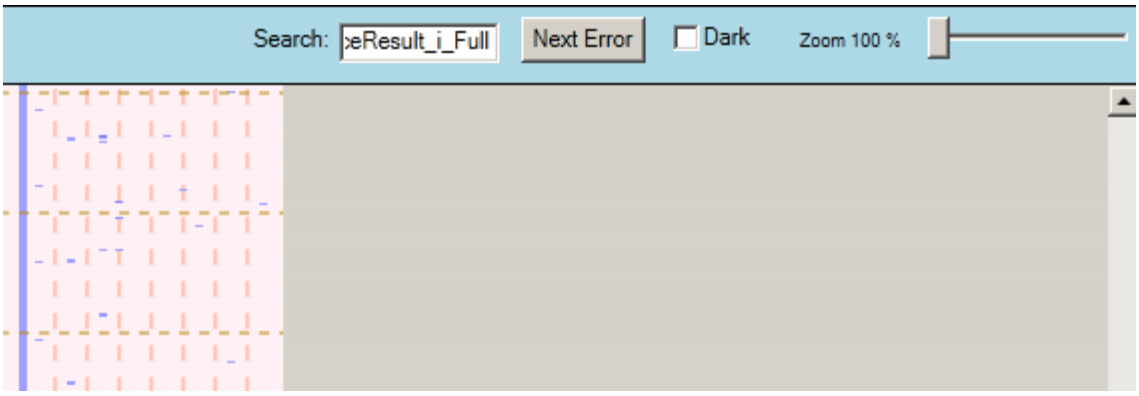
You can quickly navigate to such errors using the “Next Error” button – these errors get highlighted by a red rectangle that changes color.



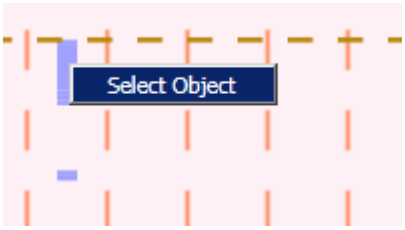
You can also search for specific object calls, by name using the “Search” text box – it will again use color changes to highlight all objects that match the search criteria:



As is the case for both advanced reports, the Zoom slider can be used to zoom in and out of the report space:

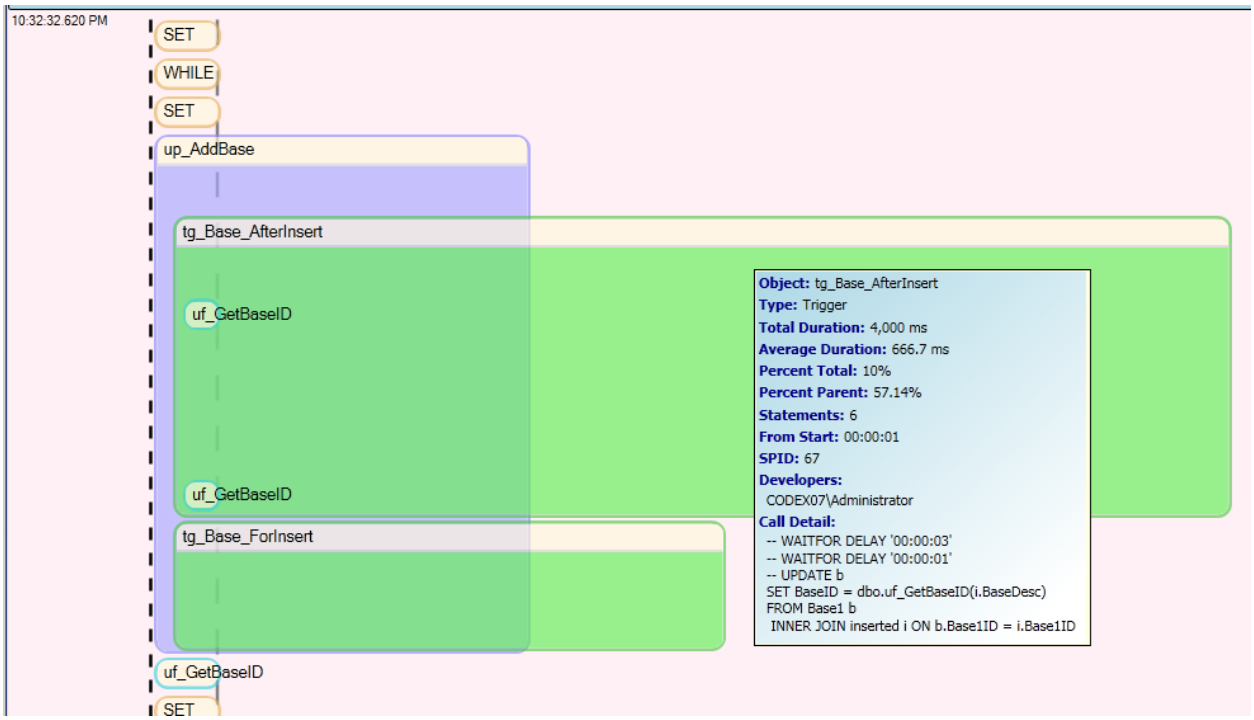


Note there's a context menu available on activity bars, to allow you to instantly jump to the object for viewing its contents, using the SQL Editor tool:

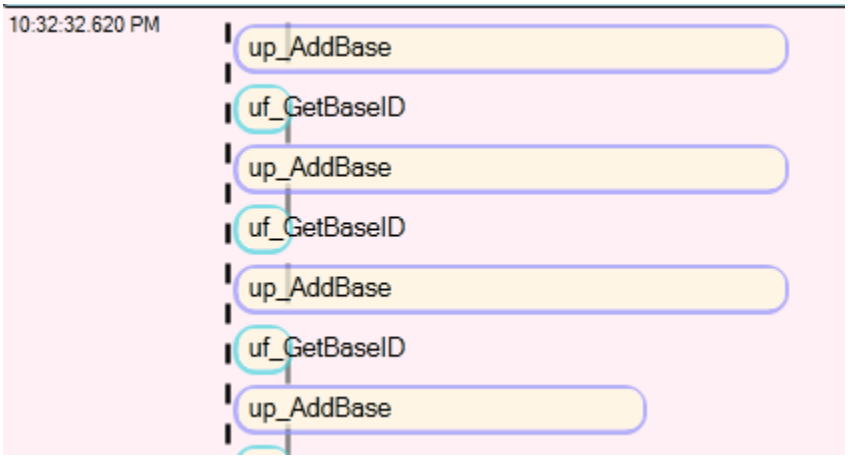


The next advanced report is the Profile Command Analysis. Let's consider the following example to understand what the report can tell us. Here we see there are some statements which are occurring in a SQL batch (SET, WHILE and another SET). Following that is a call to a stored procedure, up_AddBase. A small bit of time elapses within that procedure before a trigger fires, tg_Base_AfterInsert. A longer amount of time elapses as that trigger executes. tg_Base_AfterInsert also makes two calls to a scalar UDF called uf_GetBaseID. Neither of the calls to the UDF take much time at all. After tg_Base_AfterInsert completes, another trigger fires: tg_Base_ForInsert. After that trigger ends, the stored procedure ends as well. More activity continues in the SQL batch after the procedure ends. The tool-tip shown is for the tg_Base_AfterInsert trigger and from it we learn that it ran for 4 seconds. That 4 seconds represents 10% of the total trace capture and 57% of the current call to up_AddBase. It began 1 second from the start of the trace data available. The developers who worked on the object up_AddBase include only one: CODEX07\Administrator. Some statement-level detail is presented as well.

The fact that tg_Base_AfterInsert took 57% of the time of its parent up_AddBase also gets reflected in the width of its rectangle. The second and third dashed lines of the report represent 0% and 100% respectively.

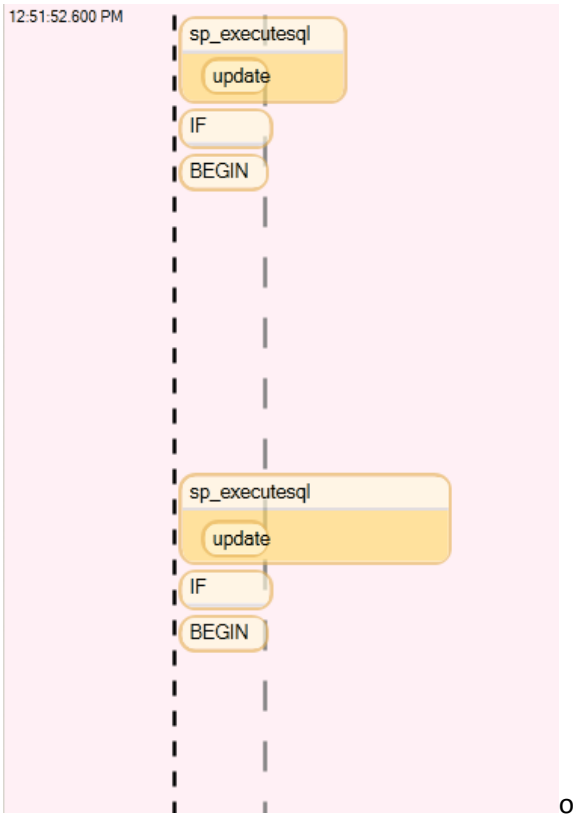


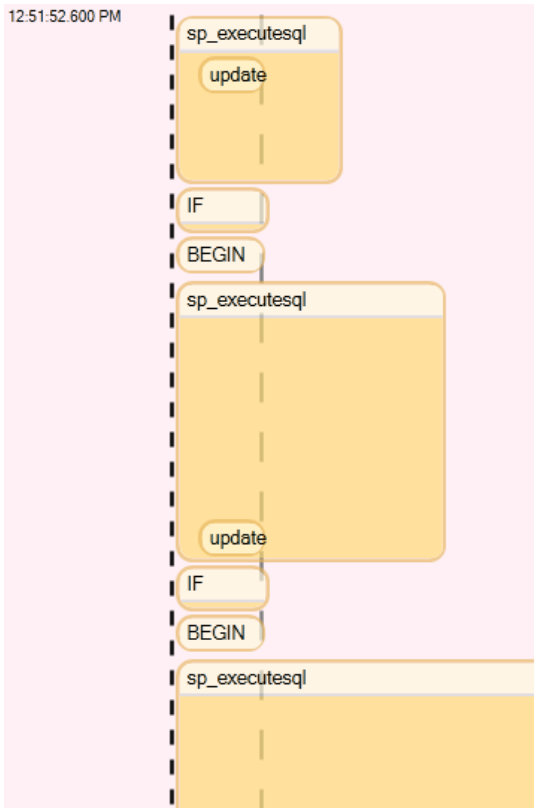
This report was run with the type of “Statement Summary” and “Object Type” (for coloring). If it were run with the type of “Procedure By Order,” you’d see this:



Notice that all detail has been stripped out and you’re only looking at the calls originating from the main SQL batch.

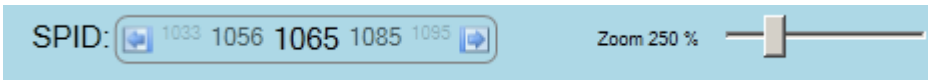
The “Omit Whitespace” checkbox is important since you may see things like this:



or:


The first example does include whitespace and it tells us a lot: there are time gaps between calls to sp_executesql in this example. However if there are very long gaps, omitting whitespace can be helpful to avoid having to "hunt" for activity by scrolling. The "Statement Spacing" setting also becomes important since it represents the total length that can be allocated, and if whitespace is included, it is counted as part of the report length, explaining why the relative position of the "update" statement in the second call to sp_executesql is different between these two views.

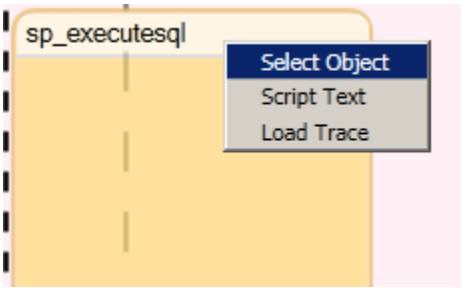
This report presents all data for *one SPID at a time*. If there are multiple SPID's involved for a given trace capture, the available SPID numbers are presented in a rolodex format at the top of the report:



And of course the Zoom slider is available as well to control the scaling of the report, in turn letting you see more data or more detail.

Calls that returned in error are always highlighted in red, regardless of the selected coloring scheme (here, tg_Base_AfterInsert had failed):

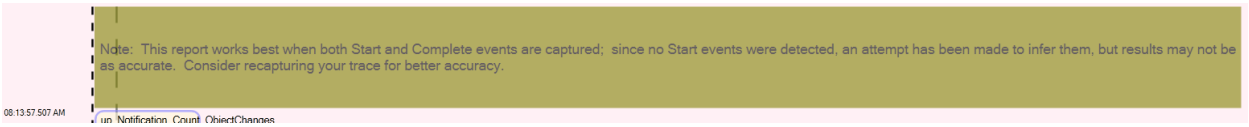
As with the SPID Over Time report, a context menu is available on report objects:



The "Script Text" option presents the actual SQL statement in the SQL Editor, ready for re-execution. The "Load Trace" option invokes the Manage Stored Trace screen, defaulting search parameters that isolate the particular statement fairly well, letting you load the raw data into the trace grid, as it would have looked during its original capture.

It's also important to know that if you capture a trace session where you have a completion type of event with no corresponding start event, the Profile Command Analysis report will warn you that there are some events which will not be included in the report. The report essentially looks for the first paired start / end and anything prior to that will be excluded:

Similar rules apply at the end of the report where a start is captured without a paired completion. Also, the Profile Command Analysis report works best when a trace has been used which includes both start and completion event types. If you only have completion events, it will still work, but may not be as accurate, since durations are used to try and infer where the start event would have been. When this situation is detected, it's provided as a warning at the start of the report area:



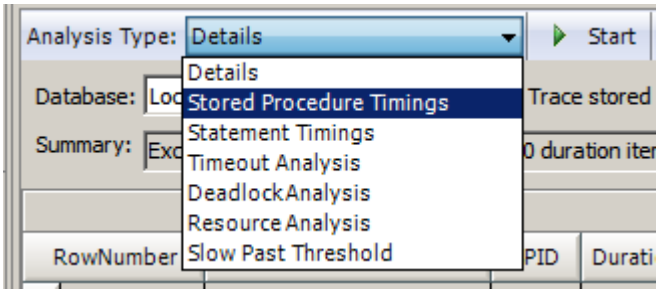
The power of this report can be appreciated by the fact we can do the following:

- Look for "wide bars" very easily, which represent relatively slow executors
- Understand where there were gaps or delays – which when combined with the SPID Over Time report allows one to see the context of activities
- See things you'd never see in a regular SQL Profiler trace: for example, developers who've modified the object

Furthermore, since we're working with data saved in the repository, one can fairly easily communicate issues within a team. (A future release of SQL-Hero will allow you to run these reports by using a small "report parameter file" which can be shared among team members.)

Trace Grid Reports

Returning to trace detail data that's already loaded in the detail grid, there are a number of analysis reports specific to what's present in the grid itself:



"Stored procedure timings" summarizes RPC completions in a way that lets you analyze average, minimum and maximum durations, among other things:

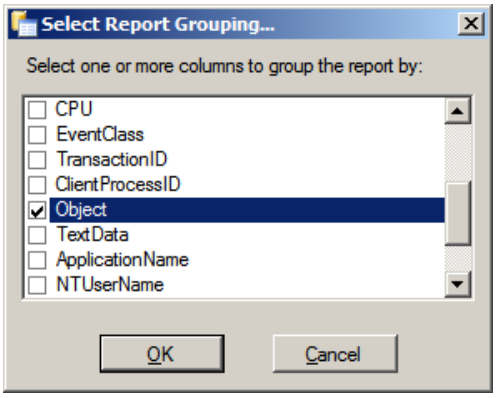
Results... (134 rows)

AvgDuration	Object	MaxDuration	MinDuration	StDevDuration	Count	AvgReads	AvgWrites	AvgCPU
780.8	up_Contact_i	976.0	0.0	436.48	5	28.0	0.8	0.0
780.8	up_EntityContact_i	976.0	0.0	436.48	5	32.8	1.2	3.0
650.67	up_EDIJobItemLog_s_ByKey	976.0	0.0	563.49	3	5.0	0.0	0.0
650.67	up_ReservationVersion_s_ByKey	976.0	0.0	563.49	3	3.0	0.0	0.0
618.46	up_FinParamType_s_ForList	976.0	0.0	471.41	202	52.0	0.0	0.39
542.22	up_LocationTile_s_ByTileNumber	976.0	0.0	514.4	9	5.0	0.0	0.0
532.36	up_ContactPhone_s_ByKey	976.0	0.0	497.42	22	19.45	0.0	0.73

"Statement timings" lets you analyze statement timings within stored procedures, providing similar information.

"Timeout analysis" and "Deadlock analysis" aim to show the call chain that led to timeout and deadlock situations. These require that you've captured both statement start and completion events.

The “Resource analysis” report asks you to select one or more columns which are grouped on:



In this example we’re grouping by object name and we would see:

Results... (95 rows)

Object	Count	PercentCount	TotalReadWriteCPU	PercentReadWriteCPU	AverageReadWriteCPU	MaxReadWriteCPU	TotalDuration	PercentDuration	AverageDuration	MaxDuration	DurationCountRatio	DurationElapsedRati
up_GLAccountNumber_s_ByKey	2828	1.41	5,392.0	0.03	1.91	29.0	443,286.0	0.13	156.75	65,429.0	0.09	3.88
uf_GetOperationalCustomer	1650	0.82	1,661.0	0.01	1.01	18.0	20,496.0	0.01	12.42	976.0	0.01	0.18
up_Location_s_ForList	432	0.21	19,614.0	0.12	45.4	375.0	4,158,175.0	1.2	9,625.41	134,765.0	5.58	36.42

Some important columns here include the count of calls for each object, the percent of all calls for that object, the total accumulated duration, the percent duration (higher numbers imply more resource use), average duration, duration-to-count ratio (higher numbers point to an object that’s taken more time to execute relative to the number of times it has been called).

The “Slow Past Threshold” report allows you to look for “outliers”: specific instances of events that take longer than would be typical based on the object’s average duration.