White Paper: ibec_StartTraceSession

Identify unusually high database load: take two

PDF Download

As we explained earlier, IBEBlock is a set of DDL, DML and other statements that are executed on the server and on the client side, and which include some specific constructions applicable only in IBExpert or IBEScript (excluding the free versions of these products), independent of the database server version.

IBEBlock is an extremely complex and capable piece of software, which can help you to increase your productivity and save both time and money spent on your projects.

Imagine you have a database under heavy load, and you would like to identify which attachment is responsible for that. So, you can execute a select on the MON$ tables which will show the most active connections, and if one of the connections has an unusually high load compared to all others, this connection should be traced into a log.

Trace and audit

If you are already a Firebird user, you have probably heard about Firebird's Trace and Audit services, initially developed from the TraceAPI contributed by Nickolay Samofatov that he developed for Red Soft Database. This feature enables you to look inside the Firebird engine and log various events for real-time analysis.

Two different kinds of traces can be performed: user traces and a system audit.

There are three general cases for use:

- Constant audit of engine activity
- Interactive trace of specified activity in specified database(s).
- Engine activity logging for a specified period of time (be it hours or a whole day) for later analysis.

For a more detailed explanation, please refer to https://www.ibexpert.net/ibe/pmwiki.php?n=Doc.TraceAndAudit.

Nevertheless, an audit/trace does not replace the monitoring tables, because with the monitoring tables the user can query a snapshot of the current state of a database, whereas an audit/trace can be compared to a sequential stream of logged statements. For example, if you wish to know the number of currently active transactions inside a database, simply query the MON$TRANSACTIONS table. With the audit/trace facility, this probably can only be done with advanced data analysis tasks on the log data stream (if at all).

IBExpert version 2022.06.10 introduces ibec_StartTraceSession (among other useful new features and improvements). Please first check the for reference.

IBExpert version 2022.11.14 includes an improved ibec_StartTraceSession.

ibec_StartTraceSession starts a trace session using the Firebird Services API and writes trace data into a file. Additional filtering of trace items is possible via a callback block.

Syntax

```
function ibec_StartTraceSession(ConnectParams : string; TraceConfig : 
string; OutputFile : string; FilterBlock : variant; ProgressBlock : variant)
: variant;
```

ConnectParams - list of connection params and some additional options delimited with semicolon:

- Server - server name including the port number if necessary.
- User - user name.
- Password – password.
- ClientLib - path to a client library DLL.
- MaxFileSize - maximum size of the result file in megabytes. If not specified the result file size will be limited to 100 megabytes.
- StopAfter - determines the duration of the trace session in minutes. The default value is 60 minutes.
- IncludeStatistics - some statistical data (total and max. per second, number of processed lines and events) will be written at the beginning of the trace data.
- AppendToExisting | AppendMode - if specified, trace data will be added to the existing file (if it exists).
- ParseData option. If specified, the event text will be parsed and the results will be passed to the filter block. In that case the first input parameter of the filter block will contain an array of parsed data instead of a single value, as with the without ParseData option. The very first value contains plain (not parsed) event item text in both cases. To access items of the parsed data array there are some constants defined in the TraceAPI namespace (see example below).
- NoWait option. If specified, IBEBlock will continue execution just after starting a trace session, so it is possible to start and control several trace sessions from a single block. In this case it is necessary to create a loop which will be executed while at least one session is alive (see example below). Otherwise, all trace sessions will be terminated at the end of the block execution.
- SessionName option. Assigns a user-defined name to a trace session.

TraceConfig - Firebird trace configuration.

OutputFile - name of a result file which will contain the trace data.

FilterBlock - a callback block which performs additional filtering of trace items if necessary. This block is called for every trace item and accepts a trace item text as an input parameter (*EventSource*). The output parameter *NeedSkip* determines whether the trace item should be skipped i.e. not included into the result file.

Support of a second input parameter has been added to the filter block.

- *SessionData[0]* or *SessionData['SessionObject']* - pointer to the trace session object.
- *SessionData[1]* or *SessionData['SessionName']* - name of the trace session.
- *SessionData[2]* or *SessionData['SessionObject']* - internal Firebird ID of the trace session.

If no additional filtering is needed pass NULL or an empty string as the *FilterBlock* value.

ProgressBlock - a callback block that is intended to display the tracing progress. It receives array of some statistical data:

- *Data[0]* (or *Data['LinesProcessed']*) - number of processed lines.
- *Data[1]* (or *Data['EventsProcessed']*) - number of processed trace items/events.
- *Data[2]* (or *Data['EventsMatched']*) - number of trace items written to the result file.
- *Data[3]* (or *Data['OutputSize']*) - current size of the result file in bytes.

The output parameter Threshold determines a delay in milliseconds before the next call of ProgressBlock.

Support of a second input parameter has been added to the progress block, exactly as with the filter block.

- *SessionData[0]* or *SessionData['SessionObject']* - pointer to the trace session object.
- *SessionData[1]* or *SessionData['SessionName']* - name of the trace session.
- *SessionData[2]* or *SessionData['SessionObject']* - internal Firebird ID of the trace session.

ibec_StartTraceSession returns NULL if a trace session is started successfully, otherwise it returns an error message as a result. The previous implementation of *ibec_StartTraceSession* function didn't return an error message if there was something wrong. It's fixed in the current implementation.

There is new *TraceAPI.GetActiveTraceSessions* function. By default (the Options param is NULL or contains unknown data) it returns the number of active trace sessions.

If the *Return=NamesList* option is specified, it will return a comma-delimited list of names of active trace sessions. Also, a TraceAPI namespace has been added. It contains two functions and some constants. *TraceAPI.StartTraceSession* is just an alias for the *ibec_StartTraceSession* function.

One example of usage can be the following:

```
execute ibeblock
as

  -- Filter block
  declare ibeblock filter_block (EventSource variant , SessionData variant)
  returns (
    NeedSkip boolean)
  as
  begin
    NeedSkip = EventData[@TraceAPI.TED_EVENT_TYPE] = 'START_TRANSACTION';  -
- Event type
    if (not NeedSkip) then
      NeedSkip = EventData[@TraceAPI.TED_DURATION] <= 0;
  end;
  -- Progress block
  declare ibeblock progress_block (data variant)
  returns (
    threshold integer = 1000)
  as
  begin
    for i = 0 to ibec_High(data) do
      data[i] = ibec_Cast(data[i], __typeString);

    s = ibec_Format('| %15.15s | %16.16s | %14.14s | %16.16s |', data[0],
data['EventsProcessed'], data['EventsMatched'], data['OutputSize']);
    ibec_ProgressEx(s, __poNoCRLF + __poReplaceLast);
  end;

begin
  sConfig = ibec_LoadFromFile('C:\temp\trace.conf');
  sOutFile = 'C:\Temp\tracedata.txt';
  sServer = 'myserver/3044’;
  sClientLib = '"D:\FB4_CLIENT\fbclient.dll"';


  Res = @TraceAPI.StartTraceSession('Server=' || sServer || ';
                                    User=SYSDBA; Password=masterkey;
                                    ClientLib=' || sClientLib || ';
                                    MaxFileSize=10; StopAfter=10;
IncludeStatistics; ParseData; NoWait',
                               :sConfig,
                               :sOutFile,
                               filter_block,
                               progress_block);
  if (Res is not null) then  -- Error caused
    ibec_ShowMessage('ERROR: ' || Res);
  else
  begin
    iTime1 = ibec_GetTickCount();
    ibec_Progress('Trace session started');
    ibec_Progress('+-----------------+------------------+----------------+--
-----------------+');
    ibec_Progress('| Lines processed | Events processed | Events matched |
Output file size |');
    ibec_Progress('+-----------------+------------------+----------------+--
-----------------+');
  end;

  while (@TraceAPI.GetActiveTraceSessions('Return=NamesList') <> '') do
  --while (@TraceAPI.GetActiveTraceSessions('') > 0) do
  begin
    -- ibec_Pause should be used instead of ibec_Sleep here
    ibec_Pause(2000);
  end;
  iTime2 = ibec_GetTickCount();
  sSessionTime = ibec_FormatFloat('0.00', (iTime2 - iTime1) / 1000 / 60);

  ibec_Progress('');
  ibec_Progress('+----------------+-----------------+---------------+-----
--------------+');
  ibec_Progress('Session was active ' || sSessionTime || ' minutes');
  ibec_Progress('That''s all, folks!');
end;
```

Let’s now return to our original task, to identify which connection generates an unusually high database load.

If you are an experienced user, you can try a select on any combination of MON$ tables to get interesting attachment IDs as results, and you can later change your query.

My approach here will be to store the current state of page reads/writes in a new table, for example, every minute, and check which attachments did the largest amount of reads/writes within the last 10 minutes and is not already actively traced.

Also, if the total number of reads+writes is below 100000 within last 10 minutes, it should be ignored.

I have created a table in the database to hold the data collected every minute:

```
CREATE TABLE IBE$MON (
    ATTACHMENTS_ID  BIGINT NOT NULL,
    STATEMENT_ID    BIGINT NOT NULL,
    TRANSACT_ID     BIGINT,
    OPERATIONS      BIGINT,
    TS              TIMESTAMP DEFAULT current_timestamp
);
```

The following indices will help with our query later:

```
CREATE INDEX IBE$MON_IDX1 ON IBE$MON (ATTACHMENTS_ID);
CREATE INDEX IBE$MON_IDX2 ON IBE$MON (TS);
```

Since we have the new version of *ibec_StartTraceSession*, it is possible to start and control several trace sessions from a single block, and I have eliminated separate scripts so that everything needed is in a single IBEBlock:

```
execute ibeblock
as

begin
  sConfig = ibec_LoadFromFile('c:\temp\trace.conf');
  sOutFile = 'c:\Temp\tracedata.txt';
  sServer = 'localhost';
  sClientLib = '"C:\Users\Titan\Desktop\output_Win32\fbclient.dll"';

  sCRLF = ibec_CRLF();
  FBSrc  = 
ibec_CreateConnection(__ctFirebird,'DBName="localhost:C:\db\mydb.fdb";
                 ClientLib=' || sClientLib || ';
                 user=SYSDBA; password=masterkey; names=UTF8;
sqldialect=3');

  Res = @TraceAPI.StartTraceSession('Server=' || sServer || ';
                                     User=SYSDBA; Password=masterkey;
                                     ClientLib=' || sClientLib || ';
                                     MaxFileSize=10; StopAfter=10;
IncludeStatistics; ParseData; NoWait',
                                :sConfig,
                                :sOutFile,
                                null,
                                null);

  if (Res is not null) then  -- Error caused
    ibec_ShowMessage('ERROR: ' || Res);

  while (@TraceAPI.GetActiveTraceSessions('Return=NamesList') <> '') do
  begin

    ibec_UseConnection(FBSrc);

     --snapshot of mon$ tables every minute (see ibec_Pause after commit)
and stored in our table for time-related purposes

      insert into ibe$mon (
        attachments_id,
        statement_id,
        transact_id,
        operations)
      select st.mon$attachment_id,
        st.mon$statement_id,
        st.mon$transaction_id,
        cast(sum(io.mon$page_reads)+
        sum(io.mon$page_writes) as integer)
      from mon$statements st
      join mon$attachments a on a.mon$attachment_id = st.mon$attachment_id
      join mon$io_stats io on (st.mon$stat_id = io.mon$stat_id)
      group by 1, 2, 3;

      COMMIT;

    -- ibec_Pause should be used instead of ibec_Sleep here
    ibec_Pause(60000);

    for select IBE$MON.attachments_id, sum(IBE$MON.operations) io from
IBE$MON
    where (IBE$MON.ts > dateadd(-10 minute to current_timestamp))
    group by IBE$MON.attachments_id
    into :att_id, :IO do
      if (IO > 10000/*or any other threshold load value*/) then begin
        sLoopConfig = 'database ' || sCRLF ||
                           '{' || sCRLF ||
                           'enabled = true ' || sCRLF ||
                           'connection_id = ' || att_id || sCRLF ||
                           'log_statement_prepare = true ' || sCRLF ||
                           'log_statement_free = true ' || sCRLF ||
                           'log_statement_start = true ' || sCRLF ||
                           'log_statement_finish = true ' || sCRLF ||
                           'time_threshold = 0 ' || sCRLF ||
                           '}';

        sLoopOutFile = 'c:\Temp\tracedata_' || att_id || '.txt';

        LoopRes = @TraceAPI.StartTraceSession('Server=' || sServer || ';
                                     User=SYSDBA; Password=masterkey;
                                     ClientLib=' || sClientLib || ';
                                     MaxFileSize=10; StopAfter=1;
IncludeStatistics; ParseData; NoWait',
                                :sLoopConfig,
                                :sLoopOutFile,
                                null,
                                null);

        --check can be ignored because it will raise error when trace file
of attachmentid is in use
        if (LoopRes is not null) then  -- Error caused
          ibec_ShowMessage('Loop ERROR: ' || LoopRes || ' Attachment id: '
|| att_id);

      end;
    COMMIT;

  end;
  ibec_CloseConnection(FBSrc);
end;
```

You can save this IBEBlock to file and then execute this script as follows:

```
ibescript.exe ibeTraceLoad.sql
```

If you compare this IBEBlock script to the one at beginning of this document, you will notice that I constructed second trace options in code, instead of using the trace config file as I have done for the server instance:

```
sConfig = ibec_LoadFromFile('C:\temp\trace.conf');
```

in comparison to:

```
sConfig = ''database '' || sCRLF ||
                         ''{'' || sCRLF ||
                         ''enabled = true '' || sCRLF ||
                         ''connection_id = '' || att_id || sCRLF ||
                         ''log_statement_prepare = true '' || sCRLF ||
                         ''log_statement_free = true '' || sCRLF ||
                         ''log_statement_start = true '' || sCRLF ||
                         ''log_statement_finish = true '' || sCRLF ||
                         ''time_threshold = 0 '' || sCRLF ||
                         ''}'';
```

However, you can use your preferred method of specifying the trace config.

In a nutshell, our main script here, *ibeTraceLoad.sql*, will check the history of the MON$ tables from our table for 10 minutes, and if the threshold value is exceeded, it will execute trace sessions, even in parallel if there is more than one connection generating an abnormal load on the database.

Please note that *ibec_StartTraceSession* is run for 10 minutes, but that value is adjustable for your own needs.

I decided to check for high load connections once per minute, but if the trace session is already in progress, it will skip it. Also, I decided to use multiple files to output the trace results. For each attachment generating abnormal load there is a separate trace log, but you can use a single file if you prefer.