Article
David Loveluck · Aug 27, 2019 28m read

Open Exchange

APM – Monitoring SQL Query Performance

Since Caché 2017 the SQL engine has included new set of statistics. These record the number of times a query is executed and the time it takes to run.

This is a gold mine for anyone monitoring and trying to optimize the performance of an application that includes many SQL statements but it isn ’ t as easy to access the data as some people want.

This article and the associated sample code explains how to use this information and how to routinely extract a summary of daily statistics and keep a historic record of the SQL performance of your application.

What is recorded?

Every time an SQL statement is executed the time taken is recorded. This is very light weight and you can ’ t turn it off. The data includes the number of times a query has been executed in the day and the total time taken by the query.

To minimize cost, the statistics are kept in memory. Each job writes the statistics to disks periodically and a task “ Update SQL query statistics ” puts them all in the stats table. The task is usually scheduled to run hourly, but can be fired off manually, if you want to see the statistics while testing a query, but that whole process requires a little patience.

Warning: In InterSystems IRIS 2019 and earlier these statistics are not collected for embedded SQL in **classes or routines** that have been **deployed** using the **%Studio.Project:Deploy mechanism**. Nothing will break with the sample code, but it might fool you (it fooled me) into thinking everything was OK because nothing showed up as expensive.

How do you see the information?

You can see the list of queries in the management portal. Go to the SQL page and click on the ‘ SQL Statements ’ tab. This is good for a new query you are running and looking at; but if there are thousands of queries running that can become unmanageable.

The alternative is to use SQL to search for the queries. The information is stored in tables in the INFORMATION_SCHEMA schema. This schema has a number of tables and I have include some sample SQL queries at the end of this article.

From IRIS 2019, the statistics are stored in the STATEMENT_DAILY_STATS table. In earlier versions you need to look at ^rINDEXSQL as you can see in the example code.

When are the statistics removed?

On recent releases, it needs a combination of compiling the class or purging the query, and running

"Cleanup SQL Statement Index". But on earlier releases they seemed to be removed more often.

At a live site it is reasonable to expect that the statistics will be kept for more than a day or so, but the tables holding the statistics cannot be relied on as a long term reference source for running reports or long term analysis. On a development or test system, the chances of them getting deleted are higher.

How can you summarize the information?

I recommend extracting the data every night into permanent tables that are easier to work with when generating performance reports. There is a chance that some information is lost if classes are compiled during the day but this is unlikely to make any real difference to your analysis of slow queries.

The code below is an example of how you could extract the statistics into a daily summary for each query. The example accesses teh global directly for compatibility with older versions. It includes three short classes:

- A task that should be run every night.
- DRL.MonitorSQL is the main class that extracts and stores the data from the INFORMATIONSCHEMA tables.
- The third class DRL.MonitorSQLText is an optimization that stores the (potentially long) query text once and only stores the hash of the query in the statistics for each day.

The sample accesses ^rINDEXSQL directly so that it is compatible with older versions of IRIS, but today, I would write it with the SQL queries described below.

Tables in INFORMATIONSCHEMA schema

As well as the statistics, the tables in this schema keep track of where queries, columns, indices etc. are used. Typically the SQL statement is the starting table and it is joined on something like “ Statements.Hash=OtherTable.Statement ” .

A query for accessing these tables directly to find the most expensive queries for one day would be…

```
SELECT
  Stats.Statement,
  Loc.Location,
  Stats.Day,
  Stats.StatCount,
  Round(Stats.StatTotal,2) TotalTime,
  Round(Stats.StatVariance,2) Variance,
  Round((Stats.StatTotal/Stats.StatCount),3) AvgTime,
  SQL.Statement
FROM INFORMATION_SCHEMA.STATEMENT_DAILY_STATS Stats
LEFT JOIN INFORMATION_SCHEMA.STATEMENTS SQL on Stats.Statement=SQL.Hash
LEFT JOIN INFORMATION_SCHEMA.STATEMENT_LOCATIONS Loc on Stats.Statement=Loc.Statement
where Day='04/12/2022'
ORDER by TotalTime DESC
```

Whether you are considering setting up a more systematic process or not, I recommend that everyone with a large application that uses SQL runs this query today.

If a particular query shows up as expensive or slow, you can get the history by running

```
SELECT
Stats.Day,
Stats.StatCount,
Stats.StatTotal,
Stats.Statement
FROM INFORMATION_SCHEMA.STATEMENT_DAILY_STATS Stats
where Stats.Statement='FaWA3+/Dz0wqCyotECEe39wX2WQ='
order by Stats.Day
```

Notes on the sample extract code

The task extracts the previous day and should therefore be scheduled shortly after midnight.

You can export more historic data if it exists. To extract the last 120 days

Do ##class(DRL.MonitorSQL).Capture($h-120,$h-1)

The sample code reads the ^rIndex global directly because the earliest versions of the statistics did not expose the Data to SQL.

The variation I have included loops through all namespaces in the instance but that is not always appropriate.

How to query the extracted data

Once the data is extracted, you can find the heaviest queries by running

```
SELECT top 20
S.RunDate,S.RoutineName,S.TotalHits,S.SumpTIme,S.Hash,t.QueryText
from DRL.MonitorSQL S
left join DRL.MonitorSQLText T on S.Hash=T.Hash
where RunDate='08/25/2019'
order by SumpTime desc
```

Also if you choose the hash for an expensive query you can see the history of that query with

```
SELECT S.RunDate,S.RoutineName,S.TotalHits,S.SumpTIme,S.Hash,t.QueryText
from DRL.MonitorSQL S
left join DRL.MonitorSQLText T on S.Hash=T.Hash
where S.Hash='CgOlfRw7pGL4tYbiijYznQ84kmQ='
order by RunDate
```

Real Example

Earlier this year I looked at the most expensive queries from a live site. One query was averaging less than 6 seconds but was being called 14,000 times per day adding up to nearly 24 hours elapsed time every day. Effectively one core was fully occupied with this query. Worse still, the second query that takes an hour was a variation on the first.

RunDate	RoutineName	Total Hits	Total Time	Hash	QueryText (abbreviated)
03/16/2019		14,576	85,094	5xDSguu4PvK04se2pPiOexeh6aE=	DECLARE C

					CURSOR FOR SELECT * INTO :%col(1) , :%col(2) , :%col(3) , :%col(4) …
03/16/2019		15,552	3,326	rCQX+CKPwFR9zOplmtMhxVnQxyw=	DECLARE C CURSOR FOR SELECT * INTO :%col(1) , :%col(2) , :%col(3) , :%col(4) , …
03/16/2019		16,892	597	yW3catzQzC0KE9euvlJ+o4mDwKc=	DECLARE C CURSOR FOR SELECT * INTO :%col(1) , :%col(2) , :%col(3) , :%col(4) , :%col(5) , :%col(6) , :%col(7) ,
03/16/2019		16,664	436	giShyiqNR3K6pZEt7RWAcen55rs=	DECLARE C CURSOR FOR SELECT * , TKGROUP INTO :%col(1) , :%col(2) , :%col(3) , ..
03/16/2019		74,550	342	4ZClMPqMfyje4m9Wed0NJzxz9qw=	DECLARE C CURSOR FOR SELECT …

Table 1: Actual results from customer site showing one query taking almost 24 hours total time

Code Sample for extracting statistics daily

#Best Practices #Monitoring #Performance #SQL #Caché #InterSystems IRIS
Check the related application on InterSystems Open Exchange

Source URL:https://community.intersystems.com/post/apm-%E2%80%93-monitoring-sql-query-performance