Foglight Skills 101

Episode 4

Mastering SQL Server Performance with Quest Foglight

Janis Griffin
Senior Systems Consultant

ORACLE ACE

Quest
Where Next Meets Now.

Who Am I?

Current – 30+ Years in Oracle®, DB2®, ASE, SQL Server®, MySQL®, PostgreSQL

DBA and Developer

- Specialize in Performance Tuning
- Customers Common Question: How do I tune it?

ORACLE ACE

Janis.Griffin@quest.com

Twitter® - @DoBoutAnything

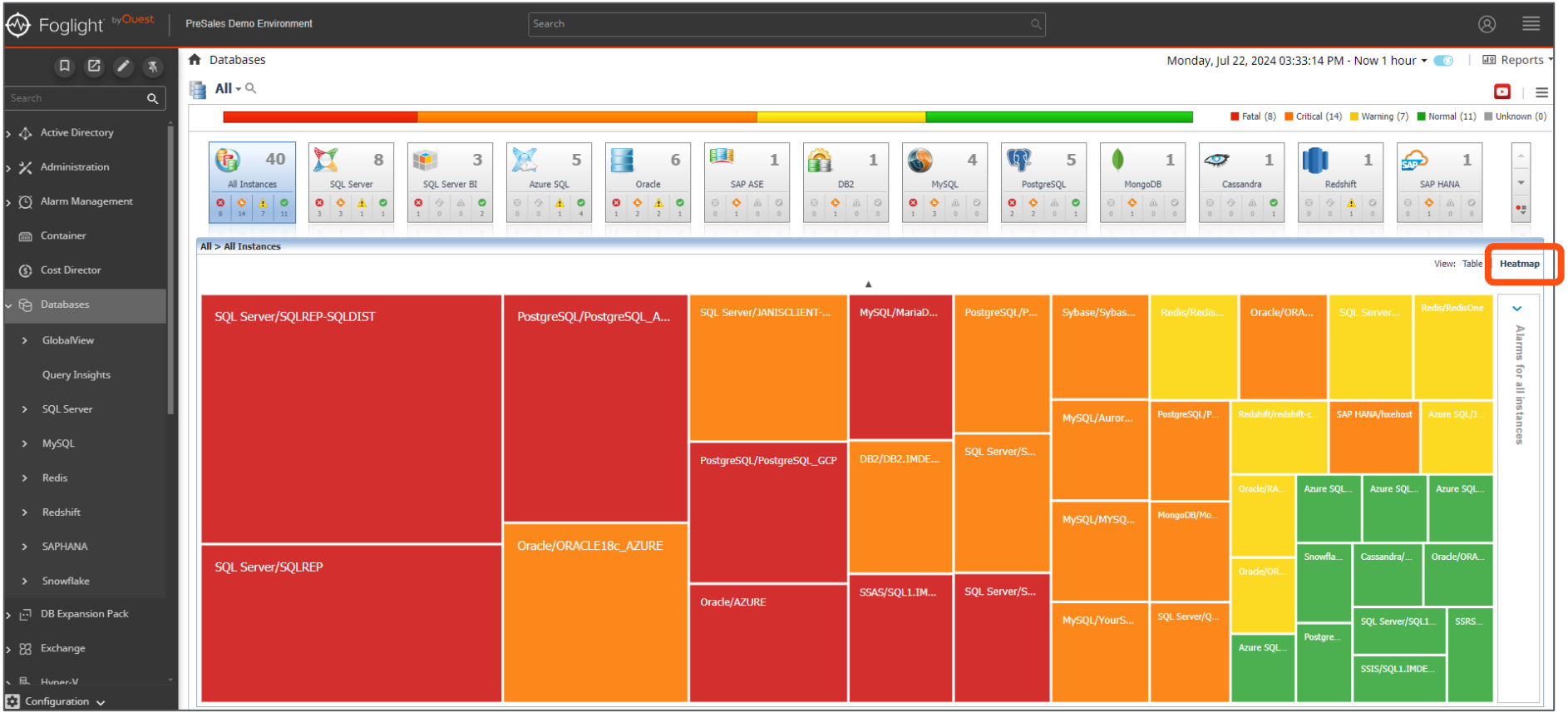
Quest®

Where Next Meets Now.

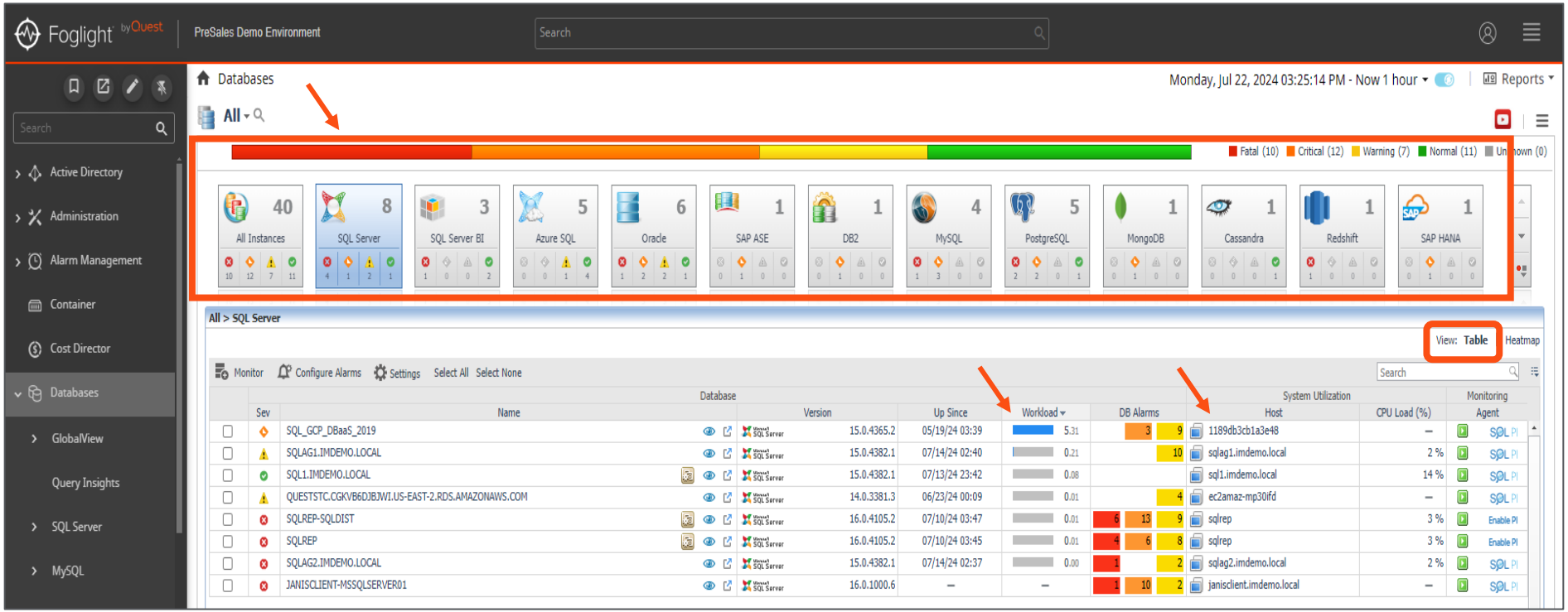
Today's Focus

- **Utilize Query Insights**
- **Monitor & evaluate adaptive baselines & change tracking**
  - Including historical examination & decision making
- **Review key metrics & query execution plans**
- **Use wait types to quickly identify bottlenecks**
  - To gain insights on the best tuning approach
- **Consider tuning with SQL Optimizer**

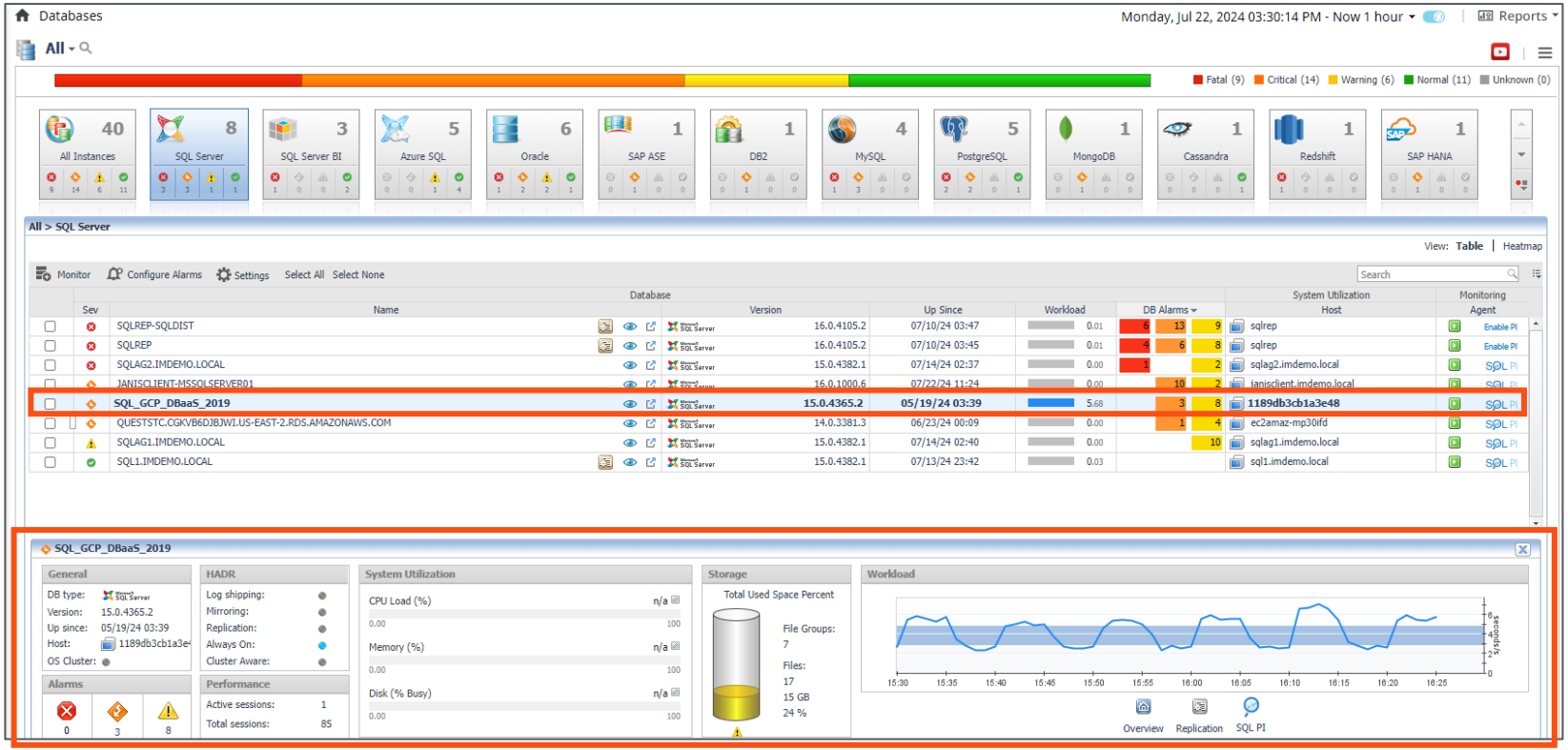
Foglight Shows Company Level Database Pain



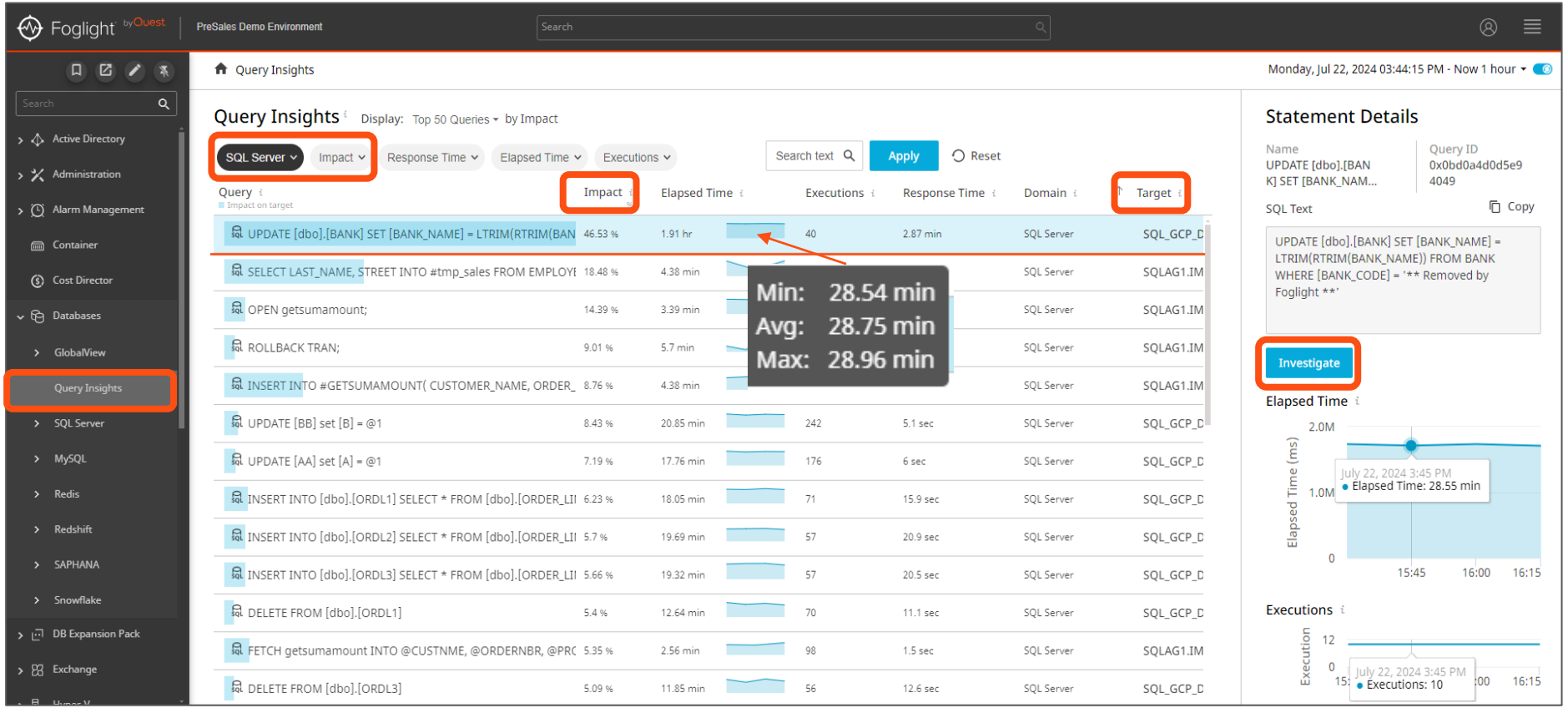
Global Company or Database Specific Views



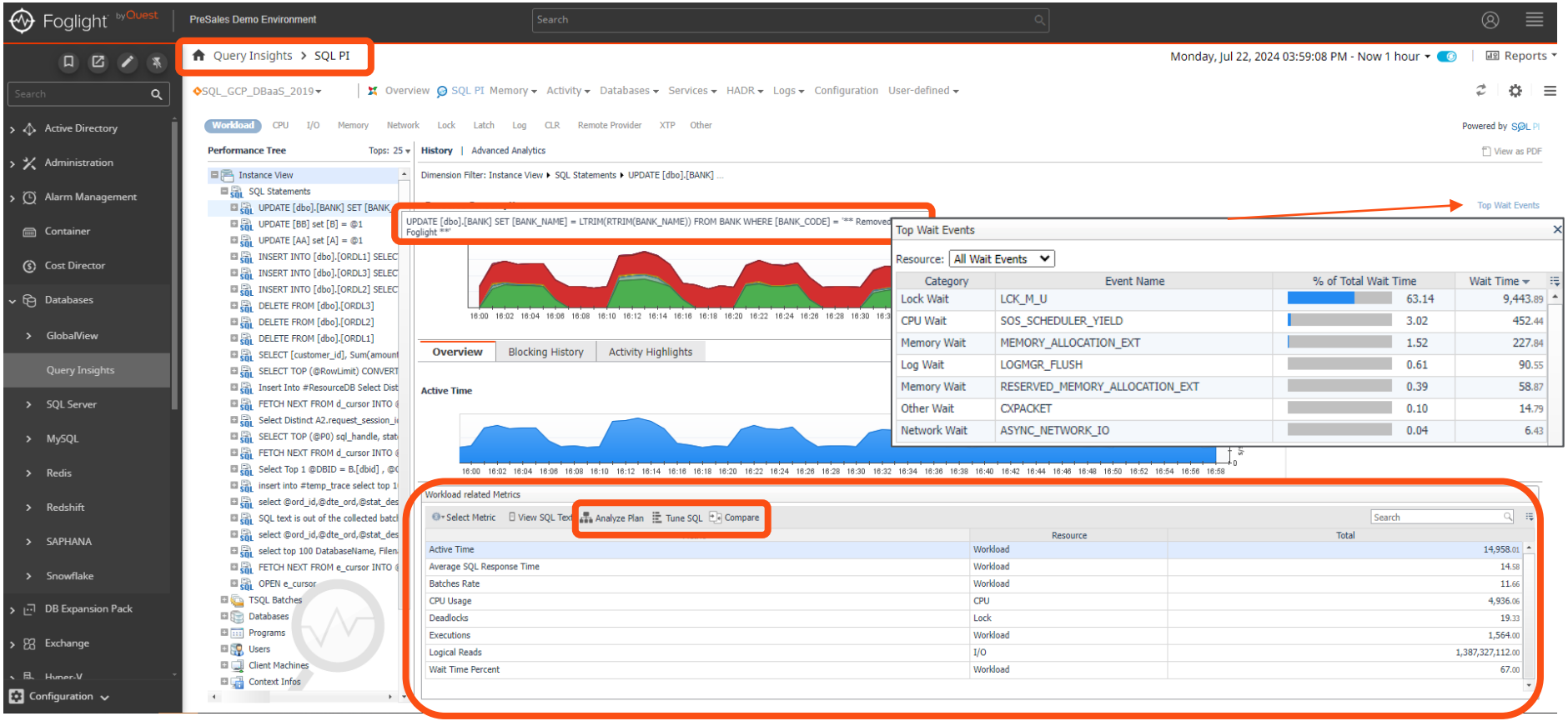
Quick View of Workloads / Baselines



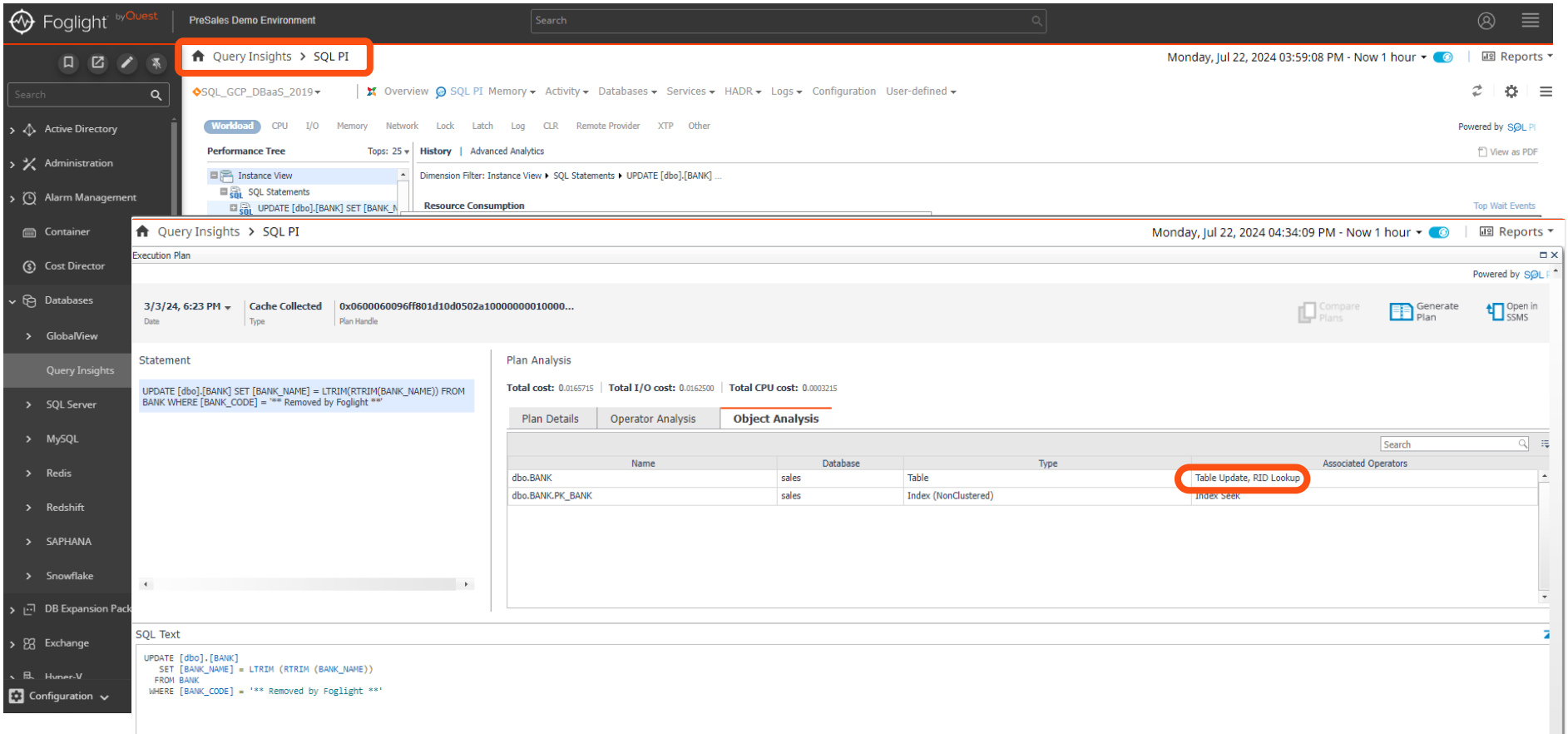
Get Query Insights over All or Selected Domains



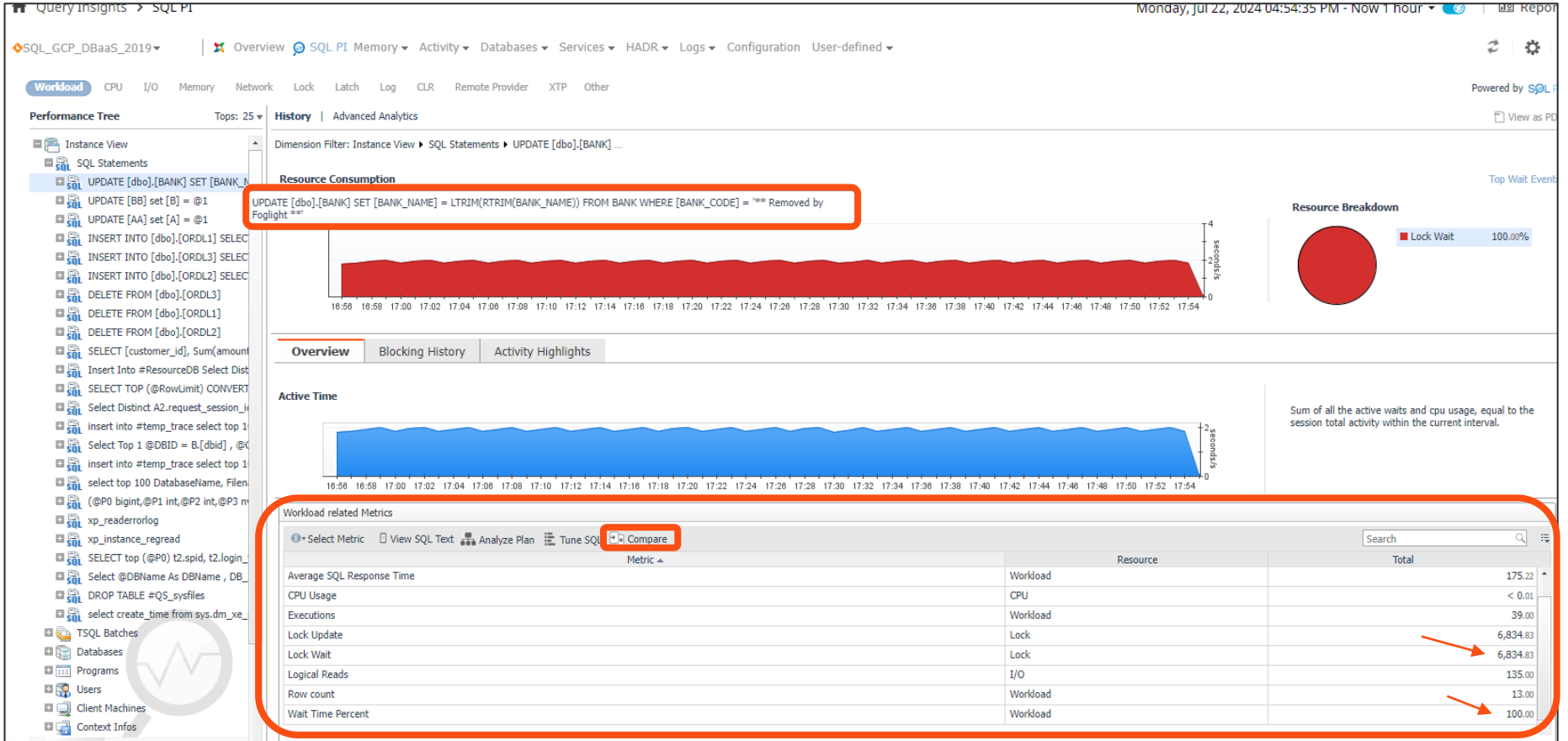
Query Insights – Investigate SQL PI



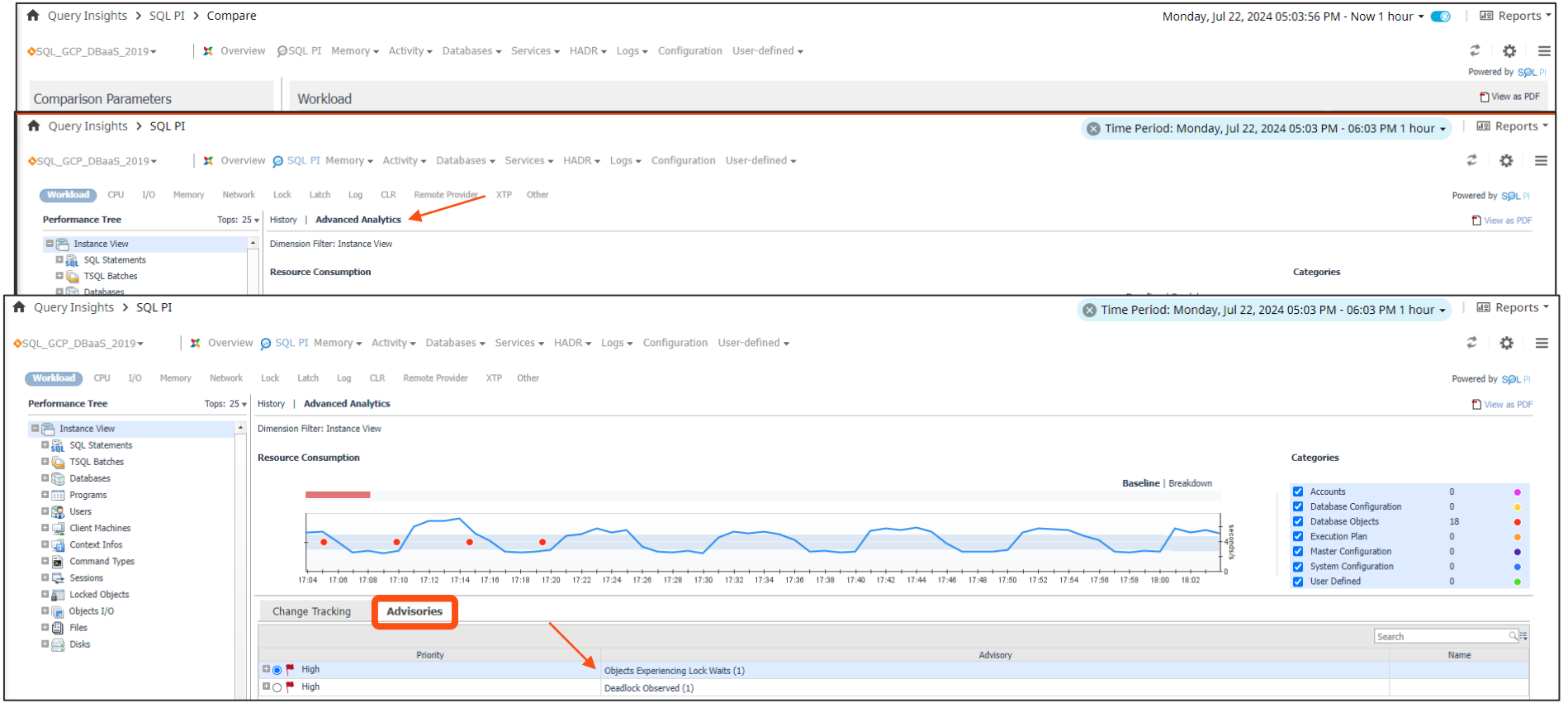
SQL PI – Analyze Plan



SQL PI - Compare



SQL PI – Advanced Analytics



Advisories

Objects Experiencing Lock Waits

Name

Objects Experiencing Lock Waits

Description

Object lock contention was observed during the analyzed period, taking 62.22% of response time.

Figure 1 illustrates the total wait for lock for the monitored SQL Server instance as compared to the overall active time of the instance

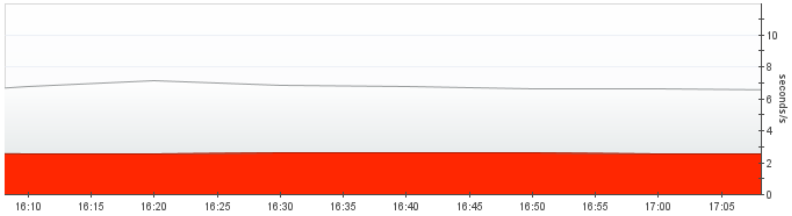


Table 1 shows the list of top objects that experienced Lock Contention.

Locked Objects	Lock Wait Time (seconds)	% of Overall Response Time
sales.dbo.BANK.PK_BANK [INDEX]	7,026.65	75.43
sales.dbo.BB [TABLE]	1,251.99	13.44
sales.dbo.AA [TABLE]	1,104.32	11.85

Click on the Locked Object name to find out more about the identity of the blockers of that object, the amount of sessions involved, and the identity of the locked sessions.

Key Metrics

Table 2 displays the trend of related key metrics, recorded during the analyzed period.

Metric	Average	Min	Max
Blocked Lock Requests	16.91	6.00	33.00
Table Lock Escalations	10.70	0.00	40.00
Lock Timeouts	23.78	6.00	56.00
Average Lock Wait Time (ms)	12,248,568,325.27	12,244,140,552.00	12,253,095,061.00

Recommendation

Review the application logic and consider techniques to reduce contention caused by concurrent updates to data.

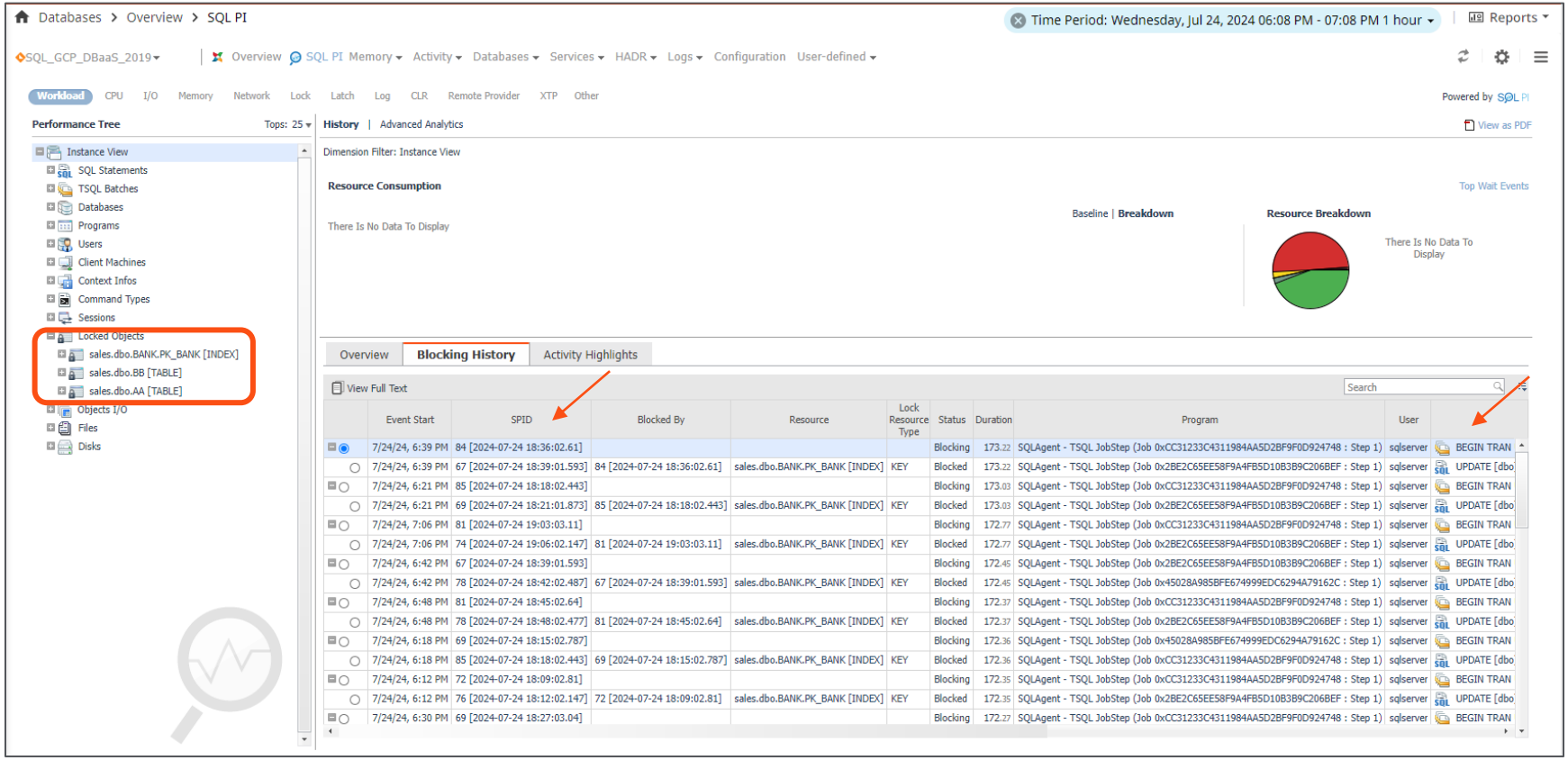
Some of the main causes of table locking include the following:

- Contention for a specific row in the database. The application design may require that many processes update or lock the same row in the database. For example, when primary keys are generated using a sequence table.
- Table locks caused by non-indexed foreign keys, or ineffective / missing indexes, or out-of-date statistics. When a non-indexed foreign key is updated, the parent table could be subjected to a table lock until the transaction is complete. If the indexes defined are ineffective or missing, the SQL optimizer chooses non-optimal query plans. Non-optimal plans can also be chosen when a qualifying index exists, if statistics are out of date.
- Long-running transactions. Individual transact-SQL statements may perform well, but maintaining transactions for extended operations can dramatically increase the amount of contention within a database. Investigate transaction control logic to determine where operations can be broken up into smaller, component transactions. In addition, review the use of transaction isolation levels in application code to determine whether a less restrictive isolation level can be used.

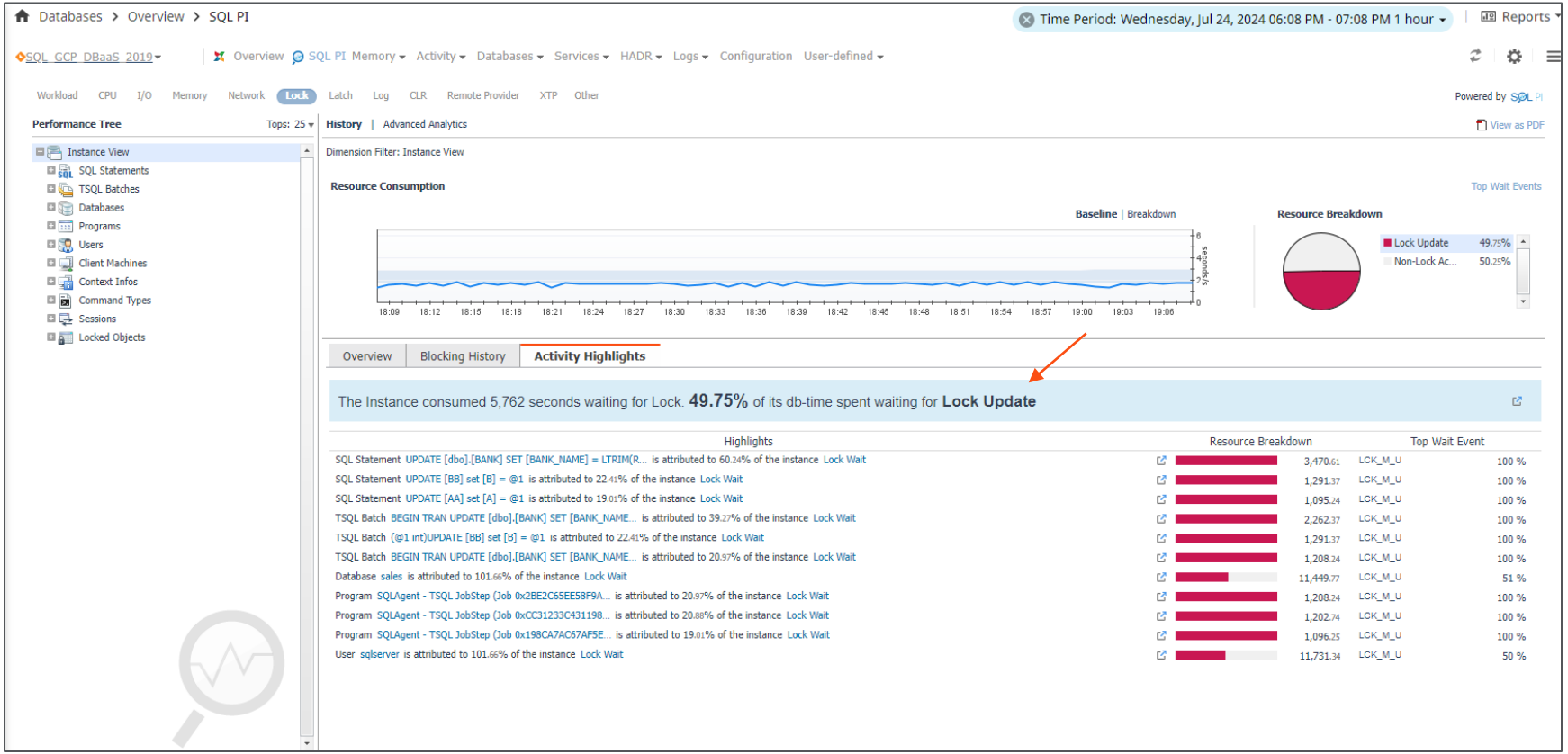
Next steps

- Review Blocking History for a detailed report of all sessions involved in the blocking scenario during the period describec
- Review Activity Highlights to find out top contributors to the lock wait.

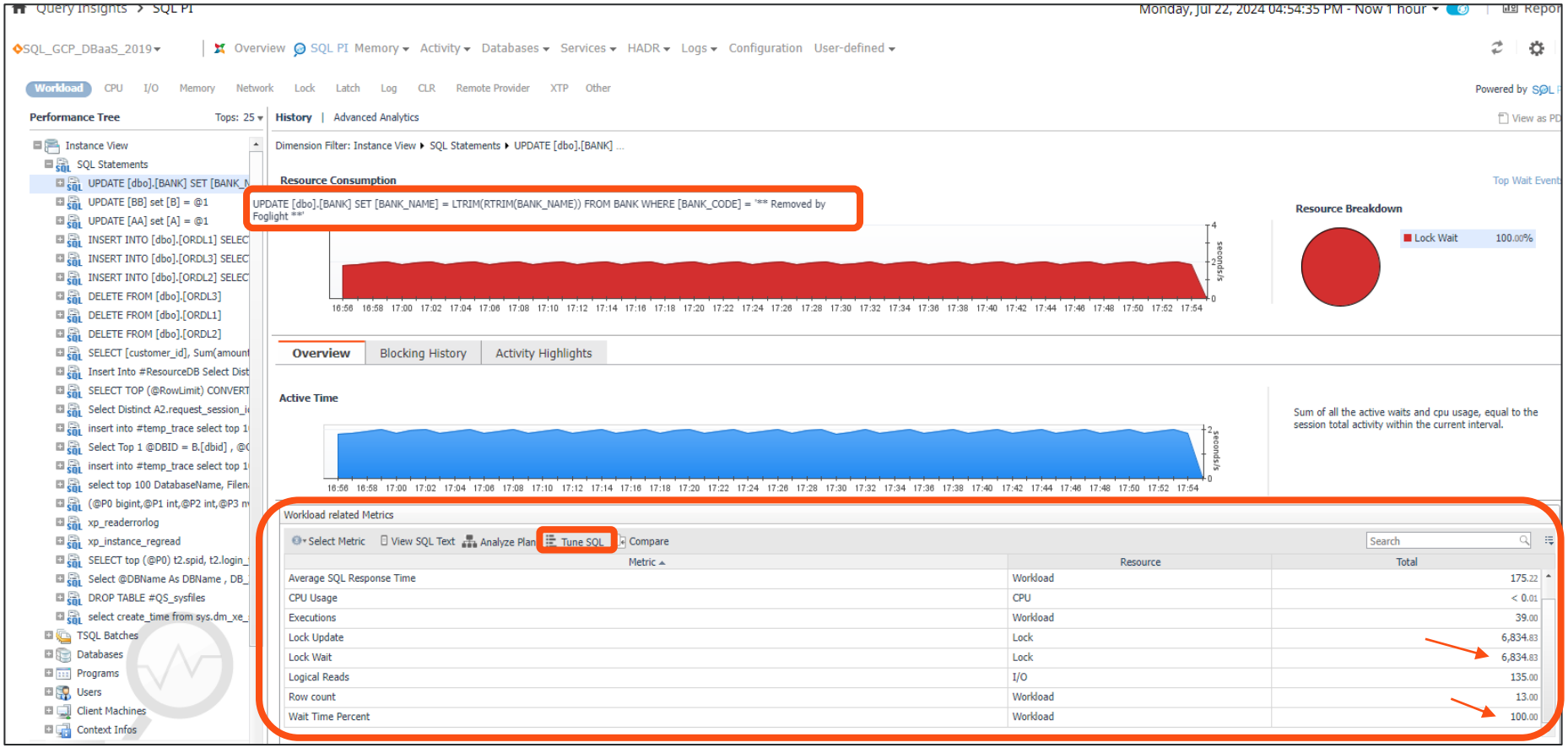
Blocking History



Activity Highlights

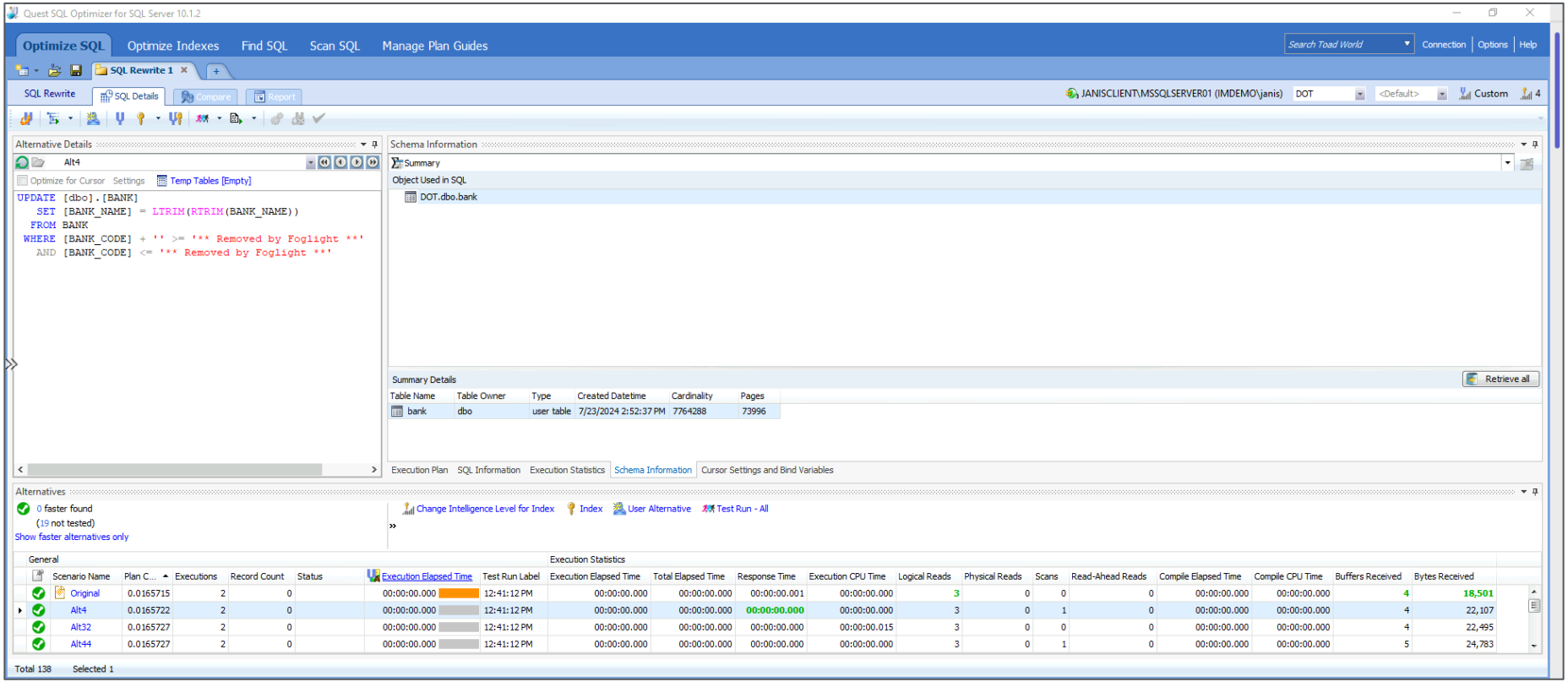


SQL PI – Tune SQL



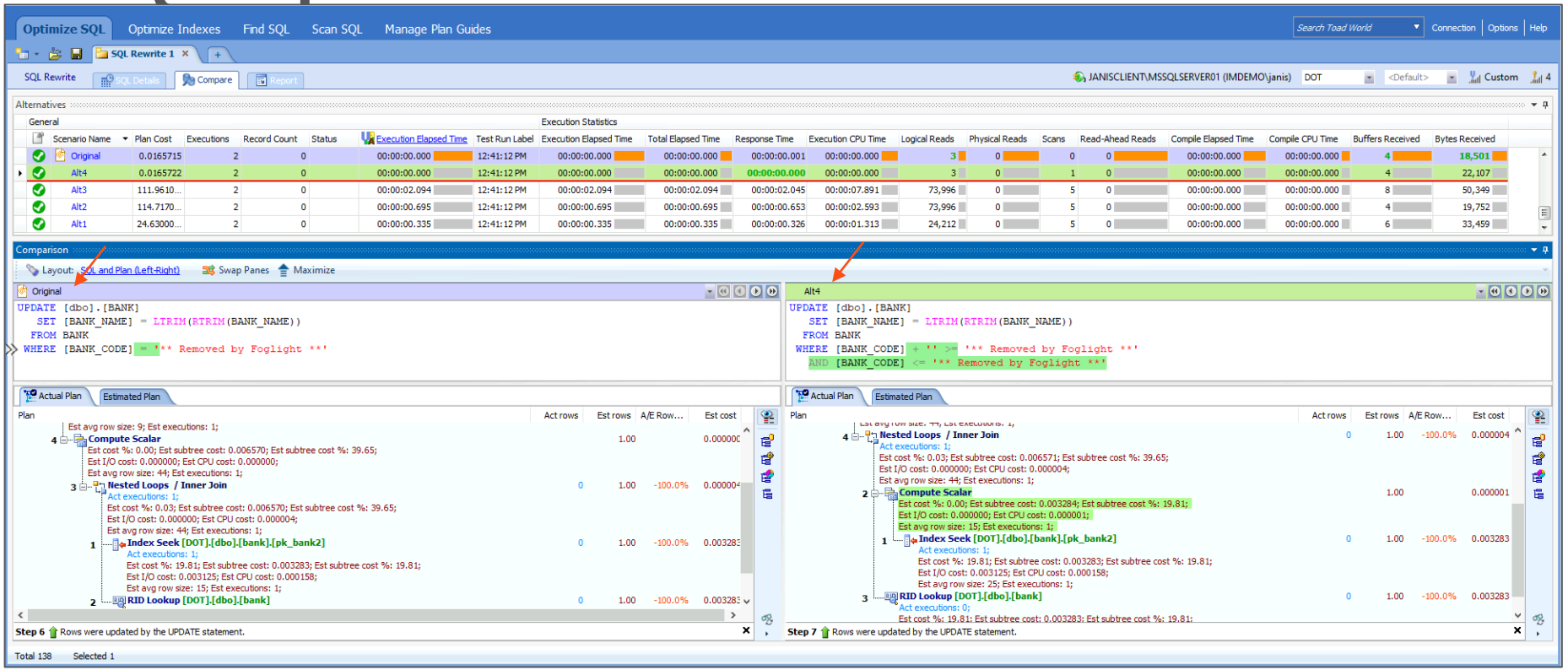
SQL Optimizer

Quest



SQL Optimizer

Quest



Change Heap to Clustered Index

```
CREATE TABLE [dbo].[bank](
    [bank_code] [varchar](20),
    [bank_name] [varchar](50) NULL,
    [address] [varchar](50) NULL,
    [state] [varchar](2) NULL,
    [zip] [varchar](10) NULL,
 CONSTRAINT [PK__BANK] PRIMARY KEY CLUSTERED
 ([bank_code])
)ON [PRIMARY]

insert into bank select * from bank_old;
```

Before

7/24/24, 8:09 PM (Date) | Cache Collected (Type) | 0x0600070086d9220d20db61036b010000010000... (Plan Handle)

Compare Plans | Generate Plan | Open in SSMS

Statement

UPDATE [dbo].[BANK] SET [BANK_NAME] = LTRIM(RTRIM(BANK_NAME)) FROM BANK WHERE [BANK_CODE] = '** Removed by Foglight **'

Plan Analysis

Total cost: 0.0165715 | **Total I/O cost:** 0.0162500 | **Total CPU cost:** 0.0003215

Plan Details | Operator Analysis | Object Analysis

Search

Operator	Object	Operator Cost	Subtree Cost	I/O Cost	CPU Cost	Rows	Arguments
Table Update (Update)	dbo.bank	60.35 %	0.0165715	0.0100000	0.0000010	1	[DOT].[dbo].[bank].[bank_name] = [Expr1003]
Compute Scalar		0.00 %	0.0065705	0.0000000	0.0000001	1	
Nested Loops (Inner Join)		0.03 %	0.0065704	0.0000000	0.0000042	1	
Index Seek	dbo.bank.pk_bank2	19.81 %	0.0032831	0.0031250	0.0001581	1	
RID Lookup	dbo.bank	19.81 %	0.0032831	0.0031250	0.0001581	1	

After

7/24/24, 8:04 PM (Date) | Cache Collected (Type) | 0x060007004ad3961f50d9da446b010000010000... (Plan Handle)

Compare Plans | Generate Plan | Open in SSMS

Statement

UPDATE [dbo].[BANK] SET [BANK_NAME] = LTRIM(RTRIM(BANK_NAME)) FROM BANK WHERE [BANK_CODE] = '** Removed by Foglight **'

Plan Analysis

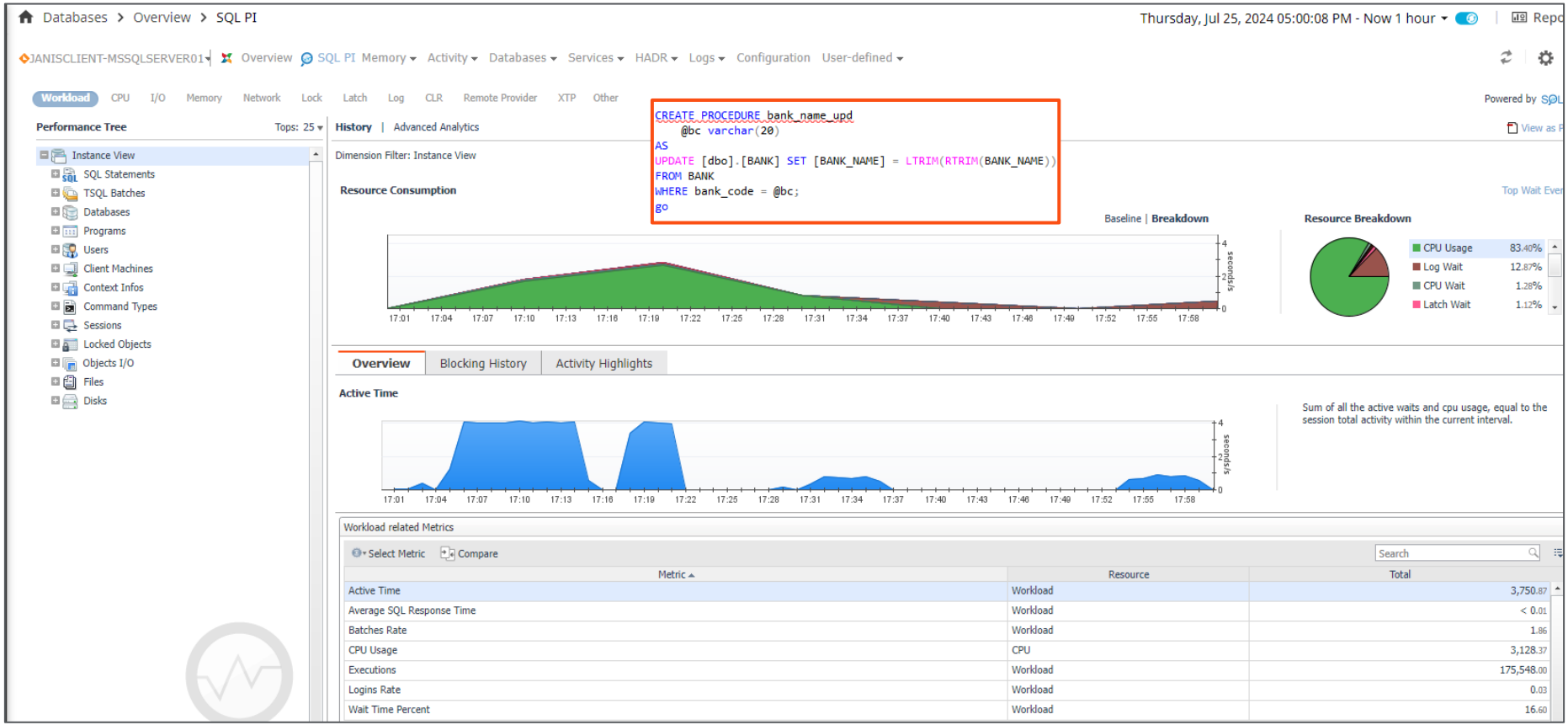
Total cost: 0.0132842 | **Total I/O cost:** 0.0100000 | **Total CPU cost:** 0.0000010

Plan Details | Operator Analysis | Object Analysis

Search

Operator	Object	Operator Cost	Subtree Cost	I/O Cost	CPU Cost	Rows	Arguments
Clustered Index Update (Update)	dbo.bank.PK__BANK	100.00 %	0.0132842	0.0100000	0.0000010	1	[DOT].[dbo].[bank].[bank_name] = [Expr1002]

Create Reusable Stored Procedure



Comparison of Changes = Improvement

- **Heap to Clustered Index (on bank_code)**
- **Change query from passing literals to parameters**
  - With cached stored procedure

Another SQL Optimizer Example

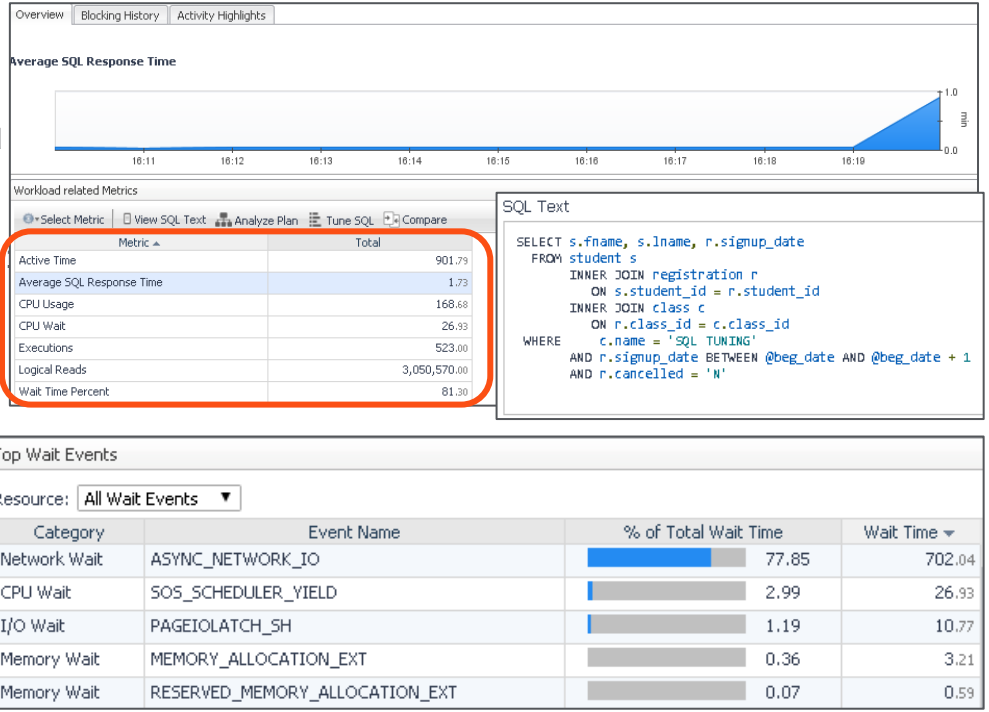
Quest

Who registered yesterday for SQL Tuning?

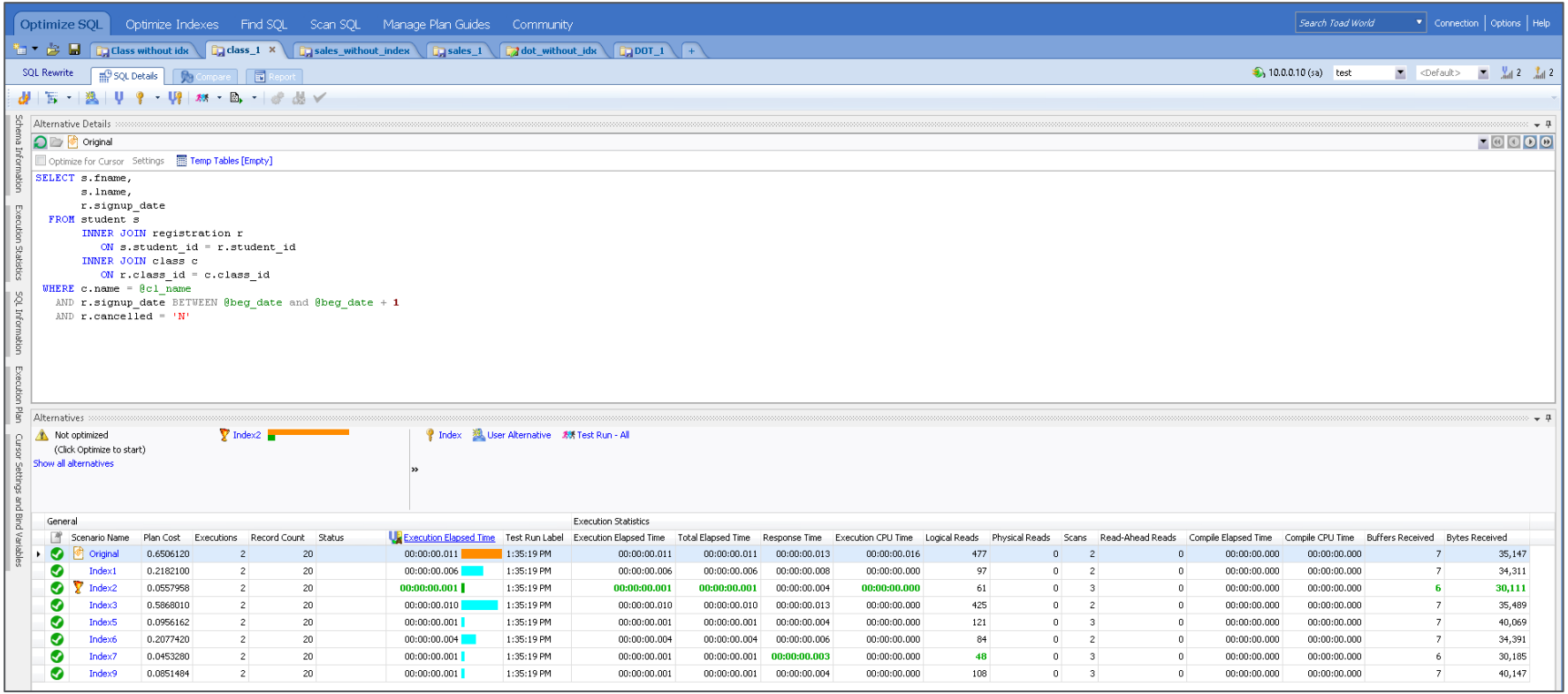
```
SELECT s.fname, s.lname, r.signup_date
FROM   student s
   INNER JOIN registration r ON s.student_id = r.student_id
   INNER JOIN class c ON r.class_id = c.class_id
WHERE  c.name  = 'SQL TUNING'
AND    r.signup_date BETWEEN :beg_date AND :beg_date +
AND    r.cancelled = 'N'
```

Execution Stats –3,050,570 Logical Reads
Execution Time – Approx. 2 seconds
Wait Type – ASYNC_NETWORK_IO

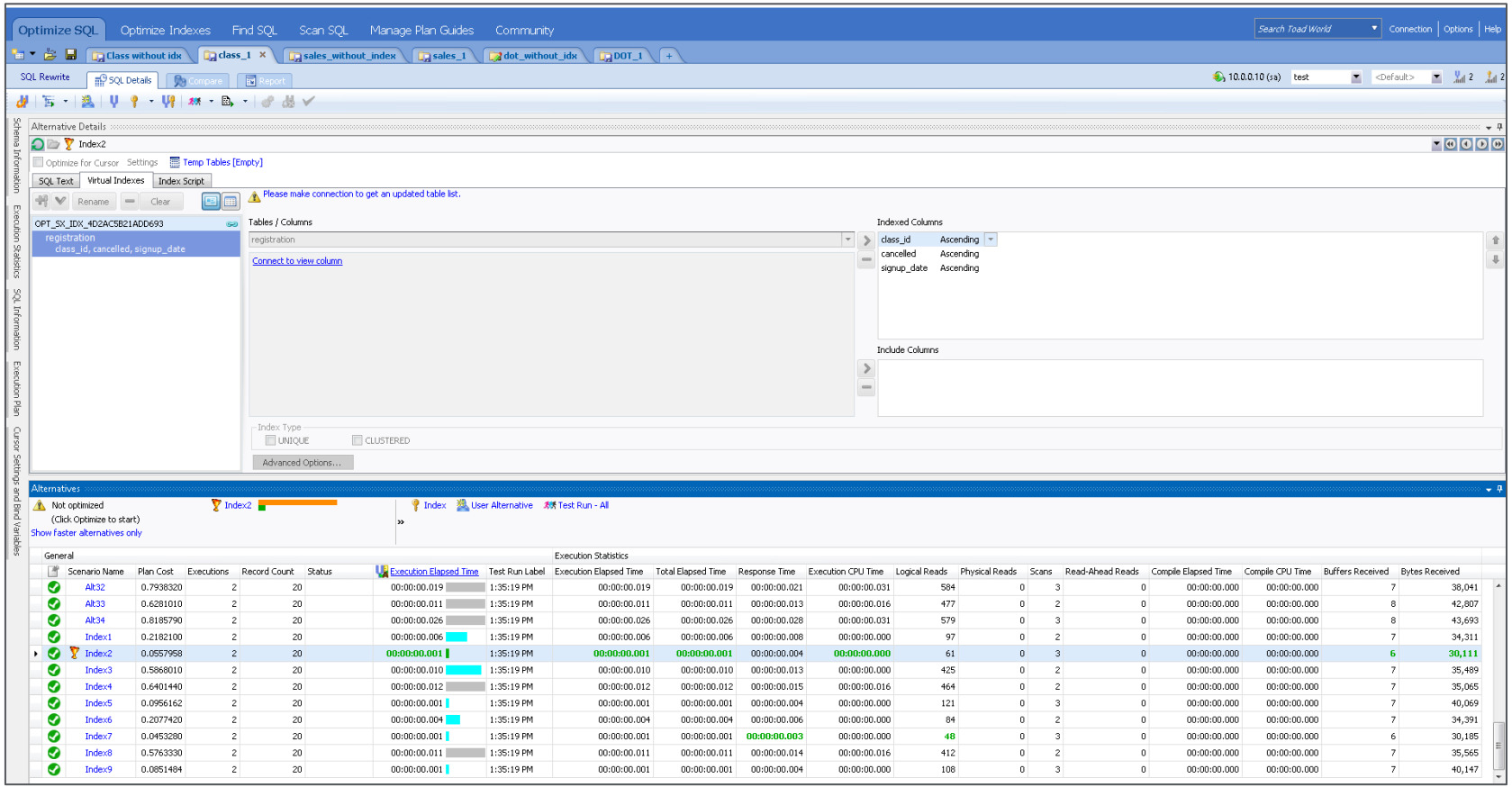
10 Minute Time slice



Sql Optimizer Indexes .011 to .001

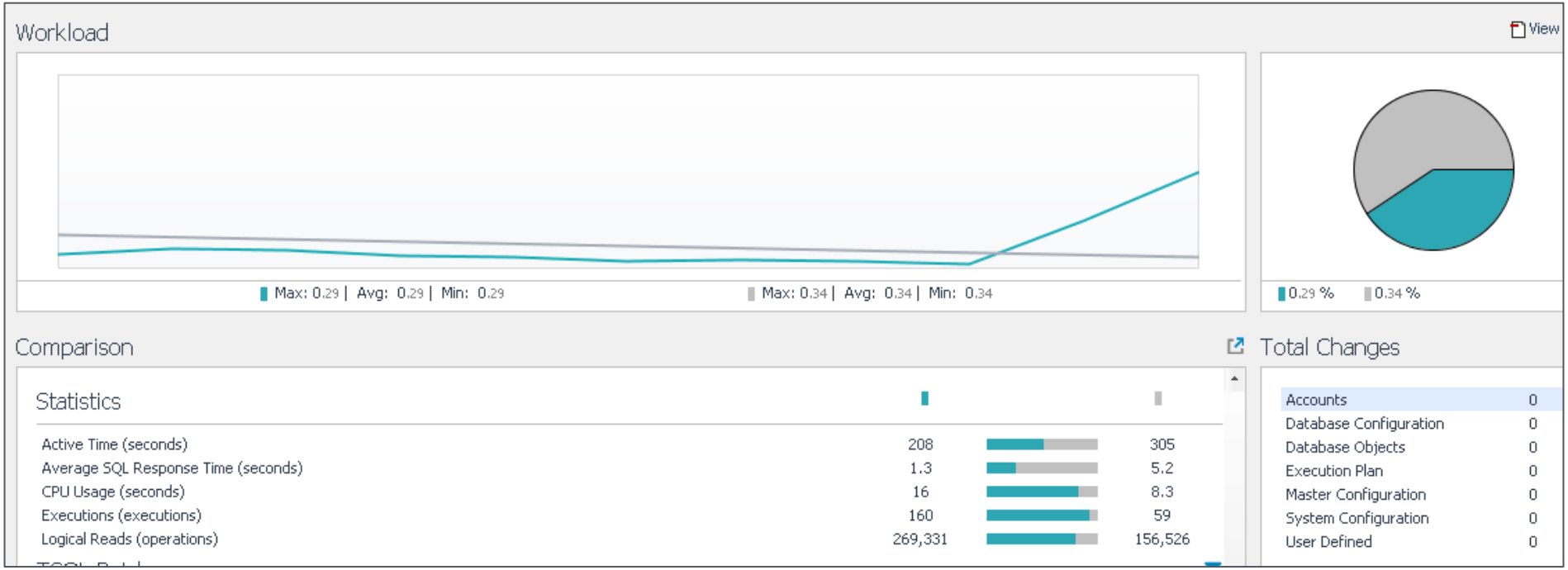


Sql Optimizer Indexes .011 to .001



Improvement?

- **4x faster, 3x throughput**



Summary

Quest

1. **Focus on the queries that impact performance the most**
   - Query Insights
2. **Review baselines & changes**
   - Utilize compare & change tracking features
3. **Review key metrics & query execution plans**
   - Response time, executions, logical_reads & row_counts
4. **Use wait types to identify bottlenecks**
   - They give clues to best tuning approach
5. **Determine if it's a poorly written query or database design flaw**
6. **Consider tuning with SQL Optimizer**

More Performance Tuning Help…

Quest

Resources Blogs Forums

Quest Products Solutions Support & Services Partners About Free Trials Request Pricing

The Zombie Survival Guide to Database Performance Tuning

Download Your Free E-book

Business Email

Download E-book

By downloading, you are registering to receive marketing email from us. To opt-out, follow steps described in our Privacy Policy.

reCAPTCHA protects this site. See Google's Privacy Policy and Terms of Use.

You know that feeling of banging your head against a SQL query or an entire database for minutes – then hours – and watching your day vanish and feeling your brain turn to mush?

Nobody wants you to be a database zombie.

That's why we've pulled together a few decades of our experience in improving database performance to bring you this guide. You'll find six steps on database tuning, with plenty for both newbies and old hands to latch onto and learn from. We've also included a section with variations for Microsoft SQL Server, Oracle Database, MySQL and PostgreSQL.

Zombie Survival Guide

Thank You

Looking Forward to Seeing You There

Register Today!

Foglight 101 Series: Episode 5

Exploring REST API Functionality

Clay Jackson

May 21st

Quest

Where Next Meets Now.