

---

# SQL Performance Performance Methodology

---

---

## Performance Methodology

---

The overall structure or methodology used for discovery, capture, analysis, resolution and knowledge transfer looks like the following:

1. Capture of database SQL performance metrics, via in-house tools
2. Prioritization of SQL performance offenders (including quantifiable prioritization by metrics such as CPU Time, Elapsed Time, Logical Reads, Physical Reads, etc.). Quality and amount of SQL prioritization metrics depend on usefulness of in-house tools. This information will be in reports called "Overall Analysis Reports".
3. Analysis of SQL performance offenders (including execution plan analysis and assessment of SQL versus database performance issues to address excessive metrics listed above in #2).
4. Iterative performance solution testing of SQL offenders (the focus being on optimal SQL execution plans via performance tactics such as indexing, statistic strategies, SQL changes, or other SQL performance tactics).
5. Documentation of all performance solutions with a summary of each performance issue, a summary of the solution, a detailed report of the performance recommendation with before and after metrics showing overall benefit. Recommendations will be in reports called "Individual Performance Recommendations".
6. Extensive knowledge transfer to any interested staff through mentoring of all solutions.

Even though the above is a typical framework for discovery, analysis, resolution and knowledge transfer, there has to be a starting point. This requires initial inquiring about any known performance issues or concerns. This could require asking questions of infrastructure personnel (database, operating system, network), application team personnel (developers, managers) and business personnel (end users). If it's a B2B application, even interviewing end users at the other business is helpful. This is all in an effort to inventory perceived performance issues in the database, hardware and software.

Based on that information there will be focus on the prevailing pain points. Whether it's general hardware issues (saturation of CPU, Memory or DASD) or end user experience issues (long response times, timeouts), analysis will typically require the use of tools to focus on SQL's that exhibit high CPU consumption, high logical reads (consistent gets), high physical reads, and long SQL durations. General hardware issues can be sustained or in peaks. General hardware issues would be considered a system or aggregate approach to performance, while end user experience issues would be considered a transactional approach. Each approach has it's own merit and it's usually a combination of the two needed in order to achieve maximum optimization.

Sometimes, there will be no initial information from which to assess a starting point and regardless if there are no perceived end user experience performance issues, there should be an assessment of the “Top 10%” critical functionality for each application. This is the functionality that is executed either all the time every day or that it represents mission critical value regardless of execution frequency. It is this functionality where all possible avenues of performance optimization should be considered. In a recurring benchmark or system testing phase, this functionality should always be tested regardless if it’s code had any changes in that particular version. In a production scenario, this functionality should have regular monitoring and alerting mechanisms that are actionable and timely.

Once offending SQL’s are gathered and prioritized, the following is an approach for analysis of each SQL:

1. Format the SQL into a readable form
2. Gather statistics about all tables and indexes (do this for production and a testing environment)
  - a. Make note of any objects that could have insufficient statistics.
    - i. Tables are fine for sampling less than 100% usually, especially if very large
    - ii. Indexes should have full computed statistics unless the statistics window is being compromised and then only carefully chosen indexes should be sampled. Fully sampled indexes are best for giving the Optimizer all considerations for plan construction.
3. Gather all indexing structures, with columns in order as they appear in the index (do this for production and a testing environment)
4. Get values with which to test for any bind variables (do this for production and a testing environment)
5. Capture original performance metrics using #4 values that bring back data in both test and production. Capture CPU, Logical Reads (consistent gets), Physical Reads, Total Time and Total Rows. Do this in the production and testing environment.
6. Capture the original SQL execution plan using the SQL in #5 in both production and testing environment.
7. Make note of any difference between the production and testing plans. If different, steps need to be taken to ensure they are the same before continuing analysis. This sometimes requires exporting the production statistics and importing them into the testing environment.
8. Perform iterative testing of the SQL in the testing environment until resolved. This particular step can be a significant effort with an explanation too vast to put in this document. There is where the experience of the analyst is employed and mentoring can be engaged to broaden skillsets.
9. Document SQL performance resolution in a deliverable report that contains
  - a. Before/after performance metrics (CPU, Elapsed Time, Logical Reads, Physical Reads)
  - b. The original SQL form
    - i. Any commentary about the problematic area(s) of the SQL
  - c. The original SQL execution plan
    - i. Any commentary about the problematic area(s) of the plan

- d. The proposed recommendation with full rationale. This is critical for knowledge transfer and solution history.
- e. The tuned SQL form and/or execution plan
  - i. Any commentary on the observed effects of the SQL or plan