Progress®



Optimize OpenEdge Performance

OpenEdge®

Copyright

© 2021 Progress Software Corporation and/or its subsidiaries or affiliates. All rights reserved.

These materials and all Progress® software products are copyrighted and all rights are reserved by Progress Software Corporation. The information in these materials is subject to change without notice, and Progress Software Corporation assumes no responsibility for any errors that may appear therein. The references in these materials to specific platforms supported are subject to change.

DataDirect Cloud, DataDirect Connect, DataDirect Connect64, DataDirect XML Converters, DataDirect XQuery, DataRPM, Defrag This, Deliver More Than Expected, DevReach (and design), Icenium, Inspec, Ipswitch, iMacros, Kendo UI, Kinvey, MessageWay, MOVEit, NativeChat, NativeScript, OpenEdge, Powered by Chef, Powered by Progress, Progress, Progress Software Developers Network, SequeLink, Sitefinity (and Design), Sitefinity, Sitefinity (and design), SpeedScript, Stylus Studio, Stylized Design (Arrow/3D Box logo), Styleized Design (C Chef logo), Stylized Design of Samurai, TeamPulse, Telerik, Telerik (and design), Test Studio, WebSpeed, WhatsConfigured, WhatsConnected, WhatsUp, and WS_FTP are registered trademarks of Progress Software Corporation or one of its affiliates or subsidiaries in the U.S. and/or other countries.

Analytics360, AppServer, BusinessEdge, Chef Automate, Chef Compliance, Chef Desktop, Chef Habitat, Chef WorkStation, Corticon.js, Corticon Rules, Data Access, DataDirect Autonomous REST Connector, DataDirect Spy, DevCraft, Fiddler, Fiddler Everywhere, FiddlerCap, FiddlerCore, FiddlerScript, Hybrid Data Pipeline, iMail, JustAssembly, JustDecompile, JustMock, KendoReact, NativeScript Sidekick, OpenAccess, PASOE, Pro2, ProDataSet, Progress Results, Progress Software, ProVision, PSE Pro, Push Jobs, SafeSpaceVR, Sitefinity Cloud, Sitefinity CMS, Sitefinity Digital Experience Cloud, Sitefinity Feather, Sitefinity Insight, Sitefinity Thunder, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, Supermarket, SupportLink, Unite UX, and WebClient are trademarks or service marks of Progress Software Corporation and/or its subsidiaries or affiliates in the U.S. and other countries. Java is a registered trademark of Oracle and/or its affiliates. Any other marks contained herein may be trademarks of their respective owners.

July 2021

Product version: Progress OpenEdge 12.4

Updated: 2021/06/30

Table of Contents

Preface....7

Part I: Get started....9

Learn about OpenEdge performance....9
About performance tuning....12

Part II: By design....13

Optimize performance with OpenEdge features....13

Part III: Develop....15

Consider factors that affect application performance....16
Create OpenEdge databases that perform well....16
Code applications that perform well....20
Apply best practices for ABL....20
Apply best practices for SQL....22
Apply best practices for third-party data servers....22

Part IV: Configure....23

Initialize ABL clients....23
Initialize OpenEdge database servers....27
Initialize Progress Application Server for OpenEdge....29

Part V: Tune....33

Learn about tuning....34
Tune ABL code....35
Tune OpenEdge database servers....38
Tune Progress Application Servers for OpenEdge....41

Part VI: Learn more....45

Related solutions....45

Preface

Purpose

This guide provides guidelines to optimize the performance of your OpenEdge applications.

Audience

This guide is intended for solution architects, database administrators, and application developers interested in improving the performance of their OpenEdge applications. Solution architects, database administrators and application developers learn about decisions they can make during design and development that can help their applications perform well in production. Additional tools and techniques are provided for generating a baseline for performance and then you monitor, tune, and test to verify that the changes you made improved performance.

Organization

This content is organized based on the different application life cycle phases of designing, developing, optimizing and tuning.

This guide consists of the following parts:

- Get started introduces when performance tuning is required:
  - Learn about OpenEdge performance on page 9: An overview of the process for building applications that perform well by introducing performance considerations during each phase of the application life cycle, from development to production
  - About performance tuning on page 12: A scientific approach to tuning your ABL code, database servers, and applications servers
- By design identifies features that support optimal performance:
  - Optimize performance with OpenEdge features on page 13: OpenEdge features designed to improve performance just by upgrading to the current release
- Develop lists guidelines and best practices to apply during your development phase:
  - Consider factors that affect application performance on page 16: Identifies universal factors that affect performance including system, database, application code, and application servers, and describes their interdependence
  - Create OpenEdge databases that perform well on page 16: Offers guidelines for creating logical and physical database components
  - Code applications that perform well on page 20: Introduces best practices for ABL, SQL, and DataServers
- Configure introduces parameters that improve run-time performance for ABL clients, database servers, and application servers during testing and staging:
  - Initialize ABL clients on page 23
  - Initialize OpenEdge database servers on page 27
  - Initialize Progress Application Server for OpenEdge on page 29
- Tune introduces the performance tuning life cycle, and provides links to performance tuning resources for ABL, databases, and application servers. This topic also describes the scientific approach to identifying issues in testing, staging, and production environments:

- Learn about tuning on page 34
- Tune ABL code on page 35
- Tune OpenEdge database servers on page 38
- Tune Progress Application Servers for OpenEdge on page 41

- Learn more provides links to other guides and how to become more involved in contributing feedback to Progress:
  - Related solutions on page 45

Documentation Conventions

See Documentation Conventions for an explanation of the terminology, format, and typographical conventions used throughout the OpenEdge content.

Get started

For details, see the following topics:

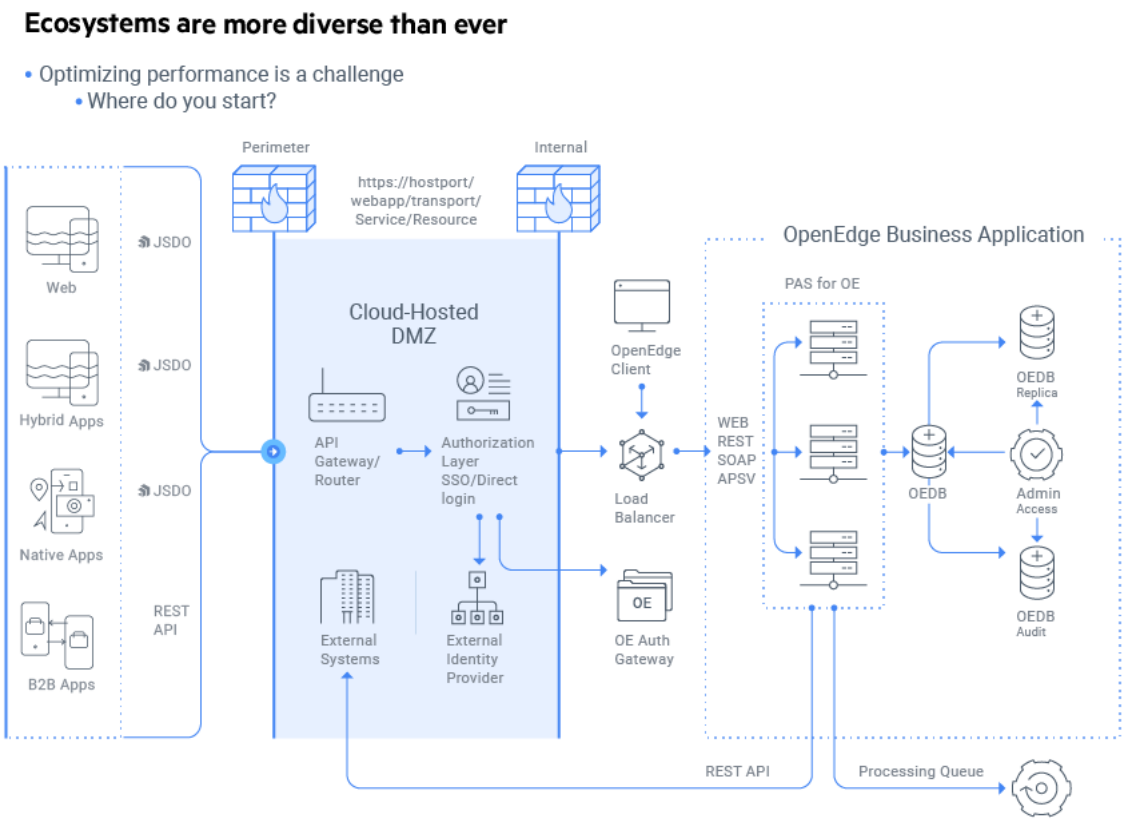
- Learn about OpenEdge performance
- About performance tuning

Learn about OpenEdge performance

As solution architects, database administrators, and application developers, you are building more complex applications than you built in the past. This guide identifies the performance gains you can achieve by using efficient ABL code and by tuning your Progress OpenEdge database servers and Progress Application Server (PAS) for OpenEdge instances.

Why optimize OpenEdge performance?

With more moving parts in modern applications, performance can suffer. To improve performance, you must **increase throughput** which means that you must **identify and remove potential bottlenecks** in your OpenEdge applications. Bottleneck symptoms are slow or unresponsive systems that impede the ability of users to do their jobs efficiently.



The source of the bottleneck could be caused by:

- Poorly written code
- Poorly designed databases
- Poorly tuned application servers
- High system loads

A handful of techniques are used to improve performance. Among them are code optimization, load balancing, caching strategy, distributed computing, and self-tuning. If your users are experiencing delays, then those delays are a sign that your overall application would benefit from performance testing and tuning. Reducing bottlenecks is critical to optimizing performance. To accomplish optimal performance, you must employ good habits throughout your development life cycle.

Characteristics of optimal OpenEdge performance

Every enterprise is concerned with application performance. In some enterprises, the measurement is simply user satisfaction:

- Is the application responsive?
- Are reports running quickly?
- Are the users able to do their jobs?

In other cases, monitoring and improving performance are integral to their success:

- What is the latency on order placement?
- How efficiently are batch processes running?
- Are system resources consumed by potential memory leaks?

Regardless of how you measure optimal performance, applications that perform well have the following characteristics:

- Build application on an efficient and easy-to use development platform
  - Robust, efficient OpenEdge database
  - Easy-to use ABL for building a modern application with fast data access
  - Secure, scalable Progress Application Server for OpenEdge
  - Support for CI/CD pipeline to develop enhancements quickly
  - Designed for continuous availability to eliminate costly downtime
- Efficient code
  - Uses the best indexing to retrieve records
  - Avoids record contention
    - Manages transactions well
    - Manages record locks well
  - Avoids memory leaks
  - Follows general coding best practices
- Well-designed databases
  - Apply logical database design principles
  - Follow guidelines for defining optimal physical database structures
  - Use Type II storage areas to keep related data grouped closer together
- Tuned application servers
  - Sized to perform correctly under varying loads
  - Monitored for server health and load balanced

About performance tuning

To get the best possible performance from your applications, you must continuously monitor and tune your ABL code, OpenEdge database, and Progress Application Server for OpenEdge. You must iterate in order to optimize performance.

Performance considerations

Each part of this guide is designed to move you closer to optimum application performance:

- **By design** offers new performance features demonstrate the performances gains achieved simply by moving to the current release of OpenEdge.
- **Develop** provides guidelines to create your physical database and code your applications to perform well.
- **Configure** gives tips for initializing database servers and application server instances in test, staging, and production.
- **Tune**, looks at the iterative process of continuously improving performance.

Obtaining optimal performance is a process that starts as you build your applications. For that reason, it is important to apply the recommendations in this guide during each phase of development, testing, staging, and production. Applying the tuning recommendations to your production environment may not yield the same benefits as applying the principles throughout your development life cycle. There are key opportunities to optimize your application performance in each phase.

Note: Every application is unique. A one-size-fits-all approach is not realistic. This guide offers a baseline for many applications. The goal is to introduce concepts and tools for you to consider and apply to your application environment.

Tuning: main steps

Tuning is a continuous process, and using a scientific approach to tuning is a best practice. The scientific approach to tuning relies on isolating and changing a single factor at a time. The isolated change is tested and if that change improves performance when the tests are re-run, then a performance gain can be verified. If performance does not improve, then the isolated change can be reversed, and an alternate factor can be changed and tested in the next iteration.

The steps in the performance tuning cycle include:

1. *Monitor* your overall application performance to look for opportunities to improve performance.
2. *Analyze* the results you are monitoring through OpenEdge Management, PROMON, virtual system tables, OpenEdge HealthScanner, or log files. Determine where the issue is: Is it in the code, the application server, or the database server?
3. *Isolate* the symptom. Determine which logs, tools, and tests to use to isolate the source of the bottlenecks?
4. *Change* a single setting at a time.
5. *Test* for any improvements and possible loss of performance.
6. *Repeat* the cycle and monitor the overall performance of your application, to confirm whether there are continued performance gains.

II

By design

For details, see the following topics:

- Optimize performance with OpenEdge features

Optimize performance with OpenEdge features

Each OpenEdge release includes performance improvements to the ABL, the OpenEdge database, and the Progress Application Server (PAS) for OpenEdge. In some cases, the benefits are automatically applied when you upgrade to the latest OpenEdge release. In other cases, the decision to enable performance features is yours .

OpenEdge provides the following performance enhancements by default:

- **Multi-threaded database server**—Runs the same workload much faster. In some cases, in one-third of the time compared to a prior OpenEdge single-threaded database server. For more information about how to increase the speed of your OpenEdge application by moving to OpenEdge 12 releases, see Progress OpenEdge 12 Database Performance Improvements.
- **Increased throughput with server-side join**—Queries that join multiple database tables are now resolved in the database server rather than on remote clients. As a result, fewer records may be returned to the remote client, leading to faster performance. For more information, see Great performance out-of-the-box with server-side join and Server-Side Joins with Dynamic Queries
- **Improved database startup parameter defaults**—Based on feedback from customer and real-world applications, the default database parameters are designed to perform better for most customers.

  For more information, see 'Default values for database startup parameters' in *Improve the Performance of the OpenEdge Database*.

- **Improved application startup parameters**—Based on feedback from customer and real-world applications, the Progress Application Server for OpenEdge startup parameters are designed to perform better for most customers.

  For more information, see Initialize Progress Application Server for OpenEdge on page 29.

- **Support for more sequences by default**—Removes the sequence limit, allowing database administrators to have additional sequences required to maintain unique IDs.

  For more information, see 'Default Maximum numbers of unique sequences' in *Improve the Performance of the OpenEdge Database*.

- **Reduced locking delays in buffer hash tables**—Enables database administrators to tailor the number of latches allocated for the buffer pool hash table (BHT) entries which reduces locking delays and provides faster access to data.

  For more information, see 'Reduced locking delays in buffer hash table' in *What's New in OpenEdge 12*.

- **More relevant SQL query information in the `protrace.pid` file**—Captures information required for debugging such as the username and database name, to gain a better insight into SQL application issues.

  For more information, see 'SQL Protrace crash reports' in *OpenEdge Data Management: SQL Development*.

While the previous enhancements are enabled by default, the following feature requires that developers opt-in to use:

- **Extend variable-length database extents functionality**—Allows database administrators to extend and mark variable-length `ai`, `bi`, and `db` extents as fixed and to create and pre-extend new variable-length extents. Extending or pre-extending the size of variable-length extents reduces overhead and improves run time by minimizing resource requests to the file system.

  For more information, see 'Extend Variable-length database extents functionality' in *Improve the Performance of the OpenEdge Database*.

III

Develop

For details, see the following topics:

- Consider factors that affect application performance
- Create OpenEdge databases that perform well
- Code applications that perform well

Consider factors that affect application performance

As you begin to design and plan for the database and application servers required for your OpenEdge application, there are some universal factors that affect the performance. Performance problems can be caused and mitigated by the following factors:

- Overall application architecture
  - Hardware capabilities including memory, graphics, storage, networking, and so on
  - Operating system settings
  - Network topology and throughput
- Front-end application code
  - Client hardware capabilities including memory, graphics, storage, networking, and so on
  - Coding best practices
- Database considerations
  - Logical database design
  - Physical database design
  - Startup parameters
- Application server considerations
  - Startup parameters
  - Load balancing
  - Continuous monitoring
- Any combination of the previously mentioned items

Note: Even though memory, hardware, and connectivity are mentioned, the focus of this content is on those things you configure, monitor, and tune in OpenEdge applications and servers. This guide does not address the CPU, memory, and network complexities that may also be a source of bottlenecks because those areas are unique to your ecosystem and are not OpenEdge products.

Create OpenEdge databases that perform well

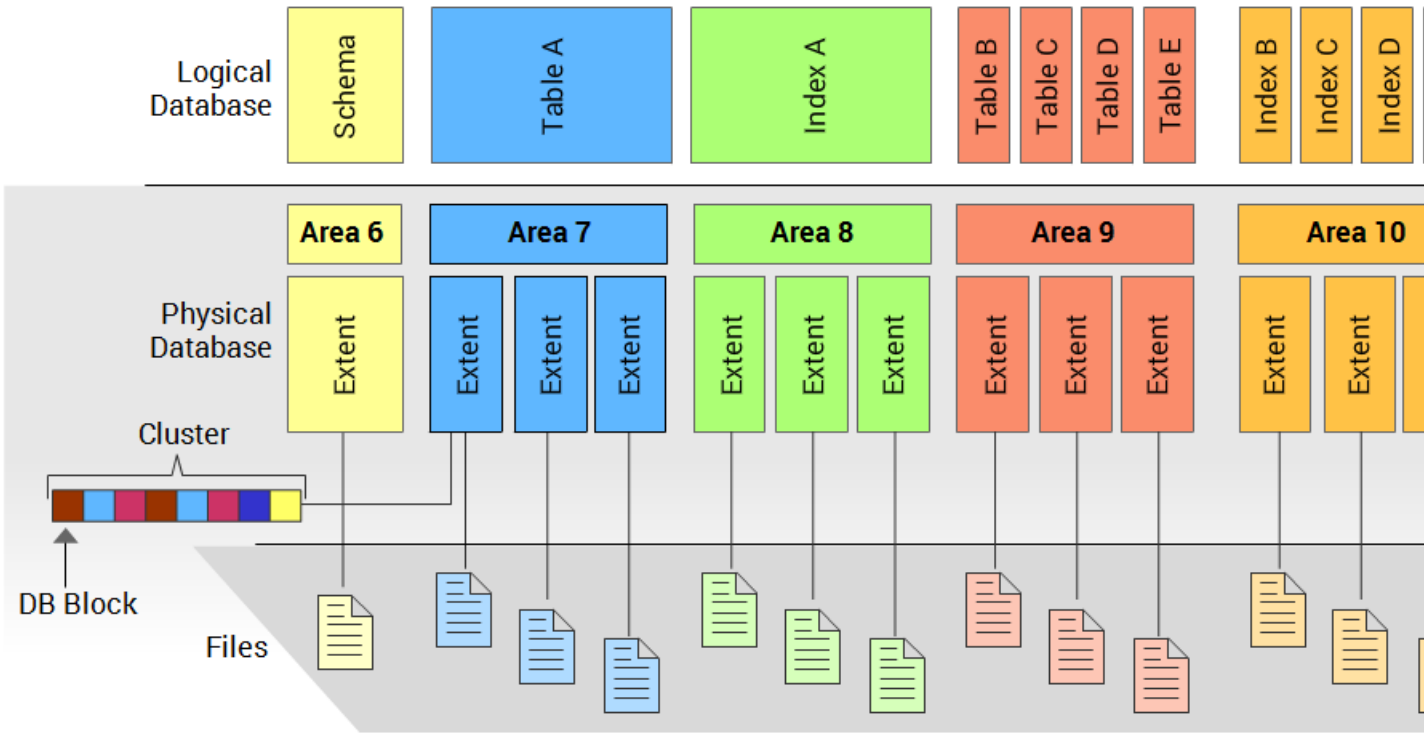
The foundation of your ABL application is a well-designed OpenEdge database. Well-planned logical and physical database components are critical to building applications that perform optimally. The guidelines in this section are recommendations for an average database. Each database configuration is different. Your database may have special considerations that require further tuning or an alternative configuration.

A well-planned logical and physical database provides the best foundation for application performance.

For those less familiar with database design concepts, see 'Introduction to Databases' in *OpenEdge Database Essentials*. For those more familiar with these concepts, a summary of considerations is provided with links to the guidelines for creating each component.

Identify database components

When you work with a database, it is important to identify the difference between the logical database components and the physical database components. Logical database components are important because they help determine how the ABL compiler queries and returns data to your applications. The related tables and indexes you create are used by the ABL code to efficiently retrieve your data. The physical database with its storage areas, clusters, blocks, and extents allow you to organize the data on the disk for efficient retrieval. This diagram shows both logical and physical components.



Design logical database components

Application developers rely on the foundation built by the database administrator's logical and physical database design choices. Database administrators design and build tables, indexes, and sequences to support an application developer's performance requirements. The choices made in the logical design support the coding guidelines for ABL performance content. Developers cannot build efficient queries if database administrators do not provide the necessary indexes.

For more information on the logical database components, see 'Logical database components' in *Overview of OpenEdge RDBMS Architecture*.

Apply indexing guidelines to the logical database

Create the right indexes for your application. Having the appropriate indexes defined allows the database to retrieve the right records with the minimal amount of disk I/O. On the downside, if you have too many indexes, then the database may spend time updating unnecessary indexes during record updates. For example, a one record update might need multiple I/O operations to maintain each index where the field is used. Database administrators and application developers must discuss the index requirements for overall application performance.

Design physical database components

Based on the logical database design decisions, a database administrator builds the physical database. The physical database components include:

- **Storage area**—The largest storage unit of a physical database that enables you to control the types, sizes, and locations of your database files. To design for performance, you can customize the number and type of storage areas, and group related information into areas that operate efficiently. For more information, see 'Storage areas' in *OpenEdge Database Essentials*, 'Guidelines for creating storage areas', 'Guidelines for creating storage areas for application data', and 'Specific guidelines for creating storage areas for application data' in *Create a Database*.
- **Extent**—Extents correspond to a physical file on a disk. Extents allow you to allocate space based on your application requirements. A storage area is made up of one or more extents. Sizing extents correctly in the design phase, reduces the need to dynamically increase the size of the extent on production systems later. To design for performance, you can customize the number and size of the extents. For more information, see 'Extents', 'Guidelines for creating extents' in *Create a Database* , and 'Creating fixed AI extents' in *Implementing Roll-Forward Recovery Using After-Imaging*.
- **Cluster**—A database structure that contains a range of contiguous database blocks that belong to the same object. Each extent consists of one or more clusters. To design for performance, you can customize the number of database blocks per cluster. Clusters improve performance of OpenEdge databases by reducing record fragmentation and record scatter. For more information, see 'Clusters' and 'Guidelines for specifying cluster size' in *Create a Database*.
- **Database block**—The smallest storage unit of a physical database. Each cluster consists of multiple database blocks. To design for performance, you can customize the number of database blocks per cluster. Your block size should match your OS default block size for efficiency. For more information, see 'Database blocks' and 'Guidelines for specifying database block size' in *Create a Database*.

Apply the best practices for creating an optimized physical database

The top tips for creating a physical database that performs well include:

1. Distribute application data, BI data, and AI data across different disks to avoid disk contention. Place frequently accessed areas on the fastest disk.
2. Place application data in separate storage areas. Having each area dedicated to its own table or set of tables allows finer grained performance tuning and maintenance.
3. Use fixed and variable extents effectively. For each storage area, use at least two fixed extents and one variable extent.
   - Fixed extents are pre-allocated and pre-formatted which makes writing to them efficient.
   - Variable extents increase in size dynamically and are best used for capturing overflow data. Variable extents tend to be less efficient because the space needs to be allocated and formatted.
4. Choose the best allowable number of records per block for each storage area. To choose the best number you must:
   a. Identify the mean record size for a representative table for the area.
   b. Calculate the optimal number of records per block.
   c. Set the best allowable number of records per block for the area.
5. Use Type II storage areas for application data because Type II storage areas support clustering. Clustering reduces block fragmentation and scattering, and allocates space more efficiently.

   Note: Type II storage areas are required for transparent data encryption, multi-tenant databases and table partitioning.

Here is a video that demonstrates designing an OpenEdge database for performance:

Designing an OpenEdge Database for Performance

Apply best practices to database creation

Progress recommends that you consider these best practices when you create your database:Based on expert advice, here's a short list of impactful decisions that can make when you initially create your database.

- **Place the database on a dedicated machine.**

  If the only thing running on the machine is your database server, you know immediately that the database is the source of the performance issues.

- **Use a database block size that matches your operating system's block size.**

  Larger block sizes provide greater I/O efficiency and are a more efficient use of storage. If your operating system supports 8-KB blocks, then use 8 KB for your block size. If your operating system uses 4-KB blocks, then use 4KB for your block size. For example, on Windows, use 4-KB block size to match the Windows' default block size. You get the best performance when the database block size matches the file system's page size or is a multiple of the file systems page size.

- **Determine when to use variable database extents.**

  In the past, there were warnings against using variable database extents because it takes time to extend whenever database needs to write more blocks onto disk. Using fixed database extents, which use pre-allocated space, are faster because the database doesn't need to lose time extending them on disk. Additionally, variable database extents are more likely to get very fragmented over time, whereas fixed database extents may allocate more continuous disk space. With the improvements in disk speed, this is no longer an issue.

- **Considerations for using RAID 5 and RAID 6 storage systems**

Progress previously cautioned against using RAID 5 and RAID 6 storage systems due to performance issues. These systems improved over time, and their reliability increased dramatically. RAID 5 and 6 are good options for read-oriented operations. RAID 5 and 6 can experience a degradation of performance for high-volume write operations if the volume of data being written exceeds the cache memory available to the RAID controller. The degree of performance loss could be significant. If you are doing online backups of the entire operating system, then these performance issues may appear.

For guided practice working with your database, Progress offers the following learning paths on the Progress Education Community:

- OpenEdge Database Administration: Core
- OpenEdge Database Administration: Advanced Enterprise Features

Code applications that perform well

To improve performance, you must increase your throughput. Writing efficient queries and profiling your application during development to identify potential bottlenecks can save you time and money when you incorporate that code in the larger testing, staging, and production environments.

During development, to increase the throughput in your code, you want to:

- Apply best practices for ABL on page 20
- Apply best practices for SQL on page 22
- Apply best practices for third-party data servers on page 22

Apply best practices for ABL

There are a lot of things ABL developers can do to improve the efficiency of their code. Writing efficient queries, managing record locks, and using the ABL Profiler to identify code that is taking the largest percentage of execution time are keys to improve ABL code efficiency during the development phase.

Manage data contention

When two or more users attempt to retrieve and possibly update the same data, the users may encounter data contention or what Progress refers to as record contention. ABL prioritizes data integrity through a combination of *record scope* and *transaction scope*.

For more information, see 'Transaction and record scope management considerations' in *Application Development with PAS for OpenEdge*.

To avoid contention and bottlenecks when you access data Progress recommends being explicit in your code when managing record locking and transaction scope.

- Use `NO-LOCK` on read-only record requests. `NO-LOCK` improves performance for other users who are able to update records without being blocked by unnecessary shared or exclusive locks. Add the `SHARE-LOCK` and `EXCLUSIVE-LOCK` options only when needed for data updates. Use comments to provide details to other developers about your record locking choices. For more information, see 'Record Scope' and 'Read records with NO-LOCK' in *Develop ABL Applications*.
- Use the `TRANSACTION` keyword to limit the size of transactions. Add comments to clearly identify the transaction scoping strategy. Limiting the size of transactions helps limit the length of the record lock and avoids others waiting for record locks to be released. For more information, see 'Ensure a small transaction size' in *Application Development with PAS for OpenEdge*.
- Review the ABL triggers. Triggers are blocks of ABL code that execute whenever a specific database event occurs. For example, when you write a record to a database, a `WRITE` event occurs. For more information, see 'Best practices for ABL triggers' in *OpenEdge Web Paper: ABL Database Triggers and Indexes*.
- Use the `compile LISTING` on your development code, to review your record and transaction scope. For more information, see 'Generate a procedure listing file' in *Develop ABL Applications* .

Write efficient queries

The ABL compiler is a rule-based optimizer that examines your queries and applies a series of rules to the query to determine the index to select for retrieving your requested records. The rules that the compiler uses are ***eliminations rules***. The compiler looks at all indexes and ***selects an index by elimination***. The compiler performs a pre-pass to eliminate indexes where the the first field in the index is not in the search or sort criteria. An index that may have a second field mentioned will not be as efficient as a leading component. Only leading commponent indexes are considered for use by the compiler.

For more information about how an index is selected, see 'General rules for choosing a single index' and 'How ABL chooses and brackets indexes to satisfy queries' in *OpenEdge Web Paper: ABL Database Triggers and Indexes*.

Use the `COMPILE` statement with the `XREF` option to identify the indexes used for the queries in your code. For more information, see 'COMPILE XREF option' in *Troubleshoot ABL Applications*.

Build an efficient temp-table

Temp-tables provide the same opportunity to optimize performance as database tables provide, only on a smaller scale. When you build a temp-table, do the following:

- Add only those fields necessary for the current task.
- Add only those related tables for the current task.
- Add only those indexes that are required for the current task.

Datasets are based on the underlying temp-table definitions, so make every effort to build efficient temp-tables.

Identify and avoid potential ABL bottlenecks

Development is the best time to start reviewing your code for potential bottlenecks. Developers can use the ABL Profiler utility to review their code for potential bottlenecks. For example, you can look for which part of the application is taking the largest percentage of your application processing time. Originally created for technical support engineers, profiling can identify problems on development, test, staging, and production environments.

Here is an introductory video that provides a tour of the ABL Profiler utility:

Get Started with Developer Studio

For more information, see, Using Profiling to Develop Better ABL Code , 'Profiler View' in *Progress Developer Studio for OpenEdge Online Help*, 'PROFILER system handle' in *ABL Reference*, and 'Profiler (-profile) startup parameter' in *Startup Command and Parameter Reference*.

Apply best practices for SQL

Progress offers SQL access to OpenEdge databases. This functionality allows you to integrate data stored in an OpenEdge database with applications written in SQL. The following content provides links to the guidelines for using SQL queries efficiently with OpenEdge databases.

Apply best practices for SQL query performance

The OpenEdge SQL engine contains a query optimizer that analyzes SQL queries and produces the best possible execution plan. The plan contains information such as which tables to access, in what order should they be accessed , and with which indexes should they be accessed. To produce a good query plan, the optimizer analyzes the query and considers many methods for each query execution step. For more information about the query optimizer, see:

- 'Understand optimization' in *OpenEdge® Data Management: SQL Development*
- What to look for in a SQL-92 query plan for performance issues
- Hints to improve SQL queries
- How to increase performance by running UPDATE-STATISISTICS

Apply best practices for third-party data servers

OpenEdge has the ability to connect to different data servers. This ability allows ABL application developers to seamlessly integrate ABL applications with systems of record that are stored in third-party databases. To use this technology efficiently, Progress provides guidelines for getting optimal performance from your applications that connect to third party data servers.

Apply best practices for third-party data server performance

OpenEdge has the ability to connect to different data servers. For more information, see:

- Guidelines for using data servers in *Use the Oracle Data Server*
- Writing queries for performance in *Use Microsoft SQL Server DataServer*
- Guidelines for using data servers in *Use the Oracle Data Server*
- Writing queries for performance in *Use Microsoft SQL Server DataServer*

IV

Configure

For details, see the following topics:

- Initialize ABL clients
- Initialize OpenEdge database servers
- Initialize Progress Application Server for OpenEdge

Initialize ABL clients

With your application code and physical database created, the next step is to deploy and initialize clients. Using procedure libraries and optimal client startup parameters can improve you application performance in test, staging, and production environments.

Store r-code in procedure libraries

You can organize and store r-code in an ABL procedure library. OpenEdge provides two types of procedure libraries: standard and memory mapped. A standard library contains r-code that executes in local memory. Using a procedure library this provides the following performance gains:

- Faster access to r-code
- Fewer file open and close operations
- Less I/O to temporary files

A memory-mapped library contains r-code that executes in shared memory. If your application is deployed on a machine that supports running multiple ABL clients or if your application is deployed on a Progress Application Server for OpenEdge, then deploy your application in memory-mapped procedure libraries. A memory-mapped procedure library has the following advantages:

- Fast access to r-code
- Less memory needed because multiple clients access the same r-code segments in shared memory

For more information, see 'Load and execute r-code from a standard procedure library' and 'Load and execute r-code from a memory mapped procedure-library' in *Manage ABL Applications*.

Manage ABL clients

On client machines, OpenEdge consumes system resources to execute ABL procedures, store variables and record buffers, sort records, and store temporary files. When you manage client performance, be aware of key performance factors such as procedure loading and execution, temporary file I/O, and sorting I/O. In addition to these factors, application design and coding can significantly affect client performance. The following table describes some key startup parameters that improve performance.

Table 1: Client startup parameters

Parameter	How performance is improved
`-asyncqueuesize`	Specifies the number of bytes that you want OpenEdge to allocate for the send and response queues for a PAS for OpenEdge connection. Use this parameter to maximize performance of asynchronous requests. OpenEdge creates the send and response queues for each connection; they are not used unless you make asynchronous requests to the PAS for OpenEdge instance. Choose a value based on the amount of data that you expect to flow between the client and the PAS for OpenEdge instance for these asynchronous requests. If you make a lot of calls, then set a higher byte level. If you do not make asynchronous requests, then set a lower byte value, to reduce the space allocated to queues that you will not use. For more information, see Async Queue Size (-asyncqueuesize).
`-Bt`	Specifies the amount of memory the temp-table database manages uses to manage the temp-table database blocks. If all the memory is used, then the temp-table database writes the blocks to disk using a `DBI` file. These writes to disk cause swapping and can negatively affect performance. This setting controls the temp-table database in the same way that `-B` is controls your OpenEdge database. For more information, see Number of buffers for temporary tables (-Bt).
`-D`	Controls the number of directory entries within which the AVM tries to manage the r-code files for your application. Each distinct `.r` file loaded by your application uses a directory entry, even when the `.r` file is loaded from a procedure library. If multiple persistent procedures or OOABL objects are running from the same `.r` file, then that `.r` file only consumes *one directory entry*. If you run out of directory entries, then the AVM looks for the least recently used entry that does not have an active program running it, and flushes that entry. If the `.r` file for that flushed entry is needed later, then It must be re-read into a new directory entry. If the directory entries are not managed properly, then this can lead to thrashing, continually flushing entries, and re-reading entries, which are bad for performance. For more information, see Directory Size (-D).

Parameter	How performance is improved
-mmax	Controls the execution buffer where r-code is managed. Each .r file is consists of multiple segments. These segments are loaded into memory for execution by the AVM. Some segments are only needed temporarily, for example, during initialization. The AVM locks the segments in memory only as long as they are needed. Any time compiled code is read into memory, it takes up room in the execution buffer. If there is not room within the -mmax setting, then the AVM swaps unlocked segments to disk. These unlocked segments are stored in the .rcd temporary file. **Note:** **1.** If you run out of this resource and the AVM cannot find any r-code segments to swap to disk, then the AVM automatically increases the -mmax setting. Swapping is bad for performance. **2.** Any r-code that resides in a Progress procedure Llibrary (.pl) is not swapped to the temporary .rcd file. The AVM uses the procedure library file as its swap space. **3.** Any r-code that is in a memory-mapped Progress procedure library (.pl) does not consume any -mmax space; instead, the AVM uses the (shared) memory allocated by the OS to manage the .pl file when it is first opened. For more information, see Maximum memory (-mmax).
-q	A quick request is useful for testing, staging, and production environments. This parameter tells the AVM to run compiled code without checking for a pre-compiled file that may have changed. Production environments only have compiled code, so suppressing the check avoids overhead on testing, staging, and production systems. For more information, see Quick Request (-q).
-reusableObjects	Improves the performance of applications that use ABL classes. When your application deletes an ABL object, either explicitly or through garbage collection, that object goes into the reusable object cache. The next time your application instantiates a new object of that same class, the AVM looks in the reusable object cache, and reuses the object instead of creating the object . This is particularly helpful if the object defines one or more temporary tables because temporary tables are more expensive to create from scratch. When an object goes into the cache, if the cache is full, then the least recently used object is deleted and is not available for reuse. For more information, see Re-usable Objects Cache (-reusableObjects).
-T	Tells the AVM where to store temporary files. These include the .rcd (r-code swap), .lbi (local before-image), .dbi (temp-table database), and other files. This is less of an issue for fast disks, but the performance loss is worse for slow disks like NFS-mounted disks. For more information, see Temporary Directory (-T) .

Initialize OpenEdge database servers

The next step is to initialize the OpenEdge database server using the best possible configuration. This is easier than ever, with the new database server defaults.

Tune operating system resources to improve database performance

Because system and application performance can vary greatly depending on the configuration, use the performance recommendations provided and make adjustments for your specific configuration.

For best performance, Progress recommends that you use a dedicated server for your OpenEdge database to prevent outside applications from negatively affecting your database performance. Additionally, you should review the impact your operating system settings can have on your overall database performance. For more information, see 'Tunable operating system resources' in *OpenEdge Database Essentials*.

Initialize your development, test, and staging database server

Once your application is ready to move to the test, staging, and production environments, you must set system, network and startup parameters that will optimize your performance.

- **Check system configurations**

  The potential for improving the performance of your OpenEdge database depends on your m. Some options might not be available on your hardware or operating system platform. For more information, see 'Manage database performance' in *Manage the OpenEdge Database*.

- **Set network communications parameters**

OpenEdge provides default startup parameters that help with performance. The following table describes the network communication parameters that are initialized to help your network run efficiently.

Table 2: Best practices for network communication parameters

Parameter	How performance is improved
`-Mm`	Default value is 8192. Tune this parameter to avoid record fragmentation. For more information, see Message Buffer Size (-Mm).
`-prefetchDelay`	Enabled by default. Use this parameter to delay sending the first network message for queries with prefetch capabilities. When enabled, `-prefetchNumRecs` and `-prefetchFactor` parameters determine the duration of the delay. For more information, see Prefetch delay (-prefetchDelay).
`-prefetchNumRecs`	Enabled by default. Use this parameter to increase the number of records placed in a network message before it is sent to a remote client. Enabling this parameter may cause a remote client to wait longer for full network messages that contain prefetched records, but the overall system performance is expected to improve by saving on network traffic. Another possible side effect is that an individual remote client see might see irregular responses from the server. For more information, see Prefetch Num Recs (-prefetchNumRecs) Search.
`-prefetchFactor`	Default value is 100. Use this parameter to establish a percentage of a network message that is required to contain prefetched data before sending the message to a remote client that is waiting for a response. If users experience unacceptable delays on a consistent basis, consider reducing the percentage to a network packet size that meets the expected response times. In most cases, 100 is preferred. For more information, see Prefetch factor (-prefetchFactor). Use Prefetch Priority (`-preFetchPriority`) to utilize a "pollskip" that adds n records to the network message of an in-process prefetch query without additional polling. Adding more than one record to the message at a time can greatly reduce the number of `poll()` calls being made on a large database configuration. The expected result is that reducing `poll()` calls improves overall system performance by decreasing the "System CPU" cycles allow for increased "User CPU" cycles. The pollskip only occurs if the previous `poll()` call returned nothing waiting.
`-preFetchPriority`	Default value is 1. Use this parameter to add *n* records to the network message of an in-process prefetch query without additional polling. Adding more than one record to the message at a time can greatly reduce the number of `poll()` calls being made on a large database configuration. The expected result is that reducing `poll()` calls improves overall system performance by decreasing the "System CPU" cycles allow for increased "User CPU" cycles. The pollskip only occurs if the previous `poll()` call returned nothing waiting. For more information, see Prefetch Priority (-prefetchPriority).

Here is a video that shows you how these parameters can improve performance:

Tuning Network Communication for Performance

Apply best practices for databases parameters and configurations

- **Put the before-image (BI) extent on a separate disk from the data extent, if possible**—Writing to the BI extent should be as efficient as possible with no interference from other activity. If you have many databases on the same machine, then you put the BI extent with the data extents.
- **Stripe data extents**—If you are using a RAID array, then there is no separate disk. i there are multiple RAID sets on your machine, then put it on a different RAID set on the same system.
- **Set the BI cluster size to 16 MB**—Because larger BI cluster sizes increase the duration of checkpoints, allowing more time for modified database blocks to be written to disk by the page writers, Progress recommends setting the cluster size to 16 MB. The duration for the last 8 checkpoints are displayed in PROMON's Checkpoints option. If the checkpoints are at least a minute long, then you do not need to make any changes. Longer checkpoints are acceptable, but the checkpoints are not needed. If they are shorter than a minute, then increase the cluster size.

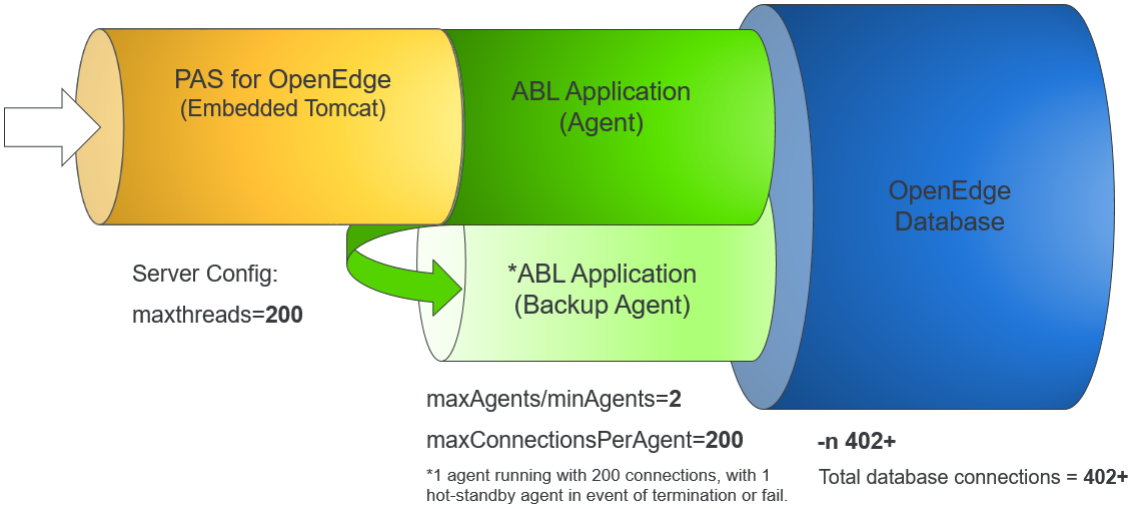
  If you have a workgroup database server license, then keep the cluster size small. Use a BI cluster size of 512 KB or less because a workgroup license does not include asynchronous page writers.
- **Leave -bibufs set to 64**—The `-bibufs` parameter specifies the number of before-image buffers. This parameter is useful only when running the before-image writer (BIW).
- **Use two disks for AI extents, with the extents alternating between them**—The purpose of after-image journaling is to provide a way to recover your database if the disk(s) holding your database fail. **Do not** store any AI extents on the same disks as the data extents. Alternating between two disks allows filled extents to be archived without slowing down the writing of the current extent. If you do not have enough disks, put all the AI extents on the same disk. If you are using a RAID array, then there is no separate disk, unless there are multiple RAID sets on the system. If there are multiple RAID sets, then put the AI files on a different RAID set on the same system.
- **Run at least one asynchronous page writer (APW)**—The asynchronous page writer's job is to write modified database blocks to disk so that the database server does not have to do it and so that the modified blocks do not have to all be written to disk at the very end of a checkpoint. PROMON reports these writes as buffers flushed. The value for buffers flushed should be a number less than 10 for almost every checkpoint.

Initialize Progress Application Server for OpenEdge

The next step is to initialize the Progres Application Server for OpenEdge instances using the best possible configuration. This is easier than ever, with the new Progress Application Server for OpenEdge default parameters and recommendations.

Take advantage of tested default parameters

The default settings for the Progress Application Server for OpenEdge are tuned and ready to use. The default values are shown in the following figure that demonstrates how these parameters streamlines access to the OpenEdge database ensuring that adequate resources are available for ABL application agents and backup agents:



The embedded Tomcat server has a `maxthreads` value of 200 to make sure there is the necessary throughput to avoid a bottleneck. The `maxAgents` value is set to 2 for the ABL application, to provide a single agent and a backup agent. Each agent has a `maxConnectionsPerAgent` of 200 allowing many threads to be available to process the requests. The `-n` parameter allows a total of 402+ connections to the database. These recommendations are to get the best performance for a single instance.

Take advantage of tuned property files

When you create a PAS for OpenEdge instance, the properties of the instance are automatically set to certain default values. Each PAS for OpenEdge instance has its own `/conf` directory that contains multiple configuration files, including:

- **`openedge.properties`**—Contains the properties that must be set to create an instance. See the `openedge.properties.README`.
- **`catalina.properties`**—Contains the Java properties used by an instance and its ABL web applications. For more information, see the `catalina.properties.README`.
- **`appserver.properties`**—Linked to the `catalina.properties file`. Any changes made to the `catalina.properties` files are dynamically reflected in `appserver.properties`. For more information, see the `appserver.properties.README`.
- **`jvm.properties`**—Contains a list of JVM startup command-line options.
- **`server.xml`**—Contains the server features for the instance

Manage Application Servers with Progress tools

OpenEdge provide a variety of tools to help you monitor and manage your Progress Application servers for OpenEdge instances. These tools include command line utilities and APIs for administrators who want to automate their monitoring and management tasks.

- **TCMAN**—A command-line utility for configuring and managing PAS for OpenEdge instances. This utility also enables you to view details such as operating system and server information, detailed server status, memory leaks, and HTTP session information. For more information, see TCMAN Reference in *PAS for OpenEdge Administrative Tools Reference*.
- **PASMAN**—A command-line utility that runs as a TCMAN wrapper from the $DLC/bin directory. This utility allows PAS for OpenEdge system administrators to manage multiple instances.
- **OpenEdge Management**—A browser-based administration tool that allows you to create and manage PAS for OpenEdge instances. This tool runs as either a Windows or a UNIX client of the AdminServer. OpenEdge Management can create, configure, start, stop, and delete a PAS for OpenEdge instance, and you can use OpenEdge Management to view log files. OpenEdge Management also allows you to monitor various aspects of your PAS for OpenEdge environment. In order to create and manage instances with OpenEdge Management, the `oemanager.war` file must be deployed to the instance. For more information, see 'Use OpenEdge Management to configure an instance' in *Manage Progress Application Server (PAS) for OpenEdge*.
- **OEPROP**—A command-line utility used to display and manage properties in the `instance-name/conf/openedge.properties` file.

  For more information, see "OEPROP" in *Manage Progress Application Server (PAS) for OpenEdge*.
- **REST APIs**—REST APIs give you visibility into your PAS for OpenEdge environment and let you take appropriate actions based on your findings. REST APIs are enabled by the `oemanager.war` web application and can be viewed through the Swagger UI. REST API reference documentation is available in the REST API Reference for oemanager.war topic in *PAS for OpenEdge Administrative Tools Reference*.

  For more information, see 'Use Swagger to explore management REST APIs' in *Manage Progress Application Server (PAS) for OpenEdge*.
- **OEJMX**—A command-line utility that can list all of the MBeans, including attributes and operations for a PAS for OpenEdge instance running on a local machine, and also run MBean queries. OEJMX does everything that deploying `oemanager.war` can do, but without opening access to the instance through the web server. For more information, see 'Use OEJMX to manage and monitor an instance' in *Manage Progress Application Server (PAS) for OpenEdge*.

Apply best practices to application server startup parameters

- **Tune asynchronous request**—For each connection, OpenEdge allocates memory for the send and response queues with a default size. If asynchronous requests exceed the send and response queue size, then OpenEdge stores additional queued requests in a temp-table associated with the client session. Therefore, performance can vary based on whether requests are stored in memory or in the temp-table.

  To help you maximize the performance of asynchronous requests, OpenEdge provides the Async Queue Size (`-asyncqueuesize`) client startup parameter. This parameter allows you to specify the number of bytes that you want OpenEdge to allocate for the send and response queues for a PAS for OpenEdge connection.

  The Async Queue Size parameter only applies when you create the PAS for OpenEdge connection on the client, and it applies equally to each PAS for OpenEdge connection. Although OpenEdge creates the send and response queues for each connection, they are not used unless you make asynchronous requests to the PAS for OpenEdge instance. Therefore, you can maximize the PAS for OpenEdge performance by choosing a value based on the amount of data that you expect to flow between the client and the PAS for OpenEdge instance for these asynchronous requests. Similarly, if you are not making asynchronous requests, then you can choose a low value, to reduce the space allocated to queues that you are not planning to use.

For more information, see the Async Queue Size (`-asyncqueuesize`) entry in OpenEdge Deployment: 'Startup Command and Parameter Reference' in *Application Development with PAS for OpenEdge*.

V

Tune

For details, see the following topics:

- Learn about tuning
- Tune ABL code
- Tune OpenEdge database servers
- Tune Progress Application Servers for OpenEdge

Learn about tuning

After the initial design and development of your OpenEdge application and the configuration of your OpenEdge database and Progress Application Server for OpenEdge instances, it is time to tune further. This phase requires you to iterate the following steps shown in the tuning cycle:

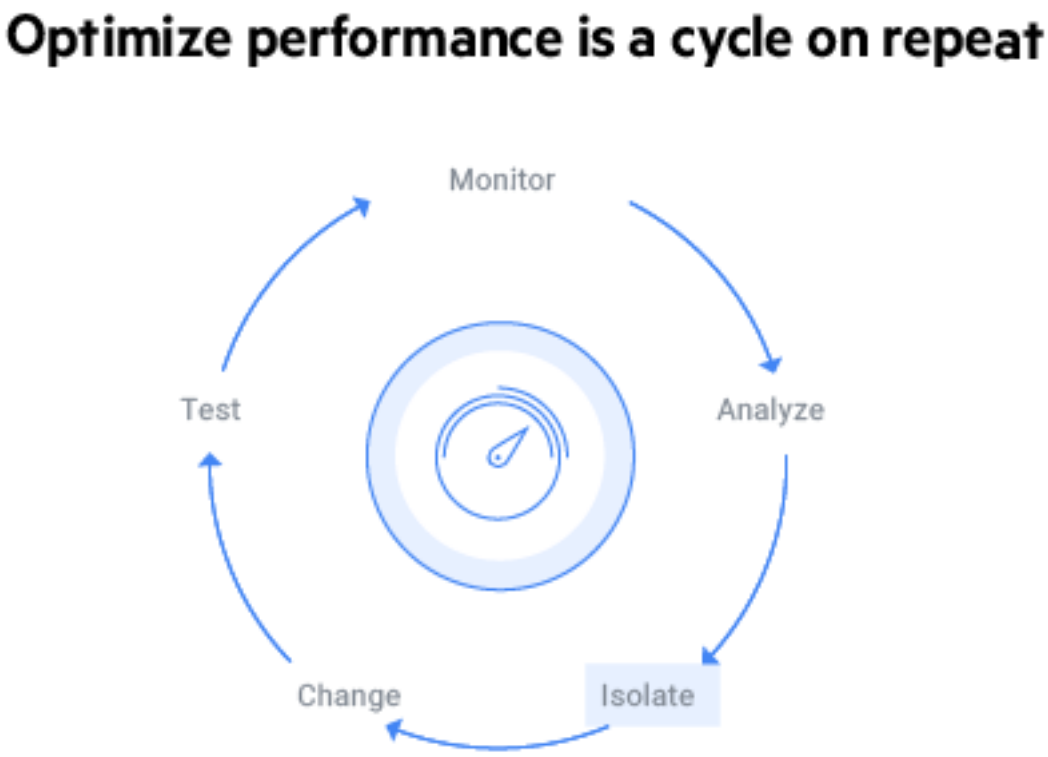


Figure 1: Optimization is a repetitive cycle

How can you tell if your application needs performance tuning? The leading indicators are that your users are experiencing unnecessarily long wait times or that your application is not responsive. These problems can be caused by increased load with some degree of decreasing performance. Modifying a system to handle a higher load is performance tuning.

For systematic tuning, follow these steps:

1. Monitor your performance: Assess the problem and establish numeric values that categorize acceptable behavior.
2. Analyze the performance based on the monitored measurements before you attempt to tune the system. In other words, have an accurate baseline of comparison to see if each change made improves the performance.
3. Isolate the improvement you want to test to a single change.
4. Change only one part of the system at a time to test whether the performance improvements.
5. Test the change and compare the performance against the performance data gathered in the analyze step to confirm that the system performance has improved. If the change makes the performance better, then adopt it. If the change makes the performance worse, then restore the original.

OpenEdge provides several tools to assist you in the critical phase of monitoring. The remainder of this topic describes tools and processes for use in fine-tuning your OpenEdge application. The key to success is appropriate monitoring and benchmarks are run to isolate, change, and test one option at a time. The content is grouped by ABL, OpenEdge database servers, and Progress Application Server for OpenEdge.

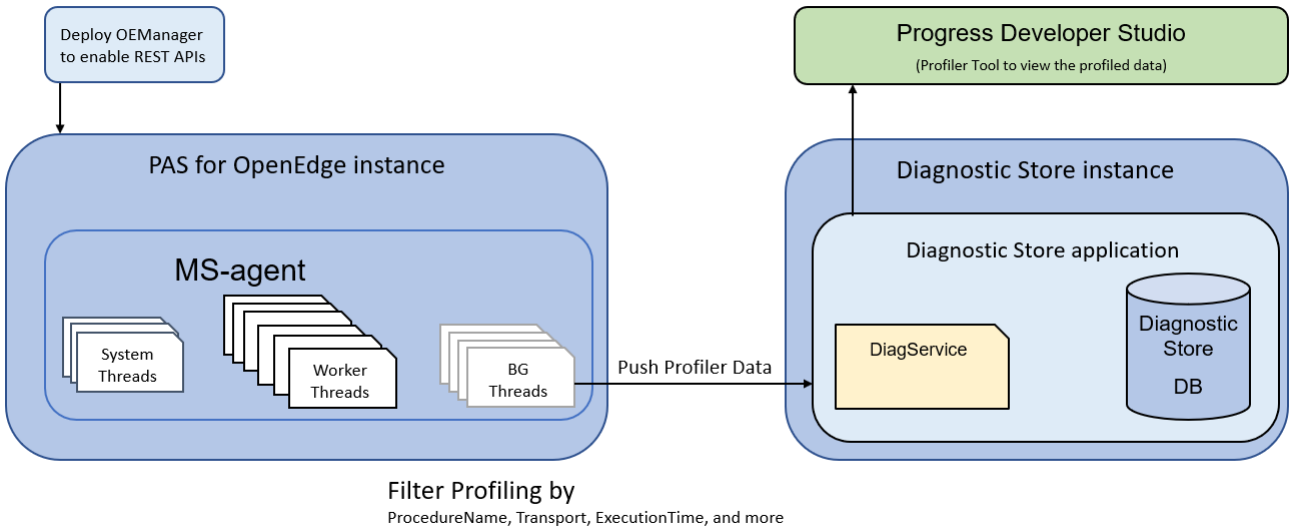
Tune ABL code

Tuning ABL application code in test, staging and production environments is typically done by tuning how well the code is performing after being deployed to a Progress Application Server for OpenEdge instance. You can perform server-side profiling, write custom logging programs using the `LOG-MANAGER` system handle, and enable client parameters to collect usage statistics to monitor how well your ABL code is performing.

Monitor ABL code performance with server-side profiling

Profiling your application during development is the recommended method for catching performance bottlenecks.

Figure 2: Server-side ABL performance profiling architecture



To use server-side profiling for a PAS for OpenEdge instance:

1. Enable server-side profiling.

   For more information, see 'PAS for OpenEdge server-side tuning' in *Manage Progress Application Server (PAS) for OpenEdge*.

2. The performance data is pushed to a diagnostic store, which is a web application deployed on a PAS for OpenEdge instance, that receives performance data from deployed ABL applications and stores that data in an OpenEdge database. You can either create a new PAS for OpenEdge instance for the diagnostic store, or you can use an existing instance.

   For more information, see 'Configure the diagnostic store' in *Manage Progress Application Server (PAS) for OpenEdge*.

3. View the data using a Profiler Viewer in Progress Developer Studio.

   For more information, see 'Profiler View' in *Progress Developer Studio for OpenEdge Online Help*.

Monitor ABL code performance with LOG-MANAGER

Some developers use the LOG-MANAGER to capture logging details and write debug messages to help identify performance issues. The LOG-MANAGER system handle turns on logging details for the following:

- **4GLTrace**—Logs of execution details:what was called, when, for how long, and what was passed in and out.

  For more information, see 'ABL trace logging' in *Troubleshoot ABL Applications*.

- **QryInfo**—Logs queries (each open query and FOR EACH block) executed in an application.

  For more information, see 'Query information log entry type and logging levels' in *Troubleshoot ABL Applications*.

- **4GLTrans**—Logs transaction processing and subtransactions in ABL procedures.

  For more information, see 'ABL transaction logging' in *Troubleshoot ABL Applications*.

Reviewing the logs can help reveal potential bottlenecks caused by poorly written code with poor index selections and open transactions. There are Knowledge Base articles with scripts to help search for useful information in the log files. For more information about managing logs, see Script to help manage abl client log sizes when extended logging is enabled and Script to help manage PAS for OpenEdge log sizes when extended logging is enabled.

Collect ABL client statistics

There are a set of client startup parameters available to collect usage statistics. With this information, you can identify client usage, segment statistics, and details about the call stack.

Table 3: Client startup parameters for tuning

Parameter	How performance is monitored
-y	Collects procedure access and usage statistics throughout the OpenEdge session. For more information, see 'Statistics (-y)' in *Startup Command and Parameter Reference*.
-yd	Writes segment statistics to the client monitor file (client.mon, by default). For more information, see 'Segment Statistics (-yd)' in *Startup Command and Parameter Reference*.
-yx	Collects procedure call statistics and writes them to an output file. For more information, see 'Statistics With Cross-reference (-yx)' in *Startup Command and Parameter Reference*.

Tune OpenEdge database servers

After the initial design and development of your OpenEdge database it is time to tune the database server for optimal performance. The tools for managing and monitoring your database servers are presented and tuning tips for database servers are provided.

Monitor OpenEdge database servers with Progress tools

The main tools for monitoring and managing performance include test text:

- **OpenEdge Management**—A browser-based management tool to monitor database servers, files, networks, OpenEdge components, and system resources in an OpenEdge environment.
- **PROMON**—An OpenEdge database administration utility to monitor and manage your database servers. PROMON cannot be run remotely, PROMON must run the same machine as the database server.

  For more information, see 'PROMON utility' in *Manage the OpenEdge Database*.
- **Virtual system tables (VSTs)**—A way to provide ABL and SQL applications access to the same database information that is collected with the PROMON utility. The virtual system tables, or schema tables, have no physical records until the database manager generates them at run time. You can write ABL or SQL application to retrieve information as run-time data for those table to monitor and collect data for analysis. You can archive performance data for further review and analysis. Using the data gather, you can write utilities for monitoring database performance that match your monitoring needs. With VSTs, you can perform remote monitoring; this is an advantage over monitoring with PROMON.

  For more information, see 'Access VST files with ABL or SQL' in *Manage the OpenEdge Database*.

  Progress provides a complete guide for managing database server performance. For more information, see 'Managing database performance' in *Manage the OpenEdge Database*.

Apply best practices for tuning database servers

These recommendations focus on things you may want to tune after your database server has been running for a while.

Note: These suggestions are dependent on your hardware and database server licensing.

Start your database server tuning with the following:

- **Review the indexes for your application.** It is important to index properly. As applications evolve, developers and database administrators must periodically review the indexes. Having the appropriate indexes defined allows the database to find records with a minimal amount of disk I/O. On the downside, if you have too many indexes, then the database spends time accessing unnecessary indexes during record updates. For example, one record update might require multiple I/O operations to maintain each index where the field is used.
- **Tune your database buffer parameters.** Tuning the number of available database buffers can improve your overall performance by reducing disk I/O. PROUTIL is used to set the buffer size online. Use PROMON to monitor how often the database is accessed to gain insight into the database buffer performance.

  This video demonstrates how to increase the buffer size online:

  Tuning Database Buffers Online
- **Stripe data extents on as many separate spindles as possible.** One disk can do one data transfer at a time. Two disk can do two transfers at a time. The more disks you have, the better. Be sure to evenly balance the I/O load among the available disks. The most effective way to balance the load across multiple disks is to create a stripe set that combines all the disks into one logical disk, and spread the data evenly across them. But, if you lose one disk, then you lose all the disks, so you must combine striping with mirroring to get reliability.

  Another way to balance the I/O load is to create data extents, with each extent containing a piece of the database. For example, if you have 4 disk drives, then create 16 extents, and put 4 on each disk drive. Put extent 1 on the first disk drive, extent 2 on the second, extent 3 on the third, and so on. A drawback to this manual striping is that as the database grows the balance is disturbed when you add extents.
- **Run the before-image writer (BIW).** The before-image writer's job is to write filled BI buffers to disk so the database server does not have to do it. This gives the database server more time to do useful work. With self-service clients, the server and the client are in the same process, but you still want the server to do useful work.
- **Manage before-image file growth** You can use the `-biscantime` parameter to tune the behavior for before-image space management. In some rare cases, BI files may keep growing even when the BI space management is running. This could happen under heavy database activities with very fast checkpoints. Reducing the value of `-biscantime` lets the before-image manager (BIM) process detect and reuse BI clusters more frequently at a cost of minor performance impact when compared to a bigger `-biscantime` value. Note that, in most cases, the default value of `-biscantime` (10) works well without any tuning. Setting the value of `-biscantime` to zero (0) turns off BI space management. (If you set `-biscantime` to 0 online, the before-image writer stays idle.) Whenever you set `-biscantime` to a non-zero value, you must start the before-image manager.

  For more information, see "Before-image scan time (-biscantime)" and "Start and stop a BIM" in *Manage the OpenEdge Database*.
- **If you use after-image journaling (you should, but not for performance reasons), run the after-image writer (AIW).**

  The after-image writer's job is to write filled AI buffers so the server does not have to do it. This gives the server more time to do useful work. With self-service clients, the server and the client are in the same process, but you still want the server to do useful work.

- **Set the AI block size to the same value as the BI block size.** Increasing the size of AI blocks lets the engine read and write more AI data at one time. This can reduce I/O rates on disks where the AI files are located. In general, the default AI block size (8 KB) is sufficient for systems with low transaction rates. However, if performance monitoring indicates that AI writes are a performance bottleneck and your platform's I/O subsystem can take advantage of larger writes, then increasing the AI block size may improve performance. A larger AI block size may also improve performance for roll-forward recovery processing.

  For more information, see 'Increase the AI block size' in *Manage the OpenEdge Database*.

- **Set -spin to fifty thousand**

  Finding the optimal value for spin is hard. By setting the `-spin` value higher, you can reduce the resource waits. With fast systems, use the number of processors times 20,000. Fifty thousand is a good place to start. If you set the `-spin` value higher and that value does not reduce the resource waits, then increasing it further can adversely affect the CPU utilization. To view the resource waits value:

  1. Access PROMON, and enter `R&D` at the main menu.
  2. Choose **Other Displays**.
  3. Choose **Performance Indicators**, to view the resource waits. Resource waits is the last item in the list.

- **Perform database dump and load.**

  Depending on the database structure, it is possible that over time records will fragment within database extent files, especially if there are many disparate tables located in the same area. Run a database analysis to determine if there is record fragmentation. All large tables with a Scatter Factor greater than 2 benefit from dump and load, or they could be moved into their own data area (with the appropriate records-per-block factor).

Monitor with third-party tools for your OpenEdge database server

ProMonitor is an OpenEdge performance monitoring tool available through Progress Professional Services.

Learn more about OpenEdge database performance tuning

Progress offers the Progress OpenEdge Database Performance Tuning course. This course is available as instructor-led training or self-paced training on the Progress Education Community.

Tune Progress Application Servers for OpenEdge

Monitor, tune and test your Progress Application Server for OpenEdge instances. Using Progress tools to monitor your Progress Application Servers for OpenEdge. Use OEJMX and REST APIs to monitor and manage your instances. Load balance your application servers with help from the OpenEdge HealthScanner. Analyze data collected by the deferred logging to decide how best to improve performance. Explore third-party tools including JConsole, JMX, and JConsole to monitor and manage your PAS for OpenEdge instances.

Monitor and manage Progress Application Servers for OpenEdge using Progress tools

In addition to the management tools introduced earlier, the following tools are useful to monitor performance on test, staging, and production application servers:

- **OEJMX**—A command-line utility that you can use to request information about Managed Beans (MBeans) and to run MBean queries. MBeans are the interfaces used for managing the resources of a PAS for OpenEdge server. The output of OEJMX requests and queries is written to a file.

  For more information, see 'Use OEJMX to manage and monitor an instance' in *Manage Progress Application Server (PAS) for OpenEdge*.

  You can use the following queries to get information about your PAS for OpenEdge server:

  - **Get active session information**—Provides details about client connections including `avgQueueDepth`, `maxConcurrentClients`, `maxQueueDepth`, reads, requests, and `timesQueued`. The `maxQueueDepth` can be used to determine whether there are any bottlenecks causing the requests to be queued. You may be able to improve performance by reducing the number of queue requests.

    For more information, see 'Get active session information query' in *Manage Progress Application Server (PAS) for OpenEdge*.

  - **Get transport metrics**—Provides detailed metrics for the APSV, SOAP, REST, and WEB transports in an ABL web application. Metrics include requests, `maxConnectTime`, `successfulRunRequests`, `successfulRequests`, `avgConnectTime`, and `accessTime` that identify areas for performance improvements.

    For examples of transport metrics, see 'Get transport metrics for an ABL web application' in *PAS for OpenEdge Administration Tools Reference*.

  - **Get metrics for a specific ABL application**—Provides information about the number of requests that were executed, failed, and queued while running on a specific ABL application. This metric shows whether clients timed out. If there are no read or write errors, then that indicates that the server is running well.

    For an example, see 'Get metrics for an ABL application' in *PAS for OpenEdge Administration Tools Reference*.

  - **Get hung client connections**—Provides information about the number of requests that are still running in a given ABL application, to identify client connections that are hung. Specify the number of milliseconds to elapse before sending information about hung client connections.

    For an example, see 'Get hung client connections' in *PAS for OpenEdge Administration Tools Reference*.

- **OpenEdge HealthScanner**—Checks the overall health of a server and assigns a score from 0-100% as well as a status code. System administrators configure an acceptable threshold for a server to be considered in good operational health. Servers with scores below the threshold can be taken out of service by an elastic load balancer (ELB).

  For more information, see 'Primary use case' in *Manage Progress Application Server (PAS) for OpenEdge*.

- **Deferred logging**—Gives system administrators the ability to run an on-demand monitoring check on the instance. The in-memory buffer is configured independently from the agent log, and automatically writes to the agent log file in the event of a crash or when it is called through an API. You can configure deferred logging to get additional information beyond the standard logging messages. When an agent crashes, the deferred logging buffer is printed to the agent log file. Alternatively, you can invoke an API to print the deferred logging buffer to the agent log file or clear the deferred log buffer.

  Here is a video that provides how you can use deferred logging:

  Deferred Logging in PAS for OpenEdge

For more information about the tools used in the video, see

- **REST APIs**—'Flush deferred log buffer to agent log file' and 'Reset deferred log buffer' in *PAS for OpenEdge Administration Tools Reference*

- **Swagger**—'Use Swagger UI to explore management REST APIs' in *Manage Progress Application Server (PAS) for OpenEdge*
- **OEJMX queries**—'Flush deferred log buffer to agent log file query' and 'Reset deferred log buffer query' in *PAS for OpenEdge Administration Tools Reference*

Tune by making one change at a time

A well-tuned PAS for OpenEdge instance balances the use of the OS, CPU, memory, and processes to service the largest number of concurrent client requests with the best response time. A poorly tuned PAS for OpenEdge instance can overload the server instance, multi-session agent, or OS which degrades the client response times or causes server outages.

Monitor PAS for OpenEdge with third-party tools

Java Monitoring and Management Console (JConsole)—An application included with Java, that can be used to monitor and manage PAS for OpenEdge instances. On production servers, JConsole is commonly used for monitoring and tuning, and for removing hung sessions. Use JConsole with a remote connection to a production server to avoid adding the overhead of JConsole into the values returned. The usual security concerns apply when you enable remote administrative access to an instance.

For more information, see 'Run JConsole remotely' in *Manage Progress Application Server (PAS) for OpenEdge*.

Java Management Extensions (JMX)— The technology provides dynamic access to applications and other resources by using Java objects called Managed Beans (MBeans). One or more MBeans instrument a resource through the use of the MBean's attributes, actions (defined as methods), and notifications. For more information about JMX and MBeans, see Introduction to JMX in the Java documentation.

VI

Learn more

For details, see the following topics:

- Related solutions

Related solutions

All of the recommendations for designing, developing, configuring, and tuning for performance are based on the expectation that customers are adopting good application development habits. Progress offers several other solutions guides that provide details about how to build modern, secure applications that are continuously available and can leverage a CI/CD pipeline for deployment. Other solution guides that you may find useful include:

- *Modernize OpenEdge Applications*
- *Authenticate Users Defined in Microsoft Active Directory*

Customer feedback

Progress collects customer feedback through the Customer Validation Program (CVP) and through our Knowledge Base and Community posts, to improve the overall performance of our products. Through these channels, Progress prioritizes the performance enhancements our customers want and need. Progress encourages you to provide feedback in the comments section, to improve this guide, and to add any tips you may use, to help others improve their performance.