WebSphere® Process Server

IBM

Version 6.0

# Tuning

**Note**

Before using this information, be sure to read the general information in “Notices” on page 17.

**29 September 2005**

This edition applies to version 6, release 0, of Process Server (product number 5724-L01) and to all subsequent releases and modifications until otherwise indicated in new editions.

To send us your comments about this document, email doc-comments@us.ibm.com. We look forward to hearing from you.

# Contents

**Tuning business processes . . . . . . . . . . 1**
Tuning long-running processes . . . . . . . . . . 2
  Balancing the hardware resources . . . . . . . . . . 2
  Specifying initial database settings . . . . . . . . . . 3
  Planning queue manager settings . . . . . . . . . . 7
  Tuning the application server . . . . . . . . . . 8
  Fine-tuning WebSphere MQ Messaging Provider . . . . . . . . . . 9
  Fine-tuning the database . . . . . . . . . . 10
Tuning microflows . . . . . . . . . . 11
Processes that contain task activities . . . . . . . . . . 11
  Assign staff dynamically only when necessary . . . . . . . . . . 12
  Reduce concurrent access to work items . . . . . . . . . . 12
  Reduce query response time . . . . . . . . . . 12
  Set the interval for refreshing staff queries . . . . . . . . . . 13
  Use the Everybody parameter with care . . . . . . . . . . 13
  Filter data . . . . . . . . . . 13
  Avoid scanning whole tables . . . . . . . . . . 14

**Notices . . . . . . . . . . 17**
Programming interface information . . . . . . . . . . 19
Trademarks and service marks . . . . . . . . . . 19

# Tuning business processes

Use this task to improve the performance of business processes.

After successfully running business processes, you can perform this task to improve performance.

1. Define how to measure the baseline performance, and which measurements you want to optimize.

   For example, for some business applications, it is preferable to reduce the response time for end-users under peak-load conditions. For other applications, the rate that the system can process transactions might be more important than the actual duration of each transaction.
2. Make baseline measurements.

   Make the baseline measurements under conditions of load, time-of-day, and day-of-week that are appropriate for tuning your application.
3. Tune the processes.

   Depending on whether your application uses long-running processes or microflows, perform one of the following steps:
   - To tune long-running processes, perform the steps that are described in "Tuning long-running processes" on page 2. These processes are characterized by needing external stimuli or human interaction. Their performance therefore depends on the performance of the Business Process Choreographer database and the messaging service.
   - To tune microflows, perform the steps that are described in "Tuning microflows" on page 11. These processes tend to run for only a short time. They use the database only for audit logging, if enabled, and to retrieve the template information. They do not use messaging support for storing persistent data. These processes involve no human interaction.
4. **Optional:** Tune the application.

   Many different options are available to achieve the same functionality in an application, and some of them are more efficient than others. Identify and review any performance-critical code. Maximize asynchronicity, and ensure that actions are not unnecessarily serialized. Consider the overhead of serializing and deserializing data that is passed along a chain of activities. Consider shortening timeouts that do not result in error conditions. Identify opportunities to cache the results of database queries.
5. **Optional:** Review the current configuration for performance bottlenecks that can be eliminated.

   Possibilities to consider include:
   - Installing more processors, more memory, and faster disks.
   - Storing the database logs on different physical disks from the data, and distributing the data on several disks.
   - If the database supports table spaces, such as DB2® or Oracle, distributing the INSTANCE table space across two or more disks or a striped RAID array.
   - Not mixing application server and non-application server-related workloads on the same logical disk.
   - Using DB2, rather than Cloudscape™, for optimal performance.

- Using WebSphere® MQ, rather than the embedded messaging provider, for better performance.
- Acquiring enough licenses for the message queueing system and the database system to run on enough CPUs.
- Using a local queue manager on any application server whose CPU is not overloaded, if you have a central queue manager setup (one queue manager for each cluster). This setup eliminates the network delays that are associated with process navigation messages, reduces the workload on the central queue manager, and reduces the load on the network connection, all of which can help to speed up other processes that rely on a heavily loaded central queue manager.

6. Repeat the benchmark measurements under similar load conditions to those of the baseline measurements.

   Keep a permanent record of the application performance measurements to objectively measure any future changes in performance.

The business processes are configured to run measurably faster.

## Tuning long-running processes

Use this task to improve the performance of long-running business processes.

Long-running processes can include user-interaction, asynchronous invocations, multiple receives, picks, and event handlers, for example; they use database and messaging subsystems for storing persistent states. You can improve the performance of long-running processes by tuning the database, the messaging system, and the application server settings.

- "Balancing the hardware resources"
- "Specifying initial database settings" on page 3
- "Planning queue manager settings" on page 7
- "Tuning the application server" on page 8
- "Fine-tuning WebSphere MQ Messaging Provider" on page 9
- "Fine-tuning the database" on page 10

**Related tasks**

"Tuning microflows" on page 11
Use this task to improve the performance of microflows.

### Balancing the hardware resources

Use this task to improve the performance of long-running business processes, by balancing the hardware resources.

Before you start to tune the system, verify that the computer used is well balanced, so that the available CPU, memory, and disk space resources have the right relationship. A computer with fast CPUs, but low memory or low disk-access performance, is hard to tune.

1. Allocate enough disks.

   For long-running processes, good disk performance and multiple, fast disk drives are as important as a fast processor and a sufficient amount of memory.

   Separate disks provide independence of operation and are not affected by other operations. A striped RAID array distributes disk operations over several disks. This can provide improved performance over an equal number of single disks. Use a striped RAID array to overcome bottlenecks on single disks.

For a system that runs long-running processes and does not use a RAID array, set up six or more disk drives, as follows:

- One disk for the operating system and the swap space (page file on Windows®, paging space on AIX®, swap space on Solaris, paging space on HP-UX).
- One disk for WebSphere Application Server and its transaction log.
- One disk for the transaction log of the message queuing system.
- One disk for the persistent queue storage of the message queuing system.
- One disk for the transaction log of the database management system.
- One or more disks for the Business Process Choreographer database in the database management system. If more disks are available, distribute the INSTANCE tablespace on two or more disks.

2. Allocate enough memory.

   The amount of memory to allocate depends on the platform:

   - For a Windows system with 3 GB of physical memory, a local database, and a local WebSphere MQ queue manager, use the following memory allocation:
     – 256 MB for Windows systems
     – 1024 MB for WebSphere Application Server
     – 256 MB for WebSphere MQ, if WebSphere MQ is used
     – 1.5 GB for the database
   - For an AIX system with 8 GB of physical memory, a local database, and local WebSphere MQ queue manager, use the following memory allocation:
     – 512 MB for AIX systems
     – 1024 MB for WebSphere Application Server
     – 512 MB for WebSphere MQ
     – 3 GB for the database

     **Tip:** To help ensure optimum performance, do not allocate all memory to the database, because the files you use and file caching, for example, also consume memory. Avoid situations in which data must be swapped to disk because insufficient high-speed memory is available.
   - For the application server, when running Business Process Choreographer, allocate more than the default amount of memory. Tuning the heap size of the application server is described in "Tuning the application server" on page 8.

3. Move workload to other machines.

   Consider which applications or subsystems can be moved to other machines.

Your computer hardware is now well balanced.

## Specifying initial database settings

Use this task to specify initial DB2 database settings. Note that this information is provided only as an example.

**Attention:** The following information relates to the Business Process Choreographer database. For information about tuning a WebSphere default messaging database, see Tuning and problem solving for messaging engine data stores in the WebSphere Application Server Network Deployment information center.

To achieve good database operation, specify the initial database settings. You will fine-tune the settings later, in “Fine-tuning the database” on page 10.

1. Separate the log files from the data files.

   Putting the database log file on a disk drive that is separate from the data tends to improve performance, provided that sufficient disk drives are available. If few disk drives are available, distributing the table spaces, as described in the previous section, is usually more beneficial than putting the database log on a separate drive.

   For example, if you use DB2 on a Windows system, you can change the location of the log files for the database named `BPEDB` to the `F:\db2logs` directory, by entering the following command:

   ```
   db2 UPDATE DB CFG FOR BPEDB USING NEWLOGPATH F:\db2logs
   ```

2. Create table spaces.

   After you create the database, explicitly create table spaces. Example scripts to create table spaces are provided by Business Process Choreographer in the `ProcessChoreographer` subdirectory of your WebSphere Application Server installation. Customize these scripts to accommodate the needs of a particular scenario. Your goal, when creating the table spaces, is to distribute input and output operations over as many disk drives as possible that are available to DB2. By default, these scripts create the following table spaces:

   **AUDITLOG**
   Contains the audit trail tables for processes and tasks. Depending on the degree of auditing that is used, access to tables in this table space can be significant. If auditing is turned off, tables in this table space are not accessed.

   **COMP**
   Contains the compensation tables for business processes from Business Process Choreographer Version 5. Depending on the percentage of compensable processes and activities, the tables in this table space might require high disk bandwidth. If compensation is not used within business processes, the tables in this table space are not used.

   **INSTANCE**
   Holds the process instance and task tables. It is always used intensively, regardless of the kind of long-running process that is run. Where possible, spread this table space over several disk drives.

   **SCHEDTS**
   Contains the tables that are used by the WebSphere scheduling component. Access to tables in the scheduler table space is usually low, because of the caching mechanisms used in the scheduler.

   **STAFFQRY**
   Contains the tables that are used to temporarily store staff query results that are obtained from staff registries like Lightweight Directory Access Protocol (LDAP). When business processes contain many person activities, tables in this table space are frequently accessed.

   **TEMPLATE**
   Contains the tables that store the template information for processes and tasks. The tables are populated during the deployment of an application. At run time the access rate is low. The data is not updated, and only new data is inserted during deployment.

   **WORKITEM**
   Holds the tables that are required for work item processing. Work items

are used for human task interaction. Depending on the number of human tasks in the business processes, access to the tables in this table space can vary from a low access rate to significantly high access rate. The access rate is not zero, even when no explicit human tasks are used, because work items are also generated to support administration of long-running processes.

To create a database for high performance, perform the following actions:

a. Create the database.

   On Windows, you can specify a destination drive. The command creates a database on the destination drive in a directory named the same as the default DB2 instance of the machine. So, for example, if the database is to be created on drive `D:`, and the local default instance is DB2, the database data goes to `D:\DB2`. Thus, to create a DB2 database in the `D:` directory for Business Process Choreographer, enter the following command:

   ```
   CREATE DATABASE BPEDB ON D: USING CODESET UTF-8 TERRITORY en-us;
   ```

   On UNIX® and Linux®, enter the following command:

   ```
   CREATE DATABASE BPEDB ON /wasdbfs USING CODESET UTF-8 TERRITORY en-us;
   ```

   where `/wasdbfs` specifies a directory.

b. Create the table spaces on the disks if the database supports table spaces.

   For example, the following script is based on the `createTablespaceDb2.ddl` file that is located in the `ProcessChoreographer` subdirectory of your WebSphere Application Server installation. In the following example, it creates the database, table spaces, and tables that use three disk drives on a Windows system:

   ```
   - Scriptfile to create tablespaces for DB2 8.1 and 8.2
   - to run the script call
   -      db2 connect to BPEDB
   -      db2 -tf createTablespaceDb2.ddl
   CREATE TABLESPACE AUDITLOG
     MANAGED BY SYSTEM
     USING( 'F:/BPEDB_TS/AUDITLOG' );

   CREATE TABLESPACE COMP
     MANAGED BY SYSTEM
     USING( 'D:/BPEDB_TS/COMP' );

   CREATE TABLESPACE INSTANCE
     MANAGED BY SYSTEM
     USING( 'E:/BPEDB_TS/INSTANCE' );

   CREATE TABLESPACE STAFFQRY
     MANAGED BY SYSTEM
     USING( 'D:/BPEDB_TS/STAFFQRY' );

   CREATE TABLESPACE TEMPLATE
     MANAGED BY SYSTEM
     USING( 'D:/BPEDB_TS/TEMPLATE' );

   CREATE TABLESPACE WORKITEM
     MANAGED BY SYSTEM
     USING( 'D:/BPEDB_TS/WORKITEM' );

   -- start import scheduler DDL: createTablespaceDB2.ddl
   CREATE TABLESPACE SCHEDTS MANAGED BY SYSTEM USING( 'D:/BPEDB_TS/SCHEDTS' );
   -- end import scheduler DDL: createTablespaceDB2.ddl
   ```

c. Create the Business Process Choreographer tables.

Use the script for your database that is provided in the `ProcessChoreographer` directory. For example, for DB2, use the `createSchemaDb2.ddl` file.

3. Tune the database.

   Use a capacity planning tool for your initial database settings.

   If you are using DB2, start the DB2 configuration advisor from the DB2 Control Center, by selecting **DB2 configuration advisor** from the context menu of the Business Process Choreographer database. Do the following actions:

   a. Allocate memory to DB2.

      For **Server**, allocate to DB2 only as much memory as is physically available to it without swapping.

   b. Specify the type of workload.

      For **Workload**, select **Mixed** (queries and transactions).

   c. For **Transactions**, specify the length of the transactions and the estimated number of transactions to be processed each minute.

      Select **More than 10**, to indicate that long transactions are used.

      Then, in the **Transactions per minute** field, select the estimated number of transactions processed each minute. To determine this number, assume that each activity in the process has one transaction. The number of transactions performed in one minute is then as follows:

      *number of transactions performed each minute = number of processes completed each minute * number of activities in each process*

   d. Tune the database for faster transaction performance and slower recovery.

      For **Priority**, select **Faster transaction performance**.

   e. Tune the database initially with no data in it.

      For **Populated**, select **No**.

   f. Tune the parallel connections setting.

      For **Connections**, specify the maximum number of parallel connections that can be made to the application server. Guidelines for determining this value are as follows:

      - The number of database connections required is determined by the number of Java™ DataBase Connectivity (JDBC) connections to the WebSphere Application Server. The JDBC connections are provided by the JDBC connection pool, which is in the WebSphere Application Server. For *p* JDBC connections, *p* * 1.1 database connections are required. How to estimate a realistic value for *p* is described in “Tuning the application server” on page 8.
      - If Business Process Choreographer and the database are installed on the same physical machine, Business Process Choreographer needs no remote database connections. However, because remote connections might be required for remote database management, specify a low value, rather than zero.
      - If Business Process Choreographer and DB2 are installed on separate machines, set the number of remote applications in accordance with the rule previously described for local connections.

   g. Lock the rows you want to read.

      For **Isolation**, select **Read stability**. This isolation level is required for Business Process Choreographer.

   The configuration advisor displays suggested changes. You can either apply the changes now, or save them to a file to apply later.

Your long-running processes are running as fast as possible under the current environment and loading conditions.

## Planning queue manager settings

Use this task to plan queue manager settings.

To achieve the best performance for long-running processes, tune the message queuing system for maximum performance of persistent messages.

If you use default messaging, follow the instructions given in Service integration in the WebSphere Application Server Network Deployment information center, to set up and tune the data stores for the messaging engines.

If you use the IBM® WebSphere MQ messaging product, rather than the default messaging services, complete the following steps.

1. Tune MQ parameter settings.

   Tune the following MQ parameter settings:
   - Log file pages
   - Log buffer page
   - Log primary files
   - Log secondary files
   - Log default path
   - Maximum channels
   - Channel application bind type

   The default locations for both the persistent queue data and the MQ logs is the MQ installation directory. Put the data storage for the persistent queues and the WebSphere MQ logs on different disk drives. By changing the path to the log file to refer to another disk drive, you can change the location for the MQ logs. Make these changes before you create the queue managers for Business Process Choreographer.
2. Tune WebSphere MQ service properties for Business Process Choreographer.

   These values must be set before you create the queue managers that are used by Business Process Choreographer. Set each parameter to its maximum value, as shown in the following table:

*Table 1. Tuning WebSphere MQ service properties for Business Process Choreographer*

| Parameter | Value | Comment |
|---|---|---|
| Log file pages | 16384 | On Windows systems, not all versions of WebSphere MQ support setting the number of log file pages to 16384 by using the MQ administration tools. In this case, change the value of the Windows registry key: `HKEY_LOCAL_MACHINE\SOFTWARE\IBM\MQSeries\CurrentVersion\Configuration\LogDefaults` to: `16384` |
| Log primary files | 10 | |
| Log secondary files | 53 | |

*Table 1. Tuning WebSphere MQ service properties for Business Process Choreographer (continued)*

| Parameter | Value | Comment |
|---|---|---|
| Log buffer pages | 512 | |

3. Tune the queue manager properties.

   Specify the queue manager properties for the maximum number of channels and the type of binding for the channel application, as shown in the following table:

*Table 2. Tuning queue manager properties*

| Queue manager properties | Value |
|---|---|
| Maximum channels | Use the default |
| Channel application bind type | FASTPATH |

Your queue manager is operating optimally.

## Tuning the application server

Use this task to tune the application server.

Before you start this task, you must have specified the initial settings for the database, as described in "Specifying initial database settings" on page 3.

To ensure that the business process container can preform optimally, you need to adjust the server settings.

1. Estimate the application server resources that you need for Business Process Choreographer.

   Each business process container requires the following resources to run properly. The names shown are the default names that are used in a standard installation of Business Process Choreographer.

   The activation specifications for the process server are found in the administrative console under **Resources** → **JMS Providers** → **Default messaging** → **JMS activation specification**.

      BPEApiActivationSpec

      BPEInternalActivationSpec

      HTMInternalActivationSpec

   The concurrency of the process navigation depends on the number of concurrent endpoints set in BPEInternalActivationSpec and HTMInternalActivationSpec (for processes containing human tasks).

   One data source, to read and write business process state information to a database: `jdbc/BPEDB`

   For each possible concurrent navigation step in a long-running process instance:

   - One Java database connectivity (JDBC) connection to BPEDB
   - One concurrent endpoint session on the BPEInternalActivationSpec
   - Two JDBC connections to the messaging data store (JMS): one each for get and put
   - Additional database connections if you are using the Business Process Choreographer API to start processes or to interact with running processes

- Additional JMS sessions, to support system overloads situations where additional messages are generated and need to be served

**Restriction:** The number of concurrent endpoints sessions (*m*) must not exceed 90% of the number of JDBC connections to the process server database BPEDB and must not exceed half the number of JDBC connections to the messaging databases.

For example, to process up to 30 navigational steps concurrently, you need 30 concurrent endpoints on the BPEInternalActivationSpec. This requires 30 * 1..1 JDBC connections to the process server database and at least 60 JDBC connections on the messaging database

Your estimate for the maximum number of parallel JDBC connections that are required at any time provides the value for *p*, which is used in the following steps.

2. Tune the JDBC settings for the process server database.

   Change the JDBC settings, as follows:

   a. For the **JDBC provider**, set **Max Connections** to the value *p*. The value of *p* must not be greater than the number of connections allowed by the database.
   b. For the **Data source**, set the SQL **Statement cache size** to 500.
3. Tune the **JDBC** settings for the messaging database.

   To change the settings, use the administrative console. Set to at least twice the number of concurrent endpoints set for the BPEInternalActivationSpec. Allow for at least 10% expansion.

   Here are some guidelines for the size of the server heap:

   - 256 MB is too low, and results in poor performance.
   - 512 MB is adequate as an initial heap size for many systems.
   - 1024 MB is a reasonable upper limit. Make sure that the total amount of memory that is allocated to applications is no larger than the amount of physical memory available. Otherwise, paging significantly reduces your system performance and might affect the stability of your system.
4. Rebalance the memory allocation.

   If your database is running on the same computer as your application server, you will probably need to subtract any extra memory that is allocated for heap size from the memory that is allocated to the database. Because all remaining physical memory should be allocated to the databases, use the following equation to rebalance the allocation for the database:

   *memory for the database = physical memory - memory for operating system - memory for application server*

The application server is tuned for improved performance.

## Fine-tuning WebSphere MQ Messaging Provider

Use this task to improve the rate at which WebSphere MQ processes messages if you use IBM WebSphere MQ Messaging Provider.

If, after you complete the tuning actions, less than 80% of your CPU capacity is used, or if the CPU usage graph shows bursts of activity, experiment with the custom property `non.asf.receive.timeout`. To change this property, use the administrative console.

CPU usage is improved by smoothing peaks, or bursts, of activity.

Create the custom property `non.asf.receive.timeout`, on the message listener service of the application server, with a value of 2000.

This setting switches the message listener service to non-ASF mode.

The rate at which WebSphere MQ processes messages is improved.

## Fine-tuning the database

Use this task to fine-tune the database.

The business process container and business processes must be running.

A common problem is that the database runs out of lock list space, resulting in lock escalation, which severely impacts performance. Depending on the structure of the business processes run, you might therefore need to customize the settings of certain performance-related parameters in your database management system.

**Note:** If you are not using DB2, refer to your database management system documentation for information about monitoring the performance of the database, identifying and eliminating bottlenecks, and fine-tuning its performance. The rest of this topic offers advice for DB2 users.

1. Tune the lock list space, to help ensure optimum performance.

   Check the `db2diag.log` file for your DB2 instance. Look for entries like the following example:

   ```
   2005-07-24-15.53.42.078000 Instance:DB2 Node:000
   PID:2352(db2syscs.exe) TID:4360 Appid:*LOCAL.DB2.027785142343
   data management sqldEscalateLocks Probe:4 Database:BPEDB

   ADM5503E The escalation of "10" locks on table "DB2ADMIN.ACTIVITY_INSTANCE_B_T"
   to lock intent "X" has failed. The SQLCODE is "-911".
   ```

   This type of message indicates that the parallelism for business process applications has improved to the point where the number of available locks is now too small. Increase the LOCKLIST value to approximately 10 * *p*, where *p* is your estimate for the maximum number of parallel JDBC connections that are required at any time. For example, if you sized your Business Process Choreographer database, BPEDB, with a value of *p*=50, enter the following command:

   ```
   db2 UPDATE DB CFG FOR BPEDB USING LOCKLIST 500
   ```

2. Fine-tune your DB2 performance.

   If you used the DB2 configuration advisor, your database throughput is already good. You can, however, further improve the performance, as follows:

   - Follow the best practices for database tuning that are described in the DB2 online documentation, books, and articles.
   - Use DB2 monitors, and examine the `db2diag.log` file for more information about bottlenecks within the database.
   - Following are some DB2 parameters whose tuning can improve performance:

     AVG_APPLS

     DBHEAP

     LOG_FILSIZ

     MINCOMMIT

     NUM_IO_CLEANERS

     PACKCACHESIZ

SHEAPTHRES
SOFTMAX
SORTHEAP

Your long-running processes are running as fast as possible under the current environment and loading conditions.

## Tuning microflows

Use this task to improve the performance of microflows.

Microflows run in memory, without any user-interaction or persistent messaging support. Database access is required only to retrieve template data and if audit logging is enabled for the microflow. Microflows are synchronous, and the entire processing occurs in a single transaction. The performance of microflows depends largely on the following general server tuning parameters:

1. Tune the JVM heap size.

   By increasing the Java heap size, you can improve the throughput of microflows, because a larger heap size reduces the number of garbage collection cycles that are required. Keep the value low enough to avoid heap swapping to disk. For guidelines on the size of the server heap, see the relevant step in “Tuning the application server” on page 8.
2. Tune the Enterprise JavaBeans™ (EJB) cache size.

   This parameter specifies the number of buckets in the active instance list within the EJB container. Setting this parameter equal to the maximum expected number of active instances improves performance. If this value is small compared with the expected maximum number of active instances, the container attempts to passivate some active instances, resulting in lower performance.
3. Tune the thread pool size of the object request broker (ORB).

   If several clients are to be able to connect simultaneously to the server-side ORB, increase the ORB thread pool size to improve performance.

Your microflows are running as fast as possible under the current environment and loading conditions.

## Processes that contain task activities

Although many choices you make for your workflow system might appear individually to be the best possible choices, some choices conspire to reduce the system efficiency of the system.

Some ways to improve the efficiency of your workflow system are as follows:

- “Assign staff dynamically only when necessary” on page 12
- “Reduce concurrent access to work items” on page 12
- “Reduce query response time” on page 12
- “Use the Everybody parameter with care” on page 13
- “Filter data” on page 13
- “Avoid scanning whole tables” on page 14

## Assign staff dynamically only when necessary

Dynamic staff assignments use more system resources than do static staff assignments. To help improve system efficiency, use static staff assignments.

You can use custom properties and process variables, which are also known as *%-variables*, to achieve dynamic behavior from a human task in your process. This function enables you to delay the decision about the group of potential owners from modelling time to run-time.

However, when you assign staff dynamically, the best results must be compute for each instance. This approach is expensive of resources.

When you use static staff assignments (when you set the potential owner to group MyGroup at modelling time, for example), the results of this staff query are cached. They can then be reuse by other instances of the process.

Use static staff assignments, in preference to dynamic staff assignments, whenever possible, to achieve the best performance.

## Reduce concurrent access to work items

When two or more people try to claim the same work item, only one person will succeed. The other person is denied access.

Only one person can claim a work item. When several people attempt to work with the same work item at the same time, the probability of collision increases. Collisions cause delays, because of lock waits on the database or rollbacks. Some ways to avoid, or at least reduce, the incidence of collision are as follows:

- If concurrent access is high, limit the number of users who can access a particular work item to 10 through 50. This should reduce the incidence of the message that the work item has already been claimed.
- Avoid unnecessary work item queries from clients, by using intelligent claim mechanisms. For example, you might take one of the following steps:
  - Try to claim another item from the list if the first claim is unsuccessful.
  - Always claim a random work item.
  - Reduce the number of people in each group.
  - Limit the size of the work item list, either by using a custom property in the where clause of the query or by setting a low threshold.
  - Minimize or avoid dynamic staff queries.
  - Use a client caching mechanism for work item queries, to avoid running several queries at the same time.

## Reduce query response time

Reduce the time that the database takes to respond to queries.

When you use a custom client, make sure that the queries set a threshold. From a usability viewpoint, retrieving hundreds or thousands of items is typically undesirable, because the larger the number of database operations, the longer the task takes to complete, and because a person can manage only a small number of results at a time. By specifying a threshold, you minimize database load and network traffic, and help to ensure that the client can present the data quickly.

A better way to handle a query that returns a large number of items might be to rewrite the query, to return a smaller result set of items. You can do this by querying work items for only a certain process instance or work items with only a certain date.

You can also reduce the query result by using a filter criterion. For information about filtering data, see “Filter data.”

## Set the interval for refreshing staff queries

Set the interval at which the database refreshes staff queries.

Staff queries are resolved by the specified staff repository. The result is stored in the Business Process Choreographer database. However, these staff queries are not resolved each time they are used. Instead, a timestamp indicates when a staff resolution ″expires″ and the query is to be refreshed. You can define this timestamp by setting the StaffQueryResultValidTimeSeconds property in the bpe.properties file.

If your staff repository changes infrequently, consider using a large value for this property and refreshing your staff queries on demand by using the refreshStaffQuery.jacl script.

## Use the Everybody parameter with care

Use the **Everybody** parameter in a staff query only when you are developing and testing business processes.

When you use the **Everybody** parameter in a staff query, you might achieve some slight performance improvement, because no attempt is made to verify the staff query result by reading data from the configured staff repository. Everybody can then see everything. This visibility can create issues with access control. Furthermore, when a large number of concurrent users use the **Everybody** parameter, the risk of collision increases, because the number of work items that can be claimed is high and the result set for a query is large.

Use the **Everybody** parameter in only the following situations:
- While developing and testing business processes
- When few users are using the system concurrently, and you consider authorization unimportant

## Filter data

Improve database performance when filtering tasks or process instance data.

Process variables are stored in a serialized form that is not suitable for queries. Although you can retrieve instance data for every item in the work list, to do so requires that all objects in the work list be materialized. Such operations place unnecessary demands on system resources.

You can, instead, query process data as follows:
- Query on a parameter.

  You can map this parameter to a custom property of the business process or human task. The process or task can have any number of custom properties. You can include custom properties when you run a work item query. However, each custom property makes the query more complex and requires more time to complete. Therefore, to avoid degrading system performance unnecessarily, limit the number of custom properties in any query.

If you have multiple custom properties and the database performance is degraded, try to encode the parameters in a single custom property. A custom property can be up to 254 characters long. You can encode the query in name–value syntax.

**Example**

```
setCustomProperty(
     piid,
     "MyCustomPropertyName",
      "priority=1 duedate=1099066021605 locale=en_us")
```

In the query you would state something like:

```
PROCESS_ATTRIBUTE.VALUE like 'priority=1%'
```

or

```
PROCESS_ATTRIBUTE.VALUE like '%duedate=1099066021605%'
```

**Attention:** Queries for strings that start with a % character require a full index or table scan. Therefore, put the properties used most often at the first position, to enable efficient database queries through the index.

- Query on parameters that are passed to a business process or human task by its input message and are then constant.

  You can query these parameters by mapping them to the business process description or human task description.

  **Example**

  The modeler defines the business process or human task description as containing a string of the pattern *%variableName.memberName%*:

  ```
  %inputMessage.language%
  ```

  The variable `%inputMessage.language%` is resolved when the business process instance or human task instance is created at run-time. You can then query the corresponding data by means of the process or task description.

  In the query for a process description, you would state something like:

  ```
  PROCESS_INSTANCE.DESCRIPTION='%locale=en_us%'
  ```

  In the query for a task description, you would state something like:

  ```
  TASK_DESC.DESCRIPTION='%locale=en_us%'
  ```

## Avoid scanning whole tables

When you use the query application programming interfaces (APIs), to list the objects in the database, you can specify filters that narrow the results you want to retrieve. In these filters, you can specify the values and ranges of object attributes.

When database queries are processed, the filter information is translated into WHERE clauses in a Structured Query Language (SQL) statement. These WHERE clauses map the object attributes to column names in the affected database tables.

If your query specifies a filter that does not translate to an indexed table column, the SQL statement will probably cause the table to be scanned. This scanning impacts performance negatively an increases the risk of deadlocks. Although this performance impact can be tolerated if it happens only a few times a day, it could adversely affect efficiency were it to occur several times a minute.

In such circumstances, a custom index can dramatically reduce the impact. In a real customer situation, a custom index helped to reduce the API response time from 25 seconds to 300 milliseconds. Instead of reading 724 000 rows of the database table, only six rows had to be read,

Depending on the filter criteria that you specify, some columns might not be included in an index. If you think that this is the case, and if a table scan is used, resulting in slow query performance, check the access path of the statement, using DB2 Explain, for example. If necessary, define a new index.

# Notices

## Programming interface information

Programming interface information, if provided, is intended to help you create application software using this program.

General-use programming interfaces allow you to write application software that obtain the services of this program's tools.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

**Warning:** Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

## Trademarks and service marks

Process Server, Version 6.0

Printed in USA