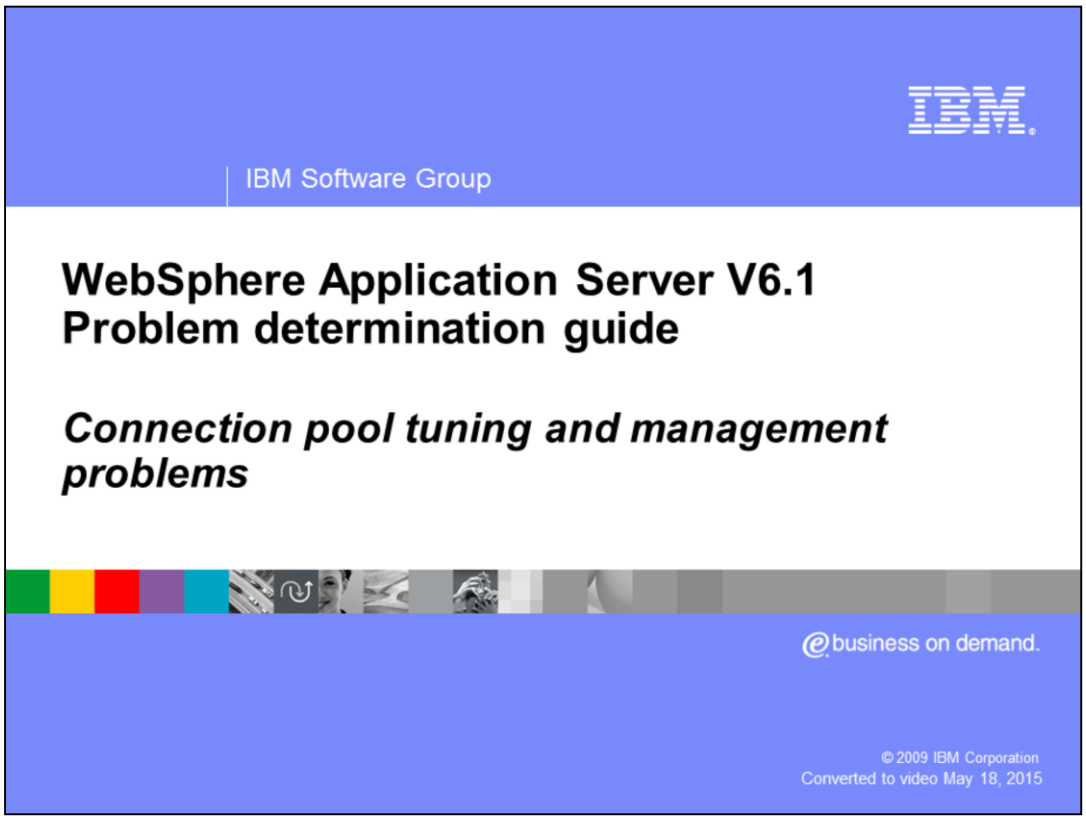
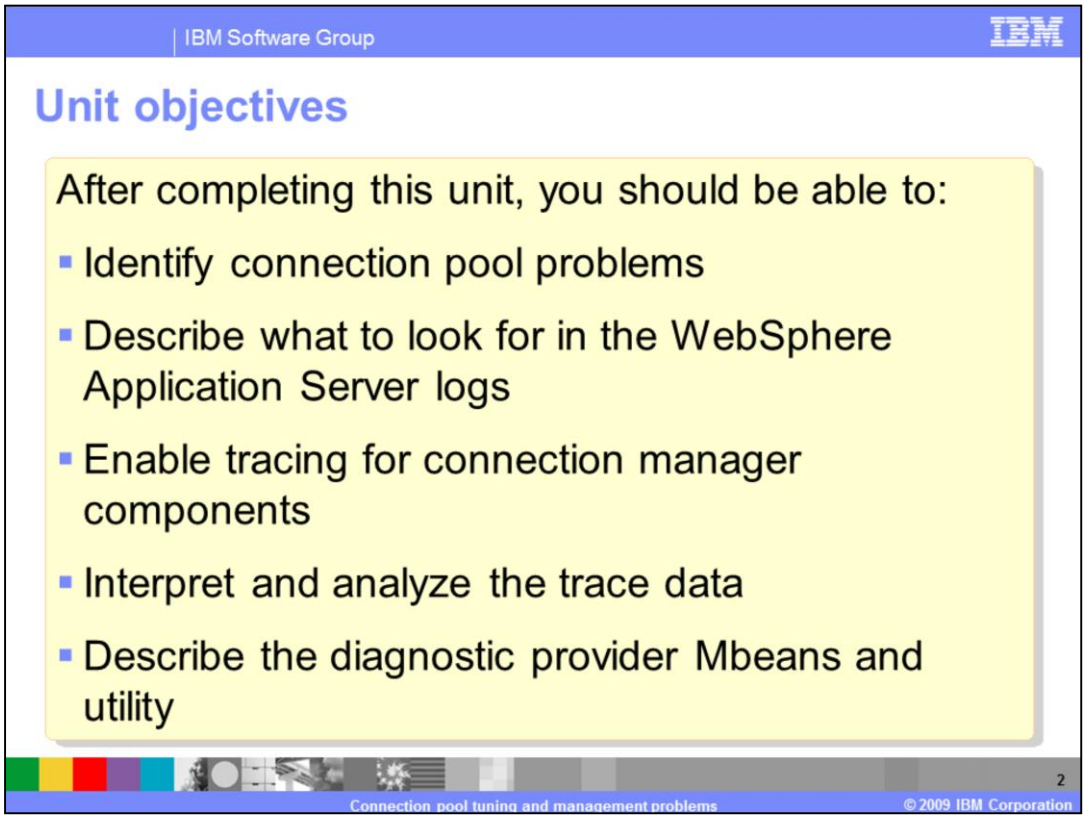
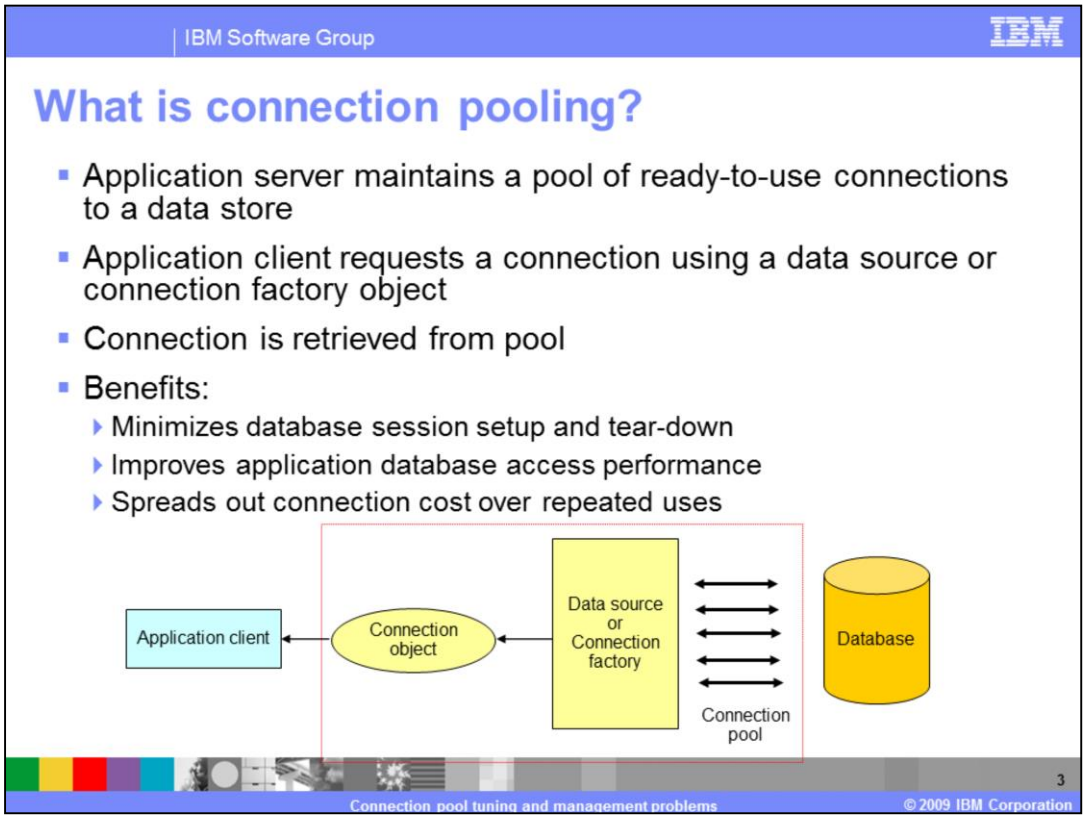


This unit describes how to troubleshoot connection pool tuning and management problems in WebSphere® Application Server.



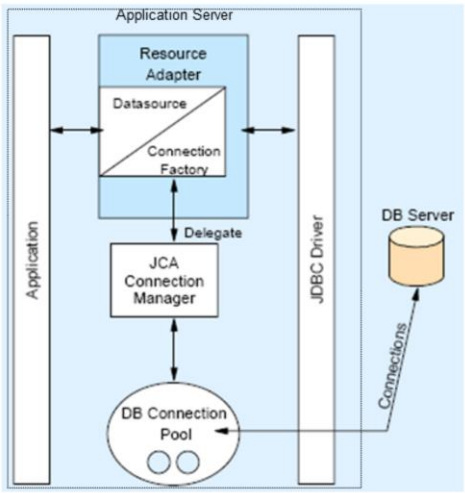
After you complete this section, you will be able to identify problems in connection pools, know how to use Tivoli® Performance Viewer, to monitor a connection pool, understand connection pool tracing data, and perform the problem determination tasks to troubleshoot a connection pool problem.



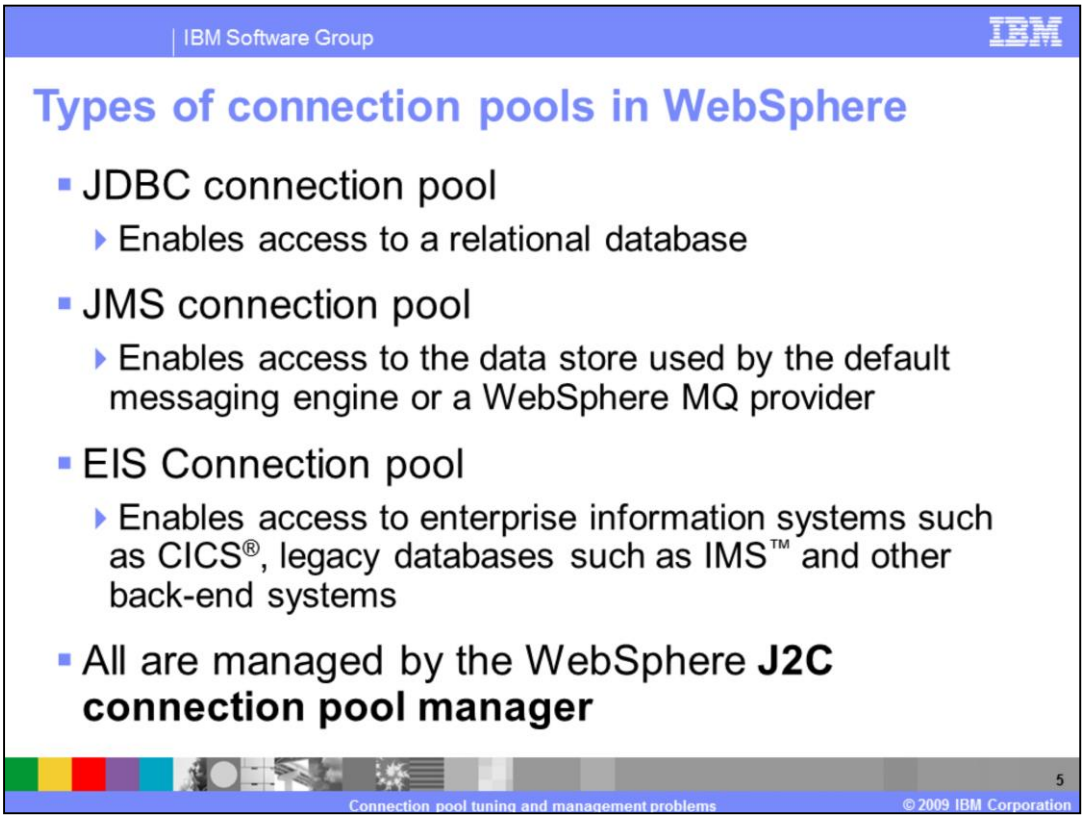
Applications need to acquire a connection to a data store each time they want to retrieve information from the store. The average connection object is one to two megabytes in size and contains a great deal of information about the connection context. Creating and terminating those connections is actually a very time consuming operation and so it can easily slow down the application. To fix this problem, WebSphere Application Server uses a pool of connections that can be reused by applications. This allows the cost of establishing each connection to be spread out across several requests and can significantly improve performance. An application that needs to access the data store will request a connection from the pool and return the connection when it is finished. An example of this work flow is illustrated on the slide.

JCA connection pooling architecture

- WebSphere Application Server implements the Java EE Connector Architecture (JCA) V1.5 specification
- Connection pooling is supported by two components:
  - JCA connection manager (called J2C connection pool manager in WebSphere)
  - Relational resource adapter

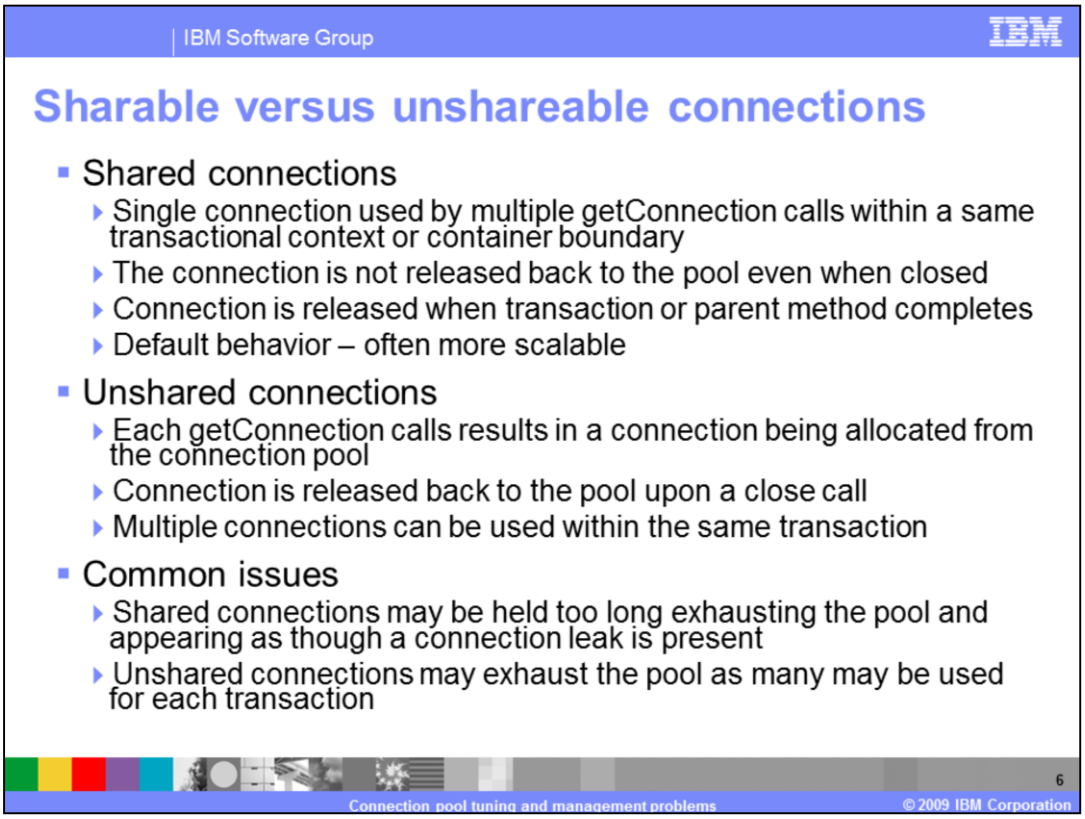


WebSphere Application Sever implements connection pooling by following the Java™ EE Connection Architecture version 1.5 specification. There are several objects involved in pooling connections but they can be grouped into two basic components, the JCA connection manager and the Relational Resource Adapter. An application that needs a database connection will go to the Resource Adapter to retrieve a Connection Factory. The Connection Factory will delegate the request to the correct Connection Manager. The Connection Manager is responsible for either returning an existing connection from the pool of available connections or creating a new one if none are available. The application releases the connection when it is finished interacting with the database and the Connection Pool will return it to the pool.

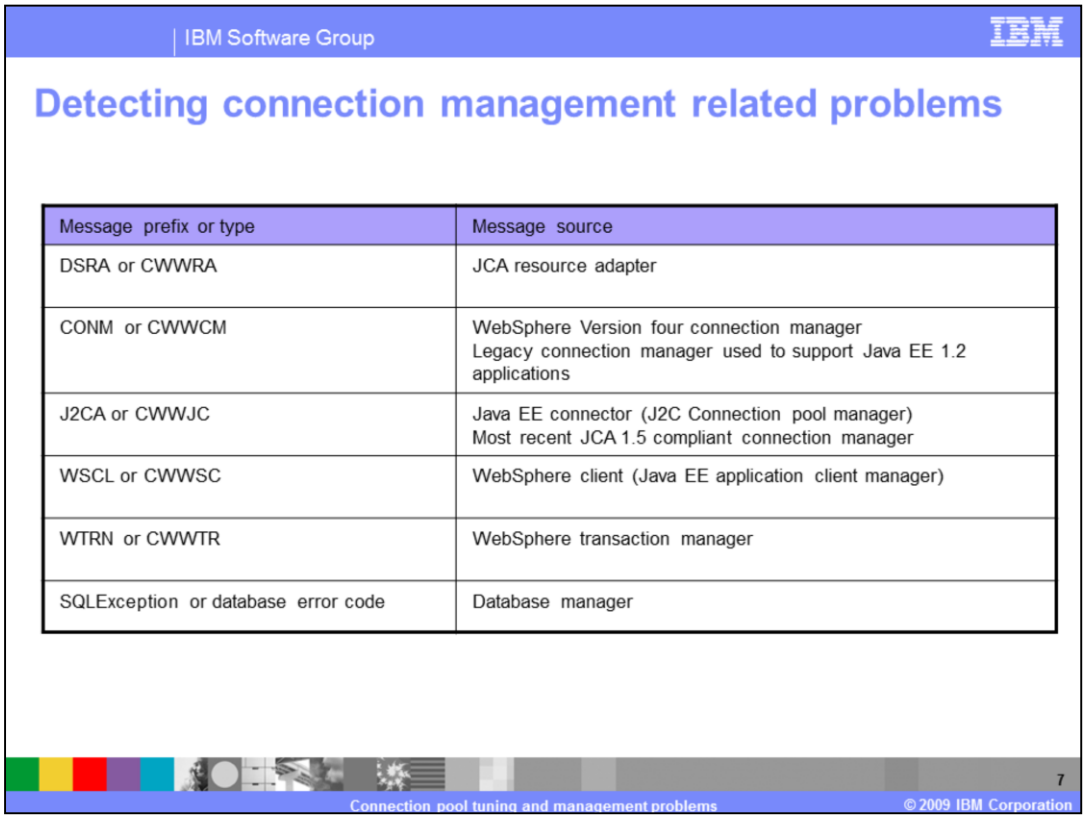


WebSphere Application Server uses a J2C connection pool manager to maintain three different connection pools. The JDBC connection pool is used to manage connections to relational databases such as DB2®. This pool can be adjusted by navigating to the JDBC Providers panel of the WebSphere administrative console. There is also a JMS pool for managing requests for connections to Service Integration Bus messaging engines or WebSphere MQ Queue Managers. This pool is maintained in the JMS Providers panel of the WebSphere administrative console. Finally, WebSphere provides an EIS connection pool that manages connections to CICS and legacy back-end

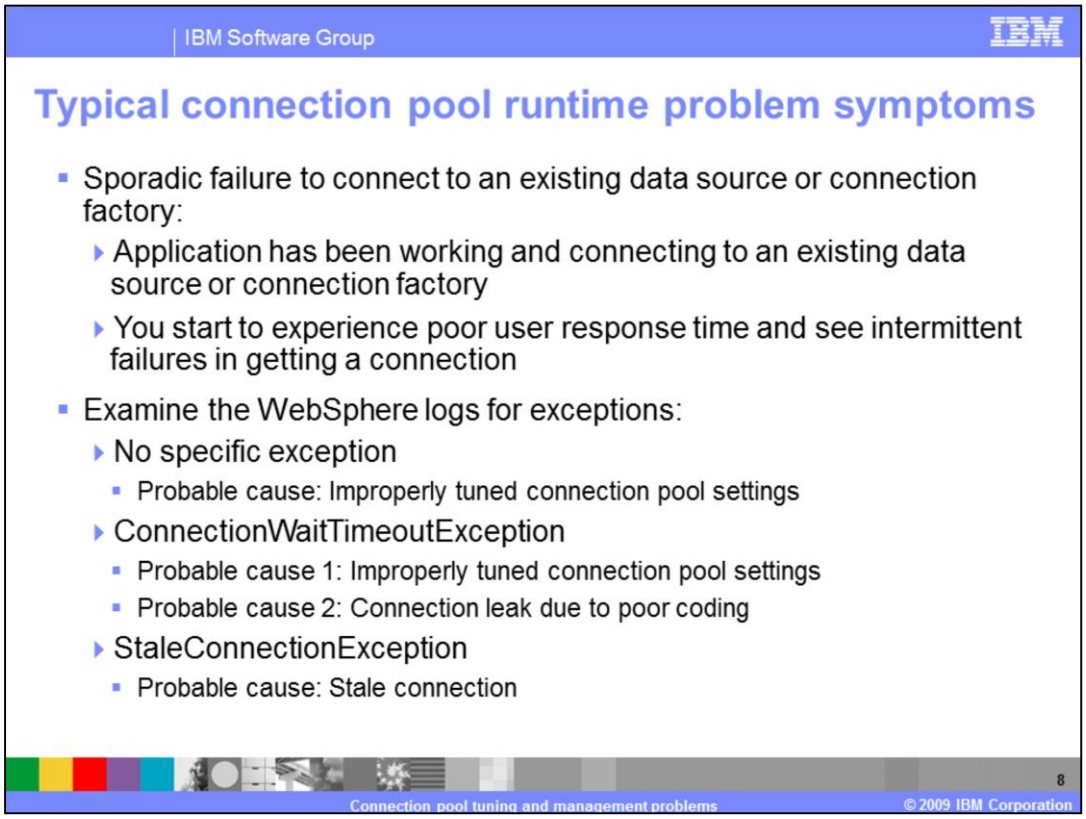
systems such as IMS. This connection pool is controlled through the WebSphere administrative console un the Resource Adapters panel.



Connection pooling in WebSphere Application Server is managed according to the Java Connector Architecture (JCA) and enables the use of shareable or unshareable connections. In WebSphere Application Server, connections are shareable by default. The use of shareable connections means that, if conditions allow it, different requests for connections by the application may actually receive a handle for the same physical connection to the resource. The benefits of this are improved performance and a reduction in the number of physical connections that need to be managed. When the application closes a shareable connection it is not returned to the free pool. Rather, it remains ready for another request within the same transaction for a connection to the same resource. Shareable connections obtained by an application within a local transaction containment are kept reserved in the shared pool for use within that local transaction containment until the it ends, even if the application explicitly closes the connection. This behavior is beneficial to many applications, but can have unintended consequences for others.

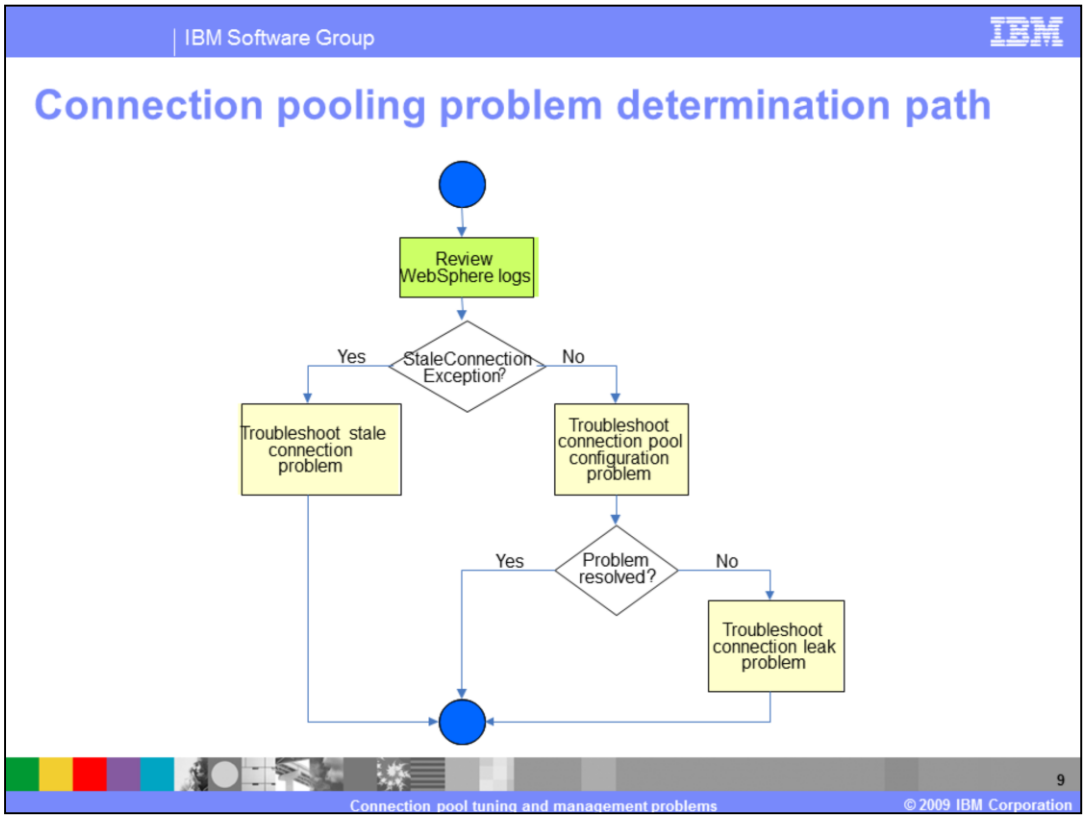


Connection related log messages are sent to the SystemOut and SystemErr logs in the appropriates profile‘s log directory. The System logs are the best starting place to determine if you have a problem in one of the connection pools. WebSphere Application Server maintains connection pools for multiple connection types so it stands to reason that there are several different messages that pertain to the different connection types. The various connection based prefixes are listed on this slide along with the connection types that they pertain to. The best way to determine if you are experiencing connection problems is to search the logs for any of the message prefixes then correlate them to the appropriate message source.



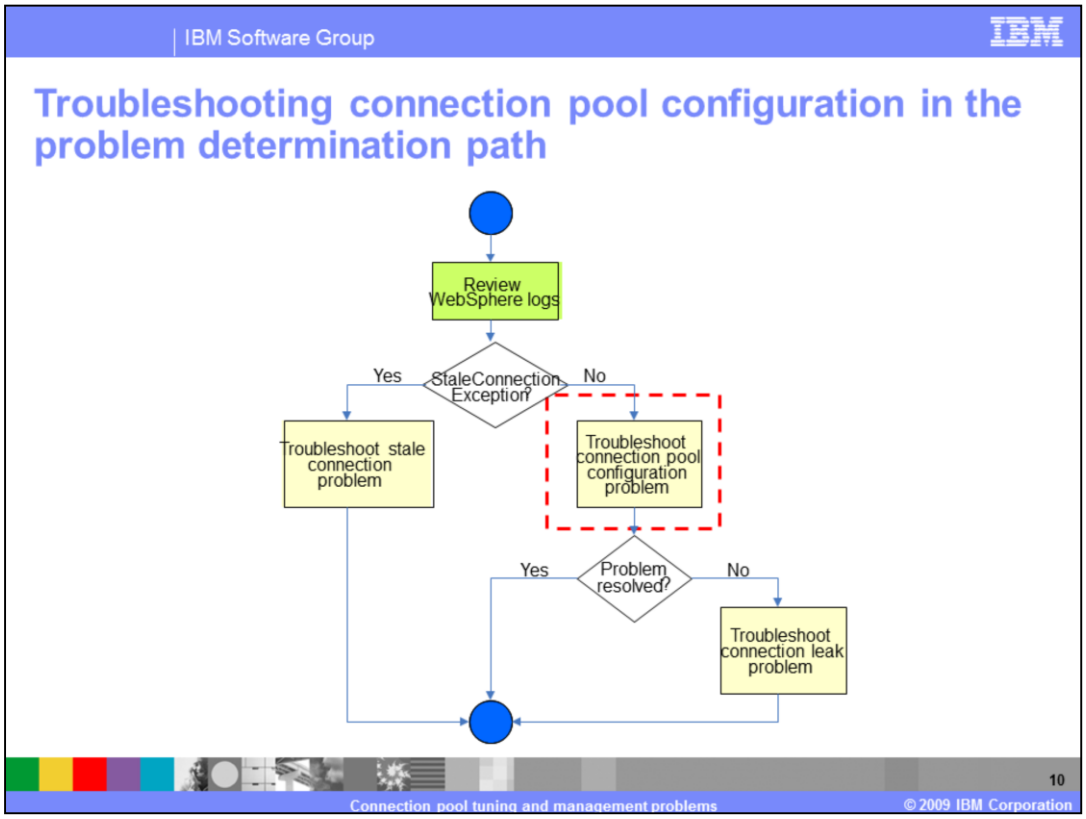
Most people find out they are experiencing a connection pool problem by noticing symptoms in their application‘s behavior instead of noticing events in the SystemOut and SystemErr logs. There is typically a problem with a connection pool when an application experiences sporadic failures when trying to connect to a data source. This means the application was able to connect to the data source and work normally but then started to see intermittent failures or a increase in user response time. In either case, the next step is to check the log files to help narrow down the possible cause for the sporadic behavior. There are several possible outcomes from checking the log files. One outcome is that you do not find connection exceptions. In this case, the problem is likely due to something other than the connection pool. Another outcome is you find ConnectionWaitTimeoutException message in the log files then there are two probable causes. You either need to change the connection pool settings or there is a connection leak somewhere in the system.

Finally, if you find StaleConnectionException in the log then, as the exception indicates, there is probably a problem with connections going stale.

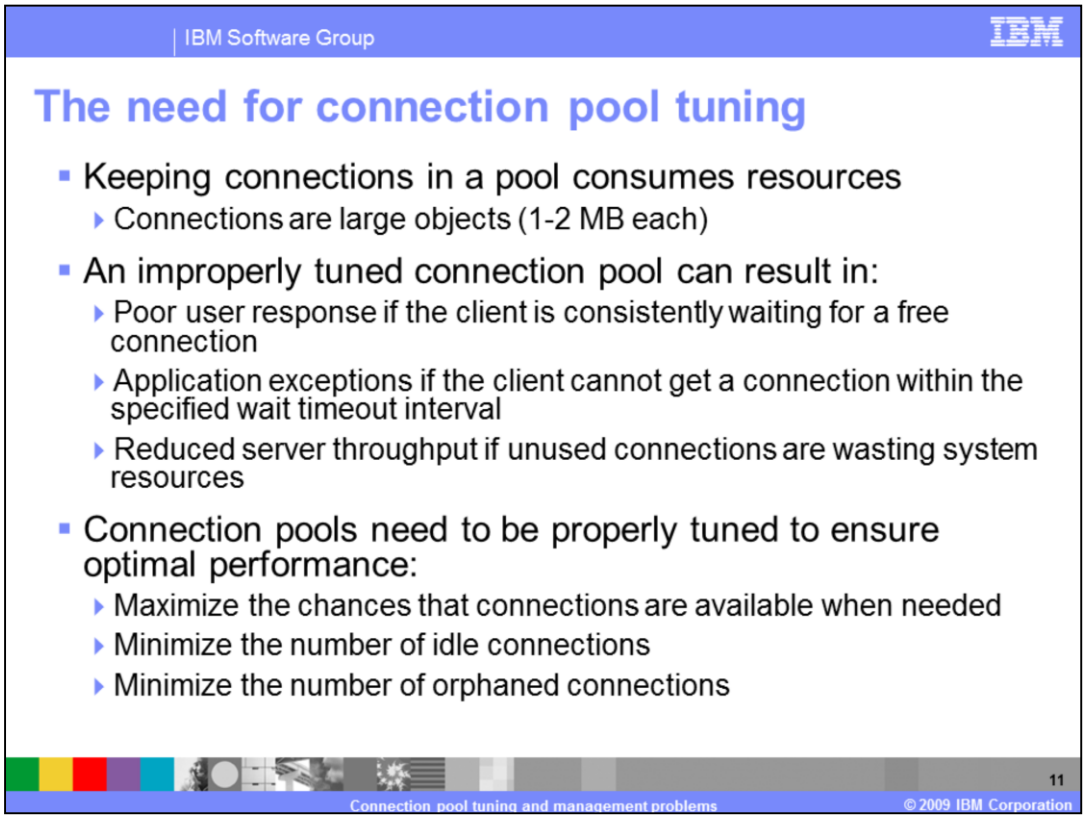


If it helps, you can think of troubleshooting a connection pool problem in terms of a decision tree. The tree starts with the assumption that you are seeing sporadic behavior from your application or you have another reason to believe there is a problem in the connection pool. From there, you review the logs and look for Stale Connection Exceptions. If you find any the next step is to begin troubleshooting a stale connection problem. Otherwise, it is best to review the connection pool configuration and make sure it is not causing the problem. If the configuration checks out then you should begin troubleshooting a possible connection leak in the connection pool.

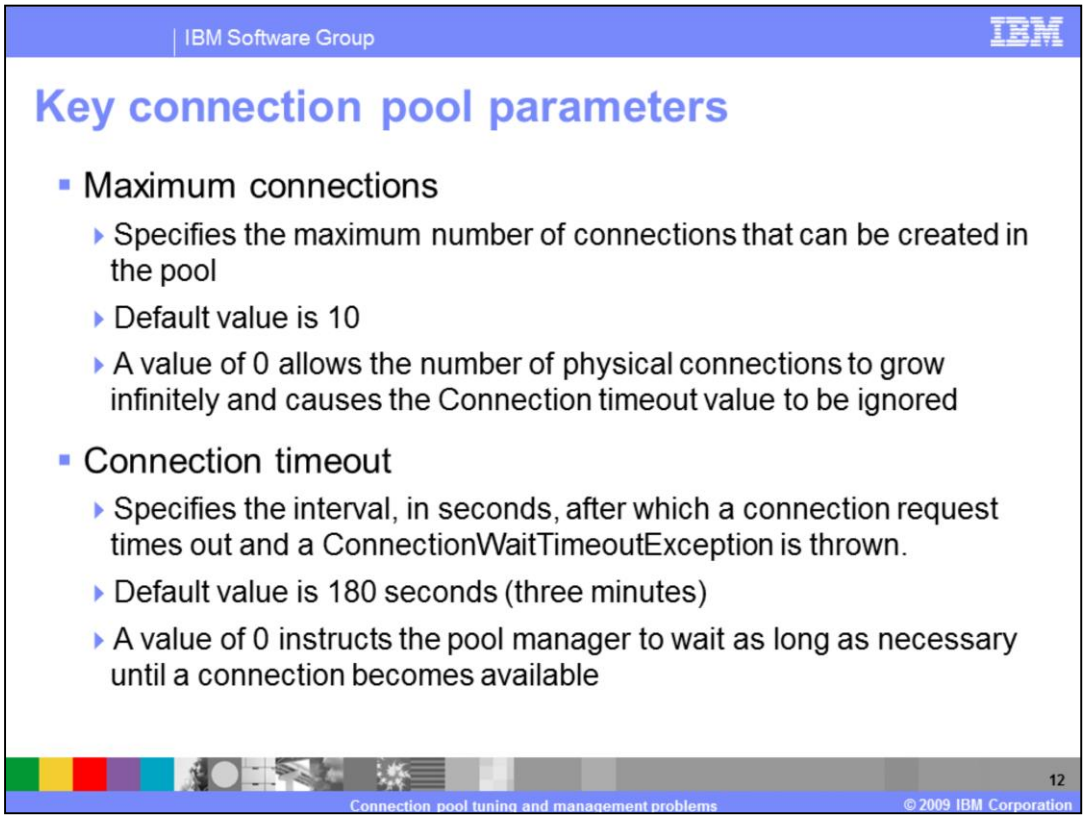
From here, you will take a detailed look at each of the three troubleshooting steps.



In the case where the symptom for a connection pool problem is not accompanied by a StaleConnectionException in the WebSphere logs, start your problem determination effort by looking at the connection pool configuration to rule out any performance tuning issues.

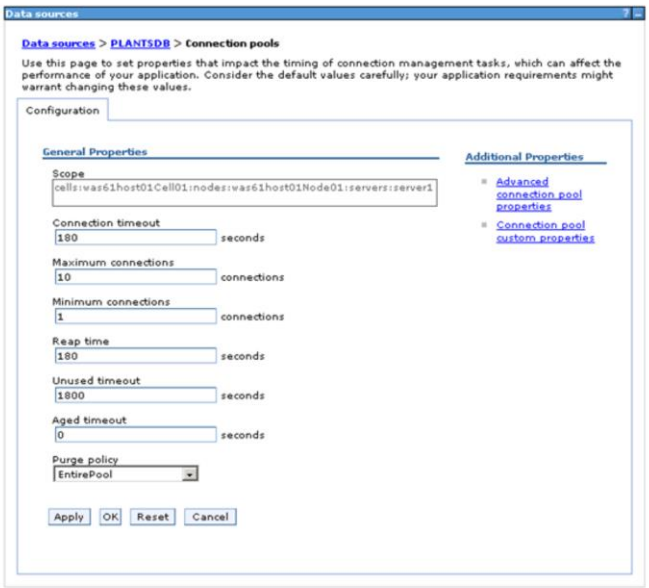


Connection pools allow you to set a range for the number of connections that will maintained by WebSphere Application Server. It is important to get the tuning parameters right otherwise you might cause problems for the application. Setting the pool size too small can slow down the application because it will have constantly wait for free connections. Setting it too large will waste resources and possibly impact the server's throughput. In general, you want to try and tune a connection pool to achieve three goals. First, you want to maximize the chance that connections are available when needed. This means setting the connection pool size so that it is big enough to have free connections when they are needed. Second, you want to minimize the number of idle connections because connections that are not being used are overhead that reduces the server‘s throughput. Finally, you want to set the connection timeout so that it minimizes the number of orphaned connections but does not interfere with connections that are operating normally.

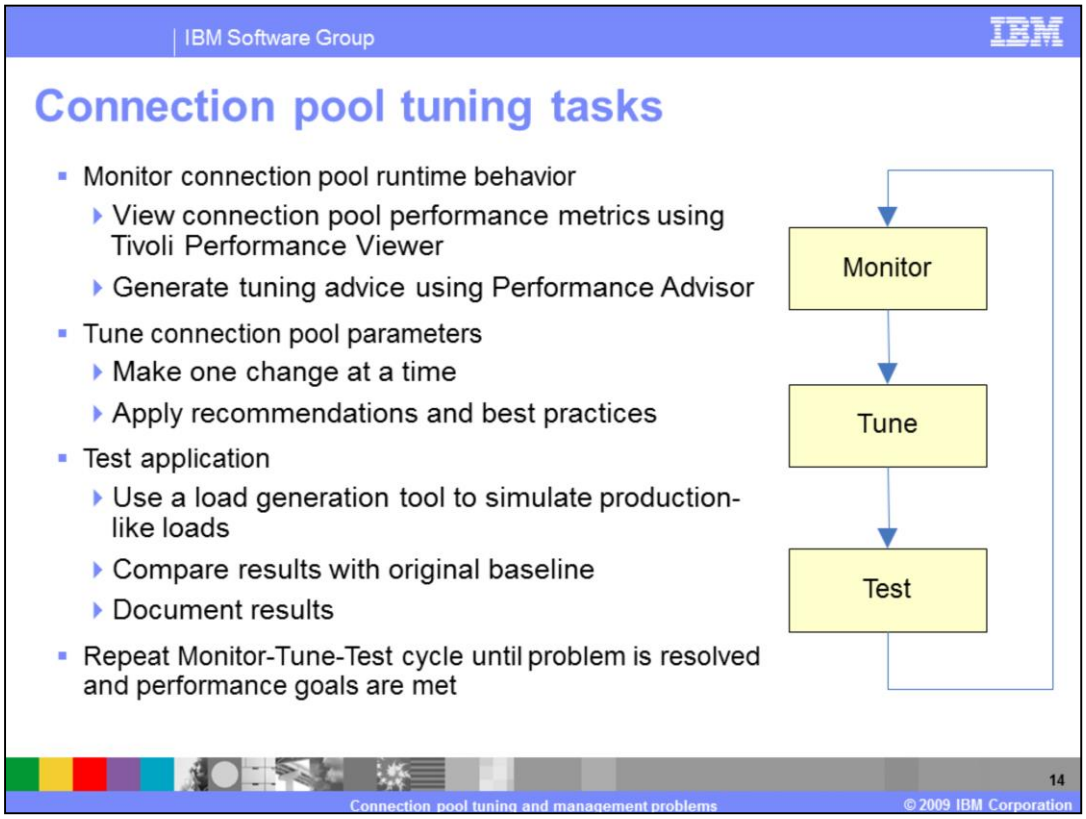


There are a few of the connection pool parameters that play a significant role in achieving the goal of an optimized connection pool. The first of these parameters is the maximum connections count. This value governs the maximum size of the connection pool. If the pool has already reached the maximum size it will not allow a new connection to be created and will instead force a request to wait for an existing connection to free up. However, you can set the maximum value to 0 and allow the pool to grow without constraint. This will also cause the Connection Timeout value to be ignored. The connection timeout is how long a connection request will wait for a free connection before it quits and creates a ConnectionWaitTimeoutException. You can also disable the connection timeout by setting it to 0 and allowing a request to wait as long as it takes to receive a connection. Setting the connection pool to 0 in and allowing unconstrained growth can be a bad idea. Resources should be load tested in order to find the best value and should always be bounded. Failure to do so can lead to memory exhaustion and worse, overloading the resources that are being connected to.

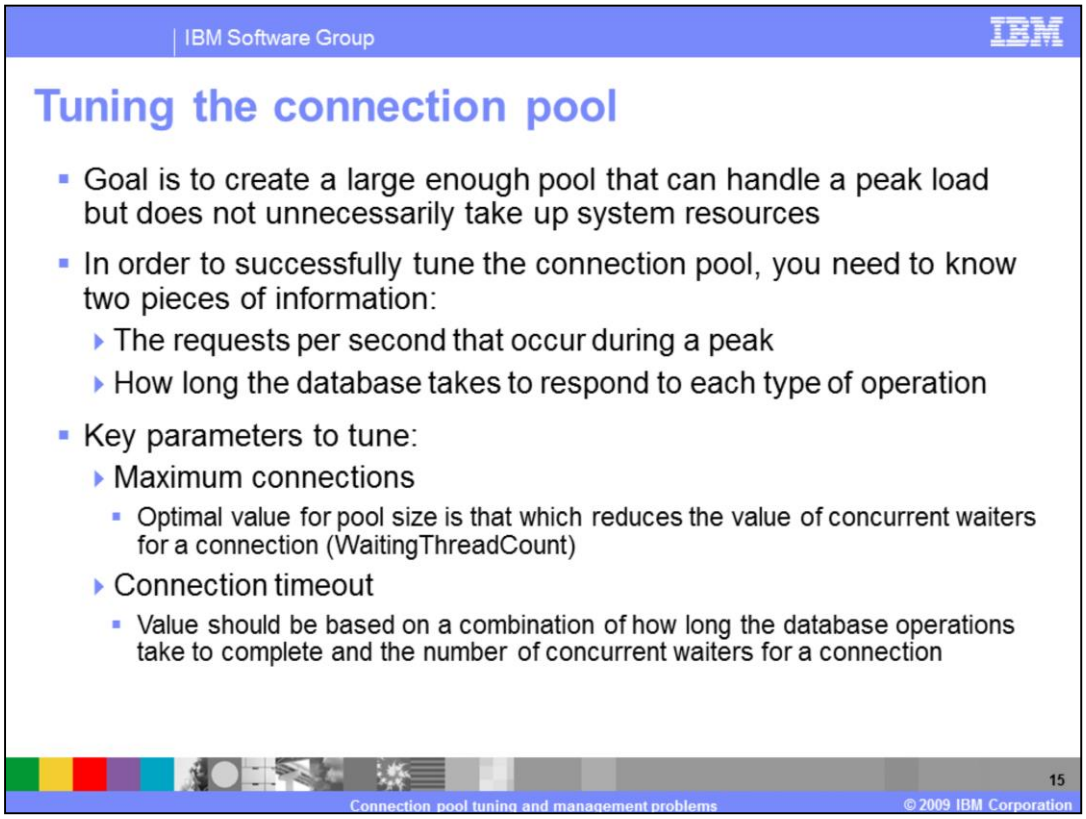
Connection pool parameters in the administrative console



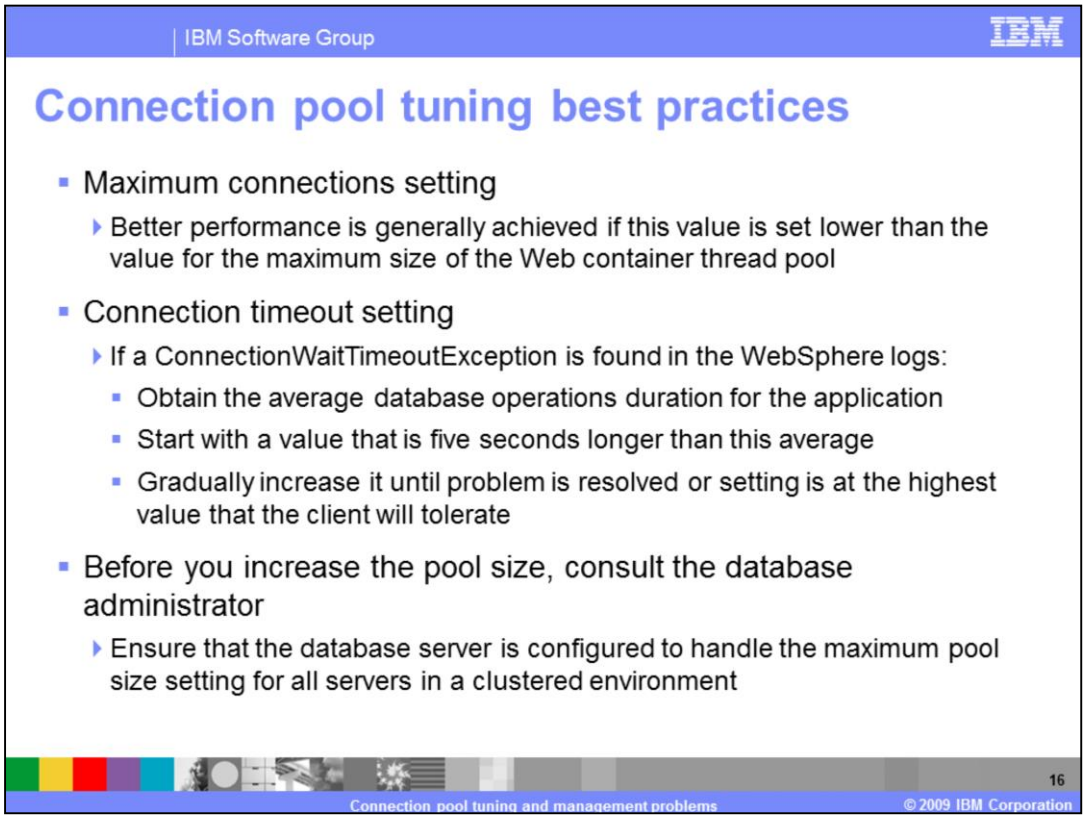
There are several other connection pool properties that can be configured in the WebSphere administrative console. Minimum Connections, for example, specifies the number of physical connections that should be maintained. Note, this does not mean the connection pool will start with the minimum number of connections but that it will not go beneath that value once it reaches it. Many of the connection properties interact with each other. For example, the Reap time specifies, in seconds, the interval between runs of the connection pool's maintenance thread. This value will affect the accuracy of both the Unused timeout and the Aged timeout.



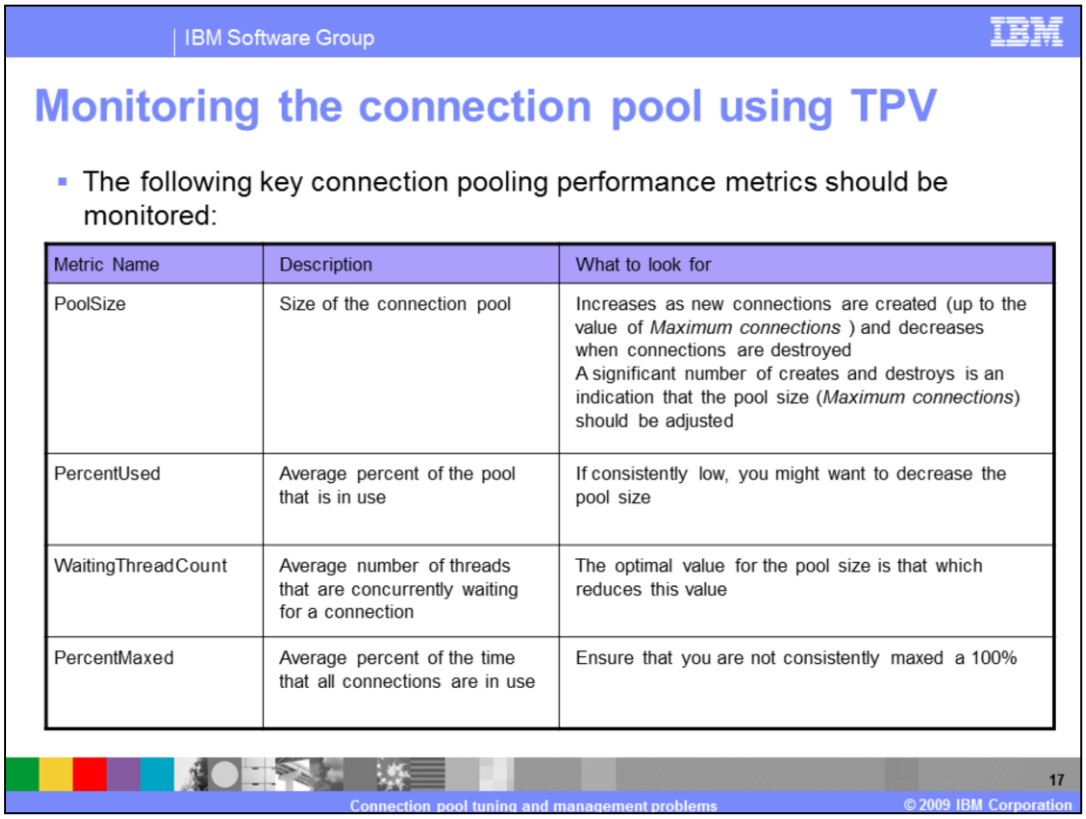
Performance tuning, in general, is an iterative and incremental process consisting of multiple Monitor-Tune-Test cycles. Having the right tools, including a load generation tool to simulate real-world users for load testing, is critical to ensure successful results. The art of performance tuning is a mixture of documentation, test data, and experience. There are some tools that can assist with this practice such as the Tivoli Performance Advisor provided in WebSphere Application Server V6 via the administrative console, but the suggestions that it offers still need to be verified through load testing. The general method for getting the correct value is to increase the timeout and connection parameters until the timeout issue disappears and then backing them off until any wasted resources are recovered. Note that the Performance Monitoring Infrastructure (PMI) is enabled by default in WebSphere V6 and can used to measure performance metrics during the testing.



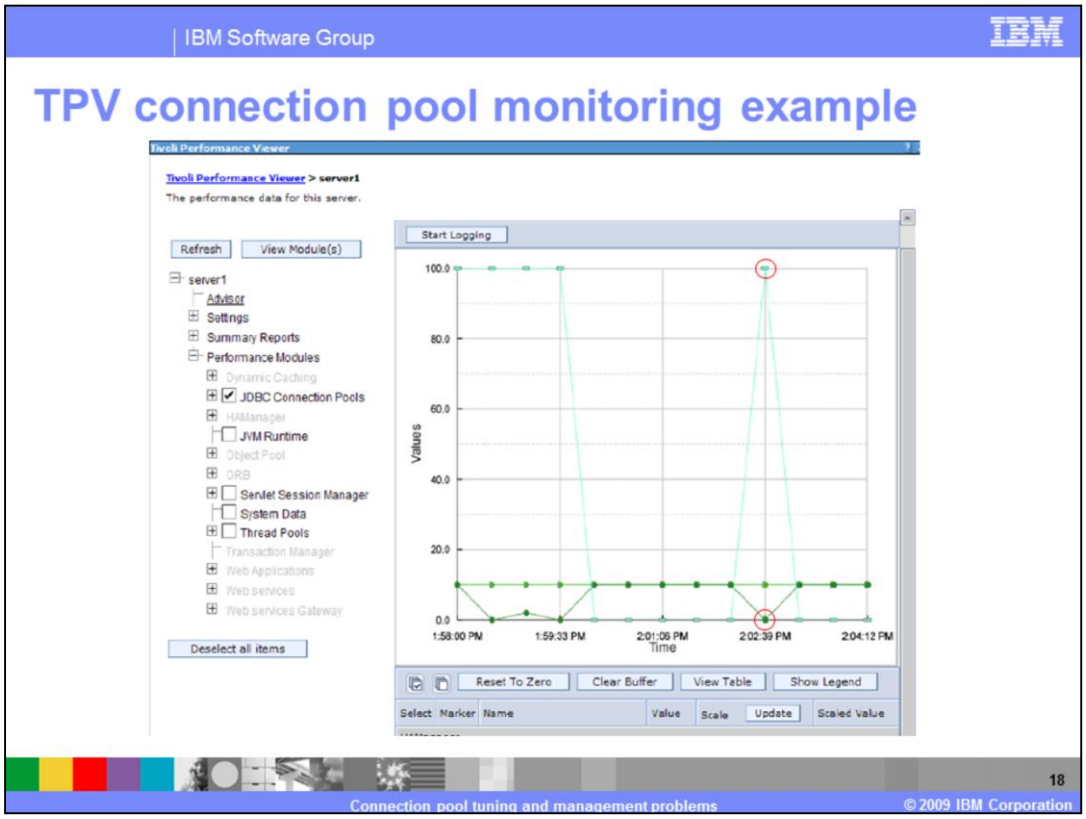
When tuning, the correct parameter values can only be discovered through trial and error. In particular, the two parameters that will have the greatest effect on correcting connection pool configuration errors are the connection timeout and maximum connections. If the time taken to complete a database operation is greater than the amount of time a thread is willing to wait for a resource, then increasing the number of available connections will not solve the problem. Conversely, if the connections are short-lived, then increasing their number can lead to the application server being overloaded in other areas during a peak in requests because the extra connections are unnecessarily consuming resources. Also, the number of idle connections during off peak periods should be weighed against the pool ramp up time when a peak occurs. By understanding the nature of these parameters and the nature of the database operations that will occur during a peak load, an optimal configuration can be achieved leading to optimal performance with the lowest possible overhead.



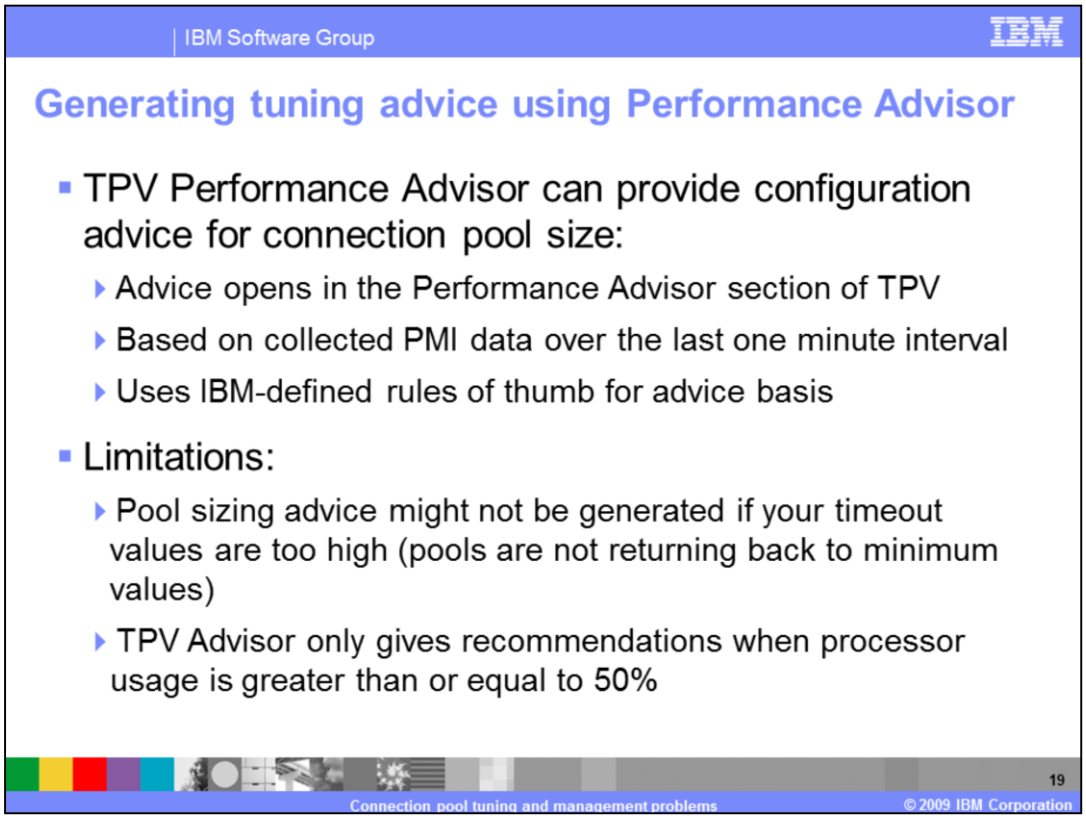
The database connection pool Minimum and Maximum connections values are often misunderstood. If you set a maximum of 40 connections and a minimum of 10 connections, the pool will not start with 10 connections. The value of 10 connections minimum, is actually a low water mark. Until there are 10 connections required concurrently, the pool will only contain the maximum amount of concurrent connections required up to that point. Therefore, if the number of concurrent connections has only ever reached six, then the pool will contain six connections. Once the number of connections needed exceeds 10, the number of connections in the pool will not drop below 10 until the pool is cleaned out. For example, after the reap time expires, all unused connections are destroyed until the Minimum connections threshold is reached. Configuring a data sources should be done in consultation with the database administrator. For instance, the connection pool size should not be larger than the number of agents or connections allowed on the database server. This can become an issue, especially in a cluster, because each application server will allocate its own pool. To compute the maximum connections the database must support, multiply the connection pool size by the size of the cluster.



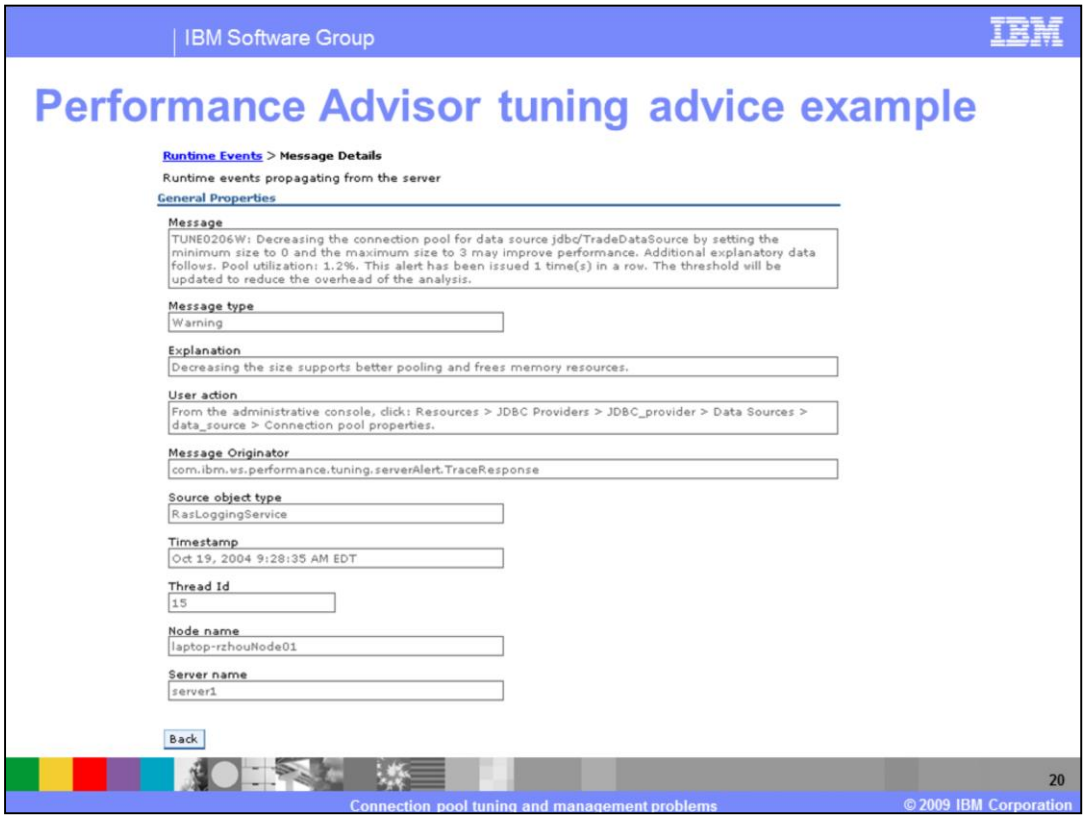
Here are some of the key metrics that WebSphere Application Server monitors in the PMI. These metrics are displayed in TPV under the JDBC Connection Pools and JCA Connection Pools modules. Monitoring the PoolSize metric will allow you to examine how the application uses connections from the pool and note trends and behaviors. The WaitingThreadCount metric is a good way to determine if your pool is sized too small. This metric tracks the number of threads that are having to wait for a connection to become available before completing their requested operations. Lastly, the PercentMaxed value is a key point for determining if your pool size is too small. It is ok to hover in the high percentages for short periods of time however, a pool that is 100% utilized for long periods of time is bound to have waiting threads and long response times that will adversely affect the application performance.



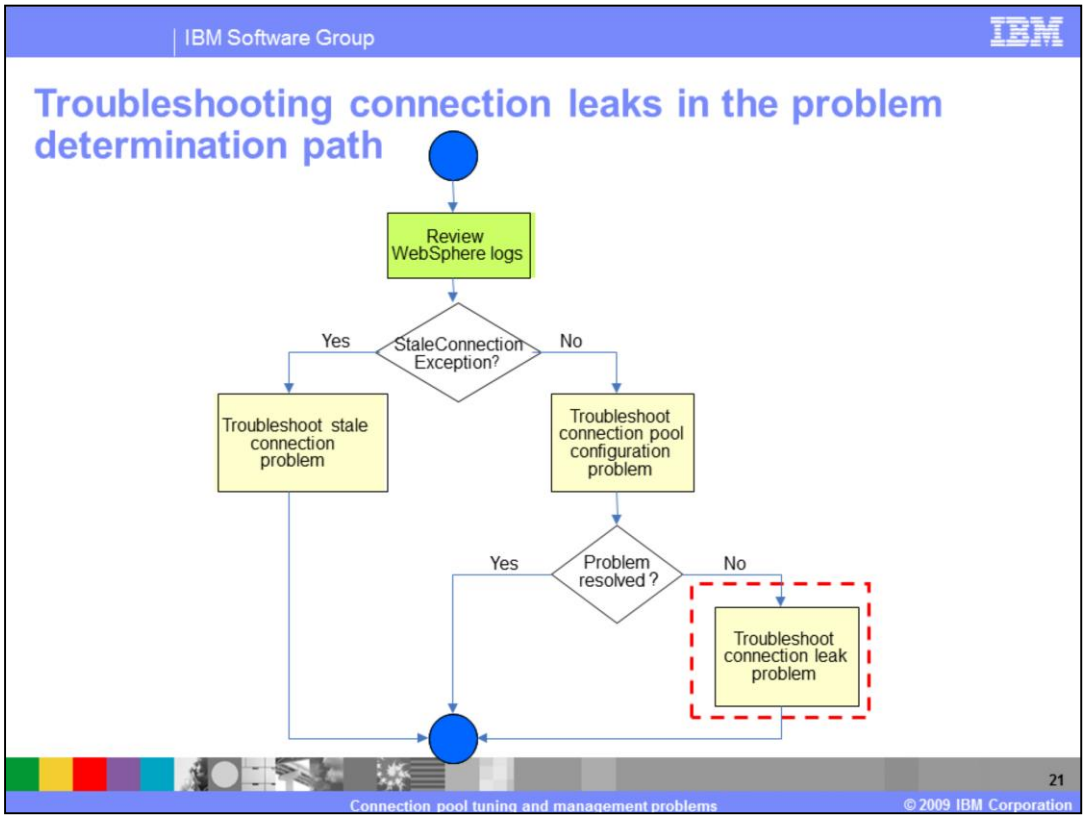
This is a screen capture of the Tivoli Performance Viewer displaying information on the JDBC Connection Pools. The cyan colored graph plots the PercentUsed metric. The dark green graph plots the FreePoolSize metric. And the green graph plots the CreateCount metric. In this example, ten connections have been created, reaching the default maximum connections value for the pool. Notice that, as expected, when the FreePoolSize is zero indicating no connection available in the pool, the PercentUsed value is at 100%.



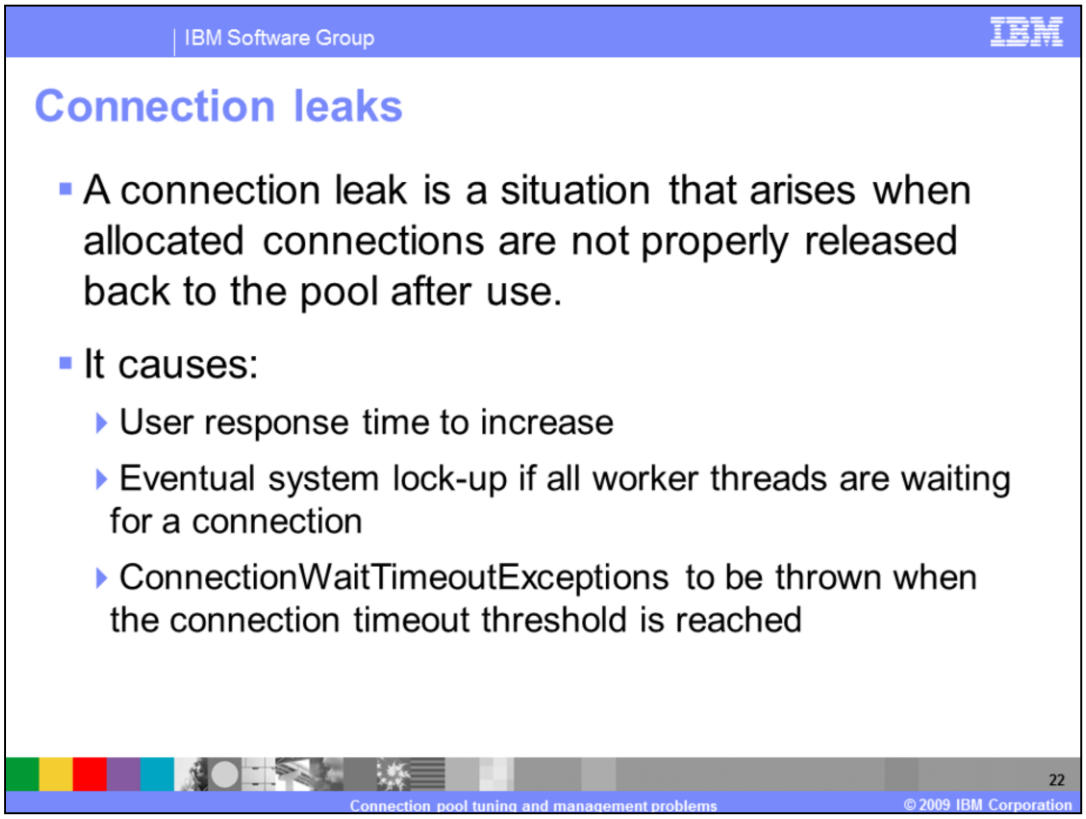
TPV Performance Advisor is one of the ways that WebSphere Application Server can provide tuning advice. TPV Performance Advisor runs on demand and outputs recommendations to a graphical interface in the administrative console. It‘s recommendations are based on situations it observes. For example, if it observes that the number of connections is continuously low then it will recommend that you lower the size of the connection pool. The TPV Performance Advisor can be accessed and configured in the administrative console.



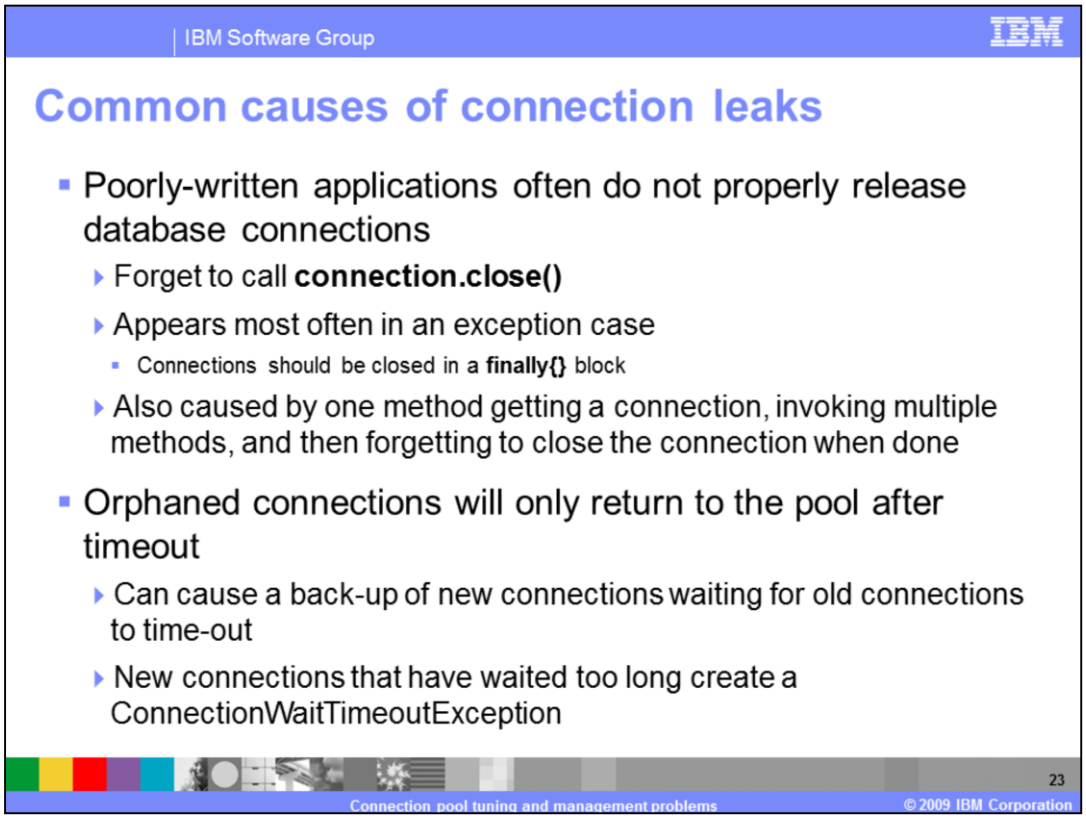
This slide shows an example of connection pool tuning advice generated by the Runtime Performance Advisor. From the screen capture we can see a tuning recommendation stating that the minimum size of the TradeDataSource pool should be reduced to zero and the maximum size of the pool should be three based on the data collected by the advisor component. Note that the threshold for the message adjusts automatically to prevent the system from issuing multiple, redundant messages for the same values.



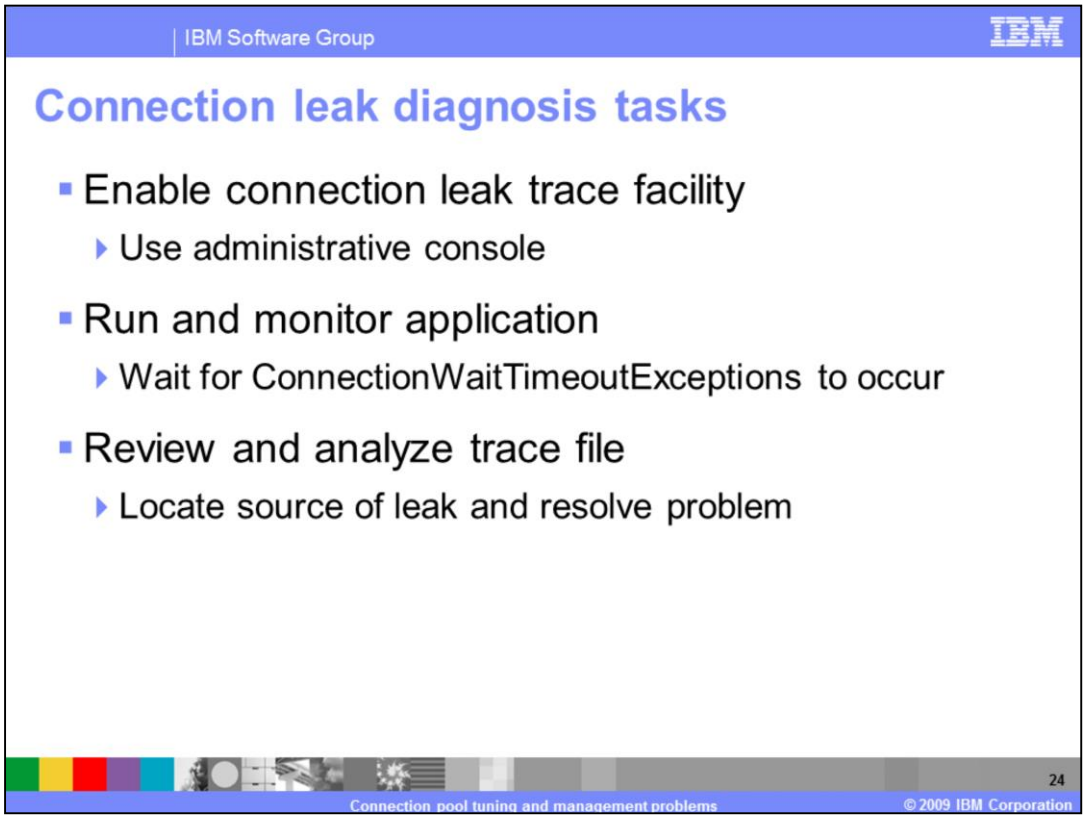
After you have ruled out the possibility of a stale connection and connection pool tuning problem, consider the possibility of a connection leak.



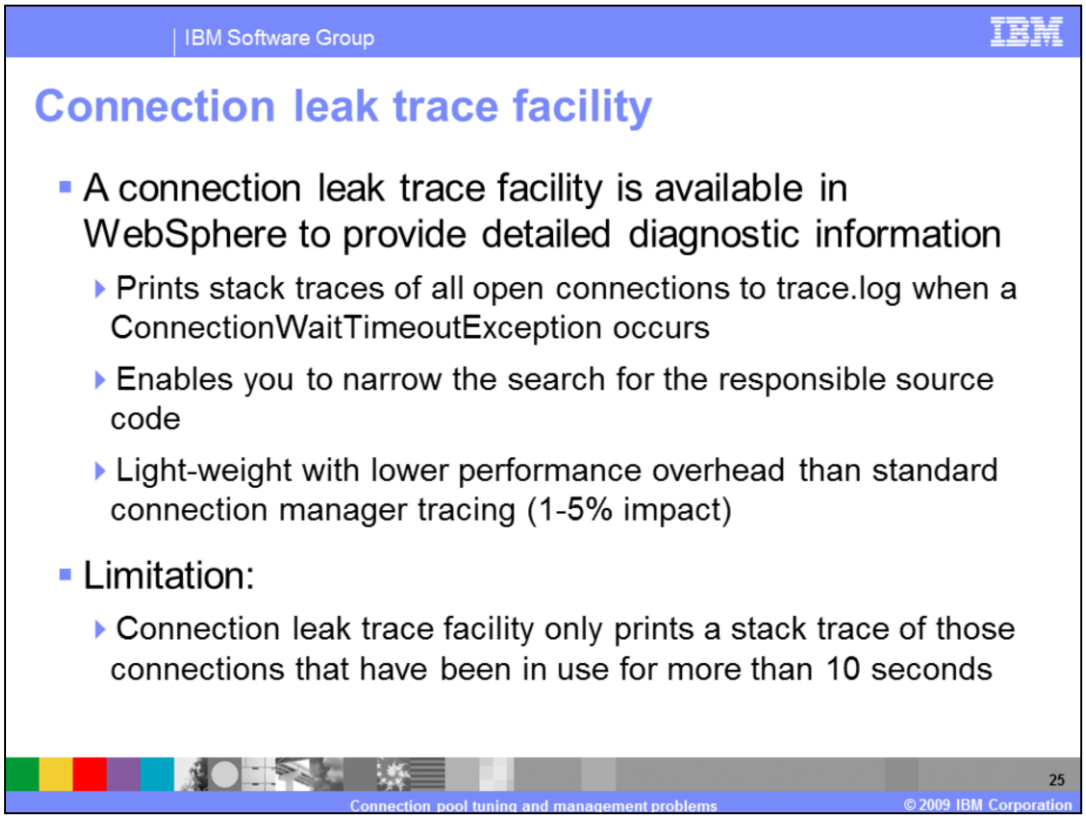
A connection leak is typically identified by a ConnectionWaitTimeoutException in the WebSphere logs. WebSphere Application Server is smart enough to eventually time-out orphaned connections and return them to the pool, but for an application that makes frequent use of database connections, this might not be enough corrective action to maintain throughput levels. New connections can get queued up waiting for the database while old connections are waiting to be timed out. This can bring the application grinding to a halt.



The most common reason for a connection leak is when an application does not defensively manage the connections it requests from WebSphere Application Server. This often happens because the application does not properly close connections. The connections should be called in the finally code block of a try/catch/finally construct to ensure that connections are closed properly. Unfortunately, connection leaks have traditionally been hard to diagnose because the error messages do not typically provide specific enough information about the source of the problem. A source code review is typically needed to find where the connections are not being properly closed.



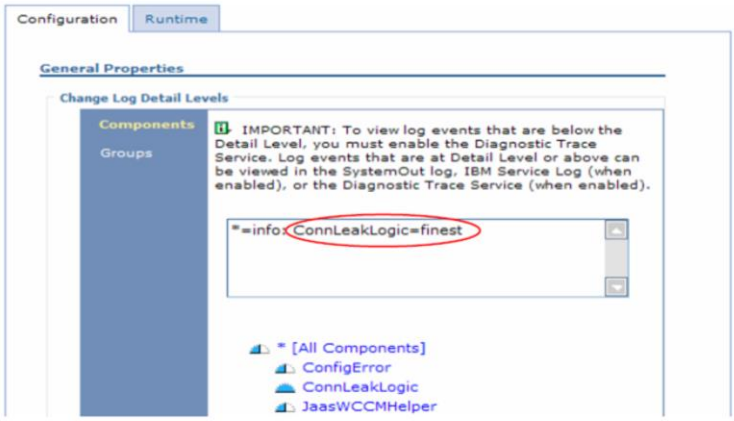
To assist in locating the cause of a connection leak, there are tools built into WebSphere Application Server that can help narrow down the search. The most useful of which is the connection leak trace facility. Connection leak tracing will allow you to gather more detailed information about the leak. The trace utility can help you determine if connections are not being closed or if the application should be redesigned to use fewer connections.



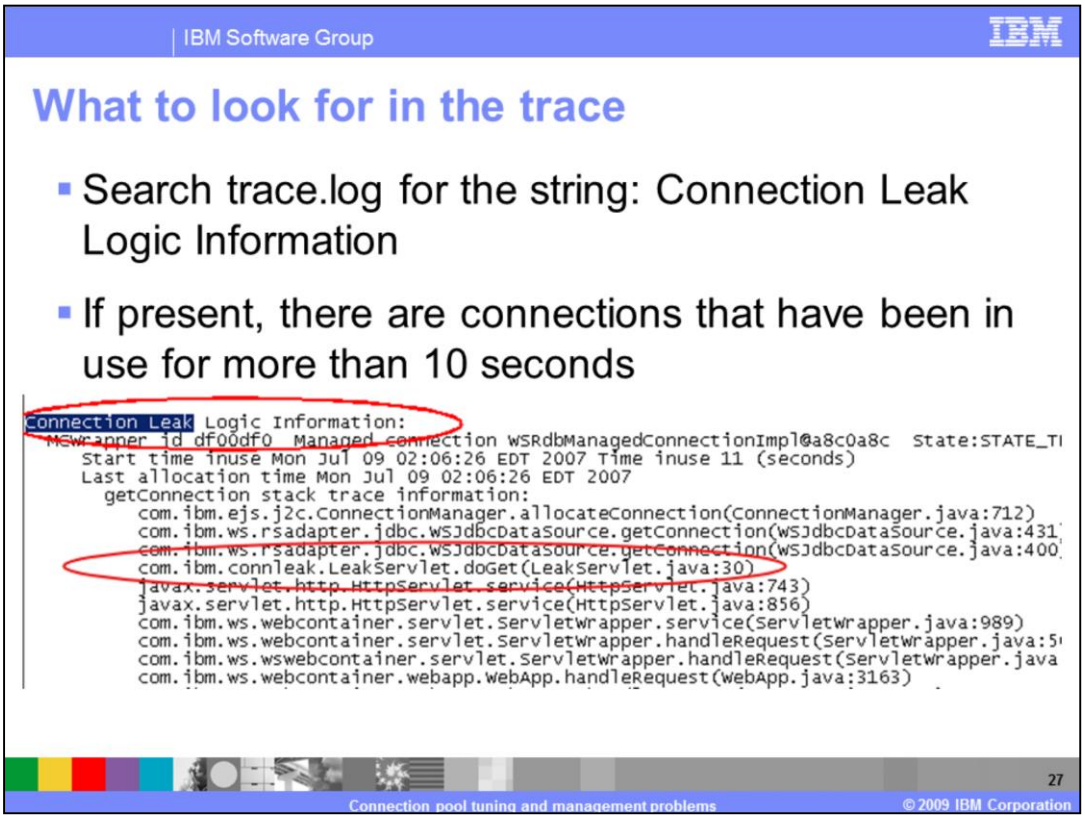
When a thread times out waiting on connection from a full connection pool, it will create a ConnectionWaitTimeoutException. When this exception is thrown, the connection leak tracer will print out the stack traces for every open connection. It does so only when a problem has occurred, providing instant recognition of when it occurred and incurs reduced overhead when compared to the full WebSphere tracing mechanism. This feature is useful because it shows you the call stacks for all open connections at the time of the exception. This enables you to significantly narrow your search area when you look at the application's source code to try and find the responsible code. It is also helpful to IBM support, because it will help distinguish between application problems and WebSphere defects. When you enable the connection leak trace facility, for every time interval, the WebSphere connection pool manager checks how long a connection has been in use and prints the stack trace to the trace log file. Currently the default time interval is unchangeable and defaults to ten seconds. If you have a need to change the default value, contact IBM technical support to obtain an iFix that allows you to add a custom property to the data source configuration.

Enabling the connection leak trace facility

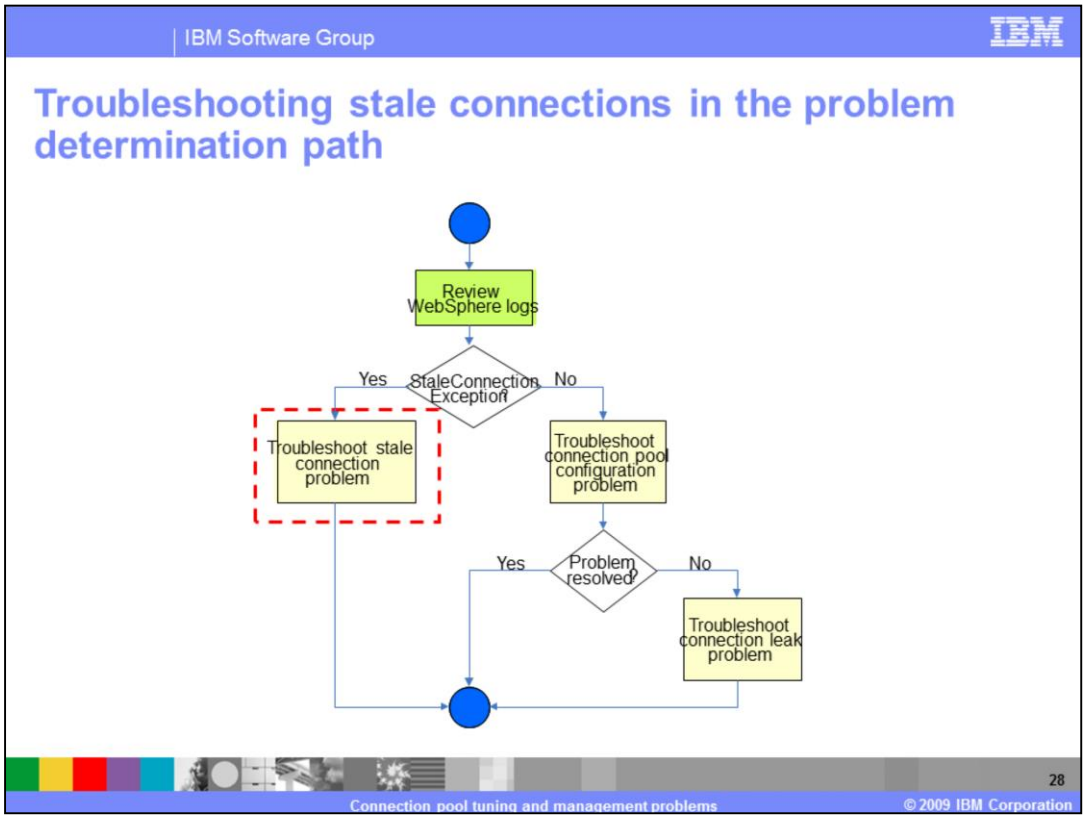
- Enabled using the ConnLeakLogic trace group and the finest level



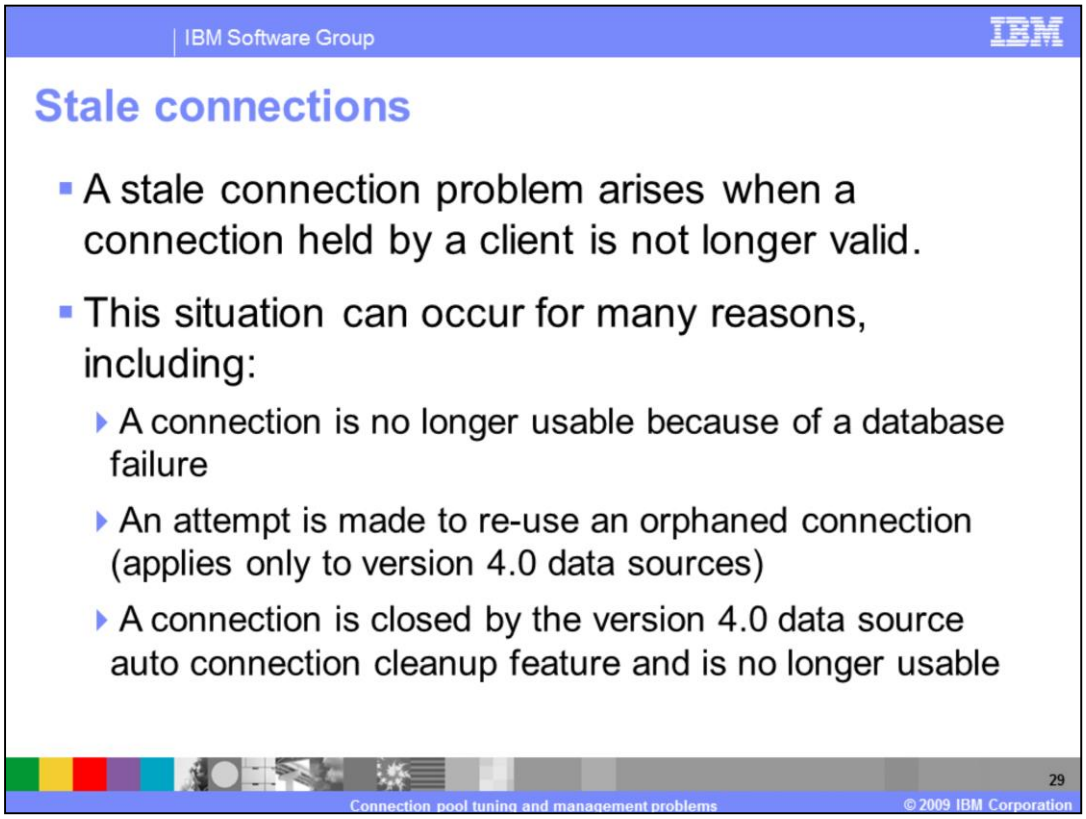
The connection leak trace facility is enabled through the administrative console. To enable the facility, start by navigating to Troubleshooting, then Logs and Trace. Select the application server you wish to trace on and then select Diagnostic Trace. Make sure logging is enabled and then click the Change log detail levels link. You can then specify the required trace level under the ConnLeakLogic category.



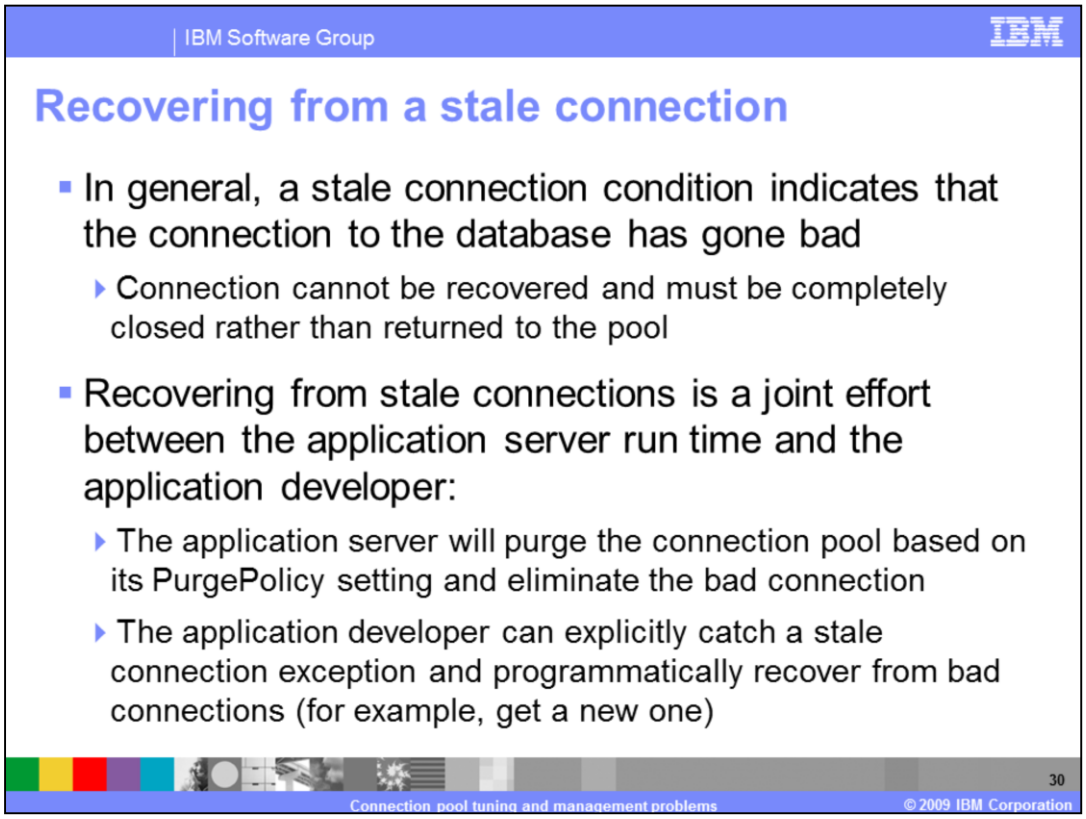
There are a few key lines to look for when you start evaluating the trace files. You will first want to look for a line that contains the string Connection Leak Logic Information, followed by a colon. This indicates the start of the connection leak logic output. From there, you should check the time in use and the stack trace for each of the connections. In this example trace, the doGet() method of SnoopServlet has been using a connection for 20 seconds and is therefore a good suspect for a source of a leaking connection.



If you find StaleConnectionExceptions in the WebSphere Application Server logs then your choices are clear; start looking for stale connection problems

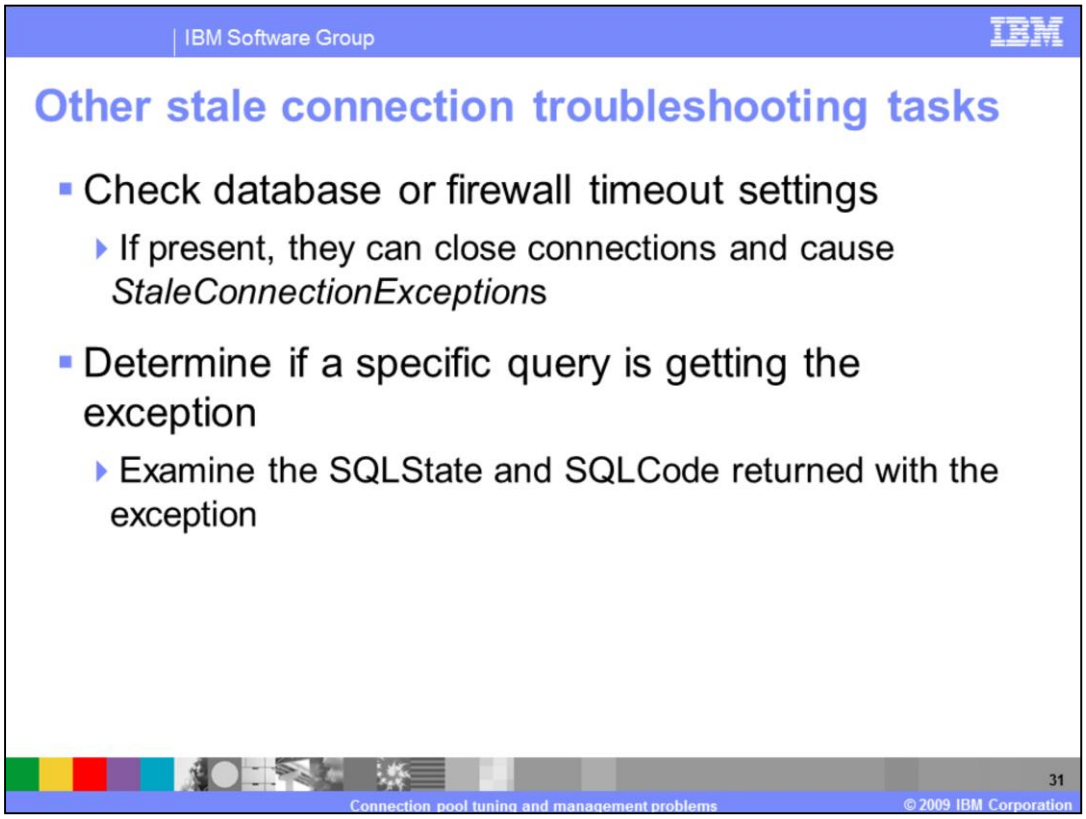


A stale connection is essentially a connection that is held by a client but is no longer a valid connection. One way this can happen is if the other end of the connection, a database for example, experiences a failure and is no longer available. Stale connections can also occur in Version 4.0 data sources when the connection is closed by the connection cleanup feature but the client is still trying to use it. This will happen if the connection has not been in use at least twice the unused timeout value. At this point, the connection is orphaned and the client will error if it tries to use the connection again.

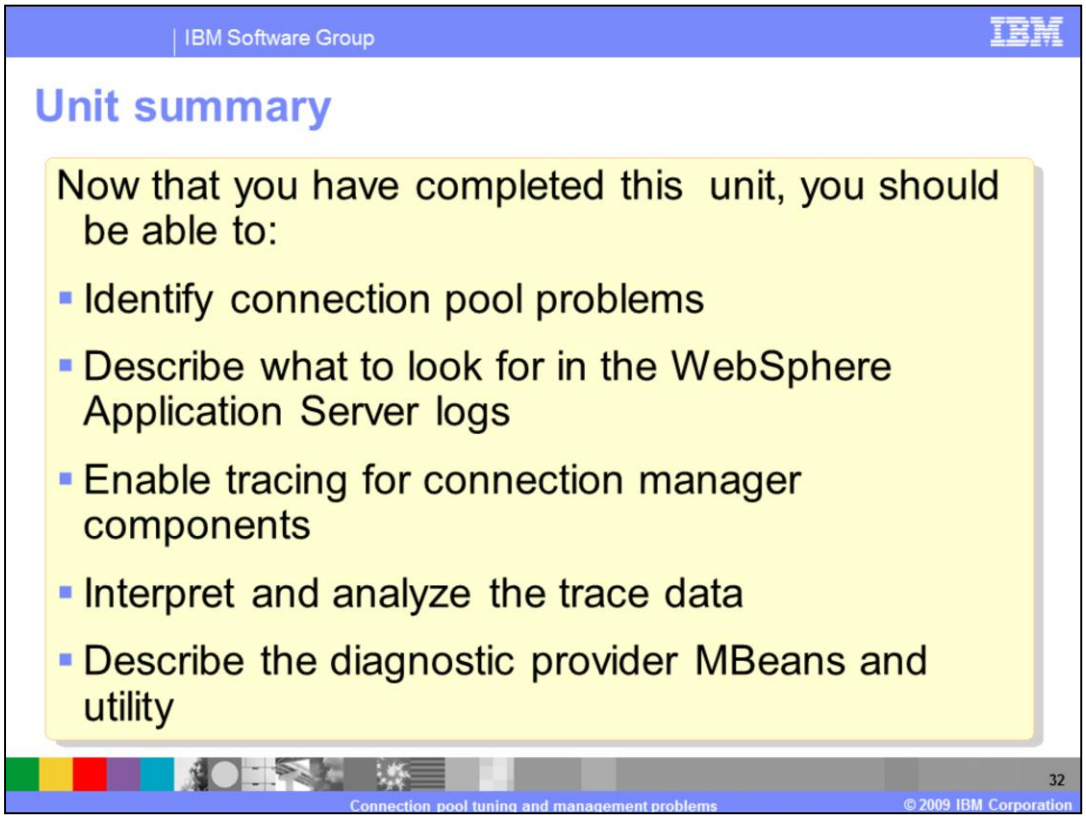


An individual connection can not be recovered once it creates a StaleConnectionException. Instead, the best way to recover from this type of exception is by explicitly catching it. Catching a StaleConnectionException while running within the context of a transaction will allow you the avoid having to repeat the entire transaction. One option is to try and complete the pending transaction with a new connection. It is important to note that the application server will also take actions to recover from a StaleConnectionException depending on the PurgePolicy setting. It can either clear the entire connection pool, assuming that if one connection went bad then all

other connections will likely have the same problem, or just clear the stale connection.



There are several other reasons why a connection might become stale, many of which exist beyond the control of WebSphere Application Server. One common reason is a discrepancy between the firewall timeout settings and the connection timeout settings. It is generally a good practice to make sure the connection pool aged timeout is less than the firewall‘s timeout and that both are less than the database timeout. It is also possible that you are experiencing a StaleConnectionException because the returned SQLCode maps to a StaleConnection. If you aren‘t able to find the source of the problem by taking a quick look at the various components involved in the connection then your best bet is to turn on tracing and gather more information. This is can be very useful when a connection is unusable because of a SQLException that did not immediately map to a StaleConnectionException but eventually resulted in one being thrown.



Now that you have completed this unit, you should be able to identify connection pool problems, describe what to look for in the WebSphere Application Server logs, enable tracing for connection manager components, interpret and analyze the trace data, and describe the diagnostic provider MBeans and utility.

Trademarks, copyrights, and disclaimers