CHAPTER 2

DB2 for z/OS Overview

This chapter presents the tasks, services, structures, architecture, and components of DB2 11 for z/OS, which is required knowledge for a DB2 system administrator.

CSECTs in DB2

This short overview shows how the DB2 code is structured internally. A subcomponent is a group of related CSECTs.

- In DB2, each object module contains a single Control Section (CSECT).
- Typically, a CSECT will perform a single function, and the object module and CSECT have the same name.
- CSECT names and message identifiers begin with *DSN* in DB2.
- You can find the readable data set associated with a CSECT for DB2 in member DSNWMODS in library SDSNSAMP.
- A subcomponent is identified by the fourth character of a DB2 CSECT name.
- There are three groups of subcomponents:
  - » Distributed Data Facility (DDF) Services
  - » Database Services (DBAS)
  - » System Service Address Space (SSAS)

Distributed Data Facility Services

Running as an additional address space in DB2, DDF consists of one subcomponent called the *distributed data facility*. DDF controls connecting distributed applications to DB2 for z/OS. The naming convention for the subsystem is xxxxDIST.

Following are the resource managers associated with DDF. These subcomponents execute in the DDF services address space:

- Data communications resource manager (DCRM)
- Distributed transaction manager (DTM)
- Distributed relational data system manager (DRDS)
- Distributed data interchange services (DDIS)

DB2 Distributed Relational Data Architecture (DRDA) subsystems and other relational databases can communicate with DDF by using TCP/IP or VTAM on the same network. DDF supports two network protocols—SNA and TCP/IP—and the DRDA database communication protocol.

DRDA is set of protocols for databases that describe the architecture, allowing connection and access to distributed relational data in multiple database systems. DRDA defines what must be exchanged and how it must be exchanged, and then it coordinates the communication between systems. The three components in DRDA are the application requestor, the application server, and the database server.

Database Services

The DBAS uses SSAS and z/OS to handle the actual database structures. The goal of the DBAS is to manage the physical structures and data, execute SQL, and control the buffers. Within the DBAS, you have three main components: the relational data system (RDS), the data manager (DM), and the buffer manager (BM). Even though each is an independent component, they work together to make a proper subsystem of z/OS. The DBAS address space is also referred to as the Advanced Database Management Facility (ADMF) address space.

Subcomponents of interest that execute in the database services address space are:

- Data manager
- LOB manager (LOBM)
- Service controller
- Stored procedures manager

- Data space manager
- Relational data system (RDS)
- Utilities (work with associated code in an allied address space)
- Buffer manager

IRLM

DB2 requires the address space subsystem services of the internal resource lock manager (IRLM), which resides in its own address space. This is a separate IRLM from the IMS IRLM. IRLM works with DB2 to serialize access to your data. DB2 requests locks from IRLM to ensure data integrity when applications, utilities, and commands all attempt to access the same data.

You must still specify PC and MAXCSA DSNZPARM parameters, but their values are no longer used. We keep the parameters for compatibility reasons in DB2 11. Specific system-site specifications determine the amount of available storage for IRLM private control blocks, including locks. All IRLM locks are in the IRLM private address space; locks are not placed in the extended common service area (ECSA) any longer.

IRLM control block structures are estimated at 540 bytes per lock and are above the 2 GB bar.

System Services Address Space

SSAS manages logs, agent services, and more by executing subcomponents in the system services address space. This address space is also called the *data system control facility* (DSCF) space. A few of the subcomponents that execute in the SSAS are as follows:

- System parameter manager
- Recovery manager
- Recovery log manager
- Group manager
- Distributed transaction manager
- Storage manager
- Agent services manager
- Message generator
- Initialization procedures
- Instrumentation facilities

- General command processor
- Subsystem support

Allied Address Spaces

DB2 communicates with other address spaces in your z/OS environment, and we refer to these as *allied address spaces*. DB2 communicates with the following "allied agents" to facilitate requests:

- TSO attachment facility
- Standalone utilities
- Subsystem support
- Message generator (this is standalone only) DSN1SDMP
- IMS attachment facility
- Call attachment facility
- CICS attachment facility
- Resource Recovery Services attachment facility (RRSAF)
- Utilities

DB2 controls connections or threads to these allied agents through a DSNZPARM value of CTHREAD (defaults to 200, maximum of 20,000, updatable online). This value defines the number of concurrently allocated threads for our local connections. If we find that we are waiting for a connection to access the DB2 subsystem, we may need to increase the number of allied connections through CTHREAD. This, along with MAXDBAT, protects the virtual storage allocation. Remember to not overcommit your virtual storage resources. You might need to increase MAXDBAT if the number of remote threads is queued with work waiting. The sum of the values of CTHREAD and MAXDBAT is limited to a maximum of 20,000.

Because some utilities use parallelism, we will have one thread for each utility and an additional thread for each subtask. So a single utility may be using many threads. You must specify a value for CTHREAD that will accommodate utility parallelism.

Non-Allied Address Spaces

Remember that DB2 does not communicate with the DB2 precompiler (PRE). However, the precompiler may require an allied address space, depending on the precompiler

options you have selected. The full message generator for DB2 resides in the SSAS. You can also run this standalone in allied or non-allied address spaces.

DB2 Subcomponents

The software that comprises the resource managers is usually responsible for managing a specific resource. The resource being managed can be physical or logical. There is usually one subcomponent for one resource manager. However, some exceptions apply; for example, the precompiler is not a resource manager, and the instrumentation facilities subcomponent contains two resource managers.

A resource manager identifier (RMID) identifies a resource manager. This number identifies the source of diagnostic output in your dumps. Appendix A has a list of subcomponents and identifiers for your reference.

Work Requests in DB2

Our DB2 tasks and agents are subcomponents that run in an allied address space. A DB2 work request is represented by an agent. Classes of agents include system agents, allied agents, and database access agents. DB2 tracks the agent (the work) through an agent control element (ACE). Each ACE is associated with one or more execution blocks (EBs).

A z/OS execution unit (TCB or SRB) and EB have a one-to-one relationship. An EB describes each unique unit of dispatch work that can be either in task control block (TCB) or in service request block (SRB) mode. The user's home address TCB is pointed to by all allied agents' primary EB with which it is related.

An *execution unit* describes each unique unit of dispatch. This unit of dispatch can be in either a TCB or SRB mode in z/OS.

In the DB2 address space, execution units that are created in TCB mode are called *service tasks*. Resource managers in DB2 can dynamically delete and create service tasks. When DB2 is initialized, service tasks are created and usually exist until DB2 stops. These service tasks remain idle until their services are needed in DB2.

Examples of permanent service tasks are as follows:

- System service tasks
  - » Log manager
  - » Recovery manager
- Database services tasks
  - » Buffer manager

  - » Data manager
- DDF tasks
  - » Distributed transaction manager

DB2 11 System Structure Basics

DB2 uses several types of private address spaces, and each type requires storage.

- DB2 DDF address space (DSN1DIST)
- IRLM address space (IRLMPROC)
- DB2 SSAS (DSN1MSTR)
- DB2 database services address space (DSN1DBM1)
- DB2 allied agent address spaces
- DB2 stored procedures address spaces (WLM)
- DB2 administrative scheduler address space

When we start our DB2 subsystems, there is a recommended dispatching priority for these address spaces in z/OS. IRLM is started first; without locking to protect our resources, we cannot begin. Then, we start our DB2 performance monitors, followed by DBM1, and then the MSTR address space. Figure 2.1 provides an overview of the subsystem.

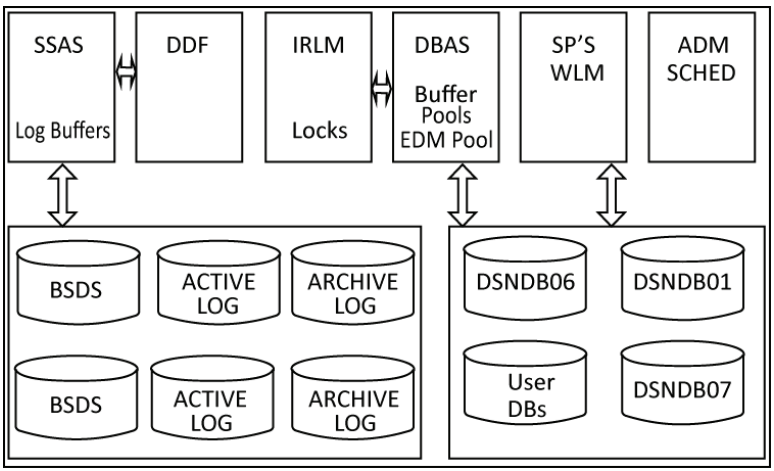

Figure 2.1: Subsystem overview

Attachment

An attachment facility provides the interface between DB2 and another environment, like Time Sharing Option (TSO). In TSO and your batch environments, you can use the TSO, call attachment facility (CAF), and Resource Recovery Services (RRS) attachment facilities to access DB2, as Figure 2.2 shows. Other attachment facilities are DB2 subcomponents that run in the user's address space, which include CICS and IMS.

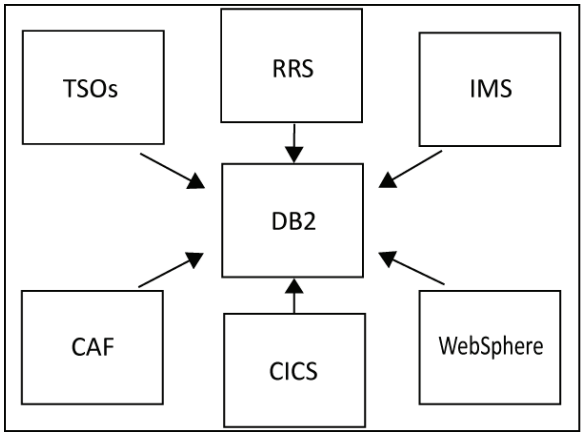

Figure 2.2: Attachment facilities

Call Attachment Facility

For TSO and batch applications needing tight control over their session environment, DB2 does provide the CAF as a connection option. Programs can explicitly control the state of their connections to DB2 by using connection functions that CAF supplies.

First, make available the load module CAF or DSNALI. When the language interface is available, your program can use CAF to connect to DB2 by including SQL statements or instrumentation facility interface (IFI) calls in your program. You can also access CAF by explicitly writing CALL DSNALI statements.

Resource Recovery Services

DB2 supports RRS, which is a newer implementation of attachment capability. This z/OS system feature will coordinate the two-phase commit processing of recoverable resources.

RRS runs in its own address space and can be started and stopped independently of DB2. Starting z/OS RRS allows you to run an RRSAF application. RRS needs to be equal or higher priority than the dispatching priority of DB2.

After you start RRS, you can start or restart an RRSAF connection. The load module known as the RRSAF language interface, or DSNRLI, must be available. Then, your program can use RRSAF to connect to DB2 by including SQL statements or IFI calls in your program, or by using a CALL DSNRLI statement to invoke RRSAF connection functions to establish a connection between DB2 and RRS, and to allocate DB2 resources.

Stored procedures running in a WLM address space require the RRS attachment facility. Resources such as SQL tables, DL/I databases, MQSeries message programs that run in batch or TSO, and recoverable VSAM files within a single transaction scope can use the RRS attachment facility. Keep in mind that the DB2 command DISPLAY THREAD will include the RRS unit of recovery (UR) IDs for DB2 threads.

Threads

When DB2 allocates a thread, that thread's storage is held until deallocation. In the macro DSN6SPRM setting of CONTSTOR, you control each active thread's working storage area by specifying the value of YES. The default for CONTSTOR is NO. If you are experiencing a high private virtual storage usage in DBM1, setting this value to YES may reduce the unused storage. That is because YES will allow DB2 to periodically check the thread storage not used from a committing process, and return that storage to the operating system. You can modify this parameter dynamically with the DB2 SET SYSPARM command.

DB2 will examine the storage pool's thread use and, if the thread has used more than 2 MB or if there are more than 50 commits, will free and release those storage blocks that are no longer in use to the operating system. This action can reduce the amount of virtual storage used in DBM1, especially for long-running threads. However, because of associated CPU overhead by GETMAIN and FREEMAIN requests to the operating system, you need to carefully consider the benefits.

Also keep in mind that the BIND parameter of RELEASE(DEALLOCATE) requires more virtual storage from the increase in size of the package or plan with this parameter. Using RELEASE(DEALLOCATE) will nullify the process of using CONTSTOR YES. No storage contraction will occur. With RELEASE(DEALLOCATE), package structures persist until full thread deallocation.

To help release storage, we have the PKGREL_COMMIT system parameter that we can set to YES or NO. If set to YES, it allows a persistent DB2 thread—at commit or rollback—to implicitly release a package bound with RELEASE(DEALLOCATE) that is active on that thread if a BIND REPLACE, REBIND PACKAGE, online schema change DDL, or online REORG with deferred ALTER operation needs to quiesce or invalidate the package.

The executing holder of the thread, at its next commit, changes to RELEASE(COMMIT) for that thread. This behavior is available only for active persistent threads. If the thread is waiting for resources to complete and is not currently active in DB2, the BIND, DDL, REBIND, or utility statement cannot break in.

Storage

Different subpools are allocated for storage. Storage pool 229 (SP229) is storage acquired by a GETMAIN or released by a FREEMAIN in DBM1. User thread storage for allied and database access threads (DBATs) is a user of this storage. DB2 storage is mostly allocated in SP229 Key 7.

The amount of virtual storage being used is collected by RMF via VSTOR(D, xxxxDBM1), and reported in the Virtual Storage Private Area Report (VSTOR) through SMF Type 78-2 records. The option REPORTS(VSTOR(D, xxxxDBM1)) will give you an idea of the total virtual storage consumption. In DB2, you can use instrumentation facility component identifiers (IFCIDs) 225 and 217 to see the consumption of virtual storage in DBM1.

Real and Auxiliary Storage

DB2 uses the ECSA and the z/OS Shared Memory Facility. In z/OS, the 64-bit address space includes the virtual line at the 16 MB address and a virtual line called the *bar* that marks the 2 GB address (Figure 2.3). Storage below the 2 GB address is referred to as *below the bar*. Storage above the 2 GB address is known as *above the bar*. The area above the bar is intended for data; no programs run above the bar.

No area above the bar is common to all address spaces, and no system control blocks exist above the bar. An IBM reserve area of storage is also available above the bar for special uses.

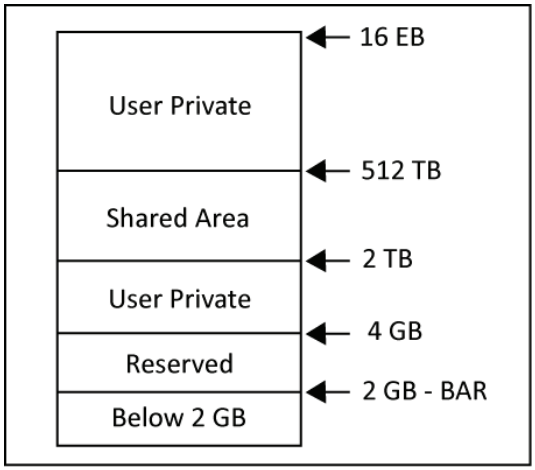

Figure 2.3: Memory

You can set a limit on how much virtual storage above the bar each address space can use. This limit is called the MEMLIMIT, which is a keyword on the JOB and EXEC statements. MEMLIMIT limits the total amount of usable virtual storage above the bar for a single user.

- If you do not set a MEMLIMIT, the system default is 0, meaning that no address space can use virtual storage above the bar.
- To use virtual storage above the bar, you must set the MEMLIMIT explicitly. You can set an installation default MEMLIMIT through the System Management Facility (SMF) SMFPRM*xx* in the PARMLIB.
- You can also set a MEMLIMIT for a specific address space in the Job Control Language (JCL) that creates the address space or by using the SMF exit routine IEFUSI.
- The default takes effect if a job does not specify MEMLIMIT on the JCL JOB or EXEC statement or REGION=0 in the JCL. The MEMLIMIT specified in an IEFUSI exit routine overrides all other MEMLIMIT settings.
- If the job specifies REGION=0 in the JCL and the IEFUSI exit routine limits the REGION size but does not set MEMLIMIT, MEMLIMIT defaults to the REGION size above 16 MB.

Distributed Data Facility

The DDF address space is started as part of the startup procedure in DSN1DIST. The DB2 11 improvements for DDF are available in all modes. Note that DDF uses 64-bit storage.

Processing Flow

DDF reduces the CPU processing time by using service request blocks (SRBs), not task control blocks (TCBs). Tasks can request an SRB to be scheduled that in turn requests a service to take place. The requested service can be in the same address space as the SRB or a different address space. The data that the task and the service share must be in common storage.

Enclave

An enclave is an independent dispatchable unit of work that can span multiple address spaces, and can include multiple SRBs and TCBs. The enclave is an anchor for accumulating the resources that a transaction consumes regardless of where it may be executing. It provides a way to account for resources consumed by multiple work units, even across multiple address spaces. The MVS System Resource Manager (SRM) manages the enclaves separately according to their goals or priorities.

DB2 owns all the enclaves coming into the system through DDF, and they are created by DDF on the incoming connection when the first SQL statement starts to execute.

- A connection request comes to DDF and is associated with a DBAT.
- In the first SQL statement, DDF calls WLM to create the enclave.
- The enclave is the basis for assigning resources to the DDF transaction.
- You assign performance goals to enclave transactions.
- The enclave is the basis of reporting thread performance.
- WLM manages the enclave based on assigned workload characteristics.

Deletion of an enclave depends on whether the DBAT can be pooled.

- If DBAT can be pooled, the enclave is deleted.
- If DBAT cannot be pooled, the enclave is deleted only at thread termination time.
- If DBAT becomes type 1 inactive (private protocol connection), the enclave is deleted.

When a request comes in over TCP/IP using DRDA, DB2 will schedule an SRB. DB2 will then create an enclave for each transaction and classify the request. When queries access DB2 via DRDA over TCP/IP, connections are dispatched within z/OS as enclave SRBs. A portion of this enclave SRB work can be directed to a z/System Integrated Information Processor (zIIP) engine. Only DRDA work coming from TCP/IP is zIIP-engine eligible. DB2 notifies the WLM that an enclave is eligible to direct a portion of the work to a zIIP processor. WLM along with the z/OS dispatcher will dispatch the work to a zIIP or to a general processor.

Management of the work to DDF is done with the use of MVS enclaves to exchange data across address spaces and the WLM. Running in DDF, these processes can access the database address space by using cross-memory services (CMS). Through CMS, the data and programs can be synchronously accessed in different address spaces. DDF, DBM1, and IRLM allocate control blocks above the 2 GB bar in 64-bit addressing mode.

Other DDF Information

Native stored procedures invoked from DRDA over TCP/IP, or queries that access DB2 via DRDA over TCP/IP, are dispatched in z/OS as enclave SRBs. A portion of their work can be directed to the zIIP engine.

DDF and z/OS Shared Virtual Memory

DB2 11 for z/OS supports 64-bit addressability in DDF by using z/OS shared virtual memory (SVM). SVM is a virtual storage type that allows multiple address spaces to share storage. The shared memory exists only once in z/OS, instead of in each address space. SVM is available to address spaces that are registered with z/OS as being able to share this storage.

Before V9, when DDF invoked DB2 to process a request, the data was copied from the DDF address space to the DBM1 address space, and then copied back to DDF at the completion of the request by using cross-memory moves. Virtual storage in the DBM1 address space below the 2 GB bar is reduced by using DDF running in 64-bit mode and the ability to access SVM areas. This virtual storage type allows multiple address spaces to share storage; it is always addressable and avoids cross-memory (XM) moves between DDF and DBM1, reducing data formatting and data movement, and improving performance.

The DSN1DBM1 creates a virtual shared object (VSO) in SVM at DB2 initialization time. As DSN1MSTR, DSN1DBM1, and DSN1DIST go through their local storage initialization, they are registered to use this VSO.

For DB2, in IEASYS*xx*, you have 1 TB of contiguous shared private storage per DB2 in the HVSHARE parameter. Whereas the HVCOMMON parameter provides 6 GB of contiguous share extended private storage per DB2.

To permit access to the VSO, all DB2 address spaces for the subsystem are registered with z/OS. These address spaces are registered with z/OS to allow sharing of this storage and to provide visibility to this storage.

A memory object in SVM is a contiguous range of virtual addresses.

- These pages are allocated by programs as a number of application pages.
- The pages are in 1 MB multiples on a 1 MB boundary.
- They exist once for each address space.

At the time each address space is terminated during shutdown, they request their interest in the VSO be deleted. At DB2 termination, the shared memory object is freed. It is interesting to note that almost all the control blocks for DDF have moved from ECSA to shared memory.

DB2 utilities CHECK INDEX, LOAD, REBUILD, REORG, and RUNSTATS also exploit the DB2 VSO. Using shared memory objects avoids the movement of rows between the batch and DBM1 address spaces and reduces CPU usage. The environmental descriptor manager (EDM) pool also takes advantage of VSO.

Shared Memory Objects

The size of this shared memory addressing area is between 2 TB and 512 TB. To use the shared memory, an address space must be registered because no automatic addressability or access is available to it.

Use the macro IARV64 in z/OS to provide virtual storage services for DB2. The address spaces are registered and controlled by the z/OS HVSHARE parameter in IEASYSxx in the PARMLIB. The HVSHARE parameter in IARV64 dictates how much virtual storage can be shared, and it allows multiple address spaces to share virtual storage above 2 GB. Define a large enough value for HVSHARE to satisfy all component requests for shared memory within your z/OS image.

To see the currently defined storage and the total amount of storage currently allocated, you can issue this z/OS command:

```
DISPLAY VIRTSTOR,HVSHARE
```

The following parameters allocate and allow access to data in a shared memory object by using the IARV64 macro parameters. To allocate the shared memory object, execute the GETSHARED service:

```
IARV64 GETSHARED
```

Create a memory object that can be shared across multiple address spaces:

```
IARV64 SHAREMEMOBJ
```

Parameters define the program request to obtain access to the shared memory object. An address space can issue more than one SHAREMEMOBJ request for the same memory object. Separate each of the requests for the same memory object by specifying a different user token.

When tracking the space usage in DDF, use IFCIDs 0225 and 0217. In both 0217 and 0225, fields have been added to record the amount of virtual shared storage used by the ssnmDBM1 address space. In 0225, the trace records are changed from SMF type 102 to SMF type 100 subtype 4. Make sure that the statistics class 6 is activated to obtain this information. For each DB2 subsystem above the 2 GB bar, DB2 requires 1 TB of 64-bit shared private storage.

Shared Memory Storage Requirements Restriction

This shared memory storage enhancement requires the following (Figure 2.4 shows the data shared area):

- z/Architecture, z/OS V1R13 or later
- Enablement of 64-bit virtual shared storage
- System configured with sufficient shared private storage to allow all shared storage exploiters on this LPAR to allocate their shared objects

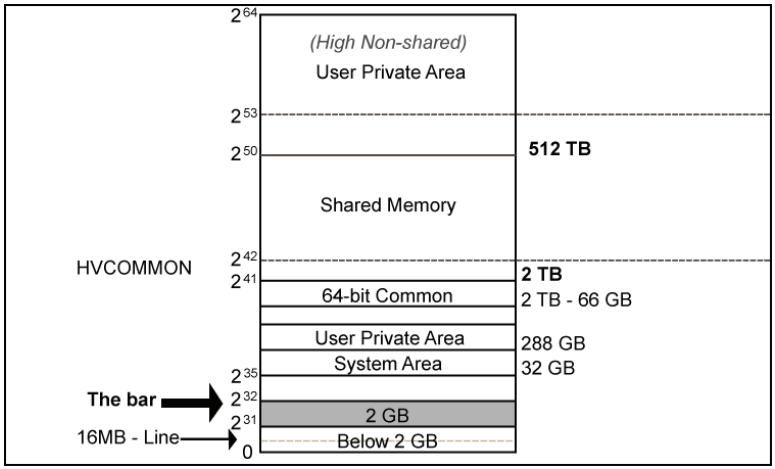

Figure 2.4: Data shared area

DSNZPARM Parameters for DDF

Distributed thread processing support is handled in DSNZPARM parameters MAXTYPE1, CMTSTAT, CONDBAT, and MAXDBAT, which are in DSN6SYSP and DSN6FAC macros.

DBATs in DBM1 are distributed thread connections. Thread pooling can occur in DB2 z/OS—the threads are actually inactive connections waiting for more work and associated with a remote requester.

```
INACTIVE DBAT:
```

An inactive DBAT (in the past called a *type 1 inactive thread*) has the same characteristics as inactive threads that were available in releases before DB2 11, and require a large number of threads supporting large number of connections.

We define the number of inactive DBATs in DSN6FAC in MAXTYPE1. But take care when setting this parameter. Overallocation can adversely affect performance on the system.

```
INACTIVE CONNECTION:
```

An inactive connection (previously called a *type 2 inactive thread*) uses less storage than an inactive DBAT and is preferred, but not all threads can be inactive connections. The connections are disassociated from the thread.