CHAPTER 2

DB2 for z/OS Overview

This chapter reviews the tasks, services, structure, architecture, and components of DB2 9 for z/OS that constitute required knowledge for a DB2 system administrator.

CSECTs and Subcomponents

Let's begin with a short overview of how the DB2 code is structured internally:

- In DB2, each object module contains a single control section (CSECT).
- A CSECT typically performs one function, and the object module and CSECT have the same name.
- Member **DSNWMODS** in library **SDSNSAMP** contains the readable data set associated with a CSECT for DB2.
- CSECT names and message identifiers begin with the letters "DSN" in DB2.
- The fourth character of a DB2 CSECT name identifies a subcomponent. For example, the prefix **DSNJ** indicates the recovery log manager subcomponent, and these letters are used for the module and message prefixes related to that facility.

DB2 subcomponents are groups of closely related DB2 for z/OS modules that work together to provide a general function. There are three groups of subcomponents in DB2:

- System services
- Database services
- Distributed Data Facility (DDF) services

DB2 Resource Managers

The software that comprises the DB2 resource managers is usually responsible for managing a specific resource. The resource being managed can be physical or logical. DB2 usually has one subcomponent per resource manager, but exceptions exist. 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. The RMID indicates the source of diagnostic output in your dumps. For your reference, the appendix at the end of this book provides a list of subcomponents and identifiers.

Address Spaces

In DB2, there are four major address spaces, some of which are known by several different acronyms:

- The main address space (**DSN1MSTR**), also known as the system services address space (SSAS) or the Data Systems Control Facility (DSCF)
- The data manager address space (**DSN1DBM1**), also known as the database services address space (DBAS or DSAS) or the Advanced Database Management Facility (ADMF)
- The Distributed Data Facility address space (**DSN1DIST**)
- The Internal Resource Lock Manager address space (**IRLMPROC**)

System Services

System services manage logs, agent services, and more by executing various subcomponents in the system services address space. This address space is also called the Data Systems Control Facility space.

Here are a few of the subcomponents that execute in the SSAS:

- 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

Database Services

Database services use system services and z/OS to handle the actual database structures. The database services address space consists of three main components:

- Buffer manager
- Data manager (DM)
- Relational data system

The function of the DBAS is to manage the physical structures and data, execute SQL, and manage the buffers. Even though these are independent components, they work together to make a proper subsystem of z/OS. The database services address space is also referred to as the Advanced Database Management Facility address space.

Subcomponents of interest that execute in the DBAS are:

- LOB manager
- Service controller
- Stored procedures manager
- Data space manager
- Utilities (these work with associated code in an allied address space)

Distributed Data Facility Services

Running as an additional address space in DB2, the DDF services consist of one subcomponent called the Distributed Data Facility. DDF controls the connecting of distributed applications to DB2 for z/OS. The naming convention for this subsystem is ***xxx*DIST**.

Four resource managers are associated with DDF:

- Data Communications Resource Manager
- Distributed Data Interchange Services
- Distributed Relational Data System Manager
- Distributed Transaction Manager

These subcomponents execute in the DDF address space.

DB2 Distributed Relational Data Architecture (DRDA) subsystems and other relational databases can communicate with DDF by using Transmission Control Protocol/Internet Protocol (TCP/IP) or Virtual Telecommunications Access Method (VTAM) on the same network. DDF supports two network protocols, Systems Network Architecture (SNA) and TCP/IP, as well as the DRDA database communications protocol.

DRDA is set of database protocols that describe the architecture that allows 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

coordinates the communications between systems. Three components make up DRDA:

- Application requestor
- Application server
- Database server

Internal Resource Lock Manager

DB2 also requires the address space subsystem services of the Internal Resource Lock Manager (IRLM), which resides in its own address space. (Note that this "IRLM" is different from the Information Management System, or IMS, Resource Lock Manager.) The lock manager works with DB2 to serialize access to data. DB2 requests locks from IRLM to ensure data integrity when applications, utilities, and commands all attempt to access the same data.

In DB2 9, you must continue to specify the IRLM-related subsystem parameters **PC** and **MAXCSA**, but their values are no longer used. IBM has retained these parameters for compatibility reasons. Specific system site specifications now 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 no longer placed in the extended common service area (ECSA). IRLM control block structures are estimated at 540 bytes per lock and reside above the 2 GB bar. **DXB**, not **DSN**, is the prefix for IRLM.

Other Address Spaces

DB2 communicates with other address spaces, known as *allied address spaces*, in the z/OS environment. DB2 communicates with these "allied agents" to facilitate requests. Here is a list of allied agents with which DB2 communicates:

- Time Sharing Option (TSO) attachment facility
- Subsystem support
- Message generator, stand-alone only (**DSN1SDMP**)
- IMS attachment facility

- Call attachment facility
- Customer Information Control System (CICS) attachment facility
- Resource Recovery Services (RRF) attachment facility
- Utilities

Connections or threads to these allied agents are controlled through the subsystem parameter **CTHREAD** (which defaults to **200**, has a maximum value of **2000**, and is updatable online). The **CTHREAD** setting defines the number of concurrently allocated threads for local connections. If you find that you are waiting for a connection to access the DB2 subsystem, you might need to increase the number of allied connections specified through **CTHREAD**.

The **CTHREAD** setting, along with the **MAXDBAT** DSNZPARM, protects the virtual storage allocation. Be careful not to overcommit your virtual storage resources. If the number of remote threads is queued with work waiting, you might need to increase the **MAXDBAT** value.

Utilities use parallelism, so you will have one thread for each utility and an additional thread for each subtask. Thus, a single utility may be using many threads. You need to make sure to specify a **CTHREAD** value that will accommodate utility parallelism.

Non-allied address spaces do not communicate with DB2. Several subcomponents execute in non-allied address spaces:

- DB2 does not communicate with the DB2 precompiler (PRE), but 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 system services address space. The message generator can also run stand-alone in allied or non-allied address spaces.
- Portions of the instrumentation subcomponent run in a non-allied address space.
- DB2 stand-alone utilities run in non-allied address spaces.

Work Requests in DB2

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

A one-to-one relationship exists between a z/OS execution unit and an execution block. An EB is used to describe each unique unit of dispatch work, which can be dispatched in either task control block (TCB) or service request block (SRB) mode in z/OS. All allied agents to which the primary EB is related point to the user's home address TCB.

In the DB2 address space, when execution units are created in TCB mode, they are known as *service tasks*. Resource managers in DB2 can dynamically delete and create service tasks. When you initialize DB2, service tasks are created, and these usually exist until DB2 is stopped. The service tasks remain idle until their services are needed in DB2.

Examples of permanent service tasks include the following:

- System service tasks
- Log manager
- Recovery manager
- Database services tasks
- Buffer manager
- Data manager
- DDF tasks
- Distributed transaction manager

DB2 9 System Structure Basics

Figure 2.1 provides an overview of the DB2 subsystems.

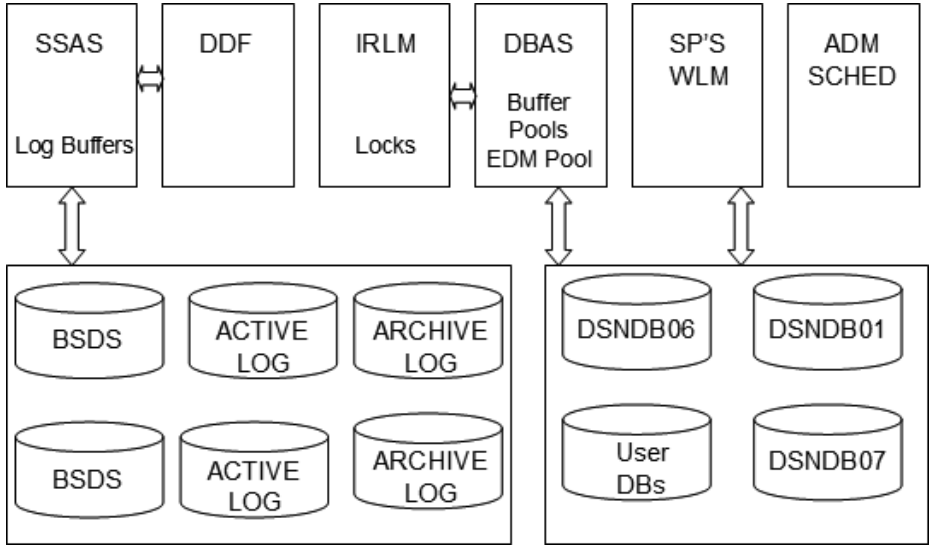

Figure 2.1: Subsystem overview

DB2 uses several types of private address spaces, each requiring storage:

- DB2 system services address space (**DSN1MSTR**)
- DB2 database services address space (**DSN1DBM1**)
- DB2 DDF address space (**DSN1DIST**)
- IRLM address space (**IRLMPROC**)
- DB2 allied agent address spaces
- DB2 stored procedures address spaces (established by the Workload Manager, or WLM)
- DB2 administrative scheduler address space

When you start your DB2 subsystems, there is a recommended dispatching priority for these address spaces in z/OS: Without locking to protect your resources, you

cannot begin, so IRLM is started first. Next, you start the DB2 performance monitors, then the **DBM1** address space, and then the **MSTR** address space.

Attachment Facilities

An *attachment facility* provides the interface between DB2 and another environment, such as TSO. In TSO and your batch environments, you can use the TSO, call, and Resource Recovery Services (RRS) attachment facilities to access DB2. Other attachment facilities, including those for CICS and IMS, are DB2 subcomponents that run in the user's address space.

Figure 2.2 depicts how the various attachment facilities interact with DB2.

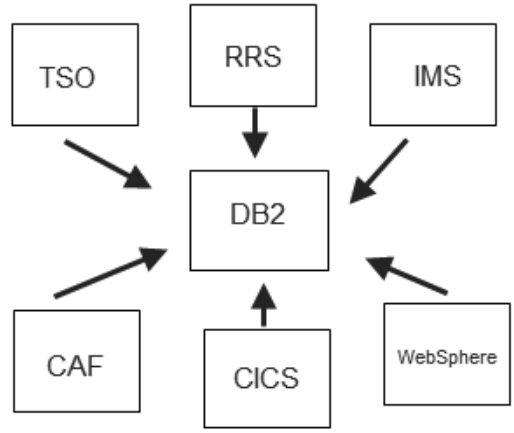

Figure 2.2: Attachment facilities

Call Attachment Facility

For TSO and batch applications that require tight control over their session environment, DB2 provides the Call Attachment Facility (CAF) as an option for connection. Programs can explicitly control the state of their connections to DB2 by using connection functions supplied by CAF.

First, make available the CAF load module, **DSNALI**. Once this language interface module 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 writing explicit **CALL DSNALI** statements.

Resource Recovery Services

DB2 supports Resource Recovery Services, which is a newer implementation of an attachment capability. This z/OS system feature coordinates the two-phase commit processing of recoverable resources.

RRS runs in its own address space and can be started and stopped independently of DB2. Once z/OS RRS is started, you can run the Resource Recovery Services Attachment Facility (RRSAF) application. RRS needs to be at an equal or higher priority than the dispatching priority of DB2.

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

Stored procedures running in a WLM address space require the RRS attachment facility. Resources such as SQL tables, Data Language/I (DL/I) databases, MQSeries message programs running in batch or TSO, and recoverable Virtual Storage Access Method (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 a thread is allocated, the storage used by that thread is held until de-allocation. The **CONTSTOR** parameter in macro **DSN6SPRM** controls each active thread's working storage area. The default value for **CONTSTOR** is **NO**. If you are experiencing a high private virtual storage usage in **DBM1**, setting **CONTSTOR** to **YES** may reduce the unused storage. This setting enables DB2 to periodically check the thread storage that is unused from a committing process and return that storage

to the operating system. You can set the **CONTSTOR** parameter dynamically using the DB2 **SET SYSPARM** command.

DB2 will examine the thread use of the storage pool, and if the thread has used more than 2 MB or if there are more than 50 commits, the storage blocks that have been used and are no longer in use are freed and returned to the operating system. This procedure can reduce the amount of virtual storage used in **DBM1**, especially for long-running threads. However, you incur associated CPU overhead from **GETMAIN** and **FREEMAIN** requests to the operating system, so you need to carefully consider the benefits.

One other thing to keep in mind: Use of the **RELEASE**(**DEALLOCATE**) parameter on the **BIND** command requires more virtual storage due to the increase in the size of the package or plan that occurs with this parameter. Specifying **RELEASE(DEALLOCATE)** nullifies the process of using **CONTSTOR YES**, and no storage contraction will take place. That is because a **COMMIT** does not trigger the **CONTSTOR YES** processing. Until the thread is actually deallocated, it just keeps getting bigger.

Storage

DB2 allocates different subpools for storage. Storage pool 229 (**SP229**) is storage acquired by a **GETMAIN** request and released by a **FREEMAIN** request in **DBM1**. Allied and database access threads are users of this storage.

The z/OS Resource Measurement Facility (RMF) tracks how much virtual storage you are using and reports this information in the Virtual Storage Private Area Report (VSTOR) in System Management Facility (SMF) type 78-2 records. To get an idea of the total virtual storage consumption, you can set the option **REPORTS(VSTOR(D, *xxxx*DBM1))**. In DB2, you can use Instrumentation Facility Component Identifiers (IFCIDs) 217 and 225 to view the consumption of virtual storage in **DBM1**.

Real and Auxiliary Storage

DB2 uses the extended common service area of z/OS virtual storage and the z/OS Shared Memory Facility. In z/OS, the 64-bit address space includes a virtual line at

the 16 MB address and a virtual line called “the bar” that marks the 2 GB address. Storage below the 2 GB address is referred to as “below the bar,” and storage above this address is “above the bar.” The area above the bar is intended for data, and no programs run above the bar. Figure 2.3 depicts the memory areas of z/OS.

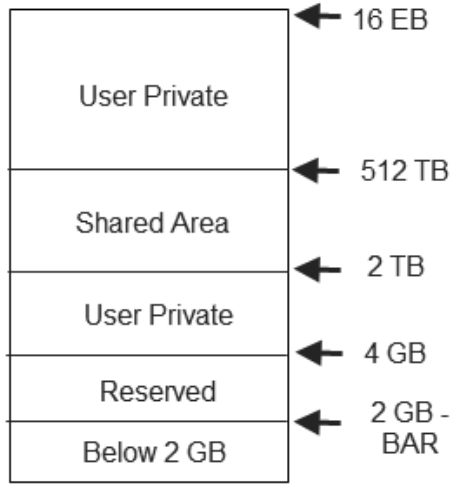

Figure 2.3: z/OS memory

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

You can set a limit on how much virtual storage above the bar each address space can use. This limit is represented by the **MEMLIMIT** keyword on the **JOB** and **EXEC** statements. The **MEMLIMIT** setting 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.
- If you want to use virtual storage above the bar, you must set a **MEMLIMIT** explicitly. You can set an installation default **MEMLIMIT** through SMF member **SMFPRM*xx*** in **PARMLIB**.
- You can also set a **MEMLIMIT** for a specific address space either in the JCL that creates the address space or by using SMF exit **IEFUSI**.

- The default takes effect if a job does not specify **MEMLIMIT** on the JCL **JOB** or **EXEC** statement or set **REGION=0** in the JCL.
- The **MEMLIMIT** specified in an **IEFUSI** exit routine overrides all other **MEMLIMIT** settings.
- If **REGION=0** is specified in the JCL and the **IEFUSI** exit 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 the **DSN1DIST** address space. The DB2 9 improvements for DDF are available in all modes, and DDF now uses 64-bit storage.

Processing Flow

DDF reduces the CPU processing time by using service request blocks rather than task control blocks. Tasks can request that an SRB be scheduled that requests a service to take place. The SRB can be in the same address space or a different one. The data that is shared by the task and the service must reside 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 consumed by a transaction 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 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:

1. A connection request comes to **DDF**, associated with a DBAT.

2. At 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.
3. WLM manages the enclave based on assigned workload characteristics.

Deletion of an enclave depends on whether the DBAT can become pooled:

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

When a request comes in over TCP/IP using DRDA, DB2 schedules an SRB. DB2 then creates an enclave for each transaction and classifies the request. When queries access DB2 via DRDA over TCP/IP, connections are dispatched within z/OS as enclave SRBs. A portion of each 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, dispatches 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.

The **DDF**, **DBM1**, and **IRLM** address spaces 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, and a portion of their work can be directed to the zIIP engine.

If DRDA work is requested via SNA, it will not be zIIP-eligible. Some batch and non-native SQL stored procedures are not implemented in SRB mode or in enclave SRBs that would not be eligible for zIIP processing.

In DB2 9, you can also still communicate using private protocol from DB2 to DB2, although this method is not recommended. You should take a look at the tools provided in DB2 9 to help you migrate away from private protocol. One of those tools is the private to DRDA protocol REXX tool, **DSNTP2DP**. During migration, the customized **DSNTIJPD** job invokes the **DSNTP2DP** tool. You can no longer **BIND** plans and packages for private protocol because **DSN6SYSP.DBPROTCL** in the DSNZPARMs has been eliminated. To help with the migration task, the generic collection ID **DSNCOLLID** is defined to maintain collections of remote packages.

DDF and z/OS Shared Virtual Memory

DB2 9 for z/OS supports 64-bit addressability in DDF using z/OS *shared virtual memory (SVM)*. SVM is a new virtual storage type that permits 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 DB2 9, 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 using cross-memory (XM) moves. Virtual storage in the **DBM1** address space below the 2 GB bar is reduced in DB2 9 by running DDF in 64-bit mode and accessing the SVM areas. This new virtual storage type lets multiple address spaces share storage, is always addressable, avoids access register (AR) and XM moves between **DDF** and **DBM1**, reduces data formatting and data movement, and improves performance.

The database services address space creates a new virtual shared object (VSO) in shared virtual memory at DB2 initialization time. As the **DBM1**, **MSTR**, and **DIST** address spaces go through their local storage initialization, they are registered to use this VSO.

DB2 startup requires 128 GB of 64-bit shared private storage for each DB2 subsystem above the 2 GB bar for shared memory objects. The default size is 2 TB; DB2 requires a minimum of 128 GB.

All DB2 address spaces for the subsystem are registered with z/OS to access the virtual shared object. The address spaces are registered with z/OS to be able to share this storage and also have 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, it requests that its 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.

The DB2 utilities **CHECK INDEX**, **LOAD**, **REBUILD**, **REORG**, and **RUNSTATS** also take advantage of the DB2 VSO. The use of 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 the shared memory addressing area is between 2 TB and 512 TB. To use this memory, an address space must be registered; there is no automatic addressability or access to it. The macro **IARV64** in z/OS provides virtual storage

services for DB2. The z/OS parameter **HVSHARE** in member **IEASYS*xx*** in library **PARMLIB** registers and controls the address spaces.

In the **IARV64** macro, parameter **HVSHARE** governs how much virtual storage can be shared and permits multiple address spaces to share virtual storage above 2 GB. Be sure to define a high enough value for **HVSHARE** to satisfy all component requests for shared memory within your z/OS image.

To see the current defined storage and how much total storage is currently allocated, you can issue this z/OS command:

```
DISPLAY VIRTSTOR,HVSHARE
```

Two **IARV64** macro parameters allocate and allow access to data in a shared memory object:

- **IARV64 GETSHARED**
  - » The shared memory object is allocated by the **GETSHARED** service.
  - » This service creates a memory object that can be shared across multiple address spaces.
- **IARV64 SHAREMEMOBJ**
  - » Parameters define the program request to the **SHAREMEMOBJ** service to get access to the shared memory object.
  - » An address space can issue more than one **SHAREMEMOBJ** request for the same memory object.
  - » You separate each request for the same memory object by specifying a different user token.

IFCIDs 217 and 225 track the space usage in DDF. In both 217 and 225, fields have been added to record the amount of virtual shared storage used by the ***ssnm*DBM1** address space. In 225, the trace records are changed from SMF type 102 to SMF type 100 subtype 4. To obtain this information, you need to make sure that statistics class 6 is activated. For each DB2 subsystem above the 2 GB bar, DB2 requires 128 GB of 64-bit shared private storage.

Shared Memory Storage Requirements Restriction

When using shared memory, you should see a decrease in overall storage allocation to **DSN1DIST**. Note that shared memory is not charged against the **MEMLIMIT** of an address space.

The shared memory storage enhancement requires

- z/Architecture, z/OS 1.7 or later
- Enablement of 64-bit virtual shared storage
- Sufficient shared private storage configured to allow all shared storage exploiters on the logical partition (LPAR) to allocate their shared objects

Figure 2.4 depicts multiple address spaces sharing storage. This configuration reduces data movement and data formatting and decreases virtual storage because the data exists only once. The data in the shared area can be shared across address spaces X, Y, and Z.

Figure 2.4: Data shared area

Informational APAR II14203 describes the TCP/IP and z/OS prerequisite APARs that are required to enable the shared memory enhancement. To take advantage of this functionality, make sure TCP/IP and z/OS have the required maintenance

before migrating to DB2 9 for z/OS. Shared virtual memory resides above the 2 GB bar in z/OS and is organized as a virtual shared object that programs create.

Distributed Thread Processing

Database access threads in the **DBM1** address space are distributed thread connections. DB2 for z/OS supports thread pooling, which is actually inactive connections waiting for more work and associated with a remote requester.

An *inactive DBAT* (formerly called a type 1 inactive thread) has the same characteristics as the inactive threads that were available in releases before DB2 9. Inactive DBATs require a large number of threads to support a large number of connections. We refer to inactive type 1 DBATs as "real DBATs." These threads are idle between units of work. They use private protocol and the old style of inactive processing.

The **MAXTYPE1** parameter in macro **DSN6FAC** defines the number of inactive DBATs permitted by DB2. Be careful when setting this parameter because over-allocation can adversely affect system performance.

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

Threads that are not currently processing a unit of work are called *pooled threads*. Pooled threads can be reused for other connections, either new or inactive. Pooled threads typically represent a small number of threads that can be used to service a large number of connections and provide better resource utilization.

These DDF threads are defined in macro **DSN6FAC** by parameter **CMTSTAT**. To see the inactive type 2 DBATs in your system, use this **DISPLAY THREAD** command:

```
-DISPLAY THREAD TYPE(INACTIVE)
```

The purpose of a DBAT is to reuse an existing connection in **DBM1**. Without this pooling, each new connection request would create a new thread (DBAT) in **DBM1** each time. A pool of DBATs is created and maintained dynamically for use by any

inbound DRDA connection. When work is committed, the DBAT is released back to the pool. With or without connection pooling, DDF creates a new connection when an application or agent request is made to DDF in DB2.

There is a separation of connections in DDF from the DBATs in **DBM1**. This separation results in no repeated creation and destruction of DBATs, which saves CPU time and reduces the memory requirements for DBATs and virtual storage. See PK75626 for this reference; PK77228 provides further information.

DSNZPARM Values to Control Threads

Several subsystem parameters work in conjunction with parameter **MAXTYPE1** to handle distributed thread processing support:

- **CMSTAT**: The **CMSTAT** parameter (field **DDF THREADS** on installation panel **DSNTIPR**) specifies the status a thread should take after a commit or rollback or when it no longer holds cursors against resources. The default value is **ACTIVE**. Change this setting to **INACTIVE** to enable DBAT pooling for DRDA access and more effective WLM classification per unit of work.
- **POOLINAC**: The **POOLINAC** parameter (field **POOL THREAD TIMEOUT** on installation panel **DSNTIP5**) sets a timeout value for the automatic termination of idle DBATs. The default setting is **120** (seconds).
- **MAXDBAT**: The **MAXDBAT** parameter (field **MAX REMOTE ACTIVE** on installation panel **DSNTIPE**) specifies the maximum number of concurrently active DBATs. As you find threads queued for remote work, this value should increase. The default value is **200**, and the setting cannot exceed **2000**. If you raise or lower the **MAXDBAT** value, you also raise or lower the storage requirements below the 2 GB bar in **DBM1**.
- **CONDBAT**: The **CONDBAT** parameter (field **MAX REMOTE CONNECTED** on installation panel **DSNTIPE**) sets a limit on the total number of inbound DDF connections. It is highly recommended that the setting for **MAXDBAT** be less than that for **CONDBAT**.

A rule of thumb is to set **CMTSTAT** to **INACTIVE**, adjust **CONDBAT** to the maximum number of connected DBATs that provide good performance, and set **MAXDBAT** to the maximum acceptable number of active DBATs.

To determine the total number of threads that can access data in DB2, add the values of **MAXDBAT** and **CTHREAD**. IFCID 225 provides details for major consumers. If you overextend the number of threads, you can potentially overextend virtual storage and cause DB2 to abend with abend code 878 (no storage available).

To help you monitor these situations and choose values for **MAXDBAT** and **CTHREAD**, IBM provides a no-cost program that writes to a comma-delimited trace output data set from IFCID 225 (SMF 102) for analysis. You can obtain this utility by going to *http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/PRS3431* (document ID PRS3431 by Judy Ruby-Brown at IBM).

Depending on the communications protocol used, each DDF connection consumes approximately 7.5 KB of memory inside the **DIST** address space. In addition, each active DBAT consumes about 200 KB of memory at a minimum, depending on the type of SQL activity requested.

Private Protocol Access

We know that private protocol is going away in a future DB2 release. In your efforts to move away from it, you need to determine which applications use private protocol access and which sites those applications access. To obtain this information, run the following performance trace for IFCIDs 157 and 168:

```
-START TRACE(PERFM) CLASS(30) IFCID(157,168) DEST(GTF)
```

The trace report will show:

- SQL statements that reference aliases before they have been resolved
- Whether the package or plan that the statement is running under was bound with **DBPROTOCOL(PRIVATE)**
- The aliases that are referenced in the statement

To display additional details about the DDF environment, enter this command:

```
-DISPLAY DDF DETAIL
```

Native Stored Procedures in DB2 9

Native stored procedures — that is, those whose bodies are written entirely in SQL — are eligible for zIIP engine processing if they are invoked from DRDA TCP/IP connections. SQL queries that access DB2 via DRDA over TCP/IP connections are dispatched within z/OS as enclave SRBs, and z/OS directs a portion of this work to the zIIP. DRDA work that is requested via SNA is not zIIP-eligible. Native stored procedures are stored in the DB2 directory.

Database Management

The DB2 catalog is a database called **DSNDB06** that contains table spaces that reflect your objects, security, and access packages and plans. Most DB2 system table spaces use the naming convention **SYSIBM.SYS*xxxxxx*** — for example, **SYSIBM.SYSTABLESPACE**.

You can have the catalog and user-defined databases, table spaces, and index spaces start automatically when DB2 is started. To specify this, use the **RESTART** option and the object definition **ALL** on the **DSNTIPS** installation panel.

Indexes on the DB2 catalog or directory tables are no different from indexes on any other tables created in DB2. DB2 catalog data can be accessed via indexes and links; there are no hashes in the catalog.

Before migration to DB2 9 for z/OS, a **REORG** of your catalog is recommended. You should increase the size of the underlying VSAM clusters/linear data sets for the catalog and directory before migrating.

In DB2 9, you can define up to 500 indexes against DB2 catalog tables. Each release of DB2 increases the number of table spaces, tables, and indexes in the **DSNDB06** database, so you always need to keep in mind the new space requirements for these additional objects.

Sizing of the catalog has also increased. The default sizes for an installation are specified in megabytes and range from a small-site size of 199 MB to 747 MB for an extra-large site.

Even though the **ALTER** statement has become a powerful tool for online schema changes, you cannot alter any column of the DB2 catalog.

Because of object changes in DB2 9 (e.g., the maximum number of partitions supported in a table space), IBM has made adjustments to the catalog tables. For example, catalog tables have been changed to reflect the **MAXPARTITIONS** setting in **SYSIBM.SYSTABLESPACE**, which is the value you specified in the **CREATE** or **ALTER** statement. Be careful, because this setting does not reflect the physical number of existing partitions. You can find the allocated number of partitions in the **PARTITION** column; for example, you would look for a **G** for partition-by-growth table spaces.

Regardless of the number used for the **MAXPARTITIONS** parameter, only one row is added to **SYSIBM.SYSTABLEPART** when you create the table space. Additional rows are added to this catalog table as your table grows, and additional partitions are allocated if the amount of data exceeds the associated data set size (**DSSIZE**).

Three new **SYSIBM.SYSSTOGROUP** columns — **DATACLAS**, **MGMTCLAS**, and **STORCLAS** — contain the storage management subsystem (SMS) classes used on the **CREATE STOGROUP** or **ALTER STOGROUP** statement. If these new parameters are not defined, DB2 uses the management class and storage class assigned to the corresponding automatic class section (ACS) routine.

DB2 9 Catalog Tables

Table 2.1 describes the catalog tables that are new in DB2 9 for z/OS.

Table 2.1: New catalog tables in DB2 9 for z/OS

Table	Description
SYSIBM.SYSCONTEXT	Contains one row for each trusted context.
SYSIBM.SYSCONTEXTAUTHIDS	Contains one row for each authorization ID with which the trusted context can be used.

2.1: New catalog tables in DB2 9 for z/OS (continued)

Table	Description
SYSIBM.SYSCTXTTRUSTATTRS	Contains one row for each list of attributes for a given trusted context.
SYSIBM.SYSDEPENDENCIES	Records dependencies between objects.
SYSIBM.SYSENVIRONMENT	Records environment variables when an object is created.
SYSIBM.SYSINDEXSPACESTATS	Contains realtime statistics for index spaces.
SYSIBM.SYSJAVAPATHS	Records the complete Java archive (JAR) class resolution path and the dependencies that one JAR has on the JARs in its Java path.
SYSIBM.SYSKEYTARGETS	Contains one row for each key-target that is participating in an extended index definition.
SYSIBM.SYSKEYTARGETSTATS	Contains partition statistics for selected key-targets. For each key-target, a row exists for each partition in the table. Rows are inserted when the **RUNSTATS** utility collects indexed key statistics or non-indexed key statistics for a partitioned table space. No row is inserted if the table space is non-partitioned.
SYSIBM.SYSKEYTARGETS_HIST	Contains rows from the **SYSKEYTARGETS** table. When rows are added or changed in **SYSKEYTARGETS**, the rows are also written to this table.
SYSIBM.SYSKEYTGTDIST	Contains one or more rows for the first key-target of an extended index key.
SYSIBM.SYSKEYTGTDISTSTATS	Contains zero or more rows per partition for the first key-target of a data-partitioned secondary index. Rows are inserted when **RUNSTATS** scans a data-partitioned secondary index. No row is inserted if the index is a secondary index.
SYSIBM.SYSKEYTGTDIST_HIST	Contains rows from the **SYSKEYTGTDIST** table; whenever rows are added or changed in **SYSKEYTGTDIST**, the rows are also written to this table.
SYSIBM.SYSOBJROLEDEP	Lists the dependent objects for each role.
SYSIBM.SYSROLES	Contains one row for each role.
SYSIBM.SYSROUTINESTEXT	Serves as an auxiliary table for the **TEXT** column of **SYSIBM**.**SYSROUTINES** and is required to hold the large object (LOB) data.
SYSIBM.SYSTABLESPACESTATS	Contains realtime statistics for table spaces.
SYSIBM.SYSXMLRELS	Contains one row for each XML table that is created for an XML column.
SYSIBM.SYSXMLSTRINGS	Contains rows that each hold a single string and its unique ID that together are used to condense XML data. The string can be an element name, attribute name, name space prefix, or namespace uniform resource identifier (URI).
SYSIBM.XSRCOMPONENT	Serves as an auxiliary table for the binary large object (BLOB) column **COMPONENT** in **SYSIBM**.**SYSXSROBJECTCOMPONENTS**. It is located in LOB table space **SYSXSRA3**.

2.1: New catalog tables in DB2 9 for z/OS (continued)	
Table	Description
SYSIBM.XSROBJECTS	Contains one row for each registered XML schema. Rows can be changed only using static SQL statements issued by the DB2-supplied XML schema repository (XSR) stored procedures.
SYSIBM.XSROBJECTCOMPONENTS	Contains one row for each component (document) in an XML schema. Rows in this table can be changed only using static SQL statements issued by the DB2-supplied XSR stored procedures.
SYSIBM.XSROBJECTGRAMMAR	Serves as an auxiliary table for the BLOB column **GRAMMAR** in **SYSIBM.SYSXSROBJECTS**. It is located in LOB table space **SYSXSRA1**.
SYSIBM.XSROBJECTHIERARCHIES	Contains one row for each component (document) in an XML schema to record the XML schema document hierarchy relationship. Rows in this table can be changed only using static SQL statements issued by the DB2-supplied XSR stored procedures.
SYSIBM.XSROBJECTPROPERTY	Serves as an auxiliary table for the BLOB column **PROPERTIES** in **SYSIBM.SYSXSROBJECTS**. It is located In LOB table space **SYSXSRA2**.
SYSIBM.XSRPROPERTY	Serves as an auxiliary table for the BLOB column **COMPONENT** in **SYSIBM.SYSXSROBJECTCOMPONENTS**. It is located in LOB table space **SYSXSRA3**.

Realtime Statistics Tables in DB2 9

During DB2 enable-new-function (ENF) mode processing, job **DSNTIJEN** moves the realtime statistics data from your user-defined tables to the catalog tables **SYSIBM.SYSTABLESPACESTATS** and **SYSIBM.SYSINDEXSPACESTATS**. After the job moves the data to the catalog tables, you can drop the user-defined tables.

In conversion mode (CM), the realtime statistics data remains in the user-defined tables. If you revert to conversion* (CM*) mode, DB2 keeps the realtime statistics data in the catalog tables and does not use the user-defined tables.

Object Management in the Catalog

In DB2, the data manager handles the manipulation of the system catalog tables. The DM relies on database descriptors (DBDs) to manage data. Each DBD corresponds to a single database and contains subdescriptors called object descriptors (OBDs). The internal structure of a DBD is a complicated hierarchical

network of OBDs that are chained together. Each OBD has a unique identifier called an object identifier (OBID). The chain pointers that are used are the actual OBIDs, and DB2 employs an algorithm to locate an OBD within a DBD.

Management of the Catalog

In the day-to-day administration of the catalog, various utilities and jobs help you manage the physical structure and organization of the catalog.

To improve query performance, you should reorganize the indexes on the catalog tables to reduce table size and improve performance. You generally do not reorganize the entire set of catalog tables unless you are migrating to a new release of DB2. You might reorganize your catalog tables once a year, if that often. If you do decide to reorganize the catalog tables, keep in mind that there may be associated directory table spaces that also should be reorganized.

Table 2.2 lists the catalog table spaces and the corresponding directory table spaces that would require reorganization. For example, if you reorganize the **SYSPLAN** catalog table space, you would also reorganize the directory table space **SCT02**.

Table 2.2: Catalog and directory table spaces

Catalog table space (DSNDB06.*xxxx*)	Directory table space (DSNDB01.*xxxx*)
SYSDBASE	DBD01
SYSPLAN	SCT02
SYSPKAGE	SPT01

Remember to always take a full image copy before and after you reorganize catalog or directory objects. In addition, if you should need to recover the catalog or directory objects, you must do so in a particular order. All table spaces associated with the directory and the catalog must be recovered to the same point in time.

Some utilities that support the catalog, such as **COPY**, **LOAD**, **REBUILD INDEX**, **RECOVER**, **REORG**, and **TABLESPACE** have been updated to reflect changes in

DB2 9, such as the new partition-by-growth table space structure. The following utilities and installation jobs have also undergone changes:

- **DSN1COPY**: You can now use the **DSN1COPY** DB2 stand-alone utility to copy VSAM data sets. Remember that the row format (RRF or BRF) must be the same for the data set from which or into which you are copying. Using the utility's **CHECK** option, you can check each page of a data set.
- **DSNTIJID**: The **DSNTIJID** installation job initializes the system data sets associated with the bootstrap data set (BSDS), catalog, directory, and active logs.
- **DSNTIJTC**: Installation job **DSNTIJTC** invokes the **CATMAINT** utility to tailor your catalog. **CATMAINT** updates the catalog during the migration or installation of a new release of DB2. **DSNTIJTC** contains jobs that perform tailoring of the catalog. It also creates and updates indexes on catalog tables.
- **DSNTIJIC**: The **DSNTIJIC** job also provides an image copy of the catalog and directory for backup, enabling recovery of the catalog and directory. In DB2 9, the job has been modified to copy to disk instead of to tape, but it is limited to two disk volumes. You will have to make modifications to the job to change the number of disk volumes.
- **DSN1CHKR**: The **DSN1CHKR** stand-alone utility is a service aid used for verifying the integrity of the DB2 catalog and directory table spaces for potential data inconsistencies. It checks for broken links, or chains and records that are not part of any chain or link. The utility executes outside the control of DB2. Its use requires a detailed knowledge of DB2 data structures.
- **DSNTIJEN**: In ENFM* or ENFM, job **DSNTIJEN** invokes the **CATENFM** utility to update the catalog for the new release. If this job does not complete successfully, job **DSNTIJNF**, which is used to put DB2 into NFM, will return an error. After **DSNTIJEN** finishes, the catalog conversion is complete. See PK7728 for command list (CLIST) changes to job **DSNTIJEN**.

DB2 Directory

DSNDB01 is the name of the DB2 directory database. This underlying VSAM data set should be in your primary Integrated Catalog Facility (ICF) catalog. You cannot access the information in the directory using SQL. No descriptions of these structures are provided in the catalog for you to see.

The **DSNDB01** database consists of the five table spaces **DBD01**, **SCT02**, **SPT01**, **SYSLGRNX**, and **SYSUTILX**. Each table space is contained in a VSAM linear data set. An example of the naming convention is **DSNDB01.DBD01**. The size of the EDM pool (both above and below the bar) that supports the **DBD01**, **SCT02**, and **SPT01** table spaces is calculated during the installation process and displayed on the **DSNTIPC** panel. The general recommendation is to make the EDM pool 10 times the size of the largest DBD or plan, whichever is greater.

EDM Pool

The EDM pool is a system buffer pool that minimizes I/O against the catalog and the directory. It contains the database descriptor, the cursor table (CT), the package table (PT), the skeleton cursor table (SKCT), the skeleton package table (SKPT), the plan and package authorization cache, and a dynamic SQL skeleton for dynamic SQL caching. These are separate areas of storage, not part of one contiguous EDM pool. CTs and SKCTs result from a static **BIND** of a **PLAN**. PTs and SKPTs result from a static **BIND** of a **PACKAGE** (use of **ACQUIRE(USE)** is implied).

DB2 9 introduces some changes to the storage in the **DBM1** address space. A portion of the CTs and PTs are now above the 2 GB bar, along with a new component above the bar for SKPTs and SKCTs, called the EDM skeleton pool. This skeleton pool is set by a new parameter on installation panel **DSNTIPB**.

In DB2 9, **DBM1** below-the-bar storage relief for heavy package and plan activity is significant. To take advantage of all the improvements, you must perform a DB2 9 rebind. An average estimated reduction is from 20 percent to 90 percent. During the monitoring of your EDM pool pages in use, if this statistic is steadily less than 50 percent, your EDM pool size is probably too large. In general, EDM pool utilization should be around 80 percent.

In DB2 9 new-function mode, native SQL procedures are converted to a representation that is stored in the database directory as other SQL statements are. The stored procedure parameter list options are stored in the database catalog tables as in previous DB2 releases.

When you call a native SQL procedure in DB2 9, the procedure is loaded from the DB2 directory, and the DB2 engine then runs the procedure. Several additional functions and extensions in DB2 9 provide consistency with the SQL standards and the rest of the IBM DB2 family.

In DB2 9, the DB2 catalog and directory use buffer pool **BP0**. If the structure to process a statement is not already present, it is read into database buffer pool **BP0** in 4 KB pages and copied from there into the EDM pool areas. Where the sections are placed in the EDM pool depends on whether a package (SKPT, PT) or a plan (SKCT, CT) is being processed, as Figure 2.5 illustrates.

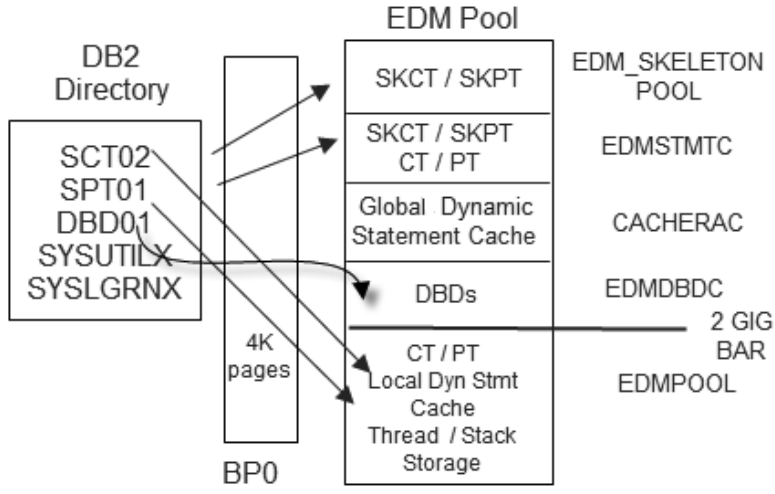

Figure 2.5: EDM pool

SKCTs and SKPTs are now above the 2 GB bar. The static SQL sections (CT/PT) are split between above and below the 2 GB bar. Distributed applications and some storage acquired for dynamic SQL statement execution (which includes the parse trees and a portion of runtime blocks) has also moved above the 2 GB bar. Tables, object blocks, and mapping blocks that are associated with the EDM fixed storage pools have moved above the bar. Fixed storage pools contain the larger object

identifying control blocks and the small mapping control blocks that map each block of EDM storage above or below the bar.

EDM Statement Cache

The EDM global dynamic (SQL) statement cache resides above the 2 GB bar in the EDM pool. It consists of a pool of pages used for prepared (**PREPARE**) SQL statements in the **DBM1** address space and contains either the prepared SQL statement (called a short prepare) or a full prepare.

The global dynamic statement cache includes a statement cache table, **DSN_STATEMENT_CACHE_TABLE**, that is used by the **EXPLAIN STMTCACHE ALL** statement that was introduced in DB2 8 for z/OS. The contents of the cache table are nearly identical to IFCID 316 and 317 statistics. The **EXPLAIN STMTCACHE ALL** statement extracts all the statements from the global cache and inserts one row into the table for each entry. You can also extract a single statement from the global dynamic statement cache by using the **EXPLAIN STMTCACHE STMT_ID** statement.

DB2 9 provides four caching options:

- Local dynamic statement caching
- Global dynamic statement caching
- Full caching
- No caching

Caching of dynamic SQL statements and statement text typically reduces the **PREPARE** operations required for those statements. You control the local and dynamic cache using DSNZPARMs, **BIND** options, and application constructs.

To enable local statement caching, use the **KEEPDYNAMIC(YES)** option of the **BIND** command, which will keep a copy of the prepared statement and the statement string. The **MAXKEEPD** DSNZPARM controls the maximum number of prepared statements to keep past a commit point. The statement text is always kept. The implicit **PREPARE** eliminates the need for an application to execute multiple **PREPARE**s for the same statement.

To enable global statement caching, set the **CACHEDYN** DSNZPARM to **YES**. Global statement caching permits the reuse of prepared statements across units of work within the program and across program executions. The prepared statements, called skeleton dynamic statements (SKDSs), are cached in the global dynamic statement cache. These statements can be copied into local storage when possible; such statements, known as short prepare statements, are dynamic statements in the global cache. You can monitor the global statement cache using IFCID 316 and IFCID 317.

Full caching is a combination of local and global statement caching that provides the ability to avoid prepare operations completely (a feature known as prepare avoidance). Statements kept in local thread storage are not invalidated across commits. To enable full caching, set **CACHEDYN=YES**, **KEEPDYNAMIC(YES)**, and **MAXKEEPD>0**.

SCT02: Skeleton Cursor Table

When you bind a plan, DB2 creates a structure called a *skeleton cursor table* in the **SCT02** table space. The SKCT contains the internal form of the SQL statements that are in your application program. As a plan is executed, DB2 uses this information to access the data it needs. In DB2 9, the SKCTs move above the 2 GB bar in the EDM pool.

SKCTs are stored as a sequence of SKCT sections because a single SKCT can be longer than the maximum record length supported by DB2. These sections begin with a skeleton cursor table parent record (SCTR), which contains as much of the SKCT section as the record can fit. The SKCT section is stored in this record if it fits; if it does not fit, it is stored in one or more SCTRs. Each SCTR is identified by a unique section/sequence number.

SPT01: Skeleton Package Table

A *skeleton package table* is created when you **BIND** a package. It contains the internal form of the SQL statements in your application program. In DB2 9, these structures move above the 2 GB bar in the EDM pool. When initially loaded to execute, they are copied to buffer pool **BP0** and then into the EDM pool in sections.

These sections can include a header and other sections that are the SQL statements to execute.

Parameter **EDM_SKELETON_POOL** (field **EDM SKELTON POOL SIZE** on installation panel **DSNTIPC**) determines the minimum size of the EDM skeleton pool.

Table space **SPT01** is used to store SKPTs. These tables store the access paths to DB2 data. DB2 uses this information to access the data it needs when a package is executed.

Because a single SKPT can be longer than the maximum DB2-supported record length, SKPTs are stored as a sequence of SKPT sections. The skeleton package table parent record (SPTR) contains as much of the SKPT section as the record can fit. It is possible that the entire SKPT section can be stored in this single record. If the record cannot hold the entire SKPT section, the rest of the SKPT is stored in one or more SPTRs records. Each SPTR is identified by a unique section or sequence number.

APAR PK80375 enables compression of the **SPT01** table space. As you use plan stability and increase the space requirements, this capability will help you manage storage consumption. Be aware that the 64 GB limit can be a constraint if you enable plan stability.

DBD01: Database Descriptors

Database descriptors uniquely represent databases within DB2, including user databases, **DSNDB06**, and **DSNDB07**. A one-to-one relationship exists between the database structures and a DBD. The descriptors of objects such as files, page sets, fan sets, and records are contained in each DBD format.

The DBD contains object identifiers (OBIDs) that define the objects within the database to DB2. Each **CREATE** statement for databases, table spaces, and tables is given a unique internal number, or OBID.

In DB2 9, the DBDs are above the 2 GB bar. This area is used for the definitions of objects located in user databases, starting with the database, table space, tables, and so on. The OBIDs are strung together in a hierarchical list of dependencies that

identifies each database. For each database, you will have a construct to describe the objects based on the OBIDs. The CLIST calculates the DBD cache size; the default size is 11,700 KB.

Services are available to maintain the DBDs as well as to access these internal objects within DB2. These services provide the following functions and services:

- Retrieve, insert, replace, and delete the internal objects in **DSNDB01**
- Retrieve and update data stored in the system directory
- Maintain DBDs
- Maintain and retrieve SKCT blocks and SKPT blocks
- DM data manipulation services
- DM database descriptor management
- Many other DM services for access to DBDs
- Services used by the service controller subcomponent

DBDs do not require contiguous storage, but they do require 34 KB pieces. The data manager environmental descriptor manager function provides services, such as access and management, to the internal objects. DBDs are stored in the **DBD01** page set in chained records. A parent, called a DBDR, is connected via a link to a child records, called DBDSs.

DBDs can have many sections. A DBD is a contiguous block of information representing a database that contains OBDs for various DM objects within the database. Each object's object identifier represents the name of the object that was defined with Data Definition Language (DDL) statements.

To reclaim the storage in the DBD, you can use the **MODIFY RECOVERY** utility.

Fact Summary About Pages in the EDM Pool

EDM Relational Data Server (RDS) pool (CT and PT sections reside above and below the bar):

- Each executing user application must have a CT or a PT with access paths to execute.
- Each user has a working copy (CT or PT) of the SKCT or SKPT that is executing. The CT or PT does not have to have contiguous storage. These are sections of copies of the SKCT or SKPT that are chained together.
- CT pages (cursor tables in use):
  - » Stored in sections
  - » Contain a working copy of an SKCT
- PT pages (package table sections in use):
  - » Can already be in the EDM pool, no I/O
  - » Stored in sections
  - » Contain a working copy of an SKPT
- The authorization cache is in the RDS pool.

EDM skeleton pool (both completely above 2 GB):

- The EDM skeleton pool is new in DB2 9. It is defined in the DSNZPARMs as **EDM_SKELETON_POOL**. This parameter, which specifies the size of the EDM skeleton pool, defaults to 5,120 KB in size at install time.
- The pool space is not automatically increased or decreased. To override the current value, use the **SET SYSPARM** command.
- The EDM skeleton pool is used for skeleton cursor tables (SKCTs) and skeleton package tables (SKPTs). It also has control blocks used for fixed pools, mapping blocks, object blocks and hash tables.
- SKCT pages (skeleton cursor table):
  - » Shared by users
  - » Created in **SCT02** when you bind a plan

  - » Describes the structure of the SQL statements in application plans and consists of executable SQL and RDS control structures related to access paths
    - ■ As each SQL call is made, the required sections are loaded into the pool. They remain there until a **BIND REPLACE**, **FREE**, or **REBIND** command is executed or until least-recently-used (LRU) replacement takes place.
  - » Stored in a sequence of SKCT sections that are loaded for execution
    - ■ The first section, called an SCTR, is a parent record. If additional space is required to store the sections, one or more SCTRs, each identified by a unique section or sequence number, are chained.
- SKPT pages (skeleton package table):
  - » Created in **SPT01** when you **BIND** a package
  - » Applies to packages, which are shared among the plans that reference them
  - » Describes a collection of packages grouped by a **BIND PACKAGE** with a collection ID
  - » Contains a directory, header, and one or more sections of SQL

EDM database descriptor pool (above the bar) in DBD01:

- The EDM database descriptor pool contains the database descriptor (DBD) pages.
- A DBD is created whenever the **CREATE DATABASE** statement is executed.
- Database structures are cached here.
- Each DBD describes a database and all its objects and contains access information.
- DB2 uses an algorithm to locate an object descriptor (OBD) within a DBD. There are two descriptions for a database. One is in the catalog, and the other is the internal representation of the data in the DBD in the directory. This is a one-to-one relationship.

- The OBD has subdescriptors of all the objects contained in the database, including table spaces, tables, indexes, constraints, and relationships with the LOB columns.
- Internally, there are six types of OBDs: file, page set, record type, fan set, check constraint, and auxiliary relationship. The OBDs contain information about how the records are organized, stored, and accessed and represent the internal representation of the objects.
- You can find object identifiers in the DB2 catalog under the object table definitions. Columns in them will have **DBID**, **PSID**, **OBID**, and **ISOBID** for database, table spaces, tables, and indexes.
- A DBD starts as a single block. As it grows due to objects being defined with **CREATE**, additional blocks can be added. It is read from DASD as a chain of blocks into storage.
- If you use the **DROP** statement to remove an object from the database, the object identifier is not automatically removed from the DBD hash chain.

SYSUTILX

For every utility job running in DB2, a row is placed in table space **SYSUTILX** in the DB2 directory. This row is used if you have to restart the utility. When the utility finishes running, the row is removed. Information for a copy of **SYSUTILX** is located in the log.

Keep in mind the following points related to **SYSUTILX**:

- You cannot **REORG** the **SYSUTILX** table space.
- The **SYSUTILX** table is a dependent of the **SYSUTIL** table.
- Rows in **SYSUTILX** are uniquely identified by a utility identifier and sequence number.
- When information in the parent record exceeds the record size of table **SYSUTIL**, a record is created in the **SYSUTILX** table.

The **SYSUTIL** table stores the status of DB2 utilities that are stopped or started. Each record in the table is uniquely identified by a utility identifier. Each row contains the information for one utility execution step. When a utility finishes running, the corresponding entries in the **SYSUTIL** table are deleted.

SYSLGRNX

The DB2 **MODIFY** utility checks user authorization, issues appropriate messages, deletes specified records, and updates the **SYSIBM.SYSLGRNX** table in the DB2 directory. **SYSLGRNX** is a log range table space that tracks the opening and closing of table spaces, partitions, and indexes. DB2 tracks the information by the relative byte address (RBA) that is written in the log after the most recent copy, reducing the amount of time required to recover an object.

The directory table spaces and associated indexes do not have entries in **SYSIBM.SYSLGRNX**, even if they were defined with **COPY YES**.

The size of the directory depends on the number of user databases, packages, plans, and tables in DB2. At installation, the **DSNTINST** CLIST calculates the sizes of the EDM pools (above and below the 2 GB bar), the EDM statement cache, the EDM database descriptor cache, and the EDM skeleton pool. Look at your calculated sizes on the **DSNTIPC** installation panel.

When a table space is open for updates, or when an index is defined with **COPY YES**, DB2 records the recovery log range times in **SYSLGRNX**. This information provides an efficient way for DB2 to access the appropriate log records for recovery, without having to scan every record in the recovery log for a specific table.

Catalog/Directory Access Methods

The DB2 directory uses links that exist between the DBD parent record for a given DBD and the database child record records.

Another access path to data in the DB2 catalog or directory is called a link. A link consists of a record identifier and a hash, which are, respectively, a page number

and an offset to an anchor point. A link is a parent/child relationship between two tables or records. There are links that exist between rows of tables in the DB2 catalog.

Hashing Algorithms

The method of hashing is used to access data only in the DB2 directory and in the **DBD01** page set. A database descriptor is created when you issue the **CREATE DATABASE** statement. The DB2 data manager allocates 4 KB of storage initially and formats a DBD.

- The hash key for **DBD01** is the database identifier (DBID).
- Index **DSNSCT02** is used to access data in the **SCT02** DB2 directory page set.
- Indexes **DSNSPT01** and **DSNSPT02** are used to access data in the **SPT01** page set.
- Table space **SYSUTILX** uses indexes **DSNLUX01** and **DSNLUX02** to access data.
- Table space **SYSLGRNX** uses indexes **DSNLLX01** and **DSNLLX02** to access data.

DSNZPARMs for the EDM Pool

The following table provides a simple key to relate the various terms involved with the EDM pool to the subsystem parameters associated with the pools in the EDM pool:

Term	Related DSNZPARM
RDS pool	**EDMPOOL**
DBD pool	**EDMDBDC**
Statement pool	**EDMSTMTC**
Skeleton pool	**EDM_SKELETON_POOL**

You also have subsystem parameter **EDMBFIT**; specify **NO** (the default) to optimize performance or **YES** to optimize storage utilization. Parameter **EDMBFIT** controls

how space is freed for EDM pools that are greater than 40 MB. **YES** specifies a better-fit algorithm to handle **EDMPOOL** full conditions for these large pool sizes. **NO** specifies a first-fit algorithm, usually for smaller **EDMPOOL**s, and is set when class latch 24 contention starts to exceed 500 contentions per second. Latches are used to serialize access to many memory resources. The object of a latch is a page in DB2. Latch class 24 is used for the EDM pool least-recently-used (LRU) chain and the buffer manager page unlatch and prefetch. You can find the description of each class in the **DSNDQVLS** macro in **SDSNMACS**.

Storage Above and Below the Bar

Virtual Storage Constraint Relief (VSCR) below the bar in the **DBM1** address space is achieved by moving the plan and package skeletons completely above the bar into their separate EDM pools. In DB2 9, the EDM fixed storage pools (FSPs) are moved above the bar; they include all the hash tables, mapping blocks, and object blocks. This area contains small mapping control blocks that map each block of EDM storage both above and below the bar and the larger object identifying control blocks. By moving these control blocks above the bar, scaling of the number of EDM objects can occur without affecting below-the-bar storage usage.

The last control block in DB2 9 that is associated with each dynamic SQL statement is moved above the bar. The EDM statement cache pool can expand above the bar without increase to any statement cache control blocks below the bar.

Dynamic SQL statements now have a larger portion split above the bar than for static SQL in DB2 9 plans or packages. This split has to do with the extra storage required in dynamic statement storage for **DESCRIBE** column information and **PREPARE** options. These options, of course, do not exist as part of the statement storage for static statements. The **DESCRIBE** and **PREPARE** storage is now all above the bar. Individual statement storage is generally larger for dynamic SQL than for static SQL.

The moving of some short-term control blocks above the bar has reduced peak storage usage for **BIND** dynamic **PREPARE**. This short-term storage (called parse tree storage), which is held during a full prepare of SQL statements, has decreased

significantly. Some estimates of peak storage usage for full prepares have been reduced by 10 percent.

Other items that have moved above the bar are some tracing storage and mini-plan storage. Stack storage, system thread storage, and user thread storage usage has remained about the same, but some of the savings may be offset by increases in other areas.

To reduce the EDM latch class 24 serializations, the below-the-bar EDM pool cache now has no LRU objects. You must size the below-the-bar EDM pool for peak usage. No automatic expansion of this storage pool occurs, and the possibility of an EDM pool full condition exists. You modify this setting via a DSNZPARM change and activate the change using the **SET SYSPARM LOAD** statement. Parameter **EDMPOOL** in **DSN6SPRM** is the value associated with below-the-bar 2 GB bar storage.

You should carefully size the below-the-bar EDM pool to accommodate peak usage plus a cushion for fragmentation. Best practices dictate that the size of the EDM pool storage below the bar be between 110 percent and 130 percent of peak usage. This sizing will take into account any pool fragmentation and the need to allocate contiguous storage for any section.

If you do experience EDM pool full situations, IFCID 31 will identify the object type, size, and requestor. If this IFCID is active, it produces trace records only when the EDM pool full situation occurs.

In addition, IFCID 02 collects EDM statistics. Take a look at member **DSNWMSGS** in library **SDSNSAMP** for field definitions of this and other trace records. **DSNDQISE** and **DSNDQWS1** contain the field definitions for this trace record.

It is a good idea to add 10 percent to 30 percent to your initial estimate, just to have a cushion of storage in case you need it. The number of dynamic statements that are active below the bar can cause this critical shortage. The only way to increase this area is by using the **SET SYSPARM** statement.

A new DB2 9 subsystem parameter, **CACHEDYN_FREELOCAL** in macro **DSN6SPRM**, can free cached dynamic statements to relieve the shortage

below the bar. This setting applies only if you have activated the **BIND** option **KEEPDYNAMIC(YES)**.

The **CACHEDYN_FREELOCAL** parameter is changeable online via the **SET SYSPARM RELOAD** statement. Changing the value of this parameter changes the triggering level at which DB2 9 tries to free local SQL cache storage below the bar.

The values **0** (zero, the default in DB2 8) through **3** are available to represent the trigger levels DB2 uses to reclaim local SQL cache below-the-bar storage. Specifying **1** (one), the default in DB2 9, causes DB2 to free some cached dynamic statements if the storage usage is high. If you specify **0**, no cached dynamic statements are freed. The built-in **DBM1** storage monitor writes information to the console to let you know how much storage is currently consumed as limits are met. The IFCID 225 trace record also collects information about the cache storage.

If you size the EDM storage pools too small, you could see increased I/O on the **DBD01**, **SPT01**, and **SCT02** directories, along with increased response times and a smaller number of threads running concurrently because of lack of storage.

To achieve the above- and below-the-bar storage for PTs and CTs, you must rebind your plans and packages in DB2 9 to move the relevant sections of these objects to 64-bit storage addressing.

Storage Pools

Dynamic SQL execution storage can be extensive and can be one of the largest factors driving **DBM1** below-the-bar virtual storage constraints. Two new IFCIDs, 217 and 225, collect information about storage above and below the bar. Statistics about the storage manager pools are produced at the interval specified by parameter **STATIME** in the DSNZPARMs.

DB2 8 introduced 30 global variable-length storage pools that contain the dynamic SQL statement storage used for statement execution in below-the-bar storage. In DB2 9, the individual statement storage is split between above-the-bar and below-the-bar portions. We use these values instead of pool totals to understand how peaks in the cached SQL statements drive the total storage use in **DBM1**.

To collect IFCID 217 and IFCID 225 records, you must activate the DB2 **STATIME** interval. This parameter specifies the number of minutes between statistics collections. The default interval for **STATIME** is now five minutes (**5**).

IFCID 225 provides a summary report of the storage being used in the **DBM1** address space, and IFCID 217 produces a detail report of activity. IFCID 225 records collect data on the first failure.

IFCID 217 produces SMF type 102 storage detail data records. IFCID 225 is added to **STATISTICS** trace class 1, and its statistics are written as SMF type 100 with a subtype of 4 records. DB2 keeps an internal cache of IFCID 225 records so that storage changes over time can be evaluated during dump analysis. No separate instrumentation fields for the accumulated size of the 30 statement cache pools are kept.

You can activate collection of these statistics using the following command:

```
-START TRACE(GLOBAL) IFCID(217)
```

MAXKEEPD

You use the **MAXKEEPD** subsystem parameter to limit the number of dynamic statements that are held in the cache after a commit point. These locally cached statements can consume large quantities of **DBM1** virtual storage. This setting applies to full caching only. If a large number of **PREPARE** operations are occurring on a system and application logic is not committing frequently, **MAXKEEPD** can be exceeded before the release of the cached statements can occur.

Stored Procedures

Native and external SQL procedures can be called in DB2 9 in new-function mode.

- The PL/SQL native logic for the SQL is stored in the DB2 directory.
- Native SQL procedures are executed entirely in the DB2 **DBM1** engine and should outperform external SQL procedures.
- Native SQL procedures can be created starting in DB2 9 NFM:

- » You use the **CREATE PROCEDURE** statement to create a native SQL procedure.
- » The **FENCED** or **EXTERNAL** keyword cannot be used.
- » Above-the-bar storage is used for native stored procedures.

External SQL procedures will continue to work in DB2 9 either in conversion mode or NFM.

- You use the **CREATE PROCEDURE** statement to create an external SQL procedure.
- The **FENCED** or **EXTERNAL** keyword is required.

External SQL procedures from prior releases are executed in the WLM environment. **DSNTPSMP** is an SQL procedure processor you can use to prepare the external procedure. **DSNTPSMP** requires DB2 for z/OS REXX language support and WLM stored procedure address space (SPAS) with the setting **NUMTCB=1**. This procedure can be invoked only by an SQL call from an application program or from IBM DB2 Data Studio Developer.

DSN1CHKR

If you suspect that data inconsistencies exist in the link or hash chains in the pages of the DB2 catalog or directory, you can run the **DSN1CHKR** stand-alone utility to validate one or more data pages in question in the DB2 catalog or directory.

BINDNV DSNZPARM

The new DSNZPARM **BINDNV** controls the authority needed to add a new package in DB2. When **BINDNV** is set to **BIND**, the creation of a new package requires only the **BIND** privilege. With this privilege, if a user has **BIND** authority on a package, he or she can create a new **VERSION** of a package. If you set the **BINDNV** parameter to **BINDADD**, only users with **BINDADD** authorization can add a new **VERSION**.

RID Pools

The row identifier (RID) pool is an area of local storage reserved for sort processing of record identifiers from an index or indexes (multiple index processing). A RID is made up of a page number and a row identifier that defines the row in the data page of the table for that key value. In general, there are multiple **LEAF PAGE**s in the index structure containing multiple entries of RIDs for each row in a table.

RIDs are stored in leaf index pages and identify the records stored in data pages. A RID can be a four- or five-byte page number followed by a one-byte page ID map. Five-byte RIDs are associated with large table spaces that are defined with a **DSSIZE** that is 4 GB or larger or with LOBs (auxiliary tables). Four-byte RIDs are associated with all other table space definitions; these identifiers consist of a three-byte page number and a one-byte ID pointer.

Each RID describes a page number and an ID that points to a row in the table space. Figure 2.6 shows the relationship between the RID and the row of data in the table.

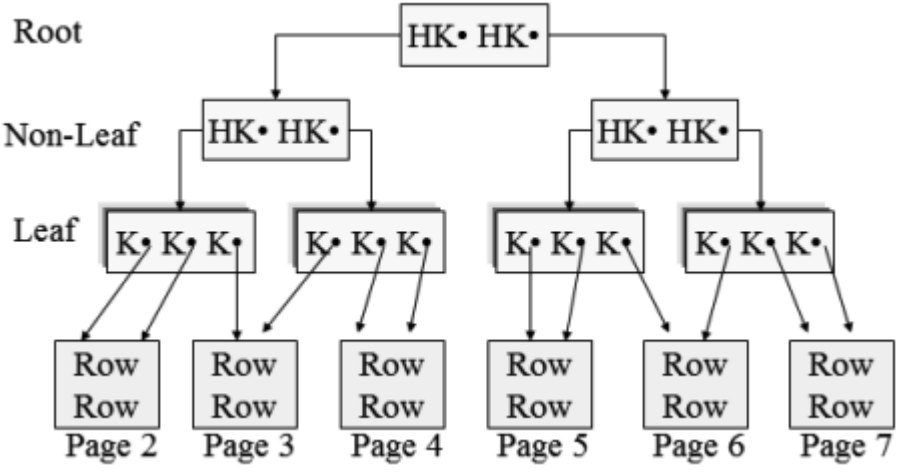

Figure 2.6: Relationship between RID and data row

The **MAXRBLK** subsystem parameter in macro **DSN6SPRM** defines the number of RID blocks allocated for the RID pool storage. **MAXRBLK** is stored internally as a number that is divided by 32 because DB2 allocates RID blocks in 32 KB chunks.

The **DSNTIPC** migration panel displays the **RID POOL SIZE** field, whose default is 8,000 KB. Acceptable values are 0 KB or a setting between 128 KB and 10,000,000 KB. A single RID list maximum size would be approximately 26 million RIDs.

During processing, the index RIDs are selected and sorted in the RID pool according to page number within RID. This list of RIDs, in page number order, provides access to the data pages within the table space in order. DB2 uses this sorted list to access the table rows by reading 32 pages per I/O operation and trying to read ahead one block of 32 pages as well.

The RID pool is split into two parts. The area below the 2 GB bar (approximately 25 percent) stores the RID maps, and the area above the 2 GB bar (approximately 75 percent) contains the RID list. No space is actually allocated in the pool until RID storage space is needed. At that point, the space is allocated in 32 KB RID blocks as needed. Each new agent requesting RID pool usage is given two 32 KB blocks, one for the RID list and one for the RID map, for a total of 64 KB per agent.

The RID pool is used in hybrid joins, multiple index processing, list prefetching, and the enforcement of unique keys during row updates. If the RID pool runs out of space, the SQL statement reverts to sequential processing at the point of failure. The more common RID pool failures can also be caused by not running **RUNSTATS**. When enabling list prefetch, the optimizer sets a threshold of 25 percent of the number of rows in the table. If the RIDs exceed this number, processing reverts to sequential processing.

RID sorts occur totally in memory. You can track RID list processing with IFCID 003 and IFCID 002. You can also determine whether list prefetch has ended if more than 25 percent of the rows in table are accessed with IFCID 125 in a performance trace.

Optimistic locking also uses the RIDs and a change token to ensure integrity of the data being read.

DB2 9 uses dynamic prefetch more frequently, and the formula for tracking pages now tracks the cluster ratio with prefetch quantity and buffer pool size

considerations. The algorithm has been enhanced to count each RID in the **CLUSTERRATIO** formula rather than just the distinct key values. In addition, a new column in **SYSINDEXES** called **DATAREPEATFACTOR** tracks whether rows are found on a page other than the current one. This information, along with the **CLUSTERRATIO**, can distinguish between dense and sequential rows.

The DB2 9 enhancement of star join group or pair-wise join depends on the resources of the RID pool. Using this join method requires a one-column index on a fact table that supports each dimension table join column. Don't forget that list prefetch is disabled on a **VOLATILE** table. Also, **OPTIMIZE FOR 1 ROW** will avoid RID processing. **RID** is also a new built-in function that returns the RID of a row.

Although there is no RID list in the **EXPLAIN** tables, a column **PREFETCH** type of **L** indicates that RIDs are being processed to read data pages.

Physical Rows and Pages for Table Spaces

The physical structure of a table space consists of 4 KB pages. Each table space type has a specific structure. Figure 2.7 shows the basic format of the table space pages. In DB2 8, the table space structure added a system page to support online schema versioning.

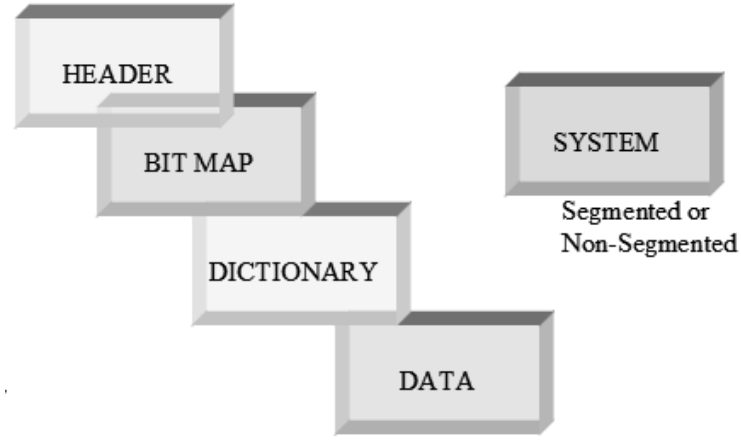

Figure 2.7: Table space pages

Header Page

The first page in any table space type or index is a header page. All header pages have the format shown in Figure 2.8.

Figure 2.8: Header page format

Bit Maps or Space Maps

There are actually six space maps, or bit map formats, but we will look at only two here. The two types of bit maps inside the structure of a table space are one map type for segmented table spaces and one for nonsegmented table spaces. The bit map page distinguishes which data pages have free space to accommodate additional row inserts. Each space or bit map covers a specific range of pages, based on the page size and whether the table space is a segmented or a non-segmented table space.

Figure 2.9 depicts the structure of the segmented bit map.

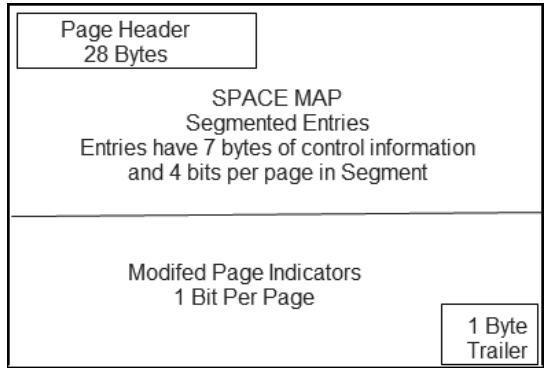

Figure 2.9: Segmented bit map

In the segmented bit map, each four-bit entry indicates the amount of free space remaining in the corresponding data page. Table 2.3 describes the binary values in the four-bit entries.

Table 2.3: Binary values for segmented bit map

Binary entry	Meaning
B'0000'	Page is empty — not formatted.
B'0001'	Page is empty — caused by a mass delete statement.
B'0010'	Page is empty — typically caused by normal delete functions.
B'0010'	Page has free space greater than or equal to the maximum record size.
B'0100–B'1010'"	These bit settings are for variable-length records.
B'1011'	Space is less than, greater than, or equal to minimum size of record.
B'1111'	Page is full.

Space or bit maps associated with nonsegmented table spaces have a different layout. The nonsegmented, or partitioned, space map (Figure 2.10) covers a fixed number of pages. Each two-bit entry corresponds to a page within the range that the space map covers.

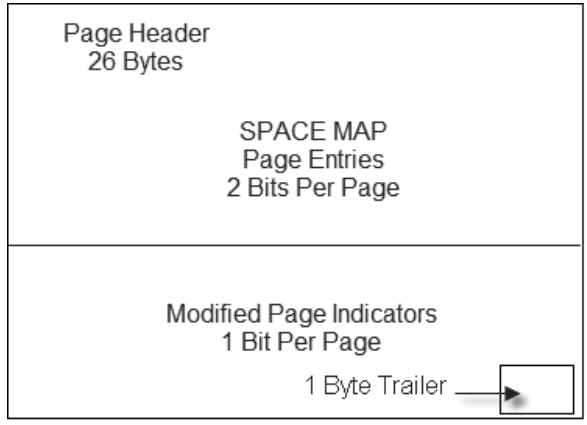

Figure 2.10: Nonsegmented bit map

So, as an example, the number of pages per space map would be as follows:

- A 4 KB page size for any type or partition would cover 10,764 pages.
- An 8 KB page size for any type or partition would cover 21,680 pages.

- A 16 KB page size for any type or partition would cover 43,528 pages.
- A 32 KB page size and number of pages would depend on the number of partitions and whether a table space is associated with a member cluster.

You can find more information about sizing in the *DB2 Diagnosis Guide and Reference* (LY37-3201).

The two bits describe how much free space is left in the corresponding data page. Table 2.4 lists the four possible values.

Table 2.4: Binary values for nonsegmented bit map

Binary value	Meaning
B'00'	Free space is greater or equal to maximum record size.
B'01'	Free space is greater or equal to average record size.
B'10'	Free space is greater or equal to the minimum record size.
B'11'	Free space is less than the minimum — full page.

The modified page indicators in the second part of the nonsegmented or partitioned space map indicate whether the page has changed since the last time the **COPY** utility was run. Interestingly enough, this information is available only if the table space was created or altered with the **TRACKMOD YES** option. Whenever a full image **COPY** is run or an incremental copy is made, the bits are reset to **0**.

Large objects have two types of space maps, and index spaces have a space map entries page, but in an index space map there are two differences: First, there is one bit to each corresponding page, and second, there are no modified page indicators.

Dictionary Pages

If you have compression set on in your table space, you will have dictionary pages. The dictionary pages will come after the first space map and before the data pages. There can be up to sixteen 4 KB pages in your table space structure to form the dictionary pages. These dictionary pages are created with the **LOAD** or **REORG** utility, using the proper settings, to build new dictionary pages or to keep the already-built dictionary pages built for compression. When the table space is accessed, the compression dictionaries are stored above the 2 GB bar.

The SQL statement to turn on compression is **CREATE** (or **ALTER**) **TABLESPACE ... COMPRESS YES**.

Remember that you can run the DB2 stand-alone utility **DSN1COMP** to estimate the amount of compression. The unit of compression in ESA is a table row. This type of compression is not used in work files, the catalog, the directory, LOB table spaces, or table indexes.

Data Pages

The data pages (Figure 2.11) all start with one page header followed by rows with a row header and then the row data. One byte in the row header supports the "version" of the row.

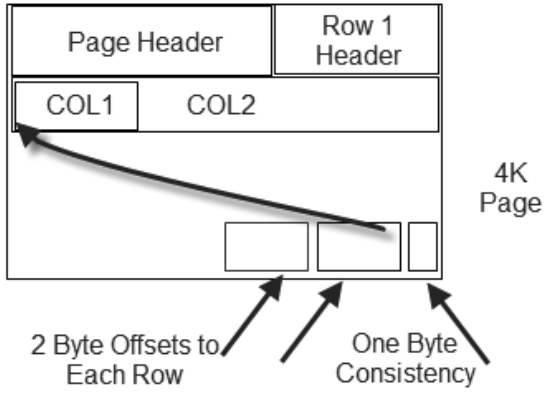

Figure 2.11: Data page

System Page

A system page helps with the management of online **ALTER** schema changes or versions of the object. The system page tracks format changes to the rows in a table space. System pages are inserted in the table space as you make versioning changes. The **COPY** utility gives you a choice of whether to copy these pages or not; the default is **YES**.

SYSIBM.SYSOBDS in **DSNDB06** contains one row for each table space or index that can be recovered to an image copy that was made before the first version

was generated (version 0, unaltered objects). The **SYSODBS** table contains the information listed in Table 2.5.

Table 2.5: SYSIBM.SYSODBS table space

Column name	Definition
CREATOR	Authorization ID
NAME	Object name
DBID	Internal database ID
PSID	Internal table space or index space ID
OBID	Internal table/index ID
OBDTYPE	Object type
VERSION	Version when altered
CREATDTS	Timestamp
OBD (varchar 30000)	OBDREC
IBMREQD	IBM flag

If you look in **SYSIBM.SYSTABLESPACE**, you can also find entries for the DBID, OBID, and PSID for identification of the internal definition of your databases, tables, and index spaces.

Base and Clone Tables

DB2 9 provides support for fast replacement of tables. You generate a copy of the base table by building a *clone*. You can insert or load data into the clone table and use the SQL **EXCHANGE** statement to perform a fast replacement of the original data. Interestingly, the table space name is the same for both the base and the clone table. The **RUNSTATS** utility does not collect statistics on clone tables.

SYSIBM.SYSCOPY will reflect the **CREATE** of a clone with the new **ICTYPE** of **C**. Revisions to columns in the DB2 catalog include new column descriptions and values and changed data types, column lengths, or both.

In DB2 9, a new column of interest is the **INSTANCE** column, which always has a value of **1** or **2**. This column has been added to some DB2 catalog tables to reflect the base versus the clone data sets.

Universal Table Spaces

DB2 9 features new table spaces to provide key functions of both segmented and partitioned table spaces. This combination of features is called a universal table space (UTS). There are two types of UTSs:

- *Partition by growth (PBG)*: These table spaces allow segmented tables to be partitioned as they grow without the need for key ranges.
- *Partition by range (PBR)*: These table spaces are just like the existing partition table spaces but are segmented.

Partition-by-Growth Table Spaces

Partition-by-growth is now the default table space type in DB2 9. A partition-by-growth table space is very useful for table spaces whose tables lack a suitable partitioning key but are expected to exceed the 64 GB limit. You do not define a partitioning key with PBG. Each PBG table space holds a single table.

If you plan to make changes to **DSSIZE** and **SEGSIZE**, you will need to do a **DROP** to change your table space; there is no **ALTER** option. A partition-by-growth table space can grow up to 128 TB; the **MAXPARTITIONS**, **DSSIZE**, and page size values determine the maximum size.

The following restrictions apply when using PBG:

- The **LOAD** utility does not allow the **LOAD PART** option for partition-by-growth table spaces.
- Only non-partitioning indexes are allowed with PBG, and the table spaces must be storage-group–controlled.
- The **REORG** utility does not delete the existing partitions, even if they are no longer needed.
- A PBG table space is incompatible with the **ADD PARTITION** and **ROTATE PARTITION** options of the **ALTER TABLE** statement.

Partition-by-Range Table Spaces

Segmented table spaces that are approaching the 64 GB limit can be converted to partition-by-range universal table spaces that support up to 128 TB. You can choose your **DSSIZE** parameter value. If you specify **LARGE**, the size is limited to 64 partitions. To move above this range, you will need **DSSIZE**. Parameter **DSSIZE**, the page size, and the number of partitions determine the maximum size of the table.

PBRs offer better space management and improved delete performance due to the segmentation bit map structure that was originally used in segmented table spaces. To specify a PBR, you use the parameters **NUMPARTS** and **SEGSIZE** in one **CREATE TABLESPACE** statement. These types of table spaces still require a partitioning column.

Index Structure

Tables support multiple indexes in DB2, and each index has a format consisting of a B-tree structure. The data row diagram that we looked at previously (reproduced in Figure 2.12) depicts a clustering index.

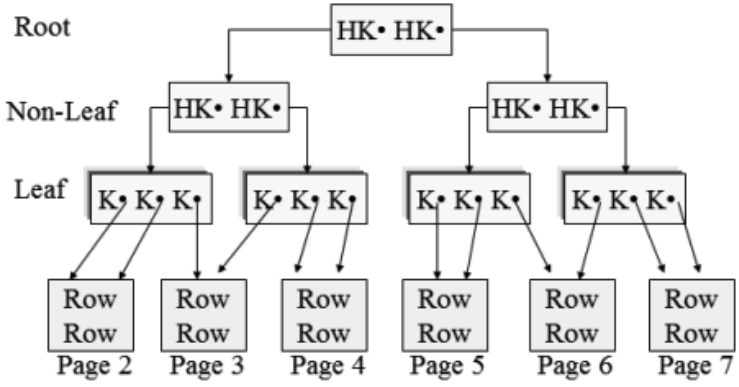

Figure 2.12: Index structure

In this clustering index structure, each block represents a 4 KB page. The first, or highest, level is the root, followed by multiple non-leaf pages, and then the leaf pages, which contain the key and row ID that point to the data in the table.

The contents of the root and non-leaf pages are the highest key and a pointer to the next level. The contents of the leaf pages are the full key and the address location of the associated row in the table space (RID).

DB2 supports clustering indexes, unique indexes, non-unique indexes, partitioning indexes, partitioned indexes, and no indexes at all on tables.

BRF and RRF Formats and Logging

Up to now, table row formats in DB2 have been what we call *basic row format (BRF)*. DB2 9 introduces a new row format, called *reordered row format (RRF)*, for user data. In DB2 9, the catalog and directory tables remain in basic row format.

If you are in NFM and your table space does not use an **EDITPROC** or **VALIDPROC** routine and you use either the **REORG** or **LOAD REPLACE** utility, the row format will be changed from BRF to RRF.

During migration from DB2 8 to DB2 9, DB2 by default ignores the **KEEPDICTIONARY** parameter of the **LOAD** and **REORG** utilities when converting tables from BRF to RRF. If you want the **KEEPDICTIONARY** parameter to be honored, you must set the subsystem parameter **HONOR_KEEPDICTIONARY** to **YES**.

In RRF, if a table contains any varying-length columns (with or without nulls), all fixed-length columns are placed at the beginning of the row, followed by the offsets to the varying-length columns, followed by the values of the varying-length columns.

ROWID and indicator columns are treated like varying-length columns. Row IDs are **VARCHAR(17)**. A LOB indicator column is **VARCHAR(4)**, and an XML indicator column is **VARCHAR(6)**. The LOB indicator column is stored in a base table in place of a LOB or XML column and indicates whether the LOB or XML value for the column is null or zero in length.

You use the **DSN1COPY** stand-alone utility to populate one table space to another table space. Be aware that the row formats of the two table spaces must match. If they do not, the utility will fail, or unpredictable results may cause integrity

problems. Use the following SQL statement to check the row format of your table space before executing **DSN1COPY**:

```
SELECT DBNAME, TSNAME, PARTITION, FORMAT FROM SYSIBM.SYSTABLEPART ...
```

If the **FORMAT** column in the **SYSTABLEPART** catalog table has a value of **R**, the table space or partition is in RRF. If the **FORMAT** column contains a blank value, the table space or partition is in BRF.

Figure 2.13 illustrates the two row formats in DB2.

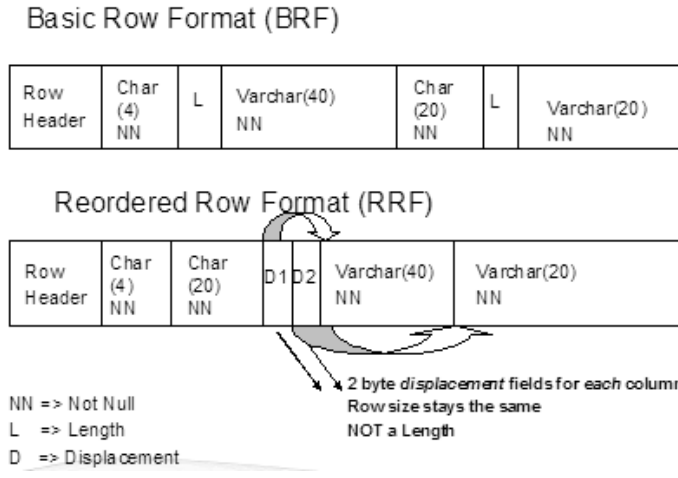

Figure 2.13: Row formats

Index Compression

DB2 9 provides index compression, but this compression does not use a dictionary. Index compression occurs at the page level via software compression, but only for the leaf level in the index structure. The pages are compressed at write time and are decompressed during reading. This technique to compress the index keys and RIDs is called *prefix compression*.

The index keys are composed of two parts: the key of the index and associated page number and an offset to the data. The key map, associated with each entry

on the leaf, is not compressed. This information is actually rebuilt at the time of decompression.

To work with compression, DB2 transparently manages an I/O work area that is separate from the buffer pool. DB2's deferred write engine asynchronously compresses a leaf page from a buffer into an I/O work area and then writes the page to disk. Compression begins immediately on the contents of the index if you use the following statement:

```
ALTER INDEX COMPRESSION YES
```

This compression cannot be activated at the partition level; it is for the whole index. The compression occurs, not key by key, but by the leaf page level. There is no ability to randomly find a key in the index and decompress that key. A random probe of the index requires a B-tree search. There are no compression dictionaries; this is compression done on-the-fly at the page level.

The index page size must be larger than 4 KB, and the index data is compressed "down" to 4 KB size on disk.

You can have 32 KB, 16 KB, or 8 KB index page sizes for indexes in NFM of DB2 9. Turning on compression can save you up to eight times the disk space and reduce the frequency of index page splits. IBM's estimates for compression savings run from 25 percent to 75 percent disk savings. To see whether you should compress your indexes, run **DSN1COMP**; this utility has been enhanced in DB2 9 to provide information about possible page sizes.

Note that an **IMAGE COPY** of a compressed index creates an uncompressed output file. In addition, the log records for the index keys are not compressed.

Asymmetrical Split on Index Pages

Support for asymmetrical splitting of index pages is a new feature in DB2 9. In earlier releases, when all the RID space of an index leaf page is consumed during inserts, page split processing takes places, and DB2 allocates a new page in the index, moving half of the entries from the old page to the new index page. If you

are performing sequential inserts and adding index keys in ascending order, the freed space in the old page of the index will never be reused.

With this enhancement, DB2 detects various insert patterns in an index and chooses an algorithm from several available to DB2 to use for page splitting. These asymmetrical algorithms are available in NFM of DB2 9. Asymmetric page splitting, along with an increase in the index page size from 4 KB to 8 KB, 16 KB, or 32 KB, provides better space use and also reduces contention in the index.

Active Logs

DB2 uses dual logging for the active log, archive log, and bootstrap data sets. When the active log is offloaded to the archive log, the first information that goes to tape or disk is a copy of the BSDS.

The primary function of the dual active logs is to provide recoverability by recording the changes to your data within the system. DB2 uses automatic offloading of these logs to the archive logs to establish a baseline for user data recoverability.

Once DB2 is started, the logs remain allocated exclusively to DB2 until DB2 terminates. The active logs are defined in the BSDS and are allocated dynamically for use. You define the number and size of the log data sets on installation panel **DSNTIPL**. A DB2-provided offline utility, **DSNJU003** (change log inventory), lets you add new logs or replace an active log. To use this utility, you must stop DB2, make your changes, and then restart the DB2 subsystem.

To minimize the filling and "spilling" of logs to archive logs, make sure the active log data sets are large enough to avoid frequent offloading. In addition, be sure to set the **OUTBUFF** DSNZPARM value large enough so that log buffers can avoid the need to wait for a buffer to become available. **OUTBUFF** specifies the output buffer size used in writing active log data sets.

The BSDS should be converted in DB2 9, so you will have 93 active logs available. These are VSAM linear data sets stored as dual sets for log recoverability. The **DSNTIPL** installation panel's **NUMBER OF LOGS** field permits a maximum entry of **31**, the traditional number supported in the BSDS. Check APAR PK77228 in

DB2 9 for a fix that updates this maximum to **93** to match the new BSDS format required by DB2 9.

Records in the dual active log show the activity for updates, deletes, and inserts to table spaces, index spaces, and system events. When log records are written, they are broken into segments. Records are first written sequentially into the output log buffers located in the system services address space before they are written into the active logs. These records are VSAM-formatted control intervals (CIs), which have been assigned an ever-increasing RBA or, in a data-sharing environment, a log sequence number (LRSN) that is written to the active log.

Output log buffers collect the activity against data and system events. When these buffers become full, or when DB2 forces them at commit time, they are written to the active log set. You could thus have log records written several times. The system services address space handles the log buffers, and they are defined with the installation CLIST.

As the dual active log sets fill, they automatically spill or roll off to the assigned archive log sets in a process called offloading. The next set of active logs continues to record system information while this offloading occurs. Active logs are "wrap-around," provided the first log has been archived. DB2 does support striped active log data sets.

Log Definition

Entries on the **DSNTIPL** installation panel describe the logging environment. Macros **DSN6LOGP** and **DSN6ARVP** contain the parameters that control logging.

The system uses the **UPDATE RATE** and **ARCHIVE LOG FREQ** fields of the installation CLIST to calculate the data set size of each active log data set. Your management of the logs and the archives is used in system recoverability.

A log record is identified by an RBA in the first byte of its header. This number uniquely identifies this record.

The log print record is split into four parts: summary information, log record header, log record sub-header, and logged data. To understand more about the

formats of the logs, look at the mapping information in member **DSNDQJ00** in the **SDSNMACS** library. Log records include undo/redo records, in-place updates that represent both the before and after images, and partial image records.

Each log record has a header that indicates its type and the DB2 subcomponent that made the record. Log records can contain compressed data if a table contains compressed data. Reading compressed data requires access to the data dictionary.

When a change is made to a database, DB2 writes a unit of recovery log record that describes the change. These rows, which are logged in the active log, are called undo/redo records. Also written is a Begin UR (begin a unit of recovery) record, which records the first request to change a database. If a unit of work is committed or rolled back, you will see also an End Phase 2 record to reflect that the work has been completed. Exception records can also be written. These are registered as database exception table (DBET) log records to record database exception states when any database table space, index space, or partition is in an exception state.

Checkpoint log records are written when the checkpoint frequency specified by the **CHECKPOINT FREQ** field on panel **DSNTIPL** is reached. Many records can be written for a single checkpoint. The checkpoint will log the current status of DB2 and register the log RBA of the checkpoint in the bootstrap data set.

Log Data Capture

DB2 log data is available for recovery purposes in real time through the log capture exit routine. This online exit captures the data that DB2 writes to the active log, but it does not return data or enter data to DB2.

It is better to use the Instrumentation Facility Interface rather than the log capture exit to read data from the log. You can also read the records with IFI while DB2 is running. Doing so requires a program that uses the IFI commands **READA** and **READS** to capture the log information into a buffer. (You can return the records directly to the program.) To gather the information, you would first run a **START TRACE** command, using IFCID 126 to accumulate active log records into a buffer. This will cause records to be recorded into the buffer as well as to the active log

in DB2. Use the following command to start the performance trace to collect the records:

```
-START TRACE(P) CLASS(30) IFCID(126) DEST(OPX)
```

If you set the **DATA CAPTURE CHANGES** option on a **CREATE TABLE** or an **ALTER TABLE** statement, you can retrieve the log information using programs such as Remote Recovery Data Facility (RRDF) or DB2 DataPropagator.

Recovery Log Manager

Within DB2, the recovery log manager (RLM) maintains the DB2 recovery log by writing log records and retrieving log information in response to resource manager requests. RLM maintains the DB2 BSDS that contains linkage information pointing to all the DB2 log data sets, and it controls the movement of full active log data sets to the archive log. It supports other DB2 resource managers by reading and writing data on the DB2 recovery log. RLM also copies active log data sets to archive log data sets when they become full.

Log records are assigned ascending RBAs and are mapped into physical data sets as described in the BSDS. DB2 data-sharing log records are read from the data-sharing group (or from selected members of the data-sharing group) in merged LRSN sequence.

The recovery log manager provides:

- The **RECOVER BSDS** command.
- The **ARCHIVE LOG** command (the change log inventory utility job that updates the BSDS).
- Print log map utility: Produces a listing showing the log RBAs and LRSNs contained in each log data set. All active and archive data sets that contain log data are included in the listing, as well as all active log data sets that are available for use, all conditional restart control records, the subsystem checkpoint queue, and the DDF communications record.

- DB2 stand-alone log services: Provide an assembler language macro capability that enables user-written application programs to read the DB2 log from a z/OS job environment.
- Log format utility: Pre-formats new active log data sets.

Log I/O Enhancements

DB2 9 for z/OS provides the following enhancements for log I/O:

- The number of active log input buffers is increased from 15 to 120.
- Archive log files are now read using Basic Sequential Access Method (BSAM) instead of Basic Direct Access Method (BDAM).
- Archive log files can use Data Facility Storage Management Subsystem (DFSMS) striping and compression.
- Archive log files input buffers are increased from 1 to *N*, where *N* is proportional to the number of stripes. For each stripe, DB2 uses 10 tracks' worth of buffers, regardless of the block size.
- Archive log buffer processing is converted from **AMODE(24)** to **AMODE(31)**, so DB2 can now move these buffers above the 16 MB line, enabling them to be much larger.
- By exploiting z/OS 1.7 **DSNTYPE=LARGE** disk data set support, installations can have 4 GB DASD archive logs. Active logs no longer require multi-volume allocations.

Installation Panel DSNTIPL

The system services address space in DB2 uses two log buffer areas: one for input and one for output to the active log. You define these log buffer areas, along with other key components, on installation panel **DSNTIPL**. In DB2 9, the panel provides the following fields associated with logging:

OUTPUT BUFFER

The **OUTPUT BUFFER** field specifies the size of the output buffer that is to be used for writing active log data sets. Acceptable values range from 40 KB to 400,000 KB.

Log writes can be synchronous and asynchronous. The logs are written synchronously at commit time. If you reach the write threshold specified in **OUTBUFF** for the log buffer, the flushing is asynchronous. If your performance traces report log buffer write failures greater than zero, set **OUTBUFF** higher; it is not large enough. This setting is updatable using subsystem parameter **OUTBUFF** in macro **DSN6LOGP**.

NUMBER OF LOGS

This field defines the number of data sets to be established for each copy of the active log. If you have converted the BSDS, the number of logs can be defined from **2** to **93**.

ARCHIVE LOG FREQ

This field sets the interval (number of hours) to be used in offloading the active log. The default is **24**, which ensures that the log is offloaded at least once each day.

UPDATE RATE

The default **UPDATE RATE** setting assumes that 400 bytes of data will be logged for each update, insert, or delete. When the CLIST is run, this field and the **ARCHIVE LOG FREQ** field together determine the size of the logs.

LOG APPLY STORAGE

This field sets the maximum amount of **DBM1** storage used by the fast log apply process. During restart, this process is always enabled. This setting is updatable using subsystem parameter **LOGAPSTG** in macro **DSN6SYSP**.

CHECKPOINT FREQ

This field specifies the interval (in minutes or in number of log records) that will start a new checkpoint in DB2. The default setting for the **CHKFREQ** subsystem parameter in macro **DSN6SYSP** is 500,000 records. Use the **SET LOG** command to change the time or number of records between checkpoints in DB2.

FREQUENCY TYPE

This field indicates whether the units specified for checkpoint frequency represent a number of log records (**LOGRECS**) or time (**MINUTES**).

UR CHECK FREQ

This field specifies the number of checkpoint cycles DB2 goes through before issuing a warning message to the console. The value of your checkpoint interval divided by your limit for issuing commits is the best value to use for the system. This setting is updatable using subsystem parameter **URCHKTH** in macro **DSN6SYSP**.

UR LOG WRITE CHECK

This field indicates how many log records are to be written by an uncommitted unit of recovery before a warning message is issued, providing a good indicator of long-running URs. This setting is updatable using subsystem parameter **URLGWTH** in macro **DSN6SYSP**.

LIMIT BACKOUT

This field indicates whether to postpone backward log processing. Possible values are **AUTO** (the default), **YES**, and **NO**. This setting is updatable using subsystem parameter **LBACKOUT** in macro **DSN6SYSP**.

BACKOUT DURATION

This field defines how much log to process for back-out when the **LIMIT BACKOUT** field is set to **YES** or **AUTO**.

RO SWITCH CHKPTS

This field defines the number of consecutive checkpoints permitted after a partition or page set is updated. When this threshold is reached, DB2 will convert the partition or page set to read-only status. Table spaces for which **NOT LOGGED** has been specified are always converted to read-only after one checkpoint. The default **RO SWITCH CHKPTS** value is **5** (five checkpoints). This setting is updatable using subsystem parameter **PCLOSEN** in macro **DSN6SYSP**.

RO SWITCH TIME

Specified in minutes, this field's value works with that of the **RO SWITCH CHKPTS** field to change the partition or page set from read-write to read-only status. For infrequently used table spaces, this setting can reduce the recovery and logging process. The default value is **10** (minutes). For **NOT LOGGED** table spaces, DB2 converts the partition or page set after one minute, regardless of this field's value. The **RO SWITCH TIME** setting is updatable using subsystem parameter **PCLOSET** in macro **DSN6SYSP**.

LEVELID UPDATE FREQ

This field controls how often (in number of checkpoints) the level ID of a table space set or partition is updated. It also controls how often the value that the **RECOVER LOGONLY** utility uses as a starting point for log apply is updated. The default is **5** (checkpoints). This setting is updatable using subsystem parameter **DLDFREQ** in macro **DSN6SYSP**.

Notes on Logs

Some additional points to note related to the active logs:

- DB2 must pre-format a VSAM control area before writing the active log records the first time. Use the **DSNJLOGF** stand-alone utility to pre-format the active log data sets before they are used by DB2.
- If you compress your data, the log information about that data is also compressed. However, any indexes you compress are not logged as compressed records.
- Logging rates have improved in DB2 9, and striping is supported for archive logs.
- The DB2 **ARCHIVE LOG** command truncates the current active log data sets. This operation runs an asynchronous offload and updates the BSDS with a record of the offload.
- The **SET LOG** command flushes only the log buffers.
- The **STOP DATABASE** command makes certain the active log is not allocated to DB2. You can then use the **DSN1LOGP** utility to read the active log.

- Choices in DB 9 include **LOG YES** or **LOG NO** for logging attributes on the **CREATE TABLESPACE** and **ALTER TABLESPACE** statements. You can have different logging attributes for partition table spaces. Specifying **LOG NO** for a segmented table space that contains multiple tables affects all the tables in that segmented table space. You can find the setting in the catalog in the **SYSIBM.SYSTABLESPACE LOG** column. *Use the **LOG NO** option with caution.*

It is no surprise that **SYSLGRNX** (system log range) records are not kept for **NOT LOGGED** table spaces. Note in **SYSCOPY** that the column **LRSN** indicates the point in the log where the logging attribute was changed. A new column, **LOGGED**, has been added to the **SYSCOPY**. The value **Y** indicates logging, **N** is for no logging, and a blank indicates the row was built before DB2 9.

The Log Itself

The VSAM control interval holds 4,089 bytes of DB2 log information. This is what is referred to as the physical record. When information is written, this is the logical record, and its length depends on the space available in the CI.

All logical records have a header, called a log record header (LRH), that contains control information. A suffix, called the log control interval definition (LCID), describes how the record segments are placed into the physical control interval.

Each log record has a code or type that describes the event that recorded the record. There are also subtype codes to produce a more in-depth event description.

You can use mapping macros provided in macro **DSNDQJ00** in the **SDSNMACS** data set to interpret the log record formats for records and subtypes. Each macro is self-documenting with notes. You can also acquire log statistics from your monitor to calculate the minimum megabytes per second required to write to the active log data set to understand your logging requirements.

Size of the Active Log Data Sets

The frequency of offloading to the archive is key to determining the size of the logs. The CLIST uses the **UPDATE RATE** and **ARCHIVE LOG FREQUENCY** fields on installation panel **DSNTIPL** to determine the disk space required by the logs.

Basically, the size is calculated as follows to find the hours in the archive period for one log; you would double this result for dual logging:

*Disk space required by logs = (Data change log record size) * (Data change rate per hour) * (Hours in archive period)*

Log Utilities

DB2 9 includes enhancements to several stand-alone utilities related to the logs.

DSN1LOGP

The **DSN1LOGP** utility, which formats the contents of the recovery log for display, now detects possible erroneous recovery information. If you attempt to print a range of records but the range is no longer in the BSDS because the archive logs have rolled off, error message DSN1224I indicates that the range could not be found and returns an RC4.

When you specify the utility's **SUMMARY(YES)** option, the report shows whether objects are logged or not logged. This option might not be available if the specified range of log records does not include the Begin UR log record.

Note that a clone table's object identifier will have the eighth, or high-order, bit set on, so the OBID for the clone might be **x'8xxx'** and the base table would have **x'0xxx'** as the OBID. When reading the output of the **DSN1LOGP** utility, you need to be aware of this clone naming convention.

DSNJU003

You use the change log inventory utility, **DSNJU003**, to create conditional restart control records. In DB2 9, you can now use timestamps for your normal conditional restart and for the **RESTORE** system utility.

DSNJU004

The print log map utility, **DSNJU004**, runs as a batch job. You can use it to print the BSDS contents and the conditional restart record. You can also use this utility to determine the current log configuration. Each checkpoint that is displayed now shows the stored clock value in the checkpoint queue section. Each log record

includes a suffix that describes how the record segments are placed in the physical control intervals.

The print log map utility output has changed regarding DDF. Several keywords related to request identification have been added. These changes are reflected in the output of utility **DSNJU003**. **DSNJU004** now prints the DDF information at the beginning of the output, before the log information.

To determine how much space is left in the log, obtain the high-written RBA in the log and then subtract it from **x'FFFFFFFFFFFF'**. If you have applied APAR PK27611, use RBA **x'FFFF00000000'** instead.

DSNJU004 provides an assortment of data you will find very useful. Its information includes:

- Names of active and archive log sets
- BSDS information
- Active logs available, log starting/ending RBA values
- Checkpoint record contents in BSDS
- System-level backup information (new in DB2 9)
- Checkpoint queue contents
- Utility and system timestamps
- Conditional restart control records
- Information in the system Coded Character Set Identifier (CCSID)

System Checkpoints

When you have a long-running unit of recovery, DB2 records a large number of log records from the beginning to the end of the transaction. Usually, the jobs running are doing a large number of updates without proper committing within the program. The **DSNTIJUZ** job includes a parameter, **URLGWTH**, that deals with long-running reader threshold. The parameter's default value is **0** (zero), which means that long-running UOR checking is not activated by default.

Installation panel **DSNTIPN** provides the value for checkpoint frequency (in the **CHECKPOINT FREQ** field). In DB2 9, system checkpoints now store information at the table level to track segmented table spaces independently. Stored in the UR checkpoint record along with table space and partition level, the checkpoints record information about each modified object uncommitted unit of recovery. This process assists in the back-out processing at start-up time.

In DB2 9, the updating of **SYSLGRNX** entries is deferred beyond the start-up, allowing faster restart of DB2. The entries are updated at the first system checkpoint that follows the restart.

Archive Logs

The archive logs are sequential data sets and support up to 10,000 archive log data sets per log copy. In DB2 9 CM, all active logs support BSAM to read the archive data sets. In NFM, archive logs can be defined as extended format (EF) data sets and must be SMS-managed. As a result of the extended formatting, compression and striping are allowed.

Conversion of Archive Log Processing from AMODE(24) to AMODE(31)

Before DB2 9, some customers reported running short of storage below the 16 MB line in the **MSTR** address space during archive log processing. Storage constraint relief and increased archive log processing are a result of converting using 31-bit mode in DB2. In addition, DB2 9 moves these buffers above the 16 MB line and, when possible, uses dual buffering for archive log reads or writes. This support is available starting in DB2 9 CM.

Bootstrap Data Set

The bootstrap data set is an inventory manager for the active and archive logs, passwords for the directory and catalog, and conditional restart and checkpoint record information. The BSDS is the only VSAM key-sequenced data set (KSDS) in DB2. It contains name and status information for DB2 and RBA range specifications for all active and archive log data sets.

Each time an active log is archived, a copy of the BSDS goes along with it. The BSDS is the first entry on the tape or disk archive log backup. You need take no special steps to keep the BSDS updated with records of DB2 logging events; the system handles this task automatically.

Each BSDS requires 3.5 MB of space. DB2 automatically allocates two BSDS copies at installation time. When moving to DB2 9, make sure you convert your BSDS by running the **DSNJCNVB** conversion utility, which is part of job **DSNTIJUZ**.

To migrate to DB2 9, your BSDSs must be in the new expanded format, which supports up to 10,000 archive log volumes and up to 93 active log data sets for each copy of the log. This new format became available beginning with DB2 8 NFM. When executing the job, expect a return code of **888** from this step if your BSDSs have already been converted. Job **DSNTIJUZ** accepts return codes of **0** and **888** from the **DSNTCNVB** step.

Your shop might require changes to active or archive logs that necessitate a corresponding change to the contents of the BSDS, such as:

- Adding more active log data sets
- Recovering a damaged BSDS
- Discarding outdated archive log data sets
- Copying active log data sets to newly allocated data sets
- Moving log data sets to other devices
- Adding or changing the DDF communications record
- Creating or canceling control records for conditional restart

The DB2 batch change log inventory utility, **DSNJU003**, lets you change the contents of the BSDS.

Do not run utility **DSNJU003** when DB2 is active. DB2 must be inactive, or unpredictable results may occur.

To change the BSDS:

1. Make sure you have a backup of the BSDS.
2. Issue the **STOP DB2 MODE(QUIESCE)** command to stop the DB2 subsystem.
3. Run utility **DSNJU003**.
4. Restart DB2 with the **START DB2** command.

Using the Access Method Services (IDCAMS) **REPRO** command, you can copy an active log data set, but only when DB2 is down. DB2 allocates the active log data sets as exclusive (**DISP=OLD**) at DB2 start-up.

Virtual Buffer Pools

Buffer pools are memory allocations for the storage of pages of information from table spaces and index spaces. The total storage in all buffer pools should not exceed 1 TB. In DB2 9, buffer pools have moved above the 2 GB bar. Make sure your buffer pools are backed up by real storage. Buffer pools are located and managed by the **DBM1** address space.

During installation or migration, you specify a name and size for each buffer pool. An operator command, **ALTER BP**, is available in case you need to change these values.

Buffer Pool Definition

Panel **DSNTIP1** (Figure 2.14) is the first of two installation/migration CLIST panels that deal with the buffer pools.

```
 DSNTIP1         INSTALL DB2 - BUFFER POOL SIZES - PANEL 1
===>
     _


 1 DEFAULT  4-KB BUFFER POOL FOR USER DATA ===> BP0       BP0    - BP49
 2 DEFAULT  8-KB BUFFER POOL FOR USER DATA ===> BP8K0     BP8K0  - BP8K9
 3 DEFAULT 16-KB BUFFER POOL FOR USER DATA ===> BP16K0    BP16K0 - BP16K9
 4 DEFAULT 32-KB BUFFER POOL FOR USER DATA ===> BP32K     BP32K  - BP32K9
 5 DEFAULT BUFFER POOL FOR USER LOB DATA   ===> BP0       BP0    - BP32K9
 6 DEFAULT BUFFER POOL FOR USER XML DATA   ===> BP16K0    BP16K0 - BP16K9
 7 DEFAULT BUFFER POOL FOR USER INDEXES    ===> BP0       BP0    - BP32K9
Enter buffer pool sizes in number of pages.
 8 BP0  ==> 20000        18 BP10 ==> 0           28 BP20 ==> 0
 9 BP1  ==> 0            19 BP11 ==> 0           29 BP21 ==> 0
10 BP2  ==> 0            20 BP12 ==> 0           30 BP22 ==> 0
11 BP3  ==> 0            21 BP13 ==> 0           31 BP23 ==> 0
12 BP4  ==> 0            22 BP14 ==> 0           32 BP24 ==> 0
13 BP5  ==> 0            23 BP15 ==> 0           33 BP25 ==> 0
14 BP6  ==> 0            24 BP16 ==> 0           34 BP26 ==> 0
15 BP7  ==> 0            25 BP17 ==> 0           35 BP27 ==> 0
16 BP8  ==> 0            26 BP18 ==> 0           36 BP28 ==> 0
17 BP9  ==> 0            27 BP19 ==> 0           37 BP29 ==> 0
```

Figure 2.14: Installation panel DSNTIP1 – Buffer pool sizes (panel 1)

Here, you specify the default buffers for user data, LOB and XML data, and indexes in DB2 9. These settings apply to objects that are created implicitly and to objects that are created explicitly without the **BUFFERPOOL** clause. Table 2.6 lists the subsystem parameters and default values associated with these entries.

Table 2.6: Buffer pool size DSNZPARMs

DSNZPARM	Description	Acceptable values	Default
DSN6SYSP TBSBPOOL	Default 4 KB buffer pool for user data	Any 4 KB buffer pool name	**BP0**
DSN6SYSP TBSBP8K	Default 8 KB buffer pool for user data	Any 8 KB buffer pool name	**BP8K0**
DSN6SYSP TBSBP16K	Default 16 KB buffer pool for user data	Any 16 KB buffer pool name	**BP16K0**
DSN6SYSP TBSBP32K	Default 32 KB buffer pool for user data	Any 32 KB buffer pool name	**BP32K**
DSN6SYSP TBSBPLOB	Default buffer pool for user LOB data	Any 4 KB, 8 KB, 16 KB, or 32 KB buffer pool name	**BP0**
DSN6SYSP TBSBPXML	Default buffer pool for user XML data	Any 16 KB buffer pool name	**BP16K0**
DSN6SYSP IDXBPOOL	Default buffer pool for user indexes	Any 4 KB, 8 KB, 16 KB, or 32 KB buffer pool name	**BP0**

Figure 2.15 shows the second installation/migration CLIST panel where you define the buffer pools, **DSNTIP2**. You use this panel to specify the total number of buffers for a given virtual buffer pool from **BP30** to **BP32K9**.

```
 DSNTIP2         INSTALL DB2 - BUFFER POOL SIZES - PANEL 2
===> _

Enter buffer pool sizes in number of pages.
 1 BP30  ==> 0          18 BP47   ==> 0          35 BP16K4 ==> 0
 2 BP31  ==> 0          19 BP48   ==> 0          36 BP16K5 ==> 0
 3 BP32  ==> 0          20 BP49   ==> 0          37 BP16K6 ==> 0
 4 BP33  ==> 0          21 BP8K0  ==> 1000       38 BP16K7 ==> 0
 5 BP34  ==> 0          22 BP8K1  ==> 0          39 BP16K8 ==> 0
 6 BP35  ==> 0          23 BP8K2  ==> 0          40 BP16K9 ==> 0
 7 BP36  ==> 0          24 BP8K3  ==> 0          41 BP32K  ==> 250
 8 BP37  ==> 0          25 BP8K4  ==> 0          42 BP32K1 ==> 0
 9 BP38  ==> 0          26 BP8K5  ==> 0          43 BP32K2 ==> 0
10 BP39  ==> 0          27 BP8K6  ==> 0          44 BP32K3 ==> 0
11 BP40  ==> 0          28 BP8K7  ==> 0          45 BP32K4 ==> 0
12 BP41  ==> 0          29 BP8K8  ==> 0          46 BP32K5 ==> 0
13 BP42  ==> 0          30 BP8K9  ==> 0          47 BP32K6 ==> 0
14 BP43  ==> 0          31 BP16K0 ==> 500        48 BP32K7 ==> 0
15 BP44  ==> 0          32 BP16K1 ==> 0          49 BP32K8 ==> 0
16 BP45  ==> 0          33 BP16K2 ==> 0          50 BP32K9 ==> 0
17 BP46  ==> 0          34 BP16K3 ==> 0
```

Figure 2.15: Installation panel DSNTIP2 – Buffer pool sizes (panel 2)

Remember that the total amount of your buffer pool storage cannot exceed 1 TB. In the 64-bit addressing space, DB2 can have up to 16 exabytes of virtual storage addressability by a single DB2 address space.

Buffer Pool Management

You can manage the buffer pools yourself through tuning and monitoring, or you can use the services of dynamic buffer pool size adjustments so that the system's memory resources can be more effectively used to achieve workload performance goals. This new service is available in conversion mode in DB2 9.

DB2 9 supports Workload Manager–assisted buffer pool management with z/OS 1.8. This WLM service assists in making dynamic buffer pool size adjustments on realtime monitored workloads. With dynamic buffer pool assist, a DB2 subsystem on the same LPAR might have non-critical buffer pools decreased in size while those pages are reassigned to a critical subsystem.

The automatic management of buffer pool storage provides sizing information to WLM. Buffer pools are registered with WLM, and DB2 communicates with WLM every time an allied agent encounters delays relating to read I/O and periodically reports the buffer pool size and random read hit ratios to WLM.

WLM projects the effect of making a change to the size of the buffer pool over time from collected data maintained in a histogram. It plots the size and hit ratio over time to determine whether work is achieving its goals and then determines the appropriate course of action.

WLM decides whether increasing the buffer pool size could help achieve the performance goal if I/O delays are occurring. If sufficient storage is available, WLM could decide to increase a buffer pool or even decrease one buffer pool and increase another. If the adjustment is made, the results will be just as though you had issued an **ALTER BUFFERPOOL VPSIZE** command.

To enable this feature, use the **AUTOSIZE(YES)** setting on the **CREATE TABLESPACE** command. You can modify individual buffer pools by using the **ALTER BUFFERPOOL** command with **AUTOSIZE(YES/NO)**. The default for automatic buffer pool adjustment is off (**NO**).

To summarize, DB2 takes the following steps to perform automatic management of buffer pool storage:

1. Registers the buffer pool with the WLM
2. Provides the sizing information to WLM
3. Communicates to WLM each time allied agents encounter delays due to read I/O
4. Periodically reports buffer pool size and random read hit ratios to WLM

Buffer Pool Details

Buffer pools temporarily store pages of data from table spaces or indexes in storage. DB2 buffer pool pages are not paged out; the EDM pools and the sort pool are held in real storage. For the DB2 buffer pools, the EDM pool, and working storage, the amount of real storage must be the same as the amount of virtual storage. Paging activity in the buffers is an indication of a problem.

DB2 lets you use up to 50 different 4 KB buffers and up to 10 different buffer pools each for 8 KB, 16 KB, and 32 KB buffers. You can set the size of each of these buffer pools separately during installation.

As of DB2 9, using above-the-bar real storage enables the use of very large buffers. As a result, applications can avoid a substantial amount of read and write I/O.

The following points summarize the process flow of an application with respect to the buffer pools:

1. The program accesses a row of a table.
2. DB2 places the page that contains that row in a buffer.
3. Is the data already in a buffer? If so, the application program does not have to wait for it to be retrieved from disk.
4. Has the row been changed? If so, data in the buffer must be written back to disk eventually. A write operation might be delayed until DB2 takes a

checkpoint or until one of the related write thresholds is reached. (The data-sharing environment writing mechanism differs somewhat.)

5. The data remains in the buffer until DB2 decides to use the space for another page. Data can be read or changed without a disk I/O operation until written.

Buffer Pool Analyzer and ALTER BUFFERPOOL Statement

You can use the Buffer Pool Analyzer for z/OS to obtain recommendations about buffer pool allocation changes and perform "what if" analysis of your buffer pools. To change the size and other characteristics of a buffer pool or to enable DB2 automatic buffer pool size management for a buffer pool, you use the DB2 **ALTER BUFFERPOOL** command. You can issue this command at any time while DB2 is running.

In DB2 9, you can specify 4 KB, 8 KB, 16 KB, or 32 KB buffer pools for user indexes. Indexes that are created during conversion mode require a 4 KB buffer pool. If you do not specify a 4 KB buffer pool in the **BUFFERPOOL** clause when you create an index in CM, DB2 issues the following error:

```
SQLCODE = -676, ERROR: ONLY A 4K PAGE BUFFERPOOL CAN BE USED FOR AN INDEX
```

Work File Database

The work file database is the only temporary database in DB2 9. It is used for all temporary tables. In DB2 9, the work file database merged with the **TEMP** database; the **TEMP** database is no longer used. Job **DSNTIJTM** creates data sets for the work file database.

The work file database **DSNDB07** houses work file table spaces that are logical work files used in sorting. This sorting could be the result of an SQL **ORDER BY**, **GROUP BY**, **DISTINCT**, **JOIN**, or other request. Multiple table spaces are defined for 4 KB and 32 KB sizes in DB2 9, requiring buffer pools to be assigned to these table spaces as well.

You should increase your space for 32 KB work files. You can reallocate the space by stopping the **DSNDB07** database, deleting the old work file data sets, and redefining the work file data sets as larger or increasing the number of data sets.

Next, issue the **CREATE TABLESPACE** statement for each new work file. Then, start the **DSNDB07** database.

When allocating multiple table spaces in the sort work file database, keep in mind that DB2 9 gives preference to the least recently used table space whose secondary quantity (**SECQTY**) is **0** (zero). It is advisable to allocate multiple work file table spaces with a **SECQTY** of **0** to facilitate efficient concurrent I/O.

When allocating the table spaces in the work file database for use by declared global temporary tables (DGTTs) in your applications, define the table spaces with a non-zero secondary quantity (**SECQTY>0**) in the work file. DGTTs do not span multiple table spaces; they are limited to one table space. Allocating your work files this way will minimize the contention for space in the work files. You will need to monitor and tune your definitions of the table spaces in the work file database to ensure efficient space use. For more information, see IBM Tech Note 1386786 and maintenance PK70060/UK46839 and PK67691/UK47354.

Utilities do not run against the work file database. Statistics are kept on the table spaces in columns **NACTIVE**, **SPACE**, and **EXTENTS** in the database.

When sorting begins, rows are written to work files. At the end of the input phase, the rows are sorted. If multiple work files are used, they are merged together to produce one work file at the end of the input phase.

When an application needs to sort data, the work files are allocated on a least recently used basis for a particular sort. These work files are logical work files (LWF) that reside in the work file table spaces in **DSNDB07** in a non–data-sharing environment. DB2 uses a buffer pool when writing to the LWF.

Only the buffer pool size limits the number of work files that can be used for sorting. DB2 can support large sorts by allocating a single logical work file to several physical work file table spaces. Very large buffer pools can also help avoid disk I/O.

If you think global temporary tables are monopolizing your sort pool, take a look at performance class 8 using IFCID 0311. Various factors can impact the performance of the sort process, such as the size of the sort pool, the size of the

row being sorted, and whether I/O contention is occurring. Keep your work files in buffer pools separate from other data.

Things to Know About Work Files

DB2 uses 32 KB work file data sets when the total work file record size, which includes the record overhead, exceeds 100 bytes.

You can use the sort summary trace record (IFCID 0096) to show the number of sorted records, the record size, and whether a merge phase was required.

Using Parallel Access Volume (PAV) disk devices is a way to minimize I/O contention.

As I/Os occur in the merge phase of a sort, DB2 uses sequential prefetch to bring eight pages into the buffer pool. If there are insufficient pages in the work file buffer pool, DB2 reduces the prefetch quantity to four or less and might disable prefetching entirely.

In the buffer pool that supports only 4 KB or 32 KB work files, set **VPSEQT** (the sequential steal threshold) to 99 percent to avoid overwriting space maps. The default setting is 80 percent, which means 20 percent of the buffers can go unused.

If you have SQL that uses an in-memory sparse index scan as an access method, know that in-memory sparse index is based on the sorted work file. DB2 uses the sparse index to fetch records from the work file rather than a table space scan.

Increase the **DWQT** (deferred write threshold) and **VDWQT** (vertical deferred write threshold) values. If you reach the threshold set for these values, writes are scheduled.

You cannot use **RECOVER** on the **DSNDB07** work files (except in a data-sharing environment).

If **DSNDB01.DBD01** is stopped or inaccessible, the descriptor is not loaded into main storage, and the work file will not be allocated. To get around this situation, stop and restart the work file database after **DBD01** is available.

Remember that the default buffer pool for 4 KB sorting in DB2 is **BP0**. You should change the default from **BP0** to **BP07** or your choice of buffer pool.

During an **INSERT** statement, a temporary work file result table is populated, and it uses work file space. Until this work file space goes away, no other process can use that same work file space. The space can be released by the program at commit, rollback, or deallocation time. Use IFCID 0311 in performance trace class 8 to distinguish these tables from other uses of the work file. This IFCID is the global temp table usage trace. Among other things, it gives you the program name and information about the package that uses the global temporary table.

In prior releases, if you granted privileges on the TEMP database, you have to grant those privileges on the work file database to avoid authorization errors.

In DB2 9, the **DSNTIJTM** job creates the default storage group **SYSDEFLT**, defines the database that is for temporary work files, and binds DB2 REXX Language Support. By changing the parameters in the last step of installation job **DSNTIJTM**, you can specify a different user-managed storage group. You can create additional work file table spaces after running **DSNTIJTM** by using the **DSNTWFG** exec program in job **DSNTIST**. Check the comment block in this job step for information about the parameters for **DSNTWFG**.

If storage space is not a problem, wait to drop the temporary database. Falling back to DB2 8 will require you to re-create it.

You must define at least one of the table spaces in the work file database with a page size of 32 KB. Static scrollable cursor result tables and declared global temporary tables both require a 32 KB work file database table space. There are no 8 KB or 16 KB table spaces in the work file database. You will find a sample query to check your work file database in **SDSNSAMP** member **DSNTESQ**.

Rows of a declared global temporary table reside in a table space in a work file database. DB2 will not create an implicit table space for the DGTT table.

There cannot be more than 500 table spaces in the work file database for 4 KB or 32 KB work files.

For all agents on the local DB2, the number of declared global temporary table indexes cannot be greater than 10,000.

DB2 9 lets you use the installation CLIST to add table spaces to the work file database as part of the **MIGRATE** process. Earlier releases allowed this process only in **INSTALL** mode. The segment size of the table spaces is restricted to 16 until DB2 enters new-function mode. During migration, job **DSNTIJTC** creates and updates indexes on the catalog tables. The migration job will fail if you do not have enough storage.

Your buffer pool hit ratio for the work file buffer pool is a measure of how often a page access (a getpage) is satisfied without requiring an I/O operation. Make your buffer pools large enough to increase the buffer hit ratio.

Sort Pool

The sort pool is part of the database services address space. DB2 uses a tournament sort; in this technique (depicted in Figure 2.16), the algorithm produces logical work files called runs. These runs are intermediate sets of ordered data. If sufficient room exists, sorting is done in the sort pool; otherwise, the work file database is used.

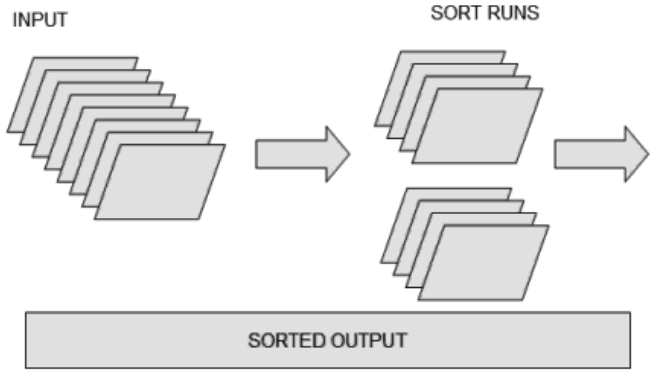

Figure 2.16: Sorted output

The **SORT POOL SIZE** field on installation panel **DSNTIPC** (or the subsystem parameter **SRTPOOL** in macro **DSN6SPRM**) sets the amount of storage required for the sort pool.

The buffer pools, sort pool, and RID pool have been moved above the 2 GB bar and are no longer included in the region size calculation. Local storage for the sort pool is created above the 2 GB bar at allocation time. The above-the-bar sort pool contains sort tree nodes. This local sorting can be invoked when a cursor within a program requires a sort; this usually is prompted by an **ORDER BY** clause.

DB2 allocates 240 KB as a minimum and 128 MB as a maximum for each concurrent sort. The default size of the sort pool is 2 MB. DB2 starts with 240 KB and then adds more storage until either the limit is reached or a maximum number of nodes is populated in the sort tree (32 KB). The DB2 in-memory sort work area maintains the storage boundaries for each concurrent sort operation.

You can change the sort pools and other CLIST calculations on installation panel **DSNTIPC** (Figure 2.17). Using the **SORT POOL SIZE** field, you can make the sort pool as large as possible. The default value, as previously noted, is 2 MB.

```
DSNTIPC         INSTALL DB2 - CLIST CALCULATIONS - PANEL 1
===>
 You can update the DSMAX, EDMPOOL, EDMPOOL STATEMENT CACHE, EDM DBD CACHE,
 EDM SKELETON POOL, SORT POOL, and RID POOL sizes if necessary.
                                      Calculated   Override
 1  DSMAX - MAXIMUM OPEN DATA SETS    =     9960                (1-100000)
 2  DSNT485I  EDMPOOL STORAGE SIZE    =    13142  K            K
 3  DSNT485I  EDM STATEMENT CACHE     =    56693  K            K
 4  DSNT485I  EDM DBD CACHE           =    11700  K            K
 5  DSNT485I  EDM SKELETON POOL SIZE  =     5120  K            K
 6  DSNT485I  BUFFER POOL SIZE        =      101  M
 7  DSNT485I  SORT POOL SIZE          =     2000  K            K
 8  DSNT485I  RID POOL SIZE           =     8000  K            K
 9  DSNT485I  DATA SET STORAGE SIZE   =    17928  K
10  DSNT485I  CODE STORAGE SIZE       =    38200  K
11  DSNT485I  WORKING STORAGE SIZE    =    55800  K
12  DSNT486I  TOTAL MAIN STORAGE      =      238  M
13  DSNT487I  TOTAL STORAGE BELOW 16M =     1159  K WITH SWA ABOVE 16M LINE
14  DSNT438I  IRLM LOCK MAXIMUM SPACE =        2  G, AVAILABLE =   2 G
```

Figure 2.17: Installation panel DSNTIPC – CLIST calculations

To investigate further, take a look at IFCID 217 (storage sizes) to find out the total amount of storage available for storage manager pools.

System Error Reporting

In MVS, most messages are issued from the **JES2/JES3** subsystems, Data Facility Product (DFP), various system products, and application programs. The locations

of these messages can be found at the console, hard-copy log, job log, or **SYSOUT** data set. Other messages can be found in the MVS system log data set (**SYSLOG**). In z/OS, the job log is used to show the start sequence of DB2.

The DB2 system administrator has the responsibility to debug problems within DB2. Sometimes these problems may be the DB2 software, a user job, or a transaction. To aid in determining which component has failed, you capture the memory contents in a dump. There are different ways to take these dumps and different data sets that store this information.

Dumps, along with other information (e.g., the console log), can provide information about where the components have failed. The information you collect can be written by MVS in four types of system data sets. Figure 2.18 shows the four types of error data that can be collected.

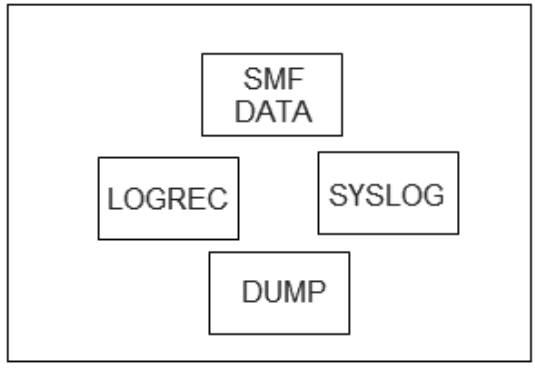

Figure 2.18: Locations of error data collected

SMF data produces analysis reports, **LOGREC** data contains statistical data about machine failures such as I/O errors, **SYSLOG** data (which resides in the **JES2/3** spool space), holds console messages and other system communications, and **DUMP** data sets record areas of virtual storage.

System Display and Search Facility

The z/OS System Display and Search Facility (SDSF) lets you monitor, manage, and control z/OS. You can control job processing, control output, browse jobs without printing, and manage system resources. Data is displayed on panels in

tabular format. You select a desired function on an SDSF panel, or you can enter **SDSF** or system commands on the **COMMAND INPUT** line.

- You can manage jobs on the Status (option **ST**) panel. Here, you will see the status of jobs on any queue, including held and non-held output.
- **DA** shows only active jobs (address spaces).
- **O** (output) displays information about output that is ready to be printed.
- **H** (help output) shows the jobs on hold.

Operators and system programmers have a more robust SDSF Primary Option menu (Figure 2.19) that includes the system log, WLM resources, and more. You can use this menu to manage jobs.

```
 Display  Filter  View  Print  Options  Help
-------------------------------------------------------------------------
          ----------------  SDSF PRIMARY OPTION MENU  ---------
COMMAND INPUT ===> _

DA     Active users                        INIT  Initiators
I      Input queue                         PR    Printers
O      Output queue                        PUN   Punches
H      Held output queue                   RDR   Readers
ST     Status of jobs                      LINE  Lines
                                           NODE  Nodes
LOG    System log                          SO    Spool offload
SR     System requests                     SP    Spool volumes
MAS    Members in the MAS
JC     Job classes                         ULOG  User session log
SE     Scheduling environments
RES    WLM resources
ENC    Enclaves
PS     Processes

END    Exit SDSF
```

Figure 2.19: Expanded SDSF Primary Option menu

Using SDSF, you can view the system log online, view a merged sysplex log (**LOG O**), view a separate display of system requests (including action messages and write to operator with reply messages, or WTORs), and search for specific information using SDSF commands.

The SDSF **FIND** command lets you search the system log. New log data is added to the bottom of the log, so you might want to repeat the find or enter a command that repeats the search at some time interval, such as **BOT &20**, which goes to the bottom of the log every 20 seconds.

SYS1.LOGREC

The **SYS1.LOGREC** data set is a repository of information about system-level software errors or hardware problems. You can use **LOGREC** as a starting point for diagnosing a system problem. You typically look at logs (which can include **SYSLOG**, **OPERLOG**, the job logs, and traces associated with the problem you are investigating) from the starting time frame of the problem. The problems in need of investigation can be varied and may include loops, abends, system hangs, performance problems, and output problems.

DB2 Tools for Problem Diagnosis

Several DB2 commands are available to assist in DB2 problem diagnosis:

- **DISPLAY DATABASE(*) USE/LOCKS LIMIT(*)**
- **DISPLAY UTILITY (*)**
- **DISPLAY THREAD(*) DETAIL**
- **D A,ALL** (or **D A,ssid*** or **D A,IRL***)
- **D GRS,CONTENTION**
- **D OPDATA**

The MVS system log data set (**SYSLOG**) contains the output from these commands; be sure to keep this information. The command output is also stored in the master trace table. The master trace table output in a dump includes the most recently issued system messages. Use these commands before performing a dump for the DB2 address spaces to capture information.

Data-sharing environments should find two additional commands useful in collecting information:

- **F irlmproc,STATUS,ALLI**
- **F irlmproc,STATUS,ALLD**

Running the DB2 **DIAGNOSE** utility with the **MEPL** option produces a Module Entry Point Listing, which shows the PTF level of the DB2 modules. The DB2 **MSTR** dump will contain this data as well.

If the DB2 subsystem or the **MSTR** address space is hung, the **DISPLAY** commands will not produce a response. In that case, the only other option is a dump of the related DB2 address spaces.

Dumps

Certain codes and messages generated by DB2 require you to dump the DB2 address spaces. You typically will dump the **DBM1**, **MSTR**, or **IRLM** address space. You may be asked to provide a DDF, DIST, or SPAS dump. When problems occur, look for all available diagnosis information, which might include traces, dumps, **SYS1.LOGREC** entries, error messages, hardware device problem information, and any other information you can assemble to help diagnosis the issue.

Several dump types are provided in z/OS:

- Abend
- Snap
- Stand-alone
- Supervisor call (SVC)
- Transaction

The type of dump you select depends on the problem you are experiencing and the data you may need to acquire.

If you experience a program problem or an abnormal end to an authorized program, you can request an *abend dump*. Three types of abend dumps can be produced:

- **SYSABEND** is the largest of the abend type dumps.
- **SYSUDUMP** is the smallest of the abend dumps.

- **SYSMDUMP** is the only abend dump you can format using the Interactive Problem Control System (IPCS). It contains a summary of the failing program and some system data relating to the task.

If you are testing a problem program and need a dump while the program is running, a *snap dump* is appropriate. This type of dump shows one or more virtual storage areas that a running program requests. Snaps are preformatted; you cannot use IPCS to format a snap dump.

A *stand-alone dump* is called for when a system problem occurs — for example, when system processing is slow or stops or when the system is in a wait or a loop.

If you have a system problem but the system continues to process, a *supervisor call dump* is recommended. There are two ways to use SVC dumps:

- When the system experiences an unexpected system error but continues to process
- When an operator or an authorized program requests an SVC dump

An SVC dump produces a summary dump with control blocks and system code and can be analyzed using IPCS.

A transaction dump is a CICS formatted dump for the program that was active at the time the dump was requested. A transaction dump indicates where the error occurred within the program.

Based on the error you receive (e.g., 04E or 04F abends), you may be asked to obtain a dynamic dump, which is sent to the **SYS1.DUMP*xx*** data set. You might also be requested to copy the BSDS using the **DSNJU004** utility.

Under z/OS, DB2 supports IPCS and the DB2 dump formatter/printing routine, **DSNWDPRD**. To invoke **DSNWDPRD**, use the following command:

```
DB2 VERBEXIT DSNWDMP
```

The *DB2 Diagnosis Guide* and the z/OS dump and printing documentation provide more information about this control statement.

SMF

A DB2 trace produces SMF records. The DB2 Instrumentation Facility Component (IFC) provides a trace facility. The analysis of the trace records must take place outside DB2.

You can send data to SMF in several ways:

- By writing an application to read and report information from the SMF data set
- By using OMEGAMON to format, print, and interpret DB2 trace output
- By using Tivoli Decision Support for z/OS to collect data and create reports

SMF must be operational before you can send data to it. You use member **SMFPRM*xx*** in **SYS1.PARMLIB** to activate SMF and indicate which types of records SMF will accept. Be sure to specify the **ACTIVE** parameter and the proper **TYPE** sub-parameter.

The EREP Program

The z/OS error facilities can collect information about hardware and software errors in the form of an error log. You can then use the Environmental Record Editing and Printing (EREP) program to produce reports from this data. The error log can be written to an MVS data set or, in a Sysplex environment, to a log stream in the coupling facility. Or, you may instead decide not to record the error log at all.

The **LOGREC** parameter of the **IEASYS*xx*** member of **SYS1.PARMLIB** defines the location of the error log. The **ICFEREP1** utility offloads the log to a history data set, which you can then use to make periodic tape backups and to print reports. Another utility, **IFCOFFLD**, lets you perform an emergency offload of the EREP data set. This alternative is quick but does not record any statistical information. If the data set fills up, recording stops and Z/OS continues with no error log recording. The manuals detail the recommended methods for coding and scheduling the offload and reporting jobs.

When an internal error occurs, DB2 records it in **SYS1.LOGREC** — known as the "logrec" or the Error Recording Data Set (ERDS). You typically obtain a listing of **SYS1.LOGREC** data sets by executing program **IFCEREP1**, the EREP job. The EREP program can format error reports.

A downloadable tool called the Logrec Viewer lets you view the **LOGREC** using the Interactive System Productivity Facility (ISPF). For more information and the download link for this tool, go to *http://www-03.ibm.com/systems/z/os/zos/downloads/logrec_viewer.html*.

IPCS VERBX

You can use the EREP program or the IPCS **VERBX LOGDATA** command to format software records recorded in **SYS1.LOGREC**. The IPCS diagnostic tool is provided in MVS to help diagnose software failures. Its facilities include formatting and analysis support for the traces and dumps produced by MVS, program products, and applications running on MVS.

MVS dumps fall into two categories: formatted dumps and unformatted dumps. You can use IPCS to format and analyze unformatted dumps. IPCS does not work with formatted dumps.

IPCS services provide a tool to format dumps and traces in both batch and online mode. Commands provided by the facility let you interrogate components to review storage locations or control blocks. The most common IPCS command is **VERBX** (Verb Exit), which formats data by product. DB2 dumps that use IPCS can be processed with the **VERBX DSNWDMP** IPCS command, which formats the DB2 dump data.

IPCS LOGDATA

When searching the data, you can use IPCS subcommands to pinpoint the problem (e.g., a wait, loop, or abend), or you might look for a CSECT name or component name that, through systems and messages, directs you to the problem.

Before you can use the data, you need to format the dump. The **VERBX LOGDATA** command formats **SYS1.LOGREC** records from the storage buffer. Input to IPCS is either an unformatted stand-alone or SVC dump. Be aware that it must be a complete dump.

LookAt

IBM's LookAt facility runs on VM, TSO, and Microsoft Windows and takes the user directly to an online reference that opens to the section addressing a subject message ID. For more information about LookAt, go to *http://www.ibm.com/eserver/zseries/zos/bkserv/lookat*.

DB2 Start-Up and Shutdown Messages

Console messages for the active address spaces provide an incredible amount information. You can view these messages using SDSF.

Prefixes on the messages indicate the component associated with the problem, such as **DFH** for CICS or **DSN** for DB2. Figure 2.20 shows an example of messages issued upon DB2 start-up, beginning with the job initiator message $HASP373.

```
$HASP373 xxxxMSTR STARTED
DSNZ002I - SUBSYS ssnm SYSTEM PARAMETERS
           LOAD MODULE NAME IS dsnzparm-name
DSNY001I - SUBSYSTEM STARTING
DSNJ127I - SYSTEM TIMESTAMP FOR BSDS= xx.xxx xx:xx:xx.x
DSNJ001I - csect CURRENT COPY n ACTIVE LOG DATA
           SET IS DSNAME=...,
           STARTRBA=...,ENDRBA=...
DSNJ099I - LOG RECORDING TO COMMENCE WITH
           STARTRBA = xxxxxxxxxxxx
$HASP373 xxxxDBM1 STARTED
DSNR001I - RESTART INITIATED
DSNR003I - RESTART...PRIOR CHECKPOINT RBA=xxxxxxxxxxxx
DSNR004I - RESTART...UR STATUS COUNTS...
           IN COMMIT=nnnn, INDOUBT=nnnn, INFLIGHT=nnnn,
.....more messages follow.....
```

Figure 2.20: DB2 start-up system messages

SYS1.DUMPxx

The **DUMP** command requests a system dump (SVC dump) of virtual storage. The data set can be a pre-allocated dump data set named **SYS1.DUMP*xx***, or you can automatically allocate a dump data set based on your installation's specified naming convention. As previously noted, different types of dumps are available to analyze problems. The processes to acquire these dumps are discussed in a series of IBM Redbooks called the *ABCs of z/OS System Programming*.

If you are looking at an SVC dump of a DB2 address space, examine the **LOGREC** symptom string to help determine the failing DB2 component.

MVS Tracing

There is only one way to activate a Generalized Trace Facility (GTF) trace: you must enter the **START GTF** command from the console that has master authority. You select your cataloged procedure or IBM-supplied procedure for GTF. The IBM GTF cataloged procedure in located **SYS1.PROCLIB**. It defines the GTF operation, how much storage you will need, the output destination, and the trace data sets.

A GTF trace shows the system processing events over time. This type of trace uses more CPU time than a system trace. The trace operates in its own address space as a system task.

DB2 Tracing

To run a DB2 trace, you can issue the **START TRACE** command from an MVS console, a DB2I command panel, a DSN session, a CICS or IMS terminal, or an IFI program. You must have the proper authorization of **SYSADM**, **SYSCTRL**, or **SYSOPR**. The command must include a trace type option of **PERFM**, **STAT**, **AUDIT**, **ACCTG**, or **MONITOR**. For example:

```
-START TRACE (PERFM) DEST(GTF) PLAN(plan_name, ... ) CLASS(class)
```

DB2 Trace Output

The **START TRACE** command's **DEST** option specifies where the trace output is to be recorded. You typically will request that the trace be sent either to GTF or to SMF.

The MVS GTF record identifier for DB2 trace records is X'0FB9'. The SMF record type depends on the IFCID record:

IFCID record	SMF record type
1 (system services statistics)	100
2 (database services statistics)	100
3 (agent accounting)	101
202 (dynamic system parameters)	100
230 (data sharing global statistics)	100
239 (agent accounting overflow)	101
All others	102

To trace all performance class records and write to GTF, you would start the trace as follows:

```
-START TRACE(PERFM) DEST(GTF) CLASS(*)
```

Chapter 6 provides more information about tracing.

Storage Management Subsystem

Disk storage has changed rapidly over the past few years, resulting in the delivery of new functions and improved performance. To keep pace and make use of these disk improvements, DB2 has undergone many changes.

Extended Address Volume (EAV)

An Extended Address Volume (EAV) supports 223 GB per volume on z/OS 1.10 and the IBM System Storage DS8000 to address the problem of running out of z/OS addressable disk storage due to the four-digit device number limit. An EAV is 65,536 cylinders or larger. Data sets on an EAV are eligible to have extents in the extended addressing space. This includes the VSAM data types for DB2, both SMS and non-SMS managed. EAV provides constraint relief for DB2 applications that use large VSAM data sets.

Parallel Access Volume

DB2 9 also continues to use Parallel Access Volume and Multiple Allegiance features of the IBM TotalStorage Enterprise Storage Server (ESS) and DS8000. PAV enables more than one I/O operation to be processed on a single logical 3390 volume, significantly reducing device queue delays. This is done by assigning multiple addresses to volumes. Implementing PAV devices is accomplished using static or dynamic alias management.

With this support, your careful data set placement methodology of the past can be replaced with the use of PAV and SMS group policies for data separation. PAV reduces I/O responses by reducing I/O supervisor queue (IOSQ) time, and SMS can be used to "spread" data for partitions across volumes. Together, the two reduce I/O contention.

In addition, the SMS groups provide effective organization of data for recoverability when you use the **BACKUP** and **RESTORE** system utilities. SMS grouping can also aid in the cloning of systems.

PAV requires a fiber channel connection (FICON) attachment feature. It is possible to vary the number of parallel accesses to a PAV.

DSVCI

DB2 8 introduced the ability to have control interval sizes that are the same size as the page being allocated. You enable this support through the online updatable DSNZPARM **DSVCI**. Making the CI size the same as the page size allows for data integrity because you can be sure that the whole row is intact when it is externalized.

Before this enhancement, there were some instances of data integrity exposures for 8 KB, 16 KB, and 32 KB objects when writing to disk at the extent boundary. The integrity exposure caused DB2 not to allow these objects to use VSAM multi-striping and required you to run a **SET LOG SUSPEND** command when doing split mirror design backups. In addition, concurrent copy required **SHRLEVEL REFERENCE** for 32 KB pages and did not allow **CHANGE**. With **DSVCI**, the integrity issue has been resolved, and these functions are now available for all page sizes.

SMS Storage

SMS storage lets the operating system take over storage tasks that we usually do manually. To implement SMS, you define a volume pooling structure made up of storage groups. Routines are executed by the system to control the allocation of the storage groups. Automatic class selection (ASC) routines define which data sets can be allocated in which storage group.

DFSMS implements the policies put in place regarding how hardware resources and space should be handled. Installation panel **DSNTIPA3** (Figure 2.21) lets you define the SMS data class, management class, and storage class associated with your data sets in DB2.

```
DSNTIPA3        INSTALL DB2 - DATA PARAMETERS PANEL 2
===> _

Check parameters and reenter to change:
 1  PERMANENT UNIT NAME    ==> 3390              Device type for MVS catalog
                                                 and partitioned data sets
 2  TEMPORARY UNIT NAME    ==> SYSDA             Device type for
                                                 temporary data sets

                                  ---------------- SMS -----------------
                              VOL/SER    DATA CLASS    MGMT CLASS    STOR CLASS
                              -------    ----------    ----------    ----------
 3  CLIST ALLOCATION       ==> DSNV01  ==>           ==>           ==>
 4  NON-VSAM DATA          ==> DSNV01  ==>           ==>           ==>
 5  VSAM CATALOG, DEFAULT, ==> DSNV02  ==>           ==>           ==>
    AND WORK FILE DATABASE
 6  DIRECTORY AND CATALOG  ==>         ==>           ==>           ==>
    DATA
 7  DIRECTORY AND CATALOG  ==>         ==>           ==>           ==>
    INDEXES
 8  LOG COPY 1, BSDS 2     ==>         ==>           ==>           ==>
 9  LOG COPY 2, BSDS 1     ==>         ==>           ==>           ==>
```

Figure 2.21: Installation panel DSNTIPA3 – SMS data parameters

SMS Classes and Groups

The constructs within the basic structure of SMS consist of classes and groups: data class, management class, storage class, storage group, aggregate group, tape library, optical library, and optical drives, as well as a base configuration for the systems and groups.

Data classes actually apply to non-SMS or SMS managed data sets. You can specify space parameters associated with your data sets.

Storage classes are only for SMS managed data sets, supplying information about dynamic cache management, concurrent copy, or sequential data set striping.

The management class applies to SMS data sets, which supply a list of migration information about backup and retention. The DFSMS Hierarchical Storage Management (DFSMShsm) tool uses these attributes for storage management.

Storage groups apply to SMS and relate to the physical storage for data sets and objects. The six types of storage groups are pool, object backup, tape, dummy, VIO, and object.

These classifications allow service levels for data sets to be assigned and permit the system to be managed automatically.

Parallel Sysplex and DB2

Parallel Sysplex is a clustering approach for S/390 systems. A group of IBM mainframes linked in this type of environment is called a system complex, or sysplex.

Applications running on more than one DB2 subsystem can write and read to the same databases concurrently and share the same DB2 catalog. Data sharing gives the ability to cluster together up to 32 DB2 subsystems to provide shared data, availability, and workload management and to take advantage of parallel sysplex on the zSeries and z/OS. The parallelism across participating DB2 members and the ability to add processors to scale vertically provides a near-linear support.

Coupling Facility

A coupling facility (CF) consists of one or more processors running specialized code that coordinates events across multiple members of the data-sharing group. Three structures make up the CF, and all three reside in one or more CFs:

- Group buffer pools
- Shared communications area (SCA)
- Lock structure

The group buffer pools contain data and index pages being accessed across the data-sharing group. The SCA is for recovery and startup across the group. The lock structure maintains the integrity of the data across members. Data sharing in DB2 is an option.

Large Mainframe Systems and Hardware Resources

The IBM z9 Integrated Information Processor (zIIP) is a specialty engine for the System z9 mainframe. The z/OS operating system manages and directs the work between general-purpose processors and the zIIPs. DB2 9 redirects more processing to the zIIP engine. The zIIP engine is designed so that a program can work with z/OS and have all or a portion of its enclave SRB work directed to the zIIP.

DB2 8 for z/OS was the first DB2 version to exploit the zIIP with System z9 109 and z/OS 1.6 or later. No changes are required for DB2 8 for z/OS applications to use the zIIP. The PTFs listed in info APAR II14219 provide more information about the requirements for DB2 8 and DB2 9 of DB2.

Workloads Benefiting from zIIP

Workloads in DB2 that benefit the most from the zIIP engines are business intelligence (BI), enterprise resource planning (ERP), customer relationship management (CRM), data warehousing, and DB2 utilities.

For data warehousing, those requests that use parallel queries, including star schemas, benefit from the offloading of work to the zIIP. The DB2 utility functions used to maintain index maintenance structures also benefit. Applications such as BI, CRM, and ERP and other applications that use DRDA over a TCP/IP connection (enclave SRBs, not stored procedures or user-defined functions) also benefit from the offloading of work to the zIIP engine.

Estimates show the following possible reductions in Class 1 CPU time with the zIIP:

- **REBUILD INDEX**: 5 to 20 percent
- **LOAD** or **REORG** of a partition with one index only: 10 to 20 percent
- **LOAD** or **REORG** of an entire table space: 10 to 20 percent
- **REBUILD INDEX** of a logical partition of a non-partitioning index: 40 percent
- **REORG INDEX**: 10 to 20 percent
- **LOAD** or **REORG** of a partition with more than one index: 30 to 60 percent
  - When **LOAD** or **REORG** is used with many partitions or indexes and CPU enclave SRB during index rebuild phase: less than 10 percent

Workload Manager

The Workload Manager component of z/OS implements dynamic workload management. WLM buffer pool management is also available in conversion mode.

In DB2 9, the maximum name length of the default WLM environment (defined in subsystem parameter **DSN6SYSP**, macro **WLMENV**) is increased from 18 to 32 characters.

WLM can dynamically redistribute or allocate server resources such as I/O, memory, and CPU across workloads (groupings of work) based on defined goals and resource demand within a z/OS image.

WLM manages workloads by assigning goal and work priorities based on your business requirements. WLM uses these goals to dynamically adjust access to storage and processor resources.

WLM uses several constructs in managing workloads. These include service definitions, service policies, service classes, report classes, and resource groups.

- *Service definitions* contain a set of classification rules that separate incoming work into distinct service classes and multiple service policies. A service definition deals with the management of work that needs to be identified and grouped into classes.
- A *service policy* is a named set of goals associated with the service classes. A policy applies to all work running in a sysplex. You can have one policy active at a time.
- *Service classes* are associated with a base goal.
- *Report classes* provide greater granularity than service classes. You can define up to 999 report classes.
- *Resource groups* control the CPU service units per second consumption of a set of classes. They define a minimum and maximum amount of service that the service classes should not exceed. Resource groups are not required.

You use the WLM ISPF application panels to define a service definition. Then, from the menus, you create your workloads, policies, resource groups, service classes, and other definitions.

Service Classes

Service classes are a named collection of work within a workload. They define the runtime requirements for that work. Assigning an importance to each service class gives the workloads within it a preference; this is the first step in separating work into distinct service classes. WLM examines all service classes every 10 seconds to determine which classes might need help with resources. Be careful not to over-allocate service classes, or WLM may need to make too many adjustments to your work assignments.

Each service class is associated with a goal and an importance to manage work. Goals for service classes can be defined as average response time, execution velocity (the acceptable amount of work delay), percentile response time (ending work within a certain time limit), or discretionary (when there is no specific goal definition or no specific importance for work in the service class).

An appropriate installation-defined service goal for the DB2 address spaces **MSTR**, **DBM1**, and **DIST** is a high velocity goal. Most of the DB2 thread work applies to the user goal. User work runs under separate goals for the enclave.

Work is classified into distinct service classes by subsystem type, which usually is related to an application. You define the service classes and goals in a service definition.

The two primary system service classes are **SYSTEM** and **SYSSTC**. Classify address spaces that support the system or its operation as service class **SYSSTC**, and classify all system-related address spaces as service class **SYSTEM**.

A third class, named **SYSOTHER**, is predefined in the system. You should classify all work so that **SYSOTHER** is not used.

Enclave

We previously discussed enclaves as they relate to DDF. DB2,WebSphere, and other major components and middleware applications on z/OS also exploit enclave

management. In enclave management, all functions are given to the end user in defining goals, monitoring execution, and observing the progress of work in the system. WLM can manage the enclave directly, independently of the address spaces where the execution units belong.

Stored Procedures

The stored procedure type dictates the process used to create a stored procedure. DB2 for z/OS supports the following types of stored procedures:

- *Native SQL procedure*: The body of this type is written in SQL, and no C program is generated by DB2. Native SQL procedures are WLM-managed.
- *External SQL procedure*: The body of an external SQL procedure is written in SQL, and DB2 does generate an associated C program.
- *External stored procedure*: These are written in a host language.

In addition, Java stored procedures and user-defined functions can contain SQL statements. A client program written in any supported language invokes the Java stored procedure. Java Database Connectivity (JDBC) type 4 connections are recommended to engage the zIIP engine. JDBC requests are sent to DB2 using JDBC driver type 2 or driver type 4.

Administrative Tasks

You can use the administrative task scheduler to run tasks, which can be JCL jobs or stored procedures. Once defined, these tasks are stored in two redundant task lists: **SYSIBM.ADMIN_TASKS** and a VSAM data set defined in **ADMTDD1**.

You manage the scheduler task list using DB2 stored procedures that add or remove tasks (**ADMIN_TASK_ADD** and **ADMIN_TASK_REMOVE**). DB2 also provides user-defined functions that help you monitor the task list (**ADMINT_TASK_LIST**) and the status of tasks in DB2 **(ADMIN_TASK_STATUS**).

The scheduler is a started task named in DB2, and it starts when DB2 is brought up. The scheduler's name is **ADMTPROC**.

Because the scheduler manages two redundant task lists, it can continue working even if one of the task lists becomes damaged or unavailable. If a task list is corrupted, you will receive the message DSNA679I. You can recover the VSAM data set for the **ADMIN_TASKS** task list. As soon as this data set is available, the scheduler will perform an autonomic recovering of the contents.

z9 Processor and Specialty Engines

IBM's zIIP specialty engine is a less expensive alternative to CPU costs. Workloads are eligible to be offloaded to zIIPs if they run under a z/OS enclave SRB. Workload Manager in z/OS manages the workloads that are eligible for offloading. WLM directs the work between the zIIP and the central processor without any changes to your programs.

zIIPs and Workloads

Workloads that can offload work to the zIIP include native stored procedures, some portions of star-schema parallel queries, and parts of the index build and maintenance process of the **REORG**, **LOAD**, and **REBUILD INDEX** utilities. IFI includes DB2 support to reflect zIIP- and CPU-related performance information. WLM algorithms verify the buffer pool size, adjust it (if necessary) to prevent out-of-storage conditions, and try first to take storage from other buffer pools to make the adjustment.

WLM Services and zIIP

Stored procedures, DDF, and buffer pools are all managed by WLM. SMF records for enclave reporting are types 30 and 72:

- Type 30: The record contains the resources consumed at the address space level.
- Type 72: Have a service class record and a report class if it is in the service policy. So, with WLM, you can separate the enclaves into different classes, either report or service, to get a better idea how the work is being done in DDF.

Activating zIIP for DB2

Hardware zIIP engines z9 BC and z9 EC models have the zIIP engines available. No further action is required to implement zIIP use once the hardware and software are installed.

RMF

The Resource Measurement Facility Workload Activity report summarizes resource consumption by workload and by service class periods within workloads. The **PROJECTCPU=YES** option enables RMF to monitor DB2 to assess how zIIP consumption would be used. You can run this projection capability at any time.

The **PROJECTCPU=YES** option enables z/OS to collect zIIP usage data as though a zIIP was configured when the target workload is being run. When you use this parameter, RMF and SMF show the calculated zIIP time, which is used to gain an accurate zIIP projection.

Practice Questions

Question 1

Name the three major groups of subcomponent code structure in DB2.

- ❍ A. SSAS, DBAS, DDF
- ❍ B. SSAS, DBAS, IRLM
- ❍ C. SSAS, DBAS, SPAS
- ❍ D. SSAS, DBAS, WLM

Question 2

At DB2 address space termination, what happens to the DB2 shared memory object area?

- ❍ A. It continues to run, but the VSO is deleted.
- ❍ B. It is freed, and interest in the VSO is deleted.
- ❍ C. It will continue to be available on the next start-up of DB2.
- ❍ D. It will not be affected.

Question 3

An inactive connection in DB2 was previously called:

- ❍ A. Type 2 inactive thread
- ❍ B. Inactive DBAT
- ❍ C. Active DBAT
- ❍ D. Type 2 active thread

Question 4

When trying to establish the total number of threads that can access data in DB2, what should you do?

- ❍ A. Add **MAXDBAT** and **CTHREAD**.
- ❍ B. Subtract **CTHREAD** from **MAXDBAT**.
- ❍ C. Divide **MAXDBAT** by **CTHREAD**.
- ❍ D. Check **MAXDBAT** only.

Question 5

Native stored procedures, if invoked from DRDA TCP/IP connections to DB2, may:

- ❍ A. Be eligible for zap processing
- ❍ B. Be eligible for zIIP processing
- ❍ C. Be eligible for zIIP and zap processing
- ❍ D. Are not eligible for either zIIP or zap processing

Question 6

What are the associated pools in the EDM pool (RDS)?

- ❍ A. **EDMDBDC, EDMSTMTC, EDM_SKELETON_POOL**
- ❍ B. **EDMDBDC, EDMSTMTC**
- ❍ C. **EDMDBDC, EDM_SKELETON_POOL**
- ❍ D. **EDMDBDC**

Question 7

The DSNZPARM parameter **MAXKEEPD** is used to:

- ❍ A. Limit the number of threads
- ❍ B. Limit the number of dynamic statements held in the cache
- ❍ C. Limit the number of statistics kept on dynamic cache
- ❍ D. Limit the number of threads to keep

Question 8

Which DSNZPARM defines the number of RID blocks in the RID pool storage?

- ❍ A. **CONDBAT**
- ❍ B. **URLGWTH**
- ❍ C. **NUMTCB**
- ❍ D. **MAXRBLK**

Question 9

What buffer sizes are supported for the **DSNDB07** database?

- ❍ A. 4 KB, 8 KB, 16 KB, 32 KB
- ❍ B. 4 KB, 32 KB
- ❍ C. 4 KB, 16 KB
- ❍ D. 4 KB, 8 KB

Question 10

What are the types of dumps in z/OS?

- ❍ A. Transaction, abend, stand-alone
- ❍ B. SVC, transaction, abend, stand-alone, snap
- ❍ C. Stand-alone and snap
- ❍ D. Stand-alone, abend, snap, dump