目录

第一部分 数据库体系架构

第1章 DM 逻辑结构概述

1.1 数据库和实例

在 DM7 之前版本的 DM 数据库中，“数据库”和“实例”这两个术语经常可以互相替换，意义也很相近。在新版本 DM7 数据库中，“数据库”和“实例”这两个概念之间有着很大的差别，甚至可以说它们是两个完全不同的实体。

1.1.1 数据库

在有些情况下，数据库的概念包含的内容会很广泛。如在单独提到 DM 数据库时，可能指的是 DM 数据库产品，也有可能是正在运行的 DM 数据库实例，还可能是 DM 数据库运行中所需的一系列物理文件的集合等。但是，当同时出现 DM 数据库和实例时，DM 数据库指的是磁盘上存放在 DM 数据库中的数据的集合，一般包括：数据文件、日志文件、控制文件以及临时数据文件等。

1.1.2 实例

实例一般是由一组正在运行的 DM 后台进程/线程以及一个大型的共享内存组成。简单来说，实例就是操作 DM 数据库的一种手段，是用来访问数据库的内存结构以及后台进程的集合。

DM 数据库存储在服务器的磁盘上，而 DM 实例则存储于服务器的内存中。通过运行 DM 实例，可以操作 DM 数据库中的内容。在任何时候，一个实例只能与一个数据库进行关联（装载、打开或者挂起数据库）。在大多数情况下，一个数据库也只有一个实例对其进行操作。但是在 DM 共享存储集群（DMRAC）中，多个实例可以同时装载并打开一个数据库（位于一组由多台服务器共享的物理磁盘上）。此时，我们可以同时从多台不同的计算机访问这个数据库。

1.2 DM 逻辑存储结构

DM 数据库为数据库中的所有对象分配逻辑空间，并存放在数据文件中。在 DM 数据库内部，所有的数据文件组合在一起被划分到一个或者多个表空间中，所有的数据库内部对象都存放在这些表空间中。同时，表空间被进一步划分为段、簇和页（也称块）。通过这种细分，可以使得 DM 数据库能够更加高效地控制磁盘空间的利用率。图 1.1 显示了这些数据结构之间的关系。

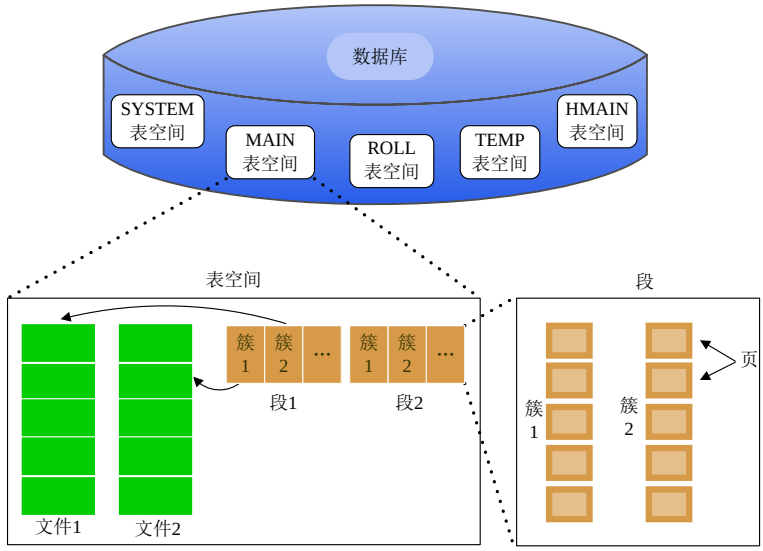


图 1.1 显示表空间、数据文件、段、簇、页的关系

可以看出，在 DM7 中存储的层次结构如下：

1. 数据库由一个或多个表空间组成；
2. 每个表空间由一个或多个数据文件组成；
3. 每个数据文件由一个或多个簇组成；
4. 段是簇的上级逻辑单元，一个段可以跨多个数据文件；
5. 簇由磁盘上连续的页组成，一个簇总是在一个数据文件中；
6. 页是数据库中最小的分配单元，也是数据库中使用的最小的 IO 单元。

1.2.1 表空间

在 DM 数据库中，表空间由一个或者多个数据文件组成。DM 数据库中的所有对象在逻辑上都存放在表空间中，而物理上都存储在所属表空间的数据文件中。

在创建 DM 数据库时，会自动创建 5 个表空间：SYSTEM 表空间、ROLL 表空间、MAIN 表空间、TEMP 表空间和 HMAIN 表空间。

1. SYSTEM 表空间存放了有关 DM 数据库的字典信息，用户不能在 SYSTEM 表空间创建表和索引。
2. ROLL 表空间完全由 DM 数据库自动维护，用户无需干预。该表空间用来存放事务运行过程中执行 DML 操作之前的值，从而为访问该表的其他用户提供表数据的读一致性视图。
3. MAIN 表空间在初始化库的时候，就会自动创建一个大小为 128M 的数据文件 MAIN.DBF。在创建用户时，如果没有指定默认表空间，则系统自动指定 MAIN 表空间为用户默认的表空间。
4. TEMP 表空间完全由 DM 数据库自动维护。当用户的 SQL 语句需要磁盘空间来完成某个操作时，DM 数据库会从 TEMP 表空间分配临时段。如创建索引、无法在内存中完成的排序操作、SQL 语句中间结果集以及用户创建的临时表等都会使用到 TEMP 表空间。
5. HMAIN 表空间属于 HTS 表空间，完全由 DM 数据库自动维护，用户无需干涉。当用户在创建 HUGE 表时，未指定 HTS 表空间的情况下，充当默认 HTS 表空间。

每一个用户都有一个默认的表空间。对于 SYS、SYSSSO、SYSAUDITOR 系统用户，默认的用户表空间是 SYSTEM，SYSDBA 的默认表空间为 MAIN，新创建的用户如果没有指定默认表空间，则系统自动指定 MAIN 表空间为用户默认的表空间。如果用户在创建表的时候，

指定了存储表空间 A，并且和当前用户的默认表空间 B 不一致时，表存储在用户指定的表空间 A 中，并且默认情况下，在这张表上面建立的索引也将存储在 A 中，但是用户的默认表空间是不变的，仍为 B。

一般情况下，建议用户自己创建一个表空间来存放业务数据，或者将数据存放在默认的用户表空间 MAIN 中。

用户可以通过执行如下语句来查看表空间相关信息。

SYSTEM、ROLL、MAIN 和 TEMP 表空间查看语句：

```
SELECT * FROM V$TABLESPACE;
```

HMAIN 表空间查看语句：

```
SELECT * FROM V$HUGE_TABLESPACE;
```

结果集中各字段的含义请参考附录部分动态性能视图章节中对 V$TABLESPACE 的介绍，关于表空间该如何使用，请参考第 15 章。

1.2.2 记录

数据库表中的每一行是一条记录。在 DM 中，除了 Huge 表，其他的表都是在数据页中按记录存储数据的。也就是说，记录是存储在数据页中的，记录并不是 DM 数据库的存储单位，页才是。由于记录不能跨页存储，这样记录的长度就受到数据页大小的限制。数据页中还包含了页头控制信息等空间，因此 DM 规定每条记录的总长度不能超过页面大小的一半。

1.2.3 页

数据页（也称数据块）是 DM 数据库中最小的数据存储单元。页的大小对应物理存储空间上特定数量的存储字节，在 DM 数据库中，页大小可以为 4KB、8KB、16KB 或者 32KB，用户在创建数据库时可以指定，默认大小为 8KB，一旦创建好了数据库，则在该库的整个生命周期内，页大小都不能够改变。图 1.2 显示了 DM 数据库页的典型格式。

图 1.2 DM 数据页的组成

页头控制信息包含了关于页类型、页地址等信息。页的中部存放数据，为了更好地利用数据页，在数据页的尾部专门留出一部分空间用于存放行偏移数组，行偏移数组用于标识页上的空间占用情况以便管理数据页自身的空间。

在绝大多数情况下，用户都无需干预 DM 数据库对数据页的管理。但是 DM 数据库还是提供了选项供用户选择，使得在某些情况下能够为用户提供更佳的数据处理性能。

FILLFACTOR 是 DM 数据库提供的一个与性能有关的数据页级存储参数，它指定一个数据页初始化后插入数据时最大可以使用空间的百分比（100），该值在创建表/索引时可以指定。设置 FILLFACTOR 参数的值，是为了指定数据页中的可用空间百分比（FILLFACTOR）和可扩展空间百分比（100-FILLFACTOR）。可用空间用来执行更多的 INSERT 操作，可扩展空间用来为数据页保留一定的空间，以防止在今后的更新操作中增加列或者修改变长列的

长度时，引起数据页的频繁分裂。当插入的数据占据的数据页空间百分比低于 FILLFACTOR 时，允许数据插入该页，否则将当前数据页中的数据分为两部分，一部分保留在当前数据页中，另一部分存入一个新页中。

对于 DBA 来说，使用 FILLFACTOR 时应该在空间和性能之间进行权衡。为了充分利用空间，用户可以设置一个很高的 FILLFACTOR 值，如 100，但是这可能会导致在后续更新数据时，频繁引起页分裂，而导致需要大量的 I/O 操作。为了提高更新数据的性能，可以设置一个相对较低（但不是过低）的 FILLFACTOR 值，使得后续执行更新操作时，可以尽量避免数据页的分裂，提升 I/O 性能，不过这是以牺牲空间利用率来换取性能的提高。

1.2.4 簇

簇是数据页的上级逻辑单元，由同一个数据文件中 16 个或 32 个连续的数据页组成。在 DM 数据库中，簇的大小由用户在创建数据库时指定，默认大小为 16。假定某个数据文件大小为 32MB，页大小为 8KB，则共有 32MB/8KB/16=256 个簇，每个簇的大小为 8K*16=128K。和数据页的大小一样，一旦创建好数据库，此后该数据库的簇的大小就不能够改变。

1． 分配数据簇

当创建一个表/索引的时候，DM 为表/索引的数据段分配至少一个簇，同时数据库会自动生成对应数量的空闲数据页，供后续操作使用。如果初始分配的簇中所有数据页都已经用完，或者新插入/更新数据需要更多的空间，DM 数据库将自动分配新的簇。在缺省情况下，DM 数据库在创建表/索引时，初始分配 1 个簇，当初始分配的空间用完时，DM 数据库会自动扩展。

当 DM 数据库的表空间为新的簇分配空闲空间时，首先在表空间按文件从小到大的顺序在各个数据文件中查找可用的空闲簇，找到后进行分配；如果各数据文件都没有空闲簇，则在各数据文件中查找空闲空间足够的，将需要的空间先进行格式化，然后进行分配；如果各文件的空闲空间也不够，则选一个数据文件进行扩充。

2． 释放数据簇

对于用户数据表空间，在用户将一个数据段对应的表/索引对象 DROP 之前，该表对应的数据段会保留至少 1 个簇不被回收到表空间中。在删除表/索引对象中的记录的时候，DM 数据库通过修改数据文件中的位图来释放簇，释放后的簇被视为空闲簇，可以供其他对象使用。当用户删除了表中所有记录时，DM 数据库仍然会为该表保留 1-2 个簇供后续使用。若用户使用 DROP 语句来删除表/索引对象，则此表/索引对应的段以及段中包含的簇全部收回，并供存储于此表空间的其他模式对象使用。

对于临时表空间，DM 数据库会自动释放在执行 SQL 过程中产生的临时段，并将属于此临时段的簇空间还给临时表空间。需要注意的是，临时表空间文件在磁盘所占大小并不会因此而缩减，用户可以通过系统函数 SF_RESET_TEMP_TS 来进行磁盘空间的清理。

对于回滚表空间，DM 数据库将定期检查回滚段，并确定是否需要从回滚段中释放一个或多个簇。

1.2.5 段

段是簇的上级逻辑分区单元，它由一组簇组成。在同一个表空间中，段可以包含来自不同文件的簇，即一个段可以跨越不同的文件。而一个簇以及该簇所包含的数据页则只能来自一个文件，是连续的 16 或者 32 个数据页。由于簇的数量是按需分配的，数据段中的不同簇在磁盘上不一定连续。

1． 数据段

段可以被定义成特定对象的数据结构，如表数据段或索引数据段。表中的数据以表数据段结构存储，索引中的数据以索引数据段结构存储。DM 以簇为单位给每个数据段分配空间，当数据段的簇空间用完时，DM 数据库就给该段重新分配簇，段的分配和释放完全由 DM 数据库自动完成，可以在创建表/索引时设置存储参数来决定数据段的簇如何分配。

当用户使用 CREATE 语句创建表/索引时，DM 创建相应的数据段。表/索引的存储参数用来决定对应数据段的簇如何被分配，这些参数将会影响与对象相关的数据段的存储与访问效率。对于分区表，每个分区使用单独的数据段来容纳所有数据，对于分区表上的非分区索引，使用一个索引数据段来容纳所有数据，而对于分区索引，每个分区使用一个单独索引数据段来容纳其数据。表的数据段和与其相关的索引段不一定要存储在同一表空间中，用户可以在创建表和索引时，指定不同的表空间存储参数。

2． 临时段

在 DM 数据库中，所有的临时段都创建在临时表空间中，这样可以分流磁盘设备的 I/O，也可以减少由于在 SYSTEM 或其他表空间内频繁创建临时数据段而造成的碎片。

当处理一个查询时，经常需要为 SQL 语句的解析与执行的中间结果准备临时空间。DM 数据库会自动地分配临时段的磁盘空间。例如，DM 在进行排序操作时就可能需要使用临时段，当排序操作可以在内存中执行，或设法利用索引就可以执行时，就不必创建临时段。对于临时表及其索引，DM 数据库也会为它们分配临时段。

临时段的分配和释放完全由系统自动控制，用户不能手工进行干预。

3． 回滚段

DM 数据库在回滚表空间的回滚段中保存了用于恢复数据库操作的信息。对于未提交事务，当执行回滚语句时，回滚记录被用来做回滚变更。在数据库恢复阶段，回滚记录被用来做任何未提交变更的回滚。在多个并发事务运行期间，回滚段还为用户提供读一致性，所有正在读取受影响行的用户将不会看到行中的任何变动，直到他们事务提交后发出新的查询。

DM 数据库提供了全自动回滚管理机制来管理回滚信息和回滚空间，自动回滚管理消除了管理回滚段的复杂性。此外，系统将尽可能保存回滚信息，来满足用户查询回滚信息的需要。事务被提交后，回滚数据不能再回滚或者恢复，但是从数据读一致性的角度出发，长时间运行查询可能需要这些早期的回滚信息来生成早期的数据页镜像，基于此，数据库需要尽可能长时间的保存回滚信息。DM 数据库会收集回滚信息的使用情况，并根据统计结果对回滚信息保存周期进行调整，数据库将回滚信息保存周期设为比系统中活动的最长的查询时间稍长。

第2章 DM 物理存储结构

DM 数据库使用了磁盘上大量的物理存储结构来保存和管理用户数据。典型的物理存储结构包括：用于进行功能设置的配置文件；用于记录文件分布的控制文件；用于保存用户实际数据的数据文件、重做日志文件、归档日志文件、备份文件；用来进行问题跟踪的跟踪日志文件等，如图 2.1 所示。

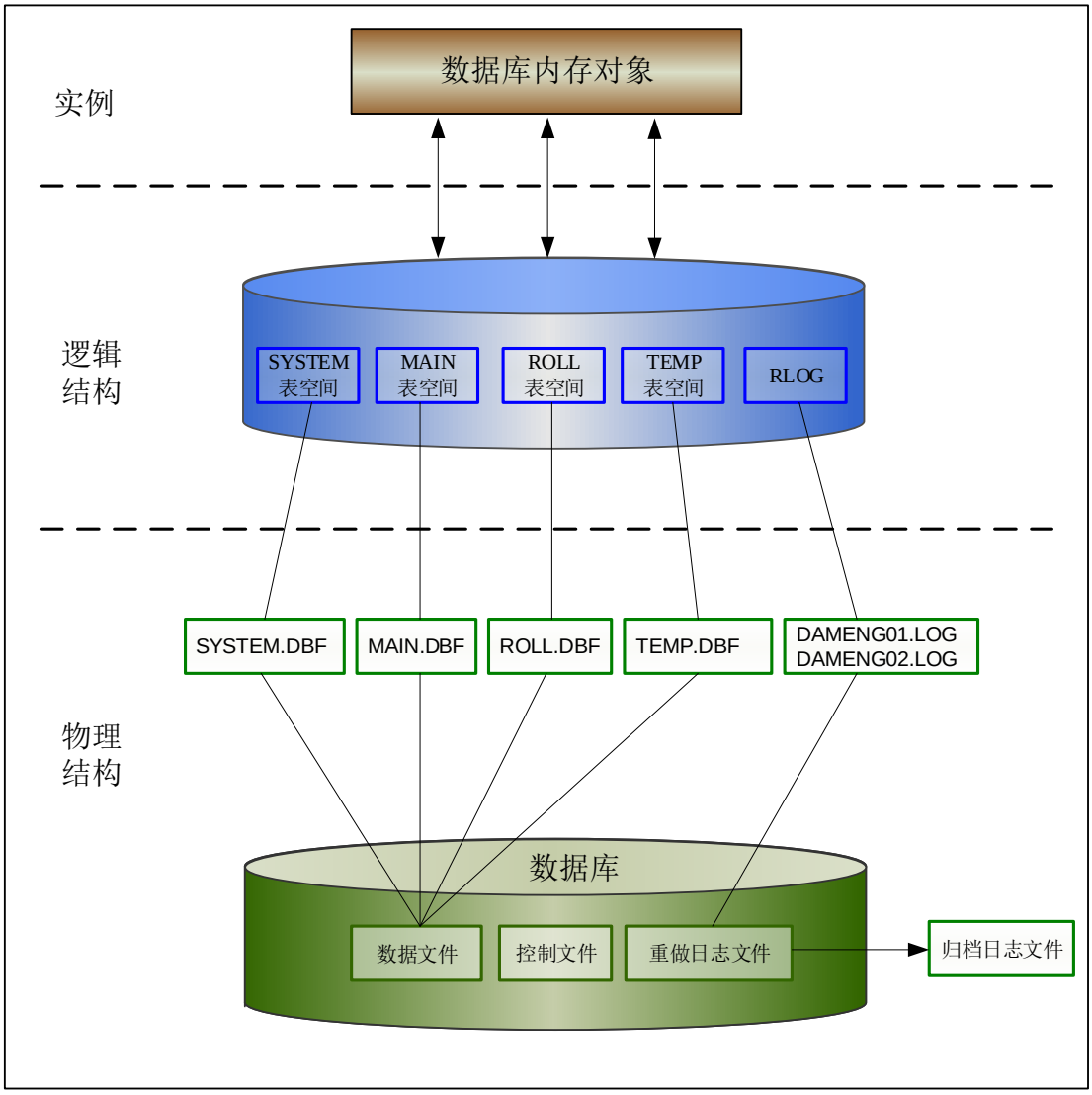


图 2.1 DM 物理存储结构示意图

2.1 配置文件

配置文件是 DM 数据库用来设置功能选项的一些文本文件的集合，配置文件以 ini 为扩展名，它们具有固定的格式，用户可以通过修改其中的某些参数取值来达成如下两个方面的目标：

1. 启用/禁用特定功能项；
2. 针对当前系统运行环境设置更优的参数值以提升系统性能。

2.1.1 DM 数据库服务配置

1) dm.ini

每创建一个 DM 数据库，就会自动生成 dm.ini 文件。dm.ini 是 DM 数据库启动所必须的配置文件，通过配置该文件可以设置 DM 数据库服务器的各种功能和性能选项，主要的配置内容见表 2.1。

参数属性分为三种：静态、动态和手动。

静态，可以被动态修改，修改后重启服务器才能生效。

动态，可以被动态修改，修改后即时生效。动态参数又分为会话级和系统级两种。会话级参数被修改后，新参数值只会影响新创建的会话，之前创建的会话不受影响；系统级参数的修改则会影响所有的会话。

手动，不能被动态修改，必须手动修改 dm.ini 参数文件，然后重启才能生效。

动态修改是指 DBA 用户可以在数据库服务器运行期间，通过调用系统过程 SP_SET_PARA_VALUE()、SP_SET_PARA_DOUBLE_VALUE()和 SP_SET_PARA_STRING_VALUE()对参数值进行修改。

表 2.1 dm.ini 配置项

参数名	缺省值	属性	说明
控制文件相关参数（注意：本类参数不建议用户修改）			
CTL_PATH	安装时指定	手动	控制文件路径
CTL_BAK_PATH	安装时指定	手动	控制文件备份路径，缺省路径为“SYSTEM_PATH/CTL_BAK”，在初始化库或没有配置该项时均指定为缺省路径。 备份文件命名格式： “DM_年月日时分秒_毫秒.CTL” 备份文件在初始化库和每次修改 DM.CTL 控制文件后生成
CTL_BAK_NUM	10	手动	控制文件备份个数限制，取值 1~100，在此限制之外，会再多保留一个备份文件，在生成新的备份文件时，如果当前已存在的备份文件个数大于指定值，则自动删除创建时间最早的备份文件，小于或等于指定值的情况下，不会有删除操作，缺省值为 10
SYSTEM_PATH	安装时指定	手动	系统库目录
CONFIG_PATH	安装时指定	手动	指定 DMSERVER 所读取的配置文件（DMMAL.INI, DMARCH.INI, DMTIMER.INI 等）的路径。缺省使用 SYSTEM_PATH 路径。不允许指定 ASM 目录
TEMP_PATH	安装时指定	手动	临时库文件路径
BAK_PATH	安装时指定	手动	备份路径
BAK_POLICY	0	手动	备份还原版本策略。取值 0、1 或 2。缺省为 0。为 0 表示同时支持 BAK1 和 BAK2 版本；为 1 时只能使用 BAK1 版本；为 2 表示只支持 BAK2 版本。BAK1 为备份还原的老版本，BAK2 为备份还原的新版本。

实例名			
INSTANCE_NAME	DMSERVER	手动	实例名（一般情况下，长度不超过 128 个字符；但是在数据守护、DM 共享存储中，长度不超过 16 个字符）
内存相关参数			
MAX_OS_MEMORY	95	静态	DM 服务器能使用的最大内存占操作系统物理内存与虚拟内存总和的百分比，有效值范围（40~100）。当取值 100 时，服务器不进行内存的检查 注：对于 32 位版本的 DM 服务器，虚拟内存最大为 2G
MEMORY_POOL	200	静态	共享内存池大小，以兆为单位。共享内存池是由DM管理的内存。有效值范围：32位平台为（64~2000），64位平台为（64~67108864）
MEMORY_TARGET	0	静态	共享内存池在扩充到此大小以上后，空闲时收缩回此指定大小，以M为单位，有效值范围：32 位平台为（0~2000），64 位平台为（0~67108864），0表示不限制
MEMORY_EXTENT_SIZE	1	静态	共享内存池每次扩充的大小，以 M 为单位，有效值范围（1~10240）
MEMORY_LEAK_CHECK	0	动态，系统级	是否开启对所有内存池的泄漏检查，0：不开启；1：开启
MEMORY_MAGIC_CHECK	2	静态	是否开启对所有内存池的校验。0：不开启；1：开启校验，校验码基于分配出的块地址计算，在被分配空间的头部和尾部写入校验码；2：增强校验，在 1 的基础上，如果是内存池分配的，则对尾部未使用空间也计算校验码，写入未使用空间的头部
MEMORY_BAK_POOL	4	静态	系统备份内存池大小，以兆为单位。系统备份内存池是由 DM 管理的内存。有效值范围（2~10000）
HUGE_MEMORY_THRESHOLD	0	静态	设置超过多大的常规内存分配优先从 HUGE_BUFFER 走，以 KB 为单位。有效值 0~1M。0 表示不从 HUGE_BUFFER 分配
HUGE_MEMORY_PERCENTAGE	50	静态	指示 HUGE_BUFFER 中可以借用作常规内存分配的空间百分比，有效值为 0~100。
HUGE_BUFFER	8	静态	HUGE 表使用的缓冲区大小，以兆为单位。有效值范围（8~1048576）
BUFFER	100	静态	系统缓冲区大小，以兆为单位。推荐值：系统缓冲区大小为可用物理内存的 60%～80%。有效值范围（8~1048576）
BUFFER_POOLS	19	静态	BUFFER 系统分区数，每个 BUFFER 分区的大小为 BUFFER/BUFFER_POOLS。有效值范围（1~512）
FAST_POOL_PAGES	3000	静态	快速缓冲区页数。有效值范围（0~99999）。FAST_POOL_PAGES 的值最多不能超过 BUFFER 总页数的一半，如果超过，系统会自

			动调整为 BUFFER 总页数的一半
KEEP	8	静态	KEEP 缓冲区大小，以兆为单位。有效值范围（8~1048576）
RECYCLE	64	静态	RECYCLE 缓冲区大小，以兆为单位。有效值范围（8~1048576）
RECYCLE_POOLS	19	静态	RECYCLE 缓冲区分区数，每个 RECYCLE 分区的大小为 RECYCLE/RECYCLE_POOLS。有效值范围（1~512）
MULTI_PAGE_GET_NUM	1	静态	缓冲区最多一次读取的页面数。有效值范围（1~128） 注：当数据库加密和 SSD_BUF_SIZE>0 时不支持多页读取，此时 dm.ini 中此参数值无效
MAX_BUFFER	100	静态	系统最大缓冲区大小，以兆为单位。采用动态缓冲区时，可以扩展到的最大缓冲区页数（>=BUFFER）。有效值范围（8~1048576）
SORT_BUF_SIZE	2	动态，会话级	排序缓存区最大值，以 M 为单位。有效值范围（1~2048）
HAGR_HASH_SIZE	100000	动态，会话级	HAGR 操作时，建立 HASH 表的桶个数。有效值范围（10000~100000000）
MAL_LEAK_CHECK	0	动态，系统级	是否打开 MAL 内存泄露检查。0：关闭；1：打开
HJ_BUF_GLOBAL_SIZE	500	动态，系统级	HASH 连接操作符的数据总缓存大小（>= HJ_BUF_SIZE），系统级参数，以兆为单位。有效值范围(10~500000)
HJ_BUF_SIZE	50	动态，会话级	单个 HASH 连接操作符的数据总缓存大小，以兆为单位。有效值范围（2~100000）
HJ_BLK_SIZE	1	动态，会话级	HASH 连接操作符每次分配缓存（BLK）大小，以兆为单位，必须小于 HJ_BUF_SIZE。有效值范围（1~50）
HAGR_BUF_GLOBAL_SIZE	500	动态，系统级	HAGR、DIST、集合操作、SPL2、NTTS2 以及 HTAB 操作符的数据总缓存大小（>= HAGR_BUF_SIZE），系统级参数，以兆为单位。有效值范围(10~1000000)
HAGR_BUF_SIZE	50	动态，会话级	单个 HAGR、DIST、集合操作、SPL2、NTTS2 以及 HTAB 操作符的数据总缓存大小，以兆为单位。有效值范围（2~500000）。 如果 HAGR_BUF_SIZE 设置的值满足范围且大于 HAGR_BUF_GLOBAL_SIZE，那么会在 HAGR_BUF_GLOBAL_SIZE/2 和 500000 两个值中，选出较小的那个，作为新的 HAGR_BUF_SIZE 值。
HAGR_BLK_SIZE	1	动态，会话级	HAGR、DIST、集合操作、SPL2、NTTS2 以及 HTAB 操作符每次分配缓存（BLK）大小，以兆为单位，必须小于 HAGR_BUF_SIZE。有效值范围（1~50）
MTAB_MEM_SIZE	8	静态	MTAB 缓存 BDTA 占用内存空间的大小，以 KB 为单位，有效值范围（1~1048576）

FTAB_MEM_SIZE	0	静态	FTAB 缓存 BDTA 占用内存空间的大小，以 KB 为单位。取值范围为(0~64 * 1024)。0 表示使用 MTAB，大于 0 时才使用 FTAB。当取值小于 32 时，FTAB_MEM_SIZE 均使用 32
MMT_SIZE	0	动态，会话级	是否使用 MMT。0：不启用；其他有效值：启用，并确定单个映射文件大小。有效值范围（0~64），单位 MB
MMT_GLOBAL_SIZE	4000	动态，系统级	系统总共使用 MMT 的文件总大小，单位 MB，有效值范围（10~1000000），仅在 MMT_SIZE 大于 0 时有效
MMT_FLAG	1	动态，会话级	MMT 存储数据方式。1：按页存储；2：BDTA 存储。仅在 MMT_SIZE 大于 0 时有效
DICT_BUF_SIZE	5	静态	字典缓冲区大小，以兆为单位，有效值范围（1~2048）。单位：MB
HFS_CACHE_SIZE	160	动态，系统级	HUGE 表 I/U/D 时 HDTA_BUFFER 缓存池大小，单位为 MB。有效值范围（160~ 2000）
VM_STACK_SIZE	256	静态	系统执行时虚拟机堆栈大小，单位为 K，堆栈的空间是从操作系统中申请的，有效值范围（64~256*1024）
VM_POOL_SIZE	64	静态	系统执行时虚拟机内存池大小，在执行过程中用到的内存大部分是从这里申请的，它的空间是从操作系统中直接申请的，有效值范围（32~1024*1024）
VM_POOL_TARGET	32768	静态	虚拟机内存池能扩充到的最大大小，以 KB 为单位，有效值范围（0~10*1024*1024），0 表示不限制
SESS_POOL_SIZE	64	动态，系统级	会话缓冲区大小，以 KB 为单位，有效值范围（16~1024*1024）。若所申请的内存超过实际能申请的大小，则系统将按 16KB 大小重新申请
SESS_POOL_TARGET	32768	动态，系统级	会话缓冲区能扩充到的最大大小，以 KB 为单位，有效值范围（0~10*1024*1024），0 表示不限制
RT_HEAP_TARGET	8192	动态，系统级	会话上用于动态对象存储的 RT_HEAP 最大可扩展到的大小，以 K 为单位，有效值范围（8192~10*1024*1024）
VM_MEM_HEAP	0	静态	VM 是否使用 HEAP 分配内存。1：是，0：否
VM_SQL_TREE_CACHE_SIZE	0	动态，系统级	SQL 运行树缓冲区大小，以 MB 为单位，设置为 0 则不缓存。有效值范围(0~1024*512)
RFIL_RECV_BUF_SIZE	16	静态	控制服务器启动时，进行 REDO 操作过程中，REDO 日志文件恢复时 BUFFER 的大小，以 MB 为单位，有效值范围（16~4000）
N_MEM_POOLS	1	静态	内存池的数量，有效值范围（1~128）
COLDATA_POOL_SIZE	0	动态，系统级	COLDATA 池的大小，以 M 为单位
HAGR_DISTINCT_HASH_TABLE_SIZE	10000	动态，会话级	分组 DISTINCT 操作中 HASH 表的大小(桶数)。取值范围为(10000~100000000)

CNNTB_HASH_TABLE_SIZE	100	动态，会话级	指定 CNNTB 操作符中创建 HASH 表的大小。有效值范围（100~100000000）
GLOBAL_RTREE_BUF_SIZE	100	动态，会话级	R 树全局缓冲区大小，以 MB 为单位
SINGLE_RTREE_BUF_SIZE	10	动态，会话级	单个 R 树的缓冲区大小，以 MB 为单位
SSD_BUF_SIZE	0	静态	指定 SSD 缓冲区大小，以兆为单位。取值范围（0~ 4294967294），0 表示关闭
SSD_FILE_PATH	需要时指定	静态	SSD 缓冲区文件所在的文件夹路径，管理员保证其在 SSD 分区上
SSD_REF_BUF_SIZE	80	静态	SSD 缓冲专用 BUF 大小，SSD_BUF_SIZE 不为 0 时有效。以兆为单位，有效值范围（20~4096）
SSD_FLUSH_INTERVAL	10	静态	SSD 缓冲刷盘轮询间隔，以 MS 为单位，取值范围（0~1000）
SSD_FLUSH_STEPS	100	静态	SSD 缓存刷盘向前页数，有效值范围（10~100000）
MEMORY_LEAK_CHECK	0	动态，系统级	是否开启内存泄漏检测。0：否；1：是，此时系统对每一次内存分配都登记到动态视图 V$MEM_REGINFO 中， 并在释放时解除登记
MEM_MAGIC_CHECK	1	静态	是否开启内存校验。0：否；1：是，通常用于调试版本，打开此开关可以在内存错引发更严重问题之前主动终止系统
线程相关参数			
WORKER_THREADS	4	静态	工作线程的数目，有效值范围（1~64）
TASK_THREADS	4	静态	任务线程个数，有效值范围（1~1000）
UTHR_FLAG	0	手动	用户线程标记，1：启用；0：不启用。 启用用户线程时，并行查询失效，并行查询的相关参数不起作用
FAST_RW_LOCK	1	手动	快速读写锁标记，1 表示启用，0 不启用
SPIN_TIME	4000	静态	线程在不能进入临界区时，自旋的次数。有效值范围（0~4000）
WORK_THRD_STACK_SIZE	1024	静态	工作线程堆栈大小，以 KB 为单位。有效值范围（64~4096）
WORKER_CPU_PERCENT	0	手动	工作线程占 CPU 的比重，仅非 WINDOWS 下有效，有效值范围（0~100）。0 表示不限制，相当于 100
NESTED_C_STYLE_COMMENT	0	动态，系统级	是否支持 C 风格的嵌套注释。0：不支持；1：支持
查询相关参数			
USE_PLN_POOL	1	静态	是否重用执行计划。0：禁止执行计划的重用；1：启用执行计划的重用功能 ；2：对不包含显式参数的语句进行常量参数化优化；3：即使包含显式参数的语句，也进行常量参数化优化
DYN_SQL_CAN_CACHE	1	动态，系统级	是否缓存动态语句的执行计划。0：不缓存；1：当 USE_PLN_POOL 不为 0 时，缓存动态语句

			的执行计划
RS_CAN_CACHE	0	静态	结果集缓存配置。0：禁止重用结果集；1：强制模式，此时默认缓存所有结果集，但可通过 RS_CACHE_TABLES 参数和语句 HINT 进行手动设置；2：手动模式，此时默认不缓存结果集，但可通过语句 HINT 对必要的结果集进行缓存
RS_CACHE_TABLES	空串	手动	指定可以缓存结果集的基表的清单，当 RS_CAN_CACHE=1 时，只有查询涉及的所有基表全部在此参数指定范围内，该查询才会缓存结果集。当参数值为空串时，此参数失效。
RS_CACHE_MIN_TIME	0	动态，系统级	结果集缓存的语句执行时间下限，只有实际执行时间不少于指定时间值的查询，其结果集才会被缓存，仅在 RS_CAN_CACHE=1 时有效。默认值 0，表示不限制；有效值范围（0~4294967294），以 MS 为单位
RS_BDTA_FLAG	0	静态	是否以 BDTA 形式返回结果集。0：以行为单位返回结果集；2：以 BDTA 形式返回结果集
RS_BDTA_BUF_SIZE	32	静态	配置消息长度，单位为 K。有效值范围（8~32768）
RESULT_SET_LIMIT	10000	动态，会话级	一次请求可以生成的结果集最大个数。有效值范围（1~65000）
RESULT_SET_FOR_QUERY	0	动态，会话级	是否生成非查询结果集。0：生成；1：不生成
SESSION_RESULT_SET_LIMIT	10000	动态，系统级	会话上结果集个数上限，有效值范围（100~65000）
BUILD_FORWARD_RS	0	静态	仅向前游标是否生成结果集。0：不生成；1：生成
MAX_OPT_N_TABLES	6	动态，会话级	优化器在处理连接时，一次能优化的最大表连接个数。有效值范围（3~8）
CNNTB_MAX_LEVEL	20000	动态，会话级	层次查询的最大支持层次。有效值范围（1~100000）
BATCH_PARAM_OPT	0	静态	是否启用批量参数优化，0：不启用；1：启用，默认不启用。当置为 1 时，不返回操作影响的行数
CLT_CONST_TO_PARAM	0	静态	是否进行语句的常量参数化优化，0：不进行；1：进行
LIKE_OPT_FLAG	7	动态，会话级	LIKE 查询的优化开关。0：不优化；1：对于 LIKE 表达式首尾存在通配符的情况，优化为 POSITION()函数；对于 LIKE 表达式首部存在通配符，并且条件列存在 REVERSE()函数索引时，优化为 REVERSE()函数；2：对于 COL1 LIKE COL2 || '%'的情况，优化为 POSITION()函数；4：对于 COL1 LIKE ‘A’ || ‘B%’的情况，优化为 COL1 LIKE ‘AB%’。支持使用上述有效值的组合值，如 5 表示同时进行 1 和 4 的优化
FILTER_PUSH_DOW	0	动态，会	对单表条件是否下放的不同处理方式。0：表示

N		话级	条件不下放：1：表示层次查询中将 START WITH 条件进行下放；2：表示在新优化器下对外连接、半连接进行下放条件优化处理；4：语义分析阶段考虑单表过滤条件的选择率，超过 0.5 则不下放，由后面进行代价计算选择是否下放，参数值 4 仅在参数取值包含 2 时有效，即将参数值设为 6 时有效；8：表示尝试将包含非相关子查询的布尔表达式进行下放。 支持使用上述有效值的组合值，如 6 表示同时进行 2 和 4 的优化
USE_MCLCT	2	动态，会话级	MPP/LPQ 下，是否替换 MGAT/LGAT 通讯操作符为 MCLCT/LCLCT。0：不替换；1：MPP 下将操作符 MGAT 替换为 MCLCT；2：MPP 下将操作符 MGAT 替换为 MCLCT，或 LPQ 下将操作符 LGAT 替换为 LCLCT
MPP_OP_JUMP	1	动态，会话级	MPP 系统中操作符的跳转开关，是否支持通讯操作符的跳转功能。0：不支持；1：支持
PHF_NTTS_OPT	1	动态，会话级	MPP 系统中是否进行 NTTS 计划的优化，打开时可能减少计划中的 NTTS 操作符。0：不支持；1：支持
MPP_MOTION_SYNC	200	动态，会话级	通讯操作符同步时认定的邮件堆积数，堆积超过该值则要进行同步检查。取值 0-100000，0 表示不进行同步检查
USE_FTTS	0	动态，会话级	执行过程中产生的临时数据的存放格式。0：用临时表空间的数据页存放；1：用临时文件存放
UPD_DEL_OPT	2	动态，会话级	删除更新计划优化方式，取值范围 0、1、2 单节点计划，0：不优化 1，2:都可优化 NTTS MPP 计划，0：不优化； 1：优化删除更新计划，不优化 NTTS；2：优化删除更新计划，同时优化 NTTS
ENABLE_DIST_IN_SUBQUERY_OPT	0	动态，系统级	优化子查询里面的 IN，0：不优化；1：优化
MAX_OPT_N_OR_BEXPS	7	动态，会话级	能参与优化的最大 OR 分支个数，超过时布尔表达式仅用于过滤使用。有效值范围（7~64）
USE_HAGR_FLAG	0	动态，会话级	当带有 DISTINCT 的集函数不能使用 SAGR 操作符时，是否使用 HAGR，0：不使用；1：使用
DTABLE_PULLUP_FLAG	1	动态，会话级	是否在语法分析阶段对派生表进行上拉优化处理，0：不优化；1：优化
VIEW_PULLUP_FLAG	0	动态，会话级	是否对视图进行上拉优化，把视图转换为其原始定义，消除视图。可取值 0、1、2。 0：不进行视图上拉优化； 1：对不包含别名和同名列的视图进行上拉优化； 2：对包含别名和同名列的视图也进行上拉优化
VIEW_PULLUP_MAX_TAB	7	动态，会话级	对视图进行上拉优化支持的表的个数。有效值范围：（1~16）

STR_NULL_OPS_COMPATIBLE	0	动态，会话级	当两个字符串相加时，若两个字符串中有一个为 NULL，是否将结果置为 NULL。0：否；1：是
GROUP_OPT_FLAG	4	动态，会话级	分组项优化参数开关。0：不优化；1：非 MYSQL 兼容模式下（即 COMPATIBLE_MODE 不等于 4），支持查询项不是 GROUP BY 表达式； 4：表示对于多级分区，并行下允许尝试不生成多个 AGR 支持使用上述有效值的组合值，如 5 表示同时进行 1 和 4 的优化
HAGR_PARALLEL_OPT_FLAG	0	动态，会话级	MPP、并行下对 GROUP BY、分析函数等的优化开关。0：不优化；1：无 DISTINCT 时 HAGR 按照分组列分发；2：有 DISTINCT 时，HAGR 按照分组列分发；4：去除多余的通讯操作符；8：同 4，仅限于外连接操作；16：分析函数按照 PARTITION BY 列分发；32：MPP+LPQ 下，AAGR 的优化处理 支持使用上述有效值的组合值，如 5 表示同时进行 1 和 4 的优化
HAGR_DISTINCT_OPT_FLAG	0	动态，会话级	MPP 下是否对 HAGR+DISTINCT 进行优化，0：不优化；1：优化
REFED_EXISTS_OPT_FLAG	1	动态，会话级	是否把相关 EXISTS 优化为非相关 IN 查询。0：否；1：是
REFED_OPS_SUBQUERY_OPT_FLAG	0	动态，会话级	是否将 OP ALL/SOME/ANY 相关子查询转换为 EXISTS 相关子查询。0：否；1：是；2：仅对包含 PARTITION JOIN 连接的相关子查询进行转换（已废弃，不进行转换） 支持使用上述有效值的组合值，如 3 表示同时进行 1 和 2 的优化，由于值为 2 的优化方式已被废弃，因此值为 3 时按照值为 1 进行处理
MAX_PHC_BE_NUM	512	动态，会话级	优化阶段存放临时布尔表达式的个数。有效值范围（512~ 20480000）
HASH_PLL_OPT_FLAG	0	动态、会话级	是否裁剪 HASH SEMI/INNER 连接右边的分区表。取值 0 或 1。1 裁剪，0 不裁剪
PARTIAL_JOIN_EVALUATION_FLAG	1	动态，会话级	是否对去除重复值操作的下层连接进行转换优化，0：不优化；1：优化 此参数仅在参数 OPTIMIZER_MODE 为 1 时才有效
USE_FK_REMOVE_TABLES_FLAG	1	动态，会话级	是否利用外键约束消除冗余表。0：不启用；1：启用
MPP_HASH_LR_RATE	10	动态，会话级	MPP 下，对 HASH JOIN 节点，可以根据左右儿子 CARD 代价的比值，调整 HASH_JOIN 的左右儿子的 MOTION 添加，从而影响计划。如果 CARD 比值超过此值，则小数据量的一方全部收集到主 EP 来做。有效值范围（1~4294967294）
LPQ_HASH_LR_RAT	30	动态，会	LPQ 下，对 HASH JOIN 节点，可以根据左右

E		话级	儿子 CARD 代价的比值，调整 HASH_JOIN 的左右儿子的 MOTION 添加，从而影响计划。 如果 CARD 比值超过此值，则小数据量的一方全部收集到主 EP 来做。有效值范围（1~4294967294）
USE_HTAB	1	动态，会话级	表示计划中是否能使用 HTAB。0：否；1：是
SEL_ITEM_HTAB_FLAG	0	动态，会话级	当 USE_HTAB=1 时才有效。 当查询项中有相关子查询时，是否做 HTAB 优化。0：不优化；1：优化
OR_CVT_HTAB_FLAG	1	动态，会话级	当 USE_HTAB=1 时才有效。 当查询条件 OR 中含有公共因子时，是否允许使用 HTAB 来进行优化。0：不使用；1：使用；2：增强 OR 表达式转换为 HTAB 条件检查，当存在嵌套连接时，不生成 HTAB，以避免缓存过多数据影响性能
CASE_WHEN_CVT_IFUN	5	动态，会话级	是否将 CASE WHEN THEN ELSE END 语句转换为 IFOPERATOR 函数。0：不转换；1：转换；2：转换，且有限制地进行表达式重用；4：对于 CASE WHEN 查询表达式不考虑 THEN…ELSE 表达式重用 支持使用上述有效值的组合值，如 5 表示同时进行 1 和 4 的优化
OR_NBEXP_CVT_CASE_WHEN_FLAG	0	动态，会话级	是否将OR 转化为CASE WHEN THEN ELSE END 语句。0：不转换；1：转换
NONCONST_OR_CVT_IN_LST_FLAG	0	动态，会话级	是否开启 IN LIST 优化：将不含有常量的 OR 表达式转换成 IN LIST。0 表示不开启，1 表示开启
OUTER_CVT_INNER_PULL_UP_COND_FLAG	1	动态，会话级	当外连接转化为内连接时，是否打开连接条件。0：不打开；1：打开
OPT_OR_FOR_HUGE_TABLE_FLAG	1	动态，会话级	是否使用 HFSEK 优化 HUGE 表中列的 OR 过滤条件。0：不使用；1：使用
ORDER_BY_NULLS_FLAG	0	动态，会话级	ASC 升序排序时，控制 NULL 值返回的位置。取值 0 或 1。1 表示 NULL 值在最后返回，0 表示 NULL 值在最前面返回。在参数等于 1 的情况下，NULL 值的返回与 ORACLE 保持一致。DESC 降序时该参数无效
SUBQ_CVT_SPL_FLAG	1	动态，会话级	控制相关子查询的实现方式，0：不优化；1：使用 SPL2 方式实现相关子查询；2：DBLINK 相关子查询是否转换为函数，由参数 ENABLE_DBLINK_TO_INV 取值决定；4：将多列 IN 转换为 EXISTS；8：将引用列转换为变量 VAR；16：用临时函数替代查询项中的相关查询表达式 支持使用上述有效值的组合值，如 5 表示同时进行 1 和 4 的优化

ENABLE_RQ_TO_SPL	1	动态，会话级	是否将相关子查询转换为 SPL2 方式，0：不优化；1：优化
MULTI_IN_CVT_EXISTS	0	动态，会话级	多列 IN 是否转换为等价的 EXISTS 过滤。0：不转换；1：转换
PRJT_REPLACE_NPAR	1	动态，会话级	是否将引用列转换为变量 VAR，并替换查询表达式中的引用列，0：不优化；1：优化
ENABLE_RQ_TO_INV	0	动态，会话级	相关查询表达式是否转换为函数方式实现。0：不转换；1：转换
SUBQ_EXP_CVT_FLAG	0	动态，会话级	是否将带有聚集函数且没有 GROUP BY 的相关查询表达式优化为非相关查询表达式。 0：使用普通的去相关性处理； 1：带有聚集函数且没有 GROUP BY 的相关查询表达式优化为非相关查询表达式； 2：将子查询列均来自上层查询且子查询中不包含层次查询、ROWNUM、TOP 的相关子查询的 FROM 项改写为(SELECT TOP 1 1 FROM ...)派生表，减少中间结果集； 8：将 EXISTS 子查询的查询项转换为常量 0，兼容 ORACLE 处理方式，同时优化子查询对查询项的处理过程。如果子查询为集合查询，则不进行转换； 16：通过原始语句判断查询表达式是否相同。 支持使用上述有效值的组合值，如 3 表示同时进行 1 和 2 的优化
USE_REFER_TAB_ONLY	0	动态，会话级	处理相关子查询时是仅将相关的表下放，或者连同上方的 SEMI/HASH JOIN 一起下放。0：一起下放；1：仅下放相关表
REFED_SUBQ_CROSS_FLAG	1	动态，会话级	是否将相关子查询与外层表优化为 CROSS JOIN，0：不优化；1：优化
IN_LIST_AS_JOIN_KEY	0	动态，会话级	搜索多表连接方式时，对于索引连接（INDEX JOIN）的探测，NEXP_IN_LST 表达式类型可以作为多表连接的 KEY。 取值范围： 0：表示搜索多表连接方式时，对于索引连接（INDEX JOIN）的探测，NEXP_IN_LST 不可以作为连接 KEY； 1：表示搜索多表连接方式时，对于索引连接（INDEX JOIN）的探测，把 NEXP_IN_LST 当做普通等值 KEY 的处理方式来生成 INDEX JOIN 的连接 KEY。
OUTER_JOIN_INDEX_OPT_FLAG	0	动态，会话级	外连接优化为索引连接的优化开关。1：优化；0：不优化。
OUTER_JOIN_FLATING_FLAG	0	动态，会话级	优化外连接。 0：不启用优化； 1：启用优化，例如，使得类似这样的连接 A LEFT (B CROSS C) ON A.C1=B.C1 优化为 A LEFT (B JOIN (DISTINCT A) ON

			A.C1=B.C1 CROSS C) ON A.C1=B.C1，相当于把 A 和 B 的过滤器平坦化到了下层查询中，使得 B 获取的中间结果较小
TOP_ORDER_OPT_FLAG	0	动态，会话级	优化带有 TOP 和 ORDER BY 子句的查询，使得 SORT 操作符可以省略。优化的效果是尽量使得 ORDER BY 的排序列所对应的基表可以使用包含排序列的索引，从而可以移除排序 SORT 操作符，减少排序操作。如果排序列不属于同一个基表，或者排序列不是基表列，则肯定是不可以优化。 取值： 0：不启用该优化 1：启用该优化
TOP_DIS_HASH_FLAG	1	动态，会话级	是否通过禁用 HASH JOIN 方式来优化 TOP 查询，取值： 0：不优化； 1：当 OPTIMIZER_MODE 为 0 时，TOP 下方连接禁用 HASH JOIN；当 OPTIMIZER_MODE 为 1 时，TOP 下方最近的连接倾向于不使用 HASH JOIN； 2：当 OPTIMIZER_MODE 为 0 时，TOP 下方连接禁用 HASH JOIN；当 OPTIMIZER_MODE 为 1 时，TOP 下方所有连接都倾向于不使用 HASH JOIN
ENABLE_RQ_TO_NONREF_SPL	0	动态，会话级	相关查询表达式转化为非相关查询表达式，目的在于相关查询表达式的执行处理由之前的平坦化方式转化为一行一行处理，类似 ORACLE 的每行处理机制。 0：不启用该优化；1：对查询项中出现的相关子查询表达式进行优化处理；2：对查询项和 WHERE 表达式中出现的相关子查询表达式进行优化处理
OPTIMIZER_MODE	1	动态，会话级	DM 优化器的模式，0：老优化器模式；1：新优化器模式
OPTIMIZER_MAX_PERM	7200	动态，会话级	控制计划探测过程中的最大排列数，如果大于该阈值，则减少相应的排列，减少探测计划的时间，有效值范围（1~4294967293）
ENABLE_INDEX_FILTER	0	动态，会话级	是否进行索引过滤优化，可取值为 0、1、2。 0 不进行优化； 1 使用索引过滤优化，如果过滤条件涉及的列包含在索引中，那么索引进行 SSEK2 后就可以使用此过滤条件，可以减少中间结果集； 2 在取值为 1 的基础上，将 IN 查询列表转换为 HASH RIGHT SEMI JOIN。
OPTIMIZER_DYNAMIC_SAMPLING	0	动态，会话级	当统计信息不可用时是否启用动态统计信息。取值范围：0~12。0：不启用；1~10：启用，采用率 10%~100%；11：启用，由优化器确定采样率（0.1%~99.9%）；12：同 11，但收集

			的结果会持久化保存
NONREFED_SUBQUERY_AS_CONST	0	动态，会话级	是否将非相关子查询转化为常量处理。0：不进行优化，对非相关子查询使用连接方式处理；1：将非相关子查询转换为常量，作为过滤条件使用
HASH_CMP_OPT_FLAG	0	动态，会话级	是否启用静态哈希表的优化。0：不启用优化；1：分组中的 DISTINCT 开启此优化；2：对 HASH 连接启用该优化；4：对 HAGR 分组计算启用该优化；8：DISTINCT 操作开启此优化。支持使用上述有效值的组合值，如 5 表示同时进行 1 和 4 的优化
OUTER_OPT_NLO_FLAG	0	动态，会话级	外连接或内连接是否支持右孩子为非基表情况下的 INDEX JOIN。0：不启用；1：启用外连接下的优化；2：启用内连接下的优化；3：同时启用内连接和外连接。 该参数仅在 OPTIMIZER_MODE = 0 时生效
DISTINCT_USE_INDEX_SKIP	2	动态，会话级	DISTINCT 列是否使用索引跳跃扫描（单列索引或复合索引）。专门用于 SQL 语句中有 DISTINCT 的场景，DISTINCT 列的查询的一种优化方式，在索引上跳跃着扫描。0：不使用；1：强制使用；2：根据代价选择是否使用。 该参数只在 OPTIMIZER_MODE=1 时有效
USE_INDEX_SKIP_SCAN	0	动态，会话级	是否使用复合索引跳跃扫描。专门用于 WHERE 子句等值条件的查询列中包含了复合索引的列，但该列又不是复合索引的前导列，这种情况下，选择是否使用复合索引跳跃扫描。和 INDEX_SKIP_SCAN_RATE 搭配使用。 0：不使用； 1：根据代价选择是否使用。通过代价计算选择，倾向于选择前导列 DISTINCT 值少，搜索列 DISTINCT 值较多的索引； 2：强制使用。只要能够找到索引可以使用跳跃扫描，就强制使用
INDEX_SKIP_SCAN_RATE	0.0025	动态，会话级	复合索引跳跃扫描的代价调节开关。当列的（DISTINCT 数/总行数）比值大于该值时，就不再使用索引跳跃扫描的方式。取值范围 0~1。和 USE_INDEX_SKIP_SCAN 搭配使用
SPEED_SEMI_JOIN_PLAN	1	动态，会话级	是否加速半连接的探测过程。 0：不加速；1：加速计划探测；2：加速计划探测和执行
COMPLEX_VIEW_MERGING	0	动态，会话级	对于复杂视图(一般含有 GROUP 或者集函数等）会执行合并操作，使得 GROUP 分组操作在连接之后才执行。0：不启用；1：对不包含别名和同名列的视图进行合并；2：视图定义包含别名或同名列时也进行合并
OP_SUBQ_CVT_IN_FLAG	1	动态，会话级	当查询条件为=(SUBQUERY)，是否考虑转换为等价的 IN(SUBQUERY)。0：不转换；1：转换

HLSM_FLAG	1	动态，会话级	控制多列非相关 NOT IN 的查询实现方式，1：当数据量较大，HASH BUF 放不下时，采用 MTAB 方式处理；2：当数据量较大，HASH BUF 放不下时，采用 B 树方式处理，使用细粒度扫描；3：当数据量较大，HASH BUF 放不下时，采用 B 树方式处理，使用粗粒度扫描
DEL_HP_OPT_FLAG	0	动态，会话级	控制分区表的操作优化，0：不优化；1：打开分区表 DELETE 优化；2：控制范围分区表创建的优化处理，转换为数据流方式实现；4：允许语句块中的间隔分区表自动扩展；8：开启对 TRUNCATE 分区表的优化处理。 支持使用上述有效值的组合值，如 7 表示同时进行 1、2、4 的优化
OPTIMIZER_OR_NBEXP	0	动态 会话级	OR 表达式的优化方式。0：不优化；1：生成 UNION_FOR_OR 操作符时，优化为无 KEY 比较方式；2：OR 表达式优先考虑整体处理方式；4：相关子查询的 OR 表达也优考虑整体处理方式；8：OR 布尔表达式的范围合并优化；16：同一列上同时存在常量范围过滤和 IS NULL 过滤时的优化，如 C1 > 5 OR C1 IS NULL。 支持使用上述有效值的组合值，如 7 表示同时进行 1、2、4 的优化
CNNTB_OPT_FLAG	0	动态，会话级	是否使用优化的层次查询执行机制。0：不使用；1：强制使用；2：优化器自动决定是否使用
ADAPTIVE_NPLN_FLAG	3	动态，会话级	是否启用自适应计划机制，仅 OPTIMIZER_MODE=1 时生效。0：不启用；1：对索引连接、嵌套含 VAR 连接等复杂连接启用自适应计划；2：ORDER BY 在 HASH 连接时启用自适应计划；3：同时启用 1 和 2 的优化机制
MULTI_UPD_OPT_FLAG	0	动态，会话级	是否使用优化的多列更新。0：不使用，仍按照语句改写方式实现；1：利用多列 SPL 功能加以实现。
ENHANCE_BIND_PEEKING	0	静态	是否使用自适应的绑定变量窥探开关。0：不使用；1：使用
HAGR_HASH_ALGORITHM_FLAG	0	动态，会话级	HAGR 中 HASH 算法选择标记。0：按照移位方式计算；1：按照异或方式计算
DIST_HASH_ALGORITHM_FLAG	0	动态，会话级	DISTINCT 中 HASH 算法选择标记。0：按照移位方式计算；1：按照异或方式计算
UNPIVOT_ORDER_FLAG	0	动态，会话级	是否对 UNPIVOT 的结果按照公共列排序，公共列指表中未出现在 UNPIVOT 的列。0：不排序；1：排序
VIEW_FILTER_MERGING	2	动态，会话级	是否对视图条件进行合并优化以及如何优化，0：不优化；1：尽可能地进行视图条件合并；2：自动判断是否进行视图条件合并
OPT_MEM_CHECK	0	动态，会话级	内存紧张时，优化器是否缩减计划探测空间，仅 OPTIMIZER_MODE=0 时生效。0：不缩减计划探测空间；1：缩减计划探测空间；

ENABLE_JOIN_FACTORIZATION	0	动态，会话级	是否启用连接分解，即 UNION ALL 分支间存在公共部分时是否进行公因子提取。0：不启用；1：启用
ERROR_COMPATIBLE_FLAG	0	动态、会话级	是否对子查询同名列进行报错。取值 0 或 1。0 报错，1 不报错
ENABLE_PARTITION_WISE_OPT	0	动态，会话级	是否利用水平分区表的分区信息进行排序、分组的计划优化，即在满足条件的情况下对每个子表单独排序、分组，或者进行子表间的归并排序。0：不启用；1：启用
EXPLAIN_SHOW_FACTOR	1	动态，会话级	显示执行计划的行数和嵌套层数的基数分别为 1000 和 100，实际显示的行数和嵌套层数为基数* EXPLAIN_SHOW_FACTOR。有效值范围（1~100）
HASH_PLL_OPT_FLAG	0	动态，会话级	当进行 HASH SEMI/INNER 连接时，若右表为分区表，是否对分区表进行裁剪。0：否；1：是
PLACE_GROUP_BY_FLAG	0	动态，会话级	在含有集函数的查询中，是否允许先分组减少数据量后再进行连接。0：不允许；1：允许
ENABLE_NEST_LOOP_JOIN_CACHE	0	动态，会话级	是否考虑缓存嵌套循环连接中间结果以加速执行。取值 0 或 1。0：不启用；1：启用
ENABLE_DBLINK_TO_INV	0	动态，会话级	DBLINK 相关子查询是否转换为函数。0：不转换；1：转换
ENABLE_CREATE_BM_INDEX_FLAG	1	静态	是否允许创建位图索引。0：不允许创建位图索引，位图索引作为普通 B 树索引进行创建；1：允许创建位图索引
检查点相关参数			
CKPT_RLOG_SIZE	100	动态，系统级	产生多大日志文件后做检查点，以兆为单位。有效值范围（0~4294967294）
CKPT_DIRTY_PAGES	10000	动态，系统级	产生多少脏页后产生检查点，以页为单位。有效值范围（0~4294967294）
CKPT_INTERVAL	300	动态，系统级	指定检查点的时间间隔。以秒为单位，为 0 时表示不自动定时做检查点。有效值范围（0~2147483647）
CKPT_FLUSH_RATE	5.00	动态，系统级	检查点刷盘比例。有效值范围（0~100.00）
CKPT_FLUSH_PAGES	1000	动态，系统级	检查点刷盘的最小页数。有效值范围（1000~100000）
CKPT_WAIT_PAGES	128	动态，系统级	检查点一次发起的最大写入页数，等待这些页写入磁盘完成、调整检查点信息后，再发起新的刷盘请求，避免过于集中发起写磁盘请求，操作系统 IO 压力过大，导致 IO 性能下降。有效值范围（1~1024）
FORCE_FLUSH_PAGES	8	动态，系统级	调度线程启动刷脏页流程时，每个 BUFFER POOL 写入磁盘的脏页数。有效值范围（0~1000）
IO 相关参数			
DIRECT_IO	0	静态	非 WINDOWS 下有效，0：使用 OS 文件系统缓

			存；1：不使用 OS 文件系统缓存，采用系统模拟的异步 IO，异步 IO 的线程数由 IO_THR_GROUP 控制；2：不使用 OS 文件系统缓存，采用系统提供的 NATIVE IO 机制， 要求 LINUX 内核 2.6 以上
IO_THR_GROUPS	2	静态	非 WINDOWS 下有效，表示 IO 线程组个数。有效值范围（1~512）
HIO_THR_GROUPS	2	静态	HUGE 缓冲区 IO 线程组数目。 有效值范围 （1 ~ 512）
FAST_EXTEND_WITH_DS	LINUX/AIX/BSD 缺省为 1,其他缺省 0	动态，系统级	是否按实际磁盘占用大小扩展文件，非 WINDOWS 有效 0：扩展文件为空洞文件 1：按 DISK SPACE 扩展文件，文件的逻辑大小与实际磁盘占用大小一致
FIL_CHECK_INTERVAL	0	动态，系统级	指定检查数据文件是否存在的时间间隔，单位：S。有效值范围（0~4294967294）。0 表示不检查。
数据库相关参数			
MAX_SESSIONS	100	静态	系统允许同时连接的最大数，同时还受到 LICENSE 的限制，取二者中较小的值，有效值范围（1~65000）
MAX_CONCURRENT_TRX	0	静态	表示系统支持同时运行事务数的最大值。有效值范围（0~1500），0 表示不控制 注：这个参数仅在需要超大数量连接时才需要设置
CONCURRENT_TRX_MODE	0	静态	限流模式，0：以事务为单位进行限流；1：以 SQL 为单位进行限流
TRX_VIEW_SIZE	512	静态	事务视图中本地事务 ID 的缓存初始化个数。取值范围（16~65000）
MAX_SESSION_STATEMENT	100	动态，系统级	单个会话上允许同时打开的语句句柄最大数，有效值范围（64~20480）
MAX_CONCURRENT_OLAP_QUERY	0	静态	OLAP 模式下能同时执行的复杂查询个数，超过限制后，复杂查询将在执行阶段进入等待。0 表示不做限制。有效值范围（0~100） 注：此处的复杂查询是指计划中存在行数超过参数 BIG_TABLE_THRESHHOLD*10000 的节点，含有哈希连接、哈希分组等耗内存操作符，且不涉及到系统表
BIG_TABLE_THRESHHOLD	1000	动态，会话级	OLAP 环境下对复杂查询最大并发数（参数 MAX_CONCURRENT_OLAP_QUERY）进行限制时，认定是否为复杂查询中表的行数下限，以万为单位。取值范围(0~4294967294)
MAX_EP_SITES	64	手动	MPP 环境下 EP 站点的最大数量，有效值范围（2~1024）
PORT_NUM	5236	静态	服务器通讯端口号， 有效值范围（1024~65534）
FAST_LOGIN	0	静态	是否在登录时记录登录历史信息。0记录，1不

			记录
DDL_AUTO_COMMIT	1	手动	指定 DDL 语句是否自动提交，1：自动；0：手动。当 COMPATIBLE_MODE=1 时，DDL_AUTO_COMMIT 的实际值均为 0。
COMPRESS_MODE	0	动态，会话级	建表时是否缺省压缩。0：不进行；1：进行
PK_WITH_CLUSTER	1	动态，会话级	在建表语句中指定主关键字时，是否缺省指定为 CLUSTER，0：不指定；1：指定 注：该参数对列存储表和堆表无效
EXPR_N_LEVEL	200	动态，会话级	表达式最大嵌套层数。有效值范围（30~1000）
N_PARSE_LEVEL	100	动态，会话级	表示对象 PROC、VIEW、PKG、CLASS 的最大解析层次，如果层次过深则报错返回。有效值范围（30~1000）
MAX_SQL_LEVEL	500	动态，系统级	指定 DM 虚拟机允许的最大栈帧数。有效值范围（100~1000）。此参数值设置过大可能会导致内存暴涨，管理员需斟酌设置
BDTA_SIZE	1000	静态	BDTA 缓存的记录数。有效值范围（1~10000）
OLAP_FLAG	2	动态，会话级	启用联机分析处理，0：不启用；1：启用；2：不启用，同时倾向于使用索引范围扫描
JOIN_HASH_SIZE	500000	动态，会话级	HASH JOIN 操作时，HASH 表大小，以 CELL 个数为单位。有效值范围（1~250000000）
USE_DHASH_FLAG	0	动态，会话级	是否使用动态 HASH。0：不使用；1：使用
HFILES_OPENED	256	静态	设置可以同时打开的列存储表数据文件个数。有效值范围（60-10000）
FAST_COMMIT	0	动态，系统级	批量提交事务的个数，有效值范围（0~100）
ISO_IGNORE	0	动态，会话级	是否忽略显示设置的事务隔离级别，1 表示忽略，0 不忽略。
TEMP_SIZE	10	静态	默认创建的临时表空间大小，以兆为单位。有效值范围（10~1048576）
TEMP_SPACE_LIMIT	0	动态，系统级	临时表空间大小上限，以 M 为单位。0 表示不限制临时表空间大小。 有效范围（0~ 4294967294）。 注意：TEMP_SPACE_LIMIT 一定要大于等于 TEMP_SIZE
FILE_TRACE	0	静态	日志中是否记录文件操作，0：不记录；1：记录
COMM_TRACE	1	动态，会话级	服务器日志是否记录通信中产生的警告信息。0：不记录；1：记录
ERROR_TRACE	0	动态，会话级	服务器日志中是否记录语法分析出错的语句。1：是，0：否
CACHE_POOL_SIZE	10	静态	SQL 缓冲池大小，以兆为单位。有效值范围：32 位平台下为（1~2048）；64 位平台下为（1~67108864）。单位：MB

STAT_COLLECT_SIZE	10000	动态，会话级	统计信息收集时，样本的最小行数。有效值范围（0~10000000）
STAT_ALL	0	动态，会话级	在估算分区表行数时，控制一些优化。0：不采样所有分区子表；1：采样所有分区子表；2：优先采用统计信息中的行数。 支持使用上述有效值的组合值，如 3 表示优先采用统计信息中的行数，如果没有找到对应统计信息，则去采样所有分区子表。
PHC_MODE_ENFORCE	0	动态，会话级	控制连接的实现方式。0：优化器根据代价情况自由选择连接方式；1：考虑使用 NEST LOOP INNER JOIN；2：考虑使用索引连接；4：考虑使用哈希连接；8：考虑使用归并连接 支持使用上述有效值的组合值，如 6 表示优化器根据代价情况在索引连接和哈希连接间进行选择
ENABLE_HASH_JOIN	1	动态，会话级	是否允许使用哈希连接，0：不允许；1：允许。
ENABLE_INDEX_JOIN	1	动态，会话级	是否允许使用索引连接，0：不允许；1：允许。
ENABLE_MERGE_JOIN	1	动态，会话级	是否允许使用归并连接，0：不允许；1：允许
MPP_INDEX_JOIN_OPT_FLAG	0	动态，会话级	MPP 下，是否可以使用索引左外连接，0：不使用；1：使用
MPP_NLI_OPT_FLAG	0	动态，会话级	MPP 下嵌套连接的优化方式，0：嵌套连接操作的左右孩子添加 MPP GATHER，汇总到主 EP 执行， 父节点添加 MPP SCATTER；3：嵌套连接操作的左右孩子都添加 MPP GATHER 和 MPP SCATTER
MAX_PARALLEL_DEGREE	1	动态，会话级	用来设置默认并行任务个数。取值范围：1~128。缺省值 1，表示无并行任务。当 PARALLEL_POLICY 值为 1 时该参数值才有效
PARALLEL_POLICY	0	静态	用来设置并行策略。取值范围：0、1 和 2，缺省为 0。其中，0 表示不支持并行；1 表示自动并行模式；2 表示手动并行模式
PARALLEL_THRD_NUM	10	静态	用来设置并行工作线程个数。有效值范围（1~1024）
PARALLEL_MODE_COMMON_DEGREE	1	动态，会话级	并发模式程度。此值越大，扫描代价将越大，越倾向于使用索引连接而不是哈希连接，需酌情使用。有效值范围（1~1024）
PUSH_SUBQ	0	动态，会话级	是否下放子查询使先做。0：相关子查询不下放，非相关子查询在不存在单表过滤条件时下放；1：不存在单表过滤条件时下放；2：始终考虑下放
OPTIMIZER_AGGR_GROUPBY_ELIM	1	动态，会话级	当对派生视图进行分组查询，且分组项是派生视图分组项的子集时，是否考虑将两层分组进行合并。0：不优化；1：将两层分组进行合并
UPD_TAB_INFO	0	静态	系统启动时是否更新表信息（如行数），1：启用更新；0：不更新。
ENABLE_IN_VALUE_LIST_OPT	6	动态，会话级	是否允许 IN LIST 表达式优化。0：不优化。1：将 IN LIST 表达式在语义分析阶段优化为

			CONSTV 处理； 2：在查询计划确定后，把 IN LIST 表达式转化为 CONSTV；4：允许 IN LIST 参与等值传递；8：从 IN VALUE 中构造出范围条件（仅限于 HUGE 表） 支持使用上述有效值的组合值，如 3 表示同时进行 1 和 2 的优化
ENHANCED_BEXP_TRANS_GEN	1	动态，会话级	是否允许非等值布尔表达式和外连接 ON 条件中的传递闭包。0：不允许；1：允许
ENABLE_DIST_VIEW_UPDATE	0	动态，系统级	是否支持更新含有 DISTINCT 的视图，1：允许；0：不允许，当 COMPATIBLE_MODE=1 时，ENABLE_DIST_VIEW_UPDATE 的实际值均为 1
STAR_TRANSFORMATION_ENABLED	0	动态，会话级	是否允许对星形模型查询改写以使用位图连接索引。0：不启用；1：启用 此参数仅在 OLAP_FLAG 为 1 时才有效
MONITOR_INDEX_FLAG	0	动态，会话级	是否对索引进行监控，0：关闭自动监控，可使用 ALTER INDEX 语句启用索引监控；1：打开自动监控，对用户定义的二级索引进行监控；2：禁止索引监控
RAISE_CASE_NOT_FOUND	0	动态，系统级	CASE 语句 NOT FOUND 是否抛出异常。0：不抛出；1：抛出
FIRST_ROWS	100	动态，系统级	结果集一个消息中返回的最大行数，有效值范围（1~1000）
LIST_TABLE	0	动态，会话级	默认情况下，创建的表是否为堆表，0：否；1：是
ENABLE_SPACELIMIT_CHECK	1	静态	是否启用检查 SPACE LIMIT。1：启用；0：不启用
BUILD_VERTICAL_PK_BTREE	0	手动	HUGE表上创建的主键是否需要创建物理B树。0：不创建；1：创建
BDTA_PACKAGE_COMPRESS	0	动态，会话级	是否启动bdta压缩传递功能，1：压缩；0：不压缩
HFINS_PARALLEL_FLAG	0	动态，系统级	是否启动HUGE表查询插入优化。0：否；1：对非HUGE分区表进行优化；2：对HUGE分区表也进行优化
HFINS_MAX_THRD_NUM	100	动态，系统级	HUGE表查询插入优化时并行的最大线程数，有效值范围（4~200）
LINK_CONN_KEEP_TIME	15	静态	DBLINK空闲连接保持时间，以分钟为单位，为0时表示不主动释放空闲DBLINK连接。
DETERMIN_CACHE_SIZE	5	动态，会话级	确定性函数值缓冲区大小，以MB为单位
NTYPE_MAX_OBJ_NUM	1000000	动态，系统级	复合数据类型中包含的对象或者字符串的总个数，以及查询中包含的变量的总个数，有效值范围为（2000，100000000）
CTAB_SEL_WITH_CONS	0	动态，系统级	查询建表时，是否对原始表上的约束进行拷贝。1：是，0：否。
HLDR_BUF_SIZE	8	动态，系统级	进行 HUGE 表导入时，缓冲区的大小，单位 MB，有效值范围（4~1024）

HLDR_BUF_TOTAL_SIZE	4294967294	动态，系统级	HLDR 资源控制，系统所有 HLDR 使用的 HLDR_BUF 空间的总量，单位为 M。取值范围：100~ 4294967294
HLDR_REPAIR_FLAG	0	动态，系统级	使用 dmfldr 装载数据时如发生错误，对错误的处理方式：0：完全回滚方式，数据将回到装载前的状态；1：修复方式，系统将数据修复到最后一个完整装载的数据区
HLDR_FORCE_COLUMN_STORAGE	1	动态，系统级	装载最后一个数据区是否强制列存。1：是；0：否
HLDR_HOLD_RATE	1.5	动态，系统级	HLDR 最多保持重用 HLDR_BUF 的个数与目标表列数的比例。取值范围：1~65535
HLDR_MAX_RATE	2	动态，系统级	HLDR 同时使用的 HLDR_BUF 的最大个数与目标表列数的比例。取值范围：2~65535
HUGE_ACID	0	动态，系统级	是否支持 HUGE 表增删改与查询的并发，取值范围：0、1、2，默认值为 0 0：不支持 HUGE 表增删改与查询的并发 1：支持 HUGE 表增删改与查询的并发，操作符 HFLKUP 检查数据区是否上锁时使用二分查找 2：支持 HUGE 表增删改与查询的并发，操作符 HFLKUP 检查数据区是否上锁时使用遍历查找
HFS_CHECK_SUM	1	静态	是否为 HUGE 表数据区进行和校验：0：否；1：是
HBUF_DATA_MODE	0	静态	HUGE 表数据缓冲区中缓存数据的格式，取值范围：0、1，默认值为 0 0：格式与数据文件中的数据块格式一致 1：当数据块是压缩加密时，缓存解密解压后的数据
SEC_INDEX_PARALLEL_INSERT_FLAG	0	动态，系统级	二级索引是否使用并行插入优化标记，0：否；1：是
FILL_COL_DESC_FLAG	0	动态，系统级	服务器返回给客户端的结果集是否含有列描述信息，0：否；1：是
BTR_SPLIT_MODE	0	动态，系统级	B 树叶子节点分裂方式，0：按对半分裂；1：在插入点进行分裂
BLOB_OUTROW_REC_STOR	5	动态，系统级	大数据行外记录式存储配置参数。如果使用记录式存储，可以在一个数据页中存放多个大数据。有效值范围（0~10），0 表示不使用记录式存储；其他设置指定记录式存储的分组数，可提高并发效率
TS_RESERVED_EXTENTS	64	静态	系统为每个表空间提前预留的簇个数，减少系统在执行过程中申请不到空间的情况。有效值范围（2 ~ 1024）。 RAC 环境下，该参数所有站点需要保持一致
TS_SAFE_FREE_EXTENTS	512		系统认为安全的 FREE EXTENTS 空间，有效值范围（128 ~65534）。 RAC 环境下，该参数所有站点需要保持一致
TS_MAX_ID	8192		限制系统所支持的最大表空间 ID，有效值范围（5 ~ 65517），

			与系统实际最大 ID 比较，取大。 如果设置为 90，但是系统最大 ID 已经为 100 了，最后结果是 100。 RAC 环境下，该参数所有站点需要保持一致
TS_FIL_MAX_ID	255		限制每个表空间所支持的最大文件个数，范围（2 ~255）。 RAC 环境下，该参数所有站点需要保持一致
DECIMAL_FIX_STORAGE	0	动态，会话级	是否将长度为 1~18 的 DECIMAL/DEC 类型转换为定长方式存储。0：否；1：是。 若转换为定长存储，则 DECIMAL(P,S)的转换规则为：当 0<P<10 时，存储为 4 字节整形值；当 10<=P<19 时，存储为 8 字节整形值
ENABLE_HUGE_SECIND	1	动态，系统级	是否支持事务型 HUGE 表二级索引的维护，0：不支持；1：支持
PWR_FLUSH_PAGES	10000	动态，系统级	控制生成特殊 PWR 记录的频率，每写入 PWR_FLUSH_PAGES 页到磁盘，即生成一条特殊的 PWR 记录,标记之前的 PWR 记录是有效的。有效值范围（0~4294967294）
REDO_UNTIL_LSN	空串	手动	系统故障重启时，重做 REDO 日志的最大 LSN 值。 使用这个参数要十分慎重，一般只在数据库文件系统损坏，系统无法正常启动情况下使用，通过指定这个参数，将数据库恢复到稍早时间点，让数据库可以正常启动，以便抢救部分数据。 注意：使用这个参数后，会截断指定 LSN 之后的所有日志。另外，系统启动后，一定不要忘记将参数值重新修改为缺省值。
IGNORE_FILE_SYS_CHECK	0	静态	系统重启时，是否检查 SYSTEM/ROLL/MAIN 表空间文件系统；取值范围 0，1；默认 0，检查文件系统。1，不检查文件系统。在实际使用空间比较大情况下，可以考虑关闭文件系统检查，提高系统启动速度。
FILE_SCAN_PERCENT	100.0	动态，系统级	V$DATAFILE 查询时控制抽样比例，最多抽样 10000 页； 在 50 页以下的不抽样。有效值范围（0~100.0），0 相当于 100
STARTUP_CHECKPOINT	0	静态	系统启动时 REDO 后是否强制做完全检查点。0：不强制；1：强制
CHECK_SVR_VERSION	1	静态	数据库记录的执行码版本比当前 SERVER 版本高时，是否报错。0：不报错；1：报错，服务器不能启动成功
BAK_USE_AP	1	动态，系统级	备份还原实现策略。 1：DMAP 插件方式，要求必须启动 DMAP 服务，可支持第三方备份。DMAP 插件执行，改造了备份还原任务子系统，允许指定并行度，大幅提升了备份还原的效率，特别是加密、压缩的处理效率。

			2：无插件方式，不依赖 DMAP，由主进程（DMSERVER、DMRMAN）自身执行备份还原，但不支持第三方备份。
TRXID_UNCOVERED	0	动态，会话级	是否禁止二级索引覆盖。0：否，使用二级索引覆盖，当更新不涉及二级索引时，查询最大 TRXID 可能比聚集索引的最大 TRXID 小；1：是
预先装载表相关参数			
LOAD_TABLE	空串	手动	在服务器启动时预先装载的普通表的完整表名,即"模式名.表名",多个表之间用逗号分隔,最多可指定10个表
LOAD_HTABLE	空串	手动	在服务器启动时预先装载的HUGE表的完整表名，即"模式名.表名"，多个表之间用逗号分隔
客户端缓存			
CLT_CACHE_TABLES	空串	手动	指定可以在客户端缓存的表。表名必须带模式名前缀，如果表名或模式名中包含特殊字符，需要使用双引号包含。如果指定多个缓存表，须以逗号间隔。服务器最多支持指定100个可缓存表。为避免参数值太长导致INI文件分析困难，允许在INI文件中设置多行CLT_CACHE_TABLES参数
REDO 日志相关参数			
RLOG_CRC	0	静态	是否为日志页生成 CRC 校验码并进行校验。0：否；1：是
RLOG_BUF_SIZE	512	静态	单个日志缓冲区大小（以日志页个数为单位），取值只能为 2 的次幂值，最小值为 1，最大值为 20480
RLOG_POOL_SIZE	128	静态	最大日志缓冲区大小（以 M 为单位）。有效值范围（1~1024）
RLOG_PARALLEL_ENABLE	0	静态	是否启动并行日志，1：启用；0：不启用
RLOG_APPEND_LOGIC	0	静态	是否启用在日志中记录逻辑操作的功能，取值范围 0、1、2 0：不启用；1：如果有主键列，记录 UPDATE 和 DELETE 操作时只包含主键列信息，若没有主键列则包含所有列信息；2：不论是否有主键列，记录 UPDATE 和 DELETE 操作时都包含所有列的信息
RLOG_APPEND_SYSTAB_LOGIC	0	静态	是否启用在日志中记录系统表逻辑操作的功能，启用 RLOG_APPEND_LOGIC 后有效，取值范围 0、1。 0：不启用；1：启用
RLOG_RESERVE_SIZE	40960	动态，系统级	INSERT/DELETE/UPDATE 等操作预留的日志空间大小（以日志页个数为单位）。有效值范围（2048~262144）。 注：若 RLOG_RESERVED_SIZE 设置不足可能使得日志空间不够,则在 RLOG_CHECK_SPACE 为 1 时可能会导致服务器主动退出以保证日志

			文件不被破坏
RLOG_CHECK_SPACE	1	动态，系统级	是否检查日志空间，取值范围 0、1。 1：日志刷盘时，检查日志空间是否足够，如果不够，则生成错误日志并强制退出，以确保数据文件不被破坏。 0：如果系统配置为不预留日志空间，则 RLOG_CHECK_SPACE 要强制设置为 0，即不检查日志空间
RLOG_SAFE_SPACE	128	静态	安全的可用日志空间大小（以 M 为单位）。有效值范围(0 ~ 1024)。当系统的可用日志空间小于这个值时，自动触发检查点释放日志空间
RLOG_SAFE_PERCENT	25	静态	安全的可用日志空间比例（FREE_SPACE / TOTAL_SPACE * 100）。有效值范围(0 ~ 100)。当系统的 可用日志空间 * 100 / 系统日志总空间 小于这个值时，自动触发检查点释放日志空间
RLOG_SEND_APPLY_MON	64	静态	数据守护中，对于主库，用于指定统计最近 N 次主库到每个备库的归档发送时间；对于备库，用于指定统计最近 N 次备库重演日志的时间，N 为此参数设置的值。有效值范围(16~1024)
REDO_PRE_LOAD	128	静态	重做联机日志文件或者利用归档日志文件恢复时每次预加载的文件大小，单位 M，有效值范围（0~8192）。如果取值为 0，则不执行预加载优化
REDO_PWR_OPT	1	静态	系统故障重启时，是否启动 PWR 优化；取值范围 0，1；默认 1，不再重做已经写入磁盘数据页的 REDO 日志，提高故障重启速度。0，取消优化，严格按照 REDO 日志生成顺序依次重做 REDO 日志。
REDO_IGNORE_DB_VERSION	0	静态	启动重做 REDO 日志时，是否检查版本信息。0：忽略版本检查，直接使用新版本重做 REDO 日志；1：正常检查版本，不兼容的库会报错，需要使用对应版本启动并正常关闭后，再用新版本执行码启动
ELOG_REPORT_LINK_SQL	0	动态，会话级	是否记录 DBLINK 执行的 SQL 到服务器日志文件中。0：不记录；1：记录
REDOS_BUF_SIZE	1024	静态	备库日志堆积的内存限制，堆积的日志 BUF 占用内存超过此限制则延迟响应，等待重演释放部分内存后再响应。以兆为单位，有效值范围（0~65536M），0 表示无内存限制
REDOS_BUF_NUM	4096	静态	备库日志 BUF 允许堆积的数目限制，超过限制则延迟响应主库，等待堆积数减少后再响应。以个数为单位，有效值范围（0~99999）
REDOS_MAX_DELAY	1800	静态	备库重演日志 BUF 的时间限制，超过此限制则认为重演异常，服务器自动宕机，防止日志堆积、主库不能及时响应用户请求。以秒为单位，

			取值范围（0~7200）。0 表示无重做时间限制
REDOS_PRE_LOAD	32	静态	备库重演日志时预加载的日志 BUF 个数，备库在等待当前日志 BUF 重演完成之前，根据此参数指定的个数，提前解析后面 N 个重演日志任务并预加载数据页到缓存中，以加快备库的重演速度，避免备库出现大量的日志堆积。 取值范围（0~99999），默认值为 32，0 不会有预加载动作。
事务相关参数			
ISOLATION_LEVEL	1	静态	系统默认隔离级别。1：读提交；3：可串行化
DDL_WAIT_TIME	10	动态，会话级	DDL 操作的锁超时时间，以秒为单位。有效值范围（0~60）
MPP_WAIT_TIME	10	动态，会话级	设置 MPP 下默认的封锁等待超时，单位为秒，有效值为（0~600）
FAST_RELEASE_SLOCK	1	动态，系统级	是否启用快速释放 S 锁，1：启用；0：不启用
SESS_CHECK_INTERVAL	3	动态，会话级	循环检测会话状态的时间间隔，以秒为单位。有效值范围（1~60）
LOCK_TID_MODE	1	动态，系统级	SELECT FOR UPDATE 封锁方式。0：结果集记录小于 100 行，直接封锁 TID，否则升级为表锁； 1：不升级表锁,一律使用 TID 锁
NOWAIT_WHEN_UNIQUE_CONFLICT	0	静态	插入数据时，如果和未提交数据有 UNIQUE 约束的冲突，是否等待未提交事务结束，0：等待，直至未提交事务结束；1：不等待，立即返回错误
UNDO_EXTENT_NUM	16	静态	表示系统启动时，为每个工作线程分配的回滚簇个数。有效值范围（1~ 256）
MAX_DE_TIMEOUT	10	动态，会话级	C、JAVA 外部函数的执行超时时间，以秒为单位。有效值范围（1~3600）
TRANSACTIONS	75	静态	指定一个会话中可以并发的自治事务数量。有效值范围（1~1000）
MVCC_RETRY_TIMES	5	静态	指定发生 MVCC 冲突时的最大重试次数。有效值范围（1~4294967294） 注：MPP 下此参数无效，发生 MVCC 冲突时将直接报错
ENABLE_FLASHBACK	0	动态，系统级	是否启用闪回查询，0：不启用；1：启用
UNDO_RETENTION	90	动态，系统级	事务提交后回滚页保持时间，单位为秒。有效值范围（0~86400） 注：类型为 DOUBLE，可支持毫秒
PURGE_WAIT_TIME	100	动态，系统级	检测到系统清理动作滞后（待清理事务提交时间-当前系统时间>UNDO_RETENTION）情况下，系统等待时间（MS）。有效范围（0~ 60000），0 表示不等待
PSEG_PAGE_OPT	1	动态，系统级	回滚页 PURGE 时优化，0：不使用；1：使用
PSEG_RECV	1	动态，系	系统故障重启时，是否回滚活动事务并 PURGE

		统级	已经提交事务。1：是；0：否，跳过回滚活动事务和 PURGE 已经提交事务的步骤。在回滚表空间出现异常、损坏、系统无法正常启动时，可将 PSEG_RECV 设置为 0，让系统启动；但存在一定风险，未提交事务的修改将无法回滚，破坏事务的原子性；另外，已提交未 PURGE 的事务，将导致部分存储空间无法回收。
ENABLE_IGNORE_PURGE_REC	0	动态，会话级	当返回 EC_RN_NREC_PURGED（-7120）错误（回滚记录版本太旧，无法获取用户记录）时的处理策略；0：报错；1：忽略这一条记录，继续执行
ROLL_ON_ERR	0	动态，系统级	服务器执行出错时的回滚策略选择，0：回滚当前语句；1：回滚整个事务
XA_TRX_IDLE_TIME	60	动态，系统级	允许游离的 XA 事务活动的时间，单位为秒。有效值范围(30~300)
ENABLE_TMP_TAB_ROLLBACK	1	动态，系统级	临时表操作是否生成回滚记录，0：不生成；1：生成 注：置为 0 时，临时表的 DML 操作无法回滚
安全相关参数			
PWD_POLICY	2	动态，系统级	设置系统默认口令策略。0： 无策略；1： 禁止与用户名相同；2： 口令长度不小于 9；4：至少包含一个大写字母（A-Z）；8 ：至少包含一个数字（0－9）；16：至少包含一个标点符号（英文输入法状态下，除"和空格外的所有符号；若为其他数字，则表示配置值的和，如 3＝1+2,表示同时启用第 1 项和第 2 项策略。当 COMPATIBLE_MODE=1 时，PWD_POLICY 的实际值均为 0
ENABLE_ENCRYPT	1	静态	通讯加密所采用的方式。0：不加密；1：SSL 加密，此时如果没有配置好 SSL 环境，则通讯仍旧不加密；2：SSL 认证，不加密，此时如果服务器 SSL 环境没有配置则服务器无法正常启动，如果客户端 SSL 环境没有配置则无法连接服务器
ENABLE_UDP	0	静态	是否支持 UDP。0：否；1：单发单收；2：多发多收。缺省使用 TCP
UDP_MAX_IDLE	15	静态	UDP 最大等待时间，单位秒，超过则认为连接断开。取值范围(5~60)
UDP_BTU_COUNT	8	静态	UDP 最大不等待连续发送消息块数，当不连续差值大于它时则等待发送。取值范围(4~32)
ENABLE_AUDIT	0	动态，系统级	审计开关，0：关闭；1：打开普通审计；2 打开普通审计和实时审计
AUDIT_FILE_FULL_MODE	1	静态	审计文件满后的处理方式，1：删除文件；2：不删除文件，也不添加审计记录
AUDIT_MAX_FILE_SIZE	100	动态，系统级	审计文件的最大大小，以兆为单位。有效值范围（1~4096）
AUDIT_IP_STYLE	0	静态	审计记录 IP 字段显示格式。

			0：仅显示 IP 1：显示 IP 和主机名
ENABLE_OBJ_REUSE	0	静态	是否支持客体重用，0：不支持；1：支持。该参数设置仅安全版有效
ENABLE_REMOTE_OSAUTH	0	静态	是否支持远程操作系统认证，0：不支持；1：支持。 该参数设置仅安全版有效
MSG_COMPRESS_TYPE	2	静态	与客户端的通信消息是否压缩，0：不压缩；1：压缩；2：系统自动决定每条消息是否压缩
ENABLE_STRICT_CHECK	0	静态	是否检查存储过程中 EXECUTE IMMEDIATE 语句的权限，1：表示检查；0：不检查
MAC_LABEL_OPTION	1	动态，系统级	用于控制SP_MAC_LABEL_FROM_CHAR过程的使用范围。 0：只有SSO可以调用 1：所有用户都可以调用 2：所有用户可以调用，但是非SSO用户不会主动创建新的LABEL
LDAP_HOST	空串	手动	LDAP服务器IP
COMM_ENCRYPT_NAME	空串	静态	消息加密算法名。如果为空则不进行通信加密；如果给的加密算法名错误，则用使用加密算法DES_CFB。DM支持的加密算法名可以通过查询动态视图V$CIPHERS获取
COMM_VALIDATE	1	动态，系统级	是否对消息进行校验。0：不检验；1：检验
ENABLE_EXTERNAL_CALL	0	动态，系统级	是否允许创建或执行外部函数。0：不允许；1：允许
EXTERNAL_JFUN_PORT	6363	动态，系统级	执行JAVA外部函数时使用的dmagent的端口号。取值范围：128~65535
ENABLE_PL_SYNONYM	0	动态，系统级	是否可以通过全局同义词执行非系统用户创建的包或者存储过程。1是，0否。 0的情况下，在解析过程/包名时，如果借助了同义词，则这些对象要么是系统内部创建的，或者其创建者必须为系统用户，否则一律报错
ENABLE_DDL_ANY_PRIV	0	动态，系统级	是否可以授予和回收DDL相关的ANY系统权限。1是，0否
FORCE_CERTIFICATE_ENCRYPTION	0	静态	非加密通讯的情况下是否开启用户名密码强制证书解密。1 是，0 否。 私钥证书 dm_login.prikey 需要提前拷贝到 ini 参数 SYSTEM_PATH 指定的目录下
SEC_PRIV_MODE	0	静态	表示权限管理模式。 0：表示传统模式 1：表示专用机模式 2：表示 EVAL 测评模式
REGEXP_MATCH_PATTERN	0	动态，会话级	指定正则表达式的匹配模式。0:支持非贪婪匹配；1:仅支持贪婪匹配
兼容性相关参数			
BACKSLASH_ESCAPE	0	动态，会话级	语法分析对字符串中的反斜杠是否需要进行转义处理，0：不进行转义处理；1：进行转义处理

STR_LIKE_IGNORE_MATCH_END_SPACE	1	动态，会话级	LIKE 运算中是否忽略匹配串的结尾 0。0：不忽略；1：忽略
CLOB_LIKE_MAX_LEN	31	静态	LIKE 语句中 CLOB 类型的最大长度，单位 KB，有效值范围（8~102400）
EXCLUDE_DB_NAME	空串	静态	服务器可以忽略的数据库名列表，各数据库名以逗号“,”隔开，数据库名不需要加引号。
MS_PARSE_PERMIT	0	静态	是否支持 MS SQLSERVER 的语法。 0：不支持；1：支持； 2：在 MS_PARSE_PERMIT =1 的基础上，兼容 MS SQLSERVER 的查询项中支持“标识符=列名”或“@变量名=列名”用法。 当 COMPATIBLE_MODE=3 时， MS_PARSE_PERMIT 的实际值为 1
COMPATIBLE_MODE	0	静态	是否兼容其他数据库模式。0：不兼容，1：兼容 SQL92 标准，2：兼容 ORACLE，3：兼容 MS SQL SERVER，4：兼容 MYSQL，5：兼容 DM6，6：兼容 TERADATA
CASE_COMPATIBLE_MODE	1	动态，系统级	涉及不同数据类型的 CASE 运算，是否需要兼容 ORACLE 的处理策略。 0：不兼容； 1：兼容，本模式下，当函数 DECODE()中的多个 CASE 类型不一致时，DECODE 会从其中选择一个类型进行匹配 2：兼容，本模式下，当函数 DECODE()中的多个 CASE 类型不一致时，DECODE 根据第一个 CASE 的类型来决定匹配类型
EXCLUDE_RESERVED_WORDS	空串	静态	语法解析时，需要去除的保留字列表，保留字之间以逗号分隔
COUNT_64BIT	1	动态，会话级	COUNT 集函数的值是否设置为 BIGINT。0：否；1：是
CALC_AS_DECIMAL	0	静态	0：默认值，表示整数类型的除法、整数与字符或 BINARY 串的所有四则运算，结果都处理成整数 1：表示整数类型的除法全部转换为 DEC(0，0)处理 2：表示将整数与字符或 BINARY 串的所有四则运算都转换为 DEC(0，0)处理 备注：该参数只有在 USE_PLN_POOL 为 0 或 1 时有效。当 USE_PLN_POOL 为 2 或 3 时，按照 CALC_AS_DECIMAL=2 处理
CMP_AS_DECIMAL	0	静态	0：默认值，表示不对字符串与整型的比较结果类型做任何修改 1：表示字符串与整型转换为 BIGINT 比较数据溢出时，则转换为 DEC 进行比较 2：表示字符串与整型的比较强制转换为 DEC 进行比较

CAST_VARCHAR_MODE	1	动态，系统级	在字符串向整型转换进行比较时，是否将溢出值转为特殊伪值。0：否；1：是
PL_SQLCODE_COMPATIBLE	0	静态	默认值为 0；如果设置为 1，则 PL 的异常处理中，SQLCODE 的错误码值需要尽量与 ORACLE 一致
LEGACY_SEQUENCE	0	动态，系统级	序列兼容参数，0：与 ORACLE 兼容；1：与旧版本 DM 兼容，不推荐使用
DM6_TODATE_FMT	0	静态	TO_DATE 中的 HH 格式小时制，0：12 小时制；1：24 小时制（与 DM6 兼容）
PK_MAP_TO_DIS	0	动态，系统级	专门用于 MPP 环境中，在建表语句中指定了 PK 列，没有指定分布方式时，是否自动将 PK 列作为 HASH 分布列，将整个表转为 HASH 分布。1 是；0 否
DROP_CASCADE_VIEW	0	动态，会话级	删除表或者视图的时候级联删除视图，0：只删除表或者视图；1：删除表或者视图时删除关联的视图，当 COMPATIBLE_MODE=1 时，DROP_CASCADE_VIEW 的实际值为 1
PROXY_PROTOCOL_MODE	0	动态，系统级	控制代理模式的开启，0：不开启；1：开启。开启代理模式后，服务器才可以解析代理向服务器发送的代理协议消息，从而才可以获得真实客户端 IP， 并且设置限制连接的 IP 后会检查真实客户端 IP 是否有效，查询 V$SESSIONS 中的 CLNT_IP 时也会显示真实客户端 IP 地址
用户请求跟踪相关参数			
SQL_TRACE_MASK	1	动态，系统级	LOG 记录的语句类型掩码，是一个格式化的字符串，表示一个 32 位整数上哪一位将被置为 1，置为 1 的位则表示该类型的语句要记录，格式为：位号:位号:位号。列如：3:5:7 表示第 3，第 5，第 7 位上的值被置为 1。每一位的含义见下面说明（2~17 前提是:SQL 标记位 24 也要置）: 1 全部记录(全部记录并不包含原始语句) 2 全部 DML 类型语句 3 全部 DDL 类型语句 4 UPDATE 类型语句(更新) 5 DELETE 类型语句(删除) 6 INSERT 类型语句(插入) 7 SELECT 类型语句(查询) 8 COMMIT 类型语句(提交) 9 ROLLBACK 类型语句(回滚) 10 CALL 类型语句(过程调用) 11 BACKUP 类型语句(备份) 12 RESTORE 类型语句(恢复) 13 创建对象操作(CREATE DDL) 14 修改对象操作(ALTER DDL) 15 删除对象操作(DROP DDL) 16 授权操作(GRANT DDL) 17 回收操作(REVOKE DDL)

			22 绑定参数 23 存在错误的语句（语法错误，语义分析错误等） 24 是否需要记录执行语句 25 是否需要打印计划和语句执行的时间 26 是否需要记录执行语句的时间 27 原始语句（服务器从客户端收到的未加分析的语句） 28 是否记录参数信息，包括参数的序号、数据类型和值 29 是否记录事务相关事件
SVR_LOG_FILE_NUM	0	动态，系统级	总共记录多少个日志文件，当日志文件达到这个设定值以后，再生成新的文件时，会删除最早的那个日志文件，日志文件的命令格式为 LOG_COMMIT_时间.LOG。当这个参数配置成 0 时，则按传统的日志文件记录，也就是 LOG_COMMIT01.LOG 和 LOG_COMMIT02.LOG 相互切换着记录。有效值范围（0~1024）
SVR_LOG	0	动态，系统级	是否打开 SQL 日志功能，0：表示关闭；1：打开，按照 SQLLOG.INI 中配置记录 SQL 日志；2：打开，按文件中记录数量切换日志文件，日志记录为详细模式；3：打开，不切换日志文件，日志记录为简单模式，只记录时间和原始语句
SVR_LOG_NAME	SLOG_ALL	动态，系统级	使用 SQLLOG.INI 中预设的模式的名称
SVR_LOG_SWITCH_COUNT	100000	动态，系统级	一个日志文件中的 SQL 记录条数达到多少条之后系统会自动将日志切换到另一个文件中。有效值范围（1000~ 10000000）
SVR_LOG_ASYNC_FLUSH	0	动态，系统级	是否打开异步 SQL 日志功能，0：表示关闭；1：表示打开
SVR_LOG_MIN_EXEC_TIME	0	动态，系统级	详细模式下，记录的最小语句执行时间，单位为毫秒。执行时间小于该值的语句不记录在日志文件中。有效值范围（0~ 4294967294）
SVR_LOG_FILE_PATH	..\LOG	动态，系统级	日志文件所在的文件夹路径
系统跟踪相关参数			
GDB_THREAD_INFO	0	静态	系统强制 HALT 时，是否打印线程堆栈信息到日志文件中。0：不打印；1：打印
TRACE_PATH	SYSTEM_PATH	手动	存放系统 TRACE 文件的路径。不允许指定 ASM 路径。默认的 TRACE_PATH 是 SYSTEM_PATH；如果 SYSTEM_PATH 保存在 ASM 上，则../CONFIG_PATH/TRACE 作为 TRACE_PATH。
MONITOR 监控相关参数			
ENABLE_MONITOR	1	动态，系统级	用于打开或者关闭系统的监控功能。1：打开；0：关闭。

MONITOR_TIME	1	动态，系统级	用于打开或者关闭时间监控。该监控项的生效必须是在 ENABLE_MONITOR 打开的情况下。1：打开；0：关闭。
MONITOR_SYNC_EVENT	0	动态，系统级	用于打开或者关闭同步事件的监控。该监控项的生效必须是在 ENABLE_MONITOR 打开的情况下。1：打开；0：关闭。该参数涉及到的动态视图为 V$SYSTEM_EVENT、V$SESSION_EVENT。
MONITOR_SQL_EXEC	0	动态，会话级	操作符、虚拟机栈帧、执行计划节点的监控开关。该监控项的生效必须是在 ENABLE_MONITOR 打开的情况下。0：关闭监控；1：打开监控；2：打开监控，在 1 的基础上，又增加了表达式运行时操作符的统计。该参数涉及的动态视图为 V$STKFRM、V$SQL_PLAN_NODE、V$SQL_NODE_HISTORY。
MEMORY_MONITOR	0	静态	是否进行内存泄露监控。0：不进行；1：进行
ENABLE_FREQROOTS	0	静态	指定FAST_POOL的管理方式。0：静态，系统启动时一次性载入内容，之后不再变化；1：动态，根据MAX_FREQ_ROOTS和MIN_FREQ_CNT参数值动态调整FAST_POOL内容；2：静态+动态模式，先在系统启动时载入内容，之后运行时根据MAX_FREQ_ROOTS和MIN_FREQ_CNT参数值动态调整
MAX_FREQ_ROOTS	200000	静态	收集常用B树地址的最大数量，仅在ENABLE_FREQROOTS不为0时有效，有效值范围（0~10000000）
MIN_FREQ_CNT	100000	静态	最常使用 B 树地址的阈值，仅在 ENABLE_FREQROOTS 不为 0 时有效，有效值范围（0~10000000）
LARGE_MEM_THRESHOLD	1000	动态、系统级	大内存监控阈值。单位 K，取值范围（0~10000000）。其中 0~100 关闭统计，100 以上才统计。 一条 SQL 语句使用的内存值超过这个值，就认为是使用了大内存，此时开启大内存监控。使用了大内存的 SQL 语句记录在 V$LARGE_MEM_SQLS，V$SYSTEM_LARGE_MEM_SQLS 视图中。
ENABLE_MONITOR_DMSQL	1	动态,会话级	启用动态监控 SQL 执行时间功能标记， 0：不启用；1：监控。 只监控多条或复杂 SQL，例如包含引用包、嵌套子过程、子方法、动态 SQL 的 SQL 语句。监控的记过记录在 V$DMSQL_EXEC_TIME 视图中。
数据守护相关参数			
DW_UDP_PORT	0	手动	守护进程的 UDP 端口号,需要和 DMWATCH.INI 中的同名配置项保持一致。有效值范围（1024~65534），为 0 时表示没有配置 UDP 通信方式

			的守护进程，不允许和 DW_PORT 同时配置
INST_UDP_PORT	0	手动	实例接收守护进程消息的 UDP 端口号，需要和 DMWATCH.INI 中的同名配置项保持一致。有效值范围（1024~ 65534），为 0 时表示没有配置 UDP 通信方式的守护进程，不允许和 DW_PORT 同时配置
DW_ERROR_TIME	60	动态，系统级	服务器认定守护进程未启动的时间，有效值范围（0~1800），单位为 S，默认 60S。 如果服务器距离上次收到守护进程消息的时间间隔在设定的时间范围内，则认为守护进程处于活动状态，此时，不允许手工执行修改服务器模式、状态的 SQL 语句； 如果超过设定时间仍没有收到守护进程消息，则认为守护进程未启动，此时，允许手工方式执行这类 SQL 语句
DW_PORT	0	手动	服务器和守护进程之间的 TCP 通信端口，服务器监听此端口,接收守护进程的 TCP 连接请求。有效值范围（1024~ 65534），为 0 时表示没有配置 TCP 通信方式的守护进程，不允许和 DW_UDP_PORT/INST_UDP_PORT 同时配置
ALTER_MODE_STATUS	1	动态，系统级	是否允许手工修改服务器的模式和状态，1：允许；0：不允许。 数据守护环境下建议配置为 0，实例处于主机或备机模式后，不允许用户直接通过 SQL 语句修改服务器的模式和状态
ENABLE_OFFLINE_TS	1	动态，系统级	是否允许 OFFLINE 表空间，0：不允许；1：允许；2：备库不允许。 数据守护环境下建议配置为 2
SESS_FREE_IN_SUSPEND	60	动态，系统级	远程归档失败会导致系统挂起，为了防止应用的连接一直挂住不切换到新主库，设置该参数，表示归档失败挂起后隔一段时间自动断开所有连接。有效值范围（0-1800），单位为 S，0 表示不断开。
SUSPEND_WORKER_TIMEOUT	600	动态，系统级	在一些ALTER DATABASE等操作过程中，需要先暂停所有工作线程，如果超过此设置时间，仍有部分工作线程在执行中，则会报错，并终止当前操作。有效值范围（60~3600），单位为 S
SUSPENDING_FORBIDDEN	0	手动	是否禁止挂起工作线程，0：不禁止；1：禁止
全文索引相关参数			
CTI_HASH_SIZE	100000	动态，会话级	使用全文索引查询时，用于设置关键字匹配的哈希表大小。有效值范围（1000~10000000）
CTI_HASH_BUF_SIZE	50	动态，会话级	使用全文索引查询时，用于设置哈希缓存内存大小，以 M 为单位。有效值范围(1~4000)
USE_FORALL_ATTR	0	动态，会话级	是否使用 FORALL 语句的游标属性，0：不使用；1：使用
ALTER_TABLE_OPT	0	动态，会	是否对加列、修改列、删除列操作进行优化，0：

		话级	全部不优化：1：全部优化；2：打开快速加列，对于删除列和修改列与1等效
DCP 相关参数			
ENABLE_DCP_MODE	0	静态	服务器是否作为 DCP 集群代理运行，0：否；1：是
DCP_PORT_NUM	5237	静态	DCP 管理端口号
DCP_CONN_POOL_SIZE	1000	动态，系统级	DCP 连接池大小，有效值范围（1~100000）
配置文件相关参数			
MAL_INI	0	静态	是否启用 MAL 系统，0：不启用；1：启用
ARCH_INI	0	动态，系统级	是否启用归档，0：不启用；1：启用
REP_INI	0	静态	是否启用复制，0：不启用；1：启用
LLOG_INI	0	静态	是否启用逻辑日志，0：不启用；1：数据复制使用
TIMER_INI	0	静态	是否启用定时器，0：不启用；1：启用
MPP_INI	0	静态	是否启用 MPP 系统，0：不启用；1：启用
其他			
IDLE_MEM_THRESHOLD	50	动态，系统级	可用物理内存的报警阀值，单位为兆，有效值范围（10~6000）
IDLE_DISK_THRESHOLD	1000	动态，系统级	磁盘可用空间的报警阀值，单位为兆，有效值范围（50~50000）
IDLE_SESS_THRESHOLD	5	动态，系统级	有限的会话数阀值，有效值范围（1~10）
ENABLE_PRISVC	0	静态	是否启用服务优先级的功能，0：不启用；1：启用
ENABLE_INJECT_HINT	0	动态，会话级	是否启用 SQL 指定 HINT 的功能，0：不启用；1：启用
RAC 相关参数			
RAC_N_CTLS	10000	静态	LBS/GBS 控制页数目。有效值范围（10000~4294967294） 注：系统可能会根据 MAX_BUFFER 重新计算此参数的值，所有 DMDSC 节点要确保 RAC_N_CTLS、MAX_BUFFER 设置值相同
RAC_N_POOLS	1	静态	LBS/GBS 池数目。有效值范围（1~1024）
RAC_ENABLE_MONITOR	1	动态，系统级	是否启用 RAC 请求时间监控，0：不启用；1：启用。监控的内容参见 V$RAC_REQUEST_STATISTIC 和 V$RAC_REQUEST_PAGE_STATISTIC
HA_INST_CHECK_FLAG	1	手动	是否启用 DMINST.SYS 文件检测多个实例同时启动同一份数据库数据，0：不启用；1：启用
HA_INST_CHECK_TIME	0	手动	DMINST.SYS 文件检测校验次数，有效值范围（3~180）
HA_INST_CHECK_IP	需要时指定	手动	HA 实例启动检测 IP，是指 HA 系统中另外一台机器的 IP 地址，考虑到多网卡情况，最多允许配置 5 个 IP。如果需要配置多个 IP，则编辑

			多个配置项即可，比如： HA_INST_CHECK_IP = 192.168.0.8 HA_INST_CHECK_IP = 192.168.0.9
HA_INST_CHECK_PORT	65534	手动	HA 实例监听端口，数据库实例启动后，监听此端口的连接请求，并使用此端口号连接 HA_INST_CHECK_IP 判断另外一个 HA 实例是否已经启动。 只允许配置一个监听端口，所有 HA_INST_CHECK_IP 使用同一个端口进行检测，有效值范围(1024 ~ 65534)

2) dmmal.ini

dmmal.ini 是 MAL 系统的配置文件。dmmal.ini 的配置项见表 2.2。需要用到 MAL 环境的实例，所有站点 dmmal.ini 需要保证严格一致。

表 2.2 dmmal.ini 的配置项

项目	项目意义	字段	字段意义
MAL_CHECK_INTERVAL	检测线程检测间隔，默认 30S，范围（0-1800），如果配置为 0，则表示不进行链路检测	无	无
MAL_CONN_FAIL_INTERVAL	检测线程认定链路断开的时间，默认 10S，范围（2-1800）	无	无
MAL_LEAK_CHECK	是否打开 MAL 内存泄露检查（0：关闭，1：打开），默认 0	无	无
MAL_LOGIN_TIMEOUT	MPP/DBLINK 等实例间登录时的超时检测间隔（3-1800），以秒为单位，默认 15s	无	无
MAL_BUF_SIZE	单个 MAL 缓存大小限制，以兆为单位。当此 MAL 的缓存邮件超过此大小，则会将邮件存储到文件中。有效值范围（0~500000），默认为 100	无	无
MAL_SYS_BUF_SIZE	MAL 系统总内存大小限制，单位：M。有效值范围（0~500000），默认为 0，表示 MAL 系统无总内存限制	无	无
MAL_VPOOL_SIZE	MAL 系统使用的内存初始化大小，以兆为单位。有效值范围（1~500000），默认为 10，此值一般要设置的比 MAL_BUF_SIZE 大一些	无	无

MAL_COMPRESS_LEVEL	Mal 消息压缩等级，取值范围 0-10。0 代表不进行消息压缩，为默认值 1 – 9 表示采用 lz 算法，从 1 到 9 表示压缩速度依次递减，压缩率依次递增。 10 表示采用 QUICKLZ 算法，压缩速度高于 LZ 算法，压缩率相对低	无	无
MAL_TEMP_PATH	指定临时文件的目录。当邮件使用的内存超过 MAL_BUF_SIZE或者 MAL_SYS_BUF_SIZE 时，将新产生的邮件保存到临时文件中。如果缺省，则新产生的邮件保存到TEMP.DBF文件中	无	无
[MAL_INST1]	实例配置	MAL_INST_NAME	实例名。MAL 系统中的各数据库实例不允许同名。
		MAL_HOST	实例主库
		MAL_PORT	监听端口
		MAL_INST_PORT	数据库实例服务器的端口，对于每个实例，一定要和对应 DM.INI 中的 PORT_NUM 保持一样。（在 MPP 下以及读写分离的即时归档主备系统下一定要配置）
		MAL_INST_HOST	连接数据库服务器使用的 IP 地址。（在 MPP 下以及读写分离的即时归档主备系统下一定要配置）
		MAL_DW_PORT	服务器对应的守护进程使用的 TCP 端口，用于守护进程之间，以及守护进程和监视器之间的 TCP 通信（控制文件方式的数据守护环境下需要配置）。
		MAL_LINK_MAGIC	MAL 链路网段标识，有效值范围（0-65535），默认 0。设置此参数时，同一网段内的节点都设置相同，不同网段内的节点设置的值必须不一样。

注意：dmmal.ini 和 dm.ini 中都可配置 MAL_LEAK_CHECK，启动时以 dmmal.ini 中设置为准，如果 dmmal.ini 中没有配置，则以 dm.ini 中配置为准，两个都没配置时，则默认 0。此参数在 dm.ini 中可动态更改。

3) dmarch.ini

dmarch.ini 用于本地归档和远程归档。dmarch.ini 的配置项见表 2.3。

表 2.3 dmarch.ini 的配置项

项目	项目意义	字段	字段意义
		ARCH_WAIT_APPLY	是否需要等待备库做完日志，1 表示需要，0 表示不需要，默认为 1
[ARCHIVE_LOCAL1]	本地归档配置	ARCH_TYPE	归档类型
		ARCH_DEST	归档路径
		ARCH_FILE_SIZE	单个归档文件大小，单位 MB，取值范围（64~2048），默认为 1024MB，即 1G
		ARCH_SPACE_LIMIT	归档文件空间限制，单位 MB，取值范围（1024~4294967294），0 表示无空间限制
ARCHIVE_REALTIME	实时归档配置	ARCH_TYPE	归档类型
		ARCH_DEST	归档目标实例名
ARCHIVE_ASYNC	异步归档	ARCH_TYPE	归档类型
		ARCH_DEST	归档目标实例名
		ARCH_TIMER_NAME	定时器名称
ARCHIVE_TIMELY	即时归档	ARCH_TYPE	归档类型
		ARCH_DEST	归档目标实例名
REMOTE	远程归档配置	ARCH_TYPE	归档类型
		ARCH_DEST	归档目标实例名
		ARCH_FILE_SIZE	单个归档文件大小，单位 MB，取值范围（64~2048），默认为 1024MB，即 1G
		ARCH_SPACE_LIMIT	归档文件空间限制，单位 MB，取值范围（1024~4294967294），0 表示无空间限制
		ARCH_INCOMING_PATH	对应远程归档存放在本节点的实际路径

相关说明

1. 归档类型 ARCH_TYPE 有以下几种：
   - 本地归档　　LOCAL（一台主库最多配 8 个）
   - 远程实时归档　　REALTIME（一台主库最多配 8 个）
   - 远程异步归档　　ASYNC（一台主库最多配 8 个）
   - 即时归档　　TIMELY（一个主库最多配 8 个）
   - 远程归档　　REMOTE（一个主库最多配 8 个）
2. 配置名[ARCHIVE_*]表示归档名，在配置文件中必须唯一。
3. 不能存在相同实例名的不同归档；
4. 不能存在 DEST 相同的不同归档实例；
5. ARCH_TIMER_NAME 为定制的定时器名称，定时器配置见 dmtimer.ini；
6. ARCH_SPACE_LIMIT 表示归档文件的磁盘空间限制，如果归档文件总大小超过这个值，则在生成新归档文件前会删除最老的一个归档文件。

4) dm_svc.conf

DM 安装时生成一个配置文件 dm_svc.conf，不同的平台所在目录有所不同。

1． 32 位的 DM 安装在 Win32 操作平台下，此文件位于%SystemRoot%\system32 目录；

2． 64 位的 DM 安装在 Win64 操作平台下，此文件位于%SystemRoot%\system32 目录；

3． 32 位的 DM 安装在 Win64 操作平台下，此文件位于%SystemRoot%\SysWOW64 目录；

4． 在 Linux 平台下，此文件位于/etc 目录。

dm_svc.conf 文件中包含 DM 各接口及客户端需要配置的一些参数，具体的配置项如表 2.4 所示。

表 2.4 dm_svc.conf 配置项介绍

配置项	缺省值	简述
服务名	无	连接服务名，参数值格式为 IP[:PORT],IP[:PORT],.....
TIME_ZONE	操作系统时区	指明客户端的默认时区，设置范围为：-779~840M，如 60 对应+1:00 时区
LANGUAGE	操作系统语言	当前数据库服务器使用的语言，会影响帮助信息错误和提示信息。支持的选项为：CN（表示中文）和 EN（表示英文）。可以不指定，若不指定，系统会读取操作系统信息获得语言信息，建议有需要才指定。
CHAR_CODE	操作系统编码格式	客户端使用的编码格式，会影响帮助信息和错误提示信息，要与客户端使用的编码格式一致。支持的选项为：PG_UTF8（表示 UTF8 编码）；PG_GBK/PG_GB18030（两者都表示 GBK 编码）；PG_BIG5（表示 BIG5 编码）；PG_ISO_8859_9（表示 ISO88599 编码）；PG_EUC_JP（表示 EUC_JP 编码）；PG_EUC_KR（表示 EUC_KR 编码）；PG_KOI8R（表示 KOI8R 编码）；PG_ISO_8859_1（表示 ISO_8859_1 编码）。 可以不指定，若不指定，系统会读取操作系统信息获得编码信息，建议有需要才指定。
COMPRESS_MSG	0	是否启用消息压缩。0：不启用；1：启用
LOGIN_ENCRYPT	1	是否进行通信加密。0：不加密；1：加密
DIRECT	Y	是否使用快速装载。Y：使用；N：不使用
DEC2DOUB	0	指明在 DPI、DMODBC、DCI、DMPHP 和 DM PRO*C 中，是否将 DEC 类型转换为 DOUBLE 类型。0：不转换；1：转换
KEYWORDS	无	标识用户关键字，所有在列表中的字符串，如果以单词的形式出现在 SQL 语句中，则这个单词会被加上双引号。该参数主要用来解决用户需要使用 DM7 中的保留字作为对象名使用的状况。
ENABLE_RS_CACHE	0	是否进行客户端结果集缓存。0：不进行；1：进行
RS_CACHE_SIZE	10	设置结果集缓冲区大小，以 M 为单位。有效值为 1~65535，如果设置太大，可能导致空间分配失败，进而使缓存失效
RS_REFRESH_FREQ	10	结果集缓存检查更新的频率，以秒为单位，有效值为 0~10000，如果设置为 0，则不需检查更新
CONNECT_TIMEOUT	5000	连接超时时间，单位为毫秒。0 表示无限制
LOGIN_MODE	0	是否仅登录到主库或备库。0：主库不存在的情况下可连接备

		库；1：只连接主库；2：只连接备库；3：存在备库时优先连接备库
SWITCH_TIME	3	在服务器之间切换的次数，有效值范围 1~9223372036854775807
SWITCH_INTERVAL	200	在服务器之间切换的时间间隔，单位为毫秒，有效值范围 1~9223372036854775807
RW_SEPARATE	0	是否启用读写分离。0：不启用；1：启用
RW_PERCENT	25	读写分离分发比例，有效值范围 0~100
FORCE_CERTIFICATE_ENCRYPTION	0	非加密通讯的情况下是否开启用户名密码强制证书加密。1 是，0 否
DEXP 配置项		
DUMMY	2	写文件时，发现文件已存在的处理方式。0：报错；1：直接覆盖文件；2：询问用户，有交互信息
DPC_NEW 配置项		
DPC_TRACE	无	DPC_NEW 的 TRACE 文件路径，不配置时不写 TRACE 信息
ENABLE_SSL	0	是否启用 SSL。1/Y/Y：启用 SSL；其他值：不启用
SSL_CONFIG	无	在启用 SSL 的前提下，设置 SSL 值，格式如下：SSL_CONFIG=((USER=(用户名 1) SSL_PATH=(SSL 路径 1) SSL_PWD=(SSL KEY1)) (USER=(用户名 2) SSL_PATH=(SSL 路径 2) SSL_PWD=(SSL KEY2)))
.NET PROVIDER 配置项		
TRACE	NONE	是否启用.NET PROVIDER 的 TRACE 功能。NONE：不启用；DEBUG：打印到控制台；NORMAL：打印到执行目录下的“PROVIERTRACE.TXT”文件中；TRACE：打印到执行目录下的“PROVIERTRACE.TXT”文件中，比 NORMAL 内容要更详细一些；THREAD：每个线程的 TRACE 分别打印到执行目录下的“PROVIERTRACE 线程号.TXT”文件中

dm_svc.conf 配置文件的内容分为全局配置区和服务配置区。全局配置区在前，可配置表 2.4 中所有的配置项，服务配置区在后，以“[服务名]”开头，可配置除了服务名外的所有配置项。服务配置区中的配置优先级高于全局配置区。

下面是一个 dm_svc.conf 的例子：

```
#  以#开头的行表示是注释
#  全局配置区
O2000=(192.168.0.1:5000,192.168.0.2:5236)
O3000=(192.168.0.1:5236,192.168.0.3:4350)
TIME_ZONE=(+480)     #表示+8:00 时区
LOGIN_ENCRYPT=(0)
DIRECT=(Y)

#  服务配置区
[O2000]
TIME_ZONE=(+540)     #表示+9:00 时区
LOGIN_MODE=(2)
SWITCH_TIME=(3)
SWITCH_INTERVAL=(10)
```

需要说明的是，如果对 dm_svc.conf 的配置项进行了修改，需要重启客户端程序，修

改的配置才能生效。

5) sqllog.ini

sqllog.ini用于sql日志的配置，当且仅当INI参数SVR_LOG=1时使用。

如果在服务器启动过程中，修改了sqllog.ini文件。修改之后的文件，只要调用过程SP_REFRESH_SVR_LOG_CONFIG() 就会生效。

sqllog.ini的详细配置请参考表2.5。

表 2.5 sqllog.ini 的配置项

参数名	缺省值	属性	说明
SQL_TRACE_MASK	1	动态，系统级	LOG 记录的语句类型掩码，是一个格式化的字符串，表示一个 32 位整数上哪一位将被置为 1，置为 1 的位则表示该类型的语句要记录，格式为：位号:位号:位号。列如：3:5:7 表示第 3，第 5，第 7 位上的值被置为 1。每一位的含义见下面说明（2~17 前提是:SQL 标记位 24 也要置):: 1 全部记录(全部记录并不包含原始语句) 2 全部 DML 类型语句 3 全部 DDL 类型语句 4 UPDATE 类型语句(更新) 5 DELETE 类型语句(删除) 6 INSERT 类型语句(插入) 7 SELECT 类型语句(查询) 8 COMMIT 类型语句(提交) 9 ROLLBACK 类型语句(回滚) 10 CALL 类型语句(过程调用) 11 BACKUP 类型语句(备分) 12 RESTORE 类型语句(恢复) 13 创建对象操作(CREATE DDL) 14 修改对象操作(ALTER DDL) 15 删除对象操作(DROP DDL) 16 授权操作(GRANT DDL) 17 回收操作(REVOKE DDL) 22 绑定参数 23 存在错误的语句（语法错误，语义分析错误等） 24 是否需要记录执行语句 25 是否需要打印计划和语句和执行的时间 26 是否需要记录执行语句的时间 27 原始语句（服务器从客户端收到的未加分析的语句） 28 是否记录参数信息，包括参数的序号、数据类型和值 29 是否记录事务相关事件
FILE_NUM	0	动态，系统级	总共记录多少个日志文件，当日志文件达到这个设定值以后，再生成新的文件时，会删除最早的

			那个日志文件，日志文件的命令格式为 DMSQL_实例名_日期时间.LOG。 当这个参数配置成 0 时，只会生成两个日志相互切换着记录。有效值范围（0~1024）。例如，当 FILE_NUM=0，实例名为 PDM 时，根据当时的日期时间，生成的日志名称为： DMSQL_PDM_20180719_163701.LOG， DMSQL_PDM_20180719_163702.LOG
SWITCH_MODE	0	手动	表示 SQL 日志文件切换的模式： 0：不切换 1：按文件中记录数量切换 2：按文件大小切换 3：按时间间隔切换
SWITCH_LIMIT	100000	动态，系统级	不同切换模式 SWITCH_MODE 下，意义不同： ◆ 按数量切换时，一个日志文件中的 SQL 记录条数达到多少条之后系统会自动将日志切换到另一个文件中。一个日志文件中的 SQL 记录条数达到多少条之后系统会自动将日志切换到另一个文件中。有效值范围（1000~ 10000000） ◆ 按文件大小切换时，一个日志文件达到该大小后，系统自动将日志切换到另一个文件中，单位为 M。有效值范围（1~ 2000） ◆ 按时间间隔切换时，每个指定的时间间隔，按文件新建时间进行文件切换，单位为分钟。有效值范围（1~ 30000）
ASYNC_FLUSH	0	动态，系统级	是否打开异步 SQL 日志功能。0：表示关闭；1：表示打开
MIN_EXEC_TIME	0	动态，系统级	详细模式下，记录的最小语句执行时间，单位为毫秒。执行时间小于该值的语句不记录在日志文件中。有效值范围（0~ 4294967294）
FILE_PATH	..\LOG	动态，系统级	日志文件所在的文件夹路径
BUF_TOTAL_SIZE	10240	动态，系统级	SQL 日志 BUFFER 占用空间的上限，单位为 KB，取值范围（1024~1024000）
BUF_SIZE	1024	动态，系统级	一块 SQL 日志 BUFFER 的空间大小，单位为 KB，取值范围（50~409600）
BUF_KEEP_CNT	6	动态，系统级	系统保留的 SQL 日志缓存的个数，有效值范围（1~ 100）
PART_STOR	0	手动	SQL 日志分区存储，表示 SQL 日志进行分区存储的划分条件。 0 表示不划分；1 表示 USER：根据不同用户分布存储

ITEMS	0	手动	配置 SQL 日志记录中的那些列要被记录。 该参数是一个格式化的字符串，表示一个记录中的那些项目要被记录，格式为：列号:列号:列号。列如：3:5:7 表示第 3，第 5，第 7 列要被记录。0 表示记录所有的列 1 TIME 执行的时间 2 SEQNO 服务器的站点号 3 SESS 操作的 SESS 地址 4 USER 执行的用户 5 TRXID 事务 ID 6 STMT 语句地址 7 APPNAME 客户端工具 8 IP 客户端 IP 9 STMT_TYPE 语句类型 10 INFO 记录内容 11 RESULT 运行结果，包括运行用时和影响行数（可能没有）
USER_MODE	0	手动	SQL 日志按用户过滤时的过滤模式，取值 0：关闭用户过滤 1：白名单模式，只记录列出的用户操作的 SQL 日志 2：黑名单模式，列出的用户不记录 SQL 日志
USERS	空串	手动	打开 SVR_LOG_USER_MODE 时指定的用户列表。格式为：用户名：用户名：用户名

sqllog.ini 中配置块的使用方法：

sqllog.ini 中配置块在 INI 参数 SVR_LOG=1 时使用。且 INI 参数 SVR_LOG_NAME 必须和 sqllog.ini 中的 SVR_LOG_NAME 名称一样，sqllog.ini 配置块才会生效。若 SVR_LOG 为 1，但不存在 sqllog.ini 或 sqllog.ini 配置错误，则仍旧使用 dm.ini 中的“用户请求跟踪相关参数”。

例如：下面是一个 SVR_LOG_NAME 为 SLOG_ALL 的 sqllog.ini 的例子：

```
    BUF_TOTAL_SIZE              = 10240             #SQLs Log Buffer Total
Size(K)(1024~1024000)
    BUF_SIZE             = 1024         #SQLs Log Buffer Size(K)(50~409600)
    BUF_KEEP_CNT          = 6           #SQLs Log buffer keeped count(1~100)

    [SLOG_ALL]
       FILE_PATH    = ..\log
       PART_STOR    = 0
       SWITCH_MODE  = 2
       SWITCH_LIMIT   = 128
       ASYNC_FLUSH   = 1
       FILE_NUM = 5
       ITEMS    = 0
       SQL_TRACE_MASK  = 1
       MIN_EXEC_TIME = 0
       USER_MODE   = 0
```

```
        USERS =
```

2.1.2 复制配置

1) dmrep.ini

dmrep.ini 用于配置复制实例，具体内容请见“数据复制”章节。dmrep.ini 的配置项见表 2.6。

表 2.6 dmrep.ini 的配置项

项目	项目意义	字段	字段意义
[REP_RPS_INST_NAME]	该复制结点所属的复制服务器的实例名		
[REP_MASTER_INFO]	作为主服务器的相关信息，其中一条记录就是一个复制关系	REP_ID	复制关系的 ID；同时也是逻辑日志的 ID，由复制服务器生成
[REP_SLAVE_INFO]	作为从服务器的相关信息，其中一条记录就是一个复制关系	REP_ID	复制关系的 ID
		MASTER_INSTNAME	复制关系主服务器的实例名
		ADD_TICK	复制关系创建的时间，用于检验复制关系的正确性
[REP_SLAVE_TAB_MAP]	作为从服务器时，记录的复制映射的信息，其中一条记录就是一个复制映射	REP_ID	映射所属的复制的 ID
		SRC_TAB_ID	复制源对象表的 ID
		DST_TAB_ID	复制目标对象表的 ID
		READONLY_MODE	1：只读模式，0：非只读模式
[REP_SLAVE_SRC_COL_INFO]	作为从服务器时，记录的复制映射中使用的复制源对象的列信息	REP_ID	所属映射所在的复制关系的 ID
		SRC_TAB_ID	所属映射的复制源对象的表 ID
		COL_ID	列 ID
		SQL_PL_TYPE	列类型
		LEN	长度
		PREC	精度

2) dmllog.ini

dmllog.ini 用于配置逻辑日志，具体内容请见“数据复制”章节。dmllog.ini 的配置项见表 2.7。

表 2.7 dmllog.ini 的配置项

项目	项目意义	字段	字段意义
[LLOG_INFO]	逻辑日志的整体信息	ID	逻辑日志 ID，若参与复制，则与复制关系 ID 相同；-1 全局逻辑

			日志
		LLOG_PATH	逻辑日志本地归档路径
		LOCAL_ARCH_SPACE_LIMIT(G)	本地归档文件的空间限制，以 G 为单位，-1 表示无限制
		REMOTE_ARCH_FLAG	逻辑日志远程归档是否有效标识，0 表示远程归档无效；1 表示远程归档有效
		REMOTE_ARCH_INSTNAME	远程归档的归档实例名，若参与复制，即为从服务器的实例名
		REMOTE_ARCH_TYPE	远程归档的类型，同步或异步
		REMOTE_ARCH_TIMER	远程归档使用的定时器，同步时无意义
[LLOG_TAB_MAP]	逻辑日志记录的对象的信息	LLOG_ID	逻辑日志的 ID
		SCH_ID	逻辑日志记录的模式 ID
		TAB_ID	逻辑日志记录的表 ID

3) dmtimer.ini

dmtimer.ini 用于配置定时器，用于数据守护中记录异步备库的定时器信息或数据复制中记录异步复制的定时器信息。dmtimer.ini 的配置项见表 2.8。

表 2.8 dmtimer.ini 的配置项

项目	项目意义	字段	字段意义
[TIMER_NAME1]	定时器信息	TYPE	定时器调度类型： 1：执行一次 2：按日执行 3：按周执行 4：按月执行的第几天 5：按月执行的第一周 6：按月执行的第二周 7：按月执行的第三周 8：按月执行的第四周 9：按月执行的最后一周 10:日历表达式
		FREQ_MONTH_WEEK_INTERVAL	间隔月或周数
		FREQ_SUB_INTERVAL	间隔天数
		FREQ_MINUTE_INTERVAL	间隔分钟数
		REPEAT_INTERVAL	日历表达式。语法详见《DM8 系统包使用手册》手册 DBMS_SCHEDULER 包相关章节
		START_TIME	开始时间
		END_TIME	结束时间
		DURING_START_DATE	开始时间点
		DURING_END_DATE	结束时间点
		NO_END_DATE_FLAG	是否结束标记
		DESCRIBE	定时器描述
		IS_VALID	有效标记

2.2 控制文件

每个 DM 数据库都有一个名为 dm.ctl 的控制文件。控制文件是一个二进制文件，它记录了数据库必要的初始信息，其中主要包含以下内容：

1. 数据库名称；
2. 数据库服务器模式；
3. OGUID 唯一标识；
4. 数据库服务器版本；
5. 数据文件版本；
6. 数据库的启动次数；
7. 数据库最近一次启动时间；
8. 表空间信息，包括表空间名，表空间物理文件路径等，记录了所有数据库中使用的表空间，数组的方式保存起来；
9. 控制文件校验码，校验码由数据库服务器在每次修改控制文件后计算生成，保证控制文件合法性，防止文件损坏及手工修改。

在服务器运行期间，执行表空间的 DDL 等操作后，服务器内部需要同步修改控制文件内容。如果在修改过程中服务器故障，可能会导致控制文件损坏，为了避免出现这种情况，在修改控制文件时系统内部会执行备份操作。备份策略如下：

◆ 策略一 在修改 dm.ctl 之前，先执行一次备份，确定 dm.ctl 修改成功后，再将备份删除，如果 dm.ctl 修改失败或中途出现故障，则保留备份文件。

◆ 策略二 在修改 dm.ctl 成功之后，根据 dm.ini 中指定的 CTL_BAK_PATH/CTL_BAK_NUM 对最新的 dm.ctl 执行备份，如果用户指定的 CTL_BAK_PATH 是非法路径，则不再生成备份文件，在路径有效的情况下，生成备份文件时根据指定的 CTL_BAK_NUM 判断是否删除老的备份文件。

注意：

- 如果 dm.ctl 文件存放在裸设备上，则【策略一】不会生效。
- 如果指定的 CTL_BAK_PATH 是无效路径，则【策略二】也不会生效。
- 如果【策略一】和【策略二】的条件都满足，则都会生效执行，否则只执行满足条件的备份策略，如果都不满足，则不会再生成备份文件。
- 如果是初始化新库，在初始化完成后，会在“SYSTEM_PATH/CTL_BAK”路径下对原始的 dm.ctl 执行一次备份。

2.3 数据文件

数据文件以 dbf 为扩展名，它是数据库中最重要的文件类型，一个 DM 数据文件对应磁盘上的一个物理文件，数据文件是真实数据存储的地方，每个数据库至少有一个与之相关的数据文件。在实际应用中，通常有多个数据文件。

当 DM 的数据文件空间用完时，它可以自动扩展。可以在创建数据文件时通过 MAXSIZE 参数限制其扩展量，当然，也可以不限制。但是，数据文件的大小最终会受物理磁盘大小的限制。在实际使用中，一般不建议使用单个巨大的数据文件，为一个表空间创建多个较小的数据文件是更好的选择。

数据文件在物理上按照页、簇和段的方式进行管理，详细结构请参考第一章的内容。

数据文件按数据组织形式，可以分为如下几种：

1. B 树数据

行存储数据，也是应用最广泛的存储形式，其数据是按 B 树索引组织的。普通表、分区

表、B 树索引的物理存储格式都是 B 树。

一个 B 树包含两个段，一个内节点段，存放内节点数据；一个叶子段，存放叶子节点数据。其 B 树的逻辑关系由段内页面上的记录，通过文件指针来完成。

当表上没有指定聚簇索引时，系统会自动产生一个唯一标识 rowid 作为 B 树的 key 来唯一标识一行。

2． 堆表数据

堆表的数据是以挂链形式存储的，一般情况下，支持最多 128 个链表，一个链表在物理上就是一个段，堆表采用的是物理 rowid，在插入过程中，rowid 在事先已确定，并保证其唯一性，所以可以并发插入，插入效率很高，且由于 rowid 是即时生成，无需保存在物理磁盘上，也节省了空间。

3． 列存储数据

数据按列方式组织存储，每个列包含 2 个段，一个段存放列数据，一个段存放列的控制信息，读取列数据时，只需要顺序扫描这两个段。在某些特殊应用场景下，其效率要远远高于行存储。

4． 位图索引

位图索引与 B 树索引不同，每个索引条目不是指向一行数据，而是指向多行数据。每个索引项保存的是一定范围内所有行与当前索引键值映射关系的位图。

数据文件中还有两类特殊的数据文件：ROLL 和 TEMP 文件。

1)． ROLL 文件

ROLL 表空间的 dbf 文件，称为 ROLL 文件。ROLL 文件用于保存系统的回滚记录，提供事务回滚时的信息。回滚文件整个是一个段。每个事务的回滚页在回滚段中各自挂链，页内则顺序存放回滚记录。

2)． TEMP 文件

TEMP.DBF 临时数据文件，临时文件可以在 dm.ini 中通过 TEMP_SIZE 配置大小。

当数据库查询的临时结果集过大，缓存已经不够用时，临时结果集就可以保存在 TEMP.DBF 文件中，供后续运算使用。系统中用户创建的临时表也存储在临时文件中。

2.4 重做日志文件

重做日志，又叫 REDO 日志，指在 DM 数据库中添加、删除、修改对象，或者改变数据，DM 都会按照特定的格式，将这些操作执行的结果写入到当前的重做日志文件中。重做日志文件以 log 为扩展名。每个 DM 数据库实例必须至少有 2 个重做日志文件，默认两个日志文件为 DAMENG01.log、DAMENG02.log，这两个文件循环使用。

重做日志文件主要用于数据库的备份与恢复。理想情况下，数据库系统不会用到重做日志文件中的信息。然而现实世界总是充满了各种意外，比如电源故障、系统故障、介质故障，或者数据库实例进程被强制终止等，数据库缓冲区中的数据页会来不及写入数据文件。这样，在重启 DM 实例时，通过重做日志文件中的信息，就可以将数据库的状态恢复到发生意外时的状态。

重做日志文件对于数据库是至关重要的。它们用于存储数据库的事务日志，以便系统在出现系统故障和介质故障时能够进行故障恢复。在 DM 数据库运行过程中，任何修改数据库的操作都会产生重做日志，例如，当一条元组插入到一个表中的时候，插入的结果写入了重做日志，当删除一条元组时，删除该元组的事实也被写了进去，这样，当系统出现故障时，通过分析日志可以知道在故障发生前系统做了哪些动作，并可以重做这些动作使系统恢复到故障之前的状态。

2.5 归档日志文件

日志文件分为联机日志文件和归档日志文件。DM 数据库可以在归档模式和非归档模式下运行。只有当数据库处于归档模式下，才将联机日志文件中的内容保存到硬盘中，形成归档日志文件。

重做日志文件就是联机日志文件。联机日志文件指的是系统当前正在使用的日志文件，创建数据库时，联机日志文件通常被扩展至一定长度，其内容则被初始化为空，当系统运行时，该文件逐渐被产生的日志所填充。对日志文件的写入是顺序连续的。然而系统磁盘空间总是有限，系统必须能够循环利用日志文件的空间，为了做到这一点，当所有日志文件空间被占满时，系统需要清空一部分日志以便重用日志文件的空间，为了保证被清空的日志所“保护”的数据在磁盘上是安全的，这里需要引入一个关键的数据库概念—检查点。当产生检查点时，系统将系统缓冲区中的日志和脏数据页都写入磁盘，以保证当前日志所“保护”的数据页都已安全写入磁盘，这样日志文件即可被安全重用。

归档日志文件，就是在归档模式下，重做联机日志被连续拷贝到归档日志后，所生成了归档日志文件。归档日志文件以归档时间命名，扩展名也是 log。但只有在归档模式下运行时，DM 数据库在重做联机日志文件时才能生成归档日志文件。

采用归档模式会对系统的性能产生影响，然而系统在归档模式下运行会更安全，当出现故障时其丢失数据的可能性更小，这是因为一旦出现介质故障，如磁盘损坏时，利用归档日志，系统可被恢复至故障发生的前一刻，也可以还原到指定的时间点，而如果没有归档日志文件，则只能利用备份进行恢复。

归档日志还是数据守护功能的核心，数据守护中的备库就是通过重做日志来完成与主库的数据同步的。

2.6 逻辑日志文件

如果在 DM 数据库上配置了复制功能，复制源就会产生逻辑日志文件。逻辑日志文件是一个流式的文件，它有自己的格式，且不在第一章所述的页，簇和段的管理之下。

逻辑日志文件内部存储按照复制记录的格式，一条记录紧接着一条记录，存储着复制源端的各种逻辑操作。用于发送给复制目的端。详细内容请看“数据复制”章节。

2.7 备份文件

备份文件以 bak 为扩展名，当系统正常运行时，备份文件不会起任何作用，它也不是数据库必须有的联机文件类型之一。然而，从来没有哪个数据库系统能够保证永远正确无误地运行，当数据库不幸出现故障时，备份文件就显得尤为重要了。

当客户利用管理工具或直接发出备份的 SQL 命令时，DM Server 会自动进行备份，并产生一个或多个备份文件，备份文件自身包含了备份的名称、对应的数据库、备份类型和备份时间等信息。同时，系统还会自动记录备份信息及该备份文件所处的位置，但这种记录是松散的，用户可根据需要将其拷贝至任何地方，并不会影响系统的运行。

2.8 跟踪日志文件

用户在 dm.ini 中配置 SVR_LOG 和 SVR_LOG_SWITCH_COUNT 参数后就会打开跟踪日志。跟踪日志文件是一个纯文本文件，以“dm_commit_日期_时间”命名，默认生成在 DM

安装目录的 log 子目录下面，管理员可通过 ini 参数 SVR_LOG_FILE_PATH 设置其生成路径。

跟踪日志内容包含系统各会话执行的 SQL 语句、参数信息、错误信息等。跟踪日志主要用于分析错误和分析性能问题，基于跟踪日志可以对系统运行状态有一个分析，比如，可以挑出系统现在执行速度较慢的 SQL 语句，进而对其进行优化。

系统中 SQL 日志的缓存是分块循环使用，管理员可根据系统执行的语句情况及压力情况设置恰当的日志缓存块大小及预留的缓冲块个数。当预留块不足以记录系统产生的任务时，系统会分配新的用后即弃的缓存块，但是总的空间大小由 ini 参数 SVR_LOG_BUF_TOTAL_SIZE 控制，管理员可根据实际情况进行设置。

打开跟踪日志会对系统的性能会有较大影响，一般用于查错和调优的时候才会打开，默认情况下系统是关闭跟踪日志的。若需要跟踪日志但对日志的实时性没有严格的要求，又希望系统有较高的效率，可以设置参数 SQL_TRACE_MASK 和 SVR_LOG_MIN_EXEC_TIME 只记录关注的相关记录，减少日志总量；设置参数 SVR_LOG_ASYNC_FLUSH 打开 SQL 日志异步刷盘提高系统性能。

2.9 事件日志文件

DM 数据库系统在运行过程中，会在 log 子目录下产生一个“dm_实例名_日期”命名的事件日志文件。事件日志文件对 DM 数据库运行时的关键事件进行记录，如系统启动、关闭、内存申请失败、IO 错误等一些致命错误。事件日志文件主要用于系统出现严重错误时进行查看并定位问题。事件日志文件随着 DM 数据库服务的运行一直存在。

事件日志文件打印的是中间步骤的信息，所以出现部分缺失属于正常现象。

2.10 数据重演文件

调用系统存储过程 SP_START_CAPTURE 和 SP_STOP_CAPTURE，可以获得数据重演文件。重演文件用于数据重演，存储了从抓取开始到抓取结束时，DM 数据库与客户端的通信消息。使用数据重演文件，可以多次重复抓取这段时间内的数据库操作，为系统调试和性能调优提供了另一种分析手段。

第3章 DM 内存结构

数据库管理系统是一种对内存申请和释放操作频率很高的软件，如果每次对内存的使用都使用操作系统函数来申请和释放，效率会比较低，加入自己的内存管理是 DBMS 系统所必须的。通常内存管理系统会带来以下好处：

1． 申请、释放内存效率更高；

2． 能够有效地了解内存的使用情况；

3． 易于发现内存泄露和内存写越界的问题。

DM 数据库管理系统的内存结构主要包括内存池、缓冲区、排序区、哈希区等。根据系统中子模块的不同功能，对内存进行了上述划分，并采用了不同的管理模式。

3.1 内存池

DM Server 的内存池包括共享内存池和其他一些运行时内存池。

动态视图 V$MEM_POOL 详细记录了当前系统中所有的内存池的状态，可通过查询这个动态视图掌握 DM Server 的内存使用情况。

3.1.1 共享内存池

共享内存池是 DM Server 在启动时从操作系统申请的一大片内存。在 DM Server 的运行期间，经常会申请与释放小片内存，而向操作系统申请和释放内存时需要发出系统调用，此时可能会引起线程切换，降低系统运行效率。采用共享内存池则可一次向操作系统申请一片较大内存，即为内存池，当系统在运行过程中需要申请内存时，可在共享内存池内进行申请，当用完该内存时，再释放掉，即归还给共享内存池。

DM 系统管理员可以通过 DM Server 的配置文件（dm.ini）来对共享内存池的大小进行设置，共享池的参数为 MEMORY_POOL，该配置默认为 200M。如果在运行时所需内存大于配置值，共享内存池也可进行自动扩展，INI 参数 MEMORY_EXTENT_SIZE 指定了共享内存池每次扩展的大小，参数 MEMORY_TARGET 则指定了共享内存池能扩展到的最大大小。

3.1.2 运行时内存池

除了共享内存池，DM Server 的一些功能模块在运行时还会使用自己的运行时内存池。这些运行时内存池是从操作系统申请一片内存作为本功能模块的内存池来使用，如会话内存池、虚拟机内存池等。

3.2 缓冲区

3.2.1 数据缓冲区

数据缓冲区是 DM Server 在将数据页写入磁盘之前以及从磁盘上读取数据页之后，数据页所存储的地方。这是 DM Server 至关重要的内存区域之一，将其设定得太小，会导致

缓冲页命中率低，磁盘 IO 频繁；将其设定得太大，又会导致操作系统内存本身不够用。

系统启动时，首先根据配置的数据缓冲区大小向操作系统申请一片连续内存并将其按数据页大小进行格式化，并置入“自由”链中。数据缓冲区存在三条链来管理被缓冲的数据页，一条是“自由”链，用于存放目前尚未使用的内存数据页，一条是“LRU”链，用于存放已被使用的内存数据页(包括未修改和已修改)，还有一条即为“脏”链，用于存放已被修改过的内存数据页。

LRU 链对系统当前使用的页按其最近是否被使用的顺序进行了排序。这样当数据缓冲区中的自由链被用完时，从 LRU 链中淘汰部分最近未使用的数据页，能够较大程度地保证被淘汰的数据页在最近不会被用到，减少 IO。

在系统运行过程中，通常存在一部分“非常热”（反复被访问）的数据页，将它们一直留在缓冲区中，对系统性能会有好处。对于这部分数据页，数据缓冲区开辟了一个特定的区域用于存放它们，以保证这些页不参与一般的淘汰机制，可以一直留在数据缓冲区中。

1） 类别

DM Server 中有四种类型的数据缓冲区，分别是 NORMAL、KEEP、FAST 和 RECYCLE。其中，用户可以在创建表空间或修改表空间时，指定表空间属于 NORMAL 或 KEEP 缓冲区。RECYCLE 缓冲区供临时表空间使用，FAST 缓冲区根据用户指定的 FAST_POOL_PAGES 大小由系统自动进行管理，用户不能指定使用 RECYCLE 和 FAST 缓冲区的表或表空间。

NORMAL 缓冲区主要是提供给系统处理的一些数据页，没有特定指定缓冲区的情况下，默认缓冲区为 NORMAL；KEEP 的特性是对缓冲区中的数据页很少或几乎不怎么淘汰出去，主要针对用户的应用是否需要经常处在内存当中，如果是这种情况，可以指定缓冲区为 KEEP。

DM Server 提供了可以更改这些缓冲区大小的参数，用户可以根据自己应用需求情况，指定 dm.ini 文件中 BUFFER(100MB)、KEEP(8MB)、RECYCLE(64MB)、FAST_POOL_PAGES(3000)值（括号中为默认值），这些值分别对应是 NORMAL 缓冲区大小、KEEP 缓冲区大小、RECYCLE 缓冲区大小、FAST 缓冲区数据页总数。

2） 读多页

在需要进行大量 I/O 的应用当中，DM 之前版本的策略是每次只读取一页。如果知道用户需要读取表的大量数据，当读取到第一页时，可以猜测用户可能需要读取这页的下一页，在这种情况下，一次性读取多页就可以减少 I/O 次数，从而提高了数据的查询、修改效率。

DM Server 提供了可以读取多页的参数，用户可以指定这些参数来调整数据库运行效率的最佳状态。在 DM 配置文件 dm.ini 中，可以指定参数 MULTI_PAGE_GET_NUM 大小（默认值为 16 页），来控制每次读取的页数。

如果用户没有设置较适合的参数 MULTI_PAGE_GET_NUM 值大小，有时可能会给用户带来更差的效果。如果 MULTI_PAGE_GET_NUM 太大，每次读取的页可能大多都不是以后所用到的数据页，这样不仅会增加 I/O 的读取，而且每次都会做一些无用的 I/O，所以系统管理员需要衡量好自己应用需求，给出最佳方案。

3.2.2 日志缓冲区

日志缓冲区是用于存放重做日志的内存缓冲区。为了避免由于直接的磁盘 IO 而使系统性能受到影响，系统在运行过程中产生的日志并不会立即被写入磁盘，而是和数据页一样，先将其放置到日志缓冲区中。那么为何不在数据缓冲区中缓存重做日志而要单独设立日志缓冲区呢？主要是基于以下原因：

1． 重做日志的格式同数据页完全不一样，无法进行统一管理；

2． 重做日志具备连续写的特点；

3． 在逻辑上，写重做日志比数据页 IO 优先级更高。

DM Server 提供了参数 RLOG_BUF_SIZE 对日志缓冲区大小进行控制，日志缓冲区所

占用的内存是从共享内存池中申请的，单位为页数量，且大小必须为 2 的 N 次方，否则采用系统默认大小 512 页。

3.2.3 字典缓冲区

字典缓冲区主要存储一些数据字典信息，如模式信息、表信息、列信息、触发器信息等。每次对数据库的操作都会涉及到数据字典信息，访问数据字典信息的效率直接影响到相应的操作效率，如进行查询语句，就需要相应的表信息、列信息等，这些字典信息如果都在缓冲区里，则直接从缓冲区中获取即可，否则需要 I/O 才能读取到这些信息。

DM7 采用的是将部分数据字典信息加载到缓冲区中，并采用 LRU 算法进行字典信息的控制。缓冲区大小设置问题，如果太大，会浪费宝贵的内存空间，如果太小，可能会频繁的进行淘汰，该缓冲区配置参数为 DICT_BUF_SIZE，默认的配置大小为 5M。

DM7 采用缓冲部分字典对象，那会影响效率吗？数据字典信息访问存在热点现像，并不是所有的字典信息都会被频繁的访问,所以按需加载字典信息并不会影响到实际的运行效率。

但是如果在实际应用中涉及对分区数较多的水平分区表访问，例如上千个分区，那么就需要适当调大 DICT_BUF_SIZE 参数值。

3.2.4 SQL 缓冲区

SQL 缓冲区提供在执行 SQL 语句过程中所需要的内存，包括计划、SQL 语句和结果集缓存。

很多应用当中都存在反复执行相同 SQL 语句的情况，此时可以使用缓冲区保存这些语句和它们的执行计划，这就是计划重用。这样带来的好处是加快了 SQL 语句执行效率，但同时给内存也增加了压力。

DM Server 在配置文件 dm.ini 提供了参数来支持是否需要计划重用，参数为 USE_PLN_POOL，当指定为非 0 时，则启动计划重用；为 0 时禁止计划重用。DM 同时还提供了参数 CACHE_POOL_SIZE（单位为 MB），来改变 SQL 缓冲区大小，系统管理员可以设置该值以满足应用需求，默认值为 10M。

结果集缓存包括 SQL 查询结果集缓存和 DMSQL 程序函数结果集缓存，在 INI 参数文件中同时设置参数 RS_CAN_CACHE=1 且 USE_PLN_POOL 非 0 时 DM 服务器才会缓存结果集。DM 还提供了一些手动设置结果集缓存的方法，具体可参看 24.10 节。

客户端结果集也可以缓存，但需要在配置文件 dm_svc.conf 中设置参数：

```
ENABLE_RS_CACHE = (1)   //表示启用缓存;
RS_CACHE_SIZE = (100)   //表示缓存区的大小为 100M, 可配置为 1-65535
RS_REFRESH_FREQ = (30)  //表示每 30 秒检查缓存的有效性，如果失效，自动重查； 0 表示不检查。
```

同时在服务器端使用 INI 参数文件中的 CLT_CACHE_TABLES 参数设置哪些表的结果集需要缓存。另外，FIRST_ROWS 参数表示当查询的结果达到该行数时，就返回结果，不再继续查询，除非用户向服务器发一个 FETCH 命令。这个参数也用于客户端缓存的配置，仅当结果集的行数不超过 FIRST_ROWS 时，该结果集才可能被客户端缓存。

3.3 排序区

排序缓冲区提供数据排序所需要的内存空间。当用户执行 SQL 语句时，常常需要进行排序，所使用的内存就是排序缓冲区提供的。在每次排序过程中，都首先申请内存，排序结束

后再释放内存。

DM Server 提供了参数来指定排序缓冲区的大小，参数 SORT_BUF_SIZE 在 DM 配置文件 dm.ini 中，系统管理员可以设置其大小以满足需求，由于该值是由系统内部排序算法和排序数据结构决定，建议使用默认值 2M。

3.4 哈希区

DM7 提供了为哈希连接而设定的缓冲区，不过该缓冲区是个虚拟缓冲区。之所以说是虚拟缓冲，是因为系统没有真正创建特定属于哈希缓冲区的内存，而是在进行哈希连接时，对排序的数据量进行了计算。如果计算出的数据量大小超过了哈希缓冲区的大小，则使用 DM7 创新的外存哈希方式；如果没有超过哈希缓冲区的大小，实际上使用的还是 VPOOL 内存池来进行哈希操作。

DM Server 在 dm.ini 中提供了参数 HJ_BUF_SIZE 来进行控制，由于该值的大小可能会限制哈希连接的效率，所以建议保持默认值，或设置为更大的值。

除了提供 HJ_BUF_SIZE 参数外，DM Server 还提供了创建哈希表个数的初始化参数，其中，HAGR_HASH_SIZE 表示处理聚集函数时创建哈希表的个数，建议保持默认值 100000。

3.5 SSD 缓冲区

固态硬盘采用闪存作为存储介质，因没有机械磁头的寻道时间，在读写效率上比机械磁盘具有优势。在内存、SSD 磁盘、机械磁盘之间，符合存储分级的条件。为提高系统执行效率，DM Server 将 SSD 文件作为内存缓存与普通磁盘之间的缓冲层，称为“SSD 缓存”。DM Server 在的 dm.ini 中提供参数 SSD_BUF_SIZE 和 SSD_FILE_PATH 来配置 SSD 缓冲，SSD_BUF_SIZE 指定缓冲区的大小，单位是 M，DM Server 根据该参数创建相应大小的文件作为缓冲区使用；SSD_FILE_PATH 指定该文件所在的文件夹路径，管理员需要保证设置的路径是位于固态磁盘上。

默认 SSD 缓冲区是关闭的，即 SSD_BUF_SIZE 为 0。若要配置 SSD 缓冲区，将其设置为大于 0 的数并指定 SSD_FILE_PATH 即可。

第4章 管理 DM 线程

DM 服务器使用“对称服务器构架”的单进程、多线程结构。这种对称服务器构架在有效地利用了系统资源的同时又提供了较高的可伸缩性能,这里所指的线程即为操作系统的线程。服务器在运行时由各种内存数据结构和一系列的线程组成，线程分为多种类型，不同类型的线程完成不同的任务。线程通过一定的同步机制对数据结构进行并发访问和处理，以完成客户提交的各种任务。DM 数据库服务器是共享的服务器,允许多个用户连接到同一个服务器上,服务器进程称为共享服务器进程。

DM 进程中主要包括监听线程、IO 线程、工作线程、调度线程、日志线程等，以下分别对它们进行介绍。

4.1 监听线程

监听线程主要的任务是在服务器端口上进行循环监听，一旦有来自客户的连接请求，监听线程被唤醒并生成一个会话申请任务，加入工作线程的任务队列，等待工作线程进行处理。它在系统启动完成后才启动，并且在系统关闭时首先被关闭。为了保证在处理大量客户连接时系统具有较短的响应时间，监听线程比普通线程优先级更高。

4.2 工作线程

工作线程是 DM 服务器的核心线程，它从任务队列中取出任务，并根据任务的类型进行相应的处理，负责所有实际的数据相关操作。

DM7 的初始工作线程个数由配置文件指定，随着会话连接的增加，工作线程也会同步增加，以保持每个会话都有专门的工作线程处理请求。为了保证用户所有请求及时响应，一个会话上的任务全部由同一个工作线程完成，这样减少了线程切换的代价，提高了系统效率。当会话连接超过预设的阀值时，工作线程数目不再增加，转而由会话轮询线程接收所有用户请求，加入任务队列，等待工作线程一旦空闲，从任务队列依次摘取请求任务处理。

4.3 IO 线程

在数据库活动中，IO 操作历来都是最为耗时的操作之一。当事务需要的数据页不在缓冲区中时，如果在工作线程中直接对那些数据页进行读写，将会使系统性能变得非常糟糕，而把 IO 操作从工作线程中分离出来则是明智的做法。IO 线程的职责就是处理这些 IO 操作。

通常情况下，DM Server 需要进行 IO 操作的时机主要有以下三种：

1． 需要处理的数据页不在缓冲区中，此时需要将相关数据页读入缓冲区；

2． 缓冲区满或系统关闭时，此时需要将部分脏数据页写入磁盘；

3． 检查点到来时，需要将所有脏数据页写入磁盘。

IO 线程在启动后，通常都处于睡眠状态，当系统需要进行 IO 时，只需要发出一个 IO 请求，此时 IO 线程被唤醒以处理该请求，在完成该 IO 操作后继续进入睡眠状态。

IO 线程的个数是可配置的，可以通过设置 dm.ini 文件中的 IO_THR_GROUPS 参数来设置，默认情况下，IO 线程的个数是 2 个。同时，IO 线程处理 IO 的策略根据操作系统平台的不同会有很大差别，一般情况下，IO 线程使用异步的 IO 将数据页写入磁盘，此时，系统将所有的 IO 请求直接递交给操作系统，操作系统在完成这些请求后才通知 IO 线程，这种

异步 IO 的方式使得 IO 线程需要直接处理的任务很简单，即完成 IO 后的一些收尾处理并发出 IO 完成通知，如果操作系统不支持异步 IO，此时 IO 线程就需要完成实际的 IO 操作。

4.4 调度线程

调度线程用于接管系统中所有需要定时调度的任务。调度线程每秒钟轮询一次，负责的任务有以下一些：

1. 检查系统级的时间触发器，如果满足触发条件则生成任务加到工作线程的任务队列由工作线程执行；
2. 清理 SQL 缓存、计划缓存中失效的项，或者超出缓存限制后淘汰不常用的缓存项；
3. 检查数据重演捕获持续时间是否到期，到期则自动停止捕获；
4. 执行动态缓冲区检查。根据需要动态扩展或动态收缩系统缓冲池；
5. 自动执行检查点。为了保证日志的及时刷盘，减少系统故障时恢复时间，根据 INI 参数设置的自动检查点执行间隔定期执行检查点操作；
6. 会话超时检测。当客户连接设置了连接超时时，定期检测是否超时，如果超时则自动断开连接；
7. 必要时执行数据更新页刷盘；
8. 唤醒等待的工作线程。

4.5 日志 FLUSH 线程

任何数据库的修改，都会产生重做 REDO 日志，为了保证数据故障恢复的一致性，REDO 日志的刷盘必须在数据页刷盘之前进行。事务运行时，会把生成的 REDO 日志保留在日志缓冲区中，当事务提交或者执行检查点时，会通知 FLUSH 线程进行日志刷盘。由于日志具备顺序写入的特点，比数据页分散 IO 写入效率更高。日志 FLUSH 线程和 IO 线程分开，能获得更快的响应速度，保证整体的性能。DM7 的日志 FLUSH 线程进行了优化，在刷盘之前，对不同缓冲区内的日志进行合并，减少了 IO 次数，进一步提高了性能。

如果系统配置了实时归档，在 FLUSH 线程日志刷盘前，会直接将日志通过网络发送到实时备库。如果配置了本地归档，则生成归档任务，通过日志归档线程完成。

4.6 日志归档线程

日志归档线程包含异步归档线程，负责远程异步归档任务。如果配置了非实时归档，由日志 FLUSH 线程产生的任务会分别加入日志归档线程，日志归档线程负责从任务队列中取出任务，按照归档类型做相应归档处理。

将日志 FLUSH 线程和日志归档线程分开的目的是为了减少不必要的效率损失，除了远程实时归档外，本地归档、远程异步归档都可以脱离 FLUSH 线程来做，如果放在 FLUSH 线程中一起做会严重影响系统性能。

4.7 日志 APPLY 线程

在配置了数据守护的系统中，创建了一个日志 APPLY 线程。当服务器作为备库时，每次接收到主库的物理 REDO 日志生成一个 APPLY 任务加入到任务队列，APPLY 线程从任务队列中取出一个任务在备库上将日志重做，并生成自己的日志，保持和主库数据的同步或一

致，作为主库的一个镜像。备库数据对用户只读，可承担报表、查询等任务，均衡主库的负载。

4.8 定时器线程

在数据库的各种活动中，用户常常需要数据库完成在某个时间点开始进行某种操作，如备份；或者是在某个时间段内反复进行某种操作等。定时器线程就是为这种需求而设计的。

通常情况下，DM Server 需要进行定时操作的事件主要有以下几种：

1． 逻辑日志异步归档；

2． 异步归档日志发送（只有在 PRIMARY 模式下，且是 OPEN 状态下）；

3． 作业调度。

定时器线程启动之后，每秒检测一次定时器链表，查看当前的定时器是否满足触发条件，如果满足，则把执行权交给设置好的任务，如逻辑日志异步归档等。

默认情况下，达梦服务器启动的时候，定时器线程是不启动的。用户可以设置 dm.ini 中的 TIMER_INI 参数为 1 来设置定时器线程在系统启动时启动。

4.9 逻辑日志归档线程

逻辑日志归档用于 DM7 的数据复制中，目的是为了加快异地访问的响应速度，包含本地逻辑日志归档线程和远程逻辑日志归档线程。当配置了数据复制，系统才会创建这两个线程。

1． 本地逻辑日志归档线程

本地归档线程从本地归档任务列表中取出一个归档任务，生成到逻辑日志，并将逻辑日志写入到逻辑日志文件中。如果当前逻辑日志的远程归档类型是同步异地归档并且当前的刷盘机制是强制刷盘，那么就生成一个异地归档任务加入到临时列表中。

2． 远程逻辑日志归档线程

远程归档线程从远程归档任务列表中取出一个归档任务，并根据任务的类型进行相应的处理。任务的类型包括同步发送和异步发送。

4.10 MAL 系统相关线程

MAL 系统是 DM 内部高速通信系统，基于 TCP/IP 协议实现。服务器的很多重要功能都是通过 MAL 系统实现通信的，例如数据守护、数据复制、MPP、远程日志归档等。MAL 系统内部包含一系列线程，有 MAL 监听线程、MAL 发送工作线程、MAL 接收工作线程等。

4.11 其他线程

事实上，DM 数据库系统中还不止以上这些线程，在一些特定的功能中会有不同的线程，例如回滚段清理 PURGE 线程、审计写文件线程、重演捕获写文件线程等，这里不一一列出。

4.12 线程信息的查看

为了增加用户对 DM 数据库内部信息的了解，以及方便数据库管理员对数据库的维护，DM 提供了很多动态性能视图，通过它们用户可以直观地了解当前系统中有哪些线程在工作，

以及线程的相关信息。相关动态视图见表 4.1。

表 4.1 DM 线程相关的动态视图

名称	说明
V$LATCHES	记录当前正在等待的线程信息
V$THREADS	记录当前系统中活动线程的信息
V$WTHRD_HISTORY	记录自系统启动以来，所有活动过线程的相关历史信息。
V$PROCESS	记录服务器进程信息

以上视图的详细定义参考本手册中附录 2“动态性能视图”。

第5章 DM7 的升级

相对于 DM6，DM7 不仅对原有功能做了大幅度的改进，也提供了不少新特色。DM7 的系统表同时进行了重新设计和组织，因此原有的 DM6 数据文件 DM7 不能直接识别和加载。旧版本的数据，必须升级为 DM7 的数据。这一章将会介绍如何将 DM6 的数据升级为 DM7 的数据。

5.1 选择升级方法

DM7 提供 2 种方法进行数据升级：1.利用数据迁移工具（DTS）；2.利用数据导入导出工具（dm6 的 dmloader，dm7 的 dmfldr）。数据迁移工具 DTS，由于支持众多模式对象的迁移（例如表、视图、索引、序列等），所以适用于将 DM6 的数据库升级至 DM7。数据导入导出工具由于处理单个表时效率较高，所以适用于单独将某些大表升级至 DM7。

因此对 DM 数据库进行升级前，需要根据待升级数据选择合适的升级方法。如果进行全库升级，那就选择数据迁移工具；如果涉及数据量很大的单个表，那就选择数据导入导出工具；如果既要导入全库，其中又包含数据量非常大的表，那么可以 2 种方法结合，在利用 DTS 迁移全库时，对于大表选择只迁移表结构，不迁移数据，然后再利用 DM 数据导入导出工具完成大数据表的升级。

5.2 升级前准备工作

数据升级前需要从整体考虑升级工作，制定升级规划和操作步骤。通常来说，升级工作按照如下流程进行：

1. 在目标数据库上重建原库的用户；
2. 迁移元数据结构，包括表、视图、序列；
3. 迁移数据；
4. 重建索引；
5. 迁移其它数据库对象，如存储过程、存储函数、触发器、同义词等等。注意为了保证迁移数据的正确性，触发器迁移成功后，需要禁用触发器；
6. 重建权限体系，包括建立角色，授予权限；
7. 启用所有触发器。

至此，数据升级工作完成。

依据数据升级工作流程，在正式进行数据升级前，我们需要准备创建用户、角色以及授予权限的 SQL 脚本，创建存储过程、存储函数、触发器、同义词等数据对象的 SQL 脚本。由于 dm7 的存储模块同 dm6 有些差别，所以存储过程、存储函数、触发器的 SQL 脚本可能需要进行部分修改。

对于元数据结构和数据的升级，需要根据对象选择合适的方法。因此，我们推荐升级工作进行前确定数据量大的表使用数据导入导出工具完成迁移。

小数据量表上的索引可以利用数据迁移工具 DTS 进行重建。对于大数据量表上的索引，建议在数据导入完成后通过 sql 脚本重建索引。为此，我们还需要准备索引重建的 SQL 脚本。

注意：数据迁移工具 DTS 可以得到上述除用户、角色、授权外的 SQL 脚本。方法是选择 DM 迁移到 SQL 脚本文件，配置迁移策略为不迁移数据。

5.3 使用数据迁移工具

由于数据迁移工具 DTS 采用向导式界面引导用户进行数据迁移，所以用 DTS 进行数据升级比较简单，这里简单描述下整体过程及注意事项。

在目标数据库上重建原库用户后，我们即可利用 DTS 迁移元数据及数据了。注意工作流程应当遵循我们设定的升级规划逐步完成。对于计划利用数据导入导出工具完成数据迁移的大表，配置迁移策略为不迁移数据。如果需要数据迁移完成后重建索引，那么配置迁移策略为不迁移索引。

如果发生迁移错误，我们需要详细查看错误日志信息，找到出错 sql 脚本。总体来说，最常见的错误原因为依赖对象不存在，或者 sql 语法错误。对于依赖对象不存在的错误，可以在依赖对象定义后再次进行迁移。对于 sql 语法错误，可以修改 sql 脚本后在 disql 中单独执行。

5.4 使用数据导入导出工具

使用数据导入导出工具是为了提高大数据量表升级效率而采用的方法，步骤如下：

1. 在 DM7 上创建目标表结构。可利用 DTS 完成，或者通过 sql 脚本完成；
2. 利用 dm6 的 dmloader，将大表数据导出形成文本文件；
3. 利用 dm7 的 dmfldr，导入文本文件；
4. 在 dm7 上重建该表索引。

5.5 升级后期工作

元数据和数据迁移完成后，首先检查数据的正确性。数据正确后，还有以下 4 个工作有待完成：

1. 重建索引；
2. 迁移存储过程、存储函数、触发器、同义词等数据库对象；
3. 重建权限体系，包括建立角色，授予权限；
4. 启用触发器。

这些工作均通过执行 sql 脚本完成。

第二部分 基础数据库管理

第6章 DM 系统管理员

为了保证数据库系统的安全性，DM 数据库采用“三权分立”或“四权分立”的安全机制，“三权分立”时系统内置三种系统管理员，包括数据库管理员、数据库安全员和数据库审计员，四权分立”时新增了一类用户，称为数据库对象操作员。它们各司其职，互相制约，有效地避免了将所有权限集中于一人的风险，保证了系统的安全性。

6.1 DM 系统管理员的类型

在现实生活中，任何一个系统如果将所有的权利都赋予给某一个人，而不加以监督和控制，势必会产生权利滥用的风险。从数据库安全角度出发，一个大型的数据库系统有必要将数据库系统的权限分配给不同的角色来管理，并且各自偏重于不同的工作职责，使之能够互相限制和监督，从而有效保证系统的整体安全。

DM 数据库实现了 B1 级安全特性。“三权分立”的安全机制，将系统管理员分数据库管理员、数据库安全员和数据库审计员三类。在安装过程中，DM 数据库会预设数据库管理员账号 SYSDBA、数据库安全员账号 SYSSSO 和数据库审计员账号 SYSAUDITOR，其缺省口令都与用户名一致。“四权分立”的安全机制，将系统管理员分数据库管理员、数据库对象操作员、数据库安全员和数据库审计员四类，在“三权分立”的基础上，新增数据库对象操作员账户 SYSDBO,其缺省口令为 SYSDBO。用户需要在安装过程中或者安装完毕后立即修改缺省口令，避免因口令泄漏造成的安全问题。

1. 数据库管理员（DBA）

“三权分立”的安全机制，每个 DM 数据库至少需要一个数据库管理员来管理，负责评估数据库运行所需的软、硬件环境、安装和升级 DM 数据库、配置 DM 数据库参数、创建主要的数据库存储结构（表空间）和对象（如表、视图、索引、角色、用户等）、监控和优化数据库性能、数据导入导出以及数据库的备份和恢复等。

“三权分立”时数据库管理员既可进行系统管理和维护工作，也可对数据内容进行增删查改动作。根据国产数据库军事使用要求，数据库管理员只能进行系统管理和维护工作，不能对数据内容进行增删查改，数据库应用人员则可操作数据内容，而不能管理和维护系统。

“四权分立”的安全机制，在原有“三权分立”基础上调整自主访问控制权限，只具有“三权分立”中 DBA 角色预设的一部分与数据库管理相关的明确的数据库权限，如数据库创建、备份、还原和校验等，具体见附录 7。

2. 数据库安全员（SSO）

对于很多对安全性要求不高的系统来说，C2 级安全特性已经能够工作得很好，此时不需要考虑通过数据库安全员来进一步加强系统的安全机制。但是在很多大型的系统中，安全性还是至关重要的，有必要由安全员来制定安全策略，强化系统安全机制，此时数据库安全员的主要任务就是制定安全策略，定义新的数据库安全员，设置系统的安全等级、范围和组，并为主、客体定义安全标记，从而全面提升系统安全性。

3. 数据库审计员（AUDITOR）

数据库审计员可以设置要审计的对象和操作、定义新的数据库审计员、查看和分析审计记录。通过设置审计，几乎可以跟踪任何人在系统内执行的任何操作，为事后追查提供便利。

4． 数据库对象操作员（DBO）

数据库对象操作员是“四权分立”新增加的一类用户，可以创建数据库对象，并对自己拥有的数据库对象（表、视图、存储过程、序列、包、外部链接）具有所有的对象权限并可以授出与回收，但其无法管理与维护数据库对象。

6.2 数据库管理员的任务

每个数据库至少需要一个 DBA 来管理，DBA 可能是一个团队，也可能是一个人。在不同的公司，数据库管理员的职责可能也会有比较大的区别，总体而言，数据库管理员的职责主要包括以下任务。

1． 评估数据库服务器所需的软、硬件运行环境

通常情况下，DBA 不会直接配置除数据库之外的软、硬件环境，不过在某些情况下还是需要。即便不用负责配置其他软、硬件，还是应该负责为软、硬件的选型提供指南和规范，以便最终配置能够提供满足业务需要的最优级别的性能、可靠性和可扩展性，同时保证总体费用在预算范围内。

作为 DBA，需要结合实际应用负载、总体费用、性能预期目标等来评估可供选择的软、硬件运行环境。其中实际应用负载和总体费用基本上都是确定的，DBA 的目标就是保证在负载和费用基本不变的情况下，实现系统总体性能的最优化。DBA 主要考虑的因素包括：

1） 操作系统/中间件等通用软件的稳定性、性能、安全性；
2） 处理器的个数和性能；
3） 内存容量和性能；
4） 网络带宽；
5） 存储容量和读写性能；
6） HBA 卡传输性能。

2． 安装和升级 DM 服务器

作为数据库管理员，应负责安装 DM 数据库服务器软件。具体的安装配置步骤请参考《DM7 安装手册》。需要注意的是，和所有软件一样，应先在测试系统上完成安装，这样可以确保不会影响到生产系统。

随着 DM 数据库的发展，会在 DM 数据库发布后提供服务包和安全更新。正确地安装服务包和安全更新是至关重要的任务，它影响到系统的稳定性、性能和安全。DBA 需要清楚地了解服务包和安全更新对现有系统的影响，然后考虑是否进行升级。是在生产系统上应用新的升级包之前，必须在测试环境下进行确认。

3． 数据库结构设计

在安装 DM 数据库之后，DBA 就可以开始进行数据库结构设计。数据库的结构设计直接影响系统的综合性能，是 DBA 最为重要的工作任务之一。

DBA 可以依次参考第 7 章来创建和配置 DM 数据库，参考第 8 章来启动和关闭数据库，参考第 16 章来实施计划好的数据库存储结构部署工作，参考第 9-15 章、第 18-21 章为每个对象规划数据库对象和存储特性的相关性的设计。通过在创建对象之前规划每个对象和它的物理存储之间的关系，可以直接影响数据库的性能。

4． 监控和优化数据库的性能

监控和优化数据库的性能是数据库管理员的核心职责之一。对系统进行监控有助于在问题发生时识别它们，性能优化则有助于消除那些问题并防止它们发生。监视长时间运行的查

询有助于发现可以改进的领域，如增加索引、重写查询、改进应用程序逻辑等，还可以发现由于缺少内存/内存分配不合理而导致的性能降低，以便调整 DM 数据库的内存使用参数，或为服务器增加更多的内存。DBA 需要提供对于各种问题的解决方案，努力优化 DM 数据库的各个方面，必要时，可以直接拨打 DM 数据库产品售后服务热线来寻求帮助。

5. 计划和实施备份与故障恢复

即便是最好、最稳定的系统也会存在意外，如断电、硬件故障等。DBA 必须对各种可能的故障和计划中的停机提供解决方案。一般来说，系统的可靠性与成本之间存在着类似线性的关系，越是可靠的解决方案，其整体成本必然也会更高，DBA 必须在企业限定的成本内制定可行的可靠性解决方案。

DM 数据库提供了最基本的故障恢复、备份与还原功能，另外还提供了数据守护、数据复制、共享存储集群等高可靠、高可用解决方案。DBA 可参考本书第五部分、《DM7 备份与还原》、《DM7 数据守护与读写分离集群 V2.1》和《DM7 共享存储集群》来选择合适的解决方案。

6.3 数据库安全员的任务

有些应用对于安全性有着很高的要求，传统的由 DBA 一人拥有所有权限并且承担所有职责的安全机制可能无法满足企业实际需要，此时数据库安全员和数据库审计员两类角色和用户就显得异常重要，它们对于限制和监控数据库管理员的所有行为都起着至关重要的作用。

数据库安全员的主要职责就是制定并应用安全策略，强化系统安全机制。数据库安全员 SYSSSO 是 DM 数据库初始化的时候就已经创建好的，可以以该用户登录到 DM 数据库来创建新的数据库安全员。SYSSSO 或者新的数据库安全员都可以制定自己的安全策略，在安全策略中定义安全级别、范围和组，然后基于定义的安全级别、范围和组来创建安全标记，并将安全标记分别应用到主体（用户）和客体（表），以便启用强制访问控制功能。数据库安全员不能对用户数据进行增、删、改、查，也不能执行普通的 DDL 操作，他们只负责制定安全机制，将合适的安全标记应用到主体和客体，通过这种方式可以有效的对 DBA 的权限进行限制，DBA 此后就不能直接访问添加有安全标记的数据，除非安全员给 DBA 也设定了与之匹配的安全标记，DBA 的权限受到了有效的约束，详细的安全特性介绍请参考《DM7 安全管理》。

6.4 数据库审计员的任务

我们可以想象一下，某个企业内部 DBA 非常熟悉公司内部 ERP 系统的数据库设计，该系统包括了员工工资表，里面记录了所有员工的工资，公司的出纳通过查询系统内部员工工资表来发放工资。传统的 DBA 集所有权利于一身，可以很容易修改工资表，从而导致公司工资账务错乱。为了预防该问题，可以采用前面数据库安全员制定安全策略的方法，避免 DBA 或者其他数据库用户具有访问该表的权限。为了能够及时找到 DBA 或者其他用户的非法操作，在 DM 数据库中还可以在系统建设初期，由数据库审计员（SYSAUDITOR 或者其他由 SYSAUDITOR 创建的审计员）来设置审计策略（包括审计对象和操作），在需要时，数据库审计员可以查看审计记录，及时分析并查找出幕后真凶。

从上面的介绍我们也可以看到，在 DM 数据库中，审计员的主要职责就是创建和删除数据库审计员，设置/取消对数据库对象和操作的审计设置，查看和分析审计记录等。关于审计的详细介绍，请参考《DM7 安全管理》。

第7章 创建和配置 DM 数据库

DM 数据库可以在安装 DM 软件时创建，也可以在安装 DM 软件之后，通过数据库配置工具或 dminit 来手工创建数据库，创建数据库时要使用初始化参数。

7.1 创建 DM 数据库

用户创建数据库之前，需要规划数据库，如数据库名、实例名、端口、文件路径、簇大小、页大小、日志文件大小、SYSDBA 和 SYSAUDITOR 等系统用户的密码等，然后可以使用图形化界面或者 dminit 创建数据库。用户可以在安装 DM 数据库软件时创建数据库，也可以在安装之后创建数据库。

在创建数据库之前需要做如下准备工作：

1. 规划数据库表和索引，并估算它们所需的空间大小；
2. 确定字符集。所有字符集数据，包括数据字典中的数据，都被存储在数据库字符集中，用户在创建数据库时可以指定数据库字符集，如不指定则使用默认字符集 GB18030；
3. 规划数据库文件的存储路径，可以指定数据库存储路径、控制文件存放路径、日志文件存放路径等，应注意在指定的路径或文件名中尽量不要包含中文字符，否则可能由于数据库与操作系统编码方式不一致导致不可预期的问题；
4. 配置数据库时区，如中国是+8:00 时区；
5. 设置数据库簇大小、页大小、日志文件大小，在数据库创建时由 EXTENT_SIZE、PAGE_SIZE、LOG_SIZE 初始化参数来指定，并且在数据库创建完成之后不能修改此参数。

创建数据库之前，必须满足以下必要条件：

1. 安装必需的 DM 软件，包括为操作系统设置各种环境变量，并为软件和数据库文件建立目录结构；
2. 必须有足够的内存来启动 DM 数据库实例；
3. 在执行 DM 的计算机上要有足够的磁盘存储空间来容纳规划的数据库。

7.2 使用数据库配置工具创建数据库

数据库配置工具提供了一个图形化界面来引导用户创建和配置数据库。使用数据库配置工具来创建数据库是用户的首选，因为数据库配置工具更趋于自动化，当使用数据库配置工具安装完成时，数据库也做好了使用准备工作。数据库配置工具可以通过 DM 数据库安装之后作为一个独立的工具来启动。数据库配置工具可以执行以下任务：

1. 创建数据库；
2. 改变数据库的配置；
3. 删除数据库；
4. 管理模板；
5. 自动存储管理；
6. 注册数据库服务。

7.2.1 启动数据库配置工具

数据库配置工具可以通过 DM 支持的模板或用户自定义的模板来创建数据库。
通过下列步骤来启动数据库配置工具（以 Windows 操作系统为例）：

1. 使用通过验证的具有管理权限组的成员登录到计算机上，安装 DM 数据库软件，并创建和运行数据库；
2. 在 Windows 操作系统中启动数据库配置工具。选择“开始”→“程序”→“达梦数据库”→“客户端”→“数据库配置工具”，启动数据库配置工具，将出现数据库配置工具界面，按照数据库配置工具操作，如图 7.1 所示。

图 7.1 启动数据库配置工具

7.2.2 使用数据库配置工具创建数据库

在数据库配置工具操作窗口中，选择“创建数据库”选项启动能够创建和配置一个数据库的向导，如图 7.2 所示。这个向导引导用户完成以下操作：

1. 数据库模板；
2. 数据库目录；
3. 数据库标识；
4. 数据库文件；
5. 初始化参数；
6. 口令管理；
7. 创建示例库；
8. 创建摘要；
9. 创建数据库。

图 7.2 数据库配置工具操作界面

1） 数据库模板

在这个窗口中可以选择需要创建数据库的类型，DM 预定义了一些模板，如一般用途、联机分析处理模板或联机事务处理模板。如图 7.3 所示。

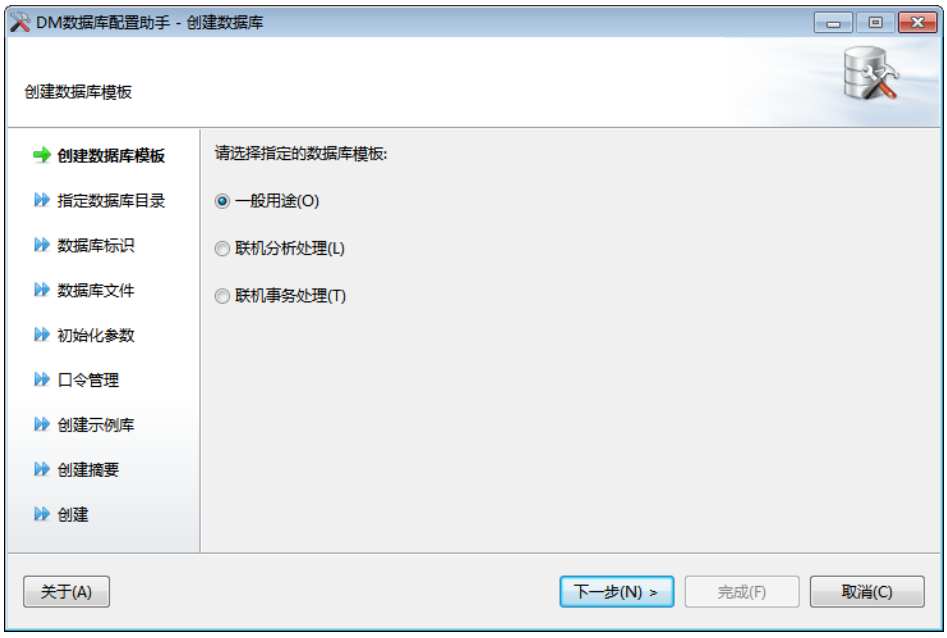


图 7.3 数据库模板选择

2） 数据库目录

指定数据库目录。如图 7.4 所示。

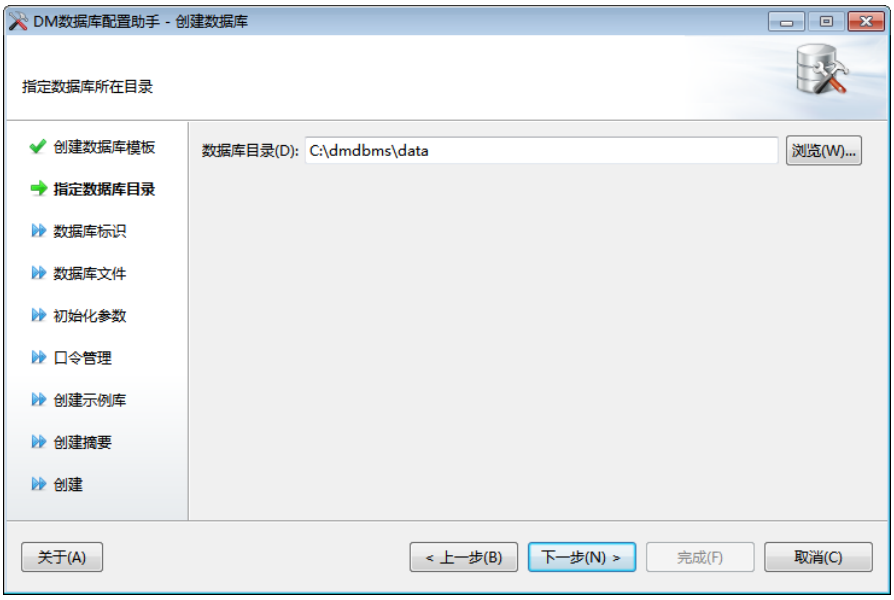


图 7.4 指定数据库目录

3） 数据库标识

在“数据库名（D）”文本框中，输入数据库名；在“实例名（I）”文本框中输入数据库实例名；在“端口（P）”文本框中，输入端口号。如图 7.5 所示。

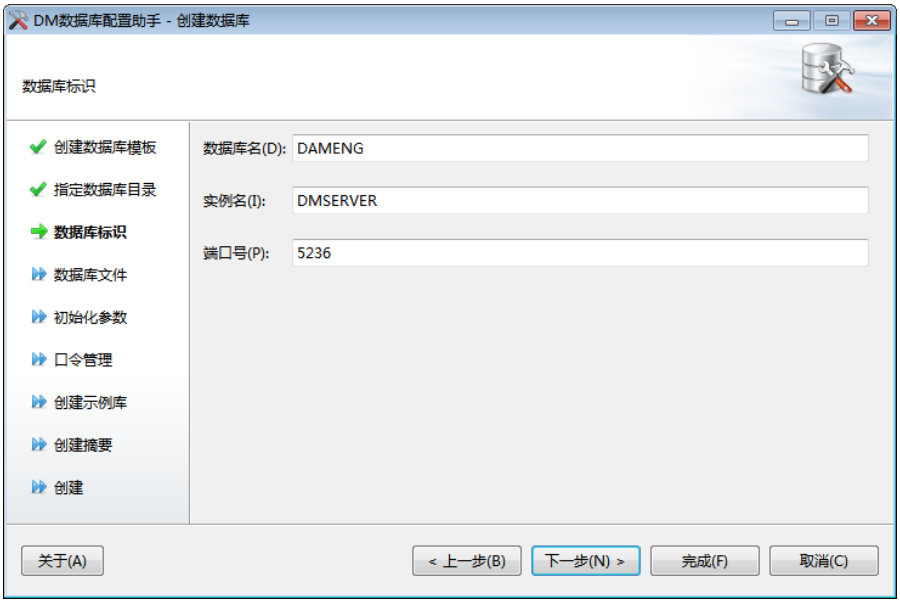


图 7.5 设置数据库标识

4） 数据库文件

如图 7.6 所示，此界面包含四个选项卡：“控制文件”、“数据文件”、“日志文件”和“初始化日志”，可以通过双击路径来更改文件路径。

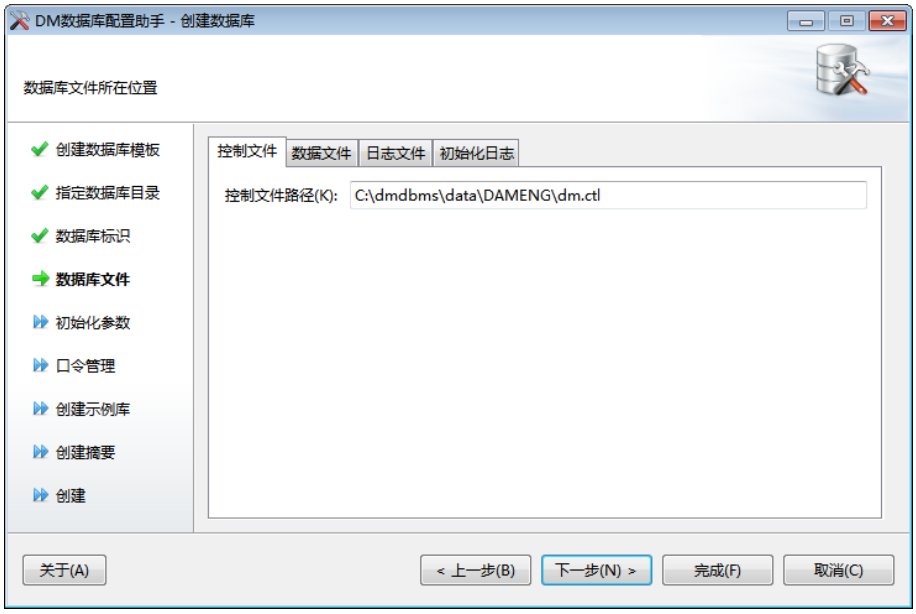


图 7.6 设置数据库文件路径

控制文件

与配置文件类似，控制文件对系统的运行及性能有很大的影响，但不同的是，配置文件中的配置项可以随意更改，而控制文件中的控制信息一般在系统第一次创建完毕后就不能随意更改，所以控制文件不是一个文本文件，而是一个二进制文件。另外控制文件一旦被破坏或丢失仍就无法手工重新创建，基于这一点，对控制文件的保护显得尤为重要。

整个系统中只有一个控制文件，其中存储的控制信息包括主要数据文件路径、日志文件路径、LSN 信息等。由于控制文件对系统至关重要，如果控制文件损坏，系统将无法启动，因此，DM 允许在创建数据库时指定多个控制文件的镜像。这些控制文件的内容是一模一样的，系统每次写控制文件时会顺序对它们进行修改。如果系统在写某一个控制文件时发生硬件故障导致该文件损坏，可以通过其他的控制文件来恢复这个损坏的控制文件，之后重新启动数据库。

数据文件

“数据文件”选项卡用来指定系统表空间路径、用户表空间路径、回滚表空间路径和临时表空间路径。同时还可以指定系统表空间镜像、用户表空间镜像、回滚表空间镜像路径。三个镜像文件是分别和系统表空间、用户表空间、回滚表空间一模一样的文件，当系统表空间、用户表空间、回滚表空间文件损坏时，就可以使用相应的镜像文件来替换。

数据文件是数据库中最重要的文件类型之一，这是数据最终要存储的地方，每个数据库至少有一个与之相关的数据文件，通常情况下，将会有多个。为了理解 DM Server 如何组织这些文件，以及数据在它们内部是如何组织的，必须理解数据页和簇的概念，它们都是 DM Server 用于保存数据库对象的分配单元。

数据页是系统进行磁盘 IO 和缓冲区调度的单元，其大小在数据库创建时就固定下来了，而且一旦固定就不可更改，它们的容量也都是相同的。所有数据页的格式大致相同。

簇是数据文件中一个连续的分配空间，簇由多个数量固定的数据页组成。数据文件对空间的标识都以簇为单位，每个数据文件都维护着两条链，一条为半空簇的链，另一条为自由簇的链，其中半空链用于标识文件中所有被用过一部分的簇，自由链则标识文件中所有未被用过的簇。通常情况下，系统在分配空间时，以簇为单位分配会更有效。

日志文件

重做日志文件对于 DM Server 是至关重要的。它们用于存储数据库的事务日志，以便系统在出现系统故障和介质故障时能够进行故障恢复。在 DM Server 中，任何修改数据库

的操作都会产生重做日志，这样，当系统出现故障时，通过分析日志可以知道在故障发生前系统做了哪些动作，并可以重做这些动作使系统恢复到故障之前的状态。

初始化日志

初始化日志用来指定初始化过程中生成的日志文件所在路径。

5） 初始化参数

数据文件使用的簇大小，即每次分配新的段空间时连续的页数，只能是 16 页或 32 页，缺省使用 16 页。

数据文件使用的页大小，可以为 4K、8K、16K 或 32K，选择的页大小越大，则 DM 支持的元组长度也越大，但同时空间利用率可能下降，缺省使用 8K。

日志文件使用的大小，默认是 64，范围为 64 和 2048 之间的整数，单位为 M。如图 7.7 所示。

时区设置，默认是+08:00，范围为-12:59 和+14:00 之间，如图 7.7 所示。

页面检查，默认是不启用，选项包括不启用、简单检查、严格检查，如图 7.7 所示。

字符集，默认是 GB18030，选项包括 GB18030、Unicode、EUC-KR，如图 7.7 所示。

图 7.7 初始化参数

6） 口令管理

为了数据库管理安全，提供了为数据库的 SYSDBA 和 SYSAUDITOR 系统用户指定新口令功能，如果安装版本为安全版，将会增加 SYSSSO 和 SYSDBO 用户的密码修改。用户可以选择为每个系统用户设置不同口令（留空表示使用默认口令），也可以为所有系统用户设置同一口令。口令必须是合法的字符串，不能少于 9 个或多于 48 个字符。如图 7.8 所示。

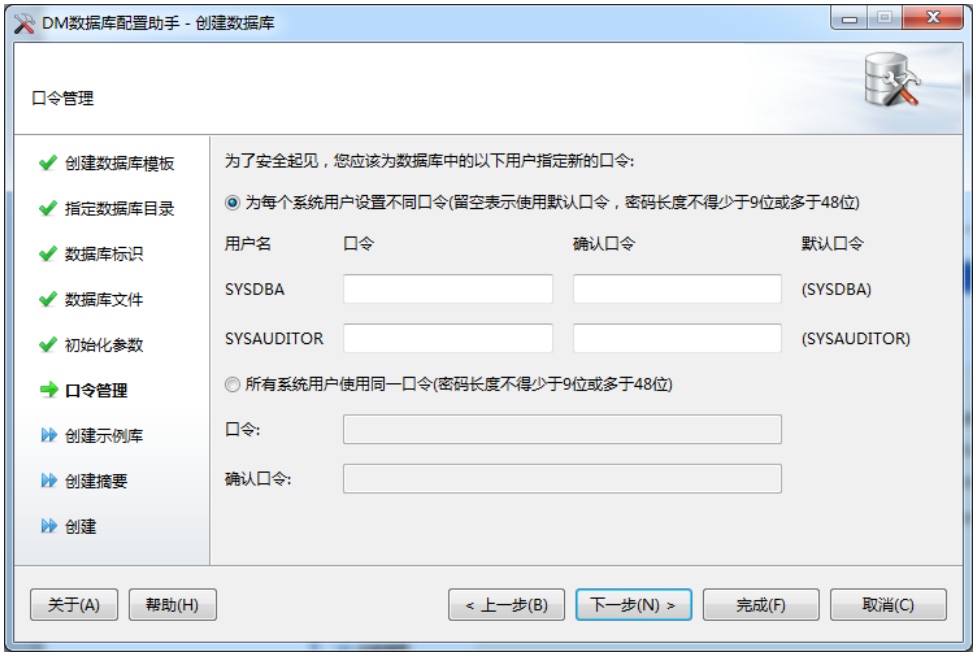


图 7.8 口令管理

7） 创建示例库

示例库 BOOKSHOP 模拟武汉代理图书的某销售公司，该公司欲建立在线购物平台来拓展其代理产品的销售渠道。该示例在 DM 各演示程序中使用。安装该示例后，将在数据库中创建 BOOKSHOP 表空间，同时创建 RESOURCES、PERSON、PRODUCTION、PURCHASING、SALES、OTHER 这 6 个模式和相关的表。

示例库 DMHR 模拟武汉达梦数据库有限公司人力资源管理系统。安装完该示例库，将创建一个模式 DMHR 和一个表空间 DMHR，在 DMHR 模式下创建 REGION、CITY、LOCATION、DEPARTMENT、JOB、EMPLOYEE、JOB_HISTORY 等 7 张表，并插入数据。

如图 7.9 所示。

图 7.9 创建示例库

8） 创建摘要

列举创建数据库纲要，会列举创建时指定的数据库名、示例名、数据库目录、端口、控制文件路径、数据文件路径、日志文件路径、簇大小、页大小、日志文件大小、标识符大小写是否敏感、是否使用 unicode 等信息，方便用户确认创建信息是否符合自己的需求，及时返回修改。如图 7.10 所示。

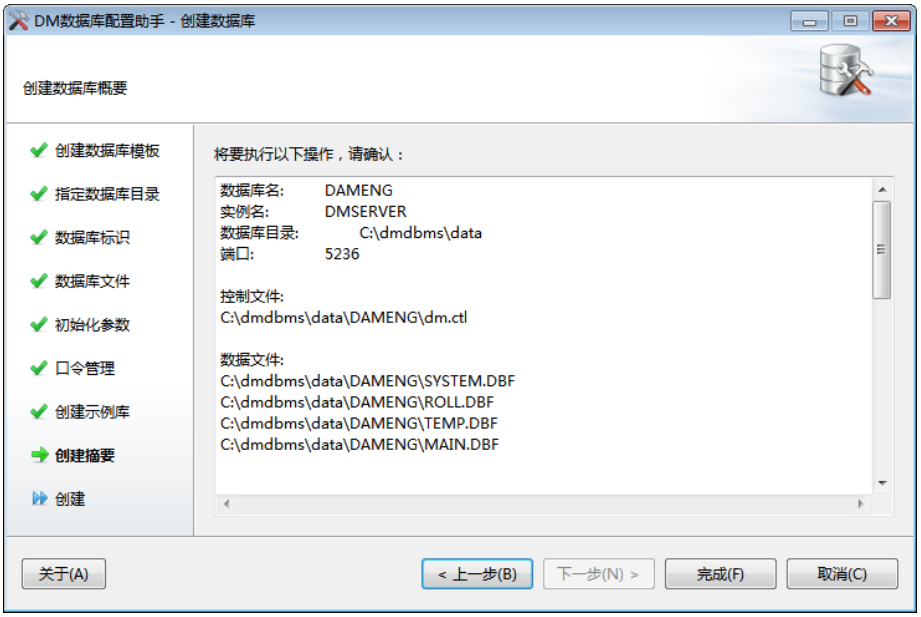


图 7.10 创建摘要

9） 创建数据库

核对完创建信息后，开始创建数据库、创建并启动实例、创建示例库。如图 7.11 所示。

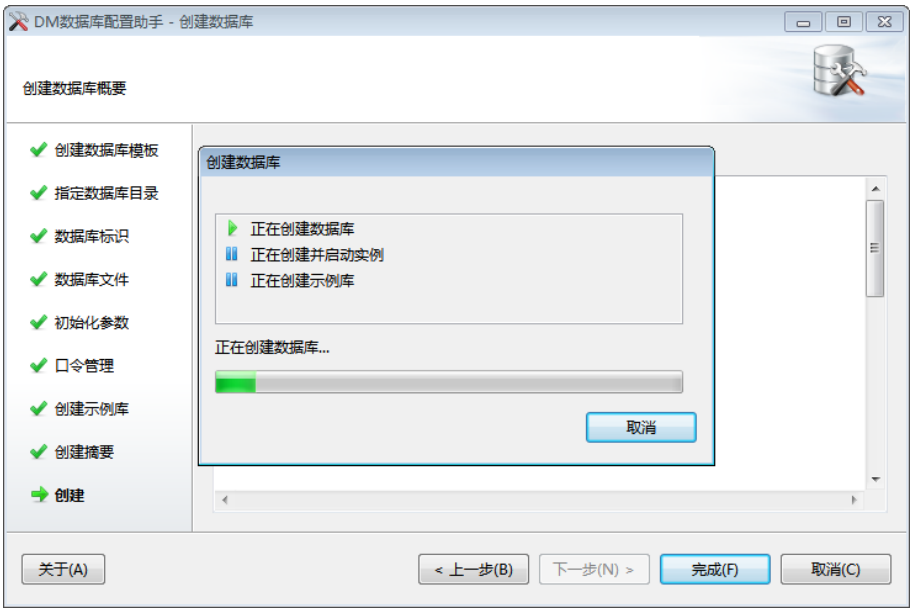


图 7.11 创建数据库

安装完成之后将显示对话框，提示安装完成或错误反馈信息，如图 7.12 所示。

图 7.12 创建数据库完成

如果数据库配置工具运行在 Linux、Solaris、AIX 和 HP-UNIX 系统中，使用非 root 系统用户在创建数据库完成时，将弹出提示框，提示应以 root 系统用户执行以下命令，用来创建数据库的随机启动服务，如图 7.13 所示。

图 7.13 执行命令提示

7.3 使用 dminit 创建数据库

在安装 DM 的过程中，用户可以选择是否创建初始数据库，如果当时没有创建，也可以在完成安装后，利用初始化库工具 dminit 来创建。系统管理员可以利用该工具提供的各种参数，设置数据库存放路径、段页大小、是否对大小写敏感以及是否使用 unicode，创建出满足用户需要的初始数据库。该工具位于安装目录的 bin\目录下。更多详细的功能请参考《DM7_dminit 使用手册》。

在 Windows“命令提示符”窗口中输入带参数的 dminit 命令启动 dminit 工具，命令格式为 dminit [para=value][para=value]……，参数说明见表 7.1。

表 7.1 dminit 工具参数说明

参数	含义	取值	备注
INI_FILE	已有 INI 文件的路径，此 INI 文件用于将其所有参数值作为当前新生成 INI 文件的参数值	合法的路径。文件路径长度最大为 257(含结束符)，不包括文件名	可选
PATH	初始数据库存放的路径，默认路径为 dminit.exe 当前所在的工作目录	合法的路径。文件路径长度最大为 257(含结束符)，不包括文件名	可选
CTL_PATH	初始数据库控制文件的路径，默认值 windows 下是 **PATH****DB_NAME**\dm.ctl，linux 下是/**PATH**/**DM_NAME**/dm.ctl(粗体表示前面设置的参数)	合法的路径。文件路径长度最大为 257(含结束符)，不包括文件名	可选

LOG_PATH	初始数据库日志文件的路径，默认值 windows 下是 **PATH****DB_NAME**\DB_NAME01.log 和 **PATH****DB_NAME**\DB_NAME02.log，linux 下是 **PATH**/**DB_NAME**/DB_NAME01.log 和 **PATH**/**DB_NAME**/DB_NAME02.log（粗体表示前面设置的参数）	合法的路径。文件路径长度最大为 257(含结束符)，不包括文件名。日志文件路径个数不超过 10 个	可选
EXTENT_SIZE	数据文件使用的簇大小，即每次分配新的段空间时连续的页数	只能是 16 页或 32 页之一，缺省使用 16 页	可选
PAGE_SIZE	数据文件使用的页大小，可以为 4K、8K、16K 或 32K 之一，选择的页大小越大，则 DM 支持的元组长度也越大，但同时空间利用率可能下降，缺省使用 8K	只能是 4K、8K、16K 或 32K 之一	可选
LOG_SIZE	日志文件使用的簇大小，以 M 为单位，默认每个日志文件大小为 256M	64 和 2048 之间的整数	可选
CASE_SENSITIVE	标识符大小写敏感，默认值为 Y。当大小写敏感时，小写的标识符应用双引号括起，否则被转换为大写；当大小写不敏感时，系统不自动转换标识符的大小写，在标识符比较时也不区分大小写	只能是 Y, y, N, n,1,0 之一	可选
CHARSET/UNICODE_FLAG	字符集选项。0 代表 GB18030,1 代表 UTF-8,2 代表韩文字符集 EUC-KR	取值 0、1 或 2 之一。默认值为 0	可选
AUTO_OVERWRITE	0 不覆盖，表示建库目录下如果没有同名文件，直接创建。如果遇到同名文件时，屏幕提示是否需要覆盖，由用户手动输入是与否（y/n，1/0）； 1 部分覆盖，表示覆盖建库目录下所有同名文件； 2 完全覆盖，表示先清理掉建库目录下所有文件再重新创建。 默认值为 0。	只能是 0,1,2 之一	可选
LENGTH_IN_CHAR	VARCHAR 类型对象的长度是否以字符为单位。 1：是，设置为以字符为单位时，定义长度并非真正按照字符长度调整，而是将存储长度值按照理论字符长度进行放大。所以会出现实际可插入字符数超过定义长度的情况，这种情况也是允许的。同时，存储的字节长度 8188 上限仍然不变，也就是说，即使定义列长度为 8188 字符，其实际能插入的字符串占用总字节长度仍然不能超过 8188； 0：否，所有 VARCHAR 类型对象的长	取值 0 或 1。默认值为 0	可选

	度以字节为单位		
USE_NEW_HASH	字符类型在计算 HASH 值时所采用的 HASH 算法类别。0：原始 HASH 算法；1：改进的 HASH 算法。默认值为 1。	取值 0 或 1	可选
SYSDBA_PWD	初始化时设置 SYSDBA 的密码，默认为 SYSDBA	合法的字符串，长度在 6 到 48 个字符之间	可选
SYSAUDITOR_PWD	初始化时设置 SYSAUDITOR 的密码，默认为 SYSAUDITOR	合法的字符串，长度在 6 到 48 个字符之间	可选
DB_NAME	初始化数据库名字，默认是 DAMENG	有效的字符串，不超过 128 个字符	可选
INSTANCE_NAME	初始化数据库实例名字，默认是 DMSERVER	有效的字符串，不超过 128 个字符	可选
PORT_NUM	初始化时设置 dm.ini 中的 PORT_NUM，默认 5236	取值范围：1024~65534	可选
TIME_ZONE	初始化时区，默认是东八区	格式为[正负号]小时[：分钟]（正负号和分钟为可选）。时区设置范围为：-12:59~+14:00	可选
PAGE_CHECK	是否启用页面内容校验，0：不启用；1:简单校验;2:严格校验(使用 CRC16 算法生成校验码)。默认 0	取值范围：0~2	可选
RAC_NODE	高性能集群的节点数目	取值范围 2~16，单站点时不写	可选
EXTERNAL_CIPHER_NAME	设置默认加密算法	有效的字符串，不超过 128 个字符	可选
EXTERNAL_HASH_NAME	设置默认 HASH 算法	有效的字符串，不超过 128 个字符	可选
EXTERNAL_CRYPTO_NAME	设置根密钥加密引擎	有效的字符串，不超过 128 个字符	可选
ENCRYPT_NAME	全库加密密钥使用的算法名。算法可以是 DM 内部支持的加密算法，或者是第三方的加密算法。默认使用"AES256_ECB"算法加密	合法的字符串，最长为 128 个字节	可选
RLOG_ENC_FLAG	设置联机日志文件和归档日志文件是否加密	取值 Y/N，y/n，1/0，默认 N	可选
USBKEY_PIN	USBKEY PIN,用于加密服务器根密钥	合法的字符串，最长为 48 个字节	可选
BLANK_PAD_MODE	设置字符串比较时，结尾空格填充模式是否兼容 ORACLE	取值 0 或 1。0 不兼容，1 兼容。默认为 0	可选
SYSTEM_MIRROR_PATH	指定 system.dbf 文件的镜像路径	绝对路径，默认为空	可选
MAIN_MIRROR_PATH	指定 main.dbf 文件的镜像路径	绝对路径，默认为空	可选
ROLL_MIRROR_PATH	指定 roll.dbf 文件的镜像路径	绝对路径，默认为空	可选
MAL_FLAG	初始化时设置 dm.ini 中的	取值 0 或 1	可

	MAL_INI，默认 0		选
ARCH_FLAG	初始化时设置 dm.ini 中的 ARCH_INI，默认 0	取值 0 或 1	可选
MPP_FLAG	Mpp 系统内的库初始化时设置 dm.ini 中的 mpp_ini，默认 0。	取值 0 或 1	可选
CONTROL	指定初始化配置文件路径。初始化配置文件是一个保存了各数据文件路径设置等信息的文本。使用 control 初始化时，若文件已存在，系统会屏幕打印提示，然后直接覆盖	主要用于将数据文件放在裸设备或 rac 环境下	可选
DCP_MODE	是否是 DCP 代理模式	取值：1 是；0 否。默认值为 0	可选
DCP_PORT_NUM	DCP 代理模式下管理端口	取值范围：1024~65534。当 DCP_MODE=1 时，该参数才有效	可选
SYSSSO_PWD	初始化时设置 SYSSSO 的密码，默认为 SYSSSO，仅在安全版本下可见和可设置	合法的字符串，长度在 6 到 48 个字符之间	可选
SYSDBO_PWD	初始化时设置 SYSDBO 的密码，默认为 SYSDBO，仅在安全版本下可见和可设置	合法的字符串，长度在 6 到 48 个字符之间	可选
PRIV_FLAG	是否是四权分立。默认值为 0（不使用），四权分立的具体权限见附录 7。默认情况下，使用三权分立。仅在安全版本下可见和可设置	只能是 0 或 1	可选
ELOG_PATH	指定初始化过程中生成的日志文件所在路径	合法的路径。文件路径长度最大为 257(含结束符)，不包括文件名	可选
HELP	显示帮助信息		可选

说明：dminit 一般是要有参数的，如果没有带参数，系统就会引导用户设置。另外，参数、等号和值之间不能有空格。Help 参数的后面不用添加'='号。

命令举例：

```
dminit PATH=c:\dm7data PAGE_SIZE=16
```

如果创建成功，则屏幕显示如图 7.13 所示。此时在 c 盘根目录下出现 dm7Data 文件夹，内容包含初始数据库 DAMENG 的相关文件和初始化文件 dm.ini。将 dm.ini 文件拷贝到 DM 安装目录的 bin 下，DM 服务器就可以启动该初始数据库了。

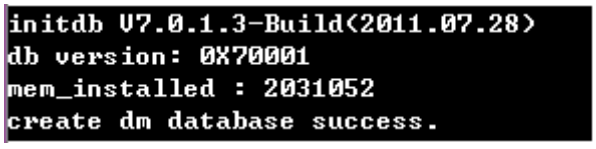


图 7.14 创建成功后的屏幕显示

7.4 注册数据库服务

注册数据库服务，该功能用于将使用命令行工具生成的数据库，重新注册成系统服务，方便用户管理与控制。

用户注册数据库服务可以通过图形化界面和 shell 脚本两种方式，本节讲述的是通过数据库配置工具注册数据库服务，通过 shell 脚本注册数据库服务详细操作请参见《DM7_Linux 服务脚本手册》。如图 7.16 所示。

图 7.16 选择操作方式

用户选择 dm.ini 文件来注册相应的数据库，并可修改相应的端口号和实例名，也可以选择是否以配置状态启动数据库，如图 7.17 所示。

图 7.17 要删除的数据库

正在注册数据库服务，如图 7.18 所示。

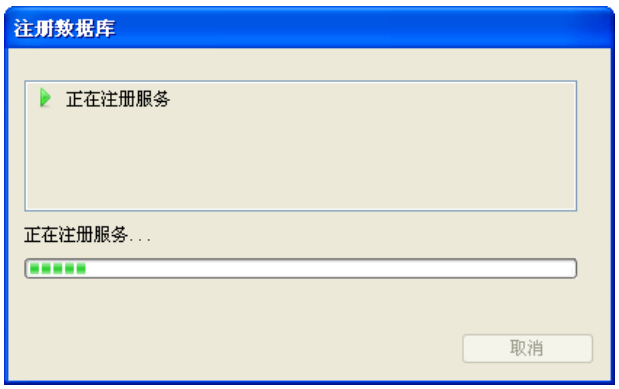


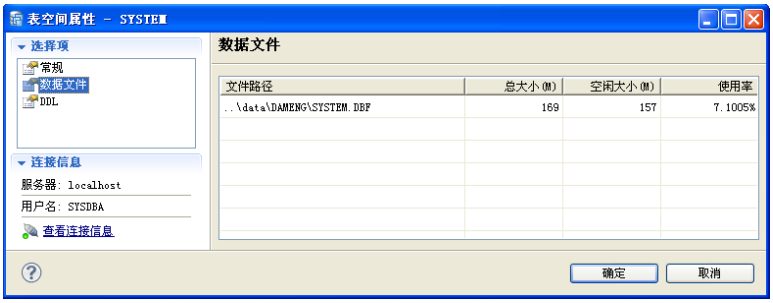
图 7.18 注册数据库服务

注册完成之后将显示对话框，提示完成或错误反馈信息，如图 7.19 所示。

图 7.19 注册数据库完成

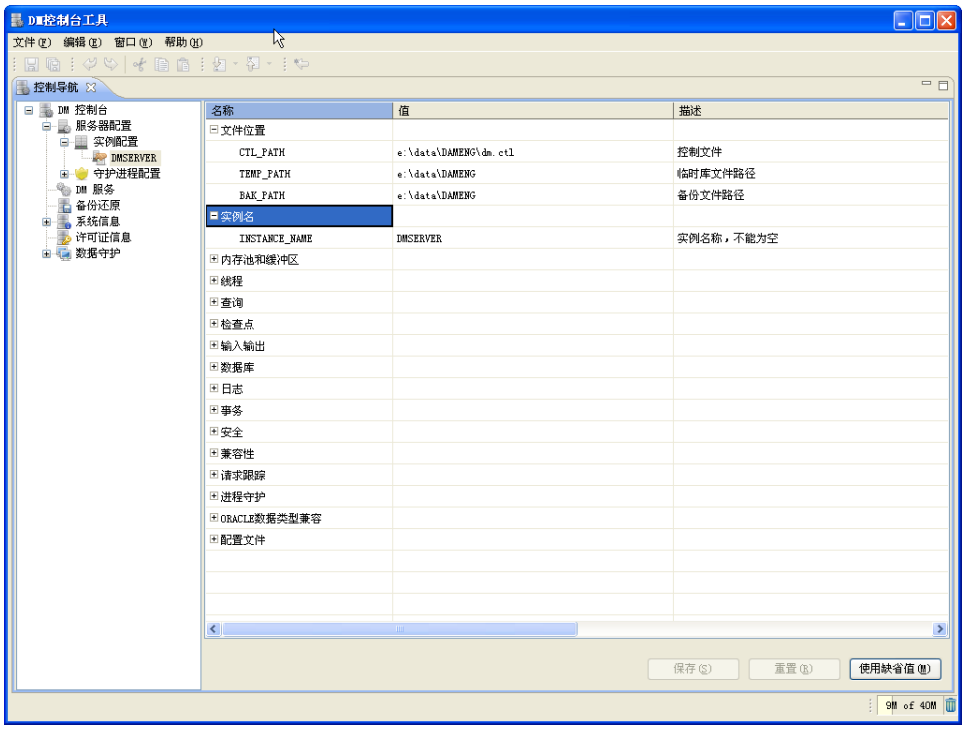
7.5 查看数据库信息

登录数据库管理工具，通过表空间属性，可以查看数据文件路径、总空间大小、空闲空间大小、使用率。如图 7.20 所示。



7.20 查看数据文件

通过 DM 控制台工具，查看实例配置属性，如图 7.21 所示，可以查看到如下信息：文件位置、实例名、内存池和缓冲区、线程、查询、检查点、输入输出、数据库、日志、事务、安全、兼容性、请求跟踪、进程守护、ORACLE 数据类型兼容和配置文件。



7.21 查看数据文件

7.6 删除数据库

删除数据库，包括删除数据库的数据文件、日志文件、控制文件和初始化参数文件。为了保证删除数据库成功，必须保证 dmserver 已关闭。

可以使用数据库配置工具来删除数据库。如图 7.22 所示。

图 7.22 选择操作方式

根据数据库名称，选择要删除的数据库，如图 7.23 所示，也可以通过指定数据库配置文件删除数据库。

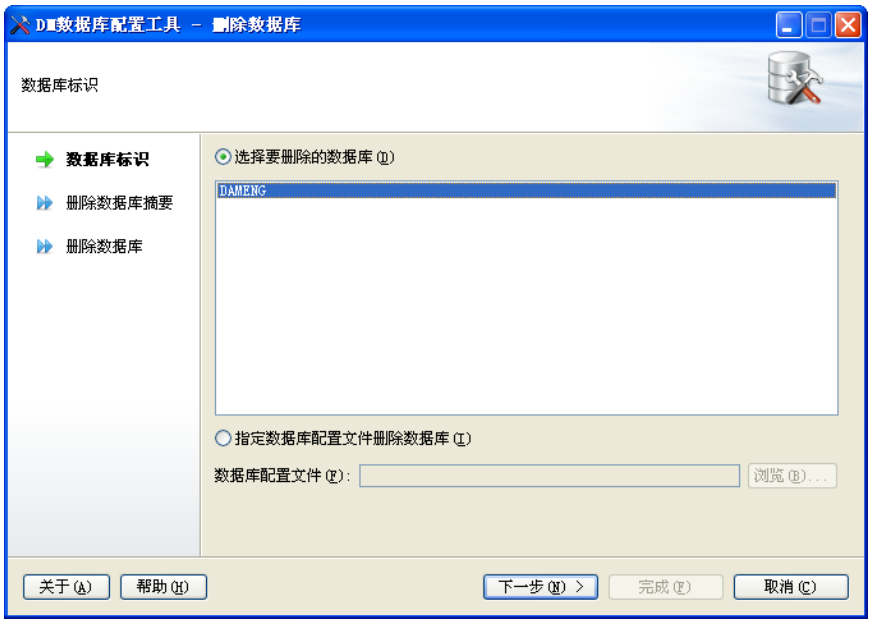


图 7.23 要删除的数据库

确认将删除的数据库名、实例名、数据库目录，如图 7.24 所示。

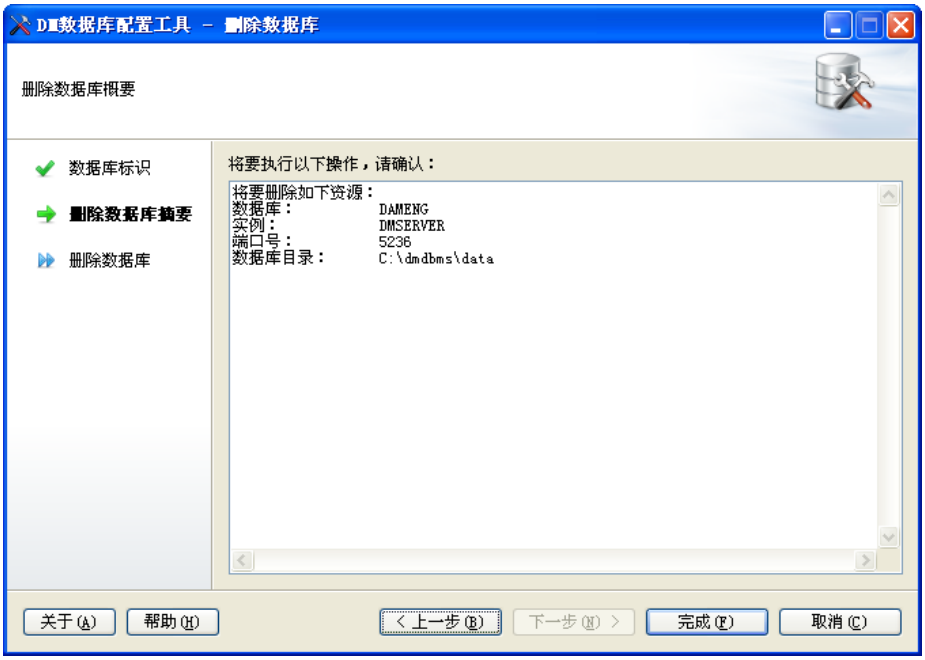


图 7.24 删除数据库概要

首先停止实例，然后删除实例，最后删除数据库，如图 7.25 所示。

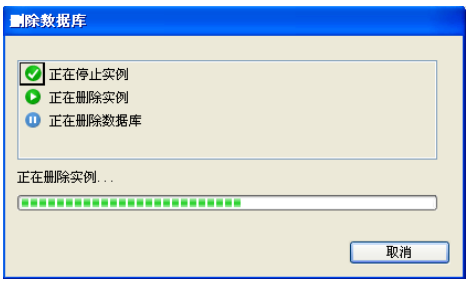


图 7.25 删除数据库

删除完成之后将显示对话框，提示完成或错误反馈信息，如图 7.26 所示。

图 7.26 删除数据库完成

如果数据库配置工具运行在 Linux、Solaris、AIX 和 HP-UNIX 系统中，使用非 root 系统用户在删除数据库完成时，将弹出提示框，提示应以 root 系统用户执行以下命令，用来删除数据库的随机启动服务，如图 7.27 所示。

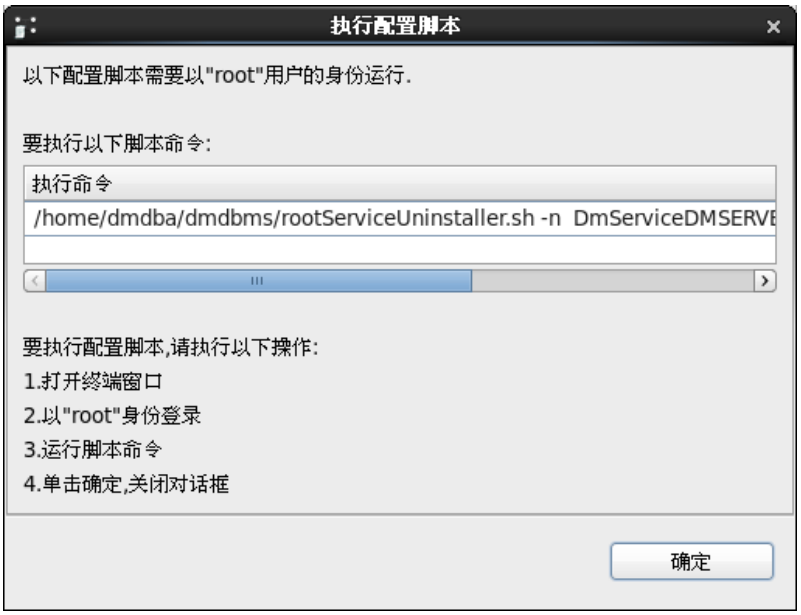


图 7.27 执行命令提示

7.7 删除数据库服务

删除数据库服务，只删除用于启动和停止数据库的服务文件，不会删除数据库的数据文件、日志文件、控制文件和初始化参数文件。

用户删除数据库服务可以通过图形化界面和 shell 脚本两种方式，本节讲述的是通过数据库配置工具删除数据库服务，通过 shell 脚本删除数据库服务详细操作请参见 7.8 节数据库服务配置脚本。如图 7.28 所示。

图 7.28 选择操作方式

根据数据库服务名称，选择要删除的数据库服务，如图 7.29 所示，也可以通过指定数据库配置文件删除数据库服务。

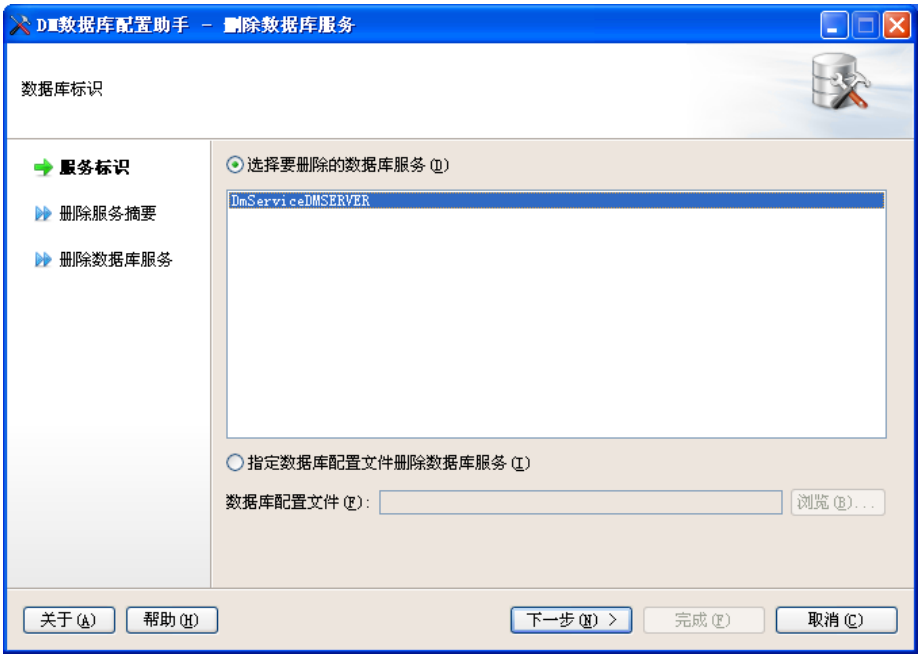


图 7.29 要删除的数据库服务

确认将删除的数据库名、实例名、数据库服务名、数据库目录，如图 7.30 所示。

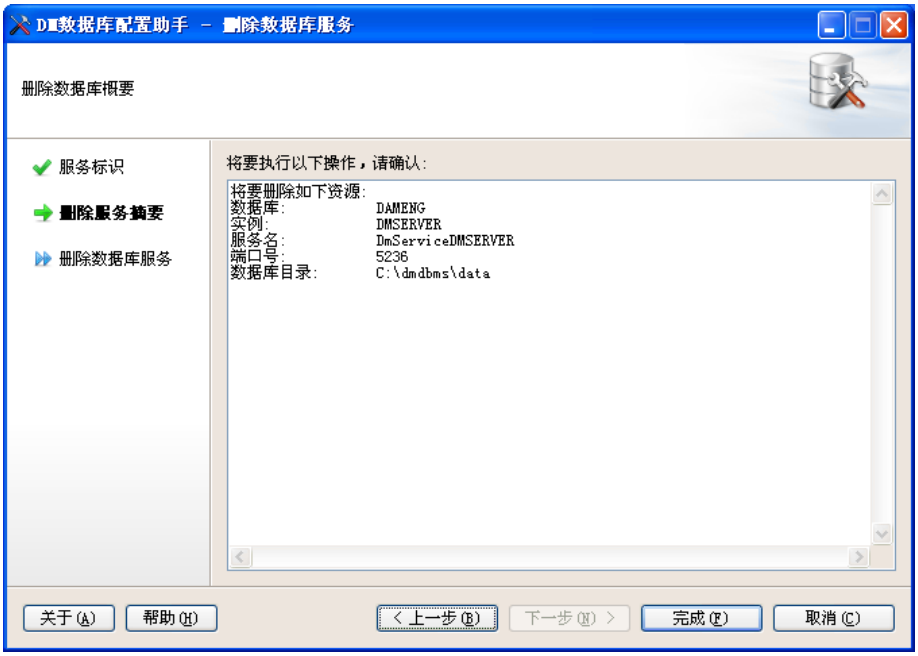


图 7.30 删除数据库概要

首先检查数据库服务，然后删除数据库服务，如图 7.31 所示。

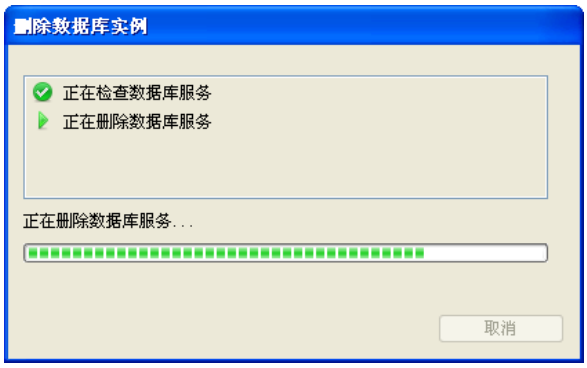


图 7.31 删除数据库服务

删除完成之后将显示对话框，提示完成或错误反馈信息，如图 7.32 所示。

图 7.32 删除数据库服务完成

如果数据库配置工具运行在 Linux、Solaris、AIX 和 HP-UNIX 系统中，使用非 root 系统用户在删除数据库服务完成时，将弹出提示框，提示应以 root 系统用户执行以下命令，用来删除数据库的随机启动服务，如图 7.33 所示。

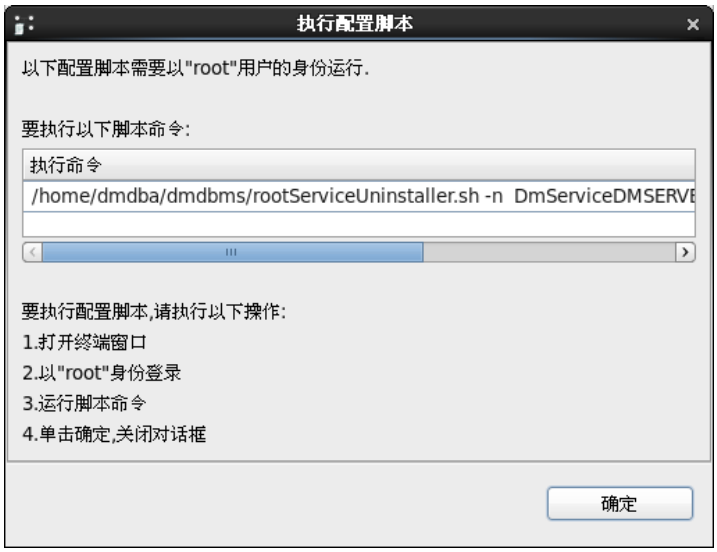


图 7.33 执行命令提示

第8章 启动和关闭数据库

DM 支持多种操作系统。常用的操作系统基本上可以划分为 Windows 操作系统和 Linux 操作系统。下面分别介绍这两种操作系统下启动和关闭 DM 数据库的操作。

8.1 启动数据库

8.1.1 Windows 系统

1. 菜单方式

安装 DM 数据库后(默认情况下安装成功后 DM 服务会自动启动)，在 Windows 的开始菜单选项中选择如图 8.1 所示的菜单项中的 DM 服务查看器可以启动 DM 数据库。

图 8.1 DM 数据库菜单方式启动 1

点击 DM 服务查看器选项后，会弹出如图 8.2 所示的界面：

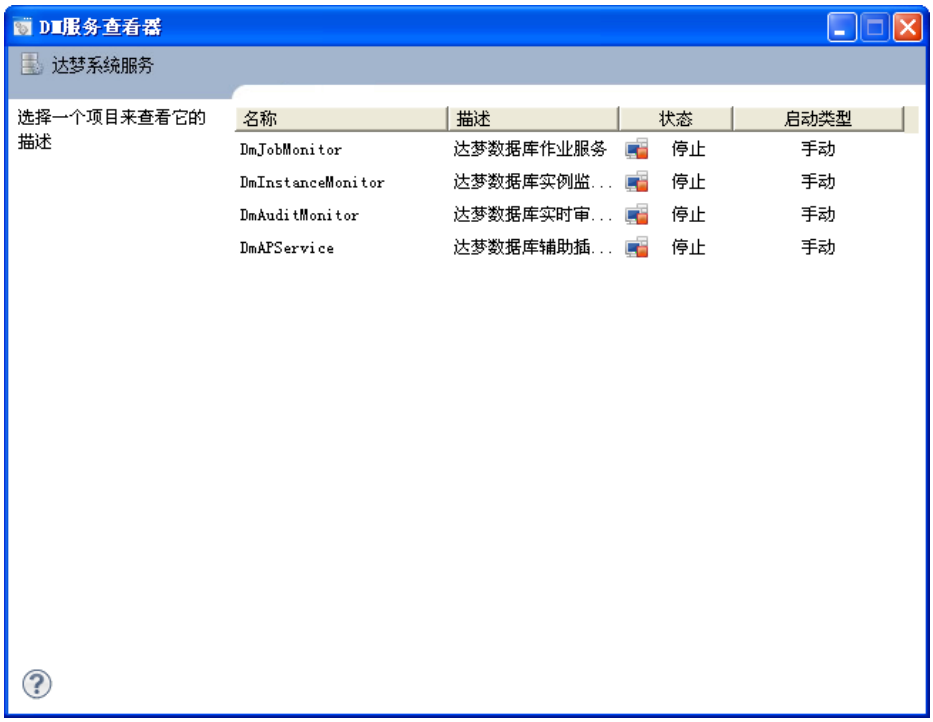


图 8.2 DM 数据库菜单方式启动 2

在弹出界面中选中所要启动的数据库，点击鼠标右键，在菜单栏中选择启动。

2. Windows 服务方式

安装 DM 数据库并且新建一个 DM 实例后。Windows 的服务中会自动增加一项和该实例

名对应的服务。例如新建一个实例名为 DMSERVER 的 DM 数据库，Windows 的服务中会增加一项名称为“DmServiceDMSERVER”的服务。打开 Windows 的管理工具，选择“服务”，打开 Windows 服务控制台，如图 8.3 所示，选择“DmServiceDMSERVER”，用鼠标在工具栏点击启动按钮或者点击鼠标右键，在菜单栏中选择“启动”，启动 DM 数据库。

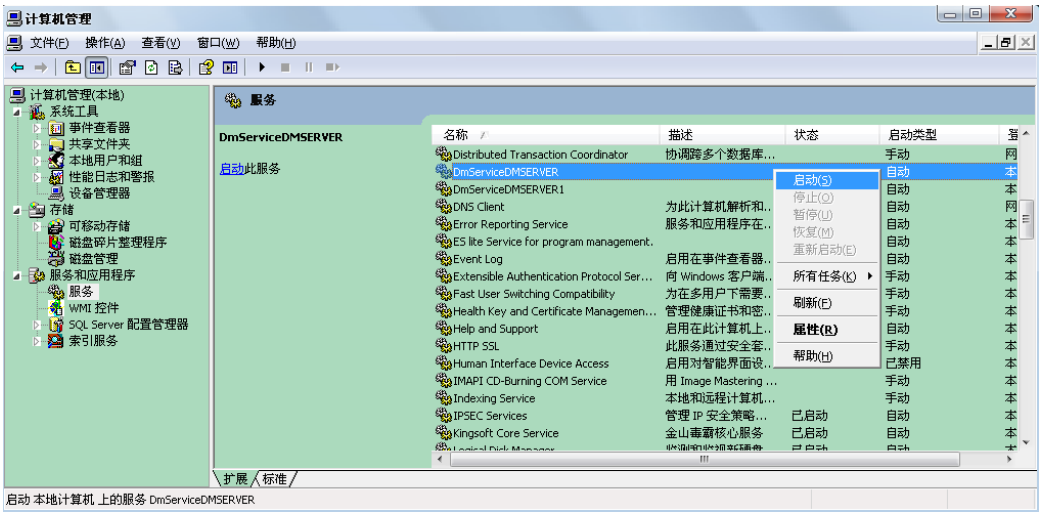


图 8.3 DM 数据库服务方式启动

3. 命令行方式

进入 DM 安装目录下的 bin 目录，直接打开应用程序 dmserver 就可以启动 DM 数据库。或者先打开 Windows 命令提示符工具，在命令工具中执行命令进入 DM 服务器的目录，再执行 dmserver 的命令启动 DM 数据库，如图 8.4 所示。

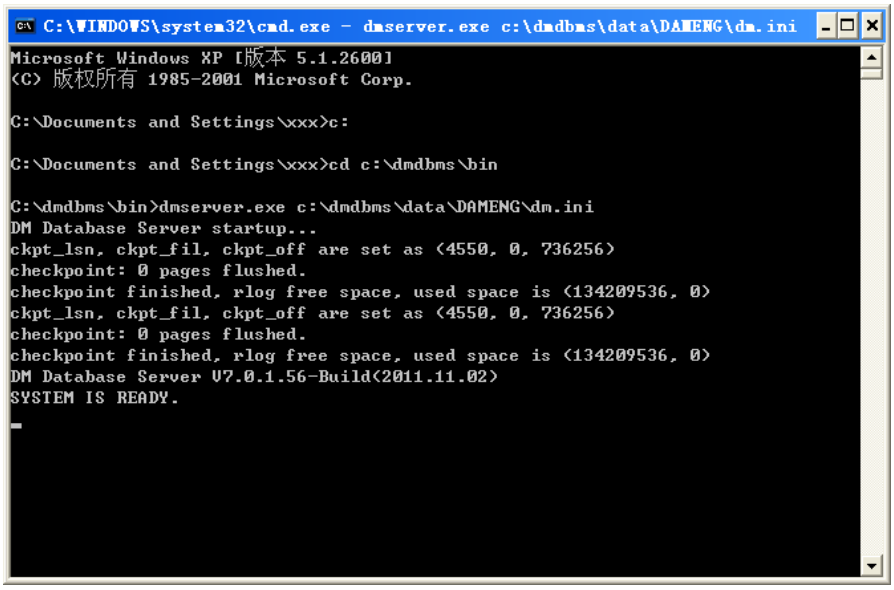


图 8.4 DM 数据库命令行方式启动

命令行方式启动参数如下：

```
dmserver [ini_file_path] [-noconsole] [mount]
```

说明：

1. Dmserver 命令行启动参数可指定 dm.ini 文件的路径，非控制台方式启动及指定数据库是否以 MOUNT 状态启动。关于数据库状态见下一节介绍；
2. Dmserver 启动时可不指定任何参数，默认使用当前目录下的 dm.ini 文件，如果当前目录不存在 dm.ini 文件，则无法启动；
3. Dmserver 启动时可以指定-noconsole 参数。如果以此方式启动，则无法通过在

控制台中输入服务器命令。

当不确定启动参数的使用方法时，可以使用 help 参数，将打印出格式、参数说明和使用示例。使用方法如下：

```
dmserver help
```

当以控制台方式启动 Dmserver 时，用户可以在控制台输入一些命令，服务器将在控制台打印出相关信息或执行相关操作。支持的命令见下表。

表 8.1 Dmserver 控制台支持的命令

命令	操作
EXIT	退出服务器
LOCK	打印锁系统信息
TRX	打印等待事务信息
CKPT	设置检查点
BUF	打印内存池中缓冲区的信息
MEM	打印服务器占用内存大小
SESSION	打印连接个数
DEBUG	打开 DEBUG 模式

8.1.2 Linux 系统

1. 菜单方式

安装 DM 数据库后(默认情况下安装成功后 DM 服务会自动启动)，在 Linux 的开始菜单选项中选择启动服务器菜单项可以启动 DM 数据库。启动方式类似 Windows。

2. Linux 服务方式

安装 DM 数据库后，在/etc/rc.d/init.d 中有名称为 DmService 开头的文件，文件全名为 DmService+实例名（例：如果实例名为 DMSERVER，则相对应的服务文件为 DmServiceDMSERVER ）。以实例名为 DMSERVER 为例，在终端输入./DmServiceDMSERVER start 或者 service DmServiceDMSERVER start 即可启动 DM 数据库。

3. 命令行方式

在终端进入 DM 安装目录下的 bin 目录，执行./dmserver 启动 DM 数据库，参数选项同 Windows。

8.1.3 检查 LICENSE

无论是在何种操作系统下运行，DM 数据库在启动时都会进行 LICENSE 检查。若 LICENSE 过期或 KEY 文件与实际运行环境不配套，DM 服务器会强制退出。

可通过查看 V$LICENSE 了解所安装的 DM 数据库的 LICENSE 信息。

8.2 数据库状态和模式

DM 数据库包含以下几种状态：

1. 配置状态（MOUNT）：不允许访问数据库对象，只能进行控制文件维护、归档配置、数据库模式修改等操作；

2． 打开状态（OPEN）：不能进行控制文件维护、归档配置等操作，可以访问数据库对象，对外提供正常的数据库服务；

3． 挂起状态（SUSPEND）：与 OPEN 状态的唯一区别就是，限制磁盘写入功能；一旦修改了数据页，触发 REDO 日志、数据页刷盘，当前用户将被挂起。

OPEN 状态与 MOUNT 和 SUSPEND 能相互转换，但是 MOUNT 和 SUSPEND 之间不能相互转换。

DM 数据库包含以下几种模式：

1． 普通模式（NORMAL）：用户可以正常访问数据库，操作没有限制；

2． 主库模式（PRIMARY）：用户可以正常访问数据库，所有对数据库对象的修改强制生成 REDO 日志，在归档有效时，发送 REDO 日志到备库；

3． 备库模式（STANDBY）：接收主库发送过来的 REDO 日志并重做。数据对用户只读。

三种模式只能在 MOUNT 状态下设置，模式之间可以相互转换。

对于新初始化的库，首次启动不允许使用 mount 方式，需要先正常启动并正常退出，然后才允许 mount 方式启动。

一般情况下，数据库为 NORMAL 模式，如果不指定 MOUNT 状态启动，则自动启动到 OPEN 状态。

在需要对数据库配置时（如配置数据守护、数据复制），服务器需要指定 MOUNT 状态启动。当数据库模式为非 NORMAL 模式（PRIMARY、STANDBY 模式），无论是否指定启动状态，服务器启动时自动启动到 MOUNT 状态。

8.3 关闭数据库

8.3.1 Windows 系统

1． 菜单方式

在 Windows 的开始->程序菜单中选择达梦数据库->DM 服务查看器，在弹出的界面中，选中要关闭的数据库，点击鼠标右键，在菜单栏中选择停止。

2． Windows 服务方式

安装 DM 数据库并且新建一个 DM 实例后。Windows 的服务中会自动增加一项和该实例名对应的服务。例如新建一个实例名为 DMSERVER1 的 DM 数据库，Windows 的服务中会增加一项名称为“DmServiceDMSERVER1”的服务。打开 Windows 的管理工具，选择服务，打开 Windows 服务控制台，如图 8.5 所示，选择“DmServiceDMSERVER1”，用鼠标在工具栏点击停止按钮或者点击鼠标右键，在菜单栏中选择停止，关闭 DM 数据库。

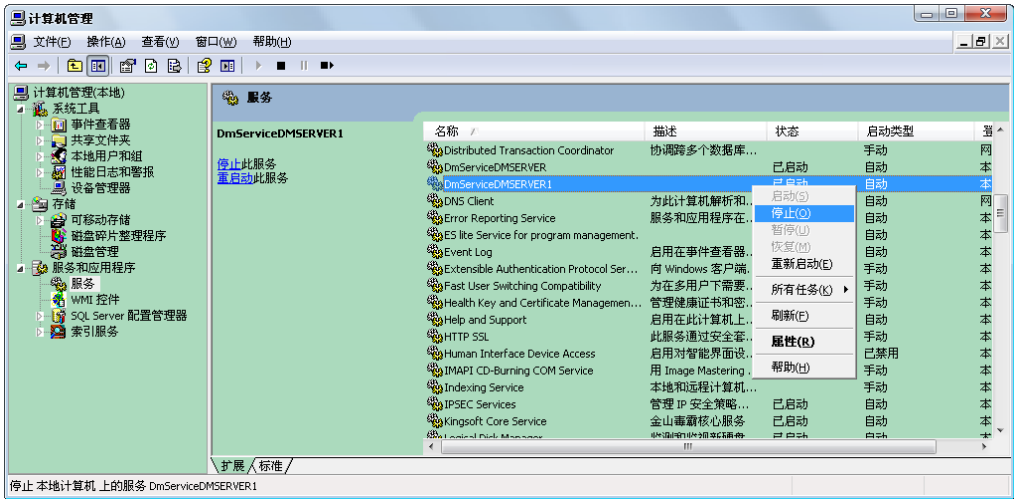


图 8.5 DM 数据库服务方式停止

3. 命令行方式

在启动数据库的命令工具中输入 exit，然后回车，关闭 DM 数据库。如图 8.6 所示。

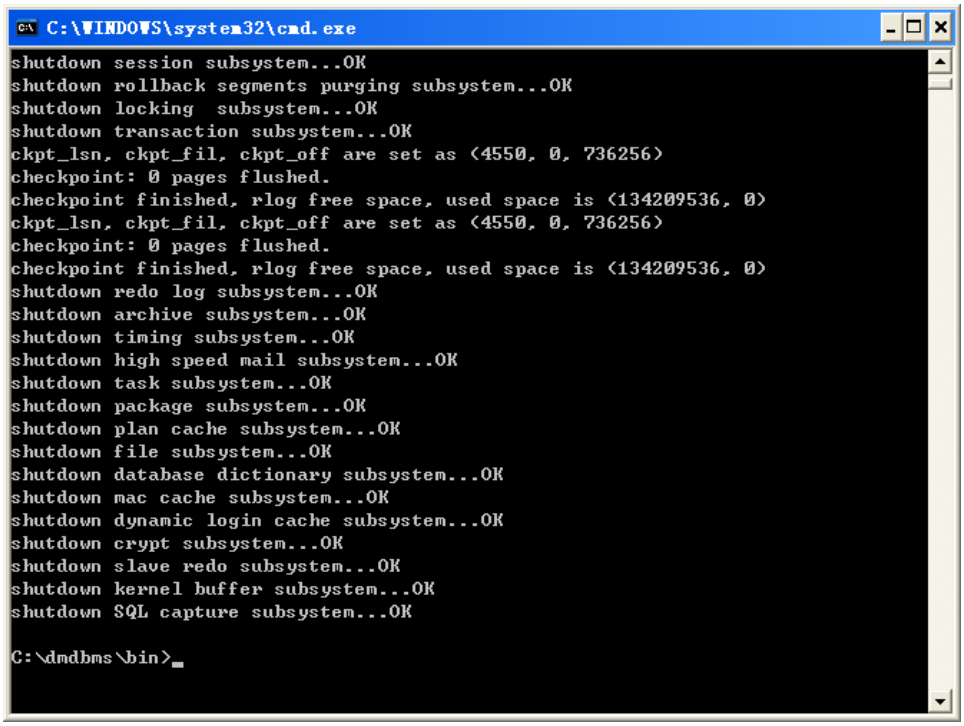


图 8.6 DM 数据库命令行方式停止

8.3.2 Linux 系统

1. 菜单方式

在 Linux 的开始菜单中选择关闭数据库的菜单项可以关闭 DM 数据库。关闭方式类似 Windows。

2. Linux 服务方式

进入/etc/rc.d/init.d，以实例名为 DMSERVER 为例，在命令行工具中输入./DmServiceDMSERVER stop 即可关闭 DM 数据库。

3. 命令行方式

在启动数据库的命令工具中输入 exit，然后回车，退出 DM 数据库。关闭方式类似 Windows。

第9章 管理模式对象的空间

模式对象的空间管理关系到空间的有效使用和数据的合理分布，本章介绍了模式对象相关的存储参数、多余空间的回收机制、模式对象上的空间限制、如何查看已使用的存储空间以及各种数据类型实际使用的空间大小。

9.1 设置存储参数

9.1.1 普通表和索引

对于普通表和索引，DM7 提供了以下的存储参数：

1. 初始簇数目 INITIAL：指建立表时分配的簇个数，必须为整数，最小值为 1，最大值为 256，缺省为 1；
2. 下次分配簇数目 NEXT：指当表空间不够时，从数据文件中分配的簇个数，必须为整数，最小值为 1，最大值为 256，缺省为 1；
3. 最小保留簇数目 MINEXTENTS：当删除表中的记录后，如果表使用的簇数目小于这个值，就不再释放表空间，必须为整数，最小值为 1，最大值为 256，缺省为 1；
4. 填充比例 FILLFACTOR：指定插入数据时数据页的充满程度，取值范围从 0 到 100。默认值为 0，等价于 100，表示全满填充，未充满的空间可供页内的数据更新时使用。插入数据时填充比例的值越低，可由新数据使用的空间就越多；更新数据时填充比例的值越大，更新导致出现的页分裂的几率越大；
5. 表空间名：在指定的表空间上建表或索引，表空间必须已存在，默认为用户缺省的表空间。

如表 PERSON 建立在表空间 TS_PERSON 中，初始簇大小为 5，最小保留簇数目为 5，下次分配簇数目为 2，填充比例为 85。

```
CREATE TABLE PERSON.PERSON
( PERSONID  INT IDENTITY(1,1) CLUSTER PRIMARY KEY,
SEX CHAR(1) NOT NULL,
NAME VARCHAR(50) NOT NULL,
EMAIL VARCHAR(50),
PHONE VARCHAR(25))
STORAGE
     ( INITIAL     5,  MINEXTENTS   5,
NEXT        2,  ON   TS_PERSON,   FILLFACTOR    85);
```

也可以在分区表上指定某个分区的存储参数，如下面的建表语句指定了 PAR2 分区存储在 TS_PAR2 表空间上。

```
CREATE TABLE PARTITION_TABLE
(C1 INT,
C2 INT)
PARTITION BY RANGE(C1)
 (PARTITION PAR1 VALUES LESS THAN(5),
PARTITION PAR2 VALUES LESS THAN(100) STORAGE (ON TS_PAR2));
```

9.1.2 堆表

对于堆表可以指定并发分支 BRANCH 和非并发分支 NOBRANCH 的数目，其范围是（1=<BRANCH <= 64，1<=NOBRANCH<=64），堆表最多支持 128 个链表，具体可参考第 18 章“管理堆表”。

如下例创建的 LIST_TABLE 表有并发分支 2 个，非并发分支 4 个。

```
CREATE TABLE LIST_TABLE(C1 INT) STORAGE(BRANCH (2,4));
```

9.1.3 HUGE 表

HUGE 表是建立在自己特有的 HTS 表空间上的。建立 HUGE 表如果不使用默认的表空间，则必须要先创建一个 HUGE TABLESPACE，默认 HTS 表空间为 HMAIN。

如建立一个名称 HTS_NAME 的 HTS 表空间,表空间路径为为 e:\HTSSPACE。示例如下：

```
CREATE HUGE TABLESPACE HTS_NAME PATH 'e:\HTSSPACE';
```

对于 HUGE 表可以指定如下参数：

1. 区大小（一个区的数据行数）。 区大小可以通过设置表的存储属性来指定，区的大小必须是 2 的多少次方，如果不是则向上对齐。取值范围：1024 行~1024*1024 行。默认值为 65536 行。
2. 是否记录区统计信息，即在修改时是否做数据的统计。
3. 所属的表空间。创建 HUGE 表，需要通过存储属性指定其所在的表空间，不指定则存储于默认表空间 HMAIN 中。HUGE 表指定的表空间只能是 HTS 表空间。
4. 文件大小。创建 HUGE 表时还可以指定单个文件的大小，通过表的存储属性来指定，取值范围为 16M~1024*1024M。不指定则默认为 64M。文件大小必须是 2 的多少次方，如果不是则向上对齐。
5. 日志属性。1）LOG NONE：不做镜像；2）LOG LAST：做部分镜像；3）LOG ALL：全部做镜像。

如下面的建表语句：STUDENT 表的区大小为 65536 行，文件大小为 64M，指定所在的表空间为 HTS_NAME，做完整镜像，S_COMMENT 列指定的区大小为不做统计信息，其它列（默认）都做统计信息。

```
CREATE HUGE TABLE STUDENT
(
    S_NO            INT,
    S_CLASS         VARCHAR,
    S_COMMENT           VARCHAR(79) STORAGE(STAT NONE)
)STORAGE(SECTION(65536) , FILESIZE(64), ON HTS_NAME) LOG ALL;
```

9.2 收回多余的空间

DM7 中表和索引对象的所占用的簇要么是全满的状态要么是半满的状态，空闲的簇会被系统自动回收。

9.3 用户和表上的空间限制

9.3.1 用户的空间限制

用户占用的空间是其下所有用户表对象占用空间的总和。可以限制用户使用空间的大小，当用户创建表，创建索引，或者插入更新数据超过了指定的空间限制时，会报空间不足的错误。如创建用户 TEST_USER 时可指定该用户使用的最大磁盘空间为 50M。

```
CREATE USER TEST_USER IDENTIFIED BY TEST_PASSWORD DISKSPACE LIMIT 50;
```

对用户的空间限制也可进行更改，如修改用户 TEST_USER 的磁盘空间限制为无限制。

```
ALTER USER TEST_USER DISKSPACE UNLIMITED;
```

9.3.2 表对象的空间限制

表对象占用的空间是其上所有索引占用空间的总和。可以限制表对象使用空间的大小，当在表对象上创建索引或者插入更新数据超过了指定的空间限制时，会报空间不足的错误。如创建表 TEST 时可指定该表对象可使用的最大磁盘空间为 500M。

```
CREATE TABLE TEST (SNO INT, MYINFO VARCHAR) DISKSPACE LIMIT 500;
```

对表对象空间的限制也可进行更改，如修改表 TEST 的磁盘空间限制为 50M。

```
ALTER TABLE TEST MODIFY DISKSPACE LIMIT 50;
```

9.4 查看模式对象的空间使用

9.4.1 查看用户占用的空间

可以使用系统函数 USER_USED_SPACE 得到用户占用空间的大小，函数参数为用户名，返回值为占用的页的数目。

```
SELECT USER_USED_SPACE('TEST_USER');
```

9.4.2 查看表占用的空间

可以使用系统函数 TABLE_USED_SPACE 得到表对象占用空间的大小，函数参数为模式名和表名，返回值为占用的页的数目。

```
SELECT TABLE_USED_SPACE('SYSDBA', 'TEST');
```

9.4.3 查看表使用的页数

可以使用系统函数 TABLE_USED_PAGES 得到表对象实际使用页的数目，函数参数为模式名和表名，返回值为实际使用页的数目。

```
SELECT TABLE_USED_PAGES('SYSDBA', 'TEST');
```

9.4.4 查看索引占用的空间

可以使用系统函数 INDEX_USED_SPACE 得到索引占用空间的大小，函数参数为索引 ID，返回值为占用的页的数目。

```
SELECT INDEX_USED_SPACE(33555463);
```

9.4.5 查看索引使用的页数

可以使用系统函数 INDEX_USED_PAGES 得到索引实际使用页的数目，函数参数为索引 ID，返回值为实际使用页的数目。

```
SELECT INDEX_USED_PAGES(33555463);
```

9.5 数据类型的空间使用

各种数据类型占用的空间是不同的，下表 9.1 列出了主要数据类型所需要的空间。

表 9.1 主要数据类型所需的空间

数据类型	所需空间
CHAR	SIZE 为 1~8188 字节， 具体情况受到页面大小和记录大小的共同限制
VARCHAR	SIZE 为 1~8188 字节， 具体情况受到页面大小和记录大小的共同限制
TINYINT BIT BYTE	需要 1 个字节
SMALLINT	需要 2 个字节
INT	需要 4 个字节
BIGINT	需要 8 个字节
REAL	需要 4 个字节
FLOAT	需要 8 个字节
DOUBLE DOUBLE PRECISION	需要 8 个字节
DEC DECIMAL NUMERIC	SIZE 为 1~20 个字节
BINARY	SIZE 为 1~8188 字节， 具体情况受到页面大小和记录大小的共同限制
VARBINARY	SIZE 为 1~8188 字节， 具体情况受到页面大小和记录大小的共同限制
DATE	需要 3 个字节
TIME	需要 5 个字节
TIMESTAMP DATETIME	需要 8 个字节
TIME WITH TIME ZONE	需要 7 个字节
TIMESTAMP WITH TIME ZONE	需要 10 个字节

INTERVAL YEAR INTERVAL MONTH INTERVAL YEAR TO MONTH	需要 12 个字节
INTERVAL DAY INTERVAL HOUR INTERVAL MINUTE INTERVAL SECOND INTERVAL DAY TO HOUR INTERVAL DAY TO MINUTE INTERVAL DAY TO SECOND INTERVAL HOUR TO MINUTE INTERVAL HOUR TO SECOND INTERVAL MINUTE TO SECOND	需要 24 个字节
BLOB IMAGE LONGVARBINARY	SIZE 为 1~2G 字节
CLOB TEXT LONGVARCHAR	SIZE 为 1~2G 字节

第10章 管理表

表是数据库中数据存储的基本单元，是对用户数据进行读和操纵的逻辑实体。表由列和行组成，每一行代表一个单独的记录。表中包含一组固定的列，表中的列描述该表所跟踪的实体的属性，每个列都有一个名字及各自的特性。

列的特性由两部分组成：数据类型（dataType）和长度（length）。对于 NUMERIC、DECIMAL 以及那些包含秒的时间间隔类型来说，可以指定列的小数位及精度特性。在 DM 系统中，CHAR、CHARACTER、VARCHAR 数据类型的最大长度由数据库页面大小决定，数据库页面大小在初始化数据库时指定。DM 系统具有 SQL-92 的绝大部分数据类型，以及部分 SQL-99、Oracle 和 SQL Server 的数据类型。

为了确保数据库中数据的一致性和完整性，在创建表时可以定义表的实体完整性、域完整性和参考完整性。实体完整性定义表中的所有行能唯一地标识，一般用主键、唯一索引、UNIQUE 关键字及 IDENTITY 属性来定义；域完整性通常指数据的有效性，限制数据类型、缺省值、规则、约束、是否可以为空等条件，域完整性可以确保不会输入无效的值；参考完整性维护表间数据的有效性、完整性，通常通过建立外键联系另一表的主键来实现。

如果用户在创建表时没有定义表的完整性和一致性约束条件，那么用户可以利用 DM 所提供的基表修改语句来进行补充或修改。DM 系统提供基表修改语句，可对基表的结构进行全面的修改，包括修改基表名、列名、增加列、删除列、修改列类型、增加表级约束、删除表级约束、设置列缺省值、设置触发器状态等一系列修改功能。

本章描述管理表的几个方面，包括以下内容：

1. 管理表的准则；
2. 创建表；
3. 更改表；
4. 删除表；
5. 清空表；
6. 查看表信息。

10.1 管理表的准则

10.1.1 设计表

表是数据库设计过程中的基本构件，基于来自应用开发者的有关应用如何运作和所期望的数据类型，数据库管理员应与应用开发者一起工作，并认真规划每个表，具体需要做到以下几点：

1. 规范化表，估算并校正表结构，使数据冗余达到最小；
2. 为每个列选择合适的数据类型，是否允许为空等，并根据实际情况判断是否需要对列进行加密或压缩处理；
3. 建立合适的完整性约束，管理约束可查看 15 章管理完整性约束的内容；
4. 建立合适的聚集索引。每个表（列存储表，堆表除外）都含一个聚集索引，默认以 ROWID 建立，而建立合适的聚集索引，可以有效加快表的检索效率；
5. 根据实际需要，建立合适类型的表。DM 支持的表类型包括普通表、临时表、水平分区表、垂直分区表、堆表和列存储表。本章只介绍普通表和临时表，其他类型表将在其他章节中重点介绍。

10.1.2 指定表的存储空间上限

在创建表时指定 SPACE LIMIT 子句，可以对表的存储空间指定上限。DM 支持对表的存储空间指定大小，单位是 MB，即表的大小可由管理员指定，便于表的规模管理。当表的所有索引所占用的存储空间超过指定大小时，表将不能再新增数据。

10.1.3 指定表的存储位置

创建表时，在 STORAGE 子句中，可对表指定存储的表空间。如果没有指定，则该表将创建在用户的默认表空间中。

在创建表时，通过指定合适的表空间，有以下优点：

1. 提高数据库系统的性能，因为不同的数据库表可能对应不同的数据文件，可减少对相同文件的竞争；
2. 减少数据库管理的时间，数据库表分布在不同的表空间中，即使一个表空间损坏，也不影响其他表空间上数据库表的正常访问。

10.2 创建表

10.2.1 创建普通表

如果要在所属模式中创建新表，需要有 CREATE TABLE 数据库权限；而要在其他用户的模式中创建新表，则需要有 CREATE ANY TABLE 数据库权限。创建表时，应当为表指定一个表空间，否则，表将在 MAIN 创建。下面给出一个创建一个简单表的例子。

```
CREATE  TABLE  EMPLOYEE (
  EMPNO  INT             PRIMARY KEY,
  ENAME  VARCHAR(15)     NOT NULL,
  JOB        VARCHAR(10),
  MGR        INT
     CONSTRAINT EMP_FKEY REFERENCES EMPLOYEE(EMPNO),
  HIREDATE   DATE            DEFAULT (CURDATE),
  SALARY         FLOAT,
  DEPTNO     TINYINT         NOT NULL
CONSTRAINT DEPT_FKEY REFERENCES DEPT(DEPTNO))
STORAGE (
INITIAL         50,
NEXT            50,
MINEXTENTS      10,
FILLFACTOR      80,
ON              USERS);
```

在上述 CREATE TABLE 语句中，在 users 表空间上建立了 employee 表，并有几个完整性约束，其中包含定义在不同列上的一个主键和外键。约束将在 15 章中讨论。

创建表之后，可以使用 INSERT 命令插入数据或使用达梦数据导入导出工具装载数据，还可以直接使用 CREATE TABLE AS SELECT 创建一个表。

10.2.2 指定表的聚集索引

表（列存储表和堆表除外）都是使用 B+树(以下简称 B 树)索引结构管理的，每一个普通表都有一个聚集索引，数据通过聚集索引键排序，根据聚集索引键可以快速查询任何记录。

当建表语句未指定聚集索引键，DM 的默认聚集索引键是 ROWID，即记录默认以 ROWID 在页面中排序。ROWID 是 B 树为记录生成的逻辑递增序号，表上不同记录的 ROWID 是不一样的，并且最新插入的记录 ROWID 最大。很多情况下，以 ROWID 建的默认聚集索引并不能提高查询速度，因为实际情况下很少人根据 ROWID 来查找数据。

因此，DM 提供三种方式供用户指定聚集索引键：

1. CLUSTER PRIMARY KEY：指定列为聚集索引键，并同时指定为主键，称为聚簇主键；
2. CLUSTER KEY：指定列为聚集索引键，但是是非唯一的；
3. CLUSTER UNIQUE KEY：指定列为聚集索引键，并且是唯一的。

例如，创建 student 表，指定 stu_no 为聚簇主键。

```
CREATE  TABLE  STUDENT(
  STUNO      INT             CLUSTER PRIMARY KEY,
  STUNAME    VARCHAR(15)     NOT NULL,
  TEANO      INT,
  CLASSID    INT
);
```

指定聚簇索引键后，如果查询条件中含有聚簇索引键，可以定位记录在 B 树上的位置，使查询性能大大提高。然而，插入记录也需要根据聚簇索引键定位插入位置，有可能导致页面的分裂而影响插入性能。

在 dm.ini 配置文件中，可以指定配置项使表中的主键自动转化为聚簇主键，该配置项为 PK_WITH_CLUSTER。默认情况下，PK_WITH_CLUSTER 为 1，即建表时指定的主键自动转化为聚簇主键；若为 0，则主键不会自动变为聚簇主键。

10.2.3 指定表的填充因子

上文提到，每个普通表都含有一个聚集索引，指定表的填充因子，即指定聚集索引的填充因子。索引的填充因子指在新建和重组索引时，页面记录存储空间占页面总大小的百分比。而这部分预留空间是为更新字段时使用的，一个有效的填充因子可以大大减少由更新记录导致的页面拆分。

例如，上述的创建 employee 表语句中的 STORAGE 子句，指定了 FILLFACTOR 为 80，即填充因子为 80%。

当填充因子取值低，则需要更多的页来存储数据，因而读取范围大，会影响性能。而当填充因子取值高，则在更新数据时可能造成大量的页拆分，页拆分需要消耗较多 CPU 和 I/O 资源，同样会影响性能。原则上，在只读表上应该设置填充因子高，而有大量更新的表上应该设置较低的值。默认情况下，DM 新建的表和索引的填充因子是 100，可根据实际情况设置合适的填充因子大小。

10.2.4 查询建表

为了创建一个与已有表相同的新表，或者为了创建一个只包含另一个表的一些行和列的新表，可以使用 CREATE TABLE AS SELECT(CTAS)命令。使用该命令，可以通过使用 WHERE

条件将已有表中的一部分数据装载到一个新表中，或者可以通过 SELECT * FROM 子句将已有表的所有数据装载到创建的表中，如以下程序片段所示：

```
CREATE TABLE NEW_EMP
AS
SELECT * FROM EMPLOYEE;
```

如果用户通过单表的全表查询进行建表操作，则可以通过将 INI 参数 CTAB_SEL_WITH_CONS 置为 1 进行原始表上约束的拷贝，列上能拷贝的约束包括默认值属性、自增属性、非空属性以及加密属性，表上能拷贝的约束包括唯一约束、PK 约束以及 CHECK 约束。

10.2.5 创建临时表

当处理复杂的查询或事务时，由于在数据写入永久表之前需要暂时存储一些行信息或需要保存查询的中间结果，可能需要一些表来临时存储这些数据。DM 允许创建临时表来保存会话甚至事务中的数据。在会话或事务结束时，这些表上的数据将会被自动清除。

临时表中的数据不能像在其它永久表中的数据那样进行备份，当事务结束或会话断开时，数据就会被清空。在临时表创建过程中，不会像永久表和索引那样自动分配数据段，而是仅当第一次执行 DML 语句时，才会为临时表在临时表空间中分配空间。并且，对于不同的会话，临时表上的数据是独享的，不会互相干扰，即会话 A 不能访问会话 B 临时表上的数据。

对复杂查询的传统响应方式之一是使用一个视图，使复杂查询更易于操作。但是，视图在每次访问时都需要执行，因而大大降低了性能。而通过 AS SELECT 子句建立的临时表是将复杂查询的结果通过临时 B 树记录了下来，下次访问不用重新执行查询就可以获得数据，并且会话或事务结束后数据将自动删除，是复杂查询的一个优秀的解决方案，且提高了性能。

DM 临时表支持以下功能：

1. 在临时表中，会话可以像普通永久表一样更新、插入和删除数据；
2. 临时表的 DML 操作产生较少的 REDO 日志；
3. 临时表支持建索引，以提高查询性能；
4. 在一个会话或事务结束后，数据将自动从临时表中删除；
5. 不同用户可以访问相同的临时表，每个用户只能看到自己的数据；
6. 临时表的数据量很少，意味着更高效的查询效率；
7. 临时表的表结构在数据删除后仍然存在，便于以后的使用；
8. 临时表的权限管理跟普通表一致。

临时表 ON COMMIT 关键词指定表中的数据是事务级还是或会话级的，默认情况下是事务级的。

1. ON COMMIT DELETE ROWS：指定临时表是事务级的，每次事务提交或回滚之后，表中所有数据都被删除；
2. ON COMMIT PRESERVE ROWS：指定临时表是会话级的，会话结束时才清空表，并释放临时 B 树。

下面的例子创建一个事务级的临时表：

```
CREATE GLOBAL TEMPORARY TABLE TMP_EMP(
 EMPNO  INT     PRIMARY KEY,
 ENAME  VARCHAR(15)     NOT NULL,
 JOB    VARCHAR(10))
ON COMMIT DELETE ROWS;
```

10.3 更改表

想更改的表如果在所属的模式中，用户必须具有 ALTER TABLE 数据库权限；若在其他模式中，用户必须有 ALTER ANY TABLE 的数据库权限。

通过更改表，用户可以对数据库中的表作如下修改：

1. 添加或删除列，或修改现有的列的定义（列名、数据类型、长度、默认值）。其中，对于添加列，当设置 INI 参数 ALTER_TABLE_OPT 为 1 时，添加列采用查询插入实现，可能会导致 ROWID 的改变；ALTER_TABLE_OPT 为 2 时，系统开启快速加列功能，对于没有默认值或者默认值为 NULL 的新列，系统内部会标记为附加列，能够达到瞬间加列的效果，此时 ROWID 不会改变，若有默认值且默认值不为 NULL，则仍旧采取查询插入实现。
2. 添加、修改或删除与表相关的完整性约束；
3. 重命名一个表；
4. 启动或停用与表相关的完整性约束；
5. 启动或停用与表相关的触发器；
6. 修改表的 SPACE LIMIT；
7. 增删自增列。

10.4 删除表

当一个表不再使用时，可以将其删除。删除表时，将产生以下结果：

1. 表的结构信息从数据字典中删除，表中的数据不可访问；
2. 表上的所有索引和触发器被一起清除；
3. 所有建立在该表上的同义词、视图和存储过程变为无效；
4. 所有分配给表的簇标记为空闲，可被分配给其他的数据库对象。

一般情况下，普通用户只能删除自己模式下的表。若要删除其他模式下的表，则必须具有 DROP ANY TABLE 数据库权限。

以下语句可删除 employee 表：

```
DROP TABLE employee;
```

如果要删除的表被其他表引用，即其他表的外键引用了表的任何主键或唯一键，则需要在 DROP TABLE 语句中包含 CASCADE 选项，如：

```
DROP TABLE employee CASCADE;
```

10.5 清空表

有些情况下，当表的数据不再使用时，需要删除表的所有行，即清空该表。DM7 支持以下方式来删除表中的所有的行：

1. 使用 DELETE 语句；
2. 使用 DROP 和 CREATE 语句；
3. 使用 TRUNCATE 语句。

10.5.1 使用 DELETE

使用 DELETE 语句能删除表中的行。例如，下面的语句删除 employee 表中的所有行：

```
DELETE FROM employee;
```

但是，使用 DELETE 清空表，当表有很多行时，会消耗很多系统资源。因为，DELETE 操作需要 CPU 时间，并且会产生大量的 REDO 日志和 UNDO 记录。另外，如果表上关联了元组级触发器，每删除一行，就会启动一次触发器。这都需要大量的系统资源。

10.5.2 使用 DROP 和 CREATE

使用 DROP 删除一个表，然后创建一个同名的表，也可以达到清空表的效果。例如，下面的语句先删除 employe 表，之后再重新创建。

```
DROP TABLE EMPLOYEE;
CREATE TABLE EMPLOYEE(…);
```

当删除和重新创建表时，所有与之相关联的索引、完整性约束和触发器也被删除。同样，所有针对被删除表的授权也会被删除。

10.5.3 使用 TRUNCATE

使用 TRUNCATE 语句能删除表中的所有行。例如，下面的语句清空 employee 表。

```
TRUNCATE TABLE EMPLOYEE;
```

TRUNCATE 语句为我们提供了一种快速、有效地删除表所有行的方法。并且 TRUNCATE 是一个 DDL 语句，不会产生任何回滚信息。执行 TRUNCATE 会立即提交，而且不能回滚。

TRUNCATE 语句并不影响与被删除的表相关联的任何结构、约束、触发器或者授权。 另外，DM 数据库 TRUNCATE 表后，原来分配给该表的空间会被释放，供其他数据库对象使用，大大提高空间的利用效率。

一般情况下，普通用户只能 TRUNCATE 自己模式下的表。若要 TRUNCATE 其他模式下的表，则必须具有 DROP ANY TABLE 数据库权限。

如果要清空的表被其他表引用，即其他表的外键引用了表的任何主键或唯一键，并且子表不为空或子表的外键约束未被禁用，则不能 TRUNCATE 该表。

10.6 查看表信息

10.6.1 查看表定义

创建表后，可以通过 SP_TABLEDEF 系统过程查看表的定义。

```
CALL SP_TABLEDEF('SYSDBA', 'EMPLOYEE');
```

DM 通过提供的 TABLEDEF 函数来显示当前表的定义。当表多次进行 ALTER TABLE 后，显示的表定义将是最后一次修改后的建表语句。

10.6.2 查看自增列信息

DM 支持 INT 和 BIGINT 两种数据类型的自增列，并提供以下函数查看表上自增列的当前值、种子和增量等信息：

1. IDENT_CURRENT：获得表上自增列的当前值；
2. IDENT_SEED：获得表上自增列的种子信息；
3. IDENT_INCR：获得表上自增列的增量信息。

```
CREATE TABLE IDENT_TABLE (
```

```
    C1          INT     IDENTITY(100, 100),
    C2          INT
);
SELECT IDENT_CURRENT('SYSDBA.IDENT_TABLE');
SELECT IDENT_SEED('SYSDBA.IDENT_TABLE');
SELECT IDENT_INCR('SYSDBA.IDENT_TABLE');
```

10.6.3 查看表的空间使用情况

DM 使用段、簇和页实现数据的物理组织。DM 支持查看表的空间使用情况，包括：

1. TABLE_USED_SPACE：已分配给表的页面数；
2. TABLE_USED_PAGES：表已使用的页面数。

```
CREATE TABLE SPACE_TABLE (
    C1          INT,
    C2          INT
);
SELECT TABLE_USED_SPACE('SYSDBA','SPACE_TABLE');
SELECT TABLE_USED_PAGES('SYSDBA','SPACE_TABLE');
```

第11章 管理索引

11.1 管理索引的准则

索引是与表相关的可选的结构（聚簇索引除外），它能使对应于表的 SQL 语句执行得更快，因为有索引比没有索引能更快地定位信息。DM7 索引能提供访问表的数据的更快路径，可以不用重写任何查询而使用索引，其结果与不使用索引是一样的，但速度更快。

DM7 提供了几种最常见类型的索引，对不同场景有不同的功能，它们是：

1． 聚集索引：每一个普通表有且只有一个聚集索引；
2． 唯一索引：索引数据根据索引键唯一；
3． 函数索引：包含函数/表达式的预先计算的值；
4． 位图索引：对低基数的列创建位图索引；
5． 位图连接索引：针对两个或者多个表连接的位图索引，主要用于数据仓库中；
6． 全文索引：在表的文本列上而建的索引。具体内容请参考第 19 章。

索引在逻辑上和物理上都与相关的表的数据无关，作为无关的结构，索引需要存储空间。创建或删除一个索引，不会影响基本的表、数据库应用或其他索引。当插入、更改和删除相关的表的行时，DM7 会自动管理索引。如果删除索引，所有的应用仍继续工作，但访问以前被索引了的数据时速度可能会变慢。

11.1.1 在表中插入数据后创建索引

一般情况下，在插入或装载了数据后，为表创建索引会更加有效率。如果在装载数据之前创建了一个或多个索引，那么在插入每行时 DM7 都必须更改和维护每个索引，使得插入效率降低。

11.1.2 索引正确的表和列

使用下面的准则来决定何时创建索引：

1． 如果需要经常地检索大表中的少量的行，就为查询键创建索引；
2． 为了改善多个表的连接的性能，可为连接列创建索引；
3． 主键和唯一键自动具有索引，在外键上很多情况下也创建索引；
4． 小表不需要索引。

选取表中的索引列时可以考虑以下几点：

1． 列中的值相对比较唯一 ；
2． 取值范围大，适合建立索引；
3． CLOB 和 TEXT 只能建立全文索引、BLOB 不能建立任何索引。

11.1.3 为性能而安排索引列

在 CREATE INDEX 语句中列的排序会影响查询的性能。通常，将最常用的列放在最前面。

如果查询中有多个字段组合定位，则不应为每个字段单独创建索引，而应该创建一个组

合索引。当两个或多个字段都是等值查询时，组合索引中各个列的前后关系是无关紧要的。但是如果是非等值查询时，要想有效利用组合索引，则应该按等值字段在前，非等值字段在后的原则创建组合索引，查询时只能利用一个非等值的字段。

11.1.4 限制每个表的索引的数量

一个表可以有任意数量的索引。但是，索引越多，修改表数据的开销就越大。当插入或删除行时，表上的所有索引也要被更改；更改一个列时，包含该列的所有索引也要被更改。因此，在从表中检索数据的速度和更新表的速度之间有一个折衷。例如，如果一个表主要仅用于读，则索引多就有好处；如果一个表经常被更新，则索引不宜多建。

11.1.5 估计索引大小和设置存储参数

创建索引之前先估计索引的大小能更好地促进规划和管理磁盘空间。可以用索引以及回滚段、重做日志文件的组合估计的大小来决定支持所期望的数据库所需的磁盘空间的大小。通过这些估计，就可以购买合适的硬件和做出其他正确的决定。

用单个索引估计的大小能更好地管理索引使用的磁盘空间。创建索引时，可以设置适当的存储参数，并改善使用该索引的应用的 I/O 性能。例如，假设在创建索引之前估计索引的最大大小，之后就可以在创建该索引时设置适当的存储参数，就能很少为表的数据段分配簇。并且，所有的该索引的数据都被保存在相对连续的磁盘空间扇区中，这就减少了使用该索引的磁盘 I/O 操作所需的时间。

11.1.6 为每个索引指定表空间

可以在除临时表空间、日志表空间和回滚段表空间外的其他任何表空间中创建索引，也可以在其索引的表的相同或不同的表空间中创建索引。如果表及其索引使用相同的表空间 能更方便地对数据库进行管理（如表空间或文件备份）或保证应用的可用性，因为所有有关的数据总是在一起联机。然而，将表及其索引放在不同的表空间（在不同磁盘上）产生的性能比放在相同的表空间更好，因为这样做减少了磁盘竞争。但是将表及其索引放在不同的表空间时，如果一个表上某索引所在的表空间脱机了，则涉及这张表的 SQL 语句可能由于执行计划仍旧需要使用被脱机的索引而不能成功执行。

11.2 创建索引

本节描述如何创建索引。要在用户自己的模式中创建索引，至少要满足如下条件之一：

1. 要被索引的表是在自己的模式中 ；
2. 在要被索引的表上有 CREATE INDEX 权限；
3. 具有 CREATE ANY INDEX 数据库权限。

要在其他模式中创建索引，用户必须具有 CREATE ANY INDEX 数据库权限。

11.2.1 明确地创建索引

可以用 CREATE INDEX 语句明确地创建索引。如下语句在 emp 表的 ename 列上创建一个名为 emp_ename 的索引，该索引使用表空间 users。

```
CREATE INDEX emp_ename ON emp(ename)
STORAGE (
INITIAL 50,
NEXT    50,
ON      USERS);
```

注意，上述语句为该索引明确地指定了几个存储设置和一个表空间。如果没有给索引指定存储选项，则 INITIAL 和 NEXT 等存储选项会自动使用表空间的默认存储选项。

11.2.2 创建聚集索引

DM7 中表（列存储表和堆表除外）都是使用 B+树索引结构管理的，每一个普通表都有且仅有一个聚集索引，数据都通过聚集索引键排序，根据聚集索引键可以快速查询任何记录。

当建表语句未指定聚集索引键时，DM7 的默认聚集索引键是 ROWID。若指定索引键，表中数据都会根据指定索引键排序。

建表后，DM7 也可以用创建新聚集索引的方式来重建表数据，并按新的聚集索引排序。例如，可以对 emp 表以 ename 列新建聚集索引。

```
CREATE CLUSTER INDEX clu_emp_name ON emp(ename);
```

新建聚集索引会重建这个表以及其所有索引，包括二级索引、函数索引，是一个代价非常大的操作。因此，最好在建表时就确定聚集索引键，或在表中数据比较少时新建聚集索引，而尽量不要对数据量非常大的表建立聚集索引。

创建聚集索引的约束条件：

1. 每张表中只允许有一个聚集索引，如果之前已经指定过 CLUSTER INDEX 或者指定了 CLUSTER PK，则用户新建立 CLUSTER INDEX 时系统会自动删除原先的聚集索引。但如果新建聚集索引时指定的创建方式（列，顺序）和之前的聚集索引一样，则会报错；
2. 指定 CLUSTER INDEX 操作需要重建表上的所有索引，包括 PK 索引；
3. 删除聚集索引时，缺省以 ROWID 排序，自动重建所有索引；
4. 若聚集索引是默认的 ROWID 索引，不允许删除；
5. 聚集索引不能应用到函数索引中；
6. 垂直分区表在建表后不能更改聚集索引；如果建表时没有指定聚集索引，后续也不能再指定；
7. 不能在列存储表上新建/删除聚集索引；
8. 建聚集索引语句不能含有 partition_clause 子句；
9. 在临时表上增删索引会使当前会话上临时 b 树数据丢失。

11.2.3 明确地创建唯一索引

索引可以是唯一的或非唯一的。唯一索引可以保证表上不会有两行数据在键列上具有相同的值，非唯一索引不会在键列上施加这个限制。

可用 CREATE UNIQUE INDEX 语句来创建唯一索引，如下例子创建一个唯一索引：

```
CREATE UNIQUE INDEX dept_unique_index ON dept (dname)
STORAGE (ON users);
```

用户可以在希望的列上定义 UNIQUE 完整性约束，DM7 通过自动地在唯一键上定义一个唯一索引来保证 UNIQUE 完整性约束。

11.2.4 自动创建与约束相关的唯一索引

DM7 通过在唯一键或主键上创建一个唯一索引来在表上实施 UNIQUE KEY 或 PRIMARY KEY 完整性约束。当启用约束时 DM7 自动创建该索引。如下面的语句会自动在表 emp 的 name 列上创建一个唯一索引。

```
ALTER TABLE EMP ADD CONSTRAINT PK_EMP_NAME PRIMARY KEY (NAME);
```

11.2.5 创建基于函数的索引

基于函数的索引促进了限定函数或表达式的返回值的查询，该函数或表达式的值被预先计算出来并存储在索引中。正确使用函数索引，可以带来以下好处：

1. 创建更强有力的分类，例如可以用 UPPER 和 LOWER 函数执行区分大小写的分类；
2. 预先计算出计算密集的函数的值，并在索引中将其分类。可以在索引中存储要经常访问的计算密集的函数，当需要访问值时，该值已经计算出来了。因此，极大地改善了查询的执行性能；
3. 增加了优化器执行范围扫描而不是全表扫描的情况的数量。

例如，考虑如下 WHERE 子句中的表达式

```
CREATE INDEX IDX ON EXAMPLE_TAB(COLUMN_A + COLUMN_B);
SELECT * FROM EXAMPLE_TAB WHERE COLUMN_A + COLUMN_B < 10;
```

因为该索引是建立在 column_a + column_b 之上的，所以优化器可以为该查询使用范围扫描。优化器根据该索引计算查询代价，如果代价最少，优化器就会选择该函数索引，column_a + column_b 就不会重复计算。

创建函数索引有以下约束条件：

1. 函数索引表达式可以由多列组成，不同的列不能超过 63 个；
2. 函数索引表达式里面不允许出现大字段和时间间隔类型列；
3. 不支持建立分区函数索引；
4. 函数索引表达式的长度理论值不能超过 816 个字符（包括生成后的指令和字符串）；
5. 函数索引不能为 CLUSTER 或 PRIMARY KEY 类型；
6. 表达式不支持集函数和不确定函数，不确定函数为每次执行得到的结果不确定，系统中不确定函数包括：RAND、SOUNDEX、CURDATE、CURTIME、CURRENT_DATE、CURRENT_TIME、CURRENT_TIMESTAMP、GETDATE、NOW、SYSDATE、CUR_DATABASE、DBID、EXTENT、PAGE、SESSID、UID、USER、VSIZE、SET_TABLE_OPTION、SET_INDEX_OPTION、UNLOCK_LOGIN、CHECK_LOGIN、GET_AUDIT、CFALGORITHMSENCRYPT、SF_MAC_LABEL_TO_CHAR、CFALGORITHMSDECRYPT、BFALGORITHMSENCRYPT、SF_MAC_LABEL_FROM_CHAR、BFALGORITHMSDECRYPT、SF_MAC_LABEL_CMP；
7. 快速装载不支持含有函数索引的表；
8. 当表中含有行前触发器并且该触发器会修改函数索引涉及列的值时，不能建立函数索引；
9. 若函数索引中要使用用户自定义的函数，则函数必须是指定了 DETERMINISTIC 属性的确定性函数；
10. 若函数索引中使用的确定性函数发生了变更或删除，用户需手动重建函数索引；
11. 若函数索引中使用的确定性函数内有不确定因素，会导致前后计算结果不同的情况。在查询使用函数索引时，使用数据插入函数索引时的计算结果为 KEY 值；修改时可能会导致在使用函数索引过程中出现根据聚集索引无法在函数索引中找到相应记录的情况，对此进行报错处理。

11.2.6 创建位图索引

位图索引主要针对含有大量相同值的列而创建。位图索引被广泛引用到数据仓库中，创建方式和普通索引一致，对低基数（不同的值很少）的列创建位图索引，能够有效提高基于该列的查询效率。且执行查询语句的 where 子句中带有 AND 和 OR 谓词时，效率更加明显。

如下例子创建一个位图索引：

```
CREATE  BITMAP INDEX  S1  ON  PURCHASING.VENDOR (VENDORID);
```

位图索引具有以下约束：

1. 支持普通表、堆表和水平分区表创建位图索引；
2. 不支持对大字段创建位图索引；
3. 不支持对计算表达式列创建位图索引；
4. 不支持在 UNIQUE 列和 PRIMARY KEY 上创建位图索引；
5. 不支持对存在 CLUSTER KEY 的表创建位图索引；
6. 仅支持单列或者不超过 63 个组合列上创建位图索引；
7. MPP 环境下不支持位图索引的创建；
8. 不支持快速装载建含有位图索引的表；
9. 不支持全局位图索引；
10. 包含位图索引的表不支持并发的插入、删除和更新操作。

11.2.7 创建位图连接索引

位图连接索引是一种提高通过连接实现海量数据查询效率的有效方式，主要用于数据仓库环境中。区别于上一节所说的建立在单表上的位图索引，位图连接索引是针对两个或者多个表的连接而建立的位图索引，同时保存了连接的位图结果。对于索引列中的每一个值，位图连接索引在索引表中保存了对应行的 ROWID。

如下例子创建一个位图连接索引：

```
create bitmap index SALES_CUSTOMER_NAME_IDX
on SALES.SALESORDER_HEADER(SALES.CUSTOMER.PERSONID)
from   SALES.CUSTOMER, SALES.SALESORDER_HEADER
where  SALES.CUSTOMER.CUSTOMERID = SALES.SALESORDER_HEADER.CUSTOMERID;
```

使用说明

1. 适用于常规索引的基本限制也适用于位图连接索引；
2. 用于连接的列必须是维度表中的主键或存在唯一约束；如果是复合主键，则必须使用复合主键中的所有列；
3. 当多个事务同时使用位图连接索引时，同一时间只允许更新一个表；
4. 连接索引创建时，基表只允许出现一次；
5. 不允许对存在 cluster key 的表创建位图连接索引；
6. 位图连接索引表（命名为 BMJ$_索引名）仅支持 select 操作，其他操作都不支持：如 insert、delete、update、alter、drop 和建索引等；
7. 不支持对位图连接索引所在事实表和维度表的备份还原，不支持位图连接索引表的表级备份还原；
8. 不支持位图连接索引表、位图连接索引以及虚索引的导出导入；
9. 位图连接索引及其相关表不支持快速装载；
10. 位图连接索引名称的长度限制为：事实表名的长度+索引名称长度+6<128；
11. 仅支持普通表、堆表和 HUGE 表；
12. WHERE 条件只能是列与列之间的等值连接，并且必须含有所有表；

13. 事实表上聚集索引和位图连接索引不能同时存在；
14. 不支持对含有位图连接索引的表中的数据执行 DML，如需要执行 DML,则先删除该索引；
15. 含有位图连接索引的表不支持下列 DDL 操作：删除、修改表约束，删除、修改列，更改表名。另外，含位图连接索引的堆表不支持添加列操作；
16. 不允许对含有位图连接索引的表并发操作；
17. 创建位图连接索引时，在存储参数中可指定存储位图的字节数，有效值为：1~ 128，服务器自动校正为 4 的倍数，默认值为 48。如 STORAGE(SECTION(4))，表示使用 4 个字节存储位图信息。

11.3 重建索引

当一个表经过大量的增删改操作后，表的数据在物理文件中可能存在大量碎片，从而影响访问速度。另外，当删除表的大量数据后，若不再对表执行插入操作，索引所处的段可能占用了大量并不使用的簇，从而浪费了存储空间。

可以使用重建索引来对索引的数据进行重组，使数据更加紧凑，并释放不需要的空间，从而提高访问效率和空间效率。DM7 提供的重建索引的系统函数为：

```
SP_REBUILD_INDEX(SCHEMA_NAME varchar(256), INDEX_ID int);
```

SCHEAM_NAME 为索引所在的模式名，INDEX_ID 为索引 ID。

使用说明：

1．水平分区子表，临时表和系统表上建的索引不支持重建

2．虚索引和聚集索引不支持重建

例如，需要重建索引 emp_name，假设其索引 ID 为 1547892，那么使用以下语句重建索引：

```
SP_REBUILD_INDEX('SYSDBA', 1547892);
```

11.4 删除索引

用户可能出于以下某项原因需要删除一个索引：

1． 不再需要该索引；
2． 该索引没有为针对其相关的表所发布的查询提供所期望的性能改善。例如，表可能很小，或者尽管表中有许多行但只有很少的索引项；
3． 应用没有用该索引来查询数据。

要想删除索引，则该索引必须包含在用户的模式中或用户必须具有 DROP ANY INDEX 数据库权限。索引删除之后，该索引的段的所有簇都返回给包含它的表空间，并可用于表空间中的其他对象。

如何删除索引，取决于是否是用 CREATE INDEX 语句明确地创建该索引的，是则可以用 DROP INDEX 语句删除该索引。如下面的语句删除 emp_ename 索引。

```
DROP INDEX emp_ename;
```

然而，不能直接删除与已启用的 UNIQUE KEY 键或 PRIMARY KEY 键约束相关的索引。要删除一个与约束相关的索引，必须停用或删除该约束本身。如下面的语句删除主键约束 pk_emp_name，同时删除其对应的索引。

```
ALTER TABLE emp DROP CONSTRAINT pk_emp_name;
```

除了删除普通索引，DM7 还提供删除聚集索引，只要其聚集索引是通过 CREATE CLUSTER INDEX 明确建立的。例如，下面的语句删除 emp 表的聚集索引 clu_emp_name。

```
DROP INDEX clu_emp_name;
```

删除聚集索引其实是使用 ROWID 作为索引列重建聚集索引，即跟新建聚集索引一样会重建这个表以及其所有索引。

删除表就自动删除了所有与其相关的索引。

11.5 查看索引信息

创建索引后，可以通过 INDEXDEF 系统函数查看索引的定义。

```
INDEXDEF(INDEX_ID int, PREFLAG int);
```

INDEX_ID 为索引 ID，PREFLAG 表示返回信息中是否增加模式名前缀。例如，需要查看索引 emp_name 的定义，假设其索引 ID 为 1547892，那么使用以下语句查看索引定义。

```
SELECT INDEXDEF(1547892, 0);
或 SELECT INDEXDEF(1547892, 1);
```

第12章 管理触发器

DM 是一个具有主动特征的数据库管理系统，其主动特征包括约束机制和触发器机制。约束机制主要用于对某些列进行有效性和完整性验证；触发器（TRIGGER）定义当某些与数据库有关的事件发生时，数据库应该采取的操作。通过触发器机制，用户可以定义、删除和修改触发器。DM 自动管理和运行这些触发器，从而体现系统的主动性，方便用户使用。

触发器是一种特殊的存储过程，它在创建后就存储在数据库中。触发器的特殊性在于它是建立在某个具体的表或视图之上的，或者是建立在各种事件前后的，而且是自动激发执行的，如果用户在这个表上执行了某个 DML 操作（INSERT、DELETE、UPDATE），触发器就被激发执行。

触发器常用于自动完成一些数据库的维护工作。例如，触发器可以具有以下功能：

1． 可以对表自动进行复杂的安全性、完整性检查；
2． 可以在对表进行 DML 操作之前或者之后进行其它处理；
3． 进行审计，可以对表上的操作进行跟踪；
4． 实现不同节点间数据库的同步更新。

触发器与存储模块类似，都是在服务器上保存并执行的一段 DMSQL 程序语句。不同的是，存储模块必须被显式地调用执行，而触发器是在相关的事件发生时由服务器自动隐式地激发。触发器是激发它们的语句的一个组成部分，即直到一个语句激发的所有触发器执行完成之后该语句才结束，而其中任何一个触发器执行的失败都将导致该语句的失败，触发器所做的任何工作都属于激发该触发器的语句。

触发器为用户提供了一种自己扩展数据库功能的方法。可以使用触发器来扩充引用完整性，实施附加的安全性或增强可用的审计选项。关于触发器应用的例子有：

1． 利用触发器实现表约束机制（如：PRIMARY KEY、FOREIGN KEY、CHECK 等）无法实现的复杂的引用完整性；
2． 利用触发器实现复杂的事务规则（如：想确保薪水增加量不超过 25%）；
3． 利用触发器维护复杂的缺省值（如：条件缺省）；
4． 利用触发器实现复杂的审计功能；
5． 利用触发器防止非法的操作。

触发器是应用程序分割技术的一个基本组成部分，它将事务规则从应用程序的代码中移到数据库中，从而可确保加强这些事务规则并提高它们的性能。

DM 提供了四种类型的触发器：

1． 表级触发器：基于表中的数据进行触发；
2． 事件触发器：基于特定系统事件进行触发；
3． 时间触发器：基于时间而进行触发。

12.1 触发器的使用

触发器是依附于某个具体的表或视图的特殊存储过程，它在某个 DML 操作的激发下自动执行。在创建触发器时应该仔细考虑它的相关信息。具体来说，应该考虑以下几个方面的问题：

1． 触发器应该建立在哪个表/视图之上；
2． 触发器应该对什么样的 DML 操作进行响应；
3． 触发器在指定的 DML 操作之前激发还是在之后激发；
4． 对每次 DML 响应一次，还是对受 DML 操作影响的每一行数据都响应一次。

在确定了触发器的实现细节后，现在就可以创建触发器了，创建触发器的语法格式为：

```
CREATE [OR REPLACE] TRIGGER 触发器名[WITH  ENCRYPTION]
    BEFORE|AFTER|INSTEAD OF
    DELETE|INSERT|UPDATE [OF 列名]
    ON 表名
    [FOR EACH ROW [WHEN 条件]]
    BEGIN
        DMSQL 程序语句
    END;
```

用户如果要在自己的模式中创建触发器，需要具有 CREATE TRIGGER 数据库权限。如果希望能够在其它用户的模式中创建触发器，需要具有 CREATE ANY TRIGGER 数据库权限。

在创建触发器的语法结构中，用方括号限定的部分是可选的，可以根据需要选用。创建触发器的命令是 CREATE TRIGGER，根据指定的名字创建一个触发器。OR REPLACE 子句的作用是如果已经存在同名的触发器，则删除它，并重新创建。

触发器可以是前激发的（BEOFRE），也可以是后激发的（AFTER）。如果是前激发的，则触发器在 DML 语句执行之前激发。如果是后激发的，则触发器在 DML 语句执行之后激发。用 BEFORE 关键字创建的触发器是前激发的，用 AFTER 关键字创建的触发器是后激发的，这两个关键字只能使用其一。INSTEAD OF 子句仅用于视图上的触发器，表示用触发器体内定义的操作代替原操作。

触发器可以被任何 DML 命令激发，包括 INSERT、DELETE、UPDATE。如果希望其中的一种、两种或者三种命令能够激发该触发器，则可以指定它们之间的任意组合，两种不同的命令之间用 OR 分开。如果指定了 UPDATE 命令，还可以进一步指定当表中的哪个列受到 UPDATE 命令的影响时激发该触发器。

FOR EACH ROW 子句的作用是指定创建的触发器为元组级触发器。如果没有这样的子句，则创建触发器为语句级触发器。INSTEAD OF 触发器固定为元组级触发器。

由关键字 BEGIN 和 END 限定的部分是触发器的代码，也就是触发器被激发时所执行的代码。代码的编写方法与普通的 DMSQL 语句块的编写方法相同。

在触发器中可以定义变量，但必须以 DECLARE 开头。触发器也可以进行异常处理，如果发生异常，就执行相应的异常处理程序。

例如，下面创建的触发器是为了监视用户对表 emp 中的数据所进行的删除操作。如果有这样的访问，则打印相应的信息。

```
CREATE OR REPLACE TRIGGER DEL_TRG
BEFORE DELETE
ON emp
BEGIN
    PRINT '您正在对表 emp 进行删除操作';
END;
```

如果在表 EMP 上进行 DELETE 操作，则激发这个触发器，例如：

```
SQL> DELETE FROM EMP;
```

您正在对表 EMP 进行删除操作

从触发器的执行情况可以看出，无论用户通过 DELETE 命令删除 0 行、1 行或者多行数据，这个触发器只对每次 DELETE 操作激发一次，所以这是一个典型的语句级触发器。

如果一个触发器不再使用，那么可以删除它。删除触发器的语法为：

```
DROP TRIGGER 触发器名;
```

例如，要删除刚才创建的触发器 DEL_TRG，使用语句为：

```
SQL> DROP TRIGGER DEL_TRG;
```

触发器的创建者和数据库管理员可以使触发器失效。触发器失效后将暂时不起作用，直到再次使它有效，使触发器失效的命令格式为：

```
ALTER TRIGGER 触发器名    DISABLE;
```

触发器失效后只是暂时不起作用，它仍然存在于数据库中，使用命令可以使它再次起作用，使触发器再次有效的命令格式为：

```
ALTER TRIGGER 触发器 ENABLE;
```

例如，下面的两条命令先使触发器 DEL_TRG 失效，然后使其再次有效：

```
SQL> ALTER TRIGGER DEL_TRG DISABLE;
SQL> ALTER TRIGGER DEL_TRG ENABLE;
```

12.2 表级触发器

表级触发器都是基于表中数据的触发器，它通过针对相应表对象的插入/删除/修改等 DML 语句触发。

■ 触发动作

激发表级触发器的触发动作是三种数据操作命令，即 INSERT、DELETE 和 UPDATE 操作。在触发器定义语句中用关键字 INSERT、DELETE 和 UPDATE 指明构成一个触发器事件的数据操作的类型，其中 UPDATE 触发器会依赖于所修改的列，在定义中可通过 UPDATE OF <触发列清单>的形式来指定所修改的列，<触发列清单>指定的字段数不能超过 128 个。

■ 触发级别

根据触发器的级别可分为元组级 (也称行级)和语句级。

元组级触发器，对触发命令所影响的每一条记录都激发一次。假如一个 DELETE 命令从表中删除了 1000 行记录，那么这个表上的元组级 DELETE 触发器将被执行 1000 次。元组级触发器常用于数据审计、完整性检查等应用中。元组级触发器是在触发器定义语句中通过 FOR EACH ROW 子句创建的。对于元组级触发器，可以用一个 WHEN 子句来限制针对当前记录是否执行该触发器。WHEN 子句包含一条布尔表达式，当它的值为 TRUE 时，执行触发器；否则，跳过该触发器。

语句级触发器,对每个触发命令执行一次。例如,对于一条将 500 行记录插入表 TABLE_1 中的 INSERT 语句，这个表上的语句级 INSERT 触发器只执行一次。语句级触发器一般用于对表上执行的操作类型引入附加的安全措施。语句级触发器是在触发器定义语句中通过 FOR EACH STATEMENT 子句创建的，该子句可缺省。

■ 触发时机

触发时机通过两种方式指定。一是通过指定 BEFORE 或 AFTER 关键字，选择在触发动作之前或之后运行触发器；二是通过指定 INSTEAD OF 关键字，选择在动作触发的时候，替换原始操作，INSTEAD OF 允许建立在视图上，并且只支持行级触发。

在元组级触发器中可以引用当前修改的记录在修改前后的值，修改前的值称为旧值，修改后的值称为新值。对于插入操作不存在旧值，而对于删除操作则不存在新值。

对于新、旧值的访问请求常常决定一个触发器是 BEFORE 类型还是 AFTER 类型。如果需要通过触发器对插入的行设置列值，那么为了能设置新值，需要使用一个 BEFORE 触发器，因为在 AFTER 触发器中不允许用户设置已插入的值。在审计应用中则经常使用 AFTER 触发器，因为元组修改成功后才有必要运行触发器，而成功地完成修改意味着成功地通过了该表的引用完整性约束。

12.3 事件触发器

前面表级触发器都是基于表中数据的触发器，它通过针对相应表对象的插入/删除/修改等 DML 语句触发。DM 还支持事件触发器，包括库级和模式级触发器，这类触发器并不依赖于某个表，而是基于特定系统事件触发的，通过指定 DATABASE 或某个 SCHEMA 来表示事件

触发器的作用区域。创建事件触发器的用户需要拥有 CREATE_TRIGGER 或 CREATE_ANY_TRIGGER 的权限。

可以触发的事件包含以下两类：

1. DDL 事件，包括 CREATE、ALTER、DROP、GRANT、REVOKE 以及 TRUNCATE；
2. 系统事件，包括 LOGIN/LOGON、LOGOUT/LOGOFF、AUDIT、NOAUDIT、BACKUP DATABASE、RESTORE DATABASE、TIMER、STARTUP、SHUTDOWN 以及 SERERR（即执行错误事件）。

所有 DDL 事件触发器都可以设置 BEFORE 或 AFTER 的触发时机，但系统事件中 LOGOUT，SHUTDOWN 仅能设置为 BEFORE，而其它则只能设置为 AFTER。模式级触发器不能是 LOGIN/LOGON、LOGOUT/LOGOFF、SERERR、BACKUP DATABASE、RESTORE DATABASE、STARTUP 和 SHUTDOWN 事件触发器。

与数据触发器不同，事件触发器不能影响对应触发事件的执行。它的主要作用是帮助管理员监控系统运行发生的各类事件，进行一定程度的审计和监视工作。

12.4 时间触发器

从 DM7 开始，触发器模块中新增了一种特殊的事件触发器类型，就是时间触发器，时间触发器的特点是用户可以定义在任何时间点、时间区域、每隔多长时间等等的方式来激发触发器，而不是通过数据库中的某些操作包括 DML、DDL 操作等来激发，它的最小时间精度为分钟。

时间触发器与其它触发器的不同只是在触发事件上，在 DMSQL 语句块（BEGIN 和 END 之间的语句）的定义是完全相同的。

时间触发器的创建语句如下：

```
CREATE [OR REPLACE] TRIGGER 触发器名 WITH ENCRYPTION
    AFTER TIMER ON DATABASE
    {时间定义语句}
    BEGIN
        执行语句
    END;
```

时间定义语句的语法规则较多，具体可参考《DM7_SQL 语言使用手册》中时间触发器相关章节。

下面的例子在每个月的第 28 天，从早上 9 点开始到晚上 18 点之间，每隔一分钟就打印一个字符串“Hello World”。

```
CREATE OR REPLACE TRIGGER timer2
    AFTER TIMER ON DATABASE
    FOR EACH 1 MONTH DAY 28
    FROM TIME '09:00' TO TIME '18:00' FOR EACH 1 MINUTE
    DECLARE
    str VARCHAR;
    BEGIN
        PRINT 'HELLO WORLD';
    END;
    /
```

时间触发器实用性很强，如在凌晨（此时服务器的负荷比较轻）做一些数据的备份操作，对数据库中表的统计信息的更新操作等类似的事情。同时也可以作为定时器通知一些用户在未来的某些时间要做哪些事情。

12.5 触发器总结

表级触发器的触发事件包括某个基表上的 INSERT、DELETE 和 UPDATE 操作，无论对于哪种操作，都能够为其创建 BEFORE 触发器和 AFTER 触发器。如果触发器的动作代码不取决于受影响的数据，语句级触发器就非常有用。例如，可以在表上创建一个 BEFORE INSERT 语句触发器，以防止在某些特定期限以外的时间对一个表进行插入。

每张基表上可创建的触发器的个数没有限制，但是触发器的个数越多，处理 DML 语句所需的时间就越长，这是显而易见的。创建触发器的用户必须是基表的创建者，或者拥有 DBA 权限。注意，不存在触发器的执行权限，因为用户不能主动"调用"某个触发器，是否激发一个触发器是由系统来决定的。

对于语句级和元组级的触发器来说，都是在 DML 语句运行时激发的。在执行 DML 语句的过程中，基表上所创建的触发器按照下面的次序依次执行：

1． 如果有语句级前触发器的话，先运行该触发器；

2． 对于受语句影响每一行：

1） 如果有行级前触发器的话，运行该触发器；

2） 执行该语句本身；

3） 如果有行级后触发器的话，运行该触发器；

3． 如果有语句级后触发器的话，运行该触发器。

需要注意的是，同类触发器的激发顺序没有明确的定义。如果顺序非常重要的话，应该把所有的操作组合在一个触发器中。触发器功能强大，但需要谨慎使用，过多的触发器或复杂触发器过程脚本会降低数据库的运行效率。

还需要注意的是，在 DM7 的数据守护环境下，备库上定义的触发器是不会被触发的。

第13章 管理视图、序列和同义词

视图、序列和同义词是数据库中比较重要的模式对象概念，达梦数据库支持这些模式对象的管理。

13.1 管理视图

视图是关系数据库系统提供给用户以多种角度观察数据库中数据的重要机制，它简化了用户数据模型，提供了逻辑数据独立性，实现了数据共享和数据的安全保密。视图是数据库技术中一个十分重要的功能。从系统实现的角度讲，视图是从一个或几个基表（或视图）导出的表，但它是一个虚表，即数据字典中只存放视图的定义（由视图名和查询语句组成），而不存放对应的数据，这些数据仍存放在原来的基表中。当对一个视图进行查询时，视图将查询其对应的基表，并且将所查询的结果以视图所规定的格式和次序进行返回。因此当基表中的数据发生变化时，从视图中查询出的数据也随之改变了。从用户的角度来讲，视图就像一个窗口，透过它可以看到数据库中用户感兴趣的数据和变化。当用户所需的数据是一张表的部分列、或部分行，或者数据是分散在多个表中，那么就可以创建视图来将这些满足条件的行和列组织到一个表，而不需要修改表的属性、甚至创建新的表。这样不仅简化了用户的操作，还可以提高数据的逻辑独立性，实现数据的共享和保密。

例如，为了防止非公司人员查询供应商的具体信息，包括 email、phone 等，这时可以建立视图，针对不用的用户建立查询供应商信息的视图。

```
CREATE VIEW normal_view AS SELECT name FROM person;
CREATE VIEW special_view AS SELECT name, sex, email, phone FROM person;
```

然后把视图 normal_view 的权限授予普通用户，只允许他们查询供应商的姓名信息，把 special_view 视图的权限授予高级用户，允许他们查看供应商的 email、电话等信息，这样就较好地实现了供应商信息的安全与保密。

关于视图的定义、删除、查询和修改可以参考《DM7_SQL 语言使用手册》。

用户也可以对那些经常进行的查询建立相应视图，如果经常使用的查询语句较庞大，这样可以节省繁琐语句的书写。

由于视图没有直接相关的物理数据，所以不能被索引。

DM 提供了一些以"V$"开头的视图来供用户了解当前服务器的使用情况。这些视图的数据在服务器运行期间一直变化，并且数据反映的主要是系统的性能信息，因此它们被称为动态性能视图。用户对动态性能视图只能进行查询操作，动态视图的具体内容可参考附录 2。

13.2 管理序列

序列（sequence）是 DM 数据库中的数据库实体之一。通过使用序列，多个用户可以产生和使用一组不重复的有序整数值。比如可以用序列来自动地生成主关键字值。序列通过提供唯一数值的顺序表来简化程序设计工作。

当一个序列第一次被查询调用时，它将返回一个预定值，该预定值就是在创建序列时所指定的初始值。默认情况下，对于升序序列，序列的缺省初始值为序列的最小值，对于降序序列，缺省初始值为序列的最大值。可以指定序列能生成的最大值，默认情况下，降序序列的最大值缺省为-1，升序序列的最大值为 $2^{63}-1$；也可以指定序列能生成的最小值，默认情况下，升序序列的最小值缺省为 1，降序序列的最小值为-2^{63}。序列的最大值和最小值可以指定为 longint（4 个字节）所能表示的最大和最小有符号整数。

在随后的每一次查询中，序列将产生一个按其指定的增量增长的值。增量可以是任意的正整数或负整数，但不能为 0。如果此值为负，序列是下降的，如果此值为正，序列是上升的。默认情况下，增加缺省为 1。

一旦序列生成，用户就可以在 SQL 语句中用以下伪列来存取序列的值：

1. CURRVAL 返回当前的序列值；
2. NEXTVAL 如果为升序序列，序列值增加并返回增加后的值；如果为降序序列，序列值减少并返回减少后的值。

序列可以是循环的，当序列的值达到最大值/最小值时，序列将从最小值/最大值计数。使用一个序列时，不保证将生成一串连续不断递增的值。例如，如果查询一个序列的下一个值供 insert 使用，则该查询是能使用这个序列值的唯一会话。如果未能提交事务处理，则序列值就不被插入表中，以后的 insert 将继续使用该序列随后的值。

序列在对编号的使用上具有很大用处，如果想对表建立一个列专门用来表示编号，如订单号，这样就可以使用序列，依次递增生成，用户不需进行特殊管理，这给用户带来了很大方便。如果用户需要间隔的编号，创建序列时指定 INCREMENT，就可以生成用户需要的编号。关于序列的具体使用，请参考《DM7_SQL 语言使用手册》。

13.3 管理同义词

同义词相当于模式对象的别名，起着连接数据库模式对象和应用程序的作用。假如模式对象需要更换或者修改，则不用修改应用程序而直接修改同义词就可以了。

同义词是用来实现下列用途的数据库对象：

1. 为可以存在于本地或远程服务器上的其他数据库对象（称为基础对象）提供备用名称；
2. 提供抽象层以免客户端应用程序对基础对象的名称或位置进行更改。

同义词的好处在于用户可能需要某些对象在不同的场合采用不用的名字，使其适合不同人群的应用环境。例如，创建表 product，如果客户不认识这个英文词，这时可以增加同义词，命名“产品”，这样客户就有较直观的观念，一目了然。

关于同义词的具体使用，请参考《DM7_SQL 语言使用手册》。

13.4 查看视图、序列和同义词信息

视图、序列以及同义词的定义信息可以通过查看系统表 SYSOBJECTS 和 SYSTEXTS 得到，如查看普通视图 view_1 的信息，可以执行：

```
    SELECT b.* FROM  SYS.SYSOBJECTS a, SYS.SYSTEXTS b WHERE a.ID = b.ID and a.NAME
LIKE 'VIEW_1%';
```

可以得到：

```
    ID            SEQNO          TXT
    1        16778216         0          CREATE VIEW VIEW_1 AS SELECT ID FROM
SYSOBJECTS;
    2      16778216       1        SELECT ID FROM SYSOBJECTS
```

这样可以很明确知道 VIEW_1 视图的定义。

视图的定义信息也可以通过 SP_VIEWDEF 系统过程来查看。对于物化视图而言，其定义只有通过该系统函数才能完整地获取。

```
    CALL  SP_VIEWDEF('SYSDBA', 'VIEW1');
```

第14章 模式对象的常规管理

用户的模式(SCHEMA）指的是用户账号拥有的对象集，在概念上可将其看作是包含表、视图、索引和权限定义的对象。在 DM 中，一个用户可以创建多个模式，一个模式中的对象（表、视图等）可以被多个用户使用。模式不是严格分离的，一个用户可以访问他所连接的数据库中有权限访问的任意模式中的对象。

系统为每一个用户自动建立了一个与用户名同名的模式作为其默认模式，用户还可以用模式定义语句建立其它模式。

采用模式的原因有几点：

1． 允许多个用户使用一个数据库而不会干扰其它用户；
2． 把数据库对象组织成逻辑组，让它们更便于管理；
3． 第三方的应用可以放在不同的模式中，这样可以避免和其它对象的名字冲突。模式类似于操作系统层次的目录，只不过模式不能嵌套。

DM7 模式可以通过 SQL 语句进行操作。DM7 模式主要包含以下的模式对象：

1． 表；
2． 视图；
3． 索引；
4． 触发器；
5． 存储过程/函数；
6． 序列；
7． 全文索引；
8． 包；
9． 同义词；
10．类；
11．外部链接。

在引用模式对象的时候，一般要在模式对象名前面加上模式名。具体格式如下：

[模式名].对象名

当然，在当前模式和要引用的模式对象所属的模式相同时，可以省略模式名。如果我们访问一个表时，没有指明该表属于哪一个模式，系统就会自动给我们在表前加上缺省的模式名。类似地，如果我们在创建对象时不指定该对象的模式，则该对象的模式为用户的缺省模式。

模式对象之外的其他对象统一称为非模式对象，非模式对象主要包括以下几种对象：

1． 用户；
2． 角色；
3． 权限；
4． 表空间。

14.1 在单个操作中创建多个模式对象

在 DM7 中，使用 CREATE SCHEMA 语句就可以创建一个空的模式对象，也可以在创建模式时，同时创建多个模式对象。CREATE SCHEMA 语句的语法说明参见《DM7_SQL 语言使用手册》。

下面的语句在创建模式 TEST 的同时，创建了属于模式 TEST 的一个序列和两张表。

```
CREATE SCHEMA TEST
```

```
CREATE SEQUENCE ADDRESS_SEQ INCREMENT BY 1
CREATE TABLE ADDRESS(
     ADDRESSID INT,
         ADDRESS1 VARCHAR(60) NOT NULL,
         ADDRESS2 VARCHAR(60),
         CITY VARCHAR(30) NOT NULL,
         POSTALCODE VARCHAR(15) NOT NULL)
CREATE TABLE ADDRESS_TYPE(
     ADDRESS_TYPEID INT PRIMARY KEY,
         NAME VARCHAR(50) NOT NULL);
```

可以用下面的格式来引用 TEST 模式中的 ADDRESS 表：TEST.ADDRESS，如向 ADDRESS 表中插入一条数据。

```
INSERT INTO TEST.ADDRESS VALUES(1,'武汉市关山一路特 1 号光谷软件园 C6 栋 5 层','上海市闸北区江场三路 28 号 301 室','上海市','200436');
```

14.2 重命名模式对象

要重命名一个模式对象，那么这个模式对象必须在指定的模式里面。用户可以采用下面几种方式来重命名模式对象：

1. 删除原有的模式对象，重新创建；
2. 使用 ALTER … RENAME 语句（TABLE）。

如果采用的是删除并重建模式对象的方式来重命名模式对象，那么所有的基于这个模式对象的授权都将失效。在模式对象重建之后，基于原对象的权限和角色必须重新授予。

如果使用的是 ALTER … RENAME 语句来重命名模式对象，那么所有基于原模式对象的权限将转移到重命名后的模式对象上，不需要重新授权。如重命名 TEST 上的 ADDRESS 表：

```
ALTER TABLE TEST.ADDRESS RENAME TO ADDRESS1;
```

如果一个用户 USER1 在 ADDRESS 上拥有查询、插入的权限，那么重命名之后，USER1 在 ADDRESS1 上同样拥有查询、插入权限。

在重命名一个模式对象之前，需要考虑以下影响：

1. 所有基于重命名模式对象的视图和 DMSQL 语句块都将失效，在使用之前，必须重新编译；
2. 所有基于重命名模式对象的同义词在使用的时候都会报错。

14.3 启用和停用触发器

触发器（TRIGGER）定义当某些与数据库有关的事件发生时，数据库应该采取的操作。触发器是在相关的事件发生时由服务器自动地隐式激发的。触发器是激发它们的语句的一个组成部分，即直到一个语句激发的所有触发器执行完成之后该语句才结束，而其中任何一个触发器执行的失败都将导致该语句的失败，触发器所做的任何工作都属于激发该触发器的语句。

一个触发器有两种不同的状态：

1. 启用（ENABLED）：处于开启状态的触发器在触发条件满足时，执行触发体。缺省状态下，新创建的触发器都处于开启状态；
2. 禁止（DISABLED）：处于禁止状态的触发器在触发条件满足时，也不会执行触发体。

要设置触发器的状态，可以通过 ALTER TRIGGER 语句来完成。用户必须满足以下条件：

1） 拥有该触发器；

2） 有 ALTER ANY TRIGGER 权限。

用户可以通过使用带有 ENABLED/DISABLED 选项的 ALTER TRIGGER 语句来启用/禁止触发器。如启用 addr_trig 触发器：

```
ALTER TRIGGER addr_trig ENABLED;
```

如禁用 addr_trig 触发器：

```
ALTER TRIGGER addr_trig DISABLED;
```

14.4 管理完整性约束

完整性约束规则，限制表中一个或者多个列的值。约束子句可以出现在 CREATE TABLE 或者 ALTER TABLE 语句中，确定约束条件并指定受到约束的列。

14.4.1 完整性约束状态

用户可以指定一个约束是启用（ENABLE）或禁止（DISABLED）。如果启用一个约束，那么在插入数据或者是更新数据时会对数据进行检查，不符合约束的数据被阻止进入。如果约束是禁止（DISABLED）的，不符合约束的数据被允许进入数据库。

1. 禁止约束

要执行完整性约束定义的规则，约束应当设置为开启状态。但是，在下面的情况下，从性能的角度考虑，可以暂时将完整性约束禁用。

1） 导入大量的数据到一张表中；

2） 做批处理操作并对一张表做大规模修改时；

3） 导入导出一张表。

在上面三种情形下，暂时禁用完整性约束能提高系统的性能，特别是在数据仓库的情况下。在禁用约束的情况下，可以将违反约束规则的数据插入到表中，因此，用户应当在上面列表中的操作结束之后启用约束。

2. 启用约束

在启用完整性约束的情况下，不满足约束规则的行是不能插入到表中的。

14.4.2 定义完整性约束

下面的 CREATE TABLE 和 ALTER TABLE 语句在定义的时候就启用完整性约束：

```
CREATE TABLE t_con (
id NUMBER(5) CONSTRAINT t_con_pk PRIMARY KEY);
ALTER TABLE t_con
ADD CONSTRAINT t_con_pk PRIMARY KEY (id);
```

使用 ALTER TABLE 语句来启用完整性约束，可能会失败，这是因为表中已存在的数据可能违反完整性约束条件。

当启用 UNIQUE 或者 PRIMARY KEY 约束时，系统就会创建一个对应的索引。

14.4.3 修改或删除现有的完整性约束

用户可以使用 ALTER TABLE 语句来启用、禁止、删除一个约束。当用户使用 UNIQUE

或者 PRIMARY KEY 约束时，系统会创建对应的索引；当这个约束被删除或者是被禁用，索引就会被删除。

1. 禁用已被启用的约束

下面的 ALTER TABLE 语句可以禁止完整性约束：

```
ALTER TABLE t_con DISABLE CONSTRAINT t_con_pk;
```

当有外键引用 UNIQUE 或者 PRIMARY KEY 列时，用户不能禁用 UNIQUE 或者 PRIMARY KEY 约束。

2. 删除约束

用户可以通过 ALTER TABLE 语句和 DROP CONSTRAINT 参数来删除完整性约束。如：

```
ALTER TABLE t_con DROP CONSTRAINT t_con_pk;
```

如果有外键引用约束(UNIQUE 或者 PRIMARY KEY)时，在删除约束时必须加上 CASCADE 参数，否则不能删除。

14.4.4 查看约束信息

用户可以在系统表 SYSOBJECTS 和 SYSCONS 中查询约束的信息。例如，在 SYSOBJECTS 系统表中查找约束名为 t_con_pk 的信息。

```
SELECT * FROM SYSOBJECTS WHERE NAME='T_CON_PK';
```

查找所有约束的信息：

```
SELECT * FROM SYSOBJECTS WHERE TYPE$='CONS';
```

系统表 SYSOBJECTS 和 SYSCONS 的详细描述详见附录数据字典部分。

14.5 管理对象依赖性

在实际应用中，许多模式对象需要引用其他的对象。例如，一个视图中包含一个查询，其中引用了其他的表或视图。一个模式对象引用了别的对象被称为依赖对象，被引用的对象称为被依赖对象。

达梦数据库在对象被调用的时候自动重新编译，确保对象有效。

14.6 管理对象名称解析

SQL 语句中的对象名字是由两部分组成，之间用点号隔开。下面描述的是达梦数据库管理系统怎样解析对象名字。

1. 如果只有一个名字，而没有点号。
   在当前模式下寻找是否存在相同名字的对象，如果找到，则返回；否则报错；
2. 如果有点号，首先检测对象名的第一部分，如在 TEST.ADDRESS 中，TEST 就是第一部分。
   1) 寻找哪一个模式的名字和第一部分相同，如果找到，以此模式进行步骤 2)；否则，以当前模式进行步骤 2)；
   2) 在模式中需找是否有与对象名第二部分同名的对象，如果找到且待解析对象名只有两个部分，则返回；如果找到但待解析对象名多于两个部分，则转步骤 3)；否则报错；
   3) 在模式中查找包含在前一个对象中并且和待解析对象名当前解析部分相同名字的对象，如果找到，循环步骤 3)，直至所有的部分检测结束后再返回；如

果没有找到，则直接报错。如 TEST.SCHOOL.CLASS.STUDENT，在 TEST 模式下中的 SCHOOL 对象中寻找 CLASS 的对象，然后在 CLASS 对象中寻找 STUDENT 对象。

14.7 显示有关模式对象的信息

在 DM 数据库系统中，模式对象的信息主要记录在 SYSOBJECTS 系统表中。可以通过下面的语句来查询模式对象的相关信息。

```
SELECT * FROM SYSOBJECTS WHERE TYPE$ = 'SCHOBJ' OR TYPE$ = 'TABOBJ';
```

但是，SYSOBJECTS 系统表并不能将模式对象的所有信息存储起来。用户应该和其它系统表一起获得更加详细的信息。

如果用户想获得索引更详细的信息，可以查询 SYSINDEXES 表；如果用户想了解存储过程、函数等的定义信息，可以查询 SYSTEXTS 表；如果用户想了解更详细的约束的相关信息，可以查询 SYSCONS 表。（系统表的详细介绍参考附录 1。）

第三部分 高级数据库管理

第15章 数据库布局和存储管理

规划数据库结构时需要考虑如何管理数据库中的相关文件， 每个表空间存储什么数据，在表空间中创建几个多大的数据文件，以及数据文件存储的位置等。本章介绍了表空间的管理、数据文件的管理、重做日志文件的管理、回滚空间的管理和控制文件的管理。

15.1 管理表空间

表空间的管理操作需要 DM 服务器处于打开状态下。

15.1.1 创建表空间

创建表空间时需要指定表空间名和其拥有的数据文件列表。比如创建名为 bookshop 的表空间，并指定该空间上拥有 2 个数据文件，每个数据文件的大小为 128M。

```
CREATE TABLESPACE bookshop DATAFILE 'd:\bookshop1.dbf' SIZE 128,
'd:\bookshop2.dbf' SIZE 128;
```

理论上最多允许有 65535 个表空间，但用户允许创建的表空间 ID 取值范围为 0~32767，超过 32767 的只允许系统使用，ID 由系统自动分配，ID 不能重复使用，即使删除掉已有表空间，也无法重复使用已用 ID 号，也就是说只要创建 32768 次表空间后，用户将无法再创建表空间。

15.1.2 扩展表空间

表空间通过数据文件来扩展，表空间的大小等于构成该表空间的所有数据文件的大小之和。所以要扩展表空间可以通过添加新的数据文件或者扩展表空间中已有的数据文件完成。数据文件的添加和扩展可见下一节。

15.1.3 删除表空间

只可以删除用户创建的表空间并且只能删除未使用过的表空间。删除表空间时会删除其拥有的所有数据文件。例如删除 bookshop 表空间。

```
DROP TABLESPACE bookshop;
```

15.1.4 修改表空间名

可修改已存在的由用户创建的表空间的名称。比如可修改 bookshop 表空间名为

books。

```
    ALTER TABLESPACE bookshop RENAME TO books;
```

15.1.5 修改表空间状态

用户表空间有联机和脱机两种状态。系统表空间、回滚表空间、重做日志表空间和临时文件表空间不允许脱机。设置表空间状态为脱机状态时，如果该表空间有未提交的事务，则脱机失败报错。脱机后可对表空间的数据进行备份。例如修改 bookshop 表空间状态为脱机。

```
    ALTER TABLESPACE bookshop OFFLINE;
```

修改 bookshop 表空间状态为联机。

```
    ALTER TABLESPACE bookshop ONLINE;
```

注意：MPP 环境下，可能发现节点间的表空间不一致情况，如：EP01 为 ONLINE 状态，EP02 为 OFFLINE 状态，这个时候，无论执行 ONLINE 还是 OFFLINE 都是报错。需要用户介入，才可以解决问题。用户 LOCAL 方式登陆实例，并执行 SP_SET_SESSION_LOCAL_TYPE(1)，使得该会话可以执行 DDL 操作，再执行 ONLINE 或者 OFFLINE 即可。

15.1.6 修改表空间数据缓冲区

用户表空间可以切换使用的数据缓冲区，系统表空间、回滚表空间、重做日志表空间和临时文件表空间不允许修改数据缓冲区。可以使用的数据缓冲区有 NORMAL 和 KEEP。表空间修改成功后，并不会立即生效，而是需要服务器重启。缓冲池名 KEEP 是达梦的保留关键字，使用时必须加双引号。例如将 bookshop 表空间绑定到 KEEP 缓冲区。

```
    ALTER TABLESPACE bookshop CACHE= "KEEP";
```

15.1.7 查询表空间与数据文件对应关系

可以通过查询动态视图 V$TABLESPACE 得到系统中所有表空间的信息，通过查询动态视图 V$DATAFILE 得到系统中所有数据文件的信息。将两个动态视图以表空间 ID 为连接条件，通过以下查询可以得到表空间上对应的数据文件。

```
    SELECT ts.NAME, df.PATH FROM V$TABLESPACE AS ts, V$DATAFILE AS df WHERE ts.ID
= df.GROUP_ID;
```

15.1.8 表空间文件失效检查

LINUX 操作系统中，被进程打开的文件仍可以在 OS 系统中被删除，因此存在 DM7 数据文件可能被误删的风险。如果数据文件被删除，DM7 系统能够及时检测出来，并立刻停止对其继续使用并通知用户。

在 dm.ini 中参数 FIL_CHECK_INTERVAL 的值指定 DM7 系统检查数据文件是否仍存在的时间间隔。将其设为 0 表示不进行检查。

也可以通过系统过程 SP_FILE_SYS_CHECK()来手动的进行检查。

系统一旦检测出某个表空间内的数据文件被删除，则与该表空间所有的操作都将会失败，并报错该表空间内有数据文件被删除。

15.1.9 表空间失效文件恢复

LINUX 系统中被删除的文件，只要其句柄没有被关闭，可以在/proc/<pid>/fd 中找到其对应的文件副本。其中<pid>指打开该文件的进程 id。

利用该方法，结合 OS 命令，DM7 提供失效文件的恢复方案如下：

1、 调用系统过程 SP_TABLESPACE_PREPARE_RECOVER(tablespace_name)准备进行恢复；

2、 如果使用过程中 DM 报错表空间数据文件被删除，通过操作系统的 ps 命令找到当前 dmserver 的 PID：ps -ef|grepdmserver；

3、 使用操作系统 ls 命令查看被删除文件对应的副本：ls /proc/<PID>/fd-l，会发现被删除的文件后有(deleted)字样；

4、使用操作系统的 cp 命令将文件复制到原位置 cpbak_fildata_file_path_dir；

5、 复制成功后，调用系统过程 SP_TABLESPACE_RECOVER(ts_name)完成表空间失效文件的恢复。

注意，要保证数据文件正确修复，需要保证在 SP_TABLESPACE_PREPARE_RECOVER 后进行数据文件的复制。

15.2 管理数据文件

管理数据文件的操作需要 DM 服务器处于打开状态下。

15.2.1 添加数据文件

可以在用户表空间中添加数据文件。添加的数据文件大小最小为 4096*页大小，如页大小为 8K，则可添加的文件最小值为 4096*8k=32M。比如在 bookshop 表空间中添加大小为 64M 的数据文件。

```
ALTER TABLESPACE bookshop ADD DATAFILE 'd:\book.dbf' SIZE 64;
```

一个表空间中，数据文件和镜像文件一起不能超过 256 个。例如，如果创建表空间的时候已经指定了 1 个数据文件，那么添加数据文件的时候，最多只能添加 255 个了。

15.2.2 扩展数据文件的大小

可以扩展用户表空间中已存在的数据文件的大小。比如扩展 bookshop 表空间中数据文件 book.dbf 大小至 128M。

```
ALTER TABLESPACE bookshop RESIZE DATAFILE 'd:\book.dbf' TO 128;
```

15.2.3 指定数据文件的扩展属性

可以指定数据文件是否可以扩展，每次扩展的空间大小以及数据文件可扩展到的最大空间大小，子句的语法为：

```
AUTOEXTEND OFF|ON [NEXT <文件扩展大小>] [MAXSIZE <文件限制大小>]
```

OFF 表示文件不可扩展，ON 表示文件可扩展。文件扩展大小表示当需要扩展文件时，文件一次增大的空间大小，取值范围是 0—2048，单位是 M。文件限制大小表示文件可扩展

的最大空间大小，为 0 或者 UNLIMITED 表示无限制，单位是 M。缺省情况下，文件扩展大小是 1M，文件的最大大小是无限制的。

创建表空间时可指定文件的扩展属性。如创建表空间时指定数据文件的扩展属性为可自动扩展，每次扩展大小为 10M，最大可扩展到 100M：

```
CREATE TABLESPACE bookshop DATAFILE 'd:\book.dbf' SIZE 32 AUTOEXTEND ON NEXT 10 MAXSIZE 100;
```

在表空间中添加文件时可指定文件的扩展属性。如添加数据文件时指定扩展属性为不可自动扩展：

```
ALTER TABLESPACE bookshop ADD DATAFILE 'd:\book.dbf' SIZE 1024 AUTOEXTEND OFF;
```

可修改表空间中已存在的数据文件的扩展属性。如修改数据文件的扩展属性为可自动扩展：

```
ALTER TABLESPACE bookshop DATAFILE 'd:\book.dbf'  AUTOEXTEND ON;
```

15.2.4 修改数据文件的路径

可以修改用户表空间中已存在数据文件的路径，待修改的数据文件所在表空间必须处于脱机状态并且只可修改用户创建的表空间中文件的路径。如修改 bookshop 表空间中文件 book.dbf 的路径为 e:\ book.dbf。

```
ALTER TABLESPACE bookshop RENAME DATAFILE 'd:\book.dbf' TO 'e:\book.dbf';
```

15.3 管理重做日志文件

15.3.1 添加重做日志文件

在服务器打开状态下，可以添加新的重做日志文件。添加的数据文件大小最小为 4096*页大小，如页大小为 8K，则可添加的文件最小值为 4096*8k=32M。如添加重做大小为 128M 的重做日志文件 DAMENG03.log。

```
ALTER DATABASE ADD LOGFILE 'd:\DAMENG03.log' size 128;
```

15.3.2 扩展重做日志文件

在服务器打开状态下，可以扩展已有的重做日志文件的大小。如扩展重做日志文件 DAMENG03.log 到 256M。

```
ALTER DATABASE RESIZE LOGFILE 'd:\DAMENG03.log' to 256;
```

15.4 管理回滚空间

回滚空间的管理和用户表空间的管理基本是一样的，区别是回滚空间的空间名固定为 ROLL，不可修改。可增加和扩展回滚空间中的回滚文件，设置回滚空间的扩展属性，相关操作可参考 16.2 管理数据文件中的说明。

回滚文件的路径记录在控制文件里面，可以使用 dmctlcvt 工具在 DM 服务器关闭的状态下对控制文件进行修改。使用 dmctlcvt 工具将控制文件转换为文本文件，编辑文本文件中要修改的文件的路径后再使用 dmctlcvt 工具将文本文件转换为控制文件即可。

首先转换控制文件到文本文件：

```
dmctlcvt  c2t D:\dm.ctl D:\ctl.txt
```

编辑 ctl.txt 文本文件中 fil_path=d:\roll.dbf 为 fil_path=e:\ roll.dbf，保存文本文件。复制 d:\roll.dbf 文件为 e:\ roll.dbf。

最后转换文本文件到控制文件：

```
dmctlcvt t2c D:\ctl.txt D:\dm.ctl
```

这种修改文件路径的方法也可用于重做日志文件，系统表空间文件等路径的修改。

15.5 管理控制文件

可以在 dm.ini 中通过设置 CTL_PATH 配置参数的值来指定控制文件的路径，缺省控制文件 dm.ctl 在数据目录下。例如可以把 dm.ctl 文件复制到 D 盘下，同时修改 dm.ini 中 CTL_PATH = D:\dm.ctl。

第16章 管理分区表和分区索引

在大型的企业应用或企业级的数据库应用中，要处理的数据量通常达到 TB 级，对于这样的大型表执行全表扫描或者 DML 操作时，效率是非常低的。

为了提高数据库在大数据量读写操作和查询时的效率，达梦数据库 DM7 提供了对表和索引进行分区的技术，把表和索引等数据库对象中的数据分割成小的单位，分别存放在一个个单独的段中，用户对表的访问转化为对较小段的访问，以改善大型应用系统的性能。

DM7 提供了水平和垂直分区两种分区方式。水平分区包括范围、哈希和列表三种方式，企业可以使用合适的分区方法，如日期（范围）、区域（列表），对大量数据进行分区。由于 DM7 划分的分区是相互独立且可以存储于不同的存储介质上的，完全可满足企业高可用性、均衡 IO、降低维护成本、提高查询性能的要求。

16.1 分区的概念

分区是指将表、索引等数据库对象划分为较小的可管理片段的技术，每一个片段称为分区子表或分区索引。一个表被分区后，对表的查询操作可以局限于某个分区进行，而不是整个表，这样可以大大提高查询速度。

DM7 支持对大表进行水平或垂直划分。例如，通讯公司将用户通话记录保存在一张表中，一年这个表产生 40GB 的数据。假设要对用户的通话信息按照季度进行统计，那么这样的统计需要在全表范围内进行。如果对表按季度进行水平分区，那么每个分区的大小平均为 10GB 左右，这样在进行统计时，只需在 10GB 的范围内进行即可。然而，假设现在只需统计用户通话的某些信息，通话信息表已经进行了垂直分区，并且需要统计的信息都在同一个垂直分区中，那么只需在那个分区中进行分区扫描即可，扫描范围也可大大减少。

DM7 采用子表方式创建分区表，分区表作为分区主表，而每一个分区以一个子表实体存在，即每一个分区都是一个完整的表，一般命名为主表名_分区名。对于水平分区，子表跟主表具有相同的逻辑结构，即分区子表与分区主表有相同的列定义和约束定义。对于垂直分区表，子表上的列是主表上列的子集。在 DM7 分区表中，主表本身不存储数据，所有数据只存储在子表中，从而实现不同分区的完全独立性。水平分区子表删除后，会将子表上的数据一起删除。

由于每一个分区都以一个子表作为实体，那么不同分区可以存储于相同表空间，也可以位于不同的表空间中。将这些分区放在不同的表空间中具有以下的好处：

1． 减少所有数据都损坏的可能性，一个表空间损坏不影响其他表空间，提高可用性；

2． 恢复时间大大减少；

3． 可以将同一个表中的数据分布在不同的磁盘上，从而均衡磁盘上的 I/O 操作；

4． 提高了表的可管理性、可利用性和访问效率。

分区操作对现存的应用和运行在分区表上的标准 DML 语句来说是透明的。但是，可以通过在 DML 中使用分区子表名字来对应用进行编程，使其充分利用分区的优点。

16.2 分区的方法

达梦数据库 DM7 支持对表进行水平分区和垂直分区。对于水平分区，提供以下分区方式：

1. 范围(range)水平分区：对表中的某些列上值的范围进行分区，根据某个值的范围，决定将该数据存储在哪个分区上；
2. 哈希（hash）水平分区：通过指定分区编号来均匀分布数据的一种分区类型，通过在 I/O 设备上进行散列分区，使得这些分区大小基本一致；
3. 列表（list）水平分区：通过指定表中的某个列的离散值集，来确定应当存储在一起的数据。例如，可以对表上的 status 列的值在('A', 'H', 'O')放在一个分区，值在('B', 'I', 'P')放在另一个分区，以此类推；
4. 多级分区表：按上述三种分区方区进行任意组合，将表进行多次分区，称为多级分区表。

而垂直分区就是将表按列分拆为多个分区，每个分区子表包含较少的列。如果查询表上的某些列，并且这些列都在一个分区中，那么只需扫描这个分区即可，可以减少 I/O 量。

16.3 创建水平分区表

在创建表的语法中，使用 partition 子句指定分区方式和分区列，以及分区的名字等信息，即可创建分区表。而分区子表可以指定 storage 子句，设置子表的存储属性，如所属表空间等；如果不指定，则继承分区主表的存储特性及表的其他属性。同时，支持多级分区表。

水平分区表的 ROWID 与其主表属性一致：LIST 表的水平分区表的 ROWID 是物理的；普通表的水平分区表的 ROWID 是逻辑的，且每个子表的 ROWID 都是从 1 开始增长，但是最终返回前，ROWID 的高字节会补充上子表序号。

16.3.1 创建范围分区表

范围分区是按照某个列或几个列的值的范围来创建分区，当用户向表中写入数据时，数据库服务器将按照这些列上的值进行判断，将数据写入相应的分区中。

在创建范围分区时，首先要指定分区列，即按照哪些列进行分区，然后为每个分区指定数据范围。范围分区支持 MAXVALUE 范围值的使用，MAXVALUE 相当于一个比任何值都大的值。范围分区非常适用于数据按时间范围组织的表，不同的时间段的数据属于不同的分区。

例如，以下语句创建一个范围分区表 callinfo，用来记录用户的 2010 年的电话通讯信息，包括主叫号码、被叫号码、通话时间和时长，并且根据季度进行分区。

```
CREATE  TABLE  callinfo(
caller  CHAR(15),
callee  CHAR(15),
time        DATETIME,
duration    INT
)
PARTITION BY RANGE(time)(
```

```
    PARTITION p1 VALUES LESS THAN ('2010-04-01'),
    PARTITION p2 VALUES LESS THAN ('2010-07-01'),
    PARTITION p3 VALUES LESS THAN ('2010-10-01'),
    PARTITION p4 VALUES EQU OR LESS THAN ('2010-12-31')  --'2010-12-31'也可替换为MAXVALUE
    );
```

值得注意的是，MAXVALUE 之间无法比较大小。如下所示：

```
    create table callinfo (
    caller  CHAR(15),
    callee  CHAR(15),
    time        DATETIME,
    duration    INT
    )
    partition by range(caller, callee)
            (
            partition p1 values less than ('a', 'b'),
            partition p2 values less than (maxvalue, 'd'),
            partition p3 values less than (maxvalue,maxvalue)
                );     ----报"范围分区值非递增"错误，建分区表失败
```

在创建分区表时，首先通过"PARTITION BY RANGE"子句指定分区的类型为范围分区，然后在这个子句之后指定一个或多个列作为分区列，如 callinfo 的 time 字段。

表中的每个分区都可以通过"PARTITION"子句指定一个名称。并且每一个分区都有一个范围，通过"VALUES LESS THAN"子句可以指定上界，而它的下界是前一个分区的上界。如分区 p2 的 time 字段取值范围是['2010-04-01', '2010-07-01')。如果通过"VALUES EQU OR LESS THAN"指定上界，即该分区包含上界值，如分区 p4 的 time 字段取值范围是['2010-10-01', '2010-12-31']。另外，可以对每一个分区指定 storage 子句，不同分区可存储在不同表空间中。

如果分区表包含多个分区列，采用多列比较方式定位匹配分区。首先比较第一个分区列值，如果无法确定目标分区，则继续比较后续分区列值，直到确定目标分区为止。匹配过程参看表 17.1。

表 17.1 多分区列匹配

插入记录	分区范围值		
	(10,10,10)	(20,20,20)	(30,30,30)
(5,100,200)	满足		
(10,10,11)		满足	
(31,1,1)	不满足	不满足	不满足

当在分区表中执行 DML 操作时，实际上是在各个分区子表上透明地修改数据。当执行 SELECT 命令时，可以指定查询某个分区上的数据。例如，以下语句查询 callinfo 表中分区 p1 的数据。

```
    SELECT * FROM callinfo PARTITION (p1);
```

16.3.2 创建 LIST 分区表

范围分区是按照某个列上的数据范围进行分区的，如果某个列上的数据无法通过划分范围的方法进行分区，并且该列上的数据是相对固定的一些值，可以考虑使用 LIST 分区。一般来说，对于数字型或者日期型的数据，适合采用范围分区的方法；而对于字符型数据，取值比较固定的，则适合于采用 LIST 分区的方法。

例如，创建一个产品销售记录表 sales，记录产品的销量情况。由于产品只在几个固定的城市销售，所以可以按照销售城市对该表进行分区。

```
CREATE  TABLE  sales(
sales_id    INT,
saleman CHAR(20),
saledate    DATETIME,
city        CHAR(10)
)
PARTITION BY LIST(city)(
PARTITION p1 VALUES ('北京', '天津'),
PARTITION p2 VALUES ('上海', '南京', '杭州'),
PARTITION p3 VALUES ('武汉', '长沙'),
PARTITION p4 VALUES ('广州', '深圳')
);
```

在创建 LIST 分区时，通过“PARTITION BY LIST”子句指定对表进行 LIST 分区，然后在每个分区中分区列的取值通过 VALUES 子句指定。当用户向表插入数据时，只要分区列的数据与 VALUES 子句指定的数据之一相等，该行数据便会写入相应的分区子表中。

注意的是，LIST 分区的分区键必须唯一。

16.3.3 创建哈希分区表

在很多情况下，用户无法预测某个列上的数据变化范围，因而无法实现创建固定数量的范围分区或 LIST 分区。

在这种情况下，DM7 哈希分区提供了一种在指定数量的分区中均等地划分数据的方法，基于分区键的散列值将行映射到分区中。当用户向表中写入数据时，数据库服务器将根据一个哈希函数对数据进行计算，把数据均匀地分布在各个分区中。在哈希分区中，用户无法预测数据将被写入哪个分区中。

现在重新考虑产品销售表的例子。如果销售城市不是相对固定的，而是遍布全国各地，这时很难对表进行 LIST 分区。如果为该表进行哈希分区，可以很好地解决这个问题。

```
CREATE  TABLE  sales01(
sales_id    INT,
saleman CHAR(20),
saledate    DATETIME,
city        CHAR(10)
)
PARTITION BY HASH(city)(
PARTITION p1,
```

```
PARTITION p2,
PARTITION p3,
PARTITION p4
);
```

如果不需指定分区表名，可以通过指定哈希分区个数来建立哈希分区表。

```
CREATE  TABLE  sales02(
sales_id    INT,
saleman CHAR(20),
saledate    DATETIME,
city        CHAR(10)
)
PARTITION BY HASH(city)
PARTITIONS 4 STORE IN (ts1, ts2, ts3, ts4);
```

PARTITIONS 后的数字表示哈希分区的分区数，STORE IN 子句中指定了哈希分区依次使用的表空间。使用这种方式建立的哈希分区表分区名是匿名的，DM7 统一使用 DMHASHPART+分区号（从 0 开始）作为分区名。例如，需要查询 sales 第一个分区的数据，可执行以下语句：

```
SELECT * FROM sales02 PARTITION (dmhashpart0);
```

16.3.4 创建多级分区表

在很多情况下，经过一次分区并不能精确地对数据进分类，这时需要多级分区表。

例如，创建一个产品销售记录表 sales，记录产品的销量情况。由于产品需要按地点和销售时间进行统计，则可以对该表进行 LIST-RANGE 分区。

```
DROP TABLE SALES;
CREATE  TABLE  SALES(
SALES_ID     INT,
SALEMAN CHAR(20),
SALEDATE    DATETIME,
CITY          CHAR(10)
)
PARTITION BY LIST(CITY)
 SUBPARTITION BY RANGE(SALEDATE) SUBPARTITION TEMPLATE(
    SUBPARTITION P11 VALUES LESS THAN ('2012-04-01'),
    SUBPARTITION P12 VALUES LESS THAN ('2012-07-01'),
    SUBPARTITION P13 VALUES LESS THAN ('2012-10-01'),
    SUBPARTITION P14 VALUES EQU OR LESS THAN (MAXVALUE))
(
    PARTITION P1 VALUES ('北京', '天津')
    (
      SUBPARTITION P11_1 VALUES LESS THAN ('2012-10-01'),
      SUBPARTITION P11_2 VALUES EQU OR LESS THAN (MAXVALUE)
    ),
    PARTITION P2 VALUES ('上海', '南京', '杭州'),
```

```
        PARTITION P3 VALUES (DEFAULT)
    );
```

在创建多级分区表时，指定了子分区模板，同时子分区 P1 自定义了子分区描述 P11_1 和 P11_2。P1 有两个子分区 P11_1 和 P11_2。而子分区 P2 和 P3 有四个子分区 P11、P12、P13 和 P14。

DM 支持最多八层多级分区。

下面给出一个三级分区的例子，更多级别的分区表的建表语句语法类推。

```
    CREATE TABLE STUDENT(NAME VARCHAR(20), AGE INT, SEX VARCHAR(10) CHECK (SEX
IN ('MAIL','FEMAIL')), GRADE INT CHECK (GRADE IN (7,8,9)))
    PARTITION BY LIST(GRADE)
      SUBPARTITION BY LIST(SEX) SUBPARTITION TEMPLATE
      (
        SUBPARTITION Q1 VALUES('MAIL'),
        SUBPARTITION Q2 VALUES('FEMAIL')
      ),
      SUBPARTITION BY RANGE(AGE) SUBPARTITION TEMPLATE
      (
        SUBPARTITION R1 VALUES LESS THAN (12),
        SUBPARTITION R2 VALUES LESS THAN (15),
        SUBPARTITION R3 VALUES LESS THAN (MAXVALUE)
       )
    (
      PARTITION P1 VALUES (7),
      PARTITION P2 VALUES (8),
      PARTITION P3 VALUES (9)
    );
```

16.4 在水平分区表建立索引

DM7 支持对水平分区表建立普通索引、唯一索引、聚集索引和函数索引。创建索引时若未指定 GLOBAL 关键字则建立的索引是局部索引，即每一个表分区都有一个索引分区，并且只索引该分区上的数据。如果指定了 GLOBAL 关键字则建立的索引是全局索引，即每个表分区的数据都被索引在同一个 B 树中。目前，仅堆表的水平分区表支持 GLOBAL 全局索引。堆表上的 primary key 会自动变为全局索引。

例如，在 sales 表上的 saledate 上建立索引。

```
    CREATE INDEX ind_sales_saldate ON sales(saledate);
    CREATE UNIQUE INDEX ind_sales_city ON sales(city);
```

对于非全局索引而言，建立分区索引后，每一个分区子表都会建立一个索引分区，负责索引分区子表的数据。由于每个索引分区只负责索引本分区上的数据，其他分区上的数据无法维护。因此，当对水平分区表建立非全局唯一索引时，只能建立分区键索引，即分区键必须都包含在索引键中。只有当分区键都包含在索引键中，才能对分区主表保证索引键唯一。全局唯一索引不受此约束。

另外，不能在水平分区表上建立局部唯一函数索引。

16.5 维护水平分区表

创建水平分区表后，DM7 提供了对分区表的修改，功能包括：

1. 增加分区：建立水平分区表后，可根据实际需要新增一个分区；
2. 删除分区：建立水平分区表后，可根据实际需要删除一个分区；
3. 合并分区：将相邻的两个范围分区合并为一个分区。合并分区通过指定两个分区名进行，将相邻的两个分区的数据进行合并，构建新的大分区。只能在范围分区上进行合并分区；
4. 拆分分区：将某一个范围分区拆分为相邻的两个分区。拆分分区时指定的常量表达式值必须是原范围分区的有效范围值。只能在范围分区上进行拆分分区；
5. 交换分区：将分区数据跟普通表数据交换功能，普通表必须跟分区表同构（拥有相同的列和索引）。不支持含有加密列的分区表交换分区。

在 DM7 中，由于局部索引反映基础表的结构，因此当对表的分区和子分区进行修改操作时，会自动地对局部索引进行相应的修改。

16.5.1 增加分区

DM7 支持用 ALTER TABLE ADD PARTITION 语句将新分区增加到最后一个现存分区的后面。例如，范围分区表 callinfo 现需要记录用户的 2011 年的第一季度的通讯信息，那么，需要为 2011 年第一季度增加一个分区，并将其存储在表空间 ts5 中。

```
ALTER TABLE callinfo
ADD PARTITION p5 VALUES LESS THAN ('2011-4-1') STORAGE (ON ts5);
```

对于范围分区，增加分区必须在最后一个分区范围值的后面添加，要想在表的开始范围或中间增加分区，应使用 SPLIT PARTITION 语句。

对于 LIST 分区，增加分区包含的离散值不能已存在于某个分区中。例如，为 LIST 分区表 sales 添加一个分区管理拉萨和呼和浩特的销售情况。

```
ALTER TABLE sales
ADD PARTITION p5 VALUES ('拉萨', '呼和浩特') STORAGE (ON ts5);
```

只能对范围分区和 LIST 分区增加分区，不能对哈希分区增加分区。并且增加分区不会影响分区索引，因为分区索引只是局部索引，新增分区仅是新增分区子表，并更新分区主表的分区信息，其他分区并不发生改变。

16.5.2 删除分区

DM7 支持用 ALTER TABLE DROP PARTITION 语句将分区删除。例如，范围分区表 callinfo 现需要删除记录用户的 2011 年的第一季度的通讯信息，那么，只需删除 callinfo 的分区 p1 即可。

```
ALTER TABLE callinfo DROP PARTITION p1;
```

只能对范围分区和 LIST 分区进行删除分区，哈希分区不支持删除分区。跟增加分区一样，删除分区不会影响分区索引，因为分区索引只是局部索引，删除分区仅是删除分区子表，并更新分区主表的分区信息，其他分区并不发生改变。

16.5.3 交换分区

假设上文提到的 callinfo 表是用于维护最近 12 个月的用户通话信息，超过 12 个月的订单需要迁移到该季度的通话信息历史表中，并且每一个季度都有一个相应的历史表。如果没有使用水平分区，需要较多的删除和插入操作，并产生大量的 redo 和 undo 日志。

如果使用分区表，如上文提到的 callinfo，只需使用交换分区即可完成以上功能。例如，2011 年第二季度已到了，需删除 2010 年第二季度的通话记录，因此，可通过以下脚本来实现：

```
CREATE  TABLE  callinfo_2011Q2(
caller  CHAR(15),
callee  CHAR(15),
time        DATETIME,
duration    INT
);
--交换分区
ALTER TABLE callinfo EXCHANGE PARTITION p2 WITH TABLE callinfo_2011Q2;
--删除原分区
ALTER TABLE callinfo DROP PARTITION p2;
--新增分区，记录 2011 年第二季度通话记录
ALTER TABLE callinfo
ADD PARTITION p6 VALUES LESS THAN ('2011-7-1') STORAGE (ON ts2);
```

通过交换分区实现分区 p2 和新建表 callinfo_2011Q2 的数据交换，表 callinfo_2011Q2 将得到 2010 年第二季度的通话记录，而分区 p2 数据将被清空。交换分区采用数据字典信息交换的技术，几乎不涉及 IO 操作，因此效率非常高。

仅范围分区和 LIST 分区支持交换分区，哈希分区表不支持。并且分区交换要求分区表跟交换表具有相同的结构（相同的表类型、相同的 BRANCH 选项、相同的列结构、相同的索引、相同的分布方式），分区交换但并不会校验数据，如交换表的数据是否符合分区范围等，即不能保证分区交换后的分区上的数据符合分区范围。

16.5.4 合并分区

要想将两个范围分区的内容融合到一个分区，就要使用 ALTER TABLE MERGE PARTITION 语句。如果分区的数据很少，或相对其他分区某些分区的数据量较少，导致 I/O 不均衡，就可以考虑使用合并分区。

例如，可将 callinfo 的 2010 第 3 季度和第 4 季度合并成一个分区：

```
ALTER TABLE callinfo MERGE PARTITIONS p3, p4 into partition p3_4;
```

仅范围分区表支持合并分区，并且合并的分区必须是范围相邻的两分区。另外，合并的分区会导致数据的重组和分区索引的重建，因此，合并分区可能会比较耗时，所需时间取决于分区数据量的大小。

16.5.5 拆分分区

ALTER TABLE 语句的 SPLIT PARTITION 子句被用于将一分区中的内容重新划分成两个新的分区。当一个分区变得太大以至于要用很长时间才能完成备份、恢复或维护操作时，就应考虑做分割分区的工作，还可以用 SPLIT PARTITION 子句来重新划分 I/O 负载。

例如，将合并后的 p3_4 拆分为原两分区 p3 和 p4，分别记录 2010 年第三和第四季度的通话记录。

```
ALTER TABLE callinfo SPLIT PARTITION p3_4 AT ('2010-9-30') INTO (PARTITION p3, PARTITION p4);
```

需要注意的是，仅范围分区表支持拆分分区，并且拆分的分区值会落在低分区中。拆分分区另一个重要用途是作为新增分区的补充。通过拆分分区，可以对范围分区表的开始或中间范围添加分区。

另外，拆分分区会导致数据的重组和分区索引的重建，因此，拆分分区可能会比较耗时，所需时间取决于分区数据量的大小。

16.6 水平分区表的限制

DM7 水平分区表有如下限制条件：

1. 分区列类型必须是数值型、字符型或日期型，不支持 BLOB、CLOB、IMAGE、TEXT、LONGVARCHAR、BIT、BINARY、VARBINARY、LONGVARBINARY、时间间隔类型和用户自定义类型为分区列；
2. 范围分区和哈希分区的分区键可以多个，最多不超过 16 列；LIST 分区的分区键必须唯一；
3. 水平分区表指定主键和唯一约束时，分区键必须都包含在主键和唯一约束中；
4. 水平分区表不支持临时表；
5. 不能在水平分区表上建立自引用约束；
6. 普通环境中，水平分区表的各级分区数的总和上限是 65535；MPP 环境下，水平分区表的各级分区总数上限取决于 INI 参数 MAX_EP_SITES，上限为 2 ^(16 - $\log_2$MAX_EP_SITES)。比如：当 MAX_EP_SITES 为默认值 64 时，分区总数上限为 1024；
7. 不允许对分区子表执行任何 DDL 操作；
8. 哈希分区支持重命名、删除约束、设置触发器是否启用的修改操作；
9. 范围分区支持分区合并、拆分、增加、删除、交换、重命名、删除约束、设置触发器是否生效操作；
10. LIST 分区支持分区增加、删除、交换、重命名、删除约束、设置触发器是否生效操作；
11. LIST 分区范围值不能为 NULL；
12. LIST 分区子表范围值个数与数据页大小和相关系统表列长度相关，存在以下限制：
    1） 4K 页，单个子表最多支持 120 个范围值；
    2） 8K 页，单个子表最多支持 254 个范围值；
    3） 16K\32K 页，单个子表最多支持 270 个范围值；
13. 对范围分区增加分区值必须是递增的，即只能在最后一个分区后添加分区。LIST

分区增加分区值不能存在于其他已存在分区；

14. 当分区数仅剩一个时，不允许删除分区；
15. 仅能对相邻的范围分区进行合并，合并后的分区名可为高分区名或新分区名；
16. 拆分分区的分区值必须在原分区范围中，并且分区名不能跟已有分区名相同。拆分分区值落在低分区中；
17. 与分区进行分区交换的普通表，必须与分区表拥有相同的列及索引，但交换分区并不会对数据进行校验，即交换后的数据并不能保证数据完整性，如 CHECK 约束；分区表与普通表创建的索引顺序要求一致；
18. 水平分区表仅支持建立局部索引，不支持建立全局索引，即在分区表上建立索引，每一个表分区都有一个索引分区，并且只索引该分区上的数据，而分区主表上的索引并不索引数据；
19. 不能对水平分区表建立唯一函数索引和全文索引；
20. 不能对分区子表单独建立索引；
21. 在未指定 ENABLE ROW MOVEMENT 的分区表上执行更新分区键，不允许更新后数据发生跨分区的移动，即不能有行迁移；
22. 不能在分区语句的 STORAGE 子句中指定 BRANCH 选项；
23. 不允许引用水平分区子表作为外键约束；
24. 多级分区表最多支持八层；
25. 多级分区表支持下列修改表操作：新增分区、新增列、删除列、删除表级约束、修改表名、设置与删除列的默认值、设置列 NULL 属性、设置列可见性、设置行迁移属性、启用超长记录、with delta、新增子分区、删除子分区、修改二级分区模板信息；
26. 水平分区表支持的列修改操作除了多级分区表支持的操作外，还支持：设置触发器生效/失效、修改除分区列以外的列名、修改列属性、增加表级主键约束、删除分区、SPLITE/MERGE 分区和交换分区；
27. 水平分区表中包含大字段、自定义字段列，则定义时指定 ENABLE ROW MOVEMENT 参数无效，即不允许更新后数据发生跨分区的移动；
28. 间隔分区表的限制说明：
    1) 仅支持一级范围分区创建间隔分区；
    2) 只能有一个分区列，且分区列类型为日期或数值；
    3) 对间隔分区进行 SPLIT，只能在间隔范围内进行操作；
    4) 被 SPLIT/MERGE 的分区，其左侧分区不再进行自动创建；
    5) 不相邻的间隔的分区，不能 MERGE；
    6) 表定义不能包含 MAXVALUE 分区；
    7) 不允许新增分区；
    8) 不能删除起始间隔分区；
    9) 间隔分区表定义语句显示到起始间隔分区为止；
    10) 自动生成的间隔分区，均不包含边界值；
    11) 间隔表达式只能为常量或日期间隔函数。日期间隔函数为：NUMTOYMINTERVAL、NUMTODSINTERVAL；数值常量可以为整型、DEC 类型；
    12) MPP 下不支持间隔分区表。

16.7 创建垂直分区表

DM7 垂直分区的主要目的是，使查询得以扫描较少的数据，减少 I/O 量，从而提高查询性能。例如，某个表包含七列，通常只引用该表的前三列，那么将该表的后四列拆分到一个单独的表中将有利于提高查询性能。

与水平分区相似，定义一个垂直分区表，系统将自动为其创建一个分区主表和若干个分区子表。分区主表不保存实际数据，只保存表定义、分区信息，实际数据保存在分区子表中。

每个分区子表包含定义的分区子表列和聚集索引列。并且，在所有的分区子表上建立了聚集索引。这样拆分后，每个子表只包含较少列数据，通过聚集索引列进行行匹配，各个分区子表中的每个逻辑行与其他分区子表的相同逻辑行匹配，可以方便的还原主表定义的完整数据。

由于用户数据都存储在分区子表中，而子表对用户是透明的，因此，当用户需要获得分区主表所有列数据时，分区子表间会根据聚集索引键进行连接操作还原主表数据。而对主表的插入、更新和删除都将影响到相应的分区中。

如果查询只使用到垂直分区表的某些列，系统判断查询需要的数据是否只涉及某个分区子表，如果是，直接使用分区子表替代分区主表来进行查询优化。系统只需以较少的 I/O 代价，即可获得需要的数据。

以下是一个垂直分区表的例子：

```
CREATE TABLE orders(
    orderid     INT CLUSTER PRIMARY KEY,
    status          INT,
    price           DOUBLE,
    orderdate       DATETIME,
    customer        VARCHAR(20),
    comments        VARCHAR(256)
)
PARTITION BY COLUMN (
(orderid, status, price, orderdate) AS orders_p1 STORAGE (ON ts1),
(customer, comments) AS orders_p2 STORAGE (ON ts2)
);
```

订单表 orders 中经常访问的字段 orderid、status、price 和 orderdate 存放在垂直分区 orders_p1 中，其他字段存储在分区 orders_p2 中。orders 表中 orderid 是聚簇主键， orderid 虽然不是 orders_p2 的分区列，但也存储了聚簇主键 orderid 的副本，这样分区 orders_p1 和 orders_p2 就可通过 orderid 联系起来。

16.8 垂直分区表的限制

DM7 垂直分区表的限制如下：

1. 包含 IDENTITY 列的表不允许定义垂直分区表；
2. 垂直分区表不允许建立触发器；
3. 除 CLUSTER KEY 列外，其他任何列只能出现在一个分区中；
4. 垂直分区表上允许建立外键约束；
5. 垂直分区表上不允许建立除 UNIQUE 外的 CHECK 约束；

6. 不允许跨分区定义索引；
7. 一张垂直分区表最多允许有 32 个分区；
8. 禁止用户对垂直分区表子表的 INSERT/UPDATE/DELETE 直接操作，但允许直接 SELECT；
9. 禁止通过 ALTER TABLE 方式为垂直分区表增加 PK 约束；
10. ALTER TABLE 目前只支持为主表的列添加默认值、ALTER TABLE RENAME 表名（主表或子表）和 DROP CONSTRAINT，其余均报错返回；
11. 垂直分区表不支持建表的 BRANCH 选项；
12. 垂直分区表上视图不支持更新 CLUSTER KEY 列；
13. 垂直分区表中所有的大字段列必须位于同一个子表中；
14. 垂直分区表子表必须至少包含一个非 CLUSTER KEY 的列；
15. 垂直分区表在建表后不能更改聚集索引；如果建表时没有指定聚集索引，后续也不能再指定；
16. 垂直分区表上视图的 CHECK 约束不能跨分区，并且此时视图不能嵌套；
17. 垂直分区表不支持自引用约束；
18. 垂直分区表不支持 return into 功能。

第17章 管理列存储表

17.1 什么是列存储

达梦数据库中，表的数据存储方式分为行存储和列存储。行存储是以记录为单位进行存储的，数据页面中存储的是完整的若干条记录；列存储是以列为单位进行存储的，每一个列的所有行数据都存储在一起，而且一个指定的页面中存储的都是某一个列的连续数据。

下面以基表（表 17.1）为例，对行、列存储进行对比。

表 17.1 基表

NAME 列	ID 列
SYSCLASSES	268435457
SYSCOLUMNS	268435458
SYSCONS	268435459
SYSDUAL	268435460
SYSGRANTS	268435461

下图展示了相同的表数据在行存储表与列存储表中的不同存储方式。

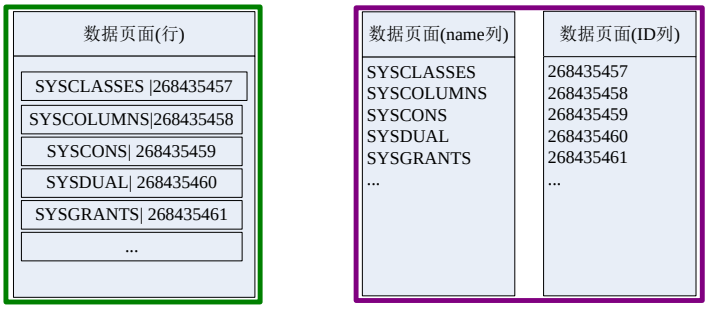


图 17.1 行存储（左）和列存储对比图

Huge File System（检查 HFS）是达梦数据库实现的，针对海量数据进行分析的一种高效、简单的列存储机制。列存储表（也称为 HUGE 表）就是建立在 HFS 存储机制上的一种表。

17.2 什么是 HUGE 表

HUGE 表是建立在自己特有的表空间 HTS（HUGE TABLESPACE，即 HUGE 表空间）上的。最多可创建 32767 个 HUGE 表空间，其相关信息存储在动态视图 V$HUGE_TABLESPACE 中。

这个表空间与普通的表空间不同。普通的表空间，数据是通过段、簇、页来管理的，并且以固定大小（4K、8K、16K、32K）的页面为管理单位；而 HUGE 表空间是通过 HFS 存储机制来管理的，它相当于一个文件系统。创建一个 HTS，其实就是创建一个空的文件目录

（系统中有一个默认 HTS，目录名为 HMAIN）。在创建一个 HUGE 表并插入数据时，数据库会在指定的 HTS 表空间目录下创建一系列的目录及文件。HFS 文件系统结构如下图所示：

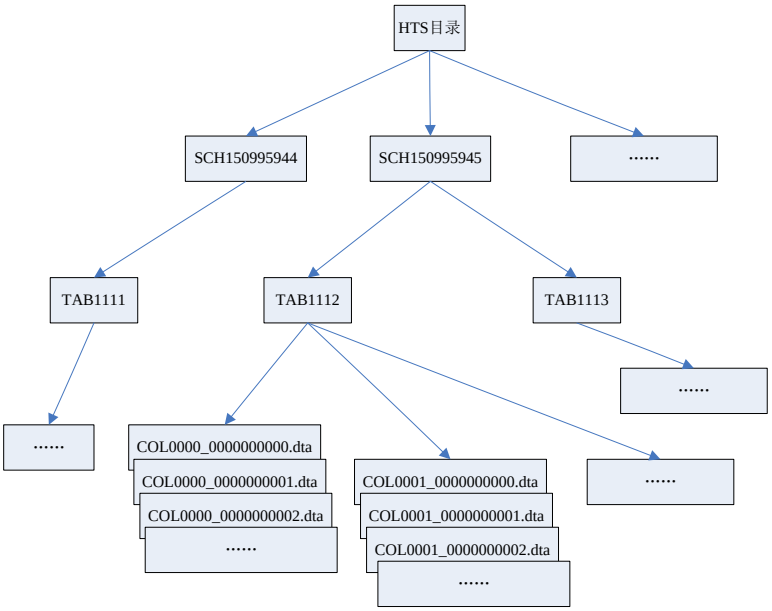


图 17.2 HFS 结构示意图

从上图可以看出，在 HTS 目录下成功创建 HUGE 表，系统内部需要经过以下步骤：

首先，在 HTS 目录下创建这个表对应的模式目录。目录名为“SCH+长度为 9 的 ID 号”组成的字符串，比如：SCH150995944。创建时如果这个目录已经存在，则无需重新创建。

其次，在模式目录下创建对应的表目录。表目录也是同样的道理，表目录名为“TAB+长度为4的ID号”组成的字符串，比如：TAB1112。表目录中存放的是这个表中所有的文件。

再次，在新创建表后插入数据时，每一个列对应一个以dta为后缀的文件，文件大小可以在建表时指定，默认为64M，文件名为“COL+长度为4的列号+_+长度为10的文件号”。例如，在图20.2中，0000表示第1列，0001表示第2列……；0000000000表示第1个文件，0000000001表示第2个文件……。最初一个列只有一个文件即可，当随着数据量不断增长，一个文件已放不下之后，系统会自动创建新的文件来存储不断增长的数据。

对于一个文件，其内部存储是按照区来管理的，区是文件内部数据管理的最小单位，也是唯一的单位。一个区中，可以存储单列数据的行数是在创建表时指定的，一经指定，在这个表的生命过程就不能再修改。所以，对于定长数据，一个区的大小是固定的；而对于变长数据，一般情况下区大小都是不相同的。每一个区的开始位置及长度在文件内都是 4K 对齐的。

HUGE 表的存储方式有以下几个优点：

1. 同一个列的数据都是连续存储的，可以加快某一个列的数据查询速度；
2. 连续存储的列数据，具有更大的压缩单元和数据相似性，可以获得远优于行存储的压缩效率，压缩的单位是区；
3. 条件扫描借助数据区的统计信息进行精确过滤，可以进一步减少 IO，提高扫描效率；
4. 允许建立二级索引；
5. 支持以 ALTER TABLE 的方式添加或者删除 PK 和 UNIQUE 约束。

DM 支持两种类型的 HUGE 表：非事务型 HUGE 表和事务型 HUGE 表，下面分别进行介

绍。

17.3 非事务型 HUGE 表

对非事务型 HUGE 表的增、删、改是直接对 HUGE 表进行写操作的，不写 UNDO 日志，不通过 BUFFER 缓存，直接操纵文件，速度快，但也因此导致不支持事务。另外，非事务型 HUGE 表中的 ROWID 是不固定的。

当非事务型 HUGE 表在操作过程中出现系统崩溃或者断电等问题时，因为修改时采取的是直接写的策略，所以有可能会出现数据不一致的问题。为了保证数据的一致性，在操作时可以适当地做一些日志来保证数据的完整性，完整性保证策略主要是通过数据的镜像来实现的，镜像的不同程度可以实现不同程度的完整性恢复。镜像文件是放在表目录中的以.mir 为扩展名的文件。DM 提供三种方案：

1. LOG NONE：不做镜像。相当于不做数据一致性的保证，如果出错只能手动通过系统函数来修复表数据，当然速度是最快的，不需要额外的 IO，这种选项如果用户明确知道自己的环境不会出现问题可以采用，效率最高。
2. LOG LAST：做部分镜像。但是在任何时候都只对当前操作的区做镜像，如果当前区的操作完成了，那么这个镜像也就失效了，可能会被下一个被操作区覆盖，这样做的好处是镜像文件不会太大，同时也可以保证数据是完整的。但有可能遇到的问题是：一次操作很多的情况下，有可能一部分数据已经完成，另一部分数据还没有来得及做的问题。如果用户能接受这个问题的话这个选择不失为最佳选择，这也是系统默认的选择。
3. LOG ALL：全部做镜像。在操作过程中，所有被修改的区都会被记录下来，当一次操作修改的数据过多时，镜像文件有可能会很大，但好处是，能够保证操作完整性。比如，在操作过程中失败了，那么这个操作会完整的撤消，不存在上面一部分修改部分还没修改的问题。

17.3.1 AUX 辅助表

对于每个 HUGE 表，相应地配备一个 AUX 辅助表来管理其数据。因为在 HUGE 表文件中只存储了数据，辅助表用来管理以及辅助系统用户操作这些数据，AUX 辅助表是在创建 HUGE 表时系统自动创建的，表名为"表名$AUX"，如果该 HUGE 表为分区表，则辅助表名为“子表名$AUX”。辅助表的表名长度不能大于 128 个字节。AUX 辅助表中每一条记录对应文件中的一个数据区，包括下面 15 列：

1. COLID：表示当前这条记录对应的区所在的列的列 ID 号；
2. SEC_ID：表示当前这个记录对应的区的区 ID 号，每一个区都有一个 ID 号，并且唯一；
3. FILE_ID：表示这个区的数据所在的文件号；
4. OFFSET：表示这个区的数据在文件中的偏移位置，4K 对齐；
5. COUNT：表示这个区中存储的数据总数（有可能包括被删除的数据）；
6. ACOUNT：表示这个区中存储的实际数据行数；
7. N_LEN：表示这个区中存储的数据在文件中的长度，4K 对齐的；
8. N_NULL：表示这个区中的数据中包括的 NULL 值的行数；
9. N_DIST：表示这个区中所有数据互不相同的行数；

10. CPR_FLAG：表示这个区是否压缩；
11. ENC_FLAG：表示这个区是否加密；
12. CHKSUM：用来较验的，该功能暂未启用；
13. MAX_VAL：表示这个区中的最大值，精确值；
14. MIN_VAL：表示这个区中的最小值，精确值；
15. SUM_VAL：表示这个区中所有值的和，精确值。

前面 7 列是用来控制数据存取的，根据这些信息就可以知道这个区的具体存储位置、长度及基本信息。后面 8 列都是用来对这个区进行统计分析的。其中，COLID 和 SEC_ID 的组合键为辅助表的聚集关键字。

17.4 事务型 HUGE 表

非事务型 HUGE 表在进行增、删、改时直接对 HUGE 表进行写操作，每次写操作需要至少对一个区进行 IO，导致 IO 量较大，且并发性能不高。

为此，DM 推出了事务型 HUGE 表，通过增加 RAUX、DAUX 和 UAUX 行辅助表，减少了事务型 HUGE 表增、删、改操作的 IO，提高效率，同时提高并行性能。事务型 HUGE 表支持 UNDO 日志，实现了事务特性。

17.4.1 RAUX 行辅助表

RAUX 行辅助表存放最后一个数据区（不够存满一个数据区）的数据，表名为"HUGE 表名$RAUX"。如果该 HUGE 表为分区表，则辅助表名为“子表名$RAUX”。辅助表的表名长度不能大于 128 个字节。

RAUX 行辅助表中内容对应于 HUGE 表中的最后一部分记录（不够存满一个数据区的）。RAUX 表与 HUGE 表结构相同，不论数据在那个表中，每一行数据的 ROWID 固定不变。

17.4.2 DAUX 行辅助表

DAUX 行辅助表记录 HUGE 表数据文件中被删除的数据，表名为"HUGE 表名$DAUX"。如果该 HUGE 表为分区表，则辅助表名为“子表名$DAUX”。辅助表的表名长度不能大于 128 个字节。

17.4.3 UAUX 行辅助表

UAUX 行辅助表记录 HUGE 表被更新的数据的新值，表名为"HUGE 表名$UAUX"。如果该 HUGE 表为分区表，则辅助表名为“子表名$UAUX”。辅助表的表名长度不能大于 128 个字节。

17.5 创建 HUGE 表

当用户确定自己要使用 HUGE 表的时候，首先需要在模式中创建新表，创建一个 HUGE

表需要有 CREATE TABLE 数据库权限，要想在其他用户的模式中创建新表需要有 CREATE ANY TABLE 数据库权限。

但是创建一个 HUGE 表时，如果不使用默认的表空间，则必须要先创建一个 HUGE TABLESPACE（HTS），创建 HTS 语法如下：

```
CREATE HUGE TABLESPACE <表空间名> PATH <表空间路径>;
```

参数说明：

1．<表空间名> 表空间的名称，表空间名称最大长度 128 字节；

2．<表空间路径> 指明新生成的表空间在操作系统下的路径。

示例如下：

```
CREATE HUGE TABLESPACE HTS_NAME PATH 'e:\HTSSPACE';
```

创建 HUGE 表的语法请参考《DM7_SQL 语言使用手册》。

在创建 HUGE 表时，根据 WITH|WITHOUT DELTA 区分创建非事务型 HUGE 表还是事务型 HUGE 表。指定 WITH DELTA，创建事务型 HUGE 表；指定 WITHOUT DELTA，则创建非事务型 HUGE 表，缺省为 WITHOUT DELTA。

例如，创建非事务型 HUGE 表 T1。

```
CREATE HUGE TABLE T1 (A INT, B INT) STORAGE(WITHOUT DELTA);
```

创建事务型 HUGE 表 T2。

```
CREATE HUGE TABLE T2 (A INT, B INT) STORAGE(WITH DELTA);
```

需要注意的是，当指定创建事务型 HUGE 表时，指定 HUGE 表镜像文件方案的选项 LOG NONE|LOG LAST|LOG ALL 失效。

另外，在创建表 HUGE 表时，可以指定表的存储属性，存储属性包括如下几个方面：

1．区大小（一个区的数据行数）

区大小可以通过设置表的存储属性来指定，区的大小必须是 2 的多少次方，如果不是则向上对齐。取值范围：1024 行~1024*1024 行。默认值为 65536 行。

例如，创建 HUGE 表 test，指定区的大小为 2048，其它默认。

```
CREATE HUGE TABLE test(name VARCHAR, sno INT) STORAGE(SECTION (2048));
```

例如，创建 HUGE 表 test，所有列都采用默认值。

```
CREATE HUGE TABLE test(name VARCHAR, sno INT);
```

2．是否记录区统计信息，即在插入时是否记下其最大值最小值

关于这一点有一个原则，如果这个列经常用作查询条件，并且数据不是很均匀，或者基本是有序的，那么做统计信息是非常有用的，反之则可以不做统计。缺省情况下，为记录区统计信息。如果想不记录，可通过设置 STAT NONE 实现。

例如，创建 HUGE 表 test，通过列存储属性指定统计信息属性（不记录区统计信息）。

```
CREATE HUGE TABLE test(name VARCHAR STORAGE (STAT NONE), sno INT);
```

又如，创建 HUGE 表 test，通过表存储属性指定统计信息属性（不记录区统计信息）。

```
CREATE HUGE TABLE test(name VARCHAR, sno INT) STORAGE (STAT NONE);
```

3．所属的表空间

创建 HUGE 表，需要通过存储属性指定其所在的表空间，不指定则存储于默认表空间 HMAIN 中。HUGE 表指定的表空间只能是 HTS 表空间，例如 HTS_NAME 为已指定 HTS 表空间。

```
CREATE HUGE TABLE test(name VARCHAR, sno INT) STORAGE (ON HTS_NAME)
```

4．文件大小

创建 HUGE 表时还可以指定单个文件的大小，通过表的存储属性来指定，取值范围为 16M~1024*1024M。不指定则默认为 64M。文件大小必须是 2 的多少次方，如果不是则向

上对齐。

```
CREATE HUGE TABLE test(name VARCHAR, sno INT) STORAGE (filesize(64));
```

5. 指定压缩级别

为特定列指定压缩级别，取值范围 0~10，分别代表不同的算法和级别。有两种压缩算法：SNAPPY 和 ZIP。10 采用 SNAPPY 算法轻量级方式压缩。2~9 采用 ZIP 算法压缩，2~9 代表压缩级别，值越小表示压缩比越低、压缩速率越快；值越大表示压缩比越高、压缩速度越慢。0 和 1 为快捷使用，默认值为 0。0 等价于 LEVEL 2；1 等价于 LEVEL 9。

例如，创建 HUGE 表 test，指定 sno 列按照最大压缩比压缩。

```
CREATE HUGE TABLE test(name VARCHAR, sno INT) COMPRESS LEVEL 1 (sno);
```

下面是一个综合的创建 HUGE 表的例子：

```
CREATE HUGE TABLE orders
(
    o_orderkey          INT,
    o_custkey           INT,
    o_orderstatus           CHAR(1),
    o_totalprice            FLOAT,
    o_orderdate         DATE,
    o_orderpriority         CHAR(15),
    o_clerk             CHAR(15),
    o_shippriority          INT,
    o_comment           VARCHAR(79) STORAGE(stat none)
)STORAGE(section(65536) , filesize(64), with delta on HTS_NAME) COMPRESS
LEVEL 9 FOR 'QUERY HIGH'  (o_comment);
```

这个例子创建了一个名为 ORDERS 的事务型 HUGE 表，ORDERS 表的区大小为 65536 行，文件大小为 64M，指定所在的表空间为 HTS_NAME，o_comment 列指定的区大小为不做统计信息，其它列（默认）都做统计信息，指定列 o_comment 列压缩类型为查询高压缩率，压缩级别为 9。

17.6 HUGE 表使用说明

HUGE 表与普通行表一样，可以进行增、删、改操作，操作方式也是一样的。但 HUGE 表的删除与更新操作的效率会比行表低一些,并发操作性能也会比行表差一些,因此在 HUGE 中不宜做频繁的删除及更新操作。总之，HUGE 表比较适合做分析型表的存储。

另外，使用 HUGE 表时应注意存在以下一些限制：

1. 建 HUGE 表时仅支持定义 NULL、NOT NULL、UNIQUE 约束以及 PRIMARY KEY，后两种约束也可以通过 ALTER TABLE 的方式添加，但这两种约束不检查唯一性；
2. HUGE 不允许建立聚簇索引，允许建立二级索引，不支持建位图索引，其中 UNIQUE 索引不检查唯一性；
3. 不支持 SPACE LIMIT（空间限制）；
4. 不支持建立全文索引；
5. 不支持使用自定义类型；
6. 不支持引用约束；
7. 不支持 IDENTITY 自增列；

8．不支持大字段列；
9．不支持建触发器；
10．不允许垂直分区；
11．不支持游标的修改操作；
12．PK 和 UNIQUE 约束不检查唯一性，对应的索引都为虚索引；UNIQUE 索引也不检查唯一性，为实索引，索引标记中不包含唯一性标记，即和普通二级索引相同；
13．不允许对分区子表设置 SECTION 和 WITH/WITHOUT DELTA；
14．当事务型 HUGE 表进行了较多增删改操作时，应对其进行数据重整操作，以提高性能。

17.7 查看有关 HUGE 表的信息

1．表定义

对一个 HUGE 表，用户可以通过 CALL SP_TABLEDEF('SYSDBA', 'ORDERS');得到这个表的定义语句，可以具体了解表的各个列的数据类型信息、存储属性等，还可以查看在这个表上是否有压缩等等。

2．数据存储情况

HUGE 表有一个很好的特点就是有 AUX 辅助表，其中用户可以利用的信息很多，因为每一条记录对应一个区，所以可以查看每一个区的存储情况，每一个列的存储情况及每一个列中具有相同区 ID 的所有数据的情况等，还包括了很精确的统计信息，用户可以通过观察 AUX 辅助表中的信息对表进行一些相应的操作。

第18章 管理堆表

18.1 什么是堆表

普通表都是以 B 树形式存放的，ROWID 都是逻辑的 ROWID，即从 1 一直增长下去。在并发情况下，每次插入过程中都需要逻辑生成 ROWID，这样影响了插入数据的效率；对于每一条数据都需要存储 ROWID 值，也会花费较大的存储空间。堆表就是基于上述两个理由而提出的。

简单地说，堆表是指采用了物理 ROWID 形式的表，即使用文件号、页号和页内偏移而得到 ROWID 值，这样就不需要存储 ROWID 值，可以节省空间。逻辑 ROWID 在插入或修改过程中，为了确保 ROWID 的唯一性，需要依次累加而得到值，这样就影响了效率，而堆表只需根据自己的文件号、页号和页内偏移就可以得到 ROWID，提高了效率。

普通表都是以 B 树形式而存储在物理磁盘上，而堆表则采用一种"扁平 B 树"方式存储，结构如下图所示。

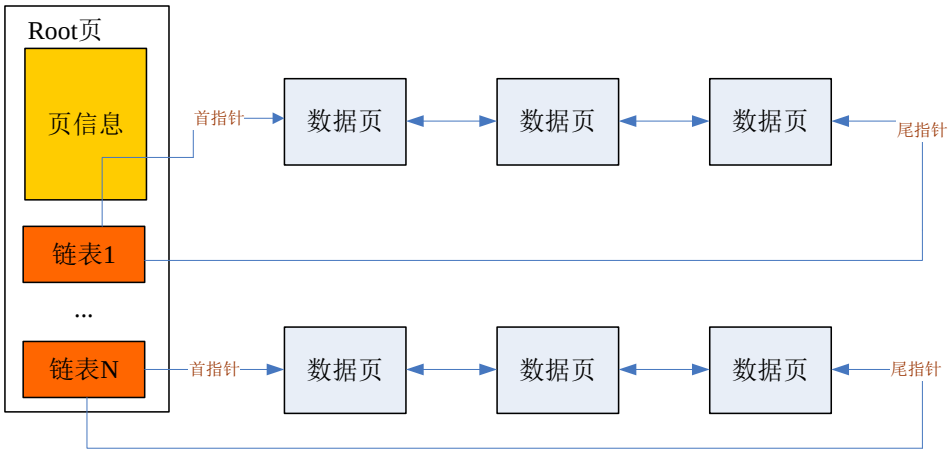


图 18.1 堆表存储方式

采用了物理 ROWID 形式的堆表，DM 服务器内部对聚集索引进行了调整，没有采用传统 B 树结构，取而代之的是"扁平 B 树"，数据页都是通过链表形式存储。为支持并发插入，扁平 B 树可以支持最多 128 个数据页链表（最多 64 个并发分支和最多 64 个非并发分支），在 B 树的控制页中记录了所有链表的首、尾页地址。对于非并发分支，如果分支数有多个，即存在多个链表，则不同的用户登录系统之后，会依据其事务 ID 号，随机选择一条链表来对堆表进行插入操作。对于并发分支，则不同用户会选择不同的分支来进行插入，如果存在多个用户选择了同一条分支的情况，才需要等待其他用户插入结束并释放锁之后才能进行插入。在并发情况下，不同用户可以在不同的链表上进行插入，效率得到较大提升。

18.2 创建堆表

堆表的创建有两种方式，一种是采用在配置文件 dm.ini 中设置参数，一种是在建表语句中显式指定堆表选项。

1. INI 参数方式

用户可以在配置文件中，添加 LIST_TABLE 参数：

1) 如果 LIST_TABLE = 1，则在未显式指定表是否为堆表或非堆表时，默认情况下创建的表为堆表；
2) 如果 LIST_TABLE = 0，则在未显式指定表是否为堆表或非堆表时，默认情况下创建的表为普通表形式。

2. SQL 语句显示指定

不管参数 LIST_TABLE 设置为何值，创建表时可以在 STORAGE 选项中指定需要创建的表形式， 与堆表创建形式相关的关键字有三个，分别是 NOBRANCH、BRANCH、CLUSTERBTR。详细语法形式参见《DM7_SQL 语言使用手册》。

1) NOBRANCH：如果指定为 NOBRANCH，则创建的表为堆表，并发分支个数为 0，非并发分支个数为 1；
2) BRANCH(n，m)：如果为该形式，则创建的表为堆表，并发分支个数为 n，非并发个数为 m；
3) BRANCH n：指定创建的表为堆表，并发分支个数为 n，非并发分支个数为 0；
4) CLUSTERBTR：创建的表为非堆表，即普通 B 树表。

如下例创建的 LIST_TABLE 表有并发分支 2 个，非并发分支 4 个。

```
CREATE TABLE LIST_TABLE(C1 INT) STORAGE(BRANCH (2,4));
```

18.3 堆表的限制

堆表由于其自身的特性，与普通表相比，也为自己增添了一些限制。堆表不支持以下功能：

1. 聚集索引

堆表采用了物理 ROWID，即通过文件号、页号和页内偏移直接生成该值。这样如果我们知道了 ROWID 值，也就知道文件号、页号和页内偏移这些变量，就可以直接定位到某条记录，所以没有必要再为堆表创建聚集索引了。在创建堆表时，系统会默认创建聚集索引，该索引只是一个根页信息。显式建立聚集索引是不允许的，如果用户需要借助聚集索引主键对数据进行排序则不推荐使用堆表。

2. 列存储

由于列存储采用了不同方式对表进行物理存储， DM 服务器暂时不支持堆表的列存储。

18.4 维护堆表

堆表在进行数据扫描过程中，有着其先天的优势。如果知道了数据记录的 ROWID，则直接可以对 ROWID 进行解码，得到该记录的文件号、页号和页内偏移，也就得到了该记录。所以建议在经常查询的列上建立二级索引，这样在进行操作中，先通过二级索引找到记录 ROWID，就可以直接找到数据，效率有较大提高。

堆表虽然支持表的 ALTER 操作，但是建议轻易不要进行此类操作。对表进行 ALTER 操作，数据记录的 ROWID 有可能发生改变，这样每次进行 ALTER 操作，都可能进行索引的重建，需要花费较多的时间。

达梦服务器支持对堆表的备份与还原操作。还原数据时，B 树数据和二级索引可以同时被还原。

18.5 查看有关堆表的信息

可以通过系统过程 SP_TABLEDEF('SCHEMA_NAME','TABLE_NAME')查看堆表的定义信息，该函数的详细信息可参考 《DM7_SQL 语言使用手册》。

第19章 全文检索

现有的数据库系统，绝大多数是以结构化数据为检索的主要目标，因此实现相对简单。比如数值检索，可以建立一张排序好的索引表，这样速度可以得到提高。但对于非结构化数据，即全文数据，要想实现检索，一般都是采用模糊查询的方式实现的。这种方式不仅速度慢，而且容易将汉字错误切分，于是产生了全文检索技术。

全文检索技术是智能信息管理的关键技术之一，其主要目的就是实现对大容量的非结构化数据的快速查找。DM 实现了全文检索功能，并将其作为 DM 服务器的一个较独立的组件，提供更加准确的全文检索功能，较好地解决了模糊查询方式带来的问题。

19.1 全文检索概述

DM 全文检索根据已有词库建立全文索引，文本查询完全在索引上进行。全文索引为在字符串数据中进行复杂的词搜索提供了有效支持。

用户可以在指定表的文本列上创建和删除全文索引。创建全文索引后全文索引未插入任何索引信息。当用户填充全文索引时，系统才将定义了全文索引的文本列的内容进行分词，并根据分词结果填充索引。用户可以在进行全文索引填充的列上使用 CONTAINS 谓词进行全文检索。

DM7 全文索引改进了原有的分词算法，为全文检索提供了更好的基础。在创建全文索引成功后，假设索引名为 INDEX_NAME，则系统会自动产生如下相关的辅助表(后面简称 I 表,P 表,N 表,D 表):CTI$INDEX_NAME$I,CTI$INDEX_NAME$P, CTI$INDEX_NAME$N 和 CTI$INDEX_NAME$D,，其表结构如 20.1~20.4 所示。I 表主键为（WORD、FIRSTID、WID），用于保存分词结果，记录词的基本信息，通过该信息就可以快速地定位到该词的基表记录；P 表主键为（PND_DOCID），用于保存基表发生的增量数据变化，用于修改全文索引时的增量填充。N 表主键为(N_DOCID),用于保存原表记录 rowid 和新词条记录的 docid 的映射关系,N_DOCID 是 unique 的;D 表主键为(DOCID),保存了所有将被删除的 docid，被删除的 docid 即将不能通过全文索引查询到。另外，如果原表有自定义聚集主键，那么 P 表和 N 表会将该自定义聚集主键列“复制”到各自表中。

表 20.1 CTI$INDEX_NAME$I 表结构

序号	字段名	类型	长度	精度	刻度	说明
1	WID	BIGINT	8	19	0	词 ID
2	WORD	VARCHAR	64	64	0	词文本，相同的词重复存储
3	TYPE	SMALLINT	2	5	0	词类型
4	FIRSTID	BIGINT	8	19	0	开始 ROWID，用于范围查找
5	LASTID	BIGINT	8	19	0	结束 ROWID
6	COUNT	INTEGER	4	10	0	词所在的文档数（即 ROWID 的个数）
7	ID_INFO	VARBINARY	4000	4000	0	保存所在文档所有的 ROWID，连续存放

表 20.2 CTI$INDEX_NAME$P 表结构

序号	字段名	类型	长度	精度	刻度	说明
1	PND_ROWID	BIGINT	8	19	0	原表记录的 ROWID

2	PND_YPTE	SMALLINT	2	5	0	INS/UPD/DEL 类型
	cluster key	...	...	...	...	原表上的自定义聚集主键

表 20.3 CTI$INDEX_NAME$N 表结构

序号	字段名	类型	长度	精度	刻度	说明
1	N_DOCID	BIGINT	8	19	0	与原表 ROWID
2	N_ROWID	BIGINT	8	19	0	原表记录 ROWID
	cluster key	...	...	...	...	原表上的自定义聚集主键

表 20.4 CTI$INDEX_NAME$D 表结构

序号	字段名	类型	长度	精度	刻度	说明
1	DOCID	BIGINT	8	19	0	待删除词的 DOCID

例如对示例库 bookshop 中的 address 表的 address1 列创建全文索引，创建的 SQL 语句如下。

```
    CREATE CONTEXT INDEX cti_address ON person.address (address1) LEXER
DEFAULT_LEXER;
```

执行成功之后，该索引信息会保存到 ctisys 模式下的 SYSCONTEXTINDEXES 系统表中，4 个自动生成的辅助表 CTI$ CTI_ADDRESSI,CTI CTI_ADDRESSP, CTI CTI_ADDRESS$N 和 CTI$ CTI_ADDRESS$D 信息保存在 SYSOBJECTS 系统表中。这四张表由服务器自动维护，用户可以查询，但不可以直接修改表内容。

DM 全文索引创建的过程中，用户可以为分词器定义分词参数，即控制分词器分词的数量，包括 5 种分词参数：

1. CHINESE_LEXER，中文最少分词；
2. CHINESE_VGRAM_LEXER，机械双字分词；
3. CHINESE_FP_LEXER，中文最多分词；
4. ENGLISH_LEXER，英文分词；
5. DEFAULT_LEXER，默认分词，为中文最少分词。

指定中文分词参数可以切分英文，但是指定英文分词参数不可以切分中文，所以使用英文分词算法对中文文本进行分词时，分词结果将为空。

中文最多分词可以将存在二义性的词划分出来，例如“和服装”会生成“和”、“和服”、“服”、“服装”和“装”。那么在查询其中任何一个词的时候都可以检索到该文本。与之相对应的是最少分词，其消除了存在二义性的词，即前面举例的文本分词的结果为“和”、“服装”两个词，这样可以减少大量冗余词的存储，能进行更准确的匹配。英文分词即根据英文分隔符的分词，对超过 32 个字节的英文单词进行了拆分处理。

全文检索的中文分词依赖系统词库，该词库是只读的，不允许修改。

19.2 创建全文索引

在 DM 中，全文索引必须在基表上定义，而不能在系统表、视图、临时表、列存储表、垂直分区表、外部表上定义。同一列只能创建一个全文索引。DM 定义全文索引时，不需要在表上先建立主关键字索引。创建全文索引的列类型可为 CHAR、CHARACTER、VARCHAR、VARCHAR2、LONGVARCHAR、TEXT 或 CLOB，例如上节对 address 表创建的全文索引列为 VARCHAR 类型。DM 的全文索引现只支持简体中文、英文语言。较为详细的语法介绍可参考《DM7_SQL 语言使用手册》。

19.3 更新全文索引

全文索引本质是借助辅助关系表存储索引数据。当对基表执行数据更新操作后，数据并不会立刻填充到辅助表中，需要用户主动同步数据用以更新全文索引，未同步到辅助表中的记录将无法通过全文索引查询到结果。

全文索引的更新包括两种方式：完全更新和增量更新。

1． 完全更新

删除原有的全文索引，对基表进行全表扫描，逐一重构索引信息。在创建全文索引成功后，需完完全更新全文索引才可以执行有效的全文检索。完全更新全文索引没有次数限制，用户可以根据需要在增量更新或者是完全更新失败以及发生系统故障后都可以执行完全更新全文索引。另外，完全更新由于完全丢弃辅助表已有数据，重新开始对基表数据进行分词并填充到辅助表，因此服务器允许这种情况下更改分词算法。

例如执行如下 SQL 可以对 address 表的全文索引进行完全更新：

```
ALTER CONTEXT INDEX cti_address ON person.address REBUILD;
```

2． 增量更新

在上一次完全更新之后，表数据发生了变化，就可以使用增量更新来更新索引。增量更新只对新发生改变的数据进行重新分词生成索引信息，所以其代价是较小的。此时，也可以使用完全更新达到同样的更新索引的效果，但代价较大。

对全文索引的基表进行数据的增删改操作时，会将该数据的操作类型和 ROWID 信息保存到 CTI$INDEX_NAME$P 表中。

例如执行如下 SQL 可以对 address 表的全文索引进行增量更新：

```
ALTER CONTEXT INDEX cti_address ON person.address INCREMENT;
```

DM 在执行更新全文索引的过程中如果在更新过程中发生任何错误，服务器将回滚到最近一次事务提交成功时的状态。

增量更新还有一种特殊情况，实时自动增量更新，即在创建全文索引时指定同步方式为"SYNC TRANSACTION"，此种方式不需要用户手动更新全文索引，而是在每次事务提交时由服务器自动完成增量更新动作。

19.4 执行全文检索

全文检索采用的分词算法与全文索引数据同步时采用的分词算法保持一致，用户可以通过系统表"CTISYS.SYSCONTEXTINDEXES"查询"WSEG_TYPE"得知全文索引的分词算法。全文检索首先对查询关键字或者句子进行分词，并将分词结果在 I 表中进行匹配，然后将匹配结果返回。DM7 支持对中英文关键字或者句子的布尔查询。

全文检索支持的检索类型有：

1． 支持英文单词的检索（单词不区分大小写）；
2． 支持全角英文的检索；
3． 支持简体中文词语的检索；
4． 支持常见单个汉字词的检索；
5． 支持简体中文长句子的检索；
6． 支持中英文混合的检索。

全文检索支持的检索方式有：

1. 在 CONTAINS 谓词内支持 AND、OR 和 AND NOT 的短语查询组合条件，例如查询 address 表中地址在洪山区太阳城的记录：

```
SELECT * FROM person.address WHERE CONTAINS(address1, '洪山区' AND '太阳城');
```

2. 全文检索支持单词或者句子的检索，例如查询 address 表中地址在江汉区发展大道的记录：

```
SELECT * FROM person.address WHERE CONTAINS(address1, '江汉区发展大道');
```

3. 不支持模糊方式的全文检索，例如查询"江汉区*"。
4. 检索条件子句可以和其他子句共同组成 WHERE 的检索条件，例如查询 address 表中地址在武汉市洪山区的记录。

```
SELECT * FROM person.address WHERE CONTAINS(address1, '洪山区') AND city LIKE
'%武汉市%';
```

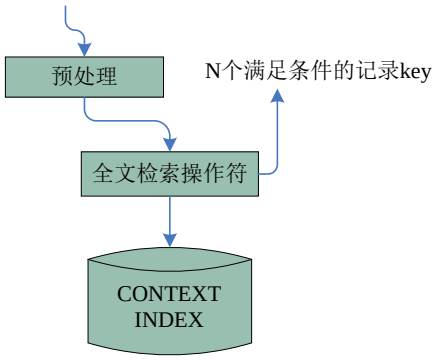


图 19.1 全文查询

如上图所示，DM 全文检索时对查询的文本做了预处理，首先根据词文本进行最少分词，并依据词频信息找出查询关键字，然后对这些关键字进行优先级排序，这样可以在第一次检索时找出所需文本的最小结果集，然后在该结果集的基础上再进行依次筛选，最终获得查询文本，这样能较大地提高查询性能。

19.5 删除全文索引

删除全文索引时，数据字典中相应的索引信息和全文索引内容都会被删除。删除索引有两种方式：使用语句删除和当模式对象发生改变时系统自动删除。第一种方式仅删除指定的基表的全文索引信息；第二种方式会删除模式对象（如数据库、基表等）上的所有全文索引。全文索引一旦删除就不能回滚，基于该表的全文检索就会失败，只能通过重新构建获得新的全文索引。例如删除 address 表，则会将表的全文索引 cti_address 也删除，包括与全文索引相关的动态表。

```
DROP CONTEXT INDEX cti_address ON person.address;
```

删除成功后，查询 ctisys 模式下的 SYSCONTEXTINDEXES 系统表会发现 cti_address 全文索引已删除。

第20章 管理事务

数据库是一个共享资源，可以被大量应用程序所共享。这些应用程序可以串行运行，但在绝大多数情况下，为了有效地利用数据库资源，多个应用程序会并发地访问数据库，这就是数据库的并发操作。此时，如果不对并发操作进行控制，则会存取不正确的数据，或破坏数据库数据的一致性。DM 数据库通过事务管理相关技术，有效解决了上述问题。本章阐述了事务的定义，并讲解如何使用事务来管理 DM 数据库。

20.1 事务简介

数据库事务是指作为单个逻辑工作单元的一系列操作的集合。一个典型的事务由应用程序中的一组操作序列组成，对于 DM 数据库来说，第一次执行 SQL 语句时，隐式地启动一个事务，以 COMMIT 或 ROLLBACK 语句/方法显式地结束事务。另外，在执行 DDL 前，DM 数据库会自动把前面的操作进行提交，DDL 前面的操作作为一个完整的事务结束，DDL 语句本身所属事务则根据“DDL_AUTO_COMMIT”配置参数决定是否隐式地提交（注意，无论 DDL_AUTO_COMMIT 参数如何设置，ALTER TABLESPACE 和 ALTER USER 操作总是自动提交的）。

COMMIT 操作会将该语句所对应事务对数据库的所有更新持久化（即写入磁盘），数据库此时进入一个新的一致性状态，同时该事务成功地结束。ROLLBACK 操作将该语句所对应事务对数据库的所有更新全部撤销，把数据库恢复到该事务初启动前的一致性状态。

我们以一个模拟的银行转账业务为例，假设一个银行客户 A 需要转 5000 元给 B，其具体业务步骤如下：

1. 从 A（ID 为 5236）的储蓄账户扣除 5000 元；

```
UPDATE account SET balance=balance-5000 WHERE id=5236;
```

2. 将 B（ID 为 5237）的储蓄账户增加 5000 元；

```
UPDATE account SET balance = balance +5000 WHERE id=5237;
```

3. 在业务日志中记录此次业务；

```
INSERT INTO trans_log VALUES(log_seq.NEXTVAL, 5236, 5237, 5000);
```

4. 提交事务。

```
COMMIT;
```

在上面的例子中，需要考虑两种情况：如果三条 SQL 语句全部正常执行，使账户间的平衡得以保证，那么此事务中对数据的修改就可以应用到数据库中；如果发生诸如资金不足、账号错误、硬件故障等问题，导致事务中一条或多条 SQL 语句不能执行，那么整个事务必须被回滚掉才能保证账户间的平衡。

DM 数据库提供了足够的事务管理机制来保证上面的事务要么成功执行，所有的更新都会写入磁盘，要么所有的更新都被回滚，数据恢复到执行该事务前的状态。无论是提交还是回滚，DM 保证数据库在每个事务开始前、结束后是一致的。

为了提高事务管理的灵活性，DM 数据库还提供了设置保存点（SAVEPOINT）和回滚到保存点的功能。保存点提供了一种灵活的回滚，事务在执行中可以回滚到某个保存点，在该保存点以前的操作有效，而以后的操作被回滚掉。可以使用 SAVEPOINT SAVEPOINT_NAME 命令创建保存点，使用 ROLLBACK TO SAVEPOINT SAVEPOINT_NAME

命令来回滚到保存点 SAVEPOINT_NAME。

20.2 事务特性

事务必须具备什么属性才是一个有效的事务呢？一个逻辑工作单元必须表现出四种属性，即原子性、一致性、隔离性和持久性，这样才能成为一个有效的事务。DM 数据库提供了各种机制以保证事务满足以上各种要求。

20.2.1 原子性

事务的原子性保证事务包含的一组更新操作是原子不可分的，也就是说这些更新操作是一个整体，对数据库而言全做或者全不做，不能部分地完成。这一性质即使在系统崩溃之后仍能得到保证，在系统崩溃之后将进行数据库恢复，用来恢复和撤销系统崩溃时处于活动状态的事务对数据库的影响，从而保证事务的原子性。系统对磁盘上的任何实际数据的修改之前都会将修改操作本身的信息记录到磁盘上。当发生崩溃时，系统能根据这些操作记录当时该事务处于何种状态，以此确定是撤销该事务所做出的所有修改操作，还是将修改的操作重新执行。

20.2.2 一致性

数据一致性是指表示客观世界同一事务状态的数据，不管出现在何时何处都是一致的、正确的、完整的。换句话说，数据一致性是任何点上保证数据以及内部数据结构的完整性，如 B 树索引的正确性。

一致性要求事务执行完成后，将数据库从一个一致状态转变到另一个一致状态。它是一种以一致性规则为基础的逻辑属性，例如在转账的操作中，各账户金额必须平衡，这一条规则对于程序员而言是一个强制的规定。事务的一致性属性要求事务在并发执行的情况下事务的一致性仍然满足。

20.2.3 隔离性

事务是隔离的，意味着每个事务的执行效果与系统中只有该事务的执行效果一样，也就是说，某个并发事务所做的修改必须与任何其他的并发事务所做的修改相互隔离。这样，只有当某个值被一个事务修改完并提交后才会影响到另一个事务。事务只会识别另一并发事务修改之前或者修改完成之后的数据，不会识别处于这中间状态的数据。事务的隔离行为依赖于指定的隔离级别。隔离级别的介绍请参考本章第 7 节。

20.2.4 持久性

持久性是指一个事务一旦被提交，它对数据库中数据的改变就是永久性的，接下来的其他操作和数据库故障不应该对其有任何影响。即一旦一个事务提交，DBMS 保证它对数据库中数据的改变应该是永久性的。如果 DM 数据库或者操作系统出现故障，那么在 DM 数据库

重启的时候，数据库会自动恢复。如果某个数据驱动器出现故障，并且数据丢失或者被损坏，可以通过备份和联机重做日志来恢复数据库。需要注意的是，如果备份驱动器也出现故障，且系统没有准备其他的可靠性解决措施，备份就会丢失，那么就无法恢复数据库了。

20.3 提交事务

提交事务就是提交事务对数据库所做的修改，将从事务开始的所有更新保存到数据库中，任何更改的记录都被写入日志文件并最终写入到数据文件，同时提交事务还会释放由事务占用的资源，如锁。如果提交时数据还没有写入到数据文件，DM 数据库后台线程会在适当时机（如检查点、缓冲区满）将它们写入。

具体说来，在一个修改了数据的事务被提交之前，DM 数据库进行了以下操作：

1. 生成回滚记录，回滚记录包含了事务中各 SQL 语句所修改的数据的原始值；
2. 在系统的重做日志缓冲区中生成重做日志记录，重做日志记录包含了对数据页和回滚页所进行的修改，这些记录可能在事务提交之前被写入磁盘；
3. 对数据的修改已经被写入数据缓冲区，这些修改也可能在事务提交之前被写入磁盘。

已提交事务中对数据的修改被存储在数据库的缓冲区中，它们不一定被立即写入数据文件内。DM 数据库自动选择适当的时机进行写操作以保证系统的效率。因此写操作既可能发生在事务提交之前，也可能在提交之后。

当事务被提交之后，DM 数据库进行以下操作：

1. 将事务任何更改的记录写入日志文件并最终写入到数据文件；
2. 释放事务上的所有锁，将事务标记为完成；
3. 返回提交成功消息给请求者。

在 DM 数据库中还存在有 3 种事务模式：自动提交模式、手动提交模式和隐式提交模式。

20.3.1 自动提交模式

除了命令行交互式工具 DISQL 外，DM 数据库缺省都采用自动提交模式。用户通过 DM 数据库的其他管理工具、编程接口访问 DM 数据库时，如果不手动/编程设置提交模式，所有的 SQL 语句都会在执行结束后提交，或者在执行失败时回滚，此时每个事务都只有一条 SQL 语句。

在 DISQL 中，用户也可以通过执行如下语句来设置当前会话为自动提交模式：

```
SET AUTOCOMMIT ON;
```

20.3.2 手动提交模式

在手动提交模式下，DM 数据库用户或者应用开发人员明确定义事务的开始和结束，这些事务也被称为显式事务。在 DISQL 中，没有设置自动提交时，就是处于手动提交模式，此时 DISQL 连接到服务器后第一条 SQL 语句或者事务结束后的第一条语句就标记着事务的开始，可以执行 COMMIT 或者 ROLLBACK 来提交或者回滚事务。

20.3.3 隐式提交

在手动提交模式下，当遇到 DDL 语句时，DM 数据库会自动提交前面的事务，然后开始一个新的事务执行 DDL 语句。这种事务提交被称为隐式提交。DM 数据库在遇到以下 SQL 语句时自动提交前面的事务：

1. CREATE;
2. ALTER;
3. TRUNCATE;
4. DROP;
5. GRANT;
6. REVOKE;
7. 审计设置语句。

20.4 回滚事务

回滚事务是撤消该事务所做的任何更改。回滚有两种形式：DM 数据库自动回滚，或者通过程序/ROLLBACK 命令手动回滚。除此之外，与回滚相关的还有回滚到保存点和语句级回滚，下面分别进行介绍。

20.4.1 自动回滚

若事务运行期间出现连接断开，DM 数据库都会自动回滚该连接所产生的事务。回滚会撤消事务执行的所有数据库更改，并释放此事务使用的所有数据库资源。DM 数据库在恢复时也会使用自动回滚。例如在运行事务时服务器突然断电，接着系统重新启动，DM 数据库就会在重启时执行自动恢复。自动恢复要从事务重做日志中读取信息以重新执行没有写入磁盘的已提交事务，或者回滚断电时还没有来得及提交的事务。

20.4.2 手动回滚

一般来说，在实际应用中，当某条 SQL 语句执行失败时，用户会主动使用 ROLLBACK 语句或者编程接口提供的回滚函数来回滚整个事务，避免不合逻辑的事务污染数据库，导致数据不一致。如果发生错误后确实只用回滚事务中的一部分，则需要用到回滚到保存点的功能。

20.4.3 回滚到保存点

除了回滚整个事务之外，DM 数据库的用户还可以部分回滚未提交事务，即从事务的最末端回滚到事务中任意一个被称为保存点的标记处。用户在事务内可以声明多个被称为保存点的标记，将一个大事务划分为几个较小的片断。之后用户在对事务进行回滚操作时，就可以选择从当前执行位置回滚到事务内的任意一个保存点。例如用户可以在一系列复杂的更新操作之间插入保存点，如果执行过程中一个语句出现错误，用户可以回滚到错误之前的某个

保存点，而不必重新提交所有的语句。当事务被回滚到某个保存点后，DM 数据库将释放被回滚语句中使用的锁。其他等待“被锁资源”的事务就可以继续执行，需要更新“被锁数据行”的事务也可以继续执行。

将事务回滚到某个保存点的过程如下：

1． 只回滚保存点之后的语句；

2． 保留该保存点，其后创建的保存点都被清除；

3． 释放此保存点之后获得的所有锁，保留该保存点之前的锁。

被部分回滚的事务依然处于活动状态，可以继续执行。DM 数据库用户可以使用 SAVEPOINT SAVEPOINT_NAME 命令创建保存点，使用 ROLLBACK TO SAVEPOINT SAVEPOINT_NAME 命令来回滚到保存点 SAVEPOINT_NAME。

20.4.4 语句级回滚

如果在一个 SQL 语句执行过程中发生了错误，此语句对数据库产生的影响将被回滚，回滚后就如同此语句从未执行过，这种操作被称为语句级回滚。语句级回滚只会使此语句所做的数据修改无效，不会影响此语句之前所做的数据修改。

当 INI 参数 ROLL_ON_ERR 为缺省值 0 时，在 SQL 语句执行过程中发生的错误，将会导致语句级回滚，例如违反唯一性、死锁（访问相同数据而产生的竞争）、运算溢出等。在 SQL 语句解析的过程中发生错误（例如语法错误），由于未对数据产生任何影响，因此不会产生语句级回滚。

20.5 事务锁定

DM 数据库支持多用户并发访问、修改数据，有可能出现多个事务同时访问、修改相同数据的情况。若对并发操作不加控制，就可能会访问到不正确的数据，破坏数据的一致性和正确性。DM 数据库采用封锁机制来解决并发问题，本节将详细介绍 DM 数据库中的事务锁定相关功能，包括锁的分类以及如何查看锁等。

20.5.1 锁模式

锁模式指定并发用户如何访问锁定资源。DM 数据库使用四种不同的锁模式：共享锁、排他锁、意向共享锁和意向排他锁。

1． 共享锁

共享锁（Share Lock，简称 S 锁）用于读操作，防止其他事务修改正在访问的对象。这种封锁模式允许多个事务同时并发读取相同的资源，但是不允许任何事务修改这个资源。

2． 排他锁

排他锁（Exclusive Lock，简称 X 锁）用于写操作，以独占的方式访问对象，不允许任何其他事务访问被封锁对象；防止多个事务同时修改相同的数据，避免引发数据错误；防止访问一个正在被修改的对象，避免引发数据不一致。一般在修改对象定义时使用。

3． 意向锁

意向锁（Intent Lock）用于读取或修改被访问对象数据时使用，多个事务可以同时

对相同对象上意向锁，DM 支持两种意向锁：

1） 意向共享锁（Intent Share Lock，简称 IS 锁）：一般在只读访问对象时使用；
2） 意向排他锁（Intent Exclusive Lock，简称 IX 锁）：一般在修改对象数据时使用。

四种锁模式的相容矩阵如下表所示，其中“Y”表示相容；“N”表示不相容。如表中第二行第二列为“Y”，表示如果已经加了 IS 锁时，其他用户还可以继续添加 IS 锁，第二行第五列为“N”，表示如果已经加了 IS 锁时，其他用户不能添加 X 锁。

表 21.1 锁相容矩阵

	IS	**IX**	**S**	**X**
IS	Y	Y	Y	N
IX	Y	Y	N	N
S	Y	N	Y	N
X	N	N	N	N

20.5.2 锁粒度

按照封锁对象的不同，锁可以分为 TID 锁和对象锁。

1． TID 锁

TID 锁以事务号为封锁对象，为每个活动事务生成一把 TID 锁，代替了其他数据库行锁的功能，防止多个事务同时修改同一行记录。DM 实现的是行级多版本，每一行记录隐含一个 TID 字段，用于事务可见性判断。

执行 INSERT、DELETE、UPDATE 操作时，设置事务号到 TID 字段。这相当于隐式地对记录上了一把 TID 锁，INSERT、DELETE、UPDATE 操作不再需要额外的行锁，避免了大量行锁对系统资源的消耗。只有多个事务同时修改同一行记录时，才会产生新的 TID 锁。例如，当事务 T1（事务号为 TID1）试图修改某行数据，而该行数据正在被另一个事务 T2（事务号为 TID2）修改，此时事务 T1 会生成一个新的 TID 锁，其锁对象为事务号 TID2，而非事务 T2。

同时多版本写不阻塞读的特性，SELECT 操作已经消除了行锁，因此 DM 中不再有行锁的概念。

2． 对象锁

对象锁是 DM 新引入的一种锁，通过统一的对象 ID 进行封锁，将对数据字典的封锁和表锁合并为对象锁，以达到减少封锁冲突、提升系统并发性能的目的。我们先看一下通常数据字典锁和表锁各自应承担的功能：

1） 数据字典锁：用来保护数据字典对象的并发访问，解决 DDL 并发和 DDL/DML 并发问题，防止多个事务同时修改同一个对象的字典定义，确保对同一个对象的 DDL 操作是串行执行的。并防止一个事务在修改字典定义的同时，另外一个事务修改对应表的数据。
2） 表锁：表锁用来保护表数据的完整性，防止多个事务同时采用批量方式插入、更新

一张表，防止向正在使用 FAST LOADER 工具装载数据的表中插入数据等，保证这些优化后数据操作的正确性。此外，表锁还有一个作用，避免对存在未提交修改的表执行 ALTER TABLE、TRUNCATE TABLE 操作。

为了实现与数据字典锁和表锁相同的封锁效果，从逻辑上将对象锁的封锁动作分为四类：

a) 独占访问（EXCLUSIVE ACCESS），不允许其他事务修改对象，不允许其他事务访问对象，使用 X 方式封锁

b) 独占修改（EXCLUSIVE MODIFY），不允许其他事务修改对象，允许其他事务共享访问对象，使用 S + IX 方式封锁

c) 共享修改（SHARE MODIFY），允许其他事务共享修改对象，允许其他事务共享访问对象，使用 IX 方式封锁

d) 共享访问（SHARE ACCESS），允许其他事务共享修改对象，允许其他事务共享访问对象，使用 IS 方式封锁

3. 显式锁定表

用户可以根据自己的需要显式的对表对象进行封锁。显式锁定表的语法如下：

```
LOCK TABLE <table_name> IN <lock_mode> MODE [NOWAIT];
```

lock_mode 是锁定的模式，可以选择的模式有 INTENT SHARE（意向共享）、INTENT EXCLUSIVE（意向排他）、SHARE（共享）和 EXCLUSIVE（排他），其含义分别如下：

1) 意向共享：不允许其他事务独占访问该表。意向共享锁定后，不同事务可以同时增、删、改、查该表的数据，也支持在该表上创建索引，但不支持修改该表的定义；

2) 意向排他：不允许其他事务独占访问和独占修改该表。被意向排他后，不同事务可以同时增、删、改、查该表的数据，不支持在该表上创建索引，也不支持修改该表定义；

3) 共享：只允许其他事务共享访问该表，仅允许其他事务查询表中的数据，但不允许增、删、改该表的数据；

4) 排他：以独占访问方式锁定整个表，不允许其他事务访问该表，是封锁力度最大的一种封锁方式。

当使用 NOWAIT 时，若不能立即上锁成功则立刻返回报错信息，不再等待。

20.5.3 查看锁

为了方便用户查看当前系统中锁的状态，DM 数据库专门提供了一个 V$LOCK 动态视图。通过该视图，用户可以查看到系统当前所有锁的详细信息，如锁的内存地址、所属事务 ID、上锁的对象、锁模式、对象 ID 以及行事务 ID。用户可以通过执行如下语句查看锁信息：

```
SELECT * FROM V$LOCK;
```

其结果看起来和下面类似：

```
ADDR       TRX_ID    LTYPE   LMODE   BLOCKED     TABLE_ID    ROW_IDX
---------- -------------------- -------------------- ------ -----
----------- ----------- --------------------
233113736    932       OBJECT   IX          0           1149           0
232810632    933       OBJECT   IX          0           1149        0
233113680    932       OBJECT   IS          0           -1          0
232810576    933       OBJECT   IS          0           -1          0
```

```
233113792    932       OBJECT   IS          0           4           0
232810688    933       OBJECT   IS          0           4           0
232810744    933       OBJECT   IS          0           33555437    0
232810520    933       TID      X       0           0           933
233113624    932       TID      X       0           0           932
232810800    933       TID      S       1           0           932
```

其中 ADDR 列表示锁的内存地址；TRX_ID 列表示锁所属的事务 ID；LTYPE 列表示被上锁对象的类型，可能是 OBJECT（对象锁）或者 TID（TID 锁）；LMODE 表示锁的模式，可能的取值有 S（共享锁）、X（排他锁）、IS（意向共享锁）、IX（意向排他锁）；BLOCKED 列表示锁是否处于上锁等待状态，0 表示已上锁成功，1 表示处于上锁等待状态；TABLE_ID 列表示表的 ID；ROW_IDX 列是 TID 锁的封锁对象事务 ID。

20.6 多版本

在多版本控制以前，数据库仅通过锁机制来实现并发控制。数据库对读操作上共享锁，写操作上排他锁，这种锁机制虽然解决了并发问题，但影响了并发性。例如，当对一个事务对表进行查询时，另一个对表更新的事务就必须等待。DM 数据库的多版本实现完全消除了行锁对系统资源的消耗，查询永远不会被阻塞也不需要上行锁，并通过 TID 锁机制消除了插入、删除、更新操作的行锁。数据库的读操作与写操作不会相互阻塞，并发度大幅度提高。

DM 数据库基于物理记录和回滚记录实现行级多版本支持，数据页中只保留物理记录的最新版本，通过回滚记录维护历史版本，所有事务针对特定的版本进行操作。

20.6.1 物理记录格式

为了适应多版本机制，高效地获取历史记录，每一条物理记录中包含了两个字段：TID 和 RPTR。TID 保存修改记录的事务号，RPTR 保存回滚段中上一个版本回滚记录的物理地址。插入、删除和更新物理记录时，RPTR 指向操作生成的回滚记录的物理地址。物理记录格式如下：

物理记录	TID	RPTR

新物理记录的 RPTR 指向当前回滚记录的物理地址。

20.6.2 回滚记录格式

回滚记录与物理记录一样，增加了两个字段：TID 和 RPTR。TID 保存回滚记录对应的事务号，RPTR 保存回滚段中上一个版本回滚记录的物理地址。

回滚记录	TID	RPTR

插入物理记录时，由于没有更老的版本数据，回滚记录的 RPTR 值为 NULL；更新和删除物理记录时，RPTR 指向原始物理记录的 RPTR。

20.6.3 可见性原则

实现多版本控制的关键是可见性判断，找到对当前事务可见的特定版本数据。DM 通过

活动事务表，确定事务的可见性。根据事务隔离级的不同，在事务启动时（串行化），或者语句执行时（读提交），收集这一时刻所有活动事务，并记录系统中即将产生的事务号 NEXT_TID。DM 多版本可见性原则：

1. 物理记录的 TRXID 等于当前事务号，说明是本事务修改的物理记录，物理记录可见；
2. 物理记录的 TRXID 不在活动事务表中，并且 TRXID 小于 NEXT_TID，物理记录可见；
3. 物理记录的 TRXID 包含在活动事务表中，或者 TRXID 大于等于 NEXT_TID，物理记录不可见。

20.6.4 历史数据获取

当物理记录对当前事务不可见时，根据物理记录和回滚记录的 RPTR 指针，向前回溯一个历史版本记录，通过此历史版本记录的 TID 字段，依据事务可见性原则判断此版本的记录对当前事务是否可见。如可见即获取到了满足当前事务的历史版本数据；如不可见则根据 RPTR 指针继续向前回溯。如果一直不能找到对当前事务的可见版本（例如此记录是一个活动事务插入的新记录），则此记录将不会添加到查询结果集中。

下面以 UPDATE 为例描述多版本的实现；依次执行事务 T1 和 T2：

事务 T1

```
CREATE TABLE TEST_UPDATE (COL_1 INT PRIMARY KEY, COL_2 VARCHAR(10));
INSERT INTO TEST_UPDATE VALUES(1, 'ABCD');
```

事务 T2

```
UPDATE test_update set col_2='xyz' where col_1 = 1;
```

执行以上两个事务以后，表 TEST_UPDATE 的记录了两个版本

物理记录：

物理记录，字段值为（1，‘XYZ’）	TID（T2 的 ID）	RPTR（回滚记录地址）

回滚记录：

回滚记录，保存了老记录的值（1，‘ABCD’）	TID(T1 的 ID)	RPTR

20.6.5 回滚段自动清理

由于需要根据回滚记录回溯、还原物理记录的历史版本信息，因此不能在事务提交时立即清除当前事务产生的回滚记录。但是，如果不及时清理回滚段，可能造成回滚段空间的不断膨胀，占用大量磁盘空间。

DM 提供了自动清理、回收回滚段空间的机制。系统定时（缺省是每间隔 1 秒）扫描回滚段，根据回滚记录的 TID，判断是否需要保留回滚记录，清除那些对所有活动事务可见的回滚记录空间。

20.7 事务隔离级

在关系型数据库中，事务的隔离性分为四个隔离级别，在解读这四个级别前先介绍几个关于读数据的概念。

1. 脏读（DirtyRead）

所谓脏读就是对脏数据的读取，而脏数据所指的就是未提交的已修改数据。也就是说，一个事务正在对一条记录做修改，在这个事务完成并提交之前，这条数据是处于待定状态的（可能提交也可能回滚），这时，第二个事务来读取这条没有提交的数据，并据此做进一步的处理，就会产生未提交的数据依赖关系，这种现象被称为脏读。如果一个事务在提交操作结果之前，另一个事务可以看到该结果，就会发生脏读。

2. 不可重复读（Non-RepeatableRead）

一个事务先后读取同一条记录，但两次读取的数据不同，我们称之为不可重复读。如果一个事务在读取了一条记录后，另一个事务修改了这条记录并且提交了事务，再次读取记录时如果获取到的是修改后的数据，这就发生了不可重复读情况。

3. 幻像读（PhantomRead）

一个事务按相同的查询条件重新读取以前检索过的数据，却发现其他事务插入了满足其查询条件的新数据，这种现象就称为幻像读。

在 SQL-92 标准中，定义了四种隔离级别：读未提交、读提交、可重复读和串行化。每种隔离级别下对于读数据有不同的要求，表 21.2 中列出四种隔离级别下系统允许/禁止哪些类型的读数据现象。其中“Y”表示允许，“N”表示禁止。

表 21.2 隔离级别表

解决 / 级别	脏读	不可重复读	幻像读
读未提交	Y	Y	Y
读提交	N	Y	Y
可重复读	N	N	Y
串行化	N	N	N

在只有单一用户的数据库中，用户可以任意修改数据，而无需考虑同时有其他用户正在修改相同的数据。但在一个多用户数据库中，多个并发事务中包含的语句可能会修改相同的数据。数据库中并发执行的事务最终应产生有意义且具备一致性的结果。因此在多用户数据库中，对数据并发访问及数据一致性进行控制是两项极为重要的工作。

为了描述同时执行的多个事务如何实现数据一致性，数据库研究人员定义了被称为串行化处理的事务隔离模型。当所有事务都采取串行化模式执行时，我们可以认为同一时间只有一个事务在运行（串行的），而非并发的。

DM 数据库支持三种事务隔离级别：读未提交、读提交和串行化。其中，读提交是 DM 数据库默认使用的事务隔离级别。可重复读升级为更严格的串行化隔离级。

20.7.1 读提交隔离级

DM 数据库的读提交隔离可以确保只访问到已提交事务修改的数据，保证数据处于一致性状态，能够满足大多数应用的要求，并最大限度的保证系统并发性能，但可能会出现不可重复读取和幻像读。

用户可以在事务开始时使用以下语句设定事务为读提交隔离级：

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

20.7.2 串行化隔离级

在要求消除不可重复读取或幻像读的情况下，我们可以设置事务隔离级为串行化。跟读提交隔离级相比，串行化事务的查询本身不会增加任何代价，但修改数据可能引发“串行化事务被打断”错误。

具体来说，当一个串行化事务试图更新或删除数据时，而这些数据在此事务开始后被其他事务修改并提交时，DM 数据库将报“串行化事务被打断”错误。应用开发者应该充分考虑串行化事务带来的回滚及重做事务的开销，从应用逻辑上避免对相同数据行的激烈竞争导致产生大量事务回滚。并结合应用逻辑，捕获“串行化事务被打断”错误，进行事务重做等相应处理。如果系统中存在长时间运行的写事务，并且该长事务所操作的数据还会被其他短事务频繁更新的话，最好避免使用串行化事务。

用户可以在事务开始时使用以下语句设定事务为串行化隔离级：

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

20.7.3 读未提交隔离级

DM 数据库除了支持读提交、串行化两种隔离级之外，还支持读未提交这种隔离级。

读未提交隔离级别是最不严格的隔离级别。实际上，在使用这个隔离级别时，有可能发生脏读、不可重复读和幻像。一般来说，读未提交隔离级别通常只用于访问只读表和只读视图，以消除可见性判断带来的系统开销，提升查询性能。

用户可以在事务开始时使用以下语句，设定事务为读未提交隔离级：

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
```

20.7.4 只读事务

除了前面所述的各种标准特性外，DM 数据库还支持只读事务，只读事务只能访问数据，但不能修改数据。并且只读事务不会改变事务原有的隔离级。

用户可以在事务开始时使用以下语句，设定事务为只读事务：

```
SET TRANSACTION READ ONLY;
```

20.8 锁等待与死锁检测

阻塞和死锁是会与并发事务一起发生的两个事件，它们都与锁相关。当一个事务正在占用某个资源的锁，此时另一个事务正在请求这个资源上与第一个锁相冲突的锁类型时，就会发生阻塞。被阻塞的事务将一直挂起，直到持有锁的事务放弃锁定的资源为止。死锁与阻塞的不同之处在于死锁包括两个或者多个已阻塞事务，它们之间形成了等待环，每个都等待其他事务释放锁。例如事务 1 给表 T1 上了排他锁，第二个事务给表 T2 上了排他锁，此时事务 1 请求 T2 的排他锁，就会处于等待状态，被阻塞。若此时 T2 再请求表 T1 的排他锁，则 T2 也处于阻塞状态。此时这两个事务发生死锁，DM 数据库会选择牺牲掉其中一个事务。

在 DM 数据库中，INSERT、UPDATE、DELETE 是最常见的会产生阻塞和死锁的语句。

INSERT 发生阻塞的唯一情况是，当多个事务同时试图向有主键或 UNIQUE 约束的表中

插入相同的数据时，其中的一个事务将被阻塞，直到另外一个事务提交或回滚。一个事务提交时，另一个事务将收到唯一性冲突的错误；一个事务回滚时，被阻塞的事物可以继续执行。

当 UPDATE 和 DELETE 修改的记录，已经被另外的事务修改过，将会发生阻塞，直到另一个事务提交或回滚。

20.9 闪回

当用户操作不慎导致错误的删改数据时，非常希望有一种简单快捷的方式可以恢复数据。闪回技术，就是为了用户可以迅速处理这种数据逻辑损坏的情况而产生的。

闪回技术主要是通过回滚段存储的 UNDO 记录来完成历史记录的还原。设置 ENABLE_FLASHBACK 为 1 后，开启闪回功能。DM 会保留回滚段一段时间，回滚段保留的时间代表着可以闪回的时间长度。由 UNDO_RETENTION 参数指定。

开启闪回功能后，DM 会在内存中记录下每个事务的起始时间和提交时间。通过用户指定的时刻，查询到该时刻的事务号，结合当前记录和回滚段中的 UNDO 记录，就可以还原出特定事务号的记录。即指定时刻的记录状态。从而完成闪回查询。闪回查询功能完全依赖于回滚段管理，对于 DROP 等误操作不能恢复。闪回特性可应用在以下方面：

1. 自我维护过程中的修复：当一些重要的记录被意外删除，用户可以向后移动到一个时间点，查看丢失的行并把它们重新插入现在的表内恢复；
2. 用于分析数据变化：可以对同一张表的不同闪回时刻进行链接查询，以此查看变化的数据。

第四部分 故障排除和性能优化

第21章 问题跟踪和解决

问题跟踪和解决是最复杂的脑力劳动之一，其复杂主要体现在问题的无法预知性和问题根源的多样性。本章的目的是提供一种相对比较系统的方法来查找并解决使用 DM 数据库过程中可能遇到的各种难以跟踪、定位的问题，为 DBA 分析、解决问题提供帮助。

21.1 问题分析

当系统出现问题，无法及时响应用户/应用请求时，可能的原因是多方面的。一般来说，DBA 应该查看和分析的内容包括：

1． 网络是否正常

DBA 可以直接使用各种工具/软件来排除网络问题。如果远程操作有问题，但是本地操作没有问题，则有可能是网络出现故障或者网络带宽耗尽。但是如果本地也有问题，不能说明网络一定没有问题，此时还是需要用其他方式确定网络是否有故障，同时进一步分析本地问题产生的原因。

2． 内存使用量

用户可以通过操作系统提供的内存检测工具/命令来查看数据库占用内存情况，看是否数据库占用了过多内存，并且开始大量使用页面文件（Windows）/交换分区（Linux/UNIX），如果数据库占用内存过多，则需要进一步分析可能的原因：是否数据库的内存相关参数设置错误，是否客户端请求的资源过多并且一直没有释放（如不断打开连接/游标，并且一直不关闭）等等。对于参数设置错误，DBA 可以通过修改参数加以解决。如果是客户端请求资源过多，DBA 可以通过查询运行时动态视图（参考附录 2：动态性能视图）来检查资源使用情况，明确问题产生的原因并予以纠正。若排除其他原因后，发现数据库内存仍在不断增长，此时可以联系达梦公司技术服务人员协助解决。

3． CPU 使用率

当发现系统响应很慢甚至无法响应时，CPU 的使用率也是一个重要的观察指标。如果 CPU 使用率一直持续 90%以上，甚至 100%，则说明 CPU 使用率过高，此时需要分析导致 CPU 使用率过高的原因。可能的原因包括：写了错误的存储过程/函数死循环逻辑；某条 SQL 语句执行计划不好导致 CPU 使用率过高（如没有建立合适的索引等）；系统内部 SQL 语句都执行正常，只是实际应用负载过大等。针对上述原因，DBA 可以分别考虑改正存储过程/函数的死循环逻辑、建立合适的索引以及提供更高配置的软/硬件环境等措施分别予以解决。

4． I/O 是否正常

I/O 性能没有满足要求是导致很多系统性能低下的原因。通常情况下，主要是两个方面的原因导致 I/O 性能瓶颈：在系统规划时没有对 I/O 性能进行估算或者估算偏差太大，导

致存储的 I/O 性能无法满足要求；其次是没有利用好数据库特性，如没有建立合适的索引，导致经常要做全表扫描，消耗大量 I/O 带宽，这可以通过查看 SQL 语句执行计划来加以分析。

5． 系统日志和 SQL 日志

最后，DBA 还可以通过查看系统日志来辅助分析问题。在 DM 数据库运行过程中，会将一些关键信息记录到安装目录下一级 log 目录下的名称为 dm_实例名_YYYYMM.log 的日志文件中，其中 YYYY 表示年份，MM 表示月份，该文件会记录下数据库服务启动/关闭的时间、系统关键错误如打开文件失败等。

另外，如果将 DM 数据库配置文件中的参数 SVR_LOG 设置为打开，则系统还会在刚才的 log 目录下生成名为 log_commit_NN.log 的文件，在该文件中记录了启用 SVR_LOG 之后数据库接收到的所有 SQL 语句等信息，DBA 也可以通过分析该文件来帮助解决问题。

21.2 监控系统性能(V$)

在 DM 数据库中，定义了一系列以 V$为前缀的系统动态视图（参考附录 2：动态性能视图），这些表只有表结构信息，没有数据，有时也称之为虚视图。查询动态视图时，服务器动态加载数据。在 DM 数据库运行过程中，系统动态视图提供了大量系统内部信息，便于数据库管理员监视服务器的运行状况，并根据这些信息对数据库进行调优，达到提高数据库性能的目的。用户可以通过普通 SQL 查询语句来查询动态视图信息，也可以通过图形化客户端工具 Monitor 来进行查看。

某些动态性能视图（如 V$SYSSTAT）需要 ENABLE_MONITOR、MONITOR_TIME、MONITOR_SYNC_EVENT、MONITOR_SQL_EXEC 参数打开时才会进行相关信息的收集。

21.3 数据库重演(REPLAY)

数据库重演（Database Replay）是 DM 中用来重现、定位和分析问题的一个重要手段，其基本原理是在数据库系统上捕获所有负载（记录外部客户端对服务器的请求），保存到二进制捕获文件，然后通过 DM 提供的数据库重演工具将捕获文件中的请求发送给捕获前由原始数据库备份恢复而来的重演测试系统上，从而帮助重现当时的场景。

用户可以调用系统过程 sp_start_capture 来启动捕获发往数据库的所有负载，并将该阶段收到的所有请求保存到二进制捕获文件中，然后使用 DM 提供的数据库重演客户端工具重放二进制捕获文件，再现当时真实环境的负载及运行情况，帮助进行问题跟踪和诊断。使用系统过程 sp_stop_capture 可以停止捕获。

Replay 的使用必须指定必要的执行参数，其调用格式为：

```
格式：DREPLAY  KEYWORD=value
例程：DREPLAY  SERVER=LOCALHOST:5236 FILE=.\test.cpt
必选参数：FILE
--如果需要获取帮助信息，可以调用 dreplay help，屏幕将显示如下信息：
关键字              说明（默认）
------------------------------------------------------------
SERVER              需要连接的服务器格式 SERVER:PORT(LOCALHOST:5236)
FILE                捕获文件及路径
HELP                打印帮助信息
```

21.4 检查数据物理一致性

DM 数据库提供了用于检查数据物理一致性的工具 dmdbchk。在数据库服务器正常关闭的情况下，可以使用 dmdbchk 对数据文件完整性进行校验，检验的内容主要包括：数据文件大小的校验；索引合法性校验；数据页面校验；系统对象 ID 校验等。在检验完毕后，dmdbchk 会在当前目录下生成一个名为 dbchk_err.txt 的检查报告，供用户查看。

dmdbchk 的使用必须指定必要的执行参数，其调用格式为：

```
格式:: dmdbchk [ini_file_path]
例程：
dmdbchk path=d:\dmdbms\bin\dm.ini
--如果需要获取帮助信息，可以调用 dmdbchk help，屏幕将显示如下信息：
关键字            说明（默认）
------------------------------------------------------------------
PATH              dm.ini 绝对路径或者当前目录的 dm.ini
HELP              打印帮助信息
```

21.5 调整配置参数

在 DM 数据库中，很多参数都是动态的，会自动响应各种负载，但是 DBA 仍然可以调用系统过程来改变 DM 实例的运行参数，从而获得更佳的性能体验。DBA 可以在 DM 数据库运行过程中执行 SF_GET_PARA_VALUE 、 SF_GET_PARA_DOUBLE_VALUE 和 SF_GET_PARA_STRING_VALUE 这三个函数来获取系统的当前配置参数，并且可以使用 SP_SET_PARA_VALUE 和 SP_SET_PARA_DOUBLE_VALUE 过程来修改静态/动态配置参数：

1. SF_GET_PARA_VALUE (scope int, paraname varchar(256))

配置参数的值类型为数值类型时使用该函数来获取当前值。SCOPE 参数为 1 表示获取 INI 文件中配置参数的值，为 2 表示获取内存中配置参数的值。

2. SF_GET_PARA_DOUBLE_VALUE(scope int, paraname varchar(8187))

配置参数的值类型为浮点型时使用该函数来获取当前值。SCOPE 参数为 1 表示获取 INI 文件中配置参数的值，为 2 表示获取内存中配置参数的值。

3. SF_GET_PARA_STRING_VALUE (scope int, paraname varchar(8187))

配置参数的值为字符串类型时用该系统函数来获取当前值。SCOPE 参数为 1 表示获取 INI 文件中配置参数的值，为 2 表示获取内存中配置参数的值。

4. SP_SET_PARA_VALUE (scope int, paraname varchar(256), value int64)

该过程用于修改整型静态配置参数和动态配置参数。SCOPE 参数为 1 表示在内存和 INI 文件中都修改参数值，此时只能修改动态的配置参数。参数为 2 表示只在 INI 文件中修改配置参数，此时可用来修改静态配置参数和动态配置参数。当 SCOPE 等于 1，试图修改静态配置参数时服务器会返回错误信息。只有具有 DBA 角色的用户才有权限调用 SP_SET_PARA_VALUE。

5. SP_SET_PARA_DOUBLE_VALUE (scope int, paraname varchar(8187),

alue double)

该过程用于修改浮点型静态配置参数和动态配置参数。SCOPE 参数为 1 表示在内存和 INI 文件中都修改参数值，此时只能修改动态的配置参数。参数为 2 表示只在 INI 文件中修改配置参数，此时可用来修改静态配置参数和动态配置参数。当 SCOPE 等于 1，试图修改静态配置参数时服务器会返回错误信息。只有具有 DBA 角色的用户才有权限调用 SP_SET_PARA_DOUBLE_VALUE。

6. SF_SET_SYSTEM_PARA_VALUE (paraname varchar(256), value int64\double\varchar（256）, deferred int, scope int64)

该过程用于修改系统整型、double、varchar 的静态配置参数或动态配置参数。DEFERRED 参数，为 0 表示当前 session 修改的参数立即生效，为 1 表示当前 session 不生效，后续再生效，默认为 0。SCOPE 参数为 1 表示在内存和 INI 文件中都修改参数值，此时只能修改动态的配置参数。参数为 2 表示只在 INI 文件中修改配置参数，此时可用来修改静态配置参数和动态配置参数。只有具有 DBA 角色的用户才有权限调用 SF_SET_SYSTEM_PARA_VALUE。

DM 的动态 INI 参数分为系统级和会话级两种级别。会话级参数在服务器运行过程中被修改时，之前创建的会话不受影响，只有新创建的会话使用新的参数值。

1. SF_SET_SESSION_PARA_VALUE (paraname varchar(8187), value bigint)

设置某个会话级 INI 参数的值，设置的参数值只对本会话有效。

2. SP_RESET_SESSION_PARA_VALUE (paraname varchar(8187))

重置某个会话级 INI 参数的值,使得这个 INI 参数的值和系统 INI 参数的值保持一致。

3. SF_GET_SESSION_PARA_VALUE (paraname varchar(8187))

获得当前会话的某个会话级 INI 参数的值。

需要注意的是，在对参数进行调整前，DBA 应该深刻理解配置参数中每个参数的含义和对系统的影响，避免由于错误的调整导致影响整个系统对外提供正常服务。对于一些关键业务，在实际调整前，建议在测试系统上先进行试验，验证通过后再在生产系统上进行调整。

21.6 优化数据库布局

数据库的布局直接影响整个系统的 I/O 性能。通常情况下，DBA 应该遵循下述原则：

1. 日志文件放在独立的物理磁盘上，保持与数据文件分开存储；
2. 预先估算并分配好磁盘空间，避免运行过程中频繁扩充数据文件；
3. 系统中不同表空间尽量分布在不同的磁盘上，这样当数据分布在多个表空间时，可以充分利用不同磁盘的并行 I/O 能力；
4. 对于分区表，无论是水平分区还是垂直分区，也尽可能将不同的分区放到不同的表空间；
5. 对于分析型应用，数据库的页大小和簇大小都可以考虑取最大值，并且在采用列存储的情况下，应该尽可能让每列存放在独立的表

第22章 动态管理/性能视图

22.1 理解动态管理视图

达梦数据库中的动态性能视图能自动收集数据库中的一些活动信息，系统管理员根据这些信息可以了解数据库运行的基本情况，为数据库的维护和优化提供依据。动态性能视图信息也是数据库中数据字典的一部分，与我们平常所说的数据字典不同的是，平常意义上的数据字典是指静态数据字典信息，也即用户访问数据字典信息时，内容不会发生改变，而动态视图信息是随着数据库的运行随时更改，具有一定的即时性。

系统管理员为了更好地了解数据库的一些运行时信息，可以查询动态视图表。首先系统管理员需要知道达梦数据库中提供了多少动态视图，有哪些类型动态视图，以及这些动态视图的用途是什么。关于这些内容可以参考附录 2。

动态视图表与静态字典信息表命名方式不同，静态字典表一般以 SYS 为前缀，如系统用户表 SYSUSERS，而动态视图则以 V$为前缀，如 V$DM_INI。

22.2 使用动态管理视图

在 DM7 中，动态视图提供的系统信息主要分为以下几个方面：

1． 系统信息

包括数据库版本、实例、统计信息、资源限制信息、进程信息、全局索引 IID 信息、事件信息；涉及的动态视图有 V$SESSIONS、V$INSTANCE、V$RESOURCE_LIMIT、V$PROCESS、V$IID、V$SYSSTAT 等。

例如查看数据库中实例信息。

```
SELECT * FROM V$INSTANCE;
```

结果：

```
NAME     HOST_NAME       SVR_VERSION  DB_VERSION         START_TIME
STATUS$ MODE$  OGUID       RAC_SEQNO          RAC_ROLE
 1  DMSERVER PC-201103131435 DM DATABASE SERVER
V7.1.2.180-BUILD(2013.09.13-34445TRUNC)
   DB VERSION: 0X70008 2013-09-16 09:10:38       OPEN    NORMAL  0   0      SLAVE
```

2． 存储信息

包括数据库信息、表空间信息、数据文件信息、日志相关信息；涉及的动态视图有 V$DATAFILE、V$DATABASE、V$TABLESPACE、V$HUGE_TABLESPACE、V$RLOGFILE 等。

例如查询表空间信息。

```
SELECT * FROM V$TABLESPACE;
```

结果：

```
ID  NAME    CACHE   TYPE$   STATUS$ MAX_SIZE    TOTAL_SIZE  FILE_NUM
ENCRYPT_NAME    ENCRYPTED_KEY
0   SYSTEM      1   0   0   1408    1   <NULL>  <NULL>
1   ROLL        1   0   0   16384   1   <NULL>  <NULL>
```

```
4   MAIN        1   0   0   16384   1   <NULL>  <NULL>
5   SYSAUX      1   0   0   16384   1   <NULL>  <NULL>
3   TEMP        2   0   0   1280    1   <NULL>  <NULL>
```

3． 内存管理信息

包括内存池使用情况、BUFFER 缓冲区信息、虚拟机信息、虚拟机栈帧信息；涉及的动态视图有 V$VPOOL、V$VMS、V$STKFRM、V$BUFFERPOOL、V$BUFFER_LRU_FIRST、V$BUFFER_UPD_FIRST、V$BUFFER_LRU_LAST、V$BUFFER_UPD_LAST、V$RLOGBUF、V$COSTPARA 等。

例如查询内存池 BUFFERPOOL 的页数、读取页数和命中率信息。

```
SELECT NAME,N_PAGES,N_LOGIC_READS,RAT_HIT FROM V$BUFFERPOOL;
```

结果：

```
行号    NAME      N_PAGES       N_LOGIC_READS         RAT_HIT
1       KEEP      1024          0                  1.0000000000E+000
2       RECYCLE   8192          52                 9.8113207547E-001
3       NORMAL    1280          1772               9.2726321298E-001
4       NORMAL    8960          7                  7.0000000000E-001
```

4． 事务信息

包括所有事务信息、当前事务可见的事务信息、事务锁信息（TID 锁、对象锁）、回滚段信息、事务等待信息；涉及的动态视图有 V$TRX、V$TRXWAIT、V$TRX_VIEW、V$LOCK、V$PURGE 等。

例如查询系统中上锁的事务、锁类型，以及表 ID 信息。

```
SELECT TRX_ID,LTYPE,LMODE,TABLE_ID,ROW_IDX FROM V$LOCK;
```

结果：

```
行号  TRX_ID    LTYPE       LMODE     TABLE_ID      ROW_IDX
1       932     OBJECT        IX       1149         0
2       933     OBJECT        IX       1149         0
3       932     OBJECT        IS       -1           0
4       933     OBJECT        IS       -1           0
5       932     OBJECT        IS       4            0
6       933     OBJECT        IS       4            0
7       933     OBJECT        IS       33555437     0
8       933     TID       X       0            933
9       932     TID       X       0            932
10      933     TID       S        0           932
```

5． 线程信息

包括所有活动线程信息、线程作业信息、线程锁信息、线程的资源等待信息；涉及的动态视图有 V$THREADS、V$LATCHES 等。

例如查看系统中所有活动的线程信息。

```
SELECT * FROM V$threads;
```

结果：

```
行号       ID         NAME       START_TIME      THREAD_DESC


-----------------------------------------------------------------------
```

```
    1   9724              DM_IO_THD     2017-06-28 09:26:46.000000
              IO THREAD

    2    11164             DM_IO_THD     2017-06-28 09:26:46.000000
              IO THREAD

    3    3892              DM_CHKPNT_THD  2017-06-28 09:26:46.000000
              FLUSH CHECKPOINT THREAD
    4    9336              DM_REDOLOG_THD 2017-06-28 09:26:46.000000
              REDO LOG THREAD,USED TO FLUSH LOG

    5   3492              DM_RAPPLY_THD  2017-06-28 09:26:46.000000
              LOG APPLY THREAD WHICH RECEIVE REDO-LOGS FROM PRIMARY SITE BY STANDBY
     SITE
```

6． 历史模块

包括 SQL 历史信息、SQL 执行节点历史信息、检查点历史信息、命令行历史信息、线程等待历史信息、死锁历史信息、回滚段历史信息、运行时错误历史信息、DMSQL 程序中执行 DDL 语句的历史信息、返回大数据量结果集的历史信息、所有活动过线程的历史信息；涉及的动态视图有 V$CKPT_HISTORY、V$CMD_HISTORY、V$DEADLOCK_HISTORY、V$PLSQL_DDL_HISTORY、V$PRE_RETURN_HISTORY、V$RUNTIME_ERR_HISTORY、V$WAIT_HISTORY、V$WTHRD_HISTORY、V$SQL_HISTORY、V$SQL_NODE_HISTORY、V$SQL_NODE_NAME 等。

例如查询系统执行的 SQL 历史信息。

```
    SELECT SESS_ID,TOP_SQL_TEXT,TIME_USED FROM V$SQL_HISTORY;
```

结果：

```
    SESS_ID        TOP_SQL_TEXT                 TIME_USED
1   187744368      INSERT INTO T1 VALUES(5,0);    21707
```

7． 缓存信息

包括 SQL 语句缓存、执行计划缓存、结果集缓存、字典缓存信息、字典缓存中的对象信息、代价信息；涉及的动态视图有 V$CACHEITEM、V$SQL_PLAN、V$CACHERS、V$CACHESQL、V$DICT_CACHE_ITEM、V$DICT_CACHE 等。

例如查看字典缓存的信息。

```
    SELECT * FROM V$DICT_CACHE;
```

结果：

```
    ADDR       POOL_ID      TOTAL_SIZE        USED_SIZE          DICT_NUM
1   0X0B56F070     1          5242880           113530            36
```

8． 会话信息

包括连接信息、会话信息；涉及的动态视图有 V$CONNECT、V$STMTS、V$SESSIONS 等。

例如查看会话信息。

```
    SELECT SESS_ID,SQL_TEXT,STATE,CREATE_TIME,CLNT_HOST FROM V$SESSIONS;
```

结果：

```
    SESS_ID        SQL_TEXT       STATE          CREATE_TIME        CLNT_HOST
```

```
1    187744368      SELECT SESS_ID,SQL_TEXT,STATE,CREATE_TIME,CLNT_HOST FROM
V$SESSIONS;    ACTIVE    2011-09-19 19:20:38.000000     FREESKYC-FB8846
```

9. **捕获信息**

捕获信息涉及的视图为 V$CAPTURE。

例如查看捕获信息。

```
SELECT * FROM V$CAPTURE;
```

结果：

```
STATE   VERSION    MSG_NUM   FILE_PATH     INIT_TIME   DURATION
FLUSH_BUF_NUM          FREE_BUF_NUM
1       1       0       0      1970-01-01 08:00:00     0       0       0
```

查看动态视图可以不单单只查询一个动态视图表，还可利用动态视图表之间的联系得到更多想要的信息。

例如，系统管理员如果要对一条 SQL 语句进行调优，需要知道每个执行节点花费了多少时间，查询 V$SQL_NODE_NAME 可以知道执行节点的名字，查询 V$SQL_NODE_HISTORY 可以查询到每个执行节点的时间，通过两个动态视图表的执行节点类型 TYPE$字段做等值连接。

如执行一条 SQL 语句，然后查询其执行节点所花费时间，假设其执行 ID（EXEC_ID）为 4。

```
SELECT * FROM t1 WHERE c1 = (SELECT d1 FROM t2 WHERE c2 = d2);
```

通过视图 V$SQL_NODE_NAME 与 V$SQL_NODE_HISTORY 视图查询结点执行时间：

```
SELECT N.NAME, TIME_USED, N_ENTER FROM V$SQL_NODE_NAME N, V$SQL_NODE_HISTORY
H WHERE N.TYPE$ = H.TYPE$ AND EXEC_ID = 4;
```

结果为：

```
        NAME    TIME_USED     N_ENTER
1       CSCN2   381     6
2       CSCN2   250     3
3       NLI2    52      14
4       SLCT2   102     8
5       HAGR2   11831   6
6       PRJT2   32      6
7       CSCN2   272     3
8       HI3     19309   9
9       PRJT2   29      6
10      NSET2   120     4
11      DLCK    23      2
```

根据结果可以看到执行计划中各执行节点花费的时间，进而对 SQL 语句进行分析和改写。

第23章 查询优化

数据库执行一条语句有多种方式，为了选择最优的执行方式，产生了查询优化器。查询优化器分析语句运行时的所有因素，选择最优的方式去执行，提高了查询效率。因此，查询优化是数据库执行 SQL 语句的重要过程，决定了数据库的查询性能。

23.1 优化目标

达梦数据库查询优化器的优化目标为最快响应时间。通过设置参数 FIRST_ROWS 来决定优先返回多少条记录给用户，而不需要等待全部结果确定后再输出，FIRST_ROWS 设置范围为 1~1000，单位为行。例如：FIRST_ROWS = 10，意思是查询出 10 条结果就立即返回给用户。可以根据实际情况，调整参数值。

23.2 查询优化器

查询优化器通过分析可用的执行方式和查询所涉及的对象统计信息来生成最优的执行计划。此外，如果存在 HINT 优化提示，优化器还需要考虑优化提示的因素。

查询优化器的处理过程包括：

1. 优化器生成所有可能的执行计划集合；
2. 优化器基于字典信息的数据分布统计值、执行语句涉及到的表、索引和分区的存储特点来估算每个执行计划的代价。代价是指 SQL 语句使用某种执行方式所消耗的系统资源的估算值。其中，系统资源消耗包括 I/O、CPU 使用情况、内存消耗等；
3. 优化器选择代价最小的执行方式作为该条语句的最终执行计划。

优化器所做的操作有：查询转换、估算代价、生成计划。

23.2.1 查询转换

查询转换是指把经过语法、语义分析的查询块之间的连接类型、嵌套关系进行调整，生成一个更好的查询计划。常用的查询转换技术包括过滤条件的下放、相关子查询的去相关性。

1. 过滤条件下放：在连接查询中，把部分表的过滤条件下移，在连接之前先过滤，可以减少连接操作的数据量，提升语句性能；
2. 相关子查询的去相关性：把与子查询相关的外表与内表采用半连接的方式执行，放弃默认采取的嵌套连接方式，对性能有较大提升。

23.2.2 估算代价

估算代价是指对执行计划的成本进行估算。执行节点之间的代价值相关性较强，一个执行节点的代价包括该节点包含的子节点代价。代价衡量指标包括选择率、基数、代价。

选择率是指满足条件的记录占总记录数的百分比。记录集可以是基表、视图、连接或分

组操作的结果集。选择率和查询谓词相关，如 name = '韩梅梅'；或者是谓词的连接，如 name ='韩梅梅' and no ='0123'。一个谓词可以看作是一个过滤器，过滤掉结果集中不满足条件的记录。选择率的范围从 0 到 1。其中，0 表示没有记录被选中，1 表示行集中所有记录都被选中。

如果没有统计信息，则优化器依据过滤条件的类型来设置对应的选择率。例如，等值条件的选择率低于范围条件选择率。这些假定是根据经验值，认为等值条件返回的结果集最少。

如果有统计信息，则可以使用统计信息来估算选择率。例如，对于等值谓词（name ='韩梅梅'），如果 name 列有 N 个不同值，那么，选择率是 N 分之一。

基数是指整个行集的行数，该行集可以是基表、视图、连接或分组操作的结果集。

代价表示资源的使用情况。查询优化器使用磁盘 I/O、CPU 占用和内存使用作为代价计算的依据，所以代价可以用 I/O 数、CPU 使用率和内存使用一组值来表示。所有操作都可以进行代价计算，例如扫描基表、索引扫描、连接操作或者对结果集排序等。

访问路径决定了从一个基表中获取数据所需要的代价。访问路径可以是基表扫描、索引扫描等。在进行基表扫描或索引扫描时，一次 I/O 读多个页，所以，基表扫描或索引全扫描的代价依赖于表的数据页数和多页读的参数值。二级索引扫描的代价依赖于 B 树的层次、需扫描的叶子块树以及根据 rowid 访问聚集索引的记录数。

连接代价是指访问两个连接的结果集代价与连接操作的代价之和。

23.2.3 生成计划

生成计划指计划生成器对给定的查询按照连接方式、连接顺序、访问路径生成不同的执行计划，选择代价最小的一个作为最终的执行计划。

连接顺序指不同连接项的处理顺序。连接项可以是基表、视图、或者是一个中间结果集。例如表 t1、t2、t3 的连接顺序是先访问 t1，再访问 t2，然后对 t1 与 t2 做连接生成结果集 r1，最后把 t3 与 r1 做连接。一个查询语句可能的计划数量是与 FROM 语句中连接项的数量成正比的，随着连接项的数量增加而增加。

23.3 数据访问路径

访问路径指从数据库中检索数据的方法。一般情况下，索引访问用于检索表的小部分数据，全表扫描用于访问表的大部分数据。OLTP 应用中，一般使用索引访问路径，因为 OLTP 中包含了许多高选择率的 SQL 语句。而决策支持系统则倾向于执行全表扫描来获取数据。从数据库中定位和检索数据的方法有：全表扫描、聚集索引扫描、二级索引扫描等。

全表扫描是指从基表中检索数据时，扫描该表中所有的数据。全表扫描方式适合检索表中大部分数据，这时比索引扫描更加有效率。

索引扫描是指通过指定语句中的索引列进行遍历来检索表中的数据。索引扫描是从基于一列或多列的索引中检索数据。索引不仅包含索引值，还包含对应表中数据的 ROWID。如果需要访问的不是索引列，这时需要通过 ROWID 或聚集索引来找到表中的数据行。

索引扫描包含聚集索引扫描和二级索引扫描。由于在聚集索引中，包含了表中所有的列值，所以检索数据时只需要扫描这一个索引就可以得到所有需要的数据。如果是二级索引，由于只包含索引列以及对应的 ROWID，如果查询列不在二级索引中则还需要扫描聚集索引来得到所需要的数据。

查询优化器选择访问路径基于以下几个因素：

1．执行语句中可能的访问路径；

2．估算每条执行路径的代价。为了选择一个访问路径，优化器首先会通过检查语句中的 FROM 子句和 WHERE 子句中的条件表达式来决定哪一个访问路径可以使用。优化器会根据可用的访问路径生成可能的执行计划集合，然后使用索引、列和表的统计信息来估算每个计划的代价。最后，优化器选择最小代价的那个执行计划。

影响优化器选择访问路径的因素有语句中的提示（HINT）和统计信息。用户可以在执行的语句中使用 HINT 来指定访问路径。而统计信息会根据表中数据的分布情况决定采用哪个访问路径会产生最小的代价。

23.4 连接

查询语句中 FROM 子句包含多个表时，我们称为连接查询。如 SELECT * FROM t1,t2 就是连接查询。

生成连接查询的执行计划，需要考虑三方面因素：

1． 访问路径

对于每张表采用何种方式来获取数据。例如：全表扫描、索引扫描等。

查询优化器会估算每种扫描方式的代价，选择代价较小的访问路径。

2． 连接方式

确定两张表之间采用哪种连接方式。例如：哈希连接、嵌套连接、归并连接、外连接。

等值连接条件一般会选择哈希连接；非等值连接条件会采用嵌套连接；连接列均为索引列时，会采用归并连接。

1） 嵌套连接：两张表进行非等值连接时会选择嵌套连接。相当于两张表进行笛卡尔集操作。此时，优化器会选择一张代价较小的表作为外表(驱动表)，另一张表作为内表，外表的每条记录与内表进行一次连接操作。

2） 哈希连接：两张表进行等值连接时会选择哈希连接。以一张表的连接列为哈希键，构造哈希表，另张表的连接列进行哈希探测，找到满足条件的记录。由于哈希命中率高，因此，在大数据量情况下，哈希连接的效率较高。哈希连接的代价是建立哈希表和哈希探测的代价。

3） 归并连接：两张表的连接列均为索引列，则可以按照索引顺序进行归并，一趟归并就可以找出满足条件的记录。如果查询列也属于索引列的子集，则归并连接只需扫描索引，会有更好的性能表现。在两表连接条件不是等值（如<，<=，>，>=）情况下时，归并排序连接很有用。

4） 外连接：外连接分为左外连接、右外连接、全外连接。作为外表的数据会全部返回，如果没有与外表匹配的记录，则填充 NULL 值。右外连接与左外连接的处理过程类似，只是外表不同，一个是左表，一个是右表。全外连接是进行左外连接和右外连接，返回两次外连接的 union 结果集。

例 1：左外连接：

```
SELECT * FROM t1 LEFT OUTER JOIN t2 ON t1.c1=t2.d1;
```

例 1 中 t1 表为外表(左表)，如果 t2 表中不存在与 t1.c1 相等的记录，则 t2 表的该行记录用 NULL 填充。右外连接与左外类似。

例 2：全外连接

```
SELECT * FROM t1 FULL OUTER JOIN t2 ON t1.c1=t2.d1;
```

例 2 中分别以 t1 为左表进行左外和右外连接，两次结果进行 union，返回最终

结果。

子查询会转换成半连接。共有四种半连接方式：哈希半连接、索引半连接、嵌套半连接、归并半连接。等值连接条件会选择哈希\索引\归并半连接，非等值连接条件会选择嵌套半连接。

1） 哈希半连接：以外表的连接列为 KEY 构造哈希表，内表的连接列进行探测来查找满足连接条件的记录；

2） 索引半连接：如果子查询的连接列为索引前导列，可采用索引半连接。处理过程为外表的数据对子查询使用索引查找，返回满足条件的记录；

3） 归并半连接：如果相关子查询的连接条件列均为索引列，可采用归并半连接。按照索引顺序，对外表、内表进行同步扫描，返回满足条件的记录；

4） 嵌套半连接：如果连接条件为非等值，可转换为嵌套半连接。处理过程为外表的每条记录去遍历内表，返回满足条件的记录。

3． 连接顺序

当超过 2 张表进行连接时，就需要考虑表之间的连接顺序。不合适的连接顺序对执行效率有较大影响。一般原则是，经过连接可以产生较小结果集的表优先处理。

一个连接查询通常会对应多个执行计划，查询优化器会根据优化规则、代价估算挑选最优的执行计划。

23.5 统计信息

对象统计信息描述数据是如何在数据库中存储的。统计信息是优化器的代价计算的依据，可以帮助优化器较精确地估算成本，对执行计划的选择起着至关重要的作用。

达梦数据库的统计信息分三种类型：表统计信息、列统计信息、索引统计信息。通过直方图来表示。统计信息生成过程分以下三个步骤：

1．确定采样的数据：根据数据对象，确定需要分析哪些数据。

1） 表：计算表的行数、所占的页数目、平均记录长度

2） 列：统计列数据的分布情况

3） 索引：统计索引列的数据分布情况

2．确定采样率

根据数据对象的大小，通过内部算法，确定数据的采样率。采样率与数据量成反比。

3．生成直方图

有两种类型的直方图：频率直方图和等高直方图。根据算法分析表的数据分布特征，确定直方图的类型。频率直方图的每个桶(保存统计信息的对象)的高度不同，等高直方图每个桶的高度相同。例如，对列生成统计信息，当列值分布比较均匀时，会采用等高直方图，否则，采用频率直方图。

在执行查询时，如果数据对象存在统计信息，代价算法可以根据统计信息中的数据，比较精确地计算出操作所需花费的成本，以此来确定连接方式、对象访问路径、连接顺序，选择最优的执行计划。

用户也可以通过修改 OPTIMIZER_DYNAMIC_SAMPLING 参数值在缺乏统计信息时进行动态统计信息收集。

23.6 执行计划

执行计划是 SQL 语句的执行方式，由查询优化器为语句设计的执行方式，交给执行器去执行。在 SQL 命令行使用 EXPLAIN 可以打印出语句的执行计划。

例如：

建表和建索引语句：

```
CREATE TABLE T1(C1 INT,C2 CHAR);
CREATE TABLE T2(D1 INT,D2 CHAR);
CREATE  INDEX  IDX_T1_C1  ON  T1(C1);
INSERT INTO T1 VALUES(1,'A');
INSERT INTO T1 VALUES(2,'B');
INSERT INTO T1 VALUES(3,'C');
INSERT INTO T1 VALUES(4,'D');
INSERT INTO T2 VALUES(1,'A');
INSERT INTO T2 VALUES(2,'B');
INSERT INTO T2 VALUES(5,'C');
INSERT INTO T2 VALUES(6,'D');
```

打印执行计划：

```
EXPLAIN SELECT A.C1+1,B.D2 FROM T1 A, T2 B WHERE A.C1 = B.D1;
```

执行计划如下：

```
1   #NSET2: [0, 16, 9]
2     #PRJT2: [0, 16, 9]; EXP_NUM(2), IS_ATOM(FALSE)
3       #NEST LOOP INDEX JOIN2: [0, 16, 9]
4         #CSCN2: [0, 4, 5]; INDEX33555535(B)
5         #SSEK2: [0, 4, 0]; SCAN_TYPE(ASC), IDX_T1_C1 (A),
SCAN_RANGE[T2.D1,T2.D1]
```

这个执行计划看起来就像一棵树，执行过程为：控制流从上向下传递，数据流从下向上传递。其中，类似[0, 16, 9]这样的三个数字，分别表示估算的操作符代价、处理的记录行数和每行记录的字节数。同一层次中的操作符，如本例中的 CSCN2 和 SSEK2，由父节点 NEST LOOP INDEX JOIN2 控制它们的执行顺序。

该计划的大致执行流程如下：

1) CSCN2：扫描 T2 表的聚集索引，数据传递给父节点索引连接；

2) NEST LOOP INDEX JOIN2：当左孩子有数据返回时取右侧数据；

3) SSEK2：利用 T2 表当前的 D1 值作为二级索引 IDX_T1_C1 定位查找的 KEY，返回结果给父节点；

4) NEST LOOP INDEX JOIN2：如果右孩子有数据则将结果传递给父节点 PRJT2，否则继续取左孩子的下一条记录；

5) PRJT2：进行表达式计算 C1+1, D2；

6) NSET2：输出最后结果；

7) 重复过程 1) ~ 4)直至左侧 CSCN2 数据全部取完。

用户如果想了解更多关于操作符的知识，请查看动态视图 V$SQL_NODE_NAME，手册的附录 4 给出了常用操作符的说明。

23.6.1 自适应计划

在 DM 优化器进行代价估算时，如果子节点是一个复杂查询，可能使得对于子节点的代价估算不准确，导致最终选择的计划在执行时并非最优计划。因此，DM 引入了自适应计划机制。

将 INI 参数 OPTIMIZER_MODE 和 ADAPTIVE_NPLN_FLAG 都置为 1，启用自适应计划机制。此时，优化器会自动判断在某些情况下，复杂查询的子节点可能导致代价估算不准确，则会为该节点生成一个备用计划节点。在实际执行到该节点时，根据准确的代价信息确定是否需要采用备用计划。

例如：

建表、插入数据并建索引：

```
DROP TABLE T1;
DROP TABLE T2;

CREATE TABLE T1(C1 INT,C2 INT);
CREATE TABLE T2(D1 INT,D2 INT);

INSERT INTO T1 VALUES(1,1);
INSERT INTO T1 VALUES(1,2);
INSERT INTO T2 VALUES(1,1);
INSERT INTO T2 VALUES(2,1);
INSERT INTO T2 VALUES(3,1);
INSERT INTO T2 VALUES(4,1);
INSERT INTO T2 VALUES(5,1);
COMMIT;

CREATE INDEX IND1 ON T1(C2);
```

打印执行计划：

```
EXPLAIN SELECT * FROM T1 ,T2 ,T1 T3 WHERE T2.D1= T3.C1 AND T1.C2=T2.D2;
```

执行计划如下：

```
1   #NSET2: [1, 2, 24]
2     #PRJT2: [1, 2, 24]; exp_num(6), is_atom(FALSE)
3       #HASH2 INNER JOIN: [1, 0, 0];  KEY_NUM(1);
4         #NEST LOOP INDEX JOIN2: [1, 2, 24]
5           #ACTRL: [1, 2, 24];
6             #HASH2 INNER JOIN: [0, 5, 16];  KEY_NUM(1);
7               #CSCN2: [0, 2, 8]; INDEX33555445(T1 as T3)
8               #CSCN2: [0, 5, 8]; INDEX33555446(T2)
9           #BLKUP2: [0, 1, 0]; IND1(T1)
10            #SSEK2: [0, 1, 0]; scan_type(ASC), IND1(T1),
scan_range[T2.D2,T2.D2]
11        #CSCN2: [0, 2, 8]; INDEX33555445(T1)
```

可以看到执行计划中有一个 ACTRL 操作符，它说明优化器为这一条 SQL 语句生成了备

用计划。ACTRL 是控制备用计划转换的操作符，其上面一层 NEST LOOP INDEX JOIN2 为默认的主计划，再上面一层 HASH2 INNER JOIN 则为备用计划。ACTRL 操作符计算下层孩子节点的代价，决定采用默认主计划还是备用计划。

需要说明的是，MPP 和并行查询不支持自适应计划。

23.7 使用索引

为了提高查询效率，用户一般会在表中创建索引。查询中的条件列为索引列时，如果索引扫描代价最小，优化器就会采用索引扫描。索引扫描有多种方式，例如，索引等值查询、索引范围查询。如果查询列属于索引列的子集，则通过索引扫描就可以获得数据，否则，还需要根据 ROWID 或者 PK 在聚集索引中定位记录。

常用的索引类型有唯一索引、组合索引、函数索引。各自有不同的使用场景。

1. 条件列具有 UNIQUE 约束，则可以创建唯一索引，减少索引扫描次数；
2. 条件列是多个列，而且可以过滤掉大部分数据，可以在多个列上创建组合索引，把等值条件列作为组合索引的首列；
3. 条件列使用确定性函数(同样环境下多次执行得到相同的结果)，可以创建函数索引，会把函数值进行存储，使用方式与普通索引一样；
4. 在空间数据应用中，可以创建空间索引提高空间查询的效率。

23.8 并行查询

23.8.1 并行查询概念

倘若没有并行查询技术，一个串行执行的查询语句只能利用 CPU 或者磁盘设备中的一个，而不能利用整个计算机的处理能力。并行查询技术的出现，使得单个 SQL 语句能利用多个 CPU 和磁盘设备的处理能力。其优势在于可以通过多个线程来处理查询任务，从而提高查询的效率。

达梦数据库为具有多个 CPU 的数据库服务器提供并行查询的功能，以优化查询任务的性能。数据库服务器只有具有多个 CPU，才能使用并行执行查询操作，来提高查询任务的速度。

达梦数据库通过三个步骤来完成并行查询：首先，确定并行任务数；其次，确定并行工作线程数；最后，执行查询。并行查询相关参数见下表：

表 24.1 并行查询相关参数

参数名	缺省值	属性	说明
MAX_PARALLEL_DEGREE	1	动态，会话级	用来设置默认并行任务个数。取值范围：1~128。缺省值 1，表示无并行任务。当 PARALLEL_POLICY 值为 1 时该参数值才有效。
PARALLEL_POLICY	0	静态	用来设置并行策略。取值范围：0、1 和 2，缺省为 0。其中，0 表示不支持并行；1 表示自动并行模式；2 表示手动并行模式。
PARALLEL_THRD_NUM	10	静态	用来设置并行工作线程个数。取值范围：1~1024。

23.8.2 确定并行任务个数

当开启自动并行（PARALLEL_POLICY=1）时，参数 MAX_PARALLEL_DEGREE 生效，控制并行查询最多使用的线程数。MAX_PARALLEL_DEGREE 缺省值为 1，表示不并行。此时若指定参数对应的 HINT “PARALLEL”，则使用 HINT 值；

当开启手动并行（PARALLEL_POLICY=2）时，参数 MAX_PARALLEL_DEGREE 失效，用户需要在语句中使用此参数对应的 HINT “PARALLEL”指定语句的并行度，否则不并行。

1. 在 INI 参数中设置默认值

INI 参数 MAX_PARALLEL_DEGREE 设置最大并行任务个数。取值范围：1~128。缺省值 1，表示无并行任务，此参数仅在 PARALLEL_POLICY 值为 1 时才有效。

例如，在 INI 参数中将 MAX_PARALLEL_DEGREE 设置为 3 的格式如下：

```
MAX_PARALLEL_DEGREE     3
```

然后，使用一般的 SQL 语句查询即可执行并行查询，不需要使用 HINT。如：

```
SELECT * FROM SYSOBJECTS;
```

2. 在 SQL 语句中使用“PARALLEL”关键字特别指定

当 PARALLEL_POLICY=2 时，需要在 SQL 语句中通过“PARALLEL”HINT 指定并行度，否则不并行。若 PARALLEL_POLICY=1，则 SQL 语句中使用的“PARALLEL”HINT 总是优先于 MAX_PARALLEL_DEGREE 参数设置。

“PARALLEL”关键字的用法是在数据查询语句的 SELECT 关键字后，增加 HINT 子句来实现。

HINT 语法格式如下：

```
/*+ PARALLEL([<表名>] <并行任务个数>) */
```

例如，下面的例子中，即使已经设置了 MAX_PARALLEL_DEGREE 默认值 3，但实际使用的为 PARALLEL 指定的任务个数 4：

```
SELECT /*+ PARALLEL(4) */ * FROM SYSOBJECTS;
```

另外，每个语句中仅能设置一次并行任务个数，如果设置了多次，则以最后一次设置为准，而且任务个数在全语句中生效。

例如，下面的例子中，使用的并行任务个数为 2。

```
SELECT /*+ PARALLEL(1) *//*+ PARALLEL(2) */ * FROM SYSOBJECTS;
```

这种方式能够为单条查询语句设置额外的并行任务个数，以此来提高某些特殊查询任务的性能。

23.8.3 确定并行工作线程数

在执行并行查询任务之前，您需要指定完成该任务的并行工作线程数。值得注意的是，实际使用的线程数并非总是等于并行工作线程数。并行工作线程数是在 INI 参数中设定的，实际使用并行工作线程数是根据系统的实际状况确定的。

1. 并行工作线程数，在 INI 参数中设定

首先，使用 PARALLEL_POLICY 参数来设置并行策略。取值范围：0、1 和 2，默认值 0。其中，0 表示不支持并行；1 表示自动并行模式；2 表示手动并行模式。

当开启本地并行（PARALLEL_POLICY>0）时，使用 PARALLEL_THRD_NUM 指定本地并行查询使用的线程数，取值范围为 1~1024，缺省值为 10。需要注意的是，若 PARALLEL_POLICY=1，如果 PARALLEL_THRD_NUM=1，则按照 CPU 个数创建并行线程。

例如，设置并行策略 PARALLEL_POLICY 为 2，即手动设置并行工作线程数；同时，设置并行工作线程数 PARALLEL_THRD_NUM 为 4 个。

```
PARALLEL_POLICY                          2
PARALLEL_THRD_NUM                        4
```

当然，并非所有的查询都适合使用并行查询。大量占用 CPU 周期的查询最适合采用并行查询的功能。例如，大型表的连接查询、大量数据的聚合和大型结果集的排序等都很适合采用并行查询。对于简单查询（常用于事务处理应用程序）而言，执行并行查询所需的额外协调工作会大于潜在的性能提升。所以，数据库管理员在确定是否需要使用并行策略的时候，需要慎重。

2. 实际使用的线程数，达梦数据库会根据每个并行查询操作自动检测

实际使用线程数是数据库在查询计划执行时初始化的时候确定的。也就是说，这不需要用户去干预，而是系统根据并行任务数和实际空闲的并行工作线程数来确定的。此操作所依据的条件如下：首先，检测达梦数据库是否运行在具有多个CPU的计算机上。只有具有多个 CPU 的计算机才能使用并行查询。这是一个硬性的限制条件。其次，检测可用的空闲工作线程是否足够。并行查询到底采用多少线程数，除了跟操作的复杂程度相关外，还跟当时的服务器状态相关，如是否有足够的可用的空闲工作线程数量等。每个并行查询操作都要求一定的工作线程数量才能够执行；而且执行并行计划比执行串行计划需要更多的线程，所需要的线程数量也会随着任务个数的提高而增加。当无法满足特定并行查询执行的线程要求时，数据库引擎就会自动减少任务个数，甚至会放弃并行查询而改为串行计划。所以，即使同一个操作在不同时候可能会采用不同的线程数。

例如，即使设置并行工作线程数为4。而实际使用的线程数可能只有3个，或者更少。

23.8.4 执行查询

当以上内容确定好之后，数据库就会执行具体的查询任务。

23.8.5 使用场景

使用手动并行模式时，只需要在 INI 参数中设置好如下 2 个参数，然后执行并行 SQL 查询语句时，需手动指定当前并行任务个数。若不指定，将不使用并行。设置的 2 个参数如下：

```
PARALLEL_POLICY     2
PARALLEL_THRD_NUM    4
```

使用自动并行模式时，一般指定如下三个参数：

```
MAX_PARALLEL_DEGREE     3
PARALLEL_POLICY    1
PARALLEL_THRD_NUM    10
```

另外，当 PARALLEL_POLICY 为 0 时，即使有并行任务，也不支持并行。

然后，执行语法格式类似“SELECT * FROM SYSOBJECTS;”的并行 SQL 语句即可，本条语句使用默认并行任务数 3。

当然，如果单条查询语句不想使用默认并行任务数，可以通过在 SQL 语句中增加 HINT，通过“PARALLEL”关键字来特别指定。此时，执行的并行 SQL 语句格式为“SELECT /*+ PARALLEL(SYSOBJECTS 4) */ * FROM SYSOBJECTS;”，本条语句使用的并行任务数为 4。

23.9 查询计划重用

如果同一条语句执行频率较高，或者每次执行的语句仅仅是常量值不同，则可以考虑使用计划重用机制。避免每次执行都需要优化器进行分析处理，可以直接从计划缓存中获取已有的执行计划，减少了分析优化过程，提高执行率。

对于计划重用，达梦数据库提供了 INI 参数 USE_PLN_POOL 来控制，当置为非 0 时，会启用计划重用。

23.10 结果集重用

执行计划的生成与优化是一个非常依赖 CPU 的操作，而执行一个查询获得结果集也是一个非常消耗资源的操作。当系统连续执行两个完全相同的 SQL 语句，其执行计划和结果集很有可能是相同的，如果重新生成和执行计划，会大大浪费系统资源。这时如果使用计划重用和结果集重用，系统的响应速度可以大大提升。

结果集重用是基于计划重用的，如果查询的计划不能缓存，则其查询结果集必然不能缓存。此外，当语句的游标属性为 FORWARD ONLY 时，默认查询不会生成结果集。而参数 BUILD_FORWARD_RS 可以强制在此类查询中生成结果集，以便进行结果集重用。

可通过设置 INI 参数 RS_CAN_CACHE 来控制结果集重用。当置为 0 时表示手动模式（MANUAL），在此模式下默认不缓存查询结果集，但是 DBA 可以通过语句提示等方法指示系统对必要的查询结果集进行缓存；置为 1 时表示强制模式（FORCE），在此模式下默认缓存所有可缓存结果集，但是 DBA 也可以通过新增的配置参数以及语句提示等方法取消某些不合适的结果集缓存。

当 RS_CAN_CACHE 为 1 时，还可以通过设置 INI 参数 RS_CACHE_TABLES 和 RS_CACHE_MIN_TIME 对缓存的结果集进行限制和过滤。RS_CACHE_TABLES 指定可以缓存结果集的基表清单，只有查询涉及的所有基表全部在参数指定范围内，此查询才会缓存结果集。RS_CACHE_MIN_TIME 则指定了缓存结果集的查询语句执行时间的下限，只有实际执行时间不少于指定值的查询结果集才会缓存。

DBA 可以通过在 SQL 语句中设置 “RESULT_CACHE”或“NO_RESULT_CACHE” HINT 手动指示查询的结果集是否缓存。如：

```
select /*+ RESULT_CACHE */ id, name from sysobjects;
```

或者

```
select /*+ NO_RESULT_CACHE */ id, name from sysobjects;
```

在语句中使用 HINT 指定结果集缓存的优先级要高于 INI 中相关参数的设置。

还可以使用系统过程 SP_SET_PLN_RS_CACHE 来强制设置指定计划结果集缓存的生效及失效。这个系统过程对结果集缓存的指定高于其它所有结果集缓存的设置。

在以下情况下，DM 不支持结果缓存：

1. 必须是单纯的查询语句计划，PL 脚本中包含查询语句也不能缓存结果集。
2. 查询语句的计划本身必须是缓存的。
3. 守护环境中的备库不支持结果集缓存。
4. MPP 等集群环境下不支持结果集缓存（3、4 两点限制都是因为无法精确控制基表的数据更新时戳）。
5. 查询语句中包含以下任意一项，其结果集都不能缓存：
   1) 包含临时表；
   2) 包含序列的 CURVAL 或 NEXTVAL；
   3) 包含非确定的 SQL 函数或包方法(现有逻辑是不支持所有 SQL 函数或包方法)；

4） 包含 RAND、SYSDATE 等返回值实时变化的系统函数；
5） 包含其它的一些实时要素。

第24章 SQL 调优

24.1 简介

SQL 调优作为数据库性能调优中的最后一个环节，对查询性能产生着直接的影响。在进行正式的 SQL 调优前，用户首先要关注下列几点：

1．达梦数据库安装时的配置参数是否符合应用场景需求；

2．达梦数据库的 INI 配置文件中各项参数是否已经处于最优配置；

3．应用系统中数据库设计是否合理。

本章将介绍定位高负载的 SQL 语句的方法，利用自动 SQL 调整功能进行优化，以及如何开发有效的 SQL 语句和使用优化器提示来影响执行计划。

24.2 调优目标

SQL 调优的整体目标是使用最优的执行计划，这意味着 IO 以及 CPU 代价最小。具体而言调优主要关注下列方面：

- **表扫描**

如果计划中对某大表使用了全索引扫描，那么用户需要关注是否存在着该表的某个查询条件使得过滤后可以淘汰至少一半的数据量。通过添加相应的索引，全索引扫描可能被转换为范围扫描或等值查找。添加的二级索引可以包含该表上所有被选择项以避免 BLKUP2 操作符的查找操作带来的第二次 IO 开销，但无疑这会增加二级索引的大小。用户需权衡二者的利弊以选择正确的处理方式。

- **连接操作的顺序和类型**

多表连接时，不同的连接顺序会影响中间结果集数量的大小，这时调优的目标就是要找到一种能使中间结果保持最小的连接顺序。

对于给定的一个连接或半连接，DM7 可以用 HASH 连接、嵌套循环连接、索引连接或者是归并连接实现。通过分析表的数据量大小和索引信息，SQL 调优目标是选择最适宜的操作符。

对半连接而言，HASH 连接还可细分为左半 HASH 和右半 HASH。用户可以通过始终对数据量小的一侧建立 HASH 来进行调优。

- **分组操作**

分组操作往往要求缓存所有数据以找到属于同一组的所有数据，在大数据量情况下这会带来大量的 IO。用户应该检查 SQL 查询和表上索引信息，如果可以利用包含分组列的索引，那么执行计划就会使用排序分组从而不用缓存中间结果。

24.3 确定高负载的 SQL

在打开监控开关（ENABLE_MONITOR=1、MONITOR_TIME=1）后，可以通过查询动态视图 V$LONG_EXEC_SQLS 或 V$SYSTEM_LONG_EXEC_SQLS 来确定高负载的 SQL 语句。前者显示最近 1000 条执行时间较长的 SQL 语句，后者显示服务器启动以来执行时间最长的 20 条 SQL 语句。例如：

```
SELECT * FROM V$LONG_EXEC_SQLS;
```

或者

```
SELECT * FROM V$SYSTEM_LONG_EXEC_SQLS;
```

24.4 自动 SQL 调整

使用查询优化向导工具，输入需要进行调整的 SQL 语句，向导工具将在分析完执行计划后给出推荐索引的提示。用户只需按提示建立相应索引即可。

24.5 开发有效的 SQL 语句

SQL 语言是一种相当灵活的结构化查询语言。用户可以利用多种不同形式的查询语句完成相同的查询功能。为了使执行效率达到最优，用户需要参考以下原则以开发出有效的 SQL 语句：

1．避免使用 OR 子句

OR 子句在实际执行中会被转换为类似于 UNION 的查询。如果某一个 OR 子句不能利用上索引则会使用全表扫描造成效率低下，应避免使用。

如果 OR 子句都是对同一列进行过滤，用户可以考虑使用 IN VALUE LIST 的过滤形式。如：

```
SELECT ... WHERE CITY = 'SHANGHAI' OR CITY = 'WUHAN' OR CITY = 'BEIJING';
```

调整为

```
SELECT ... WHERE CITY IN( 'SHANGHAI','WUHAN','BEIJING');
```

2．避免使用困难的正则表达式

在 SQL 语言中，LIKE 关键字支配通配符匹配，含通配符的表达式被称为正则表达式。有的正则表达式可以自动优化为非匹配的。例如：a LIKE 'L%'可以优化为 a>='L' AND a <'M',这样就可以用到 a 上的索引。即使没有索引，转换后的比较也更快。再如：a LIKE 'LM_'可以转化为 a>='LM' AND a<'LN' AND a LIKE 'LM_'。虽然仍然包含着通配符匹配，但大大缩小了匹配的范围。

所谓困难的正则表达式是指开头和结尾都为通配符的正则表达式，如'_L%'、'%L_'，优化器没办法缩小它们的匹配范围，也不可能用到索引而必须使用全表扫描。因此要尽可能避免这样的正则表达式。

如果仅仅是开头为通配符，用户可以在列 a 上建立 REVERSE(a)这样一个函数索引，利用函数索引反转待匹配项从而使用函数索引进行范围扫描。

3．灵活使用伪表(SYSDUAL)

首先可以利用伪表进行科学计算，执行语句 SELECT 3*4 FROM SYSDUAL，则可以得到结果 12；

其次，在某些方面使用 SYSDUAL 可提高效率。例如：查询过程中要判断表 t1 中是否有满足 condition1 条件的记录存在，可执行以下语句：

```
SELECT COUNT(*) INTO x FROM t1 WHERE condition1;
```

然后根据变量 x 的取值来判断。但是当 t1 非常大时该语句执行速度很慢，而且由于不知道 SELECT 返回的个数，不能用 SELECT *代替。事实上这个查询可以利用伪表来完成：

```
SELECT 'A' INTO y FROM SYSDUAL
WHERE EXISTS (SELECT 1 FROM t1 WHERE condition1);
```

判断 y 值，如等于'A'则 T1 中有记录。调整后的语句执行速度明显比上一句高。

另外，在 DM7 的语法里是可以省略 FROM 子句的，这时系统会自动加上 FROM SYSDUAL。

因此前面的科学计算例子可以简化为 SELECT 3*4;

4．SELECT 项避免'*'

除非用户确实要选择表中所有列，否则 SELECT *这种写法将让执行器背上沉重的负荷。因为每一列的数据不得不自下往上层层向上传递。不仅仅如此，如果用户查询的是垂直分区表，那么更大的麻烦在于垂直分区表的所有子表都要参与连接操作；如果用户查询的是列存储表，那么列存储所带来的 IO 优势将损耗殆尽。

任何时候，用户都要了解表结构和业务需求，小心地选择需要的列并一一给出名称，避免直接用 SELECT *。

5．避免功能相似的重复索引

索引并非越多越好。抛开优化器面对众多索引逐一试探所耗费的时间不谈，如果表上增删改操作频繁，那么索引的维护将会成为大麻烦，尤其是函数索引的计算开销更不能忽略。

6．使用 COUNT(*)统计结果行数

如果对单表查询 COUNT(*)且没有过滤条件，那么 DM7 优化器会直接读取相关索引中存储的行数信息，加以回滚段中其他事务插入或删除元组的行数修正，迅速地给出最终结果而避免对实际数据的读取。相比之下，COUNT(列名)会对数据进行读操作，执行效率远低于 COUNT(*)。

即使查询中含有过滤条件，由于 DM7 特有的批处理方式，COUNT(*)依旧快于其他写法。这是因为 COUNT(*)无需取得行的具体值而仅仅需要行数这一信息。

需要额外说明的是，COUNT(*)会将 NULL 值计算在内而 COUNT(列名)是不包含 NULL 值的，因此用户要结合应用场景决定是否可以使用 COUNT(*)。

7．使用 EXPLAIN 来查看执行计划

在查询语句或者插入、删除、更新语句前增加 EXPLAIN 关键字，DM7 将显示其执行计划而无需实际执行它。查阅 V$SQL_NODE_NAME 表中每个操作符的含义，用户可以很方便且直观地了解数据如何被处理及传递。如果启用了统计信息收集，那么对照执行计划和对动态视图 V$SQL_NODE_HISTORY，V$SQL_NODE_NAME 的查询结果，用户就可以知道在实际执行中每一个操作符执行的时间，进而找出性能瓶颈。

8．UNION 和 UNION ALL 的选择

UNION 和 UNION ALL 的区别是前者会过滤掉值完全相同的元组，为此 UNION 操作符需要建立 HASH 表缓存所有数据并去除重复，当 HASH 表大小超过了 INI 参数指定的限制时还会做刷盘。

因此如果应用场景并不关心重复元组或者不可能出现重复，那么 UNION ALL 无疑优于 UNION。

9．优化 GROUP BY ... HAVING

GROUP BY 最常见的实现有 HASH 分组（HAGR）和排序分组（SAGR）。前者需要缓存中间结果；如果用户在 GROUP BY 的列上建立索引，那么优化器就会判断并可能使用上该索引，这时的 GROUP BY 就会变为 SAGR。

HAVING 是分组后对结果集进行的过滤，如果过滤条件无关集函数操作，用户可以考虑将过滤条件放在 WHERE 而不是 HAVING 中。DM7 优化器会判断并自动转换部分等效于 WHERE 的 HAVING 子句，但显式地给出最佳 SQL 语句会让优化器工作得更好。

10．使用优化器提示（HINT）

利用经验对优化器的计划选择进行调整，HINT 是 SQL 调整不可或缺的一步。我们将在下一节给出详细说明。

24.6 使用优化器提示

DM 查询优化器采用基于代价的方法。在估计代价时，主要以统计信息或者普遍的数据分布为依据。在大多数情况下，估计的代价都是准确的。但在一些比较特殊的场合，例如缺少统计信息，或统计信息陈旧，或抽样数据不能很好地反映数据分布时，优化器选择的执行计划不是“最优”的，甚至可能是很差的执行计划。

开发人员和用户对于数据分布是很清楚的，他们往往能知道 SQL 语句按照哪种方法执行会最快。在这种情况下，用户可以提供一种方法，指示优化器按照固定的方法去选择 SQL 的执行计划。

DM 把这种人工干预优化器的方法称为 HINT，它使优化器根据用户的需要来生成指定的执行计划。如果优化器无法生成相应的执行计划，该 HINT 将会被忽略。

HINT 的常见格式如下所示：

```
SELECT /*+ HINT1 [HINT2]*/  列名 FROM 表名 WHERE_CLAUSE ;
UPDATE 表名 /*+ HINT1 [HINT2]*/ SET 列名 =变量 WHERE_CLAUSE ;
DELETE FROM 表名  /*+ HINT1 [HINT2]*/ WHERE_CLAUSE ;
```

需要注意的是：如果 HINT 的语法没有写对或指定的值不正确，DM 并不会报错，而是直接忽略 HINT 继续执行。

可通过 V$HINT_INI_INFO 动态视图查询 DM 支持的 HINT。HINT 参数分为两类，HINT_TYPE 为“OPT”表示分析阶段使用的参数；HINT_TYPE 为“EXEC”表示运行阶段使用的参数，运行阶段使用的参数对于视图无效。

24.6.1 索引提示

1. 使用索引

目前 DM7 提供的 HINT 为表索引的选择 HINT，它指示使用指定索引进行数据检索。

语法：

```
表名 + INDEX + 索引名
或
/*+ INDEX (表名[,] 索引名) {INDEX (表名[,] 索引名)} */
```

一个语句中最多指定 8 个索引。在后一种语法格式中，如果查询中给出了表的别名那么必须使用别名。

假设表 t1 上 id 和 name 列上都存在着单列索引。

```
--数据准备
DROP TABLE T1 CASCADE;
CREATE TABLE T1 (ID INTEGER,NAME VARCHAR(128));
CREATE INDEX IDX_T1_ID ON T1(ID);
CREATE INDEX IDX_T1_NAME ON T1(NAME);
```

例 1 在查询语句中指定索引。

```
SELECT * FROM T1 WHERE ID > 2011 AND NAME < 'XXX';
```

如果 ID 列上能过滤更多数据，建议指示用索引 IDX_T1_ID。

```
SELECT * FROM T1 INDEX IDX_T1_ID WHERE ID > 2011 AND NAME < 'XXX';
```

或

```
SELECT /*+INDEX(T1, IDX_T1_ID) */ * FROM T1 WHERE ID > 2011 AND NAME < 'XXX';
```

例 2 当有多个索引时，要指定使执行计划最优的。

```
SELECT * FROM T1 WHERE ID > 2011 AND NAME < 'XXX' ORDER BY NAME;
```

考虑到后面的 NAME 列排序操作，建议指示使用 NAME 列的索引 IDX_T1_NAME，因为

这样可以在执行过程中省略掉排序操作(执行计划中可以看出来),比使用 ID 列索引代价小。

```
    SELECT * FROM T1 INDEX IDX_T1_NAME WHERE ID > 2011 AND NAME < 'XXX' ORDER BY
NAME;
```

或

```
    SELECT /*+ INDEX(A IDX_T1_NAME)*/ * FROM T1 A WHERE ID > 2011 AND NAME < 'XXX'
ORDER BY NAME;
```

2. 不使用索引

语法:

```
    /*+ NO_INDEX (表名[,] 索引名) { NO_INDEX (表名[,] 索引名)} */
```

可以指定多个索引,则这些索引都不能被使用。一个语句中最多指定 8 个索引。

24.6.2 连接方法提示

DBA 可以通过指定两个表间的连接方法来检测不同连接方式的查询效率,指定的连接可能由于无法实现或代价过高而被忽略。如果连接方法提示中的表名(别名)或索引名无效也会被自动忽略。

```
--数据准备
    DROP TABLE T1 CASCADE;
    DROP TABLE T2 CASCADE;

    CREATE TABLE T1 (ID INTEGER,NAME VARCHAR(128));
    CREATE TABLE T2 (ID INTEGER,NAME VARCHAR(128));

    begin
    for i in 1..1000 loop
    insert into T1 values(i,'dameng'||i);
    insert into T2 values(i+500,'damengsh'||i);
    end loop;
    end;
```

1. USE_HASH

强制两个表间使用指定顺序的哈希连接,例如:

```
    EXPLAIN SELECT /*+ USE_HASH(T1, T2) */ * FROM T1, T2 WHERE T1.id = T2.id;
    1   #NSET2: [1, 9820, 104]
    2     #PRJT2: [1, 9820, 104]; exp_num(4), is_atom(FALSE)
    3       #HASH2 INNER JOIN: [1, 9820, 104];  KEY_NUM(1);
    4         #CSCN2: [0, 1001, 52]; INDEX33555536(T1)
    5         #CSCN2: [0, 1001, 52]; INDEX33555539(T2)
```

2. NO_USE_HASH

强制两个表间不能使用指定顺序的哈希连接,例如:

```
    EXPLAIN SELECT /*+ NO_USE_HASH(T1, T2) */ * FROM T1, T2 WHERE T1.id = T2.id;
    1   #NSET2: [1, 9820, 104]
    2     #PRJT2: [1, 9820, 104]; exp_num(4), is_atom(FALSE)
    3       #HASH2 INNER JOIN: [1, 9820, 104];  KEY_NUM(1);
    4         #CSCN2: [0, 1001, 52]; INDEX33555539(T2)
    5         #CSCN2: [0, 1001, 52]; INDEX33555536(T1)
```

NO_USE_HASH(T1, T2)表示不允许 T1 作为左表,T2 作为右表的哈希连接,但 T1 作为右表的哈希连接还是允许的。

3. USE_NL

强制两个表间使用嵌套循环连接，例如：

```
EXPLAIN SELECT /*+ USE_NL(a, b) */ * FROM T1 a, T2 b WHERE a.ID = b.ID;
1   #NSET2: [22482, 9820, 104]
2     #PRJT2: [22482, 9820, 104]; exp_num(4), is_atom(FALSE)
3       #SLCT2: [22482, 9820, 104]; A.ID = B.ID
4         #NEST LOOP INNER JOIN2: [22482, 9820, 104];
5           #CSCN2: [0, 1001, 52]; INDEX33555536(T1 as A)
6           #CSCN2: [0, 1001, 52]; INDEX33555539(T2 as B)
```

4. NO_USE_NL

强制两个表间不能使用嵌套循环连接，例如：

```
EXPLAIN SELECT /*+ NO_USE_NL(a, b) */ * FROM T1 a, T2 b WHERE a.ID = b.ID;
1   #NSET2: [1, 9820, 104]
2     #PRJT2: [1, 9820, 104]; exp_num(4), is_atom(FALSE)
3       #HASH2 INNER JOIN: [1, 9820, 104];  KEY_NUM(1);
4         #CSCN2: [0, 1001, 52]; INDEX33555536(T1 as A)
5         #CSCN2: [0, 1001, 52]; INDEX33555539(T2 as B)
```

5. USE_NL_WITH_INDEX

当连接情况为左表+右表索引时，强制两个表间使用索引连接，例如：

```
--数据准备
    CREATE INDEX IDX_T2_ID ON T2(ID);
--执行 EXPLAIN
    EXPLAIN SELECT /*+ USE_NL_WITH_INDEX(T1, IDX_T2_ID) */ * FROM T1, T2 WHERE
T1.ID = T2.ID;
    1   #NSET2: [6, 9800, 104]
    2     #PRJT2: [6, 9800, 104]; exp_num(4), is_atom(FALSE)
    3       #NEST LOOP INDEX JOIN2: [6, 9800, 104]
    4         #CSCN2: [0, 1000, 52]; INDEX33555536(T1)
    5         #BLKUP2: [6, 3, 0]; IDX_T2_ID(T2)
    6           #SSEK2: [6, 3, 0]; scan_type(ASC), IDX_T2_ID(T2),
scan_range[T1.ID,T1.ID]
```

6. NO_USE_NL_WITH_INDEX

当连接情况为左表+右表索引时，强制两个表间不能使用索引连接，例如：

```
    EXPLAIN SELECT /*+ NO_USE_NL_WITH_INDEX(T1, IDX_T2_ID) */ * FROM T1, T2 WHERE
T1.ID = T2.ID;
    1   #NSET2: [1, 9800, 104]
    2     #PRJT2: [1, 9800, 104]; exp_num(4), is_atom(FALSE)
    3       #HASH2 INNER JOIN: [1, 9800, 104];  KEY_NUM(1);
    4         #CSCN2: [0, 1000, 52]; INDEX33555536(T1)
    5         #CSCN2: [0, 1000, 52]; INDEX33555539(T2)
--使用完毕，请删除索引
    DROP INDEX IDX_T2_ID;
```

7. USE_MERGE

强制两个表间使用归并连接。归并连接所用的两个列都必须是索引列。例如：

```
--数据准备
    CREATE INDEX IDX_T1_ID ON T1(ID);
    CREATE INDEX IDX_T2_ID ON T2(ID);
    STAT 100 ON T1(ID);
```

```
    STAT 100 ON T2(ID);
--执行 EXPLAIN
    EXPLAIN SELECT /*+ USE_MERGE(T1,T2) */ * FROM T1, T2 WHERE T1.ID = T2.ID AND
T1.ID < 1 AND T2.ID < 1;
    1   #NSET2: [0, 1, 104]
    2     #PRJT2: [0, 1, 104]; exp_num(4), is_atom(FALSE)
    3       #MERGE INNER JOIN3: [0, 1, 104];
    4         #BLKUP2: [0, 1, 52]; IDX_T1_ID(T1)
    5           #SSEK2: [0, 1, 52]; scan_type(ASC), IDX_T1_ID(T1),
scan_range(null2,1)
    6         #BLKUP2: [0, 1, 52]; IDX_T2_ID(T2)
    7           #SSEK2: [0, 1, 52]; scan_type(ASC), IDX_T2_ID(T2),
scan_range(null2,1)
```

当连接类型为外连接时，无法使用归并连接，此时即使指定 USE_MERGE，也不起作用。

8. NO_USE_MERGE

强制两个表间不能使用归并连接，例如：

```
    EXPLAIN SELECT /*+ NO_USE_MERGE(T1,T2) */ * FROM T1, T2 WHERE T1.ID = T2.ID
AND T1.ID > 1 AND T2.ID > 1;
    1   #NSET2: [0, 1, 104]
    2     #PRJT2: [0, 1, 104]; exp_num(4), is_atom(FALSE)
    3       #SLCT2: [0, 1, 104]; T2.ID < 1
    4         #HASH2 INNER JOIN: [0, 1, 104];  KEY_NUM(1);
    5           #SLCT2: [0, 1, 104]; T2.ID < 1
    6             #NEST LOOP INDEX JOIN2: [0, 1, 104]
    7               #ACTRL: [0, 1, 104];
    8                 #BLKUP2: [0, 1, 52]; IDX_T1_ID(T1)
    9                   #SSEK2: [0, 1, 52]; scan_type(ASC), IDX_T1_ID(T1),
scan_range(null2,1)
    10                #BLKUP2: [0, 1, 0]; IDX_T2_ID(T2)
    11                  #SSEK2: [0, 1, 0]; scan_type(ASC), IDX_T2_ID(T2),
scan_range[T1.ID,T1.ID]
    12          #CSCN2: [0, 1000, 52]; INDEX33555550(T2)
--使用完毕，请删除索引
    DROP INDEX IDX_T1_ID;
    DROP INDEX IDX_T2_ID;
```

9. SEMI_GEN_CROSS

优先采用半连接转换为等价的内连接，仅 OPTIMIZER_MODE=1 有效。例如：

```
    EXPLAIN SELECT /*+ SEMI_GEN_CROSS  OPTIMIZER_MODE(1) */ COUNT(*) FROM T1 A
WHERE A.ID IN (SELECT B.ID FROM T1 B);
    1   #NSET2: [3, 1, 8]
    2     #PRJT2: [3, 1, 8]; exp_num(1), is_atom(FALSE)
    3       #AAGR2: [3, 1, 8]; grp_num(0), sfun_num(1)
    4         #HASH2 INNER JOIN: [2, 1000, 8];  KEY_NUM(1);
    5           #PRJT2: [1, 1000, 4]; exp_num(1), is_atom(FALSE)
    6             #DISTINCT: [1, 1000, 4]
    7               #CSCN2: [0, 1000, 4]; INDEX33555531(T1 as B)
    8           #CSCN2: [0, 1000, 4]; INDEX33555531(T1 as A)
```

10. NO_SEMI_GEN_CROSS

不采用半连接转换为等价的内连接，仅 OPTIMIZER_MODE=1 有效。例如：

```
    EXPLAIN SELECT /*+ NO_SEMI_GEN_CROSS  OPTIMIZER_MODE(1) */ COUNT(*) FROM T1
A WHERE A.ID IN (SELECT B.ID FROM T1 B);
    1   #NSET2: [2, 1, 4]
    2     #PRJT2: [2, 1, 4]; exp_num(1), is_atom(FALSE)
    3       #AAGR2: [2, 1, 4]; grp_num(0), sfun_num(1)
    4         #HASH LEFT SEMI JOIN2: [1, 1000, 4]; KEY_NUM(1);
    5           #CSCN2: [0, 1000, 4]; INDEX33555531(T1 as A)
    6           #CSCN2: [0, 1000, 4]; INDEX33555531(T1 as B)
```

11. USE_CVT_VAR

优先采用变量改写方式实现连接，适合驱动表数据量少而另一侧计划较复杂的场景，仅 OPTIMIZER_MODE=1 有效。例如：

```
    EXPLAIN SELECT /*+ USE_CVT_VAR OPTIMIZER_MODE(1) */ COUNT(*) FROM T1 A WHERE
A.id = 1001 AND EXISTS (SELECT 1 FROM T1 B, T1 C WHERE B.id = C.id AND A.name=
B.name);
    1   #NSET2: [1, 1, 60]
    2     #PRJT2: [1, 1, 60]; exp_num(1), is_atom(FALSE)
    3       #AAGR2: [1, 1, 60]; grp_num(0), sfun_num(1)
    4         #NEST LOOP SEMI JOIN2: [0, 1, 60];  join condition(A.NAME =
B.NAME)[with var]
    5           #SLCT2: [0, 1, 60]; A.ID = 1001
    6             #CSCN2: [0, 1000, 60]; INDEX33555531(T1 as A)
    7           #HASH2 INNER JOIN: [1, 1, 56];  KEY_NUM(1);
    8             #SLCT2: [0, 1, 52]; B.NAME = var1
    9               #CSCN2: [0, 1000, 52]; INDEX33555531(T1 as B)
    10            #CSCN2: [0, 1000, 4]; INDEX33555531(T1 as C)
```

12. NO_USE_CVT_VAR

不考虑变量改写方式实现连接，仅 OPTIMIZER_MODE=1 有效。例如：

```
    EXPLAIN SELECT /*+ NO_USE_CVT_VAR OPTIMIZER_MODE(1) */ COUNT(*) FROM  T1 A
WHERE A.id = 1001 AND EXISTS (SELECT 1 FROM T1 B, T1 C WHERE B.id = C.id AND A.name=
B.name);
    1   #NSET2: [3, 1, 60]
    2     #PRJT2: [3, 1, 60]; exp_num(1), is_atom(FALSE)
    3       #AAGR2: [3, 1, 60]; grp_num(0), sfun_num(1)
    4         #HASH LEFT SEMI JOIN2: [2, 1, 60]; KEY_NUM(1);
    5           #SLCT2: [0, 1, 60]; A.ID = 1001
    6             #CSCN2: [0, 1000, 60]; INDEX33555531(T1 as A)
    7           #HASH2 INNER JOIN: [1, 1000, 56];  KEY_NUM(1);
    8             #CSCN2: [0, 1000, 4]; INDEX33555531(T1 as C)
    9             #CSCN2: [0, 1000, 52]; INDEX33555531(T1 as B)
```

13. ENHANCED_MERGE_JOIN

一般情况下，归并连接需要左右孩子的数据按照连接列有序，使用此优化器提示时，优化器将考虑通过插入排序操作符的方式实现归并连接，仅 OPTIMIZER_MODE=1 有效。例如：

```
    EXPLAIN SELECT /*+ stat(T1 1M) stat(T2 1M) */COUNT(*) FROM T1, T2 WHERE
T1.NAME=T2.NAME AND T1.ID=T2.ID;
   --不加 hint 时计划，使用了哈希连接
   1   #NSET2: [442, 1, 104]
   2     #PRJT2: [442, 1, 104]; exp_num(1), is_atom(FALSE)
```

```
3      #AAGR2: [442, 1, 104]; grp_num(0), sfun_num(1)
4        #HASH2 INNER JOIN: [442, 1000000000000, 104];  KEY_NUM(2);
5          #CSCN2: [115, 1000000, 52]; INDEX33762063(T1)
6          #CSCN2: [115, 1000000, 52]; INDEX33762064(T2)
-- 加 HINT 后计划，通过增加排序以使用了归并连接
1   #NSET2: [436, 1, 104]
2     #PRJT2: [436, 1, 104]; exp_num(1), is_atom(FALSE)
3       #AAGR2: [436, 1, 104]; grp_num(0), sfun_num(1)
4         #MERGE INNER JOIN3: [436, 1000000000000, 104];
5           #SORT3: [122, 1000000, 52]; key_num(2), is_distinct(FALSE),
top_flag(0)
6             #CSCN2: [115, 1000000, 52]; INDEX33762063(T1)
7           #SORT3: [122, 1000000, 52]; key_num(2), is_distinct(FALSE),
top_flag(0)
8             #CSCN2: [115, 1000000, 52]; INDEX33762064(T2)
```

24.6.3 连接顺序提示

多表连接时优化器会考虑各种可能的排列组合顺序。使用 ORDER HINT 指定连接顺序提示可以缩小优化器试探的排列空间，进而得到接近 DBA 所期望的查询计划。如果连接顺序和连接方法提示同时指定且二者间存在自相矛盾，优化器会以连接顺序提示为准。

ORDER HINT

语法：/*+ ORDER (T1, T2 , T3, … tn) */

本节中的例子用到 4 个表 T1, T2 , T3, T4。

```
CREATE TABLE T1(C1 INT,C2 VARCHAR);
CREATE TABLE T2(D1 INT,D2 VARCHAR);
CREATE TABLE T3(E1 INT,E2 VARCHAR);
CREATE TABLE T4(F1 INT,F2 VARCHAR);
```

例 1：

```
SELECT * FROM t1, T2 , T3, T4 WHERE …
```

如果期望表的连接顺序是 T1, T2, T3，那么可以加入这样的提示：

```
SELECT /*+ ORDER(T1, T2, T3 )*/* FROM T1, T2 , T3, T4 WHERE …
```

在指定上述连接顺序后，T4,T1,T2,T3 或 T1,T2,T4,T3 会被考虑；T3,T1,T2 或 T1,T3,T2 不被考虑。

连接顺序也可以和连接方法同时指定用于得到更特定的执行计划。例如：

```
EXPLAIN SELECT /*+ OPTIMIZER_MODE(1), ORDER(T1,T2,T3,T4) ,USE_HASH(T1,T2),
USE_HASH(T2,T3), USE_HASH(T3,T4)*/* FROM T1,T2,T3,T4 WHERE T1.c1=T2.d1 AND T2.d2
= T3.e2 AND T3.e1 = T4.f1;
```

此时的执行计划如下：

```
1   #NSET2: [2, 1, 208]
2     #PRJT2: [2, 1, 208]; exp_num(8), is_atom(FALSE)
3       #HASH2 INNER JOIN: [2, 1, 208];  KEY_NUM(1);
4         #HASH2 INNER JOIN: [1, 1, 156];  KEY_NUM(1);
5           #HASH2 INNER JOIN: [0, 1, 104];  KEY_NUM(1);
6             #CSCN2: [0, 1, 52]; INDEX33555490(T1)
7             #CSCN2: [0, 1, 52]; INDEX33555491(T2)
8           #CSCN2: [0, 1, 52]; INDEX33555492(T3)
```

```
9        #CSCN2: [0, 1, 52]; INDEX33555493(T4)
```

24.6.4 统计信息提示

优化器在计划优化阶段会自动获取基表的行数。但是一些特殊类型的表行数估算并不准确，或者 DBA 希望了解表大小对计划影响的时候，需要手动设置表的行数。

```
语法：/*+ STAT (表名, 行数) */
```

统计信息提示只能针对基表设置，视图和派生表等对象设置无效。如果表对象存在别名则必须使用别名。行数只能使用整数，或者整数+K（千），整数+M（百万），整数+G（十亿）。行数提示设置后，统计信息的其它内容也会做相应的调整。

例 1：

```
CREATE TABLE T_S(C1 INT);
INSERT INTO T_S SELECT LEVEL FROM DUAL CONNECT BY LEVEL<= 100;
COMMIT;
STAT 100 ON T_S(C1);
EXPLAIN SELECT /*+ STAT(T_S,1M) */ * FROM T_S WHERE C1 <= 10;
```

计划如下：

```
1   #NSET2: [107, 100000, 12]
2     #PRJT2: [107, 100000, 12]; exp_num(2), is_atom(FALSE)
3       #SLCT2: [107, 100000, 12]; T_S.C1 <= 10
4         #CSCN2: [107, 100000, 12]; INDEX33555897(T_S)
```

不使用 HINT 时计划：

```
1   #NSET2: [0, 10, 12]
2     #PRJT2: [0, 10, 12]; exp_num(2), is_atom(FALSE)
3       #SLCT2: [0, 10, 12]; T_S.C1 <= 10
4         #CSCN2: [0, 100, 12]; INDEX33555897(T_S)
```

第五部分 数据高可用性

第25章 故障恢复

25.1 概述

数据库系统在运行过程中可能会发生一些故障。造成故障的原因多种多样，包括磁盘崩溃、电源故障、软件错误，甚至人为破坏。这些情况一旦发生，就可能会丢失数据，数据库系统将无法正常运行。因此，故障恢复是数据库系统必不可少的组成部分，即数据库系统必须保证即使发生故障，也可以保障数据的完整性和一致性。

支持故障恢复的技术主要是日志，日志以一种安全的方式记录数据库系统变更的历史信息，一旦系统出现故障，数据库系统可以根据日志将系统恢复至故障发生前的某个时刻。数据库系统的日志分为两种类型：一是 REDO 日志，在数据被修改后记录它的新值；另一种是 UNDO 日志，在数据被修改前记录它的旧值。

另外，当服务器处于归档模式时，如果数据库发生故障，通过备份文件和归档日志可以恢复到指定时间点。

25.2 REDO 日志

在恢复操作中最重要的结构是联机 REDO 日志。REDO 日志存放在日志表空间文件中，这些文件存储数据库所做的所有物理更改信息。达梦数据库每一个实例都有一个相关联的联机重做日志，通过重做日志可以保证数据库的完整性和一致性。

DM SERVER 的 REDO 日志用于存储被修改数据的新值，包括事务对数据文件和回滚段的修改。REDO 日志每次被修改以后，都会自动生成一个新的日志序列值 LSN（Log Sequence Number）。LSN 取值范围 0~正无穷大，新建的库 LSN 为 0，以后日志每被写入一次，LSN 值增加 1。

REDO 日志里包含有一种特殊的记录，叫 PWR（Page Written Record）日志。PWR 包括（ts_id, fil_id, page_no, page_lsn）信息。每个数据页刷盘时，都会生成一条对应的 PWR 日志。PWR 日志的 lsn 与上一个日志的 lsn 相同。在以下情况下可以利用 PWR 日志：

1．RAC 故障恢复时：利用 PWR 日志减少加载磁盘数据页的次数；

2．故障恢复时：利用 PWR 日志提升 redo 速度；

3．增量备份时：利用 PWR 日志提升备份速度。

在第 2 章中提到过，系统采用了单独的日志文件来存储 REDO 日志，即联机日志文件。DM 至少有两个日志文件，日志文件是循环利用的。日志文件由日志表空间管理。DM 支持增加日志文件和扩展日志文件大小。

例 1 增加大小为 80M 的日志文件 dameng_003.log

```
ALTER DATABASE ADD LOGFILE 'dameng_003.log' size 80;
```

例 2 将日志文件 dameng_003.log 大小更改为 100M

```
ALTER DATABASE RESIZE LOGFILE 'dameng_003.log' to 100;
```

在 MOUNT 状态下，支持对日志文件的重命名操作。

例 3 将日志文件 dameng_003.log 重命名为 dameng_004.log

```
ALTER DATABASE RENAME LOGFILE 'dameng_003.log' to 'dameng_004.log';
```

25.3 重做日志归档

DM 服务器可以运行在两种模式下，即归档模式和非归档模式，这两种模式可以由用户进行设置，系统在归档模式下运行时，会产生归档日志文件，此时系统管理员应该事先预留出足够的磁盘空间以便存储归档日志文件。通过 dm.ini 和 dmarch.ini 可以配置归档，参见第 2 章。归档日志包括以下五种类型：

25.3.1 本地归档

写入 REDO 日志到本地归档文件，在 REDO 日志写入日志文件后触发，由归档线程完成本地归档动作，最多可以设置 8 个本地归档。若磁盘空间不足，所有本地归档一旦失效，系统会被强制挂起，直到磁盘空间释放，本地归档成功后，再继续执行。

25.3.2 实时归档

在写入 REDO 日志到日志文件之前，通过 MAL 系统发送 REDO 日志到远程服务器，远程服务器收到 REDO 日志后，返回确认消息。收到确认消息后，执行后续操作，发送 REDO 日志失败，或从备库返回的数据库模式不是 STANDBY，将数据库切换为 SUSPEND 状态，阻塞所有 REDO 日志的写入操作。只能配置 1 个实时归档。

25.3.3 即时归档

即时归档在主库将 Redo 日志写入联机 Redo 日志文件后，再通过 MAL 系统将 Redo 日志发送到备库。即时归档是读写分离集群的实现基础，与实时归档的主要区别是发送 Redo 日志的时机不同。一个主库可以配置 1~8 个即时备库。

25.3.4 异步归档

在设定的时间点或者每隔设定时间，启动归档 REDO 日志发送。设置定时归档，必须确保至少有一个本地归档。系统调度线程根据设定，触发归档 REDO 日志发送事件。通过 MAL 系统，获取远程服务器的当前 LSN，生成发送归档 REDO 日志任务，加入任务队列。归档任务线程获取任务，通过 MAL 系统，发送到远程服务器。最多可以设置 8 个异步归档。

25.3.5 远程归档

远程归档就是将写入本地归档的 REDO 日志信息，发送到远程节点，并写入远程节点的指定归档目录中。远程归档与本地归档的主要区别是 REDO 日志写入的位置不同，本地归档将 REDO 日志写入数据库实例所在节点的磁盘，而远程归档则将 REDO 日志写入到其他数据库实例所在节点的指定归档目录。远程归档日志文件的命名规范和本地归档日志文件保持一致，都是以归档名+归档文件的创建时间进行组合命名。最多可以配置 8 个远程归档。

25.4 检查点

创建数据库时，联机日志文件通常被扩展至一定长度，其内容则被初始化为空，当系统运行时，该文件逐渐被产生的日志所填充。为了达到循环利用日志系统空间的目的，必须在所有日志文件空间将被占满时，系统能够自动清空一部分日志，以便重用日志文件的空间，为了保证被清空的日志所"保护"的数据在磁盘上是安全的，需要引入一个关键的数据库概念—检查点。当系统产生检查点时，将系统缓冲区中被修改过的数据页写入磁盘，以保证当前日志所"保护"的数据页都已安全写入磁盘，这样日志文件即可被安全重用。

当服务器启动和关闭时，系统都会产生检查点。服务器运行过程中，系统会自动判断是否需要执行检查点；当空闲日志空间不足时，系统自动产生一个检查点；系统调度线程也会定时产生检查点；还可以通过调用系统函数 CHECKPOINT 主动生成检查点。

系统调度线程根据 dm.ini 的参数配置，产生检查点，下面给出具体的例子和说明。

```
CKPT_INTERVAL       = 1800      #每间隔 1800 秒，产生检查点间隔
CKPT_FLUSH_RATE     = 5         #检查点的刷盘比例为 5%，将系统中所有脏页的 5%写入磁盘
CKPT_FLUSH_PAGES    = 1000      #一个检查点最少写入 1000 个脏页
```

检查点系统函数 CHECKPOINT（FLUSH_RATE），其参数 FLUSH_RATE 为 INTEGER 类型或 DOUBLE 类型，指示刷盘比例，替代 dm.ini 中的 CKPT_FLUSH_RATE，同时 CKPT_FLUSH_PAGES 参数同时生效。例如，执行 SELECT CHECKPOINT(10);会将系统中 10%脏页写入磁盘。

25.5 回滚段与回滚记录

DM SERVER 采用回滚段机制来处理 UNDO 日志。回滚段由一定数量的回滚页组成，回滚页存放的是一批回滚记录。回滚记录存放被修改数据的旧值，并有专属的格式，与物理记录格式不同。回滚段的管理同一般数据文件一样，其回滚数据页的分配、淘汰和释放也交由数据缓冲区完成。回滚文件属于回滚表空间。回滚段的数据并不会永久保留，事务结束后，由系统的 PURGE 模块释放回滚页。

REDO 日志记录了所有对数据库数据进行修改的数据，当然也包括回滚段的修改数据。因此，REDO 日志同样也会保护回滚数据。当使用重做日志恢复数据库时，系统将重做回滚段中修改了的数据，并将这些修改写回到回滚段中，从而形成回滚记录。回滚记录与重做记录类似，只不过回滚记录记录被修改数据的旧值，以达到数据回滚到原来状态的目的。

另外，回滚表空间不允许修改和删除操作。

25.6 系统故障恢复

系统故障也就是我们通常所说的系统崩溃。系统崩溃的原因比较复杂，有硬件故障，或者是数据库软件或操作系统的漏洞等。它导致整个系统停止运行，内存中的数据全部丢失，但磁盘上存储的数据仍然有效。

当系统故障发生时，系统管理员可以通过查看系统日志文件以了解系统故障发生的原因或故障发生之前系统的运行状况。多数情况下，系统在故障发生之前会自动记录产生故障的原因，但也不全是这样，如系统掉电时，发生故障的原因可能来不及记录。总之，在处理系统故障之前，系统管理员需要完全了解系统故障发生的原因，并采取相应的措施，如更换硬件，升级操作系统或数据库软件等，必要时还可以联系达梦公司的技术支持工程师，以获得必要的支持。

系统故障恢复的最后一步则是重新启动 DM 服务器。此时，系统将从最近一个检查点处

开始扫描联机日志，并重做日志记录的内容，这样系统即恢复到了故障发生前的某个很近的时刻，然后将系统中仍然活动的事务再依次利用回滚段进行回滚。由此可以看出，系统故障恢复可以保证事务的原子性和持久性，即系统故障发生时，仍然活动的事务可以被安全回滚，而已经提交的事务则保证其所修改数据的持久性。

25.7 介质故障恢复

介质故障指的是由于各种原因导致数据库系统存储在磁盘上的数据被损坏，如磁盘损坏等。介质故障是数据库系统最为严重的故障，此时系统已经无法重新启动，磁盘上的数据也无法复原。

出现介质故障后，系统管理员首先需要分析介质故障发生的原因，并采取措施，如磁盘损坏则更换磁盘等。由于系统的数据已经全部或部分丢失，此时只能依赖以前建立的备份和系统产生的归档日志文件进行恢复。由于涉及到备份，具体的介质恢复策略将在《DM8 备份与还原.docx》中详细介绍。

需要说明的是，为了避免系统在出现故障时丢失数据，应关闭磁盘缓存（若存在的话），具体事项请咨询操作系统管理员。

第26章 数据复制

26.1 概述

达梦数据复制（DATA REPLICATION）是一个分担系统访问压力、加快异地访问响应速度、提高数据可靠性的解决方案。将一个服务器实例上的数据变更复制到另外的服务器实例。可以用于解决大、中型应用中出现的因来自不同地域、不同部门、不同类型的数据访问请求导致数据库服务器超负荷运行、网络阻塞、远程用户的数据响应迟缓的问题。

26.2 重要概念

1. 数据库状态

服务器的状态，在不同的状态下，对能够进行的操作有不同的限制。详见数据守护部分。

2. 主服务器

发起复制操作的服务器，称为主服务器。

3. 从服务器

接收主服务器发送的数据并进行复制的服务器，称为从服务器。

4. 复制节点

涉及到复制的服务器，主服务和从服务的统称。一个节点既可以是主服务器也可以是从服务器。

5. 复制服务器（RPS）

在数据复制环境中，负责配置复制环境，定义复制关系的服务器。RPS 有且仅有一台，它只负责配置和监控，并不参与到复制过程中。

6. 复制关系

复制关系指明主服务器和从服务器以何种方式进行复制。按照复制的方式，复制关系分为同步复制和异步复制。

7. 同步复制关系

主服务器数据更新立即复制到从服务器。

8. 异步复制关系

主服务器和从服务器在某段时间内数据可能是不同的，主服务器数据更新不会立刻同步到从服务器，而是在经过一段时间后才进行复制。异步复制的同步时机由指定的定时器确定。

9. 逻辑日志

记录产生数据变化的逻辑操作的日志。记录的逻辑操作包括 INSERT、UPDATE、DELETE、TRUNCATE、ROLLBACK 和 COMMIT。

10. 复制源对象

主服务器上作为复制数据源的对象，可以是库、模式或表。在该对象上的操作都会被记录成逻辑日志，发送给从服务器进行复制操作。

11. 复制目标对象

从服务器上作为复制数据目标的对象。从服务器接收到逻辑日志后，将复制源对象的变化复制到复制目标对象中。

12. 复制对象映射

一对复制源对象和复制目标对象构成一个复制对象映射。构成映射的源对象和目标对象必须是同一类型的对象。根据对象的类型，复制映射分为库级、模式级和表级三个级别。其中表级要求源表和目标表结构完全一致，库级和模式级没有要求。库级和模式级复制映射会将其 DDL 也进行复制。复制映射包括只读模式和非只读模式。对于只读模式的映射，映射的目的表禁止用户更新。

13. 复制组

一组逻辑相关的复制关系可以构造成为复制组。通过复制组，可以构造出一对多复制、多对一复制、级联复制、对称复制、循环复制等复杂的逻辑复制环境。

26.3 体系构架

数据复制系统由以下部件构成：

1. 复制服务器；
2. 复制节点；
3. 各实例站点间通讯的 MAL 系统。

除了系统管理员通过复制服务器定义复制和处理异常外，其他部分的处理及主从服务器之间复制操作对于用户是透明的。

整个复制环境的配置境况如下图所示。

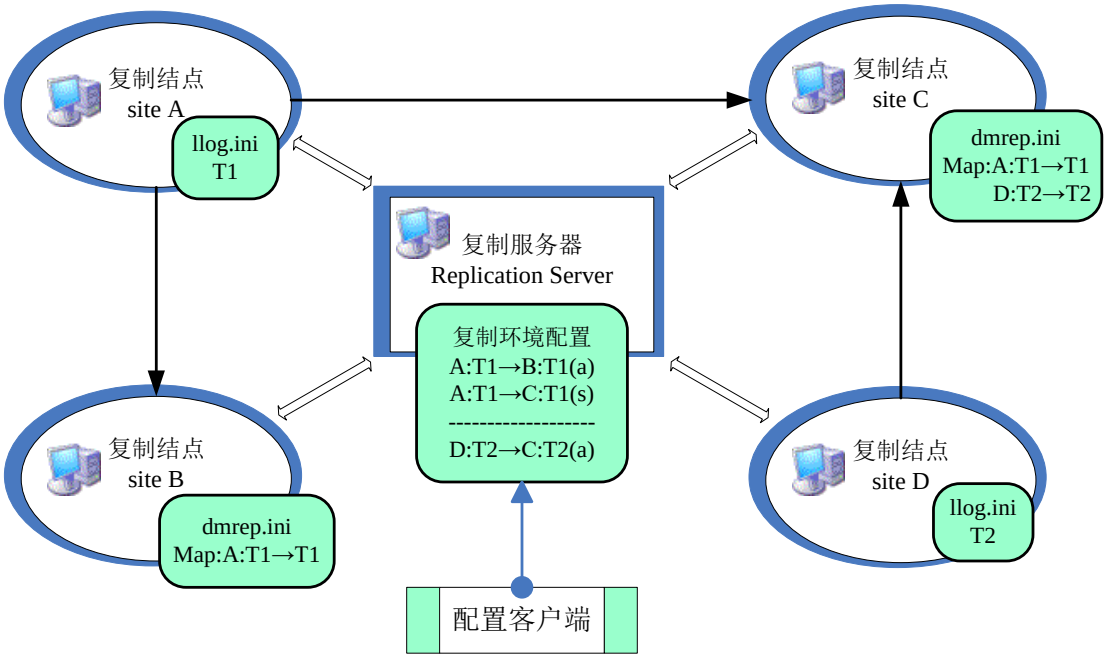


图 26.1 复制环境结构图

在整个环境中有且仅有一台复制服务器（RPS），用户通过 RPS 定义复制及复制环境，但 RPS 并不参与到复制过程中。

DM7 中，将复制逻辑日志按照配置归档到本站点指定目录称为本地归档，将日志的发送称为日志的远程归档。

复制节点上，与复制相关的配置文件有 dmtimer.ini、dmllog.ini、dmrep.ini。各个配置文件在复制中的功能如下表所示。

表 26.1 复制中各配置文件的作用

名称	简介	有效节点	功能
dmtimer.ini	定时器配置文件	主服务器	记录异步复制的定时器信息
dmllog.ini	逻辑日志配置文件	主服务器	记录复制源对象及其逻辑日志相关的信息(包括逻辑本地归档信息和远程归档信息)
dmrep.ini	复制信息配置文件	主服务器/从服务器	主服务器上记录复制相关逻辑日志的编号； 从服务器上记录复制关系的主服务器信息以及复制对象映射的信息

这些配置文件内的信息意义见第 2 章。这些文件均是 RPS 依据用户配置通过 MAL 系统自动生成或修改的，管理员不需要手动进行管理。

在数据复制过程中，除了配置文件，其他的重要文件如下表。

表 26.2 复制中的重要文件

名称	有效节点	功能
逻辑日志文件	主服务器	在数据文件夹内的 llog_01.log 和 llog_02.log 两个文件，记录在执行过程中所有的逻辑日志。
逻辑日志归档文件	主服务器	根据配置，生成在指定的逻辑日志归档路径中，名称为 llog 生成时间.log，是特定逻辑日志下所有记录的本地归档文件，也是发送给复制从服务器的数据来源
复制数据文件	从服务器	在数据文件夹下的 repx（x 为复制关系的 id）下，以 repx_y.rep（x，y 为该文件内记录的编号）为文件名，是从服务器接收到复制数据之后的临时缓存文件，也是复制执行的依据。

这些文件都是流式文件，主服务器在执行过程中，逻辑日志先记录到逻辑日志文件中，并根据配置的信息，将逻辑日志分别分发归档到不同的逻辑日志归档文件中，并在恰当的时机将逻辑日志归档文件的内容发送给从服务器。

从服务器接收到一批数据就产生一个复制数据文件将数据存储其中，防止从服务器复制速度低导致复制数据的丢失。

逻辑日志文件大小限制为 32M，两个文件交替使用。

归档路径内每个逻辑日志归档文件大小限制为 32M，文件写满后增加新文件来存放新的记录。在 dmllog.ini 中有配置归档路径内所有文件总的空间限制，若达到限制的空间，则会删除之前的文件；若配置为无空间限制，则管理员可根据复制情况进行删除。

复制数据文件是从服务器一次接收的复制数据的临时文件，其大小不会超过 32K；在其数据复制结束后会被自动删除。

26.4 配置数据复制

配置数据复制在 RPS 上进行，所有的配置接口见《DM7_SQL 语言使用手册》附录 3 的“5）数据复制管理”章节。在配置数据复制之前，需要保证复制服务器和所有待配置节点的实例名各不相同，配置好其 MAL 系统并保证网络环境正常。按照复制组、复制关系、复制表映射的顺序配置复制环境。

下面举一个简单的例子来说明数据复制的配置。

1．准备工作

参与复制的复制实例的信息如下表所示。

表 26.3 参与复制的实例信息

服务器	实例名	IP 地址	服务器端口号	MAL 端口号	文件目录
复制服务器	A	192.168.0.11	5236	5241	C:\data\DAMENG

主服务器	B	192.168.0.12	5236	5242	C:\data\DAMENG
从服务器	C	192.168.0.13	5236	5243	C:\data\DAMENG

假设 B 上存在 USER1.T1 表,C 上存在与 B 上 USER1.T1 表结构完全相同的 USER2.T2 表，现需要创建一个 B 上 USER1.T1 表到 C 上 USER2.T2 表的同步复制关系，其名称为 REPB2C。

2． 参数设置

dm.ini 配置，分别修改 dm.ini 中对应项如下表所示。

表 26.4 复制中 dm.ini 的配置

服务器	dm.ini 设置
复制服务器	INSTANCE_NAME = A PORT_NUM = 5236 MAL_INI = 1
主服务器	INSTANCE_NAME = B PORT_NUM = 5236 MAL_INI = 1
从服务器	INSTANCE_NAME = C PORT_NUM = 5236 MAL_INI = 1

dmmal.ini 配置，用户实际配置时需根据情况更改对应项。

```
MAL_CHECK_INTERVAL  = 5
MAL_CONN_FAIL_INTERVAL = 5
[MAL_INST1]
  MAL_INST_NAME     = A
  MAL_HOST          = 192.168.0.11
  MAL_PORT          = 5241
  MAL_INST_PORT     = 5236
  MAL_INST_HOST     = 192.168.0.11

[MAL_INST2]
  MAL_INST_NAME     = B
  MAL_HOST          = 192.168.0.12
  MAL_PORT          = 5242
  MAL_INST_PORT     = 5236
  MAL_INST_HOST     = 192.168.0.12

[MAL_INST3]
  MAL_INST_NAME     = C
  MAL_HOST          = 192.168.0.13
  MAL_PORT          = 5243
  MAL_INST_PORT     = 5236
  MAL_INST_HOST     = 192.168.0.13
```

每个站点的 dmmal.ini 配置必须一致,一个站点配置好后可直接拷贝到另外两个站点。

3． 复制服务器初始化

如果是第一次使用复制服务器，需要对复制服务器执行初始化操作。通过执行系统函数 SP_INIT_REP_SYS(create_flag)来初始化复制服务器。其主要作用是创建复制用户（SYSREP/SYSREP）和创建复制服务器上需要的系统表。SP_INIT_REP_SYS 的参数

create_flag 为 1 时表示创建用户和系统表，为 0 时表示删除用户和系统表。

4. 环境的配置

以上工作完成后，即可进行复制环境的配置了。

1) 启动 3 台服务器，启动的顺序不分先后。

2) 登录 RPS A，保证服务器状态为 OPEN，开始复制配置。

3) 创建复制组 REP_GRP_B2C

```
SP_RPS_ADD_GROUP('REP_GRP_B2C', '主从同步复制');
```

4) 开始设置

```
SP_RPS_SET_BEGIN('REP_GRP_B2C');
```

5) 添加复制关系

```
SP_RPS_ADD_REPLICATION ('REP_GRP_B2C', 'REPB2C', 'B 到 C 的同步复制', 'B', 'C',
NULL, 'D:\REPB2C');
```

6) 添加复制映射

```
SP_RPS_ADD_TAB_MAP('REPB2C', 'USER1', 'T1', 'USER2', 'T2', 0);
```

7) 提交设置

```
SP_RPS_SET_APPLY();
```

至此，复制环境配置完成。以上的例子只是一个最简单的复制环境。复制的配置灵活，在同一个复制组内，一个主服务器可以有多个从服务器，一个复制节点可以既是主服务器又是从服务器。管理员可以根据实际需要，配置出对称、一对多、多对一、级联、循环的复制环境。

在配置过程中或配置完成后，可以对复制的配置进行修改。修改包括复制组、复制关系、复制对象的删除和复制关系属性的修改。这些修改操作都必须在开始复制 SP_RPS_SET_BEGIN 和提交复制 SP_RPS_SET_APPLY 之间进行。若需要删除复制组，则该复制组不能处于配置阶段，即该组的配置已经提交或取消。

删除复制映射

```
SP_RPS_DROP_TAB_MAP('REPB2C', 'USER1', 'T1', 'USER2', 'T2');
```

添加一个定时器，将同步复制修改为异步复制

```
SP_RPS_ADD_TIMER('B2C_TIMER','',1,0,0,0,'19:50:33',NULL,'2011-08-24
19:50:33',NULL,1);
SP_RPS_REP_RESET_TIMER('REPB2C','B2C_TIMER');
```

修改复制的错误超时时间，超时的时间单位是秒

```
SP_RPS_SET_ROUTE_FAULT_TIMEOUT('REPB2C', 60);
```

删除复制关系

```
SP_RPS_DROP_REPLICATION('REPB2C');
```

删除整个复制组

```
SP_RPS_DROP_GROUP('REP_GRP_B2C');
```

另外，在配置或修改配置时想要取消操作，可以使用如下系统过程结束配置。

```
SP_RPS_SET_CANCEL();
```

26.5 监控数据复制

26.5.1 复制故障监控

配置完成后复制服务器 RPS 虽然不参与复制的具体执行，但是在复制过程中，还是建议保持 RPS 的运行来对复制进行监控。

复制系统内，状态分为复制节点的状态和复制关系的状态。具体内容见下表。

表 26.5 节点状态和复制关系状态

类型	子类型	意义	值
节点实例	系统状态	正常	0
		恢复	1
		系统异常	2
	网络状态	正常	0
		连接异常	4
复制关系	复制关系状态	正常	0
		连接异常	1
		待删除	512
		已在主服务器删除	528
		已在从服务器删除	513

上表复制节点的系统状态在 SYSREP.RPS_INSTANCES 中的 VALID_FLAG 字段表示；网络状态在 SYSREP.RPS_INSTANCES 表中 NET_VALID_FLAG 的字段表示。

复制关系的状态在 SYSREP.RPS_REPLICATIONS 中的 VALID_FLAG 字段表示。

复制系统的配置信息可以通过在复制 RPS 上查询系统表来实现。在 RPS 上，复制相关的信息都在 SYSREP 模式中，相关的表及其各字段的意义见附录 5。

复制的监视按复制组为单位进行，RPS 每隔 1 分钟轮询探查所有的复制组，在每个复制组内，依次探测各个复制节点，要求其返回复制节点本身的状态和其所涉及的复制的状态。

这里需要特别说明的是节点的系统异常状态是指该节点在复制环境中的状态，是一个推测值。当 RPS 无法得到节点的返回消息，其他节点的返回消息中所有涉及到该节点的复制关系都是异常时，RPS 就认为该节点系统异常。

若复制结点或复制关系发生异常，这些异常都会记录在 SYSREP.RPS_FAULT_HISTORY 表中。若其 END_TIME 字段为 NULL，表示该故障还没有结束，管理员需尽快检查复制节点的状态及其网络连接。

DM7 并没有特意设置对错误历史的管理接口，管理员可以通过对 SYSREP.RPS_FAULT_HISTORY 表的删除和查询来实现错误记录的管理。

26.5.2 复制故障处理

1. 故障超时设置和处理

在配置阶段，管理员可以配置复制关系和复制节点的故障超时时间。复制运行时当复制关系出现异常或节点出现系统异常时，RPS 会计算故障持续时间，在设置的超时时间以内，RPS 都会尝试进行故障恢复；若发现故障时间超过了设置的超时时间，则会删除该复制关系或节点，当然，删除节点时，该节点涉及的所有复制关系都会被删除。

删除复制关系时，若复制关系的主服务器或从服务器是异常状态，该复制关系不会立刻被清除，它会处于删除状态，在仍正常运行的服务器上删除；等待异常的服务器恢复后，将其从该服务器的复制配置中删除后，RPS 才会彻底的清理该复制关系；这也是复制关系状态中“待删除”、“已从主服务器上删除”和“已在从服务器上删除”状态的原因。

2. 复制关系异常修复

当 RPS 发现某复制关系异常时，RPS 会通知该复制关系的主服务器，要求其重新建立彼此间的网络连接，修复复制关系。

如果修复成功，重置 RPS 上复制关系的状态，并将复制关系的异常计时设为无效；若修复失败，则判断异常时间是否超时，若超时，则进入删除流程，否则等待到下一个监控修复

时间进行恢复。

修复成功后，如果是同步复制，则修复后立刻将异常期间的逻辑日志发送给从服务器，若是异步复制，则会等待定时器触发复制。

3． 复制节点网络故障恢复

RPS 在每次探查阶段中探查节点之前都会检查与节点间的网络连接；发现某结点无法通讯，则会跳过本轮的故障恢复阶段，直到 RPS 能够与之建立连接。

在网络故障期间，管理员可以通过查询 SYSREP.RPS_FAULT_HISTORY 表中实例网络异常的记录，尽快修复网络异常。

4． 复制节点系统异常恢复

当 RPS 在探查阶段中认定发现某节点系统异常，则会将所有该节点涉及到的复制关系置为异常状态，记录在 SYSREP.RPS_FAULT_HISTORY 中，等待管理员恢复节点系统。

复制节点系统异常恢复的过程是执行故障超时处理和修复剩余的复制关系两者统一执行的过程。节点系统恢复后，RPS 会根据故障期间故障处理的结果，删除因超时而被删除的复制关系；修复节点内剩余的所有复制关系。

5． 复制错误监控

在复制过程中，特别是在多对一或多对多的复制环境中，可能出现数据冲突。数据冲突发生后，从服务器会将其记录在相应的日志中，RPS 每隔固定时间会轮询所有结点，收集其间产生的数据冲突记录并将其插入到 SYSREP.RPS_CONFLICTS 表中，管理员可以通过查询该表来查看。

DM7 并没有特意设置对复制冲突记录的管理接口，管理员可以通过对 SYSREP.RPS_CONFLICTS 表的删除和查询来实现错误记录的管理。

26.6 复制用户和系统表

执行系统函数 SP_INIT_REP_SYS(1)创建的复制用户和系统表如下：

一 复制用户，被授予 RESOURCE 角色

```
用户名：SYSREP
密码：SYSREP
```

二 复制系统表。包含复制组表、复制节点实例表、复制关系表、复制映射表、故障历史表、冲突历史表和复制定时器表，共 7 张表。详情请参考附录 5 数据复制的系统表。

第六部分 附录

附录 1 数据字典

1． SYSOBJECTS

记录系统中所有对象的信息。

序号	列	数据类型	说明
1	NAME	VARCHAR(128)	对象名称
2	ID	INTEGER	对象 ID
3	SCHID	INTEGER	TYPE$=SCHOBJ 或者 TYPE$=TABOBJ 时表示对象所属的模式 ID，否则为 0。
4	TYPE$	VARCHAR(10)	对象的主类型。分为三种： 1）库级： UR用户，具体类型看SUBTYPE$、SCH模式、POLICY策略、GDBLINK全局DBLINK、DSYNOM全局同义词、DIR目录create directory、DMNOBJ域对象 2）模式级： SCHOBJ模式内对象，具体类型看SUBTYPE$ 3）表级： TABOBJ 表的下级对象，具体类型看SUBTYPE$
5	SUBTYPE$	VARCHAR(10)	对象的子类型。分为三种： 1）用户对象： USER用户、ROLE角色 2） 模式对象： UTAB 用户表、STAB 系统表、 VIEW 视图、 PROC 过程、SEQ 序列、PKG 包、TRIG 触发器、DBLINK 外部链接、SYNOM 同义词、 CLASS 对象类型（类）、TYPE 数据类型、JCLASS JAVA类、DOMAIN 域、CHARSET 字符集、CLLT 集合、CONTEXT 上下文 3）表对象： INDEX 索引、CNTIND 全文索引、CONS 约束
6	PID	INTEGER	对象的父对象 ID，为-1 表示当前行 PID 列无意义
7	VERSION	INTEGER	对象的版本
8	CRTDATE	DATETIME	对象的创建时间

9	INFO1	INTEGER	表对象： 表数据所在的缓冲区 ID(0xFF000000)， 数据页填充因子(0x00F00000)， BRANCH(0x000FF000) ， NOBARNCH(0x00000FF0) ， BRANCHTYPE(0x0000000F) 用户对象：BYTE(4)用户类型 视图对象：BIT(0) CHECK, BIT(1) CHECK CASCADE, BIT(2) 是否加密,BIT(4)SYSTEM 触发器对象： BIT(1) TV|EVENT FLAG，BIT(2,3) 执行类型(前或后)，BIT(4)是否加密，BIT(5)是否系统级，BIT(13)是否启用 对于 TV 触发器：BIT(6) RSFLAG, BIT(7) NEW REFED FLAG, BIT(8) OLD REFED FLAG, BIT(9) ALL NEW MDF FLAG 对于事件触发器：BIT(6,7) SCOPE, BIT(8,11) SCHEDUAL TYPE 约束对象：列数 存储过程：BIT(0)是否存储过程，BIT(1)是否加密, BIT(2)是否系统级 角色：角色类型 序列：BYTE(1)是否循环，BYTE(2)是否排序，BYTE(3)是否有缓存 同义词：是否带系统标识 包：BIT(1)文本是否加密，BIT(2)是否带系统标识
10	INFO2	INTEGER	表/用户/数据库/表空间：BYTE(4)空间限制值 视图：基表 ID
11	INFO3	BIGINT	序列：起始值 触发器：BYTE(0-3)EVENTS TV 触发器，BYTE(4)更新操作可触发的字段， BYTE(5)行前触发器中可被触发器修改值的新行字段，BYTE(6)元组级触发器中引用的字段， 事件触发器，BYTE(4)间隔，BYTE(5)子间隔，BYTE(6,7)分间隔 表：BYTE(0)表类型或临时表类型， BYTE(1)日志类型或错误响应或不可用标识，BYTE(2)是否临时表会话级， BYTE(3-4)区大小，BYTE(5)标记分布表 用户：BYTE(2)默认表空间 ID
12	INFO4	BIGINT	序列：增量

13	INFO5	VARBINARY(128)	表：BYTE(10)BLOB 数据段头 序列：BYTE(8)序列最大值，BYTE(8)序列最小值，BYTE(2)文件 ID，BYTE(4)页号，BYTE(2)序列当前位置
14	INFO6	VARBINARY(2048)	视图：BYTE(4)表或视图 ID 触发器： TV 触发器，BYTE(2)更新操作可触发字段， BYTE(2)元组级触发器前可能被触发器修改值的字段，BYTE(2)元组级触发器中引用的字段， 事件触发器，BYTE(8)开始/ BYTE(8)结束日期、BYTE(5)开始/ BYTE(5)结束时间 约束对象：(BYTE(4)ID)表列链表 同义词：BYTE(2)模式名和 BYTE(2)对象名 表： IDENTITY(BYTE(8) FOR SEED，BYTE(8) FOR INCREMENT)或 BYTE(4)列 ID
15	INFO7	BIGINT	保留
16	INFO8	VARBINARY(1024)	表： 外部表的控制文件路径 或者 BYTE(2)水平分区表记录总的子表数目，BYTE(4)垂直分区记录子表 ID
17	VALID	CHAR(1)	对象是否有效，'Y'表示有效，'N'表示失效

2． SYSINDEXES

记录系统中所有索引定义信息。

序号	列	数据类型	说明
1	ID	INTEGER	索引 ID
2	ISUNIQUE	CHAR(1)	是否为唯一索引
3	GROUPID	SMALLINT	所在表空间的 ID
4	ROOTFILE	SMALLINT	存放根的文件号
5	ROOTPAGE	INTEGER	存放根的页号
6	TYPE$	CHAR(2)	类型。BT：B 树，BM：位图，ST：空间，AR 数组
7	XTYPE	INTEGER	索引标识，联合其他字段标识索引类型。 BIT(0) 0 聚集索引，1 二级索引 BIT(1) 标识函数索引 BIT(2) 全局索引在水平分区子表上标识 BIT(3) 全局索引在水平分区主表上标识 BIT(4) 标识唯一索引 BIT(5) 标识扁平索引 BIT(6) 标识数组索引 BIT(11) 表示该位图索引是由改造后创建 BIT(12) 位图索引

			BIT(13) 位图连接索引 BIT(14) 位图连接索引虚索引 BIT(15) 空间索引 BIT(16) 标识索引是否可见
8	FLAG	INTEGER	索引标记。 BIT(0) 系统索引 BIT(1) 虚索引 BIT(2) PK BIT(3) 在临时表上 BIT(4) 无效索引 BIT(5) fast pool
9	KEYNUM	SMALLINT	索引包含的键值数目
10	KEYINFO	VARBINARY(816)	索引的键值信息
11	INIT_EXTENTS	SMALLINT	初始簇数目
12	BATCH_ALLOC	SMALLINT	下次分配簇数目
13	MIN_EXTENTS	SMALLINT	最小簇数

3. SYSCOLUMNS

记录系统中所有列定义的信息。

序号	列	数据类型	说明
1	NAME	VARCHAR(128)	列名
2	ID	INTEGER	父对象 ID
3	COLID	SMALLINT	列 ID
4	TYPE$	VARCHAR(128)	列数据类型
5	LENGTH$	INTEGER	列定义长度
6	SCALE	SMALLINT	列定义刻度
7	NULLABLE$	CHAR(1)	是否允许为空
8	DEFVAL	VARCHAR(2048)	缺省值
9	INFO1	SMALLINT	水平分区表：分区列的序号 其他表：BIT(0)压缩标记 列存储表： BIT(0)压缩标记；BIT(1-12)区大小； BIT(13)列存储的区上是否做最大最小值统计；BIT(14)是否加密列 视图：BYTE(2)多层视图中的最原始表的列 ID 存储过程：BYTE(2)参数类型
10	INFO2	SMALLINT	非垂直分区表：BIT(0)是否自增列； BIT(14)是否加密列 垂直分区表：BIT(0~9)列存储选项(列存储的表空间) ；BIT(14)是否加密列 视图：BYTE(2)多层视图中直接上层或 BYTE(2)垂直分区对应基表的列 ID 列存储表：group_id

4. SYSCONS

记录系统中所有约束的信息。

序号	列	数据类型	说明
1	ID	INTEGER	约束 ID
2	TABLEID	INTEGER	所属表 ID
3	COLID	SMALLINT	列 ID。暂时不支持，无意义。全部为-1

4	TYPE$	CHAR(1)	约束类型
5	VALID	CHAR(1)	约束是否有效
6	INDEXID	INTEGER	索引 ID
7	CHECKINFO	VARCHAR(2048)	check 约束的文本
8	FINDEXID	INTEGER	外键所引用的索引 ID
9	FACTION	CHAR(2)	前一字符对应外键的更新动作，后一字符对应外键的删除动作
10	TRIGID	INTEGER	动作触发器 ID

5. SYSSTATS

记录系统中的统计信息。

序号	列	数据类型	说明
1	ID	INTEGER	统计信息 ID
2	COLID	SMALLINT	列 ID，表级统计为-1
3	T_FLAG	CHAR(1)	对象的标记
4	T_TOTAL	BIGINT	总行数
5	N_SMAPLE	BIGINT	采样个数
6	N_DISTINCT	BIGINT	不同值的个数
7	N_NULL	BIGINT	空值个数
8	V_MIN	VARBINARY(255)	列的最小值
9	V_MAX	VARBINARY(255)	列的最大值
10	BLEVEL	TINYINT	B 树层次
11	N_LEAF_PAGES	BIGINT	叶子段总页数
12	N_LEAF_USED_PAGES	BIGINT	叶子占用的页数
13	CLUSTER_FACTOR	INTEGER	索引的 cluster_factor
14	N_BUCKETS	SMALLINT	直方图桶数目
15	DATA	BLOB	直方图数据
16	COL_AVG_LEN	INTEGER	平均行长
17	LAST_GATHERED	DATETIME(6)	最后收集时间
18	INFO1	VARBINARY(255)	预留列
19	INFO2	VARBINARY(255)	预留列

注：COL_AVG_LEN 和 LAST_GATHERED 两个字段在 V7.1.5.173 版本和之后的版本都能看到。如果使用了该版本及以后的版本服务器后，需要再退回到之前版本的服务器，那么需要在新版本上执行 SP_UPDATE_SYSSTATS(0)并正常退出之后，才能使用老版本的服务器。另外，SP_UPDATE_SYSSTATS (99)可以在 SYSSTATS 表上增加这两个列，对老库进行升级。SP_UPDATE_SYSSTATS 详细使用方法请参考《DM7_SQL 使用手册》。

6. SYSDUAL

为不带表名的查询而设，用户一般不需查看。

序号	列	数据类型	说明
1	ID	INTEGER	始终为 1

7. SYSTEXTS

存放字典对象的文本信息。

序号	列	数据类型	说明
1	ID	INTEGER	所属对象的 ID
2	SEQNO	INTEGER	视图对象文本的信息的含义，0 表示视图定义，1 表示视图的查询子句

3	TXT	TEXT	文本信息

8. SYSGRANTS

记录系统中权限信息。

序号	列	数据类型	说明
1	URID	INTEGER	被授权用户/角色 ID
2	OBJID	INTEGER	授权对象 ID，对于数据库权限为-1
3	COLID	INTEGER	表/视图列 ID，非列权限为-1
4	PRIVID	INTEGER	权限 ID
5	GRANTOR	INTEGER	授权者 ID
6	GRANTABLE	CHAR(1)	权限是否可转授，Y 可转授，N 不可转授

9. SYSAUDIT

记录系统中的审计设置。

序号	列	数据类型	说明
1	LEVEL	SMALLINT	审计级别
2	UID	INTEGER	用户 ID
3	TVPID	INTEGER	表/视图/触发器/存储过程函数 ID
4	COLID	SMALLINT	列 ID
5	TYPE	SMALLINT	审计类型
6	WHENEVER	SMALLINT	审计情况

10. SYSAUDITRULES

记录系统中审计规则的信息。

序号	列	数据类型	说明
1	ID	INTEGER	规则 ID
2	RULENAME	VARCHAR(128)	规则名
3	USERID	INTEGER	用户 ID
4	SCHID	INTEGER	模式 ID
5	OBJID	INTEGER	操作对象 ID
6	COLID	SMALLINT	列 ID
7	OPTYPE	SMALLINT	操作类型
8	WHENEVER$	SMALLINT	审计情况
9	ALLOW_IP	VARCHAR(1024)	允许的 IP
10	ALLOW_DT	VARCHAR(1024)	时间段
11	INTERVAL$	INTEGER	时间间隔
12	TIMES	INTEGER	操作次数

11. SYSHPARTTABLEINFO

记录系统中分区表的信息。

序号	列	数据类型	说明
1	BASE_TABLE_ID	INTEGER	基表 ID
2	PART_TABLE_ID	INTEGER	分区表 ID
3	PARTITION_TYPE	VARCHAR(10)	分区类型
4	PARTITION_NAME	VARCHAR(128)	分区名
5	HIGH_VALUE	VARBINARY(8188)	LIST 分区的临界值； 范围分区的分区值； 哈希分区此值为 NULL

6	INCLUDE_HIGH_VALUE	CHAR(1)	对 LIST 分区表示分区是否包含临界值； 对范围分区始终为 1；对哈希分区此值为 0
7	RESVD1	INTEGER	对子表记录：同层的第一个(最左边)子表 id 对模板记录：该模板中的子分区个数
8	RESVD2	INTEGER	保留
9	RESVD3	INTEGER	保留
10	RESVD4	VARCHAR(128)	保留
11	RESVD5	VARCHAR(2000)	保留

12. SYSMACPLYS

记录策略定义。

序号	列	数据类型	说明
1	ID	INTEGER	策略 ID
2	NAME	VARCHAR(128)	策略名

13. SYSMACLVLS

记录策略的等级。

序号	列	数据类型	说明
1	PID	INTEGER	策略 ID
2	ID	SMALLINT	等级 ID
3	NAME	VARCHAR(128)	等级名

14. SYSMACCOMPS

记录策略的范围。

序号	列	数据类型	说明
1	PID	INTEGER	策略 ID
2	ID	SMALLINT	范围 ID
3	NAME	VARCHAR(128)	范围名

15. SYSMACGRPS

记录策略所在组的信息。

序号	列	数据类型	说明
1	PID	INTEGER	策略 ID
2	ID	SMALLINT	组 ID
3	PARENTID	SMALLINT	父节点 ID
4	NAME	VARCHAR(128)	组名

16. SYSMACLABELS

记录策略的标记信息。

序号	列	数据类型	说明
1	PID	INTEGER	策略 ID
2	ID	INTEGER	标记 ID
3	LABEL	VARCHAR(4000)	标记信息

17. SYSMACTABPLY

记录表策略信息。

序号	列	数据类型	说明
1	TID	INTEGER	表 ID
2	PID	INTEGER	策略 ID

3	COLID	SMALLINT	列 ID
4	OPTIONS	BYTE	可见性

18. SYSMACUSRPLY

记录用户的策略信息。

序号	列	数据类型	说明
1	UID	INTEGER	用户 ID
2	PID	INTEGER	策略 ID
3	MAXREAD	INTEGER	最大读标记 ID
4	MINWRITE	INTEGER	最小写标记 ID
5	DEFTAG	INTEGER	默认标记 ID
6	ROWTAG	INTEGER	行级标记 ID
7	PRIVS	BYTE	特权

19. SYSMACOBJ

记录扩展客体标记信息。

序号	列	数据类型	说明
1	OBJID	INTEGER	对象 ID
2	COLID	SMALLINT	列 ID
3	PID	INTEGER	策略 ID
4	TAG	INTEGER	标记 ID

20. SYSCOLCYT

记录列的加密信息。

序号	列	数据类型	说明
1	TID	INTEGER	表 ID
2	CID	SMALLINT	列 ID
3	ENC_ID	INTEGER	加密类型 ID
4	ENC_TYPE	CHAR(1)	加密类型
5	HASH_ID	INTEGER	哈希算法 ID
6	HASH_TYPE	CHAR(1)	是否加盐
7	CIPHER	VARCHAR(1024)	密钥

21. SYSACCHISTORIES

记录登录失败的历史信息。

序号	列	数据类型	说明
1	LOGINID	INTEGER	登录 ID
2	LOGINNAME	VARCHAR(128)	登录名
3	TYPE$	INTEGER	登录类型
4	ACCPIP	VARCHAR(128)	访问 IP
5	ACCDT	DATETIME	访问时间

22. SYSPWDCHGS

记录密码的修改信息。

序号	列	数据类型	说明
1	LOGINID	INTEGER	登录 ID
2	OLD_PWD	VARCHAR(48)	旧密码
3	NEW_PWD	VARCHAR(48)	新密码
4	MODIFIED_TIME	TIMESTAMP	修改日期

23. SYSCONTEXTINDEXES

记录全文索引的信息。

序号	列	数据类型	说明
1	NAME	VARCHAR(128)	索引名
2	ID	INTEGER	索引号
3	TABLEID	INTEGER	基表号
4	COLID	SMALLINT	列号
5	UPD_TIMESTAMP	TIMESTAMP	索引更新时间
6	TIID	INTEGER	CTI$INDEX_NAME$I 表 ID
7	TRID	INTEGER	CTI$INDEX_NAME$R 表 ID
8	TPID	INTEGER	CTI$INDEX_NAME$P 表 ID
9	WSEG_TYPE	SMALLINT	分词参数类型
10	RESVD1	SMALLINT	保留
11	RESVD2	INTEGER	保留
12	RESVD3	INTEGER	保留
13	RESVD4	VARCHAR(1000)	保留

24. SYSTABLECOMMENTS

记录表或视图的注释信息。

序号	列	数据类型	说明
1	SCHNAME	VARCHAR(128)	模式名
2	TVNAME	VARCHAR(128)	表/视图名
3	TABLE_TYPE	VARCHAR(10)	对象类型
4	COMMENT$	VARCHAR(40000)	注释信息

25. SYSCOLUMNCOMMENTS

记录列的注释信息。

序号	列	数据类型	说明
1	SCHNAME	VARCHAR(128)	模式名
2	TVNAME	VARCHAR(128)	表/视图名
3	COLNAME	VARCHAR(128)	列名
4	TABLE_TYPE	VARCHAR(10)	对象类型
5	COMMENT$	VARCHAR(4000)	注释信息

26. SYSUSERS

记录系统中用户信息。

序号	列	数据类型	说明
1	ID	INTEGER	用户 ID
2	PASSWORD	VARCHAR(128)	用户口令
3	AUTHENT_TYPE	INTEGER	用户认证方式： NDCT_DB_AUTHENT/NDCT_OS_AUTHENT/NDCT_NET_AUTHENT/NDCT_UNKOWN_AUTHENT
4	SESS_PER_USER	INTEGER	在一个实例中，一个用户可以同时拥有的会话数量
5	CONN_IDLE_TIME	INTEGER	用户会话的最大空闲时间
6	FAILED_NUM	INTEGER	用户登录失败次数限制

7	LIFE_TIME	INTEGER	一个口令在终止使用前可以使用的天数
8	REUSE_TIME	INTEGER	一个口令在可以重新使用之前必须经过的天数
9	REUSE_MAX	INTEGER	一个口令在可以重新使用前必须改变的次数
10	LOCK_TIME	INTEGER	用户口令锁定时间
11	GRACE_TIME	INTEGER	用户口令过期后的宽限时间
12	LOCKED_STATUS	SMALLINT	用户登录是否锁定：LOGIN_STATE_UNLOCKED/LOGIN_STATE_LOCKED
13	LATEST_LOCKED	TIMESTAMP(19)	用户最后一次的锁定时间
14	PWD_POLICY	INTEGER	用户口令策略：NDCT_PWD_POLICY_NULL/NDCT_PWD_POLICY_1/NDCT_PWD_POLICY_2/NDCT_PWD_POLICY_3/NDCT_PWD_POLICY_4/NDCT_PWD_POLICY_5
15	RN_FLAG	INTEGER	是否只读
16	ALLOW_ADDR	VARCHAR(500)	允许的 IP 地址
17	NOT_ALLOW_ADDR	VARCHAR(500)	不允许的 IP 地址
18	ALLOW_DT	VARCHAR(500)	允许登录的时间段
19	NOT_ALLOW_DT	VARCHAR(500)	不允许登录的时间段
20	LAST_LOGIN_DTID	VARCHAR(128)	上次登录时间
21	LAST_LOGIN_IP	VARCHAR(128)	上次登录 IP 地址
22	FAILED_ATTEMPS	INTEGER	将引起一个账户被锁定的连续注册失败的次数
23	ENCRYPT_KEY	VARCHAR(256)	用户登录的存储加密密钥

27．SYSOBJINFOS

记录对象的依赖信息。

序号	列	数据类型	说明
1	ID	INTEGER	被依赖类的 ID
2	TYPE$	VARCHAR(100)	对象依赖类型
3	INT_VALUE	INTEGER	对象类型对应的值
4	STR_VALUE	VARCHAR(2048)	如果是域对象，表示 DOMAIN+域 ID。其他对象暂未利用
5	BIN_VALUE	VARBINARY(2048)	暂未利用

28．SYSRESOURCES

记录用户使用系统资源的限制信息。

序号	列	数据类型	说明
1	ID	INTEGER	用户 ID
2	CPU_PER_CALL	INTEGER	用户的一个请求能够使用的 CPU 时间上限（单位：秒）
3	CPU_PER_SESSION	INTEGER	一个会话允许使用的 CPU 时间上限（单位：秒）
4	MEM_SPACE	INTEGER	会话占有的私有内存空间上限（单位：MB）
5	READ_PER_CALL	INTEGER	每个请求能够读取的数据页数

6	READ_PER_SESSION	INTEGER	一个会话能够读取的总数据页数上限
7	INFO1	VARCHAR(256)	一个会话连接、访问和操作数据库服务器的时间上限（单位：10 分钟）

29. SYSCOLINFOS

记录列的附加信息，例如是否虚拟列。

序号	列	数据类型	说明
1	ID	INTEGER	表 ID
2	COLID	SMALLINT	列 ID
3	INFO1	INTEGER	第一位表示是否虚拟列
4	INFO2	INTEGER	备用
5	INFO3	INTEGER	备用

30. SYSUSERINI

记录定制的 INI 参数。

序号	列	数据类型	说明
1	USER_NAME	VARCHAR(256)	用户名
2	PARA_NAME	VARCHAR(256)	ini 参数名
3	INT_VALUE	BIGINT	整型参数的值
4	DOUBLE_VALUE	FLOAT	浮点类型参数值
5	STRING_VALUE	VARCHAR(4000)	字符类型参数的值。

31. SYSDEPENDENCIES

记录对象间的依赖关系。

序号	列	数据类型	说明
1	ID	INTEGER	对象 ID
2	TYPE$	VARCHAR(17)	对象类型，包括：TABLE，VIEW,MATERIALIZED VIEW,INDEX,PROCEDURE,FUNCTION,TRIGGER,SEQUENCE,CLASS,JCLASS,TYPE,PACKAGE,SYNONYM,DOMAIN
3	REFED_ID	INTEGER	被引用对象 ID
4	REFED_TYPE$	VARCHAR(17)	被引用对象类型，包括类型同 TYPE$
5	DEPEND_TYPE	VARCHAR(4)	默认为"HARD",当 TYPE$为 MATERIALIZED VIEW 和 INDEX 时，其值为"REF"

32. SYSINJECTHINT

记录已指定的 SQL 语句和对应的 HINT。

序号	列	数据类型	说明
1	NAME	VARCHAR(128)	规则名称
2	DESCRIPTION	VARCHAR(256)	规则的详细描述
3	VALIDATE	VARCHAR(5)	规则是否生效，取值 TRUE/FALSE
4	SQL_TEXT	TEXT	规则中的 SQL 语句
5	HINT_TEXT	TEXT	为 SQL 语句指定的 HINT 内容
6	CREATOR	VARCHAR(128)	规则创建人
7	CRTDATE	DATETIME	规则创建时间

8	INFO1	INT	保留字段
9	INFO2	VARBINARY(128)	保留字段
10	INFO3	VARBINARY(1024)	保留字段

附录 2 动态性能视图

动态视图存储着数据库的配置及数据库中变化的活动状态信息。在 MPP 环境下，动态视图只存储当前节点的信息。

与备份还原相关的动态视图此处不作介绍，请参考《DM7 备份与还原》。

1） 资源管理

1. V$DICT_CACHE_ITEM

显示字典缓存中的字典对象信息。

序号	列	数据类型	说明
1	TYPE	VARCHAR(128)	字典对象的类型。类型如下：DB、TABLE、VIEW、INDEX、USER、ROLE、PROC、TRIGGER、CONSTRAINT、SCHEMA、SEQUENCE、DBLINK、SYSROLE、PACKAGE、OBJECT、SYNOM、CRYPT、CIPHER、IDENTITY、SYS PRIVILEGE、OBJ PRIVILEGE、POLICY、RULE、COLUMN、DOMAIN、CHARSET、COLLATION、CONTEXT INDEX、REGEXP REWRITE、NORMAL REWRITE、CONTEXT、DIRECTORY
2	ID	INTEGER	字典对象 ID
3	NAME	VARCHAR(128)	字典对象的名称
4	SCHID	INTEGER	字典对象所属模式
5	PID	INTEGER	父 ID

2. V$DICT_CACHE

显示字典缓存信息。

序号	列	数据类型	说明
1	ADDR	VARCHAR(20)	地址
2	POOL_ID	INTEGER	缓冲区 ID
3	TOTAL_SIZE	INTEGER	总大小
4	USED_SIZE	INTEGER	实际使用大小
5	DICT_NUM	INTEGER	字典对象总数

3. V$BUFFERPOOL

页面缓冲区动态性能表，用来记录页面缓冲区结构的信息。

序号	列	数据类型	说明
1	ID	INTEGER	缓冲区 ID
2	NAME	VARCHAR(20)	缓冲区名称 NORMAL/KEEP/RECYCLE/FAST
3	PAGE_SIZE	INTEGER	基缓冲区页大小，不包括扩展池页
4	N_PAGES	INTEGER	页数
5	N_FIXED	INTEGER	数据页被引用的次数
6	FREE	INTEGER	空闲页数目
7	N_DIRTY	INTEGER	脏页数目
8	N_CLEAR	INTEGER	非空闲页数目
9	N_TOTAL_PAGES	INTEGER	页大小，包括扩展池页

10	N_MAX_PAGES	INTEGER	最多的页数
11	N_LOGIC_READS	INTEGER	READ 命中的次数
12	N_DISCARD	INTEGER	淘汰的页数
13	N_PHY_READS	INTEGER	READ 未命中的次数
14	N_PHY_M_READS	INTEGER	READ 为命中，批量读的次数
15	RAT_HIT	FLOAT	命中率
16	N_EXP_BUFFERPOOL	INTEGER	扩展缓冲区个数
17	N_UPD_REMOVE	INTEGER	从 update 链表删除页总数
18	N_PHY_WRITE	INTEGER	物理写入磁盘总数
19	N_UPD_PUT	INTEGER	RAC 远程读取数据后，加入 update 链表总数
20	N_UPD_SEARCH	INTEGER	RAC 远程读取数据后，查找 update 链表插入位置扫描总数

4． V$BUFFER_LRU_FIRST

显示所有缓冲区 LRU 链首页信息。

序号	列	数据类型	说明
1	POOL_ID	INTEGER	缓冲区 ID
2	SPACE_ID	INTEGER	表空间 ID
3	FILE_ID	INTEGER	文件 ID
4	PAGE_NO	INTEGER	数据在文件中的页号
5	RWLOCK	BIGINT	线程锁地址
6	PAGE	BIGINT	数据页对应的内存块地址
7	HASH	BIGINT	HASH 地址
8	LSN	BIGINT	页的当前 LSN
9	LRU_NEXT	BIGINT	下一个非空闲页地址
10	LRU_PREV	BIGINT	上一个非空闲页地址
11	UPD_NEXT	BIGINT	下一个脏页地址
12	UPD_PREV	BIGINT	上一个脏页地址
13	N_FIXED	INTEGER	数据页被引用的次数
14	STATUS	INTEGER	页状态。1：空闲；2：使用；4:正在读；5：正在写
15	ACCESS_CNT	INTEGER	正在访问的页号
16	FIRST_LSN	BIGINT	第一次被修改时对应的日志文件号
17	FIRST_FIL	INTEGER	第一次被修改时对应的已经刷盘的最新文件号
18	FIRST_OFF	BIGINT	第一次被修改时对应的已经刷盘的最新文件偏移

5． V$BUFFER_UPD_FIRST

显示所有缓冲区 UPDATE 链首页信息。

序号	列	数据类型	说明
1	POOL_ID	INTEGER	缓冲区 ID
2	SPACE_ID	INTEGER	表空间 ID
3	FILE_ID	INTEGER	文件 ID
4	PAGE_NO	INTEGER	数据在文件中的页号
5	RWLOCK	BIGINT	线程锁地址
6	PAGE	BIGINT	数据页对应的内存块地址
7	HASH	BIGINT	HASH 地址
8	LSN	BIGINT	页的当前 LSN
9	LRU_NEXT	BIGINT	下一个非空闲页地址

10	LRU_PREV	BIGINT	上一个非空闲页地址
11	UPD_NEXT	BIGINT	下一个脏页地址
12	UPD_PREV	BIGINT	上一个脏页地址
13	N_FIXED	INTEGER	数据页被引用的次数
14	STATUS	INTEGER	页状态。1：空闲；2：使用；4:正在读；5：正在写
15	ACCESS_CNT	INTEGER	正在访问的页号
16	FIRST_LSN	BIGINT	第一次被修改时对应的日志文件号
17	FIRST_FIL	INTEGER	第一次被修改时对应的已经刷盘的最新文件号
18	FIRST_OFF	BIGINT	第一次被修改时对应的已经刷盘的最新文件偏移

6． V$BUFFER_LRU_LAST

显示所有缓冲区 LRU 链末页信息。

序号	列	数据类型	说明
1	POOL_ID	INTEGER	缓冲区 ID
2	SPACE_ID	INTEGER	表空间 ID
3	FILE_ID	INTEGER	文件 ID
4	PAGE_NO	INTEGER	数据在文件中的页号
5	RWLOCK	BIGINT	线程锁地址
6	PAGE	BIGINT	数据页对应的内存块地址
7	HASH	BIGINT	HASH 地址
8	LSN	BIGINT	页的当前 LSN
9	LRU_NEXT	BIGINT	下一个非空闲页地址
10	LRU_PREV	BIGINT	上一个非空闲页地址
11	UPD_NEXT	BIGINT	下一个脏页地址
12	UPD_PREV	BIGINT	上一个脏页地址
13	N_FIXED	INTEGER	数据页被引用的次数
14	STATUS	INTEGER	页状态。1：空闲；2：使用；4:正在读；5：正在写
15	ACCESS_CNT	INTEGER	正在访问的页号
16	FIRST_LSN	BIGINT	第一次被修改时对应的日志文件号
17	FIRST_FIL	INTEGER	第一次被修改时对应的已经刷盘的最新文件号
18	FIRST_OFF	BIGINT	第一次被修改时对应的已经刷盘的最新文件偏移

7． V$BUFFER_UPD_LAST

显示所有缓冲区 UPDATE 链末页信息。

序号	列	数据类型	说明
1	POOL_ID	INTEGER	缓冲区 ID
2	SPACE_ID	INTEGER	表空间 ID
3	FILE_ID	INTEGER	文件 ID
4	PAGE_NO	INTEGER	数据在文件中的页号
5	RWLOCK	BIGINT	线程锁地址
6	PAGE	BIGINT	数据页对应的内存块地址
7	HASH	BIGINT	hash 地址
8	LSN	BIGINT	页的当前 LSN
9	LRU_NEXT	BIGINT	下一个非空闲页地址
10	LRU_PREV	BIGINT	上一个非空闲页地址
11	UPD_NEXT	BIGINT	下一个脏页地址
12	UPD_PREV	BIGINT	上一个脏页地址

13	N_FIXED	INTEGER	数据页被引用的次数
14	STATUS	INTEGER	页状态。1：空闲；2：使用；4:正在读；5：正在写
15	ACCESS_CNT	INTEGER	正在访问的页号
16	FIRST_LSN	BIGINT	第一次被修改时对应的日志文件号
17	FIRST_FIL	INTEGER	第一次被修改时对应的已经刷盘的最新文件号
18	FIRST_OFF	BIGINT	第一次被修改时对应的已经刷盘的最新文件偏移

8． V$CACHEITEM

显示缓冲区中缓冲项的相关信息。在 ini 参数 USE_PLN_POOL !=0 时才统计。

序号	列	数据类型	说明
1	ADDRESS	BIGINT	CACHE 项的地址
2	TYPE$	VARCHAR	CACHE 项的类型：SQL，计划（PLN），结果集（RS）
3	OVERFLOW	CHAR	是否溢出
4	IN_POOL	CHAR	是否在内存池中
5	DISABLED	CHAR	是否可用
6	N_FIXED	INTEGER	此缓冲项被引用的次数
7	TS_VALUE	INTEGER	时间戳
8	ITEM_SIZE	BIGINT	缓存节点大小（字节数）
9	N_HIT	INTEGER	节点命中次数
10	N_DIS_FIXED	INTEGER	内部保留字段

9． V$CACHERS

显示结果集缓冲区的相关信息。在 ini 参数 USE_PLN_POOL !=0 时才统计。

序号	列	数据类型	说明
1	CACHE_ITEM	BIGINT	结果集在缓冲区中的地址，对应 V$CACHEITEM 中的 ADDRESS
2	PLN	BIGINT	结果集对应查询计划，对应 V$SQL_PLAN 的 PLN_ADDR
3	N_TABLES	INTEGER	结果集涉及的表的数目
4	TABLEID	VARCHAR(256)	结果集涉及的表的 ID，用空格隔开
5	MEM_SIZE	BIGINT	缓存节点大小（字节数）
6	EXEC_TIME	INTEGER	结果集生成所消耗的时间，单位：ms

10．V$CACHESQL

显示 SQL 缓冲区中 SQL 语句的信息。在 ini 参数 USE_PLN_POOL !=0 时才统计。

序号	列	数据类型	说明
1	CACHE_ITEM	BIGINT(19)	在内存中的位置，对应 V$CACHEITEM 中的 ADDRESS
2	SQL	VARCHAR(4096)	SQL 文本
3	LEN	INTEGER	SQL 文本长度
4	HASH_VALUE	INTEGER	HASH 值

11．V$SQLTEXT

显示缓冲区中的 SQL 语句信息。

序号	列	数据类型	说明
1	SQL_ADDR	VARBINARY(8)	语句在缓存中的地址
2	SQL_ID	INTEGER	语句编号（唯一标识）
3	N_EXEC	INTEGER	语句执行次数
4	HASH_VALUE	INTEGER	语句 HASH 值
5	CMD_TYPE	VARCHAR(16)	语句类型（来自语句类型，例

			如 DRO_STMT_CTAB）
6	SQL_TEXT	VARCHAR(1000)	SQL 语句内容（如果超过 1 千字符，则分段存储）
7	SQL_NTH	INTEGER	SQL 语句段号（从 0 开始）
8	HASH	VARBINARY(8)	SQL 语句的哈希节点地址
9	LINK_ADDR	VARBINARY(8)	下一个 SQL 语句地址

12．V$SQL_PLAN

显示缓冲区中的执行计划信息。在 ini 参数 USE_PLN_POOL !=0 时才统计。

序号	列	数据类型	说明
1	PLN_ADDR	VARBINARY(8)	计划在缓存中的地址
2	HASH_VALUE	INTEGER	计划 HASH 值
3	SQL_ID	INTEGER	语句编号（唯一标识）
4	PLN_TYPE	VARCHAR(16)	计划类型
5	SQLSTR	VARCHAR(1000)	语句内容
6	RT_METHOD	VARBINARY(8)	计划的运行时方法
7	SVPT	VARCHAR(128)	计划的保存点名称
8	N_LIT_PARAS	INTEGER	常量参数个数
9	N_CLNT_PARAS	INTEGER	客户端绑定参数个数
10	N_COLS	INTEGER	涉及到的列个数
11	SEL_UPDATABLE	CHAR(1)	是否为查询更新
12	N_NDCTS	INTEGER	涉及到的字典对象个数
13	N_SUBPLNS	INTEGER	子计划个数
14	N_SUBPGS	INTEGER	子过程个数
15	PRE_COMMIT	CHAR(1)	执行 DDL 之前先提交事物
16	IS_RECURSIVE	CHAR(1)	是否为递归调用
17	BPARAM_CAN_OPT	CHAR(1)	参数是否可以优化
18	NDCT_VERSION	INTEGER	字典对象版本号
19	CAN_REUSE	CHAR(1)	计划是否可重用
20	HAS_SQL	CHAR(1)	是否有对应的 SQL 语句
21	HASH	VARBINARY(8)	计划哈希节点地址
22	SCHID	INTEGER	模式 ID
23	USER_ID	INTEGER	用户 ID
24	OBJ_ID	INTEGER	对象 ID
25	RS_CAN_CACHE	CHAR(1)	结果集是否缓存
26	RS_CAN_CLT_CACHE	CHAR(1)	客户端是否缓存结果集
27	RS_MUTEX	VARBINARY(8)	互斥量地址
28	N_TABLES	INTEGER	涉及的表个数
29	LINK_ADDR	VARBINARY(8)	下一个计划地址
30	PHD_TIME	DATETIME	生成计划时间
31	OPTIMIZER	VARCHAR(128)	优化方式
32	TABLEID	VARCHAR(256)	计划涉及的表的 ID，用空格隔开
33	SQLCACHE	BIGINT	对应 V$CACHESQL 中的 CACHE_ITEM
34	RET_CMD	SMALLINT	返回响应命令字
35	STMT_TYPE	INTEGER	SQL 语句类型

36	MEM_SIZE	BIGINT	缓存节点大小（字节数）
37	RS_CAN_CACHED_IN_RULE	CHAR	依据基本规则是否可以缓存结果集（'Y'或'N'），实际是否缓存还受结果集缓存相关参数影响，或者被手工强制设置不缓存

13. V$MEM_POOL

显示所有的内存池信息。

序号	列	数据类型	说明
1	ADDR	BIGINT	内存结构地址
2	NAME	VARCHAR(128)	内存池名称
3	IS_SHARED	CHAR(1)	是否是共享的
4	CHK_MAGIC	CHAR(1)	是否打开了内存校验
5	CHK_LEAK	CHAR(1)	是否打开了泄漏检查
6	IS_OVERFLOW	CHAR(1)	是否已经触发 BAK_POOL 的分配
7	IS_DSA_ITEM	CHAR(1)	是否是 DSA(Dameng Share Area)项目， 目前一律为 N
8	ORG_SIZE	BIGINT	初始大小，以字节数为单位
9	TOTAL_SIZE	BIGINT	当前总大小，以字节数为单位
10	RESERVED_SIZE	BIGINT	当前分配出去的大小，以字节数为单位
11	DATA_SIZE	BIGINT	当前分配出去的数据占用大小，以字节数为单位
12	EXTEND_SIZE	BIGINT	每次扩展的块大小，以字节数为单位
13	TARGET_SIZE	BIGINT	扩展的目标大小，以字节数为单位
14	EXTEND_LEN	INTEGER	扩展链长度
15	N_ALLOC	INTEGER	累计分配了几次
16	N_EXTEND_NORMAL	INTEGER	TARGET 范围内累计扩展次数
17	N_EXTEND_EXCLUSIVE	INTEGER	超过 TARGET 累计扩展次数
18	N_FREE	INTEGER	累计释放次数
19	MAX_EXTEND_SIZE	BIGINT	当前最大的扩展块，以字节数为单位
20	MIN_EXTEND_SIZE	BIGINT	当前最小的扩展块，以字节数为单位
21	FILE_NAME	VARCHAR(256)	本池创建点所在的源文件名
22	FILE_LINE	INTEGER	创建点所在的代码行
23	CREATOR	INTEGER	创建者线程号

14. V$MEM_REGINFO

显示系统当前已分配并未释放的内存信息，当 MEMORY_LEAK_CHECK 为 1 时才会在此动态视图注册信息。

序号	列	数据类型	说明
1	POOL	VARCHAR(128)	注册项来源内存池名称
2	FNO	INTEGER	文件编号
3	LINENO	INTEGER	文件中的代码行编号
4	REFNUM	INTEGER	引用次数
5	RESERVED_SIZE	BIGINT	调用点累计分配未释放的内存量，以字节数为单位

6	DATA_SIZE	BIGINT	调用点累计分配未释放的净数据量，以字节数为单位
7	FNAME	VARCHAR(256)	源文件名

15. V$GSA

显示全局 SORT 内存缓冲区的使用情况。

序号	列	数据类型	说明
1	ADDR	BIGINT	缓冲块的内存地址
2	SIZE	BIGINT	缓冲块大小，以字节数为单位

16. V$MEM_HEAP

显示系统当前内存堆的信息，仅当系统启动时 MEMORY_LEAK_CHECK 为 1 时有效。

序号	列	数据类型	说明
1	ADDR	BIGINT	内存堆对象的地址
2	FILE_NAME	VARCHAR(256)	本内存堆创建点所在的源文件名
3	FILE_LINE	INTEGER	创建点所在的代码行
4	DATA_LEN	BIGINT	内存堆占用空间大小，以字节为单位
5	BLK_LEN	INTEGER	内存堆包含的 block 个数

17. V$LARGE_MEM_SQLS

最近 1000 条使用大内存的 sql 语句。一条 sql 语句使用的内存值超过 ini 参数 LARGE_MEM_THRESHOLD，就认为使用了大内存。

序号	列	数据类型	说明
1	SESS_ID	BIGINT	SESSION 的 ID
2	SQL_ID	INT	语句的 SQL ID
3	SQL_TEXT	VARCHAR(1024)	SQL 文本
4	MEM_USED_BY_K	BIGINT	使用的内存数，以 k 为单位
5	FINISH_TIME	TIMESTAMP(0)	执行结束时间
6	N_RUNS	INT	执行次数
7	SEQNO	INTEGER	编号
8	TRX_ID	BIGINT	事务号
9	SESS_SEQ	INTEGER	会话序列号，用来唯一标识会话

18. V$SYSTEM_LARGE_MEM_SQLS

系统中使用大内存最多的 20 条 sql 语句。字段定义与 v$large_mem_sqls 相同。

19. V$SCP_CACHE

显示计划缓存池信息。

序号	列	数据类型	说明
1	SCP_ADDR	VARBINARY(8)	缓存池地址
2	PLN_CNT	BIGINT	计划缓存总数
3	RS_CNT	BIGINT	结果集缓存总数
4	SQL_CNT	BIGINT	SQL 缓存总数
5	PKGINFO_CNT	BIGINT	包信息缓存总数
6	LRU_SIZE	INTEGER	LRU 链表的大小
7	DISABLE_SIZE	INTEGER	失效缓存节点数
8	DISCARD	BIGINT	缓存对象淘汰次数
9	DISCARD_MEM	BIGINT	缓存对象淘汰的大小

10	N_ADD_FAIL	BIGINT	添加缓存失败次数

20. V$DB_SYSPRIV_CACHE

系统权限缓存信息。

序号	列	数据类型	说明
1	ID	INTEGER	字典对象的 ID
2	UID	INTEGER	用户 ID
3	DDL_PRIV	VARBINARY(512)	DDL 权限
4	DML_PRIV	VARBINARY(16)	DML 权限,由 4 个 INT 值组成 0~3 字节:is_any&is_tab 4~7 字节：!is_any&is_tab 8~11 字节：is_any&!is_tab 12~15 字节：!is_any&!is_tab

21. V$DB_OBJPRIV_CACHE

对象权限缓存信息。

序号	列	数据类型	说明
1	ID	INTEGER	字典对象的 ID
2	UID	INTEGER	用户 ID
3	COLID	INTEGER	列 ID
4	OBJ_PRIV	VARBINARY(8)	对象权限
5	COL_PRIV	VARBINARY(8)	列权限

22. V$SQL_STAT

语句级资源监控内容。记录当前正在执行的 SQL 语句的资源开销。需要 ENABLE_MONITOR=1 才开始监控。其中 5~58 列中的监控项，可以通过 SP_SET_SQL_STAT_THRESHOLD()设置监控阀值，超过阀值才开始监控。具体使用参考《DM7_SQL 语言使用手册》。

列号	列名	类型	说明
1	SESSID	BIGINT	会话 id
2	SESS_SEQ	INTEGER	会话序列号，用来唯一标识会话
3	SQL_TXT	VARCHAR(1024)	语句
4	SQL_ID	INTEGER	语句编号
5	EXEC_TIME	BIGINT	执行时间(MS)
6	PARSE_CNT	BIGINT	解析次数
7	PARSE_TIME	BIGINT	解析时间(MS)
8	HARD_PARSE_CNT	BIGINT	硬解析次数
9	HARD_PARSE_TIME	BIGINT	硬解析时间(MS)
10	SEL_SQL_CNT	BIGINT	执行的查询语句总数
11	INS_SQL_CNT	BIGINT	执行的插入语句总数
12	DEL_SQL_CNT	BIGINT	执行的删除语句总数
13	UPD_SQL_CNT	BIGINT	执行的更新语句总数
14	DDL_SQL_CNT	BIGINT	执行的 DDL 语句总数
15	SEL_IN_PL_CNT	BIGINT	执行的语句块中的查询语句总数
16	INS_IN_PL_CNT	BIGINT	执行的语句块中的插入语句总数
17	DEL_IN_PL_CNT	BIGINT	执行的语句块中的删除语句总数
18	UPD_IN_PL_CNT	BIGINT	执行的语句块中的更新语句总数
19	DYN_EXEC_CNT	BIGINT	执行的语句块中的动态执行语句总数

20	DDL_EVT_TRG_CNT	BIGINT	DDL 事件触发器触发次数
21	STMT_BF_TRG_CNT	BIGINT	语句级 BEFORE 触发器触发次数
22	STMT_AF_TRG_CNT	BIGINT	语句级 AFTER 触发器触发次数
23	ROW_BF_TRG_CNT	BIGINT	行级 BEFORE 触发器触发次数
24	ROW_AF_TRG_CNT	BIGINT	行级 AFTER 触发器触发次数
25	INSTEAD_OF_TRG_CNT	BIGINT	INSTEAD OF 触发器触发次数
26	OPTIMIZED_SORT_CNT	BIGINT	最优排序次数
27	ONE_WAY_SORT_CNT	BIGINT	单路排序次数
28	MULTI_WAY_SORT_CNT	BIGINT	多路排序次数
29	RUNTIME_OBJ_ALLOC_CNT	BIGINT	运行时对象创建次数
30	RUNTIME_OBJ_SIZE_CNT	BIGINT	运行时对象占用空间大小
31	RUNTIME_OBJ_RECLAIM_CNT	BIGINT	运行时对象回收次数
32	LONG_ROW_CVT_CNT	BIGINT	超长记录字段压缩次数
33	LOGIC_READ_CNT	BIGINT	逻辑读页次数
34	PHY_READ_CNT	BIGINT	物理读页次数
35	PHY_MULTI_READ_CNT	BIGINT	物理读多页次数
36	RECYCLE_LOGIC_READ_CNT	BIGINT	临时表空间逻辑读次数
37	RECYCLE_PHY_READ_CNT	BIGINT	临时表空间物理读次数
38	HBUF_LOGIC_READ_CNT	BIGINT	HBUF 逻辑读次数
39	HBUF_PHY_READ_CNT	BIGINT	HBUF 物理读次数
40	HBUF_PHY_WRITE_CNT	BIGINT	HBUF 物理写次数
41	HBUF_PHY_READ_SIZE	BIGINT	HBUF 物理读总大小
42	HBUF_PHY_WRITE_SIZE	BIGINT	HBUF 物理写总大小
43	UNDO_PAGE_CHANGES_CNT	BIGINT	undo 页变化次数
44	RECYCLE_PAGE_CHANGES_CNT	BIGINT	临时页变化次数
45	DATA_PAGE_CHANGES_CNT	BIGINT	数据页变化次数
46	IO_WAIT_TIME	BIGINT	I/O 等待时间(MS)
47	TAB_SCAN_CNT	BIGINT	统计全表扫描次数
48	HASH_JOIN_CNT	BIGINT	统计哈希连接的次数
49	BTR_SPLIT_CNT	BIGINT	B 树分裂次数
50	BTR_PAGE_DISCARD_CNT	BIGINT	数据页丢弃次数
51	BTR_LEVEL_DISCARD_CNT	BIGINT	B 树层丢弃次数
52	BTR_LEFT_TRY_CNT	BIGINT	B 树左移次数
53	BTR_DIRECT_UPDATE_CNT	BIGINT	B 树直接更新次数
54	BTR_INSDEL_UPDATE_CNT	BIGINT	B 树插入删除更新次数
55	BTR_UPDATE_2ND_CONFLICT_CNT	BIGINT	二级索引更新冲突次数
56	UPDATE_MVCC_RETRY_CNT	BIGINT	多版本更新重试次数
57	DELETE_MVCC_RETRY_CNT	BIGINT	多版本删除重试次数
58	MAX_MEM_USED	BIGINT	内存使用峰值(K)

23. V$SQL_STAT_HISTORY

语句级资源监控内容。记录历史 SQL 语句执行的资源开销。需要 ENABLE_MONITOR=1 才开始监控。视图的格式和 V$SQL_STAT 一样。单机最大行数为 10000。

24. V$HLDR_TABLE

记录当前系统中所有 HLDR 使用 HLDR_BUF 的情况。

列号	列名	类型	说明

1	TABLE_ID	INTEGER	装载表 ID
2	N_ALLOC	INTEGER	申请 HLDR_BUF 的次数
3	N_FAIL	INTEGER	申请 HLDR_BUF 失败的次数
4	N_WAIT	INTEGER	申请 HLDR_BUF 时等待的次数
5	WAIT_TIME	BIGINTEGER	申请 HLDR_BUF 等待的总时长
6	N_BUF	INTEGER	申请的 HLDR_BUF 的个数

2） 段簇页

25. V$SEGMENT_INFOS

显示所有的段信息。

序号	列	数据类型	说明
1	TS_ID	INTEGER	表空间 ID
2	SEG_ID	INTEGER	段 ID
3	TYPE	VARCHAR(256)	段的类型（INNER 表示内节点段，LEAF 表示叶子节点段，LIST 表示堆表段和 BLOB 表示 BLOB 段）
4	OBJ_ID	INTEGER	若段为内节点段或叶子节点段，则为索引 ID；若为 BLOB 段，则为 BLOB 所在的表 ID
5	INODE_FILE_ID	INTEGER	段的 inode 所在的页的文件号
6	INODE_PAGE_NO	INTEGER	段的 inode 所在的页的页号
7	INODE_OFFSET	INTEGER	段的 inode 所在的页的页内的偏移
8	N_FULL_EXTENT	INTEGER	段中的满簇数
9	N_FREE_EXTENT	INTEGER	段中的空闲簇数
10	N_FRAG_EXTENT	INTEGER	段中的半满簇数
11	N_FRAG_PAGE	INTEGER	段中的半满簇中总页数

26. V$SEGMENTINFO

索引叶子段信息视图。查询该视图时，一定要带 WHERE 条件，并且必须是等值条件。

序号	列	数据类型	说明
1	INDEX_ID	INTEGER	索引 ID
2	SEG_ID	INTEGER	段 ID
3	LOGIC_PAGE_NO	INTEGER	逻辑页号
4	PHY_PAGE_NO	INTEGER	物理页号
5	PHY_FIRST_PAGE_IN_EXTENT	INTEGER	物理页所属的 extent 的第一个物理页
6	FIL_ID	INTEGER	文件 ID
7	USE_BYTES	BIGINT	已使用的字节数
8	FREE_BYTES	BIGINT	空闲的字节数

27. V$BTREE_INNER_PAGES/V$BTREE_LEAF_PAGES

索引的叶子段/内节点段的页信息视图。查询该视图时，一定要带 WHERE 条件，并且必须是等值条件。如：select * from v$btree_leaf_pages where index_id = *id*;

序号	列	数据类型	说明
1	INDEX_ID	INTEGER	索引 ID
2	LEVEL	INTEGER	页所在的层数
3	GROUP_ID	INTEGER	页的表空间号
4	FILE_ID	INTEGER	页的文件号

5	PAGE_NO	INTEGER	页的页号
6	PAGE_LSN	BIGINT	页的版本号
7	N_REC	INTEGER	页的记录数

注意：在 V$BTREE_INNER_PAGES 视图中，索引 BTREE 的控制页的 level 指定为-1。

28. V$BTREE_LIST_PAGES

LIST 索引的叶子段的页信息视图。查询该视图时，一定要带 WHERE 条件，并且必须是等值条件。如：select * from v$btree_list_pages where index_id = *id*;

序号	列	数据类型	说明
1	INDEX_ID	INTEGER	索引 ID
2	LIST_NO	INTEGER	页所在的链表编号
3	GROUP_ID	INTEGER	页的表空间号
4	FILE_ID	INTEGER	页的文件号
5	PAGE_NO	INTEGER	页的页号
6	PAGE_LSN	BIGINT	页的版本号
7	N_REC	INTEGER	页的记录数

29. V$TABLE_LOB_PAGES

表中的大字段的页信息视图。查询该视图时，一定要带 WHERE 条件，并且必须是等值条件。如：select * from v$table_lob_pages where table_id = *id*;

序号	列	数据类型	说明
1	TABLE_ID	INTEGER	表 ID
2	LOB_ID	BIGINT	大字段编号
3	TYPE	INTEGER	LOB 段页的类型
4	GROUP_ID	INTEGER	页的表空间号
5	FILE_ID	INTEGER	页的文件号
6	PAGE_NO	INTEGER	页的页号
7	PAGE_LSN	BIGINT	页的版本号

注意：在 TABLE_LOB_PAGES 视图中，LOB 段的首页输出时，LOB_ID 为-1。

30. V$RESOURCE_LIMIT

显示表、用户的空间限制信息。

序号	列	数据类型	说明
1	NAME	VARCHAR(128)	表/用户名称
2	ID	INTEGER	表/用户 ID
3	TYPE	VARCHAR(10)	表/用户类型
4	SPACE_LIMIT	INTEGER	空间限制大小，以页为单位
5	SPACE_USED	INTEGER	空间实际使用大小，以页为单位

31. V$SEGMENT_PAGES

段中数据页的信息视图。查询该视图时，一定要带 WHERE 条件指定 GROUP_ID 和 SEG_ID，并且必须是等值条件。例如：select * from v$segment_pages where group_id=1 and seg_id = 200;

序号	列	数据类型	说明
1	GROUP_ID	INTEGER	表空间 ID
2	SEG_ID	INTEGER	段编号
3	FILE_ID	INTEGER	页的文件号
4	PAGE_NO	INTEGER	页的页号

32. V$PSEG_SYS

显示当前回滚段信息。

序号	列	数据类型	说明
1	N_ITEM	INTEGER	系统中存在回滚项目的个数
2	EXTENT_SIZE	INTEGER	每次新申请的回滚页数目
3	ALLOC_PAGES	BIGINT	回滚页分配的空间大小
4	EXTEND_NUM	BIGINT	回滚页分配的簇大小
5	RECLAIM_PAGES	BIGINT	回滚段重新回收的页数
6	TAB_ITEMS	BIGINT	回滚段缓存表的个数（有 storage 选项不计）
7	TAB_HASH_SIZE	BIGINT	回滚段缓存的 hash 表的大小
8	OBJ_COUNT	BIGINT	purge 涉及对象的个数
9	OBJ_HASH_SIZE	BIGINT	purge 涉及对象的 hash size

33. V$PSEG_ITEMS

显示回滚系统中当前回滚项信息。

序号	列	数据类型	说明
1	NTH	INTEGER	ITEM 序号
2	N_PAGES	INTEGER	回滚页数
3	N_EXTEND	INTEGER	扩展回滚段次数
4	N_PURGED_PAGES	INTEGER	已 PURGE 的页数
5	N_USED_PAGES	INTEGER	正在使用的页数
6	N_UREC_BYTES	BIGINT	生成回滚记录的总字节数
7	N_COMMIT_TRX	INTEGER	已经提交未 PURGE 的事务数
8	RESERVE_TIME	INTEGER	事务提交后，最长的保留时间，单位秒
9	STACK_SIZE	INTEGER	空闲回滚页堆栈中总页数
10	STACK_SP	INTEGER	空闲回滚页堆栈中空闲页数
11	MAX_COMMIT_TRX	INTEGER	已提交未 PURGE 的最大事务数
12	N_FREE_NODE	INTEGER	提交队列的空闲个数
13	N_QUEUE_ITEM	INTEGER	队列项的个数，每个项管理一组提交队列
14	FIRST_COMMIT_TIME	INTEGER	第一个提交事务的时间到当前时间的间隔
15	N_PURGING_TRX	INTEGER	正在 PURGE 的事务数

34. V$PSEG_COMMIT_TRX

显示回滚项中已提交但未 PURGE 的事务信息。

序号	列	数据类型	说明
1	ITEM_NTH	INTEGER	隶属回滚项的序号
2	TRX_ID	BIGINT	事务 ID
3	CMT_TIME	TIMESTAMP	事物提交时间
4	FPA_FILE_ID	INTEGER	起始页地址文件号。-1 表示该事务没有修改数据，所以回滚文件号为空
5	FPA_PAGE_NO	INTEGER	起始页地址页 ID。-1 表示该事务没有修改数据，所以回滚页为空

35. V$PSEG_PAGE_INFO

显示当前回滚页信息。

序号	列	数据类型	说明
1	ITEM_NTH	INTEGER	隶属回滚项的序号
2	FILE_ID	INTEGER	文件号
3	PAGE_NO	INTEGER	页号
4	TRX_ID	BIGINT	对应事务 ID
5	STATUS	INTEGER	0 ACTIVE，1 COMMIT，2 PURGE，3 PRECOMMIT
6	N_UREC	INTEGER	回滚记录个数
7	N_USED_BYTES	INTEGER	页内已使用字节数

36．V$PURGE

显示当前 PURGE 回滚段信息。

序号	列	数据类型	说明
1	OBJ_NUM	INTEGER	待 PURGE 事务个数
2	IS_RUNNING	CHAR	是否正在运行 PURGE(Y/N)
3	PURG_FOR_TS	CHAR	是否正在 PURGE 表空间（Y/N）

37．V$PURGE_PSEG_OBJ

显示 PURGE 系统中，待 PURGE 的所有 PSEG 对象信息。

序号	列	数据类型	说明
1	TRX_ID	BIGINT	事务 ID
2	NEED_PURGE	CHAR	是否需要 PURGE（Y/N）
3	TAB_NUM	INTEGER	涉及表数

38．V$PURGE_PSEG_TAB

显示待 PURGE 表信息。

序号	列	数据类型	说明
1	TRX_ID	BIGINT	事务 ID
2	TAB_ID	INTEGER	表 ID
3	GROUP_ID	INTEGER	控制页所在表空间 ID
4	FILE_ID	INTEGER	控制页文件 ID
5	PAGE_NO	INTEGER	控制页编号
6	TAB_TYPE	INTEGER	表类型：0 普通表、1 全局临时表、2 本地临时表、4 垂直分区表主表、5 垂直分区表子表、6 范围分区表主表、7 范围分区表子表、8HASH 分区表主表、9HASH 分区表子表、10 位图连接索引表、11LIST 分区表主表、12LIST 分区表子表、13 外部表、14 记录类型数组所用的临时表、15 DBLINK 远程表、19HUGE TABLE、24HUGE 表范围分区表主表、25HUGE 表范围分区表子表、26HUGE 表 HASH 分区表主表、27HUGE 表 HASH 分区表子表、28HUGE 表 LIST 分区表主表、29HUGE 表 LIST 分区表子表、32 位图索引表
7	ROW_COUNT	BIGINT	insert/delete 操作影响表行数。每插入一行加 1，每删除一行减 1

3) 数据库信息

39. V$LICENSE

显示 LICENSE 信息，用来查询当前系统的 LICENSE 信息。

序号	列	数据类型	说明
1	LIC_VERSION	VARCHAR(256)	许可证版本号
2	SERIES_NO	VARCHAR(256)	LICENSE 文件序列号
3	SERVER_SERIES	VARCHAR(256)	服务器颁布类型。P：个人版、S：标准版、E：企业版、A：安全版、T:可信版、X：定制版
4	SERVER_TYPE	VARCHAR(256)	服务器发布类型。1：正式版、2：测试版、3：试用版、4：其他
5	SERVER_VER	VARCHAR(256)	服务器版本号
6	EXPIRED_DATE	DATE	有效日期
7	AUTHORIZED_CUSTOMER	VARCHAR(256)	用户名称
8	AUTHORIZED_USER_NUMBER	BIGINT	授权用户数
9	CONCURRENCY_USER_NUMBER	BIGINT	并发连接数
10	MAX_CPU_NUM	BIGINT	最大 CPU 数目
11	NOACTIVE_DEADLINE	DATE	未激活状态截止日期
12	HARDWARE_ID	VARCHAR(256)	绑定的硬件编码，根据 HARDWARE_TYPE 不同，此项内容对应的获取和检测方式不同
13	CHECK_CODE	VARCHAR(16)	校验码
14	PRODUCT_TYPE	VARCHAR(8)	产品类型：内容为：DM7/DM6/DMETLV4/DMETLV3/DMHSV3
15	PROJECT_NAME	VARCHAR(128)	项目名称
16	CPU_TYPE	VARCHAR(24)	授权运行的 CPU 类型
17	OS_TYPE	VARCHAR(24)	授权运行的操作系统
18	MAX_CORE_NUM	INTEGER	授权最大 CPU 核个数，0 表示无限制
19	HARDWARE_TYPE	VARCHAR(24)	硬件绑定类型。1：MAC 地址、 2：cpu id、3：harddriver id
20	CLUSTER_TYPE	VARCHAR(24)	授权使用的集群类型，格式为字符串“XXXX”，每一位上 0 表示禁止，1 表示授权使用 第 1 个字符：表示数据守护 第 2 个字符：表示 MPP 第 3 个字符：表示读写分离 第 4 个字符：表示 DSC 例如，“0010”表示授权该可作为读写分离集群的节点使用
21	DATE_GEN	DATE	KEY 的生成日期
22	SERVER_SERIES_NAME	VARCHAR(128)	定制版版本名称
23	TABLE_RECORD_NUMBER_LIMIT	INTEGER	单表最大数据行数限制，以万为单位，0 表示无限制
24	TOTAL_SPACE_LIMIT	INTEGER	数据总空间大小限制，以 G 为单位，0 表示无限制

40. V$VERSION

显示版本信息，包括服务器版本号与 DB 版本号。

序号	列	数据类型	说明
1	BANNER	VARCHAR(80)	版本标识

41. V$DATAFILE

显示数据文件信息。

序号	列	数据类型	说明
1	GROUP_ID	INTEGER	所属的表空间 ID
2	ID	INTEGER	数据库文件 ID
3	PATH	VARCHAR (256)	数据库文件路径
4	CLIENT_PATH	VARCHAR (256)	数据库文件路径，专门提供给客户端
5	CREATE_TIME	TIMESTAMP(0)	数据库文件创建时间
6	STATUS$	TINYINT	状态
7	RW_STATUS	TINYINT	读写状态：1 读，2 写
8	LAST_CKPT_TIME	TIMESTAMP(0)	最后一次检查点时间。只对 online 的 DB 做统计，否则就是空
9	MODIFY_TIME	TIMESTAMP(0)	文件修改时间
10	MODIFY_TRX	BIGINT	修改事务
11	TOTAL_SIZE	BIGINT	总大小。单位：页数
12	FREE_SIZE	BIGINT	空闲大小。单位：页数
13	FREE_PAGE_NO	BIGINT	数据文件中连续空白页的起始页号
14	PAGES_READ	BIGINT	读页
15	PAGES_WRITE	BIGINT	写页
16	PAGE_SIZE	INTEGER	页大小，以 BYTE 为单位
17	READ_REQUEST	INTEGER	读请求
18	WRITE_REQUEST	INTEGER	写请求
19	MAX_SIZE	INTEGER	文件最大大小，以 M 为单位
20	AUTO_EXTEND	INTEGER	是否支持自动扩展：1 支持，0 不支持
21	NEXT_SIZE	INTEGER	文件每次扩展大小，以 M 为单位
22	MIRROR_PATH	VARCHAR(256)	镜像文件路径

42. V$DATABASE

显示数据库信息。

序号	列	数据类型	说明
1	NAME	VARCHAR (128)	数据库名称。
2	CREATE_TIME	TIMESTAMP	数据库创建时间
3	ARCH_MODE	CHAR(1)	归档模式：归档或不归档。默认为不归档
4	LAST_CKPT_TIME	TIMESTAMP(0)	最后一次检查点时间。只对 ONLINE 的 DB 做统计，否则就是空
5	STATUS$	TINYINT	状态。1：启动；2：启动，redo 完成；3：MOUNT；4：打开；5：挂起；6：关闭
6	ROLE$	TINYINT	角色。0：普通；1：主库；2：备库
7	MAX_SIZE	BIGINT	最大大小。0 代表只受操作系统限制
8	TOTAL_SIZE	BIGINT	总大小
9	RAC_NODES	INTEGER	RAC集群系统中的实例总数
10	OPEN_COUNT	INTEGER	数据库open次数
11	STARTUP_COUNT	BIGINT	数据库启动次数

12	LAST_STARTUP_TIME	TIMESTAMP	数据库最近一次启动时间

43. V$IID

显示下一个创建的数据库对象的 ID。该视图提供用户可以查询下一个创建对象的 ID 的值，可以方便用户查询预知自己所要建立对象的信息。

序号	列	数据类型	说明
1	NAME	VARCHAR(100)	对象名称
2	VALUE	BIGINT	对象值

44. V$INSTANCE

显示实例信息。

序号	列	数据类型	说明
1	NAME	VARCHAR (128)	实例名称
2	HOST_NAME	VARCHAR(128)	主机名称
3	INSTANCE_NAME	VARCHAR (128)	实例名称
4	INSTANCE_NUMBER	INTEGER	实例ID，单节点默认1，MPP或RAC环境下为实例序号加1
5	SVR_VERSION	VARCHAR (128)	服务器版本
6	DB_VERSION	VARCHAR (40)	数据库版本
7	START_TIME	TIMESTAMP	服务器启动时间
8	STATUS$	VARCHAR(128)	系统状态
9	MODE$	VARCHAR(128)	模式
10	OGUID	INTEGER	控制文件的 OGUID
11	RAC_SEQNO	INTEGER	RAC 序号
12	RAC_ROLE	VARCHAR(32)	RAC 系统角色（MASTER/SLAVE）

45. V$RESERVED_WORDS

保留字统计表，记录保留字的分类信息。

RES_FIXED=N 的关键字，通过 ini 参数 EXCLUDE_RESERVED_WORDS 设置之后会失效，此视图不会再记录。

序号	列	数据类型	说明
1	KEYWORD	VARCHAR(30)	关键字名字
2	LENGTH	INTEGER	关键字长度
3	RESERVED	VARCHAR(1)	是否是保留字
4	RES_SQL	VARCHAR(1)	是否是 SQL 保留字，不能作 SQL 中的标识符
5	RES_PL	VARCHAR(1)	是否是 DMSQL 程序保留字，不能作 DMSQL 程序语句块中的标识符
6	RES_SCHEMA	VARCHAR(1)	是否是模式保留字，不能作模式标识符
7	RES_VARIABLE	VARCHAR(1)	是否是变量保留字，不能作变量标识符
8	RES_ALIAS	VARCHAR(1)	是否是别名保留字，不能作别名标识符
9	RES_FIXED	VARCHAR(1)	关键字是否可以 EXCLUDE，Y 不可以，N 可以

46. V$ERR_INFO

显示系统中的错误码信息。

序号	列	数据类型	说明
1	CODE	INTEGER	错误码

2	ERRINFO	VARCHAR(512)	错误码描述

47. V$HINT_INI_INFO

显示支持的 HINT 参数信息。

序号	列	数据类型	说明
1	PARA_NAME	VARCHAR(128)	参数名称
2	HINT_TYPE	VARCHAR(16)	HINT 类型，OPT：优化分析阶段使用；EXEC：执行阶段使用

4）数据库对象相关

数据库对象包括：表空间、序列、包、索引和函数等。

48. V$TABLESPACE

显示表空间信息，不包括回滚表空间信息。

序号	列	数据类型	说明
1	ID	INTEGER	表空间 ID
2	NAME	VARCHAR (128)	表空间名称
3	CACHE	VARCHAR (20)	CACHE 名
4	TYPE$	TINYINT	表空间类型：1 DB 类型，2 临时表空间
5	STATUS$	TINYINT	状态。0 ONLINE，1 OFFLINE，2 RES_OFFLINE 3 CORRUPT
6	MAX_SIZE	BIGINT	最大大小。0 代表只受操作系统限制（暂无实际意义）
7	TOTAL_SIZE	BIGINT	总大小(以页为单位)
8	FILE_NUM	INTEGER	包含的文件数
9	ENCRYPT_NAME	VARCHAR(128)	加密算法名
10	ENCRYPTED_KEY	VARCHAR(500)	加密密钥，16 进制

49. V$HUGE_TABLESPACE

显示 HUGE 表空间信息。

序号	列	数据类型	说明
1	ID	INTEGER	表空间 ID
2	NAME	VARCHAR (128)	表空间名称
3	PATHNAME	VARCHAR (256)	表空间路径

50. V$SEQCACHE

显示当前系统中缓存的序列的信息。

序号	列	数据类型	说明
1	ID	INTEGER	序列 ID
2	NAME	VARCHAR(128)	序列名
3	NEXTVAL	BIGINT	序列的值
4	CACHE_NUM	INTEGER	创建序列时指定的缓存值的个数
5	N_USED	INTEGER	已经使用的缓存值的个数
6	OVERFLOW	INTEGER	当前序列是否已溢出的标识

51. V$PKGPROCS

显示包中的方法信息。

序号	列	数据类型	说明
1	PKGID	INTEGER	包 ID
2	PKGNAME	VARCHAR(128)	包名
3	MTDID	INTEGER	方法 ID，每个包中方法从 1 开始编号
4	MTDNAME	VARCHAR(128)	方法名
5	IS_PROC	CHAR(1)	是否是过程，'Y'表示是过程，'N'表示是函数

52．V$PKGPROCPARAMS

显示包中方法的参数信息。

序号	列	数据类型	说明
1	NAME	VARCHAR(128)	参数名
2	PKGID	INTEGER	包 ID
3	MTDID	INTEGER	方法 ID
4	PARAMID	SMALLINT	参数 ID，每个方法的参数从 1 开始编号，0 表示返回值
5	TYPE$	VARCHAR(128)	参数类型
6	LENGTH$	INTEGER	参数长度
7	SCALE	SMALLINT	参数精度
8	NULLABLE	CHAR(1)	参数是否可为空
9	IOFLAG	CHAR(2)	参数输入输出标记，'I'表示输入，'O'表示输出，'IO'表示输入输出，'R'表示返回值
10	DEFVAL	VARCHAR(2000)	参数缺省值

53．V$DB_CACHE

数据字典缓存表，用于记录数据字典的实时信息。

序号	列	数据类型	说明
1	DB_ADDR	VARBINARY(8)	数据字典地址
2	POOL_ID	INTEGER	缓存池 ID
3	TOTAL_SIZE	INTEGER	缓存池总空间（字节）
4	USED_SIZE	INTEGER	实际使用的空间（字节）
5	DICT_NUM	INTEGER	缓存池中字典对象的总数
6	SIZE_LRU_DISCARD	BIGINT	所有被淘汰字典对象空间的总和（字节）
7	LRU_DISCARD	INTEGER	字典对象被淘汰的次数
8	DDL_DISCARD	INTEGER	DDL 操作导致字典对象被淘汰的次数

54．V$DB_OBJECT_CACHE

数据字典对象缓存表，用于记录数据字典中每个对象的信息。

序号	列	数据类型	说明
1	TYPE	VARCHAR(32)	字典对象类型
2	ID	INTEGER	字典对象的 ID
3	NAME	VARCHAR(128)	字典对象的名称
4	SCHID	INTEGER	所在模式的 ID
5	PID	INTEGER	所属对象的 ID
6	STATUS	VARCHAR(16)	状态
7	T_SIZE	INTEGER	对象结构所占空间
8	R_SIZE	INTEGER	对象在缓存中实际的空间

9	VERSION	INTEGER	对象的版本号
10	ID_HASH	VARBINARY(8)	对象 ID 的 HASH 值
11	NAME_HASH	VARBINARY(8)	对象 NAME 的 HASH 值
12	LOAD_TIME	DATETIME	对象被加载的时间
13	N_DECODE	INTEGER	对象 DECODE 的次数
14	N_SEARCH	INTEGER	对象 SEARCH 的次数

55. V$OBJECT_USAGE

记录索引监控信息。

序号	列	数据类型	说明
1	INDEX_NAME	VARCHAR(128)	索引名
2	SCH_NAME	VARCHAR(128)	模式名
3	TABLE_NAME	VARCHAR(128)	表名
4	MONITORING	VARCHAR(3)	是否被监控(YES\NO)，仅在 MONITOR_INDEX_FLAG=0 时有效
5	USED	VARCHAR(3)	是否被使用(YES\NO)
6	START_MONITORING	VARCHAR(19)	开始监控时间，仅在 MONITOR_INDEX_FLAG=0 时有效
7	END_MONITORING	VARCHAR(19)	停止监控时间，仅在 MONITOR_INDEX_FLAG=0 时有效

56. V$IFUN

显示数据库提供的所有函数。

序号	列	数据类型	说明
1	NAME	VARCHAR(128)	函数名
2	ID	INTEGER	函数 ID
3	ARG_NUM	INTEGER	参数个数
4	HAS_RT_FACT	CHAR(1)	是否存在运行时因素，‘Y’：是；‘N’：否
5	IS_READONLY	CHAR(1)	是否只读
6	IS_MPP_BRO	CHAR(1)	MPP 下是否需要广播
7	IS_MPP_FORBIDEN	CHAR(1)	MPP 下是否禁用该函数
8	IS_MOUNT_ONLY	CHAR(1)	系统 MOUNT 状态下是否可以使用该函数
9	IS_INCLUDE_SQL	CHAR(1)	函数中是否会执行 SQL 语句
10	EXTRA_ATTR	VARCHAR(1024)	额外的属性信息
11	CLASS$	VARCHAR(128)	函数所属的类别
12	COMMENT$	VARCHAR(1024)	函数说明

57. V$IFUN_ARG

显示数据库提供的所有函数的参数。

序号	列	数据类型	说明
1	ID	INTEGER	函数 ID
2	NAME	VARCHAR(128)	函数参数名
3	SEQ	INTEGER	参数序号
4	DATA_TYPE	VARCHAR(64)	参数数据类型
5	LEN	INTEGER	参数数据长度
6	PREC	INTEGER	数据数据精度
7	IO_TYPE	VARCHAR(8)	参数输入输出类型

8	COMMENT$	VARCHAR(1024)	参数说明

58．V$SYSSTAT

显示系统统计信息。

序号	列	数据类型	说明
1	ID	INTEGER	统计对象 ID
2	CLASSID	INTEGER	统计对象所属类别 ID。1：字典信息；2：SQL；3：事务；4：检查点；5：RLOG；6：UNDO；7：IO；8：B 树；9：网络；10：文件；11：内存；12：CPU；13：OS；14：缓冲区；15：限流控制；20：其它
3	NAME	VARCHAR(128)	统计对象名
4	STAT_VAL	BIGINT	统计值

59．V$JOBS_RUNNING

显示系统中正在执行的作业信息。

序号	列	数据类型	说明
1	SID	BIGINT	作业执行的 SESSION 的 ID
2	JOBID	INTEGER	作业号
3	FAILURES	INTEGER	作业自上一次成功以来的失败次数，暂不支持
4	LAST_DATE	DATETIME(6)	最后一次成功运行工作的时间
5	THIS_DATE	DATETIME(6)	本次运行的开始时间
6	INSTANCE	INTEGER	能够运行或正在运行作业的实例的 ID 号。 单节点上默认值为 1，如果是 DM MPP 或 DMDSC 环境，默认值为实例序号加

5）数据库配置参数

60．V$PARAMETER

显示 ini 参数和 dminit 建库参数的类型及参数值信息（当前会话值、系统值及 dm.ini 文件中的值）。

序号	字段	类型	说明
1	ID	INTEGER	ID 号
2	NAME	VARCHAR(80)	参数名字
3	TYPE	VARCHAR(200)	参数类型。 READ ONLY：手动参数，表示服务器运行过程中不可修改； IN FILE：静态参数，只可修改 ini 文件； SYS 和 SESSION：动态参数，ini 文件和内存同时可修改，其中，SYS 系统级参数；SESSION 会话级参数
4	VALUE	VARCHAR(4000)	参数的值（当前会话）
5	SYS_VALUE	VARCHAR(4000)	参数的值（系统）
6	FILE_VALUE	VARCHAR(4000)	参数的值（ini 文件）
7	DESCRIPTION	VARCHAR2(255)	参数描述

61．V$DM_INI

所有 ini 参数和 dminit 建库参数信息。

序号	列	数据类型	说明
1	PARA_NAME	VARCHAR (128)	参数名称
2	PARA_VALUE	VARCHAR (256)	系统参数值
3	MIN_VALUE	VARCHAR (256)	最小值
4	MAX_VALUE	VARCHAR (256)	最大值
5	MPP_CHK	CHAR(1)	是否检查 MPP 节点间参数一致性。Y 是，N 否
6	SESS_VALUE	VARCHAR(256)	会话参数值
7	FILE_VALUE	VARCHAR(256)	INI 文件中参数值
8	DESCRIPTION	VARCHAR(256)	参数描述

62．V$DM_ARCH_INI

归档参数信息。

序号	列	数据类型	说明
1	ARCH_NAME	VARCHAR (128)	归档名称
2	ARCH_TYPE	VARCHAR (128)	归档类型
3	ARCH_DEST	VARCHAR (256)	对于 LOCAL 归档，表示归档路径；对于 REMOTE 归档，表示本节点归档要发送到的实例名；对于其余类型的归档，表示归档目标实例名
4	ARCH_FILE_SIZE	INTEGER	归档文件大小
5	ARCH_SPACE_LIMIT	INTEGER	归档文件的磁盘空间限制，单位是 MB
6	ARCH_HANG_FLAG	INTEGER	如果本地归档时磁盘空间不够，是否让服务器挂起。无实际意义，对本地归档类型和远程归档类型显示为1，其他归档类型显示为NULL
7	ARCH_TIMER_NAME	VARCHAR (128)	对于异步归档，表示定时器名称；其余类型归档显示 NULL
8	ARCH_IS_VALID	CHAR(1)	归档状态，是否有效
9	ARCH_WAIT_APPLY	INTEGER	性能模式，是否等待重演完成，取值 0：高性能模式， 1：数据一致模式。本地归档取值 NULL
10	ARCH_INCOMING_PATH	VARCHAR(256)	对 REMOTE 归档，表示远程节点发送过来的归档在本地的保存目录；其余类型归档显示 NULL

63．V$DM_MAL_INI

MAL 参数信息。

序号	列	数据类型	说明
1	MAL_TYPE	VARCHAR (128)	MAL 名
2	INST_NAME	VARCHAR (256)	站点名
3	IP	VARCHAR (30)	IP 地址
4	MAL_PORT	INTEGER	端口号
5	INST_IP	VARCHAR (256)	对应实例 IP 地址
6	INST_PORT	INTEGER	对应实例的端口号
7	MAL_DW_PORT	INTEGER	MAL_INST_NAME实例守护进程监听端口
8	MAL_LINK_MAGIC	INTEGER	MAL链路网段标识

64．V$DM_REP_RPS_INST_NAME_INI

数据复制服务器参数信息。

序号	列	数据类型	说明
1	INST_RPS	VARCHAR (256)	RPS 服务器的实例名

65．V$DM_REP_MASTER_INFO_INI

数据复制主库参数信息。

序号	列	数据类型	说明
1	REP_ID	INTEGER	复制 ID

66. V$DM_REP_SLAVE_INFO_INI

数据复制从机参数信息。

序号	列	数据类型	说明
1	REP_ID	INTEGER	复制 ID
2	MASTER_INSTNAME	VARCHAR (256)	主站点实例名

67. V$DM_REP_SLAVE_TAB_MAP_INI

数据复制从机表对应关系参数信息。

序号	列	数据类型	说明
1	REP_ID	INTEGER	复制 ID
2	SRC_TAB_ID	INTEGER	源表 ID
3	DST_TAB_ID	INTEGER	目的表 ID

68. V$DM_REP_SLAVE_SRC_COL_INFO_INI

数据复制从机列对应关系参数信息。

序号	列	数据类型	说明
1	REP_ID	INTEGER	参数值
2	SRC_TAB_ID	INTEGER	源表 ID
3	DST_ TAB_ID	INTEGER	目的表 ID
4	COL_ID	INTEGER	列 ID
5	SQL_PL_TYPE	INTEGER	SQL 类型
6	LEN	INTEGER	SQL 长度
7	PREC	INTEGER	游标类型

69. V$DM_LLOG_INFO_INI

逻辑日志信息参数信息。

序号	列	数据类型	说明
1	LLOG_NAME	VARCHAR (128)	逻辑日志类型名
2	ID	INTEGER	ID
3	LLOG_PATH	VARCHAR (256)	路径
4	REP_FLAG	BIT	是否存在复制关系
5	REP_INSTNAME	VARCHAR (256)	复制关系从站点实例名
6	REP_TYPE	VARCHAR (128)	表示同步或异步的复制关系
7	REP_TIMER	VARCHAR (128)	表示当异步复制关系时，本地定时器的名字

70. V$DM_LLOG_TAB_MAP_INI

逻辑日志与表对应的参数信息。

序号	列	数据类型	说明
1	LLOG_ID	INTEGER	日志 ID
2	TAB_ID	INTEGER	表 ID

71. V$DM_TIMER_INI

定时器参数信息。

序号	列	数据类型	说明
1	TIMER_NAME	VARCHAR(128)	定时器名称

2	TYPE	TINYINT	类型
3	FREQ_MONTH_WEEK_INTERVAL	INTEGER	月或周的间隔
4	FREQ_SUB_INTERVAL	INTEGER	间隔天数
5	FREQ_MINUTE_INTERVAL	INTEGER	分钟间隔
6	START_TIME	TIME	开始时间
7	END_TIME	TIME	结束时间
8	DURING_START_DATE	DATETIME	有效时间段的开始日期时间
9	DURING_END_DATE	DATETIME	有效时间段的结束日期时间
10	NO_END_DATE_FLAG	CHAR(1)	无结束日期标识
11	DESCRIBE	VARCHAR(256)	描述
12	IS_VALID	CHAR(1)	是否有效
13	REPEAT_INTERVAL	VARCHAR(4000)	日历表达式

72. V$OBSOLETE_PARAMETER

已作废的 INI 信息。

序号	列	数据类型	说明
1	PARA_NAME	VARCHAR (128)	参数名称

73. V$OPTION

安装数据库时的参数设置。

序号	列	数据类型	说明
1	PARA_NAME	VARCHAR (128)	参数名称
2	PARA_VALUE	VARCHAR (256)	系统参数值

6) 日志管理

74. V$RLOG

显示日志的总体信息。通过该视图可以了解系统当前日志事务号 LSN 的情况、归档日志情况、检查点的执行情况等。

序号	列	数据类型	说明
1	CKPT_LSN	BIGINT	最近一次检查点 LSN
2	FILE_LSN	BIGINT	已经到盘上的 LSN
3	FLUSH_LSN	BIGINT	当前准备刷盘的 LSN
4	CUR_LSN	BIGINT	当前的 LSN
5	NEXT_SEQ	INTEGER	下一页页号
6	N_MAGIC	INTEGER	本次运行，日志的 magic
7	DB_MAGIC	INTEGER	数据库的 magic
8	FLUSH_PAGES	INTEGER	Flush 链表中的总页数
9	FLUSHING_PAGES	INTEGER	正在刷盘的总页数
10	CUR_FILE	INTEGER	记录刷文件前当前文件的 ID
11	CUR_OFFSET	BIGINT	记录刷文件前 cur_file 的 free
12	CKPT_FILE	INTEGER	最近一次检查点对应的当时的文件号
13	CKPT_OFFSET	BIGINT	最近一次检查点对应的当时的文件偏移
14	FREE_SPACE	BIGINT	目前可用的日志空间

15	TOTAL_SPACE	BIGINT	日志总空间
16	SUSPEND_TIME	TIMESTAMP	挂起时间戳
17	UPD_CTL_LSN	BIGINT	系统修改控制文件的 ptx->lsn
18	N_RESERVE_WAIT	INTEGER	日志空间预申请等待的个数

75. V$RLOGBUF

显示日志 BUFFER 信息。通过该视图可以查询日志 BUFFER 的使用情况，如 BUFFER 状态、总大小、已使用大小，这样可以带来帮助，如 BUFFER 剩余空间过小产生的失败等。

序号	列	数据类型	说明
1	BEGIN_LSN	BIGINT	log buf 起始 LSN
2	END_LSN	BIGINT	log buf 结束 LSN
3	TOTAL_PAGES	INTEGER	日志 buffer 的总大小页数
4	N_PAGES	INTEGER	Buffer 中剩下可用的页数

76. V$RLOGFILE

显示日志文件的具体信息。包括文件号、完整路径、文件的状态、文件大小等等。

序号	列	数据类型	说明
1	GROUP_ID	INTEGER	表空间 ID
2	FILE_ID	INTEGER	文件 ID
3	PATH	VARCHAR(256)	文件路径
4	CLIENT_PATH	VARCHAR(256)	文件路径，专门提供给客户端
5	CREATE_TIME	TIMESTAMP	创建时间
6	RLOG_SIZE	BIGINT	文件大小

77. V$ARCHIVED_LOG

显示当前实例的所有归档日志文件信息。此动态视图还有一些未列出的保留列，查询时均显示 NULL。

序号	列	数据类型	说明
1	NAME	VARCHAR(513)	归档日志文件名
2	THEAD#	BIGINT	默认为 0
3	SEQUENCE#	INTEGER	日志文件的序号
4	FIRST_CHANGE#	BIGINT	日志文件所记录日志的最小 LSN 值
5	NEXT_CHANGE#	BIGINT	日志文件所记录日志的最大 LSN 值
6	FIRST_TIME	DATETIME	日志文件所记录的日志的起始时间
7	NEXT_TIME	DATETIME	日志文件所记录的日志的截止时间
8	ARCHIVED	VARCHAR(3)	默认为 ARCHIVED
9	DELETED	VARCHAR(3)	默认为 NO
10	STATUS	VARCHAR(1)	默认为 A
11	IS_RECOVERY_DEST_FILE	VARCHAR(3)	默认为 NO

78. V$LOGMNR_LOGS

显示当前会话添加的需要分析的归档日志文件。此动态视图还有一些未列出的保留列，查询时均显示 NULL。

序号	列	数据类型	说明
1	LOG_ID	INTEGER	日志文件 ID 号
2	FILENAME	VARCHAR(512)	日志文件 ID 名

3	LOW_TIME	DATETIME	日志文件创建时间
4	HIGH_TIME	DATETIME	日志文件最后修改时间
5	DB_ID	INTEGER	日志文件的实例 ID 号
6	DB_NAME	VARCHAR(8)	日志文件的实例名
7	THREAD_ID	BIGINT	默认为 0
8	THREAD_SQN	INTEGER	日志文件的序号
9	LOW_SCN	BIGINT	日志文件最小 LSN
10	NEXT_SCN	BIGINT	日志文件最大 LSN
11	DICTIONARY_BEGIN	VARCHAR(3)	默认为 NO
12	DICTIONARY_END	VARCHAR(3)	默认为 NO
13	TYPE	VARCHAR(8)	默认为 ARCHIVED
14	BLOCKSIZE	INTEGER	默认为 512
15	FILESIZE	BIGINT	日志文件的大小
16	STATUS	INTEGER	默认为 0

79. V$LOGMNR_PARAMETERS

显示当前会话 START_LOGMNR 启动日志文件分析的参数。此动态视图还有一些未列出的保留列，查询时均显示 NULL。

序号	列	数据类型	说明
1	START_DATE	DATETIME	过滤分析归档日志的起始时间
2	END_DATE	DATETIME	过滤分析归档日志的截止时间
3	START_SCN	BIGINT	过滤分析归档日志的起始 LSN
4	END_SCN	BIGINT	过滤分析归档日志的截止 LSN
5	OPTIONS	INTEGER	分析归档日志的选项

80. V$LOGMNR_CONTENTS

显示当前会话日志分析的内容。此动态视图还有一些未列出的保留列，查询时均显示 NULL。

序号	列	数据类型	说明
1	SCN	BIGINT	当前记录的 LSN
2	START_SCN	BIGINT	当前事务的起始 LSN
3	COMMIT_SCN	BIGINT	当前事务的截止 LSN
4	TIMESTAMP	DATETIME	当前记录的创建时间
5	START_TIMESTAMP	DATETIME	当前事务的起始时间
6	COMMIT_TIMESTAMP	DATETIME	当前事务的截止时间
7	XID	BINARY(8)	当前记录的事务 ID 号，为 BIGINT 类型的事务 ID 直接转换为 16 进制，并在前面补 0
8	OPERATION	VARCHAR(32)	操作类型 OPERATION 和 OPERATION_CODE 分别为：INTERNAL 0、INSERT 1、DELETE 2、UPDATE 3、BATCH_UPDATE 4 、DDL 5 、START 6、COMMIT 7、SEL_LOB_LOCATOR 9、LOB_WRITE 10 、LOB_TRIM 11 、SELECT_FOR_UPDATE 25 、LOB_ERASE 28、MISSING_SCN 34、ROLLBACK 36、UNSUPPORTED、UNSUPPORTED 255、SEQ MODIFY 37
9	OPERATION_CODE	INTEGER	

10	ROLL_BACK	INTEGER	当前记录是否被回滚，1:是，0：否
11	TABLE_NAME	VARCHAR(128)	操作的表名
12	ROW_ID	VARCHAR(20)	对应记录的行号
13	USERNAME	VARCHAR(128)	执行这条语句的用户名
14	RBASQN	INTEGER	对应的归档日志文件号
15	RBABLK	INTEGER	RBASQN 所指日志文件的块号，从 0 开始
16	RBABYTE	INTEGER	RBABLK 所指块号的块内偏移
17	DATA_OBJ#	INTEGER	对象 ID 号
18	DATA_OBJV#	INTEGER	对象版本号
19	SQL_REDO	VARCHAR(4000)	客户端发送给数据库的 SQL 语句。
20	SQL_UNDO	VARCHAR(4000)	暂不支持。
21	RS_ID	VARCHAR(32)	记录集 ID
22	SSN	INTEGER	连续 SQL 标志。如果 SQL 长度超过单个 sql_redo 字段能存储的长度，则 SQL 会被截断成多个 SQL 片段在结果集中“连续”返回。
23	CSF	INTEGER	与 SSN 配合，最后一个片段的 csf 值为 0，其余片段的值均为 1。没因超长发生截断的 SQL 该字段值均为 0。
24	REDO_VALUE	BIGINT	用于数据挖掘新值
25	UNDO_VALUE	BIGINT	用于数据挖掘旧值
26	CSCN	BIGINT	与 COMMITSCN 一样，已过时

7）会话

81. V$CONNECT

显示活动连接的所有信息。

序号	列	数据类型	说明
1	NAME	VARCHAR (128)	连接名称
2	SADDR	BIGINT	会话地址
3	CREATE_TIME	TIMESTAMP(0)	会话创建时间
4	STATUS$	VARCHAR(128)	连接状态
5	TYPE$	VARCHAR(128)	连接类型
6	PROTOCOL_TYPE	TINYINT	协议类型
7	IP_ADDR	VARCHAR(23)	IP 地址

82. V$SESSIONS

显示会话的具体信息，如执行的 sql 语句、主库名、当前会话状态、用户名等等。

序号	列	数据类型	说明
1	SESS_ID	BIGINT	会话 ID
2	SESS_SEQ	INTEGER	会话序列号，用来唯一标识会话
3	SQL_TEXT	VARCHAR(1000)	取 sql 的头 1000 个字符
4	STATE	VARCHAR(8)	会话状态。共 6 种状态：CREATE 创建、STARTUP 启动、

			IDLE 空闲、ACTIVE 活动、WAIT 等待、UNKNOWN 未知
5	N_STMT	INTEGER	STMT 的容量
6	N_USED_STMT	INTEGER	已使用的 STMT 数量
7	SEQ_NO	INTEGER	会话上语句的序列号
8	CURR_SCH	VARCHAR(128)	当前模式
9	USER_NAME	VARCHAR(128)	当前用户
10	TRX_ID	BIGINT	事务 id
11	CREATE_TIME	DATETIME	会话创建时间
12	CLNT_TYPE	VARCHAR(128)	客户端类型（4 种）
13	TIME_ZONE	VARCHAR(6)	时区
14	CHK_CONS	CHAR	是否忽略约束检查
15	CHK_IDENT	CHAR	是否忽略自增列指定列表检查
16	RDONLY	CHAR	是否是只读会话
17	INS_NULL	CHAR	列不存在 default 选项，是否插入空值
18	COMPILE_FLAG	CHAR	是否忽略编译选项，用于 view，procedure，trigger……的编译
19	AUTO_CMT	CHAR	是否自动提交
20	DDL_AUTOCMT	CHAR	Ddl 语句是否自动提交
21	RS_FOR_QRY	CHAR	非查询语句生成结果集标记
22	CHK_NET	CHAR	是否检查网络
23	ISO_LEVEL	INTEGER	隔离级。0：读未提交；1：读提交；2：可重复读；3：串行化
24	CLNT_HOST	VARCHAR(128)	客户端主机名
25	APPNAME	VARCHAR(128)	应用程序名
26	CLNT_IP	VARCHAR(128)	客户端 IP
27	OSNAME	VARCHAR(128)	客户端操作系统名
28	CONN_TYPE	VARCHAR(20)	连接类型
29	VPOOLADDR	VARCHAR(128)	内存池的首地址
30	RUN_STATUS	VARCHAR(20)	记录运行状态，可能值为“IDLE”与“RUNNING”
31	MSG_STATUS	VARCHAR(20)	记录消息处理状态，可能值为“RECIEVE”（已接受）与“SEND”（已发送）
32	LAST_RECV_TIME	DATETIME	记录最近接收的消息时间
33	LAST_SEND_TIME	DATETIME	记录最近发送的消息时间
34	DCP_FLAG	CHAR(1)	1） 是否是通过 DCP_PORT 登录 DCP 服务器的会话，Y 表示是，N 表示否 2） 是否是通过 DCP 代理连接到 MPP 的会话，Y 表示是，N 表示否
35	THRD_ID	INTEGER	线程 ID
36	CONNECTED	INTEGER	连接是否正常：1 表示连接正常，0 表示连接已经关闭
37	PORT_TYPE	INTEGER	连接端口类型：2 表示 TCP 连接，12 表示端口已经关闭，13 表示 UDP 连接
38	SRC_SITE	INTEGER	源节点，65535 表示本地 SESSION，其他值代表 MPP 集群用户直接登录的节点号
39	MAL_ID	BIGINT	邮箱 ID
40	CONCURRENT_FLAG	INTEGER	是否占用了限流资源
41	CUR_LINENO	INTEGER	存储过程执行时记录当前的行号

42	CUR_MTDNAME	VARCHAR(128)	存储过程执行时记录当前的方法名
43	CUR_SQLSTR	VARCHAR(128)	存储过程执行时记录当前执行的 SQL 的前 100 个字符

83. V$SESSION_SYS

显示系统中会话的一些状态统计信息。

序号	列	数据类型	说明
1	N_SESS	INTEGER	会话个数
2	N_FREEING	INTEGER	正在释放的会话个数
3	ALLOW_NEW_LOGIN	INTEGER	是否允许登录
4	MAX_CONCURRENT_TRX	INTEGER	限流的最大并行数
5	CONCURRENT_TRX_MODE	INTEGER	限流模式
6	CURR_CONCURRENT_TRX	INTEGER	可用资源
7	N_WAIT	INTEGER	等待执行的任务数

84. V$OPEN_STMT

连接语句句柄表，用于记录 SESSION 上语句句柄的信息。

序号	列	数据类型	说明
1	SESS_ADDR	VARBINARY(8)	SESSION 地址
2	SESSION#	INTEGER	SESSION 的 ID
3	STMT_ID	INTEGER	STMT 的 ID
4	STMT_SEQNO	INTEGER	STMT 的执行序号
5	TYPE	VARCHAR(16)	语句类型
6	REP_TYPE	VARCHAR(16)	复制语句类型
7	EXEC_DIRECT	CHAR(1)	是否直接执行
8	PLN_ADDR	VARBINARY(8)	执行计划的地址
9	ROWCOUNT	BIGINT	影响记录的行数
10	ROWCOUNT_FOR_DML	BIGINT	DML 影响的行数
11	ROWCOUNT_REQUEST	BIGINT	请求的行数
12	BATCH_FLAG	CHAR(1)	批量处理标识
13	N_PARA_ROWS	INTEGER	多行参数时的行数
14	NTH_PARA_ROW	INTEGER	多行参数时的当前行序号
15	CUR_PARA_OFF	INTEGER	参数的当前偏移
16	PARA_BUF	VARBINARY(8)	参数的内存地址
17	SQL_STR	VARCHAR(1000)	执行语句的前 1000 个字符
18	BPARAM_CAN_OPT	CHAR(1)	绑定批量参数时是否可优化
19	CURSOR_FORWARD_ONLY	CHAR(1)	对应游标是否 FORWARD_ONLY
20	CURSOR_NAME	VARCHAR(128)	对应游标名
21	CURSOR_KEEP_NAME	CHAR(1)	对应游标是否保存名字
22	CURSOR_REF_ID	INTEGER	引用游标语句句柄 ID
23	CURSOR_REF_SEQNO	INTEGER	引用语句的执行序号
24	MPP_EXEC_ID	INTEGER	MPP 下的执行 ID
25	VM_ADDR	VARBINARY(8)	VM 地址
26	DBG_FLAG	CHAR(1)	是否 DEBUG
27	STMT_DBG	VARBINARY(8)	DEBUG 的语句句柄地址
28	MPP_DDL_FLAG	CHAR(1)	MPP 下 DDL 标识
29	REMOTE_OPERATION	CHAR(1)	是否是 DBLINK 的远程操作

30	RS_BDTA_FLAG	CHAR(1)	返回的结果集是否直接将 BDTA 打包
31	RS_BDTA_SIZE	INTEGER	返回的打包结果集时最大的行数
32	STMT_LINK	VARBINARY(8)	用于串连语句的链表地址
33	LPQ	VARBINARY(8)	并行查询地址
34	RS_INFO	VARBINARY(8)	结果集地址
35	MPLN_ADDR	VARBINARY(8)	从 EP 上保留 MPLN
36	BIND_PARAM	VARBINARY(8)	绑定参数地址
37	BAK_SYS	VARBINARY(8)	备份链表信息
38	RT_STACK	VARBINARY(8)	MPP 从 EP 的运行时信息堆栈

85. V$SESSION_HISTORY

显示会话历史的记录信息，如主库名、用户名等，与 V$SESSIONS 的区别在于会话历史记录只记录了会话一部分信息，对于一些动态改变的信息没有记录，如执行的 SQL 语句等。

序号	列	数据类型	说明
1	SESS_ID	BIGINT	会话 ID
2	SESS_SEQ	INTEGER	会话序列号，用来唯一标识会话
3	CURR_SCH	VARCHAR(128)	当前模式
4	USER_NAME	VARCHAR(128)	当前用户
5	CREATE_TIME	DATETIME	会话创建时间
6	CLNT_TYPE	VARCHAR(128)	客户类型（4 种）
7	TIME_ZONE	VARCHAR(6)	时区
8	RDONLY	CHAR	是否是只读会话
9	DDL_AUTOCMT	CHAR	DDL 语句是否自动提交
10	RS_FOR_QRY	CHAR	非查询语句生成结果集标记
11	CHK_NET	CHAR	是否检查网络
12	CLNT_HOST	VARCHAR(128)	客户主库名
13	APPNAME	VARCHAR(128)	应用程序名
14	CLNT_IP	VARCHAR(128)	客户端 IP
15	OSNAME	VARCHAR(128)	客户端操作系统名
16	CONN_TYPE	VARCHAR(20)	连接类型

86. V$CONTEXT

显示当前会话所有上下文的名字空间、属性和值。

序号	列	数据类型	说明
1	NAMESPACE	VARCHAR(30)	上下文名字空间
2	ATTRIBUTE	VARCHAR(30)	名字空间属性
3	VALUE	VARCHAR(4000)	属性值

87. V$SESSION_STAT

记录每个 session 上的相关统计信息。

序号	列	数据类型	说明
1	SESSID	BIGINT	会话 ID
2	NET_BYTES_R	BIGINT	网络收到的字节数
3	NET_BYTES_S	BIGINT	网络发送的字节数
4	PAESE_CNT	BIGINT	解析次数
5	PARSE_TIME	BIGINT	解析时间

6	HARD_PARSE_CNT	BIGINT	硬解析次数
7	HARD_PARSE_TIME	BIGINT	硬解析时间
8	SEL_SQL_CNT	BIGINT	执行的查询语句总数
9	INS_SQL_CNT	BIGINT	执行的插入语句总数
10	DEL_SQL_CNT	BIGINT	执行的删除语句总数
11	UPD_SQL_CNT	BIGINT	执行的更新语句总数
12	DDL_SQL_CNT	BIGINT	执行的 DDL 语句总数
13	SEL_IN_PL_CNT	BIGINT	执行的语句块中的查询语句总数
14	INS_IN_PL_CNT	BIGINT	执行的语句块中的插入语句总数
15	DEL_IN_PL_CNT	BIGINT	执行的语句块中的删除语句总数
16	UPD_IN_PL_CNT	BIGINT	执行的语句块中的更新语句总数
17	DYN_EXEC_CNT	BIGINT	执行的语句块中的动态执行语句总数
18	DDL_EVT_TRG_CNT	BIGINT	DDL 事件触发器触发次数
19	STMT_BF_TRG_CNT	BIGINT	语句级 BEFORE 触发器触发次数
20	STMT_AF_TRG_CNT	BIGINT	语句级 AFTER 触发器触发次数
21	ROW_BF_TRG_CNT	BIGINT	行级 BEFORE 触发器触发次数
22	ROW_AF_TRG_CNT	BIGINT	行级 AFTER 触发器触发次数
23	INSTEAD_OF_TRG_CNT	BIGINT	INSTEAD OF 触发器触发次数
24	OPTIMIZED_SORT_CNT	BIGINT	最优排序次数，最优排序指排序操作全部在排序缓冲区中完成
25	ONE_WAY_SORT_CNT	BIGINT	单路排序次数，单路排序指排序操作不能在排序缓冲区中完成，需要把待排序数据存放到磁盘一次
26	MULTI_WAY_SORT_CNT	BIGINT	多路排序次数，多路排序指排序操作不能在排序缓冲区中完成，需要把待排序数据存放到磁盘两次以上
27	RUNTIME_OBJ_ALLOC_CNT	BIGINT	运行时对象创建次数
28	RUNTIME_OBJ_SIZE_CNT	BIGINT	运行时对象占用空间大小
29	RUNTIME_OBJ_RECLAIM_CNT	BIGINT	运行时对象回收次数
30	LONG_ROW_CVT_CNT	BIGINT	超长记录字段压缩次数
31	LOGIC_READ_CNT	BIGINT	逻辑读页次数
32	PHY_READ_CNT	BIGINT	物理读页次数
33	PHY_MULTI_READ_CNT	BIGINT	物理读多页次数
34	RECYCLE_LOGIC_READ_CNT	BIGINT	临时表空间逻辑读次数
35	RECYCLE_PHY_READ_CNT	BIGINT	临时表空间物理读次数
36	HBUF_LOGIC_READ_CNT	BIGINT	HBUF 逻辑读次数
37	HBUF_PHY_READ_CNT	BIGINT	HBUF 物理读次数
38	HBUF_PHY_WRITE_CNT	BIGINT	HBUF 物理写次数
39	HBUF_PHY_READ_SIZE	BIGINT	HBUF 物理读总大小
40	HBUF_PHY_WRITE_SIZE	BIGINT	HBUF 物理写总大小
41	UNDO_PAGE_CHANGES_CNT	BIGINT	undo 页变化次数
42	RECYCLE_PAGE_CHANGES_CNT	BIGINT	临时页变化次数
43	DATA_PAGE_CHANGES_CNT	BIGINT	数据页变化次数
44	IO_WAIT_TIME	BIGINT	I/O 等待时间
45	TAB_SCAN_CNT	BIGINT	统计全表扫描次数
46	HASH_JOIN_CNT	BIGINT	统计哈希连接的次数

47	BTR_SPLIT_CNT	BIGINT	B 树分裂次数
48	BTR_PAGE_DISCARD_CNT	BIGINT	数据页丢弃次数，指被淘汰的 B 树叶子节点的数量
49	BTR_LEVEL_DISCARD_CNT	BIGINT	B 树层丢弃次数，指被淘汰的 B 树中间节点的数量
50	BTR_LEFT_TRY_CNT	BIGINT	B 树左移次数
51	BTR_DIRECT_UPDATE_CNT	BIGINT	B 树直接更新次数
52	BTR_INSDEL_UPDATE_CNT	BIGINT	B 树插入删除更新次数
53	BTR_UPDATE_2ND_CONFLICT_CNT	BIGINT	二级索引更新冲突次数
54	UPDATE_MVCC_RETRY_CNT	BIGINT	多版本更新重试次数
55	DELETE_MVCC_RETRY_CNT	BIGINT	多版本删除重试次数

8） SQL 执行相关

88. V$SQL_HISTORY

当 INI 参数 ENABLE_MONITOR=1 时，显示执行 SQL 的历史记录信息；可以方便用户经常使用的记录进行保存。

序号	列	数据类型	说明
1	SEQ_NO	INTEGER	序列号
2	SQL_ID	INTEGER	当前语句的 SQL ID
3	SESS_ID	BIGINT	会话 ID
4	SESS_SEQ	INTEGER	会话序列号，用来唯一标识会话
5	TRX_ID	BIGINT	事务 ID
6	THREAD_ID	BIGINT	线程 ID
7	TOP_SQL_TEXT	VARCHAR(1000)	栈帧中第一个 SQL
8	SEC_SQL_TEXT	VARCHAR(1000)	栈帧中第二个 SQL
9	THRD_SQL_TEXT	VARCHAR(1000)	栈帧中第三个 SQL
10	START_TIME	DATE_TIME	SQL 执行的起始时间
11	TIME_USED	BIGINT	SQL 执行所使用时间（usec）
12	IS_OVER	CHAR	是否结束
13	EXEC_ID	INTEGER	SQL 执行 ID
14	VM	BIGINT	执行 SQL 的虚拟机
15	STKFRM	BIGINT	当前的栈帧
16	STK_LEVEL	INTEGER	当前栈帧的级别
17	BYTES_DYNAMIC_ALLOCED	BIGINT	动态分配字节数
18	BYTES_DYNAMIC_FREED	BIGINT	动态释放字节数
19	CUR_SQL_NODE	BIGINT	当前的 SQL 节点
20	MAL_ID	BIGINT	邮件标识号
21	N_LOGIC_READ	INTEGER	语句逻辑读的次数
22	N_PHY_READ	INTEGER	语句物理读的次数
23	AFFECTED_ROWS	INTEGER	语句影响的行数
24	HARD_PARSE_FLAG	INTEGER	语句硬解析标记，0：软解析；1：语义解析；2：硬解析
25	MPP_EXEC_ID	INTEGER	MPP 会话句柄上的执行序号，同一个会话上的每个节点上值相同

89. V$SQL_NODE_HISTORY

通过该视图既可以查询 SQL 执行节点信息，包括 SQL 节点的类型、进入次数和使用时间等等；又可以查询所有执行的 SQL 节点执行情况，如哪些使用最频繁、耗时多少等。

当 INI 参数 ENABLE_MONITOR 和 MONITOR_SQL_EXEC 都开启时，才会记录 SQL 执行节点信息。如果需要时间统计信息，还需要打开 MONITOR_TIME。

序号	列	数据类型	说明
1	SEQ_NO	INTEGER	序列号
2	EXEC_ID	INTEGER	执行 ID
3	NODE	BIGINT	节点 ID
4	TYPE$	INTEGER	节点类型
5	BYTES_DYNAMIC_ALLOCED	BIGINT	动态分配字节数
6	BYTES_DYNAMIC_FREED	BIGINT	动态释放字节数
7	N_ENTER	INTEGER	节点进入次数
8	TIME_USED	INTEGER	节点执行所耗时
9	PLN_OP_ID	INTEGER	MPP 模式下，节点所属通讯操作符中的序号
10	BYTES_SEND	INTEGER	发送的字节数
11	BYTES_RECV	INTEGER	接收的字节数
12	ROWS_SEND	INTEGER	发送的行数
13	ROWS_RECV	INTEGER	接收的行数
14	BDTA_SEND	INTEGER	发送 BDTA 的次数
15	BDTA_RECV	INTEGER	接收 BDTA 的次数
16	MAL_ID	BIGINT	邮件标识号
17	MPP_EXEC_ID	INTEGER	MPP 会话句柄上的执行序号，同一个会话上的每个节点上值相同

90. V$SQL_NODE_NAME

显示所有的 sql 节点描述信息，包括 sql 节点类型、名字和详细描述。

序号	列	数据类型	说明
1	TYPE$	INTEGER	节点类型
2	NAME	VARCHAR(24)	节点的名字
3	DESC_CONTENT	VARCHAR(128)	节点的详细描述

91. V$COSTPARA

显示 SQL 计划的代价信息。

序号	列	数据类型	说明
1	RAN_IO_R_COST	DEC(5,3)	随机读代价
2	RAN_IO_W_COST	DEC(5,3)	随机写代价
3	SEQ_IO_R_COST	DEC(5,3)	顺序读代价
4	SEQ_IO_W_COST	DEC(5,3)	顺序写代价
5	SEL_RATE_EQU	DEC(5,3)	等于谓词的选择率
6	SEL_RATE_SINGLE	DEC(5,3)	非等于谓词的选择率
7	INDEX_JOIN_HIT_RATIO DEC	DEC(5,3)	索引连接命中率

92. V$LONG_EXEC_SQLS

当 INI 参数 ENABLE_MONITOR=1、MONITOR_TIME=1 打开时，显示系统最近 1000 条执行时间超过预定值的 SQL 语句。默认预定值为 1000 毫秒。可通过 SP_SET_LONG_TIME 系统函数修改，通过 SF_GET_LONG_TIME 系统函数查看当前值。

序号	列	数据类型	说明
1	SESS_ID	BIGINT	SESSION 的 ID
2	SQL_ID	INTEGER	语句的 SQL ID
3	SQL_TEXT	VARCHAR(1024)	SQL 文本
4	EXEC_TIME	BIGINT	执行时间，单位毫秒
5	FINISH_TIME	TIMESTAMP(0)	执行结束时间
6	N_RUNS	INTEGER	执行次数
7	SEQNO	INTEGER	编号
8	TRX_ID	BIGINT	事务号
9	SESS_SEQ	INTEGER	会话序列号，用来唯一标识会话

93．V$SYSTEM_LONG_EXEC_SQLS

当 INI 参数 ENABLE_MONITOR=1、MONITOR_TIME=1 打开时，显示系统自启动以来执行时间最长的 20 条 SQL 语句，不包括执行时间低于预定值的语句。

序号	列	数据类型	说明
1	SESS_ID	BIGINT	SESSION 的 ID
2	SQL_ID	INTEGER	语句的 SQL ID
3	SQL_TEXT	VARCHAR(1024)	SQL 文本
4	EXEC_TIME	BIGINT	执行时间，单位毫秒
5	FINISH_TIME	DATETIME(0)	执行结束时间
6	N_RUNS	INTEGER	执行次数
7	SEQNO	INTEGER	编号
8	TRX_ID	BIGINT	事务号
9	SESS_SEQ	INTEGER	会话序列号，用来唯一标识会话

94．V$VMS

显示虚拟机信息。

序号	列	数据类型	说明
1	ID	INTEGER	虚拟机 ID
2	TRX_ID	BIGINT	事务 ID
3	STMT_ID	INTEGER	语句 ID
4	EXP_FLAG	CHAR	表达式标识
5	VSTACK_SIZE	INTEGER	栈大小
6	VSTACK	BIGINT	栈基址
7	VTOP	INTEGER	栈顶离基址的距离
8	VUSED	INTEGER	栈已用空间的大小
9	MEMOBJ	BIGINT	内存地址
10	STKFRM_DEPTH	INTEGER	栈深度
11	FREE_STKFRMS	INTEGER	已释放的栈
12	CURR_FRM	BIGINT	当前栈
13	IP	BIGINT	IP 地址
14	RT_HEAP	BIGINT	DMSQL 程序所使用的堆
15	SQL_NO	INTEGER	SQL 语句树的序号
16	RS_SEQ_NO	INTEGER	结果集序号
17	ECPT_CODE	INTEGER	返回的 CODE
18	ERR_DESC	VARCHAR(256)	错误描述
19	ROW_AFFECHED	BIGINT	结果集的行数

20	SQL_TYPE	INTEGER	SQL 语句类型
21	N_FUNS	INTEGER	已使用函数的数目

95. V$STKFRM

显示虚拟机栈桢信息。该参数必须在 INI 参数 ENABLE_MONITOR 和 MONITOR_SQL_EXEC 都开启时才有信息。

序号	列	数据类型	说明
1	VM_ID	INTEGER	虚拟机 ID
2	FRAME_LEVEL	INTEGER	栈的层次
3	ADDR	BIGINT	地址
4	RET_IP	BIGINT	返回的指令地址
5	CURR_METHOD	BIGINT	当前方法
6	LOCAL_SPACE	INTEGER	临时变量空间大小
7	ARG_SPACE	INTEGER	参数空间大小
8	OFF_LOCALS	BIGINT	每个变量基于原栈顶的偏移
9	OFF_ARGS	BIGINT	每个参数基于原栈顶的偏移
10	CURR_BP	BIGINT	新的栈顶
11	IP_BASE	BIGINT	指令基址
12	SQL_VM_NODE	BIGINT	VM 地址
13	SQL_TRX_IDS	BIGINT	事务 ID 地址
14	SYSTEM_FLAG	CHAR	系统标记
15	FRAME_SIZE	INTEGER	栈的大小

96. V$STMTS

显示当前活动会话的最近的语句的相关信息。

序号	列	数据类型	说明
1	STMT_ID	INTEGER	语句 ID，从 iid 页取得
2	TYPE$	INTEGER	语句类型，供系统内部使用
3	EXEC_DIRECT	CHAR	是否立即执行标记
4	SESS_ID	BIGINT	对应的 Session 的 ID
5	SESS_SEQ	INTEGER	会话序列号，用来唯一标识会话
6	PLN	BIGINT	查询计划
7	DML_RCNT	BIGINT	DML 操作所影响的行数
8	REQ_RCNT	BIGINT	prepare 或 execute 查询要求返回的行数
9	BATCH_FLAG	CHAR	是否批量绑定参数
10	N_PARA_ROWS	INTEGER	多行参数时的行数目
11	NTH_PARA_ROW	INTEGER	多行参数时的当前行序号
12	CUR_PARA_OFF	INTEGER	参数的当前偏移
13	PARA_BUF	BIGINT	参数的内存地址
14	SQL_TEXT	VARCHAR(1000)	SQL 语句
15	CURSOR_FORWARD_ONLY	CHAR	是否是只能向前移动的游标
16	CURSOR_NAME	VARCHAR(128)	游标名字
17	CURSOR_KEEP_NAME	CHAR	是否保存游标名字
18	CURSOR_REF_ID	INTEGER	引用的游标语句句柄
19	CURSOR_REF_SEQNO	INTEGER	引用语句的执行序号

97. V$SQL_PLAN_NODE

当 INI 参数 ENABLE_MONITOR 和 MONITOR_SQL_EXEC 都开启时，显示执行计划的节点信息。

序号	列	数据类型	说明
1	PLN_ADDR	VARBINARY(8)	计划地址
2	SQL_ID	INTEGER	语句编号(唯一标识)
3	NODE_ADDR	VARBINARY(8)	节点地址
4	PARENT	VARBINARY(8)	父节点地址
5	LEFT_CHILD	VARBINARY(8)	左孩子地址
6	RIGHT_CHILD	VARBINARY(8)	右孩子地址
7	HAS_INVOKE	CHAR(1)	是否包含函数
8	OPERATION	VARCHAR(30)	操作符名称
9	OPTIONS	VARCHAR(30)	扫描类型
10	TABLE#	INTEGER	表 ID
11	TABLE_NAME	VARCHAR(128)	表名
12	TABLE_ALIAS	VARCHAR(128)	表别名
13	INDEX_NAME	VARCHAR(128)	索引名称
14	SCHID	INTEGER	模式 ID
15	DEPTH	INTEGER	节点层次
16	CARD	BIGINT	行数
17	BYTES	BIGINT	字节数
18	COST	BIGINT	代价
19	CPU_COST	BIGINT	CPU 代价
20	IO_COST	BIGINT	IO 代价
21	SCAN_RANGE	VARCHAR(128)	扫描范围
22	FILTER	VARCHAR(1000)	过滤表达式
23	JOIN_COND	VARCHAR(1000)	连接表达式

98. V$SQL_SUBPLAN

显示子计划信息。

序号	列	数据类型	说明
1	PLN_ADDR	VARBINARY(8)	计划地址
2	SQL_ID	INTEGER	语句编号（唯一标识）
3	SUBPLN	VARBINARY(8)	子计划地址（SUBPLN 在 V$SQL_PLAN_DCTREF 中即为 PLN_ADDR）

99. V$SQL_PLAN_DCTREF

显示所有执行计划相关的详细字典对象信息。

序号	列	数据类型	说明
1	PLN_ADDR	VARBINARY(8)	计划地址
2	SQL_ID	INTEGER	语句编号(唯一标识)
3	TYPE	VARCHAR(16)	字典对象类型（DATABASE、TABLE、VIEW、INDEX、USER、ROLE、PROCEDURE、TRIGGER、CONSTRAINT、SCHEMA、SEQUENCE、LOGIN、DBLINK、SYSROLE、PACKAGE、OBJECT、SYNOM、CRYPTION、CIPHER、IDENTITY、SYS PRIVILEGE、OBJ PRIVILEGE、POLICY、RULE、COLUMN、DOMAIN、CHARSET、COLLATION、CONTEXT INDEX、USER LOGIN、REGEXP REWRITE、NORMAL REWRITE、UNKNOWN）
4	STATUS	VARCHAR(16)	字典对象状态。共四种状态（NORMAL、CREATING、

			ALTERING、DROPPING)，本视图中的执行计划只涉及 DML 语句的 NORMAL 状态
5	R_SIZE	INTEGER	字典对象大小（字节）
6	VERSION	INTEGER	字典对象版本号
7	NAME	VARCHAR(128)	字典对象名称
8	SCHID	INTEGER	模式 ID
9	ID	INTEGER	字典对象 ID
10	UID	INTEGER	用户 ID
11	TS_ID	INTEGER	表空间 ID

100. V$MTAB_USED_HISTORY

显示系统自启动以来使用 MTAB 空间最多的 50 个操作符信息。

序号	列	数据类型	说明
1	EXEC_ID	INTEGER	语句的执行 ID
2	SQL_TEXT	VARCHAR(1024)	SQL 文本
3	STK_LEVEL	INTEGER	当前栈帧的级别
4	CUR_SQL_NODE	BIGINT	当前的 SQL 节点
5	OP_SEQ_NO	INTEGER	操作符在执行计划中序号
6	OP_TYPE$	INTEGER	操作符节点类型
7	MTAB_USED_BY_M	INTEGER	操作符使用的 MTAB 空间，以 M 为单位
8	MEM_USED_BY_M	INTEGER	MTAB 使用的内存空间，以 M 为单位
9	MMT_FILE_ALLOC	INTEGER	操作符使用 MMT 时以 MMT_SIZE 为大小分配的 MMT 文件个数
10	MMT_PAGES_USED	INTEGER	操作符使用 MMT 时实际使用的页数
11	MTAB_TYPE	VARCHAR(10)	MTAB 类型。FLUSH：刷盘时产生；DESTROY：释放 MTAB 时产生；NSORT：排序过程产生
12	MAL_ID	BIGINT	邮件标识号
13	MPP_EXEC_ID	INTEGER	MPP 会话句柄上的执行序号，同一个会话上的每个节点上值相同

101. V$SORT_HISTORY

当 INI 参数 ENABLE_MONITOR=1 都打开时，显示系统自启动以来使用排序页数最多的 50 个操作符信息。

序号	列	数据类型	说明
1	EXEC_ID	INTEGER	语句的执行 ID
2	SQL_TEXT	VARCHAR(1024)	SQL 文本
3	STK_LEVEL	INTEGER	当前栈帧的级别
4	CUR_SQL_NODE	BIGINT	当前的 SQL 节点
5	OP_SEQ_NO	INTEGER	操作符在执行计划中序号
6	OP_TYPE$	INTEGER	操作符节点类型
7	N_PAGES	INTEGER	排序使用的页数
8	N_MERGE_SORT	INTEGER	归并排序的趟数
9	SORT_BYTES	INTEGER	参与排序的每行字节数（估算值）
10	SORT_ROWS	BIGINT	参与排序的行数
11	MAL_ID	BIGINT	邮件标识号
12	MPP_EXEC_ID	INTEGER	MPP 会话句柄上的执行序号，同一个会话上的每个节点上值相同

102. V$HASH_MERGE_USED_HISTORY

HASH MERGE 连接操作符使用的缓存信息。

序号	列	数据类型	说明
1	TYPE$	INTEGER	节点类型
2	SEQNO	INTEGER	序号
3	EXEC_ID	INTEGER	执行 ID
4	SQL_TEXT	VARCHAR(1000)	执行 SQL 文本
5	MERGE_USED	INTEGER	HASH MERGE 使用的缓存空间，单位 MB
6	MAL_ID	BIGINT	邮件标识号

103. V$PLSQL_DDL_HISTORY

记录 DMSQL 程序中执行的 DDL 语句，主要监控 truncate table 和 Execute immediate DDL 语句的情况。

序号	列	数据类型	说明
1	SEQNO	INTEGER	该类语句的执行编号
2	SESS_ID	BIGINT	执行会话 ID
3	SESS_SEQ	INTEGER	会话序列号，用来唯一标识会话
4	TRX_ID	BIGINT	执行事务 ID
5	EXEC_ID	INTEGER	虚拟机的执行序号
6	VM_ID	INTEGER	虚拟机 ID
7	STKFRM_LEVEL	INTEGER	当前栈帧在虚拟机的层次
8	DDL_FROM	VARCHAR(10)	若为 TRUNCATE TABLE，则为'TRUNC'，否则为'EXEC'
9	SQL_TEXT	VARCHAR(1000)	获得该层栈帧执行的 SQL 语句，若字符长度超过 1000，则取前 1000 个字符。
10	DDL_TIME	DATETIME	语句执行时间
11	MAIL_ID	BIGINT	邮件标识号
12	MPP_EXEC_ID	INTEGER	MPP 会话句柄上的执行序号，同一个会话上的每个节点上值相同

104. V$PRE_RETURN_HISTORY

记录大量数据返回结果集的历史信息（查询大量数据产生）。

序号	列	数据类型	说明
1	SEQNO	INTEGER	编号
2	TRX_ID	BIGINT	事务 ID
3	VM_ID	INTEGER	虚拟机 ID
4	EXEC_ID	INTEGER	执行 ID，记录执行的次数
5	SESS_ID	BIGINT	会话 ID
6	SESS_SEQ	INTEGER	会话序列号，用来唯一标识会话
7	SQL_TEXT	VARCHAR(1000)	返回的 SQL 语句
8	N_ROWS	BIGINT	返回的行数
9	IS_FIRST	CHAR	是否第一次返回
10	RETURN_TIME	DATETIME	记录返回的时间
11	MAL_ID	BIGINT	邮件标识号
12	MPP_EXEC_ID	INTEGER	MPP 会话句柄上的执行序号，同一个会话上的每个节点上值相同

105. V$DMSQL_EXEC_TIME

记录动态监控的 sql 语句执行时间。当 ENABLE_MONITOR_DMSQL=1 时才会记录监控的 sql 语句。

序号	列	数据类型	说明
1	EXEC_ID	INTEGER	语句执行号
2	TYPE$	VARCHAR(32)	语句类型:SQL DYNAMIC SQL METHOD
3	SEQ	INTEGER	相对序号:dynamic sql 的指令偏移、sql 语句序号、方法的指令偏移
4	LEVEL	INTEGER	执行层次
5	CALLER	VARCHAR(256)	调用方法名
6	METHOD	VARCHAR(256)	方法名
7	EXEC_CNT	INTEGER	执行次数
8	LAST_EXEC_TICK	BIGINT	最近一次执行时戳 （微秒）
9	TIME_USED	BIGINT	执行时间 （毫秒)
10	MAL_ID	BIGINT	邮件标识号
11	MPP_EXEC_ID	INTEGER	MPP 会话句柄上的执行序号，同一个会话上的每个节点上值相同

106. V$VIRTUAL_MACHINE

显示活动的虚拟机信息。

序号	列	数据类型	说明
1	ID	INTEGER	虚拟机 ID
2	SESSID	BIGINT	回话 ID
3	STMT_ID	INTEGER	语句 ID
4	EXP_FLAG	CHAR	表达式标识
5	VSTACK_SIZE	INTEGER	栈大小
6	VSTACK	BIGINT	栈基址
7	VTOP	INTEGER	栈顶离基址的距离
8	VUSED	INTEGER	栈已用空间的大小
9	STKFRM_DEPTH	INTEGER	栈深度
10	FREE_STKFRMS	INTEGER	已释放的栈
11	CURR_FRM	BIGINT	当前栈
12	IP	BIGINT	IP 地址
13	SQL_NO	INTEGER	SQL 语句树的序号
14	REUSE_CNT	INTEGER	虚拟机重用次数

9) 事务和检查点

107. V$TRX

显示所有活动事务的信息。通过该视图可以查看所有系统中所有的事务以及相关信息，如锁信息等。

序号	列	数据类型	说明
1	ID	BIGINT	当前活动事务的 ID 号
2	NEXTID	BIGINT	下一个事务 ID 号
3	MIN_ACTIVE_ID	BIGINT	所有活动事务 ID 号最小者

4	STATUS	VARCHAR(20)	当前事务的状态
5	ISOLATION	INTEGER	隔离级。0：读未提交；1：读提交；2：可重复读；3：串行化
6	READ_ONLY	CHAR	是否是一个只读事务
7	SESS_ID	BIGINT	当前事务的所在会话
8	SESS_SEQ	INTEGER	会话序列号，用来唯一标识会话
9	INS_CNT	INTEGER	插入回滚记录个数
10	DEL_CNT	INTEGER	删除回滚记录个数
11	UPD_CNT	INTEGER	更新回滚记录个数
12	UPD_INS_CNT	INTEGER	更新插入回滚记录个数
13	UREC_SEQNO	INTEGER	当前 UNDO 记录的递增序列号
14	WAITING	BIGINT	事务等待的锁
15	START_LSN	BIGINT	事务的起始 LSN
16	ROLLBACK_FLAG	INTEGER	是否在回滚
17	XID	VARBINARY(140)	XA 事务唯一标识

108. V$TRXWAIT

显示事务等待信息。

序号	列	数据类型	说明
1	ID	BIGINT	事务 ID
2	WAIT_FOR_ID	BIGINT	所等待的事务 ID
3	WAIT_TIME	INTEGER	当前等待时间

109. V$TRX_VIEW

显示当前事务可见的所有活动事务视图信息。根据达梦多版本规则，通过该视图可以查询系统中自己所见的事务信息；可以通过与 v$trx 表的连接查询它所见事务的具体信息。

序号	列	数据类型	说明
1	SELF_ID	BIGINT	活动事务 ID
2	ACTIVE_ID	BIGINT	所见的事务活动事务 ID

110. V$RECV_ROLLBACK_TRX

显示数据库启动时回滚的所有事务信息。

序号	列	数据类型	说明
1	TRX_ID	BIGINT	事务号
2	BEGIN_LSN	BIGINT	事务启动逻辑日志的 LSN 值，只在 INI 参数开启逻辑日志情况下有效
3	BEGIN_TIME	TIMESTAMP	事务启动时间，只在 INI 参数开启逻辑日志情况下有效
4	N_UPAGES	INTEGER	回滚页总数
5	N_URECS	INTEGER	回滚记录总数

111. V$LOCK

显示活动的事务锁信息。

序号	列	数据类型	说明
1	ADDR	BIGINT	锁地址
2	TRX_ID	BIGINT	所属事务 ID
3	LTYPE	VARCHAR(10)	锁类型：TID 锁、对象锁
4	LMODE	CHAR(2)	锁模式：S 锁、X 锁、IX 锁、IS 锁

5	BLOCKED	INTEGER	是否阻塞
6	TABLE_ID	INTEGER	对应表锁、字典对象 ID
7	ROW_IDX	BIGINT	TID 锁对象事务 ID

112. V$DEADLOCK_HISTORY

记录死锁的历史信息。

序号	列	数据类型	说明
1	SEQNO	INTEGER	编号
2	TRX_ID	BIGINT	事务 ID
3	SESS_ID	BIGINT	会话 ID
4	SESS_SEQ	INTEGER	会话序列号，用来唯一标识会话
5	SQL_TEXT	VARCHAR(1000)	产生死锁的 SQL 语句
6	HAPPEN_TIME	DATETIME	死锁发生的时间

113. V$FLASHBACK_TRX_INFO

显示闪回信息。

序号	列	数据类型	说明
1	START_TRXID	BIGINT	闪回开始事务 ID
2	START_TIMESTAMP	TIMESTAMP	开始时间
3	COMMIT_TRXID	BIGINT	闪回完成事务 ID
4	COMMIT_TIMESTAMP	TIMESTAMP	完成时间
5	LOGIN_USER	VARCHAR(256)	闪回用户名
6	UNDO_CHANGE#	INTEGER	回滚记录序号
7	OPERATION	CHAR(1)	DML 操作类型。 D：删除；U：修改；I：插入；N：更新插入（专门针对 CLUSTER PRIMARY KEY 的插入）； C：事务提交；P：预提交记录；O：default
8	TABLE_NAME	VARCHAR(256)	表名
9	TABLE_OWNER	VARCHAR(256)	表所属用户名
10	ROW_ID	BIGINT	ROWID
11	UNDO_SQL	VARCHAR(3900)	回滚 SQL 语句

114. V$CKPT_HISTORY

显示检查点历史信息。

序号	列	数据类型	说明
1	SEQNO	INTEGER	序列号
2	CKPT_CMD	VARCHAR(24)	检查点
3	THREAD_ID	BIGINT	线程 ID
4	START_TIME	DATETIME	开始时间
5	TIME_USED	BIGINT	从开始到结束使用时间
6	PAGE_FLUSHED	INTEGER	刷盘数
7	FIL	INTEGER	文件号
8	OFF	BIGINT	文件偏移量
9	CKPT_LSN	BIGINT	检查点 LSN

115. V$CKPT

显示系统检查点信息。

序号	列	数据类型	说明
1	CKPT_RLOG_SIZE	INTEGER	检查点日志大小
2	CKPT_DIRTY_PAGES	INTEGER	脏页的数量。产生多少脏页后，才强制产生检查点
3	CKPT_INTERVAL	INTEGER	系统强制产生检查点的时间间隔(s)
4	CKPT_FLUSH_RATE	DEC(5,2)	每次检查点脏页的刷盘比例
5	CKPT_FLUSH_PAGES	INTEGER	每次检查点至少刷盘的脏页数
6	LAST_BEGIN_TIME	TIMESTAMP	最近一次执行的开始时间
7	LAST_END_TIME	TIMESTAMP	最近一次执行的结束时间
8	CKPT_LSN	BIGINT	最近一次检查点 LSN
9	CKPT_FILE	BIGINT	最近一次检查点对应的当时的文件号
10	CKPT_OFFSET	BIGINT	最近一次检查点对应的当时的文件偏移
11	STATE	VARCHAR(128)	检查点状态，只有2种。0 NONE，其他 PROCESSING
12	CKPT_TOTAL_COUNT	BIGINT	检查点已做的个数
13	CKPT_RESERVE_COUNT	BIGINT	预先申请日志空间的次数
14	CKPT_FLUSHED_PAGES	BIGINT	检查点已刷页的个数
15	CKPT_TIME_USED	BIGINT	检查点从开始到结束经历的时间

10) 事件

116. V$WAIT_HISTORY

通过该视图可以查询等待事件的具体信息，如等待的线程 id，会话 id 等。可以查看具体等待事件的信息，如果某个事务等待时间过长，则可以查询到具体事务信息以及所在的线程和所牵涉的对象，分析原因进行优化等操作。

序号	列	数据类型	说明
1	THREAD_ID	BIGINT	线程 ID
2	TRX_ID	BIGINT	事务 ID
3	WAIT_CLASS	INTEGER	等待类型号
4	WAIT_OBJECT	BIGINT	等待对象
5	WAIT_START	DATETIME	等待事件的开始时间
6	WAIT_TIME	BIGINT	等待时间（单位：usec）
7	SPACE_ID	INTEGER	表空间号
8	FILE_ID	INTEGER	文件号
9	PAGE_NO	INTEGER	页号

117. V$EVENT_NAME

显示当前系统所支持的等待事件的类型汇总信息。

序号	列	数据类型	说明
1	EVENT#	SMALLINT	事件编号
2	EVENT	VARCHAR(32)	事件名称。dbfile read：读用户数据文件；dbfile multi read：批量读用户数据文件；dbfile read wait：写用户数据文件；dbfile remote read：rac 引起的读磁盘；instance recovery read：recorvery 引起的读磁盘；dbfile remote write：rac 引起的写磁盘；dbfile discard write：buf 不够引起的写磁盘；dbfile ckpt

			write：ckpt 引起的写磁盘；dbfile extend：扩库文件引起的写磁盘；shared memory pool busy：共享内存并发使用冲突；buffer busy wait：buffer 并发使用冲突；dict cache busy：字典 cache 并发使用冲突；plan cache busy：计划 cache 并发使用冲突；redo log system busy：redo log 系统并发冲突；redo log buffer busy：redo log 缓冲区并发冲突；kernel busy：kernel 并发冲突；table lock busy：表锁系统并发冲突；tid lock busy：行锁系统并发冲突；parallel bdta pool busy：并行 bdta pool 并发冲突；iid system busy：iid 系统并发冲突；session system busy：session 系统并发冲突；public vpool busy：公共 vpool 并发冲突；pseg queue busy：pseg 队列并发冲突；pseg stack busy：pseg 堆栈并发冲突；page busy wait：数据页并发冲突； table lock wait：事务间表锁等待发生；trxid lock wait：事务间行锁等待发生；dead lock：事务间产生死锁；transaction system busy：事务系统并发冲突；transaction view busy：事务可见性视图并发冲突；purge system busy：purge 系统并发冲突；file system busy：文件系统并发冲突；asm system busy：ASM 系统并发冲突； out of share mem pool：共享内存池不足；out of share coldata pool：共享 coldata 池不足：network send wait：网络发送等待；network recv wait：网络接收等待
3	WAIT_CLASS#	INTEGER	事件类别编号：0、1、2、3、4、5、6。分别和 WAIT_CLASS 中的等待事件类别 User I/O、System I/O 、Concurrency、Commit、Transaction、Memory、Network 一一对应
4	WAIT_CLASS	VARCHAR(32)	事件类别名称：User I/O、System I/O 、Concurrency、Commit、Transaction、Memory、Network

118. V$SYSTEM_EVENT

显示自系统启动以来所有等待事件的详细信息。

序号	列	数据类型	说明
1	EVENT#	INTEGER	事件编号
2	EVENT	VARCHAR(32)	事件名称。
3	TOTAL_WAITS	INTEGER	等待次数
4	TIME_WAITED	INTEGER	等待时间（单位：1/100 秒）
5	TIME_WAITED_MICRO	BIGINT	等待时间（单位：微秒）
6	AVERAGE_WAIT_MICRO	INTEGER	平均等待时间（单位：微秒）
7	SMAX_TIME	INTEGER	最长等待时间（单位：微秒）

8	SMIN_TIME	INTEGER	最短等待时间（单位：微秒）
9	WAIT_CLASS#	INTEGER	等待事件类别编号
10	WAIT_CLASS	VARCHAR(32)	等待事件类别名称

119. V$SESSION_EVENT

显示当前会话等待事件的所有信息。

序号	列	数据类型	说明
1	SESSADDR	VARBINARY(8)	会话地址
2	SESSION#	INTEGER	会话 ID
3	EVENT#	INTEGER	事件编号
4	EVENT	VARCHAR(32)	事件名称
5	TOTAL_WAITS	INTEGER	等待次数
6	TIME_WAITED	INTEGER	等待时间（单位：1/100 秒）
7	TIME_WAITED_MICRO	BIGINT	等待时间（单位：微秒）
8	AVERAGE_WAIT_MICRO	INTEGER	平均等待时间（单位：微秒）
9	SMAX_TIME	INTEGER	最长等待时间（单位：微秒）
10	SMIN_TIME	INTEGER	最短等待时间（单位：微秒）
11	WAIT_CLASS#	INTEGER	等待事件类别编号
12	WAIT_CLASS	VARCHAR(32)	等待事件类别名称

120. V$SESSION_WAIT_HISTORY

显示会话等待事件的历史信息。

序号	列	数据类型	说明
1	SESSADDR	VARBINARY(8)	会话地址
2	SESSION#	INTEGER	会话 ID
3	SQL_ID	INTEGER	语句的 SQL ID
4	EVENT#	INTEGER	事件编号
5	EVENT	VARCHAR(32)	事件名称
6	P1TEXT	VARCHAR(32)	等待事件对应的参数 1 说明(如果没有足够参数，则为 NULL，以下参数一样)
7	P1	VARCHAR(256)	等待事件对应的参数 1 值
8	P2TEXT	VARCHAR(32)	等待事件对应的参数 2 说明
9	P2	INTEGER	等待事件对应的参数 2 值
10	P3TEXT	VARCHAR(32)	等待事件对应的参数 3 说明
11	P3	INTEGER	等待事件对应的参数 3 值
12	P4TEXT	VARCHAR(32)	等待事件对应的参数 4 说明
13	P4	INTEGER	等待事件对应的参数 4 值
14	TIME_WAITED	INTEGER	等待时间（单位：1/100 秒）
15	TIME_WAITED_MICRO	INTEGER	等待时间（单位：微秒）
16	WAIT_CLASS#	INTEGER	等待事件类别编号
17	WAIT_CLASS	VARCHAR(32)	等待事件类别名称

121. V$DANGER_EVENT

数据库重要事件和行为信息视图。

序号	列	数据类型	说明
1	OPTIME	DATETIME(6)	重要事件或行为发生的时间
2	OPERATION	VARCHAR(1024)	重要事件或行为执行的语句

3	OPUSER	VARCHAR(128)	执行重要事件或行为的用户

122. V$TASK_QUEUE

任务队列信息

序号	列	数据类型	说明
1	WAITING	INTEGER	等待处理任务数
2	READY	BIGINT	已处理任务数
3	TOTAL_WAIT	BIGINT	已处理任务的总等待时间
4	AVERAGE_WAIT	INTEGER	已处理任务的平均等待时间

123. V$TRACE_QUEUE

事件跟踪任务队列信息

序号	列	数据类型	说明
1	WAITING	INTEGER	等待处理任务数
2	READY	BIGINT	已处理任务数
3	TOTAL_WAIT	BIGINT	已处理任务的总等待时间
4	AVERAGE_WAIT	INTEGER	已处理任务的平均等待时间

11) 进程和线程

124. V$PROCESS

显示当前进程信息。

序号	列	数据类型	说明
1	PID	INTEGER	进程 ID
2	PNAME	VARCHAR(256)	进程名
3	TRACE_NAME	VARCHAR(256)	SQL 日志路径，若 INI 参数 SVR_LOG 为 0，则值为空串
4	TYPE$	TINYINT	类型

125. V$THREADS

显示系统中所有活动线程的信息。

序号	列	数据类型	说明
1	ID	BIGINT	线程 ID
2	NAME	VARCHAR(128)	线程名
3	START_TIME	DATETIME	线程开始时间
4	THREAD_DESC	VARCHAR(1024)	线程描述

126. V$LATCHES

显示正在等待的线程信息。

序号	列	数据类型	说明
1	OBJECT	BIGINT	等待的对象
2	REQUEST_TYPE	CHAR	等待的锁类型：S 锁，X 锁。
3	THREAD_ID	BIGINT	等待线程 ID
4	N_READERS	INTEGER	读线程个数
5	WRITER	BIGINT	写线程 ID
6	N_WRITERS	INTEGER	写线程拥有该锁的次数
7	WRITE_WAITING	CHAR	是否有写线程在等待，如果有，则不让读线程进入
8	N_READERS_WAIT	INTEGER	读等待个数

9	N_WRITERS_WAIT	INTEGER	写等待个数
10	N_IO_WAIT	INTEGER	IO 等待个数
11	SPACE_ID	INTEGER	页面缓冲控制信息的表空间 ID
12	FILE_ID	INTEGER	页面缓冲控制信息的文件 ID
13	PAGE_NO	INTEGER	数据在文件中的页号

127. V$WTHRD_HISTORY

通过本视图可以观察系统从启动以来,所有活动过线程的相关历史信息。其中 CHG_TYPE 有 REUSE_OK（本 SESSION 重用成功)、REUSE_FAIL (重用失败)、TO_IDLE(不重用，直接变 IDLE)等几种类型。

序号	列	数据类型	说明
1	SEQNO	INTEGER	序号
2	THREAD_ID	BIGINT	线程号
3	CHG_TYPE	VARCHAR(24)	线程类型
4	CHG_TIME	DATETIME	更改时间

12) 系统信息

128. V$SYSTEMINFO

系统信息视图。

序号	列	数据类型	说明
1	N_CPU	INTEGER	CPU 个数
2	TOTAL_PHY_SIZE	BIGINT	物理内存总大小
3	FREE_PHY_SIZE	BIGINT	剩余物理内存大小
4	TOTAL_VIR_SIZE	BIGINT	虚拟内存总大小
5	FREE_VIR_SIZE	BIGINT	剩余虚拟内存大小
6	TOTAL_DISK_SIZE	BIGINT	磁盘总大小
7	FREE_DISK_SIZE	BIGINT	剩余磁盘大小
8	DRIVER_NAME	VARCHAR(5)	驱动器名称
9	DRIVER_TOTAL_SIZE	BIGINT	驱动器总空间大小
10	DRIVER_FREE_SIZE	BIGINT	驱动器剩余空间大小
11	LOAD_ONE_AVERAGE	FLOAT	每分钟平均负载
12	LOAD_FIVE_AVERAGE	FLOAT	每五分钟平均负载
13	LOAD_FIFTEEN_AVERAGE	FLOAT	每十五分钟平均负载
14	CPU_USER_RATE	FLOAT	用户级的 CPU 使用率
15	CPU_SYSTEM_RATE	FLOAT	用户级的 CPU 使用率
16	CPU_IDLE_RATE	FLOAT	用户级的 CPU 使用率
17	SEND_BYTES_TOTAL	BIGINT	发送的总字节数
18	RECEIVE_BYTES_TOTAL	BIGINT	接收的总字节数
19	SEND_BYTES_PER_SECOND	BIGINT	当前每秒发送字节数
20	RECEIVE_BYTES_PER_SECOND	BIGINT	当前每秒接收字节数
21	SEND_PACKAGES_PER_SECOND	BIGINT	当前每秒发送数据包数
22	RECEIVE_PACKAGES_PER_SECOND	BIGINT	当前每秒接收数据包数

129. V$CMD_HISTORY

通过本视图可以观察系统的一些命令的历史信息。其中 cmd 指的是 SESS_ALLOC，

SESS_FREE,CKPT,TIMER_TRIG,SERERR_TRIG,LOG_REP,MAL_LETTER, CMD_LOGIN 等。

序号	列	数据类型	说明
1	CMD	VARCHAR(24)	命令
2	THREAD_ID	BIGINT	线程 ID
3	SESS_ID	BIGINT	SESSION 的 ID
4	SESS_SEQ	INTEGER	会话序列号，用来唯一标识会话
5	TRX_ID	BIGINT	事务 ID
6	STMT_ID	INTEGER	语句 ID
7	START_TIME	DATETIME	命令开始时间
8	TIME_USED	BIGINT	命令从开始执行到执行结束花费的时间

130. V$RUNTIME_ERR_HISTORY

监控运行时错误历史。异常分为三种：一种是系统异常，用户没有捕获，由 vm_raise_runtime_error 产生；第二种是用户异常，用户捕获错误，并抛出自定义异常，由 nthrow_exec 产生；第三种是语法异常，语法未通过，由 nsvr_build_npar_cop_out 产生。

V$RUNTIME_ERR_HISTORY 视图中各个列的含义如下：

序号	列	数据类型	说明
1	SEQNO	INTEGER	该类语句的执行编号
2	SESS_ID	BIGINT	执行会话 ID
3	SESS_SEQ	INTEGER	会话序列号，用来唯一标识会话
4	TRX_ID	BIGINT	执行事务 ID
5	EXEC_ID	INTEGER	虚拟机的执行序号
6	VM_ID	INTEGER	虚拟机 ID
7	STKFRM_LEVEL	INTEGER	当前栈帧在虚拟机的层次
8	SQL_TEXT	VARCHAR(1000)	获得该层栈帧执行的 SQL 语句，若字符长度超过 1000，则取前 1000 个字符
9	SU_FLAG	CHAR	U =用户异常，S =系统异常，P =语法异常
10	ECPT_CODE	INTEGER	异常错误号
11	ECPT_DESC	VARCHAR(256)	异常描述
12	MAL_ID	BIGINT	邮件标识号
13	ERR_TIME	DATETIME	错误产生的时间
14	MPP_EXEC_ID	INTEGER	MPP 会话句柄上的执行序号，同一个会话上的每个节点上值相同

13) MAL 系统

131. V$MAL_SYS

MAL 系统信息视图。如果是数据守护环境，则只显示主库的 MAL 系统信息。

序号	列	数据类型	说明
1	SYS_STATUS	INTEGER	MAL 系统状态：0:OPEN,1:PRE_SHUTDOWN,2:SHUTDOWN
2	STMT_ID	INTEGER	MAL 系统当前 stmtid
3	NEXT_MAL_ID	BIGINT	下一个 MAL_ID

4	MAL_PORT	INTEGER	MAL 监听端口
5	N_SITE	INTEGER	MAL 配置的站点数目
6	MAL_NUM	INTEGER	MAL 系统邮箱数目
7	MAL_SEQ_NO	INTEGER	站点本身的 MAL 序号
8	EMPTY_LET_NUM	INTEGER	空邮件数目
9	MAL_CHECK_INTERVAL	INTEGER	链路检测间隔
10	MAL_CONN_FAIL_INTERVAL	INTEGER	认定链路断开的时间间隔
11	MAL_COMPRESS_LEVEL	INTEGER	邮件压缩级别
12	MAL_BUF_SIZE	INTEGER	单个 MAL 缓存大小限制，以 M 为单位。当 MAL 的缓存邮件超过此大小，会将邮件存储到文件中。有效值范围（0~500000），默认为 100，配置为 0 表示无限制
13	MAL_SYS_BUF_SIZE	INTEGER	MAL 系统总内存大小限制，以 M 为单位。有效值范围（0~500000），默认为 0，表示无限制
14	MAL_VPOOL_SIZE	INTEGER	MAL 配置的总的 POOL 大小，以 M 为单位。有效值范围（1~500000），默认为 128
15	MAL_TEMP_PATH	VARCHAR(256)	指定临时文件的目录。当邮件使用的内存超过 MAL_BUF_SIZE 或者 MAL_SYS_BUF_SIZE 时，将新产生的邮件保存到临时文件中。如果缺省，则新产生的邮件保存到 temp.dbf 文件中

132. V$MAL_INFO

MAL 邮箱信息视图。

序号	列	数据类型	说明
1	MAL_ID	BIGINT	MAL 标示号
2	ORG_SITE_NO	INTEGER	创建 MAL 的原始站点号
3	DEST_SITE_NO	INTEGER	目标站点号
4	MAX_BUF_SIZE	BIGINT	保存邮件的最大缓存大小
5	USED_BUF_SIZE	BIGINT	已经使用的缓存大小
6	RECEIVE_NUM	INTEGER	收到的邮件数
7	DISCARD_NUM	INTEGER	废弃的邮件数
8	LETTER_NUM	INTEGER	普通邮件数目
9	PLN_LET_NUM	INTEGER	执行计划中的邮件总数目
10	WAIT_FLAG	INTEGER	等待状态。1：等待 0：非等待
11	MCPR_FLAG	INTEGER	MAL 任务处理情况。0：RESUME； 1：MPLN；2：CACEL；4：PAUSE
12	SESS_ID	BIGINT	MAL 对应的会话 ID，和 V$SESSIONS 中的 SESS_ID 对应
13	DONE_NUM	INTEGER	处理的邮件数
14	TOTAL_TIME	BIGINT	邮件处理的总用时

133. V$MAL_LETTER_INFO

MAL 上的信件信息视图。

序号	列	数据类型	说明
1	MAL_ID	BIGINT	MAL 标示号
2	IS_IN_MEM	INTEGER	是否在缓存中
3	FILE_ID	INTEGER	所在临时表空间的文件号

4	PAGE_NO	INTEGER	所在临时表空间的页号
5	STMT_ID	INTEGER	句柄 ID
6	PLN_OP_ID	INTEGER	通讯操作符的计划序号
7	ORG_SITE	INTEGER	原始站点号
8	MPP_EXEC_ID	INTEGER	操作符的执行序号
9	BUILD_TIME	BIGINT	邮件发送序号
10	P_SRC_SITE	INTEGER	邮件的物理站点号
11	L_SRC_SITE	INTEGER	邮件的逻辑站点号
12	MSG_LEN	INTEGER	信件的消息长度

134. V$MAL_USING_LETTERS

服务器中正在使用或者使用过但是没有释放的邮件信息，用于检查 MAL 系统潜在的内存泄露。

序号	列	数据类型	说明
1	MAL_ID	BIGINT	MAL 标示号
2	STMT_ID	INTEGER	句柄 ID
3	PLN_OP_ID	INTEGER	通讯操作符的计划序号
4	ORG_SITE	INTEGER	原始站点号
5	SRC_SITE	INTEGER	源站点号
6	DEST_SITE	INTEGER	目标站点号
7	MPP_EXEC_ID	INTEGER	操作符的执行序号
8	BUILD_TIME	BIGINT	邮件发送序号
9	P_SRC_SITE	INTEGER	邮件的物理站点号
10	L_SRC_SITE	INTEGER	邮件的逻辑站点号
11	CMD_TYPE	VARCHAR(128)	邮件的消息类型
12	MSG_LEN	INTEGER	信件的消息长度

14) 通讯

135. V$DBLINK

动态使用到的数据库链接信息视图。

序号	列	数据类型	说明
1	LINK_CONN	BIGINT	DBLINK 连接句柄
2	LINK_ID	INTEGER	DBLINK 的 ID
3	LINK_NAME	VARCHAR(128)	DBLINK 的名称
4	SCH_ID	INTEGER	DBLINK 模式 ID
5	OWNER_ID	INTEGER	DBLINK 用户 ID
6	IS_PUBLIC	VARCHAR(3)	是否为 PUBLIC
7	LOGIN_NAME	VARCHAR(128)	登录名
8	HOST_NAME	VARCHAR(128)	主库名
9	PORT_NUM	INTEGER	端口号
10	LOGGED_ON	VARCHAR(3)	DBLINK 当前是否已链接
11	HETEROGENEOUS	VARCHAR(3)	DBLINK 同步链接为 YES，异步链接为 NO
12	PROTOCOL	VARCHAR(6)	DBLINK 通信协议
13	IN_USE	VARCHAR(3)	当前 DBLINK 句柄是否正在被使用

15) MPP

136. V$MPP_CFG_SYS

MPP 系统配置信息视图。

序号	列	数据类型	说明
1	SYS_STATE	VARCHAR(128)	系统状态
2	N_SITE	INTEGER	总站点数
3	N_ERR_SITE	INTEGER	故障站点数
4	SELF_EP_SEQNO	INTEGER	当前站点序号

137. V$MPP_CFG_ITEM

MPP站点配置信息视图。

序号	列	数据类型	说明
1	SERVICE_NAME	VARCHAR(128)	服务名
2	INST_NAME	VARCHAR(128)	实例名
3	EP_SEQNO	INTEGER	站点序号
4	STATE	VARCHAR (128)	站点状态

138. V$MAL_SITE_INFO

MAL站点信息视图，MPP模式下，自动收集MPP各个站点的信息。

序号	列	数据类型	说明
1	SRC_SITE_SEQ	INTEGER	发送邮件的源站点
2	DEST_SITE_SEQ	INTEGER	目标站点序号
3	MAL_PORT_NUM	INTEGER	目标站点到本站点的 MAL 链路数
4	BUILD_TIME	BIGINT	下一个邮件发送序号
5	LBTAPFDS	BIGINT	收到的邮件中已处理的最后一个邮件的序号
6	CUR_LETTER_NUM	INTEGER	当前保存不连续邮件的个数
7	MAX_LETTER_NUM	INTEGER	目前为止不连续邮件的最大个数
8	TOTAL_LINK_NUM	INTEGER	已创建到目标站点的 MAL_LINK 数
9	FREE_LINK_NUM	INTEGER	当前空闲的 MAL_LINK 数
10	SEND_LETTER_NUM	INTEGER	当前发送的邮件数

16) RAC

139. V$RAC_EP_INFO

显示实例信息。

序号	列	数据类型	说明
1	EP_NAME	VARCHAR (128)	实例名称
2	EP_SEQNO	INTEGER	RAC 序号
3	EP_GUID	BIGINT	EP 唯一标识码
4	EP_TIMESTAMP	BIGINT	EP 时间戳
5	EP_MODE	VARCHAR(32)	EP 模式
6	EP_STATUS	VARCHAR(32)	EP 状态

140. V$RAC_GBS_POOL

显示 GBS 控制结构的信息。

序号	列	数据类型	说明
1	N_CTL	INTEGER	GBS 控制块总数
2	N_FREE_CTL	INTEGER	空闲的 GBS 控制块数目
3	N_SUB_POOL	INTEGER	GBS_POOL 个数

141. V$RAC_GBS_POOLS_DETAIL

显示分片的 GBS_POOL 详细信息。

序号	列	数据类型	说明
1	POOL_ID	INTEGER	GBS_POOL 编号
2	N_USED_CTL	INTEGER	正在使用的 GBS 控制块数目
3	N_REQUEST	INTEGER	正在等待 GBS 控制块的请求数目
4	N_FREE_REQUEST	INTEGER	空闲的 GBS 请求控制块数目

142. V$RAC_GBS_CTL

显示 GBS 控制块信息。多个 pool，依次扫描。

序号	列	数据类型	说明
1	POOL_ID	INTEGER	GBS_POOL 编号
2	TS_ID	INTEGER	表空间号
3	FILE_ID	INTEGER	文件号
4	PAGE_NO	INTEGER	页号
5	ACCESS_MAP	INTEGER	曾经访问此数据页的 EP
6	FRESH_EP	INTEGER	最新数据所在 EP
7	FRESH_LSN	BIGINT	最新修改对应的 LSN 值
8	N_REPLACED	INTEGER	控制块被替换次数
9	N_REVOKED	INTEGER	权限被回收次数
10	N_OWNER	INTEGER	拥有权限的 EP 数
11	N_REQUEST	INTEGER	请求授权的 EP 数
12	N_REVOKING	INTEGER	正在回收权限的 EP 数
13	N_REVOKE_X_ONLY	INTEGER	页的优化次数

143. V$RAC_GBS_CTL_DETAIL

显示 GBS 控制块详细信息。多个 pool，依次扫描。

序号	列	数据类型	说明
1	POOL_ID	INTEGER	GBS_POOL 编号
2	TS_ID	INTEGER	表空间号
3	FILE_ID	INTEGER	文件号
4	PAGE_NO	INTEGER	页号
5	ACCESS_MAP	INTEGER	曾经访问此数据页的 EP
6	FRESH_EP	INTEGER	最新数据所在 EP
7	FRESH_LSN	BIGINT	最新修改对应的 LSN 值
8	N_REPLACED	INTEGER	控制块被替换次数
9	N_REVOKED	INTEGER	权限被回收次数
10	N_OWNER	INTEGER	拥有权限的 EP 数
11	N_REQUEST	INTEGER	请求授权的 EP 数
12	N_REVOKING	INTEGER	正在回收权限的 EP 数
13	TYPE	VARCHAR(32)	详细信息类型（OWNER/REQUEST/REVOKING）

14	MODE	INTEGER	封锁模式，0/1/2/4： N_LATCH/X_LATCH/S_LATCH/F_LATCH
15	EP_SEQNO	INTEGER	拥有、请求、或者回收封锁的 EP
16	REAL_FLUSH	CHAR	是否真正执行刷盘请求（'Y'/'N'）
17	N_REVOKE_X_ONLY	INTEGER	页的优化次数

144. V$RAC_GBS_CTL_LRU_FIRST

显示 GBS 控制块 LRU 链表首页信息。多个 pool，依次扫描。

序号	列	数据类型	说明
1	POOL_ID	INT	GBS_POOL 编号
2	TS_ID	INT	表空间号
3	FILE_ID	INT	文件号
4	PAGE_NO	INTEGER	页号
5	ACCESS_MAP	INT	曾经访问此数据页的 EP
6	FRESH_EP	INTEGER	最新数据所在 EP
7	FRESH_LSN	BIGINT	最新修改对应的 LSN 值
8	N_REPLACED	INTEGER	控制块被替换次数
9	N_REVOKED	INTEGER	权限被回收次数
10	N_OWNER	INTEGER	拥有权限的 EP 数
11	N_REQUEST	INTEGER	请求授权的 EP 数
12	N_REVOKING	INTEGER	正在回收权限的 EP 数
13	N_REVOKE_X_ONLY	INTEGER	页的优化次数

145. V$RAC_GBS_CTL_LRU_FIRST_DETAIL

显示 GBS 控制块 LRU 链表首页详细信息。多个 pool，依次扫描。

序号	列	数据类型	说明
1	POOL_ID	INTEGER	GBS_POOL 编号
2	TS_ID	INTEGER	表空间号
3	FILE_ID	INTEGER	文件号
4	PAGE_NO	INTEGER	页号
5	ACCESS_MAP	INTEGER	曾经访问此数据页的 EP
6	FRESH_EP	INTEGER	最新数据所在 EP
7	FRESH_LSN	BIGINT	最新修改对应的 LSN 值
8	N_REPLACED	INTEGER	控制块被替换次数
9	N_REVOKED	INTEGER	权限被回收次数
10	N_OWNER	INTEGER	拥有权限的 EP 数
11	N_REQUEST	INTEGER	请求授权的 EP 数
12	N_REVOKING	INTEGER	正在回收权限的 EP 数
13	TYPE	VARCHAR(32)	详细信息类型（OWNER/REQUEST/REVOKING）
14	MODE	INTEGER	封锁模式，0/1/2/4： N_LATCH/X_LATCH/S_LATCH/F_LATCH
15	EP_SEQNO	INTEGER	拥有、请求、或者回收封锁的 EP
16	REAL_FLUSH	CHAR	是否真正执行刷盘请求（'Y'/'N'）
17	N_REVOKE_X_ONLY	INTEGER	页的优化次数

146. V$RAC_GBS_CTL_LRU_LAST

显示 GBS 控制块 LRU 链表尾页信息。多个 pool，依次扫描。

序号	列	数据类型	说明

1	POOL_ID	INTEGER	GBS_POOL 编号
2	TS_ID	INTEGER	表空间号
3	FILE_ID	INTEGER	文件号
4	PAGE_NO	INTEGER	页号
5	ACCESS_MAP	INTEGER	曾经访问此数据页的 EP
6	FRESH_EP	INTEGER	最新数据所在 EP
7	FRESH_LSN	BIGINT	最新修改对应的 LSN 值
8	N_REPLACED	INTEGER	控制块被替换次数
9	N_REVOKED	INTEGER	权限被回收次数
10	N_OWNER	INTEGER	拥有权限的 EP 数
11	N_REQUEST	INTEGER	请求授权的 EP 数
12	N_REVOKING	INTEGER	正在回收权限的 EP 数
13	N_REVOKE_X_ONLY	INTEGER	页的优化次数

147. V$RAC_GBS_CTL_LRU_LAST_DETAIL

显示 GBS 控制块 LRU 链表尾页详细信息。多个 POOL，依次扫描。

序号	列	数据类型	说明
1	POOL_ID	INTEGER	GBS_POOL 编号
2	TS_ID	INTEGER	表空间号
3	FILE_ID	INTEGER	文件号
4	PAGE_NO	INTEGER	页号
5	ACCESS_MAP	INTEGER	曾经访问此数据页的 EP
6	FRESH_EP	INTEGER	最新数据所在 EP
7	FRESH_LSN	BIGINT	最新修改对应的 LSN 值
8	N_REPLACED	INTEGER	控制块被替换次数
9	N_REVOKED	INTEGER	权限被回收次数
10	N_OWNER	INTEGER	拥有权限的 EP 数
11	N_REQUEST	INTEGER	请求授权的 EP 数
12	N_REVOKING	INTEGER	正在回收权限的 EP 数
13	TYPE	VARCHAR(32)	详细信息类型（OWNER/REQUEST/REVOKING）
14	MODE	INTEGER	封锁模式，0/1/2/4： N_LATCH/X_LATCH/S_LATCH/F_LATCH
15	EP_SEQNO	INTEGER	拥有、请求、或者回收封锁的 EP
16	REAL_FLUSH	CHAR	是否真正执行刷盘请求（‘Y’/’N’）
17	N_REVOKE_X_ONLY	INTEGER	页的优化次数

148. V$RAC_GBS_REQUEST_CTL

显示等待 GBS 控制块的请求信息。多个 POOL，依次扫描。

序号	列	数据类型	说明
1	POOL_ID	INTEGER	GBS_POOL 编号
2	TS_ID	INTEGER	表空间号
3	FILE_ID	INTEGER	文件号
4	PAGE_NO	INTEGER	页号
5	MODE	INTEGER	封锁模式，0/1/2/4： N_LATCH/X_LATCH/S_LATCH/F_LATCH
6	EP_SEQNO	INTEGER	请求封锁的 EP

149. V$RAC_LBS_POOL

显示 LBS 控制结构的信息。

序号	列	数据类型	说明
1	N_CTL	INTEGER	LBS 控制块总数
2	N_FREE_CTL	INTEGER	空闲的 LBS 控制块数目
3	N_SUB_POOL	INTEGER	LBS_POOL 个数

150. V$RAC_LBS_POOLS_DETAIL

显示分片的 LBS_POOL 详细信息。多个 POOL，依次扫描。

序号	列	数据类型	说明
1	POOL_ID	INTEGER	LBS_POOL 编号
2	N_USED_CTL	INTEGER	正在使用的 LBS 控制块数目
3	N_REQUEST	INTEGER	正在等待 LBS 控制块的请求数目
4	N_FREE_REQUEST	INTEGER	空闲的 LBS 请求控制块数目

151. V$RAC_NO_VIO_PAGE

显示不需要 VIO 的数据页信息。

序号	列	数据类型	说明
1	TS_ID	INTEGER	表空间号
2	FILE_ID	INTEGER	文件号
3	PAGE_NO	INTEGER	页号

152. V$RAC_LBS_CTL

显示 LBS 控制块信息。多个 POOL，依次扫描。

序号	列	数据类型	说明
1	POOL_ID	INTEGER	LBS_POOL 编号
2	TS_ID	INTEGER	表空间号
3	FILE_ID	INTEGER	文件号
4	PAGE_NO	INTEGER	页号
5	ACCESS_MAP	INTEGER	曾经访问此数据页的 EP
6	FRESH_EP	INTEGER	最新数据所在 EP
7	FRESH_LSN	BIGINT	最新修改对应的 LSN 值
8	N_REPLACED	INTEGER	控制块被替换次数
9	N_REVOKED	INTEGER	权限被回收次数
10	N_FIXED	INTEGER	引用记数
11	MODE	INTEGER	获得 GBS 授权的封锁模式
12	PHY_LSN	BIGINT	数据页上的最新 LSN 值
13	N_REQUEST	INTEGER	请求获得授权的工作线程数
14	N_REVOKE_X_ONLY	INTEGER	页的优化次数

153. V$RAC_LBS_CTL_LRU_FIRST

显示 LBS 的 LRU_FIRST 控制块信息。多个 POOL，依次扫描。

序号	列	数据类型	说明
1	POOL_ID	INTEGER	LBS_POOL 编号
2	TS_ID	INTEGER	表空间号
3	FILE_ID	INTEGER	文件号
4	PAGE_NO	INTEGER	页号
5	ACCESS_MAP	INTEGER	曾经访问此数据页的 EP
6	FRESH_EP	INTEGER	最新数据所在 EP

7	FRESH_LSN	BIGINT	最新修改对应的 LSN 值
8	N_REPLACED	INTEGER	控制块被替换次数
9	N_REVOKED	INTEGER	权限被回收次数
10	N_FIXED	INTEGER	引用记数
11	MODE	INTEGER	获得 GBS 授权的封锁模式
12	PHY_LSN	BIGINT	数据页上的最新 LSN 值
13	N_REQUEST	INTEGER	请求获得授权的工作线程数
14	N_REVOKE_X_ONLY	INTEGER	页的优化次数

154. V$RAC_LBS_CTL_LRU_LAST

显示 LBS 的 LRU_LAST 控制块信息。多个 POOL，依次扫描。

序号	列	数据类型	说明
1	POOL_ID	INTEGER	LBS_POOL 编号
2	TS_ID	INTEGER	表空间号
3	FILE_ID	INTEGER	文件号
4	PAGE_NO	INTEGER	页号
5	ACCESS_MAP	INTEGER	曾经访问此数据页的 EP
6	FRESH_EP	INTEGER	最新数据所在 EP
7	FRESH_LSN	BIGINT	最新修改对应的 LSN 值
8	N_REPLACED	INTEGER	控制块被替换次数
9	N_REVOKED	INTEGER	权限被回收次数
10	N_FIXED	INTEGER	引用记数
11	MODE	INTEGER	获得 GBS 授权的封锁模式
12	PHY_LSN	BIGINT	数据页上的最新 LSN 值
13	N_REQUEST	INTEGER	请求获得授权的工作线程数

155. V$RAC_LBS_CTL_DETAIL

显示 LBS 控制块详细信息。多个 POOL，依次扫描。

序号	列	数据类型	说明
1	POOL_ID	INTEGER	LBS_POOL 编号
2	TS_ID	INTEGER	表空间号
3	FILE_ID	INTEGER	文件号
4	PAGE_NO	INTEGER	页号
5	ACCESS_MAP	INTEGER	曾经访问此数据页的 EP
6	FRESH_EP	INTEGER	最新数据所在 EP
7	FRESH_LSN	BIGINT	最新修改对应的 LSN 值
8	N_REPLACED	INTEGER	控制块被替换次数
9	N_REVOKED	INTEGER	权限被回收次数
10	N_FIXED	INTEGER	引用记数
11	MODE	INTEGER	获得 GBS 授权的封锁模式
12	PHY_LSN	BIGINT	数据页上的最新 LSN 值
13	N_REQUEST	INTEGER	请求获得授权的工作线程数
14	REQUEST_MODE	INTEGER	请求的封锁模式
15	REVOKE_LSN	BIGINT	回收权限时，GBS 上的最新 LSN 值

156. V$RAC_LBS_CTL_LRU_FIRST_DETAIL

显示 LBS 的 LRU_FIRST 控制块详细信息。多个 POOL，依次扫描。

序号	列	数据类型	说明

1	POOL_ID	INTEGER	LBS_POOL 编号
2	TS_ID	INTEGER	表空间号
3	FILE_ID	INTEGER	文件号
4	PAGE_NO	INTEGER	页号
5	ACCESS_MAP	INTEGER	曾经访问此数据页的 EP
6	FRESH_EP	INTEGER	最新数据所在 EP
7	FRESH_LSN	BIGINT	最新修改对应的 LSN 值
8	N_REPLACED	INTEGER	控制块被替换次数
9	N_REVOKED	INTEGER	权限被回收次数
10	N_FIXED	INTEGER	引用记数
11	MODE	INTEGER	获得 GBS 授权的封锁模式
12	PHY_LSN	BIGINT	数据页上的最新 LSN 值
13	N_REQUEST	INTEGER	请求获得授权的工作线程数
14	REQUEST_MODE	INTEGER	请求的封锁模式
15	REVOKE_LSN	BIGINT	回收权限时，GBS 上的最新 LSN 值

157. V$RAC_LBS_CTL_LRU_LAST_DETAIL

显示 LBS 的 LRU_LAST 控制块详细信息。多个 POOL，依次扫描。

序号	列	数据类型	说明
1	POOL_ID	INTEGER	LBS_POOL 编号
2	TS_ID	INTEGER	表空间号
3	FILE_ID	INTEGER	文件号
4	PAGE_NO	INTEGER	页号
5	ACCESS_MAP	INTEGER	曾经访问此数据页的 EP
6	FRESH_EP	INTEGER	最新数据所在 EP
7	FRESH_LSN	BIGINT	最新修改对应的 LSN 值
8	N_REPLACED	INTEGER	控制块被替换次数
9	N_REVOKED	INTEGER	权限被回收次数
10	N_FIXED	INTEGER	引用记数
11	MODE	INTEGER	获得 GBS 授权的封锁模式
12	PHY_LSN	BIGINT	数据页上的最新 LSN 值
13	N_REQUEST	INTEGER	请求获得授权的工作线程数
14	REQUEST_MODE	INTEGER	请求的封锁模式
15	REVOKE_LSN	BIGINT	回收权限时，GBS 上的最新 LSN 值

158. V$RAC_GTV_SYS

显示 GTV 控制结构的信息。

序号	列	数据类型	说明
1	T_INFO_NUM	INTEGER	系统已提交、未 PURGE 事务所修改的表对象个数
2	NEXT_TRXID	BIGINT	下一个事务 ID
3	MAX_PURGABLE_TRXID	BIGINT	最大可 PURGE 的事务 ID
4	UNDO_TRXID	BIGINT	回滚段中，正在被访问的最小事务 ID
5	UNDO_CNT	INTEGER	UNDO_TRXID 被设置的次数

159. V$RAC_GTV_TINFO

显示 TINFO 控制结构的信息。

序号	列	数据类型	说明
1	TABLE_ID	INTEGER	系统已提交、未 PURGE 事务所修改的表 ID

2	TRX_ID	BIGINT	修改操作的事务 ID
3	ROWCNT	BIGINT	修改操作影响的行数

160. V$RAC_GTV_ACTIVE_TRX

显示全局活动事务信息。

序号	列	数据类型	说明
1	EP_SEQNO	INTEGER	事务所在的站点号
2	TRX_ID	BIGINT	事务 ID
3	NEXT_TRXID	BIGINT	下一个事务 ID
4	MIN_ACTIVE_TRXID	BIGINT	活动事务链表中的最小事务 ID
5	REFED_TRXID	BIGINT	活动事务链表中的事务 ID

161. V$RAC_LOCK

显示全局活动的事务锁信息。

序号	列	数据类型	说明
1	EP_SEQNO	INTEGER	拥有该锁的站点号
2	ADDR	BIGINT	锁地址
3	TRX_ID	BIGINT	所属事务 ID
4	LTYPE	VARCHAR(10)	锁类型：TID 锁、对象锁
5	LMODE	CHAR(2)	锁模式：S 锁、X 锁、IX 锁、IS 锁
6	BLOCKED	INTEGER	是否阻塞
7	TABLE_ID	INTEGER	对应表锁、字典对象 ID
8	ROW_IDX	BIGINT	被封锁事务 ID

162. V$RAC_TRX

显示所有活动事务的信息。通过该视图可以查看所有系统中所有的事务以及相关信息，如锁信息等。

序号	列	数据类型	说明
1	EP_SEQNO	INTEGER	事务所在站点号
2	ID	BIGINT	当前活动事务的 ID 号
3	NEXTID	BIGINT	下一个事务 ID 号
4	MIN_ACTIVE_ID	BIGINT	所有活动事务 ID 号最小者
5	STATUS	VARCHAR(20)	当前事务的状态
6	ISOLATION	INTEGER	事务隔离级，0：读未提交；1：读提交；2：可重复读；3：串行化
7	READ_ONLY	CHAR	是否是一个只读事务
8	SESS_ID	BIGINT	当前事务的所在会话
9	SESS_SEQ	INTEGER	会话序列号，用来唯一标识会话
10	INS_CNT	INTEGER	插入回滚记录个数
11	DEL_CNT	INTEGER	删除回滚记录个数
12	UPD_CNT	INTEGER	更新回滚记录个数
13	UPD_INS_CNT	INTEGER	更新插入回滚记录个数
14	UREC_SEQNO	INTEGER	当前 Undo 记录的递增序列号
15	WAITING	BIGINT	事务等待的锁

163. V$RAC_TRXWAIT

显示事务等待信息。

序号	列	数据类型	说明

1	EP_SEQNO	INTEGER	事务所在站点号
2	ID	BIGINT	事务 ID
3	WAIT_FOR_ID	BIGINT	所等待的事务 ID
4	WAIT_SEQNO	INTEGER	等待的事务所在站点号
5	WAIT_TIME	INTEGER	当前已等待时间（s）

164. V$RAC_TRX_VIEW

显示当前事务可见的所有活动事务视图信息。根据达梦多版本规则，通过该视图可以查询系统中自己所见的事务信息；可以通过与 V$RAC_TRX 表的连接查询它所见事务的具体信息。

序号	列	数据类型	说明
1	SELF_ID	BIGINT	活动事务 ID
2	EP_SEQNO	INTEGER	可见事务所在站点号
3	ACTIVE_ID	BIGINT	所见的事务活动事务 ID

165. V$ASMATTR

如果使用有 ASM 文件系统，可通过此视图查看 ASM 文件系统相关属性。

序号	列	数据类型	说明
1	AU_SIZE	INTEGER	单个 AU 大小，单位为字节
2	EXTENT_SIZE	INTEGER	一个簇包含的 AU 个数
3	LOCAL_CODE	VARCHAR(64)	当前所连接的 ASMSERVER 的编码格式
4	LOCAL_LANG	VARCHAR(64)	当前所连接的 ASMSERVER 使用的语言： CN:中文 EN:英文
5	USE_SHM	VARCHAR(8)	是否使用共享内存，TRUE/FALSE

166. V$ASMGROUP

如果使用有 ASM 文件系统，可通过此视图查看 ASM 磁盘组信息。

序号	列	数据类型	说明
1	GROUP_ID	INTEGER	磁盘组 ID
2	GROUP_NAME	VARCHAR(128)	磁盘组名称
3	N_DISK	INTEGER	磁盘组中包含的磁盘个数
4	AU_SIZE	INTEGER	单个 AU 大小，单位为字节
5	EXTENT_SIZE	INTEGER	一个簇包含的 AU 个数
6	TOTAL_SIZE	INTEGER	磁盘组总大小，单位为 M
7	FREE_SIZE	INTEGER	磁盘组空闲大小，单位为 M
8	TOTAL_FILE_NUM	INTEGER	磁盘组中总的文件个数，包括文件和目录

167. V$ASMDISK

如果使用有 ASM 文件系统，可通过此视图查看所有的 ASM 磁盘信息。

序号	列	数据类型	说明
1	GROUP_ID	INTEGER	所在的磁盘组 ID，如果是未使用的磁盘，则值为-1
2	DISK_ID	INTEGER	磁盘 ID，如果是未使用的磁盘，则值为-1
3	DISK_NAME	VARCHAR(128)	磁盘名称
4	DISK_PATH	VARCHAR(256)	磁盘路径
5	SIZE	BIGINT	磁盘大小，单位为 M

6	FREE_AUNO	BIGINT	磁盘最大 AU 号
7	CREATE_TIME	VARCHAR(64)	磁盘创建时间
8	MODIFY_TIME	VARCHAR(64)	磁盘最近一次修改时间

168. V$ASMFILE

如果使用有 ASM 文件系统，可通过此视图查看所有的 ASM 文件信息。

序号	列	数据类型	说明
1	FILE_ID	BIGINT	文件 ID
2	TYPE	VARCHAR(32)	类型，目录或文件
3	PATH	VARCHAR(256)	文件完整路径
4	SIZE_BYTES	BIGINT	文件实际大小，单位为字节， 目录类型的文件不占用空间，值为 0
5	SIZE_TOTAL	BIGINT	文件占用总空间大小，单位为字节， 目录类型的文件不占用空间，值为 0
6	CREATE_TIME	VARCHAR(64)	文件创建时间
7	MODIFY_TIME	VARCHAR(64)	文件修改时间
8	GROUP_ID	INTEGER	所在磁盘组 ID
9	DISK_ID	INTEGER	inode 项所在磁盘 ID
10	DISK_AUNO	INTEGER	inode 项所在磁盘 AU 编号
11	AU_OFFSET	INTEGER	inode 项在 AU 内的偏移

169. V$DCR_INFO

查看 DCR 配置的全局信息。

序号	列	数据类型	说明
1	VERSION	INTEGER	DCR 版本号
2	N_GROUP	INTEGER	DCR 配置的组个数
3	VTD_PATH	VARCHAR(256)	Voting Disk 路径
4	UDP_FLAG	INTEGER	是否使用 UDP 心跳机制，已无效
5	UDP_OGUID	BIGINT	校验用

170. V$DCR_GROUP

查看 DCR 配置的组信息。

序号	列	数据类型	说明
1	GROUP_TYPE	VARCHAR(32)	组类型，CSS/ASM/DB
2	GROUP_NAME	VARCHAR(128)	组名称
3	N_EP	INTEGER	组中配置的 EP 个数
4	DSKCHK_CNT	INTEGER	磁盘容错时间，单位秒
5	NETCHK_TIME	INTEGER	网络容错时间，单位秒

171. V$DCR_EP

查看 DCR 配置的节点信息。

序号	列	数据类型	说明
1	GROUP_NAME	VARCHAR(128)	节点所属的组名
2	EP_NAME	VARCHAR(128)	节点名称
3	EP_SEQNO	INTEGER	节点的组内序号： 对 CSS/ASM 组的节点，是自动分配的序号， 对 DB 组的节点，如果没有配置，也是自动分配的序号，否则是实际的配置序号。

4	EP_HOST	VARCHAR(128)	节点的 IP 地址，对 CSS/ASM 组的节点有效，表示登录节点的 IP 地址
5	EP_PORT	INTEGER	节点的 TCP 监听端口,对 CSS/ASM 组的节点有效，对应登录节点的端口号
6	UDP_PORT	INTEGER	节点的 UDP 监听端口，已无效
7	SHM_KEY	INTEGER	共享内存标识，初始化共享内存的标识符，对 ASM 组的节点有效
8	SHM_SIZE	INTEGER	共享内存大小，单位 M，初始化共享内存大小，对 ASM 组的节点有效
9	ASM_LOAD_PATH	VARCHAR(256)	ASM 磁盘扫描路径，对 ASM 组的节点有效

172. V$RAC_REQUEST_STATISTIC

统计 RAC 环境内 TYPE 类型请求时间。

序号	列	数据类型	说明
1	TYPE	VARCHAR(64)	请求类型
2	TOTAL_REQUEST_COUNT	BIGINT	总请求次数
3	MAX_REQUEST_TIME	INTEGER	最大请求时间，单位为微秒
4	MIN_REQUEST_TIME	INTEGER	最小请求时间，单位为微秒
5	AVERAGE_REQUEST_TIME	INTEGER	平均请求时间，单位为微秒
6	AVERAGE_RLOG_FLUSH_TIME	INTEGER	平均等待日志刷盘时间，单位为微秒

173. V$RAC_REQUEST_PAGE_STATISTIC

统计 lbs_XX 类型最耗时的前 100 页地址信息。

序号	列	数据类型	说明
1	TYPE	VARCHAR(64)	请求类型
2	TS_ID	INTEGER	表空间 ID
3	FILE_ID	INTEGER	文件 ID
4	PAGE_NO	INTEGER	页号
5	REQUEST_TIME	INTEGER	花费时间，单位为微秒

17) DCP

174. V$DCPINSTS

仅当 INI 参数 ENABLE_DCP_MODE 为 1 时才能查询此动态视图，显示 DCP 对应 MPP 集群的所有节点信息。

序号	列	数据类型	说明
1	NAME	VARCHAR(128)	节点实例名称
2	EPNO	INTEGER	节点编号
3	IP	VARCHAR(100)	IP 地址
4	PORT	INTEGER	端口号

175. V$DCP_CONNPOOL

仅当 INI 参数 ENABLE_DCP_MODE 为 1 时才能查询此动态视图，显示 DCP 缓冲区的信息。

序号	列	数据类型	说明

1	ID	BIGINT	会话 ID
2	USERNAME	VARCHAR(128)	会话用户名
3	CUR_SCH	VARCHAR(128)	当前模式名
4	EPNO	INTEGER	当前连接的 MPP 的 EPNO
5	EXEC_MODE	INTEGER	当前事务执行模式，1：探测；2：强制
6	FREE_FLAG	CHAR	是否空闲，Y：是；N：否
7	RELEASE_TS	INTEGER	上一次放回缓冲区的时间戳，仅当 FREE_FALG=Y 时有效

176. V$INSTANCE_LOG_HISTORY

用于查询服务器实例运行期间生成的最近 1 万条事件日志。

序号	参数	说明
1	SEQNO	日志序号，从 0 开始递增。每次服务器重启后清零
2	LOG_TIME	日志生成时间，精确到秒
3	PID	生成日志的进程 ID
4	THREAD_NAME	生成日志的线程名
5	LEVEL$	日志级别描述。可取值为：INFO、WARN、ERROR、FATAL
6	TXT	日志详细内容

18) 数据守护

177. V$RECOVER_STATUS

该视图需要在主库上查询，用于查询备库的恢复进度，如果已恢复完成，查询结果为空。

注意这里显示的是主库向备库发送日志的进度，由于备库重做日志也需要时间，在最后一批日志发送完成后，KBYTES_TO_Recover 为 0，Recover_PERCENT 为 100%，表示主库已经完成所有日志发送，需要等待备库将最后一批日志重做完成，此时主库的守护进程可能仍然处于 Recovery 状态，待备库重做完成后，主库的守护进程会自动切换 Open 状态。

```
RECOVER_PERCENT = (KBYTES_TOTAL - KBYTES_TO_Recover)/ KBYTES_TOTAL
```

序号	列	数据类型	说明
1	PRIMARY_NAME	VARCHAR(256)	主库实例名
2	STANDBY_NAME	VARCHAR(256)	当前正在恢复的备库实例名
3	PRIM_LSN	BIGINT	主库当前的 FLSN
4	SEND_LSN	BIGINT	主库已发给备库的最大 LSN
5	KBYTES_TOTAL	BIGINT	主库本地的日志总量，单位为 KB
6	KBYTES_TO_RECOVER	BIGINT	主库需要发送给备库的日志量，单位为 KB
7	FILES_TO_RECOVER	INTEGER	主库需要发给备库重做的日志文件个数
8	RECOVER_PERCENT	VARCHAR(64)	主库向备库已发送日志的比例

178. V$KEEP_BUF

该视图需要在备库上查询，用于查询备库上的 KEEP_BUF 信息。专门用于实时主备和 MPP 主备。

读写分离集群下备库没有 KEEP_BUF 机制，该视图查询结果为空。

序号	列	数据类型	说明
1	TYPE	VARCHAR(64)	KEEP_BUF 中保存的日志类型。
2	MIN_LSN	BIGINT	KEEP_BUF 中保存的最小 LSN

3	MAX_LSN	BIGINT	KEEP_BUF 中保存的最大 LSN
4	PTX_OFF	SMALLINT	KEEP_BUF 中的 PTX 偏移
5	LOG_SIZE	INTEGER	KEEP_BUF 中的日志长度，单位为字节

179. V$RAPPLY_SYS

该视图需要在备库上查询，用于查询备库重做日志时的一些系统信息。

序号	列	数据类型	说明
1	APPLYING	VARCHAR(64)	备库的重做线程是否正在处理任务，查询结果为 TRUE 或 FALSE
2	SEC_MAX_LSN	BIGINT	备库收到的第二大 LSN 值
3	MAX_LSN	BIGINT	备库收到的最大 LSN 值
4	TASK_NUM	INTEGER	备库上的重做任务数
5	PRIMARY_NAME	VARCHAR(128)	备库对应的主库实例名
6	HAS_KEEP_BUF	VARCHAR(64)	备库上是否有 KEEP_BUF,查询结果为 TRUE 或 FALSE
7	TASK_MEM_USED	BIGINT	备库上重做日志堆积占用的内存（字节）
8	TASK_START_TIME	TIMESTAMP	当前正在重做日志 BUF 的开始时间
9	LAST_REDO_TIME	BIGINT	最近一次日志 BUF 重做的耗时（ms）
10	TASK_NUM_APPLIED	BIGINT	备库上已经重做的日志 BUF 总个数
11	APPLIED_TOTAL_TIME	BIGINT	备库上重做的日志 BUF 的总时间（ms）

180. V$RAPPLY_LOG_TASK

该视图需要在备库上查询，用于查询备库当前重做任务的日志信息。

序号	列	数据类型	说明
1	TYPE	VARCHAR(64)	日志类型
2	MIN_LSN	BIGINT	日志的最小 LSN 值
3	MAX_LSN	BIGINT	日志的最大 LSN 值
4	PTX_OFF	SMALLINT	重做日志的 PTX 偏移
5	LOG_SIZE	INTEGER	日志长度，单位为字节

181. V$ARCH_FILE

查询本地归档日志信息。

序号	列	数据类型	说明
1	DB_MAGIC	INTEGER	数据库的 MAGIC 值
2	STATUS	VARCHAR(64)	归档日志状态
3	LEN	BIGINT	日志长度，单位为字节
4	FREE	BIGINT	写日志偏移位置
5	ARCH_LSN	BIGINT	归档文件起始 LSN，仅对归档文件有意义
6	CLSN	BIGINT	检查点的 LSN
7	ARCH_SEQ	BIGINT	归档文件起始 SEQ，仅对归档文件有意义
8	NEXT_SEQ	BIGINT	下一个 SEQ 值
9	CREAT_TIME	TIMESTAMP	归档文件创建时间
10	CLOSE_TIME	TIMESTAMP	归档文件关闭时间
11	PATH	VARCHAR(256)	归档文件所在路径，包括归档文件名

182. V$ARCH_STATUS

查询归档状态信息。

序号	列	数据类型	说明

1	ARCH_TYPE	VARCHAR(256)	归档类型
2	ARCH_DEST	VARCHAR(256)	归档目标，本地归档为归档路径，其他类型为归档目标实例名。
3	ARCH_STATUS	VARCHAR(256)	归档状态，Valid 为有效状态，Invalid 为无效状态。

183. V$MAL_LINK_STATUS

查询本地实例到远程实例的 MAL 链路连接状态。

序号	列	数据类型	说明
1	SRC_SITE	VARCHAR(256)	源站点实例名
2	DEST_SITE	VARCHAR(256)	目的站点实例名。
3	CTL_LINK_STATUS	VARCHAR(256)	控制链路连接状态，CONNECTED 表示连接已建立，DISCONNECT 表示连接断开。
4	DATA_LINK_STATUS	VARCHAR(256)	数据链路连接状态，CONNECTED 表示连接已建立，DISCONNECT 表示连接断开。

184. V$DMWATCHER

查询当前登录实例所对应的守护进程信息，注意一个守护进程可以同时守护多个组的实例，因此查询结果中部分字段（N_GROUP、SWITCH_COUNT）为守护进程的全局信息，并不是当前登录实例自身的守护信息。

另外 MPP 主备环境下，全局登录方式返回的是所有 MPP 站点上查询返回的守护进程信息，可以根据 INST_NAME 实例名字段来区分。

序号	列	数据类型	说明
1	N_GROUP	INTEGER	护进程全局信息，指实例所在的守护进程守护的组个数
2	GROUP_NAME	VARCHAR(128)	实例所在的守护进程组名
3	INST_NAME	VARCHAR(128)	实例名
4	DW_TYPE	VARCHAR(32)	实例所在守护进程组的守护类型
5	DW_MODE	VARCHAR(32)	实例所在守护进程组的守护模式
6	AUTO_RESTART	INTEGER	守护进程对本实例是否配置有自动重启， 1：自动重启 0：不自动重启
7	DW_STATUS	VARCHAR(64)	实例的守护进程状态
8	DW_SUB_STATUS	VARCHAR(64)	实例的守护进程子状态
9	LAST_MSG_TIME	DATETIME	实例最近一次收到守护进程消息的时间
10	SWITCH_COUNT	INTEGER	守护进程全局信息，指守护进程组内主库的变迁次数（包括 SWITCHOVER 主备库切换，TAKEOVER 手动/自动接管，TAKEOVER 强制接管导致的主库变迁操作）。 注意，变迁次数从当前连接实例的 dmwatcher.ctl 中的记录统计得出（只统计上面描述的操作，强制 OPEN 不包括在内），如果监视器执行了清理 dmwatcher.ctl 的操作，则变迁次数也会被清零，被清理掉的记录不会再被统计。
11	CTL_NUM	INTEGER	实例的守护进程收到的远程实例控制文件信息个数
12	INST_NUM	INTEGER	实例的守护进程收到的远程实例信息个数

13	MAX_CONN_NUM	INTEGER	实例的守护进程当前最大的 TCP 连接数

185. V$UTSK_INFO

查询守护进程向服务器发送请求的执行情况。

注意在 RUNNING 字段值为 TRUE 时，DSEQ、CODE 和 SEND_LSN 的值才有意义，另外 SEND_LSN 和 Recover_BREAK 在主库上查询才有意义，并且需要主库的守护进程处于 Recovery 状态。

序号	列	数据类型	说明
1	RUNNING	VARCHAR(64)	当前系统是否正在处理守护进程请求，TRUE 表示正在处理，FALSE 表示没有。
2	DSEQ	INTEGER	守护进程发送的报文序号
3	CODE	INTEGER	DSEQ 对应的执行结果
4	SEND_LSN	BIGINT	主库发送给备库的最大归档 LSN
5	RECOVER_BREAK	VARCHAR(64)	用于显示主库的 Recover 流程是否被中断，TRUE 表示被中断，FALSE 表示没有
6	AUTO_SWITCH	VARCHAR(64)	守护系统配置的切换模式，TRUE 表示故障自动切换，FALSE 表示故障手动切换。

19) 系统包

186. V$CACHEPKG

显示当前系统中的包的使用信息。

序号	列	数据类型	说明
1	NAME	VARCHAR(128)	包名
2	N_FIXED	INTEGER	包被引用的次数
3	DISABLED	CHAR	包是否可用
4	MEM_SIZE	INTEGER	包的内存大小
5	ADDRESS	VARBINARY(8)	包的地址
6	ID	INTEGER	包 ID
7	CUR_SCHID	INTEGER	当前的模式 ID
8	CUR_USERID	INTEGER	当前的用户 ID
9	IS_DEF_INVOKER	CHAR(1)	是否已定义者权限调用

187. V$DBMS_LOCKS

显示当前系统中的申请的 DBMS_LOCK 包封锁情况。

序号	列	数据类型	说明
1	HANDLE	VARCHAR(128)	封锁的锁对象句柄
2	ID	INTEGER	封锁的锁对象 ID
3	MODE	VARCHAR(10)	封锁的模式
4	OWN_SESS	BIGINT	申请封锁的 session
5	WAIT_SESS	BIGINT	封锁等待的 session，无等待为 NULL

188. V$DB_PIPES

记录使用 DBMS_PIPE 包创建的管道的相关信息。

序号	列名	类型	说明
1	NAME	VARCHAR(30) NOT NULL	管道的名字
2	OWNERID	BIGINT NOT NULL	管道拥有者 ID

3	TYPE	VARCHAR(7)	管道类型:PUBLIC/PRIVATE
4	PIPE_SIZE	BIGINT	管道的长度

20) 捕获

189. V$CAPTURE

显示捕获信息。

序号	列	数据类型	说明
1	STATE	INTEGER	捕获状态
2	VERSION	SMALLINT	捕获文件的版本
3	MSG_NUM	BIGINT	捕获消息的条数
4	FILE_PATH	VARCHAR(256)	捕获文件路径
5	INIT_TIME	TIMESTAMP(0)	捕获开始时间
6	DURATION	INTEGER	捕获持续时间
7	FLUSH_BUF_NUM	INTEGER	待刷盘缓冲数
8	FREE_BUF_NUM	INTEGER	空闲缓冲数

21) 审计与加密

190. V$AUDITRECORDS

显示审计记录，用来查询当前系统默认路径下的审计文件信息。此动态性能视图只有在审计开关打开时才有内容，且只有审计用户可以查询。

序号	列	数据类型	说明
1	USERID	INTEGER	用户 ID
2	USERNAME	VARCHAR(128)	用户名
3	ROLEID	INTEGER	角色 ID。 没有具体角色的用户和 SQL 序列审计，没用角色信息。
4	ROLENAME	VARCHAR(128)	角色名。 没有具体角色的用户和 SQL 序列审计，没用角色信息。
5	IP	VARCHAR(25)	IP 地址
6	SCHID	INTEGER	模式 ID
7	SCHNAME	VARCHAR(128)	模式名
8	OBJID	INTEGER	对象 ID
9	OBJNAME	VARCHAR(128)	对象名
10	OPERATION	VARCHAR(128)	操作类型名
11	SUCC_FLAG	CHAR(1)	成功标记
12	SQL_TEXT	VARCHAR(8188)	SQL 文本
13	DESCRIPTION	VARCHAR(8188)	描述信息
14	OPTIME	DATETIME	操作时间
15	MAC	VARCHAR(25)	操作对应的 MAC 地址
16	SEQNO	TINYINT	DMDSC 环境下表示生成审计记录的节点号，非 DMDSC 环境下始终 0

191. V$CIPHERS

显示系统加密算法信息。

序号	列	数据类型	说明
1	CYT_ID	INTEGER	算法 ID
2	CYT_NAME	VARCHAR(256)	算法名
3	CYT_TYPE	INTEGER	算法类型 1：分组对称加密算法 2：流式对称加密算法 3：非对称加密算法，保留 4：散列算法
4	BLOCK_SIZE	INTEGER	块大小
5	KH_SIZE	INTEGER	KEY 或 HASH 大小

192. V$EXTERNAL_CIPHERS

显示系统中所有的第三方加密算法信息。

序号	列	数据类型	说明
1	ID	INTEGER	算法 ID
2	NAME	VARCHAR(128)	算法名
3	LIB	VARCHAR(300)	算法所在的 lib 库文件名
4	VALID	CHAR	算法是否有效。‘Y’：是；‘N’：否

附录 3 执行计划操作符

操作符名称	说明
AAGR2	简单聚集：如果没有分组（group by）， 则总的就一个组，直接计算聚集函数
ACTRL	控制备用计划转换
AFUN	分析函数计算
ASCN	数组当作表来扫描
ASSERT	约束检查
BLKUP2	定位查找
BMAND	位图索引的与运算
BMCNT	位图索引的行数计算
BMCVT	位图索引的 ROWID 转换
BMMG	位图索引归并
BMOR	位图索引的或运算
BMSEK	位图索引的范围查找
CONST VALUE LIST	常量列表
CONSTC	用于复合索引跳跃扫描
CSCN2	聚集索引扫描
CSEK2	聚集索引数据定位
CTNS	用于实现全文索引的 CONTAINS
DELETE	删除数据
DELETE_REMOTE	DBLINK 删除操作
DISTINCT	去重
DSCN	动态视图表扫描
DSSEK	DISTINCT 列上索引跳跃扫描（单列索引或复合索引）
ESCN	外部表扫描
EXCEPT	集合的差运算，且取差集后删除重复项
EXCEPT ALL	集合的差运算，且取差集后不删除重复项
FAGR2	快速聚集，如果没有 where 条件，且取 count(*)，或者基于索引的 MAX/MIN 值，则可以快速取得集函数的值
FILL BTR	填充 B 树
FTTS	MPP\LPQ 下，对临时表的优化
GSEK	空间索引查询
HAGR2	HASH 分组，并计算聚集函数
HASH FULL JOIN2	HASH 全外连接
HASH LEFT JOIN2	HASH 左外连接
HASH LEFT SEMI JOIN2	HASH 左半连接
HASH LEFT SEMI MULTIPLE JOIN	多列 NOT IN
HASH RIGHT JOIN2	HASH 右外连接
HASH RIGHT SEMI JOIN2	HASH 右半连接
HASH RIGHT SEMI JOIN32	用于 OP SOME/ANY/ALL 的 HASH 右半连接

HASH2 INNER JOIN	HASH 内连接
HEAP TABLE	临时结果表
HEAP TABLE SCAN	临时结果表扫描
HFD	删除事务性型 HUGE 表数据
HFDEL2	删除非事务性型 HUGE 表数据
HFDEL_EP	MPP 下从 EP 删除非事务性型 HUGE 表数据
HFD_EP	MPP 下从 EP 删除事务性型 HUGE 表数据
HFI	事务型 HUGE 表插入记录
HFI2	MPP 下优化的事务型 HUGE 表插入记录
HFINS2	非事务型 HUGE 表插入记录
HFINS3	MPP 下优化的非事务型 HUGE 表插入记录
HFINS4	非 MPP 下，针对非事务型 HUGE 水平分区主表的插入优化，需要参数 HFINS_PARALLEL_FLAG=2
HFINS_EP	MPP 下从 EP 插入非事务型 HUGE 表数据
HFI_EP	MPP 下从 EP 插入事务型 HUGE 表数据
HFLKUP	根据 ROWID 检索非事务型 HUGE 表数据
HFLKUP2	根据 ROWID 检索事务型 HUGE 表数据
HFLKUP_EP	MPP 下从 EP 根据 ROWID 检索非事务型 HUGE 表数据
HFLKUP2_EP	MPP 下从 EP 根据 ROWID 检索事务型 HUGE 表数据
HFSCN	非事务型 HUGE 表的逐行扫描
HFSCN2	事务型 HUGE 表的逐行扫描
HFSEK	根据 KEY 检索非事务型 HUGE 表数据
HFSEK2	根据 KEY 检索事务型 HUGE 表数据
HFU	更新事务型 HUGE 表数据
HFUPD	更新非事务型 HUGE 表数据
HFUPD_EP	MPP 下从 EP 更新非事务型 HUGE 表数据
HFU_EP	MPP 下从 EP 更新事务型 HUGE 表数据
HIERARCHICAL QUERY	层次查询
HPM	水平分区表归并排序
INDEX JOIN LEFT JOIN2	索引左连接
INDEX JOIN SEMI JOIN2	索引半连接
INSERT	插入记录
INSERT3	MPP 下，查询插入优化处理
INSERT_LIST	堆表插入
INSERT_REMOTE	DBLINK 插入操作
INTERSECT	集合的交运算，且取交集后删除重复项
INTERSECT ALL	集合的交运算，且取交集后不删除重复项
LOCAL BROADCAST	本地并行模式下，消息广播到各线程，包含必要的聚集函数合并计算
LOCAL COLLECT	本地并行下数据收集处理，代替 LOCAL GATHER
LOCAL DISTRIBUTE	本地并行模式下，消息各线程的相互重分发
LOCAL GATHER	本地并行模式下，消息收集到主线程
LOCAL SCATTER	本地并行模式下，主线程向各从线程广播消息
LOCK TID	上锁
LSET	DBLINK 查询结果集

MERGE INNER JOIN3	归并内连接
MERGE SEMI JOIN3	归并半连接
MPP BROADCAST	MPP 模式下，消息广播到各站点，包含必要的聚集函数合并计算
MPP COLLECT	用于替换顶层 MPP GATHER，除了收集数据到主节点，还增加主从节点间的同步执行功能，防止从节点不断发送数据到主节点造成邮件堆积
MPP DISTRIBUTE	MPP 模式下，消息各站点的相互重分发
MPP GATHER	MPP 模式下，消息收集到主站点
MPP SCATTER	MPP 模式下，主站点向各从站点广播消息
MSYNC	MPP 下数据同步处理
MVCC CHECK	多版本检查
NCUR2	游标操作
NEST LOOP FULL JOIN2	嵌套循环全外连接
NEST LOOP INDEX JOIN2	索引内连接
NEST LOOP INNER JOIN2	嵌套循环内连接
NEST LOOP LEFT JOIN2	嵌套循环左外连接
NEST LOOP SEMI JOIN2	嵌套循环半连接
NTTS2	临时表，临时存放数据
NSET2	结果集(result set)收集，一般是查询计划的顶层节点
PARALLEL	控制水平分区子表的扫描
PIPE2	管道；先做一遍右儿子，然后执行左儿子，并把左儿子的数据向上送，直到左儿子不再有数据
PRJT2	关系的"投影"(project)运算，用于选择表达式项的计算
PSCN	批量参数当作表来扫描
REMOTE SCAN	DBLINK 远程表扫描
RN	实现 ROWNUM 查询
RNSK	ROWNUM 作为过滤条件时的计算处理
SAGR2	如果输入流是有序的，则使用流分组，并计算聚集函数
SELECT INTO2	查询插入
SET TRANSACTION	事务操作(START 除外)
SLCT2	关系的"选择"（select）运算，用于查询条件的过滤
SORT2	排序
SORT3	排序
SPL2	临时表；和 NTTS2 不同的是，它的数据集不向父亲节点传送，而是被编号，用编号和 KEY 来定位访问；而 NTTS2 的数据，主动传递给父亲节点
SSCN	直接使用二级索引进行扫描
SSEK2	二级索引数据定位
START TRANSACTION	启动会话
STAT	统计信息计算
TOPN2	取前 N 条记录
UFLT	处理 UPDATE FROM 子句
UNION	UNION 计算
UNION ALL	UNION ALL 运算

UNION ALL(MERGE)	UNION ALL 运算（使用归并）
UNION FOR OR2	OR 过滤的 UNION 计算
UPDATE	更新数据
UPDATE_REMOTE	DBLINK 更新操作

附录 4 数据复制的系统表

1） 复制组表

```
CREATE TABLE SYSREP.RPS_GROUPS (
    NAME        VARCHAR(128),       // 复制组名
    ID          INT,                    // 复制组 ID
    DESC$       VARCHAR(1000),      // 描述
    CLUSTER PRIMARY (NAME)
);
```

2） 复制节点实例表

```
CREATE TABLE SYSREP.RPS_INSTANCES(
    INST_NAME       VARCHAR(128),   // 复制节点实例名
    GRP_ID          INT,                // 复制组 ID
    INST_ID         INT,                // 实例在复制组中编号
    FAULT_TIMEOUT   INT,            // 故障超时处理值，以秒为单位，0 为立即超时
    VALID_FLAG      CHAR(1),            // 节点系统状态
    FAULT_TIME      DATETIME,       // 节点故障开始时间
    NET_VALID_FLAG  CHAR(1),            // 网络状态
    NET_FAULT_TIME  DATETIME,       // 网络故障开始时间
    CLUSTER PRIMAY KEY(GRP_ID, INST_NAME)
);
```

3） 复制关系表

```
CREATE TABLE SYSREP.RPS_REPLICATIONS(
    REP_NAME        VARCHAR(128),   // 复制名
    GRP_ID          INT,                // 复制组 ID
    REP_ID          INT,                // 复制 ID，全局唯一
    MINST_ID        INT,                // 主节点实例编号
    SINST_ID        INT,                // 从节点实例编号
    ARCH_DIR        VARCHAR(256),   // 主节点归档日志路径
    FAULT_TIMEOUT   INT,                // 故障超时处理值，以秒为单位，0 为立即超
时
    VALID_FLAG      CHAR(1),            // 复制关系状态
    FAULT_TIME      DATETIME,       // 故障开始时间
    SYNC_FLAG    INTEGER         //指定同步或异步复制
    TIMER_NAME   VARCHAR(128)    //指定异步复制的定时器（同步复制没有此项）
    DESC$           VARCHAR(1000),  // 复制描述
    CLUSTER PRIMARY KEY(GRP_ID, REP_NAME)
);
```

4） 复制映射表

```
CREATE TABLE SYSREP.RPS_TABMAPS(
    REP_ID         INT,             //复制 ID
    MSCH_NAME      VARCHAR(128),    //主表模式名
    MTAB_NAME      VARCHAR(128),    //主表名
    MSCH_ID       INT,             //主表模式 ID
    MTAB_ID        INT,             //主表 ID
    SSCH_NAME      VARCHAR(128),    //从表模式名
    STAB_NAME      VARCHAR(128),    //从表名
    SSCH_ID        INT,                //从表模式 ID
    STAB_ID        INT,             //从表 ID
    READONLY_MODE INT,              //映射模式 1:只读模式，0：非只读模式
    CLUSTER PRIMARY KEY(REP_ID, MTAB_ID, STAB_ID)
);
```

5） 复制故障历史表

```
CREATE TABLE SYSREP.REP_FAULT_HISTORY(
```

```
    GRP_NAME        VARCHAR(128),   // 复制组
    OBJ_NAME        VARCHAR(128),   // 故障节点或关系
    FAULT_TYPE      VARCHAR(128),   // 故障类型描述
    START_TIME      DATETIME,       // 故障开始时间
    END_TIME        DATETIME        // 故障结束时间
);
```

6） 复制冲突历史表

```
CREATE TABLE SYSREP.RPS_CONFLICTS
(
    SEQ_NO          BIGINT          // 冲突序号
    REP_ID          INT,            // 对应复制号
    INST_ID         INT,            // 产生冲突的节点编号
    TABLE_ID        INT,            // 冲突表 ID
    TYPE            TINYINT          // 操作类型
    OCC_TIME        DATETIME,       // 冲突产生时间
    KEY_DATA        VARCHAR(8000),  // 冲突数据的 PK 值，如包含多个 KEY 值，则以逗号分隔。如键值超长则截断
);
```

7） 复制定时器表

```
CREATE TABLE SYSREP.RPS_TIMERS (
    NAME            VARCHAR(128),       // 定时器名称
    TYPE$       INT,            // 定时类型，取值可参见 SP_RPS_CREATE_TIMER
    FERQ_INTERVAL       INT,            // 间隔天数
    FREQ_SUB_INTERVAL   INT,            // 间隔的月/周（调度类型决定）数
    FREQ_MINUTE_INTERVAL    INT,        // 间隔的分钟数
    START_TIME          TIME,       // 开始时间
    END_TIME            TIME,       // 结束时间
    DURING_START_DATE   DATETIME    // 开始时间点
    DURING_END_DATE DATETIME,       // 结束时间点
    NO_END_DATA_FLAG    INTEGER   //是否有结束日期（0：有结束日期；1：没有结束日期）
    DESC$               VARCHAR(1000),      // 定时器描述
    CLUSTER PRIMARY KEY(REP_NAME)
);
```

附录 5 DM 技术支持

如果您在安装或使用 DM 及其相应产品时出现了问题，请首先访问我们的 Web 站点 http://www.dameng.com/。在此站点我们收集整理了安装使用过程中一些常见问题的解决办法，相信会对您有所帮助。

您也可以通过以下途径与我们联系，我们的技术支持工程师会为您提供服务。

武汉达梦数据库股份有限公司

地址：武汉市关山一路特 1 号光谷软件园 C6 栋 5 层

邮编：430073

电话：(+86)027-87588000

传真：(+86)027-87588000-8039

达梦数据库（北京）有限公司

地址：北京市海淀区北三环西路 48 号数码大厦 B 座 905

邮编：100086

电话：(+86)010-51727900

传真：(+86)010-51727983

达梦数据库（上海）有限公司

地址：上海市闸北区江场三路 28 号 301 室

邮编：200436

电话：(+86)021-33932716

传真：(+86)021-33932718

地址：上海市浦东张江高科技园区博霞路 50 号 403 室

邮编：201203

电话：（+86）021-33932717

传真：（+86）021-33932717-801

达梦数据库（广州）有限公司

地址：广州市荔湾区中山七路 330 号荔湾留学生科技园 703 房

邮编：510145

电话：(+86)020-38371832

传真：(+86)020-38371832

达梦数据库（海南）有限公司

地址：海南省海口市玉沙路富豪花园 B 座 1602 室

邮编：570125

电话：(+86)0898-68533029

传真：(+86)0898-68531910

达梦数据库（南宁）办事处

地址：广西省南宁市科园东五路四号南宁软件园五楼

邮编：530003

电话：（+86）0771-2184078

传真：（+86）0771-2184080

达梦数据库（合肥）办事处
地址：合肥市包河区马鞍山路金帝国际城 7 栋 3 单元 706 室
邮编：230022
电话：（+86）0551-3711086

达梦数据库（深圳）办事处
地址：深圳市福田区皇岗路高科利大厦 A 栋 24E 邮编：518033
电话：0755-83658909
传真：0755-83658909

技术服务：
电话：400-991-6599
邮箱：dmtech@dameng.com