

目录

目录.....	1
第一部分 数据库体系架构	1
第 1 章 DM 逻辑结构概述.....	1
1.1 数据库和实例	1
1.1.1 数据库	1
1.1.2 实例	1
1.2 DM 逻辑存储结构.....	1
1.2.1 表空间	2
1.2.2 块.....	3
1.2.3 簇.....	4
1.2.4 段.....	4
第 2 章 DM 物理存储结构.....	6
2.1 配置文件	6
2.1.1 DM 数据库服务配置	7
2.1.2 数据守护配置	16
2.1.3 复制配置	18
2.1.4 审计配置	20
2.2 控制文件	20
2.3 数据文件	21
2.4 重做日志文件	21
2.5 归档日志文件	22
2.6 逻辑日志文件	22
2.7 备份文件	22
2.8 跟踪日志文件	23
2.9 事件日志文件	23
2.10 数据重演文件	23
第 3 章 DM 内存结构	24
3.1 内存池	24
3.1.1 HEAP.....	24
3.1.2 VPOOL.....	24
3.2 缓冲区.....	25
3.2.1 数据缓冲区	25
3.2.2 日志缓冲区	26
3.2.3 字典缓冲区	26
3.2.4 SQL 缓冲区.....	26
3.3 排序区.....	27
3.4 哈希区.....	27

第 4 章 管理 DM 线程	28
4.1 监听线程	28
4.2 工作线程	28
4.3 IO 线程	28
4.4 调度线程	29
4.5 日志 FLUSH 线程	29
4.6 日志归档线程	29
4.7 日志重做线程	29
4.8 日志 APPLY 线程	30
4.9 定时器线程	30
4.10 逻辑日志归档线程	30
4.11 数据守护相关线程	30
4.12 MAL 系统相关线程	31
4.13 其他线程	31
4.14 线程信息的查看	31
第 5 章 DM7 的升级	32
5.1 选择升级方法	32
5.2 升级前准备工作	32
5.3 使用数据迁移工具	33
5.4 使用数据导入导出工具	33
5.5 升级后期工作	33
第二部分 基础数据库管理	34
第 6 章 DM 系统管理员	34
6.1 DM 系统管理员的类型	34
6.2 数据库管理员的任务	34
6.3 数据库安全员的任务	36
6.4 数据库审计员的任务	36
第 7 章 创建和配置 DM 数据库	37
7.1 创建 DM 数据库	37
7.2 使用数据库配置工具创建数据库	37
7.2.1 启动数据库配置工具	37
7.2.2 使用数据库配置工具创建数据库	38
7.3 使用 dmunit 创建数据库	44
7.3.1 从命令行运行 dmunit	44
7.3.2 双击启动 dmunit	46
7.4 注册数据库服务	46
7.5 查看数据库信息	48
7.6 删除数据库	49
第 8 章 启动和关闭数据库	52
8.1 启动数据库	52
8.1.1 Windows 系统	52
8.1.2 Linux 系统	54

8.2 数据库状态和模式	54
8.3 关闭数据库	55
8.3.1 Windows 系统	55
8.3.2 Linux 系统	56
第 9 章 管理模式对象的空间	57
9.1 设置存储参数	57
9.1.1 普通表和索引	57
9.1.2 LIST 表	58
9.1.3 列存储表	58
9.2 收回多余的空间	58
9.3 用户和表上的空间限制	58
9.3.1 用户的空间限制	58
9.3.2 表对象的空间限制	58
9.4 查看模式对象的空间使用	59
9.4.1 查看用户占用的空间	59
9.4.2 查看表占用的空间	59
9.4.3 查看表使用的页数	59
9.4.4 查看索引占用的空间	59
9.4.5 查看索引使用的页数	59
9.5 数据类型的空间使用	59
第 10 章 管理表	61
10.1 管理表的准则	61
10.1.1 设计表	61
10.1.2 指定表的存储空间上限	62
10.1.3 指定表的存储位置	62
10.2 创建表	62
10.2.1 创建普通表	62
10.2.2 指定表的聚集索引	63
10.2.3 指定表的填充因子	63
10.2.4 查询建表	64
10.2.5 创建临时表	64
10.3 更改表	65
10.4 删除表	65
10.5 清空表	65
10.5.1 使用 DELETE	65
10.5.2 使用 DROP 和 CREATE	66
10.5.3 使用 TRUNCATE	66
10.6 查看表信息	66
10.6.1 查看表定义	66
10.6.2 查看自增列信息	66
10.6.3 查看表的空间使用情况	67
第 11 章 管理索引	68
11.1 管理索引的准则	68
11.1.1 在表中插入数据后创建索引	68
11.1.2 索引正确的表和列	68

11.1.3 为性能而安排索引列.....	68
11.1.4 限制每个表的索引的数量.....	69
11.1.5 估计索引大小和设置存储参数.....	69
11.1.6 为每个索引指定表空间.....	69
11.2 创建索引.....	69
11.2.1 明确地创建索引.....	69
11.2.2 创建聚集索引.....	70
11.2.3 明确地创建唯一索引.....	70
11.2.4 自动创建与约束相关的唯一索引.....	71
11.2.5 创建基于函数的索引.....	71
11.2.6 创建位图索引.....	71
11.2.7 创建位图连接索引.....	72
11.3 重建索引.....	72
11.4 删除索引.....	72
11.5 查看索引信息.....	73
第 12 章 PL/SQL	74
12.1 PL/SQL 概述.....	74
12.1.1 PL/SQL 优点.....	74
12.1.2 PL/SQL 语句块.....	74
12.2 PL/SQL 变量.....	75
12.3 PL/SQL 数据类型和操作符.....	76
12.3.1 SQL 数据类型.....	76
12.3.2 %TYPE 和%ROWTYPE.....	77
12.3.3 记录类型.....	78
12.3.4 数组类型.....	79
12.3.5 操作符.....	80
12.4 PL/SQL 流控制.....	81
12.4.1 IF 语句.....	81
12.4.2 LOOP 语句.....	82
12.4.3 WHILE 语句.....	82
12.4.4 FOR 语句.....	83
12.4.5 EXIT 语句.....	83
12.4.6 GOTO 语句.....	83
12.4.7 NULL 语句.....	84
12.5 PL/SQL 访问数据库.....	84
12.5.1 数据查询.....	84
12.5.2 数据操纵.....	85
12.5.3 数据定义.....	86
12.6 PL/SQL 与数据库交互.....	86
12.6.1 存储过程.....	86
12.6.2 存储函数.....	87
12.6.3 游标.....	88
12.6.4 引用游标.....	90
12.6.5 动态 SQL.....	91
12.7 异常处理.....	93
12.7.1 异常处理程序.....	93
12.7.2 预定义的异常.....	94

12.7.3 用户自定义的异常	95
12.8 PL/SQL 调试工具	96
12.8.1 命令行参数	96
12.8.2 功能描述	97
12.9 小结	103
第 13 章 管理触发器	104
13.1 触发器的使用	104
13.2 语句级触发器	106
13.3 行触发器	106
13.4 事件触发器	108
13.5 时间触发器	108
13.6 触发器总结	109
第 14 章 管理视图、序列和同义词	110
14.1 管理视图	110
14.2 管理序列	110
14.3 管理同义词	111
14.4 查看视图、序列和同义词信息	111
第 15 章 模式对象的常规管理	112
15.1 在单个操作中创建多个模式对象	112
15.2 重命名模式对象	113
15.3 启用和停用触发器	113
15.4 管理完整性约束	114
15.5 管理对象依赖性	115
15.6 管理对象名称解析	115
15.7 显示有关模式对象的信息	115
第 16 章 Disql	117
16.1 启动 disql	117
16.1.1 在 windows 环境中运行 disql	117
16.1.2 在命令运行 disql	117
16.2 设置和显示 disql 环境变量	119
16.2.1 SET	119
16.2.2 SHOW	120
16.3 常用的 disql 命令	121
16.3.1 help	121
16.3.2 spool	121
16.3.3 host	121
16.3.4 describe	121
16.3.5 定义本地变量	122
16.3.6 EDIT	123
16.3.7 explain	123
第三部分 高级数据库管理	124
第 17 章 数据库布局和存储管理	124

17.1 管理表空间	124
17.1.1 创建表空间	124
17.1.2 扩展表空间	124
17.1.3 删除表空间	124
17.1.4 修改表空间名	124
17.1.5 修改表空间状态	125
17.1.6 修改表空间数据缓冲区	125
17.1.7 查询表空间与数据文件对应关系	125
17.2 管理数据文件	125
17.2.1 添加数据文件	125
17.2.2 扩展数据文件的大小	125
17.2.3 指定数据文件的扩展属性	126
17.2.4 修改数据文件的路径	126
17.3 管理重做日志文件	126
17.3.1 添加重做日志文件	126
17.3.2 扩展重做日志文件	126
17.4 管理回滚空间	127
17.5 管理控制文件	127
17.5.1 修改控制文件路径	127
17.5.2 添加控制文件	127
第 18 章 快速数据加载	128
18.1 使用 dmfldr 加载数据	128
18.2 dmfldr 的控制文件	132
18.3 性能调整	133
第 19 章 管理分区表和分区索引	135
19.1 分区概念	135
19.2 分区的方法	135
19.3 创建水平分区表	136
19.3.1 创建范围分区表	136
19.3.2 创建列表分区表	137
19.3.3 创建哈希分区表	138
19.4 在水平分区表建立索引	139
19.5 维护水平分区表	139
19.5.1 增加分区	139
19.5.2 删除分区	140
19.5.3 交换分区	140
19.5.4 合并分区	141
19.5.5 拆分分区	141
19.6 水平分区表的限制	141
19.7 创建垂直分区表	142
19.8 垂直分区表的限制	143
第 20 章 管理列存储表	144
20.1 什么是列存储表	144
20.2 创建列存储表	145
20.3 维护列存储表	146

20.4 查看有关列存储表的信息	147
第 21 章 管理 HFS 表.....	148
21.1 什么是 HFS 表.....	148
21.2 创建 HFS 表.....	150
21.3 维护列存储表	151
21.4 查看有关 HFS 表的信息	152
第 22 章 管理 LIST 表	153
22.1 什么是 LIST 表.....	153
22.2 创建 LIST 表.....	154
22.3 LIST 的限制.....	154
22.4 维护 LIST 表.....	154
22.5 查看有关 LIST 表的信息	155
第 23 章 管理作业.....	156
23.1 作业概述	156
23.1.1 操作员	156
23.1.2 作业	156
23.1.3 警报	156
23.1.4 调度	157
23.2 配置作业管理系统	157
23.3 创建删除修改操作员	166
23.3.1 创建操作员	166
23.3.2 删除操作员	166
23.3.3 修改操作员	166
23.4 创建修改删除作业	167
23.4.1 创建作业	167
23.4.2 修改作业	167
23.4.3 删除作业	168
23.5 配置作业.....	168
23.5.1 作业配置开始	168
23.5.2 增加步骤	169
23.5.3 删除步骤	170
23.5.4 增加调度	170
23.5.5 删除调度	172
23.5.6 配置提交	172
23.6 创建删除修改警报	173
23.6.1 创建警报	173
23.6.2 修改警报	174
23.6.3 删除警报	174
23.7 为警报关联操作员	175
23.7.1 关联	175
23.7.2 取消	175
23.8 综合例子	175
第 24 章 全文检索	177
24.1 全文检索概述	177

24.2 创建全文索引	178
24.3 更新全文索引	178
24.4 执行全文检索	179
24.5 删除全文索引	180
第 25 章 管理事务	181
25.1 事务简介	181
25.2 事务特性	182
25.2.1 原子性	182
25.2.2 一致性	182
25.2.3 隔离性	182
25.2.4 持久性	182
25.3 提交事务	183
25.3.1 自动提交模式	183
25.3.2 手动提交模式	183
25.3.3 隐式提交模式	183
25.4 回滚事务	184
25.4.1 自动回滚	184
25.4.2 手动回滚	184
25.4.3 回滚到保存点	184
25.4.4 语句级回滚	185
25.5 事务锁定	185
25.5.1 锁模式	185
25.5.2 锁粒度	186
25.5.3 查看锁	187
25.6 多版本	188
25.6.1 物理记录格式	188
25.6.2 回滚记录格式	188
25.6.3 可见性原则	188
25.6.4 历史数据获取	189
25.6.5 回滚段自动清理	189
25.7 事务隔离级	189
25.7.1 读一致性	190
25.7.2 读提交隔离级	191
25.7.3 串行化隔离级	192
25.7.4 读未提交隔离级	193
25.7.5 只读事务	193
25.8 锁等待与死锁检测	193
25.9 闪回	194
第 26 章 管理安全性	195
26.1 概述	195
26.2 用户管理	195
26.2.1 DM 中有哪些管理用户	195
26.2.2 如何创建用户	196
26.2.3 如何修改用户信息	197
26.2.4 如何删除用户	197
26.3 权限管理	198

26.3.1 数据库权限的管理	198
26.3.2 对象权限的管理	199
26.4 角色管理	201
26.4.1 角色的创建和删除	201
26.4.2 角色中权限的添加和删除	202
26.4.3 角色的分配和回收	203
26.4.4 角色的启用和禁用	203
26.5 标记管理	203
26.5.1 如何创建策略	203
26.5.2 如何创建标记	204
26.5.3 如何对表应用策略	205
26.5.4 如何对用户应用策略	206
26.5.5 读写控制规则	207
26.5.6 扩展客体应用标记	209
26.6 数据库审计	211
26.6.1 DM 数据库审计员的创建和删除	211
26.6.2 DM 审计设置与取消	211
26.6.3 DM 审计文件管理	216
26.6.4 DM 审计信息查阅	217
26.6.5 DM 审计实时侵害检测	217
26.6.6 DM 审计分析	219
26.7 通信加密	219
26.8 存储加密	219
26.9 资源限制	221
26.10 客体重用	223
26.11 加密引擎	223
第四部分 故障排除和性能优化	225
第 27 章 问题跟踪和解决	225
27.1 问题分析	225
27.2 监控系统性能(V\$)	226
27.3 数据库重演(REPLAY)	226
27.4 检查数据物理一致性	227
27.5 调整配置参数	227
27.6 优化数据库布局	228
第 28 章 动态管理/性能视图	229
28.1 理解动态管理视图	229
28.2 使用动态管理视图	229
第 29 章 查询优化	233
29.1 优化目标	233
29.2 查询优化器	233
29.2.1 查询转换	233
29.2.2 估算代价	233
29.2.3 生成计划	234
29.3 数据访问路径	234

29.4 连接.....	235
29.5 统计信息.....	236
29.6 执行计划.....	236
29.7 使用索引.....	237
29.8 并行查询.....	237
29.8.1 并行查询概念.....	237
29.8.2 确定并行任务数.....	238
29.8.3 确定并行工作线程数.....	239
29.8.4 执行查询.....	239
29.8.5 使用场景.....	240
29.9 查询计划重用.....	240
29.10 结果集重用.....	240
第 30 章 SQL 调优.....	241
30.1 简介.....	241
30.2 调优目标.....	241
30.3 确定高负载的 SQL.....	242
30.4 自动 SQL 调整.....	242
30.5 开发有效的 SQL 语句.....	242
30.6 使用优化器提示.....	244
第五部分 数据高可用性.....	245
第 31 章 故障恢复.....	245
31.1 概述.....	245
31.2 REDO 日志.....	245
31.3 重做日志归档.....	245
31.3.1 本地归档.....	246
31.3.2 实时归档.....	246
31.3.3 同步归档.....	246
31.3.4 异步归档.....	246
31.4 检查点.....	246
31.5 回滚段与回滚记录.....	247
31.6 系统故障恢复.....	247
31.6.1 恢复流程.....	247
31.6.2 并行恢复.....	247
31.7 介质故障恢复.....	248
第 32 章 备份与还原.....	249
32.1 概述.....	249
32.2 备份.....	249
32.2.1 完全备份与增量备份.....	249
32.2.2 联机备份与脱机备份.....	250
32.2.3 物理备份.....	250
32.2.4 逻辑备份.....	252
32.2.5 B 树备份.....	254
32.2.6 备份加密与压缩.....	255
32.2.7 并行备份.....	256

32.3 还原.....	257
32.3.1 联机还原与脱机还原.....	257
32.3.2 还原数据库	257
32.3.3 还原表空间	258
32.3.4 还原用户表	258
32.3.5 逻辑还原	259
32.3.6 使用归档日志还原	261
32.3.7 还原到指定时间点	261
32.3.8 并行还原	261
第 33 章 数据守护	262
33.1 概述.....	262
33.2 基本概念	263
33.2.1 基于观察器的数据守护.....	264
33.2.2 基于守护进程的数据守护.....	264
33.3 体系结构.....	266
33.3.1 基于观察器的数据守护.....	266
33.3.2 基于守护进程的数据守护.....	267
33.4 基于观察器的数据守护	268
33.4.1 配置数据守护	270
33.4.2 启动数据守护	272
33.4.3 查看系统状态	273
33.4.4 故障自动切换	273
33.4.5 手动切换主备机	273
33.4.6 增加/删除备机	273
33.4.7 停止观察器检测	274
33.4.8 特性	274
33.4.9 故障处理	274
33.5 基于守护进程的数据守护	274
33.5.1 配置数据守护	275
33.5.2 故障自动切换和恢复.....	278
33.5.3 手动切换	279
33.5.4 查询守护进程状态	279
33.5.5 同时监控多个实例	279
33.5.6 停止对实例的监控	279
33.5.7 重启对实例的监控	279
33.5.8 网络故障检测	280
33.5.9 从命令行运行守护进程.....	280
33.5.10 守护进程提供的命令.....	281
33.5.11 从命令行运行 dmwmon.....	281
33.5.12 监视器 DMWMON 命令	282
第 34 章 数据复制.....	283
34.1 概述.....	283
34.2 重要概念	283
34.3 体系构架.....	284
34.4 配置数据复制	285
34.5 监控数据复制	287

34.5.1 复制故障监控	287
34.5.2 复制故障处理	288
34.6 初始化 RPS 脚本	289
第 35 章 大规模并行处理 MPP	292
35.1 概述	292
35.2 基本概念	292
35.3 体系架构	292
35.4 支持的功能	294
35.4.1 支持多种分布	294
35.4.2 函数和过程	294
35.4.3 支持触发器	294
35.4.4 支持分区表	294
35.4.5 自增列和序列	294
35.4.6 支持大字段	294
35.4.7 支持列存储	294
35.4.8 支持 HFS	295
35.5 数据加载	295
35.6 环境配置	295
35.6.1 普通配置	295
35.6.2 基于主备的配置	296
35.6.3 监控器使用说明	301
第六部分 附录	304
附录 1 数据字典	304
附录 2 动态性能视图	314
附录 3 数据复制的 INI 文件及系统表	339
附录 4 预设角色权限列表	344
附录 5 DM 技术支持	348

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。但是在即将提供的 DM 共享磁盘高性能集群中，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、簇和块（也称为页）。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。

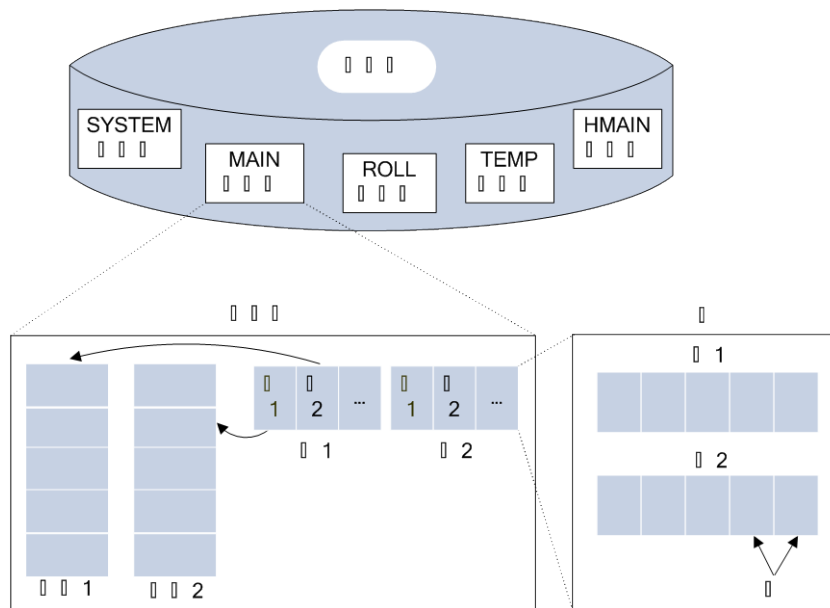


图 1.1 显示表空间、数据文件、段、簇、块的关系

可以看出，在 DM7 中存储的层次结构如下：

1. 系统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. 簇由磁盘上连续的块组成，一个簇总是在一个数据文件中；
6. 块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，同时还可以用来存放用户数据（这需要用户指定，不建议这么做），用户也可以手工创建其他表空间来存放用户数据。
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 HFS 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 SYSSO、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，而不是将数据存放在 **SYSTEM** 表空间。

用户可以通过执行如下语句来查看表空间相关信息：

```
SELECT * FROM V$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 16 章。

1.2.2 块

数据库块是 **DM** 数据库中最小的数据存储单元。块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，块大小可以为 **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，块大小都不能够改变。图 1.2 显示了 **DM** 数据库块的典型格式。

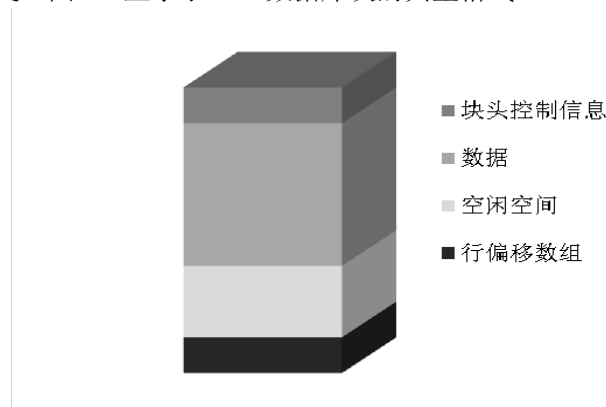


图 1.2 DM 数据库块的组成

块头控制信息包含了关于块类型、块地址等信息。块的中部存放数据，为了更好地利用数据块，在数据块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块上的空间占用情况以便管理数据块自身的空间。

在绝大多数情况下，用户都无需干预 **DM** 数据库对数据块的管理。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块级存储参数，它指定一个数据块初始化后插入数据时最大可以使用空间的百分比，该值在创建表/索引时可以指定。设置 **FILLFACTOR** 参数的值，是为了指定数据块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块保留一定的空间，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，引起数据块的频繁分裂。当插入的数据占据的数据块空间百分比低于 **FILLFACTOR** 时，允许数据插入该块，否则将当前数据块中的数据分为两部分，一部分保留在当前数据块中，另一部分存入一个新块中。

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，频繁引起块分裂，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

数据块的分裂，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。

1.2.3 簇

簇是数据块的上级逻辑单元，由同一个数据文件中 16 个或 32 个连续的数据块组成。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，块大小为 8KB，则共有 $32\text{MB}/8\text{KB}/16=256$ 个簇，每个簇的大小为 $8\text{K}*16=128\text{K}$ 。和数据块的大小一样，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块，供后续操作使用。如果初始分配的簇中所有数据块都已经用完，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。

1.2.4 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块则只能来自一个文件，是连续的 16 或者 32 个数据块。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块镜像，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。

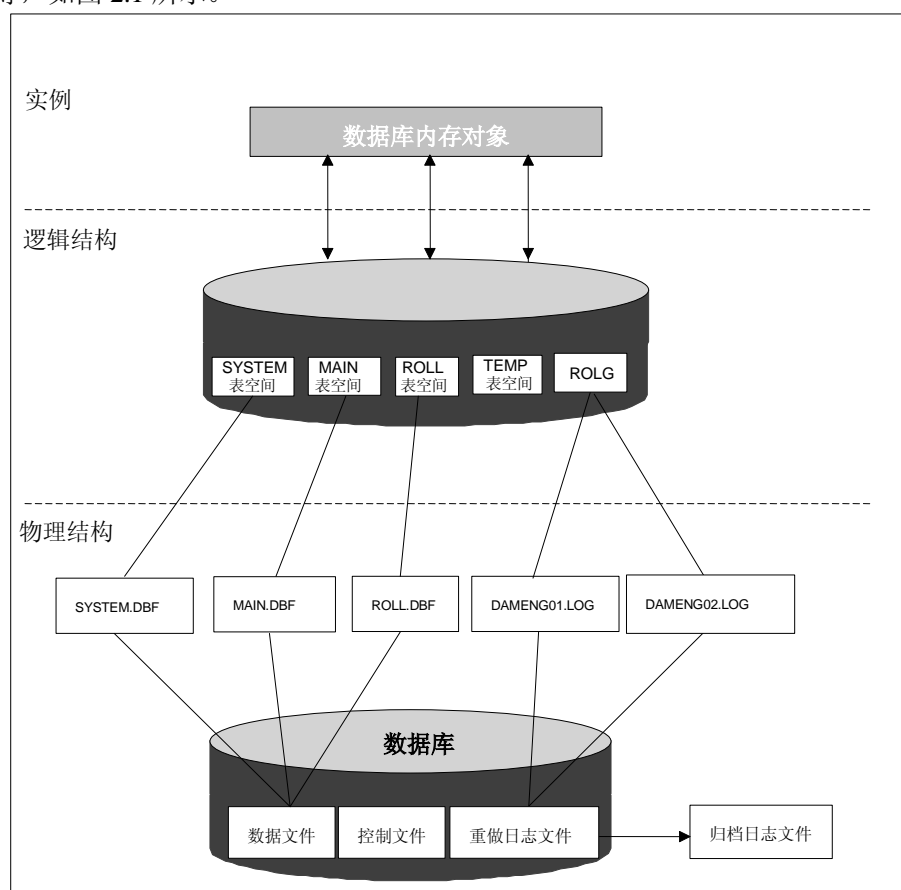


图 2.1 DM 物理存储结构示意图

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

2.1.1.1 dm.ini

每创建一个 DM 数据库，就会自动生成 dm.ini。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。

表中参数属性的“静态”指修改后重启服务器才能生效；“动态”指修改后即时生效；“手动”表示服务器运行过程中不可修改。动态参数又分为会话级和系统级两个级别，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；系统级参数的修改则会影响所有的会话。

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
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
实例名			
INSTANCE_NAME	DMSERVER	手动	实例名
内存相关参数			
MEMORY_POOL	40	静态	系统内存池大小，以兆为单位。系统内存池是由 DM 管理的内存，系统内存的申请都从内存池中申请。有效值范围（5~10000）
BUFFER	80	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）
HUGE_BUFFER	8	静态	HFS 表使用的缓冲区大小，以兆为单位。有效值范围（8~1048576）
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）
MULTI_PAGE_GET_NUM	16	动态，系统级	缓冲区最多一次读取的页面数。有效值范围（1~128）
MAX_BUFFER	700	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	排序缓存区大小，以 M 为单位。有效值范围（1~1024）
BUFFER_POOLS	1	静态	Buffer 系统分区数（1~512）
FAST_POOL_PAGES	0	静态	快速缓冲区页面数（0~99999）
FAST_ROLL_PAGES	0	静态	快速缓冲区回滚页面数（0~99999）
CACHE_POOL_SIZE	100	静态	SQL 缓冲池大小，以兆为单位。有效值范围（0~10240）
HAGR_HASH_SIZE	100000	动态，会	Hagr 操作时，建立 Hash 表的桶个数。有效值范

		话级	围（10000~100000000）
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（0~10000000）
HJ_BUF_SIZE	50	动态，会话级	Hash 连接操作符的数据总缓存大小，以兆为单位。有效值范围。有效值范围（2~100000）
HJ_BUF_GLOBAL_SIZE	500	静态	Hash 连接操作符的数据总缓存大小，系统级参数，以兆为单位。有效值范围(10~500000)
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）
MTAB_MEM_SIZE	8	静态	MTAB 缓存 BDTA 占用内存空间的大小，以 M 为单位（1~1048576）
DICTIONARY_BUF_SIZE	5	静态	字典缓冲区大小，以兆为单位。有效值范围（1~100）
MAIL_BUF_SIZE	100	静态	邮件系统邮件缓存大小，以兆为单位。当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（0~10000）
MAIL_VPOOL_SIZE	10	动态，系统级	邮件系统使用的内存初始化大小，以兆为单位。有效值范围（1~100）
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 M 字节为单位，有效值范围（16~4000）
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（64~1024*1024）
VM_POOL_SIZE	2048	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（64~1024*1024）
线程相关参数			
WORKER_THREADS	4	静态	工作线程的数目，有效值范围（1~1000）
TASK_THREADS	4	静态	任务线程个数（1~1000）
WORKER_CPU_PERCENT	0	手动	工作线程占 CPU 的比重，仅非 windows 下有效（0~100）
UTHR_FLAG	0	手动	用户线程标记，1：启用；0：不启用
MAL_UTHR_FLAG	1	手动	邮件系统是否使用用户态线程，1：使用；0：不使用
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）
FAST_RW_LOCK	1	手动	快速读写锁标记，1 表示启用，0 不启用
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

FIRST_ROWS	100	静态	结果集一个消息中返回的最大行数（1~1000）
RS_CAN_CACHE	0	静态	是否启用结果集缓存，0：不启用；1：启用
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）
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，为 0 时表示不限制。有效值范围（0~4294967294）
SESSION_RESULT_SET_LIMIT	0	动态	会话上结果集个数上限，0 代表不限制（0~4294967294）
RESULT_SET_FOR_QUERY	0	动态，会话级	是否不生成非查询结果集。0：生成；1：不生成
MAX_OPT_N_TABLES	8	动态，会话级	能优化的最大表连接个数。有效值范围（0~100）
CNNTB_MAX_LEVEL	20000	动态，会话级	层次查询的最大支持层次。有效范围（1~100000）
BATCH_PARAM_OPT	1	静态	是否启用批量参数优化，0：禁用；1：启用，默认启用。当置为 1 时，不返回操作影响的行数
PHC_MODE_ENFORCE	0	动态，会话级	允许使用的 join 实现方式，0：所有；1：允许 nest loop；2：允许 index join；4：允许 hash join；8：允许 merge join 可以使用上述值或运算的结果，如值 11=1+2+8 表示可以使用除 hash join 外所有的连接方式。任何情况下 nest loop 始终被优化器考虑。
MAX_PARALLEL_DEGREE	2	动态，会话级	水平分区表并行执行的最大任务数。有效值范围（1~128）
PHG_OPT_LEVEL	1	静态	指定 PLL 操作符上提的级别，一般用于水平分区表并行查询。2：最大程度的提升，允许 PHC 优化；1：最简单的提升如 prjt, slct，允许 PHC 优化；0：不做任何提升，不允许 PHC 优化；
ENABLE_MS_PUSH_DOWN	0	动态，系统级	在多表连接中是否允许强制采用归并连接实现子查询，1：允许；0：不允许
ENABLE_IN_VALUE_LIST_OPT	0	动态，系统级	是否允许用索引连接的方式进行 in 常量列表的过滤，1：允许；0：不允许
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
SESS_PLN_NUM	0	静态	每个会话上最多拥有的临时计划缓存个数，如果大于该个数，则会淘汰掉一部分缓存计划。0 表示不启用，也可以理解为会话上不会临时缓存计划。只有在计划重用参数 USE_PLN_POOL 启用时，该参数才会生效
SEL_RATE_EQU	OLAP_FLAG=1 时，缺省值是 0.2，否则为 0.025	静态	表示等值查询的默认选择率，浮点类型，有效值范围（0~1）。此参数缺省情况下没有在 dm.ini 文件中列出，用户可调用修改 INI 参数的系统过程对此参数进行修改，修改后可在 dm.ini 中看到此参数。
SEL_RATE_SINGLE	OLAP_FLAG=1 时，缺省值	静态	表示单表过滤器的默认选择率，浮点类型，有效值范围（0~1）。此参数缺省情况下没有在

	是 0.38, 否则为 0.05		dm.ini 文件中列出, 用户可调用修改 INI 参数的系统过程对此参数进行修改, 修改后可在 dm.ini 中看到此参数。
SEL_RATE_SIO	OLAP_FLAG=1 时, 缺省值是 1, 否则为 20	静态	表示顺序读的 IO 数量, 有效值范围 (1~100)。此参数缺省情况下没有在 dm.ini 文件中列出, 用户可调用修改 INI 参数的系统过程对此参数进行修改, 修改后可在 dm.ini 中看到此参数。
ENABLE_QUERY_EXP_OPT	0	动态, 系统级	是否允许相关查询表达式优化, 1: 允许; 0: 不允许
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 表示自动配置并行工作线程个数 (与物理 CPU 核数相同); 2 表示手动设置并行工作线程数
PARALLEL_THRD_NUM	10	静态	用来设置并行工作线程个数, 仅当 PARALLEL_POLICY 值为 2 时才启用此参数。有效值范围 (1~1024)
CTL_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)
CTL_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)
检查点相关参数			
CKPT_RLOG_SIZE	100	静态	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	静态	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	1800	静态	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 (0~4294967294)
CKPT_FLUSH_RATE	5	静态	检查点刷新级别。有效值范围 (1~100)
CKPT_FLUSH_PAGES	1000	静态	检查点刷新的最小页数。有效值范围 (1000~100000)
FORCE_FLUSH_PAGES	10	静态	调度线程强制刷页个数。有效值范围 (0~1000)
IO 相关参数			
DIRECT_IO	0	静态	非 windows 下有效, 0: 使用文件系统缓存; 1, 2: 不使用文件系统缓存
IO_THR_GROUPS	2	静态	非 windows 下有效, 表示下 IO 线程组个数。有效值范围 (0~512)
数据库相关参数			
PORT_NUM	5236	静态	服务器通讯端口号, 有效值范围 (1024~65534)
BDTA_SIZE	1000	动态, 会话级	BDTA 缓存的记录数。有效值范围 (1~10000)
BDTA_COMPRESS	0	动态, 会话级	BDTA 数据是否压缩, 0: 不压缩; 1: 压缩
OLAP_FLAG	0	动态, 会话级	启用联机分析处理, 0: 不启用; 1: 启用
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（1~102400）
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）
ENABLE_SPACELIMIT_CHECK	1	静态	是否启用检查 space limit，1：启动；0：不启用
ENABLE_MONITOR	2	静态	启用动态监控功能标记，0：不启用；1：低级别监控；2：高级别监控
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）
CTAB_SEL_WITH_CONS	0	动态，系统级	查询建表时是否复制约束，1：复制；0：不复制
DROP_CASCADE_VIEW	1	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
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	0	动态，会话级	建立主关键字时，是否缺省指定为 CLUSTER，0：不指定；1：指定
SP_REBUILD	0	静态	系统启动时是否重建存储过程，0：表示不重建；1：表示重建
EXPR_N_LEVEL	200	动态，会话级	表达式最大嵌套层数。有效值范围（10~500）
FILES_OPENED	60	静态	最大打开文件数。有效范围（60~100）
TEMP_SIZE	10	静态	临时数据库大小，以兆为单位。有效值范围（10~1048576）
FILE_TRACE	0	静态	日志中是否记录文件操作，0：不记录；1：记录
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。
LIST_TABLE	0	动态，会话级	默认情况下，创建的表是否为 list 表，0：否；1：是
NEW_FILE_POLICY	0	手动	主备环境中：备机上重做日志创建新文件路径策略，0/1 分别表示使用相对路径/绝对路径
ENABLE_PAGE_CHECK	0	静态	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)
IDLE_MEM_THRESHOLD	50	动态，系统级	可用物理内存的报警阈值，单位是兆（10~6000）
IDLE_DISK_THRESHOLD	1000	动态，系统级	磁盘可用空间的报警阈值，单位是兆（50~50000）
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
IDLE_SESS_THRESHOLD	5	动态，系统级	有限的会话数阈值（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

LOAD_TABLE	空串	静态	在服务器启动时预先装载的表的完整表名，即“模式名.表名”，多个表之间用逗号分隔，最多可指定 10 个表
日志相关参数			
LOG_BUF_SIZE	256	静态	日志缓冲区大小（以日志页个数为单位）。有效值范围（1~4294967294）
LOG_REDO_THREAD_NUM	2	静态	日志线程数（1~10）
ENABLE_LOG_PARALLELISM	0	静态	是否启用并行写 RLOG，0：不启用；1：启用
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_AUTO_COMMIT	1	动态，会话级	指定 DDL 语句是否自动提交，1：自动；0：手动
DDL_FORBIDDEN	0	动态，系统级	1 表示不允许 DDL 操作，0 代表允许
DISABLE_TABLE_LOCK	0	动态，系统级	1 表示只允许对表上意向锁（0~1）
FAST_COMMIT	0	静态	批量提交事务的个数（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。
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
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）
SERIALIZABLE_CHECK_INTERVAL	300	动态，系统级	当串行化事务在指定时间内没有提交，则强制断开连接，以秒为单位。有效值范围（60~3600）
READ_COMMITTED_CHECK_INTERVAL	900	动态，系统级	读提交事务且生成了回滚记录情况下，强制断开的检测时间。有效值范围（60~3600）
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
LOCK_TIMEOUT	10	动态，系统级	在 UTHR_FLAG=1 的模式下，锁等待时间，以秒为单位，有效范围（1~4294967294）
UNDO_EXTENT_NUM	16	手动	在回滚段为每个工作线程预留的 extent 个数，有效范围（1~2147483647）
ENABLE_FLASHBACK	0	动态，系统级	是否启用闪回查询，0：不启用；1：启用
FLASHBACK_RETENTION	900	静态	启用闪回查询时，指定可闪回的时间长度，单位为秒。有效值范围（1~86400）
MAX_DE_TIMEOUT	10	动态，会话级	C 外部函数的执行超时时间，以秒为单位。有效值范围（1~3600）
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 6；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
ENABLE_ENCRYPT	0	静态	通讯所采用的加密类型。0: 不加密; 1: SSL 加密
ENABLE_AUDIT	0	静态	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)
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	1	静态	是否加载压缩动态库, 0: 不加载; 1: 加载
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SF_LABEL_FROM_CHAR函数的使用范围。 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
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
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。
跟踪监控相关参数			
SQL_TRACE_MASK	1	手动	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。每一位的含意见下面说明: 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 表对象操作(TABLE DDL) 14 视图对象操作(VIEW DDL) 15 过程或函数对象操作(PROCEDURE DDL) 16 触发器对象操作(TRIGGER DDL) 17 序列对象操作(SEQUEN DDL) 18 模式对象操作(SCHEMA DDL) 19 库对象操作(DATABAE DDL) 20 用户对像操作(USER DDL) 21 索引对象操作(INDEX 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)
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 (0~4294967294)
SVR_LOG	0	动态, 系统级	是否打开 SQL 日志功能, 0: 表示关闭; 1: 表示日志文件非切换模式, 但输出的日志格式是详细模式; 2: 表示日志文件为切换模式, 输出的日志也是详细模式; 3: 表示日志为非切换模式, 但输出日志为简单模式
SVR_LOG_SWITCH_COUNT	500	静态	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 (0~500)
数据守护相关参数			
DW_UDP_PORT	0	静态	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 (1025~65534), 为 0 时表示没有配置守护进程
INST_UDP_PORT	0	静态	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 时表示没有配置守护

			进程
DW_MAX_SVR_WAIT_TIME	0	静态	服务器等待守护进程启动的最大时间（秒）。当此参数的值不为 0 时，实例会检测守护进程是否启动，如果守护进程没有启动的时间超过设置的值，则实例自动退出。当参数的值等于 0 时，代表服务器不检测守护进程有没有启动
Oracle 数据类型兼容			
ORACLE_TYPE_COMPATIBLE	0	动态，会话级	是否启用 oracle 数据类型兼容，0：不启用；1：启用
NUMBER_LEN_DEFAULT	22	动态，会话级	Number 类型的默认长度，有效值范围（1~38），只有当 ORACLE_TYPE_COMPATIBLE 设置为 1 时有效
NUMBER_PREC_DEFAULT	6	动态，会话级	Number 类型的默认精度，有效值范围（0~38），只有当 ORACLE_TYPE_COMPATIBLE 设置为 1 时有效
其他相关参数			
MAIL_INI	0	静态	是否启用 mail 系统，0：不启用；1：启用
ARCH_INI	0	静态	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；2：全局逻辑日志
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

2.1.1.2 dm_svc.conf

DM 安装时生成一个配置文件 dm_svc.conf，在 Windows 操作平台下此文件位于 %SystemRoot%\system32 目录，在 Linux 平台下此文件位于 /etc 目录。

dm_svc.conf 生成时文件中包含一个配置项：

```
TIME_ZONE=(time_zone)
```

TIME_ZONE 配置项用于指明客户端的默认时区。

dm_svc.conf 还支持另一个配置项 SERVERNAME，其格式与说明如下：

```
SERVERNAME=(ip[:port],ip[:port],.....)
```

说明：

1. SERVERNAME：配置的连接服务名，连接时使用
2. ip：服务器所在的 ip 地址，如果 ip 地址为 IPv6 地址，则 ip 地址需要用[]封闭，以示与端口的区别
3. port：服务器使用的连接端口，此项为可选项，不配置则使用连接上指定的端口

此配置项需要用户手工配置。用户使用 DM 的 JDBC、DPI 等接口连接服务器时，如使用配置的 SERVERNAME 进行连接，客户端会随机选择一台配置项中 ip 代表的服务器进行连接。如果连接不成功或者服务器状态不正确，客户端将顺序获取下一台（将配置的所有 ip 视为一个环）服务器进行连接，直至连接成功或者全部失败。

2.1.2 数据守护配置

2.1.2.1 dmmal.ini

dmmal.ini 是 MAL 系统的配置文件，具体内容请见第 31 章“数据守护”。dmmal.ini 的配置项见表 2.2。

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 10s，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的时间，默认 10s，范围（2-1800）	无	无
[mal_inst1]	实例配置	mal_inst_name	实例名
		mal_host	实例主机
		mal_port	监听端口

2.1.2.2 dmarch.ini

dmarch.ini 用于本地归档和远程归档，具体内容请见第 31 章“数据守护”。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
[ARCHIVE_LOCAL1]	归档配置	ARCH_TYPE	归档类型
		ARCH_DEST	归档路径
		ARCH_FILE_SIZE	归档文件大小，单位 MB
		ARCH_SPACE_LIMIT	归档文件限制，单位 MB
ARCHIVE_REALTIME	实时归档配置	ARCH_TYPE	归档类型
		ARCH_DEST	归档目标
ARCHIVE_SYNC	同步归档	ARCH_TYPE	归档类型
		ARCH_DEST	归档目标
ARCHIVE_ASYNC	异步归档	ARCH_TYPE	归档类型
		ARCH_DEST	归档目标
		ARCH_TIMER_NAME	定时器名称

相关说明

- 归档类型 ARCH_TYPE 有以下几种：

本地	LOCAL
远程实时	REALTIME (只有一个)
远程同步	SYNC
远程异步	ASYNC
- 配置名[ARCHIVE_*]表示归档名，在配置文件中必须唯一。其中归档类型为 REALTIME，SYNC 只能有一个，其归档名必须为 ARCHIVE_REALTIME 和

- ARCHIVE_SYNC。其他类型的归档可以多个；
- 不能存在相同实例名的不同归档；
 - 不能存在 DEST 相同的不同归档实例；
 - ARCH_TIMER_NAME 为定制的定时器名称，定时器配置见 dmtimer.ini；
 - ARCH_FILE_SIZE 必须是整型，仅对本地归档有效，表示归档文件大小，单位是 M，值域为 64M ~ 2048M；
 - ARCH_SPACE_LIMIT 必须是整型，仅对本地归档有效，表示归档文件的磁盘空间限制，单位是 MB，值域为 1024~4294967294 或 0，0 表示无空间限制。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。

2.1.2.3 dmwatch.ini

dmwatch.ini 用于配置守护进程，具体内容请见第 31 章“数据守护”。dmwatch.ini 的配置项见表 2.4。

表 2.4 dmwatch.ini 的配置项

项目	项目意义	字段	字段意义
[GROUP_A]	组配置	OGUID	全局唯一ID，有效值范围0-4294967294
		DW_UDP_PORT	守护进程 UDP 端口，和 dm.ini 中的同名配置项保持一致
		SWITCH_MODE	切换模式
		INST_NAME	实例名
		INST_UDP_PORT	DM 实例的 UDP 端口，和 dm.ini 中的同名配置项保持一致
		BRO_ADDR	广播地址（见广播地址计算方法）
		START_TYPE	启动方式： 0：服务方式启动 1：控制台方式启动
		PATH	用于启动实例的文件路径
		SERVICE_IP	实例对外提供服务的 IP 地址。守护进程会自动检测该 IP 能否正常对外提供服务（网卡是否故障或者是网线断开等原因造成的网络故障），如果出现网络故障，则主备机自动进行切换，保证服务不中断。如果不配置此参数，则不会检测网络状况。
		BRO_ERROR_INTERVAL	守护进程检测到本地实例故障后，向远程守护进程广播故障信息的间隔时间，单位(ms)，范围 100-1000，默认 1000ms
		FAILOVER_TIMES	守护进程收到远程守护进程广播的实例故障次数 N，超过 N 次后才进行切换，默认 3。
		ENHANCED_PORT	增强模式下必须配置，其他模式不用配置。守护进程监听链路连接请求的端口。目的是在时间段内没有收到远程守护进程的 UDP 广播后，尝试进行 TCP 连接，检查网络是否正常。两个守护进程端口配置相同。
		ENHANCED_FAR_IP	增强模式下必须配置，其他模式不用配置。远程守护进程 IP 地址，和 ENHANCED_PORT 配合使用。

局域网内广播地址的计算方法：把主机 IP 地址和子网掩码都换算成二进制数，两者进行逻辑与运算后得到网络地址，根据子网掩码中主机地址全部为 0 的部分，在网络地址对应位中全部置成 1，即得到广播地址。

举例：某主机的 IP 是 15.16.193.6，子网掩码是 255.255.248.0，求广播地址。

IP 地址二进制：00001111.00010000.11000001.00000110

子网掩码的二进制：11111111.11111111.11110000.00000000

相与后得到网络地址：00001111.00010000.11000000.00000000

将掩码中为 0 的位在网络地址中置成 1，得到广播地址：
00001111.00010000.11001111.11111111，换算成十进制是 15.16.199.255。

2.1.2.4 dmwmon.ini

dmwmon.ini 用于配置监视器，具体内容请见第 33 章“数据守护”。dmwmon.ini 的配置项见表 2.5。

表 2.5 dmwmon.ini 的配置项

项目	项目意义	字段	字段意义
[GROUP_A]	组配置	MON_OGUID	全局唯一ID，要与dmwatch.ini中的OGUID一致，有效值范围0-4294967294
		MON_UDP_PORT	监视的守护进程组的 UDP 端口，要和 DW_UDP_PORT 保持一致
		GROUP_NAME	监控组名，要与 dmwatch.ini 中的监控组保持一致
		MON_BRO_ADDR	广播地址

2.1.3 复制配置

2.1.3.1 dmrep.ini

dmrep.ini 用于配置复制实例，具体内容请见第 32 章“数据复制”。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	REP_ID	映射所属的复制的 id
		SRC_TAB_ID	复制源对象表的 id

	射	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

2.1.3.2 dmllog.ini

dmllog.ini 用于配置逻辑日志, 具体内容请见第 32 章“数据复制”。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
		TAB_ID	逻辑日志记录的表 id

2.1.3.3 dmtimer.ini

dmtimer.ini 用于配置定时器, 具体内容请见第 32 章“数据复制”。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	TIMER_NAME	定时器名
		TYPE	定时器类型
		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
		START_TIME	开始时间
		END_TIME	结束时间
		DURING_START_DATE	开始日期
		DURING_END_DATE	结束日期
		NO_END_DATE_FLAG	是否结束标记
		DESCRIBE	定时器描述
		IS_VALID	有效标记

2.1.4 审计配置

2.1.4.1 dmamon.ini

dmamon.ini 用于配置安全审计实时侵害检测，dmamon.ini 不能手动配置，只能通过工具 dmamon_ctl 来生成和修改。具体内容请见第 24 章“数据库审计”。dmamon.ini 的配置项见表 2.9。

表 2.9 dmamon.ini 的配置项

项目	项目意义	字段	字段意义
[PATH]	实时审计警报记录文件所在的文件目录	PATH	实时审计警报记录文件所在的文件目录
[SENDER]	发件人	EMAIL	发件人邮箱
		SMTP	发件人邮箱 SMTP 服务器
		USERNAME	发件人用户名
		PASSWORD	发件人密码
[RECEIVER]	收件人	EMAIL	收件人邮箱
[INTERVAL]	时间间隔	INTERVL	检查审计警报记录文件的时间间隔，单位秒

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. 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；
7. 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。

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块、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. LIST 表数据

List 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，List 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

重做日志文件以 log 为扩展名。无论何时，在 DM 数据库中添加、删除、修改对象，或

者改变数据，都会将重做日志写入到当前的重做日志文件。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。

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块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。

日志文件分为联机日志文件和归档日志文件，以上所说的重做日志文件都指联机日志文件。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块都写入磁盘，以保证当前日志所“保护”的数据块都已安全写入磁盘，这样日志文件即可被安全重用。

2.5 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。DM 数据库可以在归档模式和非归档模式下运行，当在非归档模式下运行时，DM 数据库在重用联机日志文件时不生成归档日志文件。

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，数据守护中的备机就是通过重做日志来完成与主机的数据同步的。

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块，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。详细内容请看第 32 章“数据复制”。

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

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，以实例名和生成日期命名，生成在 DM 安装目录的 log 子目录下面，其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。

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。

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_前缀加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。

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包括内存池、缓冲区、排序区、哈希区。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 的内存池指的是共享内存池，根据内存使用情况的不同，对共享内存池的使用采用了两种工作方式：HEAP 和 VPOOL。

共享内存池用于解决 DM Server 对于小片内存的申请与释放问题。系统在运行过程中，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，则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。当系统采用较好的策略管理共享内存池时，小片内存的申请与释放不会对系统影响太大。这种方式还有一个优点，可以比较容易地检测系统是否存在内存泄漏。

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 40M。而 HEAP 与 VPOOL 使用的是共享内存池中的内存，所以一般情况下，HEAP 与 VPOOL 两种方式对内存申请的大小不会超过 MEMORY_POOL 值。

3.1.1 HEAP

HEAP 工作方式采用了堆的思想。每次申请内存时，都是从堆顶上申请，如果不够，则继续向共享内存池中申请内存块，然后加入到 HEAP 中，继续供系统的申请使用，这样 HEAP 的长度可以无限增长下去；释放 HEAP 时，可以释放堆顶上的内存块，也可以释放整个 HEAP。使用内存堆来管理小片内存的申请有个特点，每次申请小片内存以后，不能单独对这片内存进行释放，也就是不用关心这片内存何时释放，它在堆释放时统一释放，能有效防止内存泄露的发生。

3.1.2 VPOOL

VPOOL 的工作方式主要是采用了“伙伴系统”的思想进行管理。申请的 VPOOL 内存分为私有和公有两种。私有 VPOOL 只提供给某个单独功能模块使用，公有 VPOOL 则提供给那些需要共享同一资源而申请的模块，所以对于公有 VPOOL 需要进行保护，而私有 VPOOL 则不需要。

VPOOL 和 HEAP 的区别在于，VPOOL 申请的出去的每片内存都可以单独的进行释放。

3.2 缓冲区

3.2.1 数据缓冲区

数据缓冲区是 DM Server 在将数据块写入磁盘之前以及从磁盘上读取数据块之后，数据块所存储的地方。这是 DM Server 至关重要的内存区域之一，将其设定得太小，会导致缓冲块命中率低，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块大小进行格式化，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块，一条是“自由”链，用于存放目前尚未使用的内存数据块，一条是“LRU”链，用于存放已被使用的内存数据块(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块。

LRU 链对系统当前使用的块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块，能够较大幅度地保证被淘汰的数据块在最近不会被用到，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）的数据块，将它们一直留在缓冲区中，对系统性能会有好处。对于这部分数据块，数据缓冲区开辟了一个特定的区域用于存放它们，以保证这些块不参与一般的淘汰机制，可以一直留在数据缓冲区中。

3.2.1.1 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 和 FAST_ROLL_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(80MB)、KEEP(8MB)、RECYCLE(64MB)、FAST_POOL_PAGES(0) 和 FAST_ROLL_PAGES(0) 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 缓冲区页面数和 FAST 缓冲区回滚页面数。

3.2.1.2 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，而是和数据块一样，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块完全不一样，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，写重做日志比数据块 IO 优先级更高。

DM Server 提供了参数 LOG_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 256 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象，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 1 时，则启动计划重用；否则禁止计划重用。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 100M。

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。

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阈值时，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。当事务需要的数据块不在缓冲区中时，如果在工作线程中直接对那些数据块进行读写，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. 需要处理的数据块不在缓冲区中，此时需要将相关数据块读入缓冲区；
2. 缓冲区满或系统关闭时，此时需要将部分脏数据块写入磁盘；
3. 检查点到来时，需要将所有脏数据块写入磁盘。

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 将数据块写入磁盘，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，负责的任务有以下是一些：

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，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，比数据块分散 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机。如果配置了本地归档或者远程同步归档，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

日志归档线程包含同步归档线程和异步归档线程，前者负责本地归档和远程同步归档任务，后者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、远程同步归档、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 日志重做线程

为了提高故障恢复效率，DM 在故障恢复时采用了并行机制重做 REDO 日志，日志重做

线程就用于日志的并行恢复。通过 INI 参数 LOG_REDO_THREAD_NUM 可配置重做线程数，默认是 2 个线程。

4.8 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。当服务器作为备机时，每次接收到主机的物理 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机上将日志重做，并生成自己的日志，保持和主机数据的同步或一致，作为主机的一个镜像。备机数据对用户只读，可承担报表、查询等任务，均衡主机的负载。

4.9 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。

4.10 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。

4.11 数据守护相关线程

在配置了数据守护的观察器上，会创建观察器的实时检测线程、同步检测线程，实现主备机之间的故障检测、故障切换以及恢复。

在配置了守护进程的数据守护方案中，数据库实例还会创建 UDP 消息的广播和接收线程，负责实例和守护进程之间的通讯，实现数据守护功能。

4.12 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。

4.13 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。

4.14 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 B “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”的安全机制，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，它们各施其职，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，采用“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，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）

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。

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。

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《DM 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 24 章来制定安全策略、创建系统用户并分配合适的用户权限，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 18-22 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 数据库提供了包括备份与还原、故障恢复、数据守护、数据复制等高可靠性解决方案，DBA 可以参考本书第五部分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本书第 24 章。

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，请参考本书第 24 章。

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. 配置数据库时区，如中国是+8:00 时区；
4. 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”→“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。

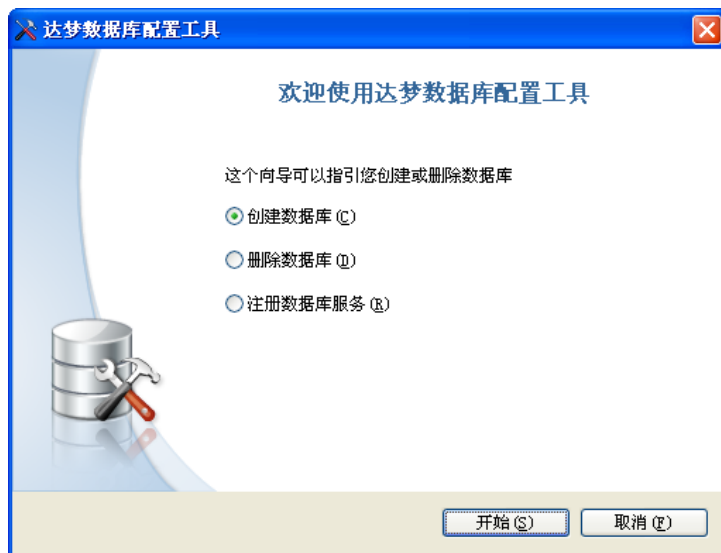


图 7.1 启动数据库配置工具

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. 创建实例库；
8. 创建摘要；
9. 创建数据库。

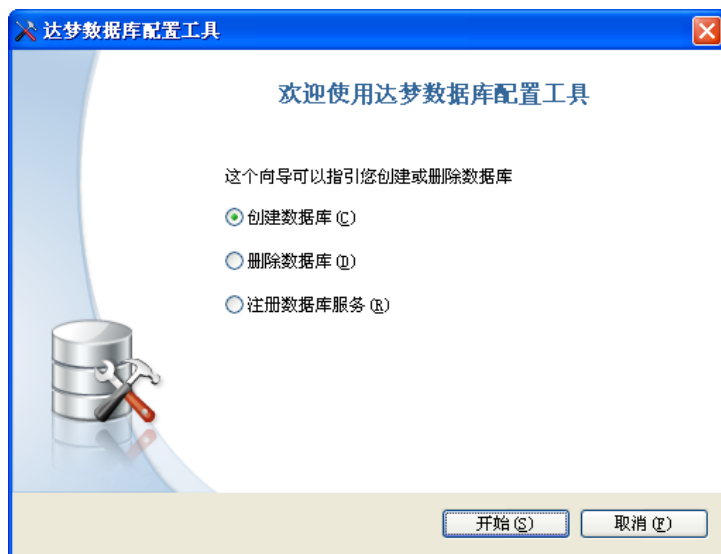


图 7.2 数据库配置工具操作界面

7.2.2.1 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，用户还可以选择是否以配置状态启动数据库和启动数据前是否自定义 INI 文件，如图 7.3 所示。



图 7.3 数据库模板选择

7.2.2.2 数据库目录

指定数据库目录。如图 7.4 所示。

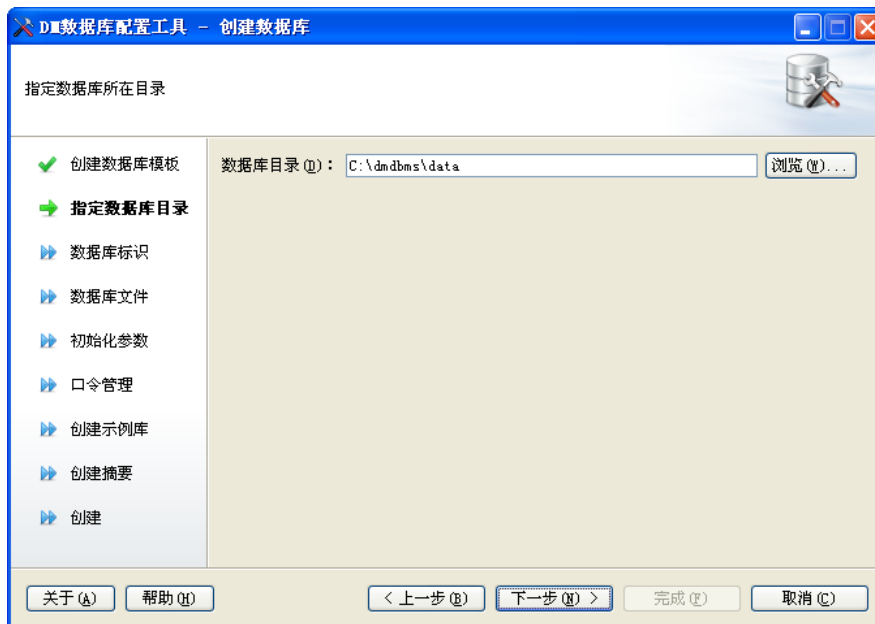


图 7.4 指定数据库目录

7.2.2.3 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

7.2.2.4 数据库文件

如图 7.6 所示，此界面包含三个选项卡：“控制文件”、“数据文件”和“日志文件”，可以通过双击路径来更改文件路径。

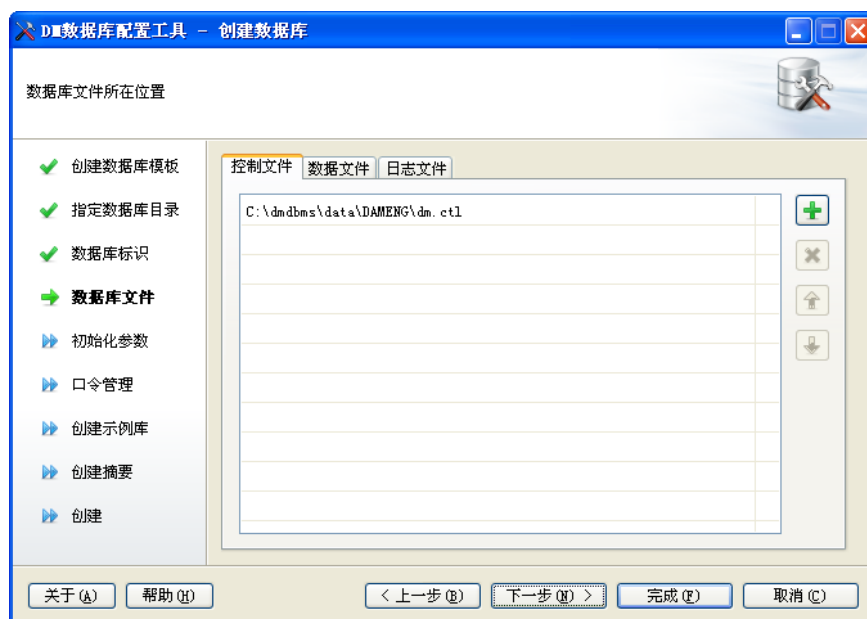


图 7.6 设置数据库文件路径

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、回滚表空间路径和临时表空间路径。

数据文件是数据库中最重要文件类型之一，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，必须理解数据块和簇的概念，它们都是 DM Server 用于保存数据库对象的分配单元。

数据块是系统进行磁盘 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。所有数据块的格式大致相同。

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块组成。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。

7.2.2.5 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。



图 7.7 初始化参数

7.2.2.6 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 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 6 个或多于 48 个字符。如图 7.8 所示。

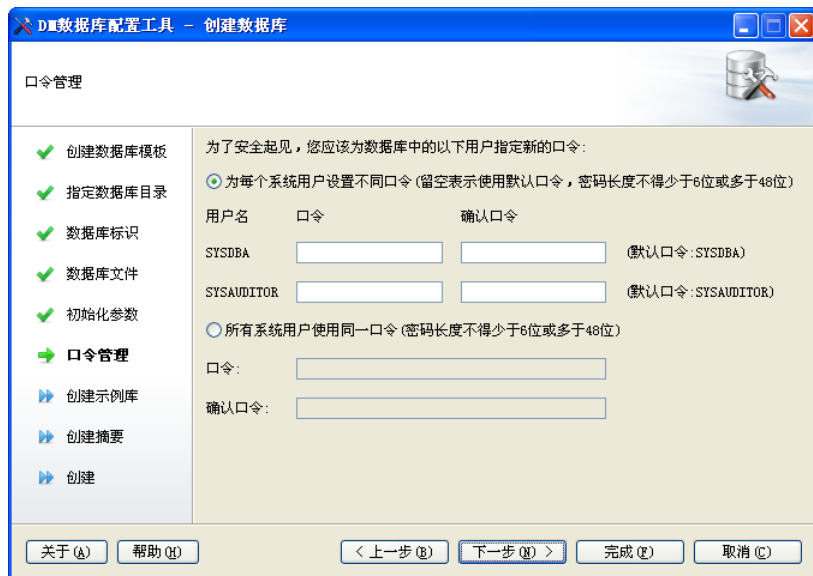


图 7.8 口令管理

7.2.2.7 创建实例库

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。如图 7.9 所示。

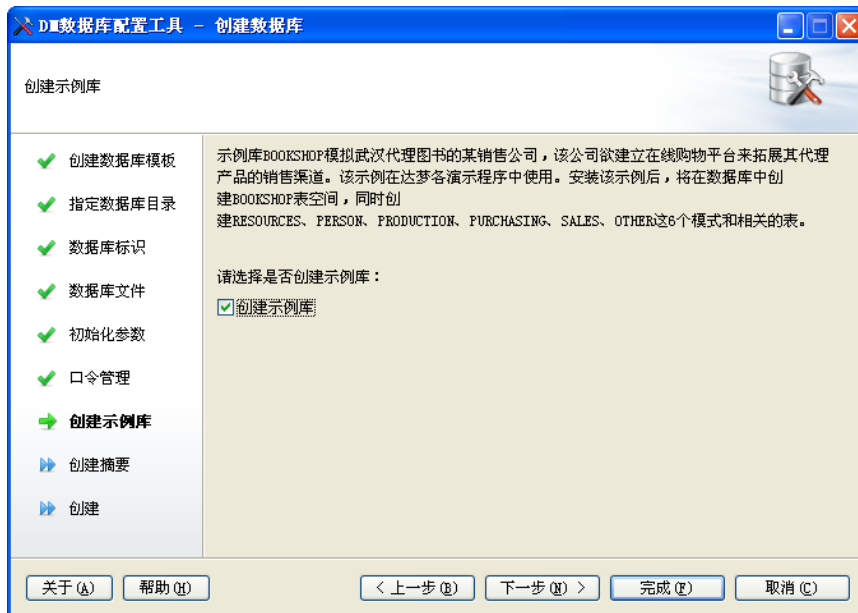


图 7.9 创建实例库

7.2.2.8 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

7.2.2.9 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。

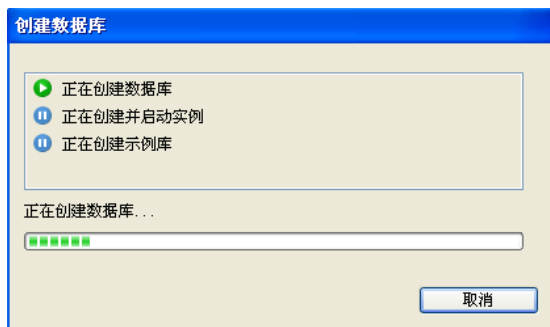


图 7.11 创建数据库

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。

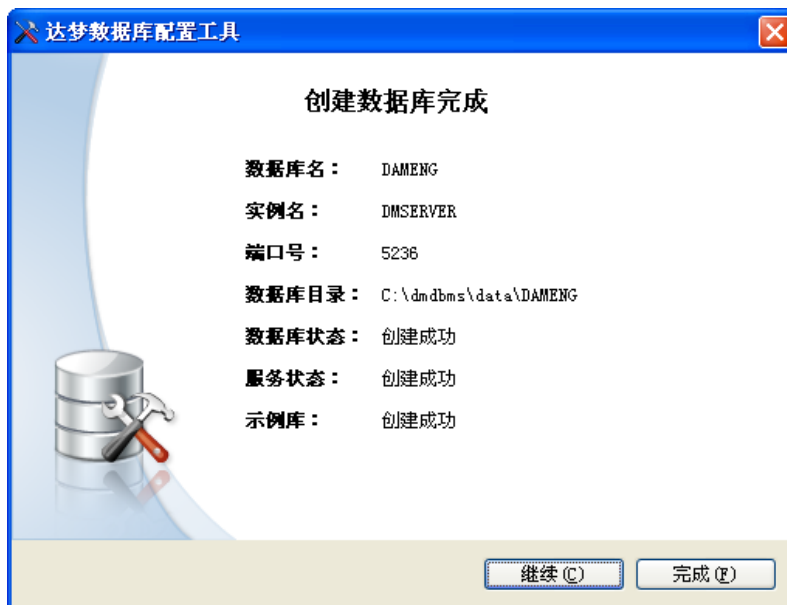


图 7.12 创建数据库完成

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。

7.3.1 从命令行运行 dminit

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	初始化文件 dm.ini 存放的路径	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
PATH	初始数据库存放的路径，默认值 windows 是下 C:\dm7Data，linux 下是 /root/dmData，AIX 下是 /public/dmdb/dmData	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 是下 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 是下 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	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
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）	可选
LOG_SIZE	日志文件使用的簇大小，默认是 512	64 和 2048 之间的整数	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
CASE_SENSITIVE	标识符大小写敏感，默认值为 Y	只能是 'Y', 'y', 'N', 'n', 'I', 'O' 之一	可选
UNICODE_FLAG	是否使用 unicode，默认值为 0（不使用）	只能是 'Y', 'y', 'N', 'n', 'I', 'O' 之一	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
SYSSSO_PWD	初始化时设置 SYSSSO 的密码，默认为 SYSSSO，仅在安全版本下可设置	合法的字符串，长度在 6 到 48 个字符之间	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。

```
initdb U7.0.1.3-Build(2011.07.28)
db version: 0X70001
mem_installed : 2031052
create dm database success.
```

图 7.13 创建成功后的屏幕显示

7.3.2 双击启动 dminit

可以直接双击 dminit.exe 启动该工具，出现如图 7.14 所示的窗口，用户输入参数值后，初始数据库被默认创建到 C 盘根目录下的 dm7Data 文件夹。参数含义见表 7.1。

```
initdb U7.0.1.3-Build(2011.07.28)
db version: 0X70001
input db name: DM
input page size(4, 8, 16, 32): 8
input extent size(16, 32): 16
input time zone: +08:00
string case sensitive? (Y/N): y
use utf8 character set? (Y/N): y
```

图 7.14 dminit 运行窗口

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。如图 7.15 所示。

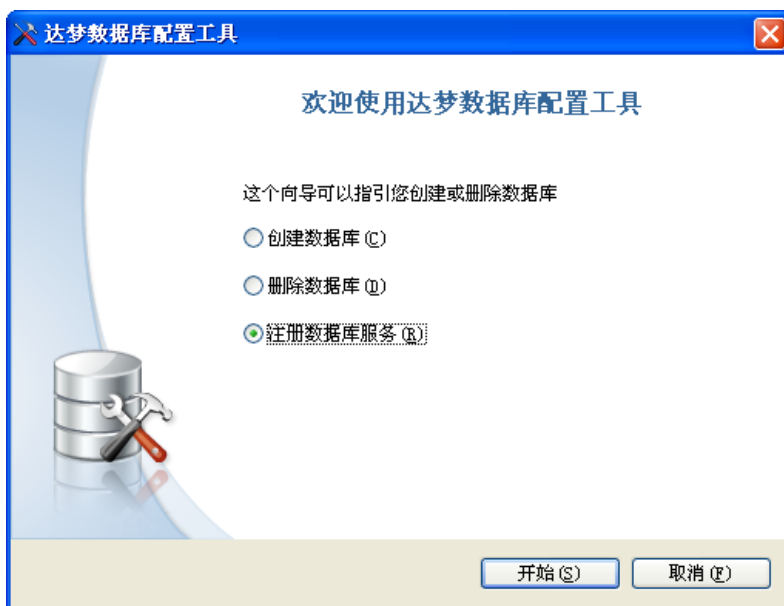


图 7.15 选择操作方式

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 7.16 所示。

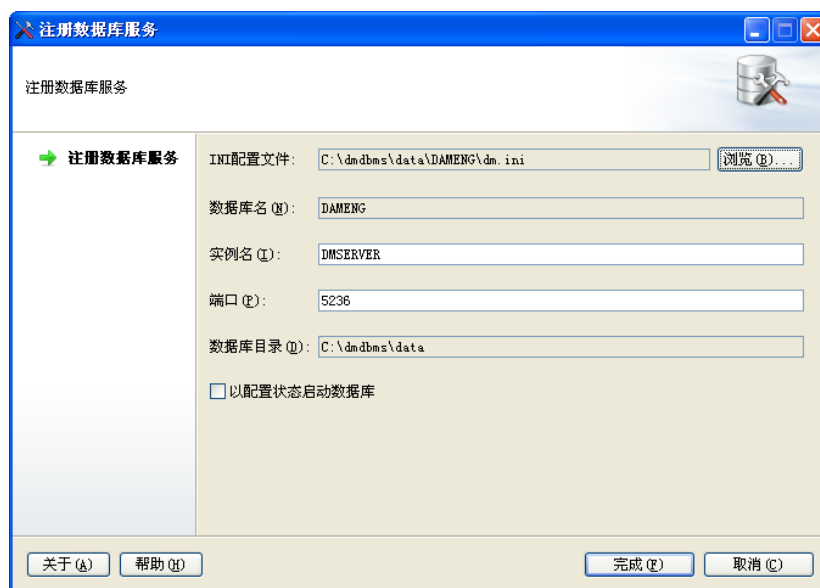


图 7.16 要删除的数据库

正在注册数据库服务，如图 7.17 所示。

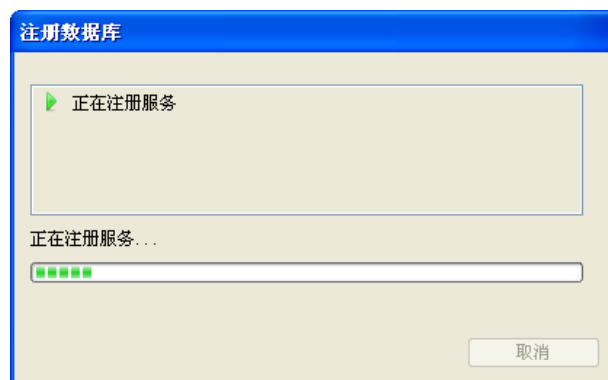


图 7.17 注册数据库服务

注册完成之后将显示对话框，提示或错误反馈信息，如图 7.18 所示。

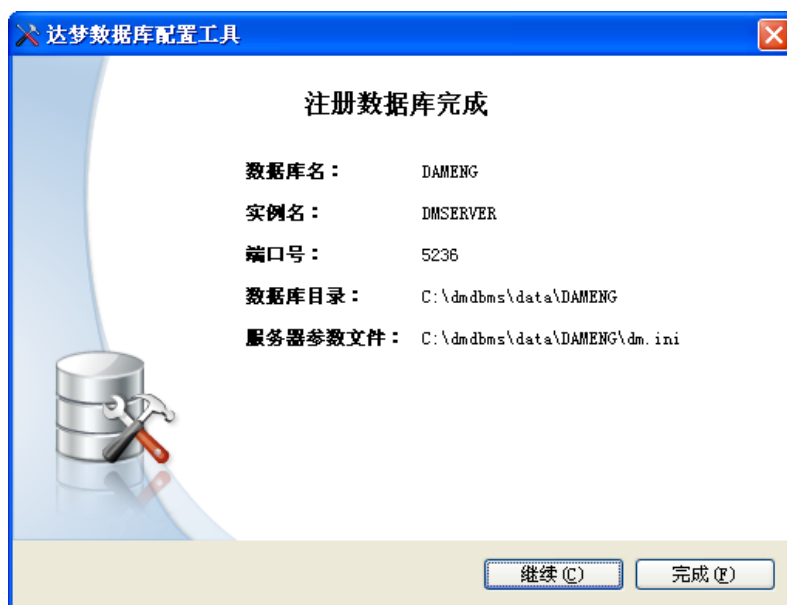


图 7.18 注册数据库完成

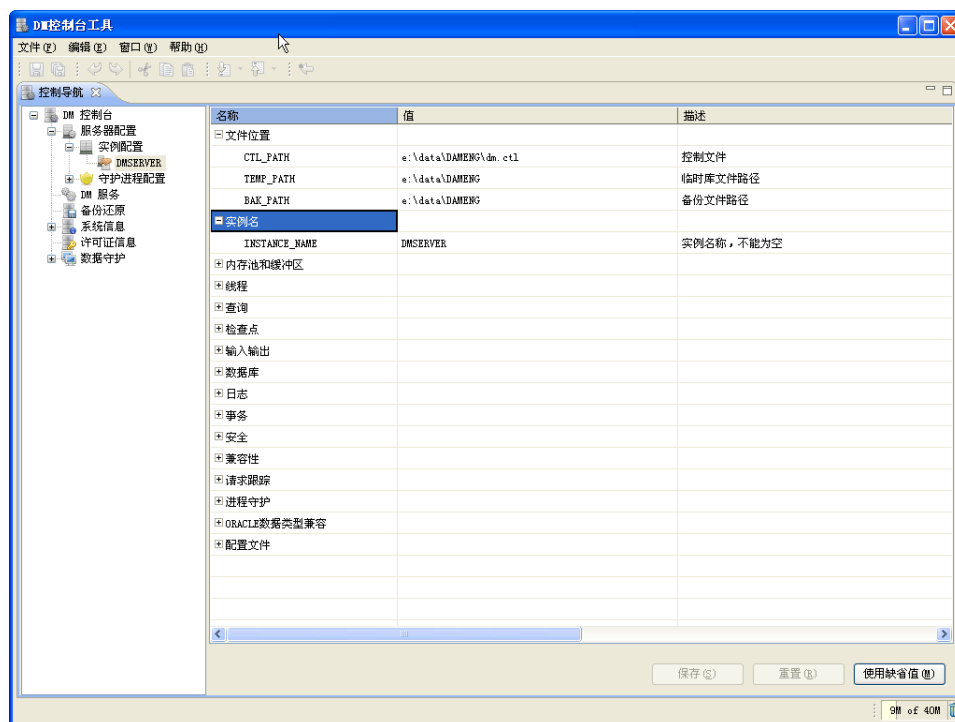
7.5 查看数据库信息

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 7.19 所示。



7.19 查看数据文件

通过 DM 控制台工具，查看实例配置属性，如图 7.20 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。



7.20 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 7.21 所示。

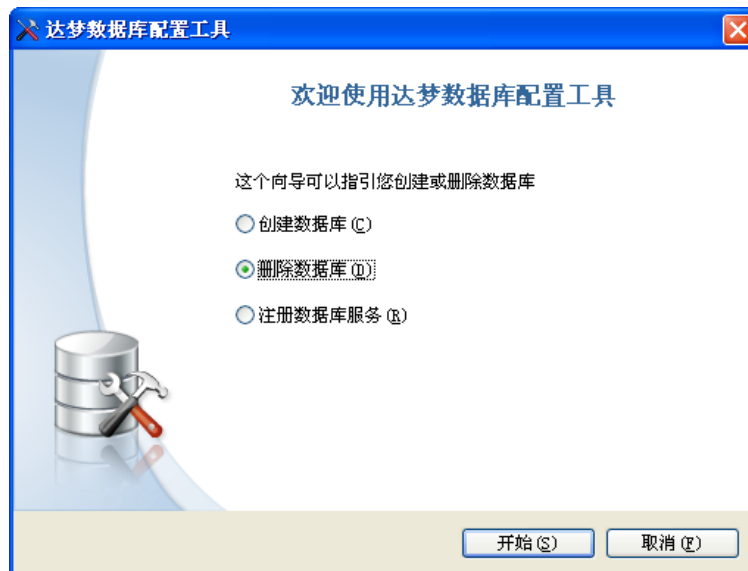


图 7.21 选择操作方式

根据数据库名称，选择要删除的数据库，如图 7.22 所示，也可以通过指定数据库配置文件删除数据库。

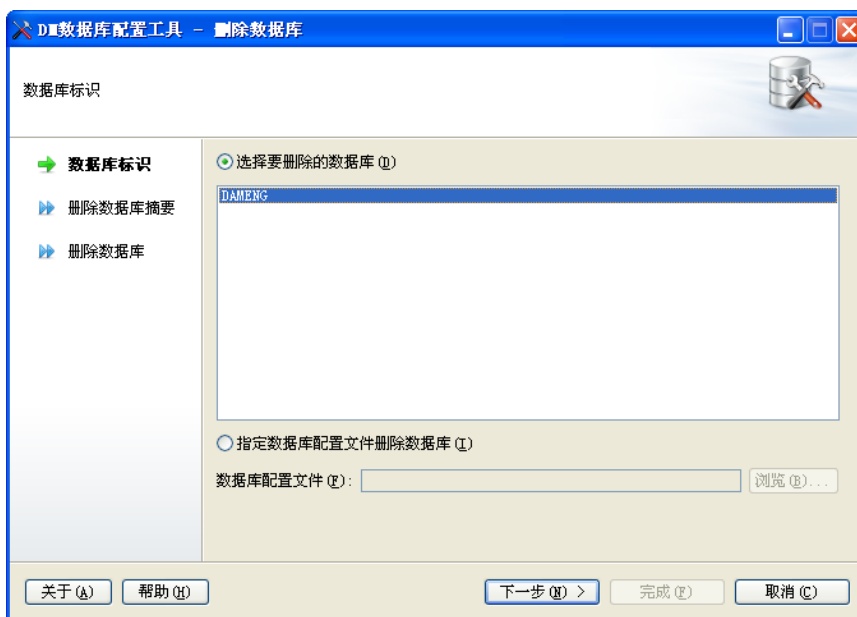


图 7.22 要删除的数据库

确认将删除的数据库名、实例名、数据库目录，如图 7.23 所示。



图 7.23 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 7.24 所示。

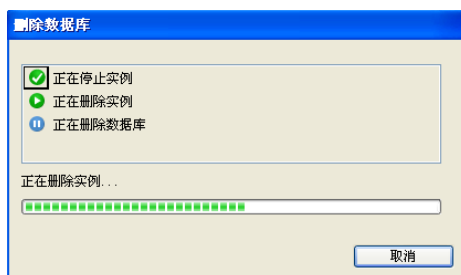


图 7.24 删除数据库

删除完成之后将显示对话框，提示或错误反馈信息，如图 7.25 所示。



图 7.25 删除数据库完成

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。

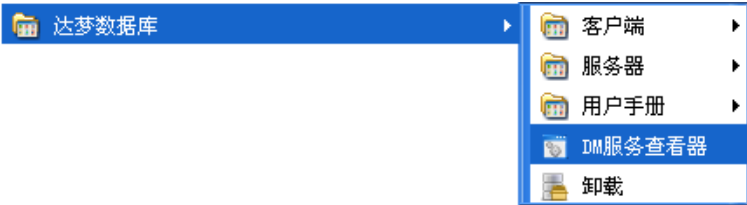


图 8.1 DM 数据库菜单方式启动 1

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。

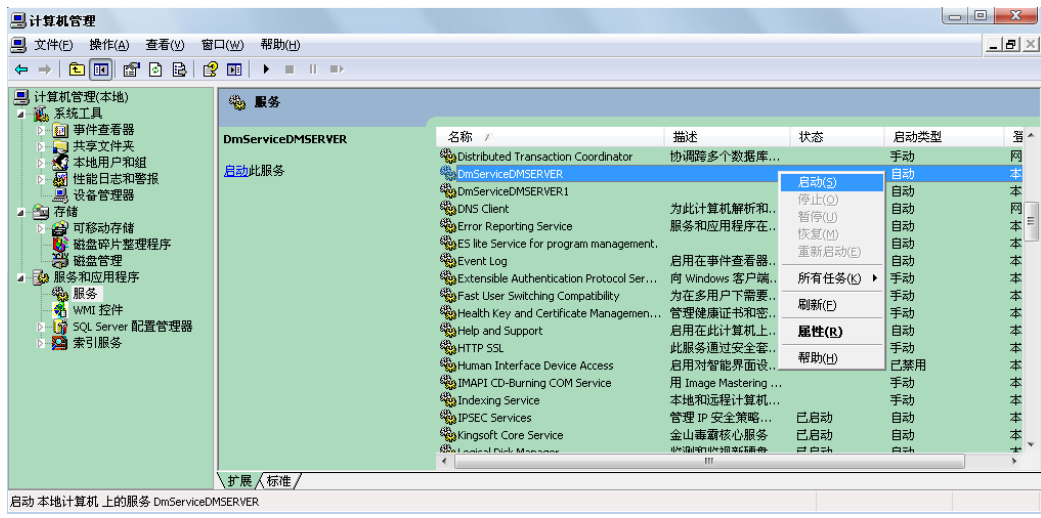


图 8.3 DM 数据库服务方式启动

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。

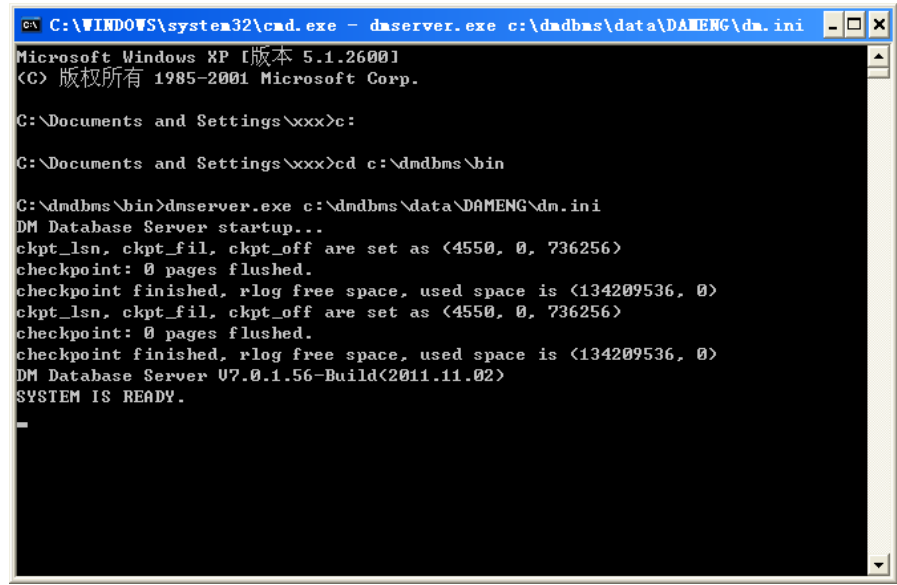


图 8.4 DM 数据库命令行方式启动

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
```

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。

8.2 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. 主机模式（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 日志到备机；
3. 备机模式（STANDBY）：接收主机发送过来的 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。

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、守护进程环境），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。

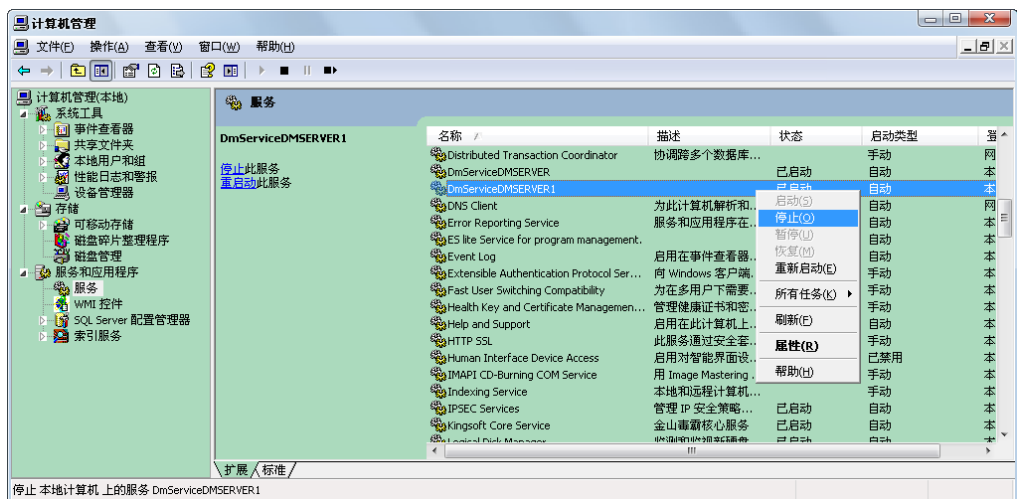
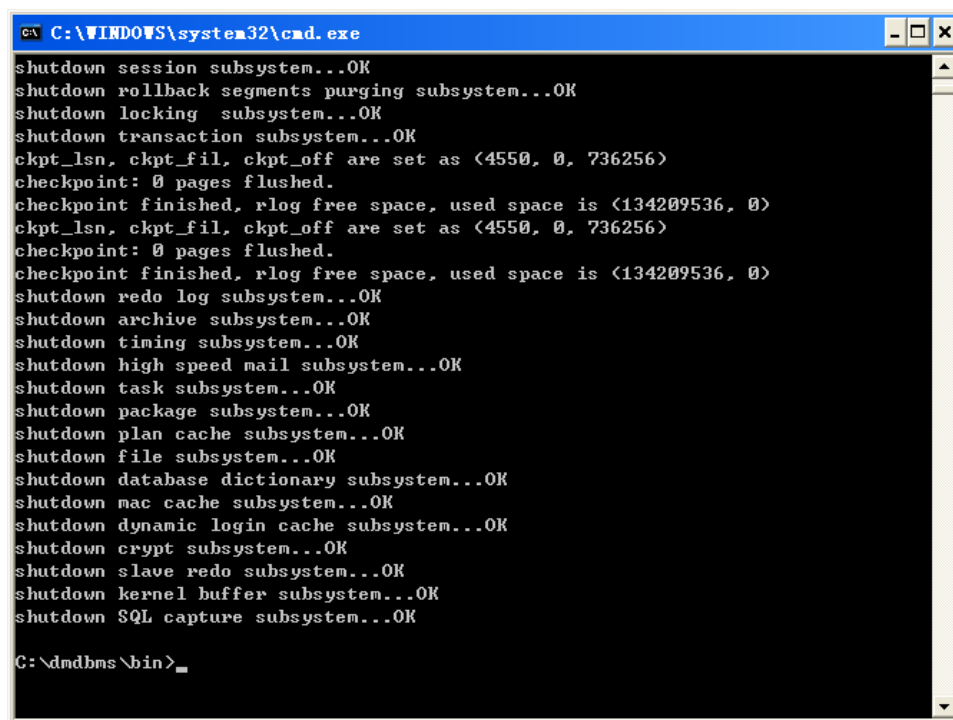


图 8.5 DM 数据库服务方式停止

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。



```
C:\WINDOWS\system32\cmd.exe
shutdown session subsystem...OK
shutdown rollback segments purging subsystem...OK
shutdown locking subsystem...OK
shutdown transaction subsystem...OK
ckpt_lsn, ckpt_fil, ckpt_off are set as <4550, 0, 736256>
checkpoint: 0 pages flushed.
checkpoint finished, rlog free space, used space is <134209536, 0>
ckpt_lsn, ckpt_fil, ckpt_off are set as <4550, 0, 736256>
checkpoint: 0 pages flushed.
checkpoint finished, rlog free space, used space is <134209536, 0>
shutdown redo log subsystem...OK
shutdown archive subsystem...OK
shutdown timing subsystem...OK
shutdown high speed mail subsystem...OK
shutdown task subsystem...OK
shutdown package subsystem...OK
shutdown plan cache subsystem...OK
shutdown file subsystem...OK
shutdown database dictionary subsystem...OK
shutdown mac cache subsystem...OK
shutdown dynamic login cache subsystem...OK
shutdown crypt subsystem...OK
shutdown slave redo subsystem...OK
shutdown kernel buffer subsystem...OK
shutdown SQL capture subsystem...OK

C:\dmdbms\bin>
```

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 LIST 表

对于 LIST 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)，LIST 表最多支持 128 个链表，具体可参考第 20 章“管理 LIST 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 列存储表

对于列存储表可以指定每列对应的区所占用的簇的大小。缺省区大小为 10 个簇。

如下面的建表语句：

```
CREATE VERTICAL TABLE TEST
(C1 INT STORAGE(SECTION(20)),
C2 VARCHAR) STORAGE(SECTION(15));
```

上面定义的 C1 列因为指定了用 20 个簇来存储，而在后面定义的 STORAGE 为 15，所以 C1 的区大小为 20 簇，C2 的区大小为 15 簇，如果后面没有 STORAGE 指定的 15 个簇，则 C2 为 10 个簇，因为系统的默认大小为 10 个簇。

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 1~8187 字节
VARCHAR	SIZE 为 1~8187 字节
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 1~8187 字节
VARBINARY	SIZE 为 1~8187 字节
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，`LIST` 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、`LIST` 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     SHORT             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 LIST 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。我们可以通过手动修改 INI 文件或通过配置修改命令来修改该配置项的值，命令如下：

```
SELECT SP_SET_PARA_VALUE(1, 'PK_WITH_CLUSTER', 1);
```

默认情况下，PK_WITH_CLUSTER 为 0，即建立的主键并不会变为聚簇主键；若为 1，主键会自动升级为聚簇主键。

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 **SELECT** 语句建立的临时表(**CTAS**)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. 临时表的锁定量很少，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，默认情况下是会话级的。

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、默认值和 **NOT NULL** 完整性约束）；
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行：

1. 使用 **DELECT** 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，如果表上关联了行级触发器，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 **employee** 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 **TABLEDEF** 系统函数查看表的定义。

```
SELECT 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(' ident_table ');
SELECT IDENT_SEED(' ident_table ');
SELECT IDENT_INCR(' ident_table ');
```

10.6.3 查看表的空间使用情况

DM 使用段、簇和块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(' space_table ');
SELECT TABLE_USED_PAGES(' 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 22 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. BLOB 和 TEXT 只能建立全文索引、CLOB 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。

如果创建了一个索引来加速存取多列的查询，如 col1、col2 和 col3 的单个索引来加快查

询,则仅存取 col1 或仅存取 col1 和 col2 的查询也被加速,但仅存取 col2 或 col3 或 col2 和 col3 的查询不会使用索引。因此,应考虑实际情况,安排索引列以获得最佳性能。

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 **LIST** 表除外）都是使用 **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；
2. 如果之前已经指定过 **CLUSTER INDEX** 或者指定了 **CLUSTER PK**，则用户新建 **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；
3. 指定 **CLUSTER INDEX** 操作需要重建表上的所有索引，包括 **PK** 索引；
4. 删除聚集索引时，缺省以 **ROWID** 排序，自动重建所有索引；
5. 若聚集索引是默认的 **ROWID** 索引，不允许删除；
6. 聚集索引不能应用到函数索引中；
7. 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；
8. 不能在列存储表上新建/删除聚集索引；
9. 建聚集索引语句不能含有 **partition_clause** 子句；
10. 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. 函数索引不能是 CLUSTER 或 PRIMARY KEY；
2. 函数索引的表达式不能是集函数或非确定性函数；
3. 函数索引的长度不能超过 816 字节（包含生成后的指令和字符串）；
4. BCP 和 FLDR 不支持含有函数索引的表；
5. 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。

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. 不支持对临时表、列存储表以及外部表创建位图索引；
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 上创建索引；
5. BCP 和 FLDR 不支持含有位图索引的表；
6. 不支持备份还原。使用备份文件还原时，如果原始表已经删除，则不会还原位图索引，同时还原后，位图索引将不存在，即不支持这类操作；如果原始表已存在，则

需要把位图索引先删除，否则还原时报错。

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. 适用于常规位图索引的限制也适用于位图连接索引；
2. 索引列都必须为维度表上的列；
3. 不可以对事实表创建位图连接索引；
4. 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；
5. 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；
6. 连接索引创建时，基表只允许出现一次；
7. 不允许对存在 cluster key 的表创建位图连接索引；
8. 位图连接索引表（命名为 **BMJ\$_表名_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。

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(SCHEAM_NAME varchar(256), INDEX_ID int);
```

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

12.1 PL/SQL 概述

PL/SQL 是达梦数据库对标准 SQL 语言的扩展，叫做过程化 SQL 语言。在 PL/SQL 程序中，包括一整套的数据类型、条件结构、循环结构和异常处理结构等，PL/SQL 可以执行 SQL 语句，SQL 语句中也可以使用 PL/SQL 函数。

12.1.1 PL/SQL 优点

与 SQL 相比，PL/SQL 体现了模块化的程序设计思想，具体优点如下：

1. PL/SQL 是一种高性能的基于事务处理的语言，能运行在 DM 环境中，支持所有数据处理命令。通过使用 PL/SQL 程序单元处理 SQL 的数据定义和数据控制元素；
2. PL/SQL 支持所有 SQL 数据类型和所有 SQL 函数，同时支持所有 DM 对象类型；
3. PL/SQL 块可以被命名和存储在 DM 数据库中，同时也能被其他的 PL/SQL 程序或 SQL 命令调用，任何客户/服务器工具都能访问 PL/SQL 程序，具有很好的可重用性；
4. 可以使用 DM 的管理工具 Manager 管理存储在服务器中的 PL/SQL 程序的安全性。可以授权或撤销数据库其他用户访问 PL/SQL 程序的能力；
5. PL/SQL 代码可以使用任何 ASCII 文本编辑器编写，所以对任何达梦数据库能够运行的操作系统都是非常便利的；
6. 对于 SQL，DM 服务器必须处理每一条 SQL 语句，在网络环境下这就意味着每一个独立的调用都必须被 DM 服务器处理，占用大量的服务器时间，容易导致网络拥挤，而 PL/SQL 是以整个语句块被服务器处理，降低了网络拥挤，同时可以大大提高访问数据库的效率。

12.1.2 PL/SQL 语句块

PL/SQL 语言的基本程序单元是语句块，主要的块形式有函数、过程和匿名块。每个语句块由关键字 DECLARE、BEGIN、EXCEPTION 和 END 划分为说明部分、执行部分和异常处理部分。其中执行部分是必须的，说明和异常处理部分可以省略。语句块可以嵌套，它可以出现在任何其他语句可以出现的位置，格式如下：

```
DECLARE
    变量声明部分
BEGIN
    可执行部分
EXCEPTION
    异常处理部分
END
```

其中变量声明部分用来定义变量、类型、游标、子程序、异常等。被声明的元素在本块范围内有效，这一部分在 PL/SQL 块中是可选的。

可执行部分是 PL/SQL 块的主体，它包含了 PL/SQL 块的可执行代码，是必不可少的部分，PL/SQL 块的主要功能体现在这一部分。

异常处理部分用于处理 PL/SQL 块在执行过程中发生的错误。如果块执行正常，则块正常结束，否则从出现错误的语句开始，转至异常处理部分开始执行，即进行异常处理，这部分在 PL/SQL 块中也是可选的。

最简单的 PL/SQL 块仅包含由 BEGIN 和 END 限定的可执行部分。例如：

```
BEGIN
    PRINT 'HELLO WORLD';
END;
```

PL/SQL 块的代码可以直接在“SQL>”提示符下输入。输入结束后，执行命令“/”即可使该 PL/SQL 块开始执行。

12.2 PL/SQL 变量

变量的定义必须放在 PL/SQL 块的变量声明部分，变量定义有两种格式，分别为：

变量名[CONSTANT] 类型 [约束] [DEFAULT 默认值]

变量名[CONSTANT] 类型 [约束] [:=初始值]

约束用于规定变量必须满足的条件，例如 NOT NULL，即指变量不能为空值。

变量名要遵守一定的命名规则。变量名必须以字母开头，包含数字、字母、下划线以及 \$、# 符号，长度不能超过 30 字符，并且不能与达梦数据库的关键字相同，变量名与大小写是无关的。

变量在定义时可以指定默认值或初始值，在 PL/SQL 块的运行过程中也可以对其赋值。赋值语句有两种方式：

(1) 直接赋值语句

变量名:=表达式 或 SET 变量名=表达式

(2) 通过 SQL SELECT INTO 或 FETCH INTO 给变量赋值

SELECT <表达式>{,<表达式>} [INTO <主变量名>{,<主变量名>}] FROM <表引用>{,<表引用>} ...;
或

FETCH [NEXT|PREV|FIRST|LAST|ABSOLUTE N|RELATIVE N]<游标名> [INTO<主变量名>{,<主变量名>}];

注：FETCH 中的 N，起始值为 0。

常量与变量相似，但常量的值在程序内部不能改变，常量的值在定义时赋予，它的声明方式与变量相似，但必须包含关键字 CONSTANT。

如果需要打印输出变量的值，则要调用 PRINT 语句，如果数据类型不一致，则系统会自动将它转换为 VARCHAR 类型输出。除了变量的声明外，变量的赋值、输出等操作都要放在 PL/SQL 块的可执行部分。

下面是一个变量声明和赋值的例子。

```
CREATE TABLE STUDENT(NAME VARCHAR(30), AGE INT, SNO INT PRIMARY KEY);
INSERT INTO STUDENT VALUES('ZHANGSAN', 18, 10010);
INSERT INTO STUDENT VALUES('LISI', 20, 10011);
CREATE OR REPLACE PROCEDURE PROC AS
    C1 CHAR(20) DEFAULT 'ABC';           --默认值
    C2 CONSTANT VARCHAR:= 'HELLO WORLD!'; --常量
    C3 INT NOT NULL := 1;                --约束
    C4 INT;
    NAME1 VARCHAR;
    SNO1 INT;
    CURSOR_1 CURSOR;
BEGIN
```

```

OPEN CURSOR_1 FOR SELECT SNO FROM STUDENT;
SET C3 = 10;
C4 := C3 * 100;
SELECT NAME1 INTO NAME1 FROM STUDENT WHERE SNO = 10010;
FETCH ABSOLUTE 1 CURSOR_1 INTO SNO1;
CLOSE CURSOR_1;
PRINT C1; PRINT C2; PRINT C3; PRINT C4;
PRINT NAME1||' '||SNO1;
END;

```

运行结果:

```

ABC
HELLO WORLD!
10
1000
ZHANGSAN 10011

```

12.3 PL/SQL 数据类型和操作符

在 PL/SQL 块中可以定义变量和数据类型,同时 PL/SQL 有一系列操作符,这使得 PL/SQL 块对数据的处理更加灵活。

12.3.1 SQL 数据类型

PL/SQL 支持所有的 SQL 数据类型,其中包括:字符数据类型、多媒体数据类型、精确数值数据类型、近似数值数据类型、一般日期时间数据类型、时间间隔数据类型,如表 12.1 所示:

表 12.1 PL/SQL 支持的数据类型列表

类型名	类型描述
CHAR(N)	固定串长度为 N 的字符串, N<=8188
VARCHAR(N)	最大字符串长度为 N 的可变长度字符串, N<=8188
IMAGE	影像数据类型, 可变长度的二进制数据, 最大长度为 2G-1
TEXT	文本数据类型, 可变长度的字符数据, 最大长度为 2G-1
BLOB	二进制大对象, 可变长度的二进制数据, 最大长度为 2G-1
CLOB	字符串大对象, 可变长度的字符数据, 最大长度为 2G-1
BIT	单个二进制位数据
BOOL	在数据库中用 1、0 进行存储, 1 表示 TRUE, 0 表示 FALSE
BINARY(N)	固定长度为 N 的二进制数据, N<=8188
VARBINARY(N)	最大长度为 N 的可变长度二进制数据, N<=8188
TINYINT	精度为 3, 刻度为 0 的有符号精确数字, 取值范围-128...127
SMALLINT	精度为 5, 刻度为 0 的有符号精确数字, 取值范围-32,768...32,767
BYTE	与 TINYINT 相似, 精度为 3, 刻度为 0
INTEGER/INT	精度为 10, 刻度为 0 的有符号精确数字, 取值范围-2[31]...2[31]-1
PLS_INTEGER	与 INTEGER 相同
BIGINT	精度为 19, 刻度为 0 的有符号精确数字值, 取值范围-2[63]...2[63]-1
DECIMAL(P,S)	精度为 P, 刻度为 S 的有符号精确数字值, 1≤P≤38, S≤P

DEC(P,S)	与 DECIMAL 相同
NUMERIC(P,S)	精度为 P, 刻度为 S 的有符号精确数字值, $1 \leq P \leq 38$, $S \leq P$
REAL	二进制精度为 24 的有符号近似数字值, 取值范围 0 或者绝对值为: 10[-38]...10[38]
FLOAT	二进制精度为 53 的有符号近似数字值, 取值范围 0 或者绝对值为: 10[-308]...10[308]
DOUBLE	二进制精度为 53 的有符号近似数字值, 取值范围 0 或者绝对值为: 10[-308]...10[308]
DATE	日期数据类型, 年月日字段, 格式与 GREGORIAN (罗马) 日历一致
TIME(P)	时间数据类型, 时分秒字段, 精度 P 指定了秒的精度
TIMESTAMP(P)	时间戳数据类型, 年月日时分秒字段, 精度 P 指定了秒的精度
INTERVAL YEAR(P)	年间隔, 即两个日期之间的年数字, P 为时间间隔的首项字段精度(后面简称为: 首精度)
INTERVAL MONTH(P)	月间隔, 即两个日期之间的月数字, P 为时间间隔的首精度
INTERVAL DAY(P)	日间隔, 即为两个日期/时间之间的日数字, P 为时间间隔的首精度
INTERVAL HOUR(P)	时间间隔, 即为两个日期/时间之间的时数字, P 为时间间隔的首精度
INTERVAL MINUTE(P)	分间隔, 即为两个日期/时间之间的分数字, P 为时间间隔的首精度
INTERVAL SECOND(P,Q)	秒间隔, 即为两个日期/时间之间的秒数字, P 为时间间隔的首精度, Q 为时间间隔秒精度
INTERVAL YEAR(P) TO MONTH	年月间隔, 即两个日期之间的年月数字, P 为时间间隔的首精度
INTERVAL DAY(P) TO HOUR	日时间间隔, 即为两个日期/时间之间的日时数字, P 为时间间隔的首精度
INTERVAL DAY(P) TO MINUTE	日时分间隔, 即为两个日期/时间之间的日时分数字, P 为时间间隔的首精度
INTERVAL DAY(P) TO SECOND(Q)	日时分秒间隔, 即为两个日期/时间之间的日时分秒数字, P 为时间间隔的首精度, Q 为时间间隔秒精度
INTERVAL HOUR(P) TO MINUTE	时分间隔, 即为两个日期/时间之间的时分数字, P 为时间间隔的首精度
INTERVAL HOUR(P) TO SECOND(Q)	时分秒间隔, 即为两个日期/时间之间的时分秒数字, P 为时间间隔的首精度, Q 为时间间隔秒精度
INTERVAL MINUTE(P) TO SECOND(Q)	分秒间隔, 即为两个日期/时间之间的分秒间隔, P 为时间间隔的首精度, Q 为时间间隔秒精度

PL/SQL 除了支持表 12.1 中所示的 SQL 数据类型外, 为了方便用户使用, DM 扩展提供了 %TYPE、%ROWTYPE、记录类型、数组类型、索引表类型和类类型。

12.3.2 %TYPE 和 %ROWTYPE

在许多情况下, 存储过程变量可以被用来处理存储在数据库表中的数据。在这种情况下, 变量应该拥有与表列相同的类型。例如表 T(ID INT, NAME VARCHAR(30)) 中有个字段 NAME 类型为 VARCHAR(30)。对应的在存储过程中, 我们可以声明一个变量: DECLARE V_NAME VARCHAR(30); 但是如果 T 中的 NAME 字段定义发生了变化, 比如变为 VARCHAR(100)。那么存储过程中的变量 V_NAME 也要做相应修改为 DECLARE V_NAME VARCHAR(100); 如果您有很多的变量以及存储过程代码, 这种处理可能是十分耗时和容易出错的。

为了解决上述问题, DM 提供了 %TYPE 类型。%TYPE 可以附加在表列或者另外一个变量上, 并返回其类型。例如: DECLARE V_NAME T.NAME % TYPE; 通过使用 %TYPE,

V_NAME 将拥有 T 表的 NAME 列所拥有的类型；如果表 T 的 NAME 列类型定义发生变化，V_NAME 的类型也随之自动发生变化，而不需要用户手动修改。

下面给出一个 %TYPE 类型的使用例子。

```
CREATE TABLE T(ID INT, NAME VARCHAR(30));
INSERT INTO T VALUES(1, 'ZHANGSAN');
CREATE OR REPLACE PROCEDURE PROC AS
DECLARE
V1 T.NAME %TYPE;
C1 CURSOR;
BEGIN
OPEN C1 FOR SELECT NAME FROM T WHERE ID = 1;
FETCH C1 INTO V1;
PRINT V1;
CLOSE C1;
END;
```

与 %TYPE 类似，%ROWTYPE 将返回一个基于表定义的运算类型，它将一个记录声明为具有相同类型的数据库行。例如：DECLARE V_TREC T % ROWTYPE; 将定义一个记录，该记录中的字段将与表 T 中的列相对应。V_TREC 变量会拥有下面这样的结构：(ID INT, NAME VARCHAR(30))，与 %TYPE 类似，如果表定义改变了，那么 %ROWTYPE 定义的变量也会随之改变。

下面是一个 %ROWTYPE 类型的使用例子。

```
CREATE OR REPLACE PROCEDURE PROC AS
DECLARE
V1 T % ROWTYPE;
C1 CURSOR;
BEGIN
OPEN C1 FOR SELECT * FROM T WHERE ID = 1;
FETCH C1 INTO V1;
PRINT V1.ID; PRINT V1.NAME;
CLOSE C1;
END;
```

12.3.3 记录类型

记录类型类似于 C 语句的结构。记录类型提供了处理分离，但又作为一个整体单元的相关变量的一种机制。例如：DECLARE V_ID INT; V_NAME VARCHAR(30); 这两个变量在逻辑上是相互关联的，因为它们对应 T 表中的相同字段。如果为这些变量声明一个记录类型，那么它们之间的关系就十分明显了。定义记录类型的语法如下：

```
TYPE 记录类型名 IS RECORD
(记录字段名 1 数据类型, [记录字段名 2 数据类型, .....]);
```

记录类型可以单独对记录中的字段赋值，也可以将一个记录直接赋值给另外一个记录。可以使用点标记引用一个记录中的字段（记录名.字段名），如果要进行记录赋值，那么这两个记录必须是相同类型的，或者记录类型中字段的数据类型是相同的。

在下面的例子中，可以看到记录类型单独字段赋值和记录直接给另外一个记录赋值。

```
DECLARE TYPE T_REC IS RECORD( ID INT, NAME VARCHAR(30));
V_REC1 T_REC;
V_REC2 T_REC;
```

```

BEGIN
V_REC1.ID := 100;
V_REC1.NAME:= '雷锋';
V_REC2 := V_REC1;
PRINT V_REC1.ID;PRINT V_REC1.NAME;
PRINT V_REC2.ID;PRINT V_REC2.NAME;
END;

```

下面的例子则显示了记录类型不同，但记录类型中字段的数据类型相同时的赋值。

```

DECLARE
TYPE T_REC IS RECORD( ID INT, NAME VARCHAR(30));
TYPE T_REC_NEW IS RECORD( ID INT, NAME VARCHAR(30));
V_REC1 T_REC;
V_REC2 T_REC_NEW;
BEGIN
V_REC1.ID := 100; V_REC1.NAME:= '雷锋';
V_REC2 := V_REC1;
PRINT V_REC2.ID;
PRINT V_REC2.NAME;
END;

```

12.3.4 数组类型

数组数据类型包括静态数组类型和动态数组类型。静态数组的长度是预先定义好的，在整个程序中，一旦给定大小后就无法改变。而动态数组则不然，它可以随程序需要而重新指定大小。动态数组的内存空间是从堆（HEAP）上分配（即动态分配）的，通过执行代码为其分配存储空间。当程序执行到这些语句时，才为其分配，程序员自己负责释放内存。需要注意的是，DM 中数组下标的起始值为 1。静态数组的语法格式为：

```
TYPE 数组名 IS ARRAY 数据类型 [常量表达式, 常量表达式, ...];
```

或

```
TYPE 数组名 IS ARRAY (常量表达式) OF 数据类型;
```

```
变量名 数组名 [常量表达式, 常量表达式, ...];
```

下面给出一个静态数组使用的例子。

```

DECLARE
TYPE ARR IS ARRAY INT[3];           --TYPE 定义数组类型
TYPE ARR2 IS ARRAY INT[2,3];        --TYPE 定义数组类型
A ARR;                               --用自己定义的数组类型申明数组
B ARR2;                              --多维数组
BEGIN
FOR I IN 1..3 LOOP  --TYPE 定义的数组
A[I] := I * 10;
PRINT A[I];
END LOOP;
FOR I IN 1..2 LOOP
    FOR J IN 1..3 LOOP
        B[I][J] := I * 10 + J;
        PRINT B[I][J];
    
```

```
END LOOP;
END LOOP;
END;
```

动态数组与静态数组的用法类似，区别只在于动态数组没有指定下标，需要动态分配空间。动态数组的语法格式为：

```
TYPE 数组名 IS ARRAY 数据类型 [...];
或
TYPE 数组名 IS TABLE OF 数据类型;
```

```
变量名 数组名[...];
```

多维动态数组分配空间的语法为：

```
数组名 := NEW 数据类型[常量表达式,...];
```

或者可以使用如下语法对多维数组的某一维度进行空间分配：

```
数组名 := NEW 数据类型[常量表达式][...];
```

下面是一个动态数组使用的例子

```
DECLARE
TYPE ARR IS ARRAY INT[];
A ARR;
BEGIN
A := NEW INT[3];          --动态分配空间
FOR I IN 1..3 LOOP
A[I] := I * 10;
PRINT A[I];
END LOOP;
PRINT ARRAYLEN(A); --函数 ARRAYLEN 用于求取数组的长度
END;
```

12.3.5 操作符

PL/SQL 支持的操作符包括：算术操作符、关系操作符、比较操作符、逻辑操作符，如表 12.2 所示。

表 12.2 PL/SQL 支持的操作符列表

操作符类型	操作符	对应操作
算术操作符	+	加
	-	减
	/	除
	*	乘
关系操作符	<	小于操作符
	<=	小于或等于操作符
	>	大于操作符
	>=	大于或等于操作符
	=	等于操作符
	!=	不等于操作符
	<>	不等于操作符
	:=	赋值操作符
比较操作符	IS NULL	如果操作数为 NULL 返回 TRUE

逻辑操作符	LIKE	比较字符串值
	BETWEEN	验证值是否在范围之内
	IN	验证操作数在设定的一系列值中
	AND	两个条件都必须满足
	OR	只要满足两个条件中的一个
	NOT	取反

12.4 PL/SQL 流控制

PL/SQL 提供了丰富的流控制语句，用于对程序的执行流程进行控制。通过流控制语句，我们可以编写更复杂的 PL/SQL 块。流控制语句分为两类，即条件判断语句和循环语句。

12.4.1 IF 语句

IF 语句是基于布尔条件的语句序列，以实现条件分支控制结构。最简单的 IF 语句格式为：

```
IF 条件 THEN
    代码
END IF;
```

如果条件成立，则执行 THEN 后面的语句，否则跳过 THEN 到 END IF 之间的语句，执行 END IF 后面的语句。如果需要在条件不成立时执行另外的代码，格式如下：

```
IF 条件 THEN
    代码 1
ELSE
    代码 2
END IF
```

即当条件成立时执行代码 1，条件不成立时执行代码 2。

IF 可以嵌套，可以在 IF 或 IF ..ELSE 语句中使用 IF 或 IF.ELSE 语句。条件表达式中的因子可以是布尔类型的参数、变量、也可以是条件谓词，控制语句中支持的条件谓词有：比较谓词、BETWEEN、IN、LIKE、和 IS NULL。含 BETWEEN 谓词的条件表达式。

下面是一个 IF.THEN..ELSE 语句的例子

```
CREATE OR REPLACE PROCEDURE P_CONDITION(A INT) AS
BEGIN
    IF A BETWEEN -5 AND 5 THEN
        PRINT 'TRUE';
    ELSE
        PRINT 'FALSE';
    END IF;
END;
```

更复杂的 IF 语句要先后判断多个条件，格式如下：

```
IF 条件 1 THEN
    代码 1
ELSEIF 条件 2
    代码 2
...
ELSE
```

```

    代码 N
END IF

```

在执行上面的 IF 语句时，首先判断条件 1，当条件 1 成立时执行代码 1，否则继续判断条件 2，条件成立则执行代码 2，否则继续判断下面的条件。如果前面的条件都不成立，则执行 ELSE 后面的代码 N。

12.4.2 LOOP 语句

循环控制的基本形式是 LOOP 语句，LOOP 和 END LOOP 之间的语句将无限次的执行。LOOP 语句的基本格式如下：

```

LOOP
    循环体
END LOOP

```

LOOP 和 END LOOP 之间的语句无限次的执行显然是不行的，那么在使用 LOOP 语句时必须使用 EXIT 或 EXIT WHEN 语句结束 LOOP 循环。EXIT 语句使循环结束，一般与 IF 语句相结合使用。EXIT WHEN 指定一个条件，当条件满足时退出循环。

下面是一个 LOOP 循环语句的例子。

```

DECLARE X INT;
BEGIN
    X:=100;
    LOOP
        X:=X+100;
        IF X>1000 THEN
            PRINT X;
            EXIT;
        END IF;
    END LOOP;
END;

```

12.4.3 WHILE 语句

WHILE..LOOP 循环语句在每次循环开始以前，先计算搜索条件，若该条件为 TRUE，语句序列被执行一次，然后控制重新回到循环顶部。若搜索条件为 FALSE，则结束循环。WHILE 语句的格式如下：

```

WHILE 条件 LOOP
    循环体
END LOOP

```

在循环内部，也可以通过 EXIT WHEN、EXIT、CONTINUE WHEN、CONTINUE 来控制循环的执行流程，如下面的例子所示。

```

CREATE OR REPLACE PROCEDURE PROC(A INT) AS
DECLARE I INT:=0;
BEGIN
    WHILE 1 LOOP
        I = I + 1;
        EXIT WHEN I > A;
    END LOOP;

```

```
PRINT I;
END;
```

12.4.4 FOR 语句

LOOP 和 WHILE 循环的循环次数都是不确定的，FOR 循环的循环次数是固定的，循环变量是一个隐式声明的变量，循环变量从起始值开始，每执行一次循环，循环变量自动加 1 或者减 1，直到与终止值相等时，循环结束。如果使用 REVERSE，则按照相反的顺序取集中的元素，即先取最后一个元素，然后依次取前面的元素，直到第一个元素。FOR 语句的格式如下：

```
FOR 循环变量 IN[REVERSE] 起始值...终止值 LOOP
    循环体
END LOOP
```

FOR 语句中的循环变量不需要定义，也不需要显式地将集合中元素赋值给它，但只能在 FOR 循环内使用。循环变量可与当前语句块内的参数或变量同名，这时该同名的参数或变量在 FOR 语句的范围内将被屏蔽。FOR 语句也可以通过 EXIT WHEN、EXIT、CONTINUE WHEN、CONTINUE 来控制循环的执行流程，如下面例子所示。

```
DECLARE I INT:=0;
BEGIN
    FOR I IN 1..5 LOOP
        PRINT I;          --循环变量 I，语句块定义的变量 I 被屏蔽了
        EXIT WHEN I = 3;
    END LOOP;
    PRINT I;              --语句块定义的变量 I
END;
```

运行结果：1 2 3 0

12.4.5 EXIT 语句

EXIT 语句与循环语句一起使用，用于终止其所在循环语句的执行，将控制转移到该循环语句外的下一个语句继续执行。EXIT 还可以和条件表达式结合起来使用，当条件满足时才会退出。EXIT 格式如下：

```
EXIT [LABEL]
[WHEN <条件表达式>]
```

EXIT 语句必须出现在一个循环语句中，否则编译器将报错。

12.4.6 GOTO 语句

GOTO 语句无条件地跳转到一个标签 LABEL 所指的位置，LABEL 在其作用域中必须是唯一的。GOTO 语句将控制权交给带有标号的语句或语句块。格式如下：

```
GOTO <LABEL>
```

但是 GOTO 语句有一些缺点，因为在一个 PL/SQL 语句块中可能会定义很多变量，如果从某一个位置直接跳转到另一个地方，会引起不同部位的变量共享现象，导致程序不易读懂和维护，所以一般情况下不要使用 GOTO 语句来从嵌套很深的循环中直接跳到外面去。为了保证 GOTO 语句的使用不会引起程序的混乱，对 GOTO 语句的使用有下列限制：

1. GOTO 语句不能跳入一个 IF 语句、循环语句或下层语句块中；
2. GOTO 语句不能从一个异常处理器跳回当前块，但可以跳转到包含当前块的上层语句块。

12.4.7 NULL 语句

NULL 语句的语法为：

NULL

该语句不做任何事情，只是用于保证语法的正确性，或增加程序的可读性。例如：

```
CREATE OR REPLACE PROCEDURE PROC(SCORE INT) AS
BEGIN
  IF SCORE = 100 THEN
    PRINT 'YOUR ARE WONDERFUL!';
  ELSE
    NULL;
  END IF;
END;
```

12.5 PL/SQL 访问数据库

编写 PL/SQL 块的目的主要是对数据进行访问，因此，在 PL/SQL 块中可以包含数据查询 SELECT 语句，也可以包含 INSERT、DELETE、UPDATE 等数据操纵语句。需要注意的是，在 PL/SQL 中不能直接包含 DDL 语句，如果要利用 PL/SQL 块完成诸如创建表、修改表结构等操作，需要通过 SQL 语句的动态执行。动态执行 DDL 语句需要在 DM.INI 参数中添加 DYN_DDL_PERMIT = 1。

通过 SQL 语句以及流控制语句，可以编写复杂的 PL/SQL 语句块，对数据库进行复杂的访问。由于 PL/SQL 块一般是在应用程序中调用执行，而不是以交互方式执行，所以在 PL/SQL 块中的 SQL 语句与一般的 SQL 语句有所不同。使用 PL/SQL 来控制对数据库的操作流程，功能更加丰富、灵活。

12.5.1 数据查询

在 PL/SQL 块中通过 SELECT 语句从数据库中检索数据。由于要对数据进行查询以及处理，而不仅仅是显示出来，所以 SELECT 语句需要采用下面的特殊格式先把数据放到一个临时空间中以便后面对它进行引用：

```
SELECT 列 1, 列 2, 列 3... INTO 变量 1, 变量 2, 变量 3...
FROM 表...
```

与一般的 SELECT 语句相比，PL/SQL 块中的 SELECT 语句使用了 INTO 语句，利用 INTO 子句，把查询到的数据存放在变量中，以进行相应的处理。例如下面的例子首先从学生表中检索学生的姓名、年龄、学号信息，分别存放在 O_NAME、O_AGE、O_SNO 本地变量中，然后将这些信息合起来输出。

```
CREATE OR REPLACE PROCEDURE PROC AS
DECLARE
  O_NAME VARCHAR;
  O_AGE INT;
```



```

O_SNO INT;
BEGIN
SELECT NAME, AGE ,SNO INTO O_NAME, O_AGE ,O_SNO
FROM STUDENT WHERE SNO = 10010;
PRINT O_NAME || ' ' || O_AGE || ' ' || O_SNO;
END;

```

需要注意的是，在 **SELECT** 语句中，需要查询的列与 **INTO** 子句中的变量在数目、类型上要一致，否则会报错。

在执行 **SELECT INTO** 语句时，可能会发生两种例外情况，一是没有查询到满足条件的数据，二是存在多行满足条件的数据，这就是通常所说的异常。达梦预定义了一些异常，例如在第一种情况下发生的是异常 **NO_DATA_FOUND**，在第二种情况下发生的异常是 **TOO_MANY_ROWS**，对于这样的异常必须做出相应处理，否则会影响 **PL/SQL** 块的正确执行。

12.5.2 数据操纵

在 **PL/SQL** 块中也可以包含 **INSERT**、**DELETE**、**UPDATE** 语句，用于对数据库中的表进行增、删、改操作，在这些语句中，可以使用数字、字符串等形式的常量，也可以使用变量。下面是一个对表进行 **INSERT** 操作的例子。

```

CREATE OR REPLACE PROCEDURE PROC AS
DECLARE
NAME1 VARCHAR;
AGE1 INT;
SNO1 INT;
BEGIN
INSERT INTO STUDENT VALUES(NAME1, AGE1, SNO1);
NAME1 = 'WANGWU';
AGE1 = 23;
SNO1 = 10013;
INSERT INTO STUDENT VALUES(NAME1, AGE1, SNO1);
END;

```

两条 **INSERT** 语句都是将三个常量作为三个列的值，第一条记录所用的变量还没有初始化，所以插入的记录的三个字段都是 **NULL**。

在删除表中的数据时，在 **DELETE** 语句的 **WHERE** 子句中也可以使用变量，例如下面的例子从 **STUDENT** 表中删除上面插入的两条记录。

```

CREATE OR REPLACE PROCEDURE PROC AS
DECLARE
NAME1 VARCHAR;
BEGIN
DELETE FROM STUDENT WHERE NAME IS NULL;
NAME1 = 'WANGWU';
DELETE FROM STUDENT WHERE NAME = NAME1;
END;

```

在修改表中数据时，在 **UPDATE** 语句的 **SET** 子句和 **WHERE** 子句中可以使用变量，如下例所示。

```

CREATE OR REPLACE PROCEDURE PROC (NAME1 VARCHAR(30), AGE1 INT) AS
BEGIN

```

```
UPDATE STUDENT SET AGE = AGE1 WHERE NAME = NAME1;
END;
```

调用存储过程 CALL PROC('ZHANGSAN',24),即可将学生名字为 ZHANGSAN 的年龄改为 24 岁。

12.5.3 数据定义

在 PL/SQL 中也可以动态执行 DDL 语句,但首先必须在 DM.INI 参数中添加 DYN_DDL_PERMIT = 1。

下面是一个在 PL/SQL 中执行 CREATE TABLE 语句的例子。

```
CREATE OR REPLACE PROCEDURE PROC AS
SQL_STMT VARCHAR(100);
BEGIN
    SQL_STMT := 'CREATE TABLE TEST(I INT)';
    EXECUTE IMMEDIATE SQL_STMT;
END;
```

调用存储过程 CALL PROC,即可创建表 TEST。

12.6 PL/SQL 与数据库交互

没有名字的 PL/SQL 程序块被称为匿名块,它仅存放在缓冲区中,只能在当前 DISQL 环境中执行。如果希望 PL/SQL 块能随时被调用执行,并且能被数据库用户共享,就需要创建存储程序。存储程序是有名字的 PL/SQL 块,它在创建时经过了编译与优化,被存放在数据库中,任何用户只要有适合的权限,就可以根据它的名字进行多次调用。存储程序在调用时无需再进行编译,因此能以很快的速度执行,但如果需要对存储程序进行修改,不能直接修改它的源代码,只能执行 CREATE 命令重新创建。

存储程序可以在 SQL 语句、应用程序以及其它 PL/SQL 块中被调用,在第一次被调用时,存储程序的代码被装载到系统全局区的共享池中,再次调用时即可直接从共享池中取出代码执行。存储程序的形式包括:存储过程、存储函数、触发器、包等,它为用户提供了一种高效率的编程手段,成为现代数据库系统的重要特征。

12.6.1 存储过程

用户在自己的模式中创建存储过程,需要具有 CREATE PROCEDURE 数据库权限,如果要在其它用户的模式中创建存储过程,则需要具有 CREATE ANY PROCEDURE 数据库权限。创建存储过程的语法为:

```
CREATE [OR REPLACE] PROCEDURE
过程名 [WITH ENCRYPTION] [(参数 1, 参数 2, ...)]
AS
    声明部分
BEGIN
    可执行部分
EXCEPTION
    异常处理部分
END;
```

其中 **OR REPLACE** 选项的作用是当同名的存储过程存在时，首先将其删除，再创建新的存储过程，前提条件是当前用户具有删除原存储过程的权限，如果没有删除权限，则创建失败。存储过程可以带有参数，这样在调用存储过程时就需指定相应的实际参数，如果没有参数，过程名后面的圆括号和参数列表就可以省略了。参数定义的格式如下：

参数名 [参数传递模式] 数据类型 [:= 默认值]

参数名 [参数传递模式] 数据类型 [DEFAULT 默认值]

参数传递模式包括：输入参数（**IN**）、输出参数（**OUT**）、输入输出参数（**IN OUT**）三种形式，其中 **IN** 是默认的参数传递模式。调用存储过程时，如果没有提供实际参数，则使用参数默认值。

WITH ENCRYPTION 为可选项，如果指定 **WITH ENCRYPTION** 选项，则对 **BEGIN** 到 **END** 之间的语句块进行加密，防止非法用户查看其具体内容。可直接创建加密后的存储过程或函数，语法为：

CREATE [OR REPLACE] PROCEDURE|FUNCTION

过程名|函数名 **WRAPPED** 密文;

加密后的存储过程或函数的定义可在 **SYS.SYSTEEXTS** 系统表中查询。

可执行部分是存储过程的核心部分，由 **SQL** 语句和流控制语句构成。异常处理部分用于处理存储过程在执行过程中可能出现的错误。

下面的例子创建了一个存储过程，功能是查看 **STUDENT** 表中学生总数。

CREATE OR REPLACE PROCEDURE STUDENT_COUNT AS

TOTAL_NUM INT;

BEGIN

SELECT COUNT(*) INTO TOTAL_NUM FROM STUDENT;

PRINT TOTAL_NUM;

END;

调用存储过程 **CALL STUDENT_COUNT**，即可获取学生总数。

所有用户都可调用自己创建的存储过程，如果要调用其他用户的存储过程，则需要对该存储过程具有 **EXECUTE** 权限，即存储过程的所有者将 **EXECUTE** 权限授予该用户。授予 **EXECUTE** 权限的语法为：

GRANT EXECUTE ON 过程名 TO 用户;

删除存储过程的语法为：

DROP PROCEDURE 过程名;

12.6.2 存储函数

存储函数与存储过程在结构和功能上十分相似，但还是有所差异的。它们的区别在于：

1. 存储过程没有返回值，调用者只能通过访问 **OUT** 或 **IN OUT** 参数来获得执行结果，而存储函数有返回值，它把执行结果直接返回给调用者；
2. 存储过程中可以没有返回语句，而存储函数必须通过返回语句结束；
3. 不能在存储过程的返回语句中带表达式，而存储函数必须带表达式；
4. 存储过程不能出现在一个表达式中，而存储函数只能出现在表达式中。

创建存储函数的语法格式为：

CREATE OR REPLACE FUNCTION

函数名 [**WITH ENCRYPTION**] (参数 1,参数 2,...) **RETURN** 返回类型

AS

声明部分

BEGIN

可执行部分

```

RETURN 表达式;
EXCEPTION
    异常处理部分
END;
```

如果用户要在自己的模式中创建存储函数，只需具有 `CREATE FUNCTION` 的数据库权限，如果要在其它用户的模式中创建存储函数，则需要具有 `CREATE ANY FUNCTION` 的数据库权限。

下面的例子创建一个存储函数，实现斐波那契数列算法。

```

CREATE OR REPLACE FUNCTION FIBONACCI(A INT) RETURN INT AS
    DECLARE S INT; LV INT; RV INT;
BEGIN
    IF A <= 0 THEN
        PRINT '请输入正数';
    ELSIF A = 1 OR A = 2 THEN
        S := 1;
    ELSE
        RV := FIBONACCI(A - 1);
        LV := FIBONACCI(A - 2);
        S := LV + RV;
        PRINT LV || '+' || RV || '=' || S;
    END IF;
    RETURN S;
END;
```

调用函数 `SELECT FIBONACCI(4)`，即可获取 4 的斐波那契数列。

每个用户都可以直接调用自己创建的存储函数，如果要调用其它用户的存储函数，则需要具有对相应存储函数的 `EXECUTE` 权限。为此，存储函数的所有者要将 `EXECUTE` 权限授予适当的用户，授予 `EXECUTE` 权限的语句格式为：

```
GRANT EXECUTE ON 函数名 TO 用户
```

删除存储函数的语句格式为：

```
DROP FUNCTION 函数名;
```

存储过程和存储函数统称为存储模块，存储模块的调用命令有三种：

1. `SELECT` 语句调用；
2. `CALL` 语句调用；
3. 直接在 `DISQL` 中输入存储过程或函数名即可调用。

上述三种调用方式的不同之处在于：

1. `SELECT` 调用存储模块必须是有返回值的，也就是说 `SELECT` 命令只能调用存储函数；
2. 如果通过 `CALL` 来调用，它不要求调用的是存储过程还是存储函数，如果存储模块有返回值，则 `CALL` 命令会忽略它的返回值，执行结果是成功或者失败；
3. 如果不通过任何语句直接调用存储模块，默认方式为 `CALL`。

12.6.3 游标

由于 DM 支持的 SQL 语言是面向集合的语言，一条 SQL 语句可以产生或处理多条记录，而 PL/SQL 语言是面向记录的，一组主变量一次只能存放一条记录，所以仅使用主变量并不能满足 SQL 语句向应用程序输出数据的要求。为解决这一问题，SQL 语言引用了游标来协调这两种不同的工作方式，从而为应用程序逐个处理查询结果提供了强有力的手段。

使用游标必须先定义，定义游标实际上是定义了一个游标工作区，并给该工作区分配了一个指定名字的指针（即游标）。游标工作区用以存放满足查询条件行的集合，因此它是一个说明性语句，是不可执行的。在打开游标时，就可从指定的基表中取出所有满足查询条件的行送入游标工作区并根据需要分组排序，同时将游标置于第一行的前面以备读出该工作区中的数据。当对行集合操作结束后，应关闭游标，释放与游标有关的资源。

DM 支持的 SQL 语言提供了有关游标的语句：定义游标语句、打开游标语句、拨动游标语句和关闭游标语句。

定义游标的语法格式为：

```
CURSOR 游标名 IS|FOR SELECT 语句;
```

```
游标名 CURSOR;
```

```
CURSOR 游标名;
```

为了处理游标中的数据，首先要打开游标。打开游标的语法格式为：

```
OPEN 游标名
```

```
OPEN 游标名 FOR SELECT 语句;
```

游标打开后，就可以取出游标中的数据，并对其进行处理。拨动游标的语法格式为：

```
FETCH [NEXT|PRIOR|FIRST|LAST|ABSOLUTE N|RELATIVE N] <游标名 >
```

```
[INTO <主变量名 >{,<主变量名>}];
```

FETCH 命令可以将一行数据中各列的值依次赋给指定的变量，需要注意的是，变量的类型、数目要与游标中的一行的各列相对应。游标在使用完后应及时关闭，以释放它所占用的内存空间，关闭游标的语法格式为：

```
CLOSE 游标名
```

当游标关闭后，不能再从游标中获取数据。如果需要，可以再次打开游标。

下面给出一个典型的使用游标的例子。

```
CREATE OR REPLACE PROCEDURE PROC AS
```

```
STR VARCHAR;
```

```
CURSOR CSR FOR SELECT NAME FROM STUDENT; --声明游标并绑定 SQL 语句
```

```
CURSOR CSR1; --先声明，打开游标的时候再绑定 SQL 语句
```

```
BEGIN
```

```
OPEN CSR;
```

```
OPEN CSR1 FOR SELECT NAME FROM SYSOBJECTS
```

```
WHERE NAME='STUDENT';
```

```
LOOP
```

```
FETCH CSR INTO STR; --从游标 CSR 中取数据;
```

```
EXIT WHEN CSR%NOTFOUND;
```

```
PRINT STR;
```

```
END LOOP;
```

```
FETCH CSR1 INTO STR; --从游标 CSR1 中取数据;
```

```
PRINT STR;
```

```
CLOSE CSR;
```

```
CLOSE CSR1;
```

```
END;
```

运行结果：

```
ZHANGSAN
```

```
LISI
```

```
STUDENT
```

在 DM 中，游标的属性主要包括以下四种，如表 12.3 所示。

表 12.3 游标的属性

属性	描述
----	----

CURSOR%ISOPEN	判断当前游标是否已经打开，如果打开，该属性值为 TRUE，否则为 FALSE
CURSOR%ROWCOUNT	表示到目前为止，用 FETCH 语句已经取到的行数
CURSOR%FOUND	表示最后一次 FETCH 操作是否从游标中取到一行数据，如果已经取到，其值为 TRUE，如果没有取到，其值为 FALSE
CURSOR%NOTFOUND	和 CURSOR%FOUND 正好相反

以上游标的属性被称为显式游标属性。在这四个游标属性中，%ISOPEN 属性用于测试游标的状态。其它三个属性用来测试 FETCH 命令的执行结果，%FOUND 和%NOTFOUND 属性用来测试最近的一次 FETCH 是否取到数据，%ROWCOUNT 属性表示自游标打开以来，到目前为止，用 FETCH 命令获取的行数。

在大多数情况下，游标的应用步骤是：打开游标；开始循环；从游标中取出一行数据并进行处理；关闭循环；关闭游标，这一类应用称为游标用于循环。但还有一种循环与这种类型不同，这就是游标 FOR 循环。用于 FOR 循环的游标与一般游标的区别在于不需要显式的打开、关闭、取数据，测试数据的存在、定义存放数据的变量等。

下面的例子进行游标 FOR 循环，将学生的年龄加 1。

```
DECLARE
CURSOR STU_CURSOR IS
SELECT * FROM STUDENT FOR UPDATE;
BEGIN
FOR STU_ROW IN STU_CURSOR LOOP
UPDATE STUDENT SET AGE = AGE + 1 WHERE CURRENT OF STU_CURSOR;
END LOOP;
COMMIT;
END;
```

从上面的例子中我们可以看到游标 FOR 循环确实很好的简化了游标的开发，我们不再需要 OPEN、FETCH 和 CLOSE 语句，也不需要 %FOUND 属性检测是否到最后一记录，这一切 DM 服务器隐式的帮我们完成了。

12.6.4 引用游标

引用游标有以下几个特点：

1. 不只局限于一个查询，可以为一个查询声明或者打开一个游标，然后对其结果集进行处理，之后又可以将这个引用游标为其它的查询打开；
2. 可以对引用游标进行赋值；
3. 可以像用一个变量一样在一个表达式中用这个引用游标；
4. 可以作为一个子程序的参数，也就是它可以像 C 语言中的变量一样充当参数；
5. 可以在 PL/SQL 的不同子程序中传递结果集。

引用游标的语法、用法与一般游标几乎是一样的，区别在于引用游标在声明的时候没有 IS 或 FOR 语句，且同一个游标变量再次打开游标的时候可以关联另一个查询语句，而原来的游标句柄会被自动关闭。还可以对引用游标进行赋值操作，语法如下：

```
DEST_CURSOR := SOURCE_CURSOR;
```

给一个引用游标变量赋值时，可以继承所有 SOURCE_CURSOR 的属性。如果 SOURCE_CURSOR 已经打开，则 DEST_CURSOR 也已经打开，也就是说 DEST_CURSOR 只是个变量，它相当于一个指针，指向的位置和 SOURCE_CURSOR 是完全一样的。

下面是一个使用引用游标的例子。

```
CREATE OR REPLACE PROCEDURE PROC AS
STR VARCHAR;
CURSOR C1 IS SELECT NAME FROM STUDENT WHERE SNO = 10010;
```

```

CURSOR C2 IS SELECT NAME FROM STUDENT WHERE SNO = 10011;
CURSOR CSR := C1;
BEGIN
OPEN C1;          --赋值后再打开
    IF CSR%ISOPEN THEN
        PRINT 'CURSOR IS ALREADY OPENED';
        FETCH CSR INTO STR;
        PRINT STR;
    ELSE
        PRINT 'CURSOR IS NOT OPENED';
    END IF;
OPEN C2;  --打开后再赋值
CSR := C2;
    IF CSR%ISOPEN THEN
        PRINT 'CURSOR IS ALREADY OPENED';
        FETCH CSR INTO STR;
        PRINT STR;
    ELSE
        PRINT 'CURSOR IS NOT OPENED';
    END IF;
CLOSE C1;
CLOSE C2;
END;

```

运行结果:

```

CURSOR IS ALREADY OPENED
ZHANGSAN
CURSOR IS ALREADY OPENED
LISI

```

12.6.5 动态 SQL

SQL 的动态执行是通过语句 EXECUTE IMMEDIATE 实现的，它的作用是执行一个指定的 SQL 语句，这个 SQL 语句可以带参数，也可以不带参数。如果带参数，则 EXECUTE IMMEDIATE 后面需要加上 USING 来指定 SQL 语句中参数的值；如果不带参数，EXECUTE IMMEDIATE 后面直接跟上需要执行的 SQL 语句即可。动态 SQL 的语法格式如下：

```

EXECUTE IMMEDIATE <SQL 动态语句文本> [INTO VARIABLES]
[USING [参数 1][参数 2]...];

```

1. 该语句只能在嵌入式方式中使用；
2. 该语句首先分析动态语句文本，检查是否有错误，如果有错误则不执行它，并在 SQLCODE 中返回错误码；如果没发现错误则执行它；
3. 一般来说，应该使用一个字符串变量来表示一个动态 SQL 语句的文本，下列语句不能作动态 SQL 语句：CLOSE、FETCH、OPEN。如下面几个例子所示：

```

CREATE OR REPLACE PROCEDURE PROC AS
CURSOR C1;
SQL_STMT1 VARCHAR := 'OPEN C1 FOR SELECT NAME FROM STUDENT';
BEGIN
    EXECUTE IMMEDIATE SQL_STMT1;

```

```
END;
```

运行结果：不支持的语句类型.ERROR CODE=-604

```
CREATE OR REPLACE PROCEDURE PROC AS --
CURSOR C1;
SQL_STMT1 VARCHAR := 'CLOSE C1;';
BEGIN
OPEN C1 FOR SELECT NAME FROM STUDENT;
EXECUTE IMMEDIATE SQL_STMT1;
END;
```

运行结果：不支持的语句类型.ERROR CODE=-604

```
CREATE OR REPLACE PROCEDURE PROC AS
CURSOR C1;
STR VARCHAR;
SQL_STMT1 VARCHAR := 'FETCH ABSOLUTE 1 C1 INTO STR;';
BEGIN
OPEN C1 FOR SELECT NAME FROM STUDENT;
EXECUTE IMMEDIATE SQL_STMT1;
PRINT STR;
CLOSE C1;
END;
```

运行结果：不支持的语句类型.ERROR CODE=-604

动态 SQL 语句不支持嵌入式异常处理 WHENEVER，即不支持下列类似语句：

```
EXEC SQL WHENEVER SQLERROR GOTO sql_error; .....
```

```
sql_error:
```

```
EXEC SQL WHENEVER SQLERROR CONTINUE;
```

动态 SQL 语句的文本也可以是一个 PL/SQL 语句块，如下例所示：

```
CREATE OR REPLACE PROCEDURE PROC AS
CURSOR C1;
SQL_STMT1 VARCHAR := 'DECLARE STR VARCHAR; BEGIN SELECT NAME INTO STR FROM
STUDENT WHERE SNO=10010; PRINT STR;END;';
BEGIN
EXECUTE IMMEDIATE SQL_STMT1;
END;
```

如果动态 SQL 语句的文本中有多条 SQL 语句，那么只执行第一条语句，如下例所示：

```
CREATE OR REPLACE PROCEDURE PROC AS
SQL_STMT VARCHAR := 'SELECT NAME FROM STUDENT WHERE SNO=10010;
SELECT NAME FROM STUDENT WHERE SNO=10011;';
BEGIN
EXECUTE IMMEDIATE SQL_STMT;
END;
```

运行结果：ZHANGSAN

对于仅执行一次的动态语句，用立即执行语句较合适。如果动态 SQL 语句是查询语句并且结果集只有一条记录，则可以通过 INTO 语句把结果存储赋值给变量，如下例所示：

```
CREATE OR REPLACE PROCEDURE PROC AS
NAME VARCHAR := 'STUDENT';
SQL_STMT VARCHAR := 'SELECT NAME FROM SYSOBJECTS WHERE NAME = ?';
STR VARCHAR;
BEGIN
```



```
EXECUTE IMMEDIATE SQL_STMT INTO STR USING NAME;
PRINT STR;

END;
```

在动态执行 SQL 语句的时候也可以对指定的 SQL 语句设置参数。指定参数有两种方式，如果某个值设置为参数，可以用“?”来表示，也可以用“:VARIABLE”来表示，如下例所示：

```
CREATE OR REPLACE PROCEDURE PROC AS
    STR VARCHAR := 'HELLO';
    START_POS INT:= 1;
    END_POS INT := 3;
    STR_SQL VARCHAR := 'SELECT SUBSTR(:X, :Y, :Z)';
BEGIN
    EXECUTE IMMEDIATE STR_SQL USING STR, START_POS, END_POS;
END;
```

运行结果：HEL

用“?”表示参数时，在传入时可以是任意的值，但参数个数一定要与“?”的个数相同，同时数据类型一定要匹配（可以互相转换也可以），不然会报数据类型不匹配的错误。

用“:VARIABLE”表示参数时，对于 VARIABLE 的名字会进行检测，如果参数列表中存在相同名字的 VARIABLE，则用这个名字指定的参数被设置的值在第一次绑定的时候确定，同时同名的其它参数都不能再被绑定参数，否则会报“参数个数不匹配.ERROR CODE = -3205”的错误，如下例所示：

```
CREATE OR REPLACE PROCEDURE PROC AS
    STR VARCHAR := 'HELLO';
    START_POS INT:= 1;
    END_POS INT := 3;
    STR_SQL VARCHAR := 'SELECT SUBSTR(X, :Y, :Y)';
BEGIN
    EXECUTE IMMEDIATE STR_SQL USING STR, START_POS, END_POS;
END;
```

运行结果：参数个数不匹配.ERROR CODE=-3205。

同名参数只能绑定一次，上例中的语句如果改成 EXECUTE IMMEDIATE STR_SQL USING STR, START_POS，即去掉 END_POS 即可。运行结果为：H。

12.7 异常处理

用户编写的 PL/SQL 块在执行过程中不可避免地会发生一些错误。这里涉及的错误并不是由于程序的语法引起的，而是因为处理的数据超出了处理的范围而引发的错误。如果给这样的错误起一个名字，这就是异常。当 PL/SQL 块在执行过程中检测到一个错误时，就会抛出相应的异常。在块中应当处理这样的异常，否则会引起应用程序的运行停止。

12.7.1 异常处理程序

异常一般是在 PL/SQL 程序中执行错误时由服务器抛出，也可以在 PL/SQL 块中由程序员在一定的条件下显式抛出，无论是哪种形式的异常，都可以在 PL/SQL 块的异常处理部分编写一段程序进行处理，如果不做任何处理，异常将被传递到调用者，由调用者统一处理。

如果要在 PL/SQL 块中对异常进行处理，就需要在异常处理部分编写处理程序。异常处

理程序的格式如下：

```
EXCEPTION
    WHEN 异常 1 [OR 异常 2 ...] THEN
        异常处理程序 1;
    WHEN OTHERS THEN
        异常处理程序 2;
END;
```

异常处理程序以关键字 **EXCEPTION** 开始，以关键字 **END** 结束。在这部分可以对多个异常分别进行不同的处理，也可以进行相同的处理。如果没有列出所有异常，可以用关键字 **OTHERS** 代替其它的异常，在异常处理程序的最后加上一条 **WHEN OTHERS** 子句，用来处理前面没有列出的所有异常。

如果 PL/SQL 块执行出错，或者遇到显式抛出异常的语句，则程序立即停止执行，转去执行异常处理程序。异常被处理结束后，整个 PL/SQL 块的执行便告结束。所以一旦发生异常，则在 PL/SQL 块的可执行部分中，从发生异常的地方开始，以后的代码将不再执行。

在 PL/SQL 块中有三种类型的异常，即预定义的异常、非预定义的异常和用户自定义的异常。下面分别介绍这几种异常的使用方法。

12.7.2 预定义的异常

为方便用户编程，DM 提供了一些预定义的异常，这些异常与最常见的 DM 错误相对应。如表 12.4 所示：

表 12.4 数据库预定义的异常

异常名称	SQLCODE	说明
TOO_MANY_ROWS	-1091	SELECT INTO 语句返回的记录数大于 1
NO_DATA_FOUND	-3421	没有数据
DUP_VAL_ON_INDEX	-2310	违反唯一性约束
INVALID_CURSOR	-3209	对游标的操作非法
ZERO_DIVIDE	-1800	游标的非法操作

此外，还有一个特殊的异常名 **OTHERS**，它处理所有没有明确列出的异常。**OTHERS** 对应的异常处理语句必须放在其他异常处理语句之后。

下面给出一个异常处理的例子。

```
DECLARE
    OBJ_NAME VARCHAR(50);
BEGIN
    SELECT NAME INTO OBJ_NAME FROM STUDENT WHERE SNO = 10010;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        PRINT '没有找到满足条件的数据';
    WHEN TOO_MANY_ROWS THEN
        PRINT '找到的记录数大于 1 条，不满足 SELECT INTO 的条件';
    WHEN OTHERS THEN
        PRINT 'OTHERS EXCEPTION';
END;
```

PL/SQL 的 **WHEN** 子句中也可以指定多个异常的名字，用 **OR** 分隔，例如：

```
WHEN NO_DATA_FOUND OR TOO_MANY_ROWS THEN
    PRINT 'SELECT 结果集不符合要求';
```

在向表的主键列上写入一个重复的值时将引发异常 DUP_AVL_ON_INDEX，如下例所示：

```
CREATE OR REPLACE PROCEDURE ADD_STUDENT
(NAME VARCHAR, AGE INT, SNO INT) AS
BEGIN
    INSERT INTO STUDENT VALUES(NAME, AGE, SNO);
EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
        PRINT '违反主键的唯一性约束';
        PRINT 'ERROR CODE: ' || '%DM_EXCEPTION_OBJ'. "CODE";
        PRINT 'DESCRIBE: ' || '%DM_EXCEPTION_OBJ'. "DESCR";
    WHEN OTHERS THEN
        PRINT 'OTHERS EXCEPTION';
END;
```

从这个例子可以看出，PL/SQL 在异常处理的异常类型中提供了两个成员，“CODE”和“DESCRIBE”，分别表示引起异常时发生错误的错误码及错误码的描述信息。在日常处理中，我们可以创建一个表用来存储在 PL/SQL 的异常处理部分出现的错误情况，或者生成日志信息，以备后续对错误的统计、分析以及处理。

引发异常的一个重要原因是处理数据时发生错误。编写 PL/SQL 的主要目的是处理数据，而 PL/SQL 块在逻辑与数据是分开的，程序员根本无法预料数据的变化，所以在编写程序时需要考虑各种可能出现异常的情况，并对这些异常进行相应的处理。

12.7.3 用户自定义的异常

除了 DM 定义的两异常外，在 PL/SQL 中还可以用户自定义异常。程序员可以把一些特定的状态定义为异常，在一定的条件下抛出，然后利用 PL/SQL 的异常机制进行处理。用户自定义异常的语法格式为：

异常名称 EXCEPTION [FOR 自定义的错误码]

或

<异常变量名> EXCEPTION;

PRAGMA EXCEPTION_INIT(<异常变量名>, <错误号>);

抛出异常的语法格式为：

RAISE 异常名

异常一般在一定的条件下抛出，因此 RAISE 语句通常跟在某个条件判断的后面，这样就把这个异常与这个条件关联起来了，抛出异常的原因可能是数据出错，也可能是满足了某个自定义的条件，处理自定义异常的方法与处理前两种异常的方法相同。

下面是一个用户自定义异常的例子。

```
CREATE OR REPLACE PROCEDURE PROC AS
DECLARE
    E1 EXCEPTION FOR -10001;
BEGIN
    RAISE E1;
EXCEPTION
    WHEN E1 THEN
        PRINT 'NORMAL EXCEPTION' || ' ' || '%DM_EXCEPTION_OBJ'. "CODE";
    WHEN OTHERS THEN
        PRINT 'OTHER EXCEPTION' || ' ' || '%DM_EXCEPTION_OBJ'. "CODE";
```

END;

12.8 PL/SQL 调试工具

DM 为用户提供了 PL/SQL 调试工具 DMDBG。DMDBG 可调试在 DISQL 中直接执行的非 DDL 语句或语句块，以便程序开发人员定位 PL/SQL 中存在的错误。用户要使用 DMDBG 调试工具，首先要执行 dbms_dbg_pkg.sql 创建调试所需要的包。

12.8.1 命令行参数

DMDBG 的整个运行过程中可以处于初始状态(S)、待执行(W)、执行(R)、调试(D)、执行结束(O)等不同的状态。

初始状态(S)：工具启动完成后，尚未设置调试语句；

待执行状态(W)：设置调试语句后，等待用户执行；

执行状态(R)：开始执行后，未中断而运行的过程；

调试状态(D)：执行到断点或强制中断后进入交互模式；

执行结束(O)：执行完当前设置的调试语句，并返回结果。

DMDBG 在不同的状态下可以执行不同的操作，具体如表 12.5 所示。备注表示不同的命令分别在哪儿几种状态下可以使用。

表 12.5 DMDBG 调试命令

命令行参数	含义	备注
LOGIN	登录	S/W/D/O
SQL	设置调试语句	S/W/O
B	设置断点	W/D/O
INFO B	显示断点信息	W/D/O
D	取消断点	W/D/O
R	执行语句	W/O
CTRL+C	中断执行	R
L	显示调试脚本	D
C	继续执行	D
N	单步执行	D
S	执行进入	D
F或FINISH	执行跳出	D
P	打印变量	D
BT	显示堆栈	D
UP	上移栈帧	D
DOWN	下移栈帧	D
KILL	结束当前执行	W/D/O
QUIT	退出调试工具	S/W/D/O

12.8.2 功能描述

1. 登录

提供交互与命令行 2 种登录方式。交互方式与 DISQL 相同，使用者根据提示依次输入服务器、用户名、口令及端口号等登录信息。

命令行登录参数格式如下：

```
DMDBG <用户名>/<口令>[@<服务器名IP>[:<端口号>]]
```

默认用户名和口令为 SYSDBA/SYSDBA，服务器 LOCALHOST，端口号为 5236，则登录命令为：

```
DMDBG SYSDBA/SYSDBA@LOCALHOST:5236
```

2. 设置语句

命令 SQL 供用户设置需要执行的 SQL 语句，其命令格式如下：

```
<SQL> ['/']
```

SQL 语句是非 DDL 语句或语句块。如果 SQL 是单条语句，则以分号 “;” 结尾；如果是语句块，则语句块后必须以单独一行的斜杠符 “/” 结束。在调试中行数从 SQL 语句实际其实行开始计数。

在实际执行前，用户可以多次调用 SQL 命令设置待调试的语句，而 DMBG 只是记录最后一次的设置语句。例如：

第一步，在 DISQL 中创建表和存储过程。

```
CREATE TABLE STUDENT (ID INT, NAME VARCHAR(20));      --创建游标类型对应的表
INSERT INTO STUDENT VALUES(10010,'ZHANGSAN');
INSERT INTO STUDENT VALUES(10011,'LISI');
```

```
CREATE OR REPLACE PROCEDURE P0 AS                      --创建需要调试的存储过程
BEGIN
PRINT 'HELLO';
END;
/
```

```
CREATE CLASS MYCYCLE                                  --定义类对象
AS
ID INT;
C1 INT;
FUNCTION MYCYCLE(ID INT , C1 INT) RETURN MYCYCLE;
END;
/
```

```
CREATE OR REPLACE CLASS BODY MYCYCLE                  --定义类体
AS
FUNCTION MYCYCLE(ID INT, C1 INT) RETURN MYCYCLE
AS
BEGIN
THIS.ID = ID;
THIS.C1 = C1;
RETURN THIS;
END;
END;
```

```

/

CREATE OR REPLACE PROCEDURE P AS                                --创建需要调试的存储过程
TYPE T_REC IS RECORD(ID INT, NAME VARCHAR(20));                --定义记录类型
TYPE ARR is ARRAY INT[4];                                       --定义数组类型
R1 T_REC;                                                         --声明记录类型
C1 CURSOR;                                                        --声明游标类型
B ARR;                                                            --声明数组类型
O MYCYCLE;                                                        --声明对象类型
I1 INT;                                                           --声明整型
V1 VARCHAR(20);                                                  --声明字符串类型
BEGIN
R1.ID = 10012;
R1.NAME = 'WANGWU';
OPEN C1 FOR SELECT * FROM STUDENT;
LOOP
FETCH C1 INTO I1,V1;
EXIT WHEN C1%NOTFOUND;
END LOOP;
FOR I IN 1..4 LOOP
B[I] := I * 10;
PRINT B[I];
END LOOP;
O = NEW MYCYCLE(1, 2);
CALL P0;
END;
/

第二步, 在 DMDBG 中调试存储过程 P, 设置调试语句如下:
DBG>SQL CALL P;

```

3. 设置断点

设置断点命令为 **B**, 可以为指定的脚本设置执行中断位置。调试执行到断点设置位置后, 将自动中断当前执行。设置断点命令有 2 种不同格式。

B <方法名>

B [<方法名>:]<行号>

如果省略方法名, 则为当前方法或语句块设置行号断点。包中的方法可以设置断点, 但不能指定行号; 如果断点设置在包上, 则可以指定行号, 但可以设置断点部分只能是该包体的初始化代码。

行号从 1 开始计数。对于方法来说, 其开始的定义声明部分语句也包含在行号计数中。对于断点的行号设置必须遵守以下原则:

定义声明部分不能设置断点;

如果一条语句跨多行, 则断点应设置在语句最后一行;

如果多条语句在同一行, 则执行中断在其中第一条语句执行前;

如果设置断点行号不能中断, 则自动将其下移到第一个可以中断的行;

可以在调试过程中动态添加断点。如果在同一位置重复设置断点, 则重复断点被忽略。

例如:

```

DBG>B 3                --没有方法名, 则在最顶层设置断点
No line 3 in method "@dbg_main". --由于顶层只有一行, 无法设置断点, 报错返回
DBG>B P:3              --不能设置断点, 则下移到第一条可以中断的行

```

```
Breakpoint 1 at SYSDBA.P, line 11@{R1.ID = 10012;}
DBG>B P:12                --直接在指定行设置断点
Breakpoint 2 at SYSDBA.P, line 12@{R1.NAME = 'WANGWU';}
```

4. 显示断点

显示断点命令为：

```
INFO B
```

该命令显示所有已设置的断点。显示的每个断点包含以下信息：

- 1) 序号。这是一个自增值，从 1 开始。设置断点时自动为其分配。
- 2) 方法名。如果为语句块，则方法名为空。
- 3) 行号。

例如：

```
DBG> INFO B
```

Num	Schema	Package	Method	What
1	SYSDBA	NULL	P	line 11@{R1.ID = 10012;}
2	SYSDBA	NULL	P	line 12@{R1.NAME = 'WANGWU';}

5. 取消断点

取消断点的命令格式为：

```
D <断点序号>
```

断点序号可以通过 **INFO B** 命令获取，删除后断点的编号不能重用。取消断点命令可以在工具运行的任何时间调用。

例如：

```
DBG>D 1                    --删除断点，成功则等待再次输入
DBG>D 3                    --删除不存在的断点，则报错
No breakpoint number 3.
DBG>INFO B                --显示删除后的断点信息
DBG> INFO B
Num      Schema  Package Method  What
2        SYSDBA  NULL     P         line 12@{R1.NAME = 'WANGWU';}
```

6. 执行语句

执行语句命令为：

```
R
```

命令 **R** 执行设置的 SQL 语句。此命令无参数，执行 **R** 命令前必须已经设置过调试语句，且调试过程中命令 **R** 不允许重复使用。

如果设置了断点，则执行到断点指定的位置时中断，中断将转入调试状态，并显示当前执行所在行的语句；否则，执行完成并显示结果。

例如：

```
DBG>R
Breakpoint 2, SYSDBA.P line: 12@{R1.NAME = 'WANGWU';}
DBG>R                    --调试过程中不可以再次运行
The program is already running.
DBG>B P:32              --调试过程中可以设置断点
Breakpoint 3 at SYSDBA.P, line 23@{CALL P0;}
```

7. 显示脚本

可以使用命令 **L**，在工具中显示指定的方法脚本，但不能显示包的单个方法脚本。此命令可以帮助用户准确设置断点位置。**L** 命令每次显示 5 行代码，多次执行 **L** 命令时，从以显示的下一行开始显示。**L** 命令格式如下：

```
L <方法名>
```

方法名可以包含模式、包等前缀。在调试状态下，可以省略方法名，则显示当前正在运

行的方法。例如：

```
DBG>L SYSDBA.P
1      CREATE OR REPLACE PROCEDURE P AS
2      TYPE T_REC IS RECORD(ID INT, NAME VARCHAR(20));
3      TYPE ARR is ARRAY INT[4];
4      R1 T_REC;
5      C1 CURSOR;
DBG>L      --注意，第二次以后显示脚本不要带方法名，否则从第一行显示
6      B ARR;
7      O MYCYCLE;
8      I1 INT;
9      V1 VARCHAR(20);
10     BEGIN
```

8. 中断执行

中断执行命令为：

CTRL+C

在正常执行过程中，当监听到用户输入 **CTRL+C**，则 **DMDBG** 强制中断当前执行，转入调试状态。该命令对于调试死循环错误非常有效。

9. 继续执行

继续执行命令为：

C

在调试状态下，可以使用命令 **C** 继续 **SQL** 语句的执行，直到运行到断点或执行完成。例如：

```
DBG>C
Breakpoint 3, SYSDBA.P line: 23@{CALL P0;}
```

10. 执行进入

执行进入命令为：

S

如果当前中断语句包含方法调用，则执行进入命令 **S** 可以将方法调用展开，执行到其语句块的第一条语句中断。如果当前行没有方法调用，则命令 **S** 与单步执行命令 **N** 等效。

例如：

```
DBG>S      --调入存储过程 P0
SYSDBA.P0 line: 1      @{CREATE OR REPLACE PROCEDURE P0 AS}
```

11. 单步执行

单步执行命令为：

N

在调试状态下，单步执行命令 **N** 可以执行到当前层次下一个可中断的行。如果当前行包含多条语句，则所有该行语句都被执行。

例如：

```
DBG>N      --单步执行存储过程 P0
SYSDBA.P0 line: 3      @{PRINT 'HELLO';}
```

12. 执行跳出

执行进入命令为：

F 或 FINISH

命令 **F** 可以执行当前函数的余下操作，如果剩余行有断点则执行到断点中断，否则返回到上层调用后中断。如果当前已经是最上层，则与单步执行命令 **N** 等效。

例如：

```
DBG>F      --跳出存储过程 P0
```


SYSDBA.P line: 32 @ {CALL P0;}

13. 打印变量

打印变量命令为：

P <变量名>

命令 P 可以打印指定的变量值。SQL 类型中，除了大字段数据，其它类型都可以打印。

PLSQL 数据类型往往为复杂数据类型，其显示格式如下：

1) 结构类型

在大括号内显示结构所有成员的值，各成员值用 “,” 分隔。

2) 数组类型

如果是数组，则可以与结构类型一样，依次显示数组所有成员的值，也可以只显示其中某一下标的值。

3) 对象类型

可以打印对象中所有的静态成员，或者打印某一个静态成员，函数可以不打印。

4) 游标类型

游标也是对象类型，其成员如表 12.6 所示。

表 12.6 游标对象成员

名称	数据类型	说明
ROWCOUNT	BIGINT	查询结果集总行数。如果游标未打开或查询失败时该值无效
FOUND	INT	最近一次 FETCH 操作是否取得数据。如果取得为 1，否则为 0。如果游标未打开或查询失败时该值无效
ISOPEN	INT	游标是否打开。打开为 1，否则为 0

P 命令只能打印当前方法所在栈帧中的变量，如果要打印其它层次方法栈帧的变量，首先需要移动到对应栈帧。打印变量的信息以 \$ 开始，从 1 开始计数，每打印一个变量，自动增加 1。

例如：

```

DBG>R                                --开始执行
Breakpoint 2, SYSDBA.P line: 12@ {R1.NAME = 'WANGWU';}
DBG> P R1.ID                          --打印记录类型中单个字段
$1 = 10012
DBG> N                                ----单步执行后，R1 所有字段都被赋值
SYSDBA.P line: 13      @ {OPEN C1 FOR SELECT * FROM STUDENT;}
DBG>P R1                              --打印记录类型 R1
$2 = {ID=10012, NAME=WANGWU}
DBG>C                                --继续运行，遇断点则中断
Breakpoint 2, SYSDBA.P line: 32@ {CALL P0;}
DBG>P C1                              --打印游标的所有成员值
$3 = {ISOPEN=1, FOUND=0, ROWCOUNT=2}
DBG>P B                                --打印数组
$4 = {10, 20, 30, 40}
DBG>P B[3]                            --数组下标以 1 开始
$5 = 30
DBG> P O.C1                           --打印对象类型中的成员
$6 = 2
DBG> P O                              --打印对象类型
$7 = MYCYCLE<0>{ID=1, C1=2}

```

14. 显示堆栈

显示堆栈命令为：

BT

命令 **BT** 可以显示当前执行的堆栈状态。首先显示当前方法执行到的行数，然后从下至上显示各层方法名称及调用参数值，输出参数不打印。

例如：

DBG>BT

#0	SYSDBA.P()	CALL P0;	--#0 表示最下层
#1	@dbg_main	CALL P	--表示栈帧只有两层

15. 下移栈帧

下移栈帧命令为：

DOWN

命令 **DOWN** 可以将当前调试栈帧下移一层。下移后，显示当前层执行到的语句行数。如果当前已经是堆栈最下层，则此命令无效。

例如：

DBG>DOWN

--存储过程已到最下层，无法下移

Bottom (innermost) frame selected; you cannot go down.

16. 上移栈帧

上移栈帧命令为：

UP

命令 **UP** 可以将当前调试栈帧上移一层。上移后，显示当前层执行到的语句行数。如果当前已经是堆栈最上层，则此命令无效。

例如：

DBG>UP

#1 @dbg_main CALL P

DBG>UP

--已经到最上层，无法上移

Initial frame selected; you cannot go up.

DBG> C

--执行结束，并显示结果

10

20

30

40

HELLO

17. 结束执行

结束执行命令为：

KILL

在调试过程中，用户可以使用命令 **KILL** 结束当前正在调试的 **SQL** 语句，并重新设置预调试的 **SQL** 语句。

KILL 之后，必须重新设置 **SQL** 语句，断点信息被清空。**R** 执行结束返回结果之后：如果没有重新设置 **SQL** 语句，则再次运行的仍是之前设置的 **SQL** 语句，断点信息可以重用；如果重新设置了 **SQL** 语句，则断点信息亦被清空。

18. 退出工具

退出 **DMDBG** 工具的命令为：

QUIT

如果不希望继续调试，可以输入命令 **QUIT**。**DMDBG** 工具将中断连接，退出执行。

12.9 小结

DM 的存储模块机制是一种技术，而不是一种独立的工具，它是和服务器紧密结合在一起的。可以认为这种技术是执行 DM PL/SQL 语言的一种虚拟机，它可以接受任何有效的存储模块，按照语言本身所规定的语义执行，并将结果返回给客户。

存储模块机制具有如下优点：

1. 提供更高的生产率：在设计应用时，围绕存储过程/函数设计应用，可以避免重复编码，提高生产率；在自顶向下设计应用时，不必关心实现的细节，编程方便；
2. 便于维护：用户的存储模块在数据库集中存放，用户可以随时查询、删除、修改它们，而应用程序可以不作任何修改，或只做少量调整；
3. 提供更好的性能：存储模块在创建时被编译成伪码序列，在运行时不需要重新进行编译和优化处理，具有更快的执行速度，可以同时被多个用户调用，并能够减少操作错误；
4. 可以使用存储过程来帮助实施数据的安全性。可以不授权用户直接访问应用程序中的一些表，而是授权用户执行访问这些表的一个过程。当过程被执行时，它将以过程拥有者的权限来执行。除非通过过程，否则用户就不能访问这些表；
5. 存储模块是一种高效访问数据库的机制，使用存储模块可显著减少应用对 DM 的访问次数，提高系统性能；
6. DM 存储模块的编写主要是采用 PL/SQL 语言，DM PL/SQL 语言对三种控制结构（分支结构、迭代结构和顺序结构）都提供了相应的语句支持。DM PL/SQL 还支持异常处理、动态 SQL、游标变量以及返回结果集等功能。同时，DM 也预先定义了一些常用的存储过程和函数供用户使用。

第 13 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 DM PL/SQL 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. 语句级触发器：对每一个触发语句触发一次；
2. 行级触发器：对受语句影响的表中的每一行触发一次；
3. 事件触发器：基于特定系统事件进行触发；
4. 时间触发器：基于时间而进行触发。

13.1 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
    BEFORE|AFTER|INSTEAD OF
    DELETE|INSERT|UPDATE [OF 列名]
    ON 表名
    [FOR EACH ROW [WHEN 条件]]
    BEGIN
        PL/SQL 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（**BEFORE**），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行触发器。如果没有这样的子句，则创建触发器为语句级触发器。**INSTEAD OF** 触发器固定为行级触发器。

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 **PL/SQL** 块的编写方法相同。在触发器中可以定义变量，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 触发器名 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;
```

13.2 语句级触发器

如果一个触发器在用户每次进行 DML 操作时被激发而且执行一次，而不管这个 DML 操作影响了多少行数据，这个触发器就是语句级触发器。

语句级触发器有前激发和后激发两种形式。前激发是在 DML 操作执行之前被激发执行的，后激发是在 DML 操作执行之后被激发执行。无论是哪种形式，触发器都将执行一次。

例如，我们创建一个前激发触发器 stu_trg，当用户对表 students 的 grade 列进行 UPDATE 操作时该触发器将被激发。创建该触发器的语句为：

```
CREATE TRIGGER stu_trg
BEFORE UPDATE OF grade
ON students
BEGIN
    PRINT '您正在修改 students 表中的 grade 列';
END;
```

如果我们执行下面的 UPDATE 语句：

```
SQL> UPDATE students SET grade = 99 WHERE sno = 100;
```

那么会得到如下结果：

```
您正在修改 students 表中的 grade 列
5 rows affects
```

从 UPDATE 语句的执行结果可以看出，实际受影响的数据有 5 行，但是触发器只执行了一次。实际上不管这条语句影响了一行、两行还是多行，甚至是没有影响任何数据，这个触发器都将在 UPDATE 语句执行之前被激发执行一次。再考虑下面的 UPDATE 语句，相信我们会对语句级触发器的执行有更深入的理解，这次我们修改一个根本不存在的学生学号为 10000：

```
SQL> UPDATE students SET grade = 99 WHERE sno = 10000;
```

那么会得到如下结果：

```
您正在修改 students 表中的 grade 列
0 rows affects
```

如果我们更关心用户对表所实施的访问本身，而不是该次访问影响的数据行数，这时我们可以在表上创建语句级触发器。

13.3 行触发器

如果在某个表上创建了一个触发器，在对这个表进行 DML 操作时，每影响一行数据，该触发器都将被激发执行一次，那么这个触发器就是行触发器。首先看一个非常简单的例子：

```
CREATE OR REPLACE TRIGGER stu_trg
AFTER DELETE
ON students
```

```

FOR EACH ROW
BEGIN
    PRINT '1 行已经被删除';
END;

```

这个触发器是后触发的，如果对表 `students` 进行 `DELETE` 操作，将激发这个触发器的执行，在创建触发器时，使用了 `FOR EACH ROW` 子句，因此这个触发器是行触发器，例如，如果执行了以下的 `DELETE` 语句，执行结果为：

```

SQL> DELETE FROM STUDENT WHERE GRADE0 = 89;
1 行已经被删除
1 行已经被删除
1 行已经被删除
1 行已经被删除
4 rows affected

```

上面的信息表明，共有 4 行数据受到影响。从触发器执行的输出信息可以看出，触发器确实执行了 4 次，也就是说，触发器执行的次数等于受 DML 操作影响的数据行数。

如果 DML 操作没有影响到任何一行数据，那么触发器将不执行。例如，如果要根据一个不存在的 `sno` 值从 `students` 表中删除数据，结果如下：

```

SQL> DELETE FROM students WHERE sno = 11111;
0 rows affected

```

可见，如果 DML 语句没有影响到任何行，触发器将不会执行。

如果我们更关心语句对每行数据的访问情况，而不是 DML 操作本身的信息，那么在表上创建行触发器更合适。

在创建行触发器时，我们可以指定一些条件，这样只有当特定的数据受到 DML 语句影响时，触发器才被激发执行，可以在 `FOR EACH ROW` 子句之后使用 `WHEN` 子句指定条件。

例如，重新考虑上面的简单的行触发器。如果只对学生学成绩为 100 分的进行监视，只有当从表 `students` 中删除 `grade` 值为 100 的行时，才激发触发器。这个触发器可以这样创建：

```

CREATE OR REPLACE TRIGGER stu_trg
AFTER DELETE
ON students
FOR EACH ROW WHEN(:OLD.grade=100)
BEGIN
    PRINT '1 行已经被删除';
END;

```

如果执行两次 `DELETE` 操作，分别删除 `grade` 为 99 的和 100 的行，执行结果为：

```

SQL> DELETE FROM stduents WHERE GRADE = 100;
1 行已经被删除
1 行已经被删除
1 行已经被删除
3 rows affected
SQL> DELETE FROM STUDENT WHERE GRADE = 99;
3 rows affected

```

可以看出，当删除成绩为 99 分的信息时，没有激发触发器，而删除 100 分的数据时，可以每行数据都激发一次触发器。

行触发器通常用于对用户的 DML 操作进行合法性检查，使得用户修改数据的操作必须按照一定的规则进行。

为了能够比较修改前和修改后的数据，在触发器的可执行代码中，可以使用两个关联行——`NEW` 和 `OLD`。它们分别表示触发器被激发时，当前行的原数据和新数据。`NEW` 表示修改后的行，通过 `NEW` 可以引用新行中的各个列的值，如 `NEW.sno`；`OLD` 则表示修改前的行。

对于 DELETE 命令, NEW 没有意义, OLD 表示被删除的一行。

在触发器的可执行代码中,如果要通过 OLD 和 NEW 引用某个列的值,要在前面加上“:”,在其它地方,则不使用“:”。

现在,我们要创建一个比较复杂的触发器,说明对 NEW 和 OLD 的应用。

```
CREATE OR REPLACE TRIGGER trig
BEFORE INSERT ON myinfo
FOR EACH ROW
WHEN (:NEW.sno = 1)
BEGIN
    :NEW.sno = :NEW.sno + 1;
    :NEW.name = :NEW.name || 'World';
    INSERT INTO myinfo VALUES(:NEW.sno, :NEW.name);
    PRINT 'HELLO WORLD';
END;
/
```

假设 myinfo 表是个空表,插入一条记录:

```
INSERT INTO myinfo VALUES(1, 'Hello ');
```

之后对 myinfo 表进行查询,查询结果为:

	SNO	NAME
1	2	Hello World
2	2	Hello World
2 rows got		

13.4 事件触发器

前面两种触发器都是基于数据的触发器,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、LOGOUT、AUDIT、NOAUDIT、BACKUP DATABASE、RESTORE DATABASE、TIMER 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 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、LOGOUT、SERERR、BACKUP DATABASE 和 RESTORE DATABASE 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。

13.5 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 PL/SQL 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
    AFTER TIMER ON DATABASE
    {时间定义语句}
    BEGIN
        PL/SQL 语句
    END;
```

时间定义语句的语法规则较多，在这里就不叙述了，可以参照《达梦数据库 SQL 手册》中对时间触发器的语法的详细描述。

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
    str VARCHAR;
    BEGIN
        PRINT 'HELLO WORLD';
    END;
/
```

时间触发器的作用是很多的，比如在凌晨（一般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等类似的事情。同时也可以作为一定时器通知一些用户在未来的某些时间要做哪些事情。

13.6 触发器总结

语句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。

对于语句级触发器和行级触发器来说，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。

第 14 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。

14.1 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 DM《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 B。

14.2 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。序列的定义存储在数据字典表 SYSSEQUENCES 中。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 $2^{31}-1$ ；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^{31} 。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 DM《SQL 语言使用手册》。

14.3 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同的名字，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 DM《SQL 语言使用手册》。

14.4 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
2	16778216	1

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 `VIEWDEF` 系统函数来查看。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。

```
SELECT VIEWDEF('SYSDBA', 'VIEW1');
```

第 15 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。

15.1 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 的《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');
```

15.2 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 PL/SQL 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。

15.3 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;
```

15.4 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。

15.4.1 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，用户应当在上面对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。

15.4.2 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 t1 (
  id NUMBER(5) CONSTRAINT t1_pk PRIMARY KEY);
ALTER TABLE t1
ADD CONSTRAINT t1_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。

15.4.3 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 t1 DISABLE CONSTRAINT t1_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 t1 DROP CONSTRAINT t1_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。

15.4.4 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 t1_pk 的信息。

```
SELECT * FROM SYSOBJECTS WHERE NAME='T1_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。

15.5 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。

15.6 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。

15.7 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 16 章 Disql

达梦的 disql 是用来与达梦数据库进行交互的客户端工具。在 disql 中，可以运行 disql 语句与 disql 命令。

通常所说的 DML、DDL 语句都是 disql 语句，当它们执行完后，都被保存在一个特定的内存区域中。用户可以通过上下键查找到这些保存在内存中的 sql 语句，并可以进行修改，然后再次执行。除了 disql 语句，在 disql 中执行的其它语句我们称之为 disql 命令。它们执行完后，不保存在内存区域中，它们一般被用来对输出的结果进行格式化显示，以便于制作报表。

disql 语句的用法在《DM_SQL 语言使用手册》中详细说明。本章主要介绍一些常用的 disql 命令。

16.1 启动 disql

为了使用 disql，必须首先要启动 disql。

16.1.1 在 windows 环境中运行 disql

如果在 windows 环境中安装了达梦数据库产品，那么可以在窗口环境中运行 disql。具体方法：“开始->程序->达梦数据库->客户端->disql”。

16.1.2 在命令运行 disql

16.1.2.1 初始登录

在命令行运行 disql 是使用 disql 命令来完成的，该命令适用于任何操作系统平台。登录可选方式如下：

```
disql [ [<option>] [<logon> | /NOLOG] [<start>] ]
<option>::=-S
<logon>::=<username>[/<password>][@<server>][:<port>][#<sslpath>]
<start>::='<file_path>
```

参数

1. -S 表示设置 DISQL 界面为无提示模式，即隐藏命令的 DISQL 标识。
2. <username>[/<password>] 指定数据库的用户名和密码，密码一定要大写。当连接到本地数据时，不需要指定用户名和密码，使用默认即可。
3. <SERVER> 指定服务器的 IP 地址。
4. /NOLOG 表示启动 DISQL 而不连接到服务器。此时可以进行 DISQL 的显示设置和本地变量操作。
5. <file_path> 要执行的 sql 脚本文件绝对路径。用户可以将多条 sql 语句保存在一个文本文件中，这样当要执行这个文件中的所有 sql 语句时，用 start 命令即可，这类似于 dos 中的批处理。值得注意的是，当进入成功进入 DISQL 界面后，`<file_path>` 用法依然有效。

示例如下：

```
disql -S SYSDBA/SYSDBA@192.168.0.80:5236 'c:\a.sql
```

值得注意的是，当进入成功进入 DISQL 界面后，<start>用法依然有效。如下所示：

```
SQL> `c:\b.sql
```

6. 如果没有/NOLOG 选项必须连接服务器，不带参数的时候提示输入用户名和密码，此时的用户名用作<logon>。且连接三次失败后退出 DISQL。

16.1.2.2 远程登录

用户进入 disql 界面后，如果还想切换到其他数据库。有两种实现方式：一是使用 login 命令；二是使用 CONN 命令。连接到远程数据库，必须在服务名处使用 IP 地址或网络服务名。

16.1.2.2.1 login /logout

1. login 登录主机建立会话

直接输入 login 命令后，屏幕会提示输入登录信息：

```
服务名：
用户名：
密码：
端口号：
SSL 路径：
SSL 密码：
USBKey Pin：
```

默认输入情况下服务器名为 localhost，用户名和密码均为 SYSDBA（密码不显示），端口号为 5236。也可以在服务名后输入 IP 地址来访问远程服务器。密码一定要大写。其中，SSL 路径、SSL 密码和 USBKey Pin 用于服务器安全设置，普通用户不必设置，直接回车即可。登录成功后会显示登录时间。

当出现“SQL>”符号时，用户就可以利用 DM 提供的 SQL 语句和数据库进行交互操作了，需要注意的是，在 DISQL 中语句应以分号“;”结束。对于执行语句块，创建触发器，存储过程，函数，包，模式等时需要用“/”启动执行。

2. logout 从登录主机注销会话

logout 从登录主机注销会话。断开连接而不退出 disql。

```
logout
```

示例如下：

```
SQL>logout
```

16.1.2.2.2 CONN[ECT] /DISCONN[ECT]

1. CONN[ECT] 连接

CONN[ECT] 命令与 login 命令相似，增加的功能是，可以在命令之后直接跟 <username>[/<password>][@<server_identifier>]，或者跟<username>，然后按照 disql 提示输入<password>来连接。

```
CONN[ECT] <username>[/<password>][@<server>]
```

或者

```
CONN[ECT] <username>
```

值得注意的是，使用该命令建立新会话时，会自动断开先前会话。

示例如下：

```
SQL>CONN SYSDBA/SYSDBA@192.168.0.80
```

2. DISCONN [ECT] 断开连接

DISCONN [ECT]：断开连接而不退出 disql。

```
DISCONN [ECT]
```

示例如下：

```
SQL>DISCONN
```

16.2 设置和显示 disql 环境变量

使用 set 命令可以对当前 disql 的环境变量进行设置。并通过 show 命令来查看当前系统中环境变量的设置情况。disql 中的环境变量如下：

1. AUTO[COMMIT]{ON|OFF}
2. DEF[INE] {c(默认的变量前缀是&)|ON (默认值) |OFF}
3. ECHO {ON (默认值) |OFF}
4. FEED[BACK] {6 (默认值) |n|ON|OFF}
5. HEA[DING] {ON (默认值) |OFF}
6. SET LINESHOW {ON 默认值|OFF}
7. NEWP[AGE] {1 (默认值) |n|NONE}
8. PAGES[IZE] {14 (默认值) |n}
9. TIMI[NG] {ON (默认值) |OFF}
10. VER[IFY] {ON (默认值) |OFF}

16.2.1 SET

用于设置 disql 系统变量的命令。该命令包含许多子命令：

```
SET <system_variable> <value>
```

<system_variable><value> 可以是上述环境变量的子句之一：

可以同时 set 多个。如：Set heading on timing on。Set 出错之后，后面的 set 就不再起作用。

如下详细进行介绍：

- 1) 设置当前 session 是否对修改的数据进行自动提交

```
SQL>SET AUTO[COMMIT] {ON|OFF}
```

on 表示打开自动提交，所有执行的 sql 语句的事务将自动进行提交；

off 表示关闭自动提交，所有执行的 sql 语句的事务将由用户显式提交，为默认设置。

- 2) 是否使用 define 定义本地变量

```
SQL> SET define {c(默认的变量前缀是&)|ON (默认值) |OFF}
```

- 3) 在用 start 命令执行一个 sql 脚本时，是否显示脚本中正在执行的 SQL 语句

```
SQL> SET ECHO {ON (默认值) |OFF}
```

- 4) 是否显示当前 sql 语句查询或修改的行数

```
SQL> SET FEED[BACK] {6 (默认值) |n|ON|OFF}
```

默认只有结果大于 6 行时才显示结果的行数。如果 set feedback 1，则不管查询到多少行都返回。当为 off 时，一律不显示查询的行数。

- 5) 是否显示列标题

```
SQL> SET HEA[DING] {ON (默认值) |OFF}
```

当 set heading off 时，在每页的上面不显示列标题，而是以空白行代替。

6) lineshow 设置是否显示行号

```
set lineshow {ON (默认值) |OFF};
```

默认为每行输出打印行号。

7) 设置页与页之间的分隔

```
SQL> SET NEWP[AGE] {1 (默认值) |n|NONE}
```

当 set newpage 0 时，会在每页的开头有一个小的黑方框。

当 set newpage n 时，会在页和页之间隔着 n 个空行。

当 set newpage none 时，会在页和页之间没有任何间隔。

8) 设置一页有多少行数

```
SQL> SET PAGES[IZE] {14 (默认值) |n}
```

如果设为 0，则所有的输出内容为一页并且不显示列标题。默认值为 14。

9) 显示每个 sql 语句花费的执行时间

```
SET TIMING {ON (默认值) |OFF}
```

10) VER[IFY] 是否列出环境变量被替换前、后的控制命令文本。默认值为 ON，表示列出命令文本。

```
SET VER[IFY] { ON (默认值) |OFF}
```

示例如下：

```
SQL> set verify off
```

```
SQL> select &a from dual;
```

输入 a 的值: 3

```
行号          3
```

```
-----
```

```
1              3
```

```
SQL> set verify on
```

```
SQL> select &a from dual;
```

输入 a 的值: 3

原值 1: select &a from dual

新值 1: select 3 from dual

```
行号          3
```

```
-----
```

```
1              3
```

16.2.2 SHOW

用于显示 disql 系统变量。

```
SHOW <system_variable>
```

system_variable 请参考 set 命令中相关介绍。

可以 SHOW 多个，SHOW 出错之后，后面的还是继续 SHOW。

示例如下：

```
SQL>SHOW heading timing
```

16.3 常用的 disql 命令

16.3.1 help

显示帮助信息。

```
help <command>
```

显示相关命令的帮助信息。<command>可以输入完整的关键字，查询某一命令用法；也可以输入关键字的首字母，查询某一字母开头的所有命令用法。

示例如下：

```
SQL>help show
```

或者

```
SQL>help s
```

16.3.2 spool

将屏幕显示的内容输出到指定文件。

```
spool <'file_path'>;
```

关闭 spool 输出。

```
spool off;
```

只有关闭 spool 输出，才会在输出文件中看到输出的内容。

示例如下：

```
SQL>spool 'c:\b.sql'
```

```
SQL>select * from sysobjects;
```

```
SQL>spool off
```

先执行上述语句，然后，查看 c:\b.sql 文件。

16.3.3 host

```
host[<command>]
```

使用该命令可以不用退出 disql 就能执行操作系统命令。如果单独执行 host，则能够直接从 disql 界面切换到操作系统。

示例如下：

```
SQL> host disql.exe
```

16.3.4 describe

获取表或视图的结构描述。

```
desc[ribe] <table>|<view>;
```

将显示表或者视图的结构信息，包括各个列的数据类型等信息。

示例如下：

```
SQL>desc sysobjects;
```

16.3.5 定义本地变量

定义本地变量的方法有两个：一是 `define`；二是 `column`。

1. define

用来定义一个本地变量的替代变量，然后对该变量赋一个 `CHAR` 类型的值；或者输出变量的值和类型。

```
DEF[INE] [<VARIABLE= text >|< VARIABLE >]
```

`DEF[INE] VARIABLE = text` 申明一个变量，如果该变量存在，则重新赋值，否则新生成一个变量，并进行赋值；

`DEF[INE] VARIABLE` 如果该变量存在，则输出特定 `VARIABLE` 的值和类型，否则报错；

`DEF[INE]` 输出 `disql` 中所有的变量的值和类型；

该命令定义的替代变量在当前的 `disql` 环境和 `NOLOG` 环境中均可以起作用。

当使用该命令定义变量时，如果变量值包含空格或区分大小写，则用引号引注。另外，使用“`define 变量名`”可以检查变量是否已经定义。

`Define` 定义的变量会保存在环境 `disql` 环境中，可以在 `sql` 语句中使用。默认变量前缀树`&`。

示例如下：

```
SQL>DEF VAR=DESK;
```

```
SQL>select * from dual where id=&VAR;
```

如果 `var` 没有定义，会提示输入变量的值；

没有定义的 `var` 不会保存在 `disql` 环境中。

关闭变量替换：

```
SET define off
```

2. column

定义一个本地列或表达式。

```
COL[UMN] [<column | expr> [<option>]]
```

```
<option> ::= NEW_VALUE variable
```

`COL[UMN]`：列举出所有的 `COLUMN` 变量信息；

`COL[UMN] column | expr`：列举出某个 `column` 或 `expr`，如果存在，则输出信息，否则报错；

`COL[UMN] column | expr option`：option 目前仅支持 `NEW_VALUE`，表示该 `column|expr` 的值，同时作为变量存在。但如果该变量未赋值，通过 `DEFINE` 查询时，不会显示该变量。

查询结果的最后一个值赋给本地变量。

示例如下：

```
SQL>COLUMN CVAR NEW_VALUE DVAR
```

```
SQL>SELECT CUSTOMER_ID CVAR FROM CUSTOMER;
```

	VAR

1	0
2	1
3	2
4	3
5	4
6	5
7	6
8	7

```

          9          8
         10         9
         11        10
SQL>DEFINE DVAR
      DEFINE DVAR  = "10" (INT)

```

通过如下方式设置将变量 **VARIABLE** 与列名 **column** 之间的关联，默认值为 **ON**，表示已关联。

```
COL[UMN] column {OFF|ON(默认值)}
```

示例如下：

```
SQL>COLUMN CVAR OFF
```

16.3.6 EDIT

对当前的输入进行编辑。

```
ED[IT][<file_name>]
```

编辑指定文件，如果指定文件不存在，则创建该文件。如果直接输入 **EDIT**，打开默认编辑器，Windows 系统中默认是 **notepad.exe**，把缓冲区中最后一条 **SQL** 语句调入 **dmsql.buf** 文件中进行编辑。

该命令可以将 **disql** 缓冲区中的内容复制到一个名为 **dmsql.buf** 的文件中，然后系统操作系统中默认的编辑器打开这个文件，并且文件内容能够进行编辑。对于 windows 系统来讲，使用 **notepad.exe** 进行编辑，而 linux 或 unix 时，使用 **vi** 进行编辑。也可以编辑指定的其他文件。

示例如下：

```
SQL>edit a.sql
或
SQL>edit
```

16.3.7 explain

用 **explain** 命令来查看查询语句的执行计划。

```
explain <sql_clause>
```

<sql_clause>请参考《DM_SQL 语言使用手册》

示例如下：

```
SQL>explain select count(*) from sysobjects;
```

第三部分 高级数据库管理

第 17 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。

17.1 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。

17.1.1 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;
```

17.1.2 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。

17.1.3 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;
```

17.1.4 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;
```


17.1.5 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;
```

17.1.6 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。例如将 bookshop 表空间绑定到 KEEP 缓冲区。

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

17.1.7 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;
```

17.2 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。

17.2.1 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;
```

17.2.2 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;
```

17.2.3 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;
```

17.2.4 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';
```

17.3 管理重做日志文件

17.3.1 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;
```

17.3.2 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;
```

17.4 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 17.2 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:\dmctl 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:\dmctl
```

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。

17.5 管理控制文件

17.5.1 修改控制文件路径

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 dmctl 在数据目录下。例如可以把 dmctl 文件复制到 D 盘下，同时修改 dm.ini 中 CTL_PATH = D:\dmctl。

17.5.2 添加控制文件

可把已有的 dmctl 复制到其他路径下，然后在 dm.ini 中增加 CTL_PATH 配置参数来指定新添加的控制文件。例如可以把 dmctl 文件复制到 D 盘下，同时在 dm.ini 文件中增加 CTL_PATH = D:\dmctl，这样 DM 就会同时维护两个控制文件。控制文件的文件名也是可修改的。

第 18 章 快速数据加载

dmfldr 命令行工具是达梦数据库提供的能够快速将文本数据载入 DM 数据库的一种数据载入方式。用户通过使用 dmfldr 工具能够把按照一定格式排序的文本数据以简单、快速、高效的方式载入到达梦数据库中。

18.1 使用 dmfldr 加载数据

dmfldr 的使用必须指定必要的执行参数，其调用格式为：

```
dmfldr keyword=value [keyword=value ...]
```

示例如下：

```
dmfldr userid=SYSDBA/SYSDBA@LOCALHOST:5236#ssl_path@ssl_pwd control='fldr.ctl' log='fldr.log'
badfile='fldr.bad' errors=999 load=10000 rows=500 character_code='SINGLE_BYTE'
```

如果需要获取帮助信息，可以调用 dmfldr help，屏幕将显示如下信息：

格式: DMFLDR KEYWORD=value

例程: DMFLDR SYSDBA/SYSDBA CONTROL='c:\fldr.ctl'

USERID 必须是命令行中的第一个参数

CONTROL 必须是命令行中的第二个参数

字符串类型参数必须以引号封闭

关键字 说明（默认）

USERID	用户名/口令 格式:USER/PWD@SERVER:PORT
CONTROL	控制文件，字符串类型
LOG	日志文件，字符串类型 (dmfldr.log)
BADFILE	错误数据记录文件，字符串类型 (dmfldr.bad)
SKIP	初始忽略逻辑行数 (0)
LOAD	需要装载的行数 (ALL)
ROWS	提交频次 (50000)
READSIZE	读取缓冲区大小 (16M)
DIRECT	是否使用快速方式装载 (TRUE)
SET_IDENTITY	是否插入自增列 (FALSE)
SORTED	数据是否已按照聚集索引排序 (FALSE)
INDEX_OPTION	索引选项 (1) 1 不刷新二级索引，数据按照索引先排序，装载完后再将排序的数据插入索引 2 不刷新二级索引，数据装载完成后重建所有二级索引
GEN_LOG	服务器是否生成日志 (TRUE)
ERRORS	允许的最大数据错误数 (100)
CHARACTER_CODE	字符编码，字符串类型 (GBK)
MODE	装载方式，字符串类型 IN 表示载入，OUT 表示载出 (IN)
MPP_LOCAL	单节点 MPP 数据 (FALSE)

HBUF_BLOCK_NUM	指定导入水平分区缓冲区块的个数
HELP	打印帮助信息

dmfldr 工具的参数及说明

1. userid

用于指定登录数据库的信息，包含用户名、密码、服务器地址、端口号和安全证书存放路径。参数值格式为：用户名/密码[@主机地址[:端口号]][#安全证书存放路径@安全证书密码]]。如果用户使用默认端口号登录则端口号可以不设置，如果登录的是本地的服务器则主机地址可以省略。此参数为必选参数且必须位于参数位置的第一个。

2. control

控制文件的路径，字符串类型。控制文件用于指定数据文件中数据的格式，dmfldr 根据控制文件指定的格式来解析数据文件。控制文件格式详见下一节的控制文件说明。此参数为必选参数且必须位于参数位置的第二个。

3. log

装载的日志文件路径，字符串类型。默认日志文件名为 fldr.log。文件记录了装载过程中的装载信息和错误信息以及统计信息。用户也可以通过设置控制文件中的 OPTION 选项来指定日志路径。如果参数及 OPTION 中同时指定了日志路径则其将以 OPTION 中指定的路径为最终路径。此参数为可选参数。

4. badfile

记录错误数据的文件路径，字符串类型。默认的错误文件名为 fldr.bad。文件记录的信息为数据文件中存在格式错误的行数据以及转换出错的行数据。用户也可以通过设置控制文件中的 OPTION 选项来指定错误数据文件的路径，同时也可以控制文件的 LOAD 节点中指定错误数据文件的路径。badfile 选项的最终值的优先选择顺序为 LOAD 节点选项，OPTION 选项，参数选项，用户可以同时对三种设置方式中的一个或多个设置，但最终的值只取一个。此参数为可选参数，作用于 mode 为 IN 的情况下，对于 mode 为 OUT 无效。

5. skip

跳过数据文件起始的逻辑行数，整形数值。默认的跳过起始行数为 0 行。如果用户指定了多个文件，且起始文件中的行数不足 skip 所指定的行数，则 dmfldr 工具会扫描下一个文件直至累加的行数等于 skip 所设置的行数或者所有文件都已扫描结束。用户也可以通过设置控制文件中的 OPTION 选项来指定 skip 参数。对于用户设置 skip 的优先顺选择序为 OPTION 选项，参数选项。此参数为可选参数，作用于 mode 为 IN 的情况下，对于 mode 为 OUT 无效。

6. load

装载的最大行数，整形数值。默认的最大装载行数为数据文件中的所有行数。load 指定的值是不包括 skip 指定的跳过的行数。用户也可以通过设置控制文件中的 OPTION 选项来设置 load 的值。load 参数值的优先选择顺序为 OPTION 选项，参数选项。此参数为可选参数，作用于 mode 为 IN 的情况下，对于 mode 为 OUT 无效。

7. rows

每次提交的行数，整形数值。默认的提交行数为 50000 行。提交行数的值表示提交到服务器的行数，并不一定代表按照数据文件中的数据顺序的行数。用户可以根据实际情况调整每次提交的行数，以达到性能的最佳点。用户也可以通过设置控制文件中的 OPTION 选项

来设置 rows 值。rows 参数值的优先选择顺序为 **OPTION** 选项，参数选项。此参数为可选参数，作用于 mode 为 IN 的情况下，对于 mode 为 OUT 无效。

8. readsize

读取缓冲区的大小，整形数值。默认大小为 16，单位为兆（M）。readsize 设置的大小决定了每次从文件读取数据块的大小，用户可以根据自身环境的情况调整读取缓冲区的大小，以达到性能的最佳点。用户也可以通过设置控制文件中的 **OPTION** 选项来设置 readsize 值。readsize 参数值的优先选择顺序为 **OPTION** 选项，参数选项。此参数为可选参数，作用于 mode 为 IN 的情况下，对于 mode 为 OUT 无效。

9. direct

装载方式，布尔值。默认为 **TRUE**。direct 指定了装载的方式，当为 **TRUE** 时，dmfldr 选择快速的载入模式，通过数据的转换和数据的封装直接对 B 树进行操作，省去了普通插入方式下各个操作符之间的跳转，提升了装载的效率，但对于约束的检查等由用户保证，dmfldr 将不处理有约束冲突的数据。当为 **FALSE** 时，dmfldr 选择普通的插入方式装载数据，可以保证数据的正确性和约束的有效性，效率比前者要低。用户也可以通过设置控制文件中的 **OPTION** 选项来设置 direct 值。direct 参数值的优先选择顺序为 **OPTION** 选项，参数选项。此参数为可选参数，作用于 mode 为 IN 的情况下，对于 mode 为 OUT 无效。

10. set_identity

设置自增列选项，布尔值。默认为 **FALSE**。如果指定 set_identity 选项值为 **TRUE**，则 dmfldr 将把从数据文件中读取的自增列值作为目标值插入数据库表中，用户应当保证每一行的自增列的值符合自增列的规则。如果 set_identity 选项值设置为 **FALSE**，则 dmfldr 将忽略数据文件中对应自增列的值，服务器将自动生成自增列的值插入每一行的对应列。用户也可以通过设置控制文件中的 **OPTION** 选项来设置 set_identity 的值。set_identity 参数值的优先选择顺序为 **OPTION** 选项，参数选项。此参数为可选参数，作用于 mode 为 IN 的情况下，对于 mode 为 OUT 无效。

11. sorted

数据是否已经按照聚集索引排序，布尔值。默认为 **FALSE**。如果设置为 **TRUE**，则用户必须保证数据已按照聚集索引排序完成，并且如果表中存在数据，需要插入的数据索引值要比表中数据的索引值大，服务器在做插入操作时顺序进行插入。如果设置为 **FALSE**，则服务器对于每条记录进行定位插入。用户也可以通过设置控制文件中的 **OPTION** 选项来设置 sorted 的值。sorted 参数值的优先选择顺序为 **OPTION** 选项，参数选项。此参数为可选参数，作用于 mode 为 IN 且 direct 为 **TRUE** 的情况下，对于其他情况此参数无效。

12. index_option

索引的设置选项，整形数值。默认为 1。index_option 的可选项有 1 和 2。1 代表服务器装载数据时先不刷新二级索引，而是将新数据按照索引预先排序，在装载完成后，再将排好序的数据插入索引；2 代表服务器在快速装载过程中不刷新二级索引数据，只在装载完成时重建所有二级索引。用户也可以通过设置控制文件中的 **OPTION** 选项来设置 index_option 的值。index_option 参数值的优先选择顺序为 **OPTION** 选项，参数选项。此参数为可选参数，作用于 mode 为 IN 且 direct 为 **TRUE** 的情况下，对于其他情况此参数无效。

13. errors

最大的容错个数，整形数值。默认为 100。当 dmfldr 在插入过程中出现错误的个数超过了 errors 所设置的数目，则 dmfldr 会停止载入，当前时间点的所有正确数据将会被提交到

服务器。如果载入过程中不允许出现错误则可以设置 `errors` 为 0，如果允许所有的错误出现，则可以设置 `errors` 为一个非常大的数。需要说明的是 `errors` 所统计的错误将不包含数据格式的误差而导致的行数据错误，而只统计在数据转换和数据插入过程中所产生的数据错误。用户也可以通过设置控制文件中的 `OPTION` 选项来设置 `errors` 的值。`errors` 参数值的优先选择顺序为 `OPTION` 选项，参数选项。此参数为可选参数，作用于 `mode` 为 `IN` 的情况下，对于 `mode` 为 `OUT` 无效。

14. `character_code`

数据文件中数据的编码格式，字符串数值。默认为 `GBK`。`character_code` 的可选项有 `GBK`，`UTF-8`，`SINGLE_BYTE` 三种，`GBK` 对应的编码为中文编码，`UTF-8` 对应的为 `UTF-8` 的国际编码，`SINGLE_BYTE` 对应的为数据文件中的数据都是单字节的字符。在 `mode` 为 `IN` 的情况下，指定编码格式为 `SINGLE_BYTE` 时，`dmfldr` 将不做字符完整性的检查，按单字节顺序读取每个字符。指定编码格式为 `GBK` 或者 `UTF-8` 时，`dmfldr` 将对每一个字符做字符的完整性检查，确保数据的正确性。用户在使用 `dmfldr` 时可以根据不同的数据文件调整编码的格式确保装载的正确性，同时如果可以确保数据文件中的数据为单字节字符，则指定 `SINGLE_BYTE` 的载入效率将优于指定 `GBK` 和 `UTF-8` 的情况。用户也可以通过指定控制文件中的 `OPTION` 选项来设置 `character_code` 的值。`character_code` 参数值的优先选择顺序为 `OPTION` 选项，参数选项。此参数为可选参数。

15. `gen_log`

服务器是否生成日志，布尔值。默认为 `TRUE`。用户也可以通过设置控制文件中的 `OPTION` 选项来设置 `gen_log` 的值。`gen_log` 参数值的优先选择顺序为 `OPTION` 选项，参数选项。此参数为可选参数，作用于 `mode` 为 `IN` 且 `direct` 为 `TRUE` 的情况下，对于其他情况此参数无效。

16. `mode`

`dmfldr` 的装载模式，字符串数值。默认值为 `IN`。`mode` 的可选项有 `IN` 和 `OUT` 两种。`IN` 对应的是从数据文件中将数据装载入数据库，控制文件的格式对应为数据文件中现有数据的格式；`OUT` 对应的是从数据库中将数据装出到数据文件，控制文件所指定的格式为最终数据存放的格式，需要说明的是在 `OUT` 模式下，如果指定了多个数据文件，则 `dmfldr` 最终只会将数据写入第一个数据文件，如果数据文件已存在，`dmfldr` 将报错。`OUT` 模式下的控制文件与 `IN` 模式下的控制文件格式相同，用户可以通过使用同一个文件进行载入和载出数据。用户也可以通过指定控制文件中的 `OPTION` 选项来设置 `mode` 的值。`mode` 参数值的优先选择顺序为 `OPTION` 选项，参数选项。此参数为可选参数。

17. `lob_directory`

指明 `dmfldr` 工具使用的大字段数据存放的目录，字符串类型。当 `mode` 为 `out` 时，`dmfldr` 生成大字段对应的数据文件 `dmfldr.lob` 存放于 `lob_directory` 指定的目录，如果未指定 `lob_directory` 则 `dmfldr.lob` 文件存放于指定的导出数据文件同一目录。对于 `mode` 为 `out` 的情况，此参数为可选参数。当 `mode` 为 `in` 时，如果有大字段数据文件 `dmfldr.lob`，则用户必须先把此文件传送到服务器本地目录，然后使用 `lob_directory` 指明存放目录。对于 `mode` 为 `in` 且装载的数据包含大字段数据文件 `dmfldr.lob`，则此参数为必选参数。`lob_directory` 参数值的优先顺序为 `OPTION` 选项，参数选项。此参数作用于 `mode` 为 `IN` 且 `direct` 为 `TRUE` 且非 `mpp` 方式导入或者 `mode` 为 `out` 的情况下，对于其他情况此参数无效。

18. `mpp_local`

指明是否是 `mpp` 模式下的单节点数据，布尔值。默认值为 `FALSE`。当服务器对应的是

mpp 模式时，如果所要装载的数据是对应于单节点上的，不需要服务器分发，则设置该参数为 TRUE，dmfldr 将向指定的节点插入数据。用户也可以通过设置控制文件中的 OPTION 选项来设置 mpp_local 的值。mpp_local 参数值的优先顺序为 OPTION 选项，参数选项。此参数为可选参数，作用于 mode 为 IN 且 direct 为 TRUE 并且服务器为 mpp 模式的情况下，对于其他情况此参数无效。

19. hbuf_block_number

指定水平分区表导入缓冲区块数目。其值为整数类型，默认为 50。该选项仅在导入表为水平分区表时起作用，用于指定导入过程中使用缓冲区块的大小。当用户导入的表的水平分区数超过默认值时，可根据系统资源扩大缓冲区块的数目，从而提高导入效率。

20. help

获取帮助信息。使用 dmfldr 工具，用户必须指明连接服务器所要使用的用户名及登录密码 (userid)，关键字 userid 可以省略，此项必须位于参数的第一项；还必须指定数据格式控制文件用于 dmfldr 对于文本文件的解析(control)，此项必须位于参数的第二项。所有字符串类型值的参数必须要以单引号封闭。

18.2 dmfldr 的控制文件

dmfldr 的控制文件语法如下：

```
[OPTIONS(  
  <id>=<value>  
  .....  
)]  
LOAD [DATA]  
INFILE [LIST] <path_name> [<row_term_option>] [,<path_name>  [<row_term_option>]]  
[BADFILE <path_name>]  
[APPEND|REPLACE|INSERT]  
INTO TABLE [<schema>.<tablename>  
[FIELDS <delimiter>]  
[<coldef_option>]  
<id> ::=参数  
<value> ::=值  
<path_name> ::=文件地址  
<path_name_list> ::=含文件列表文件地址  
<row_term_option> ::=STR [X] '<value>'  
<schema> ::=模式名  
<tablename> ::=表名  
<delimiter> ::=<单字节字符>  
<coldef_option> ::=(<col_def>{ ,<col_def>})  
<col_def> ::=<col_id> [<property_option>] [<fmt_option>] [<term_option>] [<enclosed_option>]  
<col_id> ::=列名  
<property_option> ::=FILLER|ROWNUMBER|NULL|<position_condition>  
<position_condition> ::=POSITION(<value>:<value>)
```



```
<fmt_option> ::= DATE FORMAT '<时间日期格式串>'
<term_option> ::= TERMINATED [BY] <wx_option>
<wx_option> ::= WHITESPACE|<单字节字符>
<enclosed_option> ::= [OPTIONALLY] ENCLOSE [BY] <单字节字符>
```

说明：

1. dmfldr 在处理数据文件中换行符时 windows 默认为 0x0D0A(\r\n)，非 windows 默认为 0x0A(\n)，用户应该根据现有的数据文件中的换行符做相应的调整，对应选项为 <row_term_option>，指定的<value>值为二进制的字符串值则需要指明[X]选项，反之则不需要。
2. 关于列分隔符，用户应当指定 FIELDS 或者 coldef_option 中的至少一种。
3. 关于 LIST 选项，INFILE 使用 LIST 选项时，表明实际的数据文件路径存储在 INFILE 指定的文件中，该文件可以存储多个实际的数据文件路径，使用逗号或者换行分割。
4. 关于 OPTIONS 选项，该选项支持命令行参数中除 userid，control，help 以外的所有参数的指定，每个参数值对使用空格或者换行分割。

下面是一个 dmfldr 控制文件的例子。

```
OPTIONS
(
    SKIP = 0
    ROWS = 50000
    DIRECT = TRUE
    INDEX_OPTION = 2
)
LOAD DATA
INFILE 'test.dat' STR X '0D0A'
BADFILE 'test.bad'
INTO TABLE test
FIELDS '|'
(
    F1,
    F2 DATE FORMAT 'YYYY-MM-DD',
    F3 NULL,
    F4 TERMINATED BY WHITESPACE ENCLOSED BY '('
)
```

18.3 性能调整

针对不同装载环境，用户可以通过指定不同的参数选项组合来提升装载的速度。具体可调整装载效率的参数如下：

1. rows：提交数据到服务器的频率，默认为 50000 行一次提交。用户可以根据服务器设置的 buffer 的大小调整提交的频率，以达到载入的最佳速率；
2. readsize：读取缓冲区的大小，默认为 16M。用户可以根据内存大小通过调整读取缓冲区的大小来调整读取数据文件的磁盘 IO 次数；
3. character_code：数据文件的字符编码格式，默认为 GBK。针对不同的编码方式，dmfldr 会按照编码的格式检查数据中字符的完整性。如果用户数据只涉及到单字节

的字符,用户可以通过设置此参数为 **SINGLE_BYTE**,从而取消字符完整性的检查,达到提升装载速度的目的。

如果载入方式 **direct** 为 **TRUE** 的情况下,调整以下参数也能达到效率的提升。

1. **sorted**: 数据是否已按照聚集索引排序(默认为 **FALSE**)。如果用户数据已经是按照聚集索引排序的且值都大于已存在的索引列的值,则用户可以设置此参数为 **TRUE**,从而使得服务器不需要做排序操作,提升装载效率;
2. **index_option**: 在表为空表的情况下,设置为 1,可以提升装载效率

第 19 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。

19.1 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。

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。

19.2 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，提供以下三种分

区方式:

1. 范围水平分区: 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一个分区上;
2. 哈希水平分区: 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. 列表水平分区: 通过指定表中的某些列的离散值集, 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', 'T', 'P')放在另一个分区, 以此类推。

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。

19.3 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。

19.3.1 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 19.1。

表 19.1 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 SLEELCT 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);
```

19.3.2 创建列表分区表

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，可以考虑使用列表分区。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，则适合于采用列表分区的方法。

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')
);

```

在创建列表分区时，通过“**PARTITION BY LIST**”子句指定对表进行列表分区，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，列表分区的分区键必须唯一。

19.3.3 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列表分区。

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，这时很难对表进行列表分区。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);
```

19.4 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、聚簇索引和函数索引。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。

例如，在 sales 表上的 saledate 上建立索引。

```
CREATE INDEX ind_sales_saledate ON sales(saledate);  
CREATE UNIQUE INDEX ind_sales_city ON sale(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。

19.5 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）。

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。

19.5.1 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。

对于列表分区，增加分区包含的离散值不能已存在于某个分区中。例如，为列表分区表 **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);
```

只能对范围分区和列表分区增加分区，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。

19.5.2 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;
```

只能对范围分区和列表分区进行删除分区，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。

19.5.3 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。

仅范围分区和列表分区支持交换分区，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（相同的列和索引），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。

19.5.4 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。

19.5.5 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。

19.6 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、TEXT、ROWID、BIT、BINARY、VARBINARY、时间间隔等类型和用户自定义类型作为分区列；
2. 范围分区和哈希分区的分区键可以是多个，但列表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. 水平分区表的分区数的上限是 65535;
7. 不允许对分区子表执行任何 DDL 操作;
8. 对于哈希分区, 除了表名重命名外, 不允许其他 ALTER TABLE 操作;
9. 对于范围分区和列表分区, 除了上一节所述的分区表修改操作, 不允许其他 ALTER TABLE 操作;
10. 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。列表分区增加分区值不能存在于其他已存在分区;
11. 当分区数仅剩一个时, 不允许删除分区;
12. 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;
13. 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;
14. 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;
15. 水平分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;
16. 不能对水平分区表建立唯一函数索引和全文索引;
17. 不能对分区子表单独建立索引;
18. 在分区表上执行更新分区键, 不允许更新后数据导致跨分区的移动, 即不能发生行迁移;
19. 不能在分区语句的 STORAGE 子句中指定 BRANCH 选项;
20. 不允许引用水平分区子表作为外键约束。

19.7 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的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。

19.8 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 **ALTER TABLE RENAME** 表名（主表或子表），其他均报错返回；
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。

第 20 章 管理列存储表

20.1 什么是列存储表

列存储表是相对普通的行存储表而言的，它们主要的不同在于列存储表的每一个列都是存储在一起的，而不是以记录为单位存储，所有行的同一列存储在一起。图 20.1 表示的是行存储和列存储的存储对比。其对应的基表为表 20.1。

表 20.1 基表

SYSCLASSES	268435457
SYSCOLUMNS	268435458
SYSCONS	268435459
SYSDUAL	268435460
SYSGRANTS	268435461

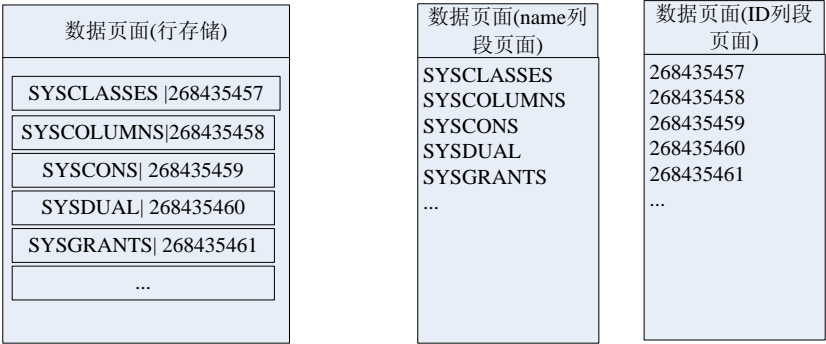


图 20.1 行列存储对比图

从上图可以看出，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；而对于列存储表，每一个段只存储一个列的数据，而且一个指定的页面中存储的都是某一个列的连续数据。

在具体存储实现方式上就是每一个列的数据存储在一个段中，并且每一个列分成若干个区，区是由物理上的簇构成，大小可以通过创建表时在语句中指定。每一个区对应一个区描述项，用来管这个区中的数据，可以通过区描述项找到这个区的具体位置；区描述项中还可以记录区内数据的最大值与最小值，在进行带条件扫描时，可以借助此信息减少不必要的 IO 操作。

为尽可能提高列数据存储的连续性，区描述页单独存放在另一个数据段中。

图 20.2 表示的是列存储表中某一个列的两个段的存储结构示意图。

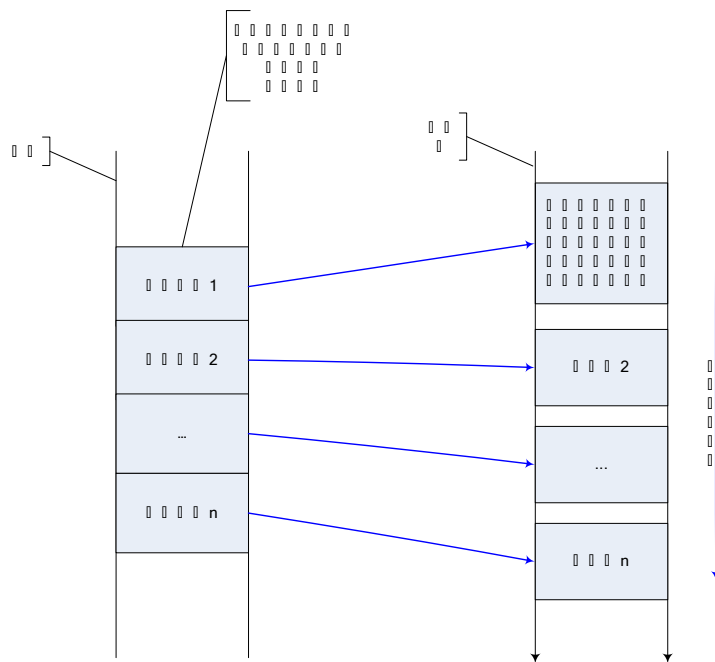


图 20.2 列存储结构示意图

列存储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；
3. 条件扫描借助数据区过滤，可以进一步减少 IO，提高扫描效率。

列存储表存在上面几个优点，当然它也有它的缺点：

1. 列存储封锁粒度较大，且不支持多版本并发控制；
2. 列存储表的插入、删除与更新操作处理都不及行存储表方便；
3. 列存储表不支持建立除 NULL、NOT NULL 和 UNIQUE 外的任何约束，也不允许建立二级索引，仅可以在建表的时候指定主键。

总之，列存储表主要针对海量数据的扫描分析，并不适用于常规的联机事务处理。

20.2 创建列存储表

在 DBA 确定自己要使用列存储表的时候，就首先需要在你的模式中创建新表，创建一个列存储表需要有 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，另外，表的拥有者还可以定义该表的表空间的限额，或通过 SQL 语句中指定 UNLIMITED SPACELIMIT 或者用默认方式 CREATE VERTICAL TABLE 创建表。

列存储表的数据也可以做压缩处理，由于数据的相似性，压缩率一般会比行存储的高。在创建表时，可以指定某一个列的存储属性，存储属性包括如下几个方面：

1. 列区大小（簇的个数）

区大小可以通过设置列的存储属性来指定，也可以通过表的存储属性来指定，两者同时指定时列存储属性优先，如果都没有指定，则采用默认值 10 个簇。

例如创建列存储表 test，name 列的区大小为 5，其它默认。

```
CREATE VERTICAL TABLE test(name VARCHAR STORAGE(SECTION (5)), sno INT);
```

创建列存储表 **test**，所有列的区大小都为 5。

```
CREATE VERTICAL TABLE test(name VARCHAR , sno INT) STORAGE (SECTION (5));
```

创建列存储表 **test**，**name** 列的区大小为 10，其它列的区大小都为 5。

```
CREATE VERTICAL TABLE test(name VARCHAR STORAGE(SECTION (10)), sno INT) STORAGE (SECTION (5));
```

创建列存储表 **test**，所有列都采用默认值。

```
CREATE VERTICAL 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。

例如创建列存储表 **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 VERTICAL TABLE test(name VARCHAR STORAGE (STAT NONE), sno INT);
```

创建列存储表 **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 VERTICAL TABLE test(name VARCHAR, sno INT) STORAGE (STAT NONE);
```

3. 列所属的表空间（列存储表的列可以单独指定表空间）

例如创建列存储表 **test**，通过列存储属性指定列所在的表空间，不指定则存储于表所在的表空间内，下面的语句指定列 **name** 对应的表空间为 **name_tablespace**，列 **sno** 和 **test** 表定义处于同一个表空间。

```
CREATE VERTICAL TABLE test(name VARCHAR STORAGE (ON name_tablespace), sno INT);
```

下面是一个综合的创建一个列存储表的例子：

```
CREATE VERTICAL TABLE orders
(
    o_orderkey          INT STORAGE(SECTION (15)),
    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(SECTION(20), STAT NONE)
)COMPRESS;
```

在这个例子中，定义了九个列，但将每个列都采取了压缩存储方式。 **o_orderkey** 列的存储属性是指定了区大小为 15 个簇，**o_comment** 列指定的区大小为 20 个簇，同时指定了不做统计信息的存储，因为这个列 **DBA** 会认为不作为一个查询条件列，其余的列都会被指定为默认值，区大小为 10，都会做统计信息存储。

20.3 维护列存储表

列存储表像行存储表一样，也可以做增删改查操作，但删除及更新操作效率比行存储低一些，所以在列存储中不宜做频繁的删除及更新操作。

20.4 查看有关列存储表的信息

1. 表定义

对一个列存储表，DBA 可以通过 `SELECT 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. 空间使用情况

列存储表支持压缩，并且压缩率是很高的，可以通过一些系统函数来查询压缩后或者未压缩数据已经使用的空间大小，来做一些对比等

第 21 章 管理 HFS 表

21.1 什么是 HFS 表

HFS（全称 Huge File System）是达梦数据库实现的针对海量数据进行分析的一种高效、简单的列存储表，它仍属于列存储的范畴。表面上看与第 20 章《管理列存储表》中的列存储表相似，只是，在底层存储结构上存在很大的不同。

HFS 表是建立在自己特有的表空间 HTS（全称 HUGE TABLESPACE）上的。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 HTS 相当于一个简单的文件系统，创建一个 HTS，其实就是创建一个空的目录（系统中有一个默认 HTS，目录名为 HMAIN）。在创建一个 HFS 表之后，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，文件系统结构如下图所示：

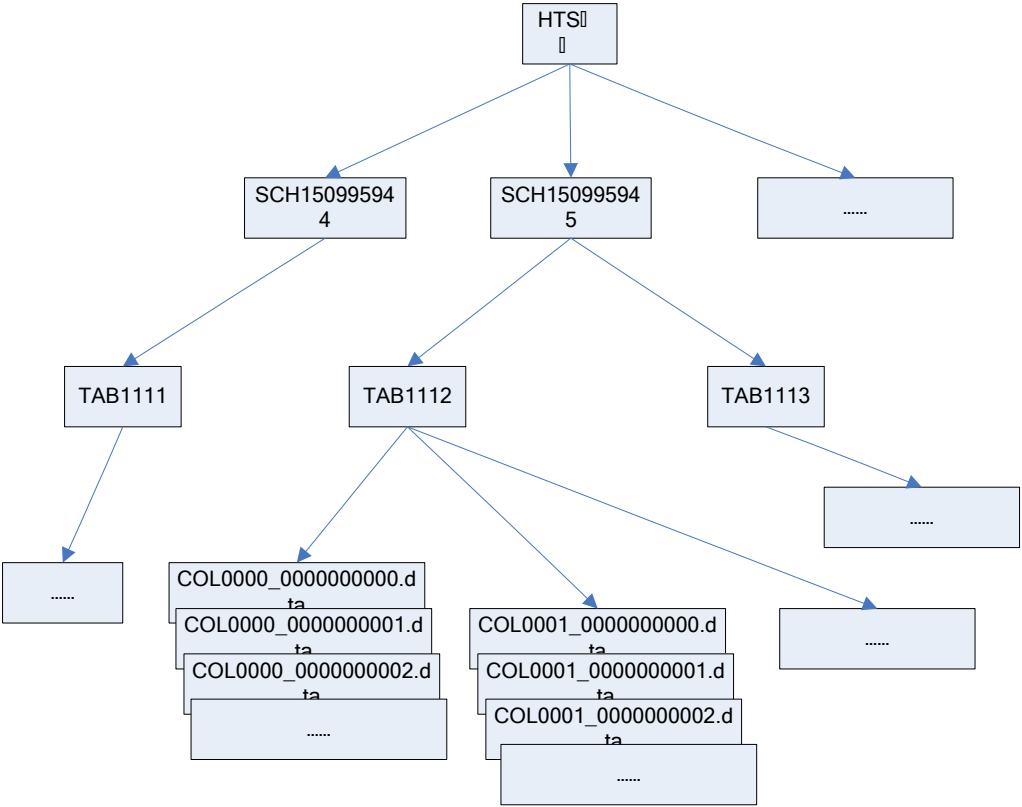


图 21.1 HFS 结构示意图

从上图可以看出，在 HTS 目录下成功创建 HFS 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+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+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，在新创建表时，每一个列对应的一个以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21.1中，0000表示第1列，0001表示第2列……；0000000000表示第1个文件，0000000001表示第二个文件……。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通过创建表时指定的，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。一个区内，存储的仅仅是数据而已，对于NULLABLE属性的列，存储的还有相应的NULL标志。每一个区的开始位置及长度在文件内都是4K对齐的。

对于一个HFS表，相应的还配备一个辅助表来管理其数据。因为在上面介绍的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，该辅助表是在创建HFS表时系统自动创建的，表名为“TABLENAME\$AUX”，TABLENAME为用户创建的HFS表名，辅助表中每一条记录对应文件中的一个数据区，辅助表包括下面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. MAX_VAL: 表示这个区中的最大值，精确值；
11. MIN_VAL: 表示这个区中的最小值，精确值；
12. SUM_VAL: 表示这个区中所有值的和，精确值；
13. CPR_FLAG: 表示这个区是否压缩；
14. ENC_FLAG: 表示这个区是否加密，该标志暂未启用；
15. CHKSUM: 用来较验的，该功能暂未启用。

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簇关键字。

HFS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. 对HFS表进行修改时，不写日志，不通过BUFFER缓存，直接操纵文件，速度快。

HFS表存在上面几个优点，当然它也有它的缺点：

4. HFS表操作时封锁粒度较大，且不支持多版本并发控制；

5. HFS 表的插入、删除与更新操作处理都不能进行回滚;
 6. HFS 表不支持建立除 NULL、NOT NULL 和 UNIQUE 外的任何约束, 也不允许建立二级索引, 仅可以在建表的时候指定主键;
 7. HFS 表不支持事务, 没有事务的特性;
 8. 不支持 SPACE LIMIT (空间限制);
- 总之, HFS 表主要针对海量数据的扫描分析, 并不适用于常规的联机事务处理。

21.2 创建 HFS 表

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

在创建表时, 可以指定表的存储属性, 存储属性包括如下几个方面:

4. 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 HFS 表 test, 指定区的大小为 5, 其它默认。

```
CREATE HUGE TABLE test(name VARCHAR, sno INT) STORAGE(SECTION (5));
```

例如, 创建 HFS 表 test, 所有列都采用默认值。

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

5. 是否记录区统计信息, 即在修改时是否做数据的统计

此处和列存储表完全相同。详情请参考《20.2 节 创建列存储表》相关内容。

6. 所属的表空间

创建 HFS 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。HFS 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)
```

7. 文件大小

创建 HFS 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));
```

8. 日志属性

HFS 表在操作过程中有可能会系统崩溃或者断电等问题, 因为修改时采取的是直接写的策略, 所以有可能会数据不一致的问题。为了保证数据的一致性, 在操作时可以适当的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。系统中主要提供三种方案：

1. **LOG NULL**：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。

示例如下：

例：CREATE HUGE TABLE test(name VARCHAR, sno INT) LOG ALL;

下面是一个综合的创建 HFS 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), on HTS_NAME) log all;
```

在这个例子中，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，做完整镜像，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。

21.3 维护列存储表

HFS 表像行存储表一样，也可以做增删改查操作，操作方式是完全一样的，但删除及更新操作效率可能比行存储低一些，所以在列存储中不宜做频繁的删除及更新操作。因此，HFS 比较适合做分析型表的存储。

同时由于其没有事务性，在操作过程中如果崩溃或者断电时，其恢复的程度或者完整性是根据在创建表时指定的日志属性决定的。如果定义为 LOG NULL，有可能造成数据的不完整，需要手动修复；如果定义为 LOG LAST，有可能造成操作的不完整；如果定义为 LOG ALL，则不会有任何不完整的问题。

21.4 查看有关 HFS 表的信息

1. 表定义

对一个列存储表，用户可以通过 `SELECT 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

HFS 表有一个很好的特点就是有一个辅助表，辅助表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，用户可以通过观察辅助表中的信息对表进行一些相应的操作。

第 22 章 管理 LIST 表

22.1 什么是 LIST 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。LIST 表就是基于上述两个理由而提出的。

简单地说，LIST 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 LIST 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 LIST 表则采用一种“扁平 B 树”方式存储，结构如图 22.1 所示。

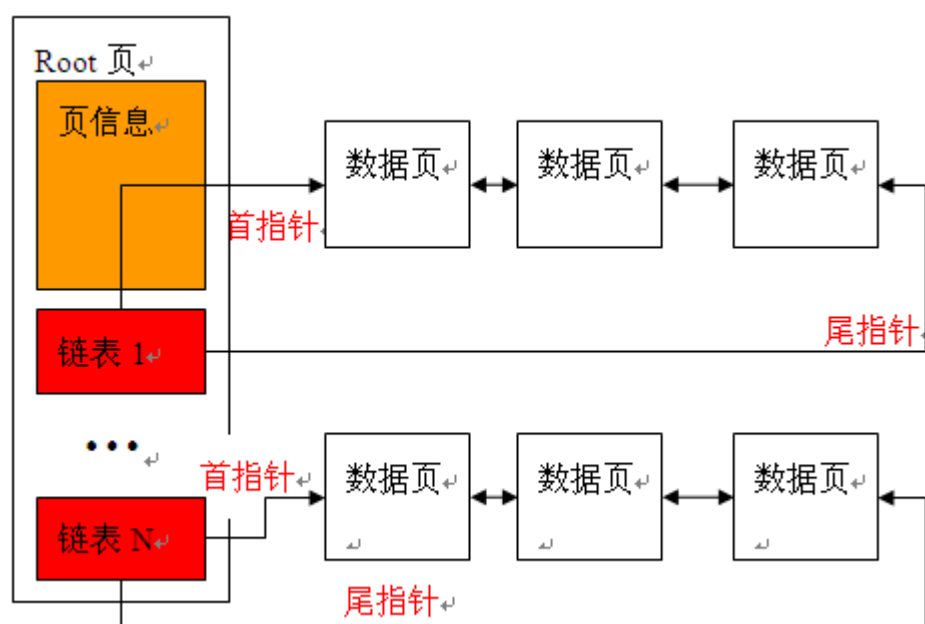


图 22.1 LIST 表存储方式

采用了物理 ROWID 形式的 LIST 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 LIST 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。

22.2 创建 LIST 表

LIST 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 LIST 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 LIST 或非 LIST 时，默认情况下创建的表为 LIST 表；
- 2) 如果 LIST_TABLE = 0，则在未显式指定表是否为 LIST 或非 LIST 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 LIST 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 DM《SQL 语言使用手册》。

- 1) NOBRANCH：如果指定为 NOBRANCH，则创建的表为 LIST 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 LIST 表，并发分支个数为 n，非并发个数为 m；
- 3) CLUSTERBTR：创建的表为非 LIST 表，即普通 B 树表。

22.3 LIST 的限制

LIST 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。LIST 表不支持以下功能：

1. 聚集索引

LIST 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 LIST 表创建聚集索引了。在创建 LIST 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 LIST 表。

2. 全文索引

DM 服务器暂时不支持此功能。

3. 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 LIST 表的列存储。

22.4 维护 LIST 表

LIST 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。

LIST 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。

如果对 LIST 表进行了备份和恢复，此时 LIST 表上的所有二级索引都不能工作，因为物理 ROWID 发生了改变，二级索引都需要全部删除后重建。达梦服务器支持对 LIST 表的备份操作，恢复数据时，只恢复 B 树数据，如果想使用二级索引，用户必须手动重建二级索引。

22.5 查看有关 LIST 表的信息

可以通过系统函数 TABLEDEF('SCHEMA_NAME','TABLE_NAME')查看 LIST 表的定义信息，该函数的详细信息可参考 DM 《SQL 语言使用手册》。

第 23 章 管理作业

在管理员的工作中，有许多管理职责每天都是固定不变的，比如，每天在下班之前要对数据库进行备份，定期对数据进行分析并生成统计表等等。这些工作通常是单调而又费时的，将这些可重复的管理任务进行自动化管理，可以节省大量的时间以执行那些缺乏可预测性或可编程性的任务。

作业与调度系统为那些可预测的管理职责和一些预定义的服务器事件提供了管理和响应功能，实现管理任务的自动化。

23.1 作业概述

DM 的作业与调度系统为用户提供了创建作业，并对作业进行调度执行以完成相应管理任务的功能。用户通过作业可以实现对数据库的操作，并将作业执行结果以通知的形式反馈到操作员。通过为作业创建灵活的调度方案可以满足在不同时刻运行作业的要求。用户还可以定义警报响应，以便当服务器发生特定的事件时通知操作员或者执行预定义的作业。

为了更好地理解作业与调度，下面介绍一些相关的概念：

23.1.1 操作员

操作员是负责维护 DM 服务器运行实例的个人。在有些企业中，操作员由单独一个人担任。在那些拥有很多服务器的大型企业中，操作员由多人一起担任。在预期的警报（或事件）发生时，可以通过电子邮件等方式将警报（或事件）的内容通知到操作员。

23.1.2 作业

作业是由 DM 代理程序按顺序执行的一系列指定的操作。作业可以执行更广泛的活动，包括运行 DMPL/SQL 脚本、定期备份数据库、对数据库数据进行检查等。可以创建作业来执行经常重复和可调度的任务，作业按照一个或多个调度的安排在服务器上执行。作业也可以由一个或多个警报触发执行，并且作业可产生警报以通知用户作业的状态（成功或者失败）。每个作业由一个或多个作业步骤组成，作业步骤是作业对一个数据库或者一个服务器执行的动作。每个作业必须至少有一个作业步骤。

23.1.3 警报

警报是系统中发生的某种事件，如发生了特定的数据库操作，或出错信号，或者是作业的启动、执行完毕等事件。警报主要用于通知指定的操作员，以便其迅速了解系统中发生了什么事情。可以为警报定义产生的条件，还可以定义当警报产生时系统采取的动作，如通知一个或多个操作员执行某个特定的作业等。

23.1.4 调度

调度是用户定义的一个时间安排，在给定的时刻到来时，系统会启动相关的作业，按作业定义的步骤依次执行。调度可以是一次性的，也可以是周期性的。

23.2 配置作业管理系统

在 DM7 的作业管理系统中，要进行作业管理，首先需要创建一些用户表来存储作业相关的对象、历史记录等信息，这些表可以由用户自行创建，也可以是使用客户端配置时由客户端负责创建，这些表有 SYSJOBS、SYSJOBSTEPS、SYSJOBSCHEDULES、SYSMAILINFO、SYSJOBHISTORIES、SYSALERTHISTORIES、SYSOPERATORS、SYSALERTS、SYSALERTNOTIFICATIONS，下面是这些表的具体定义语句：

1. SYSJOBS

```
CREATE TABLE SYSDBA.SYSJOBS (  
  ID                INT IDENTITY(1,1),  
  NAME              VARCHAR,  
  ENABLE            INT,  
  USERNAME          VARCHAR,  
  CREATETIME        VARCHAR,  
  MODIFYTIME        VARCHAR,  
  ENABLE_EMAIL      INT,  
  EMAIL_OPERID      INT,  
  EMAIL_TYPE        INT,  
  ENABLE_NETSEND    INT,  
  NETSEND_OPERID    INT,  
  NETSEND_TYPE      INT,  
  VALID             CHAR,  
  DESCRIBE          VARCHAR,  
  CLUSTER PRIMARY KEY(NAME));
```

功能解释：

SYSJOBS 存储用户定义的作业信息，每一个作业对应此表中的一条记录。每一条记录都有一个自增 ID，用来唯一表示这个作业，同时这个作业还具有一个聚簇关键字 NAME，意味着作业不可以同名。

参数解释：

ID：自增列 ID 号，用来唯一表示作业 ID；

NAME：作业名字，必须是有效的标识符，同时不能是 DM 关键字，不能重复；

ENABLE：表示作业是否启用，布尔类型，1 表示启用，0 表示不启用；

USERNAME：表示作业的创建者，也就是 SYSMAILINFO 表中的 LOGIN_NAME 所表示的登录名，这个列不可以由用户自己指定；

CREATETIME：表示作业的创建时间，由系统时间指定；

MODIFYTIME：表示作业最后一次被修改的时间，由系统时间指定；

ENABLE_EMAIL: 表示作业是否开启邮件系统, 如果开启, 则这个作业相关的一些日志会通过邮件通知操作员, 如果不开启则不会发送邮件;

EMAIL_OPERID: 如果开启了邮件通知功能, 则这个列指定通过邮件来通知哪一个操作员, 这里记录的是操作员 ID 号;

EMAIL_TYPE: 表示如果在开启了邮件发送之后, 在什么情况下发送邮件。情况分为三种: **JOB_TYPE_WHEN_COMPLETE**、**JOB_TYPE_WHEN_FAIL**、**JOB_TYPE_WHEN_SUCCESS**, 分别表示在作业执行完成后发送、在作业执行失败后发送、在作业执行成功后发送, 在存储于这个表中时, 分别用它们对应的值来表示: 2、1、0;

ENABLE_NETSEND: 表示作业是否开启网络发送, 如果开启, 则这个作业相关的一些日志会通过网络发送通知操作员, 如果不开启则不会通知, 这个功能只有在 **WINDOWS** 操作系统下才会执行, 同时必须开启服务 **MESSAGER** 才能发送成功;

NETSEND_OPERID: 如果开启了网络信息通知功能, 则这个列指定通过网络发送来通知哪一个操作员, 这里记录的是操作员 ID 号;

NETSEND_TYPE: 这个列表示如果在开启了网络发送之后, 在什么情况下发送网络信息, 这个情况也有三种, 和上面的 **EMAIL_TYPE** 是完全一样的;

VALID: 这个列表示作业的完整性, 如果不完整, 则这个作业需要重新配置, 使它的状态为完整的, 字段的值为 ‘Y’ 或者 ‘N’;

DESCRIBE: 这个列的作用是对作业设置一些描述信息, 相当于对这个作业的一些注释, 最长为 500 个字节。

2. SYSJOBSTEPS

```
CREATE TABLE SYSDBA.SYSJOBSTEPS (  
  ID                INT IDENTITY(1,1),  
  NAME              VARCHAR,  
  JOBID             INT,  
  SEQNO             INT,  
  TYPE              INT,  
  DBNAME            VARCHAR,  
  COMMAND           VARCHAR(1800),  
  SUCC_ACTION       INT,  
  FAIL_ACTION       INT,  
  RETRY_ATTEMPTS    INT,  
  RETRY_INTERVAL    INT,  
  OUTPUT_FILE_PATH  VARCHAR(256),  
  APPEND_FLAG       INT,  
  CLUSTER PRIMARY KEY(JOBID, NAME));
```

功能解释:

SYSJOBSTEPS 存储作业包括的所有步骤, 每一行存储了某个作业的某个步骤的所有属性。这个表的聚簇关键字为 **JOBID** 和步骤名, 意味着在一个指定的作业下, 不能有两个同名的步骤。

参数解释:

ID: 这个列是一个自增的 ID 列, 唯一表示一个步骤的 ID 号;

NAME: 表示一个作业下某一个步骤的步骤名, 必须是有效的标识符, 同时不能是 **DM** 关键字。同一个作业不能有同名的步骤;

JOBID: 表示步骤属于哪一个作业，这个作业由 **JOBID** 号指定，创建时这个作业必须存在才能创建成功；

SEQNO: 表示步骤在作业中的序列号；

TYPE: 表示步骤的类型，主要有以下几种：**STEP_TYPE_SQL_TXT**、**STEP_TYPE_BACKUP**、**STEP_TYPE_REORGANIZE**、**STEP_TYPE_UPDATESTAT**、**STEP_TYPE_DTS**、**STEP_TYPE_APP**，分别表示这个步骤的操作为执行一段 SQL 语句或者是语句块、重组数据库、更新数据库的统计信息、执行 DTS（数据迁移）、执行一个指定的应用程序。存储在表中时分别用值 0、1，2，3，4，5 表示；

DBNAME: 表示步骤所属的数据库名（实例名），这是系统在创建一个步骤时指定的，也就是当前运行的数据库实例名；

COMMAND: 对应在不同步骤类型下，步骤在运行时所执行的不同语句或者语句块。它与步骤类型有关，当类型为 **STEP_TYPE_SQL_TXT** 时，这个列的值就是用户指定的要执行的 SQL 语句或者语句块，如果要指定多条语句，在语句之间必须用分号隔开，不支持多条 DDL 语句一起执行，否则在执行时可能会报出不可预知的错误信息。如果是其它类型时，要执行的语句就是由系统内部根据不同类型生成的不同语句或者过程，生成时可能会用到上面的参数 **DBNAME**；

SUCC_ACTION: 指定步骤执行成功后，下一步该做什么事。有三个可选项：**STEP_ACTION_NEXT_STEP**、**STEP_ACTION_REPORT_SUCCESS** **STEP_ACTION_RETURN_STEP_1**，分别表示执行下一步、报告执行成功、返回第一个步骤继续执行，分别用 0、1、3 表示；

FAIL_ACTION: 指定步骤执行失败后，下一步该做什么事，有三个可选项：**STEP_ACTION_NEXT_STEP**、**STEP_ACTION_REPORT_FAIL**、**STEP_ACTION_RETURN_STEP_1**，分别表示执行下一步、报告执行失败、返回第一个步骤继续执行，分别用 0、2、3 表示；

RETRY_ATTEMPTS: 表示当步骤执行失败后，需要重试的次数；

RETRY_INTERVAL: 表示在每两次步骤执行重试之间的间隔时间，不能大于 10 秒钟；

OUTPUT_FILE_PATH: 表示步骤执行时如果有输出结果，输出信息的文件路径，在创建步骤时，这个路径必须是有效的。

APPEND_FLAG: 如果指定输出文件，那么这个列表示在写入文件时是否从文件尾开始写。

3. SYSJOBSCHEDULES

```
CREATE TABLE SYSDBA.SYSJOBSCHEDULES (  
  ID                INT IDENTITY(1,1),  
  NAME              VARCHAR,  
  JOBID             INT,  
  ENABLE            INT,  
  TYPE              INT,  
  FREQ_INTERVAL     INT,  
  FREQ_SUB_INTERVAL INT,  
  FREQ_MINUTE_INTERVAL INT,  
  STARTTIME         VARCHAR,  
  ENDTIME           VARCHAR,  
  DURING_START_DATE VARCHAR,  
  DURING_END_DATE   VARCHAR,
```

SCHNAME	VARCHAR,
TRIGNAME	VARCHAR,
VALID	CHAR,
DESCRIBE	VARCHAR(500),
CLUSTER PRIMARY KEY(JOBID, NAME));	

功能解释:

一个作业可以有多个调度,调度用来指定一个作业的执行情况,可以指定作业的执行方式及时间范围。**SYSJOBSCHEDULES** 表存储作业的调度信息,聚簇关键字为 **JOBID** 及调度名,意味着对于一个指定的作业,不能具有同名的调度。

参数解释:

ID: 这个列是一个自增的 **ID** 列,唯一表示一个调度的 **ID** 号;

NAME: 表示调度的名字,必须是有效的标识符,同时不能是 **DM** 关键字。一个作业不能创建同名的调度;

JOBID: 指定调度属于哪一个作业,一个作业可以有多个调度,在创建时通过作业名来指定;

TYPE: 指定调度的类型,有如下几个可选值: **SCHE_TYPE_EXEC_ONE**、**SCHE_TYPE_EXEC_DAILY**、**SCHE_TYPE_EXEC_WEEKLY**、**SCHE_TYPE_EXEC_MONTHLY_ONE_DAY**、**SCHE_TYPE_EXEC_MONTHLY_FIRST_WEEK**、**SCHE_TYPE_EXEC_MONTHLY_SECOND_WEEK**、**SCHE_TYPE_EXEC_MONTHLY_THIRD_WEEK**、**SCHE_TYPE_EXEC_MONTHLY_FORTH_WEEK**、**SCHE_TYPE_EXEC_MONTHLY_LAST_WEEK**,分别表示指定作业只执行一次、按天的频率来执行、按周的频率来执行、在一个月的某一天执行、在一个月的第一周第几天执行、在一个月的第二周的第几天执行、在一个月的第三周的第几天执行、在一个月的第四周的第几天执行、在一个月的最后一周的第几天执行,分别用 0、1、2、3、4、5、6、7、8 来表示;

FREQ_INTERVAL: 这个列的可选值与上面的不同调度类型有关,当类型为 **SCHE_TYPE_EXEC_ONE** 时,这个值无效,系统不会做检查;当类型为 **SCHE_TYPE_EXEC_DAILY** 时,这个列表示每隔几天执行,值的有效范围为 1 到 100;当类型为 **SCHE_TYPE_EXEC_WEEKLY** 时,这个列表示的是每隔几个星期执行,值的范围没有具体的限制;当类型为 **SCHE_TYPE_EXEC_MONTHLY_ONE_DAY** 时,表示每几个月中的某一天执行,值的范围没有具体的限制;当调度类型为 **SCHE_TYPE_EXEC_MONTHLY_FIRST_WEEK**、**SCHE_TYPE_EXEC_MONTHLY_SECOND_WEEK**、**SCHE_TYPE_EXEC_MONTHLY_THIRD_WEEK**、**SCHE_TYPE_EXEC_MONTHLY_FORTH_WEEK**、**SCHE_TYPE_EXEC_MONTHLY_LAST_WEEK** 时,都表示每几个月的某一周执行,值的范围也没有具体的限制;

FREQ_SUB_INTERVAL: 这个列的可选值与上面的不同调度类型有关,当类型为 **SCHE_TYPE_EXEC_ONE** 时,这个值无效,系统不会做检查;当类型为 **SCHE_TYPE_EXEC_DAILY** 时,这个列没有意义,不做检查;当类型为 **SCHE_TYPE_EXEC_WEEKLY** 时,这个列表示的是某一个星期的星期几执行,每周有七天,那么只用这个列的七个位来表示选中了星期几,从最低位开始算起,第一个位表示星期日,第二个位表示星期一...第七个位表示星期六,可以同时选中七天中的任意几天,所以这个列的有效值范围为 0x01 到 0x7F (十六进制),也就是说哪一个位的值为 1,则说明选中了这天,反之则没有选中;当类型为 **SCHE_TYPE_EXEC_MONTHLY_ONE_DAY** 时,表示将在一个月的第几天执行,此时这个列的有效值范围为 1 到 31;当调度类型为

SCHE_TYPE_EXEC_MONTHLY_FIRST_WEEK、
 SCHE_TYPE_EXEC_MONTHLY_SECOND_WEEK、
 SCHE_TYPE_EXEC_MONTHLY_THIRD_WEEK、SCHE_TYPE_EXEC_MONTHLY_FORTH_WEEK、SCHE_TYPE_EXEC_MONTHLY_LAST_WEEK 时，都表示将在某一周内第几天执行，有效值为 1 到 7，分别表示从周一到周日；

FREQ_MINUTE_INTERVAL：表示一天内每隔多少分钟执行一次，有效值范围从 1 到 1440；

STARTTIME：定义作业被调度的有效时间范围的起始时间，必须是有效的时间字符串，不可以为空；

ENDTIME：定义作业被调度的有效时间范围的结束时间，可以为空。如果不为空，必须是有效的时间字符串，同时必须要在 STARTTIME 时间的后面；

DURING_START_DATE：指定作业被调度的有效日期范围的起始日期，必须是有效的日期字符串，不可以为空；

DURING_END_DATE：指定作业被调度的有效日期范围的结束日期，可以为空。如果不为空，必须是有效的日期字符串，同时必须是在 DURING_START_DATE 日期之后；

SCHNAME：表示调度对应的驱动触发器所属的数据库模式名；

TRIGNAME：表示驱动触发器的触发器名；

VALID：表示调度是否合法，如果是 N，说明这个调度的配置是没有完成的，需要重新配置完成后才能起作用；

DESCRIBE：表示调度的注释信息，最大值为 500 个字节。

4. SYSJOBHISTORIES

```
CREATE TABLE SYSDBA.SYSJOBHISTORIES (
  ID          INT IDENTITY(1,1),
  NAME        VARCHAR,
  STEPNAME    VARCHAR,
  STATUS      VARCHAR,
  ERRRTYPE    INT,
  ERRCODE     INT,
  ERRINFO     VARCHAR(1024),
  CUR_TIME    VARCHAR,
  RETRY_ATTEMPTS INT,
  HAS_NOTIFIED INT,
  CLUSTER PRIMARY KEY(ID));
```

功能解释：

SYSJOBHISTORIES 存储作业步骤的执行情况日志。每当一个作业执行开始时都会向这个表中插入一条作业执行开始的记录，其 STATUS 为“JOB START”，作业完成时也会插入一条作业执行完成的记录，其 STATUS 为“JOB END”。每一个步骤执行开始时都会插入一条步骤开始执行的记录，其 STATUS 为“JOB STEP START”，如果为重试执行开始则为“JOB STEP RERTY START”，每一个步骤执行完成或者重试完成都会插入一条相应状态的记录。

这个表中的所有记录都是由作业在运行过程中系统自动插入的，不是由用户来操作的。

参数解释：

ID：表中每一行的唯一标识；

NAME：表示某一条历史记录是由哪一个作业产生的，用作业名表示；

STEPNAME: 表示历史记录是由哪一个步骤产生的, 用步骤名表示;
 STATUS: 表示某一条历史记录是在作业或步骤的什么状态下产生的;
 ERRTYPE: 这个列一般情况不用, 现在都是 0;
 ERRCODE: 表示在步骤执行错误后, 产生的错误码;
 ERRINFO: 表示在步骤执行错误后, 系统产生的错误描述信息;
 CUR_TIME: 表示产生这条历史记录的系统时间;
 RETRY_ATTEMPTS: 表示这条历史记录是第几次重试时产生的, 这个列记录其当前次数;
 HAS_NOTIFIED: 表示这条历史记录是否已经 (邮件及网络发送) 通知用户, 如果通知则这个值为 1, 否则为 0;

5. SYSOPERATORS

```
CREATE TABLE SYSDBA.SYSOPERATORS (
  ID          INT IDENTITY(1,1),
  NAME        VARCHAR,
  ENABLE      INT,
  EMAILADDR   VARCHAR,
  NETSEND_IP  VARCHAR,
  CLUSTER PRIMARY KEY(NAME));
```

功能解释:

SYSOPERATORS 存储作业管理系统中所有已定义操作员的信息, 以 NAME 为聚簇索引, 意味着不能具有同名的操作员。

参数解释:

ID: 每一个操作员都有一个唯一的 ID 号;
 NAME: 操作员的名字, 必须是有效的标识符, 同时不能是 DM 关键字, 不能同名;
 ENABLE: 表示操作员是否有效;
 EMAILADDR: 表示操作员的 EMAIL 地址;
 NETSEND_IP: 表示操作员所在的 IP 地址, 在创建时, 必须指定合法的 IP 地址, 否则报错。

6. SYSALERTS

```
CREATE TABLE SYSDBA.SYSALERTS (
  ID          INT IDENTITY(1,1),
  NAME        VARCHAR,
  ENABLE      INT,
  TYPE        INT,
  DBNAME      VARCHAR,
  ERRTYPE     INT,
  ERRCODE     INT,
  DELAYTIME   INT,
  ADDITION_TXT VARCHAR,
  CLUSTER PRIMARY KEY(NAME));
```

功能解释:

SYSALERTS 存储作业管理系统中所有已定义的警报信息, 聚簇索引为 NAME, 意味着

不能定义同名的警报。

参数解释：

ID：每一个警报都有一个唯一标识的 ID 号；

NAME：警报名，必须是有效的标识符，同时不能是 DM 关键字，不能同名；

ENABLE：警报是否开启；

TYPE：警报的类型，有以下几种：ALERT_TYPE_ERROR、ALERT_TYPE_EVENT，分别表示发生错误时警报及发生某些事件时警报，值分别为 0 和 1。创建时必须指定 0 或 1；

DBNAME：警报所在的数据库实例名；

ERRTYPE：这个列的值与上面的 TYPE 有关。如果 TYPE 为 ALERT_TYPE_ERROR，则 ERRTYPE 的值有以下几个可选项：ALERT_ERROR_TYPE_NORMAL、ALERT_ERROR_TYPE_STARTUP、ALERT_ERROR_TYPE_SYSTEM、ALERT_ERROR_TYPE_CONFIG、ALERT_ERROR_TYPE_PARSE、ALERT_ERROR_TYPE_PRIV、ALERT_ERROR_TYPE_RUNTIME、ALERT_ERROR_TYPE_BACKRES、ALERT_ERROR_TYPE_JOB、ALERT_ERROR_TYPE_REPLICATE、ALERT_ERROR_TYPE_OTHER、ALERT_ERROR_TYPE_CODE，分别表示常规错误、系统错误、启动错误、系统错误、分析阶段错误、权限错误、运行时错误、备份恢复错误、作业管理错误、数据库复制错误、其它错误、指定错误码，值分别用 1、2、3、4、5、6、7、8、9、10、11、12 表示；如果 TYPE 指定的是 ALERT_TYPE_EVENT，则 ERRTYPE 有如下几个可选项：ALERT_EVENT_TYPE_DDL、ALERT_EVENT_TYPE_PRIV、ALERT_EVENT_TYPE_CONNECT、ALERT_EVENT_TYPE_BACKUP，分别表示 DDL 事件、授权回收事件、连接或断开数据库事件、数据库备份恢复事件，值分别用 1、2、3、4 表示。

ERRCODE：这个列的值与上面的 TYPE 有关。如果 TYPE 指定的是 ALERT_TYPE_ERROR，则这个列指定的必须是一个要捕获的错误码，此时 ERRTYPE 必须选择 ALERT_ERROR_TYPE_CODE 才有效，同时错误码必须是小于 0 的值；如果 TYPE 指定的是 ALERT_TYPE_EVENT，则这个列指定的值与 ERRTYPE 有关，如果 ERRTYPE 为 ALERT_EVENT_TYPE_DDL，ERRCODE 可以是 ALERT_DDL_TYPE_CREATE、ALERT_DDL_TYPE_ALTER、ALERT_DDL_TYPE_DROP、ALERT_DDL_TYPE_TRUNC 中的任意多个的组合值，ERRCODE 只用最低四个位来表示它们的组合，低 1 位为 1 表示 CREATE，低 2 位为 1 表示 ALTER，低 3 位为 1 表示 DROP，低 4 位为 1 表示 TRUNCATE，如果为 0 表示不指定，所以这个列的有效范围为 1 到 15；当 ERRTYPE 为 ALERT_EVENT_TYPE_PRIV 时，ERRCODE 可以是 ALERT_PRIV_TYPE_GRANT 和 ALERT_PRIV_TYPE_REVOKE 的组合，最低位为 1 表示 GRANT，0 为不指定，上一位为 1 表示 REVOKE，0 为不指定，所以这个列的有效值范围为 1 和 2；如果 ERRTYPE 的值为 ALERT_EVENT_TYPE_CONNECT，那么 ERRCODE 可以是 ALERT_CONNECT_TYPE_LOGIN 和 ALERT_CONNECT_TYPE_LOGOUT 的任意组合，这个列的最低位为 1 表示 LOGIN，0 表示不指定，上一位为 1 表示 LOGOUT，0 为不指定，所以这个列的有效值范围为 1 和 2；如果 ERRTYPE 为 ALERT_EVENT_TYPE_BACKUP，则 ERRCODE 有两个可选项，分别是 ALERT_BACKUP_TYPE_BACKUP 和 ALERT_BACKUP_TYPE_RESTORE，最低位为 1 表示 BACKUP，0 为不指定，上一位为 1 表示 RESTORE，0 为不指定，所以这个列的有效值范围为 1 和 2；

DELEYTIME：表示警报通知操作员时需要推迟的时间，范围为 0 到 3600 秒；

ADDITION_TXT：表示警报的注释新，最长为 500 个字节。

7. SYSALERTNOTIFICATIONS

```
CREATE TABLE SYSDBA.SYSALERTNOTIFICATIONS (  
  ALERTNAME      VARCHAR,  
  OPERID         INT,  
  ENABLE_EMAIL   INT,  
  ENABLE_NETSEND INT ,  
  CLUSTER PRIMARY KEY(ALERTNAME, OPERID));
```

功能解释:

SYSALERTNOTIFICATIONS 存储警报需要通知的操作员的信息，即警报和操作员的关联信息。以 ALERTNAME 和 OPERID 为聚簇关键字的，所以对于一个指定的警报和指定的操作员，它们只能有一个关联关系。

参数解释:

ALERTNAME: 表示指定的警报名;

OPERID: 表示指定的操作员的 ID 号;

ENABLE_EMAIL: 表示是否需要通过发邮件来通知;

ENALE_NETSEND: 表示是否通过网络发送的方式来通知，只在 WINDOWS 下起作用。

8. SYSALERTHISTORIES

```
CREATE TABLE SYSDBA.SYSALERTHISTORIES (  
  ID              INT IDENTITY(1,1),  
  ALERTNAME      VARCHAR,  
  EVENT_TYPE     INT,  
  SUB_TYPE       INT,  
  USERNAME       VARCHAR(128),  
  DB_NAME        VARCHAR(128),  
  OPTIME         VARCHAR(128),  
  OPUSER         VARCHAR(128),  
  SCH_NAME       VARCHAR(128),  
  OBJ_NAME       VARCHAR(256),  
  OBJ_TYPE       VARCHAR(128),  
  GRANTEE_NAME   VARCHAR(256),  
  ERRCODE        INT,  
  CLUSTER PRIMARY KEY(ID));
```

功能解释:

SYSALERTHISTORIES 存储警报发生的历史记录，每个警报发生时都会向这个表中插入相应的记录，然后 Dmjmon 再通过扫描这个表把信息取出来通过邮件或者网络发送的方式通知指定操作员，这个表中的所有信息都是在发生警报时，由系统自动向这个表中插入的。

参数解释:

ID: 每一条警报历史记录都有一个唯一标识为 ID 号;

ALERTNAME: 表示历史记录是由哪一个警报产生的;

EVENT_TYPE: 表示历史记录的事件类型，与 SYSALERTS 表中的 TYPE 列相关。如

果 TYPE 为 ALERT_TYPE_ERROR, 则这个列的值也为 ALERT_TYPE_ERROR, 如果 TYPE 的值为 ALERT_TYPE_EVENT, 则这个值与 SYSALERTS 表中的列 ERRTYPE 相对应;

SUB_TYPE: 表示历史记录的事件子类型, 与 SYSALERTS 表中的 TYPE 列相关。如果 TYPE 为 ALERT_TYPE_ERROR, 则这个列的值也为 ALERT_TYPE_ERROR, 如果 TYPE 的值为 ALERT_TYPE_EVENT, 则这个值与 SYSALERTS 表中的列 ERRCODE 相对应;

USERNAME: 表示发生警报时, 当前的登录用户名;

DB_NAME: 表示警报所处的数据库实例名;

OPTIME: 表示警报发生时间;

OPUSER: 表示警报发生时当前的登录用户名;

SCH_NAME: 表示触发警报的数据对象所属的模式名;

OBJ_NAME: 表示触发警报的数据对象名;

OBJ_TYPE: 表示触发警报的数据库对象类型;

GRANTEE_NAME: 表示由授权或回收权限引起的警报的授权者或回收者名;

ERRCODE: 表示当警报定义的 TYPE 为错误时, 发生错误时的错误码。

9. SYSMAILINFO

```
CREATE TABLE SYSDBA.SYSMAILINFO (  
  LOGIN_NAME      VARCHAR(128) NOT NULL,  
  LOGIN_PWD       VARCHAR(128) NOT NULL,  
  SMTP_SERVER     VARCHAR(256) NOT NULL,  
  EMAIL           VARCHAR(256) NOT NULL,  
  USER_NAME       VARCHAR(128) NOT NULL,  
  USER_PWD        VARCHAR(128) NOT NULL,  
  CLUSTER PRIMARY KEY(LOGIN_NAME));
```

功能解释:

SYSMAILINFO 存储作业管理系统管理员的信息的, 每一个作业管理系统管理员对应这个表中的一条记录, 登录用户名是唯一的, 因为这个表是以第一列 LOGIN_NAME 为聚簇关键字的。

参数解释:

LOGIN_NAME: 这个列表示的是一个作业管理系统管理员在登录作业管理系统时用到的登录名; 不可以为空, 如果在某一个用户登录时指定的登录名不在这个表中, 那么它是不能够登录的。此时可能需要创建一个新的登录信息, 当然如果这个用户有这个权限才可以这么做。

LOGIN_PWD: 这个列表示上面作业管理系统管理员在登录时的密码, 如果密码不正确是不允许登录的。

EMAIL: 这个列表示在开启邮件系统后, 给操作员发邮件时用到邮件的地址。

SMTP_SERVER: 这个列与上面的参数对应, 表示上面的邮件地址对应的 SMTP 服务器地址。这个地址必须要与 EMAIL 地址相对应, 如果没有对应, 则邮件不能成功发送。

USER_NAME: 这个列表示的是在发送邮件时, 对 SMTP 服务器进行验证的邮件用户名及密码, 如果验证成功才能成功发送邮件。

USER_PWD: 这个列与上面的用户名相对应, 这个就是邮箱的登录密码。

23.3 创建删除修改操作员

23.3.1 创建操作员

在数据库中创建了上述作业管理表后，就可以进行作业的配置了。由于在创建一个作业时必须要指定操作员，所以首先需要创建一个操作员。

创建操作员可以直接通过调用系统函数来实现，函数名为 `SP_CREATE_OPERATOR`，所创建的操作员信息都会存储于上面所述的作业表 `SYSOPERATORS` 中，所以这个函数的参数属性都来源于表 `SYSOPERATORS` 中的列属性，参数如下：

1. `OPR_NAME`: 操作员名字，必须是有效的标识符，同时不能是 DM 关键字。不能有同名操作员，如果创建同名操作员，系统会报错；
2. `ENABLED`: 是否启用这个操作员，参数类型为布尔类型；
3. `EMAILADDR`: 操作员的 EMAIL 地址；
4. `NETSEND_IP`: 操作员的 IP 地址（用于网络发送），在创建时必须合法的 IP 地址，否则报错。

例如创建一个名为“WZF”的操作员。

```
SP_CREATE_OPERATOR('WZF', 1, 'tlf577@qq.com', '192.168.0.123');
```

23.3.2 删除操作员

可以通过调用系统函数来删除一个操作员，函数名为 `SP_DROP_OPERATOR`，这个函数只需要一个参数即操作员名。

如果调用这个函数时指定的操作员不存在，则系统会报操作员不存在的错误，如果存在则会从 `SYSOPERATORS` 中将这条记录删除，同时将所有这个操作员与警报的关联关系从 `SYSALERTNOTIFICATIONS` 中删除。如果某一个作业在创建时指定了这个操作员，此时并不会删除这个作业，不会对这个作业造成任何影响。

例如删除名为“WZF”的操作员。

```
SP_DROP_OPERATOR('WZF');
```

23.3.3 修改操作员

如果 DBA 希望修改某一个操作员的某些信息，DM 也提供了相应的系统函数，函数名为 `SP_ALTER_OPERATOR`，参数和 `SP_CREATE_OPERATOR` 完全一样的，只是某些参数的意义不同，有些参数是可以修改的，有些则不可以，如果需要修改则传入新的参数值，如果不需要修改则传入原来的值即可。下面具体说明：

1. `OPR_NAME`: 操作员名字，指定要修改哪一个操作员，如果这个操作员不存在，系统会报操作员不存在的错误；
2. `ENABLED`: 可修改参数，表示是否启用这个操作员，为布尔类型。如果将这个值修改为 0，表明将这个操作员修改为不可用；
3. `EMAILADDR`: 可修改参数，表示操作员的 EMAIL 地址；
4. `NETSEND_IP`: 操作员的 IP 地址（主要用于网络发送），在修改时指定的新 IP 地

址必须是合法的，不然会报错。

例如修改操作员“WZF”的信息。

```
SP_ALTER_OPERATOR('WZF', 0, 'tlf577@126.com', '192.168.0.234');
```

23.4 创建修改删除作业

23.4.1 创建作业

在创建操作员之后，就可以创建作业了。创建作业通过系统函数 `SP_CREATE_JOB` 实现。创建的每一个作业都存储在作业表 `SYSJOBS` 中。

函数 `SP_CREATE_JOB` 主要有如下参数：

1. **NAME**：作业名字，必须是有效的标识符，同时不能是 `DM` 关键字。作业名不能重复，在创建时系统会检查这个作业是否存在，如果存在则报错；
2. **ENABLE**：表示作业是否启用，布尔类型。表示启用，0 表示不启用；
3. **ENABLE_EMAIL**：表示作业是否开启邮件系统，如果开启，则这个作业相关的日志会通过邮件来通知操作员，如果不开启则不会发送邮件；
4. **EMAIL_OPTR_NAME**：如果开启了邮件通知功能，则这个参数指定的是通过邮件通知哪一个操作员，这里指定的是操作员名字，在创建时系统会检测这个操作员是否存在，如果不存在则报错；
5. **EMAIL_TYPE**：表示开启了邮件发送之后，在什么情况下发送邮件，有三种情况：`JOB_TYPE_WHEN_COMPLETE`、`JOB_TYPE_WHEN_FAIL`、`JOB_TYPE_WHEN_SUCCESS`，分别表示在作业执行完成后发送、在作业执行失败后发送、在作业执行成功后发送，对应的值分别是：2、1、0；
6. **ENABLE_NETSEND**：表示作业是否开启网络发送，如果开启，则这个作业相关的日志会通过网络发送来通知操作员，如果不开启则不会通知。这个功能只有在 `WINDOWS` 操作系统下才会执行，同时必须开启服务 `MESSAGER` 才能发送成功；
7. **NETSEND_OPTR_NAME**：如果开启了网络信息通知功能，则这个参数指定通过网络发送通知哪一个操作员，这里指定的是操作员名字。在创建时系统会检测这个操作员是否存在，如果不存在则报错；
8. **NETSEND_TYPE**：表示开启了网络发送之后，在什么情况下发送网络信息，有三种情况，和 `EMAIL_TYPE` 是完全一样的；
9. **DESCRIBE**：作业描述信息，最长 500 个字节。

例如创建一个名为“TEST”的作业。

```
SP_CREATE_JOB('TEST', 1, 1, 'WZF', 2, 1, 'WZF', 2, '每一个测试作业');
```

创建完成这个作业后，系统就会在 `SYSJOBS` 中插入一条相应的记录，但是这个作业不会做任何事情，只是一个空的作业，如果需要让它执行，还需要配置这个作业。

23.4.2 修改作业

如果 `DBA` 发现某一个作业中的信息不合理需要修改，可以调用系统函数 `SP_ALTER_JOB` 来实现。函数 `SP_ALTER_JOB` 的参数和 `SP_CREATE_JOB` 的参数完全相同，除了 `NAME` 不可修改外，其他的属性都可修改。对于可修改参数，如果要修改，则指定新

值；如果不修改，则指定原值即可。

例如下面的语句修改了作业“TEST”的一些信息。

```
SP_ALTER_JOB('TEST', 0, 1, 'DBA', 2, 1, 'DBA', 2, '修改一个作业');
```

23.4.3 删除作业

如果一个作业已经执行完成，或者由于其它什么原因需要删除，可以调用系统函数 `SP_DROP_JOB` 实现，只需要指定要删除的作业名即可，在删除时会检测这个作业是否存在，如果不存在则报错。

在删除一个作业时，系统会同时将与此作业相关联的所有对象都删除，包括步骤、调度等，也就是会分别从作业表 `SYSJOBSTEPS` 以及 `SYSSCHEDULES` 中删除属于这个作业的步骤及调度。

例如删除作业“TEST”。

```
SP_DROP_JOB('TEST');
```

23.5 配置作业

上面所述的内容都是最基本的一些操作，所创建的作业都还不能执行任何操作，只是一个空的作业，如果想要这个作业能执行一些指定的操作，还需要对这个作业进行配置。

配置一个作业主要包括以下几个步骤：

1. 指定要开始配置一个作业；
2. 为指定的作业增加步骤；
3. 为指定的作业增加调度；
4. 提交配置操作。

只有在提交作业配置后，这个作业才算创建完成，同时如果这个作业是 `ENABLE` 状态的，那么它会立即生效，也就是从提交时刻开始就会根据它所定义的调度来执行操作了。

23.5.1 作业配置开始

用系统函数 `SP_JOB_CONFIG_START` 来指定对一个作业配置的开始，参数是要配置的作业的名字。执行时会检测这个作业是否存在，如果不存在则报错。

这个函数的作用主要是为了保证作业配置的完整性，执行这个函数后当前会话会处于作业配置状态。在配置开始后，到配置提交之间，不允许做任何创建、修改、删除对象（作业、操作员、警报）的操作，同时强烈建议在配置过程中用户不要做任何 `COMMIT` 操作或者是具有自动提交操作的相关操作，比如 `DDL`（如果设置了 `DDL` 自动提交），否则在配置作业出现错误时，这些操作就不会被回滚了。在 `DM` 的作业配置过程中，如果配置出现错误时，可以直接使用 `ROLLBACK` 将错误的配置回滚到 `SP_JOB_CONFIG_START` 刚执行的状态，因为在这个函数执行时会自动提交前面所做的操作。所以在配置一个作业开始前，也需要谨慎，需要注意前面做的操作是否需要提交。

另外，如果一个数据库实例设置了 `DML` 自动提交后，系统不允许对作业进行配置操作。

例如开始对作业“TEST”进行配置。

```
SP_JOB_CONFIG_START('TEST');
```

23.5.2 增加步骤

增加作业步骤必须是在配置作业开始后才能进行，否则系统会报错，这样处理主要是为了保证作业配置的完整性。

增加作业的步骤通过系统函数 `SP_ADD_JOB_STEP` 实现，这个函数包括如下参数：

1. **JOBNAME**：表示正在给哪一个作业增加步骤，这个参数必须为上面调用 `SP_JOB_CONFIG_START` 函数时指定的作业名，否则系统会报错，同时系统会检测这个作业是否存在，不存在也会报错；
2. **STEPNAME**：表示增加的步骤名，必须是有效的标识符，同时不能是 **DM** 关键字。同一个作业不能有同名的步骤，创建时会检测这个步骤是否已经存在，如果存在则报错；
3. **TYPE**：表示步骤的类型，主要有以下几种：`STEP_TYPE_SQL_TXT`、`STEP_TYPE_BACKUP`、`STEP_TYPE_REORGANIZE`、`STEP_TYPE_UPDATESTAT`、`STEP_TYPE_DTS`、`STEP_TYPE_APP`，分别表示步骤的操作为执行一段 SQL 语句或者是语句块、重组数据库、更新数据库的统计信息、执行 DTS（数据迁移）、执行一个指定的应用程序。分别用值 0、1，2，3，4，5 表示，这个参数由用户指定；
4. **COMMAND**：这个参数对应的是在不同步骤类型下，步骤在运行时所执行的不同语句或者语句块，它与步骤类型相关，且不能为空。
 - 1) 当类型为 `STEP_TYPE_SQL_TXT` 时，这个参数的值就是用户指定要执行的 SQL 语句或者语句块，如果要指定多条语句，在语句之间必须用分号隔开，不支持多条 DDL 语句一起执行，否则在执行时可能会报出不可预知的错误信息。如果是其它类型时，要执行的语句就是由系统内部根据不同类型生成的不同语句或者过程；
 - 2) 当类型为 `STEP_TYPE_BACKUP` 时，**COMMAND** 指定的是一个字符串，但字符串有相应的格式，第一个字符指定的是备份模式，分别是 `BAK_TYPE_FULL`、`BAK_TYPE_INCR`、`BAK_TYPE_BTREE`，分别表示完全备份、增量备份、B 树备份，分别用 0、1、2 表示，创建时如果第一个字符不是这三个值中的一个，系统会报错。第二个字符表示的是备份时是否进行压缩，如果压缩就用 1 表示，如果不压缩就用 0 表示。从第三个字符开始指定的是备份文件的路径，路径命令有具体的格式，对于增量备份，因为它必须要指定一个或者多个基础备份路径，每个路径之间需要用分号隔开，之后接着是备份路径，基础备份路径与备份路径需要用“|”隔开，如果不指定备份路径，则不需要指定“|”，例如 `01E:\base_bakdir1, base_bakdir2|bakdir` 就是一个合法的增量备份命令；对于其它两种备份方式，就不需要“|”符号了，可以直接在第三个字节开始指定备份路径即可；如果不指定则是默认值，所以一个合法的非增量备份 **COMMAND** 可以表示为“01E:\bakdir”；
 - 3) 当类型为 `STEP_TYPE_APP` 时，在 **COMMAND** 中只需要指定要执行的程序名或者文件名即可。当类型为 `STEP_TYPE_UPDATESTAT` 时，在 **COMMAND** 中不需要指定任何东西，当然如果指定了也不会进行任何处理。
5. **SUCC_ACTION**：指定步骤执行成功后，下一步该做什么事，有三个可选项：`STEP_ACTION_NEXT_STEP`、`STEP_ACTION_REPORT_SUCCESS`、`STEP_ACTION_RETURN_STEP_1`，分别表示执行下一步、报告执行成功、返回第

一个步骤继续执行，分别用 0、1、3 表示；

6. **FAIL_ACTION**: 指定步骤执行失败后，下一步该做什么事，有三个可选项：**STEP_ACTION_NEXT_STEP**、**STEP_ACTION_REPORT_FAIL**、**STEP_ACTION_RETURN_STEP_1**，分别表示执行下一步、报告执行失败、返回第一个步骤继续执行，分别用 0、2、3 表示；
7. **RETRY_ATTEMPTS**: 表示当步骤执行失败后，需要重试的次数；
8. **RETRY_INTERVAL**: 表示在每两次步骤执行重试之间的间隔时间，不能大于 10 秒钟；
9. **OUTPUT_FILE_PATH**: 表示步骤执行时输出信息的文件路径，这个路径必须是有效的；
10. **APPEND_FLAG**: 如果指定输出文件，那么这个参数表示在写入文件时是否从文件尾开始写。

例如下面的语句为作业“TEST”增加了步骤“STEP1”。

```
SP_ADD_JOB_STEP('TEST', 'STEP1', 0, 'insert into myinfo values(1000, "Hello World"); ', 0, 0, 2, 1, NULL, 0);
```

“STEP1”指定的是执行 SQL 语句，其 **COMMAND** 参数指定的是向 myinfo 表中插入一条记录，执行成功和失败的下一步动作都是 **STEP_ACTION_NEXT_STEP**（执行下一步），同时指定了失败后只重试两次，时间间隔为 1 秒钟。

23.5.3 删除步骤

如果用户发现一个作业中的某个步骤不需要了，可以通过系统函数 **SP_DROP_JOB_STEP** 删除这个步骤，参数有两个：

1. **JOBNAME**: 表示正在删除哪一个作业下的步骤，这个参数必须为前面调用 **SP_JOB_CONFIG_START** 函数时指定的作业名，否则系统会报错，同时系统会检测这个作业是否存在，不存在也会报错；
2. **STEPNAME**: 表示要删除的步骤名，删除时会检测这个步骤是否存在，如果不存在则报错。

例如为作业“TEST”删除步骤“STEP1”。

```
SP_DROP_JOB_STEP('TEST', 'STEP1');
```

23.5.4 增加调度

增加调度必须是在配置作业开始后才能进行，否则系统会报错，这样处理主要是为了保证作业配置的完整性。

增加调度通过调度系统函数 **SP_ADD_JOB_SCHEDULE** 实现，其参数说明如下：

1. **JOB_NAME**: 指定要给哪一个作业增加调度，这个参数必须是配置作业开始时指定的作业名，否则报错，同时系统还会检测这个作业是否存在，如果不存在也会报错；
2. **SCHEDULE_NAME**: 表示调度的名字，必须是有效的标识符，同时不能是 DM 关键字。指定的作业不能创建同名的调度，创建时会检测这个调度是否已经存在，如果存在则报错；
3. **TYPE**: 指定调度类型，有如下几个可选值：**SCHE_TYPE_EXEC_ONE**、

SCHE_TYPE_EXEC_DAILY 、 SCHE_TYPE_EXEC_WEEKLY 、
 SCHE_TYPE_EXEC_MONTHLY_ONE_DAY 、
 SCHE_TYPE_EXEC_MONTHLY_FIRST_WEEK 、 SCHE
 _TYPE_EXEC_MONTHLY_SECOND_WEEK 、 SCHE_TYPE_EXEC_MONTHLY
 _THIRD_WEEK、SCHE_TYPE_EXEC_MONTHLY_FORTH_WEEK、SCHE_TYPE
 _EXEC_MONTHLY_LAST_WEEK，分别表示指定作业只执行一次、按天的频率来
 执行、按周的频率来执行、在一个月的某一天执行、在一个月的第一周第几天执行、
 在一个月的第二周的第几天执行、在一个月的第三周的第几天执行、在一个月的第
 四周的第几天执行、在一个月的最后一周的第几天执行，分别用 0、1、2、3、4、
 5、6、7、8 表示。当类型为 SCHE_TYPE_EXEC_ONE 时，其执行时间由下面的参
 数 DURING_START_DATE 指定；

4. **FREQ_INTERVAL**：这个参数的可选值与上面的调度类型有关，当类型为 SCHE_TYPE_EXEC_ONE 时，这个值无效，系统不会做检查；当类型为 SCHE_TYPE_EXEC_DAILY 时，这个参数表示每隔几天执行，值的有效范围为 1 到 100；当类型为 SCHE_TYPE_EXEC_WEEKLY 时，这个参数表示的是每隔几个星期执行，值的范围没有具体的限制；当类型为 SCHE_TYPE_EXEC_MONTHLY_ONE_DAY 时，这个参数表示的是每几个月中的某一天执行，值的范围没有具体的限制；当调度类型为 SCHE_TYPE_EXEC_MONTHLY_FIRST_WEEK、SCHE_TYPE_EXEC_MONTHLY_SECOND_WEEK、SCHE_TYPE_EXEC_MONTHLY_THIRD_WEEK、SCHE_TYPE_EXEC_MONTHLY_FORTH_WEEK 、 SCHE_TYPE_EXEC_MONTHLY_LAST_WEEK 时，都表示每几个月的某一周执行，值的范围也没有具体的限制；
5. **FREQ_SUB_INTERVAL**：这个参数的可选值与上面的调度类型有关，当类型为 SCHE_TYPE_EXEC_ONE 时，这个值无效，系统不会做检查；当类型为 SCHE_TYPE_EXEC_DAILY 时，这个参数没有意义，不做检查；当类型为 SCHE_TYPE_EXEC_WEEKLY 时，这个参数表示的是某一个星期的星期几执行，每周有七天，那么只用这个参数的七个位来表示选中了星期几，从最低位开始算起，第一个位表示星期日，第二个位表示星期一...第七个位表示星期六，可以同时选中七天中的任意几天，所以这个参数的有效值范围为 0x01 到 0x7F（十六进制），也就是说哪一个位的值为 1，则说明选中了这天，反之则没有选中；当类型为 SCHE_TYPE_EXEC_MONTHLY_ONE_DAY 时，这个参数表示的是将在一个月的第几天执行，此时这个参数的有效值范围为 1 到 31；当调度类型为 SCHE_TYPE_EXEC_MONTHLY_FIRST_WEEK、SCHE_TYPE_EXEC_MONTHLY_SECOND_WEEK、SCHE_TYPE_EXEC_MONTHLY_THIRD_WEEK、SCHE_TYPE_EXEC_MONTHLY_FORTH_WEEK 、 SCHE_TYPE_EXEC_MONTHLY_LAST_WEEK 时，都表示将在某一周内第几天执行，参数的有效值为 1 到 7，分别表示从周一到周日；
6. **FREQ_MINUTE_INTERVAL**：表示一天内每隔多少分钟执行一次，有效值范围从 1 到 1440；
7. **STARTTIME**：定义作业被调度的有效时间范围的起始时间，必须是有效的时间字符串，不可以为空；
8. **ENDTIME**：定义作业被调度的有效时间范围的结束时间，可以为空，但如果不为空，指定时必须必须是有效的时间字符串，同时必须要在 **STARTTIME** 时间的后面；
9. **DURING_START_DATE**：指定作业被调度的有效日期范围的起始日期，必须是有

效的日期字符串，不可以为空；

10. **DURING_END_DATE**: 指定作业被调度的有效日期范围的结束日期，可以为空，但如果不为空，必须是有效的日期字符串，同时必须是在 **DURING_START_DATE** 日期之后；

11. **DESCRIBE**: 表示调度的注释信息，最大长度为 500 个字节。

例如下面的语句为作业“TEST”增加名为“SCHEDULE3”的调度。

```
SP_ADD_JOB_SCHEDULE('TEST', 'SCHEDULE3', 1, 1, 1, 0, 1, CURTIME, '23:59:59', CURDATE,
NULL, '一个测试调度');
```

上面的例子指定新的调度名为'SCHEDULE3'，ENABLE 为 1，即启用这个调度，其调度类型为 SCHE_TYPE_EXEC_DAILY。FREQ_INTERVAL 指定为 1，说明每天都要执行，在这种类型下 FREQ_SUB_INTERVAL 参数就不会检查，FREQ_MINUTE_INTERVAL 指定的是 1，说明每隔一分钟就执行一次。STARTTIME 指定是从创建时开始，ENDTIME 指定 23:59:59，表示每天都是执行到这个时间为止，后面指定的 DURING_START_DATE 为 CURDATE，DURING_END_DATE 指定的为 NULL，表示这个调度指定的日期范围为从创建日期开始到永远，DESCRIBE 指定为'一个测试调度'，这是对这个调度的注释。

23.5.5 删除调度

如果某一个调度不需要了，则可以将其删除，调用的函数为 SP_DROP_JOB_SCHEDULE，其参数只有两个：

JOBNAME: 这个参数表示的是正在删除哪一个作业下的调度，这个参数必须为上面调用 SP_JOB_CONFIG_START 函数时指定的作业名，否则系统会报错，同时系统会检测这个作业是否存在，不存在也会报错。

SCHEDULENAME: 这个参数表示的要删除的调度的调度名，删除时会检测这个调度是否存在，如果不存在则报错。

例如删除作业 TEST 中名为 SCHEDULE3 的调度。

```
SP_DROP_JOB_SCHEDULE('TEST', 'SCHEDULE3');
```

23.5.6 配置提交

在配置过程中，可以对指定的作业增加、删除任意多个调度、步骤，但不要做提交操作以及自动提交操作，否则可能会出现作业配置不完整的问题。

在配置完成后，用户需要对前面所做的配置进行提交，表示对作业的配置已经完成，同时将这个作业加入到运行队列，可以通过系统函数 SP_JOB_CONFIG_COMMIT 实现，这个函数只有一个参数，即要提交配置的作业的名字。

调用这个函数时，系统会检测当前会话是否处于作业配置状态，如果不处于配置状态，则系统会报“非法的作业配置操作”的错误。

在执行配置提交操作后，系统会将前面所做的所有操作提交，同时将这个作业加入到运行队列，运行的内容包括这个作业下定义的所有步骤的执行内容，执行方式就是根据这个作业下定义的所有的调度定义的执行方式执行的。

23.6 创建删除修改警报

23.6.1 创建警报

达梦数据库还提供了警报的功能，允许用户定义一些自己想要捕获的动作，如果这些动作发生时，系统就会将对应信息存储到指定的表中，将想要得到的信息存储起来，以便做到实时的监控。

创建一个警报可以通过系统函数 `SP_CREATE_ALERT` 实现，这个函数的参数说明如下：

1. **NAME**：警报名，必须是有效的标识符，同时不能是 **DM** 关键字。警报不能同名，在创建时系统会检测这个警报是否存在，如果存在则报错；
2. **ENABLE**：警报是否开启；
3. **TYPE**：警报的类型，主要有以下几种：`ALERT_TYPE_ERROR`、`ALERT_TYPE_EVENT`，分别表示发生错误时警报及发生某些事件时警报，值分别为 0 和 1。创建时必须指定 0 或 1；
4. **ERRTYPE**：这个参数的值与上面的 **TYPE** 有关。如果 **TYPE** 为 `ALERT_TYPE_ERROR`，则 **ERRTYPE** 的值有以下几个可选项：`ALERT_ERROR_TYPE_NORMAL`、`ALERT_ERROR_TYPE_STARTUP`、`ALERT_ERROR_TYPE_SYSTEM`、`ALERT_ERROR_TYPE_CONFIG`、`ALERT_ERROR_TYPE_PARSE`、`ALERT_ERROR_TYPE_PRIV`、`ALERT_ERROR_TYPE_RUNTIME`、`ALERT_ERROR_TYPE_BACKRES`、`ALERT_ERROR_TYPE_JOB`、`ALERT_ERROR_TYPE_REPLICATE`、`ALERT_ERROR_TYPE_OTHER`、`ALERT_ERROR_TYPE_CODE`，分别表示常规错误、启动错误、系统错误、服务器配置错误、分析阶段错误、权限错误、运行时错误、备份恢复错误、作业管理错误、数据库复制错误、其它错误、指定错误码，值分别用 1、2、3、4、5、6、7、8、9、10、11、12 表示；如果 **TYPE** 指定的是 `ALERT_TYPE_EVENT`，则 **ERRTYPE** 有如下几个可选项：`ALERT_EVENT_TYPE_DDL`、`ALERT_EVENT_TYPE_PRIV`、`ALERT_EVENT_TYPE_CONNECT`、`ALERT_EVENT_TYPE_BACKUP`，分别表示 DDL 事件、授权回收事件、连接或断开数据库事件、数据库备份恢复事件，值分别用 1、2、3、4 表示；
5. **ERRCODE**：这个参数的值与上面的 **TYPE** 相关，如果 **TYPE** 指定的是 `ALERT_TYPE_ERROR`，则这个参数指定的必须是一个要捕获的错误码，此时 **ERRTYPE** 必须指定为 `ALERT_ERROR_TYPE_CODE` 时才有效，同时错误码必须是小于 0 的值；如果 **TYPE** 指定的是 `ALERT_TYPE_EVENT`，则这个参数指定的值与 **ERRTYPE** 有关，如果 **ERRTYPE** 为 `ALERT_EVENT_TYPE_DDL`，**ERRCODE** 可以是 `ALERT_DDL_TYPE_CREATE`、`ALERT_DDL_TYPE_ALTER`、`ALERT_DDL_TYPE_DROP`、`ALERT_DDL_TYPE_TRUNC` 中的任意多个的组合值，**ERRCODE** 只用最低四个位来表示它们的组合，低 1 位为 1 表示 `CREATE`，低 2 位为 1 表示 `ALTER`，低 3 位为 1 表示 `DROP`，低 4 位为 1 表示 `TRUNCATE`，如果为 0 表示不指定，所以这个参数的有效范围为 1 到 15；当 **ERRTYPE** 为 `ALERT_EVENT_TYPE_PRIV` 时，**ERRCODE** 可以是 `ALERT_PRIV_TYPE_GRANT` 和 `ALERT_PRIV_TYPE_REVOKE` 的组合，最低位为 1 表示 `GRANT`，0 为不指定，上一位为 1 表示 `REVOKE`，0 为不指定，所以这个参数的有效值范围为 1 到 3；如

果 ERRTYPE 的值为 ALERT_EVENT_TYPE_CONNECT，那么 ERRCODE 可以是 ALERT_CONNECT_TYPE_LOGIN 和 ALERT_CONNECT_TYPE_LOGOUT 的任意组合，这个参数的最低位为 1 表示 LOGIN，0 表示不指定，上一位为 1 表示 LOGOUT，0 为不指定，所以这个参数的有效值范围为 1 到 3；如果 ERRTYPE 为 ALERT_EVENT_TYPE_BACKUP，则 ERRCODE 有两个可选项，分别是 ALERT_BACKUP_TYPE_BACKUP 和 ALERT_BACKUP_TYPE_RESTORE，最低位为 1 表示 BACKUP，0 为不指定，上一位为 1 表示 RESTORE，0 为不指定，有效值范围为 1 到 3；

6. DELEYTIME：表示警报通知操作员时需要推迟的时间，范围为 0 到 3600 秒；
7. ADDITION_TXT：表示警报的注释字符串，最长为 500 个字节。

例如创建一个名为“ALERT1”的警报。

```
SP_CREATE_ALERT('ALERT1', 1, 0, 1, -600, 1, '错误码的测试');
```

上面的例子创建了一个名为 ALERT1 的警报，ENABLE 为 1 表示开启，TYPE 为 0 表示想要捕获的是 ALERT_TYPE_ERROR（错误），ERRTYPE 指定的是 1 表示 ALERT_ERROR_TYPE_NORMAL 普通错误，表示只有发生普通错误时警报才发生，DELEYTIME 指定可以推迟 1 秒钟通知。警报发生后，系统会将对应信息存储到上面提到的 SYSALERTHISTORIES 表中。

23.6.2 修改警报

修改警报可以通过系统函数 SP ALTER_ALERT 来实现，它的参数和 SP_CREATE_ALERT 的参数完全相同，除了 NAME 不可修改外，其他的属性都可修改。对于可修改参数，如果要修改，则指定新值；如果不修改，则指定原值即可。

例如下面的语句修改警报“ALERT1”；

```
SP ALTER_ALERT('ALERT1', 1, 1, 1, 15, 1, 'DDL 警报测试');
```

上面的例子修改警报 ALERT1，ENABLE 还是原来的值，没有被修改，TYPE 修改为 1，表示对事件的捕获，ERRTYPE 修改为 1，表示的是对事件中的 DDL 进行捕获，ERRCODE 修改为 15，从上面可以看出，这个是一个组合值，对于 DDL 事件共有四种类型（CREATE、ALTER、DROP、TRUNC），每一个位表示一个操作，这里的 15 可以被分解为 0x1111，可以看出四个位都为 1，也就是选中了这四个操作中的所有操作，DELEYTIME 没有被修改，ADDITION_TXT 被修改为'DDL 警报测试'。

修改后的警报只要系统做了 DDL 操作就会被激发，同时会将相应信息存储到上面提到的 SYSALERTHISTORIES 表中。

23.6.3 删除警报

删除警报通过调用系统函数 SP_DROP_ALERT 实现，参数为警报的名字。删除时系统会检测这个警报是否存在，如果不存在则报错。

例如下面的语句删除警报“LERT”；

```
SP_DROP_ALERT('ALERT1');
```

23.7 为警报关联操作员

23.7.1 关联

如果想要将警报的执行结果通知给指定操作员，需要将这个警报关联要通知到的操作员，关联操作员通过系统 `SP_ALERT_ADD_OPERATOR` 实现，其参数为：

1. **ALERTNAME**: 警报名，表示要将哪一个警报进行关联，在创建时系统会检测这个警报是否存在，如果不存在则报错；
2. **OPER_NAME**: 操作员名，表示要与哪一个操作员进行关联，在创建时系统会检测这个操作员是否已经存在，如果不存在则报错；
3. **ENABLEMAIL**: 表示指定警报的执行结果是否用邮件的方式通知指定的操作员，0 表示不用这种方式，1 表示通知；
4. **ENABLENETSEND**: 表示指定警报的执行结果是否用网络发送的方式通知指定的操作员，0 表示不用这种方式，1 表示通知。但这个功能只有在 **WINDOWS** 下起作用。

例如下面的语句指定 **TEST** 警报的执行结果用两种方式通知操作员 **WZF**。

```
SP_ALERT_ADD_OPERATOR('TEST', 'WZF', 1, 1);
```

一个警报可以关联多个操作员。

23.7.2 取消

当不再需要某个警报与操作员的关联，可以通过系统函数 `SP_ALERT_DROP_OPERATOR` 来取消关联其参数为：

1. **ALERTNAME**: 警报名，表示要将哪一个警报进行关联，在创建时系统会检测这个警报是否存在，如果不存在则报错；
2. **OPER_NAME**: 操作员名，表示要与哪一个操作员进行关联，在创建时系统会检测这个操作员是否已经存在，如果不存则会报错。

例如取消警报 **TEST** 与操作员 **WZF** 的关联，取消之后警报的执行结果不会再通知操作员 **WZF**。

```
SP_ALERT_DROP_OPERATOR('TEST', 'WZF');
```

23.8 综合例子

下面是作业管理的一个综合示例。

```
//创建操作员
SP_CREATE_OPERATOR('WZF', 1, 'tlf577@qq.com', '192.168.0.123');
//创建作业
SP_CREATE_JOB('TEST', 1, 1, 'WZF', 2, 1, 'WZF', 2, '一个测试作业');
//开始配置作业
SP_JOB_CONFIG_START('TEST');
//为作业增加一个步骤
```

```
SP_ADD_JOB_STEP('TEST', 'STEP2', 0, 'insert into myinfo values(1000, "STEP2 Hello
World");insert into myinfo values(1000, "STEP2 Hello World");', 0, 0, 1, 1, NULL, 0);
//为作业增加一个调度
SP_ADD_JOB_SCHEDULE('TEST', 'SCHEDULE3', 1, 1, 1, 0, 1, CURTIME, '23:59:59',
CURDATE, NULL, '一个测试调度');
//提交配置
SP_JOB_CONFIG_COMMIT('TEST');
//创建警报
SP_CREATE_ALERT('ALERT1', 1, 1, 1, 15, 1, 'DDL 警报测试');
```

第 24 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。

24.1 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动态表：CTI\$INDEX_NAME\$I 和 CTI\$INDEX_NAME\$P，其表结构如 24.1 和 24.2 所示。第一个表主键为（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；第二个表主键为（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。

表 24.1 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	BLOB	2G-1	2G-1	0	保存所在文档所有的 ROWID，连续存放

表 24.2 CTI\$INDEX_NAME\$P 表结构

序号	字段名	类型	长度	精度	刻度	说明
1	PND_DOCID	BIGINT	8	19	0	文档 ROWID
2	PND_YPTE	SMALLINT	2	5	0	INS/UPD/DEL 类型

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，2 个自动生成的动态表 `CTI$ CTI_ADDRESS$I` 和 `CTI$ CTI_ADDRESS$P` 信息保存在 `SYSOBJECTS` 系统表中。这两个表和普通表一样，具有相关权限的用户可以进行查询、插入、更新和删除操作。

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 4 种分词参数：

1. `CHINESE_LEXER`，中文最少分词；
2. `CHINESE_VGRAM_LEXER`，中文最多分词；
3. `ENGLISH_LEXER`，英文分词；
4. `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。所以对存在中文的文本进行英文分词时会报分词参数指定错误。

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。

24.2 创建全文索引

在 DM 中，全文索引必须在基表上定义，而不能在视图、临时表、`LIST` 表、分区表、外部表上定义，同时，表必须未自定义聚集主键。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 DM《SQL 语言使用手册》。

24.3 更新全文索引

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。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并没有产生相应的日志，即无法实现更新的回滚，如果在更新过程中发生任何错误，就必须执行完全更新获得正确的一致性的全文索引信息。还需要注意的是当对已创建全文索引的基表进行批量数据修改（例如查询插入、批量导入等）时，需进行完全更新才能构建数据一致的新的全文索引，因为这些批量修改的数据信息未保存在全文索引动态信息表中，所以此时执行增量更新操作起不到更新全文索引的作用。

24.4 执行全文检索

全文检索根据查询关键字或者句子，从全文索引中查询满足条件的记录。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 表中地址在洪山区太阳城的记录：

```
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%';
```

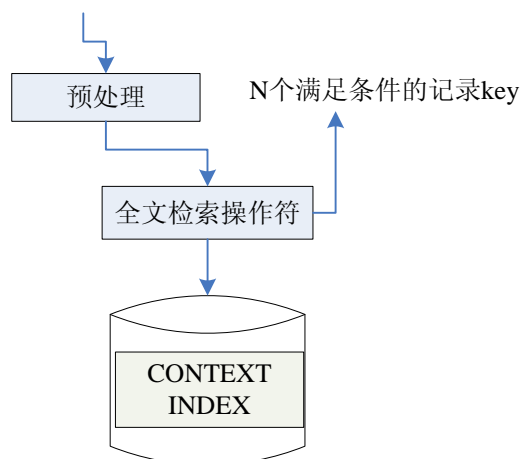


图 24.1 全文查询

如图 24.1 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。

24.5 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 25 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。

25.1 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，隐式的启动一个事务，以 COMMIT 或 ROLLBACK 语句/方法显式地结束事务。另外，在配置为 DDL 自动提交的情况，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 语句本身所属事务也会隐式地结束。

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，使帐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帐户间的平衡。

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。

25.2 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 4 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。

25.2.1 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。

25.2.2 一致性

数据一致性是指表示客观世界同一事物状态的数据，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。

25.2.3 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 25.7 节。

25.2.4 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。

25.3 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条目，重做日志条目中包含了对数据块和回滚块所进行的修改操作，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. 释放加于表或数据行上的锁，将事务标记为完成；
3. 返回提交成功消息给发出提交请求者。

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。

25.3.1 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;
```

25.3.2 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。

25.3.3 隐式提交模式

在手动提交模式下，如果服务器配置参数中的 DDL_AUTO_COMMIT 值为 1，则当遇

到某些类型的语句时，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当前语句，当前语句执行完毕，也会自动提交，并在遇到下一个语句时，自动开始一个新的事务。这种提交模式被称为隐式提交模式，相应的事务被称为隐式事务。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。

25.4 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。

25.4.1 自动回滚

若事务运行期间出现严重错误，如网络中断、客户端应用程序异常，DM 数据库都会自动回滚该事务。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。

25.4.2 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。

25.4.3 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，但其后创建的保存点都将被清除；
3. 释放此保存点之后获得的表级锁与行级锁，但该保存点之前的数据锁依然保留。

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`。

25.4.4 语句级回滚

如果在一个 SQL 语句执行过程中发生了错误，那么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，而不会导致当前事务中此语句之前的语句所做的数据修改失效。

在语句的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。而在 SQL 语句解析的过程中发生错误（例如语法错误），由于此语句还没有被执行，所以不会产生语句级回滚。

25.5 事务锁定

数据库是一个多用户使用的共享资源。当多个用户并发地存取数据时，在数据库中就会产生多个事务同时存取同一数据的情况。若对并发操作不加控制就可能会读取和存储不正确的数据，破坏数据库的一致性。

加锁是实现数据库并发控制的一个非常重要的技术。当事务在对某个数据对象进行操作前，先向系统发出请求，锁定正被修改的数据，加锁后事务就对该数据对象有了一定的控制，在该事务释放锁之前，其他的事务不能对此数据对象进行更新操作。在用户提交或回滚事务之后，锁会自动释放，只有在提交或回滚事物之后，其他用户才可以更新这些数据。

DM 数据库也采用锁定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，包括锁的各种类型以及如何查看锁等。

25.5.1 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 4 种不同的锁模式：共享锁、排他锁、意向共享锁和意向排它锁。

1. 共享锁

共享锁（Share Lock，简称 S 锁）模式用于只读操作。这种模式允许并发事务同时读取相同的资源，但是不允许任何事务修改这个资源。读取结束后会释放共享锁，除非将隔离级别设置为可串行化隔离级。

2. 排他锁

排它锁（Exclusive Lock，简称 X 锁）用于修改数据的操作，如插入、更新和删除。当某个事务占用某个资源上的排他锁时，其他事务就不能修改此数据（由于采用多版本并发

控制，读取操作仍然可以进行)，这种锁模式防止并发用户同时更新相同的数据，否则会造成数据不一致或者不正确的现象。

3. 意向锁

表是由行组成的，当我们向某个表加锁时，一方面需要检查该锁的申请是否与原有的表级锁相容，另一方面，还要检查该锁是否与表中的每一行上的锁相容。比如一个事务要在一个表上加 S 锁，如果表中的一行已被另外的事务加了 X 锁，那么该锁的申请也应被阻塞。如果表中的数据很多，逐行检查锁标志的开销将很大，系统的性能将会受到影响。为了解决这个问题，可以在表级引入新的锁类型来表示其所属行的加锁情况，这就引出了“意向锁”的概念。

意向锁的含义是如果对一个结点加意向锁，则说明该结点的下层结点正在被加锁。对任一结点加锁时，必须先对它的上层结点加意向锁。如：对表中的任一行加锁时，必须先对它所在的表加意向锁，然后再对该行加锁。这样一来，事务对表加锁时，就不再需要检查表中每行记录的锁标志位了，系统效率得以大大提高。

由两种基本的锁类型（S 锁、X 锁），可以自然地派生出两种意向锁：

- 1) 意向共享锁（Intent Share Lock，简称 IS 锁）：如果要对一个数据库对象加 S 锁，首先要对其上级结点加 IS 锁，表示它的下层结点拟（意向）加 S 锁；
- 2) 意向排它锁（Intent Exclusive Lock，简称 IX 锁）：如果要对一个数据库对象加 X 锁，首先要对其上级结点加 IX 锁，表示它的下层结点拟（意向）加 X 锁。

这样 DM 数据库对象上所加的锁类型就有 4 种：即 S、X、IS、IX。具有意向锁的多粒度封锁方法中任意事务 T 要对一个数据库对象加锁，必须先对它的上层结点加意向锁。申请封锁时应按自上而下的次序进行，释放封锁时则应按自下而上的次序进行，具有意向锁的多粒度封锁方法提高了系统的并发度，减少了加锁和解锁的开销。

四种模式锁的相容矩阵如表 25.1 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 25.1 四种表锁相容矩阵

	IS	IX	S	X
IS	Y	Y	Y	N
IX	Y	Y	N	N
S	Y	N	Y	N
X	N	N	N	N

25.5.2 锁粒度

按照封锁对象的不同，锁可以分为行锁、表锁和字典锁。

1. 行锁

行级锁只对用户正在访问的行进行锁定。如果该用户正在修改某行，那么其他用户就可以更新同一表中该行之外的数据，其他用户不能修改该行，只能查询该行。

行级锁是一种排他锁（X 锁），防止其他事务修改此行，但是不会阻止读取此行的操作。

在使用 INSERT、UPDATE、DELETE 等语句时，DM 数据库会自动应用行级锁锁定，该锁保持到事务被提交或回滚。当两个或多个会话在表的同一行上执行 DML 语句时，第一个会话在该行上加锁，其他的会话处于等待状态。当第一个会话提交后，X 锁被释放，其他会话才可以加锁。

为了减少锁的数量，降低事务系统的开销，DM7 采用了多个连续行共用一把锁的机制，实现并发度与系统资源之间的平衡。并且在多版本机制中，读操作不需要上锁，进一步减少了锁的数量，减少了系统资源的占用。DM7 取消了范围锁和锁升级机制，避免插入过程中，将行锁升级为表锁后，导致并发度大大降低的情况。

2. 表锁

表级锁保护表数据，在事务处理过程中，表级锁会在不同程度上限制对整个表的访问。一般而言，DM 数据库的隐式封锁足以保证数据的一致性，但用户可以根据自己的需要显式改变对表的封锁。显式锁定表的语法如下：

```
LOCK TABLE <table_name> IN <lock_mode> MODE;
```

lock_mode 是锁定的模式，可以选择的模式有 INTENT SHARE（意向共享）、INTENT EXCLUDE（意向排他）、SHARE（共享）和 EXCLUDE（排他），其含义分别如下：

- 1) 意向共享：允许其他用户访问和锁定该表，但是禁止排他锁定整个表。意向共享锁锁定后，在同一时刻，不同用户可以对同一个表中的数据具备增、删、改、查的功能；
- 2) 意向排他：允许其他用户访问和锁定该表，同时禁止其他用户在此表上使用共享锁和排他锁。被意向排他后，其他用户可以查询当前用户锁定的行，但是不能同时修改当前用户锁定的行；
- 3) 共享：共享锁将锁定表，仅允许其他用户查询表中的行，但不允许插入、更新或删除行。多个用户可以同时在同一个表中放置共享锁，即允许资源共享，因此得名“共享锁”。例如，如果用户每天都需要在结帐时更新日销售表，则可以在更改该表时使用共享锁以确保数据的一致性；
- 4) 排他：排它锁锁定整个表，禁止其他事务对表做任何更改或在表上应用任何类型的锁。

3. 字典锁

用户执行数据库操作时，会隐式地对访问的字典对象（比如模式对象、表对象等）上锁，对字典对象上的锁，称之为字典锁。字典锁的实现方式与表锁保持一致。

25.5.3 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、表 ID 以及行 ID 等。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
157089424	278	DICT	S	0	150994944	0
157089552	278	TABLE	IS	0	1077	0
157089040	278	TABLE	IX	0	1077	0
157089680	278	DICT	IS	0	301	0

157088912	278	DICT	S	0	301	0
157089168	278	ROW	X	0	1077	0

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 DICT（字典锁）、TABLE（表锁）、ROW（行锁）；LMODE 列表示锁的模式，可能的取值有 S（共享锁）、X（排它锁）、IS（意向共享锁）、IX（意向排它锁）；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 列是行锁的 ROWID 的索引值，即 ROW_IDX 是把 8 字节的 ROWID 中最后 1 个字节的低 6 位置为 0 后的值。

25.6 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，写操作上排它锁，这种锁机制虽然解决了并发问题，但大大影响了并发性。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。实现多版本以后，数据库所有的读操作将不需要上锁，即非阻塞查询，更新操作仍上写锁。数据库在内部维护记录的多个版本，事务将在自己特定的版本上操作。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，以及回滚段实现多版本访问控制，数据页中只保留物理记录的最新版本，通过回滚段中保存的回滚记录，还原特定版本的物理记录。

25.6.1 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：RTID 和 RPTR。RTID 保存最后修改记录的事务号，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	RTID	RPTR
------	------	------

25.6.2 回滚记录格式

回滚记录与物理记录一样，增加了两个字段：RTID 和 RPTR。RTID 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	RTID	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，新物理记录的 RPTR 指向当前回滚记录的物理地址。

25.6.3 可见性原则

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。DM7 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。DM7 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。

25.6.4 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 RTID 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：

物理记录，字段值为（1，‘XYZ’）	RTID（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’）	RTID(T1 的 ID)	RPTR
-------------------------	---------------	------

25.6.5 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。

DM7 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 RTID，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。

25.7 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。如果一个事务在提交结果之前，另一个事务可以修改和删除它，就会发生不可重复读情况。

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 25.2 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 25.2 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的模式执行时，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。

25.7.1 读一致性

DM 数据库提供的读提交和串行化隔离级都能实现高度的数据一致性及并发访问能力。这两种隔离级别都能够利用 DM 数据库的读一致性多版本并发访问控制模型及行锁技术，减少并发事务间的竞争。应用程序开发者可以使用这两种隔离级别开发符合现实要求的系统。

在读提交和串行化隔离级下，DM 数据库强制实现查询语句的读一致性，这保证了单一查询的结果集来自一个时间点，即查询开始执行的时间。因此，一个查询的结果集永远不会包含脏数据以及此查询执行时其他事务提交的数据，只有在查询执行前提交的数据对此查询才是可见的，查询语句无法看到其开始执行后提交的数据。SELECT 语句、使用子查询的 INSERT、UPDATE、DELETE 语句等，不管显式还是隐式查询，都能够保证数据一致性。

换句话说，这些查询只能看到其所在 DML 语句开始之前的数据。但是这种一致性不能保证可重复读，也不能阻止出现幻像读。

除了查询语句的读一致性，DM 数据库还能够实现事务级的读一致性。当一个事务运行在串行化隔离级下时，则事务内所有数据访问均反映的是事务开始时的数据状态。即事务内的所有查询对某个时间点（事务开始的时间）来说具备一致性，运行在串行化模式下的事务只能够看到事务自身对数据所作的修改。事务级的读一致性能够保证可重复读并可阻止出现幻像读。

在读提交隔离模式及串行化隔离模式下执行的事务都采用行锁技术，它们在更新被未提交的并发事务修改的数据行时都会发生等待，等待未提交的并发事务提交或撤销，并释放锁。当导致阻塞的事务（即未提交的并发事务）提交并释放了锁后，运行在已提交读写模式下的等待事务就能够继续执行其中的更新操作。而运行在串行化模式下的等待事务将出现无法进行串行化访问错误，因为阻塞事务在串行化等待事务开始后更新了后者所存取的数据。

在 DM 数据库行级锁及多版本并发访问控制系统的基础上，DM 数据库提供的读提交和串行化隔离级两种事务隔离级别都能够实现高一致性、高并发性及高性能。读提交和串行化两种隔离模式都能提供高度的并发访问能力，从而保证系统的高性能，同时查询不会读取未提交的数据。

应用程序的设计、开发者应当根据应用程序的性能要求、数据一致性要求，以及应用程序的编码需求来决定选择何种隔离级别。对于存在大量并发用户快速地提交事务的系统来说，应用程序设计者应该从事务处理量及响应时间的角度评估事务处理的性能。通常来说，为一个对性能要求高的系统选择事务隔离级别时，需要在数据一致性及数据并发处理间进行平衡。

另外，DM 数据库提供的读未提交不能做到数据的读一致性，建议用户小心使用。

25.7.2 读提交隔离级

对于大多数应用来说，读提交隔离级是最适合的事务隔离级别。读提交隔离能够最大限度地保证数据并发性，但在某些事务中可能会出现不可重复读取或幻像读，因此略微增加了出现数据不一致性的风险。在对性能要求较高的系统中，为了应对较高的事务到来率，系统需要提供更大的事务吞吐量和更快的响应速度，此时采用串行化隔离可能难以实现。还有一类系统，其事务到来率较低，出现不可重复读取或幻像读的风险也较低。以上两种系统均适合采用读提交隔离。

DM 数据库的读提交隔离能够确保所有查询的语句级数据一致性。即查询获得的数据是处于一致性状态下的。因此在 DM 中读提交隔离能够满足大多数应用的要求。而在没有多版本并发访问控制的数据库管理系统中，开发者可能需要采用更高级别的隔离方式。

在读提交隔离模式下，开发者不需要在应用逻辑中捕获无法进行串行化访问错误，也无需回滚并重新执行事务。在大多数应用程序中，几乎不会有在一个事务中执行同一查询多次的情况，因此在这些应用程序中，为防止出现不可重复读取或幻像读而采取的保护措施并不重要。如果开发者选择读提交隔离，就能够省略在每个事务中加入错误检查及事务重做的代码。

用户可以在事务开始时使用以下语句设定事务的隔离级别为读提交隔离级：

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

25.7.3 串行化隔离级

用户应该只在读提交隔离级无法满足需要时，才考虑可串行化隔离级。而且应该是仅对必须使用可串行化隔离级的事务设置为可串行化隔离级，其他事务应仍保持在读提交隔离级，这样可以提供更高的性能和并发度。最适合采用串行化隔离的系统是大型数据库，且其中主要运行更新少量数据的短小事务。

串行化隔离能够提供更好的数据一致性，它能阻止不可重复读取或幻像读的现象。当一个读或写事务中需要运行同一查询多次时，串行化隔离的作用更加明显。但是，在串行化隔离下，并发事务对数据库进行修改时只能顺序执行。具体来说，在串行化隔离下，DM 数据库在允许一个采用串行化隔离的事务修改某些数据行时，需要判断在此事务开始执行之前，其他所有事务对这些数据行的修改已经被提交。当一个串行化事务试图更新或删除数据，而这些数据在此事务开始后被其他事务修改并进行了提交时，DM 数据库将报错：串行化事务被打断。

某些数据库管理系统在实现串行化隔离时，无论读写操作都要对整个数据块加锁。而 DM 则采用了无阻塞查询及低粒度的行级锁技术，减少了读写操作间的竞争。对于存在较多读写竞争的应用，DM 数据库的串行化隔离与其他数据库管理系统相比能够大大地提高事务处理能力。因此，某些应用在 DM 数据库中可以采用串行化隔离，而在其他数据库管理系统则未必可行。

运行在串行化隔离模式下的事务中的所有查询所获得的数据都来自同一时间点，因此这种隔离级别适合于需要执行多个满足一致性的查询的事务。例如，汇总数据并将结果写入数据库的报表应用可以采用串行化隔离。

在使用串行化隔离事务时，应用开发者必须编码来捕获无法进行串行化访问错误，之后回滚并重做事务。在其他数据库管理系统中需要类似的代码来处理死锁。有时为了遵从已有系统的标准，或者当应用可能运行在多种数据库管理系统上时，应按照串行化隔离的要求来设计事务处理代码。具备检查并处理串行化错误功能的事务也可以运行在 DM 数据库的读提交隔离模式下，因为此种隔离模式下不会产生串行化错误。

如果系统中存在长时间运行的写事务，且其所操作的数据同时还会被大量的小事务更新，则此类系统不应采用串行化模式。因为长事务所需更新的数据可能会被其他事务抢先更新，则长事务可能需要重复地回滚，浪费系统资源。需要注意的是，其他数据库管理系统所实现的串行化隔离（使用读取锁）同样不适合上述情况，因为长事务（即便是只执行读取操作的事务）会和短小的写事务相互阻塞。

应用开发者在采用串行化隔离时应考虑回滚及重做事务所带来的开销，而在采用读取锁的数据库管理系统中，死锁会频繁出现，在这样的系统中采用串行化隔离，必须令因死锁而终止的事务回滚并重做。如果一个系统对数据访问的竞争较激烈，那么处理串行化错误将消耗大量资源。

在大多数系统中，一个事务发生无法进行串行化访问错误后重做时再次与其他事务冲突的几率较小。基于上述原因，采用串行化隔离时，容易与其他事务产生竞争的语句应该在事务开始后尽早执行。但是我们始终无法保证事务能够成功执行，因此在应用程序中应该限制重做的次数。

尽管 DM 数据库的串行化隔离模式与 SQL92 兼容，而且与采用读取锁的实现方式相比具备很多优点，但是 DM 数据库串行化隔离所包含的语义与其他数据库管理系统不完全相同。应用开发者必须注意，与其他数据库管理系统不同，在 DM 数据库中读操作不会阻塞写操作。在应用级对数据库一致性进行检查的事务需要使用 `SELECT FOR UPDATE` 之类的

技巧才能避免错误。当从其他数据库管理系统向 DM 数据库迁移应用时，尤其要注意上述问题。

用户可以在事务开始时使用以下语句设定事务的隔离级别为串行化隔离级：

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

25.7.4 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，仅当另一个事务试图删除或更改被检索的行所在的表时，才会锁定一个事务检索的行。因为在使用这种隔离级别时，行通常保持未锁定状态，所以脏读、不可重复读和幻像读都可能会发生。因此，读未提交隔离级别通常用于那些访问只读表和视图的事务，以及某些只执行 SELECT 语句的事务(只要其他事务的未提交数据对这些语句没有负面效果)。

其他事务对行所做的更改在提交之前对于使用读未提交隔离级别的事务是可见的。此类事务可以看见或访问其他事务 DDL(CREATE、ALTER 和 DROP)语句所创建的表、视图或索引，即使那些事务没有提交。类似地，如果其他事务删除了现有的表、视图或索引，那么使用读未提交隔离级别的事务就能知道这些对象不再存在了。

用户可以在事务开始时使用以下语句设定事务的隔离级别为读未提交隔离级：

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
```

25.7.5 只读事务

除了前面所述的各种标准特性外，达梦数据库还支持设置事务只读属性的语句，使用该语句后该事务只能做查询操作，不能更新数据库。该语句只能在事务开始时进行，并且该语句不会改变原先设定的事务隔离级。

设定只读事务的语句如下：

```
SET TRANSACTION READ ONLY;
```

25.8 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 的排它锁，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就是用户拥有一个建有主键约束的表。当 2 个会话同时试图向表中插入相同的数据时，其中的一个会话将被阻塞，直到另外一个会话提交或回滚。一个会话提交时，另一个会话将收到主键重复的错误。回滚时，被阻塞的会话将继续执行。

当执行 UPDATE 和 DELETE 操作的数据行已经被另外的会话锁定时，将会发生阻塞，

直到另一个会话提交或会滚。

25.9 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 **FLASHBACK_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 26 章 管理安全性

26.1 概述

数据库的安全性是指保护数据库以防止不合法的使用所造成的数据泄露、更改或破坏。安全性问题不是数据库系统所独有的，计算机系统都有这个问题，只是在数据库系统中大量数据集中存放，而且为许多用户直接共享，是宝贵的信息资源，从而使安全性问题更为突出。

数据库安全性保护措施是否有效是衡量数据库系统的重要指标之一。DM 的安全管理就是为保护存储在 DM 数据库中的各类敏感数据的机密性、完整性和可用性提供必要的技术手段，防止对这些数据的非授权泄露、修改和破坏，并保证被授权用户能按其授权范围访问所需要的数据。

本章介绍的主要内容包括：用户管理、权限管理、角色管理、标记管理（仅安全版提供）、审计、通信加密、存储加密、资源限制、客体重用（仅安全版提供）、加密引擎。这些内容属于高级系统管理范畴，是系统管理员应该掌握的内容。安全管理对信息敏感部门尤为重要，应用系统设计者和系统管理员必须熟练掌握系统的安全特性，以便建立高安全性的数据库应用系统。

虽然本章所讨论的内容是针对 DM 的，但其原理和方法与其它系统大同小异，因而，对于了解和掌握其它数据库管理系统的的海理也是有帮助的。

26.2 用户管理

用户是一个数据对象，是一系列数据库对象和权限的统称。用户所有的操作默认在自己的模式下进行，模式是一个用户所拥有的数据库对象的集合，每个用户都有自己的默认模式，用户默认模式的名字与用户名相同。例如，SYSDBA 用户的默认模式为 SYSDBA，在这个模式中包含了用户 SYSDBA 拥有的数据库对象，包括表、视图、索引、存储过程等。

26.2.1 DM 中有哪些管理用户

DM 首次提出了“三权分立”的概念，在 DM 数据库中有三类管理用户，系统管理员（SYSDBA），系统安全员（SYSSSO）和系统审计员（SYSAUDITOR）。SYSDBA 负责配置达梦数据库参数、管理数据库对象、自主访问权限的分配、数据导入导出以及数据库的备份和恢复等职责；SYSSSO 负责对系统进行强制访问控制，定义新的数据库安全员；SYSAUDITOR 负责设置审计策略。

“三权分立”的思想是将系统中所有的权限按照类型进行划分，为每个管理员分配相应的权限，管理员之间的权限互不相交，从而达到相互制约、相互协助的目的，具有较高的安全性和较强的灵活性。

26.2.2 如何创建用户

数据库系统在运行的过程中，往往需要根据实际需求创建用户，然后为用户指定适当的权限。创建用户的操作一般只能由系统预设用户 **SYSDBA**、**SYSSSO** 和 **SYSAUDITOR** 完成，如果普通用户需要创建用户，必须具有 **CREATE USER** 的数据库权限。

创建用户的命令是 **CREATE USER**，创建用户所涉及的内容包括为用户指定用户名、认证模式、口令、口令策略、空间限制、只读属性以及资源限制。其中用户名是代表用户帐号的标识符，它的命名规则是：

1. 必须以字符开始；
2. 长度为 1~128 个字符；
3. 从第二个字母开始，可以包括大小写字母、数字、_、\$和#等字符。

用来创建用户的 **CREATE USER** 命令的完整语法格式为：

```
CREATE USER 用户名 IDENTIFIED BY "口令"
PASSWORD POLICY 口令策略
DISKSPACE LIMIT 空间限制
[NOT] READ ONLY
LIMIT 资源限制
ALLOW IP 地址
NOT ALLOW IP 地址
ALLOW DATETIME 时间
NOT ALLOW DATETIME 时间
```

口令策略、地址限制以及时间限制只有安全版本才提供此功能的支持。其中 **PASSWORD POLICY** 用来指定口令策略，系统支持的口令策略有：

1. 0 无策略
2. 1 禁止与用户名相同
3. 2 口令长度不小于 6
4. 4 至少包含一个大写字母 (A-Z)
5. 8 至少包含一个数字 (0-9)
6. 16 至少包含一个标点符号

口令策略可单独应用，也可组合应用。组合应用时，如需要应用策略 2 和 4，则设置口令策略为 2+4=6 即可。

DISKSPACE LIMIT 用于为用户指定空间配额，单位为 M。也可以指定为 **UNLIMITED**，即无限制的空间配额。

只读标记表示该用户是否只能对数据库作只读操作，默认为可读写。

关于 **LIMIT** 的限制项，在 26.10 小节中有详细介绍，这里介绍其中一些：

1. **SESSION_PER_USER**：一个用户所允许的并发会话的数目，默认为无限制；
2. **CONNECT_IDLE_TIME**：一个用户会话所能持续的时间，以分钟为单位。超过这个时间，会话将自动断开。默认为无限制；
3. **FAILED_LOGIN_ATTEMPS**：允许的失败登录次数。如果用户的失败登录达到指定的次数，用户帐号将被锁定。默认为无限制；
4. **PASSWORD_LOCK_TIME**：帐号锁定时间。当用户失败的登录达到指定的次数后，用户帐号将被锁定指定的分钟数。默认值为 1 分钟；
5. **PASSWORD_LIFE_TIME**：口令的有效期。当用户口令到达有效期后，必须进行修

改。单位为天，默认为无限制；

6. **PASSWORD_GRACE_TIME**: 口令有效的延长期。对口令的有效期可以指定延长期，在延长期内用户仍然可以登录。默认值为 10 天；
7. **PASSWORD_REUSE_MAX**: 为了再次使用过去用户的口令，必须经过不同口令的次数。默认无限制；
8. **PASSWORD_REUSE_TIME**: 为了再次使用过去用过的口令，必须经过的天数。默认无限制。

其中最后两个参数一起使用。如果为两个参数都指定了具体的值，那么在指定的天数内，用户的口令必须被修改过指定的次数后，才能使用过去用过的口令。如果把两个参数都设置为 **UNLIMITED**，那么对用户的口令没有限制。

例如，下面创建对失败登录次数进行控制的用户，如果用户失败的登录次数达到 3 次，这个用户帐号将被锁定。

```
SQL> CREATE USER USER1 IDENTIFIED BY PASSWORD  
LIMIT FAILED_LOGIN_ATTEMPTS 3, PASSWORD_LOCK_TIME 5;
```

又如，下面创建对修改口令进行限制的用户，要求用户在 30 天内必须把口令修改过 5 次后，才能使用过去用过的口令。

```
SQL> CREATE USER USER2 IDENTIFIED BY PASSWORD  
LIMIT PASSWORD_REUSE_TIME 30, PASSWORD_REUSE_MAX 5;
```

26.2.3 如何修改用户信息

为了防止不法之徒盗取用户的口令，用户应该经常改变自己的口令。用户的口令不应该是类似 123455，abcdef 这样简单的字符串，更不要指定为自己的生日和姓名，也不要指定为一个英文单词，因为这样的口令很容易被破解。一个好的口令应该包含大小写字母、数字、_、#、\$等符号在内的混合字符串。统计表明，一个口令中包含的成分越复杂，就越难破译。

修改用户口令的操作一般由用户自己完成，**SYSDBA**、**SYSSSO**、**SYSAUDITOR** 可以无条件修改同类型的用户的口令。普通用户只能修改自己的口令，如果需要修改同类型其他用户的口令，必须具有 **ALTER USER** 这个数据库权限。

修改用户口令的命令是 **ALTER USER**，格式为：

```
ALTER USER 用户 IDENTIFIED BY "口令";
```

ALTER USER 命令除了修改用户口令外，还可以修改用户的口令策略、空间限制、只读属性以及资源限制。下面语句修改用户 **user1** 的空间限制为 20M：

```
SQL> ALTER USER user1 DISKSPACE LIMIT 20;
```

26.2.4 如何删除用户

一个用户不再访问数据库系统时，应该将这个用户及时地从数据库中删除，否则可能会有安全隐患。

删除用户的操作一般由 **SYSDBA**、**SYSSSO**、**SYSAUDITOR** 完成，也可以由具有 **DROP USER** 权限的用户来完成。一个用户被删除后，这个用户本身的信息，以及它所拥有的数据库对象的信息都将从数据字典中被删除。删除用户的命令是 **DROP USER**，语法格式为：

```
DROP USER 用户名;
```

例如，要删除用户 **user1**，可以执行下面的 **DROP USER** 语句：

```
SQL> DROP USER user1;
```

需要注意的是，如果一个用户的模式中已经包含一些数据库对象，那么这个用户是不能被直接删除的，在删除用户时系统将给出如下的错误信息：

```
试图删除被依赖对象[user1].error code=-1307
```

出现错误信息的原因是在用户 `user1` 的模式中包含某些数据库对象、索引不能直接删除。如果要强制删除用户，可以在 `DROP USER` 命令中使用 `CASCADE` 选项，这样用户以及所拥有的数据库对象将被一起删除。例如，删除用户 `user1` 可以执行下面的语句：

```
SQL> DROP USER user1 CASCADE;
```

26.3 权限管理

DM 数据库对用户的权限有着严密的规定，如果没有权限，用户将无法完成任何操作。用户权限有两类，即数据库权限和对象权限。数据库权限主要是指针对数据库对象的创建、删除、修改的权限，对数据库备份等权限。而对象权限主要是指对数据库对象中的数据的访问权限。数据库权限一般由 `SYSDBA`、`SYSAUDITOR` 和 `SYSSSO` 指定，也可以由具有特权的其他用户授予。对象权限一般由数据库对象的所有者授予用户，也可由 `SYSDBA` 用户指定，或者由具有该对象权限的其他用户授权。

26.3.1 数据库权限的管理

数据库权限是与数据库安全有关的最重要的权限，这类权限一般是针对数据库管理员的。数据库权限的管理主要包括权限的分配、回收和查询等操作。DM 提供了 100 余种数据库权限，表 26.1 列出了与用户有关的最重要的几种数据库权限。

表 26.1 常用的数据库权限

数据库权限	说明
CREATE TABLE	在自己的模式中创建表的权限
CREATE VIEW	在自己的模式中创建视图的权限
CREATE USER	创建用户的权限
CREATE TRIGGER	在自己的模式中创建触发器的权限
ALTER USER	修改用户的权限
ALTER DATABASE	修改数据库的权限
CREATE PROCEDURE	在自己模式中创建存储程序的权限

对于表、视图、用户、触发器这些数据库对象，有关的数据库权限包括创建、删除和修改他们的权限，相关的命令分别是 `CREATE`、`DROP` 和 `ALTER`。表、视图、触发器、存储程序等对象是与用户有关的，在默认情况下对这些对象的操作都是在当前用户自己的模式下进行的。如果要在其他用户的模式下操作这些类型的对象，需要具有相应的 `ANY` 权限。例如，要能够在其他用户的模式下创建表，当前用户必须具有 `CREATE ANY TABLE` 数据库权限，如果希望能够在其他用户的模式下删除表，必须具有 `DROP ANY TABLE` 数据库权限。

数据库权限的授予者一般是数据库管理员。普通用户被授予了某种数据库权限及其转授权时，系统允许它把所拥有的数据库权限再授予其他用户。

为用户授予权限的 `GRANT` 命令的语法格式为：

```
GRANT 权限 1, 权限 2,.....
```

```
TO 用户 1, 用户 2,.....
```

WITH ADMIN OPTION;

GRANT 命令执行后，所有指定用户都将获得指定的权限。如果希望把一个权限授予所有用户，可以用 PUBLIC 代替所有的用户名。选项"WITH ADMIN OPTION"的功能是使得权限的获得者可以再将权限授予其他用户。

例如，刚刚创建用户 user1，这个用户没有任何权限。如果把 CREATE TABLE 权限授予这个用户，那么他就可以在自己的模式中创建表了。授予这个权限的语句为：

SQL> GRANT CREATE TABLE TO user1;

这样，用户 user1 在登录数据库系统后便可以创建表了。

回收数据库权限的命令是 REVOKE，这条命令一般由 SYSDBA 执行。如果一个用户在接受某个数据库权限时是以"WITH ADMIN OPTION"方式接受的，他随后又将这个数据库权限授予了其他用户，那么他也可以将这个数据库权限从其他用户回收。REVOKE 的语法格式为：

REVOKE 数据库权限 1，数据库权限 2....

FROM 用户 1，用户 2....

例如，要将刚才授予用户 user1 的数据库权限 CREATE TABLE 回收，可以执行下面的 REVOKE 语句：

SQL> REVOKE CREATE TABLE FROM user1;

这样，用户 user1 就不能创建表了。

需要注意的是，数据库权限可以转授，但是回收时不能间接回收。假设有三个用户，第一个用户将某个数据库权限以“WITH ADMIN OPTION”的方式授予第二个用户，第二个用户将这个权限授予第三个用户，那么当地一个用户从第二个用户回收这个权限时，并不能同时从第三个用户回收这个权限，第三个用户这时仍然具有这个权限。

26.3.2 对象权限的管理

对象权限主要是对数据库对象中的数据的访问权限，这类权限主要是针对普通用户的。表 26.2 列出了主要的对象权限。

表 26.2 常用的对象权限

对象权限 数据库对象类型	表	视图	序列	存储程序
ALTER	√			
SELECT	√	√	√	
INSERT	√	√		
DELETE	√	√		
UPDATE	√	√		
REFERENCES	√			
EXECUTE				√
GRANT	√	√	√	√

SELECT、INSERT、DELETE 和 UPDATE 权限分别是针对数据库对象中的数据的查询、插入、删除和修改的权限。对于表和视图来说，删除操作是整行进行的，而查询、插入和修改却可以在一行的某个列上进行，所以在指定权限时，DELETE 权限只要指定所要访问的表就可以了，而 SELECT、INSERT 和 UPDATE 权限还可以进一步指定是对哪个列的权限。

REFERENCES 权限是指可以与一个表建立关联关系的权限，如果具有了这个权限，当

前用户就可以通过自己的一个表中的外键，与对方的表建立关联。关联关系是通过主键和外键进行的，所以在授予这个权限时，可以指定表中的列，也可以不指定。

EXECUTE 权限是指可以执行存储函数、存储过程的权限。有了这个权限，一个用户就可以执行另一个用户的存储程序。

当一个用户获得另一个用户的某个对象的访问权限后，以“模式名.对象名”的形式访问这个数据库对象。一个用户所拥有的对象和可以访问的对象是不同的，这一点在数据字典视图中有所反映。在默认情况下用户可以直接访问自己模式中的数据库对象，但是要访问其他用户所拥有的对象，就必须具有相应的对象权限。

对象权限的授予一般由对象的所有者完成，也可由 **SYSDBA**，或者具有某对象权限的用户授予，但最好由对象的所有者完成。授予对象权限的命令是 **GRANT**，回收权限的命令是 **REVOKE**，与数据库权限的操作相比，这两条命令有些不同的地方。授予对象权限的 **GRANT** 命令语法格式为：

```
GRANT 对象权限 1(列名),对象权限 2(列名)....  
ON 对象 TO 用户 1, 用户 2....  
WITH GRANT OPTION;
```

在授予对象权限时，不仅要说明是什么权限，还要指定是对那个对象的访问权限，这是与数据库权限的授予不同的地方。例如，假设当前用户是 **SYSDBA**，下面的语句将表 **t1** 的 **SELECT** 权限授予用户 **user1** 和 **user2**：

```
SQL> GRANT SELECT ON t1 TO user1,user2;
```

如果要将某个对象权限授予所有用户，可以用 **PUBLIC** 代替所有的用户名。用户访问其他用户的对象时，需要用对方的用户名限定对象。

如果在为用户指定某个对象权限时使用了“**WITH GRANT OPTION**”选项，那么这个用户在接受了这个权限后，可以将这个权限授予其他的用户。

对于 **SELECT**、**INSERT** 和 **UPDATE** 三个对象权限，还可以指定是在表中的哪个列上具有访问权限，也就是说，可以规定其他用户可以对表中的哪个列进行查询、插入和修改操作。例如，假设当前用户是 **SYSDBA**，下面的语句将表 **t1** 中对 **c1** 列的插入权限和 **c2** 列的查询权限授予用户 **user1** 和 **user2**：

```
SQL> GRANT INSERT(c1),SELECT(c2) ON t1 TO user1,user2;
```

这样用户 **user1** 和 **user2** 就可以对表 **t1** 的列 **c1** 列进行插入操作，对 **c2** 列进行查询操作，而对其他列没有任何权限。

EXECUTE 权限是指可以执行一个用户所拥有的存储函数、存储过程。假设当前用户创建了一个存储函数 **function1**，通过下面的语句可以把对这个函数的执行权限授予用户 **user1**。

```
SQL> GRANT EXECUTE ON function1 TO user1;
```

REFERENCES 权限是指其他用户的表可以与当前用户的表建立关联关系的权限。假设当前用户有一个表，在这个表的某个列上建立了主键约束。如果把这个表的 **REFERENCES** 权限授予其他用户，那么其他用户就可以通过外键将自己的表与这个表建立关联关系，也就是可以引用这个表中的主键列的值。假设当前用户是 **SYSDBA**，在表 **t1** 的 **c1** 列上建立了主键约束。假设用户 **SYSDBA** 要将表 **t1** 上的 **c1** 列的 **REFERENCES** 权限授予用户 **user1**，可以执行下面的 **GRANT** 语句。

```
SQL> GRANT REFERENCES(c1) ON t1 TO user1;
```

这样用户 **user1** 就可以创建一个表，在表上指定一个外键，并且与表 **t1** 上的 **c1** 列建立关联关系，或者修改一个已经存在的表，在表上指定一个外键，同样可以建立这种关联关系。

回收对象权限的命令是 **REVOKE**。一般回收对象权限的操作由权限的授予者完成。这条命令的语法格式为：

REVOKE 对象权限 1, 对象权限 2... ON 对象
FROM 用户 1, 用户 2...
[RESTRICT|CASCADE]

如果某个对象权限是以“WITH GRANT OPTION”方式授予一个用户的，那么这个用户可将这个权限再授予其他用户。在从这个用户回收对象权限时，需要指定为 CASCADE，进行级联回收，如果不指定，默认为 RESTRICT，则无法进行回收。

26.4 角色管理

角色是一组权限的组合，使用角色的目的是使权限管理更加方便。假设有 10 个用户，这些用户为了访问数据库，至少拥有 CREATE TABLE、CREATE VIEW 等权限。如果将这些权限分别授予这些用户，那么需要进行的授权次数是比较多的。但是如果把这些权限事先放在一起，然后作为一个整体授予这些用户，那么每个用户只需一次授权，授权的次数将大大减少，而且用户数越多，需要指定的权限越多，这种授权方式的优越性就越明显。这些事先组合在一起的一组权限就是角色，角色中的权限既可以是数据库权限，也可以是对象权限。

为了使用角色，首先在数据库中创建一个角色，这时角色中没有任何权限。然后向角色中添加权限。最后将这个角色授予用户，这个用户就具有了角色中的所有权限。在使用角色的过程中，可以随时向角色中添加权限，也可以随时从角色中删除权限，用户的权限也随之改变。如果要回收所有权限，只需将角色从用户回收即可。图 26.1 表示用户与角色之间的关系。

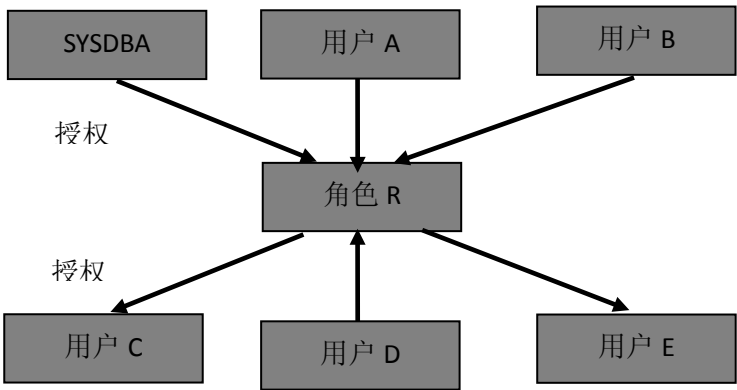


图 26.1 用户与角色的关系

26.4.1 角色的创建和删除

在数据库中有两类角色，一类是 DM 预设定的角色，一类是用户自定义的角色。DM 预定义的角色在数据库被创建之后即存在，并且已经包含了一些权限，数据库管理员可以将这些角色直接授予用户。表 26.3 列出了常见的系统角色及其包含的部分权限，具体可参见附录 4。

表 26.3 数据库预设定的角色

角色名称	所包含的权限
------	--------

DBA	ALTER DATABASE
	BACKUP DATABASE
	CREATE USER
	SELECT ANY TABLE
	CREATE ANY TABLE
RESOURCE	CREATE SCHEMA
	CREATE TABLE
	CREATE VIEW
	CREATE SEQUENCE
PUBLIC	SELECT TABLE
	UPDATE TABLE
	SELECT VIEW
DB_AUDIT_ADMIN	CREATE USER
	AUDIT DATABASE
DB_AUDIT_OPER	AUDIT DATABASE
DB_POLICY_ADMIN	CREATE USER
	LABEL DATABASE
DB_POLICY_OPER	LABEL_DATABASE

除了 DM 预设定的角色以外，用户还可以自己定义角色。一般情况下创建角色的操作只能由 SYSDBA 完成，如果普通用户要定义角色，必须具“CREATE ROLE”数据库权限。创建角色的命令是 CREATE ROLE，它的语法格式为：

```
CREATE ROLE 角色名;
```

例如，创建一个名为 role1 的角色，可以执行以下语句：

```
SQL> CREATE ROLE role1;
```

删除角色的命令是“DROP ROLE”。角色被删除时，角色中的权限都间接的被从用户回收。例如，删除刚才创建的角色语句为：

```
SQL> DROP ROLE role1;
```

26.4.2 角色中权限的添加和删除

角色刚被创建时，没有包含任何权限。用户可以将权限授予该角色，使这个角色成为一个权限的集合。向角色授权的方法与向用户授权的方法是相同的，只要将用户名用角色名代替就可以了。

需要注意的是，如果向角色授予数据库权限，可以使用 WITH ADMIN OPTION 选项，但是向角色授予对象权限时，WITH GRANT OPTION 将无意义。

例如，用户 SYSDBA 通过下列语句想角色 role1 授予若干权限：

```
SQL> GRANT SELECT ON t1 TO role1;
```

```
SQL> GRANT UPDATE(c1) ON t1 TO role1;
```

SYSDBA 用户可以通过下列语句向该角色授予若干数据库权限：

```
SQL> GRANT CREATE TABLE, CREATE VIEW TO role1;
```

如果要从角色中删除权限，可以执行 REVOKE 命令。权限回收的方法与从用户回收权限的方法相同，只是用角色名代替用户名就可以了。例如，用户将 SELECT 权限从角色 role1 回收，对应语句为：

```
SQL> REVOKE SELECT ON t1 FROM role1;
```

26.4.3 角色的分配和回收

只有将角色授予用户，用户才会具有角色中的权限。可以一次将角色授予多个用户，这样这些用户就都具有了这个角色中包含的权限。将角色授予用户的命令是 **GRANT**，授予角色的方法与授予权限的方法相同，只是将权限名用角色名代替就可以了。例如，下面的语句将角色 **role1** 授予用户 **user1** 和 **user2**：

```
SQL> GRANT role1 TO user1,user2 WITH ADMIN OPTION;
```

在一般情况下，将角色授予用户的操作是由 **DBA** 用户完成，普通用户如果要完成这样的操作，必须具有 **ADMIN ANY ROLE** 的数据库权限，或者具有相应的角色及其转授权。

要将角色从用户回收，需要执行 **REVOKE** 命令。角色被回收后，用户所具有的属于这个角色的权限都将被回收。从用户回收角色与回收权限的方法是相同的。这样的操作一般也有 **DBA** 用户完成。例如，下面的语句将角色 **role1** 从用户 **user1** 回收：

```
SQL> REVOKE role1 FROM user1;
```

26.4.4 角色的启用和禁用

某些时候，用户不愿意删除一个角色，但是却希望这个角色失效，此时，可以使用 **SP_SET_ROLE** 来设置这个角色为不可用。

```
SQL> SP_SET_ROLE('ROLE1', 0);
```

说明：

1. 只有 **SYSDBA** 用户才能启用和禁用角色，并且设置后立即生效；
2. 凡是包含禁用角色 **A** 的角色 **M**，**M** 中禁用的角色 **A** 将无效，但是 **M** 仍有效；
3. 系统表 **SYSOBJECTS** 的 **VALID** 字段表示角色是否有效；
4. 用户通过 **SP_SET_ROLE** 系统过程来设置角色是否有效；
5. **DBA**、**PUBLIC**、**RESOURCE** 角色是不能设置的。

当用户希望启用某个角色时，可以通过下面的语句来启用角色。

```
SQL> SP_SET_ROLE('ROLE1', 1);
```

26.5 标记管理

标记管理比权限管理严格，它的定义是：每一个数据对象被标以一定的密级，每一个用户也被授予某一个级别的许可证。对于任意一个对象，只有具有合法许可证的用户才可以存取。DM 利用策略和标记来实现数据库的强制访问机制，该功能仅安全版提供。

26.5.1 如何创建策略

一个策略由多个组件组成，其组件包括等级分类（**level**）和非等级类别（**category**）。每个标记必须包含至少一个等级分类组件，非等级类别组件则是可选的。

1. 等级（**level**）：

它是一线性有序的名称序列，用 $L = (l_1, l_2, \dots, l_p)$ 表示。其中 l_i ($1 \leq i \leq p$) 表示第 i 个名称，任意两个名称 l_i, l_j 之间，若 $i \leq j$ ，则 $l_i \leq l_j$ ，于是有 $l_1 \leq l_2 \leq \dots \leq l_p$ ，其中 l_1, l_2, \dots, l_p 称为等级分类（以下简称等级）。

在 DM 中，一个策略最大可定义 9999 个等级。用户在定义策略中的等级时，需要为其指定编号，其编号在 0—9999 之间（编号小的意味着级别较低）。

2. 范围

设集合 $C = \{c_1, c_2, \dots, c_m\}$ 中每一元素都是一名称， c_1, c_2, \dots, c_m 间彼此独立，无序，则集合 C 及其任意子集称为非等级类别集合，其中 c_1, c_2, \dots, c_m 称为非等级类别（以下简称为范围）。

在 DM 中，最大可定义 10000 个范围，需要用户设置编号，且编号在一个策略里面是唯一的，编号之间没有级别高低之分。

3. 组

设树 $G = \{g_1, g_2, \dots, g_m\}$ 中每一元素都是一名称， g_1, g_2, \dots, g_m 间有父子之分。只能有一个根组，每个组有且仅有一个父组（除根组外）。

在 DM 中，最大可定义 10000 个组，需要用户设置编号，且编号在一个策略里面是唯一的，编号之间没有级别高低之分。

设置策略的组成之前必须先创建一个空的策略，它的语法格式为：

```
MAC_CREATE_POLICY('策略名');
```

例如，创建一个策略名为 P1 的策略：

```
SQL> MAC_CREATE_POLICY('P1');
```

策略名在数据库中是唯一的，策略名可以进行修改，例如将策略 P1 名称修改为 P2：

```
SQL> MAC_ALTER_POLICY('P1', 'P2');
```

创建一个策略后，需要对策略添加组件，这样策略才可以被应用到表和用户上。例如，对策略 P1 添加一个等级为 100 的 L1，范围 C1,C2 以及组 G1,G2，G1 作为根组，G2 作为 G1 的孩子节点：

```
SQL> MAC_CREATE_LEVEL('P1', 100, 'L1');
```

```
SQL> MAC_CREATE_COMPARTMENT('P1', 10, 'C1');
```

```
SQL> MAC_CREATE_COMPARTMENT('P1', 20, 'C2');
```

```
SQL> MAC_CREATE_GROUP('P1', 10, 'G1', NULL);
```

```
SQL> MAC_CREATE_GROUP('P1', 20, 'G2', 'G1');
```

对于策略以及策略的组成成员，除了支持创建外，还支持相应的更改和删除操作。在对策略进行删除时，会同时清除策略的组成成员以及策略应用的相关对象。对策略的组成成员进行删除时，此成员必须是没有出现在现有标记中的，否则会报错。

例如，删除策略 P1，同时将应用的标记列进行删除：

```
SQL> MAC_DROP_POLICY('P1', 1);
```

26.5.2 如何创建标记

在 DM 中，标记用于标识数据库中主体及客体的敏感度（sensitivity）。一个标记由多个组件组成，其组件包括等级分类（level）和非等级类别（category）。每个标记必须包含至少一个等级分类组件，非等级类别组件则是可选的。

1. 标记的格式

在 DM 中，标记用字符串表示，其最大长度为 4000。在标记格式中，等级和范围必须严格按照顺序进行排列（中间不能包含空格），否则为不合法的标记。假定系统中存在策略 P1，其包含等级 L1,L2,L3,范围 C1,C2,C3,C4,组 G1,G2,G3。则下列标记为合法标记：


```
L1::  
L1:C1:  
L1:C1,C2:  
L1::G1  
L1:C1:G1,G2
```

而下列标记为不合法标记：

```
:  
:C1,C2  
C1,C2  
:C1:G1  
::G1,G2  
G1,G2
```

2. 标记的创建

标记存储在 **SYSMACLABELS** 系统表上，安全管理员可以对其进行查看。每个标记都有唯一的 **TAG** 进行区分，用户需指定 **TAG** 值，**TAG** 在数据库中是唯一的，**TAG** 的范围为 0-4294967294。

标记的创建分为两种，显式创建和隐式创建。显式创建是由 **SYSSSO** 通过执行系统函数进行创建，需要指定策略名，标记 **ID** 和标记的具体值，如果系统中已存在对应的标记 **ID** 或标记的具体值，则会报错。例如，要创建一个 **P1** 的 **TAG** 为 0 的标记 (**L1:C1:G1**)：

```
SQL> MAC_CREATE_LABEL('P1', 0, 'L1:C1:G1');
```

将在 **SYSMACLABELS** 表上记录 **TAG** 值以及标记字符串'**L1:C1:G1**'的内部格式，这个格式是对字符串'**L1:C1:G1**'进行处理后的格式。

如果要查看系统中某个标记对应的具体的值，例如，要查看刚刚创建的 **ID** 为 0 的标记的具体值：

```
SQL> SF_MAC_LABEL_TO_CHAR(0);
```

这个系统函数会返回标记值为 0 对应的标记值，即标记 (**L1:C1:G1**)。

隐式标记的创建不需要用户具有创建标记的权限，一般用于普通用户在对具有标记的表进行插入，更新等操作时使用。例如，用户想获得 **L1:C1:G1** 对应的标记 **ID** 用于插入到标记列中：

```
SQL> SF_MAC_LABEL_FROM_CHAR('P1', 'L1:C1:G1');
```

这个系统函数首先会去系统表 **SYSMACLABELS** 查找是否具有标记 (**L1:C1:G1**) 的记录，如果存在，则返回相应的标记 **ID**，如果不存在，系统会自动创建一个对应的标记值，并返回给用户相应的标记 **ID**，标记 **ID** 是系统自动生成的。

26.5.3 如何对表应用策略

一个表可以应用多个策略，但一个策略对表只能应用一次。对表应用策略时，会在表上新建一个标记列，用于记录标记，新建的标记列名必须和表中已有列名不同，同时还可以指定初始的标记值，以及是否将标记列设置为隐藏列。例如，对模式 **SYSDBA** 下的表 **T1** 应用策略 **P1**，定义标记列名为 **LABEL_COL**，初始标记值为'**L1:C1:G1**'，同时设定标记列为隐藏列：

```
SQL> MAC_APPLY_TABLE_POLICY('P1', 'SYSDBA', 'T1', 'LABEL_COL', 'L1:C1:G1', 1);
```

隐藏列是指对表进行操作时，如果不显示指定隐藏列名，将无法显示隐藏列。如果要设置为非隐藏列，只需将最后一个参数设置为 0 即可。

执行完上面的 **SQL** 语句后，会在 **T1** 表上新建一列，列名为 **LABEL_COL**，其初始值为标记字符串'**L1:C1:G1**'对应的 **TAG** 值。

取消表上对应的策略时，可以选择是否对标记列进行删除。例如，取消模式 **SYSDBA**

下的表 T1 上的策略 P1，保留标记列 LABEL_COL:

```
SQL> MAC_REMOVE_TABLE_POLICY('P1', 'SYSDBA', 'T1', 0);
```

如果要删除标记列，只需将最后一个参数设置为 1 即可。

26.5.4 如何对用户应用策略

对用户应用一个策略，要指定一个最高等级、最低等级、默认等级和行等级，其中默认等级介于最高等级和最低等级之间，最高等级限制了用户最高的读写权限，最低等级限制了用户的写权限，默认等级表示用户登录时会话标记的等级，行等级表示插入一行数据的标记使用的等级；指定范围和组，对范围需要指定读的范围、写的范围、默认范围和行级范围，其中写范围必须是读范围的子集，默认范围是读范围的子集，行级范围是写范围和默认范围的交集的子集。组的规则和范围相同。

在对用户应用策略时，必须先定义用户的安全级别。例如，对用户 USER1 应用策略 P1，设置最高等级为 L4，最低等级为 L1，默认等级为 L3，行等级为 L2:

```
SQL> MAC_USER_SET_LEVELS('P1', 'USER1', 'L4', 'L1', 'L3', 'L2');
```

对用户设置了等级后，可以对用户设置相应的范围和组。例如，对用户 USER1 应用策略 P1，设置读范围为 C1、C2，写范围为 C1，默认范围为 C1、C2，行级范围为 C1:

```
SQL> MAC_USER_SET_COMPARTMENTS('P1', 'USER1', 'C1,C2', 'C1', 'C1,C2', 'C1');
```

同时可以对用户设置组。例如，对用户 USER1 应用策略 P1，设置读组为 G1、G2，写组为 G1，默认组为 G1、G2，行级组为 G1:

```
SQL> MAC_USER_SET_GROUPS('P1', 'USER1', 'G1,G2', 'G1', 'G1,G2', 'G1');
```

在给用户应用策略时，同时可以授予策略特权。策略特权分为访问特权和行标记特权。访问特权分为两种:

1. **READ**: 读数据时不受策略影响，但写数据访问控制仍然受到强制访问控制;
2. **FULL**: 可以读写任何数据，不受策略影响。

一旦一个行的标记设定后，就需要行标记特权才能改变其标记。行标记特权有如下三种:

1. **WRITE UP**: 用户可以利用该特权提升一个行的等级，同时不改变范围和组。这个等级可以提高到用户的最高等级，而该行的原始等级可能比用户的最低等级还低;
2. **WRITE DOWN**: 用户可以利用该特权降低一个行的等级，同时不改变范围和组。这个等级可以降低到用户的最低等级，而该行的原始等级可能比用户的最低等级还低;
3. **WRITE ACROSS**: 用户可以利用该特权修改一个行的范围和组，同时不改变等级。新的范围和组只要满足在策略中是合理的就可以了，不必限于用户拥有访问权的范围和组。

例如，对用户 USER1 设置策略 P1 上的 FULL 特权和 WRITEUP 特权:

```
SQL> MAC_SET_USER_PRIVS('P1', 'USER1', 'FULL,WRITEUP');
```

为了方便用户在会话执行的过程中修改自身的标记，DM 提供了针对会话的 LABEL 设置，这些设置会挂载在会话上，一旦会话结束，这些信息就完全被抛弃。

用户可以设置会话的默认标记，但其值必须在合法的范围内，等级必须在 MIN 和 MAX 之间，范围和组必须是 READ 的子集。用户还可以设置会话的行级标记，同样必须在合法的范围内。

例如，用户 USER1 应用了策略 P1 上如下标记:

```
SQL> MAC_USER_SET_LEVELS('P1', 'USER1', 'L4', 'L1', 'L3', 'L2');
```

```
SQL> MAC_USER_SET_COMPARTMENTS('P1', 'USER1', 'C1,C2', 'C1', 'C1,C2', 'C1');
```

```
SQL> MAC_USER_SET_GROUPS('P1', 'USER1', 'G1,G2', 'G1', 'G1,G2', 'G1');
```

此时用户的默认标记是 L3:C1,C2:G1,G2,此时用户 USER1 想提高自身的默认等级，从 L3 提升到 L4，可以使用如下命令:

```
SQL> MAC_SET_SESSION_LABEL('P1', 'L4:C1,C2:G1,G2');
```

提升级别后，如果用户想恢复原始的默认标记，可以使用如下命令：

```
SQL> MAC_RESTORE_LABEL('P1');
```

如果用户不想回复原始的默认标记，而希望用将会话的默认标记作为自己全局的默认标记，可以使用如下命令：

```
SQL> MAC_SAVE_DEFAULT_LABEL('P1');
```

注意：在使用 `MAC_RESTORE_LABEL` 和 `MAC_SAVE_DEFAULT_LABEL` 时，会同时影响会话上的默认标记和行级标记。

如果要删除对用户 `USER1` 应用的策略 `P1`，可以使用如下命令。

```
SQL> MAC_USER_REMOVE_POLICY('P1', 'USER1');
```

26.5.5 读写控制规则

DM 的自主访问控制(DAC)策略优先于强制访问控制(MAC)策略。为了能访问一条元组，用户不仅首先要满足 **DAC** 条件，还必须满足 **MAC** 的条件。

用户访问表时，必须保证应用于表上的所有策略均适用于该用户，否则访问被拒绝。若表上未应用任何策略，则用户只需满足 **DAC** 条件即可。

读访问规则：

1. 用户的等级必须大于等于数据的等级；
2. 用户的标记必须包含至少一个数据的组（或者是其某一个的父亲组）；
3. 用户的标记必须包含数据的所有范围。

如果用户满足以上三个规则，即可以对数据进行读取。比较的先后顺序是等级，组，范围，具体访问规则如图 26.2 所示。

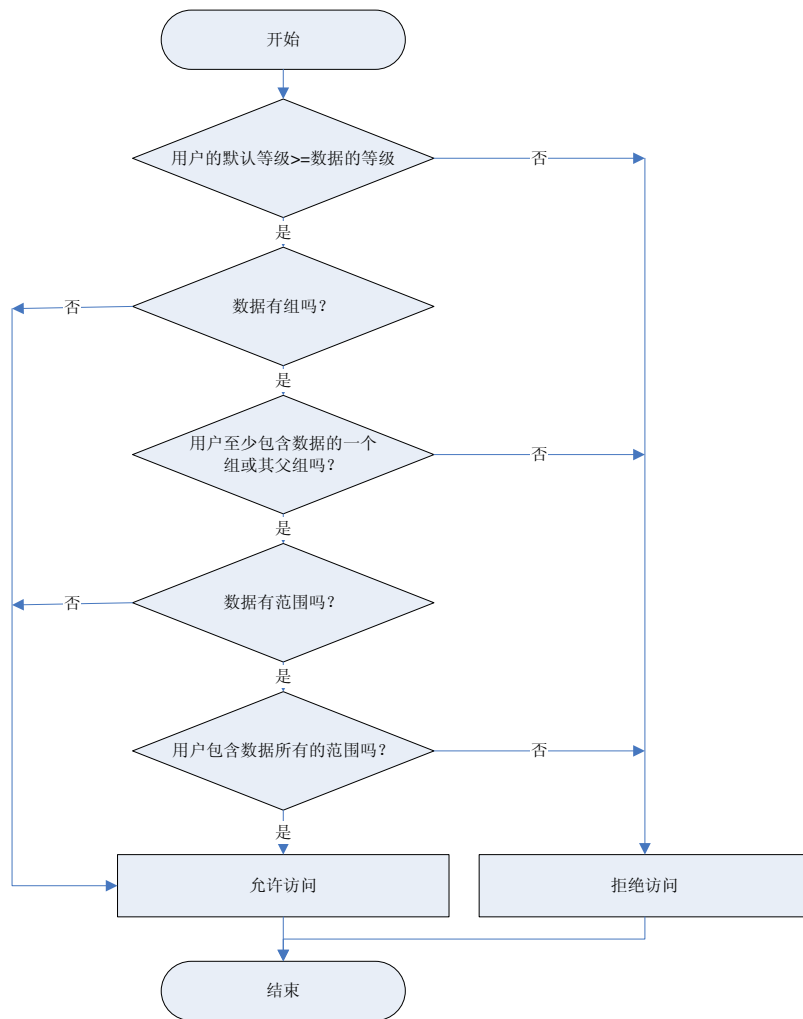


图 26.2 MAC 读访问规则

写访问规则:

(1) 数据标记的等级必须大于等于用户标记的最小等级，小于或等于用户的会话标记的等级；

(2) 用户的标记必须包含至少一个数据的组（或者是其某一个的父亲组）的写权限；

(3) 用户的标记必须包含数据的所有的范围上的写权限。

具体访问规则如 26.3 图所示。

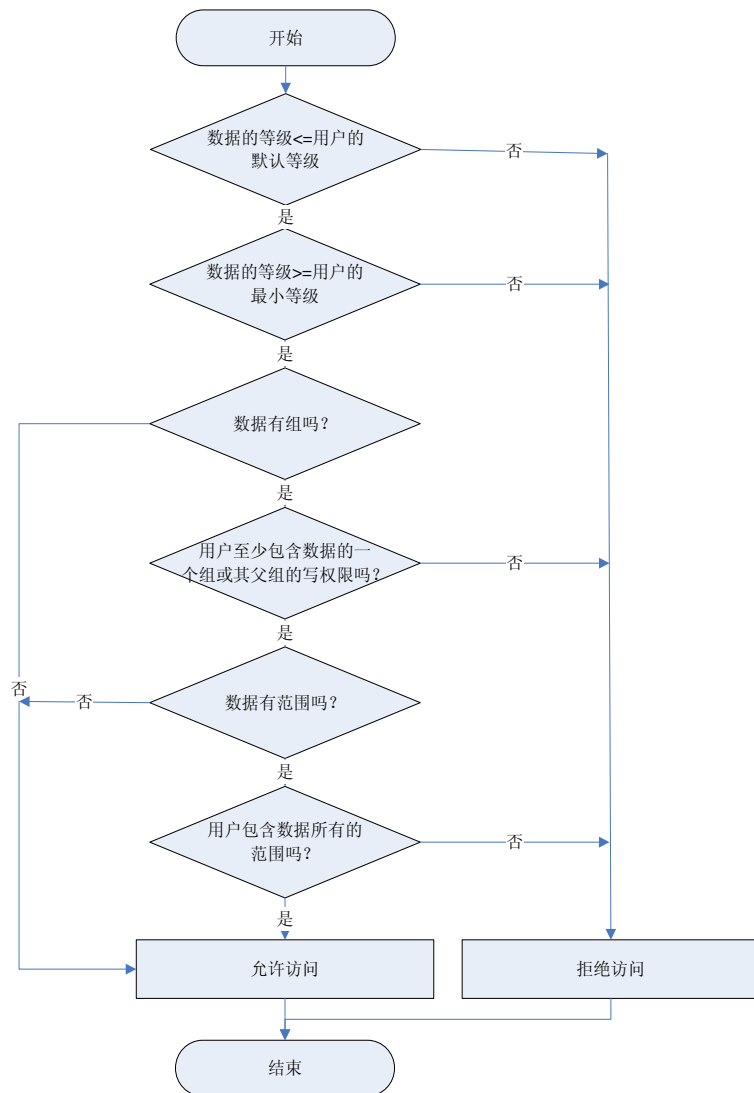


图 26.3 MAC 写访问规则

在应用于表的所有策略上，用户插入一条元组时，该元组的标记即为用户的行标记。

用户更新和删除一条元组时，必须保证用户的标记与元组的标记相等或元组的标记为 NULL，否则操作失败。

用户查询一条元组时，必须保证用户的标记能够支配元组的标记或元组的标记为 NULL。

对于视图及存储过程，当前用户在调用它们访问数据时，其策略标记与视图及存储过程访问的任何基表上元组的策略标记的比较都必须符合上述规则。

26.5.6 扩展客体应用标记

扩展客体标记支持对数据库所有的客体进行标记，这些客体包含模式、表、视图、索引、存储过程、函数、包、序列、触发器、同义词以及列等。一旦一个对象被应用了扩展客体标记，则用户只有在支配相应标记的情况下，才能访问客体。仅安全版提供此功能。

对于任何一个对象的操作，用户必须具有本层对象以及上层对象的标记，才能操作此对象，如查询一个表的一列，则用户的读标记必须支配查询列、表、以及模式的标记，才能查询此列。

对对象的不同的 DML 操作，会匹配不同的用户标记，如对对象的 SELECT、EXECUTE

以及 REFERENCE，会检查用户的读标记。对对象的 UPDATE、INSERT、DELETE，会检查用户的写标记。

对对象的 DDL 操作，也会根据用户的写标记进行相应的扩展标记检查，检查范围为改对象的标记，以及该对象上层的标记。如删除一个表，则会检查用户的写标记是否支配表、模式的标记。

创建的新对象不会含有默认的标记，需要安全员进行手工设置。

1. 应用标记

对扩展客体应用标记的系统函数如下所示。

```
MAC_APPLY_OBJ_POLICY(  
POLICY_NAME          VARCHAR(128),  
OBJ_TYPE              VARCHAR(128),  
SCH_NAME              VARCHAR(128),  
OBJ_NAME              VARCHAR(128),  
COL_NAME              VARCHAR(128),  
LABEL                 VARCHAR(4000)  
);
```

参数定义如下：

1. POLICY_NAME: 策略名称;
2. OBJ_TYPE: 必须是下面几个选项之一，“SCHEMA”、“TABLE”、“VIEW”、“INDEX”、“PROCEDURE”、“FUNCTION”、“PACKAGE”、“SEQUENCE”、“TRIGGER”、“COLUMN”、“SYNONYM”;
3. SCH_NAME: 模式名;
4. OBJ_NAME: 应用标记的客体名称，如果对模式应用策略，则此值和 SCH_NAME 值一致;
5. COL_NAME: 应用标记的列名，只有在 OBJ_TYPE='COLUMN'时，此列才有效;
6. LABEL:具体的标记值。

例如，对模式 SYSDBA 下的表 T1 应用策略 P1 的标记（L1:C1:G1）：

```
SQL> MAC_APPLY_OBJ_POLICY('P1','TABLE','SYSDBA','T1',NULL,'L1:C1:G1');
```

2. 更改标记

对扩展客体应用策略后，如果想要更改表上的标记，可以用过如下函数。

```
MAC_ALTER_OBJ_POLICY(  
POLICY_NAME          VARCHAR(128),  
OBJ_TYPE              VARCHAR(128),  
SCH_NAME              VARCHAR(128),  
OBJ_NAME              VARCHAR(128),  
COL_NAME              VARCHAR(128),  
LABEL                 VARCHAR(4000)  
);
```

例如，修改 SYSDBA 下表 T1 的标记，改为标记（L1:C1,C2:G1）：

```
SQL> MAC_ALTER_OBJ_POLICY('P1','TABLE','SYSDBA','T1',NULL,'L1:C1,C2:G1');
```

3. 删除标记

如果用户想要删除扩展客体上的对应标记，可以使用 MAC_DROP_OBJ_POLICY 函数。例如，删除 SYSDBA 下表 T1 的扩展标记：

```
SQL>MAC_DROP_OBJ_POLICY('P1','TABLE','SYSDBA','T1',NULL);
```

4. 标记匹配规则

读写标记的判断采用自上而下依次检查的原则，即如果查询表，需要检查表所属模式的标记是否匹配，然后检查表上的标记是否匹配，最后检查列级的标记是否匹配。层次的最上层是模式，然后是表、视图级别，最后是列、索引级别。

26.6 数据库审计

审计机制是 DM 数据库管理系统安全管理的重要组成部分之一。DM 数据库除了提供数据安全保护措施外，还提供对日常事件的事后审计监督。DM 具有一个灵活的审计子系统，可以通过它来记录系统级事件、个别用户的行为以及对数据库对象的访问。通过考察、跟踪审计信息，数据库审计员可以查看用户访问的形式以及曾试图对该系统进行的操作，从而采取积极、有效的应对措施。

在 DM 系统中，专门为审计设置了开关，要使用审计功能首先要打开审计开关，开关打开后，DM 对审计设置语句指定的审计对象进行审计，否则不记录审计记录。此开关的设置方法是：通过在控制台 Console 中，配置服务器 ENABLE_AUDIT 参数值启用该功能。这种方法需要重新启动 DM 服务器方可生效。设置为 1 打开普通审计，设置为 2 打开普通审计和实时审计，设置为 0 则关闭审计开关，此配置项默认值为 0。

26.6.1 DM 数据库审计员的创建和删除

1. 创建数据库审计员

DM 中设立了 AUDITOR（审计员）角色，只有具有 AUDITOR 角色权限的用户才能进行审计，可以通过对用户授予 AUDITOR 角色来给予某个用户审计的权限。只有数据库审计管理员才能创建新的审计员。

2. 回收数据库审计员的权限

一个数据库审计员能够回收另一个数据库审计员的审计权限，但由系统最初定义的数据库审计员 SYSAUDITOR 的权限是不能被回收的。

26.6.2 DM 审计设置与取消

数据库审计员指定被审计对象的活动称为审计设置。DM 提供审计设置系统函数来实现这种设置，被审计的对象可以是某类操作，也可以是某些用户在数据库中的全部行踪。只有预先设置的操作和用户才能被 DM 系统自动进行审计。当数据库审计员认为某些操作或某些用户不必再行审计，可用 DM 提供的取消审计设置系统函数将原来对某些操作或用户的审计设置清除掉。设置与取消审计是立即生效的，不需重新启动服务器。

1. 审计范围

DM 允许在三个级别上进行审计设置，如表 26.4 所示。

表 26.4 审计设置级别

审计级别	说明
系统级	系统的启动与关闭，此级别的审计记录在任何情况下都会强制产生，无法

	也无需由用户进行设置。
语句级	导致影响特定类型数据库对象的特殊 SQL 或语句组的审计。如 AUDIT TABLE 将审计 CREATE TABLE 、 ALTER TABLE 和 DROP TABLE 等语句
对象级	审计作用在特殊对象上的语句。如 test 表上的 INSERT 语句。

审计设置存放于系统表中，进行一次审计设置在系统表 **SYSAUDIT** 中增加设置的记录，取消审计则删除系统表 **SYSAUDIT** 中相应的记录。

所有的审计记录都存放在审计日志文件中，审计日志文件命名方式为 **AUDIT_自动生成的全局唯一的串_日期时间.log**，服务器每次重启生成一个日志文件。

DM 的审计分为两大类，语句级审计和对象级审计。语句级审计的动作是全局的，不对应具体的数据库对象。对象级审计发生在具体的对象上，需要指定模式名以及对象名。

审计选项见下表：

<语句级审计选项>包括的类型见表 26.5。

表 26.5 语句级审计选项

审计选项	审计的数据库操作	说明
ALL	所有的语句级审计选项	所有可审计操作
USER	CREATE USER ALTER USER DROP USER	创建 / 修改 / 删除用户操作
ROLE	CREATE ROLE DROP USER	创建 / 删除角色操作
TABLESPACE	CREATE TABLESPACE ALTER TABLESPACE DROP TABLESPACE	创建 / 修改 / 删除表空间操作
SCHEMA	CREATE SCHEMA DROP SCHEMA	创建 / 删除 / 设置当前模式操作
TABLE	CREATE TABLE ALTER TABLE DROP TABLE TRUNCATE TABLE	创建 / 修改 / 删除 / 清空基表操作
VIEW	CREATE VIEW DROP VIEW	创建 / 修改 / 删除视图操作
INDEX	CREATE INDEX DROP INDEX	创建 / 删除索引操作
PROCEDURE	CREATE PROCEDURE ALTER PROCEDURE DROP PROCEDURE	创建 / 修改 / 删除存储模块操作
TRIGGER	CREATE TRIGGER ALTER TRIGGER DROP TRIGGER	创建 / 修改 / 删除触发器操作
SEQUENCE	CREATE SEQUENCE DROP SEQUENCE	创建 / 删除序列操作
CONTEXT	CREATE CONTEXT INDEX ALTER CONTEXT INDEX DROP CONTEXT INDEX	创建 / 修改 / 删除全文索引操作
SYNONYM	CREATE SYNONYM DROP SYNONYM	创建 / 删除同义词
GRANT	GRANT	授予权限操作
REVOKE	REVOKE	回收权限操作
AUDIT	AUDIT	设置审计操作
NOAUDIT	NOAUDIT	取消审计操作
INSERT TABLE	INSERT INTO TABLE	表上的插入操作
UPDATE TABLE	UPDATE TABLE	表上的修改操作
DELETE TABLE	DELETE FROM TABLE	表上的删除操作

SELECT TABLE	SELECT FROM TABLE	表上的查询操作
EXECUTE PROCEDURE	CALL PROCEDURE	调用存储过程或函数操作
PACKAGE	CREATE PACKAGE DROP PACKAGE	创建 / 删除包规范
PACKAGE BODY	CREATE PACKAGE BODY DROP PACKAGE BODY	创建 / 删除包体
MAC POLICY	CREATE POLICY DROP POLICY ALTER POLICY	创建 / 修改 / 删除策略
MAC LEVEL	CREATE LEVEL ALTER LEVEL DROP LEVEL	创建 / 修改 / 删除等级
MAC COMPARTMENT	CREATE COMPARTMENT DROP COMPARTMENT ALTER COMPARTMENT	创建 / 修改 / 删除范围
MAC GROUP	CREATE GROUP DROP GROUP ALTER GROUP ALTER GROUP PARENT	创建 / 修改 / 删除组，更新父组
MAC LABEL	CREATE LABEL DROP LABEL ALTER LABEL	创建 / 修改 / 删除标记
MAC USER	USER SET LEVELS USER SET COMPARTMENTS USER SET GROUPS USER SET PRIVS	设置用户等级 / 范围 / 组 / 特权
MAC TABLE	INSER TABLE POLICY REMOVE TABLE POLICY APPLY TABLE POLICY	插入 / 取消 / 应用表标记
MAC SESSION	SESSION LABEL SESSION ROW LABEL RESTORE DEFAULT LABELS SAVE DEFAULT LABELS	保存 / 取消会话标记 设置会话默认标记 设置会话行标记
CHECKPOINT	CHECKPIONT	检查点（checkpoint）
SAVEPOINT	SAVEPOINT	保存点
EXPLAIN	EXPLAIN	显示执行计划
NOT EXIST		分析对象不存在导致的错误

DATABASE	ALTER DATABASE	修改当前数据库操作
CONNECT	LOGIN LOGOUT	登录 / 退出操作
COMMIT	COMMIT	提交操作
ROLLBACK	ROLLBACK	回滚操作
SET TRANSACTION	SET TRX ISOLATION SET TRX READ WRITE	设置事务的读写属性和隔离级别

<对象级审计选项>包括的类型见表 26.6。

表 26.6 对象级审计选项

审计选项(SYSAUDITRECORDS 表中 operation 字段对应的内容)	TABLE	VIEW	PROCEDURE FUNCTION
INSERT(insert)	√	√	
UPDATE(update)	√	√	
DELETE(delete)	√	√	
SELECT(select)	√	√	
EXECUTE(execute)			√
MERGE INTO	√	√	
EXECUTE TRIGGER			
LOCK TABLE	√		
ALL(所有对象级审计选项)	√	√	√

例 1 对用户 user2 进行的表修改进行审计，不管失败和成功。

```
SP_AUDIT_STMT('UPDATE TABLE', 'user2', 'ALL');
```

例 2 取消对用户 user2 进行的表修改操作的审计。

```
SP_NOAUDIT_STMT('UPDATE TABLE', 'user2', 'ALL');
```

例 3 设置对用户 user1 进行的表 SYSDBA.tab1 插入操作成功的审计。

```
SP_AUDIT_OBJECT('INSERT','user1','SYSDBA','tab1','SUCCESSFUL');
```

例 4 取消对用户 user1 进行的表 SYSDBA.tab1 删除操作失败的审计。

```
SP_AUDIT_OBJECT('DELETE','user1','SYSDBA','tab1','FAIL');
```

2. 审计的设置策略

尽管相对来说审计的开销并不是很大，但是要尽可能的限制审计对象和审计事件的数目。最大限度的降低审计带来的性能方面的影响，并且减小审计记录的规模。

当使用审计的时候，通常有如下参考标准：

1) 评价审计的目的

在对审计的目的和原因有了清晰的认识后，可以设计合适的审计策略来避免不必要的审计。

2) 理智地进行审计

审计必要的最少数目的语句、用户或者对象来获得目标信息。这样，避免在混乱的审计信息中浪费精力。

3) 审计操作的甄选

在开始审计的时候，也许不清楚应该审计那些可疑操作，这时可以适当扩大审计操作的范围，一旦记录并分析了初步的审计信息后，就应该取消一些普通的操作，而把审计的重点保留在一些可疑操作上。

4) 存档审计并清除审计跟踪

在收集到必要的审计信息后，应该通过存档审计文件来保留这些审计记录，并清除这些信息的审计跟踪。

26.6.3 DM 审计文件管理

审计信息存储在审计文件中。审计文件命名格式为 AUDIG_GUID_创建时间.log。当单个审计文件超过指定大小时，系统会自动切换审计文件，自动创建新的审计文件，审计记录将写入新的审计文件中。审计文件的大小可以通过 INI 参数 AUDIT_MAX_FILE_SIZE 指定。

随着系统的运行，审计记录将会不断增加，审计文件需要更多的磁盘空间。在极限情况下，审计记录可能会因为磁盘空间不足而无法写入审计文件，最终导致系统无法正常运行。DM 采用了两种策略处理此种情况，一种为配置 INI 参数 `AUDIT_FILE_FULL_MODE=1` 时，删除最老的审计文件，直至有足够的空间创建新审计文件。另一种配置 INI 参数 `AUDIT_FILE_FULL_MODE=2`，将不再写审计记录。此参数可通过 `SP_SET PARA_VALUE` 动态修改。DM 默认采用第一种策略。这两种策略都会导致审计记录的缺失，因此，数据管理员应该及时对审计文件进行备份。

26.6.4 DM 审计信息查阅

当使用 DM 提供的审计机制进行了审计设置后，这些审计设置信息都记录在数据字典中。只要 DM 系统处于审计活动状态，系统则按审计设置进行审计活动，并将审计信息写入审计文件。审计记录内容包括操作者的用户名、所在站点、所进行的操作、操作的对象、操作时间、当前审计条件等。具有审计权限的用户可以通过系统表 `SYSAUDIT` 查询审计设置信息，通过动态视图 `V$AUDITRECORDS` 可以查询系统默认路径下的审计文件的审计记录。

26.6.5 DM 审计实时侵害检测

26.6.5.1 审计实时侵害检测

当 INI 参数 `ENABLE_AUDIT=2` 时，开启审计实时侵害检测功能。

实时侵害检测系统用于实时分析当前用户的操作，并查找与该操作相匹配的实时审计分析规则，如果规则存在，则判断该用户的行为是否是侵害行为，确定侵害等级，并根据侵害等级采取相应的响应措施。

实时侵害检测系统可能需要进行一些操作频率上的统计，因此需要保存用户的部分历史操作记录，即保存审计分析规则操作频率门限值选项子句规定的时间间隔范围之内的操作记录，超出该时间间隔范围之外的操作记录会被删除。

为了界定不同侵害行为对系统的危害情况，实时侵害检测系统定义了四种级别的侵害等级，从四级到一级侵害行为严重程度递增。这样根据用户操作匹配的实时侵害检测规则来判断其属于哪一级的侵害行为。

四种侵害等级如下：

1. 四级：操作只违反了 IP 或者时间段设置之一，且对应累计次数在规定时间间隔内没有达到门限值，或者没有门限值设置；
2. 三级：操作同时违反了 IP 和时间段设置，且累计次数在规定时间间隔内没有达到门限值，或者没有门限值设置；
3. 二级：操作的 IP 和时间段都正常，且累计次数在规定时间间隔内达到了门限值；
4. 一级：操作只违反了 IP 或时间段设置之一，且对应累计次数在规定时间间隔内达到了门限值，或者操作同时违反了 IP 和时间段设置，且累计次数在规定时间间隔内达到了门限值。

实时侵害检测规则的设置，使得某些操作可能会触发多条审计分析规则，也就是说可能存在多条审计分析规则同时匹配的情况。在这种情况下，需要把操作记录保存到所有匹配的、

同时规定了操作频率门限值的审计分析规则下,并且使用这些审计分析规则对该操作进行分析,从所有满足的侵害等级中选择一个最高的侵害等级进行响应。

匹配实时审计分析规则按照最大匹配原则进行,即尽可能多的匹配可匹配的分析规则。判断的依据包括:登录/用户名、审计选项。对于规则中的 **WHENEVER** 子句,在匹配规则时可以忽略,都视为匹配,或称为不完全匹配。

根据实时侵害检测结果做出安全审计响应,主要分为:

1. 实时报警生成;
2. 违例进程终止;
3. 服务取消;
4. 账号断开与失效。

安全响应的分类与侵害等级的分级相对应,由低到高分四个等级。

1. 四级响应:实时报警生成,对四级侵害行为进行响应。当系统检测到四级侵害行为时,生成报警信息,记录到系统表 **SYSAUDITALARMINFO** 中;
2. 三级响应:违例进程终止,对三级侵害行为进行响应。当系统检测到三级侵害行为时,终止当前操作(用户当前连接仍然保持)。同时生成报警信息;
3. 二级响应:服务取消,对二级侵害行为进行响应。当系统检测到二级侵害行为时,强制断开用户当前连接,退出登录。同时生成报警信息;
4. 一级响应:账号锁定或失效,对一级侵害行为进行响应。当系统检测到一级侵害行为时,强制断开用户当前连接,退出登录,并且锁定账号或使帐号失效。同时生成报警信息。

达梦提供了 **dmamon** 工具用于监视审计实时侵害检测,并将报警信息以邮件的方式发送到邮箱中。**dmamon** 需要配置 **dmamon.ini**, **dmamon.ini** 只能通过工具 **dmamon_ctl** 进行创建或修改。

26.6.5.2 dmamon

dmamon 工具的参数及说明

1. userid

用于指定登录数据库的信息,包含用户名、密码、服务器地址、端口号。参数值格式为:用户名/密码[@主机地址[:端口号]] [#SSL_PATH@SSL_PWD]。

2. path

用于指定 **dmamon.ini** 文件的路径,字符串类型。**dmamon.ini** 相关信息参见 2.1.4.1。**dmamon.ini** 只能通过工具 **dmamon_ctl** 进行创建或修改。**dmamon_ctl** 参见下一节。

26.6.5.3 dmamon_ctl

dmamon_ctl 工具的参数及说明

1. file

用于指定 **dmamon.ini** 文件的路径。

启动 **dmamon_ctl** 后,以下命令可以查看或修改 **dmamon.ini** 文件

表 26.7 dmamon_ctl 命令

命令	说明
EDIT	修改 INI 文件信息
ADD	增加接收者的邮件地址信息
REMOVE	删除接收者的邮件地址信息
SHOW	显示当前控制文件的内容
SAVE	保存，保存路径必须为 dmamon.ini 的绝对路径。退出程度前须保存，否则修改无效
EXIT	退出当前程序
HELP	显示帮助信息

26.6.6 DM 审计分析

DM 提供了专门的审计分析工具 **Analyzer**，实现对审计记录的分析功能，能够根据所制定的分析规则，对审计记录进行分析，判断系统中是否存在对系统安全构成危险的活动。

只有审计用户才能使用审计分析工具 **Analyzer**。审计用户登陆审计分析工具后，通过 **Analyzer** 可以创建和删除审计规则，并可以指定对某些审计文件应用某些规则。并将审计结果以表格的方式展现出来。

审计分析规则由一系列的条件组合而成，这些条件包括：用户名、模式名、审计对象名、审计操作名、操作时间段、IP 段、时间间隔和门限次数以及这些条件的组合。对审计文件进行分析时，可以同时应用多个规则，分析结果为应用单个分析规则的交集。

Analyzer 具体用法请参见 **Analyzer** 联机帮助手册。

26.7 通信加密

DM 提供两种通信加密方式，即不加密、SSL 加密。选择是否使用通信加密以服务器端设置为准，即通过设置服务器端配置文件 dm.ini 中 EENABLE_ENCRYPT 项来指定，客户端以服务器采用的通信方式与其进行通信。EENABLE_ENCRYPT 取值 0、1 分别代表不加密、加密。EENABLE_ENCRYPT 的默认值为 0。当选择加密的情况下，需要在服务器所在目录下的 server_ssl 子目录中存放 CA 的证书、服务器的证书和服务器的密钥，同时在客户端所在目录下的 client_ssl 子目录中存放 CA 的证书、客户端的证书和客户端的密钥，这样服务器和客户端的通信即是建立在加密的 SSL 连接之上的。

26.8 存储加密

为了防止用户直接通过数据文件获取用户信息，DM 提供了透明数据加密的功能。密钥生成、密钥管理和加解密过程由数据库管理系统自动完成，用户不可见。透明加密的目的主要是保证存储在数据文件中的敏感数据的安全，并不能保护合法用户的个人私密数据。系统内置了常用的 DES，AES，RC4 等算法，以此来保护数据的安全性。

DM 支持对表上列级加密，支持建表时设置加密列，以及修改表定义时，设置加密列。存储加密支持所有的列类型，包括大字段类型。建表时指定加密列的语法格式为：

```
CREATE TABLE 表名(列名 1 列类型 ENCRYPT WITH 加密算法 AUTO BY 密钥 HASH WITH 散
```

列算法 加盐，列名 2.....);

例如，创建表 t1，对 c1 列进行加密：

```
SQL> CREATE TABLE t1(c1 INT ENCRYPT WITH DES_ECB AUTO BY DAMENG HASH WITH MD5  
SALT, c2 VARCHAR);
```

上面的语句创建了表 t1，其中 c1 列使用加密算法 DES_ECB 进行加密，密钥为 "DAMENG"，散列算法为 MD5。表 26.8 列出了系统中支持的加密算法和散列算法。

表 26.8 支持的加密算法

算法名称	算法类型	分组长度	密钥长度
DES_ECB	分组加密算法	8	8
DES_CBC	分组加密算法	8	8
DES_CFB	分组加密算法	8	8
DES_OFB	分组加密算法	8	8
DESEDE_ECB	分组加密算法	8	16
DESEDE_CBC	分组加密算法	8	16
DESEDE_CFB	分组加密算法	8	16
DESEDE_OFB	分组加密算法	8	16
AES128_ECB	分组加密算法	16	16
AES128_CBC	分组加密算法	16	16
AES128_CFB	分组加密算法	16	16
AES128_OFB	分组加密算法	16	16
AES192_ECB	分组加密算法	16	24
AES192_CBC	分组加密算法	16	24
AES192_CFB	分组加密算法	16	24
AES192_OFB	分组加密算法	16	24
AES256_ECB	分组加密算法	16	32
AES256_CBC	分组加密算法	16	32
AES256_CFB	分组加密算法	16	32
AES256_OFB	分组加密算法	16	32
RC4	流加密算法	-	16
MD5	散列算法	-	-
SHA1	散列算法	-	-

存储加密在保证数据文件的安全性的同时，也会带来一定的性能影响，不同的加密算法对性能的影响各有不同，用户需要根据自己的需求来决定是否进行加密以及加密算法的选择。

除了对存储在表中的数据进行加密外，DM 还支持对存储模块、触发器的定义进行加密，即对存储过程、存储函数和触发器的定义进行加密。

存储过程加密示例：

```
SQL> CREATE OR REPLACE PROCEDURE PROC_ARG WITH ENCRYPTION(A IN INT, B INT)
AS
BEGIN
    A:=0;
    B:=A+1;
END;
```

存储函数和触发器的加密方式和存储过程一致，在函数名后面加上“WITH ENCRYPTION”即可。

26.9 资源限制

资源限制用于限制用户对 DM 系统资源的使用。表 26.9 列出了资源限制所包含的内容。

表 26.9 资源限制项列表

资源限制项	说明	最大值	最小值	缺省值
SESSION_PER_USER	在一个实例中，一个用户可以同时拥有	32768	1	系统所能

	的会话数量			提供的最大值
CONNECT_TIME	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）	144（1 天）	1	无限制
CONNECT_IDLE_TIME	会话最大空闲时间（单位：10 分钟）	144（1 天）	1	无限制
FAILED_LOGIN_ATTEMPS	将引起一个帐户被锁定的连续注册失败次数	100	1	3
CPU_PER_SESSION	一个会话允许使用的 CPU 时间上限（单位：秒）	31536000（365 天）	1	无限制
CPU_PER_CALL	用户的一个请求能够使用的 CPU 时间上限（单位：秒）	86400（1 天）	1	无限制
READ_PER_SESSION	会话能够读取的总数据页数上限	2147483646	1	无限制
READ_PER_CALL	每个请求能够读取的数据页数	2147483646	1	无限制
MEM_SPACE	会话占有的私有内存空间上限（单位：MB）	2147483647	1	无限制
PASSWORD_LIFE_TIME	一个口令在其终止前可以使用的天数	365	1	无限制
PASSWORD_REUSE_TIME	一个口令在可以重新使用前必须经过的天数	365	1	无限制
PASSWORD_REUSE_MAX	一个口令在可以重新使用前必须改变的次数	32768	1	无限制
PASSWORD_LOCK_TIME	如果超过 FAILED_LOGIN_ATTEMPS 设置值，一个帐户将被锁定的分钟数	1440（1 天）	1	1
PASSWORD_GRACE_TIME	以天为单位的口令过期宽限时间	30	1	10

注：“无限制”表示该资源限制项失效。

若用户为系统预设用户（包括 SYSDBA、SYSAUDITOR、SYSOSS），则关于口令的 6 项资源限制项自动失效。

另外安全版提供的用户 IP 地址限制，规则如表 26.10 所示。

表 26.10 IP 地址限制规则

IP 地址限制项		集合状态			
IP 地址集合	不允许的 IP 地址	空	N	空	N
	允许的 IP 地址	Y	空	空	Y
IP 限制结果		只允许属于集合 Y 的 IP 地址	只允许属于集合 N 以外的所有 IP 地址	允许所有 IP，即无任何 IP 地址限制	只允许属于 Y-N（差集）集合的 IP 地址

用户时间段限制，规则如表 26.11 所示：

表 26.11 时间限制规则

时间段限制项		集合状态			
时间段集合	不允许的时间段	空	N	空	N
	允许的时间段	Y	空	空	Y
时间段限制结果		只允许属于	只允许属于集	允许所有时间	只允许属于 Y-N

	集合 Y 的时间段	合 N 以外的所有时间段	段，即无任何时间段限制	（差集）集合的时间段
--	-----------	--------------	-------------	------------

具体的使用方式请参考 26.2.2 节。

26.10 客体重用

在普通的环境下，数据库客体（这里主要是数据库对象、数据文件、缓存区）回收后不做处理，直接分配给新来的请求，但是有些窃密者会利用这一点编写特殊的非法进程通过数据库管理系统的内存泄露来获取数据库系统的信息。

为防止非法进程利用数据库客体的内存泄露来攻击数据库，DM 主要从内存和文件两个方面进行了处理。可以通过在控制台 Console 中配置 `ENABLE_OBJ_REUSE` 参数值来启用该功能，设置为 1，启用该功能，设置为 0 关闭该功能，该配置项默认值为 0。

内存重用：DM 从系统分配内存及释放内存时均对内存内容进行清零，以保证不利用内存中前一进程所残留内容，且不泄漏 DM 的内容给其他进程。

文件重用：DM 在系统生成、扩展及删除文件时，对文件内容也进行了清零。

26.11 加密引擎

DM 系统中内置了常用的 DES，AES，RC4 等加密算法供用户使用，以此来保护数据的安全性。然而在有些特殊的环境下，这些加密算法可能不能满足用户的需求，用户可能希望使用自己特殊的加密算法，或强度更高的加密算法。DM 的加密引擎功能则可以满足这样的需求。

用户只需要按照 DM 提供的加密引擎标准接口，封装自己的加密算法，即可以在 DM 的存储加密中使用自己的加密算法，大大提高了数据的安全性。

根据系统的需求，第三方加密动态库需要提供如下的对外接口，如表 26.12 所示。

表 26.12 第三方加密接口定义

接口名称	功能说明
<code>cipher_get_count</code>	获取实现的算法个数
<code>cipher_get_info</code>	获取算法的基本信息
<code>crypto_cipher_encrypt_init</code>	加密初始化
<code>cipher_get_cipher_text_size</code>	计算给定长度的明文，加密后得到的密文所占的字节数。
<code>cipher_encrypt</code>	加密函数
<code>cipher_cleanup</code>	回收密码算法在处理过程中申请的系统资源
<code>cipher_decrypt_init</code>	解密初始化
<code>cipher_get_plain_text_size</code>	估算存放长度为 <code>cipher_text_size</code> 的密文对应的明文的最小缓冲区长度
<code>cipher_decrypt</code>	解密函数

cipher_hash_init	散列过程的初始化工作
cipher_hash_update	计算消息 msg 的散列值
cipher_hash_final	散列值的实际长度

第四部分 故障排除和性能优化

第 27 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。

27.1 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 B：动态视图）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

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_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 1，则系统还会在刚才的 log 目录下生成名为 log_commitNN.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。

27.2 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 B：动态视图），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、V\$WAIT_HISTORY 需要 ENABLE_MONITOR 参数设置为非 0 或者 2 后才会进行相关信息的收集。

27.3 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=c:\test.cpt

必选参数：SERVER 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
FILE	捕获文件及路径
HELP	打印帮助信息

27.4 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

27.5 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(8187))`

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(8187), value int64)`

该过程用于修改变量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 角色的用户才有权调用 `SP_SET PARA_VALUE`。

5. `SP_SET PARA_DOUBLE_VALUE (scope int, paraname varchar(8187), value 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**。

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. **SP_SET_SESSION_PARA_VALUE** (paraname varchar(8187), value int64)
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。

27.6 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 28 章 动态管理/性能视图

28.1 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 B。

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，如缓冲池动态信息表 **V\$BUFFER**。

28.2 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\$SESSION**、**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		
1	DMSERVER	FREESKYC-FB8846 DM DATABASE SERVER	
V7.0.1.28-BUILD(2011.09.14)	DB VERSION: 0X70001	2011-09-	
19 19:20:14	OPEN	NORMAL	0

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\$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							
1	0	SYSTEM	1	0	0	1920	1
2	1	ROLL	1	0	0	0	1
3	4	MAIN	1	0	0	16384	1
4	3	TEMP	2	0	0	1280	1

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\$BUFFER、V\$BUFFERPOOL、V\$EXP_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：

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（字典锁、行锁、表锁）、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：

TRX_ID	LTYPE		LMODE		TABLE_ID	ROW_IDX
1	429	DICT S	-1		0	
2	429	TABLE IX	1094		0	
3	429	TABLE S	1073741828		0	
4	429	DICT IS	301		0	
5	429	ROW X	1094		0	

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、V\$WAIT_CLASS 等。

例如查看系统中资源等待的情况。

```
SELECT CLASS_NAME,TOTAL_WAITS,TIME_WAITED FROM V$WAIT_CLASS;
```

结果：

CLASS_NAME	TOTAL_WAITS		TIME_WAITED	
1	IO READING	0 0		
2	ROW LOCK WAITING	0 0		
3	TABLE LOCK WAITING	0 0		
4	DICT LOCK WAITING	0 0		
5	TASK WAITING	0 0		

6	NETWORK WAITING	0	0
7	REDO LOG FLUSH WAITING	1	392
8	PAGE CONFLICT	0	0
9	BUFFER WAITING	0	0

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、PL/SQL 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\$PURGE_RSEG_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\$CACHEPLN、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仅仅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	
11 ROWS GOT		

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 29 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。

29.1 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。

29.2 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` 优化指示，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。

29.2.1 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。

29.2.2 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 读多个块，所以，基表扫描或索引全扫描的代价依赖于表的数据块数和多块读的参数值。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。

29.2.3 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。

29.3 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。

29.4 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 WHERE 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 WHERE 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。

29.5 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、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。

29.6 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：

```
EXPLAIN SELECT A.C1,B.D2 + 1 FROM T1 A, T2 B WHERE A.C1 = B.D1;
```

执行计划如下：

```
#NSET2: [7, 4, 0]
#PRJT2: [7, 4, 0]; EXP_NUM(2), IS_ATOM(FALSE)
#NEST LOOP INDEX JOIN2: [7, 4, 0]
#CSCN2: [6, 2, 0]; INDEX33556882(T2)
#SSEK2: [6, 0, 0]; SCAN_TYPE(ASC), IT1C1(T1), 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：命令流从上向下传递，数据流从下向上传递。执行计划中，最下面，最右边的先执行，同一层次中的操作符，由上向下执行。

#SSEK2: [6, 0, 0]; scan_type(ASC), IT1C1(T1), scan_range[T2.D1,T2.D1]表示从表 T1 中获取数据，且访问路径是 SSEK2，表示的是二级索引方式；

#CSCN2: [6, 2, 0]; INDEX33556882(T2)表示使用全表扫描方式来获取 T2 表中的数据；

#NEST LOOP INDEX JOIN2: [7, 4, 0]表示对传递上来的两个数据集进行嵌套循环地生成结果集；

#PRJT2: [7, 4, 0]; exp_num(2), is_atom(FALSE)表示对嵌套循环的结果集进行投影，返回查询项中的 2 列值；

#NSET2: [7, 4, 0]最后输出结果。

29.7 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。

29.8 并行查询

29.8.1 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 29.1 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。
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 表示自动配置并行工作线程个数（与物理 CPU 核数相同）；2 表示手动设置并行工作线程数。
PARALLEL_THRD_NUM	10	静态	用来设置并行工作线程个数。取值范围：1~1024。仅当 PARALLEL_POLICY 值为 2 时才启用此参数。

29.8.2 确定并行任务数

支持两种方式指定并行查询的任务数：一是在 ini 参数中，使用 MAX_PARALLEL_DEGREE 设置默认值，此方式指定的值全局有效；二是在 sql 语句中，使用 HINT 方式，通过“PARALLEL”关键字来特别指定，此方式指定的值只在本条 sql 语句中有效。只要出现 HINT 方式，则 HINT 方式指定的并行任务数就覆盖 MAX_PARALLEL_DEGREE 设置默认值。

1. 在 ini 参数中设置默认值

在使用并行查询以前，您需要对 ini 参数列表里的 MAX_PARALLEL_DEGREE 参数进行赋值。该参数是 ini 参数列表中的固有参数，专门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。

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

您也可以在特定的 sql 语句中，通过在 HINT 中使用“PARALLEL”关键字来特别指定并行任务数。通过“PARALLEL”关键字特别指定的并行任务数，会覆盖 MAX_PARALLEL_DEGREE 设置默认值。如果单条查询语句没有特别指定，则依然使用默认并行任务个数。“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(SYSOBJECTS 4) */ * FROM SYSOBJECTS;
```

其中，表名不能省略，并行任务个数是可以省略的。若省略了并行任务个数，则仍然使用默认并行任务个数 3，例如：

```
SELECT /*+ PARALLEL(SYSOBJECTS) */ * 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(SYSOBJECTS 1) *//*+ PARALLEL(SYSOBJECTS 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。

29.8.3 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 表示自动配置并行线程个数（与物理 CPU 核数相同）；2 表示手动设置并行线程数。若已设置了 PARALLEL_POLICY 为 0 或 1，则不再需要 PARALLEL_THRD_NUM 参数。其次，使用 PARALLEL_THRD_NUM 设置并行工作线程个数，取值范围：1~1024。仅当 PARALLEL_POLICY 值为 2 时才启用此参数。

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。

29.8.4 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。

29.8.5 使用场景

对于无特殊要求的并行查询用户，可以使用默认并行任务数 `MAX_PARALLEL_DEGREE`。只需要在 `ini` 参数中设置好如下 3 个参数，然后执行并行 `sql` 查询语句，就可以启用并行查询。需要设置的 3 个参数如下：

<code>MAX_PARALLEL_DEGREE</code>	3
<code>PARALLEL_POLICY</code>	2
<code>PARALLEL_THRD_NUM</code>	4

特别需要提醒的是，只有 `PARALLEL_POLICY` 为 2 时，才需要 `PARALLEL_THRD_NUM` 参数（如上所示）。当 `PARALLEL_POLICY` 为 0 或 1 时，则不需要并行工作线程数 `PARALLEL_THRD_NUM`。如下为 `PARALLEL_POLICY` 为 1 时，只要设置 2 个参数的情况：

<code>MAX_PARALLEL_DEGREE</code>	3
<code>PARALLEL_POLICY</code>	1

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。

29.9 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 1 时，会启用计划重用。

29.10 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。

与开启计划重用方式相同，通过设置 `INI` 参数 `RS_CAN_CACHE` 来控制结果集重用，当置为 1 时，会启用结果缓存。遇到重复执行的 `SQL` 语句时，会在结果集缓冲区查找对应的结果集，直接使用。

第 30 章 SQL 调优

30.1 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。

30.2 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：

1. 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 BLKUP 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。

如果计划中对某小表进行了全索引扫描并且这是最优途径，那么可以将该表数据缓存到系统的 KEEP 缓冲区以避免频繁淘汰。该功能是通过建表时指定 STORAGE 的 CACHE 选项完成的。如：CREATE TABLE small_t(...) STORAGE (CACHE = KEEP);

2. 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。

3. 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。

30.3 确定高负载的 SQL

在设置性能监视级别为 2 后（默认值），可以通过查询动态视图 V\$SQL_HISTORY 来确定高负载的 SQL 语句。例如：

```
SELECT TOP 5 TOP_SQL_TEXT, SEC_SQL_TEXT, THRD_SQL_TEXT, TIME_USED FROM
V$SQL_HISTORY ORDER BY 4 DESC;
```

30.4 自动 SQL 调整

使用查询优化向导工具，输入需要调整的 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。

30.5 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 SYSDDUAL
WHERE EXISTS (SELECT 1 FROM t1 WHERE condition1);
```

判断 x 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 SYSDDUAL。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传层向上传递。不仅如此，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

索引并未越多越好。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(1)或者 COUNT(列名)都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. 使用优化器调试 (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。

30.6 使用优化器提示

DM 查询优化器采用基于代码的方法。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，它提供了一种人工干预优化器的方法使优化器有目的地选择执行计划。

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

假设表 t1 上 id 和 name 列上都存在着单列索引。

例 1

```
SELECT * FROM t1 WHERE id > 2011 AND name < 'xxx';
```

如果 id 列上能过滤更多数据，可以指示用索引 idx_t1_id。

```
SELECT * FROM t1 INDEX idx_t1_id WHERE id = 1 AND name = 'xxx';
```

例 2

```
SELECT * FROM t1 WHERE id > 2011 AND name < 'xxx' ORDER BY name;
```

如上面例子所示，虽然 id 能过滤更多数据，但是考虑到后面的排序操作，可以指示使用索引 idx_t1_name，这样可以省略掉 ORDER BY。

```
SELECT * FROM t1 INDEX idx_t1_name WHERE id > 2011 AND name < 'xxx' ORDER BY name;
```


第五部分 数据高可用性

第 31 章 故障恢复

31.1 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。

31.2 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。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';
```

31.3 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。归档日志包括以下四种类型：

31.3.1 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。

31.3.2 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，或从备机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。只能配置一个实时归档。

31.3.3 同步归档

在 REDO 日志写入日志文件后触发，由归档线程通过 MAL 系统发送 REDO 日志到远程系统，最多可以设置 1 个远程同步归档。

31.3.4 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。

31.4 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块写入磁盘，以保证当前日志所“保护”的数据块都已安全写入磁盘，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；当自由日志空间不足时，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，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。

执行 SELECT CHECKPOINT(10);会将系统中 10%脏页写入磁盘。

31.5 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，其回滚数据块的分配、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。

31.6 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。

31.6.1 恢复流程

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。

31.6.2 并行恢复

为了提高系统恢复的效率，达梦数据库采用了并行恢复技术，同时重做 REDO 日志，提高恢复性能。为保证恢复的正确执行，理论上应严格按日志生成顺序重做 REDO 日志。事实上，按日志生成顺序执行重做的根本原因是为了保证 REDO 日志重做完成后，数据页中保存的是最后一次更新操作生成的数据。达梦数据库在恢复时，将不相关数据页的 REDO 日志收集出来，生成多个独立的子任务，并行执行重做，充分利用多个 CPU 的运算能力，提升恢复效率。通过 INI 参数 LOG_REDO_THREAD_NUM 可配置重做线程数，默认是 2 个线程。

31.7 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备份恢复相关章节详细介绍。

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 32 章 备份与还原

32.1 概述

数据库的备份与还原是系统容灾的重要方法。在一个生产系统中，数据库往往处于核心的地位。为了保证数据的安全，人们想出了各种各样的方法，比如创建 RAC 环境，配置异地容灾，使用磁盘阵列，备份与还原等。备份意味着把重要的数据复制到安全的存储介质上，还原则意味着在必要的时候再把以前备份的数据复制到最初的位置，以保证用户可以访问这样的数据。

DM 数据库管理系统支持的备份方式包括物理备份、逻辑备份和 B 树备份，其中 B 树备份是介于物理备份和逻辑备份之间的一种形态。物理备份分为数据库级的备份和用户表空间级的备份；B 树备份分为数据库级的备份和用户表级的备份。为了提高备份恢复的安全性，减少备份文件占用磁盘空间的大小，系统支持对备份数据加密和压缩。

32.2 备份

32.2.1 完全备份与增量备份

达梦数据库管理系统支持数据库和表空间的完全备份和增量备份。完全备份是指一个备份包含指定数据库或表空间的所有数据。当用户进行完全备份时，系统需要做的事情就是将一个数据库或表空间所包含的文件内容都拷贝到用户指定的备份文件中。完全备份包含了一个数据库或表空间的所有数据，这样的备份通常很大，需要占用很多的磁盘空间，尤其是系统在运行过程中，定期产生的完全备份所占用磁盘上的空间将非常大，同时，其备份过程也会很长。为了解决这个问题，DM 提供了增量备份的功能，即不用备份当前数据库或表空间的所有数据，而只是以该数据库或表空间最近的一个备份为基础，仅仅拷贝自该备份以来所有被修改过的数据，形成一个增量备份。相对于完全备份而言，增量备份小得多，同时其备份的时间也会更短。

当用户选择在数据库或表空间上进行增量备份时，系统会自动将该数据库或表空间最近的一个备份作为其基础备份，基础备份可以是完全备份也可以是一个增量备份，但如果该数据库或表空间自创建以来还没有进行过备份，则增量备份无法进行。还要特别注意的是，如果该数据库自最近一次还原以来还没有进行过备份，则增量备份也无法进行，这是因为考虑到系统的最近那一次的还原会使以前的备份对于当前系统都失效，鉴于这一点，系统管理员在每次系统还原后就立即作一次完全备份是一个很好的习惯。

由于增量备份是基于某一个备份而生成的，所以对于增量备份的还原也有一个限制条件，即增量备份将要还原的系统，必须是刚刚利用其基础备份还原过并且还没有被启动过服务器的系统，否则增量备份无法进行还原。这个限制必须非常严格，是因为用户的数据需要受到严格的保护。如果利用基础备份还原后系统被启动过，则无法保证是否有新的事务修改过数据库，而此时利用增量备份还原可能会导致那部分被修改过的数据丢失。为了让用户能很轻松地完成这个严格的还原过程，DM 提供了在还原增量备份时自动还原基础备份的功能。需要说明的是，当该增量备份的基础备份也是增量备份时，可能会引起级联的基础备份还原。

完全备份与增量备份各有缺点和优点。完全备份的优点是，只需要一次备份的数据，就能完全将数据库还原到备份时的状态，缺点是由于对整个数据库的数据备份，因此备份文件

需要比较大的磁盘空间。由于增量备份是基于数据库的某一次备份(增量备份或是完全备份)做的操作,因此在还原时若增量备份所依赖的基础备份丢失,则该备份文件就变得不可用,相对于完全备份而言,不够可靠;另一方面,由于只是对某一段时间内修改数据的备份,因此备份文件比较小,备份速度比较快。需要说明的是,增量备份所依赖的基础备份有且仅有一个完全备份。

32.2.2 联机备份与脱机备份

DM 提供了两种方式执行备份操作,包括脱机方式和联机方式。所谓脱机方式,即数据库服务器关闭以后所进行的备份,联机方式则是指数据库服务器正常运行过程中进行的备份。由于脱机方式执行备份时需要关闭数据库,所以会对用户的访问产生影响,而且很多重要的数据库都需要进行 7 * 24 小时访问。脱机备份的优势在于无需备份日志保证数据的一致性,因为在备份过程中用户无法修改数据库中的数据,另外无论是否处于归档模式,都能进行脱机备份。联机方式支持数据库备份、用户表空间备份和用户表备份,而脱机方式只支持数据库备份。利用这两种方式进行操作时,需要注意的问题有两点:首先,在进行脱机备份时,必须保证系统是正常关闭的,如果处于不正常关闭的状态,脱机备份无法进行;其次,在进行联机库级备份和表空间备份时,必须保证系统处于归档模式,否则联机备份不能进行。

32.2.3 物理备份

32.2.3.1 备份数据库

为防止重要数据丢失,对整个数据库进行备份是最保险的方式,利用备份文件可以将数据库还原到备份时的状态。实现备份数据库的基本原理是将数据库对应的物理数据文件,包括数据页文件和回滚页文件的内容拷贝到备份文件中。DM 支持多用户并发操作数据库,在用户联机备份过程中,可能有其它用户在操作数据库,因此对于联机备份,除了备份物理数据文件还需要备份物理日志文件。备份日志文件的目的是保证数据库的完整性,这里提到的物理日志文件是归档日志文件。归档日志文件记录了备份期间所有用户对数据库进行的操作,还原时,利用数据文件和归档日志文件的内容即能将数据库恢复到备份时的状态。备份时间与备份数据量成正比,保证数据不丢失的情况下,少备份数据即能提高备份效率,因此拷贝文件不是简单地将数据文件和日志文件的所有内容拷贝到备份文件,而是仅仅拷贝使用过的数据页,这样能大大减少磁盘 IO,提高备份性能。

DM 提供联机 and 脱机两种方式执行数据库备份。联机方式通过 SQL 命令提供,脱机方式则是通过备份工具提供。

1. 联机备份

```
BACKUP DATABASE [FULL|BTREE|INCREMENT WITH BACKUPDIR '<备份目录>','<备份目录>']  
TO <备份名> [BAKFILE '<备份路径>'] [BACKUPINFO '<备份描述>'] [MAXSIZE <限制大小>]  
[IDENTIFIED BY <密钥>][WITH ENCRYPTION<TYPE>]][COMPRESSED] [PARALLEL '<并行映射文件路  
径>'];
```

参数说明:

(1) FULL|INCREMENT|BTREE 备份类型, FULL 表示完全备份, INCREMENT 表示增量备份, BTREE 表示 B 树备份;

(2) <备份目录> 基础备份所在的目录, 最大长度为 256 个字节;

(3) <备份名> 备份的名称, 在 DMDDBMS 中以此标识不同的备份;

(4) <备份路径> 备份文件存放的完整路径;

- (5) <备份描述> 备份的描述信息;
- (6) <限制大小> 最大备份文件大小, 最小值为 16M, 最大值 2G;
- (7) <密钥> 备份加密通过使用 IDENTIFIED BY 来指定密码;
- (8) WITH ENCRPYTION <TYPE> 用来指定加密类型, 0 表示不加密, 1 表示简单加密, 2 表示复杂加密;
- (9) COMPRESSED 用来指定是否压缩。如果使用, 则表示压缩, 否则表示不压缩;
- (10) PARALLEL 用来指定并行映射文件的完整路径。
- (11) <并行映射文件路径> 并行映射文件的完整路径, 最大长度为 256 个字节。
- 备份成功后会在备份默认目录下生成备份文件, 若没有指定备份文件名, 则系统自动生成以“数据库名_日期.bak”格式的备份文件, 通过该备份文件可以进行数据库还原。
- 例如, 执行数据库备份, 备份类型为完全备份, 备份文件名为 C:\database_full.bak。

```
BACKUP DATABASE FULL TO full_bak BAKFILE 'C:\database_full.bak';
```

2. 脱机备份

```
DMBACKUP KEYWORD=value
```

确认 DM 数据库服务器已停止后, 在 Windows “命令提示符”窗口中输入带参数的 DMBACKUP 命令启动该工具, 参数说明见表 32.1。

表 32.1 DMBACKUP 工具参数说明

参数	含义	备注
INI_PATH	配置文件路径, 最大长度为 256 字节	必选参数
NAME	备份名, 最大长度为 128 字节	必选参数
BAKFILE_PATH	备份文件路径	可选参数
TYPE	备份类型, 默认为完全备份, FULL 表示完全备份, INCREMENT 表示增量备份, BTREE 表示 b 树备份	可选参数
BASE_BAK_DIR	增量备份时的指定基备份目录	可选参数, 仅对增量备份有效
DESCRIBE	备份文件描述, 即备注信息, 注意: 描述如果含有空格, 需要用双引号括起来, 否则会报错	可选参数
MAXSIZE	备份文件最大大小, 取值范围为: 16-2048, 单位为 M (兆), 默认为 512	可选参数
ENCRYPT	加密类型, 0 表示不加密, 1 表示简单加密, 2 表示复杂加密	可选参数
PASSWORD	加密密码, 长度最大为 128 字节, 如果选择不加密, 此处不必指定	可选参数
COMPRESS	指定备份为压缩, 如果不选择, 则表示不压缩	可选参数
PARALLEL	指定并行备份映射文件路径	可选参数
HELP	打印帮助信息	

例如, 对名为 test 数据库执行一个完全备份:

```
DMBACKUP TYPE=FULL INI_PATH=c:\dmdata\dameng\dm.ini NAME=test
```

32.2.3.2 备份用户表空间

DM 除了支持对整个数据库备份外, 还支持备份单个用户表空间。与备份数据库一样, 备份表空间也支持完全备份和增量备份。不过, 备份表空间仅仅支持联机备份。备份用户表空间提供了一种更灵活的方式备份用户数据。通常情况下, 不同的用户表空间可以部署在不同的物理磁盘上, 当某个或某几个磁盘出现介质故障时, 只需要利用特定表空间备份还原出现故障的表空间, 在保证数据完整性的前提下, 提高了备份还原的效率。与备份数据库一样, 备份用户表空间的基本原理是拷贝数据文件。此外, 执行备份用户表空间时, 需要保证系统

处于归档模式，因为表空间的还原依赖于归档日志文件。

系统提供联机 SQL 命令执行表空间备份，不支持对系统表空间、日志表空间、回滚表空间、临时表空间执行备份操作。

```
BACKUP TABLESPACE <表空间名> [FULL| INCREMENT WITH BACKUPDIR '<备份目录>'{'<备份目录>'}] TO <备份名> [BAKFILE '<备份路径>'] [BACKUPINFO '<备份描述>'] [MAXSIZE <限制大小>] [IDENTIFIED BY <密钥>][WITH ENCRYPTION <TYPE>]] [COMPRESSED];
```

例如备份名为 t1 的表空间，备份类型为 FULL，备份文件名为 'C:\tablespace_bak.bak'。

```
BACKUP TABLESPACE t1 FULL TO TS_BAK BAKFILE 'C:\tablespace_bak.bak';
```

32.2.4 逻辑备份

达梦数据库管理系统提供了对数据库进行逻辑备份的命令行工具 `dexp.exe`，位于安装目录的 `bin\` 目录下。系统管理员可以利用它，在命令行方式下对 DM 数据库进行联机逻辑备份。通过设置参数，可以灵活地选择是否备份二级索引、数据行、权限和索引等信息，并支持对远程数据库的访问。

`dexp` 工具可以对本地或者远程数据库进行数据库级、模式级和表级的逻辑备份。备份的内容非常灵活，可以选择是否备份索引、数据行和权限，是否忽略各种约束（外键约束、非空约束、唯一约束等），在备份前可以选择生成日志文件，记录备份的过程以供查看。

表 32.2 dexp 的参数

参数	含义	备注
USERID	用户名/口令@主机名:端口号#证书路径 如： SYSDBA/SYSDBA@server:5236#ssl_path@ssl_pwd	必选参数，其中主机名、端口号和证书路径为可选项，如果不指定，使用缺省值
FILE	输出文件 (EXPDAT.DMP)	必选参数
DIRECTORY	直接路径 (N)	可选参数
FULL	导出整个数据库 (N)	可选参数
OWNER	所有者用户名列表,你希望导出哪个用户的对象	可选参数
SCHEMAS	导出的模式列表	可选参数
TABLES	表名列表 ,指定导出的 table 名称	可选参数
QUERY	用于对导出表的数据进行过滤的条件 例如: Tables=info:query="where condition", 通过 query 所制定的过滤条件, 对表 info 的导出	可选参数
PARALLEL	用于指定导出的过程中所使用的线程数目	可选参数
EXCLUDE	忽略指定的对象	如: EXCLUDE= (CONSTRAINTS, TABLES)
CONSTRAINTS	导出约束 (Y)	可选参数
GRANTS	导出权限 (N)	可选参数
INDEXES	导出索引 (Y)	可选参数
TRIGGERS	导出触发器 (Y)	可选参数
ROWS	导出数据行 (Y)	可选参数
LOG	屏幕输出的日志文件	可选参数
PARFILE	参数文件名, 如果你 exp 的参数很多, 可以存成参数文件	可选参数
FEEDBACK	每导出若干行进行导出行数信息打印	可选参数
COMPRESS	是否压缩导出数据文件(N Y)	可选参数, 默认为 N
FILESIZE	各转储文件的最大尺寸	可选参数

DROP	导出表后是否将表删除掉(N)	可选参数
HELP	显示帮助信息	可选参数

以下为简单的命令举例，仅供参考：

1. 不同方式的导出

1) FULL 方式的导出

```
dexp USERID=SYSDBA/SYSDBA FILE=db_str.dmp LOG=db_str.log FULL=Y
```

注意：在导出的过程中最好使用 **DIRECTORY** 参数以指定路径，如果不指定路径，程序将根据当前的运行环境来设置相应的导出路径；

2) OWNER 方式的导出

将用户 user001 拥有的所有对象全部导出。

```
dexp SYSDBA/SYSDBA FILE=db_str.dmp LOG=db_str.log OWNER=USER01
```

3) SCHEMAS 方式的导出

可以将实例中的某个模式进行导出，如：

```
dexp SYSDBA/SYSDBA FILE=db_str.dmp LOG=db_str.log SCHEMAS=USER01
```

注意：一般情况下用户与模式导出是相同的，但是用户可以包含多个模式，在这种情况下模式的导入与导出是用户导入导出的一个子集；

4) TABLES 方式的导出

```
dexp SYSDBA/SYSDBA#c:\data FILE=db_str.dmp LOG=db_str.log TABLES=table1,table2
```

将 SYSDBA 登录的默认数据库上的表 table1, table2 上的所有数据行、约束、索引等信息全部导出到文件，其中 “c:\data” 为数字证书路径，为可选项。

当仅导出一个表时，可以加上参数 **QUERY**，如：

```
dexp SYSDBA/SYSDBA FILE=db_str.dmp LOG=db_str.log TABLES=table1 QUERY="where condition"
```

使用 **EXCLUDE** 来排除一定的约束或表数据，如下面命令不导出约束及索引。

```
dexp SYSDBA/SYSDBA FILE=db_str.dmp LOG=db_str.log TABLES=table1
EXCLUDE=( CONSTRAINTS, INDEXES)
```

2. EXCLUDE、FEEDBACK 的使用

EXCLUDE：此参数用以指明导入时忽略的对象种类，如以下示例指明将当前用户下的 table1、table2 的对象信息导出，但不导出约束。

```
dexp SYSDBA/SYSDBA FILE=c:\db_str.dmp LOG=db_str.log TABLES=table1,table2
EXCLUDE=( CONSTRAINTS)
```

FEEDBACK：此参数用以指明在导入数据的过程中每多少行打印提示信息，如下面的例子将表 table1,table2 上的相关数据以及对象信息导入到文件 dexp.dmp 中，并且在导入表的过程中每 10000 行打印出进度信息。

```
dexp SYSDBA/SYSDBA@192.168.0.95 FULL=Y SHOW=Y FILE=dexp.dmp DIRECTORY=C:\test\
FEEDBACK=10000
```

3. FILE、DIRECTORY、FILESIZE 的使用

FILE：此参数用于指定导出的文件，可以包含多个文件，用逗号分隔。文件名可以包含通配符 %U 用于作为自动扩充文件的文件名模板。%U 表示为 2 个字符宽度的数字，由系统自动生成，起始为 01。

DIRECTORY：此参数用于指定导出文件及日志文件生成的目录，默认为 dexp 所在的目录。如果 FILE 和 LOG 参数指定的文件包含生成路径，则 FILE 和 LOG 参数中指定的路径将替代 DIRECTORY 所指定的路径，如果 FILE 和 LOG 参数指定的文件未包含路径信息，则文件将被生成到 DIRECTORY 指定的目录下。

FILESIZE：此参数用于指定单个导出文件大小的上限。可以按字节[B]、K[B]、M[B]、G[B]的方式指定大小。

以下示例表示以 FULL 方式导出实例，每个导出文件大小为 128m，文件名以 exp01.dmp

自动扩充，存储位置为 d:\exp 目录。

```
dexp USERID=SYSDBA/SYSDBA DIRECTORY=d:\exp FILE=exp%u.dmp FILESIZE=128m
FULL=y
```

4. 参数文件的使用

可以将常用的命令保存到文件中，然后使用参数文件进行导出、导入操作，如下面例子所示。

```
dexp userid=SYSDBA/SYSDBA parfile=c:\para.txt
```

其中参数文件 c:\para.txt 的内容如下。

```
FILE=db_str.dmp
LOG=db_str.log
TABLES=table1
QUERY="where condition"
ROWS=y
EXCLUDE=(INDEXES, CONSTRAINTS)
```

32.2.5 B 树备份

DM 除了提供常规的备份方式以外，还提供了一种新的备份方式——B 树备份。B 树备份是一种介于物理备份与逻辑备份的实现方式，支持单表备份和库级备份。B 树备份、还原实现了数据的无损恢复，效率高于逻辑备份、还原。B 树备份可单独备份一个表，也可单独备份某一个分区子表，或者备份整个数据库。库级的备份、还原可解决数据文件扩展后无法收缩的问题。

32.2.5.1 备份数据库

B 树备份与还原的实现是基于 DM 中的数据都是以索引形式组织的，因此只要能将所有索引的数据全部备份下来，即完成了对整个数据库的备份。由于 B 树备份的实现方式，导致备份文件中数据页是与其在索引中的相对顺序组织的，因此还原时，不能像普通还原那样直接拷贝数据到数据库文件，而是需要利用备份的数据进行索引页面重建。同时，由于索引重建，重新申请页面导致还原后索引中数据页所在的文件号以及页面号不一定与备份时的一致，因此无法利用备份时的日志文件。B 树备份无法利用日志文件，那么如何保证备份的数据库处于一致的状态呢？B 树备份数据库支持联机备份和脱机备份两种方式。对于联机 B 树备份，开始备份时，系统会对整个数据库上排它锁，防止其它用户对数据库进行更新操作；对于脱机 B 树备份，系统通过检查待备份的库是否处于正常关闭状态，保证备份的数据库处于一致的状态。区别于普通备份，B 树备份没有完全备份和增量备份的概念。B 树库级备份等价于普通备份的完全备份。B 树库级备份的联机备份接口与脱机工具与普通数据库备份一致，具体用法参考 32.2.3 节。

举例说明：

(1) 联机备份：执行一个 B 树类型数据库备份，备份文件名为 C:\database_btree.bak。

```
BACKUP DATABASE BTREE TO btr_bak BAKFILE 'C:\database_btree.bak';
```

(2) 脱机备份：对数据库 TEST 执行一个 B 树备份。

```
DMBACKUP TYPE=BTREE INI_PATH=c:\dmdata\dameng\dm.ini NAME=test;
```

32.2.5.2 备份用户表

B 树备份除了提供备份数据库的功能，还支持备份用户表。假如，某个用户因为误操作导致某个表被破坏，而数据库中其它对象是完好的。若需要恢复该表，可以借助还原表所属的表空间实现，如果表跨表空间，则涉及的表空间都需要还原，或是还原整个数据库才能达到恢复表的目的。B 树备份提供表级备份的功能，使得只需还原特定的表即可，大大提高了备份还原的灵活性。

DM 中所有表都以索引方式组织，包括三种类型索引结构：B 树索引，LIST 索引和列存储索引。一个表有且仅有一个聚簇索引，若表有唯一性约束，或是表上显式建有索引，那么该表还包含一个或多个二级索引，聚簇索引和二级索引包含了表中的所有数据。表的底层存储方式为 B 树备份用户表提供了可行性，因为只需要备份表所包含的索引数据，即完成了对表的备份。备份单个表功能实现使得 B 树备份比普通备份更灵活，为用户提供了更多层次的备份方式。另外，相比于逻辑备份，B 树备份是介于物理备份与逻辑备份的实现方式，因此效率相对更高。

系统提供联机 SQL 命令执行 B 树备份用户表。对于表级备份，系统不支持临时表和垂直分区子表的备份。

```
BACKUP TABLE <表名> TO <备份名> [BAKFILE '<备份路径>'] [BACKUPINFO '<备份描述>']  
[MAXSIZE <限制大小>] [IDENTIFIED BY <密钥>][WITH ENCRYPTION <TYPE>]] [COMPRESSED];
```

命令参数的含义可以参见 32.2.3 节。

例如备份名字为 test 的表，备份文件名为'C:\table_bak.bak'。

```
BACKUP TABLE test TO table_bak BAKFILE 'C:\table_bak.bak';
```

32.2.6 备份加密与压缩

备份文件中包含了数据库中的原始数据，为提高备份数据的安全性，无论是普通备份还是 B 树备份，DM 都提供了对备份数据进行加密的功能。系统中支持简单加密和复杂加密。简单加密和复杂加密实现原理都是在备份过程中利用经典的加密算法对数据页面加密，然后将密文写入备份文件，区别在于复杂加密方式对写入备份文件的每一页数据都进行加密，而简单加密方式只是选择性地对某些页面进行加密。显然，简单加密的效率相对于复杂加密效率要高，但同时，简单加密的备份数据不如复杂加密安全。此外，DM 还支持备份数据压缩功能。压缩功能的实现能有效缩短备份文件的大小，减少所占的磁盘空间。不论是加密还是压缩，都是在正常执行备份数据时的额外操作，与此同时，压缩和加密的备份文件用于还原时，还需要执行解压缩和解密操作，因此会对备份、还原性能造成一定的影响。系统管理员应根据实际情况，在磁盘空间利用率、备份、还原性能，以及备份数据安全性之间做权衡，确定在执行备份操作时是否需要设置压缩，加密参数。

举例说明：

(1) 联机备份数据库，并对备份数据压缩，备份文件存放路径为 C:\database_cpr.bak。

```
BACKUP DATABASE FULL TO cpr_bak BAKFILE 'C:\database_cpr.bak' COMPRESSED;
```

(2) 联机备份名为 t1 的表空间，备份类型为 FULL，对备份数据进行简单加密，加密密码为 ABCDEF，备份文件存放路径为 C:\tablespace_bak.bak。

```
BACKUP TABLESPACE t1 FULL TO ts_bak BAKFILE 'C:\tablespace_bak.bak'  
IDENTIFIED BY ABCDEF WITH ENCRYPTION 1;
```

(3) 联机备份 SYSDBA 模式下的 test 表，对备份数据进行复杂加密，加密密码为 ABCDEF，并且对备份数据压缩。备份文件存放路径为 C:\table_bak.bak。

```
BACKUP TABLE SYSDBA.test TO table_bak BAKFILE 'C:\table_bak.bak'  
IDENTIFIED BY ABCDEF WITH ENCRYPTION 2 COMPRESSED ;
```

32.2.7 并行备份

为了提高数据库备份性能，针对物理备份，系统提供了并行备份的功能。并行备份的核心思想是对数据库的备份拆分为对多个表空间的备份。系统在执行备份前，将物理存储位置临近的表空间作为基本备份单位，随后同时备份不同的备份单位，从而使得系统在执行备份时，充分利用了 CPU 资源和磁盘组资源，达到提升备份性能的目的。利用并行备份功能，需要系统管理员配置并行映射文件，映射文件格式如下所示。映射文件配置内容与系统并行备份的性能直接相关，因此，系统管理员应在充分了解数据库中各个表空间所在磁盘的物理位置，合理设置配置文件。此外，每个备份单位设置的备份目录应尽量在不同的磁盘组，这样能保证读写文件都能充分利用磁盘 IO，使备份性能最优化。

```
[表空间名, 表空间名, .....]=备份目录 1
[表空间名, 表空间名, .....]=备份目录 2
[表空间名, 表空间名, .....]=备份目录 3
.....
[default]=备份目录 N
```

说明：

1. 备份目录必需是本地物理磁盘上已经存在的目录；
2. 设置表空间名必需存在，若不存在则报错；
3. 每一行只能有一个“[表空间名, 表空间名.....]=备份目录”项；
4. 自动忽略同一备份目录下对应的多个同名表空间名，只取其中一个；
5. 一个表空间名不能对应多个备份目录，否则报错；
6. 对于相同的备份目录，DM7 自动将它们作为一个备份单位；
7. default: 没有显示设置的表空间，备份数据都存放在 default 对应的目录下；
8. [default]行必需存在，DM7 认为[default]为最后一行，不统计[default]后续行的设置；
9. 映射文件的扩展名为 parallel；
10. 不允许显示设置系统表空间的磁盘目录；
11. 对于含有特殊字符的表空间名，比如逗号、空格、中括号、等号，系统不保证解析映射文件的正确性。

举例说明：

(1)假设数据库数据分布在三个磁盘组 DISK1, DISK2 和 DISK3 中。SYSTEM, MAIN, RLOG, ROLL 表空间在磁盘组 DISK1；TS1, TS2 表空间在磁盘组 DISK2；TS3 表空间在 DISK3，配置映射文件 bak.parallel 如下。

```
[TS1, TS2]=D:\DAMENG_1
[TS3]= C:\DAMENG_2
[default]=E:\DAMENG
```

从上述配置内容可知，TS1, TS2 作为一个备份单位，备份数据存储在 D:\DAMENG_1 中；TS3 作为一个备份单位，备份数据存储在 C:\DAMENG_2 中；系统表空间作为一个备份单位，备份数据存储在 E:\DAMENG 目录中

(2) 利用 配置文件 E:\bak.parallel 执行联机并行备份操作，主备份文件为 C:\parallel.bak，各个子备份分别在 C:\DAMENG_1, D:\DAMENG_2 和 E:\DAMENG 目录中。

```
BACKUP DATABASE FULL TO PARA_BAK BAKFILE 'C:\parallel.bak' PARALLEL 'E:\ bak.parallel '
```

(3) 利用配置文件 E:\bak.parallel 执行脱机并行全量备份操作，主备份文件为 C:\parallel02.bak，各个子备份分别在 C:\DAMENG_1, D:\DAMENG_2 和 E:\DAMENG 目录中。

```
DMBACKUP TYPE=FULL INI_PATH=c:\dmdata\dameng\dm.ini NAME=TEST
PARALLEL=E:\bak.parallel
```

32.3 还原

32.3.1 联机还原与脱机还原

当系统由于种种原因出现介质故障时，需要利用系统在早些时候建立的备份和归档日志进行还原。DM 支持多层次的备份、还原，包括表级\表空间级以及数据库级。对于数据库级的还原只支持脱机还原，不提供联机还原的功能；对于表空间级和表级还原，只支持联机还原，不支持脱机还原的功能。用户可以根据具体受损数据内容选择合适的备份进行还原，若只有一个或几个表数据被破坏，则通过还原用户表功能即能还原表数据；若只有一个或几个表空间被破坏，则可以通过还原指定的几个表空间实现数据恢复；若整个数据库数据受损，则只能通过还原数据库实现数据恢复。表级、表空间级和数据库级的备份粒度逐步加大，备份文件大小以及还原消耗时间都逐步递增，因此建议用户根据实际应用情况适时对表、表空间和数据库做备份，以备恢复数据时使用。以下三节将分别介绍还原数据库，还原表空间和还原用户表功能。

32.3.2 还原数据库

DM 支持利用还原工具对数据库进行还原。根据不同备份类型，包括联机备份、脱机备份、完全备份以及增量备份，还原工具选择不同的还原策略实现恢复用户数据的目的。首先讨论联机备份与脱机备份，脱机备份仅包含数据页文件的内容，而联机备份文件除了包含数据页文件内容以外，还包括备份过程中产生的日志文件内容。因此在利用联机备份还原时，除了拷贝数据页至数据库文件以外，还需要重做备份的日志文件内容，达到还原数据库的目的。接下来讨论完全备份与增量备份，对于增量备份，还原工具会根据用户提供备份的信息搜集它的基础备份，若基础备份仍然为增量备份，则工具会继续搜集，直至搜集到一个类型为完全备份的基础备份为止。因此在这个过程中，工具会构造一个基础备份链表，备份链表基于备份间的依赖关系组织。比如，备份 A 是备份 B 的增量备份，则可以说 A 依赖于 B。若用户首先进行了一次完全备份 A，然后在 A 的基础上做了增量备份 B，随后基于增量备份 B 又做了增量备份 C，最后基于增量备份 C 做了增量备份 D。那么此时利用增量备份 D 还原数据库时，系统搜集到得备份链表如 32.1 所示，执行还原时，首先利用完全备份恢复 A 进行还原，然后依次利用增量备份 B、C 和 D 还原，最终完成整个还原过程。需要说明的是，若备份类型为 B 树备份的数据库备份是不存在增量备份的。

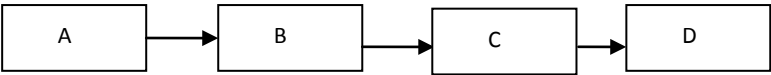


图 32.1 备份链表

还原工具的用法如下：

DMRESTORE **KEYWORD**=value 或 **KEYWORD**=(value1,value2,...,valueN)

确认 DM 数据库服务器已停止后，在 Windows “命令提示符”窗口中输入带参数的 DMRESTORE 命令启动该工具，参数说明见表 32.3。

表 32.3 DMRESTORE 工具参数说明

参数	含义	备注
INI_PATH	配置文件路径，最大长度为 256 字节	可选参数
FILE	用于恢复的备份文件完整路径	必选参数
TIME	指定恢复的时间点，默认为恢复到日志结束	可选参数

BACKUP_DIR	备份文件存放路径，用于搜索备份文件	可选参数
ARCHIVE_DIR	归档日志存放的目录，可以指定 1~8 个	可选参数
DATAFIL_PATH	指定恢复后的新文件路径(编号，路径，编号，路径)，编号从 1 开始	可选参数
PARALLEL_DIR	并行映射文件存放路径，用于搜索映射文件	可选参数
SHOW_BAK_INFO	0 表示利用备份还原，1 仅显示备份信息，不进行还原，默认值为 0	可选参数
HELP	打印帮助信息	可选参数

备注：若设置 SHOW_BAK_INFO 参数为 1，则不需要利用备份进行还原操作，可以不指定 INI_PATH。

例如，利用备份文件 c:\dmdata\backup\test.bak 还原数据库

```
DMRESTORE INI_PATH=c:\dmdata\dameng\dm.ini FILE=c:\dmdata\backup\test.bak
```

32.3.3 还原表空间

DM 提供 SQL 命令对表空间进行还原。与数据库还原类似，表空间备份分为完全备份和增量备份，但表空间备份没有联机备份和脱机备份之分，表空间备份只支持联机备份。表空间还原根据不同的表空间备份类型选择不同的还原策略对表空间进行还原。由于表空间还原依赖于归档日志，因此利用备份文件还原表空间只能还原到备份时所在数据库中。由于待还原表空间的文件名和文件数目可能与备份时的不一致，因此还原后需要更新文件系统保证文件系统是最新状态。另外，由于从备份表空间到还原表空间的时间间隔中，系统可能累积了很多归档日志，为了保证跨表空间的对象数据同步，需要重做该表空间的所有归档日志，保证数据处于最新状态。

系统提供联机 SQL 命令执行表空间还原，还原前需要将表空间置为脱机状态，不支持对系统表空间、日志表空间、回滚表空间、临时表空间执行还原操作。

```
RESTORE TABLESPACE <表空间名> FROM '<备份路径>' | BACKUP <备份文件名> [ IDENTIFIED BY <密码>] [ WITH BACKUPDIR '<备份目录>' {,<备份目录>}] [WITH ARCHIVEDIR '<归档目录>' {,<归档目录>}] [DBFILE <文件 ID> TO '<文件路径>' {, DBFILE <文件 ID> TO '<文件路径>'}];
```

参数说明：

- (1) <表空间名> 需要还原的表空间名称；
- (2) <备份路径> 备份文件存放的完整路径；
- (3) <备份文件名> 备份的名称，在 DMDBMS 中以此标识不同的备份；
- (4) <密码> 加密备份表空间时，用户设置的密码；
- (5) <备份目录> 搜集备份文件的目录；
- (6) <归档目录> 还原时，搜集归档文件的目录；
- (7) <文件 ID> 需要重新设置文件路径文件对应的文件号；
- (8) <文件路径> 设置还原后，数据文件的路径。

例如，利用备份文件 C:\ts.bak 还原表空间 t1。

```
ALTER TABLESPACE t1 OFFLINE;
RESTORE TABLESPACE t1 FROM 'C:\tablespace_bak.bak' IDENTIFIED BY ABCDEF;
ALTER TABLESPACE t1 ONLINE;
```

对于表空间联机还原，还原前需要将表空间置为脱机状态；还原后，为了使用表空间，需要重新将表空间置为联机状态。

32.3.4 还原用户表

与还原数据库和还原表空间不同，表备份文件没有完全备份和增量备份之分，这是由它的底层实现决定的。数据库备份和表空间备份都属于物理备份范畴，而表备份属于 B 树备份

范畴。还原用户表不是简单拷贝备份文件的数据，而是根据备份文件的数据重建表。重建表的本质就是重建索引的过程。因此，还原用户表，会首先还原该表的聚簇索引，然后依次还原二级索引。若待还原的表不存在，则系统会自动重建该表，并将数据恢复到备份时的状态。系统提供联机 SQL 命令执行还原用户表，不支持临时表和垂直分区子表的还原。

```
RESTORE TABLE FROM '<文件路径>' | BACKUP <备份名> [IDENTIFIED BY <密钥>]
```

参数含义参见 32.3.3 还原表空间这一节。

例如，利用备份文件 C:\table.bak 还原表。

```
RESTORE TABLE FROM 'C:\table_bak.bak' IDENTIFIED BY ABCDEF;
```

32.3.5 逻辑还原

DM 提供了对数据库进行逻辑还原的命令行工具 `dimp.exe`，位于安装目录的 `bin\` 目录下。系统管理员可以利用它，在命令行方式下对 DM 数据库进行联机逻辑还原，并支持对远程数据库的访问。

`dimp` 工具利用 `dexp` 工具生成的备份文件对数据库进行联机逻辑还原。还原的方式可以灵活选择，如是否忽略对象存在而导致的创建错误、是否导入约束、是否导入索引、导入时是否需要编译、是否生成日志等。

`dimp` 的参数 如表 32.4 所示。

表 32.4 `dimp` 的参数

参数	含义	备注
USERID	用户名/口令@主机名:端口号#证书路径 如: SYSDBA/SYSDBA@server:5236#ssl_path@ssl_pwd	必选参数，其中主机名、端口号和证书路径为可选项，如果不指定，使用缺省值
FILE	输入文件，之前导出的文件	必选参数
DIRECTORY	导入文件所在目录	可选参数
FULL	导入整个数据库 (N)	可选参数
OWNER	导入指定的用户名下的模式	可选参数
SCHEMAS	导入的模式列表	可选参数
TABLES	表名列表，指定导入的 tables 名称	可选参数
PARALLEL	用于指定导入的过程中所使用的线程数目	可选参数
IGNORE	忽略创建错误 (N)	可选参数
EXCLUDE	忽略指定的对象	如 EXCLUDE=(CONSTRAINTS, TABLES)
CONSTRAINTS	导入约束 (Y)	可选参数
GRANTS	导入权限 (N)	可选参数
INDEXES	导入索引 (Y)	可选参数
TRIGGERS	导入触发器 (Y)	可选参数
ROWS	导入数据行 (Y)	可选参数
LOG	屏幕输出的日志文件	可选参数
PARFILE	参数文件名,如果你 exp 的参数很多,可以存成参数文件	可选参数
FEEDBACK	显示每 x 行 (0) 的进度	可选参数
COMPILE	编译过程, 程序包和函数 (Y)	可选参数
INDEXFILE	将表/索引信息写入指定的文件	可选参数
REMAP_SCHEMA	SOURCE_SCHEMA:TARGET_SCHEMA 将	可选参数

	SOURCE_SCHEMA 中的数据导入到 TAR GET_SCHEMA 中 (N)	
COMPRESS	导出数据文件是否压缩(N Y)	可选参数，默认为 N
SHOW	只列出文件内容 (N)	可选参数
HELP	显示帮助信息	可选参数

以下为简单的命令举例。

1. 不同方式的导入

1) FULL 方式的导入

```
dimp USERID=SYSDBA/SYSDBA FILE=./exp/db_str.dmp LOG=db_str.log FULL=Y
```

可以仅导入其中的某些用户、模式、或表，命令如下：

```
仅导入某个用户：dimp USERID=SYSDBA/SYSDBA FILE=./exp/db_str.dmp LOG=db_str.log  
OWNER=USER01
```

```
仅导入某个模式：dimp USERID=SYSDBA/SYSDBA FILE=./exp/db_str.dmp LOG=db_str.log  
SCHEMAS=USER01
```

```
仅导入某些表：dimp USERID=SYSDBA/SYSDBA FILE=./exp/db_str.dmp LOG=db_str.log  
TABLES=user1.table1, user1.table2
```

注意：在导入的过程中，当导出方式不为 FULL 时，导入时也可以指定为 FULL 方式，此时的 FULL 会将导出文件包含的全部内容导入到指定的实例中。

2) OWNER 方式的导入

将用户 user001 拥有的所有对象全部导入：

```
dimp SYSDBA/SYSDBA FILE=./exp/db_str.dmp LOG=db_str.log OWNER=USER01
```

可以仅导入其中的模式或表，导入表时应该带相应的模式名称，如：

```
导入某个模式：dimp SYSDBA/SYSDBA FILE=db_str.dmp LOG=db_str.log SCHEMAS=USER01
```

```
导入某个表：dimp SYSDBA/SYSDBA FILE=./exp/db_str.dmp LOG=db_str.log TABLES=user1.table1
```

3) SCHEMAS 方式的导入

可以将实例中的某个模式进行导入操作，如：

```
dimp SYSDBA/SYSDBA# c:\data FILE=./exp/db_str.dmp LOG=db_str.log SCHEMAS=USER01
```

4) TABLES 方式的导入

```
dimp SYSDBA/SYSDBA FILE=./exp/db_str.dmp LOG=db_str.log TABLES= test1, test2 ROWS=Y
```

将 SYSDBA 登录的默认数据库上的表 test1, test2 上的所有数据行、约束、索引等信息全部导入到 SYSDBA 用户下。

2. IGNORE、EXCLUDE、FEEDBACK 参数的使用

IGNORE：指明当导入的表存在时，如果其值为 FALSE，则不导入数据，否则导入数据；

EXCLUDE：指明导入时忽略的对象种类，如以下示例指明在导入过程中不导入约束；

FEEDBACK：指明在导入数据的过程中每多少行打印提示信息。

示例如下：

将当前用户下的 test1、test2 的对象信息导入，但不导入约束：

```
dimp SYSDBA/SYSDBA FILE=c:\db_str.dmp LOG=db_str.log TABLES=test1, test2  
EXCLUDE=(CONSTRAINTS)
```

将导出文件中的表 test1, test2 上的相关数据以及对象信息导入到数据库中，在导入的过程中忽略表存在的警告信息，并且在导入表的过程中每 10000 行打印出进度信息：

```
SYSDBA/SYSDBA@192.168.0.95 FULL=Y SHOW=Y FILE=a.dmp IGNORE=Y DIRECTORY=C:\test\  
FEEDBACK=10000
```

3. 参数文件的使用

可以将常用的命令保存到文件中，然后使用参数文件进行导出、导入操作，如下面例子所示。


```
dimp USERID=SYSDBA/SYSDBA parfile=c:\para.txt
FILE=db_str.dmp
LOG=db_str.log
TABLES=test1
ROWS=y
EXCLUDE=(INDEXES,CONSTRAINTS)
```

32.3.6 使用归档日志还原

利用备份文件，DM 可以将数据库、表空间和表还原到备份时的状态。但用户可能更希望将数据恢复到最新的状态。这点不用担心，事实上系统足够智能，在利用备份完成还原后，会去归档目录搜索归档日志，并分析归档日志是否与还原的数据库匹配，若存在匹配的归档日志，则系统会重做这部分日志，将数据库还原到最新的状态；若没有可用的归档日志，则只能还原到备份时的状态。

32.3.7 还原到指定时间点

上节叙述了达梦系统使用归档日志将数据库还原到最新的状态，这一节讨论与归档日志相关的另一个功能：还原到指定的时间点。根据用户需求，可以将数据库还原到指定的时间点。还原到时间点的功能依赖于归档日志，在备份完成时，系统会记录一个备份时间，因此要还原的时间点一定在备份完成时间之后，否则系统会提示报错信息。借助备份文件完成还原后，开始重做归档日志，区别于完全还原将所有可用的归档日志全部重做，若指定还原到时间点，则只重做早于时间点的日志，从而达到将数据库还原到指定时间的状态的目的。需要注意的是，由于表空间只是数据库的一部分，为保证还原后，数据库中的所有数据处于最新状态，还原表空间会重做该表空间所有可用的归档日志，因此还原表空间不支持还原到时间点功能。比如用户在 2011-9-8 17:20:20 时对数据库 test 做了一个备份，在 2011-10-8 13:34:20 想将数据库 test 还原到时间点 2011-9-25 15:20:20，若用户保证从备份完成的时间点到指定时间点这段时间的归档日志都完好，则通过如下还原命令就能达到预期效果。

还原数据库 test 到指定时间 2011-9-25 15:20:20

```
DMRESTORE INI_PATH=c:\dmdata\dameng\dm.ini FILE=c:\dmdata\backup\test.bak
TIME="2011-9-25 15:20:20"
```

32.3.8 并行还原

对于并行备份产生的备份文件，系统支持利用还原工具对数据库进行并行还原。与并行备份一样，并行还原需要用户指定并行映射文件的存储位置。由于并行备份可能是增量备份，而备份的基础备份也有可能是并行备份，因此执行并行还原需要指定一个映射文件存放的目录，用于还原的并行备份的映射文件都存放在该目录下。并行还原能提高还原性能，使得数据服务迅速恢复。需要说明的是，备份时的映射文件内容应与还原时映射文件内容保持一致，但这不是必需的，系统在还原前，会对用户设置的映射文件内容进行校验，判断映射文件内容是否与备份时映射文件内容兼容，保证系统能正确还原数据库。

利用并行备份文件 c:\dmdata\backup\test.bak 还原数据库，并行映射文件为 E:\bak.parallel。

```
DMRESTORE INI_PATH=c:\dmdata\dameng\dm.ini FILE=c:\dmdata\backup\test.bak
PARALLEL_DIR=E:\
```

第 33 章 数据守护

33.1 概述

数据守护（DATA WATCH）可以满足用户对数据可靠性、安全性的需求，解决由于硬件故障、自然灾害等原因导致的数据库服务长时间中断问题，提高系统的可用性。数据守护方案是一个集成化的灾难恢复解决方案，为 DM 可靠性解决方案之一，可有效避免主机单点故障，同时备机可作为只读服务器对外提供服务，实现负载均衡。

达梦数据守护方案 DATA WATCH 有两种实现方式：一种是基于观察器的数据守护（OBSV）；另外一种是基于守护进程的数据守护（DMWATCH）。这两种方案都是将主机上的 REDO 日志通过 MAIL 系统发送到备机上，备机将接收到的 REDO 日志重做，实现主备机数据一致，如图 33.1 所示。

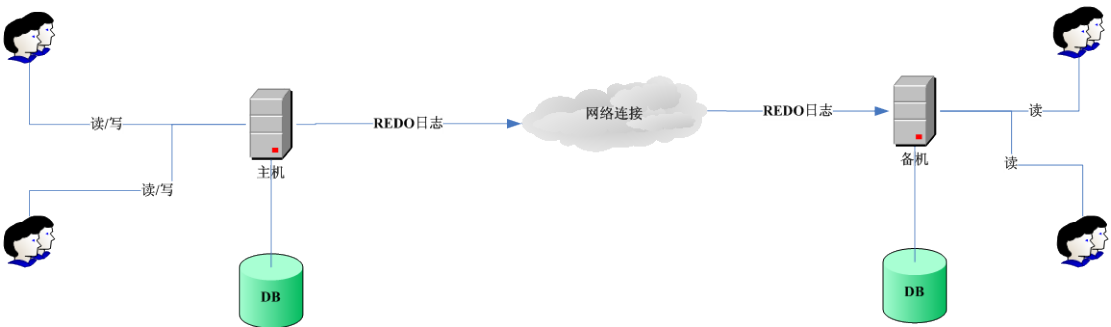


图 33.1 数据守护原理图

当主机发生故障时，备机自动切换为主机，继续提供服务，如图 33.2 所示。

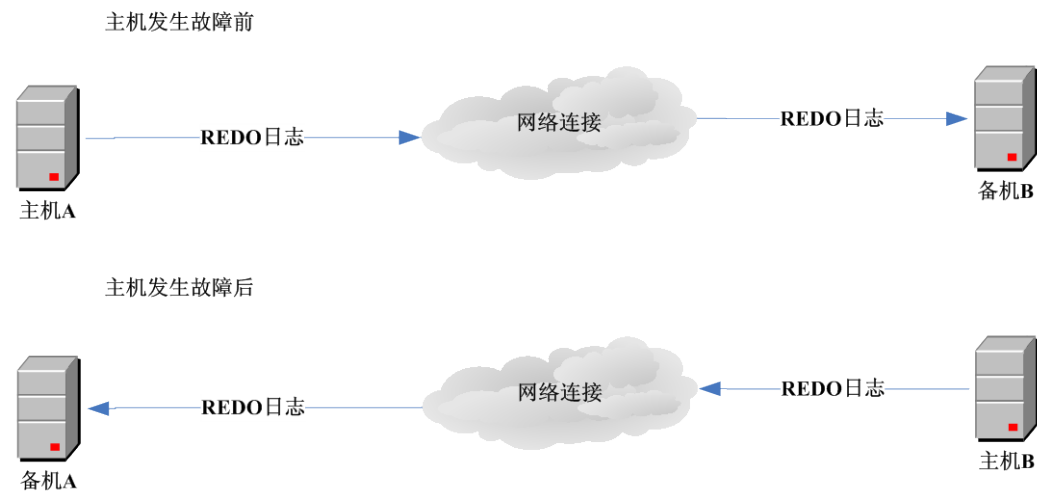


图 33.2 数据守护故障切换图

两种实现方式主要的区别如下：

1. 守护主体。基于观察器的方案守护主体是观察器，观察器负责主备的检测、故障切换和恢复。观察器是独立于主备的第三方，它的实质也是一个 DM 服务器。基于守护进程方案的守护主体是守护进程工具，它和被守护的 DM 服务器处于同一台机器上，主备机上各自配置一个守护进程，通过守护进程之间的通信处理主备的检测、故障切换和恢复；

2. 守护对象。基于守护进程的方案只能配置一台实时备机。而基于观察器的方案除了可配置一台实时备机，还可以配置一台同步备机，以及最多 8 台异步备机；
3. 守护通讯协议。基于观察器方案中观察器是利用 TCP 协议和守护实例通讯，而基于守护进程方案中守护进程是利用 UDP 协议和实例通讯。

下面分别对这两种实现方式进行详细的介绍。

33.2 基本概念

1. 数据库状态

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、修改数据库模式等操作；
- 2) 打开状态 (OPEN)：不能进行控制文件维护、归档配置、修改数据库模式等操作，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。

三种状态不能任意转换，状态转换规则如图 33.3 所示：

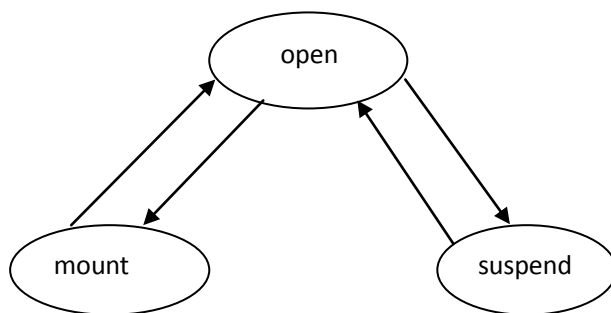


图 33.3 数据库状态转换规则

2. 数据库模式

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) 主机模式 (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。如果归档有效，会将 REDO 日志发送到备机；
- 3) 备机模式 (STANDBY)：用户只能只读访问数据库，实时、同步、异步归档失效，但本地归档仍然有效。

三种模式只有在 MOUNT 状态下设置，相互之间可切换，模式切换规则如图 33.4 所示。

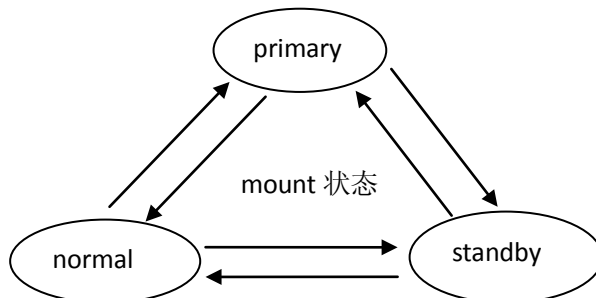


图 33.4 数据库模式切换规则

3. 邮件系统 (MAL 系统)

MAL 系统是一个内部高速通信系统，基于 TCP/IP 协议实现。

33.2.1 基于观察器的数据守护

1. 观察器 (OBSV)

观察器 OBSV 实际上也是一个 DM 服务器，在 OBSV 上创建 DATA WATCH 配置信息表，记录主机、实时备机、同步备机、异步备机、监控间隔等信息；维护、管理 OBSV 全局唯一标识码 OGUID；OBSV 使用 MAL 系统与 DATA WATCH 系统中的其他实例通信。OBSV 通过两个线程（实时检测线程和同步检测线程）来处理故障检测和故障恢复。实时检测线程负责主机、实时备机故障检测、故障恢复。同步检测线程负责同步备机、异步备机故障检测、故障恢复。

2. OBSV 组

OBSV 组明确 OBSV 需要监控哪些实例（实时备机、同步备机、异步备机）。一个实例只能属于一个 OBSV 组，一个 OBSV 组包含多个实例，一个 OBSV 最多可同时监控 8 个组。

3. 故障自动切换 (ASFO)

数据守护系统中，主机故障时，备机可以自动切换为主机，这个过程叫做故障自动切换。

4. OGUID

OGUID 用来标识、校验，相同 OGUID 值的实例、观察器属于同一个 OBSV 组。

5. 主机

数据守护系统中，支持用户所有数据库请求的数据库实例，叫做主机；一个 OBSV 组只能配置一个主机。

6. 实时备机

数据守护系统中，支持故障自动切换的数据库实例叫做实时备机。实时备机与主机之间必须配置实时归档，一个 OBSV 组只能配置一个实时备机。

7. 同步备机

数据守护系统中，不支持故障自动切换且主机的所有操作是同步传输的数据库实例叫做同步备机。主机和实时备机上配置了到同步备机的同步归档，一个 OBSV 组只能配置一个同步备机。

8. 异步备机

数据守护系统中，不支持故障自动切换且主机操作时定时传送的数据库实例，叫做异步备机。主机和实时备机上配置了到异步备机的异步归档，一个 OBSV 组最多可以配置 8 个异步备机。

9. 脑裂 (BRAIN SPLIT)

脑裂指的是在某些情况下（比如网络故障），观察器认为主机故障，将实时备机切换为主机，但实际上主机仍然在正常工作；此时，数据守护中就出现了两个同时处于工作状态的主机，均可以响应用户请求，最终导致数据一致性被破坏。

33.2.2 基于守护进程的数据守护

1. 守护进程 (DMWATCH)

守护进程 DMWATCH 是一个独立的工具，与数据库实例部署在相同站点，负责监控数

数据库实例，在数据库实例异常终止时，负责重新启动数据库实例；并且，可在数据守护方案中，替代观察器的角色，负责故障自动切换。

2. 守护进程组

相同 `OGUID` 的守护进程构成一个守护进程组，一个守护进程组必须且只能包含两个守护进程。守护进程组可以拥有多个监控组（不超过 8 个）。

3. 监控名

守护进程通过监控名，来确定哪两个实例互为主备。

4. 监控组

拥有相同监控名的一对分布在不同节点的数据库实例，构成一个监控组。

5. 守护进程故障处理策略

守护进程有三种故障处理策略：

- 1) 不支持：在此模式下，可以与观察器共存，故障切换由观察器完成，守护进程只负责重启本地数据库实例；
- 2) 基本型：在守护进程之间可相互通信的前提下，数据库实例异常时，提供故障自动切换以及故障恢复功能。适合无法确保网络可靠性的应用场景。不能与观察器同时配置；
- 3) 增强型：守护进程可自主决定是否进行故障切换，数据库实例异常或者出现硬件故障时，提供故障自动切换以及故障恢复功能。适合守护进程之间的网络是绝对可靠的应用场景。不能与观察器同时配置。

基本型与增强型的区别在于网络是否通畅以及是否是硬件故障。

例如实例 `DM1` 发生异常，当前模式是基本型，那么只有当监控发生故障的实例的守护进程 `A` 将发生故障的消息发送到另一个守护进程 `B`，并且 `B` 接收到了消息之后，才会进行故障恢复或者是故障自动切换，否则在网络状况不好的情况下，`B` 没有收到消息，那么就不会进行任何处理；如果当前模式是增强型，也就意味着网络是通畅的，当守护进程 `A` 将 `DM1` 故障的消息广播出去之后，守护进程 `B` 一定会接收到，那么就可以进行故障恢复或者状态切换。

在基本型模式下，如果发生硬件故障问题，守护进程不进行处理；而在增强型模式下，守护进程会将实例发生硬件故障的消息广播出去，通知用户进行处理。

6. 守护进程状态

守护进程有以下几种状态：

- 1) 启动状态（`STARTUP`）：守护进程处于启动状态时，不支持故障切换的模式下，直接转入 `OPEN` 状态。在支持故障切换模式下，需要等待远程守护进程的广播信息，决定其监控实例的启动模式，并进入 `OPEN` 状态；
- 2) 工作状态（`OPEN`）：守护进程处于正常工作状态时，监控数据库实例，并广播当前数据库实例的状态信息，接收其他守护进程广播的状态信息，接收监视器广播的信息；
- 3) 关闭状态（`SHUTDOWN`）：守护进程处于停止监控状态时，不再监控数据库实例的活动状态；
- 4) 故障处理状态（`FAILOVER`），守护进程进行故障切换时，设置为 `FAILOVER` 状态。例如，主机 `A` 出现故障，监控 `A` 的守护进程 `A` 状态，被设置为 `FAILOVER`，备机 `B` 切换到主机，监控 `B` 的守护进程 `B` 状态，被设置为 `FAILOVER`；
- 5) 故障恢复状态（`RECOVERY`）：守护进程在数据库实例进行历史数据同步时和故障恢复处理时，设置为 `RECOVERY` 状态。例如，主机 `A` 故障重启后，守护进程 `A` 和守护进程 `B` 状态会被设置为 `RECOVERY`；
- 6) 手动切换状态（`SWITCHOVER`）：通过监视器手动切换主备机时，守护进程设置状态为 `SWITCHOVER`。

守护进程状态转换规则见图 33.5。

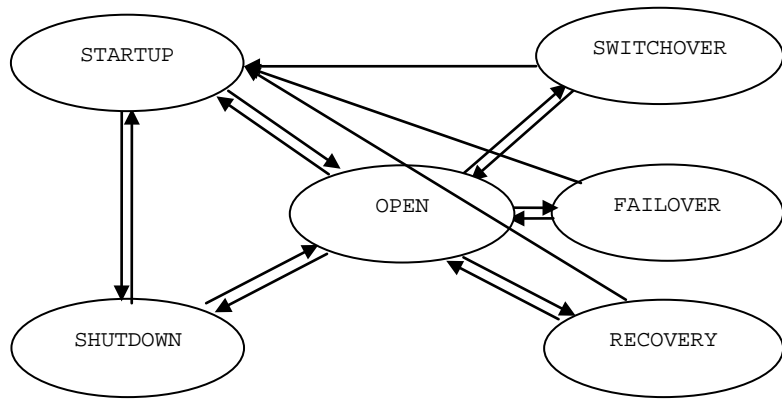


图 33.5 守护进程状态转换规则

7. 监视器 DMWMON

监视器是一个独立的工具，接收并解析守护进程广播的消息，方便用户查看数据库实例的状态、数据同步情况、以及故障切换历史等信息。用户可以通过监视器完成手工切换主备机、停止守护进程对监控组的监控、启动守护进程对监控组的监控等功能。一个监视器只能监控一组监控组。监视器只能部署在主备机之外的第三台机器上，在守护进程方案中，监视器不是必须的部件。

33.3 体系结构

33.3.1 基于观察器的数据守护

基于观察器的数据守护是一个集成化的解决方案，由以下部件构成：

1. 一个主机（PRIMARY）；
2. 一个实时备机（STANDBY）；
3. 一个观察器（OBSV）；
4. 一个同步备机；
5. 异步备机，总数不超过 8 个；
6. 各实例间通信的 MAL 系统。

用户可根据不同的场景需求配置不同的部件，其中 1 和 6 是必须配置的，其他部件可以选择性配置，例如可以只配置实时备机而不配置非实时备机（同步、异步备机）；或者只配置非实时备机而不配置实时备机。

主机对外接收用户请求，对所有修改数据库操作生成 REDO 日志，根据归档设置，通过 MAL 系统将 REDO 日志发送到备机。备机收到 REDO 日志进行重做，实现数据同步。

观察器监控数据守护中的主机和实时备机、同步备机、异步备机，当检测到故障发生时，启动相应的故障处理流程；当检测到故障站点恢复时，启动恢复流程，同步历史数据。

在备机上用户可进行只读查询操作，可进行实时数据、历史数据访问、报表生成、历史数据备份等，缓解了主机的访问压力，实现负载均衡。

基于观察器的数据守护的系统结构如图 33.6 所示。

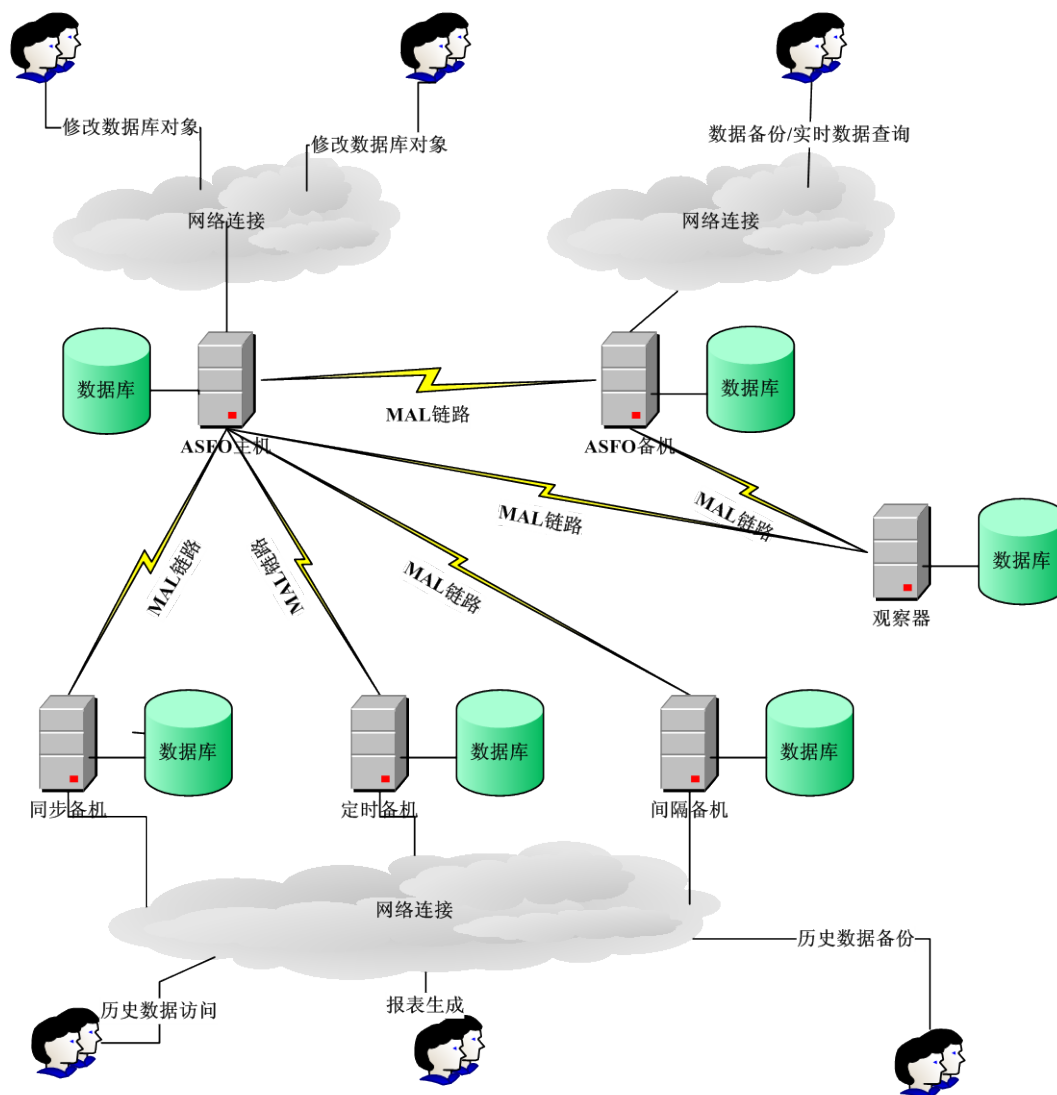


图 33.6 基于观察器的数据守护的系统结构

33.3.2 基于守护进程的数据守护

基于守护进程的数据守护方案由守护进程和监视器两个模块共同完成数据守护的功能，主要由以下几个部件构成：

1. 一个监视器 dmwmon;
2. 两个站点 site1, site2;
3. 站点 site1, site2 上面分别有各自的守护进程 dmwatch1, dmwatch2, 这两个守护进程属于同一个守护进程组, 这个守护进程组监控监控组 group_a;
4. 监控组 group_a 包含两个数据库实例: inst1 (主机), inst2 (备机)。这两个实例分别位于站点 site1, site2 上面。

说明：上面是最基本的配置，守护进程组可以拥有多个监控组，但是不能超过 8 个；监视器和站点必须在同一个网段里面。用户可根据不同的场景需求配置不同的部件。

DM 守护进程和监视器的体系结构，如图 33.7 所示。

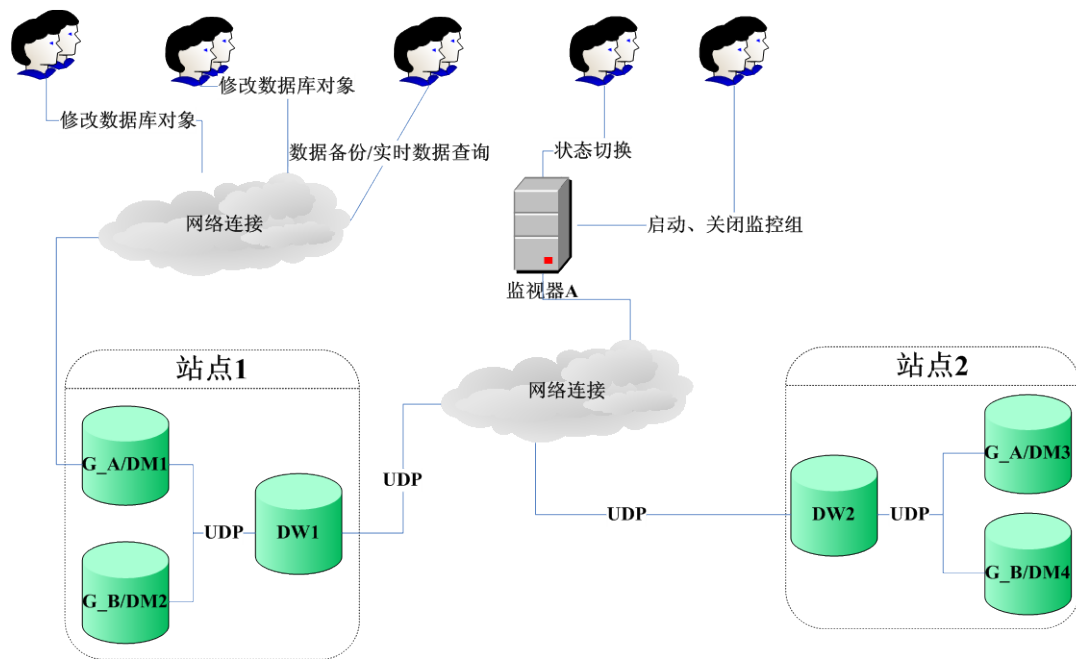


图 33.7 DM 守护进程和监视器的体系结构

说明：

1. 守护进程之间通过 UDP 广播通信；守护进程与监视器之间通过 UDP 广播通信；守护进程和其守护的数据库实例之间通过 UDP 消息通信；
2. 站点 1 的守护进程 DW1 监控实例 DM1，DM2；站点 2 的守护进程 DW2 监控实例 DM3，DM4；
3. 站点 1 中的实例 DM1 和站点 2 的实例 DM3 属于同一个监控组 G_A；站点 1 中的实例 DM2 和站点 2 的实例 DM4 属于同一个监控组 G_B；
4. 监视器 A 负责监控监控组 G_A。

1. 守护进程 DMWATCH 的结构

守护进程每个组包含一个广播线程，一个接收线程。

广播线程每秒钟广播一次实例的状态、守护进程状态等信息。接收线程接收同组的守护进程广播的消息、监视器发送的命令消息，以及数据库实例发送的数据库状态和相应的执行结果消息。

2. 监视器（DMWMON）的结构

监视器（DMWMON）包含一个广播线程，一个接收线程。每个广播线程向对应的监控组广播发送用户的命令。接收线程接收从守护进程发送的广播消息。

3. 数据库实例

数据库实例包含一个 UDP 广播线程和一个 UDP 接收线程。

广播线程每一秒钟发送一次实例的状态等信息到守护进程。接收线程接收守护进程发送的消息。

33.4 基于观察器的数据守护

基于观察器的数据守护方案可以划分为四大部件：主机、备机、观察器以及连接它们的 MAL 系统。主机接收用户请求，对所有修改数据库操作生成 REDO 日志，根据归档设置，在适当时机，通过 MAL 系统将 REDO 日志发送到备机。备机收到 REDO 日志后，解析 REDO 日志，进行重做主机操作，实现数据同步。观察器监控数据守护中的主机和实时备机，当检

测到故障发生时，启动相应的故障处理流程；当检测到故障站点恢复时，启动恢复流程，同步历史数据，如图 33.8 所示。

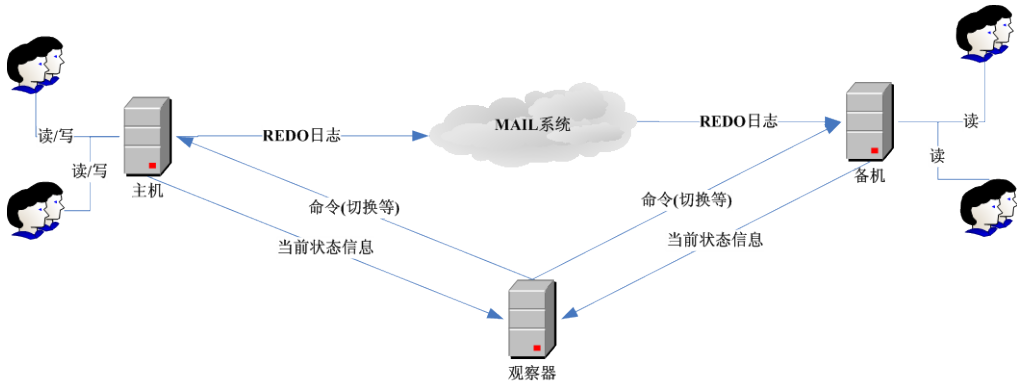


图 33.8 基于观察器的数据守护原理图

主机与单机系统具有相同的功能，实时备机只能进行只读操作。用户只能连接主机进行数据修改操作（INSERT/DELTE/UPDATE/DDl 等操作），可以连接备机进行查询等只读操作；主机根据配置，选择合适时机发送 REDO 日志到实时备机、同步备机、异步备机；备机获取 REDO 日志并重做，实现与主机的数据同步。

主机和实时备机必须至少配置一个本地归档，确保数据不会丢失。为避免大量归档日志文件占用过多磁盘空间，需要完善本地归档设置，使用容量限制，可选择转移最老归档文件到指定路径，或者删除最老归档文件，释放磁盘空间。

数据守护系统启动后，用户应该避免修改守护系统中主、备机的模式和状态，观察器会根据修改后的模式、状态进行故障处理，有可能导致守护系统的故障自动切换功能失效。如果 DBA 认为确实必要的情况下，应该先关闭观察器，再切换主、备机状态，完成修改后，再重启观察器。

数据守护系统中，各个数据库实例之间的网络应尽可能的稳定、可靠，避免由于网络不稳定导致频繁启动故障切换、恢复流程。并且，最好是配置高速网络，减少实例间数据传送时延，提高系统响应速度。

基于观察器的数据守护方案支持多备机。备机可分为实时备机、同步备机、异步备机三类。

1. 实时备机

用户登录备机，可以执行查询、数据备份等只读操作，但不允许修改任何数据库对象。可以正常生成本地归档，实时归档、同步归档、异步归档强制失效，当观察器将其切换成主机时，会使其归档配置有效。

主机任何修改产生的 REDO 日志，将实时地传递到实时备机，只有确保实时备机收到 REDO 日志后，主机才能将 REDO 日志写入本地日志文件，以及进行其他归档处理，主、备机之间的数据是实时同步的。

主机发生故障时，观察器将实时备机自动切换成主机，因此在实时备机上，需要配置到主机以及所有其他备机的归档，确保故障切换后，REDO 日志传递和数据同步。

最多只能配置一个实时备机，用于数据实时同步以及故障切换。

2. 同步备机

用户可以执行查询、数据备份等只读操作；但不允许修改任何数据库对象。可以正常生成本地归档，实时归档、同步归档、异步归档强制失效。

主机上产生的 REDO 日志，通过归档线程发送给同步备机，归档线程会等待备机的响应，主、备机之间的数据存在非常短暂的不一致。当不需要实时同步数据时，可将备机配置成同步归档模式。为了性能上的考虑，仅支持最多 1 个同步备机。

3. 异步备机

可以执行查询、数据备份等只读操作，但不允许修改任何数据库对象。可以正常生成本地归档，实时归档、同步归档、异步归档强制失效。

在设定的时间点或者时间间隔，主机自动将本地归档日志发送到异步备机。

最多支持 8 个异步备机。

实时备机和同步备机、异步备机，支持只读查询，可承担报表生成、数据备份等任务，减轻主机的负载。

备机中的事务满足以下的可见性原则：

- 1) 主机已提交事务的所有修改，对所有备机用户可见；
 - 2) 主机未提交事务的所有修改，对所有备机用户不可见。
- 用户和主备机的交互都是通过观察器来进行的。

33.4.1 配置数据守护

1. 准备工作

虽然一台机器可以同时启动多个 DM 数据库实例，但在配置数据守护系统时，为了确保系统的可靠性，原则上每个实例部署在不同的机器上，服务器版本须一致。

下面在 windows 系统下举例实现一个基本的配置（不配置同步备机、异步备机），约定如表 33.1。

表 33.1 基本配置规划举例

服务器	实例名	IP 地址	服务器端口号	MAL 端口号	文件目录
OBSV 观察器	inst1	192.168.0.11	5236	5241	C:\data\DAMENG
主机	inst2	192.168.0.12	5236	5242	C:\data\DAMENG
实时备机	inst3	192.168.0.13	5236	5243	C:\data\DAMENG

注意：脱机恢复的库必须以 MOUNT 方式启动，否则启动后 LSN 会增加，比主机的 LSN 大，导致主备环境搭建失败。

需要使主备机的数据库一致，可通过备份主机数据库，拷贝备份文件到备机上进行恢复。如果主备的数据文件目录相同（如本示例），也可在数据库服务器关闭的情况下，直接拷贝主机的数据库文件到备机相同目录内。

观察器数据库没有要求，可直接用初始化库工具 dminit 新建一个库。

2. 参数设置

1) dm.ini 配置

分别修改 dm.ini 中对应项，如下表：

表 33.2 各实例 dm.ini 关键项配置

服务器	dm.ini 设置
OBSV 观察器	INSTANCE_NAME = inst1 PORT_NUM = 5236 MAIL_INI = 1 ARCH_INI = 0
主机	INSTANCE_NAME = inst2 PORT_NUM = 5236 MAIL_INI = 1 ARCH_INI = 1
实时备机	INSTANCE_NAME = inst3 PORT_NUM = 5236

	MAIL_INI	= 1
	ARCH_INI	= 1

2) dmmal.ini 配置

dmmal.ini 配置内容如下，用户实际配置时需根据情况更改对应项。

```
[mal_inst1]
    mal_inst_name  = inst1
    mal_host       = 192.168.0.11
    mal_port       = 5241
[mal_inst2]
    mal_inst_name  = inst2
    mal_host       = 192.168.0.12
    mal_port       = 5242
[mal_inst3]
    mal_inst_name  = inst3
    mal_host       = 192.168.0.13
    mal_port       = 5243
```

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) dmarch.ini 配置

分别在主机、实时备机上进行归档配置，观察器以及同步备机、异步备机不需要归档配置。此处只需要做本地归档配置，最多能配置 8 个本地归档。本例只配置一个本地归档，内容如下：

```
[ARCHIVE_LOCAL1]
    ARCH_TYPE      = LOCAL
    ARCH_DEST      = c:\data\arch
    ARCH_FILE_SIZE  = 128      #归档文件大小限制 Mb 为单位，缺省 128M
    ARCH_SPACE_LIMIT = 0      #空间限制 MB 为单位，范围 1024 ~ 4294967294，默认 0 表
```

示无空间限制

3. 观察器 OBSV 设置

参数设置好后，开始配置观察器。

1) 数据库的启动

可按任意顺序分别启动三个数据库实例，除观察器实例外的其他实例都需要以 MOUNT 方式启动。MOUNT 方式启动的方法是在服务器参数的最后添加 MOUNT。

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

2) 系统表的创建

在观察器 OBSV 上执行一段脚本创建系统表，这些表用来记录 OBSV 组、观察器实例信息、历史信息等，提供实时查询功能，脚本内容如下：

```
CREATE SCHEMA DW;
CREATE TABLE DW.SYSDWGROUPS
(
    GROUP_NAME  VARCHAR(128),
    GROUP_DESC  VARCHAR(128),
    NASFO_INTERVAL  INT,
    ASFO_INTERVAL  INT,
    OGUID        INT,
    CFG_APPLIED   INT,
    CLUSTER PRIMARY KEY(GROUP_NAME)
```

```
);
CREATE TABLE DW.SYSDWINSTANCES
(
    INSTANCE_NAME    VARCHAR(128),
    GROUP_NAME        VARCHAR(128),
    SYS_MODE          VARCHAR(128),
    ARCH_TYPE         VARCHAR(128),
    ARCH_TIMER        VARCHAR(128),
    STATUS            VARCHAR(32),
    FLSN              BIGINT,
    CLSN              BIGINT,
    SLSN              BIGINT,
    CLUSTER PRIMARY KEY(INSTANCE_NAME)
);
CREATE TABLE DW.SYSDWHISTORY
(
    TIME$            DATETIME,
    GROUP_NAME        VARCHAR(128),
    INFO              VARCHAR(1024)
);
```

3) OBSV 的配置

在观察器 OBSV 上执行以下系统过程：

```
--增加一个观察器组
SP_DW_ADD_GROUP('test','for test');
--增加主机
SP_DW_ADD_PRIMARY('test', 'inst2');
--增加实时备机
SP_DW_ADD_STANDBY('test', 'inst3','REALTIME', '');
--设置同步备机、异步备机检测间隔，单位秒 s。取值范围 2~60s，默认值 10 秒
SP_DW_SET_NASFO_INTERVAL('test', 30);
--设置实时备机检测间隔，单位 s。取值范围 2~60s，默认值 10 秒
SP_DW_SET_ASFO_INTERVAL('test',30);
--应用设置到主备机
SP_DW_APPLY_CONFIG('test');
```

至此，DATA WATCH 观察器配置完成。

注意：所有实例的 dmmal.ini 配置必须一致。

33.4.2 启动数据守护

按照上一节的步骤，用户就可以配置一个基于观察器的数据守护系统。配置成功后，可以通过下面语句，启动刚配置的数据守护系统。

```
--启动 DATA WATCH 系统
SP_DW_STARTUP('test');
```

33.4.3 查看系统状态

在数据守护系统运行的过程中，要想查看整个系统的状态等相关信息，可以查询相关表。OGUID 值、OBSV 组、实例等信息及其之间的相互关系，可从 DW.SYSDWGROUPS / DW.SYSDWINSTANCES 系统表查询获得。系统启动、切换、恢复等历史信息可从 DW.SYSDWHISTORY 系统表中查询。

33.4.4 故障自动切换

基于观察器的数据守护方案支持发生故障时主备机自动切换。故障自动切换的前提条件是，必须配置主机和实时备机以及观察器 OBSV。

基于观察器的数据守护方案的故障自动切换功能可以达到以下的效果：

1. 主机故障时，实时备机自动切换为主机，对外提供数据库服务；
2. 实时备机故障时，主机可正常提供数据库服务；
3. ASFO 故障实例恢复后，自动同步历史数据，并作为实时备机加入系统；
4. 同步备机故障后也可自动恢复。

主备机自动切换完成后，客户端连接会根据 dm_svc.conf 配置的 SERVERNAME 配置项自动切换连接到新的主机，可参考本手册 2.1.1.2 节，前提是客户端连接时使用了 SERVERNAME。

33.4.5 手动切换主备机

在系统运行的过程中，用户可以切换主备机，即将主机变为备机，备机变为主机。系统可以进行手动切换的前提条件是，必须配置主机和实时备机以及观察器 OBSV，且主备机都在正常工作状态下，可主动交换主机和实时备机的角色。原主机作为实时备机，原实时备机作为主机。

用户可以运行下面语句，进行手动切换主备机：

```
SP_DW_SWITCHOVER('test');
```

33.4.6 增加/删除备机

基于观察器的数据守护方案支持动态增加/删除备机。其中，动态增加备机的步骤如下：

1. 在主机、或同步备机、异步备机，执行联机备份；
2. 拷贝备份文件到新增备机；
3. 在新增备机上，执行脱机恢复；
4. 以 MOUNT 方式启动新加备机实例；
5. 登录观察器，调用如下函数：

```
SP_DW_ADD_STANDBY(GROUP_NAME, INSTANCE_NAME, TYPE, TIME_INTERVAL);
```

例如，使用下面的语句为 test 组增加一个实例名为 inst4 的同步备机。

```
SP_DW_ADD_STANDBY_DYNAMIC('test','inst4','SYNC',);
```

动态删除备机则比较简单，只需登录 OBSV，调用如下语句：

```
SP_DW_DROP_STANDBY_DYNAMIC('test','inst3');
```

33.4.7 停止观察器检测

通过下面语句，可停止观察器的检测，其他实例还在运行，但是不会进行切换和恢复了。

```
--停止观察器的检测  
SP_DW_SHUTDOWN('test');
```

33.4.8 特性

1. 接口自动重定位

通过客户端设置配置文件，一旦主机故障后，客户端登录到主机的未提交事务失败回滚，并重新定位到新的主机，返回一个特殊的错误码，这个过程对用户是透明的。

2. 指定备机新增文件策略

主机上新建文件后，备机重做日志时也会新建文件，可通过指定 `dm.ini` 参数 `NEW_FILE_POLICY` 来确定新增的文件存放路径策略，值为 0 时使用相对路径，值为 1 使用绝对路径，默认为 0。

3. 跨平台

当机器的 CPU 字节序一致且操作系统位数（寻址宽度）一致时，可跨平台搭建数据守护系统。例如观察器部署在 WINDOWS 服务器上，而将主机、备机部署在 LINUX 服务器上。

4. 滚动升级

在 DM 服务器版本差别不大时，支持滚动升级，即在不影响应用的前提下，依次升级每一台服务器版本。

33.4.9 故障处理

硬件故障加入：

如果某个实例上发生硬件故障（如硬盘损坏）导致数据库文件损坏，数据库无法重新启动。当硬件修复后，可进行硬件故障加入，步骤如下：

1. 在主机、或同步备机、异步备机上执行联机备份；
2. 拷贝备份文件到故障站点上；
3. 在故障站点上，执行脱机恢复；
4. 以 MOUNT 方式启动数据库。

OBSV 检测线程会自动检测到实例已恢复，自动完成后续数据同步，加入守护系统。

33.5 基于守护进程的数据守护

上一节里面，详细讲述了达梦数据守护方案 DATA WATCH 通过观察器 OBSV 来提供数据可靠性、数据保护、灾难恢复等功能。但在某些特定应用场景下，用户不希望存在观察器，而是要求主机和备机之间相互协调，进行故障自动切换和恢复，满足用户对数据可靠性、安全性的需求，解决由于数据库实例异常、硬件故障等原因导致的数据库服务长时间中断问题，提高系统的可用性。因此，达梦数据库管理系统特意设计了守护进程 DMWATCH 和监视器 DMWMON 两个工具，由这两个工具共同完成主机和备机之间相互协调，进行故障自动切换和恢复的功能，代替观察器。

主机接收用户请求，对所有修改数据库操作，生成 REDO 日志，根据归档设置，在适当时机，通过 MAL 系统将 REDO 日志发送到备机。备机收到 REDO 日志后，解析 REDO 日志，进行重做主机操作，实现数据同步。数据库实例在响应用户的同时，不断的向守护进程广播自己的状态。守护进程收到实例的消息后，将消息广播出去。如果守护进程在一段时间内没有收到数据库实例的消息，那么将重新启动实例。当主机上的数据库实例发生故障时，主备机的守护进程进行交互，协调，并进行自动切换（增强型/基本型）。

监视器接收来自主备机上面的守护进程的广播消息，并将用户的相关命令发送到主备机上的守护进程上面，由守护进程向数据库实例发送相关操作的命令，如图 33.9 所示。

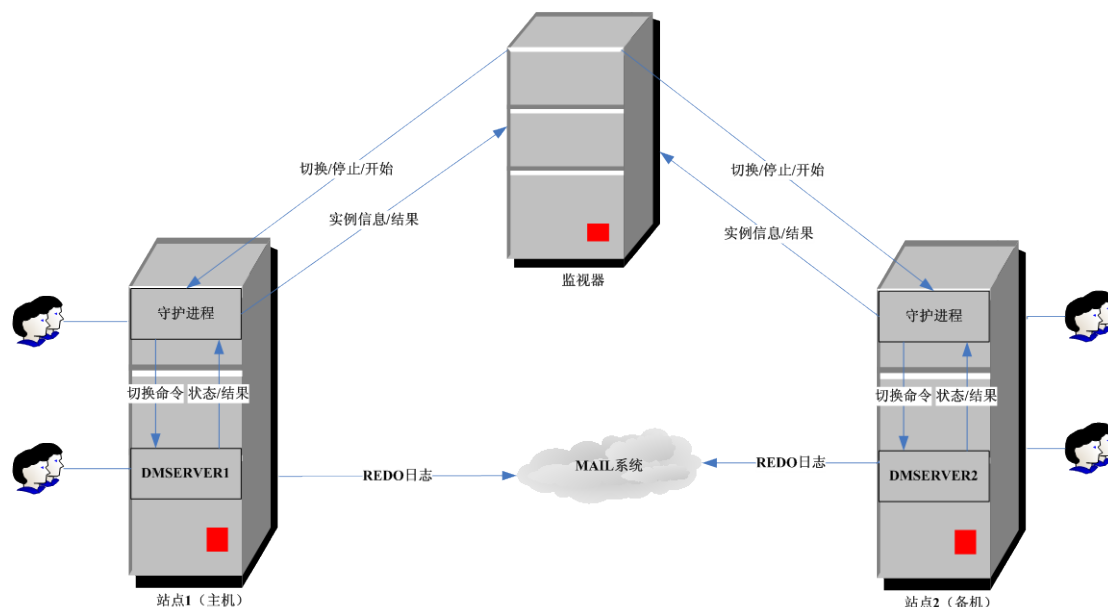


图 33.9 基于守护进程的数据守护原理图

33.5.1 配置数据守护

这里用一个 windows 下的例子来说明守护进程和监视器的配置：

假设有两个站点：site1 (192.168.0.235) 和 site2 (192.168.0.245)，site1 拥有实例 DM1，site2 拥有实例 DM2，其中 DM1 是主机，DM2 是备机。

具体的配置流程：

1. 关闭主备机上的守护进程；
2. 同步主备机数据；
3. 配置 dm.ini；
4. 配置 dmmal.ini；
5. 配置 dmarch.ini；
6. 配置 dmwatch.ini；
7. 配置 dmwmon.ini；
8. 启动主机实例到 MOUNT 状态，建立主备的 MAL 链路；
9. 通过系统过程设置主备机的 OGUID；
10. 分别修改主机和备机的模式到指定模式（PRIMARY 或者 STANDBY）；
11. 启动主备机的守护进程；
12. 启动监视器。

下面进行详细的描述。

1. 关闭主备机上的守护进程

手动杀死 site1 和 site2 上面的守护进程 DMWATCH（如果存在）。

2. 同步主备机数据

需要使主备机的数据库一致，可通过备份主机数据库，拷贝备份文件到备机上进行恢复。如果主备的数据文件目录相同（如本示例），也可在数据库服务器关闭的情况下，直接拷贝所有文件到备机数据文件目录内。

注意：脱机恢复的库必须以 MOUNT 方式启动，否则启动后 LSN 会增加，比主机的 LSN 大，导致主备环境搭建失败。

观察器的数据库没有要求，可直接用初始化库工具新建一个库。

3. 配置 dm.ini

site1 中的 DM1 的 dm.ini 的一些参数修改如下：

```
INSTANCE_NAME = DM1
DW_UDP_PORT   = 5243
INST_UDP_PORT  = 5253
MAIL_INI      = 1
ARCH_INI      = 1
```

site2 中的 DM2 的 dm.ini 的一些参数修改如下：

```
INSTANCE_NAME = DM2
DW_UDP_PORT   = 5243
INST_UDP_PORT  = 5253
MAIL_INI      = 1
ARCH_INI      = 1
```

其中 DW_UDP_PORT 是守护进程的广播端口号；INST_UDP_PORT 是实例接收守护进程消息的端口。MAIL_INI 和 ARCH_INI 设置为 1 表示启用 mail 系统和归档。

4. 配置 dmmal.ini

site1 中的 dmmal.ini 配置如下：

```
[MAL_INST1]
MAL_INST_NAME = DM1
MAL_HOST      = 192.168.0.235
MAL_PORT      = 5237
[MAL_INST2]
MAL_INST_NAME = DM2
MAL_HOST      = 192.168.0.245
MAL_PORT      = 5237
```

site1 和 site2 的 dmmal.ini 文件是完全一致的。

5. 配置 dmarch.ini

配置本地归档和实时归档，DM1 的实时归档目标为 DM2。

DM1 的 dmarch.ini 为：

```
[ARCHIVE_LOCAL1]
ARCH_TYPE      = LOCAL
ARCH_DEST      = D:\DM7DATA\DAMENG
ARCH_FILE_SIZE = 128 #单位 MB，缺省 128M
ARCH_SPACE_LIMIT = 0    #单位 MB，0 表示无限制，范围 1024~4294967294M
[ARCHIVE_REALTIME]
ARCH_TYPE      = REALTIME
ARCH_DEST      = DM2
```

DM2 的 dmarch.ini 为：


```
[ARCHIVE_LOCAL1]
ARCH_TYPE          = LOCAL
ARCH_DEST          = D:\DM7DATA\DAMENG
ARCH_FILE_SIZE     = 128
ARCH_SPACE_LIMIT   = 0
[ARCHIVE_REALTIME]
ARCH_TYPE          = REALTIME
ARCH_DEST          = DM1
```

6. 配置 dmwatch.ini

DM1 的 dmwatch.ini 为：

```
[GROUP_A]
OUGID              = 862419
DW_UDP_PORT        = 5243    #和 DM.INI 中的同名配置项一致
SWITCH_MODE        = 1      #0: 不支持; 1: 基本型; 2: 增强型
INST_NAME          = DM1
INST_UDP_PORT      = 5253    #和 DM.INI 中的同名配置项一致
BRO_ADDR           = 192.168.0.255
START_TYPE         = 1      #0: 服务方式启动; 1: 控制台方式启动
PATH               = D:\SRC-DM7\DEBUG\DMSERVER D:\DM7DATA\DAMENG\DM.INI
SERVICE_IP        = 192.168.0.235
BRO_ERROR_INTERVAL = 500
FAILOVER_TIMES     = 3
ENHANCED_PORT      = 5400    #只在增强模式下配置
ENHANCED_FAR_IP    = 192.168.0.245    #只在增强模式下配置
```

说明：

GROUP_A 是监控组，监控实例 DM1（DM2）。

守护进程向监控组 GROUP_A 发送广播到 5243 端口，守护进程接收 GROUP_A 的消息的端口是 5243。

OUGID 是标识监控组的 ID 号，其监控的实例应该配置相同的 OUGID。

监控进程负责监控 GROUP_A 的实例 DM1。

实例 DM_A 的启动路径在 D:\DM7\DAMENG\，守护进程与数据库实例之间通信的端口是 5253，实例优先级为 1。切换模式是基本型（1）。

PATH 举例说明：

WINDOWS 上 EXE：

PATH = 路径\DMSERVER 路径\DM.INI

WINDOWS 上服务方式启动：

PATH = NET START 服务名

LINUX 上 EXE：

PATH = 路径/DMSERVER 路径/DM.INI，注意：此处的路径中不能带有空格

LINUX 上服务方式启动：

PATH = SERVICE DMSERVERD RESTART

DM2 的 dmwatch.ini 为：

```
[GROUP_A]
INST_NAME          = DM2
INST_UDP_PORT      = 5253
BRO_ADDR           = 192.168.0.255
START_TYPE         = 1      #0: 服务方式启动; 1: 控制台方式启动
```

```

PATH                =D:\SRC-DM7\DEBUG\DMSERVER D:\DM7DATA\DAMENG\DM.INI
SERVICE_IP         = 192.168.0.245
OGUID               = 862419
DW_UDP_PORT         = 5243
SWITCH_MODE         = 1    #0: 不支持; 1: 基本型; 2: 增强型
BRO_ERROR_INTERVAL  = 500
FAILOVER_TIMES       = 3
ENHANCED_PORT        = 5400          #只在增强模式下配置
ENHANCED_FAR_IP      = 192.168.0.245  #只在增强模式下配置

```

7. 配置 dmwmon.ini

监视器的配置 dmwmon.ini 的文件内容为：

```

MON_OGUID           = 862419
MON_UDP_PORT        = 5243    #和 DMWATCH.INI 中的 DW_UDP_PORT 项一致
GROUP_NAME          = GROUP_A
MON_BRO_ADDR        = 192.168.0.255

```

8. 启动主备机实例到 MOUNT 状态

用下面的命令将主、备机实例启动到 mount 状态。

```
dmserver.exe D:\src-dm7\Debug\DAMENG\dm.ini mount
```

9. 通过系统过程设置主备机的 OGUID

用下面的系统过程设置主、备机实例的 OGUID 为 862419。

```
SP_SET_OGUID(862419);
```

如果 dmserver 的 OGUID 没有设置就启动，则后续 DMWATCH 会不断试图重启 dmserver 实例。

10. 修改主备和备机的模式到指定模式

用下面的系统过程设置主机实例 DM1 的模式为 PRIMARY。

```
ALTER DATABASE PRIMARY;
```

用下面的系统过程设置备机实例 DM2 的模式为 STANDBY。

```
ALTER DATABASE STANDBY;
```

11. 启动主备机的守护进程

启动 site1 上面的守护进程。

```
DMWATCH D:\src-dm7\Debug\DAMENG\dmwatch.ini
```

启动 site2 上面的守护进程。

```
DMWATCH D:\src-dm7\Debug\DAMENG\dmwatch.ini
```

12. 启动监视器

启动监视器的命令如下。

```
DMWMON D:\src-dm7\Debug\DAMENG\dmwmon.ini
```

33.5.2 故障自动切换和恢复

在基于守护进程的数据守护系统中，当主机发生故障时，系统通过主备机守护进程之间的交互，实现主备机的切换，并且实现主机的自动故障处理；当备机发生故障时，备机的守护进程负责备机的故障处理，并且在备机的故障解决之后，主机将 REDO 日志发送给备机，实现主备机之间的数据同步。

故障自动切换的前提条件是，守护进程的故障处理策略是基本型或者是增强型，不能是

不支持类型。

在发生故障时，基于守护进程的数据守护系统能够达到以下的效果：

1. 主机故障时，备机自动切换为主机，对外提供数据库服务；
2. 备机故障时，主机可正常提供数据库服务；
3. 故障实例恢复后，自动同步历史数据，并作为备机加入系统；
4. 当实例异常终止时，重新启动数据库实例。

主备机自动切换完成后，客户端连接会根据 `dm_svc.conf` 配置的 `SERVERNAME` 配置项自动切换连接到新的主机，可参考本手册 2.1.1.2 节，前提是客户端连接时使用了 `SERVERNAME`。

33.5.3 手动切换

和基于观察器的数据守护方案相似，基于守护进程的数据守护方案也支持手动切换主备机。用户通过监视器向守护进程发送 `SWITCHOVER` 命令，守护进程收到之后，和属于同一监控组的另一个守护进程进行交互，并完成切换。

手动切换的前提条件是，守护进程的故障处理策略是基本型或者是增强型，且守护进程 `DW_A` 和 `DW_B` 都处于 `OPEN` 状态，并且主机的实时归档为 `VALID` 状态。结果为：原主机切换为备机，原备机切换为主机。

33.5.4 查询守护进程状态

在系统运行的过程中，用户如果想了解当前站点的守护进程的状态等信息，可以通过 `SHOW` 命令来查询守护进程的信息。

33.5.5 同时监控多个实例

在基于守护进程的数据守护方案中，一个守护进程可以同时监控多个实例（这些实例必须在同一台主机上），并且不同实例的模式可以不同（例如，守护进程同时监控实例 `A` 和实例 `B`，并且实例 `A` 为主机模式，同时实例 `B` 为备机模式）。需要注意的是，这些实例属于不同的监控组。一个守护进程最多监控 8 个实例，太多的话，会给站点造成很大的负担，降低系统效率。

33.5.6 停止对实例的监控

如果想关闭数据库实例，必须先通过监视器发布停止监控组的命令，守护进程收到命令后，关闭对实例的监控，然后用户才能关闭实例；否则，如果直接关闭数据库实例，可能导致守护进程强制重启实例，或者故障自动切换等问题。

33.5.7 重启对实例的监控

当用户修改完配置或者其他操作完毕，可以重新开启守护进程对实例的监控。用户通过监视器向守护进程发送 `STARTUP` 的命令，守护进程收到之后，就开始监控对应的数据库实例。这有一个前提：主备机上的守护进程都是启动的，并且配置信息无误。

33.5.8 网络故障检测

当用户配置好守护进程(实时归档状态), 主备系统正常运行之后, 主机上的守护进程会自动监测主机对外提供服务的网络状况。当主机对外提供服务的网络出现故障时, 守护进程就会强制主机退出服务, 原来的备机会切换为主机, 继续对外提供服务。

当守护进程出现双主机时, 守护进程强制自己监控的实例处于备机模式并退出。守护进程设置为 SHUTDOWN 状态; 如果守护进程长时间不启动 (由 dm.ini 文件中的 DW_MAX_SVR_WAIT_TIME 参数决定), 则服务器自动退出。

33.5.9 从命令行运行守护进程

在 Windows “命令提示符” 窗口中输入带参数的命令启动 dmwatch 工具, 命令格式为 dmwatch [ini 文件[-NOCONSOLE] | HELP], 如图 33.10 所示:

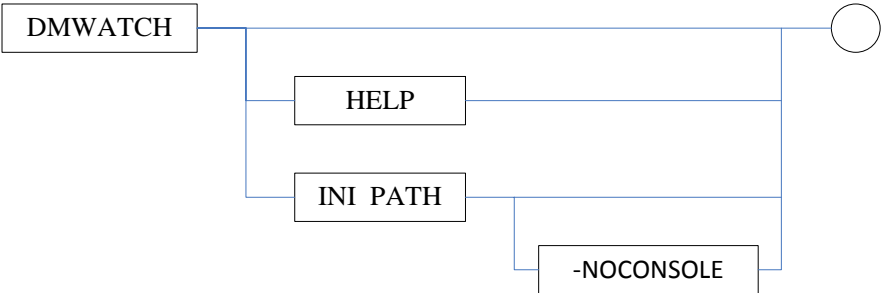


图 33.10 守护进程启动参数

参数说明见表 33.3。

表 33.3 dmwatch 工具参数说明

参数	含义	备注
ini 文件	合法的监视器的 ini 文件的位置。包括文件名	可选
- NOCONSOLE	禁用控制台	可选
HELP	显示帮助信息	可选

说明: -NOCONSOLE 参数禁用了 dmwatch 提供给用户直接使用的命令, 如 help、exit 等命令。

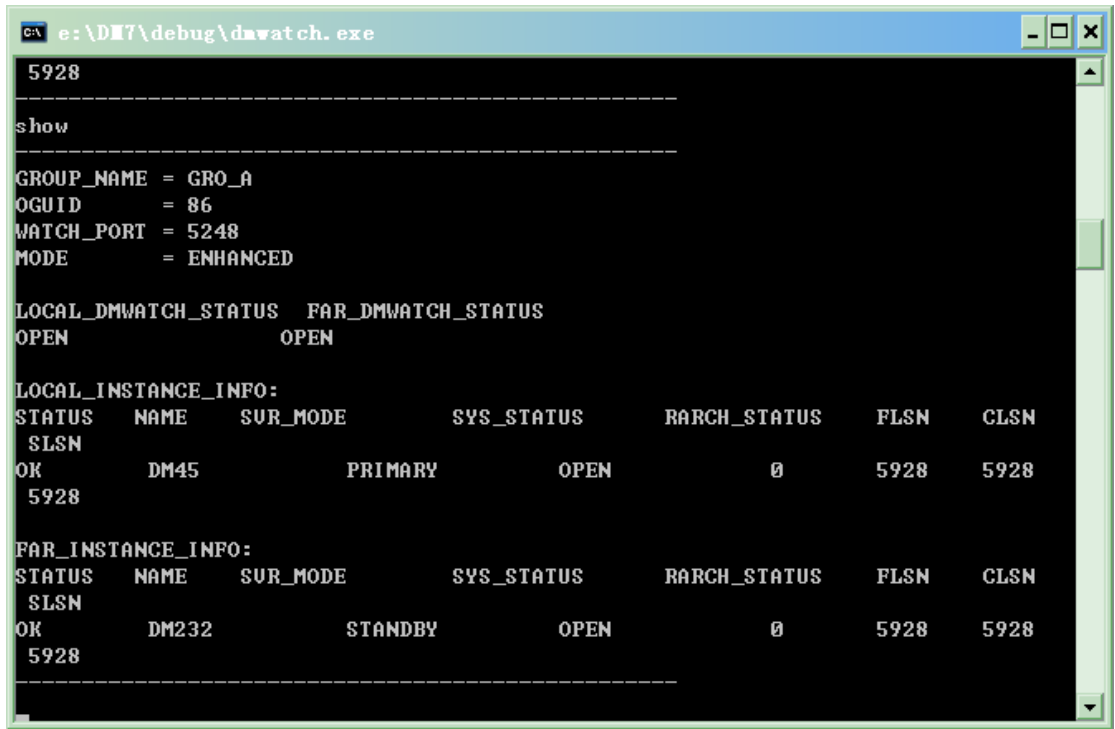


图 33.11 dmwatch 启动成功后的屏幕显示

33.5.10 守护进程提供的命令

1. 退出守护进程

EXIT

2. 显示帮助信息

HELP

3. 显示系统信息

SHOW

33.5.11 从命令行运行 dmwmon

在 Windows “命令提示符” 窗口中输入带参数的命令启动 dmwmon 工具，命令格式为 dmwmon [ini 文件[-NOCONSOLE] | HELP]，如图 33.12 所示：

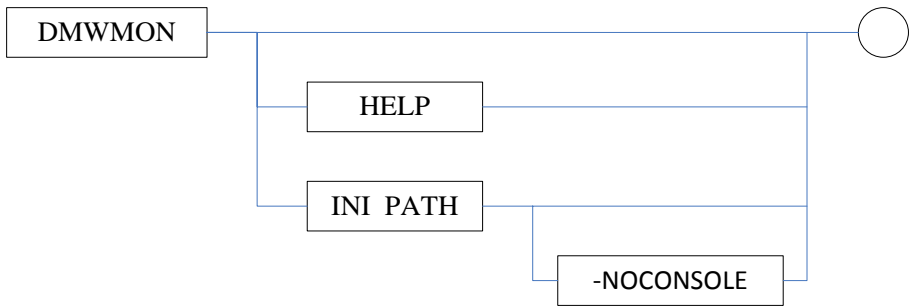


图 33.12 监视器启动参数

参数说明见表 33.4。

表 33.4 dmwmon 工具参数说明

参数	含义	备注
ini 文件	合法的监视器的 ini 文件的位置。包括文件名	可选
- NOCONSOLE	禁用控制台	可选
HELP	显示帮助信息	可选

说明：-NOCONSOLE 参数禁用了 dmwmon 提供给用户直接使用的命令，如 switchover、stop 等命令。

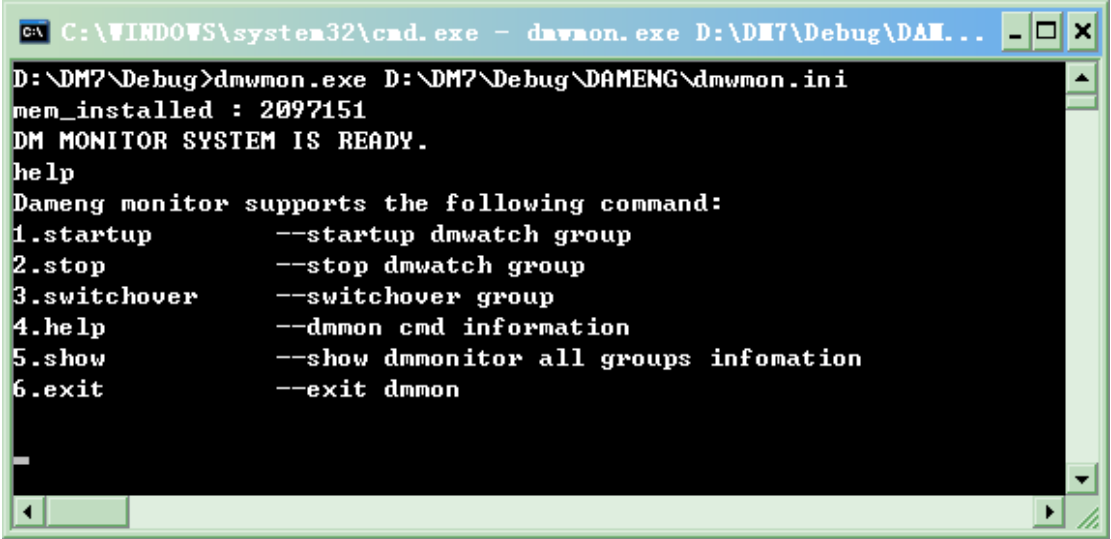


图 33.13 dmwmon 启动成功后的屏幕显示

33.5.12 监视器 DMWMON 命令

当监视器启动之后，会显示监视组的守护进程信息。监视器提供以下命令来监控数据守护系统中的守护进程：

1. 停止对监控组的监控

STOP

2. 重启对监控组的监控

STARTUP

3. 手动切换主备机

SWITCHOVER

4. 显示帮助信息

HELP

5. 显示系统信息

SHOW

7. 退出监视器

EXIT

第 34 章 数据复制

34.1 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。

34.2 重要概念

1. 数据库状态

服务器的状态，在不同的状态下，对能够进行的操作由不同的限制。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

从服务器上作为数据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。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。

34.3 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图 34.1 所示。

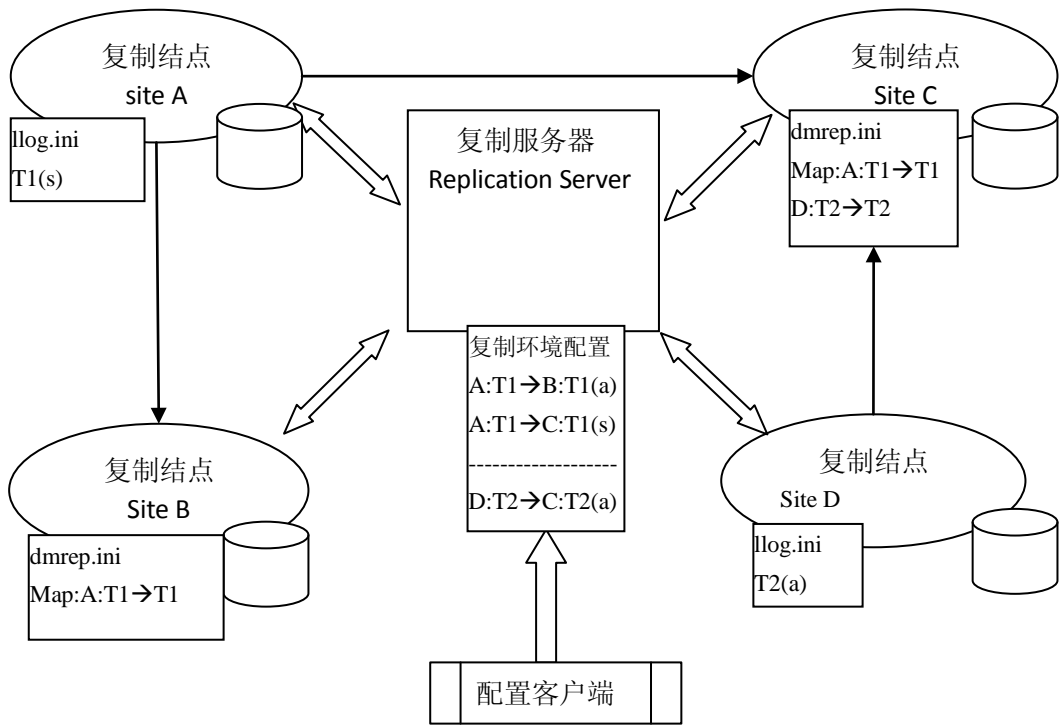


图 34.1 复制环境结构图

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表 34.1 所示。

表 34.1 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，其他的重要文件如表 34.2。

表 34.2 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。

34.4 配置数据复制

配置数据复制在 RPS 上进行，所有的配置接口见附录 3。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表 34.3 所示。

表 34.3 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 中对应项如表 34.4 所示。

表 34.4 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 MAIL_INI = 1
主服务器	INSTANCE_NAME = B PORT_NUM = 5236 MAIL_INI = 1
从服务器	INSTANCE_NAME = C PORT_NUM = 5236 MAIL_INI = 1

dmmal.ini 配置，用户实际配置时需根据情况更改对应项。

```
[MAL_INST1]
MAL_INST_NAME = A
MAL_HOST      = 192.168.0.11
MAL_PORT      = 5241
[MAL_INST2]
MAL_INST_NAME = B
MAL_HOST      = 192.168.0.12
MAL_PORT      = 5242
[MAL_INST3]
MAL_INST_NAME = C
MAL_HOST      = 192.168.0.13
MAL_PORT      = 5243
```

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，需要执行复制服务器初始化脚本来初始化复制服务器。其主要内容是创建复制用户和创建复制服务器上需要的系统表，初始化脚本参考 34.6 节。

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 REPB2C

```
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。这些删除和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。

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',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();

34.5 监控数据复制

34.5.1 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。具体内容见表 34.5。

表 34.5 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	0x200
		已在主服务器删除	0x210
		已在从服务器删除	0x201

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 C。

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的时，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。

34.5.2 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。

34.6 初始化 RPS 脚本

```
--以 DBA 身份登录，创建复制用户，并授予 RESOURCE 角色
DROP USER SYSREP;
CREATE USER SYSREP IDENTIFIED BY SYSREP;
GRANT RESOURCE TO SYSREP;

--复制用户 SYSREP 登录，创建复制信息表
--1.创建复制组表
DROP TABLE SYSREP.RPS_GROUPS;
CREATE TABLE SYSREP.RPS_GROUPS(
    NAME          VARCHAR(128),          --组名
    ID             INT,                  --组 ID
    DESC$          VARCHAR(1024),        --组描述
    CLUSTER PRIMARY KEY (NAME)
);

--2.创建复制实例表
DROP TABLE SYSREP.RPS_INSTANCES;
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     INT,                  --节点有效标记，' Y' 或' N'
    FAULT_TIME     DATETIME,             --节点故障开始时间
    NET_VALID_FLAG INT,                  --网络有效标记，' Y' 或' N'
    NET_FAULT_TIME DATETIME,             --网络故障开始时间
    CLUSTER PRIMARY KEY(GRP_ID, INST_NAME)
);

--3.创建复制关系表
DROP TABLE SYSREP.RPS_REPLICATIONS;
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),          --主节点归档日志目录
    FAULT_TIMEOUT  INT,                  --故障超时处理值，以秒为单位，0 为立即超时
```

```

VALID_FLAG    INT,                --有效标记,' Y' 或' N'
FAULT_TIME    DATETIME,           --故障开始时间
SYNC_FLAG     INT,                --同步、异步标记
TIMER_NAME    VARCHAR(128),       --异步定时器名称
DESC$         VARCHAR(1024),      --复制描述
CLUSTER PRIMARY KEY(GRP_ID, REP_NAME)
);

--4.创建复制映射表
DROP TABLE SYSREP.RPS_TABMAPS;
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
    MTAB_ID     INT,                --主表 ID
    SSCH_NAME   VARCHAR(128),       --从表模式名
    STAB_NAME   VARCHAR(128),       --从表名
    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.创建故障历史表
DROP TABLE SYSREP.RPS_FAULT_HISTORY;
CREATE TABLE SYSREP.RPS_FAULT_HISTORY(
    GRP_NAEM   VARCHAR(128),        --复制组 ID
    OBJ_NAME   VARCHAR(128),        --故障节点或关系编号
    FAULT_TYPE  VARCHAR(128),        --故障类型,' INSTANCE' 或' ROUTE'
    START_TIME  DATETIME,            --故障开始时间
    END_TIME    DATETIME             --故障结束时间
);

--6.创建冲突历史表
DROP TABLE SYSREP.RPS_CONFLICTS;
CREATE TABLE SYSREP.RPS_CONFLICTS
(
    SEQ_NO      BIGINT,              --序列号
    REP_ID      INT,                 --对应复制号
    INST_ID     INT,                 --产生冲突的实例编号
    TABLE_ID   INT,                 --冲突表 ID
    TYPE        BYTE,                --冲突类型(更新冲突, 插入冲突)
    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
    CLUSTER PRIMARY KEY(SEQ_NO)
);

```

--7.创建复制定时器表

```
DROP TABLE SYSREP.RPS_TIMERS;
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_REP_CREATE_TIMER
    FREQ_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    DATE,            --开始日期
    DURING_END_DATE      DATE,            --结束日期
    NO_END_DATA_FLAG     INT,
    DESC$                VARCHAR(1000),   --定时器描述
    CLUSTER PRIMARY KEY(NAME)
);
```

第 35 章 大规模并行处理 MPP

35.1 概述

为了支持海量数据存储和处理、高并发处理、高性价比、高可用性等需求，提供高端数据库解决方案，达梦数据库提供了大规模并行处理 MPP 架构，以极低的成本代价，为客户提供业界领先的计算性能。达梦采用完全对等无共享（share-nothing）的 MPP 架构，支持 SQL 并行处理，可自动化分区数据和并行查询，各实例无 I/O 冲突。

MPP 系统工作起来就像是一台单独的计算机，由于采用自动化的并行处理，在分析型数据库等 OLAP 应用中，查询性能比传统的单节点数据库大大提高。

MPP 系统为新一代数据仓库所需的大规模数据和复杂查询提供了先进的软件级解决方案，具有业界先进的架构和高度的可靠性，能帮助企业管理好数据，使之更好的服务于企业，推动数据依赖型企业的发展。

35.2 基本概念

1. MPP

Massive Parallel Processing，大规模数据处理。MPP 架构最多可以同时部署 1024 个节点，采用统一的并行操作数据库引擎，将数据分散在不同的数据库节点上，在高速的内部网络环境下，对于海量数据的并发查询可极大地减少 I/O，提高查询效率。

2. 执行节点（EP）

MPP 环境内每个实际运行的实例节点称为一个执行节点（基于主备的 MPP 环境内备机除外）。

3. 主、从 EP

MPP 是对等无共享架构，对整个系统来说，每个节点作用都是一样的，用户可以连接到其中的任何一个进行操作。对每个用户会话来说，执行节点具有主从之分。用户会话实际连接的那个节点称为主 EP，其余的节点都称为从 EP。

35.3 体系架构

达梦的 MPP 架构提供数据存储和查询的自动化并行，插入的数据会根据表定义的类型被自动分区保存到相应的节点，查询时在各节点的数据库引擎上并行执行查询，并将所有结果数据自动汇集到主查询节点上，数据无共享、无 I/O 冲突。

可靠性方面，提供了数据守护方案，满足用户对数据可靠性、安全性的需求，解决由于硬件故障、自然灾害等原因导致的数据库服务长时间中断问题。MPP 系统中的每个节点都可配置数据守护系统，增加多个节点数据的完整冗余镜像。节点和镜像节点之间可以互相转换成 MPP 系统中的执行节点。

系统体系架构图如图 35.1 所示：

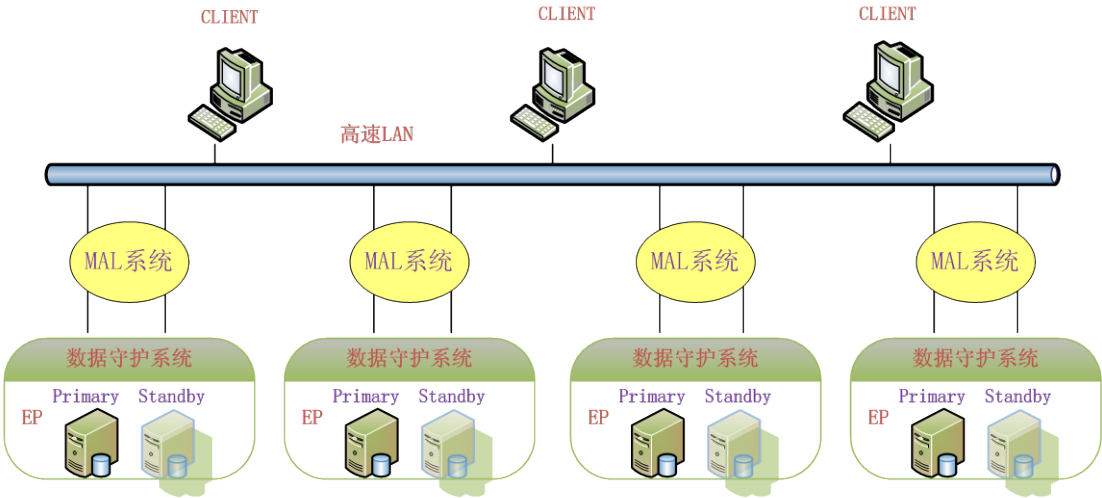


图 35.1 MPP 体系架构

客户端可连接任意一个 EP 站点进行操作，所有 EP 对客户来说都是对等的。图中互为镜像的节点是配置了守护进程的主备机，在主机发生故障时守护进程会自动将镜像备机切换为主机，继续提供对外服务，大大提高了系统的可用性和可靠性。在确保节点安全的情况下，用户也可以不用配置数据守护系统。

MPP 系统内每个节点只负责自身部分数据的读写，执行计划是完全并行执行，完全无 I/O 冲突，充分利用硬件资源，只是额外增加了数据在节点间相互传递的通信时间，在高速局域网内，通信时间占整体时间的比例更小，更能体现大规模并行处理的优势，随着系统规模的扩大，并行支路越多，优势越明显。

通用数据库主要是针对 OLTP 处理功能设计的，在运行大量小规模交易查询数据时效果最好。从结构上来说，采用了完全共享或磁盘共享架构。完全共享体系局限于单节点服务器，通常价格比较昂贵，其扩展性和性能受到相应的限制。磁盘共享体系允许系统带有多个服务器实例，这些实例与 SAN 或其它共享存储设备相连。这种体系需要通过一个狭窄的数据管道将所有 I/O 信息过滤到昂贵的共享磁盘子系统。

与此相比，基于硬件的数据仓库平台也采用完全无共享体系。在这种体系下，通讯功能部署在一个高宽带网络互连体系上，用户通过一个主控制节点执行并行查询。该体系的一个重要优势就是每个节点都有一个通往本地磁盘的独立高速通道，不但简化了体系，还提供良好的扩展性。但这些平台所面临的困难是需要价格昂贵的专用硬件，显著增加了系统成本。

达梦 MPP 的完全对等无共享体系架构结合了完全无共享体系的优点，在此基础上又前进了一步，通过软件提升处理能力，它不需要价格昂贵的专用硬件，也不采用增加主控制节点来协调所有并行处理的主从式方法，而是各个节点完全对等，更进一步简化了体系的实现，也消除了系统可能存在的主节点瓶颈问题。

综上所述，将主流架构特点比较列于表 35.1 中

表 35.1 主流架构特点比较

架构名称	特点
完全共享	局限于单节点服务器，价格昂贵，扩展性、性能受限。
共享磁盘	允许多个服务器实例共享存储设备，可有效解决单实例负载问题，具有一定的扩展性，但当节点增加到一定程度以后，由于对 I/O 资源、锁资源等的激烈竞争，反而导致性能的下降，扩展性和性能在系统规模变大时受限。同时共享磁盘等硬件成本也十分昂贵。
完全不共享	部署在高速网络，各节点相对独立，无共享 I/O，扩展性和性能良好，缺点是某

	些此类架构的系统基于硬件，价格昂贵，成本高，系统中有一个主控节点，系统规模扩充时可能成为瓶颈，主控节点无备份，容易形成单点故障。
完全对等不共享	DM7 采用的 MPP 架构，继承了完全不共享架构的优点，且各节点完全对等，不需要专用硬件，不存在主控节点，消除了潜在瓶颈以及单节点故障问题。

35.4 支持的功能

35.4.1 支持多种分布

支持随机分布、哈希分布、范围分布、LIST 列表分布、复制分布等多种分布表的类型，用户可根据实际情况选择合适的分布类型。详见《SQL 语言使用手册》中建表语法部分。

35.4.2 函数和过程

支持系统内部函数、用户定义的函数、过程等。支持函数、过程中的查询语句，支持过程中动态执行语句。

35.4.3 支持触发器

支持系统所有的触发器类型，包括事件触发器、时间触发器、DML 触发器、视图触发器（Instead of 触发器）等。

35.4.4 支持分区表

支持水平分区表、垂直分区表，即实现“数据分布后再分区”。在数据分布到各节点的基础上后，再在单个节点上将数据再次分区，可进一步提高查询性能。分布的类型和分区的类型可以混合搭配，比如建立范围分布的水平分区 LIST 表。

35.4.5 自增列和序列

所有节点维护一个可以同步的自增列和序列值。

35.4.6 支持大字段

MPP 下支持带大字段的表。

35.4.7 支持列存储

在数据仓库（OLAP）、数据挖掘等查询密集型应用中可使用列存储表，在查询只需要少数几个字段的时候，能大大减少 I/O，同时列存储也利于使用设计更好的压缩/解压算法，进一步减少存储空间。

35.4.8 支持 HFS

借助巨型表空间优化列存储数据的读写效率，从而实现对列存储表数据装载以及查询扫描的性能提升。MPP 下对巨型列存储表数据进行分布，进一步提高了列存储的读写性能。

35.5 数据加载

支持数据快速加载，MPP 下分布表的数据在主 EP 上实现重分发。用法和普通表的加载一样，详见本书《第 18 章 快速数据加载》部分。

35.6 环境配置

达梦的 MPP 环境包含两种配置模式。一种是普通配置模式，一种是基于主备的配置模式。当用户对系统的可靠性要求不高、系统可中断、能保证节点安全或进行简单的 MPP 测试时，只需使用普通配置模式，否则可使用基于主备的配置模式。

35.6.1 普通配置

普通配置模式下，每个执行节点 EP 不配置数据守护系统，系统中只包含所有 EP 节点。下面举例包含 2 个实例节点的 MPP 普通配置，步骤如下：

1. 规划

表 35.2 普通配置节点规划

实例名	机器 IP	服务器端口	MAL 端口	MPP_SEQNO
EP01	192.168.0.1	5236	5269	0
EP02	192.168.0.2	5237	5270	1

2. 配置 dm.ini

EP01 的 dm.ini 配置如下：

```
INSTANCE_NAME = EP01
PORT_NUM      = 5236

MAIL_INI      = 1
MPP_INI       = 1
```

EP02 的 dm.ini 配置如下：

```
INSTANCE_NAME = EP02
PORT_NUM      = 5237

MAIL_INI      = 1
MPP_INI       = 1
```

3. 配置 dmmal.ini

两个实例配置一样，直接拷贝即可。

```
[mal_inst1]
mal_inst_name = EP01
mal_host      = 192.168.0.1
```

```
mal_port      = 5269

[mal_inst2]
mal_inst_name = EP02
mal_host      = 192.168.0.2
mal_port      = 5270
```

4. 配置 dmmpp.ctl

dmmpp.ctl 是一个二进制文件，用户不能直接配置。用户需要首先配置 dmmpp.ini，然后利用 dmctlcvt 工具进行转换得到生成 dmmpp.ctl 文件。两个实例使用相同的 ctl 文件，只需拷贝一份。

dmmpp.ini 配置如下：

```
[service_name1]
mpp_seq_no    = 0
mpp_inst_name = EP01

[service_name2]
mpp_seq_no    = 1
mpp_inst_name = EP02
```

利用工具转换成 dmmpp.ctl。示例如下：

```
dmctlcvt t2c d:\dameng\dmmpp.ini d:\dameng\dmmpp.ctl
```

以上步骤完成之后，只需分别启动两个实例节点，如果能够登录成功，说明配置成功了。

35.6.2 基于主备的配置

基于主备的 MPP 环境配置比普通配置步骤稍复杂一些，MPP 主备系统由 MPP_MONITOR(MPP 监控器)和多个 EP(MPP 节点)及相应的备机组成。使用 MPP 架构的系统需要用户手动为 MPP_MONITOR 和每个 EP 配置两个配置文件：dmmal.ini 和 dmmpp.ctl。另外对于每个 EP，需要启用 dm.ini 中的 MAIL_INI、MPP_INI 和 ARCH_INI 参数，对于每个 EP 对应的备机需要启用 dm.ini 中的 MAIL_INI 和 ARCH_INI。比如配置如下表的节点数为 2，EP01 的有两个备机 EP11 和 EP12EP02 的有一个备机 EP21 系统，配置步骤如下：

1. 规划

主机：

表 35.3 基于主备的配置节点规划

实例名	机器 IP	端口	MAL 端口	MPP_SEQNO
EP01	192.168.0.28	5236	5269	0
EP02	192.168.0.28	5237	5270	1

备机：

表 35.4 基于主备的配置节点规划

主机	备机	机器 IP	端口	MAL 端口
EP01	EP11	192.168.0.28	5238	5271
	EP12	192.168.0.28	5239	5272
EP02	EP21	192.168.0.28	5240	5273

MONITOR:

表 35.5 基于主备的配置节点规划

	机器 IP	MAL 端口
MONITOR	192.168.0.28	5280

2. 初始化库

基于主备的 MPP 配置，要求主备机数据严格一致，因此用户配置主备时，需要保证这一点。通过将所有节点初始化，保证它们的初始状态数据是一致的。Windows 下初始化库脚本如下：

```
start "01" cmd /c "D:\DM7svn\debug\dminit path=D:\MPP_ARCH_TEST\ep01"
start "02" cmd /c "D:\DM7svn\debug\dminit path=D:\MPP_ARCH_TEST\ep02"
start "11" cmd /c "D:\DM7svn\debug\dminit path=D:\MPP_ARCH_TEST\ep11"
start "12" cmd /c "D:\DM7svn\debug\dminit path=D:\MPP_ARCH_TEST\ep12"
start "21" cmd /c "D:\DM7svn\debug\dminit path=D:\MPP_ARCH_TEST\ep21"
```

3. 配置 dm.ini

EP01 配置如下：

```
INSTANCE_NAME = EP01
PORT_NUM      = 5236

MAIL_INI      = 1
MPP_INI       = 1
ARCH_INI      = 1
```

EP02 配置如下：

```
INSTANCE_NAME = EP02
PORT_NUM      = 5237

MAIL_INI      = 1
MPP_INI       = 1
ARCH_INI      = 1
```

EP11 配置如下：

```
INSTANCE_NAME = EP11
PORT_NUM      = 5238

MAIL_INI      = 1
ARCH_INI      = 1
```

EP12 配置如下：

```
INSTANCE_NAME = EP12
PORT_NUM      = 5239

MAIL_INI      = 1
ARCH_INI      = 1
```

EP21 配置如下：

```
INSTANCE_NAME = EP21
PORT_NUM      = 5240
MAIL_INI      = 1
ARCH_INI      = 1
```

4. 配置 dmmal.ini

```

[mal_inst1]
    mal_inst_name = EP01
    mal_host      = 192.168.0.28
    mal_port      = 5269

[mal_inst2]
    mal_inst_name = EP02
    mal_host      = 192.168.0.28
    mal_port      = 5270

[mal_inst3]
    mal_inst_name = EP11
    mal_host      = 192.168.0.28
    mal_port      = 5271

[mal_inst4]
    mal_inst_name = EP12
    mal_host      = 192.168.0.28
    mal_port      = 5272

[mal_inst5]
    mal_inst_name = EP21
    mal_host      = 192.168.0.28
    mal_port      = 5273

[monitor]
    mal_inst_name = monitor
    mal_host      = 192.168.0.28
    mal_port      = 5280

```

5. 配置 dmmpp.ctl

dmmpp.ctl 是一个二进制文件，用户不能直接配置。用户需要首先配置 dmmpp.ini，然后利用 ctlevt 工具进行转换得到生成 dmmpp.ctl 文件。

dmmpp.ini 配置如下：

```

[service_name1]
    mpp_seq_no    = 0
    mpp_inst_name = EP01

[service_name2]
    mpp_seq_no    = 1
    mpp_inst_name = EP02

```

利用工具转换成 dmmpp.ctl。示例如下：

```
dmctlevt t2c C:\winnt\system32\EP01\dmmpp.ini C:\winnt\system32\EP01\dmmpp.ctl
```

备注：monitor 和所有 EP 站点共用同一套 dmmal.ini 和 dmmpp.ctl 文件。对于 EP 站点，将 dmmal.ini 和 dmmpp.ini 放在与其控制文件相同的目录下；对于 monitor，用户单独指定一个目录存放，这个目录在启动 monitor 时，通过参数 path 指定。

6. 配置 dmarch.ini

EP01 配置如下：

```
[ARCHIVE_LOCAL1]
ARCH_TYPE      = LOCAL
ARCH_DEST      = D:\MPP_ARCH_TEST\ep01\DAMENG\arch
ARCH_FILE_SIZE = 128
ARCH_SPACE_LIMIT = 0    #空间限制 Gb 为单位，范围 0~ 4096，默认 0 表示无空间限制
```

```
[ARCHIVE_MPP1]
ARCH_TYPE      = MARCH
ARCH_DEST      = EP11
```

```
[ARCHIVE_MPP2]
ARCH_TYPE      = MARCH
ARCH_DEST      = EP12
```

EP11 配置如下：

```
[ARCHIVE_LOCAL1]
ARCH_TYPE      = LOCAL
ARCH_DEST      = D:\MPP_ARCH_TEST\ep11\DAMENG\arch
ARCH_FILE_SIZE = 128
ARCH_SPACE_LIMIT = 0    #空间限制 Gb 为单位，范围 0~ 4096，默认 0 表示无空间限制
```

```
[ARCHIVE_MPP1]
ARCH_TYPE      = MARCH
ARCH_DEST      = EP01
```

```
[ARCHIVE_MPP2]
ARCH_TYPE      = MARCH
ARCH_DEST      = EP12
```

EP12 配置如下：

```
[ARCHIVE_LOCAL1]
ARCH_TYPE      = LOCAL
ARCH_DEST      = D:\MPP_ARCH_TEST\ep12\DAMENG\arch
ARCH_FILE_SIZE = 128
ARCH_SPACE_LIMIT = 0    #空间限制 Gb 为单位，范围 0~ 4096，默认 0 表示无空间限制
```

```
[ARCHIVE_MPP1]
ARCH_TYPE      = MARCH
ARCH_DEST      = EP01
```

```
[ARCHIVE_MPP2]
ARCH_TYPE      = MARCH
ARCH_DEST      = EP11
```

EP02 配置如下：

```
[ARCHIVE_LOCAL1]
ARCH_TYPE      = LOCAL
ARCH_DEST      = D:\MPP_ARCH_TEST\ep02\DAMENG\arch
ARCH_FILE_SIZE = 128
ARCH_SPACE_LIMIT = 0    #空间限制 Gb 为单位，范围 0~ 4096，默认 0 表示无空间限制
```

```
[ARCHIVE_MPP1]
ARCH_TYPE      = MARCH
ARCH_DEST      = EP21
```

EP21 配置如下:

```
[ARCHIVE_LOCAL1]
ARCH_TYPE      = LOCAL
ARCH_DEST      = D:\MPP_ARCH_TEST\ep21\DAMENG\arch
ARCH_FILE_SIZE = 128
ARCH_SPACE_LIMIT = 0      #空间限制 Gb 为单位, 范围 0~ 4096, 默认 0 表示无空间限制
```

```
[ARCHIVE_MPP1]
ARCH_TYPE      = MARCH
ARCH_DEST      = EP02
```

每个 EP 站点和备机将必须配置本地归档, 主机归档配置中包含所有备机, 每个备机归档配置包含主机和其它备机。

7. 启动各个 EP 和备机

可使用批处理脚本方式启动, 如 windows 下启动脚本:

```
start "01" cmd /c "D:\DM7svn\debug\dmsserver D:\MPP_ARCH_TEST\ep01\DAMENG\dm.ini mount"
start "02" cmd /c "D:\DM7svn\debug\dmsserver D:\MPP_ARCH_TEST\ep02\DAMENG\dm.ini mount"
start "11" cmd /c "D:\DM7svn\debug\dmsserver D:\MPP_ARCH_TEST\ep11\DAMENG\dm.ini mount"
start "12" cmd /c "D:\DM7svn\debug\dmsserver D:\MPP_ARCH_TEST\ep12\DAMENG\dm.ini mount"
start "21" cmd /c "D:\DM7svn\debug\dmsserver D:\MPP_ARCH_TEST\ep21\DAMENG\dm.ini mount"
```

8. 设置实例的状态和模式

系统认为配置了 MARCH 的实例都必需处于主机模式或是备机模式, 否则限制登录; 另外为了避免设置状态模式过程中, 产生归档日志, 因此在利用命令设置站点的状态和模式前, 先关闭归档参数, 即将参数 ARCH_INI 设置为 0, 然后以 mount 方式启动服务, 利用 ISQL 登录进行状态和模式设置, 设置完毕后, 重新设回 ARCH_INI = 1。注意: 为设置服务器状态和模式, 服务器需要以 mount 方式启动。

将站点设置为主机模式通过如下 DISQL 命令实现, 依次登录站点 EP01, EP02, 将它们设置为主机配置状态, 设置完毕后, 需要将 ARCH_INI 重新设置为 1, 然后重启服务。比如, 登录 EP01, 将其设置为主机配置状态。

```
SQL>login
服务名:
用户名:
密码:
端口号:
SSL 路径:
SSL 密码:
USBKey Pin:
服务器[LOCALHOST:5236]:处于普通配置状态
登录时间 :169.313(ms)
SQL>
SQL>alter database primary;
操作已执行。
已用时间: 179.322(ms) clock tick:590315888. Execute id is 0.
```

将站点设置为备机模式通过如下 DISQL 命令实现, 依次登录站点 EP11、EP12 和 EP21, 将它们设置为备机配置状态, 设置完毕后, 需要将 ARCH_INI 重新设置为 1, 然后重启服务。

比如，登录 EP11，将其设置为备机配置状态。

```
SQL>login
服务名:
用户名:
密码:
端口号:5238
SSL 路径:
SSL 密码:
USBKey Pin:
服务器[LOCALHOST:5238]:处于普通配置状态
登录时间 :22.545(ms)
SQL>alter database standby;
操作已执行。
已用时间: 129.892(ms) clock tick:427596594. Execute id is 0.
```

9. 启动 mpp_monitor

可使用批处理脚本方式启动，如 windows 下启动脚本：

```
start "monitor" cmd /c "D:\DM7svn\debug\dmmp_monitor path= D:\MPP_ARCH_TEST\monitor"
```

参数 path 为 dmmal.ini 和 dmmpm.ctl 等配置文件以及 montior 运行过程中产生日志文件的存放目录。启动 monitor 后，在此工具控制台上输入 status，如果状态正常，则系统创建成功。mpp_monitor 工具参数如表 35.1 所示。

表 35.1 MPP_MONITOR 工具参数说明

参数	含义	备注
PATH	dmmpm.ctl、dmmal.ini 的绝对目录或者当前目录，最大长度为 256 字节	必选参数
INTERVAL	检查 EP 站点的间隔时间, 取值范围 1~10 秒, 默认值 1	可选参数
-NOCONSOLE	以服务方式启动	可选参数
HELP	打印帮助信息	可选参数

35.6.3 监控器使用说明

按照先 EP 站点和备机站点，后 monitor 的次序启动整个 MPP 主备系统，EP 站点之间和备机站点次序不分先后。对于主站点，用户需要通过客户端命令将其设置处于主机配置状态，对于备用站点，用户需要通过客户端命令将其设置处于备机配置状态。在 EP 站点及备机正常启动后，最后启动 monitor。monitor 运行时相关信息记录在 monitor 的配置文件目录的名为 monitor.log 的文件中；每个站点的故障处理以及故障恢复日志记录在名为 monitor_站点实例名_站点编号.log 的文件中。

MPP_monitor 监控器的命令说明如下：

帮助 help:

```
help
+-----+-----+
|status   |   系统状态           |
+-----+-----+
|details  |   MPP 主备信息       |
+-----+-----+
|exit     |   退出               |
+-----+-----+
Input your command, type h for help.
```

查看状态 status:

```
status
-----MPP SYS REPORT-----
    站点数      : 2
    状态        : 正常
-----

Input your command, type h for help.
```

详细信息 details:

```
details
----- MPP SYS DETAILS -----
    站点号   实例名   是否正常   正常备机实例名   异常备机实例名
        0     EP01       Y       {EP11, EP12}       {}
        1     EP02       Y       {EP21}             {}
-----

Input your command, type h for help.
```

异常备机包括：实例故障，网络故障或者 MPP 主站点对应的 MARCH 为 INVALID。

1. 主机故障处理

使用场景：EP 主站点故障后，自动进入主机故障处理。假设把 EP01 关闭，模拟 EP01 故障，monitor 检测 EP01 故障后，选择一个备机进行切换，完成后查看。

```
details
----- MPP SYS DETAILS -----
    站点号   实例名   是否正常   正常备机实例名   异常备机实例名
        0     EP11       Y       {EP12}       {EP01}
        1     EP02       Y       {EP21}       {}
-----

Input your command, type h for help.
```

查看 mpp_ep_0.log:

```
2012-06-25 13:43:39 主站点 EP01 出现故障，系统试图进行主备切换...
/
2012-06-25 13:43:39 获取站点 EP11 信息,sys_status:3(3:mount 4:open 5:suspend) svr_mode:2(0:普通 1:
主机 2:备机) slsn:6010
/
2012-06-25 13:43:39 获取站点 EP12 信息,sys_status:3(3:mount 4:open 5:suspend) svr_mode:2(0:普通 1:
主机 2:备机) slsn:6010
/
2012-06-25 13:43:39 选择 EP11 作为新主机
/
2012-06-25 13:43:39 EP11 进入 MOUNT 状态
/
2012-06-25 13:43:39 EP11 变为主机模式
/
2012-06-25 13:43:50 EP11 进入 OPEN 状态
/
2012-06-25 13:43:50 初始化站点 EP11 的 mpp 控制信息成功!
/
2012-06-25 13:43:50 设置站点 EP11 的 mpp_ini 参数成功!
/
```

2012-06-25 13:43:50 断开站点 EP11 的会话成功!

/

2012-06-25 13:43:50 MPP 系统主备切换完毕...

2. 备机故障恢复

使用场景: 备机由于某种原因故障后, 重新正常启动, 自动加入备机。比如重新启动 EP01, 模拟 EP01 恢复。将 EP01 恢复为备机, 加入到 MPP 系统中来。

details

----- MPP SYS DETAILS -----

站点号	实例名	是否正常	正常备机实例名	异常备机实例名
0	EP11	Y	{EP01, EP12}	{}
1	EP02	Y	{EP21}	{}

Input your command, type h for help.

查看 mpp_ep_0.log

2012-06-25 13:48:46 试图为主站点 EP11 添加备用站点 EP01

/

2012-06-25 13:48:46 设置备机 EP01 mpp_ini

/

2012-06-25 13:48:46 获取站点 EP01 信息,sys_status:4(3:mount 4:open 5:suspend) svr_mode:2(0:普通 1:主机 2:备机) slsn:6010

/

2012-06-25 13:48:47 EP11 进入 SUSPEND 状态

/

2012-06-25 13:48:47 主站点 EP11 与备用站点 EP01 完成同步

/

2012-06-25 13:48:47 EP11 进入 OPEN 状态

/

2012-06-25 13:48:47 为主站点 EP11 添加备用站点 EP01 完毕

/

35.7 并行查询计划

在 MPP 模式下, 查询是通过并行查询计划来进行。所谓的 MPP 并行查询计划, 是在单节点的查询计划中合适的位置插入了通讯操作符, 负责主从 EP 相互之间的数据移动和同步。目前包含 4 个通讯操作符, 详细信息见下表 35.6:

表 35.6 通讯操作符

名称	功能
MGAT	主 EP 收集所有数据, 从 EP 将数据发送到主 EP
MDIS	各 EP 间相互分发数据
MBRO	类似 MGAT, 收集数据, 还包含聚集函数运算功能
MSCT	主 EP 发散数据到所有从 EP, 一般和 MGAT 配合使用

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	INT	对象 ID
3	SCHID	INT	TYPE\$=FILE 时表示对象所属的库 ID， 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
5	SUBTYPE\$	VARCHAR(10)	对象的子类型
6	PID	INT	对象的父对象 ID
7	VERSION	INT	对象的版本
8	CRTDATE	DATETIME	对象的创建时间
9	INFO1	INT	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 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) SCHEDULE 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	INT	表/用户/数据库/文件组: 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)行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)标记是否分布表, 0x20表示HASH分布表, 0x40表示随机分布表 用户: BYTE(2)默认表空间 ID
12	INFO4	BIGINT	表: 行数 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)行触发器前可能被触发器修改值的字段, BYTE(2)行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

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	INT	索引 ID
2	ISUNIQUE	CHAR(1)	是否为唯一索引
3	GROUPIX	SMALLINT	所在文件组的 ID
4	ROOTFILE	SMALLINT	存放根的文件号
5	ROOTPAGE	INT	存放根的页号
6	TYPE\$	CHAR(2)	类型

7	XTYPE	INT	索引标记，联合其他字段标记索引类型
8	FLAG	INT	标记是否是系统索引，虚索引
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	INT	父对象 ID
3	COLID	SMALLINT	列 ID
4	TYPE\$	VARCHAR(128)	列数据类型
5	LENGTH\$	INT	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 垂直分区表： 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 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

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	INT	约束 ID
2	TABLEID	INT	所属表 ID
3	COLID	SMALLINT	列 ID，表级约束为-1
4	TYPE\$	CHAR(1)	约束类型
5	VALID	CHAR(1)	约束是否有效
6	INDEXID	INT	索引 ID
7	CHECKINFO	VARCHAR(2048)	check 约束的文本
8	FINDEXID	INT	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	INT	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	INT	统计信息 ID
2	COLID	SMALLINT	列 ID
3	T_FLAG	CHAR(1)	对象的标记
4	N_TOTAL	INT	总行数
5	N_SMAPLE	INT	采样个数
6	N_DISTINCT	BIGINT	不同值的个数
7	N_NULL	INT	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	INT	叶子段总块数
12	N_LEAF_USED_PAGES	INT	叶子占用的块数
13	CLUSTER_FACTOR	INT	索引的 cluster_factor
14	N_BUCKETS	SMALLINT	直方图桶数目
15	DATA	VARBINARY(8188)	直方图数据

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	INT	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	INT	所属对象的 ID
2	SEQNO	INT	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	CLOB	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	INT	被授权用户/角色 ID
2	OBJID	INT	授权对象 ID，对于数据库权限为-1
3	COLID	INT	表/视图列 ID，非列权限为-1
4	PRIVID	INT	权限 ID
5	GRANTOR	INT	授权者 ID
6	GRANTABLE	CHAR(1)	权限是否可转授

9. SYSAUDIT

记录系统中的审计设置。

序号	列	数据类型	说明
1	DBID	SMALLINT	数据库 ID
2	LEVEL	SMALLINT	审计级别
3	LUID	INT	用户 ID
4	TVPID	INT	表/过程/存储过程函数 ID

5	TYPE	SMALLINT	审计类型
6	WHENEVER	SMALLINT	审计情况

10. SYSAUDITRECORDS

记录系统中审计记录信息。

序号	列	数据类型	说明
1	LOGINID	INT	登录 ID
2	LOGINNAME	VARCHAR(128)	登录名
3	USERID	INT	用户 ID
4	USERNAME	VARCHAR(128)	用户名
5	IP	VARCHAR(25)	IP 地址
6	DBID	SMALLINT	数据库 ID
7	DBNAME	VARCHAR(128)	数据库名
8	SCHID	INT	模式 ID
9	SCHNAME	VARCHAR(128)	模式名
10	OBJID	INT	对象 ID
11	OBJNAME	VARCHAR(128)	对象名
12	OPERATION	VARCHAR(128)	操作类型
13	SUCC_FLAG	CHAR(1)	成功标识
14	DESCRIBE	VARCHAR(500)	审计描述
15	OPDATE	DATE	审计日期
16	OPTIME	TIME	审计时间
17	RESVD1	INT	保留
18	RESVD2	INT	保留
19	RESVD3	INT	保留
20	RESVD4	VARCHAR(128)	保留
21	RESVD5	VARCHAR(2000)	保留

11. SYSAUDITRULES

记录系统中审计规则的信息。

序号	列	数据类型	说明
1	ID	INT	规则 ID
2	NAME	VARCHAR(128)	规则名
3	LOGINID	INT	登录 ID
4	DBID	SMALLINT	数据库 ID
5	SCHID	INT	模式 ID
6	OBJID	INT	操作对象 ID
7	OPTYPE	SMALLINT	操作类型
8	SUCC_FLAG	CHAR(1)	操作结果
9	ALLOW_DT	VARCHAR(500)	时间段
10	ALLOW_ADDR	VARCHAR(500)	网段
11	INNER1	INT	时间间隔
12	INNER2	INT	操作频率门限值
13	DESCRIBE	VARCHAR(500)	描述
14	RESVD1	INT	保留
15	RESVD2	INT	保留
16	RESVD3	INT	保留

17	RESVD4	VARCHAR(128)	保留
18	RESVD5	VARCHAR(2000)	保留

12. SYSPARTTABLEINFO

记录系统中分区表的信息。

序号	列	数据类型	说明
1	BASE_TABLE_ID	INT	基表 ID
2	PART_TABLE_ID	INT	分区表 ID
3	PARTITION_TYPE	VARCHAR(10)	分区类型
4	PARTITION_NAME	VARCHAR(8188)	分区名
5	RESVD1	INT	保留
6	RESVD2	INT	保留
7	RESVD3	INT	保留
8	RESVD4	VARCHAR(128)	保留
9	RESVD5	VARCHAR(2000)	保留

13. SYSMACPLYS

记录策略定义。

序号	列	数据类型	说明
1	ID	INT	策略 ID
2	NAME	VARCHAR(128)	策略名

14. SYSMACLVLS

记录策略的等级。

序号	列	数据类型	说明
1	PID	INT	策略 ID
2	ID	SMALLINT	等级 ID
3	NAME	VARCHAR(128)	等级名

15. SYSMACCOMPS

记录策略的范围。

序号	列	数据类型	说明
1	PID	INT	策略 ID
2	ID	SMALLINT	范围 ID
3	NAME	VARCHAR(128)	范围名

16. SYSMACGRPS

记录策略所在组的信息。

序号	列	数据类型	说明
1	PID	INT	策略 ID
2	ID	SMALLINT	组 ID
3	PARENTID	SMALLINT	父节点 ID
4	NAME	VARCHAR(128)	组名

17. SYSMACLABELS

记录策略的标记信息。

序号	列	数据类型	说明
1	PID	INT	策略 ID
2	ID	INT	标记 ID

3	LABEL	VARCHAR(4000)	标记信息
---	-------	---------------	------

18. SYSMACTABPLY

记录表策略信息。

序号	列	数据类型	说明
1	TID	INT	表 ID
2	PID	INT	策略 ID
3	COLID	SMALLINT	列 ID
4	OPTIONS	BYTE	可见性

19. SYSMACUSRPLY

记录用户的策略信息。

序号	列	数据类型	说明
1	UID	INT	用户 ID
2	PID	INT	策略 ID
3	MAXREAD	INT	最大读标记 ID
4	MINWRITE	INT	最小写标记 ID
5	DEFTAG	INT	默认标记 ID
6	ROWTAG	INT	行级标记 ID
7	PRIVS	BYTE	特权

20. SYSMACOBJ

记录扩展客体标记信息。

序号	列	数据类型	说明
1	OBJID	INT	对象 ID
2	COLID	SMALLINT	列 ID
3	PID	INT	策略 ID
4	TAG	INT	标记 ID

21. SYSCOLCYT

记录列的加密信息。

序号	列	数据类型	说明
1	TID	INT	表 ID
2	CID	SMALLINT	列 ID
3	ENC_ID	INT	加密类型 ID
4	ENC_TYPE	CHAR(1)	加密类型
5	HASH_ID	INT	哈希算法 ID
6	HASH_TYPE	CHAR(1)	是否加盐
7	CIPHER	VARCHAR(512)	密钥

22. SYSACCHISTORIES

记录登录失败的历史信息。

序号	列	数据类型	说明
1	LOGINID	INT	登录 ID
2	LOGINNAME	VARCHAR(128)	登录名
3	TYPE\$	INT	登录类型
4	ACCPID	VARCHAR(128)	访问 IP
5	ACCDT	DATETIME	访问时间

23. SYSPWDCHGS

记录密码的修改信息。

序号	列	数据类型	说明
1	LOGINID	INT	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

24. SYSCONTEXTLIBS

记录词库的信息。

序号	列	数据类型	说明
1	LIB_ID	INT	词库 ID
2	LIB_NAME	VARCHAR(32)	词库名
3	LIB_PATH	VARCHAR(256)	词库路径
4	DEF_LIB	CHAR(1)	缺省词库标记
5	RESV1	VARCHAR(10)	保留
6	RESV2	INT	保留
7	RESV3	INT	保留
8	RESV4	INT	保留

25. 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	INT	索引号
3	TABLEID	INT	基表号
4	COLID	SMALLINT	列号
5	UPD_TIMESTAMP	TIMESTAMP	索引更新时间
6	TIID	INT	CTI\$INDEX_NAMES\$I 表 ID
7	TRID	INT	CTI\$INDEX_NAMES\$R 表 ID
8	TPID	INT	CTI\$INDEX_NAMES\$P 表 ID
9	WSEG_TYPE	SMALLINT	分词参数类型
10	RESVD1	SMALLINT	保留
11	RESVD2	INT	保留
12	RESVD3	INT	保留
13	RESVD4	VARCHAR(1000)	保留

26. 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(1024)	注释信息

27. 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(1024)	注释信息

28. SYSUSERS

记录系统中用户信息。

序号	列	数据类型	说明
1	ID	INT	用户 ID
2	PASSWORD	VARCHAR(128)	用户口令
3	AUTHENT_TYPE	INT	用户认证方式: NDCT_DB_AUTHENT/NDCT_OS_AUTHENT/NDCT_NET_AUTHENT/NDCT_UNKOWN_AUTHENT
4	SESS_PER_USER	INT	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	INT	用户会话的最大空闲时间
6	FAILED_NUM	INT	用户登录失败次数限制
7	LIFE_TIME	INT	一个口令在终止使用前可以使用的天数
8	REUSE_TIME	INT	一个口令在可以重新使用之前必须经过的天数
9	REUSE_MAX	INT	一个口令在可以重新使用前必须改变的次数
10	LOCK_TIME	INT	用户口令锁定时间
11	GRACE_TIME	INT	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(0)	用户最后一次的锁定时间
14	PWD_POLICY	INT	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	INT	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	INT	将引起一个帐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

29. SYSOBJINFOS

记录 HASH 分布表的分布列信息或类对象的依赖信息。

序号	列	数据类型	说明
1	ID	INT	HASH 分布表的表 ID 或 被依赖类的 ID
2	TYPE\$	INT	HASH 分布表的类型 或 依赖于类的表或类的 ID
3	VALUE	VARBINARY(2048)	HASH 分布表的分布列信息 或 在此行为类对象依赖信息时无意义

30. 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	INT	用户 ID
2	CPU_PER_CALL	INT	用户的一个请求能够使用的 CPU 时间上限（单位：秒）
3	CPU_PER_SESSION	INT	一个会话允许使用的 CPU 时间上限（单位：秒）
4	MEM_SPACE	INT	会话占有的私有内存空间上限（单位：MB）
5	READ_PER_CALL	INT	每个请求能够读取的数据页数
6	READ_PER_SESSION	INT	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）

附录 2 动态性能视图

1. V\$ALERTINFO

警告信息动态性能表，用来记录系统所有警告信息。

序号	列	数据类型	说明
1	NAME	VARCHAR(30)	警告名
2	SID	BIGINT	会话 ID
3	CHANGED	CHAR(1)	改变标记 N/Y
4	MESSAGE	VARCHAR(1800)	警告详细信息

2. 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	IP	VARCHAR(25)	IP 地址
4	SCHID	INTEGER	模式 ID
5	SCHNAME	VARCHAR(128)	模式名
6	OBJID	INTEGER	对象 ID
7	OBJNAME	VARCHAR(128)	对象名
8	OPERATION	VARCHAR(128)	操作类型名
9	SUCC_FLAG	CHAR(1)	成功标记
10	SQL_TEXT	VARCHAR(8188)	SQL 文本
11	DESCRIPTION	VARCHAR(8188)	描述信息
12	OPTIME	DATETIME	操作时间

3. V\$BUFFER

缓冲池动态性能表，用来记录缓冲池控制页结构的信息。

序号	列	数据类型	说明
1	POOL_ID	SMALLINT	缓冲池 ID
2	SPACE_ID	SMALLINT	表空间 ID
3	FILE_ID	SMALLINT	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	INDEXID	INTEGER	页内索引 ID
10	PAGETYPE	INTEGER	页类型
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	INT	正在使用的页数
14	STATUS	SMALLINT	页状态：空闲，使用，正在读，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	SMALLINT	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

4. V\$BUFFERPOOL

页面缓冲池动态性能表，用来记录页面缓冲池结构的信息。

序号	列	数据类型	说明
1	ID	SMALLINT	缓冲池 ID
2	NAME	VARCHAR(20)	缓冲池名称 NORMAL/KEEP/RECYCLE/FAST
3	PAGE_SIZE	SMALLINT	基缓冲池页大小，不包括扩展池页
4	N_PAGES	INTEGER	页数
5	N_FIXED	INTEGER	正在使用的页数目
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	INTEGER	命中率
16	N_EXP_BUFFERPOOL	INTEGER	扩展缓冲池个数

5. V\$BUFFER_LRU_FIRST

显示所有缓冲池 LRU 链首页信息。

序号	列	数据类型	说明
1	POOL_ID	SMALLINT	缓冲池 ID
2	SPACE_ID	SMALLINT	表空间 ID
3	FILE_ID	SMALLINT	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	INT	正在使用的页数目
14	STATUS	SMALLINT	页状态：空闲，使用，正在读，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	SMALLINT	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_UPD_FIRST

显示所有缓冲池 UPDATE 链首页信息。

序号	列	数据类型	说明
1	POOL_ID	SMALLINT	缓冲池 ID
2	SPACE_ID	SMALLINT	表空间 ID
3	FILE_ID	SMALLINT	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	INT	正在使用的页数目
14	STATUS	SMALLINT	页状态：空闲，使用，正在读，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	SMALLINT	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_LRU_LAST

显示所有缓冲池 LRU 链末页信息。

序号	列	数据类型	说明
1	POOL_ID	SMALLINT	缓冲池 ID
2	SPACE_ID	SMALLINT	表空间 ID
3	FILE_ID	SMALLINT	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	INT	正在使用的页数目
14	STATUS	SMALLINT	页状态：空闲，使用，正在读，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	SMALLINT	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\$BUFFER_UPD_LAST

显示所有缓冲池 UPDATE 链末页信息。

序号	列	数据类型	说明
1	POOL_ID	SMALLINT	缓冲池 ID

2	SPACE_ID	SMALLINT	表空间 ID
3	FILE_ID	SMALLINT	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	INT	正在使用的页数
14	STATUS	SMALLINT	页状态：空闲，使用，正在读，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	SMALLINT	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

9. V\$CACHEITEM

显示缓存中 ITEM 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	INT	此 ITEM 项被引用的次数
7	TS_VALUE	INT	时间戳

10. V\$CACHEPLN

显示执行计划缓存中的相关信息。在 ini 参数 USE_PLN_POOL !=0 时才统计。并且动态视图信息不加入计划缓存。

序号	列	数据类型	说明
1	CACHE_ITEM	BIGINT	计划在缓冲池中的地址
2	TYPE\$	VARCHAR	计划的类型：SQL 语句，存储过程，触发器
3	HASH_VALUE	INT	HASH 值
4	SCHID	INT	SCHEMA 的 ID 值
5	USER_ID	INT	用户 ID
6	OBJ_ID	INT	对象 ID（过程/触发器 ID）
7	RS_CAN_CACHED	CHAR	结果集是否可以缓存在缓冲区中
8	N_RS_CACHED	INT	结果集在缓存中的数目
9	N_TABLE	INT	计划涉及的表的数目
10	TABLEID	VARCHAR(256)	计划涉及的表的 ID，用空格隔开
11	SQLSTR	VARCHAR(4096)	计划对应的 SQL 语句
12	SQLCACHE	BIGINT	对应 V\$CACHESQL 中的项目
13	DDLSCRIPT	VARCHAR(4096)	DDL 执行脚本
14	RET_CMD	SMALLINT	返回响应命令字

15	STMT_TYPE	INT	SQL 语句类型
16	N_LIT_PARA	SMALLINT	常量参数数目
17	N_CLNT_PARA	SMALLINT	用户定义的参数数目
18	N_COLS	SMALLINT	要返回的列的数目
19	SEL_UPDATABLE	CHAR	计划是否可更新

11. 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
2	PLN	BIGINT	结果集对应查询计划
3	N_TABLES	SMALLINT	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

12. 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	在内存中的位置，对应 V\$CACHEITEM 中的项目
2	SQL	VARCHAR(4096)	SQL 文本
3	LEN	INT	SQL 文本长度
4	HASH_VALUE	INT	HASH 值

13. V\$CAPTURE

显示捕获信息。

序号	列	数据类型	说明
1	STATE	INT	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	INT	捕获持续时间
7	FLUSH_BUF_NUM	INT	待刷盘缓冲数
8	FREE_BUF_NUM	INT	空闲缓冲数

14. V\$CIPHERS

显示系统加密算法信息。

序号	列	数据类型	说明
1	CYT_ID	INT	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

15. 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	END_TIME	DATETIME	结束时间
6	PAGE_FLUSHED	INTEGER	刷盘数
7	FIL	BIGINT	文件号
8	OFF	BIGINT	文件偏移量
9	CKPT_LSN	BIGINT	检查点 LSN

16. 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	INTEGER	命令从开始执行到执行结束花费的时间

17. 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	TYPES\$	VARCHAR(128)	连接类型
6	PROTOCOL_TYPE	TINYINT	协议类型
7	IP_ADDR	VARCHAR(23)	IP 地址

18. 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

19. V\$DATAFILE

显示数据文件信息。

序号	列	数据类型	说明
1	GROUP_ID	SMALLINT	所属的文件组 ID
2	ID	SMALLINT	数据库文件 ID
3	PATH	VARCHAR (256)	数据库文件路径
4	CREATE_TIME	TIMESTAMP(0)	数据库文件创建时间
5	STATUS\$	TINYINT	状态
6	RW_STATUS	TINYINT	读写状态：1 读，2 写。
7	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。
8	MODIFY_TIME	TIMESTAMP(0)	文件修改时间
9	MODIFY_TRX	BIGINT	修改事务
10	TOTAL_SIZE	BIGINT	总大小
11	FREE_SIZE	BIGINT	空闲大小
12	FREE_PAGE_NO	BIGINT	数据文件中连续空白页的起始页号
13	PAGES_READ	BIGINT	读页
14	PAGES_WRITE	BIGINT	写页
15	PAGE_SIZE	INT	页大小
16	READ_REQUEST	INT	读请求
17	WRITE_REQUEST	INT	写请求
18	MAX_SIZE	INT	文件最大大小，以 M 为单位
19	NEXT_SIZE	INT	文件每次扩展大小，以 M 为单位

20. 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：主机；2：备机
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

21. 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

22. 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	INT	字典对象的类型
2	ID	INT	字典对象 ID
3	NAME	VARCHAR(128)	字典对象的名称
4	SCHID	INT	字典对象所属模式
5	PID	INT	父 ID

23. 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	SMALLINT	缓冲池 ID
3	TOTAL_SIZE	INT	总大小
4	USED_SIZE	INT	实际使用大小
5	DICT_NUM	INT	字典对象总数

24. V\$EXP_BUFFERPOOL

显示扩展缓冲区信息，目前只有 DEFAULT BUFFERPOOL 才有扩展缓冲区。

序号	列	数据类型	说明
1	POOL_ID	SMALLINT	扩展缓冲区所在缓冲区 ID
2	POOL_NAME	VARCHAR(20)	扩展缓冲区所在缓冲区 NAME
3	PAGE_SIZE	SMALLINT	缓冲区页大小
4	N_PAGES	INTEGER	扩展缓冲区页数
5	BUFFER_START	BIGINT	扩展缓冲区起始地址
6	BUFFER_TAIL	BIGINT	扩展缓冲区尾地址

25. 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#	INT	回滚记录序号
7	OPERATION	CHAR(1)	回滚类型： 1:INS 2:UDP 3:DEL 4:UPD_INS 12:BATCH_INS 18:TRX_END
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

26. 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

27. 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	SVR_VERSION	VARCHAR (128)	服务器版本
4	DB_VERSION	VARCHAR (40)	数据库版本
5	START_TIME	TIMESTAMP	服务器启动时间
6	STATUS\$	VARCHAR(128)	系统状态
7	MODE\$	VARCHAR(128)	模式
8	OGUID	INT	控制文件的 OGUID

28. 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	SMALLINT	页面缓冲控制信息的文件组 ID
12	FILE_ID	SMALLINT	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

29. 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：字典锁、行锁、表锁
4	LMODE	CHAR(2)	锁模式：S 锁、X 锁、IX 锁、IS 锁
5	BLOCKED	INT	是否阻塞
6	TABLE_ID	INT	对应表锁、行锁 ID
7	ROW_ID	BIGINT	行锁的 ROWID

30. V\$PLSQL_DDL_HISTORY

记录 PLSQL 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 恶搞字符。
10	DDL_TIME	DATETIME	语句执行时间
11	MAIL_ID	BIGINT	邮件标识号

31. 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

32. V\$PROCESS

显示当前进程信息。

序号	列	数据类型	说明
1	PID	INT	进程 ID
2	PNAME	VARCHAR(256)	进程名
3	TRACE_NAME	VARCHAR(256)	日志路径
4	TYPE\$	BYTE	类型

33. V\$PSEG_ITEMS

显示当前回滚项信息。

序号	列	数据类型	说明
1	NTH_GRP	INT	序号
2	N_PAGES	INT	页数
3	START_POS	INT	开始位置
4	END_POS	INT	结束位置
5	N_USED_PAGES	INT	已使用页数

6	N_PURGED_PAGES	INT	已 PURGE 页数
7	N_ROLLOVER	INT	回卷的次数

34. V\$PSEG_PAGE_INFO

显示当前回滚页信息。

序号	列	数据类型	说明
1	FILE_ID	INT	文件号
2	PAGE_NO	INT	页号
3	TRX_ID	BIGINT	事物 ID
4	N_REC	INT	回滚记录个数
5	N_USED_BYTES	INT	页内已使用字节数

35. V\$PURGE

显示当前 PURGE 回滚段信息。

序号	列	数据类型	说明
1	N_FULL_EXTENT	BIGINT	FULL 簇长度
2	N_FRAG_EXTENT	VARCHAR(16)	FRAG 簇长度
3	MAX_PURGABLE_TRXID	BIGINT	最大回滚段事物 ID

36. V\$PURGE_RSEG_HISTORY

显示 PURGE 回滚段的历史信息（INSERT、UPDATE、DELETE 语句提交后产生）。

序号	列	数据类型	说明
1	SEQNO	INTEGER	编号
2	FLAG	CHAR	标记 为 P 或者 I
3	TRX_ID	BIGINT	事务 ID
4	RSEG_TIME	DATETIME	回滚时间
5	FADDR_FILE_ID	SMALLINT	段头的文件 ID
6	FADDR_FILE_NO	INTEGER	段头的页 ID
7	FADDR_OFFSET	SMALLINT	段头的偏移量
8	LAST_SEQ	INTEGER	最后一次回滚段的编号
9	INS_CNT	INTEGER	INSERT 类型回滚段数目
10	DEL_CNT	INTEGER	DELETE 类型回滚段数目
11	UPD_CNT	INTEGER	UPDATE 类型回滚段数目
12	UPD_INS_CNT	INTEGER	UPDATE_INSERT 类型回滚段数目
13	LINS_FILE_ID	SMALLINT	最后一次 INSERT 类型回滚段的文件 ID
14	LINS_PAGE_NO	INTEGER	最后一次 INSERT 类型回滚段的页 ID
15	LDEL_FILE_ID	SMALLINT	最后一次 DELETE 类型回滚段的文件 ID
16	LDEL_PAGE_NO	INTEGER	最后一次 DELETE 类型回滚段的页 ID
17	LUPD_FILE_ID	SMALLINT	最后一次 UPDATE 类型回滚段的文件 ID
18	LUPD_PAGE_NO	INTEGER	最后一次 UPDATE 类型回滚段的页 ID
19	LUPDINS_FILE_ID	SMALLINT	最后一次 UPDATE_INSERT 类型回滚段的文件 ID
20	LUPDINS_PAGE_NO	INTEGER	最后一次 UPDATE_INSERT 类型回滚段的页 ID

37. 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	ID	INT	表/用户 ID
2	SPACE_LIMIT	INT	空间限制大小，以页为单位

3	SPACE_USED	INT	空间实际使用大小，以页为单位
---	------------	-----	----------------

38. 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	INT	下一页页号
6	N_MAGIC	INT	本次运行，日志的 magic
7	DB_MAGIC	INT	数据库的 magic
8	FLUSH_PAGES	INT	Flush 链表中的总页数
9	FLUSHING_PAGES	INT	正在刷盘的总页数
10	CUR_FILE	SMALLINT	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	SMALLINT	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

39. 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	INT	日志 buffer 的总大小页数
4	N_PAGES	INT	Buffer 中剩下可用的页数

40. V\$CKPT

显示日志文件刷新的 SLOT 对象信息。

序号	列	数据类型	说明
1	CKPT_RLOG_SIZE	INT	检查点日志大小
2	CKPT_DIRTY_PAGES	INT	检查点脏页
3	CKPT_INTERVAL	INT	检查点间隔时间(s)
4	CKPT_FLUSH_RATE	INT	刷新级别
5	CKPT_FLUSH_PAGES	INT	管比例，至少刷新的页数
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 中

41. V\$RLOGFILE

显示日志文件的具体信息。包括文件号、完整路径、是否为 0 号文件、文件的状态、文件大小等等。

序号	列	数据类型	说明
1	GROUP_ID	BIGINT	文件组 ID
2	FILE_ID	SMALLINT	文件 ID
3	PATH	VARCHAR(256)	文件路径
4	CREATE_TIME	TIMESTAMP	创建时间
5	RLOG_SIZE	INT	文件大小

42. V\$RUNTIME_ERR_HISTORY

监控运行时错误历史。异常分为两种，一种是系统异常，用户没有捕获，由 vm_raise_runtime_error 产生。另一种是用户异常，用户捕获错误，并抛出自定义异常，由 nthrow_exec 产生，详见测试例子。

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 恶搞字符。
9	SU_FLAG	CHAR	U = 用户异常， S = 系统异常
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

43. V\$SESSIONS

显示会话的具体信息，如执行的 sql 语句、主机名、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
5	N_STMT	INTEGER	Stmt 的容量
6	SEQ_NO	INTEGER	序列号
7	CURR_SCH	VARCHAR(128)	当前模式
8	USER_NAME	VARCHAR(128)	当前用户
9	TRX_ID	BIGINT	事务 id
10	CREATE_TIME	DATETIME	会话创建时间
11	CLNT_TYPE	VARCHAR(10)	客户类型（4 种）
12	TIME_ZONE	VARCHAR(6)	时区
13	CHK_CONS	CHAR	是否忽略约束检查
14	CHK_IDENT	CHAR	是否忽略自增列指定列表检查检查

15	RDONLY	CHAR	是否是只读会话
16	INS_NULL	CHAR	列不存在 default 选项，是否插入空值
17	COMPILE_FLAG	CHAR	是否忽略编译选项，用于 view, procedure, trigger... 的编译
18	AUTO_CMT	CHAR	是否自动提交
19	DDL_AUTOCMT	CHAR	Ddl 语句是否自动提交
20	RS_FOR_QRY	CHAR	非查询语句生成结果集标记
21	CHK_NET	CHAR	是否检查网络
22	ISO_LEVEL	INTEGER	隔离级
23	CLNT_HOST	VARCHAR(128)	客户主机名
24	APPNAME	VARCHAR(128)	应用程序名
25	CLNT_IP	VARCHAR(128)	客户端 IP
26	OSNAME	VARCHAR(128)	客户端操作系统名
27	CONN_TYPE	VARCHAR(20)	连接类型

44. V\$SESSION_HISTORY

显示会话历史的记录信息，如主机名、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(10)	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)	客户主机名
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

45. V\$SQL_HISTORY

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	SESS_ID	BIGINT	会话 id
3	SESS_SEQ	INTEGER	会话序列号，用来唯一标识会话
4	TRX_ID	BIGINT	事务 id
5	THREAD_ID	BIGINT	线程 id
6	TOP_SQL_TEXT	VARCHAR(1000)	栈帧中第一个 sql
7	SEC_SQL_TEXT	VARCHAR(1000)	栈帧中第二个 sql
8	THRD_SQL_TEXT	VARCHAR(1000)	栈帧中第三个 sql
9	START_TIME	DATE_TIME	Sql 执行的起始时间

10	TIME_USED	BIGINT	Sql 执行所使用时间 (usec)
11	IS_OVER	CHAR	是否结束
12	EXEC_ID	INTEGER	Sql 执行 id
13	VM	BIGINT	执行 sql 的虚拟机
14	STKFRM	BIGINT	当前的栈帧
15	STK_LEVEL	INTEGER	当前栈帧的级别
16	BYTES_DYNAMIC_ALLOCED	BIGINT	动态分配字节数
17	BYTES_DYNAMIC_FREED	BIGINT	动态释放字节数
18	CUR_SQL_NODE	BIGINT	当前的 sql 节点
19	MAL_ID	BIGINT	邮件标识号

46. V\$SQL_NODE_HISTORY

通过该视图可以查询 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；

通过该视图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。

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	MAL_ID	BIGINT	邮件标识号

47. 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

48. V\$STKFRM

显示虚拟机栈帧信息。

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

49. 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, 如 select 或 update 等
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

50. V\$SYSSTAT

显示系统统计信息。

序号	列	数据类型	说明
1	ID	INT	统计对象 ID
2	CLASSID	INT	统计对象所属类别 ID 1:OS 2:CPU 3:MEM 4:IO 5:BUF 6:TRX 7:SQL 8:OTHER
3	NAME	VARCHAR(128)	统计对象名
4	STAT_VAL	BIGINT	统计值

51. V\$TABLESPACE

显示表空间信息, 不包括回滚表空间信息。

序号	ID	SMALLINT	表空间 ID
1	NAME	VARCHAR (128)	表空间名称
2	CACHE	VARCHAR (20)	CACHE 名
3	TYPE\$	TINYINT	文件组类型: 1 DB 类型, 2 临时文件组
4	STATUS\$	TINYINT	状态。1 ONLINE, 2 OFFLINE
5	MAX_SIZE	BIGINT	最大大小。0 代表只受操作系统限制
6	TOTAL_SIZE	BIGINT	总大小(以页为单位)

7	FILE_NUM	SMALLINT	包含的文件数
8	ID	SMALLINT	文件组 ID

52. 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(32)	线程名
3	START_TIME	DATETIME	线程开始时间

53. 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	NEXT_ID	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(10)	当前事务的状态
5	ISOLATION	SMALLINT	隔离级
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	RSEG_ID	BIGINT	数据库的回滚段
10	UREC_SEQNO	INTEGER	当前 Undo 记录的递增序列号
11	WAITING	BIGINT	事务等待的锁
12	DICTLOCK	BIGINT	字典缓冲区上的封锁

54. 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	INT	当前等待时间

55. V\$TRX_VIEW

该视图显示活动事务视图信息。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

56. V\$VERSION

显示版本信息，包容服务器版本号与 DB 版本号。

序号	列	数据类型	说明
1	BANNER	VARCHAR(80)	版本标识

57. 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	PL/SQL 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_AFFECED	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

58. V\$VPOOL

显示当前系统中各种内存池的信息。如数据字典缓冲池，session 池等。

序号	列	数据类型	说明
1	VPOOL_ID	INTEGER	内存池的 ID
2	BUFFER	BIGINT	内存池空间的地址
3	V_SIZE	BIGINT	内存池的大小
4	MAX_FREE	INTEGER	free_list 最大可能幂数
5	USED	BIGINT	当前已分配大小
6	IS_PUBLIC	CHAR	内存池是否是共用的
7	IS_DIRTY	CHAR	内存池是否使用了池外空间
8	V_DESC	VARCHAR(256)	内存池描述

59. V\$WAIT_CLASS

通过该动态视图可以按等待的类别查询等待的时间和次数。等待的类别有：

1. IO READING
2. ROW LOCK WAITING
3. TABLE LOCK WAITING
4. DICT LOCK WAITING
5. TASK WAITING
6. NETWORK WAITING
7. REDO LOG FLUSH WAITING
8. PAGE CONFLICT
9. BUFFER WAITING

通过查询该视图，可以得到系统中哪些资源等待的时间较长，哪些资源等待比较频繁，这样可以较好的提出相应措施，改善系统性能。

序号	列	数据类型	说明
1	CLASS_NO	SMALLINT	等待类别号
2	CLASS_NAME	VARCHAR(20)	等待类别的名字
3	TOTAL_WAITS	BIGINT	等待的次数
4	TIME_WAITED	BIGINT	等待的时间

60. 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	SMALLINT	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	SMALLINT	表空间号
8	FILE_ID	SMALLINT	文件号
9	PAGE_NO	INTEGER	页号

61. 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

62. 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

63. V\$DM_INI

实例配置信息。

序号	列	数据类型	说明
1	PARA_NAME	VARCHAR (128)	参数名称
2	PARA_VALUE	VARCHAR (256)	参数值

64. 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)	归档实例名
4	ARCH_FILE_SIZE	INT	归档文件大小
5	ARCH_SPACE_LIMIT	INT	归档文件的磁盘空间限制，单位是 MB
6	ARCH_TIMER_NAME	VARCHAR (128)	定时器名称
7	ARCH_IS_VALID	CHAR(1)	归档状态，是否有效

65. V\$DM_MAL_INI

MAL 参数信息。

序号	列	数据类型	说明
1	MAL_TYPE	VARCHAR (128)	MAL 类型
2	INST_NAME	VARCHAR (256)	站点名
3	IP	VARCHAR (30)	IP 地址
4	MAL_PORT	INT	端口号

66. 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

67. V\$DM_REP_MASTER_INFO_INI

数据复制主机参数信息。

序号	列	数据类型	说明
1	REP_ID	INT	复制 ID

68. 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	INT	复制 ID
2	MASTER_INSTNAME	VARCHAR (256)	主站点实例名

69. 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	INT	复制 ID
2	SRC_TAB_ID	INT	源表 ID
3	DST_TAB_ID	INT	目的表 ID

70. 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	INT	参数值
2	SRC_TAB_ID	INT	源表 ID
3	DST_TAB_ID	INT	目的表 ID
4	COL_ID	INT	列 ID

5	SQL_PL_TYPE	INT	Sql 类型
6	LEN	INT	Sql 长度
7	PREC	INT	游标类型

71. 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	INT	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

72. 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	INT	日志 ID
2	TAB_ID	INT	表 ID

73. 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_INTERVAL	INT	时间间隔
4	FREQ_SUB_INTERVAL	INT	月或周的间隔
5	FREQ_MINUTE_INTERVAL	INT	分钟间隔
6	START_TIME	TIME	开始时间
7	END_TIME	TIME	结束时间
8	DURING_START_DATE	DATE	开始日期
9	DURING_END_DATE	DATE	结束日期
10	NO_END_DATE_FLAG	INT	无结束日期标识
11	DESCRIBE	VARCHAR(256)	描述
12	IS_VALID	CHAR(1)	是否有效

74. V\$SYSTEMINFO

系统信息视图。

序号	列	数据类型	说明
1	N_CPU	INT	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	TOTLE_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	N_CPU	INT	CPU 个数
12	TOTAL_PHY_SIZE	BIGINT	物理内存总大小

75. V\$NETINFO

网络信息视图。

序号	列	数据类型	说明
1	RECV_BYTES	BIGINT	发送的字节数
2	SEND_BYTES	BIGINT	接收的字节数

76. V\$SEGMENTINFO

段信息视图。

序号	列	数据类型	说明
1	INDEX_ID	INT	索引 ID
2	SEG_ID	INT	段 ID
3	LOGIC_PAGE_NO	INT	逻辑页号
4	PHY_PAGE_NO	INT	物理页号
5	PHY_FIRST_PAGE_IN_EXTENT	INT	物理页所属的 extent 的第一个物理页
6	FIL_ID	INT	文件 ID
7	USE_BYTES	BIGINT	已使用的字节数
8	FREE_BYTES	BIGINT	空闲的字节数

77. V\$PARAMETER

INI 参数信息。

序号	字段	类型	说明
1	ID	INT	ID 号
2	NAME	VARCHAR(80)	参数名字
3	TYPE	VARCHAR(200)	参数类型
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

78. V\$MAL_SYS

MAIL 系统信息视图。

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	SMALLINT	mal 监听端口
5	N_SITE	SMALLINT	mal 配置的站点数目

6	MAL_NUM	INTEGER	mal 系统邮箱数目
7	MAL_SEQ_NO	SMALLINT	站点本身的 mal 序号
8	EMPTY_LET_NUM	INTEGER	空邮件数目

79. V\$MAL_INFO

MAIL 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	SMALLINT	创建 mal 的原始站点号
3	DEST_SITE_NO	SMALLINT	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	SMALLINT	等待状态.1:等待 0:非等待
11	MCPR_FLAG	SMALLINT	MAL 任 务 处 理 情 况.0:RESUME1:MPLN2:CACEL4:PAUSE

80. V\$MAL_LETTER_INFO

MAIL 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	SMALLINT	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	SMALLINT	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	SMALLINT	邮件的物理站点号
11	L_SRC_SITE	SMALLINT	邮件的逻辑站点号
12	MSG_LEN	INTEGER	信件的消息长度

81. 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	SUB_STATE	VARCHAR(128)	子系统状态
3	N_SITE	INTEGER	总站点数
4	N_COPYS	INTEGER	冗余数
5	N_ERR_SITE	INTEGER	故障站点数
6	SELF_EP_SEQNO	INTEGER	当前站点序号

82. 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
5	N_COPY_TO	INTEGER	冗余站点数
6	COPY_TO_ARR	VARCHAR(320)	冗余目标站点数组
7	N_COPY_FROM	INTEGER	被冗余站点数
8	COPY_FROM_ARR	VARCHAR(320)	被冗余站点数组

83. V\$MAL_SITE_INFO

MAIL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	SMALLINT	发送邮件的源站点
2	DEST_SITE_SEQ	SMALLINT	目标站点序号
3	MAL_PORT_NUM	SMALLINT	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

84. V\$DBLINK

数据库链接信息视图。

序号	列	数据类型	说明
1	ID	INTEGER	DBLINK 的 ID
2	DB_LINK	VARCHAR2(128)	DBLINK 的名字
3	SCH_ID	INTEGER	模式 ID
4	OWNER_ID	INTEGER	用户 ID
5	IS_PUBLIC	VARCHAR2(3)	是否为 PUBLIC
6	LOGIN_NAME	VARCHAR2(128)	登录名
7	HOST_NAME	VARCHAR2(128)	主机名
8	PORT_NUM	INTEGER	端口号
9	LOGGED_ON	VARCHAR2(3)	DBLINK 当前是否已链接
10	HETEROGENEOUS	VARCHAR2(3)	DBLINK 同步链接为 YES，异步链接为 NO
11	PROTOCOL	VARCHAR2(6)	DBLINK 通信协议
12	SITE	INTEGER	MAL 站点号
13	MPP_EXEC_ID	INTEGER	执行 ID

85. 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	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	VARCHAR2(128)	执行重要事件或行为的用户

附录 3 数据复制的 INI 文件及系统表

1. 数据复制配置文件信息

dmtimer.ini、dmllog.ini 和 dmrep.ini 配置文件信息见 2.1.4.3 节。

2. RPS 复制配置接口列表

1) 创建复制组

```
SP_RPS_ADD_GROUP(  
    GROUP_NAME VARCHAR(128),  
    GROUP_DESC VARCHAR(1000)  
);
```

参数:

GROUP_NAME 创建的复制组名称

GROUP_DESC 复制组描述

说明: 指示 RPS 创建一个新的复制组。如果已存在同名复制组则报错。

2) 删除复制组

```
SP_RPS_DROP_GROUP(  
    GROUP_NAME VARCHAR(128)  
);
```

参数:

GROUP_NAME 需删除的复制组名称

3) 创建复制关系

```
SP_RPS_ADD_REPLICATION(  
    GRP_NAME      VARCHAR(128),  
    REP_NAME      VARCHAR(128),  
    REP_DESC      VARCHAR(1000),  
    MINSTANCE     VARCHAR(128),  
    SINSTANCE     VARCHAR(128),  
    REP_TIMER     VARCHAR(128),  
    ARCH_PATH     VARCHAR(256)  
);
```

参数:

GRP_NAME, 复制组名,

REP_NAME, 复制名, 必须在 RPS 上唯一

REP_DESC, 复制描述

MINSTANCE, 主节点实例名, 必须在 RPS 的 MAL 中已配置

SINSTANCE, 从节点实例名, 必须在 RPS 的 MAL 中已配置

REP_TIMER, 复制定时器名。借助定时器, 可以设置复制数据的同步时机。如果是同步复制则为 NULL

ARCH_PATH, 主服务器上逻辑日志的完整归档路径。

4) 删除复制关系

```
SP_RPS_DROP_REPLICATION (  
    REP_NAME      VARCHAR(128)
```

```
);
```

参数:

REP_NAME, 复制关系名

5) 设置复制路径故障超时

```
SP_RPS_SET_ROUTE_FAULT_TIMEOUT (
    REP_NAME          VARCHAR(128),
    TIMEOUTS           INT
);
```

参数:

REP_NAME, 复制关系名

TIMEOUTS, 故障超时值, 以秒为单位。0 为立即超时

说明:

该接口用于设置复制路径故障处理策略。设置后, RPS 如检测到复制路径产生故障, 且故障持续超过设定的超时值后, 则需要取消故障的复制关系。

6) 设置复制节点故障超时

```
SP_RPS_SET_INST_FAULT_TIMEOUT (
    INST_NAME          VARCHAR(128),
    TIMEOUTS           INT
);
```

参数:

INST_NAME, 复制节点实例名

TIMEOUTS, 故障超时值, 以秒为单位。0 为立即超时

说明: 与复制路径故障超时设置相同, 如果超时, 则需要取消与节点相关的所有复制关系。

7) 设置复制关系的定时器

```
SP_RPS_ADD_TIMER(
    TIMER_NAME          VARCHAR(128),
    TIMER_DESC          VARCHAR(1000),
    TYPE$                INT,
    FERQ_INTERVAL        INT,
    FREQ_SUB_INTERVAL    INT,
    FREQ_MINUTE_INTERVAL INT,
    START_TIME           TIME,
    END_TIME              TIME,
    DURING_START_DATE    DATE,
    DURING_END_DATE      DATE,
    NO_END_DATA_FLAG     INT
);
```

参数:

TIMER_NAME, 定时器名

TIMER_DESC, 定时器描述

TYPE\$, 定时器类型, 取值如下:

- a) 执行一次
- b) 每日执行
- c) 每周执行
- d) 按月执行的第几天

- e) 按月执行的第一周
- f) 按月执行的第二周
- g) 按月执行的第三周
- h) 按月执行的第四周
- i) 按月执行的最后一周

FERQ_INTERVAL, 间隔天数

FREQ_SUB_INTERVAL, 间隔的月/周（调度类型决定）数

FREQ_MINUTE_INTERVAL, 间隔的分钟数

START_TIME, 开始时间

END_TIME, 结束时间

DURING_START_DATE, 开始日期

DURING_END_DATE, 结束日期

NO_END_DATA_FLAG 结束日期是否无效标识

8) 重新设置复制关系的定时器

```
SP_RPS_REP_RESET_TIMER(
    REP_NAME      VARCHAR(128),
    TIMER_NAME     VARCHAR(128)
);
```

参数:

REP_NAME, 复制名

TIMER_NAME, 新的定时器名

9) 添加复制映射

```
SP_RPS_ADD_TABLE_MAP(
    REP_NAME      VARCHAR(128),
    MTAB_SCHEMA   VARCHAR(128),
    MTAB_NAME     VARCHAR(128),
    STAB_SCHEMA   VARCHAR(128),
    STAB_NAME     VARCHAR(128)
);
```

参数:

REP_NAME, 复制关系名

MTAB_SCHEMA, 主表模式名

MTAB_NAME, 主表名

STAB_SCHEMA, 从表模式名

STAB_NAME, 从表名

10) 删除复制映射

```
SP_RPS_DROP_TABLE_MAP(
    REP_NAME      VARCHAR(128),
    MTAB_SCHEMA   VARCHAR(128),
    MTAB_NAME     VARCHAR(128),
    STAB_SCHEMA   VARCHAR(128),
    STAB_NAME     VARCHAR(128)
);
```

参数:

REP_NAME, 复制关系名

MTAB_SCHEMA, 主表模式名

MTAB_NAME, 主表名

STAB_SCHEMA, 从表模式名

STAB_NAME, 从表名

11) 开始复制设置

```
SP_RPS_SET_BEGIN(  
    GRP_NAME          VARCHAR(128),  
);
```

参数:

GRP_NAME, 复制组名

说明: 开始对指定复制组进行属性设置。创建/删除复制关系与创建/删除复制映射等接口都必须在此接口调用后执行, 否则会报错“错误的复制设置序列”。同一会话中也不能同时开始多个复制设置。

12) 提交复制设置

```
SP_RPS_SET_APPLY();
```

说明: 保存并提交本次设置的所有操作。如果需要继续设置, 则必须重新调用 SP_RPS_SET_BEGIN。

13) 放弃复制设置

```
SP_RPS_SET_CANCEL();
```

说明: 放弃本次设置的所有操作。如果需要重新设置, 则必须再次调用 SP_RPS_SET_BEGIN。

3. RPS 复制相关表

1) 复制组表

```
CREATE TABLE SYSREP.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.INSTANCE(  
    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.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_PATH     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
DESC$           VARCHAR(1000), // 复制描述
CLUSTER PRIMARY KEY(GRP_ID, REP_NAME)
);

```

4) 复制映射表

```

CREATE TABLE SYSREP.TABMAPS(
    REP_ID      INT,      // 复制 ID
    MSCH_NAME   VARCHAR(128), // 主表模式名
    MTAB_NAME   VARCHAR(128), // 主表名
    MTAB_ID     INT,      // 主表 ID
    SSCH_NAME   VARCHAR(128), // 从表模式名
    STAB_NAME   VARCHAR(128), // 从表名
    STAB_ID     INT,      // 从表 ID
    CLUSTER PRIMARY KEY(REP_ID, RT_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
(
    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
    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.TIMERS (
    REP_NAME     VARCHAR(128), // 复制名
    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 DATE, // 开始日期
    DURING_END_DATE DATE,   // 结束日期
    DESC$         VARCHAR(1000), // 定时器描述
    CLUSTER PRIMARY KEY(REP_NAME)
);

```

附录 4 预设角色权限列表

预设角色	预设数据库权限	是否允许转授
DBA	ALTER DATABASE	是
	BACKUP DATABASE	是
	CREATE USER	否
	ALTER USER	否
	DROP USER	否
	CREATE ROLE	是
	DROP ROLE	是
	ADMIN ANY ROLE	否
	ADMIN ANY DATABASE	否
	GRANT ANY OBJECT	否
	CREATE SCHEMA	是
	CREATE ANY SCHEMA	否
	DROP ANY SCHEMA	否
	CREATE TABLE	是
	CREATE ANY TABLE	否
	ALTER ANY TABLE	否
	DROP ANY TABLE	否
	INSERT TABLE	否
	INSERT ANY TABLE	否
	UPDATE TABLE	否
	UPDATE ANY TABLE	否
	DELETE TABLE	否
	DELETE ANY TABLE	否
	SELET TABLE	否
	SELECT ANY TABLE	否
	REFERENCE TABLE	否
	REFERENCE ANY TABLE	否
	GRANT TABLE	否
	GRANT ANY TABLE	否
	CREATE VIEW	是
	CREATE ANY VIEW	否
	ALTER ANY VIEW	否
	DROP ANY VIEW	否
	INSERT VIEW	否
	INSERT ANY VIEW	否
	UPDATE VIEW	否
	UPDATE ANY VIEW	否
	DELETE VIEW	否
	DELETE ANY VIEW	否
	SELECT VIEW	否
	SELECT ANY VIEW	否
	GRANT VIEW	否
	GRANT ANY VIEW	否

CREATE PROCEDURE	是
CREATE ANY PROCEDURE	否
DROP ANY PROCEDURE	否
EXECUTE PROCEDURE	否
EXECUTE ANY PROCEDURE	否
GRANT PROCEDURE	否
GRANT ANY PROCEDURE	否
CREATE SEQUENCE	是
CREATE ANY SEQUENCE	否
DROP ANY SEQUENCE	否
SELECT SEQUENCE	否
SELECT ANY SEQUENCE	否
GRANT SEQUENCE	否
GRANT ANY SEQUENCE	否
CREATE TRIGGER	是
CREATE ANY TRIGGER	否
ALTER ANY TRIGGER	否
DROP ANY TRIGGER	否
CREATE INDEX	是
CREATE ANY INDEX	否
ALTER ANY INDEX	否
DROP ANY INDEX	否
CREATE CONTEXT INDEX	是
CREATE ANY CONTEXT INDEX	否
ALTER ANY CONTEXT INDEX	否
DROP ANY CONTEXT INDEX	否
CREATE PACKAGE	是
CREATE ANY PACKAGE	否
DROP ANY PACKAGE	否
EXECUTE PACKAGE	否
EXECUTE ANY PACKAGE	否
GRANT PACKAGE	否
GRANT ANY PACKAGE	否
CREATE SYNONYM	是
CREATE ANY SYNONYM	否
DROP ANY SYNONYM	否
CREATE PUBLIC SYNONYM	是
DROP PUBLIC SYNONYM	是
CREATE REPLICATE	否
ALTER REPLICATE	否
DROP REPLICATE	否
ADMIN REPLAY	否
ADMIN BUFFER	否
CREATE TABLESPACE	是
DROP TABLESPACE	是

	CREATE ANY LINK	否
	DROP ANY LINK	否
RESOURCE	CREATE LINK	是
	CREATE SCHEMA	否
	CREATE TABLE	否
	INSERT TABLE	否
	UPDATE TABLE	否
	DELETE TABLE	否
	SELECT TABLE	否
	REFERENCE TABLE	否
	GRANT TABLE	否
	CREATE VIEW	否
	INSERT VIEW	否
	UPDATE VIEW	否
	DELETE VIEW	否
	SELECT VIEW	否
	GRANT VIEW	否
	CREATE PROCEDURE	否
	EXECUTE PROCEDURE	否
	GRANT PROCEDURE	否
	CREATE SEQUENCE	否
	SELECT SEQUENCE	否
	GRANT SEQUENCE	否
	CREATE TRIGGER	否
	CREATE INDEX	否
	CREATE CONTEXT INDEX	否
	CREATE PACKAGE	否
	EXECUTE PACKAGE	否
	GRANT PACKAGE	否
	CREATE SYNONYM	否
	CREATE PUBLIC SYNONYM	否
	CREATE TYPE	否
	CREATE LINK	否
PUBLIC	INSERT TABLE	否
	UPDATE TABLE	否
	DELETE TABLE	否
	SELECT TABLE	否
	REFERENCE TABLE	否
	GRANT TABLE	否
	INSERT VIEW	否
	UPDATE VIEW	否
	DELETE VIEW	否
	SELECT VIEW	否
	GRANT VIEW	否
	EXECUTE PROCEDURE	否
	GRANT PROCEDURE	否
	SELECT SEQUENCE	否
	GRANT SEQUENCE	否

	EXECUTE PACKAGE	否
	GRANT PACKAGE	否
DB_AUDIT_ADMIN	CREATE USER	否
	ALTER USER	否
	DROP USER	否
	AUDIT DATABASE	是
	SELECT ANY DICTIONARY	否
DB_AUDIT_OPER	AUDIT DATABASE	否
	SELECT ANY DICTIONARY	否
DB_AUDIT_PUBLIC	SELECT ANY DICTIONARY	否
DB_POLICY_ADMIN	CREATE USER	否
	ALTER USER	否
	DROP USER	否
	LABEL DATABASE	是
	SELECT ANY DICTIONARY	否
DB_POLICY_OPER	LABEL_DATABASE	否
	SELECT ANY DICTIONARY	否
DB_POLICY_PUBLIC	SELECT ANY DICTIONARY	否

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。

达梦数据库（武汉）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：400-648-9899

邮箱：tech@dameng.com