DBMaster

DBMaster 指南

SYSCOM

SYSCOM Computer Engineering Co./Corporate Headquarters

B1, 2-7F No. 115 Emei Street, Wanhua District,
Taipei City 108, Taiwan (R.O.C.)

www.dbmaker.com

www.dbmaker.com.tw/service

Document No.645049-237477/DBM54CN-M03312017-TUTO

发行日期: 2017-03-31

目录

1 简介

欢迎使用Syscom旗下产品。DBMaster是一个功能强大且使用灵活的SQL数据库管理系统(*DBMS*)，它支持交互式的结构化查询语言（SQL）、Microsoft开放式数据库连接(ODBC)标准接口，以及嵌入式的ESQL/C语言。DBMaster也支持Java工具的接口和针对COBOL语言的DCI接口。其独特的开放式架构及标准的ODBC接口可以令您自由地运用各种不同的编程工具来构建客户端应用程序，或者运用现有的ODBC适用程序来查询数据库。

DBMaster可以很容易地从个人使用的“单用户数据库”升级到企业级的“分布式数据库”。无论您的数据库是何种结构，DBMaster先进的安全性、完整性和可靠性都能保证用户重要数据的安全。另外，DBMaster的跨平台支持特性则在您需要作硬件升级时，提供给您最佳的扩展弹性。

DBMaster为您提供卓越的多媒体处理能力，可以让您存储、查询、恢复和处理各种类型的多媒体数据。利用DBMster提供的二进制大类型对象*(BLOBs)* 可以让您的多媒体数据完全享有DBMaster先进的安全性和灾难恢复功能。而利用文件对象(File Object)数据类型，也可以让您的应用程序能够直接编辑DBMaster数据库的外部文件。

本手册是针对不熟悉DBMaster的用户而设计的。它为初次使用DBMaster产品的用户提供了说明与具体运用的演示。本手册认为您对计算机操作也应该有一定的基础，并且应该熟练使用运行DBMaster的操作系统命令。有关操作系统的操作方法不在本手册范围之内。如果您遇到相关问题时，您必须参考相关的操作系统说明。

本手册会介绍运用DBMaster创建和维护数据库所必需的概念和原则等内容。这些内容列出了一个独立主题的一些要点，为了让您能够容易地了解这些内容，我们会以例子和图解来辅助说明。

本手册中SQL语言是DBMaster的执行语言，当初次出现命令时将提供相关命令的语法图。语法图包括可能的操作以及每一个命令行的语法变量。SQL命令的说明包括一些示例和使用该命令时的注意事项。

本手册中的大多概念、命令和示例都可以使用DBMatster提供的命令行工具dmSQL来实现。然而有些管理功能只能通过其他的DBMatster工具来操作。有关如何使用DBMatster工具操作的信息，请参考1.1章节的*其它相关文件*。

1.1 其它相关文件

除了本手册之外，我们还为您提供了DBMaster的其它用户手册和参考文献。要想获得有关DBMaster各方面的详细信息，请参考以下手册：

- 有关更多DBMaster数据库设计、管理、维护的信息，请参考*数据库管理员手册*。
- 有关数据库服务器管理的信息，请参考*服务器管理工具用户手册*。
- 有关DBMaster的配置信息，请参考*配置管理工具用户手册*。
- 有关更多DBMaster功能的信息，请参考*数据库管理工具用户手册*。
- 有关DBMaster COBOL接口的详细信息，请参考*DBMaster DCI用户手册*。
- 有关dmSQL中的SQL语言用法相关信息，请参考*SQL命令与函数参考手册*。
- 有关dmSQL命令行工具的信息，请参考*dmSQL使用手册*。
- 有关嵌入式ESQL/C语言的语法和使用，请参考*ESQL/C程序员参考手册*。
- 有关ODBC API和JDBC API的信息，请参考*ODBC程序员参考手册*和*JDBC程序员参考手册*。
- 有关错误信息和警告信息，请参考*错误信息参考手册*。

1.2 技术支持

在软件试用期间，Syscom Computer Engineering Co.(“Syscom”)将为您提供30天的免费email支持和电话支持。当软件注册后，我们还会再为您提供30天的免费技术支持。如此一来，您就可以获得60天的免费支持。不仅如此，在您购买软件后，Syscom对任何问题都会以email的方式为您提供技术支持。

您除了可以获得免费的技术支持外，还可以以20%的零售价购买其它产品。要想获得更多的详细资料和价格信息，请与sales@dbmaker.com.tw保持联系。

您可以通过任何一种方式(普通信件、电话或email)与Syscom技术支持保持联系，请登录至：http://www.dbmaker.com.tw/service 以获取详细信息。在与Syscom技术支持联系之前，请先查询当前数据库的常见问题解答。

无论您以何种方式与Syscom的技术支持联系时，请务必写上以下有效信息:

- 产品名和版本号
- 注册号
- 注册的用户名和地址
- 供应商/发行商地址
- 操作平台和计算机系统配置
- 错误发生前执行的动作
- 如果可以，请提供错误信息和编号
- 其它相关信息

1.3 文档协定

为方便用户的阅读和使用，本手册使用了一种标准的排版约定，注释、程序、示例和命令行都用缩进排版的方式进行了特别的设置。

协定	说明
斜体字	斜体字表示必须输入的信息占位符，例如用户名和表名。此字符可用实际的名称来替换。有时，文档也会使用斜体字来介绍新的关键字，强调着重点。
黑体字	黑体字表示文件名、数据库名、表名称、字段名、用户名和其它数据库对象。它也用于强调程序执行步骤中的菜单命令。
关键字	文字段落中， SQL语言使用的关键字都是以大写字母出现的。
小符号	文档中出现的小写字符表示键盘上的按键，两个键名之间的加号（+）表示在按住第一个键不放的同时，再按第二个键。两个键名之间的逗号（，）表示释放第一个键以后，再按第二个键。
注意	包含一些重要的信息。
程序	表示后面跟随的是程序的执行步骤或连续的项目。很多任务都是通过这种方式描述，给用户提供一个逻辑顺序步骤得以效仿。
示例	例子用来阐明描述，通常包括屏幕上出现的文本，用户也可以将这些例子输入到计算机中，通过屏幕看到运行结果。当然，示例还包括一些原型和语法。
命令行	包括文本，这些命令都可以输入计算机中，显示在屏幕上。通常用于显示SQL命令的输入输出或dmconfig.ini中的内容。

表 1-1文档协定

2 关系数据库管理系统基础

如果您对关系数据库的概念和原理或者SQL语言的使用不够熟悉，请在阅读DBMatster其他手册前先阅读本手册。

请通读本手册并建立向导数据库以完成每个章节所提供的示例。我们建议您动手完成本手册中出现的示例，在后面较长的示例中可能会产生意料之外的结果或错误。

如果您对数据库比较熟悉而仅仅想尝试某些章节的特定功能时，首先要建立相关章节的向导数据库，然后再运行手册中提供的脚本文件。

现今计算机系统最常见的任务之一就是存储和管理数据。这些数据包括：事件、图形、图片或多媒体。总之，数据库的作用就是对一个特定对象的信息收集。

在广泛使用计算机之前，信息通常以纸面的形式存储在文件夹中并整理入柜。当想要找到一些信息时，可能会先找到文件柜，找到相关文件后再查找所需信息。当存储的信息越多时，想在有限的时间内找到需要的数据就会变得相当困难。甚至记住在哪里可以找到所需信息都会变得复杂。

当人们第一次运用计算机存储信息时，是将这些信息以一种特定的、已知的格式存储在文件中。他们必须记住文件存储的位置，并且知道如何在文件夹中找到这些数据。

为了从文件中获取信息，必须先编写获取数据的特定程序。一旦这个程序存在，用户就可以迅速获取数据。然而，如果用户要改变数据存储的路径，或者想获取其它不同数据时，就需要编写一个新的程序。

公司信息系统部门的程序员通常会编写这些程序。一旦计算机上获取信息的数量增加时，对数据访问的途径要求也就会增加。用户编程需要大量数据文件的情况也相当普遍，若编写新程序将会延期数周甚至数月的时间。这就需要一个独立的数据存储系统和新方法来获取数据。

2.1 语法图

语法图列出了所有SQL命令的语法。当您使用命令行工具输入命令时，这些语法图会给您提供帮助，但是不能依赖于记住这些语法选项。请看下面语法图示例。

语法图的使用很简单，跟随图中的线条从箭头开始到结束即可。在这过程中不能被绕过的元素是在命令中必需的，可以绕过的元素是可选的。

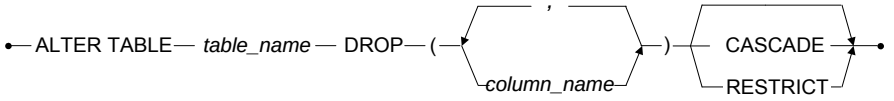


图 2-1 语法图示例

斜体字部份代表数据库对象的真实名称，而在语法图中，我们用这些斜体字占位符来代替应该填写的数据库对象的真实名称。如上图，用数据库的真实表名替换斜体字*<table_name>*。在实例数据库中，当您需要在**Customers**表上执行操作时，应该使用**Customers**来替换*<table_name>*。

注意语法图中的箭头，有时候一条命令会包含多个项目，这表现在语法图中将是一个循环。正如图中的循环路径所示：<column_name>区域可包括多个字段名，它们以逗号分隔。

2.2 关系型数据库管理系统的功能

关系型数据库管理系统用于管理存储在数据库中的数据，它通常作为记录保持系统来维护信息，以确保数据可以按照要求被获取。DBMaster以其强大的有效性和灵活性而优于其它信息系统。

一个典型的关系型数据库管理系统拥有以下诸多的特定功能：

- **数据模型**——必须拥有能够让用户轻松理解表达数据的方法。数据模型实际上是数学的一种抽象化。它确保了数据库中出现的所有数据都可以被用户获取或可视。
- **数据独立性**——在数据库结构中，确保数据不受任何物理存储变化的影响。即使数据库物理存储结构发生了变化，对特殊信息的处理也能够返回正确的结果。
- **高级语言支持**——数据库的信息应该能够通过高级语言获得。高级语言可以在对数据库存储结构不确定的情况下，允许用户定义、访问和操作数据。
- **事务管理**——提供一些方法确保对相同数据的多重处理，做到互不干扰，且允许多个用户同时使用一个数据库。
- **完整性控制**——确保数据库中的数据不存在无效值或者与相关数据不一致的情况。它防止用户意外获取无效数据或者违反数据从属性的操作。
- **访问控制**——当未被授权的用户访问数据库时，能够确保数据的安全与隐私。它可以防止未被授权的用户获取数据或者浏览一些敏感数据。
- **恢复方法**——在系统出现故障时，提供备份与恢复数据库的方法。

2.3 数据模型

数据在计算机中的物理存储方法对用户来说几乎没有意义。所有数据都是以二进制数的形式跨文件或者跨磁盘地存储在计算机中。RDBMS运用数据模型来代表已存储的数据，这种方式有利于用户轻松理解。

数据模型是一种抽象的数学方法，它提供了结构化的数据访问技巧，用来确保用户迅速操作以及被用户和应用程序恢复数据，而无需记住数据的存储位置和存储方法。

近年来有一些比较流行的数据模型，而关系数据库模型则是目前运用最为广泛的一种，DBMaster就是运用此种数据模型。关系型数据库模型通过由行和列构成的表将信息呈现给用户。每一行包含一个主题或项目的数据，每一列包含这些主题或项目的属性，例如：名称、大小、数量等。

2.4 数据的独立性

RDBMS最大的优势之一在于数据的独立性。数据的独立性允许数据库的结构发生改变时，无需应用程序或用户在访问数据时做出任何更改。数据的独立性有两种类型：*物理方式*和*逻辑方式*。

物理方式

早期的基于文件系统是以特殊的格式来存储文件中的所有信息。想要从文件中重新获取数据，了解这些文件格式的程序员必须编写相关程序。如果数据的结构发生改变，为了以适当的命令从变动后的结构中读取信息，程序也必须发生相应的变化。如果用户想要通过新的视图来访问数据，那么就必须编写新的程序。数据的组织方式和重新获取数据的技术实现方法构成了应用逻辑和代码，这种方式是数据独立的系统。

在RDBMS中，无需通过应用程序或改变用户读取数据的方式就可以更改数据库的物理结构。物理结构的改变将会影响应用程序的运行速度或效率，但不会破坏用户的应用程序，原因在于RDBMS使用抽象的数据模型使得数据库的结构对用户或者应用程序变得透明。数据从物理存储和磁盘读取方式转化为外界或逻辑视图使用的表现方法和读取实现技术。

如果物理存储结构发生变化，RDBMS就会发现这些变化并且仍旧提供相同的逻辑视图。因为逻辑视图代表外部存在的常数、应用程序以及用户的交互动作，这些交互动作是基于物理结构发生变化而数据本身未发生变化的逻辑视图之上的。因此，RDBMS具有数据物理结构的独立性。

逻辑方式

有时必须使数据的逻辑结构发生变化。只要现有数据的逻辑视图相同，那么对它们的交互和应用程序就不会产生影响。数据模型允许使用抽象特征，例如在基于文件的系统中来命名数据访问以替代其物理特征。由于增加了数据而不用改变数据项的抽象特征，因此也无需改变访问方法或技术。现有的程序和用户查询会导致运行无效，只有当使用新数据

时，现有的程序和查询才会被更改。因此，RDBMS具有*数据逻辑的独立性*。

2.5 高级语言支持

大多数数据库通常包括能够运用高级语言的能力。这些高级语言允许用户在对数据库存储结构不清楚的情况下，定义访问和操作数据。

高级查询语言可支持在数据库中通过*应用程序接口（API）*来访问数据库的信息，低级别的访问方法对创建自动的通用和重复任务的用户应用程序是非常有效的，然而它不允许执行一次性特殊查询（未计划或未预料的查询）。每当要作一次特殊查询时，用户必须书写相应的程序来执行相关查询，这样虽然会使用户得到一次锻炼的机会，但无疑也会增加用户的工作量。

高级语言使得此类特殊查询变得十分简单。大部分数据库支持的高级语言均使用类似英文的语法以方便学习。高级查询语言可以有效地执行数据库的任何功能。DBMaster使用当今社会的标准查询语言——*结构化查询语言(SQL)*。

2.6 事务管理

数据库管理系统用来存储大量信息并提供了多用户同时访问的功能。这些用户可以同时对数据进行操作；一些事务管理必须确保写入数据库的数据正确性。

什么是事务？

事务是指传统定义的一个逻辑工作单元，对数据库的一项或多项操作，必须在数据库状态一致的前提下来共同完成。对数据库的一个独立的操作可认为是一个独立的事务，这个事务要么必须成功完成并更改数据，要么失败并撤销对数据的操作。

多重操作可看作是一个独立的事务，假设数据库中存储着两种类型的信息，发送给用户的记录信息以及当前现存项的记录信息。当一项传送给客户时，发送列表中就会增加一条记录。这是数据库的一项操作。另外，发送项目的数量也必须从当前存储的项目中减去。

如果这些事务没有一起完成，数据库就会处于不一致的状态。它会导致现存项目的数量变多；传送的项目并没有从现存的项目中减去或者速度过慢；已经从现存中减去的项目并没有被传送。这些操作一起形成一个独立的事务，要么必须成功完成，要么事务将会失败。

如果事务成功完成并更改了数据，我们称之为事务提交。如果事务失败，数据不会发生更改，我们称之为*事务回滚*。

并发控制

在多用户同时访问数据时，如果发生延时将会导致执行效率下降。因此大多数数据库都支持并发处理，它允许多用户同时访问数据库。

多用户同时访问不同的数据不会出现上面的问题，但是当他们对相同的数据进行操作时就很容易出现问题。当两个用户同时对相同数据进行独立操作时，结果将无法预测。可能会出现事务读到旧数据，或者事务看似成功完成更改却可能会丢失的情况。

为防止以上类型事件的出现，事务都是*连续的*。两个事务并发执行的结果和它们按照先后顺序执行的结果相同，每位用户都可以很容易地访问数据库。有时一个事务必须等待另一个事务结束时才可以使用数据项。如果第二个事务没有等待就执行操作，那么得到的结果很可能是不准确的。

假设一个事务修改了一个数据项后继续执行其它操作，那么在执行其它操作时，第二个事务修改了相同的数据项后继续执行后面的操作。在任意一个事务提交前，第一个事务遇到错误后会回滚。RDBMS在事务发生前会返回到第一个事务执行前的数据库的状态，并且给数据项赋一个原始值。在第二个事务完成前，数据项中的这个值就会丢失。

为了使事务顺利完成并且防止访问数据库时的不协调情况，就需要并发控制。RDBMS中经常使用的并发控制方式就是*锁*。

锁的概念

数据项中的锁可以使RDBMS确保事务成功完成。当数据项被锁定时，就没有事务可以对其进行操作。在一个典型的多用户RDBMS中，这并不是一个有用的方法。

取而代之的是一个更复杂的模型：

- 存在不同类型的锁：*共享锁*和*互斥锁*。
- 存在不同级别的锁：*行锁*、*页锁*和*表锁*。

锁的类型

共享锁允许多个事务同时对一个数据项进行访问，但是事务不能对数据项进行修改。当多个事务要读取数据项的值时，并不会改变它。在这种情况下，多用户同时访问就可以实现，因为它们互不干扰。

当一个事务要修改数据项时，若允许其它事务对数据项进行修改则会使数据库中的数据前后不一致。而互斥锁则可以在其它事务访问数据项时保护数据项不被其他事务访问。在事务进行的同时，互斥锁会保证数据项在整个事务周期内保持稳定。

一个RDBMS也会使用不同级别的锁，它使用的目的更多是为了数据库性能而非并发控制。在相关RDBMS中，最小的数据项通常会有行锁。这些行构成页进而构成表。在RDBMS中，页和表可以作为一个单独项被锁定，锁定的所有数据项也包含在其中。

锁升级

如果一个事务访问了一个页中的许多行，那么获得所有独立锁的时间以及了解这些数据项所占用的资源将会非常多。使用页锁将会减少时间和资源的占用，但是耗费了对其它事务的并发控制。如果另一个事务想要在相同页中获取一个行锁，这是不允许的。这通常遵循性能优先原则。

当一个事务访问表中的许多页时将会发生类似的情况，使用表锁来代替页锁将会降低并发控制占用的时间和资源。在执行事务时可以通过手动方式对特定的数据项设置锁的级别。当RDBMS的性能需要提高时，可以使用*自动锁升级*，通过自动改变锁到更高级别使其保持可接受的并发级别。

2.7 完整性约束

通常认为数据库中包含的数据是正确的，这是数据库系统发展的主要原因，以保证在最短的时间内找到准确的数据。因此RDBMS必须有数据完整性约束的方式。完整性约束能确保数据的一致性和有效性。

不一致性将会导致数据冗余，例如在数据库中两个不同的地方存在相同的数据而RDBMS没有发现数据重复，对一个事务而言可能只有一个更新方法，之后RDBMS会给用户提供错误或者矛盾的信息，很显然数据库处在不一致状态。有完整性控制的RDBMS在遇到类似情况时将会产生错误。

在控制的过程中可能会保留冗余，RDBMS会注意所有访问途径，任何改变都会引起RDBMS的同步更新，这便是*级联更新*。数据库在进行更新时会允许暂时的状态不一致现象，但是RDBMS会保证在更新结束前用户无法获取数据。

确保数据有效是数据库的一项重要功能。薪水册数据库显示了一个员工一周工作了400小时而不是40个小时的无效数据。RDBMS无法判定此值本身的有效性，但是RDBMS的完整性约束功能允许数据库管理员限定和执行*完整性约束*。这些完整性约束会检查并确保当事务试图修改数据时，数据是有效的。

2.8 访问约束

RDBMS的集中化和多用户访问的功能要求有一些适当的安全控制形式，用以阻止未授权的访问以及限制权限的访问。安全约束大致可以划分为两类：用户授权以及事务授权。

用户授权

用户授权用来防止未授权用户的访问，通常进入系统需要用户名和密码。密码仅被用户和RDBMS掌握，且受到系统保护。但用户名和密码无法保证数据库的安全性。用户可以选择不易被破解并不含个人信息的资料作为密码，如配偶和宠物名字等。用户也不应当把密码保存在容易获得的地方，例如粘贴在电脑前。

事务权限

通常，一个数据库不会给用户相同的访问权。一些比较敏感的数据如员工工资只允许需要该数据的用户访问。在其它案例中，有的用户仅要求数据的读取权限，而有些用户则需要读取和更新数据的权限。

以销售点（POS）系统为例：商店的店员可能需要获得价格数据项，但是不允许更改价格。总公司的员工为了输入新的价格则需要读取和更改数据。

事务权限可以保护数据库，防止得到权限的用户在未被允许的情况下，有意或无意地访问数据项。RDBMS对于用户的权限均有记录，并且每当用户的事务试图访问数据时都会进行检查。如果用户没有访问权限，事务就会被拒绝。明确每位用户的权限是数据库管理员的职责所在。

2.9 RDBMS恢复

一个RDBMS可能会因为软件或硬件发生故障而受到损坏。故障通常分为两种类型：系统故障和介质故障。发生故障后可以进行恢复是RDBMS的主要优势之一，它相比基于文件系统更好。

系统故障

系统故障，又称作实例故障，是来自计算机系统的*内存故障*。造成的原因可能是突然断电、应用程序和操作系统错误、内存错误等，导致数据库管理系统的异常结束或事务的异常中止。最常见的保护方法就是使用事务日志。

事务日志记录了数据库操作的历史变化。在系统故障中事务过程的准确状态是不可靠的，当系统重启时它将不能被完成。RDBMS利用事务日志撤销已经写入磁盘的不正确的操作。

在系统发生故障之前，事务操作可能已经完成，但更改还没有被写入磁盘。在系统发生故障时，数据可能存储在RDBMS系统缓冲区内。这时，RDBMS利用事务日志重做或者*回滚*所有事务。

介质故障

介质故障发生在磁盘存储系统内，通常会引起磁盘的物理性损坏，例如单纯的读写头损坏、某段磁道损坏或者火灾、地震、水灾等天灾人祸。此时磁盘受到的损坏和丢失的数据是无法恢复的。然而当数据库提供存档或者数据监控时，还是可以被恢复的。

存档是对数据库进行定期备份，比如每晚都进行存档。它是从最后一次备份起保存发生的每一个文件事务的备份。当介质故障发生时，可使用备份来重建上次备份后的数据库。最后一次备份后的所有操作均会丢失。这种类型的存档适用于一些数据库系统，但对于网银或者航班预订系统则不适用。

数据监控包括连续不断地对整个数据库创建备份。在一个时间点上及时进行完整的数据库备份时，会创建一个事务日志的副本，数据库的所有改变都会被同时写入这两个日志文件中，相当于创建两个有效的数据库备份。如果在备份时出现介质故障，只有没有写入第二个日志的备份会遭到破坏，另一个日志中仍然会有记录，在数据库恢复的时候会给用户发送错误信息以提示数据的丢失。

无论使用哪种方法，存储备份文件的地址应不同于源数据库的地址，以确保成功恢复数据库。

3 关系数据库的架构

关系数据库的架构可以从两个方面来考虑，即逻辑架构和物理架构。逻辑架构负责数据的存储方式以及显示方式，而物理架构则注重构成一个关系数据库管理系统的各种组件。

3.1 关系数据库的逻辑架构

逻辑架构用以描述用户如何感知数据库中的数据。它所关心的只是数据看上去怎样，而并不关心关系数据库如何对数据进行处理和加工。存储在底层文件系统中的数据的存储方式对用户而言是透明的。用户在操作过程中不必担心数据在计算机中的存储位置和存储方式。这样可将数据库分成不同的抽象层次。

根据ANSI/SPARC研究组针对数据库管理系统提出的倡议，当今主流商业数据库管理系统都是基于通用*ANSI/SPARC*的体系结构，ANSI/SPARC体系结构将系统分为三级抽象层次：内模式或物理模式，模式或概念模式，外模式或视图模式。

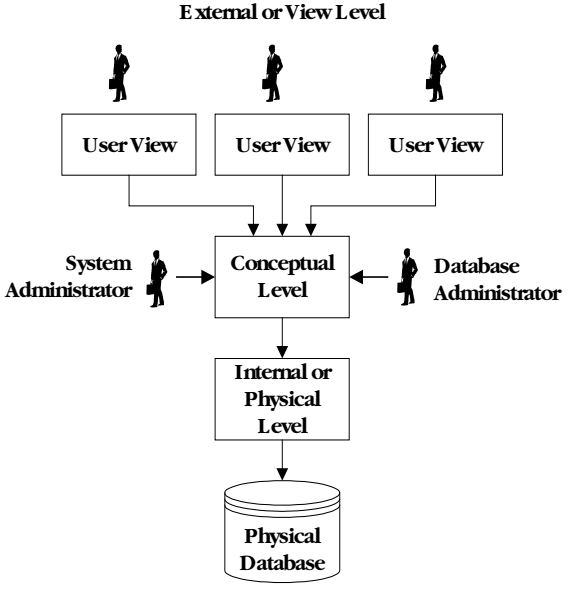


图 3-1: 典型关系数据库的逻辑结构

内模式或物理模式

物理级数据库将文件永久存储在二级存储设备中。物理模式或内模式最接近物理存储设备。它是物理数据库的底层描述，为操作系统的文件系统与高级抽象模式中的记录结构之间提供了接口。在这一层定义记录类型和存储方法，表示存储区域，存储记录的物理次序以及是否存在其它物理结构。

模式或概念模式

概念模式将整个数据库展现为一个统一的逻辑视图，从而使数据库中的所有数据看起来具有一致的风格。数据库设计中的第一步就是定义概念视图，关系数据库为此提供了数据定义语言。

模式允许关系数据库提供数据独立性。用数据定义语言创建模式，不能指定任何可以由物理模式操作的物理存储。除定义内容本身外，不能提供任何有关存储或访问的细节。

外模式或视图模式

外模式或视图模式是概念模式的一个子集，是数据库用户的数据视图，用户只能看见和访问所对应的外模式中的数据。这里的用户可以是一个程序也可以是最终使用者。一个数据库可以有多个外模式，彼此之间允许重叠交叉。

系统和数据库管理员是特殊情况。因为他们负责数据库的设计和维护，所以他们能够查看整个数据库。外模式和视图层次的功能刚好满足这两种用户的需求。

三级模式间的映射

数据库的三个抽象模式不是彼此孤立存在的。各模式之间有相互的映射。实际上有两种映射：模式/内模式映射和外模式/模式映射。

模式/内模式映射是模式层和内模式层的映射，定义了建立数据的逻辑结构(模式)与存储结构(内模式)间的对应关系。当数据库的存储结构发生

改变时，模式/内模式映射也要做相应的改变。这种映射保证了数据库中数据的物理独立性。

外模式/模式映射是外模式与概念模式间的映射，用来定义和建立某个外模式与模式间的对应关系。虽然两层模式有些类似，可是在某个外模式中的某些元素可能会与内模式中的有所不同。例如，多个字段被组合到一个虚拟的字段中，可以采用与原来字段不同的名称。如果数据库结构在模式层发生改变，那么只需改变外模式/模式映射，就可以使外模式保持不变。这种映射保证了数据库中数据的逻辑独立性。

也可能有另外一种映射，即将一个外部视图再以多个外部视图的方式表示，这可以称为外模式/外模式映射。如果存在多个外部视图都与另一个外部视图有密切的关系，这种映射是非常有用的。用户将不必一一建立这些类似的外模式/模式映射。

3.2 关系数据库的物理架构

物理架构描述访问和处理数据的各种组件以及这些组件之间的相互连接。在最低层，关系数据库的物理架构可以分为两部分：后端和前端。

后端负责管理物理级数据库，提供必要的支持以及在内模式、模式和外模式之间提供映射。关系型数据库的其它高级功能，如安全性、完整性和访问控制等都由后端完成。

前端就是运行在关系数据库上的一些程序。这些程序可以是由关系数据库厂商提供的，也可以是用户自己的或第三方的。用户只需在前端进行操作，甚至可能会感觉不到后端的存在。

还可以进一步将后端和前端细分为大多数关系型数据库系统常用的软件组件。

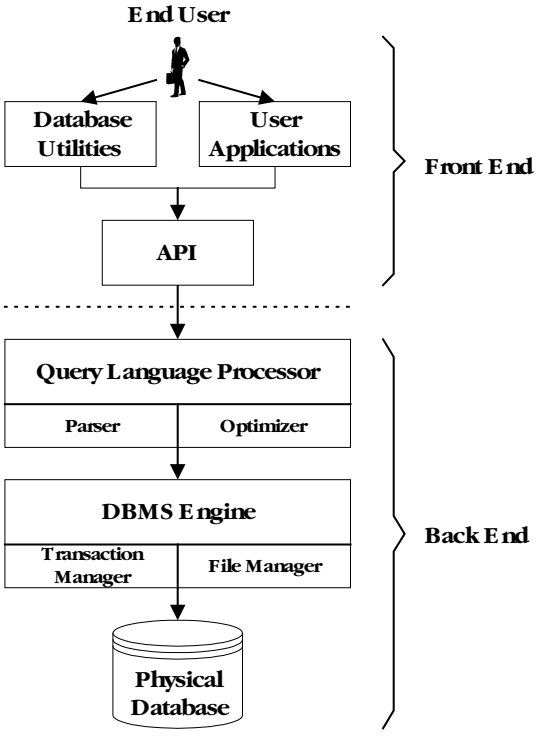


图 3-2:关系数据的常规功能和组件

应用程序和功能

对大多数用户来说，应用程序和功能是使用关系型数据库的主要接口。应用程序主要可分为三类：关系数据库厂商的、用户的和第三方的。

数据库厂商的应用程序

数据库自带的应用程序用来管理和维护数据库，所以用户无需自己编写程序就可以创建、操作数据库。 这些应用程序都是为常规的用途所创建的，对于特殊任务和重复性的工作，这些应用程序并不是最好的选择。

用户的应用程序

用户应用程序是用户针对特殊目的使用常见的编程语言编写的。编程语言通过API与关系数据库关系系统的查询语言相结合。这样用户可以在多种复杂的用户程序中使用关系数据库管理系统的查询语言功能。

第三方应用程序

第三方的应用程序可能与数据库厂商自带的某些增强功能的程序相类似，或者是应对数据库厂商自带的应用程序无法满足的一些需要。它们也可能与用户应用程序相似，为多数人都会用到的某些特殊用途而编写。

应用程序和功能列表

数据库最常用的应用程序和功能可明确地分为如下几类：

命令行工具

命令行工具直接使用关系数据库的查询语言功能，是基于字符的交互式界面。 通过命令行，可以对数据库进行操作、执行特别的查询等，并且能立即显示结果。 在不使用常规编程语言编写程序的情况下，命令行工具通常是利用数据库强大功能的唯一方法。 DBMaster的命令行工具是dmSQL。

图形化用户界面

图形化交互式界面隐藏了关系数据库和查询语言的复杂性，容易理解，方便使用。 通过图形化用户界面，新手可以不用学习查询语言就能对数据库进行访问；高级用户也省去了必须输入正确格式命令的麻烦，从而更快速地管理和操作数据库。因为图形化界面通常不可能实现所有的命令或选项，所以它所提供的功能并不像命令行工具一样完备和齐全。DBMaster包括三个主要的图形化工具：配置管理工具（JConfiguration Tool）、服务器管理工具（JServer Manager）和数据库管理工具（JDBA Tool）。

备份/恢复功能

通过备份/恢复功能，能够将数据库溃损的损失降到最低，以确保数据库能恢复到溃损发生时的一致性状态。手动备份/复制需要用户手动开启备份，自动功能则不需要用户的干预而定期自动备份数据库。适当的使用备份/复制功能可以使发生溃损的关系数据库恢复到正确而可靠的状态。

加载/卸载功能

使用加载/卸载功能，用户可以在一台机器上卸载整个数据库或数据库中的部分内容，卸载后的内容可以在同一台机器上重新加载或在远程的另一台机器上进行加载。在多数情况下这将是非常有用的，例如在某个特定的时间点及时创建数据库的备份拷贝，或者向一个新版本的数据库中加载数据，甚至可以向一个完全不同类型的数据库加载数据。加载/卸载功能也可以用来重新整理数据以提高性能，例如以特殊的方式集群数据或收回被废弃的数据所占的空间。

报告/分析功能

该功能用来对数据库中的数据进行分析报告。包括分析数据的倾向性、估算数值、或者显示满足特殊标准的数据，然后显示或打印出包含这些信息的报告。

应用程序接口 (API)

应用程序接口（API）就是一个底层程序库，它直接运行在数据库引擎上。通常用一般用途的编程语言编写软件程序时会使用到API，例如C++语言或VB语言。用户可以根据业务需要编写用户化的软件程序，而无需创建存储架构。数据库引擎负责处理数据的存储。输入一些特殊的分析或报告则需要用户程序来处理。

对每个关系数据库来说，API都是不同的。用一个关系数据库API编写的程序不能用在另一个关系数据库中。每一个API都有一个独特的函数调用，这些调用都与数据库操作紧密相连。即使两个数据库有相同的功能，它们使用的参数和函数也可能是不同的，这取决于数据库是如何设计的。不过微软的ODBC API是例外，任何支持ODBC的关系数据库都可以使用。

查询语言处理器

查询语言处理器负责接收查询语句并对语句进行转换，将其从具有英文语法的查询语言转换为关系数据库可以理解的形式。查询语言处理器通常由两部分组成：分析器和查询优化器。

分析器

分析器从应用程序或命令行工具中接收查询语句，并对语句语法进行检查以确保其正确性。分析器将一条语句拆分为基础的语法单元，检查确保每条语句都含有必要的组成结构。如果语句遵循语法规则，则传给查询优化器。

查询优化器

查询优化器将查询语句进行分析，选择最有效的方式执行查询。优化器会以不同的次序产生多个查询计划，然后计算哪个计划最优。在计算时，它会考查CPU时间、访问硬盘时间、网络通信时间、排序和扫描方法等。

关系数据库引擎

关系数据库引擎是关系数据库的核心，它负责对所有数据进行管理。关系数据库引擎通常包括两个独立的部分：事务管理和文件管理。

事务管理

事务管理维护授权表和并发控制。关系数据库通过授权表来确保事务管理器允许有权限的用户执行查询操作。授权表只能由授权用户通过命令来修改，而这些授权用户的命令也要对照授权表。数据库也会提供并发控制表来防止同时执行相互冲突的命令时发生冲突。关系数据库在执行查询语句前，要先查看并发控制表以确认另一个语句没有对其上锁。

文件管理

文件管理负责数据库的所有物理输入输出操作。它记录数据在磁盘上的物理地址，负责操作系统的任何插入、读取或写入操作。

4 数据库

通过DBMaster用户可以轻松地创建和管理数据库。广泛的跨平台支持和独特的开放式结构允许用户跨越多平台开展数据库的应用程序，并且当系统扩展时，能够轻松地将数据库移植到更大的系统中。

本手册主要指导用户使用命令行工具来实现管理数据库的功能。DBMaster同样提供图形化工具，可以简化数据库的管理操作。

本章您可以了解到：

- 如何选择一个有效的数据库名称
- 如何划分数据
- 如何使用dmSQL命令行工具
- 如何使用DBMaster其它图形工具对数据库进行日常管理
- 如何创建一个数据库
- 如何配制数据库
- 如何开启和终止一个数据库
- 如何连接和断开一个数据库

4.1 命名协定

当您安装DBMaster后，可以在同一台电脑上同时运行多个数据库。为了确定连接哪一个数据库，您需要对它们加以区分，DBMaster是通过对数据库命名来做到的。

DBMaster在**dmconfig.ini**配置文件中存储了所有本地或远程数据库的信息，所以给两个不同的数据库取相同的名称时会引起冲突。DBMaster无法确定配置文件对应的数据库是否正确，可能会给配置文件写入错误信息。数据库命名时，请务必检查**配置文件**的头部分以确定已有数据库的名称，并且选定一个新的唯一的名称。

在执行创建数据库命令前，请先仔细选择一个长度在1~128个字符之间的数据库名称。数据库名称可以包含字母、数字以及下划线，并且一旦创建，名称就不能再更改。

➲ **示例**

```
Tutorial
Parts_db
Region_1
1_Region
```

数据库名称不区分大小写。这意味着如果我们创建的数据库名称为**Tutorial**，那么当您使用**tutorial** 或者**TUTORIAL**时均为同一个数据库。本手册中使用的示例数据库就为**Tutorial**。

4.2 配置文件

DBMaster为每一个数据库存储**配置文件**，它包括数据库名称以及用户连接数据库的结构节。**配置文件**是一个有规律的ASCII文本文件，它可以通过文本编辑器来编辑。

DBMaster同样提供一个图形化工具——配置管理工具，以达到简化管理**配置文件**的目的。描述化的界面减少了需要了解DBMaster配置参数的时间并且可以将参数进行准确划分。

在多数情况下，当开启一个数据库时DBMaster会查看其配置信息。如果开启数据库后更改了配置文件，该更改在数据库下一次启动时才会生效。然而配置文件中的一些参数仅在连接数据库时才需要。您可以在连接数据库之前更改这些信息，下一次连接数据库时这些新的参数将会被使用。

这些配置参数对DBMaster的性能起到很重要的作用。您应当注意每一个配置参数对DBMaster产生的影响并且使用能够确保DBMaster顺畅运行的参数值。请参考*数据库管理员手册*以获取更多配置参数的详细信息以及控制它们的关键字。

创建

用户无需创建新的**配置文件**，因为DBMaster安装程序会自动创建一个相应的配置文件。当创建数据库时，DBMaster会检查**配置文件**并且找到结构段的名称以符合新的数据库。一旦找到匹配的配置文件段，紧接着就会检查配置选项的创建时间，并且当创建新的数据库时使用这些值。

用户在创建数据库之前应该通过文本编辑器创建数据库配置节，如果创建时间选项不是默认值，那么在创建数据库时才会生效。如果在创建数据库时DBMaster找不到**配置文件**段，它就会在搜索的第一个**配置文件**内自动创建**配置文件**段，或者在找不到配置文件时创建一个新的配置文件。当DBMaster创建一个新的配置文件节时，它会使用配置选项的默认值。总的来说，配置选项的默认值适合绝大多数数据库。

路径

在windows系统下，**配置文件**在windows路径下。当您使用**Windows**操作系统时，用户可以从开始栏里打开**配置文件**。点击开始按钮，选择*程序> DBMaster 5.4>DBMaster配置文件*。

在UNIX平台上，DBMaster可从以下三个地方来查找dmconfig.ini配置文件。

当启动数据库时，DBMaster会在下述路径中通过相关的配置节名称查找数据库**配置文件**：

1. 当前路径
2. 环境变量指定的路径
3. 安装目录(**~DBMaster\version**)

如果一个**配置文件**和节名称被找到，节中定义的关键字就会被使用。如果在文件中没有找到节名称，DBMaster会继续在下一个路径中寻找，直到成功找到为止。

格式

配置文件被划分成节，我们称这些节为数据库配置文件节，每一个数据库都有自己的配置文件节，其中设定的值控制着数据库配置文件的操作。

每个数据库配置文件节都是由节头和后面的一行或者多行关键字组成。节头部分数据库的名称在方括号内，关键字行由关键字和相关的取值组成。

➲ **示例**

```
[section_header_1]
keyword1 = value1                ;text following a semicolon is a comment
keyword2 = value2
         .
         .
[section_header_2]
keyword3 = value3 value4         ;spaces or commas may be used
keyword4 = value5                ;as delimiters between values
         .
         .
```

配置文件中的关键字不区分大小写，取值是否区分大小写取决于数据库运行的操作系统。

节名称

数据库开启时，会通过节名称来寻找数据库的相关配置文件并使用。节名称由方括号和数据库名称共同组成的，括号内需填入节的名称，左括号后写入的必须是数据库名称的第一个字符。

关键字

节名称后是一系列关键字及它们的取值，这些值在数据库开启时会被使用。通过“*关键字=取值*”语句给每个关键字赋值，如果一个关键字可以被赋予多个值，取值会被空格或者逗号隔开。这些取值可以是整型或字符型值。

如果DBMaster找不到**配置文件**，那么它将会使用默认值。根据需求选择关键字的使用是在数据库开启时还是连接时。完整的关键字列表和它们的取值范围请参考*数据库管理员手册*附录B。

注释

分号后的字符是注释的内容，是不被DBMaster执行的。您可以使用注释来提醒用户此关键字的用途，为什么您会选择一个特殊的值赋给关键字。如果想暂时更改关键字的取值，可以注释该关键字的初始值。

4.3 dmSQL

DmSQL是基于字符的，交互式的用户界面使您在使用DBMaster时可以有效地开发数据库。使用dmSQL来操作数据库、执行SQL查询可以立刻看到结果。DmSQL通常是开发数据库功能的最有效方法，而并不是用程序语言来创建程序。

开启

本手册中的示例大多使用dmSQL，您需要了解如何启动该程序，并且在使用之前熟悉该应用程序。后面的内容将会为您介绍如何为客户端/服务器端配置一个新的数据库。

WINDOWS

在Windows平台下，dmsqls和dmsqlc的功能被结合成一个独立程序－dmsql32.exe。

➲ **在Windows98 或WinNT下开启dmSQL命令行工具**

1. 点击**开始**按钮。
2. 选择**程序**－> DBMaster－> **dmSQL**。
3. 开启**dmSQL**应用程序。

UNIX

DBMaster的UNIX版本提供了单用户版本(dmsqls)和客户/服务器版本(dmsqlc)的dmSQL应用程序。在UNIX下使用dmsqls来创建单用户和客户/服务器数据库。

➲ **在UNIX下开启dmSQL命令行工具**

1. 命令行，类型：
   ```
   cd ~DBmaster/<current version>/bin
   ```
2. ***注意*** 替换DBMaster当前版本，例如：5.4。点击确认键。

3. 在命令行中敲入下列命令：
```
dmsqls
```
4. 点击确认键
5. DmSQL应用程序被开启。

工作空间

在开启dmSQL之后，Windows下dmSQL的工作平台就会出现，或者UNIX 系统下的dmSQL命令行界面就会出现。

图 4-1: Windows平台下的 dmSQL

- **标题栏**—标题栏显示了程序名称**dmSQL**以及**最小化**、**最大化**和**关闭**按钮。
- **菜单栏**－菜单栏显示了dmSQL下拉菜单栏的名称，每个菜单栏包括一系列相关命令。
- **工具栏**－工具栏是命令按钮的列表，许多常用的功能都罗列在下拉列表中。

- **命令输入区**－命令输入区是dmSQL输入命令、执行命令以及文本显示的区域。
- **状态栏**－状态栏列出了工作区域的当前状态以及当前时间。

4.4 JTools

DBMaster提供了三个基于Java、跨平台的图形化工具来管理数据库。这些工具易于使用，便捷的界面有利于用户迅速熟悉数据库的各项操作。每一个图形化工具都有自己的文档和相关内容的在线帮助。图形化工具可以在支持ODBC的任意操作系统下使用。下面将介绍这些工具和它们的功能。若想获取更多图形化工具的信息，请参考各工具的帮助文档。

配置管理工具

在4.2章中提及的“配置文件”，配置管理工具用来管理数据库的配置参数。配置管理工具提供描述性的界面使得用户无需记住配置文件中的关键字及取值，而通过图形界面对数据库进行配置。在Windows操作系统下，您可以通过点击开始>程序>DBmaster 5.4>配置管理工具来开启配置管理工具界面。有关于配置管理工具更多的内容，请参考*配置管理工具手册*或者工具中提供的帮助文档。

服务器管理工具

服务器管理工具是通过图形化界面来管理数据库的，如创建、开启、关闭、删除、备份以及恢复数据库等。如何使用服务器管理工具的内容并不包含在本手册中，但是有关于备份和恢复数据库的示例可参阅第九章“数据库恢复”。服务器管理工具便于操作，在Windows操作系统下，您可以通过点击*开始>程序>DBMaster 5.4>服务器管理工具*来开启它。有关于服务器管理工具更多的内容，请参考*服务器管理工具手册*或者工具中提供的帮助文档。

JDBA Tool

数据库管理工具为每个独立的数据库提供清晰的逻辑结构视图。数据库管理员可以创建、删除和更改对象。在Windows操作系统下，您可以通过点击*开始>程序>DBMaster 5.4>数据库管理工具*来开启它。有关于服务

器管理工具更多的内容，请参考*服务器管理工具手册*或者工具中提供的在线帮助文档。

4.5 创建数据库

创建数据库是DBMaster最基本的功能之一。为了创建示例数据库，您可以使用**CREATE DATABASE** 命令创建名为**Tutorial**的示例数据库。

示例数据库

➲ **使用dmSQL创建示例数据库**

1. 在dmSQL中键入以下命令：
```
CREATE DATABASE Tutorial;
```
2. 点击确认键.
3. 以下内容就会出现在屏幕中：
```
USE db #1 connected to db:<Tutorial> by user:<SYSADM>
```

在命令行工具中创建了一个名为**Tutorial**的空数据库。在创建数据库之前，DBMaster会在配置文件中查找是否存在Tutorial的节名。如果已经存在，DBMaster在新建数据库时会使用相关配置文件中的关键字取值。

示例中，在创建数据库前您无需创建配置文件，DBMaster会自动创建并且配置选项均使用创建时的默认值。

连接处理

使用dmSQL时一个数据库可以同时有八个连接。DBMaster使用USE项来指明当前连接的数据库。

USE是用来处理连接的。从USE#1到USE#8有八个连接。您的第一个数据库连接是USE#1、第二个是USE#2，以此类推直到USE#8。如果要在dmSQL中查看数据库当前的所有连接，可在命令行中键入USE即可。

USE命令

在执行这个命令后，dmSQL会显示数据库的所有当前连接。在示例中仅有一个链接，因此只有一个链接处理(USE#1)。这意味着数据库**Tutorial**只有一个处理USE#1，并且有一个名为**SYSADM**的用户连接数据库

➲ **使用dmSQL查看当前数据库的所有连接**

1. 在dmSQL命令行中键入USE
```
dmSQL> USE;
```
2. 点击确认键。
3. 将会显示：
```
dmSQL> USE;
USE db #1 connected to db:<TUTORIAL> by user:<SYSADM>(CURRENT)
```
4. 下面显示的是多个数据库连接：
```
dmSQL> use;
USE db #1 connected to db:<TUTORIAL> by user:<SYSADM>(CURRENT)
USE db #2 connected to db:<DBSAMPLE> by user:<SYSADM>
USE db #3 connected to db:<EXDM35> by user:<SYSADM>
```

示例中，在连接信息显示后您可以看到当前数据库**Tutorial**的连接。

通过连接操作连接一个不同于USE#1的数据库，可在连接之前更改USE#。如果要在dmSQL中更改连接，可在命令行工具中输入USE命令再写入连接数字即可。

➲ **使用dmSQL更改连接为2**

1. 键入以下命令：
```
dmSQL> USE 2;
```
2. 按确认键。

默认用户

当创建数据库时，用户自动使用默认的用户名**SYSADM**进行连接。**SYSADM**在数据库中拥有最高权限，可以创建新用户账户，并且拥有数据库对象的所有操作权限。但是由于您刚创建了数据库，所以数据库中并没有对象。

当您创建一个新的数据库时会自动开启单用户模式，同一时刻仅允许一个用户进行连接。DBMaster允许修改SYSADM密码（默认无密码），如果用户没有修改默认密码，那么任何用户都可以使用SYSADM账户连接数据库并且对数据库进行操作。如何修改SYSADM账户密码会在后面的章节进行描述。

如果允许其他用户连接数据库，那么需要在多用户模式下运行。在UNIX系统下使用单用户模式，Windows系统下可以使用多用户连接模式，或者在Windows和UNIX下均使用客户端/服务器模式。要使用多用户连接模式，则必须先关闭数据库再重新启动。在客户端/服务器模式下，需要先关闭数据库，然后在**配置文件**中添加其它关键字。

➲ **使用dmSQL关闭Tutorial数据库**

1. 在命令行中键入TERMINATE DATABASE：
```
TERMINATE DATABASE;
```
2. 点击确定键。

4.6 数据库模式

DBMaster允许您在不同的模式下启动数据库。每种模式在连接选项和访问数据库时均有所不同，通过多台计算机可以实现将数据库从一个计算机的单用户系统升级至分布式的多用户系统。

数据库模式的实现依赖于数据库服务器运行的平台，以及您想要实现的连接方式：单用户模式、多用户模式以及客户机/服务器模式。

单用户模式

单用户模式只能用于UNIX/Linux平台。对于非共享的数据库而言，此模式是DBMaster的一个简化版本。由于单用户数据库无需考虑锁、安全性和网络支持，所以此模式的优势在于小巧的应用程序和优越的执行速度。由于在此模式下只能同时建立一个连接，所以数据库无法运行别的服务器和后台程序，如备份服务器、复制服务器、全局事务处理服务器。再者，由于此模式下的数据库无法通过网络互联，所以用户只能通过主机来访问数据库。

多连接模式

多连接模式只能用于Windows平台。此模式的优势在于利用DBMaster的安全性和可靠性，数据库可以同时开启多个连接。但由于没有网络支持，所有连接都必须通过主机来访问数据库。此模式的局限性在于数据库不支持其它服务器和后台程序，如备份服务器、复制服务器或全局事务处理服务器。在Windows平台下除了不支持网络以外，使用多连接模式无需进行特殊的配置。

客户机/服务器模式

客户机/服务器模式支持所有平台。此模式允许从任何通过TCP/IP网络连接到主机的电脑端进行多个连接，并且支持DBMaster的安全性、可靠性和并发控制等特征。除此之外，还可以通过数据加密来提高网络传输的

安全性。此模式支持所有额外的服务器和后台处理程序，如备份服务器、复制服务器和全局事务处理服务器。客户机/服务器模式需要一些配置来保证数据库的运行顺畅，如果要运行一个数据库必须先连接到TCP/IP网络，并且只有通过TCP/IP协议的计算机才能够连接数据库。

修改配置文件

在修改**dmconfig.ini**配置文件后，在客户机/服务器模式下通过DBMaster服务器重启数据库，然后再连接到客户端应用程序，例如dmSQL命令行工具。

在**[Tutorial]**配置文件中增加下列内容：

```
[TUTORIAL]
DB_DbDir=C:\DBMAKER\TUTORIAL\DATABASE
DB_UsrId=SYSADM
DB_SvAdr=127.0.0.1
DB_PtNum=54321
```

DB_SvAdr

当数据库运行在客户机/服务器模式下时，则客户端和服务器端都需要设置此关键字。该关键字指定了计算机的IP地址或机器的主机名。请注意：如果您的操作系统使用的是域名系统(DNS)，您应该使用安装DBMaster的服务器端的DNS名称来替换IP地址。

DB_PtNum

当数据库运行在客户机/服务器模式下时，客户端和服务器端都需要设置此关键字。该关键字用来设定服务器端的TCP/IP端口号，对于数据库来说，客户端与服务器端的端口号必须匹配，否则连接会失败。

启动客户机/服务器模式的数据库

使用DBMaster服务器启动一个客户机/服务器数据库时，DBMaster服务器会先启动数据库并等待客户端的连接，以dmSQL命令行工具为例，当有一个客户连接后，它会接受命令并返回结果。只有拥有DBA或更高权限的用户才可以启动数据库。当运行DBMaster服务器时，要提供客户机/服务器数据库的名称、用户名以及密码。

您可以使用CONNECT命令来连接数据库，该命令适用于所有模式的数据库（单用户模式、多连接模式、客户机/服务器模式），但是当您使用的是客户机/服务器模式时，需要先启动数据库。CONNECT命令有三个参数：数据库名、用户名以及用户密码。目前使用SYSADM用户名时无需键入密码。

您可以在Windows系统下使用DISCONNECT命令来断开数据库的连接，在UNIX系统下使用TERMINATE DB命令来断开数据库连接。该命令同样适用于所有模式的数据库。此命令没有任何参数，仅通过现行的连接处理来断开数据库的连接。

Windows

➲ **在Windows下开启客户机/服务器数据库**

1. 点击**开始**按钮。
2. 选择**程序**>DBMaster，点击**DBMaster服务器**。
3. DBMaster启动服务器应用程序并显示**启动数据库**的对话框。
4. 在**数据库名称**框中选择**TUTORIAL**。
5. 在**用户名**框中键入**SYSADM**。
6. 在dmSQL命令行工具中键入以下命令：
```
dmSQL> CONNECT TO TUTORIAL SYSADM;
```
7. 点击确认健。
8. 如果要关闭数据库则可在dmSQL中键入如下命令：
```
dmSQL> DISCONNECT;
```
9. 点击确认健。

UNIX

➲ **在UNIX下开启dmServer**

1. 在命令行中键入如下命令：
```
$ cd ~DBMaster/<current version>/bin
```

注意 *替换当前DBMaster版本号。例如：5.4。*

2. 点击确认健。
3. 将当前路径更改为*~ dbmaster /version number/bin*
4. 在命令行中键入：
```
$ dmserver -u SYSADM TUTORIAL
```
5. 点击确认健。
6. DBMaster启动服务器并且运行**Tutorial**数据库。
7. 以下信息将会显示：
```
DBMaster (current version number)
Copyright 1995-2016 Syscom Computer Engineering Co. All rights reserved.
SQL Server bound to port 54321
The database has started successfully.
Database Server is running in the background mode.
Process ID = 28030
```
8. 在命令行中键入：
```
$ dmsqls
```
9. 点击确认健。
10. 开启**dmSQL**应用程序。
11. 在dmSQL中键入如下命令：
```
dmSQL> CONNECT TO TUTORIAL SYSADM;
```
12. 点击确认健。
13. 关闭数据库，在dmSQL中键入下面的命令：
```
dmSQL> TERMINATE DB;
```
14. 点击确认健。

5 表

尽管数据库已经存在并准备被使用，但是并没有地方可以存储数据。您可以将它设想为一个空的文件柜，里面没有文件夹，也没有地方存储信息。所以在数据库中，您必须创建表来存储信息。

在创建表之前要考虑存储的位置以及存储数据的类型。

5.1 表空间

DBMaster数据库可以分割成数个较小的逻辑空间，我们称这种逻辑空间为*表空间*。表空间是存储的逻辑区域，它将数据库分割成几个方便管理的存取空间，每一个表空间都包含一个或数个操作系统文件。建议您将数据放置在不同的磁盘上，这样在不同的物理磁盘上组合数据或分割数据时，可以提升访问速度。

表空间可以是固定大小也可以是自动扩展的。如果表空间的大小是固定的，那么这种表空间称为*固定表空间(Regular Tablespaces)*；如果表空间的大小可以自动扩展，那么这种表空间称为*自动扩展表空间(Autoextend Tablespaces)*。DBMaster还提供了*系统表空间(System Tablespace)*和*用户默认表空间(Default User tablespace)* 。

固定表空间

固定表空间是一个固定大小的表空间，并且包含一个或多个数据文件。如果固定表空间里的文件太小以至于不够存放所有数据，您可以用手动的方式来加大表空间的大小。每个固定表空间里最多可以包含32767个数据文件，数据库中所有文件的页总数不能超过8TB。固定表空间可以转换成自动扩展表空间。

自动扩展表空间

自动扩展表空间是一个视需要可自动延伸的表空间，其中的文件也可以自动延伸，DBMaster以插入文件的逆顺序来扩展它们。如果您不希望表空间扩展更大或者已经达到最大限制8TB时，自动扩展表空间可以转变成固定表空间。**dmconfig.ini**中定义了数据文件的页数，数据页数代表了自动扩展表空间的初始大小和固定表空间的实际大小。

系统表空间

所有DBMaster数据库都包括一个自动扩展的系统表空间。只要创建了数据库，DBMaster就会生成一个系统表空间来记录*系统目录表*。系统目录表由DBMaster来管理，并包括整个数据库的详细信息。其它表不允许存储到系统表空间中。

用户默认表空间

所有DBMaster数据库也包含一个自动扩展的默认表空间。当您创建一个数据库时，DBMaster会生成一个空的表空间来存储用户表。默认情况下，用户新建的表会自动存储在默认表空间里。当您创建的表要存储在另一个表空间时，必须指定表空间的路径。

临时表空间

临时表空间（TMPTABLESPACE）是可自动扩展的表空间，用于存储外部临时表（ETT）。临时表空间中包含两类文件：数据文件和BLOB文件，数据文件的逻辑名称为DB_TMPDB，物理名称为DB_TMPDIR/DBNAME.TDB；BOLB文件的逻辑名称为*DB_TMPBB*，物理名称为DB_TMPDIR/DBNAME.TBB。

当用户调用“create temporary table”或“select into”语句时，ETT将会自动生成并存储于临时表空间中。用户可以在临时表空间中创建临时表（当然系统将自动把ETT存储在临时表空间中），但不能在临时表空间中创建永久表。用户可以使用命令“ALTER TABLESPACE TMPTABLESPACE SET AUTOEXTEND OFF/ON;”和“ALTER DATAFILE *DB_TMPDB/ DB_TMPBB* ADD *n* PAGES;”，但不能将文件添加到临时表空间中或从临时表空间中删除文件。创建数据库时，临时表空间将会自动创建，启动数据库时，临时表空间的大小将被重置为默认大小。

- 用户只能在临时表空间中创建临时表。
- 用户不能在临时表空间中创建永久表。

- 用户不能添加文件到临时表空间或从临时表空间中删除文件。
- 用户不能删除临时表空间。

5.2 数据类型

在表中定义一个域时要选择数据类型。数据类型选择不当时会浪费大量的存储空间，或者在将数据类型转变为有用格式时，会使应用程序增加许多额外地步骤。DBMaster支持23种不同的数据类型。

BIGINT

BIGINT类型是一种精确的带正负号的数值类型，其精度和宽度分别为19位和0位。BIGINT类型数据的存储位为8个字节，其有效范围为9,223,372,036,854,775,807至−9,223,372,036,854,775,808。

如果您想将超过BIGINT或INTEGER最大有效值的数据转存为BIGINT或INTEGER类型，DBMaster会报出类型转换错误的信息，并对该操作不予执行。

➲ **示例 1**

```
37654
```

➲ **示例 2**

```
857823
```

BIGSERIAL (start)

DBMaster使用BIGSERIAL数据类型为数据库中的每一张表分配连续的整数并进行唯一识别。DBMaster能够内部自行管理这些整数，每个数字都是在每次使用DBMaster时自动产生的。

当为BIGSERIAL字段设定初始值时，就需要在定义字段时给可选参数START赋值，当START参数被忽略时初始值默认为***1***。每张表中只能有一个BIGSERIAL类型的字段。

产生的BIGSERIAL序列数是一个整型数据，这个整数是精确的数字数据类型，其精度和宽度分别为19和0位，占8个字节的存储空间，其取值范围是*−9,223,372,036,854,775,808~9,223,372,036,854,775,806*。

当您新增一笔记录时，把BIGSERIAL字段设为空值（NULL），那么DBMaster就会为新插入记录的SERIAL字段自动增加一个序列数，该序列值以1递增。

当您在新增记录中为BIGSERIAL字段赋了一个整数值，那么DBMaster不会再产生数字，而直接使用您输入的数值，同时系统中的内部序列数也不会递增1。如果您输入的整数值比在数据库中的序列数记数器的值还大，那么下次DBMaster自动产生的数字就会重设，从您新输入的值开始生成新的值。

➲ **示例 1**

```
10000, 10001, 10002, 10003, 10004, 10005, 10006, 10007
```

➲ **示例 2**

```
10000, 10001, 5000, 10002, 10003, 11000, 11001, 11002
```

BINARY (size)

BINARY类型是一种定长的数据类型，用于存储二进制数据。BINARY字段长度的取值范围是1～3992字节。当创建一个BINARY类型字段时，您可以输入size参数的值。如果录入的BINARY类型字段长度小于size，那么余下的部分将填充0。

像输入CHAR类型数据一样，在输入字符数据时，应在其前后加上单引号(‘ ’)。不过在BINARY类型字段中存储的不是您实际输入的字符，而是等价于该字符ASCII码的十六进制数。

您也可以直接输入十六进制数，当然也要加上单引号，并且还要在其后附加字母x(‘ ’x)以示其为十六进制数据。十六进制中每个字节需要两位数来表示，因此输入的十六进制数应是偶位数。

➲ **示例 1**

```
'AaBbCcDdEe'x
```

➲ **示例 2**

```
'41614262436344644565'x
```

CHAR (size)

CHAR类型是一种定长的数据类型，用于存储您从键盘输入的字符。CHAR类型字段的取值范围是1～3992字节。在定义CHAR类型字段时可输入参数size的值。

如果录入CHAR类型字段的数据长度小于size，那么余下的部分将以空格填充。输入CHAR类型数据时，必须在其前后加上单引号(' ')。每个双字节字符都占用2个字节的存储空间，所以在计算双字节字符所占用的存储空间时，需要考虑这一点。

➲ **示例 1**

```
 'This is a CHAR string.'
```

➲ **示例 2**

```
'This is another CHAR string.'
```

DATE

DATE类型数据有两种：DATE函数和DATE字段。前者表示当前日期，后者表示一个特定的日期。DATE类型是一个定长的数据类型，用来存储日期（年、月、日）。DATE类型数据所占的存储空间为4个字节，可以表示的年的取值范围为0001年—9999年。

DATE类型的数据有多种输入/输出格式。如果数据库中的日期数据显示不正确或无法输入有效的日期时，请检查它们的输入/输出格式是否正确。

➲ **示例 1a**

```
 '0001/01/01'
```

➲ **示例 1b**

```
'0001/01/01'd
```

➲ **示例 1c**

```
DATE '0001/01/01'
```

➲ **示例 2a**

```
'1999/12/31'
```

➲ **示例 2b**

```
'1999/12/31'd
```

➲ **示例 2c**

```
DATE '1999/12/31'
```

DECIMAL (NUMERIC)

DECIMAL类型是精确的带正负号的数值类型，它具有可变的精度和宽度。其中，精度指小数点前后数字的总位数，默认的精度值是***17***位，最大为38位。而宽度则指小数点后的位数，默认的宽度是***6***位。

在使用DECIMAL类型字段存储数据时，所需要的实际空间并不是根据默认精度和宽度或您在定义字段时所确定的精度和宽度而定，而是根据您所输入的值而定。

下面的公式可计算该类型数据所占用的存储空间：

$$\#\text{of bytes} = \frac{p+1}{2} + 2$$

例如，9283.83应占6个字节的存储空间。

其具体的计算为：

$$\begin{aligned}\#\text{of bytes} &= \frac{p+1}{2} + 2 \\ &= \frac{6+1}{2} + 2 \\ &= 5.5\end{aligned}$$

如果您想将超过DECIMAL类型最大值的数据（如FLOAT或DOUBLE类型）转存到DECIMAL类型字段中，那么DBMaster将显示一条类型转换错误信息，并对此操作不予执行。DECIMAL数据类型可以缩写为DEC。

➲ **示例 1**

```
3452.8373645
```

➲ **示例 2**

```
736.383732652
```

DOUBLE

DOUBLE类型是不精确的带符号数值的数据类型，尾数精度可到15位，其精度指尾数中小数点前后数字的总位数。DOUBLE类型的数据存储为8个字节，其有效输入范围为：1.0E308～–1.0E308。 能输入的最小值为：1.0E-308 和 –1.0E-308。

➲ **示例 1**

```
2.89837457884451E285
```

➲ **示例 2**

```
-1.93873634847372E-174
```

FILE

FILE类型是一种占48个字节的结构化数据类型，该数据类型类似于CLOB及BLOB类型，它将现存文件以及其它数据存储为DBMaster能连接的外部文件。DBMaster并不将这些数据存储为一个内部对象，而是存储为一个外部文件。这种存储方式使第三方工具在存取和操作数据库后，不用再将数据重新导入数据库来记录数据的改变。一个文件对象的路径长度不能超过255个字符。

FILE类型字段存储了系统表中的记录参考信息，系统表存储了数据库用于查找文件对象的信息。当查询FILE类型字段时，您并不能看到存储在该字段的真正内容，DBMaster给您显示的是存储在系统表中三个视图之一的相关信息，或文件名、文件大小和文件内容。

FILE类型有两种存储型态：系统文件对象和用户文件对象。系统文件对象可将现有文件拷贝到数据库文件对象的目录下，并指定一个唯一的名字，由数据库来管理这些文件，将没有记录连接的文件自动删除。当文件以原名字保存在原路径下时，用户文件对象将为该文件创建一个链接。因为这是用户创建的文件，所以当数据库没有记录连接到该文件时，它也不会被删除。在您将文件作为用户文件对象插入到数据库之前，DBMaster必须拥有该文件的读取权限。

当多条记录连接到同一文件时，DBMaster并不会产生额外的文件，所有记录都将共享同一个文件对象以节省磁盘空间。不过针对每个用户而

言，好像每条记录都连接到一个专用的文件对象。当更新一个共享文件时，DBMaster会产生一个新文件，同时，其它记录共享的文件也不会发生任何变动，其他用户仍然连接的是原始文件。这样在一条记录对文件进行修改时，并不会影响其它记录对文件的共享。

FLOAT

FLOAT类型是一种不精确的带正负号的数值数据类型，尾数精度可到15位，其精度指小数点前后数字的总位数。默认的FLOAT类型的数据存储为8个字节，其有效范围为：1.0E308～ –1.0E308。可以通过关键字DB_FLTDB将默认的FLOAT类型设置为REAL类型或DOUBLE类型。

能输入的最小值为：1.0E-308 and –1.0E-308。

➲ **示例 1**

```
2.89837457884451E285
```

➲ **示例 2**

```
-1.93873634847372E-174
```

INTEGER

INTEGER类型是一种精确的带正负号的数值数据类型，其精度和宽度分别为10位和0位。INTEGER类型数据的存储位为4个字节，其有效范围为2,147,483,647～-2,147,483,648。

如果您想将超过INTEGER最大有效值的数据（如DOUBLE等类型）转存为INTEGER类型，DBMaster会报出类型转换错误的信息，并对该操作不予执行。INTEGER数据类型可以简写成INT。

➲ **示例 1**

```
393848
```

➲ **示例 2**

```
-298376
```

JSONCOLS

JSONCOLS类型是一种动态字段集。DBMaster支持动态字段，但动态字段并不出现在表定义中。动态字段是源于JSON字符串的键值，且仅能在表中已有字段被声明为JSONCOLS类型的前提下才能使用。有关动态字段的详细信息，请参考*数据库管理员手册*中的*使用动态字段*章节。有关JSONCOLS字段的详细信息，请参考*数据库管理员手册*中*使用JSONCOLS类型*章节。动态字段在表中被存储为JSCONLS类型，该数据类型是LONG VARBINARY的派生类型。

➲ **示例 1**

创建一个包含JSONCOLS类型字段的表：

```
dmSQL> CREATE TABLE student(name CHAR(30), info JSONCOLS);
```

或：

```
dmSQL> CREATE TABLE student(name CHAR(30));
dmSQL> ALTER TABLE student ADD COLUMN info JSONCOLS;
```

使用JSONCOLS类型字段的字段名向表**student**中插入数据：

```
dmSQL> INSERT INTO student(name,info) VALUES
('jessia','{"desk_id":3,"birthday":"1986-09-19","score":90}');
1 rows inserted
dmSQL> INSERT INTO student(name,info) VALUES
('pine','{"desk_id":4,"birthday":"1987-03-03","score":95}');
1 rows inserted
```

使用“SELECT *”语句查询表**student**：

```
dmSQL> SET blobwidth 80;
dmSQL> SELECT * FROM student;
       NAME                             INFO
================= ================================================
jessia            {"score":90,"birthday":"1986-09-19","desk_id":3}
pine              {"score":95,"birthday":"1987-03-03","desk_id":4}
2 rows selected
```

使用JSONCOLS类型字段的字段名查询表**student**：

```
dmSQL> SELECT name, info FROM student;
```

```
       NAME                            INFO
================= ================================================
jessia            {"score":90,"birthday":"1986-09-19","desk_id":3}
pine              {"score":95,"birthday":"1987-03-03","desk_id":4}
2 rows selected
```

使用JSONCOLS类型字段的字段名更新表**student**中的数据：

```
dmSQL> UPDATE student SET info = '{"desk_id":7, "birthday":"1986-09-
19","score":88}' WHERE name='jessia';
1 rows updated
```

将字段**birthday**的数据类型更改为DATE类型：

```
dmSQL> ALTER TABLE student ADD DYNAMIC COLUMN birthday DATE;
dmSQL> SELECT info FROM student;
                                INFO
================================================================
{"score":88,"birthday":"1986-09-19","desk_id":7}
{"score":95,"birthday":"1987-03-03","desk_id":4}
2 rows selected
dmSQL> INSERT INTO student(name,desk_id,birthday,score) VALUES ('mike','8','1985-
02-15','92');
dmSQL> SELECT info FROM student;
                                INFO
================================================================
{"score":88,"birthday":"1986-09-19","desk_id":7}
{"score":95,"birthday":"1987-03-03","desk_id":4}
{"BIRTHDAY":477244800000,"DESK_ID":"8","SCORE":"92"}
3 rows selected
```

在JSONCOLS类型字段**info**上创建文本索引：

```
dmSQL> CREATE TEXT INDEX idx_stu ON student(INFO);
```

在JSONCOLS类型字段**info**上创建视图：

```
dmSQL> CREATE VIEW view1 AS SELECT info FROM student;
dmSQL> SELECT * FROM view1;
                                 INFO
```

```
================================================================
{"score":88,"birthday":"1986-09-19","desk_id":7}
{"score":95,"birthday":"1987-03-03","desk_id":4}
{"BIRTHDAY":477244800000,"DESK_ID":"8","SCORE":"92"}
3 rows selected
```

➲ **示例 2**

以下操作基于表**student**。有关表**student**详详细信息，请参考例1。使用动态字段的字段名向表**student**中插入数据：

```
/* implicit data conversion is closed by default */
dmSQL> INSERT INTO student(name,score) VALUES(?,?);
dmSQL/Val> 'demi','85';    /* it is ok */
1 rows inserted
dmSQL/Val> 'finly',82;     /* INT cannot be converted to CHAR */
ERROR (9629): value list syntax error
dmSQL/Val> END;
dmSQL> SET itcmd ON;
dmSQL> INSERT INTO student (name,score) VALUES(?,?);
dmSQL/Val> 'finly',82;     /* using implicit data conversion */
1 rows inserted
dmSQL/Val> END;
dmSQL> SET itcmd OFF;
dmSQL> INSERT INTO student(name,desk_id,birthday,score) VALUES('linda','1','1982-
01-01','91');
1 rows inserted
dmSQL> INSERT INTO student(name,desk_id,birthday,score) VALUES('glow','2','1984-
03-25','93');
1 rows inserted
dmSQL> INSERT INTO student(name,desk_id,birthday,score)
VALUES('kitty','abc','1980-02-27','97');
1 rows inserted
```

使用“SELECT *”语句查询表**student**：

```
dmSQL> SELECT * FROM student;
```

```
     NAME                          INFO
============= ============================================
jessia         {"score":88,"birthday":"1986-09-19","desk_id":7}
pine           {"score":95,"birthday":"1987-03-03","desk_id":4}
mike           {"BIRTHDAY":477244800000,"DESK_ID":"8","SCORE":"92"}
demi           {"SCORE":"85"}
finly          {"SCORE":"82"}
linda          {"BIRTHDAY":378662400000,"DESK_ID":"1","SCORE":"91"}
glow           {"BIRTHDAY":448992000000,"DESK_ID":"2","SCORE":"93"}
kitty          {"BIRTHDAY":320428800000,"DESK_ID":"abc","SCORE":"97"}
8 rows selected
```

使用动态字段的字段名查询表**student**：

```
dmSQL> SELECT name, desk_id, birthday, score FROM student;
     NAME         DESK_ID      BIRTHDAY              SCORE
=============  ===========  ============  =====================
jessia          7            19*           88
pine            4            19*           95
mike            8            19*           92
demi            NULL         NU*           85
finly           NULL         NU*           82
linda           1            19*           91
glow            2            19*           93
kitty          abc           19*           97
8 rows selected
```

使用动态字段的字段名更新或删除表**student**中的数据：

```
dmSQL> UPDATE student SET score='88' WHERE name='linda';
1 rows updated
dmSQL> DELETE FROM student WHERE desk_id='2';
1 rows deleted
```

为表中的动态字段添加描述信息：

```
dmSQL> ALTER TABLE student ADD DYNAMIC COLUMN desk_id int;
dmSQL> ALTER TABLE student ADD DYNAMIC COLUMN score DOUBLE;
```

向表**student**中插入数据：

```
dmSQL> INSERT INTO student(name, desk_id, age, score) VALUES('jane','12','1982-05-07',96);
ERROR (6150): [DBMaker] the insert/update value type is incompatible with column data type or compare/operand value is incompatible with column data type in expression/predicate
dmSQL> INSERT INTO student(name, desk_id, age, score) VALUES('jim',8,'1984-09-26',98);
1 rows inserted
dmSQL> SELECT name, desk_id, birthday, score FROM student;
     NAME         DESK_ID       BIRTHDAY            SCORE
==========  ============  ============  ======================
jessia                 7  1986-09-19    8.80000000000000e+001
pine                   4  1987-03-03    9.50000000000000e+001
mike                   8  1985-02-15    9.20000000000000e+001
demi                NULL     NULL       8.50000000000000e+001
finly               NULL     NULL       8.20000000000000e+001
linda                  1  1982-01-01    8.80000000000000e+001
kitty               NULL  1980-02-27    9.70000000000000e+001
jim                    8     NULL       9.80000000000000e+001
8 rows selected
```

更改动态字段**score**的数据类型：

```
dmSQL> ALTER TABLE student MODIFY DYNAMIC COLUMN score TYPE TO INT;
```

在动态字段**desk_id**上创建索引：

```
dmSQL> CREATE INDEX idx1 ON student(desk_id);
```

删除动态字段**birthday**的描述信息：

```
dmSQL> ALTER TABLE student DROP DYNAMIC COLUMN birthday;
```

LONG VARBINARY (BLOB)

LONG VARBINARY类型是一种变长的数据类型，用来存储任何二进制数据。每一个BLOB字段的最大长度不能超过8 T。与BINARY数据类型不同

的是，BLOB类型只在数据库中存储您输入的字节，而并不会以0来填充多余的空间。

与输入CHAR类型数据一样，在输入LONG VARBINARY 类型的数据时，您得在数据的前后加上单引号(‘ ’)。不过在LONG VARBINARY类型的字段中存储的不是您实际输入的字符，而是等价于该字符ASCII码的十六进制数据。

您也可以直接输入十六进制数，当然也要加上单引号，并且还要在后面附加字母x (‘ ’x)以示其为十六进制数据。十六进制中的每个字节用两位数来表示，因此在输入十六进制数时应是偶数位数。

➲ **示例 1**

```
'AaBbCcDdEe'x
```

➲ **示例 2**

```
'41614262436344644565'x
```

LONG VARCHAR (CLOB)

LONG VARCHAR类型是一种变长数据类型，用来存储任何从键盘输入的字符。LONG VARCHAR类型字段的最大长度不能超过8T。

与CHAR类型数据不同的是，LONG VARCHAR 类型只在数据库中存储您输入的字符，而不会以空格来填充多余的空间。当您在 LONG VARCHAR字段输入数据时，应在数据前后加上单引号(‘ ’)。每个双字节字符占用2个字节的存储空间，所以在您计算这些字段所占空间时，就应该考虑这一点。

➲ **示例 1**

```
'This is a varchar string.'
```

➲ **示例 2**

```
'This is another varchar string.'
```

NCHAR (size)

NCHAR类型是一种定长的数据类型，用来存储Unicode字符。在UTF16 Little-Endian(LE)编码制中，每个Unicode字符占两个字节。Size参数决

定了字段中字符的个数，所以必须在定义NCHAR字段时输入size参数，该参数值的取值范围是1～1996。

如果NCHAR类型字段的数据长度小于size，那么余下的部分将以空格填充。输入NCHAR类型数据时，应该在这些Unicode字符前后加上单引号(‘’)，并且还应加上前缀‘N’。

➲ **示例 1**

下面给出了Unicode数据的输入范例：

```
N'Unicode Data'
```

如果NCHAR数据是以十六进制的格式输入，那么应该在这些十六进制串前后加上单引号，并在后面附加字母‘u’。

➲ **示例 2**

下例表示以十六进制格式输入的Unicode数据，其中包含三个Unicode字符：

```
'610a620b63f1'u
```

当您在向Unicode字段录入数据而没有加前缀‘N’时，DBMaster会自动将该数据转换为Unicode类型。如果将Unicode字符输入到CHAR类型字段中，那么DBMaster会根据dmconfig.ini中的关键字**DB_LCode**值将输入的Unicode字符转换为相应的本地码，不能转换成本地码的字符将以□来表示。

NCHAR类型的同义字包括NATIONAL CHAR(size)和NATIONAL CHARACTER(size)。

NVARCHAR (size)

NVARCHAR类型是一种可变长的数据类型，用来存储Unicode字符。在UTF16 Little-Endian (LE)编码制中，每个Unicode字符占2个字节的存储空间。Size参数决定字段中的字符个数，所以必须在定义NVARCHAR字段时输入size参数，它的取值范围为1～1996。

如果NVARCHAR类型字段的数据小于size，那么余下的部分将不会以空格填充。输入NCHAR类型数据时，应该在这些Unicode字符的前后加上单引号(‘ ’)，并且加上前缀‘**N**’。

➲ **示例 1**

以下给出了Unicode字符的输入范例：

```
N'Unicode Data'
```

如果NVARCHAR数据以十六进制的格式输入，那么应该在这些十六进制串的前后加上单引号，并在后面附加字母‘**u**’。

➲ **示例 2**

下例表示以十六进制格式输入的Unicode数据，其中包含三个Unicode字符：

```
'610a620b63f1'u
```

如果您在往Unicode字段录入数据时没有加前缀**N**，那么DBMaster会自动将数据转换为Unicode类型。如果将Unicode字符输入到VARCHAR类型的字段，那么DBMaster会根据dmconfig.ini中的关键字**DB_LCode**值，将输入的Unicode字符转换成相应的本地码。不能转换成本地码的字符将以 □ 来表示。

NVARCHAR类型的同义字包括：NATIONAL CHAR VARYING(size)、NCHAR VARYING(size)、NATIONAL VARCHAR(size)和NATIONAL CHARACTER VARYING(size)。

OID

OID(object identifier)类型是一种特殊的数据类型，为存储在数据库中的对象、记录或BLOB分配一个唯一的ID。该结构化的数据类型的精度和宽度分别为10位和0位，并占据8个字节的存储空间。当插入记录时，DBMaster会自动为每条记录产生并插入一个对象编号。对象编号是由DBMaster在内部管理及维护的，用户不能直接使用它。

产生的OID值和数据库中对象的存储位置有关，这意味着两个连续生成的对象编号并不一定连续。

OID字段在表中是个隐藏字段，用SELECT * FROM CUSTOMERS这样的查询语句查询不到它的值。如果要查询OID字段，则应在查询语句中使用OID作为字段名。

尽管您可以使用对象编号来查询或更新表里的数据，但在使用SQL语句时，这种方式并不是通用的。对象编号一般用于内部编程接口，并不直接在交互式dmSQL环境中使用。

REAL

REAL类型是一种不精确的带符号的数值类型，它的尾数精度可到7位。精度是指尾数中小数点前后数字的总位数。REAL类型占用4字节的存储空间，它的有效范围为3.402823466E38 –3.402823466E38。最小的误差范围是1.175494351E-38 至 –1.175494351E-38。如果试图将一个大于最大有效值的数值，如DOUBLE类型的一个数值，转换为REAL类型，DBMaster将显示类型转换错误，并且转换失败。

➲ **示例 1**

```
3.583837E34
```

➲ **示例 2**

```
-1.873653E-21
```

SERIAL (start)

SERIAL类型是一种特殊的数据类型，用以产生一串序列数字。DBMaster为数据库中的每张表分配一个整数，并通过这些整数为相应的表生成一个唯一的序号。DBMaster能够内部自行管理和维护这些整数，每个整数被使用时，值自动递增1。

如果您想为SERIAL字段设定初始值，那么就需要在定义字段时给可选参数START赋值，否则系统将START字段的初始值默认为***1***。数据库中的每张表中只能有一个SERIAL类型的字段。

因为系统产生的序列数实际上是一个整型数据，所以SERIAL数据类型拥有整型（INTEGER）数据的所有特性。因此SERIAL类型和INTEGER类型一样，都是精确的带符号的数据类型，其精度和宽带分别为10和0位，

占4个字节的存储空间，其取值范围都是–2,147,483,646～2,147,483,646。

当您新增一笔记录时，把SERIAL字段设为空值（NULL），那么DBMaster就会为新插入记录的SERIAL字段自动增加一个序列数，该序列值以1递增。

如果您在新增记录中为SERIAL字段赋了一个整数值，那么DBMaster不会再产生数字，而直接使用您输入的数值，同时系统中的内部序列数也不会递增1。如果您输入的整数值比在数据库中的序列数记数器的值还大，那么下次DBMaster自动产生的数字就会重设，从您新输入的值开始生成新的值。

➲ **示例 1**

```
100, 101, 102, 103, 104, 105, 106, 107
```

➲ **示例 2**

```
100, 101, 50, 102, 103, 110, 111, 112
```

SMALLINT

SMALLINT类型是一种精确的带符号的数值数据类型，其精度和宽度分别为5位和0位。SMALLINT占2个字节的存储空间，取值范围为-32,768～32,767。

如果您想将超过SMALLINT最大有效值的数据（如INTEGER或DOUBLE等类型）转存为SMALLINT类型，那么DBMaster会报出类型转换错误信息，并对该操作不予执行。

➲ **示例 1**

```
4769
```

➲ **示例 2**

```
8376
```

TIME

TIME类型数据有两种：TIME函数和TIME字段。Time函数表示当前时间，其值随时间而不断改变；而TIME字段则用于存放某个固定时刻。这

两种时间类型都是定长数据类型，占4个字节的存储空间。除非您在输入时间值时，明确地标识了**AM**或**PM**，否则系统默认的时间值输入格式为24小时制。

这两种时间类型都有多种输入/输出格式。如果数据库中的时间数据不能正确显示，或者不能输入您认为有效的时间值，请检查它们的输入/输出格式是否正确。

➲ **示例 1a**

```
'22:04:05'
```

➲ **示例 1b**

```
'22:04:05't
```

➲ **示例 1c**

```
TIME '22:04:05'
```

➲ **示例 2a**

```
'10:04:05 PM'
```

➲ **示例 2b**

```
10:04:05 PM't
```

➲ **示例 2c**

```
TIME 10:04:05 PM'
```

TIMESTAMP

TIMESTAMP类型数据有两种：TIMESTAMP函数和TIMESTAMP字段。TIMESTAMP函数用于显示当前时间，其值随时间不断改变；而TIMESTAMP字段则用于显示某个固定时刻。

这两种TIMESTAMP类型都是定长的数据类型，可用来存储日期和时间数据，占11个字节的存储空间，精度和宽度分别是17和10位。有效的年的取值范围是从0001年到9999年。系统默认的时间值输入格式是24小时制，除非您在输入时间值的时候明确标识了**AM**或**PM**。

两种TIMESTAMP数据类型都按照TIME类型和DATE类型的输入输出格式来显示日期和时间。如果数据库中的时间数据不能正确显示，或者不能输入您认为是有效的值时，请检查它们的输入/输出格式是否正确。

➲ **示例 1a**
```
'1997/01/01 10:02:03'
```

➲ **示例 1b**
```
'1997/01/01 22:02:03'ts
```

➲ **示例 1c**
```
TIMESTAMP '1997/01/01 10:02:03'
```

➲ **示例 2a**
```
'01.01.1997 22:02:03'
```

➲ **示例 2b**
```
'01.01.1997 22:02:03'ts
```

➲ **示例 2c**
```
TIMESTAMP '01.01.1997 22:02:03'
```

VARCHAR (size)

VARCHAR类型是一种变长的数据类型，用来存储您从键盘上输入的字符。VARCHAR类型字段长度的有效范围为1～3992字符。在创建VARCHAR字段时，应为参数size赋值。

数据库中只存储您输入的字符。在输入VARCHAR字段值时应在字符串前后加上单引号(‘ ’)。每个双字节字符占用2个字节的存储空间，所以在您计算字段所占用的存储空间时，应考虑这一点。

➲ **示例 1**
```
' This is a VARCHAR string.'
```

➲ **示例 2**
```
' This is another VARCHAR string.'
```

Media Types

大型对象字段可以定义为媒体类型，这有助于媒体处理，例如在Microsoft Word文档中的全文搜索等功能。媒体类型包括：MsWordType、HtmlType、XmlType、MsPPTType、MsExcelType、

PDFType、MsWordFileType、HtmlFileType、XmlFileType、MsPPTFileType、MsExcelFileType以及PDFFileType。

媒体类型实际上是现有数据类型的派生类型。MsWordType、MsPPTType、MsExcelType、PDFType、HtmlType以及XmlType继承于LONG VARBINARY类型，HtmlType和XmlType继承于LONG VARCHAR，而MsWordFileType、HtmlFileType、XmlFileType、MsPPTFileType、MsExcelFileType以及PDFFileType继承于FILE类型。了解这一点对您使用ALTER TABLE来修改字段数据类型是很重要的。每种媒体类型数据的特性都继承其相应数据类型的特性。

XMLTYPE 的特性有：

- XML完整格式检查：插入/更新的 xml 内容必须是格式完全正确的。
- XML有效性：在创建xmltype字段时，任意指定一个有效的UDF，DBMaster使其xml内容有效。
- XML数据以原来的格式被保存。
- 使用 XPath 搜索查询：任意指定一个 xpath 并使用 extract 函数来查询/定位XML数据中的节点。
- 更新由XPath指定的XML内容。
- 在XPath extract上创建索引：通过对频繁查询的xpath表达式建立索引来加快查询速度。
- 不允许将xmltype字段或其他数据类型修改为xmltype。

➲ 示例

```
dmSQL> CREATE TABLE minutes (id INT, meeting_date DATE, doc MSWORDFILETYPE);
dmSQL> INSERT INTO minutes VALUES (1, 3/3/2003, 'c:\meeting\20030303.doc');
```

5.3 创建表

每张表都必须拥有表名称和字段，一张表可包含的字段数为1-2000。每一字段都包含字段名称和数据类型。有些数据的字段长度是默认的，所以并不需要特意指定，如INTEGER；有些需要指定它的精度(precision)和宽度，仅针对字段为DECIMAL的数据类型；有些需要指定数据的起始数字，如SERIAL的数据类型。

要创建一张表，必须提供表名称、字段定义以及所在表空间的名称。如果没有特别定义表空间，DBMaster会使用默认表空间来存放该表。

➲ **示例 1**

下例说明了如何创建表**Employee**：

```
dmSQL> CREATE TABLE Employee (Number SERIAL,    FirstName CHAR(15),
                              LastName CHAR(20), Manager INT,
                              Phone CHAR(15), HireDate DATE,
                              BirthDate DATE);
```

该命令创建的表**Employee**包含七个字段，分别为：**Number**、**FirstName**、**LastName**、**Manager**、**Phone**、**BirthDate**以及**HireDate**。其中**Number**字段使用SERIAL序号为员工编号，并且每新增一名员工时，员工序号会自动递增；**FirstName**、**LastName**以及**Phone**使用CHAR数据类型；**Phone**字段和姓名字段一样，都使用CHAR数据类型，因为电话号码可能会有括号、注释和时间段。最后两个字段**BirthDate**和**HireDate**使用DATE数据类型。

➲ **示例 2**

下例说明了如何在数据库**Tutorial**中创建多个表：

```
dmSQL> CREATE TABLE Regions (Number INT NOT NULL, Name CHAR(40))

dmSQL> CREATE TABLE IDTypes (Number INT NOT NULL, Type CHAR(50),
                             Description CHAR(255))

dmSQL> CREATE TABLE Customers (Number SERIAL, FirstName CHAR(15),
                               LastName CHAR(20), Phone CHAR(15),
                               IDType INT, IDRegion INT,
                               IDNumber CHAR(20), Credit SMALLINT)
```

```
dmSQL> CREATE TABLE Suppliers (Number SERIAL, Name CHAR (50),
                               Phone CHAR(15), Contact CHAR(35))

dmSQL> CREATE TABLE MovieTypes (Number INT NOT NULL, Type CHAR(30),
                                Description CHAR(255))

dmSQL> CREATE TABLE Movies (Number SERIAL,   Name CHAR(75),
                            Year CHAR(4),    Country CHAR(30),
                            Type1 INT,       Type2 INT,
                            Type3 INT,       Type4 INT,
                            Rating CHAR(10), Length SMALLINT,
                            Color CHAR(3),   BW CHAR(3),
                            Tape CHAR(3),    Disc CHAR(3),
                            Quantity SMALLINT, Supplier INT,
                            Data DATE,        Price FLOAT,
                            Rate SMALLINT)

dmSQL> CREATE TABLE Receipts (Number SERIAL, Customer INT,
                              Employee INT)

dmSQL> CREATE TABLE LineItems (Receipt INT NOT NULL, LineItem INT NOT NULL,
                               Movie INT,          Quantity SMALLINT,
                               Amount SMALLINT)
```

在上例创建的表中，有些字段被设置了关键字NOT NULL，这意味着在插入新的记录时，该字段必须已经包含数据且非空。创建表时还可以使用其它关键字。

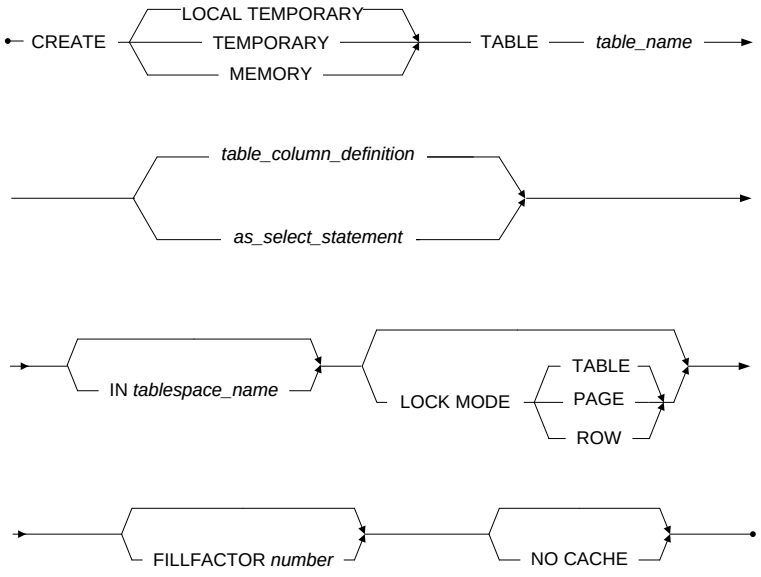


图 5-1: 创建表命令语法图

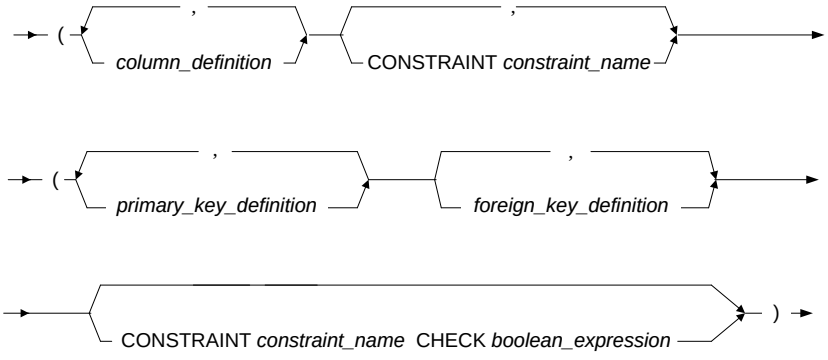


图 5-2: 创建表table_column_definition语法图

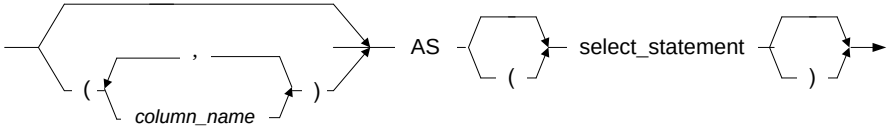


图 5-3: 创建表as_select_statement语法图

字段默认值

您可以为表中字段设定默认值，当插入一笔新记录而没有输入字段数据或字段被省略时，那么该字段将被自动赋予默认值。您可以对表中的每一字段设定默认值，如果您没有指定某个字段的默认值，那么该字段会自动预设为空值(NULL)。合法的字段默认值可以是常数、NULL或内建函数。有关内建函数的详细信息，请参考*SQL命令与函数参考手册*。

目前，DBMaster支持使用关键字USER、SYSTEM和ON UPDATE来设置插入和更新操作的默认字段属性。关键字USER/SYSTEM是可选项。

这些关键字可以指定用户是否能使用INSERT/UPDATE语句来修改带有默认值的字段的值。USER是默认的，关键字USER指定用户可以修改它的值，关键字SYSTEM指定用户不可修改它的值。关键字ON UPDATE也是可选项，该关键字指定更改其他字段的值时，带默认值的字段的值可以自动更新。这三个关键字主要用于表定义，有关更多表定义的信息请参考*SQL命令与函数参考手册*的*CREATE TABLE*、*ALTER TABLE ADD COLUMN*以及*ALTER TABLE MODIFY COLUMN*章节。

此外，当更新数据时，用户可以使用连接选项SYSTEM DEFAULT来指定带SYSTEM DEFAULT默认属性的字段的值是否被重写为默认值。如果将该选项设置为ON，则值会被更新为默认值；如果设置为OFF，则原始值会被更新为用户指定的值。该选项的默认设置为***ON***。另外，当使用INSERT/UPDATE语句将值分配给字段时，用户可以使用连接选项LOAD SYSTEM DEFAULT来指定加载数据库中的表时，带SYSTEM DEFAULT属性的字段是否被重写。如果将该选项设置为ON，则值会被更新为默认值；如果设置为OFF，则原始值会被更新为用户指定的值。该选项的默认设置为***OFF***。

➲ **示例 1**

您可以使用以下SQL命令来创建**tb_staff**表，其中字段**nation**的默认值为一个常数— 'R.O.C.'，而字段 **joinDate**的默认值为一个内建函数 **curdate()** 的返回值：

```
dmSQL> CREATE TABLE tb_staff (nation CHAR(20) DEFAULT 'R.O.C',
                                   ID INTEGER,
                                 name CHAR(30),
                             joinDate DATE DEFAULT CURDATE(),
                               height FLOAT,
                               degree VARCHAR(200),
                              picture LONG VARCHAR) IN ts_reg;
```

➲ **示例 2a**

```
dmSQL> CREATE TABLE computer(id INT, buy_time TIMESTAMP DEFAULT '2012-03-04
12:12:12', price int); //now attributes of buy_time is USER
```

```
dmSQL> INSERT INTO computer VALUES(1, '2012-10-10 10:10:20', 3400); //value of
buy_time will be replaced with '2012-10-10 10:10:20' which is specified by the
user
1 rows inserted
dmSQL> INSERT INTO computer VALUES(2, '2012-10-11 10:10:20', 5400);
1 rows inserted
dmSQL> SELECT * FROM computer;
    ID              BUY_TIME              PRICE
========== ========================== ==========
         1 2012-10-10 10:10:20               3400
         2 2012-10-11 10:10:20               5400
2 rows selected
dmSQL> UPDATE computer SET price=3200 WHERE id=1; //value of buy_time will not be
updated
1 rows updated
dmSQL> SELECT * FROM computer;
    ID              BUY_TIME              PRICE
========== ========================== ==========
         1 2012-10-10 10:10:20               3200
         2 2012-10-11 10:10:20               5400
2 rows selected
```

➲ **示例 2b**

```
dmSQL> ALTER TABLE computer MODIFY (buy_time TO buy_time TIMESTAMP DEFAULT '2012-
03-04 12:12:12' ON UPDATE); //now attributes of buy_time is USER and ON UPDATE
dmSQL> UPDATE computer SET price=3000 WHERE id=1; //value of buy_time will be
replaced with the default value'2012-03-04 12:12:12'
1 rows updated
dmSQL> SELECT * FROM computer;
    ID              BUY_TIME              PRICE
========== ========================== ==========
         1 2012-03-04 12:12:12               3000
         2 2012-10-11 10:10:20               5400
2 rows selected
```

```
dmSQL> UPDATE computer SET price=3000, buy_time='2012-10-10' WHERE id=1;//value of
buy_time will be replaced with '2012-10-10' which is specified by the user
1 rows updated
dmSQL> SELECT * FROM computer;
    ID              BUY_TIME             PRICE
========== ========================== ==========
         1 2012-10-10 00:00:00               3000
         2 2012-10-11 10:10:20               5400
2 rows selected
```

➲ **示例 2c**

```
dmSQL> ALTER TABLE computer MODIFY (buy_time TO buy_time TIMESTAMP SYSTEM DEFAULT
'2012-03-04 12:12:12'); //now attributes of buy_time is SYSTEM
dmSQL> INSERT INTO computer VALUES(3, '2012-11-10 10:10:20', 4700); //value of
buy_time will not be replaced with '2012-11-10 10:10:20' which is specified by the
user.
1 rows inserted
dmSQL> INSERT INTO computer VALUES(4, '2012-12-11 10:10:20', 2800);//value of
buy_time will not be replaced with '2012-12-11 10:10:20' which is specified by the
user.
1 rows inserted
dmSQL> SELECT * FROM computer;
    ID              BUY_TIME             PRICE
========== ========================== ==========
         1 2012-10-10 00:00:00               3000
         2 2012-10-11 10:10:20               5400
         3 2012-03-04 12:12:12               4700
         4 2012-03-04 12:12:12               2800
4 rows selected
dmSQL> UPDATE computer SET price=4500 WHERE id=3; //value of buy_time will not be
updated.
1 rows updated
dmSQL> SELECT * FROM computer;
    ID              BUY_TIME             PRICE
```

```
========== ========================= ==========
         1 2012-10-10 00:00:00               3000
         2 2012-10-11 10:10:20               5400
         3 2012-03-04 12:12:12               4500
         4 2012-03-04 12:12:12               2800
4 rows selected
```

➲ **示例 2d**

```
dmSQL> ALTER TABLE computer MODIFY (buy_time TO buy_time TIMESTAMP SYSTEM DEFAULT
'2012-03-04 12:12:12' ON UPDATE); //now attributes of buy_time is SYSTEM and ON
UPDATE
dmSQL> UPDATE computer SET price=4000, buy_time='2015-01-01' WHERE id=3; //value
of buy_time will be replaced with the default value'2012-03-04 12:12:12'
1 rows updated
dmSQL> SELECT * FROM computer;
    ID               BUY_TIME              PRICE
========== ========================= ==========
         1 2012-10-10 00:00:00               3000
         2 2012-10-11 10:10:20               5400
         3 2012-03-04 12:12:12               4000
         4 2012-03-04 12:12:12               2800
4 rows selected
```

锁模式

访问数据库时，DBMaster会自动识别对象的锁模式。DBMaster提供了三个层次的锁定模式：表锁定(TABLE)、页锁定（PAGE）和行锁定（ROW）。在创建表时，如果没有指定锁的模式，DBMaster会将表中的数据设定为页锁定（PAGE）模式。

如果将表设定为较高层次的锁模式 (例如表锁定)，那么数据库的并行存取能力将会降低，但锁定数据所需要的系统资源(共享内存)将会减少；如果将表设定为较低层次的锁定模式(例如行锁定)，那么数据库的并行存取能力将会提高，但是锁定所占用的系统资源(共享内存)就会增多。换句话说，如果您想在表锁定模式的表上新增或更改记录，除了您以外将没有

其他使用者可以同时存取该表中的数据。这是因为您在更新或新增数据时，数据库将会对整张表作一个互斥的锁定。

➲ **示例**

下例说明了如何在表上设定锁模式：

```
dmSQL> CREATE TABLE Suppliers (Number SERIAL,
                               Name CHAR (50),
                               Phone CHAR(15),
                               Contact CHAR(35) default 'Unknown')
                               LOCK MODE ROW;
```

填充因子（FILLFACTOR）

填充因子（FILLFACTOR）是用来指定新增数据时，数据页使用空间的上限（最大比例），利用填充因子可以在数据页上保留一定比例的空间。也就是说当您修改一笔记录时，如果数据页里还有足够的空间来储存这个更改的记录，就不用把这笔记录分割在不同的数据页里了。像这样把记录存放在同一个数据页里的好处在于：无需读取多个数据页就能存取一笔记录，数据存取的效率会有所提高。

➲ **示例**

下例是将**Suppliers**表的填充因子设为80%:

```
dmSQL> CREATE TABLE Suppliers (Number SERIAL,
                               Name CHAR (50),
                               Phone CHAR(15),
                               Contact CHAR(35) default 'Unknown')
                               LOCK MODE ROW FILLFACTOR 80;
```

在这种情况下如果使用的空间超过80%，那么新增的记录将无法插入到数据页中。填充因子(FILLFACTOR)的取值范围为50%-100%，默认值为100%。

取消快取

如果您需要经常读取一个非常大的表时，最好对这个表设定取消快取(NOCACHE)功能。DBMaster在存取数据库时，会在共享内存中分配数据页缓冲区来储存数据以免进行过多的磁盘存取(I/O)。 在对一个相当大

的表作扫描时，因为它拥有的数据页数大于数据页缓冲区数，所以会占尽所有的数据页缓冲区。

如果在建立表之前就知道这个表会储存大量的数据，我们可以设置这个表的取消快取功能，以后DBMaster对这个表做扫描时，就只会将一个数据页缓冲区来存放表数据。 这样数据页缓冲区就不会只被一个很大的表所占用。

➲ **示例**

设定取消快取(NOCACHE)选项：

```
dmSQL> CREATE TABLE Suppliers (Number SERIAL,
                               Name CHAR (50),
                               Phone CHAR(15),
                               Contact CHAR(35) default 'Unknown')
                               LOCK MODE ROW FILLFACTOR 80 NOCACHE;
```

临时表

您可以创建临时表来储存临时数据。临时表不会永久地储存在数据库中，只能存在于单个连接(session)中，当用户断开数据库的连接时，DBMaster会自动将临时表删除。临时表支持快速的数据操作并且只能被创建者使用。

➲ **示例**

创建一个名为**test**的临时表：

```
dmSQL> CREATE TEMPORARY TABLE test (Number SERIAL,
                                    Name CHAR(50),
                                    Phone DATE)
```

6 数据

当表和数据库模式设置好后，数据库就可以接收数据了。就像档案柜中已经有了文件和文件夹，但是里面却没有任何信息。

本章您将会了解到：

- 如何在数据库中通过增加记录或行来插入数据。
- 如何改变和更新已有记录中的数据。
- 如何查询表并找到表中数据。
- 如何删除表中不需要的数据。

6.1 插入

使用SQL语言有两种插入数据的方法。第一种使用标准的SQL语法，另一种使用主变量。通过主变量设置一条插入语句，当该命令执行时，数据插入的位置是未知的。DBMaster会设定数值状态，允许输入多条记录的数值。

➲ **示例 1**

使用SQL INSERT命令插入数值：

```
dmSQL> INSERT INTO Employee VALUES (10000, 'Gabriel', 'Davis', 10000, '228-6932',
'1/21/57', '4/24/95');
1 row inserted
```

Employee是表名，插入语句将**10000**、**Gabriel**、**Davis**、**10000**、**228-6932**、**1/21/57**以及**4/24/95**分别插入到对应的字段**Number**、**FirstName**、**LastName**、**Manager**、**Phone**、**BirthDate**以及**HireDate**中。

➲ **示例 2**

使用SQL INSERT命令对指定字段插入数值：

```
dmSQL> INSERT INTO Employee (FirstName, LastName, Manager) values ('Greg',
''Carter'', 10002);
1 row inserted
```

这条命令将值**Greg**、**Carter**以及**10002**分别插入到**FirstName**、**LastName**和**Manager**字段中。未指明的字段值均为**NULL**。

➲ **示例 3**

使用SQL INSERT命令对指定字段插入NULL：

```
dmSQL> INSERT INTO Employee VALUES (NULL, 'Dean', 'Cougar');
1 row inserted
```

上例中通过该语句在serial数列中插入了下一个SERIAL值，并将**Dean**和**Cougar**的值分别插入到**FirstName**和**LastName**字段中。在没有指定字段时，这些值会自动插入到相应字段的前三行。其它关键字也可能被用在INSERT命令中。

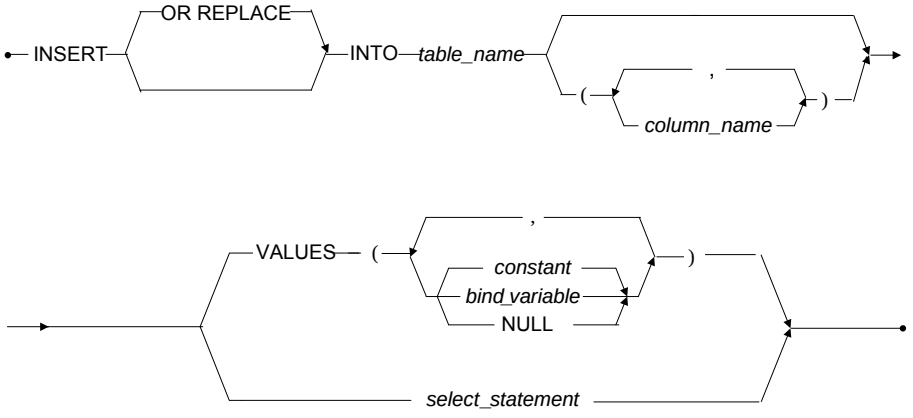


图6-1：INSERT语法图

OR REPLACE：OR REPLACE选项是可选的。当两条记录的某些字段值相同时，该选项用于确保DBMaster使用新记录来替换旧记录。也就是说，若用户插入该表的记录已经存在于表中（根据主键和唯一性索引判断），那么旧记录将会从表中删除，然后新记录插入到该表中。否则，新记录将直接插入到该表中。

有关INSERT命令的更详细信息，请参考*SQL命令与函数参考手册*。

使用主变量插入

使用主变量的INSERT语法与SQL标准语句相同。使用主变量的INSERT语句使得dmSQL命令行工具进入dmSQL/Val>屏幕提示状态。

➲ **示例 1**

下例说明如何使用主变量：

```
dmSQL> INSERT INTO Employee VALUES (?,?,?,?,?,?,?);

dmSQL/Val> NULL, 'Benson', 'Armstrong', 10002, '918-3517', '12/9/70', '3/2/93';
1 row inserted

dmSQL/Val> NULL, 'Lyn', 'Belger', 10000, '363-4511', '5/9/59', '12/6/91';
1 row inserted

dmSQL/Val> end;
```

➲ **示例 2**

下例说明如何使用等同的insert命令：

```
dmSQL> insert into Employee values (NULL, 'Benson', 'Armstrong', 10002, '918-
3517', '12/9/70', '3/2/93');

dmSQL> insert into Employee values (NULL, 'Lyn', 'Belger', 10000, '363-4511',
'5/9/59', '12/6/91');
```

➲ **示例 3**

下例说明如何使用insert和主变量来赋值：

```
dmSQL> insert into Employee values (NULL, ?, ?, 10002, ?, ?, ?);

dmSQL/Val> 'Murphy', 'Flaherty', '575-8846', '10/17/77', '11/17/90';
1 row inserted

dmSQL/Val> 'Taylor', 'Galbreath', '648-6633', '2/9/75', '10/22/94';
1 row inserted

dmSQL/Val> end;
```

➲ **示例 4**

下例说明如何使用等同的SQL插入命令：

```
dmSQL> insert into Employee values (NULL, 'Murphy', 'Flaherty', 10002, '575-8846',
'10/17/77', '11/17/90');

dmSQL> insert into Employee values (NULL, 'Taylor', 'Galbreath', 10002, '648-
6633', '2/9/75', '10/22/94');
```

不同的数据类型

DBMaster需要输入以下数据类型：

SMALLINT和INTEGER：

```
123, -252783
```

FLOAT和DOUBLE：

```
float:   30000.05, -234.56
double:  234.56e-257, 6.04E+23
```

CHAR和VARCHAR：

```
'Hello', 'DBMaster is a powerful database !'
```

BINARY：

dmSQL有两种识别二进制类型的方式：一种是十六进制格式，另一种是与CHAR类型相同的格式。

下例说明了二进制数'61'x代表了ASCII码的'a'值，相对应的十六进制为：

```
'0001020304'x, '3f2eff5c'x
```

在CHAR格式下，每个字符代表一个字节。但是该字符是以其ASCII值的形式存储在数据库中的。

CHAR：

```
'abcedf', '!@#$%'
```

在十六进制中，每两个十六进制代码代表一个字节。您可以使用十六进制的0-9、a-f (或A–F)来代表二进制数据。以十六进制格式表示的二进制数据必须被括在单引号内并且后跟字符x或X。

DATE和TIME：

```
'1994-12-20'd, '14:10:20't
```

DECIMAL：

```
 12.34, –0.123
```

插入Blob数据

DBMaster支持BLOB数据。这些数据类型是LONG VARCHAR和LONG VARBINARY，它们二者的区别与CHAR和BINARY的区别相同。下面举例说明插入BLOB数据的方法。

➲ **示例 1**

下例说明如何使用SQL命令来插入BLOB数据：

```
dmSQL> CREATE TABLE blob_table VALUES (lcharcol long varchar,
    2> lbincol long varbinary);

dmSQL> INSERT INTO blob_table ('this is blob data', '2d2d2d2d'x);

1 row inserted
```

➲ **示例 2**

下例说明用主变量插入BLOB数据：

```
dmSQL> INSERT INTO blob_table VALUES (?,'5f5f5f5f5f'x);
```

➲ **示例 3**

下例说明用主变量绑定BLOB数据：

```
dmSQL/Val> 'blob using host variable';

1 row inserted
```

或者您也可以从一个外部文件插入BLOB数据。

➲ **示例 4**

从一个外部文件插入BLOB数据，在插入模式下键入文件名：

```
dmSQL/Val> &comment.txt;

1 row inserted
```

注意 *comment.txt是当前路径下的一个文件。*

6.2 更新

数据被插入到数据库后，有可能需要改变一些数值，而SQL UPDATE命令就用于达到此目的。DBMaster提供三种更新字段数据的方法：标准的SQL、主变量以及OIDs。下图显示了更新语句的语法。

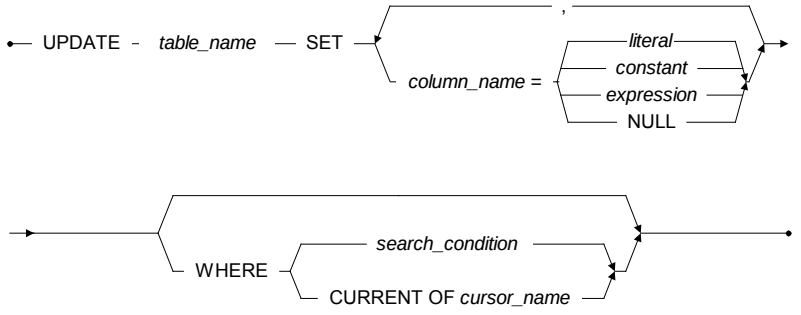


图6-2：UPDATE语法图

使用标准SQL更新

➲ **示例**

下例说明了如何使用标准SQL语句更新管理者文件：

```
dmSQL> update Employee set Manager = 10000 where LastName = 'Carter';

1 row updated
```

Employee是表名，该命令将管理者姓名更新为**Greg Carter**。

使用主变量更新

使用主变量的更新语句使得dmSQL进入dmSQL/Val>提示的赋值状态。您可以输入相关的主变量数据，使用END命令来结束赋值状态并完成更新语句。

➲ **示例 1**

下例说明了如何使用主变量更新**Employee**表：

```
dmSQL> update Employee set Phone = ? where FirstName = ? and LastName = ?;

dmSQL/Val> '736-8376', 'Gabriel', 'Davis';

1 row updated

dmSQL/Val> '837-7847', 'Lyn', 'Belger';

1 row updated

dmSQL/Val> end;
```

➲ **示例 2**

下例说明了如何使用标准SQL更新employee文件：

```
dmSQL> update Employee set Phone = '736-8376' where FirstName = 'Gabriel' and
LastName = 'Davis';

dmSQL> update Employee set Phone = '837-7847' where FirstName = 'Lyn' and LastName
= 'Belger';
```

上例更新了**Employee**表中，雇员Gabriel Davis和Lyn Belger的**Phone**字段。

使用OIDs更新

OID (object id)是一个特殊的二进制数据类型，它包含一个对象的ID记录。表的每一行都有唯一的OID，您可以在表中选择一个OID然后更新相应行的数据。OID被用在内部编程界面而不会直接在交互式的dmSQL环境界面使用。

➲ **示例**

下例说明如何使用OID更新**Employee**表：

```
dmSQL> select oid from Employee where FirstName = 'Dean' and LastName = 'Cougar';

      oid
================
0200000002000200
```

```
1 rows selected

dmSQL> update Employee set BirthDate = '12/8/70' where oid='0200000002000200'x;
```

上例更新了employee文件中，Dean Cougar的出生日期。

6.3 结果集

SELECT语句的输出结果就是一个结果集，结果集包括所有符合SELECT命令指定条件的数据。

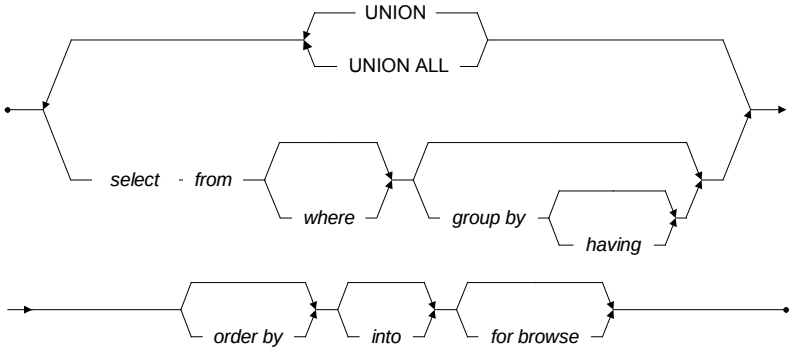


图 6-3: SELECT 语句语法

选择表

最简单的SELECT语句可以选择查看一个表中的所有信息。

➲ **示例 1**

下例说明如何查看表中所有信息：

```
select * from <table_name>
```

该命令是从指定的表中选择表所有字段的数据。星号（*）代表一个表中的所有字段。

使用下面这条命令显示**Employee**表中的所有数据。

➲ **示例 2**

下例说明了如何从**Employee**表中选择所有雇员的数据：

```
dmSQL> SELECT * from Employee;
```

下例说明了如何返回数据：

```
dmSQL> SELECT * from Employee;
```

```
 Number   FirstName   LastName        Manager     Phone    BirthDa* HireDate
======== =========== =============== ======== =========== ======== ========
   10000 Gabriel     Davis           10000    736-8376    1957-01* 1995-04*
   10001 Greg        Carter          10000    NULL        NULL     NULL
   10002 Dean        Cougar          NULL     NULL        1970-12* NULL
   10003 Benson      Armstrong       10002    918-3517    1970-12* 1993-03*
   10004 Lyn         Belger          10000    837-7847    2059-05* 1991-12*
   10005 Murphy      Flaherty        10002    575-8846    1977-10* 1990-11*
   10006 Taylor      Galbreath       10002    648-6633    1975-02* 1994-10*

7 rows selected
```

dmSQL会显示字段名称、每一个字段的所有数据以及显示的行数。

当您忘记字段名称时，无需对系统表进行操作，您可以使用该方法方便地查看表中所有字段的名称，并快速浏览表中所有字段的数据。

示例中您会发现并没有完整显示出所有字段的结果。dmSQL显示行的宽度默认是***80***，并且本手册也使用该格式的限定。

➲ **示例 *3***

下例说明如何关闭LINEWIDTH项并查看所有字段的数据：

```
dmSQL> SET LINEWIDTH off;
```

在SELECT语句中通过所有字段名来查看每个字段的所有数据。在**Employee**表中的字段名为：**Number**、**FirstName**、**LastName**、**Manager**、**Phone**、**BirthDate**以及**HireDate**。

➲ **示例 *4***

下例说明如何查看每个字段的全部数据：

```
dmSQL> SELECT Number, FirstName, LastName, Manager, Phone, BirthDate, HireDate
from Employee;
```

下例说明了如何返回数据：

```
dmSQL> SELECT * from Employee;
  Number     FirstName    LastName   Manager    Phone    BirthDate   HireDate
========== ========== ========== ========= ========= ========== ==========
     10000 Gabriel     Davis       10000     736-8376  1957-01-21  1995-04-24
     10001 Greg        Carter      10000     NULL      NULL        NULL
     10002 Dean        Cougar      NULL      NULL      1970-12-08  NULL
     10003 Benson      Armstrong   10002     918-3517  1970-12-09  1993-03-02
     10004 Lyn         Belger      10000     837-7847  2059-05-09  1991-12-06
     10005 Murphy      Flaherty    10002     575-8846  1977-10-17  1990-11-17
     10006 Taylor      Galbreath   10002     648-6633  1975-02-09  1994-10-22
```

```
7 rows selected
```

注意 *在数据库中使用的名称默认情况下是区分大小写的。*

如果输入的字段名称拼写错误或者大小写有误时，那么将会返回一个错误信息。

➲ **示例 5**

```
dmSQL> select numbe, FirstName, LastName, Manager, Phone, HireDate, BirthDate from
Employee;
ERROR (6523): invalid column name: NUMBE
```

只返回出现的第一个错误。在纠正一个错误后，如果还有其它错误出现，那么会继续返回其它错误，直至所有错误纠正为止。

如果输入的表名称有误，将会返回一个错误信息。

➲ **示例 6**

```
dmSQL> SELECT Number, FirstName, LastName, Manager, Phone, HireDate, BirthDate
FROM employee;
ERROR (6521): table or view does not exist: SYSADM.employee
```

选择字段

您可以通过指定字段名称来查询表中的相关字段。

➲ **示例 1**

下例说明了如何指定字段：

```
dmSQL> SELECT <column_name>, <column_name>, ... FROM <table_name>
```

该命令用于选择表中指定字段的数据。

➲ **示例 2**

下例说明了如何查看Employees表中的FirstName, LastName以及Phone字段：

```
dmSQL> SELECT FirstName, LastName, Phone FROM Employee;
```

以下是返回的结果：

```
dmSQL>  SELECT FirstName, LastName, Phone FROM Employee;

   FirstName          LastName           Phone
```

```
============== ==================== ==============
Gabriel        Davis                736-8376
Greg           Carter               NULL
Dean           Cougar               NULL
Benson         Armstrong            918-3517
Lyn            Belger               837-7847
Murphy         Flaherty             575-8846
Taylor         Galbreath            648-6633

7 rows selected
```

选择行

SQL允许我们选择一个数据库中的特定记录，也支持选择特定字段。这些都是通过WHERE子句来实现的。

符合WHERE子句定义的条件限定的数据才能被包含在结果集中，不符合条件的数据会被排除，在本章第四节*操作类型*中将会详细描述WHERE子句的使用。

➲ **示例**

```
dmSQL> SELECT * FROM Employee where <expression>;
```

6.4 操作类型

WHERE子句的使用有三种相关的操作类型：算子、比较符和逻辑算子。每种操作的用途各不相同，比较符是其中最常使用的。

比较符

比较符是用来比较两个值大小的操作，通常用于判定某一行是否应包含在结果集内。

OPERATOR	DESCRIPTION
=	表示相等的数学符号。
>	表示大于的数学符号。
<	表示小于的数学符号。
>=	表示大于或等于的数学符号。
<=	表示小于或等于的数学符号。
<>	表示不等于的数学符号。
BETWEEN	表示一个值的范围。
LIKE	表示匹配模式。
IN	表示数据库中的一条记录。
IS	表示特殊值的测试。

图6-4：比较符

首先，我们要演示一个非常简单的使用比较符的条件子句。

➲ **示例 1**

下例说明如何使用比较符：

```
dmSQL> SELECT * FROM Employee WHERE Number = 10006;
```

以下是返回的结果：

```
Number   FirstName     LastName      Manager     Phone     BirthDa* HireDate
======== =========== =============== ======== =========== ======== ========
```

```
   10006 Taylor      Galbreath          10002 648-6633    1975-02* 1994-10*

1 rows selected
```

通过在字段列表中使用星号（*）来查询**Employee**表中的所有字段。WHERE子句表明只有员工号码为10006时的记录会返回到结果集中。本例将返回员工号码为10006的Taylor Galbreath的记录。这种类型的查询在您知道一条记录中的唯一数据，并且想显示剩余数据时非常有效。

➲ 示例 2

下例说明了如何在所有员工姓名中查找以A、B和C开头的人名：

```
dmSQL> SELECT * FROM Employee WHERE LastName BETWEEN 'Aa' and 'Cz';
```

下面是返回的结果：

```
 Number   FirstName     LastName     Manager     Phone     BirthDa* HireDate
======== =========== =============== ======== =========== ======== ========
   10001 Greg        Carter            10000    NULL       NULL     NULL
   10002 Dean        Cougar            NULL     NULL       1970-12* NULL
   10003 Benson      Armstrong         10002    918-3517   1970-12* 1993-03*
   10004 Lyn         Belger            10000    837-7847   2059-05* 1991-12*

4 rows selected
```

通过在字段列表中使用星号（*）来查询**Employee**表中的所有字段。在WHERE子句中使用关键字BETWEEN来限定返回的结果集将是以字母A、B或C打头的员工名。实现这个操作运用了比较符BETWEEN并利用AND将条件分开。请注意，尽管此时AND与下面的逻辑符AND的使用方法相同，但它仅是BETWEEN子句的一部分而不是逻辑符AND。为了获取以字母A, B或C打头的员工名，查询使用值‘Aa’ 和 ‘Cz’。如果您仅使用字母‘A’ 和‘C’作为查询条件，那么结果仅返回以字母‘A’ 和‘B’打头的员工名而并不返回以‘C’打头的员工名。当使用‘Cz’时，您可以获得在字母Aa与Cz之间的所有员工名。一些以字母A或Czar打头的员工名将不包括在查询范围内，因为它们超出了查询的范围。

➲ 示例 3

下例说明了如何返回姓氏以字母‘C’打头的员工的信息：

```
dmSQL> SELECT * FROM Employee WHERE LastName LIKE 'C%';
```

下面是返回的结果：

```
 Number   FirstName     LastName     Manager     Phone     BirthDa* HireDate
```

```
======== ========== ============== ======== ========== ======== ========
   10001 Greg       Carter           10000    NULL     NULL       NULL
   10002 Dean       Cougar           NULL     NULL     1970-12*   NULL

2 rows selected
```

这条查询语句通过在字段列表中使用星号（*）来查询**Employee**表中的所有字段。在WHERE子句中使用关键字LIKE指定仅返回姓氏以字母‘C’打头的员工名。百分号‘%’表示字符串中字母C后可以跟任意字母，如果没有百分号的话，那么仅会返回姓氏为‘C’的员工信息。

逻辑符

逻辑符用于连接WHERE子句的两个表示式，并表明二者之间的关系。

OPERATOR	DESCRIPTION
AND	表示两个表达式必须都是真。
OR	表示需至少一个表达式是真。
NOT	表示一个表达式从一个等式中排除。

图6-5 :逻辑符

➲ **示例**

下例说明了如何返回1995年加拿大制作的电影名：

```
dmSQL> SELECT LastName FROM Employee WHERE HireDate > 01/01/1995 AND Manager =
10000;
```

返回的结果如下：

```
     LastName
==================
Davis
Belger

2 rows selected
```

算子

算子被用于演示数学计算。通常作为比较操作的一部分存在，执行一些计算后返回算式结果。

OPERATOR	DESCRIPTION
+	表示数学加法。
-	表示数学减法。
*	表示数学乘法。
/	表示数学除法。

图6-6：算子

6.5 删除

要在表中删除行时，dmSQL提供三种方法：标准的SQL、主变量以及OIDs，下图为UPDATE 语句的语法图。

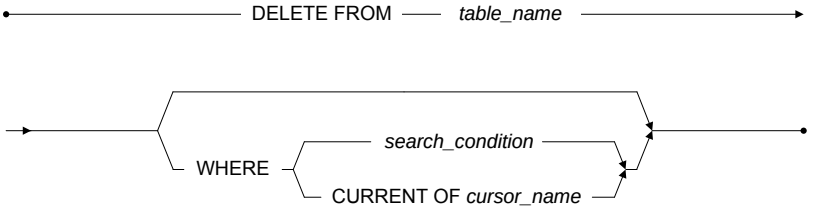


图 6-7: UPDATE 语句语法使用标准的SQL删除

使用标准的SQL删除

以下命令是删除**Employee**表中的所有行。

➲ **示例 1**

下例说明了如何删除**Employee**表中的所有行：

```
dmSQL> DELETE FROM Employee;
```

以下命令是删除**Employee**表中所有符合员工号**> 10030**的所有行。

➲ **示例 2**

下例说明如何删除**Employee**表中的行：

```
dmSQL> DELETE FROM Employee WHERE Number > 10030;
```

使用主变量删除

使用主变量删除语句使dmSQL进入dmSQL/Val>提示的界面。此时，您可以对相关的主变量输入数据，使用END退出赋值并完成更新语句。

➲ **示例 1**

下例说明了如何使用主变量删除Employee表中的员工：

```
dmSQL> DELETE FROM Employee WHERE FirstName = ?;

dmSQL/Val> 'Benson';

dmSQL/Val> 'Murphy';

dmSQL/Val> END;
```

➲ **示例 2**

下例说明了如何使用标准的SQL来完成上例中同样的命令：

```
dmSQL> DELETE FROM Employee WHERE FirstName = 'Benson';

dmSQL> DELETE FROM Employee WHERE FirstName = 'Murphy';
```

使用OIDs删除

尽管可以使用OIDs对数据进行操作，但是OID是数据库内部使用的特殊的二进制数据类型。通常不在dmSQL中直接使用OIDs。

➲ **示例**

下例说明如何使用OID删除**Employee**表中的员工**Taylor**：

```
dmSQL> SELECT OID FROM Employee WHERE FirstName = 'Taylor';

      OID
================
0100000002000800

1 rows selected

dmSQL> DELETE FROM Employee WHERE OID='0100000002000800'x;
```

7 数据库对象

数据库被划分为不同的逻辑结构，我们称之为对象。表、表空间、视图、同义字以及索引都是数据库对象。视图、同义字和索引为设定和快速访问数据提供了便捷的方法。视图和同义字支持用户自定义视图和数据库对象名称。当利用索引查寻一个字段时，将会迅速获取所需数据。

7.1 视图

DBMaster用户可定义虚拟表，即视图。所谓视图并不是一张真正的表，要建立视图必须先定义视图的名称和它所对应在数据库中表或视图的查询条件。数据库会把这个视图的定义储存在数据库中，而此视图所对应的数据并没有实际存储在数据库中。也就是说，在您浏览这个视图时，数据库会依据视图的定义，将其所对应的数据从相应的实际表和查询条件中筛选出来。

视图对数据库的查询是非常有效的。例如，对于一个相当复杂的查询指令，您可以把它定义成视图。如此一来，您就可以重复使用这个视图，而不用每次都重新输入这个复杂的查询指令。由于视图可以用它的定义来限制使用者存取的表字段和数据，所以，定义视图可以加强数据库的安全性。

由于视图是从查询表中派生出来的，因此，视图就只能用来查询数据。用户不能对视图进行更新、添加或删除数据的操作。

创建视图

每一个视图都是由名称和相关表或者其他视图的查询决定的。您可以给视图中的字段定义与其在原表中不同的字段名称。如果您没有定义视图的字段名称，视图名称将继承相关表的字段名称。

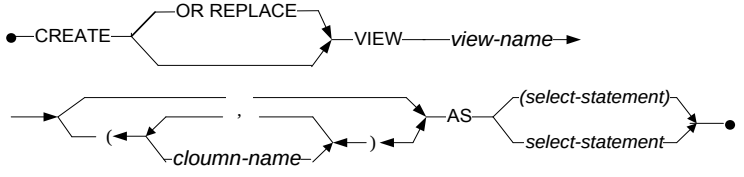


图7-1：创建视图的语法图

➲ **示例 1**

下例说明如何在**Employee**表中三个字段上创建视图并加以限定：

```
dmSQL> CREATE VIEW empView (FirstName, LastName, Telephone) AS
              SELECT FirstName, LastName, Phone FROM Employee;
```

用户仅可以通过视图**empView**查看表Employee的**FirstName**, **LastName**以及**Telephone**字段。

注意 用于定义视图的查询不包含UNION操作。CREATE OR REPLACE VIEW语法的使用：假设存在视图vi_staff，仅允许其他用户查看**tb_staff**表中的字段**name**和**ID**，如果想在不改变视图权限的前提下查看三个字段name、ID和age，我们需要更改视图的定义，用户可以使用以下SQL指令来重建视图**vi_staff**。

➲ **示例 2**

下例说明如何重建视图vi_staff，使用户可以查看tb_staff表中的三个字段： name、ID和age：

```
dmSQL> CREATE OR REPLACE VIEW vi_staff (empName, empId, empAge) AS
              SELECT name, ID, age FROM tb_staff;
```

删除视图

您可以删除一个没有用的视图。删除视图只会删掉此视图储存在数据库中的定义，而对视图所对应表的数据没有影响。

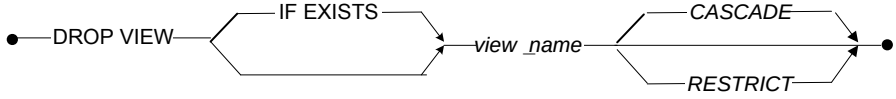


图 7-2 删除视图的语法图t

IF EXISTS: 确保在视图不存在的情况下不报错。

有关DROP VIEW命令的详细信息，请参考*SQL命令与函数参考*手册的*SQL命令*章节。

➲ **示例 1**

下例说明如何使用DROP VIEW命令删除视图：

```
dmSQL> DROP VIEW empView;
```

从**Employee**表中删除视图**empView**。

➲ **示例 2**

下例说明如何使用DROP VIEW IF EXISTS命令删除视图**vi_staff**：

```
dmSQL> DROP VIEW IF EXISTS vi_staff;
```

7.2 同义字

同义字也就是表或视图的别名。由于同义字只是一个别名的定义，所以数据库只需将它的定义储存在系统表中。

同义字主要用于简化表或视图的名称。通常情况下，DBMaster结合拥有者和对象的完整名称来识别表和视图的名称。但如果我们设置了同义字，我们就可以使用相应的同义字来访问表或视图，而无需键入它们的完整名称。由于同义字没有拥有者名称，所以它的名称必须是唯一的。同义字可通过dmSQL或JDBA工具来创建或删除。

创建同义字

假设表**Employee**的拥有者是**SYSADM**，该指令可为表**SYSADM.Employee**创建一个别名**Employee**，所有用户都可以直接使用这个同义字**Employee**来参照表**SYSADM.Employee**而无需键入全名。

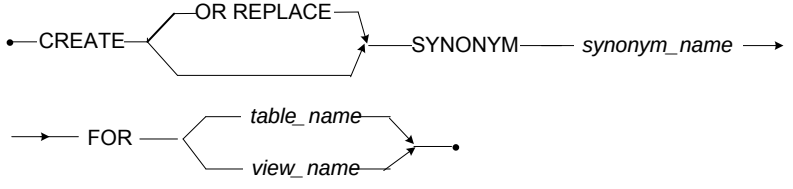


图7-3：创建同义字语法

➲ **示例 1**

下例使用CREATE SYNONYM指令创建同义字：

```
dmSQL> CREATE SYNONYM staff FOR SYSADM.tb_staff;
```

假设表tb_staff的拥有者是**SYSADM**，该指令可为表**SYSADM.tb_staff**创建一个别名**staff**，所有用户都可以直接使用这个同义字**staff**来参照表**SYSADM.tb_staff**而无需键入全名。

➲ **示例 2**

下例使用CREATE OR REPLACE SYNONYM 指令创建同义字**staff**：

```
dmSQL> CREATE OR REPLACE SYNONYM staff FOR SYSADM.tb_staff;
```

假设表**tb_staff**已存在一个别名**staff**，该指令可替换该别名而无需删除。

删除同义字

您可以删除一个不再需要的同义字。在删除同义字时，您只会将相关的定义从系统表中删除，而同义字对应的表或视图并不会被删除。

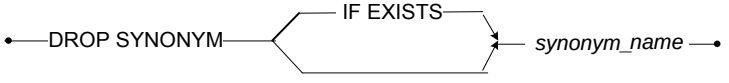


图7-4：删除视图的语法图

IF EXISTS: 确保在同义字不存在的情况下不报错。

有关DROP SYNONYM命令的详细信息，请参考*SQL命令与函数参考*手册的*SQL命令*章节。

➲ **示例 1**

下例使用DROP SYNONYM 指令删除同义字**Employee**：

```
dmSQL> DROP SYNONYM Employee;
```

➲ **示例 2**

下例使用DROP SYNONYM IF EXISTS指令删除同义字**Employee**：

```
dmSQL> DROP SYNONYM IF EXIST Employee;
```

7.3 索引

使用索引可以对数据行作快速随机访问。您可以在表上建立索引，这样可加快数据的查询。例如：当用户执行一个”SELECT Name FROM Employee WHERE Number = 10005”查询语句时，如果在字段**Number**上建有索引，那么读取数据的速度就会快很多。

一个索引可以由1-32个字段组成，表中的所有字段都可以用于建立索引。

您可以根据数据有没有重复值，来建立唯一性索引或非唯一性索引（non-unique）。如果字段建立了唯一性索引，那么这个字段除了空值（NULL）以外，不允许有相同的值。如果您在一个已有数据的表上建立唯一性索引，DBMaster会检查表中现存的数据是否唯一。如果有重复的值，DBMaster会返回一个错误信息。在表上创建一个唯一性索引后，DBMaster会检查新增或更新的数据是否和数据库中已有的数据重复，如果有重复的话，这笔数据就无法新增到数据库中。

创建索引时，您还可以指定索引中每个字段的排序为递增还是递减。例如：假设表中有5个值: 1，3，9，2，6，如果设为递增，它的顺序为:1，2，3，6，9；如果设为递减，它的顺序为: 9，6，3，2, 1。

当用户执行一条查询语句时，因为数据库选择用索引扫描数据，数据的输出顺序会受到索引顺序的影响。

➲ **示例**

如果您有个名为**friends**的表，包含**name**和**age**字段，其中为**age**字段创建一个降序的索引，可执行以下查询：

```
dmSQL> SELECT name, age FROM friends WHERE AGE > 20
```

➲ **结果**

```
      name                    age
---------------         ---------------
Jeff                              49
Kevin                             40
Jerry                             38
Hughes                            30
Cathy                             22
```

建立索引和建立表一样，都可以设定填充因子(fillfactor)。索引的填充因子是定义索引页中可以储存值的密度，索引填充因子的范围是**1%--100%**，默认值为***100%***。如果您建立索引后还需要经常更新数据，建议您将填充因子的值设定松一点，例如**60%**；如果您不需要更新该表中的数据，您可以使用表的默认填充因子100%。

用户也可以将索引创建在单独的表空间上，这样在访问多个磁盘时，可以提高磁盘的I/O效率。在创建一个索引前，建议您最好先将所有数据存入数据库，特别是在表的数据量很大时。这是因为如果您在数据加载之前先创建索引，每次在增加一笔数据时，都必须更新索引数据，这样会降低磁盘I/O的效率。 所以最好在数据载入后再建索引以提高效率。

创建索引

为表建立索引时，您必须指定索引的名称和字段。还可以设定字段的排序是递增还是递减，默认的顺序是递增。

您可以移动表到另一个表空间。若表和该表的索引位于同一表空间内，在移动表的同时可以将索引一起移动到另一个表空间；若表和该表的索引位于不同表空间内，则无法将索引移动到另一个表空间，因此我们可以在另一个表空间中重建索引。

索引不仅能在普通字段上创建，还能在表达式字段或用户自定义函数（UDF）字段上创建。为了提高XML查询性能，用户可以在XML字段上创建特殊XML索引。同一个全表索引相比，一个设计巧妙的过滤索引不仅能够提高数据库的查询性能，还能降低表索引的维护和储存成本。

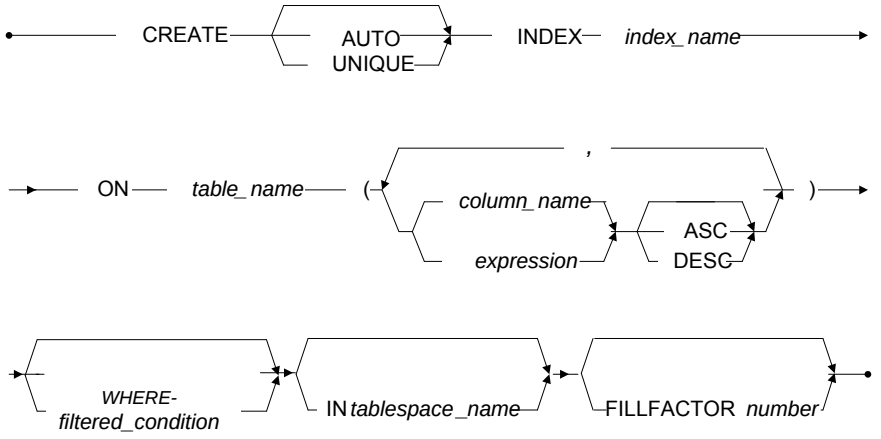


图7-5：创建索引的语法图

➲ **示例 1**

下例说明如何为**Employee**表的**Number**字段创建一个降序的索引idx1：

```
dmSQL> CREATE INDEX idx1 ON Employee (Number desc);
```

另外，创建唯一性索引时必须指明，否则DBMaster会创建出非唯一性索引。

➲ **示例 2**

下例说明如何为**Employee**表的**Number**字段创建一个唯一性索引**idx1**：

```
dmSQL> CREATE UNIQUE INDEX idx1 ON Employee (Number);
```

➲ **示例 3**

下例说明如何创建指定填充因子的索引：

```
dmSQL> CREATE INDEX idx2 ON Employee(Number, LastName DESC) FILLFACTOR 60;
```

➲ **示例 4**

在表**tb_ salary的UDF**子字符串**(nation,1,3)**上创建索引**idx_substr**：

```
dmSQL> CREATE INDEX idx_substr ON tb_salary (substring(nation,1,3) desc);
```

➲ **示例 5**

使用**提取**XML UDF创建一个索引：

```
dmSQL> CREATE INDEX idx_extr ON tb_extract (extract(id,
'/order/items/item/@product', NULL));
```

➲ **示例 6**

在表**tb_salary** 上使用**where** 子句创建一个过滤索引**filidx_income**：

```
dmSQL> CREATE INDEX filidx_income ON tb_salary(basepay+bonus,tax)where id>30;
```

删除索引

您可以使用JDBA工具或dmSQL的DROP INDEX语句删除索引。当索引成碎片或者效率低下时，应当被删除。重建索引将产生一个不含碎片的新索引。但如果这个索引是主键并且被其它表参照，就无法删除该索引。

➲ **示例**

下例说明如何删除表**Employee**的索引**idx1**：

```
dmSQL> DROP INDEX idx1 FROM Employee;
```

8 用户和权限

信息的安全对于数据库来说是至关重要的，用户不能够随意访问数据库的全部信息，同时也无法随意更改数据。DBMaster提供了几种方式来保护数据库中的数据安全并对用户的访问进行管理。

8.1 安全管理

安全管理对于限制某些数据只能被特定用户访问是至关重要的。

DBMaster提供了以下几种安全管理：

- **用户账号** — 通过用户的ID和密码来对数据库的访问进行控制
- **权限级别** — 控制不同用户的访问行为
- **表权限** — 用户可以共享某些私有数据
- **嵌套组** — 将用户分组以简化权限授予

一个唯一的用户ID和一个可更改的密码用以标识用户是否可以访问数据库，所有用户的账号都设有权限级别，这些权限指定了他们访问的许可类型。DBMaster提供了五种用户权限：CONNECT、RESOURCE、DBA (数据库管理员)、SYSDBA 和 SYSADM (系统管理员)。一个数据库可以拥有多个CONNECT、RESOURCE和DBA级别的账号，但只能拥有一个SYSADM级别的账号，该SYSADM账号专供建立、维护数据库和用户账号的用户使用。

表权限用于控制数据的访问，通过使用表权限，用户可以访问表中的数据。通过表权限可以使用户对表中的数据进行查看、插入、删除和更新操作，并且可以创建索引、参照表或在表中添加新的字段。通过使用PUBLIC密码，可以将表权限授予所有用户。

8.2 权限级别

只有授予权限的用户才可以连接到数据库，拥有*connect*权限的用户访问数据库所受的限制最多。他们不能够创建新的对象，如表或视图，只能够查看或更改PUBLIC表的数据，也不能更改或查看其他用户创建的表，直至该表的所有者、DBA或更高将该表的权限授予他们。他们被授予权限后，就可以使用被授予的任何一种表权限或PUBLIC表。

Resource

Resource权限允许用户在数据库中创建新表并删除任何一个之前由自己创建的表。作为表的所有者，他们可以创建和删除表中的视图和索引，同时也能够授予并收回他们拥有的任何一张表的权限，但是不能查看或更改其它用户创建的表，直至表的所有者或DBA将该表的权限授予他们或PUBLIC表。

DBA

DBA权限的用户对数据库拥有很大程度上的控制，并具备数据库中全部对象的所有权限。DBA可以创建表或查看、更改并授予所有表的权限，也可以创建新的数据库结构，如表空间和数据文件。为了控制同一时刻访问数据库的用户数，DBA可以创建和删除组并向组中添加或移除用户。但是DBA不能够更改当前用户的权限级别、为连接数据库的用户授予新的用户权限或更改用户密码。

SYSDBA

拥有SYSDBA权限的用户不仅可以授予或取消其他用户的CONNECT、RESOURCE和DBA权限，还可以更改非SYSADM或SYSDBA用户的密码，并可以向低权限用户设置ACL（访问控制列表）。拥有SYSDBA权限的用户具备DBA权限用户的所有权限，只有SYSADM才可以授予或取消用户的SYSDBA权限。如果SYSADM取消了用户的SYSDBA权限，此

时用户仍拥有DBA权限；如果SYSADM取消了用户的DBA权限，则该用户既无SYSDBA权限，也无DBA权限。

SYSADM

一个数据库只能拥有一个SYSADM权限用户，SYSADM ID默认分配给创建数据库的用户。SYSADM可以完全控制一个数据库并可以创建表、视图，更改和删除其他用户创建的表，为所有表授予表权限，创建新的表空间和数据文件。只有SYSADM和SYSDBA权限的用户才能够授予、收回、更改用户权限级别，并且更改用户密码。

8.3 新用户

只有SYSADM或SYSDBA权限的用户才可以为数据库添加新用户并授予CONNECT权限。

➲ **为数据库添加新用户**

1. 启动数据库并以SYSADM身份登录。
```
dmSQL> START DB tutorial SYSADM <password>;
```
2. 使用SYSADM密码连接到Tutorial数据库。
```
dmSQL> CONNECT TO tutorial SYSADM <password>;
```
3. 通过以下任何一种方式在不设密码的情况下新建一个用户账号。
```
dmSQL> GRANT CONNECT TO Judy;
dmSQL> GRANT CONNECT TO Judy NULL;
dmSQL> GRANT CONNECT TO Judy "";
```
4. 或者使用密码新建一个用户账号。
```
dmSQL> GRANT CONNECT TO Judy secret;
```

用户访问

当您以SYSADM的身份连接到数据库并准备添加一个新用户时，可使用GRANT关键字，该命令可指定用户名、密码和他们的权限级别。

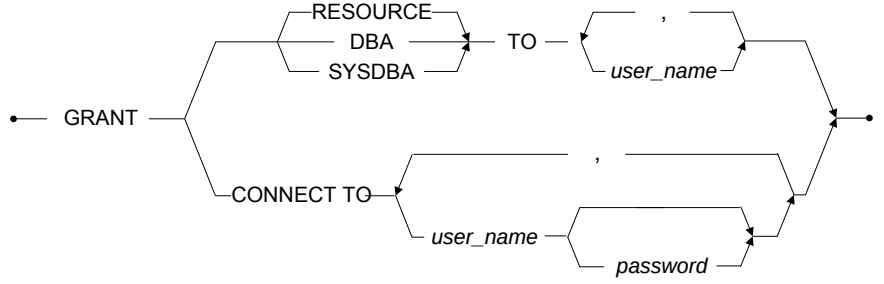


图 8-1: GRANT 语句语法

用户ID最多可包含128个字母数字混合的字符，如果首字符为数字，那么整个用户ID需要用双引号("")括起来。用户ID是否区分大小写要依据配置参数**DB_IDCap**的设置来决定。如果**DB_IDCap=0**，那么所有标识符（表

名称、用户名和密码）都将区分大小写，也就是说将有可能有两个不同的用户账号**judy**和**Judy**。**DB_IDCap**的默认值为***1***，如果不指定该参数的值，那么所有的标识符将不区分大小写。

类似于用户ID，密码的最大长度限制为16个字符。如果首字符为数字，那么也必须用双引号(“”)括起来。

在将用户权限提升为RESOURCE、DBA、SYSDBA和SYSADM之前，请先授予该用户CONNECT权限。当新建一个用户账号时，可以选择为该用户分配密码，也可以选择设置密码为空，并允许用户以后再添加密码。为了安全起见，创建用户时应该为用户分配一个临时密码，当用户首次登录数据库时，可以提示该用户更改密码。

用户每次登录数据库时，都需要键入正确的用户名，为了保持一致性，当新建用户账号时，最好使用相同格式的账号。

多用户

在同一时间为多个用户创建新的账号。除了需要在命令行同时指定多个用户名和密码外，创建多用户的语法和创建单用户的语法是一样的，

➲ **在不分配密码的情况下添加两个用户**

1. 启动数据库并以SYSADM身份登录。
```
dmSQL> START DB tutorial SYSADM <password>;
```
2. 通过SYSADM密码连接至**Tutorial**数据库。
```
dmSQL> CONNECT TO connect to tutorial SYSADM <password>;
```
3. 通过以下任何一种方式在不设密码的情况下新建用户账号。
```
dmSQL> GRANT CONNECT TO Tom, Judy;
dmSQL> GRANT CONNECT TO Tom “” , Judy “” ;
dmSQL> GRANT CONNECT TO Tom NULL, Judy NULL;
```
4. 或者使用密码新建用户账号。
```
dmSQL> GRANT CONNECT TO Tom secret1, Judy secret2;
```

8.4 提升权限级别

GRANT关键字也用于提升用户的权限级别，该命令就如同将CONNECT权限分配给一个新用户一样。如果用户已经拥有账号密码，那么就不需要为他分配新密码，除非一个新用户需要使用该账号。

➲ 示例

将用户权限从CONNECT提升为RESOURCE或DBA。

```
dmSQL> GRANT RESOURCE TO Judy;
dmSQL> GRANT DBA TO Judy;
```

为用户授予DBA权限并不会自动给该用户授予RESOURCE权限，所以降低用户权限级别就变得十分重要。如果您想授予用户RESOURCE和DBA权限，那么可使用GRANT命令两次，一次先授予RESOURCE权限，再授予DBA权限。

多用户

您也可以同时提升多用户的权限级别，除了要在命令行指定多个用户名，该语法和提升单用户的权限级别是一样的。此时所有用户都被授予相同权限级别，无法使用一条语句对多个用户授予不同级别的权限。

➲ 示例

提升两个用户的权限级别。

```
dmSQL> GRANT RESOURCE TO Tom, Judy;
dmSQL> GRANT DBA TO Tom, Judy;
```

8.5 降低权限级别

降低单用户的权限级别和提升用户权限级别相似，需用REVOKE关键字来代替GRANT关键字。

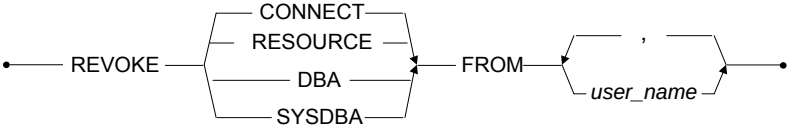


图 8-2: REVOKE语句语法

当收回一个用户的权限时，那么该用户的权限级别将会降为授予他的下一个较低的权限级别。例如：先授予一个用户CONNECT权限，再直接授予DBA权限而没有RESOURCE权限，当收回该用户的DBA权限时，他将只拥有CONNECT权限。

➲ **示例 1**

将用户的权限从DBA降低为下一个较低的权限级别。

```
dmSQL> REVOKE DBA FROM Judy;
```

➲ **示例 2**

将用户的权限从RESOURCE降低为CONECT。

```
dmSQL> REVOKE RESOURCE FROM Judy;
```

注意 *如果先后授予用户RESOURCE和DBA权限，那么在收回这两个权限后，该用户的权限级别将降低为CONNECT。*

➲ **示例 3**

将用户的权限级别从DBA降低为CONNECT：

```
dmSQL> REVOKE DBA FROM Judy;
dmSQL> REVOKE RESOURCE FROM Judy;
```

8.6 删除用户

REVOKE命令用于删除一个现有的用户，该命令和降低用户权限级别的命令是一样的。一旦用户的CONNECT权限被收回，那么该用户将从可连接数据库的用户列表中删除。

➲ **示例**

从数据库中删除一个用户：

```
dmSQL> REVOKE CONNECT FROM Judy;
```

8.7 密码

在创建用户账号时，SYSADM可为新用户分配一个临时密码。如果用户想更改密码或者SYSADM没有分配临时密码，那么用户可以更改密码或通过ALTER PASSWORD命令设置新密码。当用户忘记密码时，SYSADM也可以为该用户再次分配一个新密码。

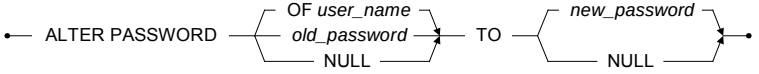


图 8-3: ALTER PASSWORD语句语法

➲ **示例 1**

为用户设置密码。

```
dmSQL> ALTER PASSWORD NULL TO newpass;
dmSQL> ALTER PASSWORD “” TO newpass;
```

➲ **示例 2**

更改用户的临时密码。

```
dmSQL> ALTER PASSWORD X9elx4 TO newpass;
```

可以将用户当前的密码更改为NULL来删除用户密码。

➲ **示例 3**

将用户的密码更改为NULL。

```
dmSQL> ALTER PASSWORD oldpass TO NULL;
dmSQL> ALTER PASSWORD oldpass TO “”;
```

SYSADM可以更改或删除任何用户的密码，其他用户不可以更改别人的密码。在更改密码时，SYSADM不需要知道当前的密码。

➲ **示例 4**

SYSADM可选择以下任何一种方式更改用户密码。

```
dmSQL> ALTER PASSWORD OF Judy TO newpass;
dmSQL> ALTER PASSWORD OF Judy TO NULL;
```

```
dmSQL> ALTER PASSWORD OF Judy TO" ";
```

8.8 组的管理

当数据库变得非常大并且拥有很多用户时，分别给不同的用户授予用户权限就变得日益困难。为了解决这个问题，DBMaster定义了用户组的概念。通过组，您可以将多个需要相同数据库权限的用户合并成一个组以简化权限级别的管理。同一时刻，您可以为组中的所有或一定数量的用户授予\取消表权限。

创建组

您可以使用CREATE GROUP命令创建组。

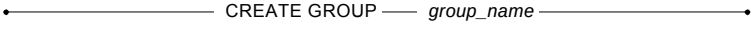


图 8-4: CREATE GROUP语句语法

组名可以包含文字数字式字符，组名可以是任意长度，但是只有前8个字符可用。例如在命名组名时，**CompanyEmployee**和**CompanyExecutives**是一样的，都为**CompanyE**。试图创建两个有效组名相同的组，将产生错误信息。

组名是区分大小写的，所以**companyExecutives**和**CompanyExecutives**将分别代表两个不同的组**companyE**和**CompanyE**。为了避免这个问题，可以将组名限制为8个或更少的字符，避免组名的描述过长。

➲ **示例**

为所有行销人员创建组：

```
dmSQL> CREATE GROUP marketing
```

将所有行销部人员添加到marketing组中，然后为该组授予权限，那么该组中的所有成员都将可以访问相同的对象。

添加组中的成员

同时在组中添加一个或多个用户。

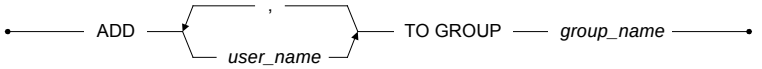


图 8-5: ADD TO GROUP语句语法

➲ **示例 1**

为组**SalesRep**添加一个新成员**Judy**：

```
dmSQL> ADD Judy TO GROUP SalesRep;
```

➲ **示例 2**

将**Judy, Jeff**和**Trent**添加到组**SalesRep**中：

```
dmSQL> ADD Judy, Jeff, Trent TO GROUP SalesRep;
```

删除组中成员

通过REMOVE GROUP命令可以删除组中的单用户或多个用户。

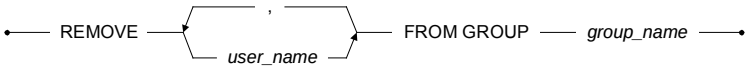


图8-6: REMOVE FROM GROUP语句语法

➲ **示例 1**

将用户**Judy**从**SalesRep**组中删除：

```
dmSQL> REMOVE Judy FROM GROUP SalesRep;
```

➲ **示例 2**

将用户**Judy**, **Jeff**和**Trent**从**SalesRep**组中删除：

```
dmSQL> REMOVE Judy, Jeff, Trent FROM GROUP SalesRep;
```

删除组

如果一个用户组为空或不再被需要，可使用DROP GROUP命令将该组删除。

图 8-7: DROP GROUP语句语法

如果组不为空，可通过上节介绍的REMOVE FROM GROUP命令删除组中的成员。

➲ **示例**

确定**SalesRep**组为空后，删除该组。

```
dmSQL> DROP GROUP SalesRep;
```

嵌套组

嵌套组能提高组的功能性，创建一个嵌套组可通过ADD TO GROUP命令，并使用组名作为用户ID，删除一个嵌套组可通过REMOVE FROM GROUP命令。

➲ **创建一个名为NYSales的组**

1. 键入以下命令。
```
dmSQL> CREATE GROUP NYSales;
```
2. 将**NYSales**添加到**SalesRep**组中。
```
dmSQL> ADD NYSales TO GROUP SalesRep;
```
3. 再将**NYSales**从**SalesRep**组中移除。
```
dmSQL> REMOVE NYSales FROM GROUP SalesRep;
```

8.9 表级别权限

表的所有者、拥有DBA或更高权限的用户可拥有7个级别的表权限，其中SELECT、INSERT、DELETE和UPDATE权限可允许用户查看并更改表中的数据，另外三个权限中，INDEX权限允许用户创建索引，ALTER权限允许用户更改表定义，REFERENCE权限允许用户为表添加参照完整性约束，表权限也可用于更改指定的字段。

Select

SELECT权限允许用户查看表或视图中的任何数据，表的所有者、拥有DBA或更高权限的用户可赋予表的SELECT权限。

Insert

INSERT权限允许用户向表中插入数据，可以通过INSERT关键字后的字段名列表为整张表或指定的字段授予INSERT权限，表的所有者、拥有DBA或更高权限的用户可赋予表的INSERT权限。

Delete

DELETE权限允许用户从表中删除数据，表的所有者、拥有DBA或更高权限用户可赋予表的DELETE权限。

Update

UPDATE权限允许用户更新表中的字段值，可以通过UPDATE关键字后的字段名列表为整张表或指定的字段授予UPDATE权限，表的所有者、拥有DBA或更高权限的用户可赋予表的UPDATE权限。

Index

INDEX权限允许用户为表创建索引，表的所有者、拥有DBA或更高权限的用户可赋予表的INDEX权限。

Alter

ALTER权限允许用户更改表的定义，表的所有者、拥有DBA或更高权限的用户可赋予表的ALTER权限。

Reference

REFERENCE权限允许用户为表创建外键，可以为整张表或指定的字段授予REFERENCE权限。在为指定字段授予REFERENCE权限时，可在REFERENCE关键字后列出字段名，表的所有者、拥有DBA或更高权限的用户可赋予表的REFERENCE权限。

8.10 GRANT权限

表的所有者、拥有DBA或更高权限的用户可为其他用户授予表权限，也可为整张表或指定字段授予权限。

通过GRANT命令为表分配权限。

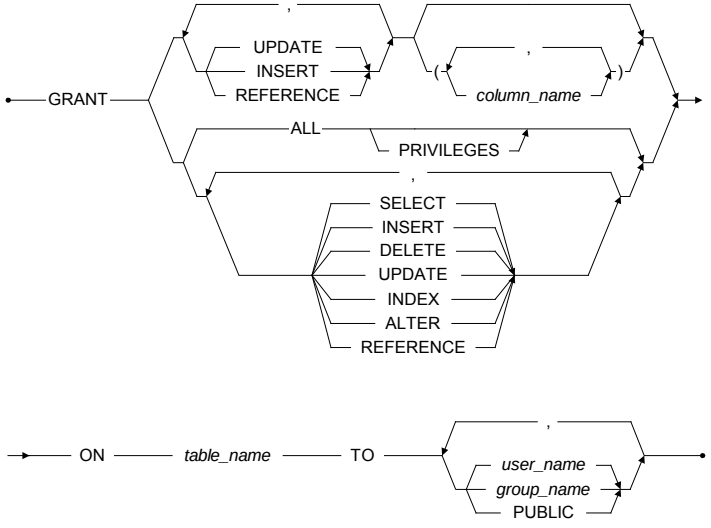


图8-8: GRANT语句语法

权限不能同时分配给表和指定的字段，您必须使用两次命令来为整张表和指定的字段授予权限。权限可以授予给单个用户、组，或通过PUBLIC关键字授予所有用户。

GRANT表权限

➲ **示例 1**

键入如下命令为用户**Judy**授予**SalesRep**表的SELECT权限。

```
dmSQL> GRANT SELECT ON SalesRep TO Judy;
```

您也可以同时授予更多的权限，在命令行中列出要指定的权限，并用逗号分开。

➲ **示例 2**

键入如下命令为用户**Judy**授予**SalesRep**表的SELECT和UPDATE权限。

```
dmSQL> GRANT SELECT, UPDATE ON SalesRep TO Judy;
```

如果要为用户授予所有表权限，可在命令行中列出所有表示权限的关键字，或使用DBMaster提供的关键字ALL。

➲ **示例 3**

为用户**Judy**授予**SalesRep**表的所有表权限：

```
dmSQL> GRANT ALL ON SalesRep TO Judy;
```

如果要为多个用户授予权限，可指定多个用户名，中间用逗号分开。

➲ **示例 4**

键入如下命令为**Judy** 和**Jeff**用户授予**SalesRep**表的SELECT 和UPDATE权限。

```
dmSQL> GRANT SELECT, UPDATE ON SalesRep TO Judy, Jeff;
```

可通过指定组名称为用户组或多个组授予权限，中间用逗号分开。

➲ **示例 5**

键入如下命令为**Personnel**和**SalesMgr**组授予**SalesRep**表的SELECT和UPDATE权限。

```
dmSQL> GRANT SELECT, UPDATE ON SalesRep TO Personnel, SalesMgr;
```

注意　　*不能同时授予多张表的权限。*

GRANT字段权限

您可以只为指定的字段授予INSERT, UPDATE和REFERENCE权限，在为字段授予权限时，不能使用相同的命令为整张表授予其它权限。

➲ **示例 1**

键入如下命令为用户**Judy**授予**SalesRep**表的**Name**字段的INSERT权限。

```
dmSQL> GRANT INSERT (Name) ON SalesRep TO Judy;
```

➲ **示例 2**

为用户**Judy**授予多个字段的INSERT权限，字段之间用逗号分开：

```
dmSQL> GRANT INSERT (Name, Age, RepOffice, Title) ON SalesRep TO Judy;
```

当使用一个命令授予多个权限时，所有的权限必须作用于相同的字段。DBMaster不能够使用同一条命令为不同的字段授予权限。

➲ **示例 3**

键入如下命令为**Judy** 用户授予**SalesRep**表的**Name**和**Age**字段的UPDATE、INSERT和REFERENCE权限。

```
dmSQL> GRANT INSERT, UPDATE, REFERENCE (Name, Age) ON SalesRep TO Judy;
```

为多个用户和组授予多个字段权限的方法和授予表权限是一样的，用户和组名称之间可用逗号分开。

8.11 REVOKE权限

表的所有者、拥有DBA或更高权限的用户可以收回其他用户的表或指定字段的权限，您可以使用REVOKE命令来收回权限。

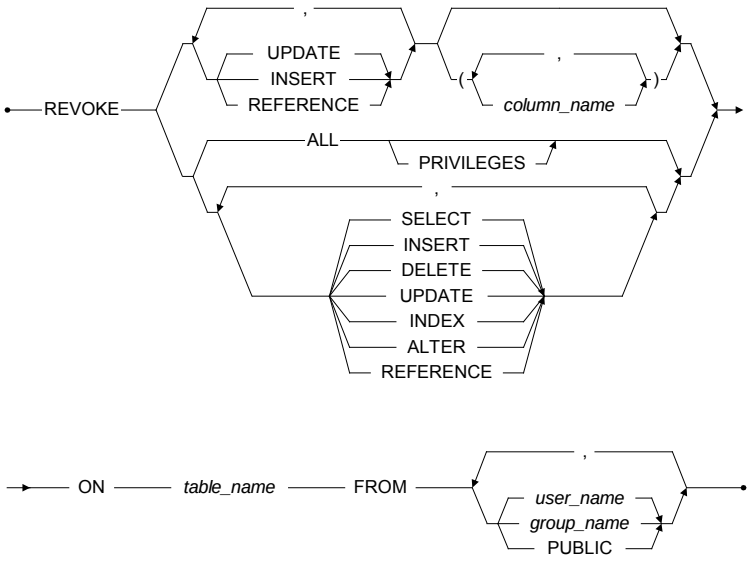


图8-9: REVOKE语句语法

表和指定字段的权限不能够同时被收回，您必须使用两次命令才能收回整张表和指定字段的权限。通过PUBLIC关键字可以收回单个用户、组或所有用户的权限。

REVOKE表权限

➲ **示例 1**

键入如下命令收回**Judy**用户的**SalesRep**表的SELECT权限。

```
dmSQL> REVOKE SELECT ON SalesRep TO Judy;
```

要收回多个权限，可在命令行中列出要收回的权限，中间用逗号分开。

➲ **示例 2**

键入如下命令收回**Judy**用户的**SalesRep**表的SELECT和UPDATE权限：

```
dmSQL> REVOKE SELECT, UPDATE ON SalesRep TO Judy;
```

如果要收回用户的所有表权限，可在命令行中列出所有表示权限的关键字（SELECT、INSERT、UPDATE、DELETE、ALTER、INDEX和REFERENCE），或使用DBMaster提供的关键字ALL。

➲ **示例 3**

键入如下命令收回**Judy**用户的**SalesRep**表的所有列出的权限：

```
dmSQL> REVOKE ALL ON SalesRep TO Judy;
```

如果要收回多个用户的权限，可指定多个用户名，中间用逗号分开。

➲ **示例 4**

键入如下命令收回**Judy**和**Jeff**用户的**SalesRep**表的SELECT和UPDATE权限：

```
dmSQL> REVOKE SELECT, UPDATE ON SalesRep TO Judy, Jeff;
```

如果要收回用户组或多个组的权限，将用户名用组名替换即可。

➲ **示例 5**

键入如下命令收回**Personnel** 和**SalesMgr**组的**SalesRep**表的SELECT和UPDATE权限：

```
dmSQL> REVOKE SELECT, UPDATE ON SalesRep TO Personnel, SalesMgr;
```

注意 *不能同时授予或收回多张表权限。*

REVOKE字段权限

您可以只收回指定字段的INSERT, UPDATE和REFERENCE权限。如果收回字段权限，就不能使用相同的命令收回整张表的其它权限。

➲ **示例 1**

键入如下命令收回**Judy**用户**SalesRep**表的Name字段的INSERT权限：

```
dmSQL> REVOKE INSERT (Name) ON SalesRep TO Judy;
```

➲ **示例 2**

键入如下命令收回**Judy**用户**SalesRep**表的多个字段的INSERT权限，字段之间用逗号分开：

```
dmSQL> REVOKE INSERT (Name, Age, RepOffice, Title) ON SalesRep TO Judy;
```

当通过命令收回多个权限时，所有权限必须被所有字段所使用。

➲ **示例 3**

键入如下命令收回**Judy**用户**SalesRep**表的**Name**和**Age**字段的UPDATE, INSERT和REFERENCE权限：

```
dmSQL> REVOKE INSERT, UPDATE, REFERENCE (Name, Age) ON SalesRep TO Judy;
```

将多个用户和组的字段权限收回方法和收回表权限的方法是一样的，用户和组名称之间可用逗号分开。

9 数据库恢复

对所有的数据库管理系统（DBMS），数据库会因为硬件或软件的问题而遭到破坏。一个RDBMS可能在毫无准备的情况下遭到严重的损害。当故障发生时，一个好的RDBMS应该能够提供一些机制来恢复这些遭到损坏的数据库。事实上，这也正是关系型数据库管理系统胜过一般档案管理系统的主要优势之一。

DBMaster的高级数据保护功能足以防止因为故障而引起的数据丢失和downtime，这些恢复，备份和还原功能确保了DBMaster数据库的可靠性和数据一致性。

本章将介绍可能出现的数据库故障的类型以及防止数据丢失的步骤。在前面的章节中，已经举例说明如何使用命令行工具dmSQL，本章将举例说明如何使用Jserver Manager工具来执行数据库的备份和恢复。

9.1 数据库发生故障的类型

数据库故障一般分为两类：系统故障和介质故障。无论哪种故障发生，都有可能引起数据库中数据的不一致性或数据丢失。当数据库遭到破坏时，RDBMS必须提供一些措施来恢复数据库，或者利用备份数据库来还原遭到损坏的数据库。

系统故障

系统故障，又称作实例故障，是来自计算机系统的内存故障。造成系统故障的原因可能是因为突然断电、应用程序或操作系统的溃损、内存错误或其它原因，导致数据库管理系统的异常结束。

当出现系统故障时，应用程序和活动事务都被异常中止。正处理事务的状态或没有完全写到磁盘的事务的可靠性将得不到保证，而这些事务应当被恢复。数据库管理系统需要利用*事务日志*或*日志文件*来对数据库作灾难恢复，这样才能将数据恢复到未遭受系统故障前的状态。

介质故障

介质故障又称为磁盘故障，是计算机系统的磁盘存储介质遭到了破坏。造成磁盘损坏的原因很多，可能是单纯的读写头损坏、某段磁道损坏或者火灾、地震、水灾等天灾人祸。

介质故障一旦发生，数据丢失就无法避免，而且可能不止一个文件本身被损坏。这时就必须依靠数据库管理系统所提供的备份和还原功能来恢复数据库。

9.2 数据库损坏后的恢复方式

恢复的目的就是要保证数据库遭到破坏后，提交的事务能够反映到数据库中，未提交的事务不能反映到数据库中，并且尽快地恢复到正常状态。

DBMaster 利用日志文件和检查点来恢复数据库，日志文件和检查点能够使用户在毫无察觉的情况下，尽快地确保所有事务的恢复。

日志文件

*日志文件*实时记录着数据库的所有更改和每次改变的状态。在系统发生故障后，利用日志文件的历史记录，DBMaster可以恢复和重做那些已经被改变但尚未写入磁盘的事务，或者取消那些被异常终止的事务。

如果数据库运行在备份模式状态下，日志文件还可以存储更多的信息，DBMaster可以使用这些信息去还原更多的数据库。这些信息在做备份前，将存储在日志文件中，并占据一定的空间。在数据库的还原过程中，DBMaster将备份日志文件中的信息添加到备份的数据库中。因此，只需要备份记录两次完整备份间数据库改变的日志文件。

检查点

*检查点*是把系统的执行状态反映到磁盘上的一个系统事件，也就是说，此时系统内存的内容与磁盘上的内容完全相同。在检查点，DBMaster会将内存区中的所有日志记录和修改过的数据页写入磁盘，并且回收不再作为备份或恢复使用的日志块。

系统发生故障后，执行检查点会节省数据库的启动时间，DBMaster会将最后一次的检查点时间和当时的活动事务状态写到日志文件的文件头处。在数据库恢复的过程中，DBMaster会利用这些信息去决定哪些事务需要重做，哪些事务需要取消，哪些事务应该被忽略。

当日志空间写满时，DBMaster会自动执行检查点，以回收一些日志块以便重用。如果这样仍不能满足完成当前事务的日志空间的需要，这个事

务将会被取消。DBMaster也会在数据库启动、关闭或执行在线备份时，自动执行检查点。

数据库管理员可以利用CHECKPOINT命令来手动执行检查点。执行检查点的最恰当时间，会受到数据库中活动事务的多少，日志文件的大小和数量，事务平均大小的影响。也就是说，由于每个数据库应该执行检查点的时间都不同，有经验的数据库管理员应根据实际需要来执行检查点。手动执行检查点可以减少事务遇到日志空间不足的情况，并可以缩短数据库启动、终止、备份的时间。

执行检查点也要占据相当长的一段时间，这段时间的长短主要取决于从上次执行检查点到现在的事务大小和数量。执行检查点时，当前活动事务需要等DBMaster决定回收哪一个日志记录后再执行。直到DBMaster将日志记录和脏数据页写入磁盘后，当前活动事务才继续执行。

恢复步骤

如果系统发生故障，或在启动过程中出现错误，DBMaster数据库会在启动时自动执行恢复动作。在恢复的过程中，DBMaster总会执行两个单独的动作：重做和取消。

恢复过程的第一个步骤是重做已经记录在日志文件中的事务。此步骤的必要性在于可能存在一些已经确认，但尚未反应到磁盘上的事务数据。然而，这些记录都已存储到日志文件中，并且通过这个步骤可以记录到数据库。执行完这个步骤，所有提交的、未提交的事务都会记录到数据库中。

第二个步骤是取消(或回滚)，是把没有得到确认的事务撤回。此步骤的必要性在于系统发生故障时，无法确定进程中事务的准确状态，因此不能继续完成。因为事务必须保证不是确认就是撤回，所以必须将这些没有得到确认的数据全部取消。做完这个步骤后，数据库中就包含了所有提交的事务数据，但是不包含未提交的事务。

DBMaster为介质故障或系统故障后启动数据库提供支持。介质故障、系统故障可能产生自动恢复机制不能修复的数据库不一致情况。在这种情况下，数据库无法启动，您通常需要通过数据库的备份来还原。如果您从未做过备份，那么可以使用数据库的强制启动模式来启动数据库，即

在**dmconfig.ini**配置文件中设置关键字**DB_ForcS**。这将允许您强制启动数据库，并且允许您导出数据库中未受损坏的数据。要获得更多有关强制启动模式的信息，请参考*数据库管理员手册*。

9.3 备份的类型

备份是为了保护数据库，以免数据库受到介质故障和其它故障的影响。在存储介质损坏后，可能有一个以上的文件遭到物理损坏而不可用，数据库管理者应该使用最近的一次备份来替换受损的文件以重建数据库。

数据库的备份分为三种：*完整备份*，*差异备份*和*增量备份*。除此之外，您还可以选择*在线备份*或*离线备份*。DBMaster支持由几种不同形式的备份模式和数据库状态组合的备份。

完整备份

完整备份是指在某时间点的一个数据库的完整拷贝，包括所有数据和日志文件的备份。您也可以创建一个配置文件**dmconfig.ini**的拷贝，保留数据库的常规配置信息。无论数据库在线还是离线，数据库管理员都允许您执行一个完整备份。

完整备份是整个数据库的拷贝，因此它需要很大的存储空间。然而利用完整备份功能恢复数据库相对节省些时间，因为您可以简单地将备份文件覆盖至受损文件上。完整备份能让数据库还原到最近一次完整备份时的状态。

差异备份

差异备份基于最近一次的完整备份，该完整备份是差异备份的基础。在创建一个差异备份时，必须存在一个差异基础。

差异备份仅包括差异基础创建之后数据的改变情况，一个差异基础通常被几个差异备份使用。在恢复数据库时，完整备份和与之相关的差异备份将会产生一个完整的数据库。

由于日志文件的频繁更改，差异备份仅包括数据文件（所有DB文件 和BB文件）和有用的日志块。在差异备份期间，只有有用的日志块会被复制。

增量备份

增量备份是指上次做过完整备份之后，更改过的日志文件的拷贝，这些文件只拷贝了上次做过备份之后的数据库的改变。因为增量备份仅包含数据库的更改，为了恢复数据库，在执行增量备份之前，您必须先执行一个完整备份或差异备份。只有当数据库在线时，您才能执行一个增量备份。

请注意增量备份由一系列日志文件组成，这些日志文件记录了备份模式（DB_BMode）开启后的所有更新数据。数据库运行于普通模式（DB_SMode = 1）时，若执行增量备份，则需先执行完整备份或差异备份。而数据库运行于复制模式（DB_SMode = 4）时，执行增量备份前无需执行完整备份或差异备份。

增量备份只备份日志文件，所以它们只需占用一小部分的存储空间。但是在还原数据库时，因为数据库管理系统必须重做此备份日志文件中的所有事务，所以会花费更多的时间。使用者可以结合使用完整备份与差异备份、增量备份，把数据库还原到前一次完整备份或差异备份与最后一次增量备份之间的任一时间点。

离线备份

*离线备份*是在数据库关闭后，对数据库做的备份。数据库管理员必须设定关闭数据库的时间，并通知所有用户及时断开连接。离线备份给用户带来了不便，因为他们必须记住在数据库关闭前，完成所有活动事务并断开连接。在执行一个完整备份时，数据库必须处于离线状态。

RDBMS不一定要提供离线备份的功能，因为在关闭数据库后，您可以通过操作系统命令去备份一个数据库。数据库管理员可以通过这个方式去执行一个离线备份，或者通过使用JServer Manager工具：一个使用方便的图形化工具，在不使用操作系统命令的情形下，轻松地实现一个离线备份。

在线备份

*在线备份*可以在数据库运行时进行。数据库管理员无需关闭数据库，用户也无需断开与数据库的连接就能执行在线备份。在线备份给用户提供了方便，因为不需要用户的任何操作。当数据库在线时，DBMaster不仅可以执行完整备份和差异备份，还可以执行增量备份。

RDBMS必须提供在线备份数据库的功能，因为RDBMS在备份的过程中仍在运行并且有用户连接。DBMaster不仅可以使用dmSQL工具和执行操作系统命令的方法来手动地执行在线备份，还可以通过JServer Manager工具：一个使用方便的图形化工具，在不使用操作系统命令的情形下，轻松地实现一个在线备份。

在线增量备份至当前

DBMaster也提供了另一种备份方式，称作在线增量备份至当前。

对于拥有一个以上日志文件的数据库而言，在线增量备份和在线增量备份至当前的不同之处在于：在线增量备份将备份从上次备份之后所有修改过的日志文件，但是不包括目前使用的这个日志文件；而在线增量备份至当前则包括了目前所使用的这个日志文件。也就是说，在线增量备份只能将数据库恢复到日志文件中所记录的最后一次提交事务的状态，而在线增量备份至当前可以将数据库恢复到目前日志文件所记录的状态。

如果数据库只有一个日志文件，在线增量备份和在线增量备份至当前是一样的。 在这种情况下，对这两种增量备份方式DBMaster都是备份当前这个日志文件。

备份方式的结合

DBMaster不仅可以执行完整备份和差异备份，还可以执行增量备份，但是它们是互斥的，所以您不能同时执行完整、差异和增量备份。同样地，DBMaster不仅可以执行在线备份，也可以执行离线备份。另外，DBMaster允许将完整、差异或增量备份与在线或离线备份结合起来使用。

DBMaster支持以下备份方式的结合：离线完整备份、在线完整备份、在线差异备份、在线增量备份。DBMaster还支持一种额外的备份方式，称作在线增量备份至当前。

对于拥有一个以上日志文件的数据库而言，在线增量备份和在线增量备份至当前的区别不大，但是很值得注意：在线增量备份将备份从上次备份之后所有修改过的日志文件，但是不包括目前使用的这个日志文件；而在线增量备份至当前则包括了目前所使用的这个日志文件。也就是说，在线增量备份只能将数据库恢复到日志文件中所记录的最后一次提交事务的状态，而在线增量备份至当前可以将数据库恢复到目前日志文件所记录的状态。

如果数据库只有一个日志文件，在线增量备份和在线增量备份至当前是一样的。在这种情况下，唯一的日志文件也就是当前使用的日志文件，这两种增量备份方式下DBMaster都会备份这个日志文件。

9.4 备份模式

备份模式决定了DBMaster是否可以执行在线增量备份、增量备份的数据类型，以及何时可以释放那些已经结束的事务所占据的日志块。DBMaster提供了三种数据库的备份模式：NONBACKUP、BACKUP-DATA和BACKUP-DATA-AND-BLOB。

不备份模式

不备份（NONBACKUP）模式无法为插入或更新的数据提供保护，在这种模式下不能执行在线增量备份。当数据库遭到系统故障时，可以使用日志文件去恢复数据库。当存储介质损坏时，可能导致数据的丢失。在执行检查点后，那些未利用的日志块都可以立即收回使用。但是日志空间一旦写满，数据库就只能恢复到最后一次作完整备份的状态。

备份数据模式

备份数据模式可以为最近一次完整备份后插入或更新的数据(不包括BLOB数据)提供保护。在这种模式下，数据库管理员可以执行在线增量备份，但是只有非BLOB数据可以储存在备份文件中。当数据库遭到系统故障时，可以使用日志文件去恢复数据库，并且可以部分恢复来自存储介质的损坏。尽管可以利用最近一次备份还原数据库至存储介质损坏的时间点，但BLOB数据的更改将会丢失。在执行检查点后，那些未利用的日志块才能被收回使用，并备份了日志文件。

备份数据和BLOB模式

备份数据和BLOB模式允许为插入或更新的数据(包括BLOB数据)提供保护。在这种模式下，数据库可以执行在线增量备份，并且所有的数据都将储存在备份文件中。当数据库遭到系统故障时，可以使用日志文件去恢复它，同时也可以恢复来自存储介质的损坏。在恢复数据库时，数据

可以还原至存储介质损坏的时间，包括所有的BLOB数据。在执行检查点后，那些未利用的日志块才能被收回使用，并备份了日志文件。

表空间的BLOB备份模式

DBMaster提供的备份模式是作用在整个数据库上的。也就是说，数据库中的所有表空间都使用同一种备份模式。如果数据库在BACKUP-DATA-AND-BLOB模式下，DBMaster会将数据(包括BLOB数据)的更改信息全部记录到日志文件中。记录日志文件中的BLOB数据会快速消耗日志空间，所以应该为频繁的备份提供较大的日志空间。

当您不希望丢失数据时，上述方法可能有效。但很多时候，您同时可能备份了很多不重要的BLOB数据。在这种情况下，很难选择到底使用哪种备份模式。为了防止这种情况发生，DBMaster允许您在创建表空间时更改数据库的备份模式。

在执行CREATE TABLESPACE命令时，您可以使用BACKUP BLOB ON /OFF 选项来表示这个指定表空间的BLOB数据重不重要，需不需要备份，ON表示重要，需要备份。OFF表示不重要，不需要备份。

每一个表空间的备份模式都是数据库备份模式和表空间备份模式的结合，它有如下规定：

- 如果数据库运行在BACKUP-DATA-AND-BLOB模式上，同时创建表空间时使用BACKUP BLOB ON选项，DBMaster将备份该表空间中的BLOB数据。
- 如果数据库运行在BACKUP-DATA-AND-BLOB模式上，同时创建表空间时使用BACKUP BLOB OFF选项，DBMaster就不会备份该表空间中的BLOB数据。
- 如果数据库运行在BACKUP-DATA模式上，在您创建表空间时，无论是否使用BACKUP BLOB ON/OFF选项，DBMaster都不会备份BLOB数据。

新建表空间的默认选项为***BACKUP BLOB ON***。当数据库运行在BACKUP-DATA-AND-BLOB模式下，表空间中BLOB数据的更改会记录到日志文件中。

备份文件对象模式

在数据库中除了可以对普通数据和BLOB数据进行备份外，用户还可以选择备份文件对象的模式。用户只能在启动备份后台服务进行完整备份的过程中备份文件对象。在备份之前，用户应该先启动备份服务，设置完整备份计划并设置备份目录。

DBMaster中定义了两种类型的文件对象：用户文件对象和系统文件对象，数据库管理员可以选择仅备份系统文件对象、备份系统和用户文件对象或两者都不选择。您可以通过**dmconfig.ini**配置文件中的**DB_BkFoM**关键字来指定文件对象的备份模式。

- **DB_BkFoM** = 0：不备份文件对象
- **DB_BkFoM** = 1：只备份系统文件对象
- **DB_BkFoM** = 2：备份系统和用户文件对象

在备份文件对象时(**DB_BkFoM** = 1, 2)，备份服务会把文件对象的外部文件复制到**DB_BkDir**关键字指定目录的“FO”子目录中。您可以通过**DB_FBkTm**和**DB_FBkTv**关键字来设定完整备份的时间计划。

➲ **示例**

以下是**dmconfig.ini**文件中的一部分，包含以下关键字：

```
[MyDB]
DB_BkSvr = 1                        ; starts the backup server
DB_FBKTm = 01/05/01 00:00:00        ; begins from midnight at May 1, 2001.
DB_FBKTV = 1-00:00:00               ; interval is every one day.
DB_BkDir = /home/dbmaster/backup    ; backup directory
DB_BkFoM = 2                        ; backup both system and user file objects
```

因为文件对象的备份模式为2，备份服务会把外部数据库的文件复制到“*/home/dbmaster/backup/FO*”目录中，如果**FO**子目录不存在，那么备份服务将自动创建它。

*FO*子目录中的文件会以一个有序数来重新命名，例如：如果源外部文件名为“**/DBMaster/mydb/fo/ZZ000123.bmp**”，那么备份服务将会把此文件复制到**FO**子目录下，并且重命名为“**fo0000000344.bak**”，也就是说它就是第344个文件对象。源文件名和新文件名都会记录到文件对象映

射文件**dmFoMap.his**中。要想获取更多有关文件对象映射文件的信息，请参考第9.7章的*备份历史文件*。

通过**DB_BkOdr**参数指定的旧的备份目录，备份服务也会将先前版本的文件对象移动到**FO**子目录下。

数据库管理员应该考虑到当执行完整备份时，启用文件对象的备份需要占用更多的时间。一个完整备份包括：(1) 如果设置了**DB_BkOdr**参数，将复制先前的完整备份；(2) 复制所有数据库文件；(3) 复制所有日志文件；(4) 如果设置了**DB_BkFoM**参数，将复制所有文件对象。为了避免备份失败，请确保由**DB_BkDir**关键字指定的备份目录所在的磁盘有足够空间可以利用。

存储过程的备份模式

DBMaster定义了三种类型的存储过程：SQL存储过程、ESQL存储过程以及JAVA存储过程。因为存储过程的源代码都将写入数据库，所以在执行完整备份期间，SQL存储过程都将以普通数据的方式存储在数据库中。在数据库中除了可以对普通数据、BLOB数据以及文件对象进行备份之外，用户还可以选择备份ESQL存储过程或JAVA存储过程。用户只能在启动后台服务进行完整备份的过程中备份ESQL存储过程和JAVA存储过程。在执行备份之前，用户应该先启动备份服务器，设置完整备份的时间计划和备份目录。要想获得更多有关设置完整备份的信息，请参考*数据库管理员手册*第15.6章*备份服务器*。

用户可以通过设置**DB_BkSPm**来指定存储过程的备份模式。

- **DB_BkSPm** = 0：不备份ESQL存储过程和JAVA存储过程。
- **DB_BkSPm** = 1：备份ESQL存储过程和JAVA存储过程。

➲ **示例**

以下是**dmconfig.ini**配置文件的一部分，其中包含以下关键字：

```
[MyDB]
DB_BkSvr = 1                        ; starts the backup server
DB_FBKTm = 14/05/01 00:00:00        ; begins from midnight at May 1, 2014
DB_FBKTV = 1-00:00:00               ; interval is every one day
DB_BkDir = /home/dbmaker/backup     ; backup directory
```

```
DB_BKSPM = 1                          ; backup all ESQL stored procedures and JAVA
stored proce
```

ESQL存储过程和JAVA存储过程备份文件的默认目录为关键字**DB_BkDir**指定目录下名为**SP**的子目录。如果用户已经在配置文件**dmconfig.ini**中设置了关键字**DB_BkOdr**，那么在进行完整备份的过程中，ESQL存储过程和JAVA存储过程的备份序列将被移动到名为SP（位于**DB_BkOdr**所指定的旧备份目录下）的子目录下，之后这些备份序列将会从存储过程的默认目录中删除。

在备份ESQL存储过程和JAVA存储过程期间，备份服务器首先会生成一个名为dmSpBk.his的文件，之后复制存储过程的文件，同时创建使用了文件扩展名.s0和.b0的文件，该文件用来载入已备份的存储过程。对于ESQL存储过程，需要备份的文件为源文件和对象文件；对于JAVA存储过程，需要备份的文件仅为对象文件。

为了方便重建存储过程，ESQL存储过程的源文件将以spnameowner.ec格式重命名，例如，假定ESQL存储过程的名称为**y1**，拥有者为SYSADM，则复制的源文件将被重命名为**y1SYSADM.ec**。

文件**dmSpBk.his**用来记录备份信息。有关更多存储过程备份列表文件的信息，请参考15.7章*备份历史文件*。

数据库管理员应该考虑到当执行完整备份时，启用文件对象的备份会需要占用更多的时间。一个完整备份包括：(1) 如果设置了**DB_BkOdr**，将复制先前的完整备份；(2) 复制所有数据库文件；(3) 复制所有日志文件；(4) 如果设置了**DB_BkFoM**，将复制所有文件对象；（5）如果设置了**DB_BkSPm**，将复制ESQL存储过程和JAVA存储过程。为了避免备份失败，请确保备份目录所在的磁盘有足够空间可以利用，备份文件的目录可通过**DB_BkDir**关键字来指定。

压缩备份文件

数据库文件有可能会变得非常大并且需要大量的可用空间来存储备份文件。如今，DBMaster支持压缩备份文件功能。您可以在**dmconfig.ini**配置文件中设置**DB_BkZip**关键字来启用或禁止该功能。若数据库正在运

行，您可以使用系统存储过程**SetSystemOption**更改BkZip来启用或禁止该功能。

DB_BkZip = 1：压缩备份文件。

DB_BkZip = 0：不压缩备份文件。(默认)

压缩文件的格式为GZIP，您可以使用任何一种GZIP兼容工具来读取压缩文件。

注意 *FO文件不能被压缩，即使您将DB_BkZip关键字设置为启用压缩备份文件模式。*

设置备份模式

DBMaster提供了几种不同的设置备份模式的方法，您可以根据您的数据库是否在线，使用dmSQL命令行工具直接更改配置文件的方式，或JServer Manager图形化工具，。

更改数据库的备份模式以提高备份保护，对使用的日志文件会产生影响。在更改备份模式之前，日志文件将记录先前没有记录的信息。当您更改备份模式时，必须执行一个完整或差异备份，它可以在恢复的过程中为备份日志文件的更新提供了一个起点。

当更改数据库的备份模式以降低备份保护时，无需额外的步骤，只是日志文件停止记录更改信息。在恢复的过程中，DBMaster将上一次的完整或差异备份作为更新日志文件的起点，但是这样会丢失一些更新数据。

数据库管理员可以在离线的状态下，使用**dmconfig.ini**配置文件或JServer Manager工具来更改数据库的备份模式，因为备份模式的更改会影响日志文件的使用，所以在使用新备份模式启动数据库之前，必须执行离线完整备份。在离线条件下更改备份模式可以不受约束，可以从任意一种备份模式更改成另一种备份模式。当用户从低备份保护转换到高备份保护时需要执行一个完整备份。有关更多使用JServer Manager工具更改备份模式的信息，请参考*服务器管理工具用户手册*。

数据库管理员可以在在线的状态下，使用dmSQL工具来更改数据库的备份模式。同时备份模式的更改会影响日志文件的使用，在启动和结束完

整备份期间，备份模式可以从低备份保护转换到高备份保护(例如：从NONBACKUP模式转换到BACKUP-DATA模式或BACKUP-DATA-AND-BLOB模式，或者从BACKUP-DATA模式转换到BACKUP-DATA-AND-BLOB模式)。

➲ **示例 1**

通过dmSQL来在线更改数据库的备份模式：

```
dmSQL> BEGIN BACKUP;
dmSQL> SET DATA BACKUP ON;
dmSQL> END BACKUP DATAFILE;
dmSQL> END BACKUP JOURNAL;
```

➲ **示例 2**

或者：

```
dmSQL> BEGIN BACKUP;
dmSQL> END BACKUP DATAFILE;
dmSQL> SET DATA BACKUP ON;
dmSQL> END BACKUP JOURNAL;
```

DBMaster不允许将数据库的备份模式调低，除非先将数据库的备份模式更改为NONBACKUP模式。例如：为了将数据库的备份模式BACKUP-DATA-AND-BLOB更改成BACKUP-DATA，用户必须先将备份模式设置为 NONBACKUP，然后根据上述规则将备份模式调高。备份模式可以从BACKUP-DATA-AND-BLOB或BACKUP-DATA 随时更改为NONBACKUP，此动作无需在完整备份开始和结束之间执行。然而，运行期间用户不能直接将备份模式从NONBACKUP模式切换到BACKUP-DATA-AND-BLOB模式，或从BACKUP-DATA-AND-BLOB模式切换到BACKUP-DATA模式。有关更多执行在线完整备份的信息，请参考后面的*在线完整备份*章节。

利用DMCONFIG.INI配置文件来更改数据库的备份模式

如果数据库处于离线状态，您可以直接使用**dmconfig.ini**配置文件中的**DB_BMode**关键字来直接更改数据库的备份模式。在下次启动数据库时，新的备份模式就会启用。如果数据库处于在线状态，当您更改**DB_BMode**关键字后，只有在重启数据库后**DB_BMode**关键字的更改才会生效。如果您将备份模式从NONBACKUP更改到BACKUP-DATA或BACKUP-DATA-AND-BLOB模式，或者从BACKUP-DATA模式更改到

BACKUP-DATA-AND-BLOB模式，请务必在此之前执行一个离线完整备份。

➲ **通过dmconfig.ini配置文件设置备份模式**

1. 通过任一种**ASCII**文本编辑器，打开**dmconfig.ini**配置文件。
2. 在配置文件相应地数据库配置节更改数据库的备份模式。
3. 将**DB_BMode**关键字的值更改为以下任一种。

```
0 - NONBACKUP mode
1 - BACKUP-DATA mode
2 - BACKUP-DATA-AND-BLOB mode
```

4. 重新启动数据库，DBMaster将使用新的备份模式来运行。

如果您希望修改的数据库的配置中没有**DB_BMode**关键字，那么您可以将此关键字添加到当前数据库配置的任一行中，关键字出现的顺序是无关紧要的；如果您没有指定**DB_BMode**的值，数据库会使用其默认值***0***（NONBACKUP模式）。

使用DMSQL设置备份模式

如果数据库处于在线状态，同时您也想使用dmSQL命令行工具，您可以在dmSQL中输入SQL SET命令来更改数据库的备份模式，但是必须在进行在线完整或差异备份期间执行此命令。当执行此命令时，新的备份模式将立即被启动。

➲ **通过dmSQL命令行工具来设置备份模式**

1. 通过dmSQL连接数据库。
2. 利用BEGIN BACKUP命令开始执行在线完整备份。
3. 在完整备份期间，通过以下SET命令之一来更改备份模式：

```
dmSQL> SET BACKUP OFF;
dmSQL> SET DATA BACKUP ON;
dmSQL> SET BLOB BACKUP ON;
```

4. 结束**在线完整备份**。

其中的SET BACKUP OFF命令表示NONBACKUP模式；SET DATA BACKUP ON 命令表示BACKUP-DATA模式；SET BLOB BACKUP ON 命令表示BACKUP-DATA-AND-BLOB模式。

使用**JServer Manager**更改数据库的备份模式

如果数据库处于离线状态，您可以使用JServer Manager图形化工具来更改数据库的备份模式。JServer Manager可以自动更改**dmconfig.ini**配置文件中的**DB_BMode**关键字。在您下次启动数据库时，新的备份模式将会启动。如果数据库处于在线状态，只有在您重启数据库后，**DB_BMode**关键字的更改才会生效。如果您将数据库的备份模式从NONBACKUP更改到 BACKUP-DATA或BACKUP-DATA-AND-BLOB模式，或者将BACKUP-DATA模式更改到BACKUP-DATA-AND-BLOB模式，请您务必在此之前执行一个离线完整备份。要想获得更多的有关如何使用JServer Manager工具来设置离线备份模式的内容，请参考*服务器管理工具用户手册*。

9.5 离线完整备份

为了执行离线完整备份，您必须拥有数据库文件的读权限和备份目录的写权限。如果您在执行备份时不得不先关掉数据库，那么您必须拥有数据库的DBA或更高权限。

您可以在不考虑备份模式的情况下执行一个离线完整备份，数据库可以运行在NON-BACKUP、BACKUP-DATA或BACKUP-DATA-AND-BLOB模式下。使用离线完整备份，您可以将数据库恢复到数据库关闭时的状态。

使用dmSQL执行离线完整备份

➲ **使用dmSQL执行离线完整备份**

1. 通知所有用户，数据库将在某时刻关闭，请他们在此之前断开连接。
2. 如果数据库处于运行状态，请用**TERMINATE DB**命令关闭数据库。如果在关闭数据库时出现错误，则重启数据库并修正此错误后再次关闭它。
3. 检查**dmconfig.ini**文件，列出所有与备份有关的文件和目录，包括文件对象目录。
4. 使用操作系统命令或直接使用工具将数据库文件（包括数据文件、日志文件、文件对象和**dmconfig.ini**文件）复制到备份目录或备份设备下。

通过JServer Manager工具执行离线完整备份

➲ **通过JServer Manager工具执行离线完整备份**

1. 在数据库服务器端启动**JServer Manager**应用程序。
2. 在主控台台中选择**备份数据库**，将出现不同的备份选项。

3. 从备份窗口中选择**离线完整备份**，**离线完整备份**窗口将出现。
4. 从**数据库名称**下拉菜单中选择一个数据库，JServer Manager将提示您登录数据库。
5. 在**用户ID**字段中输入用户的ID。
6. **注意** *任一拥有DBA或更高权限的用户都可以备份数据库。*在**密码**字段中输入密码。
7. 点击**确定**按钮，该用户将连接到数据库，**离线完整备份**窗口将显示需要备份的操作系统文件列表。
8. 点击浏览按钮(…)在备份目录字段中选择一个新的路径。
9. 点击**确定**按钮将保存备份目录中的所有文件。

   注意 *如果该文件已经存在于备份目录中，那么数据库管理员可以选择覆盖它。*
10. 检查**dmconfig.ini**文件并确定文件对象的路径。
11. 通过操作系统命令或直接使用工具将数据库文件复制到**备份**目录或**备份**设备下。

9.6 备份服务

尽管DBMaster提供了一种手动备份数据库的方式，但您仍需了解备份规则。为了避免这个问题，DBMaster提供了一种简单易用的方法，即用备份服务来自动执行在线完整、差异和增量备份。请注意，备份服务只能执行在线备份，因为只有数据库启动后，备份服务才能启动。

在JServer Manager工具的'通过备份服务备份'的程序运行期间，数据库管理员也可以轻松地实现备份操作。

备份服务运行在后台，并且根据时间计划或日志文件来执行在线完整、差异和增量备份。因为备份服务和数据库服务存在联系，可以灵活确定何时会执行一个备份，什么类型的增量备份被执行，哪一个备份选项可以被更改。数据库服务器在启动的同时，备份服务也被启动，并且一直运行直到您终止备份服务或关闭数据库服务。

当执行完整备份时，备份服务会把备份目录中最后一次的完整备份复制到旧目录中。然后将所有的数据库文件包括日志文件和**dmconfig.ini**配置文件复制到备份目录中，覆盖先前的完整备份。

当执行差异备份时，由于日志文件的频繁更改，备份服务仅拷贝数据文件（所有的DB文件和BB文件）和有用的日志块。在差异备份期间，只读表空间中的数据文件也不备份。

差异备份仅包括差异基础创建之后数据的改变情况，一个差异基础通常被几个差异备份使用。在恢复完数据库后，完整备份和与之相关的差异备份将会产生一个完整的数据库。

当执行增量备份时，备份服务会将所需的日志文件复制到备份目录中。

我们可以选择配置备份服务的选项来设置备份文件的文件名格式、备份路径的位置、旧目录的位置、备份服务执行备份的时间计划、完整备份后执行差异备份的时间间隔以及可保留的差异备份的最大数目、备份服务执行增量备份前的填充日志数以及备份服务保存备份文件的方式。

在dmSQL运行期间，可以使用**SetSystemOption**存储过程来进行备份设置，即在数据库运行期间，您可以使用**SetSystemOption**存储过程更改

BKSVR、BKDIR、BKITV、DBKTV、BKTIM、BKFUL、BKFOM、BKZIP、BKCMP、BKRTS、BKCHK、FBKTM、FBKTV、DBKMX、BKODR、BKFRM。

启动备份服务

备份服务是后台服务程序，其生命周期和数据库服务器的生命周期一样长。设置完**DB_BkSvr**关键字后，您无需启动备份服务。因为DBMaster在启动数据库的同时，将自动启动备份服务。默认值是禁止使用备份服务。仅当数据库以多用户模式启动时，备份服务才能启动。

备份服务有两种状态：激活和不激活。您可以使用**DB_BkSvr**控制备份服务状态。**DB_BkSvr**值设置为0，备份服务处于不激活状态，此时备份服务器不响应任何备份请求，即备份服务不执行任何备份；**DB_BkSvr**值设置为1，备份服务处于激活状态，此时备份服务器响应一系列备份请求，您可以执行任何备份。

激活备份服务有三种方法：在**dmconfig.ini**文件中将**DB_BkSvr**值设置为1；数据库启动后通过call setsystemoption('bksvr','1')命令将**DB_BkSvr**值更改为1；数据库运行时使用Jserver Manager中的动态设置更改备份设置。

使用备份服务执行备份前，您需要设置一些相关参数。例如：备份目录、压缩备份模式等。

设置相关参数方法如下：

- 启动数据库前，您可以在**dmconfig.ini**文件中设置相关参数。下次启动数据库时，备份服务将使用这些关键字初始化相关参数。
- 若数据库已经启动，您可以使用Jserver Manager中的**动态设置**更改参数值。此外，您还可以使用命令 *call SetSystemOption('option_name','value')*。请注意个别参数只能使用set语法设置，例如设置备份为OFF、设置数据备份为ON、设置BLOB备份为ON。

备份服务处于激活状态且在**dmconfig.ini**文件中设置适当备份参数后，您可调用系统存储过程**SetSystemOption**开始备份，该存储过程适用于任何客户端工具或用户应用程序。

➲ **示例**

下例语法用于执行在线完整、差异和增量备份：

```
dmSQL> CALL SETSYSTEMOPTION(‘STARTBACKUP’,‘1’); //执行完整备份
dmSQL> CALL SETSYSTEMOPTION(‘STARTBACKUP’,‘2’); //执行增量备份
dmSQL> CALL SETSYSTEMOPTION(‘STARTBACKUP’,‘3’); //执行差异备份
```

使用DMCONFIG.INI启动备份服务

如果数据库处于离线状态，您可以通过在**dmconfig.ini**配置文件中设置**DB_BkSvr**关键字来直接启动备份服务。 当您再次启动数据库时，备份服务也将被同时启动。如果数据库处于在线状态，只有在您重新启动数据库时，对**dmconfig.ini**配置文件中的**DB_BkSvr**关键字的更改才会生效。

➲ **通过设置dmconfig.ini配置文件启动备份服务**

1. 用任何一种ASCII文本编辑软件打开**dmconfig.ini** 文件。
2. 找到要启动备份服务的数据库配置项。
3. 确保备份模式为BACKUP-DATA或BACKUP-DATA-AND-BLOB。将**DB_BMode**设为1是BACKUP-DATA模式，**DB_BMode**设为2是BACKUP-DATA-AND-BLOB模式。
4. 改变**DB_BkSvr**关键字为1，启动备份服务。
5. 重启数据库，开始备份服务。

使用DMSQL启动备份服务

如下所示，当数据库在线时，使用dmSQL命令行工具可动态地启动备份服务。

```
dmSQL> CALL SETSYSTEMOPTION('BKSVR'', '1');
```

您可以使用*Call SetSystemOption('BkSvr', '1')*更改BkSvr。如果您想在更改BkSvr的同时更改**dmconfig.ini**配置文件中关键字**DB_BkSvr**的值，您可以使用*Call SetSystemOptionW('option', 'value')*。

启动备份服务并在**dmconfig.ini**配置文件中设置合适的备份参数后，您可通过调用系统存储过程**SetSystemOption**执行完整备份，该备份适用于任何客户端工具和用户应用程序。

```
dmSQL> CALL SETSYSTEMOPTION('STARTBACKUP','1'); //执行完整备份
dmSQL> CALL SETSYSTEMOPTION('STARTBACKUP','2'); //执行增量备份
dmSQL> CALL SETSYSTEMOPTION('STARTBACKUP','3'); //执行差异备份
```

更改增量备份时间间隔的语法如下：

```
dmSQL> CALL SETSYSTEMOPTION('bkitv', 'Interval')
```

使用JSERVER MANAGER启动备份服务

无论数据库是否在线，您都可以使用JServer Manager图形化工具来启动备份服务器。JServer Manager将自动更改**dmconfig.ini**配置文件中的关键字**DB_BkSvr**。若数据库处于离线状态，则下次数据库启动时，备份服务器也将启动。对于在离线状态下如何使用JServer Manager工具来启动备份服务器，请参考*服务器管理工具用户手册*。

➲ **通过JServer Manager工具离线启动备份服务**

1. 从主控制台或**数据库**下拉菜单中选择**启动数据库**选项，**启动数据库**对话框将出现。
2. 从**数据库名称**下拉列表框中选择将要更改设置的数据库。
3. 点击**设置**按钮，将出现**启动数据库高级设置**窗口。
4. 点击**备份**标签页。
5. 要启动备份服务器，请选中**启动备份服务**选框。
6. 选择一种备份模式。
7. 如果只需要备份数据，请选择**只备份Data**选项。
8. 如果要备份数据和BLOB文件，请选择**备份Data和BLOB**选项。
9. 点击**保存**按钮。
10. 点击**取消**按钮返回到**启动数据库**窗口。

注意 *如果**DB_BkSvr**关键字没有出现在要启动备份服务的数据库配置项中，JServer Manager工具将会自动将该关键字添加到相应的位置。*

差异备份文件名格式

差异备份文件名格式如下：

DTimeStamp_DataFileName.dif(2)

D —差异备份标志（必要的）

TimeStamp — 距1970年1月的时间 (00:00:00 GMT)

DataFileName — 数据文件的数据库名称

.dif — 差异备份文件的文件扩展名。对于一个特定的完整备份，如果相应的差异备份源文件不存在，那么该差异备份文件的扩展名必须是**.dif2**.

假设2009/12/01 14:11执行第一个差异备份, 那么生成的差异备份文件名如下：

D1259647860_DBNAME.BB.dif, D1259647860_DBNAME.DB.dif, D1259647860_DB NAMESAMPLE5.SBB.dif 和 D1259647860_DB NAME.SDB.dif. 日志文件名是 D1259647860_DB NAME.JNL

增量备份文件名格式

备份文件名格式为<I><TimeStamp><_><DB_BKFRM>，其长度不能超过256个字符，例如，I1234567890_%2F%4N%4B.JNL。timestamp是系统的10位数字数据的有效时间。<DB_BKFRM>可以包括固定文字部分和格式字符串(转义字符)。

增量备份文件名至少由三部分字符串组成：完整备份ID、数据库名称和备份标识符。在备份序列中命名增量文件时，备份服务会分配一个完整备份的ID。在恢复数据库的过程中，DBMaster会使用这个完整备份ID去重建备份序列。数据库名称用以识别此增量备份文件属于哪个数据库，备份标识符确定此增量备份在备份序列中的相对位置。

格式字符串包括三个部分：转义字符、数值长度和格式字符。以下为有效的格式字符串。

%[*x*]F—完整备份ID，其中的变量*x*可有以下四种格式：

1：完整备份ID的格式为：YYYYMMDD, 例如：20010917

2：完整备份ID的格式为：MMDD, 例如：0917

3：完整备份ID的格式为：MMDDhhmm, 例如：09171305

4：完整备份ID的格式为：DDhhmmss, 例如：17130558

%[*n*]B—备份标识符。

%[*n*]N—日志文件所属的数据库名。

转义字符表示此处是格式字符串的起点，由%表示。如果在您的文件名中希望包括％字符，您必须使用两个%(例如：%%)来表示，在%字符后面必须是单个数字或上述有效的格式字符。如果%后接其它任何字符，那么备份文件名的格式会视为无效。同时，DBMaster也会返回一个错误信息。

长度字符n是一个1到9之间的整数，表示由此格式字符串所产生的字符串长度。当格式字符串所产生的字符串长度小于此长度时，将会补0。对％【ｎ】Ｎ产生的字符串，0会补在右面，其它的字符串则补在左面。同样，字符串若超过这个长度，会被截断。数据库名称会从右边截断，而其它值会从左边截断。【ｎ】是可选项，也就是说在格式字符串中，您可以不必设定长度，备份服务会使用格式字符所产生的完整字符串长度。

格式字符用以代表特殊字符串，合法的格式字符为F, B或N。除此之外，都是无效的备份文件名格式，DBMaster会返回一个出错信息。任何没有接转义字符或转义字符加单个数字的有效格式字符都被当作文本常量处理。

Date和time型数值是从系统中获得的，如果系统的时间和日期是正确的，那么此类型的值也是正确的。这个值代表了备份序列中的日志文件的顺序，DBMaster可以通过备份服务为每一个日志文件自动提供这个值。

DBMaster提供了几种不同的设置备份文件名格式的方法。 您可以利用直接更改配置文件的方式或者使用JServer Manager图形化工具来设定备份日志文件的文件名格式。

使用dmconfig.ini配置文件设置备份文件名格式

如果数据库处于离线状态，您可以通过**dmconfig.ini**配置文件中的**DB_BkFrm**关键字来设置备份文件名格式。当您再次启动数据库时，备份服务会将此备份文件名格式应用到所有的备份日志文件中。如果数据库处于在线状态，只有重新启动数据库后，对**DB_BkFrm**关键字的更改才会生效。

➲ **使用dmconfig.ini配置文件设置备份文件格式**

1. 用任何一种ASCII文本编辑软件打开**dmconfig.ini** 文件。
2. 找到要设置备份文件的数据库配置项。
3. 将**DB_BkFrm**关键字更改为用于备份文件名格式的字符串。
4. **注意** *此字符串可以包括任何有效的格式序列和文本常量，但文件名总长度不能超过256个字符。*重启数据库，开始使用新的备份文件名格式。

使用dmSQL 设置备份文件名格式

数据库运行期间，可通过存储过程**SetSystemOption**更改备份文件名格式，语法如下所示：

➲ **语法**

```
CALL SETSYSTEMOPTION('bkfrm', 'name')
```

➲ **示例**

在dmSQL命令行中输入以下命令，将备份文件名称更改为I1234567890_%2F%4N%4B.JNL。

```
dmSQL> CALL SETSYSTEMOPTION('bkfrm', 'I1234567890_%2F%4N%4B.JNL');
```

使用JSERVER MANAGER工具设置备份文件名格式

无论您的数据库处于离线还是在线状态，都可以使用JServer Manager图形化工具来设置备份服务的备份文件名格式，JServer Manager将自动更改**dmconfig.ini**配置文件中的**DB_BkFrm**关键字。当您启动数据库时，备份服务会将这些备份文件名格式应用到所有的备份日志文件中。要想

获得更多的有关如何使用JServer Manager工具来设置备份文件名格式的内容，请参考*服务器管理工具用户手册*。

➲ **使用JServer Manager设置备份文件格式**

1. 从主控制台或**数据库**下拉菜单中选择**启动数据库**选项，**启动数据库**对话框将出现。
2. 从**数据库名称**下拉列表框中选择将要更改设置的数据库。
3. 点击**设置**按钮，将出现**启动数据库高级设置**窗口。
4. 点击**备份**标签页。
5. 勾选**启动备份服务**选框。
6. 在**备份文件格式**字段中为备份的日志文件输入一个格式。
7. 点击**保存**按钮。
8. 点击**取消**按钮返回到**启动数据库**窗口。

注意 *如果**DB_BkFrm**键字没有出现在要启动备份服务的数据库配置项中，JServer Manager工具将会自动将该关键字添加到相应的位置。*

设置多个备份路径

DBMaster可以为用户提供多个备份文件路径，该功能用于当用户将文件保存至备份目录中，但是该目录不能为完整备份提供足够的存储空间时。如果该选项已设置，那么DBMaster会将剩余的数据备份至第二个备份目录中，于是备份就可以顺利地执行下去。用户也可以使用多个备份路径来执行完整、差异备份或增量备份。当用户使用多备份文件路径来备份文件时，DBMaster须遵循以下约定：

- 当备份文件时，数据库系统会设法将每个文件逐个存储到备份目录中。例如，当将文件存储至备份目录1时，如果目录1没有足够的空间存储该文件，那么该文件将被保存到备份目录2中，以此类推。如果所有的备份目录都已填满，那么系统将返回一条错误信息。
- 当在从数据库中备份文件时，只能使用一个备份目录。

- FO必须备份在第一个备份目录中。
- 备份路径的最大数量为32。

➲ **示例**

DBMaster依据以下结构来设置多备份路径：

```
DB_BkDir = <BKDIR 1> <SIZE 1> < BKDIR 2> <SIZE 2> < BKDIR 3> <SIZE 3>…

< BKDIR n > : the n's backup path
< SIZE n > : the size of the n's backup path
```

当为数据库**DB1**设置多备份路径时，你需要在配置文件中设置**DB_BkDir**关键字指明备份路径。

```
DB_BkDir = /home/usr/DBMaster/bk 5000 /home2/backup 1000
```

当目录*/home/usr/DBMaster/bk*下的可用存储空间都已填满时，数据库会将文件备份至*/home2/backup*目录下。

备份目录

备份目录用于指定备份文件的存放位置，DBMaster支持单备份文件路径和多备份文件路径。备份服务将自动创建BkDir。然而，您最好在不同磁盘上选择一个或多个备份目录，这样可以避免在介质损坏时，发生数据库和备份文件都丢失的情况。

备份目录可以通过**dmconfig.ini**配置文件中的**DB_BkDir**关键字来设定，此关键字的值可以包含备份目录的一个完整路径或相对路径。如果您没有指定备份目录，备份服务将在数据库目录下自动创建一个名为**backup**的默认目录。数据库目录可以通过**dmconfig.ini**配置文件中的**DB_DbDir**关键字来设定，备份路径的总长度不能超过256个字符。

当数据库运行在复制模式（主或从数据库）时，BkDir需设置为单路径。若将BkDir设置为多路径，则仅第一个路径可用且忽略路径大小。此外，如果一个目录中存在几个数据库，那么备份服务最好不要使用默认备份目录，因为一个数据库的备份历史信息可能会覆盖或追加到另一个数据库的备份历史信息中。为了避免这种情况的发生，您可以为每一个数据库创建不同的路径，或者为每个数据库明确的指定一个备份路径。将不

同的数据库存放于不同的目录中，是一个更可取的办法，因为这样可以正确的查看到哪些文件属于哪个数据库。

DBMaster提供了几种不同的设置备份目录的方式，您可以根据您的数据库在线与否，使用dmSQL命令行工具直接更改配置文件的方式，或使用JServer Manager图形化工具。

使用DMCONFIG.INI配置文件来设置备份目录

如果数据库处于离线状态，可以直接在**dmconfig.ini**配置文件中设置**DB_BkDir**关键字来创建备份目录。当您再次启动数据库时，备份服务会使用您设定的这个备份目录。如果数据库处于在线状态，只有在您重启数据库后，对**DB_BkDir**关键字的更改才会生效。

➲ **通过dmconfig.ini配置文件设置数据库的备份目录**

1. 通过任何一种ASCII文本编辑器，打开数据库服务器端的**dmconfig.ini**文件。
2. 找到要设置的数据库配置项。
3. 将**DB_BkDir**关键字更改为一个包含已存在目录的字符串，用来设置备份目录。
4. 重新启动数据库，使用新的备份目录。

使用DMSQL在线设置备份目录

数据库运行期间，可通过存储过程SetSystemOption更改备份目录，语法如下所示：

➲ **语法**

```
CALL SETSYSTEMOPTION('bkdir', 'path')
```

*path*为新的备份目录全路径，字符串长度不得超过256个字符。

➲ **示例**

执行下面的dmSQL命令，改变备份路径到：
E:/storage/database/backup/WebDB

```
dmSQL> CALL SETSYSTEMOPTION('bkdir', 'E:/storage/database/backup/WebDB');
```

使用JSERVER MANAGER工具设置备份目录

如果数据库处于离线状态，您可以使用JServer Manager图形化工具来设置离线备份目录。JServer Manager将自动更改**dmconfig.ini**配置文件中的 **DB_BkDir**关键字。当您启动数据库时，备份服务会将此目录作为备份目录。如果数据库处于在线状态，JServer Manager工具可以设置是使用动态设置立即更改备份目录，还是当数据库做交互备份时延迟到重启数据库时更改备份目录。无论您的数据库处于何种状态，JServer Manager工具都将在新的备份目录中复制历史备份文件。

➲ **使用JServer Manager工具离线设置备份目录**

1. 从主控制台或**数据库**下拉菜单中选择**启动数据库**选项，**启动数据库**对话框将出现。
2. 从**数据库名称**下拉列表框中选择将要更改设置的数据库。
3. 点击**设置**按钮，将出现**启动数据库高级设置**窗口。
4. 点击**备份**标签页。
5. 勾选**启动备份服务**选框。
6. 点击**备份文件目录**旁边的编辑按钮，在出现的对话中中直接输入一个备份目录或点击旁边的浏览按钮 ... 选择一个备份目录。
7. 点击**保存**按钮。
8. 点击**取消**按钮返回到**启动数据库**窗口。

➲ **使用JServer Manager工具在线设置备份目录**

1. 从数据库下拉菜单中选择**动态设置**选项。
2. **动态设置**窗口的**备份**页面将出现。
3. 从**数据库名称**下拉列表中选择一个数据库。
4. **登录**对话框将出现。
5. 点击**确定**按钮，您要登录的数据库将出现在**动态设置**对话框窗口的**数据库名称**字段中。
6. 如果要在以下会话中应用更新设置，请选中**写到dmconfig.ini**选项框。

7. 如果只需要将更新设置应用到当前的会话，请不勾选**写到dmconfig.ini**选项。
8. 点击**备份文件目录**旁边的编辑按钮，在出现的对话中中直接输入一个备份目录或点击旁边的浏览按钮 ... 选择一个备份目录。
9. 点击**动态设置**窗口底部的**确定**按钮。

注意 *如果**DB_BkDir**键字没有出现在要启动备份服务的数据库配置项中，JServer Manager工具将会自动将该关键字添加到相应的位置。*

设置旧文件目录

旧目录是一个或一组用于保存最后一次备份之前的备份序列的目录（最多32个），为了防止数据库和备份文件在发生介质故障时都丢失，您最好不要在存放数据库文件的磁盘中创建旧文件目录。

旧目录可以通过**dmconfig.ini**配置文件中的**DB_BkOdr**关键字来设定，如果您没有指定它，备份服务将丢弃先前的备份序列。

使用DMCONFIG.INI配置文件设置旧文件目录

您可以直接更改dmconfig.ini配置文件中的DB_BkOdr关键字来设置旧文件目录。当您启动数据库时，备份服务将把此目录作为旧文件目录。如果数据库处于在线状态。只有在您重新启动数据库时，对DB_BkOdr关键字的更改才会生效。

使用dmSQL在线设置设置旧文件目录

数据库运行期间，可通过存储过程**SetSystemOption**更改旧文件备份目录，语法如下所示：

```
CALL SETSYSTEMOPTION('bkodr', 'path')
```

*path*为新的旧文件备份目录全路径，字符串长度不得超过256个字符。

➲ **示例**

在dmSQL命令行中输入以下命令，将旧文件备份路径更改为"*E:\storage\database\backup\WebDB*"。

```
dmSQL> CALL SETSYSTEMOPTION('bkodr', 'E:\storage\database\backup\WebDB');
```

使用JSERVER MANAGER工具设置旧文件目录

如果数据库处于离线状态，您可以通过JServer Manager图形化工具来设置旧文件目录的位置，JServer Manager工具将自动更改**dmconfig.ini**配置文件中的**DB_BkOdr**关键字。当您启动数据库时，备份服务会将此目录作为旧文件目录。如果数据库处于在线状态，在您重新启动数据库时，JServer Manager工具可以设置是立即更改旧文件目录还是延迟到重新启动数据库时更改旧文件目录。

➲ **使用JServer Manager工具离线设置旧备份目录**

1. 从主控制台或**数据库**下拉菜单中选择**启动数据库**选项，**启动数据库**对话框将出现。
2. 从**数据库名称**下拉列表框中选择将要更改设置的数据库。
3. 点击**设置**按钮，将出现**启动数据库高级设置**窗口。
4. 点击**备份**标签页，选中**启动备份服务器**选框。
5. 点击**前次完整备份目录**字段旁边的编辑按钮，在出现的对话框中为最后一次完整备份文件的存放位置直接输入一个路径或点击旁边的浏览按钮…选择一个目录。
6. 点击**保存**按钮。
7. 点击**取消**按钮返回到**启动数据库**窗口。

注意 *如果**DB_BkODr**键字没有出现在要启动备份服务的数据库配置项中，JServer Manager工具将会自动将该关键字添加到相应的位置。*

差异备份设置

差异备份的时间计划用于指定备份服务执行在线增量备份的时间。 此时间计划由两部分组成：初始备份时间和间隔时间。初始备份时间确定了备份服务执行第一次差异备份的日期和时间；间隔时间确定了两次差异备份相隔的时间。

初始完整备份时间可通过dmconfig.ini配置文件中的**DB_FBkTm**关键字来设置，它的格式为：YY/MM/DD HH:MM:SS。初始备份时间不存在默认值，但如果您使用JServer Manager工具来设置备份服务，JServer Manager工具会为您提供一个默认值并且将此值写入dmconfig.ini配置文件中。

间隔时间可通过**dmconfig.ini**配置文件中的**DB_DBkTv**关键字来设置，差异备份第一次执行的时间是**DB_FBkTm + DB_DBkTv**。时间间隔**DB_DBkTv**的值的格式为：D-HH:MM:SS。间隔时间不存在默认值，但如果您使用JServer Manager工具来设置备份服务，JServer Manager工具会为您提供一个1-00:00:00默认值，并且将此值写入**dmconfig.ini**配置文件中。

最后，执行完整备份后，可保留的差异备份数的最大值由关键字**DB_DbKmx**指定，当差异备份的数目超过关键字**DB_DbKmx**的值时，备份服务会删除最老的那个差异备份。

使用DMCONFIG.INI 配置文件更改差异备份设置

如果数据库处于离线状态，您可以直接使用**dmconfig.ini**配置文件中的**DB_FBkTm**和**DB_DBkTv**关键字来设置备份的时间计划。当您再次启动数据库时，备份服务将为差异备份的时间计划应用这些设置。如果数据库处于在线状态，只有在您重新启动数据库后，对**DB_FBkTm**和**DB_DBkTv**关键字的更改才会生效。

➲ **通过dmconfig.ini设置备份时间计划**

1. 使用任何一种**ASCII**文本编辑器，打开数据库服务器上的**dmconfig.ini**文件。
2. 找到要设置备份时间计划的数据库配置项。
3. 改变**DB_FBkTm**关键字为YY/MM/DD HH:MM:SS格式的日期时间值。
4. 改变**DB_DBkTv**关键字为DDDDD-HH:MM:SS格式的时间间隔值。
5. 重新启动数据库，开始使用新的备份时间计划。

使用DMSQL更改差异备份设置

可用存储过程SetSystemOption命令激活备份服务，命令如下：

```
dmSQL> CALL SETSYSTEMOPTION('BKSVR','1');
```

激活备份服务后，调用系统存储过程**SetSystemOption**开始进行差异备份。

```
dmSQL> CALL SETSYSTEMOPTION('STARTBACKUP','3')
```

可用如下语法更改差异备份的时间间隔：

```
CALL SETSYSTEMOPTION('dbktv', 'Interval')
```

使用 JSERVER MANAGER工具更改差异备份设置

如果数据库处于离线状态，您可以使用JServer Manager图形化工具来设置差异备份的时间计划。JServer Manager工具可以自动更改**dmconfig.ini**配置文件中的关键字**DB_FBkTm**和**DB_DBkTv**的值。在数据库再次启动时，备份服务器会将这些设置作为新的差异备份时间计划。如果数据库处于在线状态，JServer Manager工具可以立即更改备份时间计划，也可以延迟到重新启动数据库时，更改备份时间计划。有关如何使用JServer Manager设置增量备份的时间计划，请参考*服务器管理工具用户手册*。

设置增量备份

增量备份的时间计划用于指定备份服务执行一个在线增量备份的时间。此时间计划由两部分组成：初始备份时间和间隔时间。初始备份时间确定了备份服务执行第一次增量备份的日期和时间；间隔时间确定了两次增量备份相隔的时间。

当数据库处于在线备份时，您可以将增量备份的时间计划和日志触发值结合起来使用。也就是说，备份数据库可以根据时间计划，也可以通过设定的日志文件填充系数自动备份数据库。如果您没有指定增量备份的时间计划，备份服务将无法依据时间计划来备份数据库。

初始备份时间可通过**dmconfig.ini**配置文件中的**DB_BkTim**关键字来设置，它的格式为：YY/MM/DD HH:MM:SS。初始备份时间不存在默认值。

间隔时间可通过**dmconfig.ini**配置文件中的**DB_Bkltv**关键字来设置，它的格式为：D-HH:MM:SS。间隔时间不存在默认值，但如果您使用JServer Manager工具来设置备份服务，JServer Manager工具会为您提供一个1-00:00:00默认值，并且将此值写入**dmconfig.ini**配置文件中。

DBMaster提供了几种不同的设置增量备份时间计划的方式，您可以根据据您的数据库在线与否，选择直接更改配置文件的方式或者使用JServer Manager图形化工具。

使用DMCONFIG.INI配置文件更改增量备份设置

如果数据库处于离线状态，您可以直接使用**dmconfig.ini**配置文件中的**DB_BkTim**和**DB_Bkltv**关键字来设置备份的时间计划。当您启动数据库时，备份服务将使用这些设置的增量备份时间计划。如果数据库处于在线状态，只有在您重新启动数据库后，对**DB_BkTim**和**DB_Bkltv**关键字的更改才会生效。

➲ **通过dmconfig.ini配置文件设置时间计划**

1. 用任何一种ASCII文本编辑器打开**dmconfig.ini**文件。
2. 找到要更改备份时间计划的数据库配置项。
3. 改变**DB_BkTim**关键字为YY/MM/DD HH:MM:SS格式的日期时间值。
4. 改变**DB_Bkltv**关键字为DDDDD-HH:MM:SS格式的时间间隔值。
5. 重新启动数据库，开始使用新的备份时间计划。

使用DMSQL更改增量备份设置

在数据库运行时，存储过程SetSystemOption用于更改增量备份的启动时间和时间间隔。更改增量备份启动时间的语法如下：

```
CALL SETSYSTEMOPTION ('bktim', 'StartTime')
```

更改增量备份时间间隔的语法如下：

```
CALL SETSYSTEMOPTION ('bkitv' ,  'Interval' )
```

StartTime是首次启动增量备份的时间，格式为YY:MM:DD HH:MM:SS. Interval是增量备份的时间间隔，格式为D-HH:MM:SS。

当备份服务激活时，可调用系统存储过程**SetSystemOption** 来执行一个增量备份。

```
dmSQL> CALL SETSYSTEMOPTION(' STARTBACKUP', '2')
```

➲ **示例**

在dmSQL命令行中输入以下命令，设置增量备份时间间隔为1小时。

```
dmSQL> CALL SETSYSTEMOPTION('bkitv' , '0-1:00:00' );
```

使用JSERVER MANAGER工具更改增量备份的设置

如果数据库处于离线状态，您可以使用JServer Manager图形化工具来设置增量备份的时间计划。JServer Manager工具可以自动更改**dmconfig.ini**配置文件中的关键字**DB_BkTim**和**DB_BkItv**的值。在数据库再次启动时，备份服务器会将这些设置作为新的增量备份时间计划。如果数据库处于在线状态，JServer Manager工具可以立即更改备份时间计划，也可以延迟到重新启动数据库时，更改备份时间计划。有关如何使用JServer Manager设置增量备份的时间计划，请参考*服务器管理工具用户手册*。

➲ **使用JServer Manager工具离线设置备份时间计划**

1. 从主控制台或**数据库**下拉菜单中选择**启动数据库**选项，**启动数据库**对话框将出现。
2. 从**数据库名称**下拉列表框中选择将要更改设置的数据库。
3. 点击**设置**按钮，将出现**启动数据库高级设置**窗口。
4. 点击**备份**标签页。
5. 要启动备份服务器，请选中**启动备份服务器**选框。
6. 在**增量备份的启动时间**字段中输入增量备份的日期和时间。
7. 在**启动增量备份时间间隔**字段中输入两个成功完整备份之间的天数、小时、分钟和秒。
8. 点击**保存**按钮。
9. 点击**取消**按钮返回到**启动数据库**窗口。

➲ **使用JServer Manager工具在线设置备份时间计划**

1. 从数据库下拉菜单中选择**动态设置**选项。
2. **动态设置**窗口将出现。
3. 从**数据库名称**下拉列表中选择一个数据库。
4. **登录**对话框将出现。
5. 点击**确定**按钮，您要登录的数据库将出现在**动态设置**对话框窗口的**数据库名称**字段中。
6. 如果要在以下会话中应用更新设置，请选中**写到dmconfig.ini**选项框。
7. 如果只需要将更新设置应用到当前的会话，请清除**写到dmconfig.ini**选项。
8. 在**开始时间**字段中输入增量备份的开始的日期和时间。
9. 在**时间间隔**字段中输入两个成功增量备份之间的日期、小时、分钟和秒。
10. 点击**动态设置**窗口底部的**确定**按钮。

设置日志触发值

日志触发值指定日志文件的填充系数，当数据库处于在线备份时，您可以将增量备份的时间计划和日志触发值结合起来使用。也就是说，您可以根据时间计划来备份数据库，也可以通过设定日志文件的填充系数来备份数据库。

日志触发值可以通过**dmconfig.ini**配置文件中的**DB_BkFul**关键字来设定，此**DB_BkFul**关键字的值可以是一个50-100之间的整数，也可以设为0。50-100之间的数值表示备份服务在执行备份之前，日志文件填充的百分比。数值0表示一个日志文件填充完成引起备份服务去执行一个备份操作。设置数值0和数值100所达到的效果是一样的，因为都是在一个日志文件填充完成（100%填满）后引起一个备份操作。如果您没有指定日志触发的值，备份服务将使用默认值***90***。

DBMaster提供了几种不同的设置日志触发值的方式。您可以根据您的数据库在线与否，选择直接更改配置文件的方式或者使用JServer Manager图形化工具。

使用DMCONFIG.INI配置文件更改日志触发值

如果数据库处于离线状态，您可以直接使用**dmconfig.ini**配置文件中的**DB_BkFul**关键字来设置日志触发值。当您启动数据库时，备份服务将为日志触发值应用此设置。如果数据库处于在线状态，只有在您重新启动数据库时，对**DB_BkFul**关键字的更改才会生效。

➲ **通过配置文件dmconfig ini设置日志触发值**

1. 用任何一种ASCII文本编辑器打开**dmconfig.ini**文件。
2. 找到要更改日志触发值的数据库配置项。
3. 将关键字**DB_BkFul**更改成一个50-100之间的整数，或者将它设为0。
4. 使用新的日志触发值来重新启动数据库。

使用DMSQL更改日志触发值

在数据库运行时，存储过程SetSystemOption可用于更改日志触发值。更改日志触发值的语法如下：

```
CALL SETSYSTEMOPTION ('bkful', 'n')
```

n的取值范围为0或50-100之间的整数。将n设为0表示日志文件的填充满后将引起一个备份操作，将n设为50-100之间的整数表示在备份服务之前，日志文件的填充百分数。

➲ **示例**

执行下面的dmSQL命令，设置日志触发值为75% ：

```
dmSQL> CALL SETSYSTEMOPTION('bkful', '75');
```

使用JSERVER MANAGER工具更改日志触发值

如果数据库处于离线状态，您可以使用JServer Manager图形化工具来设置日志触发值。JServer Manager工具可以自动更改**dmconfig.ini**配置文

件中的关键字**DB_BkFul**的值。在数据库再次启动时，备份服务器会将此设置作为新的日志触发值。如果数据库处于在线状态，在您重新启动数据库时，JServer Manager工具可以立即更改日志触发值，也可以延迟到重新启动数据库时更改日志触发值。有关如何使用JServer Manager工具来设置日志触发值，请参考*服务器管理工具用户手册*。

➲ **使用JServer Manager工具离线设置日志触发值**

1. 从主控制台或**数据库**下拉菜单中选择**启动数据库**选项，**启动数据库**对话框将出现。
2. 从**数据库名称**下拉列表框中选择将要更改设置的数据库。
3. 点击**设置**按钮，将出现**启动数据库高级设置**窗口。
4. 点击**备份**标签页。
5. 设置为当日志文件填充到设置的百分比时，增量备份自动执行的方式。在**启动增量备份的日志百分比**字段中可以进行如下选择：
   - 选择**当日志文件满时开始备份**选项，当任何一个日志文件填满时，增量备份会自动执行。
   - 在**50-100**字段中输入一个介于50—100之间的整数，表示当任何一个日志文件填充到设定的百分比时，增量备份将会自动执行。
6. 点击**保存**按钮。
7. 点击**取消**按钮返回到**启动数据库**窗口。

➲ **使用JServer Manager工具在线设置日志触发值**

1. 从数据库下拉菜单中选择**动态设置**选项。
2. **动态设置**窗口将出现。
3. 从**数据库名称**下拉列表中选择一个数据库。
4. **登录**对话框将出现。
5. 点击**确定**按钮，您要登录的数据库将出现在**动态设置**对话框窗口的**数据库名称**字段中。

6. 如果要在以下会话中应用更新设置，请选中**写到dmconfig.ini**选项框。

7. 如果只需要将更新设置应用到当前的会话，请不勾选**写到dmconfig.ini**选项。

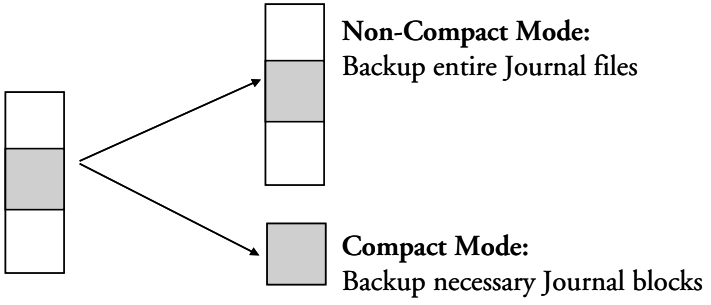
8. 当日志文件已经填充到设置的**百分比**时，增量备份可自动执行。

- 在**日志完整百分比**区域中，选择**用户默认设置**选项，当任何一个日志文件填满时，增量备份会自动执行。
- 在**50-100**字段中输入一个介于50—100之间的整数，表示当任何一个日志文件填充到设定的百分比时，增量备份将会自动执行。

9. 点击**动态设置**窗口底部的**确定**按钮。

设置压缩备份模式

执行在线增量或差异备份时，压缩备份模式用于确定备份服务是备份完整日志文件，还是只备份完整日志块。此功能是很有必要的，因为并非每一个日志块都包含恢复一个数据库所需要的数据。因此备份服务在执行备份时，只需备份必要的日志块。也就是说，设置压缩备份模式之后会节省存储空间。但是在作备份还原时，可能会花费较多时间。



压缩备份模式可以通过**dmconfig.ini**配置文件中的**DB_BkCmp**关键字来设定，此关键字的值可以为0或1。设为1表示启动压缩备份模式，设为0表示关闭压缩备份模式。如果您没有指定压缩备份模式的值，备份服务器将使用默认值***1***(启动压缩模式)。

DBMaster提供了几种不同的压缩备份模式的方法。您可以根据您的数据库在线与否，选择直接更改配置文件的方式或者使用JServer Manager图形化工具。

使用DMCONFIG.INI配置文件来设置压缩备份模式

如果数据库处于离线状态，可以直接使用**dmconfig.ini**配置文件中的**DB_BkCmp**关键字来设置备份服务器的压缩备份模式。当您启动数据库时，备份服务器会使用新设置的压缩备份模式。如果数据库处于在线状态，只有在您重新启动数据库时，对**DB_BkCmp**关键字的更改才会生效。

➲ **使用dmconfig.ini配置文件设置压缩备份模式**

1. 用任何一种ASCII文本编辑器打开**dmconfig.ini**文件。
2. 找到要更改压缩备份模式的数据库配置项。
3. 将**DB_BkCmp**关键字的值设为1，表示压缩备份模式；或者设为0，不压缩备份模式。
4. 重启数据库，新的压缩备份模式将生效。

使用DMSQL设置压缩备份模式

在数据库运行时，存储过程**SetSystemOption**可用于更改压缩备份模式。并不是日志文件中的每个日志块都需要进行备份。如果设置关键字**DB_BkCmp**值为1，备份服务器将只备份需要的日志块以节省磁盘空间。更改压缩备份模式的命令如下：

```
dmSQL> CALL SETSYSTEMOPTION('bkcmp', '1');
```

使用JSERVER MANAGER工具设置备份压缩模式

如果数据库处于离线状态，您可以使用JServer Manager图形化工具来设置压缩备份模式。JServer Manager工具可以自动更改**dmconfig.ini**配置文件中的关键字**DB_BkCmp**的值。在数据库再次启动时，备份服务器会将此设置作为新的压缩备份模式。如果数据库处于在线状态，在您重新启动数据库时，JServer Manager工具可以立即更改压缩备份模式，也可

以延迟到重新启动数据库时更改压缩备份模式。有关如何使用JServer Manager工具来设置压缩备份模式，请参考*服务器管理工具用户手册*。

➲ **使用JServer Manager工具离线设置压缩备份模式**

1. 从主控制台或**数据库**下拉菜单中选择**启动数据库**选项，**启动数据库**对话框将出现。
2. 从**数据库名称**下拉列表框中选择将要更改设置的数据库。
3. 点击**设置**按钮，将出现**启动数据库高级设置**窗口。
4. 点击**备份**标签页。
5. 要启动备份服务器，请选中**启动备份服务**选框。
6. 要启用压缩备份服务，请选中**压缩备份文件**选框。
7. 点击**保存**按钮。
8. 点击**取消**按钮返回到**启动数据库**窗口。

➲ **使用JServer Manager工具在线设置压缩备份模式**

1. 从数据库下拉菜单中选择**动态设置**选项。
2. **动态设置**窗口将出现。
3. 从**数据库名称**下拉列表中选择一个数据库。
4. **登录**对话框将出现。
5. 点击**确定**按钮，您要登录的数据库将出现在**动态设置**对话框窗口的**数据库名称**字段中。
6. 如果要在以下会话中应用更新设置，请选中**写到dmconfig.ini**选项框。
7. 如果只需要将更新设置应用到当前的会话，请不勾选**写到dmconfig.ini**选项。
8. 要启用压缩备份服务，请选中**压缩备份文件**选框。
9. 点击**确定**按钮返回到**动态设置**窗口。

完整备份的时间计划

完整备份的时间计划用于指定备份服务执行在线完整备份的时间，此时间计划由两部分组成：初始备份时间和间隔时间。初始备份时间确定了备份服务执行首次完整备份的日期和时间；间隔时间确定了两次完整备份相隔的时间。

您可以将完整\差异备份的时间计划和增量备份结合起来使用。如果您没有指定完整备份的时间计划，备份服务将不使用时间计划表来执行完整备份。

初始时间可以通过**dmconfig.ini**配置文件中的**DB_FBkTm**关键字来指定。它的格式为：YY/MM/DD HH:MM:SS，初始时间不存在默认值。

间隔时间可以通过**dmconfig.ini**配置文件中的**DB_FBkTv**关键字来指定。它的格式为：D-HH:MM:SS，间隔时间不存在默认值。

最后，您可以通过关键字**DB_BkChk**定义在执行完整备份和差异备份之前是否检查数据库，通过关键字**DB_ BkRTs**定义备份服务器在执行完整备份时，是否备份只读表空间文件。您可以在**dmconfig.ini**配置文件中设置关键字**DB_BkChk**和**DB_ BkRTs**来启用或禁止该功能。若数据库正在运行，您可以使用系统存储过程**SetSystemOption**更改BKCHK和BKRTS来启用或禁止该功能。

使用DMCONFIG.INI配置文件设置完整备份模式

如果数据库处于离线状态，可以直接使用**dmconfig.ini**配置文件中的**DB_FBkTm**和**DB_FBkTv**关键字来设定备份服务的完整备份时间计划。当您启动数据库时，备份服务将使用新设置的完整备份的时间计划。如果数据库处于在线状态，只有在您重新启动数据库时，对**DB_FBkTm**和**DB_FBkTv**的更改才会生效。

➲ **通过配置文件dmconfig.ini设置完整备份的时间计划**

1. 通过任何一种ASCII文本编辑器，打开服务器端的配置文件**dmconfig.ini**。
2. 找到要更改完整备份时间计划的数据库配置项。

3. 改变关键字**DB_FBkTm**为YY/MM/DD HH:MM:SS 格式，关键字**DB_FBkTv**为D-HH:MM:SS格式。
4. 重新启动数据库，开始使用新的完整备份时间计划。

使用dmSQL设置完整备份的时间计划

在数据库运行时，存储过程**SetSystemOption**用于更改完整备份的启动时间和时间间隔。更改完整备份启动时间的语法如下：

```
CALL SETSYSTEMOPTION('fbktm', 'StartTime')
```

更改完整备份时间间隔的语法如下：

```
CALL SETSYSTEMOPTION('fbktv', 'Interval')
```

*StartTime*是首次启动完整备份的时间，格式为YY:MM:DD HH:MM:SS. *Interval*是完整备份的时间间隔，格式为D-HH:MM:SS。

激活备份服务器后，调用系统存储过程**SetSystemOption**开始进行完整备份：

```
dmSQL> CALL SETSYSTEMOPTION('STARTBACKUP','1')；
```

➲ **示例**

在dmSQL命令行中输入以下命令，设置完整备份时间间隔为1小时。

```
dmSQL> CALL SETSYSTEMOPTION('fbktv', '0-1:00:00');
```

使用JSERVER MANAGER工具设置完整备份模式

您可以使用JServer Manager工具来设置完整备份的时间计划。JServer Manager将自动更改**dmconfig.ini**配置文件中的**DB_FBkTm**和**DB_FBkTv**关键字。在您启动数据库时，备份服务会将此设置作为新的完整备份时间计划。

➲ **使用JServer Manager工具设置完整备份时间计划：**

1. 从主控制台或**数据库**下拉菜单中选择**启动数据库**选项，**启动数据库**对话框将出现。
2. 从**数据库名称**下拉列表框中选择将要更改设置的数据库。
3. 点击**设置**按钮，将出现**启动数据库高级设置**窗口。
4. 点击**备份**标签页。

5. 要启动备份服务器，请选中**启动备份服务**选框。
6. 在**完整备份的启动时间**字段中设置完整备份的日期和时间。
7. 在**启动完整备份的时间计划**的天数和时间字段中输入两次成功完整备份之间的天数、小时、分钟和秒。
8. 点击**保存**按钮。
9. 点击**取消**按钮返回到**启动数据库**窗口。

文件对象的备份模式

在完整备份期间，数据库管理员可以通过设置文件对象的备份模式来决定备份服务是否备份文件对象。您可以只备份系统文件对象，也可以备份系统和用户文件对象。

可以通过多种方式设置文件对象的备份模式。在数据库的启动过程中，关键字**DB_BkFoM**确定了文件对象的备份模式。在数据库的运行期间，我们还可以使用dmSQL或JServer Manager图形化工具来更改它。

通过指定**DB_BkOdr**关键字，备份服务会将先前备份的文件移到旧的备份目录中。

启动文件对象备份模式会使数据库花更多的时间去执行一个完整备份，这取决于数据库中有多少文件对象。一个完整备份包括：(1) 如果设置了**DB_BkOdr**关键字，将复制先前的完整备份。 (2) 复制所有数据库文件。(3) 复制所有日志文件。 (4) 如果设置了**DB_BkFoM**关键字,将复制所有文件对象。为了避免备份失败，请确保备份目录所在的磁盘有足够空间可以利用，备份文件的目录可通过**DB_BkDir**(和**DB_BkOdr**)关键字来指定。

一个完整备份执行时，文件对象复制到在备份目录中创建的FO目录中。当文件对象复制到备份文件的目录中时，它们将被重新命名。**/FO**子目录中的文件名是以FO开头的，后面跟随一个十位数的序列数。所有备份文件对象都是.BAK扩展名。源文件名和路径与备份文件名之间的映射都被记录到映射文件**dmFoMap.his**中。

备份文件对象的映射文件

文件对象的映射文件dmFoMap.his创建在"*DB_BkDir/FO*"目录中。它是一个纯粹的ASCII文本文件，用于记录源文件名和备份文件名。格式如下：

➲ **语法**

```
Database Name: MYDB
Begin Backup FO Time: 2001.5.13 2:33
FO Backup Directory: /DBMaster/mydb/backup/FO (i.e. DB_BkDir/FO)
[Mapping List]
s, fo0000000000.bak, "/DBMaster/mydb/fo/ZZ000001.bmp"
u, fo0000000001.bak, "/home2/data/image.jpg"
....
s, fo00000002345.bak, "/DBMaster/mydb/fo/ZZ00AB32.txt"
```

" [Mapping List] "之前的内容是为用户参考提供的一个描述。"[Mapping List]" 之后的每一行代表一个记录，用于显示文件对象类型(s = 系统文件对象, u = 用户文件对象)，*/fo*子目录中的新文件以及它的源文件名和路径。此映射文件对恢复文件对象是很有必要的。

使用DMCONFIG.INI配置文件设置文件对象的备份模式

配置文件中的关键字**DB_BkFoM**确定了文件对象的备份模式：

DB_BkFoM = 0：不备份文件对象

DB_BkFoM = 1：只备份系统文件对象

DB_BkFoM = 2：备份系统和用户文件对象

如果**DB_BkFoM** = 1或2，备份服务器将把所有的文件对象复制到备份目录下的/fo子目录中。下面的时间计划为完整备份的时间计划。

➲ **示例**

在**dmconfig.ini**配置文件中，设定的文件对象备份参数。

```
[MyDB]
DB_BkSvr = 1                        ; starts the backup server
DB_FBKTm = 01/05/01 00:00:00        ; begins at midnight, May 1, 2001.
DB_FBkTv = 1-00:00:00               ; interval is once every day.
DB_BkDir = /home/dbmaster/backup    ; backup directory
DB_BkFoM = 2                        ; backup both system and user file objects
```

因为备份模式设置为2, 所以备份服务器将把所有外部文件(用户文件对象)和系统文件对象复制到*/home/DBMaster/backup/FO*目录下。如果**FO**子目录不存在，那么备份服务器将会自动创建它。

使用DMSQL设置文件对象备份模式

当数据库运行时，存储过程SetSystemOption命令可用于更改文件对象的备份模式。它的语法如下：

```
CALL SETSYSTEMOPTION ('bkfom', 'n')
```

n的取值可为0、1、2，将n设为0代表关闭文件对象的备份；将n设为1代表备份服务器在完整备份期间，可以备份所有的系统文件对象；将n设为2代表备份服务器在完整备份期间，可以备份所有的系统和用户文件对象。

➲ **示例**

为了执行一个系统和用户文件对象的完整备份，您可以在dmSQL的命令提示中输入以下命令行来配置备份服务。

```
dmSQL> CALL SETSYSTEMOPTION('bkfom', '2');
```

使用JSERVER MANAGER工具设置文件对象备份模式

设置**文件对象的备份模式**将影响文件对象在完整备份过程中的复制方式。如果您选择**不备份文件对象模式**，那么文件对象将无法被复制。如果您选择**只备份系统文件对象**模式，那么系统文件对象将在自动完整备份过程中被复制。如果您选择**备份系统和用户文件对象**模式，那么系统文件对象和用户文件对象在自动完整备份过程中都将被复制。

➲ **数据库启动期间设置文件对象备份模式：**

1. 点击**启动数据库**窗口的**设置**按钮，将出现**启动数据库高级设置**窗口。
2. 点击**启动数据库高级设置**窗口的**备份**标签页。
3. 选中**启动备份服务**选框。
4. 通过备份服务器执行完整备份：

a) 在**备份文件目录**字段中输入路径或点击该字段旁的浏览按钮 ... 选择一个备份目录。

b) 在**完整备份的启动时间**字段中输入完整备份的启动日期和时间。

c) 在**启动完整备份的时间间隔**字段中输入两次完整备份之间的天数、小时、分钟和秒数。

5. 选择文件对象的备份类型：

a) 选择**不备份文件对象**，可阻止文件对象的备份。

b) 选择**只备份系统文件对象**将只备份系统文件对象。

c) 选择**备份系统和用户文件对象**将备份所有的文件对象。

6. 点击**保存**按钮。

7. 点击**取消**按钮返回到**启动数据库**窗口，并点击**开始**按钮启动数据库。

存储过程的备份模式

存储过程的备份模式，有助于数据库管理员决定备份服务器在进行完整备份时是否备份ESQL存储过程和JAVA存储过程。

设置存储过程备份模式的方法有多种。当数据库启动时，可以通过配置文件中的关键字**DB_BkSPm**来设置存储过程的备份模式；在数据库运行期间，还可以通过dmSQL命令或JServer Manager图形化工具来更改其备份模式。

备份服务器可以将之前存储过程的备份移动到由**DB_BkOdr**指定的旧 目录下。

开启存储过程的备份会导致数据库在完整备份时占用更多的时间，这取决于数据库中存在的文件对象数量。一个完整备份包括：(1) 如果设置了**DB_BkOdr**，将复制先前的完整备份；(2) 复制所有数据库文件；(3) 复制所有日志文件；(4) 如果设置了**DB_BkFoM**，将复制所有文件对象；（5）如果设置了**DB_BkSPm**，将复制ESQL存储过程和JAVA存储过程。为了避免备份失败，请确保备份目录所在的磁盘有足够空间可以利用，备份文件的目录可通过**DB_BkDir**关键字来指定。

存储过程被复制到执行完整备份时，创建的备份目录的子目录**SP**下。当文件对象被复制到存储过程的备份目录下，存储过程会被重新命名。复制的存储过程备份信息被记录在文件**dmSpBk.his**中。

备份存储过程的信息文件

备份信息由三部分构成：数据库名称、备份时间、用来记录已备份行的备份列表。备份信息文件**dmSpBk.his**位于由关键字**DB_BkDir**所指定目录的子目录**SP**下。该文件是一个纯ASCII码的文本文件，用来记录所复制的存储过程备份信息，其格式如下所示：

```
Database Name: MYDB
Begin Backup SP time: 2014/06/20 09:13:06
[Backup List]

ESQLSP, SYSADM, A4, A4SYSADM.dll, A4SYSADM.ec
....
JAR, "", "", employee.jar, ""
```

"[Backup List]"之前的内容仅为供用户参考的说明。"[Backup List]"之后的每一行均为一条记录，显示存储过程的类型、存储过程的拥有者、存储过程的名称、存储过程的对象文件名称以及相应的源文件名称。该备份信息文件是恢复存储过程时的必要文件。

使用dmconfig.ini配置文件设置存储过程的备份模式

配置文件中的关键字**DB_BkSPm**可以设置存储过程的备份模式：

- **DB_BkSPm** = 0: 不备份ESQL存储过程和JAVA存储过程
- **DB_BkSPm** = 1: 备份ESQL存储过程和JAVA存储过程

➲ **示例**

在**dmconfig.ini**配置文件中，设定备份存储过程的参数。

```
[MyDB]
DB_BkSvr = 1                        ; starts the backup server
DB_FBkTm = 14/05/01 00:00:00        ; begins at midnight, May 1, 2014
DB_FBkTv = 1-00:00:00               ; interval is once every day
DB_BkDir = /home/dbmaker/backup     ; backup directory
DB_BkSPm = 1                        ; backup all stored procedures
```

DB_BkSPm的值为1时，备份服务器将备份ESQL存储过程和JAVA存储过程，并保存到由关键字**DB_BkDir**指定目录的子目录**SP**下。如果子目录**SP**不存在，备份服务器会自动创建。

使用dmSQL设置存储过程的备份模式

当数据库运行时，存储过程**SetSystemOption**用于更改存储过程的备份模式。更改存储过程的语法如下：

```
CALL SETSYSTEMOPTION('BKSPM','n')
```

在上述语法中，*n*的取值可为0或1。如果*n*为0，则表示在进行完整备份时，备份服务器不备份ESQL存储过程和JAVA存储过程；如果*n*为1，则表示备份服务器将备份ESQL存储过程和JAVA存储过程。

➲ 示例

配置备份服务器，使其在备份所有存储过程时执行完整备份。该操作在dmSQL命令中输入的命令如下所示：

```
dmSQL> CALL SETSYSTEMOPTION('BKSPM','1');
```

使用JServer Manager工具设置存储过程的备份模式

无论数据库是否在线，您都可以使用JServer Manager图形化工具来设置备份ESQL存储过程和JAVA存储过程。JServer Manager将自动更改**dmconfig.ini**配置文件中的关键字**DB_ BkSPm**。若数据库处于离线状态，只有在您重新启动数据库时，新的设置才会起效。对于数据库运行期间，如何设置存储过程的备份模式或如何使用JServer Manager工具的**动态设置**来设置存储过程备份模式的详细内容，请参考*服务器管理工具用户手册*。

终止备份服务器

数据库启动后，DBMaster自动开启备份服务。备份服务默认为不激活状态。您可以通过**DB_BkSvr**关键字控制备份服务的状态。**DB_BkSvr**设置为0，不激活备份服务，**DB_BkSvr**设置为1，激活备份服务。如果您想终止备份服务，您可以在**dmconfig.ini**文件中将**DB_BkSvr**关键字设为

0，也可以在数据库启动后使用*call setsystemoption('bksvr','0')*更改BkSvr。

使用DMCONFIG.INI配置文件终止备份服务

如果数据库处于离线状态，您可以直接使用**dmconfig.ini**配置文件中的**DB_BkSvr**关键字来终止备份服务。当您下次启动数据库时，备份服务将不会启动；如果数据库处于在线状态，只有在您的数据库被重新启动时，对**DB_BkSvr**关键字的更改才会生效。

➲ **使用dmconfig.ini配置文件终止备份服务**

1. 用任何一种ASCII文本编辑器打开**dmconfig.ini**文件。
2. 找到要改变备份模式的数据库配置项。
3. 改变**DB_BkSvr**关键字为0，关闭备份服务。
4. 重启数据库。

使用dmSQL终止备份服务器

在数据库运行时，存储过程**SetSystemOption**用于更改备份服务器的状态，更改更改备份服务器状态的语法如下：

```
CALL SETSYSTEMOPTION('bksvr','n')
```

n的取值可为0或1。将n设为0表示不激活备份服务器，将n设为1表示激活备份服务器。

➲ **示例**

在dmSQL命令行中输入以下命令，终止备份服务器。

```
dmSQL> CALL SETSYSTEMOPTION('bksvr', '0');
```

使用JSERVER MANAGER工具终止备份服务器

如果数据库处于离线状态，您可以使用JServer Manager图形化工具终止备份服务器。JServer Manager将自动更改配置文件**dmconfig.ini**中的关键字**DB_BkSvr**的值。在重新启动数据库时，备份服务器将不启动。如果数据库处于在线状态，只有在您重新启动数据库时，对备份服务器的更改才会生效。

➲ **使用JServer Manager工具离线终止备份服务**

1. 从主控制台或**数据库**下拉菜单中选择**启动数据库**选项，**启动数据库**对话框将出现。
2. 从**数据库名称**下拉列表框中选择将要更改设置的数据库。
3. 点击**设置**按钮，将出现**启动数据库高级设置**窗口。
4. 点击**备份**标签页。
5. 要终止备份服务，请清除**启动备份服务**选项框。
6. 点击**保存**按钮。
7. 点击**取消**按钮返回到**启动数据库**窗口。

9.7 备份历史文件

使用备份服务的自动备份功能可将需要备份的日志文件、备份时间和备份路径这些信息存储到备份历史文件中。

备份历史文件的存放位置

备份历史文件是一个文本文件，存储在**dmconfig.ini**文件中**DB_BkDir**关键字的第一个目录下。备份历史文件被创建在在线备份路径下，并且命名为**dmBackup.his**，在恢复数据库的过程中将自动使用此文件，但是离线备份将被记录在**offBackup.his**文件中。

了解备份历史文件

备份历史文件包括很多与备份id、文件名、备份时间和日期有关的信息。DBMaster使用备份历史文件来追踪备份序列，同时保证完整备份、差异备份和增量备份的一致性。

以下是备份历史文件的格式：

```
<backup_id>: file_name -> archive_file_name, time, event
```

以上语法格式表示由于事件，将名为*file_name*的文件复制成名为*archive_file_name*的活动文件。这些事件是一个说明备份的动作或原因的文本字符串，此字符串可以为JOURNAL-FULL, TIME-OUT, ON-LINE-FULL-BACKUP-BEGIN, ON-LINE-FULL-BACKUP或ON-LINE-FULL-BACKUP-END。字符串JOURNAL-FULL表示当日志满时，将执行一个增量备份；字符串TIME-OUT表示当超过时间计划的备份时间时，将执行一个差异或增量备份；字符串ON-LINE-FULL-BACKUPxxxx指它是一个完整备份。

使用备份历史文件

如果日志满载的情况经常发生，您可以降低备份日志的填充系数或缩短它的间隔时间。同样，您也可以通过检测备份历史文件发现是否备份的

时间间隔太短。如果同一个日志文件在备份历史文件中被连续的复制，那么时间间隔就有可能是太短，这样将会浪费磁盘空间，因为每一个文件可能只包含少数的更改日志块。为了避免这种情况的发生，我们可以使用压缩备份模式或延长备份的时间间隔。

如果每次都有很多的日志文件被复制，这就可能是时间间隔太长。这种情况是很危险的，因为在发生磁盘故障时，可能会丢失很多数据。用户可通过缩短备份时间间隔避免此情况的发生。

为了减少恢复介质故障的时间，您可以定期执行一个完整备份而不用考虑备份服务是否在运行。同时这将减少您所需的备份存储的数量。

了解文件对象的备份历史文件

文件对象的备份历史文件为**dmFoMap.his**，设置开启备份文件对象，您可以记录那些已经作过备份的文件对象。**dmFoMap.his**存放于"*<DB_BkDir>\FO*"目录下，它是一个纯ASCII码的文本文件，用于记录源文件名和备份文件名。

以下是文件的格式：

```
Database Name: MYDB
Begin Backup FO Time: 2001.5.13 2:33
FO Backup Directory: /DBMaster/mydb/backup/FO (i.e. DB_BkDir/FO)
[Mapping List]
s, fo0000000000.bak, "/DBMaster/mydb/fo/ZZ000001.bmp"
u, fo0000000001.bak, "/home2/data/image.jpg"
....
s, fo0000002345.bak, "/DBMaster/mydb/fo/ZZ00AB32.txt"
```

第一列中的**s**或**u**分别代表系统文件对象或用户文件对象，第二列给出了备份的文件名，第三列给出了源文件的完整路径和名称。

了解存储过程的备份历史文件

存储过程的备份历史文件为**dmSpBk.his**，将ESQL存储过程和JAVA存储过程的备份参数设为on将可以记录所有备份的存储过程。**dmFoMap.his**存放于由关键字**DB_BkDir**所指定目录的子目录**SP**下，它是一个纯ASCII码的文本文件，用于记录ESQL存储过程和JAVA存储过程的备份信息。

其格式如下所示：

```
Database Name: MYDB
Begin Backup SP time: 2014/06/20 09:13:06
[Backup List]

ESQLSP, SYSADM, A4, A4SYSADM.dll, A4SYSADM.ec
....
JAR, "", "", employee.jar, ""
```

第1列表示存储过程的类型，第2列表示存储过程的拥有者；第3列表示0存储过程的名称；第4列和第5列则分别表示存储过程对象文件的名称和相应的源文件。

9.8 复制数据库的备份

在普通数据库和主数据库端，用户可以执行完整备份、差异备份和增量备份，备份方法如前。在主数据库端，JServerManager不能执行交互式增量备份。此外，在主数据库端执行交互式完整备份时，增量备份文件无法删除。

请注意在主数据库端，可能仍有许多完整备份前的增量备份存储在复制备份序列中。同时，由于完整备份，复制服务器可能不删除增量备份文件。因此，如果下一次备份的间隔时间太长，**DB_BkDir**设置的路径中将存在大量文件。

总之，复制服务器和备份服务器必须配合很好，它们不能互相干扰。一方面，备份不能干扰复制，换句话说，无论是否有完整备份或差异备份正在执行或已执行完，复制服务器均能将所有事务复制到从端。另一方面，复制也不能损坏备份序列。

您可以使用备份序列恢复主数据库。然而，主数据库恢复后，数据库复制也随即被终止。若要继续复制数据库，您必须使用新的主数据库替换从数据库，也就是说，您必须用主数据库文件替换所有从数据库文件。

复制数据库的备份有以下限制：

- 主数据库开启后，BMode和BkSvr必须开启。
- 运行期间，不能更改主从数据库端的BMode、BkSvr、BkDir，例如，调用setsysoption ('bkdir','new-bkdir')将返回错误。
- 在主数据库和从数据库端，**DB_BkDir**应设置为单路径。如果您将**DB_BkDir**设置为多路径，仅第一条路径可用且路径大小被忽略。
- 在主数据库端，不能使用JServerManager执行交互式增量备份。
- 在从数据库端，无法执行完整备份、差异备份和增量备份。

9.9 恢复选项

恢复数据库将重建一个数据库，备份还原会利用数据库的最近一次完整备份，以及备份日志中记录的更改信息来恢复数据库。

分析恢复选项

如果数据库处于NONBACKUP模式，那么发生磁盘故障时，您的操作只能是恢复到最近的完整备份处并且重启数据库，从最后一次的完整备份以来的操作都会丢失，您必须重新执行所有的操作。

如果数据库操作在BACKUP(BACKUP-DATA或BACKUP-DATA-AND-BLOB)模式上，您可以利用几个选项来重建受损的数据库。

恢复的准备工作

在您恢复发生磁盘故障的数据库之前，请考虑以下问题：

- 您想让数据库恢复到什么时间点？

如果您想让数据库恢复到磁盘故障时，那么将备份受损数据库的所有日志文件。这些文件将帮助DBMaster把一个数据库还原到最近的时间点。

- 什么样的文件已经作过备份？

找出最近的完整备份和所有增量备份的复制点。例如，假设您在每月的30号执行一个完整备份，并且每10天执行一个增量备份，每15天执行一个差异备份，如果您的系统在5月25日遭到损坏，您需要4月30日的完整备份，5月15日的差异备份，5月10日和5月20日的增量备份，以及5月25日受损坏的日志文件。利用这些文件，DBMaster会将您的数据库恢复到5月25日受损之前的状态。有效的备份结果是由一组完整备份文件和一系列差异备份、增量备份文件组成，它是恢复数据库的前提条件。在线备份结果由命名为**dmbackup.his**的历史文件决定，离线备份结果由命名为**offbackup.his**的历史文件决定，因此备份的历史文件是非常重要的。所以DBMaster在恢复数据库时，会访问备份历史文件中的信息。

执行恢复

当执行恢复操作时，DBMaster会执行以下步骤：

➲ **恢复数据库**

- 拷贝所有完整备份文件，将数据文件、blob文件和日志文件拷贝至**dmconfig.ini**文件中**DB_DbDir**关键字指定的目录下。该操作将会重写原始的数据库文件。因此在使用恢复工具之前，建议用户手动地将源数据库文件拷贝至其它路径下，至少确定日志文件已经被保存。这样即使恢复失败，仍可以将数据库恢复到当前状态。
- 将差异备份或增量备份文件单独或同时应用到数据库中。

当使用恢复工具时，用户可以指定：

- 在系统配置文件dmconfig.ini中是否恢复数据库选项。如果选择恢复，则应该指定恢复的配置文件dmconfig.ini的完整路径。
- 如果您想使用备份结果将数据库恢复到不同于原始位置的其它位置，可修改数据文件路径中的以下关键字：**DB_DbDir**、**Db_DbFil**、**DB_UsrDb**等。 如果将备份结果转移到其它位置或其它计算机，在恢复数据库时应考虑如下几点：

  a) **dmconfig.ini**文件中数据库名称必须和备份数据库名称一致。

  b) 必要时为数据文件和BLOB文件设置BkDir及其它关键字。

  c) 如果日志文件超过一个，必须对关键字**DB_JnFil**的值进行设定，确保设定值和备份数据库中日志文件的数目一致。

  d) 如果备份文件存放于多个文件夹，关键字**DB_BkDir**的设置值必须包含这些文件夹。文件dmbackup.his必须存放于第一个BkDir中。

注意 *用户可以从存放备份结果的文件夹中复制一个已经存在的**dmconfig.ini**配置文件，并修改相关关键字，配置一个新的**dmconfig.ini**文件。*

- 如果dmBackup.his或offBackup.his不在默认的路径下时，则需要备份历史文件dmBackup.his或offBackup.his的完整路径。
- 恢复时间（RTime）。RTime指明了何时开始恢复数据库，它决定了目前的备份序列是否有效和运用哪一个增量备份文件。用户可以在使用恢复工具时设定恢复时间，或者在系统配置文件dmconfig.ini或备份配置文件dmconfig.ini中添加关键字**DB_RTime**来设定恢复时间。如果RTime未被设定，则将默认为当前时间。

DBMaster提供两种恢复方法：服务器管理工具和命令行工具中的Rollover回滚命令。

更多的服务器管理工具使用方法请参考*服务器管理工具用户手册*，命令行工具中的Rollover回滚命令，请参考*使用Rollover恢复数据库*。

使用Rollover恢复数据库

用户同样可以使用命令行工具中的Rollover来恢复数据库。同服务器管理工具中恢复数据库选项的原理相同。

Rollover的语法如下：

rollover database_name [-i inifile] [-r rtime] [-h hisfile] [-m foMapfile] [-f FOtype] [-t SpResTyp]

方括号中的六个可选参数：

-i　指定配置文件**dmconfig.ini**的完整路径。如果用户指定利用配置文件**dmconfig.ini**恢复数据库，rollover回滚操作将会用指定的配置文件**dmconfig.ini**对应的数据库配置信息取代系统配置文件**dmconfig.ini**中对应的数据库配置信息。否则DBMaster不会恢复配置文件**dmconfig.ini**。

-r　表明数据库恢复的时间。选项r是指定恢复开始时间的第一个方法，第二个方法是在系统配置文件dmconfig.ini中添加关键字**DB_RTime**，或者备份要恢复的数据库配置文件**dmconfig.ini**。如果既没有–r选项，也没有添加关键字**DB_RTime**，恢复时间将默认为当前时间。

-h　给出**dmBackup.his**文件或**offBackup.his**文件的完整路径。默认路径为***DB_BKDIR/dmBackup.his***或***DB_BKDIR/offBackup.his***。

-m　给出**dmFoMap.his**的完整路径。默认路径为***DB_BkDir/FO/dmFoMap.his***。

-f　指定将被恢复的FO文件类型。共有四个值，0代表没有FO文件要被恢复；1代表恢复系统FO文件；2代表恢复用户FO文件；3代表恢复所有FO文件。默认值为***3***。

-t　指定是否恢复存储过程。共有两个值，0代表不恢复存储过程；1代表恢复所有存储过程。默认值为***1***。