IBM® DB2 通用数据库™

数据恢复和高可用性指南与参考

版本 8

S152-0181-00

IBM® DB2 通用数据库™

数据恢复和高可用性指南与参考

版本 8

S152-0181-00

在使用本资料及其支持的产品之前，务必阅读*声明*中的一般信息。

可以在线方式或通过您当地的 IBM 代表订购 IBM 出版物。

- 要以在线方式订购出版物，可访问 IBM 出版物中心（IBM Publications Center），网址为 www.ibm.com/shop/publications/order。
- 要查找您当地的 IBM 代表，可访问 IBM 全球联系人目录（IBM Directory of Worldwide Contacts），网址为 www.ibm.com/planetwide。

在美国或加拿大，要从 “DB2 市场营销和销售中心” 订购 DB2 出版物，请致电 1-800-IBM-4YOU（426-4968）。

目录

关于本书

本书提供了关于 IBM DB2 通用数据库（UDB）备份、复原和恢复实用程序的详细信息，并向您显示如何使用它们。本书还说明了高可用性的重要性并描述了几个平台上的 DB2 故障转移支持。

谁应使用本书

本手册的目标读者是数据库管理员、应用程序员以及其它负责或想了解 DB2 数据库系统上的备份、复原和恢复操作的 DB2 UDB 用户。

我们假设您已熟悉了 DB2 通用数据库、结构化查询语言（SQL）以及 DB2 UDB 所运行的操作系统环境。此手册不包含安装 DB2 的指示信息，该指示信息的内容取决于您的操作系统。

此书的组织方式

包括下列主题:

数据恢复

第 1 章，『制订一个良好的备份与恢复策略』
讨论选择数据库和表空间恢复方法时要考虑的因素，包括备份和复原数据库或表空间，以及使用前滚恢复。

第 1 章，『制订一个良好的备份与恢复策略』
描述 DB2 备份实用程序，它用于创建数据库或表空间的备份副本。

第 3 章，『数据库复原』
描述 DB2 复原用程序，它用于重新构建先前已进行了备份的受损或毁坏的数据库或表空间。

第 4 章，『前滚恢复』
描述 DB2 前滚实用程序，它通过应用记录在数据库恢复日志文件中的事务，恢复数据库。

高可用性

第 5 章，『高可用性和故障转移支持简介』
对 DB2 提供的高可用性故障转移支持的概述。

第 6 章,『AIX 上的高可用性』

讨论在 AIX 上对高可用性故障转移恢复的 DB2 支持，当前通过“高可用性群集多重处理方式”(HACMP) AIX 版“增强的可伸缩性”(ES) 功能部件来实现。

第 7 章,『Windows 操作系统上的高可用性』

讨论在 Windows 操作系统上对高可用性故障转移恢复的 DB2 支持，这目前是通过 Microsoft Cluster Server（MSCS）实现的。

第 8 章,『Solaris 操作环境中的高可用性』

讨论在 Solaris 操作环境中对高可用性故障转移恢复的 DB2 支持，这目前是通过 Sun Cluster 3.0（SC3.0）或 Veritas Cluster Server（VCS）实现的。

附录

附录 A,『如何阅读语法图』

说明语法图中使用的约定。

附录 B,『警告、错误和完成消息』

所提供的信息解释了数据库管理器在已检测到警告或出错状态时生成的消息。

附录 C,『附加 DB2 命令』

描述与恢复相关的 DB2 命令。

附录 D,『附加 API 及相关数据结构』

描述与恢复相关的 API 以及它们的数据结构。

附录 E,『恢复样本程序』

提供包含了与恢复相关的 DB2 API 及嵌入式 SQL 调用的样本程序的代码列表以及如何使用它们的信息。

附录 F,『Tivoli Storage Manager』

提供了关于 Tivoli Storage Manager（TSM，以前称为 ADSM）产品的信息，可用于管理数据库或表空间备份操作。

附录 G,『数据库恢复的用户出口』

讨论可如何将用户出口程序与数据库日志文件一起使用，并描述了某些样本用户出口程序。

附录 H,『供应商产品的备份与复原 API』

描述了 API 的函数和使用，这些 API 使 DB2 提供了与其它供应商软件的接口。

第 1 部分 数据恢复

第 1 章 制订一个良好的备份与恢复策略

本节讨论选择数据库和表空间恢复方法时要考虑的因素，包括备份和复原数据库或表空间，以及使用前滚恢复。

包括下列主题:

开发备份与恢复策略

数据库可能会因为硬件和/或软件故障而不可用。您可能会不时遇到存储问题、电源中断、应用程序故障以及各种需要采取不同恢复措施的故障情况。如果已准备了一个进行过彻底演习的恢复策略，它可保护您的数据免被丢失。在制订恢复策略时，您需要回答的一些问题是: 数据库是可恢复的吗？恢复数据库可能花费多长时间？在备份操作之间将花费多长时间？可以为备份副本和归档日志分配多少存储空间？表空间级的备份是否足够？或是否需要进行完整的数据库备份？

数据库恢复策略应确保在数据库恢复操作需要时，所有信息都可用。它应包括一个进行数据库备份的固定调度表，在分区数据库系统中则包括缩放系统时（添加

或删除数据库分区服务器或节点时）的备份。完整的策略还应包括恢复命令脚本、应用程序、用户定义的函数（UDF）、操作系统库中的存储过程代码以及装入副本的过程。

在以下几节中还讨论了不同的恢复方法，您将发现最适用于您的商业环境的是哪种恢复方法。

数据库*备份*的概念与其它任何数据备份的概念一样：即，复制一份数据，然后将它存储在另一媒体上，以防原始媒体发生故障或毁坏。最简单的备份情况涉及关闭数据库（以确保不发生更多的事务），然后简单地对其进行备份。以后如果数据库被损坏或毁坏，就可以重新构建它。

数据库的重新构建称为*恢复*。*版本恢复*指的是使用备份操作期间创建的映象来复原数据库的先前版本。*前滚恢复*是指复原了数据库或表空间备份映象后，重新应用记录在数据库日志文件中的事务。

*崩溃恢复*是指在完成并落实所有更改（这些更改是一个或多个工作单元（事务）的一部分）之前如果发生故障，会自动恢复数据库。这是通过回滚未完成的事务，并完成在发生崩溃时仍在内存中的已落实事务来实现的。

恢复日志文件和恢复历史文件是在创建数据库时自动创建的（第 5 页的图 1）。在需要恢复丢失或损坏的数据时，这些日志文件是很重要的。不能直接修改恢复日志文件或恢复历史文件；但是通过使用 PRUNE HISTORY 命令，可以从恢复历史文件中除去条目。还可以使用 *rec_his_retentn* 数据库配置参数来指定将保留恢复历史文件的天数。

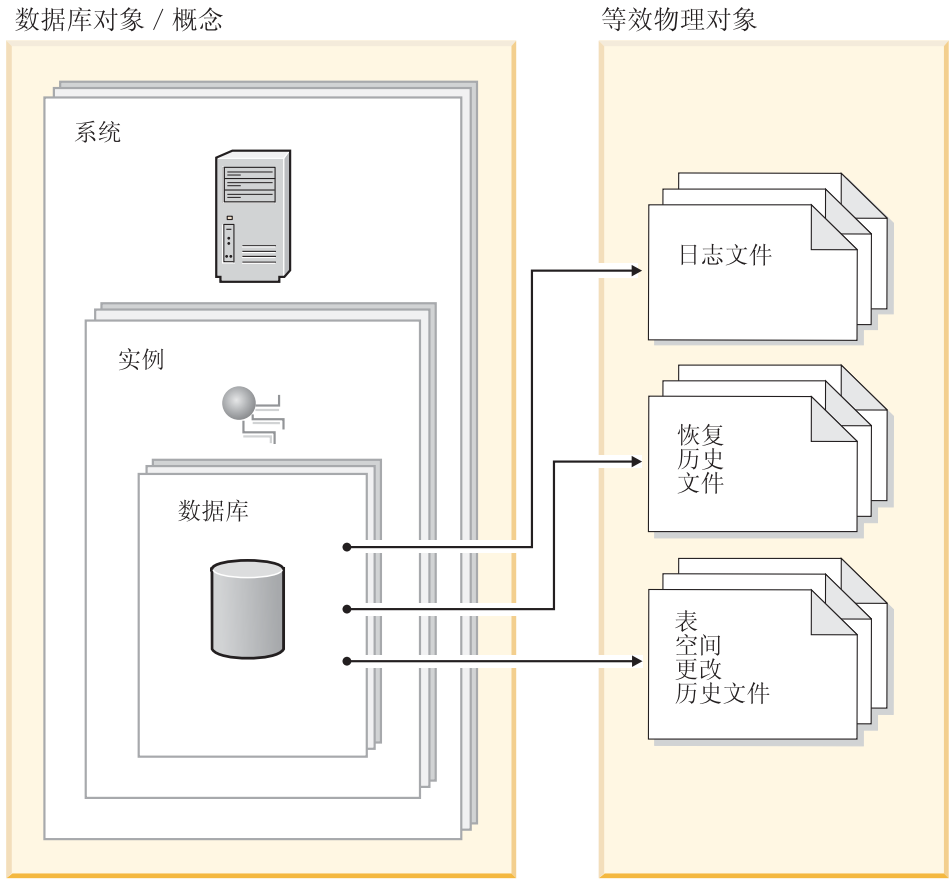


图 1. 恢复日志文件和恢复历史文件

每个数据库都包括*恢复日志*，它们用来从应用程序或系统错误中恢复。与数据库备份一起，它们用来将数据库的一致性恢复到出错时的时间点。

*恢复历史文件*包含当数据库的所有或部分必须恢复到给定时间点时，可用来确定恢复选项的备份信息的摘要。恢复历史文件用来跟踪其它操作中与恢复相关的事件，如备份与复原操作。此文件位于数据库目录中。

表空间更改历史文件（它也位于数据库目录中）包含可用来确定哪些日志文件对特定表空间的恢复是必需的信息。

那些很容易重新创建的数据可存储在不可恢复的数据库中。这些数据包括：用于只读应用程序的外部源中的数据以及不常进行更新的表；由于对它们进行的日志记录量较小，如果在复原操作之后还要进行复杂的日志文件文件管理工作和前滚操作，就不太合理了。*不可恢复数据库*的 *logretain* 和 *userexit* 数据库配置参数都

已被禁用。这表明将只保存崩溃恢复所必需的日志。这些日志称为*活动日志*，它们包含当前事务数据。使用*脱机*备份的版本恢复是解决不可恢复数据库的恢复问题的主要手段。（脱机备份表示当备份操作正在进行时，其它应用程序无法使用该数据库。）这样的数据库只能进行脱机复原。它被复原为生成备份映象时的状态且不支持前滚恢复。

那些不容易重新创建的数据应存储在可恢复的数据库中。这些数据包括在装入后它的源已被破坏的数据、手工输入到表中的数据以及在装入数据库后由应用程序或用户修改的数据。*可恢复数据库*将 *logretain* 数据库配置参数设置为“RECOVERY”或启用了 *userexit* 数据库配置参数，或两种方法同时使用。活动日志仍可用于崩溃恢复，但您还有*已归档日志*，它包含已落实的事务数据。这样的数据库只能进行脱机复原。它被复原为创建备份映象时的状态。但对于前滚恢复，可通过使用活动日志和已归档日志来将数据库*前滚*（即，越过创建备份映象的时间）至特定的时间点，或者滚动至活动日志末尾。

可恢复数据库备份操作可以脱机执行，也可以*联机*执行（联机表示在备份操作期间其它应用程序可与该数据库连接）。数据库复原和前滚操作必须始终以脱机方式执行。联机备份操作期间，前滚恢复确保捕捉了*所有*表更改，且在复原该备份时重新应用这些更改。

若有一个可恢复数据库，则可备份、复原并将个别表空间前滚，而不必对整个数据库操作。当联机备份表空间时它仍然可用，同时发生的更新记录在日志中。当对表空间执行联机复原或前滚操作时，在该操作完成之前，该表空间本身不可用，但不阻止用户访问其它表空间中的表。

相关概念:

- 第 10 页的『崩溃恢复』
- 第 21 页的『版本恢复』
- 第 22 页的『前滚恢复』
- 『Data Links server file backups』（*Post V8 GA*）
- 『Failure and recovery overview』（*DB2 Data Links Manager Administration Guide and Reference*）

相关参考:

- 『“恢复历史保留周期”配置参数 — rec_his_retentn』（*《管理指南: 性能》*）
- 『DB2 Data Links Manager system setup and backup recommendations』（*DB2 Data Links Manager Administration Guide and Reference*）

确定备份的频率

因为备份数据库需要时间和系统资源，所以恢复计划应该允许定期调度备份操作。计划可能包括完整数据库备份和增量备份操作的组合。

即使已将日志归档，也应定期执行完整的数据库备份（以便允许前滚恢复）。从表空间备份映象的集合来重新构建数据库要比从完整数据库备份映象来恢复数据库要花费更多时间。表空间备份映象在从独立的磁盘故障或应用程序错误进行恢复时很有用。

还应考虑不要覆盖备份映象和日志，应保存至少两个完整的数据库备份映象及其相关的日志，作为额外的预防措施。

若在恢复和前滚非常活跃的数据库时，应用归档日志所需的时间是主要关注的问题，应考虑进行更频繁的数据库备份所需的成本。进行更频繁的数据库备份可减少前滚时需要应用的归档日志数目。

数据库在*联机*或*脱机*时都可启动备份操作。如果它是联机的，在运行备份操作的同时，其它应用程序或进程可以与该数据库连接并读取和修改数据。如果备份操作是脱机运行的，则其它应用程序*不能*与数据库连接。

要缩短数据库处于不可用状态的时间，应考虑使用联机备份操作。仅当启用前滚恢复时，才支持联机备份操作。若启用前滚恢复且有一组完整的恢复日志，可以在需要时重建数据库。如果您的日志跨越了运行备份操作的时间段，则只能使用联机备份映象进行恢复。

脱机备份操作比联机备份操作更快，因为不存在数据文件的争用。

备份实用程序使您可备份所选的表空间。若使用 DMS 表空间，则可以将不同类型的数据存储在它们各自的表空间中，以减少备份操作所需的时间。可以将表数据保存在一个表空间中，将长整数字段和 LOB 数据保存在另一个表空间中，再将索引保存在一个表空间中。如果执行此操作并且磁盘发生故障，则可能只影响其中一个表空间。复原或前滚其中一个表空间将比复原包含所有数据的单个表空间花费更少时间。

还可以通过在不同时间对不同表空间进行备份来节省时间，只要对它们的更改不相同。因此，如果长字段或 LOB 数据不象其它数据那样作频繁地更改，则可以不那么频繁地备份这些表空间。如果长字段和 LOB 数据不是恢复所必需的，则可以考虑不备份包含该数据的表空间。若可从单独的源复制 LOB 数据，则当创建或改变一个表以包括 LOB 列时选择 NOT LOGGED 选项。

注: 考虑以下情况，如果将长字段数据、LOB 数据和索引放在单独的表空间中，而备份时不把它们放在一起：如果备份的表空间未包含所有表数据，则不能对该表空间执行时间点前滚恢复。包含一个表的任何类型数据的所有表空间都必须同时前滚至同一个时间点。

若重组一个表，应该在该操作完成后备份受影响的表空间。若不得不复原这些表空间，将不必通过数据重组来前滚。

恢复一个数据库所需的时间由两部分组成：复原备份所需的时间；以及，若允许数据库进行前滚恢复，在前滚操作期间应用日志所需的时间。当确定恢复计划时，应考虑这些恢复成本和它们对商业运作的影响。测试整体恢复计划有助于确定恢复数据库所需的时间是否适合给定的业务需求。在每次测试之后，可能要增加建立备份的频率。若前滚恢复是策略的一部分，这将会减少备份之间归档的日志数，因此也减少了复原操作之后前滚数据库所需的时间。

相关概念:

- 第 25 页的『增量备份与恢复』

相关参考:

- 第 297 页的附录 G, 『数据库恢复的用户出口』
- 第 34 页的『数据库记录的配置参数』

存储器注意事项

当确定要使用哪种恢复方法时，考虑必需的存储空间。

版本恢复方法需要空间来容纳数据库的副本和复原的数据库。前滚恢复方法需要空间来容纳数据库或表空间的副本、复原的数据库和归档的数据库日志。

若一个表包含长整数字段或大对象（LOB）列，应考虑将此数据置于单独的表空间中。这将会影响存储空间注意事项，并影响恢复计划。如果使用单独的表空间来存储长整数字段和 LOB 数据，并知道备份长整数字段和 LOB 数据所需的时间，那么可以决定使用这样一个恢复计划，它以较低的频率保存此表空间的备份。也可选择在创建或改变一个表以包括 LOB 列时，不记录对那些列的更改。这将减少必需的日志空间和对应的日志归档空间的大小。

要防止媒体故障损坏数据库且使您无法重建它，应该在不同的设备上保存数据库备份、数据库日志和数据库本身。鉴于此原因，极力建议您使用 *newlogpath* 配置参数，这样一旦创建了该数据库，就将数据库日志置于单独的设备中。

数据库日志可能会占用大量的存储器。若计划使用前滚恢复方法，则必须确定管理归档日志的方法。以下是您的选择:

- 使用用户出口程序来将这些日志复制到您环境中的另一个存储设备中。
- 当某些日志不再出现在活动的日志集中后，以手工方式将这些日志复制到存储设备或非数据库日志路径目录的一个目录中。

将相关的数据保存在一起

作为数据库设计的一部分，您将了解表之间存在的关系。这些关系可以在应用程序级表示（当事务更新多个表时），也可以在数据库级表示（表之间存在引用完整性，或者一个表上的触发器影响另一个表）。当制定恢复计划时，应该考虑这些关系。您将希望把相关的数据集备份在一起。可以在表空间级或数据库级建立这样的集合。通过将相关的数据集保存在一起，可以恢复至所有数据都一致的时间点。若您希望能够对表空间执行时间点前滚恢复，这一点尤其重要。

使用不同的操作系统

在具有多个操作系统的一个环境中工作时，必须注意: 在大多数情况下，备份和恢复计划是不能集成的。也就是说，通常不能在一个操作系统上备份数据库，然后在另一操作系统上复原该数据库。在这种情况中，应使每个操作系统的恢复计划分开并互不相关。

但在，支持具有相似结构的操作系统（例如，AIX® 和 Sun Solaris）与 32 位和 64 位操作系统之间的交叉平台备份与复原操作。在系统间传送备份映象时，必须采用二进制方式。目标系统必须与源系统具有 DB2® 的相同（或更新版本）版本。对下级系统的复原操作不受支持。

如果必须将表从一个操作系统移动到另一操作系统，但在您的环境中又不支持交叉平台的备份与复原操作，可以使用 **db2move** 命令，或导出实用程序（后跟导入或装入实用程序）。

相关参考:

- 『db2move - Database Movement Tool Command』（*Command Reference*）
- 『EXPORT Command』（*Command Reference*）
- 『IMPORT Command』（*Command Reference*）
- 『LOAD Command』（*Command Reference*）

崩溃恢复

对数据库执行的事务（也称工作单元）可能被意外中断。若在作为工作单元一部分的所有更改完成和落实之前发生故障，则该数据库就会处于不一致和不可用的状态。*崩溃恢复*是将数据库移动回一致并可用状态的进程。为此，回滚未完成的事务，并完成当发生崩溃时仍在内存中的已落实事务（图 2）。当数据库处于一致和可用状态时，它处于一种称为“一致点”的状态。

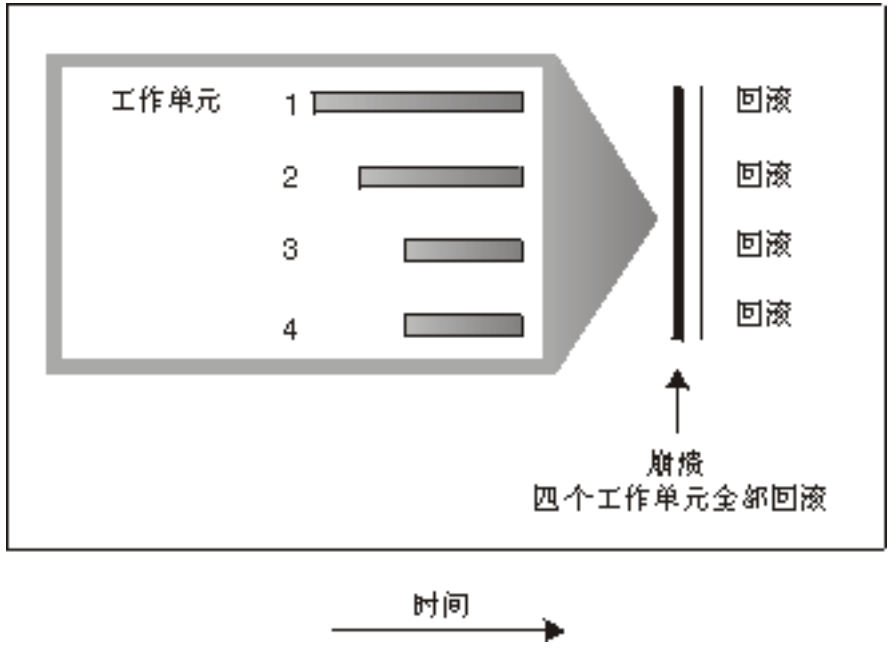


图 2. 回滚工作单元（崩溃恢复）

*事务处理失败*是由于出现了严重错误或导致数据库或数据库管理器异常结束的情况。部分完成的工作单元或发生故障时未清仓至磁盘中的 UOW 使数据库处于不一致状态。在事务处理故障之后必须恢复数据库。导致事务处理故障的情况有:

- 机器上的掉电故障，它会导致使用该机器的数据库管理器和数据库分区崩溃
- 导致 DB2® 崩溃的严重操作系统错误
- 硬件故障，例如内存毁坏、磁盘、CPU 或网络故障。

如果您希望不完整工作单元的回滚是由数据库管理器自动完成的，则应将 *autorestart* 数据库配置参数设置为 ON，以启用该自动重新启动参数。（这是缺省值。）如果不想要重新启动，则将 *autorestart* 数据库配置参数设置为 OFF。这样，将需要在数据库故障发生时发出 RESTART DATABASE 命令。如果数据库 I/O 在

发生崩溃之前已处于暂挂状态，则必须指定 RESTART DATABASE 命令的 WRITE RESUME 选项才能使崩溃恢复继续进行。管理通知日志记录数据库重新启动操作开始的时间。

如果对允许正向恢复的数据库应用崩溃恢复（即，将 *logretain* 配置参数设置为 RECOVERY，或启用 *userexit* 配置参数），且在崩溃恢复期间因个别表空间而导致发生错误，则将会让该表空间脱机，直到修复后才能访问它。崩溃恢复继续进行。在崩溃恢复完成时，该数据库中的其它表空间将是可存取的，并且可以与该数据库建立连接。但是，如果脱机的表空间包含系统目录，则必须先修复它才允许进行所有连接。

相关参考:

- 『“自动重新启动启用”配置参数 — autorestart』（《*管理指南: 性能*》）

崩溃恢复 — 详细信息

恢复已损坏的表空间

损坏的表空间带有一个或多个不能访问的容器。这通常是由媒体问题引起的: 这些问题或者是永久性的（例如，坏的磁盘），或者是临时性的（例如，脱机磁盘或未安装的文件系统）。

若损坏的表空间是系统目录表空间，则不能重新启动数据库。若不能修复容器问题但要使原始数据不受影响，可用的选项包括:

- 要复原数据库
- 要复原目录表空间。（因为数据库必须前滚，所以表空间复原只对可恢复数据库有效。）

如果损坏的表空间*不是*系统目录表空间，则 DB2® 会尝试使尽可能多的数据库部分可用。

如果损坏的表空间只是临时的表空间，应在可进行与数据库的连接后，尽快创建新的临时表空间。创建后，可使用这个新的临时表空间，并且需要临时表空间的正常数据库操作也可以继续执行。若希望的话，可删除脱机的临时表空间。使用系统临时表空间的表重组有特殊的注意事项:

- 若数据库或数据库管理器配置参数 *indexrec* 设置为 RESTART，在数据库活动期间必须重建所有无效的索引；这包括构建阶段期间发生故障的重组中的索引。
- 若在损坏的临时表空间中有未完成的重组请求，可能必须将 *indexrec* 配置参数设置为 ACCESS 以避免重新启动失败。

恢复可恢复数据库中的表空间

当需要崩溃恢复时，损坏的表空间将会脱机，并且将不可存取。它将被置于前滚暂挂状态。如果没有其它问题的话，重新启动操作将成功，并且一旦您执行下列操作，就可以再次使用损坏的表空间:

- 修复损坏的容器而不丢失原始数据，然后完成表空间前滚操作直到日志的结尾。（前滚操作将首先尝试使该表空间由脱机状态转移至正常状态。）
- 修复损坏的容器（丢失或未丢失原始数据）之后，执行表空间复原操作，然后执行前滚操作直到日志的结尾或某个时间点。

恢复不可恢复的数据库中的表空间

因为崩溃恢复是必需的，且日志不会无限期保存，因此仅当用户希望删除损坏的表空间时重新启动操作才会成功。（恢复成功完成意味着将损坏的表空间恢复到一致状态所必需的运行记录不再存在；因此，对这类表空间可以采取的唯一有效操作是删除它们。）

可通过调用未限定的重新启动数据库操作来做到这一点。若没有损坏的表空间，它将成功。如果它失败（SQL0290N），则可以查看管理通知日志文件以获得当前已损坏的表空间的完整列表。

- 如果您愿意在重新启动数据库操作完成时删除所有这些表空间，则可启动另一重新启动数据库操作，使用 DROP PENDING TABLESPACES 选项列示所有已损坏的表空间。若损坏的表空间包括在 DROP PENDING TABLESPACES 列表中，则表空间进入删除暂挂状态，在恢复之后，唯一的选项是删除该表空间。重新启动操作继续，而不恢复此表空间。若损坏的表空间未包括在 DROP PENDING TABLESPACES 列表中，则重新启动数据库操作失败，并返回 SQL0290N。
- 若您不想删除这些表空间并因此丢失其中的数据，则有以下选项:
  - 等待并修复损坏的容器（没有丢失原始数据），然后再次尝试重新启动数据库操作
  - 执行数据库复原操作。

注: 将表空间名放入 DROP PENDING TABLESPACES 列表中并不表示该表空间将处于删除暂挂状态。仅当在重新启动操作期间发现表空间损坏时才会是这样。重新启动操作成功后，应发出 DROP TABLESPACE 语句来删除每个处于删除暂挂状态的表空间（调用 LIST TABLESPACES 命令来了解哪些表空间处于此状态）。这样，可收回该空间，或者可重新创建那些表空间。

降低媒体故障的影响

要降低媒体故障的可能性，并简化从此类型的故障的恢复:

- 镜像或复制保存重要数据库的数据和日志的磁盘。
- 使用“冗余独立磁盘阵列”(RAID) 配置，例如 RAID 级别 5。
- 在分区数据库环境中，要设置一个严格的过程，来处理该目录节点上的数据和日志。因为此节点是维护数据库的关键:
  - 确保它驻留在可靠的磁盘上
  - 复制它
  - 频繁地进行备份
  - 不要在此节点上存放用户数据。

防止磁盘故障

若您担心由于磁盘故障而使数据或日志损坏的可能性，则请考虑使用某种形式磁盘故障容错。通常，这是通过使用*磁盘阵列*（一组磁盘）来完成的。

磁盘阵列有时只是指 RAID（冗余独立磁盘阵列）。磁盘阵列也可以通过处于操作系统级或应用程序级的软件来提供。硬件磁盘阵列和软件磁盘阵列的不同点在于处理输入／输出（I/O）请求的 CPU 处理的方式。对于硬件磁盘阵列，I/O 活动由磁盘控制器管理，对于软件磁盘阵列，由操作系统或应用程序完成。

硬件磁盘阵列: 在硬件磁盘阵列中，一个磁盘控制器可使用和管理多个磁盘，且使用它自己的 CPU 来完成。管理构成此阵列的磁盘所需的所有逻辑都包含在磁盘控制器中；因此，此实现与操作系统无关。

有几种类型的 RAID 体系结构，在功能和性能上有所不同，但只有 RAID 级别 1 和级别 5 是现在常用的。

RAID 级别 1 也称为磁盘镜像和双工技术。*磁盘镜像技术*使用单个磁盘控制器将数据（一个完整的文件）从一个磁盘复制到另一个磁盘。*磁盘双工技术*类似于磁盘镜像技术，但磁盘是与另一个磁盘控制器连接的（与两个 SCSI 适配器相似）。因此可较好地保护数据：任何一个磁盘发生故障时，仍可从另一个磁盘访问数据。使用磁盘双工技术时虽然磁盘控制器可能会发生故障，但不会影响可提供的数据保护。它的性能很好，但实现它需要的磁盘数是通常磁盘数的两倍。

RAID 级别 5 涉及到跨所有磁盘，按扇区进行的数据和奇偶性校验分割。奇偶性校验与数据交错，而不是存储在专用的驱动器上。数据保护性能很好：如果有任何磁盘发生故障，仍然可以使用其它磁盘上的信息以及已分割的奇偶性校验信息访问数据。读性能良好，但写性能较差。RAID 级别 5 配置需要最少三个相同的磁

盘。开销所需的磁盘空间量随阵列中磁盘的数量而变化。如果 RAID 级别 5 配置了 5 个磁盘，空间开销将是百分之二十。

使用 RAID（但不是 RAID 级别 0）磁盘阵列时，有故障的磁盘不会影响您访问阵列上的数据。当在该阵列中使用可热插入的或可热交换的磁盘时，可以在该阵列正在使用时将替换磁盘与故障磁盘交换。使用 RAID 级别 5 时，如果有两个磁盘同时发生故障，所有数据都会丢失（但同时发生磁盘故障的可能性非常小）。

您可能会考虑对日志使用 RAID 级别 1 硬件磁盘阵列或软件磁盘阵列，因为这可以使故障点具有恢复能力并提供良好的写性能（这对日志来说很重要）。如果（由于不能将时间花在磁盘故障后的数据恢复上）可靠性很关键，而写性能不是，则可以考虑使用 RAID 级别 5 硬件磁盘阵列。另外，如果写性能很重要，而不太关心所需的附加磁盘空间量，则可以考虑对您的数据以及日志使用 RAID 级别 1 硬件磁盘阵列。

有关可用的 RAID 级别的详细信息，访问以下 Web 站点：http://www.acnc.com/04_01_00.html

软件磁盘阵列: 软件磁盘阵列在许多方面与硬件磁盘阵列相同，但是磁盘流量是由操作系统或在服务器上运行的应用程序来管理的。软件阵列象其它程序一样，必须争用 CPU 和系统资源。对于 CPU 受约束的系统，它不是一个好的选项，同时要记住磁盘阵列的总体性能取决于服务器的 CPU 负载和能力。

典型的软件磁盘阵列提供磁盘镜像。虽然冗余磁盘是必需的，但是由于不需要高成本的磁盘控制器，所以软件磁盘阵列的实现相对便宜。

注意:
如果操作系统引导驱动器在磁盘阵列中，则当该驱动器发生故障时，系统将无法启动。若在磁盘阵列运行之前该驱动器发生故障，则磁盘阵列就不允许访问该驱动器。引导驱动器应与磁盘阵列分开。

降低事务故障的影响

要降低事务故障的影响，应尽量确保:

- 不间断电源
- 有足够的磁盘空间用于存储数据库日志
- 在分区数据库环境中的数据库分区服务器之间有可靠的通信链路
- 分区数据库环境中的系统时钟的同步。

相关概念:

- 第 116 页的『使分区数据库系统中的时钟同步』

从分区数据库环境中的事务处理故障中恢复

如果事务处理失败发生在分区数据库环境中，通常需要对发生了故障的数据库分区服务器和参予了该事务的任何其它数据库分区服务器都进行数据库恢复:

- 对发生了故障的数据库分区服务器的崩溃恢复发生在更正了先前的错误情况后。
- 对其它（仍活动的）数据库分区服务器的*数据库分区故障恢复*紧接在检测到故障后发生。

在分区数据库环境中，提交应用程序的数据库分区服务器是协调程序节点，而为该应用程序工作的第一个代理进程是协调代理进程。协调代理进程负责将工作分发至其它数据库分区服务器上，并跟踪哪些服务器参与了该事务。当应用程序对一个事务发出 COMMIT 语句时，该协调代理进程使用两阶段落实协议来落实该事务。在第一阶段期间，协调程序节点将 PREPARE 请求分发至参与该事务的所有其它的数据库分区服务器。然后，这些服务器用下列其中一项应答:

READ-ONLY	在此服务器中未发生任何数据更改
YES	在此服务器中发生了数据更改
NO	由于错误，服务器未准备落实

若其中一个服务器应答 NO，则回滚该事务。否则，协调程序节点开始第二个阶段。

在第二阶段期间，协调程序节点写入一条 COMMIT 日志记录，然后将 COMMIT 请求分发至应答了 YES 的所有服务器。在所有其它数据库分区服务器都已落实后，它们会将 COMMIT 的确认发送至协调程序节点。当协调代理进程从所有参与的服务器接收到所有 COMMIT 确认时，该事务完成。在此时间点，协调代理进程会写入一条 FORGET 日志记录。

活动数据库分区服务器上的事务处理故障恢复

若任何数据库分区服务器检测到另一个服务器当机，则与该发生故障的数据库分区服务器相关的所有工作都会停止:

- 若仍为活动的数据库分区服务器是一个应用程序的协调程序节点，且该应用程序是在发生故障的数据库分区服务器上运行（尚未准备 COMMIT），则会中断该协调代理进程，以便执行故障恢复。如果该协调代理进程处于 COMMIT 处理的第二个阶段，会将 SQL0279N 返回给应用程序，应用程序随之会丢失它的数据库连接。否则，协调代理进程将一个 ROLLBACK 请求分发至所有其它参与该事务的服务器，并将 SQL1229N 返回至该应用程序。
- 若发生故障的数据库分区服务器是该应用程序的协调程序节点，仍在活动服务器上为该应用程序工作的代理进程会被中断，以便执行故障恢复。除非当前事务已准备就绪且正在等待事务结果，否则在每个服务器上以本地方式回滚当前

事务。在这种情况下，该事务在活动的数据库分区服务器上处于未确定状态，而协调程序节点未意识到这个情况（因为它不可用）。

- 若该应用程序与发生故障的数据库分区服务器连接（在它发生故障之前），但是本地数据库分区服务器和发生故障的数据库分区服务器都不是协调程序节点，则会中断为此应用程序工作的代理进程。协调程序节点将向其它数据库分区服务器发送 ROLLBACK 或断开连接消息。若协调程序节点返回 SQL0279，则事务将只在仍然活动的数据库分区服务器上是未确定的。

试图向该发生故障的服务器发送请求的任何进程（如，代理进程或死锁检测器）都会得到通知：它不能发送该请求。

有故障的数据库分区服务器上的事务处理失败恢复

如果事务失败导致数据库管理器异常结束，则可以发出带有 RESTART 选项的 **db2start** 命令以在重新启动数据库分区后立即重新启动数据库管理器。如果不能重新启动数据库分区，则可以发出 **db2start** 以在另一分区上重新启动数据库管理器。

如果数据库管理器异常结束，则服务器上的数据库分区可能会处于不一致状态。要使它们可用，可以在数据库分区服务器上触发崩溃恢复：

- 通过 RESTART DATABASE 命令显式触发
- 在 *autorestart* 数据库配置参数已设置为 ON 后，通过 CONNECT 请求隐式触发

崩溃恢复将重新应用活动日志文件中的日志记录，以确保所有已完成的事务的结果都在数据库中。重新应用了这些更改后，除不确定事务外的所有未落实的事务都将本地回滚。分区数据库环境中有两种类型的不确定事务：

- 在不是协调程序节点的数据库分区服务器上，已就绪但未落实的事务就是未确定的。
- 在协调程序节点上，已落实但还未被记录为完成（即，还未写入 FORGET 记录）的事务是未确定的。当协调代理进程未从为该应用程序工作的所有服务器接收到全部 COMMIT 确认时，会发生此情况。

崩溃恢复试图通过下列其中一项操作解决所有未确定的事务。要执行的操作取决于数据库分区服务器是否为应用程序的协调程序节点：

- 若重新启动的服务器不是该应用程序的协调程序节点，它会将一个查询消息发送至该协调代理进程，以发现该事务的结果。
- 若重新启动的服务器是该应用程序的协调程序节点，它会将一个消息发送至协调代理进程仍在等待它们的 COMMIT 确认的所有其它代理进程（从属代理进程）。

有可能崩溃恢复不能解决所有不确定事务（例如，某些数据库分区服务器可能会不可用）。在这种情况中，会返回 SQL 警告消息 SQL1061W。由于不确定事务占用了资源（例如锁和活动日志空间），有可能会导致不能对数据库进行任何更改，因为不确定事务占用了活动日志空间。因此，应确定在崩溃恢复之后是否还有不确定事务，并尽快恢复解决这些不确定事务所需的所有数据库分区服务器。

若解决不确定事务所需的一个或多个服务器不能及时恢复，且需要访问其它服务器上的数据库分区，可以通过作出试探性决策来手工解决这些不确定事务。可以使用 LIST INDOUBT TRANSACTIONS 命令来查询、落实和回滚服务器上的不确定事务。

注: LIST INDOUBT TRANSACTIONS 命令还可用于分布式事务环境中。要区分这两种类型的不确定事务，LIST INDOUBT TRANSACTIONS 命令所返回的输出中的 *originator* 字段显示下列其中一项:

- DB2 通用数据库扩充企业版，它指示该事务始发于分区数据库环境中。
- XA，它指示该事物始发于分布式环境中。

标识有故障的数据库分区服务器

当一个数据库分区服务器发生故障时，应用程序通常会接收到下列其中一个 SQLCODE。检测哪个数据库管理器发生故障的方法取决于接收到的 SQLCODE:

SQL0279N
当在 COMMIT 处理期间终止的事务中涉及了数据库分区服务器时，会接收到此 SQLCODE。

SQL1224N
当发生故障的数据库分区服务器是该事务的协调程序节点时，会接收到此 SQLCODE。

SQL1229N
当发生故障的数据库分区服务器不是该事务的协调程序节点时，会接收到此 SQLCODE。

确定哪个发生了故障的数据库分区服务器是一个两阶段进程。与 SQLCODE SQL1229N 相关的 SQLCA 在 *sqlerrd* 字段的第六个数组位置包含检测到错误的服务器的节点号。（为服务器写入的节点号与 `db2nodes.cfg` 文件中的节点号对应。）在检测到错误的数据库分区服务器上，会将指示故障服务器的节点号的消息写至管理通知日志。

注: 若正在一个处理器上使用多个逻辑节点，则一个逻辑节点发生故障可能导致同一个处理器上的其它逻辑节点发生故障。

相关概念:

- 『两阶段落实』（*《管理指南: 计划》*）
- 『两阶段落实期间的错误恢复』（*《管理指南: 计划》*）

相关任务:

- 『手工解析不确定事务』（*《管理指南: 计划》*）

相关参考:

- 『db2start - Start DB2 Command』（*Command Reference*）
- 『LIST INDOUBT TRANSACTIONS Command』（*Command Reference*）

从数据库分区服务器的故障恢复

过程:

要从数据库分区服务器的故障中恢复:

1. 校正导致该故障的问题。
2. 通过从任何数据库分区服务器发出 **db2start** 命令，重新启动数据库管理器。
3. 通过在有故障的一个或多个数据库分区服务器上发出 RESTART DATABASE 命令，重新启动数据库。

相关概念:

- 第 15 页的『从分区数据库环境中的事务处理故障中恢复』

相关参考:

- 『db2start - Start DB2 Command』（*Command Reference*）
- 『RESTART DATABASE Command』（*Command Reference*）

在 DB2 Connect 配置了 DB2 同步点管理器的情况下恢复主机上的不确定事务

若在一个事务期间您的应用程序访问了主机或 AS/400 数据库服务器，则恢复不确定事务的方法会有所不同。

要访问主机或 AS/400 数据库服务器，须使用 DB2 Connect。若 DB2 Connect 配置了 DB2 同步点管理器，则恢复的步骤就会不同。

过程:

恢复主机或 AS/400 服务器上的不确定事务通常由“事务管理程序”(TM) 和 DB2 同步点管理器 (SPM) 自动执行。主机或 AS/400 服务器上的不确定事务不占用本地 DB2 位置的任何资源，但是只要该事务在该位置处是未确定的，它就会占用主机或 AS/400 服务器上的资源。若主机或 AS/400 服务器的管理员确定必须作出试探性决定，则管理员可能会与本地 DB2 数据库管理员联系（例如，通过电话）来

确定是落实还是回滚主机或 AS/400 服务器上的这个事务。若发生这种情况，可以使用 LIST DRDA INDOUBT TRANSACTIONS 命令来确定 DB2 Connect 实例中的这个事务的状态。可以使用下列步骤作为涉及到 SNA 通信环境的大多数情况的一个参考。

1. 与 SPM 连接，如下所示:

```
db2 => connect to db2spm

 数据库连接信息

 数据库产品        = SPM0500
 SQL 授权标识      = CRUS
 本地数据库别名    = DB2SPM
```

2. 发出 LIST DRDA INDOUBT TRANSACTIONS 命令，以显示 SPM 识别的不确定事务。以下示例显示一个 SPM 识别的不确定事务。db_name 是主机或 AS/400 服务器的本地别名。partner_lu 是主机或 AS/400 服务器的全限定 luname。它为主机或 AS/400 服务器提供了最佳标识，它应由主机或 AS/400 服务器的调用程序提供。luwid 提供事务的唯一标识符，且在所有主机和 AS/400 服务器上可用。若显示未确定的事务，而且若 uow_status 字段的值是 C（落实）或 R（回滚），可以使用该值来确定事务的输出。若您发出 LIST DRDA INDOUBT TRANSACTIONS 命令且带有 WITH PROMPTING 参数，则您可以用交互式方式落实、回滚或忘记该事务。

```
db2 => list drda indoubt transactions
 DRDA Indoubt Transactions:
 1.db_name: DBAS3    db_alias: DBAS3    role: AR
   uow_status: C  partner_status: I  partner_lu: USIBMSY.SY12DQA
 corr_tok: USIBMST.STB3327L
 luwid: USIBMST.STB3327.305DFDA5DC00.0001
 xid: 53514C2000000017 00000000544D4442 0000000000305DFD A63055E962000000
      00035F
```

3. 若未显示 partner_lu 和 luwid 的不确定事务，或者若 LIST DRDA INDOUBT TRANSACTIONS 命令返回下列内容:

```
db2 => list drda indoubt transactions
SQL1251W  No data returned for heuristic query.
```

则该事务已回滚。

另一个情况不大可能但是也许已经发生。若显示带有 partner_lu 的适当 luwid 的不确定事务，但 uow_status 是 “I”，则 SPM 不能确定是落实还是回滚该事务。在这种情况下，您应该使用 WITH PROMPTING 参数在 DB2 Connect 工作站上来落实或回滚该事务。然后允许 DB2 Connect 根据试探性决定与主机或 AS/400 服务器再同步。

相关任务:

- 第 20 页的『当 DB2 Connect 不使用 DB2 同步点管理器时恢复主机上的不确定事务』

相关参考:

- 『db2start - Start DB2 Command』（*Command Reference*）
- 『LIST INDOUBT TRANSACTIONS Command』（*Command Reference*）
- 『RESTART DATABASE Command』（*Command Reference*）

当 DB2 Connect 不使用 DB2 同步点管理器时恢复主机上的不确定事务

若在一个事务期间您的应用程序访问了主机或 AS/400 数据库服务器，则恢复不确定事务的方法会有所不同。

要访问主机或 AS/400 数据库服务器，须使用 DB2 Connect。若 DB2 Connect 配置了 DB2 同步点管理器，则恢复的步骤就会不同。

过程:

当在“DB2 Connect 个人版”或“DB2 Connect 企业服务器版”中的多站点更新中使用 TCP/IP 连接来更新 DB2 OS/390 版且未使用DB2 同步点管理器时，使用本节中的信息。此情况中的不确定事务的恢复不同于使用 DB2 同步点管理器的不确定事务的恢复。当在此环境中出现不确定事务时，会在客户机、数据库服务器和／或“事务管理程序”(TM) 数据库上生成一个警告条目，这取决于谁检测到该问题。该警告条目被置于 `db2alert.log` 文件中。

一旦 TM 和参与数据库及其连接再次全部可用，就会自动执行任何不确定事务的再同步。您应该允许在数据库服务器中自动执行再同步，而不是试探性地强制决定。但是，若您必须这样做，可参考下列步骤。

注: 因为未涉及到 DB2 同步点管理器，所以您不能使用 LIST DRDA INDOUBT TRANSACTIONS 命令。

1. 在 OS/390 主机上，发出命令 DISPLAY THREAD TYPE(INDOUBT)。

   从此列表中，标识您要试探性完成的事务。关于 DISPLAY 命令的详细信息，请参阅 *DB2 for OS/390 Command Reference*。显示的 LUWID 可以与“事务管理程序数据库”中的同一个 luwid 匹配。
2. 根据您要执行的操作，发出 RECOVER THREAD(<LUWID>) ACTION(ABORT|COMMIT) 命令。

   关于 RECOVER 命令的详细信息，请参阅 *DB2 for OS/390 Command Reference*。

相关任务:

- 第 18 页的『在 DB2 Connect 配置了 DB2 同步点管理器的情况下恢复主机上的不确定事务』

相关参考:

- 『LIST INDOUBT TRANSACTIONS Command』（*Command Reference*）

灾难恢复

术语*灾难恢复*用于描述在火灾、地震、恶意破坏或其它大灾害事件中复原数据库所需要执行的活动。灾难恢复的计划可以包括下列其中一项或多项:

- 在紧急情况中要使用的场地
- 用于恢复数据库的另一台机器
- 数据库备份和归档日志的非现场存储器。

若灾难恢复计划是在另一台机器上恢复整个数据库，则至少需要一个完整的数据库备份和该数据库的所有归档日志。可选择通过将归档日志应用于数据库，以便备用数据库保持最新。或者，可选择将数据库备份和日志归档保存在备用场地，并只在发生灾难之后执行复原和前滚操作。（在这种情况中，最新的数据库备份无疑是所希望的。）但是，当发生灾难时，将所有事务恢复至灾难发生时的状态一般是不可能的。

灾难恢复的表空间备份的有用性取决于发生故障的范围。通常，灾难恢复需要复原整个数据库，因此，应在备用场所保存一份完整的数据库备份。即使有每个表空间的单独备份映象，也不能使用它们来恢复该数据库。若该灾难是磁盘损坏，则可以使用该磁盘上每个表空间的表空间备份来恢复。若由于磁盘故障（或任何其它原因）而无法访问一个容器，可以将该容器复原至不同的位置。

表空间备份和完整的数据库备份都可以在任何灾难恢复计划中起到一定的作用。可用于备份、复原和前滚数据的 DB2® 设施为灾难恢复计划提供了基础。应确保已测试了恢复过程，它可以保护您的业务。

相关概念:

- 第 82 页的『在复原操作期间重新定义表空间容器（重定向复原）』

版本恢复

*版本恢复*指的是使用备份操作期间创建的映象来复原数据库的先前版本。对不可恢复数据库（即，该数据库没有归档日志）使用此方法。还可对 RESTORE DATABASE 命令使用 WITHOUT ROLLING FORWARD 选项，对可恢复数据库使用此方法。数据库复原操作将使用先前创建的备份映象来重建整个数据库。数

据库备份使您可以将数据库复原到与进行备份时完全相同的状态。但是，从备份时间到故障时间之间的每个工作单元都将丢失（参见图 3）。

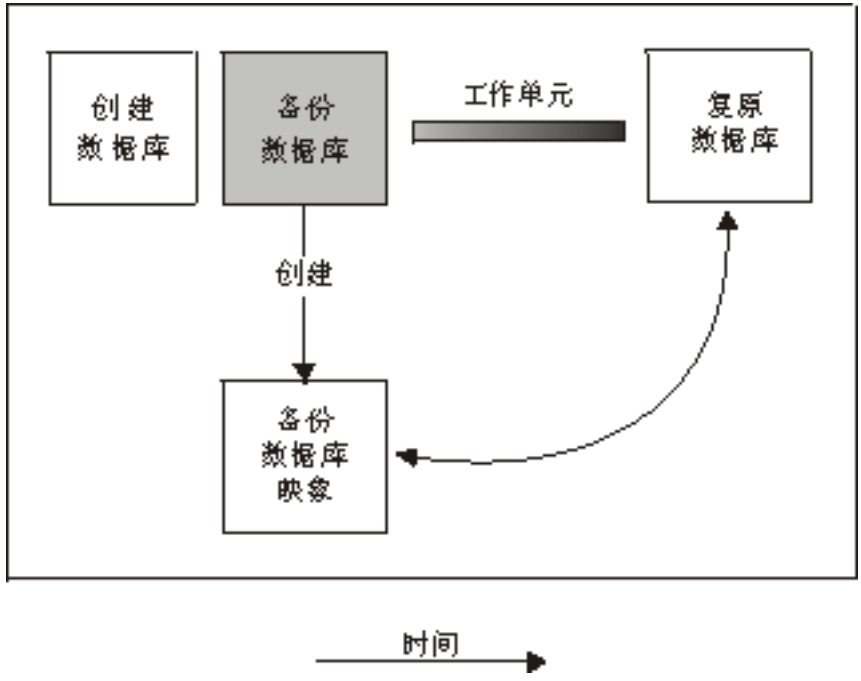


图 3. 版本恢复. 从最新的备份映象复原数据库，但在备份时间和故障时间之间处理的所有工作单元都将丢失。

使用版本恢复方法，必须定期调度和执行完整数据库备份。

在一个分区数据库环境中，数据库位于许多数据库分区服务器（或节点）上。必须复原所有分区，而且用于复原数据库操作的所有备份映象必须是同时建立的。（将备份每个数据库分区并单独复原。）同时建立的每个数据库分区的备份称为*版本备份*。

前滚恢复

要使用*前滚恢复*方法，必须已建立了数据库的备份，并且已将日志归档（通过启用 *logretain* 和／或 *userexit* 数据库配置参数）。复原数据库并指定 WITHOUT ROLLING FORWARD 选项等效于使用版本恢复方法。此数据库被复原到创建脱机备份映象时的状态。若复原数据库，但*没有*对复原数据库操作指定 WITHOUT ROLLING FORWARD 选项，则该数据库在复原操作结束时将处于前滚暂挂状态。这允许进行前滚恢复。

注: 如果数据库备份已联机，则不能使用 WITHOUT ROLLING FORWARD 选项。

要考虑的两种恢复类型是:

- *数据库前滚恢复*。在此类型的前滚恢复中，记录在数据库日志中的事务应用到以下数据库复原操作中（参见图 4）。该数据库日志记录了对该数据库所作的所有更改。这种方法将数据库恢复到在某特定时间点的状态，或恢复到故障前的状态（即，恢复到活动日志的末尾）。

  在一个分区数据库环境中，数据库位于许多数据库分区上。若执行时间点前滚恢复，则必须前滚所有数据库分区，以确保所有分区都在同一级别。若需要复原单个数据库分区，则可以执行前滚恢复至日志末尾，以将它复原至与数据库中的其它分区相同的级别。如果前滚一个数据库分区，则只可使用对日志末尾的恢复。时间点恢复适用于*所有*数据库分区。

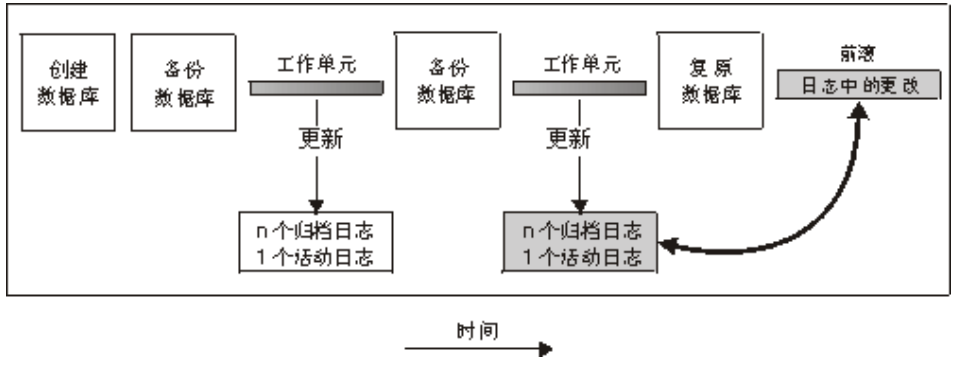


图 4. 数据库前滚恢复. 在运行时间较长的事务中，可以有多个活动的日志。

- *表空间前滚恢复*。如果启用了某个数据库对它进行前滚恢复，也可对它进行备份、复原并前滚表空间（参见第 24 页的图 5）。要执行表空间复原和前滚操作，需要整个数据库（即所有表空间）或一个或多个个别表空间的备份映象。还需要影响要恢复的表空间的运行记录。可在日志中前滚至以下两点之一:
  - 日志末尾；或
  - 一个特定时间点（称为*时间点*恢复）。

可在下列两种情况下使用表空间前滚恢复:

- 在一个表空间复原操作后，该表空间始终处于前滚暂挂状态，且必须将它前滚。调用 ROLLFORWARD DATABASE 命令将日志应用于表空间以使其前滚至某个时间点或日志末尾。
- 若一个或多个表空间在崩溃恢复后处于*前滚暂挂*状态，首先应校正该表空间的问题。某些情况下，校正表空间的问题不涉及复原数据库操作。例如，掉电可能导致表空间处于前滚暂挂状态。在这种情况下，复原数据库操作并不是必需

的。一旦校正了该表空间的问题，可使用 ROLLFORWARD DATABASE 命令将日志应用于表空间，使其恢复到日志末尾。若在崩溃恢复之前校正此问题，崩溃恢复将足以使数据库恢复到一致可用的状态。

注：若出错的表空间包含系统目录表，将不能启动数据库。必须复原 SYSCATSPACE 表空间，然后执行前滚恢复直至日志末尾。

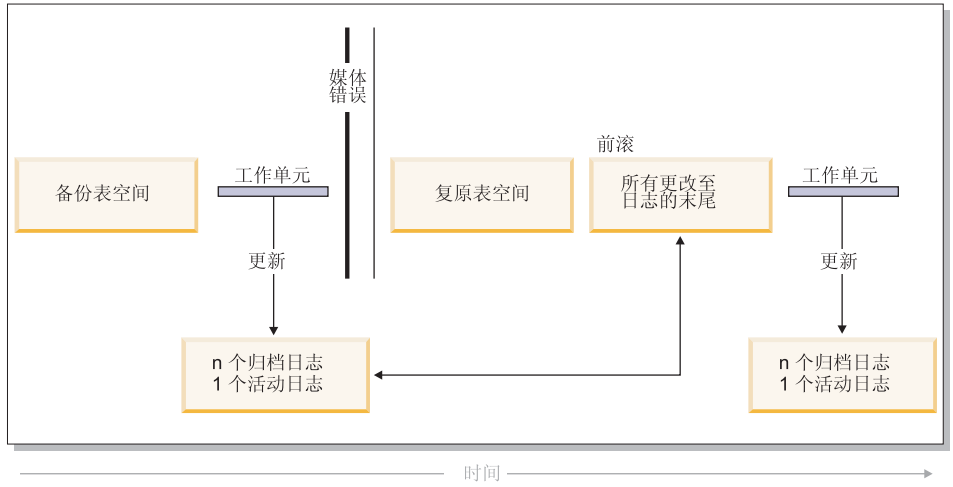


图 5. 表空间前滚恢复. 在运行时间较长的事务中，可以有多个活动的日志。

在分区数据库环境中，若要将表空间前滚至*一个时间点*，不必提供该表空间所驻留的节点（数据库分区）的列表。DB2® 将前滚请求提交给所有分区。这意味着必须在表空间驻留的所有数据库分区上复原表空间。

在分区数据库环境中，若要将表空间前滚至*日志末尾*，而又不希望在所有的分区上前滚表空间，则必须提供数据库分区的列表。若要将（所有分区上）所有处于前滚暂挂状态的表空间前滚到日志末尾，不必提供数据库分区的列表。缺省情况下，会将数据库前滚请求发送至所有分区。

相关概念:

- 第 30 页的『了解恢复日志』

相关参考:

- 第 117 页的『ROLLFORWARD DATABASE』

相关样本:

- 『dbrecov.out -- HOW TO RECOVER A DATABASE (C)』
- 『dbrecov.sqc -- How to recover a database (C)』

- 『dbrecov.out -- HOW TO RECOVER A DATABASE (C++)』
- 『dbrecov.sqC -- How to recover a database (C++)』

增量备份与恢复

由于数据库，尤其是仓库的大小持续扩充到太字节和拍字节的范围，备份和恢复这些数据库所需的时间及硬件资源也有了大幅的增长。因为对大数据库进行多个备份所需的存储量是巨大的，所以完整的数据库和表空间备份并不总是处理大数据库的最佳途径。请考虑以下问题:

- 仓库中只有一小部分数据更改时，应不需要备份整个数据库。
- 将表空间附加到现有数据库然后只备份表空间是危险的作法，原因是无法保证在表空间备份之间，已备份的表空间外没有任何更改。

DB2® 现在支持增量备份与恢复（但对长字段或大对象数据无效）。*增量备份*是一个备份映象，它只包含自上次进行备份以来有过更新的页。除了更新过的数据和索引页，每个增量备份映象还包含通常存储在完整备份映象中的初始数据库元数据（例如数据库配置、表空间定义和数据库历史等等）。

支持两种类型的增量备份:

- *增量* 增量备份映象是自从上次最新的、成功的完整备份操作以来，更改过的所有数据库数据的副本。也称为累积备份映象，因为进行的一系列增量备份中的每一个都会有上次增量备份映象的内容。增量备份映象的前身通常是同一对象最新的、成功的完整备份。
- *Delta* delta 备份映象或增量 delta 备份映象是自从上次相关表空间的成功备份（包括完整、增量或 delta 备份）以来，已更改过的所有数据库数据的副本。也称为差异备份映象或非累积备份映象。delta 备份映象的前身是最新的成功备份，包括 delta 备份映象中每个表空间的备份。

增量备份映象和 delta 备份映象的关键差别在于: 对连续不断更改的对象进行持续备份时，它们的行为不同。每个连续增量映象都包含了前一个增量映象的完整内容以及自上一次生成完整备份后更改过的或新增的任何数据。Delta 备份映象只包含自上次生成任何类型的映象后更改过的页。

允许以联机和脱机操作方式组合数据库和表空间增量备份。计划备份策略时应格外小心，因为将数据库增量备份和表空间增量备份结合在一起即暗示了数据库备份（或多个表空间的表空间备份）的前身不再需要是一个单独的映象，它可以是在不同时间进行的先前数据库和表空间备份的唯一集合。

要将数据库或表空间重新构建至一致状态，恢复进程必须以要复原的整个对象（数据库或表空间）的一致映象开始，然后必须按以下描述的次序应用每个相应的增量备份映象。

为启用对数据库更新的跟踪，DB2 支持新的数据库配置参数 *trackmod*，它可以是以下两个可接受的值中的一个:

- NO。增量备份不允许使用此配置。不会以任何方式跟踪或记录数据库页跟踪。这是缺省值。
- YES。增量备份允许使用此配置。启用了更新跟踪后，更改会对与数据库的第一个成功连接生效。必须选对该表空间进行完整备份，才能对特定表空间进行增量备份。

对于 SMS 和 DMS 表空间，此跟踪的粒度为表空间级。在表空间级的跟踪中，每个表空间的标志指示该表空间中是否存在需要备份的页。如果表空间不存在任何需要备份的页，则备份操作可以完全跳过该表空间。

对数据库的更新跟踪会对更新或插入数据的事务的运行时性能产生较小影响。

相关任务:

- 第 26 页的『从增量备份映象复原』

增量备份与恢复 — 详细信息

从增量备份映象复原

过程:

根据增量备份映象进行的复原操作通常包括下列步骤:

1. 标识增量目标映象。

   确定要复原的最终映象，并从 DB2 复原实用程序请求增量复原操作。此映象也称为增量复原的目标映象，因为它将成为要复原的最后映象。增量目标映象是使用 RESTORE DATABASE 命令中的 TAKEN AT 参数指定的。
2. 复原最新的完整数据库或表空间映象以建立一个基线，以根据它来应用每个后继的增量备份映象。
3. 按产生的次序，在“步骤 2”中复原的基线映象的顶部，恢复需要的各个完整增量备份映象或表空间增量备份映象。
4. 重复“步骤 3”直到“步骤 1”中的目标映象读了两次。在整个增量复原操作期间会访问两次目标映象。在第一次访问期间，只从映象读取初始数据，而不读取任何用户数据。只在第二次访问期间才读取并处理完整的映象。

必须访问两次增量复原操作的目标映象，以确保数据库最初是使用正确的历史、数据库配置以及将在复原操作期间创建的数据库表空间定义来配置的。如果自从进行了最初的完整数据库备份映象以来已删除了表空间，将从备份映象中读取该映象的表空间数据，但在增量复原处理期间会忽略该数据。

有两种方法来复原增量备份映象。

- 对于手工增量复原，必须对需要复原的每个备份映象发出一次 RESTORE 命令（如上面的步骤中所述）。
- 对于自动增量复原，但在指定要使用的目标映象时发出 RESTORE 命令一次。然后，DB2 使用数据库历史来确定余下的所需备份映象并复原它们。

手工增量复原示例

要复原一组增量备份映象，使用手工增量复原，并通过使用 RESTORE DATABASE 命令的 TAKEN AT *timestamp* 选项来指定目标映象，并遵循上面所述的步骤。例如:

1. db2 restore database sample incremental taken at <ts>

其中:

<ts> 指向要复原的最后一个增量备份映象（目标映象）

2. db2 restore database sample incremental taken at <ts1>

其中:

<ts1> 指向初始完整数据库（或表空间）映象

3. db2 restore database sample incremental taken at <tsX>

其中:

<tsX> 指向某个序列中的每个增量备份映象

4. 重复“步骤 3”，复原每个增量备份映象并包括 <ts>

如果对数据库复原操作使用手工增量复原并且已生成表空间备份映象，则必须以表空间映象的备份时间戳记的流年次序复原这些表空间映象。

如果想要使用手工增量复原，则可以使用 **db2ckrst** 实用程序来查询数据库历史并生成增量复原所需的备份映象时间戳记的列表。同时还生成用于手工增量复原的简化了的复原语法。建议保留备份的完整记录，并仅将此实用程序用作指南。

自动增量复原示例

要复原一组增量备份映象，使用自动增量备份，对 RESTORE DATABASE 命令指定“TAKEN AT 时间戳记”选项。使用想要复原的上一个映象的时间戳记。例如:

```
db2 restore db sample incremental automatic taken at 20001228152133
```

这样就可以使 DB2 复原实用程序自动执行本节开头所述的每个步骤。在处理的最初阶段，将读取时间戳记为 20001228152133 的备份映象，复原实用程序将验证数据库、其历史和表空间定义是否存在并有效。

在处理的第二阶段，将查询数据库历史来构建执行请求的复原操作所需的备份映象链。如果由于某些原因上述操作不可能实现，DB2 无法构建所需映象的完整链，复原操作会终止并返回错误消息。此时，不能进行自动复原，应发出带有 INCREMENTAL ABORT 选项的 RESTORE DATABASE 命令。这将清除所有余下的资源，以便可以继续手工增量复原。

注: 强烈建议不要使用 PRUNE HISTORY 命令的 FORCE 选项。此命令的缺省操作可防止您删除历史条目（从最新的完整数据库备份映象恢复时可能会需要它们），但如果使用 FORCE 选项，就有可能删除自动复原操作所需的条目。

在处理的第三个阶段期间，DB2 将复原生成链中余下的每个备份映象。如果在此阶段期间发生了错误，则将必须发出带有 INCREMENTAL ABORT 选项的 RESTORE DATABASE 命令来清除所有余下的资源。然后，必须确定是否能解决错误，才能重新发出 RESTORE 命令或再次尝试手工增量复原。

相关概念:

- 第 25 页的『增量备份与恢复』

相关参考:

- 第 84 页的『RESTORE DATABASE』
- 第 191 页的『db2ckrst — 检查增量复原映象序列』

自动增量复原的限制

1. 如果自想要从中进行复原的备份操作后已更改表空间名，且在发出表级别复原操作时使用了新名称，则将不能从数据库历史中正确生成所需的备份映象链，并发生错误（SQL2571N）。

   示例:

   ```
   db2 backup db sample -> <ts1>
   db2 backup db sample incremental -> <ts2>
   db2 rename tablespace from userspace1 to t1
   db2 restore db sample tablespace ('t1') incremental automatic taken
   at <ts2>

   SQL2571N 自动复原未能继续。原因代码: "3"。
   ```

   建议的解决办法: 使用手工增量复原。

2. 如果删除数据库，数据库历史也会删除。如果复原已删除的数据库，数据库历史将复原到它在复原备份时的状态，而自那以后的所有历史条目都将丢失。如

果稍后尝试执行需要使用其中任何一个已丢失的历史条目的自动增量复原，则 RESTORE 实用程序将尝试复原不正确的备份链，并返回“顺序不对”错误（SQL2572N）。

示例:

```
db2 backup db sample -> <ts1>
db2 backup db sample incremental -> <ts2>
db2 backup db sample incremental delta -> <ts3>
db2 backup db sample incremental delta -> <ts4>
db2 drop db sample
db2 restore db sample incremental automatic taken at <ts2>
db2 restore db sample incremental automatic taken at <ts4>
```

建议的解决办法:

- 使用手工增量复原。
- 在发出自动增量复原命令前，首先从映象 <ts4> 复原历史文件。

3. 如果将备份映象从一个数据库复原到另一个数据库，然后建立增量（delta）备份，则不能再使用自动增量复原来复原此备份映象。

   示例:

   ```
   db2 create db a
   db2 create db b

   db2 update db cfg for a using trackmod on

   db2 backup db a -> ts1
   db2 restore db a taken at ts1 into b

   db2 backup db b incremental -> ts2

   db2 restore db b incremental automatic taken at ts2

   QL2542N 根据提供的源数据库别名“B”和时间戳记“ts1”，
   找不到数据库映象文件的匹配项。
   ```

   建议的解决办法:

   - 使用如下所示的手工增量复原:

     ```
     db2 restore db b incremental taken at ts2
     db2 restore db a incremental taken at ts1 into b
     db2 restore db b incremental taken at ts2
     ```

   - 在对数据库 B 执行手工复原操作之后，发出完整数据库备份以启动新的增量链

相关概念:

- 第 25 页的『增量备份与恢复』

相关任务:

- 第 26 页的『从增量备份映象复原』

相关参考:

- 第 84 页的『RESTORE DATABASE』

了解恢复日志

所有数据库都有相关的日志。这些日志保存了有关数据库更改的记录。若需要将数据库复原至上一完整、脱机备份之前的一个点，则需要日志才能将数据前滚至故障点。

有两种类型的 DB2® 记录: *循环*和*归档*，每种类型都提供了不同级别的恢复能力:

- 当创建新数据库时，*循环*记录是缺省行为。（*logretain* 和 *userexit* *数据库配置参数设置为* NO。）对于这种类型的记录，只允许完整的脱机数据库备份。进行完整备份时，数据库必须脱机（用户不可访问）。正如它的名称所表示的那样，循环记录使用一个联机日志“环”，提供对事务故障和系统崩溃的恢复。仅使用和保留日志到确保当前事务的完整性这样一个程度。循环记录不允许将数据库在上次完整备份操作后执行的事务中前滚。上次备份操作后发生的所有更改都将丢失。因为这种类型的复原操作将数据恢复至进行完整备份的特定时间点，所以它称为*版本恢复*。

  第 31 页的图 6 显示当循环记录活动时，活动日志使用一个日志文件环。

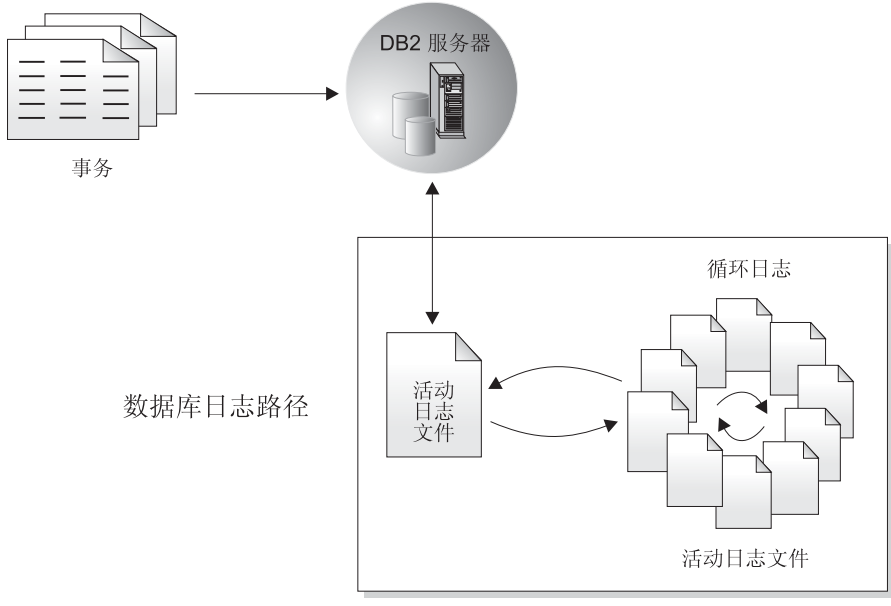


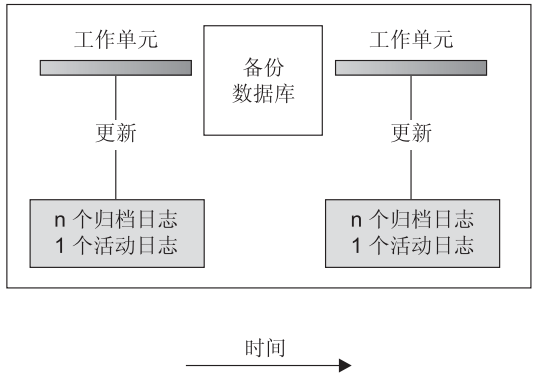
图 6. 循环记录

崩溃恢复期间，使用*活动*日志来防止故障（系统电源或应用程序错误）使数据库处于不一致的状态。RESTART DATABASE 命令在需要时使用活动日志来将数据库转变为一致和可用状态。在崩溃恢复期间将回滚记录在这些日志中的未落实的更改。那些已落实，但尚未从内存（缓冲池）写到磁盘（数据库容器）的更改将重做。这些操作确保了数据库的完整性。活动日志位于数据库日志路径目录中。

- *归档*记录是针对前滚恢复使用的。启用 *logretain* 和／或 *userexit* 数据库配置参数将导致归档记录。要对日志归档，可以选择让 DB2 将日志文件保存在活动路径中，然后手工归档它们，或者可以安装用户出口程序以自动进行归档。归档日志是曾经活动的，但不再是崩溃恢复所必需的日志。

选择归档记录的好处是前滚恢复可以使用归档日志和活动日志来将数据库重新构建至日志的结尾或特定时间点。归档日志文件可用来恢复备份完成之后所进行的更改。这不同于循环记录，在其中只能恢复至备份时间，并且在该时间之后进行的所有更改都会丢失。

仅当将数据库配置为归档记录时，才支持进行联机备份。在联机备份操作期间，将记录对数据库的所有活动。复原联机备份映象时，必须至少将日志前滚至完成备份操作的时间点。为此，日志在复原数据库时必须已归档并可用。在联机备份完成后，DB2 将强制关闭当前活动的日志，从而使日志归档。这样就可确保您的联机备份有一组完整的归档日志可用于恢复。



日志在备份操作之间使用，以跟踪对数据库的更改。

图 7. 前滚恢复中活动的和已归档的数据库日志. 在运行时间较长的事务中，可以有多个活动的日志。

有两个数据库配置参数可用来更改归档日志的存储位置: *newlogpath* 参数和 *userexit* 参数。更改 *newlogpath* 参数还会影响活动日志的存储位置。

要确定数据库日志日志路径目录中的哪些日志*范围*是归档日志，应检查数据库配置参数 *loghead* 的值。此参数指示活动的最低编号的日志。那些序号小于 *loghead* 的日志是归档日志且可移动。可使用控制中心检查此参数的值；也可使用命令行处理器和 GET DATABASE CONFIGURATION 命令来查看 “首个活动日志文件”。关于此配置参数的更多信息，请参阅*《管理指南: 性能》*一书。

相关概念:

- 第 32 页的『日志镜像』

相关参考:

- 第 297 页的附录 G, 『数据库恢复的用户出口』
- 『“第一个活动日志文件”配置参数 — loghead』（*《管理指南: 性能》*）

恢复日志详细信息

日志镜像

DB2® 支持数据库级的日志镜像。镜像日志文件有助于使您的数据库免受:

- 活动日志的意外删除
- 硬件故障导致的数据毁坏

如果担心活动日志可能已损坏（由于磁盘破坏所导致），应该考虑使用新的 DB2 配置参数 MIRRORLOGPATH，以指定要管理活动日志副本的数据库辅助路径，从而镜像存储这些日志的卷。

MIRRORLOGPATH 配置参数允许数据库将完全相同的第二份日志文件写至另一路径。建议您将第二个日志路径设置到在物理上独立的磁盘（最好该磁盘也在另一磁盘控制器上）。在那种情况下，磁盘控制器不能是单个故障点。

首次启用 MIRRORLOGPATH 时，在下次数据库启动前并未实际使用它。这与 NEWLOGPATH 配置参数相似。

如果写至活动日志路径或镜像日志路径时出错，数据库将把有问题的路径标记为“坏”，并将消息写至到管理通知日志，然后仅将后续的日志记录写至其它“好”日志路径中。DB2 将不会再次尝试使用“坏”路径，直到当前日志文件完成。当 DB2 需要打开下一个日志文件时，将会验证此路径是否有效，如果有效就开始使用它。如果无效，DB2 将不会尝试再次使用该路径，直到下一个日志文件被第一次访问时。不会尝试使日志路径同步，但 DB2 保留了关于发生的访问错误的信息，因此在归档日志文件时可使用正确的路径。如果在写至余下“好”路径时出现故障，则数据库关闭。

相关参考:

- 『“镜像日志路径”配置参数 — mirrorlogpath』（*《管理指南: 性能》*）

使用 NOT LOGGED INITIALLY 参数减少记录

若您的应用程序根据主表创建和填充了工作表，且您不担心这些工作表的可恢复性（因为它们可以从主表很容易地重新创建），则您可能希望在 CREATE TABLE 语句上指定 NOT LOGGED INITIALLY 参数创建工作表。使用 NOT LOGGED INITIALLY 参数的优点是，不记录在创建该表的同一个工作单元中对该表所作的任何更改（包括插入、删除、更新或创建索引操作）。这不仅减少了记录工作量，也可能提高应用程序的性能。还可以对现存表使用带 NOT LOGGED INITIALLY 参数的 ALTER TABLE 语句，来获得同样的结果。

注:

1. 可以在同一个工作单元中使用 NOT LOGGED INITIALLY 参数创建多个表。
2. 仍会记录对目录表和其它用户表的更改。

因为不记录表的更改，所以当决定使用 NOT LOGGED INITIALLY 表属性时应该考虑下列事宜:

- 落实时将把对表的*所有*更改清仓至磁盘。这意味着该落实可能占用更长的时间。

- 如果激活了 NOT LOGGED INITIALLY 属性且未记录发生的活动，则当语句失败或执行 ROLLBACK TO SAVEPOINT 时将回滚整个工作单元（SQL1476N）。
- 当前滚时，不能恢复这些表。如果前滚操作遇到使用 NOT LOGGED INITIALLY 选项创建或改变的表，则将该表被标记为不可用。在恢复了数据库之后，访问该表的任何尝试都将返回 SQL1477N。

  注: 当创建了一个表时，将会保持对目录表的行锁定，直到执行 COMMIT 为止。要利用不记录行为，必须在创建该表的同一个工作单元中填充该表。这就存在并发情况。

减少“声明临时表”的记录

若计划使用已说明临时表作为工作表，注意下列各项:

- 已说明临时表不是在目录中创建的；因此，不挂起锁定。
- 不对已说明临时表执行记录，甚至在第一个 COMMIT 之后也如此。
- 使用 ON COMMIT PRESERVE 选项来使各行在 COMMIT 之后仍留在表中；否则，将删除所有行。
- 只有创建已说明临时表的应用程序才能访问表的那个实例。
- 当删除应用程序与数据库的连接时，隐式删除该表。
- 工作单元中使用一个已说明临时表的操作中的错误不会导致该工作单元完全回滚。然而，更改已说明临时表内容的语句中的操作错误将删除该表中的所有行。回滚工作单元（或保存点）将删除已说明临时表中被该工作单元（或保存点）修改过的所有行。

相关概念:

- 『Application processes, concurrency, and recovery』（*SQL Reference, Volume 1*）

相关任务:

- 『创建表空间』（*《管理指南: 实现》*）

相关参考:

- 『DECLARE GLOBAL TEMPORARY TABLE statement』（*SQL Reference, Volume 2*）

数据库记录的配置参数

主日志（logprimary）

此参数指定将创建的大小为 *logfilsz* 的主日志的数量。

主日志，无论是空的还是满的，都需要相同的磁盘空间容量。因此，若配置的日志多于需要的日志，将会不必要地占用磁盘空间。若配置的日志太少，可能会遇到日志满载的情况。当选择要配置的日志数时，必须考虑建

立的每个日志的大小，以及应用程序是否可以处理日志满载的情况。对活动日志空间的总日志文件大小限制为 256 GB。

如果对现有数据库启用前滚恢复，则将主日志数更改为主日志和辅助日志之和，再加 1。对其启用前滚恢复的数据库中的 LONG VARCHAR 和 LOB 字段记录了附加信息。

辅助日志（logsecond）

此参数指定创建并用于恢复（如果需要的话）的辅助日志文件的数目。

如果主日志文件已满，可按需要一次分配一个辅助日志文件（大小为 *logfilsiz*），最多可分配由此参数指定的最大数目。如果此参数设置为 -1，则将数据库配置为无限活动日志空间。对在数据库上运行的飘浮事务的大小或数量没有任何限制。

注:

1. 必须启用 *userexit* 数据库配置参数才能将 *logsecond* 参数设置为 -1。
2. 如果此参数设置为 -1，则崩溃恢复时间可能会增加，原因是 DB2 可能需要检索归档日志文件。

日志文件大小（logfilsiz）

此参数以 4 KB 的页数指定每个配置日志的大小。

对可配置的总活动日志空间有 256 GB 的逻辑限制。此限制源自 *logfilsiz* 上的上限 262144 及（*logprimary* + *logsecond*）的上限 256。

日志文件的大小对性能有直接的影响。从一个日志切换至另一个日志需要付出性能代价。因此，从纯性能角度来说，日志文件大小越大越好。此参数还指示要归档的日志文件大小。这种情况下，日志文件大小越大并不一定越好，因为较大的日志文件大小增加了故障或导致日志交付方案中的延迟的发生机率。当考虑活动日志空间时，最好有较多的较小日志文件。例如，如果有 2 个很大的日志文件，并且事务启动接近一个日志文件的末尾，则仅有一半日志空间仍然可用。

每当释放数据库（与该数据库的所有连接都终止）时，就截断当前正写入的日志文件。因此，如果一个数据库被频繁释放时，最好不要选择较大日志文件大小，因为 DB2 将创建较大文件而只是截断它。可以使用 ACTIVATE DATABASE 命令来避免此成本，并且预先准备好缓冲池还有助于提高性能。

假定应用程序将数据库保持为打开以使打开数据库时的处理时间最短，日志文件大小应由建立脱机归档日志副本所花的时间确定。

将日志文件的丢失降低至最小程度，也是设置日志大小时的一个重要注意事项。归档会占用整个日志。若使用单个的大日志，则会增加归档之间的

时间。若媒体包含日志故障，某些事务信息将可能丢失。减小日志大小会增加归档的频率，但可以减少由于媒体故障而丢失信息，因为可以使用丢失的日志之前的那些较小的日志。

日志缓冲区（logbufsz）

此参数允许您指定在将日志记录写至磁盘之前用作这些记录的缓冲区的内存量。当发生下列任何一项事件时会将日志记录写入磁盘:

- 事务落实
- 日志缓冲区已满
- 发生了某些其它的内部数据库管理器事件。

增加日志缓冲区的大小可使与记录日志操作相关的输入／输出（I/O）活动更有效，因为将日志记录写到磁盘中的频率更低，而每次写入的记录却更多。

对组的落实次数（mincommit）

此参数允许您延迟将日志记录写入磁盘，直到执行了最小数目的落实为止。此延迟可有助于减少与写入日志记录相关的数据库管理器开销，这样若您有多个应用程序对数据库运行，且在很短的时间内该应用程序请求了许多落实，则可改进性能。

仅当此参数的值大于 1，且与该数据库连接的应用程序的数量大于此参数的值时，才会对落实进行这种分组。落实组合生效时，保持应用程序落实请求，直到经过 1 秒钟或落实请求数等于此参数的值为止。

新日志路径（newlogpath）

数据库日志最初是在 SQLOGDIR 中创建的，SQLOGDIR 是数据库目录的子目录。通过将此配置参数的值更改为指向另一目录或另一设备，可以更改放置活动日志及以后的归档日志的位置。如果数据库被配置为进行前滚恢复，则不要将当前存储在数据库日志路径目录中的归档日志移至新位置。

因为可以更改该日志路径的位置，因此前滚恢复所需的日志可以存在于不同的目录中或不同的设备上。在前滚操作期间可更改此配置参数的值，以允许您访问位于多个位置的日志。

必须跟踪这些日志的位置。

只有数据库处于一致状态时才会应用所作的更改。配置参数 *database_consistent* 将返回数据库的状态。

镜像日志路径（mirrorlogpath）

要防止主日志路径上的日志发生磁盘故障或被无意中删除的情况，可以指定在辅助（镜像）路径上维护完全相同的一组日志。要执行此操作，将此

配置参数的值更改为指向另一目录。如果数据库被配置为进行前滚恢复，则不要将当前存储在镜像日志路径目录中的归档日志移至新位置。

因为可以更改日志路径位置，因此前滚恢复所需的日志可以存在于不同的目录中。在前滚操作期间可更改此配置参数的值，以允许您访问位于多个位置的日志。

必须跟踪这些日志的位置。

只有数据库处于一致状态时才会应用所作的更改。配置参数 *database_consistent* 将返回数据库的状态。

要关闭此配置参数，将它的值设置为 DEFAULT。

注:

1. 如果主日志路径是原始设备，则此配置参数不受支持。
2. 对此参数指定的值不能是原始设备。

日志保留（logretain）

如果 logretain 设置为 RECOVERY，归档日志将保留在数据库日志路径目录中，而将数据库视为可恢复的数据库则意味着启用了前滚恢复。

用户出口（userexit）

此参数使数据库管理器调用用户出口程序来归档和检索日志。归档日志文件的位置不是活动日志路径。如果 *userexit* 设置为 ON，将启用前滚恢复。

用于存储脱机归档日志的设备和用于建立副本的软件的数据传送速度，最少必须与数据库管理器日志中写入数据的平均速率匹配。如果传送速度跟不上生成新日志数据的速度，而且记录活动持续足够长的时间，则可能会用尽磁盘空间。用尽磁盘空间的时间长短由可用磁盘空间量确定。若发生这种情况，数据库处理将停止。

当使用磁带或光盘媒体时，数据传送速度最为重要。某些磁带机需要相同的时间来复制文件，而与它的大小无关。必须确定归档设备的能力。

磁带机还有其它注意事项。归档请求的频率是很重要的。例如，如果用于完成任何复制操作的时间是 5 分钟，则日志应该足够大，以便在工作负荷高峰期间可保存 5 分钟的日志数据。磁带机可能存在设计限制，它会限制每天的操作数。当确定日志大小时，必须考虑这些因素。

注: 此值必须设置为 ON 才能启用无限活动日志空间。

注: *logretain* 和 *userexit* 数据库配置参数的缺省值不支持前滚恢复，必须对其进行更改才能使用它们。

溢出日志路径（overflowlogpath）

此参数可以用于几种函数，这取决于记录需求。可以指定位置来让 DB2 查

找前滚操作需要的日志文件。它与 ROLLFORWARD 命令的 OVERFLOW LOG PATH 选项相似，但是，不需要对发出的每个 ROLLFORWARD 命令指定 OVERFLOW LOG PATH 选项，可以只设置此配置参数一次。如果同时使用了这两个选项，则 OVERFLOW LOG PATH 选项将覆盖该前滚操作的 *overflowlogpath* 配置参数。

如果 *logsecond* 设置为 -1，则可以指定一个目录以让 DB2 存储从归档文件检索到的活动日志文件。（如果活动日志文件不再存在于活动日志路径中，则必须检索它们以用于回滚操作）。

如果未指定 *overflowlogpath*，则 DB2 会将日志文件检索到活动日志路径中。通过指定此参数，可以提供附加资源让 DB2 存储检索到的日志文件。好处包括将 I/O 成本分摊到不同磁盘中以及允许其它日志文件存储在活动日志路径中。

例如，如果对复制使用 **db2ReadLog** API，则可以使用 *overflowlogpath* 来指定一个位置让 DB2 搜索此 API 所需的日志文件。如果找不到日志文件（在活动日志路径或溢出日志路径中）且将数据库配置为启用 *userexit*，则 DB2 将检索日志文件。还可以使用此参数来指定目录以让 DB2 存储检索到的日志文件。好处包括降低活动日志路径上的 I/O 成本以及允许其它日志文件存储在活动日志路径中。

如果将原始设备配置为活动日志路径，则在想要将 *logsecond* 配置为 -1 或想要使用 **db2ReadLog** API 时必须配置 *overflowlogpath*。

要设置 *overflowlogpath*，指定一个最长 242 个字节的字符串。该字符串必须指向路径名，且它必须是全限定路径名，而不是相对路径名。该路径名必须是目录，而不是原始设备。

注: 在分区数据库环境中，节点号自动追加在路径后面。这样就维护了多个逻辑节点配置中路径的唯一性。

日志磁盘已满时停止（blk_log_dsk_ful）

可以设置此配置参数以防止当 DB2 不能在活动日志路径中创建新日志文件时出现磁盘已满的错误。相反，DB2 将尝试每 5 分钟创建日志文件，直到成功为止。每次尝试之后，DB2 都会将一条消息写至管理通知日志。确认应用程序因为日志磁盘已满情况而挂起的唯一的方法就是监视管理通知日志。在成功创建日志文件之前，尝试更新表数据的所有用户应用程序都不能落实事务。只读查询可能不会直接受影响；但是，如果查询需要存取被更新请求锁定的数据或由更新应用程序在缓冲池中修正的数据页时，只读查询也将象挂起一样。

将 *blk_log_dsk_ful* 设置为 YES 导致 DB2 遇到日志磁盘已满错误时应用程序挂起。于是您就能够解决错误，而事务继续运行。磁盘已满情况可以通过将旧的日志文件移至另一文件系统或增加文件的大小以使挂起应用程序能够完成来解决。

如果 blk_log_dsk_ful 设置为 NO，则接收到日志磁盘已满错误的事务将失败并被回滚。在某些情况下，如果事务导致日志磁盘已满错误，则数据库将崩溃。

相关概念:

- 第 39 页的『管理日志文件』
- 第 53 页的『增强恢复性能』

相关参考:

- 第 297 页的附录 G, 『数据库恢复的用户出口』
- 『“辅助日志文件数目”配置参数 — logsecond』（*《管理指南: 性能》*）
- 『“更改数据库日志路径”配置参数 — newlogpath』（*《管理指南: 性能》*）

管理日志文件

管理数据库日志时，考虑以下事项:

- 归档日志的编号方案以 S0000000.LOG 开始，直到 S9999999.LOG，符合日志文件的最大潜在大小 10000000。如果发生以下情况，数据库管理器将复位到 S0000000.LOG:
  - 数据库配置文件更改为启用前滚恢复
  - 数据库配置文件更改为*禁用*前滚恢复
  - 已使用了 S9999999.LOG。

  在复原数据库之后（执行或不执行前滚恢复），DB2® 重用日志名。数据库管理器会确保在前滚恢复期间不应用不正确的日志，但是它无法检测到必需的日志的位置。必须确保前滚恢复可以找到正确的日志。

  前滚操作成功完成后，使用的最后一个日志会截断，而记录日志操作会从下一个按顺序的日志开始。日志路径目录中，序列号大于用于前滚恢复的最后一个日志的任何日志都将重新使用。截断的日志中，跟在截断点后的任何条目都将用 0 覆盖。确保在调用前滚实用程序前建立了日志的副本。（可以调用用户出口程序来将日志复制到另一位置。）
- 如果某个数据库尚未激活（通过 ACTIVATE DATABASE 命令），DB2 会在所有应用程序已从该数据库断开连接后截断当前日志文件。下次有应用程序与该数据库连接时，DB2 开始把日志记录到一个新日志文件中。如果系统上产生了很

多小的日志文件，则可能需要考虑使用 ACTIVATE DATABASE 命令。使用该命令不仅能节省应用程序连接时初始化数据库所用的开销，还可节省分配大日志文件、截断它然后再分配新的大日志文件所用的开销。

- 因为日志文件名是重复使用的（参见图 8），所以归档日志可能会与某个数据库的两个或多个不同的*日志序列*相关。例如，如果您要恢复“备份 2”，可以使用两种可能的日志序列。如果在完整的数据库恢复期间前滚至某个时间点，然后在到达日志末尾前停止，则您已创建了一个新的日志序列。两个日志序列不能合在一起。如果联机备份映象跨越了第一个日志序列，则必须使用此日志序列来完成前滚恢复。

  如果在恢复之后创建了新的日志序列，则旧日志序列中的所有表空间备份映象都是无效的。这通常是在复原时识别的，但是如果在数据库复原操作之后立即执行表空间复原操作，则复原实用程序无法识别旧日志序列上的表空间备份映象。在实际上前滚了数据库前，将使用的日志序列都是未知的。如果表空间在旧日志序列上，它一定会由表空间前滚操作“捕获”。使用无效备份映象的复原操作可能会成功完成，但该表空间的表空间前滚操作将失败，而表空间将仍处于复原暂挂状态。

  例如，假设一个表空间级的备份操作“备份 3”在顶级日志序列的 S0000013.LOG 和 S0000014.LOG 之间完成（参见图 8）。如果要使用数据库级的备份映象“备份 2”进行复原和前滚，您将需要前滚 S0000012.LOG。然后，可以继续前滚到顶端日志序列或（较新的）底端日志序列。如果前滚底端日志序列，将不能使用表空间级的备份映象“备份 3”来执行表空间复原和前滚恢复。

  要使用表空间级的备份映象“备份 3”完成到日志末尾的表空间前滚操作，需要复原数据库级的备份映象“备份 2”，然后使用顶端日志序列前滚。一旦复原了表空间级的备份映象“备份 3”，就可以启动到日志末尾的前滚操作。

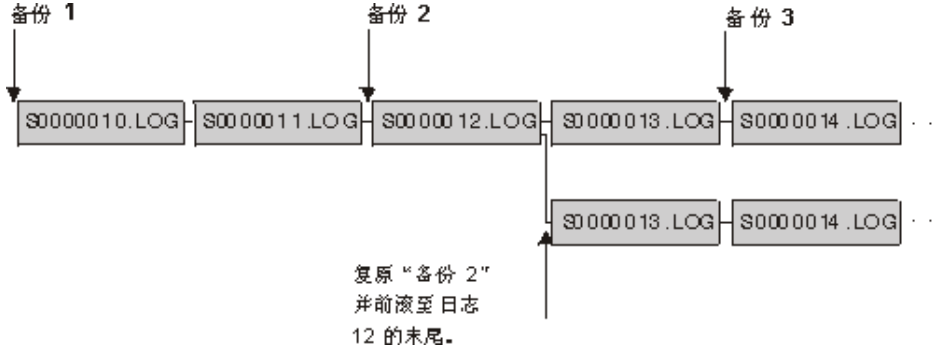


图 8. 重新使用日志文件名

相关参考:

- 第 297 页的附录 G, 『数据库恢复的用户出口』

通过用户出口程序管理日志文件

在调用用户出口程序来归档和检索日志文件时应注意以下注意事项:

- 数据库配置文件参数 *userexit* 指定在数据库的前滚恢复期间，数据库管理器是否调用用户出口程序来归档文件或检索日志文件。当前滚实用程序需要一个在日志路径目录中找不到的日志文件时，会发出检索日志文件的请求。

  注: 在 Windows® 操作系统上，不能使用 REXX 用户出口来归档日志。
- 在归档过程中，当一个日志文件满时，即使该日志文件仍是活动的且需要用于正常的处理，仍将它传送至用户出口。这使得数据的副本可以尽快地从易丢失数据的媒体中转移。传送至用户出口的日志文件保存在日志路径目录中，直到不再需要它用于正常处理为止。这样就重新使用了磁盘空间。
- DB2® 在启动用户出口程序来归档文件时，以读方式打开日志文件。在某些平台上，这可防止用户出口程序能删除日志文件。其它平台（如 AIX）允许进程（包括用户出口程序）删除日志文件。用户出口程序在日志文件归档后永远不能删除它，因为该文件可能仍是活动的并是崩溃恢复所需的。DB2 管理在归档日志文件时重新使用的磁盘空间。
- 当日志文件已归档并且不是活动的时，DB2 不会删除该文件但在需要这样的文件时将它重命名为下一个日志文件。这将改进性能，因为创建新的日志文件（而不是重命名文件）会写出所有页以保证有足够的磁盘空间。重用磁盘上的页比释放这些页再重新获得所需的页更有效率。
- 在崩溃恢复或回滚期间，DB2 将不调用用户出口程序来检索日志文件，除非 *logsecond* 数据库配置参数设置为 -1。
- 用户出口程序不保证可前滚恢复至故障点，只试图将故障范围减小。日志文件填充完后，排队等待用户出口例程使用。若在一个日志文件写完之前包含该日志的磁盘发生故障，则该日志文件中的数据将会丢失。另外，由于文件是排除等待归档的，磁盘可能会在备份了所有文件之前就发生故障，从而导致队列中的任何日志文件都丢失。
- 每个日志文件的配置大小对用户出口都有直接影响。若每个日志文件都非常大，磁盘发生故障时将会丢失大量数据。如果数据库是由小的日志文件配置的，会导致更频繁地将数据传送至用户出口例程。

  然而，若您要将数据移到慢速设备（如磁带）上，您可能希望有较大的日志文件以防建立该队列。若该队列变满时，将不处理归档与检索请求。只有在队列上有空间时，处理才继续。未处理的请求将不会自动重新排队。

- 只有配置了 *userexit* 才会执行向用户出口程序发出的归档请求，且每次都要填充一个活动的日志文件。有可能当最后与数据库断开时活动的日志文件仍然未满，但仍然会对填充了一部分的活动日志文件调用用户出口程序。

  注: 要释放未用的日志空间，在将日志文件归档前要将它截断。
- 应将该日志的副本保存到另一个物理设备上，以便在包含该日志文件的设备出现媒体故障时，前滚恢复可使用这个脱机日志文件。这个设备不应是包含该数据库数据文件的同一设备。
- 如果已启用用户出口程序并将磁带机用作日志和备份映象的存储设备，则需要确保备份映象和归档日志的目标位置不是同一磁带机。因为某些日志归档可能在备份操作进行时发生，所以当这两个进程尝试同时写至同一磁带机时可能发生错误。
- 在某些情况下，若在接收到用户出口程序对归档请求的肯定应答前数据库已经关闭，那么数据库管理器会在数据库打开时发送另一个请求。因此，一个日志文件会多次归档。
- 若用户出口程序接收到对一个不存在的文件进行归档的请求（因为有多个归档请求，而在第一次归档操作成功后即将该文件删除），或接收到对一个不存在的文件进行检索的请求（因为此文件位于另一个目录中，或已到达日志的末尾），则应忽略此请求并发送一个成功的返回码。
- 用户出口程序应该允许时间点恢复之后存在具有相同名称的不同日志文件；用户出口程序应编写为保留这两个日志文件，并将这些日志文件与正确的恢复路径相关联。
- 如果对使用同一磁带机归档日志文件的两个或多个数据库启用了用户出口程序，并且在其中一个数据库上发生了前滚操作，则其它数据库不应该是活动的。如果另一数据库尝试在前滚操作进行时归档日志文件，则可能找不到前滚操作所需的日志，或归档至磁带机的新日志文件可能覆盖先前存储在该磁带机上的日志文件。

  为防止这两种情况发生，可以确保在前滚操作期间在调用用户出口程序的节点上没有任何其它数据库是打开的，或者编写一个用户出口程序来处理这种情况。

相关概念:

- 第 39 页的『管理日志文件』

日志文件分配和除去

如果崩溃恢复可能需要数据库日志目录中的日志文件，就一定不能除去它们。当启用 *userexit* 数据库配置参数时，仅在崩溃恢复不再需要完整日志文件之后，才考虑除去该日志文件。崩溃恢复需要的日志文件称为活动日志。崩溃恢复不需要的日志文件称为归档日志。

分配新日志文件和除去旧日志文件的过程取决于 *userexit* 和 *logretain* 数据库配置参数的设置:

logretain **和** *userexit* **都设置为 OFF**

将使用循环记录。对于循环记录，不支持前滚恢复，但支持崩溃恢复。

在循环记录期间，不会生成新日志文件（辅助日志除外）且不删除旧日志文件。日志文件以循环的方式来处理。这就是说，当最后一个日志文件已满时，DB2® 开始写入第一个日志文件。

如果所有日志文件都是活动的且循环记录进程不能回绕至第一个日志文件，则日志已满情况就可能发生。当所有主日志活动且已满时，会创建辅助日志文件。一旦创建了辅助日志，在重新启动数据库之前就不能删除它。

Logretain **设置为 ON 且** *userexit* **设置为 OFF**

启用前滚恢复和崩溃恢复。已知数据库是可恢复的。当 *userexit* 设置为 OFF 时，DB2 不会从数据库日志目录中删除日志文件。每当日志文件已满，DB2 就开始将记录写至另一个日志文件，并且（如果尚未达到主辅日志的最大数目）创建新日志文件。

Userexit **设置为 ON**

当 *logretain* 和 *userexit* 设置为 on 时，会启用前滚恢复和崩溃恢复。当日志文件已满时，使用用户提供的用户出口程序对它自动归档。

通常不会删除日志文件。相反，当需要新的日志文件而没有可用的日志文件时，则重命名归档日志文件并再次使用它。一旦归档日志文件已关闭并被复制到日志归档目录中，就不会被删除或重命名。DB2 等待直到需要新日志文件，然后重命名最旧的归档日志。在恢复期间已移至数据库目录的日志文件，当不再需要该文件时就会在恢复进程期间除去它。在 DB2 用尽日志空间之前，可在数据库目录中看到旧的日志文件。

如果归档日志文件时遇到错误，则在再次尝试归档之前，日志文件的归档将暂挂 5 分钟。然后，DB2 将继续归档日志文件（当它们已满时）。在 5 分钟等待时间段内变满的日志文件在延迟之后将不会立即归档，DB2 将在一段时间内分开进行这些文件的归档。

除去旧日志文件的最简单方法是重新启动数据库。一旦重新启动了数据库，就只有新日志文件和用户出口程序归档失败的日志文件将在数据库目录中找到。

当数据库重新启动时，数据库日志目录中日志的最小数目将等于主日志的数目，这些主日志是可以通过使用 *logprimary* 数据库配置参数来配置的。可在日志目录中找到多于主日志数目的日志。如果关闭数据库时日志目录中空日志数大于重新启动数据库时 *logprimary* 配置参数的值，则可能发生此情况。如果在关闭数据库与重新启动数据库之间更改了 *logprimary* 配置参数的值，或者如果分配了辅助日志且从未使用，则将发生此情况。

当重新启动数据库时，如果空日志数小于 *logprimary* 配置参数指定的主日志的数目，则分配附加日志文件以弥补所差的日志数目。如果数据库目录中有多于主日志的空日志可用，则在重新启动数据库，可在数据库目录中找到所有可用的空日志。数据库关闭之后，已创建的辅助日志文件在重新启动数据库时将保留在活动日志路径中。

日志目录文件已满时停止事务

可以设置 *blk_log_dsk_ful* 数据库配置参数以防止当 DB2® 不能在活动日志路径中创建新日志文件时出现“磁盘已满”错误。

DB2 会在每 5 分钟即尝试创建日志文件直到成功为止。如果配置数据库时 *userexit* 参数设置为 `ON`，DB2 还会检查日志文件的归档操作是否已完成。如果有归档日志文件已成功归档，则 DB2 将不活动日志文件重命名为新的日志文件名并继续。每次尝试之后，DB2 都会将一条消息写至管理通知日志。可以确认应用程序因为日志磁盘已满情况而挂起的唯一方法就是监视管理通知日志。

在成功创建日志文件之前，尝试更新表数据的任何用户应用程序都不能落实事务。只读查询可能不会直接受影响，但如果查询需要访问被更新请求锁定的数据或由更新应用程序在缓冲池中修正的数据页时，只读查询的状态也会象挂起一样。

相关概念:

- 第 30 页的『了解恢复日志』
- 第 41 页的『通过用户出口程序管理日志文件』

按需进行的日志归档

DB2® 现在支持随时关闭（如果启用了用户出口选项，还包括归档）可恢复数据库的活动日志。因此您可以到某个已知点为止的一组完整的日志文件，然后使用这些日志文件来更新一个备用数据库。

通过调用 ARCHIVE LOG 命令或通过调用 **db2ArchiveLog** API，可以启动按需进行日志归档操作。

相关参考:

- 第 199 页的『ARCHIVE LOG』
- 『db2ArchiveLog - Archive Active Log』（*Administrative API Reference*）

使用原始日志

可使用原始设备来存储数据库日志。这样做既有优点，又有缺点。

- 优点是:
  - 可以将 26 个以上的物理驱动器与一个系统连接。
  - 文件 I/O 路径的长度较短。这可提高系统的性能。应执行基准测试，以评估对于工作负荷是否有可度量的效益。
- 缺点是:
  - 该设备不能被其它应用程序共享；*必须*将整个设备分配给 DB2。
  - 任何从该设备中备份或复制的操作系统实用程序或第三方工具不能在该设备上操作。
  - 若指定了错误的物理驱动器号，可以很容易地擦除现存驱动器上的文件系统。

可用 *newlogpath* 数据库配置参数配置原始日志。但在这样做以前，要考虑以上列出的优点和缺点，以及以下列出的附加注意事项:

- 仅允许一个设备。在操作系统级，可在多个磁盘上定义设备。DB2® 将进行操作系统调用以确定以 4 KB 页为单位的设备大小。

  如果使用多个磁盘，这样做可提供更大的设备，而由此产生的分割可通过提供更快的 I/O 吞吐量来改进性能。
- DB2 将试图写入设备的最后一个 4 KB 页。若设备大小大于 2 GB，则在不支持 2 GB 以上的设备的操作系统上，写至最后一页的尝试将失败。在这种情况下，DB2 将尝试使用所有页，直至达到受支持的极限。

  关于设备大小的信息用来指示在操作系统的支持下可用于 DB2 的设备的大小（以 4 KB 页为单位）。DB2 可写入的磁盘空间容量称为*可用的设备大小*。

  DB2 不使用设备的第一个 4 KB 页（此空间通常由操作系统用于其它用途）。这意味着可用于 DB2 的总空间是*设备大小 = 可用设备大小 - 1*。
- 不使用 *logsecond* 参数。DB2 将不分配辅助日志。活动日志空间的大小是由 *logprimary* x *logfilsiz* 而得的 4 KB 页的倍数。

- 仍将日志记录分组为日志块，每个日志块具有 4 KB 页的日志文件大小（*logfilsiz*）。日志块在原始设备中是连续的。每个数据块还包括用于存放数据块头的额外两页。这意味着设备可支持的*可用日志块数*是*设备大小*/（*logfilsiz* + 2）
- 设备必须足够大，以支持活动日志空间。即，*可用日志块数*必须大于（或等于）为 *logprimary* 配置参数指定的值。如果启用了 *userexit* 配置参数，则确保原始设备可以包含比对 *logprimary* 配置参数指定的值更多的日志。这将补偿用户出口程序归档日志文件时引起的延迟。
- 若使用的是循环记录，*logprimary* 配置参数将确定要写入设备的日志块的数量。这可能导致设备上出现未使用的空间。
- 若使用日志保留（*logretain*）而不用用户出口程序，当用完所有*可用日志块数*后，产生更新的所有操作将接收到日志已满的错误。此时，必须关闭数据库并执行脱机备份以确保可恢复性。在数据库备份操作之后，写至设备的日志记录将丢失。这表明不能使用更早的数据库备份映象来复原该数据库，然后将其前滚。若在*可用日志块数*全部用完之前建立了一个数据库备份，则可以复原并前滚该数据库。
- 若将日志保留（*logretain*）和用户出口程序一起使用，在用日志记录填充每个日志块时，为每个日志块调用用户出口程序。该用户出口程序必须能够读取设备，并将归档的日志存储为文件。DB2 将不调用用户出口程序来将日志文件检索到原始设备。而是，在前滚恢复期间，DB2 读取数据块头以确定原始设备是否包含所需的日志文件。若在原始设备中找不到必需的日志文件，DB2 将搜索溢出日志路径。若仍找不到日志文件，DB2 将调用用户出口程序以便在溢出日志路径中检索日志文件。如果不对前滚操作指定溢出日志路径，DB2 将不会调用用户出口程序来检索日志文件。
- 如果将原始设备配置为用于记录，并且正在使用 DataPropagator™（DPROP）或者调用 **db2ReadLog** API 的另一应用程序，则必须配置 *overflowlogpath* 数据库配置参数。DB2 可以调用用户出口程序来检索日志文件并返回 **db2ReadLog** API 所请求的日志数据。检索到的日志文件将放在 *overflowlogpath* 数据库配置参数指定的路径中。

相关任务:

- 『指定原始 I/O』（*《管理指南: 实现》*）

相关参考:

- 第 238 页的『db2ReadLog — 异步读取日志』
- 第 293 页的附录 F, 『Tivoli Storage Manager』

如何防止丢失日志文件

在需要删除数据库或执行时间点前滚恢复的情况下，可能会丢失以后恢复操作所需的日志文件。在这些情况下，对当前数据库日志路径目录中的所有日志进行复制是很重要的。考虑以下方案:

- 如果计划在复原操作之前删除数据库，则需要在发出 DROP DATABASE 命令之前将日志文件保存在活动日志路径中。在数据库复原后，前滚恢复可能需要这些日志文件，因为在删除数据库之前其中一些日志文件可能尚未归档。通常，不需要在发出 RESTORE 命令之前删除数据库。但是，可能必须删除数据库（或者通过指定 DROP DATABASE 命令的 AT NODE 选项来删除一个分区上的数据库），因为已经损坏到 RESTORE 命令失败的程度。还可能决定在复原操作之前删除数据库以为您自己提供新的开始。
- 如果将数据库前滚至特定时间点，则指定时间戳记之后的日志数据将被覆盖。如果在完成时间点前滚操作并重新连接至数据库之后确定实际上需要将数据库前滚至稍后的时间点，将无法实现此操作，因为日志已被覆盖。可能原始一组日志文件已归档，但是 DB2® 可调用用户出口程序来自动归档新生成的日志文件。视编写用户出口程序的情况不同，这可能导致归档日志目录中的原始的一组日志文件被覆盖。即使原始的和新的一组日志文件存在于归档日志目录中（同一文件的不同版本），还必须确定应对以后的恢复操作使用哪一组日志。

相关概念:

- 第 30 页的『了解恢复日志』

了解恢复历史文件

恢复历史文件是与每个数据库一起创建的，且在发生下列情况时自动更新:

- 备份了数据库或表空间
- 复原了数据库或表空间
- 前滚了数据库或表空间
- 创建了表空间
- 改变了表空间
- 停顿表空间
- 重命名表空间
- 删除表空间
- 装入表
- 删除表
- 重组表

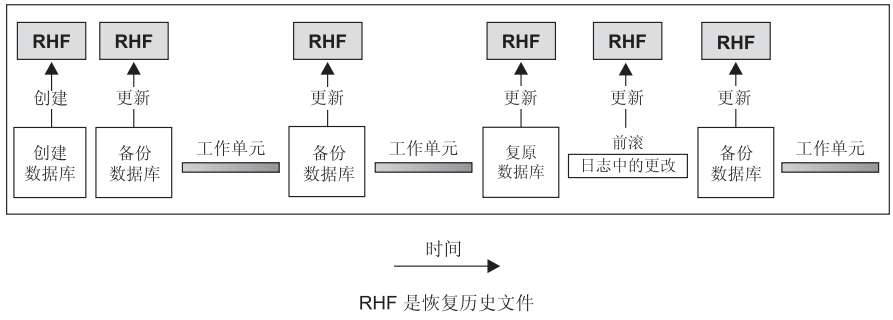


图 9. 创建和更新恢复历史文件

可以使用此文件中汇总的备份信息，将数据库的全部或部分恢复至给定的时间点。该文件中的信息包括:

- 唯一地标识每个条目的标识（ID）字段
- 已复制的部分数据库和复制方法
- 建立副本的时间
- 副本的位置（指示设备信息和访问副本的逻辑方式）
- 进行上次复原操作的时间
- 重命名表空间的时间，显示了该表空间的先前名称和当前名称
- 备份操作的状态: 活动、不活动、到期的或删除的
- 数据库备份保存的或前滚恢复操作期间处理的最后一个日志序号。

要查看恢复历史文件中的条目，使用 LIST HISTORY 命令。

每个备份操作（数据库备份、表空间备份或增量备份）都包括复制恢复历史文件。将该恢复历史文件链接至数据库。删除数据库会删除恢复历史文件。将数据库复原至新位置会复原该恢复历史文件。复原不会覆盖现有历史恢复文件，除非存在于磁盘上的文件没有任何条目。如果是这种情况，将根据备份映象复原数据库历史。

若当前数据库不能使用或不可用，且相关的恢复历史文件被损坏或被删除，则 RESTORE 命令中的一个选项只允许复原恢复历史文件。这样，可以复查该恢复历史文件，以提供有关要将哪个备份用于复原该数据库的信息。

该文件的大小由 *rec_his_retentn* 配置参数控制，该参数指定该文件中条目的保存期（以天计）。即使将此参数的值设置为零（0），仍会保留最新的完整数据库备份

（加上它的复原集）。（除去此副本的唯一方法是使用带 FORCE 选项的 FRUNE。）保存期的缺省值是 366 天。可使用 -1 将保存期设置为无限多天。在这种情况中，需要显式删除该文件。

相关参考:

- 『 “恢复历史保留周期” 配置参数 — rec_his_retentn 』（ *《管理指南: 性能》* ）
- 第 202 页的『 LIST HISTORY 』

恢复历史文件 — 无用信息收集

无用信息收集

虽然可以随时使用 PRUNE HISTORY 命令来从历史文件中除去条目，但还是建议将这种修剪任务留给 DB2 来处理。记录在恢复历史文件中的 DB2® 数据库备份的数目由 DB2 *无用信息收集*自动监视。调用 DB2 无用信息收集:

- 在完整非增量数据库备份操作成功完成之后。
- 在数据库复原操作（此处不需要前滚操作）成功完成之后。
- 在成功数据库前滚操作成功完成之后。

配置参数 *num_db_backups* 定义保留了多少活动完整（非增量）数据库备份映象。此参数的值用于从上一个条目开始扫描历史文件。

在每个完整（非增量）数据库备份操作之后，可使用 *rec_his_retentn* 配置参数来从历史文件中删除到期的条目。

*活动的数据库备份*是可使用当前日志进行复原和前滚，以恢复该数据库的当前状态的一种备份。而*不活动的数据库备份*是在复原时会将数据库移动回先前状态的一种备份。

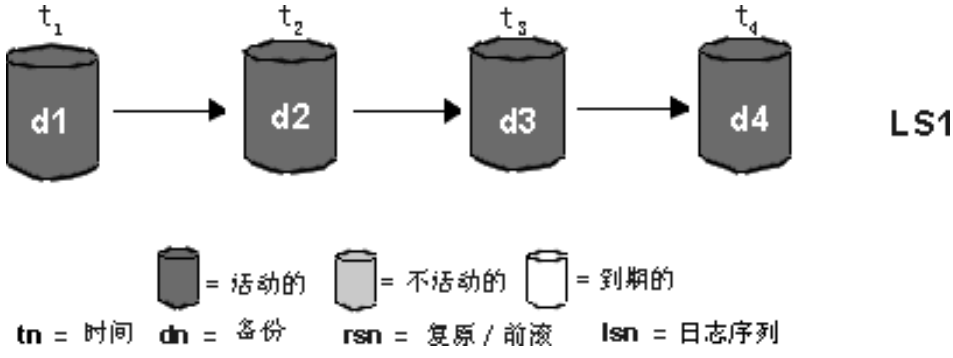


图 10. 活动数据库备份. num_db_backups 的值已设置为 4。

不再需要的所有活动数据库备份映象都标记为“到期的”。因为有更新的备份映象可用，所以可将这些映象视为不需要的。在该数据库备份映象到期前所建立的所有表空间备份映象和装入副本也标记为“到期的”。

标记为“不活动的”所有数据库备份映象以及在建立已到期的数据库备份映象前已建立的备份映象，也将标记为“到期的”。所有相关的“不活动”表空间备份映象和装入副本也标记为“到期的”。

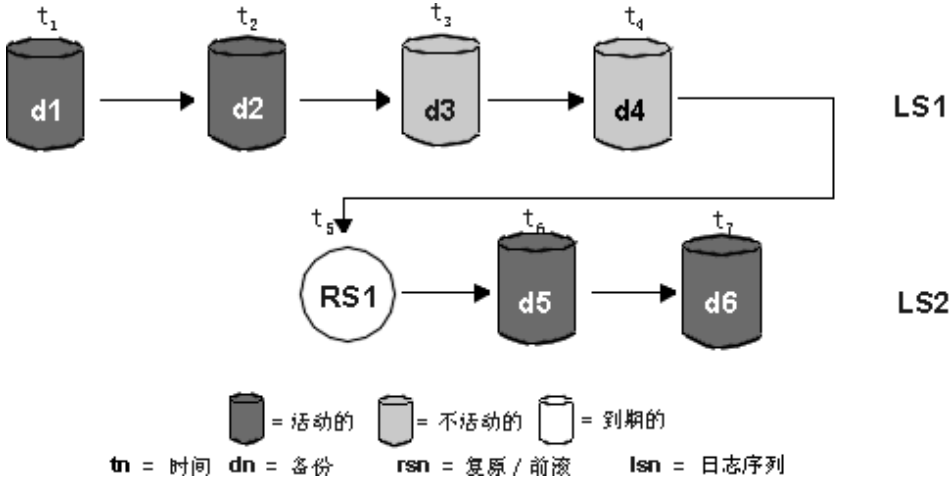


图 11. 不活动数据库备份

若复原了一个活动的数据库备份映象，但它不是历史文件中记录的最新数据库备份，则属于同一个日志序列的任何后续数据库备份映象将被标记为“不活动的”。

若复原了一个不活动的数据库备份映象，则属于当前日志序列的任何不活动的数据库备份都被再次标记为“活动的”。不再在当前日志序列中的所有数据库备份映象都标记为“不活动的”。

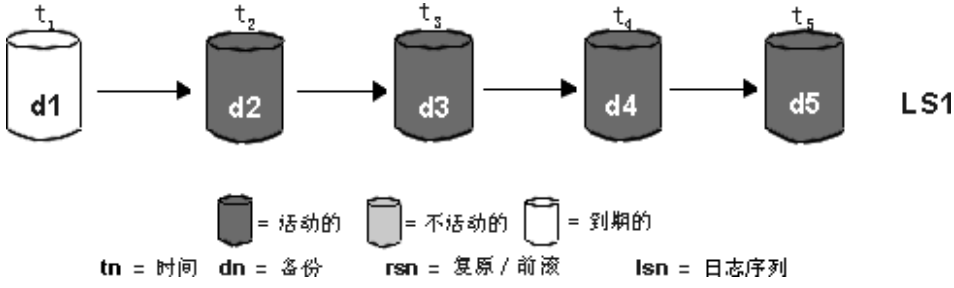


图 12. 到期数据库备份

DB2 无用信息收集也负责将 DB2 数据库或表空间备份映象的历史文件条目标记为“不活动”，条件是该备份与当前*日志序列*（也称为当前*日志链*）不对应。当前日志序列由已复原的 DB2 数据库备份映象和已处理的日志文件确定。一旦数据库备份映象复原后，所有后继的数据库备份映象都变得“不活动”，因为已复原的映象将开始一个新的日志链。（在不通过前滚来复原备份映象时实现。如果已发生了前滚操作，则日志链中在中断后建立的所有数据库备份都将标记为“不活动”。可以想像，由于已对包含已损坏的当前备份映象的整个日志序列执行了前滚实用程序，所以必须复原旧的数据库备份映象。）

如果在复原表空间级的备份映象后，数据库的当前状态不能通过应用当前日志序列来达到，则表空间级的备份映象会变为“不活动”。

如果备份映象包含 DATALINK 列，将联系负责运行 DB2 Data Links Manager 的所有 Data Links 服务器以请求进行无用信息收集。然后 DB2 无用信息收集会删除包含在到期备份中的相关 Data Links 服务器文件备份，但不删除在下次数据库备份操作之前解链的那些备份。

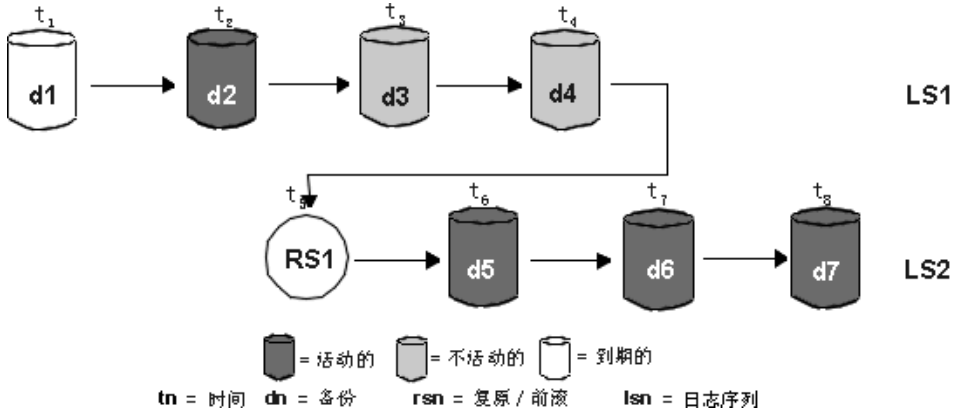


图 13. 混合活动的、不活动的和到期的数据库备份

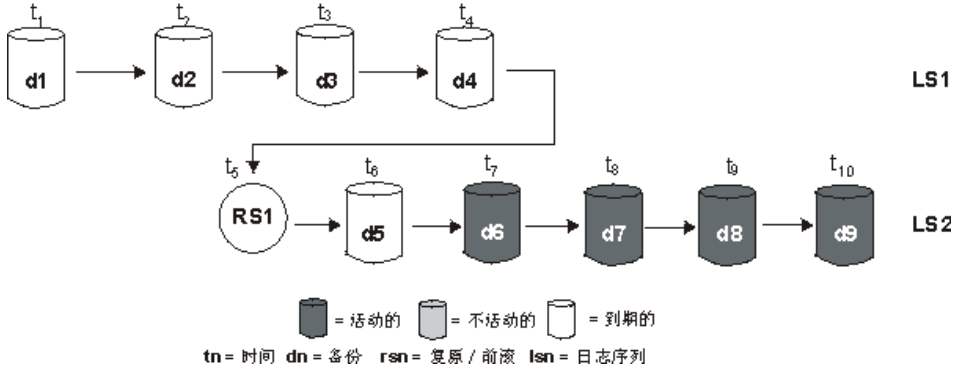


图 14. 到期日志序列

相关概念:

- 第 47 页的『了解恢复历史文件』

相关参考:

- 第 205 页的『PRUNE HISTORY/LOGFILE』

了解表空间状态

表空间的当前状态由它的*状态*反映。与恢复相关的最常见表空间状态是:

- *前滚暂挂*。表空间在复原后可发生了输入 / 输出（I/O）错误后被置于此状态。复原后，表空间可前滚到日志的末尾可某时间点。发生了 I/O 错误后，表空间必须前滚到日志的末尾。

- *正在前滚*。表空间在对它进行的前滚操作正在进行中时，被置于此状态。一旦前滚操作成功完成，表空间就不再处于“正在前滚”状态。如果取消了对表空间的前滚操作，表空间也会结束此状态。
- *复原暂挂*。如果取消了对表空间的前滚操作，或对表空间的前滚操作遇到了不可恢复的错误（此时必须再次复原并前滚表空间），会将表空间置于此状态。
- *备份暂挂*。在前滚操作的某个时间点后，或不带有复制选项的装入操作后，表空间将置于些状态。在可使用该表空间之前必须对其备份。（如果未进行备份就不能更新表空间，但允许只读操作。）

增强恢复性能

当考虑恢复性能时，应注意下列各项:

- 可以将日志置于单独的设备上，以提高频繁更新的数据库的性能。在联机事务处理（OLTP）环境中，更常见的是需要 I/O 来将数据写至日志而不是存储数据行。将日志置于单独的设备上，可将在日志和数据库文件之间进行移动所需的磁盘臂移动最小化。

  还应该考虑该磁盘上的其它文件。例如，将日志移至一个系统中用于系统调页的磁盘，而该磁盘没有足够的实内存，这样会破坏调整。
- 要缩短完成一次复原操作所需的时间:
  - 调整复原缓冲区大小。缓冲区大小必须是在备份操作期间使用的缓冲区大小的倍数。
  - 增加缓冲区的数量。

    如果使用多个缓冲区和 I/O 媒体设备，应该使用的缓冲区数至少是媒体设备数的两倍以确保它们不必等待数据。所用的缓冲区大小也影响复原操作的性能。理想的复原缓冲区大小应该是表空间的范围大小的倍数。

    若有多个具有不同范围大小的表空间，则指定是最大范围大小的倍数的一个值。

    建议的最小缓冲区数目是媒体设备的数目加上对 PARALLELISM 选项指定的数目。
  - 使用多个源设备。
  - 对于复原操作，将 PARALLELISM 选项设置为至少比源设备数大 1。
- 若一个表包含大量的长整数字段和 LOB 数据，则复原它可能会占用非常多的时间。若允许数据库进行前滚恢复，则 RESTORE 命令能够复原选择的表空间。若该长整数字段和 LOB 数据对于您的业务很重要，应参照完成这些表空间的备份任务所需的时间考虑复原这些表空间。通过将长整数字段和 LOB 数据存储在单独的表空间中，不选择复原包含该长整数字段和 LOB 数据的表空间，可以减

少完成复原操作所需的时间。若可从单独的源复制 LOB 数据，则当创建或改变一个表以包括 LOB 列时选择 NOT LOGGED 选项。若选择不复原包含长整数字段和 LOB 数据的表空间，但需要复原包含该表的表空间，则必须前滚到日志末尾，以便所有包含表数据的表空间都是一致的。

注: 若备份包含表数据的一个表空间，而未备份相关的长整数或 LOB 字段，则不能对该表空间执行时间点前滚恢复。必须将一个表的所有表空间同时前滚至同一个时间点。

- 下列说明适用于备份和复原操作:
  - 应使用多个 I/O 缓冲区和设备。
  - 分配至少两倍于正在使用的设备数量的缓冲区。
  - 不要使 I/O 设备控制器带宽超载。
  - 使用数量多的小缓冲区，而不使用数量少的大缓冲区。
  - 根据系统资源调整缓冲区的数量和大小。
  - 使用 PARALLELISM 选项

增强恢复性能 — 并行恢复

并行恢复

DB2® 使用多个代理进程来执行崩溃恢复和数据库前滚恢复。您可能会发现执行这些操作期间的性能更佳（尤其是在对称多处理机（SMP）上）；在数据库恢复期间使用多个代理进程可利用 SMP 机器上可用的多余 CPU。

并行恢复引入的代理进程类型是 db2agnsc。DB2 根据机器上 CPU 的数量，选择将用于数据库恢复的代理进程的数量。

DB2 将日志记录分发到这些代理进程以使它们可在适当的时候并发地重新应用。例如，可通过此方法使那些与插入、删除、更新、添加键和删除键操作相关的日志记录的处理并行化。因为日志记录是在页级并行化的（相同数据页上的日志记录由相同的代理进程处理），所以即使所有工作都是对一个表完成的，性能还是会增强。

相关概念:

- 第 53 页的『增强恢复性能』

第 2 章 数据库备份

本节描述了 DB2 UDB 备份实用程序，它用于创建数据库或表空间的备份副本。

包括下列主题:

备份概述

DB2® BACKUP DATABASE 命令的最简单的格式只需要您指定想要备份的数据库的别名。例如:

```
db2 backup db sample
```

如果命令成功完成，您将获得一个新的备份映象，它位于发出命令的路径或目录中。它位于此目录中的原因是，此示例中的命令不会显式指定备份映象的目标位置。例如，在 Windows® 操作系统上，此命令（在从根目录发出此命令时）会创建一个映象，且它会出现在目录列表中，如下所示:

```
 Directory of D:\SAMPLE.0\DB2\NODE0000\CATN0000\20010320

03/20/2001  12:26p      <DIR>          .
03/20/2001  12:26p      <DIR>          ..
03/20/2001  12:27p          12,615,680 122644.001
```

注: 如果 DB2 客户机和服务器不在同一系统上，则备份映象的缺省目标目录是发出命令的客户机系统的当前工作目录。此目标目录或设备必须存在于服务器系统上。

备份映象创建在您在调用备份实用程序时，用选项指定的目标位置中。位置可以是:

- 目录（对于备份到磁盘或软盘）
- 设备（对于备份到磁带）
- Tivoli® Storage Manager（TSM）服务器
- 另一供应商服务器

每当调用数据库备份操作时，都会使用总结信息自动更新恢复历史文件。此文件在数据库配置文件所在的目录中创建。

在基于 UNIX® 的系统上，在磁盘上创建的备份映象的文件名由几个元素并置组成，由句点分隔:

```
DB_alias.Type.Inst_name.NODEnnnn.CATNnnnn.timestamp.Seq_num
```

例如:

```
STAFF.0.DB201.NODE0000.CATN0000.19950922120112.001
```

在 Windows 操作系统上，使用 4 层子目录树:

```
DB_alias.Type\Inst_name\NODEnnnn\CATNnnnn\yyyymmdd\hhmmss.Seq_num
```

例如:

```
SAMPLE.0\DB2\NODE0000\CATN0000\20010320\122644.001
```

数据库别名
在调用备份实用程序时指定的，由 1 至 8 个字符组成的数据库别名。

类型
备份操作的类型，其中: 0 表示完整的数据库级备份、3 表示表空间级的备份而 4 表示由 LOAD...COPY TO 命令生成的备份映象。

实例名
从 **DB2INSTANCE** 环境变量中提取的，由 1 至 8 个字符组成的当前实例名。

节点号
节点号。在非分区数据库系统上，始终为 NODE0000。在分区数据库系统上，它是 NODE*xxxx*，其中 *xxxx* 是分配给 db2nodes.cfg 文件中的节点的号码。

目录节点号
数据库的目录节点的节点号。在非分区数据库系统中，始终为 CATN0000。在分区数据库系统中，它是 CATN*xxxx*，其中 *xxxx* 是分配给 db2nodes.cfg 文件中的节点的号码。

时间戳记

执行备份操作时的日期与时间的 14 个字符表示法。该时间戳记的格式为 *yyyymmddhhnnss*，其中:

- *yyyy* 表示年度（1995 至 9999）
- *mm* 表示月份（01 到 12）
- *dd* 表示某月中的某一天（01 到 31）
- *hh* 表示小时（00 到 23）
- *nn* 表示时间（00 到 59）
- *ss* 表示秒（00 到 59）

序列号

用作文件扩展名的一个 3 位的数字。

将备份映象写到磁带时:

- 不创建文件名，但以上描述的信息会存储在备份头中以在验证时使用。
- 必须有一个磁带机可通过标准操作系统接口来使用。在大分区数据库系统上，如果每个数据库分区服务器都有一个专用的磁带机可能不太实际。可以将这些磁带机与一个或多个 TSM 服务器连接，以便将对这些磁带机的访问权提供给每个数据库分区服务器。
- 在分区数据库系统上，还可以使用提供虚拟磁带机功能的产品，例如 REELlibrarian 4.2 或 CLIO/S。可使用这些产品来存取通过伪磁带机连接至其它节点（数据库分区服务器）的磁带机。对远程磁带机的存取权是透明地提供的，而伪磁带机可以通过标准的操作系统接口来访问。

不能备份处于不可用状态的数据库，除非该数据库处于备份暂挂状态。如果有任何表空间处于异常状态，则不能备份该数据库或该表空间，除非它处于备份暂挂状态。

如果数据库或表空间由于在执行复原操作期间发生了系统崩溃而处于部分复原状态，必须成功地复原该数据库或表空间才能对它们进行备份。

如果要备份的表空间的列表包含了临时表空间的名称，备份操作会失败。

备份实用程序可对建立不同数据库的备份副本的多个进程提供并行控制。此并行控制可使备份目标设备保持打开，直到所有备份操作结束。如果在备份操作期间发生错误，有一个打开的容器不能关闭，则其它备份操作的相同目标驱动器可能会接收到访问错误。要校正这类访问错误，必须终止导致了该错误的备份操作，并与目标设备断开连接。如果正在对并行的备份到磁带的操作使用备份实用程序，应确保这些进程的目标驱动器不是同一磁带。

显示备份信息

可通过 **db2ckbkp** 显示关于现有备份映象的信息。此实用程序允许您:

- 测试备份映象的完整性并确定是否可复原它。
- 显示存储在备份头中的信息。
- 显示有关备份映象中对象和日志文件头的信息。

相关概念:

- 第 47 页的『了解恢复历史文件』

相关参考:

- 第 189 页的『db2ckbkp — 检查备份』
- 第 293 页的附录 F, 『Tivoli Storage Manager』

使用备份所需的特权、权限和授权

特权使用户能够创建或访问数据库资源。权限级别提供将特权、高级数据库管理器维护和实用程序操作分组的方法。这两者一起用于控制对数据库管理器和它的数据库对象的访问。用户只能访问那些他们具有适当授权(即必需的特权或权限)的对象。

必须具有 SYSADM、SYSCTRL 或 SYSMAINT 权限才能使用备份实用程序。

使用备份

先决条件:

不应连接到将要备份的数据库;备份实用程序自动建立与指定数据库的连接,而此连接会在备份操作完成时终止。

数据库可以是本地的或远程的。备份映象保留在数据库服务器上,除非您使用的是存储管理产品,如 Tivoli Storage Manager (TSM)。

在分区数据库系统上,将分别备份数据库分区。此操作对于您调用实用程序的数据库分区服务器是本地的。但您可以从实例中的一个数据库分区服务器发出 **db2_all**,以对由节点号标识的服务器列表调用备份实用程序。(使用 LIST NODES 命令来标识具有用户表的节点或数据库分区服务器。)为此,必须先备份目录节点,然后,备份其它数据库分区。也可以使用“命令中心”来备份数据库分区。因为此方法不支持前滚恢复,应定期备份这些节点上的数据库。还应与您建立的任何备份副本一起,保留一份 `db2nodes.cfg` 文件,以保护可能对此文件造成的损坏。

在分布式请求系统上，备份操作适用于分布式请求数据库和存储在数据库目录（包装器、服务器和别名等等）中的元数据。不备份数据源对象（表和视图），除非它们也存储在分布式请求数据库中。

若一个数据库是使用数据库管理器的先前发行版创建的且该数据库还未迁移，则必须在将该数据库迁移之后，才能备份它。

限制:

以下限制适用于备份实用程序:

- 表空间备份操作和表空间复原操作不能同时运行，即使涉及的是不同的表空间。
- 如果想要能够在分区数据库环境中执行前滚恢复，则必须在节点列表上定期备份数据库且必须具有系统中余下节点的至少一个备份映象（即使不包含该数据库的用户数据）。下列两种情况都需要在不包含数据库的用户数据的数据库分区服务器中存在数据库分区的备份映象:
  - 在建立上一个备份之后已将一个数据库分区服务器添加到数据库系统，因此需要在此数据库分区服务器上执行正向恢复。
  - 使用时间点恢复，它要求系统中的所有数据库分区都处于前滚暂挂状态。

过程:

备份实用程序可以通过命令行处理器（CLP）、“控制中心”中的“备份数据库”笔记本或向导或 **db2Backup** 应用程序编程接口（API）来调用。

以下是通过 CLP 发出的 BACKUP DATABASE 命令的示例:

```
db2 backup database sample to c:\DB2Backups
```

要打开“备份数据库”笔记本或向导:

1. 从“控制中心”中，展开对象树，直到找到“数据库”文件夹。
2. 单击“数据库”文件夹。任何现有的数据库都会显示在窗口右边的窗格（内容窗格）中。
3. 在内容窗格中右键单击需要的数据库，然后从弹出菜单中选择“备份数据库”或“使用向导备份数据库”。“备份数据库”笔记本或“备份数据库”向导打开。

在“控制中心”中通过联机帮助设施提供了详细信息。

相关概念:

- 『Administrative APIs in Embedded SQL or DB2 CLI Programs』（*Application Development Guide: Programming Client Applications*）

- 『引入“控制中心”的插件体系结构』（*《管理指南: 实现》*）

相关任务:

- 『迁移数据库』（*《DB2 服务器快速入门》*）

相关参考:

- 『LIST DBPARTITIONNUMS Command』（*Command Reference*）
- 第 67 页的『db2Backup — 备份数据库』

备份到磁带

备份数据库或表空间时，必须正确地设置块大小和缓冲区大小，尤其在使用变量块大小（例如在 AIX 上，如果块大小已设置为零）时。

对备份时可使用的固定块大小的数量有限制。因为 DB2® 将备份映象头写为 4 KB 的块，所以会存在此限制。DB2 支持的固定块大小只有 512、1024、2048 和 4096 字节。若使用固定块大小，则可以指定任何备份缓冲区大小。但是您可能会发现，如果固定块大小不是 DB2 所支持的大小之一，则备份操作将不能成功完成。

如果数据库很大，使用固定块大小意味着您的备份操作将花费很长时间。您可能要考虑使用变量块大小。

注: 使用变量块大小当前不受支持。如果必须使用此选项，应确保您有经过良好测试的过程，以使您（使用按变量块大小创建的备份映象）进行成功的恢复。

使用变量块大小时，必须指定备份缓冲区大小小于或等于正在使用的磁带机的最大限制。要获得优化的性能，缓冲区大小必须等于正在使用的设备的最大块大小限制。

必须发出以下命令，才能在 Windows® 操作系统上使用磁带机:

```
db2 initialize tape on <device> using <blksize>
```

其中:

<device>

是有效的磁带机名。在 Windows 操作系统上，缺省值为 `\\.\TAPE0`。

<blksize>

是磁带的分块因子。它必须是 4096 的因子或倍数。缺省值是该设备的缺省块大小。

使用变量块大小从备份映象进行复原时可能会返回错误。若发生这种情况，可能需要使用适当的块大小重写该映象。以下是 AIX 上的一个示例:

```
tcl -b 0 -Bn -f /dev/rmt0 read > backup_filename.file
dd if=backup_filename.file of=/dev/rmt0/ obs=4096 conv=sync
```

备份映象转储到名为 backup_filenam.file 的文件中。**dd** 命令使用块大小 4096, 将映象转储到磁带上。

如果映象太大，不能转储到文件中，此方法可能会出现问题。有一种可能的解决方案是: 使用 **dd** 命令将该映象从一个磁带机转储至另一磁带机。只要映象在一卷磁带上放得下，就可以使用这种方法。使用两个磁带机时，**dd** 命令是:

```
dd if=/dev/rmt1 of=/dev/rmt0 obs=4096
```

若不可能使用两台磁带机，可以使用 **dd** 命令将映象转储至原始设备，然后再将该映象从原始设备转储至磁带。使用此方法的问题在于 **dd** 命令*必须*跟踪转储至原始设备的块数。此块数必须在将映象移动回磁带时指定。如果使用 **dd** 命令将映象从原始设备转储至磁带，该命令会将原始设备的整个内容转储至磁带。**dd** 实用程序不能确定使用了多少原始设备空间来保留映象。

当使用备份实用程序时，您需要知道磁带机的最大块大小限制。以下是一些示例:

设备	连接	块大小限制	DB2 缓冲区大小限制（以 4 KB 页计）
8 毫米	scsi	131,072	32
3420	s370	65,536	16
3480	s370	65,536	16
3490	s370	65,536	16
3490E	s370	65,536	16
7332（4 毫米）[1]	scsi	262,144	64
3490e	scsi	262,144	64
3590[2]	scsi	2,097,152	512
3570 (magstar MP)		262,144	64

注:

1. 7332 没有块大小的限制。256 KB 只是一个建议的值。块大小限制由父适配器确定。

2. 在 3590 支持 2 MB 的块大小时，如果性能完全可以满足您的需要，就可以尝试使用较低的值（例如 256 KB）。
3. 关于设备限制的信息，请查看设备文档或向设备供应商咨询。

备份到命名管道

现在已支持将数据库备份到基于 UNIX 的系统上的本地命名管道，或从基于 UNIX 的系统上的本地命名管道复原数据库。

先决条件:

命名管道的编写器与阅读器必须是同一台机器。管道必须存在并位于本地文件系统上。因为将命名管道视为本地设备，所以不需要指定目标是命名管道。

过程:

以下是 AIX 上的一个示例:

1. 创建命名管道:

   ```
   mkfifo /u/dmcinnis/mypipe
   ```

2. 使用此管道作为数据库备份操作的目标:

   ```
   db2 backup db sample to /u/dmcinnis/mypipe
   ```

3. 如果打算由复原实用程序来使用备份映象，则复原操作必须在备份操作之*前*调用，才不会丢失任何数据:

   ```
   db2 restore db sample into mynewdb from /u/dmcinnis/mypipe
   ```

相关任务:

- 第 58 页的『使用备份』

相关参考:

- 第 63 页的『BACKUP DATABASE』
- 第 84 页的『RESTORE DATABASE』

BACKUP DATABASE

创建数据库或表空间的备份副本。

作用域:

此命令只影响对其执行该命令的数据库分区。

授权:

以下其中一项:

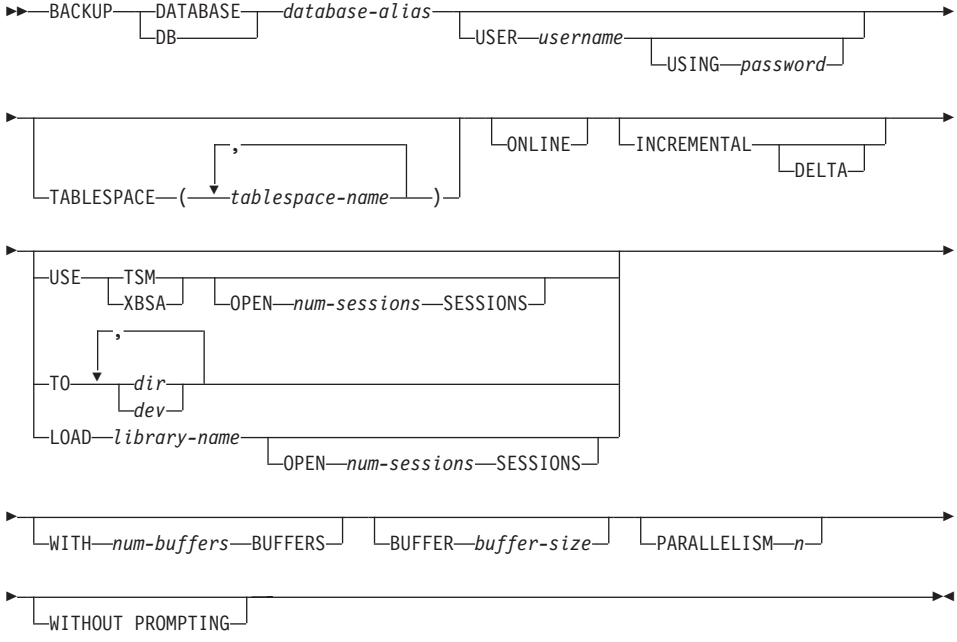
- *sysadm*
- *sysctrl*
- *sysmaint*

必需的连接:

数据库。此命令自动建立与指定数据库的连接。

注: 如果与指定数据库的连接已存在，则将终止该连接，并且专门为备份操作建立新的连接。连接在备份操作完成时终止。

命令语法:



命令参数:

DATABASE database-alias
指定要备份的数据库的别名。

USER username
标识备份数据库时要使用的用户名。

USING password
用于认证用户名的密码。如果省略密码，会提示用户输入。

TABLESPACE tablespace-name
用于指定要备份的表空间的名称列表。

ONLINE
指定联机备份。缺省值是脱机备份。联机备份只对配置为启用了 *logretain* 或 *userexit* 的数据库可用。

注: 因为 DB2 备份实用程序对包含 LOB 的对象需要 S 锁，如果 `sysibm.systables` 上有一个 IX 锁，则联机备份操作可能会超时。

INCREMENTAL
指定累积（增量）备份映象。增量备份映象是自从最新的成功完整备份操作以来，进行过更改的所有数据库数据的一份副本。

DELTA
指定非累积（delta）备份映象。delta 备份映象是自从任何类型的最新成功备份操作以来，更改过的所有数据库数据的一份副本。

USE TSM
指定备份将使用 Tivoli Storage Manager（以前称为 ADSM）输出。

OPEN num-sessions SESSIONS
要在 DB2 和 TSM 或另一备份供应商产品之间创建的 I/O 会话数。

注: 该参数在备份磁带、磁盘或其它本地设备时无效。

USE XBSA
指定将使用的 XBSA 接口。“备份服务 API” (XBSA) 是一个开放的应用程序编程接口，由需要数据存储管理的应用程序或设施进行备份或归档时使用。Legato NetWorker 是当前支持 XBSA 接口的一种存储管理器。

TO dir/dev
目录或磁带机名称列表。必须指定目录所驻留的完整路径。如果省略 USE TSM、TO 和 LOAD，则备份映象的缺省目标目录是客户机的当前工作目录。此目标目录或设备必须存在于数据库服务器上。可重复此参数以指定备份映象将跨越的目标目录和设备。如果指定了多个目标（例如，target1、target2 和 target3），target1 将首先打开。媒体头和特殊文件（包

括配置文件、表空间和历史文件）放在 target1 中。所有其它目标都是打开的，并在备份操作期间并行使用。由于对 Windows 操作系统没有通用磁带支持，所以每种类型的磁带机都需要一个唯一的设备驱动程序。要备份至 Windows 操作系统上的 FAT 文件系统，用户必须遵循 8.3 命名限制。

使用磁带机或软盘可能会生成需要用户进行输入的消息和提示。有效的响应选项是:

c 继续 — 继续使用生成了警告消息的设备（例如，已安装了新磁带时）

d 设备终止 — 只停止使用那个生成了警告消息（例如，没有磁带了）的设备

t 终止 — 异常终止备份操作。

如果磁带系统不支持唯一地引用一个备份映象，建议不要将同一数据库的多个备份件副本保留在同一磁带上。

LOAD library-name

共享库（在 Windows 操作系统上为 DLL）的名称，它包含要使用的供应商备份与复原 I/O 函数。也可以包含完整路径。如果未提供完整路径，将缺省为用户出口程序所驻留的那个路径。

WITH num-buffers BUFFERS

将要使用的缓冲区数量。缺省为 2。但是在将一个备份创建到多个位置时，具有较多的缓冲区有助于改进性能。

BUFFER buffer-size

以 4 KB 页计的缓冲区大小，在构建备份映象时使用。此参数的最小值是 8 页；缺省值是 1024 页。如果指定了缓冲区大小为零，则 数据库管理器配置参数 *backbufsz* 的值将用作缓冲区分配大小。

如果使用具有可变块大小的磁带，则将缓冲区大小减少至磁带机支持的范围。否则，备份操作虽可以成功，但生成的映象可能是不可恢复的。

在 SCO UnixWare 7 上使用磁带机时，指定缓冲区大小 16。

在 Linux 的大多数版本中，如果对向 SCSI 磁带机进行的备份操作使用 DB2 缺省缓冲区大小，会导致错误 SQL2025N，原因代码 75。要防止 Linux 内部 SCSI 缓冲区溢出，使用此公式:

```
bufferpages <= ST_MAX_BUFFERS * ST_BUFFER_BLOCKS / 4
```

其中 *bufferpages* 是 *backbufsz* 或 *restbufsz* 的值，而 `ST_MAX_BUFFERS` 和 `ST_BUFFER_BLOCKS` 在 `drivers/scsi` 目录下的 Linux 内核中定义。

PARALLELISM n

确定可由备份实用程序并行读取的表空间的数量。缺省值为 1。

WITHOUT PROMPTING

指定备份将以无人照管方式运行，而通常需要用户干涉的任何操作都将返回一个错误消息。

示例:

在以下示例中，数据库 WSDB 是在编号为 0 至 3 的这 4 个分区上定义的。可以从所有分区存取路径 /dev3/backup。分区 0 是目录分区，需要单独备份，因为这是脱机备份。要执行所有 WSDB 数据库分区的脱机备份并保存至 /dev3/backup，从其中一个数据库分区发出下列命令:

```
db2_all '<<+0< db2 BACKUP DATABASE wsdb TO /dev3/backup'
db2_all '|<<-0< db2 BACKUP DATABASE wsdb TO /dev3/backup'
```

在第二条命令中，b2_all 实用程序将依次对每个分区（分区 0 除外）发出同一备份命令。所有四个数据库分区备份映象都将存储在 /dev3/backup 目录中。

在以下示例中，数据库 SAMPLE 将备份至使用两个并行 TSM 客户机会话的 TSM 服务器。备份实用程序将使用四个缓冲区，缓冲区的大小为缺省缓冲区大小（1024 x 4K 页）。

```
db2 backup database sample use tsm open 2 sessions with 4 buffers
```

在下一示例中，在表空间级将数据库 payroll 的表空间（syscatspace 和 userspace1）备份至磁带。

```
db2 backup database payroll tablespace (syscatspace, userspace1) to
   /dev/rmt0, /dev/rmt1 with 8 buffers without prompting
```

以下是对可恢复数据库每周进行的增量备份策略的样本。包括一个每周进行的完整数据库备份操作、一个每天进行的非累积（delta）备份操作以及一个在每周中期进行的累积（增量）备份操作:

```
(Sun) db2 backup db sample use tsm
(Mon) db2 backup db sample online incremental delta use tsm
(Tue) db2 backup db sample online incremental delta use tsm
(Wed) db2 backup db sample online incremental use tsm
(Thu) db2 backup db sample online incremental delta use tsm
(Fri) db2 backup db sample online incremental delta use tsm
(Sat) db2 backup db sample online incremental use tsm
```

相关参考:

- 第 84 页的『RESTORE DATABASE』
- 第 117 页的『ROLLFORWARD DATABASE』

db2Backup — 备份数据库

创建数据库或表空间的备份副本。

作用域:

此 API 只影响对其执行该 API 的数据库分区。

授权:

以下其中一项:

- *sysadm*
- *sysctrl*
- *sysmaint*

必需的连接:

数据库。此 API 自动建立与指定数据库的连接。

备份完成时，连接将终止。

API 包含文件:

db2ApiDf.h

C API 语法:

```
/* File: db2ApiDf.h */
/* API: db2Backup */
/* ... */
SQL_API_RC SQL_API_FN
db2Backup (
  db2Uint32 versionNumber,
  void         *pDB2BackupStruct,
      struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2BackupStruct
{
  char                         *piDBAlias;
  char                         oApplicationId[SQLU_APPLID_LEN+1];
  char                         oTimestamp[SQLU_TIME_STAMP_LEN+1];
  struct db2TablespaceStruct   *piTablespaceList;
  struct db2MediaListStruct    *piMediaList;
  char                         *piUsername;
  char                         *piPassword;
  void                         *piVendorOptions;
  db2Uint32                    iVendorOptionsSize;
  db2Uint32                    oBackupSize;
  db2Uint32                    iCallerAction;
  db2Uint32                    iBufferSize;
  db2Uint32                    iNumBuffers;
  db2Uint32                    iParallelism;
   db2Uint32 iOptions;
} db2BackupStruct;

typedef SQL_STRUCTURE db2TablespaceStruct
{
  char                         **tablespaces;
  db2Uint32                    numTablespaces;
} db2TablespaceStruct;

typedef SQL_STRUCTURE db2MediaListStruct
{
  char                         **locations;
  db2Uint32                    numLocations;
  char                         locationType;
} db2MediaListStruct;
/* ... */
```

一般 API 语法:

```
/* File: db2ApiDf.h */
/* API: db2Backup */
/* ... */
SQL_API_RC SQL_API_FN
db2gBackup (
  db2Uint32 versionNumber,
  void         *pDB2gBackupStruct,
      struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gBackupStruct
{
  char                         *piDBAlias;
  db2Uint32                    iDBAliasLen;
  char                         *poApplicationId;
  db2Uint32                    iApplicationIdLen;
  char                         *poTimestamp;
  db2Uint32                    iTimestampLen;
  struct db2gTablespaceStruct *piTablespaceList;
  struct db2gMediaListStruct  *piMediaList;
  char                         *piUsername;
  db2Uint32                    iUsernameLen;
  char                         *piPassword;
   db2Uint32 iPasswordLen;
  void                         *piVendorOptions;
  db2Uint32                    iVendorOptionsSize;
  db2Uint32                    oBackupSize;
  db2Uint32                    iCallerAction;
  db2Uint32                    iBufferSize;
  db2Uint32                    iNumBuffers;
  db2Uint32                    iParallelism;
   db2Uint32 iOptions;
} db2gBackupStruct;

typedef SQL_STRUCTURE db2gTablespaceStruct
{
  struct db2Char               *tablespaces;
  db2Uint32                    numTablespaces;
} db2gTablespaceStruct;

typedef SQL_STRUCTURE db2gMediaListStruct
{
  struct db2Char               *locations;
  db2Uint32                    numLocations;
  char                         locationType;
} db2gMediaListStruct;

typedef SQL_STRUCTURE db2Char
{
   char            *pioData;
   db2Uint32       iLength;
   db2Uint32       oLength;
} db2Char;
/* ... */
```

API 参数:

versionNumber

输入。指定作为第二个参数 *pDB2BackupStruct* 传送的结构的版本和发行版级别。

pDB2BackupStruct

输入。指向 *db2BackupStruct* 结构的指针。

pSqlca

输出。指向 *sqlca* 结构的指针。

piDBAlias

输入。包含要备份的数据库的数据库别名（它是在系统数据库目录中编目的）的字符串。

iDBAliasLen

输入。一个 4 字节的无符号整数，代表以字节计的数据库别名的长度。

oApplicationId

输出。此 API 将返回一个标识为应用程序提供服务的代理进程的字符串。可以用来通过使用数据库监视器获取有关备份操作的进度的信息。

poApplicationId

输出。提供缓冲区长度 SQLU_APPLID_LEN+1（在 `sqlutil` 中定义）。此 API 将返回一个标识为应用程序提供服务的代理进程的字符串。可以用来通过使用数据库监视器获取有关备份操作的进度的信息。

iApplicationIdLen

输入。一个 4 字节的无符号整数，表示 *poApplicationId* 缓冲区的长度（以字节计）。应该等于 SQLU_APPLID_LEN+1（在 `sqlutil` 中定义）。

oTimestamp

输出。该 API 将返回备份映象的时间戳记。

poTimestamp

输出。提供缓冲区长度 SQLU_TIME_STAMP_LEN+1（在 `sqlutil` 中定义）。该 API 将返回备份映象的时间戳记。

iTimestampLen

输入。一个 4 字节的无符号整数，表示 *poTimestamp* 缓冲区的长度（以字节计）。应该等于 SQLU_TIME_STAMP_LEN+1（在 `sqlutil` 中定义）。

piTablespaceList

输入。列出要备份的表空间。只有表空间级备份操作需要该参数。对于数据库级的备份，它必须为 NULL。参见结构 DB2TablespaceStruct。

piMediaList

输入。此结构允许调用程序指定备份操作的目标。所提供的信息取决于 locationType 字段的值。locationType 的有效值（在 sqlutil.h 中定义）为:

SQLU_LOCAL_MEDIA

本地设备（磁带、磁盘或软盘的组合）。

SQLU_TSM_MEDIA

TSM。如果位置指针设置为 NULL，则使用 DB2 附带提供的 TSM 共享库。如果希望使用 TSM 共享库的另一版本，使用 SQLU_OTHER_MEDIA 并提供共享库名。

SQLU_OTHER_MEDIA

供应商产品。提供位置字段中的共享库名。

SQLU_USER_EXIT

用户出口。不需要任何附加输入（仅当服务器在 OS/2 上时才可用）。

有关更多信息，参见结构 DB2MediaListStruct。

piUsername

输入。包含尝试连接时将使用的用户名的字符串。可以为 NULL。

iUsernameLen

输入。一个 4 字节的无符号整数，表示用户名的长度（以字节计）。如果不提供用户名，则设置为零。

piPassword

输入。包含将与用户名一起使用的密码的字符串。可以为 NULL。

iPasswordLen

输入。一个 4 字节的无符号整数，代表以字节计的密码的长度。如果不提供密码，则设置为零。

piVendorOptions

输入。用于将信息从应用程序发送给供应商函数。此数据结构必须是平面的，因此不支持间接级别。应注意字节转换未完成，且未检查此数据的代码页。

iVendorOptionsSize

输入。*piVendorOptions* 字段的长度，不能超过 65535 字节。

oBackupSize

输出。备份映象的大小（以 MB 计）。

iCallerAction

输入。指定要采取的操作。有效值（在 db2ApiDf.h 中定义）为:

DB2BACKUP_BACKUP

启动备份。

DB2BACKUP_NOINTERRUPT

启动备份。指定备份将以无人照管方式运行，将尝试那些通常需要用户干预的方案而不首先返回到调用程序，或者将生成错误。例如，如果已知备份所需的所有媒体都已安装，且不希望出现实用程序提示，则使用此调用程序操作。

DB2BACKUP_CONTINUE

在用户已执行实用程序请求的某些操作后继续备份操作（例如，安装新磁带）。

DB2BACKUP_TERMINATE

在用户执行实用程序请求的某些操作失败后终止备份。

DB2BACKUP_DEVICE_TERMINATE

从备份使用的设备的列表中除去特定设备。当特定媒体已满时，备份将对调用程序返回警告（在使用余下设备继续处理时）。使用此调用程序操作再次调用备份，以从正在使用的设备的列表中除去生成警告的设备。

DB2BACKUP_PARM_CHK

用来验证参数而不执行备份。此选项在返回调用后不终止数据库连接。在成功返回此调用后，期望用户发出指定 SQLUB_CONTINUE 的调用以继续执行操作。

DB2BACKUP_PARM_CHK_ONLY

用来验证参数而不执行备份。在此调用返回前，将终止由此调用建立的数据库连接，且不需要后续调用。

iBufferSize

输入。以 4KB 分配单位（页）计的备份缓冲区大小。最小值是 8 个单元。缺省值是 1024 个单元。

iNumBuffers

输入。指定要使用的备份缓冲区的数目。最小值是 2。最大值受内存限制。可以将缺省值 2 指定为 0。

iParallelism

输入。并行性的程度（缓冲区操纵器的数量）。最小值为 1。最大值为 1024。缺省值为 1。

iOptions

输入。备份特性的位图。要通过使用一位宽度 OR 运算符来生成 *iOptions* 的值以将选项组合起来。有效值（在 db2ApiDf.h 中定义）为:

DB2BACKUP_OFFLINE

脱机提供了与数据库的独占式连接。

DB2BACKUP_ONLINE

联机允许其它应用程序在执行备份操作时存取数据库。

注: 如果用户持有对 SMS LOB 数据的锁定，则联机备份操作可能表现为挂起。

DB2BACKUP_DB

完全数据库备份。

DB2BACKUP_TABLESPACE

表空间级备份。对于表空间级备份，在 *piTablespaceList* 参数中提供表空间的列表。

DB2BACKUP_INCREMENTAL

指定累积（增量）备份映象。增量备份映象是自从最新的成功完整备份操作以来，进行过更改的所有数据库数据的一份副本。

DB2BACKUP_DELTA

指定非累积（delta）备份映象。delta 备份映象是自从任何类型的最新成功备份操作以来，更改过的所有数据库数据的一份副本。

tablespaces

指向要备份的表空间的列表的指针。对于 C，该列表是以空字符结尾的字符串。在一般情况下，它是 *db2Char* 结构的列表。

numTablespaces

tablespaces 参数中的条目数。

locations

指向媒体位置的列表的指针。对于 C，该列表是以空字符结尾的字符串。在一般情况下，它是 db2Char 结构的列表。

numLocations

locations 参数中的条目数。

locationType

指示媒体类型的字符。有效值（在 sqlutil.h 中定义）为:

SQLU_LOCAL_MEDIA

本地设备（磁带、磁盘、软盘或命名管道）。

SQLU_TSM_MEDIA
Tivoli Storage Manager。

SQLU_OTHER_MEDIA
供应商库。

SQLU_USER_EXIT
用户出口（仅当服务器在 OS/2 上时才可用）。

pioData
指向字符数据缓冲区的指针。

iLength
输入。*pioData* 缓冲区的大小。

oLength
输出。保留以备将来使用。

相关样本:

- 『dbrecov.sqc -- How to recover a database (C)』
- 『dbrecov.sqC -- How to recover a database (C++)』

备份会话 — CLP 示例

示例 1

在以下示例中，数据库 SAMPLE 被备份至使用 2 个并行 TSM 客户机会话的 TSM 服务器上。备份实用程序将使用 4 个缓冲区，它们的大小是缺省缓冲区大小（1024 x 4K 页）。

```
db2 backup database sample use tsm open 2 sessions with 4 buffers

db2 backup database payroll tablespace (syscatspace, userspace1) to
   /dev/rmt0, /dev/rmt1 with 8 buffers without prompting
```

示例 2

以下是对可恢复数据库每周进行的增量备份策略的样本。它包括一个每周进行的完整数据库备份操作、一个每天进行的非累积（delta）备份操作以及一个在每周中期进行的累积（增量）备份操作:

```
(Sun) db2 backup db kdr use tsm
(Mon) db2 backup db kdr online incremental delta use tsm
(Tue) db2 backup db kdr online incremental delta use tsm
(Wed) db2 backup db kdr online incremental use tsm
(Thu) db2 backup db kdr online incremental delta use tsm
(Fri) db2 backup db kdr online incremental delta use tsm
(Sat) db2 backup db kdr online incremental use tsm
```

示例 3

要在 Windows 环境中对磁带机启动备份操作，发出:

```
db2 backup database sample to \\.\tape0
```

相关任务:

- 第 58 页的『使用备份』

优化备份性能

要减少完成备份操作所需的时间量:

- 指定表空间备份。

  对 BACKUP DATABASE 命令使用选项 TABLESPACE，可以备份（继而恢复）部分数据库。这样便于对表数据、索引和单独表空间中的长字段或大对象（LOB）数据进行管理。

- 增大 BACKUP DATABASE 命令上 PARALLELISM 参数的值，以使它反映正在备份的表空间数。

  PARALLELISM 参数定义在从数据库读取数据时已启动的进程或线程数。每个进程或线程都指定给一个特定的表空间。备份完此表空间后，它会请求另一个表空间。但是应注意: 每个进程或线程都需要内存和 CPU 开销，对于负荷较重的系统，应该使 PARALLELISM 参数保持为缺省值 1。

- 增加备份缓冲区大小。

  理想的备份缓冲区大小是表空间范围大小的倍数。若有多个具有不同范围大小的表空间，则指定是最大范围大小的倍数的一个值。

- 增加缓冲区的数量。

  若使用多个缓冲区和 I/O 通道，应该使用的通道数至少是缓冲区数的两倍，以确保通道不必等待数据。

- 使用多个目标设备。

相关概念:

- 第 55 页的『备份概述』

相关任务:

- 第 58 页的『使用备份』

第 3 章 数据库复原

这部分描述了 DB2 UDB 复原实用程序，它用于重新构建先前进行了备份的已受损的或毁坏的数据库或表空间。

包括下列主题:

复原概述

DB2® RESTORE DATABASE 命令的最简单的格式只需要您指定想要复原的数据库的别名。例如:

```
db2 restore db sample
```

在此示例中，因为 SAMPLE 数据库存在，将返回以下消息:

```
SQL253 W 警告！正在复原到一个与备份映象数据库相同的现有数据库。
将删除数据库文件。您想继续吗？（y/n）
```

如果指定 y，而 SAMPLE 数据库的备份映象已存在，则复原操作应成功完成。

数据库复原操作需要一个独占连接: 即，启动该任务后，复原实用程序会防止其它应用程序访问数据库，直到复原操作成功完成，所以不能再对该数据库运行任何应用程序。但表空间复原操作可以联机完成。

表空间将不可用直到复原操作（后跟前滚恢复）成功完成。

如果有跨越多个表空间的表，则应该一起备份并复原这个表空间集合。

执行部分或子集复原操作时，可以使用表空间级的备份映象或完整数据库级的备份映象，并从该映象中选择一个或多个表空间。从建立备份映象时开始的所有与这些表空间相关的日志文件必须存在。

优化复原性能

要缩短完成复原操作所需的时间:

- 增加复原缓冲区大小。

  复原缓冲区大小必须是在备份操作期间指定的备份缓冲区大小的正整数倍数。如果指定了不正确的缓冲区，则分配的缓冲区将是最小的可接受大小。

- 增加缓冲区的数量。

  指定的值必须是为备份缓冲区指定的页数的倍数。最小页数为 16。

- 增加 PARALLELISM 选项的值。

  这将增加将用来在复原操作期间写至数据库的缓冲区处理器（BM）的数目。缺省值为 1。

使用复原所需的特权、权限和授权

特权使用户能够创建或访问数据库资源。权限级别提供将特权、高级数据库管理器维护和实用程序操作分组的方法。这两者一起用于控制对数据库管理器和它的数据库对象的访问。用户只能访问那些他们具有适当授权（即必需的特权或权限）的对象。

要从完整的数据库备份复原至*现有*数据库，必须具有 SYSADM、SYSCTRL 或 SYSMAINT 权限。要复原至*新*数据库，必须具有 SYSADM 或 SYSCTRL 权限。

使用复原

先决条件:

复原至*现有*数据库时，不应连接至要复原的数据库：复原实用程序自动建立与特定数据库的连接，并且此连接在复原操作完成时终止。复原到*新*数据库时，需要实例连接才能创建数据库。复原到*新远程*数据库时，必须首先连接至新数据库将驻留的实例。然后创建新数据库，指定服务器的代码页和地区。

数据库可以是本地的或远程的。

限制:

复原实用程序有以下局限性:

- 仅当先前已使用 DB2 备份实用程序备份了数据库时，才能使用复原实用程序。
- 数据库复原操作在正在运行前滚进程时不能启动。
- 仅当表空间当前已存在且是同一表空间时才能复原表空间；“同一表空间”表示该表空间未删除，然后在备份和复原操作之间重新创建。
- 不能将表空间级的备份复原到新数据库。
- 不能执行涉及系统目录表的联机表空间级复原操作。

过程:

复原实用程序可以通过命令行处理器（CLP）、“控制中心”的“复原数据库”笔记本或向导或者 **db2Restore** 应用程序编程接口（API）来调用。

以下是通过 CLP 发出的 RESTORE DATABASE 命令的示例:

```
db2 restore db sample from D:\DB2Backups taken at 20010320122644
```

要打开“复原数据库”笔记本或向导:

1. 从“控制中心”中，展开对象树，直到找到“数据库”文件夹。
2. 单击“数据库”文件夹。所有现有的数据库都显示在该窗口右边的窗格（内容窗格）中。
3. 在内容窗格中，用鼠标右键单击需要的数据库，然后从弹出菜单中选择“复原数据库”或“使用向导复原数据库”。“复原数据库”笔记本或“复原数据库”向导打开。

在“控制中心”中通过联机帮助设施提供了详细信息。

相关概念:

- 『Administrative APIs in Embedded SQL or DB2 CLI Programs』（*Application Development Guide: Programming Client Applications*）
- 『引入“控制中心”的插件体系结构』（*《管理指南: 实现》*）

相关参考:

- 第 91 页的『db2Restore — 复原数据库』

在测试和生产环境中使用增量复原

一旦对生产数据库启用增量备份与恢复，就可以使用增量或 delta 备份映象来创建或刷新测试数据库。可以通过使用手工或自动增量复原来完成此任务。要将备份映象从生产数据库复原至测试数据库，在 RESTORE DATABASE 命令上使用 INTO *target-database-alias* 选项。例如，在具有下列备份映象的生产数据库中:

```
backup db prod
备份成功。此备份的时间戳记是: <ts1>
backup db prod incremental
备份成功。此备份映象的时间戳记是: <ts2>
```

手工增量复原的一个示例将是:

```
restore db prod incremental taken at ts2 into test without prompting
DB20000I  RESTORE DATABASE 命令成功完成。

restore db prod incremental taken at ts1 into test without prompting
DB20000I  RESTORE DATABASE 命令成功完成。

restore db prod incremental taken at ts2 into test without prompting
DB20000I  RESTORE DATABASE 命令成功完成。
```

如果数据库 TEST 已存在，则复原操作将覆盖已在其中的任何数据。如果数据库 TEST 不存在，则复原实用程序将创建它，并使用备份映象的数据来填充它。

因为自动增量复原操作依赖于数据库历史，所以根据测试数据库是否存在，复原步骤将稍微有所变化。要对数据库 TEST 执行自动增量复原，它的历史必须包含数据库 PROD 的备份映象历史。该备份映象的数据库历史将替换对于数据库 TEST 已存在的所有数据库历史，条件是:

- 当发出 RESTORE DATABASE 命令时数据库 TEST 不存在，或
- 当发出 RESTORE DATABASE 命令时，数据库 TEST 存在且数据库 TEST 历史不包含任何记录。

以下示例显示对不存在的数据库 TEST 的自动增量复原:

```
restore db prod incremental automatic taken at ts2 into test without
prompting
DB20000I  RESTORE DATABASE 命令成功完成。
```

复原实用程序将创建 TEST 数据库，然后填充它。

如果数据库 TEST 确实存在且数据库历史不为空，则必须删除该数据库，才能执行自动增量复原操作，如下所示:

```
drop db test
DB20000I  DROP DATABASE 命令成功完成。

restore db prod incremental automatic taken at ts2 into test without
prompting
DB20000I  RESTORE DATABASE 命令成功完成。
```

如果不想删除该数据库，则在发出 RESTORE DATABASE 命令之前可以通过使用指向将来的时间戳记和 WITH FORCE OPTION 参数来发出 PRUNE HISTORY 命令:

```
connect to test
Database Connection Information

Database server       = <server id>
SQL authorization ID       = <id>
Local database alias       = TEST

prune history 9999 with force option
DB20000I PRUNE 命令成功完成。
connect reset
DB20000I SQL 命令成功完成。
restore db prod incremental automatic taken at ts2 into test without
prompting
QL2540W  复原成功，但在以“无任何中断”方式处理时在“数据库复原”期间遇到了
警告“2539”。
```

这种情况下，RESTORE DATABASE COMMAND 将以与数据库 TEST 不存在时的同一方式运行。

如果数据库 TEST 确定存在且数据库历史为空，不必删除数据库 TEST 就可以执行自动增量复原操作:

```
restore db prod incremental automatic taken at ts2 into test without
prompting
QL2540W  复原成功，但在以“无任何中断”方式处理时在“数据库复原”期间遇到了
警告“2539”。
```

可以继续建立测试数据库的增量或 Delta 备份而不用先建立完整数据库备份。然而，如果需要复原其中一个增量或 delta 映象，则将必须执行手工增量复原。这是因为自动增量复原操作要求自动增量复原期间复原的每个备份映象是根据同一个数据库别名创建的。

如果在使用生产备份映象完成复原操作之后建立测试数据库的完整备份映象，则可以建立增量或 delta 备份并通过使用手工或自动方式来复原它们。

相关概念:

- 第 25 页的『增量备份与恢复』

相关参考:

- 第 63 页的『BACKUP DATABASE』
- 第 84 页的『RESTORE DATABASE』
- 第 202 页的『LIST HISTORY』

在复原操作期间重新定义表空间容器（重定向复原）

在数据库备份操作期间，保留了一个记录，它记录了与正在备份的表空间相关的所有表空间容器。在复原操作期间，会检查备份映象中列示的所有容器，以确定它们是否存在并可访问。若这些容器中的一个或多个由于媒体故障（或由于任何其它原因）而不可访问，复原操作将失败。在这种情况下，复原操作需要重定向至不同的容器。DB2® 支持添加、更改或除去表空间容器。

通过调用 RESTORE DATABASE 命令并指定 REDIRECT 参数，或通过使用“控制中心”中的“复原数据库”笔记本的“容器”页，可重新定义表空间容器。调用增量备份映象的重定向复原的过程与调用非增量备份映象的过程相似：调用带有 REDIRECT 参数的 RESTORE DATABASE 命令并指定应作为增量复原数据库的依据的备份映象。

在重定向复原操作期间，如果目录和文件容器不存在，将自动创建。数据库管理器不会自动创建设备容器。

容器重定向为管理表空间容器提供了相当大的灵活性。例如，即使不支持向 SMS 表空间添加容器，您也可以通过在调用重定向复原操作时指定其它容器来达到此目的。

相关参考:

- 第 84 页的『RESTORE DATABASE』
- 第 102 页的『复原会话 — CLP 示例』

相关样本:

- 『dbrecov.out -- HOW TO RECOVER A DATABASE (C)』
- 『dbrecov.sqc -- How to recover a database (C)』
- 『dbrecov.out -- HOW TO RECOVER A DATABASE (C++)』
- 『dbrecov.sqC -- How to recover a database (C++)』

复原至现存的数据库

可以将一个完整的数据库备份映象复原至现有的数据库。备份映象可能在别名、数据库名或数据库起始值等方面与现存的数据库有所不同。

数据库*种子值*是数据库的唯一标识符，它在该数据库的整个生命期永不更改。种子值是在创建数据库时由数据库管理器指定的。DB2® 始终使用备份映象中的种子值。

复原至现有的数据库时，复原实用程序会:

- 删除现有数据库中的表、索引和长字段数据，并用备份映象中的数据来替换它们。
- 替换表示要复原的每个表空间的表条目。
- 保留恢复历史文件，除非它已损坏或者没有任何条目。如果恢复历史文件已损坏，数据库管理器会从备份映象中复制文件。
- 保留现有数据库的认证类型。
- 保留现有数据库的数据库目录。该目录定义了数据库的驻留位置与编目方式。
- 比较数据库种子值。如果种子值不同:
  - 删除与现有的数据库相关的日志。
  - 从备份映象中复制数据库配置文件。
  - 如果对 RESTORE DATABASE 命令指定了 NEWLOGPATH，应将 NEWLOGPATH 设置为 *logpath* 数据库配置参数的值。

  如果数据库种子值相同:
  - 如果映象是不可复原的数据库的，则删除日志。
  - 保留当前的数据库配置文件，除非该文件已毁坏，在这种情况下将从备份映象中复制此文件。
  - 如果对 RESTORE DATABASE 命令指定了 NEWLOGPATH，应将 NEWLOGPATH 设置为 *logpath* 数据库配置参数的值；否则应将当前日志路径复制到数据库配置文件。验证日志路径: 如果该路径不能由数据库使用，应更改数据库配置以使用缺省日志路径。

复原至新的数据库

可以创建一个新数据库并向它复原完整的数据库备份映象。如果未创建新的数据库，则复原实用程序将创建一个数据库。

复原到新数据库时，复原实用程序:

- 使用通过目标数据库别名参数指定的数据库别名创建新数据库。(如果未指定目标数据库别名，复原实用程序会使用通过源数据库别名参数指定的相同别名来创建数据库。)
- 从备份映象复原数据库配置文件。
- 如果在 RESTORE DATABASE 命令上指定了 NEWLOGPATH，应将 NEWLOGPATH 设置为 *logpath* 数据库配置参数的值。验证日志路径: 如果路径不能由数据库使用，则应更改数据库配置以使用缺省日志路径。
- 从备份映象复原认证类型。

- 从备份映象中的数据库目录复原注释。
- 复原数据库的恢复历史文件。

RESTORE DATABASE

重新构建使用 DB2 备份实用程序作了备份的损坏或毁坏的数据库。复原的数据库与创建备份副本时它所处的状态相同。此实用程序还可以使用不同于备份映象中数据库名称的名称来复原数据库（还能复原至新数据库）。

此实用程序还可以用来复原由 DB2 的前两个版本产生的备份映象。如果需要迁移，将在复原操作结束时自动调用它。

如果在备份操作时对数据库启用了前滚恢复，可在成功完成复原操作之后调用前滚实用程序将该数据库的状态恢复为损坏或毁坏以前的状态。

此实用程序还可以从表空间级备份复原。

要复原在另一个工作站平台上备份的数据库，使用 db2move 实用程序。不能将一个平台上的数据库直接复原至另一个平台。受支持版本的 Microsoft Windows 被视作是等价的。

作用域:

此命令只影响对其执行该命令的节点。

授权:

要复原至现有数据库，授予下列其中一个权限:

- *sysadm*
- *sysctrl*
- *sysmaint*

要复原至新数据库，授予下列其中一个权限:

- *sysadm*
- *sysctrl*

必需的连接:

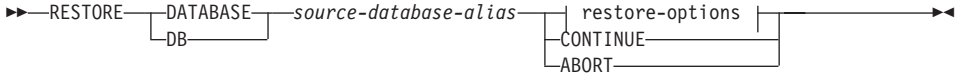
要复原至现有数据库的数据库。此命令自动建立与指定数据库的连接。

要复原至新数据库的实例和数据库。创建数据库需要实例连接。

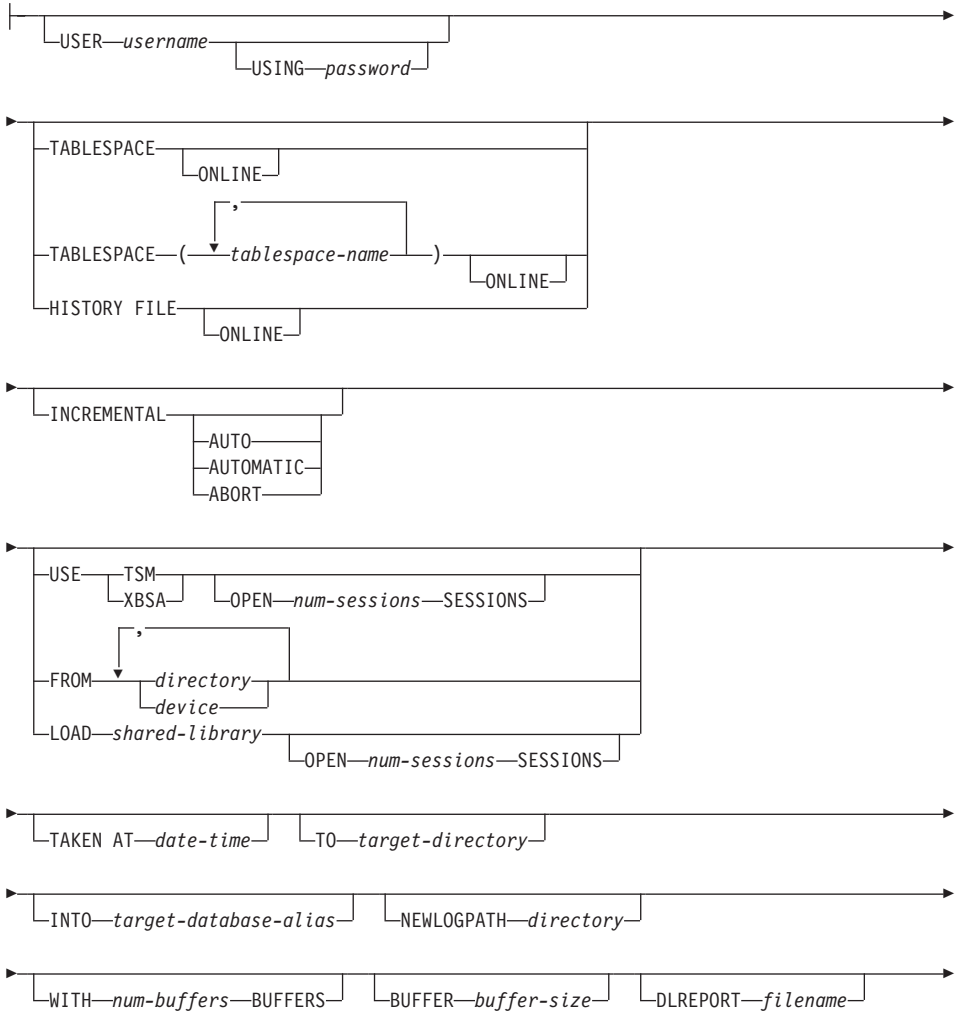
要复原至不同于当前实例的实例上的新数据库（由 **DB2INSTANCE** 环境变量的值定义），必须首先与新数据库将驻留的实例连接。

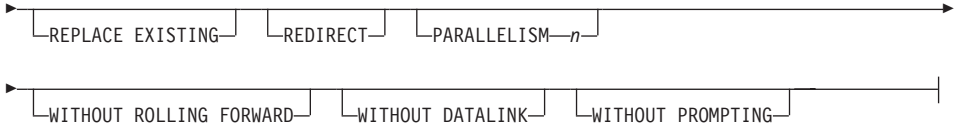
要复原至新的*远程*数据库，必须与新数据库将驻留的实例连接。

命令语法:



restore-options:





命令参数:

DATABASE source-database-alias
从其建立备份的源数据库的别名。

CONTINUE
指定已重新定义了容器，应执行重定向复原操作中的最后一步。

ABORT
此参数:

- 停止重定向复原操作。当发生需要重复一个或多个步骤的错误时此参数很有用。发出了带 ABORT 选项的 RESTORE DATABASE 后，必须重复重定向复原操作的每一步，包括带有 REDIRECT 选项的 RESTORE DATABASE。
- 在增量复原操作完成前终止它。

USER username
标识复原数据库时要使用的用户名。

USING password
用于认证用户名的密码。如果省略密码，会提示用户输入。

TABLESPACE tablespace-name
用于指定要复原的表空间的名称列表。

ONLINE
此关键字只可在执行表空间级的复原操作时应用，指定该关键字使备份映象可联机复原。这意味着在对备份映象进行复原时其它代理进程可连接至数据库，而复原指定的表空间时其它表空间中的数据将是可用的。

HISTORY FILE
指定此关键字，以从备份映象中只复原历史文件。

INCREMENTAL
不需要其它参数，INCREMENTAL 指定手工累积复原操作。在手工复原期间，用户必须对复原涉及的每个映象手工发出每个复原命令。按照以下次序完成此操作：最后一个、第一个、第二个、第三个，以此类推，并且包括最后一个映象。

INCREMENTAL AUTOMATIC/AUTO
指定自动累积复原操作。

INCREMENTAL ABORT

指定正在进行的手工累积复原操作异常中止。

USE TSM

指定将从 TSM 管理的输出复原数据库。

OPEN num-sessions SESSIONS

指定将与 TSM 或供应商产品一起使用的 I/O 会话数。

USE XBSA

指定将使用的 XBSA 接口。 “备份服务 API” (XBSA) 是一个开放的应用程序编程接口，由需要数据存储管理的应用程序或设施进行备份或归档时使用。Legato NetWorker 是当前支持 XBSA 接口的一种存储管理器。

FROM directory/device

备份映象所驻留的目录或设备。如果省略 USE TSM、FROM 和 LOAD，缺省值是当前目录。

在 Windows 操作系统上，指定目录一定不能是 DB2 生成的目录。例如，提供以下命令:

```
db2 backup database sample to c:\backup
db2 restore database sample from c:\backup
```

DB2 在 `c:\backup` 目录下生成子目录，该目录应被忽略。要精确地指定要复原哪个备份映象，可使用 TAKEN AT 参数。可能会有数个备份映象存储在同一路径中。

如果指定了若干项，而最后一项是磁带机，会提示用户放入另一磁带。有效的响应选项是:

c 继续 — 继续使用生成了警告消息的设备（例如在安装了新磁带后继续）。

d 设备终止 — 只停止使用生成了警告消息的设备（例如，在没有更多磁带时终止）。

t 终止 — 在用户执行某些由实用程序请求的操作失败后，异常终止复原操作。

LOAD shared-library

共享库（在 Windows 操作系统上为 DLL）的名称，它包含要使用的供应商备份与复原 I/O 函数。该名称可包含全路径。如果未提供全路径，该值将缺省为用户出口程序所驻留的路径。

TAKEN AT date-time

数据库备份映象的时间戳记。时间戳记在成功完成备份操作后显示，并且是备份映象的路径名的一部分。以格式 *yyyymmddhhmmss* 指定。还可指定

部分时间戳记。例如，如果有两个不同的备份映象，时间戳记分别为 19971001010101 和 19971002010101，指定 19971002 会导致使用时间戳记为 19971002010101 的那个映象。如果未指定此参数的值，源媒体上必须只有一个备份映象。

TO target-directory

目标数据库目录。如果实用程序复原到一个现有数据库，将忽略此参数。指定的驱动器和目录必须是本地的。

注: 在 Windows 操作系统上，使用此参数时指定盘符。例如，可以指定 x:*path_name* 以复原至特定路径，或如果不需要指定路径，则指定 x:。如果路径名太长，会返回错误。

INTO target-database-alias

目标数据库别名。如果目标数据库不存在，则创建它。

将数据库备份复原至现有数据库时，复原的数据库继承现有数据库的别名和数据库名。将数据库备份复原至不存在的数据库时，使用您指定的别名和数据库名创建新数据库。这一新数据库名在复原它的系统上必须是唯一的。

NEWLOGPATH directory

在复原操作后用于活动日志文件的目录的绝对路径名。此参数与数据库配置参数 *newlogpath* 有相同的功能，但此参数的影响只限于它所指定的复原操作。当备份映象中的日志路径不适合在复原操作后使用时可使用此参数，例如，在该路径不再有效时或由另一数据库使用时。

WITH num-buffers BUFFERS

将要使用的缓冲区数量。缺省值是 2。但是在正从多个源读取数据时，或已增加了 PARALLELISM 的值时，可使用较大的缓冲区数以改进性能。

BUFFER buffer-size

将用于复原操作的，以页计的缓冲区大小。此参数的最小值是 8 页；缺省值是 1024 页。如果指定缓冲区大小为零，将使用数据库管理器配置参数 *restbufsz* 的值作为缓冲区分配大小。

复原缓冲区大小必须是在备份操作期间指定的备份缓冲区大小的正整数倍数。如果指定了不正确的缓冲区，则分配的缓冲区将是最小的可接受大小。

在 SCO UnixWare 7 上使用磁带机时，指定缓冲区大小 16。

DLREPORT filename

文件名（如果指定的话）必须指定为绝对路径。在复原操作期间，作为快速协调的结果，报告变为不再链接的文件。仅当正在复原的表有 DATALINK 列类型和链接文件时，才使用此选项。

REPLACE EXISTING

如果存在一个别名与目标数据库别名相同的数据库，此参数指定复原实用程序将使用复原数据库替换现有数据库。此参数对于调用复原实用程序的脚本很有用，因为命令行处理器将不会提示用户验证是否删除了现有数据库。如果指定了 WITHOUT PROMPTING 参数，则不需要指定 REPLACE EXISTING，但在这种情况中，如果发生了通常需要用户干预的事件，操作会失败。

REDIRECT

指定重定向复原操作。要完成重定向复原操作，在此命令后应跟有一个或多个 SET TABLESPACE CONTAINERS 命令，然后跟有指定了 CONTINUE 选项的 RESTORE DATABASE 命令。

注: 必须从同一窗口或 CLP 会话调用与一个重定向复原操作相关的所有命令。

WITHOUT ROLLING FORWARD

指定在成功复原数据库后，不要将该数据库置于前滚暂挂状态。

如果成功完成复原操作之后，该数据库处于前滚暂挂状态，必须调用 ROLLFORWARD 命令，才能再次使用该数据库。

WITHOUT DATALINK

指定将把任何带有 DATALINK 列的表置于 “DataLink 协调暂挂” (DRP) 状态，以及将不协调任何链接文件。

PARALLELISM n

指定将分散在复原操作期间的缓冲区操纵器的数量。缺省值为 1。

WITHOUT PROMPTING

指定复原操作将以无人照管方式运行。那些通常需要用户干预的操作将返回一条错误消息。如果使用可更换的媒体设备（例如磁带或软盘），即使未指定此选项也会在设备结束时提示用户。

示例:

在以下示例中，数据库 WSDB 是在编号为 0 至 3 的这 4 个分区上定义的。可以从所有分区存取路径 /dev3/backup。可以从 /dev3/backup 获取下列脱机备份映象:

```
wsdb.0.db2inst1.NODE0000.CATN0000.20020331234149.001
wsdb.0.db2inst1.NODE0001.CATN0000.20020331234427.001
wsdb.0.db2inst1.NODE0002.CATN0000.20020331234828.001
wsdb.0.db2inst1.NODE0003.CATN0000.20020331235235.001
```

要首先复原目录分区，然后从 /dev3/backup 目录复原 WSDB 数据库的所有其它数据库分区，从其中一个数据库分区发出下列命令:

```
db2_all '<<+0< db2 RESTORE DATABASE wsdb FROM /dev3/backup
TAKEN AT 20020331234149
  INTO wsdb REPLACE EXISTING'
db2_all '<<+1< db2 RESTORE DATABASE wsdb FROM /dev3/backup
TAKEN AT 20020331234427
  INTO wsdb REPLACE EXISTING'
db2_all '<<+2< db2 RESTORE DATABASE wsdb FROM /dev3/backup
TAKEN AT 20020331234828
  INTO wsdb REPLACE EXISTING'
db2_all '<<+3< db2 RESTORE DATABASE wsdb FROM /dev3/backup
TAKEN AT 20020331235235
  INTO wsdb REPLACE EXISTING'
```

db2_all 实用程序将对每个指定的数据库分区发出复原命令。

以下是一个典型的重定向复原方案，用于别名为 MYDB 的数据库:

1. 发出 RESTORE DATABASE 命令，使用 REDIRECT 选项。

   ```
   db2 restore db mydb replace existing redirect
   ```

   成功地完成了步骤 1 后，在完成步骤 3 前，可发出以下命令来异常终止复原操作:

   ```
   db2 restore db mydb abort
   ```

2. 对必须重新定义容器的每个表空间发出 SET TABLESPACE CONTAINERS 命令。例如:

   ```
   db2 set tablespace containers for 5 using
      (file 'f:\ts3con1' 20000, file 'f:\ts3con2' 20000)
   ```

   要验证复原数据库的容器是否是在此步骤中指定的那些容器，可发出 LIST TABLESPACE CONTAINERS 命令。

3. 在成功地完成了步骤 1 和 2 后，发出:

   ```
   db2 restore db mydb continue
   ```

   这是重定向复原操作的最后一步。

4. 如果步骤 3 失败，或者如果已异常终止了复原操作，则可从步骤 1 开始重新启动重定向的复原。

以下是对可恢复数据库每周进行的增量备份策略的样本。包括一个每周进行的完整数据库备份操作、一个每天进行的非累积（delta）备份操作以及一个在每周中期进行的累积（增量）备份操作:

```
(Sun) backup db mydb use tsm
(Mon) backup db mydb online incremental delta use tsm
(Tue) backup db mydb online incremental delta use tsm
(Wed) backup db mydb online incremental use tsm
(Thu) backup db mydb online incremental delta use tsm
(Fri) backup db mydb online incremental delta use tsm
(Sat) backup db mydb online incremental use tsm
```

要对在星期五早上创建的映象自动进行数据库复原，可发出:

```
restore db mydb incremental automatic taken at (Fri)
```

要对在星期五早上创建的映象手工进行数据库复原，可发出:

```
restore db mydb incremental taken at (Fri)
restore db mydb incremental taken at (Sun)
restore db mydb incremental taken at (Wed)
restore db mydb incremental taken at (Thu)
restore db mydb incremental taken at (Fri)
```

使用注意事项:

格式为 db2 restore db <name> 的任何 RESTORE DATABASE 命令将执行完全数据库复原，不论正在复原的映象是数据库映象还是表空间映象。格式为 db2 restore db <name> tablespace 的任何 RESTORE DATABASE 命令将执行在映象中找到的表空间的表空间复原。其中提供了表空间列表的任何 RESTORE DATABASE 命令将执行任何显式列示的表空间的复原。

相关参考:

- 第 63 页的『BACKUP DATABASE』
- 第 117 页的『ROLLFORWARD DATABASE』
- 『db2move - Database Movement Tool Command』（*Command Reference*）

db2Restore — 复原数据库

重新构建使用 “db2Backup — 备份数据库” 备份的损坏或毁坏的数据库。复原的数据库与创建备份副本时它所处的状态相同。此实用程序还可以使用不同于备份映象中数据库名称的名称来复原数据库（还能复原至新数据库）。

此实用程序还可以用来复原在前两个发行版中创建的 DB2 数据库。

此实用程序还可以从表空间级备份复原。

作用域:

此 API 只影响从中调用该 API 的数据库分区。

授权:

要复原至现有数据库，授予下列其中一个权限:

- *sysadm*
- *sysctrl*
- *sysmaint*

要复原至新数据库，授予下列其中一个权限:

- *sysadm*
- *sysctrl*

必需的连接:

要复原至现有数据库的数据库。此 API 自动建立与指定数据库的连接，并将在复原操作完成时释放该连接。

要复原至新数据库的实例和数据库。创建数据库需要实例连接。

要复原至不同于当前实例的实例上的新数据库（由 DB2INSTANCE 环境变量的值定义），必须首先与新数据库将驻留的实例连接。

API 包含文件:

db2ApiDf.h

C API 语法:

```
/* File: db2ApiDf.h */
/* API: db2Restore */
/* ... */
SQL_API_RC SQL_API_FN
db2Restore (
  db2Uint32 versionNumber,
  void         *pDB2RestoreStruct,
      struct sqlca * pSqlca);
/* ... */

typedef SQL_STRUCTURE db2RestoreStruct
{
  char                         *piSourceDBAlias;
  char                         *piTargetDBAlias;
  char                         oApplicationId[SQLU_APPLID_LEN+1];
  char                         *piTimestamp;
  char                         *piTargetDBPath;
  char                         *piReportFile;
  struct db2TablespaceStruct   *piTablespaceList;
  struct db2MediaListStruct    *piMediaList;
  char                         *piUsername;
  char                         *piPassword;
  char                         *piNewLogPath;
  void                         *piVendorOptions;
  db2Uint32                    iVendorOptionsSize;
  db2Uint32                    iParallelism;
  db2Uint32                    iBufferSize;
  db2Uint32                    iNumBuffers;
  db2Uint32                    iCallerAction;
   db2Uint32 iOptions;
} db2BackupStruct;

typedef SQL_STRUCTURE db2TablespaceStruct
{
  char                         **tablespaces;
  db2Uint32                    numTablespaces;
} db2TablespaceStruct;

typedef SQL_STRUCTURE db2MediaListStruct
{
  char                         **locations;
  db2Uint32                    numLocations;
  char                         locationType;
} db2MediaListStruct;
/* ... */
```

一般 API 语法:

```
/* File: db2ApiDf.h */
/* API: db2gRestore */
/* ... */
SQL_API_RC SQL_API_FN
db2gRestore (
  db2Uint32 versionNumber,
  void         *pDB2gRestoreStruct,
```

```
      struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gRestoreStruct
{
  char                         *piSourceDBAlias;
  db2Uint32                    iSourceDBAliasLen;
  char                         *piTargetDBAlias;
  db2Uint32                    iTargetDBAliasLen;
  char                         *poApplicationId;
  db2Uint32                    iApplicationIdLen;
  char                         *piTimestamp;
  db2Uint32                    iTimestampLen;
  char                         *piTargetDBPath;
  db2Uint32                    iTargetDBPathLen;
  char                         *piReportFile;
  db2Uint32                    iReportFileLen;
  struct db2gTablespaceStruct *piTablespaceList;
  struct db2gMediaListStruct  *piMediaList;
  char                         *piUsername;
  db2Uint32                    iUsernameLen;
  char                         *piPassword;
   db2Uint32 iPasswordLen;
  char                         *piNewLogPath;
  db2Uint32                    iNewLogPathLen;
  void                         *piVendorOptions;
  db2Uint32                    iVendorOptionsSize;
  db2Uint32                    iParallelism;
  db2Uint32                    iBufferSize;
  db2Uint32                    iNumBuffers;
  db2Uint32                    iCallerAction;
   db2Uint32 iOptions;
} db2gBackupStruct;

typedef SQL_STRUCTURE db2gTablespaceStruct
{
  struct db2Char               *tablespaces;
  db2Uint32                    numTablespaces;
} db2gTablespaceStruct;

typedef SQL_STRUCTURE db2gMediaListStruct
{
  struct db2Char               *locations;
  db2Uint32                    numLocations;
  char                         locationType;
} db2gMediaListStruct;

typedef SQL_STRUCTURE db2Char
{
   char            *pioData;
   db2Uint32       iLength;
   db2Uint32       oLength;
} db2Char;
/* ... */
```

API 参数:

versionNumber

输入。指定作为第二个参数 pParamStruct 传送的结构的版本和发行版级别。

pDB2RestoreStruct

输入。指向 *db2RestoreStruct* 结构的指针

pSqlca

输出。指向 *sqlca* 结构的指针。

piSourceDBAlias

输入。包含了源数据库备份映象的数据库别名的字符串。

iSourceDBAliasLen

输入。一个 4 字节的无符号整数，表示源数据库别名的长度（以字节计）。

piTargetDBAlias

输入。包含了目标数据库别名的字符串。如果此参数为 null，将使用 *piSourceDBAlias*。

iTargetDBAliasLen

输入。一个 4 字节的无符号整数，表示目标数据库别名的长度（以字节计）。

oApplicationId

输出。此 API 将返回一个标识为应用程序提供服务的代理进程的字符串。可以用来通过使用数据库监视器获取有关备份操作的进度的信息。

poApplicationId

输出。提供缓冲区长度 SQLU_APPLID_LEN+1（在 `sqlutil` 中定义）。此 API 将返回一个标识为应用程序提供服务的代理进程的字符串。可以用来通过使用数据库监视器获取有关备份操作的进度的信息。

iApplicationIdLen

输入。一个 4 字节的无符号整数，表示 poApplicationId 缓冲区的长度（以字节计）。应该等于 SQLU_APPLID_LEN+1（在 `sqlutil` 中定义）。

piTimestamp

输入。一个代表备份映象的时间戳记的字符串。如果在指定的源中只有一个备份映象，则此字段是可选的。

iTimestampLen

输入。一个 4 字节的无符号整数，表示 piTimestamp 缓冲区的长度（以字节计）。

piTargetDBPath

输入。包含服务器上的目标数据库目录的相对名称或全限定名称的字符串。如果要为复原备份创建新数据库，则使用此参数；否则，不使用此参数。

piReportFile

输入。文件名必须是全限定名称（如果指定的话）。将报告在复原期间变得不可链接的 datalinks 文件（这是快速协调的结果）。

iReportFileLen

输入。一个 4 字节的无符号整数，表示 piReportFile 缓冲区的长度（以字节计）。

piTablespaceList

输入。要复原的表空间的列表。当从数据库或表空间备份映象复原表空间的子集时，使用此参数。参见 *DB2TablespaceStruct* 结构。下列限制适用:

- 数据库必须是可恢复的；即，必须启用日志保留或用户出口。
- 正在复原的数据库必须与用于创建备份映象的数据库是同一数据库。即，表空间不能通过表空间复原函数来添加到数据库。
- 前滚实用程序将确保在分区数据库环境中复原的表空间与包含相同表空间的任何其它数据库分区同步。如果请求表空间复原操作，且 *piTablespaceList* 为 NULL，则复原实用程序将尝试复原备份映象中所有表空间。

复原自备份后已重命名的表空间时，在复原命令中必须使用新的表空间名。如果使用旧名称，将找不到该表空间。

piMediaList

输入。备份映象的源媒体。所提供的信息取决于 locationType 字段的值。locationType 的有效值（在 `sqlutil` 中定义）为:

SQLU_LOCAL_MEDIA

本地设备（磁带、磁盘或软盘的组合）。

SQLU_TSM_MEDIA

TSM。如果位置指针设置为 NULL，则使用 DB2 附带提供的 TSM 共享库。如果希望使用 TSM 共享库的另一版本，使用 SQLU_OTHER_MEDIA 并提供共享库名。

SQLU_OTHER_MEDIA

供应商产品。提供位置字段中的共享库名。

SQLU_USER_EXIT

用户出口。不需要任何附加输入（仅当服务器在 OS/2 上时才可用）。

piUsername

输入。包含尝试连接时将使用的用户名的字符串。可以为 NULL。

iUsernameLen

输入。一个 4 字节的无符号整数，表示 piUsername 的长度（以字节计）。如果不提供用户名，则设置为零。

piPassword

输入。包含将与用户名一起使用的密码的字符串。可以为 NULL。

iPasswordLen

输入。一个 4 字节的无符号整数，表示 piPassword 的长度（以字节计）。如果不提供密码，则设置为零。

piNewLogPath

输入。表示要用于复原完成后进行日志记录的路径的字符串。如果此字段为 null，将使用缺省日志路径。

iNewLogPathLen

输入。一个 4 字节的无符号整数，表示 *piNewLogPath* 的长度（以字节计）。

piVendorOptions

输入。用于将信息从应用程序发送给供应商函数。此数据结构必须是平面的，因此不支持间接级别。注意，字节逆转未完成，且未检查此数据的代码页。

iVendorOptionsSize

输入。*piVendorOptions* 的长度，不能超过 65535 字节。

iParallelism

输入。并行性的程度（缓冲区操纵器的数量）。最小值为 1。最大值为 1024。缺省值为 1。

iBufferSize

输入。以 4KB 分配单位（页）计的备份缓冲区大小。最小值是 8 个单元。缺省值是 1024 个单元。对复原输入的大小必须等于用来生成备份映象的缓冲区大小或它的整数倍。

iNumBuffers

输入。指定要使用的复原缓冲区的数目。

iCallerAction

输入。指定要采取的操作。有效值（在 `db2ApiDf` 中定义）为:

DB2RESTORE_RESTORE

启动复原操作。

DB2RESTORE_NOINTERRUPT
启动复原。指定复原将以无人照管方式运行，将尝试那些通常需要用户干预的方案而不首先返回到调用程序，或者将生成错误。例如，如果已知复原所需的所有媒体都已安装，且不希望出现实用程序提示，则使用此调用程序操作。

DB2RESTORE_CONTINUE
在用户已执行实用程序请求的某些操作后继续复原操作（例如，安装新磁带）。

DB2RESTORE_TERMINATE
在用户执行实用程序请求的某些操作失败后终止复原。

DB2RESTORE_DEVICE_TERMINATE
从复原使用的设备的列表中除去特定设备。在特定设备用完它的输入时，复原将对调用程序返回警告。使用此调用程序操作再次调用复原，以从正在使用的设备的列表中除去生成警告的设备。

DB2RESTORE_PARM_CHK
用来验证参数而不执行复原。此选项在返回调用后不终止数据库连接。在成功返回此调用后，期望用户发出指定 DB2RESTORE_CONTINUE 的调用以继续执行操作。

DB2RESTORE_PARM_CHK_ONLY
用来验证参数而不执行复原。在此调用返回前，将终止由此调用建立的数据库连接，且不需要后续调用。

DB2RESTORE_TERMINATE_INCRE
在增量复原操作完成前终止它。

DB2RESTORE_RESTORE_STORDEF
初始调用。请求的表空间容器重定义。

DB2RESTORE_STORDEF_NOINTERRUPT
初始调用。复原将以不中断方式运行。请求的表空间容器重定义。

iOptions
输入。复原特性的位图。要通过使用一位宽度 OR 运算符来生成 *iOptions* 的值以将选项组合起来。有效值（在 db2ApiDf 中定义）为:

DB2RESTORE_OFFLINE
执行脱机复原操作。

DB2RESTORE_ONLINE
执行联机复原操作。

DB2RESTORE_DB
复原数据库中的所有表空间。这必须脱机运行

DB2RESTORE_TABLESPACE
仅从备份映象复原 *piTablespaceList* 参数中列示的表空间。这可以联机或脱机运行。

DB2RESTORE_HISTORY
仅复原历史文件。

DB2RESTORE_INCREMENTAL
执行手工累积复原操作。

DB2RESTORE_AUTOMATIC
执行自动累积（增量）复原操作。必须使用 DB2RESTORE_INCREMENTAL 来指定。

DB2RESTORE_DATALINK
执行协调操作。对于带有已定义 DATALINK 列的表，必须指定 RECOVERY YES 选项。

DB2RESTORE_NODATALINK
不执行协调操作。将带有 DATALINK 列的表置于 DataLink_Roconcile_pending（DRP）状态。对于带有已定义 DATALINK 列的表，必须指定 RECOVERY YES 选项。

DB2RESTORE_ROLLFWD
在成功复原后，将数据库置于前滚暂挂状态。

DB2RESTORE_NOROLLFWD
在成功复原后，不要将数据库置于前滚暂挂状态。不能对联机备份或表空间级复原指定此参数。如果成功复原后该数据库处于前滚暂挂状态，必须执行 “db2Rollforward — 前滚数据库”，才能使用该数据库。

tablespaces
指向要备份的表空间的列表的指针。对于 C，该列表是以空字符结尾的字符串。在一般情况下，它是 *db2Char* 结构的列表。

numTablespaces
tablespaces 参数中的条目数。

locations
指向媒体位置的列表的指针。对于 C，该列表是以空字符结尾的字符串。在一般情况下，它是 db2Char 结构的列表。

numLocations
locations 参数中的条目数。

locationType
指示媒体类型的字符。有效值（在 sqlutil 中定义）是:

SQLU_LOCAL_MEDIA
本地设备（磁带、磁盘、软盘或命名管道）。

SQLU_TSM_MEDIA
Tivoli Storage Manager。

SQLU_OTHER_MEDIA
供应商库。

SQLU_USER_EXIT
用户出口（仅当服务器在 OS/2 上时才可用）。

pioData
指向字符数据缓冲区的指针。

iLength
输入。*pioData* 缓冲区的大小

oLength
输出。保留以供将来使用。

使用注意事项:

对于脱机复原，此实用程序以独占方式连接至数据库。如果任何应用程序（包括调用应用程序）已连接至正在复原的数据库，则该实用程序失败。另外，如果正使用复原实用程序执行复原，且任何应用程序（包括调用应用程序）已连接至同一工作站上的任何数据库，则请求将失败。如果连接成功，则 API 锁住其它应用程序，直到复原完成为止。

将不会用备份副本替换当前数据库配置文件，除非它不可用。如果替换了该文件，则返回警告消息。

必须已使用 “db2Backup — 备份数据库” 备份数据库或表空间。

如果调用程序操作是 DB2RESTORE_NOINTERRUPT，则复原继续进行而不提示应用程序。如果调用程序操作是 DB2RESTORE_RESTORE，且实用程序正复原至现有数据库，则实用程序将控制权返回给应用程序，并出现一条消息，请求一些用户交互作用。在处理用户交互作用之后，应用程序再次调用 RESTORE DATABASE，并设置调用程序操作以指示是继续处理

（DB2RESTORE_CONTINUE）还是终止后续调用（DB2RESTORE_TERMINATE）。实用程序完成处理，并在 *sqlca* 中返回 SQLCODE。

为了在完成时关闭设备，将调用程序操作设置为 DB2RESTORE_DEVICE_TERMINATE。例如，如果用户正在从使用 2 个磁带设备的 3 个磁带卷进行复原，且其中一盘磁带已复原，则应用程序从 API 获取控制权，并出现 SQLCODE，指示到达磁带结尾。应用程序可以提示用户安装另一磁带，且如果用户指示“不要”，则使用指示媒体设备结束的调用程序操作 SQLUD_DEVICE_TERMINATE 返回至 API。将终止设备驱动程序，但对于复原涉及的余下设备，将继续处理它们的输入，直到复原集的所有片段完成复原（备份进程期间，复原集中的片段数放置在最后一个媒体设备上）。此调用程序操作可以与不同于磁带的设备（受供应商支持的设备）配合使用。

要在返回至应用程序之前执行参数检查，将调用程序操作设置为 DB2RESTORE_PARM_CHK。

执行重定向复原时将调用程序操作设置为 DB2RESTORE_RESTORE_STORDEF; 与“sqlbstsc — 设置表空间容器”一起使用。

如果复原数据库的关键阶段期间发生系统故障，则在执行复原成功之前，用户将无法成功连接至数据库。尝试连接时将检测此情况，并返回错误消息。如果未对前滚恢复配置备份数据库，且存在已启用这两个参数之一的可用当前配置文件，则在复原之后，在连接至数据库之前，用户将需要对数据库进行新的备份，或禁用日志保留和用户出口参数。

虽然将不删除复原的数据库（除非复原至不存在的数据库），但是如果复原失败，则它将不可用。

如果复原类型指定要复原备份上的历史文件，则将对数据库的现有历史文件进行复原，这将有效地擦除在正在复原的备份之后对历史文件所作的所有更改。如果不想要这样，则将历史文件复原至新数据库或测试数据库，以便查看它的内容而不破坏已进行的任何更新。

如果在执行备份操作时，对前滚恢复启用了数据库，则通过在成功执行 db2Restore 后发出 db2Rollforward，可以将数据库置于发生损坏或毁坏之前的状态。如果数据库是可恢复的，则在完成复原之后它将缺省为前滚暂挂状态。

如果数据库备份映象处于脱机状态，且在复原之后调用程序不想前滚数据库，则可以将 DB2RESTORE_NOROLLFWD 选项用于复原。这将导致数据库在复原之后立即可用。如果备份映象处于联机状态，则在完成复原时调用程序必须前滚相应日志记录。

相关样本:

- 『dbrecov.sqc -- How to recover a database (C)』
- 『dbrecov.sqC -- How to recover a database (C++)』

复原会话 — CLP 示例

示例 1

以下是对别名为 MYDB 的数据库的典型非增量重定向复原方案:

1. 发出 RESTORE DATABASE 命令，使用 REDIRECT 选项。

   ```
   db2 restore db mydb replace existing redirect
   ```

2. 对想要重新定义其容器的每个表空间发出 SET TABLESPACE CONTAINERS 命令。例如，在 Windows 环境中:

   ```
   db2 set tablespace containers for 5 using
      (file 'f:\ts3con1' 20000, file 'f:\ts3con2' 20000)
   ```

   要验证复原数据库的容器是否是在此步骤中指定的那些容器，对正重新定义其容器位置的每个表空间发出 LIST TABLESPACE CONTAINERS 命令。

3. 在成功完成了步骤 1 和 2 之后，发出:

   ```
   db2 restore db mydb continue
   ```

   这是重定向复原操作的最后一步。

4. 如果步骤 3 失败，或者如果已异常终止了复原操作，则可从步骤 1 开始重新启动重定向的复原。

注:

1. 成功完成步骤 1 之后且在完成步骤 3 之前，通过发出以下命令来异常终止复原操作:

   ```
   db2 restore db mydb abort
   ```

2. 如果步骤 3 失败，或者如果已异常终止了复原操作，则可从步骤 1 开始重新启动重定向的复原。

示例 2

以下是对别名为 MYDB 且具有下列备份映象的数据库的典型非增量重定向复原方案:

```
backup db mydb
备份成功。此备份的时间戳记是: <ts1>
backup db mydb incremental
备份成功。此备份映象的时间戳记为:  <ts2>
```

1. 发出带有 INCREMENTAL 和 REDIRECT 选项的 RESTORE DATABASE 命令。

   ```
   db2 restore db mydb incremental taken at <ts2> replace existing redirect
   ```

2. 对必须重新定义容器的每个表空间发出 SET TABLESPACE CONTAINERS 命令。例如，在 Windows 环境中:

   ```
   db2 set tablespace containers for 5 using
      (file 'f:\ts3con1' 20000, file 'f:\ts3con2' 20000)
   ```

   要验证复原数据库的容器是否是在此步骤中指定的那些容器，可发出 LIST TABLESPACE CONTAINERS 命令。

3. 在成功完成了步骤 1 和 2 之后，发出:

   ```
   db2 restore db mydb continue
   ```

4. 现在可以发出余下的增量复原命令，如下所示:

   ```
   db2 restore db mydb incremental taken at <ts1>
   db2 restore db mydb incremental taken at <ts2>
   ```

   这是重定向复原操作的最后一步。

注:

1. 成功完成步骤 1 之后且在完成步骤 3 之前，通过发出以下命令来异常终止复原操作:

   ```
   db2 restore db mydb abort
   ```

2. 成功完成步骤 3 之后且在发出步骤 4 中所有必需的命令之前，可以通过发出以下命令来异常终止复原操作:

   ```
   db2 restore db mydb incremental abort
   ```

3. 如果步骤 3 失败，或者如果已异常终止了复原操作，则可从步骤 1 开始重新启动重定向的复原。
4. 如果步骤 4 中的复原命令失败，则可以重新发出失败的命令以继续复原过程。

示例 3

以下是对同一个数据库的典型自动增量重定向复原方案:

1. 发出带有 INCREMENTAL AUTOMATIC 和 REDIRECT 选项的 RESTORE DATABASE 命令。

   ```
   db2 restore db mydb incremental automatic taken at <ts2>
          replace existing redirect
   ```

2. 对必须重新定义容器的每个表空间发出 SET TABLESPACE CONTAINERS 命令。例如，在 Windows 环境中:

```
db2 set tablespace containers for 5 using
   (file 'f:\ts3con1' 20000, file 'f:\ts3con2' 20000)
```

要验证复原数据库的容器是否是在此步骤中指定的那些容器，可发出 LIST TABLESPACE CONTAINERS 命令。

3. 在成功完成了步骤 1 和 2 之后，发出:

```
db2 restore db mydb continue
```

这是重定向复原操作的最后一步。

注:

1. 成功完成步骤 1 之后且在完成步骤 3 之前，通过发出以下命令来异常终止复原操作:

```
db2 restore db mydb abort
```

2. 如果步骤 3 失败，或者如果复原操作已异常终止，则可以在发出以下命令之后从步骤 1 开始重新启动重定向复原:

```
db2 restore db mydb incremental abort
```

相关参考:

- 第 84 页的『RESTORE DATABASE』
- 『LIST TABLESPACE CONTAINERS Command』（*Command Reference*）
- 『SET TABLESPACE CONTAINERS Command』（*Command Reference*）

第 4 章 前滚恢复

这部分描述了 DB2 UDB 前滚实用程序，它通过应用记录在数据库恢复日志文件中的事务来恢复数据库。

包括下列主题:

前滚概述

DB2® ROLLFORWARD DATABASE 命令的最简单的格式只需要您指定想要前滚恢复的数据库的别名。例如:

```
db2 rollforward db sample to end of logs and stop
```

在此示例中，命令返回:

```
                                Rollforward Status

 Input database alias                   = sample
 Number of nodes have returned status   = 1

 Node number                            = 0
 Rollforward status                     = not pending
 Next log file to be read               =
 Log files processed                    =  -
 Last committed transaction             = 2001-03-11-02.39.48.000000

DB20000I  ROLLFORWARD 命令成功完成。
```

前滚恢复的常见方法涉及:

1. 调用前滚实用程序，而不使用 STOP 选项。
2. 调用前滚实用程序，使用 QUERY STATUS 选项

   如果指定恢复到日志末尾，而返回的时间点又早于您预期的时间点，QUERY STATUS 选项会指示丢失了一个或多个日志文件。

   如果指定时间点恢复，QUERY STATUS 选项将有助于您确保前滚操作已在正确的时间点完成。
3. 使用 STOP 选项调用前滚实用程序。在操作停止后，不可能前滚其它更改。

必须已成功地复原了数据库（使用复原实用程序）才能前滚它，但对表空间则不是这样。表空间可以暂时置于前滚暂挂状态，但不需要复原操作撤销它（例如在电源中断后）。

调用前滚实用程序时:

- 如果数据库处于前滚暂挂状态，将前滚该数据库。如果表空间也处于前滚暂挂状态，则必须在数据库前滚操作完成后，再次调用前滚实用程序来前滚该表空间。
- 如果数据库*不*处于前滚暂挂状态，但该数据库中的表空间*处于*前滚暂挂状态:
  - 如果指定表空间列表，将只前滚那些表空间。
  - 如果不指定表空间列表，将前滚处于前滚暂挂状态的所有表空间。

数据库前滚操作是脱机运行的。直到前滚操作成功完成数据库才可用，除非在调用实用程序时指定了 STOP 选项，否则该操作不能完成。

表空间前滚操作可脱机运行。直到前滚操作成功完成数据库才可用。当达到日志末尾时或在调用实用程序时指定了 STOP 选项，会发生这种情况。

只要表空间中包括了 SYSCATSPACE，就可对表空间执行*联机*前滚操作。对表空间执行联机前滚操作时，该表空间不可用，但数据库中的其它表空间是可用的。

首次创建数据库时，只对它启用了循环记录。这意味着日志是可重用的，而不是被保存或归档。使用循环日志，是不能进行前滚恢复的：只能进行崩溃恢复或版本恢复。归档日志将建立备份后对数据库进行的更改记录成文档。通过将 *logretain* 数据库配置参数设置为 RECOVERY 或将 *userexit* 数据库配置参数设置为 YES，或同时设置这两个参数，启用日志归档（和前滚恢复）。这两个参数的缺省值是 NO 因为最初没有备份映象可用于复原数据库。更改这两个参数的值或其中一个的值时，数据库将处于备份暂挂状态，必须脱机备份该数据库才能再次使用它。

相关概念:

- 第 114 页的『使用装入副本位置文件』
- 第 30 页的『了解恢复日志』

相关参考:

- 第 117 页的『ROLLFORWARD DATABASE』
- 第 34 页的『数据库记录的配置参数』

使用前滚所需的特权、权限和授权

特权使用户能够创建或访问数据库资源。权限级别提供将特权、高级数据库管理器维护和实用程序操作分组的方法。这两者一起用于控制对数据库管理器和它的数据库对象的访问。用户只能访问那些他们具有适当授权（即必需的特权或权限）的对象。

必须具有 SYSADM、SYSCTRL 或 SYSMAINT 权限才能使用前滚实用程序。

使用前滚

先决条件:

不应连接到要进行前滚恢复的数据库：前滚实用程序会自动建立与特定数据库的连接，且此连接在前滚操作完成时终止。

要在复原表空间时取消正在进行的前滚操作；否则您可能会在一个表空间集合中有某些表空间处于“前滚进行中”状态，而另一些表空间则处于“前滚暂挂”状态。正在进行中的前滚操作将只在处于“前滚进行中”状态的表空间上执行。

数据库可以是本地的或远程的。

限制:

前滚实用程序有以下局限性:

- 一次只能调用一个前滚操作。如果有多个要复原的表空间，可在同一操作中指定所有表空间。
- 若在最近的备份操作后有重命名的表空间，应确保在前滚表空间时使用新名称。先前表空间名将不被识别。
- 不能取消正在运行的前滚操作。只能取消已完成的前滚操作，对那些尚未指定 STOP 选项或在完成前已失败的前滚操作则不能取消。
- 指定的时间戳记小于前一时间戳记时，不能继续将表空间前滚至某时间点。如果未指定时间点，将使用前一时间点。可以只指定 STOP 来启动前滚至某时间

点的操作，但只有在涉及的表空间都是从同一脱机备份映象复原时才能这样做。在这种情况中，不需要日志处理。如果在正在进行的前滚操作已完成或取消前对另一表空间列表启动前滚操作，会返回一条错误消息（SQL4908）。对所有节点调用 LIST TABLESPACES 命令，以确定当前有哪个表空间正在前滚（前滚进行中状态），哪些表空间正准备前滚（前滚暂挂状态）。有 3 种选择:

  – 完成所有表空间上正在进行的前滚操作。
  – 完成表空间子集上正在进行的前滚操作。（如果前滚操作将继续到某特定的时间点，从而需要涉及到所有节点，则可能无法完成表空间子集上正在进行的前滚操作。
  – 取消正在进行的前滚操作。

- 在分区数据库环境中，必须从数据库的目录节点发出前滚实用程序。

过程:

前滚实用程序可以通过命令行处理器（CLP）、“控制中心”的“前滚数据库”笔记本或 **db2Rollforward** 应用程序编程接口（API）来调用。

以下是通过 CLP 发出的 ROLLFORWARD DATABASE 命令的示例:

```
db2 rollforward db sample to end of logs and stop
```

要打开“前滚数据库”笔记本:

1. 从“控制中心”中，展开对象树，直到找到“数据库”文件夹。
2. 单击“数据库”文件夹。所有现有的数据库都显示在该窗口右边的窗格（内容窗格）中。
3. 在内容窗格中，用鼠标右键单击需要的数据库，然后从弹出菜单中选择“前滚”。“前滚数据库”笔记本打开。

在“控制中心”中通过联机帮助设施提供了详细信息。

相关概念:

- 『Administrative APIs in Embedded SQL or DB2 CLI Programs』（*Application Development Guide: Programming Client Applications*）
- 『引入“控制中心”的插件体系结构』（*《管理指南: 实现》*）

相关参考:

- 第 127 页的『db2Rollforward — 前滚数据库』

前滚表空间中的更改

如果数据库启用了前滚恢复，您可以选择备份、复原和前滚表空间，而不必前滚整个数据库。可能需要对个别表空间实现恢复策略，因为这可节省时间：复原数据库的一部分所需的时间比恢复整个数据库所需的时间少。例如，若一个磁盘损坏且它只包含一个表空间，则可以复原并前滚该表空间，而不必复原整个数据库，也不会影响用户访问该数据库的其余部分，除非损坏的表空间包含了系统目录表；在这种情况中，您不能连接至数据库。（如果包含系统目录表空间的表空间级备份映象是可用的，则可以独立复原系统目录表空间。）表空间级的备份也允许您更频繁地备份数据库的关键部分，因此所需的时间比备份整个数据库的时间少。

复原表空间后，它始终处于前滚暂挂状态。要使该表空间可以使用，必须对它执行前滚恢复。在大多数情况中，可以选择前滚至日志末尾或前滚至特定的时间点。但是不能将包含系统目录表的表空间前滚至某时间点。这些表空间必须前滚至日志末尾，以确保数据库中的所有表空间保持一致。

当前滚表空间时，DB2® 将跳过已知不包含影响该表空间的任何日志记录的文件。如果想要处理所有日志文件，则将 DB2_COLLECT_TS_REC_INFO 注册表变量设置为 false。

在数据库目录中的表空间更改历史文件（DB2TSCHG.HIS）记录应对每个表空间处理哪些日志。通过使用 **db2logsForRfwd** 实用程序可以查看此文件的内容，通过使用 PRUNE HISTORY 命令可以从中删除条目。在数据库复原操作期间，从备份映象复原 DB2TSCHG.HIS，然后在数据库前滚操作期间对它进行更新。如果没有任何信息可用于日志文件，则就好象每个表空间的恢复都需要该日志文件一样来处理它。

因为在日志不活动之后每个日志文件的信息都会清仓至磁盘，所以崩溃会导致此信息可能丢失。为弥补这一点，如果恢复操作从日志文件的中间开始，则就好象该日志包含对系统中每个表空间的修改来处理整个日志。在这之后，将处理活动日志并重新构建它们的信息。如果在崩溃情况下丢失较旧的或归档的日志文件的信息，且它们在数据文件中没有任何信息，则就好象它们包含表空间恢复操作期间对每个表空间的修改一样来处理这些日志文件。

在前滚表空间之前，调用 LIST TABLESPACES SHOW DETAIL 命令。此命令将返回*最小恢复时间*，这是表空间可前滚至的最早时间点。对表空间或表空间中的表运行数据定义语言（DDL）语句时，将更新此最小恢复时间。必须将该表空间至少前滚至最小恢复时间，以便与系统目录表中的信息同步。如果恢复多个表空间，表空间必须至少前滚至要恢复的所有表空间中的最高“最小恢复时间”。在分

区数据库环境中，对所有分区发出 LIST TABLESPACES SHOW DETAIL 命令。表空间必须前滚至所有分区上的所有表空间中的最高 “最小恢复时间”。

如果要将表空间前滚到某时间点，而且有一个表包含在多个表空间中，则必须同时前滚所有这些表空间。例如，若该表数据包含在一个表空间中，而该表的索引包含在另一个表空间中，则必须同时将这两个表空间前滚至相同的时间点。

如果表中的数据和长对象位于不同的表空间中，并且已将该长对象数据重组，则必须同时复原并前滚数据和长对象的表空间。在重组该表之后，应该为受影响的表空间建立备份。

若要将一个表空间前滚到某时间点，则表空间中的表是:

- 另一个表空间中的具体查询或分级表的基础表
- 另一个表空间中的表的具体查询或分级表

应该将两个表空间前滚到同一时间点。否则，在前滚操作结束时，具体查询或分级表将处于检查暂挂状态。具体查询表将需要完全刷新，并且分级表将被标记为不完整。

若要将一个表空间前滚至一个时间点，而该表空间中的一个表与另一个表空间所包含的另一个表存在引用完整性关系，则应同时将这两个表空间前滚至相同的时间点。否则，在前滚操作结束时，引用完整性关系中的子表将处于检查暂挂状态。以后检查子表是否存在约束违规时，需要对整个表进行检查。如果下列其中任何一个表存在，它们将与子表一起处于检查暂挂状态:

- 子表的任何后代具体查询表
- 子表的任何后代分级表
- 子表的任何后代外键表

这些表将需要处理以使它们脱离检查暂挂状态。若同时前滚这两个表空间，则该约束将在该时间点前滚操作结束保持有效。

应确保时间点表空间前滚操作不会导致在某些表空间中回滚一个事务，而在另一些表空间中落实该事务。这种情况可能在下列时候发生:

- 对一个事务已更新的表空间的子集执行时间点前滚操作，且该时间点在事务落实的时间之前。
- 要前滚至某个时间点的表空间中所包含的任何表有一个相关的触发器，或者这个表由一个触发器来更新，该触发器影响的表空间不是要前滚的表空间。

解决方案是: 找到一个可阻止这种情况发生的适当的时间点。

可以发出 QUIESCE TABLESPACES FOR TABLE 命令来创建事务一致的时间点，以将表空间前滚。停顿请求（是共享的，用于更新或独占方式）等待（通过锁

定）对那些表空间运行的所有事务完成，并阻拦新请求。准许停顿请求时，表空间处于一致状态。要确定停止前滚操作的适当时间，可查看恢复历史文件以找出停顿点，并检查它是否是在最小复原时间后发生的。

表空间时间点前滚操作完成后，表空间将置于备份暂挂状态。必须建立该表空间的备份，因为在前滚到的时间点与当前时间之间对该表空间所做的所有更新都已被除去。再也不能从先前的数据库级或表空间级备份映象将该表空间前滚到当前时间。以下示例显示表空间级的备份映象为什么是必需的，以及如何使用它。（要使表空间可用，可以备份整个数据库、处于备份暂挂状态的表空间，或包括处于备份暂挂状态的表空间的表空间集合。）

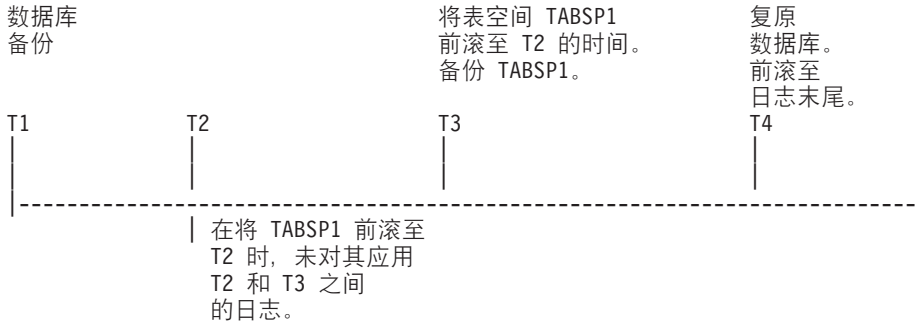


图 15. 表空间备份需求

在上述示例中，在 T1 时备份数据库。然后在 T3 时，表空间 TABSP1 前滚至某特定的时间点（T2），表空间在时间 T3 后备份。因为表空间处于备份暂挂状态，所以备份操作是必需的。表空间备份映象的时间戳记在 T3 后，但表空间是在 T2。不将 T2 和 T3 之间的日志记录应用于 TABSP1。在时间 T4 处，将使用在 T1 处创建的备份映象复原数据库，并前滚至日志末尾。表空间 TABSP1 将在时间 T3 处置于复原暂挂状态，因为数据库管理器假设在 T3 和 T4 之间，对 TABSP1 执行操作，而无需已对表空间应用了 T2 和 T3 之间的日志更改。如果这些日志更改是作为对数据库的前滚操作的一部分应用的，该假设将不成立。必须在表空间前滚到某时间点后建立的表空间级的备份，使您可将那个表空间前滚过前一时间点前滚操作（在上例中为 T3）。

假定要将表空间 TABSP1 复原至 T4，应该从 T3 之后建立的备份映象（必需的备份或稍后的一个）复原该表空间，然后将 TABSP1 前滚至日志末尾。

在前面的示例中，将数据库复原至时间 T4 最有效的方法应是按下列次序执行必需的步骤:

1. 复原数据库。
2. 复原表空间。

3. 前滚数据库。
4. 前滚表空间。

因为您在前滚数据库之前复原表空间，所以在前滚数据库时将不使用资源向表空间应用日志记录。

如果找不到时间 T3 后的 TABSP1 备份映象，或者想将 TABSP1 复原到 T3（或更早），可以:

- 将表空间前滚至 T3。无需再次复原表空间，因为它已根据数据库备份映象复原。
- 再次使用在时间 T1 建立的数据库备份来复原表空间，然后将该表空间前滚至时间 T3 之前的一个时间。
- 删除表空间。

在分区数据库环境中:

- 必须同时将一个表空间的所有部分前滚至同一时间点。这可确保该表空间在各数据库分区中是一致的。
- 如果某些数据库分区处于前滚暂挂状态，而且在其它数据库分区上，某些表空间处于前滚暂挂状态（但数据库分区不是），您必须首先前滚数据库分区，然后前滚表空间。
- 如果打算将表空间前滚到日志末尾，不必在每个数据库分区上复原: 只需在需要恢复的数据库分区上复原即可。但如果打算将表空间前滚到某时间点，则必须在每个数据库分区上复原它。

相关概念:

- 第 114 页的『使用装入副本位置文件』

相关参考:

- 第 117 页的『ROLLFORWARD DATABASE』

恢复删除的表

可能您偶然会删除仍需要其数据的表。如果是这样，您应该考虑在删除表操作后使关键的表成为可复原的。

可通过调用数据库复原操作恢复表数据，后跟一个前滚到删除表前的某时间点的数据库前滚操作。如果数据库很大，这可能会花很多时间，并使您的数据在恢复期间不可用。

DB2 删除的表恢复功能使您可使用表空间级的复原和前滚操作来恢复删除的表数据。这样可比数据库级的恢复要快，且您的数据库将对用户保持可用。

先决条件:

要使删除的表可以复原，必须对该表驻留的表空间启用 DROPPED TABLE RECOVERY 选项。这可以在表空间创建期间或通过调用 ALTER TABLESPACE 语句来完成。DROPPED TABLE RECOVERY 选项是表空间特定的，并限于对常规表空间使用。要确定是否对表空间启用了删除的表恢复，可以查询 SYSCAT.TABLESPACES 目录表中的 DROP_RECOVERY 列。缺省情况下，对新创建的数据表空间启用删除的表恢复。

对表（对该表的表空间启用了删除的表恢复）运行 DROP TABLE 语句时，将在日志文件中建立另一条目（标识删除的表）。还会在恢复历史文件中建立一个条目，包含可用于重新创建表的信息。

限制:

对可从删除的表中复原的数据类型有一些限制。不可能复原:

- 大对象（LOB）或长字段数据。对于大型表空间，不支持 DROPPED TABLE RECOVERY 选项。如果尝试复原包含 LOB 或 LONG VARCHAR 列的删除的表，这些列将在生成的导出文件中设置为 NULL。仅可对常规表空间使用 DROPPED TABLE RECOVERY 选项，而不能对临时或大型表空间使用。
- 与行类型相关的元数据。（数据已复原，但不是元数据。）将复原类型表的分层结构表中的数据。此数据包含的信息可能比已删除的类型表中出现的信息多。

过程:

一次只能复原一个删除的表。可执行下列操作来复原删除的表:

1. 通过调用 LIST HISTORY DROPPED TABLE 命令来标识删除的表。删除的表标识列示在“备份标识”列中。
2. 复原在删除该表前所建立的数据库级或表空间级备份映象。
3. 创建包含表数据的文件将写至的导出目录。该目录必须可访问所有数据库分区，或存在于每个分区上。此导出目录下的子目录是由每个数据库分区自动创建的。这些子目录的名称是 NODE*nnnn*，其中 *nnnn* 代表数据库分区或节点号。包含删除的表数据的数据文件（就如它存在于每个数据库分区上那样）将导出到称为 data 的较低子目录中。例如 \export_directory\NODE0000\data。
4. 删除表后前滚至某时间点，对 ROLLFORWARD DATABASE 命令使用 RECOVER DROPPED TABLE 选项。也可前滚至日志末尾，以使对表空间或数据库中的其它表进行的更新不会丢失。
5. 使用 CREATE TABLE 语句从恢复历史文件重新创建表。
6. 将在前滚操作期间导出的表数据导入表中。

*可以*复原与 DATALINK 列相关的链接文件的名称。导入表数据后，应使用 DB2 Data Links Manager 来协调该表。DB2 Data Links Manager 可能或可以复原这些文件的备份映象，也可能无法复原，这取决于无用信息收集是否已删除它们。

相关参考:

- 『ALTER TABLESPACE statement』（*SQL Reference, Volume 2*）
- 『CREATE TABLE statement』（*SQL Reference, Volume 2*）
- 第 117 页的『ROLLFORWARD DATABASE』
- 第 202 页的『LIST HISTORY』

使用装入副本位置文件

DB2LOADREC 注册表变量用于标识包含装入副本位置信息的文件。此文件在前滚恢复期间用于查找该装入副本。它包含以下信息:

- 媒体类型
- 要使用的媒体设备的数目
- 在表的装入操作期间生成的装入副本的位置
- 装入副本的文件名（若适用的话）

如果位置文件不存在，或文件中找不到匹配的条目，将使用日志记录中的信息。

在进行前滚恢复之前，该文件中的信息可能会被覆盖。

注:

1. 在分区数据库环境中，必须使用 **db2set** 命令对所有分区数据库服务器设置 DB2LOADREC 注册表变量。
2. 在一个分区数据库环境中，在每个数据库分区服务器中都必须存在该装入副本文件，且文件名（包括路径）必须相同。
3. 若 DB2LOADREC 注册表变量标识的文件中的一个条目无效，将使用旧的装入副本位置文件来提供替换无效条目的信息。

该位置文件提供下列信息。前五个参数必须具有有效值，它们用于标识装入副本。对于记录的每个装入副本，其整体结构是相同的。例如:

```
TIMestamp       19950725182542          * 装入时生成的时间戳记
SCHema          PAYROLL                 * 装入的表的模式
TABlename       EMPLOYEES               * 表名
DATabasename    DBT                     * 数据库名
DB2instance     TORONTO                 * DB2INSTANCE
BUFfernumber    NULL                    * 要用于恢复的缓冲区的数量
SESsionnumber   NULL                    * 要用于恢复的会话的数量
TYPeofmedia     L                       * 媒体的类型 - L 表示本地设备
```

```
                                                    A 表示 TSM
                                                    O 表示其它供应商
LOCationnumber 3                        * 位置数
   ENTry       /u/toronto/dbt.payroll.employes.001
   ENT         /u/toronto/dbt.payroll.employes.002
   ENT         /dev/rmt0
TIM            19950725192054
SCH            PAYROLL
TAB            DEPT
DAT            DBT
DB2®            TORONTO
SES            NULL
BUF            NULL
TYP            A
TIM            19940325192054
SCH            PAYROLL
TAB            DEPT
DAT            DBT
DB2            TORONTO
SES            NULL
BUF            NULL
TYP            O
SHRlib         /@sys/lib/backup_vendor.a
```

注:

1. 每个关键字的前三个字符很重要的。所有关键字必须以指定的次序排列。不接受任何空行。
2. 时间戳记的格式是 *yyyymmddhhmmss*。
3. 所有字段都是必需的，BUF 和 SES 除外（它们可以是 NULL）。如果 SES 为 NULL，将使用由 *numloadrecses* 配置参数指定的值。如果 BUF 为 NULL，缺省值是 SES+2。
4. 如果位置文件中即使只有一个条目是无效的，也会使用先前的装入副本位置文件来提供那些值。
5. 媒体类型可以是本地设备（L 指磁带、磁盘或软盘）、TSM（A）或其他供应商（O）。如果类型为 L，则需要位置数后跟位置条目。如果类型 A，不需要进一步的输入。如果类型是 O，需要共享库名。
6. SHRlib 参数指向一个库，该库具有存储装入副本数据的函数。
7. 如果调用装入操作时指定 COPY NO 或 NONRECOVERABLE 选项，并且在操作完成后不建立数据库或受影响的表空间的备份副本，不能将数据库或表空间复原到装入操作后的某时间点。即，不能使用前滚恢复将数据库或表空间重新构建到后面的装入操作中的状态。只能将数据库或表空间复原到装入操作前的某时间点。

如果要使用特定的装入副本，可使用数据库的恢复历史文件，以确定特定装入操作的时间戳记。在分区数据库环境中，恢复历史文件对每个数据库分区都是本地的。

相关参考:

- 第 293 页的附录 F, 『Tivoli Storage Manager』

使分区数据库系统中的时钟同步

应该使所有数据库分区服务器的系统时钟保持相对同步，以确保数据库操作顺利进行以及正向可复原性不受限制。数据库分区服务器之间的时间差加上一个事务的任何潜在的操作和通信延迟，应小于对 *max_time_diff*（节点之间的最大时间差）数据库管理器配置参数指定的值。

为确保日志记录时间戳记反映分区数据库系统中的事务的顺序，DB2® 使用使用每台机器上的系统时钟作为日志记录中时间戳记的基准。但是，若将系统时钟设置得提前，也自动将日志时钟设置得提前。虽然可以将系统时钟设置得落后，但是日志时钟却不能这样设置，它会保持*相同的*超前时间，直至系统时钟与此时间相匹配为止。于是，这两个时钟便同步了。这表示一个数据库节点上的短期系统时钟错误可能对数据库日志的时间戳记产生长期的影响。

例如，假定数据库分区服务器 A 上的系统时钟被错误地设置为 1999 年 11 月 7 日，而当前年份是 1997 年，并假定在数据库分区服务器上的分区中落实了更新事务之后将该错误校正。若继续使用该数据库，且过一段时间便定期更新它，则 1997 年 11 月 7 日至 1999 年 11 月 7 日之间的任何点实际上是不可通过前滚恢复达到的。当完成数据库分区服务器 A 上的 COMMIT 时，数据库日志中的时间戳记被设置为 1999，而日志的时钟会停留在 1999 年 11 月 7 日，直到系统时钟与此时间相匹配为止。若试图前滚至此时间范围内的一个时间点，则操作将在超过指定的停止点（即 1997 年 11 月 7 日）的第一个时间戳记处停止。

虽然 DB2 不能控制对系统时钟的更新，但是 *max_time_diff* 数据库管理器配置参数降低了发生这种类型的问题的机会:

- 此参数的可配置值的范围是 1 分钟至 24 小时。
- 当对非目录节点发出第一个连接请求时，数据库分区服务器会将它的时间发送至该数据库的目录节点。该目录节点就会检查请求该连接的节点上的时间以及它自己的时间是否在 *max_time_diff* 参数指定的范围之内。若超过此范围，则拒绝该连接。
- 涉及到数据库中两个以上数据库分区服务器的更新事务，必须先验证参与的数据库分区服务器上的时钟是否同步，然后才可落实该更新。若两个或多个数据

库分区服务器的时差超过了 *max_time_diff* 允许的限制，则会回滚该事务，以防止将不正确的时间传播至其它数据库分区服务器。

相关参考:

- 『 “节点间的最大时差”配置参数 — max_time_diff 』（ *《管理指南: 性能》* ）

客户机 / 服务器时间戳记转换

本节说明客户机 / 服务器环境中时间戳记的生成:

- 如果对前滚操作指定了本地时，则返回的所有消息也以本地时间表示。

  注: 在服务器上和（在分区数据库环境中）目录节点上所有时间都会转换。
- 在服务器上时间戳记字符串转换为 GMT，所以时间表示服务器的时区而不是客户机的时区。如果客户机与服务器在不同的时区，则应该使用服务器的本地时间。
- 如果时间戳记字符串接近由夏令时导致的时间更改，则了解停止时间是在时间更改之前还是之后是很重要的，这样才能正确地指定它。

相关概念:

- 第 105 页的『 前滚概述 』
- 第 116 页的『 使分区数据库系统中的时钟同步 』

ROLLFORWARD DATABASE

通过应用记录在数据库日志文件中的事务来恢复数据库。在数据库或表空间备份映象复原之后被调用，或如果由于媒体错误而导致数据库使任何表空间脱机。数据库必须是可恢复的（即，必须启用 *logretain* 和 / 或 *userexit* 这两个配置参数），才能前滚恢复数据库。

作用域:

在分区数据库环境中，仅能从目录分区调用此命令。至指定时间点的数据库或表空间前滚操作将影响 `db2nodes.cfg` 文件中列示的所有分区。至日志结尾的数据库或表空间前滚操作调用影响指定的分区。如果不指定任何分区，则它影响 `db2nodes.cfg` 文件中列示的所有分区；如果在特定分区上不需要前滚恢复，则忽略该分区。

授权:

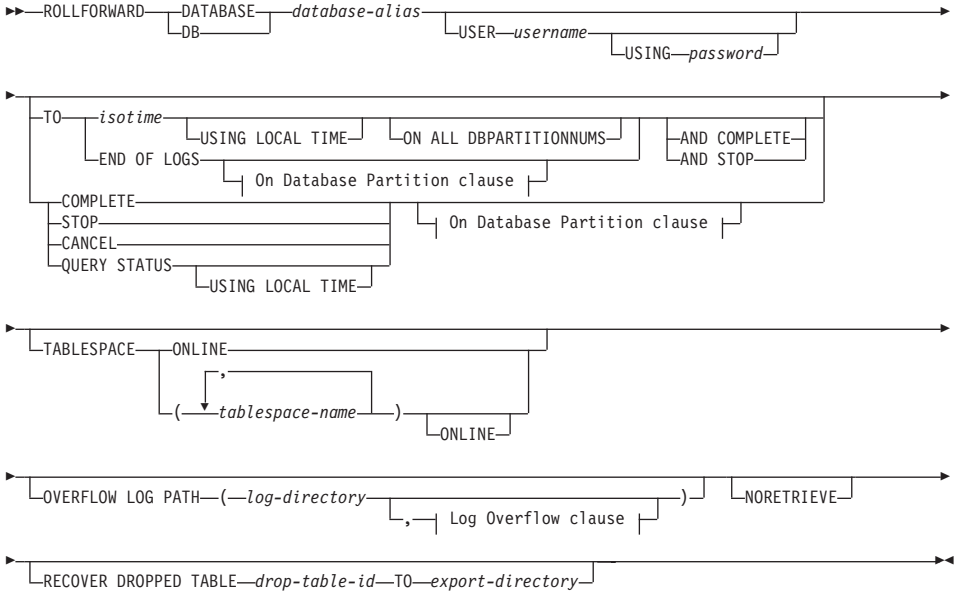
以下其中一项:

- *sysadm*
- *sysctrl*
- *sysmaint*

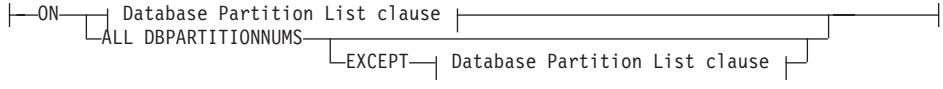
必需的连接:

无。此命令建立数据库连接。

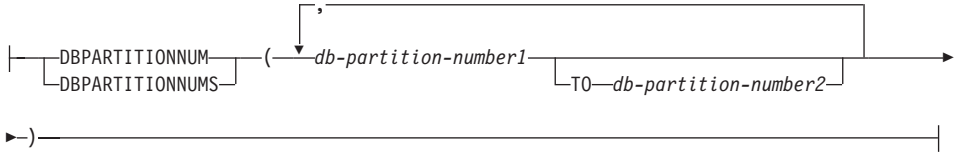
命令语法:



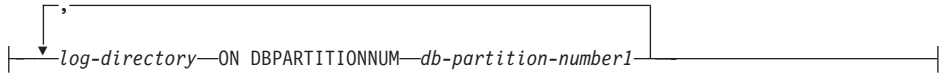
On Database Partition clause:



Database Partition List clause:



Log Overflow clause:



命令参数:

DATABASE database-alias
将对其进行前滚恢复的数据库别名。

USER username
对数据库执行前滚恢复时使用的用户名。

USING password
用于认证用户名的密码。如果省略密码，会提示用户输入。

TO

isotime
所有已落实的事务将前滚至的某个时间点（包括在该时间点落实的事务以及之前落实的事务）。

此值指定为时间戳记，即一个标识日期和时间的 7 部分字符串。格式为 *yyyy-mm-dd-hh.mm.ss.nnnnnn*（年、月、日、小时、分钟、秒和微秒），以“标准世界时”（UTC）表示。UTC 有助于避免存在与不同日志相关的相同时间戳记（例如，与夏时制相关的时间的更改）。备份映象中的时间戳记基于启动备份操作时的本地时间。CURRENT TIMEZONE 特殊注册器指定 UTC与应用程序服务器上的本地时间的不同。这种不同由一个持续时间来表示（它是一个十进制数，前两位表示小时数，接下来的两位表示分钟数而最后两位表示秒数。）从本地时间中减去 CURRENT TIMEZONE 可将本地时间转换为 UTC。

USING LOCAL TIME
允许用户前滚至是用户的当地时间而不是 GMT 时间的时间点。这使得用户更容易在其本地机器上前滚至特定时间点，并消除由于当地时间转换至 GMT 时间而导致的潜在用户错误。

注:

1. 如果对前滚操作指定了当地时间，则返回给用户的所有消息也是以当地时间表示的。注意，服务器上（如果为 MPP，则是在目录数据库分区上）的所有时间都会被转换。
2. 因为在服务器上时间戳记字符串被转换为 GMT，所以时间是服务器时区而不是客户机时区的当地时间。如果客户机与服务

器分别在不同时区，则应该使用服务器的当地时间。这不同于“控制中心”的当地时间选项，它相对于客户机来说是当地时间。

3. 如果时间戳记字符串与由于夏令时导致的时钟的时间更改紧密相关，则应了解停止时间是在时钟更改之前还是时钟更改之后，并正确指定它。

END OF LOGS

指定将应用来自数据库配置参数 *logpath* 中列示的所有联机归档日志文件的所有已落实事务。

ALL DBPARTITIONNUMS

指定事务将在 db2nodes.cfg 文件中指定的所有分区上前滚。如果不指定数据库分区子句，则这是缺省值。

EXCEPT

指定事务将在 db2nodes.cfg 文件中指定的所有分区上前滚，在数据库分区列表中指定的分区除外。

ON DBPARTITIONNUM / ON DBPARTITIONNUMS

在一组数据库分区上前滚数据库。

db-partition-number1

在数据库分区列表中指定数据库分区号。

db-partition-number2

指定第二个数据库分区号，以便从 *db-partition-number1* 直到并且包括 *db-partition-number2* 的所有分区都包括在数据库分区列表中。

COMPLETE / STOP

停止前滚日志记录，通过回滚所有未完成的事务并结束数据库的前滚暂挂状态，来完成前滚恢复进程。因此可访问正在前滚的数据库或表空间。这两个关键字是等效的，应指定其中一个而不应同时指定两个。关键字 AND 允许一次指定多个操作，例如 db2 rollforward db sample to end of logs and complete。

注: 将表空间前滚到某时间点时，表空间将被置于备份暂挂状态。

CANCEL

取消前滚恢复操作。这会将启动了正向恢复的所有分区上的数据库或者一个或多个表空间置于复原暂挂状态:

- 如果*数据库*前滚操作不在进行中（即数据库处于前滚暂挂状态），此选项会将数据库置于复原暂挂状态。

- 如果*表空间*前滚操作不在进行中（即表空间处于前滚暂挂状态），则必须指定表空间列表。列表中的所有表空间都将处于复原暂挂状态。
- 如果表空间前滚操作*正在*进行中（即至少有一个表空间处于“前滚进行中”状态），处于“前滚进行中”状态的所有表空间都将置于复原暂挂状态。如果指定了表空间列表，该列表必须包括处于“前滚进行中”状态的所有表空间。该列表上的所有表空间都将置于复原暂挂状态。
- 如果前滚至某时间点，则传入的任何表空间名都将被忽略，且处于“前滚进行中”状态的所有表空间都将置于复原暂挂状态。
- 如果使用表空间列表前滚至日志末尾，只有列示的表空间将置于复原暂挂状态。

此选项不能用于取消*实际已在运行的*前滚操作。它只能用于取消已在进行中，但此时并未实际运行的前滚操作。前滚操作可在进行中，但并未运行，如果:

- 它异常终止。
- 未指定 STOP 选项。
- 错误导致前滚操作失败。某些错误，例如对不可恢复的装入操作前滚，会使表空间处于复原暂挂状态。

注: 使用此选项时应小心，只有在因为某些表空间已处于前滚暂挂状态或复原暂挂状态，已在进行中的前滚操作才不能完成时使用该选项。如果对状态有疑问，可使用 LIST TABLESPACES 命令来判断表空间是处于“前滚进行中”状态还是前滚暂挂状态。

QUERY STATUS

列示数据库管理器已前滚的日志文件、所需的下一个归档文件，以及自从前滚处理开始以来最后一个落实的事务的时间戳记（格式为 CUT）。在分区数据库环境中，将对每个分区返回此状态信息。返回的信息包含以下字段:

数据库分区号

前滚状态

状态可以是: 数据库或表空间前滚暂挂、数据库或表空间前滚进行中、数据库或表空间前滚处理 STOP（停止）或未暂挂。

要读取的下一个日志文件

包含需要的下一日志文件名的字符串。在分区数据库环境中，如果前滚实用程序失败并有返回码指出: 日志文件丢失或已发生了日志信息的不匹配，则使用此信息。

处理的日志文件

包含已处理的日志文件名的字符串，恢复操作不再需要这些文件，因此可从目录中除去。例如，如果最旧的未落实事务是在日志文件 *x* 中开始的，则过时日志文件的范围将不包括 *x*；且范围在 *x* - 1 处结束。

Last committed transaction

包含格式为 ISO 的时间戳记 (*yyyy-mm-dd-hh.mm.ss*) 的字符串。该时间戳记标记完成前滚恢复后，最后一个已落实的事务。时间戳记适用于数据库。对于表空间前滚恢复，它是对数据库落实的最后一个事务的时间戳记。

注: 如果省略 TO、STOP、COMPLETE 或 CANCEL 子句，QUERY STATUS 将是缺省值。如果指定了 TO、STOP 或 COMPLETE，将在命令成功完成时显示状态信息。如果指定了个别表空间，将忽略它们；状态请求不能只对指定的表空间应用。

TABLESPACE

对表空间级的前滚恢复指定此关键字。

tablespace-name

对于表空间级的前滚恢复至某时间点，此选项是必需的。它允许为前滚恢复到日志末尾指定表空间的子集。在分区数据库环境中，正在前滚的每个分区上，列表中的每个表空间不必都存在。如果它*存在*，就必须处于正确的状态。

ONLINE

指定此关键字，以允许联机完成表空间级的前滚恢复。这意味着正在进行前滚恢复时，允许另一代理进程连接。

OVERFLOW LOG PATH log-directory

指定在恢复期间，将在其中搜索归档日志的备用日志路径。如果日志文件移动到由 *logpath* 数据库配置参数指定的位置以外的位置，则使用此参数。在分区数据库环境中，这是*所有分区的*（全限定）缺省溢出日志路径。可对单分区数据库指定相对溢出日志路径。

注: OVERFLOW LOG PATH 命令参数将覆盖数据库配置参数 OVERFLOWLOGPATH 的值（如果有的话）。

log-directory ON DBPARTITIONNUM

在分区数据库环境中，允许不同的日志路径覆盖特定分区的缺省溢出日志路径。

NORETRIEVE

通过允许用户禁用归档日志的检索来控制在备用机器上要前滚哪些日志文件。此操作的优点是:

- 通过控制要前滚的日志文件，可以确保备用机器比生产机器滞后 X 小时以防止用户同时影响两个系统。
- 如果备用系统不具有归档的存取权（例如，如果 TSM 是归档文件，它只允许原始机器检索文件）
- 当生产系统正在归档文件，而备用系统在检索同一文件时，它还有可能会获取不完整的日志文件。不进行任何检索就可以解决此问题。

RECOVER DROPPED TABLE drop-table-id

恢复前滚操作期间删除的表。可以使用 LIST HISTORY 命令来获取表标识。

TO export-directory

指定包含表数据的文件将被写至的目录。该目录必须对所有数据库分区都是可存取的。

示例:

示例 1

ROLLFORWARD DATABASE 命令一次允许多个操作的规范，每个规范都由关键字 AND 隔开。例如，要前滚至日志末尾，然后完成，可将以下单独的命令

```
db2 rollforward db sample to end of logs
db2 rollforward db sample complete
```

组合为:

```
db2 rollforward db sample to end of logs and complete
```

虽然两种方式是等效的，但建议您以两个步骤来完成此类操作。非常重要的是，应验证前滚操作是否在按预期的方式执行，免得需要停止该操作从而可能导致日志丢失。如果在前滚恢复期间发现了错误的日志，而该错误的日志却被解释为“日志结束”，上述验证尤其重要。此时，可使用该日志的一个未损坏的备份副本来继续对更多的日志进行前滚操作。

示例 2

前滚至日志末尾（已复原了两个表空间）:

```
db2 rollforward db sample to end of logs
db2 rollforward db sample to end of logs and stop
```

这两个语句是等效的。需要 AND STOP 或 AND COMPLETE 以使表空间前滚恢复到日志末尾。不需要表空间名称。如果未指定的话，将包括所有需要前滚恢复的表空间。如果将只前滚这些表空间的一个子集，则必须指定它们的名称。

示例 3

已复原了 3 个表空间后，将其中一个前滚到日志末尾，另两个前滚到某时间点，全部联机进行:

```
db2 rollforward db sample to end of logs tablespace(TBS1) online

db2 rollforward db sample to 1998-04-03-14.21.56.245378 and stop
   tablespace(TBS2, TBS3) online
```

应注意，两个前滚操作不能并行运行。只有在成功地完成了第一个前滚操作后，才能调用第二个命令。

示例 4

复原数据库后，前滚到某时间点，使用 OVERFLOW LOG PATH 来指定用户出口保存归档日志的目录:

```
db2 rollforward db sample to 1998-04-03-14.21.56.245378 and stop
   overflow log path (/logs)
```

示例 5（MPP）

有 3 个数据库分区: 0、1 和 2。在所有分区上定义表空间 TBS1，在节点 0 和 2 上定义表空间 TBS2。在数据库分区 1 上复原了数据库，并在数据库分区 0 和 2 上复原了 TBS1 之后，在数据库分区 1 上前滚数据库:

```
db2 rollforward db sample to end of logs and stop
```

这将返回警告 SQL1271（“数据库已恢复，但数据库分区 0 和 2 上的一个或多个表空间已脱机。”）。

```
db2 rollforward db sample to end of logs
```

此命令在数据库分区 1 和 2 上前滚 TBS1。在这种情况中，子句 TABLESPACE(TBS1) 是可选的。

示例 6（MPP）

只在数据库分区 0 和 2 上复原表空间 TBS1 之后，在数据库分区 0 和 2 上前滚 TBS1:

```
db2 rollforward db sample to end of logs
```

忽略数据库分区 1。

```
db2 rollforward db sample to end of logs tablespace(TBS1)
```

此命令失败，因为 TBS1 未对在数据库分区 1 上进行前滚恢复作好准备。报告 SQL4906N。

```
db2 rollforward db sample to end of logs on dbpartitionnums (0, 2)
   tablespace(TBS1)
```

成功完成。

```
db2 rollforward db sample to 1998-04-03-14.21.56.245378 and stop
   tablespace(TBS1)
```

此命令失败，因为 TBS1 未对在数据库分区 1 上进行前滚恢复作好准备；必须将所有部分前滚到一起。

注: 随着表空间前滚至某个时间点，将不接受该数据库分区子句。前滚操作必须在表空间所驻留的所有数据库分区上进行。

在数据库分区 1 上复原 TBS1 后:

```
db2 rollforward db sample to 1998-04-03-14.21.56.245378 and stop
   tablespace(TBS1)
```

成功完成。

示例 7（分区数据库环境）

在所有数据库分区上复原表空间后前滚至 PIT2，但不指定 AND STOP。前滚操作仍在进行中。取消并前滚至 PIT1:

```
db2 rollforward db sample to pit2 tablespace(TBS1)
db2 rollforward db sample cancel tablespace(TBS1)

** restore TBS1 on all database partitions **

db2 rollforward db sample to pit1 tablespace(TBS1)
db2 rollforward db sample stop tablespace(TBS1)
```

示例 8（MPP）

前滚恢复驻留在 db2nodes.cfg 文件中列示的 8 个数据库分区（3 至 10）上的表空间:

```
db2 rollforward database dwtest to end of logs tablespace (tssprodt)
```

前滚恢复至日志末尾（而不是时间点）这一操作成功完成。不必指定表空间所驻留的数据库分区。实用程序缺省到 db2nodes.cfg 文件。

示例 9（分区数据库环境）

前滚恢复驻留在单数据库分区数据库分区组（在数据库分区 6 上）上的 6 个小表空间:

```
db2 rollforward database dwtest to end of logs on dbpartitionnum (6)
   tablespace(tsstore, tssbuyer, tsstime, tsswhse, tsslscat, tssvendor)
```

前滚恢复至日志末尾（而不是时间点）这一操作成功完成。

使用注意事项:

如果从联机备份操作期间创建的映象进行复原，则前滚操作的指定时间点必须迟于联机备份操作完成的时间。如果前滚操作在未经过此点时就停止了，则数据库保留在前滚暂挂状态。如果表空间正在前滚，则它保留正在前滚状态。

如果一个或多个表空间正前滚至某个时间点，则前滚操作必须持续至少直至最小恢复时间，它是对此表空间或其表的系统目录的最近更新。表空间的最小恢复时间（以 “标准世界时” 或 UTC 表示）可以通过使用 LIST TABLESPACES SHOW DETAIL 命令进行检索。

前滚数据库可能需要使用磁带机进行装入恢复。如果提示输入另一磁带，用户可以使用下列其中之一进行响应:

c 继续 — 继续使用生成了警告消息的设备（例如，已安装了新磁带时）

d 设备终止 — 停止使用生成警告消息（例如，没有磁带了）的设备

t 终止 — 终止所有设备。

如果前滚实用程序找不到它需要的下一日志，将在 SQLCA 中返回日志名，而前滚恢复会结束。如果没有更多日志可用，则使用 STOP 选项来终止前滚恢复。将回滚未完成的事务，以确保数据库或表空间处于一致状态。

兼容性:

对于与版本 8 之前的版本的兼容性:

- 可以用关键字 NODE 来替代 DBPARTITIONNUM。
- 可以用关键字 NODES 来替代 DBPARTITIONNUMS。

相关参考:

- 第 63 页的『BACKUP DATABASE』
- 第 84 页的『RESTORE DATABASE』

db2Rollforward — 前滚数据库

通过应用记录在数据库日志文件中的事务来恢复数据库。在数据库或表空间备份复原之后被调用，或如果由于媒体错误而导致数据库使任何表空间脱机。数据库必须是可恢复的（即，*logretain* 和／或 *userexit* 这两个配置参数必须设置为打开），才能使用前滚恢复来恢复数据库。

作用域:

在分区数据库环境中，仅能从目录分区调用此 API。指定时间点的数据库或表空间前滚调用影响 `db2nodes.cfg` 文件中列示的所有数据库分区服务器。指定日志结尾的数据库或表空间前滚调用影响被指定的数据库分区服务器。如果未指定任何数据库分区服务器，则它影响 `db2nodes.cfg` 文件中列示的所有数据库分区服务器；如果特定数据库分区服务器上不需要任何前滚，则该数据库分区服务器被忽略。

授权:

以下其中一项:

- *sysadm*
- *sysctrl*
- *sysmaint*

必需的连接:

无。此 API 建立数据库连接。

API 包含文件:

db2ApiDf.h

C API 语法:

```
/* File: db2ApiDf.h */
/* API: db2Rollforward */
/* ... */
SQL_API_RC SQL_API_FN
db2Rollforward_api (
  db2Uint32 versionNumber,
  void *pDB2RollforwardStruct,
      struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2RollforwardStruct
{
  struct db2RfwdInputStruct   *roll_input;
  struct db2RfwdOutputStruct  *roll_output;
} db2RollforwardStruct;

typedef SQL_STRUCTURE db2RfwdInputStruct
{
  sqluint32              version;
  char                   *pDbAlias;
  db2Uint32                   CallerAction;
  char                   *pStopTime;
  char                   *pUserName;
  char                   *pPassword;
  char                   *pOverflowLogPath;
  db2Uint32                   NumChngLgOvrflw;
  struct sqlurf_newlogpath *pChngLogOvrflw;
  db2Uint32                   ConnectMode;
  struct sqlu_tablespace_bkrst_list *pTablespaceList;
  db2int32                    AllNodeFlag;
  db2int32                    NumNodes;
  SQL_PDB_NODE_TYPE      *pNodeList;
  db2int32                    NumNodeInfo;
  char                   *pDroppedTblID;
  char                   *pExportDir;
  db2Uint32                   RollforwardFlags;
} db2RfwdInputStruct;

typedef SQL_STRUCTURE db2RfwdOutputStruct
{
  char            *pApplicationId;
  sqlint32                    *pNumReplies;
  struct sqlurf_info *pNodeInfo;
} db2RfwdOutputStruct;
/* ... */
```

一般 API 语法:

```
/* File: db2ApiDf.h */
/* API: db2Rollforward */
/* ... */
SQL_API_RC SQL_API_FN
db2gRollforward_api (
  db2Uint32 versionNumber,
  void *pDB2gRollforwardStruct,
      struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2gRollforwardStruct
{
  struct db2gRfwdInputStruct  *roll_input;
  struct db2RfwdOutputStruct  *roll_output;
} db2gRollforwardStruct;

SQL_STRUCTURE db2gRfwdInputStruct
{
  db2Uint32                    DbAliasLen;
  db2Uint32                    StopTimeLen;
  db2Uint32                    UserNameLen;
  db2Uint32                    PasswordLen;
  db2Uint32                    OvrflwLogPathLen;
  db2Uint32                    DroppedTblIDLen;
  db2Uint32                    ExportDirLen;
  sqluint32                    Version;
  char                    *pDbAlias;
  db2Uint32                    CallerAction;
  char                    *pStopTime;
  char                    *pUserName;
  char                    *pPassword;
  char                    *pOverflowLogPath;
  db2Uint32                    NumChngLgOvrflw;
  struct sqlurf_newlogpath *pChngLogOvrflw;
  db2Uint32                    ConnectMode;
  struct sqlu_tablespace_bkrst_list *pTablespaceList;
  db2int32                     AllNodeFlag;
  db2int32                     NumNodes;
  SQL_PDB_NODE_TYPE       *pNodeList;
  db2int32                     NumNodeInfo;
  char                    *pDroppedTblID;
  char                    *pExportDir;
  db2Uint32                    RollforwardFlags;
};

typedef SQL_STRUCTURE db2RfwdOutputStruct
{
  char             *pApplicationId;
  sqlint32                     *pNumReplies;
  struct sqlurf_info *pNodeInfo;
} db2RfwdOutputStruct;
/* ... */
```

API 参数:

versionNumber
输入。指定作为第二个参数传送的结构版本和发行版级别。

pDB2RollforwardStruct
输入。指向 *db2RollforwardStruct* 结构的指针。

pSqlca
输出。指向 *sqlca* 结构的指针。

roll_input
输入。指向 *db2RfwdInputStruct* 结构的指针。

roll_output
输出。指向 *db2RfwdOutputStruct* 结构的指针。

DbAliasLen
输入。指定数据库别名的长度（以字节计）。

StopTimeLen
输入。指定停止时间参数的长度（以字节计）。如果不提供停止时间，则设置为零。

UserNameLen
输入。指定用户名的长度（以字节计）。如果不提供用户名，则设置为零。

PasswordLen
输入。指定密码的长度（以字节计）。如果不提供密码，则设置为零。

OverflowLogPathLen
输入。指定溢出日志路径的长度（以字节计）。如果不提供溢出日志路径，则设置为零。

Version
输入。前滚参数的版本标识。定义为 SQLUM_RFWD_VERSION。

pDbAlias
输入。包含了数据库别名的字符串。它是编目在系统数据库目录中的别名。

CallerAction
输入。指定要采取的操作。有效值（在 `sqlutil` 中定义）是:

SQLUM_ROLLFWD
前滚至由 *pPointInTime* 指定的时间点。对于数据库前滚，数据库将保持*前滚暂挂*状态。对于表空间前滚至时间点，表空间将保持*正在前滚*状态。

SQLUM_STOP
结束前滚恢复。不处理新日志记录并且不能完成未落实的事务。将取消数据库或表空间的*前滚暂挂*状态。等效参数是 SQLUM_COMPLETE。

SQLUM_ROLLFWD_STOP
前滚至由 *pPointInTime* 指定的时间点，并结束前滚恢复。将取消数据库或表空间的*前滚暂挂*状态。等效参数是 SQLUM_ROLLFWD_COMPLETE。

SQLUM_QUERY
pNextArcFileName、*pFirstDelArcFileName*、*pLastDelArcFileName* 和 *pLastCommitTime* 的查询值。返回数据库状态和节点号。

SQLUM_PARM_CHECK
验证参数而不执行前滚。

SQLUM_CANCEL
取消当前正在运行的前滚操作。数据库或表空间被置于恢复暂挂状态。

注: 前滚实际上在运行时不能使用此选项。如果前滚暂停（即等待 STOP），或在前滚期间发生了系统故障，则可以使用该选项。使用时应小心。

前滚数据库可能需要使用磁带机进行装入恢复。如果需要用户对设备进行干预，则前滚 API 将返回一条警告消息。可以使用下列三个调用程序操作中的一个以再次调用 API:

SQLUM_LOADREC_CONTINUE
继续使用生成了警告消息的设备（例如在安装了新磁带后）。

SQLUM_LOADREC_DEVICE_TERMINATE
停止使用生成了警告消息的设备（例如没有更多磁带时）。

SQLUM_LOADREC_TERMINATE
终止正由装入恢复使用的所有设备。

pStopTime
输入。包含格式为 ISO 的时间戳记的字符串。超出此时间戳记时，数据库恢复将停止。指定 SQLUM_INFINITY_TIMESTAMP 以尽可能多地前滚。对于 SQLUM_QUERY、SQLUM_PARM_CHECK 以及任何装入恢复（SQLUM_LOADREC_*xxx*）调用程序操作可能为 NULL。

pUserName
输入。包含应用程序用户名的字符串。可能为 NULL。

pPassword

输入。包含提供的用户名（如果有的话）密码的字符串。可能为 NULL。

pOverflowLogPath

输入。此参数用于指定要使用的备用日志路径。除活动日志文件以外，归档日志文件需要（由用户）移至 *logpath* 中，此实用程序才能使用它们。如果用户在 *logpath* 中没有足够的空间，可能会导致问题。因此提供了溢出日志路径。在前滚恢复期间，将首先在 *logpath* 中，然后在溢出日志路径中搜索需要的日志文件。表空间前滚恢复需要的日志文件可放在 *logpath* 或溢出日志路径中。如果调用程序不指定溢出日志路径，缺省值将是 *logpath*。在分区数据库环境中，溢出日志路径必须是有效的全限定路径，而缺省路径是每个节点的缺省溢出日志路径。在单分区数据库环境中，如果服务器是本地的，则溢出日志路径可以是相对的。

NumChngLgOvrflw

仅限于分区数据库环境。更改的溢出日志路径数。这些新的日志路径只覆盖指定数据库分区服务器的缺省溢出日志路径。

pChngLogOvrflw

仅限于分区数据库环境。指向一个结构的指针，该结构包含已更改的溢出日志路径的全限定名。这些新的日志路径只覆盖指定数据库分区服务器的缺省溢出日志路径。

ConnectMode

输入。有效值（在 `sqlutil` 中定义）是:

SQLUM_OFFLINE

脱机前滚。必须对数据库前滚恢复指定此值。

SQLUM_ONLINE

联机前滚。

pTablespaceList

输入。指向一个结构的指针，该结构包含将要前滚至日志末尾或特定时间点的表空间的名称。如果未指定，将选择需要前滚的表空间。

AllNodeFlag

仅限于分区数据库环境。输入。指示是否将前滚操作应用于 `db2nodes.cfg` 中定义的所有数据库分区服务器。有效的值为:

SQLURF_NODE_LIST

应用于 *pNodeList* 中传送的列表中的数据库分区服务器。

SQLURF_ALL_NODES

应用于所有数据库分区服务器。*pNodeList* 应为 NULL。这是缺省值。

SQLURF_ALL_EXCEPT

应用于除 *pNodeList* 中传送的列表中列示的数据库分区服务器之外的所有数据库分区服务器。

SQLURF_CAT_NODE_ONLY

仅应用于目录分区。*pNodeList* 应为 NULL。

NumNodes

输入。指定 *pNodeList* 数组中的数据库分区服务器的数目。

pNodeList

输入。指向要对其执行前滚恢复的数据库分区服务器号数组的指针。

NumNodeInfo

输入。定义输出参数 *pNodeInfo* 的大小，它必须大到足够容纳正在前滚的每个数据库分区的状态信息。在单分区数据库环境中，此参数应设置为 1。此参数的值应与正对其调用此 API 的数据库分区服务器数目相同。

pDroppedTblID

输入。字符串，它包含正对其尝试恢复的已删除的表的标识。

pExportDir

输入。已删除的表数据将导出至其中的目录。

RollforwardFlags

输入。指定前滚标志。有效值（在 `sqlpapiRollforward` 中定义）：

SQLP_ROLLFORWARD_LOCAL_TIME

允许用户前滚至是用户的当地时间而不是 GMT 时间的时间点。这使得用户更容易在其本地机器上前滚至特定时间点，并消除由于当地时间转换至 GMT 时间而导致的潜在用户错误。

SQLP_ROLLFORWARD_NO_RETRIEVE

通过允许用户禁用归档日志的检索来控制要在备用机器上前滚哪些日志文件。通过控制要前滚的日志文件，可以确保备用机器比生产机器滞后 X 小时以防止用户同时影响两个系统。如果备用系统不具有归档文件的存取权（例如，如果 TSM 是归档文件，它只允许原始机器检索这些文件），此选项会很有用。在生产系统在归档文件而备用系统在检索同一文件时，它还可以防止备用系统检索不完整的日志文件。

pApplicationId

输出。应用程序标识。

pNumReplies

输出。接收到的应答数。

pNodeInfo

输出。数据库分区应答信息。

REXX API 语法:

```
ROLLFORWARD DATABASE database-alias [USING :value] [USER username
USING password]
[rollforward_action_clause | load_recovery_action_clause]
其中 rollforward_action_clause 表示:      { TO point-in-time [AND STOP] |
      {
        [TO END OF LOGS [AND STOP] | STOP | CANCEL | QUERY STATUS
        | PARM CHECK }
        [ON {:nodelist | ALL NODES [EXCEPT :nodelist]}]
      }
    }
    [TABLESPACE {ONLINE |:tablespacenames [ONLINE]} ]
    [OVERFLOW LOG PATH default-log-path [:logpaths]]
而 load_recovery_action_clause 表示:
LOAD RECOVERY { CONTINUE | DEVICE_TERMINATE | TERMINATE }
```

REXX API 参数:

database-alias

要前滚的数据库的别名。

value 包含输出值的组合 REXX 主机变量。在下例中，XXX 表示主机变量名:

XXX.0	变量中的元素号
XXX.1	应用程序标识
XXX.2	从节点接收到的应答数
XXX.2.1.1	第一个数据库分区服务器号
XXX.2.1.2	第一个状态信息
XXX.2.1.3	第一个需要的下一归档文件
XXX.2.1.4	第一个将删除的第一归档文件
XXX.2.1.5	第一个将删除的最后归档文件
XXX.2.1.6	第一个最后落实时间
XXX.2.2.1	第二个数据库分区服务器号
XXX.2.2.2	第二个状态信息
XXX.2.2.3	第二个所需的下一归档文件
XXX.2.2.4	第二个将删除的第一归档文件
XXX.2.2.5	第二个将删除的最后归档文件

XXX.2.2.6 第二个最后落实时间

XXX.2.3.x 以此类推。

username

标识前滚数据库时要使用的用户名。

password

用于认证用户名的密码。

时间点 格式为 ISO 的时间戳记 *yyyy-mm-dd-hh.mm.ss.nnnnnn*（年、月、日、小时、分钟、秒和微秒），以“标准世界时”(UTC) 表示。

tablespacenames

包含了要前滚的表空间列表的组合 REXX 主机变量。在以下示例中 XXX 是主机变量的名称:

XXX.0 要前滚的表空间号

XXX.1 第一个表空间的名称

XXX.2 第二个表空间的名称

XXX.x 以此类推。

default-log-path

在恢复期间，将在其中搜索归档日志的缺省溢出日志路径。

logpaths

组合 REXX 主机变量，包含了要在恢复期间在其中搜索归档日志的备用日志路径的列表。在以下示例中 XXX 是主机变量的名称:

XXX.0 更改的溢出日志路径号

XXX.1.1 第一个节点

XXX.1.2 第一个溢出日志路径

XXX.2.1 第二个节点

XXX.2.2 第二个溢出日志路径

XXX.3.1 第三个节点

XXX.3.2 第三个溢出日志路径

XXX.x.1 以此类推。

nodelist

包含数据库分区服务器的列表的复合 REXX 主机变量。在以下示例中 XXX 是主机变量的名称:

XXX.0 节点号

XXX.1 第一个节点

XXX.2 第二个节点

XXX.x 以此类推。

使用注意事项:

数据库管理器 使用存储在归档日志文件和活动日志文件中的信息来重构自上次备份以来对数据库执行的事务。

调用此 API 时执行的操作取决于调用之前数据库的 *rollforward_pending* 标志。这可以通过使用 “db2CfgGet — 获取配置参数” 来查询。如果数据库处于前滚暂挂状态，则 *rollforward_pending* 标志设置为 DATABASE。如果一个或多个表空间处于 SQLB_ROLLFORWARD_PENDING 或 SQLB_ROLLFORWARD_IN_PROGRESS 状态，则它设置为 TABLESPACE。如果数据库和任何表空间都不需要前滚，则 *rollforward_pending* 标志设置为 NO。

如果调用此 API 时数据库处于前滚暂挂状态，则将前滚该数据库。在成功进行数据库前滚之后，表空间返回正常状态，除非异常状态导致一个或多个表空间脱机。如果 *rollforward_pending* 标志设置为 TABLESPACE，则仅将前滚处于前滚暂挂状态或按名称请求的表空间。

注: 如果表空间前滚异常终止，则正在前滚的表空间将被置于 SQLB_ROLLFORWARD_IN_PROGRESS 状态。在下一次调用 ROLLFORWARD DATABASE 时，将只处理处于 SQLB_ROLLFORWARD_IN_PROGRESS 状态的表空间。如果所选表空间名的集合不包括处于 SQLB_ROLLFORWARD_IN_PROGRESS 状态的所有表空间，则将不需要的表空间置于 SQLB_RESTORE_PENDING 状态。

如果数据库未处于前滚暂挂状态，且未指定任何时间点，则处于正在前滚状态的所有表空间都将前滚至日志结束。如果没有任何表空间处于正在前滚状态，则处于前滚暂挂状态的所有表空间都将前滚至日志结束。

此 API 读取日志文件，从与备份映象匹配的日志文件开始。可以通过在前滚任何日志文件之前使用调用程序操作 SQLUM_QUERY 来调用此 API 以确定此日志文件的名称。

包含在日志文件中的事务被重新应用于数据库。对日志进行处理直到信息可用为止，或直到停止时间参数指定的时间为止。

当发生下列任何一项事件时恢复会停止:

- 找不到其它日志文件
- 日志文件中的时间戳记超过由停止时间参数指定的完成时间戳记

- 读取日志文件时发生错误。

某些事务可能不会恢复。*pLastCommitTime* 中返回的值指示应用于数据库的上一次落实事务的时间戳记。

如果由于应用程序或人为错误而导致需要数据库恢复，则用户可能想要在 *pStopTime* 中提供时间戳记值，以指示恢复应停止在发生该错误的时间之前。这仅适用于完全数据库前滚恢复和表空间前滚至某时间点。它还允许恢复停止在日志读取错误发生之前，这是在先前恢复失败时确定的。

当 *rollforward_recovery* 标志设置为 DATABASE 时，在前滚恢复终止之前，数据库不可用。终止是通过使用调用程序操作 SQLUM_STOP 或 SQLUM_ROLLFORWARD_STOP 来调用 API 以使数据库脱离前滚暂挂状态完成的。如果 *rollforward_recovery* 标志为 TABLESPACE，则数据库可供使用。但是，在调用 API 执行表空间前滚恢复之前，处于 SQLB_ROLLFORWARD_PENDING 和 SQLB_ROLLFORWARD_IN_PROGRESS 状态的表空间将不可用。如果将表空间前滚至某时间点，则在成功前滚之后表空间将置于备份暂挂状态。

当 *RollforwardFlags* 选项设置为 SQLP_ROLLFORWARD_LOCAL_TIME 时，返回至用户的所有消息也是以当地时间表示的。在服务器和目录分区（如果它是分区数据库环境的话）上，将转换全部时间。因为在服务器上时间戳记字符串被转换为 GMT，所以时间是服务器时区而不是客户机时区的当地时间。如果客户机与服务器分别在不同时区，则应该使用服务器的当地时间。这不同于“控制中心”的当地时间选项，它相对于客户机来说是当地时间。如果时间戳记字符串与由于夏令时导致的时钟的时间更改紧密相关，则应了解停止时间是在时钟更改之前还是时钟更改之后，并正确指定它。

相关参考:

- 『SQLCA』（*Administrative API Reference*）

相关样本:

- 『dbrecov.sqc -- How to recover a database (C)』
- 『dbrecov.sqC -- How to recover a database (C++)』

前滚会话 — CLP 示例

示例 1

ROLLFORWARD DATABASE 命令一次允许多个操作的规范，每个规范都由关键字 AND 隔开。例如，要前滚至日志末尾，然后完成，可将以下单独的命令

```
db2 rollforward db sample to end of logs
db2 rollforward db sample complete
```

组合为:

```
db2 rollforward db sample to end of logs and complete
```

虽然两种方式是等效的，但建议您以两个步骤来完成此类操作。非常重要的是，应验证前滚操作是否在按预期的方式执行，免得需要停止该操作从而可能导致日志丢失。如果在前滚恢复期间发现了错误的日志，而该错误的日志却被解释为“日志结束”，上述验证尤其重要。此时，可使用该日志的一个未损坏的备份副本来继续对更多的日志进行前滚操作。

示例 2

前滚至日志末尾（已复原了两个表空间）:

```
db2 rollforward db sample to end of logs
db2 rollforward db sample to end of logs and stop
```

这两个语句是等效的。需要 AND STOP 或 AND COMPLETE 以使表空间前滚恢复到日志末尾。不需要表空间名称。如果未指定的话，将包括所有需要前滚恢复的表空间。如果将只前滚这些表空间的一个子集，则必须指定它们的名称。

示例 3

已复原了 3 个表空间后，将其中一个前滚到日志末尾，另两个前滚到某时间点，全部联机进行:

```
db2 rollforward db sample to end of logs tablespace(TBS1) online

db2 rollforward db sample to 1998-04-03-14.21.56.245378 and stop
   tablespace(TBS2, TBS3) online
```

应注意，两个前滚操作不能并行运行。只有在成功地完成了第一个前滚操作后，才能调用第二个命令。

示例 4

复原数据库后，前滚到某时间点，使用 OVERFLOW LOG PATH 来指定用户出口保存归档日志的目录:

```
db2 rollforward db sample to 1998-04-03-14.21.56.245378 and stop
   overflow log path (/logs)
```

示例 5 (MPP)

有 3 个节点: 0、1 和 2。在所有节点上定义表空间 TBS1，在节点 0 和 2 上定义表空间 TBS2。在节点 1 复原了数据库并在 0 和 2 上复原了 TBS1 后，在节点 1 上前滚数据库:

```
db2 rollforward db sample to end of logs and stop
```

这会返回警告 SQL1271（数据库已恢复，但节点 0 和 2 上的一个或多个表空间已脱机）。

```
db2 rollforward db sample to end of logs
```

该命令在节点 0 和 2 上前滚 TBS1。在这种情况中，子句 TABLESPACE(TBS1) 是可选的。

示例 6 (MPP)

只在节点 0 和 2 上复原表空间 TBS1 后，在节点 0 和 2 上前滚 TBS1:

```
db2 rollforward db sample to end of logs
```

忽略节点 1。

```
db2 rollforward db sample to end of logs tablespace(TBS1)
```

该命令失败，因为 TBS1 未准备好在节点 1 上进行前滚恢复。报告 SQL4906N。

```
db2 rollforward db sample to end of logs on nodes (0, 2) tablespace(TBS1)
```

成功完成。

```
db2 rollforward db sample to 1998-04-03-14.21.56.245378 and stop
   tablespace(TBS1)
```

该命令失败，因为 TBS1 未准备好在节点 1 上进行前滚恢复；必须将所有部分前滚到一起。

注: 随着表空间前滚到某时间点，将不接受节点子句。前滚操作必须在表空间所驻留的所有节点上进行。

在节点 1 上复原 TBS1 后:

```
db2 rollforward db sample to 1998-04-03-14.21.56.245378 and stop
   tablespace(TBS1)
```

成功完成。

示例 7 (MPP)

在所有节点上复原表空间后前滚至 PIT2，但不指定 AND STOP。前滚操作仍在进行中。取消并前滚至 PIT1:

```
db2 rollforward db sample to pit2 tablespace(TBS1)
db2 rollforward db sample cancel tablespace(TBS1)

** restore TBS1 on all nodes **
```

```
db2 rollforward db sample to pit1 tablespace(TBS1)
db2 rollforward db sample stop tablespace(TBS1)
```

示例 8 (MPP)

前滚恢复 db2nodes.cfg 文件中列出的 8 个节点（3 至 10）上驻留的表空间:

```
db2 rollforward database dwtest to end of logs tablespace (tssprodt)
```

前滚恢复至日志末尾（而不是时间点）的操作成功完成。不必指定表空间所驻留的节点。实用程序缺省到 db2nodes.cfg 文件。

示例 9 (MPP)

前滚恢复驻留在单节点数据库分区组（在节点 6 上）上的 6 个小型表空间:

```
db2 rollforward database dwtest to end of logs on node (6)
   tablespace(tsstore, tssbuyer, tsstime, tsswhse, tsslscat, tssvendor)
```

前滚恢复至日志末尾（而不是时间点）的操作成功完成。

第 2 部分 高可用性

第 5 章 高可用性和故障转移支持简介

成功的电子商务取决于事务处理系统不间断的高可用性，而事务处理系统是由数据库管理系统，如 DB2 驱动的，数据库管理系统必须每周 7 天，每天 24 小时可用（即 “24 x 7” ）。本节讨论以下内容:

高可用性

高可用性（HA）是一个术语，用于描述那些可随时对客户运行并可用的系统。为此:

- 必须高效地处理事务，且在高峰操作期内不能有显著的性能下降（甚至是失去了可用性）。在分区数据库环境中，DB2® 可以同时利用分区内和分区间并行性来高效处理事务。*分区内并行性*可用于 SMP 环境中，用来同时处理复杂的 SQL 语句的各种组件。*分区间并行性*在分区数据库环境中，另一方面它指的是在所有参与节点上同时处理查询；每个节点处理表行的一个子集。
- 在发生硬件或软件故障，或出现灾难性的事故时系统必须能够快速恢复。DB2 具有高级连续检查点系统和并行恢复能力，能够非常快地进行崩溃恢复。

  快速恢复能力还取决于是否有一个经过验证的备份与恢复策略。
- 增强了企业数据库能力的软件必须持续运行并可用于事务处理。要保持数据库管理器运行，必须确保有另一个数据库管理器可在该数据库管理器发生故障时代替它。这称作故障转移。*故障转移*功能允许在发生硬件故障时，自动将工作负荷从一个系统转移至另一个系统。

在永远前滚日志文件的另一机器上保留您的数据库的一份副本，可达到故障转移保护的效果。*日志交付*是一个将整日志文件复制到备用机器上的进程，可从归档设备复制，也可通过对主数据库运行的用户出口程序复制。通过这种方法，主数据库可使用 DB2 复原实用程序或分割镜像功能，复原到备用机器上。可以使用新的暂挂 I/O 支持来快速初始化新数据库。备用机器上的辅助数据库将继续前滚日志文件。如果主数据库发生故障，任何剩余的日志文件都将复制到备用机器上。前滚到日志末尾并停止操作后，所有的客户机都将重新连接到备用机器上的辅助数据库。

也可通过您可添加到系统中的特定于平台的软件来提供故障转移支持。例如:

- 高可用性群集多重处理，增强的可缩放性 AIX 版。

  有关 HACMP/ES 的详细信息，参见标题为 IBM® DB2 Universal Database™ Enterprise Edition for AIX® and HACMP/ES 的白皮书，它可以从 “DB2 UDB 和 DB2 Connect 在线支持”（DB2 UDB and DB2 Connect Online Support）Web 站点（http://www.ibm.com/software/data/pubs/papers/）获得。

- Microsoft® Cluster Server Windows® 版操作系统。

  有关 Microsoft Cluster Server 的信息，参见下列白皮书，这些白皮书可从 “DB2 UDB 和 DB2 Connect™ 在线支持”（DB2 UDB and DB2 Connect™ Online Support）Web 站点（http://www.ibm.com/software/data/pubs/papers/）获得: Implementing IBM DB2 Universal Database Enterprise - Extended Edition with Microsoft Cluster Server、Implementing IBM DB2 Universal Database Enterprise Edition with Microsoft Cluster Server 和 DB2 Universal Database for Windows: High Availability Support Using Microsoft Cluster Server - Overview。

- Sun Cluster 或 VERITAS Cluster Server，用于 Solaris 操作环境。

  有关 Sun Cluster 3.0 的信息，参见标题为 DB2 and High Availability on Sun Cluster 3.0 的白皮书，它可从 “DB2 UDB 和 DB2 Connect 在线支持”（DB2 UDB and DB2 Connect Online Support）Web 站点（http://www.ibm.com/software/data/pubs/papers/）获得。有关 VERITAS Cluster Server 的信息，参见标题为 DB2 and High Availability on VERITAS Cluster Server 的白皮书，也可以从 “DB2 UDB 和 DB2 Connect 在线支持”（DB2 UDB and DB2 Connect Online Support）Web 站点获得。

- 惠普公司的 Multi-Computer/ServiceGuard。

  有关 HP MC/ServiceGuard 的详细信息，参见讨论使用 Hewlett-Packard’s MC/ServiceGuard 高可用性软件的 IBM DB2 实现和证明的白皮书，它可从 IBM Data Management Products for HP Web 站点（http://www.ibm.com/software/data/hp/pdfs/db2mc.pdf）获得。

故障转移策略通常是基于群集系统的。一个*群集*就是一组互相连接的系统，可一起工作，相当于单个系统。每个处理器都称为该群集中的一个节点。群集技术可将故障服务器上的工作负荷承担过来，从而使服务器可在发生故障时互相备份。

IP 地址接管（或 IP 接管）是一项在服务器当机时，将服务器 IP 地址从一台机器传送到另一台机器的能力；对于客户机应用程序，这两台机器在不同的时候表现为同一服务器。

故障转移软件可能会在系统间使用*波动信号监视*或*保持活动信息包*以确认可用性。波动信号监视涉及到在一个群集中的所有节点间维护持续通信的系统服务。如果未检测到波动信号，就会对备份系统启动故障转移。最终用户通常没有意识到系统已发生故障。

市场上最常见的两种故障转移策略称为*空闲备用*和*相互接管*，虽然取决于各个供应商，与这些术语相关的配置可能会涉及到其它不同的术语。

空闲备用

在这种配置中，使用一个系统来运行 DB2 实例，而第二个处理器处于“空闲”状态或备用方式，准备在第一个系统运行的环境中发生操作系统或硬件故障时接管该实例。因为备用系统一直处于空闲状态直到需要它为止，所以整体系统性能不会受到影响。

相互接管

在此配置中，将每个系统都指定为另一系统的备份。因为备份系统在故障转移后必须做一些额外的工作：它必须做它自己的工作加上那些已前由那个发生了故障的系统完成的工作，所以整体系统性能可能会受到影响。

故障转移策略可用于对实例、分区或多个逻辑节点进行故障转移。

相关概念:

- 『并行性』（*《管理指南：计划》*）
- 第 3 页的『开发备份与恢复策略』
- 第 147 页的『通过联机分割镜像以及暂挂 I/O 支持实现的高可用性』
- 第 167 页的『Solaris 操作环境中的高可用性』
- 第 170 页的『Sun Cluster 3.0 上的高可用性』
- 第 172 页的『VERITAS Cluster Server 的高可用性』
- 第 161 页的第 7 章，『Windows 操作系统上的高可用性』
- 第 155 页的第 6 章，『AIX 上的高可用性』

通过日志交付的高可用性

日志交付是将整个日志文件复制到备用机器上的过程，可以从归档设备上复制，也可以通过对主数据库运行的用户出口程序来复制。备用数据库根据生产机器生成的日志文件不断前滚。当生产机器发生故障时，发生故障转移并出现下列情况:

- 余下日志被传送到备用机器。
- 备用数据库前滚至日志的末尾，然后停止。
- 客户机与备用数据库重新连接，并继续运行。

备用机器有它自己的资源（即，磁盘），但必须具有与生产数据库相同的物理和逻辑定义。当使用这种方法时，通过使用复原实用程序或分割镜像功能将主数据库复原到备用机器上。

要确保能够在灾难恢复情况下恢复数据库，考虑下列事宜:

- 归档位置应该在地理上独立于主位置。
- 在备用数据库位置远程镜像日志
- 使用同步镜像，以避免损失。可以通过 DB2® 日志镜像或现代磁盘子系统（例如 ESS 和 EMC）来完成此任务。同时建议使用 NVRAM 高速缓存（本地和远程）来最小化灾难恢复情况下对性能的影响。

注:

1. 当备用数据库处理指示在主数据库上发生索引重新构建的日志记录时，备用服务器上的索引将不会自动重新构建。在备用服务器脱离前滚暂挂状态之后，与数据库的首次连接或首次尝试存取索引时，将在备用服务器上重新构建该索引。如果在主服务器上重新构建所有索引，则建议备用服务器与主服务器重新同步。
2. 如果装入实用程序在主数据库上运行并指定了 COPY YES 选项，则备用数据库必须具有对副本映象的存取权。
3. 如果装入实用程序在主数据库上运行并指定了 COPY NO 选项，则应该重新同步备用数据库，否则表空间将处于复原暂挂状态。
4. 有两种方法来初始化备用机器:
   a. 通过从备份映象复原至备用机器。
   b. 通过创建生产系统的分割镜像并发出带有 STANDBY 选项的 **db2inidb** 命令。

   仅在已初始化备用机器之后，才能在备用系统上发出 ROLLFORWARD 命令。
5. 未记录的操作将不会在备用数据库上重放。这样，建议在这类操作之后重新同步备用数据库。可以通过分割镜像和暂挂 I/O 支持来完成此任务。

相关概念:

- 第 147 页的『通过联机分割镜像以及暂挂 I/O 支持实现的高可用性』

相关任务:

- 第 148 页的『将分割镜像用作备用数据库』

相关参考:

- 第 297 页的附录 G, 『数据库恢复的用户出口』

通过联机分割镜像以及暂挂 I/O 支持实现的高可用性

暂挂 I/O 支持通过对联机分割镜像处理提供一套完整的实现方案（即，分割镜像而不关闭数据库），使系统有持续的可用性。*分割镜像*是数据库的一个“瞬间”副本，可通过建立包含数据的磁盘的镜像，然后在需要副本时分割该镜像来实现。*磁盘镜像*是一个将所有数据写到两个单独的硬盘中（一个硬盘是另一个的镜像）的进程。*分割*镜像是分离数据库的主辅副本的进程。

如果不希望使用 DB2® 备份实用程序来备份大型数据库，可通过使用暂挂 I/O 和分割镜像功能根据镜象映象建立副本。此方法还可以:

- 消除来自生产机器的备份操作开销
- 提供了克隆系统的一个快捷方式
- 提供了对空闲备用故障转移的快速实现。不需要初始复原操作，如果证实前滚操作太慢或遇到了错误，重新初始化会非常快。

db2inidb 命令初始化分割镜像，因此可用来:

- 作为克隆数据库
- 作为备用数据库
- 作为备份映象

只能对分割镜像发出此命令，必须首先运行该命令才能使用分割镜像。

在分区数据库环境中，不必同时对所有分区暂挂 I/O 写操作。可以暂挂一个或多个分区的子集来为执行脱机备份创建分割镜像。如果目录节点包括在子集中，则它必须是要暂挂的最后一个分区。

在分区数据库环境中，**db2inidb** 命令必须先在每个分区中运行，才能使用来自任何这些分区的分割映象。通过使用 **db2_all** 命令，可在所有分区上同时运行该工具。

注: 确保分割镜像包含组成数据库的所有容器和目录（包括卷目录）。

相关参考:

- 第 195 页的『db2inidb — 初始化镜像数据库』

联机分割镜像处理

制作克隆数据库

限制:

不能备份克隆数据库，不能在原始系统上复原备份映象，也不能通过原始系统上生成的日志文件前滚。

过程:

要克隆一个数据库，应遵循以下步骤:

1. 暂挂主数据库上的 I/O ：

   ```
   db2 set write suspend for database
   ```

2. 使用适当的操作系统级命令来从主数据库中分割镜像。
3. 恢复主数据库上的 I/O:

   ```
   db2 set write resume for database
   ```

4. 从另一机器上连接到镜像的数据库。
5. 启动数据库实例:

   ```
   db2start
   ```

6. 初始化镜像的数据库，作为主数据库的克隆:

   ```
   db2inidb database_alias as snapshot
   ```

   注: 此命令将回滚分割发生时正在进行的事务，启动新的日志链序列以使主数据库的所有日志都不能在克隆数据库上重放。

相关概念:

- 第 147 页的『通过联机分割镜像以及暂挂 I/O 支持实现的高可用性』

相关参考:

- 第 195 页的『db2inidb — 初始化镜像数据库』

将分割镜像用作备用数据库

过程:

要将分割镜像用作备用数据库，遵循下列步骤:

1. 暂挂主数据库上的 I/O ：

   ```
   db2 set write suspend for database
   ```

2. 使用适当的操作系统级命令来从主数据库中分割镜像。
3. 恢复主数据库上的 I/O:

   ```
   db2 set write resume for database
   ```

4. 将镜像的数据库连接到另一实例。
5. 将镜像的数据库置于前滚暂挂状态:

```
db2inidb database_alias as standby
```

注: 如果只有 DMS 表空间（数据库管理空间），则可以建立完整的数据库备份以抵消在生产数据库上建立备份的开销。

6. 设置用户出口程序以从主系统中检索日志文件。
7. 将数据库前滚至日志末尾或时间点。
8. 继续检索日志文件，并通过日志前滚数据库，直到到达日志的末尾或备用数据库要求的时间点。
9. 要使备用数据库联机，发出带有指定的 STOP 选项的 ROLLFORWARD 命令。

相关概念:

- 第 147 页的『通过联机分割镜像以及暂挂 I/O 支持实现的高可用性』

相关任务:

- 第 147 页的『制作克隆数据库』
- 第 149 页的『将分割镜像用作备份映象』

相关参考:

- 第 195 页的『db2inidb — 初始化镜像数据库』

将分割镜像用作备份映象

过程:

要将分割镜像用作“备份映象”，遵循下列步骤:

1. 在主数据库上暂挂 I/O:
   ```
   db2 set write suspend for database
   ```
2. 使用适当的操作系统级命令来从主数据库中分割镜像。
3. 在主数据库上恢复 I/O:
   ```
   db2 set write resume for database
   ```
4. 主系统上发生故障，需要通过备份复原。
5. 停止主数据库实例:
   ```
   db2stop
   ```
6. 使用操作系统级命令来复制主系统上的分割数据。**不要复制分割日志文件**，因为前滚恢复需要主日志。
7. 启动主数据库实例:
   ```
   db2start
   ```
8. 初始化主数据库:

```
db2inidb database_alias as mirror
```

9. 将主数据库前滚至日志末尾或时间点，然后停止。

相关概念:

- 第 147 页的『通过联机分割镜像以及暂挂 I/O 支持实现的高可用性』

相关任务:

- 第 147 页的『制作克隆数据库』
- 第 148 页的『将分割镜像用作备用数据库』

相关参考:

- 第 195 页的『db2inidb — 初始化镜像数据库』

基于 UNIX 的系统的故障监视设施

在基于 UNIX® 的系统上，“故障监视设施”通过一起工作以确保 DB2 运行的进程序列来提高非群集的 DB2® 环境的可用性。也就是说，*init* 守护程序监视“故障监视协调程序”（FMC），FMC 监视故障监视器，而故障监视器监视 DB2。

“故障监视协调程序”（FMC）是在 UNIX 引导序列中启动的“故障监视设施”的进程。*init* 守护程序启动 FMC，并在 FMC 异常终止时重新启动 FMC。FMC 对每个 DB2 实例启动一个故障监视器。每个故障监视器作为一个守护进程来运行，并且具有与 DB2 实例相同的用户特权。一旦启动了故障监视器，就会监视 DB2 实例以确保它不会过早退出。如果有故障的监视器发生故障，则将通过 FMC 重新启动它。每个故障监视器将依次负责监视一个 DB2 实例。如果 DB2 实例过早退出，则故障监视器将重新启动它。

注:

1. 如果正使用高可用性群集产品（即 HACMP 或 MSCS），则必须关闭故障监视设施，因为实例的启动和关闭都是由群集产品控制的。
2. 仅当发出 **db2stop** 命令时，故障监视器才变得不活动。如果 DB2 实例以任何其它方式关闭，故障监视器都将再次启动它。

故障监视器注册表文件

故障监视器注册表文件是在启动故障监视器守护程序时为每台物理机器上的每个实例创建的。此文件中的值指定故障监视器的行为。可在 /sqllib/ 目录中找到该文件，文件名为 fm.<machine_name>.reg。通过使用 **db2fm** 命令，可以改变此文件。条目如下所示:

```
FM_ON = no
FM_ACTIVE = yes
START_TIMEOUT = 600
```

```
STOP_TIMEOUT = 600
STATUS_TIMEOUT = 20
STATUS_INTERVAL = 20
RESTART_RETRIES = 3
ACTION_RETRIES = 3
NOTIFY_ADDRESS = <instance_name>@<machine_name>
```

其中:

FM_ON

指定是否应启动故障监视器。如果该值设置为 NO，则将不启动故障监视器守护程序，或者如果已经启动，则将关闭它。缺省值是 NO。

FM_ACTIVE

指定故障监视器是否是活动的。仅当 FM_ON 和 FM_ACTIVE 都设置为 YES 时，故障监视器才执行操作。如果 FM_ON 设置为 YES 并且 FM_ACTIVE 设置为 NO，则将启动故障监视器守护程序，但故障监视器不活动。这意味着如果 DB2 关闭，则不会尝试将 DB2 重新联机。缺省值为 YES。

START_TIMEOUT

指定故障监视器必须在其内启动它监视的服务的时间长短。缺省值为 600 秒。

STOP_TIMEOUT

指定故障监视器必须在其内关闭它监视的服务的时间长短。缺省值为 600 秒。

STATUS_TIMEOUT

指定故障监视器必须在其内获取它监视的服务的状态的时间长短。缺省值为 20 秒。

STATUS_INTERVAL

指定用来获取受监视的服务的状态的两次连续调用之间的最小时间。缺省值为 20 秒。

RESTART_RETRIES

指定在尝试失败之后故障监视器将尝试获取受监视的服务的状态的次数。一旦达到此数字，故障监视器将执行操作以使服务回到联机状态。缺省值为 3。

ACTION_RETRIES

指定故障监视器将尝试使服务回到联机状态的次数。缺省值为 3。

NOTIFY_ADDRESS

指定故障监视器将通知消息发送至的电子邮件地址。缺省值为 <instance_name>@<machine_name>

通过使用 **db2fm** 命令可以改变此文件。例如:

要将实例 DB2INST1 的 START_TIMEOUT 值更新为 100 秒，发出:

```
db2fm -i db2inst1 -T 100
```

要将实例 DB2INST1 的 STOP_TIMEOUT 值更新为 200 秒，发出:

```
db2fm -i db2inst1 -T /200
```

要将实例 DB2INST1 的 START_TIMEOUT 值更新为 100 秒，将 STOP_TIMEOUT 值更新为 200 秒，发出:

```
db2fm -i db2inst1 -T 100/200
```

要对实例 DB2INST1 打开故障监视，发出:

```
db2fm -i db2inst1 -f yes
```

要对实例 DB2INST1 关闭故障监视，发出:

```
db2fm -i db2inst1 -f no
```

注: 如果故障监视器注册表文件不存在，则将使用缺省值。

相关参考:

- 第 152 页的『db2fm — DB2 故障监视器』

db2fm — DB2 故障监视器

控制 DB2 故障监视器守护进程。可以使用 db2fm 来配置故障监视器。

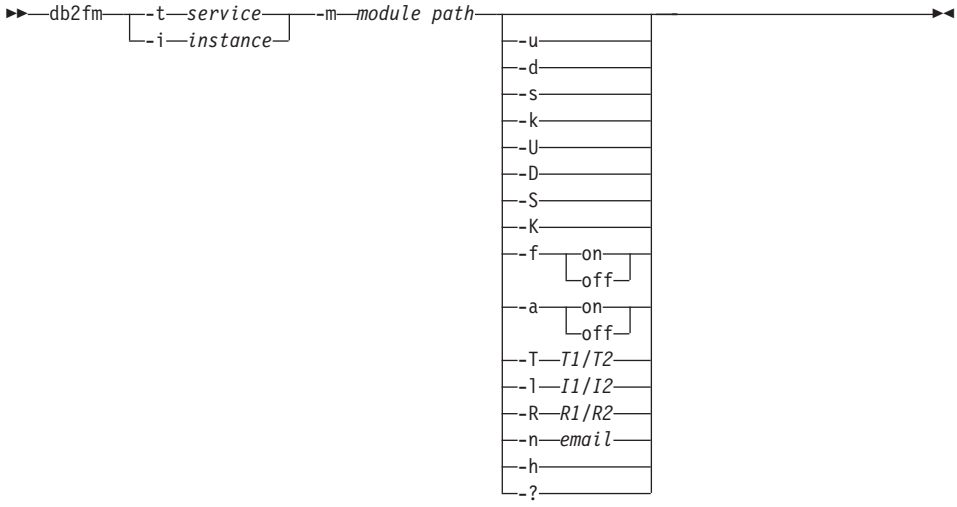
授权:

针对正对其运行命令的实例进行授权。

必需的连接:

无。

命令语法:



命令参数:

-m *module-path*

定义正在监视的产品的故障监视器共享库的全路径。缺省值是 $INSTANCEHOME/sqllib/lib/libdb2gcf。

-t *service*

对服务指定唯一的文本描述符。

-i *instance*

定义服务的实例。

-u 启动服务。

-U 启动故障监视器守护进程。

-d 终止服务。

-D 终止故障监视器守护进程。

-k 杀死服务。

-K 杀死故障监视器守护进程。

-s 返回服务的状态。

-S 返回故障监视器守护进程的状态。

注: 服务或故障监视器的状态可以是下列其中之一

- 未正确安装。
- 安装正确但不活动。
- 活动但不可用（维护）。

- 可用，或
- 未知

-f *on|off*

打开或关闭故障监视器。

注: 如果此选项设置为关闭，则将不启动故障监视器守护进程，或者如果它正在运行，则守护进程将会退出。

-a *on|off*

激活或释放故障监视。

注: 如果此选项设置为关闭，则故障监视器将不会积极进行监视，这意味着如果服务停止，则它不会尝试启动服务。

-T *T1/T2*

覆盖启动和停止超时。

例如，

- -T 15/10 分别更新两个超时
- -T 15 将启动超时更新为 15 秒
- -T /10 将停止超时更新为 10 秒

-I *I1/I2*

分别设置状态时间间隔和超时。

-R *R1/R2*

设置在放弃之前状态方法和操作重试的次数。

-n *email*

设置事件通知的电子邮件地址。

-h 打印使用情况。

-? 打印使用情况。

使用注意事项:

1. 此命令只能在 UNIX 平台上使用。

第 6 章 AIX 上的高可用性

增强的可伸缩性（ES）是高可用性群集多重处理方式（HACMP）AIX 版的一个功能部件。此功能部件提供的故障转移恢复与 HACMP 提供的相同，并与 HACMP 具有相同的事件结构。增强的可伸缩性还提供了:

- 较大的 HACMP 群集。
- 通过*用户定义事件*扩大错误检测范围。受监视的区域可触发用户定义事件，这些事件各不相同，如一个进程停止或调页空间接近其容量限制。若有必要，这类事件包括可添加至故障转移恢复进程的前事件和后事件。可将专用于其它实现的额外功能放在 HACMP 的前事件和后事件流中。

  规则文件（/usr/sbin/cluster/events/rules.hacmprd）包含 HACMP 事件。用户定义事件被添加至此文件。事件发生时要运行的脚本文件是此定义的一部分。
- HACMP 客户机实用程序，它们用于从 HACMP 群集外的 AIX® 物理节点监视和检测（一个或多个群集中的）状态更改。

HACMP ES 群集中的节点交换的消息称为*波动信号*或*保持活动的信息包*，每个节点通过此消息通知其它节点有关它的可用性的信息。已停止应答的节点导致该群集中的其余节点调用恢复。该恢复过程称为 *node_down 事件*，也可称为*故障转移*。恢复过程完成后，将节点重新集成到该群集中。这称为 *node_up 事件*。

有两种类型的事件: 一种是在 HACMP ES 的操作内期望的标准事件，另一种是与硬件和软件组件中的参数的监视相关联的用户定义事件。

其中一个标准事件是 node_down 事件。当计划应该执行什么操作来作为恢复过程的一部分时，HACMP 允许两个故障转移选项: 热（或空闲）备用和相互接管。

注: 当使用 HACMP 时，确保在引导时通过使用 **db2iauto** 实用程序不启动 DB2® 实例，如下所示:

```
db2iauto -off InstName
```

其中,

InstName 是实例的登录名。

群集配置

在*热备用*配置中，作为接管节点的 AIX 处理器节点不运行任何其它工作负荷。在*相互接管*配置中，作为接管节点的 AIX 处理器节点要运行其它工作负荷。

通常，在分区的数据库环境中，对于每个节点上的分区，DB2 通用数据库 通过以相互接管的方式来运行。例外的方案有：目录节点是热备用配置一部分。

在使用 HACMP ES 的 RS/6000® SP™ 上计划大型 DB2 安装时，需要考虑如何在 RS/6000 SP 大型机内部或之间划分群集节点。将节点和它的备份置于不同的 SP 大型机中，这样可以在一个大型机停机（即大型机电源／配电板发生故障）时进行接管。但是，发生此类故障的可能性非常小，因为每个 SP 大型机中有 *N*+1 个电源，每个 SP 交换机都有备用电路以及 *N*+1 个风扇和电源。在大型机发生故障的情况下，可能需要手工干预以复原其余的大型机。在 SP Administration Guide 中记载了此恢复过程。HACMP ES 确保了 SP 节点故障的恢复；计算机故障的恢复取决于一个或多个 SP 大型机内部群集的正确布局。

另一个计划的注意事项是如何管理大型群集。管理一个小的群集比管理一个大的群集容易得多；然而，管理一个大的群集又比管理许多小的群集容易得多。在计划过程中，要考虑在群集环境中如何使用应用程序。例如，若有一单个的、大型的、同类应用程序在 16 个节点上运行，则将配置作为单个群集管理比管理 8 个两节点群集可能要容易些。若相同的 16 个节点包含与不同网络、磁盘和节点相关的许多不同的应用程序，则可能最好将这些节点分组到更小的群集中。记住，一次只能将一个节点集成到 HACMP 群集中；启动多个群集的一个配置比启动一个大群集快得多。如果某个节点及其备份在同一群集中，那么 HACMP ES 可同时支持单个和多个群集。

HACMP ES 故障转移允许资源组对物理节点的预定义（也称作*级联*）分配。该故障转移过程还允许资源组对物理节点的*浮动*（也称作*旋转*）分配。IP 地址和外部磁盘卷组，或文件系统，或 NFS 文件系统，以及每个资源组内的应用服务器指定应用程序或应用程序组件，它们可由 HACMP ES 通过故障转移和重新集成在物理节点间进行操纵。故障转移和重新集成行为由所创建的资源组类型和该资源组中的节点数来指定。

例如，考虑一个 DB2 数据库分区（逻辑节点）。若其日志和表空间容器位于外部磁盘上，并且其它节点与那些磁盘链接，则其它那些节点可能可以访问这些磁盘并重新启动该数据库分区（在接管节点上）。这是 HACMP 自动执行的操作类型。HACMP ES 也可用来复原 DB2 实例主用户目录所使用的 NFS 文件系统。

在分区数据库环境中计划使用 DB2 UDB 进行恢复时，完整阅读 HACMP ES 文档。应阅读 Concepts, Planning, Installation, and Administration Guides，然后为您的环境构建恢复体系结构。对于根据已知故障点来标识的要恢复的每个子系统，标识您需要的 HACMP 群集，以及恢复节点（热备用或相互接管）。

强烈建议在外部磁盘配置中建立磁盘和适配器的镜像。对于为 HACMP 配置的 DB2 物理节点，需要小心确保卷组上的节点可从共享外部磁盘联机。在相互接管配

置中，这种方案需要某些附加计划，以便配对的节点可相互访问对方的卷组而不造成冲突。在分区数据库环境中，这表示所有容器名在所有数据库中都必须是唯一的。

实现唯一性的一种方式是包括分区号作为名称的一部分。当创建 SMS 或 DMS 容器时，可指定容器字符串语法的节点表达式。当指定表达式时，节点号可以是容器名的一部分，或者，若指定了附加自变量，则那些自变量的结果可以是容器名的一部分。使用自变量 ″ $N″ ([blank]$N) 来指示节点表达式。自变量必须在容器字符串的末尾，并且只能使用下列其中一种格式:

表 1. 用于创建容器的自变量. 假定该节点号为 5。

语法	示例	值
[blank]$N	″ $N″	5
[blank]$N+[number]	″ $N+1011″	1016
[blank]$N%[number]	″ $N%3″	2
[blank]$N+[number]%[number]	″ $N+12%13″	4
[blank]$N%[number]+[number]	″ $N%3+20″	22

注:

1. % 是系数。
2. 在所有情况下，从左往右对运算符求值。

下面是如何使用此特殊自变量来创建容器的一些示例:

- 创建在两节点系统上使用的容器。

```
CREATE TABLESPACE TS1 MANAGED BY DATABASE USING
   (device '/dev/rcont $N' 20000)
```

  将使用下列容器:

```
/dev/rcont0   — 在节点 0 上
/dev/rcont1   — 在节点 1 上
```

- 创建在有四个节点的系统上使用的容器。

```
CREATE TABLESPACE TS2 MANAGED BY DATABASE USING
   (file '/DB2/containers/TS2/container $N+100' 10000)
```

  将使用下列容器:

```
/DB2/containers/TS2/container100   — 在节点 0 上
/DB2/containers/TS2/container101   — 在节点 1 上
/DB2/containers/TS2/container102   — 在节点 2 上
/DB2/containers/TS2/container103   — 在节点 3 上
```

- 创建在两节点系统上使用的容器。

```
CREATE TABLESPACE TS3 MANAGED BY SYSTEM USING
   ('/TS3/cont $N%2, '/TS3/cont $N%2+2')
```

将使用下列容器:

```
/TS3/cont0   — 在节点 0 上
/TS3/cont2   — 在节点 0 上
/TS3/cont1   — 在节点 1 上
/TS3/cont3   — 在节点 1 上
```

为 HACMP ES 配置 DB2 数据库分区

一旦完成配置，HACMP ES 就启动实例中的每个数据库分区，一次启动一个物理节点。建议使用多个群集来启动多于四个节点的并行 DB2 配置。注意，在 64 节点的并行 DB2 配置中，并行启动 32 个两节点 HACMP 群集比启动四个 16 节点群集要快。

脚本文件 rc.db2pe 与 “DB2 UDB 企业服务器版” 封装在一起（并安装在每个节点的 /usr/bin 中），以对热备用或相互接管节点中的 HACMP ES 故障转移或恢复的配置提供帮助。另外，可在相互接管配置中的故障转移阶段从 rc.db2pe 内定制 DB2 缓冲池大小。（当两个数据库分区在一个物理节点上运行时，可以配置缓冲池大小以确保正确的资源分配。）

HACMP ES 事件监视和用户定义事件

进程在给定节点上中断时启动故障转移操作是用户定义事件的一个示例。可在 samples/hacmp/es 子目录中找到说明用户定义事件的示例，如关闭数据库分区并强制事务异常终止来释放页面调度空间。

规则文件 /user/sbin/cluster/events/rules.hacmprd 包含 HACMP 事件。此文件中的每个事件描述都有以下 9 个部件:

- 事件名，它必须是唯一的。
- 状态，或事件的限定符。事件名和状态是规则触发器。仅当 “HACMP ES 群集管理器” 找到具有与该事件名和状态对应的触发器的规则时，它才启动恢复。
- 资源程序路径，这是包含恢复程序的 *xxx*.rp 文件的完整路径说明。
- 恢复类型。它被保留以备后用。
- 恢复级别。它被保留以备后用。
- 资源变量名，它用于 “事件管理器” 事件。
- 实例向量，它用于 “事件管理器” 事件。这是一组格式为 “名称=值” 的元素。这些值唯一地标识系统中该资源的副本，并用扩展名唯一标识该资源变量的副本。

- 谓词，它用于“事件管理器”事件。这是资源变量与其它元素之间的关系表达式。当此表达式为真时，“事件管理”子系统生成一个事件以通知“群集管理器”和适当的应用程序。
- Rearm 谓词，它用于“事件管理器”事件。这是用来生成改变主谓词状态的事件的谓词。此谓词一般与主要谓词相反。它也可与该事件谓词一起使用来为您感兴趣的条件建立上限和下限。

每个对象在事件定义中都需要一行，即使不使用该行。若除去了这些行，则“HACMP ES 群集管理器”不能正确地对该事件定义进行语法分析，这可能会导致系统挂起。将任何以 ″#″ 开始的行视为注释行。

注: 该规则文件要求每个事件定义只能有九行（未计算任何注释行）。当在该规则文件的底部添加一个用户定义事件时，除去该文件末尾不需要的空行很重要，否则该节点将挂起。

HACMP ES 使用 PSSP 事件检测以处理用户定义事件。“PSSP 事件管理”子系统通过监视不同硬件和软件资源来提供综合的事件检测。

此过程概述如下:

1. “组服务／ES”（对于预定义事件）或“事件管理”（对于用户定义事件）将该事件通知“HACMP ES 群集管理器”。
2. “群集管理器”读取 `rules.hacmprd` 文件并确定映射至该事件的恢复程序。
3. “群集管理器”运行由恢复命令序列组成的恢复程序。
4. 恢复程序执行可能是 shell 脚本或二进制命令的恢复命令。（在 HACMP AIX 版中，恢复命令与 HACMP 事件脚本相同。）
5. “群集管理器”从恢复命令接收返回状态。一个意外的状态会“挂起”该群集，直到进行手工干预（使用 `smit cm_rec_aids` 或 `/usr/sbin/cluster/utilities/clruncmd` 命令）为止。

有关 AIX 上高可用的“IBM® DB2 通用数据库™”环境的实现和设计的详细信息，参见下列白皮书，它们可从“DB2 UDB and DB2 Connect™ 支持”（DB2 UDB and DB2 Connect™ Support）Web 站点（http://www.ibm.com/software/data/pubs/papers/）获得:

- “IBM DB2 Universal Database Enterprise Edition for AIX and HACMP/ES”
- “IBM DB2 Universal Database Enterprise - Extended Edition for AIX and HACMP/ES”

相关参考:

- 『db2start - Start DB2 Command』（*Command Reference*）

第 7 章 Windows 操作系统上的高可用性

简介

MSCS 是 Windows® NT Server、Windows 2000 Server 和 Windows .NET Server 操作系统的功能部件。此软件支持群集中两个服务器的连接（在 DataCenter Server 中最多为 4 个服务器）以获取数据和应用程序的高可用性和易于管理。MSCS 还可自动检测服务器或应用程序故障并进行恢复。它可用来移动服务器工作负荷以平衡机器利用率，以及提供计划的维护而不需要停机。

下列 DB2® 产品支持 MSCS:

- DB2 通用数据库™ 工作组服务器版
- DB2 通用数据库企业服务器版（DB2 ESE）
- DB2 通用数据库连接企业版（DB2 CEE）

DB2 MSCS 组件

群集是两个或多个节点的配置，其中每个节点都是独立的计算机系统。群集对网络客户机而言就好象是单个服务器。

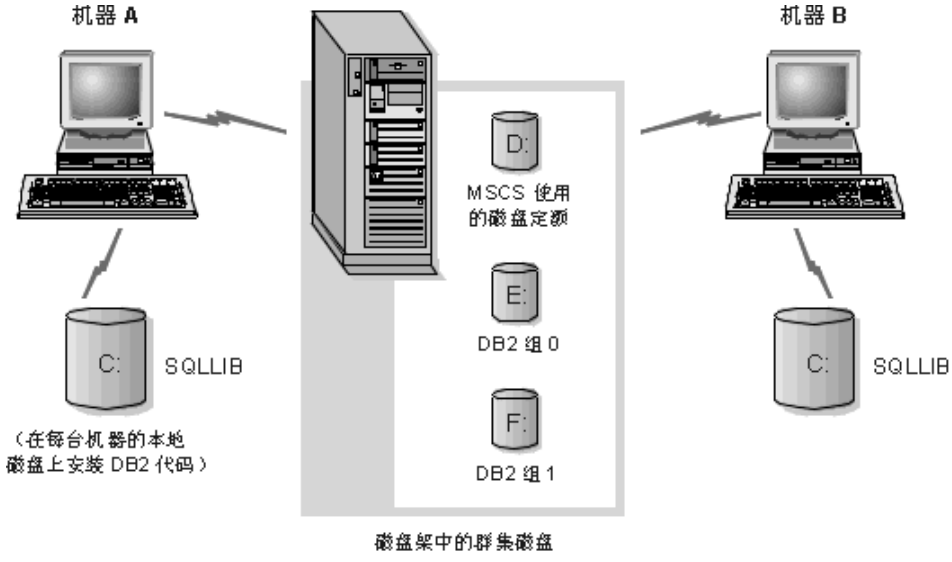


图 16. 示例 MSCS 配置

MSCS 群集中的节点是通过使用一个或多个共享存储总线和一个或多个物理上独立的网络来连接的。仅将服务器与群集连接而未将客户机与群集连接的网络称为*专用*网络。支持客户机连接的网络称为*公共*网络。每个节点上有一个或多个本地磁盘。每个共享存储总线与一个或多个磁盘连接。共享总线上的每个磁盘一次只能由群集的一个节点拥有。DB2 软件驻留在本地磁盘上。DB2 数据库文件（表、索引、日志文件等等）驻留在共享磁盘上。因为 MSCS 不支持在群集中使用原始分区，所以在 MSCS 环境中将 DB2 配置为使用原始设备是不可能的。

DB2 资源

在 MSCS 环境中，资源是由群集软件管理的实体。例如，磁盘、IP 地址或类属服务可以作为资源来管理。DB2 通过创建它自己的资源类型（名为 “DB2” ）来与 MSCS 集成。每个 DB2 资源管理一个 DB2 实例，并且在分区数据库环境中运行时，每个 DB2 资源管理一个数据库分区。DB2 资源的名称是实例名，虽然在分区数据库环境的情况下，DB2 资源的名称由实例名和分区（或节点）号组成。

联机前和联机后脚本

在将 DB2 资源联机前和联机后都可运行脚本。这些脚本分别被称为联机前和联机后脚本。联机前和联机后脚本是可以运行 DB2 和系统命令的 .BAT 文件。

在 DB2 的多个实例可以在同一机器上运行的情况下，可以使用联机前和联机后脚本来调整配置，以便可以成功启动实例。即使发生故障转移，也可以使用联机后脚本来执行手工数据库恢复。联机后脚本还可用来启动从属于 DB2 的任何应用程序或服务。

DB2 组

相关或从属资源被组织成资源组。组中的所有资源作为一个单元在群集节点之间移动。例如，在典型 DB2 单分区群集环境中，将存在包含下列资源的 DB2 组:

1. DB2 资源。DB2 资源管理 DB2 实例（或节点）。
2. IP 地址资源。IP 地址资源允许客户机应用程序与 DB2 服务器连接。
3. “网络名”资源。“网络名”资源允许客户机应用程序通过使用名称而不使用 IP 地址与 DB2 服务器连接。“网络名”资源具有与 IP 地址资源的相关性。“网络名”资源是可选的。（配置“网络名”资源可能影响故障转移性能。）
4. 一个或多个“物理磁盘”资源。每个“物理磁盘”资源管理群集中的一个共享磁盘。

注: DB2 资源的配置取决于同一组中的所有其它资源，因此仅当所有其它资源联机之后才能启动 DB2 服务器。

故障转移配置

可使用两种类型的配置:

- 热备用
- 相互接管

在一个分区数据库环境中，群集不必全部具有相同类型的配置。您可将一些群集设置为使用热备用，而将其它群集设置为使用相互接管。例如，若您的 DB2 实例由五个工作站组成，可以将两台机器设置为使用相互接管配置，将另外两个设置为使用热备用配置，而将剩余的一台机器配置为不支持故障转移。

热备用配置

在一个热备用配置中，MSCS 群集中的一台机器提供专用的故障转移支持，而另一台机器参与该数据库系统。若参与该数据库系统的机器发生故障，则该机器上的数据库服务器将在故障转移机器上启动。若在一个分区数据库系统中，您正在一台机器上运行多个逻辑节点，而该机器发生故障，则这些逻辑节点将在故障转移机器上启动。第 164 页的图 17 显示了一个热备用配置的示例。

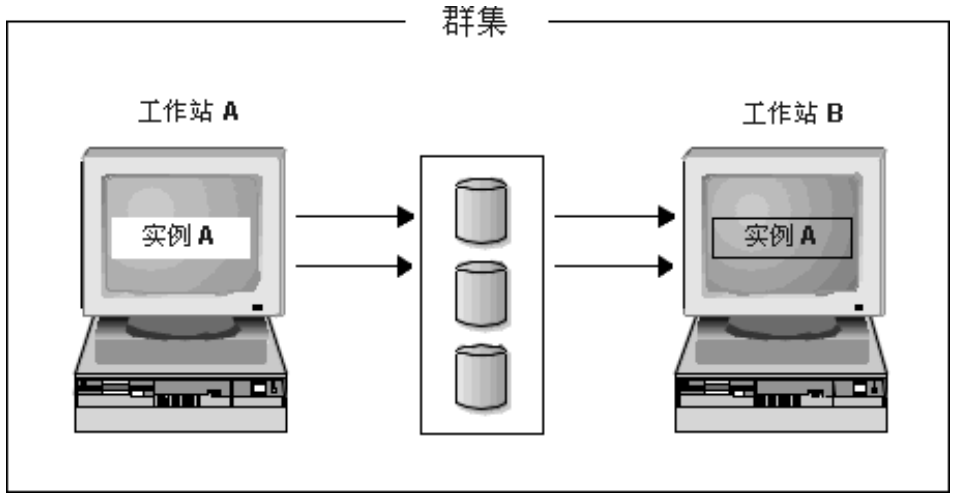


图 17. 热备用配置

相互接管配置

在一个相互接管配置中，两个工作站都参与数据库系统（即，每台机器上至少有一个数据库服务器在运行）。若 MSCS 群集中的一个工作站发生故障，则该发生故障的机器上的数据库服务器将在另一台机器上启动以运行。在一个相互接管配置中，一台机器上的数据库服务器发生故障时不会影响另一台机器上的数据库服务器。在任何给定的时间点，任何数据库服务器可在任何机器上是活动的。第 165 页的图 18 显示了一个相互接管配置的示例。

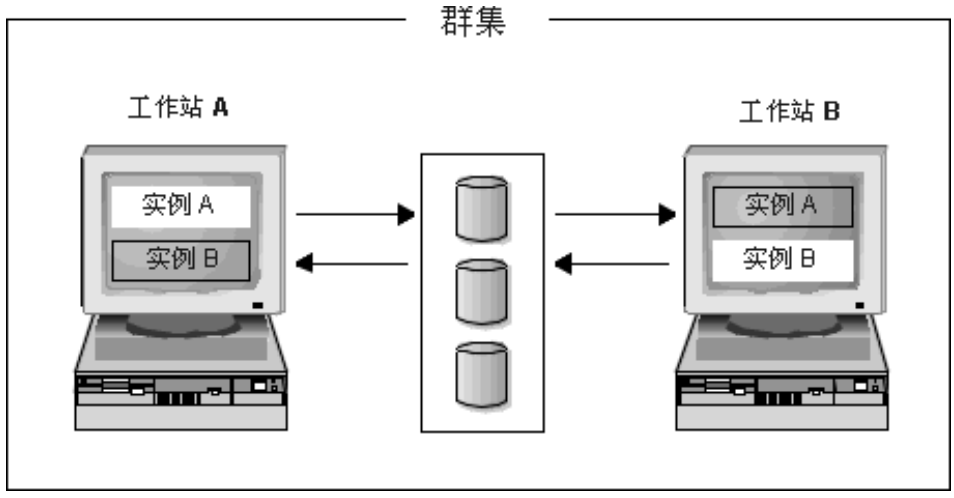


图 18. 相互接管配置

有关“Windows 操作系统”上高可用的“IBM® DB2 通用数据库”环境的实现和设计的详细信息，参见下列白皮书，它们可从“DB2 UDB 和 DB2 Connect™ 支持”（DB2 UDB and DB2 Connect™ Support）Web 站点（http://www.ibm.com/software/data/pubs/papers/）获得:

- “Implementing IBM DB2 Universal Database Enterprise - Extended Edition with Microsoft® Cluster Server”
- “Implementing IBM DB2 Universal Database Enterprise Edition with Microsoft Cluster Server”
- “DB2 Universal Database for Windows: High Availability Support Using Microsoft Cluster Server - Overview”

第 8 章 Solaris 操作环境中的高可用性

Solaris 操作环境中的高可用性

“Solaris 操作环境”中的高可用性可通过将 DB2® 与 Sun Cluster 3.0（SC3.0）或 Veritas Cluster Server（VCS）配合使用来获得。有关 Sun Cluster 3.0 的信息，参见标题为 DB2 and High Availability on Sun Cluster 3.0 的白皮书，它可从“DB2 UDB 和 DB2 Connect 在线支持”（DB2 UDB and DB2 Connect Online Support）Web 站点（http://www.ibm.com/software/data/pubs/papers/）获得。有关 VERITAS Cluster Server 的信息，参见标题为 DB2 and High Availability on VERITAS Cluster Server 的白皮书，它还可以从“DB2 UDB 和 DB2 Connect 在线支持”（DB2 UDB and DB2 Connect Online Support）web 站点获得。

注: 当使用 Sun Cluster 3.0 或 Veritas Cluster Server 时，确保在引导时通过使用 **db2iauto** 实用程序不启动 DB2 实例，如下所示:

```
db2iauto -off InstName
```

其中,

`InstName` 是实例的登录名。

高可用性

提供数据服务的计算机系统包含许多不同的组件，每个组件都有一个关联的“平均无故障时间”(MTBF)。MTBF 是部件保持可用的平均时间。高质量硬盘驱动器的 MTBF 大约是一百万小时（大约 114 年）。虽然这看起来象是很长一段时间，但每 200 个硬盘中，就有一个有可能在 6 个月内发生故障。

虽然有很多方法可以提高数据服务的可用性，但最常用的方法还是 HA 群集。当用于高可用性时，群集由两台或更多机器、一组专用网络接口、一个或多个公用网络接口以及一些共享磁盘组成。这种特殊的配置允许将数据服务从一台机器移至另一机器。通过将数据服务移至群集中的另一机器，应该能够继续提供对其数据的访问。将数据服务从一台机器移至另一机器称为*故障转移*，第 168 页的图 19 对其进行了说明。

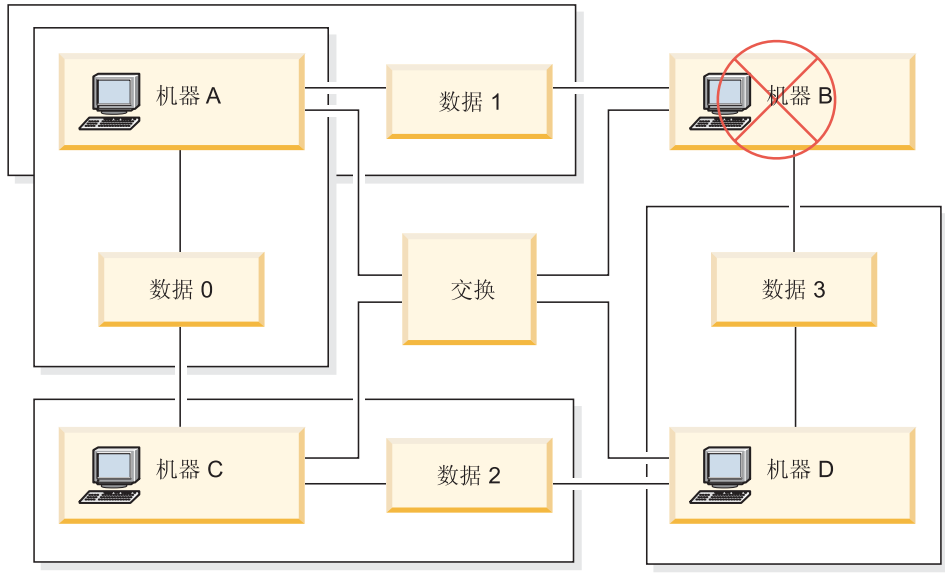


图 19. 故障转移. 当机器 B 故障时，它的数据服务被移到群集中的另一台机器上，以便仍可存取这些数据。

专用网络接口用来在群集中的机器之间发送*波动信号*消息和控制消息。公用网络接口用来直接与 HA 群集的客户机通信。HA 群集中的磁盘与群集中的两台或更多机器相连，因此，当一台机器发生故障时，另一机器可以访问它们。

在 HA 群集上运行的数据服务有一个或多个逻辑公用网络接口和一组磁盘与其关联。HA 数据服务的客户机只通过 TCP/IP 与该数据服务的逻辑网络接口相连。如果执行故障转移，则该数据服务及其逻辑网络接口和磁盘组便移至另一机器。

HA 群集的其中一个好处是可以在没有支持人员辅助的情况下恢复数据服务，并且，可以随时这样做。另一个好处是冗余。群集中的所有部件都应该是冗余的，包括机器本身。群集应当能够经受住任何单个故障点。

虽然高度可用的数据服务在本质上可以有非常大的区别，但它们还是有一些公共的需求。高度可用数据服务的客户机期望该数据服务的网络地址和主机名保持不变，并期望能够以同一方式发出请求，而不考虑该数据服务具体是在哪一台机器上。

假定一个 Web 浏览器，它正在访问高可用的 Web 服务器。请求是使用 URL（统一资源定位器）发出的，该 URL 包含主机名和指向 Web 服务器上的文件的路径。浏览器期望该主机名和路径在 Web 服务器的故障转移之后保持不变。如果浏览器从 Web 服务器下载文件，而该服务器正在进行故障转移，则该浏览器将需要重新发出请求。

数据服务的可用性是通过数据服务可供其用户使用的时间量测量的。最常用的可用性计量单位是“运行时间”的百分比；此百分比通常用数字“9”来表示:

```
99.99% => 服务（最多）停止 52.6 分钟／年
99.999% => 服务（最多）停止 5.26 分钟／年
99.9999% => 服务（最多）停止 31.5 秒／年
```

当设计和测试 HA 群集时:

1. 确保群集管理员熟悉系统并了解发生故障转移时应发生的情况。
2. 确保群集的每一部分都确实是冗余的，当它发生故障时，可以被快速替换。
3. 强制一个测试系统在受控环境中发生故障，并确保它每次都能正确地进行故障转移。
4. 跟踪每次故障转移的原因。虽然这应该不会经常发生，但找出任何导致群集不稳定的问题非常重要。例如，如果群集的一个部分一个月导致了 5 次故障转移，则找出原因并修正它。
5. 当发生故障转移时，务必通知该群集的支持人员。
6. 不能使群集过载。确保其余的系统在发生故障转移之后，仍能够处理适量的工作负荷。
7. 经常检查容易引起故障的部件（如磁盘），以便可以在发生问题之前进行替换。

容错

另一种提高数据服务可用性的方法是故障容错。*故障容错*机器内置了其所有冗余，应该能够经受住任何部件（包括 CPU 和内存）的单一故障。故障容错机器最常用在小型市场中，其实现成本比较高。机器位于不同地理位置的 HA 群集还有一个优点，即可以从仅影响那些位置的一部分的灾难中恢复。

由于 HA 群集可伸缩、易于使用以及实现成本相对较低，所以它是最常见的提高可用性的解决方案。

相关概念:

- 第 170 页的『Sun Cluster 3.0 上的高可用性』
- 第 172 页的『VERITAS Cluster Server 的高可用性』

Sun Cluster 3.0 上的高可用性

本节提供 DB2® 如何与 Sun Cluster 3.0 配合使用来达到高可用性的概述，包括对高可用性代理进程的描述，该代理进程充当两个软件产品之间的中介（参见图 20）。

图 20. DB2、Sun Cluster 3.0 和高可用性. DB2、Sun Cluster 3.0 与高可用性代理进程之间的关系。

故障转移

Sun Cluster 3.0 通过启用应用程序故障转移来提供高可用性。定期监视每个节点，群集软件自动将群集应用程序从失败的主节点重定位至指定的辅助节点。当故障转移发生时，客户机可能遇到短时间中断服务的情况，可能必须重新连接到服务器上。然而，它们将不知道它们正从其中存取应用程序和数据的物理服务器。通过允许主节点故障时由群集中的其它节点自动主管工作负荷，Sun Cluster 3.0 可以显著地减少停机时间并提高工作效率。

多主机磁盘

Sun Cluster 3.0 需要多主机磁盘存储器。这表示磁盘一次可以与多个节点连接。在 Sun Cluster 3.0 环境中，多主机存储器允许磁盘机变得高可用。驻留在多主机存储器上的磁盘机允许单个节点故障，因为通过备用服务器节点，仍然有一条物理路径指向数据。通过主节点可以全局存取多主机磁盘。如果客户机请求正通过一个节点来存取数据，且该节点故障，则将这些请求切换至具有与相同磁盘的直接连接的另一个节点。卷管理器为多主机磁盘的数据冗余提供镜像或 RAID 5 配置。通常，Sun Cluster 3.0 支持 Solstice DiskSuite 和 VERITAS Volume Manager 作为卷管理器。将多主机磁盘与磁盘镜像和组合分割区组合在一起防止节点故障和个别磁盘故障。

全局设备

使用全局设备，可以提供从任何节点对群集范围内的群集中的任何设备的高可用存取，而不必考虑设备的物理位置。所有磁盘都包括在带有指定设备标识（DID）的全局名称空间中，并且被配置为全局设备。因此，从所有群集节点上看，磁盘本身是可视的。

文件系统／全局文件系统

群集或全局文件系统是内核（在一个节点上）与基础文件系统卷管理器（在具有与一个或多个磁盘的物理连接的节点上）之间的代理。群集文件系统依赖于具有与一个或多个节点的物理连接的全局设备。它们与基础文件系统和卷管理器无关。通常，可以通过使用 Solstice DiskSuite 或 VERITAS Volume Manager 在 UFS 上构建群集文件系统。仅当磁盘上的文件系统整体上作为群集文件系统安装时，数据才变得对所有节点都可用。

设备组

所有多主机磁盘都必须受控于 Sun Cluster 框架。由 Solstice DiskSuite 或 VERITAS Volume Manager 管理的磁盘组首先是在多主机磁盘上创建的。然后，将它们注册为 Sun Cluster 磁盘设备组。磁盘设备组是一种类型的全局设备。多主机设备组是高可用的。如果当前管理设备组的节点失败，则可以通过备用路径存取磁盘。管理设备组的节点的故障不影响对设备组的存取，但执行恢复和一致性校验所需的时间除外。在此时间段，所有请求停止（对应用程序是透明的），直到系统使设备组可用为止。

资源组管理器（RGM）

RGM 提供高可用性的机制并在每个节点上作为守护程序运行。它根据预先配置策略在选择的节点上自动启动和停止资源。即使在发生节点故障的情况下，RGM 允许资源高可用，或通过在受影响的节点上停止并在另一个节点上启动它来重新引导。RGM 还会自动启动和停止特定于资源的监视器，以便检测资源故障并将失败的资源重定位至另一个节点。

数据服务

术语“数据服务”用来描述已配置为在群集而不是单个服务器上运行的第三方应用程序。数据服务包括应用程序软件和启动、停止以及监视该应用程序的 Sun Cluster 3.0 软件。Sun Cluster 3.0 提供用来控制和监视群集内的应用程序的数据服务方法。这些方法在“资源组管理器”（RGM）的控制下运行，它使用这些方法来启动、停止和监视群集节点上的应用程序。这些方法与群集框架软件和多主机磁盘一起，使应用程序成为高可用的数据服务。作为高可用的数据服务，它们可以在群集内任何单个故障后防止重要的应用程序中断，而不考虑故障是在节点

上、接口组件上还是在应用程序本身中发生的。RGM 还管理群集中的资源，包括网络资源（逻辑主机名和共享地址）和应用程序实例。

资源类型、资源和资源组

资源类型是由下列部分组成:

1. 要在群集上运行的软件应用程序。
2. RGM 用作回调方法以管理作为群集资源的应用程序的控制程序。
3. 组成群集的静态配置的部分的一组特性。

RGM 使用资源类型特性来管理特定类型的资源。

资源继承其资源类型的特性和值。这是在群集上运行的基础应用程序的实例。每个实例在群集内都需要有唯一的名称。每个资源都必须在资源组中配置。RGM 将同一节点上的某个组的所有资源同时联机和脱机。当 RGM 将资源组联机或脱机时，它对该组中的个别资源调用回调方法。

资源组当前在其上联机的节点被其主节点或主体调用。资源组由它的每一个主体管理。每个资源组都有相关联的 Nodelist 特性，由群集管理员设置，以指定资源组的所有潜在主体或主方。

有关 Sun Cluster 3.0 平台上的高可用 “IBM® DB2 通用数据库” 环境的实现和设计的详细信息，参见标题为 DB2 and High Availability on Sun Cluster 3.0 的白皮书，它可以从 “DB2 UDB 和 DB2 Connect™ 支持”（DB2 UDB and DB2 Connect™ Support）Web 站点（http://www.ibm.com/software/data/pubs/papers/）获得。

相关概念:

- 第 167 页的『Solaris 操作环境中的高可用性』
- 第 172 页的『VERITAS Cluster Server 的高可用性』

VERITAS Cluster Server 的高可用性

可以使用 VERITAS Cluster Server 来减少计划的或未计划的停机时间。它可以实现服务器联合和在异种环境中有效地管理大量应用程序。在存储区网络（SAN）和传统客户机／服务器环境中，VERITAS Cluster Server 最多支持 32 个节点群集；在网络存储环境中，VERITAS Cluster Server 可以保护一切对象，从单个关键数据库实例到非常大的多应用程序群集。本节提供 VERITAS Cluster Server 的功能部件的简短总结。

硬件需求

以下内容是当前受 VERITAS Cluster Server 支持的硬件的列表:

- 对于服务器节点:
  - Sun Microsystems 提供的运行 Solaris 2.6 或更新版本的任何 SPARC/Solaris 服务器最少应有 128MB RAM。
- 对于磁盘存储器:
  - EMC Symmetrix、IBM® Enterprise Storage Server、HDS 7700 and 9xxx、Sun T3、Sun A5000、Sun A1000、Sun D1000 以及受 VCS 2.0 或更新版本支持的任何其它磁盘存储器; VERITAS 代表会确认哪些磁盘子系统是受支持的, 或者可以参考 VCS 文档。
  - 典型环境将需要镜像专用磁盘（在每个群集节点中）以用于 DB2® UDB 二进制文件和 DB2 UDB 数据的节点之间的共享磁盘。
- 对于网络互连:
  - 对于公共网络连接, 支持基于 IP 寻址的任何网络连接。
  - 对于波动信号连接（群集内部）, 冗余波动信号连接是必需的; 这种需求可通过每个服务器使用两个附加以太网控制器或者每个服务器使用一个附加以太网控制器且每个群集使用一个共享 GABdisk 来满足

软件需求

下列 VERITAS 软件组件是限定配置:

- VERITAS Volume Manager 3.2 或更新版本、VERITAS File System 3.4 或更新版本以及 VERITAS Cluster Server 2.0 或更新版本。
- DB Edition for DB2 Solaris 版 1.0 或更新版本。

当 VERITAS Cluster Server 不需要卷管理器时, 强烈建议使用 VERITAS Volume Manager 来轻松安装、配置和管理。

故障转移

VERITAS Cluster Server 是一个可用性群集解决方案, 它通过启用应用程序故障转移来管理应用程序服务（例如 DB2 UDB）的可用性。会定期监视每个个别群集节点的状态及其相关联的软件服务通常。当发生中断应用程序服务（在这种情况下, 是 DB2 UDB 服务）的故障时, VERITAS Cluster Server 和 / 或 VCS HA-DB2 Agent 将检测到该故障并自动执行一些步骤来复原该服务。这可能包括在同一节点上重新启动 DB2 UDB 或将 DB2 UDB 移到群集中的另一个节点上并在该节点上重新启动它。如果需要将应用程序迁移到新节点, 则 VERITAS Cluster Server 将与该应用程序相关联的一切（即, 网络 IP 地址和基础存储器的所有权）移至新节点, 以使用户不会意识到实际上服务在另一个节点上运行。他们将仍然通过使用相同的 IP 地址来存取该服务, 但是这些地址现在将指向另一个群集节点。

当 VERITAS Cluster Server 发生故障转移时，用户可能看到或可能看不到服务中断。这将取决于客户机与应用程序服务的连接的类型（有状态或无状态）。在带有有状态连接的应用程序环境中（如 DB2 UDB），用户可能会发现服务的短时间中断，并且在故障转移完成之后可能需要重新连接。在带有无状态连接（如 NFS）的应用程序环境中，用户可能发现服务的短时间延迟，但通常将不会看到中断并将不需要再次登录。

通过支持应用程序作为可以在群集节点之间自动迁移的服务，VERITAS Cluster Server 不仅可以减少未计划的停机时间，而且可以缩短与计划停机时间（即，维护和升级）相关联的运行中断持续时间。还可以手工启动故障转移。如果硬件或操作系统升级必须在特定节点上执行，则 DB2 UDB 可以迁移至群集中的另一个节点，并可以执行更新，然后可以将 DB2 UDB 迁移回原始节点。

建议在这些类型的群集环境中使用的应用程序应允许崩溃。允许崩溃的应用程序在仍然保持落实数据的完整性的同时可以从意外崩溃中恢复。允许崩溃的应用程序有时被称为*群集友好*应用程序。DB2 UDB 是允许崩溃的应用程序。

共享存储器

与 “VCS HA-DB2 代理进程” 配合使用时，Veritas Cluster Server 需要共享存储器。共享存储器是具有与群集中的多个节点的物理连接的存储器。驻留在共享存储器上的磁盘机允许节点故障，因为通过一个或多个备用群集节点，指向磁盘机的物理路径仍然存在。

在 VERITAS Cluster Server 的控制下，群集节点可以通过称为 “磁盘组” 的逻辑结构来存取共享存储器。磁盘组表示逻辑定义的存储设备的集合，这些存储设备的所有权是可在群集中的节点之间自动迁移。在给定时间，只能将磁盘组导入至单个节点。例如，如果 “磁盘组 A” 导入至 “节点 1” 并且 “节点 1” 发生故障，则可以从故障节点导出 “磁盘组 A”，并将它导入至群集中的另一节点。VERITAS Cluster Server 可以同时控制单个群集中的多个磁盘组。

除了允许磁盘组定义之外，卷管理器还可以通过使用镜像或 RAID 5 在共享存储器上提供冗余数据配置。VERITAS Cluster Server 支持 VERITAS Volume Manager 和 Solstice DiskSuite 作为逻辑卷管理器。将共享存储器与磁盘镜像和分割组合区组合在一起以防止节点故障和个别磁盘或控制器故障。

VERITAS Cluster Server 全局原子广播（GAB）和低等待时间传送（LLT）

群集配置中需要节点间的通信机制，以便节点可以根据硬件和软件状态交换信息，记录群集成员资格并使所有群集节点上的此信息同步。通过低等待时间传送（LLT）运行的 “全局原子广播”（GAB）设施提供由 VERITAS Cluster Server 使

用的高速、低等待时间机制来完成此任务。GAB 被作为内核模块装入到每个群集节点上，并提供原子广播机制，以确保所有节点同时获取状态更新信息。

通过利用内核间通信能力，LLT 对需要在群集节点间交换和同步的所有信息提供高速和低等待时间传送。GAB 在 LLT 顶部运行。VERITAS Cluster Server 不将 IP 用作波动信号机制，但提供两个其它可靠选项。带有 LLT 的 GAB 可配置为充当波动信号机制，或者可将 GABdisk 配置为基于磁盘的波动信号。波动信号必须通过冗余连接运行。这些连接可以是群集节点间两个专用以太网连接或一个专用以太网连接和一个 GABdisk 连接。使用两个 GABdisks 的配置不受支持，因为节点间群集状态的交换需要专用以太网连接。

有关 GAB 或 LLT 或有关如何在 VERITAS Cluster Server 配置中配置它们的更多信息，请参阅 VERITAS Cluster Server 2.0 User's Guide for Solaris。

捆绑和企业代理进程

代理进程是用来管理特定资源或应用程序的可用性的程序。当启动代理进程时，它从 VCS 获取必要的配置信息，然后定期监视资源或应用程序并用状态来更新 VCS。通常，使用代理进程来使资源联机、使资源脱机、或监视资源，同时提供四种类型的服务：启动、停止、监视和清除。启动和停止用来使资源联机或脱机，监视用来测试特定资源或应用程序以了解它的状态，而在恢复过程中使用清除。

将各种捆绑代理进程包括为 VERITAS Cluster Server 的一部分，并在安装 VERITAS Cluster Server 时安装这些代理进程。捆绑代理进程是这样的 VCS 进程，它们管理通常在群集配置中发现的预定义资源类型（即，IP、安装、处理和共享），并且它们有助于极大地简化群集安装和配置。超过 20 个捆绑代理进程与 VERITAS Cluster Server 捆绑在一起。

企业代理进程想要将重点放在特定应用程序（例如，DB2 UDB）上。“VCS HA-DB2 代理进程”可视作“企业代理进程”，它通过 VCS Agent 框架与 VCS 交互。

VCS 资源、资源类型和资源组

资源类型是用来定义 VCS 群集中将要监视的资源的对象定义。资源类型包括资源类型名和一组与该资源相关联的特性，这些特性从高可用性方面来看是非常重要的。资源继承其资源类型的特性和值，且资源名在群集范围内必须是唯一的。

有两种类型的资源：持久和标准（非持久）。持久资源是如网络接口控制器（NIC）之类的资源，它们受监视但未被 VCS 联机或脱机。标准资源是其联机和脱机状态受 VCS 控制的资源。

受监视的最低级别对象是资源，有各种资源类型（即，共享和安装）。每个资源必须配置到资源组中，且 VCS 将使特定资源组中的所有资源同时联机和脱机。要使资源组联机或脱机，VCS 将调用启动或停止方法以用于组中每个资源。有两种类型的资源组：故障转移和并行。高可用的 DB2 UDB 配置，不论是否为分区的，都将使用故障转移资源组。

“主体”或“主方”节点是可以潜在主管资源的节点。名为 systemlist 的资源组属性用来指定群集中哪些节点可以是特定资源组中的主体。在双节点群集中，通常两个节点都包括在 systemlist 中，但是在主管几个高可用的应用程序的较大多节点群集中，可能需要确保某些应用程序服务（它们的资源在最低级别定义）永远不可能故障转移至某些节点。

可在资源组之间定义相关性，在评估各种资源故障和管理恢复时，VERITAS Cluster Server 从属于此资源组相关性层次结构。例如，如果不能使资源组 ClientApp1 联机（除非资源组 DB2 已成功启动），资源组 ClientApp1 将被视作从属于资源组 DB2。

有关带有 VERITAS Cluster Server 的高可用“IBM DB2 通用数据库”环境的实现和设计的详细信息，参见标题为“DB2 UDB and High Availability with VERITAS Cluster Server”的技术手册，可以通过访问以下 Web 站点：http://www.ibm.com/support 并搜索关键字“1045033”来查看。

相关概念:

- 第 167 页的『Solaris 操作环境中的高可用性』
- 第 170 页的『Sun Cluster 3.0 上的高可用性』

第 3 部分 附录

附录 A. 如何阅读语法图

语法图显示了应如何指定一个命令，以使操作系统能正确地解释输入了什么内容。

从左至右，从上至下，沿着水平线（主路径）读语法图。如果该行以箭头结束，命令语法将继续，且下一行也以箭头开始。用一个垂直的条标记命令语法结束。

从语法图输入信息时，应确保包括了标点符号，如引号和等号。

参数分类为关键字或变量:

- 关键字代表常量，以大写字母显示；但在命令提示符处，可用大写、小写或混合大小写输入关键字。例如，命令名就是一个关键字。
- 变量代表用户提供的名称或值，由小写字母显示；但在命令提示符处，变量可用大写、小写或混合大小写输入，除非显式声明了大小写限制。例如，文件名就是一个变量。

参数可以是关键字和变量的组合。

所需的参数显示在主路径上:

可选参数显示在主路径下:

参数的缺省值显示在路径上方:

第一个参数显示在主路径上的一个参数堆栈，表示必须选择一个参数:

第一个参数显示在主路径下的一个参数堆栈，表示可以选择一个参数:

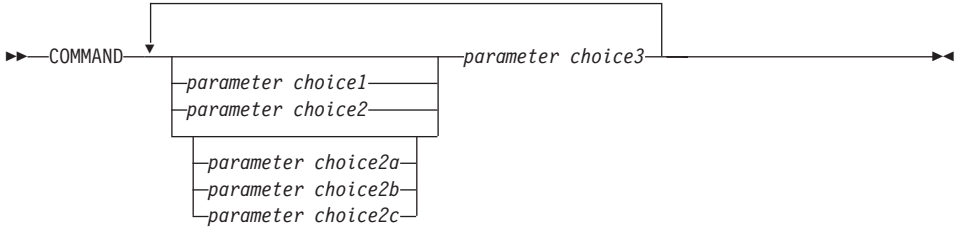
路径上返回到左边的一个箭头表示如果符合以下约定，就可重复该项:

- 如果箭头是不中断的，则可在各项由空格隔开的列表中重复该项:

- 如果箭头中有逗号，则可在各项由逗号隔开的列表中重复该项:

如果符合前面讨论过的对需的参数和可选参数的堆栈约定，就可重复参数堆栈中的项。

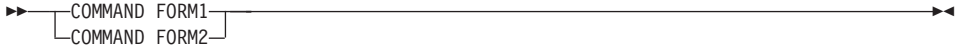
某些语法图包含了其它参数堆栈中的参数堆栈。只有符合前面讨论过的约定才能重复堆栈中的项。即，如果内部堆栈上没有重复箭头，但外部堆栈上有，则只能从内部堆栈选择一个参数，并将它与外部堆栈中的任何参数组合在一起，可重复该组合值。例如，以下语法图显示了可以将参数 *choice2a* 与参数 *choice2* 组合，然后再次重复该组合值（*choice2* 加 *choice2a*）:



某些命令前有可选路径参数:

如果未提供此参数，系统将在当前目录中搜索该命令。如果找不到该命令，系统会继续在 .profile 中列示的路径上的所有目录中搜索该命令。

某些命令有等效的语法变体:



附录 B. 警告、错误和完成消息

由各种实用程序生成的消息包括在 SQL 消息中。这些消息是由数据库管理器在已检测到警告或出错状态时生成。每条消息都有一个消息标识符，由前缀（SQL）和一个 4 位或 5 位的消息号组成。有 3 种消息类型：通知、警告和关键。已 N 结束的消息标识符是错误消息。那些以 W 结束的消息标识符表示警告或信息性消息。以 C 结束的消息标识符表示关键系统错误。

消息号也称为 *SQLCODE*。取决于它的消息类型（N、W 或 C），发送给应用程序的 SQLCODE 是正数或负数。N 和 C 产生负值，而 W 产生正值。DB2 将 SQLCODE 返回给应用程序，而应用程序可获得与该 SQLCODE 相关的消息。DB2 也会对可能是 SQL 语句结果的情况返回 *SQLSTATE* 值。某些 SQLCODE 值有相关的 SQLSTATE 值。

可使用此书中的信息来标识错误或问题，并通过适当的恢复操作来解决问题。此信息还可用于了解消息从何处产生并记录在何处。

也可从操作系统命令行访问 SQL 消息以及与 SQLSTATE 值相关的消息文本。要访问这些错误消息的帮助，可在操作系统命令提示符处输入:

```
db2 ? SQLnnnnn
```

其中 *nnnnn* 代表消息号。在基于 UNIX 的系统上，建议使用双引号分界，这将避免目录中有单字符文件名时的问题:

```
db2 "? SQLnnnnn"
```

因为 **db2** 命令是不区分大小写的，且不需要终止字母，所以接受消息标识符作为参数。因此，以下命令将产生相同的结果:

```
db2 ? SQL0000N
db2 ? sql0000
db2 ? SQL0000n
```

如果消息文本太长，在屏幕上放不下，可使用以下命令（在基于 UNIX 的操作系统上以及其它支持 "more" 管道的操作系统上）:

```
db2 ? SQLnnnnn | more
```

也可将输出重定向到一个可在稍后进行浏览的文件中。

也可从交互式输入方式调用帮助。要访问此方式，可在操作系统命令提示符处输入以下命令:

```
db2
```

要以这种方式获得 DB2 消息，可在命令提示符处输入以下命令（db2 =>）：

```
? SQLnnnnn
```

发出以下命令，可检索与 SQLSTATE 相关的消息文本：

```
db2 ? nnnnn
或
db2 ? nn
```

其中 *nnnnn* 是一个 5 字符的 SQLSTATE 值（字母数字值），而 *nn* 是一个 2 位的 SQLSTATE 类代码（SQLSTATE 值的前两位）。

附录 C. 附加 DB2 命令

系统命令

db2adutl — 使用 TSM 归档映象

允许用户查询、抽取、验证及删除备份映象、日志和使用 Tivoli Storage Manager（以前称为 ADSM）保存的装入复制映象。

在基于 UNIX 的操作系统上，此实用程序位于 sqllib/adsm 目录中。在 Windows 上，它位于 sqllib\bin 中。

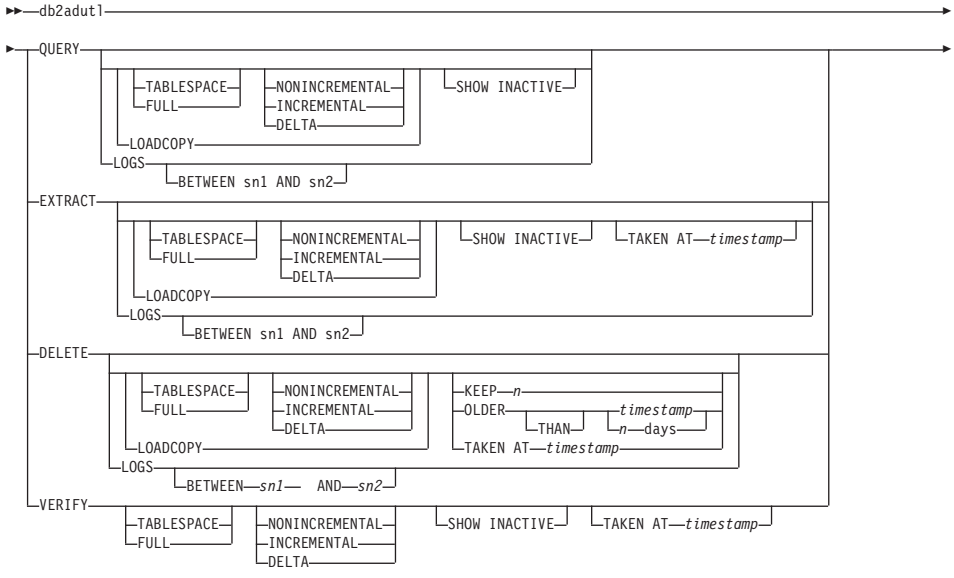
授权:

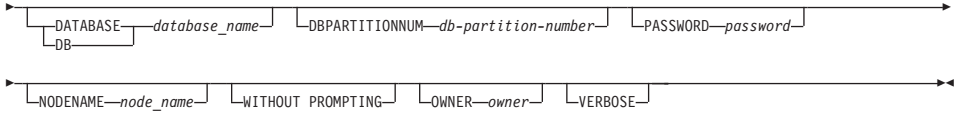
无

必需的连接:

无

命令语法:





命令参数:

QUERY
向 TSM 服务器查询 DB2 对象。

EXTRACT
从 TSM 服务器复制 DB2 对象到本地机器上的当前目录中。

DELETE
在 TSM 服务器上，停用备份对象或删除日志归档。

VERIFY
对服务器上的备份副本执行一致性检查。

注: 此参数使整个备份映象基于网络传送。

TABLESPACE
只包括表空间备份映象。

FULL 只包括完整的数据库备份映象。

NONINCREMENTAL
只包括非增量备份映象。

INCREMENTAL
只包括增量备份映象。

DELTA
只包括增量 Delta 备份映象。

LOADCOPY
只包括装入副本映象。

LOGS 只包括日志归档映象

BETWEEN *sn1* **AND** *sn2*
指定将使用日志序列号 1 和日志序列号 2 之间的日志。

SHOW INACTIVE
包括已被停用的备份对象。

TAKEN AT *timestamp*
按时间戳记指定备份映象。

KEEP *n*
按时间戳记，取消激活指定类型的全部对象，除最新的 *n* 之外。

OLDER THAN *timestamp* **或** *n* **days**
指定将取消激活时间戳记早于 *timestamp* 或 *n* 天的对象。

DATABASE *database_name*
只考虑那些与指定的数据库名相关的对象。

DBPARTITIONNUM *db-partition-number*
只考虑那些根据指定的数据库分区号创建的对象。

PASSWORD *password*
指定此节点的 TSM 客户机密码，如果需要的话。如果指定了一个数据库，但未提供密码，对 *tsm_password* 数据库配置参数指定的值将发送给 TSM；否则将不使用密码。

NODENAME *node_name*
只考虑那些与特定 TSM 节点名相关的映象。

WITHOUT PROMPTING
将不提示用户在删除对象前进行验证。

OWNER *owner*
只考虑那些由指定的所有者创建的对象。

VERBOSE
显示附加文件信息

示例:

以下是来自 `db2 backup database rawsampl use tsm` 的样本输出:

```
Backup successful. The timestamp for this backup is : 19970929130942
  db2adutl query

Query for database RAWSAMPL

Retrieving full database backup information.
   full database backup image: 1, Time: 19970929130942,
                                  Oldest log: S0000053.LOG, Sessions used: 1
   full database backup image: 2, Time: 19970929142241,
                             Oldest log: S0000054.LOG, Sessions used: 1

Retrieving table space backup information.
   table space backup image: 1, Time: 19970929094003,
                             Oldest log: S0000051.LOG, Sessions used: 1
   table space backup image: 2, Time: 19970929093043,
                             Oldest log: S0000050.LOG, Sessions used: 1
   table space backup image: 3, Time: 19970929105905,
                             Oldest log: S0000052.LOG, Sessions used: 1

Retrieving log archive information.
   Log file: S0000050.LOG
```

```
   Log file: S0000051.LOG
   Log file: S0000052.LOG
   Log file: S0000053.LOG
   Log file: S0000054.LOG
   Log file: S0000055.LOG
```

以下是来自 db2adutl delete full taken at 19950929130942 db rawsampl 的样本输出:

```
Query for database RAWSAMPL

Retrieving full database backup information.  Please wait.

   full database backup image: RAWSAMPL.0.db26000.0.19970929130942.001

    Do you want to deactivate this backup image (Y/N)? y

      Are you sure (Y/N)? y

  db2adutl query

Query for database RAWSAMPL

Retrieving full database backup information.
   full database backup image: 2, Time: 19950929142241,
                               Oldest log: S0000054.LOG, Sessions used: 1

Retrieving table space backup information.
   tablespace backup image: 1, Time: 19950929094003,
                            Oldest log: S0000051.LOG, Sessions used: 1
   tablespace backup image: 2, Time: 19950929093043,
                            Oldest log: S0000050.LOG, Sessions used: 1
   tablespace backup image: 3, Time: 19950929105905,
                            Oldest log: S0000052.LOG, Sessions used: 1

Retrieving log archive information.
   Log file: S0000050.LOG
   Log file: S0000051.LOG
   Log file: S0000052.LOG
   Log file: S0000053.LOG
   Log file: S0000054.LOG
   Log file: S0000055.LOG
```

使用注意事项:

下面每组的一个参数可以用来限制操作中包括哪些备份映象类型:

粒度:

- FULL — 只包括数据库备份映象。
- TABLESPACE — 只包括表空间备份映象。

累积度:

- NONINCREMENTAL — 只包括非增量备份映象。
- INCREMENTAL — 只包括增量备份映象。
- DELTA — 只包括增量 Delta 备份映象。

兼容性:

对于与版本 8 之前的版本的兼容性:

- 可以用关键字 NODE 来替代 DBPARTITIONNUM。

db2ckbkp — 检查备份

此实用程序可用于测试备份映象的完整性，并确定是否可复原该映象。还可用它来显示存储在备份头中的元数据。

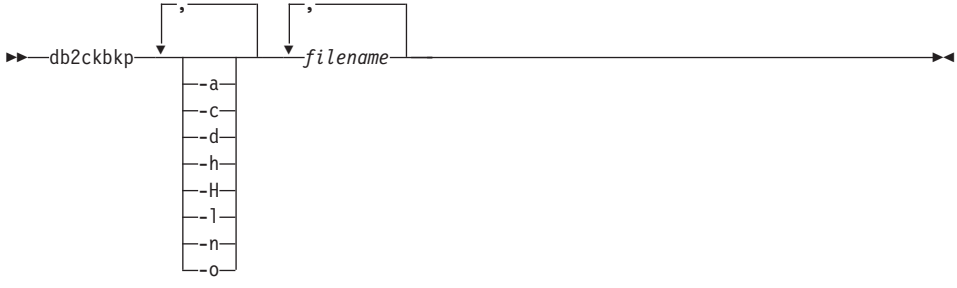
授权:

任何人都可以访问该实用程序，但用户必须对映象备份有读许可权才能对它们执行此实用程序。

必需的连接:

无

命令语法:



命令参数:

-a 显示所有可用的信息。

-c 显示检查位和检查和的结果。

-d 显示来自 DMS 表空间数据页的头部分的信息。

-h 显示媒体头信息，包括复原实用程序期望的映象的名称和路径。

-H 与 -h 显示相同信息，但仅从映象的开始部分读取 4K 媒体头信息。它不验证映象。

注: 不能将此选项与任何其它选项组合使用。

-l 显示日志文件头数据。

-n 提示安装磁带。假定每个设备一个磁带。

-o 显示对象头部分中的详细信息。

filename

备份映象文件的名称。可同时检查一个或多个文件。

注:

1. 如果完整的备份由多个对象构成，只有在将 **db2ckbkp** 用于同时验证所有对象时，该验证才会成功。
2. 检查一个映象的多个部分时，必须首先指定第一个备份映象对象（.001）。

示例:

```
db2ckbkp SAMPLE.0.krodger.NODE0000.CATN0000.19990817150714.*
[1] Buffers processed:  ##
[2] Buffers processed:  ##
[3] Buffers processed:  ##
Image Verification Complete - successful.

db2ckbkp -h SAMPLE2.0.krodger.NODE0000.CATN0000.19990818122909.001


=====================
MEDIA HEADER REACHED:
=====================
        Server Database Name           -- SAMPLE2
        Server Database Alias          -- SAMPLE2
        Client Database Alias          -- SAMPLE2
        Timestamp                      -- 19990818122909
        Database Partition Number      -- 0
        Instance                       -- krodger
        Sequence Number                -- 1
        Release ID                     -- 900
        Database Seed                  -- 65E0B395
        DB Comment's Codepage (Volume) -- 0
        DB Comment (Volume)            --
        DB Comment's Codepage (System) -- 0
        DB Comment (System)            --
        Authentication Value           -- 255
        Backup Mode                    -- 0
        Backup Type                    -- 0
        Backup Gran.                   -- 0
        Status Flags                   -- 11
        System Cats inc                -- 1
        Catalog Database Partition No. -- 0
        DB Codeset                     -- ISO8859-1
        DB Territory                   --
```

```
        Backup Buffer Size             -- 4194304
        Number of Sessions             -- 1
        Platform                       -- 0

The proper image file name would be:
SAMPLE2.0.krodger.NODE0000.CATN0000.19990818122909.001

[1] Buffers processed:  ####
Image Verification Complete - successful.
```

使用注意事项:

1. 如果备份映象使用多个会话创建的，**db2ckbkp** 可同时检查所有文件。用户负责确保序列号为 001 的会话是指定的第一个文件。
2. 此实用程序也可以验证存储在磁带上的备份映象（那些用变量块大小创建的映象除外）。通过为复原操作准备磁带、然后调用实用程序再指定磁带机名称来实现。例如，在基于 UNIX 的系统上:

   ```
   db2ckbkp -h /dev/rmt0
   ```

   在 Windows 上:

   ```
   db2ckbkp -d \\.\tape1
   ```

3. 如果映象在磁带机上，则指定磁带机路径。将提示您务必安装磁带，除非给定选项 “-n”。如果有多个磁带，则必须在给定的第一个设备路径上安装第一个磁带。（这是头中具有序列 001 的磁带）。

   检测磁带机时的缺省情况是提示用户安装磁带。用户可以在提示下进行选择。以下是提示和选项: （其中指定的设备 I 在设备路径 /dev/rmt0 上）

   ```
    请在设备 /dev/rmt0 上安装源媒体。继续（c）、
仅终止此设备（d）或异常终止此工具（t）？（c/d/t）
   ```

   对于每个指定的设备，且在该设备到达磁带结尾时，都将提示用户。

相关参考:

- 第 185 页的『db2adutl — 使用 TSM 归档映象』

db2ckrst — 检查增量复原映象序列

查询数据库历史，并生成增量复原所需的备份映象的时间戳记列表。同时还生成用于手工增量复原的简化了的复原语法。

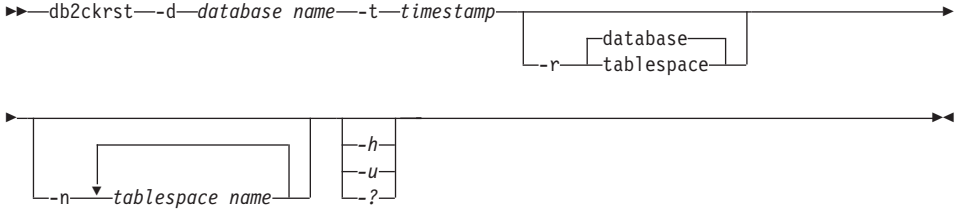
授权:

无

必需的连接:

无

命令语法:



命令参数:

-d database name file-name
指定将复原的数据库的别名。

-t timestamp
指定将增量复原的备份映象的时间戳记。

-r 指定将要执行的复原的类型。缺省值是数据库复原。

注: 如果选择了表空间但未给出表空间名，则实用程序会查找指定映象的历史条目并使用列出的要执行复原的表空间名。

-n tablespace name
指定将复原的一个或多个表空间的名称。

注: 如果选择了数据库复原类型并指定了表空间名的列表，则实用程序将使用给定的表空间名继续执行表空间复原操作。

-h/-u/-?
显示帮助信息。当指定了此选项时，其它所有的选项都会被忽略，且只显示帮助信息。

示例:

```
db2ckrst -d mr -t 20001015193455 -r database
db2ckrst -d mr -t 20001015193455 -r tablespace
db2ckrst -d mr -t 20001015193455 -r tablespace -n tbsp1 tbsp2

   > db2 backup db mr

Backup successful. The timestamp for this backup image is : 20001016001426

   > db2 backup db mr incremental

Backup successful. The timestamp for this backup image is : 20001016001445

   > db2ckrst -d mr -t 20001016001445
```

```
 Suggested restore order of images using timestamp 20001016001445 for
 database mr.
 ==========================================================================
db2 restore db mr incremental taken at 20001016001445
db2 restore db mr incremental taken at 20001016001426
db2 restore db mr incremental taken at 20001016001445
 ==========================================================================

 > db2ckrst -d mr -t 20001016001445 -r tablespace -n userspace1
 Suggested restore order of images using timestamp 20001016001445 for
 database mr.
 ==========================================================================
   db2 restore db mr tablespace ( USERSPACE1 ) incremental taken at
   20001016001445
   db2 restore db mr tablespace ( USERSPACE1 ) incremental taken at
   20001016001426
   db2 restore db mr tablespace ( USERSPACE1 ) incremental taken at
   20001016001445
 ==========================================================================
```

使用注意事项:

数据库历史必须存在，此实用程序才能使用。如果数据库历史不存在，则在使用此实用程序之前在 RESTORE 命令中指定 HISTORY FILE 选项。

如果使用 PRUNE HISTORY 命令的 FORCE 选项，则将能够删除从最新的完整数据库备份映象进行恢复所需的条目。PRUNE HISTORY 命令的缺省操作可防止所需的条目被删除。建议不要使用 PRUNE HISTORY 命令的 FORCE 选项。

不应该将此实用程序用作保留备份记录的替代方法。

db2flsn — 查找日志序列号

返回包含由指定的日志序列号（LSN）标识的日志记录的文件名。

授权:

无

命令语法:

命令参数:

-q 指定只打印日志文件名。将不打印任何错误或警告消息，状态只能通过返回码来确定。有效的错误码是:

- -100 无效输入

- -101 打不开 LFH 文件
- -102 未成功读取 LFH 文件
- -103 无效的 LFH
- -104 数据库不是可复原
- -105 LSN 太大
- -500 逻辑错误。

其它有效的返回码是:

- 0 成功执行
- 99 警告: 结果基于最后一个已知的日志文件大小。

input_LSN

一个 12 字节的字符串，它代表有前导零的内部（6 字节）十六进制值。

示例:

```
db2flsn 000000BF0030
   Given LSN is contained in log file S0000002.LOG

db2flsn -q 000000BF0030
   S0000002.LOG

db2flsn 000000BE0030
   Warning: the result is based on the last known log file size.
   The last known log file size is 23 4K pages starting from log extent 2.

   Given LSN is contained in log file S0000001.LOG

db2flsn -q 000000BE0030
   S0000001.LOG
```

使用注意事项:

日志头控制文件 SQLOGCTL.LFH 必须驻留在当前目录中。由于此文件位于数据库目录中，所以工具可以从数据库目录运行，或者控制文件可从运行工具的目录复制到该目录。

工具使用 *logfilsiz* 数据库配置参数。DB2 记录此参数的 3 个最近值，以及用每个 *logfilsiz* 值创建的第一个日志文件，从而使工具可在 *logfilsiz* 更改时正确工作。如果指定的 LSN 比 *logfilsiz* 的最早记录值要早，工具会使用该值并返回一个警告。该工具可与 UDB 版本 5.2 之前的数据库管理器一起使用；此时，即使结果正确（如果 *logfilsiz* 的值保持不变）也会返回警告。

此工具只能用于可复原的数据库。如果在配置数据库时，*logretain* 设置为 RECOVERY 或 *userexit* 设置为 ON，该数据库就是可复原的。

db2inidb — 初始化镜像数据库

在分割镜像环境中初始化镜像数据库。镜像数据库可以初始化为主数据库的克隆、置于前滚暂挂状态或用作要复原主数据库的备份映象。

授权:

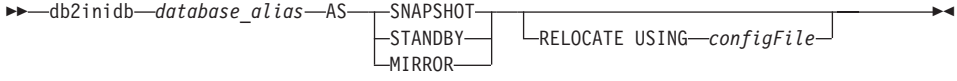
以下其中一项:

- *sysadm*
- *sysctrl*
- *sysmaint*

必需的连接:

无

命令语法:



命令参数:

database_alias
指定要初始化的数据库的别名。

SNAPSHOT
指定要初始化为主数据库的克隆的镜像数据库。

STANDBY
指定要置于前滚暂挂状态的数据库。

注: 可访存主数据库中的新日志，并应用到备用数据库。然后可在主数据库当机时，将该备用数据库用作主数据库。

MIRROR
指定要用作备份映象的镜像数据库，该备份映象可以用来复原主数据库。

RELOCATE USING configFile
指定要根据配置文件中列示的信息重新查找数据库文件。

相关参考:

- 『db2relocatedb - Relocate Database Command』（*Command Reference*）

db2mscs — 设置 Windows 故障转移实用程序

为使用 Microsoft Cluster Server（MSCS） 的 Windows 上的 DB2 故障转移支持创建基础结构。可使用此实用程序，以在单分区和分区数据库环境中都启用故障转移。

授权:

用户必须登录到属于 MSCS 群集中每个机器的 Administrators 组的域用户帐户。

命令语法:

命令参数:

-f:input_file

指定 MSCS 实用程序要使用的 DB2MSCS.CFG 输入文件。若未指定此参数，则 DB2MSCS 实用程序读取当前目录中的 DB2MSCS.CFG 文件。

-u:instance_name

此选项允许您撤销 db2mscs 操作，并将实例回复为由 instance_name 指定的非 MSCS 实例。

使用注意事项:

DB2MSCS 实用程序是用来将非 MSCS 实例变换为 MSCS 实例的独立命令行实用程序。该实用程序将创建所有 MSCS 组、资源和资源相关性。它还将存储在 Windows 注册表中的所有 DB2 信息复制至注册表的群集部分，并将实例目录移至共享群集磁盘。DB2MSCS 实用程序将输入视为由用户提供的指定应该如何设置群集的配置文件。DB2MSCS.CFG 文件是 ASCII 文本文件，它包含由 DB2MSCS 实用程序读取的参数。使用以下格式: PARAMETER_KEYWORD=parameter_value 在单独的一行上指定每个输入参数。例如:

```
CLUSTER_NAME=FINANCE
GROUP_NAME=DB2 Group
IP_ADDRESS=9.21.22.89
```

可在 DB2 安装目录下的 CFG 子目录中找到两个示例配置文件。第一个文件 DB2MSCS.EE 是单分区数据库环境的示例。第二个文件 DB2MSCS.EEE 是分区数据库环境的示例。

DB2MSCS.CFG 文件的参数如下所示:

DB2_INSTANCE

DB2 实例的名称。此参数具有全局作用域，且在 DB2MSCS.CFG 文件中应该只指定一次。

DAS_INSTANCE

“DB2 管理服务器”实例的名称。指定此参数以迁移要在 MSCS 环境中运行的“DB2 管理服务器”。此参数具有全局作用域，且在 DB2MSCS.CFG 文件中应该只指定一次。

CLUSTER_NAME

MSCS 群集名。只有指定了另一个 CLUSTER_NAME 参数后，才会在此群集中创建在此行后指定的所有资源。

DB2_LOGON_USERNAME

DB2 服务的域帐户的用户名（例如，domain\user）。此参数具有全局作用域，且在 DB2MSCS.CFG 文件中应该只指定一次。

DB2_LOGON_PASSWORD

DB2 服务的域帐户的密码。此参数具有全局作用域，且在 DB2MSCS.CFG 文件中应该只指定一次。

GROUP_NAME

MSCS 组名。如果指定了此参数，会创建一个新的 MSCS 组（如果它不存在的话）。如果该组已存在，则将它用作目标组。只有在指定了另一个 GROUP_NAME 参数后，才会在此组中创建此参数后指定的任何 MSCS 资源或将这些资源移至此组。对每个组，指定此参数一次。

DB2_NODE

要包括在当前 MSCS 组中的数据库分区服务器（或数据库分区）的分区号。如果同一机器上存在多个逻辑数据库分区，则每个数据库分区都需要一个单独的 DB2_NODE 参数。在 GROUP_NAME 参数后指定此参数，以便在正确的 MSCS 组中创建 DB2 资源。多分区数据库系统需要此参数。

IP_NAME

“IP 地址”资源的名称。IP_NAME 的值是随机的，但它在群集中必须是唯一的。当指定此参数时，会创建一个类型为“IP 地址”的 MSCS 资源。远程 TCP/IP 连接需要此参数。在单分区环境中，此参数是可选的。建议的名称是与 IP 地址相对应的主机名。

IP_ADDRESS

前导 IP_NAME 参数指定的 IP 资源的 TCP/IP 地址。如果指定了 IP_NAME 参数，则此参数是必需的。这是不由网络中任何机器使用的新 IP 地址。

IP_SUBNET

前导 IP_NAME 参数指定的 IP 资源的 TCP/IP 子网掩码。如果指定了 IP_NAME 参数，则此参数是必需的。

IP_NETWORK

前导“IP 地址”资源所属的 MSCS 网络的名称。此参数是可选的。如果不指定此参数，则使用系统检测到的第一个 MSCS 网络。必须完全按照“群集管理员”中的“网络”分支下面所示的内容来输入 MSCS 网络的名称。

注: 前四个 IP 关键字用来创建“IP 地址”资源。

NETNAME_NAME

“网络名”资源的名称。指定此参数以创建“网络名”资源。在单分区数据库环境中，此参数是可选的。必须为分区数据库环境中拥有实例的机器指定此参数。

NETNAME_VALUE

“网络名”资源的值。如果指定了 NETNAME_NAME 参数，则必须指定此参数。

NETNAME_DEPENDENCY

“网络名”资源从属的 IP 资源的名称。每个“网络名”资源都必须具有与“IP 地址”资源的相关性。此参数是可选的。如果不指定此参数，则“网络名”资源在组中具有与第一个 IP 资源的相关性。

SERVICE_DISPLAY_NAME

“一般服务”资源的显示名。如果想要创建“一般服务”资源，则指定此参数。

SERVICE_NAME

“一般服务”资源的服务名。如果指定了 SERVICE_DISPLAY_NAME 参数，则必须指定此参数。

SERVICE_STARTUP

“一般资源”服务的可选启动参数。

DISK_NAME

要移至当前组的物理磁盘资源的名称。指定所需的尽量多的磁盘资源。磁盘资源必须已存在。当 DB2MSCS 实用程序配置 DB2 实例以提供故障转移支持时，将该实例目录复制至组中的第一个 MSCS 磁盘。要为该实例目录指定另一 MSCS 磁盘，使用 INSTPROF_DISK 参数。应该完全按照“群集管理员”中所示的内容来输入使用的磁盘名。

INSTPROF_DISK

一个可选的参数，指定包含 DB2 实例目录的 MSCS 磁盘。如果不指定此参数，则 DB2MSCS 实用程序使用属于同一组的第一个磁盘。

INSTPROF_PATH

一个可选的参数，它指定要将实例目录复制至其中的确切路径。当使用 IPSHAdisks（一种 ServerRAID Netfinity 磁盘资源）时，必须指定此参数（例如 INSTPROF_PATH=p:\db2profs）。如果同时指定了 INSTPROF_PATH 与 INSTPROF_DISK，则优先使用前者。

TARGET_DRVMAP_DISK

一个可选的参数，它对多分区数据库系统指定数据库驱动器映射的目标 MSCS 磁盘。此参数指定将在其上创建数据库的磁盘，方法是从创建数据库命令指定的驱动器映射该数据库。如果未指定此参数，必须使用 DB2DRVMP 实用程序手工注册数据库驱动器映射。

DB2_FALLBACK

一个可选的参数，它控制使 DB2 资源脱机时是否应该强制关闭应用程序。如果不指定此项，则 DB2_FALLBACK 的设置将为 YES。如果不想强制关闭应用程序，则将 DB2_FALLBACK 设置为 NO。

CLP 命令

ARCHIVE LOG

关闭或截断某个可恢复数据库的活动日志文件。如果启用了用户出口，则发出归档请求。

授权:

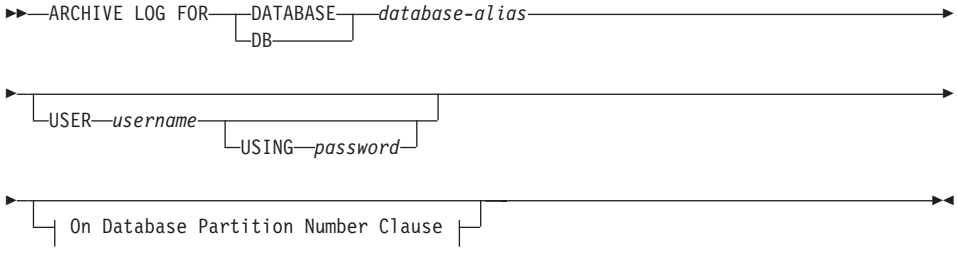
以下其中一项:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*

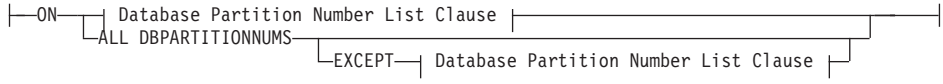
必需的连接:

无。此命令在命令执行期间建立数据库连接。

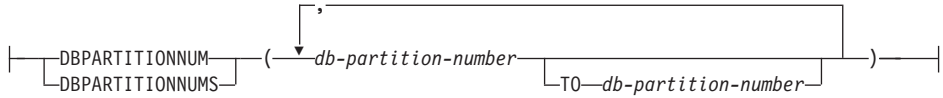
命令语法:



On Database Partition Number Clause:



Database Partition Number List Clause:



命令参数:

DATABASE database-alias
指定要归档其活动日志的数据库的别名。

USER username
标识尝试连接时要使用的用户名。

USING password
指定认证用户名的密码。

ON ALL DBPARTITIONNUMS
指定应对 db2nodes.cfg 文件中的所有数据库分区发出该命令。如果不指定数据库分区号子句，则这是缺省值。

EXCEPT
指定应对 db2nodes.cfg 文件中的所有数据库分区发出该命令，在数据库分区号列表中指定的分区除外。

ON DBPARTITIONNUM/ON DBPARTITIONNUMS
指定应该对一组数据库分区上的指定数据库归档日志。

db-partition-number
在数据库分区号列表中指定数据库分区号。

TO db-partition-number

在指定应对其归档日志的数据库分区范围时使用。从指定的第一个数据库分区号直到并包括指定的第二个数据库分区号之间的所有数据库分区都包括在数据库分区号列表中。

使用注意事项:

此命令可用于收集到某个已知点为止的一套完整的日志文件。然后日志文件可用于更新备用数据库。

仅当调用应用程序或外壳程序不与指定数据库进行数据库连接时，才能执行此命令。这防止用户对未落实的事务执行该命令。因此，ARCHIVE LOG 命令将不会强制落实用户的未完成事务。如果调用应用程序或外壳程序已与特定数据库进行了数据库连接，则该命令将终止并返回错误。如果在执行此命令时，另一个应用程序正对指定数据库执行事务，则性能会有所降低，因为该命令将日志缓冲区清仓至磁盘。尝试将日志记录写入缓冲区的任何其它事务将必须等待直到清仓完成为止。

如果在分区数据库环境中使用此命令，则可以通过使用数据库分区号子句来指定数据库分区的子集。如果未指定数据库分区号子句，则此命令的缺省行为将关闭所有数据库分区上的活动日志并将其并归档。

由于活动日志文件被截断，使用此命令将用完活动日志空间的部分。当截断的日志变得不活动时，活动日志空间将恢复其先前大小。经常使用此命令将显著降低可供事务使用的活动日志空间量。

兼容性:

对于与版本 8 之前的版本的兼容性:

- 可以用关键字 NODE 来替代 DBPARTITIONNUM。
- 可以用关键字 NODES 来替代 DBPARTITIONNUMS。

INITIALIZE TAPE

当在基于 Windows NT 的操作系统上运行时，DB2 支持对流式磁带机的备份与复原操作。使用此命令进行磁带初始化。

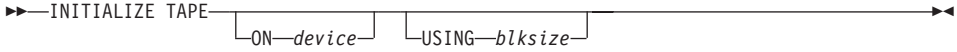
授权:

无

必需的连接:

无

命令语法:



命令参数:

ON device

指定有效的磁带机名称。缺省值是 \\.\TAPE0。

USING blksize

以字节为单位，指定设备的块大小。如果该值在设备受支持的块大小范围内，则初始化设备以使用指定的块大小。

注: 对 BACKUP DATABASE 命令和 RESTORE DATABASE 命令指定的缓冲区大小必须可被此处指定的块大小除尽。

如果未指定此参数的值，将初始化设备以使用它的缺省块大小。如果指定了值 0，会初始化该设备以使用变量长度块大小；如果设备不支持变量长度块方式，将返回一个错误。

相关参考:

- 第 63 页的『BACKUP DATABASE』
- 第 84 页的『RESTORE DATABASE』
- 第 206 页的『REWIND TAPE』
- 第 207 页的『SET TAPE POSITION』

LIST HISTORY

列示历史文件中的条目。历史文件包含对恢复和管理事件的记录。恢复事件包括完整的数据库和表空间级的备份、增量备份以及复原和前滚操作。记录的其它事件包括：创建、改变、删除或重命名表空间、重新组织表、删除表和装入。

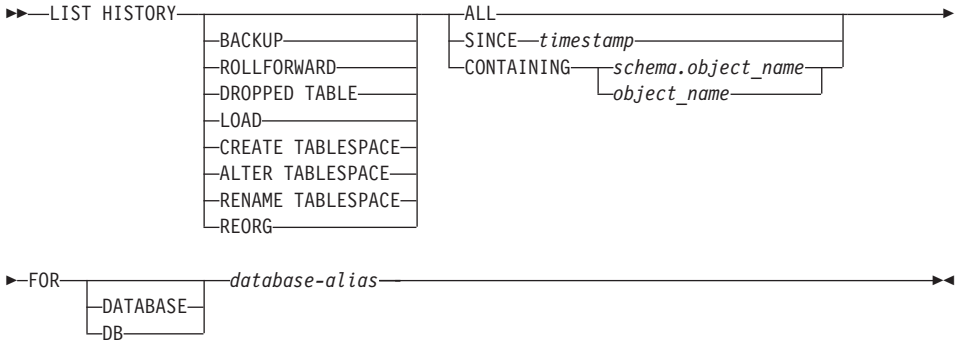
授权:

无

必需的连接:

实例。不需要显式连接。如果数据库列示为远程数据库，会在命令执行期间建立与远程节点的实例连接。

命令语法:



命令参数:

HISTORY
列出当前记录在历史文件中的全部事件。

BACKUP
列出备份与复原操作。

ROLLFORWARD
列出前滚操作。

DROPPED TABLE
列出已删除的表记录。

LOAD 列出装入操作。

CREATE TABLESPACE
列出表空间创建和删除运操作。

RENAME TABLESPACE
列出表空间重命名操作。

REORG
列示重新组织操作。

ALTER TABLESPACE
列出改变表空间操作。

ALL 列示历史文件中具有指定类型的所有条目。

SINCE timestamp
可指定的完整时间戳记（格式为 *yyyymmddhhnnss*），或初始前缀（至少为 *yyyy*）。时间戳记等于或大于所提供的时间戳记的所有条目都将列出。

CONTAINING schema.object_name
此限定名唯一地标识表。

CONTAINING object_name

此不限定名唯一地标识表空间。

FOR DATABASE database-alias

用于标识将列出其恢复历史文件的数据库。

示例:

```
db2 list history since 19980201 for sample
db2 list history backup containing userspace1 for sample
db2 list history dropped table all for db sample
```

使用注意事项:

此命令生成的报告包含以下符号:

操作

- A — 创建表空间
- B — 备份
- C — 装入副本
- D — 删除的表
- F — 前滚
- G — 重组表
- L — 装入
- N — 重命名表空间
- O — 删除表空间
- Q — 停顿
- R — 复原
- T — 改变表空间
- U — 卸装

类型
备份类型:

- F — 脱机
- N — 联机
- I — 增量脱机
- O — 增量联机
- D — Delta 脱机
- E — Delta 联机

前滚类型:

- E — 日志结束
- P — 时间点

装入类型:

- I — 插入
- R — 替换

改变表空间类型:

- C — 添加容器

R — 重新平衡

停顿类型:

S — 停顿共享
U — 停顿更新
X — 停顿铬占
Z — 停顿复位

PRUNE HISTORY/LOGFILE

用于从恢复历史文件中删除条目，或从活动的日志文件路径中删除日志文件。如果文件变得特别大而保留期又特别长，可能会需要从恢复历史文件中删除条目。如果日志是手工归档的（而不是通过用户出口程序归档的），则可能需要从活动的日志文件路径中删除日志文件。

授权:

以下其中一项:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*

必需的连接:

数据库

命令语法:

命令参数:

HISTORY timestamp

标识将删除的恢复历史文件中条目的范围。可指定一个完整的时间戳记（格式为 *yyyymmddhhmmss*）或初始前缀（最少为 *yyyy*）。时间戳记等于或小于所提供的时间戳记的所有条目都将从恢复历史文件中删除。

WITH FORCE OPTION

指定将根据指定的时间戳记来修剪条目，即使会从该文件中删除最新复原设置中的某些条目。复原集是最新的完整数据库备份，包括该备份映象的任何复原。如果未指定此参数，来自先前备份映象的所有条目都将保留在该历史中。

LOGFILE PRIOR TO log-file-name

指定一个字符串作为日志文件名，例如 S0000100.LOG。将删除指定的日志文件前（但不包括）的所有日志文件。LOGRETAIN 数据库配置参数必须设置为 RECOVERY 或 CAPTURE。

示例:

要从恢复历史文件中除去在 1994 年 12 月 1 日前（包括这一天）建立的所有复原、装入、表空间备份以及完整数据库备份的条目，可输入:

```
db2 prune history 199412
```

注: 将 199412 解释为 19941201000000。

使用注意事项:

来自历史文件的修剪备份条目会导致 DB2 Data Links Manager 服务器上的相关文件备份被删除。

REWIND TAPE

当在基于 Windows NT 的操作系统上运行时，DB2 支持对流式磁带机的备份与复原操作。使用此命令进行磁带倒带。

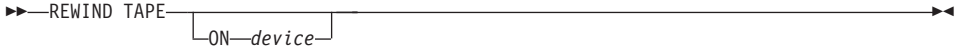
授权:

无

必需的连接:

无

命令语法:



命令参数:

ON device

指定有效的磁带机名称。缺省值是 \\.\TAPE0。

相关参考:

- 第 201 页的『INITIALIZE TAPE』
- 第 207 页的『SET TAPE POSITION』

SET TAPE POSITION

当在基于 Windows NT 的操作系统上运行时，DB2 支持对流式磁带机的备份与复原操作。使用此命令进行磁带定位。

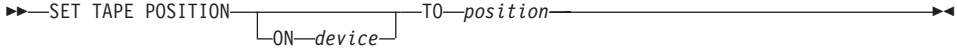
授权:

无

必需的连接:

无

命令语法:



命令参数:

ON device
指定有效的磁带机名称。缺省值是 `\\.\TAPE0`。

TO position
指定将定位磁带的标记。DB2 Windows NT/2000 版在每次备份映象后写磁带标记。值为 1 指定第一个位置，而 2 则指定第二个位置，以此类推。例如，如果将磁带定位在磁带标记 1，会定位归档 2 以进行复原。

相关参考:

- 第 201 页的『INITIALIZE TAPE』
- 第 206 页的『REWIND TAPE』

UPDATE HISTORY FILE

更新历史文件条目中的位置、设备类型或注释。

授权:

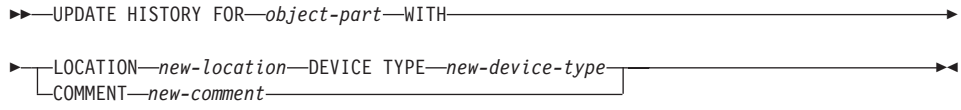
以下其中一项:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*

必需的连接:

数据库

命令语法:



命令参数:

FOR object-part

指定备份或复制映象的标识符。它是一个时间戳记，并带有从 001 到 999 的可选序列号。

LOCATION new-location

指定备份映象的新物理位置。对此参数的解释取决于设备类型。

DEVICE TYPE new-device-type

指定用于存储备份映象的新设备类型。有效的设备类型是:

D 磁盘

K 软盘

T 磁带

A TSM

U 用户出口

P 管道

N Null 设备

X XBSA

Q SQL 语句

O 其它

COMMENT new-comment

指定描述该条目的新注释。

示例:

要更新在 1997 年 4 月 13 日，上午 10:00 所建立的完整数据库备份的历史文件条目，可输入:

```
db2 update history for 19970413100000001 with
   location /backup/dbbackup.1 device type d
```

使用注意事项:

该历史文件由数据库管理员用于保留记录。它由 DB2 在内部用于自动恢复增量备份。

相关参考:

- 第 205 页的『PRUNE HISTORY/LOGFILE』

附录 D. 附加 API 及相关数据结构

db2ArchiveLog — 归档活动日志 API

关闭或截断某个可恢复数据库的活动日志文件。如果启用了用户出口，发出归档请求。

作用域

在 MPP 环境中，此 API 关闭并截断所有节点上的活动日志。

权限

以下其中一项:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*

需要的连接

此 API 自动建立与指定数据库的连接。如果连接已存在，会返回错误。

API 包含文件

db2ApiDf.h

C API 语法

```
/* File: db2ApiDf.h */
/* API: Archive Active Log */
/* ... */
SQL_API_RC SQL_API_FN
   db2ArchiveLog (
      db2Uint32 version,
      void * pDB2ArchiveLogStruct,
      struct sqlca * pSqlca);

typedef struct
{
   char * piDatabaseAlias;
   char * piUserName;
   char * piPassword;
   db2Uint16 iAllNodeFlag;
   db2Uint16 iNumNodes;
   SQL_PDB_NODE_TYPE * piNodeList;
   db2Uint32 iOptions;
} db2ArchiveLogStruct;
/* ... */
```

一般 API 语法

```
/* File: db2ApiDf.h */
/* API: Archive Active Log */
/* ... */
SQL_API_RC SQL_API_FN
   db2gArchiveLog (
      db2Uint32 version,
      void * pDB2gArchiveLogStruct,
      struct sqlca * pSqlca);

typedef struct
{
   db2Uint32 iAliasLen;
   db2Uint32 iUserNameLen;
   db2Uint32 iPasswordLen;
   char * piDatabaseAlias;
   char * piUserName;
   char * piPassword;
   db2Uint16 iAllNodeFlag;
   db2Uint16 iNumNodes;
   SQL_PDB_NODE_TYPE * piNodeList;
   db2Uint32 iOptions;
} db2gArchiveLogStruct;
/* ... */
```

API 参数

version

输入。指定作为第二个参数 *pDB2ArchiveLogStruct* 传入的变量的版本和发行级别。

pDB2ArchiveLogStruct

输入。指向 *db2ArchiveLogStruct* 结构的指针。

pSqlca

输出。指向 *sqlca* 结构的一个指针。

iAliasLen

输入。一个 4 字节的无符号整数，代表以字节计的数据库别名的长度。

iUserNameLen

输入。一个 4 字节的无符号整数，代表以字节计的用户名的长度。如果不使用用户名，则设置为零。

iPasswordLen

输入。一个 4 字节的无符号整数，代表以字节计的密码的长度。如果不使用密码，则设置为零。

piDatabaseAlias

输入。一个包含数据库别名的字符串（如系统数据库目录中编目的那样），将对该数据库进行活动日志归档。

piUserName

输入。包含尝试连接时将使用的用户名的字符串。

piPassword

输入。包含尝试连接时将使用的密码的字符串。

iAllNodeFlag

输入。仅用于 MPP。标志，它指示是否应对 `db2nodes.cfg` 文件中的所有节点应用该操作。有效的值为:

DB2ARCHIVELOG_ALL_NODES

对所有节点应用（piNodeList 应为 NULL）。这是缺省值。

DB2ARCHIVELOG_NODE_LIST

对发送到 *piNodeList* 中的节点列表中指定的所有节点应用。

DB2ARCHIVELOG_ALL_EXCEPT

对除发送到 *piNodeList* 中的节点列表中指定的那些节点以外的所有节点应用。

iNumNodes

输入。仅用于 MPP。指定 *piNodeList* 数组中的节点号。

piNodeList

输入。仅用于 MPP。指向要应用归档日志操作的节点号数组的指针。

iOptions

输入。保留以备将来使用。

用法注释

此 API 可用于收集到某个已知点为止的日志文件的完整集合。然后日志文件可用于更新备用数据库。

如果调用此 API 时，其它应用程序已有事务在进行中，将日志缓冲区刷新到磁盘中时，会发现性能略有降低；其它事务尝试将日志记录写到缓冲区必须等待刷新完成。

此 API 会导致数据库丢失其 LSN 空间的一部分，因此加速消耗有效的 LSN。

db2HistoryCloseScan — 关闭历史文件扫描

结束历史文件扫描，并释放扫描所需的 DB2 资源。必须在成功调用 db2HistoryOpenScan 之后才能调用此 API。

授权:

无

必需的连接:

实例。不必在调用此 API 之前调用 sqleatin。

API 包含文件:

db2ApiDf.h

C API 语法:

```
/* File: db2ApiDf.h */
/* API: db2HistoryCloseScan */
/* ... */
SQL_API_RC  SQL_API_FN
  db2HistoryCloseScan (
    db2Uint32 version,
    void *piHandle,
      struct sqlca * pSqlca);
/* ... */
```

一般 API 语法:

```
/* File: db2ApiDf.h */
/* API: db2GenHistoryCloseScan */
/* ... */
SQL_API_RC  SQL_API_FN
  db2GenHistoryCloseScan (
    db2Uint32 version,
    void *piHandle,
      struct sqlca * pSqlca);
/* ... */
```

API 参数:

version
输入。指定第二个参数 *piHandle* 的版本和发行版级别。

piHandle
输入。指定指向由 db2HistoryOpenScan 返回的扫描存取的句柄的指针。

pSqlca

输出。指向 *sqlca* 结构的指针。

REXX API 语法:

```
CLOSE RECOVERY HISTORY FILE :scanid
```

REXX API 参数:

scanid

包含从 OPEN RECOVERY HISTORY FILE SCAN 返回的扫描标识符的主机变量。

使用注意事项:

有关使用历史文件 API 的详细描述，参见 db2HistoryOpenScan。

相关参考:

- 第 228 页的『db2Prune — 修剪历史文件』
- 第 225 页的『db2HistoryUpdate — 更新历史文件』
- 第 220 页的『db2HistoryOpenScan — 打开历史文件扫描』
- 第 217 页的『db2HistoryGetEntry — 获取下一个历史文件条目』
- 『SQLCA』（*Administrative API Reference*）

相关样本:

- 『dbrecov.sqc -- How to recover a database (C)』
- 『dbrecov.sqC -- How to recover a database (C++)』

db2HistoryGetEntry — 获取下一个历史文件条目

获取历史文件中的下一个条目。必须在成功调用 db2HistoryOpenScan 之后才能调用此 API。

授权:

无

必需的连接:

实例。不必在调用此 API 之前调用 sqleatin。

API 包含文件:

db2ApiDf.h

C API 语法:

```
/* File: db2ApiDf.h */
/* API: db2HistoryGetEntry */
/* ... */
SQL_API_RC SQL_API_FN
  db2HistoryGetEntry (
    db2Uint32 version,
    void *pDB2HistoryGetEntryStruct,
      struct sqlca * pSqlca);

typedef struct
{
  db2Uint16 iHandle,
  db2Uint16 iCallerAction,
  struct db2HistData * pioHistData
} db2HistoryGetEntryStruct;
/* ... */
```

一般 API 语法:

```
/* File: db2ApiDf.h */
/* API: db2GenHistoryGetEntry */
/* ... */
SQL_API_RC SQL_API_FN
  db2GenHistoryGetEntry (
    db2Uint32 version,
    void *pDB2GenHistoryGetEntryStruct,
      struct sqlca * pSqlca);

typedef struct
{
  db2Uint16 iHandle,
  db2Uint16 iCallerAction,
  struct db2HistData * pioHistData
} db2GenHistoryGetEntryStruct;
/* ... */
```

API 参数:

version

输入。指定作为第二个参数 *pDB2HistoryGetEntryStruct* 传入的结构的版本和发行版级别。

pDB2HistoryGetEntryStruct

输入。指向 *db2HistoryGetEntryStruct* 结构的一个指针。

pSqlca

输出。指向 *sqlca* 结构的指针。

iHandle

输入。包含由 db2HistoryOpenScan 返回的扫描存取的句柄。

iCallerAction

输入。指定要采取的操作的类型。有效值（在 db2ApiDf 中定义）是:

DB2HISTORY_GET_ENTRY

获取下一个条目，但不带有任何命令数据。

DB2HISTORY_GET_DDL

只获取前一次访存中的命令数据。

DB2HISTORY_GET_ALL

获取下一个条目，包括所有数据。

pioHistData

输入。指向 *db2HistData* 结构的一个指针。

REXX API 语法:

```
GET RECOVERY HISTORY FILE ENTRY :scanid [USING :value]
```

REXX API 参数:

scanid

包含从 OPEN RECOVERY HISTORY FILE SCAN 返回的扫描标识符的主机变量。

value 一个复合 REXX 主机变量，对其返回历史文件条目信息。在下例中，XXX 表示主机变量名:

XXX.0	变量中第一级元素的数目（总为 15）
XXX.1	表空间元素的数目
XXX.2	已使用的表空间元素的数目
XXX.3	OPERATION（执行的操作类型）
XXX.4	OBJECT（操作的粒度）
XXX.5	OBJECT_PART（时间戳记和序列号）
XXX.6	OPTYPE（操作限定符）
XXX.7	DEVICE_TYPE（所使用的设备的类型）
XXX.8	FIRST_LOG（最早的日志标识）
XXX.9	LAST_LOG（当前日志标识）
XXX.10	BACKUP_ID（备份标识符）

XXX.11	SCHEMA（表名限定符）
XXX.12	TABLE_NAME（装入的表的名称）
XXX.13.0	NUM_OF_TABLESPACES（备份或复原过程中所涉及的表空间号）
XXX.13.1	备份／复原的第一个表空间的名称
XXX.13.2	备份／复原的第二个表空间的名称
XXX.13.3	以此类推
XXX.14	LOCATION（存储备份或副本的位置）
XXX.15	COMMENT（用于描述条目的文本）。

使用注意事项:

返回的记录将使用对 db2HistoryOpenScan 的调用上指定的值进行选择。

有关使用历史文件 API 的详细描述，参见 db2HistoryOpenScan。

相关参考:

- 第 228 页的『db2Prune — 修剪历史文件』
- 第 225 页的『db2HistoryUpdate — 更新历史文件』
- 第 220 页的『db2HistoryOpenScan — 打开历史文件扫描』
- 第 216 页的『db2HistoryCloseScan — 关闭历史文件扫描』
- 『SQLCA』（*Administrative API Reference*）
- 第 241 页的『db2HistData』

相关样本:

- 『dbrecov.sqc -- How to recover a database (C)』
- 『dbrecov.sqC -- How to recover a database (C++)』

db2HistoryOpenScan — 打开历史文件扫描

开始扫描历史文件。

授权:

无

必需的连接:

实例。不必在调用此 API 之前调用 sqleatin。如果数据库编目为远程数据库，会建立与远程节点的实例连接。

API 包含文件:

db2ApiDf.h

C API 语法:

```
/* File: db2ApiDf.h */
/* API: db2HistoryOpenScan */
/* ... */
SQL_API_RC SQL_API_FN
  db2HistoryOpenScan (
    db2Uint32 version,
    void *pDB2HistoryOpenStruct,
      struct sqlca * pSqlca);

typedef struct
{
  char *piDatabaseAlias,
  char *piTimestamp,
  char *piObjectName,
  db2Uint32 oNumRows,
  db2Uint16 iCallerAction,
  db2Uint16 oHandle
} db2HistoryOpenStruct;
/* ... */
```

一般 API 语法:

```
/* File: db2ApiDf.h */
/* API: db2GenHistoryOpenScan */
/* ... */
SQL_API_RC SQL_API_FN
  db2GenHistoryOpenScan (
    db2Uint32 version,
    void *pDB2GenHistoryOpenStruct,
      struct sqlca * pSqlca);

typedef struct
{
  char *piDatabaseAlias,
  char *piTimestamp,
  char *piObjectName,
  db2Uint32 oNumRows,
  db2Uint16 iCallerAction,
  db2Uint16 oHandle
} db2GenHistoryOpenStruct;
/* ... */
```

API 参数:

version

输入。指定作为第二个参数 *pDB2HistoryOpenStruct* 传入的结构的版本和发行版级别。

pDB2HistoryOpenStruct

输入。指向 *db2HistoryOpenStruct* 结构的一个指针。

pSqlca

输出。指向 *sqlca* 结构的指针。

piDatabaseAlias

输入。指向包含数据库别名的字符串的指针。

piTimestamp

输入。指向字符串的一个指针，该字符串指定要用于选择记录的时间戳记。将选择那些时间戳记等于或大于此值的记录。将此参数设置为 NULL，或指向零，可防止过滤使用时间戳记的条目。

piObjectName

输入。指向字符串的一个指针，该字符串指定要用于选择记录的对象名。对象可以是表或表空间。如果是表，必须提供全限定表名。将此参数设置为 NULL，或指向零，可防止过滤使用对象名的条目。

oNumRows

输出。在从 API 返回时，此参数包含匹配历史文件条目的数目。

iCallerAction

输入。指定要采取的操作的类型。有效值（在 db2ApiDf 中定义）是:

DB2HISTORY_LIST_HISTORY

列出当前记录在历史文件中的全部事件。

DB2HISTORY_LIST_BACKUP

列出备份与复原操作。

DB2HISTORY_LIST_ROLLFORWARD

列出前滚操作。

DB2HISTORY_LIST_DROPPED_TABLE

列出已删除的表记录。不返回与条目相关的 DDL 字段。要检索某个条目的 DDL 信息，必须在取装该条目之后立即使用调用程序操作 DB2HISTORY_GET_DDL 调用 db2HistoryGetEntry。

DB2HISTORY_LIST_LOAD

列出装入操作。

DB2HISTORY_LIST_CRT_TABLESPACE

列出表空间创建和删除运操作。

DB2HISTORY_LIST_REN_TABLESPACE
列出表空间重命名操作。

DB2HISTORY_LIST_ALT_TABLESPACE
列出改变表空间操作。不返回与条目相关的 DDL 字段。要检索某个条目的 DDL 信息，必须在取装该条目之后立即使用调用程序操作 DB2HISTORY_GET_DDL 调用 db2HistoryGetEntry。

DB2HISTORY_LIST_REORG
列出 REORGANIZE TABLE 操作。此值当前不受支持。

oHandle
输出。在 API 返回的基础上，此参数包含扫描访问的句柄。随后在 db2HistoryGetEntry 和 db2HistoryCloseScan 中使用它。

REXX API 语法:

```
OPEN [BACKUP] RECOVERY HISTORY FILE FOR database_alias
[OBJECT objname] [TIMESTAMP :timestamp]
USING :value
```

REXX API 参数:

database_alias
要列出其历史文件的数据库的别名。

objname
指定要用于选择记录的对象名称。对象可以是表或表空间。如果是表，必须提供全限定表名。将此参数设置为 NULL 可防止过滤使用 *objname* 的条目。

timestamp
指定要用于选择记录的时间戳记。将选择那些时间戳记等于或大于此值的记录。将此参数设置为 NULL 可防止过滤使用 *timestamp* 的条目。

value 一个复合 REXX 主机变量，对其返回历史文件信息。以下例中，XXX 表示主机变量名。

XXX.0 变量的元素数（总为 2）

XXX.1 未来扫描访问的标识符（句柄）

XXX.2 匹配历史文件条目的数目。

使用注意事项:

时间戳记、对象名称和调用程序操作的组合可用于过滤记录。只返回发送所有指定的过滤器的记录。

对象名称的过滤效果取决于指定的值:

- 因为装入操作的记录是历史文件中表的仅有信息，所以指定一个表将返回装入操作的记录。
- 指定表空间将返回表空间的备份、复原和装入操作的记录。

注: 要返回表的记录，必须将表指定为 *schema.tablename*。指定 *tablename* 将只返回表空间的返回记录。

允许每个进程最多扫描 8 个历史文件。

要列出历史文件中的每个条目，典型的应用程序通常会执行以下步骤:

1. 调用 db2HistoryOpenScan，它将返回 *oNumRows*。
2. 分配 *db2HistData* 结构，它的 *n oTablespace* 字段表示空间，其中 *n* 是任意数。
3. 将 *db2HistData* 结构的 *iDB2NumTablespace* 字段设置为 *n*。
4. 在循环中，执行以下操作:
   - 调用 db2HistoryGetEntry 以从历史文件中取装。
   - 如果 db2HistoryGetEntry 返回 SQLCODE `SQL_RC_OK`，则使用 *db2HistData* 结构的 *sqld* 字段来确定返回的表空间条目的数目。
   - 如果 db2HistoryGetEntry 返回 SQLCODE `SQLUH_SQLUHINFO_VARS_WARNING`，则表示未对 DB2 尝试返回的所有表空间分配足够的空间，应该释放并重新分配 *db2HistData* 结构，为 *oDB2UsedTablespace* 表空间条目提供足够的表空间，然后将 *iDB2NumTablespace* 设置为 *oDB2UsedTablespace*。
   - 如果 db2HistoryGetEntry 返回 SQLCODE `SQLE_RC_NOMORE`，则表示已检索所有历史文件条目。
   - 其它任何 SQLCODE 都表示有问题。
5. 取装所有信息后，调用 db2HistoryCloseScan 以释放通过调用 db2HistoryOpenScan 分配的资源。

提供宏 SQLUHINFOSIZE(*n*)（是在 `sqlutil` 中定义的）是为了帮助确定为 *n* 个 *oTablespace* 字段提供空间的 *db2HistData* 结构需要多少内存。

相关参考:

- 第 228 页的『db2Prune — 修剪历史文件』
- 第 225 页的『db2HistoryUpdate — 更新历史文件』
- 第 217 页的『db2HistoryGetEntry — 获取下一个历史文件条目』
- 第 216 页的『db2HistoryCloseScan — 关闭历史文件扫描』
- 『SQLCA』（*Administrative API Reference*）

相关样本:

- 『dbrecov.sqc -- How to recover a database (C)』
- 『dbrecov.sqC -- How to recover a database (C++)』

db2HistoryUpdate — 更新历史文件

更新历史文件条目中的位置、设备类型或注释。

授权:

以下其中一项:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*

必需的连接:

数据库。要更新非缺省数据库的历史文件中的条目，必须在调用该 API 之前先建立与该数据库的连接。

API 包含文件:

db2ApiDf.h

C API 语法:

```
/* File: db2ApiDf.h */
/* API: db2HistoryUpdate */
/* ... */
SQL_API_RC SQL_API_FN
  db2HistoryUpdate (
    db2Uint32 version,
    void *pDB2HistoryUpdateStruct,
      struct sqlca * pSqlca);

typedef struct
{
  char *piNewLocation,
  char *piNewDeviceType,
  char *piNewComment,
  db2Uint32 iEID
} db2HistoryUpdateStruct;
/* ... */
```

一般 API 语法:

```
/* File: db2ApiDf.h */
/* API: db2GenHistoryUpdate */
/* ... */
SQL_API_RC SQL_API_FN
  db2GenHistoryUpdate (
    db2Uint32 version,
    void *pDB2GenHistoryUpdateStruct,
      struct sqlca * pSqlca);

typedef struct
{
  char *piNewLocation,
  char *piNewDeviceType,
  char *piNewComment,
  db2Uint32 iEID
} db2GenHistoryUpdateStruct;
/* ... */
```

API 参数:

version
输入。指定作为第二个参数 *pDB2HistoryUpdateStruct* 传入的结构的版本和发行版级别。

pDB2HistoryUpdateStruct
输入。指向 *db2HistoryUpdateStruct* 结构的一个指针。

pSqlca
输出。指向 *sqlca* 结构的指针。

piNewLocation
输入。指向字符串的一个指针，该字符串指定备份、复原或装入副本映象的新位置。将此参数设置为 NULL，或指向零，使该值保持不变。

piNewDeviceType
输入。指向字符串的一个指针，该字符串指定存储备份、复原或装入副本映象的新设备类型。将此参数设置为 NULL，或指向零，使该值保持不变。

piNewComment
输入。指向字符串的一个指针，该字符串指定描述条目的新注释。将此参数设置为 NULL，或指向零，使该注释保持不变。

iEID 输入。可用于更新历史文件中特定条目的唯一标识符。

REXX API 语法:

```
UPDATE RECOVERY HISTORY USING :value
```

REXX API 参数:

value 一个复合 REXX 主机变量，包含与历史文件条目的新位置相关的信息。以下例中，XXX 表示主机变量名:

XXX.0 变量的元素数（必须在 1 和 4 之间）

XXX.1 OBJECT_PART（序列号从 001 到 999 的时间戳记）

XXX.2 备份或副本映象的新位置（此参数是可选的）

XXX.3 用于存储备份或副本映象的新设备（此参数是可选的）

XXX.4 新注释（此参数是可选的）。

使用注意事项:

这是一个更新功能，在此更改之前的所有信息都将被替换并不可重新创建。这些更改不会记入日志。

历史文件只用于进行记录。它不会由复原或前滚功能直接使用。在复原操作期间，可指定备份映象的位置，也可将历史文件用于跟踪位置。该信息随后可提供给备份实用程序。同样，如果移动了装入副本映象的位置，则必须提供带有新位置和存储媒体类型的前滚实用程序。

相关参考:

- 第 127 页的『db2Rollforward — 前滚数据库』
- 第 228 页的『db2Prune — 修剪历史文件』
- 第 220 页的『db2HistoryOpenScan — 打开历史文件扫描』
- 第 217 页的『db2HistoryGetEntry — 获取下一个历史文件条目』
- 第 216 页的『db2HistoryCloseScan — 关闭历史文件扫描』
- 『SQLCA』（*Administrative API Reference*）
- 第 67 页的『db2Backup — 备份数据库』

相关样本:

- 『dbrecov.sqc -- How to recover a database (C)』
- 『dbrecov.sqC -- How to recover a database (C++)』

db2Prune — 修剪历史文件

从历史文件中删除条目，或从活动日志路径中删除日志文件。

授权:

以下其中一项:

- *sysadm*
- *sysctrl*
- *sysmaint*
- *dbadm*

必需的连接:

数据库。要从任何非缺省数据库的历史文件中删除条目，必须在调用此 API 之前建立与该数据库的连接。

API 包含文件:

db2ApiDf.h

C API 语法:

```
/* File: db2ApiDf.h */
/* API: db2Prune */
/* ... */
SQL_API_RC SQL_API_FN
  db2Prune (
    db2Uint32 version,
    void *pDB2PruneStruct,
      struct sqlca * pSqlca);

typedef struct
{
  char *piString,
  db2Uint32 iEID,
  db2Uint32 iCallerAction,
  db2Uint32 iOptions
} db2PruneStruct;
/* ... */
```

一般 API 语法:

```
/* File: db2ApiDf.h */
/* API: db2GenPrune */
/* ... */
SQL_API_RC SQL_API_FN
  db2GenPrune (
    db2Uint32 version,
    void *pDB2GenPruneStruct,
      struct sqlca * pSqlca);

typedef struct
{
  db2Uint32 iStringLen;
  char *piString,
  db2Uint32 iEID,
  db2Uint32 iCallerAction,
  db2Uint32 iOptions
} db2GenPruneStruct;
/* ... */
```

API 参数:

version

输入。指定作为第二个参数 *pDB2PruneStruct* 传入的结构的版本和发行版级别。

pDB2PruneStruct

输入。指向 *db2PruneStruct* 结构的一个指针。

pSqlca

输出。指向 *sqlca* 结构的指针。

iStringLen

输入。指定 *piString* 的长度，以字节为单位。

piString

输入。指向字符串的一个指针，它指定时间戳记或日志序列号（LSN）。时间戳记或时间戳记的一部分（最少是 *yyyy* 或 year）用于选择要删除的记录。所有等于或小于该时间戳记的条目都将被删除。必须提供有效的时间戳记；NULL 参数没有缺省行为。

此参数还可用于发送 LSN，因此可修剪不活动的日志。

iEID 输入。指定可用于修剪历史文件中的单个条目的唯一标识符。

iCallerAction

输入。指定要采取的操作的类型。有效值（在 db2ApiDf 中定义）是:

DB2PRUNE_ACTION_HISTORY

除去历史文件条目。

DB2PRUNE_ACTION_LOG
从活动的日志路径中除去日志文件。

iOptions
输入。有效值（在 db2ApiDf 中定义）是:

DB2PRUNE_OPTION_FORCE
强制除去上一个备份。

DB2PRUNE_OPTION_LSNSTRING
指定 *piString* 的值是 LSN，在指定了 DB2PRUNE_ACTION_LOG 的调用程序操作时使用。

REXX API 语法:

```
PRUNE RECOVERY HISTORY BEFORE :timestamp [WITH FORCE OPTION]
```

REXX API 参数:

timestamp
包含了时间戳记的主机变量。将从历史文件中删除时间戳记等于或小于所提供的时间戳记的所有条目。

WITH FORCE OPTION
如果指定该选项，将根据指定的时间戳记来修剪历史文件，即使会从该文件中删除最新复原设置中的某些条目。如果不指定该选项，将保留最新的复原设置，即使时间戳记小于或等于作为输入指定的时间戳记。

使用注意事项:

修剪历史文件并不会删除实际的备份或装入文件。用户必须手工删除这些文件才能释放它们在存储媒体上所消耗的空间。

注意:
如果从媒体上删除了最新的完整数据库备份（不仅是从历史文件中对它进行了修剪），用户必须确保所有表空间，包括目录表空间和用户表空间都已备份。未成功进行备份可能会导致数据库无法恢复，或数据库中用户数据的某些部分丢失。

相关参考:

- 第 225 页的『db2HistoryUpdate — 更新历史文件』
- 第 220 页的『db2HistoryOpenScan — 打开历史文件扫描』
- 第 217 页的『db2HistoryGetEntry — 获取下一个历史文件条目』
- 第 216 页的『db2HistoryCloseScan — 关闭历史文件扫描』
- 『SQLCA』（*Administrative API Reference*）

相关样本:

- 『dbrecov.sqc -- How to recover a database (C)』
- 『dbrecov.sqC -- How to recover a database (C++)』

db2ReadLogNoConn — 读取日志而不进行数据库连接

从 DB2 UDB 数据库日志抽取日志记录，并查询“日志管理器”以获取当前日志状态信息。在使用此 API 之前，使用 db2ReadLogNoConnInit 来分配作为输入参数传送至此 API 的内存。在使用此 API 之后，可以使用 db2ReadLogNoConnTerm 来释放内存。

授权:

以下其中一项:

- *sysadm*
- *dbadm*

必需的连接:

数据库

API 包含文件:

db2ApiDf.h

C API 语法:

```
/* File: db2ApiDf.h */
/* API: db2ReadLogNoConn */
/* ... */
SQL_API_RC SQL_API_FN
db2ReadLogNoConn (
  db2Uint32 versionNumber,
  void *pDB2ReadLogNoConnStruct,
      struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2ReadLogNoConnStruct
{
  db2Uint32                   iCallerAction;
  SQLU_LSN                    *piStartLSN;
  SQLU_LSN                    *piEndLSN;
  char                        *poLogBuffer;
  db2Uint32                   iLogBufferSize;
  char                        *piReadLogMemPtr;
  db2ReadLogNoConnInfoStruct  *poReadLogInfo;
} db2ReadLogNoConnStruct;

typedef SQL_STRUCTURE db2ReadLogNoConnInfoStruct
{
  SQLU_LSN                    firstAvailableLSN;
SQLU_LSN       firstReadLSN ;
  SQLU_LSN                    nextStartLSN;
  db2Uint32                   logRecsWritten;
  db2Uint32                   logBytesWritten;
  db2Uint32                   lastLogFullyRead;
  db2TimeOfLog                currentTimeValue;
} db2ReadLogNoConnInfoStruct;

/* ... */
```

API 参数:

version

输入。指定作为第二个参数 *pDB2ReadLogNoConnStruct* 传送的结构的版本和发行版级别。

pParamStruct

输入。指向 *db2ReadLogNoConnStruct* 结构的指针。

pSqlca

输出。指向 *sqlca* 结构的指针。

iCallerAction

输入。指定要执行的操作。有效的值为:

DB2READLOG_READ

从起始日志序列号至结束日志序列号读取数据库日志，并返回此范围内的日志记录。

DB2READLOG_READ_SINGLE

从起始日志序列号读取单个的日志记录（无论该日志记录是可传播的还是不可传播的）。

DB2READLOG_QUERY

查询数据库日志。查询结果将通过 db2ReadLogNoConnInfoStruct 结构发送回来。

piStartLSN

输入。起始日志序列号指定了读取日志的起始关系字节地址。该值必须是某个实际日志记录的开始。

piEndLSN

输入。结束日志序列号指定了读取日志的结束关系字节地址。此值必须大于 *piStartLsn*，且不一定是某个实际日志记录的结尾。

poLogBuffer

输出。按顺序存储在指定范围内读取的所有可传播日志记录的缓冲区。该缓冲区的大小必须至少能保留一个日志记录。建议此缓冲区应有最少 32 个字节。它的最大大小取决于要求的范围的大小。缓冲区中的每个日志记录都有一个 6 字节日志序列号（LSN）的前缀，表示后跟的日志记录的 LSN。

iLogBufferSize

输入。以字节为单位指定日志缓冲区的大小。

piReadLogMemPtr

输入。在初始化调用中分配了的大小为 iReadLogMemoryLimit 的内存块。此内存包含每次调用时 API 所需的持久数据。此内存块一定不能由调用程序以任何方式重新分配或改变。

poReadLogInfo

输出。指向 *db2ReadLogNoConnInfoStruct* 结构的指针。

firstAvailableLSN

可用日志中的第一个可用 LSN。

firstReadLSN

对此调用读取的第一个 LSN。

nextStartLSN

下一个可读 LSN。

logRecsWritten

写至缓冲区字段 *poLogBuffer* 的日志记录数。

logBytesWritten
写至日志缓冲区字段 *poLogBuffer* 的字节数。

lastLogFullyRead
指示读至最后一个日志文件（即完成）的号码。

使用注意事项:

db2ReadLogNoConn API 需要必须使用 db2ReadLogNoConnInit API 分配的内存块。该内存块必须作为输入参数传送给所有后续 db2ReadLogNoConn API 调用，并且一定不能改变。

请求顺序读取日志时，该 API 需要日志序列号（LSN）范围和已分配内存。该 API 将根据初始化时指定的过滤器选项和 LSN 范围返回日志记录序列。请求查询时，读取日志信息结构将包含要对读取调用使用的有效起始 LSN。用作读取操作的结束 LSN 的值可以是以下其中一个:

- 大于调用程序指定的起始 LSN 的值。
- FFFF FFFF FFFF，由异步日志阅读器解释为可用日志的结尾。

在起始和结束 LSN 范围内读取的可传播日志记录都将返回到日志缓冲区中。日志记录不包含其 LSN，它包含在实际日志记录之前的缓冲区中。可以在 “DB2 UDB 日志记录” 一节中找到 db2ReadLogNoConn 返回的各种 DB2 UDB 日志记录的描述。

在初始读取之后，为了读取下一顺序日志记录，使用 db2ReadLogNoConnInfoStruct 中返回的 nextStartLSN 值。使用这一新起始 LSN 和有效结束 LSN 重新提交调用，然后读取下一记录块。sqlca 代码 SQLU_RLOG_READ_TO_CURRENT 表示日志阅读器已读至可用日志文件的结尾。

不再使用该 API 时，使用 db2ReadLogNoConnTerm 来终止内存。

相关参考:

- 第 235 页的『db2ReadLogNoConnInit — 初始化读取日志而不进行数据库连接』
- 第 237 页的『db2ReadLogNoConnTerm — 终止读取日志而不进行数据库连接』

db2ReadLogNoConnInit — 初始化读取日志而不进行数据库连接

分配 db2ReadLogNoConn 要使用的内存，以便从 DB2 UDB 数据库日志中抽取日志记录，并查询“日志管理器”以获取当前日志状态信息。

必需的连接:

数据库

API 包含文件:

db2ApiDf.h

C API 语法:

```
/* File: db2ApiDf.h */
/* API: db2ReadLogNoConnInit */
/* ... */
SQL_API_RC SQL_API_FN
db2ReadLogNoConnInit (
  db2Uint32 versionNumber,
  void * pDB2ReadLogNoConnInitStruct,
      struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2ReadLogNoConnInitStruct
{
  db2Uint32                   iFilterOption;
  char                        *piLogFilePath;
  char                        *piOverflowLogPath;
  db2Uint32                   iRetrieveLogs;
  char                        *piDatabaseName;
  char                        *piNodeName;
  db2Uint32                   iReadLogMemoryLimit;
  char                        **poReadLogMemPtr;
} db2ReadLogNoConnInitStruct;
/* ... */
```

API 参数:

version

输入。指定作为第二个参数 *pDB2ReadLogNoConnInitStruct* 传送的结构的版本和发行版级别。

pParamStruct

输入。指向 *db2ReadLogNoConnInitStruct* 结构的指针。

pSqlca

输出。指向 *sqlca* 结构的指针。

iFilterOption

输入。指定读取日志记录时要使用的日志记录过滤级别。有效的值为:

DB2READLOG_FILTER_OFF

读取给定 LSN 范围内的所有日志记录。

DB2READLOG_FILTER_ON

仅读取给定 LSN 范围内标记为可传播的日志记录。这是异步读取 API 的传统行为。

piLogFilePath

输入。要读取的日志文件所在的路径。

piOverflowLogPath

输入。要读取的日志文件可能位于的备用路径。

iRetrieveLogs

输入。一个选项，指定是否应该调用用户出口来检索在日志文件路径或溢出日志路径中找不到的日志文件。有效的值为:

DB2READLOG_RETRIEVE_OFF

不应该调用用户出口来检索缺少的日志文件。

DB2READLOG_RETRIEVE_LOGPATH

应该调用用户出口来将缺少的日志文件检索至指定日志文件路径。

DB2READLOG_RETRIEVE_OVERFLOW

应该调用用户出口来将缺少的日志文件检索至指定溢出日志路径。

piDatabaseName

输入。拥有正在读取的恢复日志的数据库的名称。如果指定了上面的检索选项，则此参数是必需的。

piNodeName

输入。拥有正在读取的恢复日志的节点的名称。如果指定了上面的检索选项，则此参数是必需的。

iReadLogMemoryLimit

输入。该 API 内部可能分配的最大字节数。

poReadLogMemPtr

输出。API 分配的大小为 *iReadLogMemoryLimit* 的内存块。此内存包含每次调用时 API 所需的持久数据。此内存块一定不能由调用程序以任何方式重新分配或改变。

使用注意事项:

一定不能改变 db2ReadLogNoConnInit 初始化的内存。

不再使用 db2ReadLogNoConn 时，调用 db2ReadLogNoConnTerm 来释放 db2ReadLogNoConnInit 初始化的内存。

相关参考:

- 第 231 页的『db2ReadLogNoConn — 读取日志而不进行数据库连接』
- 第 237 页的『db2ReadLogNoConnTerm — 终止读取日志而不进行数据库连接』

db2ReadLogNoConnTerm — 终止读取日志而不进行数据库连接

释放 db2ReadLogNoConn 使用内存，它是由 db2ReadLogNoConnInit 最初进行初始化的。

授权:

以下其中一项:

- *sysadm*
- *dbadm*

必需的连接:

数据库

API 包含文件:

db2ApiDf.h

C API 语法:

```
/* File: db2ApiDf.h */
/* API: db2ReadLogNoConnTerm */
/* ... */
SQL_API_RC SQL_API_FN
db2ReadLogNoConnTerm (
  db2Uint32 versionNumber,
  void * pDB2ReadLogNoConnTermStruct,
      struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2ReadLogNoConnTermStruct
{
  char                          **poReadLogMemPtr;
} db2ReadLogNoConnTermStruct;
/* ... */
```

db2ReadLogNoConnTerm — 终止读取日志而不进行数据库连接

API 参数:

version

输入。指定作为第二个参数 *pDB2ReadLogNoConnTermStruct* 传送的结构版本和发行版级别。

pParamStruct

输入。指向 *db2ReadLogNoConnTermStruct* 结构的指针。

pSqlca

输出。指向 *sqlca* 结构的指针。

poReadLogMemPtr

输出。指向在初始化调用中分配的内存块的指针。将释放此指针，并设置为 NULL。

相关参考:

- 第 231 页的『db2ReadLogNoConn — 读取日志而不进行数据库连接』
- 第 235 页的『db2ReadLogNoConnInit — 初始化读取日志而不进行数据库连接』

---

db2ReadLog — 异步读取日志

从 DB2 UDB 数据库日志抽取日志记录，并查询“日志管理器”以获取当前日志状态信息。只能对可恢复的数据库使用此 API。如果在配置数据库时，*logretain* 设置为 RECOVERY 或 *userexit* 设置为 ON，该数据库就是可恢复的。

授权:

以下其中一项:

- *sysadm*
- *dbadm*

必需的连接:

数据库

API 包含文件:

db2ApiDf.h

C API 语法:

```
/* File: db2ApiDf.h */
/* API: db2ReadLog */
/* ... */
SQL_API_RC SQL_API_FN
  db2ReadLog (
  db2Uint32 versionNumber,
    void *pDB2ReadLogStruct,
      struct sqlca * pSqlca);

typedef SQL_STRUCTURE db2ReadLogStruct
{
  db2Uint32                   iCallerAction;
  SQLU_LSN                    *piStartLSN;
  SQLU_LSN                    *piEndLSN;
  char                        *poLogBuffer;
  db2Uint32                   iLogBufferSize;
  db2Uint32                   iFilterOption;
  db2ReadLogInfoStruct        *poReadLogInfo;

typedef SQL_STRUCTURE db2ReadLogInfoStruct
{
SQLU_LSN       initialLSN ;
SQLU_LSN       firstReadLSN ;
  SQLU_LSN                    nextStartLSN;
  db2Uint32                   logRecsWritten;
  db2Uint32                   logBytesWritten;
  SQLU_LSN                    firstReusedLSN;
  db2Uint32                   timeOfLSNReuse;
  db2TimeOfLog                currentTimeValue;
} db2ReadLogInfoStruct;

typedef SQL_STRUCTURE db2TimeOfLog
{
  db2Uint32                   seconds;
  db2Uint32                   accuracy;
} db2TimeOfLog;
/* ... */
```

API 参数:

versionNumber

输入。指定作为第二个参数 *pDB2ReadLogStruct* 传送的结构的版本和发行版级别。

pDB2ReadLogStruct

输入。指向 *db2ReadLogStruct* 的指针。

pSqlca

输出。指向 *sqlca* 结构的指针。

iCallerAction

输入。指定要执行的操作。

DB2READLOG_READ
从起始日志序列号至结束日志序列号读取数据库日志，并返回此范围内的日志记录。

DB2READLOG_READ_SINGLE
从起始日志序列号读取单个的日志记录（无论该日志记录是可传播的还是不可传播的）。

DB2READLOG_QUERY
查询数据库日志。查询结果将通过 *db2ReadLogInfoStruct* 结构发送回来。

piStartLsn
输入。起始日志序列号指定了读取日志的起始关系字节地址。该值必须是某个实际日志记录的开始。

piEndLsn
输入。结束日志序列号指定了读取日志的结束关系字节地址。此值必须大于 *startLsn*，且不必是某个实际日志记录的末尾。

poLogBuffer
输出。按顺序存储在指定范围内读取的所有可传播日志记录的缓冲区。该缓冲区的大小必须至少能保留一个日志记录。建议此缓冲区应有最少 32 个字节。它的最大大小取决于要求的范围的大小。缓冲区中的每个日志记录都有一个 6 字节日志序列号（LSN）的前缀，表示后跟的日志记录的 LSN。

iLogBufferSize
输入。以字节为单位指定日志缓冲区的大小。

iFilterOption
输入。指定读取日志记录时要使用的日志记录过滤级别。有效的值为:

DB2READLOG_FILTER_OFF
读取给定 LSN 范围内的所有日志记录。

DB2READLOG_FILTER_ON
仅读取给定 LSN 范围内标记为可传播的日志记录。这是异步日志读取 API 的传统行为。

poReadLogInfo
输出。与调用和数据库日志相关的结构详细信息。

使用注意事项:

如果请求的操作是读取日志，调用程序会提供日志序列号范围以及用于保留日志记录的缓冲区。此 API 按顺序读取日志（由请求的 LSN 范围确定），并返回与具有 DATA CAPTURE 选项 CHANGES 的表相关联的日志记录，以及带有当前活动日志信息的 db2ReadLogInfoStruct 结构。如果请求的操作是查询，则该 API 会返回带有当前活动日志信息的 db2ReadLogInfoStruct 结构。

要使用“异步日志阅读器”，首先应查询数据库日志是否有有效的起始 LSN。在查询调用之后，读取日志信息结构（db2ReadLogInfoStruct）将包含要用于读取调用的有效起始 LSN（在 initialLSN 成员中）。用作读取操作的结束 LSN 的值可以是以下其中一个:

- 大于 initialLSN 的值
- FFFF FFFF FFFF，由异步日志阅读器解释为当前日志的末尾。

在起始和结束 LSN 范围内读取的可传播日志记录都将返回到日志缓冲区中。日志记录不包含它的 LSN，它包含在缓冲区中实际的日志记录前。db2ReadLog 返回的各种 DB2 日志记录的描述都可以在“DB2 UDB 日志记录”一节中找到。

要读取初始读取后的下一顺序日志记录，使用 db2ReadLogStruct 结构中返回的 nextStartLSN 字段。重新提交该调用，使用新的起始 LSN 和有效的结束 LSN。然后会读取下一个记录块。*sqlca* 代码 SQLU_RLOG_READ_TO_CURRENT 表示日志阅读器已读到了当前活动日志的结尾。

相关参考:

- 『SQLCA』（*Administrative API Reference*）

相关样本:

- 『dbrecov.sqc -- How to recover a database (C)』
- 『dbrecov.sqC -- How to recover a database (C++)』

db2HistData

此结构用来在调用 db2HistoryGetEntry 后返回信息。

表 2. db2HistData 结构中的字段

字段名称	数据类型	描述
ioHistDataID	char(8)	一个 8 字节的结构标识符和存储器转储的“眼球捕获器”。唯一有效的值是 "SQLUHINF"。此字符串不存在符号定义。

表 2. db2HistData 结构中的字段 (续)

字段名称	数据类型	描述
oObjectPart	db2Char	首 14 个字符是格式为 *yyyymmddhhnnss* 的时间戳记，它表示操作开始的时间。接着的 3 个字符是序列号。当备份映象保存在多个文件或多个磁带上时，每个备份操作都会在此文件中产生多个条目。序列号允许指定多个位置。复原和装入操作在此文件中只有一个条目，对应于相应备份的序列号 '001'。时间戳记与序列号组合在一起，必须是唯一的。
oEndTime	db2Char	格式为 *yyyymmddhhnnss* 的时间戳记，它表示操作完成的时间。
oFirstLog	db2Char	最早的日志文件标识（范围从 S0000000 到 S9999999）： • 是对联机备份应用前滚恢复所需的 • 是对脱机备份应用前滚恢复所需的 • 复原当装入操作开始时处于当前状态的完整的数据库级或表空间级备份后应用
oLastLog	db2Char	最新的日志文件标识（范围从 S0000000 到 S9999999）： • 是对联机备份应用前滚恢复所需的 • 是对脱机备份将前滚恢复应用到当前时间点所需的 • 复原当装入操作结束时处于当前状态的完整数据库级或表空间级备份时应用（如果未应用前滚恢复，将与 *oFirstLog* 相同）。
oID	db2Char	唯一的备份或表标识符。
oTableQualifier	db2Char	表限定符。
oTableName	db2Char	表名。
oLocation	db2Char	对于备份和装入副本，此字段表示保存数据的位置。对于需要本文件中的多个条目的操作，由 *oObjectPart* 定义的序列号标识了在指定的位置中找到了哪部分的备份。对于复原和装入操作，位置总是标识已保存的复原或装数据的第一部分（对应于多部分备份的序列号 '001'。*oLocation* 中的数据取决于 *oDeviceType*，有不同的解释: • 对于磁盘或软盘（D 或 K），它是全限定文件名 • 对于磁带（T），它是卷标 • 对于 TSM（A），它是服务器名称 • 对于用户出口或其它（U 或 O），它是自由格式文本。

表 2. db2HistData 结构中的字段 (续)

字段名称	数据类型	描述
oComment	db2Char	自由格式文本注释。
oCommandText	db2Char	命令文本或 DDL。
oLastLSN	SQLU_LSN	上一日志序列号。
oEID	Structure	唯一条目标识符。
poEventSQLCA	Structure	已记录的事件的结果 *sqlca*。
poTablespace	db2Char	表空间名称的列表。
ioNumTablespaces	db2Uint32	*poTablespace* 列表中的条目数。每个表空间备份都包含一个或多个表空间。每个表空间复原操作都替换一个或多个表空间。如果此字段不为零（表示表空间级的备份和复原），此文件中的下一行包含了备份或复原的表空间名称，由 18 个字符的字符串表示。一个表空间名称出现在一行上。
oOperation	char	参见表 3。
oObject	char	操作的粒度: D 表示完整的数据库、P 表示表空间而 T 表示表。
oOptype	char	参见第 244 页的表 4。
oStatus	char	条目状态: A 表示操作、D 表示已删除（将来使用）、E 表示已到期、I 表示不活动、N 表示尚未落实、Y 表示已落实或活动，a 表示活动备份，但某些 Datalink Server 尚未完成该备份，而 i 表示不活动备份，但某些 Datalink Server 尚未完成该备份）。
oDeviceType	char	设备类型。此字段确定如何解释 *oLocation* 字段: A 表示 TSM、C 表示客户机、D 表示磁盘、K 表示软盘、L 表示本地、O 表示其它（其它供应商设备支持）、P 表示管道、Q 表示游标、S 表示服务器、T 表示磁带，而 U 表示用户出口。

表 3. db2HistData 结构中的有效 oOperation 值

值	描述	C 定义	COBOL/FORTRAN 定义
A	添加表空间	DB2HISTORY_OP_ADD_TABLESPACE	DB2HIST_OP_ADD_TABLESPACE
B	备份	DB2HISTORY_OP_BACKUP	DB2HIST_OP_BACKUP
C	装入副本	DB2HISTORY_OP_LOAD_COPY	DB2HIST_OP_LOAD_COPY
D	删除的表	DB2HISTORY_OP_DROPPED_TABLE	DB2HIST_OP_DROPPED_TABLE
F	前滚	DB2HISTORY_OP_ROLLFWD	DB2HIST_OP_ROLLFWD
G	识别表	DB2HISTORY_OP_REORG	DB2HIST_OP_REORG

表 3. db2HistData 结构中的有效 oOperation 值 (续)

值	描述	C 定义	COBOL/FORTRAN 定义
L	装入	DB2HISTORY_OP_LOAD	DB2HIST_OP_LOAD
N	重命名表空间	DB2HISTORY_OP_REN_TABLESPACE	DB2HIST_OP_REN_TABLESPACE
O	删除表空间	DB2HISTORY_OP_DROP_TABLESPACE	DB2HIST_OP_DROP_TABLESPACE
Q	停顿	DB2HISTORY_OP_QUIESCE	DB2HIST_OP_QUIESCE
R	复原	DB2HISTORY_OP_RESTORE	DB2HIST_OP_RESTORE
T	改变表空间	DB2HISTORY_OP_ALT_TABLESPACE	DB2HIST_OP_ALT_TBS
U	卸装	DB2HISTORY_OP_UNLOAD	DB2HIST_OP_UNLOAD

表 4. db2HistData 结构中的有效 oOptype 值

oOperation	oOptype	描述	C/COBOL/FORTRAN 定义
B	F	脱机	DB2HISTORY_OPTYPE_OFFLINE
	N	联机	DB2HISTORY_OPTYPE_ONLINE
	I	增量脱机	DB2HISTORY_OPTYPE_INCR_OFFLINE
	O	增量联机	DB2HISTORY_OPTYPE_INCR_ONLINE
	D	delta 脱机	DB2HISTORY_OPTYPE_DELTA_OFFLINE
	E	delta 联机	DB2HISTORY_OPTYPE_DELTA_ONLINE
F	E	日志结束	DB2HISTORY_OPTYPE_EOL
	P	时间点	DB2HISTORY_OPTYPE_PIT
G	F	脱机	DB2HISTORY_OPTYPE_OFFLINE
	N	联机	DB2HISTORY_OPTYPE_ONLINE
L	I	插入	DB2HISTORY_OPTYPE_INSERT
	R	替换	DB2HISTORY_OPTYPE_REPLACE
Q	S	停顿共享	DB2HISTORY_OPTYPE_SHARE
	U	停顿更新	DB2HISTORY_OPTYPE_UPDATE
	X	停顿独占	DB2HISTORY_OPTYPE_EXCL
	Z	停顿复位	DB2HISTORY_OPTYPE_RESET

表 4. db2HistData 结构中的有效 oOptype 值 (续)

oOperation	oOptype	描述	C/COBOL/FORTRAN 定义
R	F	脱机	DB2HISTORY_OPTYPE_OFFLINE
	N	联机	DB2HISTORY_OPTYPE_ONLINE
	I	增量脱机	DB2HISTORY_OPTYPE_INCR_OFFLINE
	O	增量联机	DB2HISTORY_OPTYPE_INCR_ONLINE
T	C	添加容器	DB2HISTORY_OPTYPE_ADD_CONT
	R	重新平衡	DB2HISTORY_OPTYPE_REB

表 5. db2Char 结构中的字段

字段名称	数据类型	描述
pioData	char	指向字符数据缓冲区的指针。如果为 NULL，将不返回任何数据。
iLength	db2Uint32	输入。*pioData* 缓冲区的大小。
oLength	db2Uint32	输出。*pioData* 缓冲区中有效的数据字符数。

表 6. db2HistoryEID 结构中的字段

字段名称	数据类型	描述
ioNode	SQL_PDB_NODE_TYPE	节点号。
ioHID	db2Uint32	本地历史文件入口标识。

语言语法:

C 结构

```
/* File: db2ApiDf.h */
/* ... */
typedef SQL_STRUCTURE db2HistoryData
{
  char ioHistDataID[8];
  db2Char oObjectPart;
  db2Char oEndTime;
  db2Char oFirstLog;
  db2Char oLastLog;
  db2Char oID;
  db2Char oTableQualifier;
  db2Char oTableName;
  db2Char oLocation;
  db2Char oComment;
  db2Char oCommandText;
  SQLU_LSN oLastLSN;
  db2HistoryEID oEID;
  struct sqlca * poEventSQLCA;
  db2Char * poTablespace;
  db2Uint32 ioNumTablespaces;
  char oOperation;
  char oObject;
  char oOptype;
  char oStatus;
  char oDeviceType
} db2HistoryData;

typedef SQL_STRUCTURE db2Char
{
  char * pioData;
  db2Uint32 ioLength
} db2Char;

typedef SQL_STRUCTURE db2HistoryEID
{
  SQL_PDB_NODE_TYPE ioNode;
  db2Uint32 ioHID
} db2HistoryEID;
/* ... */
```

相关参考:

- 第 217 页的『db2HistoryGetEntry — 获取下一个历史文件条目』
- 『SQLCA』（*Administrative API Reference*）

SQLU-LSN

此并集由 db2ReadLog API 使用，包含日志序列号的定义。日志序列号（LSN）代表数据库日志中一个相对的字节地址。所有日志记录都由此号码来标识。它代表日志记录从数据库日志开始处的字节偏移。

表 7. SQLU-LSN 并集中的字段

字段名称	数据类型	描述
lsnChar	Array of UNSIGNED CHAR	指定由 6 个成员组成的字符数组日志序列号。
lsnWord	Array of UNSIGNED SHORT	指定由 3 个成员组成的短数组日志序列号。

语言语法:

C 结构

```
typedef union SQLU_LSN
{
unsigned char  lsnChar  [6] ;
unsigned short lsnWord  [3] ;
} SQLU_LSN;
```

相关参考:

- 第 238 页的『db2ReadLog — 异步读取日志』

附录 E. 恢复样本程序

带有嵌入式 SQL 的样本程序（dbrecov.sqc）

以下样本程序显示了如何使用 DB2 备份与复原 API 来:

- 备份数据库
- 复原数据库
- 前滚恢复数据库

注: dbrecov 样本文件可在 sqllib/samples/c and sqllib/samples/cpp 目录中找到。

```
/****************************************************************************
** Licensed Materials - Property of IBM
** Governed under the terms of the IBM Public License
**
** (C) COPYRIGHT International Business Machines Corp. 2002
** All Rights Reserved.
**
** US Government Users Restricted Rights - Use, duplication or
** disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
****************************************************************************
**
** SOURCE FILE NAME: dbrecov.sqc
**
** SAMPLE: How to recover a database
**
** DB2 API USED:
**        db2HistoryCloseScan -- Close Recovery History File Scan
**        db2HistoryGetEntry -- Get Next Recovery History File Entry
**        db2HistoryOpenScan -- Open Recovery History File Scan
**        db2HistoryUpdate -- Update Recovery History File
**        db2Prune -- Prune Recovery History File
**         db2CfgGet -- Get Configuration
**         db2CfgSet -- Set Configuration
**        sqlbmtsq -- Tablespace Query
**        sqlbstsc -- Set Tablespace Containers
**        sqlbtcq -- Tablespace Container Query
**         sqlecrea -- Create Database
**        sqledrpd -- Drop Database
**        sqlefmem -- Free Memory
**         db2Backup -- Backup Database
**         db2Restore -- Restore Database
**         db2ReadLog -- Asynchronous Read Log
**         db2ReadLogNoConn -- No Db Connection Read Log
**         db2Rollforward -- Rollforward Database
**
** SQL STATEMENTS USED:
**         ALTER TABLE
```

```
**         COMMIT
**         DELETE
**         INSERT
**         ROLLBACK
**
** OUTPUT FILE: dbrecov.out (available in the online documentation)
**
** For detailed information about database backup and recovery, see the
** "Data Recovery and High Availability Guide and Reference". This manual
** will help you to determine which database and table space recovery methods
** are best suited to your business environment.
**
** For more information about the sample programs, see the README file.
**

** "Programming in C and C++" section of the "Application Development Guide".
**
** For more information about building C applications, see the
** section for your compiler in the "Building Applications" chapter
** for your platform in the "Application Development Guide".
**
** For more information about SQL, see the "SQL Reference".
**
** For more information on DB2 APIs, see the Administrative API Reference.
**
** For the latest information on programming, compiling, and running DB2
** applications, refer to the DB2 application development website at
**     http://www.software.ibm.com/data/db2/udb/ad
******************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sqlenv.h>
#include <sqlutil.h>
#include <db2ApiDf.h>
#include "utilemb.h"

int DbRecoveryHistoryFilePrune(char *, char *, char *);
int DbBackupAndRestore(char *, char *, char *, char *, char *);
int DbBackupAndRedirectedRestore(char *, char *, char *, char *, char *);
int DbBackupRestoreAndRollforward(char *, char *, char *, char *, char *);
int DbLogRecordsForCurrentConnectionRead(char *, char *, char *, char *);
int DbRecoveryHistoryFileRead(char *);
int DbFirstRecoveryHistoryFileEntryUpdate(char *, char *, char *);
int DbReadLogRecordsNoConn(char *);

/* support function called by main() */
int ServerWorkingPathGet(char *, char *);

/* support function called by DbBackupAndRedirectedRestore() */
int InaccessableContainersRedefine(char *);

/* support function called by DbBackupAndRedirectedRestore() and
   DbBackupRestoreAndRollforward() */
int DbDrop(char *);
```

```
/* support function called by DbLogRecordsForCurrentConnectionRead() */
int LogBufferDisplay(char *, sqluint32);
int LogRecordDisplay(char *, sqluint32, sqluint16, sqluint16);
int SimpleLogRecordDisplay(sqluint16, sqluint16, char *, sqluint32);
int ComplexLogRecordDisplay(sqluint16, sqluint16, char *, sqluint32,
                            sqluint8, char *, sqluint32);
int LogSubRecordDisplay(char *, sqluint16);
int UserDataDisplay(char *, sqluint16);

/* support functions called by DbRecoveryHistoryFileRead() and
   DbFirstRecoveryHistoryFileEntryUpdate() */
int HistoryEntryDataFieldsAlloc(struct db2HistoryData *);
int HistoryEntryDisplay(struct db2HistoryData);
int HistoryEntryDataFieldsFree(struct db2HistoryData *);

/* DbCreate will create a new database on the server with the server's
     code page.
   Use this function only if you want to restore a remote database.
   This support function is being called by DbBackupAndRedirectedRestore()
   and DbBackupRestoreAndRollforward().  */
int DbCreate(char *, char *);

int main(int argc, char *argv[])
{
  int rc = 0;
  char nodeName[SQL_INSTNAME_SZ + 1];
  char serverWorkingPath[SQL_PATH_SZ + 1];
  char restoredDbAlias[SQL_ALIAS_SZ + 1];
  char redirectedRestoredDbAlias[SQL_ALIAS_SZ + 1];
  char rolledForwardDbAlias[SQL_ALIAS_SZ + 1];
  sqluint16 savedLogRetainValue;
  char dbAlias[SQL_ALIAS_SZ + 1];
  char user[USERID_SZ + 1];
  char pswd[PSWD_SZ + 1];

  /* check the command line arguments */
  rc = CmdLineArgsCheck3(argc, argv, dbAlias, nodeName, user, pswd);
  if (rc != 0)
  {
    return rc;
  }

  printf("\nTHIS SAMPLE SHOWS HOW TO RECOVER A DATABASE.\n");

  strcpy(restoredDbAlias, dbAlias);
  strcpy(redirectedRestoredDbAlias, "RRDB");
  strcpy(rolledForwardDbAlias, "RFDB");

  /* attach to a local or remote instance */
  rc = InstanceAttach(nodeName, user, pswd);
  if (rc != 0)
  {
    return rc;
  }
```

```
printf("\nUSE THE DB2 API:\n");
printf("  db2CfgGet -- Get Configuration\n");
printf("TO GET THE DATABASE CONFIGURATION AND DETERMINE\n");
printf("THE SERVER WORKING PATH.\n");

/* get the server working path */
rc = ServerWorkingPathGet(dbAlias, serverWorkingPath);
if (rc != 0)
{
  return rc;
}

printf("\nNOTE: Backup images will be created on the server\n");
printf("      in the directory %s,\n", serverWorkingPath);
printf("      and will not be deleted by the program.\n");

/* call the sample functions */
rc = DbRecoveryHistoryFilePrune(dbAlias, user, pswd);

rc = DbBackupAndRestore(dbAlias,
             restoredDbAlias,
          user,
          pswd,
                                        serverWorkingPath);

rc = DbBackupAndRedirectedRestore(dbAlias,
                                  redirectedRestoredDbAlias,
          user,
          pswd,
                                        serverWorkingPath);

rc = DbBackupRestoreAndRollforward(dbAlias,
             rolledForwardDbAlias,
          user,
          pswd,
                                        serverWorkingPath);

rc = DbLogRecordsForCurrentConnectionRead(dbAlias,
          user,
          pswd,
                                        serverWorkingPath);

rc = DbRecoveryHistoryFileRead(dbAlias);

rc = DbFirstRecoveryHistoryFileEntryUpdate(dbAlias, user, pswd);

rc = DbRecoveryHistoryFilePrune(dbAlias, user, pswd);

/* detach from the local or remote instance */
rc = InstanceDetach(nodeName);
if (rc != 0)
{
  return rc;
}
```

```
  rc = DbReadLogRecordsNoConn(dbAlias);

  return 0;
} /* end main */

int ServerWorkingPathGet(char dbAlias[], char serverWorkingPath[])
{
  int rc = 0;
  struct sqlca sqlca;
  char serverLogPath[SQL_PATH_SZ + 1];
  db2CfgParam cfgParameters[1];
  db2Cfg cfgStruct;
  int len;

  /* initialize cfgParameters */
  /* SQLF_DBTN_LOGPATH is a token of the non-updatable database configuration
     parameter 'logpath'; it is used to get the server log path */
  cfgParameters[0].flags = 0;
  cfgParameters[0].token = SQLF_DBTN_LOGPATH;
  cfgParameters[0].ptrvalue =
    (char *)malloc((SQL_PATH_SZ + 1) * sizeof(char));

  /* initialize cfgStruct */
  cfgStruct.numItems = 1;
  cfgStruct.paramArray = cfgParameters;
  cfgStruct.flags = db2CfgDatabase;
  cfgStruct.dbname = dbAlias;

  /* get database configuration */
  /* the API db2CfgGet returns the values of individual entries in a
     database configuration file */
  db2CfgGet(db2Version810, (void *)&cfgStruct, &sqlca);
  DB2_API_CHECK("server log path -- get");

  strcpy(serverLogPath, cfgParameters[0].ptrvalue);
  free(cfgParameters[0].ptrvalue);

  /* choose the server working path; if, for example, serverLogPath =
     "C:\DB2\NODE0001\....", we'll keep "C:\DB2" for the serverWorkingPath
     variable; backup images created in this sample will be placed under
     the 'serverWorkingPath' directory */
  len = (int)(strstr(serverLogPath, "NODE") - serverLogPath - 1);
  memcpy(serverWorkingPath, serverLogPath, len);
  serverWorkingPath[len] = '\0';

  return 0;
} /* ServerWorkingPathGet */

int DbCreate(char existingDbAlias[], char newDbAlias[])
{
  struct sqlca sqlca;
  char dbName[SQL_DBNAME_SZ + 1];
  char dbLocalAlias[SQL_ALIAS_SZ + 1];
  char dbPath[SQL_PATH_SZ + 1];
```

```
struct sqledbdesc dbDescriptor;
struct sqledbcountryinfo countryInfo;
db2CfgParam cfgParameters[2];
db2Cfg cfgStruct;

printf("\n  Create '%s' empty database with the same code set as '%s'
       database.\n", newDbAlias, existingDbAlias);

/* initialize cfgParameters */
cfgParameters[0].flags = 0;
cfgParameters[0].token = SQLF_DBTN_TERRITORY;
cfgParameters[0].ptrvalue = (char *)malloc(10 * sizeof(char));
memset(cfgParameters[0].ptrvalue, '\0', 10);
cfgParameters[1].flags = 0;
cfgParameters[1].token = SQLF_DBTN_CODESET;
cfgParameters[1].ptrvalue = (char *)malloc(20 * sizeof(char));
memset(cfgParameters[1].ptrvalue, '\0', 20);

/* initialize cfgStruct */
cfgStruct.numItems = 2;
cfgStruct.paramArray = cfgParameters;
cfgStruct.flags = db2CfgDatabase;
cfgStruct.dbname = existingDbAlias;

/* get database configuration */
db2CfgGet(db2Version810, (void *)&cfgStruct, &sqlca);
DB2_API_CHECK("server log path -- get");

/* create a new database */
strcpy(dbName, newDbAlias);
strcpy(dbLocalAlias, newDbAlias);
strcpy(dbPath, "");

strcpy(dbDescriptor.sqldbdid, SQLE_DBDESC_2);
dbDescriptor.sqldbccp = 0;
dbDescriptor.sqldbcss = SQL_CS_NONE;

strcpy(dbDescriptor.sqldbcmt, "");
dbDescriptor.sqldbsgp = 0;
dbDescriptor.sqldbnsg = 10;
dbDescriptor.sqltsext = -1;
dbDescriptor.sqlcatts = NULL;
dbDescriptor.sqlusrts = NULL;
dbDescriptor.sqltmpts = NULL;

strcpy(countryInfo.sqldbcodeset, (char *)cfgParameters[0].ptrvalue);
strcpy(countryInfo.sqldblocale, (char *)cfgParameters[1].ptrvalue);

/* create database */
sqlecrea(dbName,
         dbLocalAlias,
         dbPath,
         &dbDescriptor,
         &countryInfo,
         '\0',
```

```
                              NULL,
            &sqlca);
  DB2_API_CHECK("Database -- Create");

  /* free the allocated memory */
  free(cfgParameters[0].ptrvalue);
  free(cfgParameters[1].ptrvalue);

  return 0;
} /* DbCreate */

int DbDrop(char dbAlias[])
{
  struct sqlca sqlca;

  printf("\n  Drop the '%s' database.\n", dbAlias);

  /* drop and uncatalog the database */
  sqledrpd(dbAlias, &sqlca);
  DB2_API_CHECK("Database -- Drop");

  return 0;
} /* DbDrop */

int DbBackupAndRestore(char dbAlias[],
                       char restoredDbAlias[],
                       char user[],
                       char pswd[],
                       char serverWorkingPath[])
{
  int rc = 0;
  struct sqlca sqlca;
  db2CfgParam cfgParameters[1];
  db2Cfg cfgStruct;
  unsigned short logretain;
  char restoreTimestamp[SQLU_TIME_STAMP_LEN + 1];

  db2BackupStruct  backupStruct;
  db2TablespaceStruct   tablespaceStruct;
  db2MediaListStruct mediaListStruct;
  db2Uint32  backupImageSize;
  db2RestoreStruct restoreStruct;
  db2TablespaceStruct   rtablespaceStruct;
  db2MediaListStruct rmediaListStruct;

  printf("\n************************************\n");
  printf("*** BACK UP AND RESTORE A DATABASE ***\n");
  printf("************************************\n");
  printf("\nUSE THE DB2 APIs:\n");
  printf("  db2CfgSet -- Set Configuration\n");
  printf("  db2Backup -- Backup Database\n");
  printf("  db2Restore -- Restore Database\n");
  printf("TO BACK UP AND RESTORE A DATABASE.\n");

  printf("\n  Update \'%s\' database configuration:\n", dbAlias);
```

```
printf("    - Disable the database configuration parameter LOGRETAIN\n");
printf("        i.e., set LOGRETAIN = OFF/NO\n");

/* initialize cfgParameters */
/* SQLF_DBTN_LOG_RETAIN is a token of the updatable database configuration
   parameter 'logretain'; it is used to update the database configuration
   file */
cfgParameters[0].flags = 0;
cfgParameters[0].token = SQLF_DBTN_LOG_RETAIN;
cfgParameters[0].ptrvalue = (char *)&logretain;

/* disable the database configuration parameter 'logretain' */
logretain = SQLF_LOGRETAIN_DISABLE;

/* initialize cfgStruct */
cfgStruct.numItems = 1;
cfgStruct.paramArray = cfgParameters;
cfgStruct.flags = db2CfgDatabase | db2CfgDelayed;
cfgStruct.dbname = dbAlias;

/* set database configuration */
db2CfgSet(db2Version810, (void *)&cfgStruct, &sqlca);
DB2_API_CHECK("Db Log Retain -- Disable");

/*******************************/
/*    BACK UP THE DATABASE     */
/*******************************/
printf("\n  Backing up the '%s' database...\n", dbAlias);

tablespaceStruct.tablespaces = NULL;
tablespaceStruct.numTablespaces = 0;

mediaListStruct.locations = &serverWorkingPath;
mediaListStruct.numLocations = 1;
mediaListStruct.locationType = SQLU_LOCAL_MEDIA;

backupStruct.piDBAlias = dbAlias;
backupStruct.piTablespaceList = &tablespaceStruct;
backupStruct.piMediaList = &mediaListStruct;
backupStruct.piUsername = user;
backupStruct.piPassword = pswd;
backupStruct.piVendorOptions = NULL;
backupStruct.iVendorOptionsSize = 0;
backupStruct.iCallerAction = DB2BACKUP_BACKUP;
backupStruct.iBufferSize = 16; /*  16 x 4KB */
backupStruct.iNumBuffers = 1;
backupStruct.iParallelism =1;
backupStruct.iOptions = DB2BACKUP_OFFLINE | DB2BACKUP_DB;

/* The API db2Backup creates a backup copy of a database.
   This API automatically establishes a connection to the specified database.
   (This API can also be used to create a backup copy of a table space). */
db2Backup (db2Version810, &backupStruct, &sqlca);

  DB2_API_CHECK("Database -- Backup");
```

```
while (sqlca.sqlcode != 0)
{
  /* continue the backup operation */

  /* depending on the sqlca.sqlcode value, user action may be */
  /* required, such as mounting a new tape */

  printf("\n  Continuing the backup operation...\n");

  backupStruct.iCallerAction = DB2BACKUP_CONTINUE;

  db2Backup (db2Version810, &backupStruct, &sqlca);

  DB2_API_CHECK("Database -- Backup");
}

printf("  Backup finished.\n");
printf("    - backup image size      : %d MB\n", backupStruct.oBackupSize);
printf("    - backup image path      : %s\n", mediaListStruct.locations[0]);

printf("    - backup image time stamp: %s\n", backupStruct.oTimestamp);

/*****************************/
/*    RESTORE THE DATABASE    */
/*****************************/

strcpy(restoreTimestamp, backupStruct.oTimestamp);


printf("\n  Restoring a database ...\n");
printf("    - source image alias     : %s\n", dbAlias);
printf("    - source image time stamp: %s\n", restoreTimestamp);
printf("    - target database        : %s\n", restoredDbAlias);

rtablespaceStruct.tablespaces = NULL;
rtablespaceStruct.numTablespaces = 0;

rmediaListStruct.locations = &serverWorkingPath;
rmediaListStruct.numLocations = 1;
rmediaListStruct.locationType = SQLU_LOCAL_MEDIA;

restoreStruct.piSourceDBAlias = dbAlias;
restoreStruct.piTargetDBAlias = restoredDbAlias;

restoreStruct.piTimestamp = restoreTimestamp;
restoreStruct.piTargetDBPath = NULL;
restoreStruct.piReportFile = NULL;
restoreStruct.piTablespaceList = &rtablespaceStruct;
restoreStruct.piMediaList = &rmediaListStruct;
restoreStruct.piUsername = user;
restoreStruct.piPassword = pswd;
restoreStruct.piNewLogPath = NULL;
restoreStruct.piVendorOptions = NULL;
restoreStruct.iVendorOptionsSize = 0;
```

```
  restoreStruct.iParallelism = 1;
  restoreStruct.iBufferSize = 1024; /*  1024 x 4KB */;
  restoreStruct.iNumBuffers = 1;
  restoreStruct.iCallerAction = DB2RESTORE_RESTORE;
  restoreStruct.iOptions = DB2RESTORE_OFFLINE | DB2RESTORE_DB |
  DB2RESTORE_NODATALINK | DB2RESTORE_NOROLLFWD;

  /* The API db2Restore is used to restore a database that has been backed
     up using the API db2Backup. */
  db2Restore (db2Version810, &restoreStruct, &sqlca);

  EXPECTED_WARN_CHECK("database restore -- start");

  while (sqlca.sqlcode != 0)
  {
    /* continue the restore operation */
    printf("\n  Continuing the restore operation...\n");

    /* depending on the sqlca.sqlcode value, user action may be
       required, such as mounting a new tape */

    restoreStruct.iCallerAction = DB2RESTORE_CONTINUE;

    /* restore the database */
    db2Restore (db2Version810, &restoreStruct, &sqlca);

    DB2_API_CHECK("database restore -- continue");
  }

  printf("\n  Restore finished.\n");

  return 0;
} /* DbBackupAndRestore */

int DbBackupAndRedirectedRestore(char dbAlias[],
                                 char restoredDbAlias[],
                                 char user[],
                                 char pswd[],
                                 char serverWorkingPath[])
{
  int rc = 0;
  struct sqlca sqlca;
  db2CfgParam cfgParameters[1];
  db2Cfg cfgStruct;
  unsigned short logretain;

  char restoreTimestamp[SQLU_TIME_STAMP_LEN + 1];

  db2BackupStruct  backupStruct;
  db2TablespaceStruct   tablespaceStruct;
  db2MediaListStruct mediaListStruct;
  db2Uint32  backupImageSize;
  db2RestoreStruct restoreStruct;
  db2TablespaceStruct   rtablespaceStruct;
  db2MediaListStruct rmediaListStruct;
```

```
printf("\n*************************\n");
printf("*** REDIRECTED RESTORE ***\n");
printf("*************************\n");
printf("\nUSE THE DB2 APIs:\n");
printf("  db2CfgSet -- Upate Configuration\n");
printf("  db2Backup -- Backup Database\n");
printf("  sqlecrea -- Create Database\n");
printf("  db2Restore -- Restore Database\n");
printf("  sqlbmtsq -- Tablespace Query\n");
printf("  sqlbtcq -- Tablespace Container Query\n");
printf("  sqlbstsc -- Set Tablespace Containers\n");
printf("  sqlefmem -- Free Memory\n");
printf("  sqledrpd -- Drop Database\n");
printf("TO BACK UP AND DO A REDIRECTED RESTORE OF A DATABASE.\n");

printf("\n  Update \'%s\' database configuration:\n", dbAlias);
printf("    - Disable the database configuration parameter LOGRETAIN \n");
printf("        i.e., set LOGRETAIN = OFF/NO\n");

/* initialize cfgParameters */
/* SQLF_DBTN_LOG_RETAIN is a token of the updatable database configuration
   parameter 'logretain'; it is used to update the database configuration
   file */
cfgParameters[0].flags = 0;
cfgParameters[0].token = SQLF_DBTN_LOG_RETAIN;
cfgParameters[0].ptrvalue = (char *)&logretain;

/* disable the database configuration parameter 'logretain' */
logretain = SQLF_LOGRETAIN_DISABLE;

/* initialize cfgStruct */
cfgStruct.numItems = 1;
cfgStruct.paramArray = cfgParameters;
cfgStruct.flags = db2CfgDatabase | db2CfgDelayed;
cfgStruct.dbname = dbAlias;

/* get database configuration */
db2CfgSet(db2Version810, (void *)&cfgStruct, &sqlca);
DB2_API_CHECK("Db Log Retain -- Disable");

/*******************************/
/*    BACK UP THE DATABASE     */
/*******************************/
printf("\n  Backing up the '%s' database...\n", dbAlias);

tablespaceStruct.tablespaces = NULL;
tablespaceStruct.numTablespaces = 0;

mediaListStruct.locations = &serverWorkingPath;
mediaListStruct.numLocations = 1;
mediaListStruct.locationType = SQLU_LOCAL_MEDIA;

backupStruct.piDBAlias = dbAlias;
backupStruct.piTablespaceList = &tablespaceStruct;
```

```
  backupStruct.piMediaList = &mediaListStruct;
  backupStruct.piUsername = user;
  backupStruct.piPassword = pswd;
  backupStruct.piVendorOptions = NULL;
  backupStruct.iVendorOptionsSize = 0;
  backupStruct.iCallerAction = DB2BACKUP_BACKUP;
  backupStruct.iBufferSize = 16; /*  16 x 4KB */
  backupStruct.iNumBuffers = 1;
  backupStruct.iParallelism =1;
  backupStruct.iOptions = DB2BACKUP_OFFLINE | DB2BACKUP_DB;

  /* The API db2Backup creates a backup copy of a database.
     This API automatically establishes a connection to the
     specified database,
     (This API can also be used to create a backup copy of a table space). */
  db2Backup (db2Version810, &backupStruct, &sqlca);

    DB2_API_CHECK("Database -- Backup");

  while (sqlca.sqlcode != 0)
  {
    /* continue the backup operation */

    /* depending on the sqlca.sqlcode value, user action may be
       required, such as mounting a new tape */

    printf("\n  Continuing the backup operation...\n");

    backupStruct.iCallerAction = DB2BACKUP_CONTINUE;

    /* back up the database */
    db2Backup (db2Version810, &backupStruct, &sqlca);
  }

  printf("  Backup finished.\n");
  printf("    - backup image size      : %d MB\n",
         backupStruct.oBackupSize);
  printf("    - backup image path      : %s\n",
         mediaListStruct.locations[0]);

  printf("    - backup image time stamp: %s\n", backupStruct.oTimestamp);

  /* To restore a remote database, you will first need to create an
     empty database if the client's code page is different from the
     server's code page.
     If this is the case, uncomment the call to DbCreate(). It will
     create an empty database on the server with the server's
     code page. */

/*
  rc = DbCreate(dbAlias, restoredDbAlias);
  if (rc != 0)
  {
    return rc;
  }
```

```
*/

  /*****************************/
  /*    RESTORE THE DATABASE    */
  /*****************************/

  strcpy(restoreTimestamp, backupStruct.oTimestamp);

  rtablespaceStruct.tablespaces = NULL;
  rtablespaceStruct.numTablespaces = 0;

  rmediaListStruct.locations = &serverWorkingPath;
  rmediaListStruct.numLocations = 1;
  rmediaListStruct.locationType = SQLU_LOCAL_MEDIA;

  restoreStruct.piSourceDBAlias = dbAlias;
  restoreStruct.piTargetDBAlias = restoredDbAlias;
  restoreStruct.piTimestamp = restoreTimestamp;
  restoreStruct.piTargetDBPath = NULL;
  restoreStruct.piReportFile = NULL;
  restoreStruct.piTablespaceList = &rtablespaceStruct;
  restoreStruct.piMediaList = &rmediaListStruct;
  restoreStruct.piUsername = user;
  restoreStruct.piPassword = pswd;
  restoreStruct.piNewLogPath = NULL;
  restoreStruct.piVendorOptions = NULL;
  restoreStruct.iVendorOptionsSize = 0;
  restoreStruct.iParallelism = 1;
  restoreStruct.iBufferSize = 1024; /*  1024 x 4KB */;
  restoreStruct.iNumBuffers = 1;
  restoreStruct.iOptions = DB2RESTORE_OFFLINE | DB2RESTORE_DB |
  DB2RESTORE_NODATALINK | DB2RESTORE_NOROLLFWD;

  printf("\n  Restoring a database ...\n");
  printf("    - source image alias     : %s\n", dbAlias);
  printf("    - source image time stamp: %s\n", restoreTimestamp);
  printf("    - target database        : %s\n", restoredDbAlias);

  restoreStruct.iCallerAction = DB2RESTORE_RESTORE_STORDEF;

  /* The API db2Restore is used to restore a database that has been backed
     up using the API db2Backup. */
  db2Restore(db2Version810, &restoreStruct, &sqlca);

  EXPECTED_WARN_CHECK("database restore -- start");

  while (sqlca.sqlcode != 0)
  {
    /* continue the restore operation */
    printf("\n  Continuing the restore operation...\n");

    /* depending on the sqlca.sqlcode value, user action may be
       required, such as mounting a new tape */

    if (sqlca.sqlcode == SQLUD_INACCESSABLE_CONTAINER)
```

```
    {
      /* redefine the table space container layout */
      printf("\n  Find and redefine inaccessable containers.\n");
      rc = InaccessableContainersRedefine(serverWorkingPath);
  if (rc != 0)
      {
    return rc;
      }
    }

    restoreStruct.iCallerAction = DB2RESTORE_CONTINUE;

    /* restore the database */
    db2Restore (db2Version810, &restoreStruct, &sqlca);

    DB2_API_CHECK("database restore -- continue");
  }

  printf("\n  Restore finished.\n");

  /* drop the restored database */
  rc = DbDrop(restoredDbAlias);

  return 0;
} /* DbBackupAndRedirectedRestore */

int InaccessableContainersRedefine(char serverWorkingPath[])
{
  int rc = 0;
  struct sqlca sqlca;
  sqluint32 numTablespaces;
  struct SQLB_TBSPQRY_DATA **ppTablespaces;
  sqluint32 numContainers;
  struct SQLB_TBSCONTQRY_DATA *pContainers;
  int tspNb;
  int contNb;
  char pathSep[2];

  /* The API sqlbmtsq provides a one-call interface to the table space query
     data. The query data for all table spaces in the database is returned
     in an array. */
  sqlbmtsq(&sqlca,
           &numTablespaces,
           &ppTablespaces,
           SQLB_RESERVED1,
           SQLB_RESERVED2);
  DB2_API_CHECK("tablespaces -- get");

  /* refedine the inaccessable containers */
  for (tspNb = 0; tspNb < numTablespaces; tspNb++)
  {
    /* The API sqlbtcq provides a one-call interface to the table space
       container query data. The query data for all the containers in a table
       space, or for all containers in all table spaces, is returned in an
       array. */
```

```
sqlbtcq(&sqlca, ppTablespaces[tspNb]->id, &numContainers, &pContainers);
DB2_API_CHECK("tablespace containers -- get");

for (contNb = 0; contNb < numContainers; contNb++)
{
  if (!pContainers[contNb].ok)
  {
    /* redefine inaccessable container */
    printf("\n    Redefine inaccessable container:\n");
    printf("      - table space name: %s\n",
           ppTablespaces[tspNb]->name);
    printf("      - default container name: %s\n",
           pContainers[contNb].name);
    if (strstr(pContainers[contNb].name, "/"))
    { /* UNIX */
      strcpy(pathSep, "/");
    }
  else
    { /* Intel */
      strcpy(pathSep, "\\");
    }
    switch (pContainers[contNb].contType)
    {
      case SQLB_CONT_PATH:
        printf("      - container type: path\n");

        sprintf(pContainers[contNb].name, "%s%sSQLT%04d.%d",
                                          serverWorkingPath, pathSep,
                                          ppTablespaces[tspNb]->id,
                                          pContainers[contNb].id);
        printf("      - new container name: %s\n",
               pContainers[contNb].name);
  break;
      case SQLB_CONT_DISK:
      case SQLB_CONT_FILE:
default:
        printf("    Unknown container type.\n");
  break;
    }
  }
}

/* The API sqlbstsc is used to set or redefine table space containers
   while performing a 'redirected' restore of the database. */
sqlbstsc(&sqlca,
         SQLB_SET_CONT_FINAL_STATE,
         ppTablespaces[tspNb]->id,
         numContainers,
         pContainers);
DB2_API_CHECK("tablespace containers -- redefine");

/* The API sqlefmem is used here to free memory allocated by DB2 for use
   with the API sqlbtcq (Tablespace Container Query). */
sqlefmem(&sqlca, pContainers);
DB2_API_CHECK("tablespace containers memory -- free");
```

```
  }

  /* The API sqlefmem is used here to free memory allocated by DB2 for
     use with the API sqlbmtsq (Tablespace Query). */
  sqlefmem(&sqlca, ppTablespaces);
  DB2_API_CHECK("tablespaces memory -- free");

  return 0;
} /* InaccessableContainersRedefine */

int DbBackupRestoreAndRollforward(char dbAlias[],
                                  char rolledForwardDbAlias[],
                                  char user[],
                                  char pswd[],
                                  char serverWorkingPath[])
{
  int rc = 0;
  struct sqlca sqlca;
  db2CfgParam cfgParameters[1];
  db2Cfg cfgStruct;
  unsigned short logretain;

  char restoreTimestamp[SQLU_TIME_STAMP_LEN + 1];

  db2BackupStruct  backupStruct;
  db2TablespaceStruct   tablespaceStruct;
  db2MediaListStruct mediaListStruct;
  db2Uint32  backupImageSize;
  db2RestoreStruct restoreStruct;
  db2TablespaceStruct   rtablespaceStruct;
  db2MediaListStruct rmediaListStruct;

  db2RfwdInputStruct   rfwdInput;
  db2RfwdOutputStruct  rfwdOutput;
  db2RollforwardStruct  rfwdStruct;

  char rollforwardAppId[SQLU_APPLID_LEN + 1];
  sqlint32 numReplies;
  struct sqlurf_info nodeInfo;

  printf("\n****************************\n");
  printf("*** ROLLFORWARD RECOVERY ***\n");
  printf("****************************\n");
  printf("\nUSE THE DB2 APIs:\n");
  printf("  db2CfgSet -- Set Configuration\n");
  printf("  db2Backup -- Backup Database\n");
  printf("  sqlecrea -- Create Database\n");
  printf("  db2Restore -- Restore Database\n");
  printf("  db2Rollforward -- Rollforward Database\n");
  printf("  sqledrpd -- Drop Database\n");
  printf("TO BACK UP, RESTORE, AND ROLL A DATABASE FORWARD. \n");

  printf("\n  Update \'%s\' database configuration:\n", dbAlias);
  printf("    - Enable the configuration parameter LOGRETAIN \n");
  printf("        i.e., set LOGRETAIN = RECOVERY/YES\n");
```

```
/* initialize cfgParameters */
cfgParameters[0].flags = 0;
cfgParameters[0].token = SQLF_DBTN_LOG_RETAIN;
cfgParameters[0].ptrvalue = (char *)&logretain;

/* enable the configuration parameter 'logretain' */
logretain = SQLF_LOGRETAIN_RECOVERY;

/* initialize cfgStruct */
cfgStruct.numItems = 1;
cfgStruct.paramArray = cfgParameters;
cfgStruct.flags = db2CfgDatabase | db2CfgDelayed;
cfgStruct.dbname = dbAlias;

/* get database configuration */
db2CfgSet(db2Version810, (void *)&cfgStruct, &sqlca);
DB2_API_CHECK("Db Log Retain -- Enable");

/* start the backup operation */
printf("\n  Backing up the '%s' database...\n", dbAlias);

tablespaceStruct.tablespaces = NULL;
tablespaceStruct.numTablespaces = 0;

mediaListStruct.locations = &serverWorkingPath;
mediaListStruct.numLocations = 1;
mediaListStruct.locationType = SQLU_LOCAL_MEDIA;

backupStruct.piDBAlias = dbAlias;
backupStruct.piTablespaceList = &tablespaceStruct;
backupStruct.piMediaList = &mediaListStruct;
backupStruct.piUsername = user;
backupStruct.piPassword = pswd;
backupStruct.piVendorOptions = NULL;
backupStruct.iVendorOptionsSize = 0;
backupStruct.iCallerAction = DB2BACKUP_BACKUP;
backupStruct.iBufferSize = 16; /*  16 x 4KB */
backupStruct.iNumBuffers = 1;
backupStruct.iParallelism =1;
backupStruct.iOptions = DB2BACKUP_OFFLINE | DB2BACKUP_DB;

/* The API db2Backup creates a backup copy of a database.
   This API automatically establishes a connection to the specified database.
   (This API can also be used to create a backup copy of a table space). */
db2Backup (db2Version810, &backupStruct, &sqlca);

  DB2_API_CHECK("Database -- Backup");

while (sqlca.sqlcode != 0)
{
  /* continue the backup operation */
  printf("\n  Continuing the backup operation...\n");

  /* depending on the sqlca.sqlcode value, user action may be
```

```
        required, such as mounting a new tape. */

      backupStruct.iCallerAction = DB2BACKUP_CONTINUE;

      /* back up the database */
      db2Backup (db2Version810, &backupStruct, &sqlca);

      DB2_API_CHECK("Database -- Backup");
    }

    printf("  Backup finished.\n");
    printf("    - backup image size      : %d MB\n",
           backupStruct.oBackupSize);
    printf("    - backup image path      : %s\n",
           mediaListStruct.locations[0]);

    printf("    - backup image time stamp: %s\n", backupStruct.oTimestamp);

    /* To restore a remote database, you will first need to create an
  empty database
       if the client's code page is different from the server's code page.
       If this is the case, uncomment the call to DbCreate(). It will create
       an empty database on the server with the server's code page. */

  /*
    rc = DbCreate(dbAlias, rolledForwardDbAlias);
    if (rc != 0)
    {
      return rc;
    }
  */

    /*****************************/
    /*    RESTORE THE DATABASE    */
    /*****************************/

    strcpy(restoreTimestamp, backupStruct.oTimestamp);

    rtablespaceStruct.tablespaces = NULL;
    rtablespaceStruct.numTablespaces = 0;

    rmediaListStruct.locations = &serverWorkingPath;
    rmediaListStruct.numLocations = 1;
    rmediaListStruct.locationType = SQLU_LOCAL_MEDIA;

    restoreStruct.piSourceDBAlias = dbAlias;
    restoreStruct.piTargetDBAlias = rolledForwardDbAlias;
    restoreStruct.piTimestamp = restoreTimestamp;
    restoreStruct.piTargetDBPath = NULL;
    restoreStruct.piReportFile = NULL;
    restoreStruct.piTablespaceList = &rtablespaceStruct;
    restoreStruct.piMediaList = &rmediaListStruct;
    restoreStruct.piUsername = user;
    restoreStruct.piPassword = pswd;
    restoreStruct.piNewLogPath = NULL;
```

```
restoreStruct.piVendorOptions = NULL;
restoreStruct.iVendorOptionsSize = 0;
restoreStruct.iParallelism = 1;
restoreStruct.iBufferSize = 1024; /*  1024 x 4KB */;
restoreStruct.iNumBuffers = 1;
restoreStruct.iCallerAction = DB2RESTORE_RESTORE;
restoreStruct.iOptions = DB2RESTORE_OFFLINE | DB2RESTORE_DB |
DB2RESTORE_NODATALINK | DB2RESTORE_ROLLFWD;

printf("\n  Restoring a database ...\n");
printf("    - source image alias     : %s\n", dbAlias);
printf("    - source image time stamp: %s\n", restoreTimestamp);
printf("    - target database        : %s\n", rolledForwardDbAlias);

/* The API db2Restore is used to restore a database that has been backed
   up using the API db2Backup. */
db2Restore (db2Version810, &restoreStruct, &sqlca);

DB2_API_CHECK("database restore -- start");

while (sqlca.sqlcode != 0)
{
  /* continue the restore operation */
  printf("\n  Continuing the restore operation...\n");

  /* Depending on the sqlca.sqlcode value, user action may be
     required, such as mounting a new tape. */

  restoreStruct.iCallerAction = DB2RESTORE_CONTINUE;

  /* restore the database */
  db2Restore (db2Version810, &restoreStruct, &sqlca);

  DB2_API_CHECK("database restore -- continue");
}

printf("\n  Restore finished.\n");


/*****************************/
/*    ROLLFORWARD RECOVERY    */
/*****************************/

printf("\n  Rolling '%s' database forward ...\n", rolledForwardDbAlias);

rfwdInput.version = SQLUM_RFWD_VERSION;
rfwdInput.pDbAlias = rolledForwardDbAlias;
rfwdInput.CallerAction = SQLUM_ROLLFWD_STOP;
rfwdInput.pStopTime = SQLUM_INFINITY_TIMESTAMP;
rfwdInput.pUserName = user;
rfwdInput.pPassword = pswd;
rfwdInput.pOverflowLogPath = serverWorkingPath;
rfwdInput.NumChngLgOvrflw = 0;
rfwdInput.pChngLogOvrflw = NULL;
rfwdInput.ConnectMode = SQLUM_OFFLINE;
```

```
  rfwdInput.pTablespaceList = NULL;
  rfwdInput.AllNodeFlag = SQLURF_ALL_NODES;
  rfwdInput.NumNodes = 0;
  rfwdInput.pNodeList = NULL;
  rfwdInput.pDroppedTblID = NULL;
  rfwdInput.pExportDir = NULL;
  rfwdInput.NumNodeInfo = 1;
  rfwdInput.RollforwardFlags = 0;

  rfwdOutput.pApplicationId = rollforwardAppId;
  rfwdOutput.pNumReplies = &numReplies;
  rfwdOutput.pNodeInfo = &nodeInfo;

  rfwdStruct.roll_input = &rfwdInput;
  rfwdStruct.roll_output = &rfwdOutput;

  /* rollforward database */
  /* The API db2Rollforward rollforward recovers a database by
     applying transactions recorded in the database log files. */
  db2Rollforward(db2Version810, &rfwdStruct, &sqlca);

  DB2_API_CHECK("rollforward -- start");

  printf("  Rollforward finished.\n");

  /* drop the restored database */
  rc = DbDrop(rolledForwardDbAlias);

  return 0;
} /* DbBackupRestoreAndRollforward */

int DbLogRecordsForCurrentConnectionRead(char dbAlias[],
                                         char user[],
                                         char pswd[],
                                         char serverWorkingPath[])
{
  int rc = 0;
  struct sqlca sqlca;
  db2CfgParam cfgParameters[1];
  db2Cfg cfgStruct;
  unsigned short logretain;

  db2BackupStruct  backupStruct;
  db2TablespaceStruct   tablespaceStruct;
  db2MediaListStruct mediaListStruct;
  db2Uint32  backupImageSize;
  db2RestoreStruct restoreStruct;
  db2TablespaceStruct   rtablespaceStruct;
  db2MediaListStruct rmediaListStruct;

  SQLU_LSN startLSN;
  SQLU_LSN endLSN;
  char *logBuffer;
  sqluint32 logBufferSize;
  db2ReadLogInfoStruct readLogInfo;
```

```
db2ReadLogStruct     readLogInput;
int i;

printf("\n*****************************\n");
printf("*** ASYNCHRONOUS READ LOG ***\n");
printf("*****************************\n");
printf("\nUSE THE DB2 APIs:\n");
printf("  db2CfgSet -- Set Configuration\n");
printf("  db2Backup -- Backup Database\n");
printf("  db2ReadLog -- Asynchronous Read Log\n");
printf("AND THE SQL STATEMENTS:\n");
printf("  CONNECT\n");
printf("  ALTER TABLE\n");
printf("  COMMIT\n");
printf("  INSERT\n");
printf("  DELETE\n");
printf("  ROLLBACK\n");
printf("  CONNECT RESET\n");
printf("TO READ LOG RECORDS FOR THE CURRENT CONNECTION.\n");

printf("\n  Update \'%s\' database configuration:\n", dbAlias);
printf("    - Enable the database configuration parameter LOGRETAIN \n");
printf("        i.e., set LOGRETAIN = RECOVERY/YES\n");

/* initialize cfgParameters */
cfgParameters[0].flags = 0;
cfgParameters[0].token = SQLF_DBTN_LOG_RETAIN;
cfgParameters[0].ptrvalue = (char *)&logretain;

/* enable LOGRETAIN */
logretain = SQLF_LOGRETAIN_RECOVERY;

/* initialize cfgStruct */
cfgStruct.numItems = 1;
cfgStruct.paramArray = cfgParameters;
cfgStruct.flags = db2CfgDatabase | db2CfgDelayed;
cfgStruct.dbname = dbAlias;

/* get database configuration */
db2CfgSet(db2Version810, (void *)&cfgStruct, &sqlca);
DB2_API_CHECK("Db Log Retain -- Enable");

/* start the backup operation */
printf("\n  Backing up the '%s' database...\n", dbAlias);

tablespaceStruct.tablespaces = NULL;
tablespaceStruct.numTablespaces = 0;

mediaListStruct.locations = &serverWorkingPath;
mediaListStruct.numLocations = 1;
mediaListStruct.locationType = SQLU_LOCAL_MEDIA;

backupStruct.piDBAlias = dbAlias;
backupStruct.piTablespaceList = &tablespaceStruct;
backupStruct.piMediaList = &mediaListStruct;
```

```
backupStruct.piUsername = user;
backupStruct.piPassword = pswd;
backupStruct.piVendorOptions = NULL;
backupStruct.iVendorOptionsSize = 0;
backupStruct.iCallerAction = DB2BACKUP_BACKUP;
backupStruct.iBufferSize = 16; /*  16 x 4KB */
backupStruct.iNumBuffers = 1;
backupStruct.iParallelism =1;
backupStruct.iOptions = DB2BACKUP_OFFLINE | DB2BACKUP_DB;

/* The API db2Backup creates a backup copy of a database.
   This API automatically establishes a connection to the specified database.
   (This API can also be used to create a backup copy of a table space). */
db2Backup (db2Version810, &backupStruct, &sqlca);

DB2_API_CHECK("Database -- Backup");

while (sqlca.sqlcode != 0)
{
  /* continue the backup operation */
  printf("\n  Continuing the backup operation...\n");

  /* Depending on the sqlca.sqlcode value, user action may be
     required, such as mounting a new tape. */

  backupStruct.iCallerAction = DB2BACKUP_CONTINUE;

  /* back up the database */
  db2Backup (db2Version810, &backupStruct, &sqlca);

  DB2_API_CHECK("Database -- Backup");
}

printf("  Backup finished.\n");
printf("    - backup image size      : %d MB\n", backupStruct.oBackupSize);
printf("    - backup image path      : %s\n", mediaListStruct.locations[0]);

printf("    - backup image time stamp: %s\n", backupStruct.oTimestamp);

/* connect to the database */
rc = DbConn(dbAlias, user, pswd);
if (rc != 0)
{
  return rc;
}

/* invoke SQL statements to fill database log */
printf("\n  Invoke the following SQL statements:\n"
       "    ALTER TABLE emp_resume DATA CAPTURE CHANGES;\n"
       "    COMMIT;\n"
       "    INSERT INTO emp_resume\n"
       "      VALUES('000777', 'ascii', 'This is a new resume.');\n"
       "            ('777777', 'ascii', 'This is another new resume');\n"
       "    COMMIT;\n"
       "    DELETE FROM emp_resume WHERE empno = '000777';\n"
```

```
        "    DELETE FROM emp_resume WHERE empno = '777777';\n"
        "    COMMIT;\n"
        "    DELETE FROM emp_resume WHERE empno = '000140';\n"
        "    ROLLBACK;\n"
        "    ALTER TABLE emp_resume DATA CAPTURE NONE;\n"
        "    COMMIT;\n");

EXEC SQL ALTER TABLE emp_resume DATA CAPTURE CHANGES;
EMB_SQL_CHECK("SQL statement 1 -- invoke");

EXEC SQL COMMIT;
EMB_SQL_CHECK("SQL statement 2 -- invoke");

EXEC SQL INSERT INTO emp_resume
  VALUES('000777', 'ascii', 'This is a new resume.'),
        ('777777', 'ascii', 'This is another new resume');
EMB_SQL_CHECK("SQL statement 3 -- invoke");

EXEC SQL COMMIT;
EMB_SQL_CHECK("SQL statement 4 -- invoke");

EXEC SQL DELETE FROM emp_resume WHERE empno = '000777';
EMB_SQL_CHECK("SQL statement 5 -- invoke");

EXEC SQL DELETE FROM emp_resume WHERE empno = '777777';
EMB_SQL_CHECK("SQL statement 6 -- invoke");

EXEC SQL COMMIT;
EMB_SQL_CHECK("SQL statement 7 -- invoke");

EXEC SQL DELETE FROM emp_resume WHERE empno = '000140';
EMB_SQL_CHECK("SQL statement 8 -- invoke");

EXEC SQL ROLLBACK;
EMB_SQL_CHECK("SQL statement 9 -- invoke");

EXEC SQL ALTER TABLE emp_resume DATA CAPTURE NONE;
EMB_SQL_CHECK("SQL statement 10 -- invoke");

EXEC SQL COMMIT;
EMB_SQL_CHECK("SQL statement 11 -- invoke");

printf("\n  Start reading database log.\n");

logBuffer = NULL;
logBufferSize = 0;

/* The API db2ReadLog (Asynchronous Read Log) is used to extract records
   from the database logs, and to query the log manager for current
   log state information.
   This API can only be used on recoverable databases. */

/* Query the log manager for current log state information. */
readLogInput.iCallerAction  = DB2READLOG_QUERY;
readLogInput.piStartLSN     = NULL;
```

```
readLogInput.piEndLSN       = NULL;
readLogInput.poLogBuffer    = NULL;
readLogInput.iLogBufferSize = 0;
readLogInput.iFilterOption  = DB2READLOG_FILTER_ON;
readLogInput.poReadLogInfo  = &readLogInfo;

rc = db2ReadLog(db2Version810,
                &readLogInput,
           &sqlca);

DB2_API_CHECK("database log info -- get");

logBufferSize = 64 * 1024;
logBuffer = (char *)malloc(logBufferSize);

memcpy(&startLSN, &(readLogInfo.initialLSN), sizeof(startLSN));
memcpy(&endLSN, &(readLogInfo.nextStartLSN), sizeof(endLSN));

  /* Extract a log record from the database logs, and
   read the first log sequence asynchronously. */
readLogInput.iCallerAction  = DB2READLOG_READ;
readLogInput.piStartLSN     = &startLSN;
readLogInput.piEndLSN       = &endLSN;
readLogInput.poLogBuffer    = logBuffer;
readLogInput.iLogBufferSize = logBufferSize;
readLogInput.iFilterOption  = DB2READLOG_FILTER_ON;
readLogInput.poReadLogInfo  = &readLogInfo;

rc = db2ReadLog(db2Version810,
                &readLogInput,
           &sqlca);
  if (sqlca.sqlcode != SQLU_RLOG_READ_TO_CURRENT)
{
    DB2_API_CHECK("database logs -- read");
}
    else
{
  if (readLogInfo.logRecsWritten == 0)
  {
    printf("\n  Database log empty.\n");
  }
}

  /* display log buffer */
  rc = LogBufferDisplay(logBuffer, readLogInfo.logRecsWritten);

while (sqlca.sqlcode != SQLU_RLOG_READ_TO_CURRENT)
{
  /* read the next log sequence */

  memcpy(&startLSN, &(readLogInfo.nextStartLSN), sizeof(startLSN));

  /* Extract a log record from the database logs, and
   read the next log sequence asynchronously. */
   rc = db2ReadLog(db2Version810,
```

```
                    &readLogInput,
              &sqlca);
    if (sqlca.sqlcode != SQLU_RLOG_READ_TO_CURRENT)
    {
      DB2_API_CHECK("database logs -- read");
    }

    /* display log buffer */
    rc = LogBufferDisplay(logBuffer, readLogInfo.logRecsWritten);
  }

  /* free the log buffer */
  free(logBuffer);

  /* disconnect from the database */
  rc = DbDisconn(dbAlias);
  if (rc != 0)
  {
    return rc;
  }

  return 0;
} /* DbLogRecordsForCurrentConnectionRead */

int DbReadLogRecordsNoConn(char dbAlias[])
{
  int rc = 0;
  struct sqlca sqlca;
  char      logPath[SQL_PATH_SZ + 1];
  db2CfgParam cfgParameters[1];
  db2Cfg    cfgStruct;
  char      nodeName[] = "NODE0000\0";
  db2Uint32 readLogMemSize = 4 * 4096;
  char      *readLogMemory = NULL;
  struct db2ReadLogNoConnInitStruct readLogInit;
  struct db2ReadLogNoConnInfoStruct readLogInfo;
  struct db2ReadLogNoConnStruct readLogInput;
  SQLU_LSN startLSN;
  SQLU_LSN endLSN;
  char      *logBuffer = NULL;
  db2Uint32 logBufferSize = 0;
  struct db2ReadLogNoConnTermStruct readLogTerm;

  printf("\n*********************************\n");
  printf("*** NO DB CONNECTION READ LOG ***\n");
  printf("*********************************\n");
  printf("\nUSE THE DB2 APIs:\n");
  printf("  db2ReadLogNoConnInit -- Initialize No Db Connection Read Log\n");
  printf("  db2ReadLogNoConn -- No Db Connection Read Log\n");
  printf("  db2ReadLogNoConnTerm -- Terminate No Db Connection Read Log\n");
  printf("TO READ LOG RECORDS FROM A DATABASE LOG DIRECTORY.\n");

  /* Determine the logpath to read log files from */
  cfgParameters[0].flags = 0;
  cfgParameters[0].token = SQLF_DBTN_LOGPATH;
```

```
cfgParameters[0].ptrvalue =
  (char *)malloc((SQL_PATH_SZ + 1) * sizeof(char));

/* Initialize cfgStruct */
cfgStruct.numItems = 1;
cfgStruct.paramArray = cfgParameters;
cfgStruct.flags = db2CfgDatabase;
cfgStruct.dbname = dbAlias;

db2CfgGet(db2Version810, (void *)&cfgStruct, &sqlca);
DB2_API_CHECK("log path -- get");

strcpy(logPath, cfgParameters[0].ptrvalue);
free(cfgParameters[0].ptrvalue);

/* First we must allocate memory for the API's control blocks and log
   buffer */
readLogMemory = (char*)malloc(readLogMemSize);

/* Invoke the initialization API to set up the control blocks */
readLogInit.iFilterOption       = DB2READLOG_FILTER_ON;
readLogInit.piLogFilePath       = logPath;
readLogInit.piOverflowLogPath   = NULL;
readLogInit.iRetrieveLogs       = DB2READLOGNOCONN_RETRIEVE_OFF;
readLogInit.piDatabaseName      = dbAlias;
readLogInit.piNodeName          = nodeName;
readLogInit.iReadLogMemoryLimit = readLogMemSize;
readLogInit.poReadLogMemPtr     = &readLogMemory;

rc = db2ReadLogNoConnInit(db2Version810,
                          &readLogInit,
           &sqlca);
if (sqlca.sqlcode != SQLU_RLOG_LSNS_REUSED)
{
  DB2_API_CHECK("database logs no db conn -- initialization");
}

/* Query for the current log information */
readLogInput.iCallerAction   = DB2READLOG_QUERY;
readLogInput.piStartLSN      = NULL;
readLogInput.piEndLSN        = NULL;
readLogInput.poLogBuffer     = NULL;
readLogInput.iLogBufferSize  = 0;
readLogInput.piReadLogMemPtr = readLogMemory;
readLogInput.poReadLogInfo   = &readLogInfo;

rc = db2ReadLogNoConn(db2Version810,
                      &readLogInput,
           &sqlca);
if (sqlca.sqlcode != 0)
{
  DB2_API_CHECK("database logs no db conn -- query");
}

/* Read some log records */
```

```
logBufferSize = 64 * 1024;
logBuffer = (char *)malloc(logBufferSize);

memcpy(&startLSN, &(readLogInfo.nextStartLSN), sizeof(startLSN));
endLSN.lsnWord[0] = 0xffff;
endLSN.lsnWord[1] = 0xffff;
endLSN.lsnWord[2] = 0xffff;

readLogInput.iCallerAction   = DB2READLOG_READ;
readLogInput.piStartLSN      = &startLSN;
readLogInput.piEndLSN        = &endLSN;
readLogInput.poLogBuffer     = logBuffer;
readLogInput.iLogBufferSize  = logBufferSize;
readLogInput.piReadLogMemPtr = readLogMemory;
readLogInput.poReadLogInfo   = &readLogInfo;

rc = db2ReadLogNoConn(db2Version810,
                      &readLogInput,
           &sqlca);
  if (sqlca.sqlcode != SQLU_RLOG_READ_TO_CURRENT)
{
  DB2_API_CHECK("database logs no db conn -- read");
}
    else
{
  if (readLogInfo.logRecsWritten == 0)
  {
    printf("\n  Database log empty.\n");
  }
}

/* Display the log records read */
  rc = LogBufferDisplay(logBuffer, readLogInfo.logRecsWritten);

while (sqlca.sqlcode != SQLU_RLOG_READ_TO_CURRENT)
{
  /* read the next log sequence */
  memcpy(&startLSN, &(readLogInfo.nextStartLSN), sizeof(startLSN));

  /* Extract a log record from the database logs, and
   read the next log sequence asynchronously. */
  rc = db2ReadLogNoConn(db2Version810,
                        &readLogInput,
           &sqlca);
  if (sqlca.sqlcode != SQLU_RLOG_READ_TO_CURRENT)
  {
    DB2_API_CHECK("database logs no db conn -- read");
  }

  /* display log buffer */
  rc = LogBufferDisplay(logBuffer, readLogInfo.logRecsWritten);
}

printf("\nRead to end of logs.\n\n");
free(logBuffer);
```

```
  readLogTerm.poReadLogMemPtr = &readLogMemory;

  rc = db2ReadLogNoConnTerm(db2Version810,
                            &readLogTerm,
            &sqlca);
  if (sqlca.sqlcode != 0)
  {
    DB2_API_CHECK("database logs no db conn -- terminate");
  }

  return 0;
} /* DbReadLogRecordsNoConn */

int LogBufferDisplay(char *logBuffer, sqluint32 numLogRecords)
{
  int rc = 0;
  sqluint32 logRecordNb;
  sqluint32 recordSize;
  sqluint16 recordType;
  sqluint16 recordFlag;
  char *recordBuffer;

  /* initialize recordBuffer */
  recordBuffer = logBuffer + sizeof(SQLU_LSN);

  for (logRecordNb = 0; logRecordNb < numLogRecords; logRecordNb++)
  {
    recordSize = *(sqluint32 *)(recordBuffer);
    recordType = *(sqluint16 *)(recordBuffer + 4);
    recordFlag = *(sqluint16 *)(recordBuffer + 6);

    rc = LogRecordDisplay(recordBuffer, recordSize, recordType, recordFlag);
    /* update recordBuffer */
    recordBuffer = recordBuffer + recordSize + sizeof(SQLU_LSN);
  }

  return 0;
} /* LogBufferDisplay */

int LogRecordDisplay(char *recordBuffer,
                     sqluint32 recordSize,
                     sqluint16 recordType,
                     sqluint16 recordFlag)
{
  int rc = 0;
  sqluint32 logManagerLogRecordHeaderSize;
  char *recordDataBuffer;
  sqluint32 recordDataSize;
  char *recordHeaderBuffer;
  sqluint8 componentIdentifier;
  sqluint32 recordHeaderSize;

  /* determine logManagerLogRecordHeaderSize */
  if (recordType == 0x0043)
```

```
{ /* compensation */
  if (recordFlag & 0x0002)
  { /* propagatable */
    logManagerLogRecordHeaderSize = 32;
  }
    else
  {
    logManagerLogRecordHeaderSize = 26;
  }
}
    else
{ /* non compensation */
  logManagerLogRecordHeaderSize = 20;
}

switch (recordType)
{
  case 0x008A:
  case 0x0084:
  case 0x0041:
    recordDataBuffer = recordBuffer + logManagerLogRecordHeaderSize;
    recordDataSize = recordSize - logManagerLogRecordHeaderSize;
    rc = SimpleLogRecordDisplay(recordType,
                                recordFlag,
                                recordDataBuffer,
                                recordDataSize);
    break;
  case 0x004E:
  case 0x0043:
    recordHeaderBuffer = recordBuffer + logManagerLogRecordHeaderSize;
    componentIdentifier = *(sqluint8 *)recordHeaderBuffer;
switch (componentIdentifier)
    {
  case 1:
       recordHeaderSize = 6;
    break;
  default:
       printf("    Unknown complex log record: %lu %c %u\n",
              recordSize, recordType, componentIdentifier);
       return 1;
    }
    recordDataBuffer = recordBuffer +
                       logManagerLogRecordHeaderSize +
                     recordHeaderSize;
    recordDataSize = recordSize -
                     logManagerLogRecordHeaderSize -
                     recordHeaderSize;
    rc = ComplexLogRecordDisplay(recordType,
                                recordFlag,
                                recordHeaderBuffer,
                                recordHeaderSize,
                                componentIdentifier,
                                recordDataBuffer,
                                recordDataSize);
    break;
```

```
    default:
      printf("    Unknown log record: %lu \"%c\"\n",
             recordSize, (char)recordType);
      break;
  }

  return 0;
} /* LogRecordDisplay */

int SimpleLogRecordDisplay(sqluint16 recordType,
                           sqluint16 recordFlag,
                           char *recordDataBuffer,
                           sqluint32 recordDataSize)
{
  int rc = 0;
  sqluint32 timeTransactionCommited;
  sqluint16 authIdLen;
  char authId[129];

  switch (recordType)
  {
    case 138:
      printf("\n    Record type: Local pending list\n");
      timeTransactionCommited = *(sqluint32 *)(recordDataBuffer);
      authIdLen = *(sqluint16 *)(recordDataBuffer + 4);
      memcpy(authId, (char *)(recordDataBuffer + 6), authIdLen);
      authId[authIdLen] = '\0';
      printf("      %s: %lu\n",
             "UTC transaction committed (in seconds since 01/01/70)",
             timeTransactionCommited);
      printf("      authorization ID of the application: %s\n", authId);
      break;
    case 132:
      printf("\n    Record type: Normal commit\n");
      timeTransactionCommited = *(sqluint32 *)(recordDataBuffer);
      authIdLen = (sqluint16) (recordDataSize - 4);
      memcpy(authId, (char *)(recordDataBuffer + 4), authIdLen);
      authId[authIdLen] = '\0';
      printf("      %s: %lu\n",
             "UTC transaction committed (in seconds since 01/01/70)",
             timeTransactionCommited);
      printf("      authorization ID of the application: %s\n", authId);
      break;
    case 65:
      printf("\n    Record type: Normal abort\n");
      authIdLen = (sqluint16) (recordDataSize);
      memcpy(authId, (char *)(recordDataBuffer), authIdLen);
      authId[authIdLen] = '\0';
      printf("      authorization ID of the application: %s\n", authId);
      break;
    default:
      printf("    Unknown simple log record: %d %lu\n",
             recordType, recordDataSize);
      break;
  }
```

```
  return 0;
} /* SimpleLogRecordDisplay */

int ComplexLogRecordDisplay(sqluint16 recordType,
                            sqluint16 recordFlag,
                            char *recordHeaderBuffer,
                            sqluint32 recordHeaderSize,
                            sqluint8 componentIdentifier,
                            char *recordDataBuffer,
                            sqluint32 recordDataSize)
{
  int rc = 0;
  sqluint8 functionIdentifier;
  /* for insert, delete, undo delete */
  sqluint32 RID;
  sqluint16 subRecordLen;
  sqluint16 subRecordOffset;
  char *subRecordBuffer;
  /* for update */
  sqluint32 newRID;
  sqluint16 newSubRecordLen;
  sqluint16 newSubRecordOffset;
  char *newSubRecordBuffer;
  sqluint32 oldRID;
  sqluint16 oldSubRecordLen;
  sqluint16 oldSubRecordOffset;
  char *oldSubRecordBuffer;
  /* for alter table attributes */
  sqluint32 alterBitMask;
  sqluint32 alterBitValues;

  switch ((char)recordType)
  {
    case 'N':
      printf("\n    Record type: Normal\n");
      break;
    case 'C':
      printf("\n    Record type: Compensation\n");
      break;
    default:
      printf("\n    Unknown complex log record type: %c\n", recordType);
      break;
  }

  switch (componentIdentifier)
  {
    case 1:
      printf("      component ID: DMS log record\n");
      break;
    default:
      printf("      unknown component ID: %d\n", componentIdentifier);
      break;
  }
```

```
functionIdentifier = *(sqluint8 *)(recordHeaderBuffer + 1);
switch (functionIdentifier)
{
  case 106:
    printf("      function ID: Delete Record\n");
    RID = *(sqluint32 *)(recordDataBuffer + 2);
    subRecordLen = *(sqluint16 *)(recordDataBuffer + 6);
    subRecordOffset = *(sqluint16 *)(recordDataBuffer + 10);
    printf("        RID: %lu\n", RID);
    printf("        subrecord length: %u\n", subRecordLen);
    printf("        subrecord offset: %u\n", subRecordOffset);
    subRecordBuffer = recordDataBuffer + 12;
    rc = LogSubRecordDisplay(subRecordBuffer, subRecordLen);
    break;
  case 111:
    printf("      function ID: Undo Delete Record\n");
    RID = *(sqluint32 *)(recordDataBuffer + 2);
    subRecordLen = *(sqluint16 *)(recordDataBuffer + 6);
    subRecordOffset = *(sqluint16 *)(recordDataBuffer + 10);
    printf("        RID: %lu\n", RID);
    printf("        subrecord length: %u\n", subRecordLen);
    printf("        subrecord offset: %u\n", subRecordOffset);
    subRecordBuffer = recordDataBuffer + 12;
    rc = LogSubRecordDisplay(subRecordBuffer, subRecordLen);
    break;
  case 118:
    printf("      function ID: Insert Record\n");
    RID = *(sqluint32 *)(recordDataBuffer + 2);
    subRecordLen = *(sqluint16 *)(recordDataBuffer + 6);
    subRecordOffset = *(sqluint16 *)(recordDataBuffer + 10);
    printf("        RID: %lu\n", RID);
    printf("        subrecord length: %u\n", subRecordLen);
    printf("        subrecord offset: %u\n", subRecordOffset);
    subRecordBuffer = recordDataBuffer + 12;
    rc = LogSubRecordDisplay(subRecordBuffer, subRecordLen);
    break;
  case 120:
    printf("      function ID: Update Record\n");
    oldRID = *(sqluint32 *)(recordDataBuffer + 2);
    oldSubRecordLen = *(sqluint16 *)(recordDataBuffer + 6);
    oldSubRecordOffset = *(sqluint16 *)(recordDataBuffer + 10);
    newRID = *(sqluint32 *)(recordDataBuffer +
                            12 +
                            oldSubRecordLen +
                            recordHeaderSize +
                            2);
    newSubRecordLen = *(sqluint16 *)(recordDataBuffer +
                                     12 +
                                     oldSubRecordLen +
                                     recordHeaderSize +
                                     6);
    newSubRecordOffset = *(sqluint16 *)(recordDataBuffer +
                                        12 +
                                        oldSubRecordLen +
                                        recordHeaderSize +
```

```
                                        10);
  printf("        oldRID: %lu\n", oldRID);
  printf("        old subrecord length: %u\n", oldSubRecordLen);
  printf("        old subrecord offset: %u\n", oldSubRecordOffset);
  oldSubRecordBuffer = recordDataBuffer + 12;
  rc = LogSubRecordDisplay(oldSubRecordBuffer, oldSubRecordLen);
  printf("        newRID: %lu\n", newRID);
  printf("        new subrecord length: %u\n", newSubRecordLen);
  printf("        new subrecord offset: %u\n", newSubRecordOffset);
  newSubRecordBuffer = recordDataBuffer +
                       12 +
                       oldSubRecordLen +
                       recordHeaderSize +
                       12;
  rc = LogSubRecordDisplay(newSubRecordBuffer, newSubRecordLen);
  break;
case 124:
  printf("      function ID:  Alter Table Attribute\n");
  alterBitMask = *(sqluint32 *)(recordDataBuffer + 2);
  alterBitValues = *(sqluint32 *)(recordDataBuffer + 6);
  if (alterBitMask & 0x00000001)
  {
    /* Alter the value of the 'propagation' attribute: */
    printf("        Propagation attribute is changed to: ");
    if (alterBitValues & 0x00000001)
    {
      printf("ON\n");
    }
  else
    {
      printf("OFF\n");
    }
  }
  if (alterBitMask & 0x00000002)
  {
    /* Alter the value of the 'pending' attribute: */
    printf("        Pending attribute is changed to: ");
    if (alterBitValues & 0x00000002)
    {
      printf("ON\n");
    }
  else
    {
      printf("OFF\n");
    }
  }
  if (alterBitMask & 0x00010000)
  {
    /* Alter the value of the 'append mode' attribute: */
    printf("        Append Mode attribute is changed to: ");
    if (alterBitValues & 0x00010000)
    {
      printf("ON\n");
    }
  else
```

```
        {
          printf("OFF\n");
        }
      }
      if (alterBitMask & 0x00200000)
      {
        /* Alter the value of the 'LF Propagation' attribute: */
        printf("        LF Propagation attribute is changed to: ");
        if (alterBitValues & 0x00200000)
        {
          printf("ON\n");
        }
      else
        {
          printf("OFF\n");
        }
      }
      if (alterBitMask & 0x00400000)
      {
        /* Alter the value of the 'LOB Propagation' attribute: */
        printf("        LOB Propagation attribute is changed to: ");
        if (alterBitValues & 0x00400000)
        {
          printf("ON\n");
        }
      else
        {
          printf("OFF\n");
        }
      }
      break;
    default:
      printf("      unknown function identifier: %u\n",
             functionIdentifier);
      break;
  }

  return 0;
} /* ComplexLogRecordDisplay */

int LogSubRecordDisplay(char *recordBuffer, sqluint16 recordSize)
{
  int rc = 0;
  sqluint8 recordType;
  sqluint8 updatableRecordType;
  sqluint16 userDataFixedLength;
  char *userDataBuffer;
  sqluint16 userDataSize;

  recordType = *(sqluint8 *)(recordBuffer);
  if ((recordType != 0) &&
      (recordType != 4) &&
      (recordType != 16))
  {
    printf("        Unknown subrecord type: %x\n", recordType);
```

```
  }
  else if (recordType == 4)
  {
    printf("        subrecord type: Special control\n");
  }
     else
  {
    /* recordType == 0 or recordType == 16
     * record Type 0 indicates a normal record
     * record Type 16, for the purposes of this program, should be treated
     * as type 0
     */
    printf("        subrecord type: Updatable, ");
    updatableRecordType = *(sqluint8 *)(recordBuffer + 4);
    if (updatableRecordType != 1)
    {
      printf("Internal control\n");
    }
      else
    {
      printf("Formatted user data\n");
      userDataFixedLength = *(sqluint16 *)(recordBuffer + 6);
      printf("        user data fixed length: %u\n",
             userDataFixedLength);
      userDataBuffer = recordBuffer + 8;
      userDataSize = recordSize - 8;
      rc = UserDataDisplay(userDataBuffer, userDataSize);
    }
  }

  return 0;
} /* LogSubRecordDisplay */

int UserDataDisplay(char *dataBuffer, sqluint16 dataSize)
{
  int rc = 0;

  sqluint16 line, col;

  printf("        user data:\n");

  for (line = 0; line * 10 < dataSize; line = line + 1)
  {
       printf(" ");
    for (col = 0; col < 10; col = col + 1)
    {
      if (line * 10 + col < dataSize)
      {
        printf("%02X ", dataBuffer[line * 10 + col]);
      }
      else
      {
        printf(" ");
      }
    }
```

```
    printf("*");
    for (col = 0; col < 10; col = col + 1)
    {
      if (line * 10 + col < dataSize)
      {
        if (isalpha(dataBuffer[line * 10 + col]) ||
            isdigit(dataBuffer[line * 10 + col]))
        {
          printf("%c", dataBuffer[line * 10 + col]);
        }
      else
        {
          printf(".");
        }
      }
      else
      {
        printf(" ");
      }
    }
    printf("*");
    printf("\n");
  }

  return 0;
} /* UserDataDisplay */

int DbRecoveryHistoryFileRead(char dbAlias[])
{
  int rc = 0;
  struct sqlca sqlca;
  struct db2HistoryOpenStruct dbHistoryOpenParam;
  sqluint32 numEntries;
  sqluint16 recoveryHistoryFileHandle;
  sqluint32 entryNb;
  struct db2HistoryGetEntryStruct dbHistoryEntryGetParam;
  struct db2HistoryData histEntryData;

  printf("\n*******************************************\n");
  printf("*** READ A DATABASE RECOVERY HISTORY FILE ***\n");
  printf("*******************************************\n");
  printf("\nUSE THE DB2 APIs:\n");
  printf("  db2HistoryOpenScan -- Open Recovery History File Scan\n");
  printf("  db2HistoryGetEntry -- Get Next Recovery History File Entry\n");
  printf("  db2HistoryCloseScan -- Close Recovery History File Scan\n");
  printf("TO READ A DATABASE RECOVERY HISTORY FILE.\n");

  /* initialize the data structures */
  dbHistoryOpenParam.piDatabaseAlias = dbAlias;
  dbHistoryOpenParam.piTimestamp = NULL;
  dbHistoryOpenParam.piObjectName = NULL;
  dbHistoryOpenParam.iCallerAction = DB2HISTORY_LIST_HISTORY;

  dbHistoryEntryGetParam.pioHistData = &histEntryData;
  dbHistoryEntryGetParam.iCallerAction = DB2HISTORY_GET_ALL;
```

```
  rc = HistoryEntryDataFieldsAlloc(&histEntryData);
  if (rc != 0)
  {
    return rc;
  }

  /*****************************************/
  /* OPEN THE DATABASE RECOVERY HISTORY FILE */
  /*****************************************/
  printf("\n  Open recovery history file for '%s' database.\n", dbAlias);

  /* open the recovery history file to scan */
  db2HistoryOpenScan(db2Version810, &dbHistoryOpenParam, &sqlca);
  DB2_API_CHECK("database recovery history file -- open");

  numEntries = dbHistoryOpenParam.oNumRows;

  /* dbHistoryOpenParam.oHandle returns the handle for scan access */
  recoveryHistoryFileHandle = dbHistoryOpenParam.oHandle;
  dbHistoryEntryGetParam.iHandle = recoveryHistoryFileHandle;

  /*********************************************/
  /* READ AN ENTRY IN THE RECOVERY HISTORY FILE */
  /*********************************************/
  for (entryNb = 0; entryNb < numEntries; entryNb = entryNb + 1)
  {
    printf("\n  Read entry number %u.\n", entryNb);

    /* get the next entry from the recovery history file */
    db2HistoryGetEntry(db2Version810, &dbHistoryEntryGetParam, &sqlca);
    DB2_API_CHECK("database recovery history file entry -- read")

    /* display the entries in the recovery history file */
    printf("\n  Display entry number %u.\n", entryNb);
  rc = HistoryEntryDisplay(histEntryData);
  }

  /*****************************************/
  /* CLOSE THE DATABASE RECOVERY HISTORY FILE */
  /*****************************************/
  printf("\n  Close recovery history file for '%s' database.\n", dbAlias);

  /* The API db2HistoryCloseScan ends the recovery history file scan and
     frees DB2 resources required for the scan. */
  db2HistoryCloseScan(db2Version810, &recoveryHistoryFileHandle, &sqlca);
  DB2_API_CHECK("database recovery history file -- close");

  /* free the allocated memory */
  rc = HistoryEntryDataFieldsFree(&histEntryData);

  return 0;
} /* DbRecoveryHistoryFileRead */

int HistoryEntryDataFieldsAlloc(struct db2HistoryData *pHistEntryData)
{
```

```
  int rc = 0;
  sqluint32 tsNb;

  strcpy(pHistEntryData->ioHistDataID, "SQLUHINF");

  pHistEntryData->oObjectPart.pioData = malloc(17 + 1);
  pHistEntryData->oObjectPart.iLength = 17 + 1;

  pHistEntryData->oEndTime.pioData = malloc(12 + 1);
  pHistEntryData->oEndTime.iLength = 12 + 1;

  pHistEntryData->oFirstLog.pioData = malloc(8 + 1);
  pHistEntryData->oFirstLog.iLength = 8 + 1;

  pHistEntryData->oLastLog.pioData = malloc(8 + 1);
  pHistEntryData->oLastLog.iLength = 8 + 1;

  pHistEntryData->oID.pioData = malloc(128 + 1);
  pHistEntryData->oID.iLength = 128 + 1;

  pHistEntryData->oTableQualifier.pioData = malloc(128 + 1);
  pHistEntryData->oTableQualifier.iLength = 128 + 1;

  pHistEntryData->oTableName.pioData = malloc(128 + 1);
  pHistEntryData->oTableName.iLength = 128 + 1;

  pHistEntryData->oLocation.pioData = malloc(128 + 1);
  pHistEntryData->oLocation.iLength = 128 + 1;

  pHistEntryData->oComment.pioData = malloc(128 + 1);
  pHistEntryData->oComment.iLength = 128 + 1;

  pHistEntryData->oCommandText.pioData = malloc(128 + 1);
  pHistEntryData->oCommandText.iLength = 128 + 1;

  pHistEntryData->poEventSQLCA =
    (struct sqlca *)malloc(sizeof(struct sqlca));

  pHistEntryData->poTablespace = (db2Char *)malloc(3 * sizeof(db2Char));
  for (tsNb = 0; tsNb < 3; tsNb = tsNb + 1)
  {
    pHistEntryData->poTablespace[tsNb].pioData = malloc(18 + 1);
    pHistEntryData->poTablespace[tsNb].iLength = 18 + 1;
  }

  pHistEntryData->iNumTablespaces = 3;

  return 0;
} /* HistoryEntryDataFieldsAlloc */

int HistoryEntryDisplay(struct db2HistoryData histEntryData)
{
  int rc = 0;
  char buf[129];
  sqluint32 tsNb;
```

```
memcpy(buf, histEntryData.oObjectPart.pioData,
            histEntryData.oObjectPart.oLength);
buf[histEntryData.oObjectPart.oLength] = '\0';
printf("    object part: %s\n", buf);

memcpy(buf, histEntryData.oEndTime.pioData,
            histEntryData.oEndTime.oLength);
buf[histEntryData.oEndTime.oLength] = '\0';
printf("    end time: %s\n", buf);

memcpy(buf, histEntryData.oFirstLog.pioData,
            histEntryData.oFirstLog.oLength);
buf[histEntryData.oFirstLog.oLength] = '\0';
printf("    first log: %s\n", buf);

memcpy(buf, histEntryData.oLastLog.pioData,
            histEntryData.oLastLog.oLength);
buf[histEntryData.oLastLog.oLength] = '\0';
printf("    last log: %s\n", buf);

memcpy(buf, histEntryData.oID.pioData, histEntryData.oID.oLength);
buf[histEntryData.oID.oLength] = '\0';
printf("    ID: %s\n", buf);

memcpy(buf, histEntryData.oTableQualifier.pioData,
            histEntryData.oTableQualifier.oLength);
buf[histEntryData.oTableQualifier.oLength] = '\0';
printf("    table qualifier: %s\n", buf);

memcpy(buf, histEntryData.oTableName.pioData,
            histEntryData.oTableName.oLength);
buf[histEntryData.oTableName.oLength] = '\0';
printf("    table name: %s\n", buf);

memcpy(buf, histEntryData.oLocation.pioData,
            histEntryData.oLocation.oLength);
buf[histEntryData.oLocation.oLength] = '\0';
printf("    location: %s\n", buf);

memcpy(buf, histEntryData.oComment.pioData,
            histEntryData.oComment.oLength);
buf[histEntryData.oComment.oLength] = '\0';
printf("    comment: %s\n", buf);

memcpy(buf, histEntryData.oCommandText.pioData,
            histEntryData.oCommandText.oLength);
buf[histEntryData.oCommandText.oLength] = '\0';
printf("    command text: %s\n", buf);
printf("    history file entry ID: %u\n", histEntryData.oEID.ioHID);
printf("    table spaces:\n");

for (tsNb = 0; tsNb < histEntryData.oNumTablespaces; tsNb = tsNb + 1)
{
  memcpy(buf, histEntryData.poTablespace[tsNb].pioData,
```

```
                histEntryData.poTablespace[tsNb].oLength);
    buf[histEntryData.poTablespace[tsNb].oLength] = '\0';
    printf("      %s\n", buf);
  }

  printf("    type of operation: %c\n", histEntryData.oOperation);
  printf("    granularity of the operation: %c\n", histEntryData.oObject);
  printf("    operation type: %c\n", histEntryData.oOptype);
  printf("    entry status: %c\n", histEntryData.oStatus);
  printf("    device type: %c\n", histEntryData.oDeviceType);
  printf("    SQLCA:\n");
  printf("      sqlcode: %ld\n", histEntryData.poEventSQLCA->sqlcode);
  memcpy(buf, histEntryData.poEventSQLCA->sqlstate, 5);
  buf[5] = '\0';
  printf("      sqlstate: %s\n", buf);
  memcpy(buf, histEntryData.poEventSQLCA->sqlerrmc,
              histEntryData.poEventSQLCA->sqlerrml);
  buf[histEntryData.poEventSQLCA->sqlerrml] = '\0';
  printf("      message: %s\n", buf);

  return 0;
} /* HistoryEntryDisplay */

int HistoryEntryDataFieldsFree(struct db2HistoryData *pHistEntryData)
{
  int rc = 0;
  sqluint32 tsNb;

  free(pHistEntryData->oObjectPart.pioData);
  free(pHistEntryData->oEndTime.pioData);
  free(pHistEntryData->oFirstLog.pioData);
  free(pHistEntryData->oLastLog.pioData);
  free(pHistEntryData->oID.pioData);
  free(pHistEntryData->oTableQualifier.pioData);
  free(pHistEntryData->oTableName.pioData);
  free(pHistEntryData->oLocation.pioData);
  free(pHistEntryData->oComment.pioData);
  free(pHistEntryData->oCommandText.pioData);
  free(pHistEntryData->poEventSQLCA);

  for (tsNb = 0; tsNb < 3; tsNb = tsNb + 1)
  {
    free(pHistEntryData->poTablespace[tsNb].pioData);
  }

  free(pHistEntryData->poTablespace);

  return 0;
} /* HistoryEntryDataFieldsFree */

int DbFirstRecoveryHistoryFileEntryUpdate(char dbAlias[],
                                          char user[],
                                          char pswd[])
{
  int rc = 0;
```

```
struct sqlca sqlca;
struct db2HistoryOpenStruct dbHistoryOpenParam;
sqluint16 recoveryHistoryFileHandle;
struct db2HistoryGetEntryStruct dbHistoryEntryGetParam;
struct db2HistoryData histEntryData;
char newLocation[DB2HISTORY_LOCATION_SZ + 1];
char newComment[DB2HISTORY_COMMENT_SZ + 1];
struct db2HistoryUpdateStruct dbHistoryUpdateParam;

printf("\n*****************************************************\n");
printf("*** UPDATE A DATABASE RECOVERY HISTORY FILE ENTRY ***\n");
printf("*****************************************************\n");
printf("\nUSE THE DB2 APIs:\n");
printf("  db2HistoryOpenScan -- Open Recovery History File Scan\n");
printf("  db2HistoryGetEntry -- Get Next Recovery History File Entry\n");
printf("  db2HistoryUpdate -- Update Recovery History File\n");
printf("  db2HistoryCloseScan -- Close Recovery History File Scan\n");
printf("TO UPDATE A DATABASE RECOVERY HISTORY FILE ENTRY.\n");

/* initialize data structures */
dbHistoryOpenParam.piDatabaseAlias = dbAlias;
dbHistoryOpenParam.piTimestamp = NULL;
dbHistoryOpenParam.piObjectName = NULL;
dbHistoryOpenParam.iCallerAction = DB2HISTORY_LIST_HISTORY;
dbHistoryEntryGetParam.pioHistData = &histEntryData;
dbHistoryEntryGetParam.iCallerAction = DB2HISTORY_GET_ALL;
rc = HistoryEntryDataFieldsAlloc(&histEntryData);
if (rc != 0)
{
  return rc;
}

/*****************************************/
/* OPEN THE DATABASE RECOVERY HISTORY FILE */
/*****************************************/
printf("\n  Open the recovery history file for '%s' database.\n", dbAlias);

/* The API db2HistoryOpenScan starts a recovery history file scan */
db2HistoryOpenScan(db2Version810, &dbHistoryOpenParam, &sqlca);
DB2_API_CHECK("database recovery history file -- open");

/* dbHistoryOpenParam.oHandle returns the handle for scan access */
recoveryHistoryFileHandle = dbHistoryOpenParam.oHandle;
dbHistoryEntryGetParam.iHandle = recoveryHistoryFileHandle;

/*****************************************************/
/* READ THE FIRST ENTRY IN THE RECOVERY HISTORY FILE */
/*****************************************************/
printf("\n  Read the first entry in the recovery history file.\n");

/* The API db2HistoryGetEntry gets the next entry from the recovery
   history file. */
db2HistoryGetEntry(db2Version810, &dbHistoryEntryGetParam, &sqlca);
DB2_API_CHECK("first recovery history file entry -- read");
printf("\n  Display the first entry.\n");
```

```
/* HistoryEntryDisplay is a support function used to display the entries
   in the recovery history file. */
rc = HistoryEntryDisplay(histEntryData);

/* update the first history file entry */
rc = DbConn(dbAlias, user, pswd);
if (rc != 0)
{
  return rc;
}

strcpy(newLocation, "this is the NEW LOCATION");
strcpy(newComment, "this is the NEW COMMENT");
printf("\n  Update the first entry in the history file:\n");
printf("    new location = '%s'\n", newLocation);
printf("    new comment = '%s'\n", newComment);
dbHistoryUpdateParam.piNewLocation = newLocation;
dbHistoryUpdateParam.piNewDeviceType = NULL;
dbHistoryUpdateParam.piNewComment = newComment;
dbHistoryUpdateParam.iEID.ioNode = histEntryData.oEID.ioNode;
dbHistoryUpdateParam.iEID.ioHID = histEntryData.oEID.ioHID;

/* The API db2HistoryUpdate can be used to update the location,
   device type, or comment in a history file entry. */

/* Call this API to update the location and comment of the first
   entry in the history file: */
db2HistoryUpdate(db2Version810, &dbHistoryUpdateParam, &sqlca);
DB2_API_CHECK("first history file entry -- update");

rc = DbDisconn(dbAlias);
if (rc != 0)
{
  return rc;
}

/*******************************************/
/* CLOSE THE DATABASE RECOVERY HISTORY FILE */
/*******************************************/
printf("\n  Close recovery history file for '%s' database.\n", dbAlias);

/* The API db2HistoryCloseScan ends the recovery history file scan and
   frees DB2 resources required for the scan. */
db2HistoryCloseScan(db2Version810, &recoveryHistoryFileHandle, &sqlca);
DB2_API_CHECK("database recovery history file -- close");


/*******************************************/
/* RE-OPEN THE DATABASE RECOVERY HISTORY FILE */
/*******************************************/
printf("\n  Open the recovery history file for '%s' database.\n", dbAlias);

/* starts a recovery history file scan */
db2HistoryOpenScan(db2Version810, &dbHistoryOpenParam, &sqlca);
```

```
  DB2_API_CHECK("database recovery history file -- open");

  recoveryHistoryFileHandle = dbHistoryOpenParam.oHandle;

  dbHistoryEntryGetParam.iHandle = recoveryHistoryFileHandle;
  printf("\n  Read the first recovery history file entry.\n");

  /*************************************************************************/
  /* READ THE FIRST ENTRY IN THE RECOVERY HISTORY FILE AFTER MODIFICATION */
  /*************************************************************************/
  db2HistoryGetEntry(db2Version810, &dbHistoryEntryGetParam, &sqlca);
  DB2_API_CHECK("first recovery history file entry -- read");

  printf("\n  Display the first entry.\n");
  rc = HistoryEntryDisplay(histEntryData);

  /*****************************************/
  /* CLOSE THE DATABASE RECOVERY HISTORY FILE */
  /*****************************************/
  printf("\n  Close the recovery history file for '%s' database.\n",
         dbAlias);

  /* ends the recovery history file scan */
  db2HistoryCloseScan(db2Version810, &recoveryHistoryFileHandle, &sqlca);
  DB2_API_CHECK("database recovery history file -- close");

  /* free the allocated memory */
  rc = HistoryEntryDataFieldsFree(&histEntryData);

  return 0;
} /* DbFirstRecoveryHistoryFileEntryUpdate */

int DbRecoveryHistoryFilePrune(char dbAlias[], char user[], char pswd[])
{
  int rc = 0;
  struct sqlca sqlca;
  struct db2PruneStruct histPruneParam;
  char timeStampPart[14 + 1];

  printf("\n*************************************\n");
  printf("*** PRUNE THE RECOVERY HISTORY FILE ***\n");
  printf("*************************************\n");
  printf("\nUSE THE DB2 API:\n");
  printf("  db2Prune -- Prune Recovery History File\n");
  printf("AND THE SQL STATEMENTS:\n");
  printf("  CONNECT\n");
  printf("  CONNECT RESET\n");
  printf("TO PRUNE THE RECOVERY HISTORY FILE.\n");

  /* Connect to the database: */
  rc = DbConn(dbAlias, user, pswd);
  if (rc != 0)
  {
    return rc;
  }
```

```
  /* Prune the recovery history file: */
  printf("\n  Prune the recovery history file for '%s' database.\n",
         dbAlias);

  /* timeStampPart is a pointer to a string specifying a time stamp or
     log sequence number. Time stamp is used here to select records for
     deletion. All entries equal to or less than the time stamp will be
     deleted. */
  histPruneParam.piString = timeStampPart;
  strcpy(timeStampPart, "2010"); /* year 2010 */

  /* The action DB2PRUNE_ACTION_HISTORY removes history file entries: */
  histPruneParam.iAction = DB2PRUNE_ACTION_HISTORY;

  /* The option DB2PRUNE_OPTION_FORCE forces the removal of the last backup: */
  histPruneParam.iOptions = DB2PRUNE_OPTION_FORCE;

  /* db2Prune can be called to delete entries from the recovery history file
     or log files from the active log path. Here we call it to delete
     entries from the recovery history file.
     You must have SYSADM, SYSCTRL, SYSMAINT, or DBADM authority to prune
     the recovery history file. */
  db2Prune(db2Version810, &histPruneParam, &sqlca);
  DB2_API_CHECK("recovery history file -- prune");

  /* Disconnect from the database: */
  rc = DbDisconn(dbAlias);
  if (rc != 0)
  {
    return rc;
  }

  return 0;
} /* DbRecoveryHistoryFilePrune */
```

附录 F. Tivoli Storage Manager

调用 BACKUP DATABASE 或 RESTORE DATABASE 命令时，可以指定想要使用 Tivoli Storage Manager（TSM）产品来管理数据库或表空间备份或复原操作。TSM 客户机 API 的最小必需级别是版本 4.2.0，但在 64 位 Solaris 系统上除外，它需要 TSM 客户机 API 版本 4.2.1。

配置 Tivoli Storage Manager 客户机

可能需要执行下列步骤来配置 TSM 环境，数据库管理器才能使用 TSM 选项:

1. 必须安装和配置功能 TSM 客户机和服务器。另外，必须安装 TSM 客户机 API。
2. 设置 TSM 客户机 API 使用的环境变量:

   DSMI_DIR 标识 API 可信的代理进程文件（dsmtca）所在的用户定义目录路径。

   DSMI_CONFIG
   标识 dsm.opt 文件（它包含 TSM 用户选项）的用户定义目录路径。与另外两个变量不同，此变量应包含全限定路径和文件名。

   DSMI_LOG 标识将在其中创建错误日志（dsierror.log）的用户定义目录路径。

   注: 在多分区数据库环境中，这些设置必须在 sqllib/userprofile 目录中指定。
3. 如果对这些环境变量进行了任何更改，并且数据库管理器正在运行，则应该:
   - 使用 **db2stop** 命令停止数据库管理器。
   - 使用 **db2start** 命令启动数据库管理器。
4. 根据服务器的配置，Tivoli 客户机可能需要密码才能与 TSM 服务器交互。如果将 TSM 环境配置为使用 PASSWORDACCESS=generate，则 Tivoli 客户机需要建立它的密码。

   将可执行文件 dsmapipw 安装在实例所有者的 sqllib/adsm 目录中。这个可执行文件允许您建立和重设 TSM 密码。

   要执行 dsmapipw 命令，必须作为本地管理员或 “root” 用户来登录。当执行此命令时，将提示您输入下列信息:

- *旧密码*，它是 TSM 服务器认可的该 TSM 节点的当前密码。第一次执行此命令时，此密码是 TSM 管理员在 TSM 服务器上注册节点时所提供的密码。
- *新密码*，它是 TSM 节点的新密码，存储在 TSM 服务器上。（将提示您输入新密码两次，以检查是否有输入错误。）

注：调用 BACKUP DATABASE 或 RESTORE DATABASE 命令的用户不需要知道此密码。只需要在为初始连接建立了密码时，以及密码已在 TSM 服务器上复位后，运行 dsmapipw 命令。

使用 Tivoli Storage Manager 的注意事项

要使用 TSM 内的特定功能部件，可能需要给出使用该功能部件的对象的全限定路径名。（记住在 Windows 操作系统上，将使用 \ 来代替 /。）以下是不同对象的全限定路径名:

- 完整数据库备份对象: /<database>/NODEnnnn/FULL_BACKUP.timestamp.seq_no
- 增量数据库备份对象: /<database>/NODEnnnn/DB_INCR_BACKUP.timestamp.seq_no
- 增量 delta 数据库备份对象: /<database>/NODEnnnn/DB_DELTA_BACKUP.timestamp.seq_no
- 完整表空间备份对象为: /<database>/NODEnnnn/TSP_BACKUP.timestamp.seq_no
- 增量表空间备份对象: /<database>/NODEnnnn/TSP_INCR_BACKUP.timestamp.seq_no
- 增量 delta 表空间备份对象: /<database>/NODEnnnn/TSP_DELTA_BACKUP.timestamp.seq_no

其中 <database> 是数据库别名，NODEnnnn 是节点号。大写名称必须按显示的样子输入。

- 对于多个备份使用同一个数据库别名的情况，时间戳记和序号就成为全限定名中可区别的部分。需要查询 TSM，以确定要使用的备份版本。
- 个别备份映象放在 TSM 管理的文件空间。个别备份映象只能通过 TSM API 或通过使用这些 API 的 **db2adutl** 来处理。
- 若 Tivoli 客户机在服务器配置文件中的 COMMTIMEOUT 参数指定的时间内没有应答，则 TSM 服务器执行的会话将超时。有 3 个因素会导致超时问题:
  - 在 TSM 服务器上 COMMTIMEOUT 参数设置得太低。例如，如果在复原操作期间创建大的 DMS 表空间，就可能发生超时。此参数的建议值为 6000 秒。
  - DB2 备份或复原缓冲区可能太大。
  - 联机备份操作期间的数据库活动可能太频繁。

- 使用多个会话来增加吞吐量（仅当 TSM 服务器上有足够的硬件可用时）。

相关概念:

- 第 39 页的『管理日志文件』
- 『Tivoli Space Manager 分层存储管理器（AIX）』（《*Data Links Manager 快速入门*》）

相关参考:

- 第 185 页的『db2adutl — 使用 TSM 归档映象』

附录 G. 数据库恢复的用户出口

您可以开发一个*用户出口程序*以自动执行日志文件的归档和检索。在调用用户出口程序进行日志文件的归档或检索前，应确保 *userexit* 数据库配置参数已设置为 YES。它还使数据库可进行前滚恢复。

调用了用户出口程序后，数据库管理器将控制发送给可执行文件 db2uext2。数据库管理器将参数发送到 db2uext2，完成后程序再将返回码发送回数据库管理器。由于数据库管理器只能处理有限的一组返回条件，所以用户出口程序应能够处理错误条件（参见第 299 页的『错误处理』）。而且由于在一个数据库管理器实例中只能调用一个用户出口程序，因此必须对可能要它执行的每个操作都有一个程序节。

包括下列主题:

样本用户出口程序

对所有受支持的平台都提供了样本用户出口程序。您可以修改这些程序以适应您的特定要求。样本程序中有很好的注释信息，它有助于您进行更有效的使用。

您应该了解用户出口程序必须从活动的日志路径*复制*日志文件到归档日志路径中。不要从活动的日志路径除去日志文件。（这可能会在数据库恢复期间导致问题。）DB2® 从活动日志路径中除去恢复操作不再需要的归档日志文件。

以下是 DB2 附带的样本用户出口程序的描述。

- **基于 UNIX® 的系统**

  “DB2 基于 UNIX 的系统版”中的用户出口样本程序在 sqllib/samples/c 子目录中。虽然提供的样本是用 C 语言编码的，您的用户出口程序仍可以不同的编程语言编写。

  您的用户出口程序必须是可执行文件，名称为 db2uext2。

  这里有四个可用于基于 UNIX 的系统的样本用户出口程序:

  – db2uext2.ctsm

    此样本使用 Tivoli® Storage Manager 来归档和检索数据库日志文件。

  – db2uext2.ctape

该样本使用磁带媒体来归档和检索数据库日志文件。

– db2uext2.cdisk

该样本使用操作系统的 COPY 命令以及磁盘媒体来归档和检索数据库日志文件。

– db2uxt2.cxbsa

此样本使用 X/Open group 发布的 XBSA Draft 0.8。可将它用于归档和检索数据库日志文件。该样本只在 AIX 上受支持。

- **Windows® 操作系统**

可在 sqllib\samples\c 子目录中找到用于 DB2 Windows 操作系统版的用户出口样本程序。虽然提供的样本是用 C 语言编码的，您的用户出口程序仍可以不同的编程语言编写。

您的用户出口程序必须是可执行文件，名称为 db2uext2。

对于 Windows 操作系统，有两个样本用户出口程序:

– db2uext2.ctsm

该样本使用 Tivoli Storage Manager 来归档和检索数据库日志文件。

– db2uext2.cdisk

该样本使用操作系统的 COPY 命令以及磁盘媒体来归档和检索数据库日志文件。

调用格式

数据库管理器调用用户出口程序时，将一组参数（数据类型为 CHAR）发送到程序。调用格式取决于您的操作系统:

```
db2uext2 -OS<os> -RL<db2rel> -RQ<request> -DB<dbname>
-NN<nodenum> -LP<logpath> -LN<logname> -AP<tsmpasswd>
-SP<startpage> -LS<logsize>
```

os 指定实例是在哪个平台是运行的。有效的值有: AIX®、Solaris、HP-UX、SCO、Linux 和 NT。

db2rel 指定 DB2 发行级别。例如，SQL07020。

request 指定请求类型。有效的值是: ARCHIVE 和 RETRIEVE。

dbname 指定数据库名。

nodenum 指定本地节点号，例如 5。

logpath 指定日志文件的全限定路径。该路径必须包含尾部路径分隔符。例如 /u/database/log/path/ 或 d:\logpath\。

logname 指定要归档或检索的日志文件的名称，例如 S0000123.LOG。

tsmpasswd 指定 TSM 密码。（如果先前已指定了数据库配置参数 *tsm_password* 的值，该值将发送到用户出口程序。）

startpage 指定日志范围开始的那个设备的 4 KB 偏移页的数量。

logsize 指定日志范围的大小，以 4 KB 页为单位。只有将原始设备用于记录时，此参数才有效。

错误处理

用户出口程序应设计为提供特定并有意义的返回码，以使数据库管理器可正确地解释它们。因为用户出口程序由基本的操作系统命令处理器调用，操作系统本身可以返回错误代码。并且由于未重新映射这些错误代码，使用操作系统消息将有助于实用程序获得关于这些错误代码的信息。

表 8 显示了可由用户出口程序返回的代码，并描述数据库管理器是如何解释这些代码的。若返回码未在该表中列出，则将其值当作 32 处理。

表 8. 用户出口程序返回码. 只适用于归档和检索操作。

返回码	说明
0	成功。
4	遇到临时资源错误。[a]
8	需要操作员介入。[a]
12	硬件错误。[b]
16	用户出口程序或该程序使用的软件功能出错。[b]
20	传送给用户出口程序一个或多个参数出错。验证用户出口程序是否正确地处理指定的参数。[b]
24	找不到用户出口程序。[b]
28	输入／输出（I/O）故障或操作系统导致的错误。[b]
32	用户出口程序由用户终止。[b]
255	由用户出口程序无法为可执行文件装入库文件而导致的错误。[c]

表 8. 用户出口程序返回码 (续). 只适用于归档和检索操作。

返回码	说明
[a] 对于归档与检索请求，返回码 4 或 8 导致在五分钟后重试。如果用户出口程序继续对向同一日志文件发出的检索请求返回 4 或 8，则 DB2 将继续再试，直到成功为止。（这适用于前滚操作或对 **db2ReadLog** API（由复制实用程序使用）的调用。） [b] 用户出口请求将暂挂 5 分钟。在此期间，将忽略所有的请求，包括导致出错状态的请求。在这个时间为 5 分钟的暂挂后，再处理下一个请求。如果处理此请求时没有错误，将继续处理新的用户出口请求，且 DB2 会重新发出失败过的或先前被暂挂的归档请求。若在重试期间生成一个大于 8 的返回码，则会将请求再暂停 5 分钟。此 5 分钟的暂挂将继续，直到该校正该问题或直到停止并重新启动该数据库。一旦已从该数据库断开了与所有应用程序的连接，DB2 会对先前可能未成功归档的任何日志文件发出归档请求。如果用户出口程序对日志文件归档失败，磁盘可能会填满了日志文件，从而可能使性能降低。一旦磁盘变满，数据库管理器将不再接受应用程序对更新数据库的请求。如果调用了用户出口程序来检索日志文件，则前滚恢复会暂挂但不停止，除非指定了 ROLLFORWARD STOP 选项。如果未指定 STOP 选项，可以更正问题并继续恢复。 [c] 如果用户出口程序返回错误代码 255，可能是程序无法装入可执行文件所需的库文件。要进行验证，可手工调用用户出口程序。会显示更多信息。 **注**：在归档和检索操作期间，除返回 0 和 4 之外，对所有其它返回码都发出警告消息。该警告消息包含来自用户出口程序的返回码和提供给用户出口程序的输入参数的副本。	

附录 H. 供应商产品的备份与复原 API

供应商产品的备份与复原 API

DB2 提供了一些接口，可由第三方媒体管理产品用于存储和检索执行备份与复原操作的数据。此功能设计为将备份与复原操作的数据目标扩大到软盘、磁盘、磁带和 Tivoli Storage Manager（已将它作为 DB2 的一个标准部件来支持）。

这些第三方媒体管理产品在此附录的余下部分都将称为“供应商产品”。

DB2 定义了一组函数原型，它提供了一个用于备份和复原的通用数据接口，可由多个供应商使用。这些函数由供应商在基于 UNIX 的系统上的共享库中提供，或在 Windows 操作系统上的 DDL 上提供。当 DB2 调用这些函数时，装入由调用备份或复原例程所指定的共享库或 DLL，并调用由供应商提供的函数以执行所需的任务。

附录分为以下四个部分:

- DB2 与供应商产品的交互作用的操作概述。
- DB2 供应商 API 的详细描述。
- 使用供应商产品来调用备份与复原的详细信息。
- API 调用中使用的关于数据结构的信息。

操作概述

定义了 5 个函数，以在 DB2 和供应商产品之间提供接口:

- sqluvint — 初始化与链接到设备
- sqluvget — 从设备读取数据
- sqluvput — 向设备写数据
- sqluvend — 解链设备
- sqluvdel — 删除落实的会话

DB2 将调用这些函数，它们应当由供应商产品在基于 UNIX 的系统上的共享库中提供，或在 Windows 操作系统上的 DDL 中提供。

注: 将运行共享库或 DLL 代码，作为数据库引擎代码的一部分。因此，必须重新输入并彻底调试。函数如果有错就会影响到数据库的数据完整性。

DB2 在特定的备份或复原操作期间调用函数的顺序取决于:

- 将利用的会话数。
- 是备份操作还是复原操作。
- 对备份或复原操作指定的 PROMPTING 方式。
- 在其中存储数据的设备的特征。
- 在操作期间可能遇到的错误。

会话数

DB2 支持使用一个或多个数据流或会话来备份和复原数据库对象。使用 3 个会话的备份或复原需要有 3 个物理或逻辑设备可用。供应商设备支持正在使用中时，由供应商的函数负责管理对每个物理或逻辑设备的接口。DB2 只发送或接收流向或来自供应商提供的函数的数据缓冲区。

要使用的会话数由调用备份或复原数据库函数的应用程序作为参数来指定。此值在由 **sqluvint** 使用的 INIT-INPUT 结构中提供。

DB2 将继续初始化会话直到达到指定的会话数，或接收到来自 **sqluvint** 调用的 SQLUV_MAX_LINK_GRANT 警告返回码。为了警告 DB2 已达到了它可支持的最大会话数，供应商产品将需要代码来跟踪活动的会话数。未成功警告 DB2 可能会使 DB2 初始化会话请求失败，从而导致所有会话的终止以及整个备份或复原操作的失败。

如果是备份操作，DB2 会在每个会话开始时写一个媒体头记录。该记录包含了 DB2 用于在复原操作期间标识会话的信息。通过将序列号附加到备份映象的名称后，DB2 可唯一地标识每个会话。会话数从 1 开始，对第一个会话使用 1，然后在每次通过 **sqluvint** 调用启动另一会话以进行备份或复原操作时加 1。

备份操作成功完成后，DB2 会对它关闭的最后一个会话写一个媒体跟踪文件。跟踪文件中包含的信息告诉 DB2 使用了多少个会话来执行备份操作。在复原操作期间，这些信息将用于确保已复原所有会话或数据流。

没有错误、警告或提示时的操作

对于备份操作，DB2 会对每个会话发出以下顺序的调用。

```
sqluvint, action = SQLUV_WRITE
```

n 后加 1

```
sqluvput
```

加 1

```
sqluvend, action = SQLUV_COMMIT
```

DB2 发出 **sqluvend** 调用（操作 SQLUV_COMMIT）时，它希望供应商产品会适当地保存输出数据。对 DB2 发出返回码 SQLUV_OK 表示成功。

对 **sqluvint** 调用使用的 DB2-INFO 结构包含标识备份所需的信息。提供了序列号。可能会选择供应商产品来保存此信息。DB2 将在复原期间用它来标识将要复原的备份。

对于复原操作，要对每个会话发出的调用序列是:

```
sqluvint, action = SQLUV_READ
```

n 后加 1

```
sqluvget
```

加 1

```
sqluvend, action = SQLUV_COMMIT
```

对 **sqluvint** 调用使用的 DB2-INFO 结构中的信息将包含标识备份所需的信息。未提供序列号。DB2 希望所有的备份对象（在备份期间落实的会话输出）都将返回。返回的第一个备份对象是使用序列号 1 生成的对象，而其它所有对象都不按特定的次序复原。DB2 将检查媒体末端对象以确保处理了所有对象。

注: 并非所有供应商产品都将保留一个备份对象的名称的记录。只有在备份到磁带上或其它容量有限的媒体上时，才最可能发生这种情况。在复原会话的初始化期间，可利用标识信息来登台需要的备份对象，以使它们在有需要时可用；这在使用 juke box 或机械人系统来存储备份时最有用。DB2 总是会检查媒体头（每个会话的输出中的第一个记录），以确保复原了正确的数据。

PROMPTING 方式

启动了备份或复原操作后，有两种可能的提示方式:

- WITHOUT PROMPTING 或 NOINTERRUPT，此时供应商产品没有机会向用户写消息，或用户也没有机会作出响应。
- PROMPTING 或 INTERRUPT，此时可接收并响应来自供应商产品的消息。

对于 PROMPTING 方式，备份与复原定义了三种可能的用户响应:

- 继续

  对设备的读或写数据操作将继续。
- 设备终止

  设备将不再接收数据，会话终止。
- 终止

  终止整个备份或复原操作。

本节中的稍后部分讨论了 PROMPTING 和 WITHOUT PROMPTING 方式的使用。

设备特征

为了使供应商设备支持 API，定义了两种常用的设备类型:

- 容量有限的设备，需要用户操作才能更换媒体；例如磁带机、软盘或 CDROM 驱动器。
- 容量很大的设备，在正常操作中不需要用户处理媒体；例如 juke box 或一种智能机械人媒体处理设备。

容量有限的设备可能需要在备份或复原操作期间提示用户装入附加媒体。通常，DB2 对于备份或复原操作中媒体装入的次序并不敏感。它还为向用户发送供应商媒体处理消息提供了设施。这种提示需要启动备份或复原操作时 PROMPTING 为 on。媒体处理消息文本在返回码结构的描述字段中指定。

如果 PROMPTING 为 on，而 DB2 从 **sqluvput**（写）或 **sqluvget**（读）调用接收到 SQLUV_ENDOFMEDIA 或 SQLUV_ENDOFMEDIA_NO_DATA 返回码，DB2 将:

- 如果调用是 **sqluvput**，则标记发送给该会话的最后那个缓冲区，以重新发送。稍后它将会放到会话中。
- 使用 **sqluvend** (action = SQLUV_COMMIT) 调用该会话。如果成功（返回码 SQLUV_OK），DB2 将:
  - 从指示媒体结束情况的返回码结构，发送供应商媒体处理消息给用户。
  - 提示用户发出继续、设备终止或终止响应。
- 如果响应为*继续*，DB2 将使用 **sqluvint** 调用来初始化另一会话，如果成功，则从会话读数据或向会话写数据。要在写数据时唯一地标识会话，DB2 会对序列号进行递增。通过 **sqluvint** 使用时，DB2-INFO 结构中的序列号可用，并且在媒体头记录中（该记录是发送给会话的第一个数据记录）。

  DB2 不会启动多于启动备份或复原操作时请求的会话，或由供应商产品用 **sqluvint** 调用时的 SQLUV_MAX_LINK_GRANT 警告来指示的会话。
- 如果响应为*设备终止*，DB2 不会尝试初始化另一会话，而活动的会话数由此减 1。DB2 不允许通过设备终止响应来终止所有的会话，至少在备份或复原操作完成前要有一个会话保持活动。
- 如果响应是*终止*，DB2 将终止备份或复原操作。关于 DB2 究竟为何不终止某些会话的更多信息，参见第 305 页的『如果向 DB2 返回了出错状态』。

因为备份或复原性能通常取决于正使用的设备数，所以维护并行性很重要。对于备份操作，鼓励用户响应为继续，除非他们知道余下的活动会话将保留仍将写出的数据。对于复原操作，也鼓励用户响应为继续，直到已处理完所有媒体。

如果备份或复原方式为 WITHOUT PROMPTING，而 DB2 接收到来自某个会话的 SQLUV_ENDOFMEDIA 或 SQLUV_ENDOFMEDIA_NO_DATA 返回码，它将终止会话，且不尝试打开另一会话。如果在备份或复原操作完成前，所有会话都对 DB2 返回“媒体结束”消息，操作将失败。因此，对容量有限的设备使用 WITHOUT PROMPTING 时应特别小心，但在对容量很大的设备操作时，该方法可行。

供应商产品有可能会对 DB2 隐藏媒体安装和转换操作，因此设备看起来象是有无穷大的容量。有一些容量非常大的设备就是以这种方式操作的。在这些情况中非常关键的一点是：将已备份的所有数据返回给 DB2 时所采用的次序与数据在复原操作时的次序相同。未成功地做到这一点会导致数据丢失，但 DB2 由于已无法检测到数据丢失，而假设复原操作成功。

DB2 将数据写到供应商产品时，假设每个缓冲区都包含在一个且只有一个媒体上（例如，一盒磁带）。供应商产品会将这些缓冲区分割为跨多个媒体，而不让 DB2 知道。在这种情况中，复原操作期间处理媒体的次序就变得很关键，因为供应商产品将负责把重新构造的缓冲区从多个媒体返回给 DB2。未成功地做到这一点将导致复原操作失败。

如果向 DB2 返回了出错状态

执行备份或复原操作时，DB2 希望所有会话都成功完成，否则整个备份或复原操作都将失败。会话通过 **sqluvend** 调用（操作 = SQLUV_COMMIT）上的 SQLUV_OK 返回码，向 DB2 发出成功完成的信号。

如果遇到了不可恢复错误，DB2 将终止该会话。它们可能是 DB2 错误，或是由供应商产品返回给 DB2 的错误。因为所有会话必须成功落实才能有一个完整的备份或复原操作，一个会话失败会导致 DB2 终止与该操作相关的其它会话。

如果供应商产品使用一个不可恢复的返回码响应来自 DB2 的调用，供应商产品可通过使用 RETURN-CODE 结构的描述字段中放置的消息文本，自选提供附加信息。将对用户出现此消息文本，同时还有 DB2 信息，从而可采取校正操作。

在有的备份方案中，已成功落实了一个会话，但与备份操作相关的另一会话却遇到了不可恢复的错误。因为必须在所有会话都成功完成后才能将备份操作视为成功，所以 DB2 必须删除已落实的会话中的输出数据：DB2 发出 **sqluvdel** 调用以请求删除该对象。不将此调用视为 I/O 会话，它负责初始化并终止删除备份对象时可能需要的任何连接。

DB2-INFO 结构将不包含序列号，**sqluvdel** 将删除与 DB2-INFO 结构中余下的参数匹配的所有备份对象。

警告状态

如果设备未就绪，或发生了其它某些可校正的状态，DB2 可能会从供应商产品接收到警告返回码。对读或写操作都如此。

在 **sqluvput** 和 **sqluvget** 调用上，供应商可将返回码设置为 SQLUV_WARNING，并通过使用 RETURN-CODE 结构的描述字段中放置的消息文本，自选提供附加信息。将对用户出现此消息文本，从而可采取校正操作。用户可用以下三种方法之一来响应，继续、设备终止或终止:

- 如果响应为*继续*，DB2 将在备份操作期间，尝试使用 **sqluvput** 来重写缓冲区。在复原操作期间，DB2 将发出 **sqluvget** 调用来读下一个缓冲区。
- 如果响应为*设备终止*或*终止*，DB2 将以在发生了不可恢复的错误时采取的同样方式（例如，它将终止活动的会话并删除已落实的会话），终止整个备份或复原操作。

操作提示与技巧

本部分提供了构建供应商产品时的某些提示与技巧。

历史文件

历史文件可以用作数据库恢复操作的辅助资料。它与每个数据库相关，并且随着每次备份或复原操作而自动更新。可通过以下设施，对此文件中的信息进行查看、更新或修剪:

- 控制中心
- 命令行处理器（CLP）
  - LIST HISTORY 命令
  - UPDATE HISTORY FILE 命令
  - PRUNE HISTORY 命令
- API
  - db2HistoryOpenScan
  - db2HistoryGetEntry
  - db2HistoryCloseScan
  - db2HistoryUpdate
  - db2Prune

有关该文件的布局的信息，参见 db2HistData。

备份操作完成后，将把一个或多个记录写到该文件中。如果将备份操作的输出引导到供应商设备，则历史记录中的 DEVICE 字段会包含一个 0，且 LOCATION 字段会包含:

- 调用备份操作时指定的供应商文件名。
- 共享库的名称（如果未指定供应商文件名）。

关于指定此选项的更多信息，参见『使用供应商产品调用备份或复原操作』。

可使用“控制中心”、CLP 或 API 来更新 LOCATION 字段。如果已使用了容量有限的设备（例如可更换的媒体）来保留备份映象，就可以更新备份信息的位置，从而使媒体物理移动到另一（可能是离站）存储位置。如果是这种情况，就可以在需要进行恢复操作时，使用历史文件帮助您查找备份映象。

使用供应商产品调用备份或复原操作

可在调用 DB2 备份实用程序或 DB2 复原实用程序时，从以下各项指定供应商产品:

- 控制中心
- 命令行处理器（CLP）
- 应用程序编程接口（API）。

控制中心

“控制中心”是随 DB2 附带的，用于进行数据库管理的图形用户界面。

要指定	用于备份或复原操作的“控制中心”输入变量
使用供应商设备和库名	是 *Use Library*。指定库名（在基于 UNIX 的系统上）或 DLL 名（在 Windows 操作系统上）。
会话数	是 *Sessions*。
供应商选项	不受支持。
供应商文件名	不受支持。
传送缓冲区大小	（对于备份操作）是 *Size of each Buffer*，而（对于复原操作）不适用。

命令行处理器（CLP）

命令行处理器（CLP）可用来调用 DB2 BACKUP DATABASE 或 RESTORE DATABASE 命令。

要指定	命令行处理器参数	
	用于备份	用于复原
使用供应商设备和库名	*library-name*	*shared-library*

要指定	命令行处理器参数	
	用于备份	用于复原
会话数	*num-sessions*	*num-sessions*
供应商选项	不受支持	不受支持
供应商文件名	不受支持	不受支持
传送缓冲区大小	*buffer-size*	*buffer-size*

应用程序编程接口（API）

有两个 API 函数调用支持备份与复原操作：**db2Backup** 用于备份，而 **db2Restore** 用于复原。

要指定	（用于 **db2Backup** 和 **db2Restore**）的 **API** 参数是
使用供应商设备和库名	如下所示：在 *sqlu_media_list* 结构中，指定媒体类型 SQLU_OTHER_MEDIA，然后在 *sqlu_vendor* 结构的 *shr_lib* 中指定共享库或 DLL。
会话数	如下所示：在结构 *sqlu_media_list* 中指定 *sessions*。
供应商选项	*PVendorOptions*
供应商文件名	如下所示：在结构 *sqlu_media_list* 中，指定媒体类型 SQLU_OTHER_MEDIA，然后在结构 *sqlu_vendor* 的 *filename* 中指定文件名。
传送缓冲区大小	*BufferSize*

相关参考:

- 第 309 页的『sqluvint — 初始化和链接至设备』
- 第 312 页的『sqluvget — 从设备读取数据』
- 第 314 页的『sqluvput — 将数据写入设备』
- 第 316 页的『sqluvend — 解除设备的链接并释放其资源』
- 第 318 页的『sqluvdel — 删除落实的会话』
- 第 320 页的『DB2-INFO』
- 第 322 页的『VENDOR-INFO』
- 第 323 页的『INIT-INPUT』
- 第 325 页的『INIT-OUTPUT』

- 第 325 页的『DATA』
- 第 326 页的『RETURN-CODE』

sqluvint — 初始化和链接至设备

调用此函数可提供关于初始化的信息，以及在 DB2 和供应商设备之间建立逻辑链路的信息。

授权:

以下其中一项:

- *sysadm*
- *dbadm*

必需的连接:

数据库

API 包含文件:

sql.h

C API 语法:

```
/* File: sqluvend.h */
/* API: Initialize and Link to Device */
/* ... */
int sqluvint (
  struct Init_input   *,
  struct Init_output  *,
  struct Return_code  *);
/* ... */
```

API 参数:

Init_input
输入。包含由 DB2 提供的、用于与供应商设备建立逻辑链路的信息的结构。

Init_output
输出。包含由供应商设备返回的输出的结构。

Return_code
输出。包含要发送给 DB2 的返回码和简要文本说明的结构。

使用注意事项:

对于每个媒体 I/O 会话，DB2 都将调用此函数以获得设备句柄。如果出于任何原因，供应商函数在初始化期间遇到了错误，它都将通过返回码指出该错误。如果返回码表示出现了错误，DB2 可能会通过调用 **sqluvend** 函数，选择终止该操作。关于可能的返回码的详细信息，以及 DB2 对每个返回码的反应都包含在返回码表中（参见表 9）。

INIT-INPUT 结构中包含的元素可由供应商产品用于确定是否可进行备份或复原:

- size_HI_order and size_LOW_order

  这是备份的估计大小。它们可用于确定供应商设备是否可处理大小为这么多的备份映象。它们可用于估计保留备份所需的可更换媒体的数量。如果预计会出现问题，在第一次 **sqluvint** 调用失败时，该返回码可能会有用。

- req_sessions

  用户请求的会话数，可与估计大小和提示级别一起使用，以确定是否可进行备份或复原操作。

- prompt_lvl

  提示级别告诉供应商是否可以对某些操作进行提示，例如更换可更换的媒体（例如，将另一盒磁带放入磁带机中）。它可能会建议由于无法提示用户，所以该操作不能进行。

  如果提示级别是 WITHOUT PROMPTING，且可更换的媒体的数量大于所请求的会话数，则 DB2 将无法成功完成该操作。

DB2 为正在写的备份或将通过 DB2-INFO 结构中的字段读取的复原命名。在操作 = SQLUV_READ 的情况下，供应商产品必须检查已命名的对象是否存在。如果找不到，返回码应设置为 SQLUV_OBJ_NOT_FOUND，从而使 DB2 采取相应的措施。

初始化成功完成后，DB2 会通过发出其它数据传输函数而继续，但是可能会通过 **sqluvend** 调用在任何时候终止会话。

返回码:

表 9. 有效的 sqluvint 返回码及相应的 DB2 操作

头文件中的文字	描述	可能的下一个调用	其它注释
SQLUV_OK	操作成功。	sqluvput 和 sqluvget（参见注释）	如果 action = SQLUV_WRITE，下一个调用将是（对 BACKUP 数据）调用 sqluvput。如果 action = SQLUV_READ，则验证返回 SQLUV_OK 之前，命名对象是否存在；下一个调用将是对 RESTORE 数据调用 sqluvget。
SQLUV_LINK_EXIST	先前激活的会话。	没有进一步的调用	会话初始化失败。释放为此会话分配的内存，然后终止。因为从未建立会话，所以将不会接收到 sqluvend 调用。

表 9. 有效的 sqluvint 返回码及相应的 DB2 操作 (续)

头文件中的文字	描述	可能的下一个调用	其它注释
SQLUV_COMM_ ERROR	与设备通信错误。	没有进一步的调用	会话初始化失败。释放为此会话分配的内存，然后终止。因为从未建立会话，所以将不会接收到 sqluvend 调用。
SQLUV_INV_VERSION	DB2 和供应商的产品不兼容。	没有进一步的调用	会话初始化失败。释放为此会话分配的内存，然后终止。因为从未建立会话，所以将不会接收到 sqluvend 调用。
SQLUV_INV_ACTION	请求了一个无效的操作。还可用来表示参数的组合产生了不可能的操作。	没有进一步的调用	会话初始化失败。释放为此会话分配的内存，然后终止。因为从未建立会话，所以将不会接收到 sqluvend 调用。
SQLUV_NO_DEV_ AVAIL	此时没有设备可用。	没有进一步的调用	会话初始化失败。释放为此会话分配的内存，然后终止。因为从未建立会话，所以将不会接收到 sqluvend 调用。
SQLUV_OBJ_NOT_ FOUND	找不到指定的对象。应在对 sqluvint 调用的操作为 'R'（读），且根据 DB2-INFO 结构中指定的标准找不到请求的对象时，使用该返回码。	没有进一步的调用	会话初始化失败。释放为此会话分配的内存，然后终止。因为从未建立会话，所以将不会接收到 sqluvend 调用。
SQLUV_OBJS_FOUND	有多于 1 个对象与指定的准则匹配。当对 sqluvint 调用的操作为 'R'（读取）且有多个对象与 DB2-INFO 结构中的准则相匹配时产生此调用。	没有进一步的调用	会话初始化失败。释放为此会话分配的内存，然后终止。因为从未建立会话，所以将不会接收到 sqluvend 调用。
SQLUV_INV_USERID	指定了无效的用户标识。	没有进一步的调用	会话初始化失败。释放为此会话分配的内存，然后终止。因为从未建立会话，所以将不会接收到 sqluvend 调用。
SQLUV_INV_ PASSWORD	提供了无效的密码。	没有进一步的调用	会话初始化失败。释放为此会话分配的内存，然后终止。因为从未建立会话，所以将不会接收到 sqluvend 调用。
SQLUV_INV_OPTIONS	在供应商选项字段中遇到了无效的选项。	没有进一步的调用	会话初始化失败。释放为此会话分配的内存，然后终止。因为从未建立会话，所以将不会接收到 sqluvend 调用。
SQLUV_INIT_FAILED	初始化失败，会话将终止。	没有进一步的调用	会话初始化失败。释放为此会话分配的内存，然后终止。因为从未建立会话，所以将不会接收到 sqluvend 调用。
SQLUV_DEV_ERROR	设备错误。	没有进一步的调用	会话初始化失败。释放为此会话分配的内存，然后终止。因为从未建立会话，所以将不会接收到 sqluvend 调用。
SQLUV_MAX_LINK_ GRANT	建立了最大链接数。	sqluvput 和 sqluvget（参见注释）	将它视为 DB2 提供的警告。此警告告诉 DB2 因为已达到了它可以支持的最大会话数（注：这可能是由于设备的可用性），所以不要再打开与供应商产品的其它会话。如果 action = SQLUV_WRITE (BACKUP)，则下一个调用将是 sqluvput。如果 action = SQLUV_READ，则验证返回 SQLUV_MAX_LINK_GRANT 之前，命名对象是否存在；下一个调用将是对 RESTORE 数据调用 sqluvget。

表 9. 有效的 sqluvint 返回码及相应的 DB2 操作 (续)

头文件中的文字	描述	可能的下一个调用	其它注释
SQLUV_IO_ERROR	I/O 错误。	没有进一步的调用	会话初始化失败。释放为此会话分配的内存，然后终止。因为从未建立会话，所以将不会接收到 sqluvend 调用。
SQLUV_NOT_ENOUGH_SPACE	没有足够的空间用于存储整个备份映象，所提供的估计大小是以字节计的 64 位值。	没有进一步的调用	会话初始化失败。释放为此会话分配的内存，然后终止。因为从未建立会话，所以将不会接收到 sqluvend 调用。

sqluvget — 从设备读取数据

在初始化后，可调用此函数以从设备读取数据。

授权:

以下其中一项:

- *sysadm*
- *dbadm*

必需的连接:

数据库

API 包含文件:

sqluvend.h

C API 语法:

```
/* File: sqluvend.h */
/* API: Reading Data from Device */
/* ... */
int sqluvget (
  void * pVendorCB,
  struct Data         *,
  struct Return_code  *);
/* ... */

typedef struct Data
}
   sqlint32  obj_num;
   sqlint32  buff_size;
   sqlint32  actual_buff_size;
   void      *dataptr;
   void      *reserve;
{ Data;
```

API 参数:

pVendorCB

输入。指向为 DATA 结构（包括数据缓冲区）和 Return_code 分配的空间的指针。

Data 输入／输出。指向 *data* 结构的指针。

Return_code

输出。来自 API 调用的返回码。

obj_num

指定应检索哪个备份对象。

buff_size

指定要使用的缓冲区大小。

actual_buff_size

指定读或写的实际字节。此值应设置为输出，以指示实际读取的数据字节数。

dataptr

指向数据缓冲区的指针。

reserve

保留以备将来使用。

使用注意事项:

此函数由复原实用程序使用。

返回码:

表 10. 有效的 sqluvget 返回码及相应的 DB2 操作

头文件中的文字	描述	可能的下一个调用	其它注释
SQLUV_OK	操作成功。	sqluvget	DB2 处理数据
SQLUV_COMM_ERROR	与设备通信错误。	sqluvend, action = SQLU_ABORT[a]	会话将终止。
SQLUV_INV_ACTION	请求了一个无效的操作。	sqluvend, action = SQLU_ABORT[a]	会话将终止。
SQLUV_INV_DEV_HANDLE	无效的设备句柄。	sqluvend, action = SQLU_ABORT[a]	会话将终止。
SQLUV_INV_BUFF_SIZE	指定了无效的缓冲区大小。	sqluvend, action = SQLU_ABORT[a]	会话将终止。
SQLUV_DEV_ERROR	设备错误。	sqluvend, action = SQLU_ABORT[a]	会话将终止。

表 10. 有效的 sqluvget 返回码及相应的 DB2 操作 (续)

头文件中的文字	描述	可能的下一个调用	其它注释
SQLUV_WARNING	警告。不应使用它来表示 DB2 的媒体结束；应使用 SQLUV_ENDOFMEDIA 或 SQLUV_ENDOFMEDIA_NO_DATA 来表示 DB2 的媒体结束。但可使用此返回码来表示设备未就绪的情况。	sqluvget 或 sqluvend，action = SQLU_ABORT	
SQLUV_LINK_NOT_EXIST	当前不存在链接。	sqluvend，action = SQLU_ABORT[a]	会话将终止。
SQLUV_MORE_DATA	操作成功；有更多的数据可用。	sqluvget	
SQLUV_ENDOFMEDIA_NO_DATA	媒体结束，读取了 0 字节（例如磁带已结束）。	sqluvend	
SQLUV_ENDOFMEDIA	媒体结束 > 读取了 0 字节（例如磁带已结束）。	sqluvend	DB2 处理数据，然后处理媒体结束情况。
SQLUV_IO_ERROR	I/O 错误。	sqluvend，action = SQLU_ABORT[a]	会话将终止。

下一个调用:

[a] 如果下一个调用是 sqluvend，action = SQLU_ABORT，此会话和所有其它活动的会话将会终止。

sqluvput — 将数据写入设备

在初始化后，可调用此函数以向设备写数据。

授权:

以下其中一项:

- *sysadm*
- *dbadm*

必需的连接:

数据库

API 包含文件:

sqluvend.h

C API 语法:

```
/* File: sqluvend.h */
/* API: Writing Data to Device */
/* ... */
int sqluvput (
  void * pVendorCB,
  struct Data  *,
  struct Return_code  *);
/* ... */

typedef struct Data
}
   sqlint32  obj_num;
   sqlint32  buff_size;
   sqlint32  actual_buff_size;
   void      *dataptr;
   void      *reserve;
{ Data;
```

API 参数:

pVendorCB
输入。指向为 DATA 结构（包括数据缓冲区）和 Return_code 分配的空间的指针。

Data 输出。用将要写出的数据填充的数据缓冲区。

Return_code
输出。来自 API 调用的返回码。

obj_num
指定应检索哪个备份对象。

buff_size
指定要使用的缓冲区大小。

actual_buff_size
指定读或写的实际字节。此值应设置为指示实际读取的数据字节数。

dataptr
指向数据缓冲区的指针。

reserve
保留以备将来使用。

使用注意事项:

此函数由备份实用程序使用。

返回码:

sqluvput — 将数据写入设备

表 11. 有效的 sqluvput 返回码及相应的 DB2 操作

头文件中的文字	描述	可能的下一个调用	其它注释
SQLUV_OK	操作成功。	如果完成的话（例如，DB2 已没有更多的数据），为 sqluvput 或 sqluvend	通知其它进程操作成功。
SQLUV_COMM_ERROR	与设备通信错误。	sqluvend, action = SQLU_ABORT[a]	会话将终止。
SQLUV_INV_ACTION	请求了一个无效的操作。	sqluvend, action = SQLU_ABORT[a]	会话将终止。
SQLUV_INV_DEV_HANDLE	无效的设备句柄。	sqluvend, action = SQLU_ABORT[a]	会话将终止。
SQLUV_INV_BUFF_SIZE	指定了无效的缓冲区大小。	sqluvend, action = SQLU_ABORT[a]	会话将终止。
SQLUV_ENDOFMEDIA	已达到了媒体的末端，例如磁带的末端。	sqluvend	
SQLUV_DATA_RESEND	设备请求再次发送缓冲区。	sqluvput	DB2 将重新发送上一个缓冲区。只应进行一次该操作。
SQLUV_DEV_ERROR	设备错误。	sqluvend, action = SQLU_ABORT[a]	会话将终止。
SQLUV_WARNING	警告。不应使用它来表示 DB2 的媒体结束；应使用 SQLUV_ENDOFMEDIA 来表示。但可使用此返回码来表示设备未就绪的情况。	sqluvput	
SQLUV_LINK_NOT_EXIST	当前不存在链接。	sqluvend, action = SQLU_ABORT[a]	会话将终止。
SQLUV_IO_ERROR	I/O 错误。	sqluvend, action = SQLU_ABORT[a]	会话将终止。
下一个调用: [a] 如果下一个调用是 sqluvend, action = SQLU_ABORT，此会话和所有其它活动的会话将会终止。使用 sqluvint、sqluvdel 和 sqluvend 调用序列删除了落实的会话。			

sqluvend — 解除设备的链接并释放其资源

结束或解除设备的链接，并释放它的所有相关资源。供应商必须在返回到 DB2 前，释放未使用的资源（例如已分配的空间和文件句柄）。

授权:

以下其中一项:

- *sysadm*
- *dbadm*

必需的连接:

数据库

API 包含文件:

sql.h

C API 语法:

```
/* File: sqluvend.h */
/* API: Unlink the Device and Release its Resources */
/* ... */
int sqluvend (
  sqlint32 action,
  void * pVendorCB,
  struct Init_output  *,
  struct Return_code  *);
/* ... */
```

API 参数:

操作 输入。用于落实或异常终止会话:

- SQLUV_COMMIT （ 0 = 要落实）
- SQLUV_ABORT （ 1 = 要异常终止）

pVendorCB

输入。指向 Init_output 结构的指针。

Init_output

输出。取消分配的 Init_output 的空间。如果操作是“要落实”，数据已落实到稳定存储，以进行备份。如果操作是“要异常终止”，则清除数据以进行备份。

Return code

输出。来自 API 调用的返回码。

使用注意事项:

对已打开的每个会话调用此函数。有两个可能的操作代码:

- Commit

  将数据输出到此会话，或从此会话读取数据已完成。

  对于写（备份）会话，如果供应商返回到 DB2 时带有返回码 SQLUV_OK，DB2 会假设供应商产品已正确地保存了输出数据，并且如果在稍后的 **sqluvint** 调用中引用输出数据时，它是可访问的。

  对于读（复原）操作，如果供应商返回到 DB2 时带有返回码 SQLUV_OK，则因为可能会再次使用数据，所以不应删除数据。

如果供应商返回 SQLUV_COMMIT_FAILED，DB2 会假设整个备份或复原操作可能有问题。所有活动的会话都会由 **sqluvend** 调用（并且 action = SQLUV_ABORT）终止。对于备份操作，落实的会话接收调用序列 **sqluvint**、**sqluvdel** 和 **sqluvend**。

- Abort

  DB2 遇到了问题，不会再对会话读或写数据。

  对于写（备份）会话，供应商应删除部分输出数据集，并在删除了部分输出时使用 SQLUV_OK 返回码。DB2 会假设整个备份有问题。**sqluvend** 调用会使用操作 = SQLUV_ABORT 终止所有活动会话，落实的会话接收调用序列 **sqluvint**、**sqluvdel** 和 **sqluvend**。

  对于读（复原）会话，供应商不应删除数据（因为可能会再次需要它），但应进行清除，并返回给 DB2 SQLUV_OK 返回码。DB2 通过 **sqluvend** 调用（且 action = SQLUV_ABORT）终止所有的复原会话。如果供应商将 SQLUV_ABORT_FAILED 返回给 DB2，将不会向调用程序通知此错误，原因是 DB2 将返回第一个致命的故障，并忽略后继的故障。此时，如果 DB2 已调用了 **sqluvend**（并且 action = SQLUV_ABORT），一定会发生初始致命错误。

返回码:

表 12. 有效的 sqluvend 返回码及相应的 DB2 操作

头文件中的文字	描述	可能的下一个调用	其它注释
SQLUV_OK	操作成功。	没有进一步的调用	释放为此会话分配的所有内存，然后终止。
SQLUV_COMMIT_FAILED	落实请求失败。	没有进一步的调用	释放为此会话分配的所有内存，然后终止。
SQLUV_ABORT_FAILED	异常终止请求失败。	没有进一步的调用	

sqluvdel — 删除落实的会话

删除落实的会话。

授权:

以下其中一项:

- *sysadm*
- *dbadm*

必需的连接:

数据库

API 包含文件:

sqluvend.h

C API 语法:

```
/* File: sqluvend.h */
/* API: Delete Committed Session */
/* ... */
int sqluvdel (
  struct Init_input   *,
  struct Init_output  *,
  struct Return_code  *);
/* ... */
```

API 参数:

Init_input
输入。为 Init_input 和 Return_code 分配的空间。

Return_code
输出。来自 API 调用的返回码。将删除 Init_input 结构所指向的对象。

使用注意事项:

如果打开了多个会话且落实了某些会话，但其中一个会话失败了，将调用此函数以删除落实的会话。未指定序列号；**sqluvdel** 负责查找在特定的备份操作期间创建的所有对象并删除它们。INIT-INPUT 结构中的信息用于标识将删除的输出数据。对 **sqluvdel** 的调用负责建立从供应商设备删除备份对象所需的任何连接或会话。如果此调用的返回码是 SQLUV_DELETE_FAILED，DB2 不会通知调用程序，原因是 DB2 将返回第一个致命的故障并忽略后继的故障。此时，如果 DB2 已调用了 **sqluvdel**，一定会发生初始致命错误。

返回码:

表 13. 有效的 sqluvdel 返回码及相应的 DB2 操作

头文件中的文字	描述	可能的下一个调用	其它注释
SQLUV_OK	操作成功。	没有进一步的调用	
SQLUV_DELETE_FAILED	删除请求失败。	没有进一步的调用	

DB2-INFO

此结构包含向供应商标识 DB2 的信息。

表 14. DB2-INFO 结构中的字段. 所有字段都是以零结束的字符串。

字段名称	数据类型	描述
DB2_id	char	DB2 产品的标识符。它指向的字符串的最大长度是 8 个字符。
version	char	DB2 产品的当前版本。它指向的字符串的最大长度是 8 个字符。
release	char	DB2 产品的当前发行版。如果该字段不太重要，可设置为 NULL。它指向的字符串的最大长度是 8 个字符。
level	char	DB2 产品的当前级别。如果该字段不太重要，可设置为 NULL。它指向的字符串的最大长度是 8 个字符。
action	char	指定要进行的操作。它指向的字符串的最大长度是 1 个字符。
filename	char	用于标识备份映象的文件名。如果是 NULL，*server_id*、*db2instance*、*dbname* 和 *timestamp* 将唯一地标识备份映象。它指向的字符串的最大长度是 255 个字符。
server_id	char	标识数据库所驻留的服务器的唯一名称。它指向的字符串的最大长度是 8 个字符。
db2instance	char	db2 实例标识。这是调用命令的用户标识。它指向的字符串的最大长度是 8 个字符。
type	char	指定正在进行的备份的类型或正在执行的复原的类型。以下是可能的值: 当操作是 SQLUV_WRITE 时: 0 — 完整数据库备份 3 — 表空间级的备份 当操作是 SQLUV_READ 时: 0 — 完整复原 3 — 联机表空间复原 4 — 表空间复原 5 — 历史文件复原
dbname	char	要进行备份或复原的数据库的名称。它指向的字符串的最大长度是 8 个字符。
alias	char	要进行备份或复原的数据库的别名。它指向的字符串的最大长度是 8 个字符。
timestamp	char	用于标识备份映象的时间戳记。它指向的字符串的最大长度是 26 个字符。

表 14. DB2-INFO 结构中的字段 (续). 所有字段都是以零结束的字符串。

字段名称	数据类型	描述
sequence	char	指定备份映象的文件扩展名。对于写操作，第一个会话的值是 1，而每次通过 sqluvint 调用启动另一会话时，该值就会加 1。对于读取操作，该值总为零。它指向的字符串的最大长度是 3 个字符。
obj_list	struct sqlu_gen_list	保留以供将来使用。
max_bytes_per_txn	sqlint32	以字节为单位，向供应商指定由用户指定的传送缓冲区大小。
image_filename	char	保留以备将来使用。
reserve	void	保留以备将来使用。
nodename	char	生成备份的节点的名称。
password	char	生成备份的节点的密码。
owner	char	备份创始人的标识。
mcNameP	char	管理类。
nodeNum	SQL_PDB_NODE_TYPE	节点号。供应商接口支持大于 255 的号码。

filename 或 *server_id*、*db2instance*、*type*、*dbname* 和 *timestamp* 唯一地标识备份映象。由 *sequence* 指定的序列号标识文件扩展名。要复原某个备份映象时，必须相同的值才能检索该备份映象。取决于供应商产品，如果使用 *filename*，则其它参数可以设置为 NULL，反之亦然。

DB2-INFO

语言语法:

C 结构

```
/* File: sqluvend.h */
/* ... */
typedef struct DB2_info
{
  char                  *DB2_id;
  char                  *version;
  char                  *release;
  char                  *level;
  char                  *action;
  char                  *filename;
  char                  *server_id;
  char                  *db2instance;
  char                  *type;
  char                  *dbname;
  char                  *alias;
  char                  *timestamp;
  char                  *sequence;
  struct sqlu_gen_list  *obj_list;
  long                  max_bytes_per_txn;
  char                  *image_filename;
  void                  *reserve;
  char                  *nodename;
  char                  *password;
  char                  *owner;
  char                  *mcNameP;
  SQL_PDB_NODE_TYPE     nodeNum;
} DB2_info;
/* ... */
```

VENDOR-INFO

此结构中的信息标识设备的供应商和版本。

表 15. VENDOR-INFO 结构中的字段. 所有字段都是以零结束的字符串。

字段名称	数据类型	描述
vendor_id	char	供应商的标识符。它指向的字符串的最大长度是 64 个字符。
version	char	供应商产品的当前版本。它指向的字符串的最大长度是 8 个字符。
release	char	供应商产品的当前发行版。如果该字段不太重要，可设置为 NULL。它指向的字符串的最大长度是 8 个字符。
level	char	供应商产品的当前级别。如果该字段不太重要，可设置为 NULL。它指向的字符串的最大长度是 8 个字符。

表 15. VENDOR-INFO 结构中的字段 (续). 所有字段都是以零结束的字符串。

字段名称	数据类型	描述
server_id	char	标识数据库所驻留的服务器的唯一名称。它指向的字符串的最大长度是 8 个字符。
max_bytes_per_txn	sqlint32	受支持的最大传送缓冲区大小。由供应商以字节指定。只有在供应商初始化函数的返回码为 SQLUV_BUFF_SIZE 时才使用它，表示指定了一个无效的缓冲区大小。
num_objects_in_backup	sqlint32	用于进行完整备份的会话数。用于确定在复原操作期间，何时处理了所有备份映象。
reserve	void	保留以备将来使用。

语言语法:

C 结构

```
typedef struct Vendor_info
{
  char       *vendor_id;
  char       *version;
  char       *release;
  char       *level;
  char       *server_id;
  sqlint32   max_bytes_per_txn;
  sqlint32   num_objects_in_backup;
  void       *reserve;
} Vendor_info;
```

INIT-INPUT

此结构中包含的信息由 DB2 提供，用于设置并建立与供应商设备的逻辑链路。

表 16. INIT-INPUT 结构中的字段. 所有字段都是以零结束的字符串。

字段名称	数据类型	描述
DB2_session	struct DB2_info	以 DB2 的角度，对会话进行的描述。
size_options	unsigned short	选项字段的长度。使用 DB2 备份或复原函数时，此字段中的数据从 *VendorOptionsSize* 参数直接发送。
size_HI_order	sqluint32	以字节数估计的 DB 大小的高阶 32 位，总共 64 位。
size_LOW_order	sqluint32	以字节数估计的 DB 大小的低阶 32 位，总共 64 位。

表 16. INIT-INPUT 结构中的字段 (续). 所有字段都是以零结束的字符串。

字段名称	数据类型	描述
options	void	调用备份或复原函数时从应用程序发送此信息。此数据结构必须是平面，也就是说，不支持间接级别。字节反转未完成，而此数据的代码页也未检查。使用 DB2 备份或复原函数时，此字段中的数据从 *pVendorOptions* 参数直接发送。
reserve	void	保留以备将来使用。
prompt_lvl	char	调用备份或复原操作时，用户请求的提示级别。它指向的字符串的最大长度是 1 个字符。
num_sessions	unsigned short	调用备份或复原操作时，用户请求的会话数。

语言语法:

C 结构

```
typedef struct Init_input
{
  struct DB2_info  *DB2_session;
  unsigned short   size_options;
  sqluint32        size_HI_order;
  sqluint32        size_LOW_order;
  void             *options;
  void             *reserve;
  char             *prompt_lvl;
  unsigned short   num_sessions;
} Init_input;
```

INIT-OUTPUT

此结构包含由供应商设备返回的输出。

表 17. INIT-OUTPUT 结构中的字段

字段名称	数据类型	描述
vendor_session	struct Vendor_info	包含向 DB2 标识供应商的信息。
pVendorCB	void	供应商控制块。
reserve	void	保留以备将来使用。

语言语法:

C 结构

```
typedef struct Init_output
{
  struct Vendor_info  *vendor_session;
  void                *pVendorCB;
  void                *reserve;
} Init_output;
```

DATA

此结构包含在 DB2 和供应商设备间传输的数据。

表 18. DATA 结构中的字段

字段名称	数据类型	描述
obj_num	sqlint32	DB2 在备份操作期间指定的序列号。
buff_size	sqlint32	缓冲区的大小。

DATA

表 18. DATA 结构中的字段 (续)

字段名称	数据类型	描述
actual_buf_size	sqlint32	发送或接收的实际字节数。一定不能超出 *buff_size*。
dataptr	void	指向数据缓冲区的指针。DB2 为缓冲区分配空间。
reserve	void	保留以备将来使用。

语言语法:

C 结构

```
typedef struct Data
{
  sqlint32  obj_num;
  sqlint32  buff_size;
  sqlint32  actual_buff_size;
  void      *dataptr;
  void      *reserve;
} Data;
```

RETURN-CODE

此结构包含返回给 DB2 的错误的返回码和简短说明。

表 19. RETURN-CODE 结构中的字段

字段名称	数据类型	描述
return_code[a]	sqlint32	供应商函数中的返回码。
description	char	返回码的简短描述。
reserve	void	保留以备将来使用。
[a] 这是供应商特定的返回码，与由不同 DB2 API 返回的值不同。参见供应商产品接受的返回码的个别 API 描述。		

语言语法:

C 结构

```
typedef struct Return_code
{
  sqlint32  return_code,
  char      description[30],
  void      *reserve,
} Return_code;
```

附录 I. “DB2 通用数据库”技术信息

“DB2 通用数据库”技术信息概述

可以下列格式获取“DB2 通用数据库”技术信息:

- 书籍（PDF 和硬拷贝格式）
- 主题树（HTML 格式）
- DB2 工具的帮助（HTML 格式）
- 样本程序（HTML 格式）
- 命令行帮助
- 教程

本节是有关所提供技术信息以及可如何访问这些信息的概述。

DB2 文档的修订包

IBM 可能会阶段性地提供文档修订包。文档修订包使您可以在新信息可供使用时更新从 *DB2 HTML 文档 CD* 中安装的信息。

注: 如果您安装了文档修订包，则您的 HTML 文档将包含比 DB2 的印刷或联机 PDF 手册更新的信息。

DB2 技术信息类别

DB2 技术信息是按下列标题分类的:

- 核心 DB2 信息
- 管理信息
- 应用程序开发信息
- 商务智能信息
- DB2 Connect 信息
- 入门信息
- 教程信息
- 可选组件信息
- 发行说明

对于 DB2 资料库中的每本书，下表描述了订购硬拷贝、打印或查看 PDF 或者找出该书的 HTML 目录所需的信息。DB2 资料库中每本书的完整描述可从 IBM 出版物中心（IBM Publications Center）获得，网址为 www.ibm.com/shop/publications/order。

HTML 文档 CD 的安装目录对于各个信息类别来说是不同的:

htmlcdpath/doc/htmlcd/*%L*/*category*

其中:

- *htmlcdpath* 是安装了 HTML CD 的目录。
- *%L* 是语言标识符。例如，en_US。
- *category* 是类别标识符。例如，core 表示核心 DB2 信息。

在下表中的 PDF 文件名列中，文件名第六个位置的字符指示书籍的语言版本。例如，文件名 db2d1e80 标识英文版本的《*管理指南: 计划*》，而文件名 db2d1g80 标识该书的德语版本。下列字母用在文件名的第六个字符处以指示语言版本:

语言	标识符
阿拉伯语	w
巴西葡萄牙语	b
保加利亚语	u
克罗地亚语	9
捷克语	x
丹麦语	d
荷兰语	q
英语	e
芬兰语	y
法语	f
德语	g
希腊语	a
匈牙利语	h
意大利语	i
日语	j
韩国语	k
挪威语	n
波兰语	p
葡萄牙语	v
罗马尼亚语	8
俄语	r
简体中文	c
斯洛伐克语	7
斯洛文尼亚语	l
西班牙语	z

瑞典语	s
繁体中文	t
土耳其语	m

无书号指示该书只有联机版本而没有印刷版本。

核心 DB2 信息

此类别中的信息包括对所有 DB2 用户都很重要的 DB2 主题。不管您是程序员、数据库管理员或您将使用 DB2 Connect、DB2 仓库管理器或其它 DB2 产品，都将会发现此类别中的信息很有用。

此类别的安装目录为 `doc/htmlcd/%L/core`。

表 20. 核心 DB2 信息

书名	书号	PDF 文件名
IBM DB2 Universal Database Command Reference	SC09-4828	db2n0x80
《IBM DB2 通用数据库词汇表》	无书号	db2t0c80
《IBM DB2 通用数据库主索引》	S152-0192	db2w0c80
《IBM DB2 通用数据库消息参考第 1 卷》	G152-0177	db2m1c80
《IBM DB2 通用数据库消息参考第 2 卷》	G152-0178	db2m2c80
《IBM DB2 通用数据库新增内容》	S152-0176	db2q0c80

管理信息

此类别中的信息包括有效地设计、实现和维护 DB2 数据库、数据仓库和联合系统所需的那些主题。

此类别的安装目录为 `doc/htmlcd/%L/admin`。

表 21. 管理信息

书名	书号	PDF 文件名
《IBM DB2 通用数据库管理指南: 计划》	S152-0167	db2d1c80

表 21. 管理信息 (续)

书名	书号	PDF 文件名
《IBM DB2 通用数据库管理指南: 实现》	S152-0165	db2d2c80
《IBM DB2 通用数据库管理指南: 性能》	S152-0166	db2d3c80
IBM DB2 Universal Database Administrative API Reference	SC09-4824	db2b0x80
IBM DB2 Universal Database Data Movement Utilities Guide and Reference	SC09-4830	db2dmx80
《IBM DB2 通用数据库数据恢复和高可用性指南与参考大全》	S152-0181	db2hac80
《IBM DB2 通用数据库数据仓库中心管理指南》	S152-0188	db2ddc80
IBM DB2 Universal Database Federated Systems Guide	GC27-1224	db2fpx80
《IBM DB2 通用数据库管理和开发 GUI 工具指南》	S152-0180	db2atc80
IBM DB2 Universal Database Replication Guide and Reference	SC27-1121	db2e0x80
《IBM DB2 安装和管理卫星环境》	G152-0272	db2dsc80
IBM DB2 Universal Database SQL Reference, Volume 1	SC09-4844	db2s1x80
IBM DB2 Universal Database SQL Reference, Volume 2	SC09-4845	db2s2x80
IBM DB2 Universal Database System Monitor Guide and Reference	SC09-4847	db2f0x80

应用程序开发信息

此类别中的信息对于应用程序开发者或使用 DB2 的程序员特别有用。将可找到有关受支持的语言和编译器的信息，以及使用各种受支持的编程接口（如嵌入式 SQL、ODBC、JDBC、SQLj 和 CLI）访问 DB2 所需的文档。如果您联机查看 HTML 格式的此信息，则还可以访问一组 HTML 格式的 DB2 样本程序。

此类别的安装目录为 doc/htmlcd/%L/ad。

表 22. 应用程序开发信息

书名	书号	PDF 文件名
《IBM DB2 通用数据库应用程序开发指南: 构建和运行应用程序》	S152-0168	db2axc80
IBM DB2 Universal Database Application Development Guide: Programming Client Applications	SC09-4826	db2a1x80
IBM DB2 Universal Database Application Development Guide: Programming Server Applications	SC09-4827	db2a2x80
IBM DB2 Universal Database Call Level Interface Guide and Reference, Volume 1	SC09-4849	db2l1x80
IBM DB2 Universal Database Call Level Interface Guide and Reference, Volume 2	SC09-4850	db2l2x80
IBM DB2 Universal Database Data Warehouse Center Application Integration Guide	SC27-1124	db2adx80
IBM DB2 XML Extender Administration and Programming	SC27-1234	db2sxx80

商务智能信息

此类别中的信息描述如何使用将增强 “DB2 通用数据库” 的数据入库功能和分析功能的组件。

此类别的安装目录为 doc/htmlcd/%L/wareh。

表 23. 商务智能信息

书名	书号	PDF 文件名
IBM DB2 Warehouse Manager Information Catalog Center Administration Guide	SC27-1125	db2dix80
《IBM DB2 仓库管理器安装指南》	G152-0187	db2idc80

DB2 Connect 信息

此类别中的信息描述如何使用 “DB2 Connect 企业版” 或 “DB2 Connect 个人版” 来存取主机或 iSeries 数据。

此类别的安装目录为 doc/htmlcd/%L/conn。

表 24. DB2 Connect 信息

书名	书号	PDF 文件名
APPC, CPI-C, and SNA Sense Codes	无书号	db2apx80
IBM Connectivity Supplement	无书号	db2h1x80
《IBM DB2 Connect 快速入门, DB2 Connect 企业版》	G152-0271	db2c6c80
《IBM DB2 Connect 快速入门, DB2 Connect 个人版》	G152-0171	db2c1c80
《IBM DB2 Connect 用户指南》	S152-0172	db2c0c80

入门信息

安装和配置服务器、客户机以及其它 DB2 产品时，此类别中的信息非常有用。

此类别的安装目录为 doc/htmlcd/%L/start。

表 25. 入门信息

书名	书号	PDF 文件名
《IBM DB2 通用数据库快速入门, DB2 客户机版》	G152-0170	db2itc80
《IBM DB2 通用数据库快速入门, DB2 服务器版》	G152-0173	db2isc80
《IBM DB2 通用数据库快速入门, DB2 个人版》	G152-0175	db2i1c80
《IBM DB2 通用数据库安装与配置补遗》	G152-0174	db2iyc80
《IBM DB2 通用数据库快速入门, DB2 Data Links Manager 版》	G152-0169	db2z6c80

教程信息

教程信息介绍 DB2 功能部件并指导如何执行各种任务。

此类别的安装目录为 doc/htmlcd/%L/tutr。

表 26. 教程信息

书名	书号	PDF 文件名
《商务智能教程: 数据仓库简介》	无书号	db2tuc80
《商务智能教程: 数据入库扩展课程》	无书号	db2tac80
Development Center Tutorial for Video Online using Microsoft Visual Basic	无书号	db2tdx80
Information Catalog Center Tutorial	无书号	db2aix80
Video Central for e-business Tutorial	无书号	db2twx80
《Visual Explain 教程》	无书号	db2tvx80

可选组件信息

此类别中的信息描述如何使用可选 DB2 组件。

此类别的安装目录为 doc/htmlcd/%L/opt。

表 27. 可选组件信息

书名	书号	PDF 文件名
IBM DB2 Life Sciences Data Connect Planning, Installation, and Configuration Guide	GC27-1235	db2lsx80
IBM DB2 Spatial Extender User's Guide and Reference	SC27-1226	db2sbx80
IBM DB2 Database Data Links Manager Administration Guide and Reference	SC27-1221	db2z0x80
IBM DB2 Universal Database Net Search Extender Administration and Programming Guide **注**: 此文档的 HTML 不是从 HTML 文档 CD 安装的。	SH12-6740	N/A

发行说明

发行说明提供了特定于产品发行版和修订包级别的附加信息。它们还提供了并入到每个发行版和修订包中的文档更新的总结。

表 28. 发行说明

书名	书号	PDF 文件名
《DB2 发行说明》	参见“注”。	参见“注”。
《DB2 安装说明》	仅在产品 CD-ROM 上提供。	仅在产品 CD-ROM 上提供。

注: 发行说明的 HTML 版本可从“信息中心”或产品 CD-ROM 上获取。要在基于 UNIX 的平台上查看 ASCII 文件，参见 `Release.Notes` 文件。此文件位于 `DB2DIR/Readme/%L` 目录中，其中 *%L* 表示语言环境名称，`DB2DIR` 表示:

- `/usr/opt/db2_08_01`（在 AIX 上）
- `/opt/IBM/db2/V8.1`（在所有其它 UNIX 操作系统上）

相关任务:

- 第 336 页的『从 PDF 文件打印 DB2 书籍』
- 第 337 页的『订购打印的 DB2 书籍』
- 第 338 页的『访问联机帮助』
- 第 341 页的『通过从管理工具访问“DB2 信息中心”来查找产品信息』
- 第 342 页的『直接从 DB2 HTML 文档 CD 联机查看技术文档』

从 PDF 文件打印 DB2 书籍

可从 *DB2 PDF 文档* CD 上的 PDF 文件打印 DB2 书籍。通过使用 Adobe Acrobat Reader，可打印整本书或特定范围的那些页。

先决条件:

确保具有 Adobe Acrobat Reader。它可从 Adobe Web 站点获得，网址为 www.adobe.com。

过程:

要从 PDF 打印 DB2 书籍:

1. 插入 *DB2 PDF 文档* CD。在 UNIX 操作系统上，安装 DB2 PDF 文档 CD。有关如何在 UNIX 操作系统上安装 CD 的详细信息，请参考《*快速入门*》一书。

2. 启动 Adobe Acrobat Reader。
3. 从下列位置之一打开 PDF 文件:
   - 在 Windows 操作系统上:

     x:\doc*language* 目录，其中 *x* 表示 CD-ROM 盘符，*language* 表示两个字符的地区代码（它表示您所用的语言），例如，EN 表示英语。
   - 在 UNIX 操作系统上:

     CD-ROM 上的 */cdrom*/doc/*%L* 目录，其中 */cdrom* 表示 CD-ROM 的安装点而 *%L* 表示期望的语言环境的名称。

相关任务:

- 第 337 页的『订购打印的 DB2 书籍』
- 第 341 页的『通过从管理工具访问 “DB2 信息中心” 来查找产品信息』
- 第 342 页的『直接从 DB2 HTML 文档 CD 联机查看技术文档』

相关参考:

- 第 329 页的『 “DB2 通用数据库” 技术信息概述』

订购打印的 DB2 书籍

过程:

要订购打印的书籍:

- 与 IBM 授权经销商或市场营销代表联系。要查找您当地的 IBM 代表，查看 IBM 全球联系人目录（IBM Worldwide Directory of Contacts），网址为 www.ibm.com/planetwide。
- 在美国可致电 1-800-879-2755，在加拿大则可致电 1-800-IBM-4YOU。
- 访问 IBM 出版物中心（IBM Publications Center），网址为 www.ibm.com/shop/publications/order。

还可通过从您的 IBM 分销商订购您的 DB2 产品的文档包来获得印刷的 DB2 手册。文档包是 DB2 库中的手册的一个子集，它们被选择来帮助您使用您购买的 DB2 产品进行初步的操作。文档包中的手册与 *DB2 PDF 文档 CD* 中以 PDF 格式提供的手册相同，并包含与 *DB2 HTML 文档 CD* 中提供的文档相同的内容。

相关任务:

- 第 336 页的『从 PDF 文件打印 DB2 书籍』
- 第 339 页的『通过从浏览器访问 “DB2 信息中心” 来查找主题』
- 第 342 页的『直接从 DB2 HTML 文档 CD 联机查看技术文档』

相关参考:

- 第 329 页的『 “DB2 通用数据库”技术信息概述 』

访问联机帮助

所有 DB2 组件附带提供的联机帮助有三种类型:

- 窗口和笔记本帮助
- 命令行帮助
- SQL 语句帮助

窗口和笔记本帮助说明可在窗口或笔记本中执行的任务并描述各控件。此帮助有两种类型:

- 可从**帮助**按钮访问的帮助
- 弹出信息

帮助按钮让您可以访问概述和先决条件信息。弹出信息描述窗口或笔记本中的各控件。窗口和笔记本帮助可从具有用户界面的 DB2 中心和组件获得。

命令行帮助包括“命令”帮助和“消息”帮助。“命令”帮助说明命令行处理器中命令的语法。“消息”帮助描述产生错误消息的原因并描述为解决错误而应采取的任何操作。

SQL 语句帮助包括 SQL 帮助和 SQLSTATE 帮助。DB2 返回可作为 SQL 语句结果的条件的 SQLSTATE 值。SQLSTATE 帮助说明 SQL 语句（SQL 语句和类代码）的语法。

注: SQL 帮助对于 UNIX 操作系统不可用。

过程:

要访问联机帮助:

- 对于窗口和笔记本帮助，单击**帮助**或单击该控件，然后单击 **F1**。如果选择了**工具设置**笔记本**常规**页上的**自动显示弹出信息**复选框，则还可以通过将鼠标光标置于特定控件上来查看该控件的弹出信息。
- 对于命令行帮助，打开命令行处理器并输入:
  - 对于“命令”帮助:

    ```
    ? command
    ```

    其中 *command* 表示一个关键字或整条命令。

例如，`? catalog` 显示所有 CATALOG 命令的帮助，而 `? catalog database` 显示 CATALOG DATABASE 命令的帮助。

- 对于“消息”帮助:

  `? XXXnnnnn`

  其中 *XXXnnnnn* 表示有效消息标识符。

  例如，`? SQL30081` 将显示有关 SQL30081 消息的帮助。

- 对于 SQL 语句帮助，打开命令行处理器并输入:

  `? sqlstate` 或 `? class code`

  其中，*sqlstate* 表示有效的 5 位 SQL 状态，*class code* 表示该 SQL 状态的前 2 位。

  例如，`? 08003` 显示 08003 SQL 状态的帮助，而 `? 08` 显示 08 类代码的帮助。

相关任务:

- 第 339 页的『通过从浏览器访问“DB2 信息中心”来查找主题』
- 第 342 页的『直接从 DB2 HTML 文档 CD 联机查看技术文档』

通过从浏览器访问“DB2 信息中心”来查找主题

“DB2 信息中心”可从浏览器访问，从而使您能够访问为充分利用“DB2 通用数据库”和 DB2 Connect 所需的信息。“DB2 信息中心”还记录主要的 DB2 功能部件和组件，包括复制、数据入库、元数据和 DB2 extender。

从浏览器访问的“DB2 信息中心”包括下列主要元素:

导航树 导航树位于浏览器窗口左边的框架中。该树可展开和折叠以显示和隐藏主题、词汇表和“DB2 信息中心”中的主索引。

导航工具栏

导航工具栏位于浏览器窗口的右上边框架中。导航工具栏包含一些使您能够执行下列操作的按钮：搜索“DB2 信息中心”、隐藏导航树以及查找导航树中当前显示的主题。

内容框架

内容框架位于浏览器窗口的右下边框架中。当单击导航树中的链接、单击搜索结果或访问另一主题或主索引的链接时，内容框架会显示“DB2 信息中心”的主题。

先决条件:

要从浏览器访问“DB2 信息中心”，必须使用下列浏览器之一:

- Microsoft Explorer，版本 5 或更高版本
- Netscape Navigator，版本 6.1 或更高版本

限制:

“DB2 信息中心”只包含您选择从 *DB2 HTML 文档 CD* 安装的那些主题集。如果当您尝试访问指向某个主题的链接时 Web 浏览器返回找不到文件错误，则您必须安装 *DB2 HTML 文档 CD* 中的一个或多个附加的主题集。

过程:

要通过使用关键字进行搜索来查找主题:

1. 在导航工具栏中，单击**搜索**。
2. 在“搜索”窗口最上面的文本输入字段中，输入一个或多个与您感兴趣的领域相关的词条，并单击**搜索**。一个按准确度排列的主题列表将显示在**结果**字段中。每一单项旁的数字等级提供了匹配程度的指示（较大的数字表示较高的匹配程度）。

   输入较多的项会提高查询的精度，同时还会减少从查询返回的主题数目。
3. 在**结果**字段中，单击想要阅读的主题的标题。该主题将会显示在内容框架中。

要查找导航树中的主题:

1. 在导航树中，单击与您感兴趣的区域相关的主题类别的书籍图标。一个子类别列表将显示在该图标下面。
2. 继续单击书籍图标，直到找到包含您感兴趣的主题的类别为止。链接至主题的类别在您将光标移到类别标题上时将类别标题显示为带下划线的链接。导航树使用页图标来标识主题。
3. 单击主题链接。该主题会显示在内容框架中。

要查找主索引中的主题或项:

1. 在导航树中，单击“索引”类别。该类别展开，并在导航树中显示按字母顺序排列的链接列表。
2. 在导航树中，单击相应于与感兴趣主题相关的项的第一个字符的链接。具有该首字符的项列表将会显示在内容框架中。具有多个索引条目的项将由一个书籍图标标识。
3. 单击与您感兴趣的项相对应的书籍图标。一个子项和主题列表将显示在您单击的项下面。主题是由页图标标识的，其标题带有下划线。
4. 单击符合需要的主题的标题。该主题会显示在内容框架中。

相关概念:

- 第 347 页的『易使用性』
- 第 349 页的『从浏览器访问的 DB2 信息中心』

相关任务:

- 第 341 页的『通过从管理工具访问 “DB2 信息中心” 来查找产品信息』
- 第 343 页的『更新安装在机器上的 HTML 文档』
- 第 345 页的『对于使用 Netscape 4.x 搜索 DB2 文档进行故障诊断』
- 第 346 页的『搜索 DB2 文档』

相关参考:

- 第 329 页的『 “DB2 通用数据库” 技术信息概述』

通过从管理工具访问 “DB2 信息中心” 来查找产品信息

“DB2 信息中心” 提供了对 DB2 产品信息的快速访问且在可以使用 DB2 管理工具的所有操作系统上可用。

从工具访问的 “DB2 信息中心” 提供了六种类型的信息。

任务 可使用 DB2 执行的关键任务。

概念 DB2 的关键概念。

参考 DB2 参考信息，如关键字、命令以及 API。

故障诊断
帮助您解决常见 DB2 问题的错误消息和信息。

样本 随 DB2 提供的样本程序的 HTML 列表的链接。

教程 用来帮助您了解 DB2 功能部件的指导性辅助。

先决条件:

“DB2 信息中心” 中的某些链接指向因特网上的 Web 站点。要显示这些链接的内容，首先必须与因特网连接。

过程:

要通过从工具访问 “DB2 信息中心” 来查找产品信息:

1. 用下列方法之一启动 “DB2 信息中心”：
   - 从图形管理工具中，单击工具栏中的**信息中心**图标。还可从**帮助**菜单中选择它。
   - 在命令行中输入 **db2ic**。

2. 单击与试图查找的信息相关的信息类型的选项卡。
3. 浏览整个树并单击感兴趣的主题。“信息中心”将启动 Web 浏览器以显示信息。
4. 要查找信息而无须浏览列表，可单击列表右边的**搜索**图标。

   一旦“信息中心”启动了浏览器来显示信息，就可通过单击导航工具栏中的**搜索**图标来执行全文本搜索。

相关概念:

- 第 347 页的『易使用性』
- 第 349 页的『从浏览器访问的 DB2 信息中心』

相关任务:

- 第 339 页的『通过从浏览器访问“DB2 信息中心”来查找主题』
- 第 346 页的『搜索 DB2 文档』

直接从 DB2 HTML 文档 CD 联机查看技术文档

还可直接从 CD 读取可从 *DB2 HTML 文档 CD* 安装的所有 HTML 主题。因此，可查看文档而不必安装它。

限制:

由于“工具”帮助是从 DB2 产品 CD 而不是从 *DB2 HTML 文档 CD* 安装的，您必须安装 DB2 产品才能查看该帮助。

过程:

1. 插入 *DB2 HTML 文档* CD。在 UNIX 操作系统上，安装 *DB2 HTML 文档 CD*。有关如何在 UNIX 操作系统上安装 CD 的详细信息，参考《*快速入门*》一书。
2. 启动 HTML 浏览器并打开适当的文件:
   - 对于 Windows 操作系统:

     ```
     e:\program files\IBM\SQLLIB\doc\htmlcd\%L\index.htm
     ```

     其中 *e* 表示 CD-ROM 驱动器，%L 是想要使用的文档的语言环境，例如，**en_US** 表示英语。
   - 对于 UNIX 操作系统:

     ```
     /cdrom/program files/IBM/SQLLIB/doc/htmlcd/%L/index.htm
     ```

     其中 */cdrom/* 表示安装 CD 的地方，%L 是想要使用的文档的语言环境，例如，**en_US** 表示英语。

相关任务:

- 第 339 页的『通过从浏览器访问 “DB2 信息中心” 来查找主题』
- 第 344 页的『将文件从 DB2 HTML 文档 CD 复制到 Web 服务器』

相关参考:

- 第 329 页的『 “DB2 通用数据库” 技术信息概述』

更新安装在机器上的 HTML 文档

现在，就有可能在 IBM 进行了更新之后更新从 *DB2 HTML 文档 CD* 安装的 HTML。可用以下两种方法之一来完成:

- 使用 “信息中心” （如果安装了 DB2 管理 GUI 工具的话）。
- 通过下载和应用 DB2 HTML 文档修订包。

注: 这将不会更新 DB2 代码; 它只更新从 *DB2 HTML 文档 CD* 安装的 HTML 文档。

过程:

要使用 “信息中心” 来更新本地文档:

1. 用下列方法之一启动 “DB2 信息中心”:
   - 从图形管理工具中，单击工具栏中的**信息中心**图标。还可从**帮助**菜单中选择它。
   - 在命令行中输入 **db2ic**。
2. 确保您的机器对外部因特网具有访问权; 更新程序将从 IBM 服务器下载最新的文档修订包（如果需要的话）。
3. 从菜单中选择**信息中心** —> **更新本地文档**以启动更新。
4. 提供代理信息（如果需要的话）以连接至外部因特网。

“信息中心” 将下载并应用最新的文档修订包（如果有的话）。

要手工下载并应用文档修订包:

1. 确保机器已连接至因特网。
2. 在浏览器中打开 DB2 支持页，网址为: www.ibm.com/software/data/db2/udb/winos2unix/support。
3. 访问版本 8 的链接并查找 “文档修订包” （Documentation FixPaks）链接。
4. 通过将文档修订包级别与已安装的文档级别进行比较来确定本地文档的版本是否已过时。您机器上的此当前文档处于以下级别: **DB2 v8.1 GA**。

5. 如果有更新的文档版本，则下载适用于您的操作系统的修订包。有一个适用于所有 Windows 平台的修订包和一个适用于所有 UNIX 平台的修订包。
6. 应用修订包:
   - 对于 Windows 操作系统: 文档修订包是自解压 zip 文件。将下载的文档修订包置于一个空目录中并运行它。这将创建一个 **setup** 命令，可运行该命令来安装文档修订包。
   - 对于 UNIX 操作系统: 文档修订包是压缩的 tar.Z 文件。解压并解取该文件。这将创建一个带有称为 **installdocfix** 的脚本的名为 `delta_install` 的目录。运行此脚本来安装文档修订包。

相关任务:

- 第 344 页的『将文件从 DB2 HTML 文档 CD 复制到 Web 服务器』

相关参考:

- 第 329 页的『 “DB2 通用数据库” 技术信息概述』

将文件从 DB2 HTML 文档 CD 复制到 Web 服务器

在 *DB2 HTML 文档 CD* 上交付了整个 DB2 信息库，可将它安装在 Web 服务器上以更便于访问。将想要的语言的文档复制至 Web 服务器即可。

注: 当您通过低速连接从 Web 服务器访问 HTML 文档时，可能会遇到性能较低的情况。

过程:

要将文件从 *DB2 HTML 文档 CD* 复制到 Web 服务器，使用适当的源路径:

- 对于 Windows 操作系统:

  ```
  E:\program files\IBM\SQLLIB\doc\htmlcd\%L\*.*
  ```

  其中 *E* 表示 CD-ROM 驱动器，*%L* 表示语言标识符。
- 对于 UNIX 操作系统:

  ```
  /cdrom/program files/IBM/SQLLIB/doc/htmlcd/%L/*.*
  ```

  其中 *cdrom* 表示 CD-ROM 驱动器的安装点，*%L* 表示语言标识。

相关任务:

- 第 346 页的『搜索 DB2 文档』

相关参考:

- 『受支持的 DB2 界面语言、语言环境和代码页』（*《DB2 服务器快速入门》*）

- 第 329 页的『 “DB2 通用数据库”技术信息概述』

对于使用 Netscape 4.x 搜索 DB2 文档进行故障诊断

大多数搜索问题都与 web 浏览器提供的 Java 支持有关。此任务描述可能的解决办法。

过程:

一个 Netscape 4.x 常见问题是丢失和误置安全性类。尝试下列解决办法，尤其是当您在浏览器 Java 控制台中看到以下行时更应尝试此方法:

```
找不到类 java/security/InvalidParameterException
```

- 在 Windows 操作系统上:

  从 *DB2 HTML 文档 CD*，将提供的 *x*:program files\IBM\SQLLIB\doc\htmlcd*locale*\InvalidParameterException.class 文件复制到相对于 Netscape 浏览器安装的 java\classes\java\security\ 目录，其中 *x* 表示 CD-ROM 驱动器盘符，*locale* 表示期望的语言环境的名称。

  注: 可能必须创建 java\security\ 子目录结构。

- 在 UNIX 操作系统上:

  从 *DB2 HTML 文档 CD*，将提供的 */cdrom*/program files/IBM/SQLLIB/doc/htmlcd/*locale*/InvalidParameterException.class 文件复制到相对于 Netscape 浏览器安装的 java/classes/java/security/ 目录，其中 *cdrom* 表示 CD-ROM 的安装点，*locale* 表示期望的语言环境的名称。

  注: 可能必须创建 java/security/ 子目录结构。

如果 Netscape 浏览器仍无法显示搜索输入窗口，则尝试下列操作:

- 停止 Netscape 浏览器的所有实例以确保机器上无任何 Netscape 代码运行。然后，打开 Netscape 浏览器的新实例并再次尝试启动搜索。
- 清除浏览器的高速缓存。
- 尝试用 Netscape 的其它版本或另一种浏览器。

相关任务:

- 第 346 页的『搜索 DB2 文档』

搜索 DB2 文档

可搜索 DB2 文档库来定位所需的信息。单击 “DB2 信息中心” （从浏览器访问）导航工具栏中的搜索图标时，将打开一个弹出式搜索窗口。可能需要一分钟来装入搜索，取决于您的计算机和网络的速度。

先决条件:

需要 Netscape 6.1 或更高版本或者 Microsoft 的 Internet Explorer 5 或更高版本。确保启用了浏览器的 Java 支持。

限制:

使用文档搜索时，将存在下列限制:

- 搜索不能区分大小写。
- 不支持布尔搜索。
- 不支持通配符搜索和部分搜索。例如，对 *java**（或 *java*）的搜索将仅查找文字字符串 *java**（或 *java*），而找不到 *javadoc*。

过程:

要搜索 DB2 文档:

1. 在导航工具栏中，单击**搜索**图标。
2. 在 “搜索” 窗口最上面的文本输入字段中，输入一个或多个与您感兴趣的领域相关的词条（由空格分隔），并单击**搜索**。一个按准确度排列的主题列表将显示在**结果**字段中。每一单项旁的数字等级提供了匹配程度的指示（较大的数字表示较高的匹配程度）。

   输入较多的项会提高查询的精度，同时还会减少从查询返回的主题数目。
3. 在**结果**列表中，单击要阅读的主题的标题。主题将显示在 “DB2 信息中心” 的内容框架中。

注: 执行搜索时，第一个（最高级别的）结果自动装入到浏览器框架中。要查看其它搜索结果的内容，单击结果列表中的结果。

相关任务:

- 第 345 页的『对于使用 Netscape 4.x 搜索 DB2 文档进行故障诊断』

联机 DB2 故障诊断信息

在 DB2® UDB 版本 8 的发行版中，将不再提供 *Troubleshooting Guide*。曾经包含在此指南中的故障诊断信息都已集成到 DB2 出版物中，从而使我们能向您提供最新信息。要查找有关故障诊断实用程序和 DB2 功能的信息，可从任何工具访问“DB2 信息中心”。

如果您遇到问题且想要获取查找可能原因及解决方案的帮助，请参考 Online Support 站点。该支持站点包含了一个不断更新的大型数据库，数据库的内容涉及 DB2 出版物、技术说明、APAR（产品问题）记录、修订包和其它资源。可使用该支持站点来搜索此知识库并查找问题的可能解决方案。

访问 Online Support 站点（网址为 www.ibm.com/software/data/db2/udb/winos2unix/support），或通过单击“DB2 信息中心”中的**在线支持**按钮来访问它。现在，还可从此站点获取经常更改的信息，如内部 DB2 错误代码列表。

相关概念:

- 第 349 页的『从浏览器访问的 DB2 信息中心』

相关任务:

- 第 341 页的『通过从管理工具访问“DB2 信息中心”来查找产品信息』

易使用性

易使用性功能部件可帮助那些身体有某些缺陷（如活动不方便或视力不太好）的用户成功使用软件产品。以下是“DB2® 通用数据库版本 8”中主要的易使用性功能部件:

- 通过键盘即可对所有 DB2 功能部件进行操作，而不必使用鼠标。参见第 348 页的『键盘输入和导航』。
- DB2 允许您定制字体的大小和颜色。参见第 348 页的『界面显示的易使用性』。
- DB2 允许您接收可视或音频警告信号。参见第 348 页的『备用警告信号』。
- DB2 支持使用 Java™ Accessibility API 的易使用性应用程序。参见第 348 页的『与辅助技术的兼容性』。
- DB2 附带了以易使用的格式提供的文档。参见第 348 页的『可访问文档』。

键盘输入和导航

键盘输入

只使用键盘就可对“DB2 工具”进行操作。使用键或键组合就可执行使用鼠标完成的大多数操作。

键盘焦点

在基于 UNIX 的系统中，键盘焦点的位置是突出显示的，指示窗口的哪个区域处于活动状态且击键对何处会有影响。

界面显示的易使用性

“DB2 工具”中的功能部件增强了用户界面，使视力不太好的用户更易使用。这些易使用性方面的增强包括了对可定制字体特性的支持。

字体设置

“DB2 工具”允许您通过使用“工具设置”笔记本来选择菜单和对话框窗口中文本的颜色、大小和字体。

不依赖于颜色

不需要分辨颜色就可以使用此产品中的任何功能。

备用警告信号

可使用“工具设置”笔记本来指定是否想要通过音频或可视信号接收警告。

与辅助技术的兼容性

“DB2 工具”界面支持对屏幕阅读器启用 Java Accessibility API 并支持有某些缺陷的用户使用其它辅助技术。

可访问文档

DB2 产品系列的文档提供了 HTML 格式的版本。使您可根据浏览器中设置的显示首选项来查看文档。还允许您使用屏幕阅读器和其它辅助性技术。

DB2 教程

DB2® 教程帮助您了解“DB2 通用数据库”的各个方面。教程提供了开发应用程序、调整 SQL 查询性能、使用数据仓库、管理元数据和使用 DB2 开发 Web 服务等方面的课程，这些课程中还提供了逐步指示信息。

开始之前:

必须先从 *DB2 HTML 文档* CD 中安装教程，才能使用以下的链接来访问这些教程。

如果不想安装这些教程，则可直接从 *DB2 HTML 文档 CD* 查看这些教程的 HTML 版本。还可在 *DB2 PDF 文档 CD* 上获取这些教程的 PDF 版本。

某些教程课程使用了样本数据或代码。有关各个教程特定任务的任何先决条件的描述，参见每个教程的内容。

“DB2 通用数据库”教程:

如果从 *DB2 HTML 文档* CD 安装了教程，则可单击下表中的某个教程标题来查看该教程。

《商务智能教程: 数据仓库中心简介》
使用“数据仓库中心”来执行介绍性的数据入库任务。

《商务智能教程: 数据入库的扩展课程》
使用“数据仓库中心”来执行高级数据入库任务。

Development Center Tutorial for Video Online using Microsoft® Visual Basic
使用 Microsoft Visual Basic 的“开发中心加载件”来构建应用程序的各个组件。

Information Catalog Center Tutorial
使用“信息目录中心”来创建和管理信息目录以定位并使用元数据。

Video Central for e-business Tutorial
使用 WebSphere® 产品来开发和部署高级“DB2 Web 服务”应用程序。

《Visual Explain 教程》
使用 Visual Explain 来分析、优化和调整 SQL 语句以获取更好的性能。

从浏览器访问的 DB2 信息中心

“DB2® 信息中心”让您可以访问在您的业务中充分利用 DB2 通用数据库™ 和 DB2 Connect™ 所需的所有信息。“DB2 信息中心”文档还记录主要的 DB2 功能部件和组件，包括复制、数据入库、信息目录中心、Life Sciences Data Connect 和 DB2 extender。

从浏览器访问的“DB2 信息中心”具有以下功能部件（如果是在 Netscape Navigator 6.1 或更高版本或者 Microsoft Internet Explorer 5 或更高版本中查看）。某些功能部件需要您启用对 Java 或 JavaScript 的支持:

定期更新的文档

通过下载更新的 HTML，使您的主题保持为最新。

搜索 通过单击导航工具栏中的**搜索**来搜索安装在工作站上的所有主题。

集成的导航树

从一个导航树中就可找出 DB2 资料库中的任何主题。导航树是按信息类型组织的，如下所示:

- “任务”提供了有关如何完成目标的逐步指示信息。
- “概念”提供了主题的概述。
- “参考”主题提供了有关主题的详细信息，包括语句和命令语法、消息帮助以及需求。

主索引 从主索引访问从 *DB2 HTML 文档 CD* 中安装的信息。索引是按索引项以字母顺序组织的。

主词汇表

主词汇表定义在“DB2 信息中心”中使用的术语。词汇表是按词汇表术语以字母顺序组织的。

相关任务:

- 第 339 页的『通过从浏览器访问“DB2 信息中心”来查找主题』
- 第 341 页的『通过从管理工具访问“DB2 信息中心”来查找产品信息』
- 第 343 页的『更新安装在机器上的 HTML 文档』

附录 J. 声明

IBM 可能在其它国家或地区不提供本文档中讨论的产品、服务或功能特性。有关您当前所在区域的产品和服务的信息，请向您当地的 IBM 代理咨询。任何对 IBM 产品、程序或服务的引用并非意在明示或暗示只能使用 IBM 的产品、程序或服务。只要不侵犯 IBM 的知识产权，任何同等功能的产品、程序或服务，都可以代替 IBM 产品、程序或服务。但是，评估和验证任何非 IBM 产品、程序或服务，则由用户自行负责。

IBM 公司可能已拥有或正在申请与本文档内容有关的各项专利。提供本文档并未授予用户使用这些专利的任何许可证。您可以用书面方式将许可证查询寄往:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

有关双字节（DBCS）信息的许可证查询，请与您所在国家或地区的 IBM 知识产权部门联系，或用书面方式将查询寄往:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

本条款不适用于联合王国或任何这样的条款与当地法律不一致的国家或地区: 国际商业机器公司以“按现状”的基础提供本出版物，不附有任何形式的（无论是明示的，还是默示的）保证，包括（但不限于）对非侵权性、适销性和适用于某特定用途的默示保证。某些国家或地区在某些交易中不允许免除明示或默示的保证。因此，本条款可能不适用于您。

本信息中可能包含技术方面不够准确的地方或印刷错误。此处的信息将定期更改；这些更改将编入本资料的新版本中。IBM 可以随时对本资料中描述的产品和／或程序进行改进和／或更改，而不另行通知。

本信息中对非 IBM Web 站点的任何引用都只是为了方便起见才提供的，不以任何方式充当对那些 Web 站点的保证。该 Web 站点中的资料不是 IBM 产品资料的一部分，使用那些 Web 站点带来的风险将由您自行承担。

IBM 可以它认为合适的任何方式使用或分发您所提供的任何信息，而无须对您承担任何责任。

本程序的被许可方如果要了解有关程序的信息以达到如下目的: （i）允许在独立创建的程序和其它程序（包括本程序）之间进行信息交换，以及（ii）允许对已经交换的信息进行相互使用，请与下列地址联系:

IBM Canada Limited
Office of the Lab Director
8200 Warden Avenue
Markham, Ontario
L6G 1C7
CANADA

只要遵守适当的条件和条款，包括某些情形下的一定数量的付费，都可获得这方面的信息。

本文档中描述的许可程序及其所有可用的许可资料均由 IBM 依据 IBM 客户协议、IBM 国际程序许可证协议或任何同等协议中的条款提供。

此处包含的任何性能数据都是在受控环境中测得的。因此，在其它操作环境中获得的数据可能会有明显的不同。有些测量可能是在开发级的系统上进行的，因此不保证与一般可用系统上进行的测量结果相同。此外，有些测量是通过推算而估计的，实际结果可能会有差异。本文档的用户应当验证其特定环境的适用数据。

涉及非 IBM 产品的信息可从这些产品的供应商、其出版说明或其它可公开获得的资料中获取。IBM 没有对这些产品进行测试，也无法确认其性能的精确性、兼容性或任何其它关于非 IBM 产品的声明。有关非 IBM 产品性能的问题应当向这些产品的供应商提出。

所有关于 IBM 未来方向或意向的声明都可随时更改或收回，而不另行通知，它们仅仅表示了目标和意愿而已。

本资料中可能包含用于日常业务运作的数据和报表的示例。为了尽可能完整地说明问题，这些示例可能包含个人、公司、品牌和产品的名称。所有这些名称都是虚构的，如与实际商业企业所使用的名称和地址有雷同，纯属巧合。

版权许可证:

本资料中可能包含源语言的样本应用程序，它们举例说明了各种操作平台上的编程技术。为了开发、使用、营销或分发符合编写这些样本程序所针对操作平台的应用程序编程接口的应用程序，您可以以任何形式复制、修改和分发这些样本程

序，而不必向 IBM 付款。尚未在所有条件下彻底测试这些示例。因此，IBM 不能保证或默示这些程序的可靠性、适用性或功能。

这些样本程序或任何派生产品的每个副本或任何部分都必须包括如下版权声明:

商标

下列各项是国际商业机器公司在美国和／或其它国家或地区的商标，且已在 DB2 UDB 文档库中的至少一份文档中使用。

ACF/VTAM
AISPO
AIX
AIXwindows
AnyNet
APPN
AS/400
BookManager
C Set++
C/370
CICS
Database 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Extender
DB2 OLAP Server
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
eServer
Extended Services
FFST
First Failure Support Technology
IBM
IMS
IMS/ESA
iSeries
LAN Distance
MVS
MVS/ESA
MVS/XA
Net.Data
NetView
OS/390
OS/400
PowerPC
pSeries
QBIC
QMF
RACF
RISC System/6000
RS/6000
S/370
SP
SQL/400
SQL/DS
System/370
System/390
SystemView
Tivoli
VisualAge
VM/ESA
VSE/ESA
VTAM
WebExplorer
WebSphere
WIN-OS/2
z/OS
zSeries

下列各项是其它公司的商标或注册商标，且已在 DB2 UDB 文档库中的至少一份文档中使用:

Microsoft、Windows、Windows NT 和 Windows 徽标是 Microsoft Corporation 在美国和／或其它国家或地区的商标。

Intel 和 Pentium 是 Intel Corporation 在美国和／或其它国家或地区的商标。

Java 和所有基于 Java 的商标是 Sun Microsystems, Inc. 在美国和／或其它国家或地区的商标。

UNIX 是 The Open Group 在美国和其它国家或地区的注册商标。

其它公司、产品或服务名称可能是其它公司的商标或服务标记。

索引

[G]

[H]

[J]

[K]

[L]

[Y]

[Z]

A

B

D

E

H

I

L

M

N

O

P

R

S

T

U

V

W

X

与 IBM 联系

在美国，请致电下列其中一个号码以与 IBM 联系:

- 1-800-237-5511，可获取客户服务
- 1-888-426-4343，可了解所提供的服务项目
- 1-800-IBM-4YOU（426-4968），可获取有关 DB2 市场营销与销售的信息

在加拿大，请致电下列其中一个号码以与 IBM 联系:

- 1-800-IBM-SERV (1-800-426-7378)，可获取客户服务
- 1-800-465-9600，可了解所提供的服务项目
- 1-800-IBM-4YOU (1-800-426-4968)，可获取有关 DB2 市场营销与销售的信息

要查找您所在国家或地区的 IBM 营业处，可查看 IBM 全球联系人目录（IBM Directory of Worldwide Contacts），网址为 www.ibm.com/planetwide。

产品信息

有关 “DB2 通用数据库” 产品的信息，可打电话获取或通过万维网获取，网址为: www.ibm.com/software/data/db2/udb。

此站点包含有关技术库、订购书刊、客户机下载、新闻组、修订包、新闻和 Web 资源链接的最新信息。

您如果住在美国，请致电下列其中一个号码:

- 1-800-IBM-CALL（1-800-426-2255），可订购产品或获取一般信息。
- 1-800-879-2755，可订购出版物。

有关如何在美国以外的国家或地区与 IBM 联系的信息，请访问 IBM Worldwide 页面，网址为 www.ibm.com/planetwide。

中国印刷

S152-0181-00

Spine information:

IBM® DB2 通用数据库™ 数据恢复和高可用性指南与参考 *版本 8*