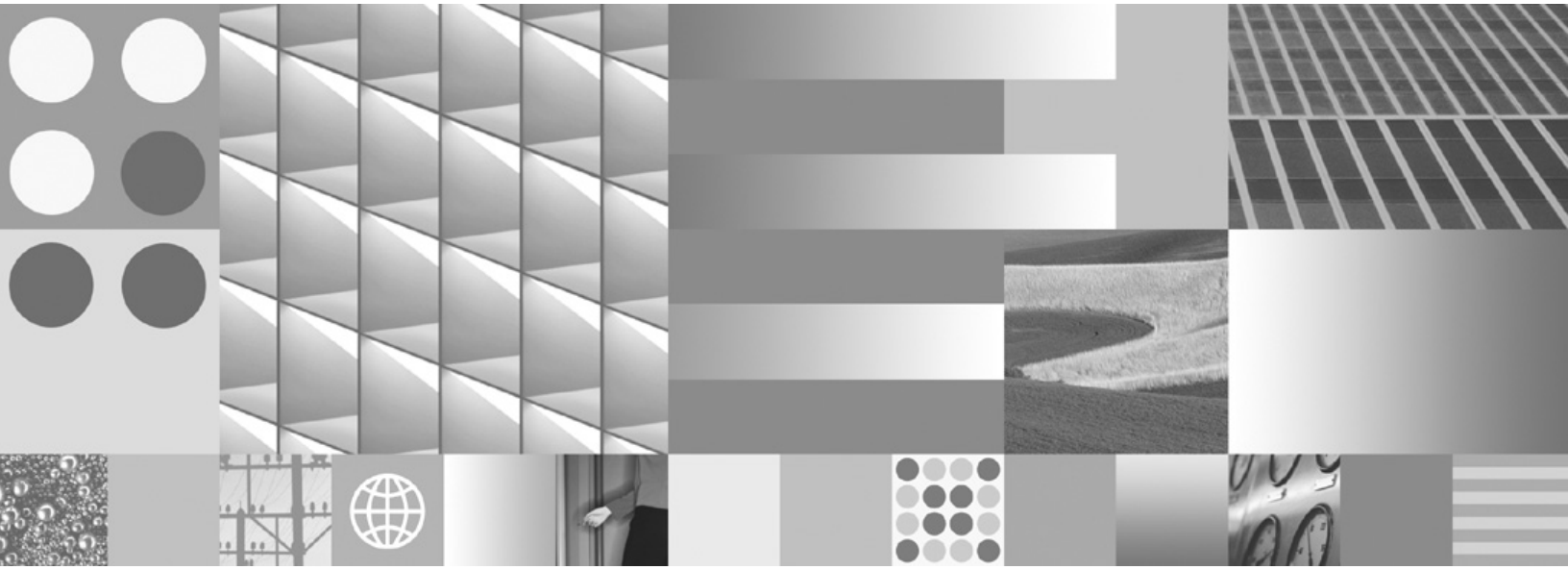
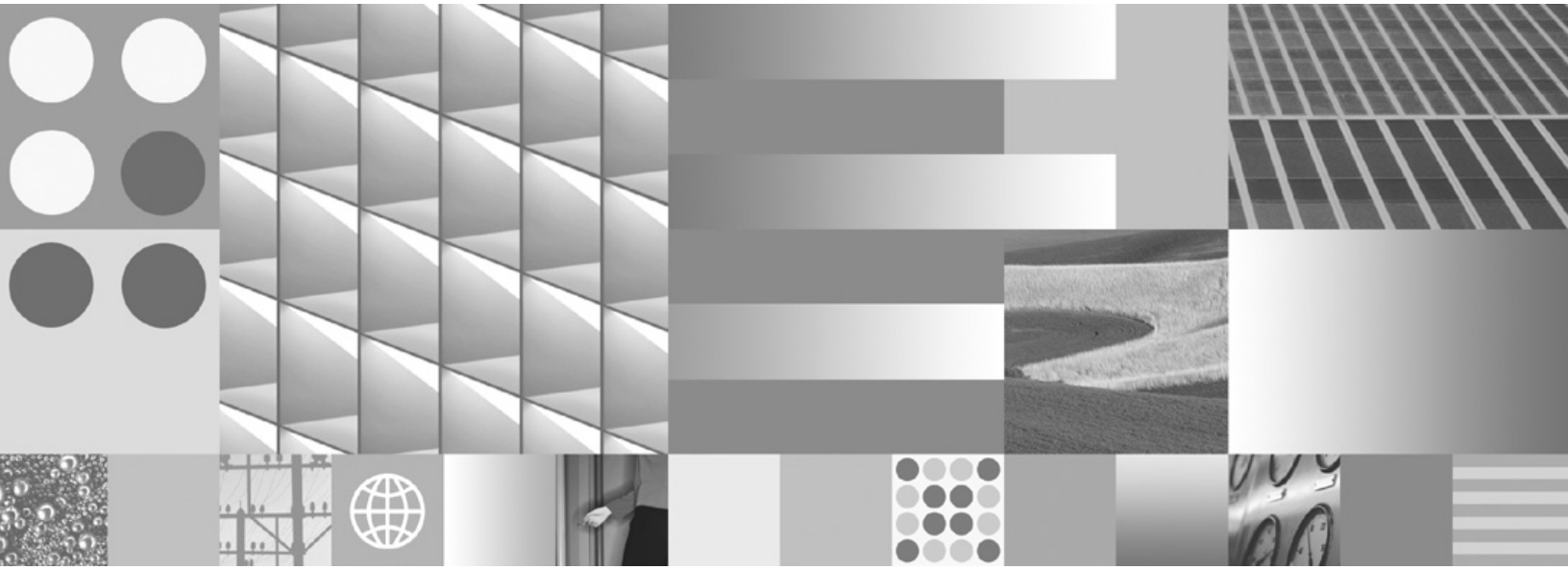
DB2 版本 9.5
Linux 版、UNIX 版和 Windows 版



调整数据库性能

S151-0613-00

DB2 版本 9.5
Linux 版、UNIX 版和 Windows 版



调整数据库性能

S151-0613-00

注意

使用此信息及其支持的产品前，请先阅读第 383 页的附录 B，『声明』下的常规信息。

修订版声明

此文档包含 IBM 的所有权信息。它在许可协议中提供，且受版权法的保护。本出版物中包含的信息不包括对任何产品的保证，且提供的任何语句都不需要如此解释。

您可在线或通过当地的 IBM 代表处订购 IBM 出版物。

- 要在线订购出版物，请转至 IBM 出版物中心，网址为: www.ibm.com/shop/publications/order
- 要查找当地的 IBM 代表处，请转至 IBM 全球联系人目录，网址为: www.ibm.com/planetwide

要从美国或加拿大的 DB2 市场和销售部订购 DB2 出版物，请致电 1-800-IBM-4YOU（426-4968）。

当您向 IBM 发送信息时，即同意授予 IBM 独一无二的权力以它认为适当且不会对您造成任何影响的方式使用或分发该信息。

目录

第 1 部分 性能元素

*性能*是计算机系统在给定工作负载的情况下的行为方式。按照系统响应时间、吞吐量和可用性来测量性能。性能还受以下因素影响:

- 系统中可用的资源
- 如何充分利用和共享这些资源。

一般情况下，可调整您的系统来改进其成本和效益比率。具体目标可能包括:

- 处理更大的或更紧迫的工作负载，而不增加处理成本

  例如，增加工作负载而不用购买新硬件或占用更多处理器时间
- 获得更快的系统响应时间或更大的吞吐量，而不增加处理成本
- 降低处理成本，而不会降低对用户的服务

将性能从技术指标转换为经济指标比较困难。调整性能在用户时间和处理器时间方面一定会提高成本，因此在对项目进行调整前应衡量其相对于可能效益的成本。其中某些效益是有形的:

- 更有效地利用资源
- 能够向系统添加更多的用户

其他效益是无形的，例如，由于响应更快而让用户更加满意。应考虑所有这些效益。

第 1 章 性能调整准则

以下准则可帮助您制定一个调整性能的总体方案。

记住递减返回定律：最大的性能收益通常来自于最初的努力。以后的更改通常只能产生越来越小的效益，并且需要更多努力。

不要只为调整而调整：进行调整以释放标识的约束。如果调整的资源不是造成性能问题的主要原因，这种调整对响应时间几乎不产生影响，除非您释放了主要约束，而且这种调整实际上会使后续调整工作更加困难。如果有可能明显提高性能的话，那么关键在于对某些作为影响响应时间的主要因素的资源的性能提高。

考虑整个系统：永远不能片面地调整一个参数或系统。在进行任何调整前，务必考虑它将对整个系统带来的影响。

一次更改一个参数：不要一次更改多个性能调整参数。即使您肯定所有更改都有好处，也没有任何办法来评估每个更改所带来的影响。如果一次更改多个参数，也不能有效地判断所做的更改的利弊。如果每次调整一个参数来改进一方面，几乎总是会影响至少一个您可能没有考虑到的其他方面。通过一次更改一个，允许您使用基准程序来评估您是否需要进行更改。

按级别测量和重新配置：和一次只应更改一个参数的理由一样，一次也只能调整系统的一个级别。可使用以下的系统级别列表作为参考:

- 硬件
- 操作系统
- 应用程序服务器和请求器
- 数据库管理器
- SQL 和 XQuery 语句
- 应用程序

检查是否存在硬件和软件问题：某些性能问题可通过维修硬件和/或修订软件来解决。如果通过维修或修订就可解决问题，就不需要花过多时间来监视和调整系统。

在升级硬件前搞清楚问题：即使增加存储器或提高处理器能力可立即改善性能，也应花时间了解系统的瓶颈所在。可能花钱增加磁盘存储器后，才发现系统没有处理能力或可利用它的通道。

在开始调整前执行回退过程：正如前面所讲，某些调整可能产生意外的性能结果。如果此调整使性能降低，应撤销该调整，改试另一种调整。如果保存了以前的设置并可重新调用它，那么撤销不正确的信息将变得非常容易。

第 2 章 开发性能提高进程

性能提高进程是一种可重复的长期方法，用于监视和调整性能的各个方面。您和您的性能小组可以根据监视的结果调整数据库服务器的配置，并可对使用数据库服务器的应用程序进行更改。

性能监视和调整决策必须以您对使用数据的应用程序类型的了解程度和数据访问的模式为依据。不同类型的应用程序具有不同的性能要求。

将下面的性能提高流程概述作为提高性能的准则。

要开发性能提高流程:

1. 定义性能目标。
2. 为系统中的主要约束建立性能指示器。
3. 开发并执行性能监视方案。
4. 连续分析监视结果以确定哪些资源需要调整。
5. 每次进行一项调整。

   即使您认为需要调整多个资源，或数个调整选项可用于您要调整的资源，也要一次进行一项调整，从而确保所做的调整工作可以产生所需的效果。在某些点，调整数据库服务器和应用程序并不能提高性能。因此，您需要升级硬件。

实际的性能调整要求综合考虑系统资源。例如，为了提高 I/O 性能，您可能会增大缓冲池的容量，但是，缓冲池的容量越大，所需的内存也就越多，这将会降低其他方面的性能。

第 3 章 用户可以提供的性能信息

需要对系统进行调整的首个征兆可能是用户提出的意见。如果您没有足够的时间来设定性能目标并通过一种完备的方式来监视和调整，可了解用户的意见以改善性能。通常可提出几个简单的问题，来确定从哪里开始查找问题。例如，可以询问用户:

- “响应慢”达到何种程度？是比预期的慢 10% 还是慢数十倍？
- 问题是何时发现的？它是最近出现的，还是一直都存在？
- 其他的用户有相同问题吗？这些用户是一两个人还是整个一组？
- 如果是一组用户遇到相同问题，他们是否与同一个局域网连接？
- 这些问题似乎与特定事务或应用程序相关吗？
- 注意到问题出现时有任何模式吗？例如，问题是否是在一天的特定时间发生，如午餐时间，或它持续的时间是长还是短？

第 4 章 性能调整限制

调整仅能够对系统的效率进行一定程度的更改。考虑要投入多少时间和费用来改善系统性能，以及要投入多少额外的时间和费用来帮助系统的用户。

例如，如果系统遇到性能瓶颈，通常可以通过调整来提高性能。如果已接近系统的性能限制，而用户数大约增加 10%，那么响应时间可能远远不止增加 10%。在这种情况下，需要确定如何调整系统来抵消降低的性能。

但是，有一个临界点，超过这个临界点再进行调整就没有用了。达到该临界点时，应考虑在环境的限制下修改您的目标和期望值。要显著提高性能，可能需要添加更多磁盘存储器、更快的 CPU、更多 CPU、更多主存储器、更快的通信链路，或者它们的组合。

第 5 章 DB2 体系结构和进程概述

有关 DB2® 体系结构和进程的一般信息可以帮助您了解对特定主题提供的详细信息。

下图提供了对 IBM® DB2 版本 9.1 的体系结构和进程的一般概述。

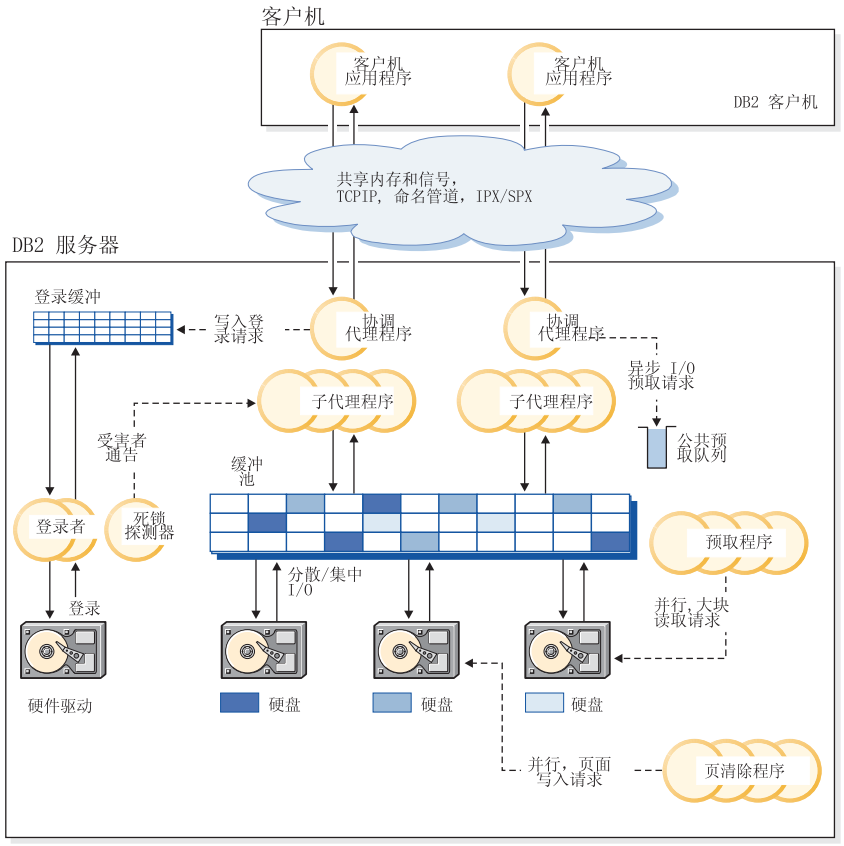


图 1. 体系结构和进程概述

在客户机端，本地和/或远程应用程序都与 DB2 客户机库链接。本地客户机使用共享内存和信号进行通信；远程客户机使用一种协议（例如，命名管道（NPIPE）或 TCP/IP）进行通信。

在服务器端，活动由引擎可调度单元（EDU）控制。在本节中的所有图形中，EDU 显示为圆圈或许多组圆圈。在 V9.5 中，EDU 在所有平台上都是作为线程来实现的。DB2 代理程序是最常见的 EDU 类型。这些代理程序代表应用程序执行大量 SQL 和 XQuery 处理。其他常见的 EDU 是预取程序和页清除程序。

可以指定一组子代理程序来处理客户机应用程序请求。如果服务器所在的机器具有多个处理器或是分区数据库的一部分，可以指定多个子代理程序。例如，在对称多处理技术（SMP）环境中，多个 SMP 子代理程序可以利用许多处理器。

所有代理程序和子代理程序都是使用特定入池算法来管理的，该算法将 EDU 的创建和破坏减至最少。

缓冲池是数据库服务器内存的一个区域，用户表数据、索引数据和目录数据的数据库页被临时地移至该区域，并可以在该处被修改。因为从内存访问数据可以比从磁盘访问数据快得多，所以缓冲池是数据库性能的重要因素。如果应用程序需要的多个数据存在于缓冲池，那么访问数据所需的时间比在磁盘上查找要少。

缓冲池以及预取程序和页清除程序 EDU 的配置控制如何迅速访问数据以及对应用程序是如何容易地可用。

- **预取程序**在应用程序需要数据之前从磁盘检索该数据，并将其移到缓冲池中。例如，如果没有数据预取程序，那么需要扫描大量数据的应用程序将必须等待数据从磁盘移到缓冲池中。应用程序的代理程序将异步预读取请求发送至公共预取队列。当预取程序可用时，它们使用大块或散射读取输入操作来将请求的页从磁盘读入缓冲池，从而实现那些请求。如果具有多个磁盘来存储数据库数据，那么可以将数据条带分割到多个磁盘上。条带分割数据让预取程序同时使用多个磁盘来检索数据。
- **页清除程序**将数据从缓冲池移回到磁盘中。页清除程序是独立于应用程序代理程序的后台 EDU。它们将查找已经修改的页，并将已更改的那些页写入磁盘。页清除程序确保缓冲池中有空间供预取程序正在检索的页使用。

如果没有独立的预取程序和页清除程序 EDU，那么应用程序代理程序将必须执行缓冲池与磁盘存储器之间的所有数据读取和写入操作。

DB2 进程技术模型

DB2 进程技术模型的知识可以帮助您确定问题的性质，因为它会帮助您理解数据库管理器及与其相关联的组件如何交互作用。

所有 DB2 服务器使用的进程技术模型使数据库服务器与客户机以及本地应用程序之间的通信更加容易。它还确保数据库应用程序独立于如数据库控制块和关键数据库文件之类的资源。

DB2 服务器必须执行各种不同的任务，例如，处理数据库应用程序请求或确保日志记录已写入磁盘。通常，每项任务都由一个独立的引擎可调度单元（EDU）执行。在先前发行版中，Linux® 和 UNIX® 环境中的大多数 EDU 是使用独立的进程实现的，而 Windows 中的大多数 EDU 是使用主 DB2 服务器进程内的操作系统线程实现的。从版本 9.5 开始，Linux 和 UNIX 环境中的 DB2 服务器现在也是线程化的，因此，目前 UNIX 和 Windows 上的 EDU 都是使用操作系统线程实现的。

在 DB2 服务器中使用多线程体系结构有很多优点。由于同一进程内的所有线程可以共享一些操作系统资源，因此，新线程需要的内存和操作系统资源比进程要少。此外，在某些平台上，线程的上下文切换时间的成本比进程要低，这样可提高性能。但是，在所有平台上使用线程模型最重要的优点是使得更容易配置 DB2 服务器，这是因为这样更容易根据需要分配较多 EDU，并且可以动态分配多个 EDU 要共享的内存（在基于

进程的模型中，一个 EDU 看不到从另一个 EDU 分配的内存）。请参阅『简化的内存配置』和『代理程序和进程技术模型配置』章节，以了解有关这些增强功能的更多详细信息。

在先前发行版中，在 Linux 和 UNIX 系统上，ps 系统命令或 db2_local_ps 命令可用来列示所有活动的 DB2 EDU。从版本 9.5 开始，这些命令不再列示 db2sysc 进程内的任何 EDU 线程。作为一种替代方法，现在可使用带有 -edus 选项的 db2pd 命令来列示所有活动的 EDU 线程。此命令可在 UNIX 和 Windows 系统上工作。

对于正在访问的每个数据库，启动各种 EDU 以处理各种数据库任务，例如，预取、通信和日志记录。数据库代理程序是一类特殊的 EDU，创建它们是为了处理数据库的应用程序请求。

每个客户机应用程序连接都有一个**协调代理程序**在数据库上运行。协调代理程序代表应用程序工作，并根据需要使用专用内存、进程间通信（IPC）或远程通信协议与其他代理程序通信。

DB2 体系结构提供一个**防火墙**，以便应用程序在不同于 DB2 的地址空间中运行。防火墙将数据库和数据库管理器与应用程序、存储过程和用户定义的函数（UDF）隔开。防火墙维护数据库中数据的完整性，原因是它禁用应用程序编程错误覆盖数据库管理器的内部缓冲区或文件。防火墙还提高了可靠性，原因是应用程序错误不能使数据库管理器崩溃。

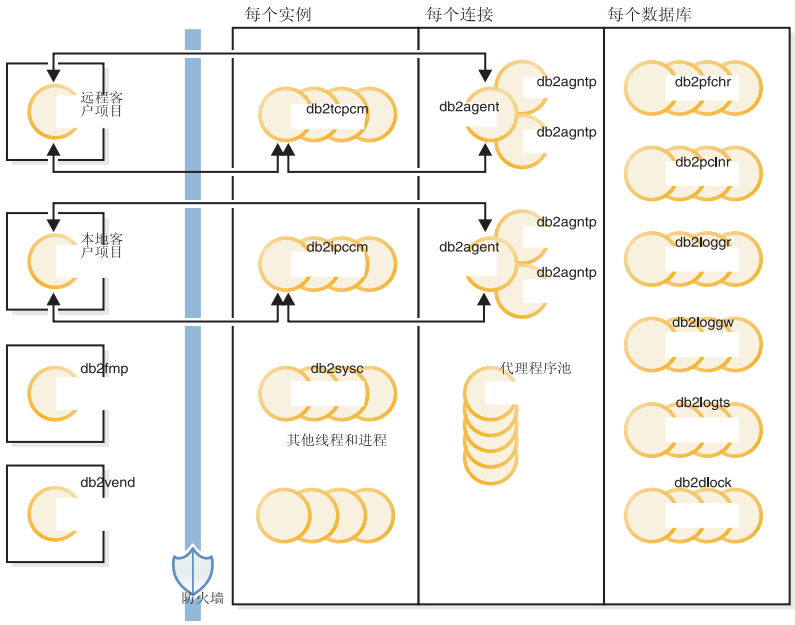


图 2. DB2 系统的进程技术模型

以下列表提供有关图中显示的对象的其他详细信息:

客户机程序

客户机程序以远程方式运行或作为数据库服务器在同一机器上运行。它们通过侦听器建立与数据库的首次联系。然后，会对它们指定协调代理程序（db2agentP）。

侦听器

客户机程序使用通信侦听器建立初始联系，侦听器是在 DB2 启动时启动的。每个已配置的通信协议都有一个侦听器，本地客户机程序使用进程间通信（IPC）侦听器（db2ipccm）。侦听器包括:

- **db2ipccm**，用于本地客户机连接
- **db2tcpcm**，用于 TCP/IP 连接
- **db2tcpdm**，用于 TCP/IP 发现工具请求

代理程序

会对来自客户机应用程序的所有连接请求（不管是本地的还是远程的）分配相应的协调代理程序（**db2agent**）。创建协调代理程序之后，它将代表应用程序执行所有数据库请求。

在启用了数据库分区功能部件（DPF）或启用了 *intra_query* 并行性的环境中，协调代理程序会将数据库请求分发给子代理程序（分别为 db2agntp 和 db2agnts）。这些代理程序为应用程序执行请求。一旦创建了协调代理程序，它就会代表其应用程序处理所有数据库请求，方法是协调对数据库执行请求的子代理程序（db2agntp）。与应用程序关联但当前处于空闲状态的子代理程序用名称 **db2agnta** 标识。

协调代理程序可能:

- 连接至具有别名的数据库。例如，“db2agent (DATA1)”连接至数据库别名“DATA1”。
- 连接至实例。例如，“db2agent (user1)”连接至实例“user1”。

DB2 进程技术模型将实例化其他类型的代理程序以执行特定操作，如独立的协调代理程序或子协调代理程序。例如，独立的协调代理程序 **db2agnti** 用于运行事件监视器，而子协调代理程序 **db2agnsc** 用于在突然关闭后以并行方式执行数据库重新启动。

空闲代理程序位于代理程序池中。这些代理程序对于来自代表客户机程序运行的协调代理程序或来自代表现有协调代理程序运行的子代理程序的请求是可用的。在涉及大量应用程序工作负载的配置中具有大小合适的空闲代理程序池有助于提高性能，这是因为可以根据需要立即使用空闲代理程序，而不必为每个应用程序连接分配一个全新的代理程序，这涉及到创建线程以及分配并初始化内存和其他资源。从版本 9.5 开始，DB2 还可以在需要时动态管理空闲代理程序池的大小。

db2fmp

设防方式进程。它负责在防火墙外执行受防护的存储过程和用户定义的函数。db2fmp 进程始终是独立的进程，但可能是多线程的，这取决于它执行的例程的类型。

db2vend

这是代表 EDU 执行供应商代码的进程，例如，执行用户出口程序以进行日志归档（仅适用于 UNIX）。

数据库 EDU

以下列表包括每个数据库使用的一些重要 EDU:

- **db2pfchr**，用于缓冲池预取程序

- **db2pclnr**，用于缓冲池页清除程序
- **db2loggr**，用于处理日志文件以处理事务处理和恢复
- **db2loggw**，用于将日志记录写入日志文件。
- **db2logts**，用于跟踪哪些表空间在哪些日志文件中具有日志记录。此信息记录在数据库目录中的 DB2TSCHG.HIS 文件中。它用于加快表空间前滚恢复的向前阶段的速度。
- **db2dlock**，用于死锁检测。在多分区数据库环境中，使用称为 **db2glock** 的附加线程来协调从每个分区上的 db2dlock EDU 收集来的信息。db2glock 仅在目录分区上运行。
- **db2taskd**，用于分发后台数据库任务。这些任务由称为 **db2taskp** 的线程执行。
- **db2hadrp**，HADR 主服务器线程
- **db2hadrs**，HADR 备用服务器线程
- **db2lfr**，用于处理各个日志文件的日志文件阅读器
- **db2shred** 处理日志页中的各个日志记录
- **db2redom**，用于重做主进程。在恢复期间，处理重做日志记录并将日志记录指定给重做工作程序来进行处理。
- **db2redow**，用于重做工作程序。在恢复期间，在重做主进程请求时处理重做日志记录。
- **db2logmgr**，用于日志管理器。管理可恢复数据库的日志文件。
- **db2wlmd**，用于自动收集工作负载管理统计信息。
- 按如下所示标识事件监视器线程:
  - **db2evm%1%2 (%3)**，其中 **%1** 可以为:
    - **g** – 全局文件事件监视器
    - **l** – 本地文件事件监视器
    - **t** – 表事件监视器
    - **gp** – 全局管道事件监视器
    - **lp** – 本地管道事件监视器

    其中 **%2** 可以为:
    - **i** – 协调程序
    - **p** – 不是协调程序

    而 **%3** 是事件监视器名称
- 按如下所示标识备份和复原线程:
  - **db2bm.%1.%2** 是备份和复原缓冲区操纵程序，而 **db2med.%1.%2** 是备份和复原介质控制器，其中
    - **%1** – 控制备份或复原会话的代理程序的 EDU 标识
    - **%2** – 用于区分属于特定备份或复原会话的线程（可能有许多个）的顺序值

    例如: `db2bm.13579.2` 标识具有 EDU 标识为 13579 的 db2agent 线程控制的第二个 db2bm 线程。

数据库服务器线程和进程

系统控制器（在 UNIX 上为 **db2sysc**，**而在 Windows 上为 db2syscs.exe**）必须存在，数据库服务器才能工作。另外，必须启动下列线程和进程，才能执行各种任务:

- **db2resync**，扫描全局再同步列表的再同步代理进程
- **db2wdog**，在 UNIX 和 Linux 操作系统上处理异常终止的看守程序
- **db2fcms**，快速通信管理器发送方守护程序
- **db2fcmr**，快速通信管理器接收方守护程序
- **db2pdbc**，并行系统控制器，用来处理来自远程节点的并行请求（仅在分区数据库环境中使用）。
- **db2cart**，用于访问配置为启用 USEREXIT 的数据库时的归档日志文件
- **db2fmtlg**，用于访问配置为启用 LOGRETAIN 但禁用 USEREXIT 的数据库时的格式化日志文件
- **db2panic**，紧急代理程序，在某特定节点上达到代理程序的限制时处理紧急请求（仅在分区数据库环境中使用）
- **db2srvlst**，管理诸如 DB2 z/OS 版的系统的地址列表
- **db2fmd**，故障监视器守护程序
- **db2disp**，客户机连接集中器分派器
- **db2acd**，主管运行状况监视器和自动维护实用程序的自主计算守护程序。此进程以前称为 **db2hmon**。
- **db2licc**，管理已安装的 DB2 许可证
- **db2thcln**，在 EDU 终止时重新启动资源（仅适用于 UNIX）
- **db2aiothr**，管理数据库分区的异步 I/O 请求（仅适用于 UNIX）
- **db2alarm**，在 EDU 请求的计时器到期时通知 EDU（仅适用于 UNIX）
- **db2sysc**，处理关键 DB2 服务器事件的主系统控制器 EDU

死锁

当两个应用程序彼此锁定对方所需的数据时就发生了死锁，这将导致两个应用程序都无法继续执行的情形。例如，在下图中，有两个应用程序正在同时运行: 应用程序 A 和应用程序 B。应用程序 A 的第一个步骤是更新表 1 的第一行，第二个步骤是更新表 2 的第二行。应用程序 B 首先更新表 2 的第二行，然后更新表 1 的第一行。在某个时间点 T1，应用程序 A 正在执行它的第一个步骤，锁定表 1 的第一行以更新此行。同时，应用程序 B 锁定表 2 的第二行以进行更新。在 T2，应用程序 A 尝试执行下一个步骤并请求锁定表 2 中的第二行以便进行更新。但是，同时应用程序 B 正在尝试锁定并更新表 1 中的第一行。由于应用程序 A 要等到它完成表 2 中第二行的更新后才释放对表 1 的第一行的锁定，而应用程序 B 要等待它锁定并更新表 1 的第一行后才释放对表 2 的第二行的锁定，这样就产生了死锁。应用程序会永远等待下去，直到“另一个”应用程序释放挂起数据上的锁定为止。

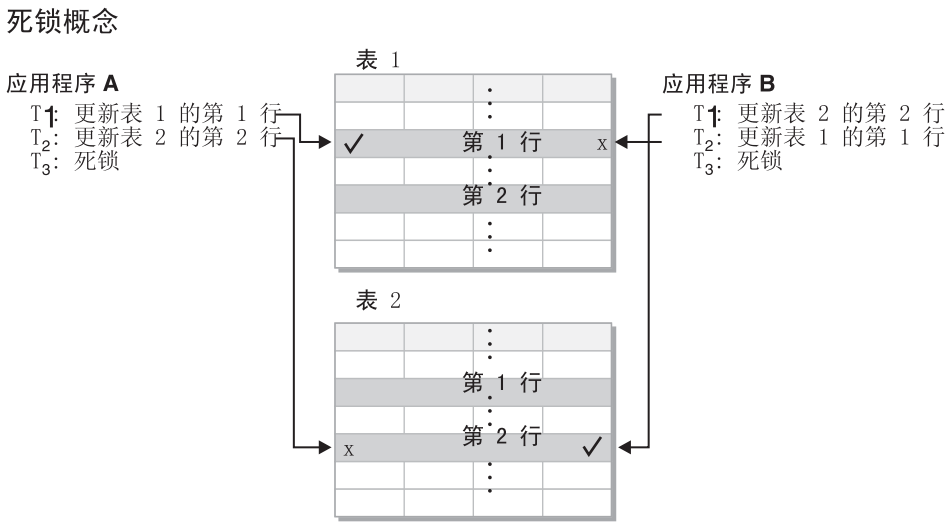


图 3. 应用程序之间的死锁

因为应用程序不会自动释放它们所需的数据的锁定，死锁检测器过程是中断死锁并允许应用程序进程继续所必需的。顾名思义，死锁检测器监视关于等待锁定的代理程序的信息，并在 *dlchktime* 配置参数所指定的时间间隔苏醒。

如果它发现死锁，那么死锁检测器任意选择一个已死锁的进程作为*牺牲进程*来回滚。牺牲进程被唤醒，并将 SQLCODE -911（SQLSTATE 40001）返回到调用应用程序，原因码为 2。数据库管理器自动回滚所选进程中未落实的事务。当回滚完成时，释放属于牺牲进程的锁定，而该死锁涉及的其他进程可以继续进行。

要确保良好的性能，为死锁检测器选择适当的时间间隔。过短的时间间隔导致不必要的开销，而太长的时间间隔使死锁将进程延迟一段不可接受的时间。例如，5 分钟的唤醒时间间隔使死锁几乎存在 5 分钟，这对于短事务处理来说看起来可是长时间。在解决死锁时的可能延迟与检测它们的开销两者之间进行平衡非常重要。

注:

1. 在分区数据库环境中，*dlchktime* 配置参数时间间隔仅在目录节点上适用。如果在分区数据库环境中检测到大量的死锁，增大 *dlchktime* 参数的值以解决锁定等待和通信等待。
2. 在分区数据库中，每个数据库分区将*锁定图*发送到包含系统目录视图的数据库分区。在此数据库分区上进行全局死锁检测。

当构造带有多个访问数据库的独立进程的应用程序以致很可能产生死锁时，出现另一个问题。例如，在一个应用程序中的数个进程访问同一个表，对该表进行读取及写入操作。如果这些进程执行只读 SQL 和 XQuery 查询，然后再对同一表执行 SQL 更新，那么由于各个进程间对同一数据潜在的争用使得死锁的机会增大。例如，如果两个进程读该表，然后更新该表，那么进程 A 试图获得对行的 X 锁定，而进程 B 对该行具有 S 锁定。为了避免发生这种死锁，访问具有修改意向的数据的应用程序应该执行下列其中一项操作:

- 执行选择操作时使用 FOR UPDATE OF 子句。此子句确保当进程 A 试图读取该数据时进行 U 锁定。将禁用行分块。
- 执行查询时使用 WITH RR USE AND KEEP UPDATE LOCKS 子句或 WITH RS USE AND KEEP UPDATE LOCKS 子句。任一子句都确保当进程 A 试图读取该数据时进行 U 锁定，并且允许行分块。

在创建数据库的同时，会创建详细死锁事件监视器。同其他监视器一样，这个事件监视器将造成一些开销。

要限制此事件监视器消耗的磁盘空间量，当它达到其输出文件的最大数目时，该事件监视器就取消激活，并且会有一条消息写入管理通知日志。将不再需要的输出文件除去即可在下次激活数据库时重新激活事件监视器。

如果不需要详细死锁事件监视器，那么可使用以下命令删除事件监视器:

```
DROP EVENT MONITOR db2detaildeadlock
```

在应用程序访问昵称的联合系统环境中，由于数据源处发生死锁而可能无法获取应用程序请求的数据。当发生这种情况时，DB2 依靠数据源的死锁处理工具。如果在多个数据源之间发生死锁，那么 DB2 依靠数据源超时机制解开死锁。

要记录有关死锁的更多信息，将数据库管理器配置参数 *diaglevel* 设置为 4。记录的信息包括已锁定的对象、锁定方式以及挂起有锁定的应用程序。还可能会记录当前动态 SQL 和 XQuery 语句或静态程序包名称。仅在 *diaglevel* 4 记录动态 SQL 和 XQuery 语句。

缺省死锁事件监视器

在创建数据库时，缺省情况下将创建并激活一个称为 DB2DETAILDEADLOCK 的死锁事件监视器。激活实例时就会自动启动该死锁事件监视器。当此监视器处于活动状态时，首次发生死锁时就会收集诊断信息，并允许调查发生死锁的原因，而不需要再现当时的情况。

要限制此事件监视器消耗的磁盘空间量，当它达到其输出文件的最大数目时，该事件监视器就取消激活，并且会有一条消息写入管理通知日志。将不再需要的输出文件除去即可在下次激活数据库时重新激活事件监视器。

此命令是使用以下语句来创建的:

```
db2 create event monitor db2detaildeadlock for deadlocks with details write to file
'db2detaildeadlock' maxfiles 20 maxfilesize 512 buffersize
17 blocked append autostart
```

WITH DETAILS 子句将提供诸如以下信息: 发生死锁时正在执行的语句; 如果 dbmon 堆中存在足够的内存，那么还将提供锁定列表。

将在数据库目录中的“db2event”目录下创建输出文件。如果您在创建数据库时未指定位置，那么可以通过查看数据库管理器配置参数 *dftdbpath* 来确定数据库目录的位置。例如，在 AIX® 系统上的样本数据库上，事件监视器输出文件可能位于以下目录中:
`NODE0000/SQL00001/db2event/db2detaildeadlock`

事件监视器将写入一个文件，最多可以有 20 个文件，每个文件的大小为 2M（512 个 4K 页）。达到 *maxfilesize*（2M）时，输出文件将关闭，然后打开一个新的输出文件。当创建的文件数达到 *maxfiles*（20）时，监视器就会将它自身关闭，并且会在管理通知日志中记录一条与以下消息相似的消息:

```
2004-12-01-22.58.24.968000 Instance: DB2 Node: 000 PID: 1116(db2syscs.exe) TID: 2540
Appid: *LOCAL.DB2.041202080328 database monitor sqm__evmgr::
log_ev_err Probe:2 Databse:XXX  ADM2001W The Event Monitor
"DB2DETAILDEADLOCK" was deactivated because the MAXFILES and
MAXFILES CREATE EVENT MONITOR parameters' limits have been reached.
```

将不再需要的输出文件除去即可在下次激活数据库时再次启动监视器。如不需要详细死锁事件监视器，可运行以下命令来删除:

```
DROP EVENT MONITOR db2detaildeadlock
```

如果您关心死锁的情况，那么请不要删除此监视器。

磁盘存储器概述

磁盘存储器性能因素

构成系统的硬件可能会影响系统的性能。作为硬件影响性能的示例，考虑一些与磁盘存储器相关的蕴含内容。

磁盘存储器管理的四个方面影响性能:

- **存储器的划分**

  如何在索引和数据之间以及表空间之间划分有限的存储器容量，在很大程度上决定了每一种资源在不同情况下的运行方式。

- **浪费的存储器**

  浪费的存储器本身可能不影响使用它的系统的性能，但浪费的存储器是一种可用于在别处改善性能的资源。

- **磁盘 I/O 的分布**

  如何良好地平衡几个磁盘存储设备和控制器上对磁盘 I/O 的需求，会影响数据库管理器从磁盘中检索信息的速度。

- **缺乏可用存储器**

  达到可用存储器限制后，可能会降低整体性能。

第 2 部分 表和索引

第 6 章 标准表的表和索引管理

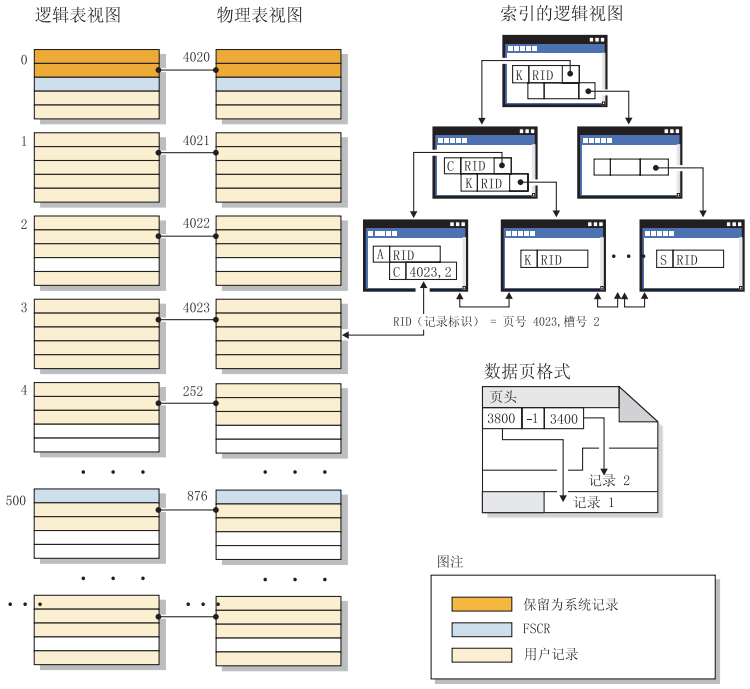


图 4. 标准表的逻辑表、记录和索引结构

在标准表中，数据在逻辑上是按数据页的列表来组织的。这些数据页根据表空间的扩展数据块大小在逻辑上分组在一起。例如，如果扩展数据块大小是 4，第 0 至 3 页是第一个扩展数据块的一部分，那么第 4 至 7 页是第二个扩展数据块的一部分，依此类推。

根据数据页大小以及记录大小的不同，每个数据页中所包含的记录数可能会有所变化。可以在表 1 中找到一个页面上可以包含的最大记录数。大多数页面只包含用户记录。但是，少数页包括特殊的内部记录，DB2 使用这些记录来管理表。例如，在标准表中，每个第 500 个数据页上都有一个“可用空间控制记录”（FSCR）。这些记录映射下面每 500 个数据页（直到下一 FSCR 为止）上的可供新记录使用的可用空间。当将记录插入表时，将使用这部分可用的可用空间。

在逻辑上，索引页组织成 B 树，这可以有效地在表中定位带有给定键值的记录。索引页上的项数不是固定的，但依赖于键的大小。对于 DMS 表空间中的表，索引页中的记录标识（RID）使用相对表空间页号，而不是对象相对页号。这使**索引扫描**能够直接访问数据页，而不需要“扩展数据块映像页”（EMP）来进行映射。

每个数据页都具有相同的格式。每个数据页开头都有一个页头。在页头后面，有一个槽目录。槽目录中的每一条目都与该页中的另一个记录相对应。该条目本身是数据页中记录开始位置的字节位移。值为 -1 的条目与已删除的记录相对应。

记录标识和页

记录标识（RID）由页号及随后的槽号组成。2 类索引记录还包含称为 ridFlag 的附加字段。该 ridFlag 存储有关索引中密钥状态的信息，例如，此密钥是否标记为已删除。在使用索引来标识 RID 之后，便使用该 RID 来到达正确的数据页和该页上正确的槽号。一旦对记录指定了 RID，在进行表重组之前，该 RID 便不会更改。

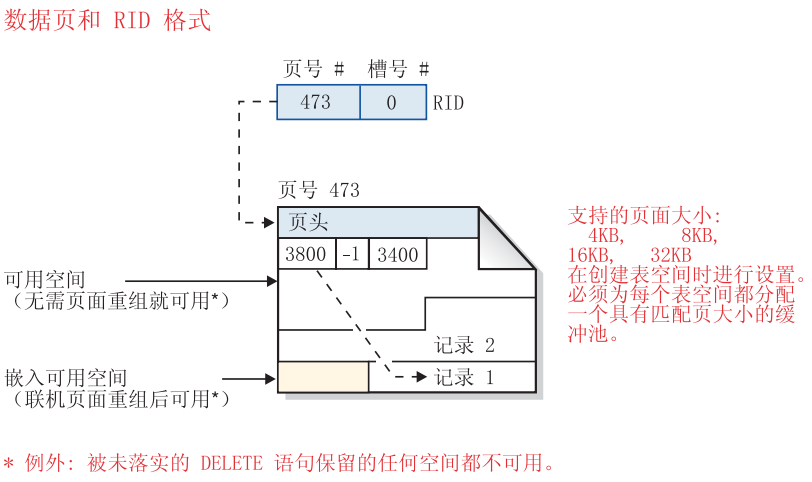


图 5. 数据页和记录标识（RID）格式

重组表时，实际删除记录后在页上留下的嵌入可用空间被转换成可使用的可用空间。根据记录在数据页上的移动重新定义 RID，以利用可使用的可用空间。

DB2 支持不同的页大小。对于有可能连续访问行的工作负载，请使用较大的页大小。例如，“决策支持”应用程序或大量使用临时表的场合使用的便是顺序访问。对于更有可能进行随机访问的工作负载，使用较小的页大小。例如，OLTP 环境中使用的便是随机访问。

标准表中的索引管理

DB2 索引使用最优的 B 树实现，它基于一种高效率且高并行性的索引管理方法，该方法使用预写记录。

最优 B 树实现的叶子页上带有双向指针，这使单个索引能支持正向或反向扫描。索引页分割通常刚好是对半的，但在高键页上除外，在那些页上，使用 90/10 分割。即，索引键的高 10% 放在新页上。这种类型的索引页分割对于特定工作负载而言非常有用，这些工作负载的 INSERT 请求通常使用新的高键完成。

从版本 8.1 开始，DB2 就使用 2 类索引。如果从 DB2 的较早版本进行迁移，那么将使用 1 类和 2 类索引，直到重组索引或执行将 1 类索引转换为 2 类索引的其他操作。索引类型决定如何从索引页除去删除键。

- 对于 1 类索引，键删除期间从索引页除去键，且当除去一页上的最后一个索引键时，便从索引中释放该页。
- 对于 2 类索引，仅当对表具有 X 锁定时，才在删除键期间从页除去索引键。如果键不能立即除去，那么标记为已删除而稍后除去。有关更多信息，参阅描述 2 类索引的节。

当创建索引时，如果已通过将 MINPCTUSED 子句设置为大于零的值来启用联机索引整理碎片时，那么可以联机合并叶子页。您指定的值是在索引叶子页上使用的空间的最小百分比的阈值。从索引页除去键之后，如果在页上使用的空间的百分比等于或小于所给的值，那么数据库管理员尝试将剩余键与相邻页上的键合并。如果有足够的空间，那么执行合并，并删除一个索引叶子页。联机索引整理碎片可以改进空间复用，但是如果 MINPCTUSED 值太高，那么尝试合并所花的时间会增加，并且成功的可能性更小。对此子句建议的值是百分之五十或更低。

注: 因为仅当从 2 类索引中的索引页除去键时联机整理碎片才发生，所以如果键仅仅标记为已删除但并未从页除去时，它不发生。

CREATE INDEX 语句的 INCLUDE 子句允许除了包括键列之外，还包括索引叶子页上的指定列。这会增加适合于纯索引访问的查询数。但是，这也会增加索引空间需求，并且，如果频繁地更新所包括的列，那么还可能会增加索引维护成本。更新包含列的维护成本小于更新键列，但大于索引中未显示的更新列。只使用键列，而不使用所包括的列来对索引 B 树进行排序。

第 7 章 MDC 表的表和索引管理

多维集群（MDC）表的表和索引组织基于与标准表组织相同的逻辑结构。与标准表类似，MDC 表是按页组织的，这些页包含分为许多列的数据行，每页上的行由行标识（RID）标识。但是，MDC 表的各页还分组为按扩展数据块大小计算的块。例如，下图显示扩展数据块大小为 4 的表，前四页（从 0 到 3 编号）是表中的第一个块。下一组页（从 4 到 7 编号）是表中的第二个块。

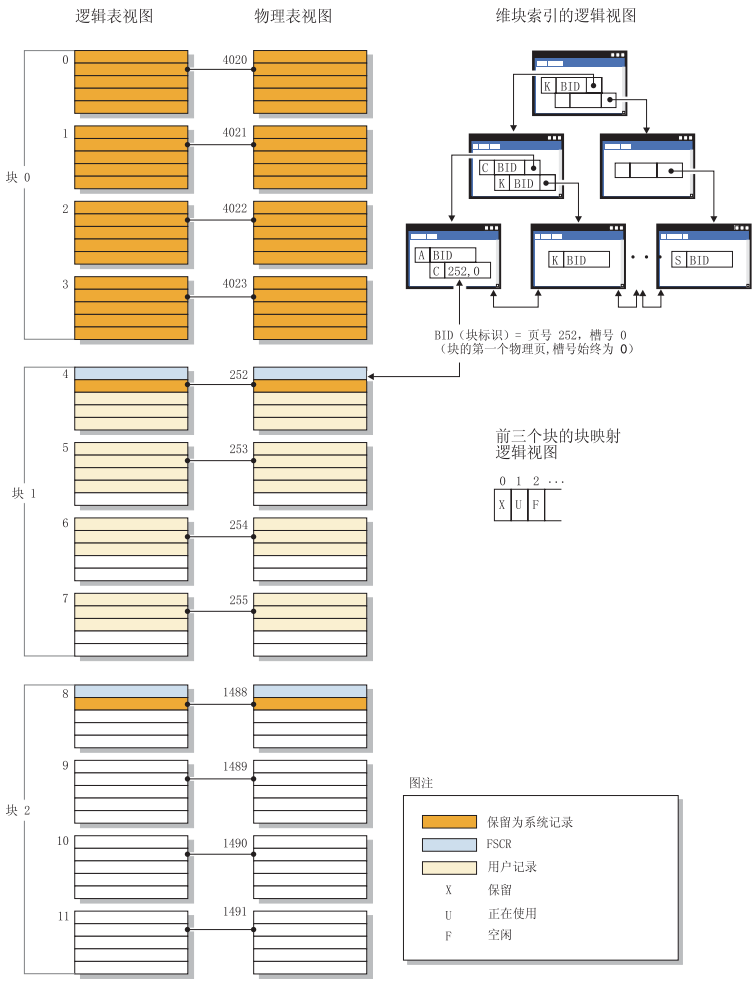


图 6. MDC 表的逻辑表、记录和索引结构

第一个块包含 DB2 用来管理表的特殊内部记录，包括可用空间控制记录（FSCR）。在后续块中，第一页包含 FSCR。FSCR 为块中每页上存在的新记录映射可用空间。当将记录插入表时，将使用这部分可用的可用空间。

顾名思义，MDC 表将对多个维上的数据进行集群。每个维都由您在 CREATE TABLE 语句的 ORGANIZE BY DIMENSIONS 子句中指定一列或一组列确定。当创建 MDC 表时，会自动创建以下两种索引:

- 维块索引，它包含指向单个维的每个已占用块的指针。
- 组合块索引，它包含所有维键列。组合块索引用于在插入和更新活动期间维护集群。

当优化器确定特定查询的最有效的访问方案时，它考虑利用维块索引的访问方案。当查询在维值上具有谓词时，优化器可以使用维块索引来标识包含这些值的扩展数据块，并从扩展数据块中进行访存。因为扩展数据块是磁盘中物理上相邻的页，所以这会导致更有效的性能并使 I/O 次数减少到最低。

另外，如果数据访问方案的分析指示特定 RID 索引将提高查询性能，那么可以创建这样的索引。

除了维块索引和组合块索引以外，MDC 表还维护包含位图的块映射，该位图指示每个块的可用性状态。在位图列表中编码下列属性:

- X（保留）： 第一个块仅包含表的系统信息。
- U（正在使用）： 此块已被使用并与维块索引相关联
- L（(已装入）： 已通过当前装入操作装入此块
- C（检查约束）： 此块由装入操作设置，用来指定装入期间的增量约束检查。
- T（刷新表）： 此块由装入操作设置，用来指定需要 AST 维护。
- F（空闲）： 如果没有设置其他属性，该块被认为是空闲的。

因为每个块都在块映射文件中具有一个条目，所以文件随表的增长而增长。作为单独对象存储此文件。在 SMS 表空间中，它是一个新的文件类型。在 DMS 表空间中，它在对象表中具有新的对象描述符。

第 8 章 MDC 表的异步索引清除

可以使用异步索引清除（AIC）来提高转出删除的性能。转出删除是一种从多维集群（MDC）表中删除符合条件的数据块的有效方法。AIC 是指在执行使索引条目失效的操作之后延迟清除索引。

在执行标准转出删除期间，将在执行删除操作时同步清除索引。对于包含许多记录标识（RID）索引的表，很大一部分删除时间花在除去索引键，索引键引用了被删除的表行。可以通过指定在落实删除之后就清除这些索引来提高转出速度。

要对 MDC 表利用 AIC，需要显式启用*延迟索引清除转出*机制。可以采用两种方法来指定延迟转出：将注册表变量 **DB2_MDC_ROLLOUT** 设置为 DEFER 以及发出 SET CURRENT MDC ROLLOUT MODE 语句。在执行延迟索引清除转出期间，各个块被标记为已转出，但是未更新 RID 索引，直到落实事务之后才会进行更新。在删除期间仍然会清除块标识（BID）索引，这是因为它们并不需要执行行级别处理。

在落实转出删除时将调用转出 AIC；如果数据库已关闭，那么在重新启动该数据库之后首次访问表时也会调用转出 AIC。在进行 AIC 时，对索引进行的所有查询（包括将访问被清除的索引的那些查询）都会执行。

每个 MDC 表都有一个协调清除程序。对于多个转出的索引清除都合并在清除程序中。清除程序将对每个 RID 索引产生一个清除代理程序，各个清除代理程序将并行更新 RID 索引。清除程序还与实用程序调速功能集成。缺省情况下，每个清除程序的实用程序影响优先级为 50（可接受的值为 1 到 100，0 表示无调速功能）。可以使用 SET UTIL_IMPACT_PRIORITY 命令或 db2UtilityControl API 来更改此优先级。

监视

因为在完成清除之后才能复用 MDC 表上已转出的块，所以监视延迟索引清除转出的进度将很有用。使用 LIST UTILITIES 监视器命令来显示被清除的每个索引的实用程序监视器条目。还可以通过延迟索引清除转出（BLOCKS_PENDING_CLEANUP）并使用 SYSPROC.ADMIN_GET_TAB_INFO_V95 表函数来查询当前正在清除的表中的块数。要查询数据库级别的 MDC 表块暂挂清除数，请使用 GET SNAPSHOT 命令。

在 LIST UTILITIES 命令的以下样本输出中，进度由每个索引中已被清除的页数指示。输出中列示的每个阶段都表示正在对表清除的一个 RID 索引。

```
db2 LIST UTILITIES SHOW DETAILS 输出。
标识                                  = 2
类型                                  = MDC ROLLOUT INDEX CLEANUP
数据库名称                            = WSDB
分区号                                = 0
描述                                  = TABLE.<schema_name>.<table_name>
开始时间                              = 06/12/2006 08:56:33.390158
状态                                  = 正在执行
调用类型                              = 自动
正在调速:
   优先级                             = 50
进度监控:
  估计已完成的百分比          = 83
   阶段号                             = 1
      描述                            = <schema_name>.<index_name>
      工作总计                        = 13 页
```

```
   已完成的工作                = 13 页
   开始时间                    = 06/12/2006 08:56:33.391566
阶段号                         = 2
   描述                        = <schema_name>.<index_name>
   工作总计                    = 13 页
   已完成的工作                = 13 页
   开始时间                    = 06/12/2006 08:56:33.391577
阶段号                         = 3
   描述                        = <schema_name>.<index_name>
   总计工作                    = 9 页
   已完成的工作                = 3 页
   开始时间                    = 06/12/2006 08:56:33.391587
```

第 9 章 索引结构

数据库管理器使用 B+ 树结构进行索引存储。一个 B+ 树有一层或多层，如下图中所示，其中，RID 表示行标识:

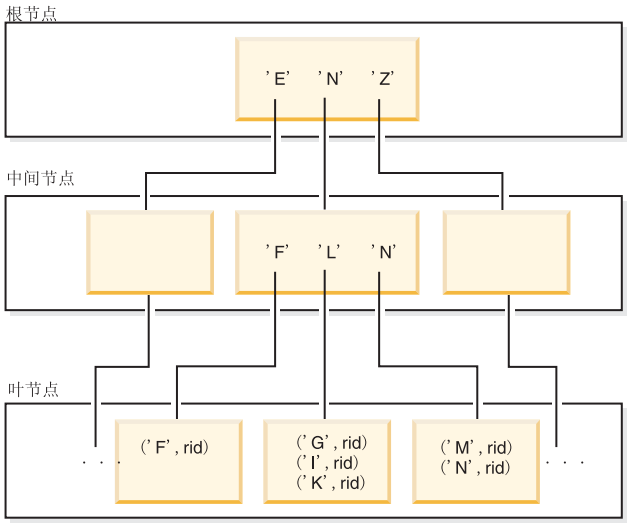


图 7. B+ 树结构

顶层称为*根节点*。底层由*叶节点*组成，底层存储了索引键值，并有一个指针指向包含键值的表中的行。根节点层和叶节点层之间的那些层称为*中间节点*。

当它查找特定的索引键值时，索引管理器会从根节点开始搜索该索引树。对于下一层的每个节点根都包含一个键。每个键的值是下一层中对应节点的最大现有键值。例如，如果一个索引有三层（如图中所示），那么，要查找一个索引键值，索引管理器搜索根节点，以查找大于或等于要查找的键的第一个键值。根节点键指向特定的中间节点。索引管理器遵循此过程遍历中间节点，直到它找到包含它需要的索引键的叶节点。

该图显示要查找的键是“I”。在根节点中大于或等于“I”的第一个键是“N”。它指向下一层的中间节点。在该中间节点中大于或等于“I”的第一个键是“L”。它指向发现有“I”的索引键及其对应的行标识的特定叶节点。行标识标识基本表中的对应行。叶节点层也可以包含指向上一层叶节点的指针。这些指针允许索引管理器在它找到范围内的某个值时按任一方向扫描叶节点以检索某个范围内的值。仅当使用 ALLOW REVERSE SCANS 子句创建了索引时，才可能有按任一方向扫描的能力。

对于多维集群（MDC）表，对给表指定的每个集群维自动创建块索引。也创建其他组合块索引，该索引包含每列中涉及表中任何维的键部分。这些索引包含指向块标识（BID）的指针代替 RID，并提供数据访问改进。

在 DB2 版本 8.1 和更高版本中，索引可以为 1 类或 2 类索引。*1 类索引*是较旧的索引样式。在较早版本的 DB2 中创建的索引就是这种索引。

2 类索引大于型 1 类索引并提供最小化的下一键锁定的功能部件。为 2 类索引的叶子页中每个 RID 存储的单字节 *ridFlag* 字节用来将 RID 标记为在逻辑上已删除，以便以后可以物理上将其除去。对于包含在索引中的每个变长列，一个附加字节存储列值的实际长度。因为某些键可以标记为删除，但尚未在物理上从索引页除去，所以 2 类索引也可以大于 1 类索引。在落实 DELETE 或 UPDATE 事务之后，可以清除标记为删除的键。

第 3 部分 进程

第 10 章 降低日志记录开销以提高查询性能

所有数据库都维护用于记录数据库更改情况的日志文件。有两种日志记录策略可供选择:

- 循环日志记录，指的是日志记录填充日志文件，然后覆盖初始日志文件中的初始日志记录。被覆盖的日志记录不可恢复。
- 保留日志记录，指的是日志文件由日志记录填满时会被存档。然后使用新的日志文件来存放日志记录。保留日志文件可以启用**前滚恢复**。前滚恢复根据日志中记录的已完成的工作单元（事务）来对数据库重新应用更改。您可以指定前滚恢复是恢复至日志末尾，还是恢复至日志末尾之前的特定时间点。

无论哪种日志记录策略，对一般数据和索引页所作的所有更改都会被写入日志缓冲区。日志缓冲区中的数据由记录器进程写入磁盘。在下列情况下，查询处理必须等待日志数据写入磁盘后才能进行:

- 运行 COMMIT 语句时
- 在将相应数据页写入磁盘之前，因为 DB2 使用预写记录。预写日志记录的好处是当执行 COMMIT 语句完成事务之后，并非所有更改的数据和索引页都需要写入磁盘。
- 在更改元数据（大多数是通过执行 DDL 语句产生的）之前
- 在将日志记录写入日志缓冲区时，如果日志缓冲区已满

DB2 以这种方法管理向磁盘写入日志数据的目的是尽可能地缩短处理延迟时间。在发生许多较小的并发事务的环境中，大多数处理延迟是由 COMMIT 语句造成的，因为此语句必须等待日志数据写入磁盘后才能进行。因此，记录器进程频繁地将少量日志数据写入磁盘会造成大量处理延迟，另外一些延迟是由日志 I/O 开销造成的。为平衡此类日志记录延迟导致的应用程序响应时间，请将 *mincommit* 数据库配置参数设置为大于 1 的值。虽然这样设置可能会对某些应用程序的 COMMIT 命令造成更长的延迟时间，但是这会使系统可在一次操作中写入更多的日志数据。

对大对象（LOB）和 LONG VARCHAR 的更改通过影子页面调度进行跟踪。除非指定日志保留，并且在使用 CREATE TABLE 语句创建表时没有使用 NOT LOGGED 子句定义 LOB 列，否则不记录 LOB 列更改。如同一般数据页一样，对 LONG 或 LOB 数据类型的分配页的更改也会被记录下来。

第 11 章 提高插入性能

当 SQL 语句使用 INSERT 来将新信息放置在表中时，INSERT 搜索算法首先搜索“可用空间控制记录”（FSCR）来查找具有足够空间的页。然而，即使 FSCR 指示某页具有足够的可用空间，该空间也可能因为被另一个事务的未落实的 DELETE 语句保留而不可用。要确保未落实的可用空间是可用的，应经常“落实”事务。

DB2MAXFSCRSEARCH 注册表变量的设置确定表中为 INSERT 而搜索的 FSCR 数目。此注册表变量的缺省值是 5。如果在指定数目的 FSCR 中找不到空间，在表的末尾处追加插入的记录。要优化 INSERT 速度，还可能会将后续记录追加至表末尾，直到填满了两个扩展数据块为止。在填满了两个扩展数据块之后，下一个 INSERT 在上一搜索结束处的 FSCR 继续搜索。

注：要优化 INSERT 速度（可能的代价是表增长得更加快），将 DB2MAXFSCRSEARCH 注册表变量设置为较小的数。要优化空间复用（可能的代价是 INSERT 速度减慢），将 DB2MAXFSCRSEARCH 设置为更大的数。

以这种方式搜索整个表中的所有 FSCR 后，追加要插入的记录而不进行附加搜索。另外，在表中某处（例如，在 DELETE 之后）创建空间之前，也不再执行使用 FSCR 的搜索。

有两个其他的 INSERT 算法选项，如下所示:

- APPEND MODE

  在此方式中，总是将新行追加至表末尾。不执行 FSCR 的搜索或维护。使用 ALTER TABLE APPEND ON 语句启用此选项，它可以提高只会增长的表（例如，日志）的性能。
- 在表上定义集群索引。

  在这种情况下，数据库管理器尝试将记录插入到带有类似索引键值的其他记录所在的页上。如果该页上没有空间，那么尝试将该记录放到周围的页中。如果仍不成功，那么使用如上所描述的 FSCR 搜索算法，除非使用最差匹配方法，而不是使用首先匹配方法。这种最差匹配方法往往能选择带有更多可用空间的页。此方法为带有此键值的行建立新的集群区。

  当对表定义集群索引时，请在装入或重组表之前使用 ALTER TABLE... PCTFREE。PCTFREE 子句指定在装入并重组之后应该留在表的数据页上的可用空间的百分比。这提高了集群索引操作能在适当的页上找到可用空间的可能性。

第 12 章 更新处理

当代理程序更新页时，数据库管理器使用以下协议来将事务必需的 I/O 最小化，并确保可恢复性。

1. 使用互斥锁定将要更新的页锁定住。将描述如何重做和撤销更改的日志记录写入日志缓冲区。作为此操作的一部分，获取日志序号（LSN），并将其存储在正在更新的页的页头中。
2. 对该页进行更改。
3. 取消锁定该页。

   因为尚未将对该页所作的更改写入磁盘，所以认为该页是“脏”页。
4. 日志缓冲区已更新。

   将日志缓冲区中的数据和“脏”数据页强制写入磁盘。

为了提高性能，这些 I/O 被延迟至一个方便的时间（例如，在系统装入时的暂停期间），或延迟至必须确保可恢复性的时间，或延迟至限制的恢复时间。具体地说，“脏”页在下面这些时候被强制写入磁盘:

- 当另一个代理程序选择它作为牺牲页的时候。
- 当由于下列原因造成页清除程序对页执行操作时:
  - 另一个代理程序选择它作为牺牲页。
  - 超过 *chngpgs_thresh* 数据库配置参数百分比值。当超过此值时，异步页清除程序便被唤醒，并将更改过的页写入磁盘。

    如果启用了主动清除页，那么此值不合适，并且不会触发页清除。
  - 超过 *softmax* 数据库配置参数百分比值。一旦超过此值，异步页清除程序便被唤醒，并将更改过的页写入磁盘。

    如果对数据库启用了主动清除页，并且已经为数据库正确配置了页清除程序数，那么始终都不会超过该值。
  - 憎恨列表中干净页的数目下降得太低。只有采用主动清除页方法时页清除程序才对此情况起作用。
  - 脏页当前添加至或者计划添加至 LSNGAP 条件。只有采用主动清除页方法时页清除程序才对此情况起作用。
- 当作为调用了 NOT LOGGED INITIALLY 子句并发出了 COMMIT 语句的表的一部分更新该页时。当执行 COMMIT 语句时，所有已更改的页都被清仓至磁盘，以确保可恢复性。

第 13 章 客户机/服务器处理模型

本地和远程应用程序进程可以使用同一数据库。远程应用程序是从作为数据库机器的远程机器启用数据库操作的应用程序。本地应用程序直接与服务器机器上的数据库相连。

DB2 管理客户机连接的方式取决于连接集中器是打开还是关闭。当 *max_connections* 数据库管理器配置参数的设置值大于 *max_coordagents* 配置参数的设置值时，连接集中器处于打开状态。

- 如果连接集中器处于“关闭”状态，每个客户机应用程序均会分配一个称为*协调代理程序*的唯一 EDU，它可以协调该应用程序的处理并与之通信。
- 如果连接集中器处于“打开”状态，每个协调代理程序均可一次管理多个客户机连接，并可协调其他工作程序代理程序以完成这项工作。对于具有许多相关瞬态连接的因特网应用程序或具有许多相关小事务的类似应用程序，连接集中器通过允许进行更多的客户机应用程序连接来提高系统的性能。它还可以减少每个连接对系统资源的使用。

下图的 DB2 服务器中的每一个圈都表示引擎可调度单元（EDU），EDU 是使用操作系统线程实现的。

在可以执行应用程序要对数据库执行的工作之前，必须在应用程序与数据库管理器之间建立一种通信方法。

在下图中的 A1 处，本地客户机首先通过 db2ipccm 建立通信。在 A2 处，db2ipccm 使用 db2agent EDU，该 EDU 成为来自本地客户机的应用程序请求的协调代理程序。然后该协调代理程序联系 A3 处的客户机应用程序以在客户机应用程序和协调程序之间建立共享内存通信。本地客户机上的应用程序与 A4 处的数据库相连。

在下图中的 B1 处，远程客户机通过 db2tcpcm EDU 建立通信。如果选择其他任何通信协议，那么使用相应的通信管理器。db2tcpcm EDU 在客户机应用程序和 db2tcpcm 之间建立 TCP/IP 通信。然后，它与 B2 处的 db2agent（此时 db2agent 成为该应用程序的协调代理程序）一起工作并将连接传送至此代理程序。在 B4 处，协调代理程序与远程客户机应用程序联系，然后连接至 B5 处的数据库。

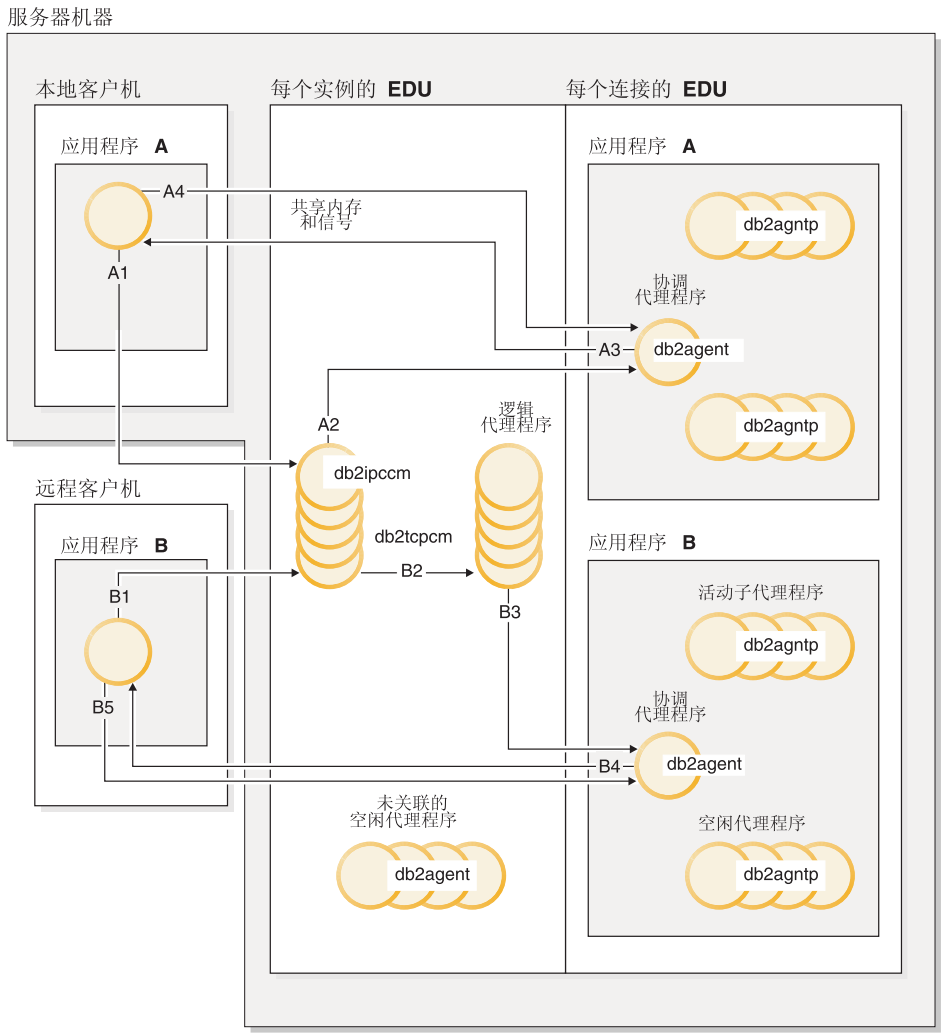


图 8. 进程技术模型概述

在此图中要注意的其他事项:

- 工作程序代理程序执行应用程序请求。
- 工作程序代理程序共有四种: 活动协调代理程序、活动子代理程序、关联子代理程序和空闲代理程序。
- 每个客户机连接均与活动协调代理程序相链接。
- 在分区数据库环境以及启用的分区内并行性环境中，协调代理程序将数据库请求分配给子代理程序（db2agntp）。子代理程序为应用程序执行请求。
- 有一个代理程序池（db2agent），空闲代理程序和池中代理程序在那里等待新工作。
- 其他 EDU 管理客户机连接、日志、两阶段落实（COMMIT）、备份和复原任务以及其他任务。

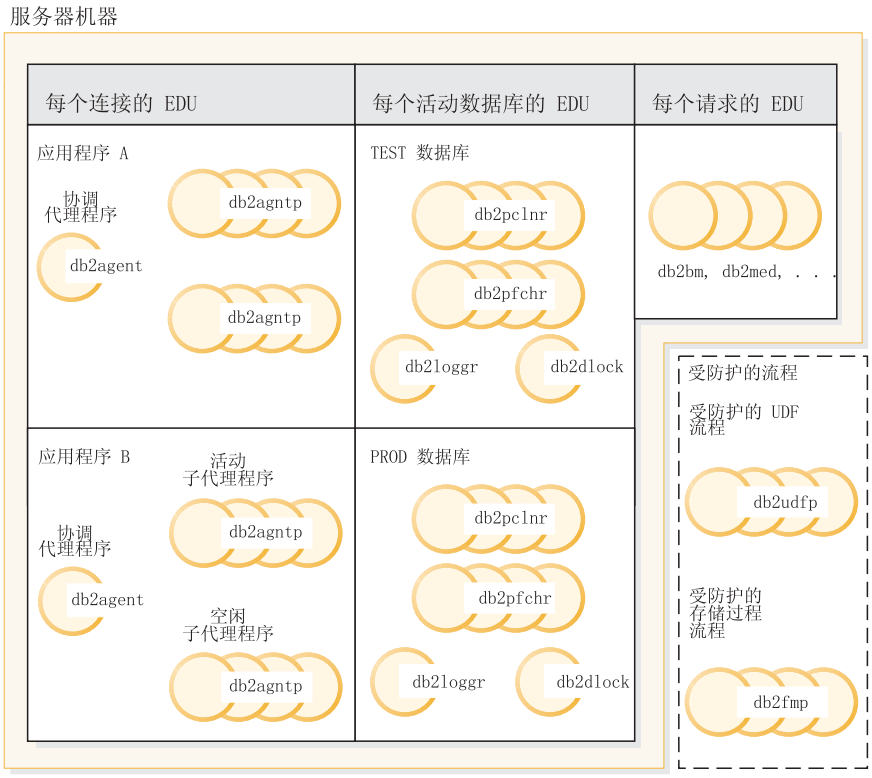


图 9. 进程技术模型（第二部分）

此图显示了作为服务器机器环境一部分的附加引擎可调度单元（EDU）。每个活动数据库均有自身的预取程序（db2pfchr）和页清除程序（db2pclnr）的共享池，并有自身的记录器（db2loggr）和死锁检测器（db2dlock）。

对受防护的用户定义的函数（UDF）和存储过程（图中并未显示）进行管理以尽可能减少与其创建和删除相关联的开销。*keepfenced* 数据库管理器配置参数的缺省值为“YES”，此值使下一次调用的存储过程可以重复使用存储过程进程。

注：为获得更好的性能，不受防护的 UDF 和存储过程直接在代理程序的地址空间运行。但是，由于它们对代理程序的地址空间的访问权不受限制，因此在使用它们之前，需要对它们进行严格的测试。

多数据库分区处理模型是单一数据库分区处理模型的逻辑扩展。实际上，单一公共代码库支持这两种方式的操作。下图显示了单一数据库分区处理模型（如前两图中所示）与多数据库分区处理模型之间存在的相似性与差异。

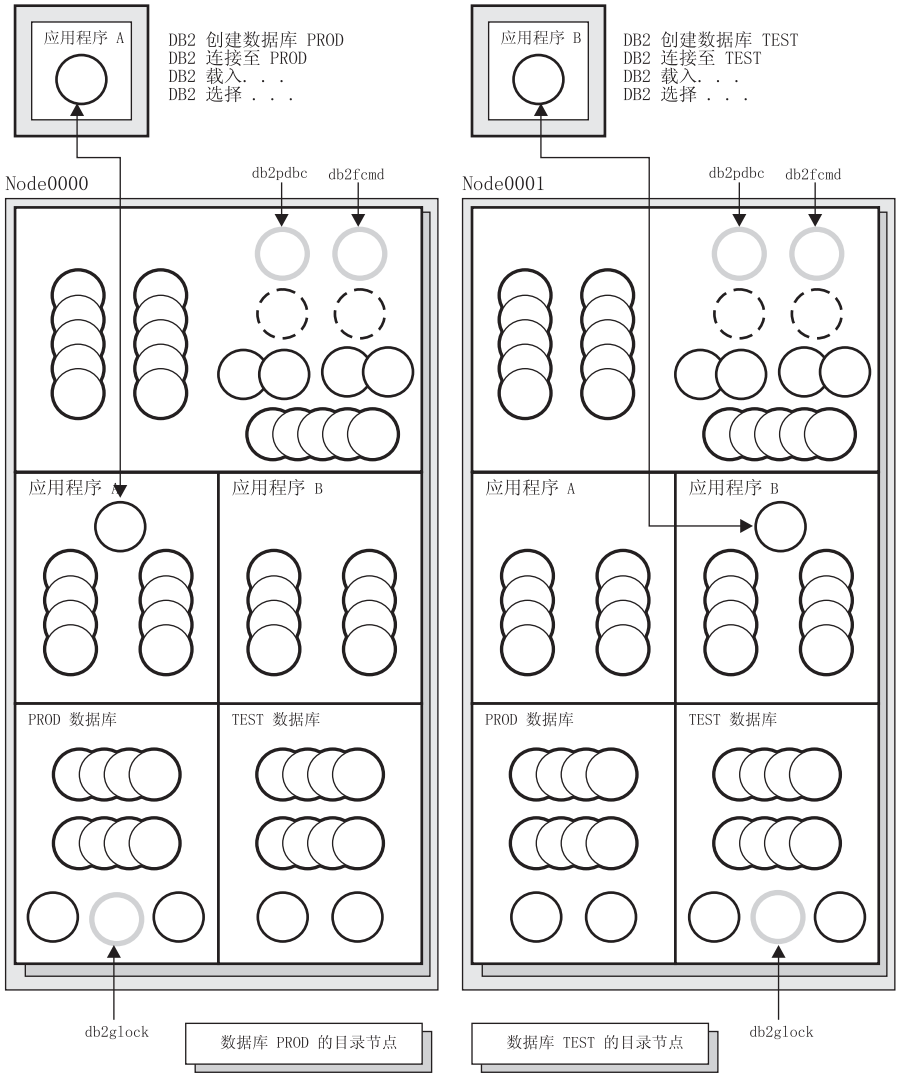


图 10. 进程技术模型和多数据库分区

在单一数据库分区处理模型与多数据库分区处理模型之间，大多数引擎可调度单元（EDU）是相同的。

在多数据库分区（或节点）环境中，其中一个数据库分区为目录节点。该目录存放与数据库中的对象相关的所有信息。

如上图所示，由于应用程序 A 在 Node0000 上创建 PROD 数据库，因此在该节点上创建 PROD 数据库的目录。同样，由于应用程序 B 在 Node0001 上创建 TEST 数据库，因此在该节点上创建 TEST 数据库的目录。您可能想在您的系统环境中的不同节点上创建数据库，从而将与每个数据库的目录相关的多余活动平均分布在这些节点上。

在多分区数据库环境中的每个节点上，可以找到与实例相关联的附加 EDU（db2pdbc 和 db2fcmd）。这些 EDU 是在数据库分区之间协调请求以及启用“快速通信管理器”（FCM）所需的。

还有一个附加的 EDU（db2glock）与数据库的目录节点相关联。该 EDU 可以控制活动数据库所在的各个节点上的全局死锁。

每个来自应用程序的 CONNECT 都由与协调代理程序相关联的连接表示，以便对该连接进行处理。*协调代理程序*是指与应用程序相互通信（即接收请求和发送回应）的代理程序。它自身可以处理请求，也可以协调多个子代理程序对请求进行处理。协调代理程序所在的数据库分区称为该应用程序的*协调程序节点*。协调程序节点还可以使用 SET CLIENT CONNECT_NODE 命令进行设置。

协调程序节点将来自应用程序的部分数据库请求发送至其他数据库分区中的子代理程序；然后协调程序节点将其他数据库分区返回的所有结果合并，并发送回给应用程序。

发出 CREATE DATABASE 命令的数据库分区称为该数据库的“目录节点”。目录表就存储在此数据库分区上。通常，所有用户表都分布在一组节点上。

注: 可以配置任意数目的数据库分区来在同一机器上运行。这称为“多逻辑分区”或“多逻辑节点”配置。在带有非常大容量的主存储器的大型对称多处理器（SMP）机器上，这样的配置非常有用。在这种环境中，可以优化数据库分区之间的通信，以使用共享内存和信号。

第 4 部分 性能调整的快速启动技巧

当启动 DB2 的新实例时，应考虑有关基本配置的下列建议:

- 使用“控制中心”的“配置顾问程序”以获取有关系统的合理初始缺省值的建议。应当为唯一硬件环境调整与 DB2 一起交付的缺省值。

  收集有关站点硬件的信息，以便可以回答向导问题。您可以立即应用建议的配置参数设置，或让该向导基于您的回答创建脚本，并在以后运行该脚本。

  此脚本也提供最一般的调整参数的列表以供以后参考。
- 使用“控制中心”和“客户机配置助手”中的其他向导来执行与性能相关的管理任务。这些任务通常是通过花费较少时间和努力就可获得重要的性能提高的任务。

  其他向导有助于提高个别表和一般数据访问的性能。这些向导包括: “创建数据库”、“创建表、索引”和“配置多站点更新”向导。“运行状况中心”提供一组监视和调整工具。
- 使用控制中心中的“设计顾问程序”或者使用 `db2advis` 命令来了解哪些索引、具体化查询表、多维集群表和数据库分区将提高查询性能。
- 使用 ACTIVATE DATABASE 命令来启动数据库。在分区数据库中，此命令激活所有数据库分区上的数据库，并避免当第一个应用程序进行连接时初始化数据库所必需的启动时间。

  注: 如果使用 ACTIVATE DATABASE 命令，必须用 DEACTIVATE DATABASE 命令关闭数据库。与数据库断开连接的最后一个应用程序不会将它关闭。
- 查阅总结表，该表列示并简要地描述了可用于数据库管理器和每个数据库的每个配置参数。

  这些总结表包含一个列，该列指示调整参数是导致性能高、中、低还是无更改，是更好还是更坏。使用此表来查找您可以调整以获取最大性能提高的参数。

第 14 章 操作性能

DB2 中的内存分配

在 DB2 中，内存分配和释放在各个时间进行。当发生指定事件（例如，应用程序连接时）时可以将内存分配给特定内存区，或者可以根据配置参数设置中的更改将其释放。

下图显示了数据库管理器为不同用途分配的各个内存区和允许控制此内存大小的配置参数。请注意，在包括多个逻辑数据库分区的企业服务器版环境中，每个数据库分区都有它自己的“数据库管理器共享内存”集。

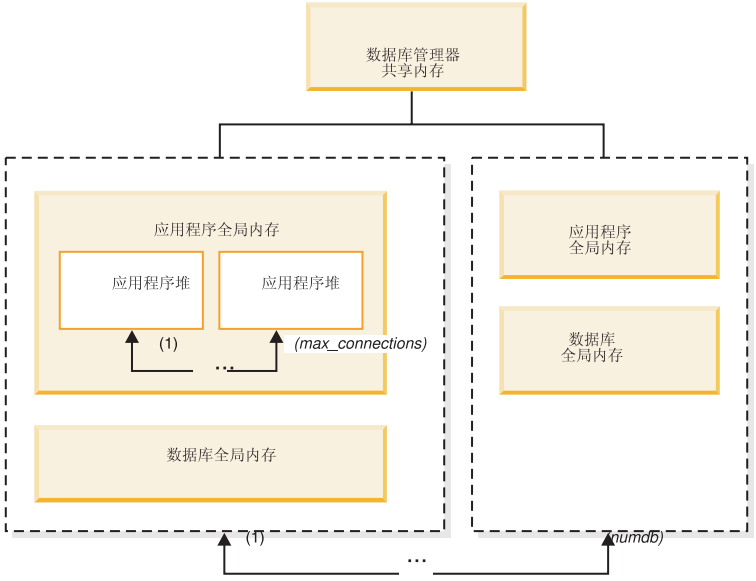


图 11. 数据库管理器所用的内存类型

在发生下列事件时，将为数据库管理器的每个实例分配内存:

- **当启动数据库管理器（db2start）时**: 分配数据库管理器全局共享内存（也称为实例共享内存），并且会一直保持此分配状态，直到数据库管理器停止运行（db2stop）为止。此区域中包含数据库管理器在管理所有数据库连接上的活动时所需的信息。DB2 将自动控制数据库管理器全局共享内存大小。
- **当第一次激活数据库或与数据库连接时**: 将分配数据库全局内存。所有与数据库连接的应用程序均使用数据库全局内存。数据库全局内存的大小由 **database_memory** 配置参数指定。缺省情况下，此参数设置为 automatic，允许 DB2 计算为数据库分配的初始内存量，并在运行时根据数据库的需要来自动配置数据库内存大小。可以设置 **database_memory** 来分配大于最初所需的内存，以便以后可以动态地分配附加内存。

可以动态地调整下列内存区，例如，减少分配给一个内存区的内存和增大另一个内存区中的内存。

- 缓冲池（使用 ALTER BUFFERPOOL DDL 语句）
- 数据库堆（包括日志缓冲区）
- 实用程序堆
- 程序包高速缓存
- 目录高速缓存
- 锁定列表（只能动态地增加此内存区，不能减少。）

在已启用数据库管理器分区内并行性配置参数（intra_parallel）的环境、已启用连接集中器的环境或者已启用数据库分区功能（DPF）的环境中，还可以分配共享排序堆作为数据库全局内存的一部分。另外，如果数据库管理器配置参数 **sheapthres** 设置为 0（缺省值），那么所有排序都将使用数据库全局内存。

- **当应用程序连接至数据库时**：分配了应用程序堆。每个应用程序都具有它自己的应用程序堆。如果需要的话，可以使用 **applheapsz** 配置参数来限制任何一个应用程序可以分配的内存量，或者使用 **appl_memory** 配置参数来限制消耗的应用程序内存总量。

  数据库管理器配置参数 **max_connections** 设置可以与实例连接或者与该实例中存在的任何数据库连接的应用程序数的上限。因为每个与数据库连接的应用程序都涉及到分配一些内存，所以允许大量并发应用程序可能将使用更多内存。
- **创建代理程序的时间**：当出现连接请求或并行环境中出现新的 SQL 请求时，系统会指定一个代理程序并为其分配代理程序专用内存。为该代理程序分配代理程序专用内存，其中包含仅用于该特定代理程序的内存，例如，专用排序堆。

此外，图中还指定了下列配置参数设置，它们用于限制为每种类型的内存区分配的内存大小。请注意，在分区数据库环境中，将在每个数据库分区中分配此类内存。

- **numdb**

  此参数指定不同应用程序可以使用的并发活动数据库的最大数目。由于每个数据库都有其自身的全局内存区，因此在增大此参数的值时，所分配的内存容量也可能会增加。
- **maxappls**

  此参数指定一个数据库可以同时连接的应用程序的最大数目。它将影响可能为该数据库分配的代理程序专用内存容量和应用程序全局内存容量。请注意，对于每个数据库，可将此参数设置为不同的值。

  另外两个需要考虑的参数是 **max_coordagents** 和 **max_connections**，它们都在实例级别应用（对 DPF 实例上的每个节点）。
- **max_connections**

  此参数用于限制任何时候可访问 DB2 服务器的连接数或实例连接数（对 DPF 实例上的每个节点）。
- **max_coordagents**

此参数用于限制一个实例中的所有活动数据库中可以同时存在的数据库管理器协调代理程序数（对 DPF 实例上的每个节点）。与 **maxappls** 和 **max_connections** 一起，此参数可以限制为代理程序专用内存和应用程序全局内存分配的内存容量。

db2mtrk 命令调用的内存跟踪程序允许您查看实例中的当前内存分配，包括有关每个内存池的下列类型的信息:

- 当前大小
- 最大大小（硬性限制）
- 最大大小（高水位标记）

数据库管理器共享内存

数据库管理器内存组织成若干个不同的内存区。下图显示了如何分配数据库管理器内存。显示的配置参数允许您控制各个内存区的大小。

图 12. 数据库管理器如何使用内存

审计缓冲区

此内存区用于数据库审计活动。此缓冲区大小由 **audit_buf_sz** 配置参数确定。

监视器堆

此内存区用于数据库系统监视的数据。此内存区大小由 **mon_heap_sz** 配置参数确定。

快速通信管理器（FCM）缓冲池

对于分区数据库系统，快速通信管理器（FCM）需要足够多的内存空间，尤其是在 **fcm_num_buffers** 的值较大时。从 FCM 缓冲池中分配需要的 FCM 内存。

数据库全局内存

“数据库全局内存”受下列配置参数影响:

- **database_memory** 参数为数据库全局内存的大小提供了一个下限。
- 下参数或因子控制数据库全局内存区的大小:
  - 缓冲池的大小。
  - 锁定列表的最大存储器（**locklist**）
  - 数据库堆（**dbheap**）
  - 实用程序堆大小（**util_heap_sz**）
  - 程序包高速缓存大小（**pckcachesz**）
  - 共享排序堆（**sheapthres_shr**）
  - 目录高速缓存（**catalogcache_sz**）

应用程序全局内存

应用程序全局内存可由 **appl_memory** 配置参数控制。可使用下列配置参数来限制任何一个应用程序可消耗的内存量:

- 应用程序堆大小（**applheaps**）
- 语句堆大小（**stmtheap**）
- 统计信息堆大小（**stat_heap_sz**）

代理程序专用内存

- 每个代理程序都需要它自己的专用内存区。DB2 服务器将根据需要创建任意个代理程序，并接受给定的已配置内存资源。可使用 **max_coordagents** 参数来控制最大协调代理程序数。
- 每个代理程序的专用内存区的最大大小由下列参数的值限制:
  - 专用排序堆大小（**sheapthres** and **sortheap**）
  - 代理程序堆栈大小（**agent_stack_sz**）

代理程序/应用程序共享内存

- 本地客户机的代理程序/应用程序共享内存段的总数受下列数据库配置参数中的较低者限制:
  - 所有活动数据库的 **maxappls** 的总数
  - **max_coordagents** 的值。

  注: 在启用了引擎集合的配置中（**max_connections** > **max_coordagents**），应用程序内存消耗将受 **max_connections** 限制。
- “代理程序/应用程序共享内存”还受下列数据库配置参数影响:
  - 应用程序支持层堆大小（**aslheapsz**）参数
  - 客户机 I/O 块大小（**rqrioblk**）参数

FCM 缓冲池和内存要求

在分区数据库系统中，数据库管理器共享内存和 FCM 缓冲池如下所示。

图 13. 使用多个逻辑节点时的 FCM 缓冲池

每个数据库分区的 FCM 缓冲区数由 *fcm_num_buffers* 配置参数控制。缺省情况下，此参数设置为 AUTOMATIC。要手动调整此参数，使用 buff_free（当前可用的缓冲区）和 buff_free_bottom（最小可用缓冲区系统监视元素）中的数据。

每个数据库分区的 FCM 信道数由 *fcm_num_channels* 配置参数控制。缺省情况下，此参数设置为 AUTOMATIC。要手动调整此参数，使用 ch_free（当前可用的信道）和 ch_free_bottom（最小可用信道系统监视元素）中的数据。

调整内存分配参数

设置内存分配参数的首要原则是：除非经过慎重分析，否则切勿将这些参数设置为最大值。此原则甚至还适用于具有最大内存的系统。许多影响内存的参数可使数据库管理器能够方便快捷地占用计算机上的所有可用内存。此外，在管理大量内存时，可能给数据库管理器部件带来额外的工作，因而产生更多开销。

某些 UNIX 操作系统在一个进程分配内存时（而不是在进程导出页至交换空间时）分配交换空间。对于这些系统，请确保为其提供与总共享内存空间相同的调页空间。

对于大多数配置参数，仅在需要时才会落实内存，并且参数设置确定特定内存堆的最大大小。但是，在下列情况下，将分配参数指定的全部内存容量：

- 锁定列表的最大存储器（*locklist*）
- 应用程序支持层堆大小（*aslheapsz*）
- FCM 缓冲器的数量（*fcm_num_buffers*）
- FCM 信道的数量（*fcm_num_channels*）
- 缓冲池

注意：

- 基准程序测试可以提供有关为内存参数设置适当值的最佳信息。在基准程序测试中，将对服务器运行典型的和最坏情况下的 SQL 语句，同时修改这些参数的值，直至找到性能开始下降的那一点。如果用图形表示性能与参数值的比率，那么曲线开

始到达水平稳定期或开始下降的那一点将指示在此点上分配其他内存并不会增加应用程序性能的值，因此只是浪费内存而已。

- 几个参数的内存分配上限可能超出了现有硬件和操作系统的内存容量。这些限制允许用于将来的内存增长。
- 要了解参数的有效范围，请参阅每个参数的详细信息。

自调整内存概述

自调整内存功能通过自动设置内存配置参数值以及调整缓冲池大小来简化内存配置任务。启用此功能之后，内存调整器就会在多个内存使用者（包括排序内存、程序包高速缓存、锁定列表内存和缓冲池）之间动态分配可用内存资源。

下表按类别列示了自调整内存主题:

表 1. 自调整内存信息概述

类别	相关主题
一般信息和限制	• 『自调整内存』 • 第 58 页的『自调整内存操作的详细信息和局限性』 • self_tuning_mem-self_tuning_mem - 自调整内存配置参数
启用和禁用	• 第 56 页的『启用自调整内存功能』 • 第 57 页的『禁用自调整内存功能』
监视	• 第 57 页的『确定启用了自调整功能的内存使用者』
DPF 注意事项	• 第 59 页的『分区数据库环境中的自调整内存』 • 第 60 页的『在分区数据库环境中使用自调整内存』
可自动调整的配置参数	• *《配置参数参考》*中的『database_memory - 数据库共享内存大小』 • *《配置参数参考》*中的『locklist - 锁定列表的最大存储量』 • *《配置参数参考》*中的『maxlocks - 升级之前锁定列表的最大百分比』 • *《配置参数参考》*中的『程序包高速缓存大小』 • *《配置参数参考》*中的『sheapthres_shr - 共享排序的排序堆阈值』 • *《配置参数参考》*中的『sortheap - 排序堆大小』

自调整内存

从 DB2 版本 9 开始，新的内存调整功能会通过自动设置一些内存配置参数的值来简化内存配置任务。启用此功能之后，内存调整器就会在下列内存使用者之间动态分配可用内存资源: 缓冲池、程序包高速缓存、锁定内存和排序内存。

调整器在 **database_memory** 配置参数定义的内存限制范围内工作。**database_memory** 值本身也可以自动调整。对 **database_memory** 启用自动调整（将它设置为 AUTOMATIC）之后，调整器将确定数据库的整体内存需求并根据当前数据库需求来增加或减少分配给数据库共享内存的内存量。例如，如果当前数据库需求很高，并且系统上有足够的可用内存，那么数据库共享内存将消耗较多的内存。如果数据库内存要求下降，或者系统上的可用内存量变得过低，就会释放一些数据库共享内存。

如果您未对自调整启用 **database_memory** 参数（未将此参数设置为 AUTOMATIC），那么整个数据库都将使用您为此参数指定的内存量，从而根据需要在数据库内存使用者之间分配内存。可以通过两种方法指定数据库使用的内存量：将 **database_memory** 设置为一个数值或者将它设置为 COMPUTED。在第二种情况下，总内存量是根据数据库启动时的数据库内存堆初始值总计而计算的。

除了使用 **database_memory** 配置参数来调整数据库共享内存以外，您还可以使其他内存使用者进行自调整，如下所示:

- 对于缓冲池，使用 ALTER BUFFERPOOL 和 CREATE BUFFERPOOL 语句。
- 对于程序包高速缓存，使用 **pckcachesz** 配置参数。
- 对于锁定内存，使用 **locklist** 和 **maxlocks** 配置参数。
- 对于排序内存，使用 **sheapthres_shr** 和 **sortheap** 配置参数。

启用自调整内存功能

自调整内存功能通过自动设置内存配置参数值以及调整缓冲池大小来简化内存配置任务。启用此功能后，内存调整器就会在几个内存使用者（包括排序、程序包高速缓存、锁定列表区域和缓冲池）之间动态地分配可用内存资源。

1. 通过将 *self_tuning_mem* 设置为 ON 来对数据库启用自调整功能。可以使用 UPDATE DATABASE CONFIGURATION 命令、SQLFUPD API 或通过控制中心中的**更改数据库配置参数**窗口来将 *self_tuning_mem* 设置为 ON。
2. 要对由内存配置参数控制的内存区域启用自调整功能，请使用 UPDATE DATABASE CONFIGURATION 命令、SQLFUPD API 或通过控制中心中的**更改数据库配置参数**窗口将相关配置参数设置为 AUTOMATIC。
3. 要对缓冲池启用自调整功能，请将缓冲池大小设置为 AUTOMATIC。可以使用 ALTER BUFFER POOL 语句（对于现有缓冲池）或 CREATE BUFFER POOL 语句（对于新缓冲池）来完成此操作。如果在 DPF 环境中将缓冲池大小设置为 AUTOMATIC，就不应该在 sysibm.sysbufferpoolnodes 中为该缓冲池定义任何条目。

注:

1. 由于自调整功能在不同内存区域之间重新分配内存，所以，必须至少启用两个内存区域（例如锁定内存区域和数据库共享内存区域）才能使自调整功能起作用。唯一的例外是由 *sortheap* 配置参数控制的内存。当仅将 *sortheap* 设置为 AUTOMATIC 时，将启用 *sortheap* 的自调整功能。
2. 要为自调整功能启用 *locklist* 配置参数，还必须为自调整功能启用 *maxlocks*，因此当 *locklist* 设置为 AUTOMATIC 时，*maxlocks* 也设置为 AUTOMATIC。要为自调整功能启用 *sheapthres_shr* 配置参数，还必须为自调整功能启用 *sortheap*，因此当 *sheapthres_shr* 设置为 AUTOMATIC 时，*sortheap* 也设置为 AUTOMATIC。
3. 仅当数据库管理器配置参数 *sheapthres* 设置为 0 时，才允许自动调整 *sheapthres_shr* 或 *sortheap*。
4. 自调整内存功能仅在 HADR 主服务器上运行。在 HARD 系统上激活自调整内存功能后，永远不会在辅助服务器上运行此功能，并且，仅当正确地设置配置后，此功能才会在主服务器上运行。如果运行了切换 HADR 数据库角色的命令，自调整内存操作也会切换，从而在新的主服务器上运行。

禁用自调整内存功能

可以通过将 *self_tuning_mem* 设置为 OFF 来对整个数据库禁用自调整功能。当 *self_tuning_mem* 设置为 OFF 时，设置为 AUTOMATIC 的内存配置参数和缓冲池仍为 AUTOMATIC，并且内存区保持其当前大小不变。

可以使用 UPDATE DATABASE CONFIGURATION 命令、SQLFUPD API 或通过控制中心中的**更改数据库配置参数**窗口来将 *self_tuning_mem* 设置为 OFF。

如果只有一个内存使用者启用了自调整功能，那么还可以有效地对整个数据库取消激活自调整功能。这是因为，当仅启用了一个内存区时，不能对内存进行再分布。

例如，要禁用 *sortheap* 配置参数自调整功能，可以输入以下语句:

```
UPDATE DATABASE CONFIGURATION USING SORTHEAP MANUAL
```

要禁用 *sortheap* 配置参数自调整功能，并同时将 *sortheap* 的当前值更改为 2000，请输入以下语句:

```
UPDATE DATABASE CONFIGURATION USING SORTHEAP 2000
```

在某些情况下，要对一个内存配置参数启用自调整功能，还必须对另一个相关内存配置参数启用此功能。例如，仅当启用了 *locklist* 配置参数的自调整功能时，才允许启用 *maxlocks* 配置参数的自调整功能。同样，仅当启用了 *sortheap* 配置参数的自调整功能时，才能启用 *sheapthres_shr* 配置参数的自调整功能。这意味着，如果禁用 *locklist* 或 *sortheap* 参数的自调整功能，也将分别禁用 *maxlocks* 或 *sheapthres_shr* 参数的自调整功能。

通过将缓冲池设置为特定大小，可以禁用缓冲池的自调整功能。例如，下列语句将禁用 bufferpool1 的自调整功能:

```
ALTER BUFFERPOOL bufferpool1 SIZE 1000
```

确定启用了自调整功能的内存使用者

要查看配置参数控制的内存使用者的自调整设置，请使用下列其中一种方法。

- 要从命令行查看配置参数的自调整设置，请使用 GET DATABASE CONFIGURATION 命令并指定 SHOW DETAIL 参数。

  在输出中，可以启用自调整功能的内存使用者将分组到一起，如下所示:

```
描述                                参数                  当前值                延迟的值
-------------------------------------------------------------------------------------------
自调整内存                          (SELF_TUNING_MEM) =  ON (Active)          ON
数据库共享内存大小（4KB）          (DATABASE_MEMORY) =  AUTOMATIC(37200)     AUTOMATIC(37200)
最大锁定列表存储器（4KB）          (LOCKLIST) =         AUTOMATIC(7456)      AUTOMATIC(7456)
每个应用程序的锁定列表百分比        (MAXLOCKS) =         AUTOMATIC(98)        AUTOMATIC(98)
程序包高速缓存大小（4KB）          (PCKCACHESZ) =       AUTOMATIC(5600)      AUTOMATIC(5600)
共享排序的排序堆阈值（4KB）        (SHEAPTHRES_SHR) =   AUTOMATIC(5000)      AUTOMATIC(5000)
排序列表堆（4KB）                  (SORTHEAP) =         AUTOMATIC(256)       AUTOMATIC(256)
```

- 也可以使用 db2CfgGet API 来确定是否启用调整功能。将返回下列值:

```
SQLF_OFF                    0
SQLF_ON_ACTIVE              2
SQLF_ON_INACTIVE            3
```

  SQLF_ON_ACTIVE 描述自调整功能已启用并处于活动状态的情况，而 SQLF_ON_INACTIVE 指示自调整功能已启用但当前处于不活动状态。

- 也可以在控制中心中的**数据库配置**窗口中查看配置设置。

要查看缓冲池的自调整设置，请使用下列其中一种方法。

- 要从命令行检索已启用自调整功能的缓冲池列表，请输入:

  ```
  db2 "select BPNAME, NPAGES from sysibm.sysbufferpools"
  ```

  当缓冲池启用了自调整功能时，该特定缓冲池的 sysibm.sysbufferpools 表中的 NPAGES 字段将设置为 -2。当自调整功能处于禁用状态时，NPAGES 字段将设置为缓冲池的当前大小。
- 要确定已启用自调整功能的缓冲池的当前大小，请按如下方式使用快照监视器并检查缓冲池的当前大小（bp_cur_buffsz 监视元素的值）:

  ```
  db2 get snapshot for bufferpools on db_name
  ```
- 要使用控制中心来查看缓冲池的自调整设置，请用鼠标右键单击缓冲池，并在对象详细信息窗格中查看缓冲池属性。

注意，内存调整器的反应受调整内存使用者的内存使用量所需时间的限制，这一点十分重要。例如，减小缓冲池大小的过程可能非常长，因此，为排序区域内存调整缓冲池内存大小所产生的性能优势可能不会立即体现。

自调整内存操作的详细信息和局限性

确定调整需求

为了确保在两个内存使用者之间进行公平比较，开发了一种新的公共度量。每个已调整的内存使用者根据附加内存计算预测好处，并将它报告给自调整内存进程。自调整内存使用这些数字作为内存调整的基础，从最不需要的使用者收回内存并将它重新分配给将获得最多好处的那些内存区域。

内存调整的频率

启用此功能后，自调整内存将定期检查数据库工作负载的变化性。如果工作负载不稳定（即，正在运行的查询未展示类似的内存特征），那么内存调整器重新分配内存的频率将会降低（两个调整周期之间最多间隔 10 分钟）以获得更稳定的趋势预测。对于具有更稳定内存概要文件的工作负载，内存调整器将更频繁地调整内存（两个调整周期之间最少间隔 30 秒）以便更快地汇合。

跟踪自调整内存的进度

可以使用 GET DATABASE CONFIGURATION 命令或使用快照来获取当前内存配置。自调整所作的更改将记录在 stmmlog 目录中的内存调整日志文件中。内存调整日志文件包含每个内存使用者在每个调整时间间隔内的资源需求总结。可以根据日志条目中的时间戳记确定这些时间间隔。

获得最佳配置期望所用的时间

使此功能处于已启用状态会导致快速调整参数来优化内存使用情况。最少只要 1 个小时就可以通过初始配置调整系统。大多数情况下，通常最多 10 个小时就完成调整。当针对数据库运行的查询明显展示不同的内存特征时，就会出现这种最坏的情况。

自调整内存的局限性

在少量内存可用的情况下（例如，因为 *database_memory* 的值设置得很小，或者因为多个数据库、实例或其他应用程序正在服务器上运行），自调整内存带来的性能好处将有限。

因为自调整内存根据数据库工作负载作出调整决定，所以内存特征不断变化的工作负载将限制自调整内存的能力以有效地进行调整。如果工作负载的内存特征不断变化，那么自调整内存调整内存的频率将会降低，并且将重复调整以改变目标条件。在这种情况下，自调整内存将无法获得绝对汇合，而是尝试维护调整为当前工作负载的内存配置。

分区数据库环境中的自调整内存

在分区数据库环境中使用自调整内存功能时，有一些因素决定该功能是否能适当地调整系统。

在分区数据库中启用自调整内存功能时，会将一个数据库分区指定为调整分区，所有内存调整决定都根据该数据库分区的内存和工作负载特征作出。一旦对调整分区作出调整决定，就会将内存调整分布到所有其他数据库分区上，以确保所有数据库分区都维持类似的配置。

单个调整分区模型要求只对具有类似内存需求的数据库分区使用该功能。可以使用下列准则确定您的分区数据库上是否要启用自调整内存功能。

建议在分区数据库中使用自调整的情况

当所有数据库分区都具有类似内存需求并且正在类似硬件上运行时，可以不进行任何修改就启用自调整内存。这些类型的环境共享下列特征:

- 所有数据库分区都在完全相同的硬件上，包括多逻辑节点平均分布在多个物理节点上
- 最佳或者接近最佳地分配数据
- 在数据库分区上运行的工作负载平均分布到各数据库分区上。这意味着没有一个数据库分区的一个或多个堆具有较多的内存需求。

在这种环境中，需要同等配置所有数据库分区，并且自调整内存将正确配置系统。

建议在分区数据库中小心使用自调整的情况

在环境中的大多数数据库分区具有类似内存需求并且正在类似硬件上运行的情况下，可以使用自调整内存功能，但在进行初始配置时要小心。这些系统可能有一组数据库分区用于数据，并且有一组少得多的协调程序分区和目录分区。在这些环境中，将协调程序分区和目录分区配置为与包含数据的数据库分区不同可能会有好处。

在此环境中，如果进行一些较小的设置，仍可以通过使用自调整内存功能获得好处。由于包含数据的数据库分区由大量数据库分区组成，因此应该在所有这些数据库分区上启用自调整，并应将这些数据库分区中的一个分区指定为调整分区。此外，由于目录分区和协调程序分区的配置可能不同，因此应在这些分区上禁用自调整内存。要在目录分区和协调程序分区上禁用自调整，将这些分区上的 *self_tuning_mem* 数据库配置参数更新为 OFF。

建议不要在分区数据库中使用自调整的情况

如果环境中的每个数据库分区的内存需求都不同，或者不同的数据库分区正在极不相同的硬件上运行，那么建议禁用自调整内存功能。这可以通过将所有分区上的 *self_tuning_mem* 数据库配置参数设置为 OFF 来实现。

比较不同数据库分区的内存需求

确定不同数据库分区的内存需求是否非常相似的最佳方法是查看快照监视器。如果所有分区上的下列快照元素相似（差别不超过 20%），那么认为这些分区相似。

通过发出 `get snapshot for database on <dbname>` 命令收集下列数据。

```
已分配的共享排序堆总数                  = 0
共享排序堆高水位标记                      = 0
后阈值排序数（共享内存）                  = 0
排序溢出数                              = 0

程序包高速缓存查询数                      = 13
程序包高速缓存插入数                      = 1
程序包高速缓存溢出数                      = 0
程序包高速缓存高水位标记（以字节计）      = 655360

散列连接数                              = 0
散列循环数                              = 0
散列连接溢出数                          = 0
小散列连接溢出数                        = 0
后阈值散列连接数（共享内存）            = 0

当前挂起的锁定数                        = 0
锁定等待数                              = 0
数据库等待锁定的时间（毫秒）            = 0
正在使用的锁定列表内存（以字节计）      = 4968
锁定升级数                              = 0
互斥锁定升级数                          = 0
```

通过发出 `get snapshot for bufferpools on <dbname>` 命令收集下列数据:

```
缓冲池数据逻辑读取数                = 0
缓冲池数据物理读取数                = 0
缓冲池索引逻辑读取数                = 0
缓冲池索引物理读取数                = 0
缓冲池总计读取时间（毫秒）          = 0
缓冲池总计写入时间（毫秒）          = 0
```

在分区数据库环境中使用自调整内存

在分区数据库环境中启用自调整功能之后，就会出现一个单独的数据库分区（称为*调整分区*），它监视内存配置的情况，并将任何配置更改传播到所有其他数据库分区以使所有参与数据库分区的配置保持一致。

调整分区是根据许多特征选择的，例如分区组中的数据库分区数以及已定义的缓冲池数。

- 要确定当前已指定为调整分区的数据库分区，请使用以下 ADMIN_CMD:

  ```
  CALL SYSPROC.ADMIN_CMD( 'get stmm tuning dbpartitionnum' )
  ```

- 要更改调整分区，请使用以下 ADMIN_CMD:

  ```
  CALL SYSPROC.ADMIN_CMD( 'update stmm tuning dbpartitionnum <db_partition_num>')
  ```

  发出此命令时，将以异步方式或者在数据库下次启动时更新调整分区。

- 要让内存调整器自动重新选择调整分区，请对 <db_partition_num> 值输入 -1。

在 DPF 系统上启动内存调整器

由于自调整功能要求所有分区都处于活动状态才能正确地调整多分区系统上的内存，所以，在 DPF 环境中，仅当使用显式 ACTIVATE DATABASE 命令激活数据库时，才会启动内存调整器。

对给定数据库分区禁用自调整功能

- 要对一部分数据库分区禁用自调整功能，请将不想调整的数据库分区的 *self_tuning_mem* 配置参数设置为 OFF。
-

  要对特定数据库分区上由配置参数控制的一部分内存使用者禁用自调整功能，请在该数据库分区上将相关配置参数值或缓冲池大小设置为 MANUAL 或特定值。但是，建议使自调整配置参数值在所有正在运行的分区中一致。
- 要对某个数据库分区上的特定缓冲池禁用调整功能，请发出指定了大小值的 ALTER BUFFER POOL 命令并指定 PARTITIONNUM 参数值，以指示要禁用自调整功能的分区。

  使用 PARTITIONNUM 子句在特定数据库分区上指定大小的 ALTER BUFFERPOOL 语句将在 SYSCAT.SYSBUFFERPOOLNODES 目录中为给定缓冲池创建例外条目，如果例外条目已存在，那么此语句将更新该条目。如果缺省缓冲池大小设置为 AUTOMATIC，那么当此目录中的某个缓冲池存在例外条目时，该缓冲池将不会参与自调整操作。要除去例外条目，以便可以对缓冲池重新启用自调整功能:

  1. 通过发出 ALTER BUFFERPOOL 语句并将缓冲池大小设置为特定值来对此缓冲池禁用调整功能。
  2. 发出另一个 ALTER BUFFERPOOL 语句并指定 PARTITIONNUM 子句，以便将此数据库分区上缓冲池的大小设置为缺省缓冲池大小。
  3. 发出另一个 ALTER BUFFERPOOL 语句，将大小设置为 AUTOMATIC，从而启用自调整功能。

在不均匀的环境中启用自调整内存

理想情况下，数据应该均匀地分布在所有数据库分区中，每个分区上运行的工作负载的内存需求应该比较接近。如果数据分布不均匀，以致一个或多个数据库分区包含的数据显著多于或少于其他数据库分区，就不应该对这些不规则的数据库分区启用自调整功能。这也适用于不同数据库分区上的内存需求不均匀的情况。例如，如果只在一个分区上执行需要大量资源的排序操作，或者某些数据库分区使用的硬件与别的分区不同并且有更多的可用内存，就会发生这种情况。在此类环境中，仍然可以对某些数据库分区启用自调整功能。要在不均匀环境中利用自调整内存功能，请确定一组具有相似数据和内存需求的数据库分区并对它们启用自调整功能。对于其余分区，应该手动进行内存配置。

缓冲池管理

缓冲池为数据库页提供工作内存和高速缓存。缓冲池通过允许从内存（而不是磁盘）中读取数据，来改善数据库系统的性能。由于内存的访问速度比磁盘的访问速度快得多，因此，数据库管理器需要读写磁盘的次数越少，性能就越好。由于大多数页数据处理发生在缓冲池内，因此配置缓冲池是唯一的最为重要的调整环节。

当应用程序访问表的某行时，数据库管理器会在缓冲池中查找包含该行的页。如果该页未存在于缓冲池中，那么数据库管理器将从磁盘中读取该页并将它放置在缓冲池中。一旦该页位于缓冲池中，数据库管理器就可以使用数据来处理查询。

激活数据库时会为缓冲池分配内存。要连接的第一个应用程序可能导致数据库被隐式激活。此外，在运行数据库管理器时，还可以创建和删除缓冲池以及调整缓冲池的大小。可使用 ALTER BUFFERPOOL 语句来增大缓冲池的大小。缺省情况下，对 ALTER BUFFERPOOL 语句指定 IMMEDIATE 关键字并且在内存可用时，一输入命令就会分配内存。如果内存不可用，那么该语句将作为 DEFERRED 执行，并且将在重新激活数据库时分配内存。如果减小缓冲池的大小，那么将在事务落实时释放内存。在取消激活数据库时释放缓冲池内存。在最后一个应用程序退出时可能会隐式取消激活数据库，这取决于激活数据库的方式。

注: 为了避免因增大缓冲池的大小而必须增大 *dbheap* 数据库配置参数的大小，几乎所有缓冲池内存均由数据库共享内存集产生，并且可以自动调整大小。

为了确保适当的缓冲池在所有情况下均可用，DB2 创建了许多小型系统缓冲池，它们的每个页的大小为: 4K、8K、16K 和 32K。每个缓冲池的大小为 16 页。用户看不到这些缓冲池。它们不会出现在系统目录或缓冲池系统文件中。您不能直接使用或改变这些缓冲池，但 DB2 在下列情况下使用它们:

- 指定的缓冲池由于是使用 DEFERRED 关键字创建的而未启动，或者所需页大小的缓冲池由于在创建时内存不足而变得不活动。

  将一条消息写入到管理通知日志。如有必要，将表空间重新映射到系统缓冲池。性能可能明显下降。
- 在数据库连接期间无法启动普通的缓冲池

  此问题通常由严重的原因（如内存不够条件）引起。虽然 DB2 将因为系统缓冲池而充分发挥其功能，但性能会明显下降。您应立即解决此问题。当发生此问题时，您会接收到一条警告，并且系统将一条消息写入管理通知日志。

创建缓冲池时，除非显式指定另一个页大小，否则页大小将为创建数据库时指定的大小。由于仅在表空间页大小与缓冲池页大小相同时才可以将页读入缓冲池，因此您为缓冲池指定的页大小应由表空间的页大小确定。您不能在创建缓冲池之后改变它的页大小。必须用不同的页大小创建新的缓冲池。

db2mtrk 调用的内存跟踪程序允许您查看为缓冲池分配的数据库内存大小。以下样本 db2mtrk 输出显示了一个用户创建的缓冲池（用括号中的数字“1”标识）和四个系统缓冲池（用括号中的页大小标识）:

```
> db2mtrk -d
Tracking Memory on: 2005/10/24 at 12:47:26

Memory for database: XMLDB
      utilh     pckcacheh catcacheh bph (1)   bph (S32K)  bph (S16K)  bph (S8K)
```

```
64.0K     576.0K    64.0K     4.2M      576.0K      320.0K      192.0K

bph (S4K)  shsorth   lockh     dbh       other
128.0K     0         640.0K    4.2M      192.0K
```

数据页的缓冲池管理

缓冲池中的页可能*正在使用*，也可能没有使用，它们可能是*脏*页，也可能是*干净*页:

- 正在使用的页就是当前正在读取或更新的页。如果某页正在更新，那么只有更新者才能访问该页。但是，如果某页未在更新，那么可以同时有许多个阅读者。
- “脏”页包含已更改但尚未写入磁盘的数据。

这些页将一直保留在缓冲池中，直到数据库关闭、其他页需要使用某一页所占用的空间或者页从缓冲池中被显式清除为止，例如，在删除对象的过程中。以下条件用于确定要除去哪一页以引入其他页:

- 是多久前引用的页
- 再次引用该页的可能性
- 页中数据的类型
- 该页是否已在内存中更改但尚未写入磁盘（更改的页始终在被覆盖之前写入磁盘）。

除非需要空间，否则不会从缓冲池中自动除去已写入磁盘的已更改页

页清除程序代理程序

在一个经过认真调整的系统中，通常是由页清除程序代理程序将已更改的页或者“脏”页写入磁盘中。页清除程序代理程序作为后台进程执行 I/O 并允许应用程序快速运行，因为其代理程序可以执行实际事务工作。页清除程序的代理程序有时称为*异步页清除程序*或*异步缓冲区写程序*，因为它们不与其他代理程序的工作协调进行，而仅在需要时工作。

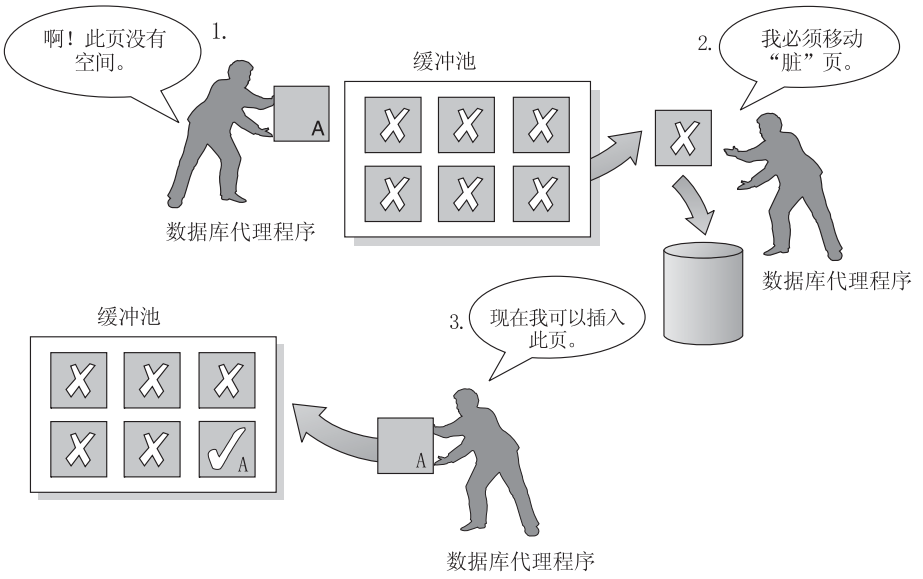
为了提高具有大量更新的工作负载的性能，您可能想启用主动清除页。这样做可提高性能，因为在选择任何给定时刻要写出哪些脏页时，页清除程序表现得更为主动。当快照揭示有许多同步数据页或索引页写操作与异步数据页或索引页写操作相关时尤其是如此。

有关更多信息，请参阅: 第 67 页的『主动的页清除』

缓冲池数据页管理的示例

下图举例说明了与执行所有 I/O 的数据库代理程序比较，页清除程序代理程序和数据库代理程序如何共享管理缓冲池的工作。

不带页清除程序



带页清除程序

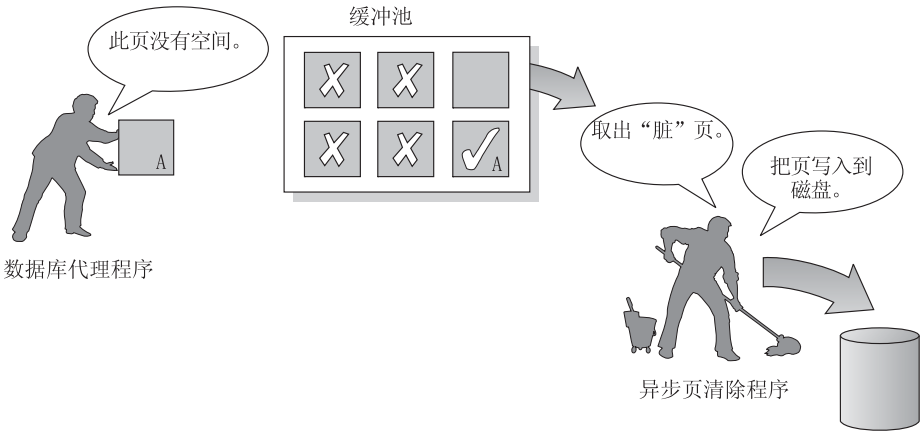


图 14. 异步页清除程序. 把“脏”页写至磁盘。

页清除和快速恢复

如果已将多页写入磁盘，那么在系统崩溃后数据库的恢复更快，因为数据库管理器可以从磁盘重建更多的缓冲池，而不必从数据库日志文件回放事务。

恢复期间必须读取的日志的大小是日志中下列记录的位置之间的差值:

- 最近写入的日志记录
- 描述对缓冲池中的数据所做的最早更改的日志记录。

页清除程序将脏页写入磁盘，以便在恢复期间需要重放的日志的大小从不超过以下值:

```
logfilsiz * softmax / 100（以 4K 页计）
```

其中:

- *logfilsiz* 表示日志文件的大小
- *softmax* 表示在数据库崩溃后要恢复的日志文件的百分比。例如，如果 *softmax* 的值为 250，那么如果发生崩溃，2.5 个日志文件将包含需要恢复的更改。

要将恢复期间日志读取时间减至最小，使用数据库系统监视器来跟踪执行页清除的次数。如果尚未对数据库启用主动清除页，那么系统监视器 *pool_lsn_gap_clns（触发的缓冲池日志空间清除程序）*监视元素将提供此信息。如果已经启用了此备用页清除，那么不应该发生这种情况，并且 *pool_lsn_gap_clns* 监视元素始终为 0。

可以使用 *log_held_by_dirty_pages* 监视元素来确定是否页清除程序未清除足够多的页数以满足用户设置的恢复条件。如果 *log_held_by_dirty_pages* 始终明显大于 *logfilsiz* * *softmax*，那么要么需要提供更多页清除程序，要么需要调整 *softmax*。

多个数据库缓冲池的管理

尽管每个数据库至少需要一个缓冲池，但可以为具有多个页大小的表空间的单个数据库创建几个缓冲池，每个缓冲池具有不同的大小或具有不同的页大小。每个缓冲池具有最小大小，这取决于平台。可以使用 ALTER BUFFERPOOL 命令来调整缓冲池大小。

新数据库具有称为 IBMDEFAULTBP 的缺省缓冲池，它的总大小是由平台确定的且缺省页大小基于创建数据库时指定的页大小。缺省页大小是作为 *pagesize* 参考数据库配置参数存储的。当创建缺省页大小的表空间且不给它分配特定缓冲池时，将给该表空间分配缺省缓冲池。可以调整缺省缓冲池调整的大小以及更改其属性，但不能将其删除。

缓冲池的页大小

创建或迁移数据库之后，可以创建其他缓冲池。可以创建使用 8 KB 页大小作为缺省值的数据库并且将用缺省页大小（在本例中为 8 KB 页大小）创建缺省缓冲池。或者，您可以创建使用 8 KB 页大小的缓冲池以及一个或多个具有相同页大小的表空间。此方法不要求您在创建数据库时更改 4 KB 缺省页大小。不能将表空间指定给使用另一种页大小的缓冲池。

注: 如果创建页大小大于 4 KB（如 8 KB、16 KB 或 32 KB）的表空间，需要给它分配使用相同页大小的缓冲池。如果此缓冲池当前不活动，DB2 尝试临时将该表空间指定给另一个使用相同页大小的活动缓冲池，或指定给第一次将客户机连接至数据库时 DB2 创建的某一缺省系统缓冲池。当再次激活该数据库且原来指定的缓冲池活动时，DB2 会将该表空间分配给该缓冲池。

使用 CREATE BUFFERPOOL 语句创建缓冲池时，可以指定特定缓冲池大小。如果未指定大小，那么它将设置为 AUTOMATIC 并由 DB2 管理。稍后要更改缓冲池的大小，使用 ALTER BUFFERPOOL 语句。

在一个分区数据库环境中，一个数据库的每个缓冲池在所有数据库分区上有相同的缺省定义，除非在 CREATE BUFFERPOOL 语句中另外指定了它，或 ALTER BUFFERPOOL 语句为特定数据库分区更改了缓冲池大小。

大缓冲池的优点

大缓冲池提供下列优点:

- 它们使频繁请求的数据页能够保存在缓冲池中，这样可更快地访问。更少的 I/O 操作可以减少 I/O 争用，从而可提供更短的响应时间并减少 I/O 操作所需的处理器资源。
- 它们提供在相同的响应时间内到达更高的事务处理速率的机会。
- 它们防止频繁使用的磁盘存储设备（如目录表）和频繁引用的用户表和索引的 I/O 争用。由于包含临时表空间的磁盘存储设备上的 I/O 争用减少，查询所需的排序也从中受益。

许多缓冲池的优点

如果下列其中任何一个条件适用于您的系统，应当只使用单个缓冲池:

- 总计缓冲空间小于 10000 个 4 KB 页。
- 找不到具备该应用程序知识来执行专门调试的人。
- 您在一个测试系统上工作。

在所有其他情况下，由于下列原因而考虑使用多个缓冲池:

- 可以把临时表空间分配给单个缓冲池中，以便为需要临时存储器的查询尤其是执行大量排序的查询提供更佳性能。
- 如果数据必须由很多小的更新事务应用程序反复地快速访问，考虑将包含该数据的表空间指定给单独的缓冲池。如果此缓冲池的大小定得合适，将有更多的机会找到它的页，以利于缩短响应时间和降低事务成本。
- 您可以将数据隔离到不同的缓冲池中，以利于特定的应用程序、数据和索引。例如，您可能要将频繁更新的表和索引置于一个单独的缓冲池中，与那些虽频繁查询但不频繁更新的表和索引分开。此更改将减少对第一组表的频繁更新对第二组表的频繁查询具有的影响。
- 对于很少使用的应用程序所访问的数据，可使用较小的缓冲池，尤其对于需要对一个很大的表非常随机地进行访问的应用程序。在这种情况下，不需要将数据保存在缓冲池中的时间超过单个查询的保存时间。最好为此数据保留一个小缓冲池，而释放多余的内存用于其他用途，如用于其他缓冲池。
- 当将不同的活动和数据隔离到不同的缓冲池之后，可从统计信息和记帐跟踪获得一个良好的、相对成本较低的性能诊断数据。

启动时缓冲池内存分配

当使用 CREATE BUFFERPOOL 命令来创建缓冲池或使用 ALTER BUFFERPOOL 语句来变更缓冲池时，所有缓冲池需要的全部内存必须可供数据库管理器使用，以便在启动该数据库时可分配所有缓冲池。如果在数据库管理器联机时创建或修改缓冲池，那么附加内存应在数据库全局内存中可用。如果在创建新缓冲池或增大现有缓冲池的大小时指定 DEFERRED 关键字，但所需内存不可用，那么数据库管理器在下次激活数据库时进行更改。

如果在数据库启动时此内存不可用，那么数据库管理器将仅使用最小大小为 16 页的系统缓冲池（每个页大小对应一个系统缓冲池）启动，并返回 SQL1478W（SQLSTATE01626）警告。数据库会继续保持在此运作状态，直到更改其配置且数据库可以完全重新启动为止。性能可能不是最佳的。数据库管理器仅使用最小大小值启动来允许您连接至数据库，以便可以重新配置缓冲池大小或执行其他关键任务。只要执行这些任务，就重新启动数据库。不要在这种状态下超长时间运行数据库。

为了避免仅使用系统缓冲池启动数据库，可以使用 DB2_OVERRIDE_BPF 注册表变量来调整所需的内存，以便它与可用内存相符。

主动的页清除

从 V8.1.4 开始，可以使用另一种方法来配置系统中的页清除。这种方法与缺省行为的差别在于，在选择任何给定时刻要写出哪些脏页方面，页清除程序表现得更为主动。这种新的页清除方法与缺省页清除方法主要在以下两方面不同:

1. 页清除程序不考虑 chngpgs_thresh 配置参数。

   当使用这种页清除方法时，页清除程序将不再对 chngpgs_thresh 配置参数的值作出反应。这种页清除方法不会尝试使一定百分比的缓冲池保持为干净的，这种页清除方法提供了一种机制，在这种机制中，将向代理程序通知刚写出的良好牺牲页的位置，因此，代理程序不必通过搜索缓冲池来查找牺牲页。当良好牺牲页的数目减少到低于可接受的值时，就会触发页清除程序，它会继续搜索整个缓冲池，写出潜在的牺牲页，并通知代理程序这些页的位置。

2. 页清除程序不再响应记录器发出的 LSN 间隔触发器。

   如果日志空间的数量中包含更新了缓冲池中的最旧页的日志记录，并且当前日志位置超过了 softmax 参数所允许的值，就认为数据库处于“LSN 间隔”状态。当使用缺省的页清除方法时，如果记录器检测到产生 LSN 间隔，那么它将触发页清除程序来写出导致 LSN 间隔状态的所有页。也就是说，它将写出比 softmax 参数允许的页更早的页。在没有产生 LSN 间隔的时间段内，页清除程序将处于空闲状态。但是，一旦产生 LSN 间隔，就会激活页清除程序来写出大量页，然后它又会返回到休眠状态。这将使 I/O 子系统达到饱和，这又会影响正在对页进行读写操作的其他代理程序。而且，当触发 LSN 间隔时，页清除程序可能不能够快速进行清除，并且 DB2 可能用完了日志空间。

   另一种页清除方法通过在更长的时间段内进行相同次数的写操作来调整此行为。清除程序通过根据活动的当前级别不但主动清除当前处于 LSN 间隔状态的页，而且清除不久将进入 LSN 间隔状态的页来完成此调整。

要使用新的页清除方法，应将 DB2_USE_ALTERNATE_PAGE_CLEANING 注册表变量设置为“ON”。

将数据预取至缓冲池

*预取*页就是在应用程序将需要一个或多个页之前，提前从磁盘中检索那些页。将索引和数据页预取至缓冲池，可以通过减少 I/O 等待时间帮助提高性能。此外，并行 I/O 增强预取效率。

有两种类别的预取:

- **顺序预取**: 应用程序在需要这些页之前，将连续的页读至缓冲池中的一种机制。
- **列表预取**: 有时称为*列表顺序预取*。有效预取一组非连续的数据页。

这两种读取数据页的方法是对数据库管理器代理程序读取方法的补充。当仅检索一页或少量连续页，但只传送一页数据时，使用数据库管理器代理程序读取方法。

预取和分区内并行性

预取对于分区内并行性的性能很重要，此分区内并行性在扫描一个索引或表时使用多个子代理程序。这些并行扫描产生更大的数据耗用速率，并需要更高的预取速率。

对于少量预取的成本，并行扫描比串行扫描更高。如果对于一个串行扫描未进行预取，那么由于代理程序始终需要等待 I/O，所以该查询运行起来更慢。如果对于一个并行扫描未进行预取，那么所有子代理程序可能需要等待，因为一个子代理程序在等待 I/O。

由于其重要性，使用分区内并行性可以更主动地执行预取。顺序检测机制容许相邻页之间有更大的间隔，以便可以将这些页视为顺序页。这些间隔的宽度随着扫描中涉及的子代理程序数目的增加而增加。

顺序预取

使用单一 I/O 操作将几个连续的页读到缓冲池中，可以大大减少应用程序开销。此外，将几个范围内的页读到缓冲池中的多个并行 I/O 操作可以有助于减少 I/O 等待时间。

当数据库管理器确定顺序 I/O 是合适的且预取可改善性能时，预取启动。对于像表扫描和表排序这类情况，数据库管理器可以轻易地确定顺序预取将有利于提高 I/O 性能。在这些情况下，数据库管理器会自动启动顺序预取。以下示例（可能需要表扫描）将适合于顺序预取:

```
SELECT NAME FROM EMPLOYEE
```

表空间的 PREFETCHSIZE 的含义

要为每个表空间定义预取的页数，在 CREATE TABLESPACE 或 ALTER TABLESPACE 语句中使用 PREFETCHSIZE 子句。在 SYSCAT.TABLESPACES 系统目录表的 PREFETCHSIZE 列中维护指定的值。

一个好方法是将 PREFETCHSIZE 值显式设置为表空间容器数、每个容器下的物理磁盘数（如果使用了 RAID 设备的话）与您的表空间的 EXTENTSIZE 值的乘积，该值是在数据库管理器使用另一个容器之前写入容器的页数。例如，如果扩展数据块大小为 16 页，且该表空间有两个容器，那么您可以将预取量设置为 32 页。如果每个容器具有 5 个物理磁盘，那么可以将预取数量设置为 160 页。

数据库管理器监视缓冲池的使用情况，以确保在另一个工作单元需要这些页的情况下，预取不会从缓冲池中除去这些页。为避免引起问题，数据库管理器可将预取的页数限制到小于您为表空间所指定的数。

预取大小可以显著影响性能，特别是对大表扫描更是如此。使用数据库系统监视器和其他系统监视工具来帮助您调整表空间的 PREFETCHSIZE。您可以收集有关是否为如下情况的信息:

- 有 I/O 等待您的查询，这可使用您的操作系统可用的监视工具来获知。
- 正在进行预取，可查看数据库系统监视器提供的 *pool_async_data_reads（缓冲池异步数据读取）*数据元素来获知。

如果有 I/O 等待并且该查询是预取数据，您可增大 PREFETCHSIZE 的值。如果预取程序不是 I/O 等待的原因，增大 PREFETCHSIZE 值将不会提高查询的性能。

在所有预取类型中，当预取大小是表空间的扩展数据块大小的倍数，并且该表空间的扩展数据块是在不同的容器中时，可以并行执行多个 I/O 操作。为获取更好性能，配置容器以使用独立的物理设备。

顺序检测

在某些情况下，并不能立即确定顺序预取是否将提高性能。在这类情况下，数据库管理器可以监视 I/O，且如果正在读取顺序页，那么数据库管理器可以激活预取。在这种情况下，数据库管理器在适当时激活和取消激活预取。这种类型的顺序预取称为*顺序检测*，并且同时适用于索引页和数据页。使用 *seqdetect* 配置参数来控制数据库管理器是否执行顺序检测。

例如，如果打开顺序检测，那么下列 SQL 语句可能从顺序预取受益:

```
SELECT NAME FROM EMPLOYEE      WHERE EMPNO BETWEEN 100 AND 3000
```

在此示例中，优化器可能已经启动使用 EMPNO 列上的索引来扫描该表。如果该表与此索引密切相关，那么数据页读取将几乎是顺序的，并且预取也可提高性能，因此将会进行数据页预取。

索引页预取也可能发生在此示例中。如果必须检查许多索引页，并且数据库管理器检测到正在进行索引页的顺序页读取，那么将进行索引页预取。

改进的顺序预取的基于块的缓冲池

从磁盘预取页由于 I/O 开销而使成本很高。如果处理与 I/O 交迭，那么可以显著提高吞吐量。大多数平台提供高性能的将连续页从磁盘读到内存的非连续部分的原语。这些原语通常称为*散射读*或*向量化 I/0*。在某些平台上，这些原语的性能不能与在较大块大小中执行 I/O 相比。

缺省情况下，缓冲池基于页，这指的是将磁盘上的连续页预取到内存非连续的页。如果可以将连续页从磁盘读到缓冲池中的连续页中，那么可以增强顺序预取。

可以为此目的创建基于块的缓冲池。基于块的缓冲池由页区域和块区域组成。页区域对于非顺序预取工作负载是必需的。块区域由块组成，其中每个块包含指定数量的连续页，它称为*块大小*。

基于块的缓冲池的最佳使用情况取决于指定的块大小。块大小是执行顺序预取的 I/O 服务器考虑按其执行基于块的 I/O 的粒度。扩展数据块是在容器中对表空间进行条带分割的粒度。因为可以将具有不同扩展数据块大小的多个表空间绑定到用同一块大小定义的缓冲池，所以考虑扩展数据块大小如何与块大小交互以便有效使用缓冲池内存。在下列情况下可能浪费缓冲池内存:

- 如果扩展数据块大小（它确定预取请求大小）小于为缓冲池指定的 BLOCK_SIZE。
- 如果在预取请求中请求的某些页已存在于缓冲池的页区域中。

I/O 服务器允许每个缓冲池块中的某些浪费的页，但是，如果将浪费过多的块，那么 I/O 服务器执行非基于块的预取到缓冲池的页区域中。这不是最佳性能。

为了获取最佳性能，将相同扩展数据块大小的表空间绑定到块大小等于表空间扩展数据块大小的缓冲池。如果扩展数据块大小大于块大小，但不是在扩展数据块大小小于块大小时，可以达到良好性能。

要创建基于块的缓冲池，可使用 CREATE 和 ALTER BUFFERPOOL 语句。

注: 基于块的缓冲池将用于顺序预取。如果应用程序不使用顺序预取，那么会浪费缓冲池的块区域。

列表预取

*列表预取*或*列表顺序预取*是一种有效访问数据页的方法，甚至在所需的数据页并不连续时，也是如此。列表预取可以与单个或多个索引访问结合使用。

如果优化器使用一个索引来访问行，它可将读取数据页延迟到从该索引获得所有行标识（RID）之后进行。例如，优化器可执行索引扫描来确定要检索的行和数据页，并给予先前定义的索引 IX1:

```
INDEX IX1:  NAME    ASC,
                        DEPT    ASC,
                        MGR     DESC,
                        SALARY  DESC,
                        YEARS   ASC
```

和下列搜索条件:

```
WHERE NAME BETWEEN 'A' and 'I'
```

如果数据未根据此索引集群，那么列表预取包括一个把从索引扫描获得的 RID 的列表排序的步骤。

预取和并行性的 I/O 服务器配置

要启用预取，数据库管理器需要启动不同的控制线程（称为 *I/O 服务器*）来读取数据页。因此，查询处理分为两个并行活动: 数据处理（CPU）和数据页 I/O。I/O 服务器等待从 CPU 处理活动发来的预取请求。这些预取请求中包含有关满足查询所需的 I/O 的描述。可能的预取方法决定数据库管理器何时及如何生成预取请求。

使用配置参数 *num_ioservers* 配置足够的 I/O 服务器，可以大大提高可使用数据预取的查询的性能。要使并行 I/O 的机会最大化，请将 *num_ioservers* 设置为至少等于数据库中的物理磁盘数。

高估 I/O 服务器的数目要好于低估 I/O 服务器的数目。如果指定了多余的 I/O 服务器，虽然这些服务器得不到使用，但其内存页会被调出。因此，性能不会受到影响。每个 I/O 服务器进程均有编号。数据库管理器始终使用编号最低的进程，因此某些编号较高的进程可能永远得不到使用。

要确定所需的 I/O 服务器数目，请考虑下列因素:

- 可以同时将预取请求写入 I/O 服务器队列的数据库代理程序的数目。
- I/O 服务器可并行工作的最大程度。

可以考虑将 *num_ioservers* 数据库配置参数的值设置为 AUTOMATIC，以便 DB2 可以根据系统配置选取智能值。

异步 I/O 的配置

在某些平台上，DB2 使用异步 I/O（AIO）来提高活动（例如，页清除和页预取）的性能。如果将容器中的数据分配到多个磁盘上，那么 AIO 是最有效的方法。此外，通过调整基本操作系统的 AIO 基础结构，也可以提高性能。

例如，在 AIX® 上，可以调整操作系统中的 AIO。当 AIO 对 SMS 或 DMS 文件容器工作时，称为 AIO 服务器的操作系统进程将会管理 I/O。如果此类服务器的数量太

少，那么 AIO 的优点可能会因 AIO 请求数太少而得不到充分的体现。要在 AIX 上配置 AIO 服务器的数目，请使用 *smit* AIO *minservers* 和 *maxservers* 参数。

使用并行 I/O 预取示例: 下图举例说明如何使用 I/O 服务器来将数据预取到缓冲池中。

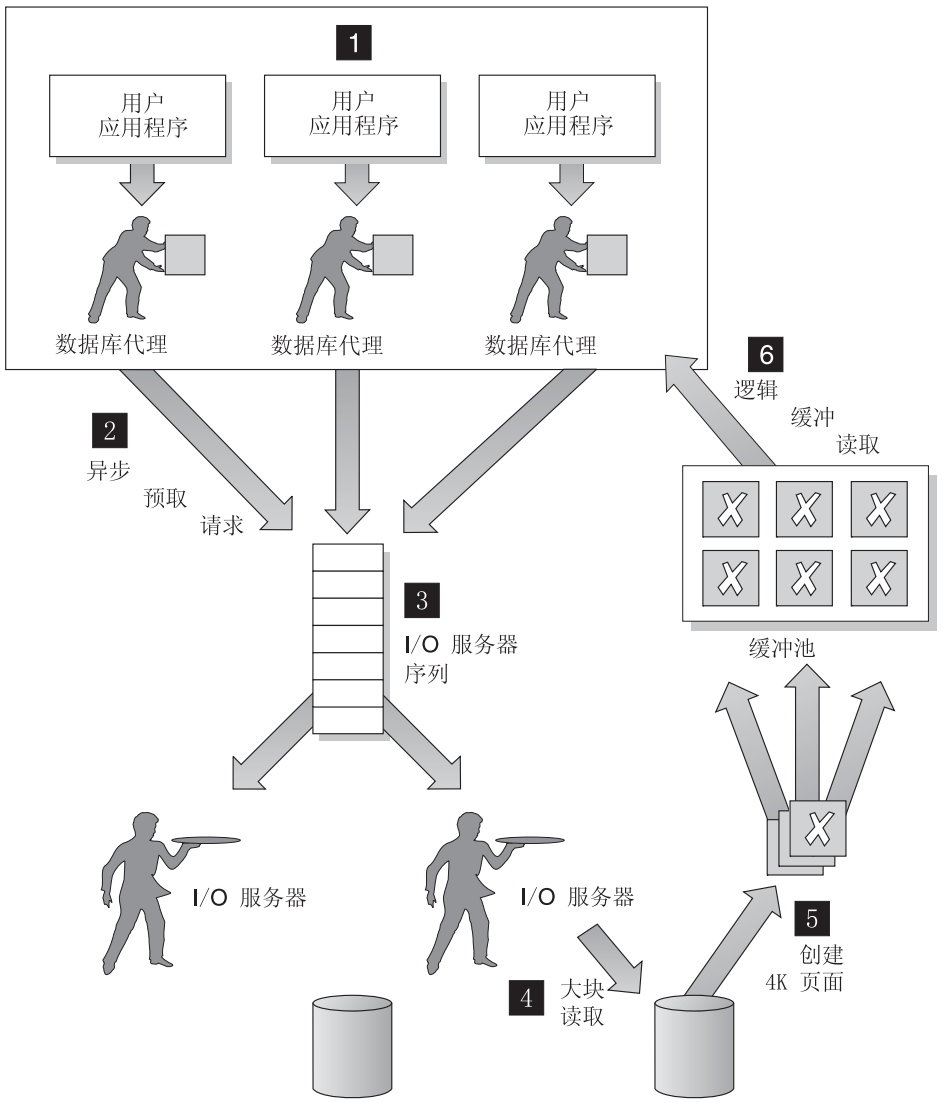


图 15. 使用 I/O 服务器预取数据

1 用户应用程序将请求传送到数据库代理程序，该代理程序已由数据库管理器指定给用户应用程序。

2，**3**
数据库代理程序确定应使用预取来获取满足该请求所需的数据，并向 I/O 服务器队列写入一个预取请求。

4，**5**
第一个可用的 I/O 服务器从这个队列读取预取请求，然后从表空间将数据读入

缓冲池。可以同时从表空间访存数据的 I/O 服务器的数目取决于该队列中预取请求的数目以及 *num_ioservers* 数据库配置参数配置的 I/O 服务器的数目。

6 数据库代理程序对缓冲池中的数据页执行必要的操作并将结果返回到用户应用程序。

并行 I/O 管理: 如果在表空间存在多个容器，数据库管理器可以启动*并行 I/O*，其中，数据库管理器使用多个 I/O 服务器处理单个查询的 I/O 需求。每个 I/O 服务器处理单个容器的 I/O 工作负载，因此可以并行方式读取几个容器。以并行方式执行 I/O 可以显著提高 I/O 吞吐量。

尽管单个 I/O 服务器可以处理每个容器的工作负载，但是，可以并行方式执行 I/O 的 I/O 服务器的实际数目限制为请求的数据所分布的物理设备的数目。因此，您需要与物理设备同样多的 I/O 服务器。

在下列情况下，以不同方式启动并行 I/O:

- **顺序预取**

  对于顺序预取，当预取大小是一个表空间的扩展数据块大小的倍数时，启动并行 I/O。然后，沿扩展数据块边界将每个预取请求分为许多小的请求。然后将这些小的请求再分配给不同的 I/O 服务器。

- **列表预取**

  对于列表预取，根据存储数据页的容器，将每个页列表分成更小的列表。然后把这些更小的列表分配到不同的 I/O 服务器。

- **数据库或表空间的备份和复原**

  为了备份或复原数据，并行 I/O 请求的数目等于备份缓冲区大小除以扩展数据块大小所得的商，最大值可达到容器数。

- **数据库或表空间的复原**

  为了复原数据，采取用于顺序预取一样的方式来启动和分割并行 I/O 请求。直接将数据从复原缓冲区移动到磁盘中，而不是将数据复原到缓冲池中。

- **装入**

  在装入数据时，您可以使用 LOAD 命令的 DISK_PARALLELISM 选项来指定 I/O 并行性的级别。如果未指定此选项，那么数据库管理器使用缺省值，该值基于与该表相关的所有表空间的表空间容器的累计数目。

为保证并行 I/O 的优化性能，确保:

- 有足够的 I/O 服务器。将 I/O 服务器的数目指定为稍大于用于数据库内所有表空间的容器的数目。
- 表空间的扩展数据块大小和预取大小是合理的。要防止过分使用缓冲池，预取大小不应太大。理想大小是扩展数据块大小、每个容器下的物理磁盘数（如果使用了 RAID 设备的话）与表空间容器数的乘积。扩展数据块大小应该稍小些，合适的值在 8 至 32 页之间。
- 容器位于不同的物理驱动器上。
- 为了确保一致的并行度，所有容器要有相同大小。

如果一个或多个容器比其他容器小，降低优化并行预取的可能性。请考虑以下示例:

- 在填充一个较小的容器后，附加的数据会储存在其余的一些容器中，导致容器变得不平衡。不平衡的容器会降低并行预取的性能，因为可从中预取数据的容器数目可能小于容器的总数。
- 如果以后添加了一个较小的容器并且对数据进行了重新平衡，那么此较小容器包含的数据要少于其他容器。相对于其他容器，此容器的少量数据将不会优化并行预取。
- 如果某个容器较大且所有其他的容器都已填充，该容器是存储附加数据的唯一容器。数据库管理器不能使用并行预取访问附加的数据。

- 当使用分区内并行性时，存在足够的 I/O 容量。在 SMP 机器上，分区内并行性可以通过在多个处理器上运行该查询来减少查询所用时间。必需足够的 I/O 容量来使每个处理器都忙于工作。一般需要附加的物理驱动器来提供该 I/O 容量。

  预取大小必须较大，以用于以较高速率进行预取并有效使用 I/O 容量。

  要求的物理驱动器的数目取决于驱动器和 I/O 总线的速度和性能以及处理器的速度。

调整排序性能: 因为查询经常需要已排序或已分组的结果，所以排序经常是必需的，且排序堆区域的适当配置对良好的查询性能很重要。在以下情况下需要排序:

- 没有索引满足请求的排序（例如，使用 ORDER BY 子句的 SELECT 语句）。
- 索引存在，但排序会比用索引更有效。
- 创建索引。
- 删除索引，这导致对索引页号进行排序。

影响排序的元素

下列因素影响排序性能:

- 下列数据库配置参数的设置:
  - 排序堆大小（*sortheap*），它指定要用于每个排序的内存量
  - 排序堆阈值（*sheapthres*）和共享排序的排序堆阈值（*sheapthres_shr*），它们控制可用于整个实例中所有排序的内存总量
- 工作负载中需要大量排序的语句数。
- 可帮助避免不需要的排序的索引是否存在。
- 使用不将排序最小化的应用程序逻辑
- 并行排序，它改善排序性能，但只有在语句使用分区内并行性时才可进行。
- 排序是*溢出的*还是*非溢出的*。如果已排序的数据不能完全放入排序堆（该排序堆是每次执行排序时分配的一块内存）中，那么它溢出到数据库所拥有的临时表中。未溢出的排序总是比那些溢出的排序执行得好。
- 排序结果是*管道式的*还是*非管道式的*。如果已排序的信息可直接返回，而不需要一个临时表来存储数据的最终排序列表，那么它是一个管道式排序。如果已排序的信息需要返回一个临时表，那么它是一个非管道排序。一个管道排序始终比一个非管道排序执行得好。

  还要注意在管道式排序中，在应用程序关闭与排序关联的游标前，不会释放排序堆。一个管道排序可继续用尽内存，直到关闭游标。

一般情况下，整个实例可用的总排序内存（*sheapthres*）应尽可能大，而不会导致过度页面调度。尽管可以在排序内存中完全执行排序，但这可以导致过度页交换。在此情况下，您失去大的排序堆的优势。因此，无论何时您调整排序配置参数，应该使用一个操作系统监视器来跟踪系统页面调度中的任何变化。

管理排序性能的技术

标识排序对性能有主要影响的特定应用程序和语句:

1. 将事件监视器设置在应用程序级别和语句级别，以帮助您标识总计排序时间最长的应用程序。
2. 在其中每个应用程序中，查找具有最长*总排序时间*的语句。

   您也可以搜索说明表，以标识有排序操作的查询。
3. 使用这些语句作为“设计顾问程序”的输入，这将标识索引并可选择创建索引以减少排序需要。

使用数据库系统监视器和基准程序测试技术，来帮助设置 *sortheap* 和 *sheapthres* 配置参数。对于每个数据库管理器和每个数据库:

1. 建立并运行一个代表性工作负载。
2. 对于每个适用的数据库，在基准程序工作负载周期内对以下性能变量收集其平均值:
   - 正在使用的总排序数（sort_heap_allocated 监视元素的值）
   - 活动排序数和活动散列连接数（active_sorts 和 active_hash_joins 监视元素的值）。
3. 将 *sortheap* 设置为每个数据库平均*使用的排序堆总和*。

   注: 随着 DB2 部分键二进制排序技术的提高，已包含了非整数数据类型键，因此排序长键时需要附加内存。如果长键用于排序，那么需要增大 *sortheap* 配置参数。
4. 设置 *sheapthres*。要估计适当的大小:
   a. 确定实例中哪个数据库具有最大的 *sortheap* 值。
   b. 确定此数据库的排序堆平均大小。

      如果太难于确定的话，那么使用最大排序堆的 80%
   c. 将 *sheapthres* 设置为平均活动排序次数乘以上面计算的平均排序堆大小。

      这是建议的初始设置。以后可以使用基准测试技术来优化此值。

还可以使用自调整内存功能来自动根据需要分配和取消分配排序所需的内存资源。要使用此功能:

- 通过将 *self_tuning_mem* 配置参数设置为“ON”来启用数据库的自调整内存功能。
- 将 *sortheap* 和 *sheapthres_shr* 配置参数设置为“AUTOMATIC”。
- 将 *sheapthres* 配置参数设置为“0”。

维护表和索引的组织

随着时间的推移，当记录分布在越来越多的数据页上时，表中的数据将变成碎片，从而增大表和索引的大小。这会增加执行查询时需要读取的页数。重组表和索引会压缩数据，从而回收浪费的空间并改善数据访问情况。

执行索引或表重组的步骤如下所示:

1. 确定是否需要重组任何表或索引。
2. 选择重组方法。
3. 对标识的对象执行重组。
4. 监视重组的进度。
5. 对于联机表重组，可以在需要时暂停重组过程，这将允许您稍后继续。
6. 评估重组的结果，并确定操作是成功还是失败。对于脱机表重组和任何索引重组，操作是同步的，并且重组的结果将在命令返回时显示出来。联机表重组是异步处理的，因此，要评估结果，您需要引用历史记录文件。
7. 如果执行了联机表重组，那么可以选择进行恢复。请参阅: 第 82 页的『恢复失败的联机表重组』
8. 收集有关已重组对象的统计信息。
9. 重新绑定访问已重组对象的应用程序。

表重组

对表数据进行大量更改之后，逻辑顺序数据可能在非顺序物理数据页，因此数据库管理器必须执行其他读操作来访问数据。另外，在删除大量行后，也需要执行其他的读操作。在这样的情况下，您可以考虑重组表以匹配索引和回收空间。此时，既可重组系统目录表，也可以重组数据库表。

注: 由于重组表的时间通常要比运行统计信息的时间长，因此您可以执行 RUNSTATS 以刷新当前的数据统计信息并重新绑定应用程序。如果刷新的统计信息并没有改善性能，那么重组可能会有所帮助。有关 REORG TABLE 实用程序的选项和行为的详细信息，参阅其命令参考。

考虑以下因素，以确定是否应重组表:

- 对查询所访问的表进行了大量的插入、更新和删除活动
- 对于使用具有高集群率的索引的查询，其性能发生了明显变化
- 在执行 RUNSTATS 以刷新统计信息后，性能没有得到改善
- REORGCHK 命令指示需要重组表
- 综合考虑查询性能不断降低所浪费的成本和重组表所需的成本（包括 CPU 时间、经过的时间和 REORG 实用程序在完成重组操作之前锁定表造成的并行性降低），以确定是否进行表重组。

减少重组表的需要

要减少重组表的需要，在创建表后执行下列任务:

- 改变表以添加 PCTFREE
- 在索引上使用 PCTFREE 来创建集群索引
- 将数据排序

- 装入数据

在执行这些任务后，具有集群索引和 PCTFREE 设置的表有助于保持原来的排序顺序。当表页上有足够的空间时，可以将新数据插入正确的页，以维护索引的集群特征。随着更多的数据插入，表页会因此而变满，记录会被追加至表的末尾，因而表将逐渐失去集群特性。

如果在创建集群索引后执行 REORG TABLE 命令，或者执行排序和 LOAD 命令，那么索引会尝试维护数据的特定顺序，这将改善 RUNSTATS 实用程序所搜集的 CLUSTERRATIO 或 CLUSTERFACTOR 统计信息。

注: 创建多维集群（MDC）表可以减少重组表的必要性。对于 MDC 表，将在 CREATE TABLE 语句的 ORGANIZE BY DIMENSIONS 子句中指定为参数的列上维护集群。但是，如果 REORGCHK 认为存在太多未使用的块或需要对块进行压缩，那么可能会建议重组 MDC 表。

选择重组表的方法

有两种不同的表重组方法: *传统*或脱机 REORG 以及*原地*或联机 REORG。

REORG 命令的 INPLACE 选项指定联机重组。如果未指定此选项，那么将运行脱机 REORG。

每种重组方法都有一些优点和缺点。下面总结了这些优缺点。选择重组方法时，应考虑哪种方法提供的优点是您优先要解决的方面，例如，如果发生故障时进行恢复比快速完成重组更重要，那么最好使用联机重组方法。

脱机重组的优点

- 表重组速度最快，尤其在不需要重组 LOB/LONG 数据时。
- 完成时精确地划分了表和索引的集群。
- 重组表之后立即重建索引。不需要单独的步骤来重建索引。
- 可以在临时表空间中构建影子副本。这减少了包含目标表或索引的表空间中需要的空间大小。
- 允许指定并使用集群索引外的索引来重新划分数据的集群，而联机重组在存在集群索引时必须使用现有集群索引。

脱机重组的缺点

- 表访问受限制。只有在重组的排序和构建阶段才允许应用程序对表进行只读访问。
- 由于使用影子副本方法，所以需要较大空间。
- 与联机 REORG 相比，对 REORG 过程的控制较少: 不能暂停然后重新启动脱机重组。

联机重组的优点

- 允许应用程序在 REORG 期间（截断阶段除外）对表进行完全访问。
- 对 REORG 过程具有更多控制: 该过程在后台异步运行，并且可以使其暂停、继续以及停止。例如，如果正在对表进行大量更新或删除操作，那么可以暂停 REORG 过程。
- 在发生故障时可恢复重组过程。
- 由于采用递增方式处理表，所以需要较少工作存储器。

- 在 REORG 进行时立即可以实现重组的优点。

联机重组的缺点

- 可能产生非最佳数据集群或非最佳索引集群，这取决于 REORG 期间访问表的事务类型。
- 在重组开始时重组的页可能更新次数更多，从而比重组过程中稍后重组的表具有更多碎片。
- 速度比脱机重组要慢。对于正常集群 REORG（不仅仅是空间回收），执行联机重组的时间可能是脱机重组所用时间的 10 到 20 倍。对于同时正在对其运行应用程序的表，或者对于定义了大量索引的表，此时间可能还要长很多。
- 联机重组是一个可恢复的过程，但这会导致日志记录要求提高。可能需要极大量的日志空间，最大可为表大小的若干倍。这取决于重组期间移动的行数、对表定义的索引数以及索引的大小。
- 将维护索引而不是进行重建，因此以后可能需要重组索引。

表 2. 联机重组与脱机重组的比较

特征	脱机重组	联机重组
性能	快	慢
完成时数据的集群因子	良好	非最佳集群
并行性（对表的访问）	从不能访问到只读访问	从只读访问到完全访问
数据存储空间要求	非常大	不是非常大
日志记录存储空间要求	不是非常大	非常大
用户控制（暂停和重新启动重组过程的能力）	较少控制	较多控制
可恢复性	完全可恢复或完全不可恢复: 成功或失败。	可恢复
索引重建	进行	不进行
支持所有类型的表	是	否
指定除集群索引外的索引	是	否
使用临时表空间	是	否

表 3. 联机重组和脱机重组支持的表类型

表类型	是否受脱机重组支持	是否受联机重组支持
多维集群表（MDC）	是[1]	否
范围集群表（RCT）	否[2]	否
追加方式表	否	否[3]
具有 LONG/LOB 数据的表	是[4]	否
具有 1 类索引的表	是[5]	否
系统目录表: SYSIBM.SYSTABLES、SYSIBM.SYSSEQUENCES、SYSIBM.SYSDBAUTH 和 SYSIBM.SYSROUTINEAUTH	是	否

1. 由于集群是通过 MDC 块索引自动维护的，所以重组 MDC 表只涉及空间回收。因为使用了块索引，所以不能指定索引。
2. RCT 的范围区域始终保持为集群。
3. 关闭追加方式后，就可以执行联机重组。
4. 重组 LONG/LOB 数据要用大量时间。重组 LONG/LOB 数据不会有助于提高查询性能；重组该数据只能是为了回收空间。
5. 重组之后，将所有索引都重建为 2 类索引。

有关执行这些表重组方法的详细信息，请参阅 REORG TABLE 语法描述。

监视表重组的进度

有关表重组的当前进度的信息将写入数据库活动的历史记录文件中。历史记录文件中包含每个重组事件的记录。要查看此文件，请执行 `db2 list history` 命令以打开包含所重组表的数据库。

此外，也可使用表快照来监视表重组的进度。不管如何设置“数据库监视器表”开关，系统均会记录表重组监视数据。

如果出现错误，SQLCA 转储将写入到历史记录文件中。对于现场表重组方法，会将其状态记录为 PAUSED。

脱机表重组

脱机表重组使用影子副本方法来构建正在重组的表的完整副本。

传统或脱机表重组包括四个阶段:

1. *排序*
   如果索引是使用 REORG TABLE 命令指定的，或者如果对表定义了集群索引，那么首先根据该索引对表行进行排序。如果指定了 INDEXSCAN 选项，那么使用索引扫描对表进行排序；否则，使用表扫描排序。此阶段仅适用于集群重组。空间回收重组在构建阶段开始。
2. *构建*
   在此阶段中，将在要重组的表所在的表空间中或使用 REORG 命令指定的临时表空间中构建整个表的已重组副本。
3. *替换*
   在此阶段中，通过从临时表空间中复制回表对象或通过指向刚刚在要重组的表所在的表空间中构建的对象来替换原始表对象。
4. *重新创建所有索引*
   重新创建在表上定义的所有索引。

可以使用快照监视器或快照管理视图来监视重组表的进度并确定重组目前处于哪个阶段。

脱机表重组中锁定条件的限制比在联机重组中更多。构建副本期间可以对表进行读访问。但是，在将原始表替换为已重组的副本期间以及在重建索引期间，需要对表进行互斥访问。

整个重组过程中都需要 IX 表空间锁定。在构建阶段，需要 U 锁定并对表挂起该锁定。U 锁定允许锁定所有者更新表中的数据。但是，任何其他应用程序都不能更新任何数

据。但允许进行读访问。替换阶段开始后，U 锁定就升级为 Z 锁定。在此阶段中，任何其他应用程序都不能访问数据。此锁定将被挂起，直到重组阶段完成为止。

脱机重组过程将创建多个文件。这些文件存储在数据库目录中，并且以表空间标识和对象标识作为前缀，例如，0030002.ROR 是表空间标识为 3 且表标识为 2 的表的重组状态文件。

以下是对 SMS 表进行脱机重组期间创建的临时文件:

- .DTR 数据影子副本文件
- .LFR 长字段文件
- .LAR 长字段分配文件
- .RLB LOB 数据文件
- .RBA LOB 分配文件
- .BMR 块对象文件（适用于 MDC 表）

以下临时文件是在索引重组期间创建的:

- .IN1 影子副本文件

下列临时文件是在排序阶段在系统临时表空间中创建的:

- .TDA 数据文件
- .TIX 索引文件
- .TLF 长字段文件
- .TLA 长字段分配文件
- .TLB LOB 文件
- .TBA LOB 分配文件
- .TBM 块对象文件

不应从系统中手动除去与重组过程关联的文件。

以脱机方式重组表:

以脱机方式重组表是整理表碎片的最快方法。重组可减少表所需的空间量并提高数据访问和查询性能。

必须具有 SYSADM、SYSCTRL、SYSMAINT 或 DBADM 权限，或者必须具有对表的 CONTROL 权限才能重组表。必须具有数据库连接才能重组表。

标识需要重组的表之后，可以对这些表运行 REORG 实用程序，并且可以选择对在这些表上定义的任何索引运行该实用程序。

1. 要使用 CLP 重组表，请发出 REORG TABLE 命令:

   ```
   db2 reorg table test.employee
   ```

   要使用临时表空间 mytemp 重组表，请输入:

   ```
   db2 reorg table test.employee use mytemp
   ```

   要重组表并根据索引 myindex 对行进行重新排序，请输入:

   ```
   db2 reorg table test.employee index myindex
   ```

2. 要使用 SQL 调用语句重组表，请使用 ADMIN_CMD 过程发出 REORG TABLE 命令:

   ```
   call sysproc.admin_cmd ('reorg table employee index myindex')
   ```

3. 要使用 DB2 管理 API 重组表，请使用 db2REORG API。

在重组表之后，应收集有关表的统计信息，以便优化器具有最准确的数据来评估查询访问方案。

脱机表重组的恢复:

在替换阶段开始之前，脱机表重组是一个完全成功或完全失败的过程。这表示如果系统在排序或构建阶段崩溃，那么重组表操作将回滚，并且不会在崩溃恢复时重新进行该操作。而是必须在恢复后重新发出 REORG TABLE 命令。

如果系统在进入替换阶段后崩溃，那么重组表操作必须完成。这是因为已完成所有重组表工作，原始表可能不再可用。在崩溃恢复期间，需要已重组对象的临时文件，而不是用于排序的临时表空间。恢复操作将完全重新开始替换阶段，并且需要恢复副本对象中的所有数据。在这种情况下， SMS 表空间与 DMS 表空间之间有一个差别: 必须将 SMS 对象从一个对象复制到另一个对象，而在 DMS 中，如果重组在相同表空间中进行，那么仅指向刚刚重组的对象并废弃原始表。将不重建索引，但在崩溃恢复期间会将索引标记为无效。数据库将根据一般规则确定重建索引的时间: 在数据库重新启动时或第一次访问索引时。

索引重建阶段出现崩溃表示我们已经拥有新对象，因此不重新执行任何操作。如上所述，将不重建索引，但在崩溃恢复期间会将索引标记为无效。数据库将根据一般规则确定重建索引的时间: 在数据库重新启动时或第一次访问索引时。

对于前滚恢复，如果旧版本的表在磁盘上，那么重新进行重组表操作。前滚操作使用在构建阶段记录的 RID 来重新应用创建了已重组表的操作顺序，并重复构建和替换阶段。索引再次被标记为无效，如上所述。这表示未执行索引扫描或扫描排序，并且如果最初使用了临时表空间，那么需要的唯一临时表空间将用于已重组对象的副本。前滚期间，由于进行并行恢复，因此可以同时重新执行多个重组表操作。在这种情况下，使用的磁盘空间将比运行时使用的要多。

提高脱机表重组的性能:

脱机表重组的性能在很大程度上由数据库环境的特征决定。

事实上，以 NO ACCESS 方式运行的重组表操作与以 ALLOW READ ACCESS 方式运行的重组表操作在性能方面没有任何差别。唯一的区分在于，对于 READ ACCESS 方式，DB2 在替换表之前升级对该表的锁定，因此实用程序可能必须等到现有扫描完成并释放其锁定。在这两种方式下，表在重组表操作的索引重建阶段不可用。

有关提高性能的技巧

如果表空间中有足够的空间，那么对原始表和表的已重组副本使用相同的表空间，而不使用临时表空间。这将节省从临时表空间中复制表所用的时间。

- 考虑在重组表之前删除不必要的索引，以便在重组表操作期间维护较少的索引。
- 确保正确设置了已重组的表所在的表空间的预取大小。
- 启用 INTRA_PARALLEL，以便使用并行处理完成索引重建。

- 调整 *sortheap* 和 *sheapthres* 数据库配置参数以控制排序空间。因为每个处理器都将执行私有排序，所以 *sheapthres* 的值至少应为 *sortheap* x *使用的处理器数*。
- 通过调整页清除程序的数目确保尽快从缓冲池中清除脏索引页。

联机表重组

联机或原地表重组允许用户重组表，并同时允许对该表进行完全访问。虽然联机表重组允许用户不中断对数据的访问，但其速度比脱机表重组要慢。

进行联机表重组时，不是立即重组整个表。而是按顺序重组表的各个部分。不会将数据复制到临时表空间：在现有表对象中移动行以重新建立集群、回收可用空间并消除溢出行。

联机表重组包括四个主要阶段:

1. 选择 N 页
   在此阶段中，DB2 将选择 N 页，其中 N 是包含要进行重组表处理的最少 32 个有序页的扩展数据块大小。
2. 腾出范围
   选定 N 页后，联机表重组将此范围内的所有行移至表中的可用页。每个被移动的行都保留一条 RP（重组表指针）记录，该记录包含行的新位置的 RID。将行作为包含数据的 RO（重组表溢出）记录插入到表的可用页中。
   重组表完成移动一组行后，它将等待表中进行的所有现有数据访问（例如，通过当前执行应用程序）完成。这些现有访问称为旧扫描程序，它们在访问表数据时使用旧 RID。在此等待过程中启动的任何访问都称为新扫描程序，它们使用新的 RID 来访问数据。在所有旧扫描程序都完成后，重组表操作将清除已移动的行、删除 RP 记录并将 RO 记录转换为正常记录。
3. 填充范围
   腾出所有行之后，将采用已重组的格式、根据使用的任何索引进行排序后的顺序并遵循定义的任何预取限制写回这些行。填充范围中的所有页后，将在表中选择下一个 N 有序页，并且再次开始该过程。
4. 截断表
   重组表中的所有页后，缺省情况下将截断表以回收空间。如果指定了 NOTRUNCATE 选项，那么不会截断已重组的表。

联机表重组期间创建的文件

联机表重组期间，将为每个数据库分区创建一个 .OLR 状态文件。此文件是名为 xxxxyyy.OLR 的二进制文件，其中 xxxx 是池标识，而 yyyy 是十六进制格式的对象标识。此文件包含从暂停状态继续联机重组所需的信息。

状态文件包括下列信息:

- 重组类型
- 所重组的表的生存 LSN
- 要腾出的下一个范围
- 重组是为了划分数据的集群还是仅回收空间
- 用于划分数据集群的索引的标识

校验和保存在 .OLR 文件中。如果该文件已损坏并导致校验和错误，或者如果表 LSN 与生存 LSN 不匹配，那么必须启动一个新的重组，并且将创建新的状态文件。

如果删除了 .OLR 状态文件，那么重组表过程不能继续，并且会返回 SQL2219 错误。必须启动新的重组表过程。

不应从系统中手动除去与重组过程关联的文件。

以联机方式重组表:

联机或原地表重组允许用户在重组表的同时访问该表。

必须具有 SYSADM、SYSCTRL、SYSMAINT 或 DBADM 权限，或者必须具有对表的 CONTROL 权限才能重组表。必须具有数据库连接才能重组表。

可以使用 CLP 命令、使用 SQL 调用语句或通过 DB2 API 来执行联机表重组。

1. 要使用 CLP 以联机方式重组表，请发出使用 INPLACE 选项的 REORG TABLE 命令:

   ```
   db2 reorg table test.employee inplace
   ```

2. 要使用 SQL 调用语句以联机方式重组表，请使用 ADMIN_CMD 过程发出 REORG TABLE 命令:

   ```
   call sysproc.admin_cmd ('reorg table employee inplace')
   ```

3. 要使用 DB2 管理 API 重组表，请使用 db2REORG API。

在重组表之后，应收集有关表的统计信息，以便优化器具有最准确的数据来评估查询访问方案。

恢复失败的联机表重组:

通常由于磁盘已满或日志记录错误之类的处理错误而导致联机表重组失败。如果联机表重组失败，那么 SQLCA 消息将写入历史记录文件中。

如果分区数据库环境中的一个或多个数据库分区遇到错误，那么返回的 SQL 代码是来自报告错误的第一个节点的 SQL 代码。

如果运行时出现故障，那么联机表重组将暂停并进行回滚。如果系统当机，那么在重新启动时，将开始进行崩溃恢复，并且会暂停并回滚重组。稍后，您可以通过使用 REORG TABLE 命令并指定 RESUME 选项来继续重组。由于完整地记录了联机表重组过程，因此可以保证重组过程可恢复。

例如，在某些情况下，联机表重组可能超过 *num_log_span* 限制。在这种情况下，DB2 将强制执行表重组并使其处于暂停状态。在快照输出中，REORG TABLE 实用程序的状态将显示为“已暂停”。

联机表重组暂停是由中断驱动的，这意味着它可以由用户（通过使用 REORG TABLE 命令的暂停选项或强制执行应用程序命令）或由 DB2 在某些情况下（例如，在系统崩溃时）暂停。

暂停并重新启动联机表重组:

用户可以暂停并重新启动正在进行的联机表重组。

要暂停或重新启动联机表重组，您必须拥有下列其中一项权限或特权:

- sysadm

- sysctrl
- sysmaint
- dbadm
- 对表的 CONTROL 特权。

1. 要暂停联机表重组，请发出使用 PAUSE 选项的 REORG TABLE 命令:

```
db2 reorg table homer.employee inplace pause
```

2. 要重新启动已暂停的联机表重组，请发出带有 RESUME 选项的 REORG TABLE 命令:

```
db2 reorg table homer.employee inplace resume
```

注:

- 暂停联机表重组后，不能开始对该表进行新的重组。必须继续暂停的重组，或者停止暂停的重组，然后再开始新的重组过程。
- 发出 RESUME 请求后，重组过程将沿用在原始 REORG 命令中指定的 TRUNCATE 选项，而无论对后续 START 或任何中间 RESUME 请求指定什么 TRUNCATE 选项。但是，如果重组处于截断阶段并且用户发出指定了 NOTRUNCATE 的 RESUME 请求，那么不会截断表并且重组将完成。
- 在复原和前滚操作后，重组不能继续。

联机表重组的锁定和并行性注意事项:

联机表重组的其中一个重要方面是如何控制锁定，因为它对于应用程序并行性非常重要。

在联机表重组过程中的某个给定时间点，操作可能拥有下列锁定:

- 为了确保能够对表空间进行写访问，获取对重组表操作所影响的表空间的 IX 锁定。
- 在整个重组表操作期间获取并挂起表锁定。锁定级别取决于重组期间允许对该表使用的访问方式:
  如果指定了 ALLOW WRITE ACCESS，那么将获取对表的 IS 锁定。
  如果指定了 ALLOW READ ACCESS，那么将获取对表的 S 锁定。
- 在截断阶段，当重组操作将行移出截断范围时，将请求对表的 S 锁定，因为旧扫描程序（重组表操作期间存在并访问记录的旧 RID 的数据访问）可能会插入新行。DB2 将等到获取此锁定后再开始截断。就在截断阶段结束时重组表操作对表进行物理截断的那一刻，S 锁定将升级为特殊的 Z 锁定。这表示直到任何现有应用程序都不拥有表锁定（包括来自 UR 扫描程序的 IN 锁定）时，重组表操作才完成。
- 还可以根据表锁定类型获取行锁定:
  如果拥有对表的 S 锁定，那么不需要单独的行级别 S 锁定，因此不需要进一步锁定。
  如果拥有对表的 IS 锁定，那么在移动行之前将获取行级别 S 锁定，并在移动完成后释放该锁定。
- 可能需要内部锁定来控制对联机表重组对象和其他联机 DB2 实用程序（如联机备份）的访问。

锁定会影响重组表和并发用户应用程序的性能。强烈建议您在执行联机表重组时检查锁定快照数据以了解锁定活动。

监视表重组

可以使用 GET SNAPSHOT 命令、SNAPTAB_REORG 管理视图或 SNAP_GET_TAB_REORG 表函数获取有关表重组操作的状态的信息。

必须连接至数据库并且具有下列权限:

- SYSMON 权限
- 对 SNAPTAB_REORG 管理视图的 SELECT 或 CONTROL 特权，或者对 SNAP_GET_TAB_REORG 表函数的 EXECUTE 特权。
- 要使用 SQL 访问有关重组操作的信息，请使用 SNAPTAB_REORG 管理视图。 例如，以下 select 语句返回有关在当前连接的数据库上对所有数据库分区执行的表重组操作的详细信息:

  ```
  SELECT SUBSTR(TABNAME, 1, 15) AS TAB_NAME, SUBSTR(TABSCHEMA, 1, 15)
     AS TAB_SCHEMA, REORG_PHASE, SUBSTR(REORG_TYPE, 1, 20) AS REORG_TYPE,
     REORG_STATUS, REORG_COMPLETION, DBPARTITIONNUM
     FROM SYSIBMADM.SNAPTAB_REORG ORDER BY DBPARTITIONNUM
  ```

  如果未重组任何表，那么返回 0 行。
- 要使用快照监视器访问有关重组操作的信息，请发出 GET SNAPSHOT FOR TABLES ON *database* 命令并检查表重组监视元素（具有前缀“reorg_”）的值。

由于脱机表重组是同步的，所以脱机表重组中出现的任何错误都会返回给实用程序的调用者（应用程序或命令行）。

联机表重组中的错误处理与脱机表重组中的错误处理略有不同。由于联机表重组是异步的，所以没有 SQL 消息写入 CLP。要查看在联机表重组期间返回的 SQL 错误，请发出“LIST HISTORY REORG”命令。

联机表重组作为 db2Reorg 进程在后台运行。这意味着从重组表进程成功返回到调用应用程序并不表示重组已成功完成。即使调用应用程序终止其数据库连接， db2Reorg 进程也会继续。

索引重组

通过删除和插入操作对表进行更新后，索引的性能会降低，其表现方式如下:

- 叶子页分段

  叶子页被分段之后，由于必须读取更多的叶子页才能访存表页，因此 I/O 操作成本会增加。
- 物理索引页的顺序不再与这些页上的键顺序相匹配（此称为*不良集群*索引）。

  叶子页出现不良集群情况后，顺序预取操作的效率将降低，因此会导致更多的 I/O 等待。
- 形成的索引大于其最有效的级别数。

  在此情况下应重组索引。

如果在创建索引时设置了 MINPCTUSED 参数，那么在删除某个键且可用空间小于指定的百分比时，数据库服务器会自动合并索引叶子页。此过程称为*联机索引整理碎片*。但是，要复原索引集群和可用空间以及降低叶级别，请使用下列其中一种方法:

- 删除并重新创建索引。

- 使用 REORG INDEXES 命令联机重组索引。

  因为此方法允许用户在重建表索引期间对表进行读写操作，所以在生产环境中可能需要选择此方法。
- 使用允许脱机重组表及其索引的选项运行 REORG TABLE 命令。

联机索引重组

在使用 ALLOW WRITE ACCESS 选项运行 REORG INDEXES 命令时，如果同时允许对指定的表进行读写访问，那么会重建该表的所有索引。进行重组时，对基础表所作的任何将会影响到索引的更改都将记录在 DB2 日志中。另外，如果有任何内部内存缓冲区空间可供使用，那么还将这些更改放在这样的内存空间中。重组将处理所记录的更改以便在重建索引时与当前写活动保持同步更新。内部内存缓冲区空间是根据需要从实用程序堆中分配的指定内存区域，它用来存储对正在创建或重组的索引所作的更改。使用内存缓冲区空间使索引重组操作能够通过这样的方式来处理更改，即先直接从内存读取，然后读取日志（如有必要），但读取日志的时间要晚得多。在重组操作完成后，将释放所分配的内存。重组完成后，重建的索引可能不是最佳集群的索引。如果为索引指定 PCTFREE，那么在重组期间，每页上均会保留相应百分比的空间。

对于分区表，支持对各个索引进行联机索引重组和清除。要对各个索引进行重组，指定索引名：`REORG INDEX index_name for TABLE table_name`

对于空间索引或多维集群（MDC）表，不支持采用 ALLOW WRITE 方式的联机索引重组。

注：REORG INDEXES/INDEX 命令的 CLEANUP ONLY 选项不能完全重组索引。CLEANUP ONLY ALL 选项将除去那些标记为“删除”且被认为要落实的键。此外，它还将释放所有标记为“删除”且被认为要落实的键所在的页。在释放页后，相邻的叶子页将会合并，前提是这样做可以在合并页上至少留出 PCTFREE 可用空间。PCTFREE 是指在创建索引时为其定义的可用空间百分比。CLEANUP ONLY PAGES 选项仅删除那些标记为“删除”且被认为要落实的所有键所在的页。

使用 CLEANUP ONLY 选项对分区表的索引进行重组时，支持任何访问级别。如果未指定 CLEANUP ONLY 选项，那么缺省访问级别 ALLOW NO ACCESS 是唯一支持的访问级别。

索引重组具有下列要求:

- 对索引和表具有 SYSADM、SYSMAINT、SYSCTRL 或 DBADM 权限，或者具有 CONTROL 特权。
- 用于存储索引的表空间的可用空间数量等于索引的当前大小

  在发出 CREATE TABLE 语句时，考虑在大型表空间中重组索引。
- 其他日志空间

  索引重组需要记录其活动。因此，重组可能会失败，尤其是在系统繁忙和记录其他并发活动时。

  注：如果具有 ALLOW NO ACCESS 选项的 REORG INDEXES ALL 命令运行失败，那么会标记索引无效并且此项操作不可撤销。但是，如果具有 ALLOW READ

ACCESS 选项的 REORG 命令或具有 ALLOW WRITE ACCESS 选项的 REORG 命令运行失败，那么可以复原原来的索引对象。

确定何时重组表和索引

在对表数据进行许多更改之后，逻辑上连续的数据可能会位于不连续的物理数据页上，在许多更新操作创建了溢出记录时尤其如此。按这种方式组织数据时，数据库管理器必须执行其他读操作才能访问顺序数据。另外，在删除大量行后，也需要执行其他的读操作。

表重组操作会整理数据碎片来减少浪费的空间，并对行进行重新排序以合并溢出记录，从而加快数据访问速度并最终提高查询性能。还可以指定根据特定索引来重新排序数据，以便查询通过最少次数据读取操作就可以访问数据。

对表数据进行大量更改将导致更新索引并使索引性能下降。索引叶子页可能变成碎片或出现不良集群情况，并且索引可能形成比所需层次要多的层次以获得最佳性能。所有这些问题都会产生更多 I/O 并导致性能下降。

下列任何因素都可能指示您应该重组表或索引:

- 自上次重组表之后，对该表进行了大量的插入、更新和删除活动
- 对于使用具有高集群率的索引的查询，其性能发生了明显变化
- 在执行 RUNSTATS 以刷新统计信息后，性能没有得到改善
- REORGCHK 命令指示需要重组表或索引（注意: 在某些情况下，REORGCHK 总是建议重组表，即使在执行了重组后也是如此）。例如，如果使用 32KB 页大小，并且平均记录长度为 15 字节且每页的最多包含 253 条记录，那么每页具有 32700- (15 x 253)=28905 个不可用字节。这意味着大约 88% 的页面是可用空间。用户应分析 REORGCHK 的建议并针对执行重组所需的成本平衡利益。
- db.tb_reorg_req（需要重组）运行状况指示器处于 ATTENTION 状态。此运行状况指示器的集合详细信息描述通过重组可获得好处的表和索引的列表。

REORGCHK 命令返回有关数据组织的统计信息，并且可以在是否需要重组特定表或索引这一问题上为您提供建议。然而，定期或在特定时间对目录统计信息表运行特定查询可以提供性能历史记录，该记录使您可以发现可能具有更广性能隐含的趋势。

要确定是否需要重组表或索引，查询目录统计信息表并监视下列统计信息:

1. 行的溢出

   查询 SYSSTAT.TABLES 视图中的 OVERFLOW 列以监视溢出值。此列中的值表示不适合其原始页的行数。当表中的可变长度列导致记录长度扩展，以致它们不再适合放入数据页上的指定位置时，行数据可能溢出。在列添加至表定义并稍后通过更新行来具体化该列时，长度也可能会更改。在这些情况下，在行中的原始位置将保留一个指针，而实际值存储在由指针指示的另一个位置。这可能会影响性能，因为数据库管理器必须根据指针来查找行的内容。这个包括两个步骤的过程增加了处理时间，并且可能还会增加需要的 I/O 数目。

   重组表数据将消除行溢出；因此，随着溢出行数的增加，重组表数据就可能会具有更多好处。
2. 访存统计信息

当以索引顺序访问表时，查询 SYSCAT.INDEXES 和 SYSSTAT.INDEXES 目录统计信息表中的以下三个列来确定预取程序的效率。这些统计信息对照基础表来体现预取程序的平均性能的特征。

- AVERAGE_SEQUENCE_FETCH_PAGES 列存储可以按表中的顺序访问的平均页数。可以按顺序访问的页适合于预取。较小的数目指示预取程序没有充分发挥作用，原因是它们不能读入由表空间的 PREFETCHSIZE 设置指定的所有页数。较大的数指示预取程序将有效地执行。对于集群索引和表，此数目应该接近 NPAGES（包含一些行的页数）的值。
- AVERAGE_RANDOM_FETCH_PAGES 列存储当使用索引来访存表行时在顺序页访问之间的平均随机表页数。当大多数页按顺序时，预取程序忽略少量的随机页，并按已配置的预取大小继续预取。当表变得更加混乱时，随机访存页数就会增加。通常由于在表的末尾或在溢出页中发生了无序的插入导致这样的无组织。当使用索引来访问某一范围内的值时，这会导致降低查询性能的访存。
- AVERAGE_SEQUENCE_FETCH_GAP 列存储当使用索引进行访存时表页序列之间的平均间隔。通过扫描索引叶子页来进行检测，每个间隔表示在表页的各个序列之间必须随机访存的平均表页数。当随机访问许多页时发生这些情况，这会中断预取程序。较大的数指示无组织的表或较差集群的索引。

3. 包含标记为已删除但未除去的 RID 的索引叶子页数

   在 2 类索引中，当 RID 标记为删除时，通常未在物理上删除 RID。这意味着可用空间可能会被这些逻辑上已删除的 RID 占用。要检索每个 RID 都被标记为已删除的叶子页的数目，查询 SYSCAT.INDEXES 和 SYSSTAT.INDEXES 统计信息表的 NUM_EMPTY_LEAFS 列。对于并非所有 RID 都标记为删除的叶子页，逻辑上已删除的 RID 的总数存储在 NUMRIDS_DELETED 列中。

   使用此信息来通过执行带 CLEANUP ALL 选项的 REORG INDEXES 估计可以回收多少空间。要想只回收所有 RID 都被标记为已删除的页中的空间，执行带有 CLEANUP ONLY PAGES 选项的 REORG INDEXES。

4. 索引的集群比率和集群因子统计信息

   集群比率统计信息存储在 SYSTCAT.INDEXES 目录表的 CLUSTERRATIO 列中。此值（在 0 到 100 之间）表示使用索引的数据集群的程度。如果收集 DETAILED 索引统计信息，0 和 1 之间的好的集群因子统计信息存储在 CLUSTERFACTOR 列中，并且 CLUSTERRATIO 的值为 -1。这两个集群统计信息中只有一个可以记录在 SYSCAT.INDEXES 目录表中。要将 CLUSTERFACTOR 值与 CLUSTERRATIO 值进行比较，可以将 CLUSTERFACTOR 乘以 100 以获得一个百分比。

   注：通常，表中只有一个索引可以具有较高的集群度。

   不是只索引访问的索引扫描在具有较高集群比率的情况下可能执行得更好。低的集群率导致此类扫描要执行更多的 I/O，因为在每个数据页经过第一次访问后，下次访问该页时，该页仍在缓冲池中的可能性减小。增大缓冲区大小也可以提高非集群索引的性能。

   如果表数据最初是对某个索引集群的，而集群统计信息指示现在很少为同一索引集群数据，那么您可能想重组该表以再次集群数据。

5. 叶子页数

查询 SYSCAT.INDEXES 表中的 NLEAF 列以便了解索引占用的叶子页数目。该数目告诉您对索引进行完整的扫描需要多少索引页 I/O。

理想情况下，索引应该尽可能占用最小的空间量，以便减少进行索引扫描所需要的 I/O 次数。随机的更新活动可导致页分割，这就增大了索引的大小。当在重组表期间重建索引时，可以使用最小空间量来构建每个索引。

注: 缺省情况下，当构建索引时，会在每个索引页上保留百分之十的可用空间。要增加可用空间量，当创建索引时指定 PCTFREE 参数。无论何时重组索引，都使用 PCTFREE 值。因为附加空间可以容纳附加索引插入，所以大于百分之十的可用空间可能缩小索引重组的频率。

6. 空数据页的数目

   要计算表中的空页数，查询 SYSCAT.TABLES 中的 FPAGES 和 NPAGES 列并从 FPAGES 数减去 NPAGES 数。FPAGES 列存储正在使用的总页数; NPAGES 列存储包含一些行的页数。当删除整个范围内的行时，可能出现空页。

   随着空页数的增多，就更需要进行表重组。重组表时将回收空页并减少表所使用的空间量。另外，因为会将空页读入缓冲池以进行表扫描，所以回收未使用的页可以提高表扫描的性能。

   当表中的总页数（FPAGES）小于或等于 NPARTITIONS * 1 个扩展数据块大小时，不建议进行表重组。如果是分区表，那么 NPARTITIONS 为数据分区数，否则为 1。在分区数据库环境中，将表的数据库分区组中的数据库分区数作为因子后，情况变为 FPAGES <= 表的数据库分区组中的数据库分区数 * NPARTITIONS * 1 个扩展数据块大小

在重组之前，请综合考虑查询性能不断降低所浪费的成本和重组表或索引所需的成本（包括 CPU 时间、耗用时间和 REORG 实用程序在完成重组操作之前锁定表造成的并行性降低），以确定是否进行表重组。

重组表和索引的成本

决定是否要重组表或索引时，必须考虑对这些对象执行重组所产生的一些开销。

重组表和索引的成本包括:

- 执行实用程序消耗的 CPU
- 运行 REORG 实用程序时并行性降低。由于重组要求进行锁定而导致并行性降低。
- 需要额外的存储器。（脱机表重组需要额外存储空间来保存表的影子副本。联机或原地表重组需要其他日志记录空间。联机索引重组需要额外存储空间来保存索引的影子副本和其他日志空间。脱机索引重组将使用较少的日志空间并且不涉及影子副本。）

在某些情况下，已重组的表可能比原始表要大，从而相应地增加了空间要求。在下列情况下，重组后表可能会增大:

- 在使用索引来确定行顺序的集群重组表中，如果表记录的长度可变（例如，使用 varchar)，那么重组结束时您可能会使用更多空间，因为某些页中包含的行数可能比原始表中的行数要少。
- 如果在重组之前但在创建表之后在表中添加了列，并且重组之后在某些行中可能是第一次实现添加的列。

- 如果自上次重组后每页上剩余的可用空间大小（由 PCTFREE 属性的值表示）已增加。
- 如果表包含 LOB，并且该 LOB 可能比以前使用更多的空间。

脱机表重组的空间要求

由于脱机重组使用影子副本方法，因此需要足够的其他存储器来容纳表的另一个副本。在原始表所在的表空间中或用户指定的临时表空间中构建影子副本。

如果使用表扫描排序，那么可能需要其他临时表空间存储器来进行排序处理。需要的其他空间可能与要重组的表一样大。如果集群索引是 SMS 类型或唯一的 DMS 类型，那么重新创建此索引不需要排序。然而，将通过扫描刚刚重组的数据来重建此索引。任何其他需要重新创建的索引都将涉及排序，并可能需要多达要重组的表大小的临时表空间。

脱机表重组生成少量控制日志记录，因此消耗相对较少的日志空间。如果重组未使用索引，那么唯一的日志记录是表数据日志记录。如果指定了索引，或者如果表上存在集群索引，那么按照将行放入新版本的表中的顺序记录这些行的 RID。每条 RID 日志记录最多包含 8000 个 RID，每个 RID 消耗 4 字节。这可能是导致在脱机表重组期间用完日志空间的一个因素。请注意，仅当数据库可恢复（LOGRETAIN=ON）时，才记录 RID。

联机表重组的日志空间要求

联机表重组所需的日志空间通常比脱机表重组所需的日志空间要大。需要的空间大小由要重组的行数、索引数、索引键的大小以及最开始表的组织情况决定。为用于表的日志空间确定一个典型的基准是一种不错的做法。

表中的每行都可能在联机表重组期间移动两次。如果提供一个索引，那么每行必须更新索引键以反映新位置，并且在完成所有对旧位置的访问后，将再次更新索引键以除去对旧 RID 的引用。移回行时，将再次执行对索引键的这些更新。将记录所有这些活动以便联机表重组完全可恢复，因此最少应有两条数据日志记录（包括行数据的每个实例）和四条索引日志记录（包括键数据的每个实例）。集群索引尤其容易填满索引页，从而导致索引分割和合并，这些分割和合并活动也必须进行记录。

由于联机表重组经常发出内部落实，所以它通常不会拥有重要的活动日志。如果在某个时间联机重组拥有大量活动日志，那么必定是在截断阶段，因为截断阶段需要 S 表锁定。如果表重组无法获取该锁定，那么它将等待并保留日志，而其他事务可能会快速填满日志。

减少重组表和索引的需要

您可以使用不同的策略来降低需要重组表和索引的频率，从而避免执行不必要的重组操作的成本。

减少重组表的需要

- 使用多分区表。表越小，需要重组的可能性就越小。
- 创建多维集群（MDC）表，将在 CREATE TABLE 语句的 ORGANIZE BY DIMENSION 子句中所指定的列中自动维护该表的集群。

- 对表打开 APPEND 方式。例如，如果这些新行的索引键值总是新的大键值，那么表的集群属性将尝试将其放置在表的末尾。在这种情况下，将表置于追加方式可能优于集群索引。
- 在创建表之后:
  - 改变表以添加 PCTFREE
  - 在索引上使用指定的 PCTFREE 来创建集群索引。
  - 将数据装入到表中之前对数据进行排序

  表上正确设置了 PCTFREE 的集群索引有助于保持原始排序顺序。当表页上有足够的空间时，可以将新数据插入正确的页，以维护索引的集群特征。随着更多的数据插入，表页会因此而变满，记录会被追加至表的末尾，因而表将逐渐失去集群特性。

  如果在创建集群索引后执行 REORG TABLE 命令，或者执行排序和 LOAD 命令，那么索引会尝试维护数据的特定顺序，这将改善 RUNSTATS 实用程序所搜集的 CLUSTERRATIO 或 CLUSTERFACTOR 统计信息。

减少重组索引的需要

- 在索引页上使用 PCTFREE 或 LEVEL2 PCTFREE 创建集群索引。范围在 0 到 99% 之间，缺省值为 10%。
- 使用 MINPCTUSED 创建索引。可能的范围在 0 到 99% 之间，建议的值为 50%。但是，请考虑使用 REORG INDEXES 命令的 CLEANUP ONLY ALL 选项来合并叶子页。

自动重组

在对表数据进行许多更改后，表和索引可能变成碎片。逻辑上连续的数据可能位于不连续的物理页面上，因此数据库管理器必须执行其他的读操作来访问数据。

由 RUNSTATS 收集的统计信息与其他信息一起来显示表中的数据分布情况。特别是，通过分析这些统计信息可以知道何时需要执行哪种类型的重组。自动重组通过使用 REORGCHK 公式来确定是否需要对表和索引进行重组。它会定期评估已经更新了统计信息的表和索引，以便了解是否需要重组。如果需要重组，那么它会在内部调度索引重组或对表进行传统表重组。这将要求执行应用程序功能而不对正在重组的表进行写访问。

可以使用 AUTO_REORG、AUTO_TBL_MAINT 和 AUTO_MAINT 数据库配置参数来启用或禁用自动重组功能部件。

在分区数据库环境中，确定执行自动重组和启动自动重组是在目录分区上完成的。只需要在目录分区上启用数据库配置参数。将在目标表所在的所有数据库分区上运行重组。

如果您不太确定何时以及如何重组表和索引，那么可以将自动重组作为整个数据库维护方案的一部分。

可以使用控制中心或运行状况中心中的“自动维护”向导来配置要进行自动重组的表和索引。

启用表和索引的自动重组

要进行高效率的数据访问和获得最佳工作负载性能，具有组织良好的表和索引数据是很关键的。对表数据进行大量更改之后，逻辑顺序数据可能在非顺序物理数据页，因此数据库管理器必须执行其他读操作来访问数据。另外，在删除大量行后，也需要执行其他的读操作。使用自动表重组来启用 DB2 管理脱机和索引重组，以使您无需担忧何时使用何种方法来重组数据。可启用 DB2 来重组系统目录表以及数据库表。

可以使用图形用户界面工具或命令行界面来打开此功能。

- 要使用图形用户界面工具来将数据库设置为自动重组:
  1. 通过以下两种方法来打开“配置自动维护”向导: 一种方法是在“控制中心”中右键单击数据库对象; 另一种方法是在“运行状况中心”中右键单击要配置自动重组的数据库实例。从弹出窗口中选择**配置自动维护**。
  2. 在此向导中，可以启用自动重组来整理数据碎片、指定想要自动重组的表以及指定用于执行 REORG 实用程序的维护窗口。
- 要使用命令行界面来将数据库设置为自动重组，请将下列配置参数设置为“ON”:
  - AUTO_MAINT
  - AUTO_TBL_MAINT
  - AUTO_REORG

使用关系索引来提高性能

访问表数据时，可使用索引来提高性能。在处理关系数据时可使用关系索引。对于 XML 数据访问，使用基于 XML 数据的索引。

尽管优化器决定是否使用关系索引来访问关系表数据，但除了下列情况以外，您必须决定哪些索引可以改善性能并创建这些索引。例外情况有维块索引和组合块索引，当创建多维集群（MDC）表时，为指定的每一维自动创建这些索引。

也必须在下列情况下执行 RUNSTATS 实用程序来收集关于关系索引的新的统计信息:

- 在创建关系索引之后
- 在更改预取大小之后

还应该定期执行 RUNSTATS 实用程序来保持统计信息为当前的。如果没有有关索引的最新统计信息，那么优化器不能确定查询的最佳数据访问方案。

注: 要确定是否在特定程序包中使用关系索引，使用说明工具。要计划关系索引，使用“控制中心”中的“设计顾问程序”或者 `db2advis` 工具来获取有关可以由一个或多个 SQL 语句使用的关系索引的建议。

关系索引比无索引的优点

如果表上不存在索引，那么必须对 SQL 查询中引用的每个表执行表扫描。表越大，表扫描所花的时间越长，因为表扫描需要顺序访问每个表行。虽然对于需要表中的大多数行的复杂查询来说，使用表扫描效率可能更高，但对于只返回部分表行的查询，索引扫描可以更有效地访问表行。

如果在 SELECT 语句中引用了关系索引列且优化器估计索引扫描比表扫描快，那么优化器选择索引扫描。索引文件一般较小，因此读取它所需的时间比读取整个表所需的时间要少，尤其在表增大时更是如此。此外，可能不需要扫描整个索引。应用于索引的谓词减少了要从数据页读取的行数。

如果对输出的排序需求可以与索引列相匹配，那么按列顺序扫描索引将允许按正确顺序检索行而不需要排序。

每个关系索引条目包含一个搜索键值和一个指向包含该值的行的指针。如果在 CREATE INDEX 语句中指定了 ALLOW REVERSE SCANS 参数，那么可以按升序和降序搜索这些值。因此，在具有正确谓词的情况下，才可能对搜索分类。也可使用关系索引来获得已排序的行，使数据库管理器在从表中读取这些行之后不必对它们排序。

除搜索键值和行指针外，关系索引还可以包括包含列，这些列是索引行中的非索引列。这样的列有可能使优化器仅从索引获取所需要的信息，而不必访问表本身。

注: 要查询的表上存在关系索引并不保证结果集已排序。只有 ORDER BY 子句才确保结果集的排序。

尽管索引可显著缩短访问时间，但是它们也可给性能带来负面影响。在创建索引之前，考虑多个索引给磁盘空间和处理时间带来的影响:

- 每个索引都需要存储器或磁盘空间。准确的容量取决于表的大小以及关系索引中的列的大小和数目。
- 对一个表执行的每个 INSERT 或 DELETE 操作都需要对该表上的每个索引进行额外的更新。对于更改索引键值的每个 UPDATE 操作，也是如此。
- LOAD 实用程序重建任何现有的关系索引或追加至现有的关系索引之后。

  可在 LOAD 命令上指定 `indexfreespace` MODIFIED BY 参数，以覆盖创建关系索引时使用的索引 PCTFREE。
- 每个关系索引都有可能对 SQL 查询添加备用访问路径以供优化器考虑，这会增加编译时间。

谨慎选择索引来满足应用程序的需要。

关系索引的计划提示

创建的关系索引应该取决于关系数据和访问该数据的查询。

使用“控制中心”中的“设计顾问程序”或者 `db2advis` 工具来查找特定查询或定义工作负载的一组查询的最佳索引。此工具为关系索引建议了这样一些性能提高功能部件: INCLUDE 列、继承的唯一索引和 ALLOW REVERSE SCANS 索引。

以下准则可帮助您确定如何创建可用于各种目的关系索引:

- 要避免某些排序，只要有可能，就通过使用 CREATE UNIQUE INDEX 语句定义主键和唯一键。
- 要改善数据检索，将 INCLUDE 列添加至唯一索引。合适的列为:
  - 被频繁访问，因此可从纯索引访问中受益的列。
  - 不需要用来限制索引扫描的范围的列
  - 不影响索引键的排序或唯一性的列。

- 要有效访问小表，使用关系索引来优化对含有较多数据页的表的频繁查询，数据页数记录在 SYSCAT.TABLES 目录视图的 NPAGES 列中。您应该:
  - 根据连接表时要使用的任何一列来创建索引。
  - 根据将用于定期搜索特定值的任何列来创建索引。
- 要有效地搜索，决定对键使用升序还是降序，这取决于将最常使用的顺序。尽管当在 CREATE INDEX 语句中指定了 ALLOW REVERSE SCANS 参数时可以按逆向方向搜索值，但是，执行按指定索引顺序的扫描比执行逆向扫描稍微更快一些。
- 要保存索引维护成本和空间:
  - 避免创建的关系索引是这些列上其他索引键的部分键。例如，如果列 a、b 和 c 上有索引，那么列 a 和 b 上的第二个索引一般用处不大。
  - 不要在所有列上任意创建关系索引。不必要的索引不仅使用空间，而且导致大量准备时间。当使用具有动态编程连接枚举的优化级别时，这对于复杂的查询特别重要。

    使用下列一般规则来确定将为表定义的关系索引的典型数目。此数目根据数据库的主要使用来确定:
    - 对于联机事务处理（OLTP）环境，创建一个或两个索引
    - 对于只读查询环境，可以创建 5 个以上索引
    - 对于混合查询和 OLTP 环境，可以创建 2 到 5 个索引。
- 要提高对父表执行删除和更新操作的性能，在外键上创建关系索引。
- 要提高涉及到 IMMEDIATE 和 INCREMENTAL MQT 的 DELETE 和 UPDATE 操作的性能，对 MQT 的隐含唯一键创建唯一关系索引，该隐含唯一键是 MQT 定义的 GROUP BY 子句中的列。
- 对于快速排序操作，在频繁用于排序关系数据的列上创建关系索引。
- 要提高多列关系索引的连接性能，如果第一个键列有多项选择，那么使用最常用“=”（等值连接）谓词指定的那一列，或使用如第一个键那样具有最多单值的那些列。
- 要帮助新插入的行根据索引进行集群并避免页分割，定义一个集群索引。集群索引应显著减少重组表的需要。

  当定义表时使用 PCTFREE 关键字来指定页上应该留下多少可用空间，才能允许将插入行适当地放在页上。也可以指定 LOAD 命令的 pagefreespace MODIFIED BY 子句。
- 要启用联机索引整理碎片，创建关系索引时使用 MINPCTUSED 选项。MINPCTUSED 指定索引叶子页中最小使用空间量的阈值并启用联机索引整理碎片。如果这些删除实际上从索引页除去键，那么这可以在键删除期间以性能损失为代价而减少重组的需要。

在下列情况下，考虑创建关系索引:

- 在最频繁处理的查询和事务的 WHERE 子句中所使用的那些列上创建关系索引。

  以下的 WHERE 子句:

  ```
  WHERE WORKDEPT='A01' OR WORKDEPT='E21'
  ```

  通常将会从 WORKDEPT 上的索引获益，除非 WORKDEPT 列包含许多重复值。
- 在将按查询所需要的顺序对行排序的一列或多列上创建关系索引。不仅在 ORDER BY 子句中，而且其他功能，如 DISTINCT 和 GROUP BY 子句都需要排序。

以下示例使用 DISTINCT 子句:

```
SELECT DISTINCT WORKDEPT
       FROM EMPLOYEE
```

数据库管理器可使用 WORKDEPT 上定义为升序或降序的索引来消除重复值。此同一个索引也可用于 GROUP BY 子句中，以将值分组，如以下示例所示:

```
SELECT WORKDEPT, AVERAGE(SALARY)
       FROM EMPLOYEE      GROUP BY WORKDEPT
```

- 使用复合键创建关系索引，该键命名语句中引用的每个列。当用此方式指定索引时，可以从纯索引检索关系数据，这比访问表更有效。

  例如，考虑下列 SQL 语句:

  ```
  SELECT LASTNAME
         FROM EMPLOYEE          WHERE WORKDEPT IN ('A00','D11','D21')
  ```

  如果为 EMPLOYEE 表的 WORKDEPT 和 LASTNAME 列定义了关系索引，那么通过扫描索引而不扫描整个表可能会更有效地处理该语句。注意，因为该谓词基于 WORKDEPT，因此此列应是该关系索引的第一列。

- 使用 INCLUDE 列创建关系索引可改善表上索引的使用。使用上述示例，可将唯一关系索引定义为:

  ```
  CREATE UNIQUE INDEX x ON employee (workdept) INCLUDE (lastname)
  ```

  指定 lastname 为 INCLUDE 列而不是索引键的一部分，意味着 lastname 只存储在索引的叶子页上。

关系索引的性能提示

考虑关于使用和管理关系索引的下列建议:

- **指定大型实用程序堆**

  CREATE INDEX 和 REORG INDEXES 都支持其他用户或其他应用程序对基础表进行写访问。如果您创建或重组的关系索引时需要对基础表执行大量更新活动，请考虑配置大型实用程序堆。在同步更新阶段，大型实用程序堆将提高创建或重组索引的速度。对正在创建或重组的索引执行的所有写活动都记录在 DB2 日志和内部内存缓冲区空间中。内部内存缓冲区空间是根据需要从实用程序堆中分配的指定内存区域，它用来存储对正在创建或重组的索引所作的更改。使用该内存区域可以使同步更新阶段能够更快地工作。在创建或重组操作完成后，将释放所分配的内存。如果确保有足够的实用程序堆来容纳对正在创建或重组的索引进行的全部或大部分更改，那么会对同步更新阶段的性能产生非常积极的影响。

- **在 SMP 机器上运行时增大 sheapthres 配置参数**

  每个子代理程序将获取 **sortheap** 配置参数所指定的内存量，以便在扫描表时避免排序溢出。您应监视排序溢出数并相应地增大 **sheapthres**。

- **为关系索引指定不同的表空间**

  可将索引存储在与表数据不同的表空间中。这样，通过减少索引访问期间读/写磁头的移动，可以更有效地使用磁盘存储器。也可以在更快的物理设备上创建索引表空间。而且，可以将索引表空间指定给不同的缓冲池，由于它们不与表数据页竞争，这可以将索引页较长时间保存在缓冲区内。

当不将索引放置在单独的表空间中时，数据页和索引页使用相同的扩展数据块大小和预取量。如果对索引使用不同的表空间，那么可为表空间的所有特征选择不同的值。因为索引通常比表小，且分布在更少的容器中，因此索引经常只具有较小的扩展数据块大小，如 8 页和 16 页。当查询优化器选择访问方案时，它考虑用于表空间的设备的速度。

- **确保集群度**

  如果 SQL 语句需要排序（例如，它包含 ORDER BY、GROUP BY 和 DISTINCT SQL 子句），那么即使索引可以满足排序，但是在下列情况下，优化器可能不会选择索引:

  – 索引集群程度较低。有关信息请查看 SYSCAT.INDEXES 的 CLUSTERRATIO 和 CLUSTERFACTOR 列。
  – 由于表很小，因此扫描该表并在内存中对答案集进行排序所消耗的成本更低。
  – 有多个索引可供访问该表。

  创建集群索引之后，以传统方式执行 REORG TABLE，这可创建组织完好的索引。要重新集群表，可以执行排序和装入来代替。通常，只能在一个索引上集群表。在构建集群索引之后构建其他索引。集群索引试图维护数据的特定顺序，以改进 RUNSTATS 实用程序收集的 CLUSTERRATIO 或 CLUSTERFACTOR 统计信息。

  要帮助维护集群比率，在装入或重组表之前，在改变表时指定适当的 PCTFREE。PCTFREE 指定的每页上的可用空间提供用于插入的空间，以便可以适当地集群这些插入。如果不为表指定 PCTFREE，那么重组会消除所有额外空间。

  注: 目前，在更新期间不维护集群，除非您正在使用范围集群表。即，如果更新了一条记录，其键值在集群索引中发生了更改，那么不必将该记录移动至新页来维护集群顺序。要维护集群，可使用 DELETE，然后使用 INSERT，而不是使用 UPDATE。

- **使表和索引统计信息保持最新**

  当创建新关系索引后，运行 RUNSTATS 实用程序来收集索引统计信息。这些统计信息使优化器能够确定使用索引是否能提高访问性能。

- **启用联机索引整理碎片**

  如果将 MINPCTUSED 子句设置为大于零，那么对关系索引启用联机索引整理碎片。当页上的可用空间为指定级别或低于指定级别而索引仍可用时，联机索引整理碎片允许通过合并叶子页来压缩索引。

- **必要时重组关系索引**

  要从索引中获得最佳性能，请考虑定期重组索引，这是因为对表进行更新会使索引页预取效率降低。

  要重组索引，删除它并重新创建它，或者使用 REORG 实用程序。

  要减小对频繁重组的需要，当创建关系索引时，指定适当的 PCTFREE 来在创建的每个索引叶子页上保留一定百分比的可用空间。在将来的活动中，可将记录插入索引，从而使索引页分割的可能性较小。页分割导致索引页既不是连续的也不是按顺序的，从而导致索引页预取效率降低。

注: 在重组关系索引时，保留创建该索引时指定的 PCTFREE。

删除并重新创建或重组关系索引也会创建大致连续且按顺序排列的一组新页，并改善索引页预取。尽管时间和资源方面的成本更高，但 REORG TABLE 实用程序也确保数据页的集群。集群为访问大量数据页的索引扫描带来很大的好处。

在对称多处理器（SMP）环境中，“传统的”重组表方式（该方式使用影子表进行快速表重组）可以使用多个处理器来重建索引。

- **分析关于关系索引使用情况的 EXPLAIN 信息**

  定期对您最频繁使用的查询运行 EXPLAIN，并验证每个关系索引是否至少使用了一次。如果有一个索引未在任何查询中使用，考虑删除该索引。

  EXPLAIN 信息也允许您查看在大型表上的表扫描是否是作为嵌套循环连接的内部表来处理的。如果是这样，那么连接谓词列上的索引或者丢失，或者被认为应用于该连接谓词的效率不高。

- **对大小变化范围大的表使用易失表**

  易失表是其大小在运行时可以从空变为很大的一种表。对于这种表，其中的基数变化范围很大，优化器可能会生成适合表扫描而不是索引扫描的访问方案。

  使用 ALTER TABLE...VOLATILE 语句声明表是“易失的”，这将允许优化器对易失表使用索引扫描。在下列情况下，无论统计信息如何，优化器将使用索引扫描而不是使用表扫描:

  – 引用的所有列都在索引中
  – 索引可以在索引扫描中应用谓词。

  如果该表是一个类型表，那么在该类型表层次结构的根表中仅支持使用 ALTER TABLE...VOLATILE 语句。

索引清除和维护

在创建索引之后，除非使索引保持压缩和有组织，否则性能会下降。考虑以下建议使索引尽可能小并效率高:

- 启用联机索引整理碎片

  使用 MINPCTUSED 子句创建索引。如果必要，删除并重新创建现有的索引。

- 如果频繁的 COMMITS 不可能，那么显式或通过锁定升级来执行频繁的 COMMIT 或获取对表的 X 锁定。

  可以在 COMMIT 后从表中物理除去标记为删除的索引键。当删除的键标记为删除时，表的 X 锁定允许在物理上除去删除的键，如下所示。

- 使用 REORGCHK 来帮助确定何时重组索引或/和表以及何时使用带 CLEANUP ONLY 选项的 REORG INDEXES。

  要在重组期间允许对索引的读和写访问，运行带有 ALLOW WRITE ACCESS 选项的 REORG INDEXES。

注: 在 DB2 V8.1 和更高版本中，所有新索引都创建为 2 类索引。一个例外是当您在已具有 1 类索引的表上添加索引时。仅在这种情况下，新索引也将是 1 类索引。要了解一个表存在什么类型的索引，执行 INSPECT 命令。要将 1 类索引转换为 2 类索引，执行 REORG INDEXES 命令。

2 类索引的主要优点如下所示:

- 可以在其长度大于 255 个字节的列上创建索引。
- 使下一键锁定的使用减小到最小，这会改善并行性。因为将键标记为删除而不是在物理上从索引页除去，所以消除了大多数下一键锁定。有关键锁定的信息，参阅讨论锁定性能含义的主题。

在下列情况下，清除标记为删除的索引键:

- 在后续插入、更新或删除活动期间

  在键插入期间，清除标记为删除和已知要落实的键（如果这样的清除可以避免执行页分割的需要并防止增大索引的大小）。

  在键删除期间，当某个页上的所有键标记为删除时，尝试查找其中所有键标记为删除并且所有删除已落实的另一个索引页。如果找到这样的页，就会将它从索引树中删除。

  如果删除键时存在对表的 X 锁定，那么会在物理上删除该键，而不只是标记为删除。在此物理删除期间，如果相同页上的任何已删除的键标记为删除且已知要落实，那么也会除去这些键。
- 当用 CLEANUP 选项执行 REORG INDEXES 命令时

  CLEANUP ONLY PAGES 选项搜索并释放索引页，该页上的所有键都标记为删除并已知要落实。

  CLEANUP ONLY ALL 选项不仅释放其上所有键都标记为删除并已知要落实的索引页，它还除去一些页上的标记为删除并已知要落实的 RID，这些页包含某些未删除的 RID。

  此选项也尝试合并相邻叶子页（如果这样做会导致合并的叶子页上具有至少 PCTFREE 可用空间）。PCTFREE 值是当创建索引时为索引定义的可用空间的百分比。缺省 PCTFREE 是百分之十。如果可以合并两页，那么将释放其中一页。

  对于分区表，鼓励您在异步索引清除后执行 RUNSTATS，以便在已拆离的数据分区中生成准确的索引统计信息。要确定表中是否存在拆离数据分区，可选中 SYSDATAPARTITIONS 表中的状态字段，然后查找值“I”（索引清除）或“D”（与相关的 MQT 拆离）。
- 索引的任何重建

  重建索引的实用程序包括下列各项:

  – 当未使用其中一个 CLEANUP 选项时的 REORG INDEXES
  – 当未使用 INPLACE 选项时的 REORG TABLE
  – 带 REPLACE 选项的 IMPORT
  – 带 INDEXING MODE REBUILD 选项的 LOAD

了解分区表上的索引行为

分区表上的索引与普通表上的索引类似，即每个索引都包含指向表的所有数据分区中的行的指针。然而，一个主要的区别是分区表上的每个索引都是一个独立的对象。在分区数据库环境中，索引采用与表相同的方式分布在各数据库分区上。因为关于分区表的索引可以独立于其他索引操作，所以在分区表上创建索引时需要注意一些关于使用哪个表空间的特殊注意事项。

即使分区表的数据分区跨多个表空间，也只在单个表空间中创建该分区表的索引。DMS 和 SMS 表空间都支持在不同于表的另一个位置使用索引。所有指定的表空间必须位于同一数据库分区组中。可以将每个索引放置在它自己的表空间中，包括大型表空间。每个索引表空间都必须使用与数据分区相同的存储机制 DMS 或 SMS。大型表空间中的索引最多可以包含 2^29 页。

分区表上的索引的其他好处包括:

- 提高了删除索引和联机索引创建的性能
- 能够针对每个表索引之间的任何表空间特征使用不同的值（例如，为了确保更好的空间利用率，对每个索引使用不同的页大小可能更合适）。
- 减少 IO 争用并提供对表索引数据更有效的并发访问。
- 删除各个索引时，空间将立即可供系统使用，而无需进行索引重组。
- 如果您选择执行索引重组，可以重组单个索引。

图 16 显示了位于单个表空间中的分区表上的非分区索引。

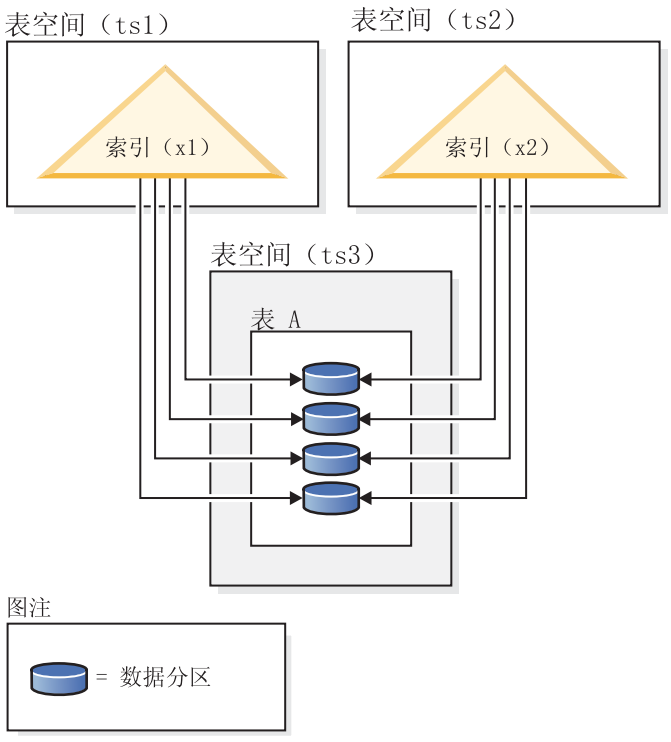


图 16. 分区表上的索引行为

图 17 显示了还分布在多个数据库分区上的分区表的索引行为。

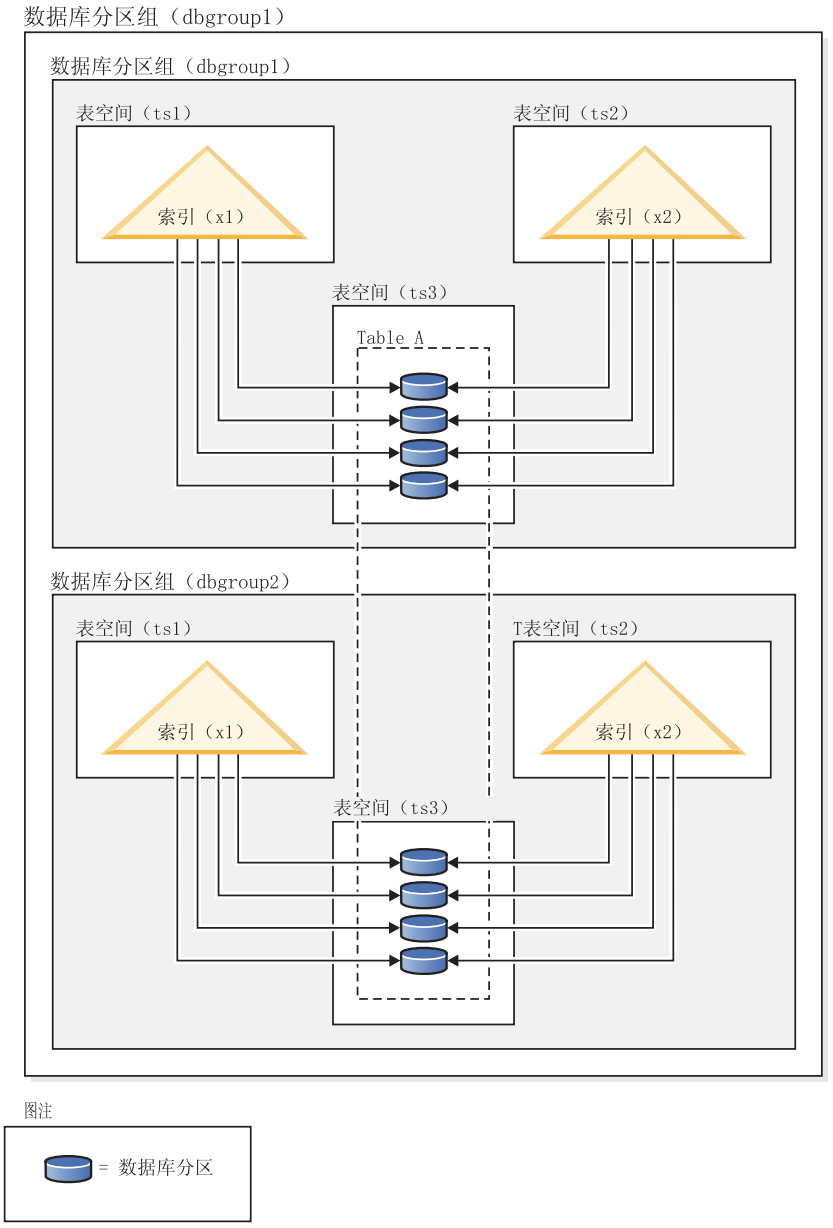


图 17. 分布式分区表的索引行为

可以使用 CREATE INDEX ...IN <tbspace1> 语句为分区表指定索引表空间，这与使用 CREATE TABLE .. INDEX IN <tbspace2> 语句指定的索引表空间不同。

只能对分区表使用 CREATE INDEX 语句中的 IN 子句来覆盖索引位置，这将允许您为索引指定表空间位置。此方法允许您根据需要将一个分区表上的不同索引放在不同表空间中。如果创建分区表时未指定放置它的非分区索引的位置，并且使用未指定特定表空间的 CREATE INDEX 语句来创建索引，那么将在第一个已连接的或可视数据分区的表空间中创建索引。按顺序对下面三种可能情况进行求值（从情况 1 开始），以确定创建索引的位置。当与其中一种情况相匹配时，求值将停止:

情况 1:
当在 CREATE INDEX ... IN <tbspace1> 语句中指定了索引表空间时，将 <tbspace1> 中指定的表空间用于此索引。
情况 2:
当在 CREATE TABLE .. INDEX IN <tbspace2> 语句中指定了索引表空间中，将 <tbspace2> 中指定的表空间用于此索引。
情况 3:
未指定任何表空间时，使用第一个已连接的或可视数据分区使用的表空间。

创建索引的位置取决于在 CREATE TABLE 语句期间执行的操作。对于非分区表，如果未指定任何 INDEX IN 子句，那么数据库将在该子句中填写您的数据表空间。对于分区表，如果您将它留空，那么它将保留为空白，此时情况 3 适用。

示例 1: 此示例假定存在分区表 sales (a int, b int, c int)，并在表空间“ts1”中创建唯一索引“a_idx”。

```
CREATE UNIQUE INDEX a_idx ON sales ( a ) IN ts1
```

示例 2: 此示例假定存在分区表 sales (a int, b int, c int)，并在表空间“ts2”中创建索引“b_idx”。

```
CREATE INDEX b_idx ONsales ( b ) IN ts2
```

异步索引清除

异步索引清除（AIC）是在使索引条目失效的操作之后的延迟索引清除。根据索引的类型，条目可以是行标识（RID）或块标识（BID）。无论是哪种标识，这些条目都将由索引清除程序除去，索引清除程序在后台异步运行。

AIC 加快从分区表拆离数据分区的速度。如果分区表包含一个或多个非分区索引，那么将启动 AIC。在这种情况下，AIC 将除去引用了已拆离的数据分区和任何已伪删除的条目的所有非分区索引条目。在清除所有索引之后，将从系统目录中除去与已拆离的数据分区相关联的标识。

注: 如果分区表定义了从属具体化查询表（MQT），那么 AIC 要在执行了 SET INTEGRITY 操作之后才启动。

当 AIC 正在进行时，将维护正常表访问。访问索引的查询将忽略尚未清除的任何无效条目。

在大多数情况下，对与分区表关联的每个非分区索引启动一个清除程序。内部任务分发守护程序负责将 AIC 任务分发给适当的数据库分区并指定数据库代理程序。

分发守护程序和清理代理程序都是内部系统应用程序。它们出现在 LIST APPLICATION 输出中，应用程序名称分别为 **db2taskd** 和 **db2aic**。为了防止意外中断，不能强制执行系统应用程序。只要数据库是活动的，分发守护程序就一直处于联机状态。清除程序在完成清理之前保持活动状态。如果在进行清理时取消激活了数据库，那么当您重新激活数据库时将继续进行 AIC。

性能

AIC 对性能产生很小影响。

需要进行瞬时行锁定测试以确定是否落实了伪删除的条目。但是，由于从未获取锁定，因此并行性不会受影响。

每个清除程序都获取最小表空间锁定（IX）和表锁定（IS）。如果清除程序确定其他应用程序正在等待这些锁定，就会释放这些锁定。如果发生这种情况，那么清除程序就会暂挂处理 5 分钟。

清除程序与实用程序调速功能集成。缺省情况下，每个清除程序的实用程序影响优先级为 50。可以使用 SET UTIL_IMPACT_PRIORITY 命令或 db2UtilityControl API 来更改此优先级。

监视

可以使用 LIST UTILITIES 命令来监视 AIC。每个索引清除程序都作为一个单独的实用程序出现在监视器中。

以下示例使用命令行处理器（CLP）界面演示了当前数据库分区中 WSDB 数据库中的 AIC 活动:

```
$ db2 list utilities show detail

标识                                 = 2
类型                                 = ASYNCHRONOUS INDEX CLEANUP
数据库名称                           = WSDB
分区号                               = 0
描述                                 = 表: USER1.SALES, 索引: USER1.I2
开始时间                             = 12/15/2005 11:15:01.967939
状态                                 = 正在执行
调用类型                             = 自动
正在调速:
      优先级                           = 50
进度监控:
            总计工作                     = 5 页
            已完成的工作                 = 0 页
            开始时间                     = 12/15/2005 11:15:01.979033

标识                                 = 1
类型                                 = ASYNCHRONOUS INDEX CLEANUP
数据库名称                           = WSDB
分区号                               = 0
描述                                 = 表: USER1.SALES, 索引: USER1.I1
开始时间                             = 12/15/2005 11:15:01.978554
状态                                 = 正在执行
调用类型                             = 自动
正在调速:
      优先级                           = 50
进度监控:
            总计工作                     = 5 页
            已完成的工作                 = 0 页
            开始时间                     = 12/15/2005 11:15:01.980524
```

在此示例中，有两个清除程序正在对 USERS1.SALES 表进行操作。一个清除程序正在处理索引 I1；另一个清除程序是处理索引 I2。进度监视部分显示需要清除的估计总索引页数和当前的干净索引页数。

状态字段指示清除程序的当前状态。通常，状态为“正在执行”。如果清除程序正在等待被指定给可用的数据库代理程序，或者如果清除程序由于锁定争用而临时暂挂，那么清除程序可能处于“正在等待”状态。

注: 因为每个数据库分区将标识仅指定给该数据库分区上的任务，所以不同数据库分区上的不同任务可能具有相同的实用程序标识。

联机索引整理碎片

用户定义的索引叶子页上的最小已使用空间量的阈值将启用联机索引整理碎片。当从叶子页删除了索引键且超过阈值时，可检查相邻的索引叶子页以确定是否可合并两个叶子页。如果某一页上有足够的空间用于合并两个相邻的页，那么立即在背景中合并。

索引的联机整理碎片只能对在版本 6 和更高版本中创建的索引使用。如果现有的索引需要合并联机的能力，那么必须删除它们然后用 MINPCTUSED 子句重新创建。将 MINPCTUSED 值设置为小于 100。MINPCTUSED 的建议值小于 50，因为目标是将两个相邻的索引叶子页合并。MINPCTUSED 的零值（也是缺省值）禁用联机整理碎片。

当除去一页上的最后一个索引键时，便从索引中释放该页。当指定了 CREATE INDEX 语句中的 MINPCTUSED 子句时，便会发生这一规则的例外情况。MINPCTUSED 子句指定索引叶子页上空间的百分比。当删除索引键时，如果页上填充空间的百分比等于或低于指定值，数据库管理器尝试合并剩余键和相邻页上的键。如果相邻页上有足够的空间，那么执行合并，并删除一个索引叶子页。

联机索引整理碎片期间，不合并索引非叶子页。但是，删除空非叶子页以供相同表上的其他索引复用。要对 DMS 存储模型中的其他对象释放这些非叶子页，或者要释放 SMS 存储模型中的磁盘空间，那么对表或索引进行全面重组。表和索引的全面重组可以使索引尽可能地小。联机索引整理碎片期间，不合并索引非叶子页。但如果它们成为空的，删除并释放它们，以便复用。可能会减少索引中的层数和叶和非叶子页的数量。

对于 2 类索引，仅当对表具有 X 锁定时，才在删除键期间从页除去键。这种操作期间，联机索引整理碎片将有效。但是，如果在键删除期间对表没有 X 锁定，键标记为已删除，但实际上并未从索引页上除去。结果整理碎片未成功。

要整理键标记为已删除但实际上仍在索引页中的 2 类索引的碎片，用 CLEANUPONLYALL 选项执行 REORG INDEXES 命令。CLEANUP ONLY ALL 选项整理索引碎片，与 MINPCTUSED 的值无关。如果用 CLEANUP ONLY ALL 执行 REORG INDEXES，那么如果两个相邻叶子页的合并至少可以在合并页上留下 PCTFREE 可用空间，那么合并这两个相邻叶子页。在索引创建时间指定 PCTFREE，且缺省为百分之十。

了解分区表上的集群索引行为

对分区表使用集群索引可获得与对常规表使用集群索引相同的好处。但是，在选择关于表分区键定义的集群索引时必须小心。

可以使用任何集群键在分区表上创建集群索引。数据库服务器尝试使用集群索引来以本地方式集群每个数据分区中的数据。在已划分集群的插入期间，将在索引中进行查询以找到合适的行标识（RID）。将从此 RID 开始查找表中的空间以插入记录。要获得维护良好且性能优良的集群，索引列与表分区键列之间应该存在关联。确保这种关联的一种方法是使表分区键列成为索引列的前缀，如下面的示例中所示:

```
PARTITION BY RANGE (Month, Region)
CREATE INDEX ...(Month, Region, Department) CLUSTER
```

虽然数据库服务器不强制此关联，但还是希望索引中的所有键按分区标识聚集在一起，以获得较好的集群。例如，如果一个表按季度进行分区，而集群索引按日期定义，由于季度与日期之间存在关联，所以可以获得具有良好性能的最佳数据集群，其中任何数据分区的所有键都聚集在索引中。

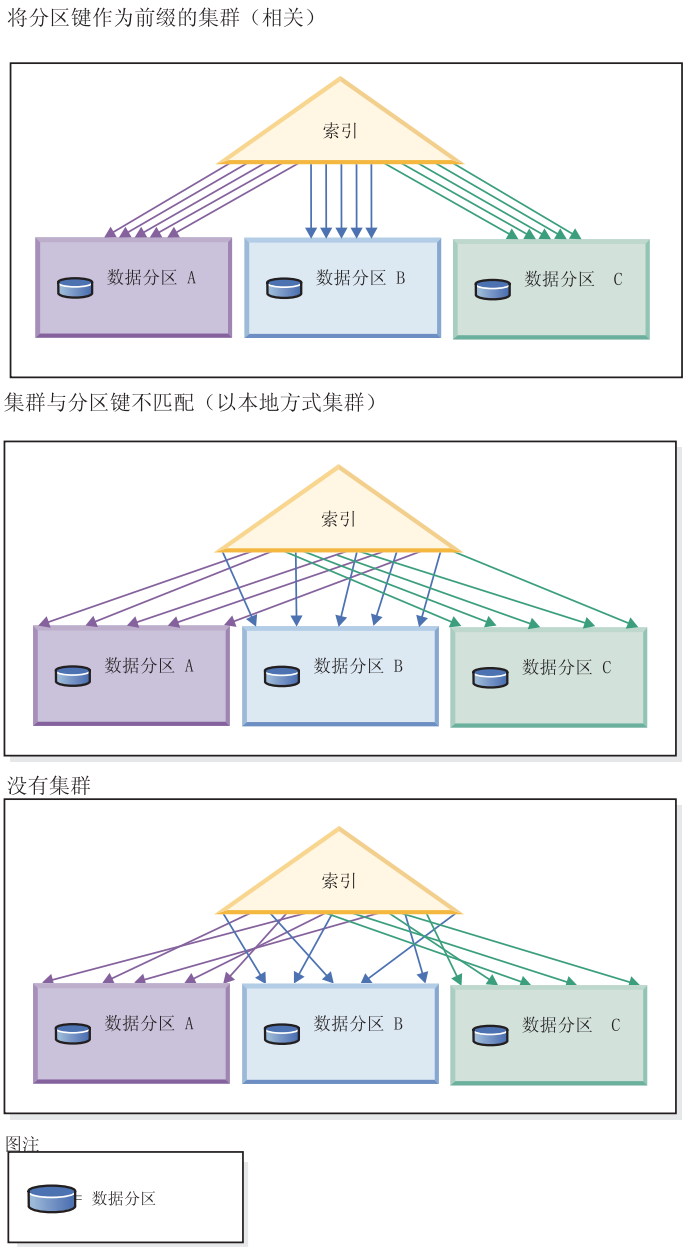


图 18. 集群索引可能对分区表产生的影响。在第一个图中，数据同时以全局和本地方式集群。

如第 103 页的图 18中所示，在给出每个示例中索引和数据的布局的情况下，当集群与表分区键相关时，可以获得最佳扫描性能。当集群与表分区键不相关时，不可能以本地方式集群索引。因为需要表分区列与索引列之间存在关联，所以根本不可能有一个完美的本地集群方案。

集群的好处包括:

- 在每个数据分区内，行以键的顺序排列。
- 集群索引改善了以键的顺序扫描整张表的性能。这是因为扫描访存第一页的第一行，然后访存同一页上的每一行，在访存了该页上的所有行之后，才移至下一页。这表示任何给定时间内都只需要表的一页位于缓冲池中。相反，如果表未集群，那么访存的每行有可能是不同页中的。除非缓冲池中有空间保存整个表，否则这会导致每页被访存多次，从而极大地减慢扫描速度。

  对于分区表，只有当表分区键是集群键的前缀（请参阅第 103 页的图 18 中的第一个图）时，才能保证扫描期间出现每页只访存一次的理想情况。但是，如果像上面描述的那样集群键与表分区键不相关，并且数据是本地集群的，那么当缓冲池中有足够的空间来保存每个数据分区的一页时，仍然可以获得集群索引的全部好处。这是因为对给定数据分区访存的每一行都位于上一次对同一数据分区访存的行的附近。（请参阅第 103 页的图 18 的第二个图）。如前面提及的那样，如果集群键与表分区键不相关，那么不能很好地维护集群，但如果您不想对表进行高级插入、更新和删除活动，使用此方法应该可以受益。

  即使缓冲池中没有足够的空间来保存每个数据分区的一页，定义集群索引仍然可以获得一些好处。

数据库代理程序

对于应用程序访问的每个数据库，各种进程或线程启动来执行各种应用程序任务。这些任务包括日志记录、通信和预取。

数据库代理程序是数据库管理器中用来处理应用程序请求的线程。在版本 9.5 中，代理程序在所有平台上都作为线程运行。

最大应用程序连接数由 *max_connections* 数据库管理器配置参数控制。每个应用程序连接的工作由单个工作程序代理程序协调。

*工作程序代理程序*执行应用程序请求，但并不与任何特定应用程序永久相连。协调代理程序与应用程序之间的关联时间最长，因为它们一直与应用程序相连，直到应用程序断开连接为止。此规则唯一例外的情况是在启用了引擎集中器时，此时协调代理程序可能在事务边界（事务落实和回滚）处终止该关联。

有四种类型的工作程序代理程序:

- 空闲代理程序
- 活动协调代理程序
- 子代理程序

空闲代理程序

这是最简单的工作程序代理程序形式。它不带出站连接，且不带本地数据库连接或实例连接。

活动协调代理程序

客户机应用程序的每个数据库连接都有一个在数据库上协调其工作的活动代理程序。在创建了协调代理程序之后，它代表它的应用程序执行所有数据库请求并使用进程间通信（IPC）或远程通信协议与其他代理程序进行通信。每个代理程序使用它自己的专用内存来运行，并与其他代理程序共享数据库管理器和数据库全局资源，如缓冲池。当事务完成时，活动协调代理程序可以成为不活动代理程序。

当客户机与数据库断开连接或者与实例拆离时，其协调代理程序将为:

- 活动代理程序。如果其他连接正在等待，那么工作程序代理程序成为活动协调代理程序。
- 被释放并标记为空闲（如果没有连接正在等待且正在自动管理池代理程序的最大数目或尚未到达该最大数目的话）。
- 被终止并释放其存储器（如果没有连接正在等待且已达到缓冲池代理程序的最大数目的话)。

子代理程序

协调代理程序会将数据库请求分发至子代理程序，然后由这些代理程序执行对该应用程序的请求。在创建了该协调代理程序之后，它将通过协调执行数据库请求的子代理程序，来代表它的应用程序处理所有数据库请求。在 DB2 版本 9.5 中，在非分区环境中和未启用查询内并行性的环境中也可能存在子代理程序。

不为任何应用程序执行工作且正在等待分配的代理程序被认为是空闲代理程序，并位于一个*代理程序缓冲池*中。这些代理程序可用于处理为客户机程序运行的协调代理程序发出的请求，或可用于现有协调代理程序运行的子代理程序。可用的代理程序数目取决于数据库管理器配置参数 *num_poolagents*。

当需要一个代理程序时，如果没有空闲的代理程序存在，动态创建一个新的代理程序。因为创建一个新代理程序需要一定量的开销，因此，如果可为客户机激活空闲代理程序，可提高 CONNECT 和 ATTACH 性能。

为应用程序执行工作的子代理程序与该应用程序*相关联*。在它完成指定的工作后，可以将它放在代理程序缓冲池中，但它仍然与原始应用程序相关。当该应用程序请求其他的工作时，数据库管理器创建新的代理程序之前，首先检查相关代理程序的空闲缓冲池。

数据库代理程序管理

大多数应用程序在连接的应用程序数与数据库可以处理的应用程序请求数之间建立一对一关系。但是，您的工作环境可能要求在连接的应用程序数与可以处理的应用程序请求数之间存在多对一关系。

以下两个数据库管理器配置参数提供了各自控制这些因子的能力:

- *max_connections* 参数，它指定允许连接的应用程序数
- *max_coordagents* 参数，它指定可以同时处理的应用程序请求数

当 *max_connections* 的值大于 *max_coordagents* 的值时，启用连接集中器。

因为每个活动协调代理程序都需要全局资源开销，所以这些代理程序数越大，达到可用数据库全局资源上限的机会也就越大。要防止达到可用数据库全局资源的上限，可将 *max_connections* 的值设置为高于 *max_coordagents* 的值。

设置 *max_connections* 和 *max_coordagents* 的值时，还可以指定 AUTOMATIC。在以下两种特定情况下使用 AUTOMATIC 将很有利:

- 如果您确信系统可以处理可能需要的所有连接，但是又想（通过限制协调代理程序数来）限制所使用的全局资源数，那么只应对 *max_connections* 参数指定 AUTOMATIC。当 *max_connections* 设置为 AUTOMATIC 并且值大于 *max_coordagents* 时，这意味着启用了连接集中器，这时将允许使用任意数目的连接（只要有足够的系统资源即可），但是，最大协调代理程序数仍然会受到限制。这可以用来同时控制内存约束和磁盘约束，其方法是: 限制同时执行的应用程序数。
- 如果您不想对系统施加人为限制从而限制最大连接数和最大协调代理程序数，但是您知道系统需要“多对一”关系（已连接的应用程序与已处理的应用程序请求之间就是这种关系）或者使用“多对一”关系会有好处，那么就应启用连接集中器并将两个参数都设置为 AUTOMATIC。当两个参数都设置为 AUTOMATIC 时，数据库管理器使用您指定的值作为表示空闲协调代理程序数与连接数之比的比值。

示例

请考虑以下设置:

- *max_connections* 参数设置为 AUTOMATIC 并且值为 300
- *max_coordagents* 参数设置为 AUTOMATIC 并且值为 100

max_connections 与 *max_coordagents* 的比值是 300:100。当连接增加时，数据库管理器将支持新的协调代理程序，因此，仅当需要时才应用集中。上述设置将变为:

- 第 1 到 100 个连接将创建新的协调代理程序
- 第 101 到 300 个连接将不会创建新的协调代理程序，它们将共享已经创建的 100 个代理程序
- 第 301 到 400 个连接将创建新的协调代理程序
- 第 401 到 600 个连接将不会创建新的协调代理程序，它们将共享已经创建的 200 个代理程序
- 依此类推...

注: 此示例假定已连接的应用程序正在执行足够多的工作来保证在每一步骤中创建新的协调代理程序。经过一段时间之后，如果已连接的应用程序不再执行任何工作，那么协调代理程序将变得不活动并且可能终止。

如果连接数减少了，但是由剩余连接执行的工作量还很大，那么协调代理程序数可能不会立即减少。*max_connections* 和 *max_coordagents* 参数不会直接影响代理程序合用或代理程序终止。正常的代理程序终止规则仍然适用，这意味着连接数与协调代理程序数之间的比值并不刚好是您指定的比值。代理程序在终止之前可能会返回到代理程序池以供复用。

如果需要进行详细程度更高的控制，建议指定简化的比值。例如，可以将上述 300:100 比值简化为 3:1 比值。如果 *max_connections* 设置为 3（AUTOMATIC）且 *max_coordagents* 设置为 1（AUTOMATIC），那么允许对每 3 个连接创建 1 个协调代理程序。

客户机连接的连接集中器改善

对于具有许多相对瞬时的连接的因特网应用程序或相似类型的应用程序，连接集中器通过允许有效地处理更多客户机连接来改善性能。它也减少每个连接的内存使用，并减少上下文转接的次数。

注: 当 *max_connections* 的值大于 *max_coordagents* 的值时，启用连接集中器。

在要求同时发生许多用户连接的环境内，可以启用连接集中器以更有效地使用系统资源。此功能部件结合了以前仅在 DB2 Connect™ 连接池中才具有的优点。《DB2 Connect 用户指南》中描述了连接池和连接集中器。首次连接后，连接集中器会减少与主机的连接时间。当请求与主机断开连接时，会断开入站连接，而在存储池中保留与主机的出站连接。当发出一个新请求以连接至主机时，数据库管理器将尝试复用该存储池中现有的出站连接。

注: 当应用程序使用连接池或连接集中器时，调整控制高速缓存数据块大小的参数，以获取最佳性能。有关更多信息，请参阅《DB2 Connect 用户指南》。

使用连接池时，DB2 Connect 只能进行入站 TCP/IP 和出站 TCP/IP 连接。

用法示例

示例 1:

考虑具有单个数据库分区的 ESE 环境，在此环境中，平均有 1000 个用户连接到该数据库。有时，连接的用户数可能会更多，因此不应设置 1000 个用户这一限制。并发事务的数目高达 200，但绝不会高于 250。事务较短。

对于此工作负载，可以设置下列数据库管理器配置参数:

- *max_coordagents* 设置为 250 以支持最大的并发事务数
- *max_connections* 设置为 AUTOMATIC 并且值为 1000，以确保支持任意数目的连接（在此示例中，任何大于 250 的值都将足以确保打开连接集中器）
- *num_poolagents* 设置为缺省值，但是该缺省值应确保数据库代理程序可用于服务入局客户机请求，只是增加创建新的代理程序的很少开销

示例 2:

在与示例 1 中的系统相似的系统中（该系统似乎对连接数没有限制），可以让协调代理程序数也根据连接数而增加。对于本示例，假定平均连接了 1000 个用户，大约有 250 个并发事务在运行，但是，有时候无法预测用户数峰值。有时候可能连接了 2000 个用户，可能预期平均有 500 个用户在执行工作。大多数时候不应允许 500 个协调代理程序，因为通常只连接了 1000 个用户，一般情况下 250 个协调代理程序就足够了。

对于此工作负载，可以按如下所示更新数据库管理器配置:

```
db2 update dbm cfg using MAX_COORDAGENTS 250 AUTOMATIC
db2 update dbm cfg using MAX_CONNECTIONS 1000 AUTOMATIC
```

这意味着随着连接数增加到超过 1000，将根据需要来创建更多协调代理程序，其最大值由连接总数来确定。在对两个参数都指定一个值并且都设置为 AUTOMATIC 的情况下，随着工作负载增加，数据库管理器会使协调代理程序数与连接数之间保持适当的比值。

示例 3:

在您不想对其启用连接集中器、但是想限制已连接的用户数（例如，一次只允许连接 250 个用户）的系统中，将数据库管理器配置参数设置为如下所示:

- 将 *max_connections* 设置为 250。
- 将 *max_coordagents* 设置为 250。

示例 4:

在您不想对其启用连接集中器并且不想限制已连接的用户数的系统中，将数据库管理器配置参数设置为如下所示:

```
db2 update dbm cfg using MAX_COORDAGENTS AUTOMATIC
db2 update dbm cfg using MAX_CONNECTIONS AUTOMATIC
```

分区数据库中的代理程序

对于分区数据库环境和启用分区内并行性的环境，每个数据库分区（即每个数据库服务器或节点）有它自己的代理程序存储池，可从中抽出子代理程序。因为有此缓冲池，子代理程序就不必在每次需要时创建或完成其工作时破坏掉。这些子代理程序仍然可以作为此缓冲池中的相关代理程序，并由数据库管理器使用，以执行与它们相关的应用程序或者新应用程序发出的新请求。

对于分区数据库环境和启用分区内并行性的环境，对系统中性能和内存成本的影响与代理程序存储池是如何调整的有很大关系:

- 代理程序缓冲池大小的数据库管理器配置参数（*num_poolagents*）影响可与一个数据库分区（也称为节点）上的应用程序保持相关的代理程序和子代理程序的总数。如果缓冲池大小太小且该缓冲池已满，那么子代理程序解除它自己与它正在其上工作的应用程序的关联并终止。因为必须常常创建子代理程序，并重新使它们与应用程序相关，所以性能会受损。

  缺省情况下，*num_poolagents* 设置为 AUTOMATIC 并且值为 100。当此参数设置为 AUTOMATIC 时，数据库管理器将自动管理池中的空闲代理程序数。

  另外，如果 *num_poolagents* 的值太小，那么一个应用程序就可能填满有相关子代理程序的缓冲池。因此，当另一个应用程序需要新的子代理程序，并且在其关联的代理程序池中没有子代理程序时，它将从其他应用程序的代理程序池中重新启动不活动的子代理程序。这种行为可确保充分利用资源。
- 必须根据在任何给定时间允许过多代理程序活动的资源成本权衡这些情况。

  例如，如果 *num_poolagents* 的值太大，使用不能用于其他任务的数据库管理器资源，那么相关的子代理程序也许很长时间都留在缓冲池中未使用。

  注: 当启用连接集中器时，由 *num_poolagents* 指定的代理程序数不一定会影响任何时候在池中处于空闲状态的代理程序的准确数目。可能会临时超过代理程序数以处理更多的工作负载活动。

其他异步进程和线程

除数据库代理程序外，其他异步数据库管理器活动还作为它们自己的进程或线程运行，包括:

- 数据库 I/O 服务器或 I/O 预取程序
- 数据库异步页清除程序
- 数据库记录器

- 数据库死锁检测器
- 通信和 IPC 侦听器
- 表空间容器重平衡程序。

数据库系统监视器信息

DB2 数据库管理器维护有关它的操作、性能和使用它的应用程序的数据。当数据库管理器运行时将维护此数据，此数据可以提供重要的性能和故障诊断信息。例如，您可以了解:

- 与一个数据库连接的应用程序数、它们的状态和每个应用程序正执行哪些 SQL 和 XQuery 语句（如果存在的话）。
- 显示数据库管理器和数据库是如何配置的以及帮助您调整它们的信息。
- 当指定数据库发生死锁时，涉及到哪些应用程序以及哪些锁定处于争用中。
- 由应用程序或数据库挂起的锁定的列表。如果应用程序由于等待锁定而不能继续执行，那么存在有关该锁定的其他信息，包括哪个应用程序挂起了该锁定。

因为要收集一些这种数据会给 DB2 的操作带来开销，所以可用**监视开关**来控制要收集哪些信息。要显式地设置监视开关，可使用 UPDATE MONITOR SWITCHES 命令或 sqlmon() API。（您必须具有 SYSADM、SYSCTRL、SYSMAINT 或 SYSMON 权限。）

可以通过获取快照或通过使用事件监视器来访问数据库管理器维护的数据。

获取快照

可以采用下面三种方式的其中一种来获取快照:

- 从命令行使用 GET SNAPSHOT 命令。
- 使用 db2GetSnapshot() API 调用编写自己的应用程序。
- 使用快照表函数来返回与数据库系统的特定区域相关的监视器数据。

使用事件监视器

当发生像事务结束、语句结束或检测到死锁这样的特定事件后，一个事件监视器将捕获系统监视器信息。可以将此信息写入文件或命名管道中。

要使用事件监视器:

1. 用控制中心或 SQL 语句 CREATE EVENT MONITOR 创建其定义。此语句在数据库系统目录中存储该定义。
2. 通过控制中心或用以下的 SQL 语句激活事件监视器:

   ```
   SET EVENT MONITOR evname STATE 1
   ```

   如果写入一个命名管道，那么在激活事件监视器之前，启动从这个命名管道中读取的应用程序。可编写自己的应用程序来执行此操作，或使用 db2evmon。一旦事件监视器被激活，并且开始向管道中写入事件，那么 db2evmon 将在生成这些事件时读取它们，并将它们写到标准输出中。
3. 读取该跟踪。如果使用文件事件监视器，那么您可以查看它用下列任何一种方式创建的二进制跟踪:
   - 使用 db2evmon 工具来将跟踪格式化为标准输出。

- 单击基于 Windows 的操作系统上控制中心中的**事件分析器**图标，以使用图形界面来查看跟踪、搜索关键字并过滤掉不想要的数据。

  注: 如果监视的数据库系统不在控制中心所在的机器上运行，在可以查看跟踪之前必须将事件监视器文件复制到控制中心所在的机器上。另一种方法是将该文件置于两台机器都可访问的共享文件系统中。

使用死锁事件监视器时的性能影响

在启用了 HISTORY 选项的情况下死锁事件监视器处于活动状态时，会以下列方式影响 DB2 数据库系统的一般性能:

- 程序包高速缓存中用于语句历史记录中所列示的高速缓存动态 SQL和 XQuery 语句的内存不会被释放，直到不再需要该特定语句历史记录（即，当前工作单元结束）为止。这可能会导致程序包高速缓存大小增大，原因是不能释放高速缓存中增加的空间使用量。
- 由于将语句信息复制到语句历史记录列表而引起的对系统性能的轻微影响。
- 每个数据库分区上的 DB2 系统监视器堆的使用量会增大，原因是为了保存该数据库分区中每个活动应用程序的语句历史记录列表。增大的量取决于每个应用程序在每个工作单元中所执行的语句数目。监视器堆的建议计算方法如下所示:

  如果事件监视器类型为 DEADLOCK 并且 WITH DETAILS HISTORY 选项正在运行，那么增加量为: X*100 个字节乘以预期运行的最大并发应用程序数目，其中 X 是应用程序的工作单元中预期的最大语句数目。如果事件监视器类型为 DEADLOCK 并且 WITH DETAILS HISTORY VALUES 选项正在运行，那么增加量为: X*Y 个字节乘以预期运行的最大并发应用程序数目，其中 Y 是被绑定到 SQL 和 XQuery 语句中的参数值的预期最大大小之和。

在启用了 VALUES 选项的情况下死锁事件监视器处于活动状态时，会以下列方式（除了前面对 HISTORY 选项列示的那些方式之外）影响 DB2 数据库系统的一般性能:

- 由于将语句信息复制到语句历史记录列表而引起的对系统性能的极轻微影响。
- 每个数据库分区上的 DB2 系统监视器堆的使用量会增大，原因是为了保存该数据库分区中每个活动应用程序的语句历史记录列表。增加量取决于每个语句所使用的数据值数目以及每个应用程序在每个工作单元中所执行的语句数目。
- 数据库管理器额外保留一份数据值可能会对性能产生影响，影响的大小取决于变量的大小和数目。

在对死锁事件监视器同时指定了 HISTORY 和 VALUES 选项时，对 DB2 系统监视器堆的内存影响可能会变得严重。要减少影响，应仅在需要时才使用这些选项。减少影响的另一个方法是在启用事件监视器之前增大所有数据库分区上的 DB2 系统监视器堆的配置大小。

当实际发生死锁且存在活动的死锁事件监视器时，系统性能就会受生成事件监视器记录的影响。影响程度及其持续时间取决于死锁中涉及到的应用程序和数据库分区的数目以及相关语句历史记录列表中的语句和数据值的数目。

有效的 SELECT 语句

因为 SQL 是一种灵活的高级语言，所以您可以编写几种不同的 SELECT 语句来检索同一数据。但是，对于不同的语句形式和不同的优化级别，性能可能相差很大。

考虑 SELECT 语句的下列准则:

- 仅指定需要的列。尽管用星号（*）指定所有列可能更简单，但会导致不必要的处理和返回不需要的列。
- 使用将答案集仅限制为您需要的那些行的谓词
- 当您需要的行数大大小于可能返回的总行数时，指定 OPTIMIZE FOR 子句。此子句影响访问方案的选择以及在通信缓冲区中分块的行数。
- 当要检索的行数很小时，仅指定 OPTIMIZE FOR *k* ROWS 子句。不需要 FETCH FIRST n ROWS ONLY 子句。但是，如果 *n* 值很大，并且您想快速获取前 *k* 行而对后续 *k* 行进行可能的延迟，那么同时指定两个子句。通信缓冲区的大小基于 *n* 和 *k* 中的较小者。下列示例显示两个子句:

  ```
  SELECT EMPNAME, SALARY FROM EMPLOYEE
          ORDER BY SALARY DESC      FETCH FIRST 100 ROWS ONLY
          OPTIMIZE FOR 20 ROWS
  ```

- 要利用行分块，指定 FOR READ ONLY 或 FOR FETCH ONLY 子句来改善性能。另外，因为在检索的行上从不挂起互斥锁定，所以并行性也会提高。也可发生其他查询重写。指定 FOR READ ONLY 或 FOR FETCH ONLY 子句以及 BLOCKING ALL BIND 选项可以类似的方式改善联合系统中查询昵称的性能。
- 对于将使用定位更新来更新的游标，指定 FOR UPDATE OF 子句以允许数据库管理器初始选择更适当的锁定级别并避免发生潜在的死锁。注意，FOR UPDATE 游标不能利用行分块。
- 对于将使用搜索更新来更新的游标，可以通过使用 FOR READ ONLY 和 USE AND KEEP UPDATE LOCKS 子句来对受影响的行强制执行 U 死锁来避免发生死锁并且仍然能够进行行分块。
- 尽可能避免数字数据类型转换。当比较值时，使用有相同数据类型的项可能更有效。如果需要转换，那么可能由于精度受限制使结果不准确，并且由于运行时转换而使性能降低。

  如果可能的话，使用下列数据类型:

  – 对于较短的列，尽量使用字符而不是可变字符
  – 尽量使用整数，而不是浮点数或小数
  – 尽量使用日期时间，而不是字符
  – 尽量使用数字，而不是字符
- 要减小排序操作发生的可能性，省略诸如 DISTINCT 或 ORDER BY 的子句或操作（如果这种操作不是必需的话）。
- 要检查一个表中是否存在某些行，选择某一单行。打开游标并访存一行，或执行单行（SELECT INTO）选择。记住，如果发现多行，那么要检查是否有 SQLCODE -811 错误。

  除非您知道表很小，否则，不要使用下列语句来检查非零值:

  ```
  SELECT COUNT(*) FROM  TABLENAME
  ```

  对于大表，对所有行计数会影响性能。
- 如果更新活动较少且表较大，那么在频繁用作谓词的列上定义索引。
- 如果同一列出现在多个谓词子句中，那么考虑使用 IN 列表。对于配合主变量使用的大型 IN 列表，循环主变量的子集可能会提高性能。

下列建议只适用于访问几个表的 SELECT 语句。

- 使用连接谓词来连接表。连接谓词是来自一个连接中不同表的两个列之间的比较。
- 在连接谓词中的列上定义索引，可以更有效地处理连接。索引也会改善 UPDATE 和 DELETE 语句的性能，这些语句包含访问几个表的 SELECT 语句。
- 如果可能的话，避免将表达式或 OR 子句与连接谓词一起使用，因为数据库管理器不能使用某些连接技术。因此，可能未选择最有效的连接方法。
- 在一个分区数据库环境中，如果可能的话，确保在连接列上对已连接的两个表进行了分区。

第 15 章 控制器实用程序

控制器可以监视对数据库运行的应用程序的行为并可以更改某些行为，这取决于在控制器配置文件中指定的规则。

控制器实例由一个前端实用程序和一个或多个守护程序组成。您启动的控制器的每个实例是特定于数据库管理器的一个实例。缺省情况下，当启动控制器时，控制器守护程序在分区数据库的每个数据库分区上启动。但是，可以指定在要监视的单个数据库分区上启动守护程序。

注：当控制器活动时，其快照请求可能会影响数据库管理器性能。要提高性能，增大控制器唤醒时间间隔以降低其 CPU 使用情况。

每个控制器守护程序收集关于对数据库运行的应用程序的信息。然后它对照您在控制器配置文件中对此数据库指定的规则检查此信息。

控制器按在配置文件中规则所指定的那样管理应用程序事务。例如，应用某个规则可能指示应用程序正使用过多的特定资源。规则将指定要执行的操作，如更改应用程序的优先级或强制应用程序与数据库断开连接。

如果与某个规则相关的操作更改应用程序的优先级，那么控制器在发生资源违例的数据库分区上更改代理程序的优先级。在分区数据库中，如果强制将应用程序与数据库断开连接，那么该操作发生，即使检测到违例的守护程序正在应用程序的协调程序节点上运行也是如此。

控制器记录它所执行的任何操作。要复查操作，查询日志文件。

启动和停止控制器

控制器实用程序监视连接至数据库的应用程序，并根据在控制器配置文件中对该数据库指定的规则更改应用程序的行为。

启动控制器之前，必须创建配置文件。

要启动或停止控制器，必须具有 *sysadm* 或 *sysctrl* 权限。

要启动或停止控制器:

1. 要启动控制器，在 DB2 命令行执行 *db2gov* 命令。 输入下列必需的参数:
   - START *database_name*

     指定的数据库名称必须与指定的配置文件中的数据库名称匹配。如果名称不相同，那么返回一条错误。注意，如果控制器是对多个数据库运行的，那么将为每个数据库启动一个守护程序。
   - *config_file_name*

     此数据库上控制器的配置文件的名称。如果该文件不在缺省位置（该位置为 `sqllib` 目录），那么必须包括路径和文件名。

- *log_file_name*

  此控制器的日志文件的基本名称。在分区数据库上，对守护程序为此控制器实例运行的每个数据库分区添加一个数据库分区号。

要在分区数据库的单个数据库分区上启动控制器，添加 *nodenum* 选项。

例如，要使用配置文件（称为 *salescfg*）和日志文件（称为 *saleslog*）在分区数据库的节点 3 上为数据库（称为 *sales*）启动控制器，请输入下列命令:

```
db2gov START sales nodenum 3 salescfg saleslog
```

要在 *sales* 数据库的所有数据库分区上启动控制器，请输入下列命令:

```
db2gov START sales  salescfg saleslog
```

2. 要停止控制器，请输入带 STOP 选项的 *db2gov* 命令。

   例如，要在 *sales* 数据库的所有数据库分区上停止控制器，请输入下列命令:

   ```
   db2gov STOP sales
   ```

   要仅在数据库分区 3 上停止控制器，请输入下列命令:

   ```
   db2gov START sales nodenum 3
   ```

控制器守护程序

当控制器守护程序启动时（当您通过 db2gov 实用程序执行它或当它唤醒时），它运行下列任务循环。

1. 它检查其控制器配置文件是否已更改或是否尚未读取。如果满足任何一个条件，那么该守护程序将读取该文件中的规则。这允许您在控制器守护程序运行时更改其行为。
2. 它请求关于正在数据库工作的每个应用程序和代理程序的资源使用统计信息的快照信息。

   注: 在某些平台上，不能从“DB2 监视器”获取 CPU 统计信息。在这种情况下，帐户规则和 CPU 限制不可用。
3. 它根据控制器配置文件中的规则检查每个应用程序的统计信息。如果规则适用于应用程序，那么控制器执行指定的操作。

   注: 控制器将累积的信息与配置文件中定义的值作比较。这表示如果使用应用程序可能已违反的新值来更新配置文件，那么与该违例相关的控制器规则在下一个控制器时间间隔将立即应用于该应用程序。
4. 它对执行的任何操作在控制器日志文件中写入一条记录。

注: 如果 *agentpri* 数据库管理器配置参数不是系统缺省值，那么不能使用控制器来调整代理程序优先级。

当控制器完成其任务后，它会在配置文件中指定的时间间隔内休眠。当经过此时间间隔之后，控制器唤醒并再次开始任务循环。

当控制器遇到错误或停止信号时，它会在结束之前先执行清除处理。通过使用已设置其优先级的应用程序的列表，清除处理复位所有应用程序代理程序的优先级。然后它

复位不再在应用程序上运行的任何代理程序的优先级。这确保在控制器结束后，代理程序不会一直以非缺省优先级运行。如果发生错误，那么控制器将一条消息写入管理通知日志，以指示它异常结束。

注：尽管控制器守护程序不是数据库应用程序，并因此不维护与数据库的连接，但它确实具有实例连接。因为它可以发出快照请求，所以控制器守护程序可以检测数据库管理器何时结束。

配置控制器

要配置“控制器”，创建配置文件，以确定“控制器”的实例监视的数据库以及它如何管理查询。

配置文件由一组规则组成。前三条规则指定要监视的数据库、写日志记录的时间间隔以及唤醒以进行监视的时间间隔。其余规则指定如何监视数据库服务器以及在特定情况下要采取的操作。

要创建“控制器”配置文件:

1. 在所有数据库分区中的已安装或可用的目录中，用描述性名称创建 ASCII 码文件。例如，监视 **sales** 数据库的控制器实例的配置文件可以称为 *govcfgsales*。
2. 在任何文本编辑器中打开该文件并输入配置信息和操作条件。

   用分号（;）结束每条规则。建议下列配置信息:

   - dbname: 要监视的数据库的名称或别名。
   - account: 分钟数，在该分钟数之后控制器实例将 CPU 使用情况统计信息写入其日志文件。
   - interval: 秒数，经过该秒数之后将唤醒控制器守护程序来监视活动。如果不指定时间间隔，那么使用缺省值 120 秒。

   例如，配置文件中的头三条规则可能看起来如下:

   ```
   { Wake up once a second, the database name is sales,
    do accounting every 30 minutes.  }
   interval 1; dbname sales; account 30;
   ```

   添加一些规则，这些规则指定要监视的条件和当规则评估为真时要采取的操作。例如，可以添加这样一条规则，该规则将工作单元（UOW）可运行的时间量限制为 1 小时，之后将被强制与数据库断开连接，如下所示:

   ```
   setlimit uowtime 3600 action force;
   ```

3. 保存该文件。

控制器配置文件

当启动控制器时，您要指定包含规则的配置文件，这些规则控制对数据库运行的应用程序。控制器评估每条规则并当规则评估为真时执行指定的操作。

如果规则需求更改，那么编辑配置文件而不必停止控制器。每个控制器守护程序检测到文件已更改，并重新读取文件。

必须在一个跨所有数据库分区安装的目录中创建配置文件，以便每个数据库分区上的控制器守护程序可以读取同一配置文件。

配置文件由三个必需的规则组成，这些规则标识要监视的数据库、写日志记录的时间间隔以及控制器守护程序的休眠时间间隔。遵循这些参数，配置文件包含一组可选的应用程序监视规则和操作。下列注释适用于所有规则:

- 将注释定界在 { } 花括号内。
- 可用大写、小写或大小写混合字符来指定大多数项。应用程序名称是一个例外，指定为 applname 规则的一个自变量，它区分大小写。
- 每条规则以一个分号（;）结束。

必需的规则

下列规则指定要监视的数据库，以及守护程序在每个活动循环后唤醒的时间间隔。这些规则中的每一条在文件中仅指定一次。

dbname
要监视的数据库的名称或别名。

account *nnn*
编写帐户记录，该记录包含每个连接在指定的分钟数内的 CPU 使用统计信息。

注: 此选项在 Windows® 环境中不可用。

如果一个短期连接会话全部发生在该帐户时间间隔内，那么不会编写任何日志记录。当编写日志记录时，它们包含 CPU 统计信息，这些信息反映自该连接的上一个日志记录以来 CPU 的使用情况。如果将控制器停止，然后重新启动，那么在两个日志记录中都可能反映 CPU 的使用情况；这些信息可通过日志记录中的应用程序标识来标识。

interval
守护程序唤醒的时间间隔，以秒计。如果不指定时间间隔，那么使用缺省值 120 秒。

控制操作的规则

遵循所需要的规则，可以添加指定如何控制应用程序的规则。这些规则由称为规则子句的更小的组件组成。如果使用的话，必须以特定的顺序在规则语句中输入子句，如下所示:

1. **desc**（可选）: 关于规则的注释，用引号括起
2. **time**（可选）: 在评估规则时的当天时间
3. **authid**（可选）: 应用程序在其下执行语句的一个或多个授权标识
4. **applname**（可选）: 连接到数据库的可执行文件或对象文件的名称。此名称区分大小写。如果应用程序名中包含空格，那么必须用双引号将应用程序名引起来。
5. **setlimit**: 控制器检查的限制。这些限制可以是几项限制（例如，CPU 时间、返回的行数或者空闲时间）的其中一项。
6. **action**（可选）: 当达到限制时要执行的操作。如果未指定操作，那么当达到限制时，控制器会将为应用程序工作的代理程序的优先级降低 10。对应用程序的操作可以包括降低其代理程序的优先级，强制它与数据库断开连接或为其操作设置调度选项。

通过在每条规则中仅使用每个子句一次，可以组合多个规则子句来形成一条规则，并用分号结束规则，如下列示例中所示:

```
desc "Allow no UOW to run for more than an hour"
setlimit uowtime 3600 action force;

desc "Slow down the use of db2 CLP by the novice user"
authid novice
applname db2bp.exe
setlimit cpu 5 locks 100 rowssel 250;
```

如果有多条规则适用于某个应用程序，那么应用所有这些规则。通常，与首先遇到的规则限制相关的操作是第一个要应用的操作。如果规则中的某个子句为 -1，那么发生您指定的异常。在这种情况下，为后续规则中的子句指定的值只能覆盖先前为*相同*子句指定的值：先前规则中的其他子句仍有效。例如，一条规则使用 rowssel 100000 uowtime 3600 子句来指定如果应用程序的耗用时间大于 1 小时，或者如果它选择多于 100000 行，那么应降低该应用程序的优先级。后续规则使用 uowtime -1 子句来指定同一应用程序可具有不受限制的耗用时间。在这种情况下，如果应用程序运行 1 小时以上，不会更改其优先级。也就是说，uowtime -1 覆盖 uowtime 3600。但是，如果它选择多于 100000 行，那么由于 rowssel 100000 仍有效而会降低其优先级。

规则应用程序的顺序

控制器在配置文件中从文件顶部到文件底部来处理规则。但是，如果后面规则的 setlimit 子句比前面的规则松散，那么仍会应用有更多限制的规则。例如，在下面的配置文件中，admin 将被限制为 5000 行而不考虑后面的规则，原因是第一条规则的限制性更强。

```
desc "Force anyone selecting 5000 or more rows"
setlimit rowssel 5000 action force;

desc "Allow user admin to select more rows"
authid admin
setlimit rowssel 10000 action force;
```

为了确保限制较少的规则能够覆盖文件中早些时间发生的限制较多的规则，可以指定 -1 选项以在应用新规则之前清除先前的规则。例如，在下面的配置文件中，初始规则将所有用户限制为使用 5000 行。第二条规则清除对 admin 的这一限制，第三条规则将对 admin 的限制重设为 10000 行。

```
desc "Force anyone selecting 5000 or more rows"
setlimit rowssel 5000 action force;

desc "Clear the rowssel limit for admin"
authid admin
setlimit rowssel -1;

desc "Now set the higher rowssel limit for admin"
authid admin
setlimit rowssel 10000 action force;
```

控制器规则元素

控制器配置文件中的每条规则都由一些子句组成，这些子句指定了规则所适用的条件以及导致该规则求值为真的操作。这些子句必须按显示的顺序指定。在子句描述中，[ ] 指示可选子句。

可选的开始元素

[desc] 指定该规则的文本描述。必须用单引号或双引号将该描述引起来。

[time] 指定对该规则求值的时间段。

必须以下列格式指定该时间段: `time hh:mm hh:mm`, 例如, `time 8:00 18:00`。如果不指定此子句, 该规则一天 24 小时都有效。

[authid]

指定该应用程序在执行时所使用的一个或多个授权标识 (authid)。必须用逗号 (,) 将多个 authid 分隔, 例如, `authid gene, michael, james`。如果在规则中不出现此子句, 那么该规则适用于所有 authid。

[applname]

指定建立与数据库的连接的可执行程序（或对象文件）的名称。

必须用逗号（,）将多个应用程序名隔开, 例如, `applname db2bp, batch, geneprog`。如果在规则中不出现此子句, 那么该规则适用于所有应用程序名。

注:

1. 应用程序名区分大小写。
2. 数据库管理器将所有应用程序名截断为 20 个字符。应确保想控制的应用程序由其应用程序名的前 20 个字符唯一地标识; 否则, 控制的可能不是要控制的应用程序。

   将在控制器配置文件中指定的应用程序名截断为 20 个字符, 以匹配其内部表示。

限制子句

setlimit

指定控制器要检查的一个或多个限制。这些限制只能是 -1 或大于 0 （例如, `cpu -1 locks 1000 rowssel 10000`）。必须至少指定这些限制（cpu、locks、rowsread 和 uowtime）中的一个, 且该规则未指定的任何限制都不受该特定规则的限制。控制器可检查下列限制:

cpu *nnn*

指定一个应用程序可消耗的 CPU 秒数。如果您指定 -1, 那么控制器不限制应用程序的 CPU 使用。

locks *nnn*

指定一个应用程序可挂起的锁定数。如果您指定 -1, 那么控制器不限制应用程序挂起的锁定数。

rowssel *nnn*

指定返回至应用程序的行数。此值将仅在协调程序节点上为非零值。如果您指定 -1, 那么控制器不限制可选择的行数。可为 *nnn* 指定的最大值是 4294967298。

uowtime *nnn*

指定工作单元（UOW）从第一次进入活动状态开始可耗用的秒数。如果指定 -1, 那么不限制耗用时间。

注: 如果使用 sqlmon（数据库系统监视器开关）API 来取消激活工作单元监视开关或时间戳记监视开关, 这将影响控制器根据工作单元耗用时间来控制应用程序的能力。控制器使用监视器来收集有关该系统的信息。如果您在数据库管理器配置文件中关闭了这些开关, 那么对于整个实例, 它也会关闭, 且控制器将不再接收此信息。

idle *nnn*

指定在执行指定的操作之前对连接允许的空闲秒数。如果指定 -1，那么不限制连接的空闲时间。

注: 某些数据库实用程序（例如，BACKUP 和 RESTORE）与数据库建立连接，然后通过 EDU 执行控制器不可视的工作。这些数据库连接看起来是空闲的，并且可能会超过空闲时间限制。为了防止控制器对这些实用程序执行操作，可以通过调用这些实用程序的授权标识对它们指定 -1。例如，为了防止控制器对由授权标识 DB2SYS 运行的实用程序执行操作，可指定“authid DB2SYS setlimit idle -1”。

rowsread *nnn*

指定一个应用程序可选择的行数。如果指定 -1，那么不限制该应用程序可选择的行数。可为 *nnn* 指定的最大值是 4294967298。

注: 此限制与 rowssel 不同。不同点是: rowsread 是为了返回结果集而必须读取的行数的计数。读取的行数包括引擎对目录表的读取，当使用索引时可能减少这种读取。

操作子句

[action]

指定超过一个或多个指定的限制时要采取的操作。您可以指定下面的操作。

注: 如果超过限制且未指定操作子句，那么控制器将把为该应用程序工作的代理程序的优先级降低 10。

nice *nnn*

指定为应用程序工作的代理程序的相对优先级更改。有效值在 -20 至 +20 之间。

要使此参数有效:

- 在基于 UNIX 的平台上，必须将 *agentpri* 数据库管理器参数设置为缺省值；否则，它将覆盖 priority 子句。
- 在 Windows 平台上，可以将 *agentpri* 数据库管理器参数与*优先级*操作一起使用。

可以使用控制器来控制在缺省用户服务超类 SYSDEFAULTUSERCLASS 中运行的应用程序的优先级。如果使用控制器来降低在此服务超类中运行的应用程序的优先级，那么代理程序将解除与其出站相关因子的关联（如果它与一个出站相关因子关联的话），并根据控制器所指定的代理程序优先级来设置其相对优先级。不能使用控制器来改变用户定义的服务超类和子类中的代理程序优先级。相反，必须使用该服务超类或子类的代理程序优先级设置来控制在这些服务类中运行的应用程序。但是，可以使用控制器在任何服务类中强制连接。

force 指定强制为该应用程序使用的代理程序。（发出 FORCE APPLICATION 以终止该协调代理程序。）

注: 在多分区数据库环境中，仅当正在应用程序的协调数据库分区上运行控制器守护程序时，才会执行强制操作。因此，如果正在分区 A 上运行控制器守护程序，而协调数据库分区为数据库分区 B 的某些应用程序超出限制，那么会跳过强制操作。

schedule [class]

调度可改善执行应用程序的代理程序的优先级，其目标是将平均响应时间减至最低，同时维护所有应用程序之间的公平性。

控制器根据下列三个条件选择用于调度的前几个应用程序:

- 挂起大多数锁定的应用程序

  此选项尝试减小锁定等待数。
- 最旧的应用程序
- 具有最短估计剩余运行时间的应用程序。

  此选项尝试允许在时间间隔内完成尽可能多的短时间活动的语句。

给予每个条件中排名前三位的应用程序比所有其他应用程序更高的优先级，也就是说，给予每个条件组中排名第一的应用程序最高的优先级，给予排名第二的应用程序第二高的优先级，以及给予排名第三的应用程序第三高的优先级。如果有个应用程序在多个条件中都排名前三位，那么给予它排名最高的那个条件的适当优先级，而给予排名第二的应用程序其他条件的次高优先级。例如，如果应用程序 A 挂起最多的锁定，但具有第三少的估计剩余运行时间，那么给予它第一个条件的最高优先级，而给予具有最短估计剩余运行时间的排第四位的应用程序对该条件的第三高的优先级。

此控制器规则选择的应用程序最多分为三类。对于每一类，控制器根据以上列示的条件选择 9 个应用程序，它们是每个类中排名前三位的应用程序。如果指定类选项，此规则所选择的所有应用程序被看作单个类，并且选择 9 个应用程序并给予它们较高的优先级，如上所述。

如果在多个控制器规则中选择了某个应用程序，那么由选择了该应用程序的最后一个规则控制该应用程序。

注: 如果使用了 sqlmon（数据库系统监视器开关）API 来取消激活该语句开关，这将影响控制器根据该语句耗用时间来控制应用程序的能力。控制器使用监视器来收集有关该系统的信息。如果您在数据库管理器配置文件中关闭了这些开关，那么对于整个实例，它也会关闭，且控制器将不再接收此信息。
调度操作可以:

- 确保不同组中的每个应用程序获取时间，而不用在所有应用程序之间均匀地划分时间。

  例如，如果 14 个应用程序（三个短的、五个中等长度和六个长的）同时运行，那么它们的响应时间可能都很长，因为它们同时使用 CPU。数据库管理员可以设置两个组：中等长度的应用程序和长应用程序。使用优先级，控制器允许所有短的应用程序运行，并确保最多三个中等长度的应用程序和三个长应用程序同时运行。为此，控制器配置文件包含一条针对中等长度应用程序的规则，和一条针对长应用程序的规则。

  以下示例显示一个控制器配置文件的一部分，以举例说明这一点:

```
desc "Group together medium applications in 1 schedule class"
applname medq1, medq2, medq3, medq4, medq5
setlimit cpu -1
```

```
action schedule class;

desc "Group together long applications in 1 schedule class"
applname longq1, longq2, longq3, longq4, longq5, longq6
setlimit cpu -1
action schedule class;
```

- 确保多个用户组（例如，机构性部门）中的每一个都获取相等的优先级。

  如果一个组正在运行大量的应用程序，那么管理员可以确保其他组仍能够获得合理的响应时间，以运行他们的应用程序。例如，对于涉及到三个部门（财务、库存和计划）的情况，所有“财务”用户可能会被置于一个组，所有“库存”用户可能会被置于第二个组，而所有“计划”用户可能会被置于第三个组。应将处理能力在这三个部门中较为均匀地划分。以下示例显示一个控制器配置文件的一部分，以举例说明这一点:

  ```
  desc "Group together Finance department users"
  authid tom, dick, harry, mo, larry, curly
  setlimit cpu -1
  action schedule class;

  desc "Group together Inventory department users"
  authid pat, chris, jack, jill
  setlimit cpu -1
  action schedule class;

  desc "Group together Planning department users"
  authid tara, dianne, henrietta, maureen, linda, candy
  setlimit cpu -1
  action schedule class;
  ```

- 允许控制器调度所有的应用程序。

  如果操作中未包括类选项，那么控制器会根据调度操作下活动应用程序的个数来创建它自己的类，并根据 DB2 查询编译器为该应用程序运行的查询所做的成本估计，来将这些应用程序置于不同的类中。管理员可以不限定地选择应用程序，以调度所有的应用程序。即，不提供 *applname* 或 *authid* 子句，*setlimit* 子句就不会产生任何限制。

注: 如果超过限制且未指定操作子句，那么控制器降低执行该应用程序的代理程序的优先级。

控制器配置文件示例

以下示例显示一个控制器配置文件，它设置几条带操作的规则:

```
 Wake up once a second, the database name is ibmsampl,
 do accounting every 30 minutes.  }
interval 1; dbname ibmsampl; account 30;

desc "CPU restrictions apply 24 hours a day to everyone"
setlimit cpu 600 rowssel 1000000 rowsread 5000000;

desc "Allow no UOW to run for more than an hour"
setlimit uowtime 3600 action force;

desc 'Slow down a subset of applications'
applname jointA, jointB, jointC, quryA
```

```
setlimit cpu 3 locks 1000 rowssel 500 rowsread 5000;

desc "Have governor prioritize these 6 long apps in 1 class"
applname longq1, longq2, longq3, longq4, longq5, longq6
setlimit cpu -1
action schedule class;

desc "Schedule all applications run by the planning dept"
authid planid1, planid2, planid3, planid4, planid5
setlimit cpu -1
action schedule;

desc "Schedule all CPU hogs in one class which will control consumption"
setlimit cpu 3600
action schedule class;

desc "Slow down the use of db2 CLP by the novice user"
authid novice
applname db2bp.exe
setlimit cpu 5 locks 100 rowssel 250;

desc "During day hours do not let anyone run for more than 10 seconds"
time 8:30 17:00 setlimit cpu 10 action force;

desc "Allow users doing performance tuning to run some of
          their applications during lunch hour"
time 12:00 13:00 authid ming, geoffrey, john, bill
applname tpcc1, tpcc2, tpcA, tpvG setlimit cpu 600 rowssel 120000 action force;

desc "Some people should not be limited -- database administrator
    and a few others.  As this is the last specification in the
  file, it will override what came before."
authid gene, hershel, janet setlimit cpu -1 locks -1 rowssel -1 uowtime -1;

desc "Increase the priority of an important application so it always
            completes quickly"
applname V1app setlimit cpu 1 locks 1 rowssel 1 action priority -20;
```

控制器日志文件

每当控制器守护程序执行操作时，它将一条记录写入它的日志文件。操作包括以下各项:

- 强制应用程序
- 读取控制器配置文件
- 更改应用程序优先级
- 遇到错误或警告
- 启动或结束

每个控制器守护程序都有一个单独的日志文件中。独立的日志文件防止文件锁定瓶颈，此瓶颈可能会在许多控制器守护程序同时写入同一个文件时产生。要将日志文件合并在一起并查询它们，可使用 db2govlg 实用程序。

日志文件存储在 sqllib 目录的 log 子目录中（在 Windows 操作系统上除外），其中，log 子目录位于实例目录下面。当通过 db2gov 命令启动控制器时，应提供该日志文件的基本名称。确保日志文件名包含数据库名称，以区分受控制的每个数据库分区上的日志文件。为了确保文件名对于分区数据库环境中的每个控制器是唯一的，自动将运行控制器守护程序的数据库分区号追加至日志文件名。

日志文件记录格式

日志文件中的每个记录具有下列格式:

```
Date Time NodeNum RecType Message
```

注: *Date* 和 *Time* 字段的格式为 yyyy-mm-dd hh.mm.ss。可以通过对此字段进行排序来合并每个数据库分区的日志文件。

NodeNum 字段指示控制器运行所在的数据库分区的编号。

RecType 字段包含不同的值，这取决于写入该日志的日志记录的类型。可记录的值有:

- START: 启动了控制器
- STOP: 停止了控制器
- FORCE: 强制了应用程序
- NICE: 更改了应用程序的优先级
- ERROR: 发生错误
- WARNING: 发生警告
- READCFG: 控制器读取了配置文件
- ACCOUNT: 应用程序记帐统计信息。
- SCHEDGRP: 发生了代理程序优先级更改。

下面更详细地描述了这些值中的某一些。

START

START 记录是在控制器启动时编写的。它具有以下格式:

```
Database = <database_name>
```

STOP STOP 记录是在控制器停止时编写的。它具有以下格式:

```
Database = <database_name>
```

FORCE

每当控制器确定根据控制器配置文件中的规则需要强制使用应用程序时，就会编写 FORCE 记录。FORCE 记录具有以下格式:

```
<appl_name> <auth_id> <appl_id> <coord_partition> <cfg_line>
<restriction_exceeded>
```

其中:

<coord_partition>

指定应用程序的协调数据库分区的编号。

<cfg_line>

指定控制器配置文件中导致强制使用应用程序的规则所在的行号。

<restriction_exceeded>

提供有关超出规则的情况的详细信息。这可包含下列值:

- CPU: 总应用程序 USR cpu 加 SYS cpu 时间（以秒计）
- Locks: 应用程序持有的总锁定数
- Rowssel: 应用程序选择的总行数
- Rowsread: 应用程序读取的总行数
- Idle: 应用程序空闲的时间

- ET（耗用时间）：自从启动应用程序的当前工作单元以来的耗用时间（超过了 uowtime setlimit）

NICE NICE 记录是在通过控制器配置文件中的指定的优先级操作更改应用程序的优先级时编写的。NICE 记录具有以下格式:

```
<appl_name> <auth_id> <appl_id> <nice value> (<cfg_line>)
<restriction_exceeded>
```

其中:

<nice value>
指定将对应用程序的代理程序的优先级值所作的增大（或减小）。

<cfg_line>
指定控制器配置文件中导致更改应用程序优先级的规则所在的行号。

<restriction_exceeded>
提供有关超出规则的情况的详细信息。这可包含下列值:

- CPU: 总应用程序 USR cpu 加 SYS cpu 时间（以秒计）
- Locks: 应用程序持有的总锁定数
- Rowssel: 应用程序选择的总行数
- Rowsread: 应用程序读取的总行数
- Idle: 应用程序空闲的时间
- ET（耗用时间）：自从启动应用程序的当前工作单元以来的耗用时间（超过了 uowtime setlimit）

ERROR
ERROR 记录是在控制器守护程序需要关闭时编写的。

WARNING
WARNING 记录在下列情况下会写入控制器日志:

- 调用 sqlefrce API 以强制使用应用程序，但该 API 返回正的 SQLCODE。
- 快照调用返回并非 1611（“SQL1661 不返回任何数据”）的正的 SQLCODE。
- 快照调用返回并非 -1224（“SQL 1224N 不能启动数据库代理程序来处理请求，或者数据库代理程序因数据库系统关闭或强制命令而终止”）或 -1032（“SQL1032N 没有发出任何启动数据库管理器命令”）的负的 SQLCODE。这些返回码在关闭先前活动的示例后发生。
- 在 UNIX 环境中，尝试安装信号处理程序，但尝试失败。

ACCOUNT
ACCOUNT 记录在下列情况下写入控制器日志:

- 自上次为此应用程序编写 ACCOUNT 记录以来，此应用程序的 agent_usr_cpu 或 agent_sys_cpu 的值已更改。
- 应用程序不再活动

ACCOUNT 记录具有以下格式:

<auth_id> <appl_id> <applname> <connect time> <agent_usr_cpu delta> <agent_sys_cpu delta>

SCHEDGRP
SCHEDGRP 记录是在下列情况下编写的:

- 将应用程序添加至调度组
- 将应用程序从一个调度组移至另一个调度组。

SCHEDGRP 记录具有以下格式:

```
<appl_name> <auth_id> <appl_id> <cfg_line> <restriction_exceeded>
```

其中:

<cfg_line>
指定控制器配置文件中导致调度应用程序的规则所在的行号。

<restriction_exceeded>
提供有关超出规则的情况的详细信息。这可包含下列值:

- CPU: 总应用程序 USR cpu 加 SYS cpu 时间（以秒计）
- Locks: 应用程序持有的总锁定数
- Rowssel: 应用程序选择的总行数
- Rowsread: 应用程序读取的总行数
- Idle: 应用程序空闲的时间
- ET（耗用时间）: 自从启动应用程序的当前工作单元以来的耗用时间（超过了 uowtime setlimit）

因为写入的是标准值，您可查询不同类型的操作的日志文件。*Message* 字段提供其他非标准信息，这些信息根据 *RecType* 字段中的值的不同而有所变化。例如，FORCE 或 NICE 记录指示 *Message* 字段中的应用程序信息，而 ERROR 记录包括错误消息。

如下所示是一个日志文件示例:

```
1995-12-11 14.54.52    0 START      Database = TQTEST
1995-12-11 14.54.52    0 READCFG    Config = /u/db2instance/sqllib/tqtest.cfg
1995-12-11 14.54.53    0 ERROR      SQLMON Error: SQLCode = -1032
1995-12-11 14.54.54    0 ERROR      SQLMONSZ Error: SQLCode = -1032
```

控制器日志文件查询

每个控制器守护程序都写入自己的日志文件中。可使用 db2govlg 实用程序来查询该日志文件。可按日期和时间排序来列示单个数据库分区和所有数据库分区的日志文件。也可根据 *RecType* 日志字段进行查询。db2govlg 的语法如下所示:

图 19. db2govlg 的语法

这些参数如下所示:

log-file 您想查询的一个或多个日志文件的基本名称。

nodenum *node-num*
控制器运行所在的数据库分区的节点号。

rectype *record-type*
您想查询的记录的类型。这些记录类型有:

- START
- READCFG

- STOP
- FORCE
- NICE
- ERROR
- WARNING
- ACCOUNT

使用此实用程序不存在授权限制。这允许所有用户查询控制器是否影响了他们的应用程序。如果您想限制对此实用程序的访问，可更改 db2govlg 文件的组许可权。

第 16 章 基准程序测试

基准程序测试是应用程序开发生命周期的一个常规部分。它是由应用程序开发人员和数据库管理员（DBA）等小组成员参与的工作，应对您的应用程序执行它，以确定当前性能并提高性能。如果已将应用程序代码编写得尽可能效率高，那么要想再改善性能，可调整数据库和数据库管理器配置参数。甚至可以调整应用程序参数来更好地满足应用程序的需求。

运行不同类型的基准程序测试来发现特定种类的信息:

- *每秒事务数*基准程序确定在某些有限的实验室条件下数据库管理器的吞吐量能力。
- *应用程序*基准程序测试与生产条件接近的条件下的相同吞吐量能力。

调整配置参数的基准程序测试基于这些“现实世界”条件，并需要使用各种参数值反复运行从您的应用程序中获取的 SQL，直到应用程序运行得尽可能有效率为止。

此处所描述的基准程序测试方法基于调整配置参数。但是，该基本技术同样可以用于调整影响性能的其他因素，如:

- SQL 语句
- 索引
- 表空间配置
- 应用程序代码
- 硬件配置

基准程序测试有助于了解数据库管理器在各种条件下是如何响应的。可以创建多个方案来测试死锁处理、实用程序性能、装入数据的不同方法以及当添加更多的用户时事务执行速率的特征，甚至还可测试使用该产品的新发行版对应用程序产生的影响。

基准程序测试方法

基准程序测试基于可重复的环境，因此在相同条件下运行的相同测试将产生可以合理比较的结果。

可通过在一个正常的环境中运行测试应用程序来开始基准程序测试。随着您缩小性能问题的范围，可以开发专用的测试用例，以限制正测试的功能的作用域。这些专用测试用例不需要仿真整个应用程序来获取有价值的信息。从简单的评估开始，仅在必要时才增加复杂程度。

好的基准程序测试或评估应包括以下特征:

- 测试是可重复的。
- 测试的每次迭代在相同系统状态下开始。
- 除非方案包括系统中执行的一定量的其他活动，否则，系统中的其他功能或应用程序是不活动的。

  注: 已启动的应用程序即使是在最小化或空闲时也会占用内存。这样增大了页面调度将使基准程序的运行结果产生偏差和违反可重复性规则的概率。
- 用于基准程序测试的硬件和软件与您的生产环境匹配。

对于基准程序测试，创建一个方案，然后在此方案中创建几次应用程序，以在每次运行期间捕获关键信息。在每次运行之后捕获关键信息在确定可以提高应用程序和数据库的性能的更改方面具有主要的重要性。

基准程序准备

在开始性能基准程序测试之前，完成应用程序对其运行的数据库的逻辑设计。设置并填充表、视图和索引。规范化表，绑定应用程序程序包并用实际数据填写表。

您还应该已确定了数据库的最终物理设计。将数据库管理器对象放在其最终磁盘位置，接大小排列日志文件，以确定工作文件和备份的位置，并测试备份过程。此外，检查程序包，以确保在可能时启用性能选项，如行分块。

您应该已在应用程序编程和测试阶段达到这样一个时候，即，将允许您创建基准程序。尽管在基准程序测试期间，可能会暴露应用程序的实际限制，但是，此处描述的基准程序的目的是测量性能，而不是检测故障或异常结束。

基准程序测试程序需要在尽可能接近于最终生产环境的条件下运行。理论上，它应该在具有相同内存和磁盘配置的同一种型号的服务器上运行。当该应用程序最终将涉及大量用户和大量数据时，这一点尤其重要。操作系统本身和基准程序直接使用的任何通信或文件服务工具也应已调整好。

确保使用生产大小的数据库运行基准程序测试。单个 SQL 语句返回的数据量以及需要的排序量应与在生产环境中相同。此规则确保应用程序将测试典型的内存需求。

要进行基准程序测试的 SQL 语句应该是*典型的*或*恶劣的*，如下所述:

典型的 SQL

典型的 SQL 包括在对应用程序的典型操作进行基准程序测试期间执行的那些语句。选择的语句将取决于应用程序的性质。例如，数据输入应用程序可能要测试 INSERT 语句，而银行业务事务可能会测试 FETCH、UPDATE 和多个 INSERT。考虑选择平均值的语句的执行频率和处理的数据量。如果该数据量过大，那么即使这些语句是典型的 SQL 语句，也将它们归为*恶劣*类别考虑。

恶劣的 SQL

归入此类别的语句包括:

- 频繁执行的语句。
- 处理大量数据的语句。
- 与时间密切相关的语句。

  例如，当接收到客户的电话时运行的应用程序，而当客户等待时，必须运行语句以检索和更新客户信息。
- 要连接最大数量的表的语句，或含有应用程序中最复杂的 SQL 的语句。

  例如，银行业务应用程序，它针对所有不同类型的帐户生成每月活动的综合客户报表。一个公用表可能会列示客户地址和帐户；但是，还必须合并其他几个表，以处理和集成所有需要的帐户事务信息。处理一个帐户所需的工作量乘以同一个周期内必须处理的几千个帐户，在这种情况下，可能节省的时间迫使提高性能要求。

- 有不良访问路径的语句，如不经常执行的且为涉及的表创建的索引不支持的语句。
- 耗时长的语句。
- 只在应用程序初始化时才执行但有不适当的资源需求的语句。

  例如，生成当天必须处理的帐户工作列表的应用程序。当启动该应用程序时，第一个主 SQL 语句产生 7 路合并，这为此应用程序用户负责的所有帐户创建了一个非常大的列表。该语句可能每天只运行几次，但是如果未将它调整好就会耗费数分钟来运行。

基准程序测试创建

当设计和实现基准程序时，考虑各种因素。由于该程序的主要目的是模拟用户应用程序，所以程序的总体结构是变化的。可以将整个应用程序用作基准程序，只需引入某种方法来对要分析的 SQL 语句进行计时。对于大的或复杂的应用程序，只包括包含重要语句的块可能更实用。

要测试特定 SQL 语句的性能，可以将这些语句以及必需的 CONNECT、PREPARE、OPEN 和其他语句以及计时机制一起单独包括在该基准程序中。

另一个要考虑的因素是要使用的基准程序的类型。一个选择是在一个时间间隔内重复运行一组 SQL 语句。执行的语句数量与此时间间隔的比率就是该应用程序的吞吐量。另一个选择是只确定执行个别 SQL 语句所需的时间。

对于所有基准程序测试，需要一个高效率的计时系统来计算个别 SQL 语句或整个应用程序的运行耗用时间。要模拟个别 SQL 语句单独执行所在的应用程序，重点是跟踪 CONNECT、PREPARE 和 COMMIT 语句的时间。但是，对于处理多个不同语句的程序，或许只有单个 CONNECT 或 COMMIT 是需要的，而可能会优先测试个别语句的执行时间。

尽管每个查询的耗用时间是性能分析中的一个重要因素，但可能不必暴露瓶颈。例如，有关 CPU 的使用情况、锁定和缓冲池 I/O 的信息可能显示该应用程序达到 I/O 限制，而不是 CPU 的使用达到满负荷。基准程序应该允许您获取此类数据，以便在需要时进行更详细的分析。

并非所有的应用程序都将从查询检索到的整组行发送至某个输出设备。例如，整个答案集可能是另一个程序的输入，因此，不发送第一个应用程序的任何行作为输出。格式化屏幕输出的数据常常产生很高的 CPU 成本，且可能无法反映用户需要。要提供准确的模拟，基准程序应该反映特定应用程序的行处理。如果将行发送至输出设备，那么效率不高的格式化可能消耗大量的 CPU 处理时间，并会误报 SQL 语句本身的实际性能。

db2batch 基准程序工具：在您的实例 `sqllib` 目录的 `bin` 子目录中提供了一个基准程序工具（`db2batch`）。此工具使用许多准则，以用于创建基准程序。此工具可以从平面文件或标准输入读取 SQL 语句，动态地描述和准备这些语句，并返回答案集。它还允许控制答案集的大小以及从此答案集发送至输出设备的行数。

可以指定提供的与性能相关的信息的级别，包括耗用时间、CPU 和缓冲池的使用情况、锁定和从数据库监视器收集的其他统计信息。如果正在对一组 SQL 语句进行计时，那么 db2batch 也汇总性能结果，并提供算术和几何平均数。对于语法和选项，在命令行上输入 db2batch -h。

基准程序测试执行

对于一种类型的数据库基准程序，选择一个配置参数并使用该参数的不同值运行该测试，直至达到最佳效果。单个测试应该包括通过相同参数值的多次迭代（例如，20 或 30 次）来执行该应用程序以获取平均计时，这可以更清楚地显示参数更改所产生的影响。

当运行基准程序时，应该将第一次迭代（称为热身运行）视为不同于后续迭代（称为正常运行）的一种特殊情况。因为热身运行包括某些启动活动（例如，初始化缓冲池），因此，花费的时间比正常运行要稍微长一些。虽然来自热身运行的信息可能实际上是有效的，但从统计角度来说是无效的。当计算一组特定参数值的平均计时或 CPU 时，仅使用来自正常运行的结果。

可以考虑使用“配置顾问程序”来创建基准程序的热身运行。“配置顾问程序”问到的问题可以涉及到在进行基准程序活动期间为正常运行调整环境配置时要考虑的若干事宜。可以从“控制中心”或通过带适当选项执行 db2 自动配置命令来启动“配置顾问程序”。

如果基准程序测试使用单个查询，那么通过刷新缓冲池来确保最小化先前查询的潜在影响。要刷新缓冲池，读取与查询无关的大量的页来填充该缓冲池。

在完成单组参数值的迭代之后，可以更改单个参数。但是，在每个迭代之间，执行下列任务，以便将基准程序的环境复原至它的初始状态:

- . 如果由于测试的需要更新了目录统计信息，那么要确保每个迭代都使用相同的统计值。
- 如果测试更新了测试中使用的数据，那么此数据必须保持一致。为此:
  - 使用 RESTORE 实用程序来复原整个数据库。数据库的备份副本包含它的先前状态，即已准备好进行下次测试。
  - 使用 IMPORT 或 LOAD 实用程序来复原该数据的导出副本。此方法只允许复原受影响的数据。应对包含此数据的表和索引运行 REORG 和 RUNSTATS 实用程序。
- 要将应用程序返回至它的原始状态，将应用程序重新绑定至数据库。

概括起来，遵循以下步骤或迭代来对数据库应用程序执行基准测试:

第一步 除下列参数外，将数据库和数据库管理器的其他调整参数保持为它们的**缺省**值:

- 对于测试的工作负载和目标很重要的那些参数。(您很少有足够的时间执行基准程序测试以调整所有参数，所以可能需要使用某些参数的最佳推测值并从该点上开始调整。)
- 日志大小，它应在应用程序的单元测试和系统测试期间确定。
- 为了使应用程序能够运行而必须更改的参数（即，为防止出现语句堆内存用完这类事件而导致产生负的 SQL 返回码所需的更改）。

对此初始情况运行一组迭代，然后计算平均计时或 CPU。

第二步 选择一个且唯一一个调整参数来测试，并更改它的值。

第三步 运行另一组迭代，然后计算平均计时或 CPU。

第四步 根据基准程序测试的结果，执行下列其中一项操作:

- 如果性能提高，那么更改同一个参数的值并返回至第三步。继续更改此参数，直到产生最大效益为止。
- 如果性能降低或保持不变，那么将该参数返回至其原来的值，返回至第二步，并选择新的参数。重复此过程，直到所有的参数都已被测试为止。

  注: 如果您想将该性能结果绘制成图表，那么要查找曲线开始上升或下降的点。

可以编写一个驱动程序，以帮助您进行基准程序测试。可使用 REXX 之类的语言来编写此驱动程序，或者对于 Linux 和 UNIX 平台，使用 shell 脚本。

此驱动程序将执行基准程序，将适当的参数传送给它，通过多次迭代驱动该测试，将环境复原至一致的状态，使用新的参数值设置下一个测试，以及收集/合并测试结果。这些驱动程序可以很灵活，它们可用于运行一整套基准程序测试，分析结果，并为给定测试提供一个最终和最优参数值报告。

基准程序测试分析示例

基准程序的输出应该包括每个测试的标识、程序执行的迭代、语句号和执行的计时。

在经过一系列测量之后，基准程序测试结果的总结可能类似于如下所示:

```
     Test     Iter.   Stmt    Timing         SQL Statement
Numbr    Numbr   Numbr   (hh:mm:ss.ss)
      002      05      01     00:00:01.34    CONNECT TO SAMPLE
      002      05      10     00:02:08.15    OPEN cursor_01
      002      05      15     00:00:00.24    FETCH cursor_01
      002      05      15     00:00:00.23    FETCH cursor_01
      002      05      15     00:00:00.28    FETCH cursor_01
      002      05      15     00:00:00.21    FETCH cursor_01
      002      05      15     00:00:00.20    FETCH cursor_01
      002      05      15     00:00:00.22    FETCH cursor_01
      002      05      15     00:00:00.22    FETCH cursor_01
      002      05      20     00:00:00.84    CLOSE cursor_01
      002      05      99     00:00:00.03    CONNECT RESET
```

图 20. 基准程序的样本结果

注: 上面报告中的数据仅供说明之用。它**不**表示测量的结果。

分析显示: CONNECT（语句 01）用去 1.34 秒，OPEN CURSOR（语句 10）用去 2 分钟 8.15 秒，FETCHES（语句 15）返回七行且延迟时间最长，为 .28 秒，CLOSE CURSOR（语句 20）用去 .84 秒，而 CONNECT RESET（语句 99）用去 .03 秒。

如果您的程序可以定界的 ASCII 格式输出数据，那么可以在以后将该数据导入数据库表或电子表格进行进一步统计分析。

基准程序报告的样本输出可能是:

```
PARAMETER        VALUES FOR EACH BENCHMARK TEST
       TEST NUMBER      001     002     003    004     005
       locklist         63      63      63      63      63
       maxappls         8       8       8       8       8
       applheapsz       48      48      48      48      48
       dbheap           128     128     128     128     128
       sortheap         256     256     256     256     256
       maxlocks         22      22      22      22      22
       stmtheap         1024    1024    1024    1024    1024
       SQL STMT         AVERAGE TIMINGS (seconds)
           01             01.34   01.34   01.35   01.35   01.36
           10             02.15   02.00   01.55   01.24   01.00
           15             00.22   00.22   00.22   00.22   00.22
           20             00.84   00.84   00.84   00.84   00.84
           99             00.03   00.03   00.03   00.03   00.03
```

图 21. 基准程序的样本计时报告

注: 上面报告中的数据仅供说明之用。它**不**表示任何测量的结果。

第 17 章 设计顾问程序

DB2 设计顾问程序是一个工具，可帮助您显著提高工作负载性能。选择要为复杂工作负载创建哪些索引、MQT、集群维或数据库分区的任务可能会令人十分头痛。“设计顾问程序”标识了提高工作负载性能所需要的所有对象。如果工作负载中存在一组 SQL 语句，那么“设计顾问程序”将为下列各项提供一些建议:

- 新的索引
- 新的具体化查询表（MQT）
- 确定集群索引的添加
- 对多维集群（MDC）表的转换
- 表的再分发
- （通过 GUI 工具）删除指定的工作负载未使用的索引和 MQT

可以使用“设计顾问程序”立即实现这些建议中的一部分建议或所有建议，也可以安排稍后实现这些建议。

“设计顾问程序”可以使用“设计顾问程序”GUI 或命令行工具来帮助简化下列任务:

规划或建立新的数据库

当设计数据库时，使用“设计顾问程序”来执行下列任务:

- 在分区数据库环境或索引、MQT 和 MDC 表的测试环境中生成设计备用项。
- 对于分区数据库环境，可以使用“设计顾问程序”来完成下列任务:
  - 将数据装入数据库前确定数据库分区策略。
  - 帮助从单一分区 DB2 数据库迁移到多分区 DB2 数据库。
  - 帮助从另一个数据库产品迁移到多分区 DB2 数据库。
- 对已手动生成的索引、MQT、MDC 表或数据库分区策略进行评估。

工作负载性能调整

在建立数据库之后，可以使用“设计顾问程序”来:

- 提高特定语句或工作负载的性能。
- 提高总体数据库性能，使用样本工作负载的性能作为衡量标准。
- 提高最频繁执行的查询（例如，由“活动监控器”标识的查询）的性能。
- 确定如何优化新键查询的性能。
- 对“运行状况中心”提供的一些建议作出响应，这些建议是针对共享内存实用程序或具有大量排序的工作负载中的排序堆问题提出的。
- 查找工作负载中未使用的对象。

设计顾问程序输出

如果使用“设计顾问程序”GUI，那么可以从“设计顾问程序”中查看、保存或实现建议。如果从命令行运行“设计顾问程序”，那么缺省情况下将把输出打印至标准输出并保存在 ADVISE_TABLE 和 ADVISE_INDEX 表中:

- 对 MQT、MDC 表和数据库分区策略建议，ADVISE_TABLE 包含 USE_TABLE='Y'。

  MQT 建议也可在 ADVISE_MQT 表中找到；MDC 建议也可在 ADVISE_TABLE 表中找到；数据库分区策略建议也可在 ADVISE_PARTITION 表中找到。每次执行"设计顾问程序"时，这些表中的 RUN_ID 值都对应于 ADVISE_INSTANCE 表中的 START_TIME 值。
- 对于索引建议，ADVISE_INDEX 表中包含 USE_INDEX='Y' 或 'R'。

每次运行"设计顾问程序"时，都还要对 ADVISE_INSTANCE 表更新一行:

- START_TIME 字段和 END_TIME 字段分别显示实用程序的启动时间和停止时间。
- 如果实用程序成功结束，那么 STATUS 字段将包含"COMPLETED"。
- MODE 字段指示是否使用了 -m 选项。
- COMPRESSION 字段指示使用的压缩类型。

可以使用 -o 选项将"设计顾问程序"建议保存至某个文件。保存的"设计顾问程序"输出包含下列元素:

- 为新的索引、MQT、数据库分区策略和 MDC 表提供的 CREATE STATEMENTS。
- 为 MQT 提供的 REFRESH 语句。
- 为新对象提供的 RUNSTATS 命令。
- 如果使用了现有 MQT 和索引并且使用它们来执行工作负载，那么它们将出现在建议的脚本中。

  注: ADVISE_MQT 表的 COLSTATS 列包含 MQT 的列统计信息。统计信息采用 XML 结构，如下所示:

```
<?xml version=\"1.0\" encoding=\"USASCII\"?>
<colstats>
        <column>
         <name>COLNAME1</name>
         <colcard>1000</colcard>
         <high2key>999</high2key>
         <low2key>2</low2key>
    </column>
    ....

        <column>
         <name>COLNAME100</name>
         <colcard>55000</colcard>
         <high2key>49999</high2key>
         <low2key>100</low2key>
    </column>
</colstats>
```

  还添加根据其定义索引的基本表。可使用 benefit 值对索引和 MQT 进行排名。也可以使用"benefit - 0.5*开销"对 MQT 进行排名，并使用"benefit - overhead"对索引进行排名。索引和 MQT 的列表后面是工作负载中的语句列表，包括 SQL 文本、语句编号、根据建议估计的性能提高（改进），以及语句使用的表、索引和 MQT 的列表。

  注: 此输出保留了 SQL 文本中的原始间隔，但为了便于阅读，将 SQL 文本分为 80 个字符的带注释行。
  MDC 和分区未明确显示在此 XML 输出中。此输出的示例如下所示:

```
--<?xml version="1.0"?>
--<design-advisor>
--<mqt>
--<identifier>
--<name>MQT612152202220000</name>
--<schema>ZILIO2 </schema>
--</identifier>
--<statementlist>3</statementlist>
--<benefit>1013562.481682</benefit>
--<overhead>1468328.200000</overhead>
--<diskspace>0.004906</diskspace>
--</mqt>
.....
--<index>
--<identifier>
--<name>IDX612152221400000</name>
--<schema>ZILIO2 </schema>
--</identifier>
--<table><identifier>
--<name>PART</name>
--<schema>TPCD </schema>
--</identifier></table>
--<statementlist>22</statementlist>
--<benefit>820160.000000</benefit>
--<overhead>0.000000</overhead>
--<diskspace>9.063500</diskspace>
--</index>
.....
--<statement>
--<statementnum>11</statementnum>
--<statementtext>
--
-- select
-- c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice,
-- sum(l_quantity) from tpcd.customer, tpcd.orders,
-- tpcd.lineitem where o_orderkey in( select
-- l_orderkey from tpcd.lineitem group by l_orderkey
-- having sum(l_quantity) > 300 ) and c_custkey
-- = o_custkey and o_orderkey = l_orderkey group by
-- c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice
-- order by o_totalprice desc, o_orderdate fetch first
-- 100 rows only
--</statementtext>
--<objects>
--<identifier>
--<name>MQT612152202490000</name>
--<schema>ZILIO2 </schema>
--</identifier>
--<identifier>
--<name>ORDERS</name>
--<schema>TPCD </schema>
--</identifier>
--<identifier>
--<name>CUSTOMER</name>
--<schema>TPCD </schema>
--</identifier>
--<identifier>
--<name>IDX612152235020000</name>
--<schema>ZILIO2 </schema>
--</identifier>
--<identifier>
--<name>IDX612152235030000</name>
--<schema>ZILIO2 </schema>
--</identifier>
--<identifier>
--<name>IDX612152211360000</name>
--<schema>ZILIO2 </schema>
```

```
--</identifier>
--</objects>
--<benefit>2091459.000000</benefit>
--<frequency>1</frequency>
--</statement>
```

注意，XML 结构可以包含多个列。对于每一列，会显示列基数（即，该列中的值的个数）以及 high2 和 low2 键（可选）。

当提供了 MQT、数据库分区或 MDC 建议时，相关的 ALTER TABLE 存储过程调用命令放在 ADVISE_TABLE 的 ALTER_COMMAND 列中。

注: ALTER TABLE 存储过程调用命令可能会因对 ALTOBJ 存储过程的表的限制而失败。

在进行一些细微的修改之后，可以将此输出文件作为 CLP 脚本来运行以创建建议的对象。您可能想执行的修改包括:

- 将所有 RUNSTATS 命令语句组合为对新对象或已修改的对象的单个 RUNSTATS 调用。
- 提供比系统生成的标识更适用的对象名。
- 除去或注释掉不想立即实现的对象的任何 DDL。

使用设计顾问程序

可以从命令行运行设计顾问程序，也可以使用控制中心中的“设计顾问程序”GUI 来运行该程序。

- **控制中心方法:**
  1. 第 137 页的『定义“设计顾问程序”的工作负载』.
  2. 打开控制中心。
  3. 在控制中心中，通过执行以下操作来运行设计顾问程序: 用鼠标右键单击适当的数据库或数据库中的特定索引，然后从弹出菜单中选择“设计顾问程序”。
  4. 在设计顾问程序生成建议完成后，您可以选择将这些建议保存到一个报告中，也可以让设计顾问程序实现某些或全部建议。
- **命令行方法:**
  1. 第 137 页的『定义“设计顾问程序”的工作负载』.
  2. 对工作负载运行 db2advis 命令。

     注: 如果数据库统计信息不是最新的，那么生成的建议就没那么可靠。
  3. 解释 db2advis 的输出并进行必要的修改。
  4. 实施选择的设计顾问程序建议。

定义“设计顾问程序”的工作负载

*工作负载*是数据库管理器必须在给定时间段内处理的一组 SQL 语句。例如，在一个月的时间内，数据库管理器可能必须处理 1000 个 INSERT、10000 个 UPDATE、10000 个 SELECT 和 1000 个 DELETE。“设计顾问程序”将分析指定的工作负载并且考虑这样一些因素：例如，工作负载语句的类型、执行特定语句的频率和要生成使运行工作负载的总成本最小的一些建议的数据库的特征。

从“设计顾问程序”GUI 工作负载页中，可以创建新的工作负载文件，或者修改先前已有的工作负载文件。可以从以下几种源将一些语句导入文件中:

- 定界文本文件
- 事件监视器表
- Query Patroller 历史记录数据表（通过从命令行使用 -qp 选项）
- EXPLAINED_STATEMENT 表中的说明语句
- 使用 DB2 快照捕获到的最新 SQL 语句。
- 工作负载管理器活动表
- 工作负载管理器事件监视器表（通过从命令行使用 -wlm 选项）

在导入 SQL 语句之后，可以添加、更改、修改或除去语句并修改它们的频率。

从命令行，使用下列语句来运行“设计顾问程序”:

- 通过直接插入在命令中而输入的单个 SQL 语句
- 在 DB2 快照中捕获到的一组动态 SQL 语句
- 工作负载文件中包含的一组 SQL 语句。
- 要使用动态 SQL 语句运行“设计顾问程序”:
  1. 使用以下命令复位数据库监视器:

     ```
     db2 reset monitor for database database-name
     ```

  2. 等待适当的时间间隔以便对数据库执行动态 SQL 语句。
  3. 发出带有 -g 选项的 db2advis 命令。如果想要将动态 SQL 语句保存在 ADVISE_WORKLOAD 表中以便将来引用，那么还应该使用 -p 选项。
- 要使用工作负载文件中包含的一组 SQL 语句来运行“设计顾问程序”:
  1. 手动创建工作负载文件，并将每个 SQL 语句用分号隔开，或者从上面列示的一个或多个源中导入 SQL 语句。
  2. 设置工作负载中的语句的频率。缺省情况下，为工作负载文件中的每个语句指定的频率为 1。SQL 语句的频率表示该语句相对其他语句在工作负载中出现的次数。例如，一个特定的 SELECT 语句在工作负载中出现了 100 次，而另一个 SELECT 语句出现了 10 次。为了表示这两个语句的相对频率，可以指定第一个 SELECT 语句的频率为 10，第二个 SELECT 语句的频率为 1。可以通过在特定语句后面插入下面这一行来手动更改该语句在工作负载中具有的频率或权重 – # SET FREQUENCY *n*，其中 *n* 是想要为该语句指定的频率值。
  3. 通过使用 -i 选项并且后跟工作负载文件的名称来运行 db2advis 命令。
- 要对 ADVISE_WORKLOAD 表中包含的工作负载运行“设计顾问程序”，运行带有 -w 选项并且后跟工作负载名称的 db2advis。

使用设计顾问程序从单一分区迁移到多分区数据库

可以使用“设计顾问程序”来帮助您从单一分区迁移到多分区数据库。“设计顾问程序”可提供有关分布数据的建议，同时提供有关新索引、具体化查询表（MQT）和多维集群（MDC）表的建议。

1. 使用 db2licm 命令注册 DB2 产品或功能部件许可证密钥
2. 在多分区数据库分区组中至少创建一个表空间。

   注：因为“设计顾问程序”只能建议对现有表空间再分发数据，所以在运行“设计顾问程序”之前，分区数据库中必须存在您想要“设计顾问程序”考虑的表空间。
3. 在使用在“设计顾问程序”GUI 中选择的数据库分区功能部件或者使用为 db2advis 命令指定的分区选项的情况下运行“设计顾问程序”。
4. 如果正在“控制中心”中使用“设计顾问程序”，那么可自动实现数据库分区建议。如果正在使用 db2advis 命令，那么在运行由“设计顾问程序”生成的 DDL 语句之前需要稍微修改 db2advis 输出文件。

设计顾问程序的局限性和限制

1. **对于索引建议的限制**
   - 对具体化查询表（MQT）建议的索引会提高工作负载性能（这与 REFRESH TABLE 性能相反）。而且，如果工作负载中不包括更新、插入或删除，那么对于 IMMEDIATE MQT，将不包括更改（例如，更新）MQT 的性能。
   - 仅当要选择多维集群时，才建议创建集群 RID 索引。顾问程序将把 RID 集群索引作为一个选项来而不是为表创建 MDC 结构。
2. **对于 MQT 建议的限制**
   - “设计顾问程序”建议不要使用增量 MQT。如果想要创建增量 MQT，可使用 REFRESH IMMEDIATE MQT 并使用您选择的登台表来将它们转换为增量 MQT。
   - 对 MQT 建议的索引用来提高工作负载性能而不是 MQT 刷新性能。
   - 如果指定的工作负载中不包括更新、插入或删除，那么不考虑更新建议的 REFRESH IMMEDIATE MQT 对性能产生的影响。建议 REFRESH IMMEDIATE MQT 对隐含的唯一键创建唯一索引，隐含的唯一键是由 MQT 查询定义的 GROUP BY 子句中的列组成的。
3. **对于 MDC 建议的限制**
   - 必须使用有代表性的数据集来填充现有表，否则“设计顾问程序”将不会对表考虑 MDC。建议最少使用 24 到 36 MB 的数据。始终将排除小于 12 个扩展数据块的表。
   - 除非对命令使用了采样选项 -r 或者在 GUI 工具中选择了 MQT 采样，否则将不考虑 MDC 对新 MQT 的建议。
   - “设计顾问程序”不会为类型表或临时表生成 MDC 建议。
   - “设计顾问程序”不会为联合表生成 MDC 建议。
   - 对于在执行“设计顾问程序”期间所使用的采样数据必须具有存储空间（大约为大型表的表数据的 1%），否则在不相关假设下将只检查已采样的表以找到基本列。在此情况下，将生成警告消息。
   - 将跳过（即，不考虑）没有收集统计信息的表。

- “设计顾问程序”不会为多列维生成建议。
- 现有表中必须存在数据以便在进行 MDC 选择时对数据采样。

4. **对数据库分区建议的限制**

   “设计顾问程序”只能对 DB2 企业服务器版上的数据库分区进行建议。如果对 db2advis 命令指定了数据库分区选项，那么会返回错误。在“设计顾问程序”GUI 中，不能对单一分区数据库环境选择数据库分区功能。

5. **其他限制**

   在执行“设计顾问程序”期间，会创建模拟目录表。当执行完“设计顾问程序”时，会删除这些表。如果未执行完“设计顾问程序”，可能会导致不删除这些表中的某些表。在此情况下，可以通过从命令行重新启动实用程序来使用“设计顾问程序”删除这些表。要除去模拟目录表，应同时指定 -f 选项和 -n 选项（对于 -n，指定不完整执行时所使用的用户名）。如果不指定 -f 选项，那么“设计顾问程序”将生成除去这些表所需要的 DROP 语句。

   注: 从版本 9.5 开始，-f 选项是缺省值。这表示如果对选择的 MQT 运行 db2advis，那么数据库管理器将使用与模式名相同的用户标识自动删除所有本地模拟目录表。

   应该另外创建一个表空间来存储这些模拟目录表，并将 DROP TABLE RECOVERY 设置为“OFF”。这样就会更容易清除，并且执行“设计顾问程序”的速度也会更快。只能在目录节点上为模拟目录表定义单独的表空间。

第 5 部分 调整数据库应用程序性能

第 18 章 应用程序注意事项

并行性问题

因为许多用户访问和更改关系数据库中的数据，所以数据库管理器必须能够允许用户进行这些更改并确保保护数据完整性。*并行性*指的是可以同时由多个交互式用户或应用程序共享资源。数据库管理器控制此访问以防止意外的后果，例如:

- **丢失更新**。两个应用程序 A 和 B，可能同时从数据库中读取相同的行并同时根据这些应用程序读取的数据为其中一列计算新的值。如果 A 用它的新值更新该行而 B 随后也更新该行，那么由 A 执行的更新将丢失。
- **访问未落实的数据**。应用程序 A 可能更新数据库中的值，而应用程序 B 在该值被落实前可能会对其进行读取。因此，如果 A 的值稍后未被落实，而是被收回，那么由 B 执行的计算将基于未落实的（可能为无效的）数据。
- **不可重复读**。某些应用程序涉及到以下一系列事件: 应用程序 A 从数据库中读取一行，然后继续处理其他请求。与此同时，应用程序 B 修改或删除该行并落实更改。稍后，如果应用程序 A 试图再次读取原始行，它将接收到已修改的行或发现该原始行已被删除。
- **幻像读现象**。在下列情况下将出现幻像读现象:
  1. 您的应用程序执行一个查询，该查询根据某些搜索条件读取一组行。
  2. 另一个应用程序插入新数据或更新现有的数据以满足您的应用程序查询。
  3. 您的应用程序从步骤 1 开始重复查询（在同一工作单元中）。

     某些附加的（“幻像”）行作为结果集的一部分返回，但在初始执行该查询（步骤 1）时不会返回这些行。

注: 因为已声明临时表只可用于声明它们的应用程序，所以这些临时表没有并行性问题。这种类型的表从应用程序声明它起开始存在，直到应用程序完成或断开连接为止。

联合数据库系统中的并行性控制

*联合数据库系统*支持应用程序和用户落实 SQL 语句，这些 SQL 语句在单个语句中引用两个或多个数据库管理系统（DBMS）或数据库。要引用数据源（它们由一个 DBMS 和数据组成），DB2 使用*昵称*。昵称是其他数据库管理器中对象的别名。在联合系统中，DB2 依靠主管请求的数据的数据库管理器的并行性控制协议。

DB2 联合系统为数据库对象提供*位置透明性*。例如，具有位置透明性，如果移动了有关表和视图的信息，那么可更新通过昵称对该信息的引用，而无需更改请求该信息的应用程序。当应用程序通过昵称访问数据时，DB2 依靠数据源数据库管理器的并行性控制协议来确保隔离级别。尽管 DB2 尝试将数据源的请求隔离级别与等效的逻辑级别匹配，但是，结果可能因数据源的容量而不同。

隔离级别和性能

*隔离级别*确定访问数据时如何锁定数据或使数据不受其他进程影响。该隔离级别将在工作单元运行期间生效。在执行 OPEN CURSOR 的工作单元期间，使用由 WITH HOLD 子句的 DECLARE CURSOR 语句声明的游标的应用程序将保持选定的隔离级别。DB2 支持下列隔离级别:

- 可重复读
- 读稳定性
- 游标稳定性
- 未落实的读。

注: 某些主机数据库服务器支持*不落实*隔离级别。对于其他数据库，此隔离级别的行为与未落实的读隔离级别一样。

每个隔离级别的详细说明按它们对性能的影响程度的降序排列，但按您访问和更新数据时需要加以关心的程度的升序排列。

可重复读

可重复读（RR）会锁定应用程序在工作单元中引用的所有行。利用"可重复读"，在打开游标的相同工作单元内一个应用程序发出一个 SELECT 语句两次，每次都返回相同的结果。利用"可重复读"，不可能出现丢失更新、访问未落实的数据和幻像行的情况。

在该工作单元完成之前，"可重复读"应用程序可以尽可能多次地检索和操作这些行。但是，在该工作单元完成之前其他应用程序均不能更新、删除或插入可能会影响结果表的行。"可重复读"应用程序不能查看其他应用程序的未落实更改。

利用"可重复读"，将会锁定引用的每一行，而不仅仅是检索的那些行。执行了适当的锁定，因此其他应用程序不能插入或更新行（该行可能要添加到查询所引用的行的列表中，如果重新执行查询的话）。这将防止出现幻像行。例如，如果您扫描 10000 行并对它们应用谓词，尽管只有 10 行满足条件，但仍会锁定全部的 10000 行。

注: "可重复读"隔离级别确保在应用程序看到数据之前所有返回的数据都保持不变，即使使用了临时表或行分块也是如此。

由于"可重复读"可能获得和挂起大量锁定，因此这些锁定可能超出可作为 *locklist* 和 *maxlocks* 配置参数的有效结果的锁定数。为了避免锁定升级，优化器在认为很可能会发生锁定升级的时候，可能选择立即获得单个表级别锁定用于索引扫描。这就像数据库管理器代表您发出了一个 LOCK TABLE 语句一样。如果不想获得表级别锁定，确保有足够的锁定可用于该事务或使用"读稳定性"隔离级别。

评估引用约束时，在一些情况下 DB2 将在内部把对子表进行扫描所使用的隔离级别升级到"可重复读"（RR），而无论用户设置的隔离级别是什么。这将导致其他锁定在落实之前一直被挂起，从而增大了出现死锁或锁定超时的可能性。为了避免出现这种情况，建议您创建仅包含一列或多列外键的索引，从而允许 RI 扫描使用此索引。

读稳定性

读稳定性（RS）只锁定应用程序在工作单元中检索的那些行。它确保在某个工作单元完成之前，在该工作单元运行期间的任何限定行读取不被其他应用程序进程更改，且确保不会读取由另一个应用程序进程所更改的任何行，直至该进程落实了这些更改。也就是说，不可能出现“不可重复读”情形。

与可重复读不同，使用“读稳定性”时，如果您的应用程序多次发出相同的查询，那么有可能看到附加的*幻像*行（*幻像读现象*）。重新引用扫描 10000 行的示例时，“读稳定性”只锁定限定的行。这样，使用“读稳定性”时，只检索 10 行，且只对那十行挂起锁定。将它与“可重复读”对比，在此示例中，可重复读会在所有的 10000 行上挂起锁定。挂起的锁定可以是共享、下次共享、更新或互斥锁定。

注：“读稳定性”隔离级别确保在应用程序*看到*数据之前所有返回的数据保持不变，即使使用了临时表或行分块也是如此。

“读稳定性”隔离级别的其中一个目标是提供较高并行性程度以及数据的稳定视图。为了有助于达到此目标，优化器确保在发生锁定升级前不获取表级锁定。

“读稳定性”隔离级别最适用于包括下列所有特征的应用程序:

- 在并发环境下运行
- 需要限定某些行在工作单元运行期间保持稳定
- 在工作单元中不会多次发出相同的查询，或者在同一工作单元中发出多次查询时并不要求该查询获得相同的回答。

游标稳定性

游标稳定性（CS）当在行上定位游标时会锁定任何由应用程序的事务所访问的行。此锁定在读取下一行或终止事务之前有效。但是，如果更改了某一行上的任何数据，那么在对数据库落实更改之前必须挂起该锁定。

对于具有“游标稳定性”的应用程序已检索的行，当该行上有任何可更新的游标时，任何其他应用程序都不能更新或删除该行。“游标稳定性”应用程序不能查看其他应用程序的未落实更改。

再次引用扫描 10000 行的示例，如果使用“游标稳定性”，将只锁定当前游标位置以下的行。当游标移离该行时，也就除去了该锁定（除非更新该行）。

使用“游标稳定性”，可能会出现不可重复读和幻像读现象。“游标稳定性”是缺省隔离级别，且应在需要最大并行性，但只看到其他应用程序中的已落实行的情况下才使用。

未落实的读

未落实的读（UR）允许应用程序访问其他事务的未落实的更改。除非其他应用程序尝试删除或改变该表，否则该应用程序也不会锁定正读取的行而使其他应用程序不能访问该行。对于只读和可更新的游标，“未落实的读”的工作方式有所不同。

只读游标可访问大多数其他事务的未落实的更改。但是，当该事务正在处理时，正由其他事务创建或删除的表、视图和索引不能使用。其他事务的任何其他更改在落实或回滚前都可被读取。

注：“未落实的读”隔离级别下的可更新操作的游标将按隔离级别是游标稳定性的方式工作。

当它使用隔离级别 UR 运行程序时，应用程序可以使用隔离级别 CS。发生这种情况的原因是因为在应用程序中使用的游标是模糊游标。由于 BLOCKING 选项，可以将模糊游标升级为隔离级别 CS。BLOCKING 选项的缺省值是 UNAMBIG。这意味着将模糊游标当作可更新的，并且隔离级别升级为 CS。要防止此升级，有两种选择:

- 修改应用程序中的游标。以便这些游标是非模糊游标。将 SELECT 语句更改为包括 FOR READ ONLY 子句。
- 将模糊游标保留在应用程序中，但是预编译程序或使用 BLOCKING ALL 和 STATICREADONLY YES 选项绑定它以允许在运行该程序时将任何模糊游标视为只读游标。

如对扫描 10000 行的“可重复读”给出的示例一样，如果使用“未落实的读”，那么不需要任何行锁定。

使用“未落实的读”，可能出现不可重复读行为和幻像读现象。“未落实的读”隔离级别最常用于只读表上的查询，或者若仅执行选择语句且不关心是否可从其他应用程序中看到未落实的数据时也最常用。

隔离级别的总结

下表按不期望的结果总结了几个不同的隔离级别。

表 4. 隔离级别总结

隔离级别	访问未落实的数据	不可重复读	幻像读现象
可重复读（RR）	不可能	不可能	不可能
读稳定性（RS）	不可能	不可能	可能
游标稳定性（CS）	不可能	可能	可能
未落实的读（UR）	可能	可能	可能

下表提供了简单的试探方法，以帮助您为应用程序选择初始隔离级别。首先考虑下表列示的方法，并参阅先前对各级因素的讨论，可能会找到另一个更适合的隔离级别。

表 5. 选择隔离级别的准则

应用程序类型	需要高数据稳定性	不需要高数据稳定性
读写事务	RS	CS
只读事务	RR 或 RS	UR

为一个应用程序选择适当的隔离级别对于该应用程序避免无法容忍的现象很重要。因为获取和释放锁定所需的 CPU 和内存资源随隔离级别的不同而不同，所以此隔离级别不但影响应用程序之间的隔离程度，而且还影响个别应用程序的性能特征。潜在的死锁情况随隔离级别的不同而不同。

指定隔离级别

因为隔离级别确定访问数据时如何锁定数据并使数据不受其他进程影响，所以您应该选择平衡并行性和数据完整性需求的隔离级别。您指定的隔离级别在工作单元运行期间生效。

注: 不能在语句级别指定 XQuery 语句的隔离级别。

可以采用几种不同的方式来指定隔离级别。使用下列试探来确定在编译 SQL 或 XQuery 语句时将使用哪种隔离级别:

静态 SQL:

- 如果在语句中指定了隔离子句，那么使用该子句的值。
- 如果在语句中没有指定隔离子句，那么使用的隔离级别是在将程序包绑定至数据库时为该程序包指定的隔离级别。

动态 SQL:

- 如果在语句中指定了隔离子句，那么使用该子句的值。
- 如果在语句中没有指定隔离子句，并且在当前会话中已经发出了 SET CURRENT ISOLATION 语句，那么使用 CURRENT ISOLATION 专用寄存器的值。
- 如果在语句中没有指定隔离子句，并且在当前会话中没有发出 SET CURRENT ISOLATION 语句，那么使用的隔离级别是在将程序包绑定至数据库时为该程序包指定的隔离级别。

静态或动态 XQuery 语句:

- 环境的隔离级别确定升级 XQuery 表达式时的隔离级别。

注: 许多商业上编写的应用程序提供用于选择隔离级别的方法。有关信息，参阅应用程序文档。

要指定隔离级别:

- **在预编译或绑定时**:

  对于用受支持的编译语言编写的应用程序，使用命令行处理器 PREP 或 BIND 命令的 ISOLATION 选项。也可以使用 PREP 或 BIND API 来指定隔离级别。

  – 如果在预编译时创建绑定文件，那么隔离级别存储在绑定文件中。如果在绑定时不指定隔离级别，那么缺省值是预编译期间使用的隔离级别。
  – 如果不指定隔离级别，那么使用游标稳定性的缺省值。

  注: 要确定程序包的隔离级别，执行下列查询:

  ```
  SELECT ISOLATION FROM SYSCAT.PACKAGES
           WHERE PKGNAME = 'XXXXXXXX'
           AND PKGSCHEMA = 'YYYYYYYY'
  ```

  其中 XXXXXXXX 是程序包的名称，而 YYYYYYYY 是程序包的模式名。这两种名称都必须大写。

- **在支持 REXX™ 的数据库服务器上**:

  当创建一个数据库时，会有多个绑定文件与该数据库绑定，这些文件支持 REXX 中的 SQL 的不同隔离级别。当创建一个数据库时，其他命令行处理器程序包也会与该数据库绑定。

  REXX 和命令行处理器使用缺省隔离级别游标稳定性与数据库连接。更改为不同的隔离级别时并不更改连接状态。必须在 CONNECTABLE AND UNCONNECTED 状态下或 IMPLICITLY CONNECTABLE 状态下执行更改。

要验证 REXX 应用程序正在使用的隔离级别，检查 SQLISL REXX 变量的值。每次执行 CHANGE SQLISL 命令时更新该值。

- **在语句级别**:

  使用 WITH 子句。语句级别的隔离级别覆盖为出现语句所在的程序包指定的隔离级别。

  可以指定下列 SQL 语句的隔离级别:

  – SELECT
  – SELECT INTO
  – 搜索的 DELETE
  – 插入
  – 搜索的 UPDATE
  – DECLARE CURSOR

  下列情况适用于为语句指定的隔离级别:

  – 在子查询上不能使用 WITH 子句
  – WITH UR 选项只适用于只读操作。在其他情况下，语句自动从 UR 更改为 CS。

- **运行时从 CLI 或 ODBC**:

  使用 CHANGE ISOLATION LEVEL 命令。对于 DB2 调用级接口（DB2 调用级接口），可以将隔离级别更改为 DB2 调用级接口 配置的一部分。在运行时，将 *SQLSetConnectAttr* 函数与 SQL_ATTR_TXN_ISOLATION 属性配合使用，来设置由 *ConnectionHandle* 引用的当前连接的事务隔离级别。也可以在 db2cli.ini 文件中使用 TXNISOLATION 关键字。

- **当在运行时使用 JDBC 或 SQLJ 时**:

  注: JDBC 和 SQLJ 是通过 DB2 上的 CLI 实现的，这意味着 db2cli.ini 设置可能影响使用 JDBC 和 SQLJ 编写和运行的内容。

  在 SQLJ 中，使用 SQLJ 概要文件定制程序（db2sqljcustomize 命令）来创建程序包。可为此程序包指定的选项包括其隔离级别。

- **对于当前会话中的动态 SQL**:

  使用 SET CURRENT ISOLATION 语句来设置在会话中发出的动态 SQL 语句的隔离级别。发出此语句将把 CURRENT ISOLATION 专用寄存器设置一个值，该值用来指定在当前会话中发出的任何动态 SQL 语句的隔离级别。一旦设置了该值，CURRENT ISOLATION 专用寄存器就会为在会话中编译的任何后续动态 SQL 语句的隔离级别，无论程序包是否发出了该语句。此隔离级别将适用，直到会话结束或者使用 RESET 选项发出了 SET CURRENT ISOLATION 语句为止。

锁定与并行性控制

要提供并行性控制并防止无控制的数据访问，数据库管理器将锁定置于缓冲池、表、数据分区、表块或表行上。*锁定*将数据库管理器资源与应用程序关联（称为*锁定所有者*），以控制其他应用程序访问同一资源的方式。

数据库管理器根据下列各项来适当地使用行级锁定或表级锁定:

- 隔离级别是在预编译或应用程序与数据库绑定时指定的。隔离级别可以是下列其中一项:
  - 未落实的读（UR）
  - 游标稳定性（CS）
  - 读稳定性（RS）
  - 可重复读（RR）

  使用不同的隔离级别来控制对未落实的数据的访问、防止丢失更新、允许对数据进行不可重复的读取以及防止幻像读取。要使性能影响最小，请使用能够满足应用程序需要的最小隔离级别。
- 优化器选择的访问方案。表扫描、索引扫描和其他数据访问方法其每一个都需要不同的数据访问类型。
- 表的 LOCKSIZE 属性。ALTER TABLE 语句上的 LOCKSIZE 子句指示访问该表时使用的锁定的详细程度。选项是 ROW（表示行锁定）、TABLE（表示表锁定）或 BLOCKINSERT（仅表示对 MDC 表的块锁定）。当对 MDC 表使用 BLOCKINSERT 子句时，除了执行块级别锁定的 INSERT 操作之外，将执行行级别锁定。当事务执行大量插入到不相交的单元格的操作时，应对 MDC 表使用 ALTER TABLE ... LOCKSIZE BLOCKINSERT 语句。这减少了数据库活动所需的锁定数。对于分区表，将按所访问的数据指示的那样首先获取表锁定，然后获取数据分区锁定。
- 专门用于锁定的内存量。专门用于锁定的内存量由 `locklist` 数据库配置参数控制。如果锁定列表已满，那么性能可能会因锁定升级以及数据库中共享对象上的并行性降低而下降。如果锁定升级频繁发生，增大 `locklist` 或 `maxlocks` 的值，或同时增大它们的值。此外，要减少同时挂起的锁定数，确保事务频繁落实以释放挂起的锁定。

虽然大多数锁定是对表发生的，但在创建、改变或删除缓冲池时，将设置缓冲池锁定。与此锁定配合使用的方式是 EXCLUSIVE (X)。收集系统监视的数据时可能会遇到这种类型的锁定。当查看快照时，您将看到使用的锁定名就是缓冲池本身的标识（标识）。

通常，除非下列其中一项为真，否则使用行级锁定:

- 选择的隔离级别是未落实的读（UR）。
- 选择的隔离级别是可重复的读（RR），访问方案需要不带谓词的扫描。
- 表 LOCKSIZE 属性是“TABLE”。
- 锁定列表已填充，引起升级。
- 通过 LOCK TABLE 语句获取了显式的表锁定。LOCK TABLE 语句使并发应用程序进程不能更改表或使用表。

如果这是 MDC 表，那么存在其他几种使用块级别锁定而不是行级别锁定的情况，这些情况包括:

- 表 LOCKSIZE 属性是“BLOCKINSERT”
- 选择的隔离级别是可重复的读（RR），访问方案需要谓词
- 仅涉及维列上的谓词的搜索的更新或删除操作

锁定升级降低了并行性。应该避免可能引起锁定升级的情况。

行锁定的持续时间随使用的隔离级别的不同而有所变化:

- UR 扫描：除非行数据正在进行更改，否则不挂起行锁定。
- CS 扫描：仅当游标定位在行上时，才挂起行锁定。
- RS 扫描：完成事务期间只挂起合格的行锁定。
- RR 扫描：完成事务期间挂起所有行锁定。

锁定属性

数据库管理器锁定具有下列基本属性：

方式 允许锁定所有者使用的访问类型以及允许锁定对象的并发用户使用的访问类型。有时称之为锁定的*状态*。

对象 正被锁定的资源。您可以显式锁定的唯一对象类型是表。数据库管理器也可强制锁定其他类型的资源，例如行、表和表空间。对于多维集群（MDC）表，也可以强制进行块锁定；对于分区表，可以强制进行数据分区锁定。正被锁定的对象确定锁定的*详细程度*。

持续时间
挂起锁定的时间长度。该查询运行的隔离级别影响锁定持续时间。

下表按照对资源的控制性递增的顺序显示锁定方式及其效果。有关各种级别的锁定的详细信息，参阅锁定方式引用表。

表 6. 锁定方式总结

锁定方式	适用的对象类型	描述
IN（无意向）	表空间、块、表和数据分区	锁定所有者可以读取对象中的任何数据，包括未落实的数据，但不能更新它的任何数据。其他并发应用程序可以读取或更新表。
IS（意向共享）	表空间、块、表和数据分区	锁定所有者可读取已锁定的表中的数据，但不能更新此数据。其他应用程序可以读取或更新表。
NS（下一键共享）	行	锁定所有者和所有并发应用程序可读取但不能更新锁定的行。在应用程序的隔离级别为 RS 或 CS 的情况下，对一个表的行获取此锁定，以代替 S 锁定。NS 锁定方式不用于下一键锁定。它在 CS 和 RS 扫描期间用来代替 S 方式以最小化下一键锁定对这些扫描的影响。
S（共享）	行、块、表和数据分区	锁定所有者和所有并发应用程序可读取（但不能更新）锁定的数据。
IX（意向互斥）	表空间、块、表和数据分区	锁定所有者和并发应用程序可以读取和更新数据。其他并发应用程序既可以读取也可以更新表。
SIX（在意向互斥下共享）	表、块和数据分区	锁定所有者可以读取和更新数据。其他并发应用程序可以读取该表。
U（更新）	行、块、表和数据分区	锁定所有者可以更新数据。其他工作单元可以读取锁定的对象中的数据，但不能尝试对其进行更新。
NW（下一键弱互斥）	行	当将行插入索引时，获取对下一行的 NW 锁定。对于 2 类索引，仅当下一行当前由 RR 扫描锁定时才发生此情况。锁定所有者可以读取但不可更新已锁定的行。除了与 W 和 NS 锁定兼容之外，此锁定方式类似于 X 锁定。
X（互斥）	行、块、表、缓冲池和数据分区	锁定所有者既可以读取也可以更新锁定的对象中的数据。只有未落实的读应用程序可以访问已锁定的对象。
W（弱互斥）	行	当将行插入未定义 2 类索引的表时，即获取对该行的此种锁定。锁定所有者可更改已锁定的行。要确定重复值在其被找到时是否已落实，此锁定也用于插入唯一索引期间。除了与 NW 锁定兼容之外，此锁定类似于 X 锁定。只有未落实的读应用程序可以访问已锁定的行。

表 6. 锁定方式总结 (续)

锁定方式	适用的对象类型	描述
Z（超级互斥）	表空间、表和数据分区	对表的此种锁定是在一定条件下获取的，例如，当改变或删除表、创建或删除表的索引或对某些类型的表重组时。没有其他的并发应用程序可以读取或更新该表。

锁定详细程度

如果某个应用程序挂起某个数据库对象上的锁定，那么另一个应用程序可能不能访问该对象。因此，锁定最少量数据并使这些数据不可访问的行级别锁定比块级别、数据分区级别或表级别对于最大化并行性更好。但是，锁定需要存储器和处理时间，因此单个表锁定最小化锁定开销。

ALTER TABLE 语句的 LOCKSIZE 子句指定行级别、数据分区级别、块级别或表级别的锁定的作用域（详细程度）。缺省情况下，使用行锁定。由这些已定义的表锁定仅请求 S（共享）和 X（互斥）锁定。ALTER TABLE 语句的 LOCKSIZE ROW 子句不会阻止正常的锁定升级发生。

在下列情况下，由 ALTER TABLE 语句定义的永久表锁定可能比使用 LOCK TABLE 语句来获得单个事务表锁定更可取:

- 表是只读的，且将始终只需要 S 锁定。其他用户也可以获取表的 S 锁定。
- 表通常由只读应用程序访问，但有时由单个用户访问以进行简要维护，而该用户需要 X 锁定。当维护程序运行时，将只读应用程序锁定在外，但在其他情况下，只读应用程序可以使用最小的锁定开销同时访问表。

对于 MDC 表，可以对 LOCKSIZE 子句指定 BLOCKINSERT 以便只在 INSERT 操作期间使用块级别锁定。当指定此项时，会对所有其他操作执行行级别锁定，但极少会对 INSERT 操作执行行级别锁定。即，在插入行时使用块级别锁定，但如果更新索引时在索引中遇到 RR 扫描，那么会使用行级别锁定作为下一键锁定。BLOCKINSERT 锁定在下列情况下可能有用:

- 多个事务正在执行对各个单元格的大量插入操作。
- 未发生多个事务并发插入至同一个单元格的情况，或者发生了这种情况，但由于每个事务向每个单元格插入足够的数据，所以用户并不关心每个事务将插入到不同的块中。

ALTER TABLE 语句全局指定锁定，它影响访问该表的所有应用程序和用户。单个应用程序可以使用 LOCK TABLE 语句来指定应用程序级别的表锁定。

锁定等待和超时

锁定超时检测是一个数据库管理器功能，它防止应用程序在异常情况下无限地等待释放锁定。例如，一个事务可能正等待另一个用户的应用程序挂起的锁定，但该用户已离开工作站，而没有允许应用程序落实释放该锁定的事务。要避免在这种情况下停止应用程序，将 **locktimeout** 配置参数设置为任何应用程序应等到获取锁定的最长时间。

设置此参数有助于避免全局死锁，特别是在分布式工作单元（DUOW）应用程序中。如果锁定请求处于暂挂的时间大于 **locktimeout** 值，那么请求应用程序将接收到一个错误并将其事务回滚。例如，如果 APPL1 尝试获取一个已由 APPL2 挂起的锁定，那么

APPL1 在超时周期到期时返回 SQLCODE -911（SQLSTATE 40001），原因码为 68。**locktimeout** 的缺省值为 -1，它关闭锁定超时检测。

注: 对于表、行、数据分区和 MDC 块锁定，应用程序可使用 SET CURRENT LOCK TIMEOUT 来覆盖数据库级别 **locktimeout** 设置。

要生成关于锁定超时的报告文件，将 **DB2_CAPTURE_LOCKTIMEOUT** 注册表变量设置为 ON。锁定超时报告包括关于导致锁定超时的锁定争用中所涉及的关键应用程序的信息，以及关于锁定本身的详细信息，如锁定名称、锁定类型、行标识、表空间标识和表标识。

要将有关锁定请求超时的更多信息记入 db2diag.log 文件，将数据库管理器配置参数 **diaglevel** 设置为 4。记录的信息包括已锁定的对象、锁定方式以及挂起有锁定的应用程序。还可能会记录当前的动态 SQL 或 XQuery 语句或静态程序包名称。仅在 **diaglevel** 4 记录动态 SQL 或 XQuery 语句。

可从锁定等待信息系统监视元素或 **db.apps_waiting_locks** 运行状况指示器获取有关锁定等待数和锁定超时的信息。

锁定超时报告

锁定超时报告功能捕获关于锁定超时事件的信息，包括关于导致锁定超时的锁定争用中所涉及的关键应用程序的信息。此功能由 **DB2_CAPTURE_LOCKTIMEOUT** 注册表变量控制。

将同时捕获关于锁定请求者（接收锁定超时错误的应用程序）和当前锁定所有者的信息，并且报告存储在数据库配置参数所确定的目录中。

如果在锁定超时报告功能处于活动状态时发生锁定超时，那么将捕获下列信息并保存在锁定超时报告文件中。

争用的锁定

- 锁定名称和类型
- 锁定详细说明，包括行标识、表空间标识和表标识。使用此信息来查询 SYSCAT.TABLES 系统目录视图以标识表名。

锁定请求者

- 应用程序标识信息，如应用程序名称和协调程序分区
- 锁定超时值
- 请求的锁定方式
- 请求 SQL 上下文（如果适用的话）
- 程序包标识
- 段条目号
- SQL 信息，例如，语句是动态还是静态的，语句的类型
- 有效的锁定隔离
- 相关语句文本（如果适用的话，请参阅 第 154 页的『功能局限性』 以了解详细信息。）
- 应用程序状态
- 当前操作

- 锁定升级

锁定所有者或代表

可能有多个锁定所有者。例如，当某个锁定被多个应用程序以共享方式拥有时，仅报告关于遇到的第一个锁定所有者的信息。第一个锁定所有者是其他锁定所有者的代表。

- 应用程序标识信息，如应用程序名称和协调程序分区
- 拥有的锁定方式
- 此分区中当前活动的 SQL 语句的列表
  1. 程序包标识
  2. 段条目号
  3. SQL 信息，例如，语句是动态还是静态的，语句的类型
  4. 有效的锁定隔离
  5. 相关语句文本（在可用时）
- 此数据库分区的当前工作单元中的不活动 SQL 语句的列表（仅当具有语句历史记录的死锁事件监视器处于活动状态时才可用）
  1. 程序包标识
  2. 段条目号
  3. SQL 信息，例如，语句是动态还是静态的，语句的类型
  4. 有效的锁定隔离
  5. 相关语句文本（在可用时）

要收集其他信息（例如，操作系统或其他相关环境信息），请使用定制 db2cos 脚本。

功能用途

DB2_CAPTURE_LOCKTIMEOUT 注册表变量控制锁定超时报告功能。

当此注册表变量设置为 ON 时，该功能捕获关于 DB2 实例中发生的每个锁定超时的基本信息，并且创建锁定超时报告。如果该注册表变量设置为空，那么该功能被禁用。

要启用锁定超时报告功能，请发出以下命令:

```
db2set DB2_CAPTURE_LOCKTIMEOUT=ON
```

要禁用锁定超时报告功能，请发出以下命令:

```
db2set DB2_CAPTURE_LOCKTIMEOUT=
```

在某些情况下，您可能希望捕获发生锁定超时之前在锁定所有者的工作单元中执行的所有语句。例如，如果锁定超时是由先前执行的 SQL 语句产生的，那么这样做很有用。要将工作单元以前的历史记录包括在当前锁定所有者的锁定超时信息中，使用语句历史记录子句激活死锁事件监视器。例如，使用下列其中一个语句:

```
create event monitor testit for deadlocks with details history write to file
path global
create event monitor testit for deadlocks with details history write to table
```

CREATE EVENT MONITOR 语句具有其他选项，例如，用于指定将写入数据的表空间和表名称的选项。有关详细信息，请参阅 CREATE EVENT MONITOR 语句描述。

具有语句历史记录功能的事件监视器影响所有应用程序并增大 DB2 数据库管理器使用的监视器堆。仅当锁定争用不是由于所涉及应用程序正在执行的当前语句导致的，或者报告的目的不仅仅是标识锁定超时中涉及的应用程序时，才使用这种事件监视器。

例如，应用程序 APPL1 执行下列语句:

```
INSERT INTO T1 VALUES (1)
SELECT * FROM T5
```

接着，APPL2 执行以下语句:

```
SELECT * FROM T1
```

在此示例中，APPL2 执行的 SELECT 语句正在等待 APPL1 执行的先前 INSERT 语句中获取的行锁定。在这种情况，只看到 APPL1 当前执行的 SELECT 语句不能帮助确定锁定争用的具体原因。锁定所有者的语句历史记录将显示先前 INSERT 语句和锁定争用的原因。

如果拥有锁定的应用程序在锁定超时的情况下未运行 SQL 语句，那么创建具有语句历史记录功能的事件监视器也有帮助。如果具有历史记录的死锁事件监视器处于活动状态，那么工作单元中应用程序的先前 SQL 语句将被写入报告中。

功能局限性

锁定超时报告并非总是捕获关于锁定争用和锁定超时事件的必需详细信息。在某些情况下（如下面列示的情况），并非所有信息都可用于锁定超时报告功能:

- SQL 语句文本并非在所有情况下都可用，例如，当锁定超时中涉及到静态 SQL 语句时不可用。
- DB2 实用程序和内部函数可以在不运行 SQL 的情况下获取锁定。例如，联机备份实用程序可以在联机备份期间处理表空间和表时获取这些对象的锁定。在备份表的同时，应用程序在等待锁定时可能超时。如果从 CLP 执行备份命令，那么应用程序名称将报告为“db2bp”（CLP），并且锁定所有者将没有活动的 SQL 语句。DB2 实用程序或内部函数在等待应用程序释放锁定时也可能会超时，并且超时请求者没有活动 SQL 语句。
- 对远程分区中应用程序的行为起作用的子代理程序可能具有关于该应用程序 SQL 语句的不完整信息和其他信息。特别是，如果锁定所有者是远程子代理程序，那么它将没有可用的应用程序的完整语句历史记录。
- 在常规 DB2 操作期间执行的内部目录表处理不使用 SQL。该功能将只标识涉及的表，但不标识导致或遇到冲突的 DB2 组件。
- 某些锁定未被特定 SQL 语句获取。它们可以表示非表对象（如程序包）或用于其他内部 DB2 处理。

锁定超时报告文件

在锁定超时报告功能处于活动状态的情况下，如果发生锁定超时，那么接收锁定超时错误的代理程序将生成一个报告文件。此文件放置在发生锁定超时错误的数据库分区的诊断数据目录路径中。

诊断数据目录路径由数据库管理器配置参数 **diagpath** 定义。如果此参数为空，请参阅 **diagpath** 参数描述以了解报告文件的位置。

在分区数据库环境中，一个应用程序可以同时在一个或多个数据库分区中执行工作并获取锁定。通过在每个发生锁定超时问题的数据库分区中生成报告，锁定超时报告功能可帮助隔离特定数据库分区。报告包括关于该分区中发生的锁定超时问题的唯一上下文相关信息。

报告存储在使用以下名称格式的文件中：db2locktimeout.*par*.*AGENTID*.*yyyy-mm-dd-hh-mm-ss*，其中

- *par* 是数据库分区号。在非分区数据库环境中，*par* 设置为 0。
- *AGENTID* 是代理程序标识。
- *yyyy-mm-dd-hh-mm-ss* 是时间戳记，由年、月、日、小时、分钟和秒组成。

以下是一个锁定超时报告文件名的示例：/home/juntang/sqllib/db2dump/db2locktimeout.000.4944050.2006-08-11-11-09-43。

注：当不再需要锁定超时报告文件时，从目录中将其删除。由于报告文件与其他诊断日志位于相同位置中，因此在允许该目录变满的情况下，DB2 系统可能会关闭。如果需要将某些锁定超时报告文件保留很长一段时间，那么将它们移至另一个目录或文件夹。

锁定转换

更改已挂起的锁定方式称为*转换*。当一个进程访问它已挂起锁定的数据对象，且访问方式需要比已挂起的锁定更严格的锁定时，将进行锁定转换。一个进程在任一时间对数据对象只能挂起一个锁定，尽管它可以通过查询间接地对同一数据对象多次请求锁定。

某些锁定方式仅适用于表，而有些锁定方式仅适用于行或块。对于行或块，如果需要 X 且挂起 S 或 U（更新）锁定，那么进行转换。

但是，在锁定转换这一点上，IX（意向互斥）和 S（共享）锁定是特殊情况。S 和 IX 锁定的严格程度相当，所以如果挂起了一个而请求另一个，那么结果转换为 SIX（带意向互斥的共享）锁定。如果所请求的方式更严格的话，所有其他的转换都导致所请求的锁定方式变成所挂起的锁定方式。

当查询更新行时，也可以发生双重转换。如果行是通过索引访问读取的且锁定为 S，那么包含该行的表具有覆盖意向锁定。但是，如果锁定类型是 IS 而不是 IX，且随后更改了行，那么表锁定将转换为 IX，而行锁定转换为 X。

在执行查询时，锁定转换通常隐式进行。了解不同的查询以及表和索引的组合获得的锁定种类，有助于您设计和调整您的应用程序。

系统监视元素 *lock_current_mode* 和 *lock_mode* 可提供有关数据库中发生的锁定转换的信息。

防止与锁定相关的性能问题

调整锁定以实现并行性和数据完整性时，应考虑下列准则：

- 频繁使用 COMMIT 语句来创建较小的工作单元以使许多用户并发访问数据。

  当应用程序在逻辑上一致时，即当更改的数据一致时，包括 COMMIT 语句。当发出 COMMIT 时，释放锁定，与声明了 WITH HOLD 的游标相关的表锁定除外。

- 在发出 COMMIT 语句之前，应先关闭 CURSOR WITH HOLD。

  在某些情况下，在结果集关闭并且事务落实后锁定仍然存在。在发出 COMMIT 语句之前关闭 CURSOR WITH HOLD 可确保释放锁定。
- 作为单独的工作单元执行 INSERT 语句

  在某些情况下，在结果集关闭并且事务落实后锁定仍然存在。作为单独的工作单元执行 INSERT 语句可确保释放锁定。
- 指定适当的隔离级别。

  即使应用程序很少读取行，也会获得锁定，所以落实只读工作单元仍很重要。这是因为在只读应用程序中通过可重复读、读稳定性及游标稳定性隔离级别获得共享锁定。借助于可重复读和读稳定性，可挂起所有锁定，直到发出 COMMIT，这可阻止其他进程更新锁定的数据，除非您使用 WITH RELEASE 子句关闭您的游标。此外，甚至在使用动态 SQL 或 XQuery 语句的未落实的读应用程序中也要获得目录锁定。

  数据库管理器确保应用程序不检索未落实的数据（其他应用程序已经更新但尚未落实的行），除非正在使用未落实的读隔离级别。
- 适当地使用 LOCK TABLE 语句。

  该语句锁定整个表。但只锁定 LOCK TABLE 语句中指定的表。不锁定指定表的父表和从属表。必须确定是否需要锁定其他可访问的表以便在并行性与性能方面达到期望的结果。在工作单元被落实或回滚之前不释放锁定。

  以共享方式锁定表

  想要访问*在时间上一致的*数据；即，表在特定时间点的最新数据。如果表活动频繁，那么确保整个表保持稳定的唯一方法是将其锁定。例如，应用程序想要抽取表的快照。但是，在应用程序需要处理表的一些行期间，其他应用程序正在更新还没有处理的行。可重复读允许这样的操作，但您不希望这样。

  另一种方法是，应用程序可以发出 LOCK TABLE IN SHARE MODE 语句: 无论是否检索到行，都不能更改任何行。然后可以按需要检索任意多行，应了解已经检索的行就在检索它们之前还没有被更改。

  使用 LOCK TABLE IN SHARE MODE，其他用户可从表中检索数据，但不能更新、删除表中的行，或将行插入表中。

  以互斥方式锁定表

  想要更新表的大部分。相对于更新表时锁定每一行，然后在落实所有更改时解锁行而言，阻止所有其他用户访问该表的开销少并且更有效。

  使用 LOCK TABLE IN EXCLUSIVE MODE，所有其他用户都被锁定在外; 没有其他应用程序可以访问该表，除非它们是未落实的读应用程序。
- 在应用程序中使用 ALTER TABLE 语句。

  带 LOCKSIZE 参数的 ALTER TABLE 语句是 LOCK TABLE 语句的备用语句。LOCKSIZE 参数允许您指定下一次表访问的 ROW 锁定或 TABLE 锁定的锁定详细程度。对于 MDC 表，它还允许您指定 BLOCKINSERT 子句的锁定详细程度。

  当新建一个表时，选择 ROW 锁定同选择缺省锁定大小无任何区别。选择 TABLE 锁定可提高查询性能，方法是限制需要获取的锁定的数目。但是，并行性可能由于所

有锁定位于完整的表上而降低。对于 MDC 表，选择 BLOCKINSERT 子句可提高 INSERT 操作的性能，方法是在块级别锁定而避免对插入使用行锁定。仍然会对所有其他操作执行行级别锁定并对键插入执行行级别锁定以保护可重复读（RR）扫描程序。BLOCKINSERT 选项对于单个事务大量插入到单元格的操作非常有用。没有 LOCKSIZE 选项会阻止正常的锁定升级。

- 关闭游标以释放它们挂起的锁定。

  当用包括 WITH RELEASE 子句的 CLOSE CURSOR 语句关闭游标时，数据库管理器尝试释放所有为游标挂起的读锁定。表读锁定是 IS、S 和 U 表锁定。行读锁定是 S、NS 和 U 行锁定。块读锁定是 IS、S 和 U 块锁定。

  WITH RELEASE 子句对正在 CS 或 UR 隔离级别下操作的游标不起作用。当对正在 RS 或 RR 隔离级别下操作的游标指定 WITH RELEASE 子句，它将结束那些隔离级别的一些保证。明确地说，RS 游标可能经历*不可重复读*现象，而 RR 游标可能经历*不可重复读*或*幻象读取*现象。

  如果一个原来是 RR 或 RS 的游标通过使用 WITH RELEASE 子句关闭后重新打开，那么获得新的读锁定。

  在 CLI 应用程序中，DB2 CLI 连接属性 SQL_ATTR_CLOSE_BEHAVIOR 可用来获得与 CLOSE CURSOR WITH RELEASE 相同的结果。
- 在分区数据库环境中，当更改影响锁定的配置参数时，确保对所有数据库分区都作了更改。

更正锁定升级问题

数据库管理器可以自动将锁定从行或块级别升级为表级别。对于分区表，数据库管理器可以自动将锁定从行或块级别升级为数据分区级别。*maxlocks* 数据库配置参数指定触发锁定升级的时间。获取触发锁定升级的锁定的表可能不受影响。首先对具有大多数锁定的表升级锁定，并从锁定其长对象（LOB）和长 VARCHAR 描述符的表开始，然后是具有次高数锁定的表，依此类推，直到挂起的锁定数减少至大约为 *maxlocks* 指定的值的一半。

在设计良好的数据库中，很少发生锁定升级。如果锁定升级将并行性降低到不可接受的程度（由 *lock_escalation* 监视元素或 *db.lock_escal_rate* 运行状况指示器指示），那么需要分析问题并决定如何解决此问题。

确保记录锁定升级信息。将数据库管理器配置参数 *notifylevel* 设置为 3（这是缺省值）或设置为 4。在 *notifylevel* 为 2 时，仅报告错误 SQLCODE。在 *notifylevel* 为 3 或 4 时，当锁定升级失败时，记录错误 SQLCODE 的信息和对其升级失败的表。只有在当前 Query 语句是当前执行的动态 Query 语句且 *notifylevel* 设置为 4 时才记录该语句。

遵循这些一般步骤来诊断不可接受的锁定升级的原因并应用一个修正措施:

1. 在管理通知日志中对其升级锁定的所有表进行分析。 此日志文件包括下列信息:
   - 当前挂起的锁定数目。
   - 在锁定升级完成之前所需的锁定数目。
   - 升级的每个表的表标识信息和表名。
   - 当前挂起的非表锁定的数目。

- 作为升级的一部分，获取新表级别的锁定。通常，将获取“S”（即，共享锁定）或“X”（即，互斥锁定）。
- 获取新表锁定级别结果的内部返回码。

2. 使用管理通知日志中的信息来决定如何解决升级问题。 考虑下列可能性:
   - 通过增大数据库配置文件中 *maxlocks* 和/或 *locklist* 参数的值来增加全局允许的锁定数。在分区数据库中，对所有数据库分区进行此更改。

     如果其他进程对该表的并发访问很重要，可以选择此方法。但是，获得记录级别锁定的开销可能使其他进程延迟，这些延迟比并发访问表节省下来的延迟更多。
   - 调整引起升级的一个或几个进程。对于这些进程，您可以显式发出 LOCK TABLE 语句。
   - 更改隔离度。但是，注意这可能导致降低并行性。
   - 增大落实的频率以减少在给定时间挂起的锁定数。
   - 为需要长 VARCHAR 或各种长对象（LOB）数据的事务考虑频繁的 COMMIT 语句。尽管在结果集具体化之前不从磁盘检索这种数据，但在首次引用数据时锁定描述符。因此，可以挂起比对包含更普通的数据的行多的锁定。

通过锁定延迟来对未落实的数据求值

为了提高并行性，DB2 现在允许在某些情况下对 CS 或 RS 隔离扫描行锁定，直到知道一条记录满足查询的谓词为止。 缺省情况下，在表扫描或索引扫描期间执行行锁定时，DB2 会先锁定已扫描的每一行，然后再确定该行是否符合查询要求。为了提高扫描的并行性，可以延迟行锁定，直到确定某行符合查询要求为止。

要利用此功能，应启用 DB2_EVALUNCOMMITTED 注册表变量。

当启用了此变量时，可对未落实的数据进行谓词求值。这意味着包含未落实的更新的行可能不符合查询要求，但是，如果等到完成更新的事务之后进行谓词求值，那么该行可能会符合查询要求。另外，在表扫描期间会跳过未落实的已删除行。如果启用了 DB2_SKIPDELETED 注册表变量，那么 DB2 在 2 类索引扫描中将跳过已删除的键。

这些注册表变量在编译时适用于动态 SQL 或 XQuery 语句，而在绑定时适用于静态 SQL 或 XQuery 语句。这意味着，即使在运行时启用了该注册表变量也不会使用避免锁定策略（除非在绑定时启用了 DB2_EVALUNCOMMITTED）。如果在绑定时启用了该注册表变量，但是在运行时未启用它，那么避免锁定策略仍然有效。对于静态 SQL 或 XQuery 语句，如果重新绑定了程序包，那么绑定时的注册表变量设置是适用的设置。静态 SQL 或 XQuery 语句的隐式重新绑定将使用 DB2_EVALUNCOMMITTED 的当前设置。

对不同访问方案的未落实数据求值的适用性

表 7. “仅 RID 索引”访问

谓词	对未落实的数据求值
无	否
SARGable	是

表 8. “仅数据”访问（关系 RID 列表或延迟 RID 列表）

谓词	对未落实的数据求值
无	否
SARGable	是

表 9. “RID 索引和数据”访问

谓词		对未落实的数据求值	
索引	数据	索引访问	数据访问
无	无	否	否
无	SARGable	否	否
SARGable	无	是	否
SARGable	SARGable	是	否

表 10. 块索引和数据访问

谓词		对未落实的数据求值	
索引	数据	索引访问	数据访问
无	无	否	否
无	SARGable	否	是
SARGable	无	是	否
SARGable	SARGable	是	是

示例

以下示例对缺省锁定行为与新的对未落实的数据进行求值的行为作了比较。

下表是 SAMPLE 数据库中的 ORG 表。

```
DEPTNUMB DEPTNAME       MANAGER DIVISION   LOCATION
-------- -------------- ------- ---------- -------------
      10 Head Office        160 Corporate  New York
      15 New England         50 Eastern    Boston
      20 Mid Atlantic        10 Eastern    Washington
      38 South Atlantic      30 Eastern    Atlanta
      42 Great Lakes        100 Midwest    Chicago
      51 Plains             140 Midwest    Dallas
      66 Pacific            270 Western    San Francisco
      84 Mountain           290 Western    Denver
```

正在对此表执行下列事务，它具有缺省“游标稳定性”（CS）隔离级别。

表 11. 对具有 CS 隔离级别的 ORG 表执行的事务

会话 1	会话 2
`connect to SAMPLE`	`connect to SAMPLE`
`+c update org set deptnumb=5 where manager=160`	
	`select * from org where deptnumb >= 10`

会话 1 中未落实的 UPDATE 对表中第一行保持互斥记录锁定，禁止会话 2 中的 SELECT 查询返回，即使会话 1 中正在更新的行当前不符合会话 2 中的查询要求也是

如此。这是因为 CS 隔离级别要求当游标位于查询访问的任何一行时必须锁定该行。只有在会话 1 释放对第一行的锁定之后，会话 2 才能获得对该行的锁定。

当对表进行扫描时，可以使用对未落实的数据进行求值这项功能来避免会话 2 中的锁定等待，该功能首先对谓词求值，然后锁定谓词求值结果为 true 的行。这样，会话 2 中的查询将不尝试锁定表中的第一行，从而提高了应用程序并行性。注意，这也意味着对于会话 1 中 deptnumb=5 的未落实的值在会话 2 中将进行谓词求值。尽管会话 1 中对更新的回滚将符合会话 2 中的查询要求，但是会话 2 中的查询仍将忽略它的结果集中的第一行。

如果保留了操作顺序，那么通过对未落实的数据进行求值仍然可提高并行性。通过使用缺省锁定行为，会话 2 将首先获取行锁定，禁止执行会话 1 中搜索到的 UPDATE，即使会话 1 中的 UPDATE 将不更改会话 2 的查询锁定的行也是如此。如果会话 1 中搜索到的 UPDATE 首先尝试了检查一些行，然后仅在这些行符合条件时才锁定它们，那么会话 1 中的查询将不会分块。

限制

下列外部限制适用于此新功能:

- 必须启用注册表变量 DB2_EVALUNCOMMITTED。
- 隔离级别必须是 CS 或 RS。
- 要发生行锁定。
- 存在 SARGable 求值谓词。
- 未落实的数据求值不适用于对目录表的扫描。
- 对于 MDC 表，可以对索引扫描延迟块锁定；但是，将不对表扫描延迟块锁定。
- 对于正在执行适当的表重组的表，将不会发生延迟锁定。
- 对于索引为类型 1 的索引扫描，将不会发生延迟锁定。
- 对于扫描预取方案，行锁定不会延迟到进行数据访问，而是在移至表中的某行之前在索引访问期间就锁定该行。
- 对于表扫描，会无条件地跳过已删除的行，但是，仅当启用了注册表变量 DB2_SKIPDELETED 时，才会跳过已删除的 2 类索引键。

用于忽略未落实插入的选项

DB2_SKIPINSERTED 注册表变量控制对于使用游标稳定性（CS）或读稳定性（RS）隔离级别的游标是否可以忽略未落实插入。DB2 数据库系统可用下列方式处理未落实插入:

- DB2 数据库系统可以等待直到 INSERT 事务完成（落实或回滚），然后相应地处理数据。这是缺省选项 OFF。

  以下示例显示了首选缺省选项 OFF 的实例:

  – 假定两个应用程序使用一个表来在它们之间传递数据，第一个应用程序将数据插入到表中，第二个应用程序从表中读取数据。第二个应用程序必须按照数据在表中的显示顺序来处理数据，如果第一个应用程序正在插入要读取的下一行，那么第二个应用程序必须等待直到插入被落实为止。在这种情况下，应使用 DB2_SKIPINSERTED 的缺省值。

– 假定一个应用程序通过删除数据然后插入数据的新映像来修改数据。在这种情况下，要避免使用 UPDATE 语句，应使用 DB2_SKIPINSERTED 的缺省值。

- DB2 数据库系统可以忽略未落实的插入，这在许多情况下可以提高并行性。如果您希望这样做，必须将该注册表变量指定为 ON。

  总之，ON 会使并行性更强，它是大部分应用程序的首选设置。当启用该注册表变量时，未落实的插入行被视为尚未插入的行。

锁定类型兼容性

在一个应用程序当前拥有对某个对象的锁定，而另一个应用程序请求对同一个对象的锁定时，就会出现锁定兼容性问题。当两种锁定方式兼容时，可以同意对该对象的第二个锁定请求。

如果请求的锁定的锁定方式与已挂起的锁定不兼容，那么不能同意锁定请求。相反，请求要等到第一个应用程序释放其锁定，并且释放所有其他现有不兼容锁定为止。

下表显示有关某些情况的信息，在这些情况下，在给定状态下，当另一个进程挂起或正在请求对同一个资源的锁定时，可以同意锁定请求。**否**指示请求者必须等待，直到所有不兼容的锁定被其他进程释放为止。注意，当请求者正等待锁定时，可能出现超时。**是**指示授予该锁定，除非更早的请求者正在等待该资源。

表 12. 锁定类型兼容性

	占有资源的状态											
请求的状态	**无**	**IN**	**IS**	**NS**	**S**	**IX**	**SIX**	**U**	**X**	**Z**	**NW**	**W**
无	是	是	是	是	是	是	是	是	是	是	是	是
IN	是	是	是	是	是	是	是	是	是	否	是	是
IS	是	是	是	是	是	是	是	是	否	否	否	否
NS	是	是	是	是	是	否	否	是	否	否	是	否
S	是	是	是	是	是	否	否	是	否	否	否	否
IX	是	是	是	否	否	是	否	否	否	否	否	否
SIX	是	是	是	否	否	否	否	否	否	否	否	否
U	是	是	是	是	是	否	否	否	否	否	否	否
X	是	是	否	否	否	否	否	否	否	否	否	否
Z	是	否	否	否	否	否	否	否	否	否	否	否
NW	是	是	否	是	否	否	否	否	否	否	否	是
W	是	是	否	否	否	否	否	否	否	否	是	否

表 12. 锁定类型兼容性 (续)

	占有资源的状态											
请求的状态	无	IN	IS	NS	S	IX	SIX	U	X	Z	NW	W

注:

I	意向
N	无
NS	下一键共享
S	共享
X	互斥
U	更新
Z	超互斥
NW	下一键弱互斥
W	弱互斥

注:

- 是 – 立即**准许**所请求的锁定
- 否 – **等待**释放挂起的锁定或出现超时

标准表的锁定方式和访问路径

本主题包含有关不同数据访问方案的标准表锁定方法的参考信息。

以下表列示了不同访问方案的每个级别的标准表获得的锁定类型。每个条目均由两部分组成: 表锁定和行锁定。虚线表示未进行某个特定级别的锁定。

注:

1. 在多维集群（MDC）环境中，还将使用附加的锁定级别 BLOCK。
2. 可以使用 SELECT 语句的锁定请求子句来显式更改锁定方式。

表 13. 不使用谓词的表扫描的锁定方式

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描	更新或删除
RR	S/-	U/-	SIX/X	X/-	X/-
RS	IS/NS	IX/U	IX/X	IX/X	IX/X
CS	IS/NS	IX/U	IX/X	IX/X	IX/X
UR	IN/-	IX/U	IX/X	IX/X	IX/X

表 14. 使用谓词的表扫描的锁定方式

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描	更新或删除
RR	S/-	U/-	SIX/X	U/-	SIX/X
RS	IS/NS	IX/U	IX/X	IX/U	IX/X
CS	IS/NS	IX/U	IX/X	IX/U	IX/X

表 14. 使用谓词的表扫描的锁定方式 (续)

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描	更新或删除
UR	IN/-	IX/U	IX/X	IX/U	IX/X

注: 在 UR 隔离级别，如果具有 1 类索引的 IN 锁定，或者如果索引的包含列中具有谓词，那么隔离级别将升级为 CS 并且锁定将升级为 IS 表锁定和 NS 行锁定。

表 15. 不使用谓词的 RID 索引扫描的锁定方式

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描	更新或删除
RR	S/-	IX/S	IX/X	X/-	X/-
RS	IS/NS	IX/U	IX/X	IX/X	IX/X
CS	IS/NS	IX/U	IX/X	IX/X	IX/X
UR	IN/-	IX/U	IX/X	IX/X	IX/X

表 16. 使用单个合格行的 RID 索引扫描的锁定方式

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描	更新或删除
RR	IS/S	IX/U	IX/X	IX/X	IX/X
RS	IS/NS	IX/U	IX/X	IX/X	IX/X
CS	IS/NS	IX/U	IX/X	IX/X	IX/X
UR	IN/-	IX/U	IX/X	IX/X	IX/X

表 17. 仅使用 Start 和 Stop 谓词的 RID 索引扫描的锁定方式

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描	更新或删除
RR	IS/S	IX/S	IX/X	IX/X	IX/X
RS	IS/NS	IX/U	IX/X	IX/X	IX/X
CS	IS/NS	IX/U	IX/X	IX/X	IX/X
UR	IN/-	IX/U	IX/X	IX/X	IX/X

表 18. 仅使用索引和其他谓词（sargs 和 resids）的 RID 索引扫描的锁定方式

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描	更新或删除
RR	IS/S	IX/S	IX/X	IX/S	IX/X
RS	IS/NS	IX/U	IX/X	IX/U	IX/X
CS	IS/NS	IX/U	IX/X	IX/U	IX/X
UR	IN/-	IX/U	IX/X	IX/U	IX/X

下面的表显示了在某些情况下的锁定方式，在这些情况下，系统将延迟读取数据页以允许将行列表:

- 使用多个索引进一步限定

- 排序以获得高效的预取

表 19. 延迟的数据页访问所用的索引扫描的锁定方式：不使用谓词的 RID 索引扫描

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描	更新或删除
RR	IS/S	IX/S		X/-	
RS	IN/-	IN/-		IN/-	
CS	IN/-	IN/-		IN/-	
UR	IN/-	IN/-		IN/-	

表 20. 延迟的数据页访问所用的索引扫描的锁定方式：在不使用谓词的 RID 索引扫描后

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描	更新或删除
RR	IN/-	IX/S	IX/X	X/-	X/-
RS	IS/NS	IX/U	IX/X	IX/X	IX/X
CS	IS/NS	IX/U	IX/X	IX/X	IX/X
UR	IN/-	IX/U	IX/X	IX/X	IX/X

表 21. 延迟的数据页访问所用的索引扫描的锁定方式：使用谓词（sargs 和 resids）的 RID 索引扫描

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描	更新或删除
RR	IS/S	IX/S		IX/S	
RS	IN/-	IN/-		IN/-	
CS	IN/-	IN/-		IN/-	
UR	IN/-	IN/-		IN/-	

表 22. 延迟的数据页访问所用的索引扫描的锁定方式：仅使用 Start 和 Stop 谓词的 RID 索引扫描

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描	更新或删除
RR	IS/S	IX/S		IX/X	
RS	IN/-	IN/-		IN/-	
CS	IN/-	IN/-		IN/-	
UR	IN/-	IN/-		IN/-	

表 23. 延迟的数据页访问所用的索引扫描的锁定方式，在仅使用 Start 和 Stop 谓词的 RID 索引扫描后

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描	更新或删除
RR	IN/-	IX/S	IX/X	IX/X	IX/X
RS	IS/NS	IX/U	IX/X	IX/U	IX/X
CS	IS/NS	IX/U	IX/X	IX/U	IX/X
UR	IS/-	IX/U	IX/X	IX/U	IX/X

表 24. 延迟的数据页访问所用的索引扫描的锁定方式，在使用谓词的 RID 索引扫描后

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描	更新或删除
RR	IN/-	IX/S	IX/X	IX/S	IX/X
RS	IS/NS	IX/U	IX/X	IX/U	IX/X
CS	IS/NS	IX/U	IX/X	IX/U	IX/X
UR	IN/-	IX/U	IX/X	IX/U	IX/X

表的锁定方式和 MDC 表的 RID 索引扫描

在多维集群（MDC）环境中，还将使用附加的锁定级别 BLOCK。以下表列示了不同访问方案在各个级别获得的锁定类型。每个条目由三部分组成：表锁定、块锁定和行锁定。虚线表示未使用某个特定级别的锁定。

注: 可以使用 SELECT 语句的锁定请求子句来显式更改锁定方式。

表 25. 不使用谓词的表扫描的锁定方式

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描或删除	更新
RR	S/-/-	U/-/-	SIX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/U	IX/X/-	IX/I/-
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-

表 26. 仅使用维列上的谓词的表扫描的锁定方式

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描或删除	更新
RR	S/-/-	U/-/-	SIX/IX/X	U/-/-	SIX/X/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/-	X/X/-
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/-	X/X/-
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/U/-	X/X/-

表 27. 使用其他谓词（sargs 和 resids）的表扫描的锁定方式

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描或删除	更新
RR	S/-/-	U/-/-	SIX/IX/X	U/-/-	SIX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

以下两个表显示了 MDC 表的 RID 索引的锁定方式。

表 28. 不使用谓词的 RID 索引扫描的锁定方式

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描删除	更新
RR	S/-/-	IX/IX/S	IX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	X/X/X	X/X/X

表 29. 使用单个合格行的 RID 索引扫描的锁定方式

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描删除	更新
RR	IS/IS/S	IX/IX/U	IX/IX/X	X/X/X	X/X/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/X	X/X/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	X/X/X	X/X/X

表 30. 仅使用 Start 和 Stop 谓词的 RID 索引扫描的锁定方式

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描删除	更新
RR	IS/IS/S	IX/IX/S	IX/IX/X	IX/IX/X	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X

表 31. 仅使用索引谓词的 RID 索引扫描的锁定方式

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描删除	更新
RR	IS/S/S	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

表 32. 使用其他谓词（sargs 和 resids）的 RID 索引扫描的锁定方式

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描删除	更新
RR	IS/S/S	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

注: 在下面的表中，显示了用于延迟的数据页访问的 RID 索引扫描的锁定方式，在 UR 隔离级别，如果具有 1 类索引的 IN 锁定，或者如果在索引的包含列中具有谓词，那么隔离级别将升级至 CS 并且锁定将升级至 IS 表锁定、IS 块锁定和 NS 行锁定。

表 33. 延迟的数据页访问使用的 RID 索引扫描的锁定方式: 不使用谓词的 RID 索引扫描

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描删除	更新
RR	IS/S/S	IX/IX/S		X/-/-	
RS	IN/IN/-	IN/IN/-		IN/IN/-	
CS	IN/IN/-	IN/IN/-		IN/IN/-	
UR	IN/IN/-	IN/IN/-		IN/IN/-	

表 34. 延迟的数据页访问使用的 RID 索引扫描的锁定方式: 在不使用谓词的 RID 索引扫描后延迟的数据页访问

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描删除	更新
RR	IN/IN/-	IX/IX/S	IX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/X	IX/IX/X

表 35. 延迟的数据页访问使用的 RID 索引扫描的锁定方式: 使用谓词（sargs 和 resids）的 RID 索引扫描

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描删除	更新
RR	IS/S/-	IX/IX/S		IX/IX/S	
RS	IN/IN/-	IN/IN/-		IN/IN/-	
CS	IN/IN/-	IN/IN/-		IN/IN/-	
UR	IN/IN/-	IN/IN/-		IN/IN/-	

表 36. 延迟的数据页访问使用的 RID 索引扫描的锁定方式: 在使用谓词（sargs 和 resids）的 RID 索引扫描后延迟的页数据访问

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描删除	更新
RR	IN/IN/-	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

表 37. 延迟的数据页访问使用的 RID 索引扫描的锁定方式: 仅使用 Start 和 Stop 谓词的 RID 索引扫描

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描删除	更新
RR	IS/IS/S	IX/IX/S		IX/IX/X	

表 37. 延迟的数据页访问使用的 RID 索引扫描的锁定方式: 仅使用 Start 和 Stop 谓词的 RID 索引扫描 (续)

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描删除	更新
RS	IN/IN/-	IN/IN/-		IN/IN/-	
CS	IN/IN/-	IN/IN/-		IN/IN/-	
UR	IN/IN/-	IN/IN/-		IN/IN/-	

表 38. 延迟的数据页访问使用的 RID 索引扫描的锁定方式: 在仅使用 Start 和 Stop 谓词的 RID 索引扫描后延迟的页数据访问

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描删除	更新
RR	IN/IN/-	IX/IX/S	IX/IX/X	IX/IX/X	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IS/-/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

MDC 表的块索引扫描的锁定

以下表列示了不同访问方案在各个级别获得的锁定类型。每个条目由三部分组成: 表锁定、块锁定和行锁定。虚线表示未进行某个特定级别的锁定。

注: 可以使用 SELECT 语句的锁定请求子句来显式更改锁定方式。

表 39. 不使用谓词的索引扫描的锁定方式

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描删除	更新
RR	S/--/--	IX/IX/S	IX/IX/X	X/--/--	X/--/--
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--
UR	IN/IN/-	IX/IX/U	IX/IX/X	X/X/--	X/X/--

表 40. 仅使用维谓词的索引扫描的锁定方式

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描删除	更新
RR	IS/-/-	IX/IX/S	IX/IX/X	X/-/-	X/-/-
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/X/-	IX/X/-

表 41. 仅使用 Start 和 Stop 谓词的索引扫描的锁定方式

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描删除	更新
RR	IS/S/-	IX/IX/S	IX/IX/S	IX/IX/S	IX/IX/S

表 41. 仅使用 Start 和 Stop 谓词的索引扫描的锁定方式 (续)

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描删除	更新
RS	IX/IX/S	IX/IX/U	IX/IX/X	IX/IX/-	IX/IX/-
CS	IX/IX/S	IX/IX/U	IX/IX/X	IX/IX/-	IX/IX/-
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/-	IX/IX/-

表 42. 使用谓词的索引扫描的锁定方式

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描删除	更新
RR	IS/S/-	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/-	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

下表列示延迟的数据页访问所使用的块索引扫描的锁定方式:

表 43. 延迟的数据页访问使用的块索引扫描的锁定方式: 不使用谓词的块索引扫描

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描删除	更新
RR	IS/S/--	IX/IX/S		X/--/--	
RS	IN/IN/--	IN/IN/--		IN/IN/--	
CS	IN/IN/--	IN/IN/--		IN/IN/--	
UR	IN/IN/--	IN/IN/--		IN/IN/--	

表 44. 延迟的数据页访问使用的块索引扫描的锁定方式: 在不使用谓词的块索引扫描后延迟的数据页访问

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描删除	更新
RR	IN/IN/--	IX/IX/S	IX/IX/X	X/--/--	X/--/--
RS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--
CS	IS/IS/NS	IX/IX/U	IX/IX/X	X/X/--	X/X/--
UR	IN/IN/--	IX/IX/U	IX/IX/X	X/X/--	X/X/--

表 45. 延迟的数据页访问使用的块索引扫描的锁定方式: 仅使用维谓词的块索引扫描

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描删除	更新
RR	IS/S/--	IX/IX/--		IX/S/--	
RS	IS/IS/NS	IX/--/--		IX/--/--	
CS	IS/IS/NS	IX/--/--		IX/--/--	
UR	IN/IN/--	IX/--/--		IX/--/--	

表 46. 延迟的数据页访问使用的块索引扫描的锁定方式：在仅使用维谓词的块索引扫描后延迟的数据页访问

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描删除	更新
RR	IN/IN/--	IX/IX/S	IX/IX/X	IX/S/--	IX/X/--
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/--	IX/X/--
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/U/--	IX/X/--
UR	IN/IN/--	IX/IX/U	IX/IX/X	IX/U/--	IX/X/--

表 47. 延迟的数据页访问使用的块索引扫描的锁定方式：仅使用 Start 和 Stop 谓词的索引扫描

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描删除	更新
RR	IS/S/--	IX/IX/--		IX/X/--	
RS	IN/IN/--	IN/IN/--		IN/IN/--	
CS	IN/IN/--	IN/IN/--		IN/IN/--	
UR	IN/IN/--	IN/IN/--		IN/IN/--	

表 48. 延迟的数据页访问使用的块索引扫描的锁定方式：在仅使用 Start 和 Stop 谓词的块索引扫描后延迟的数据页访问

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描删除	更新
RR	IN/IN/--	IX/IX/X		IX/X/--	
RS	IS/IS/NS	IN/IN/--		IN/IN/--	
CS	IS/IS/NS	IN/IN/--		IN/IN/--	
UR	IS/--/--	IN/IN/--		IN/IN/--	

表 49. 延迟的数据页访问使用的块索引扫描的锁定方式：使用其他谓词（sargs 和 resids）的块索引扫描

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描删除	更新
RR	IS/S/--	IX/IX/--		IX/IX/--	
RS	IN/IN/--	IN/IN/--		IN/IN/--	
CS	IN/IN/--	IN/IN/--		IN/IN/--	
UR	IN/IN/--	IN/IN/--		IN/IN/--	

表 50. 延迟的数据页访问使用的块索引扫描的锁定方式：在使用其他谓词（sargs 和 resids）的块索引扫描后延迟的数据页访问

隔离级别	只读和模糊扫描	游标操作		搜索的更新或删除	
		扫描	当前位置	扫描删除	更新
RR	IN/IN/--	IX/IX/S	IX/IX/X	IX/IX/S	IX/IX/X
RS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
CS	IS/IS/NS	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X
UR	IN/IN/--	IX/IX/U	IX/IX/X	IX/IX/U	IX/IX/X

对分区表的锁定行为

除整个表锁定之外，还有一个对分区表的每个数据分区的锁定。与非分区表相比，这允许更高的详细程度并提高了并行性。在 db2pd 命令、事件监视器、管理视图和表函数的输出中标识新数据分区锁定。

访问表时，锁定行为按访问的数据所指示的那样，首先获取表锁定，然后获取数据分区锁定。访问方法和隔离级别可能获取不在结果集中的数据分区的锁定。获取这些数据分区锁定之后，在表锁定期间可能会一直挂起这些锁定。例如，对索引的游标稳定性（CS）扫描可能保持对先前访问的数据分区的锁定，以便在后续键中引用该数据分区的情况下，减少重新获取该数据分区锁定的成本。数据分区锁定还承担用于确保对表空间的访问的成本。对于非分区表，表空间访问由表锁定处理。因此，对于分区表，即使在表级别存在互斥或共享锁定，也会进行数据分区锁定。

更高的详细程度允许一个事务具有对给定数据分区的互斥访问权并避免行锁定，而同时其他事务能够访问其他数据分区。这可能是由于为大量更新选择的方案或将锁定升级到数据分区级别而产生的。许多访问方法的表锁定通常是意向锁定，即使是以共享或互斥方式锁定数据分区。这将提高并行性。但是，如果在数据分区级别需要非意向锁定并且方案表明可能会访问所有数据分区，那么可能会在表级别选择非意向锁定，以防止并发事务的数据分区锁定之间出现死锁。

对 SQL LOCK TABLE 语句的锁定

对于分区表，在表级别获取对 LOCK TABLE 语句的唯一锁定；不获取数据分区锁定。这将确保没有对后续 DML 语句中的表的行锁定，并避免在行、块或数据分区级别出现死锁。在更新索引时使用 LOCK TABLE IN EXCLUSIVE MODE 还可用于保证互斥访问，这对于在大量更新期间限制 2 类索引的增长很有用。

ALTER TABLE 语句的 LOCKSIZE TABLE 参数的影响

ALTER TABLE 语句有一个用于设置 LOCKSIZE TABLE 的选项，这确保不使用意向锁定以共享或互斥方式锁定表。对于分区表来说，这种锁定策略既适用于表锁定，也适用于对访问的任何数据分区的数据分区锁定。

行锁定和块锁定升级

对于分区表，行锁定和块锁定的升级单元是升级到数据分区级别。这再次表明与其他事务相比，更容易访问表，即使数据分区升级到共享、互斥或超级互斥，并使其他未升级的数据分区不受影响。对给定数据分区进行升级后，事务可能继续对其他数据分区进行行锁定。升级的通知日志消息包括已升级的数据分区和分区表的名称。因此，锁定升级不能确保对索引的互斥访问。语句必须使用互斥表级别锁定、必须发出显式 LOCK TABLE IN EXCLUSIVE MODE 语句，或表必须使用 LOCKSIZE TABLE 属性。由优化器根据数据分区消除选择访问方法的整个表锁定。如果没有发生数据分区消除，那么要大量更新表时可以选择互斥表锁定。

解释锁定信息

SNAPLOCK 管理视图中的以下示例可以帮助您解释对分区表返回的锁定信息。

示例 1:

此 SNAPLOCK 管理视图是在脱机索引重组期间捕获的。

```
SELECT SUBSTR(TABNAME, 1, 15) TABNAME, TAB_FILE_ID, SUBSTR(TBSP_NAME, 1, 15) TBSP_NAME, DATA_PARTITION_ID, LOCK_OBJECT_TYPE,
LOCK_MODE, LOCK_ESCALATION FROM SYSIBMADM.SNAPLOCK where TABNAME like 'TP1' and LOCK_OBJECT_TYPE like 'TABLE_%'
ORDER BY TABNAME, DATA_PARTITION_ID, LOCK_OBJECT_TYPE, TAB_FILE_ID, LOCK_MODE

TABNAME         TAB_FILE_ID          TBSP_NAME       DATA_PARTITION_ID LOCK_OBJECT_TYPE   LOCK_MODE  LOCK_ESCALATION
--------------- -------------------- --------------- ----------------- ------------------ ---------- ---------------
TP1                           32768 -                              -1 TABLE_LOCK         Z                       0
TP1                               4 USERSPACE1                     0 TABLE_PART_LOCK    Z                       0
TP1                               5 USERSPACE1                     1 TABLE_PART_LOCK    Z                       0
TP1                               6 USERSPACE1                     2 TABLE_PART_LOCK    Z                       0
TP1                               7 USERSPACE1                     3 TABLE_PART_LOCK    Z                       0
TP1                               8 USERSPACE1                     4 TABLE_PART_LOCK    Z                       0
TP1                               9 USERSPACE1                     5 TABLE_PART_LOCK    Z                       0
TP1                              10 USERSPACE1                     6 TABLE_PART_LOCK    Z                       0
TP1                              11 USERSPACE1                     7 TABLE_PART_LOCK    Z                       0
TP1                              12 USERSPACE1                     8 TABLE_PART_LOCK    Z                       0
TP1                              13 USERSPACE1                     9 TABLE_PART_LOCK    Z                       0
TP1                              14 USERSPACE1                    10 TABLE_PART_LOCK    Z                       0
TP1                              15 USERSPACE1                    11 TABLE_PART_LOCK    Z                       0
TP1                               4 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                               5 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                               6 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                               7 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                               8 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                               9 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                              10 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                              11 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                              12 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                              13 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                              14 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                              15 USERSPACE1                     - TABLE_LOCK         Z                       0
TP1                              16 USERSPACE1                     - TABLE_LOCK         Z                       0

  选择了 26 个记录。
```

在此示例中，类型为 TABLE_LOCK 且 DATA_PARTITION_ID 为 -1 的锁定对象用于控制对分区表 TP1 的访问和并行性。类型为 TABLE_PART_LOCK 的锁定对象用于控制对每个数据分区的大多数访问和并行性。

此输出中还捕获了其他类型为 TABLE_LOCK 的锁定对象（TAB_FILE_ID 4 到 16），这些锁定对象没有 DATA_PARTITION_ID 值。如果某种类型的锁定中对象的 TAB_FILE_ID 和 TBSP_NAME 与分区表中的数据分区或索引相对应，那么可以使用这种类型的锁定来控制脱机备份实用程序的并行性。

影响锁定的因素

下列因素影响数据库管理器锁定的方式和详细程度:

- 应用程序执行的处理类型
- 数据访问方法
- 索引是 2 类索引还是 1 类索引
- 各种配置参数

应用程序处理的锁定和类型

为了确定锁定属性，可以将应用程序处理划分为下列类型的其中一种:

- 只读

  此类型包括所有这样的选择语句，它们本身是只读的，并具有一个显式的 FOR READ ONLY 子句，或者是模糊的（但是查询编译器因为 PREP 或 BIND 命令指定了 BLOCKING 选项的值而假定是只读的）。此处理类型只要求“共享”锁定（S、NS 或 IS）。

- 更改意向

  此类型包括具有 FOR UPDATE 子句、具有 USE AND KEEP UPDATE LOCKS 子句、具有 USE AND KEEP EXCLUSIVE LOCKS 子句的所有 SELECT 语句，或者查询编译器解释有歧义的语句以表示想进行更改。此类型使用“共享”和“更新”锁定（对于行，为 S、U 及 X；对于块，为 IX、U、X 和 S；对于表，为 IX、U 和 X）。
- 更改

  此类型包括 UPDATE、INSERT 及 DELETE，但不包括 UPDATE WHERE CURRENT OF 或 DELETE WHERE CURRENT OF。此类型要求“互斥”锁定（X 或 IX）。
- 受控游标

  此类型包括 UPDATE WHERE CURRENT OF 和 DELETE WHERE CURRENT OF。它也要求“互斥”锁定（X 或 IX）。

根据子查询语句的结果在目标表中进行插入、更新或删除数据的语句执行两种类型的处理。只读处理规则确定对在子选择语句中返回的表的锁定。更改处理规则确定对目标表的锁定。

锁定和数据访问方法

*访问方案*是优化器选择以从特定表中检索数据的方法。访问方案可对锁定方式具有很大影响。例如，当使用索引扫描来查找特定行时，优化器将可能为该表选择行级别锁定（IS）。例如，如果 EMPLOYEE 表有一个关于职员编号（EMPNO）的索引，通过索引访问可以用来选择单个职员的信息，使用包含下列 SELECT 子句的语句:

```
SELECT *
   FROM EMPLOYEE       WHERE EMPNO = '000310';
```

如果不使用索引，那么必须按顺序扫描整个表来找到所选的行，并且可因此获取单个表级锁定（S）。例如，如果 SEX 列上没有索引，那么表扫描可以用来选择所有的男性职员，使用包含下列 SELECT 子句的语句:

```
SELECT *
   FROM EMPLOYEE       WHERE SEX = 'M';
```

注: 受控游标处理使用基础游标锁定方式，直到应用程序找到要更新或要删除的行为止。对于此种类型的处理，无论游标的锁定方式是什么，总会获取互斥锁定以执行更新或删除。

范围集群表中的锁定与标准键或下一个键锁定的工作方式稍微有所不同。当访问范围集群表中的一定范围的行时，该范围内的所有行都会被锁定，即使某些行是空的也会被锁定。在标准键或下一个键锁定中，将只锁定具有现有记录的行。

引用表提供了有关哪一类访问方案获取哪种锁定的详细信息。

被延迟的数据页访问意味着对行的访问分两步进行，这导致更复杂的锁定情况。锁定获得的计时和锁定的持久性取决于隔离级别。因为“可重复读”隔离级别保留所有锁定直到事务结束，所以会保持在第一步中所获取的锁定而不必在第二步获取进一步的锁定。对于“读稳定性”和“游标稳定性”隔离级别，必须在第二步期间获取锁定。要使并行性最大化，在第一步期间不获取锁定并依靠重新应用所有谓词来确保只返回限定行。

索引类型和下一键锁定

当事务引起 1 类索引的更改时，会发生某些下一键锁定。对于 2 类索引，发生最小的下一键锁定。

- 对于 2 类索引的下一键锁定

  当将某个键插入索引时发生下一键锁定。

  在某个键插入索引期间，仅当与将跟在该索引中新键后面的键对应的行当前由 RR 索引扫描锁定时，才锁定该行。用于下一键锁定的锁定方式是 NW。在实际执行键插入之前，释放该下一键锁定。当将行插入表时，发生键插入。

  当更新某一行导致对该行的索引键的值的更改时，键插入也因为将原始键值标记为已删除而发生且将新的键值插入索引中。对于仅影响索引的包含列的更新，可适当更新键而不发生下一键锁定。

  在 RR 扫描期间，以 S 方式锁定与跟在扫描范围末尾后面的键对应的行。如果没有键跟在扫描范围末尾的后面，那么获取表结束锁定以锁定索引的末尾。如果将跟在扫描范围末尾后面的键标记为已删除，那么扫描继续锁定对应的行，直到它找到不标记为已删除的键，当它锁定该键的对应行时，或直到锁定索引的末尾为止。

- 对于 1 类索引的下一键锁定:

  在对索引的删除和插入期间和在索引扫描期间，发生下一键锁定。当更新表中的行、从表中删除行或将行插入表中时，获取对该行的 X 锁定。对于插入，这可能降级为 W 锁定。

  当从表索引删除键或将键插入其中时，锁定与索引中跟在已删除或插入的键后面的键对应的表行。对于影响键值的更新，首先删除原始的键值并插入新的值，以便获取两个下一键锁定。如下所示确定这些锁定的持续时间:

  – 在索引键删除期间，对下一键的锁定方式是 X 且挂起该锁定，直到落实时为止。
  – 在索引键插入期间，对下一键的锁定方式是 NW。仅当存在锁定的争用时，才获取此锁定，在此情况下，在将键实际插入索引之前释放该锁定。
  – 在 RR 扫描期间，以 S 方式锁定与正好超出索引扫描范围末尾的键对应的表行并在落实时间之前挂起。
  – 在 CS/RS 扫描期间，如果存在锁定的争用，那么以 NS 方式锁定与正好超出索引扫描范围末尾的键对应的行。一旦验证了扫描范围的结束，那么释放此锁定。

  键插入和键删除期间对于 1 类索引的下一键锁定可以导致死锁。以下示例显示两个事务是如何死锁的。对于 2 类索引，这种死锁不会发生。

  考虑索引的下列示例，该索引包含 6 行，并分别具有下列值: 1 5 6 7 8 12。

  1. 事务 1 删除具有键值 8 的行。以 X 方式锁定具有值 8 的行。当删除索引中对应的键时，以 X 方式锁定具有值 12 的行。
  2. 事务 2 删除具有键值 5 的行。以 X 方式锁定具有值 5 的行。当删除索引中对应的键时，以 X 方式锁定具有值 6 的行。
  3. 事务 1 插入具有键值 4 的行。以 W 方式锁定此行。当尝试将新的键插入索引时，以 NW 方式锁定具有值 6 的行。此锁定尝试将在事务 2 在此行具有的 X 锁定上等待。

4. 事务 2 插入具有键值 9 的行。以 W 方式锁定此行。当尝试将新的键插入索引时，以 NW 方式锁定具有键值 12 的行。此锁定尝试将在事务 1 在此行具有的 X 锁定上等待。

当使用 1 类索引时，此方案将导致死锁且将这些事务的其中一项事务回滚。

指定锁定等待方式策略

单个会话现在可以指定锁定等待方式策略，该策略在会话需要不能立即获取的锁定时使用。该策略指示会话是否将:

- 在不能获取锁定时返回 SQLCODE 和 SQLSTATE
- 无限等待锁定
- 为获取锁定等待一段指定的时间
- 等待锁定时使用 *locktimeout* 数据库配置参数的值

锁定等待方式策略通过新的 SET CURRENT LOCK TIMEOUT 语句指定，此语句更改 CURRENT LOCK TIMEOUT 专用寄存器的值。CURRENT LOCK TIMEOUT 专用寄存器指定在返回指示不能获取锁定的错误之前等待锁定的秒数。

传统的锁定方法会导致应用程序互相阻塞。当一个应用程序必须等待另一个应用程序释放其锁定时，阻塞就会发生。用于处理这种阻塞的影响的策略通常会提供一种机制以指定最大可接受阻塞持续时间。这就是应用程序在不能获取锁定的情况下在返回之前等待的时间。以前，只能在数据库级别通过更改 *locktimeout* 数据库配置参数的值来指定时间。

虽然 *locktimeout* 参数的值适用于所有锁定，但是此新功能只影响以下锁定类型: 行、表、索引键和多维集群（MDC）块锁定。

调整应用程序

限制选择语句的准则

优化器假定应用程序必须检索由 SELECT 语句指定的所有行。此假设最适合于 OLTP 和批处理环境。但是，在“浏览”应用程序中，查询经常定义一个可能很大的答案集，但它们只检索前几行，通常只检索填满该屏幕所需的那么多行。

要提高这种应用程序的性能，可以按下列方式修改 SELECT 语句:

- 使用 FOR UPDATE 子句来指定可由后续定位的 UPDATE 语句更新的列。
- 使用 FOR READ/FETCH ONLY 子句来使返回的列为只读的。
- 使用 OPTIMIZE FOR *n* ROWS 子句来给予检索整个结果集中前 *n* 行的优先级。
- 使用 FETCH FIRST *n* ROWS ONLY 子句来仅检索指定的几行。
- 使用 DECLARE CURSOR WITH HOLD 语句来每次检索一行。

注: 如果使用 FOR UPDATE、FETCH FIRST *n* ROWS ONLY 或 OPTIMIZE FOR *n* ROWS 子句，或者如果您声明游标为“滚动”，那么会影响行分块。

以下节描述每个方法的性能优点。

FOR UPDATE 子句

FOR UPDATE 子句通过仅包含可由后续定位的 UPDATE 语句更新的列来限制结果集。如果不带列名指定 FOR UPDATE 子句，那么包括表或视图的全部可更新列。如果指定列名，那么每个名称必须是非限定的，且必须标识表或视图的某一列。

在下列情况下，不能使用 FOR UPDATE 子句:

- 如果不能删除与 SELECT 语句关联的游标。
- 如果选择的列中至少有一列是目录表的不可更新列，且没有在 FOR UPDATE 子句中排除掉。

由于同样的目的，在 CLI 应用程序中使用 DB2 CLI 连接属性 SQL_ATTR_ACCESS_MODE。

FOR READ 或 FETCH ONLY 子句

FOR READ ONLY 子句或 FOR FETCH ONLY 子句确保返回只读结果。因为被定义为只读的视图上 SELECT 的结果表也是只读的，所以允许此子句但没有作用。

对于允许更新和删除的结果表，如果数据库管理器可以检索数据块而不是互斥锁定，那么指定 FOR READ ONLY 可能会提高 FETCH 操作的性能。不要对用于定位的 UPDATE 或 DELETE 语句的查询使用 FOR READ ONLY 子句。

由于相同的目的，可将 DB2 CLI 连接属性 SQL_ATTR_ACCESS_MODE 用于 CLI 应用程序。

OPTIMIZE FOR n ROWS 子句

OPTIMIZE FOR 子句声明它只想检索结果的一个子集或优先检索前几行。优化器然后可以优先选择把检索前几行的响应时间减至最短的访问方案。另外，作为单个块发送到客户机的行数由 OPTIMIZE FOR 子句中的“n”值来限制。因此，OPTIMIZE FOR 子句既影响服务器从数据库检索合格行的方式，又影响将合格行返回到客户机的方式。

例如，假设您定期查询职员表，来查找具有最高工资的职员。

```
SELECT LASTNAME,FIRSTNAME,EMPNO,SALARY
     FROM EMPLOYEE          ORDER BY SALARY DESC
```

您定义了一个基于 SALARY 列的降序索引。但是，由于职员是按职员号排序的，所以工资索引可能很难集群。为了尽量避免许多随机的同步 I/O，优化器将可能选择使用列表预取访问方法，此方法需要对合格的所有行的行标识排序。在将前几个合格行返回至应用程序前，此排序可能导致一个延迟。要防止此延迟，向语句添加 OPTIMIZE FOR 子句，如下所示:

```
SELECT LASTNAME,FIRSTNAME,EMPNO,SALARY
     FROM EMPLOYEE          ORDER BY SALARY DESC          OPTIMIZE FOR 20 ROWS
```

在此情况下，优化器可能选择直接使用 SALARY 索引，因为只检索具有最高工资的二十个职员。不管可以将多少行分块，将只把每二十行组成的一个块返回至客户机。

使用 OPTIMIZE FOR 子句，优化器优先选择可以避免大量操作或中断行流动（如排序）的访问方案。使用 OPTIMIZE FOR 1 ROW 最有可能影响访问路径。使用此子句可以有下列作用:

- 与组合内部表的连接顺序不太可能，因为它们需要一个临时表。

- 连接方法可以更改。一个嵌套循环连接是非常可能的选择，因为它具有低开销成本，并且通常在检索少量行时更有效率。
- 一个与 ORDER BY 子句匹配的索引更可能，因为对于 ORDER BY 不需要排序。
- 列表预取不太可能，因为此访问方法需要排序。
- 顺序预取不太可能，因为知道只需要少量的几行。
- 在一个连接查询中，在 ORDER BY 子句中包含列的表可能选作外部表，前提是该外部表上的一个索引提供 ORDER BY 子句所需的排序。

虽然 OPTIMIZE FOR 子句适用于所有优化级别，但它在优化级别 3 和更高级别下工作得最好，因为级别 3 以下的级别使用“贪婪”连接枚举方法。此方法有时会产生一个不能使它们自己很快检索前几行的多表连接的访问方案。

OPTIMIZE FOR 子句不阻止您检索全部合格行。如果确实要检索全部合格行，那么总耗用时间可能大大高于如果优化器为整个答案集进行优化所需的时间。

如果已打包的应用程序使用调用级接口（DB2 CLI 或 ODBC），可在 db2cli.ini 配置文件中使用 OPTIMIZEFORNROWS 关键字，让 DB2 CLI 自动将一个 OPTIMIZE FOR 子句追加至每个查询语句的末尾。

当从昵称选择数据时，结果可能随数据源支持的不同而变化。如果该昵称引用的数据源支持 OPTIMIZE FOR 子句且 DB2 优化器将整个查询下推至数据源，那么将在发送到数据源的远程 SQL 中生成该子句。如果数据源不支持此子句，或者如果优化器决定在本地执行最低成本方案，那么在本地应用 OPTIMIZE FOR 子句。在这种情况下，DB2 优化器优先选择把检索某查询前几行的响应时间最小化的访问方案，但可供优化器生成方案的选项略微受到限制，因此从 OPTIMIZE FOR 子句获得的性能改善可能非常小。

如果同时指定了 FETCH FIRST 子句和 OPTIMIZE FOR 子句，那么这两个值中较小的一个影响通信缓冲区大小。为了达到最优化，将这两个值看作是互不相关的。

FETCH FIRST n ROWS ONLY 子句

FETCH FIRST *n* ROWS ONLY 子句设置可检索的最大行数。将结果表限制为只包含前几行可提高性能。无论结果集可能另外包含多少行，只检索 *n* 行。

如果同时指定了 FETCH FIRST 子句和 OPTIMIZE FOR 子句，那么这两个值中较小的一个影响通信缓冲区大小。为了达到优化目的，这两个值是互相独立的。

DECLARE CURSOR WITH HOLD 语句

当用包括 WITH HOLD 子句的 DECLARE CURSOR 语句声明游标时，在落实该事务时任何打开的游标仍然打开；并且释放所有锁定，除保护打开的 WITH HOLD 游标的当前游标位置的锁定外。

如果回滚事务，那么关闭所有打开的游标且释放所有锁定以及释放 LOB 定位器。

可将 DB2 CLI 连接属性 SQL_ATTR_CURSOR_HOLD 用于 CLI 应用程序以实现同样的结果。如果有使用调用级接口（DB2 CLI 或 ODBC）的已打包的应用程序，那么在 db2cli.ini 配置文件中使用 CURSORHOLD 关键字，让 DB2 CLI 自动为每个已声明游标假设 WITH HOLD 子句。

指定行分块来减少开销

支持对所有语句和数据类型（包括 LOB 数据类型）进行行分块。行分块通过在单个操作中检索行块来减少游标的数据库管理器开销。

注: 指定的行块是内存中的许多页。它不是一个多维（MDC）表块，该块物理上映射为磁盘上的一个扩展数据块。

行分块由 BIND 或 PREP 命令的下列自变量指定:

UNAMBIG
对使用 FOR READ ONLY 子句指定的游标进行分块。

不是使用 FOR READ ONLY 或 FOR UPDATE 子句声明的游标将被阻塞，这些游标不是*模糊*游标，也不是*只读*游标。*模糊*游标不会被阻塞。

ALL 对使用 FOR READ ONLY 子句指定的游标或者未指定为 FOR UPDATE 的游标进行分块。

NO 不对任何游标进行分块。

有关只读游标和模糊游标的定义，请参阅 DECLARE CURSOR 语句。

注: 如果在 SELECT 语句中使用 FETCH FIRST *n* ROWS ONLY 子句或 OPTIMIZE FOR *n* ROWS 子句，那么每块的行数将是下列各项中的最小值:

- 上述公式中计算的值
- FETCH FIRST 子句中 *n* 的值
- OPTIMIZE FOR 子句中 *n* 的值

必须适当设置两个数据库管理器配置参数。这两个值都设置为内存的若干页。注意这些参数在块大小计算中使用的值。

- 数据库管理器配置参数 *aslheapsz* 指定本地应用程序的应用程序支持层堆大小。
- 数据库管理器配置参数 *rqrioblk* 指定数据库服务器上远程应用程序及其数据库代理程序之间的通信缓冲区大小。

在对 LOB 数据类型的行数据启用分块之前，一定要了解这样做对系统资源产生的影响。当返回 LOB 列时，服务器上将消耗更多共享内存来存储对每个数据块中的 LOB 值的引用。这种引用的数目将随数据库配置参数 *rqrioblk* 的值不同而不同。

要增加分配给堆的内存量，通过以下方法来修改数据库配置参数 *database_memory*:

- 将此参数设置为 AUTOMATIC 会告知数据库管理器自动地管理数据库内存。
- 如果此参数当前设置为用户定义的数值，那么将该值增大 256 页。

要提高引用了 LOB 值的现有嵌入式 SQL 应用程序的性能，可以使用 BIND 命令来重新绑定该应用程序并且指定 BLOCKING ALL 子句或 BLOCKING UNAMBIGUOUS 子句来请求分块。嵌入式应用程序将从服务器中检索 LOB 值，一次检索一行，直到从服务器中检索了一系列行为止。返回大型 LOB 结果的 UDF 可能会导致 DB2 还原为检索单行 LOB 数据，这种情况下服务器上将消耗大量内存。

要指定行分块:

1. 使用 *aslheapsz* 和 *rqrioblk* 配置参数的值来估计为每个块返回多少行。在两个公式中，*orl* 是以字节为单位的输出行长度。

- 对于本地应用程序，使用下列公式:

  ```
  每块行数 = aslheapsz * 4096 / orl
  ```

  每页的字节数为 4096。
- 对于远程应用程序，使用下列公式:

  ```
  每块行数 = rqrioblk / orl
  ```

2. 要启用行分块，对 PREP 或 BIND 命令中的 BLOCKING 选项指定适当的自变量。

   如果不指定 BLOCKING 选项，那么缺省行分块类型为 UNAMBIG。对于命令行处理器和调用级接口，缺省行分块类型为 ALL。

查询调整准则

遵循查询调整准则来细调应用程序中的 SQL 和 XQuery 语句。准则将帮助您最小化系统资源的使用以及从大表和复杂查询返回结果所需的时间。

注: 优化器使用的优化级别可以消除某些细调要求，因此查询编译器可以将 SQL 和 XQuery 代码重新编写为更有效的格式。

注意，优化器选择的访问方案也受其他因素的影响，这些因素包括环境注意事项和系统目录统计信息。如果您对应用程序的性能进行基准程序测试，可发现哪些调整可以改善访问方案。

使用 REOPT 绑定选项进行查询优化

要对具有主变量、专用寄存器、全局变量或参数标记的静态和动态 SQL 和 XQuery 语句启用查询优化（或重新优化），应将程序包与 REOPT 绑定选项进行绑定。当使用此选项时，将使用主变量、参数标记、全局变量或专用寄存器的值而不是编译器选择的缺省估计值来优化属于该程序包并且包含这些变量的 SQL 和 XQuery 语句的访问路径。当值可用时，执行查询时就会进行此优化。

通过与 REOPT 绑定来提高性能

如果用于输入变量（例如，参数标记、主变量、全局变量和专用寄存器）的值超出了缺省过滤因子估计的预期范围，那么在执行期间 SQL 或 XQuery 查询的执行情况可能很差。用于不知道实际数据值的一些方案的缺省过滤因子会估计以下情况: 使用实际数据值时，在运行时实际上将返回多少行。

REOPT 绑定选项指定是否让 DB2 通过使用主变量、参数标记、全局变量和专用寄存器的值来优化访问路径。REOPT 值分别由 BIND、PREP 和 REBIND 命令的下列自变量指定:

REOPT NONE

将不使用主变量、参数标记、全局变量或专用寄存器的实际值来对包含这些变量的给定 SQL 或 XQuery 语句的访问路径进行优化；而是使用这些变量的缺省估计值。对此方案进行了高速缓存，并且将来会使用此方案。这是缺省行为。

REOPT ONCE

在第一次执行查询时，将使用主变量、参数标记、全局变量或专用寄存器的实际值来优化给定 SQL 或 XQuery 语句的访问路径。对此方案进行了高速缓存，并且将来会使用此方案。

REOPT ALWAYS

将始终使用每次执行时已知道的主变量、参数标记、全局变量或专用寄存器的值来编译和重新优化给定 SQL 或 XQuery 语句的访问路径。

SQL 和 XQuery 查询中的数据采样

数据库增长得非常庞大，并且对这些数据库的查询非常复杂，使得访问与查询相关的所有数据变得很不实际，有时也没有必要。在某些情况下，用户对查找总体趋势或模式感兴趣，在这种情况下，允许得到错误在一定范围内的大致答案。加速这种查询的一种方法对数据库的随机样本执行查询。DB2 允许您在 SQL 和 XQuery 查询中执行有效采样，从而在保留高精确度的同时潜在提高根据量值顺序进行大型查询的性能。

最常见的采样应用程序用于聚集查询（例如，AVG、SUM 和 COUNT），其中可从数据样本中获取聚集的合理精确结果。还可以使用采样来获取表中实际行的随机子集以用于审计，或者用于提高数据挖掘和分析任务的速度。

DB2 提供两种采样方法：行级别采样和块级别采样。

行级别 Bernoulli 采样

行级别 Bernoulli 采样通过使用控制谓词（它包含样本中的每一行的概率为 P/100，排除样本中的每一行的概率为 1 - P/100）来获取表行的 P% 的样本。

行级别 Bernoulli 采样总是获取有效的随机样本，不管数据集群如何。但是，如果没有提供索引，这种采样的性能会很低，这是因为必须检索每行且对其应用采样谓词。如果没有任何索引，那么在执行不采样的查询时 I/O 消耗会很大。如果有索引可用，那么会提高使用这种采样的性能，这是因为对索引叶子页内的 RIDS 应用了采样谓词。一般情况下，这要求每个所选 RID 一个 I/O，每个索引叶子页一个 I/O。

系统页级别采样

系统页级别采样与行级别采样相似，但是它是对页采样而不是对行采样。将一页包含在样本中的概率为 P/100。如果包含了某页，就包含了该页中的所有行。

系统页级别采样的性能很好，因为对于包含在样本中的每一页只需要执行一次 I/O。与不采样相比，页级别采样根据量值顺序提高性能。但是，页级别采样与行级别采样比起来，聚集估计的准确性可能更差。当每个块有许多行时，或者当查询中引用的列在页中显示较高的集群度时，就很可能产生这种准确性的差异。

特定任务的最佳采样方法将由用户的时间约束和期望的准确度来确定。

指定采样方法

要对表中的数据的随机样本执行查询，可以在 SQL 语句中使用 table-reference 子句的 TABLESAMPLE 子句。要指定采样的方法，使用关键字 BERNOULLI 或 SYSTEM。

BERNOULLI 关键字指定执行行级别 Bernoulli 采样。

SYSTEM 关键字指定执行系统页级别采样，除非优化器确定执行行级别 Bernoulli 采样反而更有效。

应用程序的并行处理

DB2 主要在对称多处理器（SMP）机器上支持并行环境，但在单处理器机器上也有限程度地支持此环境。在 SMP 机器中，多个处理器可以访问数据库，这使得将复杂 SQL 请求的并行执行分到各个处理器上。

要指定编译应用程序时要实现的并行度，可使用 CURRENT DEGREE 专用寄存器或 DEGREE 绑定选项。度指同时执行的某个查询的部分数。在处理器数目和为并行度选择的值之间没有严格的关系。可以指定比机器上处理器数目多或少的数。甚至对于单处理器机器，您可以设置比以某些方式提高性能的并行度高的并行度。但是，注意，每个并行度都增加系统内存和 CPU 开销。

当使用查询的并行执行时，必须修改某些配置参数来优化性能。特别对于具有高并行度的环境，应该复查和修改控制共享内存量和预取的配置参数。

下列三个配置参数控制和管理分区内并行性。

- *intra_parallel* 数据库管理器配置参数启用或禁用并行性支持。
- *max_querydegree* 数据库配置参数设置数据库中任何查询的并行度上限。此值覆盖 CURRENT DEGREE 专用寄存器和 DEGREE 绑定选项。
- *dft_degree* 数据库配置参数设置 CURRENT DEGREE 专用寄存器和 DEGREE 绑定选项的缺省值。

如果使用 DEGREE = ANY 来编译一个查询，那么数据库管理器根据许多因素，包括处理器数目和此查询的特征，来选择分区内并行度。取决于这些因素和系统上的活动量，运行时使用的实际并行度可能低于处理器数。如果大量使用了系统，那么在查询执行之前可能降低并行性。发生这种情况是因为，分区内并行性积极使用系统资源来减少查询的耗用时间，而这可能相反地影响其他数据库用户的性能。

要显示关于优化器所选择的并行度的信息，使用“SQL 说明工具”来显示访问方案。使用数据库系统监视器来显示关于运行时实际使用的并行度的信息。

非 SMP 环境中的并行性

您可以指定并行度而不必具有 SMP 机器。例如，在一个单处理器机器上的受 I/O 约束的查询可由于将并行度声明为 2 或更高值而受益。在这种情况下，处理器可能不必等待输入或输出任务完成，就可开始处理新的查询。但是，声明并行度为 2 或更高值并不直接控制一个单处理器机器上的 I/O 并行性。诸如 Load 实用程序这样的实用程序可以不依赖这种声明来控制 I/O 并行性。关键字 ANY 也可用来设置 *dft_degree* 数据库管理器配置参数。ANY 关键字允许优化器确定分区内并行度。

第 19 章 环境注意事项

表空间对查询优化的影响

表空间的某些特征可影响查询编译器选择的访问方案:

- 容器特征

  容器特征可显著影响查询执行期间的关联 I/O 成本。当选择访问方案时，查询优化器会考虑这些 I/O 成本，包括访问不同表空间中的数据所产生的任何成本差别。优化器使用 SYSCAT.TABLESPACES 系统目录中的两列来帮助估计访问表空间中的数据的 I/O 成本:

  – OVERHEAD，它提供容器所需时间的估计（以毫秒计），在该时间后才将数据读入内存。此开销活动包括容器的 I/O 控制器开销以及磁盘等待时间，后者包括磁盘寻道时间。

    可使用以下公式来帮助估计开销成本:

    ```
    OVERHEAD = 以毫秒计的平均寻道时间
                          +（0.5 * 旋转等待时间）
    ```

    其中:

    - 0.5 表示半个旋转的平均开销
    - 对每个完整的旋转计算以毫秒计的旋转等待时间，如下所示:

      ```
      (1 / RPM) * 60 * 1000
      ```

      其中:

      - 除以每分钟旋转次数来获得每个旋转所需的分钟数
      - 乘以 60 秒/分
      - 乘以 1000 毫秒/秒

    例如，假定磁盘每分钟旋转 7200 次。通过使用旋转延迟时间公式，这将生成:

    ```
    (1 / 7200) * 60 * 1000 = 8.328 毫秒
    ```

    然后可假定平均寻道时间为 11 毫秒，使用以上结果计算 OVERHEAD 估计值:

    ```
    OVERHEAD = 11 + (0.5 * 8.328)
                    = 15.164
    ```

    得出大约 15 毫秒的 OVERHEAD 估计值。

  – TRANSFERRATE，它提供将一页数据读入内存所需的时间估计（以毫秒计）。

    如果每个表空间容器是单个物理磁盘，那么可使用以下公式来帮助估计传送成本（以毫秒/页计）:

    ```
    TRANSFERRATE = (1 / spec_rate) * 1000 / 1024000 * page_size
    ```

    其中:

    - spec_rate 表示磁盘传送速率的技术要求，以 MB/秒计
    - 除以 spec_rate 来获得秒/MB

- 乘以 1000 毫秒/秒
- 除以 1024000 个字节/MB
- 乘以页大小（以字节计）（例如，一个 4 KB 页具有 4096 个字节）

比如，假设磁盘的技术要求速率为 3 MB/秒。这将得出以下计算

```
TRANSFERRATE = (1 / 3) * 1000 / 1024000 * 4096
                       = 1.333248
```

得到大约每页 1.3 毫秒的估计 TRANSFERRATE 值。

如果表空间容器不是单个物理磁盘，而是磁盘阵列（如 RAID），那么当试图确定要使用的 TRANSFERRATE 时必须考虑其他注意事项。假设瓶颈处于磁盘级，如果阵列相对较小，那么可用磁盘数乘以 spec_rate。

但是，如果组成容器的阵列中磁盘数目较大，那么瓶颈可能不是在磁盘级别，而是在其他某个 I/O 子系统组件级别，如磁盘控制器、I/O 总线或系统总线。在这种情况下，不能假设 I/O 的吞吐量能力是 spec_rate 和磁盘数目的乘积。而必须在顺序扫描期间测量实际的 I/O 速率（以 MB 计）。例如，一个顺序扫描可能是 select count(*) from big_table，并且它的大小会是几 MB。将该结果除以组成表空间的容器数目，big_table 位于这种表空间中。将该结果替换上面给出的公式中的 spec_rate。例如，在扫描含有 4 个容器的表空间中的某个表时，所测得的 100 MB 的顺序 I/O 速率意味着每个容器 25 MB，或 TRANSFERRATE 为 (1/25) * 1000 / 1024000 * 4096 = 0.16 毫秒/页。

分配给一个表空间的每个容器可位于不同的物理磁盘上。要获得最佳结果，用于给定表空间的所有物理磁盘应具有相同的 OVERHEAD 和 TRANSFERRATE 特征。如果这些特征不相同，那么当设置 OVERHEAD 和 TRANSFERRATE 的值时您应使用平均值。

您可以从硬件规格或通过实验来获取这些列的媒体特定值。可在 CREATE TABLESPACE 和 ALTER TABLESPACE 语句上指定这些值。

在上面提到的把磁盘阵列作为容器的环境中，进行实验变得特别重要。应创建一个移动数据的简单查询，与特定于平台的测量实用程序一起使用。然后，用表空间中不同的容器配置重新运行该查询。可使用 CREATE 和 ALTER TABLESPACE 语句来更改在环境中传送数据的方式。

通过这两个值提供的 I/O 成本信息会以多种方式影响优化器，包括是否使用索引来访问数据，以及要选择将哪个表用于连接中的内部表和外部表。

- 预取

当考虑从表空间访问数据的 I/O 成本时，优化器也考虑从磁盘预取数据页和索引页可能给查询性能带来的潜在影响。预取数据页和索引页可减少将数据读入缓冲池所需的开销和等待时间。

优化器使用 SYSCAT.TABLESPACES 中 PREFETCHSIZE 和 EXTENTSIZE 列的信息，来估计将对一个表空间执行的预取量。

– 仅当创建表空间（例如，使用 CREATE TABLESPACE 语句）时才可设置 EXTENTSIZE。缺省扩展数据块大小为 32 页（每页 4 KB），通常足够了。

– 当创建表空间和/或使用 ALTER TABLESPACE 语句时，可设置 PREFETCHSIZE。缺省预取大小由 DFT_PREFETCH_SZ 数据库配置参数值来确定。复查关于估算此参数的建议并按需要进行更改，以改进数据移动。

以下显示更改 RESOURCE 表空间特征的语法示例:

```
ALTER TABLESPACE RESOURCE
        PREFETCHSIZE 64
        OVERHEAD     19.3
        TRANSFERRATE 0.9
```

当对表空间进行任何更改之后，考虑重新绑定应用程序，并执行 RUNSTATS 实用程序来收集有关索引的最新统计信息，以确保使用最佳访问方案。

影响联合数据库的服务器选项

联合系统由 DB2 DBMS（联合数据库）和一个或多个数据源组成。应在发出 CREATE SERVER 语句时，向联合数据库标识数据源。当发出这些语句时，可以包括一些服务器选项，这些选项优化并控制涉及 DB2 和指定数据源的联合系统操作的各方面。要在以后更改服务器选项，可使用 ALTER SERVER 语句。

注: 必须安装分布式连接安装选项，并将数据库管理器参数 *federated* 设置为 YES，然后才能创建服务器并指定服务器选项。

指定的服务器选项值影响查询下推分析、全局优化和联合数据库操作的其他方面。例如，在 CREATE SERVER 语句中，可以指定性能统计信息作为服务器选项值，如 *cpu_ratio* 选项，它指定数据源和联合服务器上 CPU 的相对速度。也可将 *cpu_ratio* 选项设置成指示源和联合服务器上数据 I/O 划分的相对速率的值。执行 CREATE SERVER 语句时，将此数据添加到目录视图 SYSCAT.SERVEROPTIONS 中，优化器使用它来制订数据源的访问方案。如果统计信息更改（例如，如果升级了数据源 CPU，那么这种情况可能会发生），那么使用 ALTER SERVER 语句以用此更改来更新 SYSCAT.SERVEROPTIONS。然后，优化器在下次选择数据源的访问方案时，使用新的信息。

第 20 章 目录统计信息

当查询编译器优化查询方案时，其决定主要受有关数据库表、索引和统计信息视图大小的统计信息的影响。如果使用表、索引和统计信息视图的特定列来选择行或连接表，那么优化器也使用有关这些列中的数据分布的信息。优化器使用此信息来估计每个查询的备用访问方案的成本。

除表大小和数据分布信息外，还可以收集下列信息：索引的集群比率、索引中叶子页数目、溢出其原始页的表行数以及表中已填充的页数和空页数。使用此信息来决定何时重组表和索引。

在分区数据库环境中收集某个表的统计信息时，仅收集位于执行实用程序的数据库分区上的那部分表的统计信息，或者仅收集数据库分区组中包含该表的第一个数据库分区的统计信息。收集统计信息视图的统计信息时，将收集所有数据库分区的统计信息。

当对表、统计信息视图或表及其关联索引执行 RUNSTATS 实用程序时，下列几种统计信息存储在系统目录表中:

对于表和索引:

- 正在使用的页数
- 包含行的页数
- 溢出的行数
- 表中的行数（基数）
- 对于 MDC 表，包含数据的块数
- 对于分区表，单个数据分区内数据的集群度

对于表或统计信息视图中的每列和索引键中的第一列:

- 列的基数
- 列的平均长度
- 列中次高值
- 列中次低值
- 列中的空值数

对于每个 XML 列，将收集以下统计信息。XML 列中的每行存储一个 XML 文档。给定路径或路径值对的节点计数指的是路径或路径值对可到达的节点数。给定路径或路径值对的文档计数指的是包含给定路径或路径值对的文档数。

- 空 XML 文档数
- 非空 XML 文档数
- 单值路径数
- 每个单值路径的节点计数总和
- 每个单值路径的文档计数总和
- 具有最大节点计数的 *k* 对（路径，节点计数）

- 具有最大文档计数的 *k* 对（路径，文档计数）
- 具有最大节点计数的 *k* 个三元组（路径，值，节点计数）
- 具有最大文档计数的 *k* 个三元组（路径，值，文档计数）
- 对于导向文本或属性值的每个单值路径:
  - 此路径可以接受的单值数
  - 最大值
  - 最小值
  - 文本或属性节点数
  - 包含文本或属性节点的文档数

对于指定的列组:

- 列组的基于时间戳记的名称
- 列组的基数

仅对于索引:

- 索引|条目的数目（索引|基数）
- 叶子页数
- 索引|层数
- 此索引的表数据的集群度
- 与数据分区有关的索引键的集群度
- 磁盘上按索引键顺序的叶子页与索引占用的页范围中页数的比率
- 在索引的第一列中的单值数
- 在索引的前二、三和四列中的单值数
- 在索引的所有列中的单值数
- 按索引键顺序位于磁盘上的叶子页的数目，在这些叶子页之间很少有或没有大的间隔
- 页中所有 RID 标记为删除的页数
- 页中标记为删除的 RID 数，并非页中所有 RID 都标记为删除

如果请求索引的详细统计信息，那么还可以存储关于表对索引的集群度的更详细信息以及不同缓冲区大小的页访存估计。

还可以收集关于表和索引的下列几种统计信息:

- 数据分布统计信息

  优化器使用数据分布统计信息来估计表和统计信息视图的有效访问方案，在这些表和统计信息视图中数据不是均匀分布的，且列具有大量重复值。

- 详细的索引统计信息

  优化器使用详细索引统计信息来确定通过索引访问表的有效程度。

- 子元素统计信息

  优化器对 LIKE 谓词使用子元素统计信息，特别是对于搜索嵌入字符串中的模式的那些谓词，如 `LIKE %disk%`。

不收集分布统计信息:

- 当将 *num_freqvalues* 和 *num_quantiles* 配置参数设置为零（0）时
- 当了解数据分布时，例如，当每个数据值是唯一的时候
- 当列是从未收集其统计信息的数据类型时。这些数据类型为 LONG、大对象（LOB）或结构化列
- 对于子表中的行类型，不收集表级统计信息 NPAGES、FPAGES 和 OVERFLOW
- 如果请求分位数分布，但该列中只有一个非空的值
- 对于已扩展的索引或已声明临时表

注: 可以对已声明临时表执行 RUNSTATS，但结果统计信息不会存储在系统目录中，因为已声明临时表不具有目录条目。但是，统计信息存储在为已声明临时表提供产品目录信息的内存结构中。因此，在某些情况下，对这些表执行 RUNSTATS 可能会有用。

自动收集统计信息

DB2 优化器使用目录统计信息来确定任何给定查询的最佳访问方案。如果有关表或索引的统计信息已过时或者不完整，那么可能会导致优化器选择不是最佳的方案，并且会降低执行查询的速度。但是，决定要为给定的工作负载收集哪些统计信息是很复杂的事情，并且使这些统计信息保持最新是一项很花费时间的任务。

通过使用自动收集统计信息功能（它是 DB2 的自动表维护功能的一部分），可以让 DB2 数据库管理器确定是否需要更新数据库统计信息。可以在编译语句时使用实时统计信息功能来执行自动收集统计信息操作，也可以在后台运行 RUNSTATS 实用程序来收集统计信息。可以在实时收集统计信息功能被禁用时启用后台收集统计信息。必须启用后台收集统计信息才能启用实时收集统计信息。从 DB2 版本 9 开始，当您创建新的数据库时，缺省情况下就会启用后台自动收集统计信息。从 DB2 版本 9.5 开始，当您创建新的数据库时，缺省情况下会同时启用后台自动收集统计信息和实时统计信息功能。

了解后台收集和实时收集统计信息

通过运行 RUNSTATS 实用程序，可采用同步或异步方式自动收集统计信息。*异步*收集在后台执行。当实时统计信息功能启用后，还可以在编译语句时*同步*收集统计信息。当实时统计信息功能启用后，还可以使用数据和索引管理器维护的元数据生成统计信息。查询优化器根据查询需求和表更新活动的数量来确定统计信息的收集方式。根据更新、删除和插入的次数来衡量表更新活动。

实时统计信息由优化 SQL 语句前该语句的需求确定。这样可以更及时地收集更准确的统计信息。准确的统计信息可以产生更好的查询执行计划并提高性能。未启用实时统计信息功能时，每隔 2 个小时就会执行一次异步收集统计信息。要为某些应用程序提供准确的统计信息，此收集频率可能不够高。

启用实时统计信息功能后，仍会每隔 2 个小时就执行一次异步收集统计信息检查。在下列情况下，实时统计信息还发起异步收集请求:

- 表活动频率对于同步收集来说不够高，但对于异步收集来说又太高。
- 由于表太大，同步收集统计信息使用了采样。
- 同步统计信息是生成的。
- 由于超过收集时间，同步收集统计信息失败。

最多可以同时处理两个异步请求，但只能针对不同表。一个请求由实时统计信息发起，而另一个请求由异步收集统计信息检查发起。

通过使用下面几种方法，可以使自动收集统计信息对性能的影响降到最低:

- 通过使用已调速的 RUNSTATS 实用程序执行异步收集统计信息。调速功能根据当前数据库活动来控制 RUNSTATS 实用程序消耗的资源数量: 随着数据库活动的增加，RUNSTATS 实用程序的运行速度将变慢，从而减少了它的资源需求。
- 对于每个查询，同步收集统计信息的时间限制为 5 秒。此值可由 RTS 优化准则控制。如果同步收集操作超过时间限制，那么将提交异步收集请求。
- 同步收集统计信息不会将统计信息存储在系统目录中。相反，它会将统计信息存储在统计信息高速缓存中，并稍后由异步操作存储在系统目录中。这样可避免更新系统目录所产生的开销和可能的锁定争用。后续 SQL 编译请求可使用统计信息高速缓存中的统计信息。
- 对于每个表，只能执行一个同步收集统计信息操作。其他需要同步收集统计信息的代理程序将在可能的情况下生成统计信息，并继续编译语句。在 DPF 环境中也会强制执行此行为，此环境中的不同数据库分区上的代理程序可能需要同步统计信息。
- 缺省情况下，收集的关于同步和异步操作的统计信息是基本表统计信息和分发信息，以及使用采样的详细索引统计信息。(发出的 RUNSTATS 命令带有 WITH DISTRIBUTION 和 SAMPLED DETAILED INDEXES ALL 选项。)可以通过启用统计信息概要分析来定制收集的统计信息类型，统计信息概要分析使用有关先前的数据库活动的信息来确定数据库工作负载需要的统计信息。通过为特定表创建您自己的统计信息概要文件，还可以定制收集的关于该表的统计信息类型。
- 只对缺少统计信息或具有高级别活动（根据更新、删除和插入的次数来衡量）的表进行统计信息收集。但即使在表符合统计信息收集条件时，也不会收集同步统计信息，除非查询优化需要该信息。在某些情况下，查询优化器可以不使用统计信息来选择访问方案。
- 对于异步收集统计信息检查，还对大型表（包含 4000 页以上）进行采样以确定高级表活动是否真正更改了统计信息。只有确实更改了，才会收集这些大型表的统计信息。
- 对于异步收集统计信息，自动将 RUNSTATS 实用程序安排在维护策略定义中指定的最佳维护时间段内运行。此策略还指定自动收集统计信息作用域内的一组表，这进一步减少了不必要的资源消耗。
- 同步收集统计信息和生成统计信息操作不在维护策略定义中指定的维护时间段后发生。这是因为同步请求必须立即执行并且具有的收集时间有限。同步收集统计信息和生成统计信息操作遵循用于指定自动收集统计信息作用域内的一组表的策略。
- 当正在执行自动进行的统计信息收集时，受影响的表仍然可用于常规数据库活动（更新、插入和删除），就好像没有对表运行 RUNSTATS 命令一样。
- 对于异步收集统计信息，通过使用基于目录的收集方法来运行 SYSPROC.NNSTAT 过程，从而自动刷新昵称统计信息。不收集昵称的实时统计信息（同步或生成的）。

对常规表、具体化查询表（MQT）和已声明的全局临时表（DGTT）执行实时同步统计信息收集。

不收集 DGTT 的异步统计信息。这表示实时统计信息处理将不发起 DGTT 的异步请求。

不会对下列对象执行自动收集统计信息（同步或异步）操作:

- 统计视图
- 标记为 VOLATILE 的表（在 SYSCAT.TABLES 中设置了 VOLATILE 字段的表）
- 通过直接对 SYSSTAT 目录视图发出 UPDATE 语句手动更新了其统计信息的表

  如果手动修改了统计信息，那么 DB2 认为用户正在维护表的统计信息。因此，DB2 将不维护该表的统计信息。为了让 DB2 维护已手动更新其统计信息的表的统计信息，请对该表发出手动 RUNSTATS 命令。手动更新了其统计信息的已迁移表的统计信息将由 DB2 自动维护。

在数据库分区功能部件（DPF）环境中，在单个数据库分区上收集统计信息并进行推测。如果一个表存在于多个数据库分区上并且该表没有统计信息，那么 DB2 数据库管理器总是在数据库分区组的第一个数据库分区上收集统计信息（同步和异步）。如果该表已经具有统计信息，那么在最后一次收集统计信息的数据库分区上收集统计信息。这确保统计信息的一致性，因为它们总是在相同数据库分区上收集的。

至少在数据库激活 5 分钟后才会执行非实时统计信息收集活动。

启用实时统计信息功能后，应安排已定义的维护时间段。缺省情况下，维护时间段未定义，它包括所有时间。如果没有已定义的维护时间段，那么将只进行同步统计信息收集。在这种情况下，收集的统计信息仅在内存中并且通常是使用采样收集的（小型表除外）。

将对静态和动态 SQL 执行实时统计信息处理。

已使用 IMPORT 命令截断的表将自动重组为具有旧统计信息。

对于收集了其统计信息的表，同步和异步自动收集统计信息操作将使引用这些表的已高速缓存动态语句失效。这样做是为了能够使用最新统计信息重新优化已高速缓存的动态语句。

实时统计信息和说明处理

不会对刚刚使用说明工具说明（不是执行）的查询进行实时处理。下表总结了 CURRENT EXPLAIN MODE 寄存器的不同值的行为。

表 51. CURRENT EXPLAIN MODE 寄存器的实时统计信息收集行为

CURRENT EXPLAIN MODE	是否考虑实时收集统计信息
YES	YES
EXPLAIN	NO
NO	YES
REOPT	YES
RECOMMEND INDEXES	NO
EVALUATE INDEXES	NO

自动收集统计信息和统计信息高速缓存

为了使同步收集的统计信息对所有查询可用，在 DB2 版本 9.5 中引入了统计信息高速缓存。此高速缓存是目录高速缓存的一部分。在数据库分区功能部件（DPF）环境中，此高速缓存仅位于目录数据库分区上。目录高速缓存可以存储同一 SYSTABLES 对

象的多个条目，这将增大所有数据库分区上的目录高速缓存大小。在启用了实时统计信息功能时，考虑增大 **CATALOGCACHE_SZ** 数据库配置参数。

从 DB2 版本 9 开始，配置顾问程序用来确定新数据库的初始数据库配置。配置顾问程序总是建议将 **AUTO_STMT_STATS** 配置参数设置为 ON。

自动收集统计信息和统计概要文件

根据表的有效统计概要文件同步和异步收集统计信息，但下列情况例外:

- 为了使同步收集统计信息的开销降至最低，DB2 数据库管理系统可以使用采样来收集统计信息。在这种情况下，采样率和方法可能与统计概要文件中指定的采样率和方法不同。
- 同步收集统计信息可以选择生成统计信息。不可能生成统计概要文件中指定的所有统计信息。例如，不能生成诸如 COLCARD、HIGH2KEY 和 LOW2KEY 之类的列统计信息，除非该列在某些索引中是前导列。

如果同步收集统计信息操作不能收集统计概要文件中指定的所有统计信息，那么将会提交异步收集请求。

虽然实时收集统计信息方法旨在使收集统计信息的开销降至最低，但请先在测试环境中尝试使用该方法，以确保不会对性能产生负面影响。对某些 OLTP 方案使用该方法可能会对性能产生负面影响，尤其在查询的运行时间具有上限时。

启用自动收集统计信息

要进行高效率的数据访问和获得最佳工作负载性能，具有准确而完整的数据库统计信息是很关键的。使用自动进行表维护功能的自动收集统计信息功能来更新和维护相关的数据库统计信息。可以选择通过使用以下方法在串行环境（也就是单个数据库分区在单个处理器上运行的环境）中增强此功能: 收集查询数据并生成统计信息概要文件，这可以帮助 DB2 自动收集工作负载需要的一组准确的统计信息。在 MPP 环境、某些联合环境或者启用了分区内并行性的环境中，此选项不可用。

要启用自动收集统计信息:

1. 使用“配置自动维护”向导或者命令行来配置数据库实例:
   - 要使用“配置自动维护”向导:
     a. 通过以下两种方法来打开此向导: 一种方法是在“控制中心”中右键单击数据库对象；另一种方法是在“运行状况中心”中右键单击数据库实例。
     b. 从弹出窗口中选择**配置自动维护**。在此向导中，可以启用自动收集统计信息、指定想要自动从其中收集统计信息的表以及指定用于执行 RUNSTATS 实用程序的维护窗口。
   - 要使用命令行，请将下列每个配置参数都设置为 ON:
     – **AUTO_MAINT**
     – **AUTO_TBL_MAINT**
     – **AUTO_RUNSTATS**
2. 可选: 要启用自动生成统计信息概要文件，应将下面两个配置参数都设置为 ON:
   - **AUTO_STATS_PROF**
   - **AUTO_PROF_UPD**

3. 可选：要启用收集实时统计信息，将 **AUTO_STMT_STATS** 配置参数设置为 ON。如果此配置参数设置为 ON，那么每当需要表统计信息来优化查询时，就会在编译语句时自动编译这些统计信息。

自动进行统计信息收集和概要分析使用的存储器

自动进行统计信息收集和重组功能部件将工作数据存储在数据库的表中。这些表是在 SYSTOOLSPACE 表空间中创建的。当激活数据库时，会自动使用缺省选项来创建 SYSTOOLSPACE 表空间。这些表的存储器需求与数据库中表的数目成比例，应该按每个表大约 1 KB 来计算。如果这对于数据库来说太大了，那么您可能想删除并亲自重新创建表空间，然后再适当地分配存储器。将自动重新创建表空间中的自动维护和运行状况监视器表。当删除表空间时，在这些表中捕获的任何历史记录都会丢失。

对自动收集统计信息活动进行日志记录

为了解对数据库发生了哪些统计信息收集活动，在 DB2 版本 9.5 中引入了统计信息日志。统计信息日志记录数据库的所有统计信息活动，包括自动和手动收集统计信息。

统计信息日志的缺省名称为 db2optstats.*number*.log。它位于 $DIAGPATH/events 目录中。统计信息日志是旋转日志。统计信息日志行为由 **DB2_OPTSTATS_LOG** 注册表变量控制。

可以直接查看统计信息日志的内容，或者使用 SYSPROC.PD_GET_DIAG_HIST 表函数来查询这些内容。

SYSPROC.PD_GET_DIAG_HIST 表函数返回一定数目的列，这些列包含关于任何日志记录的事件的标准信息，如时间戳记、DB2 实例名称、数据库名称、进程标识、进程名称和线程标识。日志还包含供不同日志记录功能使用的类属列。下表描述了统计信息日志如何使用这些类属列。

表 52. 统计信息日志文件中的类属列

列名	数据类型	描述
OBJTYPE	VARCHAR(64)	事件适用于的对象类型。对于统计信息日志记录，这是要收集的统计信息的类型。在统计信息收集后台进程启动或停止时，它也可以指该后台进程。它还可以指自动收集统计信息执行的活动，如采样测试、初始采样和表求值活动。 统计信息收集操作的可能值包括: **TABLE STATS** 将收集表统计信息。 **INDEX STATS** 将收集索引统计信息。 **TABLE AND INDEX STATS** 同时收集表统计信息和索引统计信息。 自动统计信息操作的可能值包括: **EVALUATION** 自动收集统计信息后台进程已开始求值阶段。在求值阶段，将检查表以确定它们是否需要统计信息，如果需要，将收集统计信息。 **INITIAL SAMPLING** 使用采样收集表的统计信息。采样的统计信息将存储在系统目录中。这将允许自动收集统计信息快速收集没有统计信息的表的统计信息。后续统计信息收集操作将收集未采样的统计信息。初始采样在自动收集统计信息的求值阶段执行。 **SAMPLING TEST** 使用采样收集表的统计信息。采样的统计信息不存储在系统目录中。将采样的统计信息与当前目录统计信息进行比较，以确定是否以及何时应收集此表的完整统计信息。采样测试在自动收集统计信息的求值阶段执行。 **STATS DAEMON** 统计信息守护程序是用来处理实时统计信息处理所提交的请求的后台进程。当后台进程启动或停止时，将对此对象类型进行日志记录。
OBJNAME	VARCHAR(255)	事件适用于的对象名称（如果可用的话）。对于统计信息日志，它包含表或索引名。如果 OBJTYPE 是 STATS DAEMON 或 EVALUATION，那么 OBJNAME 是数据库名称，而 OBJNAME_QUALIFIER 是 NULL。
OBJNAME_QUALIFIER	VARCHAR(255)	对于统计信息日志，它包含表或索引模式。

表 52. 统计信息日志文件中的类属列 (续)

列名	数据类型	描述
EVENTTYPE	VARCHAR(24)	事件类型是与此事件关联的操作或动词。统计信息日志记录的可能值包括: **COLLECT** 为统计信息收集操作对此操作进行日志记录。 **START** 当实时统计信息后台进程（OBJTYPE = STATS DAEMON）或自动收集统计信息求值阶段（OBJTYPE = EVALUATION）启动时，对此操作进行日志记录。 **STOP** 当实时统计信息后台进程（OBJTYPE = STATS DAEMON）或自动收集统计信息求值阶段（OBJTYPE = EVALUATION）停止时，对此操作进行日志记录。 **ACCESS** 试图访问一个表以收集统计信息。当对象不可用时，此事件类型用于对不成功的访问进行日志记录。
FIRST_EVENTQUALIFIERTYPE	VARCHAR(64)	第一个事件限定词的类型。事件限定词用于描述受事件影响的对象。对于统计信息日志，第一个事件限定词是事件发生时的时间戳记。 对于第一个事件限定词类型，值为 AT。
FIRST_EVENTQUALIFIER	CLOB(16k)	事件的第一个限定词。对于统计信息日志记录，第一个事件限定词是统计信息事件发生时的时间戳记。统计信息事件的时间戳记可能与 TIMESTAMP 列所表示的日志记录的时间戳记不同。
SECOND_EVENTQUALIFIERTYPE	VARCHAR(64)	第二个事件限定词的类型。对于统计信息日志记录，值可以为 BY 或 NULL。此字段不用于其他事件类型。

表 52. 统计信息日志文件中的类属列 (续)

列名	数据类型	描述
SECOND_EVENTQUALIFIER	CLOB(16k)	事件的第二个限定词。 对于统计信息日志记录，它表示收集 COLLECT 事件类型的统计信息的方式。可能的值包括: **用户** 由 DB2 用户使用 RUNSTATS、LOAD、CREATE INDEX 或 REDISTRIBUTE 命令执行统计信息收集。 **同步** 由 DB2 在编译 SQL 语句时执行统计信息收集。统计信息将存储在统计信息高速缓存中，而不是系统目录中。 **同步采样** 由 DB2 在编译 SQL 语句时使用采样执行统计信息收集。统计信息将存储在统计信息高速缓存中，而不是系统目录中。 **生成** 在编译 SQL 语句时通过使用数据和索引管理器维护的信息来生成统计信息。统计信息将存储在统计信息高速缓存中，而不是系统目录中。 **部分生成** 在编译 SQL 语句时通过使用数据和索引管理器维护的信息仅生成某些统计信息。特别是，仅生成某些列的 HIGH2KEY 和 LOW2KEY。统计信息将存储在统计信息高速缓存中，而不是系统目录中。 **异步** 由 DB2 后台进程收集统计信息并存储在系统目录中。 此字段不用于其他事件类型。
THIRD_EVENTQUALIFIERTYPE	VARCHAR(64)	第三个事件限定词的类型。对于统计信息日志记录，值可以为 DUE TO 或 NULL。

表 52. 统计信息日志文件中的类属列 (续)

列名	数据类型	描述
THIRD_EVENTQUALIFIER	CLOB(16k)	事件的第三个限定词。 对于统计信息日志记录，它表示统计信息活动无法完成的原因。 可能的值包括: **超时** 同步收集统计信息操作超过时间预算。 **错误** 由于发生错误，统计信息活动失败。 **RUNSTATS 错误** 由于发生 RUNSTATS 错误，同步收集统计信息失败。 对于某些错误，即使未能收集统计信息，SQL 语句编译操作也可能成功完成。例如，在没有足够的内存来收集统计信息时，将继续编译 SQL 语句。 **对象不可用** 由于无法访问数据库对象，未能收集该对象的统计信息。某些可能的原因包括: • 对象以超级互斥（Z）方式被锁定 • 对象所在的表空间不可用 • 需要重新创建表索引 **冲突** 由于另一个应用程序已在收集同步统计信息，所以未能执行同步统计信息收集。 检查 FULLREC 列或 db2diag.log 以了解错误详细信息。
EVENTSTATE	VARCHAR(255)	由于事件导致的对象或操作的状态。 对于统计信息日志记录，它指示统计信息操作的状态。 可能的值包括: • 启动 • 成功 • 失败

示例

在此示例中，查询通过调用 PD_GET_DIAG_HIST 返回最多当前时间戳记前一年的事件的统计信息日志记录。

```
SELECT
pid,
tid,
substr(eventtype, 1,10),
substr(objtype,1,30) as objtype,
substr(objname_qualifier,1,20) as objschema,
substr(objname,1,10) as objname,
substr(first_eventqualifier,1,26) as event1,
substr(second_eventqualifiertype,1,2) as event2_type,
substr(second_eventqualifier,1,20) event2,
substr(third_eventqualifiertype,1,6) event3_type,
substr(third_eventqualifier,1,15) event3,
```

```
substr(eventstate,1,20) as eventstate
FROM
TABLE( SYSPROC.PD_GET_DIAG_HIST
       ( 'optstats', 'EX', 'NONE',
          CURRENT_TIMESTAMP - 1 year, CAST( NULL AS TIMESTAMP ))) as sl
order by timestamp(varchar(substr(first_eventqualifier,1,26),26))
;
```

结果按 FIRST_EVENTQUALIFIER 列中存储的时间戳记进行排序，该时间戳记表示统计信息事件的时间。

```
PID   TID                EVENTTYPE  OBJTYPE                        OBJSCHEMA           OBJNAME    EVENT1                     EVENT2_TYPE EVENT2               EVENT3_TYPE EVENT3          EVENTSTATE
----- ------------------ ---------- ------------------------------ ------------------- ---------- -------------------------- ----------- -------------------- ----------- --------------- ---------
28399          1082145120 START      STATS DAEMON                   -                   PROD_DB    2007-07-09-18.37.40.398905 -           -                    -           -               成功
28389        183182027104 COLLECT    TABLE AND INDEX STATS          DB2USER             DISTRICT   2007-07-09-18.37.43.261222 BY          同步                 -           -               启动
28389        183182027104 COLLECT    TABLE AND INDEX STATS          DB2USER             DISTRICT   2007-07-09-18.37.43.407447 BY          异步                 -           -               成功
28399          1082145120 COLLECT    TABLE AND INDEX STATS          DB2USER             CUSTOMER   2007-07-09-18.37.43.471614 BY          异步                 -           -               启动
28399          1082145120 COLLECT    TABLE AND INDEX STATS          DB2USER             CUSTOMER   2007-07-09-18.37.43.524496 BY          异步                 -           -               成功
28399          1082145120 STOP       STATS DAEMON                   -                   PROD_DB    2007-07-09-18.37.43.526212 -           -                    -           -               成功
28389        183278496096 COLLECT    TABLE STATS                    DB2USER             ORDER_LINE 2007-07-09-18.37.48.676524 BY          同步采样             -           -               启动
28389        183278496096 COLLECT    TABLE STATS                    DB2USER             ORDER_LINE 2007-07-09-18.37.53.677546 BY          同步采样             DUE TO      超时            失败
28389          1772561034 START      EVALUATION                     -                   PROD_DB    2007-07-10-12.36.11.092739 -           -                    -           -               成功
28389          8231991291 COLLECT    TABLE AND INDEX STATS          DB2USER             DISTRICT   2007-07-10-12.36.30.737603 BY          异步                 -           -               启动
28389          8231991291 COLLECT    TABLE AND INDEX STATS          DB2USER             DISTRICT   2007-07-10-12.36.34.029756 BY          异步                 -           -               成功
28389          1772561034 STOP       EVALUATION                     -                   PROD_DB    2007-07-10-12.36.39.685188 -           -                    -           -               成功
```

提高大型统计信息日志的查询性能

如果统计信息日志文件非常大，可以通过将日志记录复制到表中、创建索引并收集统计信息来提高查询性能。

过程

1. 创建一个表以保存日志记录和必需的列。

```
create table db2user.stats_log
(pid       bigint,
 tid       bigint,
 timestamp timestamp,
 dbname    varchar(128),
 retcode   integer,
 eventtype varchar(24),
 objtype   varchar(30),
 objschema varchar(20),
 objname   varchar(30),
 event1_type varchar(20),
 event1      timestamp,
 event2_type varchar(20),
 event2      varchar(40),
 event3_type varchar(20),
 event3      varchar(40),
 eventstate  varchar(20)
);
```

2. 为基于 SYSPROC.PD_GET_DIAG_HIST 的查询声明游标。

```
declare c1 cursor for
SELECT
pid,
tid,
timestamp,
dbname,
retcode,
eventtype,
substr(objtype,1,30) as objtype,
substr(objname_qualifier,1,20) as objschema,
substr(objname,1,30) as objname,
substr(first_eventqualifiertype,1,20),
substr(first_eventqualifier,1,26),
substr(second_eventqualifiertype,1,20),
substr(second_eventqualifier,1,40),
substr(third_eventqualifiertype,1,20),
substr(third_eventqualifier,1,40),
```

```
substr(eventstate,1,20)
FROM
TABLE( SYSPROC.PD_GET_DIAG_HIST
       ( 'optstats', 'EX', 'NONE',
          CURRENT_TIMESTAMP - 1 year, CAST( NULL AS TIMESTAMP ))) as sl
;
```

3. 使用 LOAD 命令和“从游标装入”功能将统计信息日志记录装入到该表中。

```
load from c1 of cursor replace into db2user.stats_log;
```

4. 在表上创建索引并收集关于该表的统计信息。

```
create index sl_ix1 on db2user.stats_log(eventtype, event1);
create index sl_ix2 on db2user.stats_log(objtype, event1);
create index sl_ix3 on db2user.stats_log(objname);

runstats on table db2user.stats_log with distribution and sampled detailed
indexes all;
```

收集和更新统计信息的准则

RUNSTATS 命令收集表、索引和统计信息视图的统计信息，以为优化器提供准确信息进行访问方案选择。

在下列情况下，使用 RUNSTATS 实用程序来收集统计信息:

- 当数据已装入表中且已创建适当的索引时。
- 当在表中创建新的索引时。如果自从上次在表中运行 RUNSTATS 以来尚未修改表，那么只需要对新的索引执行 RUNSTATS。
- 当一个表已用 REORG 实用程序重组时。
- 当通过数据修改、删除和插入已大量更新表及其索引时。（此处所指的“大量”可能表示有 10% 到 20% 的表和索引数据受影响。）
- 在绑定性能非常重要的应用程序之前
- 当您想要比较当前和先前统计信息时。如果定期更新统计信息，那么可以及早发现性能问题。
- 当预取量更改时。
- 当使用了 REDISTRIBUTE DATABASE PARTITION GROUP 实用程序时。

  注: 在先前版本的 DB2 中，此命令使用了 NODEGROUP 关键字，而不是 DATABASE PARTITION GROUP 关键字。
- 使用 RUNSTATS 实用程序来收集关于 XML 列的统计信息。使用 RUNSTATS 仅收集 XML 列的统计信息时，将保留 LOAD 或上一次执行 RUNSTATS 实用程序已收集的非 XML 列的现有统计信息。如果先前已收集关于一些 XML 列的统计信息，那么在当前命令未收集关于该 XML 列的统计信息时，将删除先前收集的 XML 列的统计信息；在当前命令收集了关于该 XML 列的统计信息时，将替换先前收集的 XML 列的统计信息。

要提高 RUNSTATS 性能并保存用来存储统计信息的磁盘空间，考虑仅指定应该收集其数据分布统计信息的列。

理论上，您应在运行统计信息之后重新绑定应用程序。如果查询优化器具有新的统计信息，那么它可以选择不同的访问方案。

如果您没有足够的时间一次收集全部的统计信息，那么可以运行 RUNSTATS 来每次仅更新几个表、索引或统计信息视图的统计信息，并轮流完成该组对象。如果对选择性部分更新运行 RUNSTATS 期间由于表上的活动而产生了不一致性，那么在查询优化期间将发出警告消息（SQL0437W，原因码 6）。例如，如果执行 RUNSTATS 来收集表分布统计信息，以及在某个表活动后，再次执行 RUNSTATS 来收集该表的索引统计信息，那么可能发生这种情况。如果由于表上的活动产生了不一致并且在查询优化期间检测到这些不一致，那么发出该警告消息。当发生这种情况时，应再次运行 RUNSTATS 来更新分布统计信息。

要确保索引统计信息和表同步，执行 RUNSTATS 来同时收集表和索引统计信息。索引统计信息保留自上次运行 RUNSTATS 以来收集的大部分表和列的统计信息。如果自上次收集该表的统计信息以来已对该表做了大量修改，那么只收集该表的索引统计信息将使两组统计信息不能在所有节点上都同步。

对生产系统调用 RUNSTATS 可能会对生产工作负载的性能产生负面影响。RUNSTATS 实用程序现在支持调速选项，在执行较高级别的数据库活动期间，可以使用调速选项来限制执行 RUNSTATS 的性能影响。

在分区数据库环境中收集表的统计信息时，RUNSTATS 仅收集执行该命令的数据库分区上的表的统计信息。将此数据库分区的 RUNSTATS 结果推广到其他数据库分区。如果执行 RUNSTATS 的数据库分区不包含特定表的一部分，那么将请求发送到数据库分区组中包含该表一部分的第一个数据库分区。

收集统计信息视图的统计信息时，将收集所有包含该视图引用的基本表的数据库分区的统计信息。

考虑以下技巧来提高 RUNSTATS 的效率和已收集的统计信息的有效性:

- 仅对用来连接表的列或 WHERE、GROUP BY 以及查询的类似子句中的列收集统计信息。如果对这些列建立了索引，那么可以用 RUNSTATS 命令的 ONLY ON KEY COLUMNS 子句指定列。
- 为特定表和表中特定列定制 *num_freqvalues* 和 *num_quantiles* 的值。
- 使用 SAMPLE DETAILED 子句收集 DETAILED 索引统计信息，以减少对详细的索引统计信息执行的后台计算量。SAMPLE DETAILED 子句减少收集统计信息所需要的时间，并在大多数情况下产生足够的精度。
- 当创建已填写的表的索引时，添加 COLLECT STATISTICS 子句来在创建索引时创建统计信息。
- 当添加或除去了大量表行时，或如果更新了收集其统计信息的列中的数据，那么再次执行 RUNSTATS 来更新统计信息。
- 因为 RUNSTATS 仅收集单个数据库分区的统计信息，所以，如果数据不是在所有数据库分区中一致分发的，那么统计信息将不太准确。如果您怀疑存在变形数据分发，那么您可能想要在执行 RUNSTATS 之前使用 REDISTRIBUTE DATABASE PARTITION GROUP 命令来在各数据库分区之间再分发数据。

收集目录统计信息

收集表、索引和统计信息视图的目录统计信息，以为优化器选择查询的最佳访问方案提供信息。

对于表和索引，您必须具有下列其中一种权限或特权:

- sysadm
- sysctrl
- sysmaint
- dbadm
- 对表的 CONTROL 特权
- LOAD 权限

对于统计信息视图，您必须具有下列其中一种权限和特权:

- sysadm
- sysctrl
- sysmaint
- dbadm
- 对表的 CONTROL 特权

此外，您还必须具有访问统计信息视图中的行的特权。特别是对于统计信息视图定义中引用的每个表、视图或昵称，您必须具有下列其中一种特权:

- SYSADM 或 DBADM
- CONTROL
- SELECT

要收集目录统计信息:

1. 连接至包含要收集其统计信息的表、索引或统计信息视图的数据库。
2. 从 DB2 命令行，执行带适当选项 RUNSTATS 命令。这些选项允许您为对表、索引或统计信息视图运行的查询定制要收集的统计信息。
3. 当 RUNSTATS 完成时，发出 COMMIT 语句以释放锁定。
4. 重新绑定访问表、索引或统计信息视图的程序包，您已重新生成这些表、索引或统计信息视图的统计信息。

要使用图形用户界面来指定选项并收集统计信息，使用“控制中心”。

注:

1. 因为 RUNSTATS 实用程序不支持使用昵称，所以应以不同方式来更新联合数据库查询的统计信息。如果查询访问联合数据库，那么对所有数据库中的表执行 RUNSTATS，然后删除并重新创建访问远程表的昵称，以使新的统计信息可用于优化器。
2.

   在分区数据库环境中收集表的统计信息时，RUNSTATS 仅收集执行该命令的数据库分区上的表的统计信息。将此数据库分区的 RUNSTATS 结果推广到其他数据库分区。如果执行 RUNSTATS 的数据库分区不包含特定表的一部分，那么将请求发送到数据库分区组中包含该表一部分的第一个数据库分区。

   收集统计信息视图的统计信息时，将收集所有包含该视图引用的基本表的数据库分区的统计信息。

收集特定列的分布统计信息

为了提高 RUNSTATS 和后续查询方案分析的效率，可以仅收集查询在 WHERE、GROUP BY 和类似子句中使用的列的分布统计信息。还可以收集关于列的组合组的基数统计信息。优化器使用这种信息来在它为引用组中列的查询估计选择性时检测列相关。

在下列步骤中，假定数据库为 **sales** 并包含表 **customers**，带有索引 **custidx1** 和 **custidx2**。

必须连接至包含表和索引的数据库并具有下列其中一个权限级别:

- sysadm
- sysctrl
- sysmaint
- dbadm
- 对表的 CONTROL 特权

注: RUNSTATS 仅收集执行该命令的数据库分区上的表的统计信息。将此数据库分区的 RUNSTATS 结果推广到其他数据库分区。如果执行 RUNSTATS 的数据库分区不包含表的一部分，那么将请求发送到数据库分区组中持有表的该部分的第一个数据库分区。

要收集特定列的统计信息:

1. 连接至 **sales** 数据库。
2. 取决于您的需求，在 DB2 命令行执行下列其中一个命令:
   - 要收集列 **zip** 和 **ytdtotal** 的分布统计信息:

     ```
     RUNSTATS ON TABLE sales.customers
              WITH DISTRIBUTION ON COLUMNS (zip, ytdtotal)
     ```

   - 要收集相同列的分布统计信息，但调整分布缺省值:

     ```
     RUNSTATS ON TABLE sales.customers
              WITH DISTRIBUTION ON
              COLUMNS (zip, ytdtotal NUM_FREQVALUES 50 NUM_QUANTILES 75)
     ```

   - 要收集索引为 **custidx1** 和 **custidx2** 的列的分布统计信息:

     ```
     RUNSTATS ON TABLE sales.customer
                     ON KEY COLUMNS
     ```

   - 要仅收集表中特定列 **zip** 和 **ytdtotal** 以及包含 **region** 和 **territory** 的列组的列统计信息:

     ```
     RUNSTATS ON TABLE sales.customers
                       ON COLUMNS (zip, (region, territory), ytdtotal)
     ```

   - 假定已使用带 STATISTICS 选项的 LOAD 命令收集了非 XML 列的统计信息。要用 XML 列 **miscinfo** 的统计信息补充非 XML 统计信息:

     ```
     RUNSTATS ON TABLE sales.customers
          ON COLUMNS (miscinfo)
     ```

   - 要仅收集表中非 XML 列的列统计信息（EXCLUDING XML COLUMNS 选项优先于可能指定 XML 列的所有其他子句）:

     ```
     RUNSTATS ON TABLE sales.customers
         EXCLUDING XML COLUMNS
     ```

还可以使用“控制中心”来收集分布统计信息。

收集索引统计信息

收集索引统计信息以允许优化器评估是否应该使用索引来解析查询。

在下列步骤中，假定数据库为 **sales** 并包含表 **customers**，带有索引 **custidx1** 和 **custidx2**。

必须连接至包含表和索引的数据库并具有下列其中一个权限级别:

- sysadm
- sysctrl
- sysmaint
- dbadm
- 对表的 CONTROL 特权

带 SAMPLED DETAILED 选项执行 RUNSTATS 需要 2MB 统计信息堆。将附加的 488 个 4K 页分配给为此附加内存需求设置的 *stat_heap_sz* 数据库配置参数。如果该堆看起来太小，那么 RUNSTATS 在尝试收集统计信息之前返回一条错误。

要收集索引的详细统计信息:

1. 连接至 **sales** 数据库。
2. 取决于您的需求，在 DB2 命令行执行下列其中一个命令:
   - 要创建 **custidx1** 和 **custidx2** 的详细统计信息:

     ```
     RUNSTATS ON TABLE sales.customers AND DETAILED INDEXES ALL
     ```

   - 要创建两个索引的详细统计信息，但使用采样而不对每个索引条目执行详细计算:

     ```
     RUNSTATS ON TABLE sales.customers AND SAMPLED DETAILED INDEXES ALL
     ```

   - 要创建索引上的详细采样统计信息以及表的分布统计信息，以便索引和表统计信息一致:

     ```
     RUNSTATS ON TABLE sales.customers
                         WITH DISTRIBUTION ON KEY COLUMNS
                         AND SAMPLED DETAILED INDEXES ALL
     ```

还可以使用“控制中心”来收集索引和表统计信息。

收集有关表数据的样本的统计信息

查询优化器使用表统计信息来为给定的任何查询选择最佳访问方案，因此，使统计信息始终保持为最新的是很重要的，以准确反映表在任何给定时间的状态。对表执行的活动越多，收集统计信息的频率就应该越高。随着数据库的大小增大，找到用来收集统计信息的有效方法就显得更重要。对要收集统计信息的表数据进行随机采样可提高 RUNSTATS 性能。对于 I/O 受限制或 CPU 受限制的系统，性能可以得到极大的优化。样本越小，完成 RUNSTATS 的速度就越快。

从版本 8.2 开始，RUNSTATS 命令提供了通过使用 TABLESAMPLE 选项来收集表中的数据样本的统计信息的选项。此功能可提高收集统计信息的效率，原因是采样只使用一部分数据。并且，这些采样方法可确保准确性更高。

可以采用两种方法来指定要如何收集样本。BERNOULLI 方法对行级别的数据进行采样。在数据页的整个表扫描期间，会依次考虑每一行并根据数字参数指定的概率 P 来选择行。将只收集选择的这些行的统计信息。类似地，SYSTEM 方法对页级别的数据进行采样。因此，选择每一页的概率为 P，不选择每一页的概率为 1 - P/100。

页级别采样的性能很好，原因是对选择的每一页只需要执行一次 I/O。对于行级别采样，I/O 成本没有降低，原因是在整个表扫描过程中将扫描每个表页。但是，行级别采样还是得到了很大改进（尽管 I/O 次数没有减少），这是因为收集统计信息是很消耗 CPU 的。

在数据值的集群程度很高的情况下，行级别采样将比页级别采样提供更好的样本。与页采样相比，行级别样本集可能将更好地反映数据，原因是它将包括每个数据页中 P% 的行数。在页级别采样中，P% 的页数的所有行都将在样本集中。如果这些行是随机分布在表中的，那么对行采样获得的统计信息的准确性与对页采样获得的统计信息的准确性差不多。

除非使用了 REPEATABLE 选项，否则每个样本都是在运行 RUNSTATS 命令时随机生成的。在使用 REPEATABLE 子句的情况下，将生成一个与最后一次执行带有 TABLESAMPLE 选项的 RUNSTATS 命令时生成的样本相同的样本。在想要为具有常量数据的表生成完全相同的统计信息的情况下，用户会发现此功能是很有用的。

使用统计信息概要文件收集统计信息

RUNSTATS 实用程序提供了一个选项来注册并使用统计信息概要文件，该文件是一组选项，这些选项指定要对特定表收集的统计信息，例如，表统计信息、索引统计信息或分布统计信息。

此功能部件简化了统计信息收集，它允许您存储在发出 RUNSTATS 命令时指定的一些选项，从而，可以对表反复收集相同的统计信息而不必重新输入命令选项。

无论实际上是否正在收集统计信息，都可以注册或更新统计信息概要文件。例如，要同时注册概要文件和收集统计信息，可发出带有 SET PROFILE 选项的 RUNSTATS 命令。要在实际上并没有收集统计信息的情况下只注册概要文件，可发出带有 SET PROFILE ONLY 选项的 RUNSTATS 命令。

要使用已经注册的统计信息概要文件来收集统计信息，发出 RUNSTATS 命令，并且只指定表名和 USE PROFILE 选项。

要了解在特定表的统计信息概要文件中当前指定的选项，可以使用以下 SELECT 语句来查询目录表，其中 tablename 是想要获得其概要文件的表的名称:

```
SELECT STATISTICS_PROFILE FROM SYSIBM.SYSTABLES WHERE NAME = tablename
```

自动进行统计信息概要分析

统计信息概要文件还可以由 DB2 自动统计信息概要分析自动生成。当启用了此功能时，会收集有关数据库活动的信息，并将这些信息存储在查询反馈仓库中。可根据此数据来生成统计信息概要文件。启用此功能可以缓解统计信息与特定工作负载相关的不确定性问题，并可以收集最少的统计信息集来提供最佳的数据库工作负载性能。

可以将此功能与自动收集统计信息功能配合使用，后一项功能会根据自动生成的统计信息概要文件中包含的信息来自动安排统计信息维护。

要启用此功能，需要已经通过设置适当的配置参数启用了表的自动维护。AUTO_STATS_PROF 配置参数将激活查询反馈数据的收集，而 AUTO_PROF_UPD 配置参数将激活生成统计信息概要文件以供自动收集统计信息使用。

注: 自动生成统计信息概要文件只能以 DB2 串行方式激活，而对于某些联合环境、多分区 MPP 环境以及启用了分区内并行性的环境中的查询将禁用此功能。

生成统计信息概要文件最适用于具有下列特征的环境: 运行使用许多谓词的大而复杂的查询、谓词列的数据中经常具有关联以及对几个表进行连接和分组。它不太适用于主要是具有事务性工作负载的环境。

还可以通过其他一些方法来使用此功能:

- 在测试环境中。在测试系统中，将 AUTO_STATS_PROF 和 AUTO_PROF_UPD 设置为 ON，在该环境中可以很容易地接受运行时监视的性能开销。当测试系统使用真实的数据和查询时，这将允许了解 RUNSTATS 的统计信息参数的正确关联和设置，然后将把这些关联和设置存储在统计信息概要文件中。可以再将这些概要文件传送至生产系统，生产系统对查询有利而不会导致任何监视开销。
- 解决生产环境中的特定查询的性能问题。如果检测到一组特定查询的性能问题，并且这些问题可能是由于错误的统计信息或关联造成的，那么可以打开 AUTO_STATS_PROF 并执行某个时间段的目标工作负载。自动进行的统计信息概要分析将分析查询反馈并在 SYSTOOLS.OPT_FEEDBACK_RANKING* 表中创建建议。可以检查这些建议并按照建议来手动优化统计信息概要文件。要让 DB2 按照这些建议自动更新统计信息概要文件，应在打开 AUTO_STATS_PROF 时打开 AUTO_PROF_UPD。

注: 监视查询和将查询反馈数据存储在反馈仓库中将产生一定的性能开销。

创建查询反馈仓库

反馈仓库由采用 SYSTOOLS 模式的五个表组成，它们存储有关在执行查询期间遇到的谓词的信息以及有关统计信息收集的建议。这五个表是 OPT_FEEDBACK_QUERY, OPT_FEEDBACK_PREDICATE, OPT_FEEDBACK_PREDICATE_COLUMN, OPT_FEEDBACK_RANKING, OPT_FEEDBACK_RANKING_COLUMN。

要使用自动进行的统计信息概要分析，需要首先使用 SYSINSTALLOBJECTS 存储过程来创建查询反馈仓库。此存储过程是以 SYSTOOLS 模式创建和删除对象的常用存储过程。

调用 SYSINSTALLOBJECTS 存储过程，如下所示:

```
call SYSINSTALLOBJECTS ( toolname, action, tablespacename, schemaname)
```

其中:

toolname
: 指定要创建或删除其对象的工具的名称。在此示例中为“ASP”或“AUTO STATS PROFILING”。

action 指定要执行的操作: “C”表示创建，而“D”表示删除。

tablespacename
: 将创建反馈仓库表的表空间的名称。此输入参数是可选的。如果未指定此参数，那么将使用缺省用户空间。

schemaname

创建或删除对象时所使用的模式的名称。当前未使用此参数。

例如，要在表空间“A”中创建反馈仓库，输入：`call SYSINSTALLOBJECTS ('ASP', 'C', 'A', '')`

目录统计信息表

下列表提供关于包含目录统计信息的系统目录表和收集特定统计信息的 RUNSTATS 选项的信息。

表 53. 表统计信息（SYSCAT.TABLES 和 SYSSTAT.TABLES）

统计信息	描述	RUNSTATS 选项	
		表	索引
FPAGES	一个表所用的页数	是	是
NPAGES	包含行的页数	是	是
OVERFLOW	溢出的行数	是	否
CARD	表中的行数（基数）	是	是（注 1）
ACTIVE_BLOCKS	对于 MDC 表，已占用块的总数	是	否

注:

1. 如果该表未定义索引，而您请求了索引的统计信息，那么不更新新的 CARD 统计信息。而保留先前的 CARD 统计信息。

表 54. 列统计信息（SYSCAT.COLUMNS 和 SYSSTAT.COLUMNS）

统计信息	描述	RUNSTATS 选项	
		表	索引
COLCARD	列基数	是	是（注 1）
AVGCOLLEN	列的平均长度	是	是（注 1）
HIGH2KEY	列中的第二个最大值	是	是（注 1）
LOW2KEY	列中的第二个最小值	是	是（注 1）
NUMNULLS	列中的空值数	是	是（注 1）
SUB_COUNT	子元素的平均数	是	否（注 2）
SUB_DELIM_LENGTH	分隔每个子元素的每个定界符的平均值长度	是	否（注 2）

注:

1. 为索引键中的第一列收集列统计信息。
2. 这些统计信息提供关于列中数据的信息，这些列包含一系列用空格定界的子字段或子元素。只对类型为 CHAR、VARCHAR、GRAPHIC 和 VARGRAPHIC 的单字节字符集字符串列收集 SUB_COUNT 和 SUB_DELIM_LENGTH 统计信息。

表 55. 多列统计信息（SYSCAT.COLGROUPS 和 SYSSTAT.COLGROUPS）

统计信息	描述	RUNSTATS 选项	
		表	索引
COLGROUPCARD	列组的基数	是	否

注: RUNSTATS 不收集以下两个表中列示的多列分布统计信息。不能手动更新这些统计信息。

表 56. 多列分布统计信息（SYSCAT.COLGROUPDIST 和 SYSSTAT.COLGROUPDIST）

统计信息	描述	RUNSTATS 选项	
		表	索引
TYPE	F = 频率值 Q = 分位数值	是	否
ORDINAL	组中列的序数	是	否
SEQNO	序号 *n*，它表示第 *n* 个 TYPE 值	是	否
COLVALUE	作为字符文字的数据值或空值	是	否

表 57. 多列分布统计信息 2（SYSCAT.COLGROUPDISTCOUNTS 和 SYSSTAT.COLGROUPDISTCOUNTS）

统计信息	描述	RUNSTATS 选项	
		表	索引
TYPE	F = 频率值 Q = 分位数值	是	否
SEQNO	序号 *n*，它表示第 *n* 个 TYPE 值	是	否
VALCOUNT	如果 TYPE = F，那么 VALCOUNT 是由 SEQNO 标识的列组的 COLVALUE 的发生次数。 如果 TYPE = Q，那么 VALCOUNT 是其值小于或等于具有此 SEQNO 的列组的 COLVALUE 的行数。	是	否
DISTCOUNT	如果 TYPE = Q，那么此列包含小于或等于具有此 SEQNO 列组的 COLVALUE 的单值数目。如果不可用，那么为空。	是	否

表 58. 索引统计信息（SYSCAT.INDEXES 和 SYSSTAT.INDEXES）

统计信息	描述	RUNSTATS 选项	
		表	索引
NLEAF	索引叶子页数	否	是
NLEVELS	索引层数	否	是
CLUSTERRATIO	表数据的集群度	否	是（注 2）
CLUSTERFACTOR	更精确的集群度	否	详细的（注 1 和 2）

表 58. 索引统计信息（SYSCAT.INDEXES 和 SYSSTAT.INDEXES）(续)

统计信息	描述	RUNSTATS 选项	
		表	索引
DENSITY	SEQUENTIAL_ PAGES 与该索引占用的页范围中页数的比率（百分比）（注 3）	否	是
FIRSTKEYCARD	在索引的第一列中的单值数	否	是
FIRST2KEYCARD	在索引的前两列中的单值数	否	是
FIRST3KEYCARD	在索引的前三列中的单值数	否	是
FIRST4KEYCARD	在索引的前四列中的单值数	否	是
FULLKEYCARD	在索引的所有列中的单值数，不包括其所有 RID 标记为删除的 2 类索引中的任何键值	否	是
PAGE_FETCH_PAIRS	不同缓冲区大小的页访存估计值	否	详细的（注 1 和 2）
AVGPARTITION_CLUSTERRATIO	单个数据分区内数据的集群度	否	是（注 2）
AVGPARTITION_CLUSTERFACTOR	单个数据分区内集群度的更精确度量。	否	详细的（注 1 和 2）
AVGPARTITION_PAGE_FETCH_PAIRS	根据单个数据分区生成的不同缓冲区大小的页访存估计值	否	详细的（注 1 和 2）
DATAPARTITION_CLUSTERFACTOR	扫描索引期间数据分区引用的数目	否（注 6）	是（注 6）
SEQUENTIAL_PAGES	按索引键顺序位于磁盘上的叶子页的数目，在这些叶子页之间很少有或没有大的间隔	否	是
AVERAGE_SEQUENCE_PAGES	可按顺序访问的平均索引页数。这是预取程序可以像按顺序的那样检测的索引页数	否	是
AVERAGE_RANDOM_PAGES	在顺序页访问中的平均随机索引页数	否	是
AVERAGE_SEQUENCE_GAP	序列之间的间隔	否	是
AVERAGE_SEQUENCE_FETCH_PAGES	可按顺序访问的平均表页数。这是预取程序在使用索引访存表行时可以像按顺序的那样检测的表页数	否	是（注 4）
AVERAGE_RANDOM_FETCH_PAGES	当使用索引访存表行时在顺序页访问之间的平均随机表页数	否	是（注 4）
AVERAGE_SEQUENCE_FETCH_GAP	当使用索引访存表行时序列之间的间隔	否	是（注 4）
NUMRIDS	索引中的记录标识（RID）数，包括 2 类索引中已删除的 RID。	否	是
NUMRIDS_DELETED	索引中标记为删除的 RID 的总数，其上所有记录标识标记为删除的叶子页上的 RID 除外	否	是
NUM_EMPTY_LEAFS	索引中其上所有记录标识标记为删除的叶子页的总数	否	是
INDCARD	索引条目的数目（索引基数）	否	是

表 58. 索引统计信息（SYSCAT.INDEXES 和 SYSSTAT.INDEXES）（续）

统计信息	描述	RUNSTATS 选项	
		表	索引

注:

1. 通过在 RUNSTATS 命令上指定 DETAILED 子句来收集详细的索引统计信息。
2. 除非该表很大，否则，不使用 DETAILED 子句收集 CLUSTERFACTOR 和 PAGE_FETCH_PAIRS。如果该表超过 25 页，那么收集 CLUSTERFACTOR 和 PAGE_FETCH_PAIRS 统计信息。在这种情况下，CLUSTERRATIO 是 -1（不收集）。如果该表是一个相当小的表，那么 RUNSTATS 只填写 CLUSTERRATIO，而不填写 CLUSTERFACTOR 和 PAGE_FETCH_PAIRS。如果未指定 DETAILED 子句，那么只收集 CLUSTERRATIO 统计信息。
3. 此统计信息测量在包含属于该表的索引的文件中页数的百分比。对于只定义了一个索引的表，DENSITY 通常应为 100。优化器使用 DENSITY 来估计如果预取了索引页，那么平均可能从其他索引读取多少个不相关的页。
4. 当此表位于 DMS 表空间时，不能计算这些统计信息。
5. 即使调用命令时指定要收集统计信息，在 LOAD 或 CREATE INDEX 期间也不会收集预取统计信息。如果关闭了“顺序检测标志”配置参数（seqdetect），那么也不收集预取统计信息。
6. 如果表的 RUNSTATS 选项为“否”，表示收集表统计信息时不收集统计信息，如果索引的 RUNSTATS 选项为“是”，表示使用带 INDEXES 选项的 RUNSTATS 命令时收集统计信息。

表 59. 列分布统计信息（SYSCAT.COLDIST 和 SYSSTAT.COLDIST）

统计信息	描述	RUNSTATS 选项	
		表	索引
DISTCOUNT	如果 TYPE 是 Q，那么是小于或等于 COLVALUE 统计信息的单值数目	分布（注 2）	否
TYPE	有关行是提供高频值还是分位数统计信息的指示符	分布	否
SEQNO	按频率排列的序号，可帮助唯一标识该表中的行	分布	否
COLVALUE	要收集其频率或分位数统计信息的数据值	分布	否
VALCOUNT	数据值在列中出现的频率，对于分位数，它是小于或等于数据值（COLVALUE）的值的数目	分布	否

注:

1. 通过在 RUNSTATS 命令上指定 WITH DISTRIBUTION 子句来收集列分布统计信息。注意，除非列值严重不均匀，否则，可能不收集分布统计信息。
2. 仅对索引中的第一个键列收集 DISTCOUNT。

分布统计信息

可以收集两种数据分布统计信息:

- 频率统计信息

  这些统计信息提供关于 *num_freqvalues* 数据库配置参数的值所指定级别的具有最高重复的数目和次高重复值的数目等的列和数据值的信息。要禁用收集高频值统计信息，将 *num_freqvalues* 设置为 0。

还可以将 *num_freqvalues* 设置为每个表、统计信息视图和特定列的 RUNSTATS 选项。

- 分位数统计信息

  这些统计信息提供关于与其他值相比如何分布数据值的信息。称为 K 分位数，这些统计信息表示值 V，至少 K 个值位于该值或该值以下。可以通过按升序排序值来计算 K 分位数。K 分位数值是从范围的低端起第 K 个位置中的值。

  要指定应该将列数据值分组成的部分数，将 *num_quantiles* 数据库配置参数设置为 2 和 32,767 之间的某个值。缺省值为 20，它确保对任何相等或小于或大于谓词的优化器估计误差最大为正或负 2.5%，而对任何 BETWEEN 谓词的最大误差为正或负 5%。要禁用分位数统计信息收集，将 num_quantiles 设置为 0 或 1。

  可以对每个表或统计信息视图以及特定列设置 *num_quantiles*。

注: 如果指定较大的 *num_freqvalues* 和 *num_quantiles* 值，那么执行 RUNSTATS 时将需要更多 CPU 资源和内存，通过 *stat_heap_sz* 数据库配置参数来指定 CPU 资源和内存量。

何时收集分布统计信息要决定是否应创建和更新给定表或统计信息视图的分布统计信息，考虑以下两个因素:

- 应用程序是否使用静态或动态 SQL 和 XQuery 语句。

  分布统计信息对不使用主变量的动态查询和静态查询最有用。当使用具有主变量的查询时，优化器只能有限地利用分布统计信息。

- 列中的数据是否是均匀分布的。

  如果该表中至少有一列的数据分布非常“不均匀”，且该列频繁出现在等式或范围谓词中；即，类似如下所示的子句中，那么创建分布统计信息:

```
WHERE C1 = KEY;
   WHERE C1 IN (KEY1, KEY2, KEY3);
   WHERE (C1 = KEY1) OR (C1 = KEY2) OR (C1 = KEY3);
   WHERE C1 <= KEY;
   WHERE C1 BETWEEN KEY1 AND KEY2;
```

可能发生两种类型的不均匀数据分布（可能一起发生）:

- 可能按一个或多个子间隔集群数据，而不是均匀地在最高和最低数据值之间分布。考虑以下列，其中，数据在范围（5,10）内集群:

C1
0.0
5.1
6.3
7.1
8.2
8.4
8.5
9.1
93.6
100.0

分位数统计信息帮助优化器处理这种数据分布。

要帮助确定是否不是均匀分布的列数据，执行诸如以下示例的查询:

```
SELECT C1, COUNT(*) AS OCCURRENCES
          FROM T1
   GROUP BY C1
   ORDER BY OCCURRENCES DESC;
```

- 重复数据值可能经常出现。考虑使用下列频率分布数据的列:

数据值	频率
20	5
30	10
40	10
50	25
60	25
70	20
80	5

要帮助优化器处理重复值，创建分位数和高频值统计信息。

何时仅收集索引统计信息

在下列情况下，您可以只根据索引数据来收集统计信息:

- 自运行 RUNSTATS 实用程序以来已创建了新索引，且您不想再次收集有关该表数据的统计信息。
- 存在影响索引的第一列的许多数据更改。

要指定哪个级别的统计精度

要确定存储分布统计信息使用的精度，指定数据库配置参数 *num_quantiles* 和 *num_freqvalues*。还可以指定这些参数作为收集表或列的统计信息时的 RUNSTATS 选项。这些值设置得越高，RUNSTATS 创建和更新分布统计信息时使用的精度越大。但是，精度越大，在 RUNSTATS 执行期间和在目录表中所需要的存储器中需要使用的资源就越多。

对于大多数数据库，为 *num_freqvalues* 数据库配置参数指定 10 到 100 之间的数。理论下，应创建高频值统计信息，以便其余值的频率可近似等于最高频值的频率，或者相比之下忽略不计。数据库管理器收集的数目可能低于此数，因为只对出现多次的数据值收集这些统计信息。如果需要只收集分位数统计信息，那么将 *num_freqvalues* 设置为 0。

要设置分位数的数目，指定 20 到 50 之间作为 *num_quantiles* 数据库配置参数的设置。确定分位数数目的经验方法为:

- 确定在估计任何范围查询的行数时可允许的最大错误百分比，即 P
- 如果谓词是 BETWEEN 谓词，那么分位数数目应近似为 100/P，如果谓词是任何其他类型的范围谓词（<、<=、> 或 >=），那么分位数数目应近似为 50/P。

例如，25 个分位数导致的最大估计错误对于 BETWEEN 谓词应为 4%，而对于“>”谓词则应为 2%。通常，至少指定 10 个分位数。对于极端不均匀的数据才需要 50 个以上的分位数。如果只需要高频值统计信息，那么将 *num_quantiles* 设置为 0。如果将此参数设置为“1”，那么因为值的整个范围适合一个分位数，因此不收集分位数统计信息。

优化器如何使用分布统计信息

优化器使用分布统计信息，以便更好地估计各种可能的访问方案的成本来满足查询。

如果您不执行带有 WITH DISTRIBUTION 子句的 RUNSTATS，那么目录统计信息表仅包含以下信息：表或统计信息视图的大小、表或统计信息视图中的最高和最低值、表对于任何其索引的集群度以及索引列中的单值的数目。

除非具有关于低和高值之间的值分布的更多信息，否则，优化器假定数据值是均匀分布的。如果数据值彼此差异较大，在范围中的某些部分中集群，或包含许多重复值，那么优化器将选择次于最优的访问方案。

请考虑以下示例:

优化器需要估计包含满足等式或范围谓词的列值的行数以选择成本最低的访问方案。估计越准确，优化器将选择最优访问方案的可能性就越大。例如，考虑如下查询

```
SELECT C1, C2
    FROM TABLE1
    WHERE C1 = 'NEW YORK'
    AND C2 <= 10
```

假定 C1 和 C2 上都有一个索引。一个可能的访问方案是使用 C1 上的索引来检索 C1 = 'NEW YORK' 的所有行，然后检查每个检索的行以了解是否 C2 <= 10。一个替代的访问方案是使用 C2 上的索引来检索 C2 <= 10 的所有行，然后检查每个检索的行以了解是否 C1 = 'NEW YORK'。因为执行查询的主要成本通常是检索行的成本，所以最好的方案是需要最少检索的方案。选择此方案需要估计满足每个谓词的行数。

当已对表或统计信息视图执行 RUNSTATS 但分布统计信息仍不可用时，那么对优化器只有列的以下信息可用：第二个最高的数据值（HIGH2KEY）、第二个最低的数据值（LOW2KEY）、单值的数目（COLCARD）和行数。然后假定一列中的数据值的频率全部相等，而且数据值均匀地分布在整个间隔（LOW2KEY, HIGH2KEY）中，以此为前提估计满足等式或范围谓词的行的数目。具体地说，满足等式谓词 C1 = KEY 的行数估计为 CARD/COLCARD，而满足范围谓词 C1 BETWEEN KEY1 AND KEY2 的行数估计为:

$$\frac{\text{KEY2 - KEY1}}{\text{HIGH2KEY - LOW2KEY}} \times \text{CARD} \qquad (1)$$

仅当一列中数据值的真实分布相当均匀时，这些估计才准确。如果分布统计信息不可用，且数据值的频率彼此之间相差很大，或数据值集群在间隔（LOW_KEY,HIGH_KEY）中的几个子间隔内，那么估计可能偏差很大，且优化器选择的访问方案可能不是最优的。

当分布统计信息可用时，以上描述的错误可通过以下方法来大大减少：使用高频值统计信息来计算满足等式谓词的行数，使用高频值统计信息和分位数来计算满足范围谓词的行数。

分布统计信息使用的扩展示例

要了解优化器可以如何使用分布统计信息，首先考虑包含格式 C1 = KEY 的等式谓词的查询。

高频值统计信息示例

如果高频值统计信息是可用的，优化器可以使用这些统计信息来选择适当的访问方案，如下所示:

- 如果 KEY 是 N 个最高频值的其中一个，那么优化器使用存储在该目录中的 KEY 的频率。
- 如果 KEY 不是 N 个最高频值的其中一个，那么优化器在假定（COLCARD - N）个非高频值具有均匀分布的前提下估计满足该谓词的行数。即，估计行数为:

```
CARD - NUM_FREQ_ROWS
   --------------------          (2)
         COLCARD - N
```

其中，CARD 是表中的行数，COLCARD 是列的基数，NUM_FREQ_ROWS 是具有 N 个最高频值的其中一个最高频值的总行数。

例如，考虑具有如下数据值和频率的一列（C1）:

数据值	频率
1	2
2	3
3	40
4	4
5	1

如果仅基于最高频值（即，N = 1）的高频值统计信息可用，那么对于此列，表中的行数为 50，列基数为 5。对于谓词 C1 = 3，刚好有 40 行满足。如果优化器假定数据均匀分布，那么它估计满足该谓词的行数为 50/5 = 10，错误率为 -75%。如果优化器可以使用高频值统计信息，那么该行数估计为 40，且没有错误。

考虑另一个示例，其中有 2 行满足谓词 C1 = 1。在没有高频值统计信息的情况下，满足谓词的行数估计为 10，错误率为 400%。您可使用以下公式来计算估计错误率（百分比）:

```
   估计行数 - 实际行数
----------------------------  X 100
          实际行数
```

使用高频值统计信息（N = 1），优化器将使用以上给出的公式（2）来估计包含此值的行数，例如:

```
(50 - 40)
   --------- = 3
     (5 - 1)
```

并且错误降低了一个数量级，如下所示:

```
  3 - 2
------- = 50%
     2
```

分位数统计信息示例

分位数统计信息的下列说明使用术语“K 分位数”。一列的 *K 分位数* 是最小的数据值 V，即至少有“K”行具有小于或等于 V 的数据值。要计算 K 分位数，将该列中的所有行按数据值递增的顺序来排序；K 分位数是已排序的列中第 K 行中的数据值。

如果分位数统计信息可用，那么优化器可以更好地估计满足一个范围谓词的行数，如以下示例所示。考虑包含下列值的一列（C）：

C
0.0
5.1
6.3
7.1
8.2
8.4
8.5
9.1
93.6
100.0

并假定对于 K = 1、4、7 和 10，有可用的 K 分位数，如下所示:

K	K 分位数
1	0.0
4	7.1
7	8.5
10	100.0

首先考虑谓词 C <= 8.5。对于以上给出的数据，刚好有 7 行满足此谓词。假定数据分布均匀并使用上面的公式（1），用 LOW2KEY 替换 KEY1，那么满足该谓词的行数估计如下:

```
8.5 - 5.1
  ---------- x 10 *= 0
  93.6 - 5.1
```

其中 *= 表示“几乎相等”。此估计中的错误率近似为 -100%。

如果分位数统计信息可用，那么优化器通过如下方法来估计满足此同一谓词（C <= 8.5）的行数: 查找作为其中一个分位数的最高值的 8.5，并通过使用 K 的对应值 7 来作为估计行数。在此情况下，错误缩小为 0。

现在考虑谓词 C <= 10。刚好有 8 行满足此谓词。如果优化器必须假定数据分布均匀并使用公式（1），那么满足该谓词的行数估计为 1，且错误率为 -87.5%。

与上一个示例不同，值 10 不是存储的其中一个 K 分位数。但是，优化器可以使用分位数来估计满足该谓词的行数为 r_1 + r_2，其中，r_1 是满足谓词 C <= 8.5 的行数，而 r_2 是满足谓词 C > 8.5 AND C <= 10 的行数。在以上示例中，r_1 = 7。要估计 r_2，优化器使用线性插值:

```
                  10 - 8.5
r_2 *= ---------- x（具有值 > 8.5 和 <= 100.0 的行数）
                  100 - 8.5
                  10 - 8.5
r_2 *= ---------- x (10 - 7)
                  100 - 8.5
```

```
                1.5
r_2 *= ---- x (3)
               91.5
r_2 *= 0
```

最终估计是 r_1 + r_2 *= 7，而错误率仅为 -12.5%。

在以上示例中分位数提高了估计的准确性，因为实数据值“集群”在范围 5 - 10 之间，但标准估计公式假定数据值均匀地分布在 0 和 100 之间。

当不同数据值的频率有很大差异时，使用分位数也可提高准确性。考虑具有如下数据值和频率的一列:

数据值	频率
20	5
30	5
40	15
50	50
60	15
70	5
80	5

假定对于 K = 5、25、75、95 和 100，有可用的 K 分位数:

K	K 分位数
5	20
25	40
75	50
95	70
100	80

也假定根据 3 个最常用的值可获得高频值统计信息。

考虑谓词 C BETWEEN 20 AND 30。从数据值的分布，您可看到刚好有 10 行满足此谓词。假定数据分布均匀并使用公式（1），那么满足该谓词的行数估计如下:

```
  30 - 20
-------  x 100 = 25
70 - 30
```

它的错误率为 150%。

使用高频值统计信息和分位数，满足该谓词的行数估计为 r_1 + r_2，其中，r_1 是满足谓词（C = 20）的行数，而 r_2 是满足谓词 C > 20 AND C <= 30 的行数。使用公式（2），r_1 估计如下:

```
100 - 80
-------- = 5
  7 - 3
```

使用线性插值，r_2 估计如下:

```
    30 - 20
    ------- x (# 值 > 20 且 <= 40 的行)
    40 - 20
    30 - 20
= ------- x (25 - 5)
    40 - 20
= 10,
```

得到最终估计值 15，并使错误率降低到原来的三分之一。

详细的索引统计信息

如果使用 DETAILED 子句对索引执行 RUNSTATS，那么收集关于索引的统计信息，这允许优化器根据各种缓冲池大小估计将需要多少数据页访存。此附加信息帮助优化器作出通过索引访问表的成本的更好估计。

注: 当收集详细索引统计信息时，RUNSTATS 花费较长时间，并需要更多内存和 CPU 处理。SAMPLED DETAILED 选项（对于此信息仅计算统计上有意义的条目数）需要 2MB 统计信息堆。将附加的 488 个 4K 页分配给为此附加内存需求设置的 *stat_heap_sz* 数据库配置参数。如果堆看起来太小，那么 RUNSTATS 在尝试收集统计信息之前返回一条错误。

仅当表有足够的大小（大约 25 页）时，才收集 DETAILED 统计信息 PAGE_FETCH_PAIRS 和 CLUSTERFACTOR。在此示例中，CLUSTERFACTOR 将是在 0 和 1 之间的一个值；而 CLUSTERRATIO 将是 -1（不收集）。对于少于 25 页的表，即使为该表上的索引指定了 DETAILED 子句，CLUSTERFACTOR 仍将是 -1（不收集），而 CLUSTERRATIO 仍将是 0 和 100 之间的一个值。

DETAILED 统计信息提供关于在不同的缓冲区大小下执行一个完整的索引扫描时，访问一个表的数据页所需要的物理 I/O 的数目的简明信息。当 RUNSTATS 扫描该索引的页时，它将为不同的缓冲区大小建立模型，并收集缺页故障发生频率的估计值。例如，如果只有一个缓冲区页可用，那么索引引用的每个新页导致缺页故障。在更坏的情况下，每行可能引用不同的页，导致最多与已索引的表中的行相同的 I/O 数。另一种极端情况是，当该缓冲区大得足以容纳整个表（但不超过最大缓冲区大小）时，所有表的每页都读取一次。结果，物理 I/O 的数目是缓冲区大小的单调非递增函数。

统计信息还提供更精确估计表行对于索引顺序的集群度。涉及索引的集群的表行越少，通过索引访问表行所需要的 I/O 就越多。优化器在估计通过索引访问表的成本的同时考虑缓冲区大小和集群度。

当查询引用不包括在索引中的列时，应收集 DETAILED 索引统计信息。此外，在下列情况时，也应使用 DETAILED 索引统计信息:

- 表有多个具有不同集群度的非集群索引
- 各键值的集群度不均匀
- 索引中的值被不均匀地更新

如果没有先前的了解或不强制不同缓冲区大小下的索引扫描并接着监视引起的物理 I/O，那么很难估计这些情况。确定其中任何一个情况是否发生的成本最低的方法可能是，收集一个索引的 DETAILED 统计信息并检查这些统计信息。如果得到的 PAGE_FETCH_PAIRS 是非线性的，那么保留这些统计信息。

子元素统计信息

如果包含由空格分开的子字段或子元素的列包含在表中，并且查询在 WHERE 子句中引用这些列，那么应该收集子元素统计信息以确保最好的访问方案。

例如，假设数据库包含表 DOCUMENTS，表中每行描述一个文档，并假设在 DOCUMENTS 中有一个称为 KEYWORDS 的列，该列中包含为了文本检索目的而与该文档有关的关键字的列表。在 KEYWORDS 中的值可能如下所示:

```
'database simulation analytical business intelligence'
'simulation model fruit fly reproduction temperature'
'forestry spruce soil erosion rainfall'
'forest temperature soil precipitation fire'
```

在本示例中，每列值由 5 个子元素构成，其中每个都是一个单词（关键字），且由一个空格与其他的词分开。

用于查询时，使用 %match_all character 在这些列上指定 LIKE 谓词:

```
SELECT .... FROM DOCUMENTS WHERE KEYWORDS LIKE '%simulation%'
```

它经常有利于优化器了解某些关于列的子元素结构的基本统计信息。

当执行带有 LIKE STATISTICS 子句的 RUNSTATS 时，将收集下列统计信息:

SUB_COUNT
子元素的平均数。

SUB_DELIM_LENGTH
分隔每个子元素的每个定界符的平均长度，其中，在本上下文中，定界符是一个或多个连续的空白字符。

在 KEYWORDS 列示例中，SUB_COUNT 是 5，而 SUB_DELIM_LENGTH 是 1，因为每个定界符是一个单一的空白字符。

DB2_LIKE_VARCHAR 注册表变量影响优化器处理以下格式谓词的方式:

```
COLUMN LIKE '%xxxxxx'
```

其中 *xxxxxx* 代表任何字符串；即，搜索值从 % 字符开始的任何 LIKE 谓词。（它可能以 % 字符结束或可能不以该字符结束）。将这些称为“通配符 LIKE 谓词”。对于所有谓词，优化器必须估计有多少行与该谓词匹配。对于通配符 LIKE 谓词，优化器假设要匹配的 COLUMN 包含并置在一起的一系列元素，并且它基于字符串长度估计每个元素的长度，不包括前导和结尾 % 字符。

要检查子元素统计信息的值，查询 SYSIBM.SYSCOLUMNS。例如:

```
select substr(NAME,1,16), SUB_COUNT, SUB_DELIM_LENGTH
         from sysibm.syscolumns where tbname = 'DOCUMENTS'
```

注: 如果使用 LIKE STATISTICS 子句，RUNSTATS 可能花较长时间。例如，如果未使用 DETAILED 和 DISTRIBUTION 选项，那么 RUNSTATS 在具有 5 个字符列的表上可能运行 15% 到 40% 以及更长时间。如果指定 DETAILED 或 DISTRIBUTION 选项，那么即使开销的绝对数量相同，但开销百分比较少。如果您正在考虑使用此选项，那么应该估计对提高查询性能的开销。

用户可更新的目录统计信息

用户定义的函数的统计信息

要创建用户定义的函数（UDF）的统计信息，编辑 SYSSTAT.FUNCTIONS 目录视图。如果 UDF 统计信息可用，那么该优化器在估计各种访问方案的成本时就会使用这些统计信息。RUNSTATS 实用程序不收集 UDF 的统计信息。如果统计信息不可用，那么统计信息列的值是 -1，且优化器使用缺省值，这些缺省值假定一个简单的 UDF。

下表提供有关统计列的信息，您可以为该列提供估计以提高性能:

表 60. 函数统计信息（SYSCAT.ROUTINES 和 SYSSTAT.FUNCTIONS）

统计信息	描述
IOS_PER_INVOC	每次执行一个函数时所执行的读/写请求的估计数目。
INSTS_PER_INVOC	每次执行一个函数时所执行的机器指令的估计数目。
IOS_PER_ARGBYTE	对每个输入自变量字节所执行的读/写请求的估计数目。
INSTS_PER_ARGBYTES	对每个输入自变量字节所执行的机器指令的估计数目。
PERCENT_ARGBYTES	该函数将实际处理的输入自变量字节的估计平均百分比。
INITIAL_IOS	仅在第一次/最后一次调用函数时所执行的读/写请求的估计数目。
INITIAL_INSTS	仅在第一次/最后一次调用函数时所执行的机器指令的估计数目。
CARDINALITY	一个表函数生成的估计行数。

例如，考虑一个 UDF（EU_SHOE），它将美国鞋码转换为等效的欧洲鞋码。（这两个鞋码可以是 UDT。）对于此 UDF，可以按如下所示设置统计信息列:

- INSTS_PER_INVOC: 设置为执行下列操作所需的机器指令的估计数目:
  - 调用 EU_SHOE
  - 初始化输出字符串
  - 返回结果
- INSTS_PER_ARGBYTE: 设置为把输入字符串转换为欧洲鞋码所需的机器指令的估计数目。
- PERCENT_ARGBYTES: 设置为 100，指示要转换整个输入字符串
- INITIAL_INSTS、IOS_PER_INVOC、IOS_PER_ARGBYTE 和 INITIAL_IOS: 每个都设置为 0，因为此 UDF 只执行计算。

PERCENT_ARGBYTES 将由一个不是始终处理整个输入字符串的函数使用。例如，考虑一个 UDF（LOCATE），它接受两个自变量作为输入，并返回第一个自变量在第二个自变量中第一次出现的起始位置。假定第一个自变量的长度小得相对于第二个自变量来说变得不重要，那么平均来说要搜索第二个自变量的 75%。根据此信息，应将 PERCENT_ARGBYTES 设置为 75。以上对平均值 75% 的估计是基于下列的附加假设:

- 当有一半的时间找不到第一个自变量时，将导致搜索整个第二个自变量。
- 第一个自变量在第二个自变量中任何地方出现的可能性相同，这导致当找到第一个自变量时，搜索第二个自变量的一半（平均值）。

可使用 INITIAL_INSTS 或 INITIAL_IOS 来记录仅在第一次或最后一次调用该函数时，所执行的机器指令或读/写请求的估计数目，如，记录设置暂存区区域的成本。

要获得有关一个用户定义的函数使用的 I/O 和指令的信息，您可以使用您的编程语言编译器或可用于您的操作系统的监视工具所提供的输出。

用于建模和假设情况计划的目录统计信息

可以更改系统目录中的统计信息，以便它不反映表和索引的实际状态，但允许您检查对数据库的各种可能的更改以用于规划目的。更新所选择的系统目录统计信息的能力允许您:

- 使用生产系统统计信息为一个开发系统上的查询性能建立模型
- 执行“假设情况”查询性能分析

不要手动更新生产系统的统计信息。如果这样做，那么优化器可能不会为包含动态 SQL 或 XQuery 语句的产品查询选择最佳访问方案。

要求

您必须具有对数据库的显式 DBADM 权限，才能修改表和索引及其组件的统计信息。即，您的用户标识被记录为在 SYSCAT.DBAUTH 表中具有 DBADM 权限。属于一个 DBADM 组，并不会显式提供此权限。DBADM 可以查看所有用户的统计信息，并可以对在 SYSSTAT 模式中定义的视图执行 SQL UPDATE 语句，来更新这些统计列的值。

没有 DBADM 权限的用户只能查看某些行，这些行包含用户具有 CONTROL 特权的对象的统计信息。如果您没有 DBADM 权限，当您具有每个对象的下列特权时，可以更改单个数据库对象的统计信息:

- 对表的显式 CONTROL 特权。也可更新这些表的列和索引的统计信息。
- 对联合数据库系统中的昵称的显式 CONTROL 特权。也可更新这些昵称的列和索引的统计信息。注意，更新仅影响本地元数据（不会更改数据源表统计信息）。这些更新仅影响由 DB2 优化器生成的全局访问策略。
- 用户定义的函数（UDF）的所有权。

下面显示更新 EMPLOYEE 表的表统计信息的一个示例:

```
UPDATE SYSSTAT.TABLES
   SET  CARD   = 10000,
             NPAGES = 1000,
             FPAGES = 1000,
             OVERFLOW = 2
   WHERE TABSCHEMA = 'userid'
       AND TABNAME   = 'EMPLOYEE'
```

当手动更新目录统计信息时，您必须小心。任意更改可以严重改变后续查询的性能。即使在正用于测试或建模的非生产数据库中，您可以使用下列任何一种方法来刷新应用于这些表的更新并使统计信息处于一致状态:

- 对已更改的工作单元执行 ROLLBACK（假定尚未落实该工作单元）。
- 使用 RUNSTATS 实用程序来重新计算并刷新目录统计信息。

- 更新目录统计信息，以指示未收集统计信息。（例如，将列 NPAGES 设置为 -1，指示未收集页数统计信息。）
- 将目录统计信息替换为更改前它们所包含的数据。仅当您在进行任何更改之前使用了 *db2look* 工具来捕获统计信息的情况下，才能使用此方法。

在某些情况下，优化器可能确定某个特定的统计值或统计值的组合无效。它将使用缺省值并发出警告。但是，这类情况很少，因为该验证的大部分工作是在更新统计信息时执行的。

用于对生产数据库建模的统计信息

有时，您可能想要测试系统包含生产系统数据的一个子集。但是，为这类测试系统选择的访问方案不必与在生产系统上应选择的那些访问方案相同，除非更新该测试系统的目录统计信息和配置参数来匹配生产系统的那些参数。

可对生产数据库运行生产工具 *db2look*，以生成使测试数据库的目录统计信息与生产数据库的目录统计信息匹配所需的更新语句。可在模拟方式（`-m` 选项）下使用 *db2look* 来生成这些更新语句。在这种情况下，*db2look* 将生成一个命令处理器脚本，它包含要模拟生产数据库的目录统计信息所需的所有语句。在测试环境中通过 Visual Explain 来分析 SQL 或 XQuery 语句时，它会很有用。

可以使用 *db2look -e* 来提取 DDL 语句，以重新创建数据库数据对象，包括表、视图、索引和数据库中的其他对象。可以对另一个数据库运行用此命令创建的命令处理器脚本，来重新创建该数据库。可以在重新创建数据库和设置统计信息的脚本中一起使用 `-e` 选项和 `-m` 选项。

当对测试系统运行 `db2look` 产生的更新语句后，可使用该测试系统来验证要在生产中生成的访问方案。因为优化器使用表空间的类型和配置来估计 I/O 成本，因此测试系统必须有相同的表空间几何结构或布局。即，同一类型（SMS 或 DMS）的相同数目的容器。

db2look 工具可在 *bin* 子目录中找到。

有关如何使用此生产工具的更多信息，在命令行上输入如下命令:

```
db2look -h
```

“控制中心”还为 *db2look* 实用程序提供了一个名为“生成 DDL”的界面。使用“控制中心”可将实用程序的结果文件集成到“脚本中心”。也可从控制中心调度 *db2look* 命令。使用控制中心与使用 *db2look* 命令的一个不同之处是，在单个调用中前者只能完成一个表分析而后者最多可完成三十个表的分析。还应知道控制中心不支持 LaTex 输出和图形输出。

还可以对 OS/390 或 z/OS 数据库运行 `db2look` 实用程序。`db2look` 实用程序抽取 OS/390 对象的 DDL 和 UPDATE 统计语句。如果想要抽取 OS/390 或 z/OS 对象并在 DB2 数据库 Linux 版、UNIX 版和 Windows 版中重新创建它们，那么此实用程序非常有用。

DB2 数据库 Linux 版、UNIX 版和 Windows 版统计信息与 OS/390 版统计信息之间有一些差别。`db2look` 实用程序执行从 DB2 OS/390 或 z 0S 版 DB2 数据库 Linux 版、UNIX 版和 Windows 版的适当转换（当可以执行转换时），并将不存在其 DB2 OS/390 版中对应统计信息的 DB2 数据库 Linux 版、UNIX 版和 Windows 版统计信息设置为

缺省值（-1）。以下是 db2look 实用程序将 DB2 OS/390 或 z/OS 版统计信息映射至 DB2 数据库 Linux 版、UNIX 版和 Windows 版统计信息的方法。在下面的讨论中，“UDB_x”表示 DB2 数据库 Linux 版、UNIX 版和 Windows 版统计信息列；“S390_x”表示 DB2 OS/390 或 z/OS 版统计信息列。

1. 表层统计信息。

   UDB_CARD = S390_CARDF
   UDB_NPAGES = S390_NPAGES

   没有 S390_FPAGES。但是，DB2 OS/390 或 z/OS 版有另一项名为 PCTPAGES 的统计信息，它表示包含表行的活动表空间页的百分比。因而，有可能根据 S390_NPAGES 和 S390_PCTPAGES 计算 UDB_FPAGES，如下所示:

   ```
   UDB_FPAGES=(S390_NPAGES * 100)/S390_PCTPAGES
   ```

   没有 S390_OVERFLOW 来映射至 UDB_OVERFLOW。因而，db2look 实用程序只是将 UDB_OVERFLOW 设置为缺省值:

   ```
   UDB_OVERFLOW=-1
   ```

2. 列层统计信息。

   UDB_COLCARD = S390_COLCARDF
   UDB_HIGH2KEY = S390_HIGH2KEY
   UDB_LOW2KEY = S390_LOW2KEY

   没有 S390_AVGCOLLEN 来映射至 UDB_AVGCOLLEN，因而，db2look 实用程序只是将 UDB_AVGCOLLEN 设置为缺省值:

   ```
   UDB_AVGCOLLEN=-1
   ```

3. 索引层统计信息。

   UDB_NLEAF = S390_NLEAF
   UDB_NLEVELS = S390_NLEVELS
   UDB_FIRSTKEYCARD= S390_FIRSTKEYCARD
   UDB_FULLKEYCARD = S390_FULLKEYCARD
   UDB_CLUSTERRATIO= S390_CLUSTERRATIO

   其他没有 OS/390 或 z/OS 对应统计信息的统计信息只是设置为缺省值。也就是:

   ```
   UDB_FIRST2KEYCARD = -1
      UDB_FIRST3KEYCARD = -1
      UDB_FIRST4KEYCARD  = -1
      UDB_CLUSTERFACTOR = -1
      UDB_SEQUENTIAL_PAGES = -1
      UDB_DENSITY  = -1
   ```

4. 列分布统计信息。

   在 DB2 OS/390 或 z/OS 版 SYSIBM.SYSCOLUMNS 中，有两种类型的统计信息。类型“F”表示频率值，类型“C”表示基数。仅类型为“F”的条目适用于 DB2 数据库 Linux 版、UNIX 版和 Windows 版，这些就是将被考虑到的条目。

   ```
   UDB_COLVALUE = S390_COLVALUE
   UDB_VALCOUNT = S390_FrequencyF * S390_CARD
   ```

   此外，DB2OS/390 版 SYSIBM.SYSCOLUMNS 中没有 SEQNO 列。因为这是 DB2 所必需的，所以 db2look 自动生成一个 SEQNO 列。

用于手动更新目录统计信息的一般规则

当您更新目录统计信息时，最重要的通用规则是确保各种统计信息的有效值、范围和格式存储在统计信息视图中。保持各种统计信息之间的关系的一致性也很重要。

例如，SYSSTAT.COLUMNS 中的 COLCARD 必须小于 SYSSTAT.TABLES 中的 CARD（一列中的单值数目不能大于行数）。假定要将 COLCARD 从 100 减少到 25，将 CARD 从 200 减少到 50。如果首先更新 SYSSTAT.TABLES，那么会接收到一条错误（因为 CARD 应小于 COLCARD）。正确的顺序是首先更新 SYSSTAT.COLUMNS 中的 COLCARD，然后更新 SYSSTAT.TABLES 中的 CARD。如果要将 COLCARD 从 100 增大到 250，将 CARD 从 200 增大到 300，那么发生相反的情况。在这种情况下，必须首先更新 CARD，然后更新 COLCARD。

当检测到一个已更新的统计信息和另一项统计信息之间存在冲突时，那么发出错误消息。但是，并不是在发生冲突时始终都会发出错误消息。在某些情况下，该冲突难于检测及在错误消息中报告，尤其是两个相关的统计信息在不同目录中时更是如此。鉴于这个原因，您应小心避免导致这类冲突。

在更新目录统计信息前，您应执行的最常见的检查是:

1. 数字统计信息必须是 -1 或大于等于零。
2. 表示百分比的数字统计信息（例如，SYSSTAT.INDEXES 中的 CLUSTERRATIO）必须在 0 和 100 之间。

注: 对于行类型，子表的表级别统计信息 NPAGES、FPAGES 和 OVERFLOW 不可更新。

第一次创建表时，系统目录统计信息设置为 -1 以指示该表没有统计信息。DB2 将对 SQL 或 XQuery 语句编译和优化使用缺省值，直到收集了统计信息为止。如果新值与缺省值不一致，那么更新表或索引统计信息可能失败。因此建议您在创建表之后，在尝试更新表或索引统计信息之前执行 RUNSTATS。

用于手动更新列统计信息的规则

当您更新 SYSSTAT.COLUMNS 中的统计信息时，遵循以下准则。

- 当手动更新 SYSSTAT.COLUMNS 中的 HIGH2KEY 和 LOW2KEY 时，遵循生成的值的行为:
  - HIGH2KEY 和 LOW2KEY 值必须是相应用户列的数据类型的有效值。
  - HIGH2KEY 和 LOW2KEY 值的长度必须是 33 或目标列数据类型的最大长度这两者中的较小者，后者不包括目标列数据类型的附加引号，引号可使字符串的长度达到 68。这意味着在确定 HIGH2KEY 和 LOW2KEY 值时将只考虑相应用户列中的值的前 33 个字符。
  - 存储 HIGH2KEY/LOW2KEY 值之后，可以在 UPDATE 语句的 SET 子句中使用它们，但是进行成本计算时无须考虑这些值。对于字符串，这意味着在字符串的开头和结尾要添加单引号，并且对字符串中已经存在的每个引号另外添加了引号。在下表中，提供了用户列值以及它们在 HIGH2KEY 和 LOW2KEY 中的相应值的示例。

表 61. 数据类型的 HIGH2KEY 和 LOW2KEY 值

用户列中的数据类型	用户数据	相应的 HIGH2KEY 和 LOW2KEY 值
INTEGER	-12	-12
CHAR	abc	'abc'
CHAR	ab'c	'ab''c'

  - 只要相应列中具有 3 个以上的单值，HIGH2KEY 就应该大于 LOW2KEY。
- 列的基数（SYSSTAT.COLUMNS 中的 COLCARD 统计信息）不能大于其对应的表或统计信息视图的基数（SYSSTAT.TABLES 中的 CARD 统计信息）。
- 列中的空值数（SYSSTAT.COLUMNS 中的 NUMNULLS 统计信息）不能大于其对应的表或统计信息视图的基数（SYSSTAT.TABLES 中的 CARD 统计信息）。
- 具有如下数据类型的列不支持统计信息：LONG VARCHAR、LONG VARGRAPHIC、BLOB、CLOB 和 DBCLOB。

用于手动更新分布统计信息的规则

手动更新分布统计信息只用于对生产数据库建模或对手动构造的数据库执行 "假设情况"测试。不更新生产数据库中的分布统计信息。

确保目录中的所有统计信息是一致的。尤其是对于每一列，常用数据统计信息和分位数的目录条目必须满足下列约束:

- 高频值统计信息（在 SYSSTAT.COLDIST 目录中）。这些约束包括:
  - 列 VALCOUNT 中的值必须随 SEQNO 值的递增而不变或降低。
  - 列 COLVALUE 中的值的数目必须小于或等于该列中的单值数目，单值数目存储在目录视图 SYSSTAT.COLUMNS 的列 COLCARD 中。
  - 列 VALCOUNT 中的值的总和必须小于或等于该列中的行数，行数存储在目录视图 SYSSTAT.TABLES 的列 CARD 中。
  - 在大多数情况下，列 COLVALUE 中的值应位于该列的第二个最高和第二个最低的数据值之间，这两个值分别存储在目录视图 SYSSTAT.COLUMNS 的列 HIGH2KEY 和 LOW2KEY 中。可能有一个高频值大于 HIGH2KEY，且有一个高频值小于 LOW2KEY。
- 分位数（在 SYSSTAT.COLDIST 目录中）。这些约束包括:
  - COLVALUE 列中的值必须在 SEQNO 的值增大时不变或降低。
  - VALCOUNT 列中的值必须在 SEQNO 的值增大时增大。
  - COLVALUE 列中的最大值必须在列 VALCOUNT 中有对应的一个条目，它等于该列中的行数。
  - 在大多数情况下，列 COLVALUE 中的值应位于该列的第二个最高和第二个最低的数据值之间，这两个值分别存储在目录视图 SYSSTAT.COLUMNS 的列 HIGH2KEY 和 LOW2KEY 中。

假定具有"R"行的列 C1 的分布统计信息可供使用，而您希望修改该统计信息以对应于具有相同的数据值相对比例、但具有"（F x R）"行的列。要将高频值统计信息上调 F 倍，列 VALCOUNT 中的每一条目必须乘以 F。类似地，要将分位数上调 F 倍，列 VALCOUNT 中的每一条目必须乘以 F。如果不遵循这些规则，那么优化器在您运行该查询时可能使用错误的过滤因子并导致不可预测的性能。

用于手动更新表和昵称统计信息的规则

在 SYSSTAT.TABLES 中您可以更新的仅有的统计值为: CARD、FPAGES、NPAGES 和 OVERFLOW，而对于 MDC 表，该值为 ACTIVE_BLOCKS。注意:

1. CARD 必须大于或等于对应于该表的 SYSSTAT.COLUMNS 中的所有 COLCARD 值。
2. CARD 必须大于 NPAGES。
3. FPAGES 必须大于 NPAGES。
4. NPAGES 必须小于或等于任何索引的 PAGE_FETCH_PAIRS 列中的任何“访存”值（假定此统计信息与该索引相关）。
5. CARD 不能小于或等于任何索引的 PAGE_FETCH_PAIRS 列中的任何“访存”值（假定此统计信息与该索引相关）。

当在联合数据库系统中工作时，在手动提供或更新有关远程视图的昵称统计信息时务必小心。统计信息（例如，此昵称将返回的行数）可能不能反映评估此远程视图的实际成本，这样就可能误导 DB2 优化器。可从统计信息更新受益的情况包括在某单个基本表上定义的远程视图，且未在 SELECT 列表上应用任何列函数。复杂的视图可能需要一个复杂的调整过程，该过程可能要求调整每个查询。考虑创建基于昵称的局部视图，以便 DB2 优化器知道如何更准确地推导出该视图的成本。

用于手动更新索引统计信息的规则

当您更新 SYSSTAT.INDEXES 中的统计信息时，遵循下面描述的规则:

1. PAGE_FETCH_PAIRS（在SYSSTAT.INDEXES 中）必须遵守下列规则:
   - PAGE_FETCH_PAIRS 统计信息中的个别值必须由一系列空白定界符分开。
   - PAGE_FETCH_PAIRS 统计信息中的个别值不能超过 10 位，且必须小于最大整数值（MAXINT = 2147483647）。
   - 如果 CLUSTERFACTOR 大于零，必须始终有一个有效的 PAGE_FETCH_PAIRS 值。
   - 在单个的 PAGE_FETCH_PAIR 统计信息中必须刚好有 11 对数。
   - PAGE_FETCH_PAIRS 的缓冲区大小项的值必须呈升序。
   - 在 PAGE_FETCH_PAIRS 条目中的任何缓冲区大小值不能大于 MIN(NPAGES, 524287)（对于 32 位操作系统）或 MIN(NPAGES, 2147483647)（对于 64 位操作系统），其中 NPAGES 是相应表（在 SYSSTAT.TABLES 中）中的页数。
   - PAGE_FETCH_PAIRS 的“访存”条目的值必须呈降序，且任何个别的“访存”条目不能小于 NPAGES。PAGE_FETCH_PAIRS 条目中的“访存”大小值不能大于对应表的 CARD（基数）统计信息。
   - 如果缓冲区大小值在两个连续的对中相同，那么页读取值也必须在两个对中相同（在 SYSSTAT.TABLES 中）。

   一个有效的 PAGE_FETCH_UPDATE 是:

   ```
   PAGE_FETCH_PAIRS =
         '100 380 120 360 140 340 160 330 180 320 200 310 220 305 240 300
          260 300 280 300 300 300'
   ```

   其中

```
NPAGES = 300
    CARD   = 10000
    CLUSTERRATIO  =  -1
    CLUSTERFACTOR = 0.9
```

2. CLUSTERRATIO 和 CLUSTERFACTOR（在 SYSSTAT.INDEXES 中）必须遵守下列规则:
   - CLUSTERRATIO 的有效值是 -1 或在 0 和 100 之间。
   - CLUSTERFACTOR 的有效值是 -1 或在 0 和 1 之间。
   - CLUSTERRATIO 和 CLUSTERFACTOR 值中至少一个必须始终是 -1。
   - 如果 CLUSTERFACTOR 为正数，那么它必须伴有一个有效的 PAGE_FETCH_PAIR 统计信息。
3. 对于关系索引，下列规则适用于 FIRSTKEYCARD、FIRST2KEYCARD、FIRST3KEYCARD、FIRST4KEYCARD、FULLKEYCARD 和 INDCARD:
   - 对于单列索引，FIRSTKEYCARD 必须等于 FULLKEYCARD。
   - 对于对应的列，FIRSTKEYCARD 必须等于 COLCARD（在 SYSSTAT.COLUMNS 中）。
   - 如果其中任何一个索引统计信息是无关的信息，应将它们设置为 -1。例如，如果您有一个只有 3 列的索引，那么将 FIRST4KEYCARD 设置为 -1。
   - 对于多列索引，如果所有统计信息是相关的，那么它们之间的关系必须是:

     ```
     FIRSTKEYCARD <= FIRST2KEYCARD <= FIRST3KEYCARD <= FIRST4KEYCARD
         <= FULLKEYCARD <= INDCARD == CARD
     ```
4. 对于基于 XML 数据的索引，下列规则适用于 FIRSTKEYCARD、FIRST2KEYCARD、FIRST3KEYCARD、FIRST4KEYCARD、FULLKEYCARD 和 INDCARD:
   - 它们之间的关系必须为:

     ```
     FIRSTKEYCARD <= FIRST2KEYCARD <= FIRST3KEYCARD <= FIRST4KEYCARD
         <= FULLKEYCARD <= INDCARD
     ```
5. 下列规则适用于 SEQUENTIAL_PAGES 和 DENSITY:
   - SEQUENTIAL_PAGES 的有效值是 -1 或在 0 和 NLEAF 之间。
   - DENSITY 的有效值是 -1 或在 0 和 100 之间。

第 21 章 例程

存储过程准则

存储过程允许调用远程数据库，以便在包含许多重复情况的数据库应用程序环境中执行预先编写的过程。例如，检索一组固定数据、对数据库执行一组相同的多个请求或者返回一组固定数据，这些都可能需要多次访问该数据库。

对远程数据库处理单个 SQL 语句需要发送两个传输: 一个请求和一个接收。由于应用程序包含许多 SQL 语句，因此，它需要发送许多传输才能完成其工作。

但是，当 IBM 数据服务器客户机使用封装了许多 SQL 语句的存储过程时，整个过程中它只需要发送两个传输。

存储过程通常在与数据库代理进程分开的进程中运行。这种分开要求存储过程与代理进程通过路由器进行通信。但是，一种在代理进程中运行的特殊存储过程可以提高性能，尽管它很可能损坏数据和数据库。

这些具有风险的存储过程是作为*不受防护*的存储过程创建的。对于不受防护存储过程，没有任何对象将该存储过程与数据库代理进程使用的数据库控制结构分开。如果 DBA 要确保存储过程操作不会无意或恶意地损坏数据库控制结构，那么可以省略*不受防护*选项。

由于存在数据库被损坏的风险，所以**只有**在需要最大可能的性能好处时，才应使用*不受防护*存储过程。此外，一定要确保该过程编码正确，并且在经过彻底测试后，才允许它作为不受防护存储过程运行。如果在运行不受防护存储过程时发生致命错误，那么数据库管理器将确定是应用程序还是数据库管理器代码中发生错误，并执行相应的恢复操作。

不受防护存储过程可能毁坏数据库管理器并使其无法恢复，这可能导致数据丢失并使数据库被损坏。因此，在运行不受防护的被信任的存储过程时，一定要特别小心。在几乎所有情况下，对应用程序进行正确的性能分析会产生较好的性能，而不必使用不受防护存储过程。例如，触发器可以提高性能。

提高 SQL 过程的性能

有关 DB2 如何编译 SQL PL 和内联 SQL PL 的概述

在讨论如何提高 SQL 过程的性能之前，我们应讨论 DB2 在执行 CREATE PROCEDURE 语句时如何编译 SQL 过程。

创建 SQL 过程时，DB2 将过程主体中的 SQL 查询与过程逻辑分开。为了使性能达到最大，SQL 查询被静态编译到程序包的各节中。对于静态编译的查询，节主要包括 DB2 优化器为该查询选择的访问方案。程序包是节的集合。有关程序包和节的更多信息，请参阅《DB2 SQL 引用》。过程逻辑被编译到动态链接的库中。

在执行过程期间，每次控制权从过程逻辑转至 SQL 语句时，在 DLL 与 DB2 引擎之间就存在一个“上下文开关”。从 DB2 版本 8.1 开始，SQL 过程以“不受防护方式”运行。

也就是说，它们与 DB2 引擎在相同的地址空间中运行。因此，我们这里所说的上下文开关不是操作系统级别的完整上下文开关，而是指 DB2 内层的变化。减少频繁调用的过程（如 OLTP 应用程序中的过程）或处理大量行的过程（如执行数据清理的过程）中的上下文开关数可对性能产生显著影响。

包含 SQL PL 的 SQL 过程是通过将其各个 SQL 查询编译到程序包的各节中实现的，而内联 SQL PL 函数，顾名思义，则是通过将函数体直接插入到使用该函数的查询中来实现的。SQL 函数中的查询一起编译，就像函数体是单个查询一样。每次编译使用函数的语句时，都会编译该函数。与 SQL 过程不同，SQL 函数中的过程语句是在数据流语句层中执行的。因此，每次控制权从过程转至数据流语句或从数据流语句转至过程时，都没有上下文开关。

如果逻辑中没有副作用，请改为使用 SQL 函数

由于编译过程中的 SQL PL 与编译函数中的内联 SQL PL 不同，因此，认为某个过程代码在函数中比在过程中的执行速度要快是很合理的，但前提是该过程代码仅查询 SQL 数据并且不执行数据修改，也就是说，它对数据库内外的数据没有副作用。

仅当要执行的所有语句在 SQL 函数中受支持时，这才是一个好消息。SQL 函数不能包含用于修改数据库的 SQL 语句。同样，只有一部分 SQL PL 在函数的内联 SQL PL 中可用。例如，不能在 SQL 函数中执行 CALL 语句、声明游标或返回结果集。

以下是一个包含 SQL PL 的 SQL 过程示例，该过程可很好的替代转换为 SQL 函数以最大化性能的过程:

```
CREATE PROCEDURE GetPrice (IN Vendor CHAR&(20&),
                           IN Pid INT, OUT price DECIMAL(10,3))
LANGUAGE SQL
BEGIN
  IF Vendor eq; ssq;Vendor 1ssq;
    THEN SET price eq; (SELECT ProdPrice
                        FROM V1Table
                        WHERE Id = Pid);
  ELSE IF Vendor eq; ssq;Vendor 2ssq;
    THEN SET price eq; (SELECT Price FROM V2Table
                        WHERE Pid eq; GetPrice.Pid);
  END IF;
  END
```

以下是重写的 SQL 函数:

```
CREATE FUNCTION GetPrice (Vendor CHAR(20), Pid INT)
RETURNS  DECIMAL(10,3)
LANGUAGE SQL
BEGIN
  DECLARE price DECIMAL(10,3);
  IF Vendor = 'Vendor 1'
    THEN SET price = (SELECT ProdPrice
                      FROM V1Table
                      WHERE Id = Pid);
  ELSE IF Vendor = 'Vendor 2'
    THEN SET price = (SELECT Price FROM V2Table
              WHERE Pid = GetPrice.Pid);
  END IF;
  RETURN price;
  END
```

请记住，调用函数与调用过程不同。要调用函数，可使用 VALUES 语句或调用包含有效表达式的函数，如在 SELECT 或 SET 语句中。以下是调用新函数的有效方法:

```
VALUES (GetPrice('IBM', 324))

SELECT VName FROM Vendors WHERE GetPrice(Vname, Pid) < 10

SET price = GetPrice(Vname, Pid)
```

当 SQL PL 过程中使用一个语句就已足够时，应避免使用多个语句

虽然通常应编写简洁的 SQL，但实际操作时很容易忘记这一点。例如，下列 SQL 语句:

```
INSERT INTO tab_comp VALUES (item1, price1, qty1);
INSERT INTO tab_comp VALUES (item2, price2, qty2);
INSERT INTO tab_comp VALUES (item3, price3, qty3);
```

可以重写为单个语句:

```
INSERT INTO tab_comp VALUES (item1, price1, qty1),
                            (item2, price2, qty2),
                            (item3, price3, qty3);
```

执行多行插入所需的时间大约是执行三个原始语句所需时间的三分之一。单独地说，此处对性能的影响是微不足道的，但如果该代码段重复执行，如在循环或触发器主体中，那么可以极大地改善性能。

同样，以下 SET 语句序列:

```
SET A = expr1;
SET B = expr2;
SET C = expr3;
```

可以编写为单个 VALUES 语句:

```
VALUES expr1, expr2, expr3 INTO A, B, C;
```

如果任何两个语句之间都不存在依赖性，那么此变换将保持原始序列的语义。为了说明这一点，请考虑:

```
SET A = monthly_avg * 12;
SET B = (A / 2) * correction_factor;
```

将前面两个语句转换为:

```
VALUES (monthly_avg * 12, (A / 2) * correction_factor) INTO A, B;
```

不能保持原始语义，因为 INTO 关键字前面的表达式是“并行”求值的。这意味着指定给 *B* 的值与指定给 *A* 的值无关，这就是原始语句的本来语义。

将多个 SQL 语句减少为单个 SQL 表达式

与其他编程语言一样，SQL 语言提供两种类型的条件结构: 过程（IF 和 CASE 语句）以及函数（CASE 表达式）。在大多数情况下，使用任一类型均可表示计算，此时使用哪种类型只是习惯问题。但是，使用 CASE 表达式编写的逻辑不仅更紧凑，而且比使用 CASE 或 IF 语句编写的逻辑效率更高。

请考虑以下 SQL PL 代码段:

```
IF (Price <= MaxPrice) THEN
  INSERT INTO tab_comp(Id, Val) VALUES(Oid, Price);
ELSE
  INSERT INTO tab_comp(Id, Val) VALUES(Oid, MaxPrice);
END IF;
```

IF 子句中的条件仅仅用来决定插入到 tab_comp.Val 列中的值。为了避免在过程层与数据流层之间出现上下文开关，可以将相同逻辑表示为包含 CASE 表达式的单个 INSERT 语句:

```
INSERT INTO tab_comp(Id, Val)
       VALUES(Oid,
            CASE
               WHEN (Price <= MaxPrice) THEN Price
               ELSE MaxPrice
            END);
```

值得注意的是，可以在期望标量值的任何上下文中使用 CASE 表达式。特别是，它们可以用在赋值语句的右边。例如:

```
IF (Name IS NOT NULL) THEN
  SET ProdName = Name;
ELSEIF (NameStr IS NOT NULL) THEN
  SET ProdName = NameStr;
ELSE
  SET ProdName = DefaultName;
END IF;
```

可以重写为:

```
SET ProdName = (CASE
                  WHEN (Name IS NOT NULL) THEN Name
                  WHEN (NameStr IS NOT NULL) THEN NameStr
                  ELSE  DefaultName
                END);
```

事实上，此特定示例有一种更好的解决方案:

```
SET ProdName = COALESCE(Name, NameStr, DefaultName);
```

不要低估花一些时间分析并考虑重写您的 SQL 所带来的好处。虽然分析并重写过程要花一些时间，但性能会提高好几倍。

利用 SQL 的一次处理一个集合语义

过程构造（如循环、赋值和游标）允许我们表示仅使用 SQL DML 语句可能无法表示的计算。但当我们可以随意使用过程语句时，有可能实际上仅使用 SQL DML 语句就能表示现有计算时，我们却使用过程语句。正如前面所提及的那样，过程计算的性能可能比使用 DML 语句表示的等价计算的性能要差好几个等级。请考虑以下代码段:

```
DECLARE cur1 CURSOR FOR SELECT col1, col2 FROM tab_comp;
  OPEN cur1;
FETCH cur1 INTO v1, v2;
WHILE SQLCODE <> 100 DO
  IF (v1 > 20) THEN
    INSERT INTO tab_sel VALUES (20, v2);
  ELSE
    INSERT INTO tab_sel VALUES (v1, v2);
  END IF;
  FETCH cur1 INTO v1, v2;
  END WHILE;
```

首先，可以通过应用最后一节“将多个 SQL 语句减少为单个 SQL 表达式”中讨论的变换来改善循环主体:

```
DECLARE cur1 CURSOR FOR SELECT col1, col2 FROM tab_comp;
  OPEN cur1;
FETCH cur1 INTO v1, v2;
WHILE SQLCODE <> 100 DO
```

```
  INSERT INTO tab_sel VALUES (CASE
                                WHEN v1 > 20 THEN 20
                                ELSE v1
                              END, v2);
  FETCH cur1 INTO v1, v2;
  END WHILE;
```

但通过更仔细地观察，可以将整个代码块编写为带有子查询的 INSERT 语句:

```
INSERT INTO tab_sel (SELECT (CASE
                               WHEN col1 > 20 THEN 20
                               ELSE col1
                             END),
                             col2
                     FROM tab_comp);
```

在原始公式中，对于 SELECT 语句中的每行，过程层和数据流层之间都存在一个上下文开关。在最后的公式中，根本没有上下文开关，并且优化器可以全局优化整个计算。

另一方面，如果每个 INSERT 语句都针对不同的表，那么不可能如此大地简化过程，如下所示:

```
DECLARE cur1 CURSOR FOR SELECT col1, col2 FROM tab_comp;
  OPEN cur1;
FETCH cur1 INTO v1, v2;
WHILE SQLCODE <> 100 DO
  IF (v1 > 20) THEN
    INSERT INTO tab_default VALUES (20, v2);
  ELSE
    INSERT INTO tab_sel VALUES (v1, v2);
  END IF;
  FETCH cur1 INTO v1, v2;
  END WHILE;
```

但是，此处也可以利用 SQL 的一次处理一个集合的性质:

```
INSERT INTO tab_sel (SELECT col1, col2
                     FROM tab_comp
                     WHERE col1 <= 20);
INSERT INTO tab_default (SELECT col1, col2
                         FROM tab_comp
                         WHERE col1 > 20);
```

观察现有过程逻辑的性能是如何提高的时，会发现消除游标循环所用的任何时间都可能得到补偿。

保持通知 DB2 优化器

创建过程时，其各个 SQL 查询将被编译到程序包的各节中。DB2 优化器还将根据表统计信息（例如，表大小或列中数据值的相对频率）和编译查询时可用的索引来为查询选择执行方案。当表进行了大量更改时，最好让 DB2 再次收集关于这些表的统计信息。更新了统计信息或者创建新索引后，同样最好重新绑定与使用这些表的 SQL 过程关联的程序包，并且让 DB2 创建一些利用最新统计信息和索引的方案。

可使用 RUNSTATS 命令更新表统计信息。要重新绑定与 SQL 过程关联的程序包，可使用 DB2 版本 8.1 中提供的 REBIND_ROUTINE_PACKAGE 内置过程。例如，可使用以下命令重新绑定过程 MYSCHEMA.MYPROC 的程序包:

```
CALL SYSPROC.REBIND_ROUTINE_PACKAGE('P', 'MYSCHEMA.MYPROC', 'ANY')
```

其中“P”指示与过程相对应的程序包，而“ANY”指示在解析函数和类型时将考虑 SQL 路径中的所有函数和类型。请参阅 REBIND 命令的“命令参考”条目以了解更多详细信息。

使用数组

您可以使用数组以在应用程序与存储过程之间更有效率地传递数据以及存储和处理 SQL 过程中的瞬态数据集合，而不必使用关系表。SQL 过程中可用的数组运算符允许更有效率地存储和检索数据。由于整个数组存储在主存储器中，因此，与创建非常大的数组（大小为几个兆字节）的应用程序相比，创建大小合适的数组的应用程序可显著提高性能。请参阅*相关链接*部分以了解其他信息。

第 22 章 查询访问方案

SQL 和 XQuery 编译器进程

SQL 和 XQuery 编译器执行几个步骤来产生可以执行的访问方案。这些步骤显示在下图中并在图下面的节中进行了描述。注意，某些步骤仅针对联合数据库中的查询。

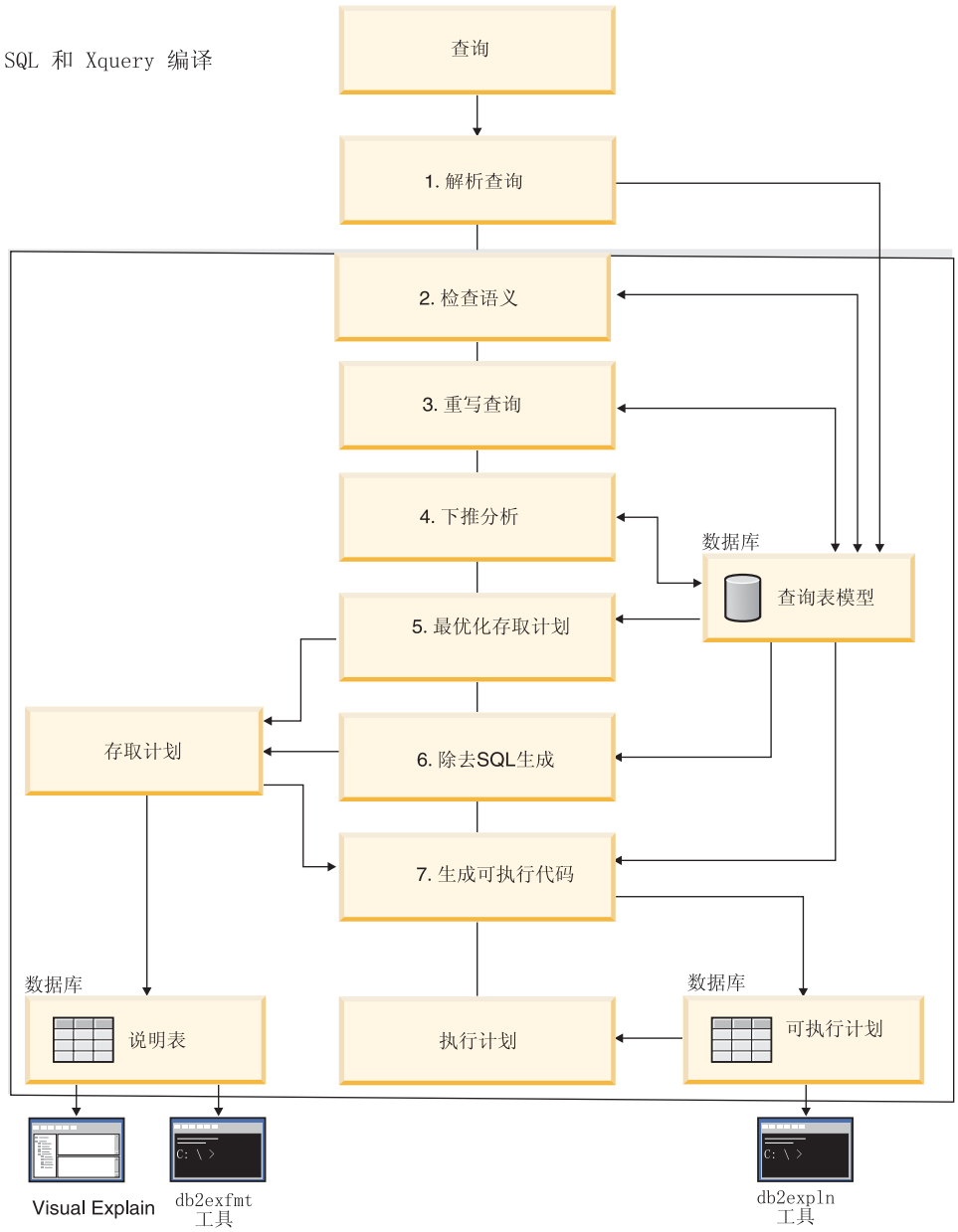


图 22. SQL 和 XQuery 编译器执行的步骤

查询图模型

*查询图模型*是一个位于内存中的内部数据库，它代表如下所述步骤中处理的查询:

1. **解析查询**

   SQL 和 XQuery 编译器分析查询以验证其语法。如果检测到任何语法错误，那么查询编译器停止处理，并将适当的错误返回至提交该查询的应用程序。当解析完成时，创建该查询的内部表示并将它存储在查询图模型中。

2. **检查语义**

   编译器确保在语句的各个部分中没有不一致。作为语义检查的一个简单示例，编译器验证为 YEAR 标量函数指定的列的数据类型是日期时间数据类型。

   编译器也将行为语义添加至该查询图模型，包括引用约束、表检查约束、触发器和视图的作用。该查询图模型包含查询的所有语义，包括查询块、子查询、相关、派生的表、表达式、数据类型、数据类型转换、代码页转换和分布键。

3. **重写查询**

   编译器使用存储在查询图模型中的全局语义来将该查询变换为更易于优化的一种形式并将结果存储在查询图模型中。

   例如，编译器可能会移动谓词，改变其应用级别并有可能提高查询性能。这种类型的操作移动称为*一般谓词下推*。在一个分区数据库环境中，下列查询操作的计算更为集中:

   - 聚集
   - 行的再分发
   - 相关子查询，它是包含对该子查询之外表列的引用的子查询。

   对于分区数据库环境中的某些查询，解除相关可能作为重写查询的一部分而发生。

4. **下推分析（联合数据库）**

   此步骤中的主要任务是向优化器建议是否可在数据源处对一个操作以远程方式求值（即*下推*）。这种类型的下推活动仅适用于数据源查询，它是对一般谓词下推操作的扩展。

   除非执行联合数据库查询，否则绕过此步骤。

5. **优化访问方案**

   通过使用查询图模型作为输入，编译器的优化器部分生成许许多备用执行方案以满足查询。要估计每个备用方案的执行成本，优化器使用有关表、索引、列和函数的统计信息。然后，它选择具有最小估计执行成本的方案。优化器使用查询图模型来分析查询语义，并获取有关各种因素的信息，包括索引、基本表、派生的表、子查询、关联和递归。

   优化器还可考虑另一类下推操作，*聚集与排序*，这种操作可将对这些操作求得的值下推到“数据管理服务”组件中来提高性能。

   在确定页大小选择时，优化器还考虑是否有不同大小的缓冲池。该环境是否包括一个分区数据库，以及是否能够为在对称多处理器（SMP）环境中实现查询内并行性改善选择的方案均属考虑范围。优化器使用此信息来帮助选择该查询的最佳访问方案。

编译器此步骤的输出是访问方案。此访问方案提供说明表中捕获的信息。可以用说明快照捕获用于生成该访问方案的信息。

6. **远程 SQL 生成（联合数据库）**

   优化器选择的最终方案可由一组对远程数据源执行操作的步骤组成。对于每个数据源执行的那些操作，远程 SQL 生成步骤将根据数据源 SQL 方言创建一个高效的 SQL 语句。

7. **生成"可执行"代码**

   在最后一个步骤中，编译器使用访问方案和查询图模型来创建查询的可执行访问方案或节。此代码生成步骤使用查询图模型中的信息，以避免重复执行需要对一个查询只计算一次的表达式。可以进行此优化的示例包括代码页转换和主变量的使用。

   要对具有主变量、专用寄存器或参数标记的静态和动态 SQL 和 XQuery 语句启用查询（重新）优化，应将程序包与 REOPT 绑定选项进行绑定。如果使用此项，将使用主变量、参数标记或专用寄存器的值而不是由编译器选择的缺省估计值来优化属于该程序包并且包含这些变量的 SQL 或 XQuery 语句的访问路径。当值可用时，执行查询时就会进行此优化。

   有关静态 SQL 和 XQuery 语句访问方案的信息存储在系统目录表中。当执行该程序包时，数据库管理器将使用存储在系统目录表中的信息来确定如何访问该数据，并提供该查询的结果。此信息由 *db2expln* 工具使用。

注: 在经常更改的表上以适当的时间间隔执行 RUNSTATS。优化器需要有关表及其数据的最新统计信息以创建最有效的访问方案。重新绑定应用程序以利用更新的统计信息。如果未执行 RUNSTATS 或优化器怀疑在空表或几乎是空的表上执行了 RUNSTATS，那么优化器可能使用缺省值或尝试根据在磁盘上存储该表所用的文件页的数目（FPAGES）来得出特定的统计信息。已占用块的总数存储在 ACTIVE_BLOCKS 列中。

查询重写方法和示例

在重写查询阶段，查询编译器将 SQL 和 XQuery 语句变换为更易于优化的格式，并因此改善可能的访问路径。重写查询对于非常复杂的查询（包括具有许多子查询或许多连接的查询）特别重要。查询生成器工具通常会创建这些很复杂的查询。

要影响应用于 SQL 或 XQuery 语句的查询重写规则的数目，更改优化级别。要查看查询重写的一些结果，使用说明工具或 Visual Explain。

可以下列三种主要方式中的任何一种重写查询:

- **操作合并**

  要构造查询以便它具有最少的操作数（尤其是 SELECT 操作），SQL 和 XQuery 编译器重写查询来合并查询操作。以下示例举例说明可合并的一些操作:

  – 示例 - 视图合并

    使用视图的 SELECT 语句可限制表的连接顺序并可引入表的冗余连接。如果在查询重写期间合并视图，那么可撤销这些限制。

  – 示例 - 子查询至连接的变换

如果某个 SELECT 语句包含子查询，那么可以限制表的顺序处理的选择。

– 示例 - 消除冗余连接

在查询重写期间，可除去冗余连接来简化 SELECT 语句。

– 示例 - 共享聚集

当查询使用不同的函数时，重写可减少需要执行的计算数。

- **操作移动**

要构造具有最少数目的操作和谓词的查询，编译器重写查询以移动查询操作。以下示例举例说明可移动的一些操作:

– 示例 - 消除 DISTINCT

在查询重写期间，优化器可移动执行 DISTINCT 操作的位置，以减少此操作的成本。在提供的扩展示例中，全部除去了 DISTINCT 操作。

– 示例 - 一般谓词下推

在查询重写期间，优化器可更改应用谓词的顺序，以便将选择性更强的谓词尽早地应用于该查询。

– 示例 - 解除相关

在分区数据库环境中，在数据库分区之间移动结果集的成本很高。减小必须广播至其他数据库分区的信息的大小和/或减少广播次数，是查询重写的目标。

- **谓词转换**

SQL 和 XQuery 编译器重写查询，以便把特定查询的现有谓词转换为更优的谓词。以下示例举例说明可转换的一些谓词:

– 示例 - 添加隐含谓词

在查询重写期间，可以将谓词添加至查询，以允许优化器在选择该查询的最佳访问方案时考虑其他表连接。

– 示例 - OR 至 IN 的变换

在查询重写期间，可以将一个 OR 谓词转换为一个 IN 谓词，以获取更有效的访问方案。SQL 和 XQuery 编译器也可将一个 IN 谓词转换为一个 OR 谓词，但前提是此变换将创建一种更有效的访问方案。

编译器重写示例: 视图合并

假定您可访问 EMPLOYEE 表的以下两个视图，一个视图显示受过高等教育的职员，另一个视图显示工资超过 $35,000 的职员:

```
CREATE VIEW EMP_EDUCATION (EMPNO, FIRSTNME, LASTNAME, EDLEVEL) AS
   SELECT EMPNO, FIRSTNME, LASTNAME, EDLEVEL
       FROM EMPLOYEE        WHERE EDLEVEL > 17
   CREATE VIEW EMP_SALARIES (EMPNO, FIRSTNAME, LASTNAME, SALARY) AS
   SELECT EMPNO, FIRSTNME, LASTNAME, SALARY
       FROM EMPLOYEE        WHERE SALARY > 35000
```

现在，假定您执行以下查询，来列示既受过高等教育而且工资又超过 $35,000 的职员:

```
SELECT E1.EMPNO, E1.FIRSTNME, E1.LASTNAME, E1.EDLEVEL, E2.SALARY
       FROM EMP_EDUCATION E1,
                 EMP_SALARIES  E2
     WHERE E1.EMPNO = E2.EMPNO
```

在查询重写期间，可合并这两个视图来创建以下查询:

```
SELECT E1.EMPNO, E1.FIRSTNME, E1.LASTNAME, E1.EDLEVEL, E2.SALARY
       FROM EMPLOYEE E1,
                 EMPLOYEE E2
     WHERE E1.EMPNO = E2.EMPNO            AND E1.EDLEVEL > 17
         AND E2.SALARY  > 35000
```

通过使用用户编写的 SELECT 语句来合并这两个视图中的 SELECT 语句，优化器在选择访问方案时可考虑更多选项。此外，如果已合并的这两个视图使用相同的基本表，还可执行其他重写。

示例 - 子查询至连接的变换

SQL 和 XQuery 编译器将执行一个包含子查询的查询，如:

```
SELECT EMPNO, FIRSTNME, LASTNAME, PHONENO
     FROM EMPLOYEE      WHERE WORKDEPT IN
               (SELECT DEPTNO
                     FROM DEPARTMENT
                   WHERE DEPTNAME = 'OPERATIONS')
```

并将它转换为如下形式的连接查询:

```
SELECT DISTINCT EMPNO, FIRSTNME, LASTNAME, PHONENO
       FROM EMPLOYEE EMP,
                 DEPARTMENT DEPT
     WHERE EMP.WORKDEPT = DEPT.DEPTNO
         AND DEPT.DEPTNAME = 'OPERATIONS'
```

一般情况下，连接的执行效率比子查询高很多。

示例 - 消除冗余连接

有时编写或生成的查询可能含有不需要的连接。在查询重写阶段期间，也可能产生类似以下的查询。

```
SELECT E1.EMPNO, E1.FIRSTNME, E1.LASTNAME, E1.EDLEVEL, E2.SALARY
       FROM EMPLOYEE E1,
                 EMPLOYEE E2
     WHERE E1.EMPNO = E2.EMPNO            AND E1.EDLEVEL > 17
         AND E2.SALARY  > 35000
```

在此查询中，SQL 和 XQuery 编译器可消除连接并将查询简化为:

```
SELECT EMPNO, FIRSTNME, LASTNAME, EDLEVEL, SALARY
       FROM EMPLOYEE        WHERE EDLEVEL > 17
         AND SALARY  > 35000
```

另一个示例假设具有该部门号的 EMPLOYEE 和 DEPARTMENT 样本表之间存在引用约束。首先，创建一个视图。

```
CREATE VIEW PEPLVIEW
         AS SELECT FIRSTNME, LASTNAME, SALARY, DEPTNO, DEPTNAME, MGRNO
                     FROM EMPLOYEE E DEPARTMENT D
                     WHERE E.WORKDEPT = D.DEPTNO
```

于是，诸如以下内容的查询:

```
SELECT LASTNAME, SALARY
   FROM PEPLVIEW
```

将变成:

```
SELECT LASTNAME, SALARY
   FROM EMPLOYEE
   WHERE WORKDEPT NOT NULL
```

注意，在这种情况下，即使用户知道可重新编写查询，他们可能也无法这样做，因为他们对基础表没有访问权。他们可能只对以上显示的视图具有访问权。因此，必须在数据库管理器内执行这种类型的优化。

在下列情况下，引用完整性连接中很可能存在冗余:

- 视图是通过连接定义的
- 查询是自动生成的。

  例如，查询管理器提供了一些自动化工具，它们会阻止用户编写优化的查询。

示例 - 共享聚集

在一个查询中使用多个函数可生成几次计算，这会占用时间。将要在查询中执行的计算数减少可改进方案。SQL 和 XQuery 编译器执行一个使用多个函数的查询，如:

```
SELECT SUM(SALARY+BONUS+COMM) AS OSUM,
       AVG(SALARY+BONUS+COMM) AS OAVG,
       COUNT(*) AS OCOUNT
   FROM EMPLOYEE;
```

用下列方式变换该查询:

```
   SELECT OSUM,
       OSUM/OCOUNT
       OCOUNT
   FROM (SELECT SUM(SALARY+BONUS+COMM) AS OSUM,
       COUNT(*) AS OCOUNT
   FROM EMPLOYEE) AS SHARED_AGG;
```

此重写将该查询从 2 次求和 2 次计数减少为 1 次求和 1 次计数。

编译器重写示例: 消除 DISTINCT

如果 EMPNO 列被定义为 EMPLOYEE 表的主键，那么以下查询:

```
SELECT DISTINCT EMPNO, FIRSTNME, LASTNAME
       FROM EMPLOYEE
```

将通过除去 DISTINCT 子句来重写:

```
SELECT EMPNO, FIRSTNME, LASTNAME
       FROM EMPLOYEE
```

在以上示例中，由于选择了主键，因此 SQL 和 XQuery 编译器知道返回的每一行已经是唯一的。在这种情况下，DISTINCT 关键字是多余的。如果未重写该查询，那么优化器将需要通过必要的处理（例如，排序）来构建一个方案，以确保这些列是相异的。

示例 - 一般谓词下推

改变谓词通常应用的级别可改善性能。例如，假定以下视图，它提供部门“D11”中所有职员的列表:

```
CREATE VIEW D11_EMPLOYEE
     (EMPNO, FIRSTNME, LASTNAME, PHONENO, SALARY, BONUS, COMM)
   AS SELECT EMPNO, FIRSTNME, LASTNAME, PHONENO, SALARY, BONUS, COMM
         FROM EMPLOYEE              WHERE WORKDEPT = 'D11'
```

并假定以下查询:

```
SELECT FIRSTNME, PHONENO
       FROM D11_EMPLOYEE
     WHERE LASTNAME = 'BROWN'
```

该编译器的查询重写阶段将把谓词 LASTNAME = 'BROWN' 下推到视图 D11_EMPLOYEE 中。这使得可以更快并可能更有效地应用该谓词。在此示例中可能执行的实际查询为:

```
SELECT FIRSTNME, PHONENO
  FROM EMPLOYEE    WHERE LASTNAME = 'BROWN'      AND WORKDEPT = 'D11'
```

谓词的下推并不限于视图。可以下推谓词的其他情况包括 UNION、GROUP BY 和派生的表（嵌套表表达式或公共表表达式）。

示例 - 解除相关

在分区数据库环境中，SQL 和 XQuery 编译器可重写以下查询:

查找正在从事程序设计项目并被少付工资的所有职员。

```
SELECT P.PROJNO, E.EMPNO, E.LASTNAME, E.FIRSTNAME,
                 E.SALARY+E.BONUS+E.COMM AS COMPENSATION
       FROM EMPLOYEE E, PROJECT P
    WHERE P.EMPNO = E.EMPNO
        AND P.PROJNAME LIKE '%PROGRAMMING%'
         AND E.SALARY+E.BONUS+E.COMM <
             (SELECT AVG(E1.SALARY+E1.BONUS+E1.COMM)
                   FROM EMPLOYEE E1, PROJECT P1
                   WHERE P1.PROJNAME LIKE '%PROGRAMMING%'
                       AND P1.PROJNO = A.PROJNO
                       AND E1.EMPNO = P1.EMPNO)
```

因为此查询是相关的，而且因为 PROJECT 和 EMPLOYEE 不可能根据 PROJNO 来分区，因此将每个项目广播至每个数据库分区是可能的。此外，子查询将不得不求值多次。

SQL 和 XQuery 编译器可按如下所示重写查询:

- 确定从事程序设计项目的职员的单值列表，并将其称为 DIST_PROJS。它必须是相异的，才能确保只对每个项目执行一次聚集:

  ```
  WITH DIST_PROJS(PROJNO, EMPNO) AS
     (SELECT DISTINCT PROJNO, EMPNO
       FROM PROJECT P1
       WHERE P1.PROJNAME LIKE '%PROGRAMMING%')
  ```

- 使用从事程序设计项目的职员的单值列表，将该列表与职员表连接，以获得每个项目的平均赔偿额 AVG_PER_PROJ:

  ```
  AVG_PER_PROJ(PROJNO, AVG_COMP) AS
     (SELECT P2.PROJNO, AVG(E1.SALARY+E1.BONUS+E1.COMM)
       FROM EMPLOYEE E1, DIST_PROJS P2
       WHERE E1.EMPNO = P2.EMPNO
       GROUP BY P2.PROJNO)
  ```

- 于是，新查询将变为:

```
SELECT P.PROJNO, E.EMPNO, E.LASTNAME, E.FIRSTNAME,
       E.SALARY+E.BONUS+E.COMM AS COMPENSATION
 FROM PROJECT P, EMPLOYEE E, AVG_PER_PROG A
WHERE P.EMPNO = E.EMPNO
  AND P.PROJNAME LIKE '%PROGRAMMING%'
  AND P.PROJNO = A.PROJNO
  AND E.SALARY+E.BONUS+E.COMM < A.AVG_COMP
```

重写的查询计算每个项目的 AVG_COMP（AVG_PRE_PROJ），然后将结果广播至包含 EMPLOYEE 表的所有数据库分区。

编译器重写示例: 隐含谓词

以下查询产生一个其部门是向“E01”报告的经理的列表，以及那些经理所负责的项目的列表:

```
SELECT DEPT.DEPTNAME DEPT.MGRNO, EMP.LASTNAME, PROJ.PROJNAME
    FROM DEPARTMENT DEPT,
               EMPLOYEE   EMP,
               PROJECT    PROJ
  WHERE DEPT.ADMRDEPT = 'E01'
      AND DEPT.MGRNO = EMP.EMPNO
      AND EMP.EMPNO  = PROJ.RESPEMP
```

该查询重写添加以下隐含的谓词:

```
DEPT.MGRNO = PROJ.RESPEMP
```

作为此重写的结果，优化器在尝试选择该查询的最佳访问方案时可以考虑其他连接。

除以上的谓词传递闭包外，查询重写还根据等式谓词隐含的传递性派生其他的本地谓词。例如，以下查询将列示其部门号大于“E00”的部门和在这些部门工作的职员的名称。

```
SELECT EMPNO, LASTNAME, FIRSTNAME, DEPTNO, DEPTNAME
    FROM EMPLOYEE EMP,
               DEPARTMENT DEPT
  WHERE EMP.WORKDEPT = DEPT.DEPTNO
      AND DEPT.DEPTNO > 'E00'
```

对于此查询，该重写阶段添加以下隐含的谓词:

```
EMP.WORKDEPT > 'E00'
```

作为此重写的结果，优化器会减少要连接的行数。

示例 - OR 至 IN 的变换

假定一个 OR 子句根据同一列将两个或更多个简单的等式谓词相连，如以下示例所示:

```
SELECT *
   FROM EMPLOYEE        WHERE DEPTNO = 'D11'
       OR DEPTNO = 'D21'
       OR DEPTNO = 'E21'
```

如果 DEPTNO 列上没有索引，将 OR 子句转换为以下 IN 谓词允许更有效地处理该查询:

```
SELECT *
   FROM EMPLOYEE        WHERE DEPTNO IN ('D11', 'D21', 'E21')
```

注: 在某些情况下，数据库管理器可以将一个 IN 谓词转换为一组 OR 子句，以便可执行索引 OR 运算。

谓词类型和访问方案

用户应用程序使用查询语句从数据库请求一组行，该查询语句指定要作为结果集返回的特定行的限定词。这些限定符通常出现在查询的 WHERE 子句。这种限定符称为*谓词*。可以将谓词分组为四种类别，按如何在求值过程中使用该谓词以及何时使用它来确定这些类别。这些类别列示如下，按最佳性能到最差性能的顺序排列:

1. 范围定界谓词
2. 索引控制谓词
3. 数据控制谓词
4. 残留谓词。

注: *SARGable* 是指可用作*搜索自变量*的术语。

下表总结谓词类别。后续各节更详细地描述每个类别。

表 62. 谓词类型特征的总结

特征	谓词类型			
	范围定界	索引控制	数据控制	残留
减少索引 I/O	是	否	否	否
减少数据页 I/O	是	是	否	否
减少内部传送的行数	是	是	是	否
减少合格行的数目	是	是	是	是

范围定界和索引控制谓词

范围定界谓词限制索引扫描的范围。它们提供索引搜索的开始和停止键值。索引控制谓词不能限制搜索范围，但可根据该索引来求值，因为在该谓词中涉及到的列是索引键的一部分。例如，考虑下列索引:

```
INDEX IX1:  NAME    ASC,
                        DEPT    ASC,
                        MGR     DESC,
                        SALARY  DESC,
                        YEARS   ASC
```

也考虑包含下列 WHERE 子句的查询:

```
WHERE  NAME  = :hv1
   AND  DEPT = :hv2        AND   YEARS > :hv5
```

前两个谓词（NAME = :hv1 和 DEPT = :hv2）是范围定界谓词，而 YEARS > :hv5 是索引控制谓词。

优化器在对这些谓词求值时使用索引数据而不是读取基本表。这些*索引控制*谓词减小需要从表中读取的行集，但它们不影响访问的索引页数。

XSCAN 数据运算符扫描也支持 XMLEXISTS 和 XMLTABLE 表达式中出现的 XML 数据上的谓词。索引范围扫描也支持这些谓词中的一部分谓词。

数据控制谓词

索引管理器不能求值但数据管理服务可以求值的谓词称为*数据控制*谓词。这些谓词通常需要访问表中的单独行。如果需要，“数据管理服务”检索对该谓词求值需要的列以及任何其他列以满足 SELECT 列表中不能从该索引获取的列。

例如，考虑在 PROJECT 表上定义的单个索引:

```
INDEX IX0:  PROJNO ASC
```

对于下列查询，然后，将 DEPTNO = 'D11' 谓词视为数据 SARGable。

```
SELECT PROJNO, PROJNAME, RESPEMP
        FROM PROJECT
     WHERE DEPTNO = 'D11'
      ORDER BY PROJNO
```

残留谓词

残留谓词比访问表需要更多的 I/O 成本。它们可具有下列特征:

- 使用相关子查询
- 使用量化子查询，这些子查询包含 ANY、ALL、SOME 或 IN 子句
- 读取 LONG VARCHAR 或 LOB 数据，该数据存储在与表分开的文件中

这种谓词由“关系数据服务”来求值。

有时，当访问数据页时，必须重新应用仅应用于索引的谓词。例如，当访问数据页时，使用索引 OR 运算或索引 AND 运算的访问方案始终要将这些谓词作为残留谓词重新应用。

联合数据库查询编译器阶段

联合数据库下推分析

对于联合数据库中的查询，优化器执行下推分析来查明是否可以在远程数据源上执行某个操作。操作可以是一个函数，例如，关系运算符、系统函数或用户函数或 SQL 运算符，例如，GROUP BY 或 ORDER BY 等。

注: 虽然 DB2 SQL 编译器具有大量有关数据源 SQL 支持的信息，但是此数据在一段时间之后可能需要调整，因为数据源可能已升级和/或定制。在这些情况下，通过更改本地目录信息来向 DB2 提供增强信息。使用 DB2 DDL 语句（如 CREATE FUNCTION MAPPING 和 ALTER SERVER）来更新目录。

如果不能将函数下推到远程数据源，那么它们可显著影响查询性能。考虑强制在本地而不在数据源对选择谓词求值的效果。这种求值可能要求 DB2 从远程数据源检索整个表，然后在本地对该谓词进行过滤。网络约束和大表大小可使性能受到损害。

未下推的运算符还可能显著地影响查询性能。例如，让 GROUP BY 运算符在本地聚集远程数据也会要求 DB2 从远程数据源检索整个表。

例如，假设昵称 N1 引用 DB2 OS/390® 或 z/OS 版数据源中的数据源表 EMPLOYEE。同时假设该表有 10,000 行，其中一列包含职员的姓，一列包含工资。考虑以下语句:

```
SELECT LASTNAME, COUNT(*)  FROM N1
   WHERE LASTNAME  > 'B' AND SALARY > 50000
   GROUP BY LASTNAME;
```

取决于 DB2 和 DB2 OS/390 或 z/OS 版上的整理顺序是否相同，考虑几种可能性:

- 如果整理顺序相同，那么可将查询谓词下推到 DB2 OS/390 版或 z/OS 版中。在数据源对结果进行过滤和分组通常比将整个表复制到 DB2 然后在本地执行那些操作更有效。对于以上查询，该谓词和 GROUP BY 操作可在数据源中执行。
- 如果整理顺序不同，那么不能在数据源对整个谓词求值。但是，优化器可能决定下推谓词的 SALARY > 50000 部分。仍然必须在 DB2 进行范围比较。
- 如果整理顺序相同，且优化器知道本地 DB2 服务器非常快，那么优化器可能决定在 DB2 中本地执行 GROUP BY 操作是最好（成本最低）的方法。在数据源中对该谓词求值。这是下推分析结合全局优化的示例。

通常，目标是确保优化器在数据源处对函数和运算符求值。许多因素影响是否在远程数据源对函数或 SQL 运算符求值。将要评估的因素分类为下列三组:

- 服务器特征
- 昵称特征
- 查询特征

影响下推机会的服务器特征

某些特定于数据源的因素会影响下推机会。一般情况下，由于 DB2 支持丰富的 SQL 方言，因此存在这些因素。与查询访问的服务器所支持的 SQL 语言相比，此语言可提供更多的功能。DB2 可弥补数据服务器中函数的不足，但这样做可能要求在 DB2 中执行操作。

SQL 能力: 每个数据源都支持不同的 SQL 语言和不同级别的功能。例如，考虑 GROUP BY 列表。大多数数据源都支持 GROUP BY 运算符，但某些数据源限制 GROUP BY 列表中的项数，或者对 GROUP BY 列表上是否允许表达式有限制。如果对远程数据源有限制，那么 DB2 可能只好在本地执行 GROUP BY 操作。

SQL 约束: 每个数据源可能有不同的 SQL 限制。例如，某些数据源需要参数标记才能将值绑定至远程 SQL 语句中。因此，必须检查参数标记限制以确保每个数据源可支持这种绑定机制。如果 DB2 不能确定一个好方法来绑定函数的值，必须在本地对此函数求值。

SQL 限制: 尽管 DB2 可能允许使用比它的远程数据源大的整数，但不能将超出远程限制的值嵌入发送给数据源的语句。因此，对此常数进行运算的函数或运算符必须在本地求值。

服务器细节: 有几个因素都归入此类别。一个示例为是将空值排序为最高或最低值，还是取决于排序。如果在某个与 DB2 不同的数据源对空值排序，那么对可空表达式的 ORDER BY 操作无法以远程方式求值。

整理顺序: 检索数据以进行本地排序和比较通常会降低性能。因此，可考虑将联合数据库配置为使用数据源所用的整理顺序。如果配置联合数据库以使用与数据源所用的相同的整理顺序，然后将 *collating_sequence* 服务器选项设置为“Y”，那么优化器可考虑下推许多查询操作（如果改进性能结果）。

如果整理顺序相同，那么可以下推下列操作:

- 字符或数字数据的比较
- 字符范围比较谓词
- 排序

但是，如果联合数据库和数据源的空字符加权不同，可能得到不正常的结果。如果将此语句落实给不区分大小写的数据源，那么比较语句可能返回意外的结果。在不区分大小写的数据源中指定给字符“I”和“i”的加权是一样的。缺省情况下，DB2 是区分大小写的，并且给字符指定不同的加权。

为了提高性能，联合服务器允许在数据源处进行排序和比较。例如，在 DB2 OS/390 或 z/OS 版中，由 ORDER BY 子句定义的排序用基于 EBCDIC 代码页的整理顺序来实现。要使用联合服务器来检索根据 ORDER BY 子句排序的 DB2 OS/390 或 z/OS 版数据，配置联合数据库以让它使用基于 EBCDIC 代码页的预定义整理顺序。

如果联合数据库和数据源的整理顺序不同，那么 DB2 将数据检索到联合数据库中。因为用户希望看到根据对联合服务器定义的整理顺序排序的查询结果，所以通过在本地对数据排序，联合服务器确保实现这个期望。如果需要查看以数据源的整理顺序排序的数据，那么用通过方式落实您的查询，或者在数据源视图中定义该查询。

服务器选项：有几个服务器选项会影响下推机会。尤其要查看 *collating_sequence*、*varchar_no_trailing_blanks* 和 *pushdown* 的设置。

DB2 类型映射和函数映射因素：DB2 提供的缺省本地数据类型映射设计成为每个数据源数据类型提供足够的缓冲区空间，这就可避免数据丢失。用户可为特定数据源定制类型映射以适合特定的应用程序。例如，如果要访问具有 DATE 数据类型（缺省情况下，它被映射至 DB2 TIMESTAMP 数据类型）的 Oracle 数据源列，那么可将本地数据类型更改为 DB2 DATE 数据类型。

在下列三种情况下，DB2 可以补偿数据源不支持的函数:

- 远程数据源中不存在此函数。
- 该函数存在，但操作数特征违反了函数限制。这种情况的一个例子是 IS NULL 关系运算符。大多数数据源支持它，但某些数据源可能有限制，如只允许 IS NULL 运算符左边存在列名。
- 如果以远程方式对该函数求值，那么它可能返回不同的结果。这种情况的一个例子是“>”（大于）运算符。对于具有不同整理顺序的数据源，如果由 DB2 在本地对大于运算符求值，那么它可能会返回不同的结果。

影响下推机会的昵称特征

以下特定于昵称的因素会影响下推机会。

昵称列的本地数据类型：确保列的本地数据类型不会妨碍在数据源对谓词求值。使用缺省数据类型映射来避免可能的溢出。但是，可能不会考虑在连接列较短的数据源处用谓词连接两个不同长度的列，这取决于 DB2 如何绑定较长的列。这种情况可影响由 DB2 优化器确定的连接顺序中存在的可能性数目。例如，将类型 NUMBER(38) 赋予使用 INTEGER 或 INT 数据类型创建的 Oracle 数据源列。将本地数据类型 FLOAT 赋予此 Oracle 数据类型的昵称列，因为 DB2 整数的范围是从 2**31 到 (-2**31)-1，它近似等于 NUMBER(9)。在这种情况下，不能在 DB2 数据源（更短的连接列）将 DB2 整数列和 Oracle 整数列连接；但是，如果 DB2 INTEGER 数据类型可容纳此 Oracle 整数列的域，那么用 ALTER NICKNAME 语句更改其本地数据类型，以便该连接可在 DB2 数据源进行。

列选项：使用 SQL 语句 ALTER NICKNAME 来添加或更改昵称的列选项。

使用 *varchar_no_trailing_blanks* 选项来标识不包含结尾空白的列。当检查对这样标明的列执行的所有操作时，编译器下推分析步骤将会考虑此信息。根据该指示，DB2 可能生成另一个形式等效的谓词以用于发送到数据源的远程 SQL 语句。用户可能会看到依靠数据源求值的另一个谓词，但最终结果应是等效的。

使用 *numeric_string* 选项来指示该列中的值是否总是不带结尾空白的数值。

下表描述这些选项。

表 63. 列选项及其设置

选项	有效设置	缺省设置
numeric_string	**"Y"** 是，此列只包含数字数据串。要点：如果列只包含数字串且后面带结尾空白，不要指定"Y"。 **"N"** 否，此列不限于数字数据串。 如果将列的 numeric_string 设置为"Y"，那么将告知优化器此列不包含可能干扰列数据排序的空格。当数据源的整理顺序与 DB2 的不同时，此选项很有用。用此选项标记的那些列不会因为整理顺序不同而从本地（数据源）求值中排除。	"N"
varchar_no_trailing_blanks	指定此数据源是否使用非空格填充的 VARCHAR 比较语义。对于不包含结尾空白的变长字符串，某些 DBMS 的非空格填充比较语义返回与 DB2 的比较语义相同的结果。如果确认在一个数据源中的所有 VARCHAR 表/视图列都不包含结尾空白，考虑对数据源将此服务器选项设置为"Y"。此选项经常用于 Oracle 数据源。确保考虑可能有昵称的所有对象（包括视图）。 **"Y"** 此数据源的非空格填充比较语义与 DB2 的类似。 **"N"** 此数据源的非空格填充比较语义与 DB2 的不同。	"N"

影响下推机会的查询特征

查询可引用一个 SQL 运算符，它可能涉及多个数据源中的昵称。该操作必须在 DB2 中执行以使用一个运算符（例如，集合运算符 UNION）来组合来自两个被引用的数据源的结果。不能在远程数据源直接对该运算符求值。

分析在何处对联合查询求值的准则

DB2 提供了两个实用程序来说明在何处对查询进行求值:

- Visual Explain。用 **db2cc** 命令启动。可用它来查看查询访问方案图。每个运算符的执行位置都包括在该运算符详细显示的内容中。

  如果下推查询，应会看到 RETURN 运算符。RETURN 运算符是标准的 DB2 运算符。对于从昵称选择数据的 SELECT 语句，还会看到 SHIP 运算符。SHIP 运算符是联合数据库操作所独有的。它可以更改数据流的服务器属性，并可将本地运算符与远程运算符区分开来。该 SELECT 语句是使用数据源支持的 SQL 语言生成的。它可包含该数据源的任何有效的查询。

  如果可以完全将 INSERT、DELETE 或 UPDATE 查询下推至远程数据库，那么 SHIP 语句可能不会出现在访问方案中。会对 RETURN 运算符显示所有以远程方式执行的 INSERT、UPDATE 和 DELETE 语句。但是，如果不能完全下推查询，SHIP 运算符会显示哪些操作是以远程方式执行的。

- SQL 说明。用 **db2expln** 或 **dynexpln** 命令启动。可用它来查看文本格式的访问方案策略。

了解为什么在数据源或 DB2 中对查询求值

在寻求增加下推机会的方法时，请考虑以下关键问题:

- 为什么不对此谓词远程求值？

  当谓词选择性很强并因此可用于过滤行和减少网络流量时，会出现此问题。远程谓词求值还影响是否可对同一数据源的两个表之间的连接进行远程求值。

  要检查的范围包括:

  – 子查询谓词。此谓词包含与另一个数据源相关的子查询吗？此谓词包含涉及此数据源不支持的 SQL 运算符的子查询吗？并非所有数据源都支持子查询谓词中的集合运算符。
  – 谓词函数。此谓词包含此远程数据源不能求值的函数吗？关系运算符也被归类为函数。
  – 谓词绑定需求。如果进行远程求值，此谓词要求某些值的绑入吗？如果是这样，它会违反此数据源的 SQL 限制吗？
  – 全局优化。优化器可能已经决定本地处理更节省成本。

- 为什么不对 GROUP BY 运算符进行远程求值？

  有几个方面可检查:

  – 对 GROUP BY 运算符的输入进行远程求值吗？如果回答是否定的，那么检查输入。
  – 数据源对此运算符有任何限制吗？例如:
    - GROUP BY 的有限项数
    - 组合的 GROUP BY 项的有限字节计数
    - 列规格仅存在于 GROUP BY 列表上
  – 数据源支持此 SQL 运算符吗？
  – 全局优化。优化器可能已经决定本地处理更节省成本。
  – GROUP BY 运算符子句包含字符表达式吗？如果包含，验证远程数据源是否与 DB2 一样区分大小写。

- 为什么不对集合运算符进行远程求值？

  有几个方面可检查:

  – 它的两个操作数完全在同一远程数据源中求值吗？答案应该是肯定的，如果不是，检查每个操作数。
  – 数据源对此集合运算符有任何限制吗？例如，大对象或长整数字段是否是此特定集合运算符的有效输入？

- 为什么不对 ORDER BY 操作进行远程求值？

  考虑以下问题:

  – 对 ORDER BY 操作的输入进行远程求值吗？如果回答是否定的，那么检查输入。
  – ORDER BY 子句包含字符表达式吗？如果是，那么远程数据源与 DB2 没有相同的整理顺序或大小写区分方式吗？

– 数据源对此运算符有任何限制吗？例如，ORDER BY 的项数受限制吗？数据源限制 ORDER BY 列表的列规格吗？

在联合数据库中的远程 SQL 生成和全局优化

对于使用关系昵称的联合数据库查询，访问策略可能会将原来的查询分为一组远程查询单元，然后组合结果。此远程 SQL 生成帮助产生查询的全局优化访问策略。

优化器使用下推分析的输出来决定是要在 DB2 以本地方式还是在数据源以远程方式对每个操作进行求值。它根据其成本模型的输出作出决定，该输出不但包括对操作进行求值的成本，还包括在 DB2 和数据源之间传送数据或消息的成本。

虽然目标是产生优化查询，但以下主要因素影响全局优化的输出，因而也会影响查询性能。

- 服务器特征
- 昵称特征

影响全局优化的服务器特征和选项

以下数据源服务器因素可能影响全局优化:

- CPU 速度的相对比率

  使用 *cpu_ratio* 服务器选项来指定与 DB2 CPU 相比，数据源 CPU 速度的快慢。低比率指示数据源计算机 CPU 比 DB2 计算机 CPU 快。如果比率较低，那么 DB2 优化器很可能会考虑将需要占用大量 CPU 资源的操作下推到数据源。

- I/O 速度的相对比率

  使用 *io_ratio* 服务器选项来指示与 DB2 系统相比，数据源系统 I/O 速度快多少或慢多少。低比率指示数据源工作站 I/O 速度比 DB2 工作站 I/O 速度快。如果比率较低，那么 DB2 优化器考虑将占用大量 I/O 资源的操作下推到数据源。

- DB2 和数据源之间的通信速率

  使用 *comm_rate* 服务器选项来指示网络容量。低速率（它指示 DB2 和数据源之间的网络通信较慢）促使 DB2 优化器减少向此数据源发送或从此数据源接收的消息数。如果将该速率设置为 0，优化器将创建需要最小网络流量的访问方案。

- 数据源整理顺序

  使用 *collating_sequence* 服务器选项来指定数据源整理顺序是否与本地 DB2 数据库整理顺序匹配。如果未将此选项设置为“Y”，那么优化器将认为从此数据源检索的数据是未排序的。

- 远程方案提示

  使用 *plan_hints* 服务器选项来指定应在数据源生成或使用方案提示。缺省情况下，DB2 不将任何方案提示发送至数据源。

  方案提示是为数据源优化器提供额外信息的语句分段。对于某些查询，此信息可以提高性能。方案提示可帮助数据源优化器决定是否使用索引、使用哪个索引或使用哪种表连接顺序。

  如果启用方案提示，那么发送到数据源的查询将包含其他的信息。例如，带方案提示发送到 Oracle 优化器的语句可能如下所示:

```
SELECT /*+ INDEX (table1, t1index)*/
         col1
         FROM table1
```

该方案提示是字符串 `/*+ INDEX (table1, t1index)*/`。

- DB2 优化器知识库中的信息

  DB2 有一个包含本地数据源数据的优化器知识库。DB2 优化器不会生成特定 DBMS 无法生成的远程访问方案。换句话说，DB2 避免生成远程数据源的优化器不能理解或接受的方案。

影响全局优化的昵称特征

以下特定于昵称的因素会影响全局优化。

索引注意事项：DB2 可以使用数据源中的索引信息来优化查询。由于此原因，DB2 可用的索引信息应是最新的，这点很重要。昵称的索引信息最初是在创建昵称时获取的。不收集视图昵称的索引信息。

创建昵称的索引规范：可为昵称创建索引规范。索引规范在目录中构建索引定义（而不是实际的索引），以供 DB2 优化器使用。使用 CREATE INDEX SPECIFICATION ONLY 语句来创建索引规范。基于昵称创建索引规范的语法与在本地表上创建索引的语法相似。

在下列情况考虑创建索引规范:

- 在创建昵称期间，DB2 无法检索数据源的任何索引信息。
- 需要视图昵称的索引。
- 想鼓励 DB2 优化器将特定的昵称用作嵌套循环连接的内部表。如果连接列不存在索引，用户可创建一个。

在基于视图的昵称发出 CREATE INDEX 语句之前，考虑您是否需要。如果视图是对一个有索引的表执行的简单 SELECT，那么在本地基于昵称创建与数据源表上的索引匹配的索引可显著改善查询性能。但是，如果是在本地对并非由简单的选择语句组成的视图（例如，通过连接两个表创建的视图）创建索引，可能会损害查询性能。例如，对通过两个表的连接构成的视图创建索引，优化器可能会选择该视图作为嵌套循环连接中的内部元素。该查询的性能会很差，因为要对连接求值几次。替代方法是为数据源视图中引用的每个表创建昵称，并在 DB2 中创建一个局部视图来引用这两个昵称。

目录统计注意事项：系统目录统计信息描述相关列中昵称的总大小和值的范围。当优化器计算出为处理包含昵称查询的最低费用路径时，使用这些统计信息。昵称统计信息与表统计信息存储在同一目录视图中。

虽然 DB2 可检索数据源中保存的统计数据时，它不能自动检测对数据源的现有统计数据的更新。而且，DB2 不能对数据源中的对象处理对象定义或结构中的更改，例如，添加一列到对象。如果更改了对象的统计数据或结构数据，那么有两个选择:

- 在数据源处运行 RUNSTATS 的等效命令。然后删除当前昵称并重新创建。如果结构信息已更改，那么使用此方法。
- 手动更新 SYSSTAT.TABLES 视图中的统计信息。此方法需要的步骤更少，但如果结构信息更改，它将不起作用。

联合数据库查询的全局分析

提供的以下两项实用程序显示全局访问方案:

- Visual Explain。从“控制中心”启动它，或执行 *db2cc* 命令，该命令启动“控制中心”。使用 Visual Explain 来查看查询访问方案图。每个运算符的执行位置都包括在该运算符详细显示的内容中。根据查询类型，还可在 SHIP 或 RETURN 运算符中找到为每个数据源生成的远程 SQL 语句。通过检查每个运算符的详细信息，可以查看 DB2 优化器作为每个运算符的输入和输出所估计的行数。还可了解执行每个运算符的估计成本，包括通信成本在内。
- SQL 说明。用 *db2expln* 或 *dynexpln* 命令启动。可用 SQL 说明来查看文本格式的访问方案策略。虽然 SQL 说明没有提供成本信息，您仍然可以获得远程优化器为远程说明功能所支持的那些数据源生成的访问方案。

了解 DB2 优化判定

考虑下列要研究的优化问题和关键范围以改善性能:

- 为什么不对同一数据源的两个昵称之间的连接进行远程求值？

  要检查的范围包括:

  – 连接操作。数据源能支持它们吗？
  – 连接谓词。能在远程数据源对连接谓词求值吗？如果回答是否定的，那么检查连接谓词。
  – 连接结果（Visual Explain 生成的）中的行数。该连接产生比组合的两个昵称更大的行集合吗？这些数值有意义吗？如果回答是否定的，那么考虑手动更新昵称统计信息（SYSSTAT.TABLES）。

- 为什么不对 GROUP BY 运算符进行远程求值？

  要检查的范围包括:

  – 运算符语法。验证可在远程数据源对该运算符求值。
  – 行数。使用 Visual Explain 检查 GROUP BY 运算符输入和输出中的估计行数。这两个数非常接近吗？如果是，那么 DB2 优化器认为在本地对此 GROUP BY 求值更有效。而且，这两个数值有意义吗？如果回答是否定的，那么考虑手动更新昵称统计信息（SYSSTAT.TABLES）。

- 为什么远程数据源没有完全对该语句求值？

  DB2 优化器执行基于成本的优化。即使下推分析指示每个运算符都可在远程数据源求值，优化器仍根据其成本估计来生成全局优化方案。有很多因素可对该方案产生影响。例如，即使远程数据源可处理原始查询中的每个操作，但其 CPU 速度远低于 DB2 的 CPU 速度，因此在 DB2 执行这些操作可能会更有利。如果结果不理想，验证 SYSCAT.SERVEROPTIONS 中的服务器统计信息。

- 为什么由优化器生成的，且完全在远程数据源求值的方案，其性能比直接在远程数据源中执行的原始查询的性能要差很多？

  要检查的范围包括:

  – DB2 优化器生成的远程 SQL 语句。确保它等于原来的查询。检查谓词排序更改。一个好的查询优化器不应对查询的谓词顺序敏感；但不幸的是，并非所有的 DBMS 优化器都一样，因此很可能远程数据源的优化器会根据输入谓词排序生成不同的

方案。如果是这样，这是远程优化器内在的问题。考虑修改 DB2 输入的谓词排序，或与远程数据源的服务机构联系以寻求帮助。

还要检查是否替换过谓词。一个好的查询优化器不应对等效的谓词替换敏感；但不幸的是，并非所有的 DBMS 优化器都一样，因此远程数据源的优化器可能会根据输入谓词生成不同的方案。例如，某些优化器不能为谓词生成传递闭合语句。

– 返回的行数。可从 Visual Explain 获得此数值。如果查询返回大量的行，那么网络流量是一个潜在的瓶颈。
– 其他函数。与原始查询相比，远程 SQL 语句包含其他函数吗？可能生成一些其他函数以转换数据类型。确保它们是必需的。

数据访问方法

查询优化器在编译一条 SQL 或 XQuery 语句时估计满足查询的不同方法的执行成本。根据它的估计，优化器选择优化访问方案。*访问方案*指定解析一条 SQL 或 XQuery 语句所需的操作顺序。当绑定一个应用程序时，会创建一个*程序包*。此程序包包含该应用程序中所有静态 SQL 和 XQuery 语句的访问方案。动态 SQL 和 XQuery 语句的访问方案在执行应用程序时创建。

有两种方法来访问表中的数据:

- 顺序扫描整个表
- 通过首先访问表上的索引来定位特定表行

要产生查询请求的结果，根据谓词的条款选择行，通常在 WHERE 子句中说明这些条款。连接访问表中选择的行来产生结果集，并可以通过分组或排序输出来进一步处理结果集。

通过索引扫描的数据访问

在数据库管理器因下列任何一个原因访问一个索引时，执行*索引扫描*:

- 在访问基本表前，缩小合格行的集合（通过扫描该索引特定范围内的行）。索引*扫描范围*（扫描的起点和终点）由比较索引列的查询中的值确定。
- 排序输出。
- 直接检索请求的列数据。如果所有请求的数据位于该索引中，那么不需要访问索引表。这就称为*纯索引访问*。

如果使用 ALLOW REVERSE SCANS 选项创建索引，那么也可以按与定义的方向相反的方向来执行扫描。

注: 如果未创建适当的索引或如果索引扫描的成本将更高，那么优化器选择表索引。当表不大、索引集群比率低或查询请求大多数表行时，索引扫描的成本可能更高。要查明访问方案使用表扫描还是索引扫描，使用说明工具。

定界范围的索引扫描

要确定索引是否可以用于特定查询，优化器从索引第一列开始对每列求值，以了解它是否可用来满足等式和 WHERE 子句中的其他谓词。*谓词*是 WHERE 子句中搜索条件的元素，它表示或隐含比较运算。在下列情况下，可使用谓词来定界索引扫描范围的:

- 测试是否等于一个常数、主变量、值为常数的表达式或关键字
- 测试“为空”还是“不为空”
- 测试是否等于基本子查询（它是不包含 ANY、ALL 或 SOME 的子查询），且该子查询没有相关列引用其直接父查询块（即，此子查询是其子选择的 SELECT）。
- 测试严格和相容不等式。

以下示例举例说明索引何时可以用来限制某个范围:

- 考虑具有下列定义的索引:

```
INDEX IX1:  NAME    ASC,
                        DEPT    ASC,
                        MGR     DESC,
                        SALARY  DESC,
                        YEARS   ASC
```

  在此情况下，下列谓词可以用来限制索引 IX1 的扫描范围:

```
WHERE  NAME  = :hv1
    AND  DEPT = :hv2
```

  或者

```
WHERE  MGR  = :hv1
       AND  NAME = :hv2
       AND  DEPT = :hv3
```

  注意在第二个 WHERE 子句中，谓词不必按键列在索引中出现的相同顺序来指定。尽管示例使用了主变量，但其他变量（例如，参数标记、表达式或常数）将会有相同的效果。

- 考虑使用 ALLOW REVERSE SCANS 参数创建的单个索引。这类索引支持以创建索引时定义的方向扫描以及支持以相反的方向扫描。该语句可能类似于如下所示:

```
CREATE INDEX iname ON tname (cname DESC) ALLOW REVERSE SCANS
```

  在这种情况下，根据 cname 中的降序值来构建索引（iname）。允许反向扫描后，尽管列中的索引被定义为按降序扫描，但也可以按升序进行扫描。实际是否以两个方向使用索引不是由您控制的，而是由优化器在创建和考虑访问方案时控制的。

在如下的 WHERE 子句中，将只使用 NAME 和 DEPT 的谓词来定界索引扫描的范围，而不使用 SALARY 或 YEARS 的谓词:

```
WHERE  NAME  = :hv1
    AND  DEPT = :hv2          AND  SALARY = :hv4
    AND  YEARS  = :hv5
```

这是因为有一个键列（MGR）将这些列与最前面两个索引键列分开，因此排序将关闭。但是，一旦该范围由 NAME = :hv1 和 DEPT = :hv2 谓词确定，那么可根据其余索引键列来对其余谓词求值。

测试不等式的索引扫描

某些不等式谓词可以对索引扫描的范围定界。有两种类型的不等式谓词:

- 严格不等式谓词

  可用作范围定界谓词的严格不等式运算符是大于（>）和小于（<）。

只考虑一个具有严格不等式谓词的列用于定界索引扫描的范围。在以下示例中，NAME 和 DEPT 列上的谓词可用于定界该范围，但不能使用 MGR 列上的谓词。

```
WHERE  NAME  = :hv1
    AND  DEPT   > :hv2
    AND  DEPT   < :hv3
    AND  MGR    < :hv4
```

- 相容不等式谓词

  以下是可用作范围定界谓词的相容不等式运算符:

  – >= 和 <=

  – BETWEEN

  – LIKE

  要定界索引扫描的范围，将考虑多个具有相容不等式谓词的列。在以下示例中，所有谓词都可用于定界索引扫描的范围:

```
WHERE  NAME  = :hv1
    AND  DEPT  >= :hv2
    AND  DEPT  <= :hv3
    AND  MGR   <= :hv4
```

  要进一步说明此示例，假定 :hv2 = 404、:hv3 = 406 和 :hv4 = 12345。数据库管理器将使用该索引扫描部门 404 和 405 的全部，但当它扫描到职员号（MGR 列）大于 12345 的第一个经理时它将停止扫描部门 406。

排序数据索引扫描

如果查询要求按排序顺序输出，而且，如果排序列从第一个索引键列开始连续出现在该索引中，那么可使用索引来对数据排序。确定顺序或排序可由类似如下的操作产生: ORDER BY、DISTINCT、GROUP BY、“= ANY”子查询、“> ALL”子查询、“< ALL”子查询、INTERSECT、EXCEPT 或 UNION。一个例外是当比较索引键列是否等于“常数值”时，该常数值是值为常数的任何表达式。在这种情况下，排序列不能是第一个索引键列。

请考虑以下查询:

```
WHERE NAME = 'JONES'
       AND DEPT = 'D93'
   ORDER BY MGR
```

对于此查询，可使用索引来对行排序，因为 NAME 和 DEPT 将始终是相同的值，因此将被排序。也就是说，前导 WHERE 和 ORDER BY 子句等效于:

```
WHERE NAME = 'JONES'
       AND DEPT = 'D93'
   ORDER BY NAME, DEPT, MGR
```

也可使用唯一索引来降低排序顺序需求。考虑下列索引定义和 ORDER BY 子句:

```
UNIQUE INDEX IX0:  PROJNO  ASC
   SELECT PROJNO, PROJNAME, DEPTNO
        FROM PROJECT
   ORDER BY PROJNO, PROJNAME
```

不需要根据 PROJNAME 列再进行排序，因为 IX0 索引确保 PROJNO 是唯一的。此唯一性确保对于每个 PROJNO 值只有一个 PROJNAME 值。

索引访问的类型

在某些情况下，优化器可能发现可以从表上的索引检索来自表中查询需要的所有数据。在其他情况下，优化器可以使用多个索引来访问表。对于范围集群表，可以通过“虚拟”索引来访问数据，“虚拟”索引将计算数据记录的位置。

纯索引访问

在某些情况下，可从索引检索所有必需的数据，而不用访问表。这称为*纯索引访问*。

要举例说明纯索引访问，考虑下列索引定义:

```
INDEX IX1:  NAME    ASC,
                        DEPT    ASC,
                        MGR     DESC,
                        SALARY  DESC,
                        YEARS   ASC
```

仅访问该索引而不用读取基本表，就可满足以下查询:

```
SELECT NAME, DEPT, MGR, SALARY
       FROM EMPLOYEE        WHERE NAME = 'SMITH'
```

但是，经常需要未出现在该索引中的列。要获取这些列的数据，必须读取表行。要允许优化器选择纯索引访问，创建具有包含列的唯一索引。例如，考虑下列索引定义:

```
CREATE UNIQUE INDEX IX1 ON EMPLOYEE
         (NAME ASC)
           INCLUDE (DEPT, MGR, SALARY, YEARS)
```

此索引实现 NAME 列的唯一性，并且也存储和维护 DEPT、MGR、SALARY 和 YEARS 列的数据，它通过仅访问索引来使得满足下列查询:

```
SELECT NAME, DEPT, MGR, SALARY
        FROM EMPLOYEE            WHERE NAME='SMITH'
```

但是，当考虑将 INCLUDE 列添加至索引时，考虑附加存储空间和维护成本是否合理。如果很少执行通过仅读取这种索引就可以满足的查询，那么成本可能不合理。

多个索引访问

优化器可以选择扫描同一表上的多个索引来满足 WHERE 子句的谓词。例如，考虑下列两个索引定义:

```
INDEX IX2:  DEPT    ASC
   INDEX IX3:  JOB     ASC,
                           YEARS   ASC
```

下列谓词可以通过使用两个索引来满足:

```
WHERE DEPT = :hv1
    OR (JOB   = :hv2
    AND YEARS >= :hv3)
```

扫描索引 IX2 产生满足 DEPT = :hv1 谓词的行标识（RID）的列表。扫描索引 IX3 产生满足 JOB = :hv2 AND YEARS >= :hv3 谓词的 RID 的列表。在访问该表之前，组合 RID 的这两个列表，并除去重复值。这称为*索引 OR 运算*。

索引 OR 运算也可用于 IN 子句中指定的谓词，如下示例中所示:

```
WHERE DEPT IN (:hv1, :hv2, :hv3)
```

尽管索引 OR 运算的目的是消去重复的 RID，但是，*索引 AND 运算*的目标是查找公共 RID。可在符合下列条件的应用程序内执行索引 AND 运算：在同一个表内的对应列上创建多个索引，且对该表运行使用多个 AND 谓词的一个查询。在这类查询中对具有索引的每一列执行多索引扫描，将产生被散列以创建位图的值。第二个位图用于探测第一个位图，以生成合格行，可读取这些行以创建最终返回的数据集。

例如，假定有下列两个索引定义：

```
INDEX IX4: SALARY   ASC
   INDEX IX5: COMM     ASC
```

可使用这两个索引来解析下列谓词：

```
WHERE SALARY BETWEEN 20000 AND 30000
         AND COMM BETWEEN 1000 AND 3000
```

在此示例中，扫描索引 IX4 产生一个满足 SALARY BETWEEN 20000 AND 30000 谓词的位图。扫描 IX5 和探测 IX4 的位图产生满足这两个谓词的合格 RID 的列表。这称为“动态位图 AND 运算”。此操作仅在满足下列条件时才执行：该表有足够大的基数，且列在合格范围内有足够多的值，或者如果使用等式谓词，那么列具有足够多的重复值。

要在扫描多个索引时实现动态位图的性能效益，可能需要更改排序堆大小（*sortheap*）数据库配置参数和排序堆阈值（*sheapthres*）数据库管理器配置参数的值。

当在访问方案中使用动态位图时，需要其他的排序堆空间。当将 *sheapthres* 设置为相对接近 *sortheap* 时（也就是，每个并发查询占用的空间减少两三倍），那么具有多个索引访问的动态位图必须以比优化器所预料的少得多的内存来运行。解决方案是增大相对于 *sortheap* 的 *sheapthres* 的值。

注： 优化器在访问单个表时不将索引 AND 运算和索引 OR 运算组合在一起。

范围集群表中的索引访问

与标准表不同，范围集群表不需要像传统的 B 型树索引那样将键值映射至某行的物理索引。但是，它利用列域的顺序特性并使用运作映射来在表中生成给定行的位置。在此映射的最简单的示例中，范围内的第一个键值是表中的第一行，范围内的第二个值是表中的第二行，依此类推。

优化器使用表的范围集群属性来根据最佳集群索引（它的唯一成本就是计算范围集群函数）来生成访问方案。保证了表中的行的集群，这是因为范围集群表保留了它们的原始键值排序。

索引访问和集群比率

当它选择访问方案时，优化器估计将所需要的页从磁盘访存至缓冲池所需要的 I/O 次数。此估计包括预测缓冲池的使用情况，因为不需要其他 I/O 来读取已在缓冲池内一页中的行。

对于索引扫描，系统目录表（SYSCAT.INDEXES）中的信息帮助优化器估计将数据页读入缓冲池的 I/O 成本。它使用 SYSCAT.INDEXES 表中以下列中的信息：

- CLUSTERRATIO 信息指示表数据关于此索引的集群度。该数越高，按索引键顺序排序的行越好。如果表行接近索引键顺序，那么在数据页位于缓冲区时可以从该页中

读取行。如果此列的值为 -1，那么优化器使用 PAGE_FETCH_PAIRS 和 CLUSTERFACTOR 信息（如果它可用的话）。

- PAGE_FETCH_PAIRS 包含用于模拟将数据页读入各种大小的缓冲池所需的 I/O 数的多对数字以及 CLUSTERFACTOR 信息。仅当您使用 DETAILED 子句对索引执行 RUNSTATS 时，才对这些列收集数据。

如果索引集群统计信息不可用，那么优化器使用缺省值，在这种情况下假定数据与该索引的集群关系不强。

数据相对于索引的集群度可大大影响性能，您应尝试保持该表上的一个索引的集群率接近 100%。

一般情况下，只能有一个索引可达到百分之百的集群，但下列情况除外：其键是集群索引键的超集的那些索引，或者在两个索引的键列之间存在实际相关的那些索引。

当您重组表时，您可指定一个索引，它将用于集群这些行并试图在插入处理期间保留此特征。因为更新和插入可能降低表相对于索引的集群率，所以您可能需要定期重组该表。要降低对由于 INSERT、UPDATE 和 DELETES 而频繁更改的表重组的频率，当改变表时使用 PCTFREE 参数。这允许把附加的插入与现有的数据集群在一起。

连接

*连接*是根据信息的某些公共域从两个或多个表组合信息的过程。当对应行中的信息符合连接条件时，一个表中的行就会与另一个表中的行配对。

例如，考虑下列两个表:

Table1

PROJ	PROJ_ID
A	1
B	2
C	3
D	4

Table2

PROJ_ID	名称
1	Sam
3	Joe
4	Mary
1	Sue
2	Mike

要将标识列具有相同值的 Table1 和 Table2 连接，使用下列 SQL 语句:

```
SELECT PROJ, x.PROJ_ID, NAME
  FROM TABLE1 x, TABLE2 y
  WHERE x.PROJ_ID = y.PROJ_ID
```

此查询得到下列一组结果行:

PROJ	PROJ_ID	名称
A	1	Sam
A	1	Sue
B	2	Mike
C	3	Joe
D	4	Mary

根据连接谓词是否存在，以及所涉及到的、由表和索引统计信息来确定的各种成本，优化器选择下列其中一种连接方法:

- 嵌套循环连接
- 合并连接
- 散列连接

当连接两个表时，将一个表选择为外部表，而将另一个表选为内部表。首先访问外部表，并只扫描它一次。是否扫描内部表多次取决于连接的类型和存在的索引。即使一个查询连接两个以上表，优化器一次也只连接两个表。如果有必要，创建临时表来保存中间结果。

可以提供显式连接运算符（如 INNER 或 LEFT OUTER JOIN）来确定如何在连接中使用表。然而，以这种方式改变查询之前，应允许优化器确定如何连接表。然后分析查询性能来决定是否要添加连接操作程序。

连接方法

当查询需要连接表时，优化器可以选择三种基本连接策略的其中一种。

- 嵌套循环连接
- 合并连接
- 散列连接

这些方法在以下节中描述。

嵌套循环连接

嵌套循环连接是用下面两种方式的其中一种方式执行的:

- 对于外部表中每个被访问的行，扫描内部表

  例如，考虑表 T1 和 T2 中的列 A 具有下列值:

外部表 T1: 列 A	内部表 T2: 列 A
2	3
3	2
3	2
	3
	1

  要执行嵌套循环连接，数据库管理器执行下列步骤:

  1. 从 T1 中读取第一行。A 的值是“2”
  2. 扫描 T2，直到发现一个匹配项（“2”），然后连接这两行
  3. 扫描 T2，直到发现下一个匹配项（“2”），然后连接这两行
  4. 扫描 T2，直至该表结尾
  5. 返回至 T1，并读取下一行（“3”）
  6. 从第一行开始，扫描 T2，直到发现匹配项（“3”），然后连接这两行
  7. 扫描 T2，直到发现下一个匹配项（“3”），然后连接这两行
  8. 扫描 T2，直至该表结尾
  9. 返回至 T1，并读取下一行（“3”）
  10. 像以前一样扫描 T2，连接匹配（“3”）的所有行。

- 对于外部表中每个被访问的行，在内部表上执行索引查找。

  如果存在下列形式的谓词，那么此方法可用于指定的谓词:

  ```
  expr(outer_table.column)  relop  inner_table.column
  ```

  其中，relop 是比较运算符（例如，=、>、>=、< 或 <=），而 expr 是基于外部表的有效表达式。请考虑以下示例:

  ```
  OUTER.C1 + OUTER.C2 <= INNER.C1
  OUTER.C4 < INNER.C3
  ```

  此方法可以显著减少在每次访问外部表时要在内部表中访问的行数，虽然这取决于许多因素，包括连接谓词的选择性。

当它对嵌套循环连接求值时，优化器在执行连接前也决定是否对外部表排序。如果它根据连接列对外部表排序，那么可以减少从磁盘访问内部表的页的读操作次数，因为很可能这些页已在缓冲池中。如果该连接使用高度集群的索引来访问内部表，且如果外部表已排序，那么可将访问的索引页的数目减至最小。

另外，如果优化器期望该连接将执行一个成本更高的、更新的排序，那么优化器也可选择在连接前执行排序。要支持 GROUP BY、DISTINCT、ORDER BY 或合并连接，可能必需更新的排序。

合并连接

合并连接（有时称为*合并扫描连接*或*排序合并连接*）需要如下格式的一个谓词 table1.column = table2.column。这称为*等式连接谓词*。合并连接需要连接列的已排序输入，这个输入可通过索引访问或通过排序来获得。如果连接列是 LONG 字段列或大对象（LOB）列，那么不能使用合并连接。

在合并连接中，同时扫描已连接的表。合并连接的外部表只扫描一次。除非外部表中出现重复的值，否则，内部表也只扫描一次。如果有重复的值出现，那么可能再次扫描内部表中的一组行。例如，如果表 T1 和 T2 中的列 A 有下列值:

外部表 T1：列 A	内部表 T2：列 A
2	1
3	2
3	2
	3
	3

要执行合并连接，数据库管理器执行下列步骤:

1. 从 T1 中读取第一行。A 的值是“2”。
2. 扫描 T2，直到发现匹配项，然后连接这两行。
3. 连续扫描 T2，当列匹配时，连接那些行。
4. 当读取 T2 中的“3”时，返回至 T1 并读取下一行。
5. T1 中的下一个值是“3”，它与 T2 匹配，因此连接那些行。
6. 连续扫描 T2，当列匹配时，连接那些行。
7. 到达 T2 结尾。

8. 返回至 T1 以读取下一行 – 注意 T1 中的下一个值与 T1 中的上一个值相同，因此从 T2 中的第一个“3”开始再次扫描 T2。数据库管理器记住此位置。

散列连接

散列连接需要如下形式的一个或多个谓词: table1.columnX = table2.columnY，在该形式中，列类型相同。对于类型为 CHAR 的列，长度必须相同。对于类型为 DECIMAL 的列，精度和小数位必须相同。对于类型为 DECFLOAT 的列，精度必须相同。列类型不能是 LONG 字段列或大对象（LOB）列。

首先，扫描指定的 INNER 表，然后将行复制到从排序堆划出的内存缓冲区中，该排序堆由 *sortheap* 数据库配置参数指定。根据在 Join 谓词的列上计算的散列值，将内存缓冲区分为若干分区。如果 INNER 表的大小超过可用的排序堆空间，那么将所选分区中的缓冲区写入临时表。

当已处理内部表时，通过首先比较对连接谓词的列计算的散列值，扫描第二个（即 OUTER）表并将该表的行与 INNER 表的行匹配。如果 OUTER 行列的散列值与 INNER 行列的散列值匹配，那么比较实际连接谓词列值。

紧接着将对应于未写入临时表的那部分表的 OUTER 表行与内存中的 INNER 表行匹配。如果 INNER 表的对应部分已写入临时表，那么也将 OUTER 行写入临时表。最后，从临时表中读取匹配的表部分对，并比较它们的行的散列值，然后检查 Join 谓词。

为了获取散列连接的所有性能效益，可能需要更改 *sortheap* 数据库配置参数和 *sheapthres* 数据库管理器配置参数的值。

为了获取散列连接的所有性能效益，可能需要更改 *sortheap* 数据库配置参数和 *sheapthres* 数据库管理器配置参数的值。

如果可以避免散列循环和溢出到磁盘，散列连接性能最佳。要调整散列连接性能，估计对 *sheapthres* 可用的最大内存量，然后调整 **sortheap** 参数。增大它的设置，直到尽可能避免散列循环和磁盘溢出，但不要达到由 *sheapthres* 参数指定的限制。

增大 *sortheap* 值也应提高具有多个类别的查询的性能。

选择最佳连接的策略

优化器使用各种方法来选择查询的最佳连接策略。这些方法包括下列搜索策略，它们由查询的优化级别来确定:

- 贪婪连接枚举
  - 可节省空间和时间
  - 单向枚举；即，一旦为两个表选择了一个连接方法，在进一步优化期间就不更改它
  - 当连接多个表时，可能丢失最佳访问方案。如果您的查询只连接两个或三个表，那么贪婪连接枚举选择的访问方案与动态规划连接枚举选择的访问方案相同。如果该查询在相同的列上有多个连接谓词（显式指定的或通过谓词传递闭包隐式生成的），那么更是如此。
- 动态规划连接枚举
  - 随着已连接表数的增加，空间和时间需求按指数规律增加
  - 可有效地、全面地搜索以获得最佳访问方案

– 类似于 DB2 OS/390 或 z/OS 版使用的策略。

连接枚举算法是对优化器所使用的方案组合数的一个重要决定因子。

星型模式连接

在一个查询中引用的表几乎总是由连接谓词联系起来。如果两个表已连接而没有连接谓词，那么形成了两个表的笛卡尔乘积。在笛卡尔乘积中，第一个表的每个合格行与第二个表的每个合格行连接，生成一个结果表，它由两个表大小的交叉乘积组成，该值一般很大。因为这类方案不可能执行得好，因此优化器甚至会避免确定这类访问方案的成本。

当将优化器级别设置为 9 时或在星型模式的特殊情况下，发生仅有的例外情况。*星型模式*包含一个称为事实表的中央表，而其他表称为维表。全部维表仅有一个单一连接，该连接将这些维表与事实表连接，而不管查询是什么。每个维表包含展开有关事实表中特定列的信息的其他值。典型的查询由多个引用维表中的值的本地谓词组成，并包含将维表与事实表连接的连接谓词。对于这些查询，在访问大的事实表之前，对多个小维表计算笛卡尔乘积可能有好处。此技术在多个连接谓词与一个多列索引匹配时有用。

DB2 可以识别对使用星型模式设计的、含至少两个维表的数据库的查询，并可以增加搜索空间来包括计算维表的笛卡尔乘积的可能方案。如果计算笛卡尔乘积的方案具有最低的估计成本，那么优化器选择该方案。

以上讨论的星型模式连接策略假设连接中使用了主键索引。另一个方案涉及到外键索引。如果事实表中的外键列是单列索引，且在所有维表中有相当高的选择性，可使用以下星型连接技术:

1. 通过下列操作处理每个维表:
   - 在维表和事实表上的外键索引之间执行半连接
   - 散列行标识（RID）值，以动态创建一个位图。
2. 对每个位图的先前位图使用 AND 谓词。
3. 在处理最后一个位图后，确定幸存的 RID。
4. 选择是否对这些 RID 排序。
5. 访存一个基本表行。
6. 将事实表与其每个维表重新连接，访问 SELECT 子句所需要的维表中的列。
7. 重新应用残留谓词。

此技术不需要多列索引。不要求事实表与维表之间存在显式引用完整性约束，尽管事实表与维表之间的关系应实际上按此方式联系。

星型连接技术创建和使用的动态位图需要排序堆内存，该内存的大小由“排序堆大小”（*sortheap*）数据库配置参数指定。

预输出连接

当优化器发现一个表中的每一行最多只需要与另一个表中的一行进行连接时，它就可以选择预输出连接。

当其中一个表的键列上有 Join 谓词时，就可以进行预输出连接。例如，考虑以下查询，它将返回职员及其直接主管的姓名。

```
SELECT EMPLOYEE.NAME AS EMPLOYEE_NAME, MANAGER.NAME AS MANAGER_NAME
FROM EMPLOYEE AS EMPLOYEE, EMPLOYEE AS MANAGER
WHERE EMPLOYEE.MANAGER_ID=MANAGER.ID
```

假定 ID 列是 EMPLOYEE 表中的一个键，并且每个职员最多只有一个主管，那么上述连接就可以避免在 MANAGER 表中继续搜索后续的匹配行。

当查询中有一个 DISTINCT 时，也可以进行预输出连接。例如，考虑以下查询，它将返回某一车型的售价超过 30000 美元的汽车的名称。

```
SELECT DISTINCT MAKE.NAME
FROM MAKE, MODEL
WHERE MAKE.MAKE_ID=MODEL.MAKE_ID AND MODEL.PRICE>30000
```

对于每种汽车，我们只需要确定它已生产的其中一个型号的售价是否超过了 30000 美元。不需要将一种汽车与它已生产的售价超过 30000 美元的所有型号都连接起来以保证查询结果的正确性。

当对 GROUP BY 添加 MIN 或 MAX 聚集函数时，也可以进行预输出连接。例如，考虑以下查询，它将返回股票代码以及在 2000 年之前发生以下情况的最近日期：特定股票的收盘价格至少比开盘价格高 10%:

```
SELECT DAILYSTOCKDATA.SYMBOL, MAX(DAILYSTOCKDATA.DATE) AS DATE
FROM SP500, DAILYSTOCKDATA
WHERE SP500.SYMBOL=DAILYSTOCKDATA.SYMBOL AND DAILYSTOCKDATA.DATE<'01/01/2000' AND
      DAILYSTOCKDATA.CLOSE / DAILYSTOCKDATA.OPEN >= 1.1
GROUP BY DAILYSTOCKDATA.SYMBOL
```

让我们将“合格的集合”定义为 DAILYSTOCKDATA 表中满足日期和价格要求的一个行集，并与 SP500 表中的特定股票代码进行连接。对于 SP500 表中的每个股票代码行，如果对 DAILYSTOCKDATA 表中的合格集合按 DATE 进行降序排列，那么只需要对每个股票代码返回合格集合中的第一行，因为第一行就是该特定股票的最近日期。对于保证查询的正确性来说，并不需要合格集合中的其他行。

组合表

当一对表的连接结果是一个称为*组合*表的新表时，此表通常成为具有另一个内部表的另一个连接的外部表。这称为“组合外连接”。在某些情况下，特别是在使用贪婪连接枚举技术时，使连接两个表的结果成为后续连接的内部表将很有用。当一个连接的内部表由连接两个或更多个表的结果组成时，此方案是一个“组合内连接”。例如，考虑如下查询:

```
SELECT COUNT(*)
FROM T1, T2, T3, T4
WHERE T1.A = T2.A AND
            T3.A = T4.A AND
            T2.Z = T3.Z
```

以下做法可能有用：将表 T1 和 T2 连接（T1xT2），然后将 T3 与 T4 连接（T3xT4），最后选择第一个连接结果作为外部表，选择第二个连接结果作为内部表。在最后的方案（（T1xT2）x（T3xT4））中，连接结果（T3xT4）称为组合内连接。根据查询优化级别，优化器对可以作为一个连接内部表的若干个表的最大数目设置不同的约束。优化级别 5、7 和 9 允许组合内连接。

分区数据库环境中重复的具体化查询表

重复具体化查询表通过允许数据库管理表数据的预先计算的值来提高分区数据库环境中频繁执行连接的性能。

考虑一个查询和重复具体表示例。作下列假定:

- SALES 表在多分区表空间 REGIONTABLESPACE 中，且在 REGION 列上分割。
- EMPLOYEE 和 DEPARTMENT 表在单一分区数据库分区组中。

根据 EMPLOYEE 表中的信息创建复制的具体化查询表。

```
CREATE TABLE R_EMPLOYEE
      AS (
              SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT
            FROM EMPLOYEE       )
  DATA INITIALLY DEFERRED REFRESH IMMEDIATE
  IN REGIONTABLESPACE
  REPLICATED;
```

要更新重复的具体化查询表的内容，运行下列语句:

```
REFRESH TABLE R_EMPLOYEE;
```

注: 使用 REFRESH 语句之后，您应该在复制表上运行 RUNSTATS，正如您在任何其他表上要运行的一样。

以下示例计算职员的销售额、部门总销售额和总计:

```
SELECT d.mgrno, e.empno, SUM(s.sales)
  FROM   department AS d, employee AS e, sales AS s
  WHERE  s.sales_person = e.lastname
        AND e.workdept = d.deptno
  GROUP BY ROLLUP(d.mgrno, e.empno)
  ORDER BY d.mgrno, e.empno;
```

如果不使用仅在一个数据库分区上的 EMPLOYEE 表，那么数据库管理器使用 R_EMPLOYEE 表，该表在存储 SALES 表的每个数据库分区上均有复制。这样性能将会增强，因为职员信息不必通过网络传送到每个数据库分区来计算该连接。

并置连接中的重复具体化查询表

重复的具体化查询表也可以有助于并置连接。例如，如果一个星型模式包含一个在 20 个节点上分布的大的事实表，那么在事实表和维表之间的连接是最有效的（如果并置了这些表）。如果所有表位于同一个数据库分区组中，那么对于一个并置连接，最多正确分区一个维表。其他的维表不能在并置连接中使用，因为事实表上的连接列与事实表的分布键不对应。

考虑在 C1 上分割称为 FACT (C1, C2, C3, ...) 的表；在 C1 上分割称为 DIM1 (C1, dim1a, dim1b, ...) 的表；在 C2 上分割称为 DIM2 (C2, dim2a, dim2b, ...) 的表；等等。

在此情况下，您会发现在 FACT 和 DIM1 之间的连接是最好的，因为并置了谓词 DIM1.C1 = FACT.C1。这两个表都在列 C1 上分割。

但是，不能并置 DIM2 和谓词 WHERE DIM2.C2 = FACT.C2 之间的连接，因为在列 C1 上分割 FACT，而不在列 C2 上分割 FACT。在此情况下，可以在事实表的数据库分区组中复制 DIM2，以便连接按本地方式在每个数据库分区上进行。

注: 此处复制的具体化查询表讨论与分区内数据库复制有关。交互数据库复制与预订、控制表以及位于不同数据库中和不同操作系统上的数据有关。

当创建复制的具体化查询表时，源表可以是数据库分区组中的单节点表或多节点表。在大多数情况下，复制的表不大，并且可以放置在单节点数据库分区组中。可以通过指定只有此表中列的子集，或通过指定使用的谓词行数，或通过使用两种方法来限制要复制的数据。为了复制具体化查询表有效，并不需要数据捕获选项。

也可以在多节点数据库分区组中创建复制具体化查询表，以便在所有数据库分区上创建源表的副本。如果向所有数据库分区广播此源表，那么大型事实表和维表之间的连接在此环境中更可能在本地执行。

未自动创建复制表上的索引。可以创建与源表上的索引不同的索引。但是，为防止未在源表上出现的约束违例，不能创建唯一索引或对复制的表使用约束。即使在源表上发生相同的约束，也禁止这些约束。

可以在查询中直接引用复制的表，但您不可以将 NODENUMBER() 谓词与复制表一起使用，来查看特定分区上的表数据。

使用说明工具来查看查询的访问方案是否使用了复制的具体化查询表。优化器所选择的访问方案是否使用复制的具体化查询表取决于连接所需要的信息。如果优化器确定向数据库分区组中的其他数据库分区广播原始源表比较便宜，那么优化器可能不使用复制具体化查询表。

分区数据库中的连接策略

在某些方面，连接策略在分区数据库环境中与非分区数据库环境中不同。可以将其他技术应用于标准连接方法以提高性能。

对于在一个分区数据库环境中进行的频繁连接所涉及的那些表，要考虑的一点是表并置。表并置提供了一种方法，即在一个分区数据库环境中，根据相同的分布键，可在同一个数据库分区中使用一个表中的数据查找另一个表中的数据。一旦被并置，要连接的数据可参与查询，而不必在执行查询活动时移动至另一个数据库分区。只将连接的答案集移动至协调程序节点。

表队列

分区数据库环境中连接技术的描述使用下列术语:

- 表队列

  在数据库分区之间或单一分区数据库中的处理器之间传送行的一种机制。
- 定向表队列

  一种表队列，其中，将行散列至其中一个接收数据库分区。
- 广播表队列

  一种表队列，其中，将行发送至所有接收数据库分区，但不散列这些行。

表队列用于下列情况下:

- 在使用分区间并行性时，将表数据从一个数据库分区传送至另一个
- 在使用分区内并行性时，传送一个数据库分区中的表数据

- 在使用单一分区数据库时，传送一个数据库分区中的表数据。

每个表队列按单方向传送数据。编译器决定何处需要使用表队列，并在方案中包括表队列。当执行方案时，数据库分区之间的连接会启动表队列。一旦进程结束，那么表队列关闭。

有几种类型的表队列:

- *异步表队列*。这些表队列称为异步，是因为它们在应用程序发出的任何 FETCH 之前读取行。当发出 FETCH 时，从表队列检索该行。

  当您在 SELECT 语句上指定 FOR FETCH ONLY 子句时，使用异步表队列。如果您只读取行，那么异步表队列会更快。
- *同步表队列*。这些表队列称为同步，是因为它们对应用程序发出的每个 FETCH 读取一行。在每个数据库分区中，将游标定位在要从该数据库分区读取的下一行上。

  当您未在 SELECT 语句上指定 FOR FETCH ONLY 子句时，使用同步表队列。在一个分区数据库环境中，如果您要更新行，那么数据库管理器将使用同步表队列。
- *合并表队列*。

  这些表队列维持顺序。
- *非合并表队列*。

  这些表队列也称为“正规”表队列。它们不保持顺序。
- *侦听器表队列*。

  这些表队列在关联子查询中使用。使用这种类型的表队列，将关联值向下传送至子查询，然后将结果向上传送回父查询块。

分区数据库环境中的连接方法

下列图形举例说明了分区数据库环境中的连接方法。

注: 在这些图表中，q1、q2、和 q3 表示示例中的表队列。引用的表分布在两个数据库分区中，以实施这些方案。箭头指示发送表队列的方向。协调程序节点是数据库分区 0。

并置连接

并置连接以本地方式在数据所在的数据库分区上发生。该数据库分区在完成连接以后将数据发送到其他数据库分区。要使优化器考虑并置连接，必须并置连接的表，且所有相应的分布键对必须参与等式连接谓词。

以下图形提供了一个示例。

注: 复制的具体化查询表提高了并置连接的可能性。

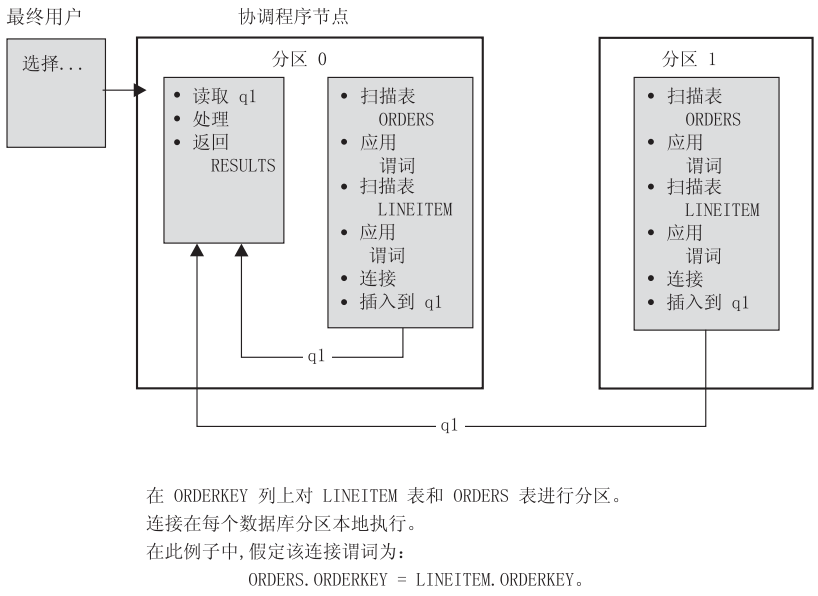


图 23. 并置连接示例

广播外部表连接

广播外部表连接是一个并行连接策略，如果在连接的表之间没有等式连接谓词，可使用该连接。在可以将它用作成本最低的连接方法的其他情况中，也可使用该连接。例如，当有一个很大的表和一个很小的表且没有在连接谓词列上对任何一个表进行分割时，可能会发生广播外部表连接。不分割这两个表，将更小的表广播至更大的表可能成本更低。以下图形提供了一个示例。

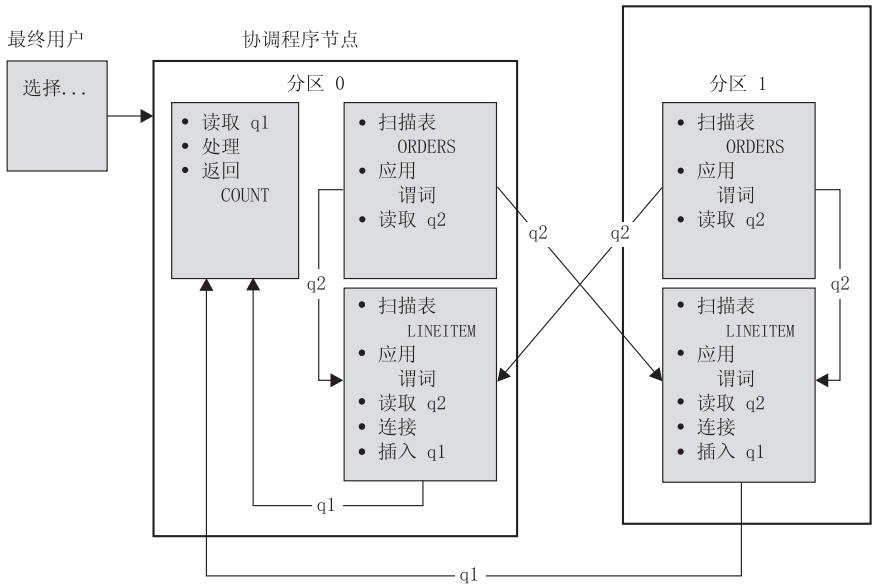
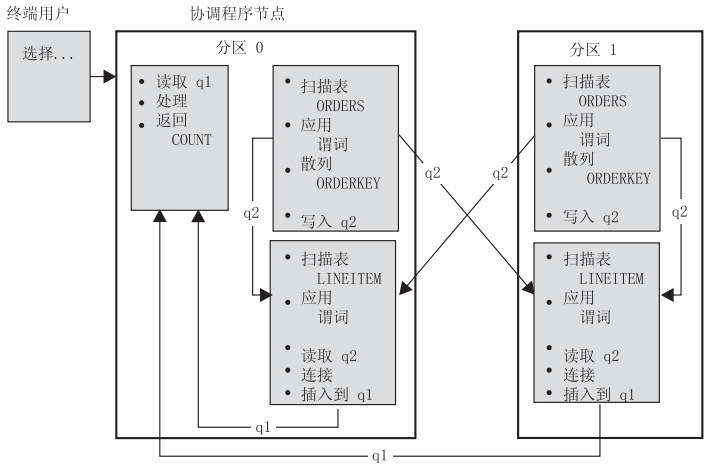


图 24. 广播外部表连接示例

定向外部表连接

在定向外部表连接策略中，根据内部表的分割属性将外部表的每一行发送至内部表的一部分。该连接在此数据库分区上进行。以下图形提供了一个示例。



LINEITEM 表在 ORDERKEY 列上进行分区。
ORDERS 表在另一列上进行分区。
ORDERS 表被散列并发送至正确的 LINEITEM 表数据库分区。
在此例子中,假定连接谓词为:
ORDERS.ORDERKEY = LINEITEM.ORDERKEY。

图 25. 定向外部表连接示例

定向内部表和外部表连接

在定向内部表和外部表策略中，根据连接列的值，将外部表和内部表的行定向到一组数据库分区。该连接在这些数据库分区上进行。以下图形提供了一个示例。以下图形中显示了一个示例。

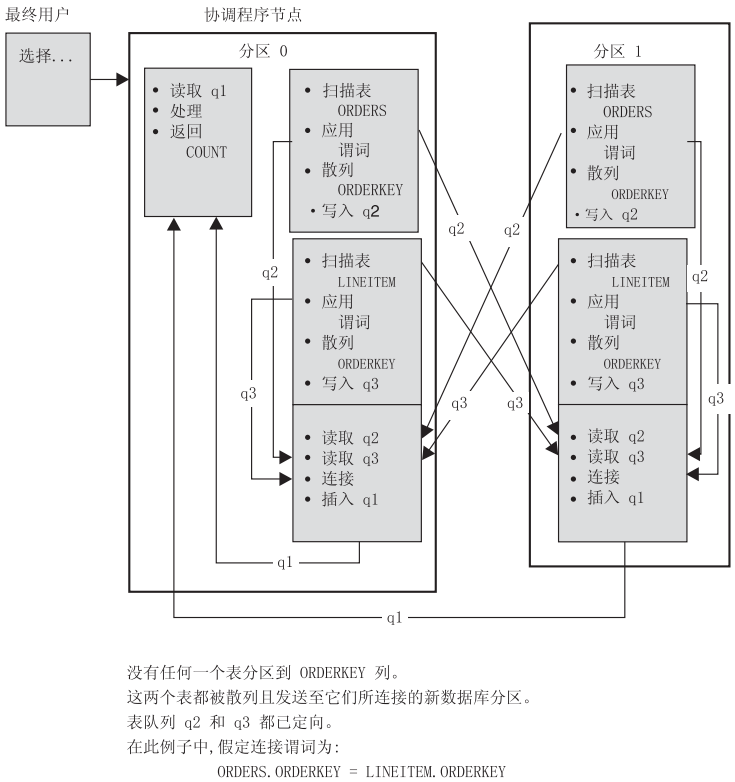
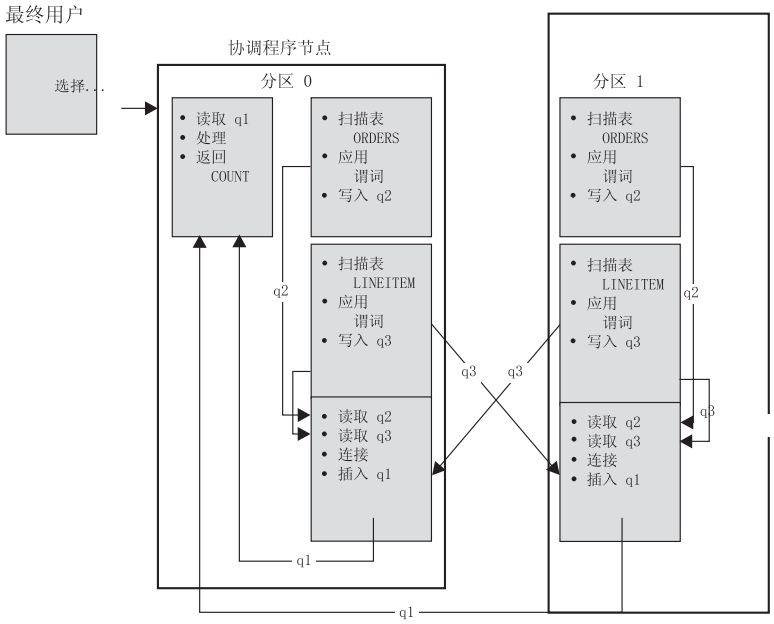


图 26. 定向内部表和外部表连接示例

广播内部表连接

在广播内表部连接策略中，将内部表广播至外连接表的所有数据库分区。以下图形提供了一个示例。

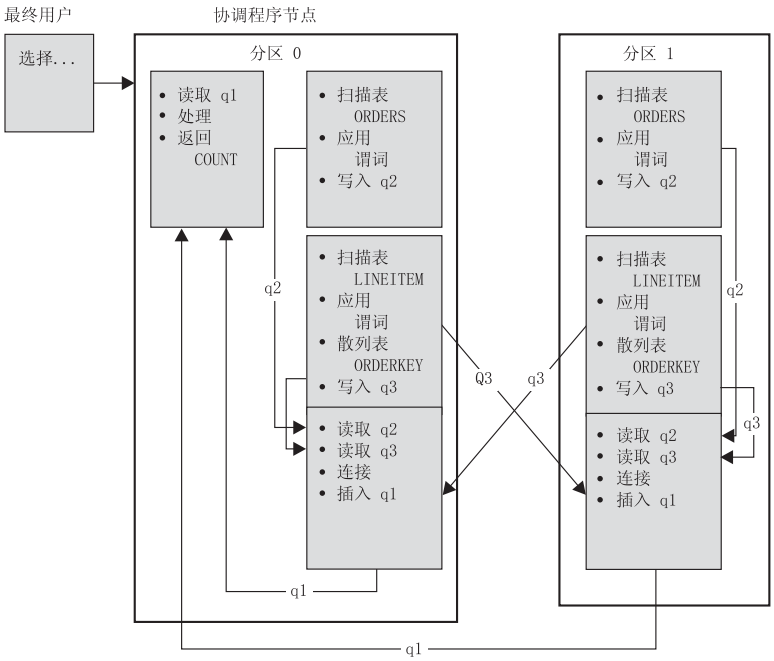


LINEITEM 表将发送至所有具有 ORDERS 表的数据库分区。
表队列 q3 将广播至外部表的所有数据库分区。

图 27. 广播内部表连接示例

定向内部表连接

使用定向内部表连接策略，根据外部表的分割属性，将内部表的每一行发送至外连接表的一个数据库分区。该连接在此数据库分区上进行。以下图形提供了一个示例。



ORDERS 表在 ORDERKEY 列上进行分区。
LINEITEM 表在另一列上进行分区。
散列 LINEITEM 表并将其发送至正确的 ORDERS 表数据库分区。
在此例子中,假定连接谓词为:
ORDERS.ORDERKEY = LINEITEM.ORDERKE。

图 28. 定向内部表连接示例

排序和分组的效果

当优化器选择一个访问方案时，它会考虑对数据排序给性能带来的影响。当没有索引满足请求的对已访存的行的排序时，执行排序。当优化器确定排序比索引扫描开销较少时，也会进行排序。优化器采用下列其中一种方式来对数据进行排序:

- 当执行查询时，通过管道传送排序结果。
- 在数据库管理器内对排序进行内部处理。

管道传送排序与非管道传送排序

如果可按单一顺序读取数据的最终排序列表，那么可以用*管道传送*这些结果。管道传送是一种比非管道传送更快的传送排序结果的方式。优化器会尽可能地选择用管道传送排序的结果。

无论是否使用管道传送排序，排序时间取决于多种因素，包括要排序的行数、键大小和行宽。如果要排序的行占用的空间超过排序堆中可用的空间，那么执行几遍排序，每一遍都要对整个行集的某个子集排序。每遍排序的结果存储在缓冲池中的一个临时表内。如果缓冲池里没有足够的空间，可以将此临时表中的页写到磁盘。当所有排序都完成，必须将这些已排序的子集合并为单个已排序的行集。如果用管道传送排序，当合并那些行时，直接将它们递交给“关系数据服务”。

将下推运算符分组并排序

在有些情况下，优化器可以选择将排序或聚集操作从“关系数据服务”组件下推到“数据管理服务”。下推这些操作将改善性能，因为这样“数据管理服务”组件可直接将数据传送至一个排序或聚集例程。如果不使用此下推，“数据管理服务”首先把此数据传送至“关系数据服务”，然后“关系数据服务”再与排序或聚集例程交互。例如，以下查询可从此优化受益:

```
SELECT WORKDEPT, AVG(SALARY) AS AVG_DEPT_SALARY
       FROM EMPLOYEE        GROUP BY WORKDEPT
```

排序中的分组操作

当排序产生 GROUP BY 操作所需的顺序时，优化器在排序时可执行 GROUP BY 聚集的部分或全部。如果每组中的行数较大，这种做法就有利。如果在排序期间执行一些分组以减少或消去排序溢出到磁盘的情况，就会更加有利。

排序中的聚集需要以下三种聚集阶段以确保返回正确结果。

1. 聚集的第一个阶段“部分聚集”计算聚集值，直到填满排序堆为止。在部分聚集中，接受未聚集数据并产生部分聚集。如果已填满排序堆，其余数据会溢出到磁盘，包括在当前排序堆中已计算的所有部分聚集。在复位排序堆之后，开始新的聚集。
2. 聚集的第二个阶段“中间聚集”提取所有溢出的排序运行结果，并根据分组键进一步聚集。由于分组键列是分布键列的一个子集，故不能完成聚集。中间聚集使用现有的部分聚集来产生新的部分聚集。此阶段并不总是会发生。它用于分区内和分区间并行性。在分区内并行性情况下，当全局分组键可用时完成分组。在分区间并行性的情况下，如果分组键是用来在数据库分区间分组的分布键的子集，从而要求再分发来完成聚集时，会发生这种情况。当每个代理程序在缩减为单个代理程序以完成聚集之前完成对其溢出的排序运行结果的合并时，在分区内并行性中将存在相似的情况。
3. 聚集的最后阶段“最终聚集”使用所有部分聚集并产生最终聚集。此步骤总是通过 GROUP BY 运算符执行。排序不能执行完整的聚集，因为它们不能保证排序不会被分割。完整的聚集接收未聚集的数据，然后产生最终聚集。如果分发不能禁止它的使用，这种聚集方法通常用于将已处于正确顺序的数据进行分组。

优化策略

分区内并行性的优化策略

如果在编译 SQL 语句时指定了并行度，优化器能选择一个访问方案来在单个数据库执行查询。

在执行时，创建称为“子代理程序”的多个数据库代理程序来执行该查询。子代理程序的数目小于或等于编译该 SQL 语句时指定的并行度。

要并行化访问方案，优化器将它划分为两个部分，每个子代理程序运行一部分，协调代理程序运行另一部分。子代理程序通过表队列将数据传送至协调代理程序或其他子代理程序。在分区数据库环境中，子代理程序能通过表队列从其他数据库分区中的子代理程序发送或接收数据。

分区内并行扫描策略

关系扫描和索引扫描可在同一个表或索引上并行执行。要进行并行关系扫描，需将表划分为由页或行组成的范围。将一个范围内的页或行分配给一个子代理程序。一个子代理程序扫描分配给它的范围，当它处理完当前范围时，会给它分配另一个范围。

要进行并行索引扫描，需根据索引键值和一个键值的索引条目数，将索引划分为许多记录范围。并行索引扫描的执行方式类似于并行表扫描，将给子代理程序分配一个范围内的记录。当子代理程序处理完当前范围时，会给它分配一个新的范围。

优化器确定扫描单位（页或行）和扫描详细程度。

并行扫描将工作均匀地分布在各子代理程序中。并行扫描的目标是平衡所有子代理程序的负载，并使它们保持相同的繁忙程度。如果繁忙子代理程序数等于可用处理器数，且磁盘没有过度处理 I/O 请求，那么表明机器资源正在得到有效的利用。

当执行查询时，其他访问方案策略可能导致数据不平衡。优化器选择并行策略，以便维护子代理程序间的数据平衡。

分区内并行排序策略

优化器能选择下列其中一个并行排序策略:

- **循环排序**

  这也称为*再分布排序*。这种方法有效地使用共享内存，将数据尽可能均匀地再分布到所有子代理程序中。它使用轮询算法来提供均匀分布。它首先为每个子代理程序创建个别排序。在插入阶段期间，子代理程序以循环方式插入每个个别的排序中以使数据分布更加均匀。

- **分区排序**

  这类似于循环排序，在循环排序中，要为每个子代理程序创建一个排序。子代理程序将一个散列函数应用于排序列，以确定应将行插入哪个排序中。例如，如果一个合并连接的内部表和外部表是一个分区排序，那么子代理程序可使用合并连接来连接对应的表部分并且并行执行。

- **复制排序**

  如果每个子代理程序需要全部排序输出，那么使用这种排序。创建一个排序，当将行插入排序时使子代理程序同步。当完成排序时，每个子代理程序都读取整个排序。如果行数较小，可使用此排序来重新平衡数据流。

- **共享排序**

  此排序与复制排序相同，只是子代理程序要对已排序的结果打开一个并行扫描以便用一种与循环排序相似的方式在子代理程序中分布数据。

分区内并行临时表

子代理程序可以协同工作，将行插入同一个表中来产生临时表。这称为共享临时表。子代理程序可以根据数据流是要复制还是要分割，在共享临时表上打开专用扫描或并行扫描。

分区内并行聚集策略

子代理程序可以并行执行聚集操作。聚集操作要求根据分组列将数据排序。如果可以保证一个子代理程序接收一组分组列值的所有行，那么该程序可以执行完整的聚集。如果由于先前的分区排序，已在分组列上分割了数据流，那么这种情况可发生。

否则，子代理程序可以执行部分聚集，并使用另一种策略来完成该聚集。其中部分策略如下:

- 通过一个合并表队列将部分聚集的数据发送至协调代理程序。协调程序完成聚集。
- 将部分聚集的数据插入一个分区排序中。在分组列上分割该排序，并保证一组分组列的所有行都包含在一个排序分区中。
- 如果需要复制数据流以便平衡处理，可将部分聚集的数据插入一个复制排序中。每个子代理程序使用复制排序完成聚集，并接收完全相同的聚集结果副本。

分区内并行连接策略

子代理程序可以并行执行连接操作。并行连接策略由数据流的特征确定。

通过在连接的内部和外部表上分区和/或复制数据流，可将连接并行化。例如，如果因为并行扫描已将一个嵌套循环连接的外部流分区，而且内部流由每个子代理程序单独重新求值，那么可将该连接并行化。如果一个合并连接的内部和外部流因为分区排序都已按值分区，那么可将该连接并行化。

MDC 表的优化策略

如果创建多维集群（MDC）表，因为优化器可以应用附加的优化策略，可以提高许多查询的性能。这些策略主要基于改进的块索引效率，但在多维上进行集群的优点也允许更快速的数据检索。

注: MDC 表优化策略也可以实现分区内并行性和分区间并行性的性能优点。

考虑 MDC 表的下列特定优点:

- 维块索引查找可以标识表的所需部分，并仅快速扫描所需的块。
- 因为块索引小于 RID 索引，所以查找更快。
- 可以在块级别执行索引 AND 和 OR 运算，并可以将这些运算与 RID 组合在一起。
- 保证在扩展数据块内集群数据，这使得检索更快。
- 当可以使用转出时，可以更快地删除行。

考虑在 **region** 和 **month** 列上定义了维且命名为 SALES 的 MDC 表的以下简单示例:

```
SELECT * FROM SALES
    WHERE MONTH='March' AND REGION='SE'
```

对于此查询，优化器可以执行维块索引查找来寻找出现在月份为三月且地区为 SE 的块。然后该查询可以仅快速扫描表的结果块来访存结果集。

转出删除

当满足允许使用转出以进行删除的条件时，就会使用删除 MDC 表中的行的更有效方法。这些条件是:

- 搜索而不是定位 DELETE 语句（即，不使用“WHERE CURRENT OF”子句）。
- 没有 WHERE 子句（将删除所有行）或 WHERE 子句中仅有的条件是针对维的。

- 未使用 DATA CAPTURE CHANGES 子句来定义表。
- 表在引用完整性关系中不是父表。
- 表没有定义“删除时”触发器。
- 未在任何立即刷新的 MQT 中使用表。
- 如果级联删除操作的外键是其表的维列的子集，那么它可能适合于转出。
- 在触发由 CREATE TRIGGER 语句上的 OLD TABLE AS 子句指定的 SQL 操作之前，DELETE 语句不能出现在对用于标识受影响行集的临时表执行的 SELECT 语句中。

对于转出删除，不会记录被删除的记录。而是会通过重新格式化页的某些部分来使包含这些记录的页看上去是空的。将会记录对重新格式化的部分所做的更改，但不会记录这些记录本身。

*立即清除转出*这一缺省行为就是在删除时就清除 RID 索引。还可以通过将注册表变量 **DB2_MDC_ROLLOUT** 设置为 IMMEDIATE，或者通过对 SET CURRENT MDC ROLLOUT MODE 语句指定 IMMEDIATE 来指定此方式。与标准删除相比，索引更新的日志记录没有变化，因此，性能提高取决于有多少个 RID 索引。RID 索引越少，性能就越好，衡量方法是总时间与日志空间的百分比。

可以通过以下公式来估计日志中剩余的空间量，其中 N 是已删除的记录数，S 是已删除的记录的总大小（包括诸如空指示符和 varchar 长度的开销），P 是包含已删除的记录的块中的页数:

```
S + 38*N - 50*P
```

此图是实际日志数据的简化形式。由于节省了为进行回滚而保留的空间，因此，活动日志空间要求将减小一半。

另外，在落实事务之后，可以使用*延迟清除转出*来更新 RID 索引。还可以通过将注册表变量 **DB2_MDC_ROLL_OUT** 设置为 DEFER，或者通过对 SET CURRENT MDC ROLLOUT MODE 语句指定 DEFERRED 来指定此方式。在延迟转出方式下，将在落实删除之后在后台异步清除 RID 索引。对于非常大型的删除任务或者表中存在大量 RID 索引时，使用这种转出方法可以非常快速地进行删除。整个清除操作的速度也提高了，这是因为在执行延迟索引清除时将并行清除索引，而在执行立即索引清除时将逐行清除索引中的每一行。另外，DELETE 语句的事务日志空间需求显著降低，这是因为索引按索引页而不是按索引键来更新异步索引清除日志。

注: 延迟清除转出需要更多内存资源，内存资源是从数据库堆中获取的。如果 DB2 无法分配它需要的内存结构，那么延迟清除转出将失败，并将一条消息写入管理员日志中。

何时使用延迟清除转出

如果对于您来说删除性能是最重要的因素，并且已经对表定义了 RID 索引，那么就应使用延迟清除转出。注意，在进行索引清除之前，对已转出的块进行基于索引的扫描会稍微降低性能，这取决于已转出的数据量。下面是在决定执行立即索引清除和延迟索引清除时应考虑的其他问题:

- 删除工作量大小: 对于非常大型的删除任务选择延迟清除转出。在对许多小型 MDC 表频繁发出维删除语句的情况下，异步清除索引对象所产生的开销在价值上要超过删除期间节省的时间所带来的好处。

- 索引的数量和类型：如果表中包含大量 RID 索引，且需要对这些索引执行行级别处理，那么应使用延迟清除转出。
- 块可用性：如果您希望由 DELETE 语句释放的块空间在落实 DELETE 语句之后立即可用，那么应使用立即清除转出。
- 日志空间：如果日志空间有限，那么应对大型删除任务使用延迟清除转出。
- 内存约束：延迟清除转出将在所有具有延迟清除暂挂的表上消耗更多数据库堆。

要在执行删除时禁止转出行为，可以将注册表变量 **DB2_MDC_ROLLOUT** 设置为 OFF，或者对 SET CURRENT MDC ROLLOUT MODE 语句指定 NONE。

分区表的优化策略

数据分区消除指的是数据库服务器根据查询谓词确定只需要访问表的一部分数据分区就可以实现查询的能力。当对分区表运行决策支持查询时，数据分区消除可以提供特定好处。

分区表使用了数据组织方案，即，表数据根据该表中一个或多个表分区键列中的值分布到多个存储对象（称为数据分区或范围）中。根据 CREATE TABLE 语句的 PARTITION BY 子句中指定的内容，给定表的数据被划分到多个存储对象中。这些存储对象可以在不同的表空间中，也可以在相同表空间中。

以下示例演示了数据分区消除所产生的性能方面的好处。如果发出以下语句:

```
CREATE TABLE custlist(subsdate DATE, Province CHAR(2), AccountID INT)
PARTITION BY RANGE(subsdate)
(STARTING FROM '1/1/1990' IN ts1,
STARTING FROM '1/1/1991' IN ts1,
STARTING FROM '1/1/1992' IN ts1,
STARTING FROM '1/1/1993' IN ts2,
STARTING FROM '1/1/1994' IN ts2,
STARTING FROM '1/1/1995' IN ts2,
STARTING FROM '1/1/1996' IN ts3,
STARTING FROM '1/1/1997' IN ts3,
STARTING FROM '1/1/1998' IN ts3,
STARTING FROM '1/1/1999' IN ts4,
STARTING FROM '1/1/2000' IN ts4,
STARTING FROM '1/1/2001' ENDING '12/31/2001' IN ts4);
```

假定您对 2000 年的客户信息感兴趣。如果发出以下查询:

```
SELECT * FROM custlist WHERE subsdate BETWEEN '1/1/2000' AND '12/31/2000';
```

正如第 275 页的图 29所显示的那样，数据库服务器确定只需要访问表空间 4（ts4）中的一个数据分区就可以解决此查询。

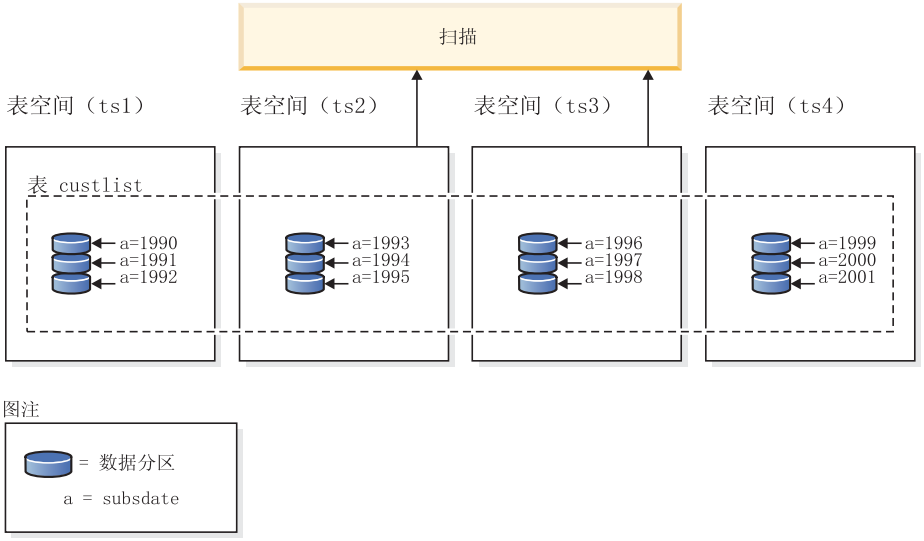


图 29. 分区表上数据分区消除所产生的性能方面的好处

图 第 276 页的图 30 中显示的另一个数据分区消除示例是索引扫描，它涉及两个索引并根据以下方案进行扫描:

```
CREATE TABLE multi (sale_date date, region char(2))
PARTITION BY (sale_date)
(STARTING '01/01/2005' ENDING '12/31/2005' EVERY 1 MONTH);
CREATE INDEX sx ON multi(sale_date);
CREATE INDEX rx ON multi(region);
```

如果发出以下查询:

```
SELECT * FROM multi WHERE
sale_date BETWEEN '6/1/2005' AND '7/31/2005' AND REGION = 'NW';
```

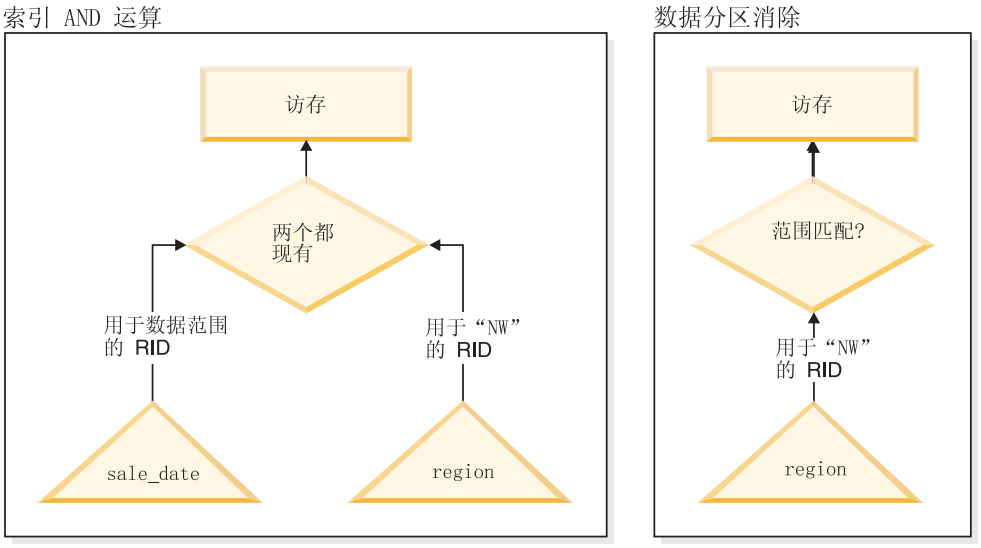


图 30. 表分区和索引“与”（AND）的优化器决策路径

在不使用表分区时，一种可能的方案是索引“与”（AND）。索引“与”（AND）执行下列任务:

- 读取每个索引中的所有相关索引条目
- 保存两组行标识（RID）
- 匹配 RID 以确定哪些 RID 同时出现在这两个索引中
- 使用 RID 来访存行

如图 30 中所示，在使用表分区的情况下，读取索引以查找 region 和 sale_date 的匹配项，从而允许快速检索匹配行。

DB2 Explain

还可以使用 DB2 说明来确定 DB2 优化器选择的分区消除。**DP Elim Predicates** 信息显示扫描了哪些数据分区来解决以下查询:

```
SELECT * FROM custlist WHERE subsdate
BETWEEN '12/31/1999' AND '1/1/2001'

Arguments:
---------
DPESTFLG: (Number of data partitions accessed are Estimated)
     FALSE
DPLSTPRT: (List of data partitions accessed)
     9-11
DPNUMPRT: (Number of data partitions accessed)
   3

DP Elim Predicates:
------------------
Range 1)
   Stop  Predicate: (Q1.A <= '01/01/2001')
 Start Predicate: ('12/31/1999' <= Q1.A)



Objects Used in Access Plan:
 ---------------------------
```

```
  Schema: MRSRINI
  Name:    CUSTLIST
  Type:    Data Partitioned Table
  Time of creation:      2005-11-30-14.21.33.857039
  Last statistics update:    2005-11-30-14.21.34.339392
    Number of columns:      3
    Number of rows:      100000
  Width of rows:      19
  Number of buffer pool pages:    1200
    Number of data partitions:    12
    Distinct row values:      No
   Tablespace name:      <VARIOUS>
```

多列支持

在使用多个列作为表分区键的情况下，数据分区消除将起作用。

例如，如果发出以下语句:

```
CREATE TABLE sales (year INT, month INT)
 PARTITION BY RANGE(year, month)
 (STARTING FROM (2001, 1) ENDING AT(2001,3) IN ts1,
 ENDING AT(2001,6) IN ts2,
 ENDING AT(2001,9) IN ts3,
 ENDING AT(2001,12) IN ts4,
 ENDING AT(2002,3) IN ts5,
 ENDING AT(2002,6) IN ts6,
 ENDING AT(2002,9) IN ts7,
 ENDING AT(2002,12) IN ts8)
```

接着，发出以下查询:

```
 SELECT * FROM sales WHERE year = 2001 AND month < 8
```

查询优化器推断只需要访问 ts1、ts2 和 ts3 中的数据分区就可以解决此查询。

注: 在多个列组成表分区键的情况下，只有当拥有组合键的前导列上的谓词时才能实现数据分区消除，因为用于表分区键的非前导列不是独立的。

多范围支持

可以使用多个范围在数据分区上实现数据分区消除（即，一起执行“或”（OR）运算）。通过使用上一个示例中创建的表，执行下列查询:

```
 SELECT * FROM sales
 WHERE (year = 2001 AND month <= 3) OR (year = 2002 and month >= 10)
```

数据库服务器只访问 2001 年的第一季度和 2002 年的最后一个季度的数据。

生成列

可以将生成列用作表分区键。

例如，可以发出以下语句:

```
CREATE TABLE sales(a INT, b INT GENERATED ALWAYS AS (a / 5))
 IN ts1,ts2,ts3,ts4,ts5,ts6,ts7,ts8,ts9,ts10
 PARTITION BY RANGE(b)
 (STARTING FROM (0) ENDING AT(1000) EVERY (50))
```

在此示例中，将生成列上的谓词用于数据分区消除。此外，当用来生成列的表达式是单调的时，数据库服务器会将源列上的谓词转换为生成列上的谓词，从而在生成列上启用数据分区消除。

例如，如果具有以下查询:

```
SELECT * FROM sales WHERE a > 35
```

数据库服务器根据（a > 35）在 b（b > 7）上生成额外谓词，从而允许数据分区消除。

连接谓词

如果将连接谓词下推到表访问级别，那么也可以在数据分区消除中使用连接谓词。连接谓词仅在嵌套循环连接（NLJN）的内部才下推到表访问级别。

请考虑下列表:

```
CREATE TABLE T1(A INT, B INT)
 PARTITION BY RANGE(A, B)
 (STARTING FROM (1, 1)
 ENDING (1,10) IN ts1, ENDING (1,20) IN ts2,
 ENDING (2,10) IN ts3, ENDING (2,20) IN ts4,
  ENDING (3,10) IN ts5, ENDING (3,20) IN ts6,
 ENDING (4,10) IN ts7, ENDING (4,20) IN ts8)

CREATE TABLE T2 (A INT, B INT)
```

使用的谓词:

```
P1: T1.A = T2.A
P2: T1.B > 15
```

在此示例中，由于不知道连接的外值，因此不能确定将在编译时访问的额外数据分区。在这种情况下，以及在使用主变量或参数标记的情况下，当绑定必需的值时，就会发生数据分区消除。

在运行时，当 T1 是 NLJN 的内部表时，会根据 T2.A 的每个外值的谓词自动进行数据分区消除。在运行时，对外值 T2.A = 3 应用谓词 T1.A = 3 和 T1.B > 15，这样就限定了访问的表空间 ts6 和 ts7 中的数据分区。

考虑表 T1 和 T2 中的列 A 具有下列值:

外部表 T2: 列 A	内部表 T1: 列 A	内部表 T1: 列 B	内部表 T1: 数据分区位置
2	3	20	ts6
3	2	10	ts3
3	2	18	ts4
	3	15	ts6
	1	40	ts3

要执行嵌套循环连接（假定对内部表进行表扫描），数据库管理器执行下列步骤:

1. 读取 T2 中的第一行。A 的值是 2。
2. 在连接谓词 T1.A = T2.A 中将 T2.A 值（它是 2）绑定到列 T2.A。该谓词就变成 T1.A = 2。
3. 使用谓词 T1.A = 2 和 T1.B > 15 应用数据分区消除。这将限定表空间 ts4 和 ts5 中的数据分区。

4. 在应用 T1.A = 2 和 T1.B > 15 之后，扫描表 T1 的表空间 ts4 和 ts5 中的数据分区，直到找到一行为止。找到的第一个合格行是 T1 的行 3。
5. 连接匹配的行。
6. 扫描表 T1 的表空间 ts4 和 ts5 中的数据分区，直到找到下一个匹配项（T1.A = 2 和 T1.B > 15）为止。再也找不到其他行。
7. 对 T2 的下一行（将 A 的值替换为 3）重复步骤 1 至 6，直到用完 T2 中的所有行为止。

具体化查询表

对于复杂查询，尤其是可能需要执行下列某些操作的查询，具体化查询表（MQT）能够极大地改进响应时间:

- 基于一个或多个维聚集数据
- 连接和聚集涉及一组表的数据
- 通常访问的数据子集（即“热”水平或垂直数据库分区）中的数据
- 在分区数据库环境中，表或表的一部分中重新分区的数据

MQT 的知识已经集成到 SQL 和 XQuery 编译器中。在该编译器中，查询重写阶段和优化器将查询与 MQT 匹配，并确定是否要用 MQT 取代访问基本表的查询。如果使用 MQT，那么说明工具可以提供关于选择了哪个 MQT 的信息。

因为 MQT 的行为在许多方面类似于常规表，所以有关使用表空间定义来优化数据访问、创建索引以及发出 RUNSTATS 的准则也适用于 MQT。

为了帮助您理解 MQT 的功能，以下示例说明了多维分析查询以及它如何利用 MQT。

在此示例中，假定这样一个数据库仓库: 它包含一组客户和一组信用卡帐户。仓库记录使用信用卡进行的交易集。每项交易都包含一批一起购买的物品。此模式分类为多星型模式，因为有两个大表，一个包含交易商品，另一个标识购买交易。

三个层次结构维描述一项交易: 产品、位置和时间。产品层次结构存储在两个分别表示产品组和产品行的标准表中。位置层次结构包含城市、州和国家或地区信息，包含在单个非标准表中。时间层次结构包含日、月和年信息，编码在单一日期字段中。日期维是使用内置函数从交易的日期字段中抽取的。此模式中的其他表表示客户的帐户信息和客户信息。

使用以下层次结构的每一级别的销售总额和销售件数来创建 MQT:

- 产品
- 位置
- 时间，由年、月和日组成。

从此存储的聚集数据可以满足许多查询。以下示例显示如何创建一个 MQT，该 MQT 按照 product group 和 product line 维、按照 city、state 和 country 维以及按照 Time 维来计算销售金额和销售件数。在它的 GROUP BY 子句中，还包括几个其他的列。

```
CREATE TABLE dba.PG_SALESSUM
      AS (
      SELECT l.id AS prodline, pg.id AS pgroup,
             loc.country, loc.state, loc.city,
             l.name AS linename, pg.name AS pgname,
```

```
            YEAR(pdate) AS year, MONTH(pdate) AS month,
            t.status,
            SUM(ti.amount) AS amount,
            COUNT(*) AS count
      FROM   cube.transitem AS ti, cube.trans AS t,
            cube.loc AS loc, cube.pgroup AS pg,
            cube.prodline AS l
      WHERE  ti.transid = t.id
         AND ti.pgid = pg.id
         AND pg.lineid = l.id
         AND t.locid = loc.id
         AND YEAR(pdate) > 1990
      GROUP BY l.id, pg.id, loc.country, loc.state, loc.city,
               year(pdate), month(pdate), t.status, l.name, pg.name
     )
DATA INITIALLY DEFERRED REFRESH DEFERRED;

REFRESH TABLE dba.SALESCUBE;
```

可利用这类预先计算的总额的查询包括以下几种:

- 按月和产品组的销售额
- 1990 年以来的总销售额
- 1995 年或 1996 年的销售额
- 产品组或产品线的销售总额
- 特定产品组或产品线在 1995、1996 年的销售总额
- 特定国家或地区的销售总额。

当 MQT 未包括以上任何一个查询的准确答案时，使用 MQT 计算答案的成本可能明显地小于使用大型基本表的成本，因为此答案的一部分已计算出来。MQT 可以减少基本数据的成本高昂的连接、排序和聚集。

下列样本查询将获得明显的性能提高，因为它们可以使用示例 MQT 中已经计算出的结果。

第一个示例返回 1995 和 1996 年的销售总额:

```
SET CURRENT REFRESH AGE=ANY

  SELECT YEAR(pdate) AS year, SUM(ti.amount) AS amount
  FROM   cube.transitem AS ti, cube.trans AS t,
               cube.loc AS loc, cube.pgroup AS pg,
               cube.prodline AS l
  WHERE  ti.transid = t.id
        AND ti.pgid = pg.id
        AND pg.lineid = l.id
        AND t.locid = loc.id
        AND YEAR(pdate) IN (1995, 1996)
  GROUP BY year(pdate);
```

第二个示例返回 1995 和 1996 年产品组的销售总额:

```
SET CURRENT REFRESH AGE=ANY

  SELECT pg.id AS "PRODUCT GROUP",
               SUM(ti.amount) AS amount
  FROM   cube.transitem AS ti, cube.trans AS t,
               cube.loc AS loc, cube.pgroup AS pg,
               cube.prodline AS l
  WHERE  ti.transid = t.id
        AND ti.pgid = pg.id
```

```
        AND pg.lineid = l.id
        AND t.locid = loc.id
        AND YEAR(pdate) IN (1995, 1996)
GROUP BY pg.id;
```

基本表越大，在响应时间中的改进可以越大，因为 MQT 的增长比基本表的增长慢得多。MQT 可以通过在构建和刷新 MQT 时执行一次计算，并对许多查询重复使用 MQT 的内容可以有效地消去查询之间的重叠工作。

说明工具

SQL 或 XQuery 编译器可捕获有关访问方案以及静态或动态 SQL 和 XQuery 语句环境的信息。所捕获的信息帮助您了解单个 SQL 或 XQuery 语句是如何执行的，以便您可以调整语句和数据库管理器配置来提高性能。

因为下列原因，收集和使用说明数据:

- 了解数据库管理器如何访问表和索引来满足查询
- 评估性能调整操作

  更改数据库管理器、SQL 或 XQuery 语句以及数据库的某个方面时，应检查说明数据来了解操作如何更改了性能。

捕获的信息包括:

- 处理查询的操作顺序
- 成本信息
- 谓词和每个谓词的选择性估计
- 对在捕获说明时 SQL 或 XQuery 语句中引用的所有对象的统计信息
- 使用主变量、参数标记或专用寄存器的值来重新优化 SQL 或 XQuery 语句。

在可以捕获说明信息之前，应创建优化器将说明信息存储在其中的关系表，并设置确定捕获哪种类型的说明信息的专用寄存器。

要显示说明信息，可以使用命令行工具或 Visual Explain。使用的工具确定您如何设置专用寄存器，这些注册表变量确定收集哪些说明数据。例如，如果您期望仅使用 Visual Explain，那么需要仅捕获快照信息。如果您想要使用其中一个命令行实用程序或使用定制 SQL 或 XQuery 语句对说明表执行详细分析，那么应捕获所有说明信息。

使用说明信息的准则

因为下列两个主要目的，使用说明信息:

- 了解应用程序性能更改的原因
- 评估性能调整工作

性能更改分析

为了帮助您了解查询性能发生变化的原因，您需要更改前后的说明信息，可以通过执行下列步骤来获得这些信息:

- 捕获作出任何更改前查询的说明信息，并保存生成的说明表，或者，可以保存 `db2exfmt` 说明工具的输出。

- 如果您不想或不能访问 Visual Explain 来查看此信息，那么保存或打印当前的目录统计信息。还可以使用 db2look 生产工具来帮助执行此任务。
- 保存或打印数据定义语言（DDL）语句，包括用于 CREATE TABLE、CREATE VIEW、CREATE INDEX 和 CREATE TABLESPACE 的那些语句。

按此方式收集的信息为将来分析提供参考点。对于动态 SQL 或 XQuery 语句，您可在首次运行应用程序时收集此信息。对于静态 SQL 和 XQuery 语句，您可在绑定时收集此信息。要分析性能更改，将已收集的信息与关于查询收集的信息和启动分析时的环境比较。

举一个简单的例子，您的分析可以显示不再将一个索引用作访问路径的一部分。使用 Visual Explain 中的目录统计信息，您可能注意到索引层数（NLEVELS 列）现在大大高于当该查询首次与数据库绑定时的索引层数。于是，您可以选择执行下列其中一项操作:

- 重组该索引
- 收集您的表和索引的新统计信息
- 当重新绑定您的查询时收集说明信息

执行其中一项操作之后，再次检查访问方案。如果再次使用索引，查询的性能可能不再是一个问题。如果仍未使用索引或如果性能仍是一个问题，那么执行第二操作并检查结果。重复这些步骤，直到解决问题为止。

性能调整工作的评估

可以采取几个操作来帮助提高查询性能，如调整配置参数、添加容器以及收集新的目录统计信息等。

当在其中任何一个区域中进行更改后，您可使用说明工具来确定该更改对所选访问方案的影响（如果有的话）。例如，如果您根据索引准则来添加索引或具体化查询表（MQT），那么说明数据可以帮助您确定是否按预期的那样实际使用该索引或具体化查询表。

尽管说明输出提供信息以允许您确定选择的访问方案及其相关成本时，但准确测量一个查询的性能提高的唯一方法是使用基准程序测试技术。

捕获说明信息的准则

如果在编译 SQL 或 XQuery 语句时请求说明数据，那么捕获该数据。考虑您期望在请求说明数据时如何使用捕获的信息。

注:

1. 如果在运行时编译增量绑定 SQL 或 XQuery 语句，那么在运行时而不是绑定时将数据放入说明表中。对于这些语句，插入的说明表限定符和授权标识是程序包所有者（而不是运行该程序包的用户）的限定符和授权标识。
2. 仅当编译 SQL 或 XQuery 语句时才捕获说明信息。在初始编译之后，当环境更改要求动态查询语句时，或当说明工具活动时，将再次编译这些动态查询语句。如果对同一查询语句发出相同的 PREPARE 语句，那么每当准备或执行此语句时，都将编译该语句并捕获说明数据。

3. 如果程序包是使用绑定选项 REOPT ONCE/ALWAYS 绑定的，那么将编译包含主变量、参数标记、全局变量或专用寄存器的 SQL 或 XQuery 语句，并且在编译时如果知道这些变量的实际值，将使用他们创建访问路径，如果不知道，那么使用缺省估计值创建访问路径。
4. 如果使用了 FOR REOPT ONCE 子句，那么尝试使指定的 SQL 或 XQuery 语句与程序包高速缓存中的同一语句相匹配。将使用此已重新优化的高速缓存查询语句的值来重新优化指定的查询语句。如果用户具有必需的访问特权，那么说明表将包含新生成的经过重新优化的访问方案和用于此重新优化的值。
5. 在多分区系统中，应在最初编译和使用 REOPT ONCE 重新优化该语句的同一数据库分区上说明该语句，否则将返回错误。

捕获说明表中的信息

- **静态或增量绑定 SQL 和 XQuery 语句**:

  在 BIND 或 PREP 命令上指定 EXPLAIN ALL 或 EXPLAIN YES 选项，或在源程序中包含静态 EXPLAIN 语句。

- **动态 SQL 和 XQuery 语句**:

  在下列任何一种情况下捕获说明表信息:

  – 将 CURRENT EXPLAIN MODE 专用寄存器设置为:
    - YES: SQL 和 XQuery 编译器捕获说明数据并执行该查询语句。
    - EXPLAIN: SQL 和 XQuery 编译器捕获说明数据，但不执行该查询语句。
    - RECOMMEND INDEXES: SQL 和 XQuery 编译器捕获说明数据并将建议索引放进 ADVISE_INDEX 表中，但不执行该查询语句。
    - EVALUATE INDEXES: SQL 和 XQuery 编译器使用用户放进 ADVISE_INDEX 表中的索引进行求值。在 EVALUATE INDEXES 方式中，说明所有动态语句，就像这些虚拟索引是可用的。如果虚拟索引可以提高语句性能，那么查询编译器将选择使用虚拟索引。否则，忽略这些索引。要了解已提出的索引是否有用，复查 EXPLAIN 结果。
    - REOPT: 当主变量、专用寄存器、全局变量或参数标记的实际值可用时，查询编译器会在执行时重新优化语句期间捕获静态或动态 SQL 或 XQuery 语句的说明数据。

- 已在 BIND 或 PREP 命令上指定了 EXPLAIN ALL 选项。查询编译器在运行时捕获动态 SQL 和 XQuery 的说明数据，即使 CURRENT EXPLAIN MODE 专用寄存器设置为 NO 时也是如此。SQL 或 XQuery 语句还执行查询并返回查询结果。

捕获说明快照信息

当请求说明快照时，将说明信息按 Visual Explain 需要的格式存储在 EXPLAIN_STATEMENT 表的 SNAPSHOT 列中。其他应用程序不能使用此格式。可从 Visual Explain 本身获取有关说明快照信息内容的其他信息。此信息包括关于数据对象和数据运算符的信息。

当编译 SQL 或 XQuery 语句并已请求说明数据时，会捕获说明快照数据，如下所示:

- **静态或增量绑定 SQL 和 XQuery 语句**:

当在 BIND 或 PREP 命令上指定了 EXPLSNAP ALL 或 EXPLSNAP YES 子句时，或当源程序包含使用 FOR SNAPSHOT 或 WITH SNAPSHOT 子句的静态 EXPLAIN 语句时，将捕获说明快照。

- **动态 SQL 和 XQuery 语句**:

  在下列任何一种情况下捕获说明快照:

  – 发出带有 FOR SNAPSHOT 或 WITH SNAPSHOT 子句的 EXPLAIN 语句。对于 FOR SNAPSHOT 子句，仅捕获说明快照信息。对于 WITH SNAPSHOT 子句，除快照信息外，也捕获所有说明信息。

  – 将 CURRENT EXPLAIN SNAPSHOT 专用寄存器设置为:

    - YES: 查询编译器捕获快照说明数据并执行该 SQL 或 XQuery 语句。

    - EXPLAIN: 查询编译器捕获快照说明数据，但不执行该 SQL 或 XQuery 语句。

  – 在 BIND 或 PREP 命令上指定 EXPLAIN ALL 选项。查询编译器在运行时捕获快照说明数据，即使 CURRENT EXPLAIN SNAPSHOT 专用寄存器的设置为 NO 也是如此。它还执行该 SQL 或 XQuery 语句。

分析说明信息的准则

说明信息的主要用途是分析查询语句的访问路径。有许多方法用来分析说明数据，以帮助您调整查询和环境。考虑下列几种分析:

- **索引使用**

  适当的索引可明显地有益于性能。使用说明输出，您可以确定是否正在使用您创建的索引来帮助进行一组特定的查询。在说明输出中，应在下列几方面查找索引的使用:

  – 连接谓词

  – 本地谓词

  – GROUP BY 子句

  – ORDER BY 子句

  – WHERE XMLEXISTS 子句

  – 选择列表

  也可使用说明工具来评估是否可使用另一个索引来代替现有的索引，或根本不使用索引。当创建新索引后，使用 RUNSTATS 命令来收集该索引的统计信息并重新编译该查询。经过一段时间，您可能通过说明数据会注意到现在正在使用表扫描，而不是索引扫描。这可能是表数据的集群发生改变所导致的。如果先前使用的索引现在具有低的集群比率，那么您可能要根据该索引重组表以集群其数据，使用 RUNSTATS 命令来收集索引和表的统计信息，然后重新编译该查询。要确定重组表是否已改进访问方案，对重新编译查询重新检查说明输出。

- **访问类型**

  分析说明输出，并查找访问数据的类型，这种访问对于您正在运行的应用程序的类型而言通常不是最优的。例如:

  – **联机事务处理（OLTP）查询**

OLTP 应用程序是用范围定界的谓词来使用索引扫描的主要候选者，因为它们倾向于对键列使用等式谓词以只返回少数合格的行。如果您的 OLTP 查询使用表扫描，应分析说明数据以确定不使用索引扫描的原因。

– **仅浏览查询**

“浏览”类型查询的搜索条件可能很含糊，这样会产生大量合格的行。如果用户通常只查看少数输出数据屏幕，您可能指定在返回一些结果前，不需要计算整个答案集。对于这种情况，用户的目标与优化器的基本操作原则不同，优化器试图将整个查询的资源消耗减小到最低程度，而不只是前几个数据屏幕。

例如，如果说明输出表明在访问方案中使用了合并扫描连接和排序运算符，那么在将任何行返回至应用程序之前将在一个临时表中实现整个答案集。在这种情况下，您可以尝试在 SELECT 语句上使用 OPTIMIZE FOR 子句来更改访问方案。如果您指定此选项，优化器可尝试选择一个访问方案，该方案在将前面几行返回至应用程序前，不在临时表中产生整个答案集。

- **连接方法**

如果一个查询连接了两个表，检查所用的连接的类型。涉及多个行的连接，如在决策支持查询中的那些连接，使用散列连接或合并连接通常会运行得更快。只涉及少数行的连接，如 OLTP 查询，通常使用嵌套的循环连接会运行得更快。但是，在任何一种情况下都可能存在运行减慢的情况，如使用本地谓词或索引，可以更改这些典型连接工作的方式。

使用访问方案来自诊断 REFRESH TABLE 和 SET INTEGRITY 语句的性能问题

EXPLAIN for REFRESH TABLE 和 SET INTEGRITY 语句允许您生成可用来自诊断这些语句的性能问题的访问方案。这将帮助您更好地维护具体化查询表（MQT）。

要获取 REFRESH TABLE 或 SET INTEGRITY 语句的访问方案，请使用下列任一种方法:

- 在 EXPLAIN 语句中使用 EXPLAIN PLAN FOR REFRESH TABLE 或 EXPLAIN PLAN FOR SET INTEGRITY 选项
- 在发出 REFRESH TABLE 或 SET INTEGRITY 语句前将 CURRENT EXPLAIN MODE 专用寄存器设置为 EXPLAIN，并在发出该语句后将 CURRENT EXPLAIN MODE 专用寄存器设置为 NO。

限制:

- REFRESH TABLE 和 SET INTEGRITY 语句不适合进行重新优化，因此，REOPT 说明方式（或说明快照）不适用于这两个语句。
- EXPLAIN 语句的 WITH REOPT ONCE 子句也指示将重新优化指定的可说明语句，它不适用于 REFRESH TABLE 和 SET INTEGRITY 语句。

方案

此方案说明如何从 EXPLAIN 和 REFRESH TABLE 语句生成访问方案并使用该方案来自诊断性能问题的原因。

第一步 创建并填充表。例如,

```
CREATE TABLE T
    (i1 INT NOT NULL,
     i2 INT NOT NULL,
     PRIMARY KEY (i1));
INSERT INTO T VALUES (1,1), (2,1), (3,2), (4,2);
CREATE TABLE MQT AS (SELECT i2, COUNT(*) AS CNT FROM T GROUP BY i2)
  DATA INITIALLY DEFERRED REFRESH DEFERRED;
```

第二步 发出 EXPLAIN 和 REFRESH TABLE 语句，如下所示:

```
EXPLAIN PLAN FOR REFRESH TABLE MQT;
```

注: 此步骤可替换为在 SET CURRENT EXPLAIN MODE 专用寄存器中设置 EXPLAIN 方式，如下所示:

```
SET CURRENT EXPLAIN MODE EXPLAIN;
REFRESH TABLE MQT;
SET CURRENT EXPLAIN MODE NO;
```

第三步 使用 db2exfmt 命令来格式化说明表的内容并获取访问方案。此工具位于实例 sqllib 目录的 misc 子目录中。

```
db2exfmt –d <dbname> -o refresh.exp -1
```

第四步 分析访问方案以确定性能问题的原因。例如，通过分析上面语句中的方案，如果 T 是大型表，那么对该表执行表扫描操作的成本将非常高。创建索引可以提高查询的性能。

说明工具

DB2 提供了一种全面的说明工具，该工具提供有关优化器为 SQL 或 XQuery 语句选择的访问方案的详细信息。存储说明数据的表可在所有受支持的平台上进行访问，它包含静态和动态 SQL 和 XQuery 语句的信息。几种工具或方法使您可灵活地捕获、显示和分析说明信息。

详细的优化器信息允许对访问方案进行深入的分析，该信息存储在说明表中，独立于实际的访问方案本身。使用下列其中一个或多个方法，从说明表获取信息:

- 使用 Visual Explain 来查看说明快照信息。

  从“控制中心”调用 Visual Explain 来查看查询访问方案的图形显示。可以分析静态 SQL 和 XQuery 语句以及动态 SQL 和 XQuery 语句。

  Visual Explain 允许您查看在另一个平台上捕获或获取的快照。例如，Windows客户机可将在 DB2 HP-UX 版服务器上生成的快照绘成图形。
- 使用 db2exfmt 工具来显示预格式化输出中的说明信息
- 使用 db2expln 和 dynexpln 工具。

  要查看静态 SQL 或 XQuery 语句的一个或多个程序包可用的访问方案，从命令行使用 db2expln 工具。db2expln 显示所选访问方案的实际实现。它不显示优化器信息。

  dynexpln 工具在自己内部使用 db2expln，该工具提供一种快速方法来说明不包含参数标记的动态 SQL 或 XQuery 语句。通过将输入 SQL 或 XQuery 语句转换为伪程序包中的静态语句，可从 dynexpln 内部使用 db2expln。当执行此操作时，该信息可能不是始终都是完全准确的。如果希望完全准确，那么使用说明工具。

  db2expln 工具通过检查生成的实际访问方案来对在运行时将发生的操作提供一个相对简洁的英语风格的概述。

- 编写您自己的针对说明表的查询。

  编写自己的查询便于处理输出和比较不同的查询或比较一段时间内的同一查询。

注: 命令行说明工具和其他工具（例如，db2batch、dynexpln 和 db2_all）位于 sqllib 目录的 misc 子目录中。如果将这些工具移出此路径，那么命令行方法可能不起作用。

下表汇总了 DB2 说明工具提供的各种工具和它们各自的特征。参考此表来选择最适合您的环境和需要的工具。

表 64. 说明设施工具

期望的特征	Visual Explain	说明表	db2exfmt	db2expln	dynexpln
GUI 界面	是				
文本输出			是	是	是
“快速和脏的”静态 SQL 和 XQuery 分析				是	
支持的静态 SQL 和 XQuery	是	是	是	是	
支持的动态 SQL 和 XQuery	是	是	是	是	是*
支持的 CLI 应用程序	是	是	是		
可用于 DRDA® 应用程序请求器		是			
详细的优化器信息	是	是	是		
适合对多个语句的分析		是	是	是	是
可从应用程序内访问的信息		是			
注: * 间接使用 db2expln; 存在一些局限性。					

显示在说明时有效的目录统计信息

说明工具捕获在说明语句时有效的统计信息。这些统计信息可能与系统目录中存储的统计信息不同，尤其在启用了收集实时统计信息时。如果填充了说明表但未创建说明快照，那么只有一部分统计信息记录在 EXPLAIN_OBJECT 表中。

为了捕获所有与正说明的语句相关的目录统计信息，在填充说明表的同时创建说明快照。然后，使用 SYSPROC.EXPLAIN_FORMAT_STATS 函数来格式化快照中的目录统计信息。

如果 db2exfmt 工具用来格式化说明信息，那么在收集了快照时，它将自动使用 SYSPROC.EXPLAIN_FORMAT_STATS 函数来显示目录统计信息。Visual Explain 自动显示快照中包含的所有统计信息。

SQL 和 XQuery 说明工具

db2expln 工具描述为 SQL 和 XQuery 语句选择的访问方案。当未捕获到说明数据时，它可用来获取所选访问方案的快速说明。对于静态 SQL 和 XQuery 语句，db2expln 检查存储在系统目录表中的程序包。对于动态 SQL 和 XQuery 语句，db2expln 检查查询高速缓存中的各节。

dynexpln 工具也可以用来描述为动态语句选择的访问方案。它为语句创建静态程序包，然后使用 db2expln 工具对它们进行描述。然而，因为可以通过 db2expln 检查动态 SQL 和 XQuery 语句，所以保留此实用程序只是为了实现向后兼容性。

这些说明工具（db2expln 和 dynexpln）位于您的实例 sqllib 目录的 bin 子目录中。如果 db2expln 和 dynexpln 不在当前目录中，它们必须在 PATH 环境变量中出现的某个目录中。

首次访问数据库时，db2expln 程序连接并使用 db2expln.bnd、db2exsrv.bnd 和 db2exdyn.bnd 文件来将它自己绑定到数据库。

要运行 db2expln，您必须对系统目录视图具有 SELECT 特权并对 db2expln、db2exsrv 和 db2exdyn 程序包具有 EXECUTE 特权。要运行 dynexpln，您必须对数据库具有 BINDADD 权限并且将用来连接至数据库的 SQL 模式必须存在，或您必须对数据库具有 EXPLICIT_SCHEMA 权限。要使用 db2expln 或 dynexpln 说明动态 SQL 和 XQuery 语句，您还必须具有正在被说明的查询语句所需要的所有特权。（注意，如果您具有 SYSADM 或 DBADM 权限，您将自动具有所有这些授权级别。）

dynexpln

dynexpln 工具仍可用，以便实现向后兼容性。然而，可以使用 db2expln 的**动态选项**来执行 dynexpln 的所有功能。

当使用 db2expln 的动态选项时，准备该语句作为真实的动态 SQL 或 XQuery 语句，且从查询高速缓存说明生成的方案。此说明输出方法提供比 dynexpln 更精确的访问方案，它将语句准备为静态 SQL 或 XQuery 语句。它也允许使用仅在动态 SQL 和 XQuery 语句中可用的功能部件，例如，参数标记。

dynexpln 的使用注意事项: 要说明动态语句，dynexpln 会为这些语句创建一个静态应用程序，然后调用 db2expln。要创建静态语句，dynexpln 生成一个含有这些语句的小 C 程序，然后调用 DB2 预编译器来创建程序包。（生成的 C 程序不完整，且不能被编译；它只包含足够预编译器构建该程序包用的信息。）

以下是 dynexpln 显示的常见消息:

- 来自 db2expln 的所有错误消息。

  由于 dynexpln 调用 db2expln，所以查看 db2expln 的大多数错误消息是可能的。
- 与数据库连接时出错。

  如果在与数据库连接时发生错误，那么会输出此消息。也将显示一个 CLI 错误消息，指示该连接未能完成的原因。更正错误的原因，再次运行 dynexpln。
- 在 dynexpln 运行之前，必须除去文件"<filename>"。

  如果运行 dynexpln 时，给定的文件存在，那么会显示此消息。除去该文件，或更改 DYNEXPLN_PACKAGE 环境变量的值，以更改将创建的文件名，然后再次运行 dynexpln。
- 在 dynexpln 运行之前，必须删除程序包"<creator>.<package>"。

  如果在运行 dynexpln 时，给定的文件存在，那么会显示此消息。删除该程序包，再运行；或更改 **DYNEXPLN_PACKAGE** 环境变量的值，以更改将创建的程序包的名称，然后再次运行 dynexpln。
- 写入文件"<filename>"时出错。

如果不能写入给定的文件，那么会显示此消息。确保 dynexpln 可以在当前目录中写入文件，并再次运行此工具。

- 读取输入文件"<filename>"时出错。

  如果不能读取用 **-f** 选项给出的文件，那么会显示此消息。确保该文件存在并且 dynexpln 可以读取该文件。然后再次运行 dynexpln。

环境变量：可以将两个不同的环境变量与 dynexpln 一起使用:

- **DYNEXPLN_OPTIONS** 是为语句构建程序包时使用的 SQL 和 XQuery预编译器选项。使用与通过 CLP 发出 PREP 命令时将要用的相同的语法变量。

  例如: DYNEXPLN_OPTIONS="OPTLEVEL 5 BLOCKING ALL"

- **DYNEXPLN_PACKAGE** 是在数据库中创建的程序包的名称。要描述的语句位于此程序包中。如果未定义此变量，那么赋予该程序包缺省值 **DYNEXPLN**。（只使用此环境变量中该名称的前八个字符。）

  该名称也用于创建 dynexpln 使用的中间文件的名称。

db2expln 和 dynexpln 输出的描述

在该输出中，每个程序包的说明信息显示在下列两个部分:

- 程序包信息，如绑定日期和相关的绑定选项
- 后跟要说明的 SQL 或 XQuery 语句的节信息，例如，节号。节信息下面，显示为 SQL 或 XQuery 语句选择的访问方案的说明输出。

一个访问方案或节的步骤以数据库管理器执行它们的顺序显示。每个主要步骤显示为一个向左对齐的标题，有关该步骤的信息以缩进形式显示在该标题下。缩进条出现在该访问方案的说明输出的左页边距中。这些条也标记操作的作用域。在返回到缩进的先前级别之前，在同一操作中处理更低级别的缩进（更远离右边）的操作。

请记住，选择的访问方案是基于输出中显示的原始 SQL 或 XQuery 语句的增强版本。例如，原始语句可能导致激活触发器和约束。另外，查询编译器的查询重写组件可以将 SQL 或 XQuery 语句重新编写为一种等效但更有效的格式。当优化器确定满足该语句的最有效的方案时，会在优化器使用的信息中包括所有这些因素。因此，在说明输出中显示的访问方案可能与对原始 SQL 或 XQuery 语句期望的访问方案完全不同。说明工具（包括说明表、SET CURRENT EXPLAIN 方式和 Visual Explain）显示用于优化的实际 SQL 或 XQuery 语句，其格式为 SQL 或 XQuery 风格的语句，它是通过逆向转换查询的内部表示而创建的。

当将 db2expln 或 dynexpln 的输出与说明工具的输出比较时，运算符标识选项（**-opids**）非常有用。每次 db2expln 或 dynexpln 开始处理说明工具中的新运算符时，就会将该新运算符标识号打印在说明方案的左边。运算符标识可以用于与各种格式的访问方案中的步骤匹配。注意，在说明工具输出中的运算符与 db2expln 和 dynexpln 显示的操作之间不会始终都存在一一对应的关系。

表访问信息: 此语句告知正在访问的表的名称和类型。它有两种使用格式:

1. 三种类型的常规表:
   - 访问表名:

     ```
     Access Table Name = schema.name  ID = ts,n
     ```

其中:

– *schema.name* 是正在访问的表的标准名称

– *ID* 是该表的 SYSCAT.TABLES 目录中对应的 TABLESPACEID 和 TABLEID

• 访问层次结构表名:

```
Access Hierarchy Table Name = schema.name  ID = ts,n
```

其中:

– *schema.name* 是正在访问的表的标准名称

– *ID* 是该表的 SYSCAT.TABLES 目录中对应的 TABLESPACEID 和 TABLEID

• 访问具体化查询表名称:

```
Access Materialized Query Table Name = schema.name  ID = ts,n
```

其中:

– *schema.name* 是正在访问的表的标准名称

– *ID* 是该表的 SYSCAT.TABLES 目录中对应的 TABLESPACEID 和 TABLEID

2. 两种类型的临时表:

• 访问临时表标识:

```
Access Temp Table  ID = tn
```

其中:

– *ID* 是 db2expln 指定的对应标识

• 访问已声明全局临时表标识:

```
Access Global Temp Table  ID = ts,tn
```

其中:

– *ID* 是该表（ts）的 SYSCAT.TABLES 目录中对应的 TABLESPACEID 以及由 db2expln（tn）指定的对应标识

在表访问语句之后，将提供附加语句以进一步描述该访问。这些语句以缩进形式显示在表访问语句之下。可能的语句包括:

- 列数
- 块访问
- 并行扫描
- 扫描伪指令
- 行访问方法
- 锁定意向
- 谓词
- 其他语句

列数

以下语句指示该表的每一行中所用的列数:

```
#Columns = n
```

块访问

以下语句指示表具有一个或多个在其上定义的维块索引:

```
Clustered by Dimension for Block Index Access
```

如果未显示此文本，那么未使用 DIMENSION 子句创建该表。

并行扫描

以下语句指示数据库管理器将使用几个子代理程序来并行读取该表:

```
Parallel Scan
```

如果未显示此文本，那么该表只能由一个代理程序（或子代理程序）读取。

扫描方向

以下语句指示数据库管理器将按倒序读取行:

```
Scan Direction = Reverse
```

如果未显示此文本，那么扫描方向为正向，这是缺省值。

行访问方法

将显示下列语句之一，指示如何访问表中的限定行:

- 关系扫描语句指示按顺序扫描该表，以查找限定行。
  - 以下语句指示将不执行数据的预取:

    ```
    Relation Scan
    | Prefetch: None
    ```

  - 以下语句指示优化器已预先确定了将预取的页数:

    ```
    Relation Scan
    | Prefetch: n Pages
    ```

  - 以下语句指示应该预取数据:

    ```
    Relation Scan
    | Prefetch: Eligible
    ```

  - 以下语句指示正在通过索引标识和访问限定行:

    ```
    Index Scan:  Name = schema.name  ID = xx
    | Index type
    | Index Columns:
    ```

    其中:

    - *schema.name* 是正在扫描的索引的标准名称
    - *ID* 是 SYSCAT.INDEXES 目录视图中的对应 IID 列。
    - 索引类型是下列其中一项:

      ```
      Regular Index (Not Clustered)
      Regular Index (Clustered)
      Dimension Block Index
      Composite Dimension Block Index
      XML 数据索引
      ```

      对于索引中的每一列，都有一行跟在后面。将以下列一种形式列示索引中的每一列:

```
n: column_name (Ascending)
n: column_name (Descending)
n: column_name (Include Column)
```

提供以下语句，以说明索引扫描的类型:

- 通过以下项显示索引的范围定界谓词:

```
#Key Columns = n
|  Start Key: xxxxx
|  Stop Key: xxxxx
```

其中 xxxxx 是下列其中一项:

- 索引开始
- 索引结束
- 包括的值: 或排除的值:

将会在该索引扫描中包括内含的键值。不会在扫描中包括排除的键值。下列其中一行将对键的每一部分给出该键的值:

```
n: 'string'
   n: nnn
   n: yyyy-mm-dd
   n: hh:mm:ss
   n: yyyy-mm-dd hh:mm:ss.uuuuuu
   n: NULL
   n: ?
```

如果出现文字串，那么仅显示前面 20 个字符。如果该文字串长度大于 20 个字符，那么在该文字串的最后显示 ...。一些键只有在执行该部分之后才能确定。因此会显示 ? 作为其值。

- 纯索引访问

如果可以从索引键获得所有需要的列，将出现此语句，且不会访问任何表数据。

- 以下语句指示将不执行索引页的预取:

```
Index Prefetch: None
```

- 以下语句指示应该预取索引页:

```
Index Prefetch: Eligible
```

- 以下语句指示将不执行数据页的预取:

```
Data Prefetch: None
```

- 以下语句指示应该预取数据页:

```
Data Prefetch: Eligible
```

- 如果存在可以传送至“索引管理器”以帮助限定索引条目的谓词，那么以下语句用于显示谓词数:

```
Sargable Index Predicate(s)
|   #Predicates = n
```

– 如果正在通过使用在访问方案中先前准备的行标识（RID）来访问限定行，将用下列语句指示它:

```
Fetch Direct Using Row IDs
```

如果表具有为它定义的一个或多个块索引，那么块或行标识都可以访问行。这由以下指示:

```
Fetch Direct Using Block or Row IOs
```

锁定意向

对于每个表访问，都会用以下语句显示将在表和行级别获取的锁定类型:

```
Lock Intents
|  Table: xxxx
|  Row  : xxxx
```

表锁定的可能值有:

- 互斥
- 意向互斥
- 无任何意向
- 意向共享
- 共享
- 共享意向互斥
- 超互斥
- 更新

行锁定的可能的值有:

- 互斥
- 下一键互斥（不在 db2expln 输出中显示）
- 无
- 共享
- 下一键共享
- 更新
- 下一键弱互斥
- 弱互斥

谓词

有两个语句，它们提供有关在访问方案中使用的谓词的信息:

1. 以下语句指示将对从分块索引中检索的每个数据块求出的谓词数。

   ```
   Block Predicates(s)
   |   #Predicates = n
   ```

2. 以下语句指示当正在访问数据时将求出的谓词数。谓词的计数不包括下推操作，例如，聚集或排序。

   ```
   Sargable Predicate(s)
   |   #Predicates = n
   ```

3. 以下语句指示一旦返回数据，将求出的谓词数:

   ```
   Residual Predicate(s)
   |   #Predicates = n
   ```

在上述语句中显示的谓词数也许未能反映在查询语句中提供的谓词数，因为谓词可以:

- 在同一个查询中应用多次
- 在查询优化过程中，通过添加隐式谓词来变换和扩展
- 在查询优化过程中，被变换和压缩成更少的谓词。

其他表语句

- 以下语句指示将只访问一行:

  ```
  Single Record
  ```

- 当此表访问所使用的隔离级别与该语句所用的不同时，就会出现以下语句:

  ```
  Isolation Level: xxxx
  ```

  由于许多原因，可能使用不同的隔离级别，包括:

  – 一个程序包是用“可重复读取”绑定的，这影响了引用完整性约束；为检查引用完整性约束而对父表进行的访问会被降级为“游标稳定性”隔离级别，以避免对此表挂起不必要的锁定。
  – 用“未落实的读”绑定的程序包发出 DELETE 或 UPDATE 语句；用于实际删除的表访问会被升级为“游标稳定性”。

- 如下语句指示如果有足够的排序堆内存可用，从临时表读取的部分或全部行将高速缓存到缓冲池外:

  ```
  Keep Rows In Private Memory
  ```

- 如果该表具有易变的基数属性集，那么会通过以下信息指出:

  ```
  Volatile Cardinality
  ```

临时表信息: 访问方案在瞬态或临时工作表中执行操作期间，它使用临时表来存储数据。仅当执行访问方案时，此表才存在。通常，当在访问方案中需要提前对子查询求值时，或当中间结果不适合可用内存时，会使用临时表。

如果需要创建临时表，那么会出现两个可能的语句中的一个。这些语句指示将创建一个临时表，并将行插入其中。该标识是引用临时表时为方便起见而由 db2expln 指定的标识。此标识以字母“t”为前缀，以指示该表是临时表。

- 以下语句指示将创建一个普通的临时表:

  ```
  Insert Into Temp Table  ID = tn
  ```

- 以下语句指示多个子代理程序将并行创建一个普通的临时表:

  ```
  Insert Into Shared Temp Table  ID = tn
  ```

- 以下语句指示将创建一个已排序的临时表:

  ```
  Insert Into Sorted Temp Table  ID = tn
  ```

- 以下语句指示多个子代理程序将并行创建一个已排序的临时表:

  ```
  Insert Into Sorted Shared Temp Table  ID = tn
  ```

- 以下语句指示将创建一个已声明全局临时表:

  ```
  Insert Into Global Temp Table  ID = ts,tn
  ```

- 以下语句指示多个子代理程序将并行创建一个已声明全局临时表:

  ```
  Insert Into Shared Global Temp Table  ID = ts,tn
  ```

- 以下语句指示将创建一个已排序已声明全局临时表:

```
Insert Into Sorted Global Temp Table  ID = ts,tn
```

- 以下语句指示多个子代理程序将并行创建一个已排序已声明全局临时表:

```
Insert Into Sorted Shared Global Temp Table  ID = ts,tn
```

上述每个语句都将后跟下列内容:

```
#Columns = n
```

它指示要插入临时表中的每一行有多少列。

已排序的临时表

已排序的临时表可以由类似如下的操作产生:

- ORDER BY
- DISTINCT
- GROUP BY
- Merge Join
- '= ANY' subquery
- '<> ALL' subquery
- INTERSECT 或 EXCEPT
- UNION（不带 ALL 关键字）

在已排序的临时表的原始创建语句后，可跟许多附加语句:

- 以下语句指示在排序中使用的键列数:

```
#Sort Key Columns = n
```

  对于排序键中的每一列，将会显示下列其中一行:

```
Key n: column_name (Ascending)
Key n: column_name (Descending)
Key n: (Ascending)
Key n: (Descending)
```

- 以下语句提供对行数和行大小的估计，以便在运行时可以分配最优的排序堆。

```
Sortheap Allocation Parameters:
#Rows     = n
Row Width = n
```

- 如果只需要排序结果的前几行，那么会显示下列内容:

```
Sort Limited To Estimated Row Count
```

- 对于在"对称多处理器"（SMP）环境中的排序，要执行的排序的类型由下列语句之一指示:

```
Use Partitioned Sort
Use Shared Sort
Use Replicated Sort
Use Round-Robin Sort
```

- 以下语句指示排序产生的结果是否留在排序堆中:

```
Piped
```

  而

```
Not Piped
```

如果指示管道排序，那么数据库管理器会将排序的输出保留在内存中，而不是将排序结果置于另一个临时表中。

- 以下语句指示在排序期间将除去重复的值:

```
Duplicate Elimination
```

- 如果正在排序中执行聚集，那么下列其中一个语句会指示它:

```
Partial Aggregation
Intermediate Aggregation
Buffered Partial Aggregation
Buffered Intermediate Aggregation
```

临时表完成

在包含用于创建临时表的下推操作的表访问之后（即，创建临时表操作是在表访问范围内发生的），有一个“完成”语句，它通过让临时表准备向后续的临时表访问提供行，来处理文件结尾。将显示下列其中一行:

```
Temp Table Completion  ID = tn
Shared Temp Table Completion  ID = tn
Sorted Temp Table Completion  ID = tn
Sorted Shared Temp Table Completion  ID = tn
```

表函数

表函数是用户定义的函数（UDF），它将数据以表的形式返回至语句。表函数由以下语句指示:

```
Access User Defined Table Function
|  Name = schema.funcname
|  Specific Name = specificname
|  SQL Access Level = accesslevel
|  Language = lang
|  Parameter Style = parmstyle
|  Fenced                      Not Deterministic
|  Called on NULL Input        Disallow Parallel
|  Not Federated               Not Threadsafe
```

特定名称唯一标识调用的表函数。其余行详细描述函数的属性。

连接信息: 有三种类型的连接:

- 散列连接
- 合并连接
- 嵌套循环连接。

当在一节中执行到要执行连接时，会显示下列其中一个语句:

```
Hash Join
   Merge Join
   Nested Loop Join
```

执行左外连接是可能的。左外连接由下列其中一个语句指示:

```
Left Outer Hash Join
   Left Outer Merge Join
   Left Outer Nested Loop Join
```

对于合并和嵌套循环连接，连接的外部表将是在输出中显示的先前访问语句所引用的表。连接的内部表将是在连接语句范围内包含的访问语句所引用的表。对于散列连接，将访问语句反向，将外部表包含在连接范围之内，而内部表出现在连接之前。

对于散列或合并连接，可能出现下列附加语句:

- 在某些环境中，连接只需要确定内部表中的任何行是否与外部表中的当前行匹配。它是通过下列语句来指示的:

  ```
  Early Out: Single Match Per Outer Row
  ```

- 在连接完成之后，应用谓词是可能的。将指示应用的谓词数，如下所示:

  ```
  Residual Predicate(s)
  |   #Predicates = n
  ```

对于散列连接，可能出现下列附加语句:

- 已根据内部表构建了散列表。如果在进行内部表访问时将散列表的构建压入一个谓词中，那么在访问内部表时用以下语句指示该构建:

  ```
  Process Hash Table For Join
  ```

- 当访问外部表时，可以构建一个探测表来提高连接的性能。在访问外部表时用以下语句指示探测表的构建:

  ```
  Process Probe Table For Hash Join
  ```

- 构建散列表所需的估计字节数由以下项表示:

  ```
  Estimated Build Size: n
  ```

- 构建探测表所需的估计字节数由以下项表示:

  ```
  Estimated Probe Size: n
  ```

对于嵌套循环连接，紧接在连接语句之后可能会出现以下附加语句:

```
Piped Inner
```

此语句指示连接的内部表是另一系列的操作的结果。这也称为*组合内连接*。

如果一个连接涉及两个以上的表，那么应从头到尾读取说明步骤。例如，假设说明输出具有下列数据流:

```
Access ..... W      Join
|  Access ..... X
   Join
|  Access ..... Y      Join
|  Access ..... Z
```

执行的步骤将是:

1. 提取 W 中合格的行。
2. 将 W 中的行与 X 中的（下一）行连接，并调用结果 P1（表示编号 1 的部分连接结果）。
3. 将 P1 与 Y 中的（下一）行连接，以创建 P2。
4. 将 P2 与 Z 中的（下一）行连接，以获取一个完整的结果行。
5. 如果 Z 中存在其他行，那么转至步骤 4。
6. 如果 Y 中存在其他行，那么转至步骤 3。
7. 如果 X 中存在其他行，那么转至步骤 2。
8. 如果 W 中存在其他行，那么转至步骤 1。

数据流信息: 在一个访问方案内，经常需要控制数据的创建以及数据从一系列操作到另一系列操作的流动。数据流的概念允许将一个访问方案内的一组操作当作一个单元来控制。数据流的开始由以下语句指示:

```
Data Stream n
```

其中，n 是由 db2expln 为方便引用而指定的唯一标识。数据流的结束由以下语句指示:

```
End of Data Stream n
```

这些语句之间的所有操作被认为是同一个数据流的一部分。

一个数据流有许多特征，在初始数据流语句之后可跟一个或多个语句，来描述这些特征:

- 如果数据流的操作依赖于访问方案中早先生成的值，那么数据流标记为:

  ```
  Correlated
  ```

- 类似于已排序的临时表，下列语句指示数据流的结果是否将保留在内存中:

  ```
  Piped
  ```

  而

  ```
  Not Piped
  ```

  同临时表的情况一样，如果执行时没有足够的内存，那么可能将管道数据流写入磁盘。该访问方案将提供这两种可能性。

- 以下语句指示此数据流只需要单个记录:

  ```
  Single Record
  ```

当访问一个数据流时，将会输出以下语句:

```
Access Data Stream n
```

插入、更新和删除信息: 这些 SQL 语句的说明文本是自我说明的。这些 SQL 操作的语句文本可能为:

```
Insert: Table Name = schema.name ID = ts,n
Update: Table Name = schema.name ID = ts,n
Delete: Table Name = schema.name ID = ts,n
Insert: Hierarchy Table Name = schema.name ID = ts,n
Update: Hierarchy Table Name = schema.name ID = ts,n
Delete: Hierarchy Table Name = schema.name ID = ts,n
Insert: Materialized Query Table = schema.name ID = ts,n
Update: Materialized Query Table = schema.name ID = ts,n
Delete: Materialized Query Table = schema.name ID = ts,n
Insert: Global Temporary Table ID = ts, tn
Update: Global Temporary Table ID = ts, tn
Delete: Global Temporary Table ID = ts, tn
```

块和行标识准备信息: 对于某些访问方案，如果对限定行和块标识（标识）排序并除去了重复的标识（如果是索引 OR 运算），或者在执行实际的表访问之前，使用一种技术来标识在所有正在访问的索引中出现的标识（如果是索引 AND 运算），那么会更有效。正如说明语句所指示，标识准备有下列三种主要用途:

- 以下语句指示“索引 OR 运算”用于准备限定标识的列表:

  ```
  Index ORing Preparation
    Block Index ORing Preparation
  ```

  *索引 OR 运算*是指建立多个索引访问，并组合结果以包括在访问的任何索引中出现的不同标识的一种技术。当通过 OR 关键字将谓词连接或存在 IN 谓词时，优化器将考虑索引 OR 运算。索引访问可以在相同索引或不同索引上进行。

- 标识准备的另一个用途是准备在列表预取期间要使用的输入数据，如下列任一项所指示:

  ```
  List Prefetch Preparation
     Block List Prefetch RID Preparation
  ```

- *索引 AND 运算*是指建立多个索引访问，并组合结果以包括在访问的所有索引中出现的标识的一种技术。索引 AND 运算的处理从下列其中一个语句开始:

  ```
  Index ANDing
     Block Index ANDing
  ```

  如果优化器已估计了结果集的大小，那么用以下语句显示该估计值:

  ```
  Optimizer Estimate of Set Size: n
  ```

  索引 AND 运算过滤器操作处理标识，并使用位过滤器技术来确定在访问的每个索引中出现的标识。以下语句指示正在为索引 AND 运算处理标识:

  ```
  Index ANDing Bitmap Build Using Row IDs
     Index ANDing Bitmap Probe Using Row IDs
     Index ANDing Bitmap Build and Probe Using Row IDs
     Block Index ANDing Bitmap Build Using Block IDs
     Block Index ANDing Bitmap Build and Probe Using Block IDs
     Block Index ANDing Bitmap Build and Probe Using Row IDs
     Block Index ANDing Bitmap Probe Using Block IDs and Build Using Row IDs
     Block Index ANDing Bitmap Probe Using Block IDs
     Block Index ANDing Bitmap Probe Using Row IDs
  ```

  如果优化器已估计了一个位图的结果集的大小，那么会用以下语句显示该估计值:

  ```
  Optimizer Estimate of Set Size: n
  ```

对于任何类型的标识准备，如果可以执行列表预取，那么将用以下语句指示它:

```
Prefetch: Enabled
```

聚集信息: 对满足指定标准的那些行执行聚集，如果存在这样的行，那么它们是由 SQL 语句谓词提供的。如果要执行某种聚集函数，会出现下列其中一个语句:

```
Aggregation
   Predicate Aggregation
   Partial Aggregation
   Partial Predicate Aggregation
   Intermediate Aggregation
   Intermediate Predicate Aggregation
   Final Aggregation
   Final Predicate Aggregation
```

谓词聚集指示在实际访问数据时，将聚集操作压入为一个谓词来处理。

在上述任何一种聚集语句之下将是要执行的聚集函数的类型指示:

```
Group By
   Column Function(s)
   Single Record
```

可从初始的 SQL 语句派生出特定的列函数。从索引读取单个记录，以满足 MIN 或 MAX 操作。

如果使用谓词聚集，那么在出现了聚集的表访问语句之后，将是聚集“完成”，它在每个组完成或文件结束时执行任何需要的处理。显示下列其中一行:

```
Aggregation Completion
   Partial Aggregation Completion
   Intermediate Aggregation Completion
   Final Aggregation Completion
```

并行处理信息: 并行执行 SQL 语句（使用分区内或分区间并行性）需要一些特殊的操作。以下描述用于并行方案的操作。

- 当运行分区内并行方案时，将使用几个子代理程序同时执行该方案的各个部分。这些子代理程序的创建由下列语句指示:

  ```
  Process Using n Subagents
  ```

- 当运行分区间并行方案时，该节被分为几个小节。每个小节被发送至一个或多个节点，以便运行。一个重要的小节是*协调程序小节*。协调程序小节是每个方案中的第一个小节。它首先获得控制权，并负责分发其他小节，然后将结果返回至调用应用程序。

  小节的分发由以下语句指示:

  ```
  Distribute Subsection #n
  ```

  可用八种方式的其中一种来确定接收小节的节点:

  – 以下示例指示将根据列值把小节发送至数据库分区组内的一个节点。

    ```
    Directed by Hash
    |  #Columns = n
    |  Partition Map ID = n, Nodegroup = ngname, #Nodes = n
    ```

  – 以下示例指示将把小节发送至一个预定的节点。（当该语句使用 NODENUMBER() 函数时，会常常看到这种情况。）

    ```
    Directed by Node Number
    ```

  – 以下示例指示将把小节发送至与给定数据库分区组中预定数据库分区号对应的节点。（当该语句使用 PARTITION() 函数时，会常常看到这种情况。）

    ```
    Directed by Partition Number
    |  Partition Map ID = n, Nodegroup = ngname, #Nodes = n
    ```

  – 以下示例指示将把小节发送至为应用程序的游标提供当前行的节点。

    ```
    Directed by Position
    ```

  – 以下示例指示只有编译该语句时确定的一个节点会接收该小节。

    ```
    Directed to Single Node
    |  Node Number = n
    ```

  – 以下示例之一指示将在协调程序节点上执行小节。

    ```
    Directed to Application Coordinator Node
       Directed to Local Coordinator Node
    ```

  – 以下示例指示将把小节发送到已列示的所有节点。

    ```
    Broadcast to Node List
    |  Nodes = n1, n2, n3, ...
    ```

  – 以下示例指示只有执行该语句时确定的一个节点会接收该小节。

    ```
       Directed to Any Node
    ```

- 表队列用于在一个分区数据库环境中的小节之间或一个对称多处理器（SMP）环境中的子代理程序之间移动数据。对表队列的描述为如下所示:

  – 以下语句指示正在将数据插入一个表队列:

```
Insert Into Synchronous Table Queue  ID = qn
   Insert Into Asynchronous Table Queue  ID = qn
   Insert Into Synchronous Local Table Queue  ID = qn
   Insert Into Asynchronous Local Table Queue  ID = qn
```

– 对于数据库分区表队列，插入表队列中的行的目的地由下列一个语句描述:

  将所有行发送至协调程序节点:

```
Broadcast to Coordinator Node
```

  将所有的行发送至给定小节运行所在的每个数据库分区:

```
Broadcast to All Nodes of Subsection n
```

  根据行中的值，将每一行发送至一个数据库分区:

```
Hash to Specific Node
```

  将每一行发送至执行该语句时所确定的数据库分区:

```
Send to Specific Node
```

  将每一行发送到随机确定的节点:

```
    Send to Random Node
```

– 在某些情况中，一个数据库分区表队列不得不临时将某些行溢出至临时表。这种可能性由下列语句标识:

```
Rows Can Overflow to Temporary Table
```

– 在包含将行插入一个表队列的下推操作的表访问之后，将有一个“完成”语句，它处理未能立即发送的行。显示下列其中一行:

```
Insert Into Synchronous Table Queue Completion  ID = qn
Insert Into Asynchronous Table Queue Completion  ID = qn
Insert Into Synchronous Local Table Queue Completion  ID = qn
Insert Into Asynchronous Local Table Queue Completion  ID = qn
```

– 以下语句指示正在从一个表队列中检索数据:

```
Access Table Queue  ID = qn
   Access Local Table Queue  ID = qn
```

  这些消息之后始终是有关检索列数的指示。

```
#Columns = n
```

– 如果表队列在接收端对行排序，那么表队列访问也将会有下列一条消息:

```
Output Sorted
   Output Sorted and Unique
```

  这些消息之后是有关用于排序操作的键数的指示。

```
    #Key Columns = n
```

  对于排序键中的每一列，显示下列其中一项:

```
    Key n: (Ascending)
    Key n: (Descending)
```

– 如果通过表队列的接收端将谓词应用于行，那么会显示下列消息:

```
Residual Predicate(s)
|   #Predicates = n
```

• 一个分区数据库环境中的某些小节明确地循环回至该小节的开始处，这由以下语句指示:

```
Jump Back to Start of Subsection
```

联合查询信息: 在联合数据库中执行 SQL 语句需要有对其他数据源执行部分语句的能力。

以下语句指示将要读取数据源:

```
Ship Distributed Subquery #n
|   #Columns = n
```

有可能对分布式子查询返回的数据应用谓词。将指示应用的谓词数，如下所示:

```
Residual Predicate(s)
|   #Predicates = n
```

发生在数据源的插入、更新或删除操作将由适当的消息指示:

```
Ship Distributed Insert #n
Ship Distributed Update #n
Ship Distributed Delete #n
```

如果在数据源显式锁定了表，将用以下语句指示它:

```
Ship Distributed Lock Table #n
```

针对数据源的 DDL 语句分割成两部分。在数据源上调用的部分由以下指示:

```
Ship Distributed DDL Statement #n
```

如果联合服务器是分区数据库，那么 DDL 语句的部分必须在目录节点上运行。这由以下指示:

```
Distributed DDL Statement #n Completion
```

单独提供每个分布式子语句的详细信息。以下描述了用于分布式语句的选项:

- 子查询的数据源通过下列其中一项显示:

  ```
  Server: server_name (type, version)
  Server: server_name (type)
  Server: server_name
  ```

- 如果数据源是关系式的，那么子语句的 SQL 显示为:

  ```
  SQL Statement:
              statement
  ```

  非关系数据源用下列项指示:

  ```
  Non-Relational Data Source
  ```

- 子语句中引用的昵称列示如下:

  ```
  Nicknames Referenced:
   schema.nickname ID = n
  ```

  如果数据源是关系式的，昵称的基本表显示为:

  ```
  Base = baseschema.basetable
  ```

  如果数据源是非关系数据源，那么昵称的源文件显示为:

  ```
  Source File = filename
  ```

- 在执行子语句之前，如果将值从联合服务器传送至数据源，那么值的数目将由下面这一项显示:

  ```
  #Input Columns: n
  ```

- 在执行子语句之后，如果将值从数据源传送至联合服务器，那么值的数目将由下面这一项显示:

```
#Output Columns: n
```

其他说明信息:

- 数据定义语言语句的节将在输出中用下列语句指示:

```
DDL Statement
```

  不对 DDL 语句提供其他说明输出。

- 用于可更新的专用寄存器的 SET 语句（如 **CURRENT EXPLAIN SNAPSHOT**）的节将在输出中用以下语句指示:

```
SET Statement
```

  不对 SET 语句提供其他说明输出。

- 如果 SQL 语句包含 DISTINCT 子句，那么输出中可能出现下列文本:

```
Distinct Filter  #Columns = n
```

  其中，n 是参与获取不同行的列数。要检索不同行的值，必须对行排序，以便可以跳过重复的行。如果数据库管理器不必明确消去重复行，如下列情况，那么不显示此语句:

  – 存在唯一的索引，且索引键中的所有列都是 DISTINCT 操作的一部分
  – 在排序期间可以消去的重复项。

- 如果下一个操作取决于特定的记录标识，将出现以下语句:

```
Positioned Operation
```

  如果位置操作是针对联合数据源的，那么该语句为:

```
Distributed Positioned Operation
```

  对于任何使用 WHERE CURRENT OF 语法的 SQL 语句，将会出现此语句。

- 如果存在必须应用于该结果但不能作为另一个操作的一部分来应用的谓词，将出现以下语句:

```
Residual Predicate Application
|   #Predicates = n
```

- 如果该 SQL 语句中有 UNION 运算符，将出现以下语句:

```
UNION
```

- 如果访问方案中有一个操作，它的唯一目的是产生供后续操作使用的行值，将出现以下语句:

```
   Table Constructor
|  n-Row(s)
```

  表构造函数可以用于将一个集合中的值变换成可以传送至后续操作的一系列行。当提示表构造函数输入下一行时，将出现以下语句:

```
Access Table Constructor
```

- 如果存在只在特定情况下才处理的操作，将出现下列语句:

```
Conditional Evaluation
|  Condition #n:
|  #Predicates = n
|  Action #n:
```

条件判定用于实现像 SQL CASE 语句这样的活动，或像引用完整性约束或触发器这样的内部机制。如果未显示任何操作，那么当该条件为真时，只处理数据处理操作。

- 如果在该访问方案中正在处理 ALL、ANY 或 EXISTS 子查询，那么将显示下列其中一个语句:
  - ANY/ALL 子查询
  - EXISTS 子查询
  - EXISTS SINGLE 子查询
- 在特定的 UPDATE 和 DELETE 操作之前，需要建立表内特定行的位置。这由下列语句指示:

  ```
  Establish Row Position
  ```

- 对于多维集群表上符合转出优化的删除将出现以下信息:

  ```
  CELL DELETE with deferred cleanup
  ```

  或者

  ```
  CELL DELETE with immediate cleanup
  ```

- 如果有要返回至该应用程序的行，将会显示以下语句:

  ```
  Return Data to Application
  |  #Columns = n
  ```

  如果该操作被下推至表访问中，它将需要一个完成阶段。此阶段为如下所示:

  ```
  Return Data Completion
  ```

- 如果正在调用存储过程，那么将出现下列信息:

  ```
  Call Stored Procedure
  |  Name = schema.funcname
  |  Specific Name = specificname
  |  SQL Access Level = accesslevel
  |  Language = lang
  |  Parameter Style = parmstyle
  |   Expected Result Sets = n
  |  Fenced                       Not Deterministic
  |  Called on NULL Input         Disallow Parallel
  |  Not Federated                Not Threadsafe
  ```

- 如果正在释放一个或多个 LOB 定位器，那么将出现下列信息:

  ```
  Free LOB Locators
  ```

db2expln 和 dynexpln 输出的示例

以下显示的示例可以帮助您了解 db2expln 和 dynexpln 的输出的布局和格式。这些示例是对 DB2 中提供的 SAMPLE 数据库运行的（除非显示不同）。每个示例都提供了简短的讨论。示例之间的主要区别用**粗体**显示。

示例一：无并行性: 本示例只请求一个列表，它包含所有职员的姓名、职务、部门名称和位置，以及他们所从事的项目的名称。此访问方案的实质是使用散列连接将每个指定的表中的相关数据连接在一起。因为没有索引可用，故访问方案在连接每个表时对它们进行关系扫描。

```
******************** PACKAGE ************************************

Package Name = "DOOLE"."EXAMPLE"  Version = ""
 Prep Date = 2002/01/04
 Prep Time = 14:05:00
```

```
 Bind Timestamp = 2002-01-04-14.05.00.415403

 Isolation Level          = Cursor Stability
 Blocking                 = Block Unambiguous Cursors
 Query Optimization Class = 5

 Partition Parallel       = No
 Intra-Partition Parallel = No

 SQL Path                 = "SYSIBM", "SYSFUN", "SYSPROC", "DOOLE"

---------- SECTION ----------------------------------------
Section = 1


SQL Statement:
    DECLARE EMPCUR CURSOR
    FOR
          SELECT e.lastname, e.job, d.deptname, d.location, p.projname
          FROM employee AS e, department AS d, project AS p
          WHERE e.workdept = d.deptno AND e.workdept = p.deptno


Estimated Cost        = 120.518692
Estimated Cardinality = 221.535980

(    6) Access Table Name = DOOLE.EMPLOYEE  ID = 2,5
                    |  #Columns = 3
                    |  Relation Scan
                    |  |  Prefetch: Eligible
                    |  Lock Intents
                    |  |  Table: Intent Share
                    |  |  Row  : Next Key Share
(    6) |  Process Build Table for Hash Join
(    2) Hash Join
                |  Estimated Build Size: 7111
                |  Estimated Probe Size: 9457
(    5) |  Access Table Name = DOOLE.PROJECT  ID = 2,7
                    |  |  #Columns = 2
                    |  |  Relation Scan
                    |  |  |  Prefetch: Eligible
                    |  |  Lock Intents
                    |  |  |  Table: Intent Share
                    |  |  |  Row  : Next Key Share
(    5) |  |  Process Build Table for Hash Join
(    3) |  Hash Join
                    |  |  Estimated Build Size: 5737
                    |  |  Estimated Probe Size: 6421
(    4) |  |  Access Table Name = DOOLE.DEPARTMENT  ID = 2,4
                    |  |  |  #Columns = 3
                    |  |  |  Relation Scan
                    |  |  |  |  Prefetch: Eligible
                    |  |  |  Lock Intents
                    |  |  |  |  Table: Intent Share
                    |  |  |  |  Row  : Next Key Share
(    4) |  |  |  Process Probe Table for Hash Join
(    1)    Return Data to Application
                    |  #Columns = 5

End of section


Optimizer Plan:

                                    RETURN
                                    (   1)
```

```
                 |
                                HSJOIN
                                     (   2)
                /      \
                  HSJOIN         TBSCAN
                  (   3)         (   6)
       /      \          |
      TBSCAN    TBSCAN   Table:
      (   4)    (   5)   DOOLE
         |         |       EMPLOYEE
  Table:      Table:
  DOOLE       DOOLE
  DEPARTMENT  PROJECT
```

该方案的第一部分访问 DEPARTMENT 和 PROJECT 表，并用散列连接来连接这些表。此连接的结果被连接到 EMPLOYEE 表。结果行返回至应用程序。

示例二：具有分区内并行性的单一分区方案: 本示例显示与第一个示例相同的 SQL 语句，但是此查询是为一个 4 路 SMP 机器编译的。

```
******************** PACKAGE ************************************

Package Name = "DOOLE"."EXAMPLE"  Version = ""
 Prep Date = 2002/01/04
 Prep Time = 14:12:38

 Bind Timestamp = 2002-01-04-14.12.38.732627

 Isolation Level          = Cursor Stability
 Blocking                 = Block Unambiguous Cursors
 Query Optimization Class = 5

 Partition Parallel       = No
 Intra-Partition Parallel = Yes (Bind Degree = 4)

 SQL Path                 = "SYSIBM", "SYSFUN", "SYSPROC", "DOOLE"

-------------------- SECTION ----------------------------------------
Section = 1


SQL Statement:
    DECLARE EMPCUR CURSOR
    FOR
          SELECT e.lastname, e.job, d.deptname, d.location, p.projname
          FROM employee AS e, department AS d, project AS p
          WHERE e.workdept = d.deptno AND e.workdept = p.deptno

Intra-Partition Parallelism Degree = 4

Estimated Cost        = 133.934692
Estimated Cardinality = 221.535980

(    2) Process Using 4 Subagents
(    7) |  Access Table Name = DOOLE.EMPLOYEE  ID = 2,5
                  |  |  #Columns = 3
                  |  |  Parallel Scan
                  |  |  Relation Scan
                  |  |  |  Prefetch: Eligible
                  |  |  Lock Intents
                  |  |  |  Table: Intent Share
                  |  |  |  Row  : Next Key Share
(    7) |  |  Process Build Table for Hash Join
(    3) |  Hash Join
                |  |  Estimated Build Size: 7111
                |  |  Estimated Probe Size: 9457
```

```
(    6) |  |  Access Table Name = DOOLE.PROJECT  ID = 2,7
               |  |  |  #Columns = 2
               |  |  |  Parallel Scan
                  |  |  |  Relation Scan
                  |  |  |  |  Prefetch: Eligible
                  |  |  |  Lock Intents
                  |  |  |  |  Table: Intent Share
                  |  |  |  |  Row  : Next Key Share
(    6) |  |  |  Process Build Table for Hash Join
(    4) |  |  Hash Join
               |  |  |  Estimated Build Size: 5737
                  |  |  |  Estimated Probe Size: 6421
(    5) |  |  |  Access Table Name = DOOLE.DEPARTMENT  ID = 2,4
                  |  |  |  |  #Columns = 3
               |  |  |  |  Parallel Scan
                  |  |  |  |  Relation Scan
                  |  |  |  |  |  Prefetch: Eligible
                  |  |  |  |  Lock Intents
                  |  |  |  |  |  Table: Intent Share
                  |  |  |  |  |  Row  : Next Key Share
(    5) |  |  |  |  Process Probe Table for Hash Join
(    2) |  Insert Into Asynchronous Local Table Queue  ID = q1
(    2) Access Local Table Queue  ID = q1  #Columns = 5
(    1)    Return Data to Application
                  |  #Columns = 5

End of section


Optimizer Plan:

                                  RETURN
                                  (   1)
                    |
                   LTQ
                  (   2)
                    |
                                 HSJOIN
                                (   3)
                /      \
                 HSJOIN          TBSCAN
                 (   4)          (   7)
          /      \          |
        TBSCAN    TBSCAN   Table:
        (   5)    (   6)   DOOLE
          |         |        EMPLOYEE
    Table:      Table:
    DOOLE       DOOLE
    DEPARTMENT  PROJECT
```

此方案几乎与第一个示例中的方案完全一样。主要区别是当该方案第一次启动时要创建四个子代理程序，而在该方案结束时表队列要收集每个子代理程序的工作结果，然后将这些结果返回至应用程序。

示例三：具有分区间并行性的多分区方案： 本示例显示与第一个示例相同的 SQL 语句，但此查询已在由三个数据库分区组成的分区数据库上进行了编译。

```
******************** PACKAGE ****************************************

Package Name = "DOOLE"."EXAMPLE"  Version = ""
 Prep Date = 2002/01/04
 Prep Time = 14:54:57

 Bind Timestamp = 2002-01-04-14.54.57.033666

 Isolation Level          = Cursor Stability
```

```
 Blocking                = Block Unambiguous Cursors
 Query Optimization Class = 5

 Partition Parallel      = Yes
 Intra-Partition Parallel = No

 SQL Path                = "SYSIBM", "SYSFUN", "SYSPROC", "DOOLE"

-------------------- SECTION -------------------------------------
Section = 1


SQL Statement:
    DECLARE EMPCUR CURSOR
    FOR
          SELECT e.lastname, e.job, d.deptname, d.location, p.projname
          FROM employee AS e, department AS d, project AS p
          WHERE e.workdept = d.deptno AND e.workdept = p.deptno


Estimated Cost        = 118.483406
Estimated Cardinality = 474.720032

Coordinator Subsection:(-----)    Distribute Subsection #2
                       |  Broadcast to Node List
                       |  |  Nodes = 10, 33, 55
(-----)    Distribute Subsection #3
                       |  Broadcast to Node List
                       |  |  Nodes = 10, 33, 55
(-----)    Distribute Subsection #1
                       |  Broadcast to Node List
                       |  |  Nodes = 10, 33, 55
(    2)    Access Table Queue  ID = q1  #Columns = 5
(    1)    Return Data to Application
                       |  #Columns = 5

Subsection #1:(    8)    Access Table Queue  ID = q2  #Columns = 2
(    3)    Hash Join
                       |  Estimated Build Size: 5737
                       |  Estimated Probe Size: 8015
(    6)    |  Access Table Queue  ID = q3  #Columns = 3
(    4)    |  Hash Join
                       |  |  Estimated Build Size: 5333
                       |  |  Estimated Probe Size: 6421
(    5)    |  |  Access Table Name = DOOLE.DEPARTMENT  ID = 2,4
                       |  |  |  #Columns = 3
                       |  |  |  Relation Scan
                       |  |  |  |  Prefetch: Eligible
                       |  |  |  Lock Intents
                       |  |  |  |  Table: Intent Share
                       |  |  |  |  Row  : Next Key Share
(    5)    |  |  |  Process Probe Table for Hash Join
(    2)    Insert Into Asynchronous Table Queue  ID = q1
                       |  Broadcast to Coordinator Node
                       |  Rows Can Overflow to Temporary Table

                Subsection #2:
(    9)    Access Table Name = DOOLE.PROJECT  ID = 2,7
                       |  #Columns = 2
                       |  Relation Scan
                       |  |  Prefetch: Eligible
                       |  Lock Intents
                       |  |  Table: Intent Share
                       |  |  Row  : Next Key Share
(    9)    |  Insert Into Asynchronous Table Queue  ID = q2
                       |  |  Hash to Specific Node
                       |  |  Rows Can Overflow to Temporary Tables
```

```
(    8)    Insert Into Asynchronous Table Queue Completion  ID = q2

Subsection #3:
(    7)    Access Table Name = DOOLE.EMPLOYEE  ID = 2,5
                        |  #Columns = 3
                        |  Relation Scan
                        |  |  Prefetch: Eligible
                        |  Lock Intents
                        |  |  Table: Intent Share
                        |  |  Row  : Next Key Share
(    7)    |  Insert Into Asynchronous Table Queue  ID = q3
                        |  |  Hash to Specific Node
                        |  |  Rows Can Overflow to Temporary Tables
(    6)    Insert Into Asynchronous Table Queue Completion  ID = q3

End of section


Optimizer Plan:

                                    RETURN
                                    (   1)
                                      |
                                     BTQ                                (   2)
                                      |
                                    HSJOIN
                                    (   3)
                                  /      \
                                HSJOIN          DTQ
                                (   4)        (   8)
                              /      \          |
                            TBSCAN     DTQ     TBSCAN
                            (   5)   (   6)   (   9)
                              |        |       |
                          Table:     TBSCAN   Table:
                          DOOLE      (   7)   DOOLE
                          DEPARTMENT   |      PROJECT
                                       |       Table:
                                       |       DOOLE
                                       |       EMPLOYEE
```

此方案与第一个示例中的方案有完全相同的细节，但是该节分成四个小节。这些小节具有下列任务:

- **协调程序小节**。此小节协调其他小节。在此方案中，它导致分发其他小节，然后使用表队列来收集要返回至应用程序的结果。
- **小节 #1**。此小节扫描表队列 q2，并使用散列连接将它与表队列 q3 的数据连接。接着第二个散列连接将表 DEPARTMENT 的数据添加进来。然后使用表队列 q1 将连接的行发送到协调程序小节。
- **小节 #2**。此小节扫描 PROJECT 表，然后将结果散列到特定节点。这些结果由小节 #1 读取。
- **小节 #3**。此小节扫描 EMPLOYEE 表，然后将结果散列到特定节点。这些结果由小节 #1 读取。

示例四: 具有分区间并行性和分区内并行性的多分区方案: 本示例显示与第一个示例相同的 SQL 语句，但此查询已在由三个数据库分区组成的分区数据库上进行了编译，这些数据库分区的每一个都在四路 SMP 机器上。

```
******************** PACKAGE ****************************************

Package Name = "DOOLE"."EXAMPLE"  Version = ""
 Prep Date = 2002/01/04
```

```
 Prep Time = 14:58:35

 Bind Timestamp = 2002-01-04-14.58.35.169555

 Isolation Level          = Cursor Stability
 Blocking                 = Block Unambiguous Cursors
 Query Optimization Class = 5

 Partition Parallel       = Yes
 Intra-Partition Parallel = Yes (Bind Degree = 4)

 SQL Path                 = "SYSIBM", "SYSFUN", "SYSPROC", "DOOLE"

-------------------- SECTION -------------------------------------
Section = 1


SQL Statement:
    DECLARE EMPCUR CURSOR
    FOR
          SELECT e.lastname, e.job, d.deptname, d.location, p.projname
          FROM employee AS e, department AS d, project AS p
          WHERE e.workdept = d.deptno AND e.workdept = p.deptno

Intra-Partition Parallelism Degree = 4

Estimated Cost        = 145.198898
Estimated Cardinality = 474.720032

Coordinator Subsection:(-----)    Distribute Subsection #2
                       |  Broadcast to Node List
                       |  |  Nodes = 10, 33, 55
(-----)    Distribute Subsection #3
                       |  Broadcast to Node List
                       |  |  Nodes = 10, 33, 55
(-----)    Distribute Subsection #1
                       |  Broadcast to Node List
                       |  |  Nodes = 10, 33, 55
(    2)    Access Table Queue  ID = q1  #Columns = 5
(    1)    Return Data to Application
                       |  #Columns = 5

Subsection #1:(    3)    Process Using 4 Subagents
(   10)    |  Access Table Queue  ID = q3  #Columns = 2
(    4)    |  Hash Join
                       |  |  Estimated Build Size: 5737
                       |  |  Estimated Probe Size: 8015
(    7)    |  |  Access Table Queue  ID = q5  #Columns = 3
(    5)    |  |  Hash Join
                       |  |  |  Estimated Build Size: 5333
                       |  |  |  Estimated Probe Size: 6421
(    6)    |  |  |  Access Table Name = DOOLE.DEPARTMENT  ID = 2,4
                       |  |  |  |  #Columns = 3
                    |  |  |  |  Parallel Scan
                       |  |  |  |  Relation Scan
                       |  |  |  |  |  Prefetch: Eligible
                       |  |  |  |  Lock Intents
                       |  |  |  |  |  Table: Intent Share
                       |  |  |  |  |  Row  : Next Key Share
(    6)    |  |  |  |  Process Probe Table for Hash Join
(    3)    |  Insert Into Asynchronous Local Table Queue  ID = q2
(    3)    Access Local Table Queue  ID = q2  #Columns = 5
(    2)    Insert Into Asynchronous Table Queue  ID = q1
                       |  Broadcast to Coordinator Node
                       |  Rows Can Overflow to Temporary Table

               Subsection #2:
```

```
(   11)    Process Using 4 Subagents
(   12)    |  Access Table Name = DOOLE.PROJECT  ID = 2,7
                       |  |  #Columns = 2
                       |  |  Parallel Scan
                       |  |  Relation Scan
                       |  |  |  Prefetch: Eligible
                       |  |  Lock Intents
                       |  |  |  Table: Intent Share
                       |  |  |  Row  : Next Key Share
(   11)    |  Insert Into Asynchronous Local Table Queue  ID = q4
(   11)    Access Local Table Queue  ID = q4  #Columns = 2
(   10)    Insert Into Asynchronous Table Queue  ID = q3
                       |  Hash to Specific Node
                       |  Rows Can Overflow to Temporary Tables

               Subsection #3:
(    8)    Process Using 4 Subagents
(    9)    |  Access Table Name = DOOLE.EMPLOYEE  ID = 2,5
                       |  |  #Columns = 3
                       |  |  Parallel Scan
                       |  |  Relation Scan
                       |  |  |  Prefetch: Eligible
                       |  |  Lock Intents
                       |  |  |  Table: Intent Share
                       |  |  |  Row  : Next Key Share
(    8)    |  Insert Into Asynchronous Local Table Queue  ID = q6
(    8)    Access Local Table Queue  ID = q6  #Columns = 3
(    7)    Insert Into Asynchronous Table Queue  ID = q5
                       |  Hash to Specific Node
                       |  Rows Can Overflow to Temporary Tables

End of section


Optimizer Plan:

                                  RETURN
                                  (   1)
                                    |
                                   BTQ
                                  (   2)
                                    |
                                   LTQ
                                  (   3)
                                    |
                                  HSJOIN
                                  (   4)
                                 /      \
                              HSJOIN    DTQ
                              (   5)   (  10)
                            /      \      |
                         TBSCAN    DTQ    LTQ
                         (   6)   (   7)  (  11)
                           |        |      |
                      Table:       LTQ    TBSCAN
                      DOOLE       (   8)  (  12)
                      DEPARTMENT    |       |
                                    TBSCAN  Table:
                                    (   9)  DOOLE
                                      |       PROJECT
                                    Table:
                                    DOOLE
                                    EMPLOYEE
```

此方案与第三个示例中的方案相似，不同的是多个子代理程序执行每个小节。另外，在每个小节结束时，在将合格行插入要散列到特定节点的第二个表队列之前，本地表队列将收集所有子代理程序的结果。

示例五：联合数据库方案： 本示例显示与第一个示例相同的 SQL 语句，但已在一个联合数据库上编译此查询，在这个联合数据库中，表 DEPARTMENT 和 PROJECT 在数据源上，而表 EMPLOYEE 在联合服务器上。

```
******************** PACKAGE ****************************************

Package Name = "DOOLE"."EXAMPLE"  Version = ""
 Prep Date = 2002/01/11
 Prep Time = 13:52:48

 Bind Timestamp = 2002-01-11-13.52.48.325413

 Isolation Level          = Cursor Stability
 Blocking                 = Block Unambiguous Cursors
 Query Optimization Class = 5

 Partition Parallel       = No
 Intra-Partition Parallel = No

 SQL Path                 = "SYSIBM", "SYSFUN", "SYSPROC", "DOOLE"

-------------------- SECTION ---------------------------------------
Section = 1


SQL Statement:
    DECLARE EMPCUR CURSOR
    FOR
          SELECT e.lastname, e.job, d.deptname, d.location, p.projname
          FROM employee AS e, department AS d, project AS p
          WHERE e.workdept = d.deptno AND e.workdept = p.deptno


Estimated Cost        = 1804.625000
Estimated Cardinality = 112000.000000

(    7) Ship Distributed Subquery #2
                |  #Columns = 2
(    2) Hash Join
                |  Estimated Build Size: 48444
                |  Estimated Probe Size: 232571
(    6) |  Access Table Name = DOOLE.EMPLOYEE  ID = 2,5
                |  |  #Columns = 3
                |  |  Relation Scan
                |  |  |  Prefetch: Eligible
                |  |  Lock Intents
                |  |  |  Table: Intent Share
                |  |  |  Row  : Next Key Share
(    6) |  |  Process Build Table for Hash Join
(    3) |  Hash Join
                |  |  Estimated Build Size: 7111
                |  |  Estimated Probe Size: 64606
(    4) |  |  Ship Distributed Subquery #1
                |  |  |  #Columns = 3
(    1)    Return Data to Application
                |  #Columns = 5

Distributed Substatement #1:
(    4) Server: REMOTE  (DB2/UDB 8.1)
        SQL Statement:

                        SELECT A0."DEPTNO", A0."DEPTNAME", A0."LOCATION"
             FROM "DOOLE"."DEPARTMENT" A0
        Nicknames Referenced:
                     DOOLE.DEPARTMENT  ID = 32768  Base = DOOLE.DEPARTMENT
        #Output Columns = 3
```

```
Distributed Substatement #2:
(    7) Server: REMOTE  (DB2/UDB 8.1)
        SQL Statement:

                        SELECT A0."DEPTNO", A0."PROJNAME"
             FROM "DOOLE"."PROJECT" A0
        Nicknames Referenced:
                     DOOLE.PROJECT  ID = 32769  Base = DOOLE.PROJECT
        #Output Columns = 2

End of section

Optimizer Plan:

                                      RETURN
                                      (   1)
                    |
                                    HSJOIN
                                (   2)
                            /      \
                          HSJOIN           SHIP
                          (   3)          (   7)
                      /      \            |
                   SHIP      TBSCAN   Nickname:
                  (   4)     (   6)   DOOLE
                     |          |        PROJECT
              Nickname:    Table:   DOOLE       DOOLE
              DEPARTMENT   EMPLOYEE
```

此方案与第一个示例中的方案有完全相同的块，但有两个表的数据来自数据源。这两个表是通过分布式子查询访问的，在本案例中，这些分布式子查询只是从那些表中选择所有行。在将数据返回至联合服务器之后，它便与来自本地表的数据连接到一起。

示例 6: XANDOR 和 XISCAN 运算符: 此示例显示了 XANDOR 运算符如何将对同一个表 *XISCANTABLE* 定义的两个独立基于 XML 数据的索引（*IDX1* 和 *IDX2*）的 XISCAN 扫描合并在一起。

```
IBM DB2 Database SQL Explain Tool

****************** DYNAMIC ***************************************

=================== STATEMENT ============================================

                Isolation Level          = Cursor Stability
                Blocking                 = Block Unambiguous Cursors
                Query Optimization Class = 5

                Partition Parallel       = No
                Intra-Partition Parallel = No

                SQL Path                 = "SYSIBM", "SYSFUN", "SYSPROC", "ATTALURI"


Query Statement:
    xquery
    for $c in db2-fn:xmlcolumn("XISCANTABLE.XMLCOL ")/a[@x="1" ]/b[@y=
                   "2" ] return $c

Section Code Page = 819

Estimated Cost = 192.266113
Estimated Cardinality = 1.800000

(    6) Index ANDing and ORing over XML
                |  Xpath is
                |  |  /child::element(a)[./child::element(b)
```

```
                |  |  /attribute::attribute(y)(:Index Search over XML 1:)
                |  |   and ./attribute::attribute(x)(:Index Search over XML 2:)
        |  |  ]
                |  Index Search over XML 1
                |  |  Access Table Name = ATTALURI.XISCANTABLE
                |  |  |  Index Scan over XML:  Name = ATTALURI.IDX1  ID = 6
                |  |  |  |  Physical Index over XML
                |  |  |  |  Index Columns:
                |  |  |  |  |  1: XMLCOL (Ascending)
                |  |  |  |  #Key Columns = 4
                |  |  |  |  Start Key: Inclusive Value
                |  |  |  |  |  |  1: ?
                |  |  |  |  |  |  2: ?
                |  |  |  |  |  |  3: ?
                |  |  |  |  |  |  4: ?
                |  |  |  |  Stop Key: Inclusive Value
                |  |  |  |  |  |  1: ?
                |  |  |  |  |  |  2: ?
                |  |  |  |  Index-Only Access
                |  |  |  |  Index Prefetch: None
                |  |  |  |  Isolation Level: Uncommitted Read
                |  |  |  |  Lock Intents
                |  |  |  |  Table: Intent None
                |  |  |  |  Row  : None
                |  |  |  |  StopKey = StartKey
                |  |  |  |  Value Start Key = ?
                |  Index Search over XML 2
                |  |  Access Table Name = ATTALURI.XISCANTABLE
                |  |  |  Index Scan over XML:  Name = ATTALURI.IDX2  ID = 4
                |  |  |  |  Physical Index over XML
                |  |  |  |  Index Columns:
                |  |  |  |  |  1: XMLCOL (Ascending)
                |  |  |  |  #Key Columns = 4
                |  |  |  |  Start Key: Inclusive Value
                |  |  |  |  |  |  1: ?
                |  |  |  |  |  |  2: ?
                |  |  |  |  |  |  3: ?
                |  |  |  |  |  |  4: ?
                |  |  |  |  Stop Key: Inclusive Value
                |  |  |  |  |  |  1: ?
                |  |  |  |  |  |  2: ?
                |  |  |  |  Index-Only Access
                |  |  |  |  Index Prefetch: None
                |  |  |  |  Isolation Level: Uncommitted Read
                |  |  |  |  Lock Intents
                |  |  |  |  Table: Intent None
                |  |  |  |  Row  : None
                |  |  |  |  StopKey = StartKey
                |  |  |  |  Value Start Key = ?
(    5) Insert Into Sorted Temp Table  ID = t1
                |  #Columns = 1
                |  #Sort Key Columns = 1
                |  |  Key 1: (Ascending)
                |  Sortheap Allocation Parameters:
                |  |  #Rows     = 2
                |  |  Row Width = 16
                |  Piped
                |  Duplicate Elimination
(    4) List Prefetch Preparation
(    4) |  Access Table Name = ATTALURI.XISCANTABLE  ID = 2,16
                |  |  #Columns = 1
                |  |  Fetch Using Prefetched List
                |  |  |  |  Prefetch: Eligible
                |  |  |  Lock Intents
                |  |  |  |  Table: Intent Share
                |  |  |  |  Row  : Next Key Share
(    2) Nested Loop Join
                |  Piped Inner
(    9) |  XML Doc Navigation
                |  |  Navigator is
```

```
                |  |  |  /fn:root($CONTEXT_NODE$())/child::element(a)(:#Xpath Predicates = 1:)
                |  |  |  [./child::element(b)(:Output nodeSeqRef :)
                |  |  |  (:#Xpath Predicates = 1:)
                |  |  |  /attribute::attribute(y) and
                |  |  |  ./attribute::attribute(x)]
(    1) Iterate over XML sequence for Xquery bindout
(    1)    Return Data to Application
                |  #Columns = 1

End of section


Optimizer Plan:

                                                    RETURN
                                                    (   1)
                                       |
                                                    NLJOIN
                                                (   2)
                                    /         \
                                                FETCH     XSCAN
                                                              (----)    (   9)
                                /           \
                                      RIDSCN     Table:
                                      (   4)     ATTALURI
                                          |        XISCANTABLE
                                       SORT
                                      (   5)
                                |
                                                XANDOR
                                           (   6)
                        /---------/      \--------\
                              XISCAN                          XISCAN
                              (   6)                          (   6)
                    /         \                     /         \
                  Index:          Table:          Index:          Table:
                       ATTALURI        ATTALURI          ATTALURI       ATTALURI
                       IDX1            XISCANTABLE       IDX2           XISCANTABLE
```

示例 7: XSCAN 运算符: 此示例显示了 XSCAN 运算符如何出现在访问方案中。此运算符处理由嵌入循环连接运算符（NLJOIN）传递的节点引用。它在访问方案中不用直接输入表示。

```
IBM DB2 Database SQL Explain Tool

******************** DYNAMIC ***************************************

=================== STATEMENT ==========================================

                Isolation Level          = Cursor Stability
                Blocking                 = Block Unambiguous Cursors
                Query Optimization Class = 5

                Partition Parallel       = No
                Intra-Partition Parallel = No

                SQL Path                 = "SYSIBM", "SYSFUN", "SYSPROC", "ATTALURI"


Query Statement:
    xquery
    for $b in db2-fn:xmlcolumn("XISCANTABLE.XMLCOL" )//book[position()<=
                    2] return $b

Section Code Page = 819

Estimated Cost = 779592.625000
Estimated Cardinality = 540000000.000000
```

```
(    4) Access Table Name = ATTALURI.XISCANTABLE  ID = 2,16
                |  #Columns = 1
                   |  Relation Scan
                   |  |  Prefetch: Eligible
                   |  Lock Intents
                   |  |  Table: Intent Share
                   |  |  Row  : Next Key Share
(    3) Nested Loop Join
                |  Piped Inner
(    6) |  XML Doc Navigation
                |  |  Navigator is
                |  |  |  /fn:root($CONTEXT_NODE$())/descendant-or-self::node()(:Output nodeSeqRef :)
          |  |  |
(    5) |  Nested Loop Join
                |  |  Piped Inner
(   11) |  |  XML Doc Navigation
                |  |  |  Navigator is
                |  |  |  |  /fn:root($CONTEXT_NODE$())/child::element(book)(:Output nodeSeqRef :)
          |  |  |  |
(   10) |  |  Aggregation
                |  |  |  Column Function(s)
(    9) |  |  Nested Loop Join
                |  |  |  Piped Inner
(   12) |  |  |  Unnest input XML sequence into stream of items with item number
(    8) |  |  Nested Loop Join
                |  |  |  Piped Inner
(   14) |  |  |  Table Constructor
                |  |  |  |  1-Row(s)
(    2) Nested Loop Join
                |  Piped Inner
(   16) |  Unnest input XML sequence into stream of items
(    1) Iterate over XML sequence for Xquery bindout
(    1)    Return Data to Application
                |  #Columns = 1

End of section

Optimizer Plan:

                                    RETURN
                                    (   1)
                    |
                                    NLJOIN
                              (   2)
                 /-/       \-----\
                       NLJOIN             *
                       (   3)             *
           /       \           |
           TBSCAN       NLJOIN      TBSCAN
           (   4)       (   5)      (  16)
        |         /      \         |
         Table:      XSCAN   *      TFunc:
         ATTALURI    (   6)  *      SYSIBM
         XISCANTABLE         |      GENROW
                                        NLJOIN
                                                              (   8)
                     /      \
                                NLJOIN     FILTER
                                     (   9)                                    (  13)
                /      \       |
                         GRPBY   TBSCAN  TBSCAN
                         (  10)  (  12)  (  14)
                |        |       |
                         XSCAN   TFunc:  TFunc:
                         (  11)  SYSIBM  SYSIBM
                                 GENROW  GENROW
```

示例 8: XISCAN 运算符: 此示例显示了 XISCAN 运算符如何扫描对表 *XISCANTABLE* 定义的 XML 数据索引 *IDX1*。

```
IBM DB2 Database SQL Explain Tool

******************** DYNAMIC ************************************

=================== STATEMENT ========================================

               Isolation Level          = Cursor Stability
               Blocking                 = Block Unambiguous Cursors
               Query Optimization Class = 5

               Partition Parallel       = No
               Intra-Partition Parallel = No

               SQL Path                 = "SYSIBM", "SYSFUN", "SYSPROC", "ATTALURI"


Query Statement:
    xquery
    for $c in db2-fn:xmlcolumn("XISCANTABLE.XMLCOL ")/a[@x="1" ] return
                    $c


Section Code Page = 819

Estimated Cost = 1666.833862
Estimated Cardinality = 18.000000

(    6) Access Table Name = ATTALURI.XISCANTABLE
                |  Index Scan over XML:  Name = ATTALURI.IDX1  ID = 4
                |  |  Physical Index over XML
                |  |  Index Columns:
                |  |  |  1: XMLCOL (Ascending)
                |  #Key Columns = 2
                |  |  Start Key: Inclusive Value
                |  |  |  |  1: ?
                |  |  |  |  2: ?
                |  |  Stop Key: Inclusive Value
                |  |  |  |  1: ?
                |  |  |  |  2: ?
                |  Index-Only Access
                |  Index Prefetch: None
                |  Isolation Level: Uncommitted Read
                |  |  Lock Intents
                |  |  Table: Intent None
                |  |  Row  : None
                |  StopKey = StartKey
                |  Value Start Key = ?
                |  Xpath is
                |  |  /child::element(a)/attribute::attribute(x)
(    5) Insert Into Sorted Temp Table  ID = t1
                |  #Columns = 1
                |  #Sort Key Columns = 1
                |  |  Key 1: (Ascending)
                |  Sortheap Allocation Parameters:
                |  |  #Rows     = 18
                |  |  Row Width = 16
                |  Piped
                |  Duplicate Elimination
(    4) List Prefetch Preparation
(    4) |  Access Table Name = ATTALURI.XISCANTABLE  ID = 2,16
                |  |  #Columns = 1
                |  |  Fetch Using Prefetched List
                |  |  |  |  Prefetch: Eligible
                |  |  Lock Intents
```

```
                  |  |  |  Table: Intent Share
                  |  |  |  Row  : Next Key Share
(    2) Nested Loop Join
                  |  Piped Inner
(    7) |  XML Doc Navigation
                  |  |  Navigator is
                  |  |  |  /fn:root($CONTEXT_NODE$())/child::element(a)
                  |  |     (:Output nodeSeqRef :)
                  |  |  |  (:#Xpath Predicates = 1:)
                  |  |  |  /attribute::attribute(x)
(    1) Iterate over XML sequence for Xquery bindout
(    1)    Return Data to Application
                  |  #Columns = 1

End of section


Optimizer Plan:

                                        RETURN
                                        (   1)
                          |
                                        NLJOIN
                                  (   2)
                             /      \
                                        FETCH     XSCAN
                                        (----)    (   7)
                        /      \
                               RIDSCN     Table:
                               (   4)     ATTALURI
                                  |         XISCANTABLE
                               SORT
                               (   5)
                     |
                               XISCAN
                               (   6)
                  /      \
                  Index:       Table:
                  ATTALURI     ATTALURI
                  IDX1         XISCANTABLE
```

说明信息的说明表和组织

所有说明信息都是按说明实例的概念组织的。一个说明实例代表说明工具对一个或多个 SQL 或 XQuery 语句的一次调用。在一个说明实例中捕获的说明信息包括编译环境以及为满足要编译的 SQL 或 XQuery 语句所选的访问方案。例如，一个说明实例可以由以下其中任何一项组成:

- 静态查询语句的一个程序包中所有合格的 SQL 或 XQuery 语句。对 SQL 语句（包括用于查询 XML 数据的语句），可以捕获 CALL、复合 SQL（动态）、DELETE、INSERT、MERGE、REFRESH、SELECT、SET INTEGRITY、SELECT INTO、UPDATE、VALUES 和 VALUES INTO 语句的说明信息。对 XQuery 语句，可以获取 XQUERY db2-fn:xmlcolumn 和 XQUERY db2-fn:sqlquery 语句的说明信息。

  注: REFRESH TABLE 和 SET INTEGRITY 语句不是静态编译的，只能动态编译。
- 增量绑定 SQL 语句的一个特定的 SQL 语句
- 动态 SQL 语句的一个特定的 SQL 语句
- 每个 EXPLAIN SQL 语句（无论是动态还是静态）

说明表信息反映访问方案中运算符和数据对象之间的关系。下图显示这些表之间的关系。

说明信息存储在下列表中:

表 65. 存储说明数据的关系表

表名	描述
EXPLAIN_ARGUMENT	包含有关每个个别运算符的唯一特征的信息（如果存在的话）。
EXPLAIN_INSTANCE	所有说明信息的主控制表。说明表中的每一行数据都显式地链接至此表中唯一的一行。有关要说明的 SQL 或 XQuery 语句的源和环境信息的基本信息保存在此表中。
EXPLAIN_OBJECT	标识那些生成用来满足 SQL 或 XQuery 语句的访问方案所需的数据对象。
EXPLAIN_OPERATOR	包含查询编译器为满足 SQL 或 XQuery 语句所需的所有运算符。
EXPLAIN_PREDICATE	标识特定的运算符所应用的谓词。
EXPLAIN_STATEMENT	包含对于不同级别的说明信息而存在的 SQL 或 XQuery 语句的文本。用户输入的原始 SQL 或 XQuery 语句与优化器用于选择访问方案的版本一起存储在此表中。 当请求说明快照时，会记录其他说明信息来描述查询优化器选择的访问方案。此信息使用 Visual Explain 需要的格式存储在 EXPLAIN_STATEMENT 表的 SNAPSHOT 列中。其他应用程序不能使用此格式。
EXPLAIN_STREAM	表示单个运算符与数据对象之间的输入和输出数据流。在 EXPLAIN_OBJECT 表中表示数据对象本身。在 EXPLAIN_OPERATOR 表中表示数据流中涉及的运算符。
EXPLAIN_DIAGNOSTIC	包含一个条目表示为 EXPLAIN_STATEMENT 表中的说明语句的特定实例生成的每条诊断消息。
EXPLAIN_DIAGNOSTIC_DATA	包含在 EXPLAIN_DIAGNOSTIC 表中记录的特定诊断消息的消息标记。这些消息标记提供特定于生成该消息的 SQL 语句的执行的更多信息。
ADVISE_WORKLOAD	允许用户向数据库描述其工作负载。表中的每一行代表工作负载中的一个 SQL或 XQuery 语句，并由相关的频率描述。db2advis 工具使用此表来收集并存储工作负载信息。
ADVISE_INSTANCE	包含有关执行 db2advis 的信息，包括开始时间。每次执行 db2advis 都会产生相应的一行。
ADVISE_INDEX	存储有关建议的索引的信息。该表可由查询编译器、db2advis 实用程序或用户填充。可以两种方式使用此表: • 获取建议的索引。 • 根据有关建议的索引的输入对索引求值。
ADVISE_MQT	包含 CREATE DDL、用来定义建议的每个 MQT 的查询、每个 MQT 的统计信息（例如，COLSTATS（每一列））（采用 XML 格式）以及 NUMROWS 等等，还包含采样查询以获得每个 MQT 的采样统计信息。
ADVISE_TABLE	存储使用为建议的 MQT、MDC 和数据库分区而最终提供的“设计顾问程序”建议来创建表时所使用的 DDL，这取决于指定的选项和生成的建议。
ADVISE_PARTITION	存储由 db2advis 生成并评估的虚拟数据库分区。

注: 在缺省情况下，并不创建上面所有的表。要创建它们，运行 *sqllib* 子目录的 *misc* 子目录中的 EXPLAIN.DDL 脚本。

目前，在 DB2 V9.5 中，您能够通过使用 SYSPROC.SYSINSTALLOBJECTS 过程来创建、废弃和验证说明表。此过程允许在特定模式和表空间中创建说明表。可以在 EXPLAIN.DLL 文件中找到相应的示例。

说明表可能对多个用户是公共的。但是，可以为一个用户定义说明表，然后可以使用相同名称来指向已定义的表来为每个附加用户定义别名。或者，可在 SYSTOOLS 模式下定义说明表。如果在动态 SQL 或 XQuery 语句的用户会话标识或静态 SQL 或 XQuery 语句的授权标识下找不到其他说明表或别名，那么说明工具将缺省为 SYSTOOLS 模式。共享公用说明表的每个用户必须对那些表有插入许可权。公用说明表的读许可权通常也应限于分析说明信息的用户。

数据对象的说明信息

单个访问方案可使用一个或多个数据对象来满足 SQL 或 XQuery 语句。

对象统计信息：说明工具记录有关该对象的信息，如下列信息:

- 创建时间
- 上次收集该对象的统计信息的时间
- 有关是否将对象（仅表或索引对象）中的数据排序的指示
- 对象（仅表或索引对象）中的列数
- 对象（仅表或索引对象）中的估计行数
- 对象占用缓冲池的页数
- 对于此对象存储所在的指定表空间，所发生的单个随机 I/O 的估计总开销（以毫秒计）
- 从指定的表空间读取 4K 页的估计传送速率（以毫秒计）
- 预取大小和扩展数据块大小（以 4K 页计）
- 使用索引集群数据的程度
- 此对象的索引使用的叶子页数和树中的级别数
- 此对象的索引中相异完整键值的数目
- 表中溢出记录的总数

数据运算符的说明信息

单个访问方案可对数据执行几个运算，来满足 SQL 或 XQuery 语句并将结果返回给您。查询编译器确定必须的操作，如表扫描、索引扫描、嵌套的循环连接或 group-by 运算符。

除显示访问方案中使用的运算符和有关每个运算符的信息外，说明信息还显示访问方案的累计效果。

估计成本信息：可以显示以下估计的运算符累计成本。这些成本是所选访问方案的成本，包括捕获其信息的运算符的成本。

- 总计成本（以 timeron 计）
- 页 I/O 数
- CPU 指令数
- 获取第一行的成本（以 timeron 计），包括必需的任何初始开销
- 通信成本（以帧计）

timeron 是发明的相对计量单位。Timeron 由优化器基于在使用数据库时会更改的内部值（如统计信息）确定。因此，不能保证对 SQL 或 XQuery 语句的 timeron 评测在每次确定以 timeron 计的估计成本时都是相同的。

运算符属性：说明工具记录下列信息来描述每个运算符的属性:

- 已访问的表的集合
- 已访问的列的集合
- 对数据排序所依据的列，如果优化器确定后续运算符可使用此排序
- 已应用的谓词的集合
- 将返回的估计行数（基数）

实例的说明信息

说明实例信息存储在 EXPLAIN_INSTANCE 表中。有关实例中的每个查询语句的其他特定信息存储在 EXPLAIN_STATEMENT 表中。

说明实例标识：下列各项提供的信息可帮助您唯一标识每个说明实例，并将查询语句的信息与该实施的给定调用相关:

- 请求说明信息的用户
- 说明请求开始的时间
- 包含说明的查询语句的程序包的名称
- 包含说明的查询语句的程序包的 SQL 模式
- 包含语句的程序包的版本
- 是否收集了快照信息

环境设置：捕获有关查询编译器优化查询所在的数据库管理器环境的信息。环境信息包括下列各项:

- DB2 级别的版本号和发行版号
- 编译该查询的并行度

  CURRENT DEGREE 专用寄存器、DEGREE 绑定选项、SET RUNTIME DEGREE API 和 *dft_degree* 配置参数确定编译特定查询的并行度。
- 该查询语句是动态的还是静态的
- 用于编译该查询的查询优化级别
- 当编译查询时指定的游标行分块的类型
- 该查询运行的隔离级别
- 编译该查询时各种配置参数的值。当获取说明快照时，记录下列参数:
  - 排序堆大小（*sortheap*）
  - 活动应用程序平均数（*avg_appls*）
  - 数据库堆（*dbheap*）
  - 锁定列表的最大存储器（*locklist*）
  - 升级前锁定列表的最大百分比（*maxlocks*）
  - CPU 速度（*cpuspeed*）
  - 通信带宽（*comm_bandwidth*）

语句标识：对每个说明实例可能已说明多个查询语句。除了唯一标识说明实例的信息外，下列信息帮助标志 识个别的查询语句:

- 语句的类型：SELECT、DELETE、INSERT、UPDATE、定位的 DELETE、定位的 UPDATE 和 SET INTEGRITY
- 程序包中发出查询语句的语句号和节号，它们记录在 SYSCAT.STATEMENTS 目录视图中

EXPLAIN_STATEMENT 表中的 QUERYTAG 和 QUERYNO 字段包含设置为说明过程的一部分的标识。对于在 CLP 或 CLI 会话期间提交的动态说明查询语句，当 EXPLAIN MODE 或 EXPLAIN SNAPSHOT 是活动时，QUERYTAG 设置为“CLP”或“CLI”。在此情况下，对于每个语句，QUERYNO 缺省为按 1 或更大的值递增的一个数。对于所有其他动态说明查询语句（不是来自 CLP、CLI 或不使用 EXPLAIN 查询语句），QUERYTAG 将设置为空白，且 QUERYNO 始终是“1”。

成本估计：对于每个说明语句，优化器记录执行所选访问方案的相关成本估计值。此成本按发明的称为 *timeron* 的相对计量单位来陈述。由于下列原因，不提供耗用时间估计值:

- 查询优化器不估计耗用时间，而只估计资源消耗。
- 优化器并非将可影响耗用时间的所有因素都考虑在模型内。它忽略不影响访问方案效率的那些因素。几个运行时因素影响耗用时间，包括：系统工作负载、资源争用量、并行处理和 I/O 的量、将行返回至用户的成本以及客户机和服务器之间的通信时间。

语句文本：对于每个说明的语句，记录查询语句文本的两个版本。一个版本是查询编译器从应用程序接收的代码。另一个版本是从查询的内部编译器表示反向转换的。尽管此转换看上去类似于其他查询语句，但它既不必遵循正确的查询语言语法，也不必从整体上反映内部表示的实际内容。提供此转换，只是为了让您了解 SQL 和 XQuery 优化器选择访问方案所在的上下文。要了解 SQL 和 XQuery 编译器如何重新编写查询以获取更好的优化，将用户编写的语句文本与查询语句的内部表示做比较。重新编写的语句也显示该环境中影响您的语句的其他元素，如触发器和约束。此“优化的”文本使用的一些关键字包括:

$C*n*
派生的列的名称，其中，*n* 表示整数值。

$CONSTRAINT$
一个标记，用于指示在编译期间添加至原始查询语句的约束的名称。它与 $WITH_CONTEXT$ 前缀一起使用。

$DERIVED.T*n*
派生的表的名称，其中，*n* 表示整数值。

$INTERNAL_FUNC$
此标记指示 SQL 和 XQuery 编译器使用的函数是否存在，该函数用于说明的查询，但不可通用。

$INTERNAL_PRED$
此标记指示在编译说明的查询期间，SQL 和 XQuery 编译器添加的谓词是否存在，但不可通用。该编译器使用一个内部谓词来满足由于触发器和约束而添加至原始查询语句的其他上下文。

$INTERNAL_XPATH$
显示一个内部表函数，该函数将单个带注释的输入 XPath 模式作为参数，并返回有一列或多列与匹配该模式的表。

RID 一个标记，用于标识特定行的“行标识”（RID）列。

$TRIGGER$
一个标记，用于指示在编译期间添加至原始查询语句的触发器的名称。它与 $WITH_CONTEXT$ 前缀一起使用。

$WITH_CONTEXT$(...)
当已将其他触发器或约束添加至原始查询语句中后，此前缀出现在文本开始处。此前缀之后显示影响查询语句的编译和解析的任何触发器或约束的名称列表。

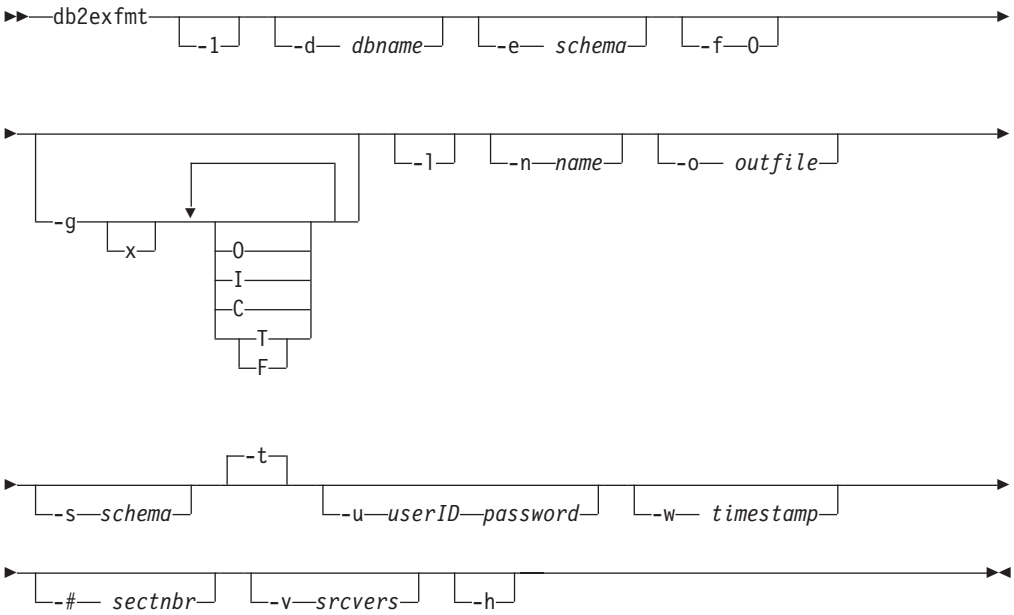
db2exfmt - 说明表格式化

可以使用 db2exfmt 工具来格式化说明表的内容。此工具位于实例 sqllib 目录的 misc 子目录中。如果说明快照可用，此工具将使用该快照中的统计信息。

权限

要使用该工具，需要有要格式化的说明表的读访问权。

命令语法



命令参数

db2exfmt
如果未指定任何选项，那么该命令将进入交互方式，并且系统会提示您进入。

-1 使用缺省值 -e % -n % -s % -v % -w -1 -# 0

如果未提供说明模式，那么环境变量 $USER 或 $USERNAME 的内容将用作缺省值。如果找不到此变量，那么会提示用户提供说明模式。

-d *dbname*
包含程序包的数据库的名称。

-e *schema*
说明表 SQL 模式。

-f 格式化标志。在此发行版中，唯一支持的值是 O（运算符总结）。

-g 图形方案。

x 关闭选项（缺省值是打开选项）。

如果仅指定 -g，那么生成一个图，后跟所有表的格式化信息。另外，可以指定下列有效值的任意组合:

O 只生成图形。不格式化表的内容。

T 在图形中每个运算符下包括总成本。

F 在图形中包括第一个元组成本。

I 在图形中每个运算符下包括 I/O 成本。

C 在图形中包括每个运算符的期望输出基数（元组数）。

允许使用这些选项的任意组合，但 F 和 T 组合除外，它们互斥。

-l 当处理程序包名时区分大小写。

-n *name*
“说明”请求的源名（SOURCE_NAME）。

-s *schema*
“说明”请求的源 SQL 模式或限定符（SOURCE_SCHEMA）。

-o *outfile*
输出文件名。

-t 将输出定向到终端。

-u *userID password*
当与一个数据库连接时，使用提供的用户标识和密码。

用户标识和密码都必须符合命名约定，且是数据库可识别的。

-w *timestamp*
说明时间戳记。指定 -1 来获取最新的说明请求。

-# *sectnbr*
源中的节号。要请求所有的节，指定零。

-v *srcvers*
说明请求源的源版本（缺省值 %）

-h 显示帮助信息。当指定了此选项时，其他所有的选项都会被忽略，且只显示帮助信息。

使用说明

除 -h 和 -l 选项外，将提示您输入任何未提供或未完整指定的参数值。

如果未提供说明表 SQL 模式，那么环境变量 USER 的值将用作缺省值。如果找不到此变量，那么会提示用户提供说明表的 SQL 模式。

源名、源 SQL 模式和说明时间戳记可用 LIKE 谓词格式提供，它允许将百分号（%）和下划线（_）作为模式匹配字符，以通过一个调用选择多个源。对于最新的说明语句，可将说明时间指定为 -1。

如果指定 -o 而不带文件名，且未指定 -t，那么会提示用户提供文件名（缺省文件名为 db2exfmt.out）。如果既未指定 -o 也未指定 -t，那么会提示用户提供文件名（缺省选项是终端输出）。如果同时指定 -o 和 -t，那么将输出定向到终端。

如果说明快照可用，db2exfmt 命令将显示该快照中的统计信息。否则，db2exfmt 不仅显示存储在 EXPLAIN_OBJECT 表中的统计信息，还显示直接从系统目录中检索到的一些统计信息。

下面是说明快照示例。

```
db2 explain plan with snapshot for query
db2exfmt
```

或者，

```
db2 set current explain mode yes
db2 set current explain snapshot yes
run the query
db2exfmt
```

优化查询访问方案

优化级别

当编译 SQL 或 XQuery 查询时，可以指定优化级别以确定优化器如何选择该查询的最有效访问方案。可使用查询编译中考虑的优化策略的编号和类型来区分优化级别。尽管可以单独指定优化技术以提高查询的运行时性能，但是，指定的优化技术越多，查询编译将需要越多的时间和系统资源。

可以在编译 SQL 或 XQuery 查询时指定下列其中一个优化器级别:

0 - 此级别指导优化器使用最少的优化来生成访问方案。此优化级别具有下列特征:

- 优化器不考虑任何非均匀分布统计信息。
- 仅应用基本的查询重写规则。
- 发生贪婪联合枚举。
- 仅允许使用嵌套循环连接及索引扫描访问方法。
- 在生成的访问方法中不使用列表预取。
- 不考虑星型连接策略。

此级别应该只用于需要最低的查询编译开销的情况。查询优化级别 0 适用于以下应用程序: 完全由访问索引结构良好的表的非常简单的动态 SQL 或 XQuery 语句组成。

1 - 此优化级别具有下列特征:

- 优化器不考虑任何非均匀分布统计信息。
- 只应用查询重写规则的一个子集。
- 发生贪婪联合枚举。
- 在生成的访问方法中不使用列表预取。

除了“合并扫描”连接及表扫描也可用以外，优化级别 1 类似于级别 0。

2 - 此级别指导优化器使用比级别 1 显著高的优化程度，而使复杂查询的编译成本显著低于级别 3 及更高级别。此优化级别具有下列特征:

- 利用了所有可用的统计信息，包括频率和分位数非均匀分布统计信息。
- 除只在极少情况下才适用的计算密集型规则外，将应用所有其他查询重写规则，包括路由对具体化查询表的查询。
- 使用了贪婪联合枚举。
- 考虑各种访问方法，包括列表预取和具体化查询表路由。
- 如果适用的话，考虑星型连接策略。

优化级别 2 除了使用“贪婪”联合枚举而不是“动态规划”以外，类似于级别 5。在所有使用“贪婪”联合枚举算法的级别中，此级别具有最高的优化程度，与级别 3 及更高级别相比，它对复杂查询的替代方案考虑较少，因而消耗的编译时间也少。建议将级别 2 用于决策支持或联机分析处理（OLAP）环境中非常复杂的查询。在这种环境下，特定查询很少完全重复，因此查询访问方案不太可能在高速缓存中停留到出现下一个查询为止。

3 - 此级别请求中等优化。此级别与 DB2 MVS/ESA™ 版、OS/390 或 z/OS 版的查询优化特征基本匹配。此优化级别具有下列特征:

- 使用非均匀分布统计信息（如果可用的话），该统计信息跟踪频繁出现的值。
- 应用大部分查询重写规则，包括子查询至连接的变换。
- 动态规划连接枚举，如下所示:
  - 组合内部表的有限使用
  - 涉及查找表的星型模式的笛卡尔乘积的有限使用
- 考虑各种访问方法，包括列表预取、索引 AND 运算和星型连接。

此级别适用于大量应用程序。此级别改进具有 4 个或更多连接的查询的访问方案。但是，优化器可能无法考虑使用缺省优化级别选择的更好方案。

5 - 此级别指导优化器使用相当大量的优化来生成访问方案。此优化级别具有下列特征:

- 使用所有可用的统计信息，包括频率和分位数分布统计信息。
- 除只在极少情况下才适用的那些计算密集型规则外，将应用所有其他查询重写规则，包括路由对具体化查询表的查询。
- 动态规划连接枚举，如下所示:
  - 组合内部表的有限使用
  - 涉及查找表的星型模式的笛卡尔乘积的有限使用
- 考虑各种访问方法，包括列表预取、索引 AND 运算和具体化查询表路由。

当优化器检测到不能保证用于复杂动态 SQL 或 XQuery 查询的其他资源和处理时间时，将减少优化。减少的范围或大小取决于机器大小和谓词数目。

当查询优化器减少查询优化量时，它继续应用正常时应用的所有查询重写规则。但是，它的确使用了贪婪联合枚举法并减少了考虑的访问方案的组合数。

对于由事务和复杂查询组成的混合环境，查询优化级别 5 是一个很好的选择。此优化级别设计成可以用高效的方式应用最有价值的查询变换和其他查询优化技术。

7 - 此级别指导优化器使用相当大量的优化来生成访问方案。级别 7 除了不减少用于复杂动态 SQL 或 XQuery 查询的查询优化量以外，它与查询优化级别 5 是相同的。

9 - 此级别指导优化器使用所有可用的优化技术。这些主要功能包括:

- 所有可用的统计信息
- 所有查询重写规则
- 联合枚举的所有可能性，包括笛卡尔乘积和任意多种组合的内部结构
- 所有访问方法

此级别可以大大扩展由优化器考虑的可能的访问方案数量。对于使用大表的很复杂且运行时间很长的查询，可以使用此级别来确定更全面优化是否将生成更好的访问方案。使用“说明”和性能测量来验证是否实际上已找到更好的方案。

选择优化级别

设置优化级别可以提供有关显式指定优化技术的某些优点，特别是由于下列原因:

- 管理非常小型的数据库或非常简单的动态查询
- 适应数据库服务器上编译时的内存限制
- 减少查询编译时间，如 PREPARE。

通过使用优化级别 5（缺省查询优化级别），大多数语句都可以使用合理数量的资源充分优化。以给定的优化级别，查询编译时间和资源消耗主要受查询的复杂性，特别是连接和子查询的数量影响。然而，编译时间和资源的使用也受所执行的优化数量影响。

查询优化级别 1、2、3、5 和 7 都比较通用。仅当您需要进一步减少查询编译时间且您了解 SQL 和 XQuery 语句相当简单时，才考虑级别 0。

技巧: 要分析长时间运行的查询，可使用 db2batch 运行该查询以确定编译和执行各花费了多长时间。如果编译需要更长时间，那么降低优化级别。如果执行需要更长时间，那么考虑更高的优化级别。

当选择优化级别时，考虑下列一般准则:

- 通过使用缺省查询优化级别（级别 5）开始。
- 要使用不是缺省值的级别，首先尝试级别 1、2 或 3。级别 0、1 和 2 使用“贪婪”连接枚举算法。
- 如果您有许多表，这些表的同一列有许多连接谓词，并且您又关心编译时间，那么使用优化级别 1 或 2。
- 对运行时间短（不到一秒）的查询，使用低优化级别（0 或 1）。这些查询趋向于具有下列特征:
  - 访问单个表或仅访问少量表
  - 访存一行或仅访存少量行

– 使用标准的唯一索引。

这种查询的典型示例是联机事务处理（OLTP）事务。

- 对于运行时间较长（大于 30 秒）的查询，使用高优化级别（3、5 或 7）。
- 级别 3 和更高的级别使用“动态编程”连接枚举算法。与级别 0、1 和 2 相比，此算法考虑更多替代方案，并且可能使编译时间也显著增加，特别是随着表的数量增加，更是如此。
- 仅当具有查询的特定异常的优化需求时，使用优化级别 9。

复杂的查询可能需要不同程度的优化，以便选择最佳访问方案。对具有下列特征的查询考虑使用更高的优化级别:

- 访问大型表
- 大量谓词
- 很多子查询
- 很多连接
- 很多集合运算符，例如 UNION 和 INTERSECT
- 很多限定行
- GROUP BY 和 HAVING 操作
- 嵌套表表达式
- 大量视图。

复杂查询的典型例子是对完全规范化数据库的决策支持查询或月结报告查询，对于这些复杂查询至少应该使用缺省查询优化级别。

对于由查询生成器产生的 SQL 和 XQuery 使用较高的查询优化级别。许多查询生成器创建运行效率不高的查询。编写得很差的查询，包括那些由查询生成器产生的查询，需要另外优化以选择一个良好的访问方案。使用查询优化级别 2 和更高的级别可以改进这样的 SQL 和 XQuery 查询。

注: 在联合数据库查询中，优化级别不适用于远程优化器。

设置优化级别

当指定优化级别时，考虑查询是使用动态 SQL 和 XQuery 语句还是静态 SQL 和 XQuery 语句，以及是否重复执行同一动态查询。对于静态 SQL 和 XQuery 语句，只耗费一次查询编译时间和资源，而产生的方案可以使用许多次。一般来说，静态 SQL 和 XQuery 语句应该总是使用缺省查询优化级别。因为动态语句是在运行时绑定和执行的，所以，考虑对动态语句进行另外的优化开销是否改善总体性能。但是，如果重复执行相同的动态 SQL 或 XQuery 语句，那么高速缓存所选访问方案。这种语句可以使用与静态 SQL 或 XQuery 语句相同的优化级别。

如果您认为一个查询可以从附加优化获益，但不能肯定，或者您关心编译时间和资源使用情况，那么您可以执行一些基准程序测试。

要指定查询优化级别，遵循下列步骤:

1. 按以下方式非正式地或用正式测试分析性能因素:

- 对于**动态**查询语句，测试应比较该语句的平均运行时间。使用下列公式来估计平均运行时间:

$$\frac{\text{编译时间 + 所有重复的执行时间的总和}}{\text{重复次数}}$$

  在此公式中，重复次数表示每次编译查询语句时，期望查询语句执行的次数。

  注: 在初始编译之后，当环境更改要求动态 SQL 和 XQuery 语句时，将重新编译这些动态 SQL 和 XQuery 语句。如果将查询语句高速缓存之后环境不更改，那么不需要再次编译，因为后来的 PREPARE 语句将复用高速缓存的语句。

- 对于**静态** SQL 和 XQuery 语句，比较语句运行时间。

  尽管您可能对静态 SQL 和 XQuery 语句的编译时间也感兴趣，但该语句的总计编译与运行时间在任何有意义的上下文中都很难估计。比较总时间不能识别这样一个事实，即每次将静态查询语句绑定时，它可以运行多次，而在运行时，一般不将它绑定。

2. 如下所示指定优化级别:
   - **动态** SQL 和 XQuery 语句使用 CURRENT QUERY OPTIMIZATION 专用寄存器指定的优化级别，该寄存器是用 SQL 语句 SET 设置的。例如，下列语句将优化级别设置为 1:

     ```
     SET CURRENT QUERY OPTIMIZATION = 1
     ```

     要确保动态 SQL 或 XQuery 语句总是使用同一个优化级别，可以将 SET 语句包含在应用程序中。

     如果尚未设置 CURRENT QUERY OPTIMIZATION 寄存器，将使用缺省查询优化级别绑定动态语句。动态和静态查询的缺省值由数据库配置参数 *dft_queryopt* 的值确定。级别 5 是此参数的缺省值。绑定选项和专用寄存器的缺省值也是从 *dft_queryopt* 数据库配置参数读取的。

   - **静态** SQL 和 XQuery 语句使用 PREP 和 BIND 命令上指定的优化级别。SYSCAT.PACKAGES 目录表中的 QUERYOPT 列记录用于绑定程序包的优化级别。如果以隐式方式或使用 REBIND PACKAGE 命令重新绑定程序包，将把同一个优化级别用于静态查询语句。要更改这种静态 SQL 和 XQuery 语句的优化级别，可使用 BIND 命令。如果未指定优化级别，那么 DB2 使用 *dft_queryopt* 数据库配置参数指定的缺省优化级别。

优化器概要文件和准则概述

DB2 优化器是业界最先进的、基于成本的优化器之一。但是，在极少情况下，此优化器可能会选择不是最佳的执行计划。当 DBA 熟悉数据库时，您可以使用诸如 db2advis、RUNSTATS 和 db2expln 之类的工具和优化级别设置来帮助您调整优化器，以获得更好的数据库性能。如果在使用了所有调整选项后未获得期望的结果，那么可以为 DB2 优化器提供显式优化准则。

优化概要文件是包含一个或多个 SQL 语句的优化准则的 XML 文档。通过使用 SQL 文本和明确标识一个 SQL 语句所需的其他相关信息建立每个 SQL 语句与其关联的优化准则之间的通信。第 330 页的图 31 演示了如何使用优化概要文件将优化准则传递给 DB2 优化器。

```
<?xml version="1.0" encoding="UTF-8">

<OPTPROFILE VERSION="9.1.0.0">
<STMTPROFILE ID="Guidelines for TPCD Q9">
      <STMTKEY SCHEMA="TPCD">
    SELECT S.S_NAME, S.S_ADDRESS, S.S_PHONE, S.S_COMMENT
    FROM PARTS P, SUPPLIERS S, PARTSUPP PS
    WHERE P_PARTKEY = PS.PS_PARTKEY AND S.S_SUPPKEY = PS.PS_SUPPKEY AND P.P_SIZE = 39 AND P.P_TYPE = 'BRASS'
AND
          S.S_NATION = 'MOROCCO' AND S.S_NATION IN ('MOROCCO', 'SPAIN') AND
          PS.PS_SUPPLYCOST = (SELECT MIN(PS1.PS_SUPPLYCOST)
                              FROM   PARTSUPP PS1, SUPPLIERS S1
                              WHERE P.P_PARTKEY = PS1.PS_PARTKEY AND S1.S_SUPPKEY = PS1.PS_SUPPKEY AND
                                                  S1.S_NATION = S.S_NATION))
     </STMTKEY>    <OPTGUIDELINES> <IXSCAN TABLE="S" INDEX="I_SUPPKEY"/> </OPTGUIDELINES>
  </STMTPROFILE>   </OPTPROFILE>
```

图 31. 使用优化概要文件来传递准则

每个 STMTPROFILE 元素都为一个应用程序语句提供一组优化准则。目标语句由 STMTKEY 子元素标识。然后，为优化概要文件提供模式限定名并将此概要文件插入到数据库中。通过在 BIND 或 PREPARE 命令中指定优化概要文件的名称，即可使用此优化概要文件来优化语句。

优化概要文件允许为优化器提供优化准则，而不需要更改应用程序或数据库配置。您只需编写一个简单的 XML 文档，将此文档插入到数据库中，然后在 BIND 或 PREPARE 命令中提供优化概要文件的名称。优化器会自动使优化准则与相应的语句匹配。

优化准则不必是全面的，但应以期望的执行计划为目标。DB2 优化器仍使用现有的基于成本的方法来选择其他可能的访问方案。以特定表引用为目标的优化准则不能覆盖一般优化参数设置。因此，指定表 A 与表 B 之间的合并连接的优化准则在优化级别 0 无效。

优化器将忽略无效或不适用的优化准则。如果有任何优化准则被忽略，那么将会生成执行计划并返回原因码为 13 的 SQL0437W 警告。然后，可以使用 EXPLAIN 语句来获取有关优化准则处理的详细诊断信息。

优化准则

优化准则和处理的类型概述: DB2 优化器优化语句的过程分为两个阶段。已优化的语句由*查询重写优化阶段*确定，此阶段将原始语句变换为语义上等价但更容易在方案优化阶段优化的语句。*方案优化阶段*通过枚举许多备用方案并选择使执行成本估计最低的备用方案来确定*已优化的语句*的最佳访问方法、连接方法和连接顺序。

在这两个优化阶段期间考虑的查询变换、访问方法、连接方法、连接顺序和其他优化备用方法由各种 DB2 参数（例如，CURRENT QUERY OPTIMIZATION 专用寄存器、REOPT 绑定选项和 *DB2_REDUCED_OPTIMIZATION* 注册表变量）控制。优化语句期间考虑的一组优化备用方法称为搜索空间。

支持下列类型的语句优化准则:

- 一般优化准则
- 查询重写准则
- 方案优化准则

应按特定顺序应用优化准则。首先应用一般优化准则，因为它们可以影响搜索空间。然后应用查询重写准则，因为它们可以影响方案优化阶段优化的语句。最后应用方案优化准则。

*一般优化准则*可用于影响一般优化参数的设置。*查询重写准则*可用于影响查询重写优化阶段考虑的变换。*方案优化准则*可用于影响在方案优化阶段考虑的访问方法、连接方法和连接顺序。

一般优化准则: *一般优化准则*可用于设置一般优化参数。其中每个准则都有语句级别范围。

查询重写优化准则: *查询重写准则*可用于影响查询重写优化阶段考虑的变换。查询重写优化阶段将原始语句变换为语义上等价的*已优化的语句*。然后，在方案优化阶段确定已优化的语句的最佳执行计划。因此，查询重写优化准则可以影响方案优化准则的适用性。

每个查询重写优化准则都与优化器的其中一条查询变换规则相对应。查询重写优化准则可以影响下列查询变换规则:

- IN-LIST-to-join
- Subquery-to-join
- NOT-EXISTS-subquery-to-antijoin
- NOT-IN-subquery-to-antijoin

查询重写优化准则并非总是适用。一次强制执行一条查询重写规则。因此，在实施后面的规则之前强制执行的一些查询重写规则可以影响与该规则相关联的查询重写优化准则。环境配置有时可以影响一些重写规则的行为，这将影响特定规则的查询重写优化准则的适用性。要每次都获得相同的结果，在强制执行查询重写规则之前，这些规则必须满足一些条件。如果查询重写组件尝试将规则应用于查询时未满足与该规则关联的条件，将忽略该规则的查询重写优化准则。如果查询重写优化准则不适用并且此准则是启用准则，那么将返回原因码为 13 的 SQL0437W 错误消息。如果查询重写优化准则不适用并且此准则是禁用准则，那么不会返回错误消息。在这种情况下，不会应用此查询重写规则，因为此规则被视为已禁用。

可以将查询重写优化准则分为两种类别: 语句级别和谓词级别。所有查询重写优化准则都支持语句级别类别。仅 INLIST2JOIN 支持谓词级别类别。语句级别查询重写优化准则适用于整个查询。谓词级别查询重写优化准则仅适用于特定谓词。如果同时指定语句级别和谓词级别查询重写优化准则，那么谓词级别准则将覆盖特定谓词的语句级别准则。

每个查询重写优化准则都由优化准则模式中的相应*重写请求元素*表示。

方案优化准则: 在基于成本的优化阶段应用方案优化准则，此阶段中将确定语句的访问方法、连接方法、连接顺序和其他执行计划详细信息。方案优化准则不需要指定执行计划的所有方面。未指定的执行计划方面由优化器以基于成本的方式确定。

有两种类别的方案优化准则:

- *accessRequest* – 访问请求指定用于满足语句中的表引用所需的访问方法。
- *joinRequest* – 连接请求指定用于执行连接操作所需的方法和顺序。连接请求由其他访问请求或连接请求组成。

访问请求优化准则与优化器的数据访问方法（如表扫描、索引扫描和列表预取）相对应。连接请求准则与优化器的连接方法（如嵌套循环连接、散列连接和合并连接）相对应。*Performance Guide* 中描述了这些方法。

形成优化准则中的表引用: 本文档Zs•3中使用的术语*表引用*指的是 SQL 语句或视图定义中的任何表、视图、表表达式或别名。 优化准则可以使用原始语句中表引用的显示名或使用与*已优化的语句*中的表引用关联的唯一相关名来标识表引用。扩展名称（即显示名的序列）可用来帮助唯一地标识视图中嵌入的表引用。如果优化准则所标识的显示名或扩展名在整个语句的上下文中不是唯一的，那么认为这些优化准则具有二义性而不会应用。此外，如果多个优化准则标识相同的表引用，那么认为标识表引用的所有优化准则相冲突，将不应用这些优化准则。下列各节详细地描述了优化准则技术的这些特定方面。由于查询过程中可能出现变化，无法保证显示名或扩展名称在优化过程中仍然存在，在这种情况下，将忽略所有以表引用为目标的准则。

使用原始语句的显示名来标识表引用

表引用是使用其显示名标识的。指定显示名的方式与在 SQL 语句中限定表的方式相同。用于指定 SQL 标识的规则适用于 TABLE 属性值。

将优化准则的 TABLE 属性值与语句的每个显示名进行比较。在此发行版的 DB2 中，只允许单个匹配项。如果 TABLE 属性值是模式限定的，那么它与任何等价的显示限定表名匹配。如果 TABLE 属性值未限定，那么它与任何等价的相关名或显示表名匹配（因此，认为 TABLE 属性值由语句的有效缺省模式隐式限定）。图 32中的示例语句说明了这些概念。假定使用缺省模式“Tpcd”优化语句。

```
SELECT S_NAME, S_ADDRESS, S_PHONE, S_COMMENT
   FROM PARTS, SUPPLIERS, PARTSUPP PS
   WHERE P_PARTKEY = PS.PS_PARTKEY AND S.S_SUPPKEY = PS.PS_SUPPKEY AND
         P_SIZE = 39 AND P_TYPE = 'BRASS';
```

图 32. 使用优化准则的 TABLE 属性值来与语句的每个显示名进行比较

正确标识语句中的表引用的 TABLE 属性值包括“″Tpcd″.PARTS”、“PARTS”和“Parts”（因为标识未定界，所以它将转换为大写）。无法标识语句中的表引用的 TABLE 属性值包括“″Tpcd2″.SUPPLIERS”、“PARTSUPP”（不是显示名）和“Tpcd.PARTS”（必须定界标识 Tpcd，否则它将转换为大写）。

可以使用显示名来以原始语句、视图、SQL 函数或触发器中的任何表引用为目标。

使用原始语句的显示名来标识视图中的表引用

优化准则可以使用扩展语法来标识视图中嵌入的表引用。

可以使用扩展语法来以原始语句、SQL 函数或触发器中的任何表引用为目标。

使用已优化的语句中的相关名来标识表引用

优化准则还可以使用与*已优化的语句*中的表引用关联的唯一相关名来标识表引用。已优化的语句是语义上与原始语句等价的语句，它由查询重写优化阶段确定。可以从 EXPLAIN 表中检索已优化的语句。优化准则的 TABID 属性用于标识已优化的语句中的表引用。

如果单个优化准则同时指定 TABLE 和 TABID 属性，那么它们必须标识相同的表引用或者将忽略此优化准则。

注: 当前无法保证在升级到新发行版的 DB2 时已优化的语句中的相关名将保持稳定状态。

模糊表引用

如果某个优化准则与多个显示名或扩展名称匹配，那么认为此优化准则无效而不会应用。

要消除歧义，可以重写视图以使用唯一相关名，也可以使用 TABID 属性。

注: 由 TABID 字段标识的表引用决不会是模糊的，因为已优化的语句中的所有相关名都是唯一的。

相冲突的优化准则

多个优化准则不能标识相同的表引用。

当两个或多个准则引用同一个表时，仅应用第一个准则；将忽略所有其他准则并发出错误消息。

用于在谓词级别指定启用 OPTION 的多个查询重写 INLIST2JOIN 重写请求元素存在限制。在谓词级别，只能在一个查询中指定一个这种 INLIST2JOIN 重写请求元素。

验证是否使用了优化准则:

优化器千方百计遵循在优化概要文件中或通过 SQL 嵌入式优化准则（也称为语句概要文件）指定的优化准则；但是，优化器可以拒绝无效或不适用的准则。

要使用 EXPLAIN 命令，EXPLAIN 表必须存在。用于创建 EXPLAIN 表的 DDL 是 EXPLAIN.DDL，它位于 sqllib 目录的 misc 子目录中。

要验证是否使用了有效的优化准则:

1. 对语句运行 EXPLAIN 命令。 如果优化准则对于使用优化概要文件或 SQL 注释的语句有效，那么优化概要文件名将作为 RETURN 运算符参数出现在 EXPLAIN_ARGUMENTS 表中。此外，如果优化准则包含了与当前语句匹配的 SQL 嵌入式优化准则或语句概要文件，那么语句概要文件的名称将作为 RETURN 运算符参数出现。这两个新参数值的类型为 OPT_PROF 和 STMTPROF。
2. 检查说明语句的结果。 可以修改下列针对说明表的查询以返回 EXPLAIN_REQUESTER、EXPLAIN_TIME、SOURCE_NAME、 SOURCE_VERSION 和 QUERYNO 的特定组合的优化概要文件名和语句概要文件名:

```
SELECT VARCHAR(B.ARGUMENT_TYPE,   9) as TYPE,
            VARCHAR(B.ARGUMENT_VALUE, 24) as VALUE

   FROM   EXPLAIN_STATEMENT A, EXPLAIN_ARGUMENT B

   WHERE  A.EXPLAIN_REQUESTER = 'SIMMEN'
       AND  A.EXPLAIN_TIME      = '2003-09-08-16.01.04.108161'
       AND  A.SOURCE_NAME       = 'SQLC2E03'
       AND  A.SOURCE_VERSION    = ''
       AND  A.QUERYNO           = 1

       AND  A.EXPLAIN_REQUESTER = B.EXPLAIN_REQUESTER
       AND  A.EXPLAIN_TIME      = B.EXPLAIN_TIME
       AND  A.SOURCE_NAME       = B.SOURCE_NAME
       AND  A.SOURCE_SCHEMA     = B.SOURCE_SCHEMA
       AND  A.SOURCE_VERSION    = B.SOURCE_VERSION
       AND  A.EXPLAIN_LEVEL     = B.EXPLAIN_LEVEL
       AND  A.STMTNO            = B.STMTNO
       AND  A.SECTNO            = B.SECTNO

       AND  A.EXPLAIN_LEVEL     = 'P'

       AND  (B.ARGUMENT_TYPE  = 'OPT_PROF' OR ARGUMENT_TYPE = 'STMTPROF')
       AND  B.OPERATOR_ID = 1
```

图 33. 使用查询返回说明语句的结果

如果优化准则处于活动状态并且说明语句与优化准则的 STMTKEY 元素中包含的语句相匹配，那么与图 34 中的查询类似的查询将在图 33 中生产输出。STMTPROF 参数的值与 STMTPROFILE 元素中 ID 属性的值相同。

```
类型      值
--------- -------------------------
    OPT_PROF  NEWTON.PROFILE1
    STMTPROF  TPCD Q9 的准则
```

图 34. 用于返回说明语句的结果的查询输出

优化概要文件

优化概要文件的结构: 本节向您介绍优化概要文件的内容。给定 DB2 发行版的有效优化概要文件内容由称为当前优化概要文件模式（COPS）的 XML 模式描述。

当前发行版的 DB2 的 COPS 列示在第 340 页的『当前优化概要文件模式』中。

优化概要文件可以包含全局准则，当概要文件有效时，此准则适用于所有 DML 语句。优化概要文件还可以包含特定准则，每个特定准则适用于程序包中的单个 DML 语句。例如:

- 可以编写一个全局优化准则，它指定对于在当前优化概要文件处于活动状态时遇到的每条语句，请求优化器引用具体化查询表 Test.SumSales 和 Test.AvgSales。
- 可以编写一个语句优化准则，它指定每当优化器遇到目标语句时，就请求使用 I_SUPPKEY 索引来访问 SUPPLIERS 表。

因此，优化概要文件包含两个元素，它们定义可用于指定两种类型的准则的主要部分: 一个全局 *OPTGUIDELINES* 元素和任意数目的 *STMTPROFILE* 元素。优化概要文件还必须包含 *OPTPROFILE* 元素，它定义包含元数据和处理伪指令的部分。

图 35 显示了 DB2 版本 9.1 的有效优化概要文件的示例。此优化概要文件包含一个全局优化准则部分和一个语句概要文件部分。

```
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.1.0.0">

  <!--
              全局优化准则部分。
              可选，但最多只能有一个全局优化准则。
  -->
     <OPTGUIDELINES>        <MQT NAME="Test.AvgSales"/>
        <MQT NAME="Test.SumSales"/>
     </OPTGUIDELINES>
  <!--
              语句概要文件部分。
              可以没有此部分，也可以有多个此部分。
  -->
  <STMTPROFILE ID="Guidelines for TPCD Q9">
        <STMTKEY SCHEMA="TPCD">
      <![CDATA[SELECT S.S_NAME, S.S_ADDRESS, S.S_PHONE,
S.S_COMMENT FROM PARTS P, SUPPLIERS S, PARTSUPP PS
WHERE P_PARTKEY = PS.PS_PARTKEY AND S.S_SUPPKEY = PS.PS_SUPPKEY AND P.P_SIZE = 39
AND P.P_TYPE = 'BRASS' AND S.S_NATION = 'MOROCCO' AND S.S_NATION IN ('MOROCCO', 'SPAIN')
AND PS.PS_SUPPLYCOST = (SELECT MIN(PS1.PS_SUPPLYCOST) FROM PARTSUPP PS1, SUPPLIERS S1
WHERE P.P_PARTKEY = PS1.PS_PARTKEY AND S1.S_SUPPKEY = PS1.PS_SUPPKEY AND
S1.S_NATION = S.S_NATION)]]>
       </STMTKEY>       <OPTGUIDELINES>            <IXSCAN TABID="Q1" INDEX="I_SUPPKEY"/>
        </OPTGUIDELINES>    </STMTPROFILE>
   </OPTPROFILE>
```

图 35. 有效的优化概要文件

OPTPROFILE 元素

优化概要文件以 OPTPROFILE 元素开头。在图 35 中的示例中，此元素包含一个版本属性，指示优化概要文件的版本为 9.1。

全局优化准则部分

全局优化准则指定适用于所有语句（优化概要文件对于这些语句有效）的优化准则。全局优化准则部分由全局 *OPTGUIDELINES* 元素表示。在图 35 中显示的示例中，此部分包含一个全局优化准则，它指定应考虑使用具体化查询表（MQT）Test.AvgSales 和 Test.SumSales 来应答优化概要文件对其有效的任何语句。

语句概要文件部分

语句概要文件定义适用于特定语句的优化准则。优化概要文件中可以没有语句概要文件，也可以有多个语句概要文件。语句概要文件部分由 *STMTPROFILE* 元素表示。在图 35 中显示的示例中，此部分包含优化概要文件对其有效的特定语句的准则。

每个语句概要文件都包含用 *STMTKEY* 和 *OPTGUIDELINES* 元素表示的语句关键字和语句级别优化准则。

语句关键字标识语句级别优化准则适用于的语句。在图 35 中显示的示例中，*STMTKEY* 元素包含原始语句文本和明确标识目标语句所需的其他信息。

优化器使用语句关键字自动将语句概要文件与相应的语句匹配。此关系允许您为应用程序中的语句提供优化准则，而不必修改应用程序。

语句概要文件的语句级别优化准则部分由 *OPTGUIDELINES* 元素表示。在成功匹配语句概要文件中的语句关键字后，优化器将在优化语句时引用关联的语句优化准则。

语句优化准则部分

在第 335 页的图 35 中的示例中，语句概要文件部分包括用 *OPTGUIDELINES* 元素表示的语句优化准则部分。

此部分指定所引用的语句期望的查询执行计划的各个方面。它由一个或多个*访问*请求和*连接*请求组成，这些请求指定访问和连接语句中的表所需的方法。第 335 页的图 35 中的准则包含一个访问请求，它指定嵌套子查询中引用的 SUPPLIERS 表使用名为 I_SUPPKEY 的索引。

在第 335 页的图 35 中，*indexScan* 元素指示索引访问请求。indexScan 元素的 *tableReference* 元素指定 SUPPLIERS 表为访问请求的目标。*index* 元素指定使用 I_SUPPKEY 索引。

准则只需要指定期望的查询执行计划的各个部分。优化器能够应用其基于成本的模型来选择方案的其余部分。

创建优化概要文件:

根据当前优化概要文件模式，优化概要文件必须有效。

由于优化概要文件可以包含许多准则的组合，所以此任务仅指定创建任何优化概要文件的公共步骤。

要创建优化概要文件:

1. 启动 XML 编辑器。如果可能的话，使用具有模式验证功能的 XML 编辑器。优化器不执行 XML 验证。
2. 使用有意义的名称创建新的 XML 文档。您可能希望名称描述文档所适用的语句范围，例如，inventory_db.xml。
3. 将 XML 声明添加至此文档。如果未指定编码格式，那么假定为 UTF-8。如果可能的话，使用 UTF-16 编码来保存文档。DB2 在处理此编码时更有效。

   ```
   <?xml version="1.0" encoding="UTF-16" ?>
   ```

4. 将优化概要文件部分添加至此文档。

   ```
   <OPTPROFILE VERSION="9.1.0.0">
      </OPTPROFILE>
   ```

5. 在优化概要文件元素内，根据需要创建全局级别准则或语句级别准则，然后保存文件。

创建语句优化准则:

下列步骤描述如何创建语句优化准则。

『创建优化概要文件』，您想要在此概要文件中插入语句准则。

要创建语句优化准则:

1. 使用所有其他调整选项。请参阅 *Performance Guide*。例如:
   a. 确保 RUNSTATS 实用程序最近更新了数据分发统计信息。

b. 确保 DB2 正在运行，并且为工作负载设置了正确的优化级别。

c. 确保优化器具有适当的索引来访问查询中引用的表。

2. 对问题语句运行 EXPLAIN 命令并分析输出以确定准则是否合适。如果您确定某个语句优化准则将有帮助，请继续。

3. 通过运行类似下列的查询来获取*原始*语句:

```
SELECT STATEMENT TEXT
       FROM EXPLAIN_STATEMENT
       WHERE  EXPLAIN_LEVEL = '0' AND
           EXPLAIN_REQUESTER = 'SIMMEN' AND
           EXPLAIN_TIME      = '2003-09-08-16.01.04.108161' AND
           SOURCE_NAME       = 'SQLC2E03' AND
           SOURCE_VERSION    = '' AND
           QUERYNO           = 1
```

4. 通过将语句文本插入到语句关键字中来编辑优化概要文件并创建语句概要文件。例如:

```
<STMTPROFILE ID="Guidelines for TPCD Q9">
   <STMTKEY SCHEMA="TPCD"><![CDATA[SELECT S.S_NAME, S.S_ADDRESS, S.S_PHONE,
             S.S_COMMENT
     FROM PARTS P, SUPPLIERS S, PARTSUPP PS
     WHERE   P_PARTKEY = PS.PS_PARTKEY AND S.S_SUPPKEY = PS.PS_SUPPKEY
     AND P.P_SIZE  = 39  AND   P.P_TYPE  = 'BRASS'  AND S.S_NATION
     = 'MOROCCO' AND
     PS.PS_SUPPLYCOST = (SELECT MIN(PS1.PS_SUPPLYCOST)
     FROM   PARTSUPP PS1, SUPPLIERS S1
     WHERE P.P_PARTKEY  = PS1.PS_PARTKEY AND S1.S_SUPPKEY = PS1.PS_SUPPKEY
     AND S1.S_NATION  = S.S_NATION)]]>
     </STMTKEY>    </STMTPROFILE>
```

5. 在语句关键字下面编写语句优化准则。使用显示名来标识访问请求和连接请求中使用的对象。以下是连接请求的示例:

```
<OPTGUIDELINES>    <HSJOIN>
     <TBSCAN TABLE='PS1'/>
     <IXSCAN TABLE='S1'
                       INDEX='I1'/>
 </HSJOIN>
</OPTGUIDELINES>
```

6. 验证并保存文件。

要测试结果，请遵循『配置 DB2 以使用优化概要文件』、第 338 页的『指定优化器将使用的优化概要文件』和第 333 页的『验证是否使用了优化准则』中的过程。如果您未获得期望的结果，请按照第 339 页的『修改优化概要文件』中的描述更改准则（或指定执行计划的更多方面）并更新优化概要文件。需要时请重复这些操作。

配置 DB2 以使用优化概要文件:

在编写好优化概要文件并且针对当前优化概要文件模式（COPS）验证了该概要文件的内容后，这些内容必须与唯一的模式限定名关联，并且必须存储在 SYSTOOLS.OPT_PROFILE 表中。

要配置 DB2 以使用优化概要文件:

1. 按照 第 360 页的『SYSTOOLS.OPT_PROFILE 表』中所示创建优化概要文件表。 此表的每行都包含一个优化概要文件: SCHEMA 和 NAME 列标识优化概要文件，而 PROFILE 列包含优化概要文件的文本。

2. 可选: 您可以授予符合数据库安全性要求的任何权限。无论设置的权限如何，优化器都可以读取此表。

3. 将此表插入到想要使用的优化概要文件中。

指定优化器将使用的优化概要文件: 要指定在程序包级别使用优化概要文件，可以使用 OPTPROFILE 绑定选项。要指定在语句级别使用优化概要文件，可以使用 CURRENT OPTIMIZATION PROFILE 专用寄存器。此专用寄存器包含动态准备好进行优化的语句所使用的优化概要文件的限定名。对于 CLI 应用程序，可以使用 CURRENTOPTIMIZATIONPROFILE 配置选项来为每个连接设置此专用寄存器。

OPTPROFILE 绑定选项设置还指定 CURRENT OPTIMIZATION PROFILE 专用寄存器的缺省优化概要文件。缺省值的优先顺序如下所示:

- 无论任何其他设置为何，OPTPROFILE 绑定选项都适用于所有静态语句。
- 对于动态语句，CURRENT OPTIMIZATION PROFILE 专用寄存器的值由下列内容确定，其优先顺序从低到高:
  - OPTPROFILE 绑定选项
  - CURRENTOPTIMIZATIONPROFILE 客户机配置选项
  - 应用程序中的最新 SET CURRENT OPTIMIZATION PROFILE 语句

将优化概要文件绑定至程序包:

通过使用 BIND 或 PRECOMPILE 命令准备程序包时，可以使用 OPTPROFILE 选项指定要用于程序包的优化概要文件。

这是将优化概要文件应用于静态语句的唯一方法，并且指定的概要文件将适用于程序包中的所有静态语句。通过这种方式指定的优化概要文件还是用于程序包中的动态语句的缺省优化概要文件。

可以使用 API（例如，sqlaprep）或从 CLP 中绑定 SQLJ 和嵌入式 SQL 中的优化概要文件。

例如，要从 CLP 中将库存数据库优化概要文件绑定至库存应用程序:

```
   db2 prep inventapp.sqc bindfile optprofile NEWTON.INVENTDB
      db2 bind inventapp.bnd
      db2 connect reset   db2 terminate   xlc -I$HOME/sqllib/include -c inventapp.c
-o inventapp.o
   xlc -o inventapp inventapp.o -ldb2 -L$HOME/sqllib/lib
```

如果没有为优化概要文件指定模式名，那么 QUALIFIER 选项将用作隐式限定符。

在应用程序内设置优化概要文件:

在应用程序中，可以通过使用 SET CURRENT OPTIMIZATION PROFILE 语句来控制动态语句的当前优化概要文件的设置。您在语句中提供的优化概要文件名必须是模式限定的名称。如果未提供模式名，那么 CURRENT SCHEMA 专用寄存器的值将用作隐式模式限定符。

指定的优化概要文件适用于所有后续动态语句，直到遇到另一个 SET CURRENT OPTIMIZATION PROFILE 语句为止。静态语句不受影响，因为在评估此设置之前已对它们进行预处理和打包。

要在应用程序内设置优化概要文件:

- 可以在整个应用程序中使用 SET CURRENT OPTIMIZATION PROFILE 语句。例如，已根据 JON.SALES 优化概要文件优化以下序列中的最后一个语句。

```
EXEC SQL  SET CURRENT OPTIMIZATION PROFILE = 'NEWTON.INVENTDB';

   /* The following statements are both optimized with 'NEWTON.INVENTDB' */
   EXEC SQL PREPARE stmt FROM SELECT ... ;
   EXEC SQL EXECUTE stmt;

   EXEC SQL EXECUTE IMMEDIATE SELECT ... ;

   EXEC SQL SET CURRENT OPTIMIZATION PROFILE = 'JON.SALES';

   /* This statement is optimized with 'JON.SALES' */
   EXEC SQL EXECUTE IMMEDIATE SELECT ... ;
```

图 36. 使用 SET CURRENT OPTIMIZATION PROFILE 语句更改概要文件

- 如果希望优化器使用在应用程序启动时有效的缺省优化概要文件，那么指定值 NULL。例如:

  ```
  EXEC SQL SET CURRENT OPTIMIZATION PROFILE = NULL;
  ```

- 如果希望优化器不使用优化概要文件，那么指定空字符串（''）。例如:

  ```
  EXEC SQL SET CURRENT OPTIMIZATION PROFILE = '';
  ```

- 如果是 CLI 应用程序，那么可以将 CURRENTOPTIMIZATIONPROFILE 参数添加至 db2cli.ini 文件。要将条目添加至此文件，可使用配置助手或 UPDATE CLI CONFIGURATION 命令。例如:

  ```
  DB2 UPDATE CLI CFG FOR SECTION SANFRAN USING CURRENTOPTIMIZATIONPROFILE JON.SALES
  ```

  这将导致在 db2cli.ini 文件中生成以下条目:

  ```
  [SANFRAN]
  CURRENTOPTIMIZATIONPROFILE=JON.SALES
  ```

  注: 应用程序中的任何 SET CURRENT OPTIMIZATION PROFILE 语句都将覆盖此设置。

修改优化概要文件:

可以通过编辑文档、针对当前优化概要文件模式（COPS）验证此文档，并将 SYSTOOLS.OPT_PROFILE 表中的原始文档替换为新版本来修改优化概要文件。但是，当引用了优化概要文件时，将在内存中编译并对其进行高速缓存；还必须除去这些引用。使用 FLUSH OPTIMIZATION PROFILE CACHE 语句从优化概要文件高速缓存中除去旧的优化概要文件，并使动态方案高速缓存中使用旧概要文件准备的所有语句都失效。

要修改优化概要文件:

1. 编辑优化概要文件以进行必需的更改并验证 XML。
2. 使用新的概要文件更新 SYSTOOLS.OPT_PROFILE 表中的行。
3. 如果未创建第 361 页的『用于清空优化概要文件高速缓存的触发器』，那么发出 FLUSH OPTIMIZATION PROFILE CACHE 语句。

   注: 清空优化概要文件高速缓存时，还会将动态方案高速缓存中使用旧优化概要文件准备的所有动态语句失效。

对此优化概要文件的任何后续引用都会导致优化器读取新的概要文件并将其重新装入到优化概要文件高速缓存中。此外，由于会使在旧优化概要文件下准备的语句软失效，所以将在新优化概要文件下准备对这些语句的任何调用，并将这些调用重新高速缓存在动态方案高速缓存中。

删除优化概要文件:

通过从 SYSTOOLS.OPT_PROFILE 表中删除不再需要的优化概要文件，可以除去此优化概要文件。当引用了优化概要文件时，将在内存中编译并对其进行高速缓存；因此，如果已使用原始概要文件，那么还必须从优化概要文件高速缓存中清空已删除的优化概要文件。

要删除优化概要文件:

1. 从 SYSTOOLS.OPT_PROFILE 表中删除优化概要文件。例如:

```
DELETE FROM SYSTOOLS.OPT_PROFILE
       WHERE SCHEMA = 'NEWTON' AND NAME = 'INVENTDB';
```

2. 如果未创建第 361 页的『用于清空优化概要文件高速缓存的触发器』，那么通过从 CLP 中运行 FLUSH OPTIMIZATION PROFILE CACHE 命令来清空可能包含在优化概要文件高速缓存中的任何版本的优化概要文件。

   注: 此操作还会导致使用旧优化概要文件准备的所有语句在动态方案高速缓存中失效。

后续对此优化概要文件的任何引用都会导致优化器发出原因码为 13 的警告 SQL0437W。

优化概要文件和准则的 XML 模式

当前优化概要文件模式:

以下是优化概要文件的模式。

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" version="1.0">
<!--*******************************************************************************************-->
<!-- Licensed Materials - Property of IBM                                                    -->
<!-- (C) Copyright International Business Machines Corporation 2007. All rights reserved.
-->
<!-- U.S. Government Users Restricted Rights;  Use, duplication or disclosure restricted by  -->
<!-- GSA ADP Schedule Contract with IBM Corp.                                                -->
<!--*******************************************************************************************-->
<!--*******************************************************************************************-->
<!-- Definition of the current optimization profile schema for V9.5.0.0                      -->
<!--                                                                                         -->
<!--   An optimization profile is composed of the following sections:                        -->
<!--                                                                                         -->
<!--    + A  global optimization guidelines section (at most one) which defines optimization -->
<!--     guidelines affecting any statement for which the optimization profile is in effect.    -->
<!--                                                                                         -->
<!--    +  Zero or more statement profile sections, each of which defines optimization       -->
<!--       guidelines for a particular statement for which the optimization profile          -->
<!--       is in effect.                                                                          -->
<!--                                                                                         -->
<!-- The VERSION attribute indicates the version of this optimization profile                -->
<!-- schema.                                                                                 -->
<!--*******************************************************************************************-->
   <xs:element name="OPTPROFILE">
               <xs:complexType>
                    <xs:sequence>
         <!-- Global optimization guidelines section. At most one can be specified. -->
         <xs:element name="OPTGUIDELINES" type="globalOptimizationGuidelinesType" minOccurs="0"/>
         <!-- Statement profile section. Zero or more can be specified  -->
         <xs:element name="STMTPROFILE" type="statementProfileType" minOccurs="0" maxOccurs="unbounded"/>
```

```
                </xs:sequence>
      <!-- Version attribute is currently optional -->
      <xs:attribute name="VERSION" use="optional">
         <xs:simpleType>
            <xs:restriction base="xs:NMTOKEN">
               <xs:enumeration value="9.5.0.0"/>
            </xs:restriction>
         </xs:simpleType>
      </xs:attribute>
   </xs:complexType>
</xs:element>
<!--*****************************************************************************************-->
<!--  Global optimization guidelines supported in this version:                              -->
<!--  + MQTOptimizationChoices elements influence the MQTs considered by the optimizer.       -->
<!--  + computationalPartitionGroupOptimizationsChoices elements can affect repartitioning   -->
<!--    optimizations involving nicknames.                                                     -->
<!--  + The REOPT element can affect how statements involving variables are optimized.        -->
<!-- ***************************************************************************************** -->
   <xs:complexType name="globalOptimizationGuidelinesType">
          <xs:sequence>
      <xs:group ref="MQTOptimizationChoices" />
      <xs:group ref="computationalPartitionGroupOptimizationChoices" />
               <xs:group ref="generalRequest"/>
                 </xs:sequence>
   </xs:complexType><!-- ************************************************************************************************-->
<!--  Elements for affecting materialized query table (MQT) optimization.                    -->
<!--                                                                                         -->
<!--  + MQTOPT - can be used to disable materialized query table (MQT) optimization.         -->
<!--           If disabled, the optimizer will not consider MQTs to optimize the statement.            -->
<!--                                                                                         -->
<!--  + MQT - multiple of these can be specified. Each specifies an MQT that should be      -->
<!--     considered for optimizing the statement. Only specified MQTs will be considered.          -->
<!--                                                                                         -->
<!--*****************************************************************************************-->
   <xs:group name="MQTOptimizationChoices">
       <xs:choice>
               <xs:element name="MQTOPT" minOccurs="0" maxOccurs="1">
                     <xs:complexType>
                           <xs:attribute name="OPTION" type="optionType" use="optional"/>
            </xs:complexType>      </xs:element>
               <xs:element name="MQT" minOccurs="0" maxOccurs="unbounded">
                     <xs:complexType>
                           <xs:attribute name="NAME" type="xs:string" use="required"/>
            </xs:complexType>      </xs:element>
         </xs:choice>
</xs:group><!-- ************************************************************************************************-->
<!--  Elements for affecting computational partition group (CPG) optimization.               -->
<!--                                                                                         -->
<!--   + PARTOPT - can be used disable the computational partition group (CPG) optimization  -->
<!--               which is used to dynamically redistributes inputs to join, aggregation,   -->
<!--                and union operations when those inputs are results of remote queries.                    -->
<!--                                                                                         -->
<!--    + PART - Define the partition groups to be used in CPG optimizations.                -->
<!--                                                                                         -->
<!-- *****************************************************************************************-->
   <xs:group name="computationalPartitionGroupOptimizationChoices">
       <xs:choice>
               <xs:element name="PARTOPT" minOccurs="0" maxOccurs="1">
                     <xs:complexType>
                           <xs:attribute name="OPTION" type="optionType" use="optional"/>
         </xs:complexType>      </xs:element>
               <xs:element name="PART" minOccurs="0" maxOccurs="1">
                     <xs:complexType>
                           <xs:attribute name="NAME" type="xs:string" use="required"/>
            </xs:complexType>      </xs:element>
         </xs:choice>
</xs:group><!-- ************************************************************************************************-->
<!--  Definition of a statement profile.                                                     -->
<!--  Comprised of a statement key and optimization guidelines.                              -->
<!--  The statement key specifies semantic information used to identify the statement to     -->
<!--  which optimization guidelines apply. The optional ID attribute provides the statement  -->
<!--  profile with a name for use in EXPLAIN output.                                         -->
<!-- *****************************************************************************************-->
   <xs:complexType name="statementProfileType">
           <xs:sequence>
        <!-- Statement key element  -->
               <xs:element name="STMTKEY" type="statementKeyType"/>
         <xs:element name="OPTGUIDELINES" type="optGuidelinesType"/>
                 </xs:sequence>
```

```
    <!-- ID attribute.Used in explain output to indicate the statement profile was used. -->
         <xs:attribute name="ID" type="xs:string" use="optional"/>
   </xs:complexType><!--*****************************************************************************************-->
<!--  Definition of the statement key. The statement key provides semantic information used  -->
<!--  to identify the statement to which the optimization guidelines apply.                  -->
<!-- The statement key is comprised of:                                                      -->
<!--      + statement text (as written in the application)                                   -->
<!--      + default schema (for resolving unqualified table names in the statement)          -->
<!--      + function path (for resolving unqualified types and functions in the statement)   -->
<!-- The statement text is provided as element data whereas the default schema and function  -->
<!--  path are provided via the SCHEMA and FUNCPATH elements, respectively.                  -->
<!--*****************************************************************************-->
   <xs:complexType name="statementKeyType" mixed="true">
         <xs:attribute name="SCHEMA" type="xs:string" use="optional"/>
         <xs:attribute name="FUNCPATH" type="xs:string" use="optional"/>
   </xs:complexType><!--*****************************************************************************************-->
<!--                                                                                      -->
<!-- Optimization guideline elements can be chosen from general requests, rewrite        -->
<!-- requests access requests, or join requests.                                          -->
<!--                                                                                      -->
<!-- General requests affect the search space which defines the alternative query        -->
<!-- transformations, access methods, join methods, join orders, and other optimizations, -->
<!-- considered by the optimizer.                                                         -->
<!--                                                                                      -->
<!-- Rewrite requests affect the query transformations used in determining the optimized  -->
<!-- statement.                                                                           -->
<!--                                                                                      -->
<!-- Access requests affect the access methods considered by the cost-based optimizer,    -->
<!-- and join requests affect the join methods and join order used in the execution plan. -->
<!--                                                                                      -->
<!--*****************************************************************************-->
   <xs:element name="OPTGUIDELINES" type="optGuidelinesType"/>
   <xs:complexType name="optGuidelinesType">
           <xs:sequence>
      <xs:group ref="generalRequest" minOccurs="0" maxOccurs="1"/>
              <xs:choice maxOccurs="unbounded">
         <xs:group ref="rewriteRequest" />
                  <xs:group ref="accessRequest"/>
                  <xs:group ref="joinRequest"/>
            </xs:choice>
                </xs:sequence>
   </xs:complexType><!--*****************************************************************************************-->
<!--  Choices of general request elements.                                              -->
<!-- REOPT can be used to override the setting of the REOPT bind option.                -->
<!--*************************************************************************** -->
   <xs:group name="generalRequest">
           <xs:sequence>
               <xs:element name="REOPT" type="reoptType" minOccurs="0" maxOccurs="1"/>
               <xs:element name="DEGREE" type="degreeType" minOccurs="0" maxOccurs="1"/>
               <xs:element name="QRYOPT" type="qryoptType" minOccurs="0" maxOccurs="1"/>
      <xs:element name="RTS" type="rtsType" minOccurs="0" maxOccurs="1"/>
                 </xs:sequence>
</xs:group><!--*****************************************************************************************-->
<!--  Choices of rewrite request elements.                                          -->
<!--*****************************************************************************-->
   <xs:group name="rewriteRequest">
           <xs:sequence>
               <xs:element name="INLIST2JOIN" type="inListToJoinType" minOccurs="0"/>
               <xs:element name="SUBQ2JOIN"   type="subqueryToJoinType" minOccurs="0"/>
               <xs:element name="NOTEX2AJ"    type="notExistsToAntiJoinType" minOccurs="0"/>
               <xs:element name="NOTIN2AJ"    type="notInToAntiJoinType" minOccurs="0"/>
                 </xs:sequence>
</xs:group><!--*************************************************************************** -->
<!--  Choices for access request elements.                                              -->
<!--  TBSCAN - table scan access request element                                        -->
<!--  IXSCAN - index scan access request element                                        -->
<!--  LPREFETCH - list prefetch access request element                                  -->
<!--  IXAND - index ANDing access request element                                       -->
<!--  IXOR - index ORing access request element                                         -->
<!--  ACCESS - indicates the optimizer should choose the access method for the table    -->
<!--*************************************************************************** -->
   <xs:group name="accessRequest">
       <xs:choice>
               <xs:element name="TBSCAN" type="tableScanType"/>
               <xs:element name="IXSCAN" type="indexScanType"/>
               <xs:element name="LPREFETCH" type="listPrefetchType"/>
               <xs:element name="IXAND" type="indexAndingType"/>
               <xs:element name="IXOR" type="indexOringType"/>
               <xs:element name="ACCESS" type="anyAccessType"/>
```

```
        </xs:choice>
</xs:group><!--****************************************************************************** -->
<!-- Choices for join request elements.                                                    -->
<!-- NLJOIN - nested-loops join request element                                            -->
<!-- MSJOIN - sort-merge  join request element                                             -->
<!-- HSJOIN - hash  join request element                                                   -->
<!-- JOIN - indicates that the optimizer is to choose the join method.                     -->
<!--****************************************************************************** -->
   <xs:group name="joinRequest">
       <xs:choice>
              <xs:element name="NLJOIN" type="nestedLoopJoinType"/>
              <xs:element name="HSJOIN" type="hashJoinType"/>
              <xs:element name="MSJOIN" type="mergeJoinType"/>
              <xs:element name="JOIN" type="anyJoinType"/>
        </xs:choice>
</xs:group><!--****************************************************************************** -->
<!--  REOPT general request element. Can override REOPT setting at the package, db,          -->
<!-- dbm level.                                                                            -->
<!--****************************************************************************** -->
   <xs:complexType name="reoptType">
         <xs:attribute name="VALUE" use="required">
          <xs:simpleType>
              <xs:restriction base="xs:string">
                          <xs:enumeration value="ONCE"/>
                          <xs:enumeration value="ALWAYS"/>
              </xs:restriction>
     </xs:simpleType>       </xs:attribute>
   </xs:complexType><!--*************************************************************************************-->
<!-- RTS general request element to enable, disable or provide a time budget for           -->
<!-- real-time statistics collection.                                                      -->
<!-- OPTION attribute allows enabling or disabling real-time statistics.                   -->
<!-- TIME attribute provides a time budget in milliseconds for real-time statistics collection.-->
<!--****************************************************************************** -->
<xs:complexType name="rtsType">
         <xs:attribute name="OPTION" type="optionType" use="optional" default="ENABLE"/>
   <xs:attribute name="TIME" type="xs:nonNegativeInteger" use="optional"/>
   </xs:complexType><!--*************************************************************************************-->
<!--   Definition of an "IN list to join" rewrite request                                 -->
<!--   OPTION attribute allows enabling or disabling the alternative.                       -->
<!--   TABLE attribute allows request to target IN list predicates applied to a           -->
<!--   specific table reference. COLUMN attribute allows request to target a specific IN list -->
<!--   predicate.                                                                           -->
<!--****************************************************************************** -->
   <xs:complexType name="inListToJoinType">
         <xs:attribute name="OPTION" type="optionType" use="optional" default="ENABLE"/>
         <xs:attribute name="TABLE"  type="xs:string"  use="optional"/>
         <xs:attribute name="COLUMN" type="xs:string"  use="optional"/>
   </xs:complexType><!--*************************************************************************************-->
<!--   Definition of a "subquery to join" rewrite request                                 -->
<!--  The OPTION attribute allows enabling or disabling the alternative.                    -->
<!--****************************************************************************** -->
   <xs:complexType name="subqueryToJoinType">
         <xs:attribute name="OPTION" type="optionType" use="optional" default="ENABLE"/>
   </xs:complexType><!--*************************************************************************************-->
<!--   Definition of a "not exists to anti-join" rewrite request                          -->
<!--  The OPTION attribute allows enabling or disabling the alternative.                    -->
<!--****************************************************************************** -->
   <xs:complexType name="notExistsToAntiJoinType">
         <xs:attribute name="OPTION" type="optionType" use="optional" default="ENABLE"/>
   </xs:complexType><!--*************************************************************************************-->
<!--   Definition of a "not IN to anti-join" rewrite request                              -->
<!--  The OPTION attribute allows enabling or disabling the alternative.                    -->
<!--****************************************************************************** -->
   <xs:complexType name="notInToAntiJoinType">
         <xs:attribute name="OPTION" type="optionType" use="optional" default="ENABLE"/>
   </xs:complexType><!--*************************************************************************************-->
<!-- Effectively the superclass from which  all access request elements inherit.           -->
<!-- This type currently defines TABLE and TABID attributes, which can be used to tie an   -->
<!-- access request to a table reference in the query.                                     -->
<!-- The TABLE attribute value is used to identify a table reference using identifiers     -->
<!-- in the original SQL statement. The TABID attribute value is used to identify a table  -->
<!-- referece using the unique correlation name provided via the                           -->
<!-- optimized statement. If both the TABLE and TABID attributes are specified, the TABID  -->
<!-- field is ignored. The FIRST attribute indicates that the access should be the first   -->
<!-- access in the join sequence for the FROM clause.                                      -->
<!--****************************************************************************** -->
   <xs:complexType name="accessType" abstract="true">
         <xs:attribute name="TABLE"  type="xs:string"  use="optional"/>
         <xs:attribute name="TABID" type="xs:string" use="optional"/>
```

```
        <xs:attribute name="FIRST" type="xs:string" use="optional" fixed="TRUE"/>
   </xs:complexType><!--*****************************************************************************************-->
<!-- Definition of an table scan access request method.                                    -->
<!--*****************************************************************************************-->
   <xs:complexType name="tableScanType">
       <xs:complexContent>
               <xs:extension base="accessType"/>
       </xs:complexContent>
   </xs:complexType><!-- *****************************************************************************************-->
<!-- Definition of an index scan access request element. The index name is optional.         -->
<!--************************************************************************************ -->
   <xs:complexType name="indexScanType">
       <xs:complexContent>
               <xs:extension base="accessType">
                     <xs:attribute name="INDEX" type="xs:string" use="optional"/>
            </xs:extension>
       </xs:complexContent>
   </xs:complexType><!--*****************************************************************************************-->
<!-- Definition of a list prefetch access request element. The index name is optional.    -->
<!--************************************************************************************ -->
   <xs:complexType name="listPrefetchType">
       <xs:complexContent>
               <xs:extension base="accessType">
                     <xs:attribute name="INDEX" type="xs:string" use="optional"/>
            </xs:extension>
       </xs:complexContent>
   </xs:complexType><!--*****************************************************************************************-->
<!-- Definition of an index ANDing access request element.                                 -->
<!-- A single index scan be specified via the INDEX attribute. Multiple indexes           -->
<!-- can be specified via INDEX elements. The index element specification supersedes the  -->
<!-- attribute specification.  If a single index is specified, the optimizer will use the -->
<!-- index as the first index of the index ANDing access method and will choose addi-     -->
<!-- tional indexes using cost. If multiple indexes are specified the optimizer will      -->
<!-- use exactly those indexes in the specified order. If no indexes are specified        -->
<!-- via either the INDEX attribute or INDEX elements, then the optimizer will choose     -->
<!-- all indexes based upon cost.                                                          -->
<!--************************************************************************************ -->
<xs:complexType name="indexAndingType">
       <xs:complexContent>
               <xs:extension base="accessType">
                     <xs:sequence minOccurs="0">
                           <xs:element name="INDEX" type="indexType" minOccurs="2" maxOccurs="unbounded"/>
                       </xs:sequence>
                     <xs:attribute name="INDEX" type="xs:string" use="optional"/>
            </xs:extension>
       </xs:complexContent>
   </xs:complexType><!--*****************************************************************************************-->
<!-- Definition of an INDEX element method. Index set is optional. If specified,          -->
<!-- at least 2 are required.                                                              -->
<!--************************************************************************************ -->
   <xs:complexType name="indexType">
         <xs:attribute name="IXNAME" type="xs:string" use="optional"/>
   </xs:complexType><!--*****************************************************************************************-->
<!-- Use for derived table access or other cases where the access method is not of        -->
<!-- consequence.                                                                          -->
<!--*****************************************************************************************-->
   <xs:complexType name="anyAccessType">
       <xs:complexContent>
               <xs:extension base="accessType"/>
       </xs:complexContent>
   </xs:complexType><!--*****************************************************************************************-->
<!-- Definition of an index ORing access                                                   -->
<!-- Cannot specify more details (e.g indexes). Optimizer will choose the details based   -->
<!-- upon cost.                                                                            -->
<!--************************************************************************************ -->
   <xs:complexType name="indexOringType">
       <xs:complexContent>
               <xs:extension base="accessType"/>
       </xs:complexContent>
   </xs:complexType><!--*****************************************************************************************-->
<!-- Effectively the super class from which join request elements inherit.                 -->
<!-- This type currently defines join element inputs and also the FIRST attribute.         -->
<!-- A join request must have exactly two nested sub-elements. The sub-elements can be     -->
<!-- either an access request or another join request. The first sub-element represents    -->
<!-- outer table of the join operation while the second element represents the inner       -->
<!-- table. The FIRST attribute indicates that the join result should be the first join    -->
<!--  relative to other tables in the same FROM clause.                                     -->
<!--************************************************************************************ -->
   <xs:complexType name="joinType" abstract="true">
```

```
        <xs:choice minOccurs="2" maxOccurs="2">
              <xs:group ref="accessRequest"/>
              <xs:group ref="joinRequest"/>
        </xs:choice>
        <xs:attribute name="FIRST" type="xs:string" use="optional" fixed="TRUE"/>
   </xs:complexType><!--*************************************************************************************** -->
<!--  Definition of nested loop join access request.  Subclass of joinType.                 -->
<!--  Does not add any elements or attributes.                                              -->
<!--************************************************************************************** -->
   <xs:complexType name="nestedLoopJoinType">
       <xs:complexContent>
              <xs:extension base="joinType"/>
       </xs:complexContent>
   </xs:complexType><!--*************************************************************************************** -->
<!--  Definition of merge join access request.  Subclass of joinType.                       -->
<!--  Does not add any elements or attributes.                                              -->
<!--************************************************************************************** -->
   <xs:complexType name="mergeJoinType">
       <xs:complexContent>
              <xs:extension base="joinType"/>
       </xs:complexContent>
   </xs:complexType><!--*************************************************************************************** -->
<!--  Definition of hash  join access request.  Subclass of joinType.                       -->
<!--  Does not add any elements or attributes.                                              -->
<!--************************************************************************************** -->
   <xs:complexType name="hashJoinType">
       <xs:complexContent>
              <xs:extension base="joinType"/>
       </xs:complexContent>
   </xs:complexType><!--*************************************************************************************** -->
<!--  Any join is a subclass of binary join. Does not extend it in any way.                 -->
<!--  Does not add any elements or attributes.                                              -->
<!--************************************************************************************** -->
   <xs:complexType name="anyJoinType">
       <xs:complexContent>
              <xs:extension base="joinType"/>
       </xs:complexContent>
   </xs:complexType><!--*************************************************************************************** -->
<!--   Allowable values for an OPTION attribute.                                            -->
<!--************************************************************************************** -->
<xs:simpleType name="optionType">
        <xs:restriction base="xs:string">
       <xs:enumeration value="ENABLE"/>
       <xs:enumeration value="DISABLE"/>
        </xs:restriction>
</xs:simpleType>
<!--*************************************************************************************** -->
<!--   Definition of the qryopt type: the only values allowed are 0, 1, 2, 3, 5, 7 and 9     -->
<!--*************************************************************************************** -->
   <xs:complexType name="qryoptType">
         <xs:attribute name="VALUE" use="required">
          <xs:simpleType>
              <xs:restriction base="xs:string">
                          <xs:enumeration value="0"/>
                          <xs:enumeration value="1"/>
                          <xs:enumeration value="2"/>
                          <xs:enumeration value="3"/>
                          <xs:enumeration value="5"/>
                          <xs:enumeration value="7"/>
                          <xs:enumeration value="9"/>
              </xs:restriction>
     </xs:simpleType>       </xs:attribute>
   </xs:complexType><!--*************************************************************************************** -->
<!-- Definition of the degree type: any number between 1 and 32767 or the strings ANY or -1  -->
<!--************************************************************************************** -->
   <xs:simpleType name="intStringType">
         <xs:union>
          <xs:simpleType>
                    <xs:restriction base="xs:integer">
                          <xs:minInclusive value="1"></xs:minInclusive>
                          <xs:maxInclusive value="32767"></xs:maxInclusive>
              </xs:restriction>
     </xs:simpleType>          <xs:simpleType>
              <xs:restriction base="xs:string">
                          <xs:enumeration value="ANY"/>
                          <xs:enumeration value="-1"/>
              </xs:restriction>
     </xs:simpleType>         </xs:union>
```

```
</xs:simpleType>
   <xs:complexType name="degreeType">
 <xs:attribute name="VALUE" type="intStringType"></xs:attribute>
   </xs:complexType></xs:schema>
```

OPTPROFILE 元素的 XML 模式: OPTPROFILE 元素是优化概要文件的根。按如下方式定义 OPTPROFILE 元素:

XML Schema

```
   <xs:element name="OPTPROFILE">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="OPTGUIDELINES"
                   type="globalOptimizationGuidelinesType"
              minOccurs="0"/>
            <xs:element name="STMTPROFILE" type="statementProfileType" minOccurs="0"
              maxOccurs="unbounded"/>
         </xs:sequence>
         <xs:attribute name="VERSION" use="optional">
            <xs:simpleType>
               <xs:restriction base="xs:NMTOKEN">
                  <xs:enumeration value="9.1.0.0"/>
               </xs:restriction>
            </xs:simpleType>         </xs:attribute>
      </xs:complexType>   </xs:element>
```

描述

可选的 OPTGUIDELINES 子元素定义优化概要文件的全局优化准则部分。每个 STMTPROFILE 子元素都定义一个语句概要文件部分。VERSION 属性标识针对其创建并验证给定优化概要文件的当前优化概要文件模式。

全局 OPTGUIDELINES 元素的 XML 模式: *globalOptimizationGuidelinesType* 类型定义全局 OPTGUIDELINES 元素的格式。

XML Schema

```
      <xs:complexType name="globalOptimizationGuidelinesType">
                  <xs:sequence>
                <xs:group ref="MQTOptimizationChoices"/>
                <xs:group ref="computationalPartitionGroupOptimizationChoices"/>
                <xs:group ref="generalRequest"/>
                  </xs:sequence>
      </xs:complexType>
```

描述

可以使用 MQTOptimizationChoices 组中的元素、computationalPartitionGroupChoices 组中的元素或者使用 REOPT 元素定义全局优化准则。

在 第 347 页的『MQT 优化选项』中定义的 MQTOptimizationChoices 组元素可用于影响 MQT 替换。在第 347 页的『计算分区组优化选项』中定义的 computationalPartitionGroupOptimizationChoices 组元素可用于影响计算分区组优化。

计算分区组优化涉及动态再分发从远程数据源读取的数据。它仅应用于分区的联合数据库配置。

在 第 348 页的『REOPT 全局优化准则』中定义的 REOPT 元素可用于影响优化引用变量的语句的时间。

MQT 优化选项: 组 *MQTOptimizationChoices* 定义一组可用于影响具体化查询表（MQT）优化的元素。特别是，可以使用这些元素来启用或禁用考虑 MQT 替换，或指定优化器考虑的完整 MQT 集。

XML Schema

```
<xs:group name="MQTOptimizationChoices">
   <xs:choice>
      <xs:element name="MQTOPT" minOccurs="0" maxOccurs="1">
         <xs:complexType>
            <xs:attribute name="OPTION" type="optionType" use="optional"/>
         </xs:complexType>         </xs:element>
      <xs:element name="MQT" minOccurs="0" maxOccurs="unbounded">
         <xs:complexType>
            <xs:attribute name="NAME" type="xs:string" use="required"/>
         </xs:complexType>         </xs:element>
   </xs:choice>
</xs:group>
```

描述

MQTOPT 元素用于启用或禁用考虑 MQT 优化。OPTION 属性可以指定值 ENABLE 或 DISABLE。OPTION 属性的缺省值是 ENABLE。

或者，可以不提供 MQT 元素或提供多个 MQT 元素。MQT 元素的 NAME 属性标识优化器要考虑的 MQT。在 NAME 属性中用于形成对 MQT 的引用的规则与形成对显示表名的引用的规则相同。如果指定了一个或多个 MQT 元素，那么优化器只考虑这些 MQT。仍然会根据成本决定是否使用一个或多个指定的 MQT 来执行 MQT 替换。

示例

以下示例说明了如何禁用 MQT 优化。

```
<OPTGUIDELINES>            <MQTOPT OPTION='DISABLE'/>
</OPTGUIDELINES>
```

以下示例说明了如何限制仅对 MQT“Tpcd.PARTSMQT”和“COLLEGE.STUDENTS”进行 MQT 优化。

```
<OPTGUIDELINES>            <MQT NAME='Tpcd.PARTSMQT'/>
      <MQT NAME='COLLEGE.STUDENTS'/>
<OPTGUIDELINES>
```

计算分区组优化选项: 组 *computationalPartitionGroupOptimizationChoices* 定义一组可用于影响计算分区组优化的元素。特别是，可以使用这些元素来启用或禁用计算分区组优化，或指定要用于计算分区组优化的特定分区组。

XML Schema

```
<xs:group name="computationalPartitionGroupOptimizationChoices">
   <xs:choice>
      <xs:element name="PARTOPT" minOccurs="0" maxOccurs="1">
         <xs:complexType>
            <xs:attribute name="OPTION" type="optionType" use="optional"/>
         </xs:complexType>         </xs:element>
      <xs:element name="PART" minOccurs="0" maxOccurs="1">
         <xs:complexType>
            <xs:attribute name="NAME" type="xs:string" use="required"/>
         </xs:complexType>         </xs:element>
   </xs:choice>
</xs:group>
```

描述

PARTOPT 元素用于启用或禁用考虑计算分区组优化。OPTION 属性可以指定值 ENABLE 或 DISABLE。OPTION 属性的缺省值是 ENABLE。

或者，可以使用 PART 元素来指定要用于计算分区组优化的分区组。PART 元素的 NAME 属性标识分区组，并且必须标识现有分区组。将根据成本决定是否使用指定的分区组执行动态再分发。

示例

以下示例说明了如何禁用计算分区组优化。

```
<OPTGUIDELINES>            <PARTOPT OPTION='DISABLE'/>
</OPTGUIDELINES>
```

以下示例显示了如何指示将分区组 WORKPART 用于计算分区组优化:

```
<OPTGUIDELINES>            <MQT NAME='Tpcd.PARTSMQT'/>
      <PART NAME='WORKPART'/>
<OPTGUIDELINES>
```

REOPT 全局优化准则: REOPT 全局优化准则控制优化包含变量（主变量、参数标记、全局变量或专用寄存器）的 DML 语句的时间。全局优化准则和语句优化准则的 REOPT 元素的描述和语法相同。请参阅第 351 页的『REOPT 请求』。

STMTPROFILE 元素的 XML 模式: *statementProfileType* 类型定义 STMTPROFILE 元素的格式。

XML Schema

```
<xs:complexType name="statementProfileType">
              <xs:sequence>
            <xs:element name="STMTKEY" type="statementKeyType"/>
            <xs:element name="OPTGUIDELINES" type="optGuidelinesType"/>
              </xs:sequence>
      <xs:attribute name="ID" type="xs:string" use="optional"/>
</xs:complexType>
```

描述

语句概要文件指定适用于特定语句（优化概要文件对此语句有效）的优化准则。语句概要文件由下列部分组成:

- 语句关键字 – 优化概要文件可以对应用程序中的多个语句有效。优化器通过使用语句关键字自动将每个语句概要文件与应用程序的正确语句匹配。此方法允许用户为语句提供优化准则，而不必编辑应用程序。

  语句关键字由应用程序中所编写的语句的文本以及明确标识正确的应用程序语句所需的其他信息组成。『STMTKEY 元素的 XML 模式』中所描述的 STMTKEY 子元素表示语句关键字。
- 语句级别优化准则 – 语句概要文件的此部分指定对语句关键字所标识的语句有效的优化准则。请参阅第 350 页的『语句级别 OPTGUIDELINES 元素的 XML 模式』。
- 语句概要文件名 – 用户提供的名称，在诊断输出中用来指示用于优化语句的特定语句概要文件。

STMTKEY 元素的 XML 模式: *statementKeyType* 类型定义 STMTKEY 元素的格式。

XML Schema

```
<xs:complexType name="statementKeyType" mixed="true">
      <xs:attribute name="SCHEMA" type="xs:string" use="optional"/>
      <xs:attribute name="FUNCPATH" type="xs:string" use="optional"/>
</xs:complexType>   </xs:schema>
```

描述

语句关键字的语句文本部分被作为起始和结束 STMTKEY 元素标记之间的数据包括。

此可选 SCHEMA 属性可用于指定语句关键字的缺省模式部分。

可选 FUNCPATH 属性可用于指定语句关键字的函数路径部分。函数路径中可以包括多个函数路径，每个路径用逗号分隔。指定的函数路径必须与编译关键字中指定的函数路径完全匹配。

示例

只要编译关键字具有缺省模式“COLLEGE”和函数路径“SYSIBM,SYSFUN,SYSPROC,DAVE”，以下示例语句关键字就与语句“select * from orders where foo(orderkey) > 20”匹配。

```
<STMTKEY SCHEMA='COLLEGE' FUNCPATH='SYSIBM,SYSFUN,SYSPROC,DAVE'>
  <![CDATA[select * from orders" where foo(orderkey) > 20]]>
  </STMTKEY>
```

注：因为语句文本包含特殊 XML 字符“>”，所以在语句文本的周围使用了 CDATA 部分（以 <![CDATA[开头并以]]> 结尾）。

语句关键字和编译关键字匹配: 语句关键字用于标识语句级别优化准则适用于的特定应用程序语句。当 DB2 编译任何静态或动态 SQL 语句时，某些参数的设置（由专用寄存器以及绑定选项或预编译选项设置）会影响编译器在语义上解释语句的方式。SQL 语句和这些特定 SQL 编译器参数的设置一起构成*编译关键字*。语句关键字的每部分都对应于编译关键字的一部分。

*语句关键字*由下列部分组成:

- 语句文本 - 应用程序中编写的语句的文本。
- 缺省模式 - 用作未限定表名的隐式量词的模式名。此部分是可选的，但如果语句文本中存在未限定的表名，那么应提供此模式名。
- 函数路径 - 解析未限定的函数和数据类型引用时使用的函数路径。此部分是可选的，但如果语句中存在未限定的用户定义的函数或用户定义的类型，那么应提供此路径。

当 DB2 编译 SQL 语句并查找活动的优化概要文件时，它将尝试使优化概要文件中的每个语句关键字与当前编译关键字匹配。如果语句关键字的每个指定部分都与编译关键字的相应部分匹配，那么认为语句关键字和编译关键字匹配。如果未指定语句关键字的某个部分，那么缺省情况下认为省略的部分匹配。实际上，语句关键字的每个未指定部分都被视为通配符，它与所有编译关键字的相应部分匹配。

DB2 找到与当前编译关键字匹配的语句关键字后，它就停止查找；因此，如果优化概要文件中有多个语句概要文件的语句关键字与当前编译关键字匹配，那么只使用第一个这种语句概要文件（按照文档顺序）。此外，在这种情况下，不会发出任何错误或警告。

语句级别 OPTGUIDELINES 元素的 XML 模式: 语句概要文件的 OPTGUIDELINES 元素定义对关联的语句关键字所标识的语句有效的优化准则。OPTGUIDELINES 元素被定义为具有类型 *optGuidelinesType*，如下所示:

XML Schema

```
          <xs:element name="OPTGUIDELINES" type="optGuidelinesType"/>
    <xs:complexType name="optGuidelinesType">
                <xs:sequence>
              <xs:group ref="general request" minOccurs="0" maxOccurs="1"/>
              <xs:choice maxOccurs="unbounded">
                    <xs:group ref="rewriteRequest"/>
                 <xs:group ref="accessRequest"/>
                 <xs:group ref="joinRequest"/>
           </xs:choice>
               </xs:sequence>
    </xs:complexType>
```

描述

optGuidelinesType 组定义 OPTGUIDELINES 元素的有效子元素集。DB2 优化器认为每个子元素是一个优化准则。可以将子元素分类为一般请求元素、重写请求元素、访问请求元素或连接请求元素。

- 一般请求元素用于指定一般优化准则。一般优化准则可用于更改优化器的搜索空间。
- 重写请求元素可用于指定查询重写优化准则。查询重写准则可用于影响在确定已优化的语句时应用的查询变换。
- 访问和连接请求元素是方案优化准则。方案优化准则可用于影响在已优化的语句的执行计划中使用的访问方法、连接方法和连接顺序。

注: 在语句概要文件中指定的优化准则优先于在优化概要文件的全局部分中指定的优化准则。

一般优化准则的 XML 模式: 一般优化准则用于指定不特定于优化过程的特定阶段的准则。一般优化准则可用于更改优化器的搜索空间。它们包括 *generalRequest* 组。

```
<!--********************************************************************************** --> \
<!--  Choices of general request elements.                                             --> \
<!-- REOPT can be used to override the setting of the REOPT bind option.               --> \
<!--********************************************************************************** --> \
   <xs:group name="generalRequest">
                <xs:sequence>
              <xs:element name="REOPT" type="reoptType" minOccurs="0" maxOccurs="1"/>
              <xs:element name="DEGREE" type="degreeType" minOccurs="0" maxOccurs="1"/>
              <xs:element name="QRYOPT" type="qryoptType" minOccurs="0" maxOccurs="1"/>
      <xs:element name="RTS" type="rtsType" minOccurs="0" maxOccurs="1"/>
                </xs:sequence>
</xs:group>
```

描述

可以使用一般请求元素来定义一般优化准则。一般优化准则影响优化搜索空间，从而可以影响重写和基于成本的优化准则的适用性。

DEGREE 请求: 可以使用 DEGREE 一般请求元素来覆盖绑定选项、dft_degree 数据库配置参数或先前的 SET CURRENT DEGREE 语句的设置。仅当配置了数据库管理器

以实现分区内并行性时才考虑 DEGREE 一般请求元素，如果没有为实现分区内并行性而配置数据库管理器，那么会返回警告。DEGREE 一般请求元素由 degreeType 定义，如下所示:

XML Schema

```
      <xs:simpleType name="intStringType">
            <xs:union>
             <xs:simpleType>
                        <xs:restriction base="xs:integer">
                              <xs:minInclusive value="1"></xs:minInclusive>
                              <xs:maxInclusive value="32767"></xs:maxInclusive>
                  </xs:restriction>
         </xs:simpleType>             <xs:simpleType>
                  <xs:restriction base="xs:string">
                              <xs:enumeration value="ANY"/>
                              <xs:enumeration value="-1"/>
                  </xs:restriction>
         </xs:simpleType>            </xs:union>
   </xs:simpleType>
      <xs:complexType name="degreeType">
            <xs:attribute name="VALUE"
                  type="intStringType"></xs:attribute>
      </xs:complexType>
```

描述

DEGREE 一般请求元素具有必需的 VALUE 属性，此属性指定 DEGREE 选项的设置。此属性可以使用 1 到 32767 之间的整数值或字符串值“-1”或“ANY”。值 -1（等同于“ANY”）指示使用的并行度由 DB2 确定。值 1 指示查询不应使用分区内并行性。

QRYOPT 请求: 可以使用 QRYOPT 一般请求元素来覆盖绑定选项、dft_qryopt 数据库配置参数或先前的 SET CURRENT QUERY OPTIMIZATION 语句的设置。QRYOPT 一般请求元素由 qryoptType 定义，如下所示:

XML Schema

```
      <xs:complexType name="qryoptType">
            <xs:attribute name="VALUE" use="required">
             <xs:simpleType>
                  <xs:restriction base="xs:string">
                              <xs:enumeration value="0"/>
                              <xs:enumeration value="1"/>
                              <xs:enumeration value="2"/>
                              <xs:enumeration value="3"/>
                              <xs:enumeration value="5"/>
                              <xs:enumeration value="7"/>
                              <xs:enumeration value="9"/>
                  </xs:restriction>
         </xs:simpleType>          </xs:attribute>
      </xs:complexType>
```

描述

QRYOPT 一般请求元素具有必需的 VALUE 属性，此属性指定 DEGREE 选项的设置。此属性可以使用以下列表中的任何值: 0、1、2、3、5、7 和 9。*Performance Guide* 包含有关每个可以使用的设置的行为信息。

REOPT 请求: 可以使用 REOPT 一般请求元素来覆盖 REOPT 绑定选项的设置。REOPT 绑定选项影响包含参数标记或主变量的语句的优化。REOPT 一般请求元素由 reoptType 定义，如下所示:

XML Schema

```
<xs:complexType name="reoptType">
      <xs:attribute name="VALUE" use="required">
       <xs:simpleType>
            <xs:restriction base="xs:string">
                        <xs:enumeration value="ONCE"/>
                        <xs:enumeration value="ALWAYS"/>
            </xs:restriction>
   </xs:simpleType>          </xs:attribute>
</xs:complexType>
```

描述

REOPT 一般请求元素具有必需的 VALUE 属性，此属性指定 REOPT 选项的设置。此属性可以使用值 ONCE 或值 ALWAYS。值 ONCE 指示应对第一组主变量或参数标记值优化语句。值 ALWAYS 指示应对每组主变量或参数标记值优化语句。*Performance Guide* 包含有关每个可以使用的 REOPT 绑定选项设置的行为的详细信息。

RTS 请求:

可以使用 RTS 一般请求元素来启用或禁用实时统计信息收集。还可以使用它来限制实时统计信息收集所花的时间。对于某些查询或工作负载，可能需要禁用或限制实时统计信息收集所花的时间，以避免语句编译时的额外开销。

```
<!--*****************************************************************************************--> \
<!-- RTS general request element to enable, disable or provide a time budget for             --> \
<!-- real-time statistics collection.                                                        --> \
<!-- OPTION attribute allows enabling or disabling real-time statistics.                     --> \
<!-- TIME attribute provides a time budget in milliseconds for real-time statistics collection.--> \
<!--***************************************************************************************** --> \
<xs:complexType name="rtsType">
         <xs:attribute name="OPTION" type="optionType" use="optional" default="ENABLE"/>
   <xs:attribute name="TIME" type="xs:nonNegativeInteger" use="optional"/>
   </xs:complexType>
```

描述

RTS 一般请求元素具有两个可选属性。

- OPTION 属性用于启用或禁用实时统计信息收集。它可以采用值 ENABLE 或 DISABLE。如果未指定选项，那么 ENABLE 是缺省值。
- TIME 属性指定在语句编译时收集单个语句的实时统计信息所用的最长时间（以毫秒计）。

如果对 OPTION 属性指定了 ENABLE，那么必须通过相应的配置参数启用自动统计信息收集和实时统计信息。否则，将不会应用优化准则，您将接收到警告消息 SQL0437W（原因码 13）。

例如，下列 RTS 请求启用实时统计信息收集并将实时统计信息收集时间限制为 3.5 秒。

```
<RTS OPTION="ENABLE" TIME="350" />
```

查询重写准则的 XML 模式: 查询重写优化准则影响查询重写优化阶段。它包括 *rewriteRequest* 组。

rewriteRequest 组定义有效重写请求元素选项的集合。重写请求包括 *INLIST2JOIN*、*SUBQ2JOIN*、*NOTEX2AJ* 和 *NOTIN2AJ*:

XML Schema

```
<xs:group name="rewriteRequest">
```

```
      <xs:sequence>
    <xs:element name="INLIST2JOIN" type="inListToJoinType" minOccurs="0"/>
    <xs:element name="SUBQ2JOIN"   type="subqueryToJoinType" minOccurs="0"/>
    <xs:element name="NOTEX2AJ"    type="notExistsToAntiJoinType" minOccurs="0"/>
    <xs:element name="NOTIN2AJ"    type="notInToAntiJoinType" minOccurs="0"/>
      </xs:sequence>
</xs:group>
```

描述

- INLIST2JOIN: 启用或禁用 IN-LIST 至连接的重写变换。可以将它用作语句级别准则，以启用或禁用查询中要进行 IN-LIST 至连接变换的所有 IN-LIST 谓词。还可以将它用作谓词级别准则，以启用或禁用要进行 IN-LIST 至连接变换的指定 IN-LIST 谓词。如果同时指定了语句级别和谓词级别准则，那么谓词准则将覆盖语句级别准则。
- SUBQ2JOIN: 启用或禁用子查询至连接的重写变换。只能将它用作语句级别准则，以启用或禁用查询中要进行子查询至连接重写变换的所有子查询。
- NOTEX2AJ: 启用或禁用 NOT-EXISTS 至反连接的重写变换。只能将它用作语句级别准则，以启用或禁用查询要进行 NOT-EXISTS 至反连接重写变换的所有 NOT-EXISTS 子查询。
- NOTIN2AJ: 启用或禁用 NOT-IN 至反连接的重写变换。只能将它用作语句级别准则，以启用或禁用查询中要进行 NOT-IN 至反连接重写变换的所有 NOT-IN 子查询。

IN-LIST 至连接的重写请求: 可以使用 *INLIST2JOIN* 请求元素来启用或禁用 IN-LIST 谓词至连接的重写变换。可以将此元素指定为语句级别准则或谓词级别准则。对于谓词级别准则，一个查询中只允许一个使用选项 ENABLE 的优化准则。

此元素由复杂类型 *inListToJoinType* 定义，如下所示:

XML Schema

```
<xs:complexType name="inListToJoinType">
   <xs:attribute name="OPTION" type="optionType" use="optional" default="ENABLE"/>
   <xs:attribute name="TABLE"  type="xs:string"  use="optional"/>
   <xs:attribute name="COLUMN" type="xs:string"  use="optional"/>
</xs:complexType>
```

描述

INLIST2JOIN 元素有三个可选属性，但是没有子元素。OPTION 属性的类型为 *optionType*，其值为“ENABLE”或“DISABLE”。如果未指定 OPTION 属性，那么缺省值为“ENABLE”。表名属性和列名属性用于指定涉及所指定的表和列的 IN-LIST 谓词。如果未指定表名属性和列名属性，或者表名属性和列名属性都指定为空字符串『』，那么将它作为语句级别准则处理。如果指定了表名或者同时指定了表名和列名，那么将它作为谓词级别准则处理。如果未指定表名或将它指定为空字符串『』，但指定了列名，那么将发出原因码为 13 的 SQLO437W，并且将忽略优化准则。

NOT-EXISTS 反连接重写请求: 可以使用 *NOTEX2AJ* 请求元素来启用或禁用 NOT-EXISTS 至反连接的重写变换。只能将此元素指定为语句级别准则。

此元素由复杂类型 *notExisitsToAntiJoinType* 定义，如下所示:

XML Schema

```
<xs:complexType name="notExistsToAntiJoinType">
   <xs:attribute name="OPTION" type="optionType" use="optional" default="ENABLE"/>
</xs:complexType>
```

描述

NOTEX2AJ 元素有一个可选属性，但是没有子元素。OPTION 属性的类型为 *optionType*，其值为“ENABLE”或“DISABLE”。如果未指定 OPTION 属性，那么缺省值为“ENABLE”。

NOT-IN 至反连接的重写请求: 可以使用 *NOTIN2AJ* 请求元素来启用或禁用 NOT-IN 至反连接的重写变换。只能将此元素指定为语句级别准则。

此元素由复杂类型 *notInToAntiJoinType* 定义，如下所示:

XML Schema

```
<xs:complexType name="notInToAntiJoinType">
   <xs:attribute name="OPTION" type="optionType" use="optional" default="ENABLE"/>
</xs:complexType>
```

描述

NOTIN2AJ 元素有一个可选属性，但是没有子元素。OPTION 属性的类型为 *optionType*，其值为“ENABLE”或“DISABLE”。如果未指定 OPTION 属性，那么缺省值为“ENABLE”。

子查询至连接的重写请求: 可以使用 *SUBQ2JOIN* 请求元素来启用或禁用子查询至连接的重写变换。只能将此元素指定为语句级别准则。

此元素由复杂类型 *subqueryToJoinType* 定义，如下所示:

XML Schema

```
<xs:complexType name="subqueryToJoinType">
   <xs:attribute name="OPTION" type="optionType" use="optional" default="ENABLE"/>
</xs:complexType>
```

描述

SUBQ2JOIN 元素有一个可选属性，但是没有子元素。OPTION 属性的类型为 *optionType*，其值为“ENABLE”或“DISABLE”。如果未指定 OPTION 属性，那么缺省值为“ENABLE”。

方案优化准则的 XML 模式: 方案优化准则可以分为访问请求和连接请求:

- 访问请求 – 访问请求指定表引用所需的访问方法。
- 连接请求 – 连接请求指定执行连接操作所需的方法和顺序。连接请求由其他访问请求或连接请求组成。

大多数访问请求选项与优化器的数据访问方法（如表扫描、索引扫描和列表预取）相对应。大多数可用的连接请求与优化器的连接方法（如嵌套循环连接、散列连接和合并连接）相对应。*Performance Guide* 中描述了这些方法。本节详细说明可用于影响方案优化的每个访问请求元素和连接请求元素。

访问请求: *accessRequest* 组定义有效访问请求元素选项的集合。访问请求指定用于满足语句中的表引用所需的方法。

XML Schema

```
<xs:group name="accessRequest">
      <xs:choice>
            <xs:element name="TBSCAN" type="tableScanType"/>
```

```
                <xs:element name="IXSCAN" type="indexScanType"/>
                <xs:element name="LPREFETCH" type="listPrefetchType"/>
                <xs:element name="IXAND" type="indexAndingType"/>
                <xs:element name="IXOR" type="indexOringType"/>
                <xs:element name="ACCESS" type="anyAccessType"/>
            </xs:choice>
    </xs:group>
```

描述

- TBSCAN、IXSCAN、LPREFETCH、IXAND 和 IXOR

  这些元素对应于 DB2 数据访问方法，只能将它们应用于语句中引用的本地表。它们不能引用昵称（远程表）或派生的表（子查询的结果）。

- ACCESS

  当连接顺序（而不是访问方法）是关注的主要问题时，将使用此元素。当目标表引用是派生的表时，必须使用此元素。优化器将根据成本选择目标表引用的访问方法。

访问类型: TBSCAN、IXSCAN、LPREFETCH、IXAND、IXOR 和 ACCESS 元素的一般方面由下面显示的抽象类型 accessType 定义。

XML Schema

```
<xs:complexType name="accessType" abstract="true">
   <xs:attribute name="TABLE"  type="xs:string"  use="optional"/>
   <xs:attribute name="TABID" type="xs:string" use="optional"/>
   <xs:attribute name="FIRST" type="xs:string" use="optional" fixed="TRUE"/>
</xs:complexType>
```

描述

所有访问请求元素都扩展复杂类型 *accessType*。每个这种元素都必须使用 *TABLE* 或 *TABID* 属性来指定目标表引用。第 332 页的『形成优化准则中的表引用』描述了如何根据访问请求元素形成正确的表引用。它们还可以指定可选的 *FIRST* 属性。如果指定了 FIRST 属性，那么此属性的值必须为 TRUE。将 FIRST 属性添加至访问请求元素表示您希望有这样一个执行计划，它指定访问请求的目标表作为相应 FROM 子句的连接顺序中的第一个表出现。每个 FROM 子句中只能有一个访问或连接请求指定 FIRST 属性。如果以同一 FROM 子句中的表为目标的多个访问或连接请求指定 FIRST 属性，那么将忽略除第一个请求外的所有请求，并且发出原因码为 13 的警告 SQL0437W。

任何访问请求: 可以使用 *ACCESS* 访问请求元素来请求优化器以基于成本的方式选择用于访问表的适当方法。此选项通常用于指定仅指示本地表与语句中的其他表的连接方式的访问请求。当引用派生的表时，必须使用此访问请求元素。派生的表是另一个子查询的结果。ACCESS 访问请求元素由复杂类型 *anyAccessType* 定义，如下所示:

XML Schema

```
<xs:complexType name="anyAccessType">
    <xs:complexContent>
            <xs:extension base="accessType"/>
    </xs:complexContent>
</xs:complexType>
```

描述

复杂类型 *anyAccessType* 是抽象类型 *accessType* 的简单扩展。未添加新元素或属性。

索引 AND 运算访问请求: 可以使用 *indexAnding* 访问请求元素来请求优化器使用索引 AND 运算数据访问方法来访问本地表。此元素由复杂类型 *indexAndingType* 定义，如下所示:

XML Schema

```
<xs:complexType name="indexAndingType">
   <xs:complexContent>
      <xs:extension base="accessType">
         <xs:sequence minOccurs="0">
            <xs:element name="INDEX" type="indexType" minOccurs="2" maxOccurs="unbounded"/>
         </xs:sequence>
         <xs:attribute name="INDEX" type="xs:string" use="optional"/>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
<xs:complexType name="indexType">
   <xs:attribute name="IXNAME" type="xs:string" use="optional"/>
</xs:complexType>
```

描述

复杂类型 *indexAndingType* 通过添加可选的 *INDEX* 属性和可选的 INDEX 子元素来扩展抽象类型 *localAccessType*。INDEX 属性可用于指定索引 AND 操作中使用的第一个索引。如果使用 INDEX 属性，那么优化器将以基于成本的方式选择其他索引和访问顺序。INDEX 元素可用于指定准确的索引和访问顺序集。INDEX 子元素的显示顺序指示执行各个索引扫描时应采用的顺序。指定 INDEX 子元素将取代指定 INDEX 属性。

- 如果未指定索引，那么优化器将以基于成本的方式选择索引和访问顺序。
- 如果通过使用属性或子元素指定了索引，那么这些索引必须标识对 TABLE 或 TABID 属性所标识的表定义的索引。
- 如果未对表定义索引，那么将忽略访问请求并发出原因码为 13 的警告 SQL0437W。

在索引 AND 运算访问请求中，块索引必须出现在记录索引前面。如果未满足此要求，那么会发出原因码为 13 的警告 SQL0437W。索引 AND 运算访问方法至少需要一个可以对每个索引建立索引的谓词。如果由于必需的谓词不存在而导致索引 AND 运算不合格，那么将忽略访问请求并发出原因码为 13 的警告 SQL0437W。如果索引 AND 运算数据访问方法不在语句的有效搜索空间中，那么将忽略访问请求并发出原因码为 13 的警告 SQL0437W。

索引 OR 运算访问请求: 可以使用 *IXOR* 访问请求元素来请求优化器使用索引 OR 运算数据访问方法来访问本地表。此元素由复杂类型 *indexOringType* 定义，如下所示:

XML Schema

```
<xs:complexType name="indexOringType">
    <xs:complexContent>
            <xs:extension base="accessType"/>
    </xs:complexContent>
</xs:complexType>
```

描述

复杂类型 *indexOringType* 是抽象类型 *accessType* 的简单扩展。未添加新元素或属性。如果索引 OR 运算访问方法不在语句的有效搜索空间中，那么将忽略访问请求并发出原因码为 13 的警告 SQL0437W。优化器将以基于成本的方式选择索引 OR 操作中使用的谓词和索引。索引 OR 运算访问方法至少需要一个可以建立索引的 IN 谓词或带有逻辑 OR 操作可以建立索引和连接的项的谓词。如果由于必需的谓词或索引不存在而导致索引 OR 运算不合格，那么将忽略请求并发出原因码为 13 的警告 SQL0437W。

索引扫描访问请求: 可以使用 *IXSCAN* 访问请求元素来请求优化器使用索引扫描来访问本地表。此元素由复杂类型 *indexScanType* 定义，如下所示:

XML Schema

```
<xs:complexType name="indexScanType">
   <xs:complexContent>
      <xs:extension base="accessType">
         <xs:attribute name="INDEX" type="xs:string" use="optional"/>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

描述

复杂类型 *indexScanType* 通过添加可选的 *INDEX* 属性来扩展抽象类型 accessType。INDEX 属性指定要用于访问表的索引的名称。

- 如果索引扫描访问方法不在语句的有效搜索空间中，那么将忽略访问请求并发出原因码为 13 的警告 SQL0437W。
- 如果指定了 INDEX 属性，那么它必须标识对 TABLE 或 TABID 属性所标识的表定义的索引。
- 如果索引不存在，那么将忽略访问请求并发出原因码为 13 的警告 SQL0437W。
- 如果未指定 INDEX 属性，那么优化器将以基于成本的方式选择索引。
- 如果未对目标表定义索引，那么将忽略访问请求并发出原因码为 13 的警告 SQL0437W。

列表预取访问请求: 可以使用 *listPrefetch* 元素来请求优化器使用列表预取索引扫描来访问本地表。此元素由复杂类型 *listPrefetchType* 定义，如下所示:

XML Schema

```
<xs:complexType name="listPrefetchType">
  <xs:complexContent>
     <xs:extension base="accessType">
        <xs:attribute name="INDEX" type="xs:string" use="optional"/>
     </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

描述

复杂类型 *listPrefetchType* 通过添加可选的 *INDEX* 属性来扩展抽象类型 *accessType*。INDEX 属性指定要用于访问表的索引的名称。

- 如果列表预取访问方法不在语句的有效搜索空间中，那么将忽略访问请求并发出原因码为 13 的警告 SQL0437W。
- 列表预取访问方法至少需要一个可以建立索引的谓词。
- 如果由于必需的谓词不存在而导致列表预取不合格，那么将忽略访问请求并发出原因码为 13 的警告 SQL0437W。
- 如果指定了 INDEX 属性，那么它必须标识对 TABLE 或 TABID 属性所指定的表定义的索引。
- 如果索引不存在，那么将忽略访问请求并发出原因码为 13 的警告 SQL0437W。
- 如果未指定 INDEX 属性，那么优化器将以基于成本的方式选择索引。

- 如果未对目标表定义适当的索引，那么将忽略访问请求并发出原因码为 13 的警告 SQL0437W。

表扫描访问请求: 可以使用 *TBSCAN* 访问请求元素来请求优化器使用顺序表扫描来访问本地表。此元素由复杂类型 *tableScanType* 定义，如下所示:

```
XML Schema

      <xs:complexType name="tableScanType">
          <xs:complexContent>
                  <xs:extension base="accessType"/>
          </xs:complexContent>
      </xs:complexType>
```

描述

复杂类型 *tableScanType* 是上面所显示的抽象类型 *accessType* 的简单扩展。未添加新元素或属性。如果表扫描访问方法不在语句的有效搜索空间中，那么将忽略访问请求并发出原因码为 13 的警告 SQL0437W。

连接请求: *joinRequest* 组定义有效连接请求元素选项的集合。连接请求指定用于连接两个表所需的方法。

```
XML Schema

      <xs:group name="joinRequest">
            <xs:choice>
                  <xs:element name="NLJOIN" type="nestedLoopJoinType"/>
                  <xs:element name="HSJOIN" type="hashJoinType"/>
                  <xs:element name="MSJOIN" type="mergeJoinType"/>
                  <xs:element name="JOIN" type="anyJoinType"/>
            </xs:choice>
   </xs:group>
```

描述

NLJOIN、*MSJOIN* 和 *HSJOIN* 连接请求元素分别对应于嵌套循环连接方法、合并连接方法和散列连接方法。*Performance Guide* 更详细地描述了这些连接方法的合格要求和执行特征。*JOIN* 连接请求元素指示优化器可以选择连接方法。当指定特定连接顺序是关注的主要问题时，可以使用此选项。

所有连接请求元素都包含两个子元素，它们表示连接操作的输入表。连接请求还可以指定可选的 FIRST 属性。

任何连接请求: 可以使用 *JOIN* 连接请求元素来请求优化器采用它选择的任何连接方法以特定顺序连接两个表。这两个输入表可以是访问请求子元素所指定的本地表或派生的表，也可以是连接请求子元素所指定的连接操作的结果。*JOIN* 连接请求元素由复杂类型 *anyJoinType* 定义，如下所示:

```
XML Schema

      <xs:complexType name="anyJoinType">
          <xs:complexContent>
                  <xs:extension base="joinType"/>
          </xs:complexContent>
      </xs:complexType>
```

描述

复杂类型 *anyJoinType* 是抽象类型 *joinType* 的简单扩展。未添加新元素或属性。

散列连接请求: 可以使用 *HSJOIN* 连接请求元素来请求优化器采用散列连接方法连接两个表。这两个输入表可以是访问请求子元素所指定的本地表或派生的表，也可以是连接请求子元素所指定的连接操作的结果。*HSJOIN* 连接请求元素由复杂类型 *hashJoinType* 定义，如下所示:

```
XML Schema

      <xs:complexType name="hashJoinType">
          <xs:complexContent>
                  <xs:extension base="joinType"/>
          </xs:complexContent>
      </xs:complexType>
```

描述

复杂类型 *hashJoinType* 是抽象类型 *joinType* 的简单扩展。未添加新元素或属性。如果散列连接方法不在语句的有效搜索空间中，那么将忽略连接请求并发出原因码为 13 的警告 SQL0437W。

合并连接请求: 可以使用 *MSJOIN* 连接请求元素来请求优化器采用合并连接方法连接两个表。这两个输入表可以是访问请求子元素所指定的本地表或派生的表，也可以是连接请求子元素所指定的连接操作的结果。*MSJOIN* 连接请求元素由复杂类型 *mergeJoinType* 定义，如下所示:

```
XML Schema

      <xs:complexType name="mergeJoinType">
          <xs:complexContent>
                  <xs:extension base="joinType"/>
          </xs:complexContent>
      </xs:complexType>
```

描述

复杂类型 *mergeJoinType* 是抽象类型 *joinType* 的简单扩展。未添加新元素或属性。如果合并连接方法不在语句的有效搜索空间中，那么将忽略连接请求并发出原因码为 13 的警告 SQL0437W。

嵌套循环连接请求: 可以使用 *NLJOIN* 连接请求元素来请求优化器采用嵌套循环连接方法连接两个表。这两个输入表可以是访问请求子元素所指定的本地表或派生的表，也可以是连接请求子元素所指定的连接操作的结果。*NLJOIN* 连接请求元素由复杂类型 *nestedLoopJoinType* 定义，如下所示:

```
XML Schema

      <xs:complexType name="nestedLoopJoinType">
          <xs:complexContent>
                  <xs:extension base="joinType"/>
          </xs:complexContent>
      </xs:complexType>
```

描述

复杂类型 *nestedLoopJoinType* 是抽象类型 *joinType* 的简单扩展。未添加新元素或属性。如果嵌套循环连接方法不在语句的有效搜索空间中，那么将忽略连接请求并发出原因码为 13 的警告 SQL0437W。

连接请求的类型: 所有连接请求元素的公共方面都由抽象类型 *joinType* 定义，如下所示:

XML Schema

```
<xs:complexType name="joinType" abstract="true">
   <xs:choice minOccurs="2" maxOccurs="2">
      <xs:group ref="accessRequest"/>
      <xs:group ref="joinRequest"/>
   </xs:choice>
   <xs:attribute name="FIRST" type="xs:string" use="optional" fixed="TRUE"/>
</xs:complexType>
```

描述

扩展 *joinType* 的连接请求元素必须正好有两个子元素。一个子元素可以是从组 *accessRequest* 中选择的访问请求元素，另一个子元素可以是从组 *joinRequest* 中选择的连接请求元素。连接请求中出现的第一个子元素指定连接操作的外部表，第二个元素指定内部表。如果指定了 FIRST 属性，那么此属性的值必须为 TRUE。将 FIRST 属性添加至连接请求元素表示您希望有这样一个执行计划，其中连接请求的目标表是相应 FROM 子句的连接顺序中的外部表。每个 FROM 子句中只能有一个访问或连接请求指定 FIRST 属性。如果以同一 FROM 子句中的表为目标的多个访问或连接请求指定 FIRST 属性，那么将忽略除初始请求外的所有请求，并且发出原因码为 13 的警告 SQL0437W。

SYSTOOLS.OPT_PROFILE 表

SYSTOOLS.OPT_PROFILE 表包含所有优化概要文件。可以采用两种方法来创建该表:

- 调用 SYSINSTALLOBJECTS 过程:

```
db2 "call sysinstallobjects('opt_profiles', 'c', '', '')"
```

- 发出 CREATE TABLE 命令:

```
CREATE TABLE SYSTOOLS.OPT_PROFILE (
                        SCHEMA  VARCHAR(128) NOT NULL,
                        NAME    VARCHAR(128) NOT NULL,
                        PROFILE BLOB (2M)    NOT NULL,
                        PRIMARY KEY ( SCHEMA, NAME )
      );
```

按如下所示定义 SYSTOOLS.OPT_PROFILE 表中的列:

SCHEMA

- 指定优化概要文件的模式限定符。模式名最多可以包含 30 个字母数字或下划线字符，但它定义为如上所示的 VARCHAR(128)。

名称

- 指定优化概要文件的基本名称。此名称最多可以包含 128 个字母数字或下划线字符。

PROFILE

- 用于定义优化概要文件的 XML 文档。

注:

1. 可以在几个不同的上下文中引用同一个优化概要文件。SYSTOOLS.OPT_PROFILE 表的 NAME 列和 SCHEMA 列一起指定由两部分组成的优化概要文件名。PROFILE 列包含用于定义优化概要文件的 XML 文档。
2. 表空间或分区组不受任何限制。

用于清空优化概要文件高速缓存的触发器: 应创建下列 SQL 过程和触发器，以确保在更新或删除 SYSTOOLS.OPT_PROFILE 表中的条目时会自动清空概要文件高速缓存。

```
CREATE PROCEDURE SYSTOOLS.OPT_FLUSH_CACHE( IN SCHEMA VARCHAR(128),
                                           IN NAME VARCHAR(128) )
LANGUAGE SQL
  MODIFIES SQL DATA
  BEGIN ATOMIC
     -- FLUSH stmt (33) + quoted schema (130) + dot (1) + quoted name (130) = 294
     DECLARE FSTMT VARCHAR(294) DEFAULT 'FLUSH OPTIMIZATION PROFILE CACHE '; --

     IF NAME IS NOT NULL THEN
         IF SCHEMA IS NOT NULL THEN
     SET FSTMT = FSTMT || '"' || SCHEMA || '".'; --
 END IF; --

   SET FSTMT = FSTMT || '"' || NAME || '"'; --

        EXECUTE IMMEDIATE FSTMT; --
     END IF; --
   END;

   CREATE TRIGGER SYSTOOLS.OPT_PROFILE_UTRIG AFTER UPDATE ON SYSTOOLS.OPT_PROFILE
   REFERENCING OLD AS O
FOR EACH ROW          CALL SYSTOOLS.OPT_FLUSH_CACHE( O.SCHEMA, O.NAME );

   CREATE TRIGGER SYSTOOLS.OPT_PROFILE_DTRIG AFTER DELETE ON SYSTOOLS.OPT_PROFILE
   REFERENCING OLD AS O
FOR EACH ROW          CALL SYSTOOLS.OPT_FLUSH_CACHE( O.SCHEMA, O.NAME );
```

管理 SYSTOOLS.OPT_PROFILE 表: 优化概要文件必须与唯一的模式限定名关联，并且必须存储在 SYSTOOLS.OPT_PROFILE 表中。可以使用 LOAD、IMPORT 和 EXPORT 命令来管理该表中的文件。例如，可以在任何 DB2 客户机中使用 IMPORT 命令来将文件中的数据插入或更新到 SYSTOOLS.OPT_PROFILE 表中。可以使用 EXPORT 命令将 OPT_PROFILE 表中的概要文件检索到文件中。

以下是将单独的输入文件中的三个新行插入到 SYSTOOLS.OPT_PROFILE 表中的示例。假定文件位于当前目录中。

1. 在单独的行中创建一个输入文件（例如，profiledata），该文件中的每行包含模式、名称和文件名:

   ```
   "ROBERT","PROF1","ROBERT.PROF1.xml"
   "ROBERT","PROF2","ROBERT.PROF2.xml"
   "DAVID", "PROF1","DAVID.PROF1.xml"
   ```

2. 执行 IMPORT 命令:

   ```
   IMPORT FROM profiledata OF DEL MODIFIED BY LOBSINFILE INSERT INTO SYSTOOLS.OPT_PROFILE
   ```

要更新现有行，可以如上面那样先删除行，然后再将其插入，或者将 INSERT_UPDATE 选项与 IMPORT 配合使用:

```
IMPORT FROM profiledata OF DEL MODIFIED BY LOBSINFILE
        INSERT_UPDATE INTO SYSTOOLS.OPT_PROFILE
```

要将 ROBERT.PROF1 概要文件检索到 ROBERT.PROF1.xml 中（假定此概要文件小于 32,700 个字节）:

```
EXPORT TO ROBERT.PROF1.xml OF DEL SELECT PROFILE FROM SYSTOOLS.OPT_PROFILE
       WHERE SCHEMA='ROBERT' AND NAME='PROF1'
```

要导出大于 32,700 个字节的数据，或者要获取更多信息，请参阅 *Command Reference* 中的 EXPORT 命令的文档。

影响查询优化的配置参数

有一些配置参数影响 SQL 或 XQuery 编译器选择的访问方案。其中的许多参数都适合单一分区数据库环境，而某些参数仅适合分区数据库环境。在同类（硬件相同）的分区数据库环境中，用于每个参数的值应该在所有数据库分区上是相同的。

注: 当动态更改配置参数时，优化器可能会由于程序包高速缓存中的旧访问方案而不立即读取已更改的参数值。要复位程序包高速缓存，执行 FLUSH PACKAGE CACHE 命令。

在联合系统中，如果大多数查询都将访问昵称，那么在更改环境之前评估您发送的查询类型。例如，在联合数据库中，缓冲池不高速缓存数据源中的页，这些数据源是 DBMS 和联合系统中的数据。因此，增大缓冲区的大小并不保证当优化器为包含昵称的查询选择访问方案时将考虑其他访问方案备用。但是，优化器可以决定数据源表的本地具体化是否是成本最低的方法，或者是否是排序操作的必需步骤。在这种情况下，增加可用的资源可能会提高性能。

下列配置参数或因子影响 SQL 或 XQuery 编译器选择的访问方案:

- 当创建或改变缓冲池时指定的缓冲池大小

  当优化器选择访问方案时，优化器要考虑将页从磁盘访存至缓冲池的 I/O 成本并估计满足查询所需要的 I/O 次数。估计包括预测缓冲池使用情况，因为不需要其他的物理 I/O 来读取已在缓冲池中的页中的行。

  优化器考虑 SYSCAT.BUFFERPOOLS 系统目录表以及在分区数据库环境中的 SYSCAT.BUFFERPOOLDBPARTITIONS 系统目录表中的 *npages* 列的值。

  读取表的 I/O 成本可影响:

  – 如何连接两个表
  – 是否将使用非集群索引来读取数据
- 缺省度（dft_degree）

  dft_degree 配置参数通过为 CURRENT DEGREE 专用寄存器和 DEGREE 绑定选项提供缺省值来指定并行性。值 1 表示无分区内并行性。值 -1 表示优化器根据处理器数目和查询类型来确定分区内并行度。

  注: 除非通过设置 *intra_parallel* 数据库管理器配置参数来启用内部并行处理，否则不会进行内部并行处理。
- 缺省查询优化级别（dft_queryopt）

  虽然在编译 SQL 或 XQuery 查询时可以指定查询优化级别，但还可以设置缺省查询优化级别。
- 活动应用程序平均数（avg_appls）

  优化器使用 *avg_appls* 参数来帮助估计运行时期间可用于所选访问方案的缓冲池的个数。较高的此参数值会影响优化器，使它选择在缓冲池的使用情况方面更节省的访问方案。如果指定值 1，那么优化器认为整个缓冲池将可用于应用程序。
- 排序堆大小（sortheap）

如果要排序的行占用的空间超过排序堆中可用的空间，那么执行几遍排序，每一遍都要对整个行集的某个子集排序。每遍排序的结果存储在缓冲池中的一个系统临时表内，可以将该表写入磁盘。当所有排序都完成时，将这些已排序的子集合并到单个排序的行集。如果排序不需要系统临时表来存储最终的已排序的数据列表，那么可认为该排序是“管道”排序。即，可以按单一的顺序访问方式来读取排序的结果。管道排序的性能优于非管道排序，所以应尽可能使用管道排序。

当选择访问方案时，优化器要通过下列操作来估计排序操作的成本，包括评估是否可使用管道传送排序:

– 估计要排序的数据量

– 查看 *sortheap* 参数，以确定是否有足够的空间通过管道传送排序。

- 锁定列表的最大存储器（locklist）和升级前锁定列表的最大百分比（maxlocks）

  当隔离级别是**可重复读（RR）**时，优化器考虑 *locklist* 和 *maxlocks* 参数的值，以确定是否可以将行级别锁定升级到表级别锁定。如果优化器估计将对表访问发生锁定升级，那么它为访问方案选择一个表级别锁定，这样就不会在查询执行期间产生锁定升级的开销。

- CPU 速度（cpuspeed）

  优化器使用 CPU 速度来估计执行特定操作的成本。CPU 成本估计和各种 I/O 成本估计有助于选择查询的最佳访问方案。

  一台机器的 CPU 速度可显著影响所选的访问方案。当安装或迁移数据库时，会自动将此配置参数设置为一个适当的值。除非您要在测试系统上为生产环境建立模型或评估硬件更改的影响，否则不应调整此参数。使用此参数为不同的硬件环境建立模型可以使您找出可为该环境选择的访问方案。要让数据库管理器重新计算此自动配置参数的值，将它设置为 -1。

- 语句堆大小（stmtheap）

  尽管语句堆的大小不影响优化器选择不同的访问路径，但是，它会影响对复杂的 SQL 或 XQuery 语句执行的优化量。

  如果未将 *stmtheap* 参数设置得足够大，您可能接收到警告，指示无足够可用内存来处理语句。例如，SQLCODE +437（SQLSTATE 01602）可能指示用于编译语句的优化量小于您请求的量。

- 通信带宽（comm_bandwidth）

  优化器使用通信带宽来确定访问路径。优化器使用此参数中的值来估计在分区数据库环境的各数据库分区服务器之间执行特定操作的成本。

- 应用程序堆大小（applheapsz）

  大模式要求应用程序堆中有足够的空间。

数据库分区组对查询优化的影响

在分区数据库环境中，优化器识别表的并置，并在确定查询的最佳访问方案时使用此并置。如果在连接查询中频繁涉及某些表，那么应该将那些表划分到分区数据库环境中的各数据库分区中，以便要连接的每个表中的行位于同一数据库分区中。在执行连

接操作期间，两个已连接的表中数据的并置会防止将数据从一个数据库分区移动至另一个数据库分区。将两个表置于同一数据库分区组中以确保并置两个表中的数据。

在分区数据库环境中，根据表的大小，将数据分布在多个数据库分区上可缩短执行查询的估计时间（或成本）。表的数目、表的大小、那些表中数据的位置以及查询类型（例如，是否需要连接）都会影响查询的成本。

多个谓词的列相关

应用程序可以包含使用连接构造的查询，以使多个连接谓词连接两个表。因为我们经常需要通过查询来确定不同表中相似的相关列之间的关系。

例如，请考虑一个制造商，他利用各种颜色、弹性和质量的原料来生产产品。制成品与所用的原料具有相同的颜色和弹性。制造商发出查询:

```
    SELECT PRODUCT.NAME, RAWMATERIAL.QUALITY
FROM PRODUCT, RAWMATERIAL
       WHERE PRODUCT.COLOR       =  RAWMATERIAL.COLOR
                 AND PRODUCT.ELASTICITY    =  RAWMATERIAL.ELASTICITY
```

此查询返回所有产品的名称和原料质量。有两个连接谓词:

```
  PRODUCT.COLOR       =  RAWMATERIAL.COLOR
    PRODUCT.ELASTICITY  =  RAWMATERIAL.ELASTICITY
```

当优化器选择一个方案来执行此查询时，它要计算每个谓词是如何进行选择的。优化器假定它们是独立的，即，对于每种颜色存在各种弹性，反过来，对于每一级别的弹性，存在每种颜色的原料。然后，优化器使用每个表的目录统计信息（基于弹性级别数和不同颜色数）来估算谓词对的总体选择性。例如，根据此估算，它可能选择嵌套循环连接而不是合并连接，反之亦然。

但是，有可能这两个谓词并不是独立的。例如，有可能只有几种颜色的高弹性材料可用，且只有其他几种颜色（与弹性材料的颜色不同）的低弹性材料可用。于是，谓词的组合选择性将排除较少的行，因此，查询将返回更多行。请考虑极端的情况，即对于每种颜色，只有一种级别的弹性，反之亦然。现在，两个谓词中任何一个在逻辑上可以完全省略，因为它被另一个谓词隐含。优化器可能不再选择最佳方案。例如，尽管合并连接的运行速度会更快，但优化器可能选择嵌套循环连接方案。

如果对那些列定义了索引，或者如果对适当的列收集并维护组列统计信息，那么 DB2 优化器就会尝试检测并补偿连接谓词的依赖性。

例如，在以上弹性示例中，您可以定义下列一个或两个索引:

```
  IX1 PRODUCT.COLOR, PRODUCT.ELASTICITY
```

或者

```
   IX2 RAWMATERIAL.COLOR, RAWMATERIAL.ELASTICITY
```

的唯一索引。

对于检测依赖性的优化器，此索引的非包含列必须是唯一的相关列。索引还可以包括包含列以允许只扫描索引。如果在连接谓词中有两个以上的相关列，那么应确保定义索引来涵盖所有这些相关列。

要让优化器考虑索引键基数以检测依赖性，一个必须符合的条件是，对于同一个表中的每一列，每个列中的单值数都必须较高。例如，假定按以上方式定义了 IX1 和 IX2。如果 PRODUCT.COLOR 中的单值数少于 RAWMATERIAL.COLOR 中的单值数，并且 PRODUCT.ELASTICITY 中的单值数少于 RAWMATERIAL.ELASTICITY 中的单值数，那么优化器将使用 IX2 来检测依赖性。通过比较列基数，使得 PRODUCT 列中的单值有可能（但不保证）包括在 RAWMATERIAL 列的单值中。为了使一个域更有可能采用另一个域，优化器还可能会这些索引列的 HIGH2KEY AND LOW2KEY 统计信息进行比较。

在创建适当的索引后，确保表的统计信息是准确并且最新的。

优化器使用唯一索引统计信息表的 FIRSTnKEYCARD 和 FULLKEYCARD 列中的信息来检测关联情况，并动态调整关联谓词的组合选择性，因而获得对连接大小和成本的更准确的估计。

另外，可以收集一组列的列组统计信息。在以上的灵活性示例中，您可以收集 PRODUCT.COLOR、PRODUCT.ELASTICITY 和/或 RAWMATERIAL.COLOR、RAWMATERIAL.ELASTCITY 这些列的统计信息。

列组统计信息是使用 RUNSTATS 的“ON COLUMNS”选项收集的。例如，要收集 PRODUCT.COLOR 和 PRODUCT.ELASTICITY 的列组统计信息，可发出以下 RUNSTATS 命令:

```
RUNSTATS ON TABLE product ON COLUMNS ((color, elasticity))
```

简单等价谓词的依赖性

除 JOIN 谓词相关以外，优化器还使用类型为 COL = 的简单等价谓词来管理依赖性。例如，请考虑不同轿车类型的表，每一种轿车都有 MAKE（即制造商）、MODEL、YEAR、COLOR 和 STYLE（如私家车、旅行车、赛车）。因为几乎每个制造商每年生产的各种型号和式样的轿车都具有相同的标准颜色，所以 COLOR 的谓词很可能独立于 MAKE、MODEL、STYLE 或 YEAR 的谓词。但是，因为只有一个轿车制造商才会生产具有特定名称的型号，所以谓词 MAKE 和 MODEL 当然不是独立的。同一个型号名由两个或两个以上的轿车制造商使用几乎是不可能的，当然轿车制造商也不想这样做。

如果 MAKE 和 MODEL 这两列上存在索引或者收集了列组统计信息，那么优化器使用关于该索引或这两列的统计信息来确定组合的单值数，并调整两列之间的依赖性的选择性或基数估计。如果这些谓词是本地等价谓词，优化器进行调整时就不需要唯一索引。

使用索引和列组统计信息来计算分组键卡

当查询要求以某种方式对数据进行分组时，优化器需要计算单值分组数，即*分组键卡*。分组需求可能是由诸如 GROUP BY 或 DISTINCT 之类的操作引起的。

请考虑以下查询:

```
SELECT DEPTNO, YEARS, AVG(SALARY)
       FROM EMPLOYEE  GROUP BY DEPTNO, MGR, YEAR_HIRED
```

如果没有任何索引或列组统计信息，优化器估算的分组数（在本例中还包括返回的行数）将是 DEPTNO、MGR 和 YEAR_HIRED 的单值数乘积。这个估算假定分组键列是

独立的。但是，如果每个经理都刚好管理一个部门，这个假定就是错误的。并且，每个部门一般不会每年都聘请职员。因此，估算的 DEPTNO、MGR 和 YEAR_HIRED 单值乘积会超出实际单值组数。

现在，请考虑具有下列定义的索引:

```
INDEX IX1: DEPTNO, MGR, YEAR_HIRED
```

在本例中，IX1 中的 FULLKEYCARD 向优化器提供了以上查询的精确单值分组数。

请考虑另一个索引定义:

```
INDEX IX2: DEPTNO, MGR, YEAR_HIRED, COMM
```

由于 IX2 的 FIRST3KEYCARD 指示了 (DEPTNO,MGR,YEAR_HIRED) 的单值组数，所以它还有助于计算分组键卡。

除索引统计信息（FIRST2KEYCARD、FIRST3KEYCARD、FIRST4KEYCARD 和 FULLKEYCARD）以外，优化器还可以利用列组统计信息来以类似的方式计算分组键卡。对 DEPTNO、MGR 和 YEAR_HIRED 收集的列组统计信息与上面的 IX1 和 IX2 有相同的优点:

```
RUNSTATS ON TABLE EMPLOYEE ON COLUMNS ((DEPTNO, MGR, YEAR_HIRED))
```

注意，如果分组键包含 5 个或更多的列，那么最好收集列组统计信息。这是因为，RUNSTATS 仅对前 4 列和任何给定索引的完全索引键列收集统计信息。

统计视图

统计信息视图允许优化器计算更准确的基数估计。基数估计是优化器使用统计信息来确定在应用谓词或执行聚集之后部分查询结果大小的过程。基数估计的准确性取决于谓词和可用的统计信息。统计信息可用来表示某一列中数据值的分布，当数据值未均匀分布时它可以提高基数估计的准确性。统计信息还可用来表示一组列中单值的数目，当这些列在统计上相关时它可以提高基数估计的准确性。但是，十分常见的一种情况是，这些统计信息可能无法表示更复杂的关系，例如，涉及到相关属性和变形属性的谓词或聚集的过滤效果（例如，*make* = 'Honda' AND *model* = 'Accord'）、与表达式结果进行比较（例如，*price* > MSRP + Dealer_markup）、跨多个表的关系（例如，*product.name* = 'Alloy wheels' and *product.key* = *sales.product_key*）或者除了涉及到独立属性和简单比较操作的谓词或聚集之外的任何对象。

统计信息视图是具有相关统计信息的视图，可用来提高对视图定义与查询定义重叠的查询进行基数估计的准确性。这是一项强大的功能，因为它能为优化器提供准确的统计信息，来确定使用一组涉及一个或多个表的复杂谓词（可能相关）的查询的基数估计。

不必在统计信息视图优化的查询中直接引用统计信息；在大多数情况下，当视图定义与查询定义重叠时，可以利用视图的统计信息。要利用此新功能，必须使用 ALTER VIEW 语句来启用视图以进行优化，并且必须用该视图上的统计信息填充系统目录表。

使用统计信息视图

在将视图的统计信息用于优化查询之前，必须启用该视图以进行优化。启用以进行优化的视图是*统计信息视图*。非统计信息视图不能进行优化。术语*常规视图*用于表示不能进行优化的视图。第一次创建视图时，不能进行优化。

- 要启用视图以进行优化，您必须拥有对视图以及定义该视图的表的 ALTER 特权。
- 要对视图调用 RUNSTATS，您必须拥有下列其中一项权限或特权:
  - SYSADM
  - SYSCTRL
  - SYSMAINT
  - DBADM
  - 对视图的 CONTROL 特权

  此外，您还必须具有能访问视图中的行的适当特权。特别是对于视图定义中引用的每个表、视图或昵称，您必须具有下列其中一项特权:
  - SYSADM
  - DBADM
  - CONTROL
  - SELECT

如果下列任何条件成立，那么不能启用视图来进行优化:

- 视图直接或间接引用 MQT。（MQT 或统计信息视图可以引用统计信息视图。）
- 它是不可用视图。
- 它是带类型视图。
- 在同一 ALTER VIEW 语句中有对另一个视图的修改。

如果进行改变以启用优化的视图的定义符合下列任何条件，那么 ALTER VIEW ENABLE OPTIMIZATION 将成功，并出现警告消息，但优化器不会使用该视图的统计信息:

- 它包含聚集或单值操作。
- 它包含 union、except 或 intersect 操作。
- 它包含标量聚集（OLAP）函数。

1. 启用视图以进行优化。

   可以使用 ALTER VIEW 语句中的新 ENABLE OPTIMIZATION 子句来启用视图以进行优化。在以后可以使用 ALTER VIEW 语句中的 DISABLE OPTIMIZATION 子句对已启用来进行优化的视图禁用优化。例如，要启用视图 myview 以进行优化，请输入以下命令:

   ```
   ALTER VIEW myview ENABLE QUERY OPTIMIZATION
   ```

   启用以进行优化的视图在其相应 SYSTABLES 条目的 PROPERTY 列的字符位置 13 中有一个“Y”。禁用优化的视图在其相应 SYSTABLES 条目的 PROPERTY 列的字符位置 13 中为空白。
2. 执行 RUNSTATS。 例如，要收集视图 myview 的统计信息，请输入以下命令:

   ```
   RUNSTATS ON TABLE db2dba.myview
   ```

要使用行级别采样方式对 10% 的行进行统计信息采样，以收集视图统计信息（包括分布统计信息），那么请输入:

```
RUNSTATS ON TABLE db2dba.myview WITH DISTRIBUTION TABLESAMPLE BERNOULLI (10)
```

注: 在 DB2 版本 9.1 之前，对统计信息视图执行 RUNSTATS 将仅填写目录统计信息表以进行手动更新，但不收集任何统计信息。在 DB2 版本 9.1 中，对统计信息视图执行 RUNSTATS 将收集统计信息。这表示 RUNSTATS 的执行时间可能比以前要长很多，这取决于视图返回的数据量。

3. 更新视图统计信息会导致更改与视图定义重叠的查询方案。如果这些查询是静态 SQL 程序包的一部分，那么必须重新绑定这些程序包以利用新统计信息生成的方案。

与优化相关的视图统计信息

只有那些描述定义统计信息视图的查询中数据分布特征的统计信息（例如，CARD 和 COLCARD），优化器才会对它们进行优化。优化器可以收集并利用下列与视图记录相关的统计信息。

表统计信息（SYSCAT.TABLES 和 SYSSTAT.TABLES）

- CARD – 视图结果中的行数。

列统计信息（SYSCAT.COLUMNS 和 SYSSTAT.COLUMNS）

- COLCARD – 视图结果的一列中单值的数目
- AVGCOLLEN – 视图结果中列的平均长度
- HIGH2KEY – 视图结果的一列中第二大的值
- LOW2KEY – 视图结果的一列中第二小的值
- NUMNULLS – 视图结果列中 NULL 的数目
- SUB_COUNT – 视图结果列中子元素的平均数目
- SUB_DELIM_LENGTH – 分隔每个子元素的每个定界符的平均长度

列分布统计信息（SYSCAT.COLDIST 和 SYSSTAT.COLDIST）

- DISTCOUNT – 如果 TYPE 是 Q，那么为小于或等于 COLVALUE 统计信息的单值数目
- SEQNO – 按频率排列的序号，可帮助唯一标识该表中的行
- COLVALUE – 要收集其频率或分位数统计信息的数据值
- VALCOUNT – 数据值在视图列中出现的频率；对于分位数，它是小于或等于数据值（COLVALUE）的值的数目

可以收集不描述数据分布的统计信息（例如，NPAGES 和 FPAGES），但优化器将忽略此信息。

方案: 使用统计信息视图来提高基数估计的准确性

在数据仓库中，事实表中的信息经常改变，而维表数据则是静态的。这表示维属性数据可能与事实表属性数据一致或不一致。当前可用于优化器的传统基本表统计信息不允许优化器辨别各个表之间的关系。可以使用统计信息视图（和 MQT）上的列和表分布统计信息来为优化器提供更正这些类型的基数估计错误所需的信息。

考虑以下查询，它计算每年七月份高尔夫球棍的销售收入:

```
SELECT  sum(f.sales_price), d2.year
FROM  product d1, period d2, daily_sales f
WHERE  d1.prodkey = f.prodkey
        AND  d2.perkey = f.perkey
        AND  d1.item_desc = 'golf club'
        AND  d2.month = 'JUL'
GROUP BY  d2.year
```

如果优化器可以确定是涉及 PRODUCT 和 DAILY_SALES 的半连接还是涉及 PERIOD 和 DAILY_SALES 的半连接最具选择性，那么对于此查询来说，星型连接查询执行方案是一个不错的选择。为了生成有效的星型连接方案，优化器必须能够选择最具选择性的半连接作为索引 And（与）操作的外部支路。

数据仓库通常包含货架上已没有的产品的记录。这将导致连接后 PRODUCT 列的分布与连接前的分布极不相同。由于缺少更好的信息，优化器将只能根据基本表统计信息来确定本地谓词的选择性，因此优化器可能对谓词 *item_desc = 'golf club'* 的选择性过于乐观。

例如，如果高尔夫球棍占所生产产品的 1%，但占最近销售量的 20%，那么优化器很可能高估了 *item_desc = 'golf club'* 的选择性，因为没有描述连接后 *item_desc* 的分布的统计信息。如果十二个月的销售量都大致相同，那么谓词 *month = 'JUL'* 的选择性大约为 8%，因此错误估计谓词 *item_desc = 'golf club'* 的选择性将导致优化器将看似更具选择性的 PRODUCT 与 DAILY_SALES 之间的半连接作为星型连接方案的索引 And 操作的外部支路。

以下示例提供了有关如何设置统计信息视图来解决此类问题的逐步说明。

以下是典型数据仓库中的一个数据库，其中 STORE、CUSTOMER、PRODUCT、PROMOTION 和 PERIOD 是维表，而 DAILY_SALES 是事实表。

表 66. STORE（63 行）

列	storekey	store_number	city	state	district	...
属性	integer	char(2)	char(20)	char(5)	char(14)	...
	非空					
	主键					

表 67. CUSTOMER（1,000,000 行）

列	custkey	名称	address	age	gender	...
属性	integer	char(30)	char(40)	smallint	char(1)	...
	非空					
	主键					

表 68. PRODUCT（19,450 行）

列	prodkey	category	item_desc	price	cost	...
属性	integer	integer	char(30)	decimal(11)	decimal(11)	...
	非空					
	主键					

表 69. PROMOTION（35 行）

列	promokey	promotype	promodesc	promovalue	...
属性	integer	integer	char(30)	decimal(5)	...
	非空				
	主键				

表 70. PERIOD（2922 行）

列	perkey	calendar_date	month	period	year	...
属性	integer	date	char(3)	smallint	smallint	...
	非空					
	主键					

表 71. DAILY_SALES（754 069 426 行）

列	storekey	custkey	prodkey	promokey	perkey	sales_price	...
属性	integer	integer	integer	integer	integer	decimal(11)	...

假设公司的经理希望了解如果他们为回头客对相同产品进行打折，那么消费者是否会再次购买该产品。此外，假设仅对全国 18 家店中的商店“01”进行调查。表 72 显示了进行的不同类别的促销，用促销百分比表示。

表 72. PROMOTION（35 行）

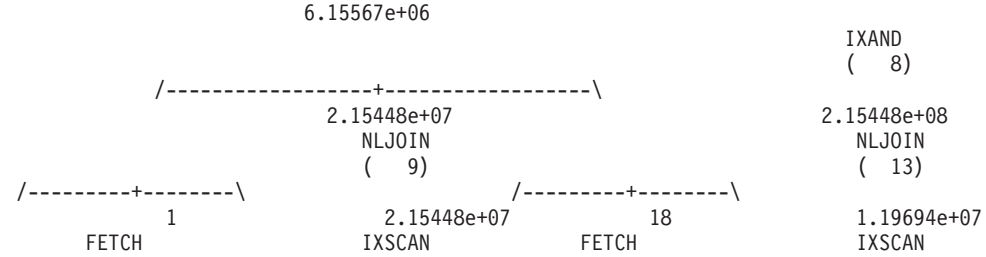
promotype	promodesc	COUNT(promotype)	促销百分比
1	回头客	1	2.86%
2	优惠券	15	42.86%
3	广告	5	14.29%
4	经理的特殊人员	3	8.57%
5	存储过多的商品	4	11.43%
6	通道末端展示	7	20.00%

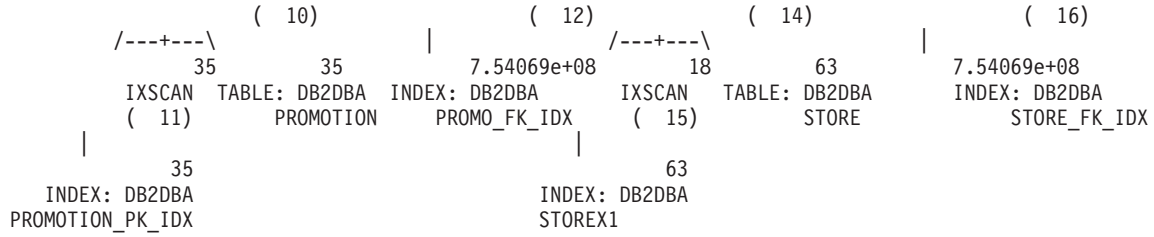
该表表明所提供的 35 种促销中只有 2.86% 来自回头客折扣（1/35 = 0.0286）。

然后，执行下列查询并返回计数 12,889,514:

```
SELECT  count(*)
FROM  store d1, promotion d2, daily_sales f
WHERE  d1.storekey = f.storekey
        AND  d2.promokey = f.promokey
        AND  d1.store_number = '01'
        AND  d2.promotype = 1
```

此查询是根据优化器生成的下列方案执行的。在此图的每个节点中，最上面的数字表示基数估计，第二行是运算符类型，括号中的数字是运算符标识。





在嵌套循环连接编号 9 中，优化器估计销售的产品中大约 2.86% 来自顾客再次以折扣价格购买相同产品（2.15448e+07 / 7.54069e+08 = 0.0286）。请注意，此百分比在连接 PROMOTION 表与 DAILY_SALES 表前后没有变化。表 73 总结了连接前后的基数估计及其百分比（过滤效果）。

表 73. 与 DAILY_SALES 连接前后的基数估计。

	连接前		连接后	
谓词	计数	合格行百分比	计数	合格行百分比
store_number = '01'	18	28.57%	2.15448e+08	28.57%
promotype=1	1	2.86%	2.15448e+07	2.86%

因为 *promotype = 1* 的可能性小于 *store_number = '01'* 的可能性，所以优化器选择 PROMOTION 与 DAILY_SALES 之间的半连接作为星型连接方案的索引 AND 操作的外部支路。这将得出使用促销类型 1 大约售出了 6,155,670 个产品的估计计数，但这不是正确的基数估计，其偏离因子为 2.09%（12,889,514/6,155,670 = 2.09）。

是什么导致优化器只能估计出符合两个谓词的实际记录数的一半？商店“01”大约占所有商店的 28.57%。如果其他商店的销售量大于商店“01”又会怎样（小于 28.57% 吗）？或者，如果商店“01”实际销售了大部分产品又会怎样（大于 28.57% 吗）？同样，表 73 中显示的使用促销类型 1 销售的 2.86% 的产品可能是误导。DAILY_SALES 中的实际百分比很可能与显示的数字不同。

可以使用统计信息视图来帮助优化器更正其估计并降低高估或低估的可能性。首先，需要创建两个统计信息视图，分别表示上述查询中的每个半连接。第一个统计信息视图提供所有日常销售的促销类型的分布。第二个统计信息视图表示所有日常销售的促销类型的分布。请注意，每个统计信息视图都能提供有关任何特定商店编号或促销类型的分布信息。在此示例中，使用 10% 的采样率来检索各个视图的 DAILY_SALES 中的记录，并将这些记录保存在全局临时表中。然后，查询这些表以收集更新两个统计信息视图所必需的统计信息。

1. 创建一个视图来表示 STORE 与 DAILY_SALES 的连接。

```
CREATE  view sv_store_dailysales as
      (SELECT  s.*
        FROM  store s, daily_sales ds
        WHERE  s.storekey = ds.storekey)
```

2. 创建一个视图来表示 PROMOTION 与 DAILY_SALES 的连接。

```
CREATE  view sv_promotion_dailysales as
      (SELECT  p.*
        FROM  promotion.p, daily_sales ds
        WHERE  p.promokey = ds.promokey)
```

3. 通过启用视图来进行查询优化，从而使该视图成为统计信息视图:

   •

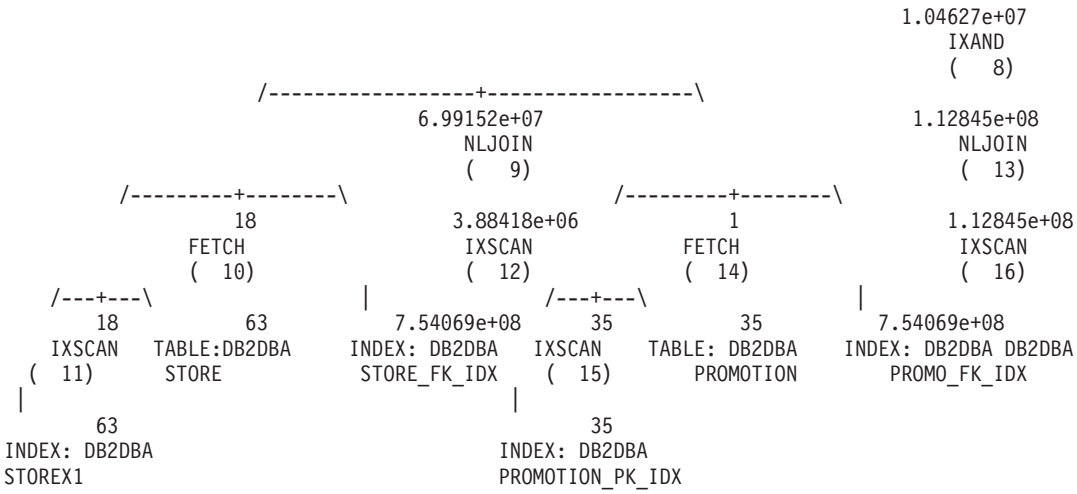
```
ALTER VIEW  sv_store_dailysales  ENABLE QUERY OPTIMIZATION
```

- ```
  ALTER VIEW  sv_promotion_dailysales  ENABLE QUERY OPTIMIZATION
  ```

4. 执行 RUNSTATS 以收集视图的统计信息:

   - ```
     RUNSTATS  on table db2dba.sv_store_dailysales  WITH DISTRIBUTION
     ```
   - ```
     RUNSTATS  on table db2dba.sv_promotion_dailysales WITH DISTRIBUTION
     ```

5. 再次运行查询，以便可以重新优化查询。在进行重新优化时，优化器将使 SV_STORE_DAILYSALES 和 SV_PROMOTION_DAILYSALES 与查询匹配，并使用视图统计信息来调整事实表与维表之间半连接的基数估计，这使得它反转在没有这些统计信息的情况下选择的半连接的原始顺序。新的方案如下所示:



请注意，此时 STORE 和 DAILY_SALES 之间的半连接是在索引 And 方案的外部支路执行的。这是因为两个统计信息视图实质上告知优化器谓词 store_number = '01' 过滤的行数比 promotype = 1 要多。表 74 总结了每个半连接的连接前后的基数估计及其百分比（过滤效果）。

请注意，此时 STORE 和 DAILY_SALES 之间的半连接是在索引 And 方案的外部支路执行的。这是因为两个统计信息视图实质上告知优化器谓词 store_number = '01' 过滤的行数比 promotype = 1 要多。这里，优化器估计大约售出了 10,462,700 个产品。此估计的偏差因子为 1.23（12,889,514 / 10,462,700 = 1.23），这比第 371 页的表 73 中得出的估计准确很多。表 74 总结了连接前后的基数估计和每个谓词的过滤效果。

表 74. 与 DAILY_SALES 连接前后的基数估计。

	连接前		连接后 *没有统计信息视图*		连接后 *具有统计信息视图*	
谓词	**计数**	**合格行百分比**	**计数**	**合格行百分比**	**计数**	**合格行百分比**
store_number = '01'	18	28.57%	2.15448e+08	28.57%	6.99152e+07	9.27%
promotype=1	1	2.86%	2.15448e+07	2.86%	1.12845e+08	14.96%

此时，优化器估计大约售出了 10,462,700 个产品。此估计的偏差因子为 1.23，这比没有统计信息视图时的估计要准确很多。
     ```

第 6 部分 附录

附录 A. DB2 技术信息概述

可以通过下列工具和方法获取 DB2 技术信息:

- DB2 信息中心
  - 主题（任务、概念和参考主题）
  - DB2 工具的帮助
  - 样本程序
  - 教程
- DB2 书籍
  - PDF 文件（可下载）
  - PDF 文件（在 DB2 PDF DVD 中）
  - 印刷版书籍
- 命令行帮助
  - 命令帮助
  - 消息帮助

注: DB2 信息中心主题的更新频率比 PDF 书籍或硬拷贝书籍的更新频率高。要获取最新信息，请安装可用的文档更新，或者参阅 ibm.com® 上的 DB2 信息中心。

可以在线访问 ibm.com 上的其他 DB2 技术信息，如技术说明、白皮书和 IBM Redbooks® 出版物。访问位于以下网址的 DB2 信息管理软件库站点: http://www.ibm.com/software/data/sw-library/。

文档反馈

我们非常重视您对 DB2 文档的反馈。如果您想就如何改善 DB2 文档提出建议，请将电子邮件发送至 db2docs@ca.ibm.com。 DB2 文档小组会阅读您的所有反馈，但不能直接答复您。请尽可能提供具体的示例，这样我们才能更好地了解您所关心的问题。如果您要提供有关具体主题或帮助文件的反馈，请加上标题和 URL。

请不要用以上电子邮件地址与 DB2 客户支持机构联系。 如果您遇到文档不能解决的 DB2 技术问题，请与您当地的 IBM 服务中心联系以获得帮助。

硬拷贝或 PDF 格式的 DB2 技术库

下列各表描述 IBM 出版物中心（网址为 www.ibm.com/shop/publications/order）提供的 DB2 资料库。可以从 www.ibm.com/support/docview.wss?rs=71&uid=swg2700947 下载 PDF 格式的英文 DB2 版本 9.5 手册和已翻译的版本。

尽管这些表标识书籍有印刷版，但可能未在您所在国家或地区提供。

表 75. DB2 技术信息

书名	书号	是否提供印刷版
Administrative API Reference	SC23-5842-00	是

表 75. DB2 技术信息 (续)

书名	书号	是否提供印刷版
Administrative Routines and Views	SC23-5843-00	否
Call Level Interface Guide and Reference, Volume 1	SC23-5844-00	是
Call Level Interface Guide and Reference, Volume 2	SC23-5845-00	是
Command Reference	SC23-5846-00	是
《数据移动指南和参考》	S151-0617-00	是
《数据恢复及高可用性指南与参考》	S151-0619-00	是
《数据服务器、数据库和数据库对象指南》	S151-0612-00	是
《数据库安全性指南》	S151-0614-00	是
Developing ADO.NET and OLE DB Applications	SC23-5851-00	是
Developing Embedded SQL Applications	SC23-5852-00	是
Developing Java Applications	SC23-5853-00	是
Developing Perl and PHP Applications	SC23-5854-00	否
Developing User-defined Routines (SQL and External)	SC23-5855-00	是
Getting Started with Database Application Development	GC23-5856-00	是
《Linux 和 Windows 上的 DB2 安装和管理入门》	G151-0623-00	是
《国际化指南》	S151-0616-00	是
《消息参考，第 1 卷》	G151-0632-00	否
《消息参考，第 2 卷》	G151-0633-00	否
《迁移指南》	G151-0622-00	是
《Net Search Extender 管理和用户指南》 **注**: DB2 信息中心中并不包含此文档的内容	S151-0761-00	是
《分区和集群指南》	S151-0615-00	是
Query Patroller Administration and User's Guide	SC23-8507-00	是
《IBM 数据服务器客户机快速入门》	G151-0625-00	否
《DB2 服务器快速入门》	G151-0624-00	是
《Spatial Extender 和地理数据管理功能部件用户指南和参考》	S151-0760-00	是
SQL Reference, Volume 1	SC23-5861-00	是
SQL Reference, Volume 2	SC23-5862-00	是

表 75. DB2 技术信息 (续)

书名	书号	是否提供印刷版
《系统监视器指南和参考》	S151-0618-00	是
《文本搜索指南》	S151-0620-00	是
《故障诊断指南》	G151-0621-00	否
《调整数据库性能》	S151-0613-00	是
《Visual Explain 教程》	S151-0634-00	否
《新增内容》	S151-0629-00	是
Workload Manager Guide and Reference	SC23-5870-00	是
《pureXML 指南》	S151-0630-00	是
《XQuery 参考》	S151-0631-00	否

表 76. 特定于 DB2 Connect 的技术信息

书名	书号	是否提供印刷版
《DB2 Connect 个人版快速入门》	G151-0627-00	是
《DB2 Connect 服务器快速入门》	G151-0628-00	是
《DB2 Connect 用户指南》	S151-0626-00	是

表 77. Information Integration 技术信息

书名	书号	是否提供印刷版
Information Integration: Administration Guide for Federated Systems	SC19-1020-01	是
Information Integration: ASNCLP Program Reference for Replication and Event Publishing	SC19-1018-02	是
Information Integration: 《联合数据源配置指南》	S151-0468-00	否
Information Integration: 《SQL 复制指南和参考》	S151-0475-00	是
Information Integration: Introduction to Replication and Event Publishing	SC19-1028-01	是

订购印刷版的 DB2 书籍

如果您需要印刷版的 DB2 书籍，可以在许多（但不是所有）国家或地区在线购买。无论何时都可以从当地的 IBM 代表处订购印刷版的 DB2 书籍。请注意，*DB2 PDF 文档* DVD 上的某些软拷贝书籍没有印刷版。例如，*DB2 消息参考*的任何一卷都没有提供印刷版书籍。

只要支付一定费用，就可以从 IBM 获取 DB2 PDF 文档 DVD，该 DVD 包含许多 DB2 书籍的印刷版。 根据您下订单的位置，您可能能够从 IBM 出版物中心在线订购书籍。如果在线订购在您所在国家或地区不可用，您总是可以从当地的 IBM 代表处订购印刷版 DB2 书籍。注意，并非 DB2 PDF 文档 DVD 上的所有书籍都有印刷版。

注：最新最完整的 DB2 文档保留在网址如下的 DB2 信息中心中：http://publib.boulder.ibm.com/infocenter/db2luw/v9r5。

要订购印刷版的 DB2 书籍：

- 要了解您是否可从所在国家或地区在线订购印刷版的 DB2 书籍，可查看 IBM 出版物中心站点，网址为：http://www.ibm.com/shop/publications/order。 必须先选择国家、地区或语言才能访问出版物订购信息，然后再按照针对您所在位置的订购指示信息进行订购。
- 要从当地的 IBM 代表处订购印刷版的 DB2 书籍：
  1. 从下列其中一个 Web 站点找到当地代表处的联系信息：
     - IBM 全球联系人目录，网址为 www.ibm.com/planetwide
     - IBM 出版物 Web 站点，网址为 http://www.ibm.com/shop/publications/order。必须先选择国家、地区或语言才能访问对应您的所在地的出版物主页。在此页面中访问“关于此站点”链接。
  2. 请在致电时说明您想订购 DB2 出版物。
  3. 请向您当地的代表处提供想要订购的书籍的书名和书号。有关书名和书号的信息，请参阅第 375 页的『硬拷贝或 PDF 格式的 DB2 技术库』。

从命令行处理器显示 SQL 状态帮助

DB2 返回描述 SQL 语句执行结果的 SQLSTATE。SQLSTATE 帮助说明 SQL 状态和 SQL 状态类代码的含义。

要调用 SQL 状态帮助，打开命令行处理器并输入：

```
? sqlstate or ? 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。

访问不同版本的 DB2 信息中心

对于 DB2 版本 9.5 主题，DB2 信息中心 URL 为 http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/

对于 DB2 版本 9 主题，DB2 信息中心 URL 为 http://publib.boulder.ibm.com/infocenter/db2luw/v9/

对于 DB2 版本 8 主题，请访问以下版本 8 信息中心 URL：http://publib.boulder.ibm.com/infocenter/db2luw/v8/

在 DB2 信息中心中以您的首选语言显示主题:

DB2 信息中心尝试以您在浏览器首选项中指定的语言显示主题。如果未提供主题的首选语言翻译版本，那么 DB2 信息中心将显示该主题的英文版。

- 要在 Internet Explorer 浏览器中以您的首选语言显示主题:
  1. 在 Internet Explorer 中，单击**工具** —> **Internet 选项** —> **语言...**按钮。“语言首选项”窗口打开。
  2. 确保您的首选语言被指定为语言列表中的第一个条目。
     - 要将新语言添加至列表，单击**添加...** 按钮。

       注: 添加语言并不能保证计算机具有以首选语言显示主题所需的字体。
     - 要将语言移至列表顶部，选择该语言并单击**上移**按钮直到该语言成为语言列表中的第一个条目。
  3. 清除浏览器高速缓存然后刷新页面以便以首选语言显示 DB2 信息中心。
- 要在 Firefox 或 Mozilla 浏览器中以首选语言显示主题:
  1. 在**工具** —> **选项** —> **高级**对话框中的**语言**部分中选择按钮。“语言”面板将显示在“首选项”窗口中。
  2. 确保您的首选语言被指定为语言列表中的第一个条目。
     - 要将新语言添加至列表，单击**添加...** 按钮以从“添加语言”窗口中选择一种语言。
     - 要将语言移至列表顶部，选择该语言并单击**上移**按钮直到该语言成为语言列表中的第一个条目。
  3. 清除浏览器高速缓存然后刷新页面以便以首选语言显示 DB2 信息中心。

在某些浏览器和操作系统组合上，可能还必须将操作系统的区域设置更改为您选择的语言环境和语言。

更新安装在您的计算机或内部网服务器上的 DB2 信息中心

如果已经在本地安装了 DB2 信息中心，您可以下载并安装 IBM 提供的更新。

更新在本地安装的 DB2 信息中心要求您:

1. 停止计算机上的 DB2 信息中心，然后以独立方式重新启动信息中心。如果以独立方式运行信息中心，那么网络上的其他用户将无法访问信息中心，因而您可以下载和应用更新。
2. 使用“更新”功能部件来查看可用的更新。如果有您希望安装的更新，那么请使用“更新”功能部件来下载并安装这些更新。

   注: 如果您的环境要求在一台未连接至因特网的机器上安装 DB2 信息中心更新，那么必须使用一台已连接至因特网的机器将更新站点镜像至本地文件系统并安装 DB2 信息中心。如果网络中有许多用户将安装文档更新，那么可以通过在本地也为更新站点建立镜像并为更新站点创建代理来缩短每个人执行更新所需要的时间。
   如果提供了更新包，那么使用更新功能来下载这些更新包。但是，只有在单机方式下才能使用更新功能。
3. 停止独立信息中心，然后在计算机上重新启动 DB2 信息中心。

注: 在 Windows Vista 上，必须以管理员身份才能运行下面所列示的命令。要启动具有所有管理员特权的命令提示符或图形工具，右键单击快捷方式，然后选择**以管理员身份运行**。

要更新安装在您的计算机或内部网服务器上的 DB2 信息中心:

1. 停止 DB2 信息中心。
   - 在 Windows 上，单击**开始 → 控制面板 → 管理工具 → 服务**。右键单击 **DB2 信息中心**服务，并选择**停止**。
   - 在 Linux 上，输入以下命令:

     ```
     /etc/init.d/db2icdv95 stop
     ```
2. 以独立方式启动信息中心。
   - 在 Windows 上:
     a. 打开命令窗口。
     b. 浏览至信息中心的安装位置。缺省情况下，DB2 信息中心安装在 <Program Files>\IBM\DB2 Information Center\Version 9.5 目录中，其中 <Program Files> 表示 Program Files 目录的位置。
     c. 从安装目录浏览至 doc\bin 目录。
     d. 运行 help_start.bat 文件:

        ```
        help_start.bat
        ```
   - 在 Linux 上:
     a. 浏览至信息中心的安装位置。缺省情况下，DB2 信息中心安装在 /opt/ibm/db2ic/V9.5 目录中。
     b. 从安装目录浏览至 doc/bin 目录。
     c. 运行 help_start 脚本:

        ```
        help_start
        ```

   系统缺省 Web 浏览器将启动以显示独立信息中心。
3. 单击“更新”按钮（  ）。在信息中心的右边面板上，单击查找更新。将显示现有文档的更新列表。
4. 要启动下载进程，请检查您想要下载的选项，然后单击安装更新。
5. 在完成下载和安装进程后，单击完成。
6. 停止独立信息中心。
   - 在 Windows 上，浏览至安装目录的 doc\bin 目录并运行 help_end.bat 文件:

     ```
     help_end.bat
     ```

     注: help_end 批处理文件包含安全终止用 help_start 批处理文件启动的进程所需的命令。不要使用 Ctrl-C 或任何其他方法来终止 help_start.bat。
   - 在 Linux 上，浏览至安装目录的 doc/bin 目录并运行 help_end 脚本:

     ```
     help_end
     ```

     注: help_end 脚本包含安全终止用 help_start 脚本启动的进程所需的命令。不要使用任何其他方法来终止 help_start 脚本。
7. 重新启动 DB2 信息中心。

- 在 Windows 上，单击**开始 → 控制面板 → 管理工具 → 服务**。右键单击 **DB2 信息中心**服务，并选择**启动**。
- 在 Linux 上，输入以下命令:

  ```
  /etc/init.d/db2icdv95 start
  ```

更新后的 DB2 信息中心将显示新的主题和更新后的主题。

DB2 教程

DB2 教程帮助您了解 DB2 产品的各个方面。这些课程提供了逐步指示信息。

开始之前

可从信息中心查看 XHTML 版的教程: http://publib.boulder.ibm.com/infocenter/db2help/ 。

某些课程使用了样本数据或代码。有关其特定任务的任何先决条件的描述，请参阅教程。

DB2 教程

要查看教程，请单击标题。

《pureXML 指南》 **中的『pureXML™』**
设置 DB2 数据库以存储 XML 数据以及如何对本机 XML 数据存储执行基本操作。

《Visual Explain 教程》 **中的『Visual Explain』**
使用 Visual Explain 来分析、优化和调整 SQL 语句以获取更好的性能。

DB2 故障诊断信息

很多故障诊断和问题确定信息可帮助您使用 DB2 产品。

DB2 文档
故障诊断信息可在 DB2 信息中心的“DB2 故障诊断指南”或“支持和故障诊断”部分找到。可在该处找到有关如何使用 DB2 诊断工具和实用程序隔离和找出问题的信息、某些最常见问题的解决方案以及有关如何解决使用 DB2 产品时可能遇到的问题的建议。

DB2 技术支持 Web 站点
如果您遇到了问题并且想要获取查找可能的原因和解决方案的帮助，请参阅 DB2 技术支持 Web 站点。该“技术支持”站点具有指向最新 DB2 出版物、技术说明、授权程序分析报告（APAR 或错误修订）、修订包和其他资源的链接。可搜索此知识库并查找问题的可能解决方案。

访问位于以下网址的 DB2 技术支持 Web 站点: http://www.ibm.com/software/data/db2/udb/support.html

条款和条件

如果符合以下条款和条件，那么授予您使用这些出版物的准用权。

个人使用: 只要保留所有的专有权声明，您就可以为个人、非商业使用复制这些出版物。未经 IBM 明确同意，您不可以分发、展示或制作这些出版物或其中任何部分的演绎作品。

商业使用: 只要保留所有的专有权声明，您就可以仅在企业内复制、分发和展示这些出版物。未经 IBM 明确同意，您不可以制作这些出版物的演绎作品，或者在您的企业外部复制、分发或展示这些出版物或其中的任何部分。

除非本准用权中有明确授权，不得把其他准用权、许可或权利（无论是明示的还是暗含的）授予其中包含的出版物或任何信息、数据、软件或其他知识产权。

当使用这些出版物损害了 IBM 的利益，或者根据 IBM 的规定，未正确遵守上述指导说明时，那么 IBM 保留自主决定撤销本文授予的准用权的权利。

您不可以下载、出口或再出口本信息，除非完全遵守所有适用的法律和法规，包括所有美国出口法律和法规。

IBM 对这些出版物的内容不作任何保证。这些出版物“按现状”提供，不附有任何种类的（无论是明示的还是暗含的）保证，包括但不限于暗含的关于适销和适用于某种特定用途的保证。

附录 B. 声明

本信息是为在美国提供的产品和服务编写的。

IBM 可能在其他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 代表咨询。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，那么由用户自行负责。

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。您可以用书面方式将许可查询寄往:

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

本条款不适用英国或任何这样的条款与当地法律不一致的国家或地区: International Business Machines Corporation“按现状”提供本出版物，不附有任何种类的（无论是明示的还是暗含的）保证，包括但不限于暗含的有关非侵权、适销和适用于某种特定用途的保证。某些国家或地区在某些交易中不允许免除明示或暗含的保证。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。

本文档可提供非 IBM Web 站点和资源的链接和引用。IBM 对本文档中引用、提供或链接到的任何非 IBM Web 站点或第三方资源不承担任何责任、保证或其他义务。对非 IBM Web 站点的链接并不意味着IBM 认可这些 Web 站点或其拥有者的内容或用途。此外，IBM并不是您使用第三方可进入的任何事务的一方，也不会为这些事务负责，即使您是通过IBM 站点了解或链接到第三方。因此，您承认并认可 IBM 不会为这些外部站点或资源的可用性负责，也不会为它们提供的任何内容、服务、产品或其他事务承担责任或义务。第三方提供的任何软件都遵守与该软件相关的许可条款和条件。

IBM 可以按它认为适当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他程序（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。

本资料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 国际软件许可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，在其他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、其出版说明或其他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、兼容性或任何其他关于非 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。

本信息可能包含在日常业务操作中使用的数据和报告的示例。为了尽可能完整地说明这些示例，示例中可能会包括个人、公司、品牌和产品的名称。所有这些名称都是虚构的，与实际商业企业所用的名称和地址的任何雷同纯属巧合。

版权许可:

本信息可能包括源语言形式的样本应用程序，这些样本说明不同操作平台上的编程方法。如果是为按照在编写样本程序的操作平台上的应用程序编程接口（API）进行应用程序的开发、使用、经销或分发为目的，您可以任何形式对这些样本程序进行复制、修改、分发，而无须向 IBM 付费。这些示例并未在所有条件下作全面测试。因此，IBM 不能担保或暗示这些程序的可靠性、可维护性或功能。

凡这些样本程序的每份拷贝或其任何部分或任何衍生产品，都必须包括如下版权声明:

©（*贵公司的名称*）（*年*）。此部分代码是根据 IBM 公司的样本程序衍生出来的。© Copyright IBM Corp. （*输入年份*）. All rights reserved.

商标

DB2 版本 9.5 文档库的各个文档中标识的公司、产品或服务名称可能是 International Business Machines Corporation 或其他公司的商标或服务标记。有关 IBM Corporation 在美国和 / 或其他国家的商标的信息在 http://www.ibm.com/legal/copytrade.shtml 中。

下列各项是其他公司的商标或注册商标，且已在 DB2 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 / 或其他国家或地区的商标。

Intel®、Intel 徽标、Intel Inside® 徽标、Intel Centrino®、Intel Centrino 徽标、Celeron®、Intel Xeon®、Intel SpeedStep®、Itanium® 和 Pentium® 是 Intel Corporation 在美国和 / 或其他国家或地区的商标。

Java™ 和所有基于 Java 的商标是 Sun Microsystems,Inc. 在美国和 / 或其他国家或地区的商标。

UNIX 是 The Open Group 在美国和其他国家或地区的注册商标。

Linux 是 Linus Torvalds 在美国和其他国家或地区的注册商标。

Adobe®、Adobe 徽标、PostScript® 和 PostScript 徽标是 Adobe Systems Incorporated 在美国和 / 或其他国家或地区的注册商标或商标。

其他公司、产品或服务名称可能是其他公司的商标或服务标记。

索引

[B]

[C]

[D]

[F]

[G]

[H]

[J]

[N]

[P]

[Q]

[R]

[S]

[T]

[W]

[X]

[Y]

[Z]

[数字]

A

B

C

D

F

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

中国印刷

S151-0613-00

Spine information:

DB2 版本 **9.5 Linux** 版、**UNIX** 版和 **Windows** 版

调整数据库性能