SinoDB 性能指南

星瑞格 SinoDB 产品系列

SinoDB

版本 12.10

目录

插图清单

表格清单

简介

本简介概述了本出版物中的信息，并描述了所使用的约定。

关于本出版物

本出版物提供了有关如何配置和操作 SinoDB® 以提高系统的总吞吐量和 SQL 查询性能的信息。本出版物也包含有关性能调优问题和方法以及分段存储准则的信息。

本出版物中的信息可以帮助您执行以下任务：

- 监视对性能至关重要的系统资源
- 标识影响这些关键资源的数据库活动
- 标识和监视对性能至关重要的查询
- 使用数据库服务器实用程序（尤其是 onstat）以监视和调优性能
- 通过以下措施消除性能瓶颈：
    - 平衡系统资源的负载
    - 调整数据库服务器的配置参数或环境变量
    - 调整数据的排列
    - 为决策支持查询分配资源
    - 创建索引以加速数据检索

性能评估和调优包括广泛的调查与实践，并可能涉及超出本出版物范围的信息。

超出本出版物范围的主题

尝试平衡工作负载经常可以连续适度提高性能。有时性能的提高非常明显。但在某些情况下，负载平衡方法还不够。在下列情况中可能需要本出版物范围之外的措施：

- 需要加以修改以便更好地使用数据库服务器或操作系统资源的应用程序
- 以降低性能的方式进行交互的应用程序
- 可能受限于使用冲突的主机
- 容量无法满足工作负载增长需要的主机
- 影响客户端/服务器或其他应用程序的网络性能问题

数据库调优不能纠正这些情况。尽管如此，当数据库服务器配置正确时，这些情况比较容易被识别和解决。

> 重要： 虽然性能方面的注意事项还包括可靠性和数据可用性，以及改进的响应时间和系统资源的高效使用，但在本出版物中只讨论响应时间和系统资源的使用。有关改进后的数据库服务器可靠性和数据可用性的讨论，请参阅《*SinoDB*® 管理员指南》中有关转换、镜像和高可用性的信息。有关备份与还原的信息，请参阅《*SinoDB*® 备份和还原指南》。

用户类型

本出版物是为以下用户编写的：

- 数据库管理员
- 数据库服务器管理员
- 数据库应用程序员
- 性能工程师

本出版物假定您有以下知识背景：

- 对于计算机、操作系统和操作系统提供的实用程序的应用知识
- 使用关系数据库的相关经验或者了解数据库概念

- 一些计算机编程经验
- 有数据库服务器管理、操作系统管理或网络管理方面的相关经验

软件依赖性

本出版物假定您使用的是 SinoDB® V12.10。

演示数据库

DB-Access实用程序随SinoDB®数据库服务器产品一起提供，它包括一个或多个以下演示数据库：

- stores_demo 数据库以一家虚构的体育用品批发商的有关信息举例说明了关系模式。星瑞格®出版物中的许多示例均基于 stores_demo 数据库。
- superstores_demo 数据库举例说明了对象关系模式。superstores_demo 数据库包含扩展数据类型、类型和表继承以及用户自定义例程的示例。

有关如何创建和填充演示数据库的信息，请参阅《*SinoDB® DB-Access* 用户指南》。有关数据库及其内容的描述，请参阅《*SinoDB® SQL* 指南*:* 参考》。

用于安装演示数据库的脚本位于 UNIX™ 平台上的 $INFORMIXDIR/bin 目录和 Windows™ 环境中的 %INFORMIXDIR%\bin 目录中。

示例代码约定

SQL 代码的示例出现在整个出版物中。除非另有说明，代码不特定于任何单个的 SinoDB® 应用程序开发工具。

如果示例中仅列出 SQL 语句，那么它们将不用分号定界。例如：您可能看到以下示例中的代码：

```
CONNECT TO stores_demo
...

DELETE FROM customer
   WHERE customer_num = 121
...

COMMIT WORK
DISCONNECT CURRENT
```

要将此 SQL 代码用于特定产品，必须应用该产品的语法规则。例如，如果使用的是 SQL API，那么必须在每条语句的开头使用 EXEC SQL ，并在每条语句的结尾使用分号（或其他合适的定界符）。 如果使用的是 DB-Access，那么必须用分号将多条语句隔开。

> 提示: 代码示例中的省略点表示在整个应用程序中可添加更多的代码，但对于正在讨论的概念是不需要显示的。

有关使用特定应用程序开发工具或 SQL API 的 SQL 语句的详细指导，请参阅您的产品文档。

符合行业标准

SinoDB® 产品符合各种标准。

基于 SinoDB® SQL 的产品完全符合 SQL-92 入门标准（发布为 ANSI X3.135-1992），即 ISO 9075:1992 标准。另外，SinoDB® 数据库服务器的许多功能都遵循 SQL-92 中级和最高级标准，以及 X/Open SQL 公共应用程序环境 (CAE) 标准。

第 1 章

性能基础信息

性能测量和调优方面的问题和方法与日常数据库服务器管理和查询执行相关。

本章节包括以下主题：

- 描述性能测量和调优的基本方法
- 提供快速入门的准则，以在小型数据库上获取可接受的初始性能
- 描述维护良好性能的角色

开发性能测量和调优的基本方法

为了维持数据库应用程序的最优性能，制定评估系统性能的计划，进行调整以维持良好的性能并在性能降低时采取纠正措施。定期的特定评估能够帮助您预测并纠正性能问题。

通过在初期识别出问题，可以有效防止其影响用户。性能问题的早期迹象通常是模糊的；用户可能会报告系统运行缓慢。用户可能会抱怨他们无法完成所有工作、事务花费过长时间完成、处理查询花费了过长的时间或者应用程序在一天中的某些时间减慢。

要确定问题的本质，必须评估系统资源的实际使用情况并估计结果。

用户通常报告以下情况的性能问题：

- 事务或查询的响应时间比预期长。
- 事务吞吐量不足以完成必需的工作负载。
- 事务吞吐量减少。

推荐使用交互式方法来优化数据库服务器性能。如果重复执行下表中的步骤没有产生预期的改进，那么引起该问题的原因可能是在一个或多个客户端应用程序中硬件资源不足或代码无效。

要优化性能，请执行以下步骤：

1. 建立性能目标。
2. 对资源利用率和数据库活动进行定期评估。
3. 标识性能问题的症状：CPU、内存或磁盘的过度使用。
4. 调整操作系统配置。
5. 调整数据库服务器配置。
6. 优化块和数据库空间配置，包括日志的位置、分类空间、临时表空间以及分类文件的空间。
7. 优化表位置、扩展数据块缩放以及分段存储。
8. 改进索引。
9. 优化后台 I/O 活动，包括日志记录、检查点和页清除。
10. 在非高峰时间调度备份和批处理操作。
11. 优化数据库应用程序的实现。
12. 重复执行步骤 2 到 11。

小型数据库上可接受性能的快速入门

如果您有一个小型数据库，每个表仅驻留在一个磁盘上且仅使用一个 CPU 虚拟处理器，那么可以执行特定评估以帮助您预测和更正性能问题。

要在小型数据库上实现可接受的初始性能，请执行以下步骤：

1. 生成表和索引的统计信息，为查询优化器提供信息，使其能够选择估算成本最低的查询计划。

   这些统计信息是为单独查询获取良好性能的最低起点。有关准则，请参阅未自动生成统计信息时更新统计信息 在第274页。要查看优化器为每个查询选择的查询计划，请参阅显示查询计划 在第265页。
2. 如果要让一个查询与其他查询并行运行，那么必须开启并行数据库查询 (PDQ) 功能。

   如果没有在多个磁盘上进行表分段存储，将不会发生并行扫描。只有一个 CPU 虚拟处理器时，将不会发生并行连接或并行分类。但是，PDQ 优先级可以获取更多的内存来执行排序。有关更多信息，请参阅并行数据库查询 *(PDQ)* 在第251页。
3. 如果将联机事务处理 (OLTP) 和决策支持系统 (DSS) 查询应用程序进行混合，您可以控制长期运行的查询可以获取的资源量，以便 OLTP 事务不会受到影响。

   有关如何控制 PDQ 资源的信息，请参阅为并行数据库查询分配资源 在第255页。
4. 监视会话并深入研究各种详细信息以改进单独查询的性能。

   有关各种工具的信息和要监视的会话详细信息，请参阅监视每个会话的内存使用情况 在第297页和监视会话和线程 在第301页。

性能目标

计划度量和调优性能时，应该考虑性能目标并确定哪些目标最重要。

为数据库服务器及其支持的应用程序建立性能目标要考虑许多注意事项。要清楚性能目标和优先级的结合并保持一致，从而可以为应用程序提供有关性能目标的现实且一致的期望。建立性能目标时，请考虑以下问题：

- 您的最高优先级会最大化事务吞吐量、最小化特定查询的响应时间或实现最佳的全面混合吗？
- 在简单的事务、扩展的决策支持查询和其他类型的请求之间数据库服务器通常处理什么类型的混合？
- 您希望在什么时候用事务处理速度与可用性或丢失特殊事务的风险相交换？
- 客户端/服务器配置中使用此数据库服务器实例吗？如果是，那么影响其性能的联网特征是什么？
- 您期望的最大用户数是多少？
- 您的配置受内存、磁盘空间或者 CPU 资源限制吗？

这些问题的答案能够帮助您为资源和应用程序混合设置现实的性能目标。

性能测量

可以使用吞吐量、响应时间、每个事务的成本和资源利用率度量来评估性能。

在接下来的主题中将描述吞吐量、响应时间和每个事务的成本。

根据上下文，资源利用率可以具有两种含义之一。该术语可以指某个特殊操作需要或使用的资源量，也可以指在某个特殊系统组件上的当前负载。在前一种情况使用该术语比较完成给定任务的方法。例如，如果给定的分类操作需要 10 兆字节的磁盘空间，那么其资源利用率就比仅需要 5 兆字节磁盘空间的另一个分类操作大。在后一种情况使用该术语，指的是（例如）在特定时间间隔内专用于特殊查询的 CPU 周期数。

有关在各种系统组件上不同载入级别的性能影响的讨论，请参阅资源利用率和性能 在第22页。

吞吐量

吞吐量评估系统的整体性能。对于事务处理系统，吞吐量通常用每秒事务数 (TPS) 或每分钟事务数 (TPM) 来评估。

吞吐量取决于以下因素：

- 主计算机的规格
- 软件中的处理开销
- 磁盘上数据的布局
- 硬件和软件都支持的并行度
- 正在处理的事务类型

测量吞吐量的方法

评估应用程序吞吐量的最佳方法是在该应用程序中包含代码，用于记录事务提交时的时间戳。

如果应用程序不直接提供对评估吞吐量的支持，您可以通过跟踪在给定的时间间隔内数据库服务器记录的 COMMIT WORK 语句的数量获得估计值。您可以使用 onlog 实用程序来获取写入日志文件的逻辑日志记录列表。可以使用此命令的信息来跟踪插入、删除和更新操作以及提交的事务。但是，在信息写入日志文件之前，您无法获取逻辑日志缓冲区中存储的信息。

如果需要更及时的反馈，可以使用 onstat -p 来收集估计值。可使用 SET LOG 语句将包含关注表的数据库的记录方式设置为不缓冲。也可以使用数据库服务器中可信的审计工具在审计日志文件中记录成功的 COMMIT WORK 事件或关注的其他事件。使用审计工具可能会增加在处理任何审计事件中所涉及的开销，从而减少整体吞吐量。

相关链接

《SinoDB 安全指南》：审计数据安全

标准吞吐量基准

事务处理性能委员会 (TPC) 提供标准的基准，这些基准能够对各种硬件配置以及数据库服务器进行合理的吞吐量比较。星瑞格®是 TPC 声誉良好的积极成员。

TPC 提供以下标准化基准来评估吞吐量：

- TPC-A

  此基准用于简单的联机事务处理 (OLTP) 比较。它表征简单事务处理系统的性能，强调更新密集型服务。TPC-A 模仿一个工作负载，该工作负载由使用重要的磁盘 I/O 活动通过网络连接的多个用户会话组成。
- TPC-B

  此基准用于对高峰数据库吞吐量进行压力测试。它使用与 TPC-A 相同的事务负载，但除去所有的联网和交互式操作以提供最佳的吞吐量评估。
- TPC-C

  此基准用于复杂的 OLTP 应用程序。它从 TPC-A 派生而来，并混合使用更新、只读事务、批处理操作、事务回滚请求、资源争用以及复杂数据库上其他类型的操作以更好地表示典型工作负载。
- TPC-D

  此基准根据非常大型的查询的完成时间评估查询处理能力。TPC-D 是围绕一组典型的业务问题建立的决策支持基准，这些问题表现为针对大型数据库（在千兆字节或太字节范围内）的 SQL 查询。

由于每个数据库应用程序均有其特定的工作负载，因此不能使用 TPC 基准来预测应用程序的吞吐量。实际达到的吞吐量很大程度上取决于您的应用程序。

响应时间

响应时间评估单个事务或查询的性能。通常将响应时间视为是从用户输入一个命令或激活一个函数到应用程序指示该命令或函数已完成所消耗的时间。

典型 SinoDB® 应用程序的响应时间包括以下操作序列。每个操作都需要一定的时间。响应时间不包括用户思考和输入查询或请求的时间：

1. 应用程序将查询转发到数据库服务器。
2. 数据库服务器执行查询最优化并检索所有用户定义例程 (UDR)。UDR 包括 SPL 例程和外部例程。
3. 数据库服务器检索、添加或更新相应的记录并执行与查询直接相关的磁盘 I/O 操作。
4. 数据库服务器执行发生在查询或事务仍处于暂挂状态期间的任何后台 I/O 操作，例如日志记录和页清除。
5. 数据库服务器将结果返回给应用程序。
6. 应用程序显示信息或发出确认并随后向用户发出新的提示。

图 *1:* 单个事务的响应时间的组成部分 在第21页包含的图显示步骤 1 到 6 中所述的操作如何作用于整体响应时间。

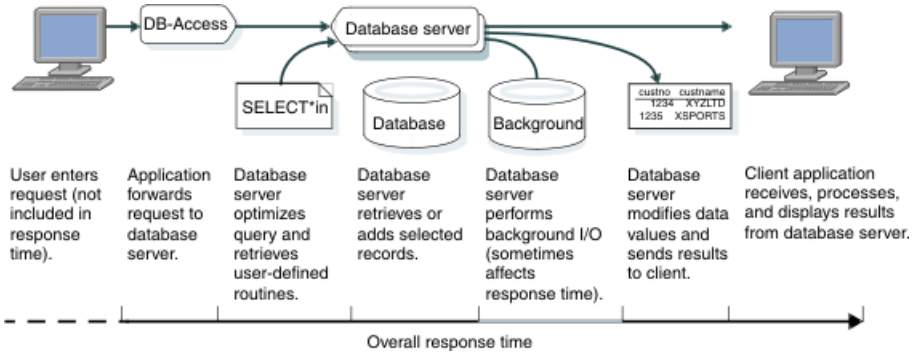

图 1: 单个事务的响应时间的组成部分

响应时间和吞吐量

响应时间和吞吐量是相关联的。在您增加总体吞吐量时一般事务的响应时间会减少。

但是，可以通过为特定查询分配不成比例的资源数量，在牺牲总体吞吐量的情况下减少该查询的响应时间。相反，可以通过限制数据库分配给大型查询的资源数来维持总体吞吐量。

当尝试在对高事务吞吐量的当前需求和对执行大型决策支持查询的即时需求之间取得平衡时，吞吐量与响应时间之间的平衡就变得明显起来。应用于查询的资源越多，可用于处理事务的资源就越少，并且查询对事务吞吐量的影响就越大。相反，提供给查询的资源越少，查询花费的时间就越长。

响应时间测量

要评估查询或应用程序的响应时间，可以使用操作系统提供的计时命令、性能监视和计时函数。

操作系统计时命令

操作系统中通常有一个可以对命令进行计时的实用程序。可以经常使用此计时实用程序评估 DB-Access 命令文件发出的 SQL 语句的响应时间。

仅适用于 UNIX™

如果您有一个执行一组标准 SQL 语句的命令文件，就可以在许多系统上使用 time 命令以获取那些命令的准确计时。

以下示例显示 UNIX™ time 命令的输出：

```
time commands.dba
...
4.3 real          1.5 user             1.3 sys
```

time 输出列出了耗用时间（实时）、用户 CPU 时间和系统 CPU 时间。如果您使用 C shell 程序，那么来自该 C shell 程序 time 命令输出的前三列分别显示用户、系统和已消耗的时间。通常，当系统 CPU 时间的比例超过了总耗用时间的三分之一时，应用程序的执行效果不佳。

time 命令收集有关应用程序的计时信息。可以使用此命令调用应用程序的实例、执行数据库操作并随后退出以获取计时数据，如下例所示：

```
time sqlapp
   (enter SQL command through sqlapp, then exit)
```

```
10.1 real         6.4 user         3.7 sys
```

可以使用一个脚本重复运行同一个测试，这样可以获得不同条件下的可比结果。您也可以通过用脚本的总消耗时间除以脚本执行的数据库操作数，得到平均响应时间的估算值。

用于监视性能的操作系统工具
操作系统通常有一个可以用来评估查询或进程的响应时间的性能监视器。

仅适用于 Windows™

可以经常使用 Windows™ 操作系统提供的性能日志和警报来评估以下时间：

- 用户时间
- 处理器时间
- 消耗的时间

应用程序内的计时函数
大多数编程语言都有日时的库函数。如果可以访问源代码，那么可以将调用对插入到此函数，以评估特定操作之间的消耗时间。

仅适用于 ESQL/C

例如，如果应用程序是用 SinoDB® ESQL/C 编写的，那么可以使用 dtcurrent() 函数来获取当前时间。要评估响应时间，可以调用 dtcurrent() 在事务开始时报告时间并在事务提交时再次报告时间。

在资源于多个进程之间共享的多道程序设计系统或网络环境中，消耗时间并不总是与执行时间相对应。大多数操作系统和 C 库都包含返回程序的 CPU 时间的函数。

每个事务的成本

每个事务的成本是财务上的量度，通常用于比较应用程序、数据库服务器或硬件平台之间的总体操作成本。可以测量每个事务的成本。

要测量每个事务的成本，请执行以下步骤：

1. 计算与运行应用程序相关的所有成本。这些成本可能包括硬件和软件的安装价格；运营成本（包括薪水）及其他费用。

   这些成本可能包括硬件和软件的安装价格；运营成本（包括薪水）及其他费用。
2. 设计应用程序有效期的事务和查询的总数。
3. 用总成本除以事务总数。

虽然该测量对于进行规划和评估很有用，但是它与达到最佳性能的日常问题几乎无关。

资源利用率和性能

典型的事务处理应用程序在其各个运行周期中需要满足的要求各不相同。日、周、月和年的峰值负载以及决策支持 (DSS) 查询或备份操作所施加的负载对于任何容量将耗尽的系统都会产生明显的影响。可以使用从特定系统派生的直接历史数据精确地测定这种影响。

必须对系统的工作负载和性能进行定期评估，以预测峰值负载并比较使用周期中不同时刻的性能评估。定期评估有助于为数据库服务器应用程序开发总体的性能概要文件。该概要文件对于确定如何可靠地提高性能具有关键意义。

有关数据库服务器提供的评估工具，请参阅数据库服务器工具 在第30页。有关操作系统提供的用于评估对系统和硬件资源的性能影响的工具，请参阅操作系统工具 在第29页。

与组件可用的总时间相比，利用率是该组件实际被占用的时间的百分比。例如，如果 CPU 在 1 分钟内总共耗用 40 秒的时间处理事务，那么它在该时间间隔内的利用率为 67%。

定期评估并记录以下系统资源的利用率：

- CPU
- 内存

- 磁盘

当某资源被过度使用或者其利用率与其他组件的利用率不成比例时，该资源对于性能是临界的。例如，当一个磁盘的利用率达到 70%，而系统中其他所有磁盘的利用率只有 30% 的时候，可以认为该磁盘是临界的或被过度使用。尽管 70% 不表示磁盘使用严重过度，但是您可以通过重新安排数据以平衡整组磁盘上的 I/O 请求，从而提高性能。

如何评估资源利用率取决于操作系统为报告系统活动和资源利用率所提供的工具。发现资源似乎使用过度后，就可以使用数据库服务器提供的性能监视实用程序来收集数据，并推断可能占用该组件上负载的数据库活动。可以调整数据库服务器的配置或操作系统，以减少那些数据库活动或将它们分散到其他组件中。某些情况下，可能需要提供额外的硬件资源来解决性能瓶颈问题。

资源利用率

只要系统资源（例如，CPU 或特定磁盘）被事务或查询占用，资源就无法用来处理其他请求。暂挂的请求必须等待资源可用才能完成。

当组件很忙而无法及时处理所有请求时，被过度使用的组件就成为活动流中的瓶颈。资源被占用的时间百分比越高，每个操作必须等待的时间也就越长。

可以使用以下公式基于处理请求的组件的总利用率来估算请求的服务时间。预期的服务时间包括用于等待的时间和使用所讨论资源的时间。将服务时间看作是计算机中单个组件占用的响应时间的一部分，如以下公式所示：

```
S= P/(1-U)
```

S

预期的服务时间。

P

操作获得资源后需要的处理时间。

U

资源利用率（用小数表示）。

如图 *2:* 单个组件的服务时间可看作资源利用率的函数 在第23页所示，随着利用率增加到 70% 以上，单个组件的服务时间显著增加。例如，如果某个事务需要给定的组件用 1 秒钟进行处理，那么当组件的利用率为 50% 时，需要用 2 秒钟处理该事务，而组件的利用率为 80% 时则需要 5 秒钟。当资源利用率达到 90% 时，那么组件可能需要 10 秒钟才能处理完事务。

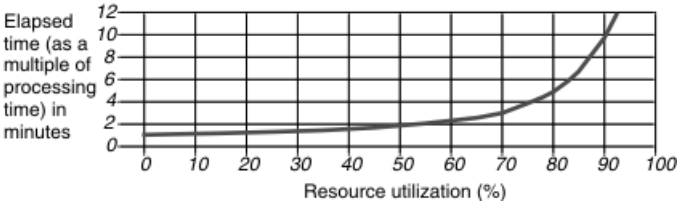

图 2: 单个组件的服务时间可看作资源利用率的函数

如果通常事务的平均响应时间从 2 秒钟或 3 秒钟骤增到 10 秒钟或更长时间，那么用户就会注意到这一点并有所抱怨。

> 重要： 监视显示利用率超过 70% 的所有系统资源或表现出如下章节所述的过度使用现象的所有资源。

当您考虑资源利用率时，也请考虑在环境中增大标准或临时数据库空间的页大小是否有利。如果您想要更长的密钥长度（比标准或临时数据库空间的缺省页大小可提供的长度长），那么可以增大页大小。

CPU 利用率

对 CPU 利用率和响应时间的评估有助于确定是否需要消除或重新调度某些活动。

您可以使用上述主题（资源利用率 在第23页）中的资源利用率公式对重负载 CPU 的响应时间进行评估。但是，如果 CPU 利用率过高，并不表示性能就一定存在问题。CPU 执行处理事务所需的所有计算。在给定时间内 CPU 执行的与事务相关的计算越多，该时间内的吞吐量就越高。只要事务吞吐量高并与 CPU 利用率成正比，那么高 CPU 利用率则表示计算机正在被最大程度地利用。

另一方面，当 CPU 利用率很高而事务吞吐量却不成比例时，CPU 将无法有效处理事务或者从事与事务处理没有直接关系的活动。CPU 周期正在转向内部日常管理任务，例如，内存管理。

可以简单地消除以下活动：

- 大型查询，在非高峰时间调度可能会更好
- 不相关的应用程序，在其他计算机上执行可能会更好

如果事务的响应时间增加到用户无法接受的延迟程度，说明处理器可能被阻塞；对于计算机而言，事务负载可能过高，以至于计算机无法管理。 如果响应时间很慢，也可能表示 CPU 没有有效地处理事务或者 CPU 周期正在被转向。

CPU 利用率很高时，详细分析数据库服务器执行的活动可揭示任何效率不高的原因（可能是由于配置不当）。有关分析数据库服务器活动的信息，请参阅数据库服务器工具 在第30页。

内存利用率

内存不是作为单个组件（例如，CPU 或磁盘）管理的，而是作为称为页的小型组件的集合进行管理。

根据操作系统的不同，内存中典型页的大小范围可从 1 到 8 千字节。内存 64 兆字节、页大小为 2 千字节的计算机大约包含 32,000 页。

当操作系统需要分配内存给进程使用时，将清除内存中所有可找到的未使用页。如果不存在可用页，内存管理系统就必须选择其他进程仍在使用但短期内最不可能使用的页。需要 CPU 周期来选择这些页。查找此类页的过程称为页扫描。需要页扫描时，CPU 利用率将提高。

内存管理系统通常使用最久未使用的算法来选择可复制到磁盘然后再释放以用于其他进程的页。CPU 识别出可换出的页时，通过将这些页中原有的数据复制到专用的磁盘中来换出原来的页图像。存储页映象的磁盘或磁盘分区称为交换磁盘、交换空间或交换区域。这种调页活动需要 CPU 周期及 I/O 操作。

最后，复制到交换磁盘的页映象必须被送回来供需要它们的进程使用。如果可用页仍然很少，就必须换出更多的页以腾出空间。由于内存随不断增加的需求和调页活动而增加，这种活动将最终导致 CPU 几乎完全被调页活动所占用。这种情况下的系统称为系统颠簸。当计算机处于系统颠簸状态时，所有有用的工作都会停止。

为了避免系统颠簸，一些操作系统在调页活动超过特定阈值后使用更近似的内存管理算法。此算法称为交换。当内存管理系统使用交换算法时，将一次拨出构成整个进程映象的所有页，而不是一次拨出一页。

每次交换操作都将释放更多的内存。但是，随着交换的继续，每个交换出去的进程必须再次读入，这将大大增加交换设备的磁盘 I/O 以及进程间切换所需的时间。随后性能会受限于数据从交换磁盘传输回内存的速度。交换是系统严重超载的现象，并且会影响吞吐量。

许多系统提供调页活动的有关信息，其中包括执行的页扫描次数、从内存中发送出（换出）的页数、以及带入内存（换入）的页数：

- 页换出是关键因素，因为操作系统只有在找不到可用页时才进行换页。
- 页扫描的高速率可及早表明内存利用率正在变成瓶颈。
- 终止进程的页将被就地释放并重新使用，因此页换入活动不能准确反映内存上的负载。高速进程回转可导致高速页换入，而不会对性能产生重大影响。

虽然估算内存服务时间的原则与资源利用率和性能 在第22页所描述的相同，但是可以使用不同的公式（与其他系统组件所用公式不同）来评估内存利用率对性能的影响。

您可以使用以下公式来计算给定 CPU 利用率级别和调页速率的预期调页延迟：

```
PD= (C/(1-U)) * R * T
```

PD

调页延迟。

C

事务的 CPU 服务时间。

U

CPU 利用率（用小数表示）。

R

页换出速率。

T

交换设备的服务时间。

随着调页增加，CPU 利用率也会增加，并且这些增加混合在一起。如果调页的速率为每秒 10 页，占 CPU 利用率的 5%，那么调页的速率增加到每秒 20 页时，CPU 利用率可能也会再增加 5%。进一步增加调页将导致 CPU 利用率急剧增加，直到 CPU 请求的预期服务时间不可接受为止。

磁盘利用率

由于磁盘之间的传输速率各不相同，大多数操作系统都不直接报告磁盘利用率。相反，它们报告每秒的数据传输量（以操作系统内存页大小为单位）。

因为每个磁盘都充当一个单独的资源，所以可使用下列基本公式来评估服务时间，这在资源利用率 在第23页中有详细描述：

```
S= P/(1-U)
```

为了比较存取时间相近的磁盘上的负载，只要比较每秒的平均传输量即可。

如果知道给定磁盘的存取时间，就可以使用操作系统报告的每秒传输量来计算磁盘的利用率。要进行此操作，用每秒的平均传输量乘以磁盘生产商列出的磁盘存取时间。根据磁盘中数据的分布，存取时间可能不同于生产商的标定值。考虑到这种可变性，应该将制造商的存取时间规格增加 20%。

以下示例说明如何计算存取时间为 30 毫秒、每秒平均 10 个传输请求的磁盘的利用率：

```
U =     (A * 1.2) * X
  =     (.03 * 1.2) * 10
  =     .36
```

U

资源利用率（这里是指磁盘的利用率）。

A

生产商列出的存取时间（秒）。

X

操作系统报告的每秒传输量。

可以使用利用率估算出要求给定磁盘传输量的事务在磁盘上的处理时间。要计算磁盘上的处理时间，可以将磁盘传输量乘以平均存取时间。包括考虑到存取时间可变性的额外 20%：

```
P = D (A * 1.2)
```

P

磁盘上的处理时间。

D

磁盘传输量。

A

生产商列出的存取时间（秒）。

例如，可以计算需要从 30 毫秒磁盘传输 20 个磁盘的事务的处理时间，如下例所示：

```
P =    20 (.03 * 1.2)
  =    20 * .036
  =    .72
```

使用计算得到的处理时间和利用率值来估算特定磁盘上 I/O 的预期服务时间，如下例所示：

```
S =    P/(1-U)
  =    .72 / (1 - .36)
  =    .72 / .64
  =    1.13
```

影响资源利用率的因素

数据库服务器应用程序的性能取决于许多因素，包括硬件和软件配置、网络配置和数据库的设计。

当尝试识别性能问题或调整系统时，必须考虑以下这些因素：

- 硬件资源

  如本章前面所述，硬件资源包括 CPU、物理内存和磁盘 I/O 子系统。
- 操作系统配置

  数据库服务器根据操作系统提供对设备的低级访问、进程安排、进程间通信和其他重要服务。

  操作系统的配置直接影响数据库服务器的性能状况。操作系统内核占用了大量的物理内存，这部分内存是数据库服务器和其他应用程序不能使用的。但是，必须为数据库服务器保留足够的内核资源供其使用。
- 网络配置和流量

  依赖于网络与数据库服务器进行通信的应用程序，以及依靠数据复制保持高可用性的系统都会受限于网络的性能约束。网络上的数据传输通常比磁盘中的数据传输要慢。网络延迟对数据库服务器和运行在主机上的其他应用程序的性能具有显著影响。
- 数据库服务器配置

  数据库服务器实例的特征（例如，CPU 虚拟处理器 (VP) 的数量、驻留及虚拟共享内存部分的大小，以及用户数）对于决定应用程序的容量和性能方面具有重要作用。
- 数据库空间、BLOB 空间和块配置

  以下因素可能会影响数据库服务器执行磁盘 I/O 和处理事务所耗用的时间：

  - 根数据库空间、物理日志、逻辑日志和临时表数据库空间的位置
  - 镜像是否存在
  - 操作系统已缓冲或未缓冲设备的使用
- 数据库和表位置

  数据库空间内表和分段的位置、单独的数据库空间中使用率高的分段的隔离，以及分段在多个数据库空间中的分布都会影响数据库服务器找到数据页并将其传输到内存的速度。
- 表空间组织和扩展数据块缩放

  分段存储策略以及扩展数据块的大小和位置会影响数据库服务器快速扫描表以查找数据的能力。避免交错的扩展数据块并且分配足够的扩展数据块以满足表的增长，可防止发生性能问题。
- 查询效率

  正确的查询结构和游标使用可以减少任何一个应用程序或用户施加的负载。提醒用户和应用程序开发者注意，其他人需要有对数据库的访问权，以及每个人的活动都会对其他人可用的资源产生影响。

- 调度后台 I/O 活动

  日志记录、检查点、页清除和其他操作（如进行备份或执行大的决策支持查询）都可能对系统施加持续的开销和很大的临时负载。尽可能在非高峰时间调度备份和批处理操作。
- 远程客户端/服务器操作和分布式连接操作

  这些操作对性能有重要影响，尤其是在协调分布式连接的主机系统上。
- 应用程序代码效率

  应用程序将自己的负载引进操作系统、网络和数据库服务器。如果这些程序不能很好地使用系统资源、产生过度的网络通信量或在数据库服务器中产生不必要的争用，那么就可能导致性能问题。应用程序开发者必须正确使用游标和锁定级别以确保良好的数据库服务器性能。

保持良好性能

所有系统用户（数据库服务器管理员、数据库管理员、应用程序设计人员和客户端应用程序用户）都会以某种方式影响性能。

数据库服务器管理员通常要协调所有用户的活动以确保系统性能达到总体预期效果。例如，操作系统管理员可能需要重新配置操作系统以增加共享内存量。关闭操作系统以安装新的配置需要关闭数据库服务器。数据库服务器管理员必须对这个停机时间作出安排，并向所有受影响的用户通知系统将在何时不可用。

数据库服务器管理员应该：

- 了解发生的所有与性能相关的活动。
- 使用户了解性能的重要性、性能相关的活动是如何影响他们的，以及如何协助以获得并保持最佳性能。

数据库管理员应该注意：

- 表和查询如何影响数据库服务器的总体性能
- 表和分段的位置
- 数据在磁盘上的分布是如何影响性能的

应用程序开发者应该：

- 仔细设计应用程序，使用数据库服务器提供的并行和排序工具，而不是尝试实施应用程序中的类似工具。
- 将锁的作用域和持续时间保持在最小值，以避免对数据库资源的争用。
- 在应用程序中包含例程，在运行时临时启用时，这些例程允许数据库服务器管理员监视响应时间和事务吞吐量。

数据库用户应该：

- 注意性能并及时向数据库服务器管理员报告问题。
- 在调度大的决策支持查询和请求时保持谦让，使用尽可能少的资源来完成工作。

第 2 章

性能监视和使用的工具

您可以使用性能监视工具来创建性能历史记录，按调度时间监视数据库资源，或者监视正在进行的事务或查询性能。

本章还包含对有关如何解读性能监视结果的主题的交叉引用

需要收集的数据类型取决于系统中运行的应用程序类型。OLTP（联机事务处理）系统上性能问题的原因与主要用于 DSS 查询应用程序的系统上问题的原因有所不同。对于多用途系统，其性能调优更加困难，需要对性能问题的起因进行周密分析。

评估当前配置

在开始调整数据库服务器配置之前，请评估当前配置的性能。您可以使用 onstat 命令或 SinoDB® 开放管理工具 (OAT) 查看配置文件的内容。

要更改某个数据库服务器特性，必须关闭该数据库服务器，这会影响生产系统。某些配置调整会在无意中降低性能或导致其他负面影响。

如果数据库应用程序符合用户的期望，请避免频繁的调整，即使这些调整理论上可能会提高性能。如果用户相当满意，请采取标准方法来重新配置数据库服务器。如果可能，请在重新配置生产系统之前使用数据库服务器的测试实例评估所要进行的配置更改。

当性能问题与备份操作有关时，也可能要检查磁带驱动器的数量或传输速率。可能需要改变表的布局或分段存储以减小备份操作的影响。有关磁盘布局和表分段存储的信息，请参阅表性能的注意事项 在第116页和索引和索引性能注意事项 在第151页。

对于客户端/服务器配置，请考虑网络性能和可用性。对网络性能的评估不属于本出版物的内容。有关监视网络活动和提高网络可用性的信息，请咨询网络管理员或参阅联网软件包的文档。

请确定是否想要通过在数据库服务器运行时自动调整其属性，以设置一些配置参数来帮助保持服务器的性能，例如，

- AUTO_AIOVPS：在 I/O 工作负载增加时添加 AIO 虚拟处理器。
- AUTO_CKPTS：增加检查点的频率以避免事务阻塞。
- AUTO_LRU_TUNING：根据服务器负载的变化，对高速缓存数据清空操作进行管理。
- AUTO_READAHEAD：更改自动预读方式或禁用查询的自动预读操作。
- AUTO_REPREPARE：在模式更改之后，重新优化 SPL 例程和重新编译预编译对象。
- AUTO_STAT_MODE：在 UPDATE STATISTICS 操作中启用或禁用只选择性更新陈旧或丢失数据分布的方式。
- AUTO_TUNE：启用或禁用所有自动调整配置参数（这些参数的值不存在于配置文件中）。
- DYNAMIC_LOGS：根据需要分配额外的日志文件。
- LOCKS：根据需要分配额外的锁。
- RTO_SERVER_RESTART：在出现问题之后达到恢复时间目标的同时提供尽可能最佳的性能。

相关链接

《SinoDB 管理员参考》: onstat -c 命令: 显示 ONCONFIG 文件内容

《*SinoDB* 管理员参考》: *onstat -g cfg* 命令: 显示配置参数的当前值

创建性能历史记录

一旦安装好数据库服务器并开始在其中执行应用程序，就应该开始对资源使用执行已调度的监视。累计数据时，可以分析性能信息。

为了积累性能分析数据，可以在操作脚本或批处理文件中使用数据库服务器工具 在第30页和操作系统工具 在第29页中描述的命令行实用程序。

性能历史记录的重要性

如果具有系统性能的历史记录，那么可以在用户报告响应慢或吞吐量不足时立即跟踪问题的原因。

如果没有历史记录，那么必须在问题发生后开始跟踪性能，并且可能无法确定问题发生的时间以及如何发生。如果在发生问题之后才尝试发现问题，就会严重延迟性能问题的解决。

为了给系统建立性能历史记录和概要文件，必须对资源利用率信息进行定期快照。

例如，如果将系统中各个磁盘的 CPU 利用率、页换出速率和 I/O 传输速率绘成图表，您可以开始识别峰值使用级别、峰值使用时间间隔以及大量负载的资源。

如果监视分段使用，则可以确定分段存储方案的配置是否正确。适当为数据库服务器配置以及在其中运行的应用程序监视其他资源的使用情况。

从以下部分的描述中选择工具，并创建用于构建磁盘、内存、I/O 和其他数据库服务器资源使用的历史记录的作业。为了帮助决定创建性能历史记录的工具，本章简要描述每个工具的输出。

创建性能历史记录的工具

在监视数据库服务器性能时，可以使用主机操作系统和命令行实用程序中的工具，您可以通过脚本或批处理文件定期执行这些工具。

在执行查询和事务时，也可以使用带图形界面的性能监视工具监视关键性能。

操作系统工具
数据库服务器依靠主机的操作系统提供对系统资源（例如，CPU、内存和各种未缓冲磁盘 I/O 接口和文件）的访问。每个操作系统都有其自己的一套实用程序来报告系统资源的使用方式。

某些操作系统的不同实现具有相同名称但不同选项和信息显示的监视实用程序。

仅适用于 UNIX™

下表列出监视系统资源的一些 UNIX™ 实用程序。

UNIX™ 实用程序	描述
vmstat 实用程序	显示虚拟内存统计信息
iostat 实用程序	显示 I/O 利用率统计信息
sar 实用程序	显示各种资源统计信息
ps 实用程序	显示活动进程信息

有关如何监视操作系统资源的详细信息，请参阅参考手册或系统管理指南。

要定期捕获系统资源的状态，可使用主机操作系统中提供的调度工具（如 cron）作为性能监视系统的一部分。

仅适用于 Windows™

可以经常使用 Windows™ 操作系统提供的性能日志和警报来监视资源，例如处理器、内存、高速缓存、线程和进程。性能日志和警报也提供图表、警报、报告以及将信息保存到日志文件供日后分析的功能。

有关如何使用性能日志和警报的更多信息，请参阅操作系统手册。

数据库服务器工具
数据库服务器提供工具和实用程序以捕获有关配置和性能的快照信息。

可以定期使用这些实用程序构建数据库活动的历史概要文件，以便与当前操作系统资源利用率数据相比较。这些比较有助于发现对系统资源利用率影响最大的数据库服务器活动。可以使用此信息识别并管理影响较大的活动，或者调整数据库服务器或操作系统的配置。

可用于性能监视的数据库服务器工具和实用程序包括：

- SinoDB® 开放管理工具 (OAT)
- onstat 实用程序
- onlog 实用程序
- oncheck 实用程序
- DB-Access 和系统监视接口 (SMI)，可用于监视应用程序内的性能
- SQL 管理 API 命令

您可以使用由 cron 调度工具调用的 onstat、onlog 或 oncheck 命令来定期捕获与性能相关的信息并构建数据库服务器应用程序的历史性能概要文件。以下部分描述这些实用程序。

使用 SQL SELECT 语句可以从应用程序查询系统监视接口 (SMI)。

SMI 表是 sysmaster 数据库（包含数据库服务器操作的动态更新信息）中表和伪表的集合。数据库服务器在内存中构造这些表但并不将其记录到磁盘上。onstat 实用程序选项从这些 SMI 表中获得信息。

您可以使用 cron 和带 DB-Access 或 onstat 实用程序选项的 SQL 脚本来定期查询 SMI 表。

> 提示： SMI 表与系统目录表不同。系统目录表包含有关每个数据库及其表（有时称为元数据或数据字典）的永久性存储信息以及更新信息。

相关链接

《SinoDB 管理员参考》: onstat 实用程序
《SinoDB 管理员参考》: onlog 实用程序
《SinoDB 管理员参考》: oncheck 实用程序
DB-Access User's Guide
《SinoDB 管理员参考》: 系统监视接口表
《SinoDB SQL 指南: 参考》: 系统目录表
《SinoDB 管理员参考》: SQL 管理 API 门户: 按权限组划分参数

onstat 实用程序显示的性能信息
onstat 实用程序显示 SMI 表中包含的广泛性能相关信息和状态信息。可以使用 onstat 实用程序检查数据库服务器的当前状态并监视该数据库服务器的活动。

要获取所有 onstat 选项的完整列表，请使用 onstat - - 命令。有关 onstat 收集的所有信息的完整显示，请使用 onstat -a 命令。

> 提示： onstat 命令显示的概要文件信息（例如 onstat -p）从数据库服务器启动时开始累积。要清除性能概要文件统计信息以便可以创建新的概要文件，请执行 onstat -z。如果使用 onstat -z 复位性能历史记录或评估方面的统计信息，请确保其他用户不会在不同时间间隔输入该命令。

下表列出显示与性能相关的一般信息的一些 onstat 命令。

表 1: 显示性能信息的 onstat 命令

onstat 命令	描述
onstat -p	显示性能概要文件，其中包括读取和写入次数、请求使用某一资源但失败的次数以及其他各种信息
onstat -b	显示有关当前使用的缓冲区的信息

onstat 命令	描述
onstat -l	显示有关物理日志和逻辑日志的信息
onstat -x	显示有关事务的信息，其中包括拥有该事物的用户的线程标识符
onstat -u	显示用户活动概要文件，该文件提供有关用户线程的信息（包括线程所有者的会话标识符和登录名）
onstat -R	显示有关缓冲池的信息，包括有关缓冲池页大小的信息。
onstat -F	显示页清除统计信息，其中包括将页清空到磁盘的每种类型的写入数
onstat -g	需要指定要显示信息的另外一个参数 例如，onstat -g mem 显示内存统计信息。

有关提供与性能相关的信息的选项的更多信息，请参阅使用 *onstat -g ppf* 命令监视分段存储 在第209页和监视数据库服务器资源 在第31页。

相关链接

《*SinoDB* 管理员参考》: *onstat -g* 监视选项

使用 SinoDB® 开放管理工具 (OAT) 监视性能

SinoDB® 开放管理工具 (OAT) 提供多种方式以收集、查看和分析性能数据。

使用 OAT，您可以执行以下操作：

- 收集性能统计信息。
- 查找并消除数据库服务器性能瓶颈。
- 标识和监视对性能起关键作用的查询。
- 提高检查点性能和管理 LRU 队列。
- 管理对表和表分段行中数据的压缩
- 监视关键系统资源（CPU、内存、磁盘和虚拟处理器）。
- 监视与跟踪锁定。
- 优化磁盘布局。
- 调整缓冲区高速缓存。
- 使用向下钻取查询。
- 查看 SQL 语句高速缓存。
- 使用自动统计信息更新。
- 查看历史性能图形。
- 浏览用户会话。

监视数据库服务器资源

监视特定数据库服务器资源以确定性能瓶颈以及潜在的故障点，并改善资源使用情况和响应时间。

监视系统资源的最有用命令之一是 onstat -g 及其众多选项。

监视影响 CPU 利用率的资源

线程、网络通信和虚拟处理器会影响 CPU 利用率。可以使用 onstat -g 参数来监视线程、网络通信和虚拟处理器。

使用以下 onstat -g 命令选项来监视线程。

onstat -g 选项	描述
act	显示活动的线程。
ath	显示所有线程。 sqlexec 线程代表客户端会话部分；rstcb 值与 onstat -u 命令的用户字段相对应。
cpu	显示上次执行线程的时间、线程使用了多少 CPU 时间、线程执行的次数以及有关服务器中执行的所有线程的其他统计信息。
rea	显示就绪线程。
sle	显示所有睡眠的线程。
sts	显示每个线程使用的最大堆栈和当前堆栈。
tpf *tid*	显示 *tid* 的线程概要文件。 如果 *tid* 为 0，那么此参数显示所有线程的概要文件。
wai	显示等待的线程，包括等待互斥或条件或要让出的所有线程。

使用以下 onstat -g 命令选项来监视网络。

onstat -g 命令选项	描述
ntd	按服务显示网络统计信息。
ntt	显示网络用户时间。
ntu	显示网络用户统计信息。
qst	显示队列统计信息。

使用以下 onstat -g 命令选项来监视虚拟处理器。

onstat -g 命令选项	描述
glo	显示全局多线程信息，包括有关虚拟处理器的 CPU 使用信息、会话总数以及其他多线程全局计数器。
sch	显示每个 VP 的信号量操作次数、循环次数以及忙碌等待的次数。
spi	显示虚拟处理器旋转超过 10000 次后获取的旋转锁。 要减少争用，可以减少虚拟处理器的数量、降低计算机的负载，或者在某些平台上使用虚拟处理器的 no-age 或 processor affinity 选项。如果 sh_lock 互斥锁具有高度竞争的旋转锁，那么可以通过设置 VP_MEMORY_CACHE_KB 配置参数为 CPU 虚拟处理器创建专用内存高速缓存。

onstat -g 命令选项	描述
wst	显示等待统计信息。

监视内存利用率

可以使用一些特定 onstat -g 命令选项来监视内存利用率。

使用以下 onstat -g 选项来监视内存利用率。要获取总体内存信息，请在允许那些可选参数的命令中省略 *table name*、*pool name* 或 *session id*。

表 2: 用于监视内存利用率的 onstat -g 选项

参数	描述
ffr *pool name* | *session id*	为共享内存池或者会话显示可用的分段
dic *table name*	为共享内存字典中高速缓存的每个表显示一行信息 如果提供了特定表名作为参数，那么此参数将显示有关该表的内部 SQL 信息。
dsc	为数据分布高速缓存中高速缓存的分布统计信息的每列显示一行信息。
mem *pool name* | *session id*	显示与会话相关的池的内存统计信息 如果省略 *pool_name* | *session id*，那么该参数将显示所有会话的池信息。
mgm	显示内存分配管理器资源信息，包含： • PDQ 配置参数的值 • 内存和扫描信息 • 载入信息，如正在等待内存的查询数量、正在等待扫描的查询数量、正在等待以更高 PDQ 优先级执行的查询数量以及正在等待查询槽的查询数量。 • 活动的查询以及每个门的查询数量 • 自由资源的统计信息 • 查询的统计信息 • 资源/锁循环预防计数，显示系统为避免潜在死锁而马上激活查询的次数
nsc *client id*	按客户端标识符显示共享内存状态 如果省略 *client id*，那么该参数显示所有客户端状态区域。
nsd	显示池线程的网络共享内存
nss *session id*	按会话标识符显示网络共享内存状态 如果省略 session id，那么该参数显示所有会话状态区域。
osi	显示有关您的操作系统资源和参数的信息，包含共享内存和信号量参数、计算机上当前配置的内存量以及未使用的内存量 当服务器未联机时使用此选项。

参数	描述
prc	为 UDR 高速缓存中高速缓存的每个用户定义例程（SPL 例程或者用 C 或 Java™ 编程语言编写的外部例程）分别显示一行信息
seg	显示共享内存段统计信息 该参数显示所有附加段的数量和大小。
ses *session id*	显示会话标识符的内存使用情况 如果省略 session id，那么此参数将显示所有会话的内存使用情况。
ssc	为 SQL 语句高速缓存中高速缓存的每个查询分别显示一行信息
stm *session id*	显示会话标识符的每个 SQL 语句的内存使用情况 如果省略 session id，那么此参数将显示所有会话的内存使用情况。
ufr *pool name* | *session id*	按用户或会话显示分配的池分段

相关链接

《*SinoDB* 管理员参考》*: onstat -g* 监视选项

监视磁盘 I/O 利用率

您可以使用某些特定 onstat -g 参数和 oncheck 实用程序来确定磁盘 I/O 操作对应用程序是否高效。

使用 onstat -g 监视 I/O 利用率

可以使用一些特定 onstat -g 命令参数来监视磁盘 IO。

使用以下 onstat -g 命令参数来监视磁盘 I/O 利用率。

onstat -g 参数	描述
iof	按块或文件显示异步 I/O 统计信息 此参数与 onstat -d 相似，不同的是也显示有关非块文件的信息。此参数显示有关临时数据库空间和排序文件的信息。
iog	显示异步 I/O 全局信息
ioq	显示异步 I/O 排队统计信息
iov	按虚拟处理器显示异步 I/O 统计信息

有关使用各种 onstat 输出的详细案例研究，请参阅案例分析和示例 在第311页。

使用 oncheck 实用程序监视 I/O 利用率

磁盘 I/O 操作通常是查询响应时间最长的部分。可以使用 oncheck 实用程序来监视磁盘 I/O 操作。

连续分配的磁盘空间可改善连续的磁盘 I/O 操作，因为数据库服务器可读取较大的数据块并使用预读取功能来减少 I/O 操作数。

oncheck 实用程序显示有关磁盘上存储结构的信息，包括块、数据库空间、BLOB 空间、扩展数据块、数据行、系统目录表和其他选项的信息。也可以使用 oncheck 确定表中存在的扩展数据块数以及表是否占用连续空间。

oncheck 实用程序提供以下适用于连续空间和扩展数据块的选项和信息。

选项	信息
-pB	BLOB 空间简单大对象（TEXT 或 BYTE 数据） 有关如何使用此选项来确定 BLOB 页大小效率的信息，请参阅使用 *oncheck -pB* 输出确定 *BLOB* 页填充度 在第96页。
-pe	块和扩展数据块 有关如何使用此选项来监视扩展数据块的信息，请参阅检查扩展数据块的交错情况 在第133页和消除交错的扩展数据块 在第134页。
-pk	索引键值。 有关如何提高此选项性能的信息，请参阅提高索引检查的性能 在第166页。
-pK	索引键和行标识符 有关如何提高此选项性能的信息，请参阅提高索引检查的性能 在第166页。
-pl	索引叶键值 有关如何提高此选项性能的信息，请参阅提高索引检查的性能 在第166页。
-pL	索引叶键值和行标识符 有关如何提高此选项性能的信息，请参阅提高索引检查的性能 在第166页。
-pp	按表或段分页 有关如何使用此选项来监视空间的信息，请参阅考虑扩展数据块数的上限 在第133页。
-pP	按块分页 有关如何使用此选项来监视扩展数据块的信息，请参阅考虑扩展数据块数的上限 在第133页。
-pr	根保留页 有关如何使用此选项的信息，请参阅估算具有固定长度行的表 在第119页。
-ps	智能大对象空间中智能大对象和元数据使用的空间。
-pS	智能大对象空间中由智能大对象和元数据使用的空间以及存储特征 有关如何使用此选项来监视空间的信息，请参阅监视智能大对象空间 在第125页。
-pt	表或段使用的空间 有关如何使用此选项来监视空间的信息，请参阅估算表大小 在第118页。
-pT	表使用的空间，包括索引 有关如何使用此选项来监视空间的信息，请参阅 *DDL* 操作的定点变更性能 在第145页。

有关使用 oncheck 来监视空间的更多信息，请参阅估算表大小 在第118页。有关在 oncheck 执行期间并行的更多信息，请参阅提高索引检查的性能 在第166页。

相关链接

《*SinoDB* 管理员参考》: *oncheck* 实用程序

监视事务

您可以使用 onlog 和 onstat 实用程序来监视事务。

使用 onlog 实用程序监视事务

onlog 实用程序显示逻辑日志的全部或选定部分。此实用程序有助于识别有问题的事务或判断与高利用率时期相对应的事务活动，如数据库活动和系统资源消耗的周期性快照所示。

此 onlog 实用程序可以从选定的日志文件、整个逻辑日志或先前日志文件的备份磁带获取输入。

当读取仍在磁盘中的逻辑日志文件时，必须谨慎使用 onlog，因为试图读取未发布的日志文件会使其他数据库活动停止。为获得最大程度的安全性，请先备份逻辑日志文件，然后读取备份文件的内容。如果能做到小心谨慎，那么可以使用 onlog -n 选项限制 onlog 仅用于已发布的逻辑日志文件。

要检查逻辑日志文件的状态，可使用 onstat -l。

相关链接

《SinoDB 管理员参考》: onlog 实用程序

使用 onstat 实用程序监视事务

如果事务的吞吐量不太大，可使用一些 onstat 实用程序命令来识别可能是瓶颈的事务。

使用以下 onstat 实用程序命令来监视事务。

onstat 命令	描述
onstat -x	显示事务信息，如持有的锁数量和隔离级别。
onstat -u	显示有关每个用户线程的信息
onstat -k	显示每个会话持有的锁数
onstat -g sql	显示此会话执行的最后一条 SQL 语句

相关链接

《SinoDB 管理员参考》: onstat 实用程序

监视会话和查询

监视会话和线程对于执行查询的会话以及执行插入、更新和删除操作的会话都很重要。可监视会话和线程的某些信息来确定应用程序使用的资源量是否比例不当。

要监视数据库服务器活动，可以查看活动会话的数量和它们正使用的资源量。

监视每个会话的内存使用情况

可以使用某些特定 onstat -g 命令参数来获取每个会话的内存信息。

使用以下命令参数来获取每个会话的内存信息。

onstat -g 命令参数	描述
ses	在一行中显示所有活动会话的摘要信息
ses *session id*	按 *session id* 显示会话信息
sql *session id*	按会话显示 SQL 信息 如果省略 *session id*，那么此参数将显示所有会话的摘要。

onstat -g 命令参数	描述
stm *session id*	显示会话中每个预编译 SQL 语句使用的内存量 如果省略 *session id*，那么此参数将显示所有预编译语句的信息。

有关会话监视的命令行实用程序的示例和讨论，请参阅监视每个会话的内存使用情况 在第297页和监视会话和线程 在第301页。

使用 SET EXPLAIN 语句

可以使用 SET EXPLAIN 语句或 EXPLAIN 指令来显示优化器为单个查询创建的查询计划。

有关更多信息，请参阅显示查询计划 在第265页。

第 3 章

配置对 CPU 利用率的影响

操作系统和 SinoDB® 配置参数共同影响 CPU 利用率。您可以更改直接影响 CPU 利用率的 SinoDB® 配置参数的设置，并且可以调整不同类型工作负载的设置。

与管理多个数据库的单个数据库服务器实例相比，在同一主机上运行的多个数据库服务器实例性能较差。多个数据库服务器实例无法像单个数据库服务器那样有效平衡其负载。在性能极为重要的生产环境上避免运行多个数据库实例。

影响 CPU 利用率的 UNIX™ 配置参数

您的数据库服务器分布包括一个机器说明文件，其中包含 UNIX™ 配置参数的推荐值。因为 UNIX™ 参数影响 CPU 利用率，所以应该将机器说明文件中的值与当前操作系统配置相比较。

以下 UNIX™ 参数会影响 CPU 利用率：

- 信号量参数
- 设置打开文件描述符最大数量的参数
- 内存配置参数

UNIX™ 信号量参数

信号量通常是每个大小为 1 字节的内核资源。数据库服务器的信号量不包含在为其他软件包分配的资源之内。您可以设置某些 UNIX™ 信号量参数。

每个数据库服务器实例都需要以下几组信号量：

- 一组用于与数据库服务器一起启动的每组虚拟处理器 (VP)（多达 100 个）
- 一组用于可在数据库服务器运行时动态添加的每个额外 VP
- 一组用于通过共享内存通信接口连接的每组用户会话（不超过 100）

> 提示： 为了获取最佳性能，请分配足够的信号量，以使期望的 ipcshm 连接数增加一倍。请使用 NETTYPE 配置参数来为此两倍数量的连接配置数据库服务器轮询线程。

由于实用程序（例如，onmode）使用共享内存连接，因此您必须为数据库服务器的每个实例配置最少两组信号量集：一组用于初始集的 VP，一组用于数据库服务器实用程序使用的共享内存连接。SEMMNI 操作系统配置参数通常会指定要分配的信号量集数。有关如何设置与信号量相关参数的信息，请参阅操作系统的配置指示信息。

SEMMSL 操作系统配置参数通常会指定每组信号量的最大数。将此参数设置为至少 100。

某些操作系统需要您配置所有组中信号量的最大总数，该值通常由 SEMMNS 操作系统配置参数指定。使用以下公式计算每个数据库服务器实例需要的信号量总数：

```
SEMMNS = init_vps + added_vps + (2 * shmem_users) + concurrent_utils
```

init_vps

与数据库服务器一起启动的虚拟处理器 (VP) 数。此数值包含 CPU、PIO、LIO、AIO、SHM、TLI、SOC 和 ADM VP。最小值为 15。

added_vps

您要动态添加的 VP 数。

shmem_users

允许用于此数据库服务器实例的共享内存连接数。

concurrent_utils

可以连接到此实例的并发数据库服务器实用程序数。建议您最少允许使用六个实用程序连接：两个用于 ON-Bar，其余四个用于其他实用程序，如 onstat 和 oncheck。

如果使用需要信号量的软件包，那么 SEMMNI 配置参数必须包含数据库服务器和其他软件包需要的信号量集总数。您必须将 SEMMSL 配置参数设置为任何软件包都需要的每组最大信号量数。有关需要 SEMMNS 配置参数的系统，可以用 SEMMNI 乘以 SEMMSL 的值来计算可接受值。

相关链接

配置轮询线程 在第46页

UNIX™ 文件描述符参数

某些操作系统要求指定一个进程一次可以打开文件描述符数的限值。为指定该限值，可以使用操作系统配置参数，通常是 NOFILE、NOFILES、NFILE 或 NFILES。

数据库服务器的每个实例需要打开文件描述符的数量取决于数据库中的块数、运行的 VP 数以及数据库服务器实例必须支持的网络连接数。

使用以下公式计算数据库服务器实例需要的文件描述符数：

```
NFILES = (chunks * NUMBER_OF_AIO_VPS) + NUMBER_of_CPU_VPS + net_connections
```

chunks

要配置的块数。

net_connections

您在以下其中一个位置指定的网络连接数：

- sqlhosts 文件
- NETTYPE 配置条目

网络连接包括指定为 ipcshm 连接类型以外的所有类型。

每个打开文件描述符的长度大约等于内核中的整数。要增加系统允许的块数或连接数，分配额外的文件描述符是一种简便的方法。

UNIX™ 内存配置参数

操作系统中的内存配置可以影响其他资源，包括 CPU 和 I/O。

整个系统负载的物理内存不足可能导致系统颠簸，如内存利用率 在第24页所述。数据库服务器的内存不足会导致过多的缓冲区管理活动。有关配置内存的更多信息，请参阅配置 *UNIX* 共享内存 在第59页。

影响 CPU 利用率的 Windows™ 配置参数

SinoDB® 分布包括一个机器说明文件，该文件包含 Windows™ 上 SinoDB® 配置参数的推荐值。将此文件中的值与当前 ONCONFIG 配置文件设置进行比较。

SinoDB® 在后台运行。要获取最佳性能，需要为前台和后台应用程序赋予同样的优先级。

在 Windows™ 上，要更改前台和后台应用程序的优先级，请转至 Start > Settings > Control Panel，打开 System 图标并单击 Advanced Tab。选择 Performance Options 按钮并选择 Applications 或 Background Services 单选按钮。

操作系统中的内存配置可以影响其他资源，包括 CPU 和 I/O。整个系统负载的物理内存不足可能导致系统颠簸，如内存利用率 在第24页所述。SinoDB® 的内存不足会导致过多的缓冲区管理活动。当在 Control Panel 上的 System 图标中设置 Virtual Memory 值时，请确保有足够的分页空间用于全部物理内存。

影响 CPU 利用率的配置参数和环境变量

某些配置参数和环境变量会影响 CPU 利用率。在考虑改善性能的方法时，可能需要调整这些参数和变量的设置。

数据库服务器配置文件中的以下配置参数会对 CPU 利用率产生显著的影响：

- DS_MAX_QUERIES
- DS_MAX_SCANS
- FASTPOLL
- MAX_PDQPRIORITY
- MULTIPROCESSOR
- NETTYPE
- OPTCOMPIND
- SINGLE_CPU_VP
- VPCLASS
- VP_MEMORY_CACHE_KB

以下环境变量会影响 CPU 利用率：

- OPTCOMPIND
- PDQPRIORITY
- PSORT_NPROCS

当在客户端应用程序的环境中设置 OPTCOMPIND 环境变量时，该环境变量表示执行连接操作的首选方式。此变量将覆盖 OPTCOMPIND 配置参数设置的值。有关如何选择首选连接方法的详细信息，请参阅优化访问方法 在第44页。

在客户端应用程序的环境中设置 PDQPRIORITY 环境变量时，该环境变量将限定 CPU VP 利用率、共享内存，以及可以分配到客户端启动的任何查询的其他资源的百分比。

客户端也可以使用 SQL 中的 SET PDQPRIORITY 语句来设置 PDQ 优先级的值。分配给任何查询的实际百分比取决于 MAX_PDQPRIORITY 配置参数设置的因子。有关如何限制可以分配给查询的资源的更多信息，请参阅限制查询中的 *PDQ* 资源 在第45页。

当在客户端应用程序的环境中设置 PSORT_NPROCS 时，该环境变量表示该应用程序可以使用的并行排序线程数。数据库服务器将对所有应用程序的每个查询使用的排序线程数强制限制在 10 个以下。有关并行排序和 PSORT_NPROCS 的更多信息，请参阅为临时表和排序文件配置数据库空间 在第90页。

相关链接

《SinoDB 管理员参考》: 数据库配置参数

《SinoDB SQL 指南: 参考》: 环境变量

指定虚拟处理器类信息

使用 VPCLASS 配置参数来指定虚拟处理器类、数据库服务器应该为特定类启动的虚拟处理器数，以及所允许的最大数目。

要执行用户定义例程 (UDR)，您可以定义新的虚拟处理器类将 UDR 执行从其他在 CPU 虚拟处理器上执行的事务中隔离出来。通常，编写用户定义例程来支持用户定义数据类型。

如果不希望用户定义例程影响 CPU 类中用户查询的正常处理，您可以使用 CREATE FUNCTION 语句将该例程分配给用户定义的虚拟处理器类。在 VPCLASS 配置参数中指定的类名必须与 CREATE FUNCTION 语句的 CLASS 修饰符中指定的名称相匹配。

有关 VPCLASS 配置参数的 cpu 和 num 选项的使用准则，请参阅设置 *CPU VP* 数 在第41页。

相关链接

《SinoDB 管理员参考》: VPCLASS 配置参数

《SinoDB SQL 指南: 语法》: CREATE FUNCTION 语句

设置 CPU VP 数

您可以配置数据库服务器使用的 CPU 虚拟处理器数 (VP)。分配的 CPU VP 数不能超过可用于为其服务的 CPU 处理器数。

当数据库服务器启动时，CPU VP 数会自动增加到数据库服务器计算机上 CPU 处理器数的一半，除非 SINGLE_CPU_VP 配置参数已启用。但是，您可能希望基于性能需要而更改 CPU VP 数。

您可以使数据库服务器根据需要添加 CPU VP，最多可达到计算机上 CPU 处理器数。在 VPCLASS 设置中包含 autotune=1 选项：

```
VPCLASS cpu,autotune=1
```

如果未将 VPCLASS 配置参数设置为 autotune=1，请遵循以下准则来设置 CPU VP 数：

单处理器计算机

对于单处理器计算机，请指定一个 CPU VP：

```
VPCLASS cpu,num=1
```

双处理器计算机

对于双处理器系统，您可以使用两个 CPU VP 运行来提高性能。要测试性能是否提高，请在 onconfig 文件中将 VPCLASS 配置参数的 num 字段设为 1，然后在运行时执行 onmode -p 命令动态添加 CPU VP。

主要是数据库服务器的多处理器计算机

对于主要用作数据库服务器且带有四个或多个 CPU 的多处理器系统，您应该在 onconfig 文件中将 VPCLASS 配置参数的 num 选项设为处理器总数减一。例如，如果有 4 个 CPU，那么使用以下规范：

```
VPCLASS cpu,num=3
```

当使用此设置时，会有一个处理器用于运行数据库服务器实用程序或客户端应用程序。

主要不是数据库服务器的多处理器计算机

对于并非主要用于支持数据库服务器的多处理器系统，您可以使用较少的 CPU VP 启动以允许在系统上执行其他活动，然后在必要时逐步添加更多的 CPU VP。

支持逻辑 CPU 的多核或硬件多线程计算机

要让使用多核处理器或硬件多线程的多处理器系统支持比物理处理器更多的逻辑 CPU，您可以根据可用于该用途的逻辑 CPU VP 数指定 CPU VP 数。其他逻辑 CPU 可提供的处理量可能只是专用物理处理器可支持的一小部分。

在安装多核处理器的系统上，大部分情况下，最佳配置与具有等于核心总数的单个处理器数的系统所考虑配置相同。将 CPU VP 数设为 N-1（其中 N 是核心数），对于 CPU 密集型工作负载而言，应该接近最佳效果。

在 CPU 对每个核心使用多线程的计算机上，操作系统将显示比实际处理核心更多的逻辑处理器。要利用更多 CPU 线程，必须使用在 N 与 M 范围之间（其中 N 是核心数，且 M 是系统报告的逻辑 CPU 总数）的 CPU VP 数配置数据库服务器。实现最佳性能的 CPU VP 数将取决于工作负载。

当增加 CPU VP 数来对每个核心使用更多线程时，性能的预期提高仅是专用物理处理器或核心可提供的一小部分。

如果要将 SinoDB® 从多 CPU/多核系统迁移到每个核心有多个线程的系统，请特别注意处理器亲缘关系。将 SinoDB® CPU VP 绑定到操作系统的逻辑处理器时，您必须注意 CPU 的体系结构。如果您不确定，应该禁用 CPU 亲缘关系，以允许操作系统将 CPU VP 调度为具有可用资源的逻辑处理器。使用亲缘关系而不了解逻辑 CPU 与处理核心之间的关系可能会导致性能严重下降。

例如，要将 8 个已配置 CPU VP 中的每一个绑定到每个核心有 2 个线程的 8 核系统（16 个逻辑 CPU）上的不同核心，请使用以下设置：

```
VPCLASS cpu,num=8,aff=(0-14/2)
```

相关链接

《SinoDB 管理员参考》：VPCLASS 配置参数

对 CPU VP 禁用进程优先级老化

使用 VPCLASS 配置参数的 noage 选项来在支持此功能的操作系统上对数据库服务器 CPU VP 禁用进程优先级老化。由于长时间运行的进程的处理时间不断增加使得操作系统降低其优先级时，即发生优先级老化。您可能希望禁用优先级老化功能，因为它会导致数据库服务器进程的性能随着时间的推移而下降。

您的数据库服务器分布包括一个机器说明文件，该文件包含有关您的数据库服务器版本是否支持此功能的信息。

如果操作系统支持此功能，那么指定 VPCLASS 的 noage 选项。

相关链接

《SinoDB 管理员参考》：VPCLASS 配置参数

指定处理器亲缘关系

使用 VPCLASS 参数的 aff 选项来指定您希望绑定 CPU VP 或 AIO VP 的处理器。将 CPU VP 分配到特定的 CPU 时，该 VP 只在此 CPU 上运行。但是，其他进程也可以在此 CPU 上运行。

在支持处理器亲缘关系的多处理器主机上，数据库服务器支持将 CPU VP 自动绑定到处理器的功能。您的数据库服务器分布包括一个机器说明文件，该文件包含有关您的数据库服务器版本是否支持此功能的信息。

可以将处理器亲缘关系用于以下部分描述的用途。

相关链接

《SinoDB 管理员参考》：VPCLASS 配置参数

分散计算影响

您可以使用处理器亲缘关系来分散 CPU 虚拟处理器 (VP) 和其他进程的计算影响。在专门用于数据库服务器的计算机上，除了保留一个 CPU 以外，将 CPU VP 分配到其他所有的 CPU 会达到最大的 CPU 利用率。

在同时支持数据库服务器和客户端应用程序的计算机上，您可以通过操作系统将应用程序绑定到某些 CPU。通过执行此操作，您可以有效保留其余的 CPU 以供数据库服务器 CPU VP 使用，并使用 VPCLASS 配置参数将这些 CPU VP 绑定到其余的 CPU。将 VPCLASS 配置参数的 aff 选项设置为绑定 CPU VP 的 CPU 编号。例如，以下 VPCLASS 设置将 CPU VP 分配到处理器 4 至 7：

```
VPCLASS cpu,num=4,aff=(4-7)
```

指定一系列处理器时，也可以指定范围内的递增值，用于指示范围中应分配给虚拟处理器的 CPU。例如，可以指定将虚拟处理器分配给范围 0-6 中的每隔一个 CPU，从 CPU 0 开始。

```
VPCLASS CPU,num=4,aff=(0-6/2)
```

虚拟处理器将分配给 CPU 0、2、4、6。

如果指定 VPCLASS CPU,num=4,aff=(1-10/3)，将虚拟处理器分配给范围 1-10 中的每隔两个 CPU，从 CPU 1 开始。虚拟处理器将分配给 CPU 1、4、7、10。

指定多个值或范围时，这些值和范围不必是递增的，也不必按照任何特定顺序指定。例如，可以指定 aff=(8,12,7-9,0-6/2)。

数据库服务器以循环模式将 CPU 虚拟处理器分配给 CPU，从您在 *aff* 选项中指定的第一个处理器编号开始。如果指定的 CPU 虚拟处理器数多于物理 CPU 数，那么数据库服务器会继续从首个 CPU 开始分配 CPU 虚拟处理器。例如，假设您指定了以下 VPCLASS 设置：

```
VPCLASS cpu,num=8,aff=(4-7)
```

数据库服务器会进行以下分配：

- CPU 虚拟处理器编号 0 到 CPU 4
- CPU 虚拟处理器编号 1 到 CPU 5
- CPU 虚拟处理器编号 2 到 CPU 6
- CPU 虚拟处理器编号 3 到 CPU 7
- CPU 虚拟处理器编号 4 到 CPU 4
- CPU 虚拟处理器编号 5 到 CPU 5
- CPU 虚拟处理器编号 6 到 CPU 6
- CPU 虚拟处理器编号 7 到 CPU 7

相关链接

《*SinoDB* 管理员参考》：*VPCLASS* 配置参数

隔离 AIO VP 与 CPU VP

在运行数据库服务器和客户端（或其他）应用程序的系统上，您可以将异步 I/O (AIO) VP 绑定到某些 CPU（与通过操作系统将其他应用程序进程绑定的 CPU 相同）。通过这种方式，您可以将客户端应用程序和数据库 I/O 操作与 CPU VP 隔离。

当客户端进程用于数据输入或需要等待用户输入的其他操作时，这种隔离尤其有用。由于通常情况下，AIO VP 活动迅速突发，接着出现一段空闲期等待磁盘操作，因此您可以经常交错客户端和 I/O 操作，而不会使它们互相造成负面影响。

将 CPU VP 绑定到处理器不会妨碍其他进程在该处理器上运行。没有绑定到 CPU 的应用程序（或其他）进程可以自由地在任何可用的处理器上运行。在专门用于数据库服务器的计算机上，您可以使 AIO VP 能够自由运行在任何处理器上，这将减少等待 I/O 的数据库操作上的延迟。增加 AIO VP 的优先级，通过确保数据从磁盘到达后被快速处理的方式，可以进一步改善性能。

避免使用特定 CPU

数据库服务器将 CPU VP 依次分配到 CPU，从该参数中指定的 CPU 编号开始。您可能想要避免将 CPU VP 分配给具有专用硬件或操作系统功能（例如，中断处理）的特定 CPU。

设置 AIO VP 数

使用 VPCLASS 配置参数的 aio 和 num 选项来指示数据库服务器最初启动的 AIO 虚拟处理器的数量。

如果您的操作系统不支持内核异步 I/O (KAIO)，那么数据库服务器会使用 AIO 虚拟处理器 (VP) 管理所有数据库 I/O 请求。

如果 VPCLASS 配置参数没有指定在 onconfig 文件中启动的 AIO VP 数，那么最初启动 AIO VP 的数量等于使用 AIO 的块数，最大值为 128。

您可以启用数据库服务器以根据需要增加 AIO VP 的数量来提高性能。在 VPCLASS 配置参数设置中包含 autotune=1 选项：

```
VPCLASS aio,autotune=1
```

AIO 虚拟处理器的推荐数量取决于配置支持的磁盘数。如果在您的平台上没有实现 KAIO，那么建议您为包含数据库表的每个磁盘分配一个 AIO 虚拟处理器。可以为数据库服务器频繁访问的每个块添加附加的 AIO 虚拟处理器。

数据库服务器版本的机器说明文件会指示操作系统是否支持 KAIO。如果支持 KAIO，那么机器说明将描述如何在特定的操作系统上启用 KAIO。

如果您的操作系统支持 KAIO，那么 CPU VP 将向操作系统而不是 AIO 虚拟处理器发出异步 I/O 请求。在这种情况下，只能配置一个 AIO 虚拟处理器，以及为每个不使用 KAIO 的文件块配置两个额外 AIO 虚拟处理器。

如果使用熟文件（即已缓冲的文件）并且使用 DIRECT_IO 配置参数启用直接 I/O，那么您可以减少 AIO 虚拟处理器的数量。如果数据库服务器实现了 KAIO 并且启用了直接 I/O，那么数据库服务器将尝试使用 KAIO，因此您可能不需要多个 AIO 虚拟处理器。临时数据库空间不使用直接 I/O。如果您拥有临时数据库空间，那么可能需要多个 AIO 虚拟处理器。

即使使用 DIRECT_IO 配置参数启用了直接 I/O，如果文件系统不支持直接 I/O 或 KAIO，那么您仍必须为每个不使用 KAIO 的活动数据库空间块分配两个额外的 AIO 虚拟处理器。

分配 AIO 虚拟处理器的目的在于分配足够多的虚拟处理器以使 I/O 请求队列的长度保持简短（即队列中的 I/O 请求尽可能少）。如果 I/O 请求队列始终保持较短，那么会在 I/O 请求出现的同时对其进行处理。onstat -g ioq 命令可用于监视 AIO 虚拟处理器的 I/O 队列长度。

分配足够的 AIO VP 以适应 I/O 请求的峰值数。通常情况下，分配几个额外的 AIO VP 有益无害。要在数据库服务器处于联机方式时启动附加的 AIO VP，请使用 onmode -p 命令。在联机方式中，您无法删除 AIO VP。

相关链接

《SinoDB 管理员参考》: AUTO_AIOVPS 配置参数

《SinoDB 管理员参考》: VPCLASS 配置参数

使用多个 CPU VP 时设置 MULTIPROCESSOR 配置参数

如果正在运行多个 CPU VP，那么将 MULTIPROCESSOR 配置参数设置为 1。将 MULTIPROCESSOR 设置为 1 时，数据库服务器会按照适合多处理器的方式执行锁定操作。否则，将此参数设置为 0。

CPU VP 数用作确定查询的扫描线程数的因子。扫描线程数为 CPU VP 数的倍数（或因子）时，查询执行效果最佳。添加或移除一个 CPU VP 可以改善大型查询的性能，因为这将在 CPU VP 中均匀分布扫描线程。例如，如果具有 6 个 CPU VP 并扫描 10 个表分段，那么如果将 CPU VP 数减少到 5 个（被 10 整除），就会发现响应时间更快。可以使用 onstat -g ath 监视每个 CPU VP 的扫描线程数，或者可以使用 onstat -g ses 关注特定的会话。

相关链接

《SinoDB 管理员参考》: MULTIPROCESSOR 配置参数

使用一个 CPU VP 时设置 SINGLE_CPU_VP 配置参数

如果只执行一个 CPU VP，那么将 SINGLE_CPU_VP 配置参数设置为 1。否则，将该参数设置为 0。

> 重要： 如果将 SINGLE_CPU_VP 参数设置为 *1*，那么 VPCLASS 配置参数的 num 选项值也必须为 *1*。
>
> 注： 数据库服务器将用户定义的虚拟处理器类（即，使用 VPCLASS 定义的 VP）视为 CPU VP。因此，如果将 SINGLE_CPU_VP 设置为非零，那么将无法创建任何用户定义的类。

将 SINGLE_CPU_VP 参数设置为 1 时，您将无法在数据库服务器处于联机方式时添加 CPU VP。

相关链接

《SinoDB 管理员参考》: SINGLE_CPU_VP 配置参数

《SinoDB 管理员参考》: VPCLASS 配置参数

优化访问方法

OPTCOMPIND 配置参数有助于查询优化器为应用程序选择适当的访问方法。当优化器检查连接计划时，OPTCOMPIND 会指示执行顺序表对的连接操作的首选方法。

如果 OPTCOMPIND 等于 0，那么即使表扫描速度可能加快，优化器也将优先考虑现有索引（嵌套循环连接）。如果 OPTCOMPIND 设置为 1 并且给定查询的隔离级别设置为 Repeatable Read，那么优化器将使用嵌套循环连接。

当 OPTCOMPIND 等于 2 时，即使表扫描可以暂时锁定整个表，优化器也将只基于成本来选择连接方法。有关 OPTCOMPIND 和其他连接方法的更多信息，请参阅 *OPTCOMPIND* 对查询计划的影响 在第226页。

要为特定的应用程序或用户会话设置 OPTCOMPIND 的值，请设置这些会话的 OPTCOMPIND 环境变量。此环境变量的值与配置参数具有相同的范围和语义。

相关链接

《*SinoDB* 管理员参考》: *OPTCOMPIND* 配置参数

设置会话中 OPTCOMPIND 的值
可以在不同种类查询的会话中设置或更改 OPTCOMPIND 的值。为此，请使用 SET ENVIRONMENT OPTCOMPIND 语句，而不是 OPTCOMPIND 配置参数或 OPTCOMPIND 环境变量。

对于 DSS 查询，应该将 OPTCOMPIND 值设置为 2 或 1，并应该确保隔离级别未设置为 Repeatable Read。对于 OLTP 查询，可将该值设置为 0 或 1，而隔离级别不设置为 Repeatable Read。

您使用 SET ENVIRONMENT OPTCOMPIND 命令输入的值优先于在 ONCONFIG 文件中通过 OPTCOMPIND 环境变量或 OPTCOMPIND 配置参数指定的缺省设置。当发出 SET ENVIRONMENT OPTCOMPIND 语句的例程退出时，或直到同一例程通过发出以下语句将 OPTCOMPIND 值重置为系统缺省值时，将复原缺省 OPTCOMPIND 设置。

```
SET ENVIRONMENT OPTCOMPIND DEFAULT;
```

任何其他用户会话或例程都不受您执行的 SET ENVIRONMENT OPTCOMPIND 语句的影响，因为其范围对于其发布的例程而言是本地的，而不是整个会话。

相关链接

《*SinoDB SQL* 指南: 语法》: *OPTCOMPIND* 会话环境选项

限制查询中的 PDQ 资源

MAX_PDQPRIORITY 配置参数限制查询可以使用的并行数据库查询 (PDQ) 资源的百分比。使用 MAX_PDQPRIORITY 来限制大型 CPU 密集型查询对事务吞吐量的影响。

要限制大型 CPU 密集型查询对事务吞吐量的影响

将 MAX_PDQPRIORITY 配置参数的值设置为一个整数，该整数表示查询可请求以下 PDQ 资源的百分比：

- 内存
- CPU VP
- 磁盘 I/O
- 扫描线程

> 当查询请求一定百分比的 PDQ 资源时，数据库服务器分配的量将是所请求量的 MAX_PDQPRIORITY 百分比，如以下公式所示：
>
> ```
> Resources allocated = PDQPRIORITY/100 * MAX_PDQPRIORITY/100
> ```
>
> 例如，如果客户端使用 SET PDQPRIORITY 80 语句请求 80% 的 PDQ 资源，但是 MAX_PDQPRIORITY 设置为 50，那么数据库服务器将只分配 40% 的资源（请求的 50%）给客户端。
>
> 对于决策支持和联机事务处理 (OLTP)，设置 MAX_PDQPRIORITY 使数据库服务器管理员能够控制个别决策支持查询对并发 OLTP 性能的影响。想要将更多的资源分配给 OLTP 处理时，请减小 MAX_PDQPRIORITY 的值。想要将更多的资源分配给决策支持处理时，请增大 MAX_PDQPRIORITY 的值。

有关如何控制 PDQ 资源使用情况的更多信息，请参阅为并行数据库查询分配资源 在第255页。

相关链接

《*SinoDB* 管理员参考》: *MAX_PDQPRIORITY* 配置参数

限制 CPU 密集型查询的性能影响

DS_MAX_QUERIES 配置参数指定可以在任意时刻运行的决策支持查询的最大数。由于低 PDQ 优先级的查询按比例使用的资源较少，因此可以同时运行此类查询的数量更多。可以使用 DS_MAX_QUERIES 配置参数来限制 CPU 密集型查询的性能影响。

DS_MAX_QUERIES 配置参数仅控制具有非零的 PDQ 优先级的查询。

数据库服务器将 DS_MAX_QUERIES 的值和 DS_TOTAL_MEMORY 的值一起使用来计算分配给查询的内存量。有关数据库服务器如何将内存分配给查询的更多信息，请参阅 *DS_TOTAL_MEMORY* 配置参数和内存利用率 在第63页。

相关链接

《*SinoDB* 管理员参考》: *DS_MAX_QUERIES* 配置参数
DS_TOTAL_MEMORY 配置参数和内存利用率 在第63页

限制可并行运行的 PDQ 扫描线程的数量

DS_MAX_SCANS 配置参数限制可同时运行的 PDQ 扫描线程的数量。该配置参数防止数据库服务器被多个决策支持查询的扫描线程淹没。

要计算分配给查询的扫描线程数，请使用以下公式：

```
scan_threads = min (nfrags, (DS_MAX_SCANS * pdqpriority / 100
   * MAX_PDQPRIORITY / 100) )
```

nfrags

具有最多分段数的表中的分段数。

pdqpriority

由 PDQPRIORITY 环境变量或 SQL 语句 SET PDQPRIORITY 设置的 PDQ 优先级值。

减少扫描线程数可以缩短大型查询在就绪的队列中等待的时间，尤其是当同时提交许多大型查询的时候。但是，如果扫描线程数小于 *nfrags*，那么查询一旦开始，就会花费较长时间。

例如，如果查询需要扫描一个表中的 20 个分段，但是 *scan_threads* 公式使该查询在仅有 10 个扫描线程时就开始，每个扫描线程按顺序扫描两个分段。查询执行需要的时间将是使用 20 个扫描线程所需要时间的两倍。

相关链接

《*SinoDB* 管理员参考》: *DS_MAX_SCANS* 配置参数

配置轮询线程

NETTYPE 配置参数为您数据库服务器实例支持的每个连接类型配置轮询线程。如果数据库服务器实例支持多个接口或协议上的连接，那么必须对每个连接类型指定单独的 NETTYPE 配置参数。

通常，每个与 dbservername 关联的连接类型包含单独的 NETTYPE 参数。在 DBSERVERNAME 和 DBSERVERALIASES 配置参数中列出 dbservername。将连接类型与 sqlhosts 信息中的 dbservername 关联起来。 有关连接类型和 sqlhosts 信息的详细信息，请参阅《*SinoDB*® 管理员指南》中的连接配置。

相关链接

UNIX 信号量参数 在第38页
《*SinoDB* 管理员参考》: *NETTYPE* 配置参数

指定连接协议

第一个 NETTYPE 条目（其指定给定连接类型的协议）适用于所有与该类型关联的 dbservername。该连接类型的后续 NETTYPE 条目将被忽略。

仅用于内对外通信的连接类型才需要 NETTYPE 条目，即使这些连接类型未在 sqlhosts 信息中列出。

仅适用于 UNIX™

以下协议适用于 UNIX™ 平台：

- IPCSHM
- TLITCP
- IPCSTR
- SOCTCP
- TLIIMC
- SOCIMC
- SQLMUX
- SOCSSL

仅适用于 Windows™

以下协议适用于 Windows™ 平台：

- SOCTCP
- IPCNMP
- SQLMUX
- SOCSSL

相关链接

《SinoDB 管理员参考》：NETTYPE 配置参数

为轮询线程指定虚拟处理器类

NETTYPE 条目动态配置或添加的每个轮询线程均在单独的 VP 上运行。轮询线程可以在以下两类的其中一类 VP 上运行：NET（网络） 和 CPU。网络 VP 类包含 SOC、STR、SHM 和 TLI。为获取最佳性能，请使用 NETTYPE 条目为 CPU VP 类仅分配一个轮询线程。然后通过在 NETTYPE 配置参数值中指定 NET 将其他所有轮询线程分配给网络 VP 类。

相关链接

《SinoDB 管理员参考》：NETTYPE 配置参数

指定连接数和轮询线程数

每个轮询线程的最佳连接数对于单处理器计算机大约为 300，而对于多处理器计算机大约为 350，但这会根据平台和数据库服务器工作负载的不同而变化。

一个轮询线程可以支持 1024 或更多的连接。如果启用了 FASTPOLL 配置参数，那么可能可以配置的轮询线程更少，但您应该测试性能以确定适合您环境的最佳配置。

每个 NETTYPE 条目都会配置特定连接类型的轮询线程数、每个轮询线程的连接数，以及运行这些轮询线程的虚拟处理器类。如果每个线程的连接数超过 350 且当前连接类型的轮询线程数少于 CPU VP 数，那么您可以通过指定 CPU VP 类、添加轮询线程（其数量不能超过 CPU VP 数）以及重新设置每个线程的连接数来提高性能。每个线程的缺省连接数为 50。

> 重要： 每个 ipcshm 连接都需要信号量。有些操作系统需要配置计算机上运行的所有软件包可请求的信号量的最大数。要获取最佳性能，在为共享内存通信分配信号量时，将实际 ipcshm 连接数翻倍。请参阅 *UNIX 信号量参数* 在第38页。

如果您的计算机为单处理器且只为一个连接类型配置了数据库服务器实例，那么可以省略 NETTYPE 参数。数据库服务器使用 sqlhosts 信息中提供的信息来建立客户端/服务器连接。

如果您的计算机为单处理器且为多个连接类型配置了数据库服务器实例，那么为每个连接类型包含单独的 NETTYPE 条目。如果任意一个类型的连接数明显超过了 300，那么将分配两个或更多的轮询线程（最多为 CPU VP 数）并指定 NET VP 类，如以下示例所示：

```
NETTYPE ipcshm,1,50,CPU
NETTYPE tlitcp,2,200,NET # supports 400 connections
```

对于 ipcshm，轮询线程的数量对应于内存段的数量。例如，如果 NETTYPE 设置为 3,100 并且您只想要一个轮询线程，那么请将该轮询线程设置为 1,300。

如果您的计算机为多处理器，而且只为一个连接类型配置了数据库服务器实例，并且连接数没有超过 350，那么可以使用 NETTYPE 在 CPU 或 NET VP 类上指定单个轮询线程。如果连接数超过 350，那么将 VP 类设置为 NET，增加轮询线程数并重新计算 *conn_per_thread*。

> 重要： 您应该仔细区分网络连接的轮询线程和共享内存连接的轮询线程，每个 CPU 虚拟处理器都应该运行一个共享内存连接的轮询线程。配置 TCP 连接以在网络虚拟处理器中运行，并且配置维持响应所需的最少 TCP 连接数。配置共享内存连接以在每个 CPU 虚拟处理器中运行。

相关链接

《SinoDB 管理员参考》：NETTYPE 配置参数

《SinoDB 管理员参考》：VPCLASS 配置参数

提高连接性能和可伸缩性 在第48页

提高连接性能和可伸缩性

您可以在 NUMFDSERVERS 和 NS_CACHE 配置参数中指定信息并使用多个侦听线程来提高连接性能和可伸缩性。

SinoDB® SQL 会话可在 CPU VP 之间迁移。通过使用 NUMFDSERVERS 配置参数来指定在 VP 之间分布 TCP/IP 连接时使用的轮询线程数，从而可以提高 UNIX™ 上网络连接的性能和可伸缩性。如果数据库服务器具有高速率的新建连接和断开连接请求，或者您发现网络共享文件 (NSF) 锁之间存在大量争用，那么指定 NUMFDSERVERS 信息将非常有用。

您还应该查看 NETTYPE 配置参数中的信息并在必要时进行更改，该配置参数定义了特定连接类型的轮询线程数、每个轮询线程的连接数以及运行这些轮询线程的虚拟处理器类。指定 NETTYPE 配置参数信息，如下所示：

```
NETTYPE connection_type,poll_threads,conn_per_thread,vp_class
```

在 UNIX™ 上，如果 vp_class 为 NET，那么 poll_threads 可以是大于或等于 1 的值。如果 vp_class 为 CPU，那么 poll_threads 数可以是 1 到 CPU VP 数。在 Windows™ 上，poll_threads 可以是大于或等于 1 的值。

例如，假设您在 NETTYPE 配置参数中指定 8 个轮询线程，如下所示：

```
NETTYPE soctcp,8,300,NET
```

您也可以在 NUMFDSERVERS 配置参数中指定 8，以允许服务器使用全部 8 个轮询线程来处理在 VP 之间迁移的网络连接。

您可以使用 NS_CACHE 配置参数来定义主机名/IP 地址高速缓存、服务高速缓存、用户高速缓存和组高速缓存中各个条目的最大保留时间。服务器从高速缓存获取信息的速度比在查询操作系统时获取更快。

可以通过使用多个侦听线程来提高连接请求的服务。在为 onimcsoc 或 onsoctcp 协议指定 DBSERVERNAME 和 DBSERVERALIASES 配置参数信息时，您可以为 sqlhosts 信息中的数据库服务器别名指定多个侦听线程数。该数量的缺省值为 1。

DBSERVERNAME 和 DBSERVERALIASES 配置参数定义具有 sqlhosts 信息中相应条目的数据库服务器名称 (dbservername)。sqlhosts 信息中的每个 dbservername 参数具有用于指定接口/协议组合的 nettype 条目。数据库服务器为每个唯一 nettype 条目运行一个或多个轮询线程。

您可以使用 onstat -g ath 命令来显示有关所有线程的信息。

相关链接

《SinoDB 管理员参考》：NETTYPE 配置参数

《SinoDB 管理员参考》：NUMFDSERVERS 配置参数

《SinoDB 管理员参考》：NS_CACHE 配置参数

《SinoDB 管理员参考》：DBSERVERNAME 配置参数

《*SinoDB* 管理员参考》: *DBSERVERALIASES* 配置参数
《*SinoDB* 管理员指南》: 多个侦听线程
《*SinoDB* 管理员指南》: 在 *NS_CACHE* 配置参数中设置的名称服务最大保留时间
指定连接数和轮询线程数 在第47页
使用 *onstat -g ath* 输出监视线程 在第302页

启用快速轮询

如果操作系统平台支持快速轮询，那么使用 FASTPOLL 配置参数可启用或禁用网络的快速轮询。如果具有大量的连接，那么快速轮询将会有所帮助。

例如，如果数据库服务器具有 300 多个并发连接，那么可以启用 FASTPOLL 配置参数以获取更好的性能。

相关链接

《*SinoDB* 管理员参考》: *FASTPOLL* 配置参数

网络缓冲池

TCP/IP 连接的缓冲区大小会影响内存和 CPU 利用率。调整这些缓冲区能够容纳典型请求，借消除将请求拆分为多个消息的需求来提升 CPU 利用率。

但是，必须谨慎使用此功能；数据库服务器为活动连接动态分配指定大小的缓冲区。除非小心调整缓冲区大小，否则可能会占用大量内存。有关如何调整网络缓冲区大小的详细信息，请参阅网络缓冲区大小 在第50页。

数据库服务器为客户端的请求消息从全局内存池动态分配网络缓冲区。在数据库服务器处理客户端请求之后，缓冲区将返回至公共网络缓冲池，该缓冲池在使用 SOCTCP、IPCSTR 或 TLITCP 网络连接的会话之间可以共享。

此公共网络缓冲池具有以下优点：

- 防止从全局内存池中频繁分配和释放
- 使用较少的 CPU 资源在公共网络缓冲池中为每个网络传输分配和释放网络缓冲区
- 减少对共享内存的分配和释放的争用

在峰值活动期间，可用网络缓冲池会增加。为了防止大量未使用内存在网络活动不再频繁时仍然保留在这些网络缓冲池中，数据库服务器将在可用缓冲区数达到特定阈值时释放可用的缓冲区。

数据库服务器可提供下列特性，以进一步减少对可用网络缓冲区分配和释放的争用：

- 每个会话都有专用的可用网络缓冲池，以防止公共网络缓冲池或共享内存的全局内存池中对网络缓冲区的频繁分配和释放
- 可以指定大小超过 4 KB 的缓冲区，以接收客户端的网络信息包或消息

作为系统管理员，可以通过下列方法控制可用缓冲区阈值和每个缓冲区的大小：

- NETTYPE 配置参数
- IFX_NETBUF_PVTPOOL_SIZE 环境变量
- sqlhosts 信息中的 IFX_NETBUF_SIZE 环境变量和 b（客户端缓冲区大小）选项

网络缓冲区

数据库服务器会设置一个可用网络缓冲区的阈值，以防止网络缓冲池共享内存的频繁分配和释放。此阈值使数据库服务器能够将可用网络缓冲区数与您在 NETTYPE 配置参数中指定的连接数关联起来。

数据库服务器为客户端的请求消息动态分配网络缓冲区。数据库服务器处理完客户端请求后，将返回缓冲区到网络可用缓冲池。

如果可用缓冲区数大于阈值，数据库服务器将把分配给缓冲区的内存中超过阈值的部分返回全局池。

数据库服务器使用以下公式计算网络缓冲池中可用缓冲区的阈值：

```
free network buffers threshold =
   100 + (0.7 * number_connections)
```

number_connections 值是您在 NETTYPE 条目的第三个字段中为不同类型的网络连接（SOCTCP、IPCSTR 或 TLITCP）指定的连接总数。此公式不使用共享内存 (IPCSHM) 的 NETTYPE 条目。

如果没有在 NETTYPE 参数的第三个字段中指定值，那么数据库服务器将对每个与 SOCTCP、TLITCP 和 IPCSTR 协议相对应的 NETTYPE 条目使用 50 个连接的缺省值。

支持专用网络缓冲区

数据库服务器为使用 SOCTCP、IPCSTR 或 TLITCP 网络连接的每个会话提供对专用网络缓冲区的支持。

对于很多连接和会话持续保持活动的情况，这些专用网络缓冲区具有以下优点：

- 公共网络缓冲池的争用更少
- 使用较少的 CPU 资源在公共网络缓冲池中为每个网络传输分配和释放网络缓冲区

IFX_NETBUF_PVTPOOL_SIZE 环境变量指定每个会话的专用网缓冲池大小。缺省大小为一个缓冲区。

使用下表中的 onstat 实用程序命令来监视网络缓冲区的使用情况。

命令	输出字段	描述
onstat -g ntu	q-pvt	该会话的专用池中可用缓冲区的当前数和最大数
onstat -g ntm	q-exceeds	超出可用缓冲区阈值的次数

onstat -g ntu 命令显示 q-pvt 输出字段的以下格式：

```
current number / highest number
```

如果可用缓冲区数（q-pvt 字段中的值）一直为 0，那么可以执行以下操作之一：

- 用环境变量 IFX_NETBUF_PVTPOOL_SIZE 增加缓冲区数。
- 用环境变量 IFX_NETBUF_SIZE 增加每个缓冲区的大小。

q-exceeds 字段指示超出共享网络可用缓冲池的阈值的次数。当超出此阈值时，数据库服务器将未使用的网络缓冲区（超出此阈值部分）返回共享内存中的全局内存池。最佳情况下，该值应该为 0 或一个较小的数，这样服务器就不分配或释放全局内存池中的网络缓冲区。

相关链接

《SinoDB SQL 指南: 参考》: IFX_NETBUF_PVTPOOL_SIZE 环境变量 (UNIX)
《SinoDB SQL 指南: 参考》: IFX_NETBUF_SIZE 环境变量

网络缓冲区大小

IFX_NETBUF_SIZE 环境变量指定公共网络缓冲池和专用网缓冲池中每个网络缓冲区的大小。

缺省缓冲区大小为 4 KB。

IFX_NETBUF_SIZE 环境变量使数据库服务器能够在一次系统调用中接收超过 4 KB 的消息。较大的缓冲区大小可以减少接收每个信息包所需的开销。

如果知道客户端发送的信息包大于 4 KB，那么将增大 IFX_NETBUF_SIZE 值。客户端可能在下列任何情况下发送较大的信息包：

- 载入表
- 插入大于 4 KB 的行
- 发送简单大对象

sqlhosts 文件的 b 选项允许客户端发送和接收超过 4 KB 的消息。sqlhosts 选项的值通常应该与 IFX_NETBUF_SIZE 值匹配。

可以使用以下 onstat 命令查看网络缓冲区大小：

```
onstat -g afr global | grep net
```

输出中的 size 字段将显示网络缓冲区大小（以字节为单位）。

相关链接

《SinoDB 管理员指南》: 连接配置

《SinoDB SQL 指南: 参考》: IFX_NETBUF_SIZE 环境变量

虚拟处理器和 CPU 利用率

数据库服务器处于联机状态时，您可以启动和停止属于某些类的虚拟处理器 (VP)。

当数据库服务器处于联机状态时，您可以使用 onmode -p 为以下类启动附加的 VP：CPU、AIO、PIO、LIO、SHM、TLI 和 SOC。只有在数据库服务器处于联机状态时，您才可以删除 CPU 类的 VP。

您应该仔细区分网络连接的轮询线程和共享内存连接的轮询线程，每个 CPU 虚拟处理器都应该运行一个共享内存连接的轮询线程。TCP 连接应该仅处于网络虚拟处理器中，并且您应该只有维持响应所需的最少的 TCP 连接数。共享内存连接应该仅处于 CPU 虚拟处理器中，并且应在每个 CPU 虚拟处理器中运行。

添加虚拟处理器

只要您添加一个网络 VP（SOC 或 TLI），便添加了一个轮询线程。每个轮询线程均运行在单独的 VP 中，它可以是 CPU VP 或相应网络类型的网络 VP。

因为添加更多的 VP 可以增加 CPU 资源上的负载，所以如果 NETTYPE 值指示可用的 CPU VP 可以处理轮询线程，那么数据库服务器将该轮询线程分配给该 CPU VP。如果为所有的 CPU VP 均分配了轮询线程，那么数据库将添加另一个网络 VP 来处理轮询线程。

监视虚拟处理器

监视虚拟处理器以确定为数据库服务器配置的虚拟处理器数对于当前活动程度是否最佳。

要监视虚拟处理器：

- 使用命令行实用程序，例如 onstat-g ioq 来查看信息。请参阅使用一些 *onstat-g* 命令监视虚拟处理器 在第51页。
- 当服务器检测到 AIO 虚拟处理器无法满足 I/O 工作负载时，AUTO_AIOVPS 配置参数可使数据库服务器自动增加 AIO 虚拟处理器和 page-cleaner 线程的数量。
- 查询 SMI 表。请参阅使用 *SMI* 表监视虚拟处理器 在第53页。

使用一些 onstat-g 命令监视虚拟处理器
可以使用 onstat-g glo、onstat-g rea 和 onstat-g ioq 命令来监视虚拟处理器。
使用 onstat -g glo 命令监视虚拟处理器
使用 onstat-g glo 命令可以显示有关正在运行的每个虚拟处理器的信息，也可以显示每个虚拟处理器类的累计统计信息。

onstat -g glo 命令提供以下类型的信息：

- 正在运行的会话线程数
- 线程切换、让出或需要自旋多次才能获得锁存器或资源的频率
- 正在运行的虚拟处理器类以及每个类运行所花费的时间
- 针对每个虚拟处理器类正在运行的虚拟处理器数
- 正在运行的虚拟处理器以及每个虚拟处理器运行所花费的时间
- 每个虚拟处理器的效率

使用 onstat -g rea 命令可以确定是否需要增加虚拟处理器的数量。

相关链接

《SinoDB 管理员参考》: onstat -g glo 命令: 显示全局多线程信息

使用 onstat-g rea 命令监视虚拟处理器 在第52页

使用 onstat-g rea 命令监视虚拟处理器
使用 onstat-g rea 命令监视就绪队列中的线程数。

onstat-g rea 显示以下信息：

- 输出中的 status 字段在线程位于就绪的队列中时显示 ready 值。
- vp-class 输出字段显示执行线程的虚拟处理器类。

如果在就绪的队列中虚拟处理器类（例如，CPU 类）的线程数增加，那么可能需要将更多此类虚拟处理器添加到您的配置中。

```
Ready threads:
tid     tcb     rstcb   prty    status                vp-class   name

6       536a38  406464  4       ready                   3cpu    main_loop()
28      60cfe8  40a124  4       ready                   1cpu    onmode_mon
33      672a20  409dc4  2       ready                   3cpu    sqlexec
```

图 3: onstat-g rea 输出

相关链接

使用 onstat -g glo 命令监视虚拟处理器 在第51页

《SinoDB 管理员参考》: onstat -g rea 命令: 显示准备就绪的线程

使用 onstat-g ioq 命令监视虚拟处理器
使用 onstat-g ioq 命令可确定您是否需要分配其他的 AIO 虚拟处理器。

onstat-g ioq 命令在 len 列下显示 I/O 队列的长度，如下图所示。也可以在 maxlen 列中查看最大队列长度（自从数据库服务器启动后）。如果 I/O 队列的长度不断增加，那么 I/O 请求的累积速度将超过 AIO 虚拟处理器处理请求的速度。如果 I/O 队列的长度继续显示 I/O 请求不断累积，那么考虑添加 AIO 虚拟处理器。

```
onstat -g ioq

AIO I/O queues:
q name/id    len maxlen totalops  dskread dskwrite  dskcopy
  adt   0      0      0        0        0        0        0
  msc   0      0      1       12        0        0        0
  aio   0      0      4       89       68        0        0
  pio   0      0      1        1        0        1        0
  lio   0      0      1       17        0       17        0
  kio   0      0      0        0        0        0        0
  gfd   3      0      3      254      242       12        0
  gfd   4      0     17      614      261      353        0


onstat -d
Dbspaces
address  number   flags    fchunk   nchunks  flags    owner    name
a1de1d8  1        1        1        1        N        informix rootdbs
a1df550  2        1        2        1        N        informix space1
 2 active, 32,678 maximum
Chunks
address  chk/dbs offset   size     free     bpages   flags pathname
a1de320  1   1   0        75000    66447             PO-   /ix/root_chunk
a1df698  2   2   0        500      447               PO-   /ix//chunk1
```

```
  2 active, 32,678 maximum
```

图 4: onstat-g ioq 和 onstat -d 输出

AIO 虚拟处理器维护的每个块在 onstat-g ioq 输出中均具有一行（由 q name 列中的值 gfd 标识）。可以将 onstat -g ioq 中的行与实际块相关联，因为这些块与 onstat -d 输出处于同样的顺序。例如，在 onstat-g ioq 输出中，有两个 gfd 队列。第一个 gfd 队列保存 root_chunk 的请求，因为它与 onstat -d 输出中显示的第一个块相对应。同样，第二个 gfd 队列保存 chunk1 的请求，因为它与 onstat -d 输出内的第二个块相应。

如果数据库服务器既有裸设备又有熟文件（即已缓冲的文件），那么 gfd 队列仅与 onstat -d 输出中的熟文件相对应。

相关链接

《SinoDB 管理员参考》: onstat -g ioq 命令: 显示 I/O 队列信息

使用 SMI 表监视虚拟处理器
您可以从系统监视接口 (SMI) 表获取要用于监视虚拟处理器的信息。

要查询 SMI 表，您必须连接到 sysmaster 数据库。查询 sysvpprof SMI 表可获取有关当前正在运行的虚拟处理器的信息。此表包含以下各列。

列	描述
vpid	虚拟处理器的标识号
class	虚拟处理器的类
usercpu	用户 CPU 花费的秒数
syscpu	系统 CPU 花费的秒数

专用内存高速缓存

每个 CPU 虚拟处理器 (VP) 或租户 VP 都可以有专用内存高速缓存来缩短对内存块的访问时间。

在数据库服务器中线程请求的所有内存分配均由内存池执行。当内存池的内存块不足以满足内存分配请求时，会从全局内存池中分配块。因为所有线程都使用相同的全局内存池，所以会发生争用。专用内存高速缓存允许每个虚拟处理器保留其自己的一组内存块，该组内存块可用于绕过全局内存池。专用内存高速缓存的初始分配由全局内存池执行。当释放块时，它们将会释放到特定虚拟进程上的专用内存高速缓存中。当请求内存分配时，线程首先会检查专用内存高速缓存中的块是否可以满足分配。否则，线程将向全局内存池请求分配内存。

要确定专用内存高速缓存是否可以提高数据库服务器的性能，请执行 onstat -g spi 命令并查找 sh_lock 互斥锁。如果 onstat -g spi 命令输出显示 sh_lock 互斥锁的争用，请尝试创建专用内存高速缓存。

您可以通过指定所有专用内存高速缓存的初始组合大小来设置 VP_MEMORY_CACHE_KB 配置参数，从而启用专用内存高速缓存。缺省情况下，专用内存高速缓存的总大小受限于 VP_MEMORY_CACHE_KB 配置参数的大小值。您可以将模式设置为 DYNAMIC，以基于相关 VP 的工作负载自动增加或减少每个专用内存高速缓存的大小。在动态模式下，专用内存高速缓存的总大小可能会超过 VP_MEMORY_CACHE_KB 配置参数的值，但不能超过 SHMTOTAL 配置参数的值。

您可以通过执行 onstat -g vpcache 命令来查看有关 VP 专用内存高速缓存的统计信息。通过执行 onstat -g mem 命令可以查看有关内存池的统计信息。

注意: 如果您有多个 VP，那么专用内存高速缓存可以增加数据库服务器使用的内存量。

相关链接

《SinoDB 管理员参考》: VP_MEMORY_CACHE_KB 配置参数

《SinoDB 管理员参考》: onstat -g vpcache 命令: 显示 CPU 虚拟处理器和租户虚拟处理器专用内存高速缓存的统计信息

《SinoDB 管理员参考》: onstat -g mem 命令: 显示池内存统计信息

《*SinoDB* 管理员参考》: *onstat -g spi* 命令: 显示使用长自旋的旋转锁

连接和 CPU 利用率

一些应用程序具有大量的客户端/服务器连接。打开和关闭连接可能占用大量系统 CPU 时间。

以下主题描述您可以缩短打开和关闭连接所需的系统 CPU 时间的方法。

多路复用连接和 CPU 利用率

许多传统的非线程 SQL 客户端应用程序使用多个数据库连接执行单个用户的任务。每个数据库连接都建立到该数据库服务器的单独网络连接。多路复用连接设施在数据库服务器中提供一个网络连接功能以处理客户端应用程序的多个数据库连接。

多路复用连接使数据库服务器可以创建多个数据库连接，而不额外消耗其他网络连接所需的计算机资源。

当非线程客户端使用多路复用连接时，数据库服务器仍然创建与非多路复用连接相同数量的用户会话和用户线程。但是，网络连接数在使用多路复用连接时减少了。反之，数据库服务器使用一个多路复用侦听线程，以允许多个数据库连接共享同一个网络连接。

要提高非线程客户端的响应时间，可以使用多路复用连接执行 SQL 查询。性能提高的程度取决于以下因素：

- 网络连接总数的降低和由此造成的系统 CPU 时间的减少

  通常，处理网络连接的系统调用会占用大量系统 CPU 时间。因此，系统 CPU 时间降低的最大幅度与网络连接总数的减少成正比。
- 系统 CPU 时间的降低幅度与用户 CPU 时间的比率

  如果查询简单，并且使用的用户 CPU 时间很少，那么当使用多路复用连接时，响应时间可大幅度减少。但是如果查询复杂，并且使用大量的用户 CPU 时间，性能可能不会改进。

  要了解每个虚拟处理器的系统 CPU 时间量和用户 CPU 时间量，请使用 onstat -g glo 选项。

要对非线程客户端应用程序使用多路复用连接，必须在启动数据库服务器之前执行以下步骤：

1. 使用 DBSERVERALIASES 配置参数定义别名。例如，指定：

   ```
   DBSERVERALIASES ids_mux
   ```
2. 使用 sqlmux 将别名的 SQLHOSTS 条目添加为 nettype 条目，该条目在 SQLHOSTS 文件中为第 2 列。例如，指定：

   ```
   ids_mux  onsqlmux  ......
   ```

   该条目中的其他字段 hostname 与 servicename 必须存在，但不能忽略。
3. 在 sqlhosts 文件或注册表中指定客户端用于数据库服务器连接的 m=1，从而启用选定连接类型的多路复用。
4. 在 Windows™ 平台上，也必须设置 IFX_SESSION_MUX 环境变量。

   警告: 在 Windows™ 上，多线程应用程序不得使用多路复用连接功能。如果多线程应用程序在 sqlhosts 注册表条目中启用了多路复用选项，而且也定义了 IFX_SESSION_MUX 环境变量，那么可能产生灾难性后果，可能造成系统崩溃和数据损坏。

相关链接

《*SinoDB ESQL/C* 程序员指南》: 多路复用连接

《*SinoDB* 管理员指南》: 支持多路复用连接

第

4

章

配置对内存利用率的影响

操作系统和 SinoDB® 配置参数的组合可能会影响内存利用率。

您可以更改直接影响内存利用率的 SinoDB® 配置参数的设置，并且可以调整不同类型工作负载的设置。

通过设置操作系统配置参数为数据库服务器分配共享内存时，应考虑主机上可用的物理内存量。通常，如果增加数据库服务器的共享内存空间，就可以改善数据库服务器的性能。您必须在专用于数据库服务器的共享内存量与 VP 和其他进程的内存需求之间取得平衡。

相关链接

内存分配管理器 在第254页

共享内存

必须在操作系统中为数据库服务器配置足够的共享内存资源。反过来，共享内存不足会影响性能。

数据库服务器线程和进程需要共享内存，以通过共享对内存段的访问权来共享数据。

SinoDB® 使用的共享内存可分为以下部分，其中每个包含一个或多个共享内存段：

- 常驻部分
- 虚拟部分
- 消息部分
- 缓冲池部分

常驻部分和消息部分是静态的；必须为它们分配足够的内存，数据库服务器才能进入联机方式。（通常，必须重新启动操作系统以重新配置共享内存。）数据库服务器共享内存的虚拟部分会动态增加，但是仍必须在操作系统共享内存分配中为该部分保留足够的初始量。

所需空间量是数据库服务器共享内存所有三个部分需要的总量。使用 SHMTOTAL 配置参数指定共享内存的总量。

LOCKS 配置参数指定锁表的初始大小。如果会话分配的锁数超过了 LOCKS 的值，那么数据库服务器将动态增加该锁表的大小。如果期望锁表动态地增长，那么将 SHMTOTAL 设置为 0。当 SHMTOTAL 为 0 时，对分配的内存总量（包括共享内存）没有任何限制。

相关链接

《SinoDB 管理员参考》: LOCKS 配置参数

《SinoDB 管理员参考》: SHMTOTAL 配置参数

共享内存的常驻部分

共享内存的常驻部分包括记录数据库服务器状态的共享内存区域，包括锁、日志文件，以及数据库空间、块和表空间的位置。

用于 LOCKS、LOGBUFF 和 PHYSBUFF 配置参数的设置有助于确定常驻部分的大小。

除了这些影响常驻部分大小的配置参数以外，RESIDENT 配置参数还可影响内存利用率。计算机支持强制驻留而 RESIDENT 配置参数设置为锁定常驻或锁定常驻和虚拟部分的值时，常驻部分绝不会换页。

数据库服务器的机器说明文件指示您的操作系统是否支持强制驻留。

在支持大页的 AIX®、Solaris 和 Linux™ 系统上，IFX_LARGE_PAGES 环境变量可将大页用于锁定在物理内存中的非消息共享内存段。如果操作系统命令已配置大页而 RESIDENT 配置参数指定共享内存的某些或全部常驻部分和虚拟部分锁定在物理内存中，那么 SinoDB® 会将大页用于相应共享内存段（前提是有足够的大页可用）。使用大页可以在大内存配置方面具有显著的性能优势。

相关链接

影响内存利用率的配置参数 在第60页

《*SinoDB SQL 指南: 参考*》: *IFX_LARGE_PAGES 环境变量*

共享内存的虚拟部分

SinoDB® 使用共享内存的虚拟部分来根据需要向每个数据库服务器子系统分配内存。

数据库服务器共享内存的虚拟部分包括以下组件：

- 大缓冲区（用于大型读取和写入 I/O 操作）
- 排序空间池
- 活动的线程控制块、堆栈和堆
- 用户会话数据
- SQL 语句、数据字典信息和用户定义例程的高速缓存
- 网络接口消息缓冲区和其他信息的全局池

onconfig 文件中的 SHMVIRTSIZE 配置参数提供虚拟部分的初始大小。随着虚拟部分中额外空间需求的上升，数据库服务器将按 SHMADD 配置参数指定的增量来添加共享内存。EXTSHMADD 配置参数会配置为用户定义例程和 DataBlade® 例程添加的虚拟扩展共享内存段的大小。对分配给数据库服务器的共享内存总量的限制由 SHMTOTAL 参数来指定。

虚拟部分的大小主要取决于正在运行的应用程序及查询的类型。根据应用程序的不同，对虚拟部分的初始估计范围可以从低至每个用户 100 KB 到高达每个用户 500 KB，如果打算使用数据分布，可另外再加上 4 MB。

计算机支持强制驻留而 RESIDENT 配置参数设置为锁定虚拟段的值时，锁定的虚拟段绝不会换页。

在支持大页的 AIX®、Solaris 和 Linux™ 系统上，IFX_LARGE_PAGES 环境变量可将大页用于锁定在物理内存中的非消息共享内存段。如果操作系统命令已配置大页而 RESIDENT 配置参数指定共享内存的某些或全部常驻部分和虚拟部分锁定在物理内存中，那么 SinoDB® 会将大页用于相应共享内存段（前提是有足够的大页可用）。使用大页可以在大内存配置方面具有显著的性能优势。

相关链接

影响内存利用率的配置参数 在第60页

《*SinoDB SQL 指南: 参考*》: *IFX_LARGE_PAGES 环境变量*

创建数据分布 在第275页

《*SinoDB 管理员参考*》: *EXTSHMADD 配置参数*

《*SinoDB 管理员参考*》: *SHMADD 配置参数*

《*SinoDB 管理员参考*》: *SHMTOTAL 配置参数*

《*SinoDB 管理员参考*》: *SHMVIRTSIZE 配置参数*

共享内存的消息部分

共享内存的消息部分包含共享内存通信接口使用的消息缓冲区。这些缓冲区所需的空间量取决于使用给定网络接口允许的用户连接数。

如果没有使用特定的接口，那么在操作系统中分配共享内存时就不需要为其分配空间。

共享内存的缓冲池部分

共享内存的缓冲池部分包含一个或多个缓冲池。数据库空间使用的每个页大小都有一个缓冲池。

当数据库服务器启动时，BUFFERPOOL 配置参数会指定缓冲池的大小。如果缓冲池是可扩展的，那么数据库服务器将增加共享内存的缓冲池部分中缓冲池的大小。

通过执行 onstat -g buf 命令以及在每个缓冲池的 Total Mem 字段中添加值，您可以确定共享内存的缓冲池部分的当前大小。例如，以下输出显示一个缓冲池的内存为 32 MB：

```
Fg Writes     LRU Writes    Avg. LRU Time Chunk Writes  Total Mem
0             0             nan           10883         32Mb
```

每个缓冲池的最大大小取决于共享内存的可用量和 BUFFERPOOL 配置参数的值。

相关链接

《*SinoDB* 管理员指南》*:* 共享内存的缓冲池部分

《*SinoDB* 管理员参考》*: BUFFERPOOL* 配置参数

《*SinoDB* 管理员参考》*: onstat -g buf* 命令*:* 显示缓冲池的概要文件信息

估算共享内存常驻部分的大小

可以使用公式在分配操作系统共享内存时估算共享内存的常驻部分大小（以 KB 为单位）。

计算的结果是估算值，通常略微超出用于共享内存常驻部分的实际内存。

计算以下估算值来确定 64 位服务器上共享内存的常驻部分。显示的大小可能会有所变化，且计算是近似的。

要估算共享内存常驻部分的大小，请执行以下操作

1. 计算下列公式中的值：

```
locks_value = LOCKS * 136
logbuff_value = LOGBUFF * 1024 * 3
physbuff_value = PHYSBUFF * 1024 * 2
```

2. 使用以下公式来计算常驻部分的估算大小（以 KB 为单位）：

```
rsegsize = 1.02 * (locks_value + logbuff_value
           + physbuff_value + 1,200,000) / 1024
```

估算共享内存虚拟部分的大小

可以使用公式来估算共享内存虚拟部分的初始大小。在 SHMVIRTSIZE 配置参数中指定初始大小。

估算共享内存虚拟部分的初始大小的公式如下：

```
shmvirtsize = 固定开销 + 共享结构 +
              (mncs * 专用结构) +
              其他缓冲区
```

要使用上述公式估算 SHMVIRTSIZE 值，请执行以下操作：

1. 估算固定开销部分的值，如下所示：

```
固定开销 = 全局池 + 启动后的线程池
```

a) 执行 onstat -g mem 命令获取分配给会话的池大小。
b) 从 totalsize 值中减去 freesize 字段中的值即可得到分配给每个会话的字节数。
c) 估算启动后的线程池变量的值。此变量部分取决于虚拟处理器数。

2. 使用以下公式来估算共享结构的值：

```
共享结构 = AIO 向量 + 排序内存 +
                      数据库空间备份缓冲区 +
                      数据字典高速缓存大小 +
                      用户定义例程高速缓存大小 +
                      histogram 池 +
                      STMT_CACHE_SIZE（SQL 语句高速缓存）+
                      其他池（请参阅 onstat 显示。）
```

3. 估算公式的下一部分，如下所示：
   a) 用以下公式估算 mncs 的值（为并行会话的最大数量）：

   ```
   mncs = 轮询线程数 * 每个轮询线程连接数
   ```

   轮询线程数的值是在 NETTYPE 配置参数的第 2 个字段中指定的值。

   每个轮询线程连接数的值是在 NETTYPE 配置参数的第 3 个字段中指定的值。

   在高峰处理期间执行 onstat -u 命令时，也可能会得到并发会话最大数的估算值。onstat -u 输出的最后一行包含最大的并发用户线程数。

   b) 估算专用结构 的值，如下所示：

   ```
   专用结构 = 堆栈 + 堆 +
                    会话控制块结构
   ```

   堆栈

   通常为 32 KB，但取决于用户定义例程中的递归。可以用 onstat -g sts 选项来获取每个线程的堆栈大小。

   堆

   大约为 15 KB。使用 onstat -g stm 选项时，您可以获取 SQL 语句的堆大小。

   会话控制块结构

   每个会话使用的内存量。onstat -g ses 选项在为每个会话标识符列出的 total memory 列中显示内存量（以字节为单位）。

   c) 将步骤 3a 和 3b 的结果相乘得到公式的以下部分：

   ```
   mncs * 专用结构
   ```

4. 估算其他缓冲区的值，以说明分配给某些功能（例如，智能大对象的轻量级 I/O 操作）的专用缓冲区。每个用户大约 180 KB。
5. 将步骤 1 到 4 的结果相加可获取 SHMVIRTSIZE 配置参数的估算值。

   提示： 当数据库服务器正以稳定的工作负载运行时，可以使用 onstat -g seg 来获取共享内存虚拟部分的实际大小的精确值。然后，您可以使用此命令报告的共享内存值来重新配置 SHMVIRTSIZE。

要为稍后添加到虚拟共享内存中的段指定大小，请设置 SHMADD 配置参数。使用 EXTSHMADD 配置参数可以指定为用户定义例程和 DataBlade® 例程添加的虚拟扩展段的大小。

下表列出了其他说明如何估算内存中共享结构大小的主题。

表 3: 共享内存结构的信息

共享内存结构	更多信息
排序内存	估算排序所需的内存 在第165页
数据字典高速缓存	数据字典配置 在第71页
数据分布高速缓存（histogram 池）	数据分布配置 在第72页

共享内存结构	更多信息
用户定义例程 (UDR) 高速缓存	存储在 *UDR* 高速缓存中的 *SPL* 例程可执行格式 在第234页
SQL 语句高速缓存	启用 *SQL* 语句高速缓存 在第296页 监视和调整 *SQL* 语句高速缓存 在第73页
其他池	要查看分配给不同池的内存数量，可使用 onstat -g mem 命令。

相关链接

《*SinoDB* 管理员参考》: *SHMVIRTSIZE* 配置参数

《*SinoDB* 管理员参考》: *NETTYPE* 配置参数

会话内存 在第82页

《*SinoDB* 管理员参考》: *onstat -g mem* 命令: 显示池内存统计信息

估算共享内存消息部分的大小

可以估算共享内存消息部分的大小（以 KB 为单位）。

使用以下公式来估算共享内存消息部分的大小：

```
msegsize = (10,531 * ipcshm_conn + 50,000)/1024
```

ipcshm_conn

可以使用共享内存接口建立的连接数，由 ipcshm 协议的 NETTYPE 参数确定。

相关链接

《*SinoDB* 管理员参考》: *NETTYPE* 配置参数

配置 UNIX™ 共享内存

在 UNIX™ 上，可以为数据库服务器配置共享内存段。

在 UNIX™ 上，执行以下步骤配置数据库服务器配置需要的共享内存段。有关如何设置与共享内存相关的参数的信息，请参阅您操作系统的配置指示信息。

要为数据库服务器配置共享内存段：

1. 如果操作系统对共享内存段大小没有限制，可以进行以下操作：
   a) 将最大段大小的操作系统配置参数（通常是 SHMMAX 或 SHMSIZE）设置为数据库服务器配置所需的总大小。该大小包括启动数据库服务器实例所需的内存量，以及为虚拟部分动态增长而分配的共享内存量。
   b) 对于数据库服务器的每个实例，至少将最大段数的操作系统配置参数（通常是 SHMMNI）设置为 1。
2. 如果操作系统具有段大小限制，那么将采取以下操作：
   a) 将最大段大小的操作系统配置参数（通常是 SHMMAX 或 SHMSIZE）设置为系统允许的最大值。
   b) 使用以下公式来计算您的数据库服务器实例的段数。如果有余数，那么向上舍入到最接近的整数。

   ```
   SHMMNI = total_shmem_size / SHMMAX
   ```

   total_shmem_size

   分配给数据库服务器使用的共享内存总量。
3. 将最大段数的操作系统配置参数（通常是 SHMMNI）设置为一个值，当该值乘以 SHMMAX 或 SHMSIZE 时，将产生数据库服务器的共享内存总量。如果计算机专用于一个数据库服务器实例，那么总量最多可以达到虚拟内存大小的 90%（物理内存加交换空间）。
4. 如果操作系统使用 SHMSEG 配置参数来指示进程可以附加的最大共享内存段数，那么为此参数设置的值应等于或大于分配给任意数据库服务器实例的最大段数。

有关在操作系统中配置共享内存的更多技巧，请参阅 UNIX™ 的机器说明文件或 Windows™ 的发行说明文件。

相关链接

SHMADD 和 EXTSHMADD 配置参数和内存利用率 在第67页

使用 onmode -F 释放共享内存

可以执行 onmode -F 命令来释放进程不可用或不再需要的共享内存段。

数据库服务器不会自动释放操作过程中添加的共享内存段。将内存分配给数据库服务器虚拟部分后，该内存仍无法供主机中运行的其他进程使用。当数据库服务器运行大的决策支持查询时，可能会获得大量共享内存。查询完成后，数据库服务器将不再需要这些共享内存。但是，即使不再需要，数据库服务器分配用于执行查询的共享内存仍将保留并分配给虚拟部分。

onmode -F 命令将查找并释放数据库服务器仍然保留但不使用的 8 KB 共享内存块。虽然该命令的运行时间很短暂（一两秒钟），但是 onmode -F 在运行时会极大地限制用户活动。该实用程序运行时，附带多个 CPU 和 CPU VP 的系统性能通常会稍有下降。

应该在空隙时间段使用操作系统调度工具（如 UNIX™ 上的 cron）执行 onmode -F。另外，在执行会显著增加数据库服务器共享内存大小的所有任务（例如，大型决策支持查询、索引构建、排序或备份操作）后，请考虑执行该实用程序。

相关链接

《SinoDB 管理员参考》: onmode -F: 释放未使用的内存段

影响内存利用率的配置参数

ONCONFIG 文件中的大量配置参数会影响内存利用率和性能。

以下配置参数显著影响内存利用率：

- BUFFERPOOL
- DS_NONPDQ_QUERY_MEM
- DS_TOTAL_MEMORY
- EXTSHMADD
- LOCKS
- LOGBUFF
- LOW_MEMORY_MGR
- LOW_MEMORY_RESERVE
- PHYSBUFF
- RESIDENT
- SHMADD
- SHMBASE
- SHMTOTAL
- SHMVIRTSIZE
- SHMVIRT_ALLOCSEG
- STACKSIZE
- 内存高速缓存参数（请参阅配置和监视内存高速缓存 在第69页）
- 网络缓冲区大小（请参阅网络缓冲池 在第49页）

SHMBASE 参数指示数据库服务器共享内存的初始地址。当按照机器说明文件或发行说明文件中的指示信息设置时，该参数对性能没有明显的影响。有关每个文件的路径名，请参阅本指南的简介。

DS_NONPDQ_QUERY_MEM 参数增加可用于非 PDQ 查询的内存量。如果 PDQ 优先级设置为 0，那么只能使用此参数。有关更多信息，请参阅为使用散列连接、聚合和其他内存密集型元素的查询分配更多内存 在第290页。

以下各节描述了与本节开头列出的一些配置参数相关的性能影响和注意事项。

相关链接

共享内存的常驻部分 在第55页
共享内存的虚拟部分 在第56页
《*SinoDB* 管理员参考》: *LOW_MEMORY_MGR* 配置参数
《*SinoDB* 管理员参考》: *LOW_MEMORY_RESERVE* 配置参数

设置缓冲池、逻辑日志缓冲区和物理日志缓冲区的大小

为 BUFFERPOOL、DS_TOTAL_MEMORY、LOGBUFF 和 PHYSBUFF 配置参数指定的值取决于正在使用的应用程序类型（OLTP 或 DSS）以及页大小。

表 *4: OLTP* 和 *DSS* 应用程序的准则 在第61页列出了这些参数的推荐设置或设置这些参数的准则。

有关估算共享内存常驻部分大小的信息，请参阅估算共享内存常驻部分的大小 在第57页。此计算包括计算缓冲池、逻辑日志缓冲区、物理日志缓冲区以及锁表的大小。

表 4: OLTP 和 DSS 应用程序的准则

配置参数	OLTP 应用程序	DSS 应用程序
BUFFERPOOL	缓冲区空间所需的物理内存百分比取决于系统上可用的内存量以及用于其他应用程序的内存量。	对于轻度扫描、查询和排序，设置为小缓冲区值并增加 DS_TOTAL_MEMORY 值。 对于诸如通过缓冲池读取数据的索引建立等操作，需配置大量的缓冲区。
DS_TOTAL_MEMORY	设置为 SHMTOTAL 值的 20% 到 50%（以 KB 为单位）。	设置为 SHMTOTAL 值的 50% 到 90%。
LOGBUFF	逻辑日志缓冲区大小的缺省值为 64 KB。 如果您决定使用一个较小的值，那么数据库服务器会生成一条消息，指示这样不能获得最佳性能。如果使用小于 64 KB 的逻辑日志缓冲区，那么会对性能产生影响，但不会对事务完整性产生影响。 如果将数据库或应用程序定义为使用缓冲的日志记录，那么增加 LOGBUFF 大小（超过 64 KB）会提高性能。	由于通常会对 DSS 应用程序关闭数据库或表的日志记录，因此可以将 LOGBUFF 设置为 32 KB。
PHYSBUFF	物理日志缓冲区大小的缺省值为 128 KB。 如果启用了 RTO_SERVER_RESTART 配置参数，请使用 PHYSBUFF 的缺省值 (512 KB)。 如果您决定使用一个小于缺省值的值，那么数据库服务器会生成一条消息，指示这样不能获得最佳性能。如果使用小于缺省大小的物理日志缓冲区，那么会对性能产生	由于大部分 DSS 应用程序实际上并不进行日志记录，因此可以将 PHYSBUFF 设置为 32 KB。

配置参数	OLTP 应用程序	DSS 应用程序
	影响，但不会对事务完整性产生影响。	

相关链接

《*SinoDB* 管理员参考》: *BUFFERPOOL* 配置参数

《*SinoDB* 管理员参考》: *DS_TOTAL_MEMORY* 配置参数

《*SinoDB* 管理员参考》: *LOGBUFF* 配置参数

《*SinoDB* 管理员参考》: *PHYSBUFF* 配置参数

《*SinoDB* 管理员参考》: *RTO_SERVER_RESTART* 配置参数

BUFFERPOOL 配置参数和内存利用率

BUFFERPOOL 配置参数会指定缓冲池的属性。在 BUFFERPOOL 配置参数字段中定义的信息会影响内存利用率。

如果有使用不同页大小的数据库空间，那么您可以有多个缓冲池。onconfig 配置文件中每个页大小都包含一个 BUFFERPOOL 行。例如，在页大小为 2 KB 的计算机上，onconfig 文件最多可以包含 9 行，包括缺省规范。当您创建具有不同页大小的数据库空间时，该页大小的缓冲池会自动创建（如果不存在的话）。页大小的 BUFFERPOOL 条目会添加到 onconfig 文件。BUFFERPOOL 配置参数字段值与缺省规范相同。

BUFFERPOOL 配置参数会控制数据库服务器可用的数据缓冲区数。这些缓冲区位于共享内存的缓冲池部分，并用于将数据库数据页高速缓存在内存中。

增加缓冲区的数量会增加所需数据页可能由于先前的请求而已经位于内存中的可能性。但是，分配过多的缓冲区可能会影响内存管理系统并导致过度的操作系统调页活动。要利用 64 位寻址机器上可用的大内存，可以增加缓冲池的大小。

缓冲池的大小对数据库 I/O 和事务吞吐量有显著影响。您可以通过使缓冲池可扩展来确保缓冲池具有足够的缓冲区。当缓冲池可扩展时，数据库服务器将根据需要扩展缓冲池以提高性能。

缓冲池的大小等于缓冲区的数量乘以页大小。缓冲区空间所需的物理内存百分比取决于系统上可用的内存大小以及用于其他应用程序的内存大小。对于具有大量可用物理内存（4 GB 或更大）的系统，缓冲区空间可能为物理内存的 90%。对于可用物理内存较小的系统，缓冲区空间可能在物理内存的 20% 到 25% 之间。

例如，如果系统的页大小为 2 KB，物理内存为 100 MB，那么可以将 buffers 字段中的值设为 10,000 到 12,500，这将分配 20 MB 到 25 MB 的内存。

在指定缓冲池大小后，计算所有其他共享内存参数。

> 注： 如果使用非缺省的页大小，那么可能需要增大物理日志的大小。如果频繁对非缺省的页进行更新，那么可能需要将物理日志大小增大 150% 到 200%。要调整物理日志，可能需要一些试验。可根据物理日志触发器检查点的填充频率来按需调整物理日志的大小。

您可以使用 onstat -g buf 来监视缓冲池统计信息，包括缓冲池的读取高速缓存率。该速率表示查询请求页时共享内存缓冲区中已存在的数据库页的百分比。（如果页尚不存在，则数据库服务器必须将其从磁盘复制到内存中。）如果数据库服务器在缓冲池中找到该页，那么将花费较少的时间在磁盘 I/O 上。 因此，您需要高读取缓存率以获得良好的性能。对于许多用户读取少量数据集的 OLTP 应用程序，目标是实现 95% 或更高的读取高速缓存率。如果缓冲池是可扩展的，那么可以指定数据库服务器扩展缓冲池的读取高速缓存命中率。

使用操作系统中的内存管理监视实用程序（例如 UNIX™ 上的 vmstat 或 sar）来记录页扫描和分页活动的级别。如果这些级别在数据库活动高峰期间突然上升或上升到不可接受的级别，请减小缓冲池的大小。

智能大对象和缓冲区

根据您的情况，您可以采取以下其中一种措施为使用智能大对象的应用程序实现更好的性能：

- 如果您的应用程序频繁访问大小为 2 KB 或 4 KB 的智能大对象，那么请使用缓冲池使其保存在内存中的时间更长。使用以下公式来增加 buffers 字段的值：

  ```
  Additional_buffers = numcur_open_lo *
                       (lo_userdata / pagesize)
  ```

  在这个公式中：

  - *numcur_open_lo* 是可以从 onstat -g smb fdd 命令获得的并发打开的智能大对象的数量。
  - *lo_userdata* 是要缓冲的智能大对象数据的字节数。
  - *pagesize* 是计算机的缺省页大小（以字节为单位）。

  通常，尝试拥有足够的缓冲区来为每个并发打开的智能大对象保存两个智能大对象页。附加页可用于预读。
- 在共享内存的虚拟部分中使用轻量级 I/O 缓冲区。

  只有当您在超过 8000 字节的操作中读取或写入智能大对象时才使用轻量级 I/O 缓冲区，并且很少访问。也就是说，如果读取或写入函数调用在单个函数调用中读取大量数据，请使用轻量级 I/O 缓冲区。

  在使用轻量级 I/O 缓冲区时，可以防止智能大对象大量涌入缓冲池，并为多个用户频繁访问的其他数据页留下更多可用的缓冲区。

相关链接

《*SinoDB* 管理员参考》: *BUFFERPOOL* 配置参数

《*SinoDB* 管理员指南》: 监视缓冲区

智能大对象的轻量级 *I/O* 在第100页

BUFFERPOOL 及其对页清除的影响 在第111页

DS_TOTAL_MEMORY 配置参数和内存利用率

DS_TOTAL_MEMORY 配置参数将对查询可以获取的共享内存量设置上限。可以使用该参数限制大型内存密集型查询对性能的影响。该参数设置得越高，大型查询可以使用的内存越多，而可用于处理其他查询和事务的内存就越少。

对于 OLTP 应用程序，将 DS_TOTAL_MEMORY 设置在 SHMTOTAL 值的 20% 到 50% 之间（以 KB 为单位）。对于涉及大型决策支持 (DSS) 查询的应用程序，将 DS_TOTAL_MEMORY 值增加到 SHMTOTAL 的 50% 到 80% 之间。如果为 DSS 查询专门使用数据库服务器实例，那么将此参数设置为 SHMTOTAL 的 90%。

量子单位是分配给查询的内存的最小增量。内存分配管理器 (MGM) 将内存按量子单位分配给查询。数据库服务器根据以下公式使用 DS_MAX_QUERIES 的值与 DS_TOTAL_MEMORY 的值来计算内存量：

```
quantum = DS_TOTAL_MEMORY / DS_MAX_QUERIES
```

数据库服务器在分配内存时可以动态调整量子大小。为允许同时进行各自带有较小量子的更多查询，建议您增加 DS_MAX_QUERIES 配置参数的值。

相关链接

内存分配管理器 在第254页

《*SinoDB* 管理员参考》: *DS_TOTAL_MEMORY* 配置参数

限制 *CPU* 密集型查询的性能影响 在第46页

确定 DS_TOTAL_MEMORY 的算法

如果没有设置 DS_TOTAL_MEMORY 配置参数或将此配置参数设置为不恰当的值，那么数据库服务器将派生 DS_TOTAL_MEMORY 的值。

只要数据库服务器更改指定给 DS_TOTAL_MEMORY 的值，它就会将以下消息发送至控制台：

```
DS_TOTAL_MEMORY recalculated and changed from old_value Kb
           to new_value Kb
```

变量 *old_value* 表示配置文件中指定给 DS_TOTAL_MEMORY 的值。变量 *new_value* 表示数据库服务器派生的值。

接收到上述消息时，可以使用该算法查明数据库服务器认为不恰当的值。然后可以根据调查结果采取纠正措施。

以下各节记录数据库服务器用来派生 DS_TOTAL_MEMORY 新值的算法。

派生决策支持内存的最小值
在数据库服务器为 DS_TOTAL_MEMORY 配置参数派生新值的算法的第一部分中，数据库服务器将建立决策支持内存的最小量。

为 DS_MAX_QUERIES 配置参数指定值时，数据库服务器将根据以下公式设置决策支持内存的最小量：

```
min_ds_total_memory = DS_MAX_QUERIES * 128 kilobytes
```

没有为 DS_MAX_QUERIES 配置参数指定值时，数据库服务器将改用以下基于 VPCLASS 配置参数中信息值的公式：

```
min_ds_total_memory = NUMBER_CPUVPS * 2 * 128 kilobytes
```

派生决策支持内存的工作值
在数据库服务器为 DS_TOTAL_MEMORY 配置参数派生新值的算法的第二部分中，数据库服务器将建立决策支持内存量的工作值。

数据库服务器在该算法的第三部分和最后部分验证此数量。

DS_TOTAL_MEMORY 配置参数已设置的情况
当 DS_TOTAL_MEMORY 配置参数已设置时，数据库服务器将检查是否设置 SHMTOTAL 配置参数，然后确定使用哪个公式来计算决策支持内存的量。

设置 SHMTOTAL 时，数据库服务器将使用以下公式来计算决策支持内存量：

```
IF DS_TOTAL_MEMORY <= SHMTOTAL - nondecision_support_memory THEN
   decision_support_memory = DS_TOTAL_MEMORY
ELSE
    decision_support_memory = SHMTOTAL -
                  nondecision_support_memory
```

此算法有效防止将 DS_TOTAL_MEMORY 设置为数据库服务器无法分配给决策支持内存的值。

未设置 SHMTOTAL 时，数据库服务器将设置决策支持内存，使之与 DS_TOTAL_MEMORY 中指定的值相等。

相关链接

《*SinoDB* 管理员参考》: *DS_TOTAL_MEMORY* 配置参数

DS_TOTAL_MEMORY 配置参数未设置的情况
当 DS_TOTAL_MEMORY 配置参数未设置时，数据库服务器将使用其他源计算决策支持内存的数量值。

设置 SHMTOTAL 时，数据库服务器将使用以下公式来计算决策支持内存量：

```
decision_support_memory = SHMTOTAL -
                  nondecision_support_memory
```

数据库服务器发现您没有设置 SHMTOTAL 后，将设置决策支持内存，如下所示：

```
decision_support_memory = min_ds_total_memory
```

有关变量 min_ds_total_memory 的描述，请参阅派生决策支持内存的最小值 在第64页。

检查决策支持内存的派生值
在数据库服务器为 DS_TOTAL_MEMORY 配置参数派生新值的算法的最后部分中，数据库服务器将验证共享内存量是否大于 *min_ds_total_memory* 且小于计算机可能的最大内存空间。

当数据库服务器发现决策支持内存的派生值小于 *min_ds_total_memory* 变量值时，它会设置决策支持内存，使其等于 *min_ds_total_memory* 的值。

当数据库服务器发现决策支持内存的导出值大于计算机最大可能的内存空间时，它会设置决策支持内存，使其等于最大可能的内存空间。

LOGBUFF 配置参数和内存利用率
LOGBUFF 配置参数确定为每个缓冲区（共 3 个）保留的共享内存量，这些缓冲区将保存逻辑日志记录直至这些记录被清空到磁盘上的逻辑日志文件为止。缓冲区的大小确定填充的频率并由此确定必须清空到磁盘上逻辑日志文件的频率。

如果记录智能大对象，那么增加逻辑日志缓冲区的大小可以防止频繁地清空到磁盘上的逻辑日志文件中。

相关链接

《SinoDB 管理员参考》: LOGBUFF 配置参数
影响关键数据的配置参数 在第90页

LOW_MEMORY_RESERVE 配置参数和内存利用率
LOW_MEMORY_RESERVE 配置参数会保留特定内存量（以 KB 为单位），以供数据库服务器在需要关键活动且该服务器的可用内存有限时使用。

如果通过将新 LOW_MEMORY_RESERVE 配置参数设置为指定的值（以 KB 为单位）来启用该参数，那么即使接收到内存不足错误，关键活动（例如，回滚活动）仍然可以完成。

相关链接

《SinoDB 管理员参考》: LOW_MEMORY_RESERVE 配置参数
《SinoDB 管理员参考》: onstat -g seg 命令: 显示共享内存段的统计信息

PHYSBUFF 配置参数和内存利用率
PHYSBUFF 配置参数确定为每个缓冲区（共 2 个）保留的共享内存量，这些缓冲区充当要修改的数据页的临时存储空间。缓冲区的大小确定填充的频率并由此确定必须清空到磁盘上物理日志的频率。

为系统页大小的平均增量 PHYSBUFF 选择一个值。

相关链接

《SinoDB 管理员参考》: PHYSBUFF 配置参数

LOCKS 配置参数和内存利用率

LOCKS 配置参数指定锁表的初始大小。锁表为会话使用的每个锁保存条目。在锁表中每个锁使用 120 个字节。配置共享内存时必须提供此数量的内存。

如果会话所需的锁数超过 LOCKS 配置参数中设置的值，那么数据库服务器会尝试使锁表的大小增加一倍。每次锁表溢出时（当所需锁数量大于锁表的当前大小时），数据库服务器就会增加锁表大小，可高达 99 次。每次数据库服务器增加锁表的大小时，该服务器都会尝试将锁表大小翻倍。但是，服务器会将每次的实际增加值限制为不超过 *表 5: 32 位和 64 位平台上的最大锁数* 在第66页中显示的已添加锁的最大值。数据库服务器增加锁表大小满 99 次以后，该服务器将不再增加锁表的大小，这时需要锁的应用程序将收到一条错误消息。

下表显示了 32 位和 64 位平台上允许的最大锁数

表 5: 32 位和 64 位平台上的最大锁数

平台	最大初始锁数	最大动态锁表扩展数	每个锁表扩展添加的最大锁数	允许的最大锁数
32 位	8,000,000	99	100,000	8,000,000 + (99 x 100,000)
64 位	500,000,000	99	1,000,000	500,000,000 + (99 x 1,000,000)

LOCKS 配置参数的缺省值为 20,000。

要为 LOCKS 配置参数估算一个不同的值，可以估算查询所需的最大锁数并用此估算值乘以并发用户数。可以使用下表中的准则来估算查询需要的锁数。

每个语句的锁数	隔离级别	表	行	键	TEXT 或 BYTE 数据	CLOB 或 BLOB 数据
SELECT	Dirty Read	0	0	0	0	0
SELECT	Committed Read	1	0	0	0	0
SELECT	Cursor Stability	1	1	0	0	CLOB 或 BLOB 值使用 1 个锁，或（如果使用了字节范围锁定）每个范围使用 1 个锁
SELECT	Indexed Repeatable Read	1	满足条件的行数	满足条件的行数	0	CLOB 或 BLOB 值使用 1 个锁，或（如果使用了字节范围锁定）每个范围使用 1 个锁
SELECT	Sequential Repeatable Read	1	0	0	0	CLOB 或 BLOB 值使用 1 个锁，或（如果使用了字节范围锁定）每个范围使用 1 个锁
INSERT	不适用	1	1	索引数	TEXT 或 BYTE 数据中的页数	CLOB 或 BLOB 值使用 1 个锁
DELETE	不适用	1	1	索引数	TEXT 或 BYTE 数据中的页数	CLOB 或 BLOB 值使用 1 个锁
UPDATE	不适用	1	1	每个更改的键值使用 2 个	新旧 TEXT 或 BYTE 数据中的页数	CLOB 或 BLOB 值使用 1 个锁，或（如果使用了字节范围锁定）每个范围使用 1 个锁

重要： 在 SQL 语句 DROP DATABASE 的执行过程中，数据库服务器在数据库中每个表上获取并保留一个锁，直到整个 DROP 操作完成。确保 LOCKS 的值足够大以容纳数据库中的最大表数。

相关链接

配置和管理锁使用情况 在第182页

《*SinoDB* 管理员参考》: *LOCKS* 配置参数

RESIDENT 配置参数和内存利用率

RESIDENT 配置参数指定是否为数据库服务器共享内存的常驻部分强制使用共享内存驻留。此配置参数只有在支持强制驻留的计算机上有效。

数据库服务器中的常驻部分包含用于数据库读写活动的缓冲池。当这些缓冲区保留在物理内存中时会改善性能。

您应该将 RESIDENT 参数设置为 1。如果强制驻留不是计算机上的选项，那么数据库服务器会发出错误消息并忽略此配置参数。

在支持 64 位寻址的机器上，可以有很大的缓冲池，并且数据库服务器共享内存的虚拟部分也可以很大。虚拟部分包含有各种内存高速缓存，这些缓存可改善访问同一个表的多个查询的性能（请参阅配置和监视内存高速缓存 在第69页）。除常驻部分以外，要将虚拟部分驻留在物理内存中，需要将 RESIDENT 参数设置为 -1。

如果缓冲池非常大，但是物理内存却不是很大，那么可以将 RESIDENT 设置为大于 1 的值以指示保留在物理内存中的内存段数。此规范仅驻留缓冲池的子集。

可以用以下方式为共享内存的常驻部分打开或关闭驻留：

- 使用 onmode 实用程序在数据库服务器处于联机状态时暂时反转共享内存驻留的状态。
- 更改 RESIDENT 参数，以便在下次启动数据库服务器共享内存时打开或关闭共享内存驻留。

相关链接

《SinoDB 管理员参考》: RESIDENT 配置参数

SHMADD 和 EXTSHMADD 配置参数和内存利用率

SHMADD 配置参数是指定数据库服务器动态添加到虚拟部分的每个共享内存增量的大小。EXTSHMADD 配置参数会指定用户定义例程或 DataBlade® 例程在用户定义的虚拟处理器中运行时所添加的虚拟扩展段的大小。确定增量的大小要权衡利弊。

添加共享内存使用 CPU 周期。每个增量越大，需要的增量数越少，但是可用于其他进程的内存也越少。通常首选增加较大的增量；但是当内存负载很重时（扫描率或页调出率很高），较小的增量能使争用的程序之间更好地共享内存资源。

在 64 位操作系统上，SHMADD 值的范围为 1024 到 4294967296 KB；在 32 位操作系统上为 1024 到 524288 KB。下表包含根据物理内存的大小来设置 SHMADD 的建议。

内存大小	SHMADD 值
256 MB 或更少	8192 KB（缺省值）
257 - 512 MB	16,384 KB
大于 512 MB	32,768 KB

EXTSHMADD 值的范围与 SHMADD 值的范围相同。

> 注： 根据平台限制和 SHMMAX 内核参数的值，共享内存段可大至 4 TB。使用 onstat -g seg 命令可显示数据库服务器当前正在使用的共享内存段数。

相关链接

《SinoDB 管理员参考》: SHMADD 配置参数

《SinoDB 管理员参考》: EXTSHMADD 配置参数

配置 *UNIX* 共享内存 在第59页

SHMTOTAL 配置参数和内存利用率

SHMTOTAL 配置参数对数据库服务器实例可以使用的共享内存量设置了绝对上限。

如果 SHMTOTAL 配置参数设置为 0 或未指定，那么数据库服务器会根据需要继续连接其他共享内存，直到系统上没有任何可用的虚拟内存为止。

除以下情况外，一般可以将 SHMTOTAL 配置参数设置为 0：

- 必须限制数据库服务器用于其他应用程序或由于其他原因使用的虚拟内存量。
- 操作系统耗尽交换空间，无法正常运行。在这种情况下，您可以将 SHMTOTAL 设置为比计算机上可用的交换空间总量少几个 MB 的值。
- 您正在使用自动低内存管理。

相关链接

《*SinoDB* 管理员参考》: *SHMTOTAL* 配置参数

SHMVIRTSIZE 配置参数和内存利用率

SHMVIRTSIZE 参数指定启动数据库服务器时要分配的共享内存的虚拟部分大小。共享内存的虚拟部分保存特定于会话和请求的数据以及其他信息。

虽然数据库服务器可以根据需要为虚拟部分增加共享内存增量，以处理大型查询或承担峰值负载，但是分配共享内存也会增加事务处理的时间。因此，您应该设置 SHMVIRTSIZE 以提供足够大的虚拟部分来满足正常的日常操作需求。SHMVIRTSIZE 的大小可以大至 SHMMAX 配置参数所允许的大小。

必须是正整数的 SHMVIRTSIZE 最大值为：

- 4 TB（在 64 位数据库服务器上）
- 2 GB（在 32 位数据库服务器上）

对于初始设置，建议您使用以下更大的值：

- 8000
- *connections* * 350

connections 变量是在 sqlhosts 信息中由一个或多个 NETTYPE 配置参数指定的所有网络类型的连接数。（缺省情况下，数据库服务器使用 *connections* * 200。）

系统利用率达到稳定的工作负载之后，您就可以为 SHMVIRTSIZE 重新配置新值。如使用 *onmode -F* 释放共享内存 在第60页中所述，可以在工作负载高峰或大型查询过后，指示数据库服务器释放不再使用的共享内存段。

相关链接

《*SinoDB* 管理员参考》: *SHMVIRTSIZE* 配置参数

SHMVIRT_ALLOCSEG 配置参数和内存利用率

SHMVIRT_ALLOCSEG 配置参数指定数据库服务器应该分配内存的阈值。此配置参数也定义警报事件安全代码，如果服务器无法分配新的内存段，就激活此代码以确保数据库服务器不会用完内存。

设置 SHMVIRT_ALLOCSEG 配置参数时，您必须执行以下操作：

- 指定使用的内存百分比或服务器上剩余的 KB 总数。您不能使用负数或介于范围 0 和 .39 之间的值。
- 指定警报事件安全代码，该代码为介于范围 1（不需注意）到 5（致命）的值。如果未指定事件安全代码，那么服务器会将其设置为缺省值 3。

示例 1：

```
SHMVIRT_ALLOCSEG 3000, 4
```

这指定了如果数据库服务器在虚拟内存中有 3000 KB 的剩余量，而无法分配额外的千字节内存，那么服务器将发出级别为 4 的警报。

示例 2：

```
SHMVIRT_ALLOCSEG .8, 4
```

这指定了如果数据库服务器在虚拟内存中有 20% 的剩余量，而无法分配额外的千字节内存，那么服务器将发出级别为 4 的警报。

相关链接

《*SinoDB* 管理员参考》: 事件警报参数

《*SinoDB* 管理员参考》: *SHMVIRT_ALLOCSEG* 配置参数

STACKSIZE 配置参数和内存利用率

STACKSIZE 配置参数表示每个线程的初始堆栈大小。数据库服务器将此参数表示的空间量分配给每个活动线程。此空间来自于数据库服务器共享内存的虚拟部分。可以减少数据库服务器动态添加的共享内存量。

要减少数据库服务器动态添加的共享内存量，您可以估算系统运行的平均线程数所需的堆栈空间量，并将此量包含在为 SHMVIRTSIZE 配置参数设置的值中。

要估算您需要的堆栈空间量，可使用以下公式：

```
stacktotal = STACKSIZE * avg_no_of_threads
```

avg_no_of_threads

是平均线程数。可以定期监视活动线程数以确定此量。使用 onstat -g sts 检查线程的堆栈使用。根据您的工作负载，一般的估算介于连接总数（在 ONCONFIG 文件的 NETTYPE 参数中指定）的 60% 和 70% 之间。

数据库服务器也通过使用此堆栈的用户线程来执行用户定义例程 (UDR)。编写用户定义例程的编程人员应该采取以下措施避免堆栈溢出：

- 不要使用大的自动阵列。
- 避免过深地调用序列。
- 仅对于 *DB-Access:* 使用 mi_call 来管理递归调用。

如果无法使用这些措施来避免堆栈溢出，那么可以使用 CREATE FUNCTION 语句的 STACK 修饰符来增加特定例程的堆栈。

相关链接

《*SinoDB* 管理员参考》: *STACKSIZE* 配置参数

配置和监视内存高速缓存

数据库服务器使用高速缓存将信息存储到内存中，而不是执行磁盘读取或其他操作来获取信息。这些内存高速缓存为访问相同表的多个查询改善性能。您可以设置一些配置参数来提高每个高速缓存的效率。您可以通过执行 onstat 命令来查看有关内存高速缓存的信息。

下表列出对性能影响最大的主内存高速缓存以及如何配置和监视这些高速缓存。

表 6: 主内存高速缓存

高速缓存名称	高速缓存描述	配置参数	onstat 命令
数据字典	存储有关表定义（如列名和数据类型）的信息。	DD_HASHSIZE：高速缓存中的最大存储区数。 DD_HASHMAX：每个存储区中的表数	onstat -g dic
数据分布	存储列的分布统计信息。	DS_POOLSIZE：高速缓存中的最大条目数。 DS_HASHSIZE：高速缓存中的存储区数。	onstat -g dsc
SQL 语句	存储分析和优化的 SQL 语句。	STMT_CACHE：启用 SQL 语句高速缓存。	onstat -g ssc

高速缓存名称	高速缓存描述	配置参数	onstat 命令
		STMT_CACHE_HITS：SQL 语句在经过高速缓存之前所运行的次数。 STMT_CACHE_NOLIMIT：当分配的内存超过 STMT_CACHE_SIZE 配置参数的值时，禁止向 SQL 语句高速缓存插入条目。 STMT_CACHE_NUMPOOL：SQL 语句高速缓存的内存池数。 STMT_CACHE_SIZE：SQL 语句高速缓存的大小（以 KB 为单位）。	
UDR	存储频繁使用的用户定义例程和 SPL 例程。	PC_POOLSIZE：高速缓存中用户定义例程和 SPL 例程的最大数量。 PC_HASHSIZE：UDR 高速缓存中的存储区数。	onstat -g prc

下表列出其他内存高速缓存以及如何配置和监视这些高速缓存。

表 7: 其他内存高速缓存

高速缓存名称	高速缓存描述	配置参数	onstat 命令
访问方法	存储用户定义的访问方法。	无。	onstat -g cac am
聚合	存储用户定义的聚合。	DS_POOLSIZE DS_HASHSIZE	onstat -g cac agg
AQT 字典	存储数据库服务器用来确定哪些查询可由 SinoDB® Warehouse Accelerator 进行处理的加速查询表。	无。	onstat -g cac aqt
强制转型	存储用户定义的强制转型。	DS_POOLSIZE DS_HASHSIZE	onstat -g cac cast
外部指令	存储外部指令。	无。	onstat -g cac ed
LBAC 安全策略信息	存储 LBAC 安全策略。	PLCY_POOLSIZE PLCY_HASHSIZE	onstat -g cac lbacplcy
LBAC 凭证内存	存储 LBAC 凭证。	USRC_POOLSIZE USRC_HASHSIZE	onstat -g cac lbacusrc
运算符类实例	存储用户定义的运算符类。	DS_POOLSIZE DS_HASHSIZE	onstat -g cac opci
过程名称	存储用户定义例程和 SPL 例程名称。	PC_POOLSIZE PC_HASHSIZE	onstat -g cac prn
例程解析	存储用户定义例程解析信息。	DS_POOLSIZE	onstat -g cac rr

高速缓存名称	高速缓存描述	配置参数	onstat 命令
		DS_HASHSIZE	
二级瞬态	在高可用性集群中的辅助服务器上存储瞬态未命名复杂数据类型。	DS_POOLSIZE DS_HASHSIZE	onstat -g cac ttype
扩展类型标识符	存储用户定义类型的标识符。	DS_POOLSIZE DS_HASHSIZE	onstat -g cac typei
扩展类型名称	存储用户定义类型的名称。	DS_POOLSIZE DS_HASHSIZE	onstat -g cac typen

相关链接

存储在 *UDR* 高速缓存中的 *SPL* 例程可执行格式 在第234页

《*SinoDB* 管理员参考》: *onstat -g cac* 命令: 显示有关高速缓存的信息

《*SinoDB* 管理员参考》: *onstat -g dsc* 命令: 显示分布式高速缓存信息

《*SinoDB* 管理员参考》: *onstat -g prc* 命令: 显示使用 *UDR* 或 *SPL* 例程的会话

《*SinoDB* 管理员参考》: *onstat -g ssc* 命令: 显示 *SQL* 语句出现次数

《*SinoDB* 管理员参考》: 数据库配置参数

数据字典高速缓存

数据库服务器第一次访问表时，将从磁盘上的系统目录表中检索其需要的有关表信息（例如，列名和数据类型）。数据库服务器访问该表后，就将该信息放在共享内存的数据字典高速缓存中。

图 *5:* 数据字典高速缓存 在第71页显示数据库服务器如何将此高速缓存用于多个用户。用户 1 第一次访问 tabid 120 的列信息。数据库服务器将该列信息放在数据字典高速缓存中。当用户 2、用户 3 和用户 4 访问同一个表时，数据库服务器不必从磁盘读取以访问该表的数据字典信息。相反，而是从内存中的数据字典高速缓存读取字典信息。

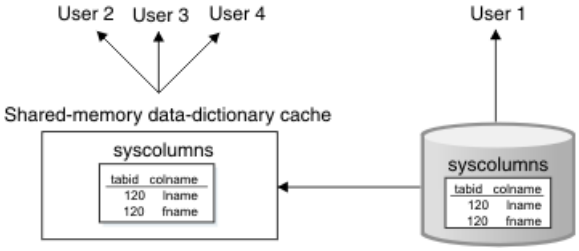

图 5: 数据字典高速缓存

数据库服务器仍然将系统目录表的页放在缓冲池中，就像处理所有其他数据和索引页一样。但是，数据字典高速缓存提供了额外的性能优势，因为数据字典信息的组织格式更有效，并且允许快速检索。

数据字典配置

数据库服务器使用散列算法在数据字典高速缓存内存储和查找信息。DD_HASHSIZE 和 DD_HASHMAX 配置参数将控制数据字典高速缓存的大小。

要修改数据字典高速缓存中的存储区数，使用 DD_HASHSIZE（必须为质数）。要修改可以在一个存储区中存储的表数，使用 DD_HASHMAX。

对于大中型系统，开始时可以为这些配置参数设置以下值：

- DD_HASHSIZE：503
- DD_HASHMAX：4

使用以上设置值，在数据字典高速缓存中最多可以存储 2012 个表的信息，每个散列存储区最多可包含 4 个表。

如果存储区的大小达到了最大值，数据库服务器将按照最近最少使用的机制来清除数据字典中的条目。

相关链接

《SinoDB 管理员参考》: DD_HASHSIZE 配置参数

《SinoDB 管理员参考》: DD_HASHMAX 配置参数

数据分布高速缓存

在 MEDIUM 或 HIGH 方式中，查询优化器使用 UPDATE STATISTICS 语句生成的分布统计信息来确定最低成本的查询计划。优化器第一次访问某列的分布统计信息时，数据库服务器将从磁盘上的 sysdistrib 系统目录表检索统计信息，并将该信息置于内存中的数据分布高速缓存内。

图 6: 数据分布高速缓存 在第72页显示数据库服务器如何对多个用户访问数据分布高速缓存。当优化器第一次为用户 1 访问列分布统计信息时，数据库服务器将该分布统计信息放在数据分布高速缓存中。当优化器为访问同一列的用户 2、用户 3 和用户 4 确定查询计划时，数据库服务器不必从磁盘中读取来访问表的数据分布信息。相反，而是从共享内存中的数据分布高速缓存读取分布统计信息。

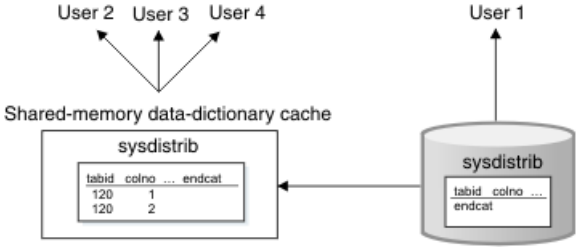

图 6: 数据分布高速缓存

数据库服务器最初将 sysdistrib 系统目录表的页放在缓冲池中，与处理所有其他数据和索引页一样。但是，数据分布高速缓存提供了额外的性能优势。它具有以下特点：

- 组织格式更有效
- 组织方式允许快速检索
- 避免了缓冲池管理的开销
- 释放缓冲池中的更多页，以供实际的数据页而不是系统目录页使用
- 减少了对系统目录表的 I/O 操作

数据分布配置

数据库服务器使用散列算法在数据分布高速缓存内存储和查找信息。DS_POOLSIZE 配置参数控制数据分布高速缓存的大小，以及可以在数据分布高速缓存中存储的列分布总数。DS_POOLSIZE 配置参数的值表示数据分布高速缓存中最大分布数的一半。

要修改数据分布高速缓存中的存储区数，请使用 DS_HASHSIZE 配置参数。

例如，使用 DS_POOLSIZE 的缺省值 127 和 DS_HASHSIZE 的缺省值 31，可能能够在数据分布高速缓存中存储大约 254 列的分布。高速缓存已满时，数据库服务器会自动将高速缓存的大小增大 10%。

为 DS_HASHSIZE 和 DS_POOLSIZE 设置的值取决于以下因素：

- 某些特定列的数量，您在 HIGH 或 MEDIUM 方式中对这些列执行 UPDATE STATISTICS 语句，而且您预期这些列最常用于那些频繁执行的查询中。

  如果在为表执行 UPDATE STATISTICS 时没有指定列，那么数据库服务器将为表中的所有列生成分布信息。

  您可以将 DD_HASHSIZE 和 DD_HASHMAX 的值用作 DS_HASHSIZE 和 DS_POOLSIZE 的准则。DD_HASHSIZE 和 DD_HASHMAX 可指定数据字典高速缓存的大小，该高速缓存存储信息以及查询访问的表的统计信息。

  对于大中型系统，开始时可以设置以下值：

  - DD_HASHSIZE 503
  - DD_HASHMAX 4
  - DS_HASHSIZE 503

- DS_POOLSIZE 1000

  通过执行 onstat -g dsc 命令查看实际使用情况来监视这些高速缓存，从而可以相应地调整这些参数。
- 可用内存量

  列存储分布时所需的内存量取决于执行 UPDATE STATISTICS 的级别。单个列的分布可能需要 1 KB 至 2 MB，这取决于您是指定 MEDIUM 还是 HIGH 方式，或者在执行 UPDATE STATISTICS 时输入了更精确的解析百分比。

如果数据分布高速缓存太小，那么可能会发生以下性能问题：

- 数据库服务器使用 DS_POOLSIZE 值以确定何时从数据分布高速缓存中移除条目。但是，如果优化器需要使用另一个查询的已删除分布信息，那么数据库服务器必须从磁盘上的 sysdistrib 系统目录表重新访问这些信息。访问磁盘上 sysdistrib 所需的额外 I/O 和缓冲池操作会加到查询的总响应时间中。

  数据库服务器尝试将数据分布高速缓存中的条目数维持在 DS_POOLSIZE 值。如果条目总数达到 DS_POOLSIZE 的一个内部阈值，那么数据库服务器将按照最近最少使用的机制从数据分布高速缓存中移除条目。散列存储区中的条目数可以超过此 DS_POOLSIZE 值，但数据库服务器最终会在内存需求降低时减少条目数。
- 如果 DS_HASHSIZE 很小而 DS_POOLSIZE 很大，那么溢出列表可能很长，并且在高速缓存中需要更多搜索时间。

  当散列存储区已经包含条目时会发生溢出。 如果多个分布散列到同一个存储区中，那么数据库服务器将保留一个溢出列表以存储和检索第一个分布之后的分布信息。

  如果 DS_HASHSIZE 和 DS_POOLSIZE 大小几乎相同，那么溢出列表可能较小甚至不存在，这会浪费内存。但是，未使用的内存量整体来说可以忽略。

如果看到以下情况，那么您可能要更改 DS_HASHSIZE 和 DS_POOLSIZE 配置参数的值：

- 如果数据分布高速缓存在大多数时间处于已满状态，而常用列没有列入 distribution name 字段中，请尝试增加 DS_HASHSIZE 和 DS_POOLSIZE 配置参数的值。
- 如果条目总数比 DS_POOLSIZE 配置参数的值小得多，那么可以减小 DS_HASHSIZE 和 DS_POOLSIZE 配置参数的值。
- 如果命中数未均匀分布在各个散列列表中，请通过增加 DS_HASHSIZE 配置参数的值来增加散列列表数。可以调整散列列表数，使每个散列列表具有最少的高命中条目数。

相关链接

《SinoDB 管理员参考》: DD_HASHSIZE 配置参数

《SinoDB 管理员参考》: DD_HASHMAX 配置参数

《SinoDB 管理员参考》: DS_POOLSIZE 配置参数

《SinoDB 管理员参考》: onstat -g dsc 命令: 显示分布式高速缓存信息

监视和调整 SQL 语句高速缓存

SQL 语句高速缓存会存储优化的 SQL 语句，以便执行相同 SQL 语句的多个用户能够实现一些性能改进。

这些性能改进为：

- 响应时间缩短，因为绕过优化步骤，如图 7: 使用 *SQL* 语句高速缓存时的数据库服务器操作 在第74页所示
- 占用的内存量减少，因为数据库服务器在用户间共享查询数据结构

有关 SQL 语句高速缓存对单个查询的性能影响的更多信息，请参阅使用 *SQL* 语句高速缓存优化查询 在第295页。

图 7: 使用 *SQL* 语句高速缓存时的数据库服务器操作 在第74页显示数据库服务器如何对多个用户访问 SQL 语句高速缓存。

- 当数据库服务器第一次为用户 1 执行 SQL 语句时，数据库服务器会检查完全一样的 SQL 语句是否在 SQL 语句高速缓存中。如果不在高速缓存中，那么数据库服务器会分析该语句，确定最佳的查询计划，然后执行该语句。

- 当用户 2 执行相同的 SQL 语句时，数据库服务器会在 SQL 语句高速缓存中查找该语句，但不会优化该语句。
- 同样，如果用户 3 和用户 4 执行相同的 SQL 语句，数据库服务器也不会优化该语句。相反，它会使用内存中 SQL 语句高速缓存内的查询计划。

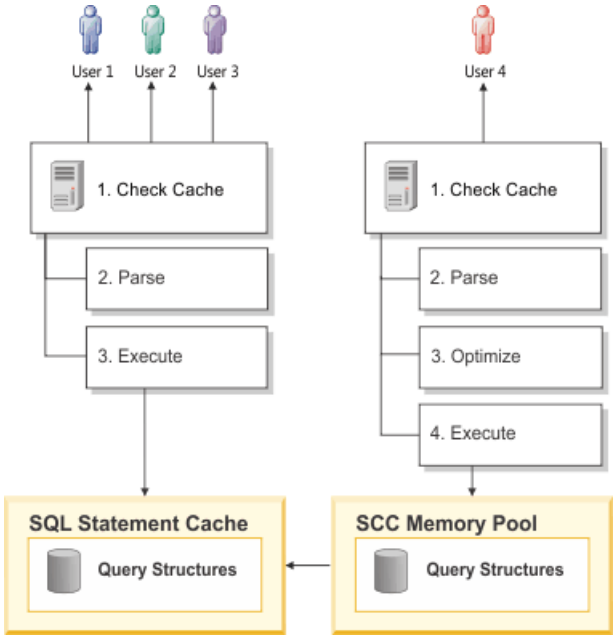

图 7: 使用 SQL 语句高速缓存时的数据库服务器操作

预编译语句和语句高速缓存
在本质上为单个会话高速缓存的预编译语句。这意味着如果多次执行预编译语句或多次打开单个游标，那么会话将使用相同的预编译查询计划。

如果某个会话准备好一个语句，然后多次执行该语句，那么 SQL 语句高速缓存将不会影响性能，因为该语句在 PREPARE 语句中只优化一次。

但是，如果其他会话也准备这个相同的语句，或者第一个会话多次准备该语句，那么语句高速缓存通常会提供直接的性能优势，因为数据库服务器只对查询计划计算一次。当然，即使初始会话只准备语句一次，它也可能从语句高速缓存中获取微小的益处，因为其他会话使用的内存较少，而数据库服务器对其他会话提供的工作也较少。

SQL 语句高速缓存配置
STMT_CACHE 配置参数的值可以启用或禁用 SQL 语句高速缓存。

有关 STMT_CACHE 配置参数的值如何启用 SQL 语句高速缓存的更多信息，请参阅启用 *SQL* 语句高速缓存 在第296页。

图 *8:* 影响 *SQL* 语句高速缓存的配置参数 在第75页显示数据库服务器如何对 SQL 语句高速缓存使用相关配置参数的值。图下附有更详细的说明。

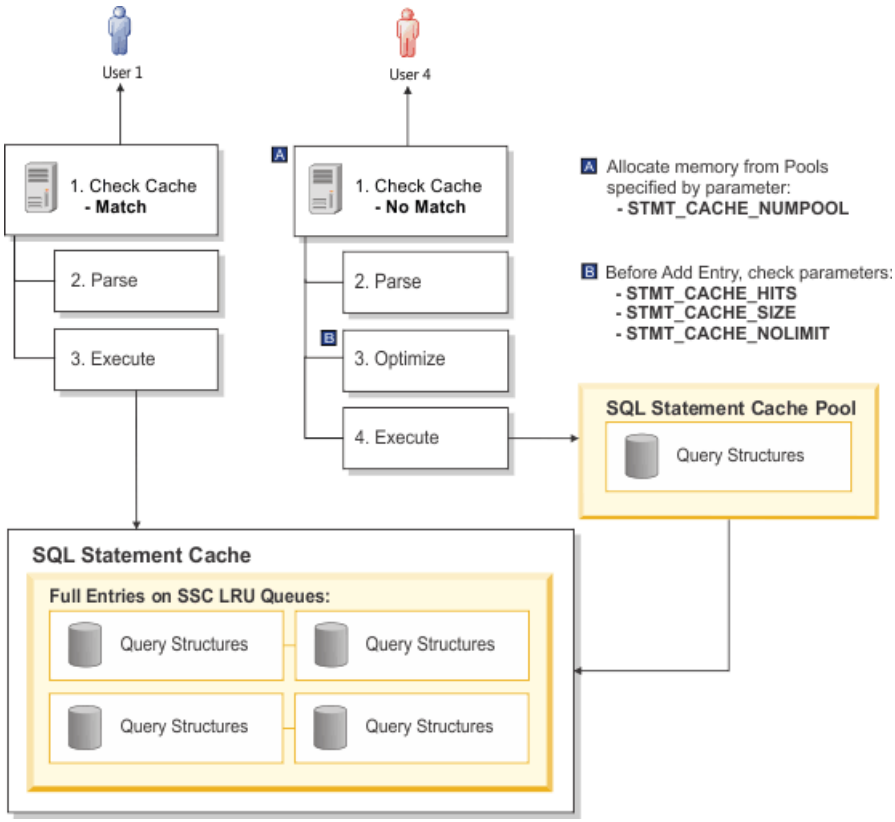

图 8: 影响 SQL 语句高速缓存的配置参数

当数据库服务器为用户使用 SQL 语句高速缓存时，意味着数据库服务器会执行以下操作：

- 首先检查 SQL 语句高速缓存中是否有与用户正在执行的 SQL 语句匹配的语句
- 如果 SQL 语句与某条目匹配，那么使用 SQL 语句高速缓存中的查询内存结构执行该语句（图 *8: 影响 SQL 语句高速缓存的配置参数* 在第75页中的用户 2）
- 如果 SQL 语句没有匹配的条目，那么数据库服务器会检查其是否符合高速缓存的条件。

  有关哪些因素使 SQL 语句符合高速缓存条件的信息，请参阅 *SQL statement cache qualifying criteria*。
- 如果 SQL 语句满足条件，那么向高速缓存插入一个条目用于该语句的后续执行。

以下参数影响数据库服务器是否将 SQL 语句插入高速缓存（图 *8: 影响 SQL 语句高速缓存的配置参数* 在第75页中的用户 1）：

- STMT_CACHE_HITS 指定使用高速缓存中的条目执行该语句的次数（称为命中数）。数据库服务器根据命中数插入以下其中一个条目：
  - 如果 STMT_CACHE_HITS 的值为 0，那么插入完全高速缓存的条目，其中包含 SQL 语句的文本以及查询内存结构。
  - 如果 STMT_CACHE_HITS 的值不为 0 且该语句不存在于高速缓存中，那么插入包含 SQL 语句的文本的仅含键条目。SQL 语句的后续执行会增加命中数。
  - 如果 STMT_CACHE_HITS 的值等于仅含键条目的命中数，那么添加查询内存结构使该条目变为完全高速缓存的条目。
- STMT_CACHE_SIZE 指定 SQL 语句高速缓存的大小，而 STMT_CACHE_NOLIMIT 指定是否将高速缓存的内存限制为 STMT_CACHE_SIZE 的值。如果您不指定 STMT_CACHE_SIZE 参数，那么将为缺省值 524288（512 * 1024）字节。

  STMT_CACHE_NOLIMIT 的缺省值为 1，这表示数据库服务器将把条目插入 SQL 语句高速缓存中，即使内存总量可能超过 STMT_CACHE_SIZE 的值。

  当 STMT_CACHE_NOLIMIT 设置为 0 时，如果高速缓存的当前大小不会超过内存限制，那么数据库服务器将 SQL 语句插入高速缓存。

以下有关 STMT_CACHE_HITS、STMT_CACHE_SIZE、STMT_CACHE_NOLIMIT、 STMT_CACHE_NUMPOOL 的章节提供更多详细信息以说明下列配置参数如何影响 SQL 语句高速缓存，以及可能需要更改其缺省值的理由。

SQL 语句执行次数
当启用 SQL 语句高速缓存时，缺省情况下数据库服务器会将满足条件的 SQL 语句及其内存结构立即插入 SQL 语句高速缓存中。

如果工作负载的即席查询数不成比例，那么在数据库服务器将完全高速缓存的条目置入语句高速缓存之前，请使用 STMT_CACHE_HITS 配置参数来指定执行 SQL 语句的次数。

当 STMT_CACHE_HITS 配置参数大于 0 且 SQL 语句已执行的次数少于 STMT_CACHE_HITS，那么数据库服务器将仅含键的条目插入高速缓存。此规范最大程度地降低了未共享的内存结构所占用的语句高速缓存量，从而为应用程序常用的 SQL 语句留出了更多内存空间。

监视 SQL 语句高速缓存上的命中数，以确定工作负载是否正有效使用此高速缓存。以下部分描述监视 SQL 语句高速缓存命中数的方式。

相关链接

《SinoDB 管理员参考》: STMT_CACHE_HITS 配置参数

SQL 语句高速缓存中一次性查询过多 在第79页

监视 SQL 语句高速缓存的命中数
要监视 SQL 语句高速缓存中的命中数，请执行 onstat -g ssc 命令。

onstat -g ssc 命令显示 SQL 语句高速缓存中完全高速缓存的条目。图 *9: onstat -g ssc 输出* 在第76页显示 onstat -g ssc 输出的示例。

```
onstat -g ssc

Statement Cache Summary:
#lrus   currsize  maxsize   Poolsize  #hits   nolimit
4       49456     524288    57344     0       1

Statement Cache Entries:

lru hash ref_cnt hits flag heap_ptr      database           user
------------------------------------------------------------------------------
  0  153       0    0   -F  a7e4690      vjp_stores         virginia
  SELECT * FROM customer, orders
    WHERE customer.customer_num = orders.customer_num
    AND order_date > "01/01/07"

  1  259       0    0    -F  aa58c20     vjp_stores         virginia
  SELECT * FROM customer, orders
    WHERE customer.customer_num = orders.customer_num
    AND order_date > "01/01/2007"

  2  232       0    1    DF  aa3d020     vjp_stores         virginia
  SELECT C.customer_num, O.order_num
    FROM customer C, orders O, items I
    WHERE C.customer_num = O.customer_num
    AND O.order_num = I.order_num

  3  232       1    1    -F  aa8b020     vjp_stores         virginia
  SELECT C.customer_num, O.order_num
    FROM customer C, orders O, items I
    WHERE C.customer_num = O.customer_num
    AND O.order_num = I.order_num
```

```
    Total number of entries: 4.
```

图 9: onstat -g ssc 输出

要监视数据库服务器在高速缓存内读取 SQL 语句的次数，请查看以下输出列：

- 在 onstat -g ssc 输出的 Statement Cache Summary 部分中，#hits 列是 SQL_STMT_HITS 配置参数的值。

  在 图 9: *onstat -g ssc* 输出 在第76页中，输出的 Statement Cache Summary 部分中的 #hits 列具有 0 值，这是 STMT_CACHE_HITS 配置参数的缺省值。

  > 重要： 数据库服务器仅在语句完全相同时才使用 SQL 语句高速缓存中的条目。图 9: *onstat -g ssc* 输出 在第76页中的前两个条目不相同，因为各自在 order_date 过滤器中包含不同的文字值。

- 在 onstat -g ssc 输出的 Statement Cache Entries 部分中，hits 列显示数据库服务器执行高速缓存中每个单独 SQL 语句的次数。换言之，该列显示数据库服务器使用高速缓存中的内存结构的次数而不是优化语句以再次生成语句的次数。

  数据库服务器第一次将语句插入到高速缓存中时，hits 值为 0。

  - 图 9: *onstat -g ssc* 输出 在第76页中前两个 SQL 语句的 hits 列值为 0，表示每个语句都已插入高速缓存中但尚未从高速缓存中执行。
  - 图 9: *onstat -g ssc* 输出 在第76页中最后两个 SQL 语句的 hits 列值为 1，表示这两个语句都已从高速缓存中执行过一次。

  各个条目的 hits 值指示内存结构的共享程度。hits 列中较高的值表示 SQL 语句高速缓存对于改善性能和内存使用率的帮助较大。

有关 onstat -g ssc 所显示输出字段的完整描述，请参阅 *onstat -g ssc* 输出中的 *SQL* 语句高速缓存信息 在第81页。

确定 SQL 语句高速缓存中非共享条目的数量
要确定在 SQL 语句高速缓存中存在的非共享条目数，请执行 onstat -g ssc all。

onstat -g ssc all 选项除显示 SQL 语句高速缓存中已完全高速缓存的条目外，还显示仅含键的条目。

要确定高速缓存中存在的非共享条目数：

1. 将 onstat -g ssc all 输出与 onstat -g ssc 输出相比较。
2. 如果这两个输出之间的差别显示 SQL 语句高速缓存中存在的许多非共享条目，那么应提高 STMT_CACHE_HITS 配置参数的值以允许更多共享语句驻留在高速缓存中，从而减少 SQL 语句高速缓存的管理开销。

您可以使用以下一种方法来更改 STMT_CACHE_HITS 参数值：

- 更新 ONCONFIG 文件以指定 STMT_CACHE_HITS 配置参数。必须重新启动数据库服务器以使新值生效。

  您可以使用文本编辑器来编辑 ONCONFIG 文件。然后，用 onmode -ky 命令关闭数据库服务器并用 oninit 命令重新启动。

- 数据库服务器在运行时动态增大 STMT_CACHE_HITS 配置参数：

  可以使用以下任意方法在运行时复位 STMT_CACHE_HITS 值：

  - 发布 onmode -W 命令。以下示例指定在向语句高速缓存添加新查询之前需要三个 (3) 实例：

    ```
    onmode -W STMT_CACHE_HITS 2
    ```

  - 调用 SQL 管理 API 的 ADMIN 或 TASK 函数。以下示例等同于上述示例中的 onmode 命令：

    ```
    EXECUTE FUNCTION TASK("ONMODE", "W", "STMT_CACHE_HITS", "2");
    ```

  如果动态增大 STMT_CACHE_HITS 而不更新配置文件，然后重新启动数据库服务器，那么 STMT_CACHE_HITS 设置将还原 ONCONFIG 文件中的值。因此，如果希望以后重新启动后该设置保持不变，请修改 ONCONFIG 文件。

监视和调整 SQL 语句高速缓存的大小
如果 SQL 语句高速缓存太小，那么会发生性能问题。可以监视 SQL 语句高速缓存大小的有效性。

会发生以下性能问题：

- 频繁执行的 SQL 语句不在高速缓存中

  最常使用的语句应该保留在 SQL 语句高速缓存中。如果 SQL 语句高速缓存不够大，那么当其他语句进入高速缓存时，数据库服务器可能没有足够的空间来保留这些语句。有关后续执行，数据库服务器必须重新分析、重新优化并将 SQL 语句重新插入高速缓存中。尝试增加 STMT_CACHE_SIZE。
- 数据库服务器花费大量时间清除 SQL 语句高速缓存

  数据库服务器尝试通过使用阈值（STMT_CACHE_SIZE 参数的 70%）来确定何时从 SQL 语句高速缓存中移除条目，从而防止 SQL 语句高速缓存分配大量内存。如果新条目引起 SQL 语句高速缓存的大小超过阈值，那么数据库服务器在插入新条目前会移除最近最少使用的条目（当前未使用）。

  但是，如果后续查询需要使用已移除的内存结构，那么数据库服务器必须重新分析并重新优化 SQL 语句。重新生成这些内存结构所需要的额外处理时间会使查询的总响应时间增加。

您可以使用 STMT_CACHE_SIZE 配置参数来设置内存中 SQL 语句高速缓存的大小。该参数的值是以 KB 为单位表示的大小。如果未设置 STMT_CACHE_SIZE，那么缺省值为 512 KB。

onstat -g ssc 输出显示 maxsize 列中 STMT_CACHE_SIZE 的值。在 图 9: *onstat -g ssc* 输出 在第76页中，该 maxsize 列值为 524288，即缺省值 (512 * 1024 = 524288)。

使用 onstat -g ssc 和 onstat -g ssc all 选项监视 SQL 语句高速缓存大小的有效性。如果在高速缓存中看不到应用程序最常用的 SQL 语句的条目，那么 SQL 语句高速缓存可能太小或者有过多的非共享 SQL 语句占用了高速缓存。以下各章节描述如何确定这些情况。

相关链接

《SinoDB 管理员参考》: STMT_CACHE_SIZE 配置参数

更改 SQL 语句高速缓存的大小
可以分析 onstat -g ssc all 输出以确定 SQL 语句高速缓存是否太小。如果高速缓存太小，那么您可予以更改。

要确定 SQL 语句高速缓存是否过小：

1. 执行 onstat -g ssc all 可确定高速缓存是否太小。
2. 查看 onstat -g ssc all 输出的语句高速缓存条目部分中以下输出列的值：
   - flags 列显示高速缓存中 SQL 语句的当前状态。

     第二个位置中的 F 值，指示该语句当前已完全高速缓存。

     第二个位置中的 - 值，指示只有语句文本（仅含键的条目）位于高速缓存中。第二个位置中带有此 - 值的条目出现在 onstat -g ssc all 输出中，但不出现在 onstat -g ssc 输出中。
   - hits 列显示 SQL 语句已经执行的次数，不包括插入高速缓存时的第一次执行。

   如果在高速缓存中看不到应用程序最常用语句的完全高速缓存的条目，且 hits 列中的值相对于占用高速缓存的条目来说很大，那么说明 SQL 语句高速缓存太小。

要更改 SQL 语句高速缓存的大小：

1. 更新 STMT_CACHE_SIZE 配置参数的值。
2. 重新启动数据库服务器以使新值生效。

相关链接

《SinoDB 管理员参考》: STMT_CACHE_SIZE 配置参数

SQL 语句高速缓存中一次性查询过多
当数据库服务器将许多只使用一次的查询放入高速缓存中时，它们可能会取代其他应用程序常用的语句。可以查看 onstat -g ssc all 输出以确定是否过多未共享的 SQL 语句占用高速缓存。如果是这样，您可以防止未共享的 SQL 语句被完全高速缓存。

查看 onstat -g ssc all 输出的 Statement Cache Entries 中以下输出列的值。如果看到大量具有以下两个值的条目，那么说明过多的非共享 SQL 语句占用了高速缓存：

- flags 列第二个位置中的值为 F

  第二个位置中的 F 值指示该语句当前已完全高速缓存。
- hits 列值为 0 或 1

  hits 列显示 SQL 语句已经执行的次数，不包括插入高速缓存时的第一次执行。

增加 STMT_CACHE_HITS 配置参数的值以防止未共享的 SQL 语句被完全高速缓存。

相关链接

《SinoDB 管理员参考》: STMT_CACHE_HITS 配置参数

SQL 语句执行次数 在第76页

内存限制和大小
虽然数据库服务器尝试清除 SQL 语句高速缓存，但是有时候由于当前正使用而不能移除该条目。在这种情况下，SQL 语句高速缓存的大小可能会超过 STMT_CACHE_SIZE 配置参数的值。

STMT_CACHE_NOLIMIT 配置参数的缺省值为 1，这表示即使高速缓存的当前大小可能大于 STMT_CACHE_SIZE 参数的值，数据库服务器也会插入该语句。

如果 STMT_CACHE_NOLIMIT 配置参数的值为 0，那么当大小超过 STMT_CACHE_SIZE 的值时，数据库服务器不会将完全限定的条目或仅含键的条目插入 SQL 语句高速缓存中。

使用 onstat -g ssc 选项监视 SQL 语句高速缓存的当前大小。查看 onstat -g ssc 输出的以下输出列中的值：

- currsize 列显示 SQL 语句高速缓存中当前分配的字节数。

  在 *图 9: onstat -g ssc 输出* 在第76页中，currsize 列值为 11264。
- maxsize 列显示 STMT_CACHE_SIZE 的值。

  在 *图 9: onstat -g ssc 输出* 在第76页中，maxsize 列值为 524288，即缺省值 (512 * 1024 = 524288)。

当 SQL 语句高速缓存已满且用户当前正在执行其中的所有语句时，用户执行的任何新 SQL 语句都会导致 SQL 语句高速缓存增长超过 STMT_CACHE_SIZE 指定的大小。当数据库服务器不再使用 SQL 语句高速缓存内的 SQL 语句时，该服务器将释放 SQL 语句高速缓存中的内存，直至大小达到 STMT_CACHE_SIZE 阈值为止。但是，如果数以千计的用户同时执行几个即席查询，那么 SQL 查询高速缓存会在移除任何语句之前迅猛增长。在这种情况下，可采取以下操作之一：

- 将 STMT_CACHE_NOLIMIT 配置参数设置为 0 以防止在高速缓存大小超过 STMT_CACHE_SIZE 参数的值时执行任何插入操作。
- 将 STMT_CACHE_HITS 参数设置为大于 0 的值以防止高速缓存未共享的 SQL 语句。

您可以使用以下一种方法来更改 STMT_CACHE_NOLIMIT 配置参数值：

- 更新 ONCONFIG 文件以指定 STMT_CACHE_NOLIMIT 配置参数。必须重新启动数据库服务器以使新值生效。
- 使用 onmode -W 命令可在数据库服务器运行时动态覆盖 STMT_CACHE_NOLIMIT 配置参数。

  ```
  onmode -W STMT_CACHE_NOLIMIT 0
  ```

  如果重新启动数据库服务器，那么该值将恢复为 ONCONFIG 文件中的值。因此，如果希望以后重新启动时保持该设置，那么修改 ONCONFIG 文件。

相关链接

《SinoDB 管理员参考》: STMT_CACHE_HITS 配置参数

多个 SQL 语句高速缓存池
在某些情况下，启用 SQL 语句高速缓存时，数据库服务器会从一个池中为查询结构分配内存。

这些情况为：

- 当数据库服务器在高速缓存中没有找到匹配条目时
- 当数据库服务器在高速缓存中找到匹配的仅含键条目，且命中数达到 STMT_CACHE_HITS 配置参数的值时

随着用户数量的增加，该池可能会成为瓶颈。STMT_CACHE_NUMPOOL 配置参数使您能够配置多个 sscpools。

可以监视 SQL 语句高速缓存中的池来确定以下情况：

- SQL 语句高速缓存池的数量对于您的工作负载已足够。
- SQL 语句高速缓存的大小或限制没有引起过多的内存管理。

相关链接

《SinoDB 管理员参考》: STMT_CACHE_NUMPOOL 配置参数

SQL 语句高速缓存池数
启用 SQL 语句高速缓存 (SSC) 时，数据库服务器会从 SSC 池为未链接的 SQL 语句分配内存。STMT_CACHE_NUMPOOL 配置参数的缺省值是 1。随着用户数量的增加，这个 SSC 池可能会成为瓶颈。

SSC 池上长自旋的数量表示该 SSC 池是否为瓶颈。

使用 onstat -g spi 选项来监视 SSC 池上的长自旋数。onstat -g spi 命令显示获取资源上的锁存器之前系统中需要等待的资源列表。在等待期间，线程旋转（或循环），尝试获取资源。onstat -g spi 输出显示为获取资源必须等待的次数（Num Waits 列）以及总循环数（Num Loops 列）。onstat -g spi 输出仅显示至少等待一次的资源。

图 10: onstat -g spi 输出 在第80页显示 onstat -g spi 输出示例的摘录。*图 10: onstat -g spi 输出* 在第80页指示对于任何 SSC 池都没有发生任何等待（Name 列没有列出任何 SSC 池）。

```
Spin locks with waits:
Num Waits    Num Loops    Avg Loop/Wait     Name
34477        387761       11.25             mtcb sleeping_lock
312          10205        32.71             mtcb vproc_list_lock
```

图 10: onstat -g spi 输出

如果在 SSC 池上看到数量过多的长自旋（Num Loops 列），请增加 STMT_CACHE_NUMPOOL 配置参数中的 SSC 池数以提高性能。

相关链接

《SinoDB 管理员参考》: STMT_CACHE_NUMPOOL 配置参数

SQL 语句高速缓存池大小和当前高速缓存大小
使用 onstat -g ssc pool 选项来监视每个 SQL 语句高速缓存 (SSC) 池的使用情况。

onstat -g ssc pool 命令显示每个池的大小。onstat -g ssc 选项在 currsize 列中显示 SQL 语句高速缓存的累积大小。此当前大小是由插入到高速缓存中的语句从 SSC 池分配的内存大小。由于并非从 SSC 池分配内存的所有语句都插入到高速缓存中，当前的高速缓存大小可能要小于 SSC 池的总大小。通常，所有 SSC 池的总大小不会超过 STMT_CACHE_SIZE 值。

图 11: onstat -g ssc pool 输出 在第80页显示 onstat -g ssc pool 输出的示例。

```
onstat -g ssc pool


Pool Summary:
name         class addr     totalsize freesize #allocfrag #freefrag
sscpool0     V     a7e4020  57344      2352     52         7

Blkpool Summary:
```

```
name         class addr     size      #blks
```

图 11: onstat -g ssc pool 输出

onstat -g ssc pool 输出的 Pool Summary 部分为该高速缓存中的每个池列出以下信息。

列	描述
name	SQL 语句高速缓存 (SSC) 池的名称
class	已创建池的共享内存段。对于 SSC 池，此值始终是“V”，代表共享内存的虚拟部分。
addr	SSC 池结构的共享内存地址
totalsize	此 SSC 池的总大小（以字节为单位）
freesize	此 SSC 池中可用字节数
#allocfrag	在此 SSC 池中已分配的连续内存区域数
#freefrag	此 SSC 池中未使用的连续内存区域数

onstat -g ssc poo 输出的 Blkpool Summary 部分为该高速缓存中的所有池列出以下信息。

列	描述
name	SSC 池的名称
class	已创建池的共享内存段。对于 SSC 池，此值始终是“V”，代表共享内存的虚拟部分。
addr	SSC 池结构的共享内存地址
totalsize	此 SSC 池的总大小（以字节为单位）
#blks	组成所有 SSC 池的 8 KB 块的数量

onstat -g ssc 输出中的 SQL 语句高速缓存信息
onstat -g ssc 命令显示 SQL 语句高速缓存的摘要信息。

onstat -g ssc 命令显示 SQL 语句高速缓存的以下信息。

表 8: onstat -g ssc 输出中的 SQL 语句高速缓存信息

列	描述
#lrus	LRU 队列数。使用多个 LRU 队列便于实现同时查找和插入高速缓存条目。
currsize	当前分配给 SQL 语句高速缓存中的条目的字节数
maxsize	STMT_CACHE_SIZE 配置参数中指定的字节数
poolsize	SQL 语句高速缓存中所有池的累积字节数。使用 onstat -g ssc pool 选项可监视单个池使用情况。
#hits	STMT_CACHE_HITS 配置参数的设置，它指定在查询插入高速缓存之前执行的次数
nolimit	STMT_CACHE_NOLIMIT 配置参数的设置

onstat -g ssc 命令为高速缓存中每个完全高速缓存条目列出以下信息。onstat -g ssc all 选项为完全高速缓存的条目和仅含键的条目列出以下信息。

列	描述
lru	LRU 标识符
hash	hash-bucket 标识符
ref_cnt	当前使用此语句的会话数
hits	用户从高速缓存读取查询的次数（不包括该语句第一次进入高速缓存时）
flags	显示标志代码。 位置 1 的标志代码为： D 指示该语句已被删除 一个语句的某一相关性更改时，会从高速缓存中将该语句删除（不再使用）。例如，执行表的 UPDATE STATISTICS 时，优化器统计信息可能会发生更改，从而导致高速缓存中 SQL 语句的查询计划过时。在此情况下，数据库服务器在下次尝试使用该语句时会将其标记为已删除。 - 指示该语句未被删除 位置 2 的标志代码为： F 指示查询高速缓存条目已完全高速缓存并包含该查询的内存结构 I 指示该语句正在被移至完全高速缓存状态的过程中 - 指示未将语句完全高速缓存 当执行语句的次数少于 STMT_CACHE_HITS 配置参数的值时，该语句不能完全高速缓存。在第二个位置中带有此值的条目出现在 onstat -g ssc all 输出中，而不是 onstat -g ssc 输出中。
heap_ptr	指向该语句相关堆的指针
database	执行 SQL 语句的数据库
user	执行 SQL 语句的用户
statement	测试是否匹配时使用的语句文本

会话内存

数据库服务器将共享内存的虚拟部分主要用于用户会话。每个用户会话分配的内存大部分都用于 SQL 语句。可以确定哪个会话和哪些语句具有高内存利用率。如果必要的话，可以设置 SESSION_LIMIT_MEMORY 配置参数来限制可用于会话的内存量。

使用以下实用程序选项可以确定哪个会话和准备就绪的 SQL 语句内存利用率高：

- onstat -g mem

- onstat -g stm

onstat -g mem 选项显示所有会话的内存使用情况。可以通过查看 totalsize 和 freesize 输出列来找到正使用大多数内存的会话。下图显示 onstat -g mem 输出的示例。此输出示例显示三个用户会话的内存利用率，这三个用户会话在 names 输出列中值为 14、16 和 17。

```
onstat -g mem

Pool Summary:
name         class addr     totalsize freesize #allocfrag #freefrag
...
14           V     a974020  45056     11960    99         10
16           V     a9ea020  90112     10608    159        5
17           V     a973020  45056     11304    97         13
...
Blkpool Summary:
name         class addr     size      #blks
mt           V     a235688  798720    19
global       V     a232800  0         0
```

图 12: onstat -g mem 输出

要显示由每个预编译语句所分配的内存，请使用 onstat -g stm 选项。下图显示 onstat -g stm 输出的示例。

```
onstat -g stm


session   25 ------------------------------------------------
 sdblock  heapsz  statement ( '*' = Open cursor)
 d36b018    9216  select sum(i) from t where i between -1 and ?
 d378018    6240  *select tabname from systables where tabid=7
 d36b114    8400  <SPL statement>
```

图 13: onstat -g stm 输出

图 *13: onstat -g stm* 输出 在第83页中输出内的 heapsz 列显示语句使用的内存量。如果该语句上有一个打开的游标，那么语句文本前会显示一个星号 (*)。该输出不显示 SPL 例程中的单个 SQL 语句。

要只显示一个会话的内存，可在 onstat -g stm 选项中指定会话标识符。有关示例，请参阅使用 *onstat -g mem* 和 *onstat -g stm* 输出监视会话内存 在第305页。

设置 SESSION_LIMIT_MEMORY 配置参数以限制会话可分配的内存量，并可防止单个会话独占系统资源。该限制不适用于拥有管理权限的用户，例如用户 informix 或 DBSA 用户。

例如，要将每个会话限制为 10 MB 内存，请在 ONCONFIG 文件中设置 SESSION_LIMIT_MEMORY 10240。

相关链接

估算共享内存虚拟部分的大小 在第57页

《*SinoDB* 管理员参考》: *SESSION_LIMIT_MEMORY* 配置参数

数据复制缓冲区和内存利用率

数据复制需要数据库服务器的两个实例，一个主实例和一个辅实例，分别运行在两台计算机上。如果为数据库服务器执行数据复制，那么数据库服务器在逻辑日志记录发送到辅助数据库服务器之前会将其保留在数据复制缓冲区中。

数据复制缓冲区的大小始终与逻辑日志缓冲区大小一样。

内存锁存器

数据库服务器使用锁存器控制对共享内存结构（如缓冲池或 SQL 语句高速缓存的内存池）的访问权限。可以获取有关锁存器使用的统计信息以及特定锁存器的信息。这些统计信息可用作衡量系统活动的标准。

该统计信息包括线程为获取一个锁存器而等待的次数。如果存在大量等待锁存器的现象，通常是因为大量的处理操作中数据库服务器为多数事务记录日志。

有关特定锁存器的信息包括一个线程所持有的所有锁存器列表以及正在等待锁存器的任何线程。该信息使您能够发现存在的任何特定资源争用情况。

您作为数据库管理员将无法配置或调整锁存器数。但是，您可增加数据库服务器放置锁存器的内存结构的数量，以减少锁存器等待数。例如，您可以调整 SQL 语句高速缓存内存池数或 SQL 语句高速缓存 LRU 队列数。有关更多信息，请参阅多个 *SQL* 语句高速缓存池 在第80页。

警告： 请勿停止正持有锁存器的数据库服务器进程。否则，数据库服务器将立即异常终止。

使用命令行实用程序监视锁存器

可以通过执行 onstat -p 或 onstat -s 来获取有关锁存器的信息。

使用 onstat -p 监视锁存器
执行 onstat -p 以获取 lchwaits 字段中的值。该字段用于存储一个线程需要等待共享内存锁存器的次数。

图 *14: 显示 lchwaits 字段的 onstat -p 部分输出* 在第84页显示 onstat -p 输出的一部分，该输出显示的是 lchwaits 字段。

```
...
ixda-RA  idx-RA   da-RA   logrec-RA    RA-pgsused lchwaits
5        0        204      0           148        12
```

图 14: 显示 lchwaits 字段的 onstat -p 部分输出

相关链接

《SinoDB 管理员参考》: onstat -p 命令: 显示概要文件计数

使用 onstat -s 监视锁存器
执行 onstat -s 以获取一般锁存器信息。该输出包括 userthread 列，其中列出正等待锁存器的所有用户线程的地址。

可以将此地址与 onstat -u 输出中的用户地址比较以获取用户进程标识号。

图 *15: onstat -s 输出* 在第84页显示 onstat -s 输出的示例。

```
...
Latches with lock or userthread set
name     address  lock wait userthread
LRU1     402e90   0    0        6b29d8
bf[34]   4467c0   0    0        6b29d8
...
```

图 15: onstat -s 输出

使用 SMI 表监视锁存器

可以查询 sysprofile SMI 表以获取线程等待锁存器的次数。

sysprofile 表的 latchwts 列包含线程等待锁存器的次数。

第 5 章

配置对 I/O 活动的影响

数据库服务器的配置会影响 I/O 活动。

以下因素会影响 I/O 活动：

- 对块和数据库空间的分配会创建 I/O 热点，或具有不均衡 I/O 活动量的磁盘分区。
- 分配关键数据、排序区域以及用于临时文件和建立索引的区域时，系统会在各个磁盘上产生间歇性的负载。
- 对于预读的配置可提升单个 I/O 操作的效能。
- 对于后台 I/O 任务（比如日志记录和页清除）的配置可影响 I/O 吞吐量。

块和数据库空间配置

使用的磁盘数以及块、数据库空间和 BLOB 空间的配置会影响数据库服务器的性能。可以通过规划磁盘使用以及块、数据库空间和 BLOB 空间的配置来改善性能。

数据库中的所有数据将存储在磁盘上。数据库服务器将相应数据页复制到磁盘和从磁盘上复制相应数据页的速度决定了您应用程序的性能。

磁盘通常是 I/O 路径中最慢的组件，用于存放完全在一台主机上运行的事务或查询。网络通信也会在客户端/服务器应用程序中引起延迟，但是数据库服务器管理员通常无法控制这些延迟。有关数据库服务器管理员为改善网络通信所执行操作的信息，请参阅网络缓冲池 在第49页和连接和 *CPU* 利用率 在第54页。

当用户过于频繁请求页时，磁盘会变得使用过度或饱和。在以下情况下会发生饱和现象：

- 将磁盘用于多种用途，如用于日志记录和活动的数据库表。
- 同一个磁盘上驻留了完全不同的数据。
- 表扩展数据块相互交错。

应用程序所要求的各种功能以及数据库服务器执行的控制一致性功能可以为应用程序确定磁盘、块和数据库空间的最佳布局。数据库服务器可用的磁盘越多，就越容易平衡这些磁盘上的 I/O。有关这些因素的更多信息，请参阅表性能的注意事项 在第116页。

本节概述块、数据库空间和 BLOB 空间初始配置的重要问题。当决定如何在磁盘上布置块和数据库空间时，请考虑以下问题：

- 关键数据的放置和镜像
- 负载均衡
- 减少争用
- 易于备份与还原

与循环分段存储一起使用，您可平衡磁盘和控制器上的块、节省时间和处理错误。在单个磁盘上放置多个块可提高吞吐量。

将磁盘分区与块相关联

您应该将块分配给整个磁盘分区。当块与磁盘分区（或设备）一致时，就很容易跟踪磁盘空间的使用情况，并且可以避免由误算的偏移所引起的错误。

块的最大值为 4 TB。

将数据库空间与块相关联

您应该将一个块与一个数据库空间相关联，特别是在数据库空间用于表分段时。

有关表放置和布局的更多信息，请参阅表性能的注意事项 在第116页。

将系统目录表与数据库表放置在一起

当包含特定数据库的系统目录的磁盘出现故障时，整个数据库在系统目录还原之前将无法访问。由于可能无法访问，请勿在单个数据库空间中集群所有数据库的系统目录表。而是将系统目录表与其描述的数据库表放置在一起。

要在表数据库空间中创建系统目录表：

1. 在表要驻留的数据库空间中创建数据库。
2. 使用 SQL 语句 DATABASE 或 CONNECT 使该数据库成为当前的数据库。
3. 输入 CREATE TABLE 语句来创建表。

数据库空间块的熟文件的 I/O

在 UNIX™ 上，可以控制用于数据库空间块的熟文件（即已缓冲的文件）的直接 I/O 使用情况。

在 UNIX™ 上，可以用两种方式分配磁盘空间：

- 使用通过操作系统缓冲的文件。这些文件常常称为熟文件（即已缓冲的文件）。
- 使用未缓冲的磁盘存取，也称为原始磁盘空间。

当数据库空间驻留在原始磁盘设备（也称为字符专用设备）上时，数据库服务器将使用未缓冲的磁盘存取。原始磁盘在数据库服务器内存和磁盘间直接传送数据而不会复制数据。

一般应该在 UNIX™ 系统上使用原始磁盘设备以取得更好的性能，可能优先使用熟文件，比原始设备更容易分配和管理。如果使用熟文件，您也许可以通过启用 SinoDB® 直接 I/O 选项来获取更好的性能。

此外，SinoDB® 支持 AIX® 操作系统上单独的并发 I/O 选项。如果在 AIX® 上启用并发 I/O，将同时获得无缓冲的 I/O 和并发 I/O。使用并发 I/O 可以并发写入文件的两个部分。（在其他一些操作系统和文件系统上，启用直接 I/O 也会启用并发 I/O，这是同一文件系统直接 I/O 功能的一部分。）

要确定最佳性能，请对系统上的数据库空间和表布局执行基准测试。

直接 I/O (UNIX™)

在 UNIX™ 上，可以使用直接 I/O 来改善熟文件（即已缓冲的文件）的性能。直接 I/O 会很有益，因为其避免文件系统缓冲。因为直接 I/O 使用无缓冲的 I/O，所以磁盘读写的效率更高（相对于仅访问文件系统缓冲区的那些读写操作）。

直接 I/O 通常要求数据分布在磁盘扇区边界上。

直接 I/O 也允许使用内核异步 I/O (KAIO)，其可以进一步提高性能。通过使用直接 I/O 和 KAIO（如可用）可使用于数据库空间块的熟文件的性能与原始设备的性能接近。

如果您的文件系统支持对数据库空间块所用的页大小进行直接 I/O，那么数据库服务器将进行如下操作：

- 缺省情况下不使用直接 I/O。
- 如果 DIRECT_IO 配置参数设置为 1，使用直接 I/O。
- 缺省情况下，将直接 I/O 和 KAIO（如果文件系统支持）结合使用。
- 如果设置了环境变量 KAIOOFF，那么不将 KAIO 和直接 I/O 结合使用。

如果 SinoDB® 对块使用直接 I/O，而另一程序尝试不使用直接 I/O 来打开块文件，那么此打开操作通常会成功，但是可能会发生性能降级。因为文件系统尝试通过在冲突的打开操作过程中切换到缓冲的 I/O 而不使用直接 I/O，或者通过在每个直接 I/O 操作之前清空文件系统高速缓存并在每个直接写入后使文件系统高速缓存无效来确保每次打开都可以看到相同的文件数据，所以会发生性能降级。

SinoDB® 不会将直接 I/O 用于临时数据库空间。

相关链接

《*SinoDB 管理员参考》: DIRECT_IO 配置参数 (UNIX)*

直接 I/O (Windows™)

直接 I/O 用于 Windows™ 平台的数据库空间块（不管 DIRECT_IO 配置参数的值如何）。

并发 I/O (仅限 AIX®)

在 AIX® 操作系统中，可以将并发 I/O 以及直接 I/O 用于使用熟文件（即已缓冲的文件）的块。并发 I/O 可以改善性能，因为其允许并发进行对某文件的多个读写操作，而没有对非争用读写操作进行通常的串行化。

您具有在多个磁盘放置的单个块文件中的数据时，并发 I/O 会特别有利。

并发 I/O（可以通过将 DIRECT_IO 配置参数设置为 2 来启用）包括避免文件系统缓冲的益处，且在使用直接 I/O 而无并发 I/O 时会受到相同的限制和 KAIO 的使用。因此，启用并发 I/O 时，您将同时获取无缓冲的 I/O 和并发 I/O。

如果 SinoDB® 将并发 I/O 用于块，而另一程序（例如外部备份程序）尝试不使用并发 I/O 就打开同一块文件，那么打开操作将失败。

SinoDB® 没有将直接或并发 I/O 用于熟文件（这些文件用于临时数据库空间块）。

相关链接

《*SinoDB 管理员参考》: DIRECT_IO 配置参数 (UNIX)*

启用直接 I/O 或并发 I/O 选项 (UNIX™)

使用 DIRECT_IO 配置参数来在 UNIX™ 上启用直接 I/O 选项，或者在 AIX® 上启用并发 I/O 选项。

先决条件：

- 必须作为用户 root 或 informix 登录。
- 直接 I/O 或并发 I/O 必须可用且文件系统必须支持用于数据库空间块的页大小的直接 I/O。

要启用直接 I/O，请将 DIRECT_IO 配置参数设置为 1。

要在 AIX® 操作系统上启用并发 I/O 以及直接 I/O，请将 DIRECT_IO 配置参数设置为 2。

如果不希望启用直接 I/O 或并发 I/O，请将 DIRECT_IO 配置参数设置为 0。

相关链接

《*SinoDB 管理员参考》: DIRECT_IO 配置参数 (UNIX)*

确认使用直接或并发 I/O 选项 (UNIX™)

可以确认和监视对熟文件块使用直接 I/O 或并发 I/O（在 AIX® 上）。

可以通过以下方式确认对直接 I/O 或并发 I/O 的使用：

- 显示 onstat -d 信息。

  onstat -d 命令显示包含标志的信息，此标志标识对熟文件块使用了直接 I/O，还是并发 I/O（在 AIX® 上）或者两者均不使用。
- 验证 DIRECT_IO 配置参数设置为 1（对于直接 I/O）还是 2（对于并发 I/O）。

相关链接

《*SinoDB 管理员参考》: DIRECT_IO 配置参数 (UNIX)*

《*SinoDB* 管理员参考》*: onstat -d* 命令*:* 显示块信息

关键数据的放置

包含系统保留页、物理日志以及数据库空间（包含逻辑日志文件）的一个或多个磁盘，对于数据库服务器的运行来说至关重要。如果这些元素中的任何一个不可用，数据库服务器将无法运行。缺省情况下，数据库服务器将所有这三个关键元素置于根数据库空间中。

要为关键数据获得适当的放置策略，必须在数据可用性和最大的日志记录性能之间进行权衡。

缺省情况下，数据库服务器也会将临时表和排序文件置于根数据库空间中。应该使用 DBSPACETEMP 配置参数和 DBSPACETEMP 环境变量将这些表和文件分配到其他数据库空间。有关详细信息，请参阅为临时表和排序文件配置数据库空间 在第90页。

考虑对关键数据组件使用单独磁盘

如果将根数据库空间、逻辑日志和物理日志放在不同磁盘上的不同数据库空间中，您可以获得一些明显的性能优势。用于每个关键数据组件的磁盘应该位于不同的控制器上。

此方法具有以下优点：

- 将日志记录活动与数据库 I/O 隔离开，并使物理日志 I/O 请求与逻辑日志 I/O 请求能同时获得服务
- 减少从故障中恢复所需的时间

  但是，除非磁盘被镜像，否则可能会有这样的风险，即当磁盘出现故障时，包含关键数据的磁盘可能会受到影响，这将导致数据库服务器终止运行，并需要从 0 级备份中完全还原所有的数据。
- 允许使用相对较小的根数据库空间，其中只包含保留页、数据库分区以及 sysmaster 数据库

  在许多情况下，10,000 KB 就足够了。

数据库服务器使用不同的方法配置关键数据的不同部分。要为根数据库空间和物理日志分配适当的数据库空间，请设置适当的数据库服务器配置参数。要将逻辑日志文件分配到适当的数据库空间，请使用 onparams 实用程序。

有关影响关键数据每个部分的配置参数的更多信息，请参阅影响关键数据的配置参数 在第90页。

考虑对关键数据组件使用镜像

考虑对包含关键数据的数据库空间使用镜像。镜像这些数据库空间可以确保数据库服务器在单个磁盘遇到故障时可以继续运行。

但是，根据给定数据库空间的 I/O 请求的混合情况，需要在镜像的容错和 I/O 性能之间进行权衡。对使用模式为读取密集的数据库空间进行镜像时，性能优势极为明显；对写入密集的数据库空间进行镜像时，性能会稍微有些降低。

大多数的现代存储设备都具有优越的镜像能力，您可以使用这些设备来代替数据库服务器的镜像能力。

当镜像生效时，可以使用两个磁盘来处理读取请求，数据库服务器能处理数量更大的此类请求。但是，每个写入请求要执行两次物理写入操作，并要在执行完这两个物理操作之后才完成写入。写入操作是并行执行的，但要在两个磁盘中较慢的那个执行更新操作之后，请求才完成。因此，当您镜像写入密集的数据库空间时，会感觉到性能稍微有些降低。

考虑对根数据库空间制作镜像

如果您镜像根数据库空间，并将其内容限制为只读或很少访问的表时，可以最小的性能代价获得某种程度的容错功能。

如果您将更新密集的表置于其他非镜像数据库空间中，那么在出现磁盘故障时，可以使用数据库服务器的备份与还原实用程序对那些表执行热还原。对根数据库空间进行镜像之后，数据库服务器在修复出现故障的磁盘时可以保持联机并为其他事务提供服务。

镜像根数据库空间时，务必将第一个块放在与镜像不同的设备上。MIRRORPATH 配置参数应与 ROOTPATH 具有不同的值。

相关链接

《*SinoDB* 管理员参考》: *MIRRORPATH* 配置参数

《*SinoDB* 管理员参考》: *ROOTPATH* 配置参数

考虑对智能大对象块制作镜像

如果对包含元数据页的块制作镜像，那么可以实现较高可用性和更快存取。

智能大对象空间是由存储智能大对象的一个或多个块组成的一个逻辑存储单元，其中这些智能大对象由 CLOB（字符大对象）或 BLOB（二进制大对象）数据组成。

智能大对象空间的第一个块包含一组特殊的页（称为元数据），元数据可用于在智能大对象空间中定位智能大对象。如果创建块时在 onspaces 命令中指定了附加块，那么添加到智能大对象空间的附加块也可以具有元数据页。

由于以下原因，请考虑镜像包含元数据页的块：

- 更高的可用性

  不访问元数据页，用户就无法访问智能大对象空间中的任何智能大对象。如果智能大对象空间的第一个块包含所有的元数据页，并且包含该块的磁盘不可用，那么您将无法访问智能大对象空间中的智能大对象，即使它驻留在另一个磁盘的块上。要实现高可用性，请至少镜像智能大对象空间的第一个块以及包含元数据页的任何其他块。
- 更快的访问速度

  通过镜像包含元数据页的块，您可以将读取活动分散在包含主块和镜像块的磁盘上。

相关链接

《*SinoDB* 管理员指南》: 智能大对象空间

镜像制作及其对逻辑日志的影响

逻辑日志是写入密集型的。如果对包含逻辑日志文件的数据库空间制作镜像，那么您会遇到双写入性能的轻微降级。但是，可以通过选择适当的逻辑缓冲区大小和日志记录方式将日志记录生成 I/O 请求的速率调整到特定范围。

有关轻微的双写入性能降级的详细信息，请参阅考虑对关键数据组件使用镜像 在第88页。

使用未缓冲且符合 ANSI 标准的日志记录时，对每个提交的事务，数据库服务器都会请求将日志缓冲区清空到磁盘（如果镜像了数据库空间，那么将清空到两个磁盘）。缓冲日志记录生成的 I/O 请求远少于无缓冲或符合 ANSI 标准的日志记录生成的 I/O 请求。

用无缓冲日志记录，日志缓冲区只有在被填充并且所包含的全部事务都已完成后才写入磁盘中。如果增加逻辑日志缓冲区的大小，还可以进一步降低逻辑日志 I/O 的频率。但是，缓冲日志记录将事务保存在任何部分填充的缓冲区中，这种缓冲区在系统出现故障时很容易丢失数据。

虽然缓冲日志记录可以保证数据库的一致性，但在出现故障时无法保证特定的事务。逻辑日志缓冲区越大，在出现故障后还原服务时需要重新执行的事务也就越多。

与物理日志不同，您不能在初始数据库服务器配置中，为逻辑日志文件指定备用的数据库空间。反之，先使用 onparams 实用程序将逻辑日志文件添加到备用数据库空间，然后从根数据库空间删除逻辑日志文件。

相关链接

《*SinoDB* 管理员参考》: *onparams* 实用程序

镜像制作及其对物理日志的影响

物理日志是写入密集型的，其活动在检查点以及清空缓冲区的数据页时发生。当激活了页清除程序线程时，也会发生物理日志的 I/O。如果对包含物理日志的数据库空间制作镜像，那么您会遇到双写入性能的轻微降级。

有关轻微的双写入性能降级的详细信息，请参阅考虑对关键数据组件使用镜像 在第88页。

要将物理日志的 I/O 保持在最低，可以调整检查点时间间隔以及 LRU 的最小和最大阈值。（请参阅 *CKPTINTVL* 及其对检查点的影响 在第104页和 *BUFFERPOOL* 及其对页清除的影响 在第111页）。

影响关键数据的配置参数

可配置根数据库空间以及逻辑和物理日志的配置参数会影响关键数据。

可以使用以下配置参数配置根数据库空间：

- ROOTNAME
- ROOTOFFSET
- ROOTPATH
- ROOTSIZE
- MIRROR
- MIRRORPATH
- MIRROROFFSET

这些参数确定根数据库空间初始块的位置和大小并为该块配置镜像（如果有的话）。（如果镜像了初始块，那么根数据库空间中的其他块也必须镜像）。否则，这些参数不会对性能产生重大影响。

以下配置参数会影响逻辑日志：

- LOGSIZE
- LOGBUFF

LOGSIZE 配置参数会确定每个逻辑日志文件的大小。LOGBUFF 配置参数会确定共享内存中三个逻辑日志缓冲区的大小。

PHYSFILE 配置参数会确定根数据库空间中物理日志的初始大小。只有在创建实例时才会使用该配置参数。

相关链接

LOGBUFF 配置参数和内存利用率 在第65页
检查点和物理日志 在第105页

为临时表和排序文件配置数据库空间

使用临时表或大型排序操作的应用程序需要大量的临时空间。要提高这些应用程序的性能，请使用 DBSPACETEMP 配置参数或 DBSPACETEMP 环境变量以将一个或多个数据库空间指定给临时表和排序文件。

根据创建临时空间的方式，在不设置 DBSPACETEMP 时，数据库服务器对临时表和排序文件使用以下缺省位置：

- 当您使用 CREATE TABLE 语句的 TEMP TABLE 子句创建显式临时表且没有在 IN dbspace 或 FRAGMENT BY 子句中为表指定数据库空间时，缺省位置为当前数据库的数据库空间

  此操作会严重影响该数据库空间的 I/O。如果镜像了根数据库空间，临时表或排序文件的 I/O 的双写入性能会稍微降低。
- 当您使用 SEKLECT 语句的 INTO TEMP 选项创建显式临时表时，缺省位置为根数据库空间

  此操作会严重影响根数据库空间的 I/O。如果镜像了根数据库空间，临时表或排序文件的 I/O 的双写入性能会稍微降低。
- 使用以下其中一个变量指定的操作系统目录或文件：
  - 在 UNIX™ 中，PSORT_DBTEMP 环境变量指定的操作系统目录（如果已设置该变量）

    如果没有设置 PSORT_DBTEMP，那么数据库服务器将排序文件写入 /tmp 目录中的操作系统文件空间。
  - 在 Windows™ 中，在 Control Panel > System 上用户环境变量窗口的 TEMP 或 TMP 中指定的目录。

  数据库服务器使用操作系统目录或文件来定向以下数据库操作所产生的任何溢出：
  - 带 GROUP BY 子句的 SELECT 语句
  - 带 ORDER BY 子句的 SELECT 语句
  - 散列连接操作

- 嵌套循环连接操作
- 索引构建

警告： 如果没有为 DBSPACETEMP 配置参数或 DBSPACETEMP 环境变量指定值，那么数据库服务器将此操作系统文件用于隐式临时表。如果此文件系统的空间不足，无法保存排序文件，那么执行排序的查询将返回错误。同时，在移除排序文件之前，操作系统可能会受到严重影响。

可通过为存储临时表和排序文件而专门创建的临时数据库空间来改善性能。使用 DBSPACETEMP 配置参数和 DBSPACETEMP 环境变量可以将这些表和文件分配到临时数据库空间。

当在 DBSPACETEMP 配置参数或 DBSPACETEMP 环境变量中指定数据库空间时，可以获得以下性能优势：

- 减少对根数据库空间、生产数据库空间或操作系统文件的 I/O 影响
- 当为临时表指定多个数据库空间且 PDQ 优先级设置为大于 0 时，将在临时文件中使用并行排序（以处理 ORDER BY 或 GROUP BY 之类的查询子句，或在执行 CREAT INDEX 时对索引键进行排序）。
- 将两个或更多的数据库空间分配到不同的磁盘上时，可以提高数据库服务器创建临时表的速度
- 执行 SELECT....INTO TEMP 语句时自动跨数据库空间对临时表进行分段

下表显示用于创建临时表的语句以及有关创建临时表的位置的信息。

创建临时表的语句	记录数据库	WITH NO LOG 子句	FRAGMENT BY 子句	创建临时表的位置
CREATE TEMP TABLE	是	否	否	根数据库空间
CREATE TEMP TABLE	是	是	否	在 DBSPACETEMP 中指定的数据库空间之一
CREATE TEMP TABLE	是	否	是	无法创建临时表。错误 229/196
SELECT ..INTO TEMP	是	是	否	按循环分段（仅限在 DBSPACETEMP 中指定的不记入日志的数据库空间中）

重要： 请使用 DBSPACETEMP 配置参数或 DBSPACETEMP 环境变量来提高排序操作的性能，并防止数据库服务器意外填充文件系统。列出的数据库空间必须由作为未缓冲设备来分配的块组成。

相关链接

《SinoDB 管理员参考》: DBSPACETEMP 配置参数

在 *DBSPACETEMP* 配置参数中指定临时表 在第92页

《SinoDB SQL 指南: 语法》: CREATE TEMP TABLE 语句

《SinoDB SQL 指南: 语法》: INTO TEMP 子句

创建临时数据库空间

可以创建专门用于临时表和排序文件的数据库空间。数据库服务器对临时数据库空间不执行逻辑或物理日志记录，并且临时数据库空间从不作为整个系统备份的一部分进行备份。

要创建专门用于临时表和排序文件的数据库空间，可使用 onspaces -t。为了获取最佳性能，请遵循以下准则：

- 如果创建多个临时数据库空间，那么在不同的磁盘上创建每个数据库空间以平衡 I/O 影响。
- 不要在单个磁盘上放置多个临时数据库空间。

不能镜像用 onspaces -t 创建的临时数据库空间。

重要： 如果数据库含有日志记录，那么必须将 WITH NO LOG 子句包含到 SELECT... INTO TEMP 语句中才能将显式临时表放入 DBSPACETEMP 配置参数和 DBSPACETEMP 环境变量中列出的数据库空间。否则，数据库服务器将显式临时表存储到根数据库空间中。

相关链接

《*SinoDB* 管理员参考》*: DBSPACETEMP* 配置参数

《*SinoDB* 管理员参考》*: create tempdbspace* 参数*:* 创建临时数据库空间 *(SQL* 管理 *API)*

《*SinoDB* 管理员参考》*: onspaces -c -d:* 创建数据库空间

在 DBSPACETEMP 配置参数中指定临时表

DBSPACETEMP 配置参数指定一个数据库空间列表，缺省情况下数据库服务器将临时表和排序文件放在这些数据库空间中。该配置参数中列出的数据库空间可能部分或全部是临时数据库空间，它们是专门为存储临时表和排序文件而保留的。

如果数据库服务器通过 SELECT INTO TEMP 操作（该操作会创建 TEMP 表）将数据插入一个临时表中，那么该临时表会使用循环分布式存储。其分段会在 DBSPACETEMP 配置参数或 DBSPACETEMP 环境变量中列出的临时数据库空间中创建。例如，以下查询使用循环分布式存储：

```
SELECT col1 FROM tab1
   INTO TEMP temptab1 WITH NO LOG;
```

DBSPACETEMP 配置参数使数据库管理员可以限制数据库服务器用于临时存储的数据库空间。

> 重要： DBSPACETEMP 配置参数没有在 onconfig.std 文件中设置。要实现临时表和排序文件的最佳性能，请使用 DBSPACETEMP 指定两个或更多位于不同磁盘上的数据库空间。

提示：

- 如果在磁盘数有限的小型系统上工作，并且无法将临时数据库空间置于不同的磁盘驱动器上，那么可以考虑使用 1（或可能为 2）个临时数据库空间。这可能会减少与临时数据库空间关联的日志记录。
- 如果有许多磁盘驱动器，那么可以并行执行许多操作（例如排序、连接和临时表），而无需多个临时数据库空间。您拥有的临时数据库空间数与您希望发散 I/O 的程度相关。一开始采用 4 个临时数据库空间比较合适。如果创建过多小型临时数据库空间，那么将没有足够空间来执行大型对象的非并行创建。

相关链接

为临时表和排序文件配置数据库空间 在第90页

《*SinoDB* 管理员参考》*: DBSPACETEMP* 配置参数

分布方案 在第191页

《*SinoDB SQL* 指南*:* 语法》*: CREATE TEMP TABLE* 语句

覆盖会话的 DBSPACETEMP 配置参数

要覆盖 DBSPACETEMP 配置参数，可以针对临时表和排序文件使用 DBSPACETEMP 环境变量。此环境变量指定了一个使用逗号和冒号分隔的数据库空间列表，当前会话的临时表就放在这些数据库空间中。

> 重要： 请使用 DBSPACETEMP 配置参数或 DBSPACETEMP 环境变量来提高排序操作的性能，并防止数据库服务器意外填充文件系统。

您应该使用 DBSPACETEMP 而不是 PSORT_DBTEMP 环境变量来指定排序文件，原因如下：

- DBSPACETEMP 通常会产生更好的性能。

  当数据库空间驻留在字符专用设备（也称为原始磁盘设备）上时，数据库服务器使用未缓冲的磁盘存取。未缓冲的 I/O 比常规（已缓冲的）操作系统文件的更快，因为数据库服务器直接管理 I/O 操作。
- PSORT_DBTEMP 指定放置排序文件的一个或多个操作系统目录。

  这些操作系统文件可能意外填充计算机，因为数据库服务器并不管理这些文件。

估算用于数据库空间和散列连接的临时空间

可以估算并增加用于数据库空间和散列连接的临时空间量。这样可以防止内存溢出到磁盘上的临时空间。

可以使用以下原则估算要分配的临时空间量：

- 对于 OLTP 应用程序，应分配至少相当于该表的 10% 的临时数据库空间。
- 对于 DSS 应用程序，应分配至少相当于该表的 50% 的临时数据库空间。

散列连接（其工作方式是从连接中一个表的行构造散列表，然后在其他表的行探测该散列表）可使用大量内存并有可能导致磁盘上临时空间的溢出。散列表大小由用来构造该散列表的表大小（通常为连接中两个表中的较小者）决定，在应用过滤器之后，可减少行的数量并且可能减少列的数量。

将散列连接分区组织到页中。每页都有一个头。64 位平台上数据库中的头部信息和元组比 32 平台上构件中的要大。每个页的大小为基本页大小（根据系统的不同为 2K 或 4K），除非单个行需要更多的空间。如果需要更多空间，那么可以向行的长度添加字节。

可以使用以下公式估算在散列连接中散列表需要的内存量：

```
hash_table_size = (32 bytes + row_size_smalltab) * num_rows_smalltab
```

其中，row_size_smalltab 和 num_rows_smalltab 分别指散列连接的两个表中较小者的行大小和行数。

例如，假设您拥有一个长度为 80 字节的页头以及一个长度为 48 字节的行头。由于必须将每行调整为 8 字节，因此您可能需要将行长度增加 7 个字节，如以下公式所示：

```
per_row_size = 48 bytes + rowsize + mod(rowsize, 8)
page_size = base_page_size (2K or 4K)
rows_per_page = round_down_to_integer((page_size - 80 bytes) / per_row_size)
```

如果 rows_per_page 值小于 1，那么将 page_size 值增加至 base_page_size 的最小倍数，如以下公式所示：

```
size = (numrows_smalltab / rows_per_page) * page_size
```

可使用 DS_NONPDQ_QUERY_MEM 配置参数来为所有查询（除 PDQ 查询外）配置排序内存。但是如果 PDQ 优先级设置大于 0，那么该设置没有任何作用。

有关更多信息，请参阅散列连接 在第211页和为使用散列连接、聚合和其他内存密集型元素的查询分配更多内存 在第290页。

相关链接

《*SinoDB* 管理员参考》: *DS_NONPDQ_QUERY_MEM* 配置参数

PSORT_NPROCS 环境变量

PSORT_NPROCS 环境变量指定数据库服务器可用于排序查询的最大线程数。当查询包含一个大的排序操作时，可以并行执行多个排序线程来改善查询的性能。

当 PDQ 优先级值为 0 并且 PSORT_NPROCS 大于 1 时，数据库服务器将使用并行排序。PDQ 的管理不会限制这些排序。 换言之，虽然以并行方式执行排序，但数据库服务器不会将排序视为 PDQ 活动。当 PDQ 优先级为 0 时，数据库服务器不会通过任何 PDQ 配置参数控制排序。

当 PDQ 优先级大于 0 且 PSORT_NPROCS 大于 1 时，查询将从并行排序和 PDQ 功能（例如，并行扫描和额外内存）两方面获益。用户可以使用 PDQPRIORITY 环境变量为查询请求 PDQ 资源的特定部分。您可以使用 MAX_PDQPRIORITY 参数来限制此类用户请求的数量。有关 MAX_PDQPRIORITY 的更多信息，请参阅限制查询中的 *PDQ* 资源 在第45页。

数据库服务器为排序分配相对较少的内存，且该内存在 PSORT_NPROCS 排序线程之间分配。没有足够的内存可供分配时，排序进程会使用磁盘上的临时空间。有关为排序分配内存的更多信息，请参阅估算排序所需的内存 在第165页。

> 重要： 为了提高排序操作的性能，如果计算机具有多个 CPU，请将 PSORT_NPROCS 的初始值设置为 2。如果后续的 CPU 活动低于 I/O 活动，可以增加 PSORT_NPROCS 的值。

有关在索引构建期间进行排序的更多信息，请参阅提高索引构建的性能 在第164页。

为临时智能大对象配置智能大对象空间

应用程序可以将临时智能大对象用于仅在用户会话生命期内需要的文本、图像或其他用户定义数据类型。这些应用程序不要求日志记录临时大对象。日志记录将 I/O 活动添加到逻辑日志并增加内存使用率。

可以将临时智能大对象存储到永久智能大对象空间或临时智能大对象空间中。

- 永久智能大对象空间

  如果您在常规智能大对象空间中存储临时智能大对象，并保留无日志记录的缺省属性，那么对象的更改不会被记录，但元数据始终会被记录。
- 临时智能大对象空间

  更新存储在临时智能大对象空间中的临时智能大对象的应用程序会快得多，因为数据库服务器不记录临时智能大对象空间中的元数据或用户数据。

要提高更新临时智能大对象的应用程序性能，请在 mi_lo_specset_flags 或 ifx_lo_specset_flags API 函数中指定 LOTEMP 标志，并为临时智能大对象指定临时智能大对象空间。数据库服务器使用以下位置优先顺序放置临时智能大对象：

- 创建智能大对象时在 mi_lo_specset_sbspace 或 ifx_lo_specset_sbspace API 函数中指定的智能大对象空间

  在 API 函数中指定临时智能大对象空间，这样将不记录对象和元数据的更改。在 API 函数中指定的智能大对象空间将覆盖 SBSPACETEMP 或 SBSPACENAME 配置参数可能指定的任何缺省智能大对象空间。
- 使用 CREATE TABLE 语句的 TEMP TABLE 子句创建显式临时表时在 IN Sbspace 子句中指定的智能大对象空间

  在 IN Sbspace 子句中指定临时智能大对象空间，这样将不记录对象和元数据的更改。
- 在 SBSPACENAME 配置参数中指定的永久智能大对象空间（如果不在 SBSPACETEMP 配置参数中指定智能大对象空间）

如果没有使用以上任何一种方法指定临时智能大对象空间，那么在您尝试创建临时智能大对象时，数据库服务器将发出以下错误消息：

```
-12053  Smart Large Objects: No sbspace number specified.
```

创建临时智能大对象空间

要创建一个智能大对象空间专门用于临时智能大对象，请使用带有 -t 选项的 onspaces -c -S。

为了获取最佳性能，请遵循以下准则：

- 如果创建多个临时智能大对象空间，那么在不同的磁盘上创建每个智能大对象空间以平衡 I/O 影响。
- 不要在单个磁盘上放置多个临时智能大对象空间。

数据库服务器对临时智能大对象空间不执行逻辑或物理日志记录，并且临时智能大对象空间从不作为整个系统备份的一部分进行备份。不能镜像用 onspaces -t 创建的临时智能大对象空间。

> 重要： 如果数据库含有日志记录，那么必须将 WITH NO LOG 子句包含到 SELECT... INTO TEMP 语句中才能将临时智能大对象放入 SBSPACETEMP 配置参数中列出的数据库空间。否则，数据库服务器会将临时智能大对象存储到 SBSPACENAME 配置参数中列出的智能大对象空间中。

相关链接

《SinoDB 管理员参考》: onspaces -c -S: 创建智能大对象空间

《SinoDB 管理员指南》: 创建临时智能大对象空间

指定用于临时存储的数据库空间

SBSPACETEMP 配置参数指定一个智能大对象空间列表，缺省情况下数据库服务器将临时智能大对象放置在这些智能大对象空间中。此配置参数中列出的智能大对象空间可能部分或全部是临时智能大对象空间，它们是专门为存储临时智能大对象而保留的。

> 重要： SBSPACETEMP 配置参数没有在 onconfig.std 文件中设置。为了实现临时智能大对象的最佳性能，请在不同磁盘上使用 SBSPACETEMP 指定两个或更多的智能大对象空间。

相关链接

《SinoDB 管理员参考》: SBSPACETEMP 配置参数

简单大对象的放置

可以将简单大对象存储在与表所驻留的同一个数据库空间中或 BLOB 空间中。

BLOB 空间是由仅存储简单大对象（TEXT 或 BYTE 数据）的一个或多个块组成的逻辑存储单元。有关存储智能大对象（如 BLOB、CLOB 或多重表示数据）的智能大对象空间的更多信息，请参阅影响智能大对象 *I/O* 的因素 在第98页。

如果使用 BLOB 空间，可以在与表（该表和数据相关联）不同的磁盘上存储简单大对象。可以在同一个 BLOB 空间中存储与不同表相关联的简单大对象。

您可以使用 onspaces 实用程序或 SQL 管理 API 命令（即使用带有 admin() 或 task() 函数的 create blobspace 参数）来创建 BLOB 空间。

使用 CREATE TABLE 语句创建与简单大对象相关联的表时，将简单大对象分配给 BLOB 空间。

简单大对象不进行记录也不通过缓冲池。但是，检查点的频率会影响访问 TEXT 或 BYTE 数据的应用程序。有关更多信息，请参阅 *LOGSIZE* 和 *LOGFILES* 及其对检查点的影响 在第105页。

相关链接

《SinoDB SQL 指南: 语法》: CREATE TABLE 语句

《SinoDB 管理员参考》: create blobspace 参数: 创建 BLOB 空间 (SQL 管理 API)

《SinoDB 管理员参考》: onspaces -c -b: 创建 BLOB 空间

BLOB 空间相对于数据库空间的优势

如果将简单大对象与相关联的表分开存储在单独磁盘上的 BLOB 空间中，而不是将对象存储在数据库空间中，那么您可以获得一些性能优势。

将简单大对象存储在 BLOB 空间中的性能优势为：

- 您对表和简单大对象有并行访问权。
- 与存储在数据库空间中的简单大对象不同，BLOB 空间数据将直接写入磁盘。简单大对象不通过常驻共享内存，从而使内存页可留作他用。
- 不记录简单大对象，这会减少已记录数据库的日志记录 I/O 活动。

有关更多信息，请参阅将简单大对象存储在表空间或单独的 *BLOB* 空间中 在第122页。

BLOB 页大小注意事项

BLOB 空间分成称为 *BLOB* 页的单元。数据库服务器从 BLOB 空间中检索以 BLOB 页大小为单位的简单大对象。创建 BLOB 空间时，您可以按磁盘页的倍数指定 BLOB 页的大小。

配置的最佳 BLOB 页大小取决于以下因素：

- 简单大对象的大小分配
- 最大的简单大对象的检索速度与将简单大对象存储到大的 BLOB 页中时所浪费的磁盘空间量之间的权衡

要尽快检索简单大对象，请使用最大的简单大对象的大小（向上取整到最接近的磁盘页大小增量）。此方案确保数据库服务器即使在单个 I/O 请求中也能检索最大的简单大对象。 虽然此方案保证了最快的检索，但存在浪费磁盘空间的可能性。因为简单大对象存储在各自的 BLOB 页（或 BLOB 页组）中，所以即使简单大

对象只占该页的一部分，数据库服务器也会为每个 BLOB 页保留同样大小的磁盘空间。使用较小的 BLOB 页可以更好地利用磁盘，特别是当简单大对象的大小存在很大差别时。

要实现理论上磁盘空间的最大利用率，您可以使 BLOB 页的大小与标准磁盘页相同。这样，许多（如果不是绝大部分）简单大对象将需要多个 BLOB 页。因为数据库服务器为每个 BLOB 页都获取锁并发出不同的 I/O 请求，所以此方案的执行效果很差。

实际上，缩放的折衷方案使用最常见的简单大对象的大小作为 BLOB 页的大小。例如，假设表中有 160 个简单大对象值，其大小分布如下：

- 其中 120 个值都是每个 12 KB。
- 其他 40 个值是每个 16 KB。

可以选择以下其中一种 BLOB 页大小：

- 12 KB 的 BLOB 页比 16 KB 的 BLOB 页存储效率高，如以下两种计算所示：
  - 12 KB

    此配置使大多数简单大对象值需要单个 BLOB 页而其他 40 个值需要两个 BLOB 页。在此配置中，每个较大值的第二个 BLOB 页中都浪费了 8 KB。总的浪费空间如下：

    ```
    wasted-space = 8 kilobtyes * 40
                 = 329 kilobytes
    ```

  - 16 KB

    在此配置中，在 120 个简单大对象的数据块中浪费了 4 KB。总的浪费空间如下：

    ```
    wasted-space = 4 kilobtyes * 120
                 = 640 kilobytes
    ```
- 如果您的应用程序对 16 KB 简单大对象值的访问比较频繁，那么数据库服务器必须为每个 BLOB 页执行单独的 I/O 操作。在这种情况下，16 KB 的 BLOB 页比 12 KB 的 BLOB 页的检索速度更快。

BLOB 空间可包含的最大页数为 2147483647。因此，BLOB 空间的大小限于 BLOB 页大小 x 2147483647。这包括组成 BLOB 空间的所有块中的 BLOB 页。

> 提示： 如果表具有多个简单大对象列且数值的大小并不接近，那么将数据存储到不同的 BLOB 空间中，每个 BLOB 空间都有相应大小的 BLOB 页。

优化 BLOB 空间 BLOB 页大小
在评估 BLOB 空间存储策略时，可以按两个标准评估效率：每个简单大对象所需的 BLOB 页填充度和 BLOB 页数。

BLOB 页填充度指的是每个 BLOB 页中的数据量。TEXT 和 BYTE 数据存储在不能共享 BLOB 页的 BLOB 空间中。因此，如果单个简单大对象仅需要 20% 的 BLOB 页，那么该页剩余的 80% 就不可用。

但是，应避免 BLOB 页太小。当存储每个简单大对象需要数个 BLOB 页时，会增加存储的开销费用。例如，由于每个 BLOB 页必须获取一个锁，因此需要更多的锁用于更新。

获取 BLOB 空间存储统计信息
要帮助确定每个 BLOB 空间的最佳 BLOB 页大小，请使用 oncheck -pB 命令。

该命令为每个表（或数据库）列出以下统计信息：

- 每个 BLOB 空间中表（或数据库）使用的 BLOB 页数
- 作为表（或数据库）的一部分存储的每个简单大对象使用的 BLOB 页的平均填充度

使用 oncheck -pB 输出确定 BLOB 页填充度
oncheck -pB 命令显示描述 BLOB 页平均填充度的统计信息。这些统计信息为数据库或表中单个简单大对象提供了存储效率的度量。

如果您发现大量简单大对象的统计信息中显示的填充度百分比很低，那么更改 BLOB 空间中的 BLOB 页的大小会提高数据库服务器的性能。

oncheck -pB 和 onstat -d update 命令都显示有关可用 BLOB 页数的相同信息。onstat -d update 命令显示与 onstat -d 相同的信息以及每个 BLOB 空间块的准确可用 BLOB 页数。

用数据库名或表名作为参数执行 oncheck -pB。以下示例检索存储在 stores_demo 数据库的 sriram.catalog 表中的所有简单大对象的存储信息：

```
oncheck -pB stores_demo:sriram.catalog
```

oncheck -pB 输出

图 16: oncheck -pB 的输出 在第97页显示此命令的输出。

```
             BLOBSpace Report for stores_demo:sriram.catalog

Total pages used by table           7

BLOBSpace usage:
Space   Page             Percent Full
Name    Number    Pages  0-25%  26-50%  51-75  76-100%
------------------------------------------------------------
blobPIC 0x300080  1      x
   blobPIC 0x300082  2      x
         ------
Page Size is 6144      3

bspc1   0x2000b2  2             x
bspc1   0x2000b6  2                    x
             ------
Page Size is 2048      4
```

图 16: oncheck -pB 的输出

Space Name 是 BLOB 空间的名称，该空间包含一个或多个作为表（或数据库）存储的简单大对象。

Page Number 是 BLOB 空间中特定简单大对象的起始地址。

Pages 是存储此简单大对象所需的数据库服务器页数。

Percent Full 是按 BLOB 空间列出的，此表或数据库中每个 BLOB 空间的平均 BLOB 页填充度的度量。

Page Size 是此 BLOB 空间的 BLOB 页大小（以字节为单位）。BLOB 页大小始终是数据库服务器页大小的倍数。

该示例输出指示四个简单大对象作为表 sriram.catalog 的一部分存储。两个对象存储在 BLOB 空间 blobPIC 的大小为 6144 字节的 BLOB 页中。另两个对象存储在 BLOB 空间 bspc1 中大小为 2048 字节的 BLOB 页中。

该输出顶部出现的摘要信息 Total pages used by table 是存储简单大对象所需的 BLOB 页的简单总和。总和不能说明所使用的 BLOB 页的大小、所存储的简单大对象数或所存储的总字节数。

Percent Full 标题下显示的效率信息是不精确的，但能够提醒管理员注意 TEXT 和 BYTE 数据的存储趋势。

解释 BLOB 页平均填充度
您可以分析 oncheck -pB 命令的输出来计算平均填充度。

使用 *oncheck -pB* 输出确定 *BLOB* 页填充度 在第96页中列出的第一个简单大对象存储在 BLOB 空间 blobPIC 中，并需要一个 6144 字节的 BLOB 页。BLOB 页已填充 51% 到 75 %，这意味着大小在 0.51 * 6144 = 3133 字节和 0.75 * 6144 = 4608 字节之间。此简单大对象大小的最大值必须小于或等于 6144 字节的 75%，即 4608 字节。

BLOB 空间 blobPIC 下列出的第二个对象需要两个 6144 字节的 BLOB 页用于存储，即总数为 12,288 字节。所有已分配的 BLOB 页的平均填充度为 51% 到 75%。 因此，对象大小的最小值必须大于 12,288 字

节的 50%，即 6144 字节。该简单大对象大小的最大值必须小于或等于 12,288 字节的 75%，即 9216 字节。平均填充度并不意味着每个页的填充度都是 51% 到 75%。如果有两个 BLOB 页，其中第一个 BLOB 页填充度为 100%，而第二个 BLOB 页的填充度为 2% 到 50%，那么这两个 BLOB 页的平均填充度的计算结果为 51% 到 75%。

现在考虑 BLOB 空间 bspc1 中的两个简单大对象。这两个对象的大小几乎相同。两个对象都要求 2048 字节的 BLOB 页，且每个对象的平均填充度都是 76% 到 100%。这些简单大对象大小的最小值必须大于已分配的 BLOB 页的 75%，即 3072 字节。每个对象大小的最大值都稍小于 4096 字节（在开销的允许范围内）。

使用 oncheck -pB 输出分析效率标准
您可以分析 oncheck -pB 命令的输出来确定是否存在效率更高的存储策略。

通过查看 图 *16: oncheck -pB* 的输出 在第97页中为 BLOB 空间 bspc1 显示的效率信息，数据库服务器管理员可能断定更好的 TEXT 和 BYTE 数据存储策略是将 BLOB 页大小从 2048 字节加倍到 4096 字节。（BLOB 页大小始终是数据库服务器页大小的倍数。）如果数据库服务器管理员做出此更改，那么页填充度的度量将保持不变，而更新简单大对象时需要锁的数量将减少一半。

BLOB 空间 blobPIC 的效率信息显示没有明显的改进建议。blobPIC 中的两个简单大对象的大小有相当大的差异，而且并没有最佳的存储策略。通常，大小相近的简单大对象的存储效率要高于大小不同的简单大对象。

影响智能大对象 I/O 的因素

智能大对象空间是一个逻辑存储单元，它由一个或多个可存储智能大对象（如 BLOB、CLOB 或多重表示数据）的块组成。智能大对象空间的磁盘布局、某些配置参数的设置以及某些 onspaces 实用程序选项会影响智能大对象的 I/O。

DataBlade® API 和 SinoDB® ESQL/C 应用程序编程接口也提供影响智能大对象 I/O 操作的函数。

重要： 对于大多数应用程序，您应该使用数据库服务器对磁盘存储信息计算的值。

相关链接

《SinoDB 管理员指南》: 智能大对象空间
《SinoDB ESQL/C 程序员指南》: 什么是 SinoDB ESQL/C?
《SinoDB DataBlade API 程序员指南》: DataBlade API 概述

智能大对象空间的磁盘布局

可以在与表（该表与数据相关联）分开的磁盘上创建智能大对象空间。可以在同一智能大对象空间内存储与不同表相关联的智能大对象。将智能大对象与相关联的表分开存储在不同磁盘上的智能大对象空间中时，数据库服务器会提供一些性能优势。

这些性能优势为：

- 您对表和智能大对象有并行访问权。
- 当选择不在智能大对象空间中记录数据时，可以减少已记录的数据库的日志记录 I/O 活动。

要创建智能大对象空间，请使用 onspaces 实用程序。使用 CREATE TABLE 语句来创建与智能大对象相关联的表时，可以将智能大对象分配到智能大对象空间。

相关链接

《SinoDB 管理员参考》: onspaces -c -S: 创建智能大对象空间
《SinoDB SQL 指南: 语法》: CREATE TABLE 语句

影响智能大对象空间 I/O 的配置参数

SBSPACENAME、BUFFERPOOL 和 LOGBUFF 配置参数会影响智能大对象空间的 I/O 性能。

如果在定义一列 CLOB 或 BLOB 数据类型时没有指定智能大对象空间名称，那么 SBSPACENAME 配置参数会指示缺省的智能大对象空间名称。要减少磁盘争用并提供更好的负载均衡，那么将缺省智能大对象空间放在与表数据分开的磁盘上。

对于缺省页大小缓冲池和任何非缺省页大小缓冲池，BUFFERPOOL 配置参数都会为缓冲池中的缓冲区和 LRU 队列指定缺省值。内存缓冲池的大小会影响智能大对象的 I/O 操作，因为缓冲池是这些对象共享内存的缺省区域。如果应用程序频繁访问智能大对象，那么将这些对象保留在缓冲池中是很有利的。智能大对象只使用缺省页大小的缓冲池。有关估算为智能大对象增加的缓冲池量的信息，请参阅 *BUFFERPOOL* 配置参数和内存利用率 在第62页。

缺省情况下，数据库服务器将智能大对象读入共享内存常驻部分的缓冲区中。有关使用轻量级 I/O 缓冲区的更多信息，请参阅智能大对象的轻量级 *I/O* 在第100页。

LOGBUFF 配置参数会影响日志记录 I/O 活动，这是因为其指定了共享内存中逻辑日志缓冲区的大小。这些缓冲区的大小决定其被填满所需的时间以及相应的清空到磁盘的频率。

如果记录智能大对象用户数据，那么请增加逻辑日志缓冲区的大小，以防止频繁清空到磁盘上的这些日志文件。

相关链接

《SinoDB 管理员参考》: SBSPACENAME 配置参数

《SinoDB 管理员参考》: BUFFERPOOL 配置参数

《SinoDB 管理员参考》: LOGBUFF 配置参数

影响智能大对象空间 I/O 的 onspaces 选项

使用 onspaces 实用程序创建智能大对象空间时，您可以指定影响 I/O 性能的信息。此信息包括扩展数据块的大小、缓冲方式（是否希望服务器使用轻量级 I/O）和日志记录。

智能大对象空间扩展数据块

将智能大对象添加到表时，数据库服务器将磁盘空间以称为扩展数据块的单位分配到智能大对象空间。 每个扩展数据块都是一个块，由来自智能大对象空间物理上连续的页组成。

即使智能大对象空间包含多个块，每个扩展数据块都完整地分配到一个块内，从而保持连续。连续性对于 I/O 性能很重要。

当数据页为连续时，磁盘臂运动就可在数据库服务器按顺序读取行时减到最小。扩展数据块机制是以下互相冲突的需求之间的一种折衷：

- 有些智能大对象的大小是事先未知的。
- 不同表中智能大对象的数量可能在不同时间、以不同速率增长。
- 检索整个对象时，要获取最佳性能，单个智能大对象的所有页最好是相邻的。

因为您可能无法预知智能大对象的数量和大小，所以不能指定智能大对象的扩展数据块长度。因此，数据库服务器只有在需要时添加扩展数据块，但是要获得更好的性能，任何一个扩展数据块中的所有页都应该是连续的。此外，数据库服务器创建与上一个扩展数据块相邻的新扩展数据块时，它会将这两个扩展数据块视为一个扩展数据块。

智能大对象空间数据块中的页数由以下其中一种方法确定：

- 数据库服务器根据一组启发式搜索计算智能大对象的扩展数据块大小（如写操作中的字节数）。例如，如果某个操作要求写入 30 KB，数据库服务器将尝试分配一个 30 KB 大小的扩展数据块。
- 当您在应用程序中打开智能大对象空间时，智能大对象的最终大小由以下其中一个函数指示：
  - 对于 *DB-Access:* DataBlade® API mi_lo_specset_estbytes 函数。有关打开智能大对象和设置估算字节数的 DataBlade® API 函数的更多信息，请参阅《*SinoDB® DataBlade® API* 程序员指南》。
  - 对于 *ESQL/C:* SinoDB® ESQL/C ifx_lo_specset_estbytes 函数。有关打开智能大对象和设置估算字节数的 SinoDB® ESQL/C 函数的更多信息，请参阅《*SinoDB® ESQL/C* 程序员指南》。

这些函数是设置扩展数据块大小的最佳方法，因为它们可以减少智能大对象中的扩展数据块数量。数据库服务器将尝试把整个智能大对象作为一个扩展数据块进行分配（如果块中存在该大小的可用扩展数据块的话）。

- 创建或更改智能大对象空间时 onspaces 命令的 -Df 选项中的 EXTENT_SIZE 标志

大多数管理员不使用 onspaces EXTENT_SIZE 标志，因为数据库服务器会根据启发式搜索来计算扩展数据块的大小。但是可以考虑在以下情况中使用 onspaces EXTENT_SIZE 标志：

- 许多单页的扩展数据块分散在整个智能大对象空间中。
- 几乎所有的智能大对象长度都相同。

- 定义 CLOB 或 BLOB 列时 CREATE TABLE 语句的 EXTENT SIZE 关键字

  大多数管理员在创建或替换表时不使用 EXTENT SIZE 关键字，因为数据库服务器会根据启发式搜索计算扩展数据块大小。但是，如果几乎所有的智能大对象长度都相同，可以考虑使用此 EXTENT SIZE 关键字。

  重要： 对于大多数应用程序来说，应该使用数据库服务器为扩展数据块大小计算的值。请勿使用 DataBlade® API mi_lo_specset_extsz 函数或 SinoDB® ESQL/C ifx_lo_specset_extsz 函数设置智能大对象的扩展数据块大小。

如果知道智能大对象的大小，建议您在 DataBlade® API mi_lo_specset_estbytes() 函数或 SinoDB® ESQL/C ifx_lo_specset_estbytes() 函数中指定大小，而不是在 onspaces 实用程序或 CREATE TABLE 或 ALTER TABLE 语句中指定。这些函数是设置扩展数据块大小的最佳方式，因为数据库服务器将整个智能大对象作为一个扩展数据块进行分配（如果它在此块中有连续存储的话）。

超过 1 MB 的扩展数据块大小在 I/O 上优势不大，因为数据库服务器最多以 60 KB 的倍数执行读写操作。但是，数据库服务器在元数据区中注册智能大对象的每个扩展数据块，因此，大型智能大对象会有很多扩展数据块条目。当数据库服务器访问这些扩展数据块条目时，其性能会下降。在这种情况下，如果在 mi_lo_specset_estbytes() 函数或 ifx_lo_specset_estbytes() 函数中指定了智能大对象的最终大小，那么可以减少元数据区中扩展数据块条目的数量。

有关更多信息，请参阅改善智能大对象的元数据 *I/O* 在第124页。

智能大对象的轻量级 I/O

如果不使用缓冲池，管理员和编程人员可以选择使用轻量级 *I/O*。轻量级 I/O 操作在共享内存虚拟部分的会话池中使用专用缓冲区。

缺省情况下，智能大对象经过共享内存常驻部分中的缓冲池。尽管智能大对象比其他数据的优先级要低，但当应用程序访问许多智能大对象时，缓冲池可能会填满。单个应用程序就可以使用智能大对象填满缓冲池，而且几乎不留下其他应用程序所需的数据空间。此外，当数据库服务器将许多页扫描到缓冲池中时，与检查进出的各页相关联的开销和争用可能会成为瓶颈。

重要： 只有当您在大于 8080 字节的读写操作中读写智能大对象并很少访问时，才使用专用缓冲区。即，如果在单个函数调用中很少执行读取大量数据的读写函数调用，那么轻量级 I/O 可提高 I/O 性能。

相关链接

BUFFERPOOL 配置参数和内存利用率 在第62页

智能大对象轻量级 I/O 的优势

因为数据库服务器未使用缓冲池，所以轻量级 I/O 会提供一些性能优势。

轻量级 I/O 提供以下优势：

- 在一次 I/O 操作中传输较大的数据块

  这些 I/O 块可大到 60 KB。但是这些字节必须是相邻的，这样数据库服务器才能在一次 I/O 操作中传输这些字节。
- 读取许多页时可以绕过缓冲池的开销
- 当读取智能大对象的许多连续页时，可以避免经常访问的页被挤出缓冲池

为智能大对象使用轻量级 I/O 缓冲区时，数据库服务器在一次 I/O 操作中可能读取多个页。单个 I/O 操作可读入数个智能大对象页，最多可读取一个数据块的大小。有关何时指定扩展数据块大小的信息，请参阅智能大对象空间扩展数据块 在第99页。

指定智能大对象的轻量级 I/O
要在创建智能大对象空间时指定使用轻量级 I/O，请使用 onspaces -c -S 命令的 -Df 选项中的 BUFFERING 标签。

BUFFERING 的缺省值为 ON，这意味着使用缓冲池。在 onspaces 命令中指定的（如果不指定，那么使用缺省值）缓冲方式是智能大对象空间内存储的所有智能大对象的缺省缓冲方式。

> 重要： 通常，如果智能大对象的读写操作小于 8080 字节，那么创建智能大对象空间时不要指定缓冲方式。如果正读取或写入短的数据块（如 2 KB 或 4 KB），那么保留缺省值“buffering=ON”以便获取更好的性能。

编程人员在用 DataBlade® API 和 SinoDB® ESQL/C 函数创建、打开或替换智能大对象时可以覆盖缺省缓冲方式。DataBlade® API 和 SinoDB® ESQL/C 应用程序编程接口提供 LO_NOBUFFER 标志以允许轻量级 I/O 用于智能大对象。

> 重要： 只有在操作中读取或写入的智能大对象大于 8080 字节，并且您很少访问这些对象时，才使用 LO_NOBUFFER 标志。即，如果在单个函数调用中很少执行读取大量数据的读写函数调用，那么轻量级 I/O 可提高 I/O 性能。

相关链接

《SinoDB 管理员参考》: onspaces -c -S: 创建智能大对象空间

《SinoDB ESQL/C 程序员指南》: 什么是 *SinoDB ESQL/C?*

《SinoDB DataBlade API 程序员指南》: DataBlade API 概述

日志记录
如果决定记录有关存储在智能大对象空间中数据的所有写操作，那么逻辑日志 I/O 活动和内存利用率会增加。

有关更多信息，请参阅影响智能大对象空间 *I/O* 的配置参数 在第98页。

表 I/O

数据库服务器最常执行的功能之一是将数据和索引页从磁盘引入内存。页可以为简短的事务逐个读取，为某些查询顺序读取。可以配置数据库服务器引入内存的页数，并可配置顺序扫描的 I/O 请求计时。

也可以指定当某个查询从暂时不可用的数据库空间请求数据时数据库服务器的响应方式。

以下各节描述这些读取页的方法。

有关智能大对象的 I/O 信息，请参阅影响智能大对象 *I/O* 的因素 在第98页。

顺序扫描

当数据库服务器执行数据或索引页的顺序扫描时，寻找适当的起始页在大多数情况下会导致 I/O 等待。要显著提高顺序扫描的性能，可以在每个 I/O 操作中引入一组连续的页。

在顺序扫描中，随第一页一起引入其他页的操作称为预读取。

顺序扫描所需的 I/O 操作定时同样至关重要。如果扫描线程在处理完每一批页后，必须等待下一组页的引入，那么会产生延迟。对第二个以及后续的读取请求（要求提前引入所需的页）进行定时可以最大限度地提高顺序扫描的效率。引入的页和预先读取 I/O 请求的频率取决于内存缓冲区中空间的可用性。如果每批引入的页太多，或者各批引入太频繁，那么预读取操作可能会使页清除增加到无法接受的程度。

相关链接

《SinoDB 管理员指南》: 预读操作

轻度扫描

某些表的顺序扫描可以使用轻度扫描来读取数据。轻度扫描可通过利用会话内存来绕过缓冲池以直接从磁盘中读取。

与使用缓冲池进行顺序扫描和跳过大型表扫描相比，轻度扫描可以提供性能优势。这些优点包括：

- 读取许多数据页时可以绕过缓冲池的开销
- 当为单个查询读取许多连续页时，可以避免经常访问的页被强行排除出缓冲池。

在以下这些条件下会发生轻度扫描：

- 优化器选择对表的顺序扫描或跳跃式扫描。
- 表中的数据量超过 1 MB。
- 该查询满足以下其中一个锁定条件：
  - 隔离级别为 Dirty Read（或数据库无事务日志记录）。
  - 该表在整个表上至少有一个共享锁，并且隔离级别不是 Cursor Stability。

注： Repeatable Read 隔离中的顺序扫描会自动获取表上的共享锁。

轻度扫描无法访问的表

轻度扫描只在用户表（其数据行存储在表空间）上执行。轻度扫描不用于访问索引或者存储在 BLOB 空间、智能 BLOB 空间或分区 BLOB 中的数据。同样，轻度扫描不用于访问系统目录表中的数据或 sysadmin、sysmaster、sysuser 和 sysutils 等系统数据库的表和伪表中的数据。

影响轻度扫描的配置设置

如果 BATCHEDREAD_TABLE 配置参数或 SET ENVIRONMENT 语句的 IFX_BATCHEDREAD_TABLE 会话环境选项设置为 0，那么轻度扫描将无法用于访问具有可变长度行的表或行长度大于包含表的数据库空间的页大小的表。可变长度行包括具有可变长度列的表，例如，VARCHAR、LVARCHAR 或 NVARCHAR 以及被压缩的表。

您可以使用 SET ENVIRONMENT 语句的 IFX_BATCHEDREAD_TABLE 会话环境选项或 onmode -wm 命令覆盖当前会话的 BATCHEDREAD_TABLE 配置参数的设置。可以使用 onmode -wf 命令更改 ONCONFIG 文件中 BATCHEDREAD_TABLE 的值。

轻度扫描过程中 onstat 输出的示例

如果您有长时间运行的扫描，那么您可以查看 onstat -g scn 命令输出来检查扫描进度，以确定扫描将耗费多长时间才能完成，并查看该扫描是轻度扫描还是缓冲池扫描。

以下示例显示了轻度扫描的 onstat -g scn 的部分输出。Scan Type 字段中的词 Light 确定了扫描为轻度扫描。

```
SesID  Thread Partnum  Rowid  Rows Scan'd  Scan Type  Lock Mode  Notes
17     48     300002   207    15           Light                 Forward row lookup
```

相关链接

《SinoDB 管理员参考》: BATCHEDREAD_TABLE 配置参数

《SinoDB 管理员参考》: onstat -g scn 命令: 显示扫描信息

不可用的数据

表 I/O 的另一方面适用于以下情形：查询请求访问位于暂时不可用的数据库空间中的表或分段。当数据库服务器确定由于磁盘故障导致数据库空间不可用时，缺省情况下针对该数据库空间的查询会失败。数据库服务器允许您指定查询可跳过的数据库空间（当其不可用时）。

有关指定在不可用时可由查询跳过的数据库空间的信息，请参阅 *DATASKIP 如何影响表 I/O* 在第103页。

警告： 如果包含查询所请求的数据的数据库空间在 DATASKIP 配置参数中列出且当前由于磁盘故障而不可用，那么数据库服务器返回给查询的数据可能与数据库的实际内容不一致。

影响表 I/O 的配置参数

AUTO_READAHEAD 配置参数更改查询的自动预读取方式或禁用自动预读取。此外，DATASKIP 配置参数可允许或禁止跳过数据。

当 SinoDB® 检测到查询遇到 I/O 时，自动预读取处理会通过发出异步页请求来帮助提高查询性能。异步页请求通过使用从磁盘检索数据并将其置于缓冲池中所需的处理来覆盖查询处理，以便提高查询性能。您也可以使用 SQL 的 SET ENVIRONMENT 语句的 AUTO_READAHEAD 环境选项来启用或禁用会话的 AUTO_READAHEAD 配置参数值。

相关链接

《SinoDB 管理员参考》: AUTO_READAHEAD 配置参数

DATASKIP 如何影响表 I/O

DATASKIP 配置参数允许您指定哪些数据库空间（如果有）在由于磁盘故障不可用时可以跳过。您可以列出特定的数据库空间并为所有的数据库空间打开或关闭数据跳过功能。

启用数据跳过功能时，数据库服务器会将 SQLWARN 数组中的第六个字符设置为 W。

> 警告： 数据库服务器无法确定当数据库空间被跳过时，查询的结果是否一致。如果数据库空间包含表分段，那么执行查询的用户必须确保对于精确的查询结果并不需要该分段中的行。启用 DATASKIP 使带有不完整数据的查询能够返回结果，这些结果可能与数据库的实际状态不一致。如果不小心，该数据会产生不正确或易误解的查询结果。

相关链接

《SinoDB 管理员参考》: DATASKIP 配置参数

《SinoDB SQL 指南: 教程》: SQLWARN 数组

后台 I/O 活动

后台 I/O 活动不直接为 SQL 请求服务。许多此类的活动对维护数据库的一致性以及数据库服务器操作的其他方面至关重要。毕竟，它们造成 CPU 中的开销并占用 I/O 带宽。

这些需要开销的活动会占用查询和事务的时间。如果没有适当配置后台 I/O 活动，那么这些活动的过高开销会限制应用程序的事务吞吐量。

以下列表显示一些后台 I/O 活动：

- 检查点
- 日志记录
- 页清除
- 备份与复原
- 回滚和恢复
- 数据复制
- 审计

虽然随着活动的增加，检查点发生的频率也会增大，但不管是否有许多数据库活动发生，检查点都会发生。其他后台活动（如日志记录和页清除）随着数据库使用的增加会更频繁地发生。某些活动（如备份、还原或快速恢复）只按计划或者在异常情况下才会发生。

大多数情况下，调整后台 I/O 活动包括在适当的检查点时间间隔、日志记录模式和日志大小以及页清除阈值之间达到平衡。触发后台 I/O 活动的阈值和时间间隔经常是相互作用的；对一个阈值的调整可能会将性能瓶颈转移到另一个阈值上。

以下各节描述与影响这些后台 I/O 活动的配置参数相关联的性能影响和注意事项。

影响检查点的配置参数

RTO_SERVER_RESTART、CKPTINTVL、LOGSIZE、LOGFILES、PHYSFILE 和 ONDBSPACEDOWN 配置参数会影响检查点。

RTO_SERVER_RESTART 及其对检查点的影响

RTO_SERVER_RESTART 配置参数指定 SinoDB® 从意外中断恢复所要的时间（以秒计）。

启用此配置参数的性能优势在于：

- 通过使用日志重放所需的数据页为缓冲区设置种子来加快恢复以满足 RTO_SERVER_RESTART 策略。

启用此配置参数的性能缺点在于：

- 增加物理日志活动，可能轻微影响事务性能
- 增加检查点频率，因为物理日志空间将被更快用完（可增加物理日志的大小以避免检查点频率的增加）。

当启用 RTO_SERVER_RESTART 时，数据库服务器将：

- 如果事务可能耗尽了物理或逻辑日志资源（将导致事务阻塞），那么尝试通过更频繁地触发检查点来确保非分块检查点在检查点处理过程中不会耗尽临界资源。
- 忽略 CKPTINTVL 配置参数。
- 自动控制检查点频率以符合 RTO 策略并防止服务器日志资源耗尽。
- 自动调整 AIO 虚拟处理器和 cleaner 线程的数量并自动调节 LRU 清空。

如果数据库服务器不满足 RTO_SERVER_RESTART 策略的要求，那么服务器会打印消息日志中的警告信息。

相关链接

《SinoDB 管理员参考》: RTO_SERVER_RESTART 配置参数

自动检查点、LRU 调整以及 AIO 虚拟处理器调整

数据库服务器自动调整检查点频率以避免事务阻塞。要完成这一操作，服务器会监视物理和逻辑日志消耗以及过去检查点性能的信息。然后在必要时，服务器会更加频繁地触发检查点以避免事务阻塞。

可以通过将 onmode -wf AUTO_CKPTS 设置为 0 来自动关闭检查点调整，或将 AUTO_CKPTS 配置参数设置为 0。

由于数据库服务器在检查点处理期间不会阻塞事务，因此应该暂缓 LRU 清空。如果服务器在耗尽物理日志（可能会导致事务阻塞）之前不能完成检查点处理，并且如果不能增加物理日志的大小，那么可以将服务器的 LRU 清空配置为更高频率。增加 LRU 清空会影响事务性能，但可以减少事务阻塞。如果您没有将服务器的清仓设为更频繁，那么只有当服务器无法为页替换找到低优先级缓冲区时，才会自动将 LRU 清空调整为更频繁。

如果 AIO 虚拟处理器的 VPCLASS 配置参数设置为 autotune=1，那么当服务器检测到 AIO 虚拟处理器无法满足 I/O 工作负载时，数据库服务器会自动增加 AIO 虚拟处理器和 page-cleaner 线程的数量。

自动 LRU 调整会影响所有的缓冲池并调整 BUFFERPOOL 配置参数中的 lru_min_dirty 和 lru_max_dirty 值。

相关链接

《SinoDB 管理员参考》: AUTO_CKPTS 配置参数

《SinoDB 管理员参考》: BUFFERPOOL 配置参数

《SinoDB 管理员参考》: VPCLASS 配置参数

LRU 调整 在第115页

CKPTINTVL 及其对检查点的影响

如果未启用 RTO_SERVER_RESTART 配置参数，CKPTINTVL 配置参数将指定数据库服务器通过检查来确定是否需要检查点的频率（以秒计）。

当启用 RTO_SERVER_RESTART 配置参数时，数据库服务器将忽略 CKPTINTVL 配置参数。相反，服务器自动触发检查点以维持 RTO_SERVER_RESTART 策略。

如果所有数据在检查点时间间隔到期时物理上保持一致，那么数据库服务器将跳过该检查点。

检查点也会在以下任一情况中出现：

- 只要物理日志填充度达到 75% 时
- 在物理日志填充度不到 75%，但存在大量的脏分区时。

  发生这种情况是因为数据库服务器在检查物理日志填充度是否达到 75% 时，也会检查是否满足以下条件：

  ```
  (使用的物理日志页数 + 脏分区数量) >=
  (物理日志大小 * 9) /10)
  ```

  分区表示在检查点处理期间进入物理日志的一个页，具有一个维护其相关信息（例如，行数和数据页数）的页，并在更新后会变为脏分区。

如果您将 CKPTINTVL 设置为长时间间隔，那么可以使用物理日志功能来根据实际的数据库活动而不是任意时间单位来触发检查点。但是，长检查点时间间隔会增加出现故障时进行恢复所需的时间。根据吞吐量和数据可用性需求，您可以将初始检查点时间间隔设置为 5、10 或 15 分钟，同时应该理解，根据物理日志记录的活动，检查点可能会更频繁地发生。

数据库服务器将消息写入消息日志，记录完成检查点的时间。要读取这些消息，请使用 onstat -m。

相关链接

《*SinoDB* 管理员参考》: *CKPTINTVL* 配置参数

LOGSIZE 和 LOGFILES 及其对检查点的影响

LOGSIZE 和 LOGFILES 配置参数会间接影响检查点，因为其指定逻辑日志文件的大小和数量。当数据库服务器检测到下一个即将成为当前逻辑日志文件的逻辑日志文件包含最新的检查点记录时，那么发生检查点。

如果您需要释放包含最近检查点的逻辑日志文件，那么数据库服务器必须在当前逻辑日志文件中写入一个新的检查点记录。如果逻辑日志文件备份和释放的频率加快，那么检查点发生的频率也会加快。虽然检查点会阻塞用户的处理，但是不会再持续很长的时间。因为其他因素（如物理日志大小）也会决定检查点的频率，所以这一影响并不是很重要。

启用动态日志分配功能时，逻辑日志的大小对长事务阈值的影响不再像其在以前版本的数据库服务器中的那么大。有关详细信息，请参阅 *LTXHWM* 和 *LTXEHWM* 及其对日志记录的影响 在第110页。

LOGSIZE、LOGFILES 和 LOGBUFF 配置参数也会影响日志记录 I/O 活动和逻辑备份。有关更多信息，请参阅影响日志记录的配置参数 在第106页。

相关链接

《*SinoDB* 管理员参考》: *LOGFILES* 配置参数

《*SinoDB* 管理员参考》: *LOGSIZE* 配置参数

《*SinoDB* 管理员指南》: 估计逻辑日志文件的数量

检查点和物理日志

PHYSFILE 配置参数指定初始物理日志的大小。当物理日志填充度达到 75% 时或存在大量的脏分区时，将会出现检查点。

事务生成物理日志活动的速度可以影响检查点性能。要在检查点处理过程中避免事务阻塞，请考虑物理日志的大小及填充速度。

您可以通过为物理日志创建可扩展的物理日志空间，使数据库服务器根据需要扩展物理日志大小来改善性能。

例如，不执行更新的操作不会生成前映象。如果数据库的大小增加而应用程序很少更新数据，那么不会生成许多物理日志记录。在这种情况下，您可能不需要大的物理日志。

同样，如果应用程序更新相同的页，那么您可以定义较小的物理日志。数据库服务器仅写入对页执行的第一个更新的前映象，如以下操作：

- 插入、更新和删除包含用户定义数据类型 (UDT)、智能大对象和简单大对象的行

- ALTER 语句
- 创建或修改索引（B 型树、R 型树或用户定义索引）的操作

因为每次发生检查点后会回收物理日志，所以物理日志的空间必须足以容纳检查点之间所发生更改的前映象。如果由于数据库服务器用尽物理日志空间而需频繁地触发检查点，那么请考虑增加物理日志的大小。

如果您增加检查点的时间间隔，或者如果您预期更新活动会增加，那么您可能希望增加物理日志的大小。

物理日志是维护 RTO_SERVER_RESTART 策略的重要部分。要确保具有足够的物理日志空间，请将物理日志的大小设置为所有缓冲池大小的至少 110%。

您可以使用 onparams 实用程序更改物理日志的位置和大小。您可以在激活事务时更改物理日志而不用重新启动数据库服务器。

相关链接

影响关键数据的配置参数 在第90页

《*SinoDB* 管理员参考》: *PHYSFILE* 配置参数

《*SinoDB* 管理员指南》: 用于估计物理日志大小的策略

《*SinoDB* 管理员指南》: 更改物理日志的位置和大小

《*SinoDB* 管理员指南》: 物理日志空间

ONDBSPACEDOWN 及其对检查点的影响

ONDBSPACEDOWN 配置参数是指定当某个 I/O 错误指示数据库空间关闭时数据库服务器做出的响应。缺省情况下，数据库服务器将任何不包含关键数据的数据库空间标识为 down，并继续处理。关键数据包括根数据库空间、逻辑日志或物理日志。

要还原对数据库的访问，您必须先备份所有逻辑日志，然后对关闭的数据库空间执行热还原。

无论 ONDBSPACEDOWN 是如何设置的，只要在包含关键数据的非镜像数据库空间上出现禁用 I/O 错误，数据库服务器就会终止运行。在这种情况下，您必须对数据库服务器执行冷还原，才能重新开始正常的数据库操作。

ONDBSPACEDOWN 的值对临时数据库空间没有影响。对于临时数据库空间，不管怎样设置 ONDBSPACEDOWN，数据库服务器都将继续处理。如果临时数据库空间需要修订，可将其删除并重新创建。

当 ONDBSPACEDOWN 设置为 2 时，数据库服务器将继续处理下一个检查点，然后暂挂所有更新请求的处理。数据库服务器将反复重试产生错误的 I/O 请求，直到修复了数据库空间并完成请求，或者直到数据库服务器管理员介入。管理员可使用 onmode -O 将数据库空间标为 down，然后在数据库空间仍不可用时继续处理，或者使用 onmode -k 使数据库服务器停机。

> 重要: 将 ONDBSPACEDOWN 设置为 *2* 会严重影响更新请求的性能，因为它们会由于数据库空间关闭而暂挂。当您在 ONDBSPACEDOWN 中使用本设置时，请确保监视器的状态为数据库空间。

当将 ONDBSPACEDOWN 设置为 1 时，数据库服务器会将所有数据库空间视为关键数据库空间。任何非镜像数据库空间无法使用时都将终止正常的处理，并要求执行冷还原。当数据库空间关闭时，终止运行并执行冷还原所造成的性能影响会是非常严重的。

> 重要: 如果决定将 ONDBSPACEDOWN 设置为 *1*，请考虑镜像所有的数据库空间。

相关链接

《*SinoDB* 管理员参考》: *ONDBSPACEDOWN* 配置参数

影响日志记录的配置参数

LOGBUFF, PHYSBUFF, LOGFILES, LOGSIZE, DYNAMIC_LOGS, AUTO_LLOG, LTXHWM, LTXEHWM, SESSION_LIMIT_LOGSPACE, SESSION_LIMIT_TXN_TIME 和 TEMPTAB_NOLOG 配置参数会影响日志记录。

日志记录、检查点和页清除对于保持数据库一致性是必要的。在检查点的频率（或逻辑日志的大小）和出现故障时恢复数据库所需的时间之间存在着直接的权衡关系。因此，当您试图减少这些活动的开销时，在恢复过程中您所能接受的延迟是主要的考虑因素。

LOGBUFF 和 PHYSBUFF 及其对日志记录的影响
LOGBUFF 和 PHYSBUFF 配置参数会影响日志记录 I/O 活动，因为它们分别指定共享内存中的逻辑日志缓冲区和物理日志缓冲区的大小。这些缓冲区的大小确定缓冲区填充的速度以及相应的需要清空到磁盘的频率。

相关链接

《*SinoDB* 管理员参考》: *LOGBUFF* 配置参数
《*SinoDB* 管理员参考》: *PHYSBUFF* 配置参数

LOGFILES 及其对日志记录的影响
指定逻辑日志文件数量的 LOGFILES 配置参数会影响日志记录。

当您初始化或重新启动数据库服务器时，该服务器将创建您在 LOGFILES 配置参数中所指定数量的逻辑日志文件。

您添加逻辑日志文件可能出于以下原因：

- 增加分配给逻辑日志的磁盘空间
- 更改逻辑日志文件的大小
- 允许打开的事务回滚
- 作为将逻辑日志文件移至不同数据库空间的操作的一部分

相关链接

《*SinoDB* 管理员参考》: *LOGFILES* 配置参数
《*SinoDB* 管理员指南》: 估计逻辑日志文件的数量

计算分配给逻辑日志文件的空间
如果所有的逻辑日志文件都大小相同，那么可以计算分配给逻辑日志文件的总空间。

要计算分配给这些文件的空间，请使用以下公式：

```
逻辑日志空间总数 = LOGFILES * LOGSIZE
```

如果要添加的逻辑日志文件大小并非 LOGSIZE 配置参数所指定，那么无法使用 LOGFILES * LOGSIZE 表达式计算逻辑日志的大小。相反，需要添加磁盘上每个单独日志文件的大小。

使用 onstat -l 实用程序监视逻辑日志文件。

LOGSIZE 及其对日志记录的影响
LOGSIZE 配置参数会指定每个逻辑日志文件的大小。在系统完全投入使用之前，很难预测数据库服务器系统需要多少逻辑日志空间。

逻辑日志空间的大小 (LOGFILES * LOGSIZE) 由以下策略确定：

恢复时间目标 (RTO)

这是您能够承受没有系统的时间。如果您的目标只是失败恢复，总的日志空间只要足以容纳两个检查点周期的所有事务即可。当启用 RTO_SERVER_RESTART 配置参数并且服务器具有低于 4 GB的组合缓冲池时，您可以将总的日志空间配置为组合缓冲池大小的 110%。日志空间过多不会影响性能；但是，日志空间过少会使检查点和事务阻塞更加频繁。

恢复点对象 (RPO)

这描述了在发生灾难时想要还原的数据的存在时间。如果目的是保护事务性工作，那么最佳 LOGSIZE 应该是每个 RPO 单元所完成工作量的倍数。由于数据库服务器支持部分日志备份，因此最佳的日志大小并不重要，非最佳的日志大小只意味着更频繁的日志文件更改。RPO 以时间为单位来评估。如果业务规则是在发生整个站点灾难时系统不能丢失超过十分钟的事务性数据，那么应该每隔十分钟进行一次日志备份。

您可以使用管理并执行已调度管理任务的调度程序来设置自动日志备份。

长事务数

如果您有需要大量日志空间的长事务，那么应该给这些日志分配空间。日志空间不足将影响事务性能。

根据日志记录活动发生的数量以及可能发生灾难性故障的次数来选择日志大小。如果您不能承受超过一小时的数据丢失，那么可以创建许多小的日志文件，每个日志文件保存一小时内的事务。启用连续日志备份。小逻辑日志文件很快就会填满，这意味着逻辑日志备份将更频繁。

如果您的系统非常稳定，可以执行大量日志记录活动，那么可以选择较大的日志来提高性能。使用大的日志文件后，连续日志备份发生的频率会降低。同样，也应该考虑事务比率最大值和备份设备的速度。请勿让整个逻辑日志填满。启用连续日志备份，并在逻辑日志中留下足够的空间来处理最长的事务。

备份进程会阻碍涉及到与逻辑日志文件位于同一磁盘上的数据的事务处理。如果有足够的可用逻辑日志磁盘空间，那么您可以等到用户活动较少的时段，再备份逻辑日志文件。

相关链接

《*SinoDB* 管理员参考》：*LOGSIZE* 配置参数

《*SinoDB* 管理员指南》：调度程序

估算记录数据库空间时的逻辑日志大小

要估算逻辑日志的大小，请使用公式或 onstat -u 信息。

使用以下公式获取 LOGSIZE 的初始估算值（以 KB 为单位）：

```
LOGSIZE = (connections * maxrows * rowsize) / 1024) / LOGFILES
```

在此公式中：

- *connections* 是在 sqlhosts 信息中由一个或多个 NETTYPE 参数指定的所有网络类型的最大连接数。如果在配置文件中通过设置多个 NETTYPE 配置参数而配置了多个连接，那么将每个 NETTYPE 参数的 users 字段相加，并用此总数替换上述公式中的 *connections*。
- *maxrows* 是单个事务中要更新的最大行数。
- *rowsize* 是行的平均大小（以字节为单位）。可以通过将行中列的长度（来自 syscolumns 系统目录表）相加来计算 *rowsize*。
- *1024* 是必需的约数，因为您以 KB 为单位指定 LOGSIZE。

要获取高峰活动期间更好的估算值，请执行 onstat -u 命令。onstat -u 输出的最后一行包含最大并发连接数。

当事务包括简单大对象或智能大对象时，您需要调整逻辑日志的大小，如以下各节所述。

也可以通过添加另一个逻辑日志文件来增加专用于逻辑日志的空间量。

相关链接

《*SinoDB* 管理员指南》：手动添加逻辑日志文件

估算记录简单大对象时的逻辑日志大小

要为频繁执行 BLOB 空间中 TEXT 或 BYTE 数据更新的应用程序获取更好的整体性能，请减少逻辑日志的大小。

在备份分配了 BLOB 页的逻辑日志之前，这些 BLOB 页不能重新使用。当 TEXT 或 BYTE 数据活动较多时，通过提高可用 BLOB 页的可用性来平衡更频繁的检查点对性能产生的影响。

当使用 BLOB 空间中不稳定的 BLOB 页时，较小的日志可改善对必须重新使用的简单大对象的访问。在分配了简单大对象的日志被清空到磁盘之前，简单大对象不能重新使用。在这种情况下，您必须以性能为代价来进行调整，因为那些较小日志文件的备份将更为频繁。

估算记录智能大对象时的逻辑日志大小

如果计划记录智能大对象用户数据，那么必须确保日志大小比正写入的数据量大得多。即使不记录智能大对象，智能大对象元数据页将始终被记录。

记录智能大对象时，请遵循以下准则：

- 如果要将数据附加到智能大对象，那么增加的日志记录活动大约等于写入智能大对象的数据量。

- 如果要更新智能大对象（覆盖数据），那么增加的日志记录活动大约是写入智能大对象的数据量的两倍。数据库服务器为更新事务同时记录智能大对象的前映象和后映象。当更新智能大对象时，数据库服务器只记录前映象或后映象的更新部分。
- 元数据更新较少影响日志记录。即使总是记录元数据，记录的字节数通常比智能大对象小得多。

DYNAMIC_LOGS 及其对日志记录的影响
动态日志文件分配功能可防止由于数据库服务器未用完日志空间而由长事务回滚导致的挂起问题。DYNAMIC_LOGS 配置参数会指定动态日志文件分配功能是关闭还是开启，或者使服务器暂停以允许手动添加逻辑日志文件。

动态日志分配使您能够执行以下操作：

- 在系统活动时，甚至是在快速恢复期间添加逻辑日志文件。
- 在当前日志文件之后紧跟着插入逻辑日志文件，而不是将其附加到最后。
- 立即访问逻辑日志文件（即使根数据库空间没有备份）。

DYNAMIC_LOGS 配置参数的缺省值为 2，这表示当数据库服务器检测到下一个日志文件包含打开的事务时，会自动在当前日志文件之后分配新的逻辑日志文件。数据库服务器会在以下时刻自动检查当前日志之后的日志是否仍然包含打开的事务：

- 在写入日志记录（不是读取和应用日志记录）时立即切换到新的日志文件后
- 在作为逻辑恢复的最后阶段发生的事务清除阶段开始时

  逻辑恢复在快速恢复结束以及冷恢复或前滚结束时发生。
- 在事务清除（回滚打开的事务）期间，可能会切换到新的日志文件日志

  因为要写回滚的日志记录，所以数据库服务器也会在此切换后进行检查。

当使用 DYNAMIC_LOGS 的缺省值 2 时，数据库服务器会为您确定新逻辑日志的位置和大小：

- 数据库服务器根据以下条件确定要分配新日志文件的磁盘：
  - 镜像的数据库空间优先
  - 在没有其他关键数据库空间可用之前，避免使用根数据库空间
  - 最后考虑的空间是未镜像和非关键数据库空间
- 数据库服务器为新逻辑日志文件的大小使用最大日志文件和最小日志文件的平均大小。如果对于此平均大小没有足够的连续磁盘空间可用，那么数据库服务器会搜索下一个最小平均大小的空间。数据库服务器为新日志文件最少分配 200 KB 的空间。

如果要控制附加日志文件的位置和大小，那么将 DYNAMIC_LOGS 设置为 1。当数据库服务器切换日志文件时，仍会检查下一个活动的日志是否包含打开的事务。如果在下一个活动的日志中的确发现了打开的事务，那么数据库服务器会执行以下操作：

- 发出警报事件 27（需要日志）
- 将警告消息写入联机日志
- 暂停并等待管理员用 onparams -a -i 命令行选项手动添加日志

可以编写在警报事件 27 发生时执行的脚本，以执行带有要用于新日志的位置的 onparams -a -i。该脚本也可以执行 onstat -d 命令，以检查是否有足够的空间并执行带有足够空间的位置的 onparams -a -i 命令。必须使用 -i 选项在当前日志文件之后添加新的日志。

如果将 DYNAMIC_LOGS 设置为 0，那么数据库服务器在切换日志文件时仍会检查下一个活动日志是否包含打开的事务。如果在下一个活动的日志中的确找到了打开的事务，那么数据库服务器会发出以下警告：

```
WARNING: The oldest logical log file (%d) contains records
from an open transaction (0x%p), but the Dynamic Log
Files feature is turned off.
```

相关链接

《*SinoDB* 管理员参考》：*DYNAMIC_LOGS* 配置参数
《*SinoDB* 管理员指南》：快速恢复

AUTO_LLOG 及其对日志记录的影响
逻辑日志不足会触发频繁的检查点，阻塞检查点或长检查点，从而影响性能。AUTO_LLOG 配置参数会控制数据库服务器是否自动添加逻辑日志以提高性能。

如果在安装过程中创建了服务器，那么会自动启用 AUTO_LLOG 配置参数。否则，您可以编辑 AUTO_LLOG 配置参数的值。

如果启用了 AUTO_LLOG 配置参数，那么数据库服务器会在下列情况下自动添加逻辑日志文件：

- 当最后 20 个检查点的大部分是由逻辑日志填满引起的时候
- 当逻辑日志空间不足导致阻塞检查点时
- 当逻辑日志空间不足导致长检查点时

在服务器停止添加逻辑日志以提高性能之前，AUTO_LLOG 配置参数也会指定新逻辑日志文件的数据库空间以及所有逻辑日志文件的最大大小。以下准则显示了您可能需要的逻辑日志的最大空间量估算值，具体取决于访问数据库服务器的并发用户数量：

- 1 - 100 个用户：200 MB
- 101 - 500 个用户：5 MB
- 501 - 1000 个用户：1 GB
- 超过 1000 个用户：2 GB

AUTO_LLOG 配置参数的设置和 DYNAMIC_LOGS 配置参数的设置不会相互影响。

相关链接

《SinoDB 管理员参考》：AUTO_LLOG 配置参数

LTXHWM 和 LTXEHWM 及其对日志记录的影响
LTXHWM 和 LTXEHWM 配置参数定义长事务水位标记。

动态日志文件功能发布后，长事务高水位标记不再那么重要，因为服务器不会耗尽日志空间，除非您耗尽系统上可用的物理磁盘空间。LTXHWM 参数在数据库服务器开始检查可能的长事务并将其回滚时，仍然会指示逻辑日志的填充度。LTXEHWM 仍然会指示数据库服务器暂挂新事物活动的时间点，以便定位并回滚长事务。这些事件应该很少发生，但如果发生，它们会造成应用程序中的严重后果。

在正常操作情况下，使用 LTXHWM 和 LTXEHWM 的缺省值。但是，由于以下某个原因您可能想要更改这些缺省值：

- 在长事务的回滚过程中允许其他事务继续更新活动（这些活动要求访问日志）

  在这种情况下，增加 LTXEHWM 的值可提高长事务回滚互斥访问日志的点。
- 执行已调度的未知长度的事务，例如记录的大量负载

  在这种情况下，增加 LTXHWM 的值使事务有机会在达到高水位标记之前完成。

相关链接

《SinoDB 管理员参考》：LTXEHWM 配置参数
《SinoDB 管理员参考》：LTXHWM 配置参数

TEMPTAB_NOLOG 及其对日志记录的影响
TEMPTAB_NOLOG 配置参数允许禁用临时表上的日志记录。这样设置可提高性能并防止 SinoDB® 在使用高可用性数据复制 (HDR) 时传输临时表。

要禁用临时表上的日志记录，可将 TEMPTAB_NOLOG 配置参数设置为 1。

相关链接

《SinoDB 管理员参考》：TEMPTAB_NOLOG 配置参数

SESSION_LIMIT_LOGSPACE 及其对日志记录的影响
SESSION_LIMIT_LOGSPACE 配置参数指定会话可用于单个事务的最大日志空间量，并且可以防止单个会话独占逻辑日志。

SESSION_LIMIT_LOGSPACE 不适用于拥有管理权限的用户，例如用户 informix 或 DBSA 用户。

相关链接

《*SinoDB* 管理员参考》: *SESSION_LIMIT_LOGSPACE* 配置参数

《*SinoDB* 管理员参考》: *SESSION_LIMIT_TXN_TIME* 配置参数

SESSION_LIMIT_TXN_TIME 及其对日志记录的影响

SESSION_LIMIT_TXN_TIME 配置参数限制事务在会话中可运行的时间，并且可以防止单个会话事务独占逻辑日志。

数据库服务器会终止超过 SESSION_LIMIT_TXN_TIME 限制的事务，并在数据库服务器消息日志中生成错误。

SESSION_LIMIT_TXN_TIME 不适用于拥有管理权限的用户，例如用户 informix 或 DBSA 用户。

影响页清除的配置参数

多个配置参数（包括 CLEANERS 和 RTO_SERVER_RESTART 配置参数）会影响页清除。如果很少清除页，那么执行查询的 sqlexec 线程可能无法找到所需的可用页。

如果 sqlexec 线程找不到其需要的可用页，那么线程会启动前台写入并等待页被释放。因为前台写入会降低性能，所以应避免使用。要减少前台写入的频率，可以增加页清除程序的数量或减小触发页清除的阈值。

使用 onstat -F 监视前台写入的频率。

以下配置参数会影响页清除：

- BUFFERPOOL，包含 lrus、lru_max_dirty 和 lru_min_dirty 值

  在 V10.0 之前使用 BUFFERS、LRUS、LRU_MAX_DIRTY 和 LRU_MIN_DIRTY 配置参数指定的信息，现在使用 BUFFERPOOL 配置参数指定。
- CLEANERS
- RTO_SERVER_RESTART

CLEANERS 及其对页清除的影响

CLEANERS 配置参数指示要运行的 page-cleaner 线程的数量。对于支持少于 20 个磁盘的安装，建议对每个包含数据库服务器数据的磁盘使用一个 page-cleaner 线程。对于支持 20 到 100 个磁盘的安装，建议每两个磁盘使用一个 page-cleaner 线程。对于更大的安装，建议每四个磁盘使用一个 page-cleaner 线程。

如果增加了 LRU 队列的数量，那么也要按比例增加 page-cleaner 线程的数量。

相关链接

《*SinoDB* 管理员参考》: *CLEANERS* 配置参数

BUFFERPOOL 及其对页清除的影响

BUFFERPOOL 配置参数指定在共享内存缓冲池内设置的最近最少使用的 (LRU) 队列数。缓冲池分布在 LRU 队列之间。配置更多的 LRU 队列后可使用更多的页清除程序，并且能够减小每个 LRU 队列。

对于单处理器系统，应该将 BUFFERPOOL 配置参数的 lrus 字段设置为最小值 8。对于多处理器系统，将 lrus 字段设置为最小值 8 或 CPU VP 数（这两个值中的较大者）。

lrus、lru_max_dirty 和 lru_min_dirty 值控制检查点之间对磁盘清空页的频率。自动 LRU 调整（由 AUTO_LRU 配置参数设置）会影响所有缓冲池并调整 BUFFERPOOL 配置参数中的 lru_min_dirty 和 lru_max_dirty 值。

如果您增加 lru_max_dirty 和 lru_min_dirty 的值来提高事务吞吐量，那么不要更改 lru_max_dirty 和 lru_min_dirty 之间的差值。

当检查点处理期间缓冲池非常大或出现事务阻塞时，请查看消息日志以确定正在触发事务阻塞的资源。如果物理或逻辑日志相当低并触发事务阻塞，那么增加导致事务阻塞的资源的大小。如果不能增加资源的大小，那么考虑通过减少 lru_min_dirty 和 lru_max_dirty 设置使得 LRU 清空更频繁，这样可使服务器在资源检查点处理期间有较少的页需要清空到磁盘。

要监视 LRU 队列中脏页的百分比，请使用 onstat -R命令。如果脏页数总是超过 lru_max_dirty 限制，那么说明您的 LRU 队列或页清除程序过少。首先，使用 BUFFERPOOL 配置参数增加 LRU 队列数。如果脏页的百分比仍超过 lru_max_dirty 限制，那么请使用 CLEANERS 配置参数来增加页清除程序数。

相关链接

BUFFERPOOL 配置参数和内存利用率 在第62页

《*SinoDB* 管理员参考》: *BUFFERPOOL* 配置参数

《*SinoDB* 管理员指南》: 要配置的 *LRU* 队列数

RTO_SERVER_RESTART 及其对页清除的影响

在您重新启动 SinoDB® 并将其置于联机或静默模式后，RTO_SERVER_RESTART 配置参数允许您使用恢复时间目标 (RTO) 标准来设置 SinoDB® 从问题中恢复所需的时间量（以秒计）。

启用此配置参数时，数据库服务器会自动调整 AIO 虚拟处理器和 cleaner 线程的数量并自动调整 LRU 清空。

使用 AUTO_LRU_TUNING 配置参数来指定服务器启动时启用还是禁用自动 LRU 调整。

相关链接

《*SinoDB* 管理员参考》: *RTO_SERVER_RESTART* 配置参数

《*SinoDB* 管理员参考》: *AUTO_LRU_TUNING* 配置参数

影响备份与还原的配置参数

影响所有操作系统上备份与还原的四个配置参数也会影响后台 I/O。其他配置参数会影响 UNIX™ 上的备份与还原。

以下配置参数会影响对所有操作系统的备份与还原：

- BAR_MAX_BACKUP
- BAR_NB_XPORT_COUNT
- BAR_PROGRESS_FREQ
- BAR_XFER_BUF_SIZE

此外，以下配置参数会影响对 UNIX™ 的备份与还原：

- LTAPEBLK
- LTAPEDEV
- LTAPESIZE
- TAPEBLK
- TAPEDEV
- TAPESIZE

ON-Bar 配置参数

BAR_MAX_BACKUP、BAR_NB_XPORT_COUNT、BAR_PROGRESS_FREQ 和 BAR_XFER_BUF_SIZE 是影响后台 I/O 的一些 ON-Bar 配置参数。

BAR_MAX_BACKUP 配置参数指定每个 ON-Bar 命令的最大备份进程数。该配置参数也定义了并行度来确定并发运行的进程数，其中包括用于备份与还原整个系统的进程。如果达到运行进程的数量，那么只有当正在运行的进程结束其操作时才开始后续进程。

BAR_NB_XPORT_COUNT 指定每个备份或还原进程的共享内存数据缓冲区数。

BAR_PROGRESS_FREQ 指定备份或还原进程消息在活动日志中显示的频率（以分钟计）。

BAR_XFER_BUF_SIZE 指定缓冲区的大小（以页计）。

相关链接

《*SinoDB* 备份和还原指南》: *BAR_MAX_BACKUP* 配置参数

《*SinoDB* 备份和还原指南》: *BAR_NB_XPORT_COUNT* 配置参数

《*SinoDB* 备份和还原指南》: *BAR_PROGRESS_FREQ* 配置参数

《*SinoDB 备份和还原指南*》: *BAR_XFER_BUF_SIZE 配置参数*

ontape 配置参数 (UNIX™)
在 UNIX™ 上，LTAPEBLK、LTAPEDEV、LTAPESIZE、TAPEBLK、TAPEDEV 和 TAPESIZE 是会影响 ontape 实用程序的配置参数。

在 UNIX™ 上，LTAPEBLK、LTAPEDEV 和 TAPESIZE 配置参数为使用 ontape 执行的逻辑日志备份指定块大小、设备和磁带大小。TAPEBLK 配置参数为使用 ontape、onload 和 onunload 执行的数据库备份指定块大小。

TAPEDEV 指定磁带设备。TAPESIZE 为这些备份指定磁带大小。

相关链接

《*SinoDB 备份和还原指南*》: *ON-Bar 和 ontape 配置参数和环境变量*

影响回滚和恢复的配置参数

OFF_RECVRY_THREADS、ON_RECVRY_THREADS、PLOG_OVERFLOW_PATH 和 RTO_SERVER_RESTART 配置参数会影响恢复。LOW_MEMORY_RESERVE 配置参数会保留特定内存量（以 KB 为单位），以供数据库服务器在需要关键活动（例如，回滚活动）时使用。

OFF_RECVRY_THREADS 和 ON_RECVRY_THREADS 及其对快速恢复的影响
OFF_RECVRY_THREADS 配置参数指定数据库服务器执行冷恢复或快速恢复时运行的恢复线程数。ON_RECVRY_THREADS 配置指定数据库服务器执行热恢复时运行的恢复线程数。

要提高快速恢复的性能，请使用 OFF_RECVRY_THREADS 配置参数增加恢复线程数。当启动快速恢复时，数据库服务器将创建 LGR 内存池并为每个恢复线程从池中分配将近 100 KB。当快速恢复完成时将释放 LGR 池及其内存。由于高可用性集群中的辅助服务器几乎总是处于快速恢复模式中，因此 LGR 内存池几乎总是出现在辅助服务器上。

在设置 OFF_RECVRY_THREADS 配置参数时，请遵循以下准则：

- 如果您具有足够的共享内存，请将线程数设置为频繁更新的表数或分段数。平衡线程数与共享内存量。
- 在单 CPU 主机上，请将线程数设置为 10 - 30 或 40。过多线程的开销会超出并行线程带来的优势。

热恢复与其他数据库操作同时进行。为了减少热恢复对其他用户的影响，您可以使分配给热恢复的线程少于分配给冷恢复的线程。但是，要在执行热恢复的同时重放逻辑日志事务，请在 ON_RECVRY_THREADS 配置参数中指定更多的线程。

相关链接

《*SinoDB 管理员参考*》: *OFF_RECVRY_THREADS 配置参数*
《*SinoDB 管理员参考*》: *ON_RECVRY_THREADS 配置参数*

PLOG_OVERFLOW_PATH 及其对快速恢复的影响
在快速恢复期间发生物理日志文件溢出的情况下，PLOG_OVERFLOW_PATH 配置参数会指定数据库服务器使用的磁盘文件（名为 plog_extend.servernum）的位置。

在快速恢复期间，数据库服务器在执行第一个检查点时会移除 plog_extend.servernum 文件。

相关链接

《*SinoDB 管理员参考*》: *PLOG_OVERFLOW_PATH 配置参数*

RTO_SERVER_RESTART 及其对快速恢复的影响
在您重新启动 SinoDB® 并将其置于联机或静默模式后，RTO_SERVER_RESTART 配置参数允许您使用恢复时间目标 (RTO) 标准来设置 SinoDB® 从问题中恢复所需的时间量（以秒计）。

相关链接

《*SinoDB 管理员参考*》: *RTO_SERVER_RESTART 配置参数*

LOW_MEMORY_RESERVE 配置参数和内存利用率
LOW_MEMORY_RESERVE 配置参数会保留特定内存量（以 KB 为单位），以供数据库服务器在需要关键活动且该服务器的可用内存有限时使用。

如果通过将新 LOW_MEMORY_RESERVE 配置参数设置为指定的值（以 KB 为单位）来启用该参数，那么即使接收到内存不足错误，关键活动（例如，回滚活动）仍然可以完成。

相关链接

《SinoDB 管理员参考》: LOW_MEMORY_RESERVE 配置参数
《SinoDB 管理员参考》: onstat -g seg 命令: 显示共享内存段的统计信息

影响数据复制和审计的配置参数

数据复制和审计是可选的。如果使用这些功能部件，可以设置影响数据复制性能和审计性能的配置参数。

要使性能改善立即见效，可以禁用以下功能，前提是系统的操作需求允许您这样做。

影响数据复制的配置参数
同步数据复制可能会增大在日志清空后释放日志缓冲区所需的时间量。DRINTERVAL、DRTIMEOUT 和 HDR_TXN_SCOPE 配置参数可以调整同步和系统性能。

对于辅助数据库服务器，DRINTERVAL 配置参数指示数据复制缓冲区是同步还是异步清空的。如果此参数设置为异步清空，那么将指定清空之间的时间间隔。每次清空都影响 CPU 并将数据通过网络发送到辅助数据库服务器。

如果 DRINTERVAL 配置参数设置为 0，那么会使用 HDR_TXN_SCOPE 配置参数指定的同步方式。HDR_TXN_SCOPE 配置参数指定 HDR 复制是完全同步、接近同步还是异步。

- 在完全同步方式下，事务需要在 HDR 辅助服务器上进行完成确认，才能完成。
- 在异步方式下，事务不需要在 HDR 辅助服务器上进行已接收或已完成的确认，即可完成。
- 在接近同步方式下，事务需要在 HDR 辅助服务器上进行已接收的确认，才能完成。

DRTIMEOUT 配置参数指定两个数据库服务器中的任意一个等待另一个传输确认的时间间隔。如果主数据库服务器没有接收到期望的确认，它会将事务信息添加到 DRLOSTFOUND 配置参数中指定的文件。如果辅助数据库服务器没有收到确认，它将按 DRAUTO 配置参数的指定来更改数据复制方式。

相关链接

《SinoDB 管理员参考》: DRINTERVAL 配置参数
《SinoDB 管理员参考》: DRTIMEOUT 配置参数
《SinoDB 管理员参考》: DRLOSTFOUND 配置参数
《SinoDB 管理员参考》: DRAUTO 配置参数
《SinoDB 管理员参考》: HDR_TXN_SCOPE 配置参数
《SinoDB 管理员参考》: onstat -g dri 命令: 显示高可用性数据复制信息
《SinoDB 管理员指南》: 将主服务器数据复制到辅助服务器
《SinoDB 管理员指南》: HDR 复制的完全同步模式
《SinoDB 管理员指南》: HDR 复制的接近同步模式
《SinoDB 管理员指南》: HDR 复制的异步模式

影响审计的配置参数
ADTERR 和 ADTMODE 配置参数会影响审计性能。

ADTERR 配置参数指定数据库服务器是否停止处理审计记录遇到错误的用户会话。如果将 ADTERR 设置为停止这样的会话，那么该会话的响应时间将延长，直到其中一次写入审计记录的连续尝试获得成功。

ADTMODE 配置参数根据您使用 onaudit 实用程序指定的审计记录来启用或禁用审计。记录将写入 AUDITPATH 参数指定的目录中的文件。AUDITSIZE 参数会指定每个审计记录文件的大小。

审计对性能的影响主要是由您选择记录的审计事件所决定的。根据所审计的用户和事件，这些配置参数的影响可能千差万别。

很少发生的事件（例如连接到数据库的请求）对性能的影响很小。经常发生的事件（例如读取任意行的请求）则会产生大量的审计活动。要审计此类经常发生的事件的用户越多，对性能的影响也就越大。

相关链接

《SinoDB 安全指南》: ADTERR 配置参数

《SinoDB 安全指南》: ADTMODE 配置参数

《SinoDB 安全指南》: 审计数据安全

LRU 调整

用于清空检查点之间的每个缓冲池的 LRU 设置对于检查点的性能来说并不重要。只有当为页替换保持足够的清除页时，LRU 设置才是必要的。

LRU 清空的缺省设置为 lru_min_dirty 的 50% 以及 lru_max_dirty 的 60%。

如果由于检查点的性能您已经将数据库服务器的 LRU 清空配置为更高频率，那么至少可以将 LRU 清空减小到缺省值。

如果 AUTO_LRU_TUNING 配置参数为开启状态，那么在以下情况下数据库服务器会自动调整 LRU 清空：

- 强制页替换以执行前台写入，以便查找空页。在这种情况下，针对发生前台写入的特定缓冲池，将 LRU 清空频率提高 5%。
- 强制页替换以使用标记为高优先级（意味着访问较为频繁）的缓冲区。在这种情况下，针对使用高优先级缓冲区进行页替换的特定缓冲池，将 LRU 清空频率提高 1%。
- 当 RTO_SERVER_RESTART 配置参数为开启状态，并且清空该缓冲池花费的时间比客观恢复的时间更长时，将所有缓冲池的 LRU 清空频率提高 10%。

出现检查点后，如果页替换在前一个检查点时间间隔内执行了前台写入，那么数据库服务器会将 LRU 设置增加 5%，并在每个后续检查点处持续提高 LRU 清空频率，直至前台写入停止或指定缓冲池的 lru_max_dirty 降低到 10% 以下。例如，如果页替换执行前台写入，且缓冲池的 LRU 设置为 80 和 90，那么数据库服务器会将其调整为 76 和 85.5。

除了前台写入，当缺页故障替换高优先级缓冲区以及非高优先级缓冲区在更改的 LRU 队列中时，LRU 清空将被调整为更高频率。自动 LRU 调整只会使 LRU 清空更频繁；它们不会减少 LRU 清空。自动 LRU 调整不是永久的并且不记录在 ONCONFIG 文件中。

LRU 清空将复位成启动数据库服务器的 ONCONFIG 文件中的值。

AUTO_LRU_TUNING 配置参数指定服务器启动时启用还是禁用自动 LRU 调整。

相关链接

自动检查点、*LRU* 调整以及 *AIO* 虚拟处理器调整 在第104页

《SinoDB 管理员参考》: AUTO_LRU_TUNING 配置参数

《SinoDB 管理员参考》: RTO_SERVER_RESTART 配置参数

第 6 章

表性能的注意事项

一些性能问题与未分段表和表分段相关联。

问题包括：

- 将表放置于磁盘以增大吞吐量并减少争用
- 对表、BLOB 页、智能大对象空间和扩展数据块的空间估计
- 对表所作的添加或删除历史数据的更改
- 使数据库反向规范化以减少开销

将表置于磁盘中

数据库服务器所支持的表驻留在一个或多个磁盘的一个或多个分区中。在创建表时，通过将表分配给数据库空间来控制表在磁盘上的存储位置。

数据库服务器所支持的表驻留在一个或多个磁盘的一个或多个分区中。在通过将表分配给数据库空间来创建表时，控制表在磁盘上的存储位置。数据库空间由一个或多个块组成。每个块对应整个或部分磁盘分区。将块分配给数据库空间后，这些块中的磁盘空间就可用来存储表或表分段。

配置块并将其分配给数据库空间时，必须将数据库空间的大小与每个数据库空间要包含的表或分段联系起来。要估算表的大小，请遵循估算表大小 在第118页中的指示信息。

数据库管理员 (DBA) 负责创建表，并通过以下其中一种方法将该表分配给数据库空间：

- 通过使用 CREATE TABLE 语句的 IN DBSPACE 子句
- 通过使用当前数据库的数据库空间

  发出 CREATE TABLE 语句之前，由 DBA 发出的最新 DATABASE 或 CONNECT 语句设置当前数据库。

DBA 可以在多个数据库空间对表进行分段（如规划分段存储策略 在第188页中所述），或者使用 ALTER FRAGMENT 语句将表移动到另一个数据库空间。对于变更表的存储位置，ALTER FRAGMENT 语句提供了最简单的方法。然而，在数据库服务器处理变更时，该表不可用。每次调度表或分段的移动时，应使受影响的用户达到最少。

对于将表在数据库空间之间进行移动，还有其他方法：

- 可以使用 SQL 语句 LOAD 和 UNLOAD 或 onload 和 onunload 实用程序从表中卸载数据，然后将该数据移动到另一个数据库空间中。
- 可以将数据载入外部表以及从外部表卸载数据。

使用 LOAD 和 UNLOAD 或 onload 和 onunload 在数据库之间移动表会包含以下周期：数据从表中复制到磁带，然后重新载入系统。这些周期中会呈现窗口漏洞，在此期间某个表可能会变得与数据库的剩余部分不一致。为了防止表变得不一致，在发生数据传送时，必须限制对磁盘上保留的版本的访问。

根据大小、分段存储策略和与表相关联的索引，卸载然后重新载入表可能会比改变分段存储要快。而对于其他表，可能是改变分段存储更快。对于要移动或者重新分区的表，您可进行试验，以确定哪种方法更快。

相关链接

《*SinoDB SQL 指南: 语法*》: *ALTER FRAGMENT* 语句
《*SinoDB SQL 指南: 语法*》: *LOAD* 语句
《*SinoDB SQL 指南: 语法*》: *UNLOAD* 语句
《*SinoDB 迁移指南*》: *onunload* 和 *onload* 实用程序
《*SinoDB 管理员指南*》: 使用外部表移动数据
《*SinoDB SQL 指南: 语法*》: *CREATE EXTERNAL TABLE* 语句

隔离高使用率的表

您可以将 I/O 活动频繁的表放置在专用磁盘设备上。这样就减少了对存储于该表中的数据的争用。

如果磁盘驱动器的性能级别不同，可以将使用率最高的表置于最快的驱动器上。将两个使用率高的表分开放在不同的磁盘设备上，在多个应用程序同时对这两个表进行频繁的 I/O 访问时，或者在两表之间形成连接时，这样可以减少对磁盘存取的争用。

要将高使用率的表隔离在其自己的磁盘设备上，可以将此设备分配给块，并将此块分配给数据库空间，然后把此表置于您创建的数据库空间中。图 *17:* 隔离高使用率的表 在第117页显示三个使用率高的表，每个表放在三个磁盘中单独的数据库空间中。

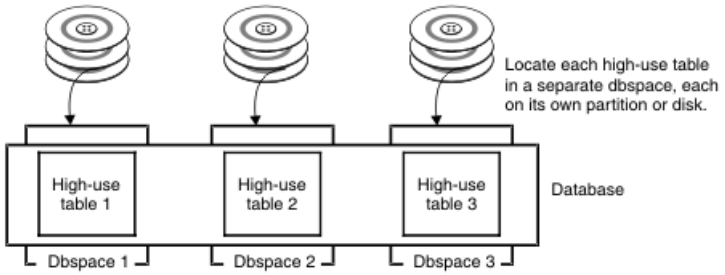

图 17: 隔离高使用率的表

将高使用率的表置于磁盘的中间分区中

要使磁盘头移动尽可能少，可将访问最频繁的数据置于接近磁盘中部（不靠近中心，也不靠近边缘）的分区中。这种方法可以使找到频繁访问的表中的数据而产生的磁盘头移动最少。

下图显示将访问最频繁的数据置于接近磁盘中部的分区中。

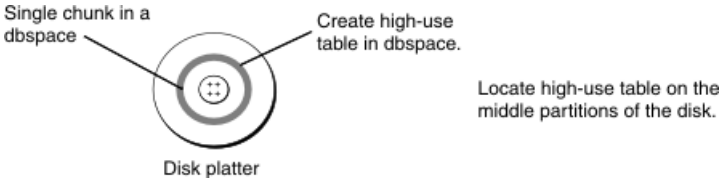

图 18: 高使用率的表位于中间分区的磁盘盘片

要将高使用率的表置于磁盘的中间分区，请创建裸设备，该裸设备由驻留在磁盘的轴和外边缘之间中间地带的柱面组成。（有关如何创建裸设备的指示信息，请参阅针对您的操作系统的《*SinoDB*® 管理员指南》。）分配块，使其与此裸设备相关联，如《*SinoDB*® 管理员参考》中所述。然后使用与初始且唯一的块相同的块来创建数据库空间。创建高使用率的表时，请将表置于此数据库空间中。

使用多个磁盘

可以将多个磁盘用于数据库空间、逻辑日志、临时表和排序文件。

将多个磁盘用于数据库空间

将多个磁盘用于数据库空间以助于将 I/O 分布在包含若干小表的数据库空间上。

数据库空间可以包括多个块，每个块均可以代表一个不同的磁盘。块的最大值为 4 TB。这种分配使您可以将数据库空间中的数据分布在多个磁盘上。图 *19:* 分布在三个磁盘上的数据库空间 在第118页显示分布在三个磁盘上的数据库空间。

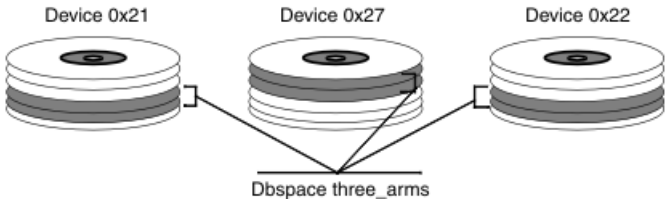

图 19: 分布在三个磁盘上的数据库空间

由于不能将此类型的分布式数据库空间用于并行数据库查询 (PDQ)，您应该使用分布方案 在第191页中描述的表分段存储技术将高使用率的大型表分开放于多个数据库空间中。

将多个磁盘用于逻辑日志
您可以通过循环方式，将逻辑日志分布在多个磁盘上的不同数据库空间中，从而提高逻辑备份的性能。此方案使得数据库服务器可以在一个磁盘上进行日志备份，而在其他磁盘上执行日志记录操作。

将逻辑日志和物理日志保存在单独的设备上，通过减少单个设备上的 I/O 争用来提高性能。在初始化数据库服务器时，逻辑日志和物理日志会在根数据库空间中创建。初始化后，您就可以将其移动到其他数据库空间。

将临时表和排序文件分布在多个磁盘中
在为临时表和排序文件定义数据库空间后，可以将与临时表和排序文件关联的 I/O 分布在多个磁盘上。对于需要大量临时空间用于临时表或大排序操作的应用程序，这样做可以改善性能。

要为临时表和排序文件定义数个数据库空间，可使用 onspaces -t。将这些数据库空间放在不同的磁盘上并在 DBSPACETEMP 配置参数中将其列出，使与临时表和排序文件关联的 I/O 分布在多个磁盘上，如图 *20: 临时表和排序文件的数据库空间* 在第118页所示。您可以将包含常规表的数据库空间列入 DBSPACETEMP。

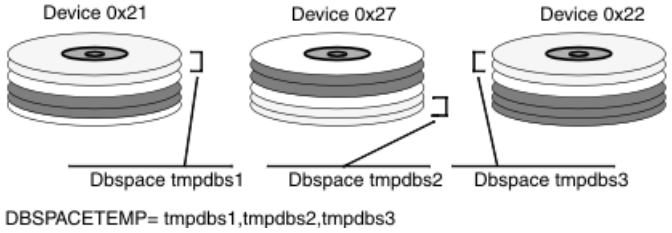

图 20: 临时表和排序文件的数据库空间

用户可以用 DBSPACETEMP 环境变量来指定其用于临时表和排序文件的数据库空间的列表。有关详细信息，请参阅为临时表和排序文件配置数据库空间 在第90页。

将表置于磁盘上时备份和还原注意事项

决定在何处放置表或分段时，请记住如果包含数据库空间的设备发生故障，那么即使其他数据库空间中的表和分段可以访问，仍会造成该数据库空间中的所有表或表分段均不可访问。在发生磁盘故障时，想要限制数据的不可用性，可能会影响您在特定的数据库空间中决定将哪些表分在同一组。

虽然在包含关键性数据的数据库空间发生故障时必须执行冷恢复，但是如果非关键性的数据库空间发生故障，那么只需要执行热恢复即可。要使冷恢复的影响减少到最小，可能会影响用于存储关键性数据的数据库空间。

影响未分段表和表分段性能的因素

很多因素会影响单个表或表分段的性能。这些因素包括表或分段的位置、表或分段的大小、所用的索引策略、各个表扩展数据块的大小和位置以及对表的访问频率。

估算表大小

可以计算表大小的近似值（在磁盘页中）。

有关对索引大小计算的描述，请参阅估算索引页 在第153页。

分配给表的磁盘页称为表空间。表空间包括数据页。单独的表空间包括索引页。如果简单大对象（TEXT 或 BYTE 数据）与不在备用数据库空间中存储的表关联，那么保存简单大对象的页也包含于表空间中。

表空间不与数据库空间中的任何固定区域相对应。组成表的扩展数据块和索引可以分散在整个数据库空间中。

表的大小包括表空间中的所有页：数据页和存储简单大对象的页。存储在单独的 BLOB 空间中的 BLOB 页不包含在表空间中，并且不计为表大小的一部分。

后面的部分将描述如何估算表空间中每一类型页的页数。

> 提示： 如果存在合适的样本表，或者您可以使用模拟数据构建现实大小的样本表，那么无需进行估算。可以执行 oncheck -pt 来获取准确数字。

估算数据页数

如何估算表的数据页数，取决于表包含的是长度固定的行还是长度可变的行。

估算具有固定长度行的表

可以估算具有固定长度行的表的大小（以页数计）。具有固定长度行的表没有 VARCHAR 或 NVARCHAR 数据类型的任何列。

执行以下步骤来估算具有固定长度行的表大小（以页数计）。

要估算页大小、行大小、行数和数据页数：

1. 使用 onstat -b 获取页大小。

   此输出的最后一行中的 buffer size 字段会显示页大小。
2. 将该值减去 28，可以得出每个数据页中的页头。

   得出的量为 *pageuse*。
3. 要计算行的大小，可将表定义中所有列的宽度相加。TEXT 和 BYTE 列各使用 56 字节。

   如果已经创建了表，可以使用以下 SQL 语句获得行的大小：

   ```
   SELECT rowsize FROM systables WHERE tabname =
      'table-name';
   ```
4. 估算表格预计包含的行数。

   此值称为 *rows*。根据行大小是小于还是大于 *pageuse*，计算表所需的数据页数的过程也会有所不同。
5. 如果行的大小小于或等于 *pageuse*，那么使用以下公式计算数据页数。

   trunc() 函数符号表示要向下舍入到最接近的整数。

   ```
   data_pages = rows / trunc(pageuse/(rowsize + 4))
   ```

   每页的最大行数是 255，该值与行的大小无关。

   > 重要： 虽然数据库服务器接受的行的最大值约为 32 KB，但如果行的大小超过页的大小，性能就会降低。有关分解宽表以提高性能的信息，请参阅对数据模型进行反向规范化以提高性能 在第147页。
6. 如果行的大小大于 *pageuse*，那么数据库服务器会将该行在各页之间进行拆分。

   包含行的初始部分的页称为主页。包含行的后续部分的页称为余页。如果一行跨两页以上，那么某些余页就会完全由该行的数据填充。如果行的尾部未用完一页，那么可以和其他行的尾部结合一起填充该部分余页。数据页的数量是主页、完整余页和部分余页的总数。

   1. 计算主页的数量。

      主页的数量和行数相等：

      ```
      homepages = rows
      ```

2. 计算完整余页的数量。

   首先使用以下公式计算剩余的行大小：

   ```
   remsize = rowsize - (pageuse + 8)
   ```

   如果 *remsize* 小于 *pageuse* - 4 ，那么没有完整余页。

   如果 *remsize* 大于 *pageuse* - 4 ，在以下公式中使用 *remsize* 以获得完整余页的数量：

   ```
   fullrempages = rows * trunc(remsize/(pageuse - 8))
   ```

3. 计算部分余页的数量。

   首先，在减去单行的主页和完整余页后，计算行的剩余部分的大小。在以下公式中，remainder() 函数符号表示除法后取余：

   ```
   partremsize = remainder(rowsize/(pageuse - 8)) + 4
   ```

   数据库服务器使用与页大小相关的特定大小阈值，以确定部分余页的使用数量。使用以下公式计算部分余页与页的比率：

   ```
   partratio = partremsize/pageuse
   ```

   使用下表中的相应公式来计算部分余页的数量。

partratio 值	计算部分余页数的公式
小于 .1	partrempages = *rows*/(trunc((*pageuse*/10)/*remsize*) + 1)
小于 .33	partrempages = *rows*/(trunc((*pageuse*/3)/*remsize*) + 1)
大于或等于 .33	partrempages = *rows*

4. 使用以下公式合计页数：

   ```
   tablesize = homepages + fullrempages + partrempages
   ```

估算具有可变长度行的表
可以估算带有 VARCHAR 或 NVARCHAR 数据类型列的可变长度行的表的大小。

当表包含一个或多个 VARCHAR 或 NVARCHAR 列时，它的行长度可变。这些可变长度给计算带来了不确定性。必须根据您对数据的理解确定一个估算每个 VARCHAR 列的典型大小的公式，并在估算表的大小时使用该公式计算的值。

> 重要： 当数据库服务器为可变大小的行分配空间时，如果可用空间无法容纳具有最大大小的附加行，那么认为页已满。

要估算具有可变长度行的表的大小，必须根据对数据的理解进行以下估算，并在其中选择一个值：

- 表的最大大小，根据所有 VARCHAR 或 NVARCHAR 列允许的最大宽度进行计算
- 表的预期大小，根据每个 VARCHAR 或 NVARCHAR 列的典型宽度进行计算

要估算数据页的最大数量：

1. 要计算 *rowsize*，应将所有列宽度的最大值加在一起。
2. 将此值用作 *rowsize*，并按照估算具有固定长度行的表 在第119页中所述进行计算。生成的值称为 *maxsize*。

要估算数据页的预计数量：

1. 要计算 *rowsize*，应将每个宽度可变的列的典型宽度加在一起。建议您使用列中最频繁出现的宽度作为该列的典型宽度。如果您无权访问数据，或不需要使用这些宽度制成表格，您可以选择使用最大宽度的一定比例，如 2/3 (.67)。
2. 将此值用作 *rowsize*，并按照估算具有固定长度行的表 在第119页中所述进行计算。生成的值称为 *projsize*。

选择中间值作为表的大小
实际表大小应该是预期数据页数 (*projsize*) 与最大数据页数 (*maxsize*) 之间的某个值。

根据您对数据的理解，在此范围中选择一个您认为最合理的值。您对数据越不熟悉，就越应该选择更保守（更高）的估算值。

估算简单大对象占用的页数

可以估算所有简单大对象的总页数，或者可以根据简单大对象的中值大小估算页数。

BLOB 页可以驻留在该表所在的数据库空间中，也可以驻留在 BLOB 空间中。有关何时使用 BLOB 空间的更多信息，请参阅将简单大对象存储在表空间或单独的 *BLOB* 空间中 在第122页。

用于估算 BLOB 页的以下方法会产生比较保守（高）的估算值，因为单个 TEXT 或 BYTE 列不需要占用表空间中的整个 BLOB 页。换言之，表空间中的 BLOB 页可以包含多个 TEXT 或 BYTE 列。

要估算 BLOB 页数：

1. 用 onstat -b 获取页大小。
2. 用下列公式来计算 BLOB 页的可用部分：

```
bpuse = pagesize - 32
```

3. 对于 blobsize *n* 的每个字节，使用以下公式计算该字节占用的页数 (*bpages_n*)：

```
bpages1 = ceiling(bytesize1/bpuse)
bpages2 = ceiling(bytesize2/bpuse)
...
bpages_n = ceiling(bytesize_n/bpuse)
```

ceiling() 函数表示应该向上舍入到最接近的整数值。

4. 求出所有简单大对象的页数总和，如下所示：

```
blobpages = bpages1 + bpages2 + ... + bpagesn
```

或者，您可以根据简单大对象（TEXT 或 BYTE 数据）的中值大小（即，出现最频繁的简单大对象数据大小）进行估算。这种方法精确度较低，但更容易计算。

要根据简单大对象的中值大小估算 BLOB 页的数量：

1. 计算中值大小的简单大对象所需的页数，如下所示：

```
mpages = ceiling(mblobsize/bpuse)
```

2. 将该值乘以简单大对象的总数，如下所示：

```
blobpages = blobcount * mpages
```

将简单大对象存储在表空间或单独的 BLOB 空间中
在磁盘上创建简单大对象列时，您可以选择将列数据存储在表空间或单独的 BLOB 空间中。通常，可以通过将简单大对象数据存储在单独 BLOB 空间中，并将智能大对象和用户定义数据存储在智能大对象空间中来提高性能。

在以下示例中，TEXT 值存储在表空间中，而 BYTE 值存储在名为 rasters 的 BLOB 空间中：

```
CREATE TABLE examptab
   (
   pic_id SERIAL,
   pic_desc TEXT IN TABLE,
   pic_raster BYTE IN rasters
   )
```

有关在单独的 BLOB 空间中存储简单大对象数据的信息，请参阅估算简单大对象占用的页数 在第121页。

TEXT 或 BYTE 值始终与表的行分开存储；只有 56 字节的描述符与行存储在一起。然而，简单大对象至少要占用一个磁盘页。该描述符所指向的简单大对象可以与表行（同一表空间中的）驻留在磁盘上的同一组扩展数据块中，也可以驻留在单独的 BLOB 空间中。

当简单大对象存储于表空间中时，其数据页散布在许多包含行的页上，从而显著增加了表的大小。当数据库服务器只读取行而不读取简单大对象时，磁盘臂必须比分开存储 BLOB 页时移动得更远。在以下情况中，数据库服务器只扫描行页：

- 数据库服务器执行任何不检索简单大对象列的 SELECT 操作时
- 数据库服务器使用过滤器表达式测试行时

另一个注意事项是，往来于数据库空间的磁盘 I/O 在数据库服务器的共享内存中进行缓冲。页会被存储以备不时之需，并且在写入页时，在发生实际的磁盘写操作之前，请求程序可以继续处理。但是，由于 BLOB 空间数据预期会很大，因此往来于 BLOB 空间的磁盘 I/O 不进行缓冲，并且直至所有的输出均写入 BLOB 空间后，才允许请求程序继续进行。

为了获得最佳性能，在以下任何一种情况下均将简单大对象列存储在 BLOB 空间中：

- 单个数据项每个都大于一页或两页
- TEXT 或 BYTE 数据的页数相当于行数据页数的一半以上

估算简单大对象的表空间页数
在估算表所需的空间时，应包括要存储在该表空间中的所有简单大对象的 BLOB 页。对于相对较小且固定的表，可以通过分隔行页和 BLOB 页来获得专用 BLOB 空间的效果。

要在数据库空间中分隔行页与 BLOB 页：

1. 载入整个表，表中含有简单大对象列为 null 的行。
2. 创建所有索引。
   现在，行页和索引页是连接的。
3. 更新所有行，以安装简单大对象。
   现在，在表空间中，BLOB 页出现在行页和索引数据之后。

管理表空间 tblspace 的第一个和下一个扩展数据块的大小

表空间 tblspace 是一些页的集合，这些页描述数据库空间中的所有表空间的位置和结构。每个数据库空间都有一个表空间 tblspace。当您创建数据库空间时，可使用 TBLTBLFIRST 和 TBLTBLNEXT 配置参数来为根数据库空间中的表空间 tblspace 指定第一个和下一个扩展数据块大小。

可使用 onspaces 实用程序来为非根数据库空间中的表空间 tblspace 指定第一个和下一个扩展数据块大小。

如果您希望减少表空间 tblspace 扩展数据块的数量，并且减少需要将表空间 tblspace 扩展数据块置于非主块时的发生频率，那么请指定第一个和下一个扩展数据块大小。

将第一个扩展数据块大小指定为大于缺省值的功能为管理空间提供了灵活性。创建扩展数据块时，可在创建数据库空间期间保留空间，这样可以降低需要在不是初始块的块中创建附加扩展数据块的风险。

您只能在创建数据库空间时指定第一个和下一个扩展数据块的大小。创建数据库空间之后，您将无法改变第一个和下一个扩展数据块大小的规格。另外，您还不能指定临时数据库空间、智能大对象空间、BLOB 空间或外部空间的扩展数据块大小。

如果您未指定表空间 tblspace 的第一个和下一个扩展数据块的大小，那么 SinoDB® 将使用现有的缺省扩展数据块大小。

相关链接

《*SinoDB* 管理员参考》：*TBLTBLFIRST* 配置参数

《*SinoDB* 管理员参考》：*TBLTBLNEXT* 配置参数

《*SinoDB* 管理员指南》：为表空间 *tblspace* 指定第一个和下一个扩展数据块大小

管理智能大对象空间

智能大对象空间是由存储智能大对象的一个或多个块组成的逻辑存储单元。您可以估算智能大对象所需的存储量，改善元数据 I/O，监视智能大对象空间以及更改存储特征。

估算智能大对象占用的页数

在估算表所需的空间时，您也应考虑任何智能大对象（例如，CLOB、BLOB 或多个代表性的数据类型）的智能大对象空间存储量，这些智能大对象为表的组成部分。智能大对象空间包含用户数据区域和元数据区域。

CLOB 和 BLOB 数据存储于驻留在用户数据区域的智能大对象页中。元数据区域包含智能大对象属性，如平均大小和该智能大对象是否进行日志记录。有关智能大对象空间的更多信息，请参阅《*SinoDB*® 管理员指南》。

估算智能大对象空间和元数据区域的大小

智能大对象空间的第一个块必须有元数据区域。添加智能大对象时，数据库服务器会给该元数据区域添加更多的控制信息。

如果在初始分配之后给该智能大对象空间添加块，那么可以对元数据空间执行以下操作之一：

- 缺省情况下，在新的块上分配另一个元数据区域。

  此操作具有以下优点：

  - 此操作更简单，因为数据库服务器会根据智能大对象的平均大小，自动计算并给添加的块上分配新的元数据区域
  - 将 I/O 操作分布在多个磁盘的元数据区域上
- 使用现有的元数据区域

  如果指定 onspaces -U 选项，那么数据库服务器不会在新的块中分配元数据空间。而改为必须在其他块的其中一个中使用元数据区域。

另外，数据库服务器会保留 40% 的用户区域以在元数据区域空间耗尽的情况下使用。因此，如果分配的元数据已满，数据库服务器将开始使用用户区域中为附加的控制信息所保留的空间。

可以让数据库服务器为您计算在初始块和每一个添加的块上的元数据区域大小。不过，您可能需要明确指定元数据区域的大小，以确保智能大对象空间不会耗尽元数据空间和 40% 的保留区域。可以使用以下其中一种方法来明确指定要分配的元数据空间量：

- 指定 onspaces -Df 选项上的 AVG_LO_SIZE 标签。

  在不指定 -Ms 选项的情况下，数据库服务器会使用该值来计算要分配的元数据区域大小。如果不指定 AVG_LO_SIZE，数据库服务器将使用缺省值 8 KB来计算元数据区域的大小。
- 在 onspaces 实用程序的 -Ms 选项中指定元数据区域的大小。

  使用手动调整新块的元数据区域大小 在第124页描述的过程来估算在 onspaces -Ms 选项中指定的值。

手动调整新块的元数据区域大小
每个块均可以包含元数据，但总量必须具有足够的空间以容纳所有的 LO 头（每个平均长度为 570 字节）和块可用列表（列出块中所有可用的扩展数据块）。

以下过程假设您知道智能大对象空间大小，并需要分配更多的元数据空间。

要手动估算新块的元数据区域大小：

1. 使用 onstat -d 选项可从 Metadata size 字段获取当前元数据区域的大小。
2. 估算预期驻留在智能大对象空间中的智能大对象数量及其平均大小。
3. 使用以下公式计算元数据区域的总大小：

```
元数据总千字节数 = (LOcount*570)/1024 +
                          (numchunks*800) + 100
```

LOcount

为预期在所有智能大对象空间块（包括新的）中拥有的智能大对象数。

numchunks

为智能大对象空间中块的总数。

4. 要获取元数据的额外必需区域，应从步骤 3 中获取的值中减去步骤 1 中获取的当前元数据大小。
5. 添加另一个块时，在 onspaces -a 命令的 -Ms 选项中指定您在步骤 4 中获取的值。

计算新块的元数据区域的示例
本主题包含一个显示如何估算两个智能大对象空间块所需元数据大小的示例。

假设 onstat -d 选项中的 Metadata size 字段显示当前元数据区域为 1000 页。如果系统页大小为 2048 字节，那么此元数据区域大小为 2000 KB，如以下公式所示：

```
当前元数据大小 = (metadata_size * pagesize) / 1024
               = (1000 * 2048) / 1024
             = 2000 KB
```

假设预期在这两个智能大对象空间块中有 31,000 个智能大对象。以下公式将计算两个块必需的元数据区域的总大小，小数部分向上舍入：

```
元数据总大小 = (LOcount*570)/1024 + (numchunks*800) + 100
             = (31,000 * 570)/1024 + (2*800) + 100
             = 17256 + 1600 + 100
           = 18956 KB
```

要获取元数据所需的额外区域：

1. 从元数据总值减去当前元数据大小。

```
额外元数据大小 = 元数据总大小 - 当前元数据大小
                   = 18956 - 2000
                 = 16956 KB
```

2. 将该块添加到智能大对象空间时，使用 onspaces -a 命令的 -Ms 选项来指定 16,956 KB 的元数据。

```
% onspaces -a sbchk2 -p /dev/raw_dev1 -o 200 -Ms 16956
```

改善智能大对象的元数据 I/O

智能大对象空间中的元数据页包含了有关智能大对象空间中的智能大对象位置的信息。通常，对这些页的读取会很密集。可以通过将其重新分发来改善元数据 I/O。

您可以使用以下一种方法来为这些页分发 I/O：

- 建立包含元数据的块的镜像。

有关镜像的意义的更多信息，请参阅考虑对关键数据组件使用镜像 在第88页。

- 将元数据页置于磁盘最快的部分。

  因为元数据页是智能大对象空间中读取最密集的部分，所以将元数据页存放于磁盘中部可以使磁盘搜寻时间最小。要将元数据页放置于特定位置，可在创建智能大对象空间或用 onspaces 实用程序添加块时，使用 -Mo 选项。
- 将元数据页散布在多个磁盘上。

  要将元数据页散布在多个磁盘上，可在智能大对象空间中创建多个块，每个块驻留在单独的磁盘上。 使用 onspaces 实用程序向智能大对象空间添加块时，可指定 -Ms 选项来为元数据信息分配页。

  尽管数据库服务器试图将元数据信息及其对应数据保存在同一个块中，但不能保证数据库服务器一定会做到这一点。
- 减少每个智能大对象占用的扩展数据块数。

  当智能大对象跨多个扩展数据块时，元数据区域会为每个扩展数据块包含单独的描述符。要减少为每个智能大对象必须读取的描述符条目数，请在创建智能大对象时指定智能大对象最终大小的期望值。

  如果在以下其中一个函数中指定最终大小，那么数据库服务器会将智能大对象作为单个扩展数据块进行分配（如果它在此块中有连续的存储空间）：

  - DataBlade® API mi_lo_specset_estbytes 函数
  - SinoDB® ESQL/C ifx_lo_specset_estbytes 函数

  有关打开智能大对象和设置估算字节数的函数的更多信息，请参阅《*SinoDB® ESQL/C* 程序员指南》和《*SinoDB® DataBlade® API* 程序员指南》。

  有关调整扩展数据块大小的更多信息，请参阅智能大对象空间扩展数据块 在第99页。

  重要： 为获得最高的数据可用性，可为所有包含元数据的智能大对象空间块建立镜像。

监视智能大对象空间

可以监视对智能大对象的 I/O 操作的效果。为获取更好的 I/O 性能，应将所有智能大对象分配到连续的扩展数据块中。

有关调整扩展数据块大小的更多信息，请参阅智能大对象空间扩展数据块 在第99页。

连续性提供以下 I/O 性能方面的好处：

- 使磁盘臂动作最少
- 读取智能大对象时，要求的 I/O 操作较少
- 进行大的顺序读取时，可以利用轻量级 I/O，它可以在单个 I/O 操作中读取更大的数据块（60 KB或者更多，这取决于您的平台）

您可以使用以下命令行实用程序来监视对智能大对象的 I/O 操作的效果：

- oncheck -cS, -pe 和 -pS
- onstat -g smb s 选项

以下部分将描述如何使用这些实用程序选项来监视智能大对象空间。

使用 oncheck -cS 监视智能大对象空间

oncheck -cS 选项检查用户数据区域中的智能大对象扩展数据块和智能大对象空间分区。

图 *21: oncheck -cS* 输出 在第126页显示了 s9_sbspc 的 -cS 选项的输出示例。

Sbs#、Chk# 和 Seq# 列中的值对应于 -pS 输出中 Space Chunk Page 的值。 Bytes 和 Pages 列分别以字节数和页数显示了每个智能大对象的大小。

要计算智能大对象的平均大小，可以对 Size (Bytes) 列中的数目进行汇总，然后除以智能大对象的数目。在 图 *21: oncheck -cS* 输出 在第126页中，分配的平均字节数是 2690，如以下计算所示：

```
Average size in bytes = (15736 + 98 + 97 + 62 + 87 + 56) / 6
```

```
                    = 16136 / 6
                    = 2689.3
```

有关如何指定智能大对象大小以影响扩展数据块大小的信息，请参阅智能大对象空间扩展数据块 在第99页。

```
Validating space 's9_sbspc' ...

Large Objects
  ID                Ref Size       Allocced       Creat   Last
  Sbs# Chk#  Seq#  Cnt    (Bytes)    Pages Extns Flags   Modified
  ---- ---- ----- ---- ---------- -------- ----- ----- ------------------------
     2    2     1    1      15736        8     1 N-N-H Thu Jun 21 16:59:12 2007
     2    2     2    1         98        1     1 N-K-H Thu Jun 21 16:59:12 2007
     2    2     3    1         97        1     1 N-K-H Thu Jun 21 16:59:12 2007
     2    2     4    1         62        1     1 N-K-H Thu Jun 21 16:59:12 2007
     2    2     5    1         87        1     1 N-K-H Thu Jun 21 16:59:12 2007
     2    2     6    1         56        1     1 N-K-H Thu Jun 21 16:59:12 2007
```

图 21: oncheck -cS 输出

Extns 字段以页数显示了分配给每个智能大对象的最小扩展数据块大小。

使用 oncheck -pe 监视智能大对象空间

oncheck -pe 选项显示的信息包括块的大小（以页数表示）、所用页数、可用页数、块中所有表的列表以及表的初始页编号和表的长度（以页数表示）。此选项也会显示智能大对象是否占用智能大对象空间内的连续空间。

执行 oncheck -pe 以显示以下信息，从而确定智能大对象是否占用智能大对象空间中的连续空间：

- 使用 SBLOBSpace LO 标识每个智能大对象

  SBLOBSpace LO 后面括号中的三个值分别对应 -cS 输出中的 Sbs#、Chk# 和 Seq# 列。
- 每个智能大对象的偏移量
- 每个智能大对象使用的磁盘页数（非智能大对象页数）

  提示： oncheck -pe 选项以数据库服务器页的形式（而非智能大对象页的形式）提供了有关智能大对象空间的使用情况的信息。

图 22: 显示邻接空间使用情况的 oncheck -pe 输出 在第126页显示输出的示例。在此示例中，size 字段显示第一个智能大对象占用了 8 页。因为 offset 字段显示第一个智能大对象开始于第 53 页，且第二个智能大对象开始于第 61 页，所以第一个智能大对象占用的是邻接的页。

```
Chunk Pathname                                      Size      Used      Free
                                                    1000       940        60

    Description                                       Offset     Size
    ------------------------------------------------- -------- --------
    RESERVED PAGES                                           0        2
    CHUNK FREELIST PAGE                                      2        1
    s9_sbspc:'informix'.TBLSpace                             3       50
    SBLOBSpace LO [2,2,1]                                   53        8
    SBLOBSpace LO [2,2,2]                                   61        1
    SBLOBSpace LO [2,2,3]                                   62        1
    SBLOBSpace LO [2,2,4]                                   63        1
    SBLOBSpace LO [2,2,5]                                   64        1
    SBLOBSpace LO [2,2,6]                                   65        1
...
```

图 22: 显示邻接空间使用情况的 oncheck -pe 输出

使用 oncheck -pS 监视智能大对象空间
oncheck -pS 选项显示有关智能大对象空间分区中智能大对象扩展数据块和元数据区域的信息。如果不在命令行上指定智能大对象空间名称，那么 oncheck 会检查和显示所有智能大对象空间的元数据。

图 23: oncheck -pS 输出 在第127页显示了 s9_sbspc 的 -pS 输出的示例。

要显示有关智能大对象的信息，可执行以下命令：

```
oncheck -pS spacename
```

oncheck -pS 输出显示了智能大对象空间中每个智能大对象的以下信息：

- 空间块页
- 以字节数表示的每个智能大对象的大小
- DataBlade® API 和 SinoDB® ESQL/C 函数使用的对象标识符
- 每个智能大对象的存储特征

使用 onspaces -c -S 创建智能大对象空间时，可以使用 -Df 选项来为智能大对象指定各种存储特征。创建智能大对象空间后，可以使用 onspaces -ch 来更改属性。oncheck -pS 输出中的 Create Flags 字段显示了每个智能大对象的这些存储特征和其他属性。在 图 23: oncheck -pS 输出 在第127页中，Create Flags 字段显示 LO_LOG，因为在 -Df 选项中，LOGGING 标签设置为 ON。

```
Space Chunk Page = [2,2,2] Object ID = 987122917
  LO SW Version             4
  LO Object Version         1
  Created by Txid           7
  Flags               0x31 LO_LOG LO_NOKEEP_LASTACCESS_TIME LO_HIGH_INTEG
  Data Type                 0
  Extent Size               -1
  IO Size                   0
  Created                   Thu Apr 12 17:48:35 2007
  Last Time Modified        Thu Apr 12 17:48:43 2007
  Last Time Accessed        Thu Apr 12 17:48:43 2007
  Last Time Attributes Modified Thu Apr 12 17:48:43 2007
  Ref Count                 1
  Create Flags        0x31 LO_LOG LO_NOKEEP_LASTACCESS_TIME LO_HIGH_INTEG
  Status Flags              0x0 LO_FROM_SERVER
  Size (Bytes)              2048
  Size Limit                -1
  Total Estimated Size      -1
  Deleting TxId             -1
  LO Map Size               200
  LO Map Last Row           -1
  LO Map Extents            2
  LO Map User Pages         2
```

图 23: oncheck -pS 输出

使用 onstat -g smb 监视智能大对象空间
onstat -g smb s 选项显示智能大对象空间属性。

使用 onstat -g smb s 选项可显示以下特征，这些特征将影响每个智能大对象空间的 I/O 性能：

- 日志记录状态

  如果应用程序正在更新临时智能大对象，那么不要求记录日志。您可以关闭日志记录以减少逻辑日志的 I/O 活动量、CPU 利用率和内存资源。
- 平均智能大对象大小

  平均大小和扩展数据块大小应该相近，以减小在整个智能大对象中进行读取时所需的 I/O 操作数。avg s/kb 输出字段以 KB 为单位显示了平均智能大对象大小。在 图 24: onstat -g smb s 输出 在第128页中，avg s/kb 输出字段显示该值为 30 KB。

在任一以下函数中指定智能大对象的最终大小，以将该对象作为单个扩展数据块进行分配：

- DataBlade® API mi_lo_specset_estbytes 函数
- SinoDB® ESQL/C ifx_lo_specset_estbytes 函数

有关打开智能大对象和设置估算字节数的函数的更多信息，请参阅《*SinoDB® ESQL/C* 程序员指南》和《*SinoDB® DataBlade® API* 程序员指南》。

- 第一个扩展数据块大小、下一个扩展数据块大小和最小扩展数据块大小

  如果在 onspaces 的 -Df 选项中设置扩展数据块标签，那么 1st sz/p、nxt sz/p 和 min sz/p 输出字段会显示这些扩展数据块大小。在 图 *24: onstat -g smb s* 输出 在第128页中，这些输出字段显示值 0 和 -1，因为在 onspaces 中没有设置这些标签。

```
sbnum 7    address 2afae48
  Space    : flags    nchk owner    sbname
             -------- 1    informix client
  Defaults : LO_LOG LO_KEEP_LASTACCESS_TIME

  LO       : ud b/pg flags    flags    avg s/kb max lcks
             2048    0        -------- 30       -1
  Ext/IO : 1st sz/p nxt sz/p min sz/p mx io sz
           4         0         0         -1

  HdrCache : max    free
             512    0
```

图 24: onstat -g smb s 输出

更改智能大对象的存储特性

如果创建了智能大对象空间，但未在 onspaces -c -S 命令的 -Df 选项中指定值，那么使用缺省的存储特征和属性（例如日志记录和缓冲）。在监视智能大对象空间之后，可能会要更改新的智能大对象的存储特征、日志记录状态、锁模式或其他属性。

数据库管理员或程序员可以使用以下方法来覆盖这些存储特征和属性的缺省值：

- 数据库管理员可以使用以下 onspaces 选项之一：
  - 当首次使用 onspaces -c -S 命令创建智能大对象空间时指定值。
  - 在创建智能大对象空间后，用 onspaces -ch 命令更改值。

  在 onspaces 的 -Df 选项的标签选项中指定这些值。有关 onspaces 实用程序的更多信息，请参阅《*SinoDB®* 管理员参考》。
- 数据库管理员可以在 CREATE TABLE 或 ALTER TABLE 语句的 PUT 子句中指定值。

  这些值会覆盖 onspaces 实用程序中的值，而且只对存储于特定表的相关联列中的智能大对象有效。其他智能大对象（来自其他表的列）可能也驻留在相同的智能大对象空间中。其他列会继续使用 onspaces 定义的智能大对象空间的存储特征和属性（或者使用缺省值，如果 onspaces 没有对其进行定义），除非这些列也使用 PUT 子句来覆盖特定列的存储特征和属性。

  如果您不在 PUT 子句中指定智能大对象列的存储特征，那么将从智能大对象空间继承这些特征。

  如果在创建带有智能大对象列的表时未指定 PUT 子句，那么数据库服务器将智能大对象存储在系统缺省智能大对象空间中，该空间由 ONCONFIG 文件中的 SBSPACENAME 配置参数指定。在此情况下，将从 SBSPACENAME 智能大对象空间继承存储特征和属性。
- 程序员可使用 DataBlade® API 和 SinoDB® ESQL/C 中的函数来变更智能大对象列的存储特征。

  有关智能大对象的 DataBlade® API 函数信息，请参阅《*SinoDB® DataBlade® API* 程序员指南》。有关智能大对象的 SinoDB® ESQL/C 函数信息，请参阅《*SinoDB® ESQL/C* 程序员指南》。

表 *9:* 变更智能大对象空间的存储特征和其他属性 在第129页总结了变更智能大对象存储特征的方法。

表 9: 变更智能大对象空间的存储特征和其他属性

存储特征或属性	系统缺省值	由 onspaces 实用程序中 -Df 选项指定的特定于系统的存储特征	CREATE TABLE 或 ALTER TABLE 的 PUT 子句指定的列级别存储特征	由 DataBlade® API 函数指定的存储特征	由 ESQL/C 函数指定的存储特征
上次访问时间	OFF	ACCESSTIME	KEEP ACCESS TIME, NO KEEP ACCESS TIME	是	是
锁模式	BLOB	LOCK_MODE	否	是	是
日志记录状态	OFF	LOGGING	LOG, NO LOG	是	是
数据完整性	HIGH INTEG	否	HIGH INTEG, MODERATE INTEG	是	否
扩展数据块大小	无	EXTENT_SIZE	EXTENT SIZE	是	是
下一个扩展数据块的大小	无	NEXT_SIZE	否	否	否
最小扩展数据块的大小	在 Windows™ 上 2 KB，在 UNIX™ 上 4 KB	MIN_EXT_SIZE	否	否	否
智能大对象大小	8 KB	智能大对象空间中所有智能大对象的平均大小：AVG_LO_SIZE	否	特定智能大对象的估算大小，特定智能大对象的最大大小	特定智能大对象的估算大小，特定智能大对象的最大大小
缓冲池使用情况	ON	BUFFERING	否	LO_BUFFER 和 LO_ NOBUFFER 标志	LO_BUFFER 和 LO_ NOBUFFER 标志
智能大对象空间的名称	SBSPACE-NAME	不在 -Df 选项中。在 onspaces -S 选项中指定的名称。	智能大对象所驻留的现有智能大对象空间的名称：PUT ... IN 子句	是	是
在多个智能大对象空间中的分段存储	无	否	循环分布方案：PUT ... IN 子句	循环或基于表达式的分布方案	循环或基于表达式的分布方案
上次访问时间	OFF	ACCESSTIME	KEEP ACCESS TIME, NO KEEP ACCESS TIME	是	是

变更智能大对象列
创建或修改表时，您具有若干选项来选择特定智能大对象列的存储特征和其他属性（例如日志记录状态、缓冲、数据完整性和锁定粒度）。

创建或修改可以存储 BLOB 或 CLOB 对象的表时，可以使用以下选项：

- 使用创建智能大对象空间时设置的值。这些值使用以下其中一种方法指定：
  - 使用 onspaces -c -S 命令的 -Df 选项的各种标志
  - 对于任何未指定的标志，使用系统缺省值。

有关更改 -Df 标志的缺省存储特征的准则，请参阅影响智能大对象空间 *I/O* 的 *onspaces* 选项 在第99页。

- 使用 CREATE TABLE 语句的 PUT 子句为特殊特征或属性指定非缺省值，包括智能大对象空间数、扩展数据块大小、日志记录、缓冲、数据完整性状态以及锁定粒度。

  在 PUT 子句中未指定特殊特征或属性时，其缺省值为 onspaces -c -S 命令中设置的值或系统缺省值（例如，无日志记录）。

稍后，您可以使用 ALTER TABLE 语句的 PUT 子句来更改 BLOB 列或 CLOB 列的可选存储特征。请参阅表 *9:* 变更智能大对象空间的存储特征和其他属性 在第129页以获取您可以更改的特征和属性。

您可以使用 ALTER TABLE 语句的 PUT 子句来执行以下操作：

- 在向表添加新的 BLOB 列或 CLOB 列时，指定智能大对象特征和存储位置。

  新列中的智能大对象的特征可以与现有列的特征不同。
- 更改现有列的智能大对象特征。

  新列特征只适用于新行中在 ALTER TABLE PUT 语句发布后插入的智能大对象。在 ALTER TABLE PUT 语句修改列之前，任何仍存在于该列中的智能大对象的旧特征保持不变。

例如，superstores_demo 数据库的 catalog 表中的 BLOB 数据存储于 s9_sbspc 中，日志记录关闭，扩展数据块大小为 100 KB。您可以使用 ALTER TABLE 语句的 PUT 子句来打开日志记录，并将新的智能大对象存储于不同的智能大对象空间中。

有关使用 CREATE TABLE 语句更改智能大对象空间扩展数据块的信息，请参阅智能大对象空间中智能大对象的扩展数据块大小 在第132页。

相关链接

《*SinoDB* 管理员指南》*:* 智能大对象空间日志记录

《*SinoDB SQL* 指南*:* 语法》*: CREATE TABLE* 语句

管理扩展数据块

向表添加行时，数据库服务器以扩展数据块为单位来分配磁盘空间。每个扩展数据块都是一批来自数据库空间且在物理上相邻接的页。即使数据库空间包含多个块，每个扩展数据块也会完全分配在单个的块中，从而保证邻接。

连续性对于性能很重要。当数据页是连续的且数据库服务器在预先读取、轻度扫描或轻量级 I/O 操作期间顺序读取行时，磁盘臂的动作将最少。有关这些操作的更多信息，请参阅顺序扫描 在第101页、轻度扫描 在第101页和影响智能大对象空间 *I/O* 的配置参数 在第98页。

扩展数据块机制是以下互相冲突的需求之间的一种折衷：

- 大多数数据库空间在几个表之间共享。
- 有些表的大小事先未知。
- 各个表可能在不同的时间以不同的速率增长。
- 表的所有页应该邻接，以获得最佳性能。

如果有需要更多扩展数据块的表，并且数据库服务器已用完分区头页上的空间，那么该数据库服务器会自动分配扩展的辅助分区头页以容纳新的扩展数据块条目。数据库服务器可为任何分区分配最多 32767 个扩展数据块，除非表的大小限制扩展数据块数。

因为表的大小未知，所以数据库服务器无法预先分配表空间。因此，数据库服务器只有在需要时添加扩展数据块，但是要获得更好的性能，任何一个扩展数据块中的所有页都应该是连续的。另外，数据库服务器创建一个与先前扩展数据块相邻的扩展数据块时，它会将这两个扩展数据块视为单个扩展数据块。

经常更新的表可能会随着时间推移变为分段表，这会降低每次服务器访问该表时的性能。对表取消分段可使数据行更紧密的集合在一起，并避免分区头页溢出问题。

选择表扩展数据块大小

创建表时，可以为数据库空间中表的数据行和分段表的每个分段以及智能大对象空间中的智能大对象指定扩展数据块大小。数据库服务器会计算智能大对象空间中智能大对象的扩展数据块大小。

数据库空间中表的扩展数据块大小

在创建表时，您可以指定第一个扩展数据块的大小，也可以指定表增长时要添加的扩展数据块的大小。还可以修改数据库空间中表内第一个扩展数据块的大小，以及修改新的后续扩展数据块的大小。

以下 SQL 语句的示例创建了一个表，其初始扩展数据块为 512 KB，添加的扩展数据块为 100 KB：

```
CREATE TABLE big_one (…column specifications…)
   IN big_space
   EXTENT SIZE 512
   NEXT SIZE 100
```

扩展数据块大小和下一扩展数据块大小的缺省值为系统磁盘页大小的 8 倍。例如，如果磁盘页为 2 KB，那么该缺省长度为 16 KB。

可以使用带有 MODIFY EXTENT SIZE 子句的 ALTER TABLE 语句来更改数据库空间中表的第一个扩展数据块的大小。更改第一个扩展数据块的大小时，SinoDB® 将更改记录在系统目录和分区页上，但是仅在重新构建表或创建新分区或分段时才执行实际更改。

在以下某个情境中，您可能希望更改数据库空间中表的第一个扩展数据块的大小：

- 如果创建表的第一个扩展数据块大小较小，那么您需要不断添加大量后续扩展数据块，以及表会在多个扩展数据块中分段且数据分散。
- 如果创建表的第一个扩展数据块比存储的数据量大得多，将浪费空间。

以下示例将数据库空间中表的第一个扩展数据块的大小更改为 50 KB：

```
ALTER TABLE customer MODIFY EXTENT SIZE 50;
```

对第一个扩展数据块的更改将记录在系统目录表和磁盘上的分区页中。但是，对第一个扩展数据块大小的更改不会立即生效。而是在重新构建表的更改发生时，服务器会使用新的第一个扩展数据块大小。

例如，如果表的第一个扩展数据块大小为 8 KB，而您使用 ALTER TABLE 语句将此值更改为 16 KB，那么服务器不会删除当前的第一个扩展数据块并以新的大小重新构建。相反，服务器只有在诸如创建表的集群索引或从表拆离分段之类的操作后重新构建表时，新的第一个扩展数据块大小 (16 KB) 才会生效。

如果在带有 MODIFY EXTENT SIZE 子句的 ALTER TABLE 语句之前执行无 REUSE 选项的 TRUNCATE TABLE 语句，那么在当前第一个扩展数据块中没有任何更改。

使用 MODIFY NEXT SIZE 子句来更改要添加的下一个扩展数据块的大小。此更改不会影响已存在的后续扩展数据块。

以下示例将表的下一扩展数据块的大小更改为 50 KB：

```
ALTER TABLE big_one MODIFY NEXT SIZE 50;
```

以下几种表的下一个扩展数据块大小不会对性能产生显著影响：

- 小表，即定义为只含有一个扩展数据块的表。如果此类表频繁使用，那么其大部分将缓冲于内存中。
- 对于很少使用的表，不管其大小有多大，均不会对性能产生重要影响。
- 驻留在专用数据库空间中的表总会接收到与其旧的扩展数据块相邻接的新扩展数据块。由于邻接的原因，这些扩展数据块表现为一整个大的扩展数据块，因此这些扩展数据块的大小也不重要。

避免创建大量扩展数据块

给这些类型的表分配扩展数据块大小时，唯一的注意事项就是要避免创建大量的扩展数据块。大量的扩展数据块会使数据库服务器花费额外的时间找数据。另外，对于允许的扩展数据块数目，还存在一个上限。（考虑扩展数据块数的上限 在第133页包含此主题。）

为表扩展数据块分配空间的技巧

除了块的大小以外，扩展数据块的大小没有上限。块的最大值为 4 TB。如果知道表的最终大小（或者对预计其最终大小有把握，且误差在 25% 以内），那么可将其全部空间均分配于初始的扩展数据块。如果表稳定增长到未知大小，那么向表分配下一扩展数据块大小，让每个表均可分到少量扩展数据块，从而共享数据库空间。

为表扩展数据块分配空间

要为表扩展数据块分配空间：

1. 决定如何在表之间分配空间。

   例如，可以按 0.4: 0.2: 0.3 的比率，将数据库空间分配给三个表（保留 10% 用于小表和开销）。
2. 为每个表分配其数据库空间份额的四分之一作为其初始扩展数据块。
3. 而将八分之一的份额用作下一扩展数据块大小。
4. 用 oncheck 定期监视表的增长。

数据库空间填满后，可能没有足够的邻接空间来创建指定大小的扩展数据块。在这种情况下，数据库服务器会尽可能分配最大的连续扩展数据块。

相关链接

《SinoDB 管理员参考》: TBLTBLFIRST 配置参数

《SinoDB 管理员参考》: TBLTBLNEXT 配置参数

《SinoDB SQL 指南: 语法》: MODIFY EXTENT SIZE

表分段的扩展数据块大小

对现有表进行分段时，您可能想要调整下一扩展数据块大小，因为每个分段所要求的空间比最初的未分段表所要求的小。

如果定义未分段表时使用了很大的下一扩展数据块大小，数据库服务器会对每个分段都使用相同大小的下一扩展数据块，这样会导致磁盘空间的过量分配。每个分段只需要整个表空间的一部分。

例如，如果将上述 big_one 样本表分段到 5 个磁盘上，那么可以将下一扩展数据块大小改为最初大小的 1/5。以下示例将下一扩展数据块大小改为最初大小的 1/5：

```
ALTER TABLE big_one MODIFY NEXT SIZE 2;
```

相关链接

《SinoDB SQL 指南: 语法》: MODIFY NEXT SIZE 子句

智能大对象空间中智能大对象的扩展数据块大小

创建表时，应该使用数据库服务器为智能大对象空间中的智能大对象计算的扩展数据块大小。或者，可以使用智能大对象的最终大小（如打开应用程序中的智能大对象空间时特定函数所示）。

打开以下某个应用程序时，可以使用智能大对象的最终大小：

- *DB-Access:* 使用 DataBlade® API mi_lo_specset_estbytes 函数。有关打开智能大对象和设置估算字节数的 DataBlade® API 函数的更多信息，请参阅《*SinoDB® DataBlade® API* 程序员指南》。
- *For ESQL/C:* 使用 SinoDB® ESQL/C ifx_lo_specset_estbytes 函数。有关打开智能大对象和设置估算字节数的 SinoDB® ESQL/C 函数的更多信息，请参阅《*SinoDB® ESQL/C* 程序员指南》。

有关调整扩展数据块大小的更多信息，请参阅智能大对象空间扩展数据块 在第99页。有关更多信息，请参阅监视智能大对象空间 在第125页。

监视活动的表空间

监视表空间以确定哪些表是活动的。活动表是线程当前打开的表。

onstat -t 选项的输出包括表空间号和以下四个字段。

npages
分配给表空间的页数

nused
此分配池中使用的页数

nextns
使用的扩展数据块数

npdata
使用的数据页数

如果特定操作需要的页比可用的页多（npages 减去 nused），那么需要新的扩展数据块。如果此块上的可用空间足够多，数据库服务器会在这里分配扩展数据块，否则，数据库服务器会在其他可用块中寻找空间。如果没有块包含足够的连续空间，那么数据库服务器将使用在数据库空间中可找到的最大连续空间块。图 25: onstat -t 输出 在第133页显示了此选项的输出示例。

```
Tblspaces
 n address  flgs ucnt tblnum   physaddr npages nused  npdata nrows  nextns
 0 422528   1    1    100001   10000e   150    124    0      0      3
 1 422640   1    1    200001   200004   50     36     0      0      1
54 426038   1    6    100035   1008ac   3650   3631   3158   60000  3
62 4268f8   1    6    100034   1008ab   8      6      4      60     1
63 426a10   3    6    100036   1008ad   368    365    19     612    3
64 426b28   1    6    100033   1008aa   8      3      1      6      1
193 42f840  1    6    10001b   100028   8      5      2      30     1
 7 active, 200 total, 64 hash buckets
```

图 25: onstat -t 输出

监视扩展数据块数的上限和扩展数据块的交错情况

可以监视扩展数据块数的上限。也可以检查和消除扩展数据块的交错情况。

分区的最大扩展数据块数为 32767。

考虑扩展数据块数的上限

不允许表获取大量的扩展数据块，因为允许的扩展数据块数存在上限。达到此限制值后，尝试添加扩展数据块会在 INSERT 请求之后产生错误 -136（No more extents）。

为了确保不超过限制值，数据库服务器会执行以下操作：

- 在每次创建扩展数据块时，数据库服务器都会检查扩展数据块的数量。如果正在创建的扩展数据块的数目为 16 的倍数，数据库服务器会自动为表的下一扩展数据块大小进行加倍。因此，在创建扩展数据块时，数据库服务器每 16 次就会将下一扩展数据块的大小加倍。
- 数据库服务器创建一个与先前扩展数据块相邻的扩展数据块时，会将这两个扩展数据块视为单个扩展数据块。

检查扩展数据块的交错情况

如果两个或更多增长中的表共享一个数据库空间，那么其中一个表空间的扩展数据块可能会放在另一个表空间中的多个扩展数据块之间。当发生这种情况时，那么这些扩展数据块已交错。如果为找到表而进行的磁盘搜寻必须跨越多个扩展数据块，那么性能会受到负面影响，对于顺序扫描尤其如此。

交错会使表的扩展数据块之间产生间隔。图 26: 交错表扩展数据块 在第133页显示表扩展数据块之间的间隔。

图 26: 交错表扩展数据块

尝试优化表扩展数据块的大小，以分配连续的磁盘空间，这会限制磁头移动。也可以考虑将表置于分开的数据库空间中。

通过监视块来定期检查扩展数据块的交错情况。执行 oncheck -pe 以获取块中信息的物理布局。会出现以下信息：

- 数据库空间名称和所有者
- 数据库空间中块的数量
- 表的顺序布局和每个块中的可用空间
- 每个表扩展数据块或可用空间的专用页数

该输出对确定扩展数据块的交错程度很有用。如果尽管有足够的可用页数，数据库服务器仍无法在块中分配扩展数据块，那么此块可能已严重交错。

消除交错的扩展数据块
可以通过使用 UNLOAD 和 LOAD 语句重新组织表，创建或变更集群的索引或使用 ALTER TABLE 语句来消除交错的扩展数据块。
重新组织数据库空间和表以消除扩展数据块的交错情况
您可以重新构建数据库空间以消除交错的扩展数据块，以便每个表的扩展数据块都是连续的。

重新组织过的表在数据库空间中的顺序并不重要，但是每个重新组织过的表的页应该是连续的，这样在顺序读取表时无需进行长时间的搜寻。当磁盘臂非顺序读取表时，也只在表所占用的空间上移动。

图 27: 重新组织数据库空间以消除交错的扩展数据块

要重新组织数据库空间中的表：

1. 对于 *DB-Access* 用户: 使用 DB-Access 中的 UNLOAD 语句将数据库空间中的表单独复制到磁带。
2. 删除数据库空间中的所有表。
3. 使用 LOAD 语句或 dbload 实用程序重新创建表。

使用 LOAD 语句重新创建属性与原来相同（包括相同扩展数据块大小）的表。

也可以用 onunload 实用程序卸载表以及用伴随的 onload 实用程序重新载入表。

相关链接

《SinoDB SQL 指南: 语法》: LOAD 语句
《SinoDB SQL 指南: 语法》: UNLOAD 语句
《SinoDB 迁移指南》: onunload 和 onload 实用程序

创建或变更集群索引
根据情况，您可以通过创建集群索引或更改集群索引来消除扩展数据块交错现象。使用 CREATE INDEX 或 ALTER INDEX 语句的 TO CLUSTER 子句时，数据库服务器会对表进行排序并重新构建表。

TO CLUSTER 子句对物理表中的行重新排序以匹配索引中的顺序。有关更多信息，请参阅集群 在第158页。

在以下条件下，TO CLUSTER 子句会消除交错的扩展数据块：

- 块必须包含连续的空间，其大小应足以重建每个表。
- 数据库服务器必须使用此邻接空间重建表。

  如果在此较大的邻接空间之前还存在可用空间块，数据库服务器可能会首先分配较小的空间块。数据库服务器从块的起始处为 ALTER INDEX 进程分配空间，寻找可用空间块（大于或等于为下一个扩展数据块指定的大小）。如果数据库服务器用分散在块中较小的可用空间块重建表，那么将不会消除扩展数据块交错现象。

要显示可用空间块的位置和大小，请执行 oncheck -pe 命令。

要使用 ALTER INDEX 语句的 TO CLUSTER 子句：

1. 对于块中的每个表，除想要建立集群的索引以外，删除其他所有分段索引或拆离的索引。
2. 使用 ALTER INDEX 语句的 TO CLUSTER 子句为其余的索引建立集群。

   通过重新排列行来重建表时，此步骤可以消除扩展数据块的交错现象。
3. 重建所有其他索引。

对索引建立集群前无需删除索引。但是，ALTER INDEX 进程比 CREATE INDEX 更快，因为数据库服务器使用索引按集群顺序来读取数据行。另外，生成的索引压缩程度更高。

要防止问题再次发生，可考虑增加表空间扩展数据块的大小。

使用 ALTER TABLE 消除扩展数据块的交错情况
如果您使用 ALTER TABLE 语句来添加或删除列或者更改列的数据类型，那么数据库服务器将复制并重新构建该表。当数据库服务器重新构建整个表时，会将此表重新写入数据库空间的其他区域。但是，如果其他表也在此数据库空间中，那么无法保证新的扩展数据块相互是邻接的。

> 重要： 对于在 ADD、DROP 和 MODIFY 子句中指定的某些类型的操作，数据库服务器不会在 ALTER TABLE 操作期间复制和重新构建表。在这些情况下，数据库服务器会在更新每行时（而不是在 ALTER TABLE 操作期间）使用定点变更算法来修改每一行。有关此定点变更算法的条件的更多信息，请参阅定点变更 在第141页。

回收扩展数据块中未使用的空间

数据库服务器将磁盘空间作为扩展数据块的一部分分配给某个表空间之后，该磁盘空间就一直专用于该表空间。即使所有扩展数据块页在删除数据之后均变空，该磁盘空间对其他表仍为不可用（除非回收此空间）。

> 重要： 删除表中的行后，数据库服务器会复用该空间以将新的行插入相同的表中。本节描述了回收未使用的空间以供其他表使用的过程。

对于不需要占用原先分配的全部空间量的表，您可能会想要调整其大小。您可以重新分配较小的数据库空间，并释放不需要的空间以供其他表使用。

作为数据库服务器管理员，您可以通过重建表来回收空扩展数据块中的磁盘空间，使其可供其他用户使用。要重建表，可以使用以下任一 SQL 语句：

- ALTER INDEX
- UNLOAD 和 LOAD
- ALTER FRAGMENT

使用 ALTER INDEX 回收空扩展数据块中的空间
如果带有空扩展数据块的表包含索引，那么您可以执行带有 TO CLUSTER 子句的 ALTER INDEX 语句。对索引建立集群会在数据库空间中的不同位置重建表。

执行带有 TO CLUSTER 子句的 ALTER INDEX 语句时，会释放与此表的上一版本关联的所有扩展数据块。同时，该表的新建版本也没有空的扩展数据块了。

相关链接

《SinoDB SQL 指南: 语法》: ALTER INDEX 语句

集群 在第158页

通过卸载、重新创建或重新载入表来回收空扩展数据块中的空间
如果表不包含索引，那么您可以使用 UNLOAD 和 LOAD 语句或 onunload 和 onload 实用程序来卸载表，重新创建表（在相同的数据库空间或在其他数据库空间中），以及重新载入数据。

相关链接

《SinoDB SQL 指南: 语法》: LOAD 语句

《SinoDB SQL 指南: 语法》: UNLOAD 语句

《SinoDB 迁移指南》: onunload 和 onload 实用程序

使用 ALTER FRAGMENT 释放空扩展数据块中的空间
可以使用 ALTER FRAGMENT 语句来重新构建表。执行此语句时，其将释放扩展数据块中已分配给该表的空间。

有关 ALTER FRAGMENT 语句语法的更多信息，请参阅《*SinoDB® SQL* 指南*:* 语法》。

使用 TRUNCATE 关键字管理扩展数据块释放

TRUNCATE 为 SQL 关键字，它能从表及 B 型树索引结构中快速删除活动行，而不会删除该表及其模式、访问权限、触发器、约束和其他属性。通过此 SQL 数据定义语言语句，您可以取消填充本地表并复用该表而无需重新创建，或者您可以释放先前保留数据行和 B 型树结构的存储空间。

存在 TRUNCATE 的两个实现：

- 第一个实现称为“快速截断”，适用于大多数表。
- 第二个实现称为“慢速截断”，适用于包含不透明或智能大对象数据类型的表，或者包含继承索引（该继承索引在数据类型层次结构中 ROW 类型上定义）的表。

使用 TRUNCATE TABLE 语句而不是 DELETE 语句的性能优势远远好于快速截断实现，因为该实现不检验或运行表中的所有行。慢速截断实现针对包含不透明或智能大对象数据类型或继承索引（在数据类型中 ROW 类型上定义）的表，因为截断操作会检验包含这些项的每一行。

有关使用 TRUNCATE 的更多信息，请参阅《*SinoDB® SQL* 指南*:* 语法》。

对分区取消分段来合并扩展数据块

可以通过对分区取消分段以合并非邻接的扩展数据块，从而提高性能。

随着时间推移，经常更新的表可能变为分段表，这会降低每次服务器访问该表时的性能。对表取消分段可使数据行更紧密的集合在一起，并避免分区头页溢出问题。

对索引取消分段会让条目更集中，这会提高存取表信息的速度。

提交取消分段请求之后，将无法停止该请求。此外，存在无法取消分段的特定对象，且如果与取消分段请求冲突的其他操作正在运行，那么将无法对分区取消分段。

提示： 对分区取消分段之前：

- 查看*Partition defragmentation*中有关重要限制和注意事项的信息。
- 执行 oncheck -pt and pT 命令来确定特定表或分段的扩展数据块数。

要对表、索引或分区取消分段，请执行带有 defragment 参数的 EXECUTE FUNCTION 命令。您可以指定要取消分段的表名、索引名或分区号。

您可以使用 onstat -g defragment 命令来显示有关活动取消分段请求的信息。

相关链接

《SinoDB 管理员指南》: 自动优化数据存储
《SinoDB 管理员参考》: onstat -g defragment 命令: 显示分区扩展块碎片重整信息
《SinoDB 管理员参考》: oncheck -pt 和 -pT: 显示表或分段的表空间
《SinoDB 管理员参考》: defragment 参数: 动态碎片重整分区扩展块 (SQL 管理 API)

将多个表分段存储在单个数据库空间

您可以将相同的表或索引的多个分段存储在单个数据库空间中，从而减少分段表所需的数据库空间总数。必须为要存储在相同数据库空间中的每个分段指定一个名称。在单个数据库空间中存储多个表或索引分段可简化数据库空间的管理。

当数据库空间位于较快的设备上时，可利用该功能提高在不同数据库空间中存储的每个分段的查询性能。

有关更多信息，请参阅《*SinoDB®* 管理员指南》中有关管理分区的信息。

显示表和索引分区的列表

使用 onstat -g opn 选项显示系统中当前打开的表和索引分区的列表（按线程标识符排列）。

有关 onstat -g opn 输出的示例以及输出字段的说明，请参阅《*SinoDB*® 管理员参考》。

更改表以提高性能

可以通过删除索引、连接或拆离分段和变更表定义来更改表以改善性能。也可以通过在 OLTP 数据库中卸载和载入表来为决策支持应用程序创建数据库。

出于以下种种原因，您可能想要更改现有的表：

- 为了定期刷新大的决策支持表的数据
- 为了添加或删除某个时段的历史数据
- 为了满足不同数据分析的需要，而在大的决策支持表中添加、删除或修改列

载入和卸载表

可以通过定期载入从活动的 OLTP 数据库卸载的表来创建用于决策支持应用程序的数据库。

可以使用以下一种或多种方法来迅速载入大的表：

- 外部表
- 非日志记录表

  数据库服务器对以下各项提供支持：

  - 在日志记录数据库中创建非日志记录或日志记录表。
  - 将表在非日志记录和日志记录之间进行相互转换。

  两种表类型为 STANDARD（日志记录表）和 RAW（非日志记录表）。您可以使用任意载入实用程序（例如，dbimport）来载入原始表。

以下部分将会描述：

- 日志记录表和非日志记录表的优点
- 使用非日志记录表载入数据的逐步过程

相关链接

《SinoDB 管理员指南》: 使用外部表移动数据

《SinoDB SQL 指南: 语法》: CREATE EXTERNAL TABLE 语句

日志记录表的优势

日志记录类型选项指定日志记录特征，这些特征可以改善对表的各种批量操作的性能。

STANDARD 类型对应于先前版本的日志记录数据库中的表，是不指定表类型情况下发出 CREATE TABLE 语句时使用的缺省日志记录类型。

标准表具有以下特点：

- 日志记录，以允许从档案中回滚、恢复和还原。
- 从备份中恢复
- 所有插入、删除和更新操作
- 保持数据完整性的约束
- 快速检索少量行的索引

OLTP 应用程序通常使用标准表。OLTP 应用程序通常具有以下特征：

- 实时插入、更新和删除事务

这些事务的日志记录和恢复对保留数据非常关键。锁定对于允许并行访问和确保所选数据的一致性非常关键。

- 一次更新、插入或删除一行或几行

  索引加速了对这些行的访问。索引只需要少量的 I/O 操作便能访问相关的行，但是通过扫描表来找到相关的行可能需要很多 I/O 操作。

非日志记录表的优势
根据非日志记录表（也称为原始表）的特性，您可以快速载入非常大的数据仓库表。

原始表具有以下特征：

- 不使用 CPU 和 I/O 资源来记录日志。
- 避免类似超出逻辑日志空间的问题。
- 在快速载入时使用互斥锁模式，因此在载入时其他用户无法访问该表。
- 不支持参考约束和唯一约束，因此不需要检查约束的开销。

快速载入大型标准表
可以将大的现有标准表更改为非日志记录表，然后载入该表。

要快速载入大型现有标准表：

1. 删除索引、参考约束和唯一约束。
2. 将表更改为非日志记录。

   以下 SQL 语句示例将 STANDARD 表更改为非日志记录表：

   ```
   ALTER TABLE largetab TYPE(RAW);
   ```

3. 使用载入工具（例如 dbexport）载入表。

   有关 dbexport 和 dbload 的更多信息，请参阅《*SinoDB*® 迁移指南》。
4. 执行非日志记录表的 0 级备份。

   将修改过的任何非日志记录表转换成 STANDARD 类型之前，您必须对其进行 0 级备份。0 级备份提供了还原数据的起始点。
5. 在事务中使用非日志记录表前，应将其更改为日志记录表。

   以下 SQL 语句示例将原始表更改为标准表：

   ```
   ALTER TABLE largetab TYPE(STANDARD);
   ```

   警告： 请勿在多个用户可修改数据的事务内使用非日志记录表。如果需要在这样的事务中使用非日志记录表，请设置 Repeatable Read 隔离级别或者以互斥方式锁定该表，以防止出现并行性问题。

   有关标准表的更多信息，请参阅上一节日志记录表的优势 在第137页。
6. 重新创建索引、参考约束和唯一约束。

快速载入新的非日志记录表
快速创建新的非日志记录表并载入该表。

要快速创建和载入新的大表，请执行以下操作：

1. 在日志记录数据库中创建非日志记录表。

   以下 SQL 语句示例会创建非日志记录表：

   ```
   CREATE DATABASE history WITH LOG;
   CONNECT TO DATABASE history;
   CREATE RAW TABLE history (...
   );
   ```

2. 使用载入工具（例如 dbexport）载入表。有关 dbexport 和 dbload 的更多信息，请参阅《*SinoDB*® 迁移指南》。
3. 执行非日志记录表的 0 级备份。

   将修改过的任何非日志记录表转换成 STANDARD 类型之前，您必须对其进行 0 级备份。0 级备份提供了还原数据的起始点。
4. 在事务中使用非日志记录表前，应将其更改为日志记录表。

   以下 SQL 语句示例将原始表更改为标准表：

   ```
   ALTER TABLE largetab TYPE(STANDARD);
   ```

   警告： 请勿在多个用户可修改数据的事务内使用非日志记录表。如果需要在这样的事务中使用非日志记录表，请设置 Repeatable Read 隔离级别或者以互斥方式锁定该表，以防止出现并行性问题。

   有关标准表的更多信息，请参阅上一节日志记录表的优势 在第137页。
5. 对查询过滤器中最常使用的列创建索引。
6. 根据需要，创建参考约束和唯一约束。

删除索引以提高表更新效率

在某些应用程序中，可以将大多数表的更新限制在一个时段内进行。您可以设置系统，让所有的更新在夜间或者指定日期进行。如果以批处理方式进行更新，您就可以在更新时删除所有非唯一索引，然后建立新索引。

该策略有两个优点：

- 如果更新程序在更新表的同时不需要更新索引，那么该程序的运行会快得多。
- 重新创建的索引更有效率。

有关何时删除索引的更多信息，请参阅非唯一索引 在第159页。

要装入没有索引的表：

1. 删除该表（如果存在）。
2. 不指定任何唯一约束的情况下创建表。
3. 将所有行载入表中。
4. 更改该表，使之符合唯一约束。
5. 创建非唯一索引。

如果无法保证载入的数据符合所有唯一约束，那么必须在载入行之前建立唯一索引。如果这些行按照至少一个索引的正确顺序显示，那么可以节约时间。如果可以选择，使之成为包含最大键的行。此策略可以使必须读取和写入的叶子页的数量最少。

有效创建和启用参考索引

对包含数据的现有表创建或启用外键约束时，有时您可以通过减少数据库服务器搜索违规行的时间以获得更好的性能。

通过在 DML 操作期间维护数据库的参考完整性和通过支持与星型模式相关表的有效连接查询执行路径，外键约束可以提高数据库中 DML 操作的性能，在该数据库中，每个维表的主键对应于事实表的外键。

当您使用 ALTER TABLE ADD CONSTRAINT 或 ALTER TABLE MODIFY 语句定义现有表的外键约束时，如果参考表的列已具有对应于外键约束的键的唯一索引或主键约束，那么可能可以减少验证新外键约束所需的时间。对已包含数据的表创建外键约束时，数据库服务器会检查该表任何违反约束的行。如果存在索引，那么数据库服务器会基于成本来决定是否扫描表中每一行的违规情况或是否只扫描索引阀值。

对于大表，只扫描索引值可以显著改善性能，除非没有满足以下其中一个要求：

- ALTER TABLE 语句只创建一个外键约束。
- ALTER TABLE 语句不会创建或启用 CHECK 约束。

- ALTER TABLE 语句不会更改表中任何现有列的数据类型。
- 外键列不包含用户定义的数据类型 (UDT) 或内置不透明数据类型。
- 外键约束的新模式不是 DISABLED。
- 该表不与活动的违规表关联。

除了一个或多个违规行的情况外，ALTER TABLE ADD CONSTRAINT 或 ALTER TABLE MODIFY 语句可以在上述某些要求没有得到满足时创建并验证外键约束，但是数据库服务器将不会考虑使用索引键算法来验证外键约束。扫描整个表的额外验证成本往往与表的大小成正比。

使用索引扫描验证启用外键约束

要验证启用的外键约束，数据库服务器将执行全表扫描以搜索违规行，除非外键列值上已存在唯一索引或主键约束。在这种情况下，除非不满足以下一个或多个要求，否则数据库服务器会考虑使用索引扫描进行验证：

- SET CONSTRAINTS 语句只启用一个外键约束。
- 相同的语句不能启用 CHECK 约束。
- 外键列不包含用户定义的数据类型 (UDT) 或内置不透明数据类型。
- 外键约束的新模式不是 DISABLED。
- 该表不与活动的违规表关联。

除非表中有一个或多个违规行，否则 SET CONSTRAINTS 语句可以在上述某些要求没有得到满足时启用和验证外键约束，但数据库服务器将不会考虑使用索引键算法来验证外键约束。对于非常大的表，全表扫描的额外验证成本可能很高。

跳过外键约束的验证

在上述 ALTER TABLE 和 SET CONSTRAINTS 操作中，目标都是使用更有效的算法来验证参考约束。通过推迟或避免由 ALTER TABLE ADD CONSTRAINT 语句创建的 ENABLED 或 FILTERING 外键约束的验证，或 DISABLED 外键约束被重置为 ENABLED 或 FILTERING 模式，至少可以暂时提高效率。

当需要将强制参考约束的表从 OLTP 环境移到另一个数据库或数据仓库时，此功能可能非常有用。如果可用于重新定位的时间不足以检查违规情况，那么可能需要导出表并在没有验证的情况下恢复其约束。如果在表导出之前删除或禁用约束，那么这些表不太可能包括违规行。

在创建外键约束、导出外键约束或将其模式从 DISABLED 更改时，可以使用三种替代机制绕过已启用或过滤外键约束的验证：

- 您可以在约束模式规范中包含 NOVALIDATE 关键字
  - ALTER TABLE ADD CONSTRAINT 语句的约束模式规范
  - 或 SET CONSTRAINTS ENABLED 语句的约束模式规范
  - 或 SET CONSTRAINTS FILTERING WITH ERROR 语句的约束模式规范
  - 或 SET CONSTRAINTS FILTERING WITHOUT ERROR 语句的约束模式规范
- 如果您打算执行多个 ALTER TABLE ADD CONSTRAINT 或 SET CONSTRAINTS 语句，那么请执行 SET ENVIRONMENT NOVALIDATE ON 语句来禁用当前会话期间的外键约束验证。

  设置此会话环境选项使 NOVIOLATE 成为在 DDL 语句运行时启用或过滤参考约束的缺省模式。
- 如果您正在迁移数据，那么请在 dbimport 命令中包含 -nv 选项。

  -nv 命令行选项的作用是处理创建或启用外键约束的任何 ALTER TABLE ADD CONSTRAINT 或 SET CONSTRAINTS 语句的约束模式，以便将 ENABLED、FILTERING WITH ERROR 或 FILTERING WITHOUT ERROR 约束模式规范分别实现为 ENABLED NOVALIDATE、FILTERING WITH ERROR NOVALIDATE 或 FILTERING WITHOUT ERROR NOVALIDATE 模式。

在每种情况下，在 DDL 语句中都不会对现有行进行约束验证。

在创建或更改外键约束模式的 DDL 操作之外，NOVALIDATE 关键字或 dbimport 的 -nv 命令行标志的效果不会持续。在后续的 DELETE、INSERT、MERGE 和 UPDATE 操作期间相同约束会强制参考完整性。参考约束的 NOVALIDATE 模式不会在 sysobjstate 系统目录表中注册。

如果在可能已包含违反外键约束的行的表上使用 NOVALIDATE 约束模式，那么用户有责任验证数据中是否存在违规行。

连接或分离分段

可以使用 ALTER FRAGMENT ATTACH 和 DETACH 语句来执行数据仓库类型操作。ALTER FRAGMENT DETACH 提供了快速删除表数据段的方法。同样，ALTER FRAGMENT ATTACH 提供了通过利用分段存储技术向现有表递增载入大量数据的方法。

有关如何利用 ALTER FRAGMENT 语句 ATTACH 和 DETACH 选项的性能增强的更多信息，请参阅提高连接和拆离分段的操作性能 在第201页。

变更表定义

数据库服务器使用以下其中一种算法来处理 SQL 中的 ALTER TABLE 语句：慢速变更、定点变更或快速变更。

慢速变更

数据库服务器使用慢速变更算法来处理 ALTER TABLE 语句时，其他用户可能会长时间无法使用该表。

表可能不可用，因为数据库服务器：

- 在 ALTER TABLE 操作期间以互斥方式锁定了表
- 为了将表转换成新的定义而制作了表的副本
- 在 ALTER TABLE 操作期间转换数据行
- 可以将 ALTER TABLE 语句视为长事务，如果超过 LTXHWM 阈值，会将其终止

由于数据库服务器复制表以将该表转换为新定义，因此慢速变更操作需要的空间至少为原始表大小的两倍与日志空间之和。

ALTER TABLE 语句对列的更改无法定点执行时，数据库服务器将使用慢速变更算法：

- 添加或删除 ROWIDS 关键字创建的列
- 添加或删除 REPLCHECK 关键字创建的列
- 删除 TEXT 或 BYTE 数据类型的列
- 将 SMALLINT 列修改为 SERIAL、SERIAL8 或 BIGSERIAL
- 将 INT 列转换为 SERIAL、SERIAL8 或 BIGSERIAL
- 修改列的数据类型，这样旧数据类型的某些可能值将无法转换为新数据类型（例如，如果将某列的数据类型 INTEGER 修改为 CHAR(n)，那么在 n 值小于 11 的情况下，数据库服务器将使用慢速变更算法。INTEGER 需要 10 个字符加一个负号字符来表示可能的最小负数值。）
- 使用某种方法来修改分段存储列的数据类型，该方法中的值转换可能会导致行移动到其他分段
- 表中包含用户定义的数据类型、智能大对象或者 LVARCHAR、SET、MULTISET、ROW 或 COLLECTION 数据类型时，添加、删除或修改任意列
- 修改 VARCHAR 或 NVARCHAR 列的原始大小或保留规范
- 添加 ERKEY 影子列

定点变更

定点变更算法具有优于慢速变更算法的诸多性能优势

定点变更算法：

- 增加了表的可用性

  ALTER TABLE 操作使用定点变更算法时，其他用户稍后可以访问该表，因为数据库服务器只在更新表定义和重建包含更改列的索引时才锁定该表。

  对于要求每天 24 小时，每周 7 天不停运行的应用程序系统，这种对表可用性的增强可以提高系统吞吐量。

  使用定点变更算法时，数据库服务器锁定表的时间比慢速变更算法要短，因为数据库服务器：

  - 将表转换成新的定义时，不必为表制作副本

  - 在 ALTER TABLE 操作期间，不转换数据行
  - 当您以后更新或插入行时，在变更操作之后使用最新定义来定点变更物理列。数据库服务器对驻留在您更新的每页上的行进行转换。
- 需要的空间比慢速变更算法所需要的少

  ALTER TABLE 操作使用慢速变更算法时，数据库服务器会复制表以将该表转换为新定义。ALTER TABLE 操作要求空间至少为原始表大小的两倍与日志空间之和。

  ALTER TABLE 操作使用定点变更算法时，可以为非常大的表节省大量的空间。
- 在 ALTER TABLE 操作期间提高系统吞吐量

  在定点变更操作期间，数据库服务器不会对表数据的任何更改进行日志记录。不对更改进行日志记录有以下优点：

  - 对于非常大的表，可以节省大量的日志空间。
  - 变更操作不是长事务。

如果 check_for_ipa 调度程序任务已启用，那么具有一个或多个待处理定点变更操作的每个表在 sysadmin 数据库的 ph_alert 表中列出。警报文本为：Table *database*:*owner*.*table_name* has outstanding in place alters. 警报类型为参考。

相关链接

《SinoDB 管理员参考》: ph_alert 表

定点变更操作的条件

数据库服务器只将定点变更算法用于处理 ALTER TABLE 语句的某些 ADD、DROP 或 MODIFY 操作，并且只有在表模式或 ALTER TABLE 语句不需要慢速变更算法时使用。

可用定点完成的 ALTER TABLE 操作

数据库服务器可以在以下 ALTER TABLE 操作中使用定点变更算法：

- 除了防止定点变更操作的条件 在第144页中列出的数据类型外，添加内置数据类型的列。
- 除了包含 TEXT 或 BYTE 数据类型的列或使用 ROWIDS 关键字创建的列外，删除内置数据类型的列。
- 在 Enterprise Replication 中，添加或删除 CRCOLS 关键字创建的列。
- 修改满足以下条件的列：数据库服务器可以将其旧数据类型的所有可能值转换为新数据类型。
- 仅当数据类型转换后值的变更不需要将任何数据行从一个分段移动到另一个分段时，修改该表分段存储表达式中的列。

下表显示 ALTER TABLE MODIFY 语句使用定点变更算法来转换支持的数据类型列的条件。

Key:

All = 数据库服务器将定点变更算法用于特定列操作的所有情况。
nf = 数据库服务器在修改的列不是表分段存储表达式的一部分时使用定点变更算法。

表 10: 使用定点变更算法的 MODIFY 操作和条件

对列的操作	条件
将 SMALLINT 列转换为 INTEGER 列	All
将 SMALLINT 列转换为 BIGINT 列	All
将 SMALLINT 列转换为 INT8 列	All
将 SMALLINT 列转换为 DEC(p2,s2) 列	p2-s2 >= 5
将 SMALLINT 列转换为 DEC(p2) 列	p2-s2 >= 5 OR nf
将 SMALLINT 列转换为 SMALLFLOAT 列	All
将 SMALLINT 列转换为 FLOAT 列	All

对列的操作	条件
将 SMALLINT 列转换为 CHAR(n) 列	n >= 6 AND nf
将 INT 列转换为 INT8 列	All
将 INT 列转换为 DEC(p2,s2) 列	p2-s2 >= 10
将 INT 列转换为 DEC(p2) 列	p2 >= 10 OR nf
将 INT 列转换为 SMALLFLOAT 列	nf
将 INT 列转换为 FLOAT 列	All
将 INT 列转换为 CHAR(n) 列	n >= 11 AND nf
将 SERIAL 列转换为 INT8 列	All
将 SERIAL 列转换为 DEC(p2,s2) 列	p2-s2 >= 10
将 SERIAL 列转换为 DEC(p2) 列	p2 >= 10 OR nf
将 SERIAL 列转换为 SMALLFLOAT 列	nf
将 SERIAL 列转换为 FLOAT 列	All
将 SERIAL 列转换为 CHAR(n) 列	n >= 11 AND nf
将 SERIAL 列转换为 BIGSERIAL 列	All
将 SERIAL 列转换为 SERIAL8 列	All
将 SERIAL8 列转换为 BIGSERIAL 列	All
将 BIGSERIAL 列转换为 SERIAL8 列	All
将 DEC(p1,s1) 列转换为 SMALLINT 列	p1-s1 < 5 AND (s1 == 0 OR nf)
将 DEC(p1,s1) 列转换为 INTEGER 列	p1-s1 < 10 AND (s1 == 0 OR nf)
将 DEC(p1,s1) 列转换为 INT8 列	p1-s1 < 20 AND (s1 == 0 OR nf)
将 DEC(p1,s1) 列转换为 SERIAL 列	p1-s1 < 10 AND (s1 == 0 OR nf)
将 DEC(p1,s1) 列转换为 BIGSERIAL 列	p1-s1 < 20 AND (s1 == 0 OR nf)
将 DEC(p1,s1) 列转换为 SERIAL8 列	p1-s1 < 20 AND (s1 == 0 OR nf)
将 DEC(p1,s1) 列转换为 DEC(p2,s2) 列	p2-s2 >= p1-s1 AND (s2 >= s1 OR nf)
将 DEC(p1,s1) 列转换为 DEC(p2) 列	p2 >= p1 OR nf
将 DEC(p1,s1) 列转换为 SMALLFLOAT 列	nf
将 DEC(p1,s1) 列转换为 FLOAT 列	nf
将 DEC(p1,s1) 列转换为 CHAR(n) 列	n >= 8 AND nf
将 DEC(p1) 列转换为 DEC(p2) 列	p2 >= p1 OR nf
将 DEC(p1) 列转换为 SMALLFLOAT 列	nf

对列的操作	条件
将 DEC(p1) 列转换为 FLOAT 列	nf
将 DEC(p1) 列转换为 CHAR(n) 列	n >= 8 AND nf
将 SMALLFLOAT 列转换为 DEC(p2) 列	nf
将 SMALLFLOAT 列转换为 FLOAT 列	nf
将 SMALLFLOAT 列转换为 CHAR(n) 列	n >= 8 AND nf
将 FLOAT 列转换为 DEC(p2) 列	nf
将 FLOAT 列转换为 SMALLFLOAT 列	nf
将 FLOAT 列转换为 CHAR(n) 列	n >= 8 AND nf
将 CHAR(m) 列转换为 CHAR(n) 列	n >= m OR (nf AND not ANSI mode)
增加字符类型列的长度	Not in ANSI mode databases
增加 DECIMAL 或 MONEY 列的长度	All

防止定点变更操作的条件

当表包含不透明数据类型、用户定义的数据类型、LVARCHAR 数据类型、BOOLEAN 数据类型或智能大对象（BLOB 或 CLOB）时，数据库服务器不会使用定点变更算法，即使要变更的列是可以支持定点变更操作的数据类型。

如果 ALTER TABLE DROP 语句指定 BYTE 或 TEXT 列或 ROWIDS 关键字，或者如果 ALTER TABLE ADD 语句包含 ROWID 关键字，那么不会使用定点变更算法。

如果 ALTER TABLE MODIFY 语句中的任何列数据类型无法通过定点变更操作进行转换，或者如果分段表需要数据移动，那么数据库服务器将使用慢速变更算法（而不是使用定点变更算法）进行数据类型转换。

例如，数据库服务器不会在以下情况中使用定点变更算法：

- 当需要多个算法时

  例如，假设 ALTER TABLE MODIFY 语句将 SMALLINT 列转换为 DEC(8,2) 列并将 INTEGER 列转换为 CHAR(8) 列。第一个列的转换是定点变更操作，但是第二个列的转换是慢速变更操作。数据库服务器使用慢速变更算法来执行该语句。
- 当 ALTER TABLE 操作将数据记录移动到其他分段

  例如，假设您有一个包含两个整数列和以下分段存储表达式的表：

  ```
  col1 < col2 IN dbspace1, REMAINDER IN dbspace2
  ```

  如果发出 ALTER TABLE MODIFY 语句将整数值转换为字符值，那么数据库服务器会在变更操作前将行 (4, 30) 存储在 dbspace1 中，但是在变更操作后将其作为字符 '30' < '4'（而不是作为整数 4 < 30）存储在 dbspace2 中。
- 当数据库服务器无法将旧数据类型的所有可能值转换为新数据类型时。

  例如，您无法将 BIGSERIAL 列转换为 SERIAL 列，因为修改的列无法存储超过 SERIAL 值范围的 BIGSERIAL 值。（但是，如果表中其他列与定点变更操作的任何其他限制不冲突，那么您可以通过定点变更操作将列从 SERIAL 更改为 BIGSERIAL。）

相关链接

《SinoDB 数据库设计和实现指南》: SinoDB 数据类型

DECIMAL

DML 语句的性能注意事项

如果数据库服务器在数据操作语言 (DML) 语句（INSERT、UPDATE、DELETE 和 SELECT）执行期间检测到任何旧版页，那么会执行附加操作。这些操作会影响性能。

每次执行使用定点变更算法的 ALTER TABLE 语句时，数据库服务器将创建新版本的表结构。数据库服务器保持对表定义的所有版本的跟踪。数据库服务器对版本状态以及所有版本结构和更改结构进行复位，直到整个表转换为最终格式或执行慢速变更。

如果数据库服务器在 DML 语句（INSERT、UPDATE、DELETE 和 SELECT 语句以及指定 Insert、Update 或 Delete 子句的 MERGE 语句）执行期间检测到任何旧版页，那么会执行以下操作：

- 对于 UPDATE 语句，数据库服务器将整个数据页或全部数据页转换成最终格式。
- 对于 INSERT 语句，数据库服务器将插入的行转换为最终格式，并将其插入最合适的页。数据库服务器将最合适页中的现有行转换成最终格式。
- 对于 DELETE 语句，数据库服务器不会将数据页转换为最终格式。
- 对于 SELECT 语句，数据库服务器不会将数据页转换为最终格式。

  如果查询要访问的行还没有转换为新的表定义，那么您可能会注意到个别查询的性能稍微有些下降，这是因为数据库服务器在返回每一行之前，都会将其重新格式化。

DDL 操作的定点变更性能

对数据定义语言 (DDL) 语句的定点变更操作会降低性能。因此，监视未完成的定点变更，因为许多未完成的变更操作会影响后续 ALTER TABLE 语句。

oncheck -pT 命令显示未完成的定点变更操作的数据页版本。数据页仍存在旧定义时，定点变更是 *outstanding* 状态。

图 *28: customer* 表的 *oncheck -pT* 输出示例 在第145页显示对 customer 演示表运行四个定点变更操作之后以下 oncheck 命令所产生输出的一部分：

```
oncheck -pT stores_demo:customer

...
Home Data Page Version Summary

            Version           Count

            0 (oldest)           2
            1           0
            2           0
            3           0
            4 (current)           0
...
```

图 28: customer 表的 oncheck -pT 输出示例

图 *28: customer* 表的 *oncheck -pT* 输出示例 在第145页中的 Count 字段显示了当前正在使用该版本的表定义的页数。此 oncheck 输出显示四个版本均未完成：

- 最老版本的 Count 字段中的 2 表示有两页在使用最老版本。
- 接下来四个版本的 Count 字段中的 0 表示没有页转换成最新的表定义。

  重要： 对表执行的定点变更越来越多时，每个后续 ALTER 语句或对含有未完成变更的表执行的 SQL 语句会花费比上一个语句更多的时间。要维持高效的性能，可以定期移除未完成的定点变更操作。

您可以通过执行带有 table update_ipa 或 fragment update_ipa 参数的 admin( ) 或 task( ) SQL 管理命令来移除定点变更操作。可以包含 parallel 选项来并行执行该操作。例如，以下语句从名为 auto 的表中并行移除定点变更操作：

```
EXECUTE FUNCTION task('table update_ipa parallel','auto');
```

如果您的目标是节省运行时 CPU，那么可计划对表保留尽可能少的未完成变更操作（一般不超过 3 或 4 个）。如果您的目标是节省磁盘空间，且您的变更操作会添加或增加列，那么保留未完成的定点变更有助于减少磁盘空间。但是，如果您需要转换为数据库服务器的较早版本，那么数据页不可以包含不完整 ALTER TABLE 或 ALTER FRAGMENT 操作。

在对表或分段的所有未完成的定点变更操作都完成后，oncheck -pT 命令会在 Count 字段中显示表当前版本的总数据页数。

相关链接

《*SinoDB* 迁移指南》：解决未完成的定点变更操作

变更作为索引一部分的列

如果变更的列是索引的一部分，那么表仍会定点变更，但此种情况下数据库服务器会默示地重新建立一个或多个索引。如果不需要重建索引，应在执行更改操作之前将其删除或禁用。执行这些步骤可以提高性能。

但是，如果修改的列是主键或外键，且您想要保持这种约束，那么您必须在 ALTER TABLE 语句中再次指定这些关键字，而且数据库服务器将重建索引。

例如，假设您使用以下 SQL 语句创建表并更改父表：

```
CREATE TABLE parent
   (si SMALLINT PRIMARY KEY CONSTRAINT pkey);
CREATE TABLE child
   (si SMALLINT REFERENCES parent ON DELETE CASCADE
   CONSTRAINT ckey);
INSERT INTO parent (si) VALUES (1);
INSERT INTO parent (si) VALUES (2);
INSERT INTO child (si) VALUES (1);
INSERT INTO child (si) VALUES (2);
ALTER TABLE parent
   MODIFY (si INT PRIMARY KEY CONSTRAINT pkey);
```

该 ALTER TABLE 示例将 SMALLINT 列转换为 INT 列。由于 ALTER TABLE 语句指定了 PRIMARY KEY 关键字和 pkey 约束，因此数据库服务器将保留主键。在 MODIFY 子句中指定 PRIMARY KEY 约束时，数据库服务器也会在相同的主键列上静默创建 NOT NULL 约束。然而，数据库服务器会删除该主键的任何引用约束。因此，您还必须为子表指定以下 ALTER TABLE 语句：

```
ALTER TABLE child
   MODIFY (si int references parent on delete cascade
         constraint ckey);
```

即使主键或外键列上的 ALTER TABLE 操作重建了索引，数据库服务器仍将使用定点变更算法。定点变更算法在性能方面具有以下优势：

- 无需为了将表转换成新定义而制作表的副本。
- 在变更操作期间不转换数据行。
- 并不重建对表的所有索引。

警告： 如果变更作为视图的一部分的表，必须重建该视图以获得表的最新定义。

快速变更

ALTER TABLE 语句更改表的属性但不影响数据时，数据库服务器将使用快速变更算法。

使用 ALTER TABLE 语句执行以下操作时，数据库服务器将使用快速变更算法：

- 更改下一扩展数据块大小。
- 添加或删除约束。
- 更改表的锁模式。
- 不修改列类型的情况下更改唯一索引属性。
- 使用 ADD VERCOLS 关键字为行版本控制添加影子列。

使用快速变更算法时，数据库服务器对表的锁定只保持很短的时间。在某些情况下，数据库服务器锁定系统目录表只是为了更改属性。无论哪种情况，无法查询此表的时间均很短。

对数据模型进行反向规范化以提高性能

可能需要取消数据模型规范化以减少开销并优化性能。

《*SinoDB® SQL* 指南: 教程》中描述的实体关系数据模型会生成不包含冗余或派生数据的表。根据关系数据库的原则，这些表的结构组织得很好。

有时，为了满足对高性能的特别需求，您可能需要采用理论上并非上策的方法对数据模型进行修改来取消数据模型规范化。本节将描述一些修改及其相关成本。

缩短行

通常，行短的表比行长的表具有更好的性能，因为磁盘 I/O 以页为单位而不是以行为单位执行。表的行越短，每页上的行数也就越多。每页的行数越多，顺序读取表所需的 I/O 操作就越少，并且从缓冲区可以执行的非顺序存取的可能性越大。

实体关系数据模型将一个实体的所有属性都放在该实体的单个表中。对于某些实体，这种策略会产生长度难以使用的行。

要缩短这些行，可以将列拆成一些分开的表，并通过每个表中的复制键值进行关联。由于行变短了，因此查询性能应该会提高。

排除长字符串

最占空间的属性通常是字符串。要缩短行，可以将长字符串从实体表中移除。

可以使用以下方法来排除长字符串：

- 使用 VARCHAR 列。
- 使用 TEXT 数据。
- 将字符串移到伴随表。
- 建立符号表。

将 CHAR 列转换为 VARCHAR 列以缩短行 (GLS)

数据库可能包含 CHAR 列，您可以将这些列转换为 VARCHAR 列。CHAR 列中的文本字符串平均长度比列宽短至少 2 个字节时，您可以使用 VARCHAR 列来缩短行的平均长度。

VARCHAR 数据可立即与大多数现有程序、表单和报表兼容。您可能需要重新编译应用程序开发工具生成的任何表单以识别 VARCHAR 列。修改表的模式后，应经常使用样本数据库测试表单和报表。

有关其他字符数据类型的信息，请参阅《*SinoDB® GLS* 用户指南》。

将长字符串转换为 TEXT 数据类型列

如果某个字符串填充了一半或一半以上磁盘页，那么您应考虑将其转换为单独 BLOB 空间中的 TEXT 数据类型列。

行页中列的长度只有 56 字节，一页中可以容纳的行数比包含长字符串时要多。但是，TEXT 数据类型不能自动与现有程序兼容。访存 TEXT 值所需的应用程序比将 CHAR 值载入程序的代码稍微复杂一些。

将字符串移到伴随表

如果将不足半页的字符串当作 TEXT 数据进行处理，那么这些字符串将浪费磁盘空间，但是您可以将其从主表移至伴随表。

如果将表拆分为两个表，主表和伴随表，那么请在每个表中重复主键。

构建符号表

如果列所包含的字符串在每行中不唯一，那么可以将这些字符串移到只存储唯一副本的表中。

例如，customer.city 列包含城市名称。有些城市名称在列中是重复的，并且大多数行在字段的尾部都有一些空白。使用 VARCHAR 数据类型可以消除这些空白，但不能消除重复。

您可以创建名为 cities 的表，如以下示例所示：

```
CREATE TABLE cities (
   city_num SERIAL PRIMARY KEY,
   city_name VARCHAR(40) UNIQUE
)
```

可以更改 customer 表的定义，这样其 city 列会变成外键，引用 cities 表中的 city_num 列。

要将新客户的城市插入 cities，必须更改向 customer 插入新行的所有程序。SQL 通信区 (SQLCA) 的 SQLCODE 字段中的数据库服务器返回码可以指示由于重复键而导致插入失败。这不是逻辑错误；仅表示现有的某客户在该城市中。有关 SQLCA 的更多信息，请参阅《*SinoDB® SQL* 指南*:* 教程》。

除了更改插入数据的程序以外，还必须更改检索城市名称的所有程序和存储查询。这些程序和存储查询必须使用新的 cities 表的连接以获得数据。插入行的程序中和某些查询中的额外复杂性是放弃数据模型中的理论正确性的结果。进行更改前，请确保它会节省磁盘空间或执行时间。

分割宽表

对于那些行太宽而影响性能的实体，请考虑其所有属性。寻找某个主题或原则以将其分为两组。然后将表拆分为两个表，主表和伴随表，并在每个表中重复主键。

较短的行使您能快速查询或更新每个表。

按大小拆分

可以拆分实体表的一个原则是大小。将大的属性（通常为字符串）移到伴随表。将数字和其他小的属性保留在主表中。在演示数据库中，可以将 ship_instruct 列从 orders 表中拆分出去。可以称伴随表为 orders_ship。它有两列，复制自 orders.order_num 列的主键，和原来的 ship_instruct 列。

按使用频率拆分

拆分实体的另一个原则是使用频率。如果有几个属性很少被查询，可将其移到伴随表。例如，在演示数据库中，可能只有一个程序查询 ship_instruct、ship_weight 和 ship_charge 列。在这种情况下，可以将其移到伴随表。

按更新频率拆分

更新所需的时间要比查询长，而且在更新过程中，更新程序要锁定索引页和数据行，以防止查询程序访问此表。如果可以将一个表分割为两个伴随表，一个用于最常更新的实体，而另一个用于最常查询的实体，那么总体响应时间通常会缩短。

拆分表的性能成本

将表进行拆分会使用额外的磁盘空间，并增加了复杂性。每行的主键会有两份副本，每个表一份。同时还有两份主键索引。您可以使用之前章节中描述的方法来估算增加的页数。

由于返回的列数较少，因而您必须修改使用 SELECT * 的现有程序、报表和表单。使用两个表的属性的程序、报告和表单必须执行连接，以将表组合在一起。

在这种情况下，插入或删除行时，变更的是两个表，而不是一个。如果不协调这两个表的变更（例如使它们在一个事务中），将会失去语义上的完整性。

冗余数据

规范化的表不包含冗余数据。每个属性只在一个表中出现。

规范化的表也不包含派生数据。相反，从现有属性计算得出的数据将被选为基于这些属性的表达式。

将表规范化可使占用的磁盘空间最小，并使对表的更新尽可能简便。然而，规范化的表经常会强制您使用连接和聚合函数，而那些进程可能是非常耗时的。

作为备选方法，假定您了解所涉及的交易，那么可以引进包含冗余数据的新列。

添加冗余数据

正确的数据模型是将所有属性保留在所描述的实体表中，从而避免冗余性。如果在不同的上下文中需要属性数据，那么要将表进行连接。但是连接会耗时。如果频繁的连接影响了性能，那么可以通过复制连接数据到其他表来消除连接。

在 stores_demo 数据库中，manufact 表包含制造商的名称及其交货时间。实际的工作数据库可能包含供应商的许多其他属性，例如地址和销售代表姓名。

manufact 的内容主要是 stock 表的补充。假设时间关键型应用程序频繁引用某种特殊商品的订货至交货的时间，但不引用 manufact 的其他列。对于每次这样的引用，数据库服务器必须读取 2 到 3 页的数据以执行查找。

可以向 stock 表添加新列 lead_time，并用 manufact 相应行中的 lead_time 列的副本进行填充。这样安排消除了查找，从而使应用程序加快。

和派生数据一样，冗余数据会占用空间，并带来完整性方面的风险。在上一段描述的示例中，可能会存在每个制造商的订货至交货时间的许多额外副本。（每个制造商可能会在 stock 中多次出现。）插入或更新 manufact 的一行的程序也必须更新 stock 的多个行。

简而言之，完整性风险是指数据的冗余副本可能不精确。如果更改 manufact 中的订货至交货时间，那么 stock 列会过时，直到该列也进行更改。使用派生数据时，应定义冗余数据可能出错的条件。

有关数据库设计的更多信息，请参阅《*SinoDB*® 数据库设计和实现指南》。

减少具有可变长度行的表中的磁盘空间

如果将 MAX_FILL_DATA_PAGES 配置参数设为 1，那么数据库服务器可向具有可变长度行的表中的每个页插入更多行。允许每页具有更多可变长度的行既有其优点也有缺点。

每个页允许更多可变长度行的潜在优势：

- 减少存储数据所需的磁盘空间
- 使服务器对缓冲池的使用更有效
- 减少表扫描时间

利用 MAX_FILL_DATA_PAGES 以使每个页允许更多可变长度行的潜在缺点：

- 服务器可能以不同物理顺序存储行。
- 当页已满时，对行中可变长度列的更新可导致行扩展以至于在页中不再适合。这将导致服务器将行划分到两页中，从而增加了该行的访问时间。

如果启用了 MAX_FILL_DATA_PAGES 配置参数，那么服务器将向最近修改过的带有现有行的页添加新行，并且添加该行之后至少保留 10% 的可用页，以便将来扩展页中所有的行。如果没有启用 MAX_FILL_DATA_PAGES 配置参数，那么只有当页上有足够的空间允许新行扩展至其最大长度时，服务器将添加行。

如果启用了 MAX_FILL_DATA_PAGES 配置参数并且想要其影响现有的可变长度行，那么必须重新载入现有的表。

通过压缩表和分段来减少磁盘空间

您可以通过压缩表中的数据和表分段来减少磁盘空间。压缩数据之后，您可以重新打包数据以合并表或分段中的可用空间，并缩小数据的空间以将可用空间返回到数据库空间。

压缩对于具有大量 I/O 活动的应用程序以及减少磁盘空间使用量至关重要的应用程序而言是有利的。但是，如果应用程序运行时具有很高的缓冲区高速缓存命中率而高性能比空间使用量更重要，那么您可能不希望使用压缩功能，因为压缩可能会稍微降低性能。

压缩数据、整合数据以及返还可用空间具有以下优点：

- 显著节省磁盘存储空间
- 减少压缩分段的磁盘使用量
- 显著节省逻辑日志使用量，这样在完成压缩操作后可节省额外空间并可以防止高吞吐量 OLTP 的瓶颈。
- 页读取更少，因为更多行可以置于一个页中
- 缓冲池更小，因为更多数据置于同一大小的池中
- I/O 活动减少，原因如下：
  - 与未压缩行相比，更多压缩行置于一个页中
  - 压缩行的插入、更新和删除操作的日志记录更小
- 可以压缩按时间分段数据的不常访问的较旧分段，而让频繁访问的更多近期数据处于未压缩的格式
- 可以释放表不再需要的空间
- 备份与还原更快

由于与未压缩数据相比，压缩数据占用更少页且每页中有更多行，因此查询优化器可能在压缩后选择不同的计划。

您可以通过并行执行这些操作来加速压缩和重新打包。

相关链接

《SinoDB 管理员指南》: 压缩

《SinoDB 管理员参考》: table 或 fragment 参数: 压缩数据和优化存储 (SQL 管理 API)

第

7

章

索引和索引性能注意事项

SinoDB® 提供多个类型的索引。一些性能问题与索引相关联。

索引的类型

SinoDB® 使用 B 型树索引、R 型树索引、函数索引以及 DataBlade® 模块为用户定义的数据提供的索引。服务器也会使用森林树 (FOT) 索引，这是 B 型树索引的替代项。

相关链接

什么是函数索引? 在第169页

B 型树索引

SinoDB® 将 B 型树索引用于包含内置数据类型的列（称为传统 *B* 型树索引），用于包含一维用户定义的数据类型的列（称为一般 *B* 型树索引），以及用户定义的数据类型返回的值。

内置数据类型包括 character、datetime、integer 和 float 等。有关内置数据类型的更多信息，请参阅《*SinoDB® SQL* 指南: 参考》。

用户定义的数据类型包括不透明和单值数据类型。有关用户定义的数据类型的更多信息，请参阅《*SinoDB®* 用户自定义例程和数据类型开发者指南》。

用户定义函数的返回值可以是内置或用户定义的数据类型，但不是简单大对象（TEXT 或 BYTE 数据类型）或智能大对象（BLOB 或 CLOB 数据类型）。有关如何使用函数索引的更多信息，请参阅使用函数索引 在第169页。

有关如何估算 B 型树索引大小的信息，请参阅估算索引页 在第153页。

常规索引页的结构

常规索引按照页层次结构排列（技术上称为 *B* 型树）。

下图显示索引的 B 型树结构。该层次结构的最顶层包含单独的根页。中间层（在需要时）包含一些枝页。每个枝页包含一些条目，这些条目指向该索引的下一级别的页子集。索引的最底层包含一组叶子页。每个叶子页包含索引条目的列表，这些条目指向表中的行。

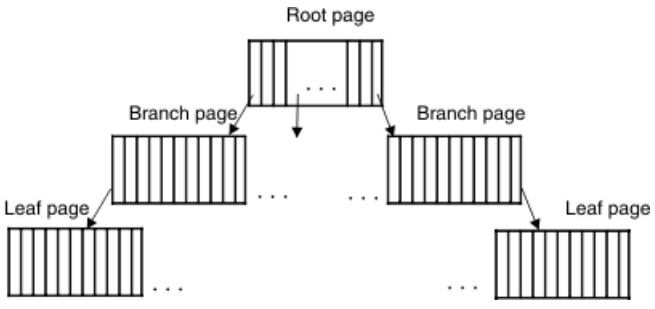

图 29: 索引的 B 型树结构

一个索引所需的层次数取决于该索引中唯一键的数量，以及每页所能包含的索引条目数。每页中条目的数量又取决于建立索引的列的大小。

如果一个给定表的索引页能包含 100 个键，那么最多包含 100 行的表只需要一个单层索引：根页。如果该表增长到超过 100 行，在 101 和 10,000 行之间，那么它需要两层索引：1 个根页和 2 到 100 个叶子页。如果该表增长到超过 10,000 行，其大小在 10,001 和 1,000,000 行之间，那么它需要三层索引：根页、一组数量为 100 的枝叶以及最多可达 10,000 个的叶子页。

包含于叶子页中的索引条目按照键值顺序排序。索引条目由键和一个或多个行指针组成。键就是一行数据的索引列的副本。行指针提供地址，用于定位包含该键的行。唯一索引包含表中每行的一个索引条目。

有关 SinoDB® 的特殊索引的信息，请参阅用户定义的数据类型上的索引 在第166页。

相关链接

森林树索引 在第152页

森林树索引

森林树索引类似于 B 型树索引，但具有多个根节点，并且级别数可能更少。多个根节点可以缓解根节点争用情况，因为更多并发用户可以访问索引。森林树索引也可以通过减少缓冲区读操作中涉及的级别数，以提高查询的性能。

您可以创建森林树索引作为 B 型树索引的替代项，但不能作为 R 型树索引或其他类型索引的替代项。

与包含一个根节点的传统 B 型树索引不同，森林树索引相当于一个划分为更小子树（可将其视为存储区）的大型 B 型树索引。这些子树包含多个根节点和叶。下图显示森林树索引的结构。

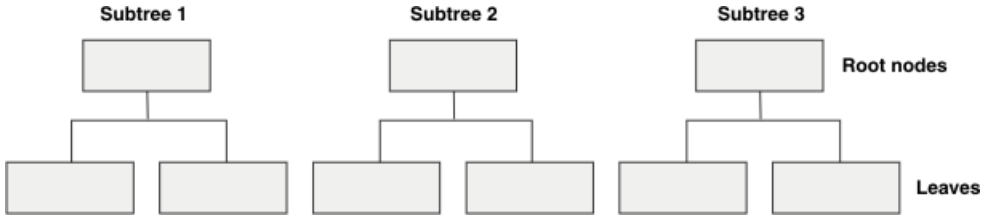

图 30: 森林树索引的结构

SinoDB® 按以下方法存储和检索子树中的项：

1. 从创建索引时选择的列计算散列值。
2. 将散列值映射到存储或检索行的子树。

森林树索引是拆离索引。服务器不支持森林树连接索引。

使用 SQL 的 CREATE INDEX 语句和 HASH ON 子句来创建森林树索引。

使用 SQL 的 SET INDEXES 语句来启用或禁用森林树索引。

可以根据 SET EXPLAIN 输出中 Index Name 字段的 FOT 指示符来标识森林树索引。

通过查看包含具有森林树索引的表的数据库中 sysindices 表的信息，可以查找森林树索引中散列列和子树的数量。

服务器对待森林树索引的方式与对待 B 型树索引的方式相同。因此，在使用日志记录的数据库中，您可以控制 B 型树扫描程序线程如何从森林树索引和 B 型树索引除去删除的操作。

限制：您不能执行以下操作：

- 对具有复杂数据类型 (UDT) 的列或函数列创建森林树索引。
- 在创建森林树索引时使用 CREATE INDEX 语句的 FILLFACTOR 选项（因为索引是从上到下进行构建的）。
- 创建集群森林树索引。
- 对森林树索引执行 ALTER INDEX 语句。
- 对集群环境内辅助服务器数据库中的森林树索引执行 SET INDEXES 语句。
- 在使用聚合的查询中使用森林树索引，包括最小和最大范围值。

- 直接对森林树索引的 HASH ON 列执行范围扫描。

  但是，您可以对 HASH ON 列列表中未列出的列执行范围扫描。要对 HASH ON 列列表中列出的列执行范围扫描，必须额外创建一个 B 型树索引，其中包含适合范围扫描的列列表。这一额外的 B 型树索引可以具有与森林树索引相同的列列表，加上或减去一列。
- 对 OR 索引路径使用森林树索引。数据库服务器不会对索引列上具有 OR 谓词的查询使用森林树索引。

相关链接

使用森林树索引提高查询性能 在第160页
检测根节点争用 在第160页
创建森林树索引 在第161页
禁用和启用森林树索引 在第161页
确定是否要使用森林树索引 在第162页
常规索引页的结构 在第151页
《*SinoDB SQL* 指南: 语法》: *CREATE INDEX* 语句
《*SinoDB SQL* 指南: 语法》: *HASH ON* 子句
使用森林树索引提高查询性能 在第160页
检测根节点争用 在第160页

R 型树索引

SinoDB® 将 R 型树索引用于空间数据（例如二维或三维数据）。

有关调整 R 型树索引大小的信息，请参阅《*SinoDB® R-Tree* 索引用户指南》。

DataBlade® 模块提供的索引

DataBlade® 模块可以包含用户定义的数据类型。DataBlade® 模块也可以为新数据类型提供用户定义的索引。

有关每个 DataBlade® 模块提供的数据类型和函数的更多信息，请参阅每个 DataBlade® 模块的用户指南。有关如何确定数据库中可用索引类型的信息，请参阅确定可用的访问方法 在第168页。

估算索引页

与表相关联的索引页可能会显著增加数据库空间的大小。

缺省情况下，数据库服务器创建的索引与表位于相同的数据库空间，但位于不同的表空间。要将索引置于单独的数据库空间，应在 CREATE INDEX 语句中指定 IN 关键字。

虽然无法显式地指定索引的扩展数据块大小，但是可以估算索引所占用的页数，以确定是否为单个或多个数据库空间分配了足够的空间。

索引扩展数据块大小

数据库服务器根据相应表的扩展数据块大小来确定索引的扩展数据块大小，而不管索引是否分段。

用于估算连接索引的扩展数据块大小的公式
对于连接索引，数据库服务器使用索引键大小与行大小的比率来为索引分配适当的扩展数据块大小。

以下公式显示数据库服务器如何使用索引键大小与行大小的比率：

```
索引扩展数据块大小 = (index_key_size /
table_row_size) *
   table_extent_size
```

在此公式中：

- *index_key_size* 是已建立索引的一个或多个列的总宽度加上 5（用于键描述符）。

- *table_row_size* 是行中所有列的总数。
- *table_extent_size* 是在 CREATE TABLE 语句的 EXTENT SIZE 关键字中指定的值。

如果索引不是唯一的，那么扩展数据块大小将减少 20%。

对索引的下一扩展数据块大小，数据库服务器也使用相同的比率：

```
索引下一扩展数据块大小 =
(index_key_size/table_row_size)*
   table_next_extent_size
```

用于估算拆离索引的扩展数据块大小的公式
对于拆离索引，数据库服务器使用索引键大小与某些开销字节数之和与行大小的比率来为索引分配适当的扩展数据块大小。

以下公式显示数据库服务器如何使用索引键大小与某些开销字节数之和与行大小的比率：

```
拆离索引扩展数据块大小 = ( (index_key_size +
9) /                              table_row_size) *
 table_extent_size
```

例如，假设您有以下值：

```
index_key_size = 8 字节
table_row_size = 33 字节
table_extent_size = 150 * 2 KB 页
```

以上述公式计算扩展数据块大小，如下所示：

```
拆离索引扩展数据块大小 = ( (8 + 9) /
33) * 150 * 2 KB 页
                        = (17/33) * 300 KB
                        = 154 KB
```

重要： 对于非唯一索引，此公式会计算减少了 20% 的扩展数据块大小。

估算常规索引页

可以使用一系列公式来估算索引页的大小。

要估算索引页的数量：

1. 合计单个或多个索引列的宽度。

   所得到的值称为 colsize。将 *colsize* 加上 4，得到 *keysize*，该值是索引中键的实际大小。例如，如果 *colsize* 为 6，那么 *keysize* 的值为 10。

2. 计算唯一条目占行总数的期望比例。

   后续步骤中的公式将此值看作 *propunique*。

   如果索引是唯一的，或者只有极少数的重复值，那么对 *propunique* 赋值为 1。

   如果很大比例的条目是重复的，那么用表中唯一索引条目的个数除以表的行数，将所得的小数值赋给 *propunique*。例如，如果表中的行数为 4,000,000，唯一索引条目的个数为 1,000,000，那么 *propunique* 的值为 .25。

   如果 *propunique* 的结果小于 .01，那么在以下的计算中使用值 .01。

3. 选用以下其中一个公式估算典型索引条目的大小，选择的依据是该表是否分段：

a) 对于未分段表，使用以下公式：

```
entrysize = (keysize * propunique) + 5 + 4
```

5 代表未分段表中行指针所占字节数。

对于非唯一索引，数据库服务器存储索引节点中每一行的行指针，但键值只存储一次。entrysize 值表示每个索引条目的平均长度，即使某些条目只包括行指针。

例如，如果 *propunique* 为 .25，那么每个唯一键值的平均行数为 4。如果 *keysize* 为 10，那么 *entrysize* 的值为 11.5，按如下方式计算：(10 * 0.25) + 5 + 4 = 2.5 + 9 = 11.5。以下计算显示所有 4 行必需的空间：

```
4 行的空间 = 4 * 11.5 = 46
```

如果使用键值所占空间大小与 4 个行指针所占空间大小相加，那么所得的空间需求与上述结果相同，如以下公式所示：

```
4 行的空间 = 10 + (4 * 9) = 46
```

b) 对于分段表，使用以下公式：

```
entrysize = (keysize * propunique) + 9 + 4
```

9 代表分段表中行指针所占字节数。

4. 使用以下公式估算每个索引页中的条目数：

```
pagents = trunc(pagefree/entrysize)
```

在此公式中：

- *pagefree* 是页大小减去页头（对于大小为 2 KB 的页，页头为 2020）。
- *entrysize* 是在上一步中估算的典型索引条目的大小。

trunc() 函数符号表示应该向下舍入到最接近的整数值。

5. 使用以下公式估算叶子页的数目：

```
leaves = ceiling(rows/pagents)
```

在此公式中：

- *rows* 为表中期望的行数。
- *pagents* 是在上一步中估算的每个索引页的条目数。

ceiling() 函数符号表示应该向上舍入到最接近的整数值。

6. 使用以下公式估算索引的第二层枝页的数目：

$$branches_0 = ceiling(leaves/node\_ents)$$

使用以下公式计算 *node_ents* 的值：

```
node_ents = trunc( pagefree / ( keysize + 4) + 4)
```

在此公式中：

- *pagefree* 是页大小减去页头（对于大小为 2 KB 的页，页头为 2020）。
- keysize 是 *colsize* 加上 4。在步骤 1 中获取此值。

在该公式中，4 表示页节点指针所占字节数。

7. 如果 $branches_0$ 的值大于 1，那么索引中包含更多的层次。

   要计算索引的下一层中包含的页数，请使用以下公式：

   $$branches_{n+1} = ceiling(branches_n / node\_ents)$$

   在此公式中：

   - $branches_n$ 是计算出的上一索引层中的枝数。
   - $branches_{n+1}$ 是下一层中的枝数。
   - $node\_ents$ 是步骤 6 中计算的值。

8. 对该索引中的每一层，均重复步骤 7 中的计算，直到 $branches_{n+1}$ 的值等于 1。
9. 将步骤 6 到步骤 8 中计算出的所有枝层的页数进行汇总，所得的总数称为 *branchtotal*。
10. 使用以下公式计算压缩索引中的页数：

    $$compactpages = (leaves + branchtotal)$$

11. 如果数据库服务器实例对索引使用一个填充因子，那么该索引的大小就会增加。

    缺省的填充因子是 90%。您可以使用 FILLFACTOR 配置参数来更改所有索引的填充因子值。您也可以使用 SQL 中 CREATE INDEX 语句的 FILLFACTOR 子句来更改个别索引的填充因子。

    要在估算索引页时并入填充因子，请使用以下公式：

    $$indexpages = 100 * compactpages / FILLFACTOR$$

以上估算方法只用作准则。由于可能会删除旧行和插入新行，所以页内的索引条目数会发生变化。对于大多数索引而言，这种估算索引页的方法会得到一个保守（偏高）的估算值。要得到更精确的值，请使用真实数据构建一个大型测试索引，并使用 oncheck 实用程序检查其大小。

> 提示： 森林树索引可以大于 B 型树索引。估算森林树索引的大小时，估算值会应用于索引中的每个子树。然而，您必须聚合存储区来计算总估算值。

管理索引

在一次查询中，对适当列的索引可以节省数千、数万，或者在极端情况下甚至数百万次的磁盘操作。然而，索引也需要成本。

对于任何列或列组合而言，索引是必需的，而且应该是唯一的。但是，正如查询和查询优化器 在第210页中所讨论的，使用索引也可以使查询优化器能够加速查询。

优化器能够以以下方法使用索引：

- 用非顺序访问替换对表的重复顺序扫描
- 在处理只对已建立索引的列进行命名的表达式时，避免读取行数据
- 执行 GROUP BY 和 ORDER BY 子句时，要避免排序（包括构建一个临时表）

相关链接

使用函数索引 在第169页

索引的空间开销

索引的第一项成本是磁盘空间。使用索引会向数据库空间添加很多页，在一个索引表中，很容易出现索引页和行页一样多的情形。此外，在使用多种语言的环境中，为每种语言创建的索引需要附加磁盘空间。

当您考虑空间成本时，也请考虑在环境中增大标准或临时数据库空间的页大小是否有利。如果您想要更长的密钥长度（比缺省页大小可提供的长度长），那么可以增大页大小。如果增大页大小，那么此大小必须是缺省页大小的整数倍，且不大于 16 KB。

如果应用程序包含大小较小的行，那么您可能不希望增大页大小。对于随机访问小行的应用程序，增大页大小可能会降低性能。此外，较大页的页锁还会锁定更多的行，这在某些情况下可降低并发性。

您可以通过压缩拆离的 B 型树索引、整合索引中的可用空间并将该可用空间返还给数据库空间来节省磁盘空间。

相关链接

《*SinoDB* 管理员指南》：*B* 型树索引压缩

索引的时间开销

索引的第二项成本是表修改所花费的时间。

以下描述假设必须读取大约两页才能定位某个索引条目。如果索引包含一个根页、一层枝页和一组叶子页，就属于这种情况。假设根页已经在缓冲区中。对于很大的表，其索引至少有两层中间层，因此数据库服务器引用这样一个索引时，大约会读取三页。

据推测，定位一个正在改变的行需要使用一个索引。用于该索引的页可能会在数据库服务器的共享内存中的页缓冲区中找到。但是，用于其他任何需要改变的索引的页都必须从磁盘中读取。

在这些假设下，索引维护将增加各种修改所花费的时间。如以下列表所示：

- 从表中删除行时，数据库服务器必须从所有索引中删除该行的条目。

  数据库服务器必须查找已删除行（读取两页或三页）的条目，并重写叶子页。更新该索引的写操作在内存中执行，并且在包含修改页的最近最少使用的 (LRU) 缓冲区清除后就会刷新叶子页。此操作必须访问两页或三页以对索引页进行读取操作（如果需要），还必须访问一个延迟页以对修改页进行写操作。
- 插入行时，数据库服务器必须将该行的条目插入所有索引中。

  数据库服务器必须找到一个地方，用于输入每个索引中（读取两页或三页）插入的行并重写（输出一个延迟页），因此每个索引需要访问三个或者四个中间页。
- 在更新一行时，数据库服务器必须查找应用到变更列的每个索引中的条目（读取两页或三页）。

  数据库服务器必须重写叶子页，以消除旧的条目（输出一个延迟页），然后在同一个索引中定位新的列值（再读取两页或三页）和输入的行（再输出另一个延迟页）。

插入和删除会更改叶子页中的索引数。虽然实际上每个 *pagents* 操作均要求一些附加工作，以处理填充或清空的叶子页，但如果 *pagents* 大于 100，这些附加工作所花费的时间不会超过总时间的 1%。估计 I/O 影响时，常常将其忽略。

简而言之，如果随机插入或删除一行，大约每个索引要增加三到四页的 I/O 操作。更新某行时，对于应用于变更列的每个索引大约要增加六到八页的 I/O 操作。如果事务回滚，那么所有此类工作必须撤销。因此，回滚事务会花费很长的时间。

由于行本身的变更只需要两页的 I/O 操作，因此索引维护显然是数据修改中最耗时的部分。有关降低此成本的一种方法的信息，请参阅集群 在第158页。

尚未回收的索引空间

后台线程 B 型树扫描程序标识了具有最多尚未回收索引空间的索引。尚未回收的索引空间会降低性能，并导致服务器承担额外的工作。选择某个索引进行扫描时，将扫描该索引的整个叶以找到删除（脏）的项，这些项已提交但尚未从索引中移除。必要时，B 型树扫描程序将移除这些项。

B 型树扫描程序允许多个线程。

使用 BTSCANNER 配置参数可指定要启动的 B 型树扫描程序线程的数量以及数据库服务器启动时 B 型树扫描程序线程的优先级。有关详细信息，请参阅《*SinoDB*® 管理员参考》。

您可以从命令行调用 B 型树扫描程序。

列的索引

可以为表中的一个或多个列创建索引。对于必须是唯一的，且未指定为主键的列而言，索引是必需的。

此外，必须对列添加满足以下条件的索引：

- 用于未被指定为外键的连接中
- 经常用于过滤器表达式
- 经常用于排序或分组
- 不包括重复键
- 服从集群索引

大型表中已过滤的列
如果一个列经常用于过滤大表中的行，那么应考虑为其放置一个索引。优化器可以使用此索引来查询所需要的列，从而避免对整个表进行顺序扫描。

假设您有一个包含很大邮件列表的表。如果您发现邮政编码列经常用于过滤行的子集，就应考虑为该列放置一个索引。

只有当该列的选择性很高时（即，只有当小部分行具有任何一个索引值时），此策略才能显著节省时间。通过索引进行的非顺序访问比顺序访问多使用一些磁盘 I/O 操作，因此如果列的过滤器表达式需要过滤四分之一以上的行，那么数据库服务器可能也会顺序读取该表。

通常，在以下情况下，为一个过滤器列建立索引可以节省时间：

- 该列用于许多查询或慢速查询中的过滤器表达式。
- 该列包含至少 100 个唯一值。
- 大部分的列值在行中出现的比例低于 10%。

排序依据和分组依据列
可以对表的一个或多个排序列放置索引。然后数据库服务器将该索引用于以最高效的方式对查询结果排序。

如果大量行必须排序或分组，那么数据库服务器必须按顺序排列行。数据库服务器执行此任务的一种方法是选择所有行放入一个临时表，再对该临时表排序。但是，正如查询和查询优化器 在第210页中所述，如果已对排序列建立索引，优化器有时会通过索引按排序顺序读取这些行，从而避免最终的排序。

因为索引中的键是按照顺序排列的，所以索引实际上就代表了对表进行排序的结果。通过在一个或多个排序列中放置索引，您就可以在创建索引时使用单个排序来取代在查询中的多个排序。

避免带有重复键的列
索引中的重复键会导致性能问题。可以采取措施来避免这些问题。

如果索引中允许使用重复键，那么与给定键值匹配的所有条目会分组到列表。数据库服务器使用这些列表定位与请求的键值相匹配的行。如果索引列的选择性很高，这些列表通常会很短。但是如果只有少数唯一值，列表会很长，并可能跨越多个叶子页。

在选择性低（即，相对于行数，不同的值很少）的列上放置索引可能会降低性能。在这种情况下，数据库服务器不仅必须搜索与该键值匹配的整个行集，还必须锁定所有受影响的数据和索引页。这个过程也会影响其他更新请求的性能。

要更正此问题，可以用具有较高选择性的组合索引代替该低选择性列上的索引。在该索引中，将低选择性的列用作第一个列，将高选择性的列用作第二个列。数据库服务器必须搜索一些行以定位和应用更新，而组合索引限制了这些行的数目。

您可以用任何第二列来分散键值，只要其值不发生更改，或者与真实键同时改变。第二列越短越好，因为其值将复制到索引中，从而增大索引的大小。

集群
集群是一种用于排列表中的行的方法，目的是使这些行在磁盘中的物理顺序与索引中条目的顺序密切相关。

如果您知道某个表是按照确定索引排序的，就可以避免排序。您也可以确定，按该列搜索该表时，顺序读取要比非顺序读取有效。这些要点均包含于查询和查询优化器 在第210页中。

> 提示： 有关通过将索引改变为集群来消除交错的扩展数据块的信息，请参阅创建或变更集群索引 在第134页。

在 stores_demo 数据库中，orders 表在 postal-code 列上具有一个索引 zip_ix。以下语句使数据库服务器将 customer 表中的行按邮政编码的降序排列：

```
ALTER INDEX zip_ix TO CLUSTER
```

要将表在非索引列上进行集群，必须创建一个索引。以下语句按订单日期对 orders 表重新进行排序：

```
CREATE CLUSTER INDEX o_date_ix ON orders (order_date ASC)
```

要对表重新进行排序，数据库服务器必须复制该表。在上述示例中，数据库服务器读取表中所有行，并建立索引。然后再顺次读取索引条目。对每个条目，读取表中匹配的行，并将其复制到一个新表中。新表的行按想要的顺序排列。此新表将替换掉旧表。

在更改一个表时，并不保留集群。插入新行时，在物理上会存储于表的末尾，而不管它们的内容。更新行和更改集群列的值时，就会将那些行写回到表中原来的位置。

集群可以在行的顺序被正在进行的更新打乱之后还原。以下语句对表重新排序，从而按照索引顺序还原数据行：

```
ALTER INDEX o_date_ix TO CLUSTER
```

重新集群通常比初始集群快，因为读取几乎已集群的表中的行在 I/O 影响方面与顺序扫描类似。

集群和重新集群要花费大量空间和时间。要避免集群，可以在一开始就以想要的顺序来建立表。

相关链接

使用 *ALTER INDEX* 回收空扩展数据块中的空间 在第135页

影响集群程度的配置参数

sysindexes 或 sysindices 表中的 clust 字段代表索引的集群程度。多个配置参数的值会影响 clust 字段。

该字段的值受以下因素影响：

- BUFFERPOOL 配置参数指定的缓冲池大小
- DS_MAX_QUERIES 配置参数，指定可并行运行的最大 PDQ 查询数

这些配置参数都会影响单个用户会话的可用缓冲区空间量。附加缓冲区可使集群效果更佳（sysindexes 或 sysindices 表中 clust 值更小）。

可以通过执行以下一项或两项任务来创建更多的缓冲区：

- 通过更新 BUFFERPOOL 配置参数的值来增大缓冲池大小
- 减小 DS_MAX_QUERIES 配置参数的值

相关链接

《SinoDB 管理员参考》: BUFFERPOOL 配置参数

《SinoDB 管理员参考》: DS_MAX_QUERIES 配置参数

非唯一索引

在某些应用程序中，大多数表更新可以限制在一个时段内进行。您可以设置系统，让所有的更新在夜间或者指定日期进行。此外，以批处理方式执行更新时，您就可以在更新时删除所有非唯一索引，然后建立新索引。此策略可以改善性能。

删除非唯一索引可具有以下积极影响：

- 由于要更新的索引更少，因此更新程序会运行得更快。通常，删除索引，在无索引的情况下进行更新，然后重新建立索引，所花费的总时间比在有索引的情况下进行更新要短。（有关更新索引的时间成本的讨论，请参阅索引的时间开销 在第157页。）
- 新建的索引效率更高。频繁的更新会弱化索引结构，使索引中包含很多部分完整的叶子页。这种弱化会降低索引的有效性并浪费磁盘空间。

为了节省时间，应确保批处理更新程序按照主键索引定义的顺序请求行。该顺序使主键索引的各页按顺序读取，而且每页只读取一次。

使用 LOAD 语句或 dbload 实用程序时，使用索引还将降低表的植入速度。载入一个没有索引的表是一个很快的过程（和盘对盘顺序复制速度一样），但是更新索引会增加很大的开销。

要避免此开销，可以执行以下操作：

1. 删除该表（如果存在）。
2. 不指定任何唯一约束的情况下创建表。
3. 将所有行均载入表中。
4. 更改该表，使之符合唯一约束。
5. 创建非唯一索引。

如果无法保证载入的数据能够符合所有唯一约束，那么必须在载入行之前建立唯一索引。如果这些行是以至少一种索引正确排序的，就能够节省时间。如果可以选择，使之成为包含最大键的行。此策略可以最大限度地减少必须读取和写入的叶子页的数量。

使用森林树索引提高查询性能

森林树索引是一种建立索引的备用方法，可缓解在许多并发用户访问传统 B 型树索引时可能发生的性能瓶颈和根节点争用。

森林树索引不同于 B 型树索引之处在于前者具有多个根节点和更少的级别。多个根节点可以缓解根节点争用情况，因为更多并发用户可以访问索引。

如果您了解到特定表具有深度表，那么通过创建树中包含更少级别的森林树索引，可提高性能。例如，假设您创建了一个索引，其中某个列是包含字符数据的 100 个字节的列。如果该表中包含大量行，那么树可能包含六个或七个级别。如果创建森林树索引而非 B 型树索引，那么您可以创建具有四个级别的多个树，以便每个索引遍历仅进入四个级别深度而非七个级别深度。

相关链接

森林树索引 在第152页
森林树索引 在第152页

检测根节点争用

您可以分析 onstat -g spi 命令的输出来标识森林树索引可缓解的性能瓶颈。

要检测根节点争用并确定是否需要森林树索引：

1. 执行 onstat -g spi | sort -nr 命令来显示有关具有长自旋的自旋锁的信息。

   onstat -g spi 命令的输出显示具有等待的自旋锁，当多个线程并发在某个索引中读取或写入且特定线程在第一次尝试获取锁失败时，会发生这种情况。
2. 分析 onstat -g spi 输出。查找这些列中的循环和等待信息：

   Num Waits：线程等待自旋锁的总次数。
   Num Loops：线程成功获取自旋锁之前的尝试总数。
   Avg Loop/Wait：获取自旋锁的平均尝试数，计算方式为 Num Loops / Num Waits。

   例如，以下输出片段显示具有大量等待和循环的自旋锁：

```
Spin locks with waits:
Num Waits Num Loops Avg Loop/Wait Name
332480    1568908   4.72  fast mutex, 3:bf[1234] 0x2d00008  0x1028a0d8000
39722     498769    12.56 mutex lock, name = log
20761     101831    4.90  fast mutex, 7:bf[62] 0x1300003  0x109da128000
14818     77680     5.24  mutex lock, name = MGM mutex
6523      34350     5.27  fast mutex, 3:bf[362] 0x20008e   0x10289a08000
```

3. 查询 sysmaster:systabnames，其中部件号的十六进制表示法如 onstat -g spi 输出中所示。如果 tabname 表示索引名，那么该索引为森林树候选值。

   例如，执行此查询：

```
echo "select tabname, hex(partnum) from systabnames
 where hex(partnum) =  '0x02d00008'" | dbaccess sysmaster -

tabname       daily_market_idx
(expression)  0x02d00008

$ echo 'select tabname, hex(partnum) from systabnames'
 where hex(partnum) = 0x01300003 | dbaccess sysmaster -

tabname       trade_history_idx
(expression)  0x01300003

$ echo 'select tabname, hex(partnum) from systabnames'
 where hex(partnum) = 0x0020008E | dbaccess sysmaster -

tabname       trade_request_idx2
(expression)  0x0020008E
```

相关链接

森林树索引 在第152页
森林树索引 在第152页
《*SinoDB* 管理员参考》*: onstat -g spi* 命令*:* 显示使用长自旋的旋转锁

创建森林树索引

使用带有 HASH ON 子句的 CREATE INDEX 语句来创建森林树索引。

先决条件：确定是否需要森林树索引来减少性能瓶颈和争用或者减少传统 B 型树索引中的级别数。

要创建森林树索引：

1. 选择索引的列并确定要创建的子树数量。
2. 使用带有 HASH ON 子句的 CREATE INDEX 语句来创建索引：

   例如，以下命令会在 C1 列上创建具有 100 个子树（存储区）的森林树索引：

```
CREATE INDEX fotidx ON tab(c1) hash on (c1) with 100 buckets
```

   创建森林树索引之后，会启用该索引。

您可以监视 onstat -g spi 命令输出来验证根节点争用是否不再发生。如果确定了高度争用的自旋锁所导致的性能瓶颈，那么您可以重建具有更多存储区的森林树索引。

相关链接

森林树索引 在第152页
《*SinoDB SQL* 指南*:* 语法》*: CREATE INDEX* 语句
《*SinoDB SQL* 指南*:* 语法》*: HASH ON* 子句

禁用和启用森林树索引

如果想要服务器停止更新森林树索引以及在查询期间停止使用森林树索引，那么可以使用 SQL 的 SET Database Object Mode 语句的 INDEXES DISABLED 选项来禁用森林树索引。准备好将索引用于生产环境之后，可以使用 INDEXES ENABLED 选项将其重新启用。

要禁用森林树索引：

执行 SQL 的 SET INDEXES DISABLED 语句。

例如，对于名为 fotidx 的索引，请指定：

```
SET INDEXES fotidx DISABLED;
```

您可以重新启用已禁用的森林树索引，例如，通过指定：

```
SET INDEXES fotidx ENABLED;
```

相关链接

森林树索引 在第152页

在森林树索引上执行范围扫描

虽然无法在森林树索引的 HASH ON 列上直接执行范围扫描，但可以在 HASH ON 列列表中未列出的列上执行范围扫描。要在 HASH ON 列列表中列出的列上执行范围扫描，您必须创建包含范围扫描相应列列表的其他 B 型树索引。

要为范围扫描创建索引：

1. 创建具有至少一个未散列的列的森林树索引。

   例如，指定：

   ```
   CREATE INDEX idx1 on tab(c1,c2) HASH ON (c1) with 100 buckets;
   ```

   您可以直接在列 c2 上执行范围扫描，但不能在 HASH ON 列列表中列出的列 c1 上执行。
2. 要在 HASH ON 列列表中列出的列上执行范围扫描，请创建包含范围扫描相应列列表的其他 B 型树索引。这一额外的 B 型树索引可以具有与森林树索引相同的列列表，加上或减去一列。

   例如，指定：

   ```
   CREATE INDEX idx2 on tab(c1, c2, c3);
   ```

相关链接

《SinoDB SQL 指南: 语法》: CREATE INDEX 语句

《SinoDB SQL 指南: 语法》: HASH ON 子句

确定是否要使用森林树索引

通过查看 SET EXPLAIN 输出，您可以确定某个索引是否为森林树索引。森林树索引在输出的 Index Name 字段中具有 FOT。

在部分 SET EXPLAIN 输出的以下示例中，informix.fot_idx 为森林树索引的名称。

```
Estimated Cost: 1
Estimated # of Rows Returned: 1

  1) informix.t: INDEX PATH

    (1) Index Name: informix.fot_idx (FOT)
        Index Keys: c1 c2   (Serial, fragments: ALL)
        Lower Index Filter: informix.t.c1 = 1
```

相关链接

森林树索引 在第152页

查找森林树索引中散列列和子树的数量

通过查看包含具有森林树索引的表的数据库的 sysindices 表中的信息，可以查找森林树索引中散列列和子树的数量。

要查看有关森林树索引的信息：

1. 查询 sysindices 表以获取索引。
2. 转至包含森林树索引的行并查看 nhashcols 和 nbuckets 列中的信息。

在联机环境中创建和删除索引

当数据库及其相关联的表持续可用时，可使用 CREATE INDEX ONLINE 和 DROP INDEX ONLINE 语句在联机环境中创建和删除索引。

在构建索引期间，CREATE INDEX ONLINE 语句使您能够创建索引而无需对表放置互斥锁。即使正在对表进行读取或更新时，您也可以使用 CREATE INDEX ONLINE 语句。这意味着可立即开始创建索引。

当您联机创建索引时，数据库服务器会使用一个标志来记录该操作，这样，数据恢复和还原操作可重新创建该索引。

当您联机创建索引时，可使用 ONLIDX_MAXMEM 配置参数来限制共享内存中分配给 *preimage* 日志池和 *updator* 日志池的内存量。如果您计划在对某个表列执行 CREATE INDEX ONLINE 语句时在该列上完成其他操作，那么可能希望这样做。有关此参数的更多信息，请参阅联机创建索引时限制内存分配 在第164页。

DROP INDEX ONLINE 语句使您即使在事务隔离级别为 Dirty Read 时也能删除索引。

使用 CREATE INDEX ONLINE 语句创建索引的优点是：

- 如果需要新索引来提高表的查询性能，那么可以立即创建该索引，而无需对表放置锁。
- 数据库服务器可在更新表时创建索引。
- 该表在索引构建期间可用。
- 查询优化器可建立更好的查询计划，因为优化器可更新未锁定表中的统计信息。

使用 DROP INDEX ONLINE 语句删除索引的优点是：

- 您可以删除低效率的索引，而不会妨碍正在进行中的使用该索引的查询。
- 将索引标上标记后，查询优化器就不会将该索引用于表上新的 SELECT 操作。

如果您对正在被更新的表发出 DROP INDEX ONLINE 语句，那么该操作将在表更新完成后才会发生。您发出 DROP INDEX ONLINE 语句后，任何人都无法引用该索引，但是并发操作可使用该索引，直到这些操作终止为止。在所有用户都已完成对该索引的访问之前，数据库服务器将等待删除该索引。

以下是在联机环境中创建索引的示例：

```
CREATE INDEX idx_1 ON table1(col1) ONLINE
```

以下是在联机环境中删除索引的一个示例：

```
DROP INDEX idx_1 ONLINE
```

有关 CREATE INDEX ONLINE 和 DROP INDEX ONLINE 语句的更多信息，请参阅《*SinoDB® SQL* 指南*:* 语法》。

无法联机创建或删除索引时

在某些情况下不能使用 CREATE INDEX ONLINE 和 DROP INDEX ONLINE 语句。

无法使用 CREATE INDEX ONLINE 语句的情况：

- 在改变表的同时创建索引
- 创建集群索引
- 创建虚拟索引接口 (VII) /R 型树索引

- 创建函数索引
- 创建由时间间隔分段存储策略分区的索引
- 创建由时间间隔分段存储策略分区的表索引

无法使用 DROP INDEX ONLINE 语句的情况：

- 删除虚拟索引接口 (VII) /R 型树索引
- 删除集群索引

在联机环境中创建连接索引

可使用 CREATE INDEX ONLINE 语句创建连接索引，但是该语句只在事务隔离级别为 Dirty Read 时才运行。

索引创建在表上获取互斥锁，并在创建连接索引之前，等待所有其他扫描该表的并发进程使用索引分区退出。如果该表正在被读取或更新，那么 CREATE INDEX ONLINE 语句将在锁模式设置期间等待互斥锁。

联机创建索引时限制内存分配

ONLIDX_MAXMEM 配置参数会限制分配给单个 *preimage* 池和单个 *updator* 日志池的内存量。

preimage 和 updator 日志池（pimage_<partnum> 和 ulog_<partnum>）是 CREATE INDEX ONLINE 语句执行时创建的共享内存池。当该语句执行完成后，将释放这些池。

ONLIDX_MAXMEM 配置参数的缺省值是 5120 KB。您可以指定的最小值是 16 KB；最大值是 4294967295 KB。

启动数据库服务器之前，您可以设置 ONLIDX_MAXMEM 配置参数，或通过 onmode -wf 和 onmode -wm 命令动态地对其进行更改。

提高索引构建的性能

可以通过调整 PDQ 优先级并为整个索引分配足够内存和临时空间来改善索引构建的性能。

只要可能，数据库服务器就会使用并行处理，以缩短索引构建的响应时间。并行处理的数量取决于索引中分段的数量，以及 PSORT_NPROCS 环境变量的值。即使 PDQ 优先级的值为 0，数据库服务器仍将使用并行处理方式构建索引。

通常您可以通过执行以下步骤来提高索引构建的性能：

1. 将 PDQ 优先级设置为大于 0 的值，以获取多于缺省 128 KB 的内存。

   如果将 PDQ 优先级设置为大于 0 的值，那么索引构建将利用附加内存进行并行处理。

   要设置 PDQ 优先级，请使用 PDQPRIORITY 环境变量或 SQL 中的 SET PDQPRIORITY 语句。
2. 不要设置 PSORT_NPROCS 环境变量。

   如果您的计算机具有多个 CPU，那么数据库服务器将在对索引键进行排序时为每个排序使用两个线程，并且不会设置 PSORT_NPROCS。排序的数量取决于索引中分段的数量、键的数量、键大小以及 PDQ 内存配置参数的值。
3. 分配足够的内存和临时空间以建立整个索引。
   a) 估算数据库服务器可能需要用于排序的虚拟共享内存量。

      有关更多信息，请参阅估算排序所需的内存 在第165页。
   b) 使用 DS_TOTAL_MEMORY 和 DS_MAX_QUERIES 配置参数指定更多内存。
   c) 如果没有足够的可用内存，那么要估算完整索引构建所需的临时空间量。

      有关更多信息，请参阅估算用于索引构建的临时空间 在第165页。
   d) 使用 onspaces -t 实用程序可创建大型临时数据库空间，并在 DBSPACETEMP 配置参数或 DBSPACETEMP 环境变量中指定这些数据库空间。

      有关如何优化临时数据库空间的信息，请参阅为临时表和排序文件配置数据库空间 在第90页。

估算排序所需的内存

要计算数据库服务器可能需要用于排序的虚拟共享内存量，须估算可能并发的最大排序数，并将此数乘以平均行数和平均行大小。

例如，如果估算会并发 30 个排序，行的平均大小为 200 字节，表中的平均行数为 400，那么可以如下估算数据库服务器用于排序所需要的共享内存量：

```
30 sorts * 200 bytes * 400 rows = 2,400,000 bytes
```

您可以使用 DS_NONPDQ_QUERY_MEM 配置参数来配置可用于非 PDQ 查询的排序内存。

> 重要： 如果 PDQ 优先级设置为 0，那么您只可以使用该参数。如果 PDQ 优先级大于 0，那么其设置无效。

DS_NONPDQ_QUERY_MEM 的最小且缺省的值为 128 KB。受支持的最大值为 DS_TOTAL_MEMORY 的 25%。有关更多信息，请参阅为使用散列连接、聚合和其他内存密集型元素的查询分配更多内存 在第290页。

如果 PDQ 优先级大于 0，那么数据库服务器为排序分配的最大共享内存量将由内存分配管理器 (MGM) 控制。MGM 使用 PDQ 优先级的设置和以下配置参数来确定为排序分配多少内存：

- DS_TOTAL_MEMORY
- DS_MAX_QUERIES
- MAX_PDQPRIORITY

有关为并行处理分配内存的更多信息，请参阅为并行数据库查询分配资源 在第255页。

估算用于索引构建的临时空间

可以估算整个索引构建所需的临时空间的字节数。

要估算索引构建所需的临时空间量，请执行以下步骤：

1. 增加索引列的总宽度或用户定义函数的返回值。此值称为 *colsize*。
2. 根据索引是否已连接，使用以下其中一个公式估算用于排序的典型项的大小：
   a) 对于未分段表，以及不使用显式分段存储策略而建立索引的分段表，请使用以下公式：

   ```
   sizeof_sort_item = keysize + 4
   ```

   b) 对于对索引进行显式分段的分段表，请使用以下公式：

   ```
   sizeof_sort_item =

   keysize + 8
   ```

3. 用以下公式估算排序所需的字节数：

   ```
   temp_bytes = 2 * (rows *
   sizeof_sort_item)
   ```

   此公式使用了因子 2，因为中间排序使用临时空间运行时所有对象均存储了两次。如果没有足够内存用于在内存中执行完整的排序，那么会运行中间排序。

   rows 的值就是表中总行数的期望值。

将多个索引分段存储在单个数据库空间

可在单个数据库空间中存储同一索引的多个分段，以减少分段表所需的数据库空间的总数。必须为您要在相同数据库空间中存储的每个分段指定一个名称。在单个数据库空间中存储多个索引分段简化了数据库空间的管理。

当数据库空间位于较快的设备上时，可利用该功能提高在不同数据库空间中存储每个分段的查询性能。

有关更多信息，请参阅《*SinoDB*® 管理员指南》中有关管理分区的信息。

提高索引检查的性能

oncheck 实用程序为使用行锁定的表提供更好的并发性。当表使用页锁定时，oncheck 在执行索引检查时在表上放置共享锁。在 oncheck 检查或打印索引信息时，共享锁不允许其他用户对表执行更新、插入或删除。

如果该表使用页锁定，在您执行不带 -x 选项的 oncheck 时，数据库服务器会返回以下消息：

```
WARNING: index check requires a s-lock on stable whose
lock level is page.
```

有关 oncheck 锁定的详细信息，请参阅《*SinoDB*® 管理员参考》。

以下总结描述了在索引检查过程中执行的锁定：

- 缺省情况下，在您使用带 -ci、-cI、-pk、-pK、-pl 或 -pL 选项的 oncheck 检查索引时，数据库服务器不会对表放置共享锁，除非该表使用页锁定。在 oncheck 检查带有页锁定的表的索引时，它会对表放置一个共享锁，这样，其他用户将无法执行更新、插入或删除操作，直到该检查完成。
- 如果在索引检查期间，未在使用行锁的表中放置共享锁，那么 oncheck 实用程序在索引检查过程中得到的结果将是不精确的。为了绝对保证完整索引检查的精确性，请执行带 -x 选项的 oncheck。通过使用 -x 选项，oncheck 对表放置一个共享锁，那么其他用户将无法执行更新、插入或删除，直到该检查完成。

可以查询 systables 系统目录表来查看表的当前锁级别，如以下 SQL 语句的示例所示：

```
SELECT locklevel FROM systables
   WHERE tabname = "customer"
```

如果在 locklevel 列没有看到 R 值（对行），那么您可以修改锁级别，如以下 SQL 语句的示例所示：

```
ALTER TABLE tab1 LOCK MODE (ROW);
```

行锁定可能会产生其他副作用，比如引起锁使用的全面增加。有关锁定级别的更多信息，请参阅锁定 在第174页。

用户定义的数据类型上的索引

您可以定义自己的数据类型，以及对这些数据类型进行操作的函数。可以对某些种类的用户定义数据类型定义索引。

DataBlade® 模块也为数据库服务器提供扩展数据类型和函数。

您可以对以下用户定义的数据类型定义索引：

- 不透明数据类型

  不透明数据类型是一种基本数据类型，可以用来定义列，定义方法和使用内置类型时的方法相同。不透明数据类型存储单个值，且数据库服务器无法将其分成多个部分。有关创建不透明数据类型的信息，请参阅《*SinoDB*® *SQL* 指南: 语法》和《*SinoDB*® 用户自定义例程和数据类型开发者指南》中的

CREATE OPAQUE TYPE 语句。有关每个 DataBlade® 模块提供的数据类型和函数的更多信息，请参阅每个 DataBlade® 模块的用户指南。

- 单值数据类型

  单值数据类型与现有的不透明或者内置数据类型的表现形式相同，但它们属于不同的类型。有关单值数据类型的信息，请参阅《*SinoDB® SQL* 指南*:* 参考》以及《*SinoDB® SQL* 指南*:* 语法》中的 CREATE DISTINCT TYPE 语句。

有关数据类型的更多信息，请参阅《*SinoDB® SQL* 指南*:* 参考》。

为用户定义的数据类型定义索引

对于内置数据类型，在为新的数据类型定义索引时，您可以缩短查询的响应时间。

在 SinoDB® 对以下对象使用索引的情况下，查询的响应时间可能会缩短：

- 用于连接两个表的列
- 用作查询的过滤器的列
- ORDER BY 或 GROUP BY 子句中的列
- 用作查询过滤器的函数的结果

有关何时通过内置数据类型的索引可以提高查询性能的更多信息，请参阅通过添加或移除索引来提高性能 在第280页。

SinoDB® 和 DataBlade® 模块提供各种不同类型的索引（又称为辅助访问方法）。辅助访问方法是一组数据库服务器函数，用于建立、访问和复制索引结构。这些函数封装了一些索引操作，例如如何扫描、插入、删除或更新索引中的节点。

要对用户定义的数据类型创建索引，您可以使用以下任一辅助访问方法：

- 一般 B 型树索引

  对于检索一个范围内的数据值的查询，B 型树索引很适用。有关更多信息，请参阅*B* 型树辅助访问方法 在第167页。
- R 型树索引

  对于在多维数据上搜索，R 型树索引很适用。有关更多信息，请参阅《*SinoDB® R-Tree* 索引用户指南》。
- DataBlade® 模块为新的数据类型提供的辅助访问方法

  支持某种确定类型数据的 DataBlade® 模块也会为该新数据类型提供新的索引。有关更多信息，请参阅使用 *DataBlade* 模块提供的索引 在第170页。

您可以在一个或多个列上对用户定义的函数的结果值创建函数索引。有关更多信息，请参阅使用函数索引 在第169页。

选择所需索引类型后，您可能也需要为辅助访问方法扩展一个运算符类。有关如何扩展运算符类的更多信息，请参阅《*SinoDB®* 用户自定义例程和数据类型开发者指南》。

B 型树辅助访问方法

SinoDB® 为数据库表中的列提供了一般 *B* 型树索引。在传统的关系数据库系统中，B 型树访问方法只处理内置数据类型，因此它只能比较内置数据类型的两个键。一般 B 型树索引是由 SinoDB® 提供的 B 型树的扩展版本，用于支持用户定义的数据类型。

> 提示： 有关 B 型树索引的结构以及如何估算 B 型树索引大小的更多信息，请参阅估算索引页 在第153页。

SinoDB® 使用一般 B 型树作为内置的辅助访问方法。此内置辅助访问方法以名称 btree 注册于 sysams 系统目录表中。使用 CREATE INDEX 语句（不带 USING 子句）创建索引时，数据库服务器会创建一般 B 型树索引。有关更多信息，请参阅《*SinoDB® SQL* 指南*:* 语法》中的 CREATE INDEX 语句。

> 提示： *SinoDB®* 也定义了其他辅助访问方法，即 R 型树索引。有关如何使用 R 型树索引的更多信息，请参阅《*SinoDB® R-Tree* 索引用户指南》。

B 型树索引的使用情况
对于检索一个范围内的数据值的查询，B 型树索引很适用。如果要建立索引的数据具有逻辑顺序，该顺序应用小于、大于和等于的概念，那么一般 B 型树索引是对数据建立索引的有用方法。

最初，一般 B 型树索引对所有内置数据类型支持关系运算符 (<,<=,=,>=,>)，并将数据按词典顺序进行排序。

如果对以下对象定义了一般 B 型树索引，优化器会考虑是否使用 B 型树索引来执行查询：

- 用于连接两个表的列
- 用作查询的过滤器的列
- ORDER BY 或 GROUP BY 子句中的列
- 用作查询过滤器的函数的结果

扩展一般 B 型树索引
最初，一般 B 型树可以对内置数据类型之一的数据建立索引，并按字母表顺序排序。但是，可以为其他某些数据类型扩展一般 B 型树。

您可以扩展一般 B 型树以支持以下数据类型的列和函数：

- 您想要 B 型树索引支持的用户定义的数据类型（不透明和单值数据类型）

  在这种情况下，您必须扩展一般 B 型树索引的缺省运算符类。
- 您想要用与一般 B 型树索引所用的字母表顺序不同的顺序来排序的内置数据类型

  在这种情况下，您必须定义不同于缺省一般 B 型树索引的运算符类。

运算符类是一组与非传统 B 型树索引相关联的函数（运算符）。有关运算符类的更多详细信息，请参阅选择索引的运算符类 在第170页。

确定可用的访问方法
为补充 SinoDB® 提供的内置 B 型树辅助访问方法，您的企业可能已安装可实现其他辅助访问方法的 DataBlade® 模块。如果存在其他访问方法，那么这些方法将定义在 sysams 系统目录表中。可以查询 sysams 系统目录以确定其他访问方法是否可用。

要确定数据库可用的辅助访问方法，可使用以下 SELECT 语句查询 sysams 系统目录表：

```
SELECT am_id, am_owner, am_name, am_type FROM sysams
   WHERE am_type = 'S';
```

am_type 列中的“S”值将访问方法标识为辅助访问方法。此查询返回以下信息：

- am_id 和 am_name 列标识辅助访问方法。
- am_owner 列标识访问方法的所有者。

在符合 ANSI 标准的数据库中，访问方法名称在用户的名称空间中必须唯一。访问方法名称总是以所有者开始，格式为 am_owner.am_name。

缺省情况下，SinoDB® 在 sysams 系统目录表中为两种辅助访问方法提供了以下定义，btree 和 rtree。

访问方法	am_id 列	am_name 列	am_owner 列
一般 B 型树	1	btree	“informix”
R 型树	2	rtree	“informix”

> 重要： sysams 系统目录表不包含用于内置主要访问方法的行。该主要访问方法是 SinoDB® 的内部访问方法，不需要在 sysams 中定义。然而，该内置主要访问方法始终是可用的。

如果在 sysams 系统目录表中找到了附加行（am_id 值大于 2 的行），那么数据库会支持附加的用户定义的访问方法。检查 am_type 列中的值，以确定用户定义的访问方法是主要访问方法还是辅助访问方法。

有关 sysams 系统目录表各列的更多信息，请参阅《*SinoDB® SQL* 指南: 参考》。有关如何确定数据库中可用的运算符类的信息，请参阅确定可用的运算符类 在第172页。

用户定义的辅助访问方法
如果小于、大于和等于的概念不适用于要建立索引的数据，那么您可以考虑使用用户定义的辅助访问方法，而不是内置辅助访问方法（即 B 型树索引）。您可以使用用户定义的辅助访问方法来访问其他索引结构，例如 R 型树索引。

如果数据库支持用户定义的辅助访问方法，您可以指定数据库在访问某一特殊索引时使用该访问方法。有关如何确定数据库定义的辅助访问方法的信息，请参阅确定可用的访问方法 在第168页。

要选择用户定义的辅助访问方法，请使用 CREATE INDEX 语句的 USING 子句。USING 子句指定辅助访问方法的名称用于您创建的索引。此名称必须列于 sysams 系统目录表中的 am_name 列，而且必须是辅助访问方法（sysams 的 am_type 列为“S”）。

您在 CREATE INDEX 的 USING 子句中指定的辅助访问方法必须已在 sysams 系统目录中定义。如果还未定义辅助访问方法，那么 CREATE INDEX 语句将失败。

如果省略 CREATE INDEX 语句的 USING 子句，那么数据库服务器将使用 B 型树索引作为辅助访问方法。有关更多信息，请参阅《*SinoDB® SQL* 指南: 语法》中的 CREATE INDEX 语句。

R 型树索引
SinoDB® 对包含空间数据（例如地图和图表）的列支持 *R* 型树索引的使用。R 型树索引使用树型结构，其节点存储指向更低层节点的指针。

R 型树的叶是数据页的集合，这些数据页存储 *n* 维形状。有关 R 型树索引的结构以及如何估算 R 型树索引大小的更多信息，请参阅《*SinoDB® R-Tree* 索引用户指南》。

使用函数索引
您可以基于一个或多个列中的实际值建立列索引。您也可以基于用户定义的函数从参数返回的一列或多列值创建函数索引。

重要: 数据库服务器对定义函数索引的用户定义例程 (UDR) 施加了以下限制：

- 参数不能是集合数据类型的列值。
- 函数不能返回大对象（包括内置类型 BLOB、BYTE、CLOB 和 TEXT）。
- 函数不能为 VARIANT 函数。
- 函数不能包含 SQL 的任何 DML 语句。
- 函数必须为 UDR 而非内置函数。但是，您可以创建调用并返回 SQL 内置函数中值的 SPL 包装程序。

要决定是使用列索引还是使用函数索引，请确定列索引对于要建立索引的数据是否为正确选择。对于典型查询，某些数据类型的列索引可能没有用处。例如，以下查询询问多少张图像较暗：

```
SELECT COUNT(*) FROM photos WHERE
darkness(picture) > 0.5
```

对 picture 数据本身的索引并不提高查询性能。小于、大于和等于的概念对图像数据类型而言并不是很有意义。然而，使用 darkness() 函数的函数索引却能提高查询性能。您也可以使用运行频率足够高的用户定义的函数，这样当您基于该函数的值建立索引时，就能够提高性能。

相关链接
管理索引 在第156页

什么是函数索引?
函数索引可以是 B 型树索引、R 型树索引或 DataBlade® 模块提供的用户定义索引类型。

如果建立了函数索引，数据库服务器会计算用户定义的函数的值，并将其作为键值存储于索引中。如果表中的数据更改并导致索引键的某个值也发生更改，数据库服务器就会自动更新函数索引。

对于那些既返回用户定义的数据类型（不透明和单值）的值，又返回内置数据类型的值的函数，可以使用函数索引。但是，如果函数返回简单大对象数据类型（TEXT 或 BYTE），那么您不能定义函数索引。

有关索引类型的更多信息，请参阅为用户定义的数据类型定义索引 在第167页。有关函数索引的空间需求的信息，请参阅估算索引页 在第153页。

相关链接

索引的类型 在第151页

何时使用函数索引？
优化器考虑是否使用函数索引来访问 SELECT 子句中或 WHERE 子句的过滤器中函数的结果。
创建函数索引
可以对用户定义的函数构建函数索引。用户定义的函数可以是外部函数或 SPL 函数。

要对用户定义的函数构建函数索引：

1. 如果用户定义的函数是外部函数，那么为其编写代码。
2. 使用 CREATE FUNCTION 语句在数据库中注册用户定义的函数。
3. 使用 CREATE INDEX 语句构建函数索引。

例如，要对 darkness() 函数创建函数索引：

1. 为用户定义的 darkness() 函数编写代码，该函数对数据类型执行操作，并返回一个十进制值。
2. 使用 CREATE FUNCTION 语句在数据库中注册用户定义的函数：

```
CREATE FUNCTION darkness(im image)
RETURNS decimal
EXTERNAL NAME '/lib/image.so'
LANGUAGE C NOT VARIANT
```

   在此示例中，您可以对函数索引使用缺省运算符类，因为 darkness() 函数的返回值是内置数据类型 DECIMAL。
3. 使用 CREATE INDEX 语句构建函数索引。

```
CREATE TABLE photos
(
    name char(20),
    picture image
...
);
CREATE INDEX dark_ix ON photos (darkness(picture));
```

   在此示例中，假设用户定义的数据类型 image 已在数据库中定义。

现在，如果指定 darkness() 函数为查询中的过滤器，那么优化器会考虑函数索引：

```
SELECT count(*) FROM photos WHERE
darkness(picture) > 0.5
```

也可以用用户定义的函数建立一个组合索引。有关更多信息，请参阅使用组合索引 在第281页。

使用 DataBlade® 模块提供的索引

DataBlade® 模块可以提供用户可访问的新数据类型。DataBlade® 模块也可以为新数据类型提供新索引。

有关每个 DataBlade® 模块提供的数据类型和函数的更多信息，请参阅该 DataBlade® 模块的用户指南。有关如何确定数据库中可用的索引类型的信息，请参阅确定可用的访问方法 在第168页。

选择索引的运算符类

大多数情况下，使用为辅助访问方法定义的缺省运算符。但是，如果想要按照另外的顺序排序数据，或者为用户定义的数据类型提供索引支持，那么您就必须扩展运算符类。

有关如何扩展运算符类的更多信息，请参阅《*SinoDB*® 用户自定义例程和数据类型开发者指南》。

运算符类

运算符类是一组与辅助访问方法相关联的函数的名称。这些函数允许辅助访问方法存储和搜索特殊数据类型的值。

数据库服务器的查询优化器会使用运算符类来确定索引是否能以最小的成本处理查询。运算符类向查询优化器指明了以下两点：

- 出现在 SQL 语句中的哪些函数可以使用给定的索引求值

  这些函数称为该运算符类的策略函数。
- 索引使用哪些函数来对策略函数求值

  这些函数称为该运算符类的支持函数。

利用运算符类提供的信息，查询优化器可以确定给定的索引是否适用于此查询。当以下条件为真时，查询优化器会考虑是否对给定查询使用该索引：

- 查询中特殊的一列或多列中存在一个索引。
- 对于存在的索引，那么在与之相关联的运算符类的策略函数中会有一个与查询中的一列或多列上的操作相匹配。

查询优化器复查表的可用索引，并将索引键与查询过滤器中指定的列相匹配。如果过滤器中的列与某个索引键相匹配，且过滤器中的函数是该运算符类中的策略函数，那么优化器在确定哪个查询计划具有最小执行成本时会包括该索引。通过这种方式，优化器就可以确定哪个索引能够以最小成本处理此查询。

SinoDB® 将有关运算符类的信息存储于 sysopclasses 系统目录表中。

辅助访问方法的策略和支持函数

SinoDB® 使用辅助访问方法的策略函数，以帮助查询优化器确定某个特定的索引对于数据类型的特定操作是否适用。

如果索引存在，并且过滤器中的运算符与运算符类中的策略函数之一相匹配，那么优化器会考虑是否为该查询使用此索引。

SinoDB® 使用辅助访问方法的支持函数建立和访问索引。这些函数并不由最终用户直接调用。如果查询过滤器中的一个运算符与策略函数之一相匹配，那么辅助访问方法将使用支持函数来遍历索引并获得结果。标识实际使用的支持函数将留给辅助访问方法来完成。

缺省运算符类

每种辅助访问方法均有一个与之相关联的缺省运算符类。缺省情况下，CREATE INDEX 语句会将缺省运算符类与索引关联起来。

例如，以下 CREATE INDEX 语句对 postalcode 列创建 B 型树索引，并自动将缺省 B 型树运算符类与该列关联起来：

```
CREATE INDEX postal_ix ON customer(postalcode)
```

有关如何为索引指定新的缺省运算符类的更多信息，请参阅用户定义的运算符类 在第173页。

内置 B 型树运算符类

内置辅助访问方法（即一般 B 型树）有一个缺省运算符类，称为 btree_ops，在 sysopclasses 系统目录表中定义。

缺省情况下，创建 B 型树索引时，CREATE INDEX 语句将 btree_ops 运算符类与该 B 型树索引关联起来。例如，以下 CREATE INDEX 语句在 orders 表的 order_date 列上创建一般 B 型树索引，并将 B 型树辅助访问方法的缺省运算符类与该索引关联起来：

```
CREATE INDEX orddate_ix ON orders (order_date)
```

SinoDB® 使用 btree_ops 运算符类来指定：

- 策略函数，可指示查询优化器查询中的哪些过滤器可以使用 B 型树索引
- 支持函数，以建立和搜索 B 型树索引

B 型树策略函数
btree_ops 运算符类为 btree 访问方法定义了策略函数的名称。

btree_ops 运算符类定义的策略函数为：

- lessthan (<)
- lessthanorequal (<=)
- equal (=)
- greaterthanorequal (>=)
- greaterthan (>)

这些策略函数都是运算符函数。也就是说，每个函数均与运算符相关联，在此处，是与关系运算符相关联。有关关系运算符函数的更多信息，请参阅《*SinoDB*® 用户自定义例程和数据类型开发者指南》。

在查询优化器检查包含某一列的查询时，也将检查是否已对该列定义一个 B 型树索引。如果存在这样的索引，并且如果该查询包含 btree_ops 运算符类支持的其中一个关系运算符，那么优化器可以选择 B 型树索引来执行查询。

B 型树支持函数
btree_ops 运算符类有一个支持函数，该支持函数是一个比较函数，称为 compare()。

compare() 函数为用户定义的函数，该函数返回一个整数值，以表明其第一个参数是等于、小于还是大于其第二个参数，如下所示：

- 当第一个参数等于 第二个参数时，返回值 0
- 当第一个参数小于第二个参数时，返回小于 0 的值。
- 当第一个参数大于第二个参数时，返回大于 0 的值。

B 型树辅助访问方法使用 compare() 函数遍历一般 B 型树索引中的节点。要在一般 B 型树索引中搜索数据值，辅助访问方法将使用 compare() 函数比较查询中的键值和索引节点中的键值。比较结果就可以确定辅助访问方法是否需要搜索下一层索引，或者该键是否驻留在当前节点。

一般 B 型树访问方法也会使用 compare() 函数来为一般 B 型树索引执行以下任务：

- 在建立索引前对键进行排序
- 确定一般 B 型树索引中键的线性顺序
- 对关系运算符进行求值
- 搜索索引中的数据值

数据库服务器使用 compare() 函数可对 SELECT 语句中的比较表达式进行求值。要为不透明数据类型提供对这些比较的支持，必须编写 compare() 函数。有关更多信息，请参阅《*SinoDB*® 用户自定义例程和数据类型开发者指南》。

使用 B 型树索引处理 SELECT 语句中的 ORDER BY 子句时，数据库服务器也会使用 compare() 函数。但是，如果索引不使用 btree-ops 运算符类，那么优化器将不使用该索引来执行 ORDER BY 操作。

确定可用的运算符类
通过查询 sysopclasses 系统目录表来确定您数据库可用的运算符类。

数据库服务器为内置辅助访问方法（即一般 B 型树索引）提供缺省的运算符类。另外，您的环境中可能还安装了可实现其他运算符类的 DataBlade® 模块。所有运算符类均在 sysopclasses 系统目录表中定义。

要确定您数据库可用的运算符类，可使用以下 SELECT 语句来查询 sysopclasses 系统目录表：

```
SELECT opclassid, opclassname, amid, am_name
FROM sysopclasses, sysams
WHERE sysopclasses.amid = sysams.am_id
```

此查询返回以下信息：

- opclassid 和 opclassname 列标识运算符类。
- am_id 和 am_name 列标识相关联的辅助访问方法。

缺省情况下，数据库服务器在 sysopclasses 系统目录表中为 btree_ops 和 rtree_ops 这两个运算符类提供以下定义。

访问方法	opclassid 列	opclassname 列	amid 列	am_name 列
一般 B 型树	1	btree_ops	1	btree
R 型树	2	rtree_ops	2	rtree

如果在 sysopclasses 系统目录表中发现有附加行（opclassid 值大于 2 的行），那么数据库将支持用户定义的运算符类。检查 amid 列中的值，以确定运算符类所属的辅助访问方法。

sysams 系统目录表中的 am_defopclass 列会存储辅助访问方法的缺省运算符类的运算符类标识符。要确定给定的辅助访问方法的缺省运算符类，可以执行以下查询：

```
SELECT am_id, am_name, am_defopclass, opclass_name
FROM sysams, sysopclasses
WHERE sysams.am_defopclass = sysopclasses.opclassid
```

缺省情况下，数据库服务器提供以下缺省运算符类。

访问方法	am_id 列	am_name 列	am_defopclass 列	opclass_name 列
一般 B 型树	1	btree	1	btree_ops
R 型树	2	rtree	2	rtree_ops

有关 sysopclasses 和 sysams 系统目录表各列的更多信息，请参阅《*SinoDB® SQL* 指南: 参考》。有关如何确定数据库中可用访问方法的信息，请参阅确定可用的访问方法 在第168页。

用户定义的运算符类

CREATE INDEX 语句指定用于索引的每个组件的运算符类。如果不指定运算符类，CREATE INDEX 语句将为您创建的辅助访问方法使用缺省运算符类。您可以为索引的组件使用用户定义的运算符类。

要为索引的特殊组件指定用户定义的运算符类，您可以执行以下操作：

- 使用数据库已定义的用户定义的运算符类。
- 如果数据库定义了 R 型树辅助访问方法，那么使用 R 型树运算符类。有关 R 型树的更多信息，请参阅《*SinoDB® R-Tree* 索引用户指南》。

如果对您要使用的辅助访问方法，数据库服务器支持多个运算符类，那么您可以为特定索引指定使用哪一个运算符类。有关如何确定数据库所定义的运算符类的信息，请参阅确定可用的运算符类 在第172页。

组合索引的每一部分都可以指定一个不同的运算符类。在建立索引时应选择运算符类。在 CREATE INDEX 语句中，指定要用在索引键规范中每个列或函数名称之后的运算符类的名称。每个名称均必须列于 sysopclasses 系统目录表的 opclassname 列，而且必须与索引使用的辅助访问方法相关联。

例如，如果数据库定义 abs_btree_ops 辅助访问方法以定义新的排序顺序，那么以下 CREATE INDEX 语句会指定 table1 表将 abs_btree_ops 运算符类与 col1_ix B 型树索引关联起来：

```
CREATE INDEX col1_ix ON table1(col1 abs_btree_ops)
```

在 CREATE INDEX 语句中指定的运算符类必须通过 CREATE OPCLASS 语句在 sysopclasses 系统目录中定义。如果该运算符类还未定义，那么 CREATE INDEX 语句将失败。有关如何创建运算符类的信息，请参阅《*SinoDB®* 用户自定义例程和数据类型开发者指南》。

第 8 章

锁定

数据库服务器会使用可影响并行和性能的锁。您可以监视和管理锁。

锁定

锁定是一种软件机制，可设置来防止其他程序使用某个资源。可以对单行或键、一页数据或索引键、整个表或整个数据库放置锁。

附加类型的锁可用于智能大对象。有关更多信息，请参阅智能大对象的锁 在第185页。

单个事务中锁定的最大行数或页数由已配置的锁总数控制。锁定行或页的表数不是显式控制的。

锁定粒度

锁保护的信息的级别和类型称为锁定粒度。锁定粒度会影响性能。

如果用户无法访问某一行或某个键，他可以等待其他用户对该行或键进行解锁。如果用户锁定了整页，那么更多用户要等待页中某行的可能性更高。

多个用户访问一组行的能力称为并发性。数据库管理员的目标就是提高并行性，从而在不牺牲单个用户性能的基础上提高整体性能。

键值锁

如果用户删除事务中的行，那么该行无法锁定，因为它不存在。但是，数据库必须以某种方式记录该行在事务结束之前还是存在的。数据库服务器使用键值锁来锁定已删除的行。

数据库服务器删除某行时，表索引中的键值并不马上移除。而是每一个键值都标记为已删除，并在键值上加上锁。

其他用户可能会遇到标记为已删除的键值。数据库服务器必须确定是否存在某个锁。如果存在，那么不进行删除，数据库服务器会将一条锁定错误发回至应用程序（或者如果用户执行 SET LOCK MODE TO WAIT，那么它将等待该锁定被解除）。

键值锁其中一个最重要的用处是确保唯一关键字在删除其事务的整个过程中，均保持唯一。如果没有这种保护机制，用户 A 可能会删除事务中的唯一关键字，而且用户 B 可能会在事务提交之前以相同的关键字插入某行。此种应用场合使用户 A 不可能进行回滚。键值锁防止了用户 B 在用户 A 的事务结束之前插入行。

页锁

创建不带 LOCK MODE 子句的表时，页锁定是缺省方式。用锁定页代替仅锁定行，数据库服务器就可以锁定包含该行的整个页。如果更新同一页上的多行，那么数据库服务器只使用该页上的一个锁。

插入或更新行时，数据库服务器在数据页上创建页锁。在某些情况下，当您简单使用 SELECT 语句读取某行时，数据库服务器将创建页锁。

当您插入、更新或删除某个关键字时（插入、更新或删除某行时会自动执行），那么数据库服务器会在包含索引中的关键字的页上创建锁。

重要： 索引页上的页锁相比数据页上的页锁，更能大大减少并行性。索引页很密集，其中含有很多关键字。通过锁定索引页将使大量潜在关键字对于其他用户不可用，直到您解除该锁定为止。使用页锁的表不支持 *Committed Read* 隔离 在第177页部分所述的 USELASTCOMMITTED 并行功能。

对于那些普通用户一次更改大量行的表来说，页锁非常有用。例如，如果一个 orders 表所包含的订单通常是被逐个插入和查询的，那么这样的订单表就不适合使用页锁。但是，如果 orders 表中包含旧订单，每晚均用白天下的所有订单进行更新，这样的表适合使用页锁。在这种情况下，用于访问表的隔离级别的类型是重要的。有关更多信息，请参阅隔离级别 在第177页。

表锁

在数据仓库环境中，查询可能更适合获取更大粒度的锁。例如，如果查询访问表中的大多数行，那么它获取较少的表锁时，与获取很多页锁或行锁相比，其效率会有所提高。

数据库服务器可以设置两类表锁：

- 共享锁

  其他所有用户均不能向表中写内容。
- 互斥锁

  其他所有用户均不能从表中读取，或者向表中写内容。

这两种表锁的另外一个重要区别在于放置的锁的实际数目：

- 在共享方式中，数据库服务器放置一个共享锁于表上，通知其他用户不能执行更新。另外，数据库服务器还为每个更新、删除或插入的行添加锁。
- 在互斥方式中，数据库服务器仅放置一个互斥锁于表上，而不管它更新了多少行。如果要更新表中的大多数行，请在表上放置互斥锁。

  重要： 表上的表锁能够从根本上降低更新并发性。在任何给定的时间，只有一项更新事务可以访问该表，并且该更新事务将所有其他事务均锁在外面。然而，多个只读事务仍能同时访问该表。在那些载入数据，然后由多个用户进行查询的数据仓库环境中，此行为很有用。

您可以将表在表级别锁定和其他级别锁定之间来回切换。对于在某些时间段使用数据仓库方式中的表而在其他时间段不使用的情形，这种切换锁定级别的能力很有用。

事务通过 LOCK TABLE 语句来指示数据库服务器对表使用表级别锁定。以下示例在表上放置互斥锁：

```
LOCK TABLE tab1 IN EXCLUSIVE MODE;
```

以下示例在表上放置共享锁：

```
LOCK TABLE tab1 IN SHARE MODE:
```

在某些情况下，数据库服务器会放置其自己的表锁。例如，如果隔离级别是 Repeatable Read，而且数据库服务器必须读取表的大部分，那么它将自动放置表锁，而不是设置行或页锁。数据库服务器创建或删除索引时，也会在表上放置表锁。

数据库锁

使用 DATABASE 语句打开数据库时，您可以在整个数据库上放置锁。数据库锁防止除当前用户以外的任何人读取或更新数据库。

以下语句打开并锁定销售数据库：

```
DATABASE sales EXCLUSIVE
```

配置锁模式

创建表时，缺省锁模式是 page。可以通过设置 IFX_DEF_TABLE_LOCKMODE 环境变量或 DEF_TABLE_LOCKMODE 配置参数以在创建或变更表时更改锁模式（从而提高或降低并发性）。

如果您知道大多数应用程序可能会从行锁模式中获益，那么可以执行以下操作：

- 在每条 CREATE TABLE 语句或 ALTER TABLE 语句中，使用 LOCK MODE ROW 子句。
- 将 IFX_DEF_TABLE_LOCKMODE 环境变量设置为 ROW，以便您以后在会话中创建的所有表均会使用 ROW，而无需在 CREATE TABLE 语句或 ALTER TABLE 语句中指定锁模式。
- 将 DEF_TABLE_LOCKMODE 配置参数设置为 ROW，以便您以后在数据库服务器中创建的所有表均会使用 ROW，而无需在 CREATE TABLE 语句或 ALTER TABLE 语句中指定锁模式。

如果使用 IFX_DEF_TABLE_LOCKMODE 环境变量或 DEF_TABLE_LOCKMODE 配置参数更改锁模式，那么现有表的锁模式将不受影响。现有表仍继续使用创建时定义的锁模式。

此外，如果先前将表的锁模式更改为 ROW，而随后执行了 ALTER TABLE 语句以改变表的其他某个特征（例如，添加列或者更改扩展数据块大小），那么您将无需指定锁模式。锁模式仍为 ROW，而不设置为缺省的 PAGE 方式。

您仍可以通过在 CREATE TABLE 语句或 ALTER TABLE 语句中指定 LOCK MODE 子句来重设个别表的锁模式。

以下列表显示了表上锁模式的优先顺序：

- 系统缺省的是页锁。如果未设置配置参数和环境变量，或者未在 SQL 语句中指定 LOCK MODE 子句，那么数据库服务器将使用此系统缺省值。
- 如果设置了 DEF_TABLE_LOCKMODE 配置参数，那么在未设置环境变量或者未在 SQL 语句中指定 LOCK MODE 子句时，数据库服务器将使用此值。
- 如果设置了 IFX_DEF_TABLE_LOCKMODE 环境变量，那么该值将覆盖 DEF_TABLE_LOCKMODE 配置参数和系统缺省值。如果未在 SQL 语句中指定 LOCK MODE 子句，那么数据库服务器将使用该值。
- 如果在 CREATE TABLE 语句或 ALTER TABLE 语句中指定 LOCK MODE 子句，那么该值将覆盖 IFX_DEF_TABLE_LOCKMODE 和 DEF_TABLE_LOCKMODE 配置参数以及系统缺省值。

将锁模式设置为等待

应用程序进程遇到锁时，数据库服务器的缺省行为是返回一条错误。然而，您可以执行 SQL 语句来将锁模式设置为等待。该操作指定在移除锁后应用程序进程才会继续。

要在锁释放之前暂挂当前进程，请执行以下 SQL 语句：

```
SET LOCK MODE TO WAIT;
```

也可以指定在发出错误之前进程要等待锁释放的最大秒数。在以下示例中，数据库服务器在发出错误之前会等待 20 秒：

```
SET LOCK MODE TO WAIT 20;
```

要返回到缺省行为（不等待锁），请执行以下语句：

```
SET LOCK MODE TO NOT WAIT;
```

使用 SELECT 语句的锁

数据库服务器放置的锁类型和持续时间取决于应用程序中设置的隔离级别、数据库方式（日志记录、非日志记录或 ANSI），以及 SELECT 语句是否位于更新游标中。这些锁能够影响整体性能，因为它们影响并行性。

隔离级别

在 SELECT 语句执行期间，放置在数据上的锁数目和持续时间取决于用户设置的隔离级别。隔离的类型会影响整体性能，因为它影响并行性。

在执行 SELECT 语句之前，可以使用 SET ISOLATION 语句（它是 ANSI SQL-92 标准的 SinoDB® 插件）或兼容 ANSI/ISO 的 SET TRANSACTION 语句设置隔离级别。两个语句之间的主要区别在于 SET ISOLATION 具有一个附加的隔离级别 Cursor Stability，并且不同于 SET ISOLATION 的另外一点是 SET TRANSACTION 在某个事务中不能多次执行。SET ISOLATION 语句是 ANSI SQL-92 标准的 SinoDB® 插件。SET ISOLATION 语句可以更改会话的持久隔离级别。

Dirty Read 隔离

Dirty Read 隔离（或 ANSI Read Uncommitted）级别不会在 SELECT 语句执行期间所访存的任何行上放置任何锁。Dirty Read 隔离非常适用于查询的静态表。

如果同时发生更新活动，那么请小心使用 Dirty Read 隔离。通过使用 Dirty Read，读取器可以读取尚未向数据库提交的行，该行可能在回滚期间被消除或更改。例如，考虑以下应用场合：

```
用户 1 启动事务。
用户 1 插入行 A。
用户 2 读取行 A。
用户 1 回滚行 A。
```

用户 2 读取行 A，几秒钟后用户 1 回滚该行。实际上，用户 2 读取了从未提交到数据库的行。回滚的未提交数据在应用程序中可能是个问题。

因为数据库服务器不会为查询检查或放置任何锁，Dirty Read 隔离向所有隔离级别提供了最优性能。但是，因为回滚的未提交数据可能有问题，所以使用 Dirty Read 隔离时请小心。

由于回滚的未提交数据问题只是事务问题，因此没有事务的数据库（从而不允许事务）使用 Dirty Read 作为缺省隔离级别。实际上，对于不具有事务日志记录的数据库，Dirty Read 是唯一允许的隔离级别。

Committed Read 隔离

带有 Committed Read 隔离（或者 ANSI Read Committed）隔离级别的阅读器将在返回某行之前检查锁。通过检查锁，阅读器不能返回任何未提交的行。

在 Committed Read 期间，数据库服务器不会真正对行加上任何锁。它只是简单地在内部锁表中检查所有现存的行。

如果日志方式不符合 ANSI，那么对于具有日志记录的数据库，Committed Read 为缺省隔离级别。对于使用不符合 ANSI 日志记录方式创建的数据库，Committed Read 为适用于大多数活动的隔离级别。对于符合 ANSI 标准的数据库，Repeatable Read 为缺省隔离级别。

减少 Committed Read 隔离级别冲突风险的方法

在 Committed Read 隔离级别中，如果当前会话无法获取锁或数据库服务器检测到死锁，那么其他会话持有的锁可能会导致 SQL 操作失败。（如果两个用户都保持锁定而每个用户又都想获取另一个用户拥有的锁时，将发生死锁。）SQL 的 SET ISOLATION COMMITTED READ 语句的 LAST COMMITTED 关键字选项可减少锁冲突的风险。

即使其他并发会话持有互斥锁，SQL 的 SET ISOLATION COMMITTED READ 语句的 LAST COMMITTED 关键字选项也将指示服务器返回行的最新提交版本。您可以将 LAST COMMITTED 关键字选项用于 B 型树和函数索引、支持事务日志记录的表，以及不具有页级别锁定或互斥锁的表。有关更多信息，请参阅《*SinoDB® SQL* 指南*:* 语法》中 SET ISOLATION 语句的信息。

对于使用事务日志记录创建的数据库，当使用 Dirty Read 或 Committed Read 隔离级别（或 Read Uncommitted 的 ANSI/ISO 级别或 Read Committed 的 ANSI/ISO 级别）的会话尝试读取并发会话持有共享锁的行时，您可以设置 USELASTCOMMITTED 配置参数来指定数据库服务器是否使用上次提交的数据版本，而不是等待锁释放。上次提交的数据版本是任何更新发生之前存在的数据版本。

如果没有为 USELASTCOMMITTED 配置参数或 USELASTCOMMITTED 会话环境变量设置任何值或 NONE 值，那么 COMMITTED READ 或 READ COMMITTED 隔离级别中的会话将等待任何互斥锁释放，除非 SQL 的 SET ISOLATION COMMITTED READ LAST COMMITTED 语句指示数据库服务器读取最新提交的数据版本。

只有发生并发冲突更新时，设置 USELASTCOMMITTED 配置参数以与 Committed Read 隔离级别一起运行才能够影响性能。发生并发冲突更新时，查询性能取决于事务的动态。例如，使用上次提交的数据版本的阅读器可能需要通过其他并发事务来撤销对行进行的更新。该情境涉及读取一个或多个日志记录，从而增加可以影响性能的 I/O 流量。

相关链接

《SinoDB 管理员参考》: USELASTCOMMITTED 配置参数

Cursor Stability 隔离

带有 Cursor Stability 隔离的阅读器要求当前取得的行上有一个共享锁。这个操作会确保在当前用户取回一个新行之前，其他用户不能更新该行。

在图 *31: 为游标稳定性加上的锁* 在第178页中游标的示例中，在 *fetch a row* 处，数据库服务器将释放上一行的锁，并对正在访存的行放置锁。在 *close the cursor* 处，服务器将释放最后一行的锁。

```
set isolation to cursor stability
declare cursor for SELECT * FROM customer
open the cursor
while there are more rows
   fetch a row
   do work
end while
close the cursor
```

图 31: 为游标稳定性加上的锁

如果不使用游标取得数据，Cursor Stability 隔离就与 Committed Read 的行为一致。实际上没有加上锁。

Repeatable Read 隔离

Repeatable Read 隔离（ANSI Serializable 和 ANSI Repeatable Read）是最严格的隔离级别。使用 Repeatable Read，数据库服务器会锁定事务持续期间检查到的（不仅仅是取得的）所有行。

图 *32: 为可重复读加上的锁* 在第178页中的示例显示对于可重复读，数据库服务器何时放置和释放锁。在 *fetch a row* 处，服务器对正在访存的行和为检索此行而检查的每一行放置锁。在 *close the cursor* 处，服务器将释放最后一行的锁。

```
set isolation to repeatable read
begin work
declare cursor for SELECT * FROM customer
open the cursor
while there are more rows
    fetch a row
    do work
end while
close the cursor
commit work
```

图 32: 为可重复读加上的锁

在检查多行的处理过程中，Repeatable Read 十分有用，但在事务期间不能做任何更改。例如，假设某个应用程序必须检查属于一个人的三个帐户的帐户结余。该应用程序首先得到第一个帐户的结余，然后得到第二

个。但是同时另外一个应用程序开始了一个事务，该事务为第三个帐户登入借方并为第一个帐户贷款。在原先的应用程序获得第三个帐户的帐户结余时，该帐户已被记入借方。但是，原先的应用程序没有记录第一个帐户的借方。

在使用 Committed Read 或 Cursor Stability 时，就会发生上述情况。而使用 Repeatable Read 就不会发生这种情况。由于初始应用程序在事务结束之前都会为每个接受检查的帐户保持一个读取锁定，因此第二个应用程序对第一个帐户的更改尝试将失败（或等待，这取决于 SET LOCK MODE）。

因为即使被检查的行被锁定，如果数据库服务器顺序读取该表的话，与查询结果无关的大量的行也会被锁定。因此，当数据库服务器可以用索引访问一个表时就会使用 Repeatable Read 隔离。如果存在索引，而优化器选择了顺序扫描，那么您可以使用强制使用索引的指令。但是，强制在查询路径中进行更改可能会对查询性能产生负面影响。

锁定非日志记录表

当您在事务中使用表时，数据库服务器不会对非日志记录表上的页或行加上锁。但是，可以锁定非日志记录表以防止其他用户在修改非日志记录表时出现并发性问题。

使用以下其中一种方法可以防止其他用户在修改非日志记录表时出现并发性问题：

- 在整个事务期间以互斥方式锁定表。
- 在整个事务期间使用 Repeatable Read 隔离级别。

重要： 非日志记录原始表用于数据的快速载入。您应该先将表更改为 STANDARD，然后在事务中使用该表或修改其中的数据。

更新游标

更新游标是一种特殊类型的游标，当行可能被潜在更新时，应用程序会用到它。更新游标使用可提升的锁，在其中，数据库服务器在应用程序访存行时对该行放置一个更新锁。应用程序使用更新游标和 UPDATE...WHERE CURRENT OF 来更新该行时，锁将更改为互斥锁。

更新锁位于应用程序访存的行上时，其他用户仍然可以查看该行。

在某些情况下，数据库服务器可能会对已检查但实际上尚未取得的行加上锁。是否发生此类行为取决于数据库服务器如何执行 SQL 语句。

更新游标的优点在于，您可以放心地查看某行，不用担心其他用户会在您查看的时候以及在您更新该行之前对该行进行更改或使用更新游标查看该行。

如果不更新该行，那么在您执行下一个 FETCH 语句或者关闭游标时，数据库服务器的缺省行为是释放更新锁。然而，如果执行带有 RETAIN® UPDATE LOCKS 子句的 SET ISOLATION 语句，那么在事务结束之前，数据库服务器都不会释放任何当前存在的或后续加上的更新锁。

图 *33: 释放更新锁时* 在第179页中的代码显示出数据库服务器何时使用游标放置和释放更新锁。在 *fetch row 1* 处，数据库服务器为第 1 行添加更新锁。在 *fetch row 2* 处，该服务器释放第 1 行的更新锁并为第 2 行添加更新锁。然而，在数据库服务器执行带 RETAIN® UPDATE LOCKS 子句的 SET ISOLATION 语句后，它不会释放任何更新锁，直到事务结束为止。在 *fetch row 3* 处，它对第 3 行放置更新锁。在 *fetch row 4* 处，它对第 4 行放置更新锁。在 *commit work* 处，服务器将释放第 2、3 和 4 行的更新锁。

```
declare update cursor
begin work
open the cursor
fetch row 1
fetch row 2
SET ISOLATION TO COMMITTED READ
    RETAIN UPDATE LOCKS
fetch row 3
fetch row 4
commit work
```

图 33: 释放更新锁时

在符合 ANSI 标准的数据库中，通常不需要更新游标，因为任何选择游标的作用与不带 RETAIN® UPDATE LOCKS 子句的更新游标的作用是一样的。

图 *34: 提升更新锁时* 在第180页中的代码显示了数据库服务器将更新锁提升为互斥锁。在 *fetch the row* 处，服务器对正在访存的行放置更新锁。在 *update the row* 处，服务器将该锁提升为互斥锁。在 *commit work* 处，它会释放该锁。

```
declare update cursor
begin work
open the cursor
fetch the row
do work
update the row (use WHERE CURRENT OF)
commit work
```

图 34: 提升更新锁时

要使用更新游标，请在应用程序中运行 SELECT FOR UPDATE。

使用 INSERT、UPDATE 和 DELETE 语句加上的锁

执行 INSERT、UPDATE 或 DELETE 语句时，数据库服务器将使用互斥锁。互斥锁意味着在数据库服务器移除锁之前，其他用户不能更新或删除项目。

此外，除非其他用户使用 Dirty Read 隔离级别，否则他们不能查看行。

数据库服务器何时移除互斥锁取决于数据库是否支持事务日志记录：

- 如果数据库支持日志记录，那么数据库服务器会在事务完成（提交或回滚）时移除所有互斥锁。
- 如果数据库不支持日志记录，那么数据库服务器会在 INSERT、MERGE、UPDATE 或 DELETE 语句完成后立即移除所有互斥锁，但如果锁所在的行当前正在被访存到更新游标中，那么另当别论。

  在此情况下，在对行执行访存操作期间会保留锁定，但仅限于在服务器访存下一行之前，或者在服务器通过将锁提升为互斥锁来更新当前行之前。

在非日志记录数据库中，为更新而访存的列上的可提升更新锁可在最初创建锁的 INSERT、MERGE、UPDATE 或 DELETE 语句仍在运行时，通过对数据库执行 DDL 操作进行释放。要降低并发会话修改已解锁的行时发生数据损坏的风险，请限制那些对支持事务日志记录的数据库使用可提升更新锁的操作。

内部锁表

数据库服务器在内部锁表中存储锁。当数据库服务器读取一行时，它会检查锁表中是否列出了该行或与该行相关联的页、表或数据库。如果锁表中已列出，数据库服务器也必须检查锁类型。

下表显示锁表可包含的锁类型。

锁类型	描述	通常用于添加锁的语句
S	共享锁	SELECT
X	互斥锁	INSERT, UPDATE, DELETE
U	更新锁	更新游标中的 SELECT
B	字节锁	任何更新 VARCHAR 列的语句

只有在缩小 VARCHAR 列中数据值的大小时，才会生成字节锁。只有前滚和回滚操作才存在字节锁，因此只有在处理一个使用日志记录的数据库时，才创建字节锁。只有在使用行级别锁定时，字节锁才会出现在 onstat -k 输出中；否则，字节锁就会与页锁合并。

另外，锁表还可能存储意向锁，这是一种与前面显示的相同的锁类型。在某些情况下，用户可能需要注册要锁定某个项目的可能意向，从而使其他用户不能对该项目加上锁。

根据操作类型和隔离级别，数据库服务器可能继续读取该行并对该行放置自己的锁，或者可能等待锁被释放（如果用户执行了 SET LOCK MODE TO WAIT）。下表显示了在另一个用户保持着某种类型的锁定时，一个用户可以加上的锁。例如，如果一个用户对某个项目保持着互斥锁定，而另一个用户请求任何类型的锁（互斥、更新或者共享）时，均会收到一条错误。

	保持 X 锁定	保持 U 锁定	保持 S 锁定
请求 X 锁定	否	否	是
请求 U 锁定	否	否	是
请求 S 锁定	否	是	是

监视锁

可以分析有关锁的信息并通过查看包含存储锁的内部锁表中的信息来监视锁。

使用 onstat -k 查看锁表。图 *35: onstat -k* 输出 在第181页显示了 onstat -k 输出的示例。

```
Locks
address  wtlist   owner    lklist   type     tblsnum  rowid    key#/bsiz
300b77d0 0        40074140 0        HDR+S    10002    106        0
300b7828 0        40074140 300b77d0 HDR+S    10197    123        0
300b7854 0        40074140 300b7828 HDR+IX   101e4    0          0
300b78d8 0        40074140 300b7854 HDR+X    101e4    102        0
 4 active, 5000 total, 8192 hash buckets
```

图 35: onstat -k 输出

在此示例中，用户正在向表中插入一行。该用户保持以下锁定（以所显示的顺序描述）：

- 对数据库的共享锁
- 对 systables 系统目录表中某行的共享锁
- 对表的意向互斥锁
- 对行的互斥锁

要确定锁应用到的表，请执行以下 SQL 语句。对于 *tblsnum*，替换上 onstat -k 输出中的 tblsnum 字段显示的值。

```
SELECT *
   FROM SYSTABLES
   WHERE HEX(PARTNUM) = "tblsnum";
```

其中，*tblsnum* 为 onstat -k 返回的修改值。例如，如果 onstat -k 返回 10027f，那么 *tbslnum* 为 0x0010027F。

您也可以查询 sysmaster 数据库中的 syslocks 表，以获取有关每个活动锁的信息。syslocks 表包含以下各列。

列	描述
dbsname	保持锁定的数据库
tabname	保持锁定的表的名称
rowidlk	保持锁定的行的标识符（0 指示表锁定。）
keynum	行的关键字数

列	描述
type	锁类型
owner	锁所有者的会话标识符
waiter	锁定的第一个等待者的会话标识符

配置和管理锁使用情况

LOCKS 配置参数指定内部锁表的初始大小。如果数据库服务器增大锁表的大小，那么应该增大 LOCKS 配置参数的大小。

有关如何确定 LOCKS 配置参数的初始值的信息，请参阅 *LOCKS* 配置参数和内存利用率 在第65页。

如果会话所需的锁数超过了在 LOCKS 配置参数中设置的值，那么数据库服务器会尝试使锁表的大小增加一倍。每次锁表溢出时（当所需锁数量大于锁表的当前大小时），数据库服务器就会增加锁表大小，最多可高达 99 次。每次数据库服务器增加锁表的大小时，该服务器都会尝试将锁表大小翻倍。但是，服务器会将每次的实际增加值限制为不超过 表 *11: 32* 位和 *64* 位平台上的最大锁数 在第182页中显示的已添加锁的最大值。数据库服务器增加锁表大小满 99 次以后，该服务器将不再增加锁表的大小，这时需要锁的应用程序将收到一条错误消息。

32 位和 64 位平台上允许的最大锁数

下表显示了允许的最大锁数。

表 11: 32 位和 64 位平台上的最大锁数

平台	最大初始锁数	最大动态锁表扩展数	每个锁表扩展添加的最大锁数	允许的最大锁数
32 位	8,000,000	99	100,000	8,000,000 + (99 x 100,000)
64 位	500,000,000	99	1,000,000	500,000,000 + (99 x 1,000,000)

查看与锁表大小增加相关的消息

每次数据库服务器增加锁表的大小时，该服务器将在消息日志文件中放置一条消息。您应该定期监视消息日志文件，如果看到数据库服务器已增加锁表大小，那么应增加 LOCK 配置参数的大小。

监视锁数不足错误

要监视应用程序接收到锁数不足错误的次数，请查看 onstat -p 输出中的 ovlock 字段。您也可以从 sysmaster 数据库中的 sysprofile 表看到类似的信息。以下各行包含相关统计信息。

行	描述
ovlock	会话尝试超过最大锁数的次数
lockreqs	会话请求锁的次数
lockwts	会话等待锁的次数

检查应用程序如何使用锁

如果数据库服务器正在使用大量的锁，那么您可以检查各个应用程序如何使用锁，如下所示：

1. 使用 onstat -u 监视会话可查看某个特别的用户是否正在使用大量的锁（locks 列中的值特别大）。
2. 如果的确有某个特别用户使用大量锁，那么检查应用程序中的 SQL 语句，以确定是应该锁定表，还是应该使用单独的行锁或页锁。

表锁比单独的行锁效率要高，但降低了并行性。

减少加在表上的锁数的一种方法是将表由使用行锁改为使用页锁。然而，页锁降低了表的整体并行性，这将影响到性能。

您也可以按照互斥方式通过锁定表来减少放置在该表上的锁数。

相关链接

LOCKS 配置参数和内存利用率 在第65页

监视锁定等待和锁定错误

可以查看有关会话、锁使用情况和锁定等待的信息。

如果应用程序执行 SET LOCK MODE TO WAIT，那么数据库服务器将等待锁被释放，而不是返回一条错误。异乎寻常的长时间等待锁会给用户留下应用程序挂起的印象。

在图 *36: 显示锁使用情况的 onstat -u 输出* 在第183页中，onstat -u 输出显示会话标识符 84 正在等待锁定（Flags 字段第一列中的 L）。要找出锁的所有者，请使用 onstat -k 命令。

```
onstat -u

Userthreads
address  flags   sessid user     tty    wait    tout locks nreads nwrites
40072010 ---P--D 7      informix -         0       0     0     35      75
400723c0 ---P--- 0      informix -         0       0     0      0       0
40072770 ---P--- 1      informix -         0       0     0      0       0
40072b20 ---P--- 2      informix -         0       0     0      0       0
40072ed0 ---P--F 0      informix -         0       0     0      0       0
40073280 ---P--B 8      informix -         0       0     0      0       0
40073630 ---P--- 9      informix -         0       0     0      0       0
400739e0 ---P--D 0      informix -         0       0     0      0       0
40073d90 ---P--- 0      informix -         0       0     0      0       0
40074140Y-BP---81   lsuto    4  50205788       0     4    106     221
400744f0 --BP--- 83     jsmit    -         0       0     4      0       0
400753b0 ---P--- 86     worth    -         0       0     2      0       0
40075760 L--PR--84     jones    3  300b78d8   -1     2      0       0
 13 active, 128 total, 16 maximum concurrent

onstat -k

Locks
address  wtlist   owner    lklist   type    tblsum rowid  key#/bsiz
300b77d0 0        40074140 0        HDR+S   10002  106     0
300b7828 0        40074140 300b77d0 HDR+S   10197  122     0
300b7854 0        40074140 300b7828 HDR+IX  101e4  0       0
300b78d84007576040074140300b7854 HDR+X  101e4  100     0
300b7904 0        40075760 0            S   10002  106     0
300b7930 0        40075760 300b7904     S   10197  122     0
 6 active, 5000 total, 8192 hash buckets
```

图 36: 显示锁使用情况的 onstat -u 输出

要找到会话标识符 84 正在等待的锁的所有者，请执行下列操作：

1. 在 onstat -u 输出的 wait 字段中，得到锁的地址 (300b78d8)。
2. 在 onstat -k 输出的 Locks address 字段中，找到该地址 (300b78d8)。

   onstat -k 输出中，该行的 owner 字段包含了用户线程的地址 (40074140)。
3. 在 onstat -u 输出的 Userthreads 字段中找到该地址 (40074140)。

   在 onstat -u 输出中，该行的 sessid 字段包含拥有此锁的会话标识符 (81)。

要消除争用问题，您可以让用户平稳地退出应用程序。如果这种方法还无法解决问题，您可以停止应用程序进程，或者使用 onmode -z 移除会话。

监视可用锁数

您可以通过查看 onstat -L 命令输出来查找可用锁列表上的当前可用锁数。

相关链接

《*SinoDB* 管理员参考》*: onstat -L* 命令*:* 显示可用锁的数量

监视死锁

可以监视死锁。当两个用户均保持锁定，并且每一个都想获取对方拥有的锁时，就会发生死锁。

例如，用户 pradeep 对行 10 保持锁定。用户 jane 在行 20 上保持一个锁定。假设 jane 想要在行 10 上放置锁，而 pradeep 想要在行 20 上放置锁。如果两个用户均执行 SET LOCK MODE TO WAIT，那么他们可能永远相互等待下去。

SinoDB® 使用锁表自动检测死锁，并在他们出现之前就将他们停止。在一个锁被获准之前，数据库服务器会检查每个用户的锁列表。如果一个用户对请求者想要锁定的资源保持着锁定，那么数据库服务器会为该用户遍历锁等待列表，以查看该用户是否正在等待请求者保持的任何锁定。如果答案为是，那么请求者会收到一条死锁错误。

如果应用程序频繁地更新同一行，死锁错误可能是不可避免的。但是，某些应用程序可能总在相互发生争用。请检查正在产生大量死锁的应用程序，并试着在不同的时间执行这些应用程序。

要监视死锁数，请使用 onstat -p 输出中的 deadlks 字段。

在分布式事务中，数据库服务器不会检查其他数据库服务器系统的锁表，因此无法在死锁发生前就检测到。但是，您可以设置 DEADLOCK_TIMEOUT 配置参数。DEADLOCK_TIMEOUT 会在返回一条错误之前指定数据库服务器将等待远程数据库服务器响应的秒数。尽管除分布式死锁以外，还有其他原因也可能导致延迟，但此机制可以避免一个事务无限期地挂起。

要监视分布式死锁超时的个数，请使用 onstat -p 输出中的 dltouts 字段。

监视会话使用的隔离级别

onstat -g ses 和 onstat -g sql 输出显示会话当前正在使用的隔离级别。

下表汇总了 onstat -g ses 和 onstat -g sql 输出中 IsoLvl 列内的值。

DR
: Dirty Read

CR
: Committed Read

CS
: Cursor Stability

CRU
: 使用 RETAIN® UPDATE LOCKS 的 Committed Read

CSU
: 使用 RETAIN® UPDATE LOCKS 的 Cursor Stability

DRU
: 使用 RETAIN® UPDATE LOCKS 的 Dirty Read

LC
: Committed Read, Last Committed

RR
: Repeatable Read

如果出现大量锁争用，请检查会话的隔离级别，以确保其对于应用程序来说是合适的。

智能大对象的锁

智能大对象有几个唯一的锁定行为，因为它们的列通常比表中其他列要大的多。

本节将讨论以下唯一行为：

- 智能大对象的锁类型
- 字节范围锁定
- 锁提升
- 智能大对象的 Dirty Read 隔离级别

智能大对象的锁类型

数据库服务器使用以下粒度级别之一以锁定智能大对象：

- 智能大对象空间块头分区
- 智能大对象
- 智能大对象的字节范围

缺省锁定粒度是在智能大对象级别。换言之，当您更新智能大对象时，在缺省情况下，数据库服务器会锁定正在更新的智能大对象。

只有在数据库服务器提升对智能大对象的锁定时，才会发生对智能大对象空间块头分区的锁定。有关更多信息，请参阅锁提升 在第187页。

字节范围锁定

可以只锁定智能大对象的特定字节范围，而不是锁定整个智能大对象。

字节范围锁定很有用，因为它允许多个用户同时更新同一个智能大对象，只要他们更新的是不同部分。而且，用户在其他用户更新或读取相同智能大对象的不同部分时，也能读取该智能大对象的某一部分。

图 37: 字节范围锁定示例 在第185页显示了在单个智能大对象上放置两个锁。第一个锁放置在字节 2、3 和 4 上，第二个锁单独放置在字节 6 上。

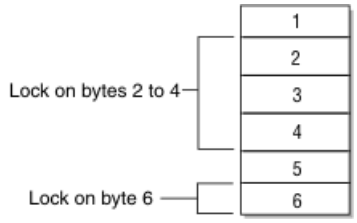

图 37: 字节范围锁定示例

数据库服务器如何管理字节范围锁

数据库服务器管理锁表中的字节范围锁，其方式与管理对行、页和表加上其他锁的方式相似。但是，锁表也必须存储字节范围。

如果对当前已锁定的字节范围的相邻字节范围放置第二个锁，那么数据库服务器会将这两个锁合并为整个范围上的一个锁。

如果用户持有图 37: 字节范围锁定示例 在第185页中显示的锁，而该用户又请求给字节 5 加上锁，那么数据库服务器会将加在字节 2 到字节 6 上的锁合并为一个锁。

同样，如果用户给字节范围锁中包括的一部分字节解锁，数据库服务器可能会将该锁拆分为多个字节范围锁。在图 37: 字节范围锁定示例 在第185页中，用户可能对字节 3 解锁，这样会导致数据库服务器将用于字节 2 到字节 4 的一个锁更改为用于字节 2 的一个锁和用于字节 4 的一个锁。

使用字节范围锁

缺省情况下，数据库服务器会给整个智能大对象加上锁。然而，可以启用字节范围锁定。

要使用字节范围锁，必须执行以下操作之一：

- 使用 onspaces 实用工具，为存储智能大对象的智能大对象空间设置字节范围锁定。以下示例为新的智能大对象空间设置了字节范围锁定：

```
onspaces -c -S slo -g 2 -p /ix/9.2/liz/slo -o 0 -s 1000
   -Df LOCK_MODE=RANGE
```

  如果将智能大对象空间缺省锁模式设置为字节范围锁定，那么数据库服务器在更新任何存储于智能大对象空间中的智能大对象时，只锁定必需的字节。
- 要在打开智能大对象时为其设置字节范围锁定，可以使用以下其中一种方法：
  - 在 *DB-Access* 中*:* 在 mi_lo_open() DataBlade® API 函数中设置 MI_LO_LOCKRANGE 标志。
  - 在 *ESQL/C* 中*:* 在 ifx_lo_open() SinoDB® ESQL/C 函数中设置 LO_LOCKRANGE 标志。对各个智能大对象设置字节范围锁定时，数据库服务器在选择或更新智能大对象时，只会隐式锁定必要的字节。
- 要显式锁定一个字节范围，可以使用以下其中一个函数：
  - 对于 *DB-Access:* mi_lo_lock()
  - 对于 *ESQL/C:* ifx_lo_lock()

    这些函数可锁定您为智能大对象指定的字节范围。如果使用其中一个函数指定互斥锁，那么 UPDATE 语句在更新已锁定字节时将不在智能大对象上放置锁。

    事务结束时，数据库服务器会释放用 mi_lo_lock() 或 ifx_lo_lock() 放置的互斥字节范围锁。数据库服务器基于与使用 SELECT 语句放置的锁相同的规则以及隔离级别来释放使用 mi_lo_lock() 或 ifx_lo_lock() 放置的共享字节范围锁。也可以使用以下其中一个函数来释放共享字节范围锁：
  - 对于 *DB-Access:* mi_lo_unlock()。有关 DataBlade® API 函数的更多信息，请参阅《*SinoDB® DataBlade® API* 程序员指南》。
  - 对于 *ESQL/C:* ifx_lo_unlock()。有关 SinoDB® ESQL/C 函数的更多信息，请参阅《*SinoDB® ESQL/C* 程序员指南》。

监视字节范围锁
可以使用 onstat -k 来列出所有字节范围锁。使用 onstat -K 命令列出字节范围锁以及字节范围锁的所有等待者。

图 *38: onstat -k* 输出中的字节范围锁 在第186页显示了 onstat -k 输出中节选的部分内容。

```
Byte-Range Locks
rowid/LOid   tblsnum  address  status   owner    offset   size     type
104          200004   a020e90  HDR
[2, 2, 3]             a020ee4  HOLD     a1b46d0  50       10           S
202          200004   a021034  HDR
[2, 2, 5]             a021088  HOLD     a1b51e0  40       5            S
102          200004   a035608  HDR
[2, 2, 1]             a0358fc  HOLD     a1b4148  0        500          S
                      a035758  HOLD     a1b3638  300      100          S
 21 active, 2000 total, 2048 hash buckets
```

图 38: onstat -k 输出中的字节范围锁

字节范围锁在 onstat -k 输出中生成以下信息。

列	描述
rowid	包含已锁定的智能大对象的行标识符
LOid	以下三个值：智能大对象空间数、块数和顺序数（代表在块中位置的值）
tblsnum	保存智能大对象的表空间的数量
address	锁的地址

列	描述
status	锁的状态 HDR 是一个占位符。HOLD 代表在 owner 列中指定的用户拥有该锁。WAIT（只在使用 onstat -K 时显示）表示在“owner”列中指定的用户正在等待该锁。
owner	所有者（或等待者）的地址 用 onstat -u 中的地址交叉引用此值。
offset	在锁定字节的智能大对象中的偏移量
size	锁定的字节数，从“offset”列中的值开始
type	S（共享锁）或 X（互斥锁）

设置字节范围锁定的锁数量
使用字节范围锁定时，数据库服务器可以使用多个锁，因为一个智能大对象上可能有多个锁。尽管锁表在用完空间后仍会增长，但您可能希望增加 LOCKS 配置参数的值以与锁使用量匹配，这样数据库服务器无需动态分配更多的空间。

确保监视 onstat -k 所用的锁数，这样可确定是否需要增加 LOCKS 配置参数的值。

锁提升

数据库服务器使用锁提升来减少对智能大对象所保持的锁定总数。太多的锁会导致性能降低，因为数据库服务器会频繁地搜索锁表，以确定某个对象上是否存在一个锁。

如果某个事务持有的锁数超过了数据库服务器分配的当前锁数的 33%，数据库服务器就会试图将所有现有字节范围锁提升为智能大对象上的一个锁。

如果一个用户对一个智能大对象（不是对智能大对象的字节范围）所持有的锁定数等于或超过锁表当前容量的 10%，那么数据库服务器就会尝试将所有智能大对象锁提升为对智能大对象头分区的一个锁。对于那些更新、载入或删除大量智能大对象的应用程序，这种锁提升提高了性能。例如，如果不使用锁提升，一个删除数百万个智能大对象的事务就会消耗掉整个锁表。锁提升算法中内置了死锁避免功能。

可以通过 rowid 列中的 0 以及表空间编号（该编号高序的最开始的一个半字节与存储智能大对象的数据库空间编号相对应）来标识 onstat -k 中的智能大对象的头分区。例如，如果列出的表空间数为 0x200004（高位 0 已截去），那么数据库空间数 2 等于 onstat -d 中列出的数据库空间数。

即使数据库服务器试图提升一个锁，它可能也无法做到。例如，数据库服务器可能无法将字节范围锁提升为一个智能大对象锁，因为其他用户对同一个智能大对象加上了字节范围锁。如果数据库服务器不能提升一个字节范围锁，它就不更改锁，处理照常继续进行。

Dirty Read 隔离级别和智能大对象

可以对智能大对象使用 Dirty Read 隔离级别。

有关 Dirty Read 如何影响一致性的信息，请参阅 *Dirty Read* 隔离 在第177页。

可以按照以下其中一种方法对智能大对象设置 Dirty Read 隔离级别：

- 使用 SET TRANSACTION MODE 或 SET ISOLATION 语句。
- 在以下某个函数中使用 LO_DIRTY_READ 标志：
  - 对于 *DB-Access:* mi_lo_open()
  - 对于 *ESQL/C:* ifx_lo_open()

如果智能大对象的一致性不重要，但是行中其他列的一致性比较重要，那么您可以将隔离级别设置为 Committed Read、Cursor Stability 或 Repeatable Read，然后使用 LO_DIRTY_READ 标志来打开该智能大对象。

第 9 章

分段存储准则

关系数据库系统中导致低性能最常见的原因之一是对驻留在单个 I/O 设备上的数据进行争用。对于高使用率的表，适当的分段存储可以显著减少 I/O 争用。以下主题将讨论在使用表分段存储时涉及的性能注意事项。

数据库服务器支持表分段存储（也称为分区），这使您能够将单个表的数据存储在多个磁盘设备上。分段存储使您可以根据某些算法或方案定义表内的行或索引键的组。可以将每个组或分段（也称为分区）存储在与特定物理磁盘关联的单独数据库空间内。

有关分段存储和并行执行的信息，请参阅并行数据库查询 *(PDQ)* 在第251页。

有关分段存储概念和方法的简介，请参阅《*SinoDB*® 数据库设计和实现指南》。有关管理分段的 SQL 语句的信息，请参阅《*SinoDB*® *SQL* 指南: 语法》。

规划分段存储策略

可以决定数据库的分段存储目标并修改策略以满足该目标。

分段存储策略由以下两个部分组成：

- 分布方案，指定如何将行分组到分段

  在 CREATE TABLE、CREATE INDEX 或 ALTER FRAGMENT 语句的 FRAGMENT BY 子句中，指定分布方案。
- 放置分段的数据库空间集

  在这些 SQL 语句的 IN 子句（存储选项）中，指定数据库空间集。

要制订分段存储策略，请执行以下操作：

1. 确定主要的分段存储目标，该目标在很大程度上取决于访问表的应用程序的类型。
2. 根据主要分段存储目标制定以下决策：
   - 是将表数据或表索引进行分段，还是两者均进行分段
   - 对于表而言，行或索引关键字的理想分布是什么
3. 选择基于表达式的分布方案或循环分布方案：
   - 如果选择基于表达式的分布方案，那么必须设计合适的分段表达式。
   - 如果选择循环分布方案，那么由数据库服务器确定哪些行放入特定的分段。

   有关更多信息，请参阅分布方案 在第191页。
4. 要完成分段存储策略，您必须确定分段数和分段位置：
   - 分段数取决于主要分段存储目标。
   - 分段位置取决于配置中的可用磁盘数。

当您规划分段存储策略时，请注意这些空间和页问题：

- 尽管 2 KB 的页上可以放置 4 TB 的块，但由于行标识格式的限制，数据库空间中只能使用 32 GB。

- 对于分段表，所有分段均必须使用相同的页大小。
- 对于分段索引，所有分段均必须使用相同的页大小。
- 一个表可在一个数据库空间中，而该表的索引可在另一个数据库空间中。这些数据库空间无需具有相同的页大小。

分段存储目标

可以分析应用程序和工作负载以确定分段存储目标，并确定分段存储目标之间是否平衡。

分段存储目标可以包括：

- 对于单个查询的性能提高

  要提高单个查询的性能，可适当对表进行分段并设置与资源相关的参数以指定系统资源的使用（内存和 CPU 虚拟处理器等）。
- 减少查询和事务之间的争用

  如果数据库服务器主要用于联机事务处理 (OLTP)，并只偶尔用于决策支持查询，那么在针对表的并发查询执行索引扫描以返回某些行时，您可以经常使用分段存储以减少争用。
- 增加数据可用性

  合理地进行数据库空间分段存储，可以在设备发生故障时提高数据的可用性。故障设备上的表分段可以快速还原，而且其他分段仍可访问。
- 数据载入性能的提高

  您可以使用带有 ATTACH 子句的 ALTER FRAGMENT ON TABLE 语句将数据快速添加到非常庞大的表中。有关更多信息，请参阅提高连接和拆离分段的操作性能 在第201页。

分段表的性能主要受以下因素控制：

- 用于为分段分配磁盘空间的存储选项（如考虑物理分段存储因素 在第191页中所述）
- 用于将行分配到单个分段的分布方案（如分布方案 在第191页中所述）

通过分段存储策略提高查询性能

如果分段存储的主要目标是提高单个查询的性能，那么请尝试将表的所有行平均分布到不同的磁盘。如果数据库服务器无需等待从一个行数多于其他分段的表分段中检索数据，就可以减少总体的查询完成时间。

如果查询通过对表的较大部分执行顺序扫描来访问数据，那么只需对表的行进行分段，不要对索引分段。如果对索引分段，那么查询必须穿过分段边界以访问数据，这样查询性能可能会比不分段时还要低。

如果查询通过读取索引来访问数据，您可以通过对索引和表使用相同的分布方案来提高性能。

如果使用循环分段存储，那么不要对索引分段。可以考虑将该索引放在与其他表分段分开的数据库空间中。

有关更多提高查询性能的信息，请参阅分段消除的查询表达式 在第198页和提高个别查询性能 在第265页。

减少查询和事务之间的争用

分段存储可以减少多个查询和 OLTP 应用程序使用的表中的数据争用。在很多针对某个表的同步查询执行索引扫描以返回很少几行时，分段存储常常可以减少争用。

对于属于这种载入类型的表而言，应使用分布方案对索引键和数据行都进行分段，以便每个查询可以从扫描中移除不需要的分段。使用基于表达式的分布方案。有关更多信息，请参阅消除分段的分布方案 在第197页。

要对表进行分段以减少争用，应从研究哪些查询访问表的哪些部分入手，然后对数据分段。这样一些查询被路由到一个分段，而其他查询则访问不同的分段。数据库服务器在评估该表的分段存储规则时，会执行此路由。最后，在单独的磁盘中存储分段。

能否成功地减少争用，取决于您对表中数据的分布以及对表查询的规划的了解程度。例如，如果表查询的分布设置使访问所有行的比例大致相同，那么可以试着在各分段间平均分布行。但是，如果对某些值的访问比例要高于其他值，那么可以通过将这样的行分布到多个分段中以平衡访问比例，从而弥补这种差异。有关更多信息，请参阅设计基于表达式的分布方案 在第193页。

增加数据可用性
当您将表和索引分段分布到不同的磁盘或者设备上时，将提高数据在磁盘或设备发生故障情况下的可用性。数据库服务器此时仍然允许访问存储于正常运行的磁盘或设备上的分段。

这种可用性对以下类型的应用程序具有重要意义：

- 不需要访问不可用分段的应用程序

  不要求数据库服务器访问不可用分段中的数据的查询仍然可以成功地从可用的分段中检索数据。例如，如果分布表达式使用了单个列，那么数据库服务器可以在不访问分段的情况下判断出分段中是否包含某行。如果查询只访问包含在可用分段中的行，那么即使表中有些数据不可用，查询也可以成功完成。有关更多信息，请参阅设计基于表达式的分布方案 在第193页。
- 接受数据不可用性的应用程序

  对于一些应用程序，其设计方法可能使它们能够接受分段中数据的不可用性，并要求它们具有检索可用数据的能力。为指定哪些分段可以跳过，这些应用程序可以在执行查询之前执行 SET DATASKIP 语句。或者，数据库服务器管理员也可以使用 onspaces -f 选项，以指定哪些分段不可用。

如果分段存储的目标是数据可用性的提高，那么可以同时对表行和索引键进行分段，这样如果磁盘驱动器发生故障，一些数据仍将可用。如果应用程序必须总是可以访问某个数据子集，那么可将子集中的那些行一起放入同一个镜像的数据库空间中。

增加备份与还原的粒度
在决定如何在磁盘间分布数据库空间时，必须考虑备份与还原因素。

要考虑的备份与还原因素为：

- 数据可用性。在决定何处放置表或分段时，请记住如果包含数据库空间的设备发生故障，那么该数据库空间中的所有表或表分段均不可访问，即使其他数据库空间中的表和分段仍可访问。如果发生磁盘故障，限制数据不可用性的需要可能会影响到您在特定的数据库空间中将哪些表分在同一组。
- 冷还原和热还原。虽然在包含关键数据的数据库空间发生故障的情况下必须执行冷还原，但是如果只是非关键的数据库空间发生故障，则只需执行热还原。使冷还原的影响减少到最小的愿望可能会影响用于存储关键性数据的数据库空间。

有关备份与还原的更多信息，请参阅《*SinoDB*® 备份和还原指南》。

检查数据和查询

要确定分段存储策略，您必须收集有关可能分段的表的信息。也必须知道表中数据的使用方式。

要收集有关表的信息：

1. 标识对于性能至关重要的查询，以确定这些查询是联机事务处理 (OLTP) 还是决策支持系统 (DSS) 查询。
2. 使用 SET EXPLAIN 语句可确定数据的访问方式。

   有关 SET EXPLAIN 语句输出的信息，请参阅显示由优化器选择的查询计划的报告 在第216页。要确定数据的访问方式，有时您只需检查 SELECT 语句和表的模式。
3. 确定每个查询检查数据的哪一部分。

   例如，如果在大部分时间中读取表中的某些行，那么您可以在小的分段中将它们进行隔离以减少其他分段的 I/O 争用。
4. 确定哪些语句创建临时文件。

   决策支持查询通常会创建和访问很大的临时文件，而临时数据库空间的位置对于性能而言非常关键。
5. 如果在一个决策支持查询中，某些特定的表总是连接在一起，那么应将这些表的分段分布在不同的磁盘上。
6. 检查表中的列，以确定哪个分段存储方案对于决策支持查询而言，可以使每个扫描线程的繁忙程度相同。

要了解列值如何分布，可使用 UPDATE STATISTICS 语句在列上创建分布，并使用 dbschema 检查分布。

```
dbschema -d database -hd table
```

考虑物理分段存储因素

在对表进行分段时，与表有关的物理位置问题适用于各个表分段。由于每个分段均驻留在磁盘上其自己的数据库空间，因此必须分别针对每个磁盘上的分段解决这些问题。

有关适用于表的放置问题的详细信息，请参阅表性能的注意事项 在第116页。

分段表和未分段表之间存在以下区别：

- 对于分段表，将每个分段置于单独的、指定的数据库空间，或在单个数据库空间内创建的表的多个指定分段中。

  对于未分段表，可以置于当前数据库的缺省数据库空间。

  不管表是否分段，您都应该在每个磁盘上为每个数据库空间创建一个块。
- 分段表的扩展数据块大小通常小于等价的未分段表的扩展数据块大小，因为分段不会增长到和整个表一样大。有关如何估算要分配的空间的更多信息，请参阅估算表大小 在第118页。
- 在分段表中，行指针并不是磁盘上指向该行的唯一保持不变的指针。在索引中，数据库服务器内部使用分段标识符和行指针的组合指向某行。这两个字段是唯一的，但是在该行的存在期间可能会改变。由于应用程序不能访问分段标识符，因此您应该使用主键来访问分段表中的特定行。有关更多信息，请参阅《*SinoDB*® 数据库设计和实现指南》。
- 连接索引或未分段表的索引的行指针使用 4 个字节。拆离索引对每个键值使用 8 字节的磁盘空间以存储分段标识符和行指针的组合。有关如何估算索引空间的更多信息，请参阅估算索引页 在第153页。有关连接索引和拆离索引的更多信息，请参阅将索引分段的策略 在第194页。

决策支持查询通常会创建和访问大的临时文件，而临时数据库空间的位置对于性能而言是一个关键因素。有关临时文件放置的更多信息，请参阅将临时表和排序文件分布在多个磁盘中 在第118页。

分布方案

在决定是否将表行和/或索引键分段以及决定应该如何在各分段间分布行和键之后，您可决定用于实现此分布的方案。SinoDB® 支持分段间的随机分布以及分段间基于值的分布。

分段间的随机分布

循环分段存储

此种类型的分段存储在分段中逐个放入行，循环经过一系列分段以便均匀分布各行。

对于智能大对象，您可以在 CREATE TABLE 或 ALTER TABLE 语句的 PUT 子句中指定多个数据库空间以按照循环分布方案分布智能大对象，从而使每个空间中智能大对象数大致相等。

分段间基于值的分布

基于表达式的分段存储

此种类型的分段存储将包含指定值的行放在同一个分段中。您可以指定分段存储表达式以定义将一组行分配到每个分段的标准，该标准可以作为范围规则，也可以作为某个仲裁规则。

您可以指定剩余分段，用于存放所有不匹配任何其他分段标准的行，不过剩余分段会降低基于表达式的分布方案的效率。

基于列表的分段存储

此类型的分段存储将包含指定值（与离散值列表中某个指定值相匹配）的行放在同一分段中。对于每个分段，将一个或多个常量表达式的列表指定为对应于表中一个或多个列的分段表达式。计算分段表达式的列或列集称为分段键。

您可以选择指定剩余分段，用于保留不与任何其他分段条件相匹配的所有行。您也可以选择指定 NULL 分段，用于将缺少数据的行存储在分段键列中（因为其分段表达式是 NULL 或 IS NULL）。

按列表分段存储和按表达式分段存储之间最重要的区别是每个分段的列表中每个值在相同表或索引的所有分段列表之间必须唯一。

基于时间间隔的分段存储

此类型的分段存储将数据分入分段中，这些分段基于特定时间间隔内相同分段中单个数字、DATE 或 DATETIME 列的分段键范围内的量化值。至少指定一个范围表达式作为定义每个分段的分段键值上限的分段表达式，并指定数据库服务器自动创建的系统定义分段范围大小的时间间隔表达式。

您可以选择定义一个 NULL 段来存储段键列有缺失数据的行，但不支持或不需要任何剩余分段。数据库服务器会自动创建新分段来存储具有任何现有分段范围之外的非 NULL 分段键值的行。使用范围表达式定义的分段称为范围分段，数据库服务器在运行时创建的系统定义分段称为时间间隔分段。此类型的分布方案有时称为范围间隔分布策略。

相关链接

在 *DBSPACETEMP* 配置参数中指定临时表 在第92页

《*SinoDB SQL* 指南: 语法》: *List fragment* 子句

《*SinoDB SQL* 指南: 语法》: *Interval fragment* 子句

选择分布方案

选择分布方案时，必须考虑数据平衡的难易程度，是否希望消除分段，以及数据跳过功能的效果。

表 *12:* 分布方案比较 在第192页比较循环分布方案和基于表达式的分布方案。

表 12: 分布方案比较

分布方案	数据平衡的难易程度	分段消除	数据跳过
循环方式	自动。随时间平衡数据。	数据库服务器无法消除分段。	在使用数据跳过特性时，无法确定事务的完整性是否受到损害。但是，可以插入到使用循环来分段的表。
基于表达式	需要具有数据分布方面的知识。	如果表达式中使用了一列或两列，那么数据库服务器可以为那些具有范围表达式或等式表达式的查询消除分段。	在使用数据跳过特性时，您可以确定事务的完整性是否受到损害。如果这些行的相应分段不可用，那么无法插入行。

选择哪种分布方案，取决于以下因素：

- 您想利用表 *12:* 分布方案比较 在第192页中的哪种特性
- 查询是否要对整个表进行扫描
- 您是否知道要添加的数据的分布
- 应用程序是否要删除很多行
- 是否在表中循环数据

基本上，循环方案提供了最容易而且最可靠的方法来平衡数据。然而，如果使用循环分布，您就无法知道某个行位于哪个分段，而且数据库服务器也无法消除分段。

通常，只有满足以下条件时，循环才是正确的选择：

- 查询要扫描整个表。
- 您不知道要添加的数据的分布。
- 应用程序不需要删除很多行。（如果需要，那么负载均衡可能会受到负面影响。）

如果满足下列任何一个条件，那么基于表达式的方案会是将数据进行分段的最佳选择方案：

- 应用程序调用大量决策支持查询，这些查询将扫描表的特定部分。

- 您知道数据分布是什么。
- 计划在数据库中循环数据。

如果计划基于某个列（例如日期）的值定期添加或删除大量数据，您可以在分布方案中使用该列。然后可以使用 alter fragment attach 和 alter fragment detach 语句在表中循环数据。

ALTER FRAGMENT ATTACH 和 DETACH 语句相对块载入和块删除具有以下优点：

- 其他用户仍可以访问其余的表分段。他们不能访问的只是您附加或者拆离的分段。
- 由于性能的提高，执行 ALTER FRAGMENT ATTACH 或 DETACH 语句比块载入或者大量删除快得多。

有关更多信息，请参阅提高连接和拆离分段的操作性能 在第201页。

在某些情况下，合适的索引方案可以巧妙解决特定分布方案的性能问题。有关更多信息，请参阅将索引分段的策略 在第194页。

设计基于表达式的分布方案

设计基于表达式的分布方案的第一个步骤是确定表中数据的分布情况，尤其是使用分段存储表达式的列中的值分布情况。

要获得此信息，可对表执行 UPDATE STATISTICS 语句，然后使用 dbschema 实用程序来检查分布情况。

知道了数据分布之后，您就可以设计一个分段存储规则以根据分段存储目标的要求在分段之间分布数据。如果主要目标是提高性能，那么分段表达式应该在分段之间对行进行均匀分布。

如果分段存储的主要目标是提高并发性，那么应该分析访问该表的查询。如果对某些行访问的比例高于其他行，您可以通过选择对创建的分段使用不均匀数据分布来进行弥补。

尽量不要在分布表达式中使用那些经常更新的列。此类更新可能导致行从一个分段移至其他分段（也就是说，从一个分段中删除，又添加到另一个分段），并且该活动增加了 CPU 和 I/O 开销。

要实现最佳分段消除特征，应尝试不使用 REMAINDER 分段来创建基于单个列的非重叠区域。只要查询优化器可以根据基于表达式的分段存储规则（通过该规则可将行分配到分段）确定 WHERE 子句选择的值没有驻留在这些分段上，那么数据库服务器将从查询计划中消除这些分段。有关更多信息，请参阅消除分段的分布方案 在第197页。

改进分段存储的建议

可以改进分段存储以实现在决策支持和 OLTP 查询方面的优化性能。

以下建议是对表和索引进行分段的准则：

- 要实现决策支持查询方面的最优性能，应对表进行分段，以提高并行性，但不要对索引进行分段。拆分索引，并将它们放入单独的数据库空间。
- 要实现 OLTP 查询方面的最佳性能，请使用分段索引以减少会话之间的争用。通常您可以按键值对索引进行分段，这意味着 OLTP 查询只需查看一个分段就可以找到行的位置。

  如果键值无法减少争用，如每个用户查看同一组键值（例如，日期范围）时，那么您可以考虑 WHERE 子句中使用的其他值对索引进行分段。要减少分段管理，可以考虑不对某些索引进行分段，尤其是您无法找到一个好的分段存储表达式来减少争用时。
- 如果决策支持查询顺序读取表，那么可以对数据使用循环分段存储。如果表中的所有列都不能使用基于表达式的分段存储方案，那么对于在多个磁盘间平均分布数据而言，循环分段存储不失为一个好方法。但是，在大多数 DSS 查询中将读取所有的分段。
- 要减少所需数据库空间的总数和搜索所需的时间，可在相同的数据库空间中存储多个指定分段。
- 如果您正在使用表达式，应创建它们，这样在磁盘之间平衡的是 I/O 查询而不是数据的数量。例如，如果大多数查询只访问表中的一部分数据，那么应设置分段存储表达式在磁盘间散布表的活动部分，即使这样会导致行分布的不是很均匀。
- 使分段存储表达式保持简单。您想要分段存储表达式有多复杂，它们就可以有多复杂。但是，计算复杂的表达式会花费更多的时间，并有可能导致无法从查询中消除分段。
- 合理排列分段存储表达式，以便可以先在表达式中测试每个数据库空间的最具限制性的条件。当数据库服务器为一个给定分段按照标准测试一个值时，只要该分段的任何一个条件的结果为假，测试就会停止。因此，如果将最有可能为假的条件放在最前面，那么在数据库服务器转到下一个分段前，需要求值

的条件就会减少。例如，在下面的表达式中，当尝试插入一个值为 25 的行时，数据库服务器会测试所有 6 个不等式条件：

```
x >= 1 and x <= 10 in dbspace1,
x > 10 and x <= 20 in dbspace2,
x > 20 and x <= 30 in dbspace3
```

通过比较，以下表达式中只有 4 个条件需要测试：dbspace1 的第一个不等式条件 (x <= 10)、dbspace2 的第一个条件 (x <= 20)，以及 dbspace3 的两个条件：

```
x <= 10 and x >= 1 in dbspace1,
x <= 20 and x > 10 in dbspace2,
x <= 30 and x > 20 in dbspace3
```

- 应避免使用任何需要数据类型转换的表达式。类型转换会增加计算表达式所花费的时间。例如，为便于比较，DATE 数据类型将隐式转换为 INTEGER。
- 除非您不介意增加管理成本，否则请不要对经常改变的列进行分段。例如，如果对日期列进行分段，并且删除较旧的行，那么具有最早日期的分段就可能清空，而具有最近日期的分段有可能填满。最终，您必须删除旧的分段，并为更新的订单添加新的分段。
- 不要对每个表均进行分段。应标识出访问最频繁的那些关键表。在一个磁盘上，只放置一个表的一个分段。
- 不要对小的表进行分段。在许多磁盘上将一个小表进行分段，可能根本不值启动所有扫描线程来访问这些分段的开销。同样，应该根据系统中处理器的数量来平衡分段的数量。
- 当您在未分段表上定义一个分段存储策略时，应检查下一扩展数据块大小，以确保未给每个分段分配大量的磁盘空间。

将索引分段的策略

对表进行分段时，根据使用的分段存储方案（除了启用自动定位时的循环分布方案以外），对与该表相关联的索引进行隐式分段。当 AUTOLOCATE 配置参数或环境选项设为正整数时，使用循环分布方案的表上的索引不会进行分段。您可以使用 CREATE INDEX 语句的 FRAGMENT BY 子句对任何表的索引进行分段。

分段表的每个索引根据自己的扩展数据块占用其本身的表空间。

可以用以下两种策略之一对索引进行分段：

- 与表使用相同的分段存储策略
- 与表使用不同的分段存储策略

连接索引

连接索引是隐式遵循表分段存储策略（分布方案和分段所在的数据库空间集）的索引。当您在分段表上创建一个索引时，除非使用循环分布方案并启用自动定位，否则该索引为连接的索引。当 AUTOLOCATE 配置参数或环境选项设为正整数时，使用循环分布方案的表上的索引不会进行分段。

要创建连接索引，请不要在 CREATE INDEX 语句中指定分段存储策略或存储选项，如以下 SQL 语句的示例所示：

```
CREATE TABLE tb1(a int)
   FRAGMENT BY EXPRESSION
      (a >=0 AND a < 5) IN dbsbspace1,
      (a >=5 AND a < 10) IN dbspace2
      ...
   ;

CREATE INDEX idx1 ON tb1(a);
```

对于使用基于表达式的分布方案或循环分布方案的分段表，也可以在单个数据库空间中创建表或索引的多个分区。这使您能够减少所需的数据库空间数，从而简化数据库空间的管理。

要创建具有分区的连接索引，请在 SQL 语句中包括分区名称，如此示例中所示：

```
CREATE TABLE tb1(a int)
   FRAGMENT BY EXPRESSION
     PARTITION part1 (a >=0 AND a < 5) IN dbs1,
     PARTITION part2 (a >=5 AND a < 10) IN dbs1
          ...
 ;

    CREATE INDEX idx1 ON tb1(a);
```

可在 CREATE TABLE、CREATE INDEX 和 ALTER FRAGMENT ON INDEX 语句中使用“PARTITION BY EXPRESSION”而不是“FRAGMENT BY EXPRESSION”，如以下示例所示：

```
ALTER FRAGMENT ON INDEX idx1 INIT PARTITION BY EXPRESSION
           PARTITION part1   (a <= 10) IN dbs1,
           PARTITION part2   (a <= 20) IN dbs1,
            PARTITION part3  (a <= 30) IN dbs1;
```

使用 ALTER FRAGMENT 语法将不具有分区的分段索引更改为具有分区的索引。下面的语法显示了您可以如何将分段索引转换为包含分区的索引：

```
CREATE TABLE t1 (c1 int) FRAGMENT BY EXPRESSION
     (c1=10) IN dbs1, (c1=20) IN dbs2, (c1=30) IN dbs3
CREATE INDEX ind1 ON t1 (c1) FRAGMENT BY EXPRESSION
     (c1=10) IN dbs1, (c1=20) IN dbs2, (c1=30) IN dbs3
```

```
ALTER FRAGMENT ON INDEX ind1 INIT FRAGMENT BY EXPRESSION
     PARTITION part_1 (c1=10) IN dbs1, PARTITION part_2 (c1=20) IN dbs1,
     PARTITION part_3 (c1=30) IN dbs1,
```

创建包含分区的表或索引将提高性能，这是通过使数据库服务器能够更快进行搜索和减少所需数据库空间数来实现的。

通过对索引键使用与对表数据所使用的相同的规则，数据库服务器对连接索引进行分段，所采用的分布方案与对表采用的相同。因此，连接索引具有以下物理特征：

- 索引分段数与数据分段数相同。
- 每个连接索引分段与相应的表数据驻留在相同的数据库空间中，但在不同的表空间中。
- 连接索引或未分段表的索引使用 4 个字节作为每个索引条目的行指针。有关如何估算索引空间的更多信息，请参阅估算索引页 在第153页。

SinoDB® 不支持森林树连接索引。

拆离索引

拆离索引是具有使用 CREATE INDEX 语句显式设置的单独分段存储策略的索引。

以下 SQL 语句示例创建了拆离索引：

```
CREATE TABLE tb1 (a int)
      FRAGMENT BY EXPRESSION
         (a <= 10) IN tabdbspc1,
         (a <= 20) IN tabdbspc2,
         (a <= 30) IN tabdbspc3;

CREATE INDEX idx1 ON tb1 (a)
      FRAGMENT BY EXPRESSION
         (a <= 10) IN idxdbspc1,
         (a <= 20) IN idxdbspc2,
         (a <= 30) IN idxdbspc3;
```

此示例描述了常见的分段存储策略，它对索引分段的方法与对表分段的方法相同，但是它为索引分段指定了不同的数据库空间。这种分段存储策略将索引分段放在与表不同的数据库空间中，可提高一些操作的性能，如备份、恢复等等。

除非指定了不建议使用的 IN TABLE 语法，否则，在缺省情况下，CREATE INDEX 语句创建的所有新索引都在与数据不同的表空间中分离和存储。

要创建具有分区的拆离索引，请在 SQL 语句中包括分区名称，如此示例中所示：

```
CREATE TABLE tb1 (a int)
      FRAGMENT BY EXPRESSION
            PARTITION part1 (a <= 10) IN dbs1,
            PARTITION part2 (a <= 20) IN dbs2,
            PARTITION part3 (a <= 30) IN dbs3;

    CREATE INDEX idx1 ON tb1 (a)
          FRAGMENT BY EXPRESSION
            PARTITION part1   (a <= 10) IN dbs1,
            PARTITION part2   (a <= 20) IN dbs2,
            PARTITION part3   (a <= 30) IN dbs3;
```

可在 CREATE TABLE、CREATE INDEX 和 ALTER FRAGMENT ON INDEX 语句中使用 PARTITION BY EXPRESSION 关键字而不是 FRAGMENT BY EXPRESSION 关键字。

如果不想对索引进行分段，可以将整个索引放入一个单独的数据库空间。

您可以使用表达式对任何表进行索引分段。但是，您不能为索引显式创建循环分段存储方案。每当使用循环分段存储方案对表进行分段时，都会将伴随表的所有索引转换为拆离索引，以实现最佳性能。

拆离索引具有以下物理特征：

- 每个拆离索引分段均驻留在与相应的表数据不同的表空间中。因此，数据和索引页不能在表空间中交错。
- 拆离索引分段具有自己的扩展数据块和表空间标识符。表空间标识符也称为分段标识符和分区号。一个拆离索引对每个索引条目使用 8 个字节的磁盘空间以存储分段标识符和行指针的组合。有关如何估算索引空间大小的更多信息，请参阅估算索引页 在第153页。

森林树索引是拆离索引。它们不能是连接索引。

在系统目录表 sysfragments 中，数据库服务器存储每个表和索引分段的位置，以及其他相关信息。可以查看 sysfragments 系统目录表来访问以下有关分段表和索引的信息：

- partn 列中的值是表或者索引分段的分区号或分段标识符。拆离索引的分区号与相应表分段的分区号不同。
- strategy 列中的值是用于分段存储策略的分布方案。

有关此 sysfragments 系统目录表包含的列值的完整描述，请参阅《*SinoDB® SQL* 指南: 参考》。有关如何使用 sysfragments 来监视分段的信息，请参阅监视分段使用情况 在第208页。

分段表索引的限制

在数据库服务器扫描一个分段索引时，必须扫描多个索引分段，并把结果合并在一起。（有一个例外，如果根据某些索引键范围规则对索引进行分段，扫描就不会跨过分段边界。）由于这种需要，如果对索引进行了分段，那么索引扫描的性能就可能受到影响。

出于这些性能方面的考虑，数据库服务器对索引有以下限制：

- 不能使用循环对索引进行分段。
- 不能用包含不在索引键中的列的表达式来对唯一索引进行分段。

例如，以下语句是无效的：

```
CREATE UNIQUE INDEX ia on tab1(col1)
   FRAGMENT BY EXPRESSION
```

```
      col2<10 in dbsp1,
      col2>=10 AND col2<100 in dbsp2,
      col2>100 in dbsp3;
```

将临时表分段的策略

您可以对位于多个数据库空间（它们驻留在不同的磁盘上）的显式临时表进行分段。

您可以使用 CREATE TABLE 语句的 TEMP TABLE 子句来创建临时的分段表。然而，您无法变更分段临时表的分段存储策略（而可以在永久表中变更分段存储策略）。数据库服务器在删除临时表的同时，还删除为该临时表创建的分段。

您可以为显式临时表定义您自己的分段存储策略，或者由数据库服务器动态确定分段存储策略。

有关显式和隐式临时表的更多信息，请参阅《*SinoDB*® 管理员指南》。

消除分段的分布方案

分段消除是数据库服务器的一项功能，可以减少数据库操作中涉及的分段数。此功能能够显著提高性能，同时减少对分段驻留的磁盘的争用。

分段消除缩短了给定查询的响应时间，同时也提高了查询之间的并行性。由于数据库服务器无需读取不必要的分段，因而减少了查询的 I/O。同时也减少了 LRU 队列中的活动。

如果使用了合适的分布方案，那么数据库服务器可以从以下数据库操作中消除分段：

- SQL 中 SELECT、INSERT、delete 或 update 语句的访存部分。

  在实际搜索之前，如果对这些 SQL 语句进行了优化，那么数据库服务器可以消除分段。
- 嵌套循环连接

  当数据库服务器从外表中得到键值时，就可以消除分段，从而在内表中搜索。

数据库服务器是否可以从一个搜索中消除分段，取决于以下两个因素：

- 正在搜索的表的分段存储策略中的分布方案
- 查询表达式的形式（SELECT、INSERT、delete 或 update 语句的 WHERE 子句中的表达式）

分段消除的分段存储表达式

表达式中的某些运算符导致自动消除分段。

当使用以下任何运算符来定义分段存储策略时，对于表的查询而言，可发生分段消除。

```
IN
=
<
>
<=
>=
AND
OR
NOT
IS NULL（仅适用于不使用 AND 或 OR 运算符与其他表达式相组合的情况）
```

如果分段存储表达式使用以下任何运算符，那么对于表的查询不会发生分段消除。

```
!=
IS NOT NULL
```

有关允许分段消除的分段存储表达式的示例，请参阅分段消除的有效性 在第199页。

分段消除的查询表达式

查询表达式（WHERE 子句中的表达式）可以包含简单表达式、非简单表达式和多个表达式。

对于分段消除，数据库服务器只考虑使用简单表达式或由某些运算符组合在一起的多重简单表达式。

简单表达式包含以下部分：

```
column operator value
```

column

是单个列名

除了用 NCHAR、NVARCHAR、BYTE 和 TEXT 数据类型定义的列以外，数据库服务器支持对其他所有列类型的分段消除。

operator

必须是等式运算符或范围运算符

value

必须是文本变量或主变量

以下显示的是简单表达式示例：

```
name = "Fred"
date < "08/25/2008"
value >= :my_val
```

以下是非简单表达式示例：

```
unitcost * count > 4500
price <= avg(price)
result + 3 > :limit
```

基于运算符，数据库服务器对分段消除考虑使用两种类型的简单表达式：

- 范围表达式
- 等式表达式

查询中的范围表达式
通过在 WHERE 子句中使用五个关系运算符的任意组合，数据库服务器可以对查询处理一个或两个列的分段消除。

范围表达式使用以下关系运算符：

```
<
>
<=
>=
!=
```

如果这些范围表达式与以下运算符组合在一起，那么数据库服务器也可以消除分段：

```
AND, OR, NOT
IS NULL, IS NOT NULL
MATCH, LIKE
```

如果范围表达式包含 MATCH 或 LIKE，那么在字符串以通配符结束的情况下，数据库服务器也可以消除分段。以下示例显示能够利用分段消除的查询表达式：

```
columna MATCH "ab*"
```

```
columna LIKE "ab%" OR columnb LIKE "ab*"
```

查询中的等式表达式
通过在 WHERE 子句中使用等式运算符的组合，数据库服务器可以对查询处理一个或多个列分段消除。

等式表达式使用以下等式运算符：

```
=, IN
```

如果这些等式表达式与以下运算符结合使用，那么数据库服务器也可以消除分段：

```
AND, OR
```

分段消除的有效性

如果使用循环分布方案对表进行分段，那么数据库服务器不能消除分段。而且，不是所有基于表达式的分布方案都提供了相同的分段消除行为。

下表总结了基于表达式的分布方案和查询表达式的不同组合的分段消除行为。分段表中的分区不会影响下表中显示的分段消除行为。

表 13: 基于表达式的分布方案和查询表达式的不同类型的分段消除

查询（WHERE 子句）表达式的类型	单个列上的非重叠分段	单个列上的重叠或非邻接分段	多个列上的非重叠分段
范围表达式	分段可以消除。	分段不能消除。	分段不能消除。
等式表达式	分段可以消除。	分段可以消除。	分段可以消除。

此表说明了分布方案可启用分段消除，但是分段消除的有效性取决于指定查询的 WHERE 子句。

例如，可以考虑用以下表达式进行分段的表：

```
FRAGMENT BY EXPRESSION
100 < column_a AND column_b < 0 IN dbsp1,
100 >= column_a AND column_b < 0 IN dbsp2,
column_b >= 0 IN dbsp3
```

如果 WHERE 子句具有以下表达式，那么数据库服务器不能从搜索中消除任何分段：

```
column_a = 5 OR column_b = -50
```

但是，如果 WHERE 子句具有以下表达式，那么数据库服务器可以消除数据库空间 dbsp3 中的分段：

```
column_b = -50
```

而且，如果 WHERE 子句具有以下表达式，那么数据库服务器还可以消除数据库空间 dbsp2 和 dbsp3 中的两个分段：

```
column_a = 5 AND column_b = -50
```

分段表中的分区不会影响分段消除行为。

单个列上的非重叠分段
从分段消除的角度出发，对单个列创建非重叠分段的分段存储规则是首选的分段存储规则。

这种分布方案的优点在于，数据库服务器可以消除那些具有范围表达式和等式表达式查询的分段。您应该在设计分段存储规则时满足这些条件。图 *39:* 单个列上的非重叠分段的示例 在第200页给出了这种类型的分段存储规则的示例。

```
...
FRAGMENT BY EXPRESSION
a<=8 OR a IN (9,10) IN dbsp1,
10<a AND a<=20 IN dbsp2,
a IN (21,22,23) IN dbsp3,
a>23 IN dbsp4;
```

图 39: 单个列上的非重叠分段的示例

您可以使用范围规则或基于单个列的仲裁规则来创建非重叠分段。您可以使用关系运算符，以及 AND、IN、OR 或 BETWEEN。使用 BETWEEN 运算符时一定要小心。数据库服务器对 BETWEEN 关键字进行语法分析时，它包含了在值范围中指定的结束点。在您的表达式中应避免使用 REMAINDER 子句。如果使用 REMAINDER 子句，那么数据库服务器始终不能消除剩余分段。

单个列上的重叠分段
单个列上的分段可以是重叠和非邻接的。您可以使用任何范围、MOD 函数或基于单个列的仲裁规则。

此类别的分段存储规则的唯一限制是您应该在单个列的基础上创建分段存储规则。

图 *40:* 单个列上的重叠分段的示例 在第200页显示这种类型的分段存储规则的示例。

```
...
FRAGMENT BY EXPRESSION
a<=8 OR a IN (9,10,21,22,23) IN dbsp1,
a>10 IN dbsp2;
```

图 40: 单个列上的重叠分段的示例

如果您使用这种类型的分布方案，那么数据库服务器可以对等式搜索消除分段，而不是对范围搜索。由于所有 INSERT 和许多 UPDATE 操作可以执行等式搜索，因而此分布方案仍然有用。

如果使用的表达式不能创建具有邻接值的非重叠的分段，这种备选分布方案可以接受。例如，如果表随时间不断增长，那么您可能想要使用 MOD 函数规则将分段保持为相似的大小。由于每个分段中值均不相邻，因而使用 MOD 函数规则的基于表达式的分布方案属于这种类别。

非重叠分段，多列
数据库服务器使用仲裁规则来定义基于多个列的非重叠分段。

下图显示两个列的非重叠分段的示例。

```
...
FRAGMENT BY EXPRESSION
0<a AND a<=10 AND b IN ( 'E',  'F',  'G') IN dbsp1,
0<a AND a<=10 AND b IN ( 'H',  'I',  'J') IN dbsp2,
10<a AND a<=20 AND b IN ( 'E',  'F',  'G') IN dbsp3,
10<a AND a<=20 AND b IN ( 'H',  'I',  'J') IN dbsp4,
20<a AND a<=30 AND b IN ( 'E',  'F',  'G') IN dbsp5,
20<a AND a<=30 AND b IN ( 'H',  'I',  'J') IN dbsp6;
```

图 41: 两个列上的非重叠分段的示例

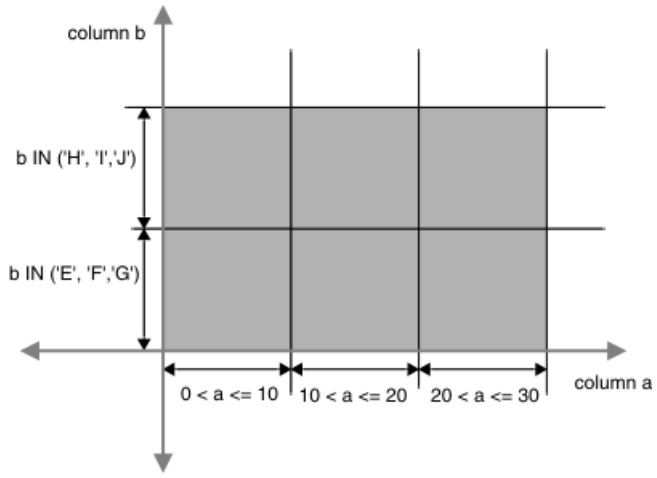

图 42: 两个列上的非重叠分段的图解示例

如果您使用这种类型的分布方案，那么数据库服务器可以对等式搜索消除分段，而不是对范围搜索。由于所有 INSERT 操作和许多 UPDATE 操作可以执行等式搜索，因而此分布方案仍然有用。在表达式中应避免使用 REMAINDER 子句。如果使用 REMAINDER 子句，那么数据库服务器始终不能消除剩余分段。

如果使用基于单个列的表达式无法获得足够的粒度，那么可以接受此备选方案。

提高连接和拆离分段的操作性能

当您使用 ALTER FRAGMENT ATTACH 和 DETACH 语句在非常大的表中添加或移除大量数据时，可以执行多个步骤来提高 ATTACH 和 DETACH 操作的性能。

ALTER FRAGMENT DETACH 语句提供了快速删除表数据段的方法。同样，ALTER FRAGMENT ATTACH 语句提供了一种通过利用分段存储技术向现有表递增载入大量数据的方法。但是，数据库服务器对继续存在的表重新建立索引时执行 ALTER FRAGMENT ATTACH 和 DETACH 语句将花费很长时间。

数据库服务器为 ALTER FRAGMENT 语句的 ATTACH 和 DETACH 操作提供了性能优化，这使数据库服务器能够重新使用继续存在的表的索引。因此，数据库服务器可以消除 ATTACH 或 DETACH 操作期间的索引建立，从而：

- 缩短执行 ALTER FRAGMENT ATTACH 和 ALTER FRAGMENT DETACH 语句所花费的时间
- 提高表的可用性

ALTER FRAGMENT 操作需要对操作中涉及的所有表进行互斥存取和互斥锁定。当您使用 FORCE_DDL_EXEC 环境选项为数据库服务器指定时间限制以强行排除在 ALTER FRAGMENT ON TABLE 操作涉及的表中打开的或锁定的其他会话中的所有事务时，也可以使用 SET LOCK MODE TO WAIT 语句将该秒数指定为等待限制。

如果由于并行会话中的 DDL 事务，数据库服务器无法对表进行互斥存取和互斥锁定，那么服务器将开始回滚表中已打开或锁定的事务，直至达到指定的时间限制。您可能希望对忙碌的系统启用 FORCE_DDL_EXEC 选项并发出 SET LOCK MODE TO WAIT 语句，可能是一天运行 24 小时，且您不希望在变更分段前等待并行会话中的事务关闭。

提高 ALTER FRAGMENT ATTACH 的性能

如果数据库满足特定需求，那么您可以利用 ALTER FRAGMENT ATTACH 语句的性能优化。

要利用性能优化，您必须满足以下所有需求：

- 为表和索引分段制定合适的分布方案。
- 确保因分段表达式而产生的分区之间没有发生数据移动。
- 更新所有参与表的统计信息。
- 如果对继续存在的表的索引是唯一的，那么应使附加表的索引也唯一。

重要: 只有使用日志记录的数据库能够从 ALTER FRAGMENT ATTACH 语句的性能提高中获益。如果没有日志记录，数据库服务器会使用同一个表的多个副本，以确保发生故障时数据的可恢复性。此要求无法重新使用现有索引分段。

用于复用索引的分布方案
可以使用三种分布方案的其中一种使 ALTER FRAGMENT 语句的连接操作能够复用现有索引。

这些分布方案为：

- 以对表所用的相同方法对索引进行分段
- 以对表所用的相同分段表达式集对索引进行分段
- 连接未分段表以形成分段表

以对表所用的相同方法对索引进行分段
如果在创建索引时未指定分段存储策略，那么使用与表相同的方法对索引进行分段。

分段存储策略是指分布方案和放置分段的数据库空间集。有关详细信息，请参阅规划分段存储策略 在第188页。

以对表所用的相同方法对索引进行分段的示例

假设您用以下 SQL 语句创建分段表和索引：

```
CREATE TABLE tb1(a int)
   FRAGMENT BY EXPRESSION
      (a >=0 AND a < 5) IN db1,
      (a >=5 AND a <10) IN db2;

CREATE INDEX idx1 ON tb1(a);
```

假设您接着创建一个未分段表，然后决定将它附加到前面的分段表。

```
CREATE TABLE tb2 (a int, CHECK (a >=10 AND a<15))
   IN db3;

CREATE INDEX idx2 ON tb2(a)
   IN db3;

ALTER FRAGMENT ON TABLE tb1
      ATTACH
         tb2 AS (a >= 10 and a<15) AFTER db2;
```

如果在现有的和新的表分段之间未发生数据移动，那么附加操作可以利用现有索引 idx2。如果不发生数据移动：

- 数据库服务器重新使用索引 idx2，并将其转换为索引 idx1 的分段。
- 索引 idx1 仍保持与表 tb1 使用相同的分段存储策略。

如果数据库发现表 tb2 中的一行或多行属于表 tb1 先前存在的分段，那么数据库服务器将执行以下操作：

- 删除并重建索引 idx1，使其包括那些原先在表 tb1 和 tb2 中的行
- 删除索引 idx2

有关如何确保现有表分段和新表分段之间不发生数据移动的更多信息，请参阅确保在连接分段时不发生数据移动 在第204页。

使用与表相同的分布方案对索引进行分段
在创建的索引使用与对表所用的相同分段表达式时，使用与表相同的分布方案来对索引进行分段。

数据库服务器基于表达式树的等价性而不是基于代数等价性来确定分段表达式是否相同。例如，考虑以下两个表达式：

```
(col1 >= 5)
```

```
(col1 = 5 OR col1 > 5)
```

虽然这两个表达式在代数上是等价的，但它们不是相同的表达式。

使用与表相同的分布方案对索引进行分段的示例

假设您用以下 SQL 语句创建两个分段表和索引：

```
CREATE TABLE tb1 (a INT)
   FRAGMENT BY EXPRESSION
      (a <= 10) IN tabdbspc1,
      (a <= 20) IN tabdbspc2,
      (a <= 30) IN tabdbspc3;
CREATE INDEX idx1 ON tb1 (a)
   FRAGMENT BY EXPRESSION
      (a <= 10) IN idxdbspc1,
      (a <= 20) IN idxdbspc2,
      (a <= 30) IN idxdbspc3;

CREATE TABLE tb2 (a INT CHECK a> 30 AND a<= 40)
   IN tabdbspc4;
CREATE INDEX idx2 ON tb2(a)
   IN idxdbspc4;
```

假设您接着用以下 SQL 语句示例将表 tb2 附加到表 tb1：

```
ALTER FRAGMENT ON TABLE tb1
   ATTACH tb2 AS (a <= 40);
```

由于以下原因，数据库服务器可以为该连接操作消除索引 idx1 的重建：

- 索引 idx1 的分段存储表达式与表 tb1 的分段存储表达式完全相同。数据库服务器会执行以下操作：
  - 将索引 idx1 的分段存储扩展到数据库空间 idxdbspc4
  - 将索引 idx2 转换为索引 idx1 的分段
- 由于 CHECK 约束与产生的附加表的分段存储表达式相同，因而没有任何行将从一个分段迁移到其他分段。

  有关如何确保现有表分段和新表分段之间不发生数据移动的更多信息，请参阅确保在连接分段时不发生数据移动 在第204页。

将未分段表连接在一起

将两个未分段的表组合成一个分段的表时，可以利用 ALTER FRAGMENT ATTACH 操作的性能优势。

例如，假设您用以下 SQL 语句创建两个未分段表和索引：

```
CREATE TABLE tb1(a int) IN db1;
   CREATE INDEX idx1 ON tb1(a) in db1;
CREATE TABLE tb2(a int) IN db2;
   CREATE INDEX idx2 ON tb2(a) in db2;
```

您可能想要用以下分布方案的示例将这两个未分段表组合起来：

```
ALTER FRAGMENT ON TABLE tb1
   ATTACH
      tb1 AS (a <= 100),
      tb2 AS (a > 100);
```

如果在 tb1 和 tb2 的分段之间不发生数据迁移，那么数据库服务器会用以下分段存储策略重新定义索引 idx1：

```
CREATE INDEX idx1 ON tb1(a) F
```

```
FRAGMENT BY EXPRESSION
   a <= 100 IN db1,
   a > 100 IN db2;
```

重要: 此行为会导致早于 V7.3 和 V9.2 的数据库服务器使用不同的分段存储策略。在更早的版本中，ALTER FRAGMENT ATTACH 语句在数据库空间 db1 中创建未分段的拆离索引。

确保在连接分段时不发生数据移动

可以通过建立一致检查约束表达式并验证分段表达式是否重叠来确保在连接分段时不会移动数据。

要确保在连接分段时不发生数据移动，请执行以下操作：

1. 对附加表建立检查约束，假设 ALTER FRAGMENT ATTACH 操作之后，检查约束与分段表达式相同。
2. 用非重叠表达式定义分段。

例如，您可能会用以下 SQL 语句创建分段表和索引：

```
CREATE TABLE tb1(a int)
   FRAGMENT BY EXPRESSION
      (a >=0 AND a < 5) IN db1,
      (a >=5 AND a <10) IN db2;

CREATE INDEX idx1 ON tb1(a);
```

假设您又创建另一个未分段的表，然后决定将它附加到前面的分段表。

```
CREATE TABLE tb2 (a int, check (a >=10 and a<15))
   IN db3;

CREATE INDEX idx2 ON tb2(a)
   IN db3;

ALTER FRAGMENT ON TABLE tb1
   ATTACH
      tb2 AS (a >= 10 AND a<15) AFTER db2;
```

此 ALTER FRAGMENT ATTACH 操作利用现有索引 idx2，因为在示例中执行了以下步骤以阻止现有的表分段和新的表分段之间的数据移动：

- CREATE TABLE tb2 语句中的检查约束表达式与 ALTER FRAGMENT ATTACH 语句中的表 tb2 的分段表达式相同。
- CREATE TABLE tb1 中指定的分段表达式与 ALTER FRAGMENT ATTACH 语句不重叠。

因此，数据库服务器在数据库空间 db3 中保留索引 idx2，并将其转换成索引 idx1 的分段。索引 idx1 仍保持与表 tb1 使用相同的分段存储策略。

连接表的索引

数据库服务器试图重新使用附加表的索引作为所生成索引的分段。但是，附加表的相应索引可能不存在，或者可能由于磁盘格式不匹配而不能使用。在这些情况下，对附加表建立索引可能比对合成表建立整个索引要快。

SinoDB® 将估算对生成的表创建整个索引的成本。然后，服务器将此成本与为附加表构建单独的索引分段所用成本相比较，选择成本较低的索引构建。

自动收集的新索引统计信息

成功运行 CREATE INDEX 语句时，无论 ONLINE 关键字存在与否，SinoDB® 都将自动收集新创建索引的以下统计信息：

- 索引级别的统计信息，等同于在 LOW 方式的 UPDATE STATISTICS 操作中收集到所有类型的索引的统计信息，包括 B 型树、虚拟索引接口和函数索引。
- 对于一般 B 型树索引的透明主索引列，列分布统计信息等同于在 HIGH 方式的 UPDATE STATISTICS 操作中生成的分布。对于少于一百万行的表，HIGH 方式的分辨率是 1.0，对于更大的表，该值为 0.5。由于具有更多数据容器的统计信息，因而多于一百万行的表具有更好的分辨率。

查询优化器为创建新索引的表设计查询计划时，自动收集的分布统计信息可用于该优化器。

附加表之前运行 UPDATE STATISTICS

要确保成本估算的正确性，您应在附加表之前对所有参与的表执行 UPDATE STATISTICS 语句。UPDATE STATISTICS 语句的 LOW 模式足以使优化器获取适当的统计信息来确定重建索引的成本估算值。

有关 UPDATE STATISTICS 语句的更多信息，请参阅《*SinoDB® SQL* 指南*:* 语法》。

相应索引不存在时的情境示例
表没有可充当所生成索引的分段的列索引时，数据库服务器估算构建列的索引分段的成本，将此成本与重新构建生成表的所有分段的整个索引相比，选择成本较少的索引构建。

假设您用以下 SQL 语句创建分段表和索引：

```
CREATE TABLE tb1(a int, b int)
   FRAGMENT BY EXPRESSION
      (a >=0 AND a < 5) IN db1,
      (a >=5 AND a <10) IN db2;
CREATE INDEX idx1 ON tb1(a);
```

假设您接着又创建两个未分段的表，然后决定将它们附加到前面的分段表。

```
CREATE TABLE tb2 (a int, b int,
   CHECK (a >=10 and a<15)) IN db3;
CREATE INDEX idx2 ON tb2(a) IN db3;
CREATE TABLE tb3 (a int, b int,
   CHECK (a >= 15 and a<20)) IN db4;
CREATE INDEX idx3 ON tb3(b) IN db4;

ALTER FRAGMENT ON TABLE tb1
   ATTACH tb2 AS (a >= 10 and a<15) tb3 AS (a >= 15 and a<20);
```

三个 CREATE INDEX 语句自动计算 HIGH 方式中每个索引的主列的分布统计信息，以及 LOW 方式的索引统计信息和表统计信息。

唯一需要 UPDATE STATISTICS LOW FOR TABLE 语句的时间是在某个情境的 CREATE INDEX 语句之后，在该情境中表具有其他预先存在的索引，如此示例中所示：

```
CREATE TABLE tb1(col1 int, col2 int);
CREATE INDEX index idx1 on tb1(col1);
  (equivalent to update stats low on table tb1)
LOAD from tb1.unl insert into tb1; (load some data)
CREATE INDEX idx2 on tb1(col2);
```

由于 CREATE INDEX 语句不更新名称为 idx1 的预先存在索引的索引级别统计信息，因而语句 CREATE INDEX idx2 on tb1(col2) 不完全等同于 UPDATE STATISTICS LOW FOR TABLE tb1。

在上述示例中，表 tb3 没有对列 a 的索引，该索引可以作为所生成索引 idx1 的分段。数据库服务器估算对已用完的表 tb3 的列 a 建立索引分段的成本，并将此成本与为生成表的所有分段重建整个索引的成本相比较。数据库服务器会选择成本较低的索引构建。

表的索引不可用时的情境示例
表的索引不可用时，数据库服务器会估算构建索引分段的成本，并将此成本与重新构建生成表的所有分段的整个索引相比，选择成本较少的索引构建。

假设您按照前面的部分创建了表和索引，但第三个表的索引指定了第一个表也使用的数据库空间。 以下 SQL 语句显示了此种应用场合：

```
CREATE TABLE tb1(a int, b int)
   FRAGMENT BY EXPRESSION
      (a >=0 AND a < 5) IN db1,
      (a >=5 AND a <10) IN db2;
CREATE INDEX idx1 ON tb1(a);
CREATE TABLE tb2 (a int, b int, check (a >=10 and a<15))
   IN db3;
CREATE INDEX idx2 ON tb2(a)
   IN db3;

CREATE TABLE tb3 (a int, b int, check (a >= 15 and a<20))
   IN db4;
CREATE INDEX idx3 ON tb3(a)
   IN db2 ;
```

此示例对数据库空间 db2 中的表 tb3 创建了索引 idx3。因此，索引 idx3 不可用，因为索引 idx1 在数据库空间 db2 中已经具有一个分段，并且分段存储策略不允许在给定的数据库空间中指定多个分段。

数据库服务器再次估算对已用完的表 tb3 的列 a 建立索引分段的成本，并将此成本与为生成表的所有分段重建整个索引 idx1 的成本相比较。然后，数据库服务器会选择成本较低的索引构建。

提高 ALTER FRAGMENT DETACH 的性能

可以为表和索引分段制定相应的分布方案，以及消除执行 ALTER FRAGMENT DETACH 语句期间的索引构建来提高 ALTER FRAGMENT DETACH 语句的性能。

要在 ALTER FRAGMENT DETACH 语句的执行过程中消除索引建立，请使用以下某个分段存储策略：

- 以对表所用的相同方法对索引进行分段。
- 以对表所用的相同分布方案对索引进行分段。

> 重要： 只有使用日志记录的数据库能够从 ALTER FRAGMENT DETACH 语句的性能提高中获益。如果没有日志记录，数据库服务器将使用同一个表的多个副本，以确保发生故障时数据的可恢复性。此要求无法重新使用现有索引分段。

以对表所用的相同方法对索引进行分段
如果在创建分段表并随后创建索引时未指定分段存储策略，那么使用将表分段的相同方法来对索引分段，除非分布方案为循环的且启用自动定位。当 AUTOLOCATE 配置参数或环境选项设为正整数时，使用循环分布方案的表上的索引不会进行分段。

例如，假设您用以下 SQL 语句创建分段表和索引：

```
CREATE TABLE tb1(a int)
   FRAGMENT BY EXPRESSION
      (a >=0 AND a < 5) IN db1,
      (a >=5 AND a <10) IN db2,
      (a >=10 AND a <15) IN db3;
CREATE INDEX idx1 ON tb1(a);
```

数据库服务器将索引键分段到数据库空间 db1、db2 和 db3 中，分段时使用的列 a 值的范围与表的相同，因为 CREATE INDEX 语句没有指定分段存储策略。

假设您接着决定用以下 SQL 语句拆离第三个分段中的数据：

```
ALTER FRAGMENT ON TABLE tb1
```

```
   DETACH db3 tb3;
```

由于索引的分段存储策略与表的相同，因而 ALTER FRAGMENT DETACH 语句在拆离操作后不重建索引。数据库服务器会删除数据库空间 db3 中的索引分段，更新系统目录表，并消除索引构建。

使用与表相同的分布方案对索引进行分段
如果创建的索引使用与表相同的分段表达式，那么应用与表相同的分布方案来对索引进行分段。

公共分段存储策略使用与表相同的方法来对索引进行分段，但是它为索引分段指定不同的数据库空间。此分段存储策略把索引分段与表分别放在不同的数据库空间中，可提高一些操作的性能，如备份、恢复等等。

例如，假设您用以下 SQL 语句创建分段表和索引：

```
CREATE TABLE tb1(a int, b int)
   FRAGMENT BY EXPRESSION
      (a >=0 AND a < 5) IN db1,
      (a >=5 AND a <10) IN db2,
      (a >=10 AND a <15) IN db3;

CREATE INDEX idx1 on tb1(a)
   FRAGMENT BY EXPRESSION
      (a >=0 AND a< 5) IN db4,
      (a >=5 AND a< 10) IN db5,
      (a >=10 AND a<15) IN db6;
```

假设您接着决定用以下 SQL 语句拆离第三个分段中的数据：

```
ALTER FRAGMENT ON TABLE tb1
   DETACH db3 tb3;
```

由于索引的分布方案与表的相同，因而 ALTER FRAGMENT DETACH 语句在拆离操作后不重建索引。数据库服务器删除数据库空间 db3 中的索引分段，更新系统目录表，并消除索引构建。

变更表分段时强行排除事务

在使用日志记录的数据库中，您可以启用服务器强行排除 ALTER FRAGMENT ON TABLE 操作的目标表中打开的或持有锁的事务。具有 DBA 访问权限的用户可以通过启用 SET ENVIRONMENT 语句的 FORCE_DDL_EXEC 会话环境选项来完成此操作。

如果您不想在变更分段前等待会话关闭，那么您可能希望对忙碌的系统执行此操作，也许是一天运行 24 小时的系统。

但是，请注意通过强行排除并行事务来避免等待锁释放，数据库服务器将会关闭 Update 游标并回滚其他用户的事务。

先决条件：

- 必须是用户 informix 或具有数据库的 DBA 访问权限。
- 表必须在支持事务日志记录的数据库中。

要在变更表分段时强行排除其他会话的并行事务：

1. 将 SET ENVIRONMENT 语句的 FORCE_DDL_EXEC 环境选项设置为以下某个值：
   - ON、on、'1' 或 "1"，以启用服务器强行排除发出 ALTER FRAGMENT ON TABLE 语句时表中打开的或锁定的事务，直至服务器获取表的锁和互斥访问。
   - 代表时间量（以秒为单位）的正整数。该数字值会启用服务器强行排除事务，直至服务器获取表的互斥访问和互斥锁，或直至达到指定的时间限制。如果服务器无法按指定的时间量强行排除事务，那么服务器将停止尝试强行排除事务，并且在并行事务提交或回滚时，ALTER FRAGMENT 语句会等待锁释放。

例如，要启用 FORCE_DDL_EXEC 环境选项以在发出 ALTER FRAGMENT ON TABLE 语句时运行 100 秒，请指定：

```
SET ENVIRONMENT FORCE_DDL_EXEC '100';
```

2. 将锁模式设置为等待以确保服务器将在强行排除任意事务前等待指定的时间量。

   例如，要将锁模式设置为等待 20 秒，请指定：

```
SET LOCK MODE TO WAIT "20";
```

   有关更多信息，请参阅将锁模式设置为等待 在第176页。

3. 执行一个 ALTER FRAGMENT ON TABLE 语句，例如，要连接、拆离、修改、添加或删除分段。

   以下 SQL 语句会执行以下操作：

   - 启用 FORCE_DDL_EXEC 会话环境选项 100 秒，
   - 将数据库服务器设置为最多等待 25 秒以释放锁，
   - 并且更改表 tabF 的范围分段 p2 的时间间隔大小和存储位置：

```
SET ENVIRONMENT FORCE_DDL_EXEC '100';
SET LOCK MODE TO WAIT 25;
ALTER FRAGMENT ON TABLE tabF MODIFY
   PARTITION p2 TO PARTITION p2 VALUES < 500 IN dbs0;
```

   注意：

   当上述 ALTER FRAGMENT 语句正在运行时，如果尝试访问表 tabF 中的行的其他事务的 Update 游标对分段 p2 中的行持有锁定，那么这些事务将面临被强行排除的风险。

   由于 FORCE_DDL_EXEC 会话环境选项由另一个会话启用而回滚事务后，数据库服务器会将此错误返回到事务失败的会话：

```
-458  Long transaction aborted.
```

   并行事务失败，并返回不一定是“long”的错误 -458，但是在打开或持有与此示例中的 ALTER FRAGMENT 语句修改的同一个表上的锁之后，该事务尚未提交。

当您完成启用了 FORCE_DDL_EXEC 会话环境选项的 ALTER FRAGMENT ON TABLE 操作后，可以将 FORCE_DDL_EXEC 会话环境选项关闭。例如，请指定：

```
SET ENVIRONMENT FORCE_DDL_EXEC OFF;
```

相关链接

《SinoDB SQL 指南: 语法》: FORCE_DDL_EXEC 会话环境选项

监视分段使用情况

一旦确定了分段存储策略，您就可以监视分段存储。

可以用以下方法来监视分段存储：

- 执行单个 onstat 实用程序命令来获取有关正在运行的查询的特定方面的信息。
- 执行 SET EXPLAIN 语句，然后执行查询将查询计划写入输出文件中。

使用 onstat -g ppf 命令监视分段存储

使用 onstat -g ppf 命令，您可以查看分区信息并监视 I/O 活动以验证策略并确定各分段之间 I/O 是否均衡。

onstat -g ppf 输出包括发送给每个当前打开的分段的读写请求数。由于请求可能触发多个 I/O 操作，因而这些请求无法说明发生多少单个磁盘 I/O 操作，但您可以从显示列中清楚地了解 I/O 活动。

输出中的 brfd 列显示页中缓冲区读取的次数。（每个缓冲区可包含一个页。）如果需要监视执行查询所用的时间，此信息很有用。通常，查询执行时间很大程度上取决于所需缓冲区读取数。如果客户端/服务器缓冲区的大小很小而您的数据库包含 TEXT 数据，那么查询执行时间受缓冲区读取数的影响会大很多，因为服务器会读取先前 TEXT 数据。

onstat -g ppf 输出本身不会标识分段所在的表。要确定该分段的表，请将输出中的 partnum 列与 sysfragments 系统目录表中的 partnum 列连接起来。sysfragments 表显示相关的 table id。也可以通过将 sysfragments 中的 table id 列与 systables 中的 table id 列连接来查找分段的表名。

要确定 onstat -g ppf 输出中的表名：

1. 获取 onstat -g ppf 输出的 partnum 字段中的值。
2. 将 sysfragments 系统目录表的 tabid 列与 systables 系统目录表的 tabid 列连接起来，以从 systables 中获得表名。

   在 SELECT 语句中，使用在步骤 1 中获取的 partnum 字段值。

```
SELECT a.tabname FROM systables a, sysfragments b
   WHERE a.tabid = b.tabid
      AND partn = partnum_value;
```

使用 SET EXPLAIN 输出监视分段存储

对表进行分段后，SET EXPLAIN ON 语句的输出将显示数据库服务器扫描哪个表或索引以执行查询。

SET EXPLAIN 输出使用分段号标识分段。这里的分段号与那些包含于 sysfragments 系统目录表中的 partn 列的分段号相同。

以下部分 SET EXPLAIN 输出的示例显示了一个查询，该查询利用分段消除并扫描表 t1 中的两个分段：

```
QUERY:
------
SELECT * FROM t1 WHERE c1 > 12

Estimated Cost: 3
Estimated # of Rows Returned: 2

1) informix.t1: SEQUENTIAL SCAN (Serial, fragments: 1, 2)

   Filters: informix.t1.c1 > 12
```

如果优化器必须扫描所有分段（也就是说，如果它不能从注意事项中消除任何分段），那么 SET EXPLAIN 输出将显示分段：ALL。此外，如果优化器从注意事项中消除了所有分段（也就是说，没有任何分段包含查询的信息），那么 SET EXPLAIN 输出将显示分段：NONE。

有关数据库服务器如何从注意事项中消除分段的信息，请参阅消除分段的分布方案 在第197页。

有关 SET EXPLAIN ON 语句的更多信息，请参阅显示由优化器选择的查询计划的报告 在第216页。

第 10 章

查询和查询优化器

以下主题将描述查询计划，说明数据库服务器如何管理查询优化，并讨论可用于影响查询计划的因素。这些主题也会描述 SPL 例程、UDR 高速缓存和触发器的性能注意事项。

数据库服务器中的并行数据库查询 (PDQ) 功能可以最大限度地提高查询的潜在性能。并行数据库查询 *(PDQ)* 在第251页 对 PDQ 和内存分配管理器 (MGM) 进行了描述并说明如何控制查询的资源使用。

如果您按照分段存储准则 在第188页所描述的对表进行分段，那么 PDQ 可使性能得到最大限度的改进。

提高个别查询性能 在第265页说明了如何提高特定查询的性能。

查询计划

查询优化器对执行查询的不同方式进行评估，并确定最好的方式来查询请求的数据。在此评估过程中，优化器制订了一个查询计划 来访存处理查询所需的数据行。

例如，在评估可能执行查询的不同方式时，优化器必须确定是否应该使用索引。如果查询包含了 join，那么优化器必须确定 join 计划（散列或嵌套循环）以及评估或连接表所用的顺序。

以下主题描述查询计划的组件并显示查询计划的示例。

存取计划

优化器选择用于读取表的方法称为存取计划。访问表的最简单方法是按顺序读取，这称为表扫描。当必须读取表的大部分或者表没有对查询十分有用的索引时，优化器将选择表扫描。

优化器也可以选择用索引来访问该表。如果索引中的列与查询的过滤器中的某个列相同，那么优化器可以使用索引仅检索查询所需的行。如果所请求的列在表的一个索引中，那么优化器可以使用仅键索引扫描。数据库服务器从索引检索需要的数据而不访问相关联的表。

> 重要： 优化器不会为 VARCHAR 列选择仅键扫描。如果您希望利用仅键扫描，那么请使用带 MODFIY 子句的 ALTER TABLE 将该列更改为 CHAR 数据类型。

优化器会比较每个计划的成本来确定最佳的计划。数据库服务器通过估算所需的 I/O 操作数、产生结果的计算量、访问的行数和排序等计算出成本。

连接计划

一个查询包含多个表时，SinoDB® 在查询中使用过滤器将这些表连接起来。优化器选择用于连接表的方法是连接计划。

在以下查询中，customer 和 orders 表由 customer.customer_num = orders.customer_num 过滤器连接起来：

```
SELECT * from customer, orders
WHERE customer.customer_num = orders.customer_num
AND customer.lname = "Higgins";
```

连接方法可以是嵌套循环连接或者是散列连接。

由于散列连接的特性，隔离级别设置为 Repeatable Read 的应用程序可能会暂时锁定表中涉及连接的所有记录，包含不满足连接的记录。此情况导致连接中的并发性减少。相反，嵌套循环连接锁定更少的记录，但在访问大量行时性能会降低。因此，每种连接方法都有优点和缺点。

嵌套循环连接
在嵌套循环连接中，数据库服务器扫描第一个表（或外表），然后将通过表过滤器的每一行与第二个表（或内表）中所找到的行连接起来。

图 43: 嵌套循环连接 在第211页显示了表和行以及查询读取它们的顺序：

```
SELECT * FROM customer, orders
WHERE customer.customer_num=orders.customer_num
AND order_date>"01/01/2007";
```

数据库服务器通过索引或通过表扫描访问外表。数据库服务器首先应用各种表过滤器。为了得到满足外表上的过滤器的每一行，数据库服务器将读取内表以查找匹配项。

为了得到外表中满足表过滤器的每一行，数据库服务器将读取一次内表。由于内表可能会被读取很多次，数据库服务器常常通过索引访问内表。

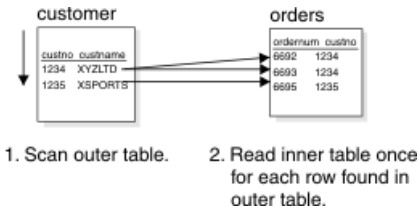

图 43: 嵌套循环连接

如果内表没有索引，那么数据库服务器可能会在查询执行时构造一个自动索引。优化器可能确定：在查询执行时构造一个自动索引的成本要比扫描内表来查找外表中每一符合条件的行的成本低。

如果优化器将某个子查询更改为嵌套循环连接，那么它可能使用嵌套循环连接的另一种变形形式，称为半连接。在半连接中，数据库服务器只读取内表一直到找到匹配项为止。换句话说，对于外表中的每一行，内表最多提供一行。有关优化器如何处理子查询的更多信息，请参阅子查询的查询计划 在第220页。

散列连接
当两个连接表中至少一个没有连接列的索引时或者当数据库服务器必须从两个表中读取许多行时，优化器常常使用散列连接。当数据库服务器执行散列连接时，任何索引和任何排序均不是必需的。

散列连接由两个活动组成：首先构建散列表（构建阶段），然后探测散列表（探测阶段）。图 44: 如何执行散列连接 在第212页显示了散列连接的详细信息。

在构建阶段，数据库服务器会读取一个表并在应用所有过滤器之后创建散列表。从概念上讲，散列表可看作一系列的存储区，每一个存储区都有一个通过应用散列功能从键值得到的地址。数据库服务器不对特定散列存储区中的键进行排序。

较少的散列表可以存放在数据库服务器共享内存的虚拟部分中。数据库服务器将磁盘上较大的散列文件存储在由 DBSPACETEMP 配置参数或由 DBSPACETEMP 环境变量所指定的数据库空间中。

在探测阶段，数据库服务器读取连接中的另一个表并应用所有的过滤器。对于符合表上的过滤器条件的每一行，数据库服务器将应用键上的散列功能并探测散列表以查找匹配行。

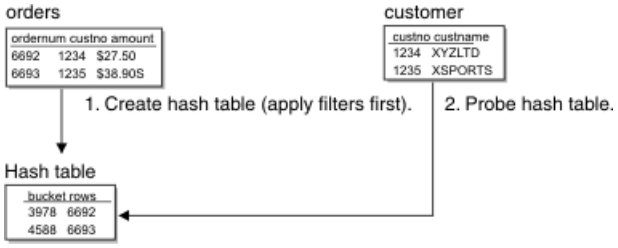

图 44: 如何执行散列连接

连接顺序
查询中表的连接顺序极其重要。连接顺序不当可能导致查询性能明显下降。

以下 SELECT 语句要求三向连接：

```
SELECT C.customer_num, O.order_num
   FROM customer C, orders O, items I
   WHERE C.customer_num = O.customer_num
      AND O.order_num = I.order_num
```

优化器可以选择以下连接顺序之一：

- 将 customer 连接到 orders。将结果连接到 items。
- 将 orders 连接到 customer。将结果连接到 items。
- 将 customer 连接到 items。将结果连接到 orders。
- 将 items 连接到 customer。将结果连接到 orders。
- 将 orders 连接到 items。将结果连接到 customer。
- 将 items 连接到 orders。将结果连接到 customer。

有关数据库服务器如何根据特定的连接顺序执行计划的示例，请参阅查询计划执行示例 在第212页。

查询计划执行示例

本主题包含使用调用三向连接的 SELECT 语句进行查询的示例，并描述一个可能的查询计划。

以下 SELECT 语句要求三向连接：

```
SELECT C.customer_num, O.order_num
   FROM customer C, orders O, items I
   WHERE C.customer_num = O.customer_num
      AND O.order_num = I.order_num
```

假定三个表都没有索引。假设优化器为两个连接都选择 customer-orders-items 路径和嵌套循环连接（实际上，优化器常常为两个没有连接列索引的表选择散列连接）。图 45: 以伪码编写的查询计划 在第212页显示了查询计划，该查询计划是以伪码编写的。有关说明查询计划信息的信息，请参阅显示由优化器选择的查询计划的报告 在第216页。

```
for each row in the customer table do:
   read the row into C
   for each row in the orders table do:
      read the row into O
      if O.customer_num = C.customer_num then
         for each row in the items table do:
            read the row into I
            if I.order_num = O.order_num then
               accept the row and send to user
            end if
         end for
      end if
   end for
```

```
end for
```

图 45: 以伪码编写的查询计划

该过程读取以下行：

- 读取 customer 表的所有行一次
- 读取 orders 表的所有行一次以查找 customer 表的每一行
- 读取 items 表的所有行一次以查找 customer-orders 对的每一行

该示例没有描述唯一可能的查询计划。另一个计划只是将 customer 和 orders 的角色对调了一下：对于 orders 的每一行，它读取 customer 的所有行，以寻找匹配的 customer_num。该计划按不同的顺序读取相同数量的行并按不同的顺序产生相同的行集合。在该示例中，两个可能的查询计划需要做的工作量不存在差异。

使用列过滤器的连接示例
列过滤器的存在会更改查询计划。列过滤器是 WHERE 表达式，该表达式减少表加入连接的行数。

以下示例显示了查询计划执行示例 在第212页中所述的带有添加的过滤器的查询：

```
SELECT C.customer_num, O.order_num
   FROM customer C, orders O, items I
   WHERE C.customer_num = O.customer_num
      AND O.order_num = I.order_num
      AND O.paid_date IS NULL
```

表达式 O.paid_date IS NULL 过滤掉一些行，从而减少 orders 表中所使用的行数。请设想一个计划，该计划是通过从 orders 读取开始的。图 *46:* 使用列过滤器的查询计划 在第213页以伪码显示该计划范例。

```
for each row in the orders table do:
   read the row into O
   if O.paid_date is null then
      for each row in the customer table do:
         read the row into C
         if O.customer_num = C.customer_num then
            for each row in the items table do:
               read the row into I
               if I.order_num = O.order_num then
                  accept row and return to user
               end if
            end for
         end if
      end for
   end if
end for
```

图 46: 使用列过滤器的查询计划

pdnull 表示 orders 中通过过滤器的行数。它是从以下查询得出的 COUNT(*) 的值：

```
SELECT COUNT(*) FROM orders WHERE paid_date IS NULL
```

如果每个 order 都有一个 customer，那么图 *46:* 使用列过滤器的查询计划 在第213页中的计划将读取以下行：

- 读取 orders 表的所有行一次
- 读取 customer 表的所有行 *pdnull* 次
- 读取 items 表的所有行 *pdnull* 次

图 *47: 以伪码编写的备用查询计划* 在第214页显示了一个备选的执行计划，该计划先从 customer 表读取。

```
for each row in the customer table do:
   read the row into C
   for each row in the orders table do:
      read the row into O
      if O.paid_date is null and
           O.customer_num = C.customer_num then
         for each row in the items table do:
            read the row into I
            if I.order_num = O.order_num then
               accept row and return to user
            end if
         end for
      end if
   end for
```

图 47: 以伪码编写的备用查询计划

因为过滤器没有在图 *47: 以伪码编写的备用查询计划* 在第214页显示的第一步中应用，所以该计划读取以下行：

- 读取 customer 表的所有行一次
- 读取 orders 表的所有行一次以查找 customer 的每一行
- 读取 items 表的所有行 *pdnull* 次

图 *46: 使用列过滤器的查询计划* 在第213页和图 *47: 以伪码编写的备用查询计划* 在第214页中的查询计划以不同的顺序产生相同的输出。这两个查询计划的不同之处在于：一个读取表 *pdnull* 次，而另一个读取表 SELECT COUNT (*) FROM customer 次。通过选择适当的计划，优化器可以将成千上万的磁盘存取保存在某个实际的应用程序中。

使用索引的连接示例
查询计划中使用索引和约束可为优化器提供选项，这些选项可以大大缩短查询执行时间。

本主题显示与使用列过滤器的连接示例 在第213页中所示查询不同的查询计划大纲，因为它是使用索引构建的。

```
for each row in the customer table do:
   read the row into C
   look up C.customer_num in index on orders.customer_num
   for each matching row in the orders index do:
      read the table row for O
      if O.paid_date is null then
         look up O.order_num in index on items.order_num
         for each matching row in the items index do:
            read the row for I
            construct output row and return to user
         end for
      end if
   end for
end for
```

图 48: 使用索引的查询计划

将索引中的关键字进行排序，以便当数据库服务器找到第一个匹配条目时，可以使用相同的关键字读取任何其他行而无需进行进一步的搜索，因为这些行是位于物理的相邻位置上。该查询计划只读取以下行：

- 读取 customer 表的所有行一次
- 读取 orders 表的所有行一次（因为每个 order 仅与一个 customer 相关联）
- 只读取 items 表中与 customer-orders 对的 *pdnull* 行相匹配的行

与不使用索引的计划相比，该查询计划明显降低了成本。同理，一个逆相计划也是可行的，该计划是先读取 orders，然后根据索引在 customer 中查找行。

表中行的物理顺序也将影响索引使用的成本。若表的排序方式与索引相关，那么按索引顺序访问多个表行的开销就会下降。例如，如果 orders 表行是根据 customer 号进行物理排序的，那么对某个给定的 customer 的顺序进行多次检索要比表随机排序时的检索快得多。

在某些情况下，使用索引可能会产生附加费用。有关更多信息，请参阅索引查找成本 在第229页。

包含索引自连接路径的查询计划

索引自连接是一种索引扫描，您可以将其视为一个含有很多小索引扫描的整体，其中每个索引扫描都具有一个由前导键列和非前导键列上的过滤器组成的唯一组合。

小索引扫描的集合构成了仅使用所有组合索引的子集的访问路径。该表在逻辑上是自连接的，并且更具选择性的非前导索引键应作为索引边界过滤器应用到前导键值的每个唯一组合。

索引自连接在以下情况中有优势：

- 索引的前导键有很多重复
- 前导键上的谓词不可选，但是非前导索引键上的谓词可选。

图 49: 包含索引自连接路径的查询的 *SET EXPLAIN* 输出 在第215页中的查询显示一个包含索引自连接路径的查询计划的 SET EXPLAIN 输出。

```
QUERY:
------
SELECT a.c1,a.c2,a.c3 FROM tab1 a WHERE (a.c3 >= 100103) AND
     (a.c3 <= 100104) AND (a.c1 >= 'PICKED      ') AND
     (a.c1 <= 'RGA2        ') AND (a.c2 >= 1) AND (a.c2 <= 7)
     ORDER BY 1, 2, 3

Estimated Cost: 155
Estimated # of Rows Returned: 1
  1) informix.a: INDEX PATH
    (1) Index Keys: c1 c2 c3 c4 c5   (Key-Only)  (Serial, fragments: ALL)
        Index Self Join Keys (c1 c2 )
          Lower bound: informix.a.c1 >= 'PICKED      ' AND (informix.a.c2 >= 1 )
          Upper bound: informix.a.c1 <= 'RGA2        ' AND (informix.a.c2 <= 7 )
        Lower Index Filter: (informix.a.c1 = informix.a.c1 AND
          informix.a.c2 = informix.a.c2 ) AND informix.a.c3 >= 100103
        Upper Index Filter: informix.a.c3 <= 100104
        Index Key Filters:  (informix.a.c2 <= 7 ) AND
                            (informix.a.c2 >= 1 )
```

图 49: 包含索引自连接路径的查询的 SET EXPLAIN 输出

在图 49: 包含索引自连接路径的查询的 *SET EXPLAIN* 输出 在第215页中，索引存在于 c1、c2、c3、c4 和 c5 列。优化器选择 c1 和 c2 作为前导键，这表明 c1 和 c2 列有很多重复。c3 列的重复很少，因此 c3 列（c3 >= 100103 和 c3 <= 100104）上的谓词具有好的选择性。

正如图 49: 包含索引自连接路径的查询的 *SET EXPLAIN* 输出 在第215页所示，索引自连接路径是使用相同索引的两个索引扫描的自连接。第一个索引扫描为前导键列（c1 和 c2）检索每个唯一值。然后使用 c1 和 c2 的唯一值探测第二个索引扫描，该扫描也使用 c3 列的谓词。因为 c3 列上的谓词具有良好的选择性：

- 嵌套循环连接内侧的索引扫描非常有效，它仅检索满足 c3 谓词的少数行。
- 索引扫描不会检索额外行。

因此，对于 c1 和 c2 的每个唯一值，在 c1、c2 和 c3 上发生了有效的索引扫描。

示例中的以下行表示优化器已经为该表选择了一个索引自连接路径，c1 和 c2 列作为该索引自连接路径的前导键：

```
Index Self Join Keys (c1 c2 )
```

```
        Lower bound: informix.a.c1 >= 'PICKED      ' AND (informix.a.c2 >= 1 )
        Upper bound: informix.a.c1 <= 'RGA2        ' AND (informix.a.c2 <= 7 )
```

此示例显示了 c1 和 c2 列的范围，可将其作为索引扫描的范围以检索索引的有效前导键。

该示例中的以下信息显示自连接：

```
(informix.a.c1 = informix.a.c1 AND informix.a.c2 = informix.a.c2 )
```

本信息表示内部索引扫描。对于自连接谓词使用的前导键列 c1 和 c2，指示了外部索引扫描的 c1 和 c2 的值。c3 列的谓词充当使内部索引扫描生效的索引过滤器。

规则索引扫描不使用 c3 列上的过滤器来定位索引扫描，因为前导键列 c1 和 c2 不具有等式谓词。

甚至当数据分布统计信息对于前导索引键列不可用的时候，INDEX_SJ 指令也可通过使用特定的索引或选择索引列表中成本最低的索引来强制索引自连接路径。AVOID_INDEX_SJ 指令可防止指定索引或索引的自连接路径。另请参阅访问方法指令 在第239页和《*SinoDB® SQL* 指南: 语法》。

查询计划评估

通过对诸如磁盘 I/O 和 CPU 成本之类的因素进行分析，优化器可对所有的查询计划进行评估。

优化器可使用由下至上、逐步缩小范围的搜索策略同时构造所有可行计划。也就是说，优化器首先构造所有可能的连接对。它会排除成本更高的任何冗余对。（冗余对是一种连接对，其中包含与另一连接对相同的表并生成与另一连接对相同的行集合。）

例如，如果两个连接都未通过使用 SELECT 语句的 ORDER BY 或 GROUP BY 子句指定经排序的行集合，那么连接对 (A x B) 相对于 (B x A) 就是冗余的。

如果查询使用附加的表，那么优化器将把每个余下的对连接到一个新表以形成所有可能的连接三元组，因而排除每个要连接的附加表的冗余三元组中成本较高的等等。当已生成一个可能的连接组合的非冗余集合时，优化器将选择执行成本看起来最低的计划。

显示由优化器选择的查询计划的报告

任何执行查询的用户都可以使用 SET EXPLAIN 语句或 EXPLAIN 指令来显示优化器所选择的查询计划。

有关如何指定指令的信息，请参阅*EXPLAIN* 指令 在第243页。用户要在查询的 SQL 语句之前输入 SET EXPLAIN ON 语句或 SET EXPLAIN ON AVOID_EXECUTE 语句，如以下示例所示。

```
SET EXPLAIN ON AVOID_EXECUTE;
SELECT * FROM customer, orders
WHERE customer.customer_num = orders.customer_num
   AND customer.lname = "Higgins";
```

如果用户无权访问 SQL 代码源，那么数据库管理员可以使用 onmode -Y 命令来动态设置 SET EXPLAIN。

数据库服务器执行 SET EXPLAIN ON 语句或使用 onmode -Y 命令动态设置 SET EXPLAIN 之后，服务器将把每个查询计划的说明写入文件以便用户输入的后续查询。

相关链接

说明输出文件 在第217页
查询统计信息部分提供性能调试信息 在第217页
《*SinoDB SQL* 指南: 语法》: *SET EXPLAIN* 语句
《*SinoDB SQL* 指南: 语法》: 使用 *FILE TO* 选项
《*SinoDB SQL* 指南: 语法》: *UNIX* 上说明输出文件的缺省名称和位置
《*SinoDB SQL* 指南: 语法》: *Windows* 上输出文件的缺省名称和位置
显示由优化器选择的查询计划的报告 在第216页
查询统计信息部分提供性能调试信息 在第217页
启用外部指令 在第249页
《*SinoDB* 管理员参考》: *onmode -Y*: 动态更改 *SET EXPLAIN*

《*SinoDB* 管理员参考》: *onmode* 和 *Y* 参数: 更改会话的查询计划度量 *(SQL* 管理 *API)*
显示由优化器选择的查询计划的报告 在第216页
启用外部指令 在第249页

说明输出文件
SET EXPLAIN 语句会启用或禁用对当前会话中查询的记录评估，包括查询优化器的计划、返回的行数的估算以及查询的相关成本。评估显示在输出文件中。

执行 onmode -Y 命令来打开动态 SET EXPLAIN 时，输出将显示在新的说明输出文件中。如果来自 SET EXPLAIN 语句的文件存在，那么数据库服务器将停止使用该文件，改为使用 onmode -Y 创建的文件，直到管理员关闭会话的动态 SET EXPLAIN 为止。

输出文件会指定外部指令是否有效。

说明输出文件的查询统计信息部分中的以下代码提供有关外部表的信息：

- xlcnv 表示载入外部表数据，并将数据插入基本表的操作。其中，x = 外部表，l = 载入，而 cnv = 转换器
- xucnv 表示卸载外部表数据，并将数据插入基本表的操作。其中，x = 外部表，u = 卸载，而 cnv = 转换器

说明输出文件的查询统计信息部分是对调试性能问题很有用的资源。请参阅查询统计信息部分提供性能调试信息 在第217页。

相关链接

《*SinoDB SQL* 指南: 语法》: *SET EXPLAIN* 语句
《*SinoDB SQL* 指南: 语法》: 使用 *FILE TO* 选项
《*SinoDB SQL* 指南: 语法》: *UNIX* 上说明输出文件的缺省名称和位置
《*SinoDB SQL* 指南: 语法》: *Windows* 上输出文件的缺省名称和位置
显示由优化器选择的查询计划的报告 在第216页
查询统计信息部分提供性能调试信息 在第217页
启用外部指令 在第249页
《*SinoDB* 管理员参考》: *onmode -Y:* 动态更改 *SET EXPLAIN*
《*SinoDB* 管理员参考》: *onmode* 和 *Y* 参数: 更改会话的查询计划度量 *(SQL* 管理 *API)*
查询统计信息部分提供性能调试信息 在第217页
启用外部指令 在第249页

查询统计信息部分提供性能调试信息
如果启用了 EXPLAIN_STAT 配置参数，那么查询统计信息部分将显示在 SQL 的 SET EXPLAIN 语句和 onmode -Y ***session_id*** 命令所显示的说明输出文件中。

说明输出文件中的查询统计信息部分显示了查询计划预期返回的估算行数、实际返回的行数以及有关查询的其他信息。可以使用查询计划整体流程和每个阶段所处理的行量的信息，来调试性能问题。

以下示例显示了 SET EXPLAIN 输出中的查询统计信息。如果已扫描或已连接的估算行数和实际行数相差很大，那么有关这些表的统计信息可能是旧的，应该更新。

```
select * from tab1, tab2 where tab1.c1 = tab2.c1 and tab1.c3 between 0 and 15

Estimated Cost: 104
Estimated # of Rows Returned: 69

  1) zelaine.tab2: SEQUENTIAL SCAN

  2) zelaine.tab1: INDEX PATH

    (1) Index Keys: c1 c3   (Serial, fragments: ALL)
        Lower Index Filter: (zelaine.tab1.c1 = zelaine.tab2.c1
                             AND zelaine.tab1.c3 >= 0 )
```

```
        Upper Index Filter: zelaine.tab1.c3 <= 15
NESTED LOOP JOIN

Query statistics:
-----------------

  Table map :
  ----------------------------
  Internal name     Table name
  ----------------------------
  t1                tab2
  t2                tab1

  type     table  rows_prod  est_rows  rows_scan  time       est_cost
  ------------------------------------------------------------------
  scan     t1     50         50        50         00:00:00   4

  type     table  rows_prod  est_rows  rows_scan  time       est_cost
  ------------------------------------------------------------------
  scan     t2     67         69        4          00:00:00   2

  type     rows_prod  est_rows  time       est_cost
  ----------------------------------------------------
  nljoin   67         70        00:00:00   104
```

图 50: SET EXPLAIN 输出中的查询统计信息

相关链接

说明输出文件 在第217页
《SinoDB SQL 指南: 语法》: SET EXPLAIN 语句
《SinoDB SQL 指南: 语法》: 使用 FILE TO 选项
《SinoDB SQL 指南: 语法》: UNIX 上说明输出文件的缺省名称和位置
《SinoDB SQL 指南: 语法》: Windows 上输出文件的缺省名称和位置
显示由优化器选择的查询计划的报告 在第216页
说明输出文件 在第217页
查询计划示例报告 在第218页
启用外部指令 在第249页
《SinoDB 管理员参考》: onmode -Y: 动态更改 SET EXPLAIN
《SinoDB 管理员参考》: onmode 和 Y 参数: 更改会话的查询计划度量 (SQL 管理 API)
显示由优化器选择的查询计划的报告 在第216页
启用外部指令 在第249页

查询计划示例报告

本部分中的主题描述查询计划示例，在分析不同种类查询的性能时您可能希望显示这些计划。

相关链接

查询统计信息部分提供性能调试信息 在第217页

单表查询

本主题显示单个表的简单查询和复杂查询的 SET EXPLAIN 输出示例。

图 51: 简单查询的部分 SET EXPLAIN 输出 在第218页显示了简单查询的 SET EXPLAIN 输出。

```
QUERY:
------
SELECT fname, lname, company FROM customer

Estimated Cost: 2
```

```
Estimated # of Rows Returned: 28

  1) virginia.customer: SEQUENTIAL SCAN
```

图 51: 简单查询的部分 SET EXPLAIN 输出

图 *52:* 复杂查询的部分 *SET EXPLAIN* 输出 在第219页显示了 customer 表上复杂查询的 SET EXPLAIN 输出。

```
QUERY:
------
SELECT fname, lname, company FROM customer
WHERE company MATCHES 'Sport*' AND
    customer_num BETWEEN 110 AND 115
ORDER BY lname

Estimated Cost: 1
Estimated # of Rows Returned: 1
Temporary Files Required For: Order By

  1) virginia.customer: INDEX PATH

        Filters: virginia.customer.company MATCHES 'Sport*'

    (1) Index Keys: customer_num   (Serial, fragments: ALL)
        Lower Index Filter: virginia.customer.customer_num >= 110
        Upper Index Filter: virginia.customer.customer_num <= 115
```

图 52: 复杂查询的部分 SET EXPLAIN 输出

图 *52:* 复杂查询的部分 *SET EXPLAIN* 输出 在第219页中的以下输出行显示了第二个查询的索引扫描的范围：

- Lower Index Filter: virginia.customer.customer_num >= 110

  以索引键值 110 开始索引扫描。
- Upper Index Filter: virginia.customer.customer_num <= 115

  以索引键值 115 停止索引扫描。

多表查询
本主题显示多表查询的 SET EXPLAIN 输出示例。

```
QUERY:
------
SELECT C.customer_num, O.order_num, SUM (I.total_price)
FROM customer C, orders O, items I
WHERE C.customer_num = O.customer_num
AND O.order_num = I.order_num
GROUP BY C.customer_num, O.order_num

Estimated Cost: 78
Estimated # of Rows Returned: 1
Temporary Files Required For: Group By

  1) virginia.o: SEQUENTIAL SCAN

  2) virginia.c: INDEX PATH

    (1) Index Keys: customer_num   (Key-Only)  (Serial, fragments: ALL)
        Lower Index Filter:
          virginia.c.customer_num = virginia.o.customer_num
NESTED LOOP JOIN
```

```
  3) virginia.i: INDEX PATH

    (1) Index Keys: order_num   (Serial, fragments: ALL)
        Lower Index Filter: virginia.o.order_num = virginia.i.order_num
NESTED LOOP JOIN
```

图 53: 多表查询的部分 SET EXPLAIN 输出

SET EXPLAIN 输出列出数据库服务器访问表的顺序和读取每个表的存取计划。图 *53: 多表查询的部分 SET EXPLAIN 输出* 在第219页中的计划指示数据库服务器将执行以下操作：

1. 数据库服务器将先读取 orders 表。

   因为 orders 表上没有过滤器，所以数据库服务器必须读取所有的行。按物理顺序读取表是成本最低的方法。
2. 对于 orders 的每一行，数据库服务器将在 customer 表中搜索匹配的行。

   搜索使用 customer_num 的索引。标志仅键意味着对于 customer 表只需要读取索引，因为只有 c.customer_num 列用在连接和输出中且该列是索引键。
3. 对于 orders 中具有一个匹配 customer_num 的每一行，数据库服务器将使用 order_num 的索引在 items 表中搜索匹配行。

键优先扫描
本主题显示使用键优先扫描 的查询示例，该扫描是一种索引扫描，其使用未列为低索引过滤器和高索引过滤器的键。

```
create index idx1 on tab1(c1, c2);

select * from tab1 where (c1 > 0) and ( (c2 = 1) or (c2 = 2))
Estimated Cost: 4
Estimated # of Rows Returned: 1

1) pubs.tab1: INDEX PATH

    (1) Index Keys: c1 c2 (Key-First) (Serial, fragments: ALL)
   Lower Index Filter: pubs.tab1.c1 > 0
   Index Key Filters:  (pubs.tab1.c2 = 1 OR pubs.tab1.c2 = 2)
```

图 54: 键优先扫描的部分 SET EXPLAIN 输出

即使在此示例中数据库服务器必须最终读取行数据以返回查询结果，数据库服务器将先通过应用附加的键过滤器尝试减少可能的行数。数据库服务器使用索引来应用附加的过滤器 c2 = 1 OR c2 = 2 之后才读取行数据。

子查询的查询计划
如果连接的成本较低，那么优化器可自动将子查询更改成连接。

例如，图 *55: 平铺子查询的部分 SET EXPLAIN 输出* 在第220页中 SET EXPLAIN ON 语句的输出示例显示了优化器将子查询中的表更改为连接中的内表。

```
QUERY:
------
SELECT company, fname, lname, phone
FROM customer c
WHERE EXISTS(
   SELECT customer_num FROM cust_calls u
      WHERE c.customer_num = u.customer_num)

Estimated Cost: 6
Estimated # of Rows Returned: 7

  1) virginia.c: SEQUENTIAL SCAN
```

```
  2) virginia.u: INDEX PATH  (First Row)

    (1) Index Keys: customer_num call_dtime   (Key-Only)
                                              (Serial, fragments: ALL)
        Lower Index Filter: virginia.c.customer_num = virginia.u.customer_num
NESTED LOOP JOIN  (Semi Join)
```

图 55: 平铺子查询的部分 SET EXPLAIN 输出

有关 SET EXPLAIN ON 语句的更多信息，请参阅显示由优化器选择的查询计划的报告 在第216页。

当优化器将子查询更改成连接时，可以使用存取计划和连接计划的几种变形形式：

- 首行（First-row）扫描

  首行扫描是表扫描的一种变形形式。当数据库服务器找到一个匹配时，表扫描将停止。
- 忽略重复索引（Skip-duplicate-index）扫描

  忽略重复索引扫描是索引扫描的一种变形形式。数据库服务器不扫描副本。
- 半连接（Semi join）

  半连接是嵌套循环连接的一种变形形式。当第一个匹配找到时，数据库服务器将停止内表扫描。有关半连接的更多信息，请参阅嵌套循环连接 在第211页。

集合派生表的查询计划

集合派生表是数据库服务器用以处理有关集合的查询的特殊方法。要使用集合派生表，查询必须在 SQL 语句的 FROM 子句中包含 TABLE 关键字。

有关如何在 SQL 语句中使用集合派生表的更多信息，请参阅《*SinoDB® SQL* 指南*:* 语法》。

虽然数据库并没有真的为集合创建一个表，但是数据库还是将其视为表来处理数据。在某些情况下，集合派生表允许开发者使用较少的游标和主变量来访问集合。

这些 SQL 语句创建一个称为 children 的集合列：

```
CREATE ROW TYPE person(name CHAR(255), id INT);
CREATE TABLE parents(name CHAR(255),
id INT,
children LIST(person NOT NULL));
```

以下查询为 children 列创建了一个集合派生表并将该集合的元素视为表中的行：

```
SELECT name, id
FROM TABLE(MUTLISET(SELECT children
FROM parents
WHERE parents.id
= 1001)) c_table(name, id);
```

此外，您也可以在 FROM 子句中指定集合派生表，如下例所示：

```
SELECT name, id
FROM (SELECT children
FROM parents
WHERE parents.id
= 1001) c_table(name, id);
```

显示数据库服务器如何完成查询的示例

在完成对集合派生表的查询时，SinoDB® 会执行若干步骤。

完成查询后，数据库服务器将执行本示例中显示的以下步骤：

1. 扫描 parent 表查找满足 parents.id = 1001 的行

该操作被列为图 56: 使用集合派生表的查询计划 在第222页显示的 SET EXPLAIN 输出中的 SEQUENTIAL SCAN。

2. 读取称为 children 的集合列的值。
3. 扫描单个集合并将 name 和 id 的值返回至应用程序。

   该操作被列为图 56: 使用集合派生表的查询计划 在第222页显示的 SET EXPLAIN 输出中的 COLLECTION SCAN。

```
QUERY:
------
SELECT name, id
FROM (SELECT children
FROM parents
WHERE parents.id
= 1001) c_table(name, id);

Estimated Cost: 2
Estimated # of Rows Returned: 1

  1) lsuto.c_table: COLLECTION SCAN
    Subquery:
    ---------
    Estimated Cost: 1
    Estimated # of Rows Returned: 1

      1) lsuto.parents: SEQUENTIAL SCAN

            Filters: lsuto.parents.id = 1001
```

图 56: 使用集合派生表的查询计划

并入父查询中的派生表
可以通过使用 SQL 将简单查询中的派生表并入父查询中（而不是并入放在临时表中的查询结果）来提高集合派生表的性能。

按照以下所示使用 SQL：

```
select * from ((select col1, col2 from tab1)) as vtab(c1,c2)
```

但是，如果查询由于涉及聚合、ORDER BY 操作或 UNION 操作而变得复杂，那么服务器将创建一个临时表。

数据库服务器使用与服务器通过 IFX_FOLDVIEW 配置参数折叠视图相似的方式折叠派生表（如启用视图折叠以提高查询性能 在第288页所述）。启用 IFX_FOLDVIEW 配置参数时，视图将并入父查询中。没有将视图并入放在临时表里的查询结果中。

以下示例显示并入主查询中的派生表。

```
select * from ((select vcol0, tab1.col1 from
        table(multiset(select col2 from tab2 where col2 > 50 ))
 vtab2(vcol0),tab1 )) vtab1(vcol1,vcol2)
 where vcol1 = vcol2

Estimated Cost: 2
Estimated # of Rows Returned: 1

  1) informix.tab2: SEQUENTIAL SCAN

        Filters: informix.tab2.col2 > 50

  2) informix.tab1: SEQUENTIAL SCAN

        Filters:
```

```
        Table Scan Filters: informix.tab1.col1 > 50

DYNAMIC HASH JOIN
    Dynamic Hash Filters: informix.tab2.col2 = informix.tab1.col1
```

图 57: 使用并入父查询中的派生表的查询计划

```
select * from (select col1 from tab1 where col1 = 100) as vtab1(c1)
left join (select col1 from tab2 where col1 = 10) as vtab2(vc1)
on  vtab1.c1 =  vtab2.vc1

Estimated Cost: 5
Estimated # of Rows Returned: 1

      1) informix.tab1: SEQUENTIAL SCAN

            Filters: informix.tab1.col1 = 100

      2) informix.tab2: AUTOINDEX PATH

        (1) Index Keys: col1   (Key-Only)
            Lower Index Filter: informix.tab1.col1 = informix.tab2.col1
            Index Key Filters:  (informix.tab2.col1 = 10 )

    ON-Filters:(informix.tab1.col1 = informix.tab2.col1
            AND informix.tab2.col1 = 10 )
    NESTED LOOP JOIN(LEFT OUTER JOIN)
```

图 58: 使用并入父查询中的派生表的第二个查询计划

以下示例显示涉及 UNION 操作的复杂查询。此时，已经创建了一个临时表。

```
select * from (select col1 from tab1 union select col2 from tab2 )
as vtab(vcol1)  where vcol1 < 50

Estimated Cost: 4
Estimated # of Rows Returned: 1

  1) (Temp Table For Collection Subquery): SEQUENTIAL SCAN
```

图 59: 创建临时表的复杂派生表查询

IBM® Data Studio 中的 XML 查询计划

IBM® Data Studio 由用于管理、数据建模以及根据数据服务器的数据构建查询的一组工具组成。EXPLAIN_SQL 例程准备查询并以 XML 格式返回查询计划。IBM® Data Studio Administration Edition 可以使用 EXPLAIN_SQL 例程以获取 XML 格式的查询计划，解释该 XML，并直观呈现该计划。

如果计划使用 IBM® Data Studio 来获取 Visual Explain 输出，那么必须在 onconfig 文件中为 SBSPACENAME 配置参数创建并指定缺省的智能大对象空间名称。EXPLAIN_SQL 例程在此智能大对象空间中创建 BLOB。

有关使用 IBM® Data Studio 的信息，请参阅 IBM® Data Studio 文档。

影响查询计划的因素

优化器确定查询计划时，将给每个可能的计划分配一个成本，然后选择成本最低的那个计划。优化器分析若干因素以确定每个查询计划的成本。

优化器用于确定每个查询计划成本的一些因素为：

- 与每个文件系统访问相关联的 I/O 请求数
- 确定哪些行满足查询谓词所需的 CPU 工作
- 对数据进行排序或分组所需的资源
- 查询可用的内存量（由 DS_TOTAL_MEMORY 和 DS_MAX_QUERIES 参数指定）

要计算每个可能的查询计划的成本，优化器将执行以下操作：

- 使用描述表数据和索引的特性和物理特征的一组统计信息
- 检查查询过滤器
- 检查可在计划中使用的索引
- 使用移动数据的成本为分布式查询执行本地或远程连接

对于在跨服务器操作中访问远程表的查询，某些特征可以显著降低性能（这是相对于对本地数据库中的表和视图执行的相应 DML 操作而言）。那些可能会限制远程表性能的查询规范包括以下规范：

- ANSI LEFT OUTER JOIN 语法
- 基于远程表派生的表
- 作为引用远程表的物化视图的 TEMP 表。

对远程视图的限制

对于涉及远程视图的查询的多次执行，可进行重新优化。即使启用语句高速缓存，优化器也不会从语句高速缓存中选取查询计划。

为表和索引保留的统计信息

优化器可评估查询计划执行成本的精确程度取决于查询优化器可用的信息。使用 UPDATE STATISTICS 语句来维护有关表及其相关联索引的简单统计信息。更新的统计信息为查询优化器提供信息，该信息可以使执行那个表上的查询所需的时间最短。

当某个表创建时，数据库服务器将启动该表的统计概要文件，且在发出 UPDATE STATISTICS 语句时会刷新该概要文件。查询优化器不会自动重新计算表的概要文件。在某些情况下，收集统计信息花费的时间可能比执行查询花费的时间长。

为了确保优化器选择一个最能反映表当前状态的查询计划，应定期执行 UPDATE STATISTICS。有关准则，请参阅未自动生成统计信息时更新统计信息 在第274页。

优化器在创建查询计划时使用以下系统目录信息：

- 表中最新的 UPDATE STATISTICS 语句中的行数
- 是否将列限制为唯一的
- 使用 UPDATE STATISTICS 语句中的 MEDIUM 或 HIGH 关键字请求时列值的分布

  有关数据分布的更多信息，请参阅创建数据分布 在第275页。
- 包含行数据的磁盘页数

优化器也使用以下有关索引的系统目录信息：

- 存在于某个表中的索引，包含它们建立索引的列（无论它们是升序还是降序并且不管它们是否是集群的）
- 索引结构的深度（执行索引查找所需工作量的度量）
- 索引条目占用的磁盘页数
- 某个索引中的唯一条目数，它可用来估算等效过滤器返回的行数
- 被索引列中的次最大和次最小键值

由于极值可能有与列中其余数据不相关的特殊含义，因此只记录次最大和次最小键值。数据库服务器假定键值在次最大和次最小值之间均匀分布。只存储这些键的前 4 个字节。如果您为与某个索引相关联的列创建了分布，那么优化器将在估算与查询相匹配的行数时使用该分布。

有关系统目录表的更多信息，请参阅《*SinoDB® SQL* 指南*:* 参考》。

查询中的过滤器

查询优化器根据要从每个表中检索的行数进行查询成本估算。反过来，估算的行数是根据 WHERE 子句所使用的每个条件表达式的选择性确定的。用来选择行的条件表达式称为过滤器。

选择性是 0 与 1 间的某个值，该值指示表中过滤器可以通过的行的比例。一个选择性的过滤器，如果通过很少的行，那么其选择性接近 0，如果几乎通过所有行，那么过滤器的选择性接近 1。有关过滤器的准则，请参阅提高过滤器选择性 在第266页。

优化器可以使用数据分布计算查询中过滤器的选择性。但是，在没有数据分布的情况下，数据库服务器将根据表索引计算不同类型的过滤器的选择性。下表列出了优化器分配给不同类型的过滤器的一些选择性值。使用数据分布计算得出的选择性甚至要比该表显示的选择性更精确。

在该表中：

- *indexed-col* 是索引中的首列或唯一列。
- *2nd-max* 和 *2nd-min* 是索引列中的次最大和次最小键值。
- 加号 (+) 表示逻辑连接（相当于 Boolean OR 运算符），乘号 (x) 表示逻辑相交（相当于 Boolean AND 运算符）。

表 14: 优化器分配给不同类型过滤器的一些选择性值

过滤器表达式	选择性 (F)
indexed-col = *literal-valueindexed-col* = *host-variableindexed-col* IS NULL	F = 1/（索引中不同键的个数）
tab1.indexed-col = *tab2.indexed-col*	F = 1/（较大索引中不同键的个数）
indexed-col > *literal-value*	F = (2nd-max - *literal-value*)/(2nd-max - 2nd-min)
indexed-col < *literal-value*	F = (*literal-value* - 2nd-min)/(2nd-max - 2nd-min)
any-col IS NULL*any-col* = *any-expression*	F = 1/10
any-col > *any-expressionany-col* < *any-expression*	F = 1/3
any-col MATCHES *any-expressionany-col* LIKE *any-expression*	F = 1/5
EXISTS *subquery*	F = 1 if *subquery* estimated to return >0 rows, else 0
NOT *expression*	F = 1 - F(*expression*)
expr1 AND *expr2*	F = F(*expr1*) x F(*expr2*)
expr1 OR *expr2*	F = F(*expr1*) + F(*expr2*) - (F(*expr1*) x F(*expr2*))
any-col IN *list*	Treated as *any-col* = $item_1$ OR . . . OR *any-col* = $item_n$.
any-col relop ANY *subquery*	Treated as any-col relop $value_1$ OR . . . OR *any-col relop* $value_n$ for estimated size of subquery *n*. 其中 *relop* 是任何关系运算符，如 <、>、>= 和 <=。

用于评估过滤器的索引

查询优化器说明索引是否可用来评估过滤器。出于此目的的考虑，索引列可以是任意带有一个索引的单列，也可以是混合索引中指定的首列。

如果索引中包含的值为必需的所有值，那么数据库服务器将不读取这些行。只要数据库服务器可以直接从索引读取值，就可以更快速地忽略对数据页的页查找。

优化器可以对以下任一情况选择一个索引：

- 当列已索引，且要比较的值是文字值、主变量或是不相关的子查询时

  数据库服务器可以首先通过在合适的索引中查找行来确定表中相关行的位置。如果没有合适的索引，那么数据库服务器必须对每个表进行完全扫描。
- 当列已索引且要比较的值是另一个表（连接表达式）中的一个列时

  数据库服务器可以使用索引来查找匹配值。以下连接表达式显示了这样一个示例：

  ```
  WHERE customer.customer_num = orders.customer_num
  ```

  如果先读取 customer 的行，那么可以将 customer_num 的值应用于 orders.customer_num 的索引。
- 处理 ORDER BY 子句时

  如果子句中的所有列都按所要求的顺序出现在单索引中，那么数据库服务器可以使用该索引来按行既定的顺序读取，从而避免了排序。
- 处理 GROUP BY 子句时

  如果子句中的所有列都出现在一个索引中，那么数据库服务器可以从索引中读取具有相同键的组，而无需在行从表中检索出来之后再进行额外的处理。

PDQ 对查询计划的影响

打开并行数据库查询 (PDQ) 功能时，优化器可以选择并行执行查询。在数据库服务器处理决策支持应用程序启动的查询时，该操作可以极大地改善性能。

有关更多信息，请参阅并行数据库查询 *(PDQ)* 在第251页。

OPTCOMPIND 对查询计划的影响

OPTCOMPIND 设置会影响优化器为单个或多个表查询选择的存取计划。可以在不同种类查询的会话中更改 OPTCOMPIND 的值。

要在会话中更改 OPTCOMPIND 的值，请使用 SET ENVIRONMENT OPTCOMPIND 命令，而不是 OPTCOMPIND 配置参数。有关使用此命令的更多信息，请参阅设置会话中 *OPTCOMPIND* 的值 在第45页。

单表查询

对于单表扫描，OPTCOMPIND 设置为 0 或 1 且当前事务隔离级别为 Repeatable Read 时，优化器将考虑两种类型的存取计划。

如果：

- 某个索引是可用的，那么优化器将使用它访问表。
- 没有索引可用，那么优化器将考虑按物理顺序扫描表。

当尚未在数据库服务器配置中设置 OPTCOMPIND 时，其缺省值为 2。当 OPTCOMPIND 设置为 2 或 1 且当前隔离级别不是 Repeatable Read 时，优化器将选择成本最低的计划来访问表。

多表查询

对于连接计划，OPTCOMPIND 设置将影响用于表的特定排序对的存取计划。

如果您希望数据库服务器选择的连接方法与它在以前版本的数据库服务器中所选的完全相同，那么将 OPTCOMPIND 设置为 0。此选项确保了与以前版本的兼容性。

如果 OPTCOMPIND 设置为 0 或设置为 1 且当前事务隔离级别为 Repeatable Read，那么优化器将优先选择嵌套循环连接。

> 重要： 当 OPTCOMPIND 设置为 0，优化器将不会选择散列连接。

如果 OPTCOMPIND 设置为 2 或设置为 1 且事务隔离级别不是 Repeatable Read，那么优化器将从那些前面所列的计划中选择成本最低的查询计划并且不优先选择嵌套循环连接。

可用内存对查询计划的影响

SinoDB® 基于 DS_TOTAL_MEMORY 和 DS_MAX_QUERIES 配置参数的值限制并行查询可使用的内存量。如果可用于查询的内存量太少以至于无法执行散列连接，那么数据库服务器将使用嵌套循环连接来代替。

有关并行查询以及 DS_TOTAL_MEMORY 和 DS_MAX_QUERIES 参数的更多信息，请参阅并行数据库查询 *(PDQ)* 在第251页。

查询的时间成本

可以调整处理查询时数据库服务器执行操作的一些响应时间效果（不是全部）。

以下成本可以通过优化的查询结构和适当的索引降低：

- 排序时间
- 数据不匹配
- 定点 ALTER TABLE
- 索引查找

有关如何优化特定的查询的信息，请参阅提高个别查询性能 在第265页。

内存活动成本

数据库服务器只可以处理内存中的数据。它必须将行读入内存，以便依靠查询的过滤器评估那些行。服务器找到满足那些过滤器的行之后，它会通过将所选列组合起来，以在内存中准备一个输出行。

这些活动中的大部分执行速度都很快。根据计算机和其工作量的不同，数据库服务器可以每秒执行成百或甚至上千次的比较。因此，内存中的工作所花费的时间通常是执行时间的一小部分。

虽然一些内存中的活动（如排序）占用相当多的时间，但是从磁盘读取一行所花费的时间要比检查内存中已有的一行所用的时间长得多。

排序时间成本

排序既需要磁盘工作量也需要内存中的工作量。内存中的工作量取决于排序的列数、组合排序键的宽度和通过查询过滤器的行组合数。可以减少排序的成本。

可以使用以下公式来计算排序操作所需的内存中的工作量：

$$W_m = (c * N_{fr}) + (w * N_{fr}\log_2(N_{fr}))$$

W_m

是内存中的工作量。

c

是要排序的列数并代表从行中抽取列值以及将它们连接成一个排序键的成本。

w

与组合排序键的宽度成比例（以字节为单位）并代表复制或比较一个排序键的工作量。*w* 的数字值很大程度上取决于使用中的计算机硬件。

N_{fr}

是通过查询过滤器的行数。

如果要排序的数据量很大，那么排序可能需要将信息临时写入磁盘。您可以将磁盘写入安排在操作系统文件空间或者数据库服务器管理的数据库空间中进行。有关详细信息，请参阅为临时表和排序文件配置数据库空间 在第90页。

磁盘工作量取决于行所出现的磁盘页的数量、满足查询谓词条件的行数、可放置于排序的页上的行数和必须执行的合并操作的数量。使用以下公式计算排序操作所需要的磁盘工作量：

$$W_d = p + (N_{fr}/N_{rp}) * 2 * (m - 1))$$

W_d

是磁盘工作量。

p

是磁盘页数。

N_{fr}

是通过过滤器的行数。

N_{rp}

是可以放置于一页上的行数。

m

代表排序必须使用的合并级别数。

系数 m 取决于可存放在内存中的排序键数。如果没有过滤器，那么 N_{fr}/N_{rp} 等于 p。

当所有键都可以存放在内存中时，m=1 且磁盘工作量等于 p。换句话说，行在内存中被读取并排序。

对于中大型表，行将按满足内存的批进行排序，然后再合并各批。当 m=2 时，行将分批进行读取、排序并写入。然后各批将再次被读取并合并，这样将导致磁盘工作量与以下值成比例：

$$W_d = p + (2 * (N_{fr}/N_{rp}))$$

过滤器条件越具体，排序的行数越少。随着行数的增加和内存量的减少，磁盘工作量会增加。

要减少排序的成本，请使用以下方法：

- 使您的过滤器条件尽可能具体（可选择的）。
- 将计划列表限制到那些与您的问题相关的列。

行读取成本

当数据库服务器需要检查不在内存中的某一行时，它必须从磁盘读取该行。数据库服务器不只是读取一行；它要读取包含该行的整页。如果该行跨越多页，那么数据库服务器要读取所有这些页。

读取一页的实际成本易变且难以预计。实际成本是下表中所示因素的组合。

因素	因素的影响
缓冲	如果需要的页已经在页缓冲区中，那么读取的成本几乎为零。
争用	如果两个或多个应用程序需要存取磁盘硬件，那么 I/O 请求会延迟。
寻道时间	磁盘所做的最慢操作是寻道；也就是将存取臂移动到存放数据的磁道上。寻道时间取决于磁盘的速度和操作开始时磁盘臂的位置。寻道时间变化范围在 0 到接近 1 秒。
等待时间	直到页开头旋转至存取臂下时传输才开始。此等待时间或旋转延迟取决于磁盘的速度和操作开始时磁盘的位置。等待时间的变化范围可从零到几毫秒。

读取一页的时间成本可以从几微秒（对于已存在于缓冲区的页）到几毫秒（当争用为零且磁盘臂已就位时），到几百毫秒（当该页处于争用中且磁盘臂位于磁盘较远的柱面上时）。

顺序访问成本

当数据库服务器按物理顺序读取某个表的行时，磁盘成本最低。

当请求页的首行时，将把磁盘页读入缓冲区页中。读入该页之后，就不需要再次读取该页；对该页上后续行的请求将从缓冲区得到满足，直至该页上所有行都已处理完毕。当一页处理完毕时，包含下一组行的页必须读入。

当您对数据库空间使用未缓冲的设备且表已正确组织时，连续行的磁盘页将放置于磁盘上的连续位置。这种安排使存取臂在顺序读取时移动很少。此外，顺序读取页时等待时间成本通常较低。

相关链接

《*SinoDB* 管理员指南》：预读操作

非顺序访问成本

磁盘设备非顺序读取表页时磁盘访问时间要远远超过该磁盘设备顺序读取同一表的时间。

只要某个表是按随机顺序读取的，那么将需要额外的磁盘存取来按所需的顺序读取行。当读取某个表的行顺序与磁盘上的物理顺序不相关时，磁盘成本就会较高。因为页不是从磁盘顺序读取的，所以在读取每页之前就会发生寻道延迟和旋转延迟。

通常，当您使用索引确定行位置时会发生非顺序访问。尽管索引条目是有顺序的，但是不能保证相邻索引条目所对应的行一定处于相同的（或相邻）的数据页。许多情况下，必须进行单独的磁盘存取来获取通过索引所定位的每行所在的页。如果表比页缓冲区大，那么可能要先消除（从缓冲区移除并写回到磁盘）包含先前所读取的某一行所在的页，才能处理对该页上另一行的后续请求。该页可能必须再次读入。

根据表与索引的相对顺序，您有时可以检索包含几个所需行所在的页。磁盘上各行的物理顺序与索引中条目顺序的对应程度称为集群。高度集群的表是指磁盘上的物理顺序与索引严格对应的表。

索引查找成本

当数据库服务器通过索引找到一行时，将会产生额外的成本。索引存储在磁盘上，其各页必须同包含所想要的行的数据页一起读入内存中。

索引查找是从根页到叶子页进行的。由于根页使用非常频繁，因此几乎总是能在页缓冲区中找到。在缓冲区中找到叶子页的机会取决于索引的大小、查询的形式和列值重复的频率。如果每个值仅在索引中出现一次且查询是一个连接，那么要连接的每行将需要对索引进行一次非顺序查找，随后还要对表中相关行进行一次非顺序访问。

从索引中读取重复值

读取带有重复项的索引将产生超出顺序读取表的额外成本。

每个条目或具有同一值的每个条目集合必须位于索引中。然后，对于索引中的每个条目，必须对表进行随机访问来读取相关联的行。但是，如果每个相同的索引值有许多重复的行，且相关联的表高度集群，那么通过索引连接所添加的成本可能会很低。

搜索索引中的 NCHAR 或 NVARCHAR 列

使用 NCHAR 或 NVARCHAR 上的索引的查询将扫描整个索引，这将造成额外的时间成本。

仅 Global Language Support (GLS)

使用特定于语言环境的比较值对建立在 NCHAR 或 NVARCHAR 列上的索引进行排序。

在某些语言环境中，比较值不是基于代码集顺序的。索引建立使用特定于语言环境的比较值来将键值存储在索引中。因此，使用 NCHAR 或 NVARCHAR 上的索引的查询将扫描整个索引，这是因为数据库服务器是以代码集顺序搜索该索引的。

定点 ALTER TABLE 成本

在某些情况下，执行 ALTER TABLE 语句时，数据库服务器将使用定点变更算法来修改每一行。变更表操作之后，数据库服务器将使用最新的定义插入行。如果查询访问还未转换成新的表定义的行，那么您可能会注意到个别查询的性能稍微有些下降，这是因为数据库服务器在返回每一行之前，都会在内存中将其重新格式化。

有关发生定点变更时的情况和性能优势的更多信息，请参阅变更表定义 在第141页。

视图成本

复杂视图的运行会比预期更慢。

您可出于以下多种原因创建表的视图：

- 限制用户可以访问的数据
- 减少写一个复杂查询所花费的时间
- 隐藏用户需要写的查询的复杂性

但是，当视图定义的复杂性导致要创建一个临时表来处理查询时，针对视图的查询可能执行得比预期要慢。该临时表称为物化视图。例如，您可以使用 union 创建一个视图来合并几个 SELECT 语句的结果。

以下 SQL 语句示例创建一个包含 union 的视图：

```
CREATE VIEW view1 (col1, col2, col3, col4)
   AS
      SELECT a, b, c, d
         FROM tab1 WHERE
      UNION
      SELECT a2, b2, c2, d2
         FROM tab2 WHERE
...
      UNION
      SELECT an, bn, cn, dn
         FROM tabn WHERE
;
```

当您创建一个包含复杂的 SELECT 语句的视图时，最终用户无需处理复杂性。最终用户只可以写一个简单的查询，如以下示例所示：

```
SELECT a, b, c, d
   FROM view1
      WHERE a < 10;
```

但是，针对 view1 的查询执行得可能比预期要慢，这是因为数据库服务器在执行该查询之前要为视图创建一个分段临时表。

如果您使用 ANSI join 中的一个视图，那么查询的执行速度可能比预期要慢。视图定义的复杂性可能将导致要创建临时表。

为了确定您是否具有一个必须建立临时表来处理视图的查询，需执行 SET EXPLAIN 语句。如果在 SET EXPLAIN 输出文件中发现 Temp Table For View，那么查询需要一个临时表来处理该视图。

小表成本

如果一个表占用页数很少以至于它可完全保存在页缓冲区中，那么该表称为小表。对小表的操作一般要比对大表的操作快。

例如，在 stores_demo 数据库中，将缩写与州名联系起来的 state 表的总大小小于 1000 字节；它只占用两页。该表以很低的成本包含于任何查询中。不管如何使用该表，第一次需要它时，最多花费两次磁盘访问从磁盘检索该表。

数据不匹配成本

如果在某一条件下使用的列的数据类型与 CREATE TABLE 语句中列的定义不同，那么 SQL 语句可能会遇到额外的开销。

例如，以下查询包含将一列与不同于表定义的数据类型值进行比较的条件：

```
CREATE TABLE table1 (a integer, );
SELECT * FROM table1
```

```
   WHERE a = '123';
```

在执行将 123 转换成整数之前，数据库服务器会重写该查询。SET EXPLAIN 输出会按其调整的格式显示查询。该数据转换没有显著的开销。

当查询将一个字符列与一个非字符值进行比较，并且数字长度不等于字符列的长度时，数据不匹配导致的额外成本是最严重的。例如，以下查询在 WHERE 子句中包含一个条件，由于缺少引号导致一个字符列等于一个整数值：

```
CREATE TABLE table2 (char_col char(3), );
SELECT * FROM table2
   WHERE char_col = 1;
```

该查询查找对 char_col 有效的下列所有值：

```
' 1'
'001'
'1'
```

这些值不必在索引键中集群在一起。因此，索引不提供获得数据的快速和正确的方法。SET EXPLAIN 输出显示此种情况的顺序扫描。

> 警告： 当 SQL 语句将一个字符列与一个长度不等于该字符列的非字符值进行比较时，数据库服务器将不使用索引。

GLS 功能成本

为某些数据集排序或建立索引会降低性能。

有关由对某些数据集建立索引而引发的性能降级的信息，请参阅搜索索引中的 *NCHAR* 或 *NVARCHAR* 列 在第229页。

如果您不需要非 ASCII 整理顺序，应该尽可能对字符列使用 CHAR 和 VARCHAR 数据类型。由于 CHAR 和 VARCHAR 数据需要简单的基于值的比较，因此对这些列进行排序和建立索引的成本要比用于非 ASCII 数据类型（例如 NCHAR 或 NVARCHAR）的成本低。

有关其他字符数据类型的更多信息，请参阅《*SinoDB® GLS* 用户指南》。

网络访问成本

跨网络移动数据会造成除直接磁盘存取以外的延迟。

当应用程序通过网络向另一台计算机上的数据库服务器发送一个查询或更新请求时，就会发生网络延迟。虽然数据库服务器是在远程主机上执行查询，但是该数据库服务器要通过网络将输出返回至应用程序。

通过网络发送的数据由命令消息和缓冲区大小的行数据块组成。虽然根据网络和计算机的不同，详细信息可能有所不同，但是数据库服务器网络活动遵循一个简单的模型，即：一台计算机（客户端）向另一台计算机（服务器）发送请求。服务器使用表的数据块作出响应。

只要通过网络进行数据交换，在以下情况下延迟就不可避免：

- 网络繁忙时，客户端必须等待它的传输时机。这样的延迟通常小于一毫秒。但是，遇到网络负载繁重时，这些延迟可能按指数级增加到几个十分之一秒甚至更多。
- 当服务器正在处理多个客户端的请求时，这些请求也许要排队等候一段时间，等候时间可能从几毫秒到几秒不等。
- 当服务器对请求进行操作时，将产生前面几节所述的磁盘存取和内存中的操作的时间成本。

响应的传输也受到网络延迟的影响。

网络访问时间变化极大。在最理想的情况下，当网络和服务器都不繁忙时，传输和排队延迟并不明显，服务器发送一行的速度与本地数据库服务器可能达到的速度几乎相同。而且，当客户端请求第二行时，该页可能已在服务器的页缓冲区中。

不幸的是，随着网络负载的增加，所有这些因素都会同时恶化。传输延迟在两个方向上都会增加，这将增加服务器上的队列长度。请求之间的延迟会导致页保留在响应端的页缓冲区中的可能性降低。因此，网络访问成本可能突然急剧变化。

如果您在分布式查询中使用 SELECT FIRST *n* 子句，那么您将仍然只能看到所请求的数据量。但是，本地数据库服务器不会向远程站点发送 SELECT FIRST *n* 子句。因此，远程站点可能返回更多的数据。

数据库服务器使用的优化器假定通过网络访问一行比访问数据库服务器中的一行的时间长。此估算值包含从磁盘检索该行和通过网络传输该行的成本。

有关可能提高网络性能的操作的信息，请参阅以下部分：

- 分布式查询的优化器估算 在第287页
- 多路复用连接和 *CPU* 利用率 在第54页
- 网络缓冲池 在第49页

SQL 在 SPL 例程中时的优化

如果 SPL 例程包含 SQL 语句，那么数据库服务器会在 SPL 例程内优化并执行 SQL。

本部分的主题包含有关数据库服务器如何以及何时优化和执行这些例程的信息。

SQL 优化

如果 SPL 例程包含 SQL 语句，那么在某一点上，查询优化器将对 SPL 例程中 SQL 的可能查询计划进行评估并选择成本最低的查询计划。数据库服务器将为每个 SQL 语句选择的查询计划置于 SPL 例程的执行计划中。

当您用 CREATE PROCEDURE 语句创建一个 SPL 例程时，数据库服务器会在此时试图优化 SPL 例程中的 SQL 语句。如果在编译时间无法检查表（因为它们不存在或不可用），那么创建将不会失败。在这种情况下，数据库服务器将在 SPL 例程第一次执行时对 SQL 语句进行优化。

数据库服务器将优化的执行计划存储在 sysprocplan 系统目录表中，以供其他进程使用。另外，数据库服务器还将有关 SPL 例程的信息（如过程名称和所有者）存储在 sysprocedures 系统目录表中，并将 SPL 例程的 ASCII 版本存储在 sysprocbody 系统目录表中。

图 *60: 系统目录表中存储的 SPL 信息* 在第232页总结了数据库服务器在编译过程中存储在系统目录表中的信息。

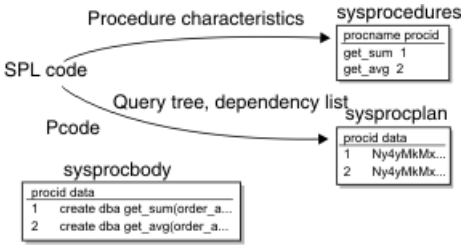

图 60: 系统目录表中存储的 SPL 信息

显示执行计划

当执行 SPL 例程时，它已经优化。可以显示 SPL 例程中包含的每个 SQL 语句的查询计划。

要显示查询计划，请在执行以下某个 SQL 语句（它们总是试图优化 SPL 例程）之前执行 SET EXPLAIN ON 语句：

- CREATE PROCEDURE
- UPDATE STATISTICS FOR PROCEDURE

例如，使用以下语句显示 SPL 例程的查询计划：

```
SET EXPLAIN ON;
```

```
UPDATE STATISTICS FOR PROCEDURE procname;
```

自动重新优化
在某些情境下，下次执行 SPL 时，数据库服务器将重新优化 SQL 语句。

如果禁用 AUTO_REPREPARE 配置参数或 IFX_AUTO_REPREPARE 会话环境变量，那么在由 SPL 例程间接引用或预编译对象引用的表模式被修复之后执行该预编译对象或 SPL 例程时，将导致以下错误：

```
-710  Table <table-name> has been dropped, altered, or renamed.
```

数据库服务器使用依赖性列表来跟踪在下次 SPL 例程执行时将导致的重新优化的变化。

在以下其中一种情形发生后，下次执行 SPL 例程时，数据库服务器将重新优化 SQL 语句：

- 执行可能变更查询计划的任何数据定义语言 (DDL) 语句（如 ALTER TABLE、DROP INDEX 和 CREATE INDEX）
- 变更使用参考约束（两个方向均可）链接到另一个表的表
- 对查询中所涉及的任一个表执行 UPDATE STATISTICS FOR TABLE

  UPDATE STATISTICS FOR TABLE 语句更改 systables 中指定表的版本号。
- 使用 RENAME 语句重命名列、数据库或索引

无论什么时候对 SPL 例程重新优化，数据库服务器都将使用重新优化的执行计划更新 sysprocplan 系统目录表。

重新优化 SPL 例程
可以执行一个重新优化 SPL 例程的 SQL 语句以防止自动重新优化。

如果您不希望在自动重新优化 在第233页所列出的其中一种情形发生后第一次执行 SPL 例程时产生自动重新优化的开销，那么请在该情形发生后立即执行带 FOR PROCEDURE 子句的 UPDATE STATISTICS 语句。这样，SPL 例程就可在任何一方用户执行它之前被重新优化。

为了避免对所有 SPL 例程不必要的重新优化，请确保您在 FOR PROCEDURE 子句中指定特定的过程名称。

```
UPDATE STATISTICS FOR PROCEDURE myroutine;
```

有关执行 UPDATE STATISTICS 的指南，请参阅未自动生成统计信息时更新统计信息 在第274页。

SPL 例程中 SQL 的优化级别
SPL 例程中设置的当前优化级别将影响 SPL 例程的优化方式。

SET OPTIMIZATION HIGH 语句所调用的算法是一种先进的、基于成本的策略，它会检查所有合理的查询计划并选择总体来讲最佳的计划。对于大型连接，此算法产生的开销可能比预计要多。极端情况下，您可能会耗尽内存。

SET OPTIMIZATION LOW 语句调用的备用算法在早期阶段就消除了不可能的连接策略，这样可以减少优化过程中的时间和资源消耗。但是，当您指定较低级别的优化时，由于在算法早期阶段最佳策略可能已被消除，因此可能无法选中最佳策略。

对于保持不变或仅仅稍微变化的 SPL 例程和包含复杂 SELECT 语句的 SPL 例程，您在创建该例程时可能希望将 SET OPTIMIZATION 语句设置为 HIGH。该优化级别将存储 SPL 例程的最佳查询计划。然后将优化设置为 LOW，再执行 SPL 例程。SPL 例程将使用最佳的查询计划，并且如果重新优化发生，那么将以更节约成本的方式执行计划。

SPL 例程的执行

数据库服务器使用 EXECUTE PROCEDURE 语句、CALL 语句或在 SQL 语句中执行 SPL 例程时，服务器将执行若干活动。

数据库服务器会执行以下活动：

- 从系统目录表中读取解释器代码并将该代码由压缩格式转换成可执行格式。如果 SPL 例程在 UDR 高速缓存中，那么数据库服务器将从高速缓存中检索而不用执行该转换步骤。

- 执行遇到的任何 SPL 语句。
- 当数据库服务器遇到一个 SQL 语句时，将从数据库中检索查询计划并执行该语句。如果还没有创建查询计划，那么数据库服务器将在执行之前优化 SQL 语句。
- 当数据库服务器结束运行 SPL 例程时或遇到 RETURN 语句时，数据库服务器将把所有结果返回给客户端应用程序。除非 RETURN 语句有一个 WITH RESUME 子句，否则 SPL 例程执行将完成。

存储在 UDR 高速缓存中的 SPL 例程可执行格式

当一个用户第一次执行 SPL 例程时，数据库服务器将可执行格式和任何查询计划存储在共享内存虚拟部分的 UDR 高速缓存中。

当另一用户执行 SPL 例程时，数据库服务器将先检查 UDR 高速缓存。当数据库服务器可从 UDR 高速缓存执行 SPL 例程时，SPL 执行性能会提高。UDR 高速缓存也会存储 UDR、用户定义的聚合和扩展的数据类型定义。

相关链接

配置和监视内存高速缓存 在第69页

调整 UDR 高速缓存

UDR 高速缓存中 SPL 例程、UDR 和其他用户定义的缺省数量为 127。可以使用 PC_POOLSIZE 配置参数更改条目数。

数据库服务器使用散列算法将 SPL 例程存储并定位在 UDR 高速缓存中。您可以使用 PC_HASHSIZE 配置参数修改 UDR 高速缓存中存储区的数量。例如：如果 PC_POOLSIZE 配置参数的值为 100，而 PC_HASHSIZE 配置参数的值为 10，那么每个存储区可以拥有数量多达 10 个的 SPL 例程和 UDR。

存储区太多将使数据库服务器在存储区装满时移出高速缓存的 SPL 例程。存储区太少将会增加一个存储区的 SPL 例程数，并且数据库服务器必须搜索一个存储区中所有的 SPL 例程来确定所需的 SPL 例程是否存在。

一个存储区中条目数达到 75% 时，数据库服务器就会将最近最少使用的 SPL 例程从该存储区中（从而也从 UDR 高速缓存中）移除，直至该存储区中 SPL 例程数量是该存储区中 SPL 例程最大数量的 50% 为止。

可以通过执行 onstat -g prc 命令来监视 UDR 高速缓存。如果 hits 字段中的数字未均匀分布在各个存储区中，请增加 PC_HASHSIZE 配置参数的值。可以调整存储区数，以使每个存储区具有最少的高命中条目数。

> 重要： PC_POOLSIZE 和 PC_HASHSIZE 也会控制该数据库服务器的其他内存高速缓存（不包括缓冲池、SQL 语句高速缓存、数据分布高速缓存和数据字典高速缓存）。当修改 SQL 例程的散列存储区的大小和数量时，也将会修改其他高速缓存（如聚合高速缓存、oplcass 和类型名高速缓存）的散列存储区的大小和数量。

相关链接

《SinoDB 管理员参考》: onstat -g prc 命令: 显示使用 UDR 或 SPL 例程的会话

《SinoDB 管理员参考》: PC_POOLSIZE 配置参数

《SinoDB 管理员参考》: PC_HASHSIZE 配置参数

触发器执行

触发器是一个数据库对象，当指定数据操作语言操作（触发事件）发生时，该对象可自动执行一个或多个 SQL 语句（触发操作）。您可以在一个表上定义一个或多个触发器以便在 SELECT、INSERT、UPDATE 或 DELETE 触发事件发生之后进行操作。

也可以定义视图上的 INSTEAD OF 触发器。当触发 INSERT、UPDATE 或 DELETE 语句尝试修改视图时，这些触发器将指定底层的表中作为触发操作执行的 SQL 语句。这些触发器称为 INSTEAD OF 触发器，因为仅执行触发 SQL 操作；不执行触发事件。有关使用触发器的更多信息，请参阅《*SinoDB® SQL* 指南: 教程》以及《*SinoDB® SQL* 指南: 语法》中有关 CREATE TRIGGER 语句的信息。

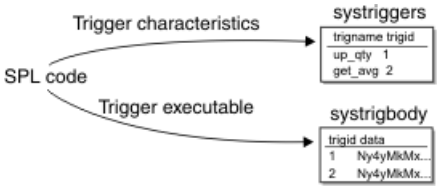

图 61: 系统目录表中存储的触发器信息

当您使用 CREATE TRIGGER 语句注册新的触发器时，数据库服务器将执行以下操作：

- 将触发器信息存储在 systriggers 系统目录表中。
- 将触发器执行的语句文本存储在 systrigbody 系统目录表中。

sysprocedures 系统目录表标识只可以作为触发器操作调用的触发器例程。

sysmaster 数据库的内存驻留表指示表或视图是否具有触发器。

只要发出 SELECT、INSERT、UPDATE 或 DELETE 语句，数据库服务器就会查看语句是否为可以激活 DML 语句操作的表或列（或者视图）的触发器的触发事件。如果语句需要激活触发器，那么数据库服务器将在触发事件之前、期间或之后从 systrigbody 表检索触发操作的语句文本并执行触发 DML 语句或 SPL 例程。对于视图上的 INSTEAD OF 触发器，数据库服务器将执行触发操作而不是触发事件。

触发器的性能影响

在许多情境中，由于从客户端到数据库服务器所传递的消息数量减少，触发器可以稍微提高性能。

例如，如果触发器触发了 5 个 SQL 语句，那么客户端将至少保存 10 条在客户端与数据库服务器（一个用于发送 SQL 语句而另一个用于在数据库服务器执行 SQL 语句之后应答）之间传递的消息。当触发器执行更多的 SQL 语句且网络速度相对较慢时，触发器能最大限度地提高性能。

当数据库服务器执行一个 SQL 语句时，必须执行以下操作：

- 确定是否必须触发触发器
- 从 systriggers 和 systrigbody 中检索触发器

这些操作只对性能造成轻微的影响，可以通过客户端和服务器之间所传递消息数的减少来抵销这些影响。

但是，对 SELECT 语句执行的触发器会有其他的性能影响。以下各部分说明了这些影响。

表层次结构中表的 SELECT 触发器

当数据库服务器执行一个包含表层次结构中涉及的表的 SELECT 语句，且该 SELECT 语句触发一个 SELECT 触发器时，如果调用此触发器的 SELECT 语句涉及连接、排序或物化视图，那么性能可能会更低。

在这种情况下，数据库服务器并不知道表层次结构中哪些列会受到影响，因此它可以通过不同方法执行查询。以下行为可能会发生：

- 在表层次结构中涉及的表中，仅键索引扫描被禁用。
- 如果数据库服务器需要对从表层次结构中涉及的表中选择的数据进行排序，那么它要将 SELECT 列表中的所有列（而不只是排序列）都复制到临时表中。
- 如果数据库服务器使用表层次结构中包含的表来建立一个散列表，以用于与另一表的散列连接，那么数据库服务器将绕过原先的设计，这意味着它将使用表的所有列（而不只是连接中的列）来创建该散列表。
- 如果该 SELECT 语句包含一个物化视图（这意味着在视图中必须为列构建一个临时表），而且该视图包含表层次结构中所涉及的表的列，那么该表的所有列（而不只是实际包含在该视图中的列）必须包含在临时表中。

SELECT 触发器和行缓冲

如果不对触发 SELECT 触发器的 SELECT 语句执行缓冲操作，那么与未触发 SELECT 触发器的同一 SELECT 语句相比，性能可能稍微有所下降。

在那些表没有触发 SELECT 触发器的 SELECT 语句中，即使客户端应用程序使用 FETCH 语句只请求一行，数据库服务器仍会将多行发回客户端并将这些行存储在缓冲区中。但是，对于包含一个或多个触发

SELECT 触发器的表的 SELECT 语句，数据库服务器只将请求的行发回客户端而不是整个缓冲区。直到触发器操作发生时，数据库服务器才能将其他行返回客户端。

第

11

章

优化器指令

优化器指令是指示查询优化器如何执行查询的注释。您可以使用优化器指令来提高查询性能。

优化器指令是什么

优化器指令是规范的格式化注释，它们向查询优化器提供如何执行查询的有关信息。

您可使用两种优化器指令：

- 以指令形式嵌入查询中的优化器指令（有关更多信息，请参阅嵌入查询中的优化器指令 在第237页。）
- 在不想更改查询中的 SQL 语句时，您创建并保存的外部优化器指令可用作对问题的临时变通的解决方案。（有关更多信息，请参阅外部优化器指令 在第237页。）

嵌入查询中的优化器指令

嵌入查询中的优化器指令是 SELECT 语句中的注释，它们向查询优化器提供有关如何执行查询的信息。您也可以将指令放在 UPDATE 和 DELETE 语句中，指示优化器如何访问数据。

优化器指令可以是显式指令（例如，“use this index”或“access this table first”），它们也可以消除可能的查询计划（例如，“do not read this table sequentially”或“do not perform a nested-loop join”）。

外部优化器指令

外部优化器指令是管理员可在 sysdirectives 目录表中创建和存储的优化器指令。然后，管理员可使用 ONCONFIG 变量使这些指令可用。

客户端用户也可指定环境变量，并且当他们不想在 SQL 语句中插入注释的情况下，可以选择在查询中使用这些优化器指令。

当短期内无法为问题的解决方案重新编写查询（例如当查询开始表现较差时）时，外部指令很有用。对于问题的长期解决方案，通过更改 SQL 语句来重新编写查询是首选方法。

外部指令只是偶尔使用。存储在 sysdirectives 目录中的指令数不应超过 50。典型的企业只需要 0 到 9 个指令。

使用优化器指令的原因

在大多数情况下，优化器选择最快的查询计划。由于查询的复杂性或者查询没有足够的数据特性相关信息而导致优化器没有选择最佳的查询计划来执行查询时，那么可以使用优化器指令。查询计划不佳将导致性能不佳。

在您决定何时使用优化器指令之前，您应该了解什么是好的查询计划。

优化器是根据不同的表访问路径、连接顺序和连接计划的成本来创建查询计划。

一些查询计划会指示：

- 当数据库服务器必须读取表的大部分数据时，不要使用索引。例如，以下查询可能读取 customer 表的大部分数据：

```
SELECT * FROM customer WHERE STATE <> "ALASKA";
```

  假定客户在所有的 50 个州中均匀分布，那么您可能估算数据库服务器必须读取表 98% 的数据。当数据库服务器必须读取大多数行时，对表进行顺序读取要比遍历索引（并随后遍历数据页）效率更高。
- 当您在索引之间选择访问表时，那么使用可以排除大多数行的索引。例如，考虑以下查询：

```
SELECT * FROM customer
WHERE state = "NEW YORK" AND order_date = "01/20/97"
```

  假定有 200,000 个客户住在纽约并且每天只有 1000 个客户订购，那么优化器最有可能选择 order_date 索引而不选择 state 索引来执行查询。
- 将小型表或带有限制性过滤器的表及早放在查询计划中。例如，请考虑以下查询：

```
SELECT * FROM customer, orders
   WHERE customer.customer_num = orders.customer_num
      AND
   customer.state = "NEVADA";
```

  在此示例中，如果您先读取 customer 表，那么可以通过条件 state = "NEVADA" 使过滤器排除大多数行。

  通过排除 customer 表中的行，数据库服务器将不必读取 orders 表（该表可能要比 customer 表大得多）中的很多行。
- 当连接过滤器中的两列都没有索引时，请选择散列连接。

  在上述示例中，如果 customer.customer_num 和 orders.customer_num 没有索引，那么散列连接可能是最佳的连接计划。
- 在下列情况下选择嵌套循环连接：
  - 数据库服务器使用其他表过滤器后，从外表中检索出的行数很少，并且内表具有一个可用于执行连接的索引。
  - 最外围表的索引可按 ORDER BY 子句的顺序返回行，而无需排序。

有关查询计划的信息，请参阅查询计划 在第210页。有关指令的更多信息，请参阅

- 使用指令的准备工作 在第238页
- 使用指令的准则 在第239页
- *SQL* 语句中支持的优化器指令的类型 在第239页

使用指令的准备工作

在大多数情况下，优化器会选择最快的查询计划。但是，您可以执行步骤来协助优化器并为使用指令做准备。

要为使用指令做准备，请确保执行以下任务：

- 执行 UPDATE STATISTICS。

  没有精确的统计信息，优化器将无法选择适当的查询计划。当表中数据明显更改（添加、更新和删除很多新行）时，执行 UPDATE STATISTICS。有关更多信息，请参阅更新行数的统计信息 在第275页。
- 创建分布。

  您试图提高缓慢查询时应该尝试的首要工作之一是在查询中所涉及的列上创建分布。分布将有关表中的数据特性的最精确信息提高给优化器。执行查询过滤器中涉及的列上的 UPDATE STATISTICS HIGH 可查看性能是否提高。有关更多信息，请参阅创建数据分布 在第275页。

在某些情况下，由于查询的复杂性或（即便有分布）有关数据特性的信息不足，查询优化器并不会选择最佳的查询规划。在这些情况下，您可以通过使用指令尝试提高某个特殊查询的性能。

使用指令的准则

指令的准则包括频繁分析查询的效率和使用负指令。

考虑以下准则：

- 经常检查某个特殊指令的有效性以确保其继续有效地运行。假设在生产程序中有一个具有多个指令的查询，该查询强制实施一个最优查询计划。一段时间后，用户添加、更新或删除大量的行，这将很大程度地改变数据的特性以至于曾经最优的查询计划不再有效。该示例说明您必须如何小心使用指令。
- 在任何可能的情况下，使用负指令（例如 AVOID_NL、AVOID_FULL 等）。当您排除了导致性能降低的行为时，您就可让优化器使用次最佳选择，而不必试图强制指定一个可能并非最优的途径。

SQL 语句中支持的优化器指令的类型

在查询中嵌入 SQL 语句中的指令。这些指令包括访问方法指令、连接顺序指令、连接计划指令和优化目标指令。

将指令作为注释包含在 SQL 语句中，该注释紧跟在 SELECT、UPDATE 或 DELETE 关键字的后面。指令中的第一个字符常常是加号 (+)。在以下查询中，ORDERED 指令指定表的连接顺序应该与这些表在 FROM 子句中所列的顺序相同。AVOID_FULL 指令指定优化器应该丢弃任何包含对所列表 (employee) 的全面扫描的计划。

```
SELECT --+ORDERED, AVOID_FULL(e) * FROM employee e, department d
> 50000;
```

有关指令的完整语法描述，请参阅《*SinoDB® SQL* 指南*:* 语法》。

要影响优化器所做的查询计划的选择，您可以变更查询的以下几个方面：

- 访问方法
- 连接顺序
- 连接方法
- 优化目标
- 星型连接指令

您也可以使用 EXPLAIN 指令而不是 SET EXPLAIN 语句来显示查询计划。以下几节将对这些方面进行详细描述。

访问方法指令

数据库服务器使用访问方法来访问表。服务器可以通过全面表扫描顺序读取表，或使用表的任一索引。访问方法指令会影响访问方法的执行。

下表列出影响访问方法的指令：

访问方法指令	描述
INDEX	指示优化器使用指定的索引访问表。如果该指令列出的索引不至一个，那么优化器将选择成本最低的索引。
AVOID_INDEX	指示优化器不使用列出的任何索引。您可以将该指令与 AVOID_FULL 指令一起使用。
INDEX_SJ	使用指定的索引或在索引列表中选择成本最低的索引来强制自连接路径，即使数据分布统计信息不可用于索引的前导索引键列。

访问方法指令	描述
	有关索引自连接路径的信息，请参阅包含索引自连接路径的查询计划 在第215页。
AVOID_INDEX_SJ	指示优化器不对指定的索引使用索引自连接路径。
FULL	指示优化器执行全面表扫描。
AVOID_FULL	指示优化器不要对列出的表执行全面表扫描。您可以将该指令与 AVOID_INDEX 指令一起使用。
INDEX_ALL or MULTI_INDEX	通过对多索引扫描使用指定的索引来访问表。 INDEX_ALL 和 MULTI_INDEX 关键字是同义词。
AVOID_MULTI_INDEX	指示优化器不要考虑对指定的表使用多索引扫描路径。

在某些情况下，强制指定一种访问方法可以更改优化器所选择的连接方法。例如，如果您使用 AVOID_INDEX 指令排除某个索引的使用，那么优化器可能选择散列连接而不是嵌套循环连接。

只有满足以下所有条件，优化器才会考虑索引自连接路径：

- 索引不具有功能键、用户定义的类型、内置不透明类型或非 B 型树索引
- 数据分布统计信息可用于考虑中的索引键列
- 表中的行数至少相当于所有可能前导键列值的唯一组合数的 10 倍。

如果满足所有这些条件，那么优化器将估算索引自连接路径的成本，并将该成本与备选访问方法的成本进行比较。然后，优化器将为表选择最佳访问方法。有关访问方法指令及其一些用法示例的更多信息，请参阅《*SinoDB® SQL* 指南*:* 语法》。

连接顺序指令

连接顺序指令 ORDERED 会指示优化器按照 SELECT 语句列出表的顺序连接这些表。

连接顺序对连接计划的影响

通过指定连接顺序，影响的可能不只是表的连接方法。

例如，请考虑以下查询：

```
SELECT --+ORDERED, AVOID_FULL(e)
* FROM employee e, department d
WHERE e.dept_no = d.dept_no AND e.salary > 5000
```

在此示例中，优化器选择用散列连接来连接表。但是，如果您对顺序进行安排以便第二个表为 employee（且必须被索引访问），那么散列连接就不可行了。

```
SELECT --+ORDERED, AVOID_FULL(e)
* FROM department d, employee e
WHERE e.dept_no = d.dept_no AND e.salary > 5000;
```

在这种情况下，优化器选择嵌套循环连接。

使用视图时的连接顺序

视图内或包含视图的查询中有 ORDERED 指令会影响连接顺序。

使用视图时，有两种情况可能会影响连接顺序：

- ORDERED 指令位于视图内。

某个视图内的 ORDERED 指令仅影响该视图内的表的连接顺序。该视图内的表必须连续连接。请考虑以下视图和查询：

```
CREATE VIEW emp_job_view as
   SELECT {+ORDERED}
   emp.job_num, job.job_name
   FROM emp, job
   WHERE emp.job_num = job.job_num;

SELECT * from dept, emp_job_view,  project
   WHERE dept.dept_no = project.dept_num
   AND emp_job_view.job_num = project.job_num;
```

ORDERED 指令指定 emp 表出现在 job 表之前。该指令不影响 dept 和 project 表的顺序。因而，所有可能的连接顺序如下所示：

- emp、job、dept、project
- emp、job、project、dept
- project、emp、job、dept
- dept、emp、job、project
- dept、project、emp、job
- project、dept、emp、job

- ORDERED 指令位于包含视图的查询中。

  如果 ORDERED 指令出现在包含视图的查询中，那么在该查询中表的连接顺序将与在 SELECT 语句中列出的表的连接顺序相同。视图中的表的连接顺序与其在视图中的排列顺序相同。

  在以下的查询中，连接顺序是 dept、project、emp、job：

```
CREATE VIEW emp_job_view AS
   SELECT
   emp.job_num, job.job_name
   FROM emp, job
   WHERE emp.job_num = job.job_num;
SELECT {+ORDERED}
   * FROM dept, project, emp_job_view
   WHERE dept.dept_no = project.dept_num
   AND emp_job_view.job_num = project.job_num;
```

  当视图无法并入查询时，是该规则的一个例外，如以下示例所示：

```
CREATE VIEW emp_job_view2 AS
   SELECT DISTINCT
   emp.job_num, job.job_name
   FROM emp,job
   WHERE emp.job_num = job.job_num;
```

  在此示例中，数据库服务器执行查询并将结果放入临时表中。该查询中表的顺序是 dept、project、temp_table。

连接方法指令

连接方法指令会影响数据库服务器在查询中连接两个表的方式。

以下指令会影响两个表之间的连接方法：

- USE_NL

  将所列出的表作为嵌套循环连接中的内表。

- USE_HASH

使用散列连接访问列出的表。您也可以选择表是用来创建散列表的，还是用来探测散列表的。

- AVOID_NL

  不要将所列出的表作为嵌套循环连接中的内表。用该指令所列出的表仍然可作为外表加入嵌套循环连接中。

- AVOID_HASH

  不要用散列连接来访问所列出的表。您可以有选择地采用散列连接，但是要对表进行限制，该表不允许为被探测的表或建立散列表所用的表。

您可以在 USE_HASH 或 AVOID_HASH 优化器指令中的表名称之后指定关键字 /BUILD：

- 使用 USE_HASH 指令时，/BUILD 关键字会指示优化器使用指定的表来构建散列表。
- 使用 AVOID_HASH 时，/BUILD 关键字会指示优化器避免使用指定的表来构建散列表。

您可以在 USE_HASH 或 AVOID_HASH 优化器指令中的表名称之后指定关键字 /PROBE：

- 使用 USE_HASH 指令时，/PROBE 关键字会指示优化器使用指定的表来探测散列表。
- 使用 AVOID_HASH 指令时，/PROBE 关键字会指示优化器避免使用指定的表来探测散列表。

优化目标指令

在某些查询中，可能希望仅查找查询结果中的前几行。或者，您可能了解到所有行都必须访问，并从查询中返回。可以使用优化目标指令来查找满足查询的前几行或满足查询的所有行。

例如，假设您想要仅查找查询结果中的前面几行，则使用 SinoDB® ESQL/C 程序打开查询的游标并执行 FETCH 以仅查找前面几行。

使用优化目标指令来对以下情况中的任一种情况的查询进行优化：

- FIRST_ROWS

  选择一个计划，该计划仅对满足查询的前几行的查找过程进行了优化。

- ALL_ROWS

  选择一个计划，该计划对满足查询的所有行（缺省行为）的查找过程进行了优化。

如果使用 FIRST_ROWS 指令，那么优化器可能将提前丢弃包含耗时活动的查询计划。例如，散列连接可能会花费大量时间来创建散列表。如果只有几行必须返回，那么优化器可能改为选择嵌套循环连接。

在以下示例中，假设数据库在 employee.dept_no 上有一个索引，而在 department.dept_no 上没有索引。没有指令，优化器将选择散列连接。

```
SELECT *
FROM employee, department
WHERE employee.dept_no = department.dept_no
```

但是，使用了 FIRST_ROWS 指令之后，由于创建散列表所需的初始开销较高，因而优化器选择嵌套循环连接。

```
SELECT {+first_rows} *
FROM employee, department
WHERE employee.dept_no = department.dept_no
```

星型连接指令

星型连接指令可以指定查询优化器如何连接两个或多个表，其中一个或多个维表通过外键与一个或多个事实表建立依赖关系。

以下指令可能会影响逻辑上参与星型模式或雪花模式的表的连接计划：

- FACT

  优化器会考虑选择在星型连接执行计划中指定的表是事实表的查询计划。

- AVOID_FACT

  优化器不会考虑选择将指定的表（或表列表中的任何表）视为事实表处理的星型连接执行计划。
- STAR_JOIN

  优化器会偏好选择星型连接执行计划（如果可用）。
- AVOID_STAR_JOIN

  优化器会选择不是星型连接的查询执行计划。

除非启用了并行数据库查询 (PDQ) 功能，否则这些星型连接指令没有影响。

相关链接

《SinoDB SQL 指南: 语法》: 星型连接指令

EXPLAIN 指令

可以使用 EXPLAIN 指令来显示优化器选择的查询计划，且可以指定显示查询计划但不执行查询。

可以使用这些指令：

- EXPLAIN

  显示优化器所选择的查询计划。
- EXPLAIN AVOID_EXECUTE

  显示优化器所选择的查询计划，但是不执行查询。

希望只显示一个 SQL 语句的查询计划时，请使用这些 EXPLAIN 指令而不是 SET EXPLAIN ON 或 SET EXPLAIN ON AVOID_EXECUTE 语句。

当在指令或 SET EXPLAIN 语句中使用 AVOID_EXECUTE 时，查询不会执行但会显示以下消息：

```
No rows returned.
```

图 62: EXPLAIN AVOID_EXECUTE 指令的结果 在第243页显示使用 EXPLAIN AVOID_EXECUTE 指令的查询的输出示例。

```
QUERY:
------
select --+ explain avoid_execute
  l.customer_num, l.lname, l.company,
  l.phone, r.call_dtime, r.call_descr
from customer l, cust_calls r
where l.customer_num = r.customer_num

DIRECTIVES FOLLOWED:
EXPLAIN
AVOID_EXECUTE
DIRECTIVES NOT FOLLOWED:

Estimated Cost: 7
Estimated # of Rows Returned: 7

  1) informix.r: SEQUENTIAL SCAN

  2) informix.l: INDEX PATH

    (1) Index Keys: customer_num   (Serial, fragments: ALL)
        Lower Index Filter: informix.l.customer_num = informix.r.customer_num
NESTED LOOP JOIN
```

图 62: EXPLAIN AVOID_EXECUTE 指令的结果

下表对 图 62: EXPLAIN AVOID_EXECUTE 指令的结果 在第243页中描述所选查询计划的相关输出行进行了描述。

图 62: EXPLAIN AVOID_EXECUTE 指令的结果 在第243页中的输出行	所选的查询计划描述
DIRECTIVES FOLLOWED: EXPLAIN AVOID_EXECUTE	使用指令 EXPLAIN 和 AVOID_EXECUTE 来显示查询计划，而不执行查询。
Estimated # of Rows Returned: 7	预估该查询返回 7 行。
Estimated Cost: 7	该估算成本值为 7，优化器使用该值来比较不同查询计划并选择成本最低的查询计划。
1) informix.r: SEQUENTIAL SCAN	将 cust_calls r 表用作外表并对其进行扫描以获取每一行。
2) informix.l: INDEX PATH	对于外表中的每一行，请使用索引获取内表 customer l 中的匹配行。
(1) Index Keys: customer_num (Serial, fragments: ALL)	使用 customer_num 列的索引，对其进行顺序扫描，并扫描所有的分段（customer l 表只有一个分段组成）。
Lower Index Filter: informix.l.customer_num = informix.r.customer_num	从外表的 customer_num 值开始进行索引扫描。

变更查询计划的指令示例

指令可以变更查询计划。可以使用特定指令来强制优化器选择特定类型的查询计划，例如一个使用散列连接和查询中显示表的顺序的查询计划。

以下示例显示了指令如何变更查询计划。

假设您有以下查询：

```
SELECT * FROM emp,job,dept
WHERE emp.location = 10
   AND emp.jobno = job.jobno
   AND emp.deptno = dept.deptno
   AND dept.location = "DENVER";
```

假设存在以下索引：

```
ix1: emp(empno,jobno,deptno,location)
ix2: job(jobno)
ix3: dept(location)
```

使用 SET EXPLAIN ON 执行查询可显示优化器使用的查询路径。

```
QUERY:
------
SELECT * FROM emp,job,dept
WHERE emp.location = "DENVER"
   AND emp.jobno = job.jobno
   AND emp.deptno = dept.deptno
   AND dept.location = "DENVER"

Estimated Cost: 5
Estimated # of Rows Returned: 1

1) informix.emp: INDEX PATH
```

```
    Filters: informix.emp.location = 'DENVER'

    (1) Index Keys: empno jobno deptno location   (Key-Only)

2) informix.dept: INDEX PATH

    Filters: informix.dept.deptno = informix.emp.deptno

    (1) Index Keys: location
        Lower Index Filter: informix.dept.location = 'DENVER'
NESTED LOOP JOIN

3) informix.job: INDEX PATH

    (1) Index Keys: jobno   (Key-Only)
        Lower Index Filter: informix.job.jobno = informix.emp.jobno
NESTED LOOP JOIN
```

图 *63: 不使用指令的可能的查询计划* 在第245页中的图表显示该查询的一个可能的查询计划。查询计划有三个层次的信息：(1) 嵌套循环连接；(2) 针对一个表的索引扫描和一个嵌套循环连接；(3) 针对其他两个表的索引扫描。

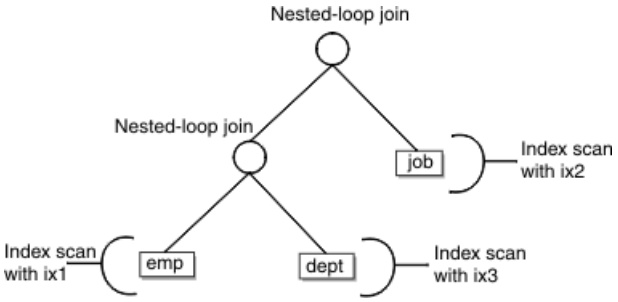

图 63: 不使用指令的可能的查询计划

您也许会担心使用嵌套循环连接不是执行该查询的最快方法。您也可能认为连接顺序不是最优的。那么您可以强制优化器选择散列连接并对查询计划中的表进行排序（根据其在查询中的顺序），因此，优化器使用图 *64: 使用指令的可能的查询计划* 在第245页显示的查询计划。此查询计划有三个层次的信息：(1) 散列连接，(2) 索引扫描和散列连接，以及 (3) 针对其他两个表的索引扫描。

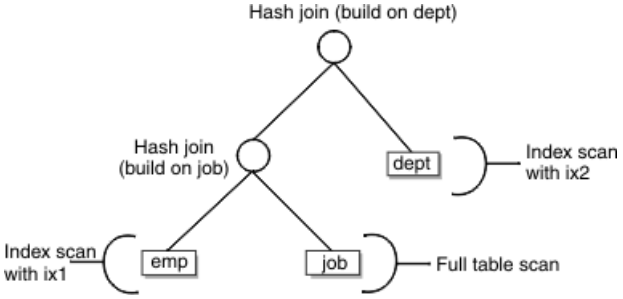

图 64: 使用指令的可能的查询计划

要强制优化器选择使用散列连接的查询计划和查询中显示的表的顺序，请使用以下部分 SET EXPLAIN 输出显示的指令：

```
QUERY:
------
SELECT {+ORDERED,
   INDEX(emp ix1),
   FULL(job),
   USE_HASH(job /BUILD),
   USE_HASH(dept /BUILD),
   INDEX(dept ix3)}
```

```
   * FROM emp,job,dept
   WHERE emp.location = 1
   AND emp.jobno = job.jobno
   AND emp.deptno = dept.deptno
   AND dept.location = "DENVER"

DIRECTIVES FOLLOWED:
ORDERED
INDEX ( emp ix1 )
FULL ( job )
USE_HASH ( job/BUILD )
USE_HASH ( dept/BUILD )
INDEX ( dept ix3 )
DIRECTIVES NOT FOLLOWED:

Estimated Cost: 7
Estimated # of Rows Returned: 1

1) informix.emp: INDEX PATH

    Filters: informix.emp.location = 'DENVER'

    (1) Index Keys: empno jobno deptno location   (Key-Only)

2) informix.job: SEQUENTIAL SCAN


DYNAMIC HASH JOIN
    Dynamic Hash Filters: informix.emp.jobno = informix.job.jobno

3) informix.dept: INDEX PATH

    (1) Index Keys: location
        Lower Index Filter: informix.dept.location = 'DENVER'

DYNAMIC HASH JOIN
    Dynamic Hash Filters: informix.emp.deptno = informix.dept.deptno
```

优化器指令的配置参数和环境变量

您可以使用 DIRECTIVES 配置参数来打开或关闭数据库服务器遇到的所有指令，或者使用 IFX_DIRECTIVES 环境变量来覆盖 DIRECTIVES 配置参数的设置。

如果 DIRECTIVES 配置参数设置为 1（缺省值），那么优化器将遵循所有指令。如果 DIRECTIVES 配置参数设置为 0，那么优化器将忽略所有指令。

可以覆盖 DIRECTIVES 的设置。如果 IFX_DIRECTIVES 环境变量设置为 1 或 ON，那么优化器将遵循用于客户端会话执行的任何 SQL 指令。如果 IFX_DIRECTIVES 为 0 或 OFF，那么优化器会忽略用于客户端会话中的任何 SQL 的指令。

SQL 语句中的任何指令均优先于 OPTCOMPIND 配置参数强制的连接计划。例如，如果查询包含 USE_HASH 指令，并且 OPTCOMPIND 设置为 0（嵌套循环连接优先于散列连接），那么优化器将使用散列连接。

优化器指令和 SPL 例程

对于在 SPL 例程中的查询，指令操作方式有所不同，这是因为在 SPL 例程中 SELECT 语句不一定要在数据库服务器执行之前立刻进行优化。

数据库服务器创建 SPL 例程时或在包含 FOR FUNCTION、FOR PROCEDURE 或 FOR ROUTINE 关键字的 UPDATE STATISTICS 语句的执行期间，优化器在 SPL 例程中为 SELECT 语句创建查询计划。

优化器在创建该查询计划的同时读取并应用指令。由于它将查询计划存储在系统目录表中，因而当执行 SELECT 语句时，将不对该语句进行重新优化。因此，当 IFX_DIRECTIVES 和 DIRECTIVES 在以下任何时间进行设置时，其设置将影响 SPL 例程中的 SELECT 语句：

- 在 CREATE PROCEDURE 语句之前
- 在 UPDATE STATISTICS FOR ROUTINE 语句之前，该指令将优化 SPL 例程中的 SQL 数据操作语句
- 在某些环境期间，当 SELECT 语句运行时有变量提供时

强制重新优化以避免索引或预编译对象异常

如果启用了 AUTO_REPREPARE 配置参数和 IFX_AUTO_REPREPARE 会话环境变量，那么在 DDL 语句修改了引用表的模式之后，SinoDB® 会自动重新编译预编译语句和 SPL 例程。如果禁用 AUTO_REPREPARE 配置参数或 IFX_AUTO_REPREPARE 会话环境变量，那么可以采取措施防止发生错误。

如果禁用了 AUTO_REPREPARE 配置参数或 IFX_AUTO_REPREPARE 会话环境变量，那么在由 SPL 例程间接引用或预编译对象引用的表模式被修复之后执行该预编译对象或 SPL 例程时，将导致以下错误。

```
-710  Table <table-name> has been dropped, altered, or renamed.
```

此错误可以与显式预编译语句一起发生。这些语句具有以下形式：

```
PREPARE statement_id FROM quoted_string;
```

在数据库服务器中预编译语句之后和执行该语句之前，该语句所引用的表可能已重命名或更改，也可能更改了表的结构。结果可能会出现一些问题。

在准备语句之后将索引添加到该表也可能导致该语句失效。如果游标引用无效的预编译语句，那么游标随后的 OPEN 命令将失败；即使 OPEN 命令具有 WITH REOPTIMIZATION 子句，也将发生故障。

如果在准备语句之后添加了索引，那么您必须重新准备该语句并重新声明游标。如果游标基于无效的预编译语句，那么您不能简单地重新打开游标。

此错误也可能与 SPL 例程一起发生。数据库服务器第一次执行新的 SPL 例程之前，将优化 SPL 例程中的代码（语句）。优化使代码依赖于过程引用的表的结构。如果表结构在过程优化之后和执行之前进行了更改，那么可能发生此错误。

每个 SPL 例程在第一次运行时（不是在创建时）进行优化。此行为说明了 SPL 例程可能在第一次运行时成功，但是随后在几乎相同的环境下可能失败。SPL 例程的失败也可能是间歇性的，这是因为在一个执行期间失败将强制内部警告在下一个执行之前重新优化过程。

数据库服务器保留 SPL 例程显式引用的表的列表。只要修改了任何一个显式引用的表，数据库服务器将在下一次执行过程时重新优化过程。

但是，如果 SPL 例程依赖于仅以间接方式引用的表，那么数据库服务器将无法检测到在更改该表之后重新优化过程的需要。例如，如果 SPL 例程调用触发器，那么可以按间接方式引用表。如果触发器引用的表（但不是由 SPL 例程直接引用）发生了更改，那么数据库服务器在执行之前不会知道是否应该重新优化 SPL 例程。更改了表之后执行过程时，将可能发生此错误。

使用两种方法中的一种可以从该错误中进行恢复：

- 发出 UPDATE STATISTICS FOR PROCEDURE 语句可强制重新优化过程。
- 重新执行过程。

要防止发生此错误，您可以强制重新优化 SPL 例程。例如，要强制重新优化名为 procedure_name 的 SPL 例程，请执行以下语句：

```
UPDATE STATISTICS FOR PROCEDURE procedure_name;
```

请注意，以下 UPDATE STATISTICS 语句具有相同效果：

```
UPDATE STATISTICS FOR ROUTINE procedure_name;
```

重要:

请记住，在使用事务日志记录的数据库中，必须在不包含任何其他 SQL 语句的事务中执行 UPDATE STATISTICS 语句。

您可以使用以下两种方式中的一种将该语句添加到程序中：

- 在每个更改对象方式的 DDL 语句之后放置 UPDATE STATISTICS 语句。
- 在每次执行 SPL 例程之前放置 UPDATE STATISTICS 语句。

为提高效率，您可以通过在程序中使用频率较少的操作（更改对象方式或执行过程）来放置 UPDATE STATISTICS 语句。大多数情况下，程序中使用频率较少的操作是更改对象方式。

遵循从该错误中进行恢复的方法时，您必须按间接方式为每个引用已更改的表的过程执行 UPDATE STATISTICS FOR PROCEDURE 语句，除非该过程也显示引用了表。

您也可以通过简单地重新执行 SPL 例程来从此错误中恢复。存储过程第一次失败时，数据库服务器将该过程标记为需要重新优化。下一次执行该过程时，数据库服务器会在执行之前对其进行重新优化。但是，执行 SPL 例程两次可能不切实际或不安全。较安全的方法是使用 UPDATE STATISTICS FOR PROCEDURE 语句来强制对该过程进行重新优化。

外部优化器指令

如果您是用户 informix，那么您可以创建、保存和删除外部指令。

创建和保存外部指令

通过创建包括查询优化器指令的关联记录并将这些记录保存在 sysdirectives 系统目录表中，可以定义外部指令。关联记录将一个或多个优化器指令的列表与特定查询文本相关联。数据库服务器可将这些优化器指令应用于相同查询文本的后续实例。

使用 SAVE EXTERNAL DIRECTIVES 语句来创建要用于一个或多个查询指令的列表的关联记录。这些指令将自动应用到相同查询的后续实例。

以下示例显示 SAVE EXTERNAL DIRECTIVES 语句，该语句将系统目录中的关联记录注册为 sysdirectives 表中的新行，可将其用作查询优化器指令。

```
SAVE EXTERNAL DIRECTIVES {+INDEX(t1,i11)} ACTIVE FOR
   SELECT {+INDEX(t1, i2) } c1 FROM t1 WHERE c1=1;
```

以下数据存储在上述 SQL 语句定义的关联记录中：

```
id              16
query           select {+INDEX(t1, i2) } c1 from t1 where c1=1
directive       INDEX(t1,i11)
directivecode   BYTE value
```

```
active          1
hashcode        -589336273
```

此处，DIRECTIVES 关键字后面的外部指令 {+INDEX(t1,i11)} 将应用于指定查询的未来实例，但会忽略直接插入的 {+INDEX(t1,i2)} 指令。

紧接在 DIRECTIVES 关键字后面的外部指令中的信息必须位于注释指示符中，就如同相同的指令会出现在 SELECT、UPDATE、MERGE 和 DELETE 语句中一样，只是如果外部指令的列表包括多个指令，那么必须使用空白字符（而非逗号 (,)）作为分隔符。

启用外部指令

创建和保存外部指令后，必须设置启用指令的配置参数和环境变量。只要在数据库服务器和客户端上都设置了外部指令，数据库服务器就会为查询搜索指令。

使用 EXT_DIRECTIVES 配置参数（位于 ONCONFIG 文件中）和 IFX_EXTDIRECTIVES 客户端环境变量的组合来启用指令。

您可使用的 EXT_DIRECTIVE 值是：

值	解释
0（缺省值）	关闭。即使启用 IFX_EXTDIRECTIVES，也无法启用指令。
1	打开。如果启用 IFX_EXTDIRECTIVES，那么可以对会话启用指令。
2	打开。即使未启用 IFX_EXTDIRECTIVES，也可以使用指令。

您也可以使用 SET ENVIRONMENT 语句的 EXTDIRECTIVES 选项在会话期间启用或禁用外部指令。使用 EXTDIRECTIVES 选项所指定的值将覆盖 ONCONFIG 文件中 EXT_DIRECTIVES 配置参数中指定的外部指令设置。

要覆盖 ONCONFIG 文件中用于启用或禁用外部指令的值：

- 要在会话期间启用外部指令，请将 1、on 或 ON 指定为 SET ENVIRONMENT EXTDIRECTIVES 的值。
- 要在会话期间禁用外部指令，请将 0、off 或 OFF 指定为 SET ENVIRONMENT EXTDIRECTIVES 的值。

要在会话期间启用在 EXT_DIRECTIVES 配置参数以及客户端 IFX_EXTDIRECTIVES 环境变量中指定的缺省值，请将 DEFAULT 指定为 SET ENVIRONMENT 语句中 EXTDIRECTIVES 选项的值。

说明输出文件会指定外部指令是否有效。

相关链接

说明输出文件 在第217页
查询统计信息部分提供性能调试信息 在第217页
显示由优化器选择的查询计划的报告 在第216页
《*SinoDB SQL* 指南: 语法》: *SET EXPLAIN* 语句
《*SinoDB SQL* 指南: 语法》: 使用 *FILE TO* 选项
《*SinoDB SQL* 指南: 语法》: *UNIX* 上说明输出文件的缺省名称和位置
《*SinoDB SQL* 指南: 语法》: *Windows* 上输出文件的缺省名称和位置
显示由优化器选择的查询计划的报告 在第216页
说明输出文件 在第217页
查询统计信息部分提供性能调试信息 在第217页
《*SinoDB* 管理员参考》: *onmode -Y*: 动态更改 *SET EXPLAIN*
《*SinoDB* 管理员参考》: *onmode* 和 *Y* 参数: 更改会话的查询计划度量 *(SQL* 管理 *API)*

删除外部指令

当不再需要某个外部指令时，DBA 或用户 informix 可使用 SQL 的 DELETE 语句从 sysdirectives 系统目录表中将其移除。

启用了外部指令且 sysdirectives 系统目录表不为空时，

- 数据库服务器将每个查询与每个 ACTIVE 外部指令的查询文本进行比较，
- 并将 DBA（或用户 informix）执行的查询与每个 TEST ONLY 外部指令进行比较。

外部指令的目的是提高与查询字符串匹配的查询的性能，但如果查询执行优化器必须将大量活动外部指令的查询字符串与每个 SELECT 语句的文本进行比较，那么使用此类指令就可能会减慢其他查询的速度。因此，星瑞格® 建议 DBA 不允许 sysdirectives 表积累过多的 ACTIVE 行。（要避免对其他查询的意外性能影响，另一种方法是通过将 EXT_DIRECTIVES 配置参数设置为 0 来禁用对外部指令的支持。将 IFX_EXTDIRECTIVES 客户端环境变量设置为 0 具有相同的效果。）

第

12

章

并行数据库查询 (PDQ)

您可以管理数据库服务器如何执行 PDQ，并可以监视数据库服务器用于 PDQ 的资源。

什么是 PDQ

并行数据库查询 (PDQ) 是一种数据库服务器功能，当服务器处理决策支持应用程序初始化的查询时，该功能可以显著提高性能。PDQ 使 SinoDB® 能够将查询的一个方面的工作分发给多个处理器。例如，如果查询要求聚合，那么 SinoDB® 可以将聚合工作分发给几个处理器。

PDQ 也包含用于资源管理的工具。

数据库服务器的另一功能 - 表分段存储，允许您将表的各部分存储在不同的磁盘上。查询的数据位于分段表时，PDQ 将提供最佳的性能优势。有关如何使用分段存储以获得最佳性能的信息，请参阅规划分段存储策略 在第188页。

相关链接

使用 *PDQ* 的数据库服务器操作 在第252页
为并行数据库查询分配资源 在第255页
管理 *PDQ* 查询 在第259页
监视用于 *PDQ* 和 *DSS* 查询的资源 在第261页

PDQ 查询的结构

每个决策支持查询都有主线程。数据库服务器可能启动附加的线程来执行查询任务（例如：扫描和排序）。根据查询必须搜索的表或分段的数量以及决策支持查询可使用的资源，数据库服务器将查询的不同组成部分分配给不同的线程。

数据库服务器启动这些 PDQ 线程，这些线程在 SET EXPLAIN 输出中列为辅助线程。

根据其功能，辅助线程可进一步分类为生产者或消费者。生产者线程给另一线程提供数据。例如,扫描线程可从与给定表相对应的共享内存读取数据并将数据传递到连接线程。在这种情况下，扫描线程被看作生产者，而连接线程则被看作消费者。反过来，连接线程也可将数据传递到排序线程。当执行此操作时，连接线程被看作生产者，而排序线程则被看作消费者。

几个生产者可以向单个消费者提供数据。当这种情况发生时，数据库服务器会建立一个内部机制，称为交换，该机制使从那些生产者到消费者的数据传输同步。例如，如果要对分段表进行排序，那么优化器通常会对每个分段调用不同的扫描线程。由于 I/O 特征有所不同，因而扫描线程预期可能完成的时间也将不同。交换用来将各种扫描线程产生的数据传递给一个或多个排序线程，而最低限度地使用缓冲。根据查询的复杂性，优化器可能会需要一个生产者、交换和消费者的多层结构。一般来讲，消费者线程与生产者线程并行工作，这样交换执行的中间缓冲量可忽略不计。

数据库服务器自动并透明地创建这些线程和交换。当完成对给定查询的处理时，它们将自动终止。数据库服务器按后续查询需要创建新的线程和交换。

使用 PDQ 的数据库服务器操作

SinoDB® 处理数据库服务器并行处理的某些 SQL 操作类型。但是，某些情况将限制 SinoDB® 可以使用的并行度。

在本部分中有关使用 PDQ 的数据库服务器操作的主题中，查询是任意 SELECT 语句。

相关链接

什么是 *PDQ* 在第251页

并行更新和删除操作

SinoDB® 并行执行某些类型的更新和删除操作。

数据库服务器采用以下两个步骤来处理 UPDATE 和 DELETE 语句：

1. 获取符合条件的行。
2. 应用更新或删除操作。

数据库服务器并行执行 UPDATE 或 DELETE 语句的第一个步骤，但以下情况是例外：

- DELETE 语句中的目标表具有可以级联到子表的参考约束。
- UPDATE 或 DELETE 语句包含一个 OR 子句且优化器选择 OR 索引来处理 OR 过滤器。
- UPDATE 语句包含一个可通过优化器转换为连接的子查询。

并行插入操作

SinoDB® 并行执行某些类型的插入操作。

服务器并行执行的插入操作类型为：

- SELECT...INTO TEMP 使用显式临时表插入。
- INSERT INTO...SELECT 使用隐式临时表插入。

使用 SELECT...INTO TEMP 语句的显式插入

数据库服务器可以向您在 SELECT....INTO TEMP 格式的 SQL 语句中指定的显式临时表并行插入行。

例如，数据库服务器可并行执行对临时表 temp_table 的插入，如以下示例所示：

```
SELECT * FROM table1 INTO TEMP temp_table
```

要执行对临时表的并行插入：

1. 设置 PDQ 优先级 > 0。

   对于您想要数据库服务器并行执行的任何查询，均必须满足该要求。

2. 将 DBSPACETEMP 设置为两个或更多数据库空间的列表。

   由于数据库服务器执行插入的方式，因此该步骤是必需的。要并行执行插入，数据库服务器要先创建一个分段临时表。为了使数据库服务器了解临时表分段的存储位置，您必须在 DBSPACETEMP 配置参数或 DBSPACETEMP 环境变量中指定两个或更多数据库空间的列表。此外，执行 SELECT...INTO 语句之前，您必须设置 DBSPACETEMP 以指示用于分段的存储空间。

数据库服务器通过以循环方式并行写入每个分段来执行并行插入。性能随着分段数的增加而提高。

使用 INSERT INTO...SELECT 语句的隐式插入

当数据库服务器处理 INSERT INTO...SELECT 格式的 SQL 语句时，也可以将行并行插入创建的隐式表。

例如，数据库服务器会并行处理以下 INSERT 语句：

```
INSERT INTO target_table SELECT * FROM source_table
```

目标表可以是一个永久表，也可以是一个临时表。

只有当目标表满足以下条件时，数据库服务器才能并行处理该类型的 INSERT 语句：

- PDQ 优先级的值大于 0。
- 目标表被分段成两个或更多数据库空间。
- 目标表没有已启用的参考约束或触发器。
- 目标表不是远程表。
- 在一个带有日志记录的数据库中，目标表不包含过滤约束。
- 目标表不包含 TEXT 或 BYTE 数据类型的列。

数据库服务器不处理引用 SPL 例程的并行插入。例如，数据库服务器从不并行处理以下语句：

```
INSERT INTO table1 EXECUTE PROCEDURE ins_proc
```

并行索引构建

索引构建可以利用 PDQ 且可以并行进行。数据库服务器会为索引构建并行执行扫描和排序。

以下操作将启动索引构建：

- 创建一条索引。
- 添加一个唯一的主键。
- 添加一个参考约束。
- 启用一个参考约束。

PDQ 生效时，索引构建的扫描将受为并行数据库查询分配资源 在第255页中所述的 PDQ 配置参数控制。

如果您的计算机具有多个 CPU，那么数据库服务器将使用两个排序线程对索引键进行排序。在索引构建过程中，用户没有设置 PSORT_NPROCS 环境变量的情况下，数据库服务器将使用两个排序线程。

并行用户定义例程

如果查询在一个表达式中包含用户定义例程 (UDR)，那么开启 PDQ 时，数据库服务器可以并行执行查询。

如果正确写入并注册了 UDR，那么数据库服务器可以执行以下并行操作：

- 并行扫描
- 使用 UDR 的并行比较

有关如何启用 UDR 的并行执行的更多信息，请参阅并行 *UDR* 在第294页。

使用 PDQ 的控制游标

当通过声明 WITH HOLD 限定符创建的控制游标没有任何锁时，就启用了 PDQ。

在以下情况下，将会为控制游标设置 PDQ：

- 使用 Dirty Read 或 Committed Read 隔离级别、ANSI 和只读游标的查询
- 使用 Dirty Read 或 Committed Read 隔离级别、非 ANSI 和不可更新游标的查询

不使用 PDQ 的 SQL 查询

数据库服务器不并行处理某些类型的查询。

例如，服务器不并行处理以下类型的查询：

- 以 Cursor Stability 隔离级别开始的查询

  对隔离级别的后续更改不影响已准备好的查询的并行性。这种情况是由并行扫描的内在性质产生的，并行扫描将同时扫描多行。
- 使用声明为 FOR UPDATE 的游标的查询
- *Select* 触发器 Action 子句的 FOR EACH ROW 部分中的查询
- *Delete* 触发器 Action 子句的 FOR EACH ROW 部分中的 DELETE 或 MERGE 语句
- *Insert* 触发器 Action 子句的 FOR EACH ROW 部分中的 INSERT 或 MERGE 语句

- *Update* 触发器 Action 子句的 FOR EACH ROW 部分中的 UPDATE 或 MERGE 语句
- 数据定义语言 (DDL) 语句。

  有关 SinoDB® 所支持的 SQL DDL 语句的完整列表，请参阅《*SinoDB® SQL* 指南: 语法》。

此外，数据库服务器不会处理 PDQ 操作中的顺序对象。如果 SQL 语句包含排序操作（例如，带有 NEXTVAL 或 CURRVAL 运算符的表达式），那么该语句将无法使用 PDQ 处理功能。

受 PDQ 影响的更新统计信息操作

未并行处理的 SQL UPDATE STATISTICS 语句将受到 PDQ 影响，这是因为它必须分配用于排序的内存。因此，UPDATE STATISTICS 语句的行为将受到与 PDQ 关联的内存管理的影响。

数据库服务器必须分配内存给 UPDATE STATISTICS 语句排序。

如果您有极大的数据库，并且索引都已分段，那么 UPDATE STATISTICS LOW 可以自动并行执行语句。有关更多信息，请参阅在超大型数据库上并行更新统计信息 在第278页。

SPL 例程和触发器以及 PDQ

涉及 SPL 例程的语句不并行执行。但是，过程内的语句要并行执行。

数据库服务器执行一个 SPL 例程时，它将不使用 PDQ 来处理包含在过程中不相关的 SQL 语句。但是，通过使用内部查询并行性（在可能的情况下），可以独立地并行执行每个 SQL 语句。因此，如果想要使用数据库服务器的并行处理功能，那么应该在数据操作语言 (DML) 语句中限制使用过程调用。有关 DML 语句的完整列表，请参阅《*SinoDB® SQL* 指南: 语法》。

数据库服务器使用内部查询并行性来处理 SQL 触发器主体中的语句，方法与处理 SPL 例程中的语句的方法相同。有关在 Select、Insert 和 Update 触发器的某些触发操作中对查询使用 PDQ 的限制，请参阅不使用 *PDQ* 的 *SQL* 查询 在第253页。

相关和不相关的子查询

数据库服务器不会使用 PDQ 处理相关的子查询。一次只有一个线程可以执行相关子查询。当一个线程执行相关子查询时，其他请求执行该子查询的线程将暂停，直到第一个线程完成。

对于不相关的子查询，实际上只有发出请求的第一个线程执行该子查询。其他线程只是使用子查询的结果并可以并行执行。

因此，强烈建议您只要可能就要使用连接而不是子查询来建立查询，以便可以利用 PDQ 查询。

OUTER 索引连接和 PDQ

在查询期间，数据库服务器将包含 OUTER 索引连接的查询的 PDQ 优先级降低为 LOW（如果将优先级设置为较高的值）。如果子查询或视图包含 OUTER 索引连接，那么数据库服务器将只降低该子查询或视图而不是父查询或其他任何子查询的 PDQ 优先级。

与 PDQ 一起使用的远程表

虽然数据库服务器可并行处理存储在远程表中的数据，但由于数据库服务器只分配一个线程用于从远程表提交和接收数据，因此数据通信是串行的。数据库服务器将需要访问远程数据库的查询的 PDQ 优先级降低为 LOW。

在此情况下，所有的本地扫描都是并行的，但所有的本地连接和远程访问都是非并行的。

内存分配管理器

内存分配管理器 (MGM) 是一个数据库服务器组件，在决策支持查询中协调内存、CPU 虚拟处理器 (VP)、磁盘 I/O 以及扫描线程的使用。MGM 使用 DS_MAX_QUERIES、DS_TOTAL_MEMORY、DS_MAX_SCANS 和 MAX_PDQPRIORITY 配置参数来确定可分配给决策支持查询的 PDQ 资源的数量。

MGM 为决策支持查询动态分配以下资源：

- 为每个决策支持查询所启动的扫描线程数量
- 可以为每个查询启动的线程数量
- 查询在数据库服务器共享内存的虚拟部分中可以保留的内存量

当您的数据库服务器系统频繁使用 OLTP，并且发现性能在下降时，您可以使用 MGM 工具限制提交给决策支持查询的资源量。在非峰值时段，您可将大部分资源指定给并行处理，从而实现决策支持查询更高的吞吐量。

MGM 为排序、散列连接和处理 GROUP BY 子句等活动的查询分配内存。决策支持查询使用的内存量不能超过 DS_TOTAL_MEMORY。

MGM 按份额增量的方式为查询分配内存。要计算份额的近似大小，请使用以下公式：

```
内存份额 = DS_TOTAL_MEMORY / DS_MAX_QUERIES
```

例如，如果 DS_TOTAL_MEMORY 为 12 MB，而 DS_MAX_QUERIES 为 4，那么份额为 3 MB (12/4)。因此，在这些值都生效的情况下，一个内存份额等于 3 MB。数据库服务器在分配内存时可以动态调整份额的大小。一般而言，份额越小，内存分配效率越高。通常您可以通过增加 DS_MAX_QUERIES 以减少内存份额的大小来提高并发查询的性能。

要监视 MGM 分配的资源，请执行 onstat -g mgm 命令。该命令只显示当前使用的内存量；而不显示分配的内存量。

MGM 也会根据 DS_MAX_SCANS 和 DS_MAX_QUERIES 参数的值为每个查询分配最大数目的扫描线程。

以下公式得出每个查询最大数目的扫描线程：

```
scan_threads = min (nfrags, DS_MAX_SCANS * (pdqpriority / 100)
   * (MAX_PDQPRIORITY / 100))
```

nfrags

是具有最多分段数的表中的分段数。

pdqpriority

是由 PDQPRIORITY 环境变量或 SQL 语句 SET PDQPRIORITY 设置的 PDQ 优先级的值。

PDQPRIORITY 环境变量和 SQL 语句 SET PDQPRIORITY 为查询请求一部分的 PDQ 资源。您可以使用 MAX_PDQPRIORITY 配置参数来限制查询可获取的请求资源的百分比，并且限制决策支持查询对 OLTP 处理的影响。

相关链接

配置对内存利用率的影响 在第55页
限制决策支持查询的优先级 在第256页
DS_TOTAL_MEMORY 配置参数和内存利用率 在第63页
《*SinoDB* 管理员参考》: *onstat -g mgm* 命令: 显示 *MGM* 资源信息

为并行数据库查询分配资源

配置数据库服务器时，需要考虑 PDQ 的使用将如何影响 OLTP、决策支持 (DSS) 应用程序和其他应用程序的用户。然后可以计划如何为 PDQ 分配资源。

数据库服务器使用 PDQ 来并行执行查询时，会给操作系统带来繁重的负载。PDQ 将特别利用以下资源：

- 内存
- CPU VP
- 磁盘 I/O（到分段表和临时表空间）
- 扫描线程

您可用以下方法控制数据库服务器如何使用资源：

- 限制并行数据库查询的优先级。
- 调整内存量。
- 限制扫描线程数量。
- 限制并发查询的数量。

相关链接

什么是 *PDQ* 在第251页

限制决策支持查询的优先级

可以通过调整 PDQPRIORITY 环境变量、MAX_PDQPRIORITY 配置参数和其他配置参数的值来限制决策支持 (DSS) 查询将消耗的并行处理资源。

单个应用程序的 PDQ 优先级的缺省值为 0，这意味着没有使用 PDQ 处理。除非以下其中一种操作覆盖了该值，否则数据库服务器将使用该值：

- 设置 PDQPRIORITY 环境变量。
- 应用程序使用 SET PDQPRIORITY 语句。

PDQPRIORITY 环境变量和 MAX_PDQPRIORITY 配置参数共同控制为并行处理分配的资源量。正确设置这些参数对 PDQ 的有效操作非常关键。

MAX_PDQPRIORITY 配置参数使您能够对 DSS 查询消耗的并行处理资源进行限制。因此，PDQPRIORITY 环境变量将设置一个合理的或建议的优先级值，并且 MAX_PDQPRIORITY 将对应用程序可要求的资源进行限制。

MAX_PDQPRIORITY 配置参数指定查询可获取的最大百分比的请求资源。例如，如果 PDQPRIORITY 为 80 且 MAX_PDQPRIORITY 为 50，那么每个活动查询将接收到相当于 DS_TOTAL_MEMORY 的 40% 的内存量，并四舍五入为最接近的份额。在此示例中，MAX_PDQPRIORITY 有效地将并发决策支持查询的数量限制为两个。后续的查询必须等到前面的查询完成之后，才能获得其运行所需的资源。

如果 PDQPRIORITY 环境变量设置了 PDQ 优先级的值，那么应用程序或用户可以使用 SET PDQPRIORITY 语句的 DEFAULT 标记来使用该值。DEFAULT 是 PDQ 优先级的 -1 值的等值符号。

您可以使用 onmode 命令行实用程序来临时更改以下配置参数的值：

- 使用 onmode -M 来更改 DS_TOTAL_MEMORY 的值。
- 使用 onmode -Q 来更改 DS_MAX_QUERIES 的值。
- 使用 onmode -D 来更改 MAX_PDQPRIORITY 的值。
- 使用 onmode -S 来更改 DS_MAX_SCANS 的值。

只要数据库服务器仍处于启动并运行状态，这些更改就保持有效。数据库服务器启动时，将使用 ONCONFIG 文件中列出的值。

有关上述参数的更多信息，请参阅配置对内存利用率的影响 在第55页。有关 onmode 的更多信息，请参阅《*SinoDB*® 管理员参考》。

如果必须定期更改决策支持参数的值（例如，为了处理报告每天晚上将 MAX_PDQPRIORITY 设置为 100），那么您可以使用调度好的操作系统作业来设置这些值。有关创建调度作业的信息，请参阅操作系统手册。

要获得数据库服务器的最佳性能，请先选择 PDQPRIORITY 环境变量和 MAX_PDQPRIORITY 参数的值，然后观察结果，再对这些参数的值进行调整。没有现成已定义的规则可以指导这些环境变量和参数值的选择。以下几节将讨论为特定需求设置 PDQPRIORITY 和 MAX_PDQPRIORITY 的策略。

相关链接

内存分配管理器 在第254页

限制 PDQ 优先级的值

可以调整 MAX_PDQPRIORITY 配置参数的值以调整 PDQ 优先级并将更多资源分配给 OLTP 或决策支持处理。

用户设置 PDQPRIORITY 环境变量或在发出查询之前发出 SET PDQPRIORITY 语句时，MAX_PDQPRIORITY 配置参数将限制数据库服务器授权的 PDQ 优先级。当应用程序或最终用户试图设置一个 PDQ 优先级时，授予的优先级将与 MAX_PDQPRIORITY 指定的值相乘。

如果要将更多的资源分配给 OLTP 处理时，请减小 MAX_PDQPRIORITY 的值。

如果想要将更多的资源分配给决策支持处理时，请设置更高的 MAX_PDQPRIORITY 值。

该值可能的范围为 0 到 100。该范围表示您可以分配给决策支持处理的资源百分比。

最大限度提高查询的 OLTP 吞吐量
有时您可能只想分配资源以获取单个 OLTP 查询而不是决策支持查询的最大吞吐量。

在这种情况下，将 MAX_ PDQPRIORITY 设置为 0，这会将 PDQ 优先级的值限制为 OFF。OFF 的 PDQ 优先级值不会阻止决策支持查询的运行。而是使查询不以并行方式执行。在该配置中，决策支持查询的响应时间会很慢。

使用 PDQ 时节省资源
如果应用程序很少使用需要并行排序和并行连接的查询，那么您应考虑使用 PDQ 优先级的 LOW 设置。

如果数据库服务器正在多用户环境中运行，那么您可以将 MAX_PDQPRIORITY 设置为 1，以一定的内部查询并行性为代价来提高内部查询性能。因为两种不同类型的并行性争用相同的资源，所以它们之间存在着折衷方案。作为折衷，您可以将 MAX_PDQPRIORITY 设置为某个中间值（可能为 20 或 30），并将 PDQPRIORITY 设置为 LOW。环境变量会将缺省行为设置为 LOW，但是 MAX_PDQPRIORITY 配置参数使单个应用程序能够使用 SET PDQPRIORITY 语句来请求更多的资源。

允许最大程度地使用并行处理
如果您希望数据库服务器为并行处理分配尽可能多的资源，那么请将 PDQPRIORITY 和 MAX_PDQPRIORITY 设置为 100。

该设置适用于并行处理不干扰 OLTP 处理的时候。

确定并行处理级别
您可以设置 PDQPRIORITY 为不同的数值来试验并行性对单个应用程序的影响。

有关如何监视并行执行的信息，请参阅监视用于 *PDQ* 和 *DSS* 查询的资源 在第261页。

对与 PDQ 优先级相关联的并行操作的限制
在查询期间，数据库服务器将包含外连接的查询的 PDQ 优先级降低为 LOW（如果设置为一个较高的值）。如果子查询或视图包含外连接，那么数据库服务器只降低该子查询或视图而不是父查询或其他任何子查询的 PDQ 优先级。

如果将需要访问远程数据库（相同或不同的数据库服务器实例）的查询的 PDQ 优先级设置为一个较高的值，那么数据库服务器会将该值降低为 LOW。在这种情况下，所有的本地扫描都是并行的，但所有的本地连接和远程访问都是非并行的。

将 SPL 例程与 PDQ 查询一起使用
在创建过程或使用 UPDATE STATISTICS 语句进行最后的手动重新编译时，数据库服务器将冻结用于优化 SPL 例程中 SQL 语句的 PDQ 优先级。您可以更改 PDQPRIORITY 的客户端值。

要更改 PDQPRIORITY 的客户端值，请将 SET PDQPRIORITY 语句嵌入您的 SPL 例程的主体中。

数据库服务器用来优化或重新优化 SQL 语句的 PDQ 优先级值是由 SET PDQPRIORITY 语句设置的值，该语句必须已在相同的过程中执行。如果尚未执行这样的语句，将使用最后一次编译或创建过程时生效的优先级值。

当一个过程正在执行时，在该过程外部当前有效的 PDQ 优先级值将会在过程内被忽略。

建议您在输入一个过程时先关闭 PDQ 优先级，然后为特定的语句再将其打开。从而避免该过程占用大量的内存，并且可以确保该过程的关键部件使用相应的 PDQ 优先级，如以下示例所示：

```
CREATE PROCEDURE my_proc (a INT, b INT, c INT)
   Returning INT, INT, INT;
SET PDQPRIORITY 0;
...
SET PDQPRIORITY 85;
SELECT ... (big complicated SELECT statement)
```

```
SET PDQPRIORITY 0;
...
;
```

调整 DSS 和 PDQ 查询的内存量

估算要分配给决策支持 (DSS) 查询的共享内存量。然后如果必要，可以调整 DS_TOTAL_MEMORY 配置参数（其指定可用于 PDQ 查询的内存量）的值。

使用以下公式作为估算分配给 DSS 查询的共享内存量的开始点：

```
DS_TOTAL_MEMORY = p_mem - os_mem - rsdnt_mem -
                  (128 kilobytes * users) - other_mem
```

p_mem

表示主机上可用物理内存的总量。

os_mem

表示操作系统的大小，包括缓冲区高速缓存。

resdnt_mem

表示 SinoDB® 常驻共享内存的大小。

users

是在 NETTYPE 配置参数中指定的期望用户（连接）的数量。

other_mem

用于其他（非 SinoDB®）应用程序的内存大小。

从该公式得到的 DS_TOTAL_MEMORY 值只充当一个开始点。要得到符合配置的值，您必须监视调页和换页。（使用随操作系统提供的工具来监视调页和换页。）页调度增加时，将降低 DS_TOTAL_MEMORY 的值，以便 OLTP 工作量处理可以继续。

分配给单个并行数据库查询的内存量取决于许多系统因素，但通常情况下，分配给单个并行数据库查询的内存量与以下公式成比例：

```
memory_grant_basis = (DS_TOTAL_MEMORY/DS_MAX_QUERIES) *
                (PDQPRIORITY / 100) *
                (MAX_PDQPRIORITY / 100)
```

但是，如果服务器实例的所有数据库上当前执行的查询需要的内存多于平均分配的估算值，那么其他查询可能将溢出到磁盘或者等待，直至并发查询完成了执行并释放了用于运行查询的足够内存资源。以下备选公式可直接估算用于单个查询的 PDQ 内存：

```
memory_for_single_query = DS_TOTAL_MEMORY  *
                (PDQPRIOIRTY / 100) *
                (MAX_PDQPRIORITY / 100)
```

限制并发扫描的数量

数据库服务器根据查询的 PDQ 优先级（结合其他因素）为该查询分配一定数量的扫描。您可以调整 DS_MAX_SCANS 配置参数值来限制并发扫描数。

DS_MAX_SCANS 和 MAX_PDQPRIORITY 配置参数允许您根据以下公式来限制用户可以分配给查询的资源：

```
scan_threads = min (nfrags, (DS_MAX_SCANS * (pdqpriority / 100)
   * (MAX_PDQPRIORITY / 100) )
```

nfrags

是具有最多分段数的表中的分段数。

pdqpriority

是 PDQPRIORITY 环境变量或 SET PDQPRIORITY 语句设置的 PDQ 优先级值。

例如，假设一个大表包含 100 个分段。如果不对允许的并发扫描数进行限制，数据库服务器将并发执行 100 个扫描线程来读取该表。此外，许多用户可启动此查询。

作为数据库服务器管理员，您应将 DS_MAX_SCANS 配置参数设置为小于此表中的分段数的值，以防止数据库服务器的多重决策支持查询所需的扫描线程数过多。您可以将 DS_MAX_SCANS 设置为 20，以确保数据库服务器为并行扫描最多并发执行 20 个扫描线程。而且，如果多个用户都启动并行数据库查询，那么根据分配给查询的 PDQ 优先级和数据库服务器管理员设置的 MAX_PDQPRIORITY 配置参数值，每个查询只能使用一定百分比的 20 个扫描线程。

管理 PDQ 查询

为处理某个查询，数据库服务器管理员、应用程序编写人员和用户对 SinoDB® 分配的资源量都有一定程度的控制。数据库服务器管理员通过使用配置参数实现控制。应用程序开发者或用户可以通过环境变量或 SQL 语句实现控制。

相关链接

什么是 *PDQ* 在第251页

使用 SET EXPLAIN 输出分析查询计划

您可以使用 SET EXPLAIN 输出来研究应用程序的查询计划。SET EXPLAIN 语句的输出显示查询优化器作出的决策。并显示了是否已使用并行扫描、响应查询所需的最大线程数以及用于查询的连接类型。

您可以重新构造一个查询或使用 OPTCOMPIND 来更改优化器处理查询的方法。

影响查询计划的选择

OPTCOMPIND 环境变量和 OPTCOMPIND 配置参数指示首选连接计划，从而帮助优化器为并行数据库查询选择相应的连接方法。要影响优化器对连接计划的选择，您可以设置 OPTCOMPIND 配置参数。

只有当应用程序不设置 OPTCOMPIND 环境变量时，才会引用分配给 OPTCOMPIND 配置参数的值。

如果希望数据库服务器选择的连接计划与数据库服务器 V6.0 以前的版本完全相同，请将 OPTCOMPIND 设置为 0。该选项确保了与数据库服务器以前版本的兼容性。

当应用程序执行散列连接时，将使用 Repeatable Read 隔离方式，可以锁定表中的所有记录。因此，您应该将 OPTCOMPIND 设置为 1。

如果您希望优化器根据成本作出决定，而不考虑应用程序的隔离级别，那么将 OPTCOMPIND 设置为 2。

可使用 SET ENVIRONMENT OPTCOMPIND 命令在会话中更改 OPTCOMPIND 的值。有关使用此命令的更多信息，请参阅设置会话中 *OPTCOMPIND* 的值 在第45页。

有关 OPTCOMPIND 和其他连接计划的更多信息，请参阅查询计划 在第210页。

动态设置 PDQ 优先级

您可以使用 SET PDQPRIORITY 语句在应用程序中动态设置 PDQ 优先级。PDQ 优先级值可以是 -1 到 100 之间的任意整数。

使用 SET PDQPRIORITY 语句设置的 PDQ 优先级将取代 PDQPRIORITY 环境变量。

SET PDQPRIORITY 语句的 DEFAULT 标记使应用程序能够将 PDQ 优先级的值还原为环境变量设置的值（如果有该值的话）。有关 SET PDQPRIORITY 语句的更多信息，请参阅《*SinoDB® SQL* 指南: 语法》。

允许数据库服务器分配 PDQ 内存

可以使用 SET ENVIRONMENT 语句的 IMPLICIT_PDQ 会话环境选项，以允许数据库服务器在当前会话期间确定要分配给查询的 PDQ 内存量。这可能会覆盖当前 PDQPRIORITY 设置。

但是，数据库服务器可以分配的最大内存量受限于系统可用的物理内存以及以下参数的设置：

- PDQPRIORITY 环境变量
- SQL 的最新 SET PDQPRIORITY 语句
- MAX_PDQPRIORITY 配置参数
- DS_TOTAL_MEMORY 配置参数
- BOUND_IMPL_PDQ 配置参数。

IMPLICIT_PDQ 会话环境选项仅在支持 PDQPRIORITY 的系统上可用。

缺省情况下，IMPLICIT_PDQ 会话环境变量设置为 OFF。IMPLICIT_PDQ 设置为 OFF 时，服务器将不会覆盖当前 PDQPRIORITY 设置。

若要允许数据库服务器根据其需求确定查询的内存分配以及在查询操作程序之间分发内存，请在发出查询之前输入以下语句：

```
SET ENVIRONMENT IMPLICIT_PDQ ON;
```

如果改为将 IMPLICIT_PDQ 值设置为 1 到 100 范围内的整数，那么数据库服务器会将其估算值按照指定的值进行缩放。例如，以下示例将会话的后续查询中的内存分配限制为当前 PDQPRIORITY 设置的一半：

```
SET ENVIRONMENT IMPLICIT_PDQ '50';
```

如果设置了较低的 IMPLICIT_PDQ 值，那么分配给查询的内存量会按比例减少，这可能会增加查询操作程序溢出量。

查询的 IMPLICIT_PDQ 功能至少需要对该查询访问的所有表的 LOW 分布统计信息。如果查询引用的一个或多个表缺少分布统计信息，那么 IMPLICIT_PDQ 设置对可用于执行查询的资源没有影响。此限制也适用于星型连接查询，在缺少统计信息的情况下不支持此类查询。

通过设置 BOUND_IMPL_PDQ 限制 PDQ 资源分配

如果 IMPLICIT_PDQ 设置为 ON 或数字值，那么也可以使用 SQL 的 SET ENVIRONMENT 语句的 BOUND_IMPL_PDQ 会话环境选项来指定分配的 PDQ 内存不应大于当前显式 PDQPRIORITY 值或范围。如果 IMPLICIT_PDQ 会话环境设置为 OFF（无论是显式设置还是缺省值），那么 BOUND_IMPL_PDQ 设置将不起作用。

例如，如果 IMPLICIT_PDQ 会话环境选项已设置，那么以下语句会强制数据库服务器将显式 PDQPRIORITY 值用作分配内存的准则：

```
SET ENVIRONMENT BOUND_IMPL_PDQ ON;
```

如果 IMPLICIT_PDQ 设置为 1 到 100 范围内的整数，那么在当前会话期间将根据该设置按百分比缩放显式 PDQPRIORITY 值。

当 BOUND_IMPL_PDQ 会话环境选项设置为 ON（或 1）时，需要数据库服务器将显式 PDQPRIORITY 设置用作可以分配给查询的内存上限。如果同时设置了 IMPLICIT_PDQ 和 BOUND_IMPL_PDQ，那么显式 PDQPRIORITY 值将确定可分配给查询的内存上限。

如果在 SET ENVIRONMENT 语句中包含整数值，那么必须将该值括在引号内。但是，不要将 ON 和 OFF 关键字括在引号内。

以下示例是带有整数值的语句：

```
SET ENVIRONMENT IMPLICIT_PDQ "50";
SET ENVIRONMENT BOUND_IMPL_PDQ "1";
```

PDQ 资源的用户控制

要指示查询的 PDQ 优先级，您可以在发出一条查询之前设置 PDQPRIORITY 环境变量或执行 SET PDQPRIORITY 语句。这些 PDQ 优先级选项允许您为查询操作申请一定数量的并行处理资源。

您申请的资源与数据库服务器为查询操作分配的资源量可能会有所不同。数据库服务器管理员使用 MAX_PDQPRIORITY 配置参数来设定用户请求资源的上限时，将会产生这种差异，如以下主题说明。

PDQ 和 DSS 查询资源的 DBA 控制

要管理数据库服务器分配给并行数据库和决策支持 (DSS) 查询的资源总量，数据库服务器管理员可设置环境变量和配置参数。

控制分配给 PDQ 的资源

要控制分配给 PDQ 的资源，您可以设置 PDQPRIORITY 环境变量。不设置 PDQPRIORITY 环境变量的查询在发出查询之前将不使用 PDQ。此外，要想对用户指定的 PDQ 优先级级别设定一个上限，您可以设置 MAX_PDQPRIORITY 配置参数。

设置 PDQPRIORITY 环境变量和 MAX_PDQPRIORITY 参数时，您可以实现对数据库服务器在 OLTP 和 DSS 应用程序之间分配的资源进行控制。例如，如果 OLTP 处理在一天中的特定时期特别繁重，那么您可能想要将 MAX_PDQPRIORITY 设置为 0。该配置参数对使用 PDQPRIORITY 环境变量的用户所请求的资源设定了一个上限，因此 PDQ 将会一直保持关闭状态直到您将 MAX_PDQPRIORITY 重新设置为一个非零值为止。

DBA 控制分配给决策支持查询的资源

DBA 可通过设置 DS_TOTAL_MEMORY、DS_MAX_SCANS 和 DS_MAX_QUERIES 配置参数来控制数据库服务器分配给决策支持查询的资源。

数据库服务器除了为决策支持内存和可以并行运行的决策支持查询数量设置限制外，还能利用这些参数在用户提交单个决策支持查询时确定分配给它们的内存量。要执行此操作，数据库服务器会先通过 DS_TOTAL_MEMORY 除以 DS_MAX_QUERIES 来计算称为份额 的内存单位。当用户发出查询时，数据库服务器将分配一定百分比的可用份额，该份额等于查询的 PDQ 优先级。

您也可以通过设置 DS_MAX_SCANS 配置参数限制数据库服务器允许并发决策支持扫描的数量。

先前版本的数据库服务器允许您在 ONCONFIG 文件中设置 PDQ 优先级配置参数。如果您的应用程序依赖于 PDQ 优先级的全局设置，那么您可以使用以下某种方法：

- 对于 *UNIX*™: 将 PDQPRIORITY 定义为 informix.rc 文件中的共享环境变量。有关 informix.rc 文件的更多信息，请参阅《*SinoDB® SQL* 指南: 参考》。
- 对于 *Windows*™: 通过一个登录概要文件，为特定组设置 PDQPRIORITY 环境变量。有关登录概要文件的更多信息，请参阅您的操作系统手册。

监视用于 PDQ 和 DSS 查询的资源

您可以监视内存分配管理器 (MGM) 为 PDQ 查询分配的资源（共享内存和线程）以及 PDQ 与决策支持 (DSS) 查询当前使用的资源。

按照以下方法监视 PDQ 资源使用情况：

- 执行单个 onstat 实用程序命令来获取有关正在运行的查询的特定方面的信息。
- 执行查询之前，执行 SET EXPLAIN 语句可将查询计划写入输出文件。

相关链接

什么是 *PDQ* 在第251页

使用 onstat 实用程序来监视 PDQ 资源

您可以使用各种 onstat 实用程序命令来确定有多少个线程处于活动状态以及这些线程使用的共享内存资源。

您可以使用 onstat -g mgm 命令来监视内存分配管理器 (MGM) 如何协调内存使用以及扫描线程。

相关链接

《*SinoDB* 管理员参考》*: onstat -g mgm* 命令*:* 显示 *MGM* 资源信息

使用 onstat 实用程序命令来监视 PDQ 线程

您可以通过执行 onstat -u 和 onstat -g ath 命令来获得有关正在为决策支持查询运行的所有线程的信息。

onstat -u 选项列出了会话的所有线程。如果会话正在运行决策支持查询，那么其输出将列出主线程和任何附加线程。例如，图 *65: onstat -u* 输出 在第262页中的会话 10 共运行着 5 个线程。

```
Userthreads
address  flags   sessid   user     tty      wait     tout locks nreads   nwrites
80eb8c   ---P--D 0        informix -        0        0    0     33       19
80ef18   ---P--F 0        informix -        0        0    0     0        0
80f2a4   ---P--B 3        informix -        0        0    0     0        0
80f630   ---P--D 0        informix -        0        0    0     0        0
80fd48   ---P--- 45       chrisw   ttyp3    0        0    1     573      237
810460   ------- 10       chrisw   ttyp2    0        0    1     1        0
810b78   ---PR-- 42       chrisw   ttyp3    0        0    1     595      243
810f04   Y------ 10       chrisw   ttyp2    beacf8   0    1     1        0
811290   ---P--- 47       chrisw   ttyp3    0        0    2     585      235
81161c   ---PR-- 46       chrisw   ttyp3    0        0    1     571      239
8119a8   Y------ 10       chrisw   ttyp2    a8a944   0    1     1        0
81244c   ---P--- 43       chrisw   ttyp3    0        0    2     588      230
8127d8   ----R-- 10       chrisw   ttyp2    0        0    1     1        0
812b64   ---P--- 10       chrisw   ttyp2    0        0    1     20       0
812ef0   ---PR-- 44       chrisw   ttyp3    0        0    1     587      227
 15 active, 20 total, 17 maximum concurrent
```

图 65: onstat -u 输出

onstat -g ath 输出也列出了这些线程，并包含指示线程角色的 name 列。由主决策支持线程启动的线程有一个名称，该名称指示它们在决策支持查询中的角色。例如，图 *66: onstat -g ath* 输出 在第262页列出了四个扫描线程，这些线程是由主线程 (sqlexec) 启动的。

```
Threads:
tid     tcb      rstcb    prty    status                 vp-class   name
...
11      994060   0        4       sleeping(Forever)        1cpu    kaio
12      994394   80f2a4   2       sleeping(secs: 51)       1cpu    btclean
26      99b11c   80f630   4       ready                    1cpu    onmode_mon
32      a9a294   812b64   2       ready                    1cpu    sqlexec
113     b72a7c   810b78   2       ready                    1cpu    sqlexec
114     b86c8c   81244c   2       cond wait(netnorm)       1cpu    sqlexec
115     b98a7c   812ef0   2       cond wait(netnorm)       1cpu    sqlexec
116     bb4a24   80fd48   2       cond wait(netnorm)       1cpu    sqlexec
117     bc6a24   81161c   2       cond wait(netnorm)       1cpu    sqlexec
118     bd8a24   811290   2       ready                    1cpu    sqlexec
119     beae88   810f04   2       cond wait(await_MC1)     1cpu    scan_1.0
120     a8ab48   8127d8   2       ready                    1cpu    scan_2.0
121     a96850   810460   2       ready                    1cpu    scan_2.1
122     ab6f30   8119a8   2       running                  1cpu    scan_2.2
```

图 66: onstat -g ath 输出

监视为运行 DSS 查询的会话分配的资源

通过执行 onstat -g ses 命令可监视分配给正在运行决策支持 (DSS) 查询的会话并由其使用的资源。

onstat -g ses 选项显示以下信息：

- 为正在运行决策支持查询的会话分配的共享内存
- 正在运行决策支持查询的会话使用的共享内存

- 数据库服务器为会话启动的线程数

例如，在 图 67: onstat -g ses 输出 在第263页中，编号为 49 的会话正在为决策支持查询运行 5 个线程。

```
session                                          #RSAM    total      used
id       user     tty      pid    hostname threads  memory     memory
57       informix -        0      -        0        8192       5908
56       user_3   ttyp3    2318   host_10  1        65536      62404
55       user_3   ttyp3    2316   host_10  1        65536      62416
54       user_3   ttyp3    2320   host_10  1        65536      62416
53       user_3   ttyp3    2317   host_10  1        65536      62416
52       user_3   ttyp3    2319   host_10  1        65536      62416
51       user_3   ttyp3    2321   host_10  1        65536      62416
49       user_1   ttyp2    2308   host_10  5        188416     178936
2        informix -        0      -        0        8192       6780
1        informix -        0      -        0        8192       4796
```

图 67: onstat -g ses 输出

标识 SET EXPLAIN 输出中的并行扫描

打开 PDQ 时，SET EXPLAIN 输出将显示优化器是否选择了并行扫描。如果优化器选择了并行扫描，那么输出将列出 Parallel。（如果关闭了 PDQ，那么输出将列出 Serial。）

如果打开 PDQ，那么优化器也会指示响应查询所需的最大线程数。SET EXPLAIN 输出中的 # of Secondary Threads 字段指示除了用户会话线程之外所需的线程数。必需的线程总数为辅助线程数加上 1。

以下示例显示了一个表的 SET EXPLAIN 输出，该表采用分段存储并将 PDQ 优先级设置为 LOW：

```
SELECT * FROM t1 WHERE c1 > 20

Estimated Cost: 2
Estimated # of Rows Returned: 2

1) informix.t1: SEQUENTIAL SCAN (Parallel, fragments: 2)

 Filters: informix.t1.c1 > 20

# of Secondary Threads = 1
```

以下 SET EXPLAIN 部分输出示例显示了一个查询，该查询在两个分段表之间具有散列连接，并且将 PDQ 优先级设置为 ON。该查询使用 DYNAMIC HASH JOIN 标记，且建立散列的表使用 Build Outer 进行标记。

```
QUERY:
------
SELECT h1.c1, h2.c1 FROM h1, h2 WHERE h1.c1 = h2.c1

Estimated Cost: 2
Estimated # of Rows Returned: 5

1) informix.h1: SEQUENTIAL SCAN (Parallel, fragments: ALL)

2) informix.h2: SEQUENTIAL SCAN (Parallel, fragments: ALL)


DYNAMIC HASH JOIN (Build Outer)
 Dynamic Hash Filters: informix.h1.c1 = informix.h2.c1

# of Secondary Threads = 6
```

以下 SET EXPLAIN 部分输出的示例显示了具有分段存储的表，其 PDQ 优先级设置为 LOW，并且也显示了已选为存取计划的索引：

```
SELECT * FROM t1 WHERE c1 < 13

Estimated Cost: 2
Estimated # of Rows Returned: 1

1) informix.t1: INDEX PATH

 (1) Index Keys: c1 (Parallel, fragments: ALL)
 Upper Index Filter: informix.t1.c1 < 13


# of Secondary Threads = 3
```

第

13

章

提高个别查询性能

可以测试、监视和改进查询。

相关链接

《*SinoDB* 迁移指南》：调整新版本的性能和调整查询

使用专用测试系统来测试查询

可以在不干扰生产数据库服务器的系统上测试查询。然而，需要注意的是，在一个独立系统上的查询测试可能会误导您做调优时的决策。

即便是将数据库服务器当成一个数据仓库，有时候您也可以在独立系统上测试查询，以便理解和查询相关的调优问题。

如果想要提高一个耗时几分钟甚至几小时的大型查询的性能，您可以准备一个缩小版的数据库来更快地完成测试。但是，请注意以下这些潜在问题：

- 优化器在小型数据库和大型数据库上所做的选择可能是不同的，即使表的相对大小相同。请验证查询计划在实际数据库和模型数据库中是否相同。
- 执行时间和表的大小通常都不是线性相关的。例如，排序时间的增长比表大小的增长要快，当索引从两层增加到三层时，索引访问的成本也会增加许多。在按比例缩小的环境中查询性能有很大提高，但这种提高应用于整个数据库时可能并不明显。

因此，在模型数据库的测试结果得出的任何结论都只是试验性的，直到在生产数据库中得到验证为止。

您通常可以调整查询或数据模型来提高性能，调整时注意以下目标：

- 如果使用的是多用户系统或网络，系统负载随时都会发生很大变化，那么应在每天的同一时刻进行实验，以获得可重复性结果。在系统负载持续较轻时启动测试，以确保测量的只是查询的影响。
- 如果将查询嵌入复杂的程序，那么您可以抽取 SELECT 语句，并将其嵌入 DB-Access 脚本。

相关链接

《*SinoDB* 迁移指南》：调整新版本的性能和调整查询

显示查询计划

在更改查询之前，可以通过显示其查询计划以确定查询所需的资源种类和数量。查询计划可以显示使用了哪些并行扫描、所需的最大线程数以及使用的索引。

研究查询计划后，请检查数据模型以确定本章建议的更改是否会改善查询。

您可使用以下其中一种方法显示查询计划：

- 进行查询之前执行以下某个 SET EXPLAIN 语句：
  - SET EXPLAIN ON

该 SQL 语句显示选择的查询计划。有关 SET EXPLAIN ON 输出的描述，请参阅显示由优化器选择的查询计划的报告 在第216页。

- SET EXPLAIN ON AVOID_EXECUTE

  该 SQL 语句显示选择的查询计划，但并不执行查询。

- 在查询中使用以下某个 EXPLAIN 指令：
  - EXPLAIN
  - EXPLAIN AVOID_EXECUTE

有关这些 EXPLAIN 指令的更多信息，请参阅 *EXPLAIN* 指令 在第243页。

提高过滤器选择性

您可以控制查询输出的信息量。指定所需的目标行时的精确度越高，快速完成查询的可能性就越大。

要控制查询输出的信息量，使用 SELECT 语句的 WHERE 子句。WHERE 子句中的条件表达式通常被称为过滤器。

有关过滤器选择性如何影响优化器选择查询计划的信息，请参阅查询中的过滤器 在第225页。以下几节提供了一些提高过滤器选择性的参考。

使用用户定义例程的过滤器

可以提高包含用户定义例程 (UDR) 的查询过滤器的选择性。

如果 UDR 具有以下特性，那么可以提高选择性：

- 函数索引

  您可以基于用户定义例程的结果值，或基于对一个或多个列进行运算的内置函数的结果值来创建函数索引。创建函数索引时，数据库服务器会计算函数的返回值，并将其存储于索引中。数据库服务器可在适当的索引中找到函数的返回值，而无需对每个符合条件的列执行函数。

  有关对用户定义的函数建立索引的更多信息，请参阅使用函数索引 在第169页。

- 用户定义的选择性函数

  您可以编写一个函数，计算符合函数要求的行的期望百分比。有关用户定义的选择性函数的简短描述，请参阅选择性和成本函数 在第294页。有关如何编写和注册用户定义的选择性函数的更多信息，请参阅《*SinoDB*® 用户自定义例程和数据类型开发者指南》。

避免使用某些过滤器

为了实现最佳性能，请避免使用带有复杂正则表达式的过滤器以及避免使用带非初始化字符串的过滤器。

避免使用复杂的正则表达式
MATCHES 和 LIKE 关键字支持通配符匹配，这在技术上称为正则表达式。对于数据库服务器而言，某些正则表达式比起其他的正则表达式执行起来要更困难。

如以下示例（查找名字不以 *y* 结尾的客户）所示，通配符位于起始位置处，这使数据库服务器必须检查列中的每个值：

```
SELECT * FROM customer WHERE fname NOT LIKE '%y'
```

您不能将索引与这样的过滤器一起使用，因为这样就必须顺序访问此示例中的表。

如果对复杂正则表达式的测试是必需的，那么应避免将其与连接结合使用。如果有必要，处理单个表并应用正则表达式的测试来选择所需的行。将结果保存在临时表中，并将该临时表与其他表连接。

对于通配符位于操作数中间或末尾的正则表达式测试，那么能够使用索引（如果存在索引的话）。

避免使用非初始子串
为了实现最佳性能，避免使用带非初始化字符串的过滤器。基于列的非初始子串的过滤器要求数据库服务器测试列中的每个值。

例如，在以下代码中，非初始子串使数据库服务器测试列中的每个值：

```
SELECT * FROM customer
   WHERE zipcode[4,5] > '50'
```

数据库服务器无法使用索引来评估这样的过滤器。

对于测试索引列的初始子字符串的过滤器，优化器使用索引进行处理。但是，进行子字符串测试会干扰使用组合索引来测试子字符串列和其他列。

使用连接过滤器和连接后过滤器

数据库服务器提供对使用 ANSI join 语法的子集的支持。

此语法包括以下关键字：

- ON 关键字，用来指定连接条件和任何可选连接过滤器
- LEFT OUTER JOIN 关键字，用来指定哪个表是主表（也称为外表）

有关此 ANSI join 语法的更多信息，请参阅《*SinoDB® SQL* 指南*:* 语法》。

在 ANSI 外连接中，数据库服务器会执行以下操作来处理过滤器：

- 在 ON 子句中应用连接条件，以确定从表（也称为内表）的哪些行连接到外表
- 连接前和连接过程中在 ON 子句中应用可选的连接过滤器

  如果在 ON 子句中对基础内表指定了连接过滤器，那么数据库服务器可在连接前和扫描内表数据的过程中应用该过滤器。ON 子句中基础从表的过滤器可提供以下附加性能优势：

  - 连接前扫描内表的行较少
  - 连接前从内表中检索行时使用了索引
  - 要连接的行较少
  - WHERE 子句中的过滤器要评估的行较少

  有关在 ON 子句中对外表指定连接过滤器时所发生情况的信息，请参阅《*SinoDB® SQL* 指南*:* 语法》。
- 连接后应用 WHERE 子句中的过滤器

  WHERE 子句中的过滤器可以减少数据库服务器需要扫描的行数，并减少返回给用户的行数。

  连接后过滤器就是指这种 WHERE 子句过滤器。

执行使用符合 ANSI 的 LEFT OUTER 语法来指定连接表和嵌套循环连接的分布查询时，将查询发送到每个参与的数据库服务器以对这些服务器的本地表执行操作。

例如，演示数据库中的 customer 表和 cust_calls 表可以跟踪客户呼叫服务部门的记录。假设某个呼叫代码在过去出现多次，而您想要了解此类呼叫是否减少。要了解客户是否不再进行该呼叫代码的呼叫，可使用外连接列出所有客户。

图 *68: ANSI join* 的 *SET EXPLAIN ON* 输出 在第267页显示了 SQL 语句示例，该语句完成此 ANSI join 查询及其 SET EXPLAIN ON 输出。

```
QUERY:
------
SELECT c.customer_num, c.lname, c.company,
c.phone, u.call_dtime, u.call_code, u.call_descr
FROM customer c
LEFT JOIN cust_calls u ON c.customer_num = u.customer_num
ORDER BY u.call_dtime

Estimated Cost: 14
Estimated # of Rows Returned: 29
```

```
Temporary Files Required For: Order By

1) virginia.c: SEQUENTIAL SCAN

2) virginia.u: INDEX PATH

   (1) Index Keys: customer_num call_dtime (Serial, fragments: ALL)
     Lower Index Filter: virginia.c.customer_num = virginia.u.customer_num

  ON-Filters:virginia.c.customer_num = virginia.u.customer_num
  NESTED LOOP JOIN(LEFT OUTER JOIN)
```

图 68: ANSI join 的 SET EXPLAIN ON 输出

请查看 图 *68: ANSI join* 的 *SET EXPLAIN ON* 输出 在第267页的 SET EXPLAIN ON 输出的以下行：

- ON-Filters: 行列出了在 ON 子句中指定的连接条件。
- 尽管该查询在 FROM 子句中只指定了 LEFT JOIN 关键字，但是 SET EXPLAIN ON 输出的最后一行显示了 ANSI join 的所有三个关键字 (LEFT OUTER JOIN)。OUTER 关键字是可选的。

图 *69: ANSI join* 中连接过滤器的 *SET EXPLAIN ON* 输出 在第268页显示了 ANSI join 的 SET EXPLAIN ON 输出，其中连接过滤器会检查 call_code 为 I 的呼叫。

```
QUERY:
------
SELECT c.customer_num, c.lname, c.company,
c.phone, u.call_dtime, u.call_code, u.call_descr
FROM customer c LEFT JOIN cust_calls u
ON c.customer_num = u.customer_num
AND u.call_code = 'I'
ORDER BY u.call_dtime

Estimated Cost: 13
Estimated # of Rows Returned: 25
Temporary Files Required For: Order By

  1) virginia.c: SEQUENTIAL SCAN

  2) virginia.u: INDEX PATH

     Filters: virginia.u.call_code = 'I'

      (1) Index Keys: customer_num call_dtime   (Serial, fragments: ALL)
          Lower Index Filter: virginia.c.customer_num = virginia.u.customer_num

ON-Filters:(virginia.c.customer_num = virginia.u.customer_num
               AND virginia.u.call_code = 'I' )
NESTED LOOP JOIN(LEFT OUTER JOIN)
```

图 69: ANSI join 中连接过滤器的 SET EXPLAIN ON 输出

图 *68: ANSI join* 的 *SET EXPLAIN ON* 输出 在第267页和 图 *69: ANSI join* 中连接过滤器的 *SET EXPLAIN ON* 输出 在第268页中的输出之间的主要区别如下：

- 优化器选择不同的索引来扫描内表。

  该新索引使用了较多的过滤器，并检索较少的行。因此，进行连接操作的行也较少。
- ON 子句连接过滤器包含附加过滤器。

Estimated # of Rows Returned 行中的值只是估算值，并不总是反映实际返回的行数。由于有附加过滤器，图 *69: ANSI join* 中连接过滤器的 *SET EXPLAIN ON* 输出 在第268页中的查询示例返回的行比 图 *68: ANSI join* 的 *SET EXPLAIN ON* 输出 在第267页中的查询返回的要少。

图 *70: ANSI join* 中 *WHERE* 子句过滤器的 *SET EXPLAIN ON* 输出 在第269页显示 ANSI join 查询的 SET EXPLAIN ON 输出，该连接查询在 WHERE 子句中只有一个过滤器。

```
QUERY:
------
SELECT c.customer_num, c.lname, c.company,
   c.phone, u.call_dtime, u.call_code, u.call_descr
FROM customer c LEFT JOIN cust_calls u
ON c.customer_num = u.customer_num
   AND u.call_code = 'I'
WHERE c.zipcode = "94040"
ORDER BY u.call_dtime

Estimated Cost: 3
Estimated # of Rows Returned: 1
Temporary Files Required For: Order By

  1) virginia.c: INDEX PATH

    (1) Index Keys: zipcode   (Serial, fragments: ALL)
        Lower Index Filter: virginia.c.zipcode = '94040'

  2) virginia.u: INDEX PATH

     Filters: virginia.u.call_code = 'I'

    (1) Index Keys: customer_num call_dtime   (Serial, fragments: ALL)
        Lower Index Filter: virginia.c.customer_num = virginia.u.customer_num

ON-Filters:(virginia.c.customer_num = virginia.u.customer_num
      AND virginia.u.call_code = 'I' )
NESTED LOOP JOIN(LEFT OUTER JOIN)

PostJoin-Filters:virginia.c.zipcode = '94040'
```

图 70: ANSI join 中 WHERE 子句过滤器的 SET EXPLAIN ON 输出

图 *69: ANSI join* 中连接过滤器的 *SET EXPLAIN ON* 输出 在第268页和 图 *70: ANSI join* 中 *WHERE* 子句过滤器的 *SET EXPLAIN ON* 输出 在第269页中的输出之间的主要区别如下：

- 为主表选择了连接后过滤器中的 zipcode 列上的索引。
- PostJoin-Filters 行显示 WHERE 子句中的过滤器。

自动统计信息更新

数据库服务器根据预定义的调度和一组到期策略来自动更新统计信息。Auto Update Statistics (AUS) 维护系统会标识需要新优化器统计信息的表和索引，并执行相应的 UPDATE STATISTICS 语句以优化查询性能。

AUS 维护系统可更新使用日志记录数据库中的表的统计信息，而不考虑数据库的语言环境。通过向查询优化器提供当前的表统计信息，AUS 维护系统可以减少低效查询导致性能下降的风险。

您可能需要根据系统来调整 AUS 到期策略或计划。AUS 维护系统驻留在 sysadmin 数据库中。

您也可以查看和调整 SinoDB® 开放管理工具 (OAT) 中表的 AUS 维护系统统计信息。

相关链接

未自动生成统计信息时更新统计信息 在第274页
未自动生成统计信息时更新统计信息 在第274页

AUS 的工作方式

Auto Update Statistics (AUS) 维护系统使用调度程序传感器、任务、阈值和表的组合来评估和更新统计信息。

调度程序任务、传感器、阈值和表驻留在 sysadmin 数据库中。缺省情况下，只有用户 informix 有权访问 sysadmin 数据库。

以下事件顺序描述了如何自动更新统计信息：

1. 调度程序的 mon_table_profile 传感器每天运行，以从 sysmaster 数据库中的 systables 表读取数据。传感器使用有关每张表的已更改内容的信息更新 sysadmin 数据库中的 mon_table_profile 表。
2. Auto Update Statistics Evaluation 任务每天从 mon_table_profile 表和 sysmaster 数据库中的 systable、sysdistrib、syscolumns 和 sysindices 表收集信息。
3. Auto Update Statistics Evaluation 任务可根据到期策略确定哪些表需要更新。
4. Auto Update Statistics Evaluation 任务生成 UPDATE STATISTICS 语句，并将其插入 sysadmin 数据库中的 aus_command 表。
5. Auto Update Statistics Refresh 任务在每周六和周日的早上 1 点到 5 点之间执行从 aus_command 表得到的 UPDATE STATISTICS 语句，并将结果插回 aus_command 表。在早上 5 点前未完成的任何 UPDATE STATISTICS 语句都将继续保留在 aus_command 表中。

下表描述了 sysadmin 数据库中组成 AUS 维护系统的任务、传感器、阈值、表和视图。

表 15: AUS 组件

组件	类型	描述
mon_table_profile	传感器	编译表概要文件信息，包括每张表上发生的更新、插入和删除总数。 它在 ph_task 表中定义。
mon_table_profile	表	存储其传感器收集的表概要文件信息。其他许多调度程序任务都使用此表的信息。
Auto Update Statistics Evaluation	任务	根据到期策略标识具有旧文件统计信息的表，并为这些表生成 UPDATE STATISTICS 语句。 它在 ph_task 表中定义。
aus_command	表	存储已经排好优先执行顺序的 UPDATE STATISTICS 语句及其执行后的状态。 aus_cmd_state 列指示每个 UPDATE STATISTICS 语句的状态： • P = 暂挂 • I = 进行中 • E = 错误 • C = 无错完成 如果命令状态为 E，那么关联的 SQL 错误代码将列于 aus_cmd_err_sql 列中，而关联的 ISAM 错误代码将列于 aus_cmd_err_isam 列中。 aus_cmd_runtime 显示更新统计信息命令完成所用的时间。aus_cmd_time 显示更新统计信息命令的开始时间。
Auto Update Statistics Refresh	任务	在每周六和周日的早上 1 点到 5 点之间执行 UPDATE STATISTICS 语句。 它在 ph_task 表中定义。

组件	类型	描述
到期策略	阈值	定义何时更新统计信息的规则。 它在 ph_threshold 表中定义。
aus_cmd_comp	视图	显示 aus_command 表已成功运行的 UPDATE STATISTICS 语句的信息。
aus_cmd_list	视图	显示 aus_command 表中已安排要运行的 UPDATE STATISTICS 语句的信息。

有关调度程序的其他功能部件的信息，请参阅《*SinoDB*® 管理员指南》中的描述。有关 sysadmin 数据库的信息，请参阅《*SinoDB*® 管理员参考》。

AUS 到期策略

Auto Update Statistics (AUS) 维护系统将到期策略当做准则，以确定用户表的修改已达到需要重新计算来更新其统计信息的程度。

在内部，AUS 维护系统会自动跳过具有当前统计信息的任何表或分段，并依优先顺序处理具有更多更改的表或分段。因此，所有表都安排用于更新统计信息。

sysadmin 数据库的 ph_threshold 表存储用于定义 AUS 到期策略的以下可配置的阈值。

表 16: AUS 到期策略阈值

阈值名称	缺省值	描述
AUS_AGE	30（天）	基于时间的到期策略。在此时间量过后将更新表的统计信息或分布，而不考虑更改了多少数据。
AUS_AUTO_RULES	1（启用）	如果已启用，那么将使用以下缺省最低准则或用户创建的分布选项中的较高者更新统计信息： • 以 LOW 方式更新所有表。 • 以 HIGH 方式更新所有行距索引键。 • 以 MEDIUM 方式更新所有非行距索引键。 • MEDIUM 方式的最低分辨率为 2.0。 • MEDIUM 方式的最低置信度为 0.95。 • HIGH 方式的最低分辨率为 0.5。 如果已对表手动执行了 UPDATE STATISTICS 语句，那么 AUS 维护系统生成的 UPDATE STATISTICS 语句将不会降低级别、分辨率、置信度或采样大小选项。 如果通过设置为 0 禁用，那么 AUS 维护系统会检查哪些列具有现有分布，并生成更新统计信息语句以刷新它们。
AUS_PDQ	10（优先级）	AUS 维护系统运行的 UPDATE STATISTICS 语句的 PDQ 优先级。缺省情况下，将并行分析每个表的所有分段。有关 PDQ 优先级的更多信息，请参阅在超大型数据库上并行更新统计信息 在第278页。
AUS_SMALL_TABLES	100（行）	对于小于此行数的表每次都将更新统计信息或分布。

更改 AUS 到期策略
您可以更改 AUS 到期策略，以根据统计信息的生存期、更改的数据量或表的大小来定制统计信息的更新频率。

必须以用户 informix 或其他授权用户身份连接 sysadmin 数据库。

要更改到期策略的值，请更新 sysadmin 数据库中 ph_threshold 表内的 value 列。

例如，如果您发现对于那些小于等于 1000 行的小型表的查询在更新其统计信息更频繁时运行更快，那么您可以更改到期策略以确保每周更新其统计信息。以下示例将 AUS_SMALL_TABLES 阈值更改为 1000：

```
UPDATE ph_threshold
SET value = 1000
WHERE name = "AUS_SMALL_TABLES";
```

新的阈值将在下次执行 Auto Update Statistics Evaluator 任务时生效。

查看 AUS 语句

您可以在 AUS 维护系统生成的 UPDATE STATISTICS 语句运行前，在 aus_cmd_list 视图中查看这些语句，并在该语句成功运行后在 aus_cmd_comp 视图中查看这些语句。这两张表都在 sysadmin 数据库中。

必须以用户 informix 或其他授权用户身份连接 sysadmin 数据库。

要查看所有调度的 UPDATE STATISTICS 语句，请执行以下语句：

```
SELECT * FROM aus_cmd_list;
```

要查看过去 30 天成功运行的所有 UPDATE STATISTICS 语句，请执行以下语句：

```
SELECT * FROM aus_cmd_comp;
```

要查看所有失败的 UPDATE STATISTICS 语句，请执行以下语句：

```
SELECT aus_cmd_exe, aus_cmd_err_sql, aus_cmd_err_isam
FROM aus_command
WHERE aus_cmd_state = "E";
```

您也可以在 SinoDB® 开放管理工具 (OAT) 中查看此信息。

在 AUS 中指定数据库的优先级

您可以在 AUS 维护系统中为每个数据库指定优先级。

缺省情况下，所有数据库都具有中等优先级。您可以为特定数据库指定高或低的优先级，以确保最重要数据库的统计信息可以最先更新。如果时间和资源允许的情况下，那么低优先级数据库的统计信息会在高优先级和中优先级数据库之后更新。例如，如果系统具有生产和测试数据库，那么可以为生产数据库指定高优先级，为测试数据库指定低优先级。您也可以为数据库禁用 AUS。

您可以在 SinoDB® 开放管理工具 (OAT) 中指定数据库的优先级。

必须以用户 informix 或其他授权用户身份连接 sysadmin 数据库。

要在 AUS 中为数据库指定优先级，请添加一行到 sysadmin 数据库中的 ph_threshold 表：

- 高优先级：请添加 name 列设为 AUS_DATABASE_HIGH 且 value 列设为数据库名称的一行。
- 低优先级：请添加 name 列设为 AUS_DATABASE_LOW 且 value 列设为数据库名称的一行。
- 禁用：请添加 name 列设为 AUS_DATABASE_DISABLE 且 value 列设为数据库名称的一行。

如果您为数据库指定多个优先级，那么优先考虑较高的优先级。

示例

以下语句将名为 my_database 的数据库的优先级设为高优先级：

```
INSERT INTO ph_threshold(id, name, task_name, value, value_type, description)
     VALUES(0,
            "AUS_DATABASE_HIGH",
            "Auto Update Statistics Evaluation",
            "my_database",
            "STRING",
            "Rank this database as high priority to get its tables done
 first");
```

重新调度 AUS

您可以更改 Auto Update Statistics Refresh 任务运行的时间和时长。

更新统计信息是资源密集型的操作。因此，缺省情况下，将在每周六和每周日的早上 1 点到 5 点之间自动更新统计信息。如果您发现在此时间段无法执行所有暂挂的 UPDATE STATISTICS 语句，或您希望更频繁地刷新统计信息，那么可以更改开始时间、结束时间以及执行此任务的每周的天数。

必须以用户 informix 或其他授权用户身份连接 sysadmin 数据库。

要更改 Auto Update Statistics Refresh 任务的调度，请更新 ph_task 表，其中 tk_name 列的值为 Auto Update Statistics Refresh。

以下示例将任务的结束时间更改为早上 6 点：

```
UPDATE ph_task
SET tk_stop_time = "06:00:00"
WHERE tk_name = "Auto Update Statistics Refresh";
```

以下示例将任务的运行天数更改为一周的每一天（缺省情况下启用周六和周日）：

```
UPDATE ph_task
SET tk_monday = "T",
tk_tuesday = "T",
tk_wednesday = "T",
tk_thursday = "T",
tk_friday = "T"
WHERE tk_name = "Auto Update Statistics Refresh";
```

禁用 AUS

您可以通过禁用 AUS 维护系统阻止自动更新统计信息。

必须以用户 informix 或其他授权用户身份连接 sysadmin 数据库。

要禁用 AUS，必须禁用 Auto Update Statistics Evaluation 任务和 Auto Update Statistics Refresh 任务：

1. 将 ph_task 表的 tk_enable 列的值更新为 F，其中 tk_name 列的值为 Auto Update Statistics Evaluation。
2. 将 ph_task 表的 tk_enable 列的值更新为 F，其中 tk_name 列的值为 Auto Update Statistics Refresh。

以下示例禁用了这两种任务：

```
UPDATE ph_task
```

```
SET tk_enable = "F"
WHERE tk_name = "Auto Update Statistics Evaluation";

UPDATE ph_task
SET tk_enable = "F"
WHERE tk_name = "Auto Update Statistics Refresh";
```

未自动生成统计信息时更新统计信息

UPDATE STATISTICS 语句会更新系统目录表中优化器使用的统计信息以确定成本最低的查询计划。

重要： 自动生成统计信息时，您无需执行 UPDATE STATISTICS 操作。

以下统计信息由 CREATE INDEX 语句（带有或不带 ONLINE 关键字）自动生成：

- 索引级别的统计信息，等同于以 LOW 方式在 UPDATE STATISTICS 操作中为 B 型树索引收集的统计信息。
- 对于一般 B 型树索引的透明主索引列，列分布统计信息等同于以 HIGH 方式在 UPDATE STATISTICS 操作中生成的分布。

为了确保优化器选择的查询计划最能反映表的当前状态，在不能自动生成统计信息时应定期执行 UPDATE STATISTICS。

提示： 如果在使用 ON-Bar 之前对 sysutils 数据库执行 UPDATE STATISTICS LOW，那么 ON-BAR 处理需要的时间将减少。

下表总结了当没有自动生成统计信息时，何时执行不同的 UPDATE STATISTICS 语句。如果需要执行 UPDATE STATISTICS 语句并且有很多表，那么可以编写一个脚本生成这些 UPDATE STATISTICS 语句。

何时执行	UPDATE STATISTICS 语句	详细信息和示例的参考
行数已发生显著变化	UPDATE STATISTICS LOW DROP DISTRIBUTIONS	更新行数的统计信息 在第275页或升级时按需要删除数据分布 在第275页
对于任何索引的所有列（除前导列以外）	UPDATE STATISTICS LOW	创建数据分布 在第275页
查询包含非索引连接列或过滤器列	UPDATE STATISTICS MEDIUM DISTRIBUTIONS ONLY	创建数据分布 在第275页
查询包含索引连接列或过滤器列	UPDATE STATISTICS HIGH 表（索引中的前导列）	创建数据分布 在第275页
查询包含多列索引，该索引定义于连接列或过滤器列	UPDATE STATISTICS HIGH 表（多列索引中第一个不同的列）	创建数据分布 在第275页
查询包含多列索引，该索引定义于连接列或过滤器列	UPDATE STATISTICS LOW 表（多列索引中的所有列）	创建数据分布 在第275页
查询包含许多小表（符合一个扩展数据块）	小表上的 UPDATE STATISTICS HIGH	创建数据分布 在第275页
查询使用 SPL 例程	UPDATE STATISTICS FOR PROCEDURE	重新优化 *SPL* 例程 在第233页

有关数据库服务器在系统目录表中保留的特定统计信息的信息，请参阅为表和索引保留的统计信息 在第224页。

相关链接

自动统计信息更新 在第269页

《SinoDB SQL 指南: 语法》: UPDATE STATISTICS 语句
自动统计信息更新 在第269页

更新行数的统计信息

执行 UPDATE STATISTICS LOW 时，数据库服务器将更新系统目录表中表、行和页数的统计信息。您应该按需要经常执行 UPDATE STATISTICS LOW，以确保行数的统计信息尽可能是最新的。

如果某个表的基数经常发生更改，那么应更频繁地对该表执行此语句。

LOW 是 UPDATE STATISTICS 的缺省方式。

以下 SQL 语句示例会更新 systables、syscolumns 和 sysindexes 系统目录表中的统计信息，但不更新数据分布。

```
UPDATE STATISTICS FOR TABLE tab1;
```

升级时按需要删除数据分布

升级到新版本的数据库服务器时，可能需要删除分布信息，以移除 sysdistrib 系统目录表中旧的分布结构。

要删除 sysdistrib 系统目录表中旧的分布结构，请执行此语句：

```
UPDATE STATISTICS DROP DISTRIBUTIONS;
```

以 LOW 方式删除分布（不收集统计信息）
您可以从 sysdistrib 表中移除分布信息；对于那些分布已删除的表，可在其系统目录中更新 systables.version 列，而无需收集任何 LOW 方式表和索引的统计信息。

您可以使用 UPDATE STATISTICS 语句中的 DROP DISTRIBUTIONS ONLY 选项来执行该操作。使用 DROP DISTRIBUTIONS ONLY 选项可以提高性能，因为在 ONLY 关键字没有跟在 DROP DISTRIBUTIONS 关键字后面时，数据库服务器不会收集由 LOW 方式选项生成的表和索引统计信息。

有关如何使用 DROP DISTRIBUTIONS ONLY 选项的详细信息，请参阅《*SinoDB® SQL* 指南: 语法》。

创建数据分布

您可以为某个表生成统计信息，也可以为查询访问的每个表构建数据分布。

（当自动生成统计信息时不需要执行 UPDATE STATISTICS 语句。）

只要执行 UPDATE STATISTICS MEDIUM 或 UPDATE STATISTICS HIGH 命令，数据库服务器就会创建向优化器提供信息的数据分布。

重要:

数据库服务器通过对列的数据进行取样、排序数据、构建分布容器和将结果插入 sysdistrib 系统目录表中来创建数据分布。

您可以通过关键字 HIGH 或 MEDIUM 控制扫描的样本大小。UPDATE STATISTICS HIGH 和 UPDATE STATISTICS MEDIUM 之间的差异在于取样的行数。根据 UPDATE STATISTICS 语句使用的置信度和分辨率，UPDATE STATISTICS HIGH 会扫描整个表，而 UPDATE STATISTICS MEDIUM 仅对一部分行采样。

对于标准索引键，您可以将 LOW 关键字和 UPDATE STATISTICS 语句一起使用。

如果已经为某一列生成了分布，那么优化器会使用该信息来估算与针对某列的查询相匹配的行数。当优化器估算返回的行数时，sysdistrib 中的数据分布会替代 syscolumns 系统目录表的 colmin 和 colmax 列中的值。

第一次使用数据分布统计信息时，尝试以 MEDIUM 方式更新所有表的统计信息，然后以 HIGH 方式更新索引所有起始列的统计信息。此策略将为您指定的列生成统计查询估算值。这些估算值平均误差范围小于表中总行数的 *percent*，其中 *percent* 是使用 MEDIUM 方式时在 RESOLUTION 子句中指定的值。MEDIUM 方式的缺省百分比值是 2.5%。（对于 HIGH 方式分布的列，缺省分辨率为 0.5%。）

使用 DISTRIBUTIONS ONLY 选项，可以在表级别上或对整个系统执行 UPDATE STATISTICS MEDIUM，因为额外列的开销并不大。

只有当使用 UPDATE STATISTICS 的 HIGH 选项时，数据库服务器才使用 DBSPACETEMP 环境变量指定的存储位置。

将 DBUPSPACE 环境变量的第三个参数设置为值 1，以防止 UPDATE STATISTICS 操作在对行进行排序时使用索引。

对于查询访问的每个表，应根据以下准则构建数据分布。另见准则下的示例。

要生成有关表的统计信息：

1. 标识出现在表的任何单列或多列索引中的所有列的集合。
2. 标识包括任何索引的所有列（前导列除外）的子集。
3. 对该子集中的每一列执行 UPDATE STATISTICS LOW。

要为查询访问的每个表构建数据分布：

1. 为表中所有不是索引起始列的列执行单个 UPDATE STATISTICS MEDIUM。

   如果表不是非常大，那么可以使用缺省参数，在此情况下，您应使用 1.0 的分辨率和 0.99 的置信度。
2. 执行以下 UPDATE STATISTICS 语句可创建非索引连接列和非索引过滤器列的分布：

```
UPDATE STATISTICS MEDIUM DISTRIBUTIONS ONLY;
```

3. 为所有索引起始列执行 UPDATE STATISTICS HIGH。为了使 UPDATE STATISTICS 语句的执行时间最短，您必须对每一列执行一条 UPDATE STATISTICS HIGH 语句，如该过程下的示例所示。
4. 如果您具有以相同列的子集开头的索引，那么对每个不同的索引中的第一列执行 UPDATE STATISTICS HIGH，如该过程下的第二个示例所示。
5. 对于表的每个单列或多列索引：

   a. 标识出现在索引中的所有列的集合。
   b. 标识包括任何索引的所有列（前导列除外）的子集。
   c. 对该子集中的每一列执行 UPDATE STATISTICS LOW。（缺省值为 LOW。）
6. 对于在步骤 3 中创建索引的表，执行以下 UPDATE STATISTICS 语句，以更新 sysindexes 和 syscolumns 系统目录表，示例如下：

```
UPDATE STATISTICS FOR TABLE t1(a,b,e,f);
```

7. 对于小型表，执行 UPDATE STATISTICS HIGH，例如：

```
UPDATE STATISTICS HIGH FOR TABLE t2;
```

由于该语句只对每条索引构造一次统计信息，因而这些步骤确保 UPDATE STATISTICS 能够快速执行。

示例

后跟索引的所有列的 UPDATE STATISTICS HIGH 语句示例

假设您具有一个表 t1，其中列 a、b、c、d、e 和 f 具有以下索引：

```
CREATE INDEX ix_1 ON t1 (a, b, c, d) ...
CREATE INDEX ix_3 ON t1 (f) ...
```

对索引的起始列执行以下 UPDATE STATISTICS 语句：

```
UPDATE STATISTICS HIGH FOR TABLE t1(a);
UPDATE STATISTICS HIGH FOR TABLE t1(f);
```

这些 UPDATE STATISTICS HIGH 语句对索引列使用高分布来替换通过 UPDATE STATISTICS MEDIUM 语句创建的分布。These UPDATE STATISTICS HIGH statements replace the

distributions created with the UPDATE STATISTICS MEDIUM statements with high distributions for index columns.

每个不同索引中第一列的 UPDATE STATISTICS HIGH 语句示例：

例如，假设您在表 t1 上具有以下索引：

```
CREATE INDEX ix_1 ON t1 (a, b, c, d) ...
CREATE INDEX ix_2 ON t1 (a, b, e, f) ...
CREATE INDEX ix_3 ON t1 (f) ...
```

步骤 *3* 在第276页 对列 a 和列 f 执行 UPDATE STATISTICS HIGH。然后对列 c 和 e 执行 UPDATE STATISTICS HIGH。

```
UPDATE STATISTICS HIGH FOR TABLE t1(c);
UPDATE STATISTICS HIGH FOR TABLE t1(e);
```

此外，您可以对列 b 执行 UPDATE STATISTICS HIGH，但是该步骤通常不是必须的。

相关链接

共享内存的虚拟部分 在第56页

《*SinoDB SQL* 指南: 语法》: *UPDATE STATISTICS* 语句

更新用于连接列的统计信息

在某些情况下，您可能希望对特定连接列执行带有 HIGH 关键字的 UPDATE STATISTICS 语句。

为建立更好的查询计划而进行性能提高和经调整的成本估算，在某些情况下，优化器很大程度上依赖于对基础数据分布的精确了解。即使已遵循创建数据分布 在第275页中的准则，您可能仍然觉得复杂查询执行得不够快。如果您的查询包含等式谓词，请执行以下其中一种操作：

- 对于在查询的 WHERE 子句中出现的特定连接列，将 UPDATE STATISTICS 语句与 HIGH 关键字一起执行。如果遵循了创建数据分布 在第275页中的准则，那么索引起始列已具有 HIGH 方式分布。
- 要确定不是索引起始列的列的 HIGH 方式分布信息是否能够提供更好的执行路径，请执行以下步骤：

要确定连接列上的 UPDATE STATISTICS HIGH 是否重要，请执行以下操作：

1. 发出 SET EXPLAIN ON 语句并重新执行查询。
2. 注意 SET EXPLAIN 输出中的估算行数和查询返回的实际行数。
3. 如果这两个数字差别很大，那么对参与连接的列执行 UPDATE STATISTICS HIGH，除非已执行此操作。

重要： 如果表非常大，那么使用 HIGH 方式执行 UPDATE STATISTICS 可能会花费很长时间。

以下示例显示包含连接列的查询：

```
SELECT employee.name, address.city
   FROM employee, address
   WHERE employee.ssn = address.ssn
   AND employee.name = 'James'
```

在该示例中，连接列是 employee 和 address 表中的 ssn 字段。这两列的数据分布必须准确反映实际数据，以便优化器可以正确确定最好的连接计划和执行顺序。

对于当前数据库外部的表，您不能使用 UPDATE STATISTICS 语句创建数据分布。有关数据分布和 UPDATE STATISTICS 语句的更多信息，请参阅《*SinoDB® SQL* 指南: 语法》。

为具有用户定义的数据类型的列更新统计信息

程序员可以编写函数来收集具有用户定义的数据类型的列的统计信息。可以将用户定义的数据类型的数据分布存储在智能大对象空间中。

由于数据库服务器无法获取有关用户定义的数据类型 (UDT) 的性质和使用方面的信息，因此它不能为用户定义的数据类型收集 syscolumns 系统目录表的 colmin 和 colmax 列的统计信息。要为具有用户定义的数据类型的列收集统计信息，程序员必须编写扩展 UPDATE STATISTICS 语句的函数。有关更多信息，请参阅《*SinoDB*® 用户自定义例程和数据类型开发者指南》中有关性能的章节。

由于用户定义的数据类型的数据分布可能非常庞大，因此您可以选择将其存储在智能大对象空间，而非 sysdistrib 系统目录表。

要将用户定义的数据类型的数据分布信息存储于智能大对象空间，请执行以下操作：

1. 使用 onspaces -c -S 命令来创建智能大对象空间。

   要确保数据分布的可恢复性，可在 -Df 选项中指定 LOGGING=ON，如以下示例所示：

   ```
   % onspaces -c -S distrib_sbsp -p /dev/raw_dev1 -o 500 -s
         20000
     -m /dev/raw_dev2 500 -Ms 150 -Mo 200 -Df
        "AVG_LO_SIZE=32,LOGGING=ON"
   ```

   有关调整智能大对象空间大小的信息，请参阅估算智能大对象占用的页数 在第123页。

   有关指定智能大对象空间的存储特征的更多信息，请参阅影响智能大对象空间 *I/O* 的配置参数 在第98页。
2. 在配置参数 SYSSBSPACENAME 中指定步骤 1 创建的智能大对象空间。Specify the sbspace that you created in step 1 in the configuration parameter SYSSBSPACENAME.
3. 将 UPDATE STATISTICS 语句与 MEDIUM 或 HIGH 关键字一起运行以生成数据分布时，通过用户定义的数据类型指定列。

要打印具有用户定义的数据类型的列的数据分布信息，请使用 dbschema -hd 选项。

在超大型数据库上并行更新统计信息

如果您有极大的数据库，并且索引都已分段，那么 UPDATE STATISTICS LOW 可以自动并行执行语句。

要启用语句以自动并行运行，必须在将 PDQ 优先级设置为 1 到 10 之间值的情况下执行 UPDATE STATISTICS LOW。如果将 PDQ 优先级设置为 1，那么一次会分析当前表的 10% 的索引分段。如果将 PDQ 优先级设置为 5，那么一次会分析当前表的 50% 的索引分段。如果将 PDQ 优先级设置为 10，那么一次会分析当前表的所有索引分段。（如果将 PDQ 优先级设置为高于 10 的值，那么 SinoDB® 会按照将 PDQ 优先级设置为 10 来操作，一次会分析当前表的所有索引分段。）

如果执行 UPDATE STATISTICS MEDIUM 或 HIGH，那么可以将 PDQ 优先级设置为高于 10 的值。因为 UPDATE STATISTICS MEDIUM 和 HIGH 语句会执行大量的排序操作，将 PDQ 优先级增加为高于 10 的值可以提供更多内存，有利于提高排序操作的速度。

为 UPDATE STATISTICS 调整内存和磁盘空间量

执行 UPDATE STATISTICS 语句时，数据库服务器将使用内存和磁盘来排序并构造数据分布。您可以影响可用于 UPDATE STATISTICS 操作的内存和磁盘空间量。

您可以使用以下方法来影响 UPDATE STATISTICS 可用的内存和磁盘空间量：

- PDQ 优先级

  设置 PDQ 优先级大于 0 时可以获取更多内存来排序。PDQ 优先级的缺省值为 0。要设置 PDQ 优先级，请使用 PDQPRIORITY 环境变量或 SQL 语句 SET PDQPRIORITY。

  有关 PDQ 优先级的更多信息，请参阅为并行数据库查询分配资源 在第255页。
- DBUPSPACE 环境变量

您可以使用 DBUPSPACE 环境变量来指定 UPDATE STATISTICS MEDIUM 或 UPDATE STATISTICS HIGH 语句在每次传递时可用来构造列分布的系统磁盘空间量（以及用于排序值的内存量）。如果指定的值太小，那么数据库服务器会用足够的空间将最大的列写入磁盘。

有关此环境变量的更多信息，请参阅《*SinoDB® SQL* 指南: 参考》。

更新统计信息操作期间的数据采样

如果您具有超过 100 K 叶子页的大型 B 型树索引，那么以 LOW 方式执行 UPDATE STATISTICS 语句时可根据采样生成索引统计信息。通过采样收集统计信息可以提高更新统计信息操作的速度。

缺省情况下，当 UPDATE STATISTICS 语句执行时，数据库服务器会按顺序读取所有索引叶子页以收集统计信息，例如叶子页数、唯一前导键值数和集群信息。对于大型索引，这可能会花费很长时间。使用采样功能时，数据库服务器会读取一小部分的索引叶子页（样本），然后根据从样本收集的统计信息推断出索引统计信息。

收集统计信息的时间变短的代价可能是所收集的统计信息的准确性。如果前导索引键的数据分布中存在重大偏差，那么采样方法可能会导致所收集的统计信息存在较大的误差范围，而这可能会影响优化器在生成查询计划时所做的决策。

您不能控制样本中的数据量。

要启用或禁用采样功能，请使用 USTLOW_SAMPLE 配置参数或 SET ENVIRONMENT 语句的 USTLOW_SAMPLE 环境选项。

相关链接

《SinoDB 管理员参考》: USTLOW_SAMPLE 配置参数

《SinoDB SQL 指南: 语法》: USTLOW_SAMPLE 环境选项

显示数据分布

可以使用 dbschema 实用程序显示数据分布。

除非列值发生很大变化，否则无需重新生成数据分布。要验证分布信息的精确性，请将 dbschema -hd 输出和相应构造的 SELECT 语句的结果进行比较。

例如，以下 dbschema 命令生成数据库 vjp_stores 中 customer 表每列的分布列表，列表中含有每个二进制存储单元中值的数量，以及不同值的数量：

```
dbschema -hd customer -d vjp_stores
```

图 71: 使用 dbschema -hd 显示数据分布信息 在第279页显示了该 dbschema -hd 命令生成列 zipcode 的数据分布信息。由于该列是 zip_ix 索引的起始列，因而 UPDATE STATISTICS HIGH 在该列上运行，如以下输出行所示：

```
High Mode, 0.500000 Resolution
```

图 71: 使用 dbschema -hd 显示数据分布信息 在第279页显示了 17 个二进制存储单元，每个二进制存储单元中都包含一个不同的 zipcode 值。

```
dbschema -hd customer -d vjp_stores

...
Distribution for virginia.customer.zipcode

Constructed on 09/18/2000

High Mode, 0.500000 Resolution

--- DISTRIBUTION ---

     (          02135 )
```

```
  1: (  1,    1, 02135 )
  2: (  1,    1, 08002 )
  3: (  1,    1, 08540 )
  4: (  1,    1, 19898 )
  5: (  1,    1, 32256 )
  6: (  1,    1, 60406 )
  7: (  1,    1, 74006 )
  8: (  1,    1, 80219 )
  9: (  1,    1, 85008 )
 10: (  1,    1, 85016 )
 11: (  1,    1, 94026 )
 12: (  1,    1, 94040 )
 13: (  1,    1, 94085 )
 14: (  1,    1, 94117 )
 15: (  1,    1, 94303 )
 16: (  1,    1, 94304 )
 17: (  1,    1, 94609 )


--- OVERFLOW ---

  1: (  2,       94022 )
  2: (  2,       94025 )
  3: (  2,       94062 )
  4: (  3,       94063 )
  5: (  2,       94086 )
```

图 71: 使用 dbschema -hd 显示数据分布信息

因为输出的 OVERFLOW 部分显示的是可能会使分布数据产生偏差的重复值，所以 dbschema 将其从分布信息中移出到单独的列表。此溢出列表中重复值数必须大于以下公式确定的临界值。图 71: 使用 *dbschema -hd* 显示数据分布信息 在第279页显示分辨率的值为 .0050。因此，该公式可以确定任何重复一次以上的值都将在溢出部分列出。

```
Overflow = .25 * resolution * number_rows
      = .25 * .0050 * 28
      = .035
```

有关 dbschema 实用程序的更多信息，请参阅《*SinoDB*® 迁移指南》。

通过添加或移除索引来提高性能

通常，通过添加索引或在某些情况下删除索引可提高查询的性能。您也可以启用优化器从索引缓冲区自动访存一组键。

要提高查询的性能，请考虑使用以下主题描述的一些方法。

此外：

- 当数据库及其相关联的表持续可用时，请考虑使用 CREATE INDEX ONLINE 和 DROP INDEX ONLINE 语句在联机环境中创建和删除索引。在构建或删除索引的持续时间内，这些 SQL 语句使您能够创建和删除索引而无需具有放置在表上的访问锁。有关更多信息，请参阅在联机环境中创建和删除索引 在第163页。
- 将 BATCHEDREAD_INDEX 配置参数设置为启用优化器，以从索引缓冲区自动访存一组键。这会减少读取缓冲区的次数。

相关链接

《*SinoDB* 管理员参考》: *BATCHEDREAD_INDEX* 配置参数

将自动索引替换为永久索引

如果查询计划包含指向大表的自动索引路径，一般可以通过添加该列上的索引来提高性能。如果定期执行查询，可通过创建永久索引来节省时间。

如果只是偶而执行查询，那么可以适当地让数据库服务器建立和丢弃索引。

使用组合索引

优化器可以通过数种方法使用组合索引（涵盖多个列的索引）。

数据库服务器可以通过以下方法对列 a、b 和 c（按此顺序）使用索引：

- 定位一个特殊行

  如果第一个过滤器是等式过滤器，且随后的列有范围（<、<=、> 和 >=）表达式，那么数据库服务器可以使用组合索引。以下过滤器示例使用组合索引中的列：

  ```
  WHERE a=1
  WHERE a>=12 AND a<15
  WHERE a=1 AND b < 5
  WHERE a=1 AND b = 17 AND c >= 40
  ```

  以下过滤器示例不能使用该组合索引：

  ```
  WHERE b=10
  WHERE c=221
  WHERE a>=12 AND b=15
  ```

- 当所有所需的列均包含于索引内时要替换表扫描

  使用索引但不引用表的扫描称为仅关键字搜索。
- 将列 a、列 ab 或列 abc 连接到另一个表
- 对列 a、ab 或 abc 实现 ORDER BY 或 GROUP BY，但不对 b、c、ac 或 bc 实现 ORDER BY 或 GROUP BY

如果以不同程度从最高到最低为序对列创建组合索引，那么执行效率最高。换言之，在 SELECT 语句中使用 DISTINCT 关键字进行查询时，返回不同行最多的列应排在组合索引的第一位。

如果应用程序执行若干长查询，且每个长查询都包含 ORDER BY 或 GROUP BY 子句，那么您有时可以通过添加无需排序即能生成这些序列的索引来提高性能。例如，以下查询在 ORER BY 子句中按不同方向对每个列进行排序：

```
SELECT * FROM t1 ORDER BY a, b DESC;
```

要在对列 a 进行升序排序并对列 b 进行降序排序时避免使用临时表，您必须对 (a, b DESC) 或 (a DESC, b) 创建组合索引。由于数据库服务器具有双向遍历能力，因此您只需创建这些索引的其中一个。有关双向遍历的更多信息，请参阅《*SinoDB® SQL* 指南: 语法》。

另一方面，当满足以下条件时，执行表扫描并对结果进行排序的成本要比使用组合索引低。

- 表相对于索引的顺序尚佳。
- 查询检索的行数在可用数据中占的比例很大。

数据仓库应用程序的索引

许多数据仓库数据库都使用星型模式，该模式由一个事实表和许多维表组成。使用星型模式或雪花模式中的表的查询可以从对事实表的适当索引中获益。

事实表通常很大，包含有关主题的定量或真实信息。维表则描述事实表的属性。

当某个维需要较低级别的信息时，会使用称为雪花模式的表层次结构来对该维建模。

请考虑如下星型模式的示例，该星型模式有一个名为 orders 的事实表，和四个名为 customers、suppliers、products 和 clerks 的维表。orders 表描述每份销售订单的详细信息，包括客

户标识符、供应商标识符、产品标识符和销售员标识符。每个维表均详细描述一个标识符。orders 表较大，而四个维表较小。

以下查询可查找供应商 Johnson 提供的硬盘在 Menlo Park 地区（邮政编码为 94025）的总直销收入：

```
SELECT sum(orders.price)
FROM orders, customers, suppliers,product,clerks
WHERE orders.custid = customers.custid
   AND customers.zipcode = 94025
   AND orders.suppid = suppliers.suppid
   AND suppliers.name = 'Johnson'
   AND orders.prodid = product.prodid
   AND product.type = 'hard drive'
   AND orders.clerkid = clerks.clerkid
   AND clerks.dept = 'Direct Sales'
```

该查询使用典型的星型连接，其中事实表与所有维表通过外键连接。每个维表都有一个选择性表过滤器。

星型连接的最优计划是对四个维表执行笛卡尔乘积，然后将结果与事实表连接起来。以下对事实表的索引允许优化器选择最优查询计划：

```
CREATE INDEX ON orders(custid,suppid,prodid,clerkid)
```

如果没有此索引，优化器可能选择先将事实表与单个维表进行连接，然后将其结果与剩余维表连接。最优计划能提供更好的性能。

有关星型模式和雪花模式的更多信息，请参阅《*SinoDB*® 数据库设计和实现指南》。

配置 B 型树扫描程序信息来改进事务处理

通过控制 B 型树扫描程序线程从索引中移除删除项的方式，可以改进使用日志记录的数据库中事务处理的性能。

从带有索引的表中删除行时，B 型树扫描程序改进了使用日志记录的数据库的事务处理。B 型树扫描程序根据优先级列表自动确定哪些索引分区要删除。B 型树扫描程序线程移除已删除的索引条目，并重新均衡索引节点。B 型树扫描程序会自动确定要删除哪些索引项。

在使用日志记录的数据库中，当在行上执行删除或更新操作时，相应的索引条目并不会马上被删除。相反，相应的索引条目会被标记为删除直到 B 型树扫描程序线程扫描该索引并将其删除。包含很多删除标记条目的索引可导致严重的性能问题，因为索引搜索需要扫描大量条目才能找到第一个有效条目。

根据您的索引，B 型树扫描的缺省设置可以提供以下扫描类型：

- 如果表中具有多个连接的索引，那么 B 型树扫描程序会使用叶扫描方式。叶扫描方式是可能针对多个连接的索引采用的唯一扫描类型。
- 如果表中只包含一个连接的索引，或如果索引已被拆离，那么 B 型树扫描程序会使用 alice（适应线性索引清除）方式。初始 alice 扫描方式已针对很少或没有超过 1 GB 索引的中小型系统进行了优化。但是，如果数据库服务器检测到 alice 方式效率低，那么将会自动调整 alice 扫描方式设置以适应更大的索引需求。

根据应用程序以及系统在索引中添加或删除键的不同顺序，索引结构可能会变得低效。

使用 BTSCANNER 配置参数来指定以下用于定义扫描方式的信息：

- 数据库服务器启动时要启动的 B 型树扫描程序线程数量

  可以将 B 型树扫描程序线程的数量配置为任意正数。由于一个 B 型树扫描程序线程将始终清理单个索引分区，因此如果偶尔或经常有大量索引分区需要进行清理，您可能想要使用多个 B 型树扫描程序线程。在运行时，通过发出 onmode -C 命令，可关闭所有 B 型树扫描程序活动。此命令会停止所有 B 型树扫描程序线程。
- 阈值，即在将索引放入优先级列表中由 B 型树扫描程序线程用于扫描和清理之前该索引中删除项的最小数目

例如，如果将阈值增加到 5000 以上，可避免在接收最多更新和删除的索引上进行频繁的 B 型树扫描程序活动。

- 范围大小（以 KB 为单位），索引或索引分段必须超过此范围之后才能使用范围扫描来清理索引
- 一个 alice 方式的值
- B 型树扫描程序线程通过合并两个部分使用的索引页来压缩索引的级别

服务器处理森林树索引的方式与处理 B 型树索引的方式相同。因此，在使用日志记录的数据库中，您可以控制 B 型树扫描程序线程如何从森林树索引和 B 型树索引除去删除。

下表总结了两种扫描方式之间的差异。

表 17: B 型树扫描程序线程的扫描方式

扫描方式	描述	性能优势或问题	更多信息
叶扫描方式	在该方式中，将彻底扫描叶级的连接索引以查找删除项。	该方式只有在连接的索引上才可以使用，并且当一个分区中存在多个连接的索引时，该方式将是服务器唯一可用方式。	叶和范围扫描方式设置 在第285页
Alice（适应线性索引清除）扫描方式	如果启用了 BTSCANNER alice 选项，那么每个索引分区将接收到一个位图，用于跟踪在索引中找到删除项的位置。这种扫描将所有未找到删除操作的索引排除在外。 这些位图的初始大小和粒度取决于所表示的分区大小以及当前整个系统的 alice 级别。通过检查需清理页或读取页的比率，服务器会定期检查每个位图的效率，并在必要时调整扫描以获取更好的信息。该方式为使用过多 I/O 的索引分配额外的资源（内存）。	使用 alice 方式时可以极大地提高性能和降低 I/O。一般而言，alice 方式比范围扫描的效率高出 64 倍，并且可以根据不适用的索引自动进行自我调整，这是范围扫描所不及的。	*Alice* 扫描方式值 在第284页
范围扫描方式	范围扫描可由 rangesize 选项来启用，在低页地址和高页地址之间的范围内执行。只有在该范围内扫描叶级的索引分区。服务器将执行轻量级扫描，即使清理涉及整个缓冲池，该扫描也不会立即使用并填满缓冲池。	不推荐使用 SinoDB® V11.10 或更高版本。Alice 扫描与范围扫描完全相同，但其效率为后者的 64 倍，使用相同的资源，并有 128 个等同范围。 当设置 alice 方式扫描时，范围扫描将不起作用。 如果决定对只有大量大型索引的系统使用范围扫描，那么请将 rangesize	叶和范围扫描方式设置 在第285页

扫描方式	描述	性能优势或问题	更多信息
		选项设置为范围扫描的最小分区大小。	

有关 BTSCANNER 配置参数的更多信息以及有关数据库服务器如何维护索引树的更多信息，请参阅《*SinoDB*® 管理员参考》中有关配置参数的章节以及有关磁盘结构和存储的章节。

使用 onstat -C 选项可监视 B 型树扫描程序活动。

使用 onmode -C 选项可在运行时更改 B 型树扫描程序的配置。

有关 onstat -C 和 onmode -C 的更多信息，请参阅《*SinoDB*® 管理员参考》。

Alice 扫描方式值
通过将 alice 选项设置为 1 到 12（最佳初始粒度）之间的任意值来启用 alice（适应线性索引清除）方式。对于很少或没有索引大于 1 GB 的中小型系统，将 alice 选项设置为 6 或 7。对于具有大型索引的系统，将 alice 设置为更高的方式。

当设置 alice 方式时，模式值越大，每个索引分区使用的内存就越大。不过，使用的内存并不算多。优点是 I/O 量较少，如下表所示。

表 18: Alice 方式设置

Alice 方式设置	内存或块 I/O
0	关闭 alice 扫描。
1	使用正好 8 个字节的内存（不调整）。
2	使用正好 16 个字节的内存（不调整）。
3	页的每个数据块将需要 512 个 I/O 操作以执行清除。
4	页的每个数据块将需要 256 个 I/O 操作以执行清除。
5	页的每个数据块将需要 128 个 I/O 操作以执行清除。
6（缺省值）	页的每个数据块将需要 64 个 I/O 操作以执行清除。
7	页的每个数据块将需要 32 个 I/O 操作以执行清除。
8	页的每个数据块将需要 16 个 I/O 操作以执行清除。
9	页的每个数据块将需要 8 个 I/O 操作以执行清除。
10	页的每个数据块将需要 4 个 I/O 操作以执行清除。
11	页的每个数据块将需要 2 个 I/O 操作以执行清除。
12	页的每个数据块将需要 1 个 I/O 操作以执行清除。

设置 alice 方式时，需要考虑内存使用率与 I/O。alice 方式设置越低，索引将使用的内存越少。alice 方式设置越高，索引将使用的内存越多。12 是最高方式值，因为其为单个内存位到 I/O 的每个实例的直接映射。

假设您有一个大小为 2 KB 的联机页和一个 I/O 大小为 256 个页的缺省 B 型树扫描程序。如果将 alice 方式设置为 6，那么内存的每个字节可以代表 131,072 个索引页 (256 MB)。如果将该方式设置为 10，那么内存的每个字节可以代表 8,192 个索引页 (16 MB)。因此，该方式的设置从 6 更改为 10 将需要 16 倍的内存，但是需要的 I/O 只有十六分之一。

如果您有一个使用 1 GB 的索引分区，那么 alice 方式设置为 6 将占用 4 个字节的内存，而 alice 方式设置为 10 将占用 64 个字节的内存，如以下公式所示：

```
( {mode block size} io per bit  *  8 bits per byte  * 256 page per io )
```

将 alice 方式设置为 3 到 12 之间的值会设置用于索引清除的初始内存量。然后，B 型树扫描程序根据过去清除操作的效率自动调整方式。

例如，如果在五次扫描（缺省情况下）之后，I/O 效率低于 75%，那么当您将该方式设置为大于 2 的值时服务器将自动调整到下一个 alice 方式。例如，如果索引当前在值为 6 的 alice 方式下运行，并且 B 型树扫描程序已至少五次清理了该索引，而且 I/O 效率低于 75%，那么该服务器将自动调整到值为 7 的方式，即下一个更高的方式。这将使所需的内存翻倍，但 I/O 减少了二分之一。

再进行五次扫描之后，服务器将重新评估该索引以再次确定 I/O 的效率，并且将继续这个操作直到方式为 12 为止。在方式为 12 时服务器将停止调整。

以下示例将 alice 方式设置为 6：

```
BTSCANNER num=2,threshold=10000,alice=6,compression=default
```

叶和范围扫描方式设置
如果表中具有多个连接的索引，那么 B 型树扫描程序会使用叶扫描方式。如果希望由叶扫描方法扫描小型索引，请将 BTSCANNER 配置参数的 rangesize 选项设置为 100。

如果您决定在分区中只存在一个索引时启用范围扫描方式，那么请将 BTSCANNER 配置参数 rangesize 选项设置为使用该方式时分区必须被扫描的最小值。以 KB 为单位指定大小。

以下示例指定：

- 服务器将启动两个 B 型树扫描程序线程。
- 当在索引中发现 50000 个已删除项时，服务器将考虑清理热列表（导致服务器执行额外工作的索引列表）中的索引。
- 将使用范围扫描方式清理分区大小大于或等于 100 KB 的索引。
- 将使用叶扫描方式清理分区大小小于 100 KB 的索引。
- 索引压缩设置在中等（缺省值）级别

```
BTSCANNER num=2,threshold=50000,rangesize=100,compression=default
```

B 型树扫描程序索引压缩级别和事务处理性能
如果两个部分使用的索引页的数据量低于压缩选项指定的级别，那么 B 型树扫描程序线程将通过合并这些页来压缩索引。可以设置压缩级别以控制查找和载入数据所需的 I/O 量。

B 型树扫描程序线程查找由于低于指定的级别而可压缩的索引页。B 型树扫描程序可以压缩具有已删除项的索引页和没有已删除项的页。

缺省情况下，B 型树扫描程序以中等级别进行压缩。下表提供有关将压缩级别更改为高或低的情况下产生的性能益处和代价的信息。

表 19: B 型树扫描程序压缩级别益处和代价

压缩级别	性能益处和代价	何时使用
低	对于预期会快速增长并带有频繁的 B 型树节点分裂的索引，低压缩级别很有利。压缩级别设置为低时，B 型树索引不需要具有中高压缩级别索引的那么多分裂，因为更多的可用空间保留在 B 型树节点中。	如果预期索引快速增长且频繁分裂，那么您可能想要将压缩级别更改为低。
高	通常，如果索引为只读或其 90% 为只读，那么高压缩级别很有利，因为搜索数据将需要遍历更少的页（和更少的 I/O）。示例可能为没有频繁更改的索引或正在进行批量（块）删除操作的索引。	在以下情况下您可能希望将压缩级别更改为高： • 如果在大部分时间读取某索引，那么在小部分时间内将发生删除和插入操作。 • 如果批量载入和更新表，并在一段时间内将其保留为只读表。

压缩级别	性能益处和代价	何时使用
	使用高级别压缩还意味着性能代价，因为其耗用更多 I/O 来更积极地压缩索引。压缩级别高的时候，选择操作将具有较少 I/O。	

如果无需将压缩级别更改为高或低，请将 BTSCANNER 配置参数的压缩选项设置为 med 或 default。

索引压缩和索引填充因子

除了指定何时尝试连接两个部分使用页的压缩选项，也可以使用 FILL FACTOR 配置参数来控制何时添加新的索引页。用 FILLFACTOR 配置参数或 CREATE INDEX 语句的 FILLFACTOR 选项定义的索引填充因子是索引构建过程中填充的每个索引页的百分比。

设置 B 型树扫描程序索引压缩的级别

SinoDB® 提供若干方式来指定 B 型树扫描程序线程压缩索引页的级别。要优化空间和事务处理，可以在索引增长迅速时降低压缩级别。如果索引有很少的删除和插入操作或者执行批量更新，那么可以提升级别。

先决条件：

- 确定调整索引压缩级别是否将改善性能。
- 通过执行 onstat -g ppf 命令来获取关于读取、删除和插入的行数的统计信息。也可以查看 sysptprof 表中的信息。
- 分析统计信息以确定是否希望更改阈值。

有关压缩级别和希望更改级别的情况的信息，请参阅*B 型树扫描程序索引压缩级别和事务处理性能* 在第285页。

用以下任意选项指定实例的压缩级别：

- 将 BTSCANNER 配置参数的 compression 字段设置为 low、med（中等）、high 或 default。（系统缺省值为 med。）
- 执行 onmode -C compression ***value*** 命令，其中 *value* 为 low、med（中等）、high 和 default。系统缺省值为 med。
- 用以下命令执行 SQL 管理 API 函数：

```
SET INDEX COMPRESSION, partition number, compression level
```

示例

按如下方式将 BTSCANNER 配置参数的压缩选项设置为 default：

```
BTSCANNER num=4,threshold=10000,rangesize=-1,alice=6,compression=default
```

按如下方式将 BTSCANNER 配置参数的压缩选项设置为 high：

```
BTSCANNER num=4,threshold=5000,compression=high
```

使用 onmode -C 来指定压缩级别，如下所示：

```
onmode  -C compression high
```

执行以下任一 SQL 管理 API 函数为具有分区编号 1048960 的索引的单个分段设置压缩级别：

```
EXECUTE FUNCTION TASK("SET INDEX COMPRESSION", 1048960, "DEFAULT");
```

```
EXECUTE FUNCTION ADMIN("SET INDEX COMPRESSION", 1048960, "LOW");
```

执行以下 SELECT 语句来对所有索引分段执行任务函数。此命令为名为 db1 的数据库中索引 idx1 的所有分段设置压缩级别。

```
SELECT sysadmin:TASK("SET INDEX COMPRESSION", partnum, "MED")
FROM sysmaster:systabnames
WHERE dbsname = 'db1' AND tabname = 'idx1';
```

也可以执行以下 SELECT TASK 语句来对所有索引分段执行任务函数，并为所有分段设置压缩级别。

```
SELECT TASK("SET INDEX COMPRESSION", partn, "MED")
FROM db1:systables t, db1:sysfragments f
WHERE f.tabid = t.tabid AND f.fragtype = 'I' AND indexname ='idx1';
```

确定索引页中的可用空间量

可以使用 oncheck -pT 命令确定每个索引页中的可用空间量。

如果表具有相对较少的更新活动并存在大量的可用空间，那么您可能想要使用较大的 FILLFACTOR 值来删除和重建索引，从而使未使用的磁盘空间可用。

分布式查询的优化器估算

优化器假定从远程数据库访问一行所花费的时间要比从本地数据库访问一行所花费的长。优化器的估算值中包括从磁盘检索该行以及通过网络传输的成本。

有关此较高的估算成本的示例，请参阅分布式查询的查询计划 在第287页。

分布式查询的缓冲区数据传输

SinoDB® 使用若干因素来确定向远程服务器发送数据和从其接收数据的缓冲区的大小。

服务器使用以下因素确定缓冲区大小：

- 行大小

  数据库服务器通过对平均移动大小（如果可获得）或列长度（来自 syscolumns 系统目录表）求和来计算行大小。
- 客户端上 FET_BUF_SIZE 环境变量的设置

  通过使用 FET_BUF_SIZE 环境变量来增加数据库服务器将行发送到远程数据库服务器或从远程数据库服务器接收行时所使用的缓冲区的大小，您也许能够降低数据传送大小和数量。

  根据行的大小，最小的缓冲区大小为 1024 或 2048 字节。如果行大小大于 1024 或 2048 字节，那么数据库服务器将使用 FET_BUF_SIZE 的值。

  有关 FET_BUF_SIZE 环境变量的更多信息，请参阅《*SinoDB® SQL* 指南*:* 参考》。

分布式查询的查询计划

可以显示为分布式查询选择的查询计划。为分布式连接显示的信息不同于为本地连接显示的信息。

下图显示为分布式查询选择的查询计划。

```
QUERY:
------
select l.customer_num, l.lname, l.company,
         l.phone, r.call_dtime, r.call_descr
       from customer l, vjp_stores@gilroy:cust_calls r
        where l.customer_num = r.customer_num
```

```
Estimated Cost: 9
Estimated # of Rows Returned: 7

  1) informix.r: REMOTE PATH

  2) informix.l: INDEX PATH

    (1) Index Keys: customer_num   (Serial, fragments: ALL)
        Lower Index Filter: informix.l.customer_num = informix.r.customer_num
NESTED LOOP JOIN
```

图 72: 分布式查询的 SET EXPLAIN ALL 选择输出, 第 3 部分

下表显示为分布式连接和本地连接选择的查询计划的主要区别。

分布式查询的图 *72:* 分布式查询的 ***SET EXPLAIN ALL*** 选择输出, 第 *3* 部分 在第287页中的输出行	仅本地查询的 图 ***62: EXPLAIN AVOID_EXECUTE*** 指令的结果 在第243页中的输出行	区别的描述
vjp_stores@gilroy: virginia.cust_calls	informix.cust_calls	远程表名以数据库和服务器名称开头。
Estimated Cost: 9	Estimated Cost: 7	优化器为分布式查询估算的成本较高。
informix.r: REMOTE PATH	informix.r: SEQUENTIAL SCAN	优化器选择在远程站点保存外部远程 cust_calls 表。
select x0.call_dtime,x0.call_descr,x0. customer_num from vjp_stores:"virginia".cust_ calls x0		本地数据库服务器发送到远程站点的 SQL 语句。远程站点重新优化该语句以选择实际计划。

改进顺序扫描

可以消除重复的顺序扫描来提高对大表顺序读取操作的性能。

顺序访问不是规划中第一个表的表可能导致性能下降，因为对于从前述表选择的每一行，可能均要对表中的每行读一次。

如果表较小，重复读取影响不大，因为表完全驻留在内存中。对内存中的表进行顺序搜索要比通过索引搜索同一个表速度快，尤其是在将这些索引页保留在内存中而将其他有用的页挤出缓冲区的情况下。

但是，当表大于数页时，重复顺序访问会导致低性能。避免此问题的一种方法是对用于连接表的列建立索引。

具有 Resource 权限的任何用户都可以建立附加索引。使用 CREATE INDEX 语句可建立索引。

索引占用的磁盘空间与关键字值的宽度和行数成比例。（请参阅估算索引页 在第153页。）同时，数据库服务器必须在每次插入、删除或更新行时更新索引，索引更新将减慢这些操作的速度。如果必要，您可以使用 DROP INDEX 语句在一系列查询之后释放索引，此操作将释放空间并使更新表的操作更加容易。

启用视图折叠以提高查询性能

通过启用视图并入，可以显著提高与视图有关的查询性能。

您可以通过将 IFX_FOLDVIEW 配置参数设置为 1 来做到这一点。

启用视图并入后，视图将并入到一个父查询中。由于视图并入父查询中，因而查询结果将不能置于临时表中。

您可以在以下类型的查询中使用视图并入：

- 包含 UNION ALL 子句和父查询的视图具有常规连接、SinoDB® 连接、ANSI join 或 ORDER BY 子句

在执行有关视图的 UNION ALL 操作的以下类型查询中，将不发生视图并入：

- 该视图具有以下任一子句：AGGREGATE、GROUP BY、ORDER BY、UNION、DISTINCT 或 OUTER JOIN（SinoDB® 或 ANSI 类型）。
- 父查询具有 UNION 或 UNION ALL 子句。

在这些情境中，创建一个临时表以保存查询结果。

减少连接和排序操作

了解查询做些什么后，可寻找一些方法，以较少的代价获得相同的产出。

以下建议可以帮助您更有效地重写查询：

- 避免或简化排序操作。
- 使用并行排序。
- 使用临时表减小排序范围。

避免或简化排序操作

在许多情况下，可以确定如何避免或减少频繁或复杂的排序操作。

排序算法是高度优化的，效率极高。对同样的数据应用排序算法与应用任何外部排序程序的执行速度一样快。对于很少发生的排序，或输出行数相对较少的排序，不必刻意避免。

但是，应尽量避免或减少大表的重复排序范围。只要能够使用索引自动生成具有适当顺序的输出，优化器就不使用排序步骤。以下因素会防止优化器使用索引：

- 一个或多个已排序列未包含在索引中。
- 列在索引和 ORDER BY 或 GROUP BY 子句中以不同的顺序命名。
- 已排序的列取自不同的表。

有关避免排序的另一种方法，请参阅使用临时表减小排序范围 在第290页。

如果必须进行排序，那么应想办法使其简化。如排序时间成本 在第227页中所讨论的，如果您可以对较少或较窄的列进行排序，那么排序速度将较快。

相关链接

使用分段索引排序 在第293页

使用并行排序

如果您无法避免排序操作，那么数据库服务器将利用多个 CPU 资源以并行执行必需的排序和合并操作。数据库服务器可以对任何查询使用并行排序，而不限于 PDQ 查询。可以控制数据库服务器用于排序行的线程数。

要控制数据库服务器用于排序行的线程数，可使用 PSORT_NPROCS 环境变量。

当 PDQ 优先级大于 0 且 PSORT_NPROCS 大于 1 时，查询将从并行排序和 PDQ 功能（例如，并行扫描和额外内存）两方面获益。用户可以使用 PDQPRIORITY 环境变量为查询请求 PDQ 资源的特定部分。可以使用 MAX_PDQPRIORITY 配置参数来限制此类用户请求的数目。有关更多信息，请参阅限制查询中的 *PDQ* 资源 在第45页。

在某些情况下，正在排序的数据量可能超出分配给查询的内存资源，这将导致对数据库空间或排序文件的 I/O 操作。有关更多信息，请参阅为临时表和排序文件配置数据库空间 在第90页。

使用临时表减小排序范围

可以使用一个表的有序临时子集提高查询速度。临时表也可以简化查询优化器的工作，使优化器避免多次排序操作，并以其他方式简化优化器的工作。

例如，假设应用程序针对具有未付余额的客户生成了一系列报告，每个主要邮政区域生成一份报告，按客户名称排序。换言之，发生了一系列查询，每个查询均采用了以下形式（使用假设的表和列名）：

```
SELECT cust.name, rcvbles.balance, ...other columns...
   FROM cust, rcvbles
   WHERE cust.customer_id = rcvbles.customer_id
      AND rcvbls.balance > 0
      AND cust.postcode LIKE '98_ _ _'
   ORDER BY cust.name
```

该查询读取整个 cust 表。对于具有指定邮政编码的每一行，数据库服务器将搜索对 rcvbles.customer_id 的索引，并执行非顺序磁盘存取，以找到所有匹配行。然后将这些行写入临时文件并进行排序。有关临时文件的更多信息，请参阅为临时表和排序文件配置数据库空间 在第90页。

如果查询只执行一次，此过程是可以接受的，但该示例中包含一系列查询，每个查询都产生同样的工作量。

备选方案是选择所有具有未付余额的客户，将他们放入一个临时表，在表中以客户名称排序，如以下示例所示：

```
SELECT cust.name, rcvbles.balance, ...other columns...
   FROM cust, rcvbles
   WHERE cust.customer_id = rcvbles.customer_id
      AND cvbls.balance > 0
   INTO TEMP cust_with_balance
```

然后可以对该临时表执行查询，如以下示例所示：

```
SELECT *
   FROM cust_with_balance
   WHERE postcode LIKE '98_ _ _'
   ORDER BY cust.name
```

每个查询均顺序读取临时表，但该表的行数比主表少。

为使用散列连接、聚合和其他内存密集型元素的查询分配更多内存

可设置某些配置参数，以便为需要排序、散列连接、聚合和其他内存密集型元素的查询提供更多内存。

如何配置可用于查询的内存量取决于该查询是否是并行数据库查询 (PDQ)。

为非 PDQ 查询配置内存

如果将 PDQ 优先级设置为 0（零），那么可以通过更改 DS_NONPDQ_QUERY_MEM 配置参数的值来更改可用于非 PDQ 查询的内存量。如果 PDQ 优先级设置为零，那么您只能使用该参数。如果 PDQ 优先级大于零，那么该设置不起任何作用。

您也可以使用 onmode -wm 或 onmode -wf 命令来更改 DS_NONPDQ_QUERY_MEM 的值。

例如，如果使用 onmode 实用程序，那么请如以下示例中所示指定一个值：

```
onmode -wf DS_NONPDQ_QUERY_MEM=500
```

DS_NONPDQ_QUERY_MEM 的最小值为 128 KB。受支持的最大值为 DS_TOTAL_MEMORY 的 25%。128 KB 是 DS_NONPDQ_QUERY_MEM 的缺省值。如果您为 DS_NONPDQ_QUERY_MEM 参数指定值，请根据查询中涉及的表行数和表行大小来确定并调整该值。

如果此值超过了 DS_TOTAL_MEMORY 值的 25%，那么 SinoDB® 可能会重新计算 DS_NONPDQ_QUERY_MEM 的初始值。

如果 SinoDB® 更改了您设置的值，那么服务器会使用以下格式发送一条消息：

DS_NONPDQ_QUERY_MEM recalculated and changed from old_value Kb to new_value Kb.

在该消息中，old_value 表示您在用户配置文件中对 DS_NONPDQ_QUERY_MEM 指定的值，而 new_value 表示 SinoDB® 确定的值。

有关估算向散列连接分配的其他空间量的公式，请参阅估算用于数据库空间和散列连接的临时空间 在第92页。

为 PDQ 查询配置内存

内存分配管理器 (MGM) 是一个 SinoDB® 组件，它在决策支持查询之间协调内存、CPU 虚拟处理器 (VP)、磁盘 I/O 以及扫描线程的使用情况。MGM 使用 DS_MAX_QUERIES、DS_TOTAL_MEMORY、DS_MAX_SCANS 和 MAX_PDQPRIORITY 配置参数来确定这些可分配给决策支持查询的 PDQ 资源量。MGM 也为散列连接之类活动的查询分配内存。有关 MGM 的更多信息，请参阅内存分配管理器 在第254页。

优化查询的用户响应时间

您可以影响 SinoDB® 优化查询并将行返回给用户所需的时间量。

优化级别

通常情况下，使用缺省优化级别 HIGH 可以获取最佳整体性能。优化语句所花费的时间通常不重要。但是，如果对应用程序的试验显示查询仍需花费太长时间，那么您可以将优化级别设置为 LOW。

如果将优化级别更改为 LOW，请检查 SET EXPLAIN 输出以查看优化器是否选择了与先前相同的查询计划。

要指定数据库服务器优化的 HIGH 或 LOW 级别，请使用 SET OPTIMIZATION 语句。

相关链接

《SinoDB SQL 指南: 语法》: SET OPTIMIZATION 语句

优化目标

优化总的查询时间和优化用户响应时间是提高查询性能的两个优化目标。

总的查询时间是将所有行返回给应用程序所花费的时间。对于批处理，或者对于那些要求在将结果返回给用户前处理所有行的查询，总的查询时间最显重要，如以下查询所示：

```
SELECT count(*) FROM orders
WHERE order_amount > 2000;
```

用户响应时间是数据库服务器将一满屏行返回给交互式应用程序所花费的时间。在交互式应用程序中，一次只能请求一满屏的数据。例如，对于以下查询，用户应用程序一次只能显示 10 行：

```
SELECT * FROM orders
WHERE order_amount > 2000;
```

哪个优化目标更重要，会影响到优化器选择的查询路径。例如，如果用户响应时间更重要，优化器可能会选择嵌套循环连接而不是散列连接来执行查询，即使散列连接可以降低总的查询时间。

指定查询性能目标

您可以优化会话内整个数据库服务器系统或各个查询的用户响应时间。

优化器的缺省行为是选择优化总查询时间的查询计划。您可以在以下几种不同级别上指定用户响应时间的优化：

- 对于数据库服务器系统

要优化用户响应时间，将 OPT_GOAL 配置参数设置为 0，如以下示例所示：

```
OPT_GOAL 0
```

将 OPT_GOAL 设置为 -1 可优化总查询时间。

- 对于用户环境

  OPT_GOAL 环境变量可以在用户应用程序启动之前设置。

  仅适用于 UNIX™

  要优化用户响应时间，可以设置 OPT_GOAL 环境变量 为 0，如以下命令示例所示：

  ```
  Bourne shell                OPT_GOAL = 0
                export OPT_GOAL

  C shell               setenv OPT_GOAL 0
  ```

  对于总查询时间的优化，可以设置 OPT_GOAL 环境变量为 -1。

- 在会话内

  您可以使用 SQL 中的 SET OPTIMIZATION 语句来控制优化目标。使用该语句设置的优化目标将保持生效，直至会话终止，或直至使用其他 SET OPTIMIZATION 语句更改该目标。

  以下语句使优化器选择有利于总查询时间优化的查询计划：

  ```
  SET OPTIMIZATION ALL_ROWS
  ```

  以下语句使优化器能够选择有利于用户响应时间优化的查询计划：

  ```
  SET OPTIMIZATION FIRST_ROWS
  ```

- 对于单个查询

  您可以使用 FIRST_ROWS 和 ALL_ROWS 优化器指令来指示优化器使用哪种查询目标。有关这些指令的更多信息，请参阅优化目标指令 在第242页。

这些级别的优先顺序如下：

- 优化器指令
- SET OPTIMIZATION 语句
- OPT_GOAL 环境变量
- OPT_GOAL 配置参数

例如，优化器指令的目标优先于 SET OPTIMIZATION 语句指定的目标。

用户响应时间优化的首选查询计划
当优化器选择用于优化用户响应时间的查询计划时，它将为每个计划计算检索查询中的首行所需的开销，然后选择开销最低的计划。在某些情况下，检索首行所需开销最低的查询计划可能并非检索查询中所有行的最佳路径。

以下各节描述了查询计划中可能存在的一些区别。

嵌套循环连接与散列连接
散列连接检索首行的成本通常比嵌套循环连接要高。使用散列连接时，检索任何行之前数据库服务器都必须建立散列表。但是，在某些情况下，如果数据库服务器使用散列连接，会缩短总查询时间。

在以下示例中，tab2 具有对 col1 的索引，但 tab1 不具有对 col1 的索引。执行查询之前执行 SET OPTIMIZATION ALL_ROWS 时，数据库服务器将使用散列连接并忽略现有的索引，如以下 SET EXPLAIN 输出部分所示：

```
QUERY:
------
```

```
SELECT * FROM tab1,tab2
WHERE tab1.col1 = tab2.col1
Estimated Cost: 125
Estimated # of Rows Returned: 510
1) lsuto.tab2: SEQUENTIAL SCAN
2) lsuto.tab1: SEQUENTIAL SCAN
DYNAMIC HASH JOIN
    Dynamic Hash Filters: lsuto.tab2.col1 = lsuto.tab1.col1
```

但是，执行查询之前执行 SET OPTIMIZATION FIRST_ROWS 时，数据库服务器将使用嵌套循环连接。以下 SET EXPLAIN 部分输出中的子句 (FIRST_ROWS OPTIMIZATION) 显示优化器对查询使用了用户响应时间优化：

```
QUERY:        (FIRST_ROWS OPTIMIZATION)
------
SELECT * FROM tab1,tab2
WHERE tab1.col1 = tab2.col1
Estimated Cost: 145
Estimated # of Rows Returned: 510
1) lsuto.tab1: SEQUENTIAL SCAN
2) lsuto.tab2: INDEX PATH
    (1) Index Keys: col1
        Lower Index Filter: lsuto.tab2.col1 = lsuto.tab1.col1
NESTED LOOP JOIN
```

表扫描与索引扫描
在数据库服务器从表中返回大量行的情况下，对于总查询时间目标而言开销较低的选项可能是扫描表，而不是使用索引。但是，检索首行时，对于用户响应时间目标而言开销较低的选项可能是使用索引来访问表。
使用分段索引排序
索引未分段时，数据库服务器可以使用索引来避免排序。但是，当索引已分段时，那么只能保证分段内排序，而不能保证分段之间排序。

通常，对于总查询时间目标而言成本最低的选择是并行扫描分段，然后使用并行排序来生成正确的顺序。但是，此选择对用户相应时间的优化目标无益。

反之，如果用户响应时间更重要，那么数据库服务器会并行读取索引分段，并且合并所有分段的数据。通常无需附加排序。

相关链接

避免或简化排序操作 在第289页

优化用户定义的数据类型的查询

访问用户定义的数据类型 (UDT) 的查询可以利用内置数据类型所使用的相同性能特征。

这些特征为：

- 索引

  如果查询访问的行较少，索引会加快检索，因为数据库服务器不需要读取表中的每一页来找到行。有关更多信息，请参阅用户定义的数据类型上的索引 在第166页。
- 并行数据库查询 (PDQ)

  访问用户定义的数据的查询可以利用并行扫描和并行执行。

  要打开查询的并行执行，可设置 PDQPRIORITY 环境变量或使用 SQL 语句 SET PDQPRIORITY。有关如何设置 PDQ 优先级和影响 PDQ 的配置参数的更多信息，请参阅为并行数据库查询分配资源 在第255页。
- 优化器指令

此外，程序员还可以编写以下函数或者 UDR 以帮助优化器为查询创建有效的查询计划：

- 可以利用并行数据库查询的并行 UDR

- 用户定义的选择性函数，用于计算满足函数条件的行的预期比例
- 用户定义的成本函数，用于计算执行用户定义例程的预期相对成本
- 用户定义的统计信息函数，UPDATE STATISTICS 语句可使用该函数生成统计信息和数据分布
- 用户定义的否定函数，为优化器提供了更多选择

以下各节总结了这些技术。有关如何编写和注册用户定义的选择性函数和用户定义的成本函数的更完整描述，请参阅《*SinoDB*® 用户自定义例程和数据类型开发者指南》。

并行 UDR

执行 UDR 的一种方法是使用查询中的表达式。如果 UDR 在查询的表达式中，那么您可以利用并行执行。

对于并行执行，UDR 必须为以下查询的其中一部分：

- WHERE 子句
- SELECT 列表
- GROUP by 列表
- 超负载的比较运算符
- 用户定义的聚合
- HAVING 子句
- 并行插入语句的 Select 列表
- 多个索引分段上的类属 B 型树索引扫描（如果 B 型树索引扫描中使用的比较函数可并行化）

例如，假设您创建一个不透明数据类型 circle、一个定义列类型为 circle 的表 cir_t 以及一个用户定义例程 area，然后执行以下查询示例：

```
SELECT circle, area(circle)
   FROM cir_t
   WHERE circle > "(6,2,4)";
```

在此查询示例中，以下操作可以并行执行：

- SELECT 列表中的 UDR area(circle)

  如果表 cir_t 已分段，那么多个 area UDR 可以按每个分段一个 UDR 的方式并行执行。
- WHERE 子句中的表达式 circle > "(6,2,4)"

  如果表 cir_t 已分段，那么表的多个扫描可以按每个分段一个扫描的方式并行执行。而且，多个 ">" 比较运算符可以按每个分段一个运算符的方式并行执行。

缺省情况下，UDR 不会并行执行。要启用 UDR 的并行执行，必须执行以下操作：

- 在 CREATE FUNCTION 或 ALTER FUNCTION 语句中指定 PARALLELIZABLE 修饰符。
- 确保 UDR 不调用非 PDQ 线程安全的函数。
- 启用 PDQ 优先级。
- 在并行数据库查询中使用 UDR。

选择性和成本函数

您可以使用 CREATE FUNCTION 语句来创建 UDR。然后，您可以使用例程修饰符来更改在该语句中指定的成本或选择性。

创建 UDR 之后，您可以将其置于 SQL 语句中。

以下示例展示了可以如何将 UDR 置于 SQL 语句中：

```
SELECT * FROM image
WHERE get_x1(image.im2) and get_x2(image.im1)
```

如果没有其他信息，优化器将无法精确估算执行 UDR 的成本。您可以向优化器提供函数的成本和选择性。数据库服务器将成本和选择性一起使用，以确定最佳路径。有关选择性的更多信息，请参阅使用用户定义例程的过滤器 在第266页。

在上述示例中，优化器无法确定先执行哪个函数，是先执行 get_x1 函数，还是 get_x2 函数。如果函数执行成本太高，那么 DBA 可以为函数分配较高的成本或选择性，这可以影响优化器更改查询计划以获取更好的性能。在上述示例中，如果 get_x1 的执行成本更高，DBA 可以为该函数分配更高的成本，这会使得优化器先执行 get_x2 函数。

您可以将以下例程修饰符添加到 CREATE FUNCTION 语句，以更改优化器分配给函数的成本或选择性：

- selfunc=*function_name*
- selconst=*integer*
- costfunc=*function_name*
- percall_cost=*integer*

有关开销或选择性修饰符的更多信息，请参阅《*SinoDB*® 用户自定义例程和数据类型开发者指南》。

UDT 的用户定义的统计信息

由于数据库服务器无法获得关于用户定义的数据类型 (UDT) 的性质和使用方面的信息，因而它不能为 UDT 收集分布或 colmin 和 colmax 信息（可在 syscolumns 系统目录表中找到）。相反，您可以创建一个特殊的函数来植入这些统计信息。

执行 UPDATE STATISTICS 时，数据库服务器将执行统计信息收集函数。

有关更新统计信息的重要性的更多信息，请参阅为表和索引保留的统计信息 在第224页。有关提高性能的信息，请参阅为具有用户定义的数据类型的列更新统计信息 在第278页。

否定函数

否定函数与其伴随函数采用相同的参数，参数顺序也相同，但返回的是 Boolean 补数。也就是说，如果一个函数对给定参数集返回 TRUE，向其否定函数传递了同样顺序的相同参数后将返回 FALSE。

在某些情况下，如果将查询的内容进行颠倒，那么数据库服务器可以更高效地处理该查询。比如说，“x 大于 y 吗？”更改为“y 小于或等于 x 吗？”。

使用 SQL 语句高速缓存优化查询

在数据库服务器执行 SQL 语句之前，必须先对语句进行语法分析和优化。优化语句可能会很耗时，这取决于 SQL 语句的大小。

数据库服务器可以将已优化的 SQL 语句存储于共享内存的虚拟部分，该区域称为 SQL 语句高速缓存。SQL 语句高速缓存 (SSC) 可由所有用户进行访问，并且允许用户在执行查询之前忽略优化步骤。此功能可以在以下几方面显著提高性能：

- 当用户执行相同 SQL 语句时减少响应时间。

  对于需要花费较长时间进行优化的 SQL 语句（通常是因为其在 WHERE 子句中包含许多表和许多过滤器），从 SQL 语句高速缓存中运行将更快，因为数据库服务器不会优化语句。
- 更少的内存使用量，因为数据库服务器在用户之间共享查询数据结构。

  当语句在选择列表中有许多列名时，使用 SQL 语句高速缓存可以减少更多的内存。

有关 SQL 语句高速缓存对整个系统的性能影响的更多信息，请参阅监视和调整 *SQL* 语句高速缓存 在第73页。

何时使用 SQL 语句高速缓存

如果多个用户执行相同的 SQL 语句，那么应用程序可能会从使用 SQL 语句高速缓存中获益。如果语句中的所有字符都完全匹配，数据库服务器会将其视为相同的语句。

例如，设想有 50 个销售代表全天执行 add_order 应用程序，如果该应用程序中包含使用主变量的 SQL 语句，那么他们都执行相同的 SQL 语句，如以下示例所示：

```
SELECT * FROM ORDERS WHERE order_num = :hostvar
```

此种应用程序会从使用 SQL 语句高速缓存中获益，因为用户在 SQL 语句高速缓存中很容易找到该 SQL 语句。

对于以下 SQL 语句，数据库服务器认为它们不能精确匹配，因为它们在 WHERE 子句中包含的文字值有所不同：

```
SELECT * FROM customer, orders
   WHERE customer.customer_num = orders.customer_num
   AND order_date > "01/01/07"
SELECT * FROM customer, orders
   WHERE customer.customer_num = orders.customer_num
   AND order_date > "01/01/2007"
```

在以下情况下，使用 SQL 语句高速缓存不会提高性能：

- 如果某个报表应用程序每晚运行一次，而且没有其他应用程序使用其执行的 SQL 语句，那么不会从使用语句高速缓存中获益。
- 如果某个应用程序准备好一个语句，然后多次执行该语句，那么使用 SQL 语句高速缓存将不会提高性能，因为该语句在 PREPARE 语句中只优化一次。

如果语句包含主变量，那么数据库服务器在将该语句存储到 SQL 语句高速缓存中时会使用占位符替换主变量。因此，优化该语句时无需数据库服务器访问主变量的值。在某些情况下，由于为某一列存储的分布信息会精确地告知优化器通过过滤器的行数，因此如果数据库服务器访问主变量的值，可能会以另外的方式优化该语句。

如果包含主变量的 SQL 语句在 SQL 语句高速缓存打开的情况下执行性能很低，那么可尝试使用 onmode -e flush 命令清空 SQL 语句高速缓存，并尝试使用在查询的多次执行中使用频率更高的值执行该查询。清空高速缓存时，数据库服务器会重新优化查询，并生成针对这些频繁使用的值进行优化的查询计划。

> 重要： 只有在某个语句未被使用时，数据库服务器才能从 SQL 语句高速缓存中清空该语句条目。如果某一应用程序准备好语句并将其继续保留，那么该条目仍在使用。在此情况下，应用程序需要先关闭该语句，才能通过清空提高性能。

使用 SQL 语句高速缓存

通常 DBA 决定是否启用 SQL 语句高速缓存。如果 SQL 语句高速缓存启用，那么各个用户可以决定对于其特定的环境或应用程序是否使用 SQL 语句高速缓存。

因为数据库服务器在管理 SQL 语句高速缓存方面会产生一些处理开销，所以您应该仅在多个用户希望共享 SQL 语句时使用 SQL 语句高速缓存。

要启用 SQL 语句高速缓存，请将 STMT_CACHE 配置参数设置为可定义以下某种方式的值：

- 除非用户明确指定不使用，否则始终使用 SQL 语句高速缓存。
- 只有在用户明确指定使用时，才使用 SQL 语句高速缓存。

有关更多信息，请参阅启用 *SQL* 语句高速缓存 在第296页。有关 STMT_CACHE 配置参数的更多信息，请参阅《*SinoDB*® 管理员参考》。

启用 SQL 语句高速缓存

STMT_CACHE 配置参数为 0（缺省值）时，数据库服务器将不使用 SQL 语句高速缓存。有两种修改此值的方式来启用 SQL 语句高速缓存。

使用以下其中一种方法可更改该 STMT_CACHE 的缺省值：

- 更新 ONCONFIG 文件以指定 STMT_CACHE 配置参数，并重新启动数据库服务器。

  如果将 STMT_CACHE 配置参数设置为 1，那么数据库服务器将在单个用户将 STMT_CACHE 环境变量设置为 1 或在应用程序中执行 SET STATEMENT CACHE ON 语句时，为该用户使用 SQL 语句高速缓存。

  ```
  STMT_CACHE 1
  ```

如果 STMT_CACHE 配置参数为 2，那么数据库服务器将所有用户的 SQL 语句存储到 SQL 语句高速缓存中，除非单个用户使用 STMT_CACHE 环境变量或 SET STATEMENT CACHE OFF 语句关闭该功能。

```
STMT_CACHE 2
```

- 使用 onmode -e 命令动态地覆盖 STMT_CACHE 配置参数。

  如果使用 enable 关键字，那么在单个用户将 STMT_CACHE 环境变量设置为 1 或在应用程序中执行 SET STATEMENT CACHE ON 语句时，数据库服务器将为该用户使用 SQL 语句高速缓存。

  ```
  onmode -e enable
  ```

  如果使用 on 关键字，数据库服务器将所有用户的 SQL 语句存储到 SQL 语句高速缓存，除非单个用户通过 STMT_CACHE 环境变量或 SET STATEMENT CACHE OFF 语句关闭该项功能。

  ```
  onmode -e on
  ```

根据 STMT_CACHE 配置参数的设置（或 onmode -e 的执行），下表总结了 SQL 语句高速缓存的用法，以及在 STMT_ CACHE 环境变量的应用程序和 SET STATEMENT CACHE 语句中的用法。

STMT_ CACHE 配置参数或 onmode -e	STMT_CACHE 环境变量	SET STATEMENT CACHE 语句	导致的行为
0（缺省值）	不适用	不适用	不使用语句高速缓存
1	0（或未设置）	OFF	不使用语句高速缓存
1	1	OFF	不使用语句高速缓存
1	0（或未设置）	ON	使用语句高速缓存
1	1	ON	使用语句高速缓存
1	1	未执行	使用语句高速缓存
1	0	未执行	不使用语句高速缓存
2	1（或未设置）	ON	使用语句高速缓存
2	1（或未设置）	OFF	不使用语句高速缓存
2	0	ON	使用语句高速缓存
2	0	OFF	用户不使用语句高速缓存
2	0	未执行	用户不使用语句高速缓存
2	1（或未设置）	未执行	用户不使用语句高速缓存

将语句置于高速缓存中
可以将 SELECT、UPDATE、INSERT 和 DELETE 语句放到 SQL 语句高速缓存中，但也有些例外。当数据库服务器检查某个 SQL 语句是否在高速缓存中时，它必须找到该语句的精确匹配。

有关异常的完整列表和精确匹配的需求列表，请参阅《*SinoDB® SQL* 指南*:* 语法》中的 SET STATEMENT CACHE 部分。

监视每个会话的内存使用情况

可以使用多个 onstat -g 命令选项来获取每个会话的内存信息。

可以通过标识使用大量内存的 SQL 语句来获取内存信息。

要标识使用大量内存的 SQL 语句：

1. 执行 onstat -g ses 命令来显示所有会话的内存，并查看具有最高内存使用率的会话。

2. 执行 onstat -g ses *session-id* 命令来显示有关具有最高内存使用率的会话的更多详细信息。
3. 执行 onstat -g stm *session-id* 命令来显示 SQL 语句使用的内存。

显示所有用户线程和会话内存使用情况
使用 onstat -g ses 命令来显示所有用户会话，并按会话标识符显示内存使用率。

会话共享 SSC 中的内存结构时，used memory 列中的值应低于高速缓存关闭时的值。例如，图 *73: 未启用 SQL 语句高速缓存时的 onstat -g ses 输出* 在第298页 显示 SQL 语句高速缓存未启用时的 onstat -g ses 输出示例，而图 *74: 启用 SQL 语句高速缓存时的 onstat -g ses 输出* 在第298页显示 SQL 语句高速缓存启用后，会话 4 中的查询再次执行时的输出。图 *73: 未启用 SQL 语句高速缓存时的 onstat -g ses 输出* 在第298页显示会话 4 使用的内存量为 45656 字节。图 *74: 启用 SQL 语句高速缓存时的 onstat -g ses 输出* 在第298页显示启用 SQL 语句高速缓存时，会话 4 使用的字节将变少 (36920)。

```
session                                          #RSAM    total    used     dynamic
id       user     tty      pid      hostname    threads  memory   memory   explain
12       informix -        0        -           0        12288    7632     off
4        informix 11       5158     smoke       1        53248    45656    off
3        informix -        0        -           0        12288    8872     off
2        informix -        0        -           0        12288    7632     off
```

图 73: 未启用 SQL 语句高速缓存时的 onstat -g ses 输出

```
session                                          #RSAM    total    used     dynamic
id       user     tty      pid      hostname    threads  memory   memory   explain
17       informix -        0        -           0        12288    7632     off
16        informix 12       5258     smoke       1         40960     38784      off
4         informix 11       5158     smoke       1        53248    36920      off
3        informix -        0        -           0        12288    8872     off
2        informix -        0        -           0        12288    7632     off
```

图 74: 启用 SQL 语句高速缓存时的 onstat -g ses 输出

图 *74: 启用 SQL 语句高速缓存时的 onstat -g ses 输出* 在第298页也显示了分配给会话 16 使用的内存，该会话与会话 4 执行相同的 SQL 语句。会话 16 分配的总内存 (40960) 和使用的内存 (38784) 均比会话 4 少（图 *73: 未启用 SQL 语句高速缓存时的 onstat -g ses 输出* 在第298页分别显示为 53248 和 45656），因为会话 16 使用了 SQL 语句高速缓存中现有的内存结构。

显示详细会话信息和内存使用情况
使用 onstat -g ses *session-id* 命令显示会话的详细信息，包括内存使用率。

以下 onstat -g ses *session-id* 输出列显示内存使用情况：

- 输出的 Memory pools 部分
  - totalsize 列显示当前分配的字节数
  - freesize 列显示未分配的字节数
- 输出的最后一行显示从 sscpool 分配的字节数。

图 *75: onstat -g ses session-id 输出* 在第298页显示会话 16 当前分配 69632 字节，其中 11600 字节分配自 sscpool。

```
onstat -g ses 14


session                                          #RSAM    total      used
id       user     tty      pid      hostname    threads  memory     memory
14       virginia 7        28734    lyceum      1        69632      67384

tid      name     rstcb    flags    curstk   status
38       sqlexec  a3974d8  Y--P---  1656     cond wait(netnorm)
```

```
Memory pools    count 1
name         class addr     totalsize freesize #allocfrag #freefrag
14           V     a974020  69632     2248     156        2

...
Sess  SQL            Current            Iso Lock       SQL  ISAM F.E.
Id    Stmt type      Database           Lvl Mode       ERR  ERR  Vers
14    SELECT         vjp_stores         CR  Not Wait   0    0    9.03

Current statement name : slctcur

Current SQL statement :
  SELECT C.customer_num, O.order_num FROM customer C, orders O, items I
    WHERE C.customer_num = O.customer_num AND O.order_num = I.order_num

Last parsed SQL statement :
  SELECT C.customer_num, O.order_num FROM customer C, orders O, items I
    WHERE C.customer_num = O.customer_num AND O.order_num = I.order_num

11600 byte(s) of memory is allocated from the sscpool
```

图 75: onstat -g ses session-id 输出

显示有关会话 SQL 语句的信息
使用 onstat -g sql *session-id* 或 onstat -g spf 命令来显示有关会话执行的 SQL 语句的信息。

下图表明 onstat -g sql *session-id* 显示的信息与 图 *75: onstat -g ses session-id* 输出 在第298页中 onstat -g ses *session-id* 命令底部的信息相同，包括从 sscpool 分配的字节数。

```
onstat -g sql 14


Sess  SQL            Current            Iso Lock       SQL  ISAM F.E.
Id    Stmt type      Database           Lvl Mode       ERR  ERR  Vers
14    SELECT         vjp_stores         CR  Not Wait   0    0    9.03

Current statement name : slctcur

Current SQL statement :
  SELECT C.customer_num, O.order_num FROM customer C, orders O, items I
    WHERE C.customer_num = O.customer_num AND O.order_num = I.order_num

Last parsed SQL statement :
  SELECT C.customer_num, O.order_num FROM customer C, orders O, items I
    WHERE C.customer_num = O.customer_num AND O.order_num = I.order_num

11600 byte(s) of memory is allocated from the sscpool
```

图 76: onstat -g sql session-id 输出

显示有关 SQL 语句在会话中使用的内存的信息
使用 onstat -g stm *session-id* 来显示有关会话中每个 SQL 语句使用的内存的信息。

下图为与 图 *75: onstat -g ses session-id* 输出 在第298页 中的 onstat -g ses *session-id* 和 图 *76: onstat -g sql session-id* 输出 在第299页中的 onstat -g sql *session-id* 相同的会话 (14) 显示 onstat -g stm *session-id* 的输出。

启用 SQL 语句高速缓存 (SSC) 时，数据库服务器会在 SSC 池中创建堆。因此，图 77: onstat -g stm session-id 输出 在第300页中的 heapsz 输出字段显示该 SQL 语句使用了 10056 个字节，它们包含在 onstat -g sql 14 所示的 SSC 池中 11600 个字节内。

```
onstat -g stm 14


session   14 ------------------------------------------------------------
 sdblock  heapsz  statement ('*' = Open cursor)
 aa11018   10056 *SELECT C.customer_num, O.order_num
FROM customer C, orders O, items I
WHERE C.customer_num = O.customer_num
AND O.order_num = I.order_num
```

图 77: onstat -g stm session-id 输出

监视 SQL 语句高速缓存的使用情况

如果注意到一直使用 SQL 语句高速缓存的某一查询的响应时间突然增加，那么查询的高速缓存条目可能已被删除。您可以通过显示 onstat -g ssc 命令输出来监视 SQL 语句高速缓存的使用情况并检查已舍弃条目或已删除条目。

当查询依赖的对象之一发生更改而导致该查询的数据字典高速缓存条目无效时，数据库服务器将从高速缓存中删除该条目。以下操作会造成依赖性检查故障：

- 可能更改查询计划的任何数据定义语言 (DDL) 语句（例如，ALTER TABLE、DROP INDEX 或 CREATE INDEX）的执行
- 变更带参考约束（两个方向均可）关联的表
- 对查询中涉及的任何表或列执行 UPDATE STATISTICS FOR TABLE
- 使用 RENAME 语句重命名列、数据库或索引

当一个条目标记为已舍弃或已删除时，数据库服务器在下次执行该 SQL 语句时必须对其重新进行语法分析和重新优化。例如，图 78: 已舍弃条目的 onstat -g ssc 命令输出示例 在第300页显示在执行第一条和第二条 SQL 语句之间，对 items 和 orders 表执行 UPDATE STATISTICS 之后，onstat -g ssc 命令显示的条目。

图 78: 已舍弃条目的 onstat -g ssc 命令输出示例 在第300页中 onstat -g ssc 输出的 Statement Cache Entries 部分显示了 flag 字段，该字段表明某个条目是否已从 SQL 语句高速缓存中舍弃或删除。

- 第一个条目的 flag 列的值为 DF，表示该条目是完全高速缓存的，但现在已因为无效而被删除。
- 第二个条目与第三个条目的语句内容完全相同，表示在 UPDATE STATISTICS 语句之后执行该条目时对其进行了重新语法分析和重新优化。

```
onstat -g ssc

...
Statement Cache Entries:

lru hash ref_cnt hits flag heap_ptr  database     user
------------------------------------------------------------------------------
...
  2  232       1    1   DF  aa3d020  vjp_stores   virginia
  SELECT C.customer_num, O.order_num
    FROM customer C, orders O, items I
    WHERE C.customer_num = O.customer_num
    AND O.order_num = I.order_num

  3  232       1    0   -F  aa8b020  vjp_stores   virginia
  SELECT C.customer_num, O.order_num
    FROM customer C, orders O, items I
    WHERE C.customer_num = O.customer_num
    AND O.order_num = I.order_num
```

```
...
```

图 78: 已舍弃条目的 onstat -g ssc 命令输出示例

监视会话和线程

可以监视活动会话和线程数以及其正在使用的资源量。监视会话和线程对于执行查询的会话以及执行插入、更新和删除操作的会话都很重要。

可监视会话和线程的某些信息来确定应用程序使用的资源量是否比例不当。

> 注: 在会话完成之前，对于具有 PDQ 优先级设置和 GROUP BY 子句的存储过程，不会释放该过程的会话线程。

使用 onstat 命令监视会话和线程

可以使用若干 onstat 实用程序命令来监视活动的会话和线程。

使用以下 onstat 实用程序命令来监视会话和线程：

- onstat -u
- onstat -g ath
- onstat -a act
- onstat -a cpu
- onstat -a ses
- onstat -g mem
- onstat -g stm

使用 onstat -g bth 和 onstat -g BTH 命令监视块线程
执行线程拥有各种对象和资源的所有权；例如，缓冲区、锁、互斥锁、决策支持内存等。在数百或数千个线程中争用这些资源可能会导致依赖链。使用 onstat -g bth 命令来显示阻塞和等待线程之间的依赖关系。使用 onstat -g BTH 命令来显示阻塞线程的会话和堆栈信息。

例如，被阻塞等待进入临界区的线程可能拥有另一个线程正在等待的行锁。第二个线程可能阻塞了在 MGM 查询队列中等待的第三个线程。通常，这种争用的持续时间很短。但是，如果某个线程被阻塞的时间足以引起注意，那么可能需要确定争用的来源。onstat -g bth 命令会发现依赖链并按顺序显示阻塞线程，然后是等待线程。您可以使用生成的争用图来诊断和纠正问题。

onstat -g bth 命令输出的以下示例具有多个正在等待资源的线程。

```
This command attempts to identify any blocking threads.

Highest level blocker(s)
 tid      name                session
 48       sqlexec             26

Threads waiting on resources
 tid      name                blocking resource            blocker
 49       sqlexec             MGM                          48
 13       readahead_0         Condition (ReadAhead)         -
 50       sqlexec             Lock (0x4411e578)            49
 51       sqlexec             Lock (0x4411e578)            49
 52       sqlexec             Lock (0x4411e578)            49
 53       sqlexec             Lock (0x4411e578)            49
 57       bf_priosweep()      Condition (bp_cond)           -
 58       scan_1.0            Condition (await_MC1)         -
 59       scan_1.0            Condition (await_MC1)         -

Run 'onstat -g BTH' for more info on blockers.
```

图 79: onstat -g bth 命令的输出

在此示例中，有四个线程正在等待线程 49 拥有的锁。线程 49 正在等待线程 48 拥有的 MGM 资源。如果执行 onstat -g BTH 命令，那么输出会显示阻塞线程的会话和堆栈信息，在这种情况下是线程 48。

相关链接

《*SinoDB* 管理员参考》*: onstat -g bth* 和 *-g BTH:* 显示被阻挡和等待中的线程

使用 onstat -u 输出监视线程

使用 onstat -u 命令来显示有关需要数据库服务器任务控制块的活动线程的信息。

活动线程包括属于用户会话的线程，以及与数据库服务器守护程序（如，页清除程序）相对应的一些线程。图 *80: onstat -u* 输出 在第302页显示 onstat -u 输出的示例。

也会使用 onstat -u 命令来确定某个用户是否正在等待资源，或持有过多的锁，或了解该用户已执行了多少 I/O 的信息。

该实用程序输出中显示以下信息：

- 每个线程的地址
- 指示线程当前状态（例如：正在等待缓冲区或正在等待检查点）、线程是否为会话的主线程以及线程的类型（例如：用户线程、守护程序线程等）的标志

  有关这些标志的信息，请参阅《*SinoDB*® 管理员参考》。
- 线程所属会话的会话标识符和用户登录标识符

  会话标识符为 0 表示守护程序线程。
- 线程是否正在等待特定的资源以及该资源的地址
- 该线程正持有的锁数量
- 线程已执行的读调用数和写调用数
- 当前活动的最大用户线程数

如果在数据库服务器正在执行快速恢复的同时执行 onstat -u，那么显示屏上可能会出现多个数据库服务器线程。

```
Userthreads
address  flags   sessid   user     tty      wait     tout locks nreads   nwrites
80eb8c   ---P--D 0        informix -        0        0    0     33       19
80ef18   ---P--F 0        informix -        0        0    0     0        0
80f2a4   ---P--B 3        informix -        0        0    0     0        0
80f630   ---P--D 0        informix -        0        0    0     0        0
80fd48   ---P--- 45       chrisw   ttyp3    0        0    1     573      237
810460   ------- 10       chrisw   ttyp2    0        0    1     1        0
810b78   ---PR-- 42       chrisw   ttyp3    0        0    1     595      243
810f04   Y------ 10       chrisw   ttyp2    beacf8   0    1     1        0
811290   ---P--- 47       chrisw   ttyp3    0        0    2     585      235
81161c   ---PR-- 46       chrisw   ttyp3    0        0    1     571      239
8119a8   Y------ 10       chrisw   ttyp2    a8a944   0    1     1        0
81244c   ---P--- 43       chrisw   ttyp3    0        0    2     588      230
8127d8   ----R-- 10       chrisw   ttyp2    0        0    1     1        0
812b64   ---P--- 10       chrisw   ttyp2    0        0    1     20       0
812ef0   ---PR-- 44       chrisw   ttyp3    0        0    1     587      227
 15 active, 20 total, 17 maximum concurrent
```

图 80: onstat -u 输出

相关链接

《*SinoDB* 管理员参考》*: onstat -u* 命令*:* 显示用户活动的概要文件

使用 onstat -g ath 输出监视线程

使用 onstat -g ath 命令来查看所有线程的列表。与 onstat -u 命令不同，此列表包括没有数据库服务器任务控制块的内部守护程序线程。

onstat -g ath 命令显示不包括会话标识符（因为并非所有的线程都属于会话）。

status 字段包含有关线程状态的信息，例如：running、cond wait、IO Idle、IO Idle、sleeping secs: *number_of_seconds* 或 sleeping forever。以下输出示例将许多线程标识为 sleeping forever。为了提高性能，您可以移除或降低标识为 sleeping forever 的线程数。

```
Threads:
tid  tcb           rstcb        prty  status              vp-class   name
2    10bbf36a8      0            2    sleeping forever      3lio    lio vp 0
3    10bc12218      0            2    sleeping forever      4pio    pio vp 0
4    10bc31218      0            2    sleeping forever      5aio    aio vp 0
5    10bc50218      0            2    sleeping forever      6msc    msc vp 0
6    10bc7f218      0            2    sleeping forever      7aio    aio vp 1
7    10bc9e540      10b231028    4    sleeping secs: 1      1cpu    main_loop()
8    10bc12548      0            2    running               1cpu    tlitcppoll
9    10bc317f0      0            3    sleeping forever      1cpu    tlitcplst
10   10bc50438      10b231780    2    sleeping forever      1cpu    flush_sub(0)
11   10bc7f740      0            2    sleeping forever      8aio    aio vp 2
12   10bc7fa00      0            2    sleeping forever      9aio    aio vp 3
13   10bd56218      0            2    sleeping forever     10aio    aio vp 4
14   10bd75218      0            2    sleeping forever     11aio    aio vp 5
15   10bd94548      10b231ed8    3    sleeping forever      1cpu    aslogflush
16   10bc7fd00      10b232630    1    sleeping secs: 26     1cpu     btscanner 0
32   10c738ad8      10b233c38    4    sleeping secs: 1      1cpu     onmode_mon
50   10c0db710      10b232d88    2    cond wait  netnorm    1cpu     sqlexec
```

图 81: onstat -g ath 输出

由主决策支持线程启动的线程有一个名称以指示其在决策支持查询中的角色。下图显示了属于某个决策支持线程的四个扫描线程。

```
Threads:
tid  tcb           rstcb        prty  status              vp-class   name
11   994060         0            4    sleeping(Forever)     1cpu    kaio
12   994394         80f2a4       2    sleeping(secs: 51)    1cpu    btclean
26   99b11c         80f630       4    ready                 1cpu    onmode_mon
32   a9a294         812b64       2    ready                 1cpu    sqlexec
113  b72a7c         810b78       2    ready                 1cpu    sqlexec
114  b86c8c         81244c       2    cond wait(netnorm)    1cpu    sqlexec
115  b98a7c         812ef0       2    cond wait(netnorm)    1cpu    sqlexec
116  bb4a24         80fd48       2    cond wait(netnorm)    1cpu    sqlexec
117  bc6a24         81161c       2    cond wait(netnorm)    1cpu    sqlexec
118  bd8a24         811290       2    ready                 1cpu    sqlexec
119  beae88         810f04       2    cond wait(await_MC1)  1cpu    scan_1.0
120  a8ab48         8127d8       2    ready                 1cpu    scan_2.0
121  a96850         810460       2    ready                 1cpu    scan_2.1
122  ab6f30         8119a8       2    running               1cpu    scan_2.2
```

图 82: 显示属于某个决策支持线程的扫描线程的 onstat -g ath 输出

相关链接

《*SinoDB* 管理员参考》*: onstat -g ath* 命令*:* 显示所有线程的信息

提高连接性能和可伸缩性 在第48页

使用 onstat -g act 输出监视线程

使用 onstat -g act 命令获得活动线程列表。onstat -g act 输出显示同时在 onstat -g ath 输出中列出的线程子集。

有关输出示例，请参阅《*SinoDB*® 管理员参考》。

相关链接

《*SinoDB* 管理员参考》*: onstat -g act* 命令*:* 显示活动线程

使用 onstat -g cpu 输出监视线程
使用 onstat -g cpu 命令显示上次执行线程的时间、线程使用了多少 CPU 时间、线程运行的次数以及关于服务器中运行的所有线程的其他统计信息。

以下输出示例显示了正在运行的每个线程的标识符和名称、运行每个线程的虚拟处理器的标识符、每个线程上次运行的天和时间、每个线程使用的 CPU 时间量、每个线程被安排运行的次数和每个线程的状态。

```
Thread CPU Info:
tid     name              vp       Last Run          CPU Time     #scheds    status
2       lio vp 0          3lio*    07/18 08:35:35       0.0000           1    IO Idle

3       pio vp 0          4pio*    07/18 08:35:36       0.0102           2    IO Idle

4       aio vp 0          5aio*    07/18 08:35:47       0.6876          68    IO Idle

5       msc vp 0          6msc*    07/18 11:47:24       0.0935          14    IO Idle

6       main_loop()       1cpu*    07/18 15:02:43       2.9365       23350    sleeping secs: 1

7       soctcppoll        7soc*    07/18 08:35:40       0.1150           1    running

8       soctcpio          8soc*    07/18 08:35:40       0.0037           1    running

9       soctcplst         1cpu*    07/18 11:47:24       0.1106          10    sleeping forever

10      soctcplst         1cpu*    07/18 08:35:40       0.0103           6    sleeping forever

11      flush_sub(0)      1cpu*    07/18 15:02:43       0.0403       23252    sleeping secs: 1

12      flush_sub(1)      1cpu*    07/18 15:02:43       0.0423       23169    sleeping secs: 1

13      flush_sub(2)      1cpu*    07/18 15:02:43       0.0470       23169    sleeping secs: 1

14      flush_sub(3)      1cpu*    07/18 15:02:43       0.0407       23169    sleeping secs: 1

15      flush_sub(4)      1cpu*    07/18 15:02:43       0.0307       23169    sleeping secs: 1

16      flush_sub(5)      1cpu*    07/18 15:02:43       0.0323       23169    sleeping secs: 1

17      flush_sub(6)      1cpu*    07/18 15:02:43       0.0299       23169    sleeping secs: 1

18      flush_sub(7)      1cpu*    07/18 15:02:43       0.0314       23169    sleeping secs: 1

19      kaio              1cpu*    07/18 14:56:42       1.4560     2375587    IO Idle

20      aslogflush        1cpu*    07/18 15:02:43       0.0657       23166    sleeping secs: 1

21      btscanner_0       1cpu*    07/18 15:00:53       0.0484         784    sleeping secs:
 61
37      onmode_mon        1cpu*    07/18 15:02:43       0.3467       23165    sleeping secs: 1

43      dbScheduler       1cpu*    07/18 14:58:14       1.6613         320    sleeping secs:
 31
44      dbWorker1         1cpu*    07/18 13:48:10       0.4264         399    sleeping forever

45      dbWorker2         1cpu*    07/18 14:48:11       1.9346        2936    sleeping forever

94      bf_priosweep()    1cpu*    07/18 15:01:42       0.0431          77    cond wait
 bp_cond
```

图 83: onstat -g cpu 命令输出

相关链接

《*SinoDB* 管理员参考》*: onstat -g cpu:* 显示运行时统计信息

使用 onstat -g ses 输出监视会话资源
使用 onstat -g ses 命令可以监视分配给会话并由其使用的资源，特别是正在运行决策支持查询的会话。onstat -g ses 命令也会显示最近终止的会话的信息。

例如，在 图 *84: onstat -g ses* 输出 在第305页中，编号为 49 的会话正在为决策支持查询运行 5 个线程。

```
session                                     #RSAM    total      used
id       user     tty      pid    hostname threads  memory     memory
57       informix -        0      -        0        8192       5908
56       user_3   ttyp3    2318   host_1   1        65536      62404
55       user_3   ttyp3    2316   host_1   1        65536      62416
54       user_3   ttyp3    2320   host_1   1        65536      62416
53       user_3   ttyp3    2317   host_1   1        65536      62416
52       user_3   ttyp3    2319   host_1   1        65536      62416
51       user_3   ttyp3    2321   host_1   1        65536      62416
49       user_1   ttyp2    2308   host_1   5        188416     178936
2        informix -        0      -        0        8192       6780
1        informix -        0      -        0        8192       4796

Last 20 Sessions Terminated

Ses ID  Username  Hostname  PID    Time               Reason
36      user_1    host_1    2122   01/19/2015.15:20   session limit txn time (60s)
40      user_1    host_1    2134   01/19/2015.15:14   session limit memory (5124 KB)
47      user_1    host_1    2140   01/19/2015.15:04   session limit logspace (10242 KB)
50      user_1    host_1    2145   01/19/2015.15:02   session limit txn time (39548 KB)
```

图 84: onstat -g ses 输出

相关链接

《*SinoDB* 管理员参考》*: onstat -g ses* 命令*:* 显示会话相关信息

使用 onstat -g mem 和 onstat -g stm 输出监视会话内存
使用 onstat -g mem 和 onstat -g stm 命令来获取有关每个会话所用内存的信息。

您可以通过 onstat -g ses 输出的 used memory 列来确定要关注哪个会话。

图 *85:* 用于确定会话内存的 *onstat -g mem* 和 *onstat -g stm* 在第305页 显示 onstat -g ses 输出的示例和会话 16 的某些 onstat -g mem 和 onstat -g stm 输出。

- onstat -g mem 命令的输出显示每个会话所用的内存总量。

  onstat -g mem 16 输出的 totalsize 列显示了分配给会话的内存总量。
- onstat -g stm 命令的输出显示分配给当前准备的 SQL 语句的总内存部分。

  下图中 onstat -g stm 16 输出的 heapsz 列显示为当前准备的 SQL 语句分配的内存量。

```
onstat -g ses

session                                     #RSAM    total      used
id       user     tty      pid      hostname threads  memory     memory
18       informix -        0        -        0        12288      8928
17       informix 12       28826    lyceum   1        45056      33752
16       virginia 6        28743    lyceum   1        90112      79504
14       virginia 7        28734    lyceum   1        45056      33096
3        informix -        0        -        0        12288      10168
2        informix -        0        -        0        12288      8928



onstat -g mem 16
```

```
Pool Summary:
name         class addr     totalsize freesize #allocfrag #freefrag
16           V     a9ea020  90112     10608    159        5
...


onstat -g stm 16

session   16 ------------------------------------------------------------
 sdblock  heapsz  statement ('*' = Open cursor)
 aa0d018   10056 *SELECT C.customer_num, O.order_num
FROM customer C, orders O, items I
WHERE C.customer_num = O.customer_num
AND O.order_num = I.order_num
```

图 85: 用于确定会话内存的 onstat -g mem 和 onstat -g stm

相关链接

《SinoDB 管理员参考》: onstat -g lap 命令: 显示轻量追加状态信息

《SinoDB 管理员参考》: onstat -g mem 命令: 显示池内存统计信息

使用 SMI 表监视会话和线程

可以使用 syssessions 和 syssesprof 系统监视接口 (SMI) 表来获取有关会话和线程的信息。

查询 syssessions 表以获得以下信息。

sid
: 会话标识符

username
: 用户的名称（登录标识符）

uid
: 用户标识符

pid
: 进程标识符

connected
: 会话启动的时间

feprogram
: 可执行程序或应用程序的绝对路径

此外，某些列中包含标志，这些标志显示以下信息：

- 会话的主线程是否正在等待锁存器、锁、日志缓冲区或事务
- 线程是否处于临界区中。

> 重要: syssessions 表中的信息由会话组织，onstat -u 输出中的信息由线程组织。而且，与 onstat -u 输出不同，syssessions 表不包括有关守护程序线程的信息，而只包括有关用户线程的信息。

查询 syssesprof 表获得会话活动概要文件。此表为每个会话包含一行，该行的各列存储会话活动统计信息（例如：持有的锁数、写入的行数、提交次数、删除次数等）。

有关 syssessions 列的完整列表以及 syssesprof 列的描述，请参阅《*SinoDB*® 管理员参考》中有关 sysmaster 数据库的章节。

监视事务

可以监视事务以跟踪打开的事务以及这些事务所持有的锁。可以使用若干 onstat 实用程序选项来查看事务、锁和会话统计信息。

以下 onstat 命令行选项用于显示会话信息。

监视对象	显示以下命令的输出	请参阅
事务统计信息	onstat -x	显示有关事务的信息 在第307页
用户会话统计信息	onstat -u	显示有关用户会话的统计信息 在第309页
锁统计信息	onstat -k	显示有关事务锁的信息 在第308页
运行 SQL 语句的会话	onstat -g sql *session-id*	显示有关执行 *SQL* 语句的会话的统计信息 在第309页

显示有关事务的信息

onstat -x 命令的输出包含可用于监视事务的信息。

onstat -x 输出为每个打开的事务包含以下信息：

- 共享内存中的事务结构地址
- 指示以下信息的标志：
  - 事务的当前状态（已连接的用户线程、暂挂的用户线程和正在等待回滚的用户线程）
  - 事务正在运行的方式（松耦合方式或紧耦合方式）
  - 事务所处的阶段（BEGIN WORK、准备提交、正在提交、已提交或正在回滚）
  - 事务的性质（全局事务、协调程序、从属事务、既是协调程序又是从属事务）
- 拥有该事务的线程
- 事务持有的锁数量
- 对 BEGIN WORK 记录进行日志记录的逻辑日志文件
- 当前逻辑日志标识符和位置
- 隔离级别
- 尝试启动恢复线程的次数
- 事务的协调程序（如果从属事务正在执行事务）
- 自上次启动数据库服务器以来执行的最大并发事务数

onstat 实用程序对于监视全局事务特别有用。例如，您可以确定一个事务在松耦合方式还是紧耦合方式下执行。这些事务方式具有以下特征：

- 松耦合方式

  在全局事务中每一分支均具有单独的事务标识符 (XID)。此方式是缺省方式。

  - 不同的数据库服务器可协调事务，但不共享资源。即使两个事务分支访问同一个数据库，它们也不能共享锁定。
  - 全局事务所有分支的记录在逻辑日志中都显示为单独的事务。
- 紧耦合方式

  在单个全局事务中，访问同一数据库的所有分支均共享相同的事务标识符 (XID)。只有使用 Microsoft™ Transaction Server (MTS) 事务管理器时，才使用该方式。

  - 不同的数据库服务器可协调事务，并共享诸如锁定和日志记录之类的资源。具有相同 XID 的分支共享锁定，且永远不会与具有相同 XID 的其他分支相遇，因为一次只有一个分支处于活动状态。
  - 具有相同 XID 的分支的日志记录将出现在逻辑日志中同一个事务下。

图 *86: onstat -x 输出* 在第308页显示 onstat -x 输出的示例。列出的最后一个事务是全局事务，如 flags 列的第五个位置中的 G 值所示。flags 列第二个位置的 T 值显示该事务运行于紧耦合方式。

```
Transactions
address flags userthread locks beginlg curlog logposit isol    retrys coord
ca0a018 A---- c9da018    0     0       5      0x18484c COMMIT  0
ca0a1e4 A---- c9da614    0     0       0      0x0      COMMIT  0
ca0a3b0 A---- c9dac10    0     0       0      0x0      COMMIT  0
ca0a57c A---- c9db20c    0     0       0      0x0      COMMIT  0
ca0a748 A---- c9db808    0     0       0      0x0      COMMIT  0
ca0a914 A---- c9dbe04    0     0       0      0x0      COMMIT  0
ca0aae0 A---- c9dcff8    1     0       0      0x0      COMMIT  0
ca0acac A---- c9dc9fc    1     0       0      0x0      COMMIT  0
ca0ae78 A---- c9dc400    1     0       0      0x0      COMMIT  0
ca0b044 AT--G c9dc9fc    0     0       0      0x0      COMMIT  0
10 active, 128 total, 10 maximum concurrent
```

图 86: onstat -x 输出

图 *86: onstat -x 输出* 在第308页中的输出显示该事务分支正持有 13 个锁。当一个事务运行于紧耦合方式时，该事务的分支将共享锁定。

显示有关事务锁的信息

onstat -k 命令的输出包含关于事务持有的锁的详细信息。

要找到相关锁，请将 onstat -x 输出的 userthread 列中的地址与 onstat -k 输出的 owner 列中的地址进行匹配。

图 *87: onstat -k 和 onstat -x 输出* 在第308页显示 onstat -x 和相应的 onstat -k 命令的输出示例。onstat -x 输出的 userthread 列中 a335898 值与 onstat -k 输出的两行的 owner 列中的值相匹配。

```
onstat -x

Transactions
address  flags userthread locks   beginlg curlog  logposit   isol    retrys coord
a366018  A---- a334018    0      0       1       0x22b048  COMMIT  0
a3661f8  A---- a334638    0      0       0       0x0       COMMIT  0
a3663d8  A---- a334c58    0      0       0       0x0       COMMIT  0
a3665b8  A---- a335278    0      0       0       0x0       COMMIT  0
a366798  A---- a335898    2      0       0       0x0       COMMIT  0
a366d38  A---- a336af8    0      0       0       0x0       COMMIT  0
 6 active, 128 total, 9 maximum concurrent

onstat -k

Locks
address  wtlist   owner    lklist   type     tblsnum  rowid    key#/bsiz
a09185c  0        a335898  0        HDR+S    100002   20a        0
a0918b0  0        a335898  a09185c  HDR+S    100002   204        0
 2 active, 2000 total, 2048 hash buckets, 0 lock table overflows
```

图 87: onstat -k 和 onstat -x 输出

在 图 *87: onstat -k 和 onstat -x 输出* 在第308页的示例中，用户正在从两个表中选择一行。该用户持有以下锁：

- 在一个数据库上的共享锁
- 在另一个数据库上的共享锁

显示有关用户会话的统计信息

onstat -u 命令的输出包含关于用户会话的统计信息。

通过将 onstat -x 输出的 userthread 列与 onstat -u 输出的 address 列进行匹配可以找到事务的会话标识符。onstat -u 输出中同一行的 sessid 列提供了该会话标识符。

例如，图 *88: 在 onstat -x 输出中获取用户线程的会话标识符* 在第309页显示 onstat -x 输出的 userthread 列中的地址为 a335898。具有相同地址的 onstat -u 中的输出行显示 sessid 列中的会话标识符 15。

```
onstat -x

Transactions
address  flags userthread locks   beginlg curlog  logposit   isol    retrys coord
a366018  A---- a334018    0      0       1       0x22b048  COMMIT  0
a3661f8  A---- a334638    0      0       0       0x0       COMMIT  0
a3663d8  A---- a334c58    0      0       0       0x0       COMMIT  0
a3665b8  A---- a335278    0      0       0       0x0       COMMIT  0
a366798  A---- a335898    2      0       0       0x0       COMMIT  0
a366d38  A---- a336af8    0      0       0       0x0       COMMIT  0
 6 active, 128 total, 9 maximum concurrent


onstat -u

address  flags    sessid  user     tty     wait     tout locks nreads   nwrites
a334018  ---P--D 1        informix -        0        0    0     20       6
a334638  ---P--F 0        informix -        0        0    0     0        1
a334c58  ---P--- 5        informix -        0        0    0     0        0
a335278  ---P--B 6        informix -        0        0    0     0        0
a335898  Y--P--- 15       informix 1        a843d70  0    2     64       0
a336af8  ---P--D 11       informix -        0        0    0     0        0
 6 active, 128 total, 17 maximum concurrent
```

图 88: 在 onstat -x 输出中获取用户线程的会话标识符

对于以松耦合方式执行的事务，onstat -u 输出中 flags 列的第一个位置可能会显示 T 值。该 T 值表示全局事务中的一个分支正在等待其他分支完成。如果全局事务中有两个不同的分支使用同一个数据库并试图同时对同一个全局事务进行操作，那么就会出现这种情况。

对于在紧耦合方式下执行的事务，此 T 值不会出现，因为对于在全局事务中访问相同数据库的所有分支在数据库服务器中均共享一个事务结构。 一次只能有一个分支处于已连接的活动状态，并且不等待锁定，因为事务拥有不同分支持有的所有锁。

显示有关执行 SQL 语句的会话的统计信息

onstat -g sql 命令的输出包含关于会话所执行的 SQL 语句的统计信息。

要获取关于每个会话执行的最后一个 SQL 语句的信息，可使用相应的会话标识符发出 onstat -g sql 命令。

图 *89: onstat -g sql 输出* 在第309页显示此选项的输出范例，该选项使用的会话标识符与从图 *88: 在 onstat -x 输出中获取用户线程的会话标识符* 在第309页的 onstat -u 示例中获得的相同。

```
onstat -g sql 15

Sess  SQL            Current            Iso Lock       SQL  ISAM F.E.
Id    Stmt type      Database           Lvl Mode       ERR  ERR  Vers
15    SELECT         vjp_stores         CR  Not Wait   0    0    9.03

Current statement name : slctcur

Current SQL statement :
  select l.customer_num, l.lname, l.company,    l.phone, r.call_dtime,
```

```
    r.call_descr from customer l, vjp_stores@gilroy:cust_calls r where
    l.customer_num = r.customer_num

Last parsed SQL statement :
  select l.customer_num, l.lname, l.company,   l.phone, r.call_dtime,
    r.call_descr from customer l, vjp_stores@gilroy:cust_calls r where
    l.customer_num = r.customer_num
```

图 89: onstat -g sql 输出

附录
A

案例分析和示例

本附录包含本出版物所介绍性能调优方法的一个案例分析和多个示例。

磁盘过载情况的案例分析

可以标识过载的磁盘以及驻留在这些磁盘上的数据库空间。找出超负荷的磁盘后，即可更正该问题。

以下案例分析说明了一种磁盘过载的情况。此案例分析显示了根据用户提供的初始报告离析症状和找出问题时所需采取的步骤，还描述了所需的更正方法。

不具有所需吞吐量的数据库应用程序将受到检查，以便确定如何提高性能。操作系统监视工具表明很大一部分进程时间都处于等待 I/O 的闲置状态。数据库服务器管理员增加了 CPU VP 数量，以使更多的处理器可用于处理并发 I/O。但是不会增加吞吐量，这表明一个或多个磁盘已超负荷。

要验证 I/O 瓶颈，数据库服务器管理员必须标识过载的磁盘以及驻留在这些磁盘上的数据库空间。

要标识过载的磁盘以及驻留在这些磁盘上的数据库空间，请执行以下操作：

1. 要检查异步 I/O (AIO) 队列，请使用 onstat -g ioq。图 *90: onstat -g ioq* 选项的输出 在第311页显示了输出。

```
AIO I/O queues:
q name/id    len maxlen totalops  dskread dskwrite  dskcopy
opt   0      0      0        0        0        0        0
msc   0      0      0        0        0        0        0
aio   0      0      0        0        0        0        0
pio   0      0      1        1        0        1        0
lio   0      0      1      341        0      341        0
gfd   3      0      1      225        2      223        0
gfd   4      0      1      225        2      223        0
gfd   5      0      1      225        2      223        0
gfd   6      0      1      225        2      223        0
gfd   7      0      0        0        0        0        0
gfd   8      0      0        0        0        0        0
gfd   9      0      0        0        0        0        0
gfd  10      0      0        0        0        0        0
gfd  11      0     28    32693    29603     3090        0
gfd  12      0     18    32557    29373     3184        0
gfd  13      0     22    20446    18496     1950        0
```

图 90: onstat -g ioq 选项的输出

在 图 *90: onstat -g ioq* 选项的输出 在第311页中，maxlen 和 totalops 两列显示了重要的结果：

- maxlen 列显示了在队列中积累的 I/O 请求的最大储备。最后的三个队列比此列列表中的任何其他队列长得多。
- totalops 列显示了通过最后三个队列完成的 I/O 操作比此列列表中其他任何队列多 100 倍以上。

maxlen 和 totalops 列显示 I/O 负载严重失衡。

检查 I/O 活动的另一方法是使用 onstat -g iov。此选项对所有 VP 的显示不够详细。

2. 要检查与每个队列关联的每个磁盘设备的 AIO 活动，请使用 onstat -g iof，如 图 *91: onstat -g iof* 选项的部分输出 在第312页所示。

```
gfd pathname         bytes read     page reads  bytes write    page writes io/s
3  /dev/infx5      85456896       41727       207394816      101267       572.9
 op type     count          avg. time
 seeks       0              N/A
 reads       13975          0.0015
 writes      51815          0.0018
 kaio_reads  0              N/A
 kaio_writes 0              N/A
```

图 91: onstat -g iof 选项的部分输出

根据块的排列方式，可以将几个队列与同一设备关联。

3. 要确定说明 I/O 负载的数据库空间，请使用 onstat -d，如 图 *92: onstat -d* 选项的输出 在第312页所示。

```
Dbspaces
address  number   flags    fchunk   nchunks  flags    owner    name
c009ad00 1        1        1        1        N        informix rootdbs
c009ad44 2        2001     2        1        N T      informix tmp1dbs
c009ad88 3        1        3        1        N        informix oltpdbs
c009adcc 4        1        4        1        N        informix histdbs
c009ae10 5        2001     5        1        N T      informix tmp2dbs
c009ae54 6        1        6        1        N        informix physdbs
c009ae98 7        1        7        1        N        informix logidbs
c009aedc 8        1        8        1        N        informix runsdbs
c009af20 9        1        9        3        N        informix acctdbs
 9 active, 32 total

Chunks
address  chk/dbs offset   size     free     bpages   flags pathname
c0099574 1   1   500000   10000    9100               PO-   /dev/infx2
c009960c 2   2   510000   10000    9947               PO-   /dev/infx2
c00996a4 3   3   520000   10000    9472               PO-   /dev/infx2
c009973c 4   4   530000   250000   242492             PO-   /dev/infx2
c00997d4 5   5   500000   10000    9947               PO-   /dev/infx4
c009986c 6   6   510000   10000    2792               PO-   /dev/infx4
c0099904 7   7   520000   25000    11992              PO-   /dev/infx4
c009999c 8   8   545000   10000    9536               PO-   /dev/infx4
c0099a34 9   9   250000   450000   4947               PO-   /dev/infx5
c0099acc 10  9   250000   450000   4997               PO-   /dev/infx6
c0099b64 11  9   250000   450000   169997             PO-   /dev/infx7
 11 active, 32 total
```

图 92: onstat -d 选项的输出

在 Chunks 输出中，pathname 列显示磁盘设备。chk/dbs 列显示驻留在每个磁盘上的块和数据库空间的数量。在此案例中，每个过载的磁盘上只定义了一个块。每个块都与数据库空间编号 9 相关联。

Dbspaces 输出显示与每个数据库空间号相关联的数据库空间的名称。在此案例中，所有三个过载的磁盘都是 acctdbs 数据库空间的一部分。

尽管原始的磁盘配置将三个完整的磁盘都分配给 acctdbs 数据库空间，但是该数据库空间中的活动表明三个磁盘是不够的。由于三个磁盘上的负载大致相同，因此不大可能是表的布局不合理或者分段不恰当。不过，您可以将其他磁盘上的分段添加到此数据库空间中的一个或多个大型表中，也可以将一些表移动到其他负载较少的磁盘上，从而获取更好的性能。

相关链接

《*SinoDB* 管理员参考》*: onstat -g iof* 命令*:* 显示异步 *I/O* 统计信息
《*SinoDB* 管理员参考》*: onstat -g ioa* 命令*:* 显示合并的 *onstat -g* 信息
《*SinoDB* 管理员参考》*: onstat -g ioq* 命令*:* 显示 *I/O* 队列信息
《*SinoDB* 管理员参考》*: onstat -g iov* 命令*:* 显示 *AIO VP* 统计信息
《*SinoDB* 管理员参考》*: onstat -d* 命令*:* 显示块信息