《Architecture of a Database System》

（中文版）

Joseph M. Hellerstein, Michael Stonebraker and James Hamilton

翻译：林子雨

厦门大学数据库实验室

http://dblab.xmu.edu.cn

中文版网址：http://dblab.xmu.edu.cn/node/459

厦门大学计算机科学系教师 林子雨 翻译作品

http://www.cs.xmu.edu.cn/linziyu

2013 年 9 月

前言

本文翻译自经典英文论文《Architecture of a Database System》，原文作者是Joseph M. Hellerstein, Michael Stonebraker 和 James Hamilton。该论文可以作为中国各大高校数据库实验室研究生的入门读物，帮助学生快速了解数据库的内部运行机制。

本文一共包括 6 章，分别是：第 1 章概述，第 2 章进程模型，第 3 章并行体系结构：进程和内存协调，第 4 章关系查询处理器，第 5 章存储管理，第 6 章事务：并发控制和恢复，第 7 章共享组件，第 8 章结束语。

本文翻译由厦门大学数据库实验室林子雨老师团队合力完成，其中，林子雨老师负责统稿校对，刘颖杰同学负责翻译第 1 章、第 2 章和第 6 章，罗道文同学负责翻译第 3 章和第 4 章，谢荣东同学负责翻译第 5 章，蔡珉星同学负责翻译第 7 章和第 8 章，并负责对林子雨老师校对结果进行二次校对。

如果对本文翻译内容有任何疑问，欢迎联系林子雨老师。

林子雨的E-mail是：ziyulin@xmu.edu.cn。

林子雨的个人主页是：http://www.cs.xmu.edu.cn/linziyu。

厦门大学数据库实验室网站是：http://dblab.xmu.edu.cn。

本文中文版的网址是：http://dblab.xmu.edu.cn/node/459。

林子雨于厦门大学海韵园

2013 年 9 月

摘 要

数据库管理系统（DBMS）广泛存在于现代计算机系统中，并且是其重要的组成部分。它是学术界以及工业界数十年研究和发展的成果。在计算机发展史上，数据库属于最早开发的多用户服务系统之一，因此，它的研究也催生了许多为保证系统可拓展性以及稳定性的系统开发技术，这些技术如今被应用于许多其他的领域。虽然许多数据库的相关算法和概念广泛见于教科书中，但关于如何让一个数据库工作的系统设计问题却鲜有资料介绍。本文从体系架构角度探讨数据库设计的一些准则，包括处理模型、并行架构、存储系统设计、事务处理系统、查询处理及优化结构以及具有代表性的共享组件和应用。当业界有多种设计方式可供选择时，我们以当前成功的商业开源软件作为参考标准。

第 7 章 共享组件

厦门大学计算机科学系教师 林子雨 编著

个人主页：http://www.cs.xmu.edu.cn/linziyu

中文版网址：http://dblab.xmu.edu.cn/node/459

2013 年 9 月

第 7 章 共享组件

本章内容介绍一些在几乎所有的商业 DBMS 中都存在、却很少在研究文献中讨论的共享组件和工具。

7.1 目录管理器

数据库的目录管理器以元数据的格式保存了系统中数据的相关信息。目录管理器记录了系统中基本实体（用户、模式、表、列、索引等）的名称及它们的关系，并且以一系列表的形式存储在数据库中。元数据保持与数据一样的格式，可以使得系统在使用上更加紧凑、简单：用户可以使用同一种语言和工具来研究其他数据的元数据，而且，内部系统用于管理元数据的代码大部分与管理其他表的代码一样。代码和语言的复用是重要的经验，但在早期的实现中常常被忽略，这对后来的开发者来说是一个重大的遗憾。在过去的十年里，本文的其中一位作者在工业界中一再目睹这种错误的出现。

出于效率原因，我们通常采用不同的方式来处理基本的目录数据和通常的表。目录中经常被访问的部分常常根据需求物化在内存中，所采用的数据结构，一般是对目录的扁平关系结构进行非规范化处理，转换成主存中的对象网络形式。内存中缺少的数据独立性是可以接受的，因为，内存中的数据结构只供查询解析器和优化器以程序化的方式使用。额外的目录数据在解析时会被缓存在查询计划中，这些数据常常也还是采用适用于查询的非规范化格式。此外，目录表通常服务于特殊情况的事务处理技巧，用来减少事务处理中的热点（hot spots）。

目录在商业应用中能变得非常庞大。例如，一个大型 ERP（企业资源计划）应用有超过 60,000 个表，每个表有 4 到 8 列，且通常每个表有 2 或 3 个索引。

7.2 内存分配器

教材介绍的 DBMS 内存管理往往全部集中在缓冲池上。实际上，数据库系统也分配了大量的内存用于其他任务。正确地管理这些内存不仅是编程的负担，也关系到性能问题。例

如，Selinger 风格的查询优化可使用大量的内存用来建立动态编程过程中的状态。查询操作符如哈希连接、排序在运行时会分配大量内存。商业系统中的内存分配可通过使用基于上下文的内存分配器来更有效、更容易地调试。

内存的上下文是内存中的数据结构，维护了一个连续的虚拟内存的区域列表，通常称其为内存池。每个区域都可以有一个小的头部，头部包含了一个上下文标签或一个指向上下文头部结构的指针。基本的内存上下文 API 包括如下调用：

- *创建给定名字或类型的上下文。*上下文的类型可以让分配器知道如何有效地处理内存分配。例如，用于查询优化器的上下文所使用的内存数量会以较小的增量逐渐增加，而用于哈希连接的上下文是以批量的方式来分配内存的。基于这样的知识，分配器会每次选择分配更大或更小的区域。
- *分配上下文中的一块内存。*这种分配方式会返回一个指针，指向内存（很像是传统的 malloc()调用）。这部分内存可能来自上下文中已存在的区域，或者，如果任何区域都不存在这样的空间，分配器会向操作系统请求一个新区域的内存，为它做标记，并链接到上下文中。
- *删除上下文中的一块内存。*这未必能使上下文删除相应的区域。内存上下文的删除有些不寻常，更为典型的行为是删除整个上下文。
- *删除上下文。*这首先会释放所有与上下文相关联的区域，然后删除上下文头部。
- *重置上下文。*这会保留上下文，但返回原始创建时的状态，往往是通过重新分配所有之前已分配区域的内存实现的。

内存上下文提供了重要的软件工程的优势。最重要的是它们作为底层的服务，是一种程序员可控的垃圾回收方式。例如，开发人员编写的优化器可以为特定的查询在优化器上下文中分配内存，而不用担心之后如何去释放内存。当优化器选择了最佳计划，它就可以针对查询从一个单独的执行器上下文中拷贝计划到内存中，然后删除查询的优化器上下文。这就不需要编写代码来小心地遍历所有优化器数据结构来删除它们的组件。它也避免了因代码的 BUG 而引起的棘手的内存泄漏。这个特征对于具有自然的“阶段性”的查询执行过程而言，是很有用的，因为，在查询执行过程中，从解析器到优化器再到执行器的整个控制流程，每个阶段的开始都会生成上下文并伴随着大量的内存分配，每个阶段的结束都会发生上下文的删除并伴随着大量的内存回收。

需要注意的是，内存上下文实际上为开发人员提供了比大多数垃圾回收器更多的控制，因为，开发人员可以在重新分配内存时控制空间和时间的局部性。上下文机制本身提供了空

间控制，允许程序员将内存划分成多个逻辑单元。时间控制是指允许程序员在适当的时候发起上下文删除操作。相反地，垃圾回收器通常在一个程序的所有内存上工作，并且会自己决定何时运行。这是尝试用 Java 来编写服务器级别的高质量代码时会遇到的一个挫折[81]。

内存上下文也为 malloc()和 free()开销相对高的平台提供了性能优势。尤其是内存上下文可以使用关于内存怎样被分配和重新分配的语义知识（通过上下文类型），然后相应地调用调用 malloc()和 free()来最小化操作系统开销。一些数据库系统的组件（比如解析器和优化器）分配了大量的小对象，然后通过上下文删除一次性释放它们。调用 free()来释放大量的小对象在多数平台上会带来很大的开销。实际上，内存分配器可以调用 malloc()来分配大区域内存，并将该内存分配给其调用者。这样做就不再需要内存重新分配，这也就意味着，malloc()和 free()所使用的压缩逻辑（compaction logic）也就不再需要了。当上下文删除时，移除大区域内存只需要少数的 free()调用。

感兴趣的读者可浏览 PostgreSQL 的开源代码，它使用了相当先进的内存分配器。

7.2.1 为查询操作符分配内存时的注意事项

数据库厂商们为空间密集型的操作符（如哈希连接和排序）所采用的内存分配方案各有不同。一些系统（比如 DB2 for zSeries）允许 DBA（数据库管理员）控制这些操作能使用的 RAM 数量，并且保证每个查询在执行时都能获得该数量的 RAM。准入控制策略确保了这一保证。在这样的系统中，操作符通过内存分配器从堆中分配它们的内存。这些系统提供了很好的性能稳定性，但是，会强制 DBA 决定如何去平衡各个子系统（如缓冲池和查询操作符）之间的物理内存使用。

其他系统（比如 Microsoft SQL Server），从 DBA 手中接管了内存分配任务，实现了内存分配的自动化管理。这些系统尽量智能地在查询执行的多个组件中分配内存，包括缓冲池中的页面缓存和查询操作符的内存使用。用于所有这些任务的内存池即缓冲池本身。因此，在这些系统中的查询操作符通过 DBMS 实现的内存分配器从缓冲池中取得内存，并且，只在连续地请求超过缓冲池页面大小的内存时才使用操作系统的分配器。

这个区别呼应了第 6.3.1 节中关于查询准备的讨论。前一类系统假设由 DBA 来完成复杂的调优工作，DBA 对系统内存各种参数进行仔细配置后，系统的工作负载将会服从于这些配置好的参数。在这些条件下，这样的系统应该总是能够像预期的那样很好地执行。后一类系统则假设 DBA 不能正确地设置这些参数，并努力以软件逻辑来代替 DBA 的手工调整。

系统保留了自适应改变其相关分配的权力，这为在可变的工作负载上获得更好的性能提供了可能性。如第 6.3.1 节所讨论的那样，从这个区别中不仅可以看出这些数据库厂商希望其产品如何被使用，也可以看出他们顾客的管理经验（和财务资源）。

7.3 磁盘管理子系统

DBMS 教材往往将磁盘视为同构对象。实际上，磁盘驱动器是复杂的异构硬件，在容量和带宽上大不相同。因此，每个 DBMS 都有一个磁盘管理子系统来处理这些问题，来管理表的分配和其他原始设备、逻辑卷或文件中的存储单元。

这个子系统的其中一个责任就是将表映射到设备和（或）文件上。表到文件一对一的映射听起来很自然，但在早期文件系统中会带来明显的问题。首先，传统的操作系统文件不能比磁盘大，而数据库表则可能需要跨越多个磁盘。其次，分配过多的操作系统文件被认为是不好的形式，因为，操作系统往往只允许少数的打开文件描述符，而且许多操作系统目录管理和备份的工具不能扩展到大量文件的情形。最后，许多早期的文件系统限制了文件大小上限为 2GB。如此小的表限制显然是无法接受的。许多 DBMS 厂商绕开了操作系统的文件系统而完全使用原始 IO，而其他厂商选择去解决这些限制。因此所有领先的商业 DBMS 可能会将一个表跨越多个文件，或在单个数据库文件中存储多个表。随着时间的推移，大多数操作系统的文件系统已改进解决了这些限制。但遗留的影响依然存在，而且现代 DBMS 往往仍将操作系统文件视为任意映射到数据库表的抽象存储单元。

更为复杂的是处理特定设备细节的代码，这些代码用来维护第 4 章所描述的时间和空间控制。基于复杂存储设备的产业今天依然存在，并且充满活力，它们把复杂存储设备伪装成磁盘驱动器，但实际上是一个大型硬件/软件系统，它的 API 还是遗留的磁盘驱动接口，如 SCSI。这些系统包括 RAID 系统和存储区域网络（SAN）设备，往往拥有超大容量和复杂的性能特征。管理员喜欢这些系统是因为它们易于安装，并且通常提供了易于管理的、位级（bit-level）的可靠性，可以支持快速失败恢复。这些特征为客户提供了重要的舒适感，远超 DBMS 恢复子系统的承诺。例如，大型 DBMS 配置一般使用存储区域网络。

不幸的是，这些系统使 DBMS 的实现变得更加复杂。例如，RAID 系统在发生错误之后和所有磁盘都正常运行时的性能表现大不相同。这潜在地使 DBMS 的 I/O 成本模型变得复杂。一些磁盘可以在开启写缓存的模式下操作，但在硬件故障时会导致数据损坏。先进的存储区域网络实现了大型的带后备电源的缓存，在一些情况下接近 TB 级，但这些系统本身具

有超过百万行的代码和相当的复杂性。复杂性带来了新的失败模式，这些问题可能是非常难以检测和正确诊断的。

RAID 系统在数据库任务上表现不佳，也使数据库的设计者失望。RAID 的构想是面向字节流的存储（Ia 后缀的 UNIX 文件），而不是用于数据库系统的面向页的存储。所以，与在多个物理设备上分区和复制的特定数据库解决方案相比较时，RAID 设备往往表现不佳。例如，Gamma 的*"chained declustering"*方法[43]，大体上与 RAID 方式一致，而且在 DBMS 环境中运行得更好。此外，大多数数据库提供了 DBA 命令来控制数据在多个设备上的分区，但 RAID 设备把多个设备隐藏在单一接口后面，破坏了这些命令。

当数据库在更加简单的方案如磁盘镜像（RAID1）下可以表现出非常好的性能时，许多用户会配置他们的 RAID 设备（RAID5），从而最小化空间开销。RAID5 有个特别不好的特征，那就是写入性能很差。这会对用户造成出人意料的瓶颈，而且，DBMS 厂商常常需要忙于向客户解释这个问题，或提供解决这些瓶颈的变通方法。无论如何，RAID 的使用（以及不当使用）实际上是商业 DBMS 必须考虑的。结果是，多数 DBMS 厂商花费大量的精力去调整他们的 DBMS，从而使其在领先的 RAID 设备上能够很好地工作。

在过去的十年，多数客户的部署是分配数据库存储到文件上，而不是直接分配到逻辑卷或原始设备上。但是，多数 DBMS 仍然支持原始设备访问，在运行大规模事务处理基准测试时，常常使用这种存储映射。而且，尽管有上面描述的一些缺点，多数企业 DBMS 存储如今还是使用 SAN。

7.4 备份服务

通过周期性更新来在网络上备份数据库常常是一种较为可行的方案。它被经常使用是为了额外的可靠性：在主服务器宕机的情况下，备份数据库可以作为稍微有些过时的“热备”（warm standby）。使热备处于不同的物理位置上，有利于在火灾或其他大灾难发生后能继续运行。备份也常常用于为大型的、地理上分散的企业提供了一个实际的分布式数据库功能。多数这样的企业将其数据库按大的地理区域（如国际或洲际）进行分区，并且在数据库的主副本上本地运行更新。查询也是本地执行的，但是，可以运行在混合的数据集合上，也就是一部分数据属于本地操作所获取的新数据，另一部分数据是从远程区域复制来的稍有些过时的数据。

如果忽略硬件技术（比如 EMC SRDF），那么，可以有三个用于备份的典型方案，但只

有第三个方案提供了高端环境所需的性能和可扩展性，当然这也是最难实现的。

1. **物理备份**：最简单的方案是在每一个备份周期物理地备份整个数据库。考虑到传送数据的带宽和在远程站点重新设置时的成本，这个方案不能扩展到大型数据库上。此外，保证数据库的事务的一致快照是困难的。因此，物理备份只作为低端客户端上的变通方案。多数数据库厂商以不鼓励通过任何软件方式使用此方案。
2. **基于触发器的备份**：在这个方案中，触发器置于数据库表中，这样，当表中发生任何插入、删除或更新操作时，一条表示变化的记录便会被放置在一个特殊的备份表中。这个备份表传送到远程站点以后，相应的变化操作就会在远程站点被“回放”（即重新执行一次）。这个方案解决了上述物理备份所提到的问题，但带来的性能损失对一些工作负载来说是不能接受的。
3. **基于日志的备份**：在一些可行的场景，基于日志的备份是应该选择的备份解决方案。在基于日志的备份中，一个日志嗅探器进程截取日志写入，并将其传送到远程系统。基于日志的备份的实现使用了两个技术：（1）读取日志并建立 SQL 语句，并在目标系统上重新执行，或（2）读取日志记录并将其传送到目标系统，系统处于持续的恢复模式，当日志记录到达时重新执行。这两个机制都有其价值，所以，Microsoft SQL Server、DB2 和 Oracle 都实现了两者。SQL Server 称第一个为日志传送，称第二个为数据库镜像。

这个方案克服了前两个备选方案的所有问题：它是低开销的，在运行的系统上只会引起最小的或不引人注意的性能开销；它提供了增量更新，因此，可以优雅地扩展数据库的大小和更新速率；它复用了 DBMS 内置的机制而不用太多额外的程序；最后，它通过日志的内置程序自然地提供了事务一致性。

大多数主流的数据库厂商都为其系统提供了基于日志的备份。提供能在多个数据库厂商之间工作的基于日志的备份，要困难得多，因为，使数据库厂商在远程终端重新执行逻辑需要知道数据库厂商的日志格式。

7.5 管理、监控和工具

每个 DBMS 都提供了一系列工具来管理其系统。这些工具很少用于基准测试，但常常能影响系统的易管理性。技术上的挑战和格外重要的特征是要让这些工具能在线运行，也就

是说，当用户查询和事务正在进行时去运行这些工具。这对于提供“24×7”服务而言是很重要的，随着电子商务在全球范围内的发展，这种服务在近几年变得更为常见了。传统上的凌晨时的“维护窗口”通常已不再存在。所以，多数数据库厂商在这几年投入大量精力来提供在线工具。在此我们总结一下这些工具提供的功能：

- **优化统计信息收集**：每个主流的 DBMS 都有一些方法来扫描表并构建关于排序或其他操作的优化器统计信息。一些统计信息（如柱状图），如果没有超大容量的内存，是很难在一次扫描中构建的。例如，可以看看 Flajolet 和 Martin 在计算一列中不同值的个数时的工作[17]。
- **物理重组和索引建立**：随着时间的推移，存取方法会因为插入和删除的模式所留下的未使用空间而变得效率低下。而且，用户可能会偶尔地请求让表在后台进行重组，例如，在不同的列上重新聚集（排序）数据，或者在多个磁盘上重新分区。在线重组文件和索引是困难的，因为，维护物理的一致性必须避免在任意长的时间里一直保持加锁状态。从这个意义上说，它和第 5.4 节中所描述的用于索引的日志和锁协议比较类似。这已经是一些研究论文[95]和专利的主要研究内容。
- **备份/导出**：所有的 DBMS 都支持物理地转储数据库到备份存储的能力。同样，因为这是个长期运行的进程，它不能简单地设置锁。因此，大多数系统执行一些“模糊（fuzzy）”转储，并配以日志逻辑来确保事务一致性。相似的方案可用于以交换格式导出数据库。
- **批量加载**：在许多场景中，大量的数据需要快速载入到数据库中。数据库厂商提供了针对高速数据导入而优化过的批量加载工具，而不是每次插入一条数据。通常这些工具通过存储管理器的自定义代码来实现的。例如，针对 B+-树的、特定的批量加载代码，会比反复调用插入树的代码要快得多。
- **监控、调优和资源管理器**：即使在托管环境中，查询会消耗超过所需的资源也是寻常的。所以，多数 DBMS 提供了工具来协助管理者辨别和预防这些问题。它通常提供了基于 SQL 的接口来通过虚拟表访问 DBMS 的性能计数器，可以显示由查询或锁、内存、临时存储等资源所导致的系统状态。在一些系统中，它也有可能去查询这些数据的历史日志。许多系统允许注册一些报警，从而在查询超过了指定性能限制时发出警报，性能限制包括运行时间、内存或锁的获取。有时候触发一个警报会造成查询中止。最后，如 IBM 的预测资源管理器工具，会尽量预防资源密集型的查询被运行。

附录 1:译者介绍

林子雨(1978－),男,博士,厦门大学计算机科学系助理教授,主要研究领域为数据库,数据仓库,数据挖掘.
主讲课程：《大数据技术基础》
办公地点：厦门大学海韵园科研 2 号楼
E-mail: ziyulin@xmu.edu.cn
个人网页：http://www.cs.xmu.edu.cn/linziyu