Db2 数据库 设计

一、简介

本文的目的是为 IBM 业务伙伴提供关于 DB2 Universal Database(UDB)for z/OS(后面将简称为 DB2)环境中 DB2 数据库性能的重要信息。本文试图从多处收集材料，并尽可能有效地将它们表述出来。本文无意包含很全面的范围，也不会包含很深的细节。

我曾打算讨论对 DB2 数据库的性能影响最大的一些因素。但是，并不是所有可能的情形都可以预测到，也不是所有潜在的考虑都能顾及到，更不用说在期望的范围内对它们进行描述了。我希望本文可以为不同环境下的 DB2 用户提供一个通用的指南，以帮助他们取得最佳的 DB2 数据库性能。本文的目的是成为一个良好的起点，用以处理任何给定安装环境下的数据库性能问题。

本文的范围是数据库设计性能。DB2 性能远不止这一部分，它肯定要受到数据库设计以外的很多因素的影响。例如，程序的编码逻辑和其中使用的实际的 SQL 语句，可以列为应用程序设计一类。DB2 系统性能可以包括诸如安装选项、缓冲池大小设置、DB2 相关地址空间的调度优先级等等之类的因素。

本文的焦点是 DB2 数据库的设计。不过，有时候这些性能因素类别之间的界线可能会有些模糊。例如，在某种安装环境下进行配置时，缓冲池大小的设置和数量的选择通常被认为是一项系统性能因素。但是，倘若是将特定的表空间和索引指派给那些缓冲池，那么这些因素又可以看作是数据库设计一类的因素了。

在这里，我假设读者对 z/OS 环境中的 DB2 有一个基本的理解。本文的头几页将讨论性能管理的一些基本概念和准则，以便进行“级别设置” 。我的建议有点综合的性质，因而不会总是详细地给出技术性的描述和语法。读者如果想了解关于这些内容的更详细的信息，那么应该去阅读关于用户本地所安装的 DB2 版本的最近的 IBM 文档。

本文的通用设计点是 DB2 for z/OS V7。虽然 DB2 for z/OS V8 已经被宣布，并且也普遍可用(generally available ，GA)，但是大部分 IBM 客户极有可能需要几个月的时间才能实现用于他们的生产系统的 DB2 V8 NFM(New Function Mode)。而且，这里还要考虑另外一个因素。虽然 DB2 的每个新版本在变得普遍可用之前，都已经在 IBM 及其客户环境下经过了广泛的测试，但是相对于一个还没有推广的、没有普遍使用的版本而言，客户们往往对于基于早先版本的 DB2 的一般建议和窍门更有信心(长时间积累的经验验证了这一结论)。我将提到 DB2 V8 的一些新特性，从性能的角度来看，这些新性能可能会影响数据库设计。

二、性能准则和方法学

1. 何时考虑性能

考虑应用程序和数据库的性能特性的时机是在那些应用程序和数据库的初期设计阶段，也就是开发过程的开始阶段。对 DB2 应用程序和数据库所需的资源进行合理的估计，这有助于用户在开发过程的早期便对设计和实现作出恰当的决定。如果等到后期才来考虑访问数据库的应用程序的性能，那么为了取得适当的响应时间和生成批处理窗口而进行一些必需的修改时，就会更加困难，而且更加消耗时间。

2. 应该关注些什么

当从性能的角度进行设计时，将大部分的精力集中在重要 DB2 数据和程序上，这种做法比较明智。在确定是什么应用程序或事务构成这一重要的工作负载时，以下特征中的一条或几条将会适用：

1) 它们代表了总体业务工作负载的很大的百分比。

2) 它们有着关键响应时间需求。

3) 它们包括复杂的逻辑和/或数据访问需求。

4) 它们访问大量的数据。

5) 它们消耗大量的资源。

6) 与那些属于公司内部的应用程序相比，它们是直接与客户打交道的(通过 Web、ATM 等)。

3. 数据库设计

数据库的设计有两个阶段：

1) 逻辑数据库设计

2) 物理数据库设计

数据库的逻辑模型仅仅是对用户的所有数据需求的一种表示，它将这些需求变成一种范式。这种模型通常就是数据建模会议的输出或最终结果。该模型实际上很少被原原本本地实现。其实，该模型只是在考虑如何实际地构造数据和将数据存储在 DBMS 之前，对数据的一种理想化的看法。

在对数据库对象的架构进行了考虑之后，逻辑模型就被转化为物理模型。在设计的这个阶段，就需要较为详细地考虑数据访问需求和性能因素。在产生物理设计的这个过程当中，有两大要素，即表设计和索引设计。下面将较为详细地讨论这两个话题。

4. DB2 性能管理的方法

为了确保 DB2 应用程序具备合格的性能，未雨绸缪胜于亡羊补牢。在设计 DB2 数据库的早期阶段就将性能因素考虑进来，这一点很重要。然后，在项目尽可能早的时候，建立一套符合 Service Level Agreement(SLA) 的性能“基准线”测量方法，这样，便可以在展示的时候和应用程序被修改的时候，跟踪性能特性和趋势。同时还应该持续地监控 DB2 系统和应用程序，从而在大的问题完全发作之前进行预测。

通常，很多客户只有到了应用程序开发项目的最后阶段才开始担心性能。但是通常没有什么好的理由需要等到那时才去考虑性能。更好的做法是，在指定了用户界面和处理逻辑之后，立即考虑数据库设计的性能特性。例如，在创建最佳索引时，应该将重要 DB2 工作(请参阅前面的讨论)中 SQL 语句的谓词作为主要指南。

即使您可以开发一个有效的初期设计，随着数据量的增加，或者在系统资源紧缺的时候，也仍有必要对应用程序和/或数据库作出修改。如果一个应用程序执行时的性能不合格，较为可取的做法通常是添加更多的列到现有的索引中，或者为一个表添加新的索引，这种做法是首选。而更改表的设计，或修改用户需求，抑或修改反规范化(de-normalizing)表，都不是很有吸引力的选择。

三、理解 DB2 性能

1. Rules-of-thumb

Rules of thumb(经验法则，也称 ROT)在规划、监控和优化 DB2 性能的时候很有用。ROT 通常是基于以前的经验(比如在一段时间内观察到的平均值)或者更复杂公式的简化。

记住这样一个事实很重要，即 ROT 只对于粗略的估计有用，而对于详细的分析用处不大。如果只是在某一类的文档中看到了一些 ROT，便欣然接受并作为精确的事实来引用，那么就会有危险。在最好的情况下，这些 ROT 是一种估计，而在最坏的情况下，这些 ROT 对于特定的 DB2 环境可能不成立。

您应该为自己的环境特别开发这些ROT(或者对它们进行调节，以适应自己的环境的特性)。应确保ROT与实际经验相关，而不是盲目地接受，这样才能对它们更有信心。一开始的时候，使用那些在您特定环境以外被使用过或者被开发出来的 ROT，这种做法可能有用。但是，当对您自己 DB2 系统中的适当数据进行收集、分析和编制文档之后，应该对这些 ROT 加以验证和修改。IBM Redbooks 是关于 ROT 的一种很好的参考资料，这些 ROT 常常作为建议被包括在性能监控工具中。

另一方面的考虑是，ROT 可能随着时间的推移而演变。硬件技术的发展，软件编程技术的提高，系统架构的变化，诸如此类的变化都可能使得 ROT 的可靠性降低，甚至变得无效。而使 ROT 随着时间变化的最大因素也许正是 DB2 本身新版本的发行。

2. DB2 工作负载

磁盘 I/O 常常是影响响应时间的最大因素，但是通过查看 GETPAGE(GP)请求，更容易理解底层的性能问题。当监控 DB2 活动和分析报告时，GETPAGE 的数量也许是 DB2 总体工作负载的最好的指示器。

某个安装环境下的很多 DB2 工作都可以无异议地归为以下几类：

1) 事务：事务是在事务管理器(例如 CICS 和 IMS/TM)控制下运行的程序。其中的 SQL 通常比较简单，但是事务量比较重。事务必须为用户提供极好的响应时间，这样应用程序才不致于要长时间地等待它们所需的资源。通常，第一个调用事务的用户将承受读取索引和数据页的成本。随后的用户则常常可以发现有些资源已经在缓冲池中。

2) 查询：查询也是程序，常常在需要决策支持时执行它。其中的 SQL 可能非常复杂，但是工作量常常远不及事务。查询的用户常常要等上数分钟甚至数小时，这取决于为了产生用户所请求的结果集，需要对多少数据进行搜索。查询常常要引起对整个表的扫描，而对结果排序是这种类型的工作负载的另一种常见特征。

3) 批处理和实用程序： 批处理和实用程序通常处理大量的数据，并且常常以一种连续的方式处理数据。这些程序在给定的窗口中完成它们的处理，这一点很重要。批处理和实用程序往往是各种系统资源的消费大户，一旦它们挤在一起，常常会使工作负载逐步上升。

3. 规范化

规范化是分析应用程序所需的数据实体，然后将这些数据实体转化成一组设计良好的结构的一个格式化的过程。逻辑数据模型的一般设计目标是正确性、一致性、非冗余和简单性。而且，关系理论的信条也要求数据库要经过规范化。

有一些按照连续编号排列的规则(也叫 范式(form))可以用来很详细地定义规范化数据。大多数专家都会建议设计者尽量遵从前三条规则，由此得到的数据就可以说是符合第三范式。

而将一个表反规范化(de-normalize)的意思是，违反该表之前遵从的一种或多种范式，从而修改规范化的设计。这种反标准化的过程常常是由于性能的原因而进行的。在大多数以关系数据库为主题的书籍当中，都可以找到关于规范化的更详细的信息。

4. DB2 表空间类型

在一个定义好的 DB2 数据库中，实际的表必须在称作表空间(table space)的 DB2 对象中创建。用户可以在 DB2 中定义 4 种不同的表空间：

1) 简单表空间：简单表空间可以包含一个以上的 DB2 表。这种表空间由页构成，每个页可以包含该表空间中定义的任何表中的行。

2) 分段表空间：分段表空间可以包含一个以上的 DB2 表。这种表空间由页组构成，页组被称作段(segment)。每个段只能包含该表空间中定义的一个表中的行。

3) 分区表空间：分区表空间只能包含一个表。根据分区(partitioning)索引的键范围，这种表空间被分成数个分区。每个分区都被看作一个独立的实体，允许 SQL 和 DB2 实用程序对其进行并发处理。

4) LOB 表空间：LOB 表空间只用于 LOB(大型对象)数据。LOB 包括三种数据类型：BLOB(二进制大型对象)、CLOB(字符大型对象)和 DBCLOB(双字节字符大型对象)。

四、表空间与表设计方面的考虑

1. 记录大小和页宽

固定长度的记录要优于可变长度的记录，因为 DB2 代码专门为处理固定长度的记录进行优化。如果记录是固定长度的，那么就无需将其从存储它的初始页面转移到其他地方。而对于可变长度的记录，其长度可能会变得不再适合初始页，因此必须将其转移到其他页。之后，每当需要访问该记录时，就必须发生额外的页引用。DB2 UDB V8 中的一种新特性允许在需要的时候修改(ALTER)某一列的宽度，这样一来，即使您不能确定将来数据长度的增长情况，也不再需要创建可变长度的记录。

一个页中所能存放的记录的数目也是值得考虑的一个方面。DB2 为页宽提供了很多选项(4KB、8KB、16KB 和 32KB)。一开始的时候，可以选择默认选项(4KB)，如果行的长度很小，或者对数据的访问基本上是随机的，则更应该选择这一选项。不过，在有些情况下，则需要考虑使用更大的页宽。如果一个表中各行的长度要大于 4KB，那么就需要使用更大的页宽，因为 DB2 不支持跨页(spanned)记录。

还有一些情况下，对于一条固定长度的记录，其总长度可能刚好比 4KB 的一半大一点点，这时一个页只能容纳一条记录。对于刚好比页宽的 1/3、1/4 大一点点的记录，情形也是类似的。这种设计不仅浪费 DASD 空间，而且，对于很多 DB2 操作，还需要消耗更多的资源。因此，对于这一类的记录，应该考虑使用更大的页宽，这样浪费的空间相对要少一些。

其他可能的页宽是 8KB、16KB 和 32KB。页宽不是在 CREATE TABLE 语句中直接指定的。相反，表的页宽是由相应的缓冲池的页宽来确定的，这个缓冲池也就是为包含该表的表空间所指定的缓冲池。要了解更多细节，请参考 DB2 SQL Reference 手册中的 CREATE TABLESPACE 语句。

2. 反规范化方面的考虑

逻辑数据模型是数据的理想蓝图。物理数据模型才是对数据的现实的实现。规范化只关注数据的意义，而没有考虑对于访问数据的应用程序的性能需求。完全规范化的数据库设计在性能方面要受到质疑。

这种性能问题的最常见的例子是连接(join)操作。通常，规范化过程最终将相关的信息放入不同的表中。于是应用程序需要从多个这样的表中访问数据。关系数据库为 SQL 语句提供了从一个以上的表中访问信息的能力，这是通过 连接多个表来完成的。连接操作可能要消耗大量的资源和时间，这取决于表的数量以及这些表各自的长度。

像 I/T 中的很多事情一样， 这里也可以考虑一种权衡的方法。对于具有经常被请求的列的多个表，一种是复制其中的数据，一种是执行表连接，两者谁的成本更高呢?在逻辑数据库设计过程中，您可以毫无保留地规范化数据模型，然后再加入一定程度的反规范化，作为潜在的性能调优的一种选择。如果您确实打算反规范化，那么一定要为此制作完整文档：较详细地描述您所采取的反规范化步骤背后的原因。

3. 设计大型的表

访问非常大的 DB2 表时，相应地要消耗很多的资源：CPU、内存和 I/O。在设计大型表的时候，为了提高性能，用户可以做的最重要的两件事是：

1) 实现分区。

2) 创建有用的索引。

下面将更详细地讨论这两个话题。

4. 使用分段的或分区的表空间

如果数据包括 LOB，那么用户就必须创建 LOB 表空间。对于非 LOB 数据，一般需要在分区表空间和分段表空间之间进行选择，这很大程度上取决于要存储的数据量，在一定长度上也取决于相关应用程序所需的数据访问类型。简单表空间已经很少被推荐了。

下面列出了分段表空间相对于简单表空间的一些性能优势：

1) 对于包含多个表的表空间，当 DB2 取得用于某一个表的锁时，这个锁不会影响对其他表的段的访问。

2) 当 DB2 扫描一个表时，只是访问与那个表相关的段。而且，空段中的页不会被提取。

3) 如果一个表被删除，在执行 COMMIT 之际，它的段就立即可以为其他表所用，而不需要执行 REORG 实用程序。

4) 如果一个表中的所有行都被删除(即 mass delete)，则在执行 COMMIT 之际，该表所有的段就立即可以为其他表所用，而不需要执行 REORG 实用程序。

5) mass delete 执行起来要高效得多，并且要写的日志信息也更少一些。

6) COPY 实用程序不会复制那些由 mass delete 操作或删除(DROP)一个表所造成的空页。

当表达到一定大小时，通过实现分区表空间可以提高易管理性和性能。如果预见到这样的增长，那么明智的做法是，在设计和创建表空间时将其定义为分区的。下面列出了分区表空间可以提供的一些潜在的优势：

1) 并行性：您可以使用 DB2 UDB 目前所使用的三种并行方式。查询并行(多条 I/O 路径)是在 DB2 V3 中引入的。Sysplex 查询并行(一个 DB2 数据共享组中的多用户和多任务)是在 DB2 UDB V5 中引入的。到现在，DB2 已得到极大的发展，并大大地增强了那些处理分区表空间的 DB2 应用程序的并行处理能力。通过增加一定的 CPU 时间，可以大大减少这些查询所需的时间。

2) 对部分数据进行操作：分区表空间允许 DB2 实用程序一次处理一个分区的数据，这样其他任务或应用程序就可以并发地对其他分区进行访问。按照类似的方式，您可以将 mass UPDATE、DELETE 或 INSERT 操作拆成多个不同的任务。除了增加可用性以外，这种技术还可以为减少完成这种 DB2 工作所需的时间提供潜力。

3) 对频繁访问的数据有更快的访问速度：如果分区索引可以将访问更频繁的行与表中其他的行分开来，那么就可以将这些数据放入到它们专用的分区中，并使用更高速的 DASD 设备。

通常，表越大，就越有理由将其创建为分区的表。但有时候为较小的表创建分区表空间也很有利。当将 查找(lookup)表与其他较大的分区的表相连接时，通过将查找表也进行分区，可以最大化并行度。

如果在连接谓词中使用分区键(partitioning key)，最后还有一点考虑需要顾及。需要按分区键进行连接的表应该有相同数量的分区，并且应该在相同的值上断开。

5. 数据压缩

DB2 提供了压缩一个表空间或分区中的数据的能力。这是通过在 CREATE TABLESPACE 语句中指定 COMPRESS YES 选项，然后对表空间执行 LOAD 或 REORG 实用程序来实现的。通过用较短的字符串替换经常出现的长字符串，可以压缩数据。这时会建立一个字典，其中包含了映射原始的长字符串与它们的替换值的信息。

在数据被存储之前压缩数据，以及在从外部存储设备读出数据时将数据解压，这都需要使用一定的 CPU 资源。但是，数据压缩也可以为性能带来好处，因为可以在更少的空间(包括 DASD 和缓冲池中的空间)中存储更多的数据，与未压缩的数据相比，这样可以减少同步读，减少 I/O 等。

在决定是否压缩一个表空间或分区时，要考虑下面一些事情：

1) 行的长度：行的长度越大(尤其是它接近页宽时)，压缩的效率就越低。在 DB2 中，行不能跨页，您可能无法实现足够的压缩来使一页可以容纳多行。

2) 表的长度：对于更大的表空间，压缩更为有效。对于非常小的表，压缩字典的大小(8KB 到 64KB)有可能会抵消掉通过压缩所节省出来的空间。

3) 数据中的模式：对于特定的表空间或分区，数据中重复出现的模式的出现频率将决定压缩的效果。有大量重复字符串的数据有巨大的压缩潜力。

4) 对压缩的估计：DB2 提供了一个独立的实用程序 DSN1COMP，通过执行该实用程序可以判断压缩数据的效果。要了解关于运行该实用程序的更多信息，请参考 DB2 Utilities Guide and Reference 手册。

5) 处理成本： 压缩和解压数据时，要消耗一定的 CPU 资源。与使用 DB2 软件模拟程序相比，使用 IBM 的同步数据压缩硬件可以大大减少所消耗的 CPU 资源(当 DB2 启动时，它将判断硬件压缩特性是否可用)。

6) 更好的字典：当使用 LOAD 实用程序来建立压缩字典时，DB2 使用所装载的前 n 行(其中 n 取决于数据的压缩程度)来决定字典的内容。REORG 使用一种抽样技术来建立字典。它不仅使用所装载的前 n 行，而且还会对该实用程序执行期间剩下的 UNLOAD 阶段中的行进行抽样。因此，REORG 常常可以产生更能代表整个表空间或分区中的数据的字典。

如果您的环境可以从压缩中得到好处，通常我们建议压缩那些 DB2 表空间和分区，因为由于更少空间容纳更多数据所带来的性能优势几乎总是大于压缩和解压数据时所需的 CPU 资源消耗。

6. 装载大型的表

当处理大量的数据时，一开始将数据装载到表中时可能会遇到性能问题。为了在装载过程中实现并行性，可以手动地创建多个 LOAD 任务，每个分区对应一个任务，或者，也可以在单个 LOAD 实用程序中装载多个分区。每个分区都展开在 I/O 子系统上，从而更易于达到最佳的并行度。

为了获得最佳性能，在 LOAD 语句中指定 SORTKEYS 参数也很重要。这将指示 DB2 将索引键传递给内存中的排序程序，而不是再次将这些键写到 DASD 上的排序工作文件中，然后从中读取这些键。SORTKEYS 还允许装载和排序之间的重叠，因为排序是作为一个单独的任务运行的。

下面给出了关于装载大型表的其他建议：

1) 一次只 LOAD 一个表。

2) 如果可能，为那些预计将历时最长的任务提供较高的优先级。

3) 将工作分布在 sysplex 上。

4) 将辅助索引拆分成数块，以实现并行性(请参阅下面题为 PIECESIZE 的讨论)。

5) 在开始装载数据时，指定 LOG NO(用以防止日志记录，以及节省日志记录所消耗的大量资源)，然后在成功装载数据之后运行一个映像拷贝。

7. 空余空间方面的考虑

分配空余空间的主要目的是使数据行的物理顺序与群集索引一致，以减少频繁重组数据的需要。此外，对行的更好的群集会使读访问的速度更快，行插入的速度也更快。然而，过多地分配空余空间可能会产生浪费的 DASD 空间，导致每次 I/O 只能传输更少的数据，缓冲池的利用效率更低，并且要扫描更多的页。

表空间和索引中空余空间的分配是由 CREATE 或 ALTER TABLESPACE 以及 CREATE 或 ALTER INDEX 语句的 PCTFREE 和 FREEPAGE 选项确定的。

PCTFREE 告诉 DB2 在装载或重组数据时，表空间或索引中的每个页要留出多少百分比的空余空间。如果没有足够的空余空间来按照恰当的顺序(也就是按群集顺序)编写行或索引，那么 DB2 就必须将多出的数据放到另一个页上。当越来越多的记录被乱序存放时，性能就会出现问题。

FREEPAGE 告诉 DB2 在装载或重组数据时，应该过多久就分配一整页的空余空间。例如，如果指定了 FREEPAGE 5，那么 DB2 将在用数据填充了 5 个页之后分配一页的空余空间。如果表行大于页宽的一半，则应该使用 FREEPAGE，因为在那样的环境下不能在一页上再 INSERT 一行。

是否定义带空余空间的表空间，以及分配多少的空余空间，这很大程度上取决于表空间中表的 INSERT 特征(其次是 DELETE 活动)。换句话说，这取决于将行添加到表中的频率，以及将行添加到表的什么地方。下面有 4 种类别：

1) 只读表：如果对于一个表没有任务更新活动，那么可以将其定义为没有空余空间。而且，也没有任何必要去运行 REORG 实用程序。

2) 随机插入：对于已经有很多行、并且 INSERT 活动的级别很低，那么一开始可以使用默认的 PCTFREE(对于表空间，该值是 5，对于索引，该值是 10)。然后使用 RUNSTATS 监控数据失序的程度，再考虑您期望运行 REORG 的频率，对 PCTFREE 进行必要的上下调整。对于 INSERT 活动级别很高的表，可能需要使用大于默认值的 PCTFREE。对于开始为空或者只包含很少行的表(例如在新数据库的部署期间)，可能需要指定一个较高的 PCTFREE，并不时地运行一下 REORG，直到这个表被填充好为止。

3) 在表的末尾插入：如果一个表中的行的长度不会增长，那么就无需分配空余空间，因为这些行是在表的末尾插入的。由于这些行是按照物理的群集顺序来写的，因此不需要运行 REORG。但是，如果一个表包含可更新的 VARCHAR 列，或者该表被压缩，那么有可能行的长度会增长，从而导致某一行被转移到其他页上。您可以通过对表空间执行 RUNSTATS，然后检查 DB2 编目表 SYSIBM.SYSTABLEPART 的 NEARINDREF 和 FARINDREF 这两列来判断上述情况。如果表变得缺乏组织，那么为表空间指定一个 PCTFREE，并继续用 RUNSTATS 监控位置不当(mislocated)的行。根据当前的数据以及您所观察到的趋势，对 REORG 的频率和 PCTFREE 的数字进行相应的调整。通过在 REORG TABLESPACE 中指定 INDREFLIMIT 和 REPORTONLY 选项，可以监控被更新的 DB2 表中位置不当的行的数目，以及隔多远会出现这样的行。

4) 在热点(hot spot)中插入：在这种情况下，表上的插入活动很频繁，而且集中在某一个位置(或几个位置)，而不是在表的末尾进行插入。这一类情况可能是最难于处理的。可以尝试增加 PCTFREE 的值。如果插入活动停留在主段(home segment)中，并且插入的行不是很长，那么可以将数行存储在相同的页中。在此情况下，另一种可以考虑的方案是使用 FREEPAGE。这时有必要严密监控表变得无组织的速度，这样就可以在性能急剧下降之前运行 REORG。

五、索引设计方面的考虑

索引也是一种 DB2 对象(一个单独的 VSAM 数据集)，它由一组排好序的键组成，这些键是从相应表中的一个列或多个列抽取出来的。很多 DB2 专家声称，只有为表空间建立恰当的索引，才是使得访问该表空间中 DB2 数据的应用程序的性能达到最佳、最有效的效果。数年前，在 I/T 中 DASD 的成本和空间是更重要的考虑因素。随着技术的发展，通过增加更多的索引(或添加列到已有的索引中)来减少 I/O，以及由此消耗的额外磁盘空间，这几年两者之间的权衡已经变得越来越有吸引力。索引所带来的主要性能好处是：

1) 提供指向表中被请求的数据行的直接指针。

2) 如果结果集要求的顺序与索引一致，则可以消除排序。

3) 如果被请求的列都包含在索引项中，则可以避免不得不读数据行的情况。

1. 分区索引

在 DB2 UDB V7 中创建分区的表空间时，DB2 根据 CREATE INDEX 语句的 PART 子句将数据划分到几个分区上。那样的索引就成为所谓的分区索引，而这种分区的方法就被称为 索引控制的分区(index-controlled partitioning)。对于分区索引，建议选择不大可能改变的键列。如果对那些列进行更新，则可能导致一行从一个分区转移到另一个分区，从而降低了性能。

DB2 V8 一个重要的特性是表控制的分区(table-controlled partitioning)。这时，当创建分区的表时，分区的边界由 CREATE TABLE 语句决定，而不是由 CREATE INDEX 语句决定。对于索引控制的分区方法，分区的表、分区索引和群集这几个概念之间有点纠缠不清。而在表控制的分区方法中，这三个概念是各自独立的。这种增加的灵活性使您可以考虑更多潜在的设计方案，因而也增加了提高 DB2 数据库及其应用程序性能的机会。

2. 何时建立索引

CREATE INDEX 语句使用户可以立即建立索引，或者将索引的建立推迟到方便的时候。如果立即建立索引，则需要扫描表空间，这样要花费比较多的时间。通过指定 DEFER，则可以推迟索引的创建。

只要有可能，应该在初次装载一个表之前创建其所有索引，因为 LOAD 实用程序建立索引的效率比 CREATE INDEX 过程要高。如果需要在一个已有的(并且被填充的)表上创建一个索引，那么可以使用 DEFER 子句。然后可以在晚些时候使用 REBUILD INDEX 实用程序，这个实用程序与 LOAD 实用程序一样，是更为有效的填充索引的方式。

3. PIECESIZE

DB2 UDB V5 中引入了一个新特性，这种特性使您可以将一个非分区索引(non-partitioning index，NPI)拆成数块，然后控制将组成索引空间的多个数据集的大小。通过使用这些小块，可以使 NPI 的索引页散步到多个数据集中。

通过在 CREATE 或 ALTER INDEX 语句中指定关键字 PIECESIZE，可以确定各块的大小。PIECESIZE 的值必须是 2 的幂，其大小可以介于 256KB 到 64GB 之间。对于常规表空间，PIECESIZE 的默认值是 2GB，对于 LARGE 表空间，默认值是 4GB。如果 NPI 极有可能显著增长，那么应选择一个更大的值。在为主空间和辅助空间(CREATE INDEX 语句的 PRIQTY 和 SECQTY 选项)的分配确定值时，也应该留意 PIECESIZE 的值。

通过使用这个选项，可以促进并行性，从而提高 NPI 的扫描性能。另一个好处是可以减少在读或更新的处理过程中对 I/O 的争用。通过指定一个较小的 PIECESIZE，可以创建更多的块，从而对块的放置有更多的控制。将这些块放在不同的 I/O 路径中，可以减少访问 NPI 所需的 SQL 操作的争用。

4. 理想的索引

通过检查应用程序中的 SQL 语句，可以建立一种想象起来很好的索引。

1) 首先，在索引中包括 WHERE 子句中的所有列，这样，就可以使用索引形成的屏蔽来拒绝结果集中不合格的行。将这些列放在索引的开始部分。这样一来，当对 SQL 语句进行 EXPLAIN 时，就可以产生最大的 MATCHCOLS 值。

2) 接下来，确保索引中这些列有适当的顺序(按照 ORDER BY 子句)，这样可以避免排序。在进行 EXPLAIN 时，通过检查 PLAN_TABLE 中所有不同的 SORT*列，便可以确认这一点。

3) 最后，如果可能的话，将 SELECT 中所有的列包括到索引当中，这样就不需要访问表中的行。这样的索引项可以提供所有被请求的数据。这在 EXPLAIN 中就表现为 INDEXONLY = Y。

在很多情况下，实现这一理想的代价太高，也不切实际，甚至是不可能的。对于一个索引中可以包括的列数，以及整个索引项的长度，都有架构上的限制(虽然这些限制已考虑到相当大的索引项长度和灵活性)。而且，也要考虑索引维护的成本。虽然建立理想化的索引可以显著提高查询性能，但是每当对 DB2 数据库执行 SQL 写操作(INSERT、UPDATE 或 DELETE)时，上述理想化的索引都会有负面的影响。因此，您常常可以选择实现只包括在 WHERE 和 ORDER BY 子句中引用到的列的索引。

六、并行处理方面的考虑

这些年，DB2 通过实现各种并行处理的方法，已经大大提高了访问数据的性能。为了提高数据密集型只读查询的性能，DB2 V3 引入了查询 I/O 并行。在这种并行中，DB2 充分利用分区表空间促成的可用 I/O 带宽。通过这种方法，DB2 可以为单个 I/O 请求启动多个并发的 I/O 请求，并在多个数据分区上执行并行 I/O 处理。这通常可以显著减少 I/O bound 查询所需的时间，而代价只是稍微增加的 CPU 时间。

DB2 V4 引入了另一种并行技术，这种技术称作查询 CP 并行。这种方法将并行处理扩展到过程密集型(process-intensive)查询中来。通过这种方法，一条查询可以使 DB2 生成多个任务，这些任务被并行地执行，以访问数据。分区表空间最能体现这种并行所带来的性能提高。

DB2 UDB V5 引入了 sysplex 查询并行，进一步扩展了并行处理。CP 并行可以在 DB2 子系统中为一条查询使用多个任务，而 sysplex 查询并行这种方法使一个 DB2 数据共享组中的所有成员可以一起处理一个查询。对于那些主要是只读形式的 I/O 密集型和处理器密集型查询，都可以从这种并行中得到好处。

1. 支持并行访问

DB2 环境中对并行的支持有一个度的问题。首先，在 DB2 子系统级，并行访问是在安装面板 DSNTIP4 上控制的。DSNTIP4 上的 MAX DEGREE 选项决定了最大并行度(并行任务的最大数量)。默认值是 0，这意味着对于 DB2 可能调用的并行度没有上限。我建议您先估计 z/OS 环境中的虚拟存储能力和局限性，这样 DB2 就不至于创建多于虚拟存储所能处理的并行任务。

您可以通过 BIND PLAN 和 BIND PACKAGE 命令的 DEGREE 选项来控制 DB2 是否利用并行处理。若指定 DEGREE(1)，表示禁止并行处理，若指定 DEGREE(ANY)，则表示支持并行处理。为获得更大的灵活性，动态 SQL 允许通过 SET CURRENT DEGREE 语句在一个计划或包中更改这个选项，该语句可以控制专用寄存器中的值。

当一个计划或包与 DEGREE(ANY)捆绑在一起，或者 CURRENT DEGREE 寄存器被设为 ANY 时，DB2 优化器将考虑对于最有效的顺序计划，并行是否可行。如果并行不可行，那么就选择次好的顺序计划。

2. 限定分区扫描

限定分区扫描允许 DB2 将数据扫描限制在一个分区表空间中。根据 SQL 谓词中的值，DB2 可以判断可能包含 SQL 语句所请求的表行的最低分区和最高分区，然后将数据扫描限制在这一范围内的分区中。为了使用这种技术，SQL 必须提供分区索引的第一个键列上的一个谓词。

3. 并行方面的建议

为了进一步促进并行处理所能带来的性能提高，下面列出了一些需要考虑的事情：

1)尽可能均匀地对表空间分区，因为数据不整齐会对并行度产生影响。一般来说，DB2 根据最大物理分区的大小将表空间分成逻辑上的几块。

2) 为 DB2 应用程序的处理分配尽可能多的中央处理器(central processor，CP)，以及尽可能多的 I/O 设备和路径。

3) 对于 I/O 密集型查询，应确保分区的数量与可以访问该表空间的 I/O 路径的数量一致。

4) 对于处理器密集型查询，应确保分区的数量等于将被分配用来在数据共享组上处理查询的 CP 的数量。

5) 将用于表空间和索引的分区放在单独的 DASD 卷中，并且，如果可能的话，要隔开控制单元，以减少 I/O 争用。

6) 按时执行 RUNSTATS 实用程序，以获得分区级的统计信息。

7) 监控虚拟缓冲池的阈值和使用情况，确保提供了足够的缓冲池空间来最大化并行度。

七、缓冲池方面的考虑

1. 缓冲池的重要性

很多专家将数据库缓冲池看作 DB2 环境中影响性能的最关键的资源。很多 DB2 的架构和设计，其基本思想都是尽可能地避免物理 I/O。

DB2 缓冲池由数个插槽(slot)的连续的内存组成。数据和索引页被从 DASD 中读出之后，便进入这些插槽，并留在其中，直到 DB2 缓冲区管理器确定那些插槽要用于其他数据。应用程序所请求的数据出现在

内存中(而不是外面的 DASD 上)的概率越大，总体性能就越好。实际上，这里的数据被重复使用，因而减少了应用程序对 I/O 的需要。

是否释放一个缓冲池槽，这是根据最近被使用(LRU)原则来决定的。DB2 维护两个 LRU 列表，一个用于被随机访问的页，另一个用于被顺序访问的页。这样可以防止大规模的表扫描完全支配缓冲池，并恶劣地影响随机操作。通过使用不同的阈值，DB2 提供了改善缓冲池性能的灵活性。在 DB2 SQL Reference 手册的第 2.7.4 节中对这些阈值进行了较为详细的讨论。

2. 为缓冲池设置适当的大小

缓冲池大小的指定要取决于可用存储(包括中央存储和扩展存储)的容量。我建议首先分析缓冲池的分配，然后逐渐增加缓冲池的大小，直到通过增加分配的空间已无法增加更多的吞吐量，或者直到 MVS 换页率已难于接受为止。为实现这一点，要使 DASD I/O 的数量持续下降，并不断增加 VPSIZE，直到换页的成本超出了通过减少 I/O 所带来的好处为止。

早些时候，GETPAGES 的数量被认为可能是对 DB2 正在运行的工作量的最好度量。缓冲池的目的是减少 I/O(异步读通常表明需要进行预取，从性能角度来看，这样做通常是值得的。另一方面，同步读常常需要对 DASD 进行随机 I/O，因为被请求的页不在缓冲池中)。会计报表显示对应于每个缓冲池的 GETPAGES 和同步读的数量。一个被普遍接受的 ROT 声称，如果 GETPAGES 对同步读的比率小于 10:1，那么应该估计对更大缓冲池的需要。

3. 多缓冲池配置

如果操作系统允许为 DB2 缓冲池分配相当大的内存，那么使用多缓冲池的配置很可能可以提高特定应用程序或数据库的性能。然而，需要清楚的是，若有了多个缓冲池，那么对这些缓冲池使用效率的监控就变得更加重要。

下面给出了关于分配多个缓冲池的一般建议：

1) 将表空间与和它们相关的索引分放到不同的缓冲池中，以减少索引 I/O。

2) 将有不同数据访问模式的数据统一放到不同的缓冲池中。批处理和查询应用程序通常要进行大量的顺序处理，而用于 OLTP 的数据访问往往更具有随机性。这为利用各种阈值处理缓冲池中某些类型的数据访问提供了一种方法。

3) 为独立的应用程序提供一个单独的缓冲池。这就为紧密监控应用程序的性能问题或测试新的应用程序提供了一种方法。

4) 如果排序的性能在您的环境中很重要，那么需要为工作文件创建一个单独的缓冲池。

5) 对于相对较小但更新频繁的表，通过一个足够大的单独的缓冲池，也许可以同时减少读和写的 I/O。

6) 为只读表(小的引用表)提供单独的缓冲池可以提高性能。

八、结束语

考虑周详的数据库设计可以提供巨大的性能收益，但是这必须在应用程序开发过程的早期便开始着手。从早期的 DB2 开始，明智的开发人员就已经使用了前面提到的很多准则，这些准则直到现在也仍然成立。但是，DB2 功能的增强、其他领域中硬件和软件技术的变化将影响当前和将来的应用程序，知道这一点至关重要。当数据库性能成为开发过程中的焦点时，您的数据库设计使得为 DB2 应用程序提供最佳性能有了更大的可能性。