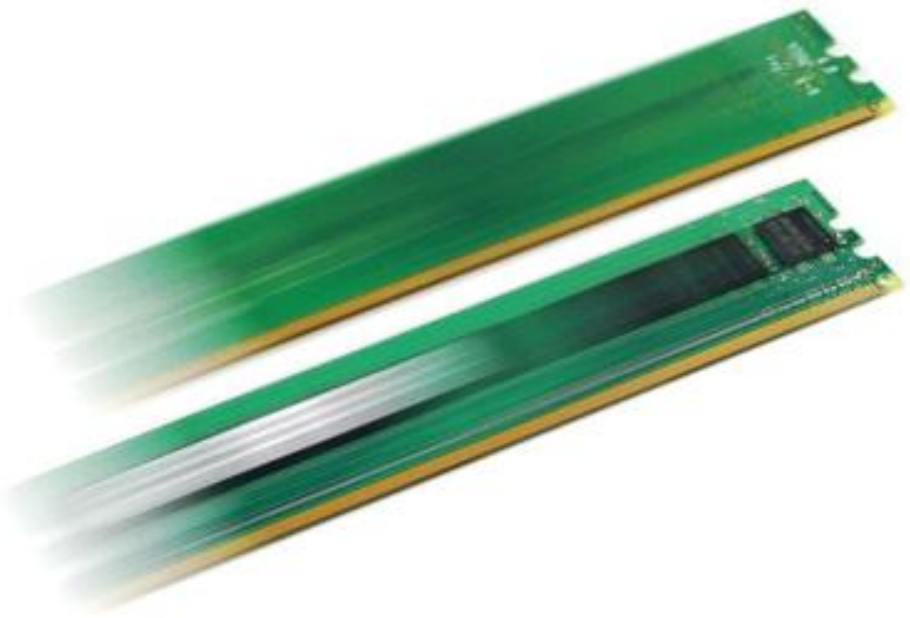
数据库电子书

主流内存数据库指南

内存数据库的目标是通过使用内存实现数据存储来提高吞吐量和降低延迟。这与使用磁盘存储的传统数据库管理系统不同，由于内部优化算法更简单，而且执行的 CPU 指令较少，所以内存内数据的速度比基于磁盘的数据库快。

- *内存数据库巡礼之 Oracle TimesTen*
- *内存数据库巡礼之 Sybase ASE*
- *嵌入式内存数据库 SQLite 简介*
- *VoltDB 内存数据库分析*

TechTarget 中国



主流内存数据库指南

内存数据库的目标是通过使用内存实现数据存储来提高吞吐量和降低延迟。这与使用磁盘存储的传统数据库管理系统不同。由于内部优化算法更简单，而且执行的 CPU 指令较少，所以内存内数据的速度比基于磁盘的数据库快。访问内存数据可以提高响应速度。——Mich Talebzadeh

内存数据库 Oracle TimesTen

随着 Oracle Exalytics 以及 SAP HANA 等内存分析设备的推出，业界的目光再一次聚焦在内存数据库产品之上。内存数据库(或 IMDB)可以是一个独立的数据库管理系统(DBMS)，如 Oracle 的 TimesTen，或者从属于 DBMS 的一个特殊数据库，如 SAP Sybase Adaptive Server Enterprise (ASE)。

IMDB 的目标是通过使用计算机内存实现数据存储来提高吞吐量和降低延迟。这与使用磁盘存储的传统数据库管理系统不同。由于内部优化算法更简单，而且执行的 CPU 指令较少，所以内存内数据的速度比基于磁盘的数据库快。访问内存数据可以提高响应速度。对于一些响应时间要求较高的应用程序，如交易、电信和国防系统，一般都会使用 IMDB。由于 IMDB 的这种特性，这些数据库使用内存要多于磁盘数据库产品。

Oracle TimesTen 和 Sybase ASE-IMDB 是一种使用过程外主内存的数据库。它们实现了 SQL 的完整支持，也支持一些特殊语言、安全性和数据库管理。这两种数据库都支持通过 SQL 访问数据。它们都具有一些磁盘数据库产品的特性。因此，使用这些产品缓存 SQL 后台持久化数据库的 SQL 请求就很简单。

TimesTen、ASE-IMDB 及现有的所有商业内存数据库都基于所谓的基于行的关系存储模型。这些产品很适合 OLTP 应用程序使用。

Oracle TimesTen 内存数据库是什么？

Oracle TimesTen 是一个全新设计的内存数据库。它使用基于行的关系模型(表、列、数据类型、索引等)实现数据存储，并使用 SQL 作为访问语言。它提供了许多 API，并且支持 Oracle PL/SQL。应用程序的访问方式与其他关系数据库完全相同。TimesTen 与传统数据库的主要区别在于它的性能要远远高于传统数据库。虽然 TimesTen 完全运行在内存中，但是它也能够在磁盘中创建数据库副本，支持数据库重启和恢复。通过检查点和事务日志，这种副本能够保持更新，因此能够从任何故障事故中恢复。TimesTen 支持一种用于实现高可用性的复制机制。它的 Cache Connect 功能可以作为后台 Oracle 数据库数据子集的高速缓存。在这种情况中，它就成为 Oracle 数据库的内存数据库缓存。

在 C/S 模式中，客户端 API 库通常使用 TCP/IP(但是有时也会使用 UNIX 域套接字或本地共享内存连接)与 TimesTen 服务器进程交换请求和响应，以执行实际的数据库访问。这正是在 TimesTen 数据库主机之外运行应用程序的常用模式。除了网络回程所造成的延迟，各个数据库访问造成的代码增加、环境切换等问题也会对性能造成影响。

只有当应用程序和 TimesTen 运行在同一台主机上时，才会使用直接模式。

在这种模式下，数据管理器 API 库基本上就是充当 TimesTen 数据库引擎的作用。API 调用是进程内函数调用，而 TimesTen 数据库(共享内存片段)会映射到应用程序的地址空间。这样可以消除数据库访问路径的切换竞争，实现最高的性能和最低的延迟，从而保持稳定的输出。

TimesTen 数据库也经常采用混合模式的并发访问，即不同的应用程序进程/线程会通过直接模式和客户端/服务器模式并发访问数据库。单个数据库可以支持最高 2000 个用户连接。如果 TimesTen 具有一个实例(管理多个 TimesTen 数据库)，那么用户连接数限制为 2500。

这两种模式在 API 层只有极少的差别，所以在一般的应用程序代码中，不需要知道或关心所使用的模式——它实际上只是一个在编译时或运行时的配置选择。

直接模式是 TimesTen 的特殊特性之一。大多数 TimesTen 生产部署都采用独立或直接模式。然而，在一些情况下也会使用客户端/服务器模式。

TimesTen 缓存不能通过一般的关系工具访问。如果应用程序拥有以数据库为中心的数据访问方式——即，使用 SQL 和 JDBS 访问数据，那么 TimesTen 是最佳的选择。然而，TimesTen 本身不支持与非 Oracle 数据库进行互操作，所以应用程序代码必须自行管理 TimesTen 与 Sybase 或其他数据库的数据传输。

TimesTen 的内存结构

TimesTen 内存结构比 Oracle 数据库简单很多。与 Oracle 不同，TimesTen 并没有数据库缓冲区、保存池或丢弃池的概念。传统数据库系统都会通过使用一些减少磁盘 I/O 的策略。这种设计策略源于磁盘 I/O 对数据库性能的影响。相反，内存数据库从一开始就不存在磁盘 I/O 问题。它们的最高优化目标就是减少内存和 CPU 需求。

TimesTen 目前支持两种索引方式：散列索引和 T 树索引。散列索引仅支持全键-值查找，但是速度非常快，而且执行过程与底层表的数量无关(只要它们的大小合适并且忽略小型表的硬件 L1/L2/L3 缓存效果)。这些索引具有很高的读取扩展性和很好的并发性。T 树索引的读取效率很高，但是散列索引效率不高。如果支持范围查找，那么它们会更灵活。但是，其中有一个弱点是在繁重写操作时并发性较差，因为每个索引修改都必须锁住整个索引。然而，在低延迟条件下，T 树索引的写性能非常好。

TimesTen 的应用

TimesTen 的主要价值在于响应时间，而吞吐量和高可用性则是第二位。TimesTen 一般更利于 OLTP 风格的工作负载，因为这些负载要求极低且极稳定的响应时间。此外，同样重要的是冗余性。通常，一个数据库服务器上会运行多个应用程序。在这些应用程序服务器上使用 TimesTen 可以减少数据库的负载，因此能够减少网络延迟和磁盘 I/O。

例如，有一些交易系统会在网页上显示价格信息。如果不使用 TimesTen，那么应用程序服务器就需要不停地从磁盘数据库查询价格信息。在达到每秒 100,000 次的查询水平时，数据库性能就会急剧下降。TimesTen 与应用层关系密切，而典型的 DBMS 则具有更广泛的用途。在实时响应与适度容量方面，使用 TimesTen 具有任何经典 DBMS 无法实现的压倒性优势。在一些数据快速变化的领域，如交易和销售数据管理，TimesTen 极有明显的优势，因为数据是实时传输的，而计算也必须以接近实时的速度完成，并且只对执行引擎产生很小的影响。另一些应用领域包括电信、国防和情报等机构。

（作者：Mich Talebzadeh　译者：曾少宁　来源：TT 中国）

内存数据库 Sybase ASE

ASE 在它的 15.5 版本中加入了自己的内存数据库(IMDB)。ASE-IMDB 可以追溯到 Sybase 的 Real Analytics Platform (RAP)，但这是 ASE 第一次以 IMDB 产品出现。

与 TimesTen 类似，ASE-IMDB 也是高性能数据库，它完全整合到 Sybase ASE 平台中。这一点与 TimesTen 相反，因为后者是一个完全独立的数据库。ASE-IMDB 可以读写同一个 Sybase ASE 中其他的数据库，并且可以接收其他 ASE 或非 ASE 数据库的数据。ASE-IMDB 还使用复制技术接收来自所有这些数据源的数据。

ASE 经典数据库是专门面向那些严格遵守 ACID(原子性、一致性、隔离性、持久性)事务语义的应用程序。这些 ACID 属性以提前写事务日志、定位持久化存储(例如，磁盘)等手段实现。在这个方面，ASE-IMDB 允许在低响应时间和高吞吐量的数据交换中放宽持久性和原子性要求。这一点与 TimesTen 相反，后者完全遵守 ACID。

要运行 ASE-IMDB，您必须拥有足够的缓存，才能够将整个数据库运行在内存中。一旦创建了这个专用的缓存，它就成为 IMDB 的设备载体，数据库就能够在这些内存设备上创建。ASE-IMDB 是基于一个可用的模板数据库创建的。模板

数据库是一个经典的 ASE 数据库。启动的 ASE-IMDB 会继承模板数据库的所有对象和数据。创建 ASE-IMDB 的典型语法是：

```
create inmemory database ASEIMDB
use ASEIMDB_template as template
on ASEIMDB_data01='4000M'
log on ASEIMDB_log01='1000M'
with durability = no_recovery
```

这种方法很简洁，数据库管理员都能够更容易地掌握。注意，在上面的语法中，它显示引用了一个模板数据库——这里是 ASEIMDB_template 。此外，持久性必须设置为 no recovery，这意味着 ASE-IMDB 数据库不可恢复。因此，ASE-IMDB 的所有内容在 ASE 服务器重启或意外关闭时都会丢失，因为这种方式不使用持久化存储。另一方面，它允许 Sybase 优化事务日志(它完全在内存中进行);由于在 ASE 重启时，IMDB 完全不需要从事务日志恢复，所以您可以获得更优的事务可扩展性和更高的性能。

ASE-IMDB 的索引算法没有任何变化。换而言之，数据库运行在内存中时，其消耗的存储空间不会有任何变化。在处理大容量数据时，需要增加大量的内存，才能够支撑内存内运行。由于您可以将常规的 ASE 数据库转储和加载到 ASE-IMDB 中，所以您可以使用 ASE 经典数据库的现有索引，从而实现全面兼容性。

ASE-IMDB 的应用

如果已经在使用 ASE，那么只需要获得授权就可以使用 ASE-IMDB。ASE-IMDB 主要面向写密集型应用程序，其中数据持久化是第二位考虑特性。我认为，应用程序必须清晰规定 IMDB 是否支持可恢复性。有一些应用程序并不需要恢复支持。这些应用程序可以部署 ASE-IMDB 以提高性能。

由于 ASE-IMDB 可以完全整合在混合结构的经典服务器中，它完全支持 ASE 本身的 SQL 语法、安全性和加密。电子商务、购物车、特殊交易系统及清理数据后提交到经典数据库的分段/中间数据库，都是适合使用 ASE-IMDB 的例子。此外，它还能够通过一些普通复制方法从其他数据源接收数据，这使得 ASE-IMDB 成为一些有快速响应需求公司的首选工具。ASE-IMDB 快速复制到其他数据库的功能仍在开发中。然而，如果您的应用程序迫切需要恢复功能，那么 ASE-IMDB 可能不适合这个要求。

（作者：Mich Talebzadeh 译者：曾少宁 来源：TT 中国）

开源内存数据库 SQLite

SQLite 是 D. Richard Hipp 用 C 语言编写的开源嵌入式数据库引擎。它是完全独立的，不具有外部依赖性。它是作为 PHP V4.3 中的一个选项引入的，构建在 PHP V5 中。SQLite 支持多数 SQL92 标准，可以在所有主要的操作系统上运行，并且支持大多数计算机语言。SQLite 还非常健壮。其创建者保守地估计 SQLite 可以处理每天负担多达 100,00 次点击率的 Web 站点，并且 SQLite 有时候可以处理 10 倍于上述数字的负载。

功能

SQLite 对 SQL92 标准的支持包括索引、限制、触发和查看。SQLite 不支持外键限制，但支持原子的、一致的、独立和持久（ACID）的事务（后面会提供有关 ACID 的更多信息）。

这意味着事务是原子的，因为它们要么完全执行，要么根本不执行。事务也是一致的，因为在不一致的状态中，该数据库从未被保留。事务还是独立的，所以，如果在同一时间在同一数据库上有两个执行操作的事务，那么这两个事务是互不干扰的。而且事务是持久性的，所以，该数据库能够在崩溃和断电时幸免于难，不会丢失数据或损坏。

SQLite 通过数据库级上的独占性和共享锁定来实现独立事务处理。这意味着

当多个进程和线程可以在同一时间从同一数据库读取数据，但只有一个可以写入数据。在某个进程或线程向数据库执行写入操作之前，必须获得独占锁定。在发出独占锁定后，其他的读或写操作将不会再发生。

内部结构

在内部，SQLite 由以下几个组件组成：SQL 编译器、内核、后端以及附件。SQLite 通过利用虚拟机和虚拟数据库引擎（VDBE），使调试、修改和扩展 SQLite 的内核变得更加方便。所有 SQL 语句都被编译成易读的、可以在 SQLite 虚拟机中执行的程序集。

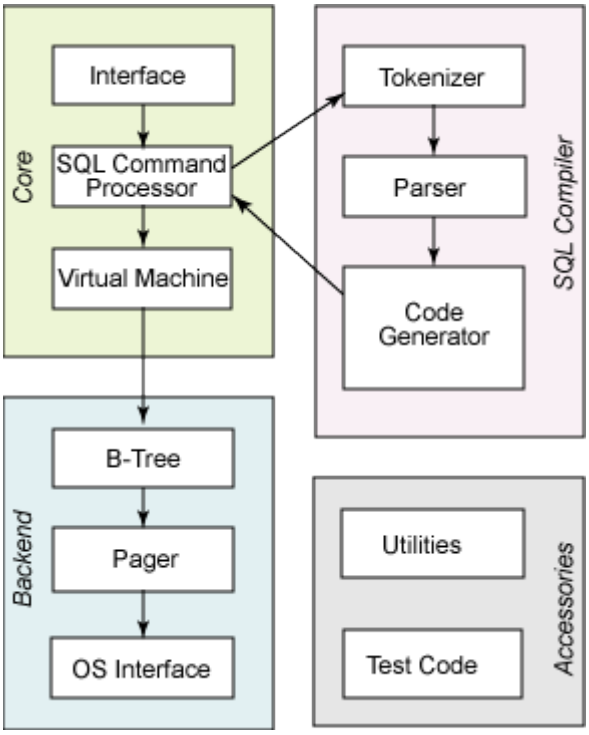


图 1. SQLite 的内部结构

SQLite 支持大小高达 2 TB 的数据库，每个数据库完全存储在单个磁盘文件

中。这些磁盘文件可以在不同字节顺序的计算机之间移动。这些数据以 B+树（B+tree）数据结构的形式存储在磁盘上。SQLite 根据该文件系统获得其数据库权限。

数据类型

SQLite 不支持静态数据类型，而是使用列关系。这意味着它的数据类型不具有表列属性，而具有数据本身的属性。当某个值插入数据库时，SQLite 将检查它的类型。如果该类型与关联的列不匹配，则 SQLite 会尝试将该值转换成列类型。如果不能转换，则该值将作为其本身具有的类型存储。

SQLite 支持 NULL、INTEGER、REAL、TEXT 和 BLOB 数据类型。

管理 SQLite

SQLite 附带一个可下载的 command-line interface for database administration。通过数据库名称可以调用此命令行程序，并且可以按照下面的方式创建新的数据库和表：

清单 1. 创建新的数据库和表

```
C:\minblogg>sqlite3 c:\minblogg\www\db\alf.db
SQLite version 3.2.1
Enter ".help" for instructions
sqlite> create table mytable(name varchar(40), age smallint);
sqlite> insert into mytable values('Nils-Erik',23);
sqlite> select * from mytable;
Nils-Erik|23
```

```
sqlite>
```

然后，可以再次打开该数据库，列出它的表和架构，并继续进行插入和删除值的操作。

清单 2. 列出表和架构

```
C:\minblogg>sqlite3 c:\minblogg\www\db\alf.db
SQLite version 3.2.1
Enter ".help" for instructions
sqlite> .tables
mytable
sqlite> select * from mytable;
Nils-Erik|23
sqlite> .schema
CREATE TABLE mytable(name varchar(40), age smallint);
sqlite
```

SQLite 还附带命令行数据库分析器，该分析器允许您显示关于任何 SQLite 数据库当前状态的详细信息。

清单 3. SQLite 分析器

```
C:\minblogg>sqlite3_analyzer www\db\alf.db
Analyzing table mytable...
Analyzing table sqlite_master...
/** Disk-Space Utilization Report For www\db\alf.db
*** As of 2005-Apr-24 18:56:40
Page size in bytes..................... 1024
Pages in the whole file (measured).... 2
Pages in the whole file (calculated).. 2
Pages that store data.................. 2          100.0%
Pages on the freelist (per header).... 0            0.0%
Pages on the freelist (calculated).... 0            0.0%
Pages of auto-vacuum overhead......... 0            0.0%
Number of tables in the database...... 2
```

```
Number of indices..................... 0
Number of named indices.............. 0
Automatically generated indices....... 0
Size of the file in bytes............. 2048
Bytes of user payload stored.......... 13           0.63%
*** Page counts for all tables with their indices ********************
MYTABLE............................... 1           50.0%
SQLITE_MASTER......................... 1           50.0%
*** All tables ******************************************************
Percentage of total database.......... 100.0%
Number of entries..................... 2
Bytes of storage consumed............. 2048
Bytes of payload...................... 91           4.4%
Average payload per entry............. 45.50
Average unused bytes per entry........ 916.50
Maximum payload per entry............. 78
Entries that use overflow............. 0            0.0%
Primary pages used.................... 2
Overflow pages used................... 0
Total pages used...................... 2
Unused bytes on primary pages......... 1833         89.5%
Unused bytes on overflow pages........ 0
Unused bytes on all pages............. 1833         89.5%
*** Table MYTABLE ***************************************************
Percentage of total database..........  50.0%
Number of entries..................... 1
Bytes of storage consumed............. 1024
Bytes of payload...................... 13           1.3%
Average payload per entry............. 13.00
Average unused bytes per entry........ 999.00
Maximum payload per entry............. 13
Entries that use overflow............. 0            0.0%
Primary pages used.................... 1
Overflow pages used................... 0
Total pages used...................... 1
Unused bytes on primary pages......... 999          97.6%
Unused bytes on overflow pages........ 0
```

```
Unused bytes on all pages.............. 999          97.6%
```

由于完全能够使用命令行界面来管理数据库，因此它可以为数据库管理员带来很大的方便。目前有许多优秀的基于 Web 的 SQLite 数据库管理系统。其中有一个是基于 PHP 的 SQLiteManager。

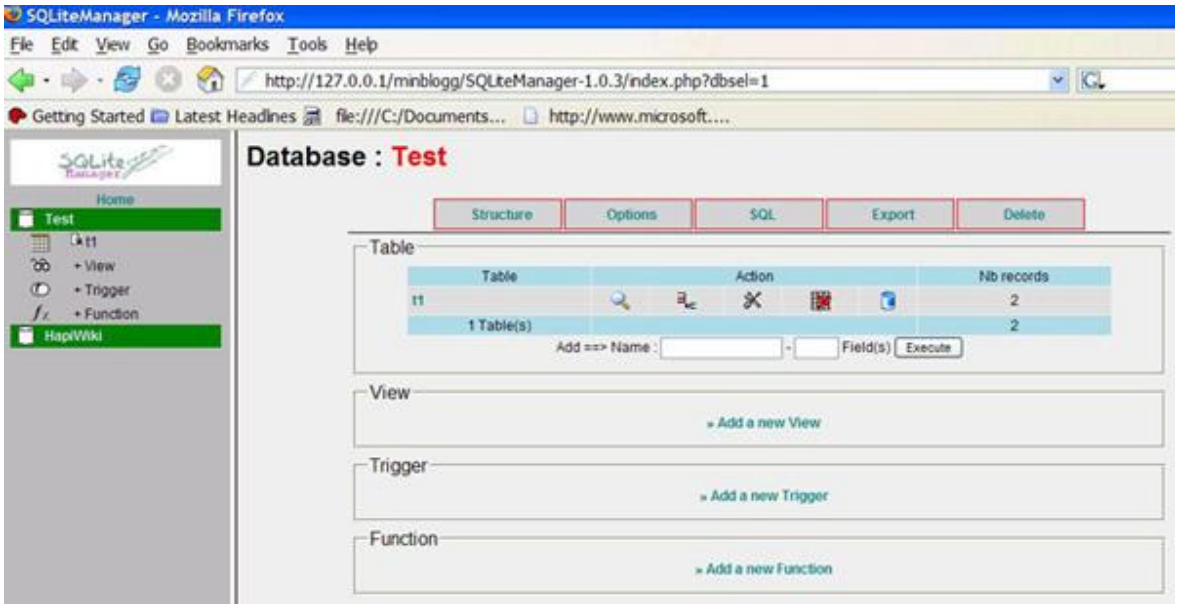


图 2. 使用 SQLiteManager 管理数据库

备份

备份 SQLite 数据库有两种方法。如果数据库正在使用中，则应从命令行界面使用 .dump 命令。这样可以创建一个包含必要命令和数据的文件，从而重新创建数据库。.dump 命令也可以用于备份数据库表。

清单 4. .dump 命令

```
sqlite> .dump
BEGIN TRANSACTION;
CREATE TABLE mytable(name varchar(40), age smallint);
```

```
INSERT INTO "mytable" VALUES('Nils-Erik', 23);
COMMIT;
sqlite>
```

如果数据库没有处于使用状态，则可以直接将数据库文件复制到安全位置。

在 PHP V5 中使用 SQLite

一个好的做法是将 SQLite 数据库与 PHP 代码分开。完成此操作的一个简便方法是创建一个 www 目录。在此目录中，创建一个用于存放 SQLite 数据库的 db 目录、一个用于存放数据库和表创建脚本的 dbscripts 目录和一个用于存放数据库备份的 backups 目录。

清单 5. 使用 PHP V5 组织 SQLite 数据库

```
2004-12-06  15:43    DIR          .
2004-12-06  15:43    DIR          ..
2005-04-23  19:55    DIR          db
2005-01-02  11:46    DIR          dbscripts
2005-01-02  11:46    DIR          backups
```

在 PHP V5 中创建 SQLite 数据库与在命令行界面中创建该数据库非常相似。如果该数据库不存在，则创建一个空数据库。

```
$db = sqlite_open('../db/ac.db');
```

创建一个表也非常容易：

清单 6. 创建表

```
$db = sqlite_open('../db/ac.db');
sqlite_query($db, 'DROP TABLE post');
sqlite_query($db, 'CREATE TABLE post (id INTEGER PRIMARY KEY,
   kategori VARCHAR(20) NOT NULL,
```

```
titel VARCHAR(75) NOT NULL, referens VARCHAR(75), status VARCHAR(20) not null,
date varchar(10)not null, synopsis VARCHAR(120), inlaegg varchar(8192))');
```

插入一条记录：

```
$sqldb = sqlite_open("../db/ac.db");
sqlite_query($sqldb, "INSERT INTO isvs VALUES ('$isvurl' , '$isvname', '$comment')");
```

并选择数据：

清单 7. 从 SQLite 数据库中选择数据

```
$sqldb = sqlite_open("www/db/ac.db");
$results = sqlite_query($sqldb, "SELECT * FROM isvs order by isvurl asc ");

while (list($isvurl, $isvname) = sqlite_fetch_array($results)) {
  sqlite_close($sqldb);
```

使用 SQLite 和数据库抽象层

两个先进的开源数据库抽象层对 SQLite 提供支持：PEAR::DB，它们包含在 PHP V5 中，并且被认为是更轻量级的 ezSQL。通过预先使用 PHP 扩展和应用程序库 (PEAR) 或 ezSQL，可将 SQLite 用于应用程序的快速复原，在以后需要时，可以将其无缝转向更具工业性质的数据库。

清单 8. 使用 ezSQL 和 SQLite

```
$users = $db->get_results("SELECT name, age FROM table1");
foreach ( $users as $user )
{
            echo $user->name;
            echo $user->age;
}
```

SQLite 使用注意事项

在确定是否在应用程序中使用 SQLite 之前，应该考虑以下几种情况：

- 目前没有可用于 SQLite 的网络服务器。从应用程序运行位于其他计算机上的 SQLite 的惟一方法是从网络共享运行。这样会导致一些问题，像 UNIX® 和 Windows® 网络共享都存在文件锁定问题。还有由于与访问网络共享相关的延迟而带来的性能下降问题。
- SQLite 只提供数据库级的锁定。虽然有一些增加并发的技巧，但是，如果应用程序需要的是表级别或行级别的锁定，那么 DBMS 能够更好地满足您的需求。
- 正如前面提到的，SQLite 可以支持每天大约 100,00 次点击率的 Web 站点 —— 并且，在某些情况下，可以处理 10 倍于此的通信量。对于具有高通信量或需要支持庞大浏览人数的 Web 站点来说，应该考虑使用 DBMS。
- SQLite 没有用户帐户概念，而是根据文件系统确定所有数据库的权限。这会使强制执行存储配额发生困难,强制执行用户许可变得不可能。
- SQLite 支持多数（但不是全部）的 SQL92 标准。不受支持的一些功能包括完全触发器支持和可写视图。请参阅 unimplemented SQL92 features。

如果您感到其中的任何限制会影响您的应用程序，那么您应该考虑使用完善的 DBMS。如果您可以解除这些限制问题，并且对快速灵活的嵌入式开源数据库引擎很感兴趣，则应重点考虑使用 SQLite。

一些能够真正表现 SQLite 优越性能的领域是 Web 站点，可以使用 SQLite 管理应用程序数据、快速应用程序原型制造和培训工具。

结束语

由于资源占用少、性能良好和零管理成本，嵌入式数据库有了它的用武之地，它将为那些以前无法提供用作持久数据的后端的数据库的应用程序提供了高效的性能。现在，没有必要使用文本文件来实现持久存储。SQLite 之类的嵌入式数据库的易于使用性可以加快应用程序的开发，并使得小型应用程序能够完全支持复杂的 SQL。这一点对于对于小型设备空间的应用程序来说尤其重要。

嵌入式数据库对于加快应用程序开发也很重要，尤其是在用于数据库抽象层（例如 PEAR::DB 或 ezSQL）时。最后，SQLite 正在积极开发中，未来一定会有新的功能，会对开源社区更有用。

（作者：Nils-Erik Frantzell 来源：DeveloperWorks）

内存数据库 IBM SolidDB

作为一种内存中关系数据库，IBM solidDB 受到全球的追捧，因为它能够提供超快的速度和超高的可用性。 顾名思义，内存中数据库完全驻留在主存中，而不是磁盘上，这使得数据访问比传统的基于磁盘的数据块要快一个数量级。这种飞跃一定程度上是由于 RAM 能够比硬盘驱动器提供更快的数据访问。

但是，solidDB 还有专为存储、搜索和处理主存中数据而设计的数据结构和访问方法。因此，即使普通的基于 磁盘的数据库将数据完全缓存在内存中，solidDB 仍可以胜出一筹。有些数据库可以提供较短的延时，但是不能处理大量的事务或并发会话。IBM solidDB 可以提供每秒数万至数十万事务的吞吐率，并且始终可以获得微秒级的响应时间（或延时）。本文探索内存中数据库与基于磁盘的数据库在结构上的差别，以及 solidDB 如何提供超快的速度。

RDBMS 的历史

当 20 世纪 60 年代数据管理系统刚刚出现时，磁盘驱动器是唯一可以在合理时间内存储和访问大量数据的地方 。RDBMS 设计者的精力主要集中于优化 I/O 和设法用驱动器的块结构来安排数据访问模式。设计策略常常围绕着共享缓冲池，数据块存放在共享缓冲池中以便重用。随着访问方法的发展，出现了像著名的 B+ 树（一种块优化索引）之类的解决方案。

与此同时，查询优化策略注重尽可能减少页面读取。在对性能的激烈争夺中，磁盘 I/O 常常是最致命的敌人，为了避免磁盘访问，往往需要牺牲处理效率。例如，对于典型的 8 KB 或 16 KB 的页面，页内处理天性是连续的，CPU 效率低于随机数据访问。然而，它仍是减少磁盘访问的流行方法。

当内存富足时代到来时，很多 DBA 不断增加缓冲池，直到缓冲池大到足以容纳整个数据库 ，这便产生了全缓存数据库（fully cached database）的概念。但是，在 RAM 缓冲池中， DBMS 仍受累于效率低下的、结构化的、面向块的 I/O 策略，这种策略原本是为处理硬盘驱动器 而创建的。

将块抛开

内存中数据库系统一个最值得注意的不同之处在于没有大数据块结构。 IBM solidDB 消除了这种块。表行和索引节点独立地存储在内存中， 所以可以直接添加数据，而不必重新组织大块结构。 内存中数据库还放弃使用大块索引（有时也称丛生树），以利于 精简结构，增加索引层数，将索引节点最小化，以避免节点内处理的成本。 最常见的内存中数据库索引策略是 T-树。然而，IBM solidDB 却使用 一种称作 trie（或前缀树）的索引，这种索引最初是为文本搜索而创建的，但是最终成为极佳的内存中索引策略。trie（此名源于单词 retrieval） 由一系列的节点组成，其中，一个给定节点的后代具有与该节点关联的相同的 字符串前缀。例如，如果单词 “dog” 被存储为 trie 中的一个节点，它将是 包含 “do” 的

节点的后代，而后者又是包含 “d” 的节点的后代。

Trie 索引可以减少键值比较，并且几乎可以消除节点内处理，从而能够提高性能。 索引包含一个节点，该节点是一个小型的指针数组，这些指针又指向更低的层。 在此，不必使用整个键值通过遍历树来进行比较，键值被分割为一些小块，每个块包含 数个比特位。每个小块便是相应层的指针数组的直接索引： 最左边的小块指向第一层节点，第二个小块指向第二层的节点，依此类推。 因此，只需进行几次数组元素的检索，便可完成整个搜索。而且，每个索引节点 是一个小数据块（在 solidDB 中大约为 256 字节），这可以带来好处， 因为这种数据块正好适合现代处理器缓存，从而可以通过有效促进缓存的使用 提高处理器的效率。这些小型数据数组是现代处理器中最有效的数据结构， solidDB 经常使用它们来最大化性能。

检查点和耐久性：提速之路

IBM solidDB 还使用其他一些技术来加快数据处理，首先便是一种获得专利的检查点（checkpointing）方法， 这种方法产生一个快照一致性检查点，同时并不阻塞正常的事务处理。 快照一致性检查点使数据库只需从一个检查点重新启动。 其他数据库产品通常不允许那样，而必须使用事务日志文件来重新计算一致状态（而 solidDB 则允许必要时关闭事务日志记录）。solidDB 解决方案之所以能够实现， 是因为做到了分配行镜像和行影子镜像（相同行的不同版本），而不

必使用低效的 块结构。只有那些与一致性快照相符的镜像被写到检查点文件，行影子使当前执行的事务 可以在检查点创建期间不受限制地运行。

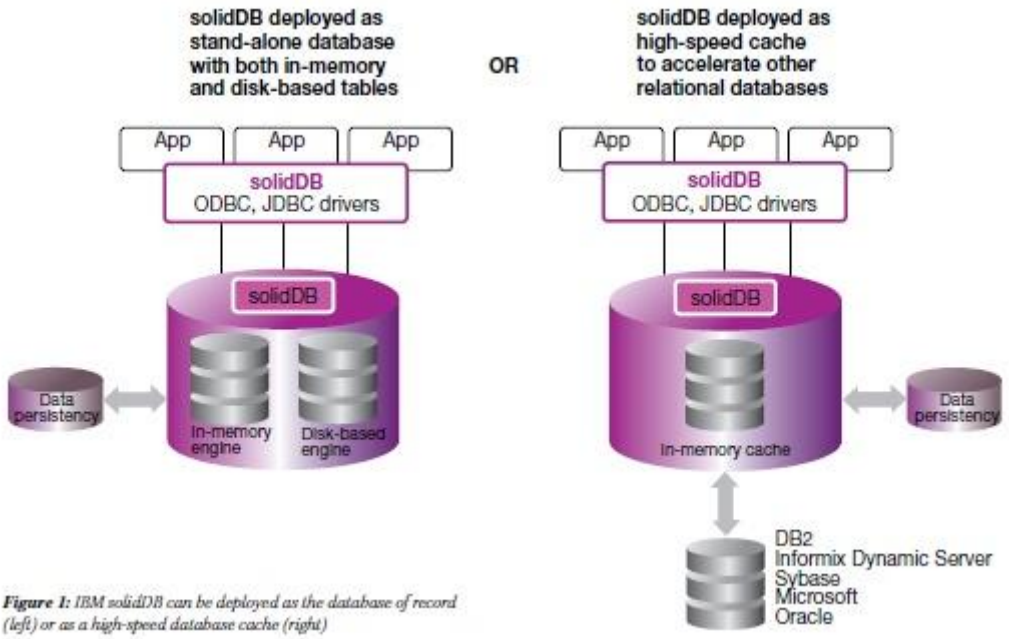


Figure 1: IBM solidDB can be deployed as the database of record (left) or as a high-speed database cache (right)

而且，solidDB 查询优化器通过以一种新的方式估计执行成本，判别内存 中的表的不同性质。查询优化集中于 CPU 密集型（CPUbound）执行路径，而全缓存数据库 将仍然集中于优化取页到大容量存储器的操作，而这已不再是问题。

IBM solidDB 使用的另一种技术是放宽事务持久性（durability）。在过去，数据库总是支持完全持久性，以保证事务提交时写的数据持久不变。 问题是，这种完全持久性会造成同步日志写，因而需要消耗资源，降低 响应速度。在很多情况下，为取得更快的响应速度，对于某些任务接受 较短的持久性，这是非常值得的。对于 solidDB，可以为给定的数据库会话乃至整个事务在运行时放宽事务持久性。

IBM solidDB 还通过帮助开发人员避免客户端/服务器交互中的进程上下文切换， 提高数据库性能。通过使用 solidDB 提供的、包含完整查询执行代码的数据库访问驱动程序， 开发人员可以有效地将应用程序与 DBMS 代码链接起来，并使用共享内存在应用程序之间共享数据。

一旦应用了所有这些措施，当应用程序负载大到使传统数据库中需要产生大量 I/O 时， 使用 solidDB 将使吞吐率有数量级的提高。而且，响应速度的提高甚至更加惊人： 查询事务的延时通常是 10 到 20 微秒，更新事务的延时通常少于 100 微秒。 在传统的基于磁盘的数据库中，对应的时间通常是以毫秒计算的。

solidDB 的速度和威力

除了这些性能优点外，solidDB 还带来其他好处。 它将一个完全事务性的内存中数据库和一个强大的、基于磁盘的数据库 组合到一个紧凑的解决方案中，并且可以透明地将同一个数据库的一部分 留在内存中，一部分留在磁盘上。而且，IBM solidDB 是市场上唯一一个 可以作为几乎任何其他基于磁盘的关系数据库的前端高速缓存来部署的产品（见图 1）。 最后，solidDB 还提供超高的可用性，将可用时间由通常的 5 个 9 提高到 99.9999%。 换句话说，如果您要寻求超快的速度，那么将会找到 IBM solidDB，但这只是 IBM solidDB 的开端。

（作者：Antoni Wolski，Sally Hartnell 来源：DeveloperWorks）

分布式内存数据库 VoltDB

VoltDB 是一个宣称性能超过 Mysql 100 倍的新型数据库。它源自 Micheal Stonebraker 一篇论文 H-Store。在这篇论文发表后，Stonebraker 成立了 VoltDB 公司带着他的一些学生开始在 OLTP 数据库领域打拼。Stonebraker 从上世纪 70 年代——数据库刚开始发展的时间——就开始在数据库领域活跃，这样的老古董提出的数据库的新想法，给了整个存储领域很大的想象空间。

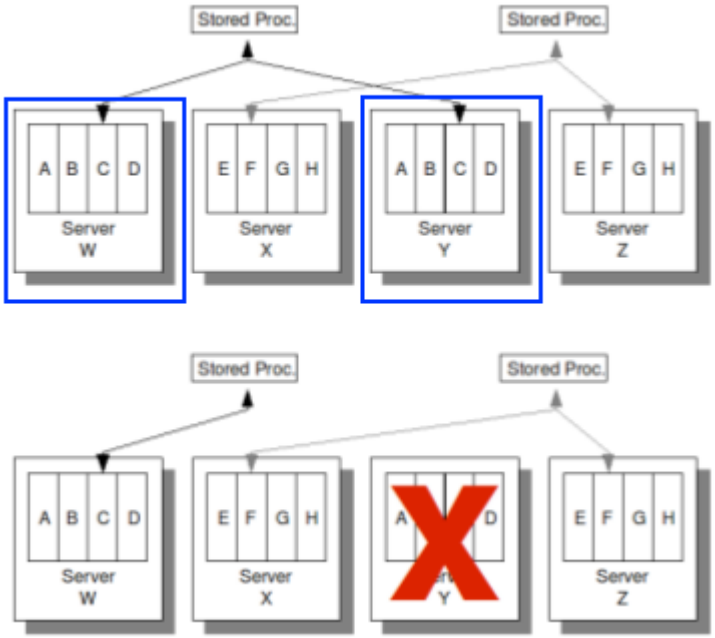
VoltDB 源起于应用领域与硬件发展翻天覆地的变化。用户的使用方法发生了变化，在数据库开始发展的阶段，事务是一个较长的过程，用户或者管理员可以在"BEGIN TRANSACTION"和"END TRANSACTION"之间慢慢地人工执行整个事务的步骤。但是现在，大部分操作是由 Web 服务端发起的书写良好的事务，用户访问的是 Web 服务器，在 Web 服务器的执行逻辑里再访问数据库，所以即使是很复杂的事务也可以很快执行完。计算机硬件的发展更是一日千里。几十 GB 的内存服务器已经很常见。以太网络也已经步入 Gbps 时代，而且正在朝向 10Gbps 方向迈进。基于以太网的集群的机器价格也降低到比 PC 机贵不了太多。VoltDB 的设计充分利用了这些特点，数据主要存储在内存中，Shared Nothing 的集群结构，单机是单线程处理事务，不是用锁而是基于 Optimistic 的方法处理事务并发，所有的事务必须以存储过程形式先提交到 VoltDB 系统。下面分开来说。

事务提交

既需要支持复杂的事务操作，又需要快速的执行过程，VoltDB 采取了一个比较极端的事务提交方式。虽然 VoltDB 支持部分 SQL 语句接口，但是不允许用户使用传统的”BEGIN TRANSACTION”和”END TRANSACTION”的语法模式，而是完全基于存储过程。用户通过写存储过程完成应用程序的逻辑，作为一个先置条件将存储过程提交到 VoltDB。运行时，用户程序调用存储过程完成事务操作，所有事务的运行逻辑是由 VoltDB 在服务器进程中完成的。这种方式保证了事务不会被人为打断，并且服务器可以预先判断各个事务的逻辑，也为事务并发处理挖掘信息。

数据分布

VoltDB 使用 Shared Nothing 结构，整个数据库的数据分散到集群的多台机器上。VoltDB 的数据分布策略是基于哈希的，存储在 VoltDB 中的每一张表，对数据的主键哈希取模后的结果对应于数据存储的节点。相比较于 BigTable 基于主键的连续范围分段的方法，哈希方法的好处是数据分散的均匀，没有动态数据调整的烦恼。但也有很多缺点，采用这种方法后，集群的规模是事先确定好的，新增机器需要停止服务后重新分布数据。另外，数据哈希被分散后，数据的连续性被打乱了，在这个数据结构上做范围查询需要动用服务这张表的所有机器，这个后面会祥说。



上面这张图描述了数据的分布方式，VoltDB 集群的每台机器都会服务多张表。从图里还能看到 VoltDB 的数据复制是基于机器单位的，蓝色框圈住的两台机器内的数据是完全同构的。

VoltDB 的哈希分布数据的方法是系统设计的简化，这种简化让 VoltDB 工程实现难度降低，可以快速的商用。天下没有免费的午餐，这个设计也是 VoltDB 功能缺陷，导致 VoltDB 无法动态扩容以及其他一些问题。

数据一致性

同一份数据的多个副本之间需要保证数据一致性，VoltDB 采用所有修改操作在每一个副本上单独更新的方式。如何保证更新操作在所有副本上以相同的顺序更改而不至于产生不一致，这就要提到 VoltDB 的并发控制方式。

VoltDB 的事务并发控制需要依赖于集群内所有机器的时间是一致的，这个可以使用 NTP 之类的时间同步协议，保证机器之间的时间差异远远小于一个交换机下的两台机器之间的 Round Trip 时间。VoltDB 对于用户每一次事务的调用分配一个时间戳，并且保证这个时间戳是全局有序的，虽然时间戳是由集群中的各台机器独自分配的，但是加上机器的序号，可以保证（机器序号，时间戳）的组合值是全局有序的。一台服务器执行事务之前，需要等待 Round Trip 时间后，如果其他机器没有开始比自己更早的事务，那么就执行自己的事务。以这种方式保证集群内多台机器之间事务的有序。数据的多个副本的更新操作也都以相同的顺序进行修改，所有副本之间保证了一致性。

事务并发处理

为了充分发挥多核机器的性能，而又不引入多线程执行事务的复杂性，VoltDB 的数据分片规模是按照集群核数来划分的。一台物理机器上可能运行多个 VoltDB 服务器进程，每个进程对应于一个核，服务器进程之间都是通过网络进行通信。在单个进程内，只使用单线程，所有的事务执行都是顺序进行的。

多个事务在多个服务器节点同时执行，VoltDB 保证如果事务之间有冲突，那么事务的执行是完全隔离的，即达到 SERIALIZABLE ISOLATION。VoltDB 会事先分析好存储过程之间的关系，如果两个事务可能存在冲突，则不让这两个进程在同一个时间执行。

在 VoltDB 的并发处理中，每一个事务在执行之前都要等待一个 Round Trip 时间，显然会增加事务执行的时延。这么做是为了确保别的节点没有发起比这个事务更早的事务，保证事务执行的顺序。在实现中，VoltDB 用了另外一种优化方法。例如 A，B 两个节点，分别要执行事务 1 和 2，A 节点开始执行事务 1 的时间是 T1，如果 A 收到 B 发了事务 2 的执行需求，并且 T2 > T1，那么 A 节点可以确认从 B 节点不会有更早的事务再发送过来，A 节点就不必等 Round Trip 时间，可以直接执行事务 1。当整个系统压力比较大时，这个优化方法效果尤其明显，事务的时延有效降低。

VoltDB 还花了很大精力在处理事务之间的逻辑关系，尽可能对事务分门别类进行处理，以期获得更好的性能。

范围查询的处理

VoltDB 取巧的采用的哈希的方法做数据分布，在面对范围查询的需求时，再次吃到苦果。哈希方法打乱了数据的连续性，对于范围查询的处理能力显著下降。VoltDB 执行某张表的范围查询，需要发送这个查询到这张表的所有数据分片上。在所有分片完成同样的范围查询，再将结果汇总，才能得到全局的准确结果。所以 VoltDB 处理范围查询会很低效

数据持久化

虽然 Stonebraker 在 H-Store 的论文里反复提到，在内存型数据库里，即使

使用 Group Commit 写操作日志也是非常低效的，但是为了保证数据的持久性，VoltDB 还是不得不采用记操作日志的办法。VoltDB 使用定期做 Snapshot 加上记操作日志来保证数据持久性，这种方法没有什么特别的地方。

（作者：淘宝核心系统团队　来源：博客）

我们的编辑团队

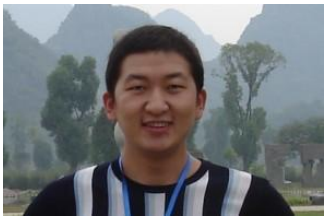
您若有何意见与建议，欢迎与我们的编辑联系。

诚挚感谢以下人员热情参与 TechTarget 中国《数据库电子书》的内容编辑工作！

诚邀更多的数据库专业人士加入我们的内容建设团队！

曾少宁

TechTarget中国特邀技术编辑。软件工程硕士学学位，4年以上软件开发经验，熟悉Oracle、Java以及Linux等领域，曾经任职于juniper等著名企业，目前从事计算机教学工作。



冯昀晖

TechTarget中国特邀技术编辑。资深软件工程师，有超过7年的政府和企业信息化软件解决方案经验，熟悉SQL Server、Oracle等数据库技术，爱好阅读、健身和中国象棋。

孙瑞

TechTarget 中国高级网站编辑，四年网络媒体从业经验。负责“TT 数据库”和“SearchBI”网站的内容建设，熟悉数据库以及商业智能等企业信息化领域，拥有计算机学士学位。