HSQLDB 中文文档

English Doc Version: 2007/8/28

Chinese Doc Version: 2008/8/1

role0523@Gmail.com

翻译说明：

Hsqldb 是一个优秀的轻量级开源的纯 Java SQL 数据库，为了方便 Hsqldb 在国内推广我们将 2007/8/28 发布的用户指南翻译成中文，不过我们建议有能力的读者阅读英文原文。大家可以从 Hsqldb 官方网站(http://hsqldb.org)获取到 Hsqldb 的发布包、源代码和文档。

对于该版本的翻译，将会不定期更新到项目主页 http://code.google.com/p/hsqldb-zh-doc/ 上，欢迎下载。

版权声明

致谢

目 录

前言 ..... **IV**

第 1 章 运行、使用 HSQLDB **1**

1.1 简介 1
1.2 运行工具 1
1.3 运行 HSQLDB 2
1.4 SERVER 模式 2
1.4.1 Hsqldb Server *3*
1.4.2 Hsqldb Web Server *3*
1.4.3 Hsqldb Servlet *3*
1.4.4 In-Process(Standalone)模式 *4*
1.4.5 Memory-Only 数据库 *5*
1.5 一般操作 5
1.5.1 关闭数据库 *5*
1.5.2 在一个 JVM 里使用多个数据库 *6*
1.5.2 创建新数据库 *6*
1.6 使用数据库引擎 6
1.6.1 不同的表类型 *7*
1.6.2 约束和索引 *7*
1.6.3 SQL 支持 *8*
1.6.4 JDBC 支持 *8*

第 2 章 SQL 问题 **9**

2.1 本章目的 9
2.2 对 SQL 标准的支持 9
2.3 约束和索引 9
2.3.1 主键约束 *9*
2.3.2 唯一性约束 *10*
2.3.3 唯一性索引 *10*
2.3.4 外键 *10*
2.3.5 索引和查询速度 *11*
2.3.6 Where 条件或者连接 *12*
2.3.7 子查询和连接 *13*
2.4 类型和算术操作 13
2.4.1 整型类型 *13*
2.4.2 其他数字类型 *14*
2.4.3 二进制和 Boolean 类型 *15*
2.4.4 Java 对象的存储和操作 *15*
2.4.5 类型的大小、精度和范围 *16*
2.5 序列和标识 16
2.5.1 标识自动增长列 *16*
2.5.2 序列 *17*
2.5.3 事务的问题 *17*
2.5.4 新特性和变化 *18*

第 3 章 UNIX 下快速起步 **19**

3.1 本章目的 19
3.2 安装 19
3.3 建立一个 HSQLDB 持久数据库实例和一个 HSQLDB SERVER 21
3.4 访问你的数据库 23
3.5 创建其他的帐号 26

3.6 关闭数据库 27
3.7 在系统后台运行 HSQLDB 27
3.7.1 可移植的 HSQLDB 初始化脚本 *27*
3.7.2 初始化脚本配置步骤 *28*
3.7.3 解决脚本配置过程中的问题 *32*

第 4 章 高级话题 **34**

4.1 本章目的 34
4.2 数据库连接 34
4.2.1 连接属性 *35*
4.3 属性文件 36
4.3.1 Server and Web Server 属性 *36*
4.3.2 从应用程序中启动 Server *37*
4.3.3 个别数据库属性 *37*
4.4 配置数据库属性的 SQL 命令 40

第 5 章 部署问题 **41**

5.1 本章目的 41
5.2 操作模式和表 41
5.2.1 操作模式 *41*
5.2.2 表 *41*
5.2.3 大对象 *42*
5.2.4 部署的环境 *42*
5.3 内存和磁盘使用 42
5.3.1 缓冲内存分配 *43*
5.4 管理数据库连接 44
5.5 升级数据库 45
5.5.1 使用脚本命令升级 *45*
5.5.2 手动改动.script 文件 *46*
5.6 备份数据库 46

第 6 章 TEXT 表 **47**

6.1 前言 47

第 7 章 TLS **48**

第 8 章 SQLTOOL **49**

第 9 章 SQL 语法 **50**

附录 A 建立 HSQLDB **51**

附录 B 第一个 JDBC 客户端例子 **52**

附录 C HSQLDB 数据库文件及其恢复 **53**

附录 D 在 OPENOFFICE.ORG 1.1.X 中运行 HSQLDB **54**

附录 E HSQLDB 测试工具 **55**

附录 F 数据库管理器 **56**

附录 G 传输工具 **57**

前言

如果大家在阅读过程中发现我们在翻译存在异议，请与作者联系。

第 1 章 运行、使用 Hsqldb

1.1 简介

hsqldb.jar 包位于/lib 目录下，它包含了一些组件和程序。每个程序需要不同的命令来运行。

Hsqldb.jar 包中有下列这些组件：

- HSQLDB RDBMS
- HSQLDB JDBC Driver
- Database Manager (Swing and AWT versions)
- Transfer Tool (AWT version) ）
- Query Tool (AWT)
- Sql Tool (command line)

其中、HSQLDB RDBMS 和 JDBCDriver 提供了 HSQLDB 的核心功能。其余的组件都是通用的数据库工具。这些通用工具可以使用在任何带有 JDBC 驱动的数据库上。

1.2 运行工具

Hsqldb 提供的所有工具都能以 java class 归档文件(也就是 jar)的标准方式运行。在下面运行 AWT 版本的 DatabaseManager 的例子中，hsqldb.jar 位于相对于当前路径的../lib 下面。

```
java -cp ../lib/hsqldb.jar org.hsqldb.util.DatabaseManager
```

如果 hsqldb.jar 位于当前路径中，运行 DatabaseManager 的命令就应该改成：

```
java -cp hsqldb.jar org.hsqldb.util.DatabaseManager
```

Hsqldb 提供的主要的工具类

- org.hsqldb.util.DatabaseManager
- org.hsqldb.util.DatabaseManagerSwing
- org.hsqldb.util.Transfer
- org.hsqldb.util.QueryTool
- org.hsqldb.util.SqlTool

其中像 DatabaseManager 或 Sql Tool 这样的工具，可以或者只能用命令行参数来运行。你可以在命令行后面加上参数-?来查看这些工具可用的参数列表。Database Manager 和 Transfer Tool 具有图形用户接口，可以进行方便的交互式操作。

1.3 运行 Hsqldb

HSQLDB 可以采用几种不同的方式运行。一般可以分为 Server 模式和 In-Process(进程内模式,也称之为 Standalone(独立)模式）。对于每种模式，Jar 中都有相应的子程序来运行 HSQLDB。

每个 HSQLDB 数据库包含了 2 到 5 个命名相同但扩展名不同的文件,这些文件位于同一个目录下。例如，名为“test”的数据库包含了以下几个文件：

- test.properties
- test.script
- test.log
- test.data
- test.backup

properties 文件描述了数据库的基本配置。 script 文件记录了表和其它数据库对象的定义，此外还有 non-cached（无缓冲）表的数据。log 文件记录了数据库最近所做的改动。data 文件包含了 cached（缓冲）表的数据，而 backup 文件是将最后一次已知的连续状态的 data 文件以 zip 方式压缩备份。所有这些文件都是必不可少的，千万不可擅自删除。但如果你的数据库没有缓冲表(cached table),test.data 和 test.backup 文件就不会存在。此外，HSQLDB 数据库可以链接到磁盘上任何地方任何格式化的文本文件，比如 CSV 列表。

“test”数据库在运行时， “test.log”文件是用来记录数据的变化，在 HSQLDB 正常关闭的时候会被删除。如果在非正常退出时，“test.log”将用来在下次启动 HSQLDB 时重复执行记录的更新操作. “test.lck”文件也可以用来记录数据库是否被打开，它在 HSQLDB 正常退出的时候会被删除。有些情况下，HSQLDB 会生成 test.data.old ”文件，但随后会被删除掉。

注意

当 HSQLDB 引擎关闭数据库的时候，它会产生扩展名为.new 的临时文件，然后会把这些文件重命名为上面所列出的文件的名称。

1.4 Server 模式

Server 模式提供了最好的访问性能（accessibility）。数据库引擎在 JVM 里运行，监听来自同一台计算机上或网络中其他计算机程序的连接请求。几个不同的程序可以连接到服务器并且查询和更新信息。应用程序（客户端）通过 HSQLDB 的 JDBC Driver 连接服务器。在大多数的服务器模式中，服务器可以在运行的时候被指定为最多 10 个数据库服务。

服务器模式可以预置属性和命令行参数，详见“高级主题”一章。根据客户端和服务器之间通信协议的不同，Server 模式可以分为以下三种：

1.4.1 Hsqldb Server

这种方式是最佳的也是最快的。采用了 HSQLDB 专有的通信协议。启动服务器的命令和上面所述的运行工具命令类似。下例中启动服务器的命令启动了带有一个（默认为一个数据库）数据库的服务器，这个数据库是一个名为"mydb.*"文件。

```
java -cp ../lib/hsqldb.jar org.hsqldb.Server -database.0 file:mydb -dbname.0 xdb
```

命令行参数-?可以用来查看该命令的可用参数列表。

1.4.2 Hsqldb Web Server

这种模式用在只能通过 HTTP 访问启用数据库服务的计算机的情况下。采用 Web Server 模式的唯一原因是客户端或服务端的防火墙对网络连接访问采取了限制。其他情况下，这种模式不推荐被使用。HSQLDB Web Server 是可以允许 JDBC 客户端通过 HTTP 连接的特殊 Web Server。从 1.7.2 版本开始，这种模式也支持事务处理。

运行 web server 的时候，只要将上例命令行中 Server 的主类（main class）替换为下面的类：

```
org.hsqldb.WebServer
```

命令行参数-?可以用来查看该命令的可用参数列表。

1.4.3 Hsqldb Servlet

这种模式和 Web Server 一样都采用 HTTP 协议，当如 Tomcat 或 Resin 等单独的 servlet 引擎(或应用服务器)提供数据库访问时,可以使用这种模式。但是 Servlet 模式不能脱离 servlet 引擎独立启动。为了提供数据库的连接，应该将 HSQLDB.jar 中的 Servlet 类安装在应用服务器上（放置在 library 目录下）。可以使用应用服务器的的属性来指定数据库，可以参考 org.hsqldb.Servlet.java 的源文件查看详细信息。

Web Server 和 Servlet 模式在客户端都只能通过 JDBC 驱动来访问。两者都不提供数据库的 web 前台终端(front end)。Servlet 模式只能启动一个单独的数据库。

请注意在应用服务器中使用数据库引擎通常不采用这种模式。

连接以 Server 模式运行的数据库

当 HSQLDB Server 运行时，客户端程序就可以通过 hsqldb.jar 中带有的 HSQLDB JDBC Driver 连接数据库。如何连接服务器的详细说明可以参见 jdbcConnection 的 Java 文档 [../src/org/hsqldb/jdbc/jdbcConnection.html]（位于 HSQLDB 发布包/doc/src 目录中）。下面是一个简单的例子，它采用 hsqldb 协议连接到本机的默认的 9001 端口。

Example 1.1. Java code to connect to the local Server above

```
try {
        Class.forName("org.hsqldb.jdbcDriver" );
    } catch (Exception e) {
        System.out.println("ERROR: failed to load HSQLDB JDBC driver.");
        e.printStackTrace();
        return;
    }

Connection c = DriverManager.getConnection("jdbc:hsqldb:hsql://localhost/xdb", "sa", "");
```

有些情况下，你也可以使用下面的代码来获取驱动(driver).

```
    Class.forName("org.hsqldb.jdbcDriver").newInstance();
```

可以注意到，在上面的链接中，没有提到数据库文件，因为这些在 Server 运行时，数据库文件就已经被指定为 dbname.0 的值了。对于每个 server 不止有一个数据库实例情况的连接 URL 也可以阅读“高级主题”一章。

安全因素的考虑

HSQLDB 以 Server 模式运行的时候，网络访问应该受到充分的保护。源 IP 地址可能会因为 TCP 过滤、防火墙程序或者独立的防火墙的使用，而受到限制。如果数据流要穿越一个不受保护的网络（比如 Internet 时），数据流应该被加密（比如采用 VPN，SSH 隧道或者 TLS）。只有安全的密码才可以使用——最重要的是，应该将为系统默认用户设置的空字符串密码修改为安全密码。如果你想公开自己的数据的话，那么完全开放的网络连接应该限制为只有通过只读的帐号来访问这些公开的数据。（比如，对于非机密数据或非特权用户可以考虑使用这种连接方式）。这些考虑因素也适用于采用 HTTP 协议运行 HSQLDB 服务器的情况下。

1.4.4 In-Process(Standalone)模式

这种模式下，数据库引擎作为应用程序的一部分在同一个 JVM 中运行。对于一些应用程序来说， 这种模式因为数据不用转换和通过网络的传送而使得速度更快一些。其主要的缺点就是默认不能从应用程序外连接到数据库。所以当应用程序正在运行的时候，你不能使用类似于 Database Manager 的外部工具来查看数据库的内容。在 1.8.0 版本中，你可以从同一个 JVM 的一个线程里面来运行一个服务器实例作为您的应用程序，从而可以提供外部连接来访问你的 In-Process 数据库。

在一个应用程序里，推荐使用 In-Process 模式的方式是：开发的时候为数据库使用一个 HSQLDB 服务器实例，然后在部属的时候切换到 In-Process 模式。

一个 In-Process 模式数据库是使用 JDBC，带有指定数据库文件路径的连接 URL 来启动的。例如，假如数据库名称为 testdb，它的数据库文件位于和相关运行应用程序命令相同的目录下，下面的代码可以用于数据库的连接：

```
Connection c = DriverManager.getConnection("jdbc:hsqldb:file:testdb", "sa", "");
```

数据库文件的路径格式在 Linux 主机和 Windows 主机上都被指定采用前斜线(”/”）。所以相对路径或相对于相同分区下相同目录路径的表达方式是一致的。例如，在 Linux 系统中你的数据库文件路径为/opt/db/testdb，你可以在 Windows 的 C:分区下，创建一个相同的目录结构，然后你可以在 Linux 和 Window 下采用相同的 URL：

```
Connection c = DriverManager.getConnection("jdbc:hsqldb:file:/opt/db/testdb", "sa", "");
```

使用相对路径的时候，这些路径表示的是相对于用于启动 JVM 的 shell 命令的路径。你可以参考 jdbcConnection 的 Java 文档[../src/org/hsqldb/jdbc/jdbcConnection.html]来获得更多详细信息。

1.4.5 Memory-Only 数据库

HSQLDB 还可以用这种方式运行——数据库不是持久化的而是全部在随机访问的内存中。因为没有任何信息写在磁盘上，所以这种模式应该用于应用数据的内部处理上，比如在 applet 或其他的一些特殊的应用程序中。这种模式通过 mem: protocol 的方式来指定：

```
Connection c = DriverManager.getConnection("jdbc:hsqldb:mem:aname", "sa", "");
```

你也可以在 server.properties 中指定相同的 URL 来运行一个 Memory-Only（仅处于内存中）服务器实例。这种用法不常用，尽限于在使用数据库服务器来交换两个客户端的数据或者是非持久化数据的特殊应用程序中使用。

1.5 一般操作

1.5.1 关闭数据库

以不同模式运行的所有数据库都可以通过以 SQL 语句发出的 SHUTDOWN 命令来关闭。从 1.7.2 版本以后，当最后一次的数据库的连接通过 JDBC 被显式关闭之后，in-process 模式的数据库将不再被关闭，除非使用 SHUTDOWN 命令。在 1.8.0 中，可以在第一次第一次打开数据库的连接中，指定 shutdown=true 这个连接属性，在最后一次链接关闭时，强制关闭数据库。
当 SHUTDOWN 命令发出，所有的活动事务（active transactions）都会被回滚（rolled back）。关闭数据库的一种特殊形式是通过 SHUTDOWN COMPACT 命令。该命令重写.data 文件，这个文件包含了存储在 CACHED 表中的信息，该命令对该文件的大小进行精简(compact)。这个命令应该被周期性执行，特别是在 cached 表中执行了许多插入，更新和删除操作的时

候。数据库结构的变换，比如删除、更改 CACHE 表或者索引，会产生大量未使用的文件空间，不过可以通过此命令进行空间的回收利用。

1.5.2 在一个 JVM 里使用多个数据库

在上述的例子中，每个服务器只有一个数据库工作以及仅能创建一个 In-Memory 数据库。不过，从 1.7.2 版本以后，HSQLDB 能够以多个服务器模式使用多个数据库，它允许对多个 In-Process 数据库和 memory-only 数据库进行并发访问。 这些性能在“高级主题”一章进行了叙述。

1.5.2 创建新数据库

当一个服务器实例启动或者建立一个 in-process 数据库连接时，如果指定的路径没有数据库存在，那么就会创建一个新的空数据库。

这个特性的副作用就是让那些新用户产生疑惑——在指定连接到已存在的数据库路径时，如果出现了什么错误的话，仍然会创建一个指向新数据库的连接。为了解决这个问题，你可以指定一个连接属性 ifexists=true 只允许和已存在的数据库建立连接而避免创建新的数据库，如果数据库不存在的话，getConnection()方法将会抛出异常。

1.6 使用数据库引擎

一旦以任何模式建立了数据库的连接，就可以使用 JDBC 方法和数据库交互。对于 jdbcConnection[../src/org/hsqldb/jdbc/jdbcConnection.html], ，jdbcDriver[../src/org/hsqldb/jdbcDriver.html], jdbcDatabaseMetadata[../src/org/hsqldb/jdbc/jdbcDatabaseMetaData.html],jdbcResultSet[../src/org/hsqldb/jdbc/jdbcResultSet.html], jdbcStatement[../src/org/hsqldb/jdbc/jdbcStatement.html], 和 jdbcPreparedStatement[../src/org/hsqldb/jdbc/jdbcPreparedStatement.html]的 JavaDoc 列出了所有支持的 JDBC 方法以及对 HSQLDB 的详细说明信息。JDBC 方法大体上可以分为：与连接相关的方法，元数据方法，数据库访问方法三种。数据库访问方法使用 SQL 命令来在数据库上执行操作，返回的结果既可以是 Java 原型（Primitive）数据类型，也可以是 java.sql.ResultSet 类的实例。

你可以使用 Database Manager 或者其他 Java 数据库访问工具来查看数据库的内容，并可以用 SQL 命令来更新数据库。这些程序在内部使用 JDBC 向数据库引擎提交你的命令，并用自然语言的格式来显示结果。

HSQLDB 中使用的 SQL 方言（dialect）与 SQL92 和 SQL200n 标准比较接近，到目前位置 HSQLDB 在小型数据库引擎中已经有可能这两个标准。“SQL 语法”一章列出了所有的 SQL 命令。

1.6.1 不同的表类型

HSQLDB 支持临时表和三种持久化表。

TEMP（临时）表是不写入磁盘，它仅仅只维持一个Connection对象的生命周期。每一个TEMP表仅对操作它的Connection来说是可见的，其他和数据库并发的连接只能访问自己的TEMP表备份。从1.8.0版本开始， TEMP表的定义遵循SQL标准的GLOBAL TEMPORARY类型。表的定义是持久的，但每个新的连接只能看到它自己的表，而这些表刚开始是空的。当连接提交以后，临时表里的内容默认会被清空。如果临时表的定义语句里边包括ON COMMIT PRESERVE ROWS,那么当连接提交发生时，临时表的内容将被保存。

三种类型的持久化表一次是 MEMORY（内存）表，CACHED（缓存）表和 TEXT（文本符）表。

Memory 表是使用 CREATE TABLE 命令的默认表类型。Memory 表数据全部驻留在内存中，但是对于表结构或内容的任何修改都被写入到<dbname>.script 文件中。script 文件在下次数据库打开的时候被 MEMORY 读取，里边的所有内容在 MEMORY 表中重新创建。所以跟 TEMP 表不同， MEMORY 表被默认为是持久的。

CACHED 表是在使用 CREATE CACHED TABLE 命令的时候生成的。它只有索引或部分数据是驻留在内存中的，所以可以允许生成大容量表而不用占用几百兆的内存。CACHED 表的另外一个优点，即使它存储了大量的数据，数据库引擎只需花费很短的时间就可以启动。它的不足是在速度上有所降低。如果你的数据集相对小的时候，尽量不要使用 CACHED 表。在小容量和大容量表共存的实际应用中，最好对小容量的表使用默认的 MEMORY 表。

TEXT 表是在 1.7.0 版本中开始支持的，它使用 CSV（逗号分割数值）或其他分隔符的文本文件作为数据源。你可以指定一个已有的 CSV 文件（比如其它数据库或程序导出的数据）作为 TEXT 表的数据源，你也可以指定一个空文件用数据库引擎来填充数据。TEXT 表的内存利用效率比较高，因为它只缓存部分文本数据和所有的索引。TEXT 表的数据源如果需要的话，可以重新分配到不同的文件。建立一个 TEXT 表所需的两个命令的详细资料请查看 TEXT 表一章。

在 Memory-Only 数据库里，MEMORY 表和 CACHED 表的声明都看视作为非持久化的内存表声明。这种模式中是不允许声明 TEXT 表的

1.6.2 约束和索引

HSQLDB 支持 PRIMARY KEY, NOT NULL, UNIQUE, CHECK 和 FOREIGN KEY（依次译作主键、非空、唯一性、检查、外键）约束。此外，它还支持唯一性索引或普通索引。这些支持相当广泛，覆盖了多字段约束和索引，再加上外键的级联更新和删除。

HSQLDB 在内部创建索引来支持主键约束、唯一性约束和外键约束：为每一个主键约束或唯一性约束创建一个唯一索引，为每个外键约束创建一个普通的索引。因为这些原因，你

不应该在受这些约束的同一字段集内创建重复的用户自定义索引。这将会导致不必要的内存和速度开支。你可以查看“SQL 问题”一章的讨论来获得更多的信息。

索引对于提高查询速度是至关重要的，当使用连接到多个表的查询时，对于每一个表的每一个连接字段必须有一个索引。当使用范围或等式从句（比如：SELECT ... WHERE acol >10 AND bcol = 0）时，需要为在条件中使用的 acol 列创建一个索引。但是索引在 ORDER BY 从句或某些 LIKE 条件中是没有任何作用的。

根据经验，HSQLDB 能够以超过每秒 100,000 行的能力来处理查询，如果需要的话，任何花费数秒以上的查询应该被检查，并且应该对表中相应的列添加索引。

1.6.3 SQL 支持

HSQLDB 支持的 SQL 语法主要是由 SQL 标准（92 和 200n）规定的。但 HSQLDB 也不是支持标准的所有特性，还有一些自己特有的扩展，在 1.8.0 版本中，数据库引擎比老版本能更好的遵循标准，其主要的改变有：

- 修正了在连接、唯一性约束或是在查询条件中的空列值的处理。
- 修正了采用连接和左外连接 select 的处理。
- 修正了包含在表达式或包含表达式参数的聚集函数的处理。

HSQLDB 所支持的命令都列举在“SQL 语法”一章。你可查看由 Bruce Momjian 写的 PostgreSQL: Introduction and Concepts 一章，它是一部不错的 SQL 基本指南，并且带有样例，你可以点击链接下载。书中所讲到的大多数 SQL 都可以用到 HSQLDB 中。但是在数据库引擎支持的关键字（OUTER, OID's 等）和使用不同的关键字上（IDENTITY/ SERIAL, TRIGGER, SEQUENCE 等）存在着一些差异。

1.6.4 JDBC 支持

在 1.7.2 版本中，对于 JDBC2 的支持已经取得了重大的进展，而且现在已经开始支持 JDBC3 的部分特征。相关的类都已经文档化了，你可以查看 org.hsqldb.jdbcXXXX 类的 JavaDoc[../src/index.html]获得帮助。

第 2 章 SQL 问题

2.1 本章目的

在本章中就在 HSQLDB 主页论坛或邮件列表中多次提出的问题进行解答，如果你打算在应用程序中使用 HSQLDB 的话，你应该阅读一下本章。

2.2 对 SQL 标准的支持

1.8.0 版本的 HSQLDB 支持 SQL92、99 和 2003 标准规定的 SQL 方言。这意味着 HSQLDB 中支持的标准特性（例如左外连接）的语法是由标准文本规定的。许多 SQL92、99 甚至更高级的特征在 HSQLDB 中得到了支持，并且对 SQL2003 标准的大多数以及一些可选的特性进行支持。然而，对于某些标准的特性没有支持，所以 HSQLDB 就没有做出支持各个级别所有的标准特性的声明。

“SQL 语法”一章列出了 HSQLDB 所支持的所有的关键字和语法。当书写有关 HSQLDB 或者转换现有的有关 HSQLDB 的 SQL DDL（数据定义语言）和 DML（数据操作语言）语句的时候，你应该查阅一下 HSQLDB 所支持的语法，并对 SQL 语句作出相应的修改。

SQL 标准中保留的关键字是不能作为表明或字段名使用的。例如，“POSITION”被作为与 Java 中的 String.indexOf()作用类似的函数加以保留。HSQLDB 目前并不限制使用它不支持其用法的关键字或用户能够区分清楚的关键字。例如，“BEGIN”是 HSQLDB 目前没有支持的关键字，所以你也可使用它作为表或者列的名称。不过你应该避免使用这些保留字，因为在 HSQLDB 以后的版本中有可能支持这些保留字，否则将拒绝含有这些保留字表定义或查询语句。全部 SQL 保留字列表请参看 org.hsqldb.Token 类。

HSQLDB 也支持一些 SQL 标准之外的关键字和表达式作为性能的增强。像 SELECT TOP 5 FROM .., SELECT LIMIT 0 10 FROM ... 或者 DROP TABLE mytable IF EXISTS 这样的表达式都是 HSQLDB 增强性能所支持。

所有被双引号标注的关键字可以被用做数据库对象。

2.3 约束和索引

2.3.1 主键约束

在 1.7.0 版本之前，一个 CONSTRAINT <name> PRIMARY KEY（名为 name 的主键约束）被在内部翻译成一个唯一的索引，另外，一个隐藏列被添加到具有额外唯一索引的表上。从 1.7.0 开始，单一列主键和多列主键(single-column and multi-column PRIMARY KEY)约束都得

到支持。它们由主键列指定的唯一索引支持，而没有额外的隐藏列来维护它们的索引。

2.3.2 唯一性约束

根据 SQL 标准，一个单一列上的唯一性约束表示不允许存在两个相同的值（空值出外），也就是说这样的列中可以一个或更多为空值（NULL）的行而不违反唯一性约束。

多个列(c1, c2, c3, ..)的唯一性约束表示这些列中的任何两个值的集合都不相等（除非至少其中有一个为空）。每一个单一列内部可以有重复的值。下面这里满足两列上的唯一性约束。

例 2.1. 满足 2 列唯一性约束的列值

1, 2
2, 1
2, 2
NULL, 1
NULL, 1
1, NULL
NULL, NULL
NULL, NULL

自从 1.7.2 版本以来，对于空值的唯一性约束和索引的处理已经向遵循 SQL 标准改变。任何唯一性约束列的值全为空的行，通常都可以被添加到表中，所以对于唯一性约束的列来说，如果其中的一个行的值为空，那么多个行就可以具有相同的值。

2.3.3 唯一性索引

在 1.8.0 中，用户定义的唯一性索引仍然可以进行声明，但是它已经不被赞成使用了，你应该使用一个唯一性约束来替代唯一性索引。

像在早期版本的 HSQLDB 中一样，名为<name>的唯一性约束通常在内部创建列的唯一索引，所以它实际上和已经废除的唯一性索引声明具有相同的效果。

2.3.4 外键

HSQLDB 从 1.7.0 版开始具有单一列外键和多列外键的特性。一个外键也可以被指定引用一个没有命名的目标列的目标表。在这种情况下，目标表的主键列被用作引用列。任何外键中的一对引用和被引用的列应该具有相同的数据库类型。当声明了一个外键时，主键表被引用的列必须具有一个唯一性约束（或主键），而在引用列里会自动创建一个非唯一性索引。例如：

CREATE TABLE child(c1 INTEGER, c2 VARCHAR, FOREIGN KEY (c1, c2) REFERENCES parent(p1, p2));

parent 表中的 (p1,p2) 列一定存在一个唯一性约束。child 表中的(c1,c2)列会自动生成一个非唯一性索引。p1 和 c1 列一定具有相同的类型（INTEGER），p2 和 c2 列一定具有相同的类型（VARCHAR）。

2.3.5 索引和查询速度

HSQLDB 没有使用索引来改善查询结果的分类，但是索引在提高查询速度上起着至关重要的作用。如果一个表没有使用索引，进行类似于 DELETE 这样的查询操作的时候，表中所有的行记录都要被检查一遍。WHERE 从句中的列中，如果有一个列建立索引的话，那么查询操作就可能直接从第一个候选行记录开始，从而减少了要检查的行记录的数量。

索引在多个表之间进行连接操作的时候显得更重要。在 SELECT ... FROM t1 JOIN t2 ON t1.c1 = t2.c2 执行的时候，对 t1 中的行一个接一个的进行操作，来查找 t2 中没有与之匹配的行。如果没有 t2.c2 没有任何索引的话，那么，对于 t1 的每一行来说，要必须检查 t2 的所有行。然而有一个索引的话，在很短的时间内就能找到匹配的行。如果，查询(query)在 t1 上还有一个条件（例如，SELECT ... FROM t1 JOIN t2 ON t1.c1 = t2.c2 WHERE t1.c3 = 4），那么在 t1.c3 上创建一个索引的话，将不需要一个接一个的检查所有的行，并且将每返回一行的查询时间降低到 1 个毫秒以下。所以，如果 t1 和 t2 各有 10000 行记录，那么没有索引的查询将会进行 100,000,000 次行检查。如果在 t2.c2 上创建一个索引的话，查询次数将会降低到 10000 次行检查和索引的查找。在 t2.c2 上有另外一个索引的话，只需要检查 4 行就可以得到第一个符合条件的结果行。

对于主键和唯一列，将对自动生成索引，否则，你要使用 CREATE INDEX 命令来定义索引。

注意：在 HSQLDB 中，多列（multiple columns）上的一个唯一性索引可以在内部被用作列表第一列上的非唯一性缩影。例如，CONSTRAINT name1 UNIQUE (c1, c2, c3);表示这是 CREATE INDEX name2 ON atable(c1)等价的表示方式，所以你只要列表第一列中一个的话，就不需要指定一个额外的索引。

在 1.8.0 版本中，一个多列的索引将会加速包含所有列的连接和值的查询。你不需要在这些列上声明任何附加的单独的索引，除非你仅对这些列的子集进行查询。例如，在三个列上拥有主键或唯一性约束或只是一个普通的索引的表的行记录，在三个列的值都在 WHERE 从句中指定了时候，查询就会比较高效。例如，SELECT ... FROM t1 WHERE t1.c1 = 4 AND t1.c2 = 6 AND t1.c3 = 8 将会使用 t1(c1,c2,c3)上的索引（如果存在的话）。

作为多键索引改进的结果，和以前相比，声明索引或约束的列的顺序带给查询速度的影响大大减少。如果包含多较多不同值的列首先出现的话，查询速度将会有一点提高。

一个多列索引不会加快仅对第二列或第三列查询的速度。第一列必须在 JOIN .. ON 或者 WHERE 条件中指出。

查询速度很大程序上取决于 JOIN .. ON 或者 FROM 从句中表的顺序。例如，下面的第二个查询在大的表中会更快一些（假如 TB.COL3 上有索引的话）。因为，如果下面的查询被用到第一表中（以及 TB.COL3 上有索引）的话，TB.COL3 可以被很快的查出来：

（TB 是一个很大的表，但符合 TB.COL3=4 条件的仅仅只有几行记录）
SELECT * FROM TA JOIN TB ON TA.COL1 = TB.COL2 AND TB.COL3 = 4;
SELECT * FROM TB JOIN TA ON TA.COL1 = TB.COL2 AND TB.COL3 = 4;

基本规则是将这样表放在首位：其中的一个列上具有一个缩小的条件(narrowing condition)。

1.7.3 版本的 HSQLDB 具有为视图或用在查询中的子选择(subselect)创建自动的，快速的(on-the-fly)索引的特征。当一个索引被连接到一个表或一个视图的时候，会被添加到一个不同于所连接的视图里。

2.3.6 Where 条件或者连接

使用 WHERE 条件连接表可能会降低执行速度。例如下面的查询通常会比较慢，甚至有索引的时候：

SELECT ... FROM TA, TB, TC WHERE TC.COL3 = TA.COL1 AND TC.COL3=TB.COL2 AND TC.COL4

上面的查询表示 TA.COL1 = TB.COL2 但是却没有显式地设置条件。如果 TA 和 TB 每个都含有 100 行的记录，那么即使在被连接的列上建有索引时，也有 10000 个组合 combinations）被连接到 TC 来满足这些列的条件。使用 JOIN 关键词，条件 TA.COL1 = TB.COL2 将必须被显示声明，它将会使 TA 和 TB 行在被 TC 连接前降低它们的结合（combinations）次数，这样将会使对大表查询的执行大大加快：

SELECT ... FROM TA JOIN TB ON TA.COL1 = TB.COL2 JOIN TC ON TB.COL2 = TC.COL3 WHERE TC.COL4 = 1

如果连接查询中表的顺序改变了的话，那么下面查询的执行速度会被大大加快，所以使用了 TC.COL1 = 1 以及将列的行的较小集合连接在一起：

SELECT ... FROM TC JOIN TB ON TC.COL3 = TB.COL2 JOIN TA ON TC.COL3 = TA.COL1 WHERE TC.COL4 = 1

在上述的例子中，数据库引擎将 TC.COL4 = 1 自动应用到 TC，仅仅连接了行的集合来使这个条件也满足其他的表。TC.COL4, TB.COL2 和 TA.COL1 上的索引如果存在的话将会被使用，进而加速查询。

2.3.7 子查询和连接

使用连接及为获得最大性能而设置表的顺序适用于所有的地方，例如，下面的第二个查询通常情况下比第一个更快，因为 TA.COL1 和 TB.COL3 有索引的缘故。

例 2.2. 查询比较

SELECT ... FROM TA WHERE TA.COL1 = (SELECT MAX(TB.COL2) FROM TB WHERE TB.COL3 = 4)

SELECT ... FROM (SELECT MAX(TB.COL2) C1 FROM TB WHERE TB.COL3 = 4) T2 JOIN TA ON TA.COL1 = T2.C1

第二个查询将 MAX(TB.COL2)转变成一个单行的表，但后将它连接到 TA。因为 TA.COL1 上存在索引，所以这个查询的速度会非常快。第一个查询将会测试 TA 中的每一行记录，重复的对 MAX(TB.COL2)进行估算。

2.4 类型和算术操作

HSQLDB 支持的所有类型的数据库表都可以被索引，也可以进行比较。所有的类型都可以用 CONVERT()库函数进行显式转换，但是在大多数情况下它们可以被自动的转换。不推荐在 LONGVARBINARY, LONGVARCHAR 和 OTHER 列使用索引,因为这些索引可能在将来的版本中是不允许的。

早期版本的 HSQLDB 在算术操作的处理上有些不足。例如，不可能把 10/2.5 插入到任何 DOUBLE 或者 DECIMAL 列。在 1.7.0 版本中，遵循下列规则的全操作(full operations)是可能的：

TINYINT, SMALLINT, INTEGER, BIGINT, NUMBER 和 DECIMAL（没有小数点）是 HSQLDB 支持的整值类型，它们在 Java 中被映射成 byte, short, int, long 和 BigDecimal。SQL 类型规定了每个类型的最大值和最小值。例如，TINYINT 的值范围是从-128 到 +127，虽然实际用于处理 TINYINT 类型的 Java 类型是 java.lang.Integer。

REAL, FLOAT, DOUBLE 在 Java 中都被映射成 double。

DECIMAL 和 NUMERIC 被映射成 java.math.BigDecimal，它们可以有很多的位数。

2.4.1 整型类型

TINYINT, SMALLINT, INTEGER, BIGINT, NUMBER 和 DECIMAL（不带小数点)在内部都是完全可以互换的，不存在数据窄化(narrowing)的情况。返回的操作结果依赖于不同的操作数类型，它可以是存在于 JDBC 结果集里面的任何相关的 Java 类型(Integer, Long or

BigDecimal)。只要返回值可以被结果类型所表示，ResultSet.getXXXX()方法就能够用来获取数据。这个类型是基于查询的，而不是实际返回的行。当向数据库的表中添加了更多数据之后返回多行记录的时候，返回一条行记录的相同查询的类型不发生变化。

如果 SELECT 语句涉及到一个简单的列或函数的话，那么返回类型就是所对应的列的类型或者函数的返回值类型。例如：

```
CREATE TABLE t(a INTEGER, b BIGINT); SELECT MAX(a), MAX(b) FROM t;
```

上面的依据将会返回一个结果集，其第一列的类型是 java.lang.Integer，第二列的类型是 java.lang.Long。然而，

```
SELECT MAX(a) + 0, MAX(b) + 0 FROM t;
```

将返回 java.lang.Long 和 BigDecimal 的值，它是作为所有返回值的统一类型的提升生成的。

在表达式中的中间整形数值的大小上，没有内建的限制。因此，你应该检查结果集列的类型，选择一个合适的 getXXXX()方法来获取该值。另外你也可以使用 getObject()方法，然后，将结果转换成 java.lang.Number，在结果上使用 intValue() 或 longValue()方法。

当表达式的结果存在数据库表的一个列中，它必须适合目标列，否则，会返回一个错误。例如，当计算 1234567890123456789012 / 12345687901234567890 时，结果可以被存储在任何类型的列中，甚至是个 TINY 列，因为它是一个很小的值。

2.4.2 其他数字类型

在 SQL 语句中，如果不是指数形式，带有小数点的数字都被当作 DECIMAL 处理。因此 0.2 被认为是 DECIMAL 类型的值，而 0.2E0 则被看作 DOUBLE 类型。

当对某个值使用 PreparedStatement.setDouble()或 setFloat()方法时，该值就被自动默认为 DOUBLE 类型处理。

当表达式的一部分是 REAL、FLOAT 或者 DOUBLE（所有同义的类型）类型时，那么结果类型则是 DOUBLE。

此外，当 DOUBLE 类型不存在时，如果表达式的一部分是 DECIMAL 或 NUMERIC 类型时，结果类型就是 DECIMAL，只要结果可以用这种数字类型表示，那么它可以从 ResultSet 中按照任何所需的数字类型来获取。这就意味着如果一个 DECIMAL 值的范围在 Double.MIN_VALUE- Double_MAX_VALUE 之间，那么它就可以被转换成 DOUBLE 类型。和整形数值相似，当表达式结果存入表的列中时，它的类型必须符合该列的类型，否则将有错误出现。

进行除法运算时，DOUBLE 和 DECIMAL 类型间的区别就显得很重要的。当操作数类型是 DECIMAL，结果的取值范围（小数点右边的位数）取的是两个操作数中较大的范围。如果是 DOUBLE 类型，那么范围则是运算的精确结果。例如，10.0/8.0（DECIMAL）等于 1.2，但是 10.0E0/8.0E0（DOUBLE）等于 1.25。如果不是除法运算，DECIMAL 类型能精确的描述算法；如果两个相乘，那么范围就是两个数范围之和。

REAL、FLOAT 和 DOUBLE 类型数值都被当作 java.long.Double 对象存入数据库中。同时也可以存储和支持一些特殊值如 NaN 和正负无穷大。这些值可以通过 JDBC PreparedStatement 方法提交给数据库，并且可以返回结果集对象。

2.4.3 二进制和 Boolean 类型

从 1.7.2 开始，BIT 通常也可以看作 BOOLEAN 的别名。BOOLEAN 列的主要描述是'true'或'false'，通过 JDBC 使用时，可以被看作 boolean 或者 String 类型。BOOLEAN 列的类型还可以使用任何数字类型的值进行初始化。这样的情况下，0 将被转化成 false，其他非 0 值将被转换成 true。

从 1.7.3 开始，BOOLEAN 类型遵循 SQL 标准，除了支持 TRUE 和 FALSE 类型以外，也提供了对 UNDEFINED 状态的支持。NULL 值被当作未定义类型处理。这一改进影响了那些包含 NOT IN 的查询。这样的查询语句，请参看测试文本文件 TestSelfNot.txt。

2.4.4 Java 对象的存储和操作

从 1.7.2 开始，对 Java 对象的存储和操作的支持有了很大的提高，通过使用各种变量的 PreparedStatement.setObject()方法，任一个序列化的 JAVA 对象都可以被直接的插入到 OTHER 类型的列中。

为了比较，在索引里边，任何两个 Java 对象都被看作是相等，除非他们中的一个是 NULL。在 OTHER 类型的列中，你不能搜索一个指定的对象或者进行连接操作。

请注意，HSQLDB 不是一个对象－关系数据库。Java 对象只是简单地存储在内部，除了将 OTHER 类型的列值赋为 NULL 或测试该列值是否为 NULL 之外，不应该对它们进行其他的操作。测试例如 WHERE object1=object2，或者 WHERE ojbect1=?并不能得到你预期的结果。任何非空的对象都能满足这样的测试。但是 WHERE object1 IS NOT NULL 就能很好的执行。

普通列的数据被分配给 Java 对象列（例如，在类似 UPDATE mytable SET objectcol = intcol WHERE…的 SQL 语句中，向列中填充一个整型或字符串）时，数据库并不会返回错误，但在将来，这样的做法时很不恰当的。因此请使用 OTHER 类型列来存贮对象而不是其他类型的数据。

2.4.5 类型的大小、精度和范围

在 1.7.2 版本以前，所有带有大小、精度或范围的限制的数据库表的列类型定义可有可无。

在 1.8.0 版本中，这样的限定必须和 SQL 标准一致。例如 INTEGER（8）这样的形式将不能被接受。除非设置了数据库的属性，要你这个限定仍然可以被忽略。SET PROPERTY ”sql.enforce_strict_size” TRUE 语句将为 CHARACTER 或 VARCHAR 列强制设置大小，并且当插入或更新一个 CHARACTER 列时填充字符串。精度和范围限制也被转化为 DECIMAL 和 NUMERIC 类型。TIMESTAMP 只能使用 0 或 6 这样的精度。

如你所料，将一个值转换成一个 CHARACTER 的限定类型将导致切割或填充。因此象 CAST(mycol AS VARCHAR(2))=’xy’这样的测试语句结果得到以 xy 开头的值。这和 SUBSTRING(mycol FROM 1 FOR 2)=’xy’是相同的。

2.5 序列和标识

SEQUENCE 关键字作为 SQL200n 标准语法的子集被引入到 1.7.2 版本中。相应的，IDENTITY 列的 SQL200n 语法也被引进。

2.5.1 标识自动增长列

每个表都可以有一个自动增长列，众所周知的就是 IDENTITY 列。一个 IDENTITY 列总是被当作表的主键处理（因此，多列主键不可能有一个 IDENTITY 列）。作为一个捷径，已经为 CREATE TABLE <tablename>(<colname>IDENTITY,…)提供了支持。

从1.7.2版本开始，SQL标准语法默认的支持指定初始值。支持的形式是： (<colname> INTEGER GENERATED BY DEFAULT AS IDENTITY(START WITH n, [INCREMENT BY m])PRIMARY KEY, ...).。同时也对BIGINT特性列提供了支持。因此，一个IDENTITY是一个由Sequence生成器生成默认值的INTEGER或者BIGINT列。

当使用INSERT INTO <tablename>…语句向表中添加一个新列时；你可以在IDENTITY列中使用NULL值，这样该列的值将会自动生成。IDENTITY（）函数返回通过连接插入到IDENTITY列中的最后一个值。使用CALL IDENTITY（）这个SQL语句取得这个值。如果想在子表中使用这个值，你可以使用语句INSERT INTO <childtable> VALUES(..., IDENTITY(),...);。在任何附加的更新或插入语句在数据库中执行前，两种类型的IDENTITY（）函数必须被调用。

接下来要使用的的IDENTITY值可以通过语句ALTER TABLE ALTER COLUMN <columnname> RESTART WITH <new value>;来设置。

2.5.2 序列

SQL200n语法和用法和现有的大多数据库引擎所支持的是有区别的。可以通过命令CREATE SEQUENCE创建Sequences，并且可以通过命令ALTER SEQUENCE在任何时候进行修改它们的当前值。一个sequence的下一个值是通过NEXT VALUE FOR <name>表达式取得。这个表达式可以用来插入或者更新表的行，你也可以在select语句中使用它。例如，如果想计算在一个Sequence序列中SELECT返回的行，可以使用：
例2.3 计算选择操作以后返回的行

```
SELECT NEXT VALUE FOR mysequence, col1, col2 FROM mytable WHERE ...
```

请注意，顺序的语义并不完全和SQL200n定义的相同。例如，如果你在相同的列插入操作中两次使用相同的序列，你将会得到两个不同的值，而不是标准所说的同一个值。

你可以通过查询SYSTEM SEQUENCES表得到下一个值，这个值将从任何一个已定义的序列中返回。SEQUENCE_NAME列保存的是名称，NEXT_VALUE列保存的是下一个被返回的值。

2.5.3 事务的问题

HSQLDB在READ_UNCOMMITTED级别上支持事务，就象0级别上的事务隔离。这就意味着在一个事务的生命周期内，其他数据库的连接可以看到这个事务对数据的改变。总的来说，对事务的支持还是不错的。象数据库突然关闭而事务被提交这样的BUG已经被修复了。然而，下面的问题将会在多个数据库连接使用事务时出现问题:

如果两个事务修改同一行，而两个事务提交时没有异常出现。这样的情况可以通过这种方式的设计来避免：应用程序数据的一致性不依赖一个事务对数据的独占性修改。你可以通过设置一个数据库属性，在这样的情况发生时会抛出一个异常：SET PROPERTY "sql.tx_no_multi_rewrite" TRUE

当一个ALTER TABLE…INSERT COLUMN或者DROP COLUMN命令导致数表的结构变化时，当前的会话被提交。如果由另一个连接启动的未提交的事务此时应在受影响的表中修改了数据，那么这个事务在ALTER TABLE命令后不可能被回滚。这一点同样也适用于ADD INDEX 或ADD CONSTRAINT命令。只有在其他连接没有使用事务时，才推荐使用ALTER等这样的命令。

在CHECKPOINT命令执行以后，未授权事务才能继续运行、提交或回滚。然而，如果数据库随后并没有使用SHUTDOWN命令正常的关闭的话，在数据库关闭时仍然是未提交状态的任何这样事务，将被在下次启动数据库时被部分提交（CHECKPOINT的状态）。不管程序中没有未授权的事务的情况，还是因为想这样的事务不可能持续过长时间，以至于非正常关闭可能影响数据的情况，都推荐使用CHECKPOINT。

2.5.4 新特性和变化

在最新的1.8.0版本中增加了许多更好的对SQL的支持。这些都在SQL语法这一章和文件在../changelog_1_8_0.txt，../changelog_1_7_2.txt中列出了。象POSITION(),SUBSTRING(), NULLIF(), COALESCE(), CASE ... WHEN .. ELSE, ANY, ALL这样的表达式和函数也在其中。另外一些改进虽然在文档中不是很明显，但能使数据库性能比以前版本有很大的改进。这其中最重要的就是能在连接（非空列将不再被连接）和外连接（现在的结果是正确的）中处理NULL值。你应该对使用新版本数据库引擎的应用程序进行测试，确定这些程序不再使用旧版本的那些错误的特性。在将来的版本中，数据库引擎仍将朝着完全支持SQL标准的方向改进，因此最好不要依赖当前版本中任何的非标准特性。

第 3 章 UNIX 下快速起步

如何快速配置并在 UNIX（包括 Mac OS X）下运行 HSQLDB

3.1 本章目的

本章主要介绍如何在 UNIX 操作系统上快速安装、运行和使用 HSQLDB。

HSQLDB 具有许多可选的特性，在这里，我只讲述它们中的一少部分。我打算讲述最基本的 UNIX 环境下 HSQLDB 安装：运行一个可以将数据持久保存的多用户数据库（我指的是数据是保存在磁盘上的，所以当数据库关闭和启动时，数据将持久保存着），我也会讲述如何将 HSQLDB 作为系统后台程序运行。

3.2 安装

打开 http://sourceforge.net/projects/hsqldb，页面，点击“**files**”链接，最新的 HSQLDB 发布包是 hsqldb 黑体字对应的栏目里面版本号最大的那个版本。看看是否有你所需要格式的最新版本的发布版。

如果你想要一个 rpm 发布包的话，你要按照上面所述方法查找 HSQLDB 的当前版本。点击 http://www.jpackage.org/站点上“freesection”中的 hsqldb 栏，看看是否有 HSQLDB 的当前版本。希望 JPckage folk 能够提供它们的 rpm 支持的 JVM 版本的文档（目前他们既没有在网站上，也没有在发布包里面提供相关的文档）（描述如何下载不在这篇文档的范畴之内）。

注意我下面的讨论中一些文件格式可能实际上并没有提供，如果这样的话，那就是我们还没有开始创建这些文件。

二进制文件的安装取决于你下载的发布包的格式。
从 a .pkg.Z 文件安装

> 这个版本仅供 Solaris 操作系统的超级用户使用。这是一个系统 V 压缩包，下载以后用 uncompress 或者 gunzip 解压该压缩包。
>
> uncompress filename.pkg.Z
>
> 你可以运行下面的指令来查看压缩包的信息
>
> pkginfo -l -d filename.pkg
>
> 以 root 身份运行 pkgadd 来安装

pkgadd -d filename.pkg

从 a .rpm 文件安装

这是 Linux 的 rpm 安装包，下载完 rpm 包之后，你可以运行下面的指令来查看它的信息：

rpm -qip /path/to/file.rpm

可以以 root 的身份来运行下面的指令来安装和更新 rpm 包

rpm -Uvh /path/to/file.rpm

Suse 的用户可能需要运行下面的指令来让 Yast 知道已经安装的 rpm 安装包:

yast2 -i /path/to/file.rpm.

从 a.zip 文件安装

将 zip 文件解压到 HSQLDB 新的主目录的上一级目录。你不需要创建 **HSQLDB_HOME** 文件夹，因为在解压过程中会以正确的名称创建它。

cd parent/of/new/hsqldb/home
unzip /path/to/file.zip

zip 中压缩的所有文件将会被解压缩到一个新的 hsqldb 文件夹下面。

现在看看自己所安装的文件. (对于 zip 文件安装，在 hsqldb 文件夹下面. 其他情况,请使用你的包管理系统提供的工具查看).其中 hsqldb 系统中最重要的文件就是 hsqldb.jar，它位于 lib 目录中。

重要提示：为了方便本章讲述，我把 **HSQLDB_HOME** 规定为：包含 hsqldb.jar 的 lib 目录的上一级目录。例如：假如你的 hsqldb.jar 的路径是/a/b/hsqldb/lib/hsqldb.jar,那么你的 **HSQLDB_HOME** 目录为/a/b/hsqldb。

如果你分发包的描述中提到了 hsqldb.jar 文件可以在你安装 Java 版本下工作，那么你已经完成了安装；否则，你还需要建立一个新的 hsqldb.jar 文件。

如果你按照上面的说明来作，却仍然不知道你的 hsqldb.jar 支持什么样的 Java 版本,你可以阅读一下 **HSQLDB_HOME**/readme.txt 和 **HSQLDB_HOME**/index.html。如果对你还是没有帮助的话，那么你可以尝试运行你的 hsqldb.jar，看看它能否正常工作，要么就建立你自己的 hsqldb.jar。

如果你不需要创建一个新的 hsqldb.jar.而使用提供的 hsqldb.jar 的话，就可以跳到文档的下一个小节(section)。

Procedure 3.1. 建立 hsqldb.jar

1. 如果你还没有 Ant，那么先从 http://ant.apache.org 网站上下载一个最新的稳定版本。进入你要安装 Ant 的目录，然后采用下面的命令解压 Ant 的压缩包：

   unzip /path/to/file.zip

   或者

   tar -xzf /path/to/file.tar.gz

   或者

   bunzip2 -c /path/to/file.tar.bz2 | tar -xzf -

   ANT 将会被安装在名为 apache-ant- 加上版本号的新的子目录中。解压完之后，你可以为该文件夹重命名。

2. 将系统变量 JAVA_HOME 设置为你的 Java JRE 或者 SDK 根目录，如下所示：

   export JAVA_HOME; JAVA_HOME=/usr/java/j2sdk1.4.0

   JAVA_HOME 的位置完全取决于你的 UNIX 的种类。Sun 的 Java rpm 分发包通常会安装到/usr/java/something. Sun 的系统 V 的 Java 分发包（包括来自 Solaris 的分发包）通常安装到/usr/something，同时会生成一个从/usr/java 到默认版本的链接（所以对于 Solaris，通常应该将 JAVA_HOME 设置为/usr/java）

3. 移除文件 **HSQLDB_HOME**/lib/hsqldb.jar，如果存在的话。

4. 进入到 **HSQLDB_HOME**/build 目录，确保 Ant 主目录下面的 bin 目录在你的搜索路径中，运行下面的命令：

   ant hsqldb

   这将会生成一个新的 **HSQLDB_HOME**/lib/hsqldb.jar.

如果你要使用默认的设置，生成 hsqldb.jar 之外的其他东西，可以查阅附录中的“生成 HSQLDB”中的内容。

3.3 建立一个 HSQLDB 持久数据库实例和一个 HSQLDB Server

如果你是使用系统相关的安装包(OS-specific package)安装的 HSQLDB，那么你可能已经具有了数据库实例和经过预配置的 Server。查看一下你的安装包是否包含一个名为 server.properties 的文件(使用安装包管理工具)。假如你已经具有数据库实例和 Server，为了理解你的建立过程，我建议在你没事 (poke around) 的时候还是阅读一下本节内容。

1. 选择一个 Unix 用户来运行数据库应用服务器。如果这个数据库是用于多用户，或者是一个产品系统（production system）（或者模拟一个产品系统），你应该专门

指定一个 Unix 用户用于运行 HSQLDB 数据库应用服务器，在我的例子中，我使用了名为 hsqldb 的用户。在这一章中，我把这个用户称为 **HSQLDB_OWNER**,因为这个用户将拥有数据库实例文件和进程。

如果帐号不存在的话，就创建一个。在所有的系统 V Unix 和大多数的类 UNIX 上（包括 Linux），你可以运行下列的指令（以 root 的身份）：

```
useradd -m -c 'HSQLDB Database Owner' -s /bin/ksh -m hsqldb
```

(BSD 族的用户可以使用一个类似的 pw useradd hsqldb...命令).

2. 成为 **HSQLDB_OWNER**. 把样例文件 **HSQLDB_HOME**/src/org/hsqldb/sample/sample-server.properties 拷贝到 **HSQLDB_OWNER** 的主目录下，重命名为 server.properties.

```
# Hsqldb Server cfg file.
# See the Advanced Topics chapter of the Hsqldb User Guide.

server.database.0 file:db0/db0

# I suggest that, for every file: database you define, you add the
# connection property "ifexists=true" after the database instance
# is created (which happens simply by starting the Server one time).
# Just append ";ifexists=true" to the file: URL, like so:
# server.database.0 file:db0/db0;ifexists=true
```

因为第一个数据库(server.database.0)的值是以 file:开始的，那么数据库实例将会被持久化到位于指定的目录下，并以指定名字开头命名的一组文件。设置数据库文件的路径到你想要的任何路径(相对路径是相对于包含 properties 文件的目录而言的)。你可以阅读“高级话题”一章的如何指定其他不同类型的数据库实例，以及如何设置监听端口和许多其他的事情。

3. 将环境变量 CLASSPATH 的值设置并输出为 **HSQLDB_HOME**（同上面所描述的）加上"/lib/hsqldb.jar"，例如：

```
export CLASSPATH; CLASSPATH=/path/to/hsqldb/lib/hsqldb.jar
```

在 **HSQLDB_OWNER** 的主目录里，运行

```
nohup java org.hsqldb.Server &
```

这将会在后台启动 Server 进程，以及将会产生新的数据库实例“db0”。当你看到包含"HSQLDB server... is online"的信息时，nohup 只是确保在退出当前的 shell 时，hsqldb 的命令没有退出（如果这是你期望的结果，那就忽略这句话）。

3.4 访问你的数据库

将文件 **HSQLDB_HOME**/src/org/hsqldb/sample/sqltool.rc 拷贝到 **HSQLDB_OWNER** 的主目录下. 用 chmod 命令授权 **HSQLDB_OWNER** 可以对这个文件进行读写操作。

```
# $Id: sqltool.rc,v 1.22 2007/08/09 03:22:21 unsaved Exp $

# This is a sample RC configuration file used by SqlTool, DatabaseManager,
# and any other program that uses the org.hsqldb.util.RCData class.
# You can run SqlTool right now by copying this file to your home directory
# and running
# java -jar /path/to/hsqldb.jar mem
# This will access the first urlid definition below in order to use a
# personal Memory-Only database.
# "url" values may, of course, contain JDBC connection properties,
# delimited with semicolons.

# If you have the least concerns about security, then secure access to
# your RC file.
# See the documentation for SqlTool for various ways to use this file.

# A personal Memory-Only (non-persistent) database.
urlid mem
url jdbc:hsqldb:mem:memdbid
username sa
password

# A personal, local, persistent database.
urlid personal
url jdbc:hsqldb:file:${user.home}/db/personal;shutdown=true
username sa
password
# When connecting directly to a file database like this, you should
# use the shutdown connection property like this to shut down the DB
# properly when you exit the JVM.

# This is for a hsqldb Server running with default settings on your local
# computer (and for which you have not changed the password for "sa").
urlid localhost-sa
url jdbc:hsqldb:hsql://localhost
username sa
password
```

```
# Template for a urlid for an Oracle database.
# You will need to put the oracle.jdbc.OracleDriver class into your
# classpath.
# In the great majority of cases, you want to use the file classes12.zip
# (which you can get from the directory $ORACLE_HOME/jdbc/lib of any
# Oracle installation compatible with your server).
# Since you need to add to the classpath, you can't invoke SqlTool with
# the jar switch, like "java -jar .../hsqldb.jar..." or
# "java -jar .../hsqlsqltool.jar...".
# Put both the HSQLDB jar and classes12.zip in your classpath (and export!)
# and run something like "java org.hsqldb.util.SqlTool...".

#urlid cardiff2
#url jdbc:oracle:thin:@aegir.admc.com:1522:TRAFFIC_SID
#username blaine
#password secretpassword
#driver oracle.jdbc.OracleDriver

# Template for a TLS-encrypted HSQLDB Server.
# Remember that the hostname in hsqls (and https) JDBC URLs must match
# the CN of the server certificate (the port and instance alias that follows
# are not part of the certificate at all).
# You only need to set "truststore" if the server cert is not approved
# by your system default truststore (which a commercial certificate
# probably would be).

#urlid tls
#url jdbc:hsqldb:hsqls://db.admc.com:9001/lm2
#username blaine
#password asecret
#truststore /home/blaine/ca/db/db-trust.store

# Template for a Postgresql database
#urlid blainedb
#url jdbc:postgresql://idun.africawork.org/blainedb
#username blaine
#password losung1
#driver org.postgresql.Driver

# Template for a MySQL database. MySQL has poor JDBC support.
#urlid mysql-testdb
#url jdbc:mysql://hostname:3306/dbname
#username root
#username blaine
```

```
#password hiddenpwd
#driver com.mysql.jdbc.Driver

# Note that "databases" in SQL Server and Sybase are traditionally used
# for the same purpose as "schemas" with more SQL-compliant databases.

# Template for a Microsoft SQL Server database
#urlid msprojsvr
#url jdbc:microsoft:sqlserver://hostname;DatabaseName=DbName;SelectMethod=Cursor
# The SelectMethod setting is required to do more than one thing on a JDBC
# session (I guess Microsoft thought nobody would really use Java for
# anything other than a "hello world" program).
# This is for Microsoft's SQL Server 2000 driver (requires mssqlserver.jar
# and msutil.jar).
#driver com.microsoft.jdbc.sqlserver.SQLServerDriver
#username myuser
#password hiddenpwd

# Template for a Sybase database
#urlid sybase
#url jdbc:sybase:Tds:hostname:4100/dbname
#username blaine
#password hiddenpwd
# This is for the jConnect driver (requires jconn3.jar).
#driver com.sybase.jdbc3.jdbc.SybDriver

# Template for Embedded Derby / Java DB.
#urlid derby1
#url jdbc:derby:path/to/derby/directory;create=true
#username ${user.name}
#password any_noauthbydefault
#driver org.apache.derby.jdbc.EmbeddedDriver
# The embedded Derby driver requires derby.jar.
# There'a also the org.apache.derby.jdbc.ClientDriver driver with URL
# like jdbc:derby://<server>[:<port>]/databaseName, which requires
# derbyclient.jar.
# You can use \= to commit, since the Derby team decided (why???)
# not to implement the SQL standard statement "commit"!!
# Note that SqlTool can not shut down an embedded Derby database properly,
# since that requires an additional SQL connection just for that purpose.
# However, I've never lost data by not shutting it down properly.
# Other than not supporting this quirk of Derby, SqlTool is miles ahead of ij.
```

我们将使用配置文件中的“localhost-sa”样例 urlid 定义。对于这个 urlid，其 JDBC 的 URL

是 jdbc:hsqldb:hsql://localhost，也就是运行在本机默认端口的 HSQLDB Server 的默认数据库实例的 URL。你可以在“高级主题”一章中，阅读有关连接其他数据库实例以及其他 Server 的 URL 的内容。

运行 SqlTool.

```
java -jar path/to/hsqldb.jar localhost-sa
```

此时，如果你得到一个提示的话，那意味着大功告成。如果安全是你关心的根本问题的话，你应该改变数据库中的特权密码，使用命令 SET PASSWORD 来改变系统管理员(SA)的密码：

```
set password "newpassword";
```

当完成操作时，使用命令“\q”退出。

如果你更改了 SA 的密码，你应该相应地修改 sqltool.rc 文件中的密码。

你也当然可以使用任何 JDBC 客户端程序访问数据库。请参考附录中的“第一个 JDBC 客户端范例”。你需要修改你的 classpath，让它包含 hsqldb.jar 和客户端程序的 class(es)。你也可以数用其它的 HSQLDB 客户端程序，比如：org.hsqldb.util.DatabasManagerSwing，它是一个与 SqlTool 有相同用途的图形化的客户端。

你可以使用任何普通的 UNIX 帐户来运行 JDBC 客户端（包括 SqlTool）只要该帐户具有读取文件 hsqldb.jar 和 sqltool.rc 的权限。关于将 sqltool.rc 放置在何处，如何执行 sql 文件，SqlTool 还有哪些特性，请查阅“SqlTool”一章。

3.5 创建其他的帐号

以 SA 或者其他管理员用户连接到数据库，运行“CREATE USER”为你的数据库实例创建一个新的帐号。HSQLDB 帐号是数据库实例相关，而不是数据库 Server 相关的。

对于当前的 HSQLDB 版本来说，只有具有 DBA 身份的用户才能都创建或者拥有数据库对象。DBA 成员具有任何操作的权限，而非 DBA 成员可能被授予了一些权限，但却不能创建或者拥有数据库对象。（不久以后，非 DBA 成员也可以能够创建数据库，前提是在目标 schema（方案）中设定了这样做的权限）。当你首次创建一个 hsqldb 数据库时，它仅仅只有一个数据库用户——SA(一个一个空字符串做密码的 DBA 帐号)。你应该设置一个密码（如前面文章中描述的方法）。你可随意创建多个其它的用户。为了将一个用户的身份设置为 DBA，你可以在“CREATE USER”中使用“ADMIN”选项，或者在创建帐号后再授予这个帐号 DBA 的角色。

如果你创建了一个用户却没有使用 ADMIN 标签（或者没有授予该用户 DBA 的角色），那么这个用户虽然可以读取数据词典表 (data dictionary tables)的内容，但是不能创建或拥有它的数据库对象。他将只有伪用户(pseudo-user)PUBLIC 所拥有的权利。为了给他更多的权限，即使是读取数据库对象的权利，对特定的数据库对象来说，你可以对这个帐号授予(GRANT)许可，授予角色（它包含了一组的权限），或者授予 DBA 角色自己。

因为仅拥有数据库帐号的人才可以对数据库进行相应的操作，所以允许其他的数据库用户可以查看数据库表中的数据通常是有意义的。为了优化性能，减少诊断，最小化管理，通常将多数据库用户需要访问的任一数据库对象向 PUBLIC（公众）授予 SELECT 的权限是最佳的选择（除非你要对你的数据库中的数据采取保密措施）。

3.6 关闭数据库

当你完成数据库实例的操作时，应当进行一次彻底的数据库关闭。你需要以 SA 或者其他具有管理员身份的用户连接上数据库。你可以使用 SqlTool，使用下面的命令：

```
java -jar path/to/hsqldb.jar --sql shutdown localhost-sa
```

你不用担心停止数据库 Server 的事情，因为它已经在所有运行的数据库实例关闭时自动关闭了。

3.7 在系统后台运行 HSQLDB

当然你可以通过 System V UniX 的 inittab（初始表）来运行 HSQLDB，但是通常的做法是通过初始化脚本来启动，这样会更快捷和易于管理。这一小节将会讲述如何建立和使用 UNIX 的初始化(init)脚本。不过这个初始化脚本只供 root 用户使用。(但这并不意味着 Hsqldb Server 是以 root 用户来启动的——通常并不是这样的）。

Init script（初始化脚本）的主要用途是用来启动在 server.properties 中配置的数据库实例的 HSQLDB Server，关闭所有的数据库实例，以及可能在 init script 配置文件中列出的所有 urlid（也可能没有列出）。这些 urlid 都必须在 sqltool.rc 文件中有记录。由于防火墙问题的缘故，如果你要运行一个 WebServer 来代替独立 Server 的话，应该确保已经建立了一个带有 webserver.properties 的健壮的 webserver，，并且调整了 sqltool.rc 中的 URL，以及在配置文件中设置了 TARGET_CLASS（目标类）。

你建立完 init script 之后，root 用户可以随时使用脚本启动和停止 HSQLDB（不仅仅在系统启动或关闭的时候）。

3.7.1 可移植的 HSQLDB 初始化脚本

设计初始化脚本的基本准则就是可移植性。它并不采用新型的的 Linux 和 HPUX 打印彩色的启动/关闭信息的方式，它也不保留子系统状态文件或使用许多 UNIX 系统提供的启动/关闭函数，因为这些特征都不是可移植的。

这一个脚本消除了这种局限性，在我测试过的 UNIX 变体上都能很好的工作，当然你也可以通过简单的修改使它在其它的 UNIX 系统上运行。当你的脚本不能与系统规定的守护进程管理器紧密结合时，你得有个经过很好的测试和运行的脚本，它能给你带来好的有价值的反馈。

3.7.2 初始化脚本配置步骤

这里我们采取的策略是，先获取用来运行独立的 Server 或者 WebServer 的 init script，至于运行何种 Server 由 TARGET_CLASS 指定。做完这些以后，你可以自定义 JVM，来在 JVM 里运行其它的 Server，运行你自己的应用程序（嵌入的）或者甚至用你自己的重载类来重载 HSQLDB 的行为。

1. 首先将初始化脚本 hsqldb 从 **HSQLDB_HOME**/bin 拷贝到你所用的 UNIX 操作系统放置初始化脚本的地方，通常的位置是/etc/init.d　or /etc/rc.d/init.d（在 System V 风格的 UNIX 上），/usr/local/etc/rc.d（在 BSD 风格的 UNIX 上）以及 /Library/StartupItems/hsqldb（在 OS X 上，不过你需要自己创建文件夹）。
2. 查看一下初始化脚本顶部的注释，它列出了不同 UNIX 平台上配置文件推荐使用的路径。和你的 UNIX 平台 CFGFILE 的值是多少。你需要拷贝样例配置文件 **HSQLDB_HOME**/src/org/hsqldb/sample/sample-hsqldb.cfg 到注释中列出的其中一个路径（根据自己的操作系统选择），按照配置文件里面的说明进行编辑。

```
# $Id: sample-hsqldb.cfg,v 1.16 2005/07/24 18:33:13 unsaved Exp $

# Sample configuration file for HSQLDB database server.
# See the "UNIX Quick Start" chapter of the Hsqldb User Guide.

# N.b.!!!! You must place this in the right location for your type of
# UNIX. See the init script "hsqldb" to see where this must be placed
# and what it should be renamed to.

# This file is "sourced" by a Bourne shell, so use Bourne shell syntax.

# This file WILL NOT WORK until you set (at least) the non-commented
# variables to the appropriate values for your system.
# Life will be easier if you avoid all filepaths with spaces or any
# other funny characters. Don't ask for support if you ignore this
# advice.

# Thanks to Meikel Bisping for his contributions. – Blaine

JAVA_EXECUTABLE=/usr/bin/java

# Unless you copied a hsqldb.jar file from another system, this
# typically resides at $HSQLDB_HOME/lib/hsqldb.jar, where
# $HSQLDB_HOME is your HSQLDB software base directory.
HSQLDB_JAR_PATH=/opt/hsqldb/lib/hsqldb.jar

# Where the file "server.properties" resides.
SERVER_HOME=/opt/hsqldb/data
```

```
# What UNIX user the server will run as.
# (The shutdown client is always run as root or the invoker of the init
# script).Runs as root by default, but you should take the time to set
# database file ownerships to another user and set that user name here.
HSQLDB_OWNER=hsqldb

# The HSQLDB jar file specified in HSQLDB_JAR_PATH above will
# automatically be in the class path. This arg specifies additional
# classpath elements.To embed your own application, add your jar file(s)
# or class base directories here, and add your main class to the
# INVOC_ADDL_ARGS setting below.
#SERVER_ADDL_CLASSPATH=/usr/local/dist/currencybank.jar

# We require all Server/WebServer instances to be accessible within
# $MAX_START_SECS from when the Server/WebServer is started.
# Defaults to 60.
# Raise this is you are running lots of DB instances or have a slow
# server.
#MAX_START_SECS=200

# Time to allow for JVM to die after all HSQLDB instances stopped.
# Defaults to 1.
#MAX_TERMINATE_SECS=0

# These are "urlid" values from a SqlTool authentication file
# ** IN ADDITION TO THOSE IN YOUR server.properties OR
# webserver.properties **file. All server.urlid.X values from your
# properties file will automatically be started/stopped/tested.
# $SHUTDOWN_URLIDS is for additional urlids which will stopped.
# (Therefore, most users will not set this at all).Separate multiple
# values with white space. NO OTHER SPECIAL CHARACTERS! Make sure to
# quote the entire value if it contains white space separator(s).
# Defaults to none (i.e., only urlids set in properties file will be
# stopped).
#SHUTDOWN_URLIDS='sa mygms'

# SqlTool authentication file used only for shutdown.
# The default value will be sqltool.rc in root's home directory, since
# it is root who runs the init script.
# (See the SqlTool chapter of the HSQLDB User Guide if you don't
# understand this).
#AUTH_FILE=/home/blaine/sqltool.rc
```

```
# Set this to either 'WebServer' or 'Server'. Defaults to Server.
# The JVM that is started can invoke many classes (see the following
# item about that), but this is the Server that is used (1) to check,
# status (2) to shut down the JVM, (3) to get urlids for #1 from the
# server's server/webserver.properties file.
#TARGET_CLASS=WebServer
# Note that you don't specify the org.hsqldb package, since you have
# no choice in the matter (you can only run org.hsqldb.Server or
# org.hsqldb.WebServer). If you specify additional classes with
# INVOC_ADDL_ARGS (described next), you do need to specify the
# full class name with package name.

# This is where you specify exactly what your HSQLDB JVM will run.
# The class org.hsqldb.util.MainInvoker will run the TARGET_CLASS
# specified above with any arguments supplied here + any other classes
# and arguments. Every additional class (in addition to the
# TARGET_CLASS)must be preceded with an empty string, so that
# MainInvoker will know you are giving a class name. MainInvoker will
# invoke the normal static main(String[]) method of each such class.
# By default, MainInvoker will just run TARGET_CLASS with no args.
# Example that runs just the TARGET_CLASS with the specified arguments:
#INVOC_ADDL_ARGS='-silent false'
# Example that runs the TARGET_CLASS plus a WebServer:
#INVOC_ADDL_ARGS="" org.hsqldb.WebServer'
# Note the empty string preceding the class name.
# Example that starts TARGET_CLASS with an argument + a WebServer +
# your own application with its args (i.e., the HSQLDB Servers are
# "embedded" in your application). (Set SERVER_ADDL_CLASSPATH too).:
#INVOC_ADDL_ARGS='-silent false "" org.hsqldb.WebServer "" #com.acme.Stone --env #
Example to run a non-TLS server in same JVM
# with a TLS server. In this case, TARGET_CLASS is Server which will
# run in TLS mode by virtue of setting TLS_KEYSTORE and TLS_PASSWORD
# h above. The "additional" Server ere overrides the 'tls' and 'port' #settings:
#INVOC_ADDL_ARGS="" org.hsqldb.Server -port 9002 -tls false"
# Note that you use nested quotes to group arguments and to specify
# the empty-string delimiter.

# For TLS encryption for your Server, set these two variables.
# N.b.: If you set these, then make this file unreadable to non-root
# users!!!! See the TLS chapter of the HSQLDB User Guide, paying
# attention to the security warning(s).
# If you are running with a private server cert, then you will also
# need to set "truststore" in the your SqlTool config file (location
# is set by the AUTH_FILE variable in this file, or it must be at the
```

```
# default location for HSQLDB_OWNER).
#TLS_KEYSTORE=/path/to/jks/server.store
#TLS_PASSWORD=password

# Any JVM args for the invocation of the JDBC client used to verify
# DB instances and to shut them down (SqlToolSprayer).
# This example specifies the location of a private trust store for TLS
# encryption.
# For multiple args, put quotes around entire value.
#CLIENT_JVMARGS=-Djavax.net.debug=ssl

# Any JVM args for the server.
# For multiple args, put quotes around entire value.
#SERVER_JVMARGS=-Xmx512m
```

3. 要么把 **HSQLDB_OWNER** 的 sqltool.rc 文件拷贝到 root 的主目录下面，要么将 AUTH_FILE 的值设置为 **HSQLDB_OWNER** 的 sqltool.rc 文件的绝对路径。这个文件由 root 用户直接阅读，即使你是以一个非 root 用户来运行 hsqldb 的（通过在配置文件中设置 HSQLDB_OWNER）。如果你复制了那个文件，还要确保使用 chmod 来限制新的 sqltool.rc 文件的访问权限。(现在初始化脚本完成这个操作).
4. 编辑 server.properties 文件。对于你定义的每一个 server.database.X，为这个数据库实例的管理员用户的 urlid 设置一个名为 server.urlid.X 的属性。

   Example 3.1. server.properties fragment

   ```
   server.database.0=file://home/hsqldb/data/db1
   server.urlid.0=localhostdb1
   ```

   警告：

   确保为每一个数据库实例添加了 urlid。如果你没有的话，随后的初始化脚本将不可能得到数据库的信息而导致无法访问，以至于给出错误的诊断信息。

   在这个例子中，你需要在 sqltool.rc 文件中将 urlid 定义为 localhostdb1

   Example 3.2. example sqltool.rc stanza

   ```
   urlid localhostdb1
   url jdbc:hsqldb:hsql://localhost
   username sa
   password secret
   ```

5. **验证初始化脚本的工作**

   只需要以 root 运行下列命令来查看你可以使用的参数

/path/to/hsqldb

注意你可以在任何时候运行下列的命令来查看 HSQLDBServer 是否正在运行：

/path/to/hsqldb status

使用每一个可能的参数来重新运行来运行脚本，你可以测试脚本是否真的没问题。如果出现什么问题的话，你可以查看“解决脚本配置过程中的问题”这一小节。

6. **让你的系统在启动和关闭的时候运行初始化脚本。**

   如果你正在使用的 UNIX 系统(像 BSD 系列或 Gentoo)有/etc/rc.conf 或者 /etc/rc.conf.local 这些文件的话，你必须在每个文件中将"hsqldb_enable" 设置成 “YES”。 (运行下列命令看看你的系统是否有这些文件： cd /etc; ls rc.conf rc.conf.local)。对于使用 System V 风格初始化的 UNIX 用户，你必须建立硬链接或者软链接，不管是采用手动方式或者使用类似于 chkconfig 或 insserv 的管理工具，还是像 run level 编辑器这样的 GUI 工具.

   下面这段只对使用 Mac OS X 的用户适用。如果你是按照上面的说明做的话，你的初始化脚本应该在这个地方： /Library/StartupItems/hsqldb/hsqldb.现在将这个文件 StartupParameters.plist 从你的 HSQLDB 版本的 src/org.hsqldb/sample 文件夹复制到与初始化脚本相同的目录下. 只要这两个文件都在/Library/StartupItems/hsqldb 目录下, 你的初始化脚本现在就被激活了 (为了可移植性考虑，初始化脚本不会当作 /etc/hostconfig 里的一项设置检查). 你可以运行下列命令将初始化脚本作为启动项来运行：

   SystemStarter {start|stop|restart} Hsqldb

   Hsqldb是一项服务的名称.看SystemStarter的man 页面，为了禁止初始化脚本，你可以清空/Library/StartupItems/hsqldb文件夹下面的文件.很难相信,Mac的用户告诉我说在系统关闭的期间，启动项根本就不运行。因此，如果你不想你的数据损坏的话，必须在关闭Mac系统之前，运行命令"SystemStarter stop Hsqldb"。

按照配置文件中的范例，添加其他的类到 server 的 JVM classpath 中，并在 JVM 中运行这些类。(可以参看 SERVER_ADDL_CLASSPATH 和 INVOC_ADDL_ARGS 项).

3.7.3 解决脚本配置过程中的问题

用 ps 命令看一下系统中是否有包含 hsqldb 的进程，尝试从任意客户端连接一下数据库。如果初始化脚本成功启动了数据库，但是却错误的报告没有成功启动的话，你的问题可能与 urlid 或者 sqltool 设置有关。如果你的数据库确实没有启动，可以跳到下一段。首先，检查一下 server.properties 或 webserver.properties 中列出的 urlid 是否正确无误，然后，检查一下你是否是以 root 身份使用 SqlTool 来连接数据库实例（对于最后的这个测试，如果你在配置如见中设置了 AUTH_FILE 的话，你可以使用--rcfile 开关来试试）。

如果你的数据库确实没有启动，你检查你是否可以用切换(su)到数据库拥有者(Owner)的帐号来启动数据库。命令 su USERNAME -c ...在大多数的 UNIX 上不能运行，除非目标用户有真正的 login shell。因此，如果你想通过禁用这个用户的 login shell 来加强安全性的话，你就违反了这个启动脚本。如果还没有奏效的话，按照下面描述的方法调试一下脚本或者寻求帮助。

为了调试脚本，要以 verbose 模式来运行，这样可以看到运行过程中的详细信息（也可以手动运行可疑的步骤）。为了使用 verbose 模式运行初始化脚本的话(实际上，可以是任何 sh shell 脚本),使用-x 或-v 开关的 sh 命令,如下：

```
sh -x path/to/hsqldb start
```

如果你不明白-v 和-x 之间的区别的话，可以参看 sh 的 man page。

如果你还有什么问题寻求帮助的话，可以使用 HSQLDB lists/forums 或这给作者 blaine.simpson@admc.com 发送 email. 如果你给作者发 email 的话，确保包含了 hsqldb 初始化脚本的修订版本号（位于以"# $Id:"开始的那行文字的头部），和运行下列命令的输出：

```
sh -x path/to/hsqldb start > /tmp/hstart.log 2>&1
```

第 4 章 高级话题

4.1 本章目的

许多在论坛或邮件组中重复出现的问题将会在本文档中进行解答。如果你打算在应用程序中使用 HSQLDB 的话，那么你应该好好阅读一下本文章。本章的内容覆盖与系统相关的问题。与 SQL 相关的问题请参看“SQL 问题”一章。

4.2 数据库连接

访问HSQLDB数据库的正常方式是通过JDBC的Connection接口。介绍提供、访问数据库服务的不同方法可以在“SQL问题”一章中找到。在我们的JdbcConnection 的JavaDoc[../src/org/hsqldb/jdbc/jdbcConnection.html]中详细讲述了如何通过JDBC连接数据库，并提供了样例。

1.7.2版本的HSQLDB，提供了访问多个数据库的新功能，与此一同引进了区分不同类型的数据库连接的统一方法。通常的驱动标识为jdbc:hsqldb，后面紧跟协议标识(mem: file: res: hsql: http: hsqls: https:)，如果是server的话后面是主机标识和端口表示，最后是数据库标识。

表4.1. Hsqldb URL 组件

Driver and Protocol	**Host and Port**	**Database**
jdbc:hsqldb:mem:	not available	accounts
小写并且是单个词语的标识符在生成第一个连接时创建内存数据库。随后仍然使用相同的连接URL连接到已经存在的数据库。 旧的URL格式：jdbc:hsqldbL创建或连接到与新的URL格式jdbc:hsqldb:mem:相同的数据库。		
jdbc:hsqldb:file:	not available	mydb /opt/db/accounts C:/data/mydb
文件路径指定了数据库文件。在上面的例子中，第一个例子指的是用来运行应用程序的java命令的目录中的一系列mydb.*文件。第二个和第三个例子指的是在主机上的绝对路径。		
jdbc:hsqldb:res:	not available	/adirectory/dbname
数据库文件可以从一个指定的jar文件中加载，方式与Java程序访问资源文件相同。这些jar文件被看作为Java命令一部分。上面的/adirectory表示一个jar里面的目录。		
jdbc:hsqldb:hsql: jdbc:hsqldb:hsqls: jdbc:hsqldb:http: jdbc:hsqldb:https:	//localhost //192.0.0.10:9500 / /dbserver.somedomain.com	/an_alias /enrollments /quickdb

主机和端口用来指定server的IP地址或者主机名以及一个可选的端口号。要连接的数据库通过一个别名来指定。这个别名是在server.properties文件中定义的一个小写的字符串，它指的是server文件系统上的一个实际数据库或者server上临时的内存数据库。下面是server.properties或webserver.properties中的几行样例。它定义了上面列出的数据库别名以及客户端的访问方法，其中涉及到不同文件和内存数据库。
database.0=file:/opt/db/accounts
dbname.0=an_alias

database.1=file:/opt/db/mydb
dbname.1=enrollments

database.2=mem:adatabase
dbname.2=quickdb

Server URL的旧格式，例如jdbc:hsqldb:hsql//localhost，可以与URL新格式jdbc:hsqldb:hsql//localhost/，连接到同一个数据库。新格式URL中的数据库别名是一个长度为零的空字符串。在下面的例子中，/home/dbmaster/ 目录下的数据库文件lists.*与空别名相关联。
database.3=/home/dbmaster/lists
dbname.3=

4.2.1 连接属性

每个新的到数据库的JDBC连接都能够指定连接属性。属性user和password通常都是必须的。在1.8.0中，下列可选的属性也经常被用到。连接属性可以通过：

DriverManager.getConnection (String url, Properties info);
在建立连接时指定。也可以在整个连接URL后面附上属性来指定。

表4.2. 连接属性

get_column_name	true	结果集中的列名
这个属性用来与其他JDBC驱动的实现兼容。当属性的值为true(默认为true)时，ResultSet.getColumnName(int c) 将返回相关的列名。如果为false时，该方法将会返回与ResultSet.getColumnLabel(int column)相同的值。例子如下： jdbc:hsqldb:hsql://localhost/enrollments;get_column_name=false 当一个结果集用在用户自定义的存储过程里面时，该属性将总是使用默认值true。		
ifexists	false	仅当数据库存在的时候连接
这个只在mem:和file:数据库上起作用。如果值为true，当URL中指定的数据库不存在时，将不会创建一个新的数据库。当该值为false时，如果数据库不存在，那么一个新的mem:或file:数据库将会被创建。在URL不规范导致数据库不能被创建的时候，将该值设为tue比较有利于解决问题。 举例如下: jdbc:hsqldb:file:enrollments;ifexists=true		
shutdown	false	当最后一个连接关闭时，关闭数据库。

这个属性用来模仿1.7.1以及更老版本的行为. 当最后一个数据库连接关闭的时候，数据库将会自动关闭。这个属性只有在第一个数据库连接建立的时候生效。第一个连接指的是打开数据库的连接。这个属性在随后的并发连接上使用时没有任何效果。 这个命令有两个用途，一个是用于测试套件，当数据库的连接从JVM环境中创建，接着立即在另一个JVM 环境创建连接；另一个用于应用程序，不容易配置环境来关闭数据库。例如在用户报告的样例中的web应用服务器，最后一个连接的关闭和web app的关闭是同时的。

此外，当一个进程内数据库的连接创建了一个新的数据库或者打开了一个已有的数据库(也就是说，它是应用程序创建的第一个数据库连接)，所有的用户自定义的数据库属性都可以作为URL属性来制定。这个特点被用来指定属性来强制更严格遵守SQL标准， 或者在数据库文件创建之前改变cache_scale或相似的属性。但是，对于新的数据库来说，推荐使用SET PROPERTY命令来完成这些设置。

4.3 属性文件

HSQLDB 应该与一组配置文件来进行不同的配置，从1.7.0以来，属性命名已经简化，许多新的属性被引进。在所有的属性文件中，值都是大小写敏感的。所有的值除了文件名以外都是必须的，而且都是小写（例如：server.silent=FALSE 将不会生效,而server.silent=false可以正常工作）。属性文件和存储在文件中的设置如下所示：

表4.3. HSQLDB属性文件

File Name	Location	Function
server.properties	运行 Server 类的命令所在的目录	HSQLDB作为使用HSQL协议通信的数据库server运行所进行的配置。
webserver.properties	运行Web Server类的命令所在的目录	HSQLDB作为使用HTTP协议通信的数据库server运行所进行的配置。
<dbname>.properties	数据库的所有文件所在的目录	对每个特定的数据库所作的配置。

运行数据库server的属性文件不会自动生成，你应该创建包含自己属性的文件，在属性文件中对每个属性都包含了server.property和value对。

每个数据库的属性文件是由数据库引擎生成的，在关闭数据库，该文件可以自己编辑。在1.8.0版本中，大多数属性文件能够通过SQL命令来修改。

4.3.1 Server and Web Server 属性

在 server.properties and webserver.properties 文件中，支持的属性及其默认值如下所示：

表4.4. server和 webserver属性

Value	Default	Description
server.database.0	test	要使用的第一个数据库文件的路径和文件名
server.dbname.0	""	第一个数据库小写的server别名
server.urlid.0	NONE	UNIX初始化脚本使用的SqlTool urlid(如果在非UNIX平台上运行Server/WebServer或没有使用我们

		提供的UNIX初始化脚本，就不要使用这个属性)。
server.silent	true	不会再控制台上显示大量的信息。
server.trace	false	在控制台显示JDBC的跟踪（trace）信息。

在1.8.0版本中，每个server能够同时开启10个不同的数据库。server.database.0属性定义了文件名/路径，而server.dbname.0定义了client用来连接的数据库的别名。数字0对于第二个数据库将会递增为1，以此类推。server.database.{0-9}的值能够使用mem:, file: 或 res:前缀和在“数据库连接”中讨论过的属性。例如：database.0=mem:temp;sql.enforce_strict_size=true;

server.properties 文件中特有的属性如下：

表4.5. server 属性

Value	Default	Description
server.port	9001 (一般情况)或554 (如果使用 TLS 加密)	用于和客户端通信的TCP/IP端口。所有的数据库都是在相同的端口监听。
server.no_system_exit	true	当数据库关闭时，没有System.exit()调用。

webserver.properties 文件中特有的属性如下：

表4.6. webserver 属性

Value	Default	Description
server.port	80	用于和客户端通信的TCP/IP端口。
server.default_page	index.html	Webserver的默认页面。
server.root	./	Webserver中网页根目录的位置。
.<extension>	?	像.html=text/html等多个属性定义了webserver中静态页面的MIME类型。对于MIME类型列表可以参看Webserver.java的源文件。

上面所有属性都可以在启动 server 的命令行中，由省略掉 server.前缀后的属性来指定。

4.3.2 从应用程序中启动 Server

如果你要从应用程序中启动Server，你应该创建Server或Web Server的实例，然后用字符串的形式指定属性启动server，不要使用命令行或批处理文件。可以从org.hsqldb.test.TestBase的源文件中找到范例。

注意

关于升级：如果你已经有自定义的属性文件，应该把属性修改为新的命名格式。注意，server.database.n 和 server.dbname.n 属性末尾数字的使用。

4.3.3 个别数据库属性

每个数据库都有自己的<dbname>.properties 文件，还包含了<dbname>.script and <dbname>.data. 属性文件包含一些重要的 key/value 对。在版本 1.8.0 中，新的 SQL 命令允许大多数数据库属性使用下面的方式进行修改：

SET PROPERTY "property_name" property_value
通过 SET PROPERTY 可以修改的属性在下面的表中进行描述。其他的属性只在属性文件中进行描述，而且这些文件只能在数据库关闭后到重启之前通过编辑.properties 文件来修改。在数据库操作时，只有下面列出的用户自定义的属性可以修改，而改变其他的属性将会导致不可预知的故障。这些属性的大多数是在 1.7.0 版本作为新特性添加进来的。

表4.6. 数据库相关属性文件的属性

Value	Default	Description
readonly	false	整个数据库是只读的。
该值为true时，使用中的数据库不能修改。如果数据库要从CD上打开，这个属性应该设置为true。在修改这个设置之前，应该使用SHUTDOWN COMPACT命令关闭数据库，确保数据的一致性和简介。（仅用在属性文件中），但也可以用于一个连接属性以只读方式打开一个普通的数据库。		
hsqldb.files_readonly	false	数据库文件将不能被写入数据。
该值为true时，MEMORY表中的数据可以被修改，也可以添加新的MEMEORY表。但在数据库关闭时，这些变动将不会被保存到下来。当该值为true时，CACHE表和TEXT表将总是只读的。（只用于属性文件中）。		
hsqldb.cache_file_scale	1	设置大数据文件的限制，一旦设置的话，这个限制将会上升到最大的8GB。
可以通过这个属性设置到8来将.data文件的容量大小从2GB增加到8GB。将这个改变应用到现有的数据库时，应该先执行SHUTDOWN脚本，然后在重新打开数据库之前将下面的property=value行添加到.properties文件中： hsqldb.cache_file_scale=8 当数据库中没有CACHED表时（例如，在一个新的数据库中），这个属性可以使用SQL命令设置（与在属性文件中直接修改属性值的方式截然相反）。（可以设置属性）		
sql.enforce_size	false	字符串列的去空白和填充
这个属性不再支持，请使用sql.enforce_sctrict_size		
sql.enforce_strict_size	false	大小的强制检查和字符串列填充
遵循关于数据库大小的和数据类型精确度的SQL标准。当该值为true是，所有在INSERT INTO或UPDATE语句影响的行记录中的CHARACTER,VARCHAR, NUMERIC 和 DECIMAL 值需要重新检查SQL表定义中指定的大小。 如果值太大的话，将会抛出一个异常，比指定的大小要小的所有的 CHARACTER 值 将用空格来填充。为了指定数据值亚秒级的解析度，对于TIMESTAMP(0)和TIMESTAMP(6) 也是允许的。当该值为false（默认值）时，插入的字符串将不做任何修改进行存储。 （可以设置属性）		
sql.tx_no_multi_rewrite	false	事务管理
在读未提交的隔离模式下，一个事务可以在另一个未提交的事务插入或更新的行记录上进行写操作。如果该值设置为true，在这种场景下进行写操作时将会抛出异常。（可以设置属性）		
hsqldb.cache_scale	14	内存缓存指数
以3×（2^{value}）表示保持在内存中的cached表的最大的行数。默认的结果是对所有的cached表，任何时候可以在内存中保持的最高行记录数为3×16384。该属性的取值范围是8-18.（可以设置属性）。如果该属性的值可以通过SET PROPERTY 来设置的话，那么在下个数据库关闭或checkpoint，它将变得比较有效。		
hsqldb.log_size	200	执行检查点是的日志大小。
该属性的值是以兆字节(MB)计算的，它指的是.log文件在checkpoint发生之前所能达到的大小。一个checkpoint重写.script文件，而且清除.log文件。这个属性值可以通过SQL 命令SET		

LOGSIZE nnn来修改。		
runtime.gc_interval	0	强制垃圾收集
每当生成一些结果集行对象或缓存行对象时，使用这个属性来强制垃圾收集。该属性的默认值0指程序不会强制进行垃圾收集。当数据库引擎在一个独占的JVM内部作为server运行时，不应该设置这个值。当数据库用在一些JRE的应用程序的进程内时，这个属性比较有用。一些JRE在做任何自动内存收集之前，会增加内存堆的大小。这个设置将会预防内存对不必要的扩大。该属性的典型值可能在10,000到100,000之间。（仅用在属性文件中）		
hsqldb.nio_data_file	true	使用nio方法访问数据库文件
当HSQLDB使用Java 1.4或更高版本进行编译及运行时，将这个属性设置为false将会避免nio方法的使用，因为nio方法将会导致速度略微降低。如果数据文件大于256MB，在第一次打开时，不要使用nio方法。如果文件增长到大于需要分配给nio访问的可用内存时，这时候应该使用nio方法。（可以设置属性）。如果在定义CACHED表之前使用，它会应用到当前的session中，否则，它将会在SHUTDOWN和重启或CHECKPOINT时生效。		
hsqldb.default_table_type	memory	使用不带条件的CREATE TABLE 创建的表类型
CREATE TABLE 命令在默认会生成一个MEMORY表。将该属性的值设置为"cached"，将会默认生成一个cached表。像CREATE MEMORY TABLE 或CREATE CACHED TABLE这样带条件的命令不受这个属性的影响。（可以设置属性）		
hsqldb.applog	0	应用程序的日志级别
默认级别0表示不记录日志。级别1会将包括所有失败的持久化相关的事件记录到日志里。这些事件将会记录在以.app.log结尾的文件中。		
textdb.*	0	新TEXT表的默认属性
覆盖数据库引擎对新创建的text表默认值的属性。在text表中SET <tablename> SOURCE <source string>命令中的设置覆盖数据库引擎和数据库属性文件的默认值。单独的textdb.*属性列在“TEXT 表”中。（可以设置属性）		

当一个进程内数据库的连接创建了一个新的数据库或者打开一个已有的数据库时（也就是说，它是第一个应用程序创建的一个数据库连接），上面列出的所有用户自定义属性可以在 URL 属性中指定。

注意：

关于升级：从 1.7.0 开始,数据库文件的位置将不再被属性文件中定义的路径覆盖。所有属于数据库的文件位于相同的目录下，属性 sql.compare_in_locale=true 将不再支持。 如果该行在属性文件中存在的话，它将数据库切换到当前默认的 collation。参见 SET DATABASE COLLATION2 命令。当 HSQLDB 用在 OpenOffice.org 时，一些属性值会有不同的默认值。这些属性和值如下：

hsqldb.default_table_type=cached

hsqldb.cache_scale=13

hsqldb.log_size=10;

hsqldb.nio_data_file=false

sql.enforce_strict_size=true

4.4 配置数据库属性的 SQL 命令

有一些数据库属性是以 SET 开头的 SQL 命令来设置的。

表 4.8 SQL 命令属性

SET WRITE_DELAY {{TRUE | FALSE} | <seconds> | <milliseconds> MILLIS
默认值TRUE表示记录在日志中的数据库变动，每隔20秒会被同步到文件系统一次。FALSE表示记录日志和同步操作之间没有任何延迟。数字0也可以用来指定同步没有延迟时间。 使用这个命令的目的是万一整个系统发生崩溃，可以控制数据库的丢失量。延迟1秒意味着写到磁盘的数据至多在崩溃前最后一秒被丢掉。所有写入的数据在这之前已经被同步或者应该是可以恢复的。这个设置应该在运行数据库引擎所使用的硬件可靠行性，使用的磁盘系统类型，电源故障的可能性等基础上来指定。所存储的数据库的性质也应该被考虑。 一般来说，当系统非常可靠时，这个设置可以使用默认值。如果系统不是非常可靠，或者数据非常关键的话，这个延迟在1秒或2秒内已经足够了。仅仅在最糟糕的情况下或者大多数是关键数据时，设置延迟为0或指定为FALSE，不过这样将会把数据库引擎的速度降低为磁盘子系统执行文件同步操作的速度。10毫秒的值可以通过添加MILLIS到命令行中来指定。但是在实际情况中，延迟100毫秒将会提供六天崩溃一次的99.99999%的可靠性。
SET LOG_SIZE <numeric value>
数据库引擎将所有的数据库变化在其发生时记录到日志文件中。这个日志文件可能被同步到上面所讲的基于WRITE_DELAY属性的磁盘上。日志文件除非是异常关闭，否则将不会被重用。也就是说，数据库进程不是被SHUTDOWN命令终结的话，或者它被SHUTDOWN IMMEDIATELY命令终结。日志默认的最大体积是200MB。当日志达到最大体积时，将会执行CHECKPOINT操作。这个操作将会把其他数据库文件保存为持久态，并且删除旧的文件。值0表示.log文件没有大小限制。
SET CHECKPOINT DEFRAG <numeric value>
当CACHED表中的记录行被更新或者删除时，大多数情况下空间可以被重用。但是，一些未使用的空间将迟早会留在.data文件中，特别是当大的表被删除或者表的结构发生变化时。 一个CHECKPOINT操作通常不重新回收空的空间，但是CHECKPOINT DEFRAG会总是回收空的空间。这个属性决定了在正常的CHECKPOINT中，或日志的大小超过的它的上限时，是否由管理员进行初始化。Numeric value指的是在要强制进行DEFRAG操作的.data 文件中，有多少M字节是空的空间。 如果该值设置得较低，将会导致频繁的 DEFRAG 操作。值 0 则表示不会执行自动的 DEFRAG 操作。默认的值是 200MB 闲置空间。
SET REFERENTIAL INTEGRITY {TRUE | FALSE}
这个值默认为TRUE。如果批量数据需要加载到数据库中时，这个属性在批量加载过程中可被设置为FALSE。它允许以任意顺序加载相关的表。该属性在批量加载完成后应该被设置回TRUE。如果已经加载的数据没有保证遵循相关的完整性约束，那么SQL查询应该在加载后运行以辨别和修改任意不遵循标准的行记录。

第 5 章 部署问题

5.1 本章目的

许多在论坛或邮件组中重复出现的问题将会在本文档中进行解答。如果你打算在应用程序中使用 HSQLDB 的话，那么你应该好好阅读一下本文章。本章的内容覆盖与系统相关的问题。与 SQL 相关的问题请参看“SQL 问题”一章。

5.2 操作模式和表

HSQLDB 有许多操作模式和特性，这可以允许它可以使用在各种不同的场景下。不通应用程序的内存使用率、速度和访问能力都受到 HSQLDB 部署的影响。

5.2.1 操作模式

是以独立服务器进程(separate server process)还是以进程内数据库(in-process database)模式运行 HSQLDB，都应该基于下面的 3 点：

- 当 HSQLDB 在一台单独的机器上以服务器模式运行时，它是不受应用程序运行的主机的硬件故障的影响的。
- 当 HSQLDB 在和应用程序相同的机器上以服务器模式运行时，它是不受应用程序崩溃和内存泄漏的影响。
- 由于传输每个 JDBC 访问的数据流带来的开销，使得 Server 模式连接比 in-process 模式的连接要慢。

5.2.2 表

TEXT 表是为其数据是可交换格式（例如，CSV）的特殊应用程序设计的。TEXT 表不适合数据的常规存储。

MEMORY 表和 CACHED 表通常用于数据存储。这两中表有以下的区别：

- 当数据库启动时，所有 MEMORY 表的数据从.script 文件中读出来，存储在内存中。相比而言，CACHED 表中的数据直到访问该表时才会读到内存中，此外每个 CACHED 表只有部分数据保留在内存中，允许表中存储比保留在内存的数据中更多的数据。
- 当数据库正常关闭时，MEMORY 表中的所有数据将会写到磁盘上。相比而言，发生改变的 CACHED 表中的数据才会写道磁盘上，还有就是所有 CACHED 表中的数据将会压缩备份。
- 所有 CACHED 表中数据缓存的大小和容量都是可以配置的。这使得 CACHED 表中的所有数据被缓存到内存中还是有可能的。这种情况，访问速度还可以，但和 MEMORY 表相比稍稍慢了一点。

- 对于通常的应用，少量数据推荐使用 MEMORY 表，大量数据集合推荐使用 CACHED 表。对于速度要求较高并且有较大的内存空间的特殊应用，MEMORY 表也是可以适用于大量数据的。

5.2.3 大对象

JDBC Clobs 由 LONGVARCHAR 类型的列支持，JDBC Blobs 由 LONGVARBINARY 类型的列支持。当大对象(LONGVARCHAR, LONGVARBINARY, OBJECT)存储在包含普通字段的表中时，最好用两个表来存储。第一个表包含普通的字段，第二个表包含大对象和 id 字段。使用这种方法有两个好处。（a）第一个表通常是以 MEMORY 表形式创建的，只有第二个表是一个 CACHED 表；（b）大对象在查询过程中可以根据它们的表示单独获取，而不用加载到内存中来查找行记录。下面是两个表和一个 select 查询的例子，它将两种类型的表分离开:

CREATE MEMORY TABLE MAINTABLE(MAINID INTEGER, ...);

CREATE CACHED TABLE LOBTABLE(LOBID INTEGER, LOBDATA LONGVARBINARY);

SELECT * FROM (SELECT * FROM MAINTABLE <join any other table> WHERE <various conditions apply>) JOIN LOBTABLE ON MAINID=LOBID;

括号内 select 查询查找符合条件的记录，不用关联到 LOBTABLE，当找到这些记录后，才从 LOBTABLE 中获取大对象数据。

5.2.4 部署的环境

用于存储 HSQLDB 数据库的文件都位于相同的目录下。新的文件总是被数据库引擎创建，而后又删除。应该知道下面两个基本的原则：

- 在数据库文件存储的文件夹中，运行 HSQLDB 的 Java 进程必须具有所有的权限。其中包括创建和删除的权限。
- 文件系统必须具有足够的空余空间，用于生成永久性文件和临时文件。.log 文件默认的最大体积为 200M。.data 文件的体积可以增至 8GB。.backup 文件的体积可以达到.data 文件的 50%。在 SHUTDOWN COMPACT(关闭数据库的一种方式)时生成的临时文件在体积上与.data 文件相同。

5.3 内存和磁盘使用

程序使用的内存可以被看作两个不同的池：数据库表数据使用的内存，建立结果集和其它内部操作使用的内存。此外，当使用事务处理时，内存用来保存回滚操作需要的信息。

自从版本 1.7.1 以来，内存使用和前几个版本比较而言，已经减小了很多。一个 MEMORY

表的内存使用是所每一行数据库记录的内存使用之和。每一个 MEMORY 表行记录是一个具有 2 个 int 或引用变量的 Java 对象，它包含了一个该行记录中字段所表示的对象数组。每一个字段都是一个如 Integer，Long，String 等的对象。此外，表中每个索引都给该行记录加上一个节点对象。每个节点对象有 6 个整形或引用变量。所以，只有一个 INTEGER 列的数据库表的每行记录将会有 4 个对象，总共有 10 个变量，其中每个变量 4 个字节——目前每个行记录要占据 80 个字节。在这之外，数据库中每个额外的列添加至少几个字节到每个行记录的上。

一个结果集行记录的内存使用更少的变量且没有索引节点，但仍然使用了很多内存。结果集中所有的行记录都是在内存中建立的，所以数据量很大的结果集是不可能的。在 server 模式的数据库中，一旦数据库 server 返回了结果集，结果集内存将会从 server 中释放出来。在进程内数据库中，当应用程序释放 java.sql.ResultSet 对象时，数据库就会释放内存。Server 模式需要额外的内存来返回结果集，因为它们将所有的结果集转化到一个字节序列中，然后传送到客户端。

当在 CACHED 表中执行 UPDATE 和 DELETE 语句时，所有有变化的行记录集，包括因为执行 UPDATE 操作改变的数据，将会在操作过程中驻留在内存中。这意味着对涉及大量数据的 CACHED 表行记录集，不可能执行 delete 或者 update 操作。对于较小的结果集是可以使用这些操作的。

当使用 SET AUTOCOMMIT OFF 开启事务支持时，所有 insert，delete 或 update 操作列表存储在内存中，因此，当进行 ROLLBACK 事务时，这些操作能够被撤销。跨越几百个数据改变的事务将会花费很多的内存直到下一个 COMMIT 或 ROLLBACK 清除了操作列表。

大多数 JVM 执行时的内存分配会一直到最大值（通常默认为 64MB），当大量内存表被使用时，或当 CACHED 表中行记录的平均大小大于几百个字节时，这个内存大小通常是不够的。内存分配的最大值可以在用来运行 HSQLDB 的命令行中进行设置。例如，用版本 1.3.0 的 Sun JVM 加上-Xmx 参数可以将最大值增加到 256MB。

1.8.0 的 HSQLDB 为存储在数据中的 Integer 或 String 等这些不可变对象使用了快速缓存。在大多数情况下，因为只有最经常使用的对象作为少量备份保存在内存中，所以这将会更进一步减少内存使用。

5.3.1 缓冲内存分配

在 CACHED 表中，数据存储在磁盘上，至多只有最大数量的行记录一直驻留在内存中。默认的值等于 3*16384 行。可以设置 hsqldb.cache_scale 的数据库属性来改变这个大小。因为任意 CACHED 表中的任意随机行记录的子集可以被保留在内存中，所以 cached 行记录需要的内存大小可以达到包含最大字段数据的行的数量之和。例如，如果有一个 100,000 行的表包含 40,000 行，每行有 1000 个字节数据的行记录和 60,000 行每行 100 个自己的行记录，那么表的缓存可以增至包含几乎 50000 个行记录，包含了所有 40000 个较大的行记录。

还有一个数据库的属性 hsqldb.cache_size_scale 可以和 hsqldb.cache_scale 属性联合起来使

用。这个属性给缓存的行记录总的大小加上了一个限制。当两个属性都使用默认值时，行记录其总大小的限制大约时 50MB。（这是行记录和索引的二进制映像的大小，因为数据用 Java objects 来表示，所以它能被转换成实际 cached 使用内存的 2 到 4 倍还要多）

如果内存有限制的话，hsqldb.cache_scale 或 hsqldb.cache_size_scale 属性的值可以减少一些。在上面的例子中，如果 hsqldb.cache_size_scale 的值从 10 降低到 8，那么总的二进制的极限将会从 50MB 降低到 12.5MB。这将会允许缓存行记录的数量达到 50000 个小的行记录，但只能有 12500 个稍大的行记录。

5.4 管理数据库连接

在所有的运行模式中（不管是 Server 模式还是 in-process 模式）数据库引擎都支持多连接。进程内（独立）模式支持来自同一个 Java 虚拟机内部的客户端连接，而服务器模式支持通过网络来自不同的客户端连接。

连接池软件可以用来连接数据库，但这通常不是必须的。在其它的数据库引擎中，使用连接池的原因不适用于 HSQLDB

- 允许在一个很耗时的查询在后台执行的时候，执行一个新的查询。在 HSQLDB1.8.0 版本是不可能的。当它执行一个查询时就会处于阻塞状态，只有完成后，才会处理下一个查询。这个功能正在开发中，将会在将来的版本中引进来。
- 考虑性能原因，限制数据库的最大并发连接数。在 HSQLDB 中，这个仅当你的应用程序设计成为每一个小任务打开和关闭连接的方式时，才比较有用。
- 在一个多线程应用中控制事务处理。这个在 HSQLDB 中也是比较有用的。例如，在一个 web 应用中，一个事务可能涉及一些处于查询或跨越网页的用户行为之间的一些进程。一个单独的连接应该被用于每个 HTTP session，这样当完成事务或回滚时，工作可以被提交。虽然这种用法不能被用在其它大多数的数据库引擎上，但是 HSQLDB 式完全可以处理超过 100 个并发的 HTTP session 作为一个单独的 JDBC 连接。

像记录用户登陆和登出的动作的应用程序，不会同时即是多线程，又是事务性的应用，不需要一个以上的连接。一个连接状态可以一直保持，仅当因为网络问题关闭时会重新打开。

当使用 1.7.2 版本之前的 HSQLDB 的进程内数据库时，应用程序必须至少保持一个数据库连接处于打开状态，否则，数据将会关闭，进一步尝试创建连接可能失败。1.7.2 版本以后，已经不需要这么做了。数据库不会自动关闭一个通过建立连接而保持打开状态的进程内数据库。关闭数据库时可以使用一个显式的 SHUTDOWN 命令，带或不带参数都可以。在 1.8.0 版本中，一个连接属性可用来恢复原来的行为。

当使用一个 server 数据库时（某种程度上是一个进程内数据库），必须特别小心避免频繁的创建和关闭 JDBC。当数据库负荷比较大时，频繁的连接和关闭数据库会导致不成功的连接尝试。

5.5 升级数据库

任何 1.8.0 以前的 HSQLDB 的发布版本都必须升级到最新的 1.8.0 版，包括使用 1.8.0 的 RC 版本创建的数据库。“使用脚本命令升级”一小节下面的说明适用于任何情况。

一旦数据库升级到 1.8.0，将不能再使用 Hypersonic 或以前版本的 HSQLDB。

在升级过程中可能存在这一些潜在的规则问题，这些应该通过编辑.script 文件来处理：

- 版本 1.8.0 不支持重复的索引名称，这在 1.7.2 版本之前是允许的。
- 版本 1.8.0 不支持表中重复的列名称，这在 1.7.0 之前是允许的
- 版本 1.8.0 不能像 1.7.2 版本之前，对于外键不能创建相同类型的索引
- 版本 1.8.0 不支持没有双引号的 SQL 标识符的表或列名称.

5.5.1 使用脚本命令升级

从 1.7.2 或 1.7.3 升级到 1.8.0，只需要简单地在老版本上执行 SET SCRIPTFORMAT TEXT 和 SHUTDOWN SCRIPT 命令，然后用新版本的引擎打开。这时升级就完成了。

从更老版本(1.7.1 及其之前版本)升级不包括 CACHED 表的数据库文件时，只需要简单地关闭旧版本，用新版本打开即可。如果.script 文件中出现什么问题地话，编辑.script 文件之后再试一次。

从更老版本(1.7.1 之前的版本)的包含 CACHED 表的数据库文件中升级的时候，使用下面的 SCRIPT 步骤。在所有版本的 HSQLDB 和 1.43 的 Hypersonic 中，使用”SCRIPT filename” 命令(作为一个 SQL 查询使用)可以让你把数据库中的所有记录，包括数据库对象定义和数据，保存到你选择的文件中。你可以使用老版本的数据库引擎导出一个脚本文件，在 1.8.0 中作为数据库打开脚本。

Procedure 5.1. 使用脚本升级步骤

1. 在老版本的数据库管理器(DatabaseManager)中打开原始的数据库。
2. 执行 SCRIPT 命令，例如，SCRIPT 'newversion.script'用来创建一个包含了数据库拷贝的.script 文件
3. 在 newversion 的例子中，在不同的文件夹下使用 1.8.0 版本的 DatabaseManager 来创建一个新的数据库。
4. SHUTDOWN 数据库
5. 复制第二步中的 newversion.script 文件覆盖 4 中数据库产生同名文件
6. 试着用 DatabaseManager 打开新的数据库。
7. 如果数据中出现什么矛盾的话，就会在控制台中报告脚本的行数，同时打开进程也被中止了。再次尝试打开之前，编辑和校正 newversion.script 文件中的错误。使用下一小节中的指导内容(手动改动.script 文件)。使用一个能够处理大文件的程序编辑器，同时对于文字中较长的行不要换行。

5.5.2 手动改动.script 文件

在 1.8.0 中，所有的 ALTER TABLE 命令对于改变数据结构和它们的名字都是可行的。但是，如果，旧的数据库由于数据冲突或者索引或列名的使用和 1.8.0 不兼容而不能被打开的话，可以手动编辑 SCRIPT 文件。

只要它们不影响现存数据的完整性，下列改动是可以使用的：

- 表、列和索引的名称可以改变。
- CREATE UNIQUE INDEX ... 到 CREATE INDEX ... 反之亦然。

一个唯一的索引常常可以被转化成一个普通的索引，只有当每行记录中列表数据是唯一的时候，一个不唯一的索引才能够被转化成一个唯一的索引。

- NOT NULL

一个非空的约束常常可以去除，只有在列中的表数据没有 null 值的情况，才添加 NOT NULL 约束。

- PRIMARY KEY

可以去除或添加一个主键约束。但当该列有一个外键引用的话，该主键不能被去除。

- COLUMN TYPES

对列类型进行一些改动是可以的。例如，一个 INTEGER 列可以改成 BIGINT，或 DATE，TIME 以及 TIMESTAMP 列可以改成 VARCHAR 列。

完成改动之后，保存修改的*.script 文件，然后你可以正常打开数据库了。

5.6 备份数据库

每一个数据库的数据由同一个文件夹中 5 个以上的文件组成。其后缀分别为*.properties, *.script, *.data, *.backup and *.log(以*.lck 结尾的文件用来控制数据库的访问，所以不应该备份)。这些文件应该备份在一起。这些文件可以在数据库引擎运行的时候进行备份，但是，千万要小心，在备份过程中，不能发生 CHECKPOINT 或 SHUTDOWN 操作。在 CHECKPOINT 之后立即进行备份时非常有效的。*.data 文件可以从备份文件中排除掉。在这种情况下，当恢复数据时，需要一个虚拟的*.data 文件（它可以为空，0 个字节）。如果数据备份被还原，数据库引擎将会扩展*.backup 文件来替换这个虚拟的文件。如果*.data 文件没有被备份，在还原之前，*.properties 文件可能修改以保证它包含了 modified=yes，而不是还原前的 modified=no。如果数据库备份是在一个 CHECKPOINT 之后进行的话，那么*.log 文件也可以被排除掉，这样就把重要的文件减少到*.properties, *.script 和 *.backup。可以使用例如把文件都压缩在一个归档文件中的正常的备份方法。

第 6 章 TEXT 表

Text 表作为 HSQLDB 将来的标准特性

6.1 前言

第 7 章 TLS

第 8 章 SqlTool

第 9 章 SQL 语法

附录 A 建立 HSQLDB

附录 B 第一个 JDBC 客户端例子

附录 C HSQLDB 数据库文件及其恢复

附录 D 在 OpenOffice.org 1.1.x 中运行 HSQLDB

附录 E　HSQLDB 测试工具

附录 F　数据库管理器

附录 G　传输工具