Choosing Light Weight Java Database

sayrohan.blogspot.in

Choosing Light Weight Java Database

sayrohan.blogspot.in

~~Recently, I had to work on a project which had need for the construction of both Server and Client type of web application. The constraint on the Client was to be extremely light weight, run on commodity machine and be quick enough to respond to the request from the Server. Client has to be written in Java and must have both light weight Java Application Server and Database~~.

~~So, I surveyed different light weight JAVA databases and short-listed most popular ones, Namely:~~
~~1. HSQLDB~~
~~2. H2~~
~~3. ObjectDB~~
4. Derby
5. Neo4j
6. OrientDB
The idea was to choose the datab**ase which can be 'embedded' in the** application and do not need any different requirement for the deployment. As part of running the Client, Database should be created on run time, dump the data in the file, shouldn't leave much of trace on the client machine, and do the required

job.

~~So, I did some google and listed pros and cons of each database and then decided to pick one of them. As part of google, I gave more weightage to the responses as mentioned in the stackoverflow site and read about opinion of different people who implemented each of these databases in practice.~~

1. HSQLDB

Pros:

1. ~~Embedded and server modes are available~~
~~2. It includes tools such as a minimal web server, command line and GUI management tools (can be run as applets), and a number of demonstration examples. It can run on Java runtimes from version 1.1 upwards, including free Java runtimes.~~
3. If you want to edit your db-data, you can just open

the file and edit the insert statements

4. Transaction Support: MVCC, or combination of locks and MVCC

5. HSQLDB has two **main table types**: MEMORY and CACHED

6. HSQLDB 2.0 supports all the core features and optional features of SQL:2008. Advanced features include user-defined SQL procedures and functions, schemas, datetime intervals, updatable views, arrays, lobs, full and lateral joins and set operations. Many non-standard functions such as TO_CHAR and DECODE are also supported. Extensions to Standard SQL include user-defined aggregate functions.

7. Used by OpenOffice, tested and **stable**

Cons:

1. Slows down proportionally t**o the size of the data**.

2. Not scalable to the data and not good option in production

3. Difficult to access outside of our app (e.g. for custom reports)

4. Transactions / disk-sync is difficult to get right, so

it's easy to lose data.
5. Apart from B+/B- tree indexes, HSQLDB doesn't support Hash, Full Text index as supported by H2
6. **Doesn't support "Cursor" type as supported by Derby**
7. No implicit support for JPA, so we may need to use Hibernate on top of HSQLDB.

2. H2

Pros:
1. For most operations, the performance of H2 is about the same or better than HSQLDB, Derby
2. Apart from B+/B- tree indexes, H2 supports Hash, Full Text index
3. It can be embedded in Java applications or run in the client-server mode
4. The disk footprint (size of the jar file) is about 1 MB
5. The main programming APIs are SQL and JDBC, however the database also supports using the PostgreSQL ODBC driver by acting like a PostgreSQL server
6. It is possible to create both in-memory tables, as well as disk-based tables

7. Table level locking and multiversion concurrency control are implemented
8. SSL / TLS connections are supported in the client-server mode, as well as when using the console application.
9. Two full text search implementations are included, a native implementation and one using Lucene
10. A simple form of high availability is implemented: when used in the client-server mode
11. The database supports protection against SQL injection by enforcing the use of parameterized statements. In H2, this feature is called 'disabling literals‘
12. An embedded web server with a browser based console application are included, as well as command line tools to start and stop a server, backup and restore databases, and a command line shell tool.

Cons:
1.Situation where H2 is slower than HSQLDB is large result sets, because they are buffered to disk if more than a certain number of records are returned. The advantage of buffering is: there is no limit on the result

set size
2. H2 doesn't support Merge Joins as supported by HSQL and Derby
3. **Doesn't support "Cursor" type as supported by Derby**
4. No implicit support for JPA, so we may need to use Hibernate on top of H2

3. Derby- not use

Pros:

1. Derby's database engine, is a full-functioned relational embedded database-engine, supporting JDBC and SQL as programming APIs. **It uses IBM DB2 SQL syntax**.
2. The network server allows clients to connect over TCP/IP using the standard DRDA protocol. The network server allows the Derby engine to support networked JDBC, ODBC/CLI, Perl and PHP.
3. An embedded database can be c*onfigured to act as a hybrid server/embedded RDBMS; to also accept TCP/IP connections from other clients in addition to clients in the same JVM*

Cons:
1. Derby is the **slowest** embedded database than compare to H2, HSQLDB, Postgres, MySQL
2. Number Type to VarChar conversion is pain in embedded derby
3. Apart from B+/B- tree indexes, **Derby doesn't support Hash, Full Text index as supported by H2**
4. Derby doesn't support common table expression but these are supported by both H2 and HSQL
5. Derby doesn't support Data Domain which is supported by H2 and HSQL
6. No implicit support for JPA, so we may need to use Hibernate on top of Derby

4. ObjectDB
Pros:
1. Can be used in both client/server and embedded model
2. ObjectDB is the only Object Oriented Database with built in support of JPA 2
3. **Performance of ObjectDB is better than Hibernate over H2, Derby, HSQLDB**

4. Database is stored as a single file
5. Tested with Tomcat, Jetty, GlassFish, JBoss and Spring
6. On the JPA website, they have mentioned some stats for the performance comparison of ObjectDB Vs Different Databases + Hibernate. http://www.jpab.org/ObjectDB.html

Cons:
As such, I didn't found any major 'con' for the ObjectDB. Only thing is that this the best choice if you are planning to use some abstraction like Hibernate on top of DB. If that is the case then ObjectDB is better choice, as it has out of box integration with JPA.

5. Neo4j and OrientDB:
Graph based databases and useful more in terms on "big-data" kind of application. Neo4j is less dependent on a rigid schema, and is more suitable to manage ad-hoc and changing data with evolving schemas. Conversely, relational databases are typically faster at performing the same operation on large numbers of

data elements.
First stable release of OrientDB is available from July, 2012.

I wrote standalone programs for first four databases (H2, HSQLDB, Derby, ObjectDB). Writing embedded database was pretty straightforward. **I am really impressed with the simplicity in the usage of JPA for the ObjectDB Database**.
Considering the pros and cons and the requirement of using legacy "SQL" written, I decided to choose H2 Database. ObjectDB was one of the promising candidates but because of the need of using the well-tested "SQLs" available for the project, it was dropped. HSQLDB was another candidate, but apparently performance wise H2 looked better promising and lot of people have recommended to use H2. H2 is been written by the same developer group that wrote HSQLDB so in a way, they tried to overcome different shortcoming in the HSQLDB.

Hope this information is helpful if you are planning to choose light weight java database for your project

Disclaimer

This information was automatically retrieved on 2013-09-05T11:00:26+00:00 from:

http://sayrohan.blogspot.in/2012/12/choos
light-weight-java-database.html

and automatically parsed, clean up and interpreted by dotEPUB.com. It is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. This e-book is not an authoritative source: please, visit the original webpage.

Help improve the dotEPUB code and report issues at:

http://code.google.com/p/dotepub/

dotEPUB’s cleanup process is partially based on a modified old version of Readability (© by arc90).

(v. 0.8.8)