REVERSE ENGINEERING

H2 DATABASE
RUNTIME TERROR

Introduction

This report contains a high-level picture of the h2 application ie the H2 database architecture. It is really important to understand the architecture of a project if we are going to make code changes, review code, or build a new feature. Sticking to an architecture helps us write code that is maintainable, extendable and scalable (from an Object Oriented Design perspective, and not from a System Design perspective). The report also gives an analysis of the Social Context of the project. This can help us understand project popularity, the status of the project(if active or not), etc.

Architecture

The h2 database application follows a layered architecture style. The layers are as shown in Fig 1.

The **Console** is the user interface of the application. Each user request from the UI goes to the **Web Server** component where it spawns a new thread for processing each request. The **JDBC Driver** layer creates a new session and JDBC connection objects for each login request. For the other requests it' ll reuse the already created session and connection objects.

The **Connection and Session layer** creates and manages the JDBC Connection and session objects. JDBC connection object represents a connection to the database. The session object represents whether the database is in embedded or remote server mode.

The input SQL strings are parsed into equivalent SQL commands using the **Parser** module. The parser generates a command execution object which is responsible for executing various SQL DDL and DML commands.

The **Table/Index/Constraints layer** is responsible for implementing various kinds of tables and indexes and enforcing database constraints like locking the table. For every session, this layer also uses **the transaction logging layer** and logs into UndoLogRecord.

The **command execution and planning layer**, then stores the data into hashmaps, using the MVStore layer. Ultimately the data is copied into a disk, but before that is done, MVStore uses various concurrent hashmaps to store data, so that it can be efficiently accessed (Since hashmap retrieval happens in constant amortized time). This layer also uses the transaction logging layer.

The MVStore layer, in turn, uses the **file system abstraction layer**, whose purpose is to create an abstraction of the file accesses so that both (in memory) and disk operations are treated the same by the higher layers. Abstraction makes it easy to write extensible code.

Comparison with the documented architecture

The documented architecture is as shown in Fig 2. There are slight differences with the as-implemented architecture. For example, the transaction logging layer is being accessed from both Command, Execution and Planning layer and Table/Index/Constraints layer in the as-implemented architecture while the documented architecture mentions that it is accessed only from the Table/Index/Constraints layer.

The documented architecture was written before the development of the MVStore Engine. It still mentions the B-tree based storage engine which was eventually replaced by the MVStore Engine. Even though it was mentioned that MVStore will replace the B-tree engine in the future, the architecture hasn't been updated yet.

The documented architecture doesn't also mention the direct relation between the Command Execution and Planning layers and the MVStore layer which we found out from the source code.

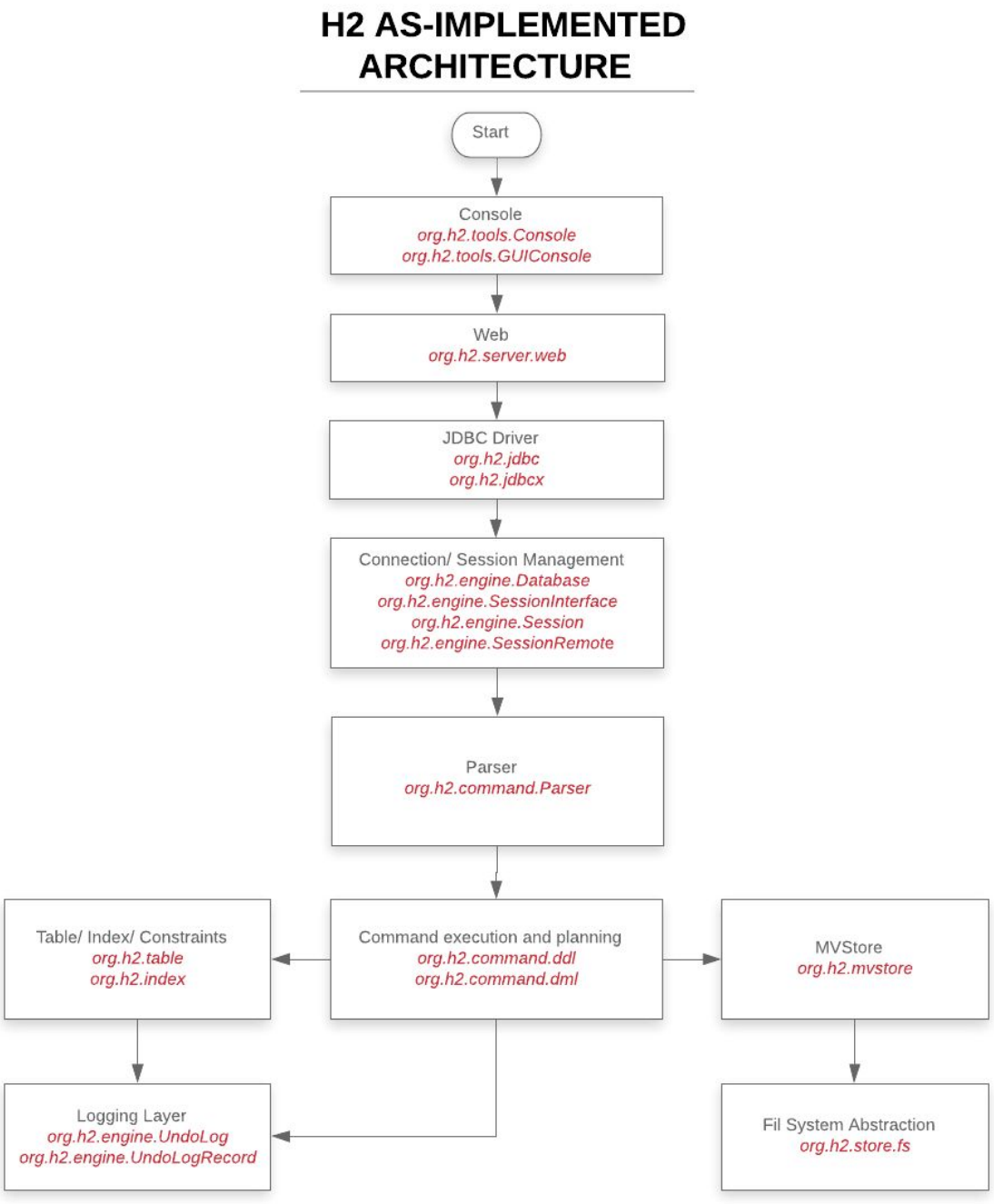
We have also identified Console and Web components to be important and worth mentioning in the architecture of the system as it represents the UI and Web components respectively. This was not mentioned in the documented architecture.

The h2 architecture as described (before implementation) was a very strong architecture, and we found that the as implemented architecture is slightly different. This is good since this means there is less **architectural drift**. Overall we have found that the h2 community has done a really good job at maintaining coding standards, styles and has kept a strict check on pull requests and merges so that any new request doesn't violate the architecture.

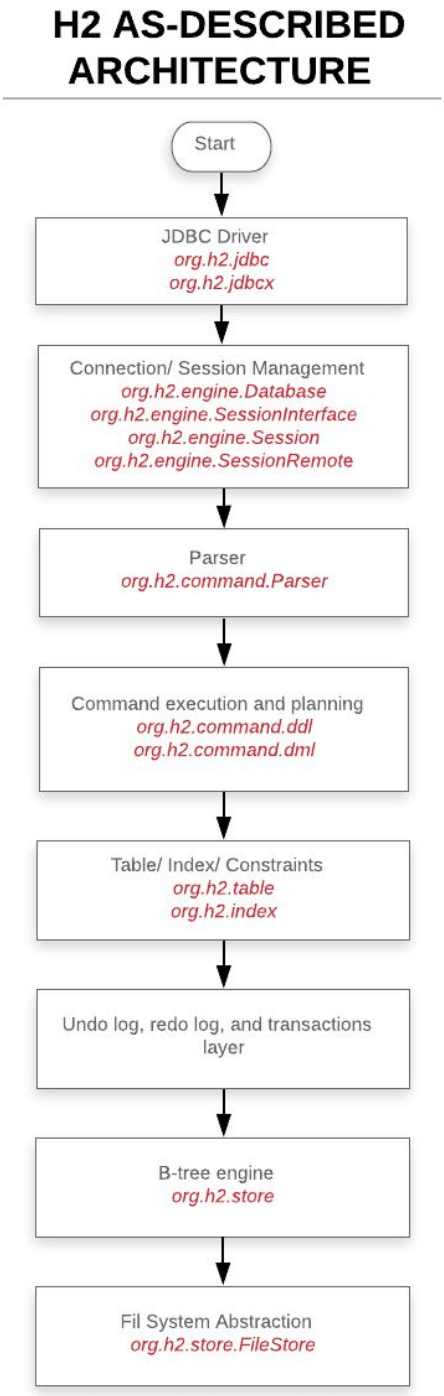
Architecture Recovery

We had a really good beacon to guide our search. The as-described architecture of H2 was present on their website. So that was a good starting point to dive into the as-implemented architecture. We saw various packages present in the H2 source code and correlated them to various layers. In each layer, we identified key classes (as mentioned on the website) and also through our method where we executed a query and debugged the flow of execution from beginning to the end. For each layer that we encountered, we checked the import statements in the key classes to see which packages are being used in these classes, which would verify our hypothesis that this layer interacts with a particular layer (since the other layer's classes are imported in this class). We started to sketch out this model and came up with the result as shown in Fig No 1.

As-Implemented Architecture (Fig 1)



Documented Architecture (Fig 2)



Social Context

State of the project

- Level of usage - Several companies across different industries like Technical, Business services, Finance, Retail, Education, etc. are using h2 database. Few examples are Cisco, Pepsico, DBS Bank, eviCore healthcare etc.
- Last release date - 10/14/2019
- Release roadmap - https://h2database.com/html/roadmap.html
- #commits as of late - 12,221
- #open issues as of late - 220
- #active developers - 116

Standards

- Only use Java 7 features (do not use Java 8/9/etc).
- Use spaces instead of tabs
- A tab consists of 4 spaces
- The checkstyle configuration for the coding style can be found at https://github.com/h2database/h2database/blob/master/h2/src/installer/checkstyle.xml
- A template of the Eclipse settings can be found at https://github.com/h2database/h2database/tree/master/h2/src/installer/eclipse.settings
- New Test cases must be integrated into src/test/org/h2/test/TestAll.java and for SQL level tests, check SQL files in src/test/org/h2/test/scripts.
- End-user documentation should be added in src/docsrc/html/*
- New grammar changes documentation should be added in src/docsrc/help/help.csv
- Changelog entry should be added in src/docsrc/html/changelog.html.
- New words should be added in src/tools/org/h2/build/doc/dictionary.txt.

The typical process of contribution

The guidelines to contribute to the project is as follows:

- The code should use only up to Java 7 features.
- The coding style used prior to the project must be followed consistently and use Checkstyle to verify the same.
- Follow the standards as mentioned in the previous section
- It should also include test cases integrated into the test suite for the submitted code and should cover at least 90% of the submitted code. To verify this, use coverage tool or build with coverage
- The test cases must be integrated into src/test/org/h2/test/TestAll.java and for SQL level it must be in src/test/org/h2/test/scripts
- Submitted code should not break the existing code. Verify by running all the given test cases
- Provide documentation for the submitted code
- Provide a changelog entry in src/docsrc/html/changelog.html
- Verify the spellings using tools like spellcheck. If new words are used, add them in src/tools/org/h2/build/doc/dictionary.txt
- Run src/installer/buildRelease to find and fix formatting errors
- Verify the formatting using build docs and build javadoc
- Submit changes using GitHub's "pull requests". You'll require a free GitHub account

Tools supporting the process

- Version control/Source code repository: Github
- Issue tracker: Github
- Build Tool: Maven
- Test Tool: JUnit with Mockito
- Code Review Tool: Github
- Continuous Integration Tool: Travis CI
- Plugins: Eclipse Checkstyle Plug-in, EclEmma Java Code Coverage

Pull requests

1. Add support for custom Lucene Analyzers #1524: The interesting part about this pull request is that it shows the care maintainers take in writing comments as well. This pull request created by username arteam had many issues, the key developers pointed out his issues out so that he could fix them and make a new pull request. For each of these fixes, **there was a separate comment.** Another interesting bit of information is that this pull was created in October 2018, and it still hasn't been merged.
2. Fix issue #2430: pgjdbc can read `bytea` data without `forceBinary` #2431: User auntyellow provided a fix for an issue that was actually a temporary workaround which wasn't actually solving the root cause of the problem as pointed out by the key developer. This actually helped the key developer to identify an existing problem that was not discovered before. Therefore, she created an issue for the same and provided a fix for it.
3. Split VARBINARY and BINARY data types #2427: The system was allowed to export the temporary data into SQL scripts with the incorrect data types. The columns with variable length data were being treated as BINARY columns instead of VARBINARY columns. So the user suggests adding a compatibility setting to load the scripts correctly that when activated treats BINARY columns as VARBINARY. This required some changes in the Parser module that lets the SQL parser rewrite the original datatype and store it as VARBINARY and also to add some additional clauses in the RUNSCRIPT command to apply the changes while executing the scripts.

4. Add own implementation of SQLType #2409: The key developer added their own implementation of SQLType. This pull request contains constant name changes and their implementations. They have changed this for their own greater convenience and to avoid confusion.

5. Chunk occupancy mask #2386: This pull request adds a new strategy to determining live pages by introducing concepts like TOC and occupancy bitmask for each chunk.

Following this new strategy shows a 3-fold reduction in reads and 2-fold write reduction compare to master. Database file growth also slows down significantly.

Issues

1. H2 DB backup issue #2428: In this issue, the creator of the issue was complaining about him/her not being able to copy the database file while it was being used. They complain about this because a version prior to an upgrade, he was able to do so. Unfortunately, one of the maintainers closed this issue, citing that this operation was never supported, and the issue creator was just getting lucky.

2. Remove incorrect rows from DatabaseMetaData.getTypeInfo() and INFORMATION_SCHEMA.TYPE_INFO #2318: This issue is interesting because it shows how particular H2 maintainers are, even with issue names. One of the key maintainers(username: katzyn) changed the title of the issue to better reflect the problem. This issue was fixed later.

3. org.h2.jdbc.JdbcSQLException: Database may be already in use: "Lock file exists" error #2203: The user says he is getting this particular SQL exception because of the existence of db.lock.db file and he that this happens during their service failover when the abrupt killing of JVM happens. One of the key maintainers points out that this happens because the user's h2 version is outdated and not supported. So he asks the user to update the database to a newer version that has a better file locking protocol.
4. NO ACTION should be different from RESTRICT #2440: Here the user is working on a database that refers to another record within the same table. When he/she tries to delete records in which the first record is referred to by the second record, a referential integrity constraint violation is occurring which was not happening in the earlier version. The user further says when he/she tested the deletion using ON DELETE CASCADE constraint, no violation is happening which makes the behaviour

of h2 different from SQL Server, Oracle etc. On further analyzing the issue, one of the key developers found that h2 incorrectly treats ON DELETE NO ACTION and ON DELETE RESTRICT commands in the same way. Ideally ON DELETE RESTRICT should prohibit deletion of a particular row if there are any matching rows while ON DELETE NO ACTION does not perform its constraint check until the entire set of rows to be deleted has been processed.

5. ON COMMIT DELETE ROWS doesn't work with GLOBAL TEMPORARY tables #2459: This is interesting because the tables should have worked in the same manner, but still this issue seems to be not replicating for local tables but only for global tables.