H2 Database
Phase 3

Section 1

Ajinkya Rode
ar2443@rit.edu

Amruta Chavan
agc9066@rit.edu

Group 3

Samyukta Bharadwaj
sb2773@rit.edu

Saurabh Puri
srp3858@rit.edu

ABSTRACT

Nowadays, open source databases are extensively used in organizations for relational database management systems. We have selected open source H2 database engine for analysis. In this report we are explaining some limitations H2 faces and the solution to one of those, which we are implementing. H2 is very similar to the MySQL as they both are relational databases but there are some features which have not been implemented in H2. We have implemented two new features which are quite useful for large databases. In addition to the two, we also implemented an additional feature on the Query Optimization part. Following sections explain more about them.

Limitations of H2 Database:

Although H2 database is easier to use and faster compared to several other open source databases, it does have certain limitations like: dropping multiple columns is not possible in H2. This feature can also be very helpful when the database is large. The major advantage with this feature is that, in a single query we can drop multiple columns without the need to write multiple queries for each column. Another feature that we are implementing is adding a new data type "enum". Enum data type is available in databases like Oracle and can be very useful when a column is to be assigned values from a list of pre-defined values.

About H2 Database

Relations and indexes: Relations in H2 are similar to that in SQL relations. In fact, in most of the cases it can be seen that H2 is faster than any other database engines, be it open source or not. It is a single connection database, that is, it has just a local connection to the database in which there are many basic operations running in parallel to the database. H2 is slow when there are large result sets,which cause the data to be buffered to disk when the limit exceeds a certain number of records but the advantage of buffering is that, there is no limit on the result set size. H2 uses Hash indexing technique for databases in the memory. As Hash indexes can only test for equality, and do not support range queries, the hash keyword is ignored for other tables having multiple indexes [5]. The relation between the data is stored in two dimensional tables. Data is stored in rows in which each row is uniquely identified by rowID. The data type of rowID is long. It is stored according to the rowID. These rowIDs are generated automatically. Indexes play a major role in increasing the performance of querying the database. Indexes for columns such as primary key or foreign key constraints are automatically generated. Using hash index also the performance is increased. Hash indexes are usually stored in hash tables and this results in increase of performance. In-memory database automatically uses hash indexes.

Record insertion into files: Insertion of new rows in a table is performed in two ways: direct and sorted. In direct insertion, query results are directly inserted in the table without any intermediate step. While in sorted insertion technique, b-tree pages are split, which improves performance and reduces disk usage. Different database engines perform insertions in a different method, but H2 supports SQL standards making it compatible with other databases. Also there are few differences in the insertion of records in the files compared to other databases. For example, in MySQL text columns are case insensitive but, in H2 they are case sensitive. Yet another way of inserting records is using prepared statement which can be used for JDBC connectivity in H2 that makes it easier to insert records into the files/database [5]. In H2 the data is stored in a single file. There are total of two file headers. Both contain the same data but it could be a case that during the process of updating the file headers, the write operation would not be completed for certain amount of time to complete the operation. This time second header comes in use while first one gets corrupted. File header contains block number, block size, chunkID, file format number, etc. Chunk is contained in every version of file format.

Keys and address insertion into indexes: H2 Database has a storage subsystem called MVStore which contains a set of stored maps. Features provided through these maps is lookup. Through lookup, we get access to first and last keys, associated with a single map, traverse through some or entire set of keys. Problem faced with maps is inefficiency that is associated with indexing and looking up. By using this map, we get an increased rate of indexing which helps to find data faster. This is helpful when we want to access a record based on the index provided by increasing the efficiency and also providing a cost effective way. The maps are analogous to a table. Every table has a primary key and it is used according to the key of the map. Data is stored in maps instead of the tabular format of rows. The snapshot consists of

these maps. The writing of data is done only when we complete copying the previous one and this method is known as Copy on Write(COW). We can even rollback to the previous version but need to see that the data is not purged.[4]

Page Fetching: All the maps with their data are stored in B-Trees. Leaf nodes serve as a root node. Leaf nodes include data stored in pages and key-value pair. Internal nodes contain pointers to the leaf nodes. Ordering of pages begins with storing root nodes and then internal and leaf nodes. This helps to increase the speed sequentially when random reads are performed. H2 uses the concept of paging like all other databases. It divides the memory into pages. Default page size is equal to 2 kilobytes when a database is created, which seems to be insufficient. Moreover, increasing the page size value does not improve performance. The size of page for existing databases can not be changed, so it needs to be set at the time of database creation only.[5]

Query processing and optimization: Querying in H2 database is similar to standard SQL. When a query is run, a connection is established with the database and an execution statement is created. As the query is run, the server which is turned on for listening, fetches the query from the input stream. Initially the Server is started manually. The object from TcpServerThread checks as to which type of query it is and creates a session accordingly which further executes the query depending on its type. For an insert query appropriate rows are inserted depending on which trigger was initiated and this process is similar for update and select commands. [5]

Modes: H2 has three types of connection modes; Embedded mode, Server mode and Mixed mode. Embedded mode is the most common and fast one, where in the database opened by the H2 application and the application itself are a part of the same JVM. This it does by using JDBC only. Server mode is the one where the H2 application operates on a database at a remote location. This is done by using JDBC or ODBC and the entire communication is done over TCP/IP and hence is slower. After server starts, only then the application can be connected to the database. This mode is slower than the embedded mode as data has to be transferred between the application and the database using TCP/IP. Lastly, as the name suggests, Mixed mode is a combination of both Embedded and Server mode. Alongside opening a database in the same JVM, the H2 application also turns on its server mode in order to enable multiple access to the database at the same time. [4] The database can be accessed using the same url by both the local and remote applications.

Implementation of the new Feature:

1) Dropping multiple columns in single query:

Our first proposal is to modify the current implementation of alter table drop column query to support multiple columns. In this section we would be explaining about the multiple column drop query and explain the design and implementation considerations, the system architecture and the design choices made to implement it.

Significance of multiple column drop query:

To delete a column, the table structure can be changed using the alter table drop column command which specifies the column to be removed. In the current implementation of H2, the alter table query can drop only one column at a time.This will not be efficient when we want to drop multiple columns from a relatively large database, because a separate drop query would have to be written for each column to be dropped. Hence having an alter table query that supports dropping multiple columns is necessary.

Design:

In this section we will first look at the original design of dropping multiple columns in different databases and the errors, if similar ones are executed in H2, are as follows:

1. *Oracle:*

```
ALTER TABLE Table_Name DROP (Column_Name 1, Column_Name 2);
```

A query with similar syntax in H2 gives the below error

```
Syntax error in SQL statement "ALTER TABLE TEMP DROP ID,[*]DROP DEPT "; SQL statement: alter table temp drop (id, dept) [42000-188] 42000/42000
```

2. *MS SQL:*

```
ALTER TABLE Table_Name DROP COLUMN Column_Name 1, Column_Name 2;
```

A query with similar syntax in H2 gives the below error

```
Syntax error in SQL statement "ALTER TABLE TEMP DROP COLUMN ID,[*]DEPT "; SQL statement: alter table temp drop column id,dept [42000-188] 42000/42000
```

3. *MySQL:*

```
ALTER TABLE Table_Name DROP Column_Name 1, DROP Column_Name 2;
```

A query with similar syntax in H2 gives the below error

```
Syntax error in SQL statement "ALTER TABLE TEMP DROP COLUMN ID,[*]DROP COLUMN DEPT "; SQL statement: alter table temp drop column id,drop column dept [42000-188] 42000/42000
```

4. *PostgreSQL:*

```
ALTER TABLE Table_Name DROP COLUMN
Column_Name 1, DROP COLUMN Column_Name 2;
```

A query with similar syntax in H2 gives the below error

```
Syntax error in SQL statement "ALTER TABLE TEMP
DROP ID,[*]DROP DEPT "; SQL statement: alter table temp
drop id,drop dept [42000-188] 42000/42000
```

New Design:

The new query design we developed as a new feature in H2 is as follows:

```
"ALTER TABLE Table_Name DROP COLUMN
(Column_Name 1,Column_Name 2);"
```

The above syntax has a simple modification i.e the column names are included in the parenthesis. All the columns included in the parenthesis are dropped in a single query.

Design Choice and Implementation:

In order to implement an alter table drop column query with feature to drop multiple columns in single input query we had shortlisted two approaches:

The first approach that we took was as follows :

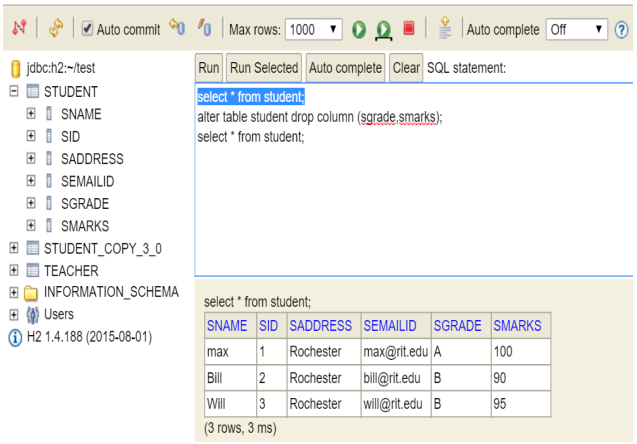
We studied the basic query processing flow by introducing multiple breakpoints and debugging the code. We observed that the most important part in any query processing was its parsing and then forming a command object which was then returned for execution.

We modified the *Parser.java* file's *parseAlterTable* function. In our implementation we are able to accept multiple columns in the drop column query. These column names were further parsed to get a list of individual columns.But still only one column was being dropped because the command object which was used for processing was overwriting all the column names with the last column name. The functionality of the *parseAlterTable* function could not be modified further because it would have affected functionality of other queries.

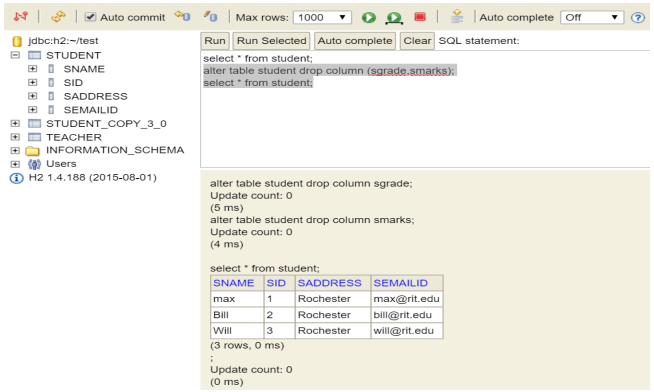
After we could not proceed with the initial approach, we planned another approach as follows:

In this approach we planned to parse the query at the very initial stage before starting any processing or calling any other function. For this we modified the *WebApp.java* file's *query* function. We parsed the query and if the query contained keywords like 'alter' and 'column' then the query string was passed through our implementation. In our implementation, we parsed the original sql string to get the name of columns that are to be dropped. We then constructed new alter table drop column queries with the individual column names. All these queries were concatenated to form a new string and separated by the ';' character. If the original sql query contained any sql statements other than alter table then those statements were also appended to the new string. The original sql string was then replaced with this new string. This sql was then forwarded for further processing which was the same as original implementation.

Result:



The above figure shows the original table "Student" with all the columns and data. In the next figure we will see the execution of multiple column drop.



In the above figure we can see that the columns 'sgrade' and 'smarks' have been dropped from from the 'student' table.

Accomplishments:

We were able to successfully implement this feature as part of the project. We tested this feature with different inputs and obtained desired results. This feature will be very useful in case of large databases where many columns are to be dropped at once.

2) Implementing "enum":

We have implemented the concept of Enumeration for H2 database. Enumeration in Java is an interface that uses Enums to add constraints to the database. Enum implementation is quite challenging. There are several things to be kept in mind. A major point to be noted for Enums is that, they are mainly useful at

compile time. This is because they are generally meant to be static and immutable.

Significance of Enum:

Enum is a datatype which contains a set of named values. For a variable it enables it to be a set of predefined constants. The variable should be equal of a value from the predefined constants. Enumerated types help to make the code more self-documented. It also helps to avoid illegal mathematical operations by the user on the values of enumerators. Enum is used when we need a predefined list of values which do not represent any kind of textual or numeric data. We have added a new datatype, ENUM. This datatype is supported in Oracle, PostgreSQL, MS SQL and MySQL. Enum is basically short for Enumerated data type. Enums are constants that when declared, and then used, make sure that no other value apart from the constants are allowed.

Design:

Example in Oracle:

```
CREATE TABLE DBSI(student_name VARCHAR(40), section ENUM(‘one’, ‘two’));
```

Example in PostgreSQL:

```
CREATE TYPE Layer AS ENUM (‘one’, ‘two’,’three’);
```

Example in MYSQL:

```
CREATE TABLE DBSI( student_name VARCHAR(40), section ENUM(‘one’, ‘two’));
```

New Design:

```
Create Table Test ( ID INT PRIMARY KEY, NAME VARCHAR(255), COLOR ENUM(“pink”, “green”) );
```

Implementation:

The implementation takes place in the following Java classes with the additions explained below:

src/main/org/h2/command/Parser.java - initially when the query is fired, there is a Parser class that parses the sql command, splits them in characters and and stores them in an ArrayList of type String. The reason we use String type is because Enums are constants and String is immutable. Storing into the ArrayList is done by taking the entire SQL string storing it character by character in an Array and then looping through it to read each character. These characters are then read using switch case condition. In the *parseColumnWithType* method inside this file, we added a new check which reads the file when the datatype is enum and then stores in as an Expression type. A new method called *readExpressionEnum* is written that takes the enum ArrayList and the column name as parameters. This method is a special case of *readExpression* method that is called only when the datatype is Enum.

In the readExpressionEnum method, the Constant Enum values within the enum ArrayList are the then converted to something called as a ValueExpression which holds constant values of any datatype and converts it into an Expression type.

src/main/org/h2/command/Column.java : For an enum, we need a list of the string constants inside it and a boolean value that states that it is an Enum value. And then we created a new constructor for it.

src/main/org/h2/expression/Expression.java - This is the Expression class. It is an abstract class that consists of all methods required to check constraints. Enum by concept also checks constraints and disallows insertion of any value that is not a part of the enum ArrayList. The Expression class’s methods optimize and mapColumns is overridden for each constraint type to perform special functions for that constraint.

src/main/org/h2/expression/Datatype.java - A new datatype is added for the enum type in the datatypes arraylist. For datatypes, attributes such as type, precision, support, case sensitive and so on would be assigned. For enums, the precision and support data are set to false as they are useful only in case of integers, long and double values. ArrayList of String constants are what is majorly required for Enums.

src/main/org/h2/value/Value.java - In order to achieve compatibility with other classes an ENUM constant has been declared with a particular number. This is then cross checked for an discrepancy with the datatype mentioned in the query in console. This has been implemented for all datatypes.

Advantages and Disadvantages of Enum:

If a large amount of data is added as constants in the Enum list and a value needs to be added or removed, then it is required to alter the entire table in order to accommodate the change. As mentioned, Enums are better to use during compile time than runtime. But this is only because they are supposed to be immutable and hence highly secure. Moreover, if Enum is used in place of Constraints, it is more compact as it is a datatype that enforces the constraint and hence doesn’t require to be specified explicitly.

Accomplishments And Failures:

The enumeration feature was added successfully. This feature implementation has not impacted existing functionalities. Create and insert are implemented successfully. However, Update and Delete functionality remains the same. Since Enum consists of immutable constants, changing a particular value in the Enum list already created is out of scope.

3) Implemented multiple ‘select *’ statements:

In addition to the previous features we also implemented third one which is on similar lines to that of multiple table selection. The

original query is designed to get the cartesian product of all the selected tables. This new query can select all data from individual tables.

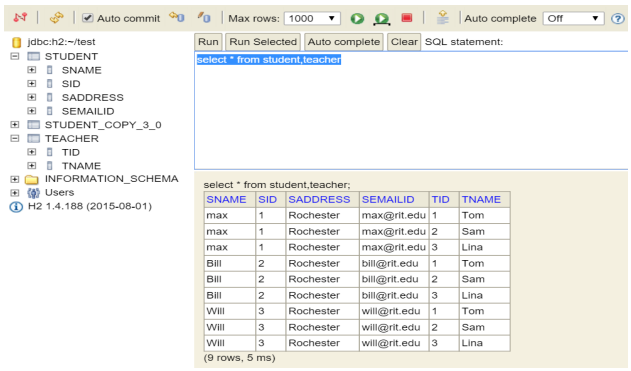
Significance of multiple select table query:

On large tables when we select multiple tables, the database generates a cartesian product, thus creating a large table. In a situation where we want to display all records of multiple tables this query wont work and thus needs to be modified slightly. The process of physically selecting a table can be time consuming and resource consuming. So to select multiple tables using the 'select *' command we implemented a new query. This query can be useful for selecting multiple tables from a large database, because a separate 'select *' query would have to be written for each table to be selected. Hence having a 'select *' query that supports selecting data from multiple tables is necessary.

Current Implementation in h2:

In this section we will first look at the original syntax of selecting multiple tables and the result of original query that leads to the cartesian product. The following image shows the result of executing the query:

"SELECT * FROM Table_Name 1, Table_Name 2;"



Our Design:

The new query design we developed as a new feature in H2 is as follows:

"SELECT * FROM (Table_Name 1, Table_Name 2);"

The above syntax is slightly modified from the original query i.e the table names are included in the parenthesis. All the columns included in the parenthesis are selected individually in a single query.

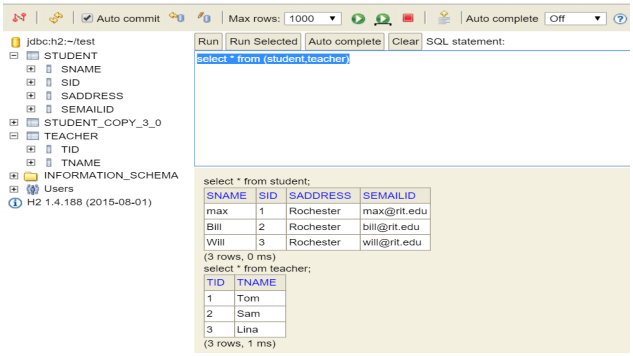
Implementation:

The approach that we took to implement this feature is as follows:

We modified the *WebApp.java* file's *query* function. We parsed the query and if the query contained keywords like 'select' and '*' then the query string was passed through our implementation. In this we parsed the original sql string to get the names of the tables on which the select operation was to be performed. We then constructed new select * queries with the individual table names. All these queries were concatenated to form a new string and separated by the ';' character. If the original sql query contained any sql statements other than the select * query then those statements were also appended to the new string. The original sql string was then replaced with this new string. This sql was then forwarded for further processing which was the same as original implementation.

The result after implementing the new feature can be explained in the following figure.

Result:



Accomplishments:

We implemented this feature as an additional feature for the project. We tested this feature with different inputs and obtained desired results. This feature will be very useful in case where the user wants to see contents of multiple tables at once for comparison.

Accomplishments And Failures:

Successfully added multiple drop column.

Successfully added multiple select databases.

Successfully added enum data type.

References:

[1] http://www.h2database.com/html/mvstore.html

[2] http://www.iasj.net/iasj?func=fulltext&aId=49884

[3] http://h2database.com/html/features.html

[4] http://www.h2database.com/html/mvstore.html#maps

[5] http://www.h2database.com/html/main.html