Re-designing of CyDIW- Cyclone Database Implementation Workbench

by

Anubhav

A technical report submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Shashi Gadia, Major Professor
Simanta Mitra
Andrew Miner

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this technical report. The Graduate College will ensure this technical report is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University
Ames, Iowa
2018

Copyright © Anubhav, 2018. All rights reserved.
DEDICATION

The last few years have been really challenging in terms of shaping my career as I wanted it to be. Today, I am half way through it and really happy for my time-period at Iowa State university (ISU). Starting from the invitation letter I have received from Computer science department at ISU for my MS program, until today, I had a great learning experience from my professors, and ISU students. Also, without the immense mental & emotional support of my family living overseas, this would have never been possible. So, I would like to dedicate this work to my Prof. Sashi Gadia & Prof. Andrew Miner, internship mentor Dr. Lauren Charles, colleague Junaid Babar & Racheet Matai, father Dr. R.P Singh, mother Mrs. Seema & dear sister Ayushee. Their immense support, guidance, feedbacks, motivation has invaluable contribution towards it.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>iv</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>v</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>vi</td>
</tr>
<tr>
<td>CHAPTER 1. INTRODUCTION:</td>
<td>1</td>
</tr>
<tr>
<td>CHAPTER 2. ISSUES &amp; CHANGES IN DETAILS</td>
<td>4</td>
</tr>
<tr>
<td>APPENDIX A. Revised language constructs and commands in CyDIW</td>
<td>12</td>
</tr>
<tr>
<td>Basic CyDIW command format</td>
<td>12</td>
</tr>
<tr>
<td>1.1 Use of variables</td>
<td>13</td>
</tr>
<tr>
<td>2 Listing all commands</td>
<td>14</td>
</tr>
<tr>
<td>2.1 For loop</td>
<td>14</td>
</tr>
<tr>
<td>2.2 While loop</td>
<td>14</td>
</tr>
<tr>
<td>2.3 If condition</td>
<td>14</td>
</tr>
<tr>
<td>2.4 If-else condition</td>
<td>14</td>
</tr>
<tr>
<td>2.5 Logical operations</td>
<td>14</td>
</tr>
<tr>
<td>2.6 Support for logging</td>
<td>14</td>
</tr>
<tr>
<td>2.7 Variables creation, assignment, destruction</td>
<td>15</td>
</tr>
<tr>
<td>2.8 Displaying formatted files</td>
<td>15</td>
</tr>
<tr>
<td>2.9 Text file manipulation</td>
<td>15</td>
</tr>
<tr>
<td>2.10 Discontinued commands</td>
<td>16</td>
</tr>
<tr>
<td>APPENDIX B. Importance of CyDIW over General scripting language</td>
<td>17</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CyDIW Graphical User Interface (GUI)</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>SystemConfiguration.xml</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>demoBenchmark.xml file</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>$EngineA_output.xml file</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>benchmark_Engines_log.xml</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>EngineA_out.xml</td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td>EngineB_out.xml</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>Saxon_out.xml</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>A Benchmarking experiment</td>
<td>18</td>
</tr>
<tr>
<td>10</td>
<td>A Benchmarking experiment continued</td>
<td>18</td>
</tr>
<tr>
<td>11</td>
<td>benchmarkQ.xml</td>
<td>18</td>
</tr>
<tr>
<td>12</td>
<td>Python Client usage within a the CyDIW GUI</td>
<td>20</td>
</tr>
</tbody>
</table>
ACKNOWLEDGMENTS

I would like to thank my committee chair, Dr. Shashi Gadia, and my committee members, Simantha Mitra, and Dr. Andrew Miner, for their guidance and support throughout the course of this project.

In addition, I would also like to thank my friends, colleagues, the department faculty and staff for making my time at Iowa State University a wonderful experience.
ABSTRACT

Cyclone Database Implementation workbench (CyDIW) is a platform that is used to implement new database prototypes, use command-based external systems (clients) & perform benchmarking. The workbench supports a scripting language custom-designed for performing comprehensive & self-contained benchmarking experiments. In this project, I have identified the flaws in the current design of the CyDIW’s scripting language and made the language more intuitive & easier for users to learn & use. In CyDIW, we have database prototypes such as Enterprise Database (EDB), NC-94(a spatiotemporal database) and other clients For Example Saxon, MySQL, OOXQuery(object-oriented XQuery), Calc (simple calculator), SPLIT, R- a statistical tool etc. which have their own behavior provided by their APIs. These are interfaced with workbench using adapters. These adapters should tell how a particular client should handle instructions/commands. It has been observed that the adapters were performing parsing of their command which was undesirable and were causing conflicts. I have performed appropriate changes in the existing clients’ adapters to make them coherent & simplified their code. This will help future developers to quickly design adapters for any new client.
CHAPTER 1. INTRODUCTION:

Cyclone Database Implementation workbench (CyDIW) is a platform that is used to implement new database prototypes, use existing command-based systems, and conduct experiments. In order to support database implementation we have used a page-based storage (A page is a unit of data storage that is brought into main memory from disk when the requested item of data is not in the RAM). There are several command based database prototypes such as EDB, NC94, etc. that have been developed over the years and many other command based 3rd party systems such as OOXQuery, Saxon, MySQL, SCalc, SPLIT, R which could also be interfaced with CyDIW workbench. These systems are also known as clients of the workbench. The experiments are batches of commands on multiple clients. Overall, CyDIW consisting of a central system supporting multiple clients for their benchmarking purpose. These clients have certain behaviors defined by their APIs. Each client supports certain commands and has to provide an adapter that executes its own commands. A client system is registered in CyDIW with a unique command prefix. A command from a client system is executed in CyDIW by using this unique command prefix. There are two ways of interfacing any new client within CyDIW. First, the user could provide an adapter for its client and register it in the CyDIW’s SystemConfiguration file as shown in Figure2. Second, by installing the client on the Operating System and use it along with the “OS” client in CyDIW. In the second case you do not need to provide any adapter as CyDIW will understand that the software package is pre-installed in the current Operating System. A client system can use commands that are native to CyDIW as well as commands from other clients. In order to understand and execute the CyDIW native commands we designed a scripting language & a simple Graphical user interface (GUI).
The GUI acts as an editor & Launchpad to write & execute different batches of commands (an Experiment).

These benchmarking experiments collects performance statistics from multiple clients based on varying parameters and plotting them without leaving the session within GUI. Therefore, these experiments are comprehensive and self-contained. For example these experiments could include client systems such as “R” which provides infrastructure to conduct experiments and could perform benchmarking by collecting statistics based on experiments performed on Crag machines used for High performance computing. We can configure these external client systems with the workbench by recording each entry in a XML document. For example the two clients SAXON & R registered with the workbench.
Figure 2 SystemConfiguration.xml

This configuration file will tell CyDIW where to look for external software packages. If a particular system entry is enabled then, at the time of running CyDIW, it will pick up all configuration setting related to it from above file and using the instructions provided in the adapter of the respective client executes the client specific command on the client system. Upon, successful execution, the run time will be logged into the log file and the output could either be viewed in the CyDIW GUI or could be redirected to an output file.
CHAPTER 2. ISSUES & CHANGES IN DETAILS

The CyDIW’s scripting language is developed using JavaCC, a parser generator. It provides a Java language extension for specifying the grammar of the programming language. In CyDIW a command comprises of a “client prefix”, “client command”,

$\text{CLIENT\_NAME}\, \text{-> CLIENT\_CMD OPTIONS;}

“OPTIONS” followed by a semicolon respectively.

These options are environmental information that tells how the command is handled by CyDIW platform in order to dispense appropriate services. The services could be as simple as saving command’s output to a file or logging the execution time of the commands in an Experiment or create your own custom logging.

“out” option sends out of the executed command to a text file.

\text{out>> [filename.txt]}

“Log Time” option records the execution time of a command on a client.

\text{Log time [>>] [filename.xml]}

The below is a complete command in CyDIW:

$\text{CLIENT\_NAME:}->\text{CLIENT\_CMD out>> [outputFileName] logTime>> <customTag> <logFileName> ;}

The job of the CyDIW’s parser is to parse a command from the CyDIW’s GUI, ensuring that it adheres to the concept of a command as described above. Once the command is parsed, we send the command to its respective client for execution. Upon successful execution of the command, the results could be displayed in the output pane of CyDIW’s GUI or using the above two Options could be captured or logged in to a file (usually an XML file.). Afterwards, these XML files could be further utilized to create different
visualizations. The client’s command is always native to the respective client and will only be understood by each client’s adapter provided from client side. By default, there are a few commands that are native to CyDIW. Each one of them have been explained in the APPENDIX A.

In this project, I have addressed many serious flaws in the existing CyDIW parser and did several major and minor changes to make the CyDIW scripting language clear and intuitive. The older version of the scripting language had a predefined prefix “CyDB” affixed to every command. This creates confusion among the users of CyDIW and doesn’t adheres to the concept of a command in CyDIW as defined above. Moreover, all the CyDIW native commands were bound to this prefix. Even the loops & conditionals were commands. Now, this behavior has been eliminated and all commands previously offered under that prefix have become part of the scripting language. Furthermore, Loops and conditional are no longer commands they are simply language constructs with no “CyDB” prefix. This simplifies the concept of a command. Another good aspect is that the commands are no longer nested inside other commands. There were many additional commands which were added over years for the ease of users of CyDIW. These commands were never been documented properly before. While rewriting the parser again, I have identified and highlighted all the running and suspended commands and created a separate documentation for CyDIW user.

**Before:**

```bash
$CyDB:> displayPDF CyWorkspace/plots.pdf;
```

**After:**

```bash
displayPDF CyWorkspace/plots.pdf;
```

Another important issue was that the user have to use a command called “RUN” for
variable substitution to happen. The “RUN” command was tied internally to “execute” command which actually runs the CyDIW command. This messy dependency of variable substitution over the run command was overdone before and was not needed at all. It actually complicated the language more. Hence, it has been corrected from the parser’s definition. Now, the variable substitution happens implicitly for any variable defined in a CyDIW experiment. For Example the variables are created and set to the desired values first.

```
declare string $$prefix;
declare string $$query;
declare string $$outfile;
declare string $$logfile;
declare string $$dataset;

set $$dataset := doc("ComS363/Demos/Datasets/auctions.xml");
set $$prefix := $EngineA;
set $$query := for $b in ($$dataset)/site/regions/africa/item[@id="item3"]return $b/name/text();
set $$outfile := $$prefix_output.xml;
set $$logfile := demoBenchmark.xml;
```

**Output:**

```
<table>
<thead>
<tr>
<th>String variables</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>logfile</td>
<td>demoBenchmark.xml</td>
</tr>
<tr>
<td>query</td>
<td>for $b in (doc(&quot;ComS363/Demos/Datasets/auctions.xml&quot;))/site/regions/africa/item[@id=&quot;item3&quot;]return $b/name/text()</td>
</tr>
<tr>
<td>prefix</td>
<td>$EngineA</td>
</tr>
<tr>
<td>dataset</td>
<td>doc(&quot;ComS363/Demos/Datasets/auctions.xml&quot;)</td>
</tr>
<tr>
<td>outfile</td>
<td>$$prefix_output.xml</td>
</tr>
</tbody>
</table>
```
Then, the variables are substituted first and we could see the contents of each variable using “list command”. Later, the substituted command is forwarded for execution. The below are few possibilities:

```plaintext
createLog <TIME_LOG> $$logfile;

$$prefix $$query out>> $$outfile log time>> <EXECUTION_TIME> $$logfile;

$$prefix $$query out>> $$outfile log time>> <EXECUTION_TIME> demoBenchmark.xml;

$$prefix $$query out>> $$prefix_output.xml log time>> <EXECUTION_TIME> demoBenchmark.xml;
```

**Output:**

Each command’s output will be logged using an element. Each element will be collected & appended to the log file one after another. This help to preserve the structure of the experiment in an xml file as shown below.

```
<?xml version="1.0"?>
<TIME_LOG>
  <EXECUTION_TIME>49</EXECUTION_TIME>
  <EXECUTION_TIME>33</EXECUTION_TIME>
  <EXECUTION_TIME>28</EXECUTION_TIME>
</TIME_LOG>
```

Figure 3 demoBenchmark.xml file.
Figure 4 $\text{EngineA\_output.xml}$ file.

Also, if there are no variables to be substituted then the command will be directly send for execution to the respective client. For Example:

```plaintext
declare string $$xmldoc;
set $$xmldoc := doc("ComS363/Demos/Datasets/auctions.xml");
list variables;

// Create an XML-based log file to gather performance statistics.
createLog <root> benchmark_Engines_log.xml;

$\text{EngineA:}>$ for $b$ in ($$xmldoc)/site/regions/africa/item[@id="item3"] return $b/name/text()
out>> EngineA_out.xml
log time >> <EngineA_time>  benchmark_Engines_log.xml;

$\text{EngineB:}>$ for $b$ in ($$xmldoc)/site/regions/africa/item[@id="item3"] return $b/name/text()
out>> EngineB_out.xml
log time >> <EngineB_time>  benchmark_Engines_log.xml;

$\text{Saxon:}>$ <NS> { for $e$ in doc("ComS363/Demos/Datasets/Emp.xml")
where $e/DName/text() = "Toys"
return <E> { $e/Name, $e/Salary } </E> }
</NS>
out>> Saxon_out.xml
log time >> <Saxon_time>  benchmark_Engines_log.xml;
```

**Output:**

```plaintext
<?xml version="1.0"?>
<EXECUTION_TIME>
  <EngineA_time>457</EngineA_time>
  <EngineB_time>238</EngineB_time>
  <Saxon_time>21</Saxon_time>
</EXECUTION_TIME>
```

Figure 5 benchmark_Engines_log.xml
Moreover, there is no internal dependency required on “CyDB” prefix or “execute” command in order to run a command. For Example:

**Before:**

```xml
<?xml version="1.0"?>
<Root>poisons</Root>
```
After:

```xml
foreach $i in (1, 2, 3) log time >> <loop1 var="($prefix($i))"> benchmarkQ.xml
{
  // Inner loop to iterate through XMark queries;
  foreach $j in [1, 5] log time >> <loop2 var="Q($j)"> benchmarkQ.xml
  {
    // warm up 3 times;
    foreach $k in [1,3]
      {
        $prefix[$i] $query[$j];
      }
  }
  // execute each query 10 times and log performance;
  foreach $k in [1,3]
    {
      $prefix[$i] $query[$j] out >> ($prefix($i)_query($j).xml log time >> <query> benchmarkQ.xml;
    }
}
```

Every component in a CyDIW command now allows CyDIW variables. If a command does not use any CyDIW variables it is sent for execution immediately; otherwise values of variables are substituted and then the command is executed. The run command has been eliminated as now every CyDIW command has the functionality offered by the run command. Flat commands with variables and elimination of run command helps users simplify coding of experiments in the scripting language of CyDIW.

During testing of the new parser it has been identified that the different clients weren’t working as expected. These clients are either a database prototype or any external command based system. Later the clients which were studied are OOQUERY - a variant of SAXON client, R, SAXON, SCalc, SPLIT, mySQL. It has been identified that in older version of CyDIW these client’s adapters were parsing the commands. The adapter shouldn’t contain any parsing logic. Therefore, I have understood what each parser for
respective client was performing and removed the unnecessary parsing happening in the adapter. Each adapter defines a logic on How to execute a command which doesn’t require any parsing at all. Moreover, R client wasn’t working correctly in the older source code so, I have added the functionality and documented it for future use.
APPENDIX A. Revised language constructs and commands in CyDIW

**Basic CyDIW command format**

A command in CyDIW is a counterpart of an assignment statement in a programming language. Thus it plays a central role in composing experiments. CyDIW commands can use variables but the variable values must be substituted before a command is executed. This aspect will be discussed in Section 1.1; until then the reader may try to understand the concept of a CyDIW command assuming that no variables are present. A CyDIW command is as follows:

```
ClientPrefix:>
    ClientCommand [>>
    out textfilename]
    [log [time | custom] <XMLTag> XMLLogFileName]
```

- **clientPrefix** is the prefix of a client system registered in CyDIW’s SystemConfiguration.xml file.
- **clientCommand** is quite literally a command from a client system, possibly including parameters. Note the distinction between a client command and a CyDIW command. A client command is a part of a CyDIW command. But a CyDIW command can invoke several services surrounding the execution of a client command. These services cater to the needs of experiments.
- If the (text-based) output of the command are to be saved and not displayed in the output pane the out>> option can be used to redirect the output to files. The files do not have to pre-exist; they will be created on execution of the CyDIW command. These files are not to be confused with the files that a client command can produce on its execution.
- If logging is desired, the one of the log options time | custom can be used. A user-defined XML tag should be provided. The tag can use XML attributes to capture the
settings in the experiment. In case of log time the elapsed time between dispatching the command to the client system and receiving the response is logged under the XML tag. In case of log custom, a piece of text returned by the client system is logged. It is recommended that the returned text should be a sequence of XML elements. This is something that the adapter can handle.

The user provided XML tag will serve as the root of the logged entry. The XMLLogFileName file has to have been created beforehand.

- The semicolon is required at the end of the command.
- Internally “:>”, “>> out”, “logtime” and “logcustom” are treated as tokens by the CyDIW language compiler. The tokens “:” or “:>” are not considered part of client prefix.

1.1 Use of variables

Variables can be used anywhere in a CyDIW command: in clientPrefix, client-command, textfilename, XMLTag, and XMLLogFileName. If a command does not include any variables it is executed as is. If variables are included, their values are first substituted and then the command is executed. Use of variables makes CyDIW commands quite versatile. For example, the variable substitution can take place in a command appearing in the body of a loop that is possibly nested. This means different executions of a command can invoke different client systems, different client commands - with possibly different inputs, the results can be directed to different files, choosing different logging options (log time or log custom), different XMLTag to record current values of parameters in the experiment, and populating the log entries in different log files.
The run command has been eliminated from CyDIW as with the use of variables every CyDIW command potentially absorbs all the functionality offered by the run command.

2 Listing all commands

- ListCommands;

This command is intended to list all CyDB native commands. A list of commands is maintained as a text file. This command simply lists the contents of that file. As more commands are added, modified, or deleted this list should be updated. When that does not happen this command becomes unreliable.

2.1 For loop

- foreach <Variable> in 
  (<ExpressionsList>) [log
  <logtime|logcustom>
  <LoopHeaderTag> <XMLFileName>]

This is now a language construct and no longer a command. Hence, the prefix “CyDB” should no longer be used. The header of the loop deposits a user defined LoopHeaderTag in the XMLFileName benchmark file when the loop is entered. This helps the XML-based log to reflect the structure of an experiment. Loops can be nested.

2.2 While loop

- while (condition) {...} [logtime]logcustom>> <LoopHeaderTag> <XMLFileName>

2.3 If condition

- if (condition) {...}

2.4 If-else condition

- if (condition) {...} else {stmt;}

2.5 Logical operations

- AND & OR

2.6 Support for logging

- createlog <RootTag> <XMLFileName>;

The named XML file is created with the user defined root tag. The initial contents of the file is <RootTag> ~ </Root-Tag>. A cursor “~” is placed at the end of the file. The log entries are deposited just preceding the cursor. Note that the contents of the log file
can be printed at any time using displayXML command.

2.7 Variables creation, assignment, destruction
- declare <int|string|int[]|string[]> <Variable> [:= <Value>]; 
- undeclare <int|string|int[]|string[]> <Variable>
- list variables;
- set <Variable> := <AssigningValues>;
- execute <Variable>;

2.8 Displaying formatted files
- displayTXT <FILENAME>:
  Displays a file in the default text editor. The file name can have any extension or no extension. It should internally be a text file.

- displayPDF <FILENAME>:
  displays file in the default PDF viewer.
- displayHTML <FILENAME>:
  display file in the default browser. The file name can have any extension or no extension. It should internally be a legal html file.

- displayXML <FILENAME>:
  An XML file is displayed in an intuitive lightweight viewer that allows elements to be expended / collapsed. We need to know who this artifact belongs to. It needs acknowledgment.

- displayJSON <<FILENAME>>;
  display JSON file in a text editor. The file name can have any extension or no extension. It should internally be a legal JSON file.

2.9 Text file manipulation
These commands allow operations on text files. The <FileName> parameter stands for name of a text file. The extension .txt is not required. Note that html and xml files are also text files. No pre-defined expectations are associated with contents of these files except that they are text files according to Windows, Mac, or Linux conventions.

- CreateTextFile <FileName>;
Creates an empty text file.

- **OpenTextFile <FileName>;**
  Opens a text file for further operations in CyDIW.

- **CloseTextFile <FileName>;**
  Closes a text file; no further operations can be performed on the file.

- **append <FileName> <String>;**
  Appends the String to the end of an open File.

- **appendln <FileName> <String>;**
  Appends the String and a end of line character to an open File.

- **appendfile <FileName1> <FileName2>;**
  Appends file FileName2 at the end of an open file FileName1. File FileName2 need not be opened.

**2.10 Discontinued commands**

- **run**
  The capability of this command is being absorbed in the most basic CyDIW commands. We have changed the meaning of the command now.

- **execute**
  Observations similar to run command.

- **Pause**
  It's never been defined

- **operationslog**
  It's never been defined
APPENDIX B. Importance of CyDIW over General scripting language.

Generally, practitioners who want to perform some benchmarks write their own scripts in some scripting languages such as python or UNIX shell scripting or any general purpose language. CyDIW is designed for a very specific purpose of conducting re-producible benchmarking experiments. In my knowledge, whenever a software package is designed, for example, MongoDB, MySQL, or even any general purpose systems such R. The developers especially have to provide language support for the well-known languages using APIs so that they can use the functionality of these packages either for building some applications on top of it or to perform some benchmarking. If they don’t then it’s the headache of the user to create an adapter to use it. In CyDIW, it’s up to the user who want to benchmark different systems or even want to use certain packages for visualization purpose or more. They could write an adapter, or directly use external client’s using “OS” client and create comprehensive experiments. For Example, the below experiment:
Figure 9 a Benchmarking experiment.

```java
// Step 1. Setup variables and a log file;
declare string[4] $prefix;
declare string[21] $query;

// Create an XML-based log file to gather performance statistics;
createLog <root> benchmarkQ.xml;</n
// Assign Query Engines to array variables;

// Load the Datasets
set $xmlDoc := doc("Com5363/Demos/Datasets/auctions.xml");

// Assigning queries to variables.
set $query1 := for $b in ($xmlDoc/site/regions/africa/item[@id="item1"]
    return $b/name/text();
...
set $query20 := let $auction := ($xmlDoc)
    return
    <result>
      <preferred>
        { count ($auction/site/people/person/profile[@income >= 100000]) }
      </preferred>
      <standard>
        { count ($auction/site/people/person/profile[@income < 100000 and @income >= 30000]) }
      </standard>
      <challenge>
        { count ($auction/site/people/person/profile[@income < 30000]) }<challenge>
      </challenge>
    </result>
</query>
```

Figure 10 a Benchmarking experiment continued.

```java
// Query execution and logging of benchmark statistics
foreach $i in (1, 2, 3) log time >> <loop1 var="($prefix[$i])"> benchmarkQ.xml
    // Inner loop to iterate through XMark queries;
    foreach $j in [1, 5] log time >> <loop2 var="Q($j)"> benchmarkQ.xml
    // warm up 3 times;
    foreach $k in [1,3]
    {
      $prefix[$i] $query[$j];
    }

    // execute each query 10 times and log performance;
    foreach $k in [1,3]
    {
      $prefix[$i] $query[$j] out >> ($prefix[$i]*)_query($j).xml log time >> <query> benchmarkQ.xml;
    }
}
```

// Step 4. Display the log file and plot performance as a graph using R;
displayFile CyWorkspace\Enginee\experimentData.xml;
$R: R CMD BATCH CyWorkspace\R\R_code.txt;
displayPDF CyWorkspace/plots.pdf;
The resulting log time stored in an XML file. Now, we can use R to plot some visualization. You could use below 2 conventions:

**With the adapter:**

```bash
$R:> CMD BATCH CyWorkspace\R\R_code.txt;
```

**Or without an adapter using OS client.**

```bash
$OS:> R CMD BATCH CyWorkspace\R\R_code.txt;
```

![CyDW Experimental Results](image)

**Figure 12 Result.**

Figure 11 benchmarkQ.xml
Another good example would be using a python client in CyDIW. For Example:

Figure 12 Python Client usage within a the CyDIW GUI.
Therefore, the user doesn’t have the compulsion of creating an adapter and run different system in CyDIW GUI with any mix & match to perform experiments.