Definitive HPCC Systems

JANUARY 2023

Day 1 Workshop:
Data Ingest, Evaluation, Profiling

Richard Taylor/Bob Foreman
Senior Software Engineers

LexisNexis RISK Solutions

Workshop Agenda

This workshop are based on the new book by Richard Taylor, and our core ECL Introduction to ECL and Advanced ECL training courses:

Definitive HPCC Systems

Volume II: Data Transformation and Delivery

Days 1 and 2: Chapters 1-3

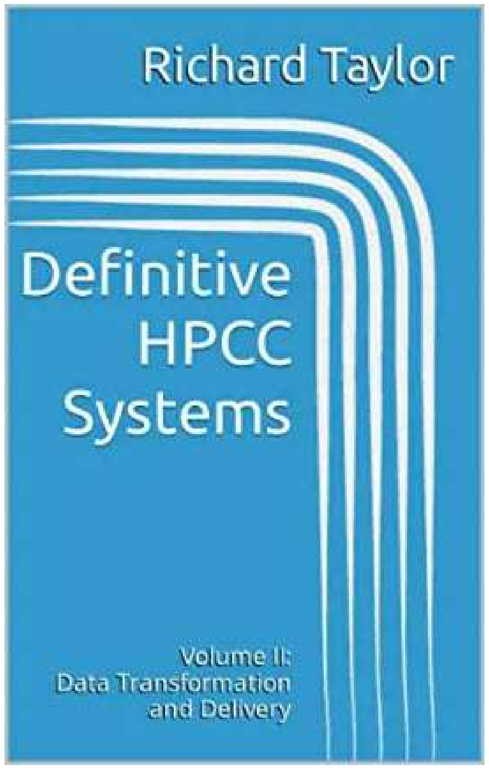
Day 3: Chapter 4

Day 4: Chapter 5 (Selections from ECL Cookbook)

Volume I and II is currently available on Amazon!

https://www.amazon.com/Definitive-HPCC-Systems-Overview-Platform-ebook/dp/B087Y1FMDH

https://www.amazon.com/Definitive-HPCC-Systems-Transformation-Delivery-ebook/dp/B0BCMZCXDD



HPCC Systems:

End to End Data Lake Management

Completely free
open source data lake solution

Out of the box capabilities for consistency and
ease of use

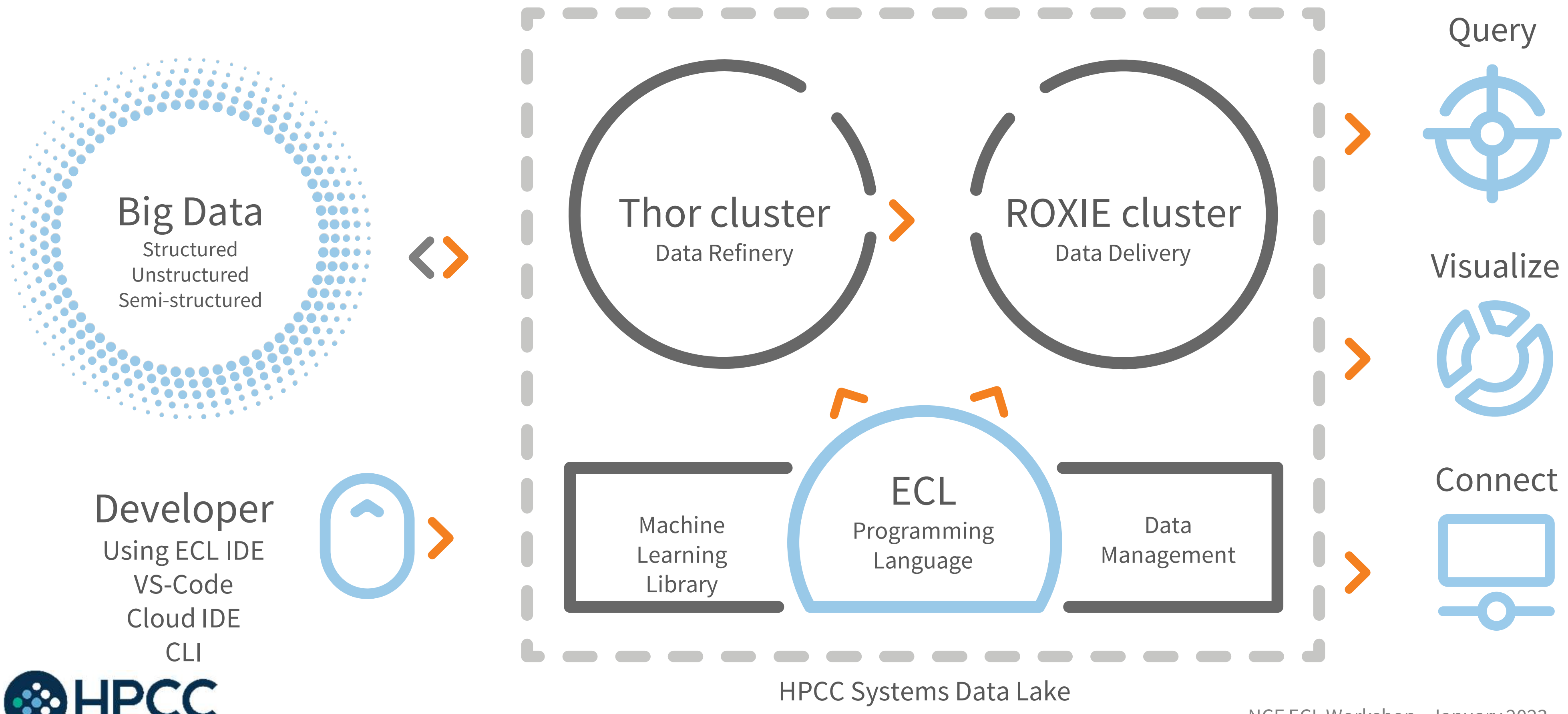
Less coding
and more using (even though we love to code)

We are your one stop shop for all your data integration, querying and analytical needs

10 YEARS HPCC SYSTEMS® OPEN SOURCE

HPCC SYSTEMS

The HPCC Systems Components



Technology — The Open Source Stack

Thor: Data Refinery Cluster
Extraction, loading, cleansing, transforming, linking and indexing

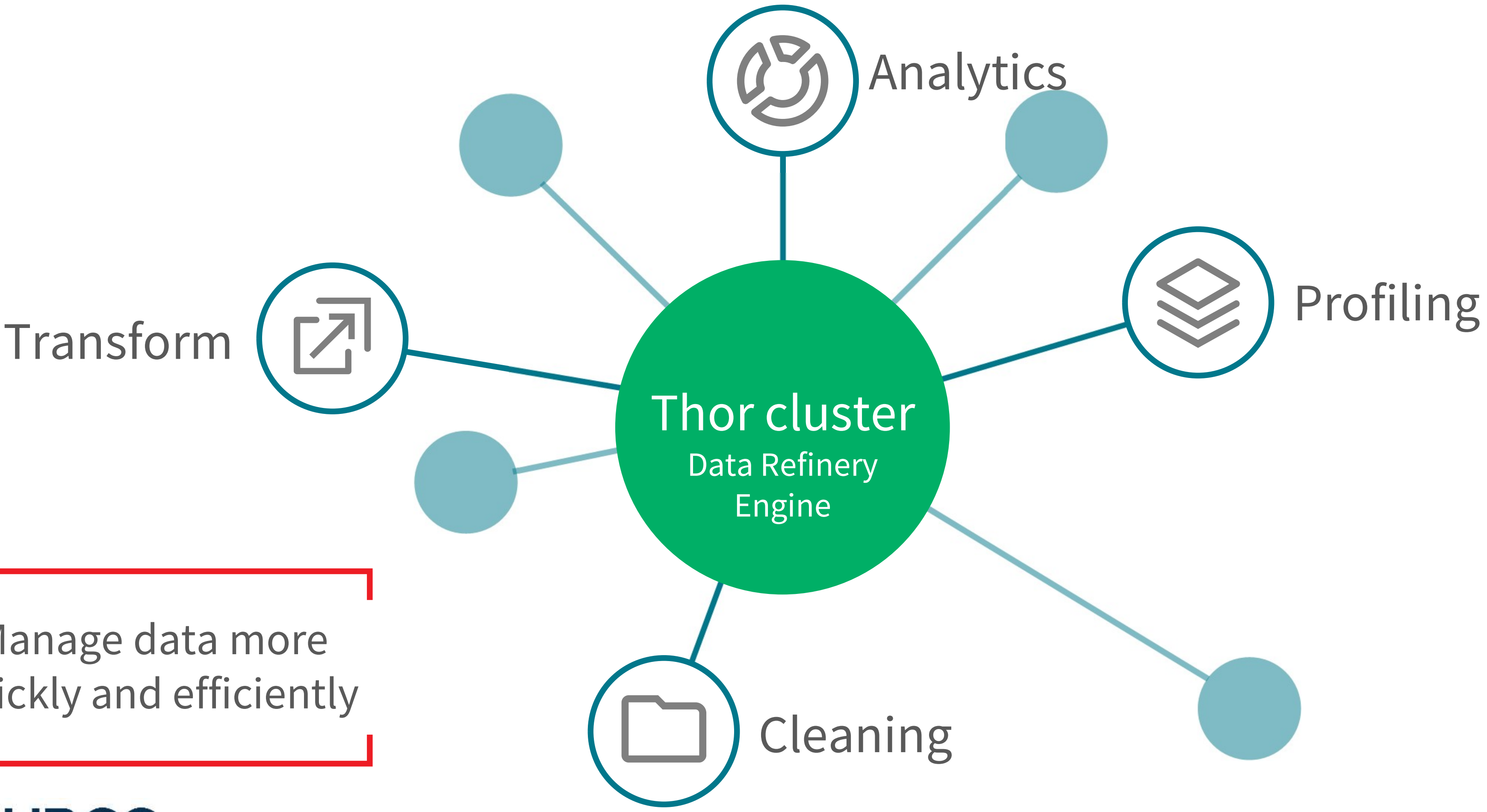
ROXIE: Data Delivery Engine
Rapid data delivery cluster with high-performance online query delivery for big data

Data Management Tools
Data profiling, cleansing, snapshot data updates, consolidation, job scheduling and automation

Machine Learning Library
Linear regression, logistic regression, decision trees and random forests

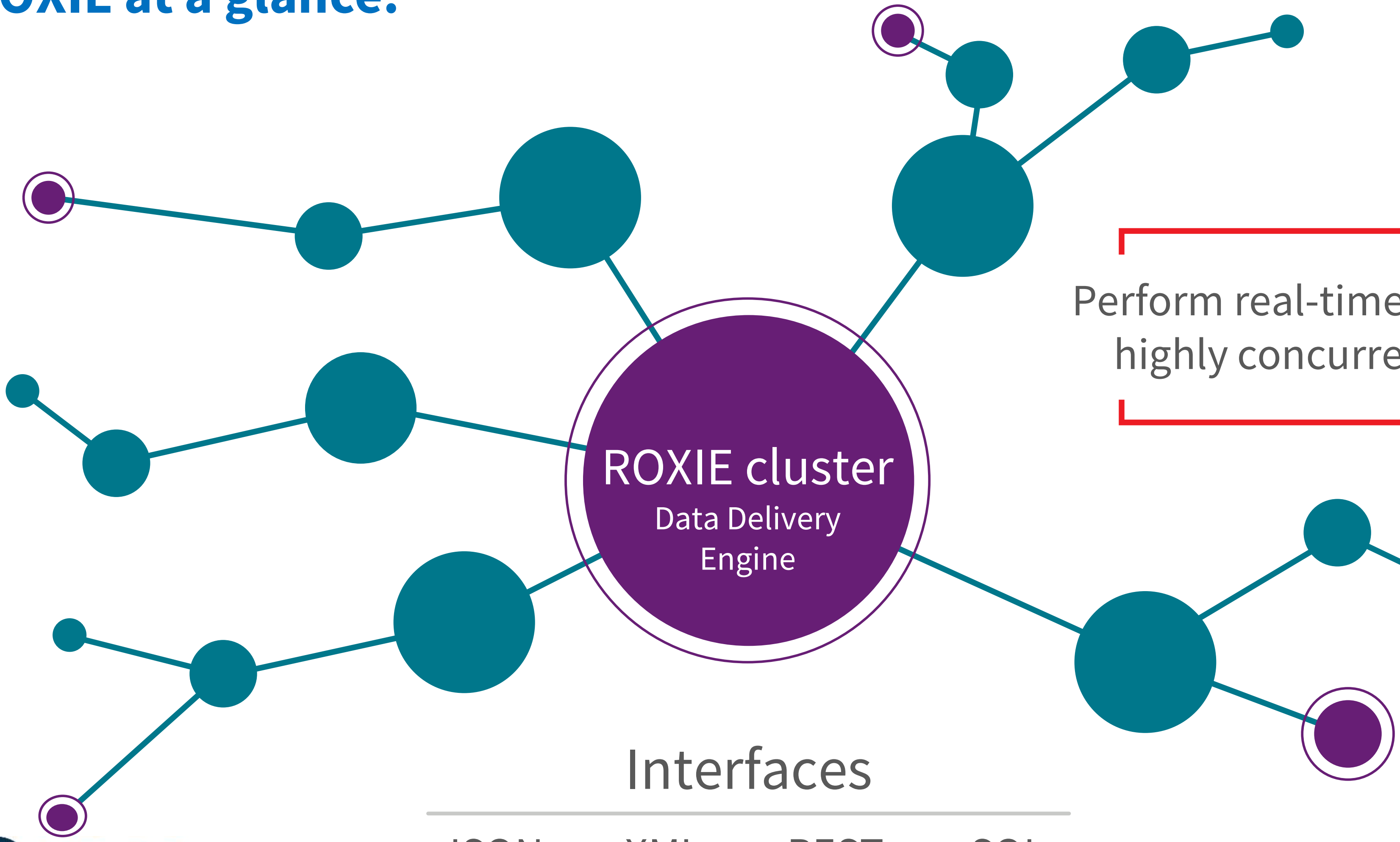
Connectivity & Third-Party Tools
New plugins to help integrate third party tools with the HPCC Systems platform

THOR at a glance:



Manage data more quickly and efficiently

ROXIE at a glance:



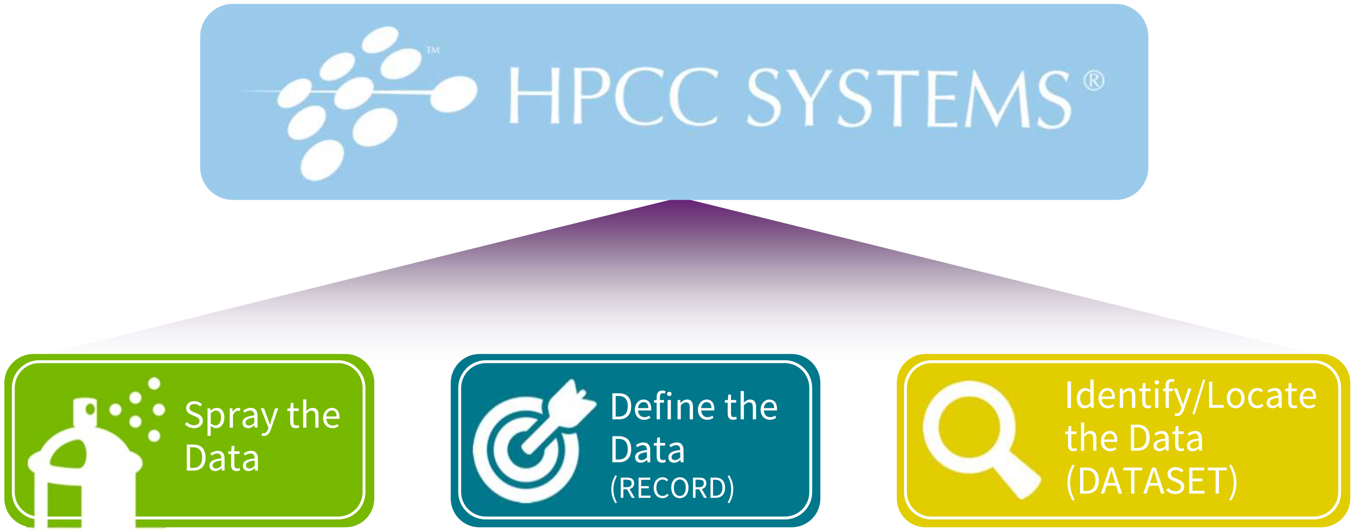
Perform real-time queries with highly concurrent delivery

Interfaces

JSON XML REST SQL

Three ECL Data Rules

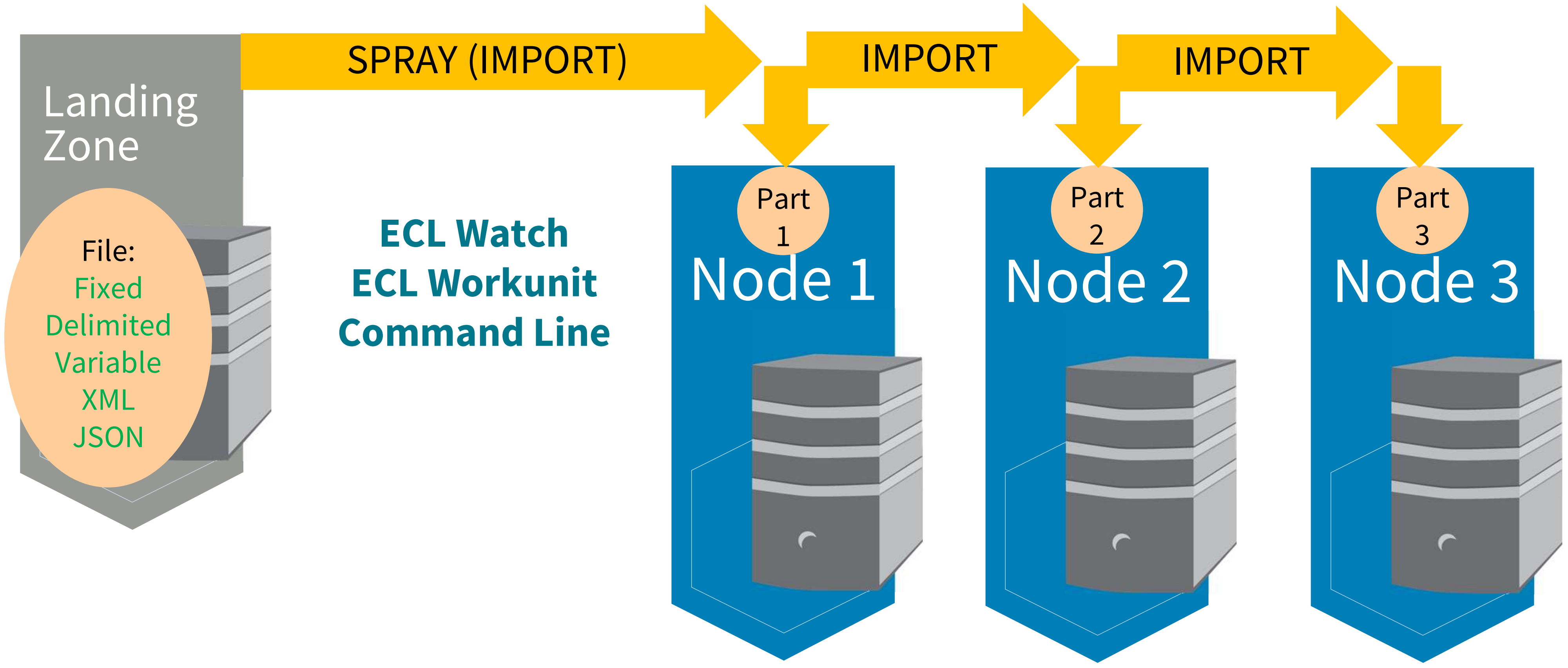
Before you begin to work on any data in the HPCC cluster, you must always do three things:



Data Ingest

- The term "Spray" (or "Import" in ECL Watch 9 and greater) is used to describe the process of copying data from external files into an HPCC Systems cluster. This term is appropriate because the HPCC Systems environment always works with distributed data files.
- So, to get a single physical data file into the cluster, the spray/import process divides the data into *n* chunks (where *n* is the number of nodes in the cluster) and puts one file part (approximately evenly sized) on each node.
- Before any file can be imported, it must first be in a location that is accessible to the cluster. That location is commonly referred to as a Landing Zone or Drop Zone – another middleware component described in the first volume of this book series.

SPRAY(Import) Operation

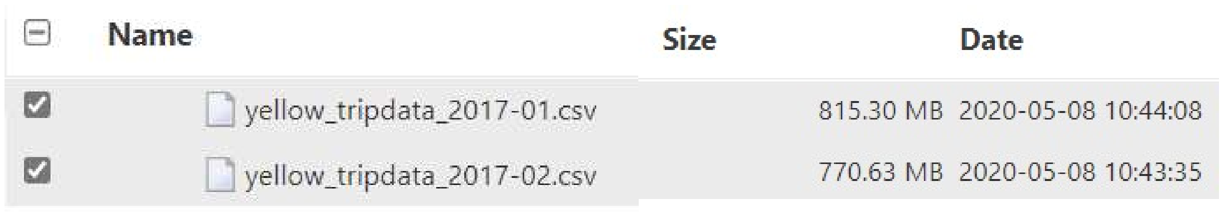


ECL Watch
ECL Workunit
Command Line

Referenced in ECL as a single logical file…

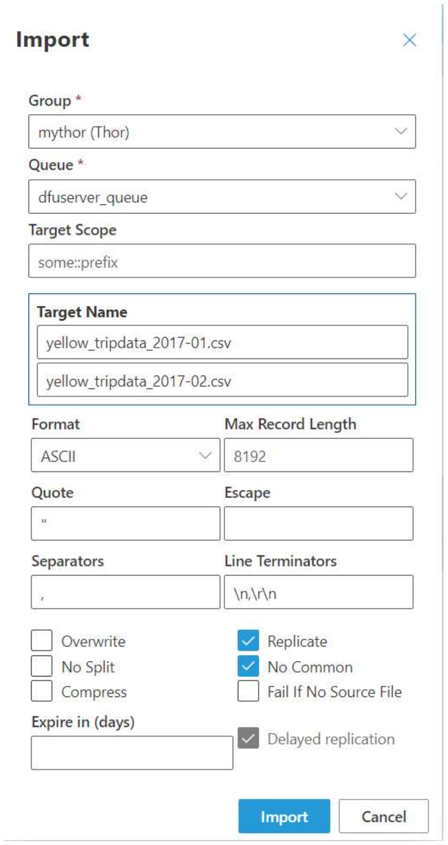
Workshop Demo Data

- We'll begin with getting some publicly available data to work with. The **New York City Taxi & Limousine Commission** makes its trip data freely available to everyone here: https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page
- Before any file can be sprayed, it must first be in a location that is accessible to the cluster. That location is commonly referred to as a Landing Zone or Drop Zone – another middleware component described in the first volume of this book series.
- After locating the files that we need to spray on our workshop cluster Landing Zone, we will use a Delimited Spray to move these files to the cluster.



	Name	Size	Date
☑	yellow_tripdata_2017-01.csv	815.30 MB	2020-05-08 10:44:08
☑	yellow_tripdata_2017-02.csv	770.63 MB	2020-05-08 10:43:35

Delimited Spray Options



Import

Group *
mythor (Thor)

Queue *
dfuserver_queue

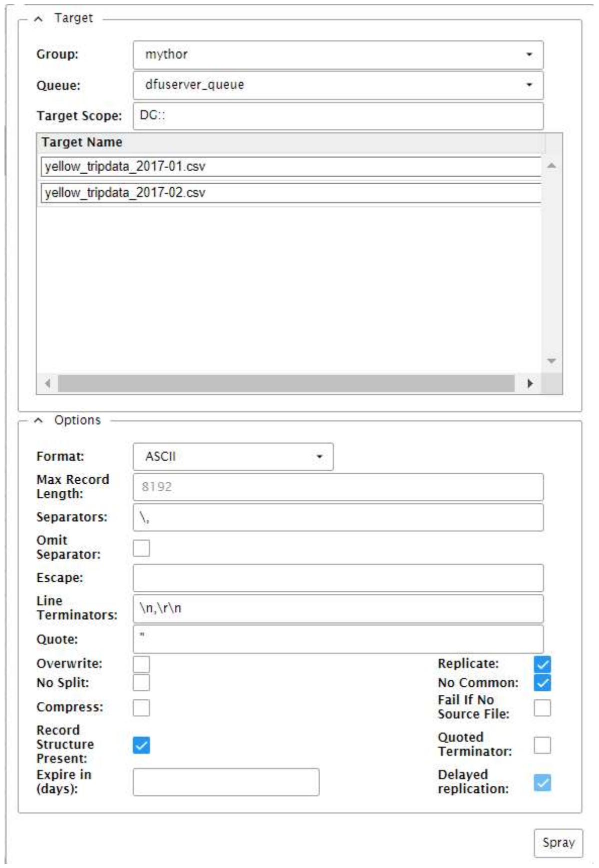
Target Scope
some::prefix

Target Name
yellow_tripdata_2017-01.csv
yellow_tripdata_2017-02.csv

Format: ASCII
Max Record Length: 8192
Quote: "
Escape:
Separators: ,
Line Terminators: \n,\r\n

- [ ] Overwrite
- [ ] No Split
- [ ] Compress
- [x] Replicate
- [x] No Common
- [ ] Fail If No Source File
- [x] Delayed replication

Expire in (days):



Target

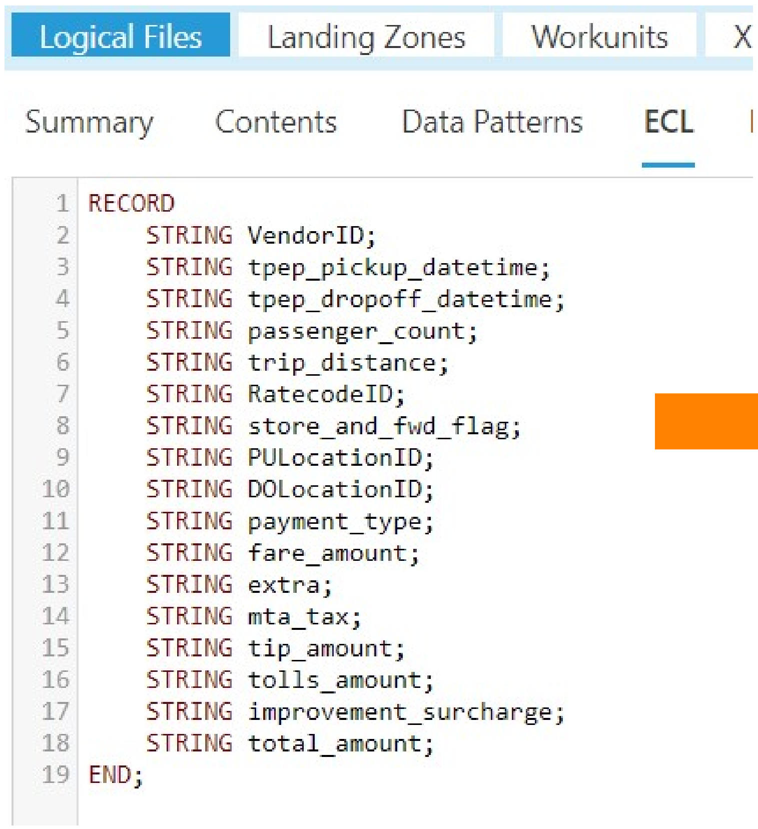
Group: mythor
Queue: dfuserver_queue
Target Scope: DC::

Target Name
yellow_tripdata_2017-01.csv
yellow_tripdata_2017-02.csv

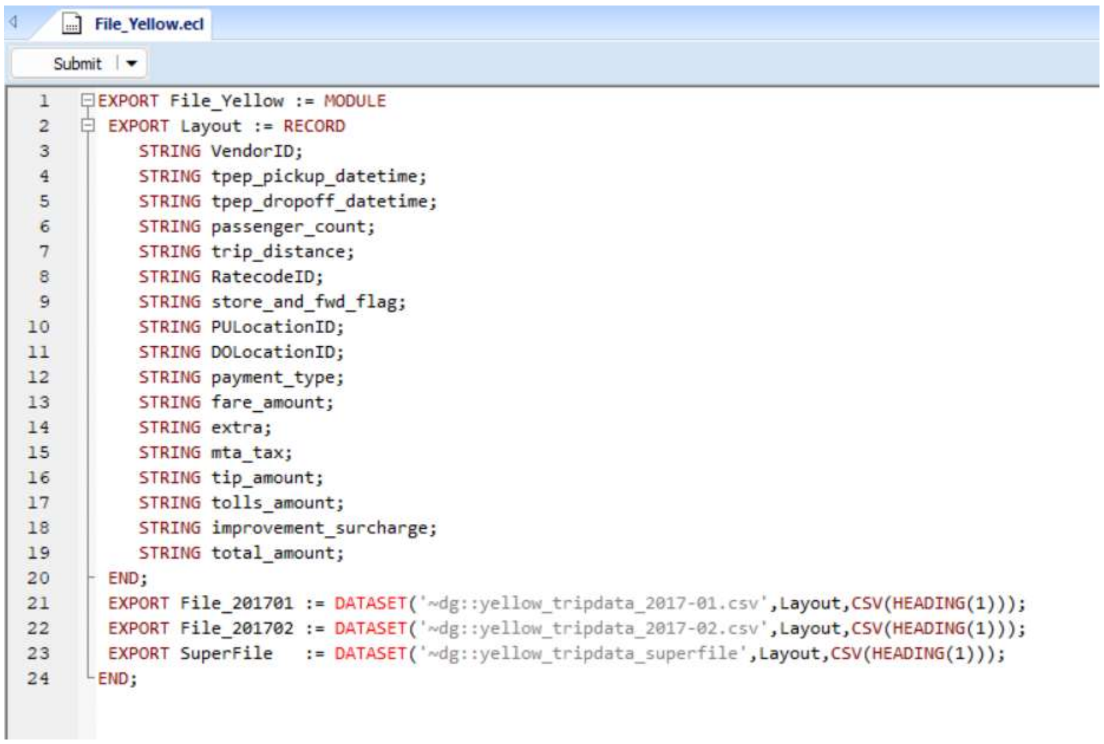
Options

Format: ASCII
Max Record Length: 8192
Separators: \,
Omit Separator: [ ]
Escape:
Line Terminators: \n,\r\n
Quote: "
Overwrite: [ ]
No Split: [ ]
Compress: [ ]
Record Structure Present: [x]
Expire in (days):
Replicate: [x]
No Common: [x]
Fail If No Source File: [ ]
Quoted Terminator: [ ]
Delayed replication: [x]

Get the RECORD, create the DATASET:



```
RECORD
    STRING VendorID;
    STRING tpep_pickup_datetime;
    STRING tpep_dropoff_datetime;
    STRING passenger_count;
    STRING trip_distance;
    STRING RatecodeID;
    STRING store_and_fwd_flag;
    STRING PULocationID;
    STRING DOLocationID;
    STRING payment_type;
    STRING fare_amount;
    STRING extra;
    STRING mta_tax;
    STRING tip_amount;
    STRING tolls_amount;
    STRING improvement_surcharge;
    STRING total_amount;
END;
```

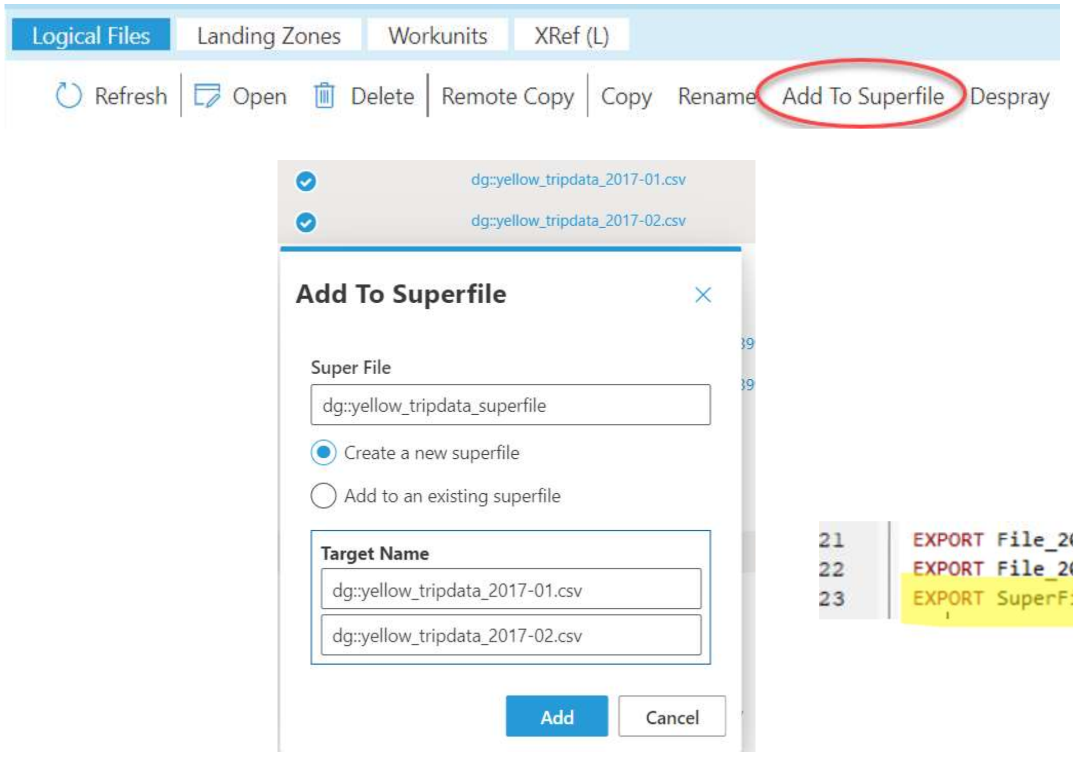


```
EXPORT File_Yellow := MODULE
 EXPORT Layout := RECORD
    STRING VendorID;
    STRING tpep_pickup_datetime;
    STRING tpep_dropoff_datetime;
    STRING passenger_count;
    STRING trip_distance;
    STRING RatecodeID;
    STRING store_and_fwd_flag;
    STRING PULocationID;
    STRING DOLocationID;
    STRING payment_type;
    STRING fare_amount;
    STRING extra;
    STRING mta_tax;
    STRING tip_amount;
    STRING tolls_amount;
    STRING improvement_surcharge;
    STRING total_amount;
 END;
 EXPORT File_201701 := DATASET('~dg::yellow_tripdata_2017-01.csv',Layout,CSV(HEADING(1)));
 EXPORT File_201702 := DATASET('~dg::yellow_tripdata_2017-02.csv',Layout,CSV(HEADING(1)));
 EXPORT SuperFile   := DATASET('~dg::yellow_tripdata_superfile',Layout,CSV(HEADING(1)));
END;
```

Combining Common Data

- Given that we have two separate logical files that both have the same structure, and that we're working with a Big Data platform, then it would be advantageous to be able to work with both as if they were a single logical file instead of two. You could simply use the ECL record set append operators (+ and &) to combine them, but the better way is to define them as sub-files in a single SuperFile.
- The SuperFiles section in the *Standard Library Reference* documents all the functions that are available for SuperFile maintenance. Also, the **Working With SuperFiles** section of the *Programmer's Guide* contains several articles that describe how to use those functions for standard Superfile maintenance processes.

Using the ECL Watch to Create a Superfile:



dg::yellow_tripdata_2017-01.csv

dg::yellow_tripdata_2017-02.csv

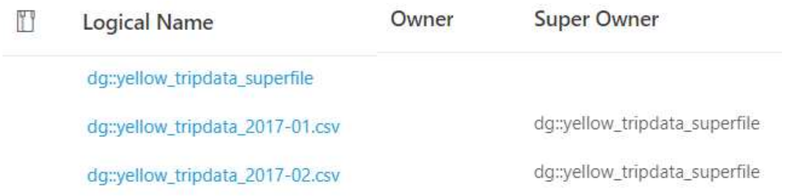
Add To Superfile

Super File: dg::yellow_tripdata_superfile

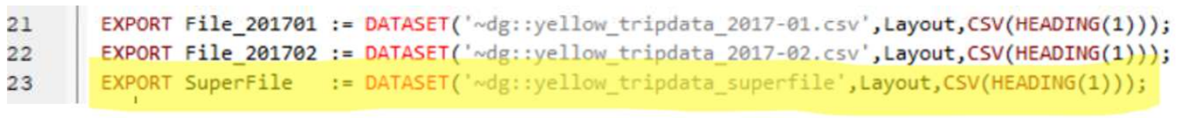
[✓ Create a new superfile]

Add to an existing superfile

Target Name
dg::yellow_tripdata_2017-01.csv
dg::yellow_tripdata_2017-02.csv



Logical Name	Owner	Super Owner
dg::yellow_tripdata_superfile		
dg::yellow_tripdata_2017-01.csv		dg::yellow_tripdata_superfile
dg::yellow_tripdata_2017-02.csv		dg::yellow_tripdata_superfile

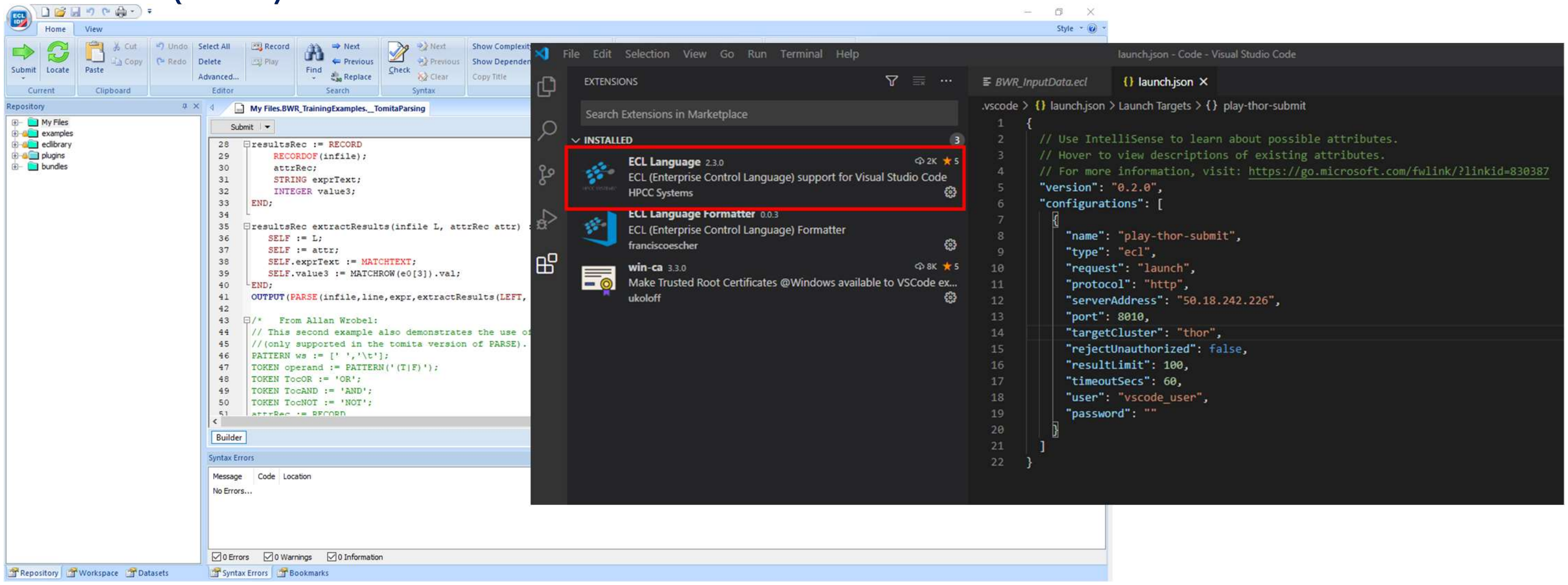


```
EXPORT File_201701 := DATASET('~dg::yellow_tripdata_2017-01.csv',Layout,CSV(HEADING(1)));
EXPORT File_201702 := DATASET('~dg::yellow_tripdata_2017-02.csv',Layout,CSV(HEADING(1)));
EXPORT SuperFile   := DATASET('~dg::yellow_tripdata_superfile',Layout,CSV(HEADING(1)));
```

Integrated Development Environments

ECL IDE (Win)

Visual Studio Code (Ux/MacOS)



No Errors...

ECL Language 2.3.0 — 2K ★ 5
ECL (Enterprise Control Language) support for Visual Studio Code
HPCC Systems

ECL Language Formatter 0.0.3
ECL (Enterprise Control Language) Formatter
franciscoescher

win-ca 3.3.0 — 8K ★ 5
Make Trusted Root Certificates @Windows available to VSCode ex...
ukoloff

```
resultsRec := RECORD
    RECORDOF(infile);
    attrRec;
    STRING exprText;
    INTEGER value3;
END;

resultsRec extractResults(infile L, attrRec attr) :
   SELF := L;
   SELF := attr;
   SELF.exprText := MATCHTEXT;
   SELF.value3 := MATCHROW(e0[3]).val;
END;
OUTPUT(PARSE(infile,line,expr,extractResults(LEFT,

/*   From Allan Wrobel:
// This second example also demonstrates the use of
//(only supported in the tomita version of PARSE).
PATTERN ws := [' ','\t'];
TOKEN operand := PATTERN('(T|F)');
TOKEN TocOR := 'OR';
TOKEN TocAND := 'AND';
TOKEN TocNOT := 'NOT';
attrRec := RECORD
```

```
{
  // Use IntelliSense to learn about possible attributes.
  // Hover to view descriptions of existing attributes.
  // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
  "version": "0.2.0",
  "configurations": [
    {
      "name": "play-thor-submit",
      "type": "ecl",
      "request": "launch",
      "protocol": "http",
      "serverAddress": "50.18.242.226",
      "port": 8010,
      "targetCluster": "thor",
      "rejectUnauthorized": false,
      "resultLimit": 100,
      "timeoutSecs": 60,
      "user": "vscode_user",
      "password": ""
    }
  ]
}
```

And CLI too! ECL.EXE

Data Profiling

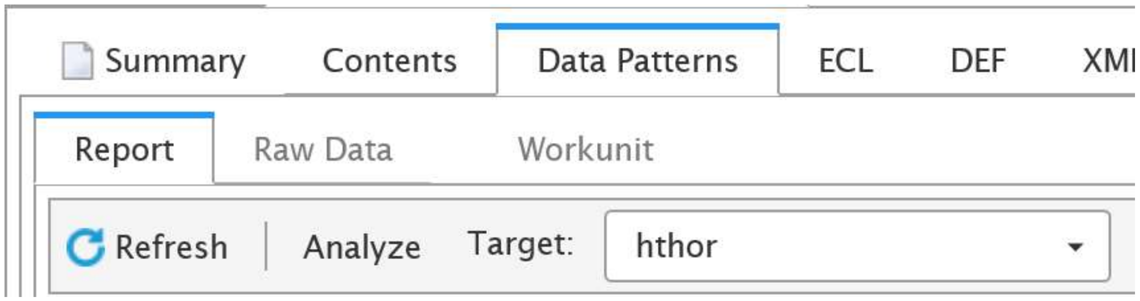
- Now that we have data in the HPCC Systems environment and have defined it for use, the next step is to explore the data to discover what's what. Whether you’re doing standard ETL processing, or any other data work, you need to completely understand the data you're working with for two primary reasons:

1. So you can craft the best possible data structures for your end result (product) database.
2. To make your data transformation processes (from raw data to final product) as efficient as possible.

So, the first step in any data ingest process, once the files are available on your cluster, is to Profile the data. This section discusses several possible ways to accomplish that task.

Built-In Data Profiling: Data Patterns

An extensive Data Profiling report is now built-in and available in the ECL Watch for all logical files. This report can be accessed via the Data Patterns tab:



There are three ways to use Data Patterns:

1. ECL Watch (via the Data Patterns tab)
2. Bundle (found on the HPCC Git Hub): https://github.com/hpcc-systems/DataPatterns.git
3. Standard Library Reference (STD.DataPatterns)

Data Profiling Questions

1. Are there any non-numeric characters in the field (IOW, is it text data or just numbers)?
2. If it is text, what is the maximum text length?
3. If it is numeric, are the values integers or floating point?
4. If it is numeric, what is the range of values?
5. How many unique values are present?
6. What do the data patterns look like?
7. How skewed are the values, and how sparsely populated?

BWR_Profile0.ecl

Profiling Every Field (Issue 1)

- So, because we started our profiling code with the first two re-definitions, you could just change the *Fld* definition's expression to name a different field and re-run the code. That would work, but that means the information would be in a separate workunit for each field, and each question's answer would show up in a separate result tab for the workunit -- so the display wouldn't be terribly "user-friendly" (especially since you are the "user" of this information).
- Let's solve the first issue: the fact that all the answers show up on separate result tabs. We can do this by writing a FUNCTION structure to contain all the answer code and produce all the results on a single tab.

Profile.ecl

Profiling *Every* Field (Issue 2)

- The second issue of automating our profiling to process every field in our dataset is accomplished using Template Language.
- Template language is a meta-language used to generate ECL code.
- Unlike ECL, the Template Language has variables (referred to as "symbols") that must be explicitly declared (with some few exceptions) and can be re-assigned values. It is also procedural, meaning it does have looping constructs and requires programming logic more similar to other procedural languages than to ECL.
- We can make use of that feature to automate ECL code generation to produce our Profile function results as a single dataset from every field in our CSV file.

BWR_Profile2.ecl

Profile Automation

- So, you could just change the previous code to run it on a different dataset. Or you can modify that code and wrap it in either a MACRO or a FUNCTIONMACRO structure. That would give you a tool that you can just call, passing it an argument naming the dataset to profile.
- Like the Template Language, both the MACRO and FUNCTIONMACRO are code generation tools.

fnMAC_Profile.ecl

Testing:

```
OUTPUT($.fnMAC_Profile($.File_Yellow.SuperFile),,'~File_Yellow::Profile::SuperFile_' +
                       (STRING8)Std.Date.Today(),NAMED('ProfileInfo'), OVERWRITE);
```

Comparing Profiles

- If you periodically receive new files to add to the collection you already have, then it's a really good idea to re-run your profile code on the new file (alone) to see if there are any significant differences in the data that might require changes to your processing code or final product data format to handle. That's what we'll tackle now.
- Let’s run our Profiling macro on both input files, and compare:

BWR_ProfileCompare.ecl

Lab Exercises

Exercises 1 and 2 – Spray and Define Your Data

- RECORD and DATASET
- Syntax Checking
- Running a Test Query
- Output a Recordset
- Looking at Raw Data

CrossTab Reports

- **CrossTab = Cross Tabulation**
- **Data Statistics**

Use of:

- TABLE Function RECORD Structures
- TABLE Function
- GROUP keyword

Basic Actions

OUTPUT

[*name* :=] **OUTPUT(***recordset* [,*format*] [,*file* [,OVERWRITE]]**)**

- *name* – Optional definition name for this action
- *recordset* – The set of records to process
- *format* – The format of the output records: a previously defined RECORD structure, or an "on-the-fly" record layout enclosed in { } braces.
- *file* – Optional name of file to write the records to. If omitted, formatted data stream returns to the command line or ECL IDE program.
- OVERWRITE – Allows file to be overwritten if it exists

The **OUTPUT** action writes the *recordset* to the specified *file* in the specified *format*.

OUTPUT Examples:

```
OUTPUT(File_Accounts.File); //Equivalent to: File_Accounts.File;

OUTPUT(Persons,{FirstName,LastName}, NAMED('Names_Only'));

OUTPUT(MyRecordset,,'~CLASS::BMF::NewData', OVERWRITE);

//THOR is default format, but you can also output to:

OUTPUT(MyRecordset,,'~CLASS::BMF::NewData', CSV);

OUTPUT(MyRecordset,,'~CLASS::BMF::NewData', XML);

OUTPUT(MyRecordset,,'~CLASS::BMF::NewData', JSON);
```

HPCC SYSTEMS

Aggregate Functions - COUNT

COUNT(*recordset***)**

COUNT(*valuelist***)**

- *recordset* – The set or set of records to process.
- *valuelist* – *A comma-delimited list of expressions to count. This may also be a SET of values.*

The **COUNT** function returns the number of records in the specified *recordset*.

```
COUNT(Person(per_state IN ['FL','NY']));
TradeCount := COUNT(Trades);
```

Aggregate Functions - MAX

MAX(*recordset , value* **)**

MAX(*valuelist***)**

- *recordset* – The set or set of records to process.
- *value* – The field or expression to find the maximum value of.
- *valuelist - A comma-delimited list of expressions to find the maximum value of. This may also be a SET of values*

The **MAX** function returns the maximum value of the specified *field* from the specified *recordset*. Returns 0 if the *recordset* is empty.

```
MaxBal := MAX(Trades, Trades.trd_bal);
```

Aggregate Functions - MIN

MIN(*recordset , value* **)**

MIN(*valuelist***)**

- *recordset* – The set or set of records to process.
- *value* – The field or expression to find the minimum value of.
- *valuelist - A comma-delimited list of expressions to find the minimum value of. This may also be a SET of values*

The **MIN** function returns the minimum value of the specified *field* from the specified *recordset*. Returns 0 if the *recordset* is empty.

```
MinBal := MIN(Trades, Trades.trd_bal);
```

Aggregate Functions - SUM

SUM(*recordset , value* **)**

SUM(*valuelist***)**

- *recordset* – The set or set of records to process.
- *value* – The expression of field in the *recordset* to sum.
- *valuelist* - *A comma-delimited list of expressions to find the sum of. This may also be a SET of values*

The **SUM** function returns the additive sum of the value contained in the specified *field* for each record of the *recordset*. Returns 0 if the *recordset* is empty.

```
SumBal := SUM(Trades, Trades.trd_bal);
```

Aggregate Functions - AVE

AVE(*recordset , value* **)**

AVE(*valuelist***)**

- *recordset* – The set or set of records to process.
- *value* – The field or expression to find the average value of.
- *valuelist - A comma-delimited list of expressions to find the average value of. This may also be a SET of values*

The **AVE** function returns the average value (arithmetic mean) of the specified *field* from the specified *recordset*. Returns 0 if the *recordset* is empty.

```
AvgBal := AVE(Trades, Trades.trd_bal);
```

Builder Window Runnable Files (BWR)

- Stored in Repository – Ready to Run!
- Rules:
  - File must have at least one action
  - No EXPORT or SHARED
  - References to other definitions must be fully qualified:

```
IMPORT $;
Persons := $.File_Persons.File;
Accounts := $.File_Accounts.File;
COUNT(Persons);
COUNT(Accounts);
OUTPUT(Persons,{ID,LastName,FirstName});
```

TABLE Function:

TABLE(*recordset, format* **[,***expression* **[,FEW|MANY] [, UNSORTED]] [,LOCAL][,KEYED])**

- *recordset* – The set of records to process.
- *format* – The RECORD structure of the output records.
- *expression* – The "group by" clause for crosstab reports. Multiple comma-delimited *expressions* create one logical "group by" clause.
- FEW – Indicates that the *expression* will result in fewer than 10,000 distinct groups.
- MANY – Indicates that the *expression* will result in many distinct groups.
- UNSORTED – Indicates you don't care about the order of the groups.
- LOCAL – Specifies independent node operation.
- KEYED - Specifies activity is part of an index read operation

The **TABLE** function is similar to OUTPUT, but instead of writing records to a file, it outputs those records into a new memory table (a new dataset in the supercomputer). The new table inherits any implicit relationality the *recordset* has unless an *expression* is present. The new table is temporary, and exists only while the query is running.

TABLE example (vertical slice):

```
//"vertical slice" TABLE:
Layout_Name_State := RECORD
    Persons.LastName;
    Persons.FirstName;
    Persons.State;
    END;
Per_Name_State := TABLE(Persons, Layout_Name_State);
```

GROUP Keyword (in TABLE :

The **GROUP** keyword replaces the *recordset* parameter of any aggregate function used in the record structure of a TABLE definition where a group by *expression* is present.

This is only used to generate a crosstab report (set of statistics) on a recordset. There is also a GROUP function which provides similar functionality.

```
// Create a crosstab report for each sex in each state
R := RECORD
  Persons.State;
  Persons.Gender;
  COUNT(GROUP);
  COUNT(GROUP,Persons.Gender = 'M') //Scoping count
END;
CTOut := TABLE(Persons,R, State, Gender);
```

TABLE example (Crosstab):

```
//"crosstab report" TABLE:
Layout_Per_State := RECORD
    Person.per_st;
    StateCount := COUNT(GROUP);
    END;

    Per_Stat := TABLE(Person,  Layout_Per_State, per_st);
```

String functions: LENGTH

LENGTH(*expression***)**

- *expression* – A string expression.

The **LENGTH** function returns the length of the string resulting from the expression.

```
INTEGER1 StrCnt := LENGTH('ABC' + 'XYZ');  // Result is 6
```

String Function: TRIM

TRIM(string*value* [, *flag*]**)**

- *stringvalue* – The string from which to remove spaces.
- *flag* – Optional. Specifies which spaces to remove. RIGHT removes trailing spaces (this is the default). LEFT removes leading spaces. RIGHT,LEFT removes both leading and trailing spaces. ALL removes all spaces.

The **TRIM** function returns the *stringvalue* with all trailing and/or leading spaces removed.

```
STRING20 StrVal := '   ABC';  // Contains 3 leading, 14 trailing spaces

VARSTRING StrVal1 := TRIM(StrVal, ALL); // Contains 3 characters
```

CROSSTAB Example:

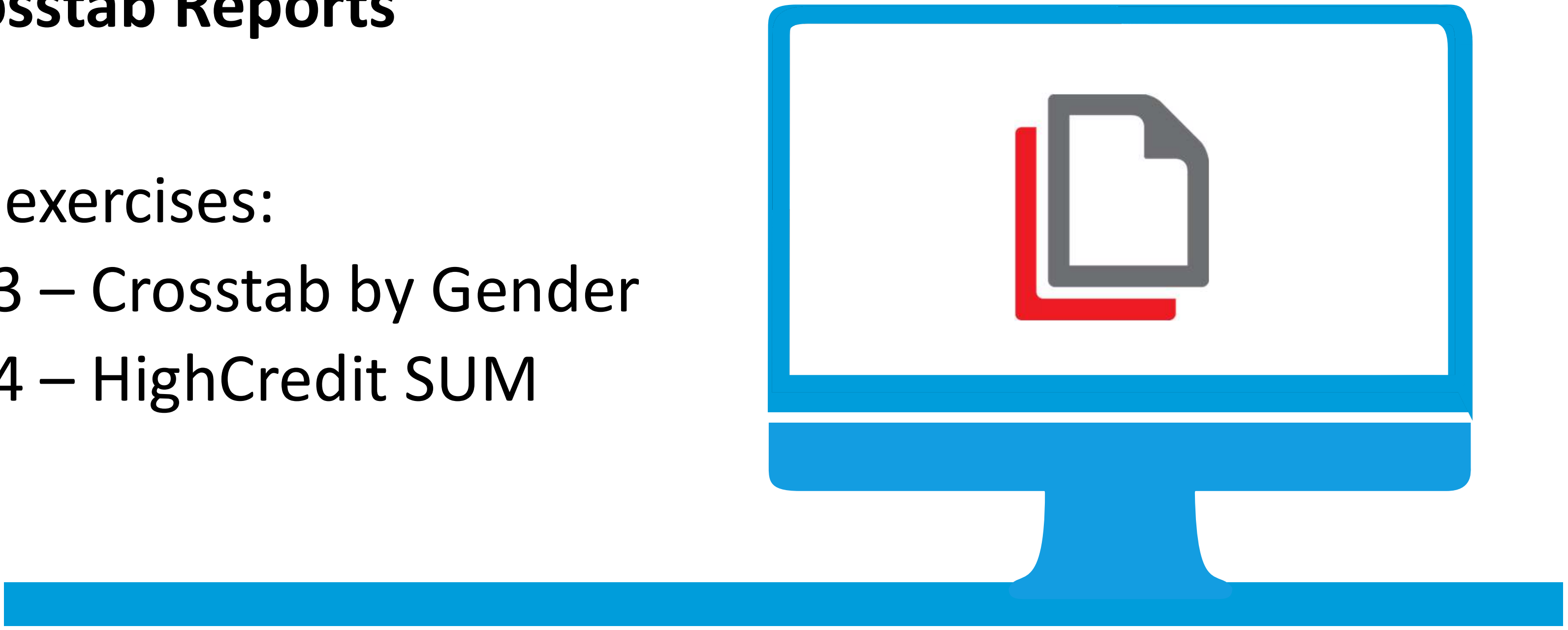
Day1.Crosstab_Example

Lab Exercises:

Crosstab Reports

Do exercises:

- 3 – Crosstab by Gender
- 4 – HighCredit SUM



INLINE DATASET

name := **DATASET(***recordset*, *recorddef***)**

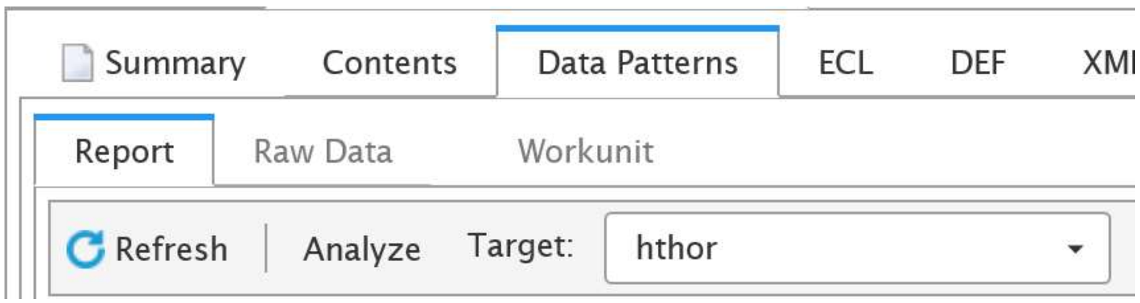
- *name* – The definition name by which the file is subsequently referenced.
- *recordset* – A set of data records contained within square brackets (indicating a set definition). Within the square brackets, each record is delimited by curly braces ({}) and separated by commas. The fields within each record are comma delimited.
- *recorddef* – The RECORD structure of the dataset.

DATASET introduces a new table into the system with the specified *recorddef* layout. This form allows you to treat an inline set of data as a data file.

```
NamesRec := RECORD
        STRING20 first_name;
        STRING20 last_name;
END;
Names := DATASET([{'John','Jones'}, {'Jane','Smith'}], NamesRec);
```

Built-In Data Profiling: Data Patterns

An extensive Data Profiling report is now built-in and available in the ECL Watch for all logical files. This report can be accessed via the Data Patterns tab:



There are three ways to use Data Patterns:

- ECL Watch (via the Data Patterns tab)
- Bundle (found on the HPCC Git Hub): https://github.com/hpcc-systems/DataPatterns.git
- Standard Library Reference (STD.DataPatterns)

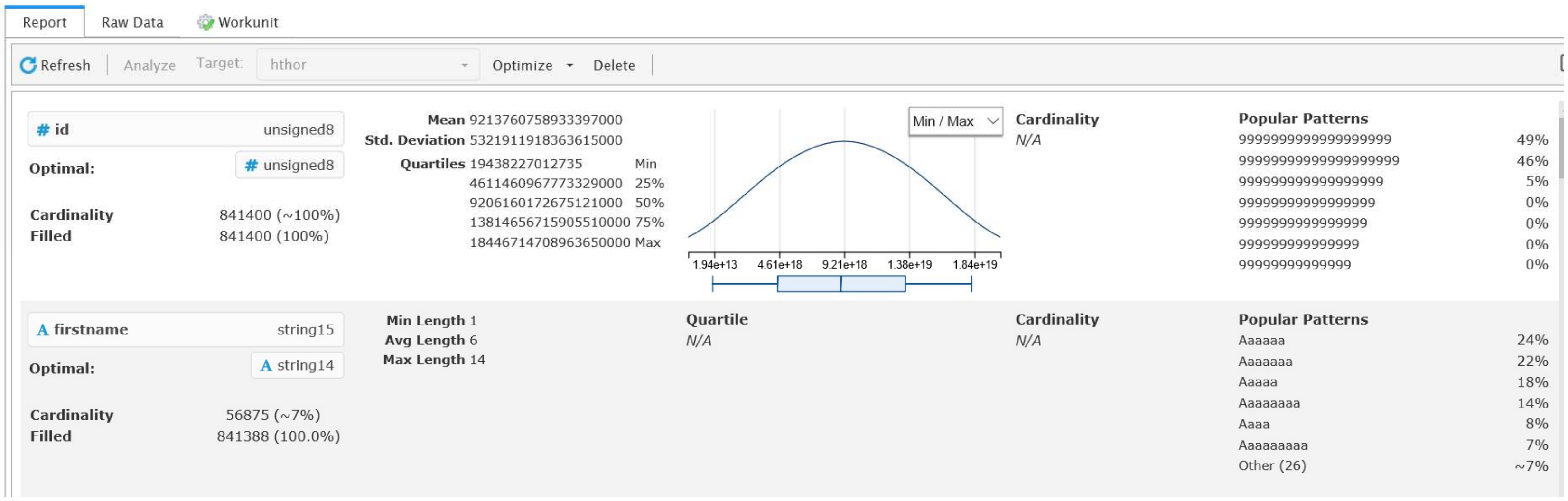
Built-In Data Profiling: Data Patterns

```
IMPORT STD.DataPatterns;

filePath := '~aaa::bmf::reptest::accounts';
ds := DATASET(filePath, RECORDOF(filePath, LOOKUP), csv);
profileResults := DataPatterns.Profile(ds);
OUTPUT(profileResults, ALL, NAMED('profileResults'));
```

Requirements (for ECL Watch Analysis);

1. File must have an ECL RECORD
2. Size of the file should not be too large (Use SAMPLE for larger files.



# id unsigned8

Optimal: # unsigned8

Cardinality 841400 (~100%)
Filled 841400 (100%)

Mean 9213760758933397000
Std. Deviation 5321911918363615000
Quartiles

19438227012735	Min
4611460967773329000	25%
9206160172675121000	50%
13814656715905510000	75%
18446714708963650000	Max

Cardinality
N/A

Popular Patterns

9999999999999999999	49%
99999999999999999999	46%
999999999999999999	5%
99999999999999999	0%
9999999999999999	0%
999999999999999	0%
99999999999999	0%

A firstname string15

Optimal: A string14

Cardinality 56875 (~7%)
Filled 841388 (100.0%)

Min Length 1
Avg Length 6
Max Length 14

Quartile
N/A

Cardinality
N/A

Popular Patterns

Aaaaaa	24%
Aaaaaaa	22%
Aaaaa	18%
Aaaaaaaa	14%
Aaaa	8%
Aaaaaaaaa	7%
Other (26)	~7%

Lab Exercises

Do exercises:

- 5 – INLINE Dataset
- 6 – Profile Persons

End of Day 1 Workshop:

More to come!!

See you tomorrow for Day 2!

Thanks for attending!

✓**Download it all at:**
https://github.com/hpcc-systems/NCF2023

✓**Contact us:** **robert.foreman@lexisnexisrisk.com**
richard.taylor@lexisnexisrisk.com