HPCC Systems workshop – Day 1

SEPTEMBER 2022

H. Watanuki
LexisNexis RISK Solutions

Day 1 – September 26th 5:00 pm (GMT-3)

1. Two-hour workshop via MS Teams

2. Resources:
    1. **PC and Github (https://github.com) account**
    2. **HPCC Systems cluster: http://training.us-hpccsystems-dev.azure.lnrsg.io:8010/**

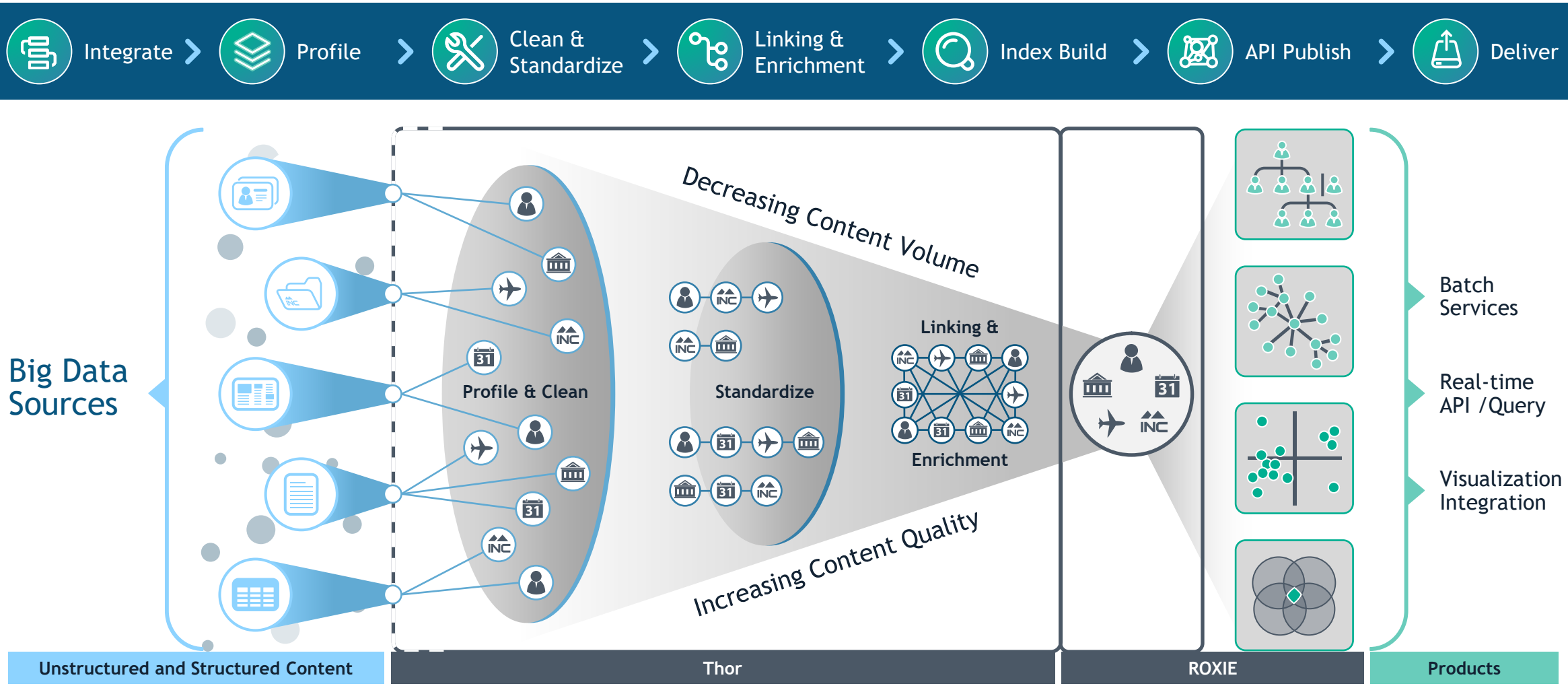
3. Core content:

    1. **Recap fundamentals of HPCC Systems:**
        1. **Background and System overview: 15 mins**
        2. **ECL cheat sheet review: 30 mins**

    2. **ETL and Data delivery with HPCC Systems:**
        1. **Main ECL primitives for ETL: 30 mins**
        2. **Query publication and usage: 30 mins**

    3. **Q&A**
        1. **Specific questions from audience: 15 mins**

HPCC Systems Data Enrichment Pipeline



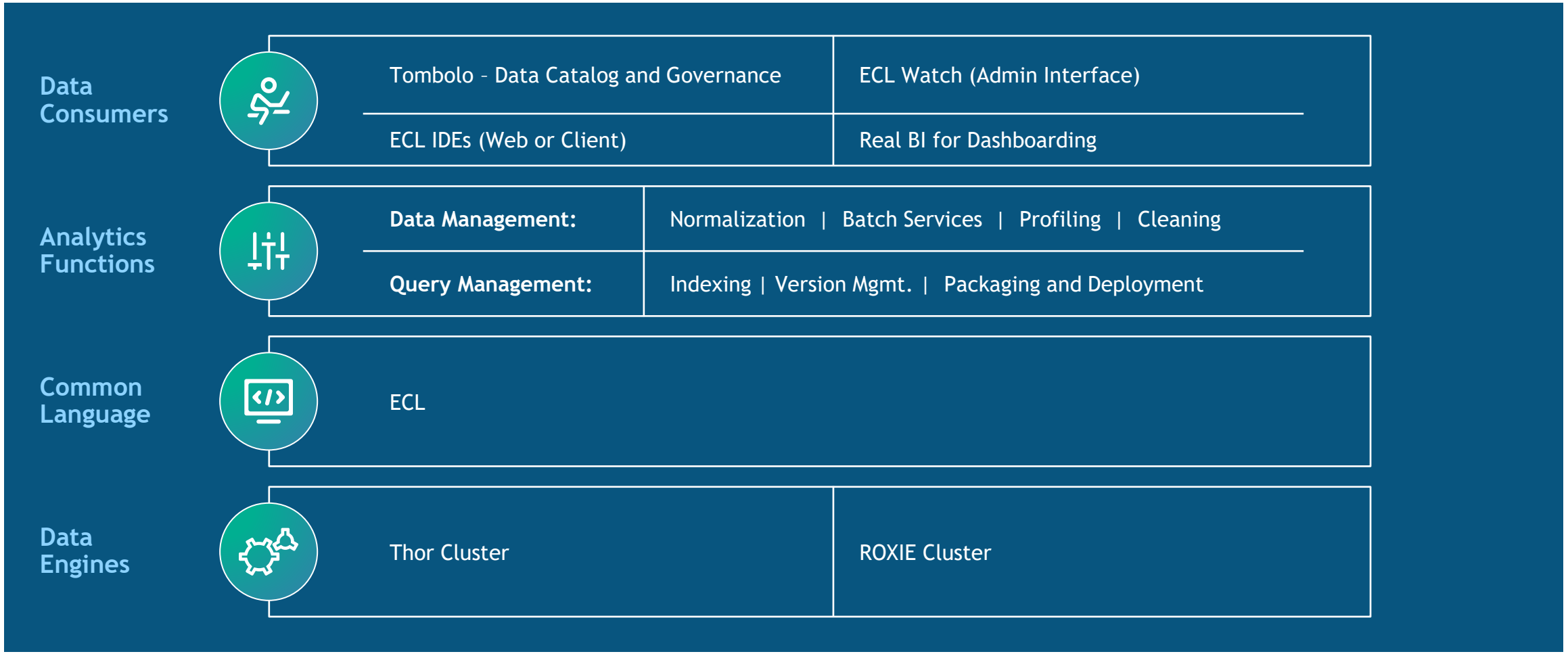
Key Aspects of HPCC Systems

- **Designed as a Data Lake Architecture**
  - Schema on Read
  - Mix Raw and Processed Data
  - Experiment and Production workloads can coexist
  - Scale as the data grows
- **Batch, Real-Time, and Streaming Data Ingestion**
- **Built to process a Fast Data Enrichment pipeline**
- **Scalable to many petabytes of data**

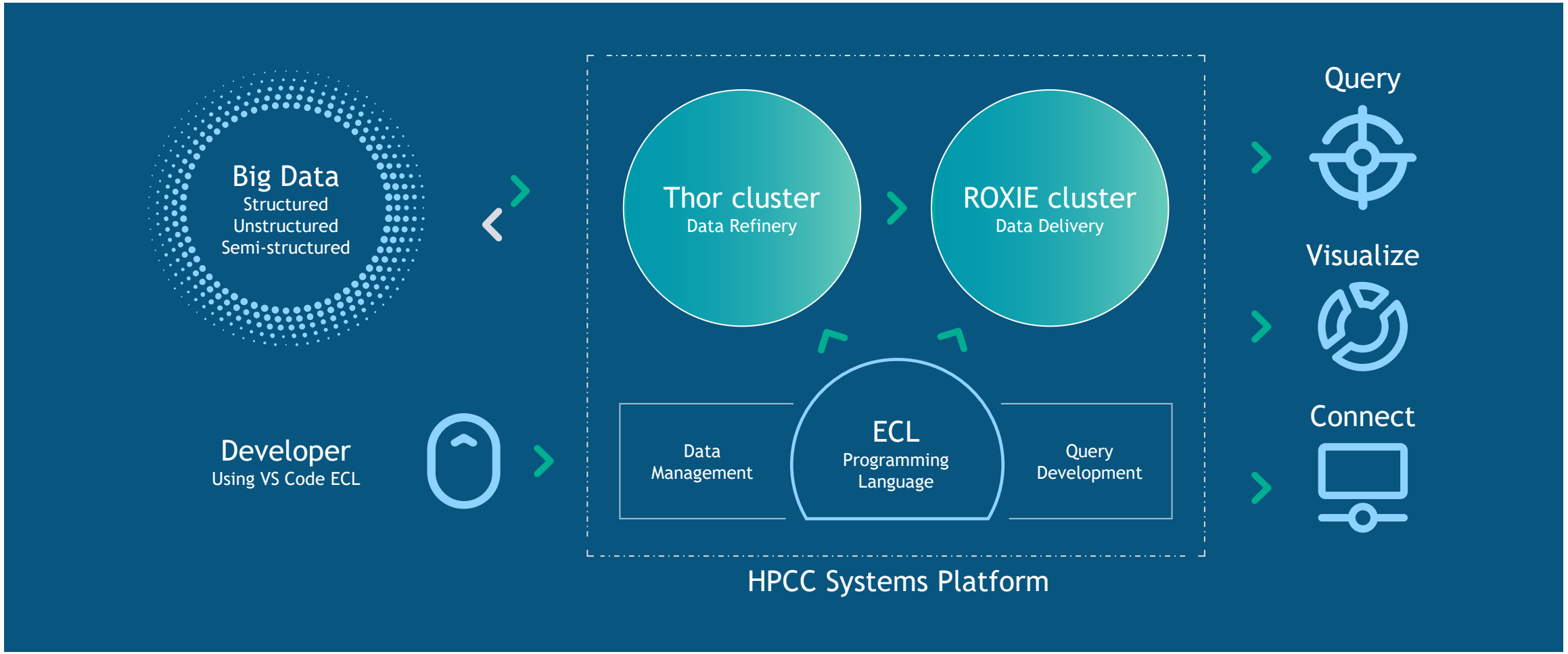
- **Real-time Data Delivery with very high concurrency**
- **Runs on Physical, Virtual, and Kubernetes** (Azure, AWS, Google Cloud)
- **ECL - Declarative and made for SQL Like programming. Compiles to Data Flow Graphs and C++.**
- **Basic ECL principle - Execute complex logic as close to the data as possible-Training and Support**

HPCC SYSTEMS

HPCC Systems Layers



The HPCC Systems Engines



Demo: HPCC Systems Data Enrichment Pipeline

Reference

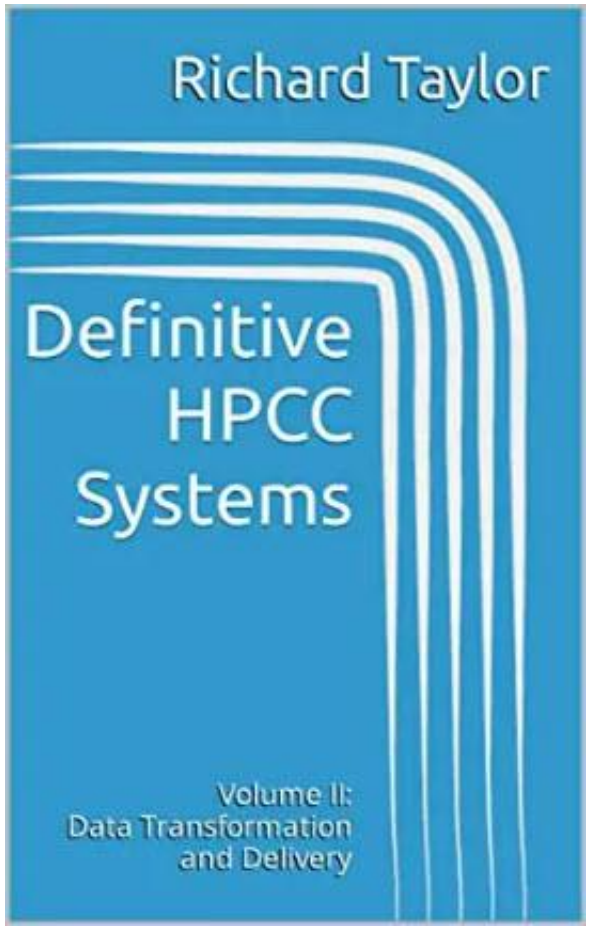
This workshop is based on the new book by Richard Taylor:

Definitive HPCC Systems
Volume II: Data Transformation and Delivery
Hour 1: Chapters 2 and 3
Hour 2: Chapter 4
Hour 3: Chapter 5 (ECL Cookbook)

Volume I and II is currently available on Amazon!

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

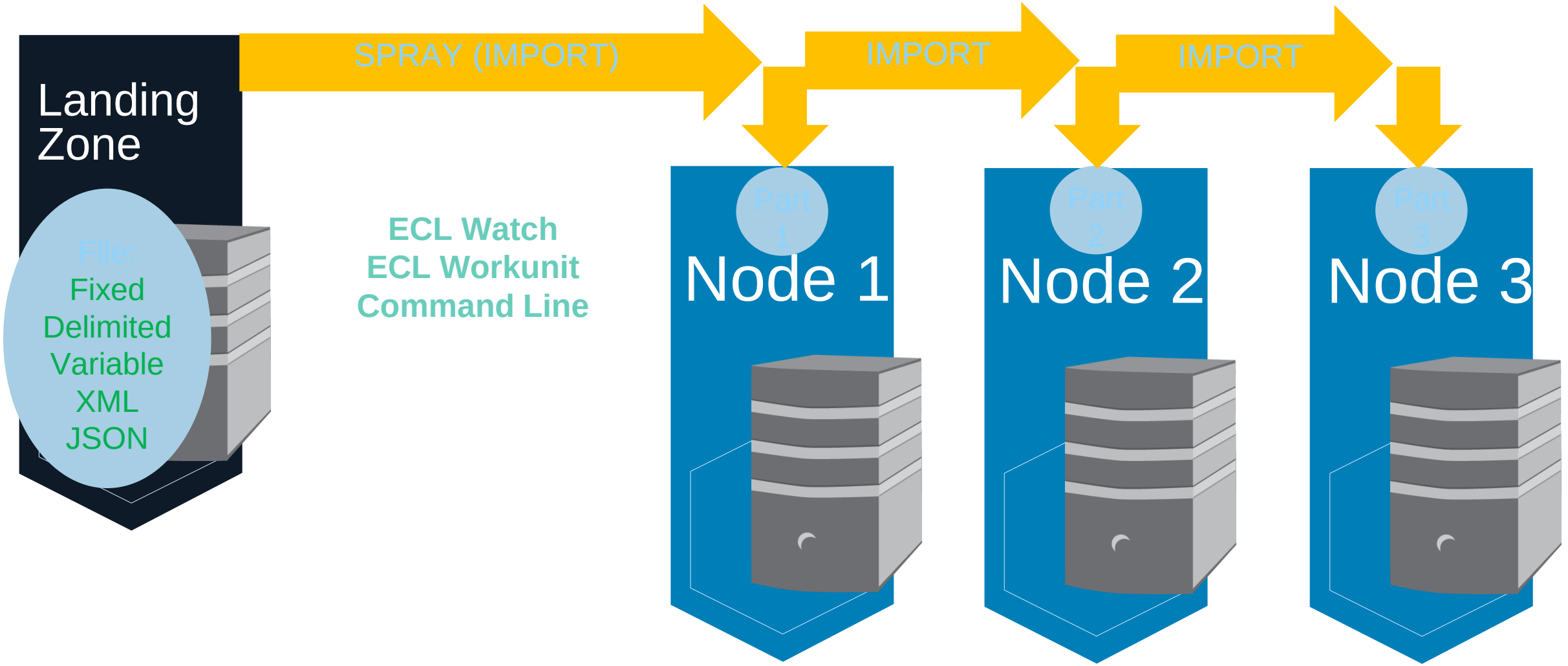
https://www.amazon.com/Definitive-HPCC-Systems-Transformation-Delivery-ebook/dp/B0BCMZCXDD



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

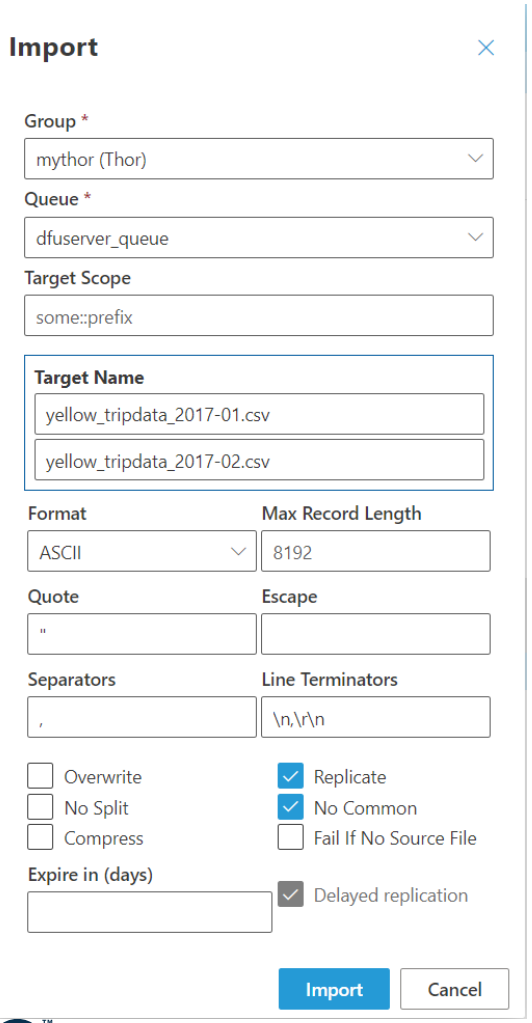


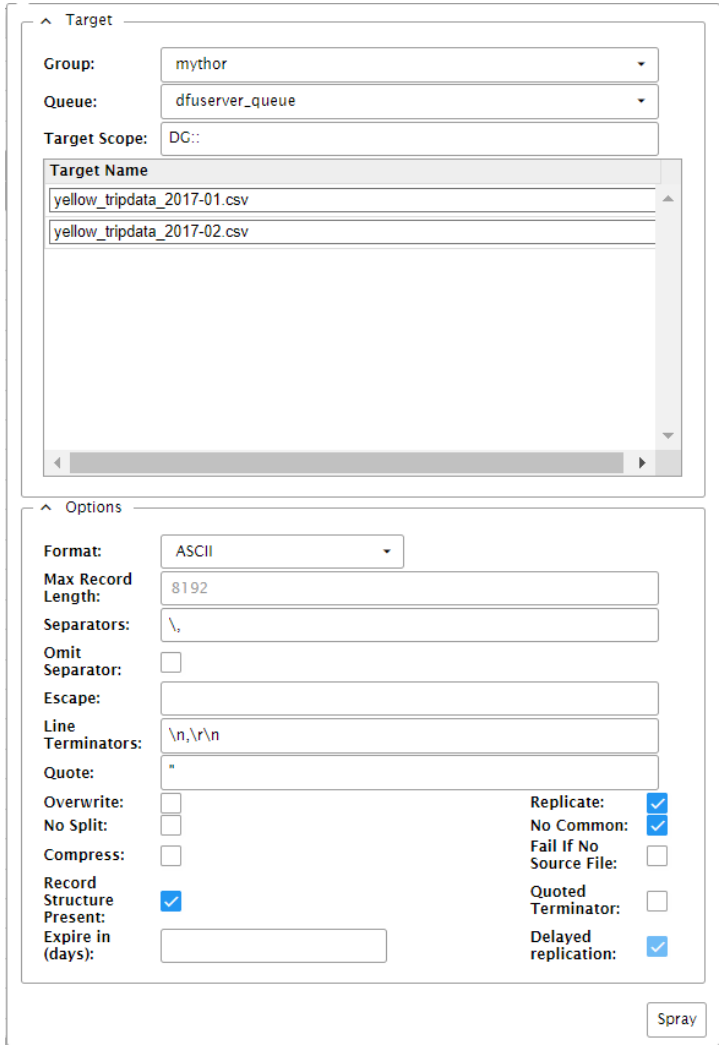
Referenced in ECL as a single logical file…

Workshop 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.

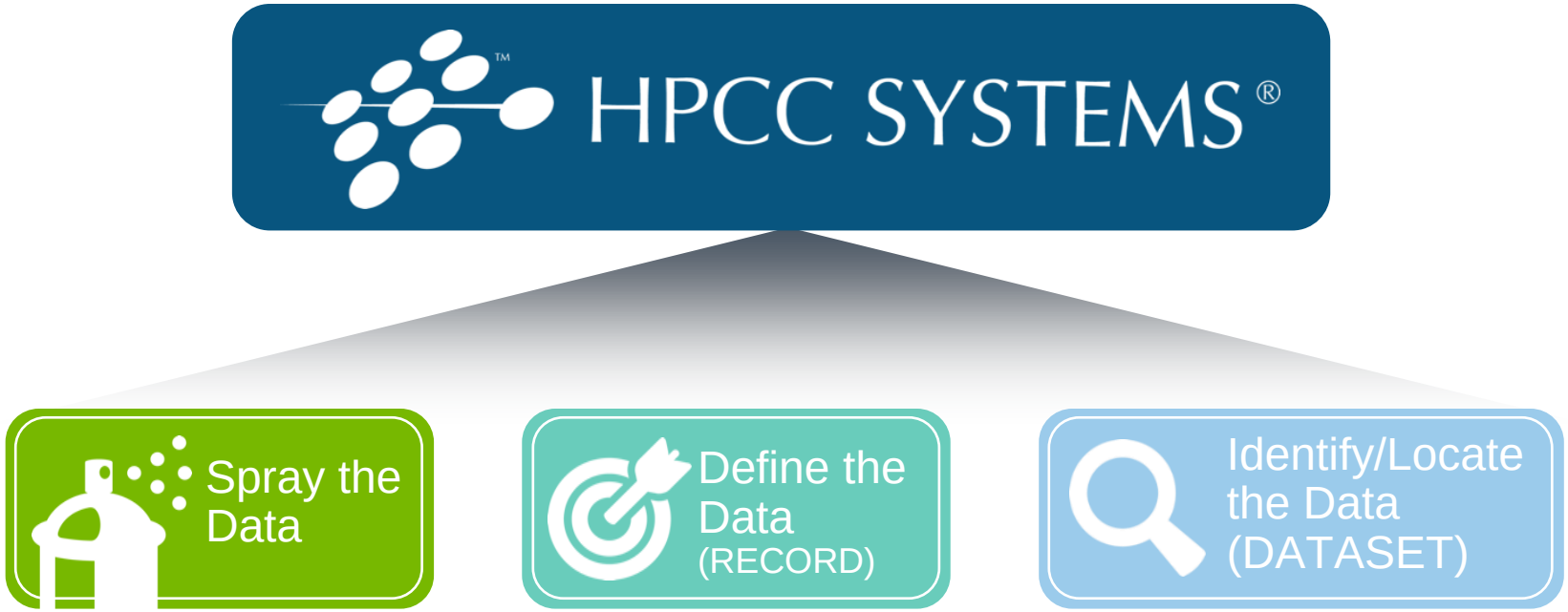
Delimited Spray Options





Three ECL Data Rules

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



Get the RECORD, create the DATASET:

Logical Files | Landing Zones | Workunits | X

Summary | Contents | Data Patterns | ECL

```
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;
```

*CommDay2022.Code.File_Yellow

Submit

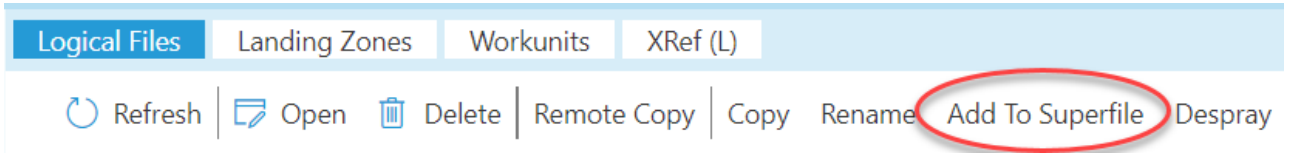
```
EXPORT File_Yellow := MODULE
 EXPORT Layout := RECORD
    STRING VendorID;
    STRING tpep_pickup_datetime;
    STRING tpep_dropoff_datetime;
    STRING passenger_count;
    STRING trip_distance;
    STRING RatecodeID;
    STRING store_and_fwd_flag;
    STRING PULocationID;
    STRING DOLocationID;
    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)));
END;
```

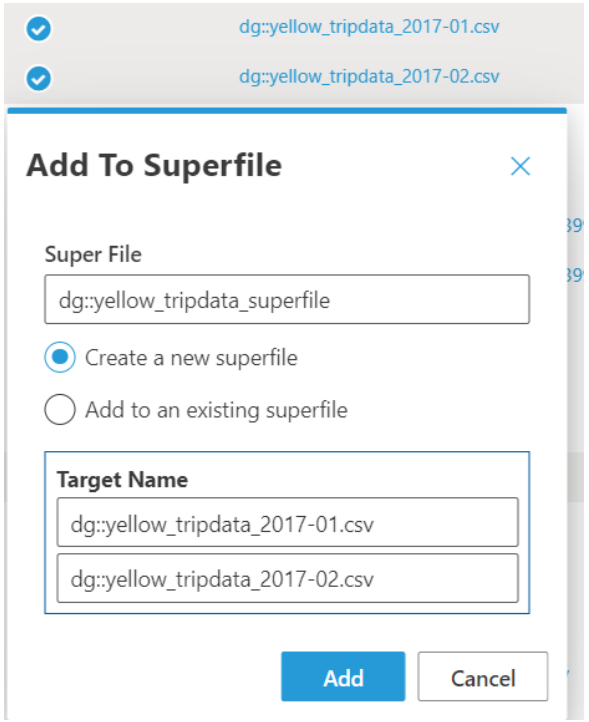
HPCC SYSTEMS

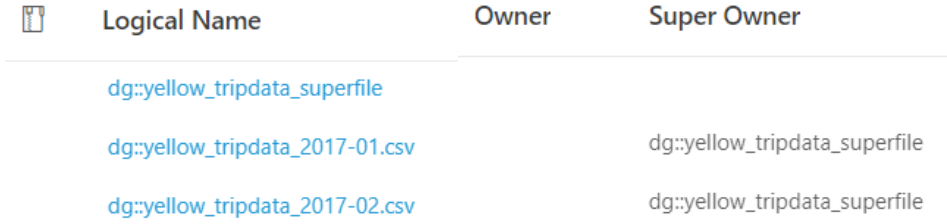
Combining Common Data

- 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:








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

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.

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

HPCC SYSTEMS

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

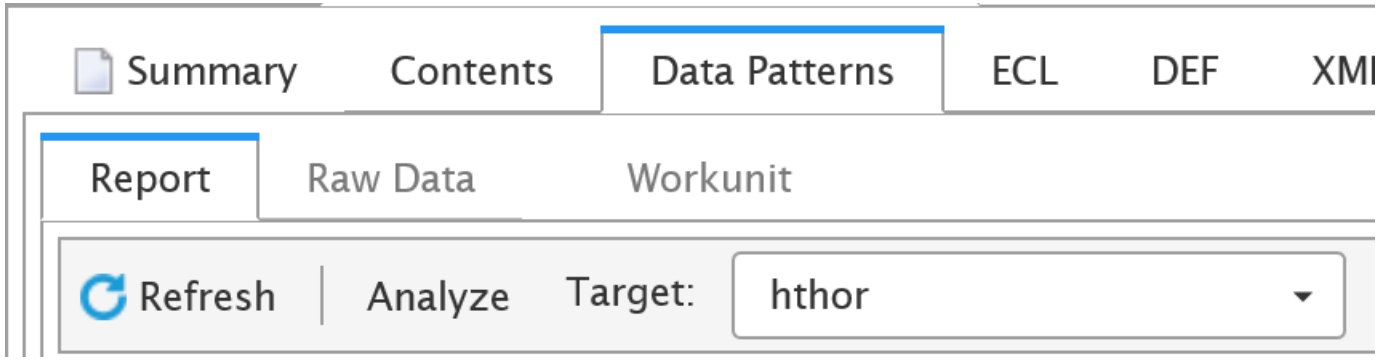
- If you periodically receive new files to add to the collection you already have, then it's a really good idea to re-run your profile code on the new file (alone) to see if there are any significant differences in the data that might require changes to your processing code or final product data format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

BWR_ProfileCompare.ecl

HPCC SYSTEMS

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 Hygiene (Cleaning and Standardization)

- After analyzing and understanding your data, the Data Hygiene step (cleaning and standardization) is the next action you need to take in your data ingest process.
- The first part of that (at least, it is in LexisNexis Risk Solutions operations) is always to add your own globally unique identifier to each input record.
- Why? Because you always want to be able to get back to the original input record your final product data is derived from.
- The "global" context refers to the universe of your own input data, not the entire world. That makes the problem much more easily solvable -- you just need to ensure that your identifier values are unique across all of your input datasets for a single product file.

BWR_BestRecord.ecl

HPCC SYSTEMS

Data Standardization

- The CSV file format (the type of files that we sprayed) represents all data items as a string. That makes a human-readable file, but not a very efficient computer-readable data storage/operation mechanism. Therefore, the next thing we need to do is to decide what data types to use to contain each field's value so that our end product data is most efficient for both storage and usability.

We can start with the RECORD structure given to us by the DataPatterns.BestRecordStructure function:

TaxiData.ecl

```
NewLayout := RECORD
    UNSIGNED1 vendorid;
    STRING19 tpep_pickup_datetime;
    STRING19 tpep_dropoff_datetime;
    UNSIGNED1 passenger_count;
    REAL4 trip_distance;
    UNSIGNED1 ratecodeid;
    STRING1 store_and_fwd_flag;
    UNSIGNED2 pulocationid;
    UNSIGNED2 dolocationid;
    UNSIGNED1 payment_type;
    REAL8 fare_amount;
    REAL4 extra;
    REAL4 mta_tax;
    REAL4 tip_amount;
    REAL4 tolls_amount;
    REAL4 improvement_surcharge;
    REAL8 total_amount;
END;
```

Data Standardization

- Now that we've defined exactly what storage format our data needs to be in going forward, we need to transform it from the input string data to the binary data types that we've decided upon.
- Data Hygiene is the step where you take the opportunity to clean up the data, getting rid of any “garbage” that you don’t want. It also gives you the opportunity to standardize your data (such as, if you get phone numbers that have been input with multiple formats you could standardize them all into a single format).

BWR_CleanTaxiData.ecl

Data Exporting

- Once you have your data cleaned and standardized, you need to think about how to get that data to your end-users. There are several ways to do that:

1. Offload the data from your Thor to another platform (such as some kind of Business Intelligence software)
2. Use the WsSQL web service to make the data available to any SQL-based software
3. Create end-user queries and publish them to ROXIE

Building our Product

The cleaned raw data we have now contains these details for each trip:

- ✓ the pickup and dropoff date/times
- ✓ the pickup and dropoff locations
- ✓ the distance traveled
- ✓ the amount of the fare

So, from this data above we can compute this information about each trip:

- ✓ during which day of the week and hour the trip started
- ✓ how long the trip took (the duration)
- ✓ how far the trip was (the distance)

Building our Product

Putting this information together into a dataset:

For every possible combination of

- ✓ The pickup and drop-off locations
- ✓ Day of week
- ✓ Hour of the day

We can return the average:

- ✓ fare amount
- ✓ how long the trip took (the duration)
- ✓ how far the trip was (the distance)

Using HPCC, ECL and ROXIE, this information will allow us to create a query that can “instantly” return the answer to the end-user’s question, because all the possible answers will have been ***pre-built.***

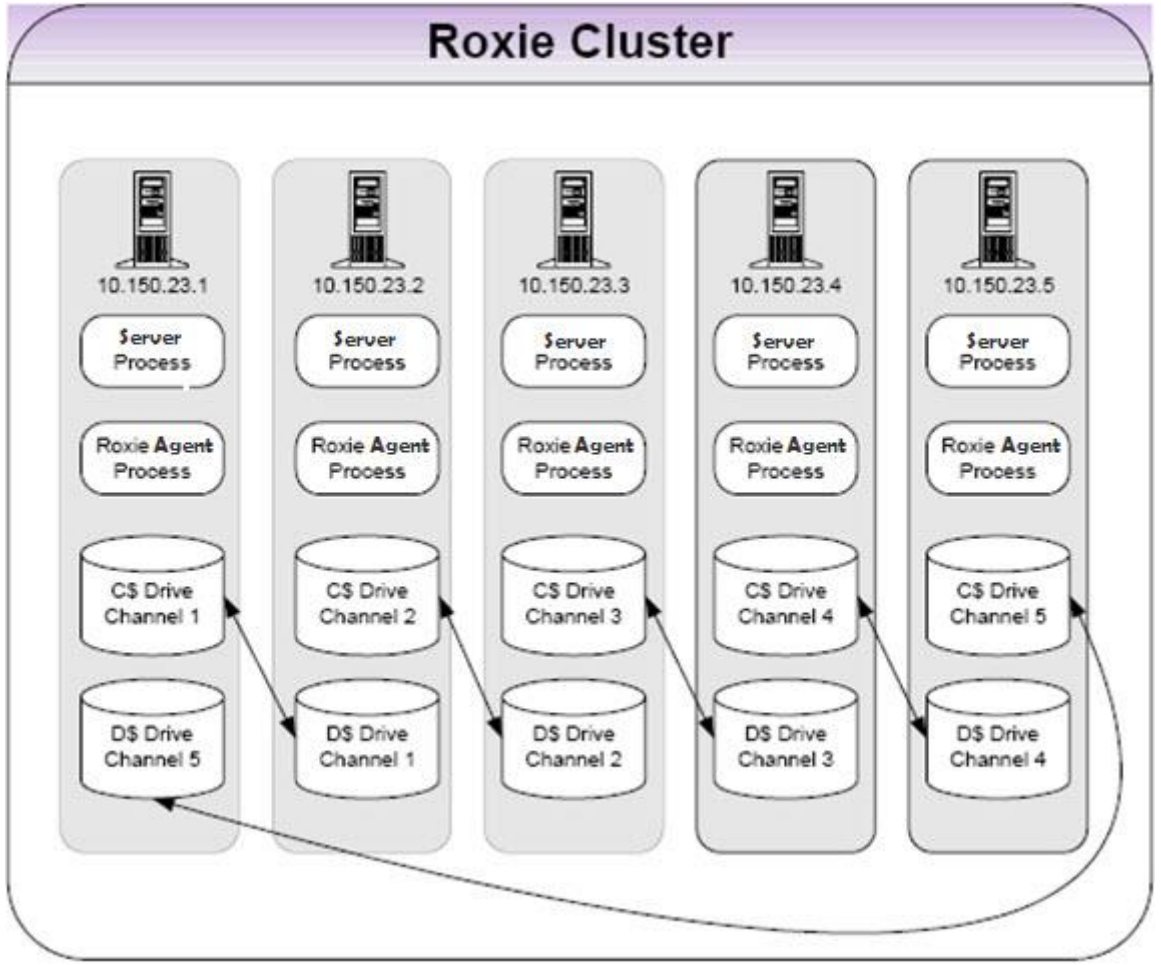
What is ROXIE?

ROXIE is also known as the HPCC "Rapid Data Delivery Engine".

It is a number of machines connected together that function as a single entity running:

Server (Farmer) processes Agent (Slave) processes

Some special configurations have Server Only or Agent Only nodes.



This example shows a 5-node ROXIE

Building our Product: Trip Duration Calculator

1. Set up time constants Utility (**Secs.ecl**):

```
EXPORT Secs := ENUM(M = 60,H = 60*60,D = 24*60*60);
```

2. Build a "TripTime" Utility (**TripTime.ecl**):

- ✓ TripTime is a FUNCTION
- ✓ Add a nested local FUNCTION to calculate total seconds in a time element.
- ✓ We have to take into account the possibility that the pickup and dropoff dates may not be the same (such as a pickup at 11:45 PM and dropoff at 1:00 AM), so the TripDays definition uses the *STD.Date.DaysBetween()* function to determine day-spanning trips (the function returns zero for same-day pickup and dropoff).
- ✓ Relying on multiplication by zero, the RETURN expression works for either single or multi-day trips to the return the total number of seconds elapsed from pickup to dropoff.

Distilling the Product

- Now that we've defined the support functions we're going to need, the next step is to actually write the code to extract the information we want from the cleaned/standardized data.
- Let's build a TABLE to extract just the fields that we want to work with.
- Create a local FUNCTION to format the elapsed trip time output.
- Create a second TABLE to generate a cross-tab report to output one record in the result for each set of unique pickup/dropoff locations, day of week, and hour of the day.

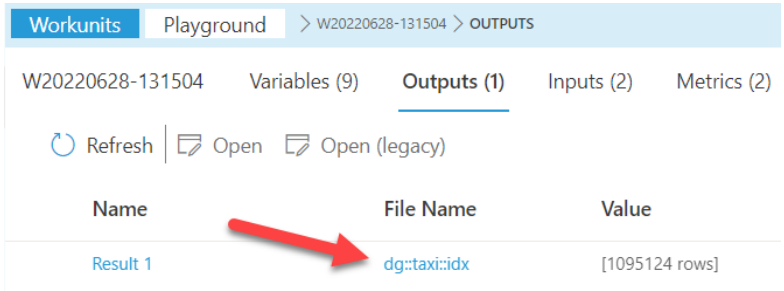
ProdData.ecl

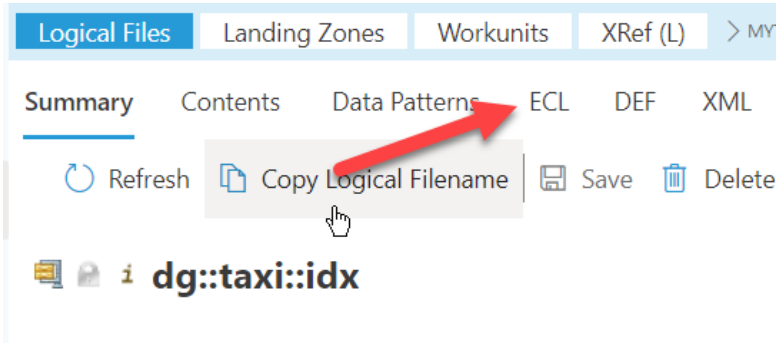
Indexing the Product

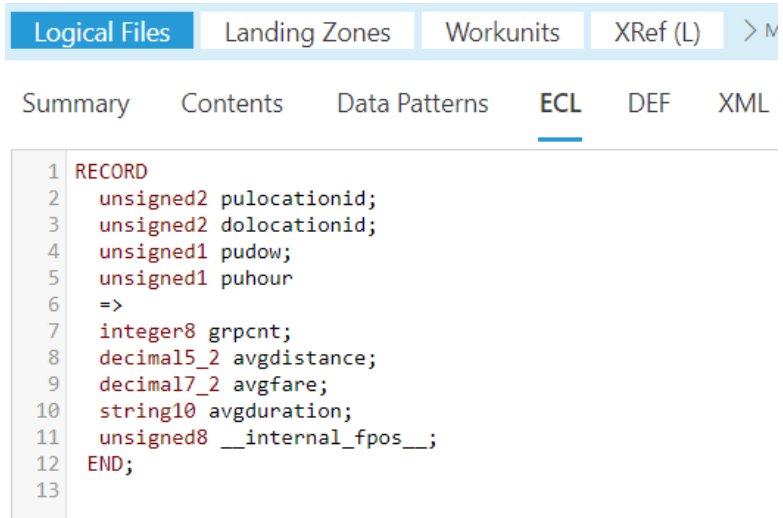
- ROXIE queries are almost always defined to get their data from INDEX files, because that's the fastest possible access to individual items in the data. Therefore, the next step in our process is to create an INDEX for our end-user query to use.
- Recreating the production INDEX as new updated data comes in will be a standard process that we want to periodically run, so we need to create definitions that we can use repeatedly.
- Building the INDEX will also generate the RECORD structure needed for the INDEX that we will define.

ProdIDX.ecl

Defining the Index





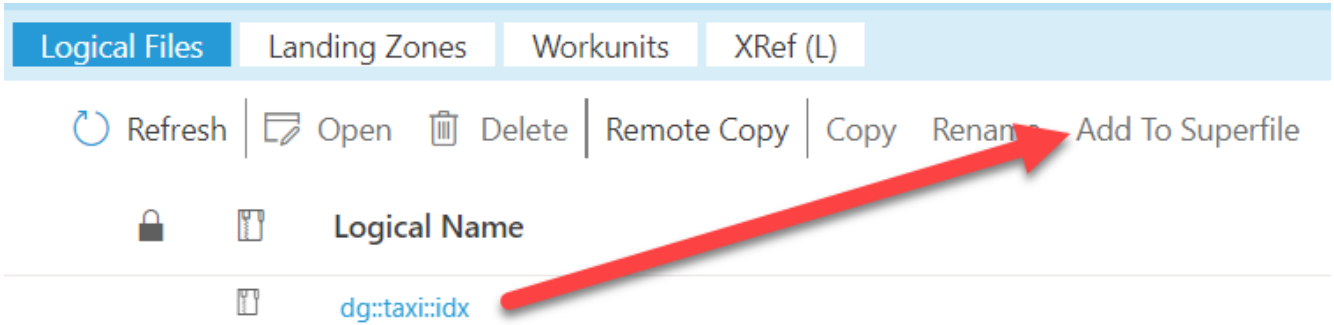


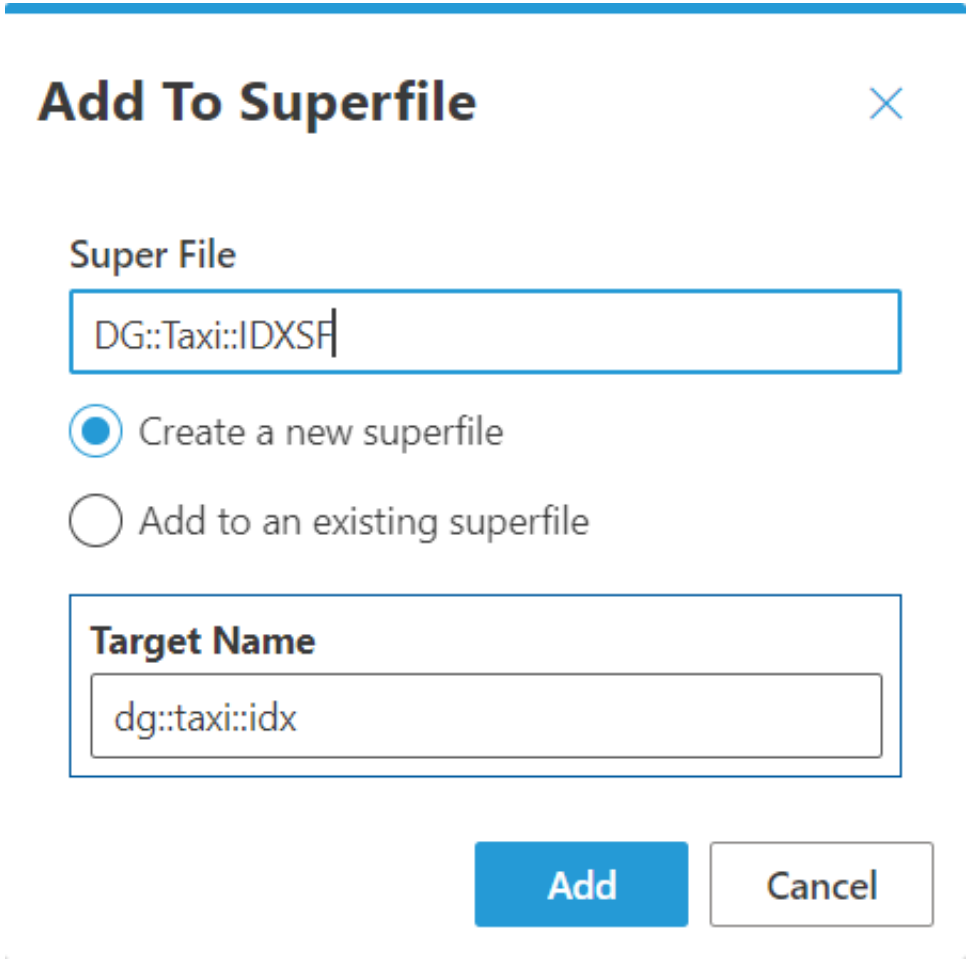
Defining the Index

- Once created, you must first define the INDEX before you can use it, so we will modify the code in your **ProdIDX.ecl** file.
- Pasting the RECORD structure from the file's Logical File Detail page's ECL tab saves you most of the typing and ensures you don't "fumble finger" the typing.
- Next, we modify the **ProdIDX.ecl** file again to add a superfile:

```
EXPORT IndexSuperFile := '~dg::Taxi::IDXSF';
…
EXPORT IDXSF := INDEX(Layout, IndexSuperFile);
```

Creating the Index Superfile





So, why do we need a SuperFile?

- ROXIE queries need to always have up-to-date data, so the INDEX will need to be rebuilt every time you have new data.
- However, you don't want to have to re-compile the query every time you update data, because really complex queries can literally take an hour or so just to compile (this is absolute truth!).
- Not only that, but it would require you to unpublish then republish the query just to update the data.
- So, the easy way to not have to re-compile when the data needs to be updated but the code hasn't changed is to define a SuperFile for the query code to use, making the SuperFile into just an alias for whatever sub-file it happens to contain.
- The trick to updating the data lies in the use of Package Maps (discussed in this Blog by Dan Camper: https://hpccsystems.com/blog/real-time-data-updates-in-roxie).

Creating the Product – Zone Lookup

- The *puLocationID* and *doLocationID* fields are just numbers, but they do relate to actual areas of New York, so it would be useful to know what they translate to. You can download a CSV file containing all the location IDs and what they reference here: https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page.
- Look for the "Taxi Zone Lookup Table (CSV)" link under the *Taxi Zone Maps and Lookup Tables* heading. Download and spray that file, then you can duplicate the code.

  ZoneLookup.ecl

##	locationid	borough	zone	service_zone
1	1	EWR	Newark Airport	EWR
2	2	Queens	Jamaica Bay	Boro Zone
3	3	Bronx	Allerton/Pelham Gardens	Boro Zone
4	4	Manhattan	Alphabet City	Yellow Zone
5	5	Staten Island	Arden Heights	Boro Zone
6	6	Staten Island	Arrochar/Fort Wadsworth	Boro Zone
7	7	Queens	Astoria	Boro Zone
8	8	Queens	Astoria Park	Boro Zone
9	9	Queens	Auburndale	Boro Zone
10	10	Queens	Baisley Park	Boro Zone
11	11	Brooklyn	Bath Beach	Boro Zone
12	12	Manhattan	Battery Park	Yellow Zone
13	13	Manhattan	Battery Park City	Yellow Zone
14	14	Brooklyn	Bay Ridge	Boro Zone
15	15	Queens	Bay Terrace/Fort Totten	Boro Zone
16	16	Queens	Bayside	Boro Zone
17	17	Brooklyn	Bedford	Boro Zone
18	18	Bronx	Bedford Park	Boro Zone
19	19	Queens	Bellerose	Boro Zone

Creating the Product – Taxi Data Service

- This service takes, as parameters, the four search terms in our INDEX.
- Optionally, the STORED workflow service can replace those parameters.
- The *dow* and *hour* parameters both have default values of 99 **allowing you to omit passing that parameter**.
- A MAP function determines which arguments to filter by using the *NoDay* and *NoHour* Boolean definitions to determine which filter expression to use.
- We use KEYED and WILD to ensure INDEX filters don't result in a full table scan.
- Results of the search are formatted to provide proper translations of numeric codes to more readable information.

TaxiDataSvc.ecl

Testing and Publishing the Product

- Before publishing, we can test the query results in THOR.
- The steps to publish are simple:
    1. **Set Target to ROXIE**
    2. **Compile *only*.**
    3. **Publish from ECL Watch**

- Test Query via myws_ecl, or through the Published Queries interface.
- This is a programmer's testing tool; it is not an end-user GUI. Its purpose is just to allow the programmer to enter values to pass to the service, and to allow you to validate the results.

Training and Support

https://hpccsystems.com/training/classes

IN-PERSON AS WELL AS
FREE ONLINE TRAINING

Learning tracks include:

- Core Platform
- Administration

Annual online Community Event including platform roadmap updates and training workshop

Monitored Stack Overflow channel for Support and Platform Issues

General Support available from Partners, including Clear Funnel, Infosys, and others

HPPC Systems is offering free courses to build your skills in working with ECL and big data!

INTRODUCTORY COURSES

Prerequisite for advanced courses

- Introduction to ECL (Part 1)
- Introduction to ECL (Part 2)

ADVANCED COURSES

$495 value each (FREE with promo)

- Advanced ECL (Part 1)
- Advanced ECL (Part 2)
- ROXIE ECL (Part 1)
- ROXIE ECL (Part 2)
- Applied ECL

HPCC SYSTEMS

HPCC Systems Community Summit

https://hpccsystems-lexisnexisrisk.expoplatform.com/

October 10-13, 2022 - General Talks, Breakout Sessions, Expo Hall, Poster Displays and Workshops

HPCC SYSTEMS


LexisNexis®
RISK SOLUTIONS