Bob Foreman/Hugo Watanuki
Senior Software Engineers
LexisNexis RISK Solutions

ECL Alive! - a Deep-Dive into Definitive HPCC Systems
Hour 2 – Data Delivery with ROXIE: Distilling the Product, Indexing, Publishing

HPCC SYSTEMS

Introduction

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

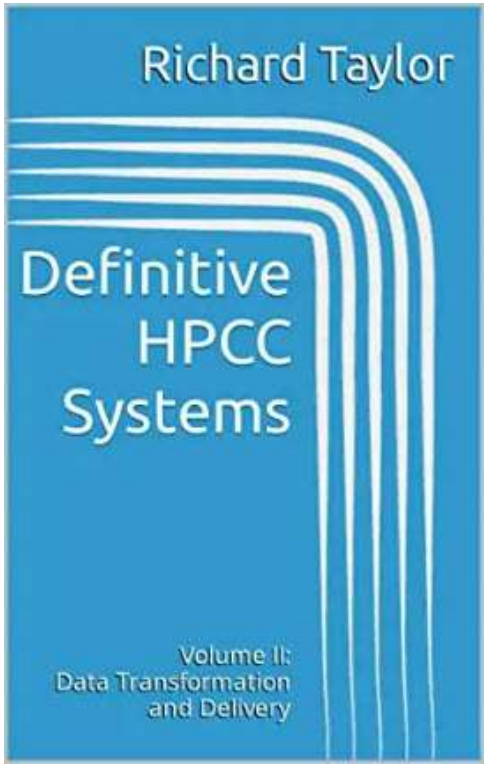
Hour 2: Chapter 4

Hour 3: Chapter 5 (ECL Cookbook)

Volume I and II are 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

- In Hour 1 of this workshop, we processed a large amount of raw data and made it all clean and standardized. The next step is to turn that into something that an end-user or customer would be willing to pay for.
- To make this data valuable to a potential customer, we first must ***distill*** the sellable (or useful) information from all that raw data. Only then can we turn it into a product.

HPCC SYSTEMS

Building our Product

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

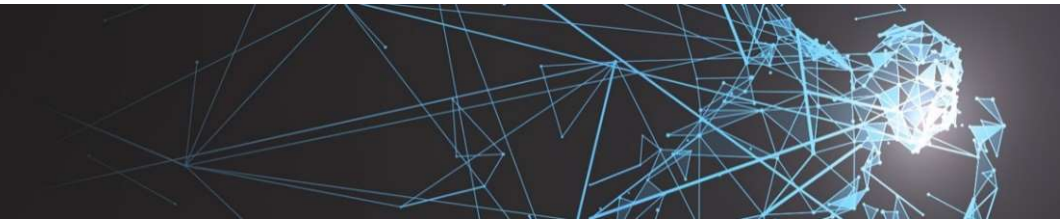
- ✓ the pickup and dropoff date/times
- ✓ the pickup and dropoff locations
- ✓ the distance traveled
- ✓ the amount of the fare

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)

HPCC SYSTEMS

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.***

HPCC SYSTEMS

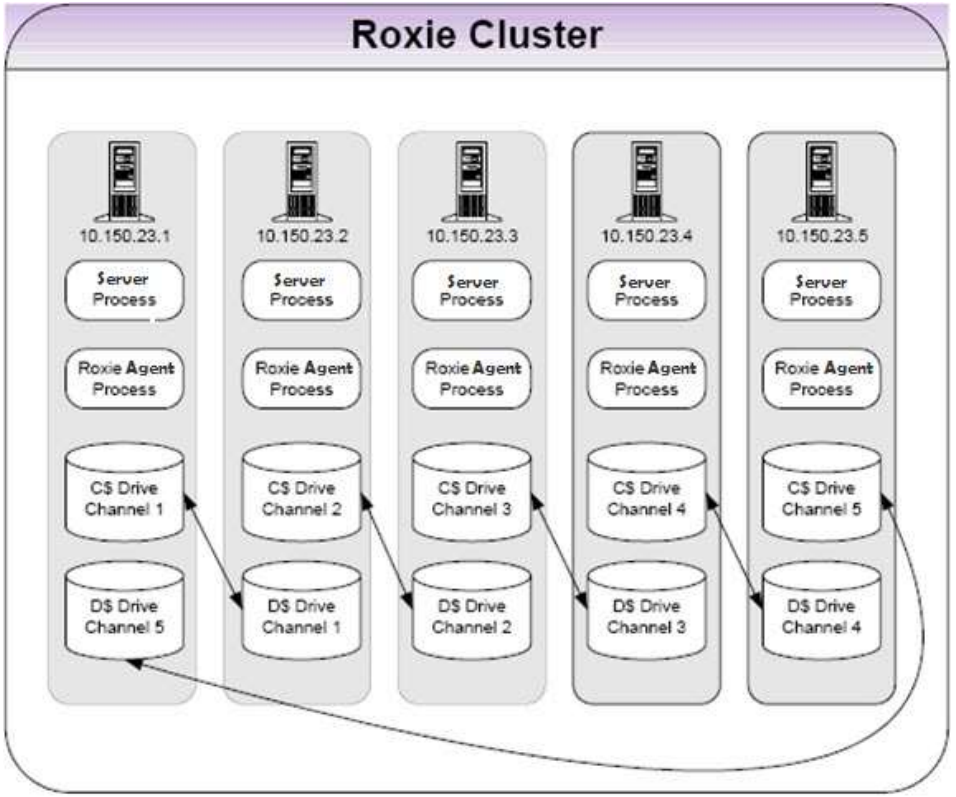
What is ROXIE?

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.



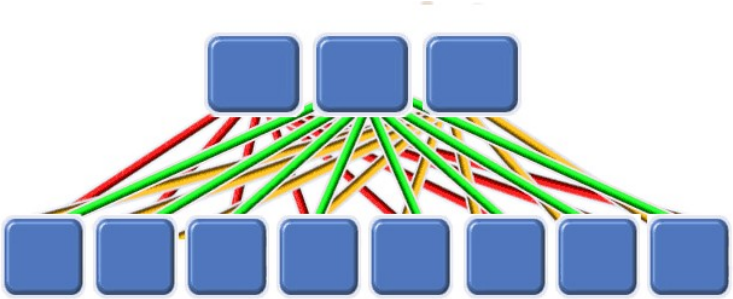
This example shows a 5-node ROXIE

What is ROXIE? (cont.)

ROXIE is a massively parallel query processing supercomputer with:

Structured Query Engine:

- Uses disk-based *indexed datasets*.
- Uses *pre-compiled* repeatable queries.



Scalable retrieval engine:

- High throughput.
- Millions of users receiving sub-second responses.

Built-in redundancy:

- Copies of data are stored on two or more nodes.
- Any peer can deliver a task for a channel.
- High availability - ROXIE cluster continues to perform even if one or more nodes fail.

Building our Product: Trip Duration Calculator

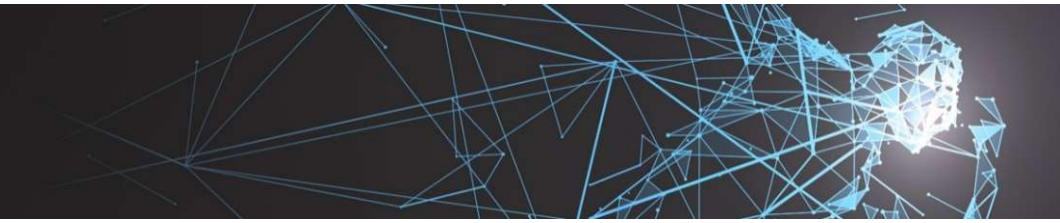
1. Set up time constants Utility (**Secs.ecl**):

```
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.

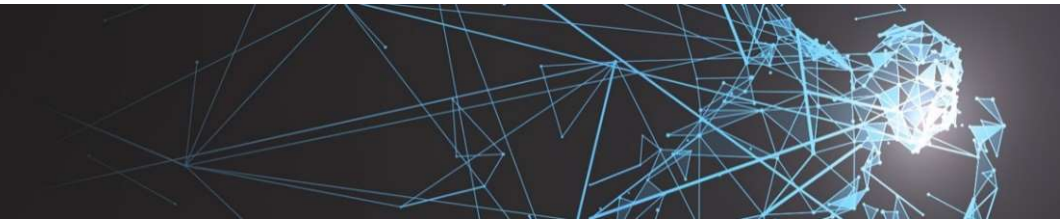
Distilling the Product



- 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

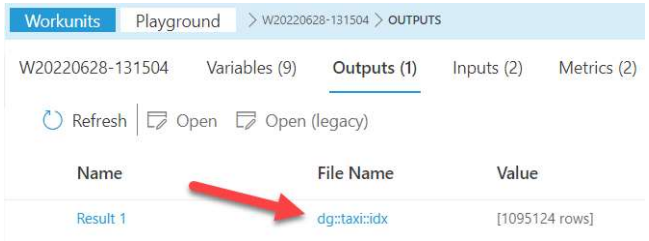
Indexing the Product

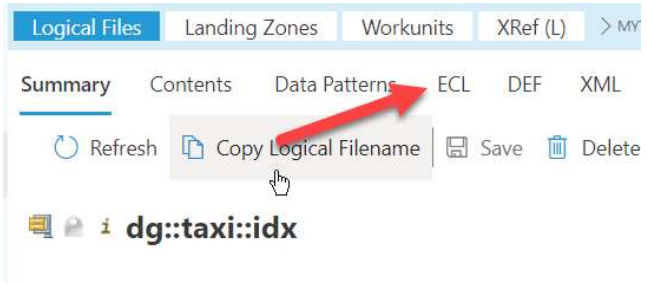


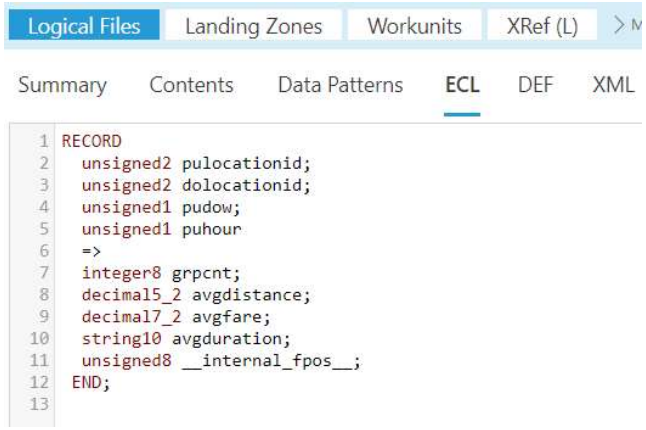
- ROXIE queries are almost always defined to get their data from INDEX files, because that's the fastest possible access to individual items in the data. Therefore, the next step in our process is to create an INDEX for our end-user query to use.
- Recreating the production INDEX as new updated data 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





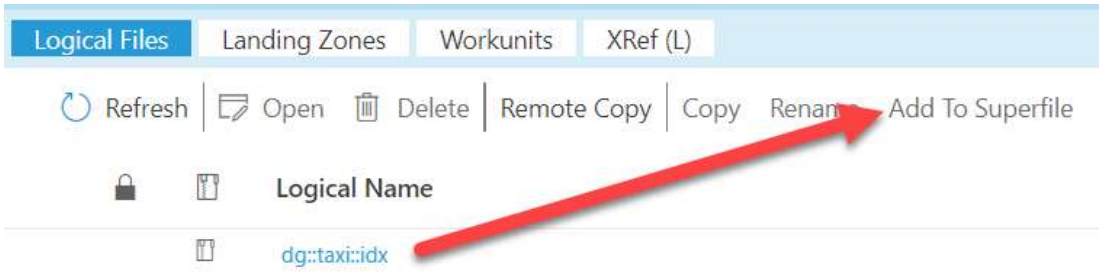


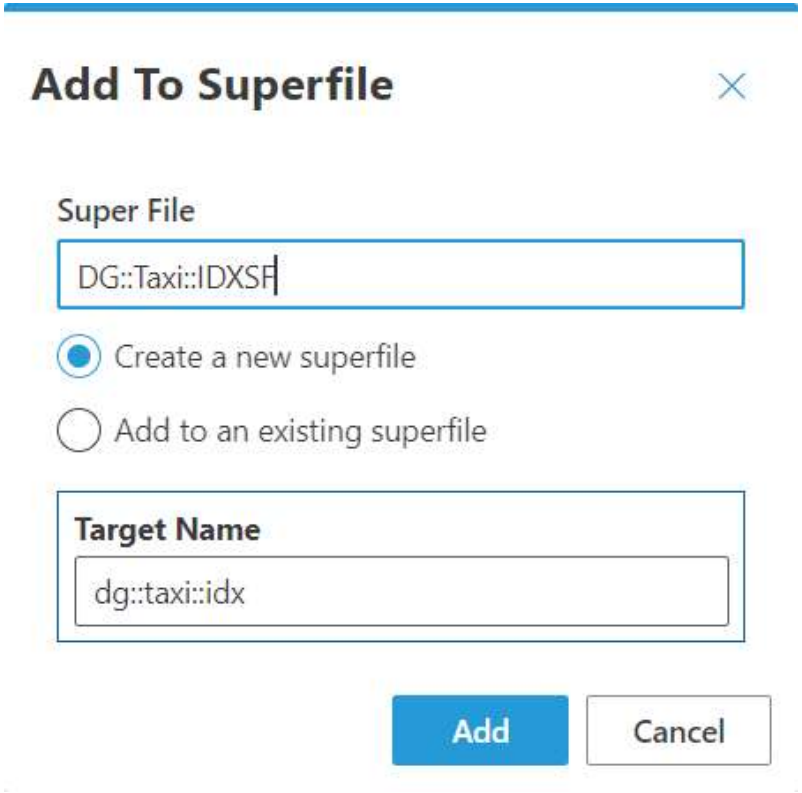
Defining the Index

- 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:

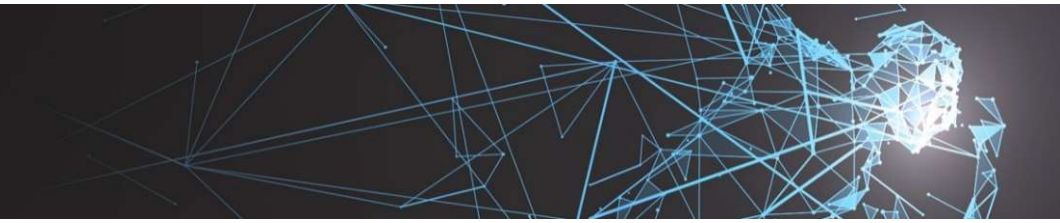
```
EXPORT IndexSuperFile := '~dg::Taxi::IDXSF';
…
EXPORT IDXSF := INDEX(Layout, IndexSuperFile);
```

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.

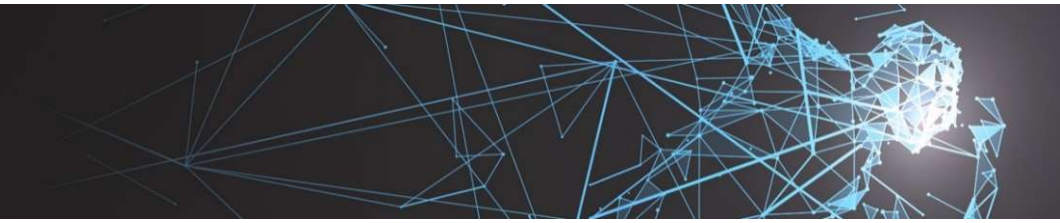
TaxiDataSvc.ecl

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.

INDEX Tips – Descending INDEXes



This section is about doing things with INDEX search terms that are not necessarily standard.

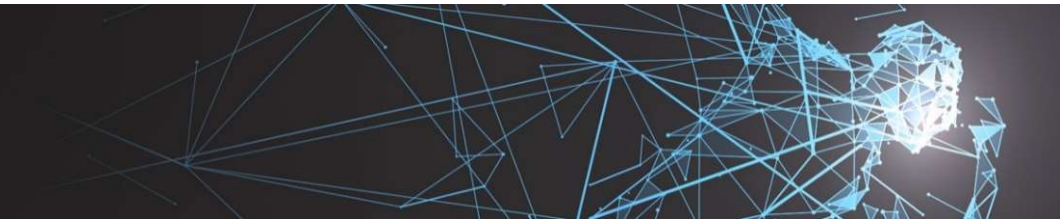
Generating Test Data – **GenDS.ecl/BitFlipStr.ecl/BWR_BuildDescendingIndex.ecl**

Before you can build any INDEX, you must first have data from which to build. Because we want to demonstrate both descending integer and string search terms with this article, we first generate all the data.

This code demonstrates:

- ✓ Use of the Standard Library Date functions to do date arithmetic.
- ✓ Generating random birthdates and names.
- ✓ Use negative integer values to create a faux descending integer INDEX search term.
- ✓ Create a function to flip all the bits in a string.
- ✓ Use bit-flipped strings to create a faux descending string INDEX search term.
- ✓ Filter those faux descending INDEXes to return specific records, ranges of records, and specific sets of records.

Bitwise Operators:



Bitwise AND	&
Bitwise OR	|
Bitwise Exclusive OR	^
Bitwise NOT	BNOT
Bitshift Right	>>
Bitshift Left	<<

INDEX Tips – BitMap in an INDEX

- Step 1: Write some new user-defined utility functions to work with bitmaps.

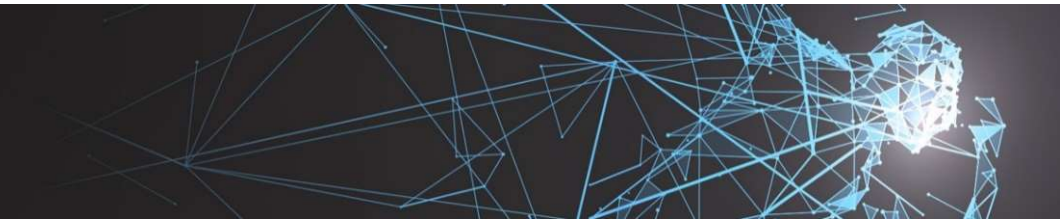
Bit.ecl
Bits.ecl

- Step 2: Add a bitmap field to our sample dataset and index it:

BWR_BitMapIndex.ecl

We define two payload indexes (BitIDX1 and BitIDX2) using the Bits bitmap field as their search term. Then the BUILD action creates the files and writes them to disk. Next, define exactly what each bit represents using ENUM.

End of Hour 2 Workshop:



And Even More to Come!!

See you tomorrow for Hour 3!

Thanks for attending!

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

✓**Contact us:** **robert.foreman@lexisnexisrisk.com**
hugo.watanuki@lexisnexisrisk.com

HPCC SYSTEMS