Definitive HPCC Systems

JANUARY 2023

Day 2 Workshop:
Data Sequencing and Standardization

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

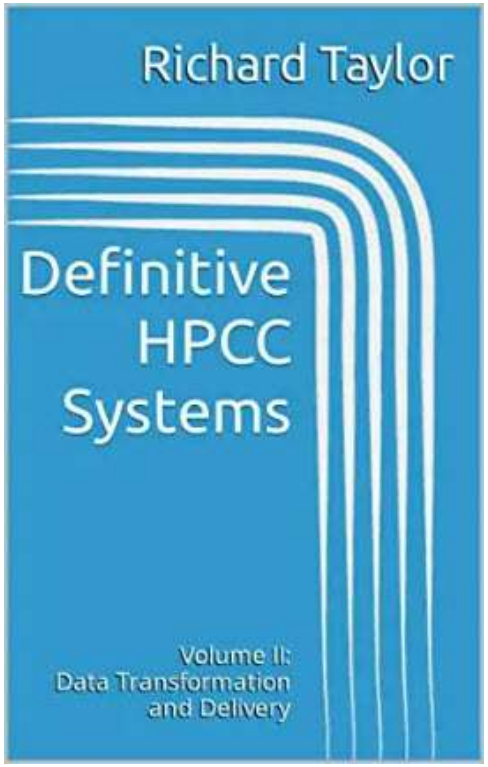
Day 3: Chapter 4

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

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

fnMAC_GenUID.ecl

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

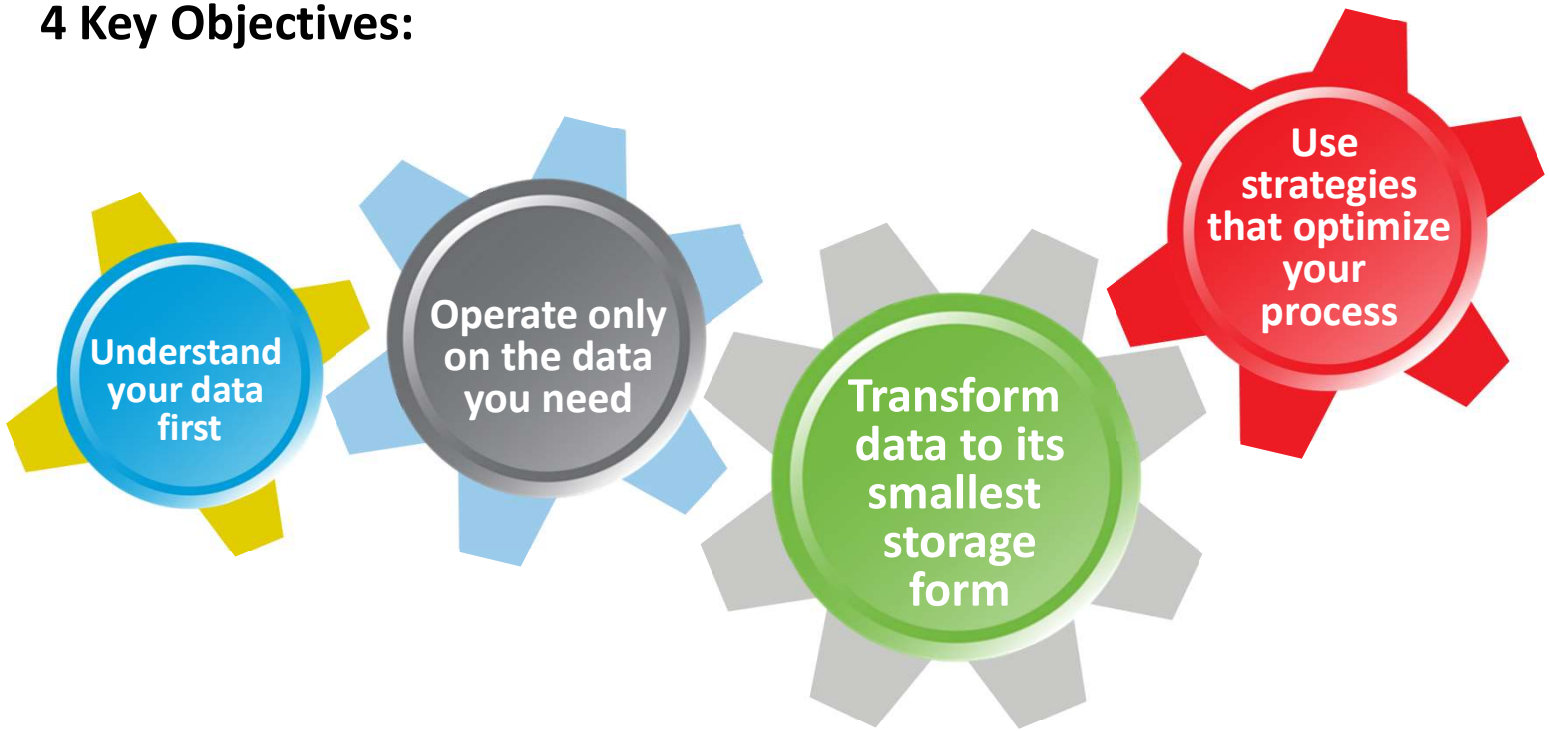
Data Exporting

- 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

ETL in ECL Objectives

4 Key Objectives:



Operate only on the data that you need:

Assign unique record IDs to all input data

- Use a MACRO, PROJECT(dataset, COUNTER), or initial FILEPOS to create them

Clean and standardize the data as appropriate

- Standardize names, addresses, dates, times, etc.

Use Vertical projections ("vertical slice" TABLE)

- Allows you to work only with the fields you actually need

Use Horizontal projections (record filters)

- Allows you to select only the records relevant to the problem.
- Exclude records with NULL values in key fields, and otherwise filter out bad or irrelevant data unless it can be cleaned.

Working with TRANSFORMs

- TRANSFORM Structure
- PROJECT Function
- PERSIST Workflow Service

TRANSFORM Structure

resulttype *funcname*(*parameterlist*) := **TRANSFORM**

SELF.*outfield* := *transformation*;

END;

- *resulttype* – The name of a RECORD structure definition specifying the output format of the function.
- *funcname* – The name of the function the TRANSFORM structure defines.
- *parameterlist* – The value types and labels of the parameters that will be passed to the TRANSFORM function.
- **SELF** – Indicates the *resulttype* structure.
- *outfield* – The name of a field in the *resulttype* structure.
- *transformation* – An expression specifying how to produce the value assigned to the *outfield*.

PROJECT Function

PROJECT(*recordset*, *transform*)**

- *recordset* – The set of records to process.
- *transform* – The TRANSFORM function to call for each record in the *recordset*.

The **PROJECT** function processes through all the records in the *recordset* performing the *transform* function on each record in turn.

The *transform* function must take at least one parameter: a **LEFT** record of the same format as the recordset. It may take an optional second parameter: an integer COUNTER specifying the number of times the transform has been called. The resulting record format can be different from the input.

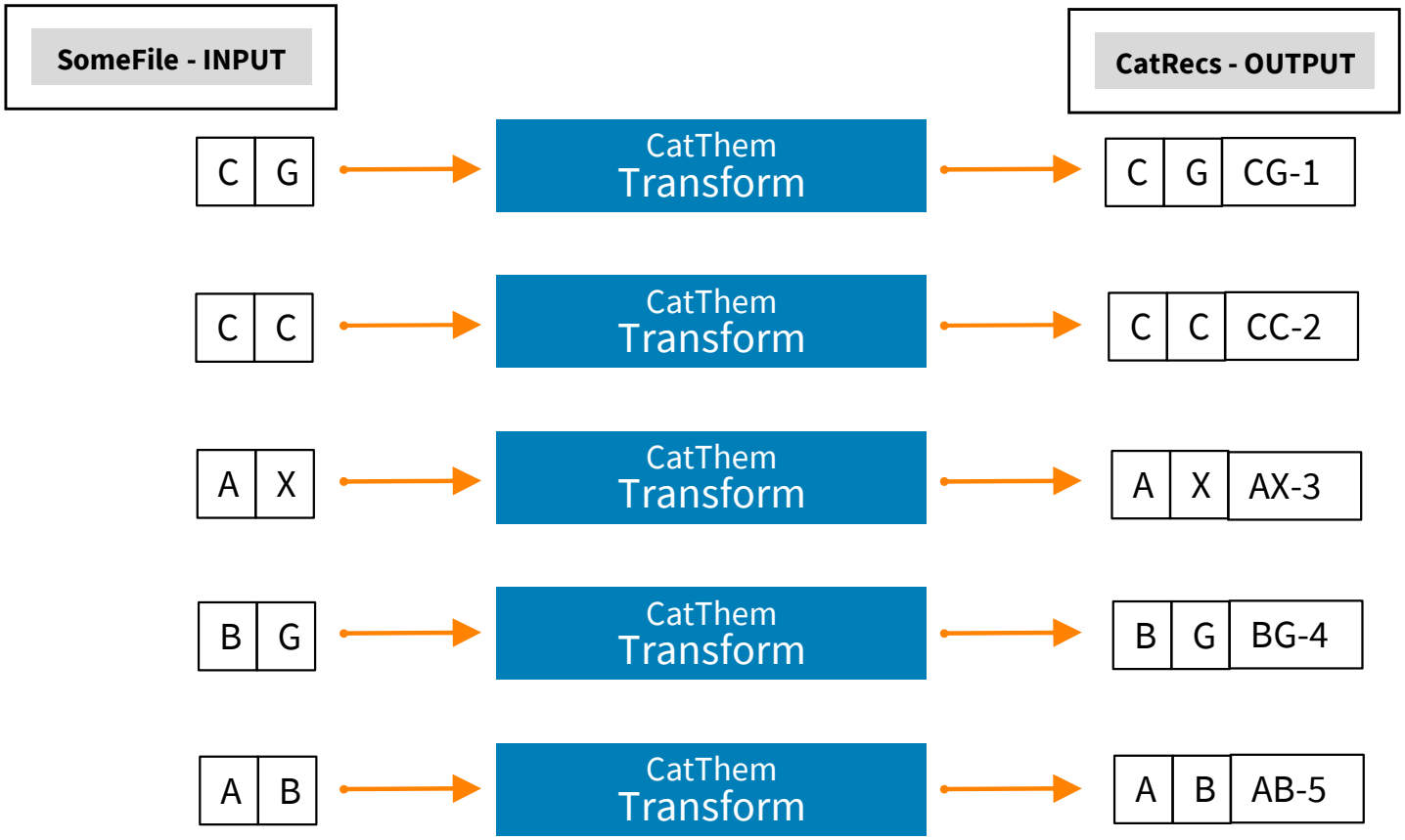
PROJECT Example:

```
YP_SlimRec := RECORD
    INTEGER         seq;
    STRING125   business_name;
    STRING100   orig_street;
    STRING75    orig_city;
    STRING2         orig_state;
    STRING9         orig_zip;
    STRING10    orig_phone10;
END;
YP_SlimRec SlimYp(YellowPagesRec Le,INTEGER cnt) := TRANSFORM
    SELF.seq := cnt;
    SELF := Le;
END;
YP_Slim := PROJECT(File_YellowPages, SlimYP(LEFT, COUNTER));
```

PROJECT Training Example:

Labs.PROJECT_Example

PROJECT Functional Examples Diagram



PERSIST Workflow Service:

PERSIST

name := *exp* : **PERSIST(***file* **[,***cluster***] [,CLUSTER(***target***)] [,EXPIRE(***days***)]);**

- *name* – The definition name.
- *exp* – The definition expression.
- *file* – A string constant defining the logical filename in which the results of the definition's *expression* are stored.
- *cluster* – A string constant specifying the name of the cluster on which to re-build the *name* if/when necessary.
- CLUSTER – Specifies writing the *file* to another cluster.
- *target* – The cluster to write the *file* to.

The **PERSIST** service stores the result of the *expression* in the specified *file* so it remains permanently available for use. This is particularly useful for recordset definitions that perform large, expensive data manipulation sequences.

Large processes can be segmented using PERSIST to both facilitate automation and minimize re-run time. PERSISTed definitions are re-computed only when the code or data used to compute them have changed.

DATA Standardization

- MODULE Structure
- Transforming data using TABLEs
- Explicit CASTING
- Measuring size of RECORDs and Fields

SIZEOF Function:

SIZEOF(*data*)

- *data* – The name of a dataset, RECORD structure, a fully-qualified field name, or a constant string expression.

The **SIZEOF** function returns the total number of bytes defined for storage of the specified *data* structure or field.

```
MyRec := RECORD
 INTEGER5        F2;
 QSTRING12       F3;
 VARSTRING12     F4;
END;
MyData := DATASET([{1,33333333333, 'A',v'A'}],MyRec);
SIZEOF(MyRec);                  //result is 27
SIZEOF(MyData);                 //result is 27
SIZEOF(MyData.F2);              //result is 5
SIZEOF(MyData.F3);              //result is 9  -12 chars stored in 9 bytes
SIZEOF(MyData.F4);              //result is 13 -12 chars plus null terminator
SIZEOF('abc' + '123');          //result is 6
```

Data Standardization Issues:

Data Standardization: transform using a TABLE!

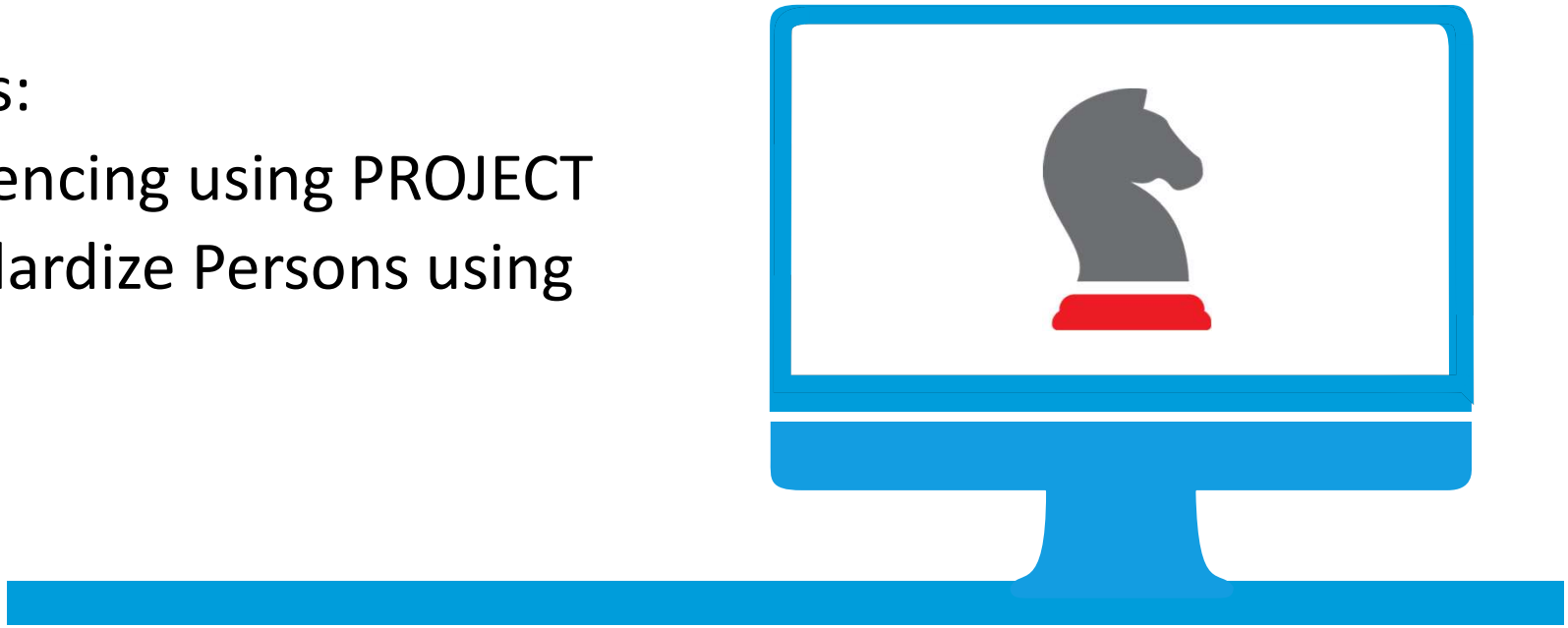
- Wrap TABLE in MODULE, define Layout and File
- Do for both Persons (STD_Persons) and Accounts (STD_Accounts)
- Default values in Layout can be used as transformation expressions:
  STRING15 FirstName := **std.Str.ToUpperCase**($.UID_Persons.FirstName);
- Dates
  STRING8 to UNSIGNED4
- Names
  Convert all to Upper Case
- Zip
  STRING5 to UNSIGNED3
- What about the Middle Name?
- What about MaritalStatus and Dependent Count?

Lab Exercises

Sequencing and Data Standardization

Do exercises:

- 7 – Sequencing using PROJECT
- 8 – Standardize Persons using TABLE



Joining Files

- JOIN Function
- INTFORMAT Function

JOIN Function

JOIN(*leftset*, *rightset*, *condition*[, *transform*] [,*type*] [,*flag*]**)**

- *leftset* – The LEFT set of records to process. This should be the larger of the two files.
- *rightset* – The RIGHT set of records to process.
- *condition* – The expression that specifies how to match records between the *leftset* and *rightset*.
- *transform* –The TRANSFORM function to call.
- *type* – The type of join to perform (default is an inner join).
- *flag* – Any option to specify exactly how the JOIN operation executes.

The JOIN function processes through the records in the *leftset* and *rightset,* evaluating the *condition* to find matching records. The *transform* function executes on each pair of matching records.

The *transform* function must take at least 2 parameters: a LEFT record formatted like the *leftset*, and a RIGHT record formatted like the *rightset*. These may be different formats, and the resulting record set can be a different format.

JOIN Types

INNER – All matching records from both record sets.

LEFT OUTER – At least one record for every record in the *leftset*.

RIGHT OUTER – At least one record for every record in the *rightset*.

FULL OUTER – At least one record for every record in both the *leftset* and *rightset*.

LEFT ONLY – One record for every record in the *leftset* for which there is no matching record in the *rightset*.

RIGHT ONLY – One record for every record in the *rightset* for which there is no matching record in the *leftset*.

FULL ONLY – One record for every record in the *leftset* and *rightset* for which there is no matching record in the opposite record set.

JOIN Flags

LOOKUP – Copies *rightset* to each node for a Many-0/1 (without MANY) or Many-0/Many lookup.

ALL – Copies *rightset* to each node for a Many-0/Many lookup – no equality portion of *condition* required.

SMART – Specifies to use an in-memory lookup when possible, but use a distributed join if the right dataset is large.

JOIN Example:

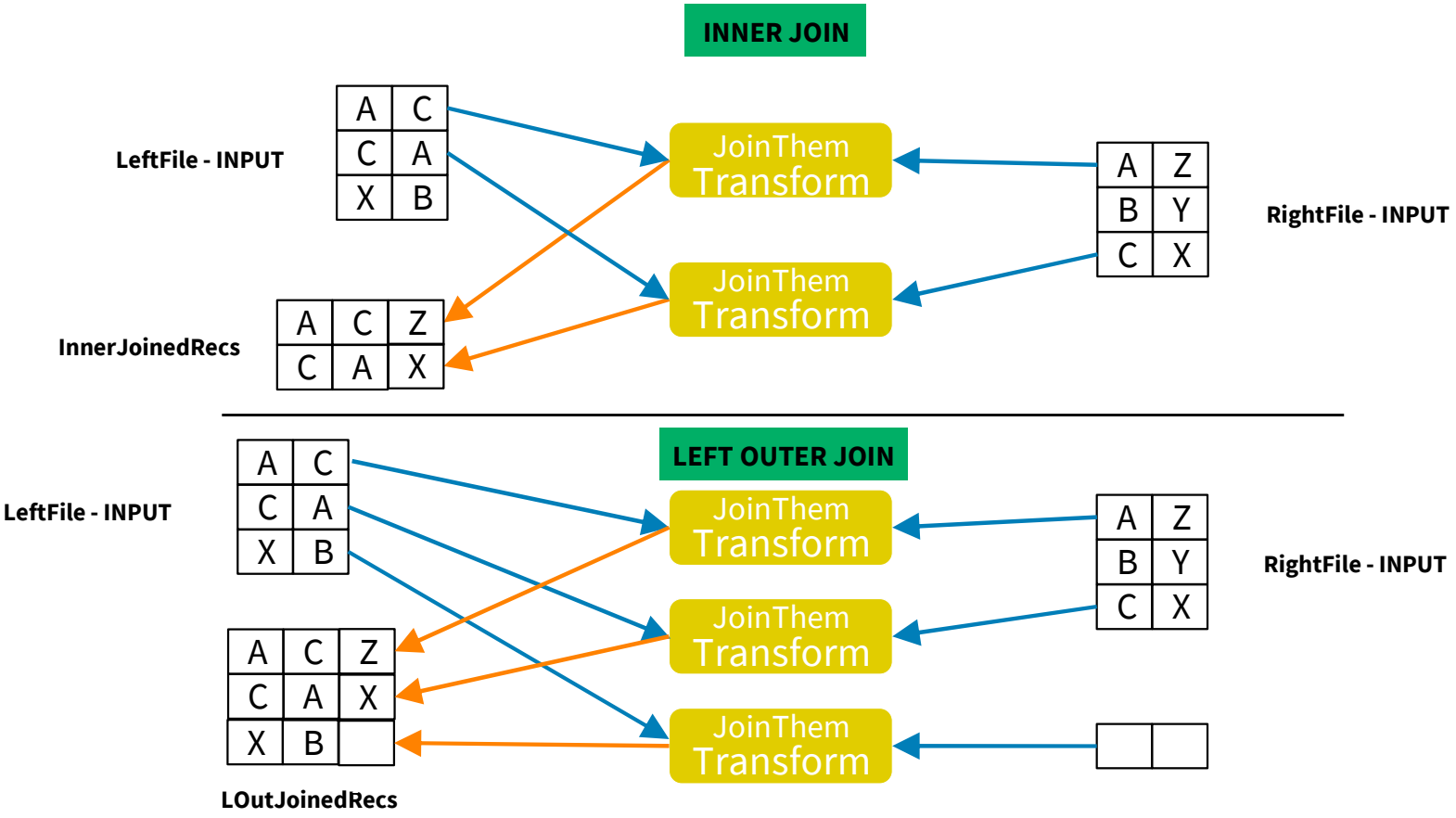
```
Layout_Comp_Temp EnhancePhones(Layout_Comp_Temp Le, Layout_Bus_Header Ri) := TRANSFORM
        SELF.phone_score := Ri.phone_score;
        SELF.phone := IF( (INTEGER)Ri.phone <> 0, Ri.phone, Le.phone);
        SELF := Le;
END;

Company_Enh := JOIN(Company_Seq_Dist,
                    Gong_Business_Dist,
                    LEFT.zip = RIGHT.zip AND
                    LEFT.address = RIGHT. address AND
                    STD.CompSimilar100(LEFT.co_name,RIGHT.co_name) <= 35,
                    EnhancePhones(LEFT, RIGHT),
                    LEFT OUTER);
```

JOIN Training Example:

- Labs.JOIN_Example

JOIN Functional Diagram:



INTFORMAT Function:

INTFORMAT(*expression, width, mode***)**

- *expression* – The *expression* that specifies the integer value to format into a string.
- *width* – The number of digits to which to format the value.
- *mode* – The format type: 1 = zero fill.

The **INTFORMAT** function returns the value of the *expression* formatted as a string of *width* characters.

```
SeqNum := 55693;
SeqNum10 := INTFORMAT(SeqNum, 10, 1); // Result is ' 0000055693'
```

Lab Exercises

Do exercises:

- 9 – Create a “SLIM Person” File using JOIN
- 10 – reJOIN the SLIM_Persons and Lookup_CSZ

The Append Operator

Recordset Operators

Append +

```
CompCombined1 := Comp1 + Comp2 + Comp3;
```

Append & (maintains record order)

```
CompCombined2 := Comp1 & Comp2 & Comp3;
```

SORT Function

SORT(*recordset*, *value***)**

- *recordset* – The set of records to process.
- *value* – An expression or key field in the *recordset* on which to sort. A leading minus sign (-) indicates a descending order sort. You may have multiple *value* parameters to indicate sorts within sorts.

The **SORT** function sorts the *recordset* according to the *values* specified. Any number of *value* parameters may be supplied, with the leftmost being the most significant sort criteria. A leading minus sign (-) on any *value* parameter indicates a descending sort for that one parameter.

```
Business_Contacts_Sort := SORT(Business_Contacts_Dist,
                                company_name, company_title,
                                lname, fname, mname, name_suffix,
                                zip, prim_name, prim_range);
HighestBals := SORT(ValidBalTrades, -trades.trd_bal);
UCC_Sort    := SORT(UCC_Dist, file_state, orig_filing_num);
```

SORT Example

- Labs.SORT_Example

DEDUP Function

DEDUP(*recset* [,*condition* [,**ALL**][**BEST** (*sort-list*)],[,**KEEP** *n*] [,*keep*]]**)**

- *recset* – The set of records to process.
- *condition* – The expression that defines “duplicate” records.
- **ALL** –Matches all records to each other using the *condition*, not just adjacent records.
- **BEST**-Provides additional control over which records are retained from a set of "duplicate" records. The first in the <sort-list> order of records are retained. BEST cannot be used with a KEEP parameter greater than 1.
- **KEEP** *n* –Specifies keeping *n* number of duplicates. The default is 1.
- *keep* –LEFT (the default) keeps the first and RIGHT keeps the last.

The **DEDUP** function removes duplicate records from the *recordset*. The *condition* defines what constitutes “duplicate” records.

```
Company_Dedup := DEDUP(Company_Init(zip<>0), company_name,
                                    zip, prim_name, prim_range, sec_range, ALL);
```

LEFT and RIGHT Keywords

LEFT.*field* / RIGHT.*field*

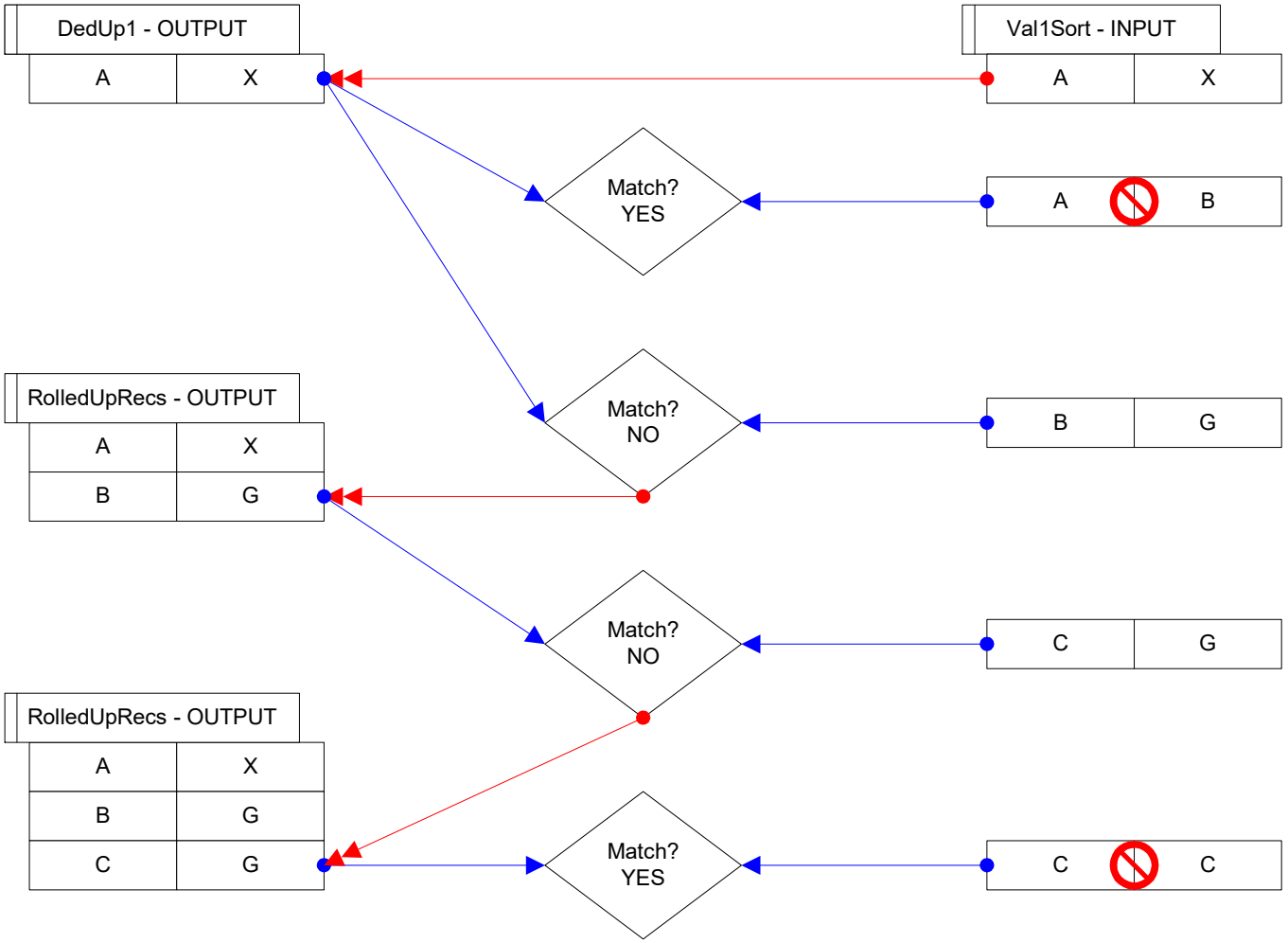
The **LEFT** and **RIGHT** keywords qualify which record a *field* is from in those operations that use pairs of records, such as DEDUP, JOIN, etc. This helps prevent any code ambiguity for the compiler.

```
SortedRecs := SORT(Person,per_last_name,per_first_name);
DeDupedRecs := DEDUP(SortedRecs,
        LEFT. per_last_name = RIGHT. per_last_name AND
        LEFT. per_first_name = RIGHT. per_first_name);
```

DEDUP Example

- Labs.DEDUP_Example
- Labs.DEDUP_All_Example

Simple DEDUP Functional Example Diagram



Lab Exercises

Do exercise:

- 11 – Compare results using APPEND, SORT and DEDUP

End of Day 2 Workshop:

More to come!!

See you tomorrow for Day 3!

Thanks for attending!

✓**Download it all at:**
https://github.com/hpccsystems-solutions-lab/DefinitiveHPCCSystems

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

HPCC SYSTEMS