HPCC SYSTEMS®

Advanced ECL (Part 1) – Working With Relational Data

Lesson 1

Introduction and Lab Exercise 1: Loading the Course Data

LexisNexis® RISK SOLUTIONS

RELX Group

Introduction and Overview:

- The Advanced ECL class contains a set of hands-on exercises that showcase ECL functions that are designed to work with Relational Data.

- We will explore:
  - Nested Child Dataset Definitions (RECORD/DATASET)
  - NORMALIZE and DENORMALIZE
  - Complex Multi-level Relational Querying

Training Data:

Training Database:
(a 3-level hierarchical relational database):

- **People**
  - **Vehicle**
  - **Property**
    - **Taxdata**

Lab Exercise 1:

1. Download and extract training data located on main course page (OnlineAdvancedECL.ZIP)
2. Upload extracted files to your target landing zone
3. Spray four (4) files to THOR – Spray Fixed

Source Filename	**Record Length**	**Destination Label**
OnlinePeople	82	ONLINE::XXX::AdvECL::People (*XXX = your initials*)
OnlineProperty	154	ONLINE::XXX::AdvECL::Property
OnlineTaxdata	68	ONLINE::XXX::AdvECL::Taxdata
OnlineVehicle	187	ONLINE::XXX::AdvECL::Vehicle

✓ We sprayed in the Intro to ECL class – review if needed.

✓Spray Options:
-Overwrite and Replicate ON (checked)
-No Split and Compress OFF (unchecked)

Lesson Completed!

Proceed to Lesson 2: Defining the Data

Advanced ECL (Part 1) – Working With Relational Data

Lesson 2

Defining the Data

RECORD Structure Review:

name **:= RECORD [(** *baserec* **)] [, MAXLENGTH(** *length* **)] [, LOCALE(** *locale* **)]**
        fields
END;

- ✓ *name* - The name of the RECORD structure.
- ✓ *baserec* - Optional. The name of a RECORD structure from which to inherit all fields. Any RECORD structure that inherits the *baserec* fields in this manner becomes compatible with any TRANSFORM function defined to take a parameter of *baserec* type (the extra *fields* will, of course, be lost).
- ✓ **MAXLENGTH -** Optional. Maximum characters in the RECORD structure or field. On the RECORD structure, it overrides any MAXLENGTH on a field definition, which overrides any MAXLENGTH specified in the TYPE structure if the *datatype* names an alien data type. This is typically used to define the maximum length of variable-length records. If omitted, the default is 4096 bytes.
- ✓ **LOCALE** - Optional. Specifies the Unicode *locale* for any UNICODE fields.
- ✓ *fields* – Field declarations.

Field Definitions:

datatype *identifier* **[:=** *defaultvalue***] [{** *modifier* **}] ;**
identifier **:=** *defaultvalue* **;**
defaultvalue **;**
sourcefield **;**
recstruct **[** *identifier* **] ;**
sourcedataset **;**
***childdataset identifier* [{ *modifier* }];**

- ✓ *datatype* - The value type of the data field.
- ✓ *identifier* - The name of the field.
- ✓ *defaultvalue* - Optional. An expression defining the source of the data.
- ✓ *modifier* - Optional. One of the keywords listed in the **Field Modifiers**.
- ✓ *sourcefield* - The name of a previously defined data field, which implicitly provides the *datatype*, *identifier*, and *defaultvalue* for the new field—all inherited from the existing field.
- ✓ *recstruct* - The name of a previously defined RECORD structure.
- ✓ *sourcedataset* - The name of a previously defined DATASET or derived recordset attribute. See the **Field Inheritance** section in the LRM.
- ✓ ***childdataset* - A child DATASET declaration.**

HPCC SYSTEMS®

DATASET Review:

name **:= DATASET(** *file, recorddef,* **THOR** *thoroptions***);**
name **:= DATASET(** *file, recorddef,* **CSV [(** *csvoptions* **)]);**
name **:= DATASET(** *file, recorddef,* **XML(** *path* **));**
name **:= DATASET(** *file, recorddef,* **JSON(** *rowpath* **));**

name – The attribute name by which the file subsequently referenced.
file – A string constant containing the logical filename.
recorddef – The RECORD structure of the dataset.
thoroptions – Options specific to THOR/FLAT datasets.
csvoptions – Options specific to CSV datasets.
path – The XPATH to the XML row tag.
command – The program call that will produce the dataset.

Normalized RECORD and DATASET:

```
EXPORT Layout_Company := RECORD
  STRING8         sic_code;
  STRING120       company_name;
  STRING10        prim_range;
  STRING2         predir;
  STRING28        prim_name;
  STRING4         addr_suffix;
  STRING2         postdir;
  STRING5         unit_desig;
  STRING8         sec_range;
  STRING25        city;
  STRING2         state;
  STRING5         zip;
  STRING4         zip4;
END;
EXPORT File_Company_List := DATASET('BASE::Company_List', Layout_Company, THOR);
```

Lab Exercise 2:

Define the data (create a RECORD and DATASET for each):

- ✓Taxdata (File_Taxdata)
- ✓Property (File_Property)
- ✓Vehicle (File_Vehicle)
- ✓People (File_People)

These ECL files are available in the *HelperFunctions.ZIP* that you downloaded from the Course web page; simply copy these files to your target repository folder, modify the DATASET filename to match the files that you sprayed in Exercise 1, and write a simple query to test the outputs.

Nested Child Dataset RECORD:

```
ChildRecord := RECORD
    UNSIGNED4       person_id;
    STRING20        per_surname;
    STRING20        per_forename;
END;
ParentRecord := RECORD
    UNSIGNED8       id;
    STRING20        address;
    STRING20        CSZ;
    STRING10        postcode;
    UNSIGNED2       numPeople;
    DATASET(ChildRecord)   children {MAXCOUNT(20)};
END;
EXPORT File_Address := DATASET('CLASS::Adr_List', ParentRecord, THOR);
```

Lesson Completed!

Proceed to Lesson 3: Denormalization

Advanced ECL (Part 1) – Working With Relational Data

Lesson 3

Using DENORMALIZE

Working with Child Datasets

- ✓ DATASET fields in RECORD Structures
- ✓ Review of TRANSFORM
- ✓ DENORMALIZE Function

TRANSFORM Structure:

```
resulttype  funcname( parameterlist ) := TRANSFORM
   SELF.outfield := transformation;
END;
```

- ✓ *resulttype* – The name of a RECORD structure attribute 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*.

DENORMALIZE Function:

DENORMALIZE(*parentoutput,childrecset,condition,transform***)**

- ✓ *parentoutput* – The set of parent records already formatted as the result of the combination.
- ✓ *childrecset* – The set of child records to process.
- ✓ *condition* – An expression that specifies how to match records between the parent and child records.
- ✓ *transform* – The TRANSFORM function to call.

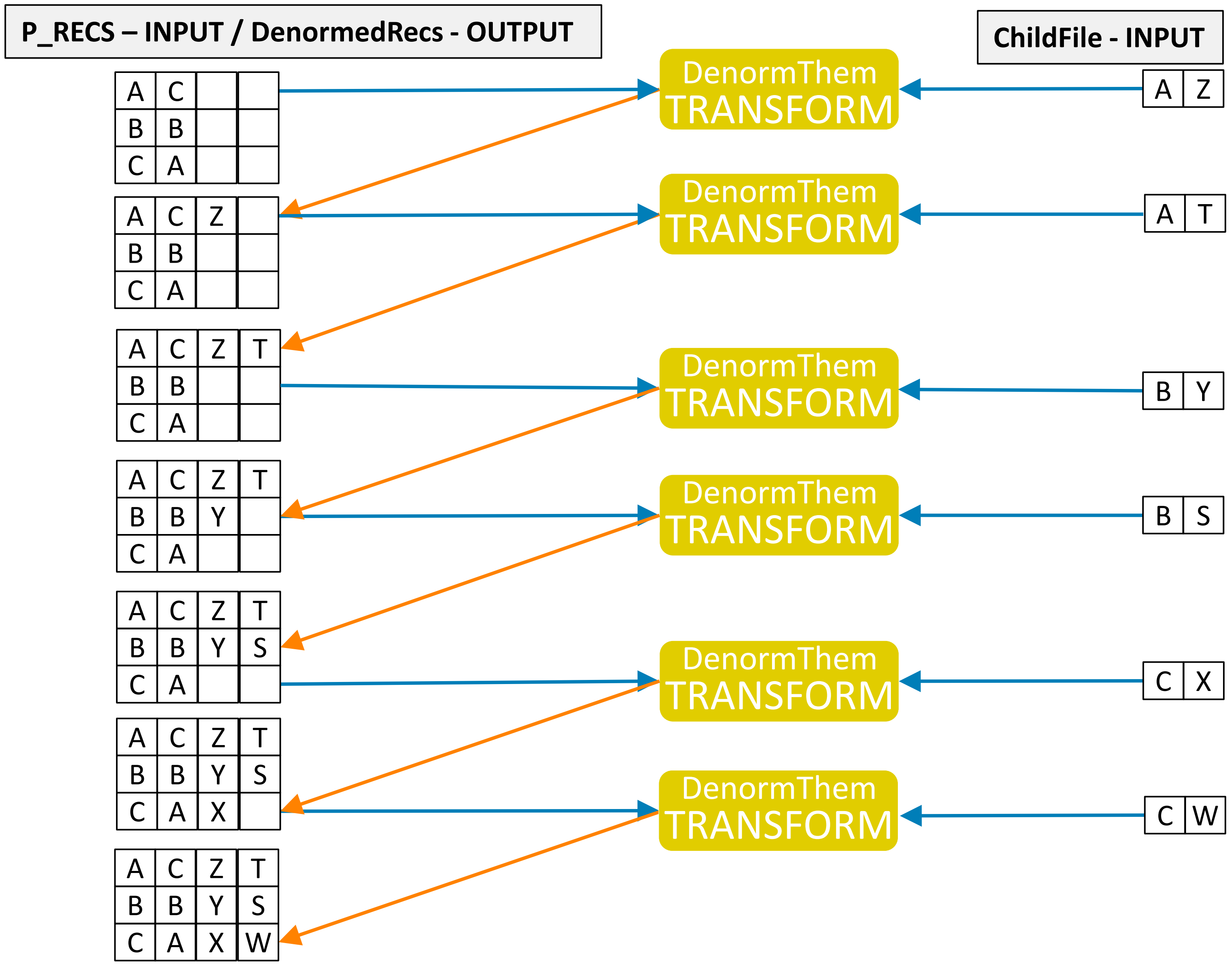
DENORMALIZE Example:

```
Layout_rev    DenormRevenue(Layout_rev Le, TempCustRec Ri) := TRANSFORM
        SELF.revenue_1 := IF(Ri.UnitCode = '01', Ri.revenue_total,Le.revenue_1);
        SELF.revenue_2 := IF(Ri.UnitCode = '02', Ri.revenue_total,Le.revenue_2);
        SELF.revenue_3 := IF(Ri.UnitCode = '03', Ri.revenue_total,Le.revenue_3);
        SELF.revenue_4 := IF(Ri.UnitCode = '04', Ri.revenue_total,Le.revenue_4);
        SELF := Le;
END;



DenormedRecs := DENORMALIZE(Revenue, Customer,
                            LEFT.BusID = RIGHT.BusID AND  LEFT.SubSidiary = RIGHT.SubSidiary ,
                            DenormRevenue(LEFT,RIGHT));
```

DENORMALIZE Training Example:

- ✓Training_Examples.DENORMALIZE_Example
- ✓View/download from ECL Playground

DENORMALIZE Functional Diagram:



DENORMALIZE Training Example:

- ✓Training_Examples. NORM_DENORM_ChildDatasets_Example
- ✓View/download from ECL Playground

Lab Exercises – A look ahead:

DENORMALIZE the data

- ✓ Do exercises:
  - ✓ 3 – Denormalize People and Vehicle
    - ✓ Get Parent file ready – simple PROJECT
    - ✓ Empty Set Usage
    - ✓ Add new field to keep child count!
  - ✓ 4 – Denormalize Property and Taxdata
    - ✓ Very similar to 3 – DATASET names change
  - ✓ 5 – Denormalize People/Vehicle and Property/Taxdata
    - ✓ People/Vehicle parent to Property/Taxdata child
  - ✓ 6 – DEFINE the new DENORMALIZED People table: Create a new DN module to use in all subsequent queries

Lesson Completed!

Proceed to Lesson 4:
…for the first of four (4) DENORMALIZE Exercises

HPCC SYSTEMS®

Advanced ECL (Part 1) – Working With Relational Data

Lesson 4

Denormalizing People and Vehicles

Training Data:

Training Database:
(a 3-level hierarchical relational database):

- **People**
  - **Vehicle**
  - **Property**
    - **Taxdata**

Lab Exercise 3:

DENORMALIZE People and Vehicle:

- ✓Get Parent file ready – simple PROJECT
- ✓Empty Set Usage
- ✓Add new field to keep child count!

Lab Exercise 3:

Exercise Spec:
Create an ECL MODULE definition file that uses DENORMALIZE and also create a Builder Window Runnable file that shows the PERSISTED output with each People record and all of its related *Vehicle* records in a single Child Dataset format output.

Requirements:

1. The ECL MODULE definition file name to create for this exercise is:
   DenormPeopleVehicles
2. The PERSIST filename must start with *~CLASS*, followed by *your initials* followed by *PERSIST::PeopleVehicles* as in this example:
   ~CLASS::XX::PERSIST::PeopleVehicles
3. Use an EXPORT DENORMALIZE function definition named **File** to add the related child *Vehicle* information to the new combined RECORD definition.
4. Create a Builder Runnable Window named **BWR_OutPeopleVehicles** to implicitly OUTPUT the **DenormPeopleVehicles.File** recordset.
5. Create a new EXPORT RECORD definition in the MODULE named **Layout_PeopleVehicles** with the combined People and Vehicle information. The vehicle information will need to be a nested child dataset.

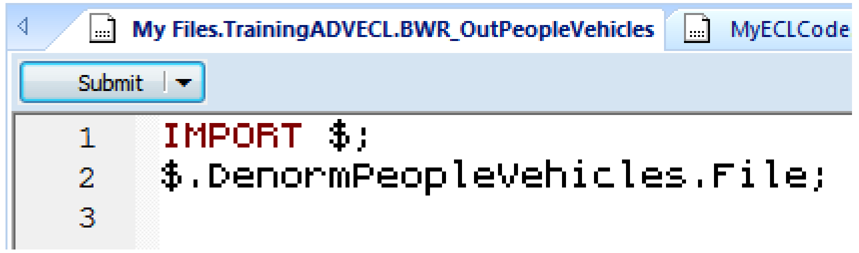
Best Practices Hints

1. Use the RECORD structure you defined for the EXPORTed *Vehicle* definition as the nested child dataset.
2. Use a PROJECT to load your normalized RECORD definition with the parent *People* data.
3. Use a DENORMALIZE statement to add the related child *Vehicle* information to the new combined RECORD definition.

Result Comparison
Use the Builder or BWR window to execute the query and generate the output. A COUNT of the recordset output should still verify **27,994** People records.

Lab Exercise 3 Solution:

```
IMPORT $;
EXPORT DenormPeopleVehicles := MODULE
 EXPORT Layout_PeopleVehicles := RECORD
  $.File_People.Layout;
  UNSIGNED1 ChildVCount;
  DATASET($.File_Vehicle.Layout) VehicleRecs{MAXCOUNT(20)};
 END;

 Layout_PeopleVehicles ParentMove($.File_People.Layout Le) := TRANSFORM
  SELF.ChildVCount := 0;
  SELF.VehicleRecs := [];
  SELF := Le;
 END;

 ParentOnly := PROJECT($.File_People.File, ParentMove(LEFT));

 Layout_PeopleVehicles ChildMove(Layout_PeopleVehicles Le,$.File_Vehicle.Layout Ri,INTEGER Cnt):= TRANSFORM
  SELF.ChildVCount := Cnt;
  SELF.VehicleRecs := Le.VehicleRecs + Ri;
  SELF := Le;
 END;

 EXPORT File := DENORMALIZE(ParentOnly,$.File_Vehicle.File,LEFT.id = RIGHT.personid, ChildMove(LEFT,RIGHT,COUNTER))
               : PERSIST('~CLASS::XXX::PERSIST::PeopleVehicles');
END;
```



```
IMPORT $;
$.DenormPeopleVehicles.File;
```

Lesson Completed!

Proceed to Lesson 5:

Denormalizing Property and TaxData

Advanced ECL (Part 1) – Working With Relational Data

Lesson 5

Denormalizing Property and Taxdata

Training Data:

Training Database:
(a 3-level hierarchical relational database):

- **People**
  - **Vehicle**
  - **Property**
    - **Taxdata**

Lab Exercise 4:

DENORMALIZE Property and Taxdata:

- ✓Very similar to Lab Exercise 3
- ✓Get Parent file ready – use a simple PROJECT
- ✓Empty Set Usage
- ✓Add new field to keep child count!

Lab Exercise 4:

Exercise Spec:
Create an ECL MODULE Definition file that uses DENORMALIZE, and also, create a Builder Window Runnable file that shows the output file with each Property record and all its related TaxData records in a single Child Dataset format output.

Requirements:

1. The ECL MODULE definition file name to create for this exercise is: **DenormProp**
2. Add an EXPORT definition in the MODULE named **File** to define the recordset of the denormalized Property and Taxdata tables.
3. Create a Builder Runnable Window named **BWR_OutDenormProp** to implicitly output the DenormProp.File recordset.
4. The PERSIST filename must start with ~CLASS, followed by your initials followed by PERSIST::PropTax as in this example:
   ~CLASS::XXX::PERSIST::PropTax
5. Create a new EXPORT RECORD definition in the MODULE named **Layout_PropTax** with the combined Property and Taxdata information. The tax data information will need to be a nested child dataset.

Best Practices Hints

1. Use the RECORD structure you defined and EXPORTed for Taxdata as the nested child dataset.
2. Use a PROJECT to load your normalized RECORD definition with the parent Property data.
3. Use a DENORMALIZE statement to add the related child Taxdata information to the new combined RECORD definition.

Result Comparison
Use the Builder window to execute the query and generate the output. A COUNT of the recordset output should still verify 166,758 Property records.

Lab Exercise 4 Solution:

```
IMPORT $;
EXPORT DeNormProp := MODULE
 EXPORT Layout_PropTax := RECORD
  $.File_Property.Layout;
  UNSIGNED1 ChildTaxCount;
  DATASET($.File_Taxdata.Layout) TaxRecs{MAXCOUNT(20)};
 END;
 Layout_PropTax ParentMove($.File_Property.Layout Le) := TRANSFORM
  SELF.ChildTaxCount := 0;
  SELF.TaxRecs       := [];
  SELF := Le;
 END;
 ParentOnly := PROJECT($.File_Property.File, ParentMove(LEFT));
 Layout_PropTax ChildMove(Layout_PropTax Le, $.File_Taxdata.Layout Ri, INTEGER Cnt):=TRANSFORM
  SELF.ChildTaxCount := Cnt;
  SELF.TaxRecs       := Le.TaxRecs + Ri;
  SELF := Le;
 END;
 EXPORT File := DENORMALIZE(ParentOnly,
                           $.File_TaxData.File,
                           LEFT.propertyid = RIGHT.propertyid,
                           ChildMove(LEFT,RIGHT,COUNTER))
                           :PERSIST('~CLASS::XXX::PERSIST::PropTax');
END;
```

Lesson Completed!

Proceed to Lesson 6:

Denormalizing
People and Property

Advanced ECL (Part 1) – Working With Relational Data

Lesson 6

Denormalizing People and Property

Training Data:

Training Database:
(a 3-level hierarchical relational database):

- People
  - Vehicle

Exercise 3

- Property
  - Taxdata

Exercise 4

Exercise 5

- People
  - Vehicle
  - Property
    - Taxdata

Lab Exercise 5:

- ✓ Denormalize People/Vehicle and Property/Taxdata
  - ✓ People/Vehicle parent to Property/Taxdata child
  - ✓ Get Parent file ready – simple PROJECT
  - ✓ Empty Set Usage
  - ✓ Add new field to keep child count!

Lab Exercise 5:

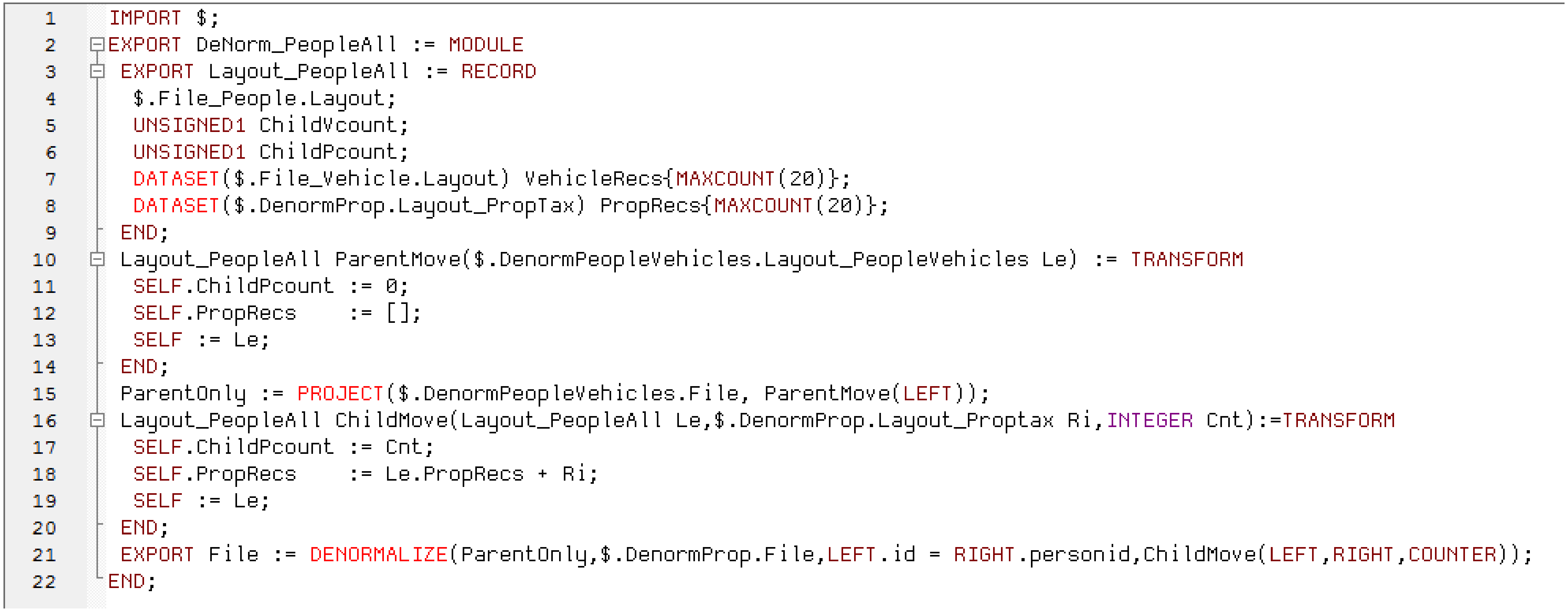
Exercise Spec:
Create an ECL MODULE Definition file that uses DENORMALIZE. Also, create a Builder Window Runnable file that produces (generates) an output file with each People record and all of its related Property and Vehicle records in a Child Dataset formats.

Requirements:
1. The ECL EXPORT definition file name to create for this exercise is: **Denorm_PeopleAll**
2. The OUTPUT filename must start with *~ONLINE*, followed by *your initials* followed by *OUT::PeopleAll* as in this example:
   ~ONLINE::XXX::OUT::PeopleAll
3. Add the RECORD definition and DATASET based on the **DenormPeopleVehicles** definition that you created and exported in Lesson 4 (Lab Exercise 3).
4. Build a completely denormalized People file by combining the EXPORTed RECORD definition in Lesson 5 (Lab Exercise 4) with the EXPORTed RECORD definition in Lesson 4. Name this new composite EXPORT RECORD definition **Layout_PeopleAll**.
5. Create a Builder Runnable Window in your repository named **BWR_OutPeopleAll** to simply OUTPUT the **Denorm_PeopleAll** recordset to disk.

Best Practices Hint
1. Use the RECORD structure you defined in Lesson 4 for the People definition that contains the nested child dataset Vehicle information.

Result Comparison
Use the Builder or Builder Window Runnable (BRW) window to execute the query, then look in the ECL Watch Logical Files list to find the newly generated file and ensure that its size is approximately 105,950,795 bytes
and that there are still 27,994 records.

Lab Exercise 5 Solution:



```
IMPORT $;
EXPORT DeNorm_PeopleAll := MODULE
 EXPORT Layout_PeopleAll := RECORD
  $.File_People.Layout;
  UNSIGNED1 ChildVcount;
  UNSIGNED1 ChildPcount;
  DATASET($.File_Vehicle.Layout) VehicleRecs{MAXCOUNT(20)};
  DATASET($.DenormProp.Layout_PropTax) PropRecs{MAXCOUNT(20)};
 END;
 Layout_PeopleAll ParentMove($.DenormPeopleVehicles.Layout_PeopleVehicles Le) := TRANSFORM
  SELF.ChildPcount := 0;
  SELF.PropRecs    := [];
  SELF := Le;
 END;
 ParentOnly := PROJECT($.DenormPeopleVehicles.File, ParentMove(LEFT));
 Layout_PeopleAll ChildMove(Layout_PeopleAll Le,$.DenormProp.Layout_Proptax Ri,INTEGER Cnt):=TRANSFORM
  SELF.ChildPcount := Cnt;
  SELF.PropRecs    := Le.PropRecs + Ri;
  SELF := Le;
 END;
 EXPORT File := DENORMALIZE(ParentOnly,$.DenormProp.File,LEFT.id = RIGHT.personid,ChildMove(LEFT,RIGHT,COUNTER));
END;
```

Lesson Completed!

Proceed to Lesson 7:

Defining Denormalized Datasets

HPCC SYSTEMS®

Advanced ECL (Part 1) – Working With Relational Data

Lesson 7

Define the Denormalized People Dataset

Training Data:

Training Database:
(a 3-level hierarchical relational database):

- **People**
  - **Vehicle**
  - **Property**
    - **Taxdata**

Denormalized Training Data:

People	Vehicle	Vehicle	•••
Vehicle	Vehicle	Property	•••
Taxdata	Taxdata	Taxdata	•••
Property	Taxdata	Taxdata	

Lab Exercise 6:

Exercise Spec:
DEFINE the new DENORMALIZED People table. Create a new **File_PeopleAll** ECL MODULE to use in all subsequent queries. Use the RECORD definitions that you created in the previous exercises in this section, and combine them *all* into a single module that you can use to reference the nested child datasets in subsequent queries in the next chapter.

Requirements:

1. The ECL definition file name to create for this exercise is **File_PeopleAll.**
2. Create a MODULE structure, and then within it, four (4) EXPORTed definitions that reference the People table and the other 3 nested child DATASETs.
3. Use the EXPORTed RECORD structure from the previous lab exercise in hour DATASET statement.
4. Create a DATASET attribute which is exported that references the denormalized People table that you created in Exercise 5.

Best Practices Hint
Use object syntax to "drill down" to all of the nested child datasets (Vehicle, Property, and Taxdata).

Result Comparison
Use a new Builder window to test the output of your new EXPORTed DATASETs. Verify that there are no errors at runtime and that the results look reasonable. **You must complete this exercise before proceeding to the next lesson.**

Lab Exercise 6 Solution:

```
IMPORT $;
EXPORT File_PeopleAll := MODULE
  EXPORT People   := DATASET('~CLASS::BMF::OUT::PeopleAll',
                                $.DeNorm_PeopleAll.Layout_PeopleAll,THOR);
  EXPORT Vehicle  := People.Vehiclerecs;
  EXPORT Property := People.PropRecs;
  EXPORT Taxdata  := People.PropRecs.TaxRecs;
END;
```

Lesson Completed!

Proceed to Lesson 8:

Querying Relational Data

HPCC SYSTEMS®

Advanced ECL (Part 1) – Working With Relational Data

Lesson 8

Querying Relational Datasets

Querying Relational Data:

Implicit Dataset Relationality
(Nested child datasets):

- ✓ Parent record fields are always in memory when operating at the level of the Child
- ✓ You may only reference the related *set* of Child records when operating at the level of the Parent
    - ➢ People
        - ➢ Vehicle
        - ➢ Property
            - ➢ Taxdata

Helper Functions:

ThisYear.ECL

```
EXPORT ThisYear := 2018; //sets the base year
```

IsValidAmount.ECL

```
EXPORT IsValidAmount(integer amt) := amt BETWEEN 1 AND 9999998;
```

IsValidYear.ECL

```
IMPORT $;
EXPORT IsValidyear(integer pyear) := pyear > 1900 and pyear <= $.ThisYear ;
```

Helper Functions:

YearsOld.ECL

```
IMPORT $;
EXPORT YearsOld(integer4 datex) := (INTEGER4) IF($.IsValidYear(datex),
                                                 ($.ThisYear - datex),
                                                 datex);
```

Limit_Value.ECL

```
EXPORT Limit_Value(n,maxval) := IF(n > maxval, maxval, n);
```

Helper Functions:

Z2JD.ECL (Zulu to Julian Date)

```
EXPORT Z2JD(STRING8 Zdate) := FUNCTION
// adapted from an algorithm described here:
// http://quasar.as.utexas.edu/BillInfo/JulianDatesG.html
  A(Y)     := TRUNCATE(Y/100);
  B(Aval) := TRUNCATE(Aval/4);
  C(Y)     := 2-A(Y)+B(A(Y));
  E(Y)     := TRUNCATE(365.25 * (Y+4716));
  F(M)     := TRUNCATE(30.6001 * (M+1));
  Yval     := IF( (INTEGER1)(Zdate[5..6]) < 3,(INTEGER2)(Zdate[1..4])-1,(INTEGER2)(Zdate[1..4]));
  Mval     := IF( (INTEGER1)(Zdate[5..6]) < 3,(INTEGER1)(Zdate[5..6])+12,(INTEGER1)(Zdate[5..6]));
  Dval     := (INTEGER1)(Zdate[7..8]);
  RETURN IF(Zdate='',0,TRUNCATE(C(Yval) + Dval + E(Yval) + F(Mval)- 1524.5));
END;
```

HPCC SYSTEMS®

EVALUATE Function:

EVALUATE(*onerecord, value***)**

- ✓ *onerecord* – A recordset consisting of a single record.
- ✓ *value* – The value to return. This may be any expression yielding a single value.

```
ValidBalTrades       := trades(ValidMoney(trades.trd_bal));
HighestBals          := SORT(ValidBalTrades, -trades.trd_bal);
Highest_Bal_Limit := EVALUATE(HighestBals[1], trades.trd_hc);
OUTPUT(person(Highest_Bal_Limit > 25000));
```

Lab Exercises Next Lesson

- ✓ Verify Helper Functions to your Repository
- ✓ Do Query Exercises:
  - ✓ 1 through 9 (Lessons 9 through 17)

Lesson Completed!

Proceed to Lesson 9

Query Exercise 1: ***PropBath_3***

Advanced ECL (Part 1) – Working With Relational Data

Lesson 9

Query Exercise 1 – PropBath_3

Query Exercise 1:

Exercise Spec:
Calculate the number of Property records with 3 or more bathrooms, ever. This will be used to calculate number of the specified properties for each person.
For the purpose of this definition, the number of "bathrooms" is defined as: "The number of full baths, **plus** the number of half baths **divided** by two, and **rounded** to the nearest integer". You will have to query both Property and its related TaxData records.

Requirements:

1. Create an EXPORT Value Definition file called **PropBath_3 .**
2. Define a local FUNCTION structure called **IsValidBathCount(UNSIGNED2 TotalBaths).**
3. Within that FUNCTION structure, define a Boolean attribute called **IsValidProperty** that will be TRUE if there are the same or more bathrooms than the value passed as the **TotalBaths parameter.** Use the criteria above to create the expression.
4. Within that FUNCTION structure, define a Boolean attribute called **IsValidTaxdata that will be TRUE if any** *Taxdata* record exists with the same or more bathrooms than the value passed as the **TotalBaths parameter.**
5. Create the RETURN expression for the FUNCTION structure:
   (1) Is the sum of the Property record's full and half baths divided by two >= to the **TotalBaths** parameter**,** OR
   (2) does a *Taxdata* record exist whose full and half baths divided by two are >= to the **TotalBaths** parameter***.***
6. Create the EXPORT Attribute definition for **PropBath_3.** Use the COUNT function, the Property dataset, and your **IsValidBathCount** function, passing it a **TotalBaths** parameter of 3.

Best Practices Hint
The solution to the problem spec for this exercise is coded as a generic FUNCTION instead of a straight solution to the single problem of finding all the properties whose number of baths is greater than or equal to 3. This approach allows us to "promote" this generic FUNCTION to SHARED or EXPORT if another similar problem comes along whose only difference is in the value to use (such as "2" or "4" or ...).

Result Comparison
Open a new Builder window and run a query on the denormalized People file like this:

```
IMPORT TrainingYourName;
COUNT(TrainingYourName.File_PeopleAll.People(TrainingYourName.PropBath_3 > 0));
```

The expected result from this query is **21561.**

Query Exercise 1 Solution:

```
IMPORT $;
Property := $.File_PeopleAll.Property;
TaxData  := $.File_PeopleAll.Taxdata;
IsValidBathCount(UNSIGNED2 TotalBaths) := FUNCTION
  IsValidProperty := Property.Full_baths +
                     ROUND(Property.Half_baths/2) >= TotalBaths;
  IsValidTaxdata  := EXISTS(Taxdata(Full_baths +
                                 (ROUND(Half_baths/2)) >= TotalBaths));
  RETURN IsValidProperty OR IsValidTaxdata;
END;
EXPORT PropBath_3 := COUNT(Property(IsValidBathCount(3)));
```

Lesson Completed!

Proceed to Lesson 10:

Query 2 - PropVehSumEx

HPCC SYSTEMS®

Advanced ECL (Part 1) – Working With Relational Data

Lesson 10

Querying Relational Datasets: Query Exercise 2 - PropVehSumEx

Query Exercise 2

Exercise Spec:

Calculate the sum of **Property** and **Vehicle** records for each person that meet the below-specified criteria, limiting the final result to a maximum of 5. To be included, the Vehicle must be a Passenger car made in the last three years whose price is > 15,000. To be included, the Property must be: a "non-apartment" built in the last ten years whose total value is > 150,000.

Requirements:

1. Create an EXPORT Value definition file called **PropVehSumEx.**
2. Create a local definition called **CarCount** to count vehicles meeting the criteria**.** Refer to the Vehicle Dataset definition to determine the appropriate fields to use for the price and make year.
3. Create a local set definition named **SetPassengerCarCodes** containing strings of the concatenated states and codes that indicate a vehicle is a passenger car. Use the table below to create the Set definition as a set of strings:

State	Code	State	Code	State	Code	State	Code	State	Code	State	Code
AL	PASS	FL	AU	MN	09	MT	PC	OK	PASS	WI	AUTO
CT	PASS	ID	CAR	MO	P	NE	P	OR	PASS		
DE	PASS	MD	PASS	MS	1	OH	PC	SC	PASS		

4. Create a local value definition named **CatCodes** that will concatenate the two fields that determine a passenger car. Check the Vehicle dataset documentation in the Data Dictionary to determine the names of the original state and vehicle type fields. Use the COUNT function to count the vehicles meeting the specified criteria. Use the **IsValidAmount, IsValidYear, and YearsOld** functions that you downloaded in an earlier lesson.
5. Create a local definition called **PropCount** to count properties meeting the criteria**.** Refer to the Property Dataset definitions to determine the appropriate fields to use for the total value and year built. Use the COUNT function to count the properties meeting the specified criteria. Use the **IsValidAmount, IsValidYear, and YearsOld** functions that you downloaded in an earlier lesson.

Result Comparison

Open a new Builder or Builder Window Runnable window and run a query on the People file like this:

```
IMPORT TrainingYourName;
COUNT(TrainingYourName.DN.People(TrainingYourName.PropVehSumEx > 0));
```

The expected result from this query is **1333.**

Query Exercise 2 Solution:

```
IMPORT $;
Vehicle  := $.File_PeopleAll.Vehicle;
Property := $.File_PeopleAll.Property;
SetPassengerCarCodes := ['ALPASS','CTPASS','DEPASS','FLAU','IDCAR',
                         'MDPASS','MN09','MOP','MS1','MTPC','NEP',
                         'OHPC','OKPASS','ORPASS','SCPASS' ,'WIAUTO'];
CatCodes := Vehicle.Orig_state + Vehicle.Vehicle_type;
CarCount := COUNT(Vehicle(CatCodes IN SetPassengerCarCodes,
                          $.IsValidAmount(Vina_price),
                          Vina_price > 15000,
                          $.IsValidYear(Year_make),
                          $.YearsOld(Year_make)<=3));
PropCount := COUNT(Property(Apt = '',
                            $.IsValidAmount(Total_value),
                            Total_value > 150000,
                            $.IsValidYear(Year_built),
                            $.YearsOld(Year_built)<=10));
EXPORT PropVehSumEx := $.Limit_Value(CarCount + PropCount, 5);
```

Lesson Completed!

Proceed to Lesson 11:

Query 3
PropTaxBeds3

HPCC SYSTEMS®

Advanced ECL (Part 1) – Working With Relational Data

Lesson 11

Querying Relational Datasets – Query 3: PropTaxBeds3

Query Exercise 3:

Exercise Spec:
Calculate the total number of properties that have, or have *ever* had, 3 or more bedrooms.

Requirements:
1. Create an EXPORT Value definition file called **PropTaxBeds3.**
2. Use the appropriate fields in the Property and TaxData datasets (remember, TaxData is a child file of Property).

Best Practices Hint
Creating simple function will help to eliminate redundant code.
For example, *value 1 is greater than or equal to value 2* would be a good candidate.

Result Comparison
Open a new Builder window and run a query on the People file like this:

```
IMPORT $;
People := $.File_PeopleAll.People;
OUTPUT(COUNT(People($.PropTaxBeds3 > 0)),NAMED('FourC_26549'));
```

The expected result from this query is **26549.**

Query Exercise 3 Solution:

```
IMPORT $;
Property := $.File_PeopleAll.Property;
Taxdata  := $.File_PeopleAll.Taxdata;

IsGTE(INTEGER A1, INTEGER A2) := A1 >= A2;

EXPORT PropTaxBeds3 := COUNT(Property(IsGTE(Bedrooms,3) OR
                                      EXISTS(Taxdata(IsGTE(Bedrooms,3)))
                                     )
                             );
```

Lesson Completed!

Proceed to Lesson 12:

Query Exercise 4 PropValSmallStreet

Advanced ECL (Part 1) – Working With Relational Data

Lesson 12

Querying Relational Datasets – Query 4: PropValSmallStreet

Query Exercise 4:

Exercise Spec:
Calculate the aggregate Property values for all properties on “small” streets (CT, LN, WAY, CIR, PL, or TRL).

Requirements:

1. Create an EXPORT Value definition file called **PropValSmallStreet.**
2. Create a local Boolean definition that will be TRUE for any of the following types of small street types:
   CT, LN, WAY, CIR, PL, TRL
3. Use the total value and assessed value fields from the Property dataset.
4. If both are valid, use the larger of the two valid values
5. If there are no properties return -9
6. If there are no “small” street properties return -9
7. If there are no “small” street properties w/valid values return -9

Best Practices Hint
Create a local definition to return the larger of two valid values. All the conditions that require a -9 return value are met by simply doing all the appropriate filtering and determining if there are any records that meet the specified condition.

Result Comparison
Open a new Builder window or Builder Window Runnable file and run a query on the People file like this:

```
IMPORT $;
People := $.File_PeopleAll.People;
OUTPUT(COUNT(People($.PropValSmallStreet > 0)),NAMED('Four_20581'));
```

The expected result from this query is **20581.**

Query Exercise 4 Solution:

```
IMPORT $;
Property := $.File_PeopleAll.Property;
Total    := $.File_PeopleAll.Property.Total_value;
Assessed := $.File_PeopleAll.Property.Assessed_value;

IsSmallStreet := Property.StreetType IN ['CT','LN','WAY','CIR','PL','TRL'];
HighValue := IF($.IsValidAmount(Total) AND $.IsValidAmount(Assessed),
                 IF(Total > Assessed,Total,Assessed),
                 IF($.IsValidAmount(Total),Total,Assessed));

SmallProperties := Property(IsSmallStreet,$.IsValidAmount(HighValue));
EXPORT PropValSmallStreet := IF(NOT EXISTS(SmallProperties),
                                -9,
                                SUM(SmallProperties,HighValue));
```

Lesson Completed!

Proceed to Lesson 13:

Query Exercise 5 PropTaxBedYearRange

Advanced ECL (Part 1) – Working With Relational Data

Lesson 13

Querying Relational Datasets – Query 5: PropTaxBedYearRange

Query Exercise 5:

Exercise Spec:

Calculate the number of Properties with exactly 3 bedrooms in a year acquired within 5-15 years ago, or exactly 3 bedrooms in a tax year within 5-15 years ago.

Requirements:

1. Create an EXPORT value definition named **PropTaxBedYearRange.**
2. Create local definitions to calculate and determine:
    a. Is the target year values within the specification range?
    b. The filter required for the Property dataset.
    c. The filter required for the Taxdata dataset.

Best Practices Hint

Don't forget to check the related *Taxdata* records, too, using the tax year and the EXISTS function.

Result Comparison

Open a new Builder or Builder Runnable window and run a query on the People file like this:

```
IMPORT $;
People := $.File_PeopleAll.People;
OUTPUT(COUNT(People($.PropTaxBedYearRange > 0)),NAMED('Five_23880'));
```

The expected result from this query is **23880.**

Query Exercise 5 Solution

```
IMPORT $;
Property := $.File_PeopleAll.Property;
Taxdata  := $.File_PeopleAll.Taxdata;
IsDateInRange(datex,lo,hi) := $.IsValidYear(datex) AND
                              $.YearsOld(datex) BETWEEN lo AND hi;


Prop_filter := Property.Bedrooms = 3 AND
               IsDateInRange(Property.Year_Acquired,5,15);

Tax_filter := Taxdata.Bedrooms = 3 AND
              IsDateInRange(Taxdata.Tax_year,5,15);
Tax_exists := EXISTS(Taxdata(Tax_filter));
EXPORT PropTaxBedYearRange := COUNT(Property(Prop_filter OR Tax_exists));
```

Lesson Completed!

Proceed to Lesson 14:

Query Exercise 6
VehicleNewTruckPrice

HPCC SYSTEMS®

Advanced ECL (Part 1) – Working With Relational Data

Lesson 14

Querying Relational Datasets – Query 6: VehicleNewTruckPrice

HPCC SYSTEMS®

LexisNexis® RISK SOLUTIONS

RELX Group

Query Exercise 6:

Exercise Spec:

Calculate the price of the newest truck and output it as a 6 character string.

A "truck" is defined as any vehicle type code using: **4, T, TK, TR, TRK, or TRUK.**

Requirements:

1. Create an EXPORT Value Definition called **VehicleNewTruckPrice.**
2. If there is more than one truck, use the one with the most recent model year.
3. If the model year is invalid, use the "best model year" and if both are invalid, exclude the vehicle.
4. If the price value is invalid, exclude the vehicle.
5. If there are multiple vehicles with the same model year, use the one with the highest price. If no vehicles meet the criteria, output blanks.

Best Practices Hint

This exercise uses complex filtering conditions in a real-world manner. It also introduces use of the SORT function to “bubble up” to first position in the sorted set a specific child record from which we can extract the single value we're interested in. Breaking out which **BestYear** field to use into a separate attribute and filtering for valid values of that allows you to also SORT by it for the most recent date. Doing a descending SORT within the year puts the largest **Vina_price** value in first position, in case there are several with the same date.

Result Comparison

Open a new Builder window and run a query on the People file like this:

```
IMPORT $;
People := $.File_PeopleAll.People;
OUTPUT(COUNT(People($.VehicleNewTruckPrice <> '')),NAMED(‘Six_6462'));
```

The expected result from this query is **6462.**

Query Exercise 6 Solution

```
IMPORT $;
Vehicle   := $.File_PeopleAll.Vehicle;
IsTruck   := Vehicle.Vehicle_type IN ['4','T','TK','TR','TRK','TRUK'];
BestYear := IF($.IsValidyear(Vehicle.model_year),
                             Vehicle.Model_year,
                             Vehicle.Best_model_year);
Trucks    := Vehicle(IsTruck,
                      $.IsValidyear(BestYear),
                      $.IsValidAmount(Vina_price));
SortedTrucks := SORT(Trucks,-BestYear,-Vina_price);
EXPORT STRING6 VehicleNewTruckPrice := IF(NOT EXISTS(SortedTrucks),
                                             '',
                                             (STRING6) SortedTrucks[1].Vina_price);
```

Lesson Completed!

Proceed to Lesson 15:

Query Exercise 7
PropTaxDataHomeAssess

Advanced ECL (Part 1) – Working With Relational Data

Lesson 15

Querying Relational Datasets – Query 7: PropTaxDataHomeAssess

EVALUATE Function:

EVALUATE(*onerecord, value*)

- ✓*onerecord* – A recordset consisting of a single record.
- ✓*value* – The value to return. This may be any expression yielding a single value.

```
ValidBalTrades      := trades(ValidMoney(trades.trd_bal));
HighestBals         := SORT(ValidBalTrades, -trades.trd_bal);
Highest_Bal_Limit := EVALUATE(HighestBals[1], trades.trd_hc);
OUTPUT(person(Highest_Bal_Limit > 25000));
```

EVALUATE Training Example:

- ✓Training_Examples.EVALUATE_Example
- ✓View/download from ECL Playground

Query Exercise 7:

Exercise Spec:
Calculate the assessed total value from the most recently reported *Taxdata* record for the most recently acquired *Property* that is not an apartment. If there are multiple properties for the same year, use the one with the highest property value. Output the result as an 8 character string.

Requirements:
1. Create an EXPORT Value Definition called **PropTaxdataHomeAssess.**
2. If the Property's year acquired is invalid, use the year built. If the year built is also invalid, exclude the property.
3. If the Property's total value is invalid, use the assessed value.
4. If the assessed value is also invalid, exclude the property. If the Taxdata tax year or Assessed total value is invalid, exclude the taxdata record.
5. Default to blank if no valid properties exist or no valid Taxdata records exist for the selected property.

Best Practices Hint
Use the SORT and EVALUATE functions to implement the definition.

Result Comparison
Open a new Builder window and run a query on the People file like this:

```
IMPORT $;
People := $.File_PeopleAll.People;
OUTPUT(COUNT(People($.PropTaxDataHomeAssess <> '')),NAMED('Seven_25722'));
```

The expected result from this query is **25722.**

Query Exercise 7 Solution:

```
IMPORT $;
Property := $.File_PeopleAll.Property;
Taxdata  := $.File_PeopleAll.Taxdata;
PropYear  := IF($.IsValidYear(Property.Year_Acquired),
                             Property.Year_Acquired,
                             Property.Year_built);
PropValue := IF($.IsValidAmount(Property.Total_value),
                               Property.Total_value,
                               Property.Assessed_value);
ValidProperty := Property($.IsValidYear(PropYear),
                         $.IsValidAmount(PropValue),
                         Apt='');
SortedProperty := SORT(ValidProperty,-PropYear,-PropValue);
ValidTaxdata   := Taxdata($.IsValidYear(Tax_year),
                         $.IsValidAmount(Assd_total_val));


SortedTaxdata  := SORT(ValidTaxdata,-Tax_year);
EXPORT STRING8 PropTaxDataHomeAssess :=
                      IF(NOT EXISTS(SortedProperty),
                         '',
                         EVALUATE(SortedProperty[1],
                                  IF(NOT EXISTS(SortedTaxdata),
                                     '',
                                     (STRING8)SortedTaxdata[1].Assd_total_val)));
```

HPCC SYSTEMS®

Lesson Completed!

Proceed to Lesson 16:

Query Exercise 8
CountUniqueMakeVehicles

Advanced ECL (Part 1) – Working With Relational Data

Lesson 16

Querying Relational Datasets – Query 8: CountUniqueMakeVehicles

Query Exercise 8:

Exercise Spec:
Calculate the number of unique vehicles owned, based on their make code. If no vehicles exist, output a -9 value.

Requirements:
1. Create an EXPORT value definition called **CountUniqueMakeVehicles.**
2. Use the DEDUP function to implement the attribute.

Result Comparison
Open a new Builder window and run a query on the People file like this:

```
IMPORT $;
People := $.File_PeopleAll.People;
OUTPUT(COUNT(People($.CountUniqueMakeVehicles > 0)),NAMED('Eight_27900'));
```

The expected result from this query is **27900**.

Query Exercise 8 Solution

```
IMPORT $;

SortedVehicles := SORT($.File_PeopleAll.Vehicle,Make_code);
DedupVehicles  := DEDUP(SortedVehicles,Make_code);

EXPORT CountUniqueMakeVehicles := IF(~EXISTS(DedupVehicles),
                                     -9,
                                     COUNT(DedupVehicles));
```

Lesson Completed!

Proceed to Lesson 17:

Query Exercise 9 FordChevyWithin90

Advanced ECL (Part 1) – Working With Relational Data

Lesson 17

Querying Relational Datasets – Query 9: FordChevyWithin90

Query Exercise 9:

Exercise Spec:

Calculate the number of Ford Vehicles purchased within 90 days of purchasing a Chevrolet Vehicle. If no Vehicles exist , output a –9 value.

Best Practices Hint

1. Create an EXPORT value definition called **FordChevWithin90**.
2. The make description data values must be: *Chevrolet* and *Ford*.
3. Use the appropriate Date functions in the Standard Library Reference to create a local FUNCTION that calculates the days between two dates.

   STD.Date.FromStringToDate

   STD.Date.DaysBetween
4. In addition, use the **ABS** function to allow flexible date parameters.
5. Use the **DEDUP** function with the **ALL** option to implement the definition, or a self JOIN as an alternate approach.

Result Comparison

Open a new Builder window and run the following query on the People file:

```
IMPORT $;
People := $.File_PeopleAll.People;
OUTPUT(COUNT(People($.FordChevWithin90 > 0)),NAMED('Nine3_572'));
```

After running this query, you should verify that there are **572** people who had bought a Chevy and a Ford within 90 days.

Query Exercise 9 Solution:

```
IMPORT $,STD;
IsWithinDays(STRING8 ldate,STRING8 rdate,INTEGER days)
   := ABS(STD.Date.DaysBetween(STD.Date.FromStringToDate(ldate, '%Y%m%d'),
                               STD.Date.FromStringToDate(rdate, '%Y%m%d'))) <= days;


Chev := 'Chevrolet';
Ford := 'Ford';
CFVehicles  := $.File_PeopleAll.Vehicle(make_description IN [Chev,Ford]);
DedupedCars := DEDUP(CFVehicles,LEFT.Make_description = Chev AND
                                RIGHT.Make_description = Ford AND
                                IsWithinDays(LEFT.Purch_date,RIGHT.Purch_date,90),
                     ALL);
EXPORT FordChevWithin90Upd := IF(~EXISTS(CFVehicles),
                              -9,
                              COUNT(CFVehicles) - COUNT(DedupedCars));
```

HPCC SYSTEMS®

Lesson Completed!

Proceed to Lesson 18: Normalizing Your Data!

HPCC SYSTEMS®

Advanced ECL (Part 1) – Working With Relational Data

Lesson 18

Normalizing Your Data

Normalizing Your Data

- ✓ NORMALIZE Function
- ✓ Using COUNTs or CHILDREN (forms of NORMALIZE)

NORMALIZE Function

NORMALIZE(*recordset, expression, transform***)**

- ✓ *recordset* – The set of records to process.
- ✓ *expression* – An numeric expression specifying the total number of times to call the *transform* for that record.
- ✓ *transform* – The TRANSFORM function to call for each record in the *recordset*.

The *transform* function must take two parameters: A LEFT record of the same format as the *recordset*, and an integer COUNTER specifying the number of times to call the *transform* for that record. The format of the resulting recordset can be different from the input.

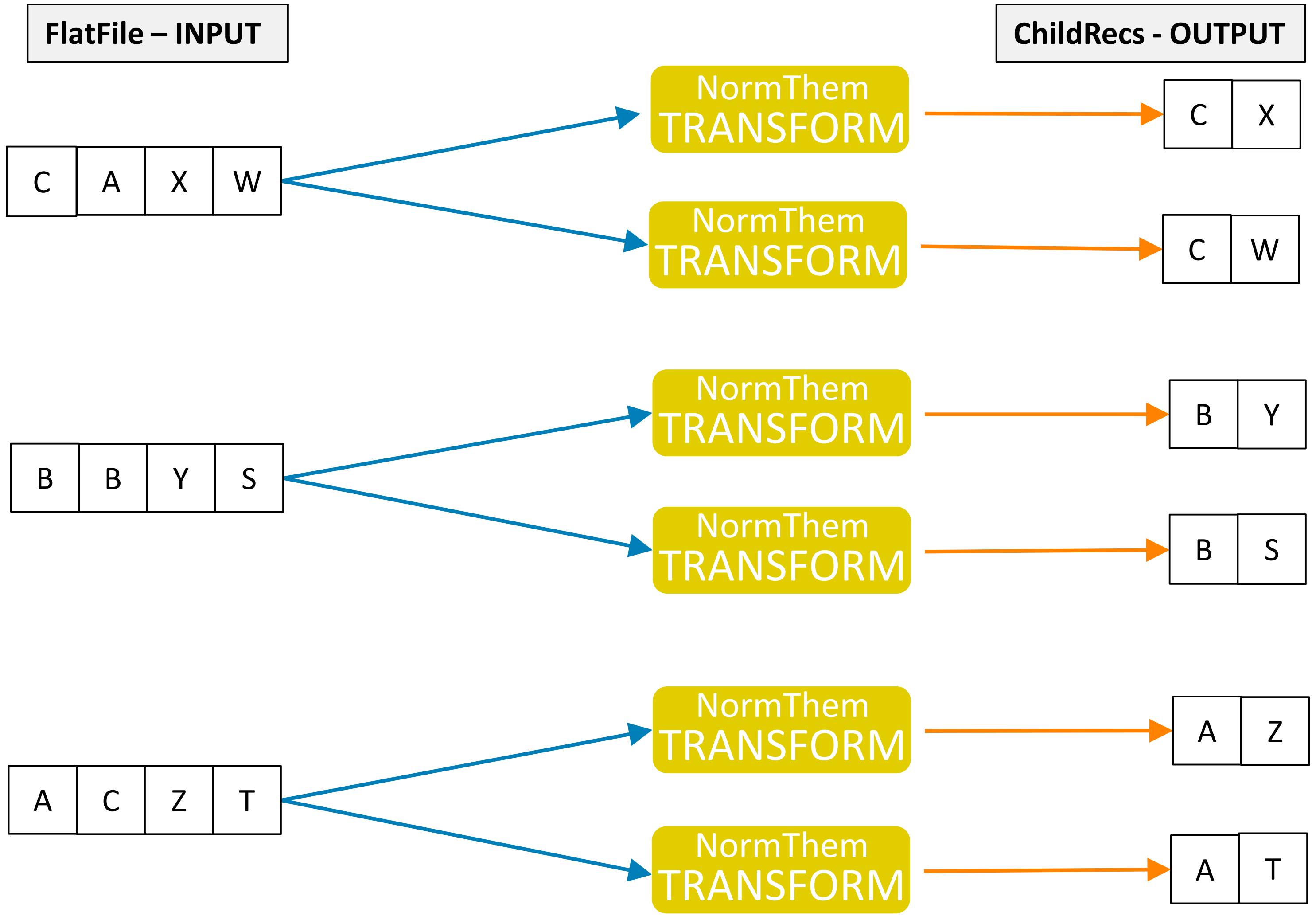
NORMALIZE Example:

```
Layout_BK_Slim := RECORD
        UNSIGNED6     id;
        UNSIGNED3     bankruptcy_date;
END;

Layout_BK_Slim NormBK(Bankrupt.Layout_BK L,INTEGER ctr) := TRANSFORM
        SELF.id := (UNSIGNED6)CHOOSE(ctr, L.Debtor_ID, L.Spouse_ID);
        SELF.bankruptcy_date := (UNSIGNED3)(L.Date_Filed[1..6]);
END;

BK_Init := NORMALIZE(Bankrupt.File_BK, 2, NormBK(LEFT, COUNTER));
```

NORMALIZE Training Example:

- ✓Training_Examples.NORMALIZE_Example

NORMALIZE Functional Diagram:



NORMALIZE Training Example:

- ✓Training_Examples. NORM_DENORM_ChildDatasets_Example

Lesson Completed!

Proceed to Lesson 19:

Lab Exercise 7
Normalizing Your Data

Advanced ECL (Part 1) – Working With Relational Data

Lesson 19

Lab Exercise 7 – Normalizing Your Data

Lab Exercise 7:

- ✓ Reference original RECORDs previously defined in Lessons 2 and 6.
- ✓ Restore Parent, and use any form of NORMALIZE on the child datasets
- ✓ Output four (4) normalized recordsets from the denormalized source you created in an earlier lesson.

Lab Exercise 7:

Exercise Spec:

Create Builder Window Runnable code using NORMALIZE that outputs four recordsets from the denormalized dataset that you created in earlier lessons in this course, including each original People record, and recordsets of each nested child; the Vehicle, Property, and Taxdata datasets. Essentially you are duplicating the original files that you sprayed at the start of this course.

Requirements:

1. The definition name to create for this exercise is:

   BWR_NormalizePeople
2. OUTPUT the four recordsets to the workunit, and verify that the results look good.

Result Comparison

Open a new Builder window and run the output and determine that the output of the normalization looks reasonable.

Lab Exercise 7 Solution:

```
IMPORT $;

//FIRST: Restore parent record (People)
ParentOut := PROJECT($.File_PeopleAll.People, $.File_People.Layout);
OUTPUT(ParentOut,NAMED('People'));

//Vehicle Next:
VOut := NORMALIZE($.File_PeopleAll.People,LEFT.VehicleRecs,TRANSFORM(RIGHT));
OUTPUT(VOut,NAMED('Vehicle'));

//Property
POut := NORMALIZE($.File_PeopleAll.People,LEFT.PropRecs,TRANSFORM($.File_Property.Layout,SELF := RIGHT));
OUTPUT(POut,NAMED('Property'));

//Finally, TaxData:
TOut := NORMALIZE($.File_PeopleAll.Property,LEFT.TaxRecs,TRANSFORM(RIGHT));
OUTPUT(TOut,NAMED('Taxdata'));
```

HPCC SYSTEMS®

Lesson Completed!

End of This Course
Congratulations!
Next Course:
Advanced ECL (Part 2)