Add an operator to NiagaraST HOWTO

### Rafael J. Fernández-Moctezuma ([rfernand@cs.pdx.edu](mailto:rfernand@cs.pdx.edu))

March 9 2008 (Rev. June 27 2008)

# Introduction

This document guides you, step by step, on the process of adding a new operator to the NiagaraST DSMS by duplicating an existing operator and changing the appropriate entries, as well as making a simple change to the operator. Exploring existing operators is probably the easiest way to determine which methods do what – as of the time of this writing, there is no unified documentation, nor are there plans to produce one.

# About Operators in NiagaraST

Operators in NiagaraST are declared *logically* and *physically*. This declaration is not redundant: consider the operator JOIN. Regardless of the physical implementation (nested loops or hash join) it has the same logical properties. This separation permits different physical implementations to exists in the system – the optimizer can make a cost-based choice when more than one physical implementation exists.

# From “select” to “selecto”

We will add a new operator called *selecto* to the operator catalog in NiagaraST. The *select* operator is defined by two classes: *niagara.logical.Select* and *niagara.physical.PhysicalSelect*. The files containing the classes are located under the /*src* directory. Duplicate these files to create the classes *niagara.physical.PhysicalSelecto* and *Niagara.logical.Selecto*. Make the appropriate changes in the class files (e.g., change the class names, include *niagara.logical.Selecto* instead of *niagara.logical.Select* in the physical operator, etc.)

Next, we will add catalog entries for *selecto*. Locate the catalog file (*/demo/niagara\_server/catalog.xml*) and duplicate the *<logical>…</logical>* entry of select under *<operators>*. Duplicate the “*select to PhysicalSelect*” rule in *<ruleset>* and rename appropriate. Last, open the DTD (*/demo/niagara\_server/queryplan.dtd*) and add the *selecto* operator name to the plan element, and duplicate the select element in the DTD. Sample entries are found in Figure 1 and Figure 2.

[…]

<operators>

[…]

<logical class="niagara.logical.Select" name="select">

<physical class="niagara.physical.PhysicalSelect" name="PhysicalSelect"/>

</logical>

[…]

</operators>

[…]

<ruleset>

[…]

<rule name="selecto to PhysicalSelecto" type="simple">

<before><logical name="selecto"/></before>

<after><op name="PhysicalSelecto"/></after>

</rule>

[…]

</ruleset>

[…]

**Figure 1.** changes in catalog.xml to declare the *selecto* operator.

[…]

<!ELEMENT plan (unnest | sort | expression | avg |

slidingAvg | sum | slidingSum | max |

slidingMax | count | slidingCount | incrmax |

incravg | select | **selecto** | dup | union | join |

dtdscan | resource | constant | timer |

prefix | punctuate | send | display | topn |

firehosescan | filescan | construct |

bucket | partitionavg | partitionmax |

accumulate | nest | magic\_construct |

windowCount | windowMax | windowAverage | windowJoin |

showTunables | collectInstrumentation |

store | load | xmlscan | dbscan | punctqc | present)+>

[…]

<!ELEMENT selecto (and | or | not | true | false | pred)>

<!ATTLIST selecto

%StandardAttributes;>

[…]

**Figure 2.** Changes in queryplan.dtd to declare the *selecto* operator.

If you followed the steps, next time you start the NiagaraST server you will see that you have 2 more registered operators (logical AND physical). Test your new *selecto* operator by writing a query plan. You should observe the same behavior as if you were using *select.*

# Simple change to selecto

We will add a flag called *special* to *selecto* to enable or disable intermittent production of tuples that match the selection predicate. First, modify the *selecto* entry in *queryplan.dtd* to add the attribute. We will only accept “yes” or “no” as values for this attribute, and will default to “no”. The result is shown in Figure 3. Notice that selecto also has standard operator attributes as part of its specification.

[…]

<!ELEMENT selecto (and | or | not | true | false | pred)>

<!ATTLIST selecto

**special (yes | no) "no"**

%StandardAttributes;>

[…]

**Figure 3.** Change in queryplan.dtd to add a flag to *selecto*.

Next, we make the appropriate changes to recognize the new flag and use it as part of *selecto*’s operation. First, we address the logical structure. We add a private string variable to hold the value to be read from the query plan, such as **private String select**. The constructor methods must now receive a string and update the value of **this.special**. The opCopy() method must also duplicate the logical class’ attributes. Add a **getSpecial()** method. Last, the **loadFromXML** function must retrieve the value of *special* from the query plan. These changes are illustrated in Figure 4.

[..]

**public** **class** Selecto **extends** UnaryOperator {

[…]

**private String special;**

**public** Selecto(Predicate pred, **String special**){

**this**.pred = pred;

**this.special = special;**

}

**public** Selecto(Selecto selOp, **String special**) {

**this**(selOp.pred,special);

}

**public** Op opCopy() {

Selecto op = **new** Selecto();

op.pred = **this**.pred;

**op.special = this.special;**

**return** op;

}

[…]

**public** **void** loadFromXML(Element e, LogicalProperty[] inputProperties, Catalog catalog)

**throws** InvalidPlanException {

**special = e.getAttribute("special");**

NodeList children = e.getChildNodes();

Element predElt = **null**;

**for** (**int** i = 0; i < children.getLength(); i++) {

**if** (children.item(i) **instanceof** Element) {

predElt = (Element) children.item(i);

**break**;

}

}

pred = Predicate.*loadFromXML*(predElt, inputProperties);

}

[…]

**public String getSpecial() {**

**return special;**

**}**

[…]

**Figure 4.** Changes to logical *selecto* to enable a new attribute.

Physical operators retrieve necessary values using the **opInitFrom(LogicalOp logicalOperator)** method. In this case, we add a string to the physical operator to hold the *special* flag value, and an internal flag to control the intermittent output of *selecto*. The *special* string in the physical operator will receive the value using the **getSpecial()** method from the logical operator, much like other attributes. Also update **opCopy**.The method **processTuple(Tuple inputTuple, int streamed)** is where the selection occurs, and the “main” method of this operator. Make the appropriate behavioural changes here. An example of the changes is shown in Figure 5.

[…]

**private** **static** **final** **boolean**[] *blockingSourceStreams* = { **false** };

// The is the predicate to apply to the tuples

**private** Predicate pred;

**private** PredicateImpl predEval;

**private String special;**

**private int flag = 0;**

**public** PhysicalSelecto() {

setBlockingSourceStreams(*blockingSourceStreams*);

}

**public** **void** opInitFrom(LogicalOp logicalOperator) {

pred = ((Selecto)logicalOperator).getPredicate();

predEval = pred.getImplementation();

**special = ((Selecto)logicalOperator).getSpecial();**

}

**public** Op opCopy() {

PhysicalSelecto p = **new** PhysicalSelecto();

p.pred = pred;

p.predEval = predEval;

**p.special = special;**

**return** p;

}

[…]

**protected** **void** processTuple(Tuple inputTuple, **int** streamId)

**throws** ShutdownException, InterruptedException {

// Evaluate the predicate on the desired attribute of the tuple

**if** (predEval.evaluate(inputTuple, **null**)) {

**if (special.equals("yes")) {**

**if (flag == 1) {**

**putTuple(inputTuple, 0);**

**flag = 0;**

**} else if (flag == 0) {**

**flag = 1;**

**}**

**} else {**

**putTuple(inputTuple, 0);**

**}**

**}**

}

[…]

**Figure 5.** Modified behavior of *selecto*.

That’s it! You have created an intermittent selector, whose intermittent behavior is controlled by a parameter specification in the query plan!