1Integrate

Performance Tips

Product version: v 1.3

Document version: v 1.0.1

Document date: 21/10/2016

1Spatial

Tennyson House

Cambridge Business Park

Cambridge

CB4 0WZ

United Kingdom

Phone: +44 (0)1223 420414

Fax: +44 (0)1223 420044

Web: www.1spatial.com

Contents

1 Performance Tips

This section provides advice on how to write rules and actions and set up data stores and sessions in order to get the best performance when running sessions in 1Integrate.

Use Indexes

An index speeds up the identification of data within a rule or action.

Specify indexes in the input mapping

In the datastore input mapping page, every attribute within a class has a tick box to allow you to specify whether the attribute should be indexed or not. The primary geometry is always indexed.

For Oracle data any other columns with Oracle indexes on them (including primary keys) will be set to be indexed by default.

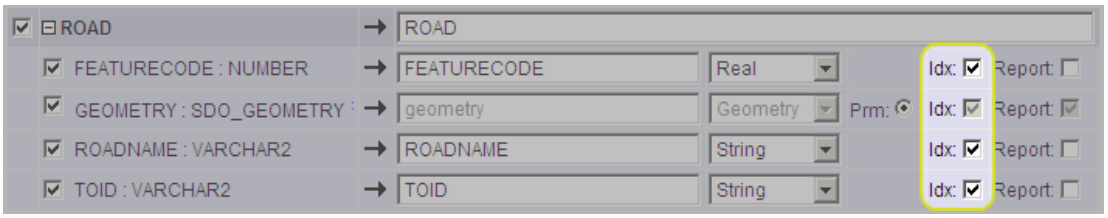

Figure 1-1: *Defining indexes in the Input Mapping of a datastore*

If you need to identify features in a rule or action using an attribute as the initial selection criteria, then you need to tick the box to index that attribute.

Identifying features means anywhere that you need to specify a class of objects other than in the rule or action's root node (e.g. using a For All, Existence, Aggregate, Chaining or Loop Over Objects construct within a rule or action).

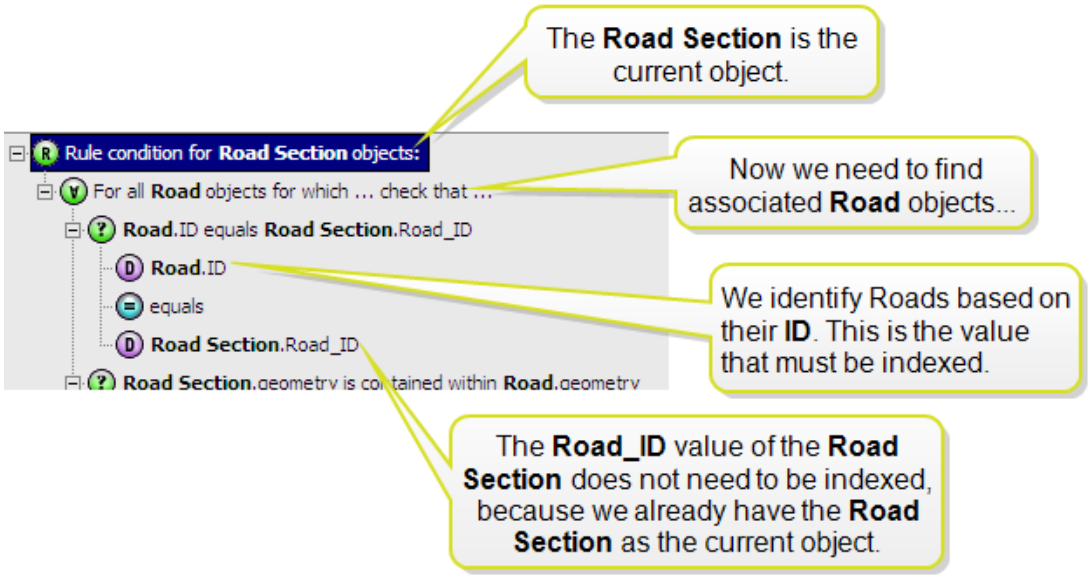

Figure 1-2: *A rule requiring an indexed value*

If you are doing a spatial search (i.e. a comparison on the geometry) to identify the nearby features before subsequently checking the ID, then the index on the id is not required. You will only need it when the attribute is used as the initial way to select the set of objects.

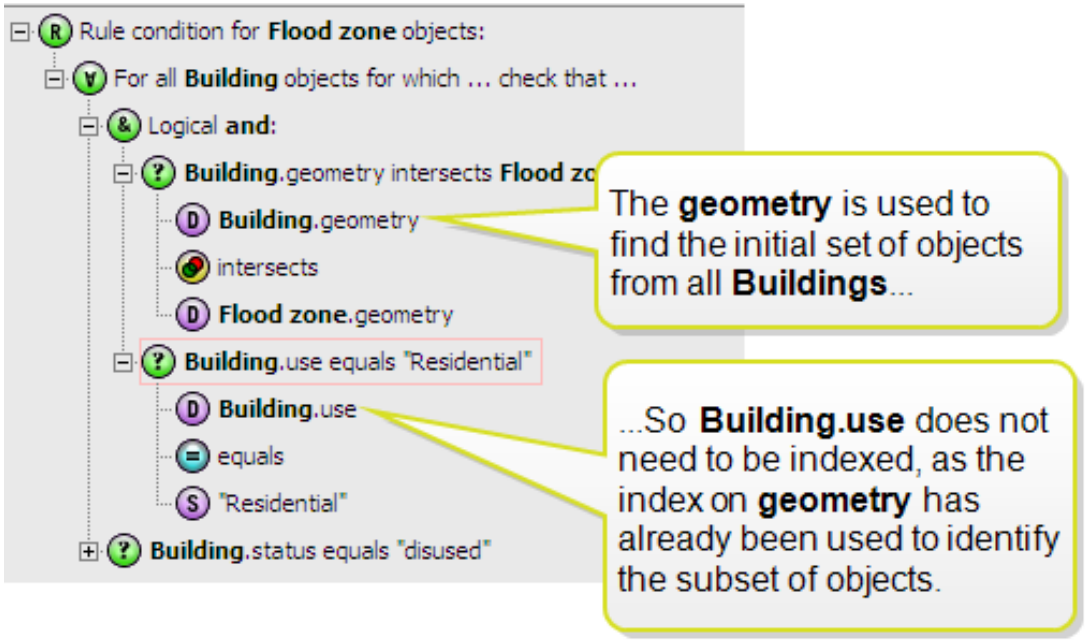

Figure 1-3: *Index not required on Building because a small set of objects already identified using another value (in this case a spatial search on the geometry)*

While you could just index all attributes just in case, this will require more disk space for the cache while the session is running and will add an overhead to the time taken to open the data within the session.

Use the unmodified value when comparing indexed data

Whenever you use an indexed value (this includes the primary geometry) as the first comparison then don't apply any built-in functions to the value.

In order to use the index, the comparison needs to run on the raw attribute value itself because this is what has been indexed, not the result of a built-in function.

For example the action displayed below will be very slow because we can't use the spatial index to identify the admin areas. We are running the **outer_ring** built-in function on each geometry first which means that every object in the class will need be tested explicitly, rather than using the index on the geometry to reduce the set.

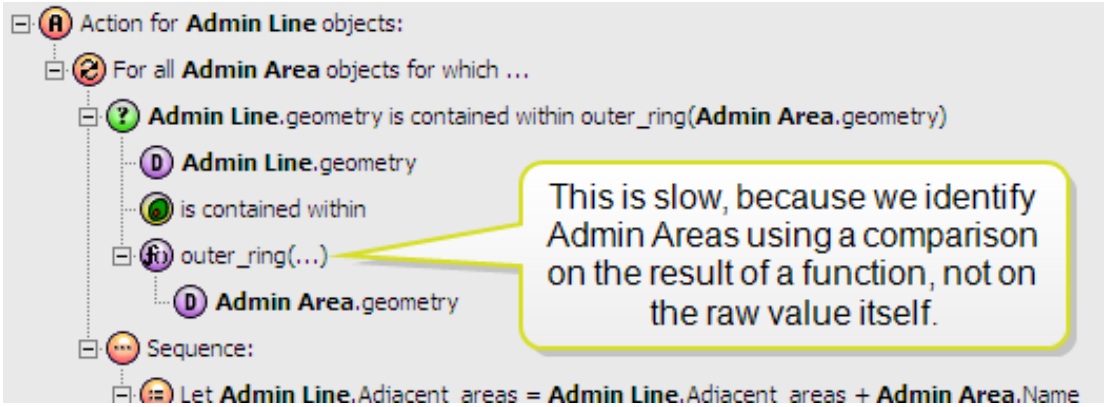

Figure 1-4: *Slow use of outer_ring function*

Instead, you may need to add a new initial condition that uses the unmodified value as the first condition, then AND it together with the more specific condition that uses the outer_ring function.

You need to ensure that the first condition finds at least the features that you want, with as few as possible additional features.

In the example below, the spatial intersects condition will find all areas that are spatially in contact with the line in some way. From this limited subset, we get the outer ring of each one and do the comparison.

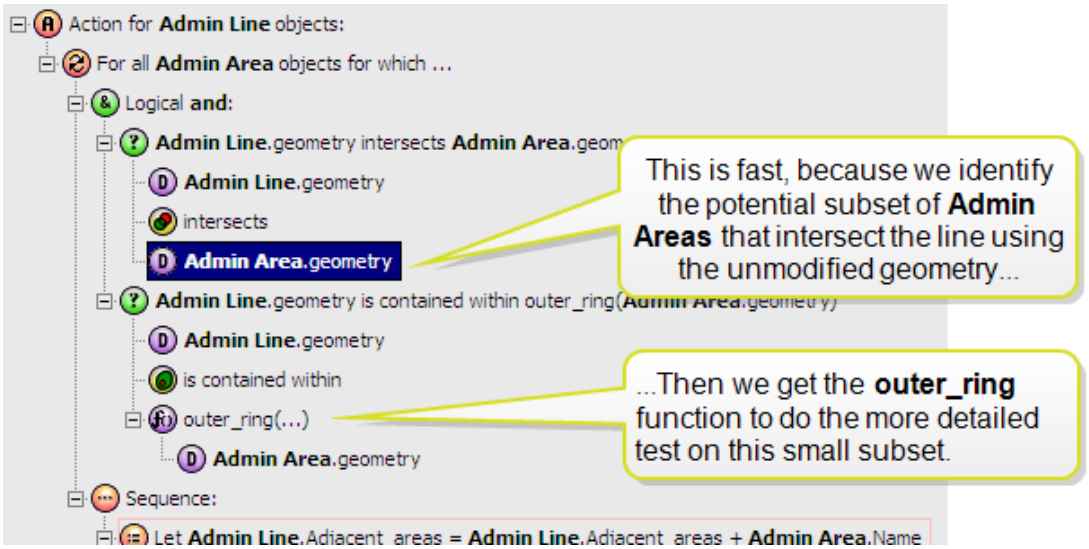

Figure 1-5: *Faster use of outer_ring function*

Order Conditions Sensibly

Conditions should be ordered sensibly, in order to reduce the selection set as quickly as possible.

If you need to identify features in a rule or action using more than one condition then you need to think about the order of these conditions. You will need to do this anywhere that you specify a class of objects other than in the rule or action's root node (i.e. a For All, Existence, Aggregate, Chaining or Loop Over Objects construct within a rule or action).

The conditions are applied in the order in which they appear, which means that you need to order them so that the selection set is narrowed down as quickly as possible. Knowledge of the data will help to decide how to achieve this. Often a selection set is reduced using a spatial *intersection* or *within distance* check but sometimes there is another attribute which can be used to reduce the selection set.

Ensuring that the first condition uses an indexed value is the most important consideration (see "Use Indexes" on page 3), but it is also useful to consider the ordering of subsequent checks, based on either their complexity or ability to reduce the selection set.

For example, In a rule which reduces the selection set spatially, then performs a more detailed geometric comparison plus a simple attribute comparison, then it will be quickest to perform the attribute comparison second and the detailed geometric comparison third. The attribute comparison will be very fast, so it is worth doing this before the final complex geometric check in order to be able to easily reduce the number of complex geometric comparisons.

In the example shown below, the action sets a road centreline to be non-drivable if it is at least 90% within footpath polygons. Here we look at the classification string value before doing a more complex spatial comparison.

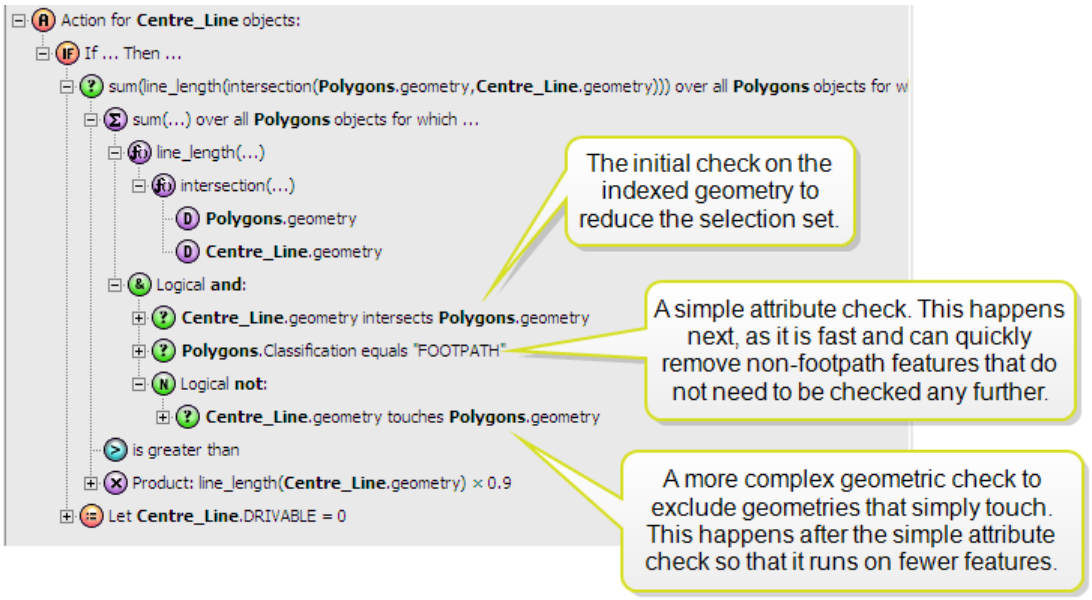

Figure 1-6: *Put simple checks before complex checks.*

Temporary Variables

When writing actions, you can reduce the amount of repeated processing within aggregate values or loops by storing computed values as temporary values.

A loop could be *existence*, *for all* or chaining conditions, as well as *while* loops or *loop over objects* operations.

If each iteration of the loop is performing some non-trivial work on the current object such as buffering, then it is quicker to perform the buffering once, assign the result to a temporary value and then use the temporary value within the loop.

The **For all** loop displayed in the action below will calculate the buffer of the current Centre-Line object for every iteration, even though the result will be identical each time.

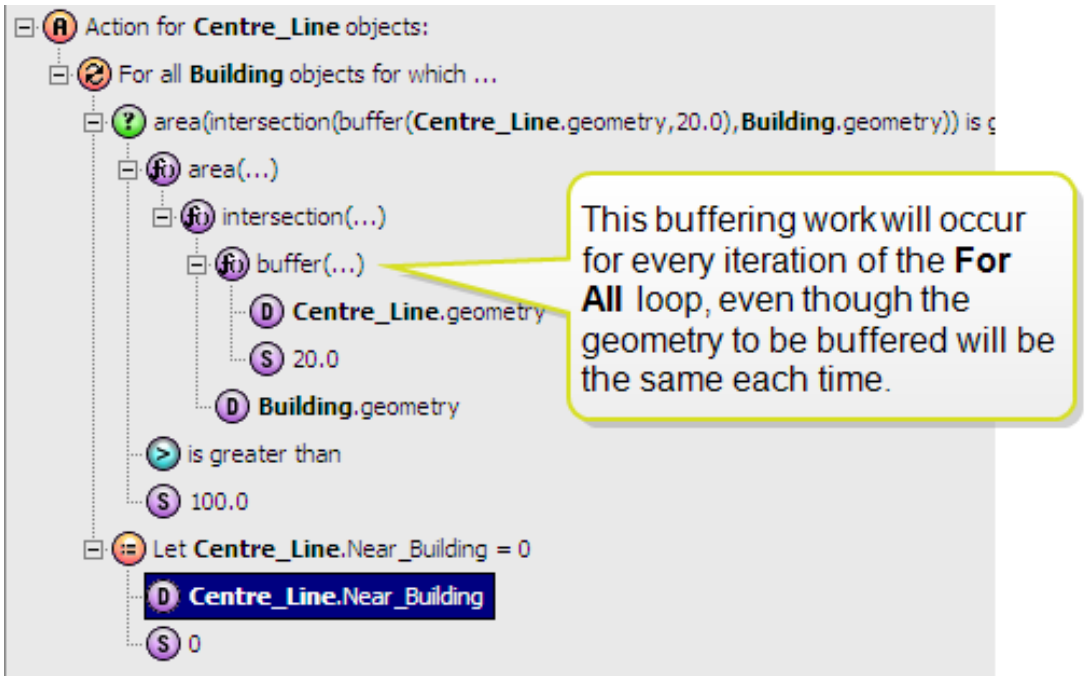

Figure 1-7: *Buffering occurs for every iteration of the loop*

The action below will be faster, as the buffer is calculated only once, stored as a temporary variable and then the temporary variable is used within the loop.

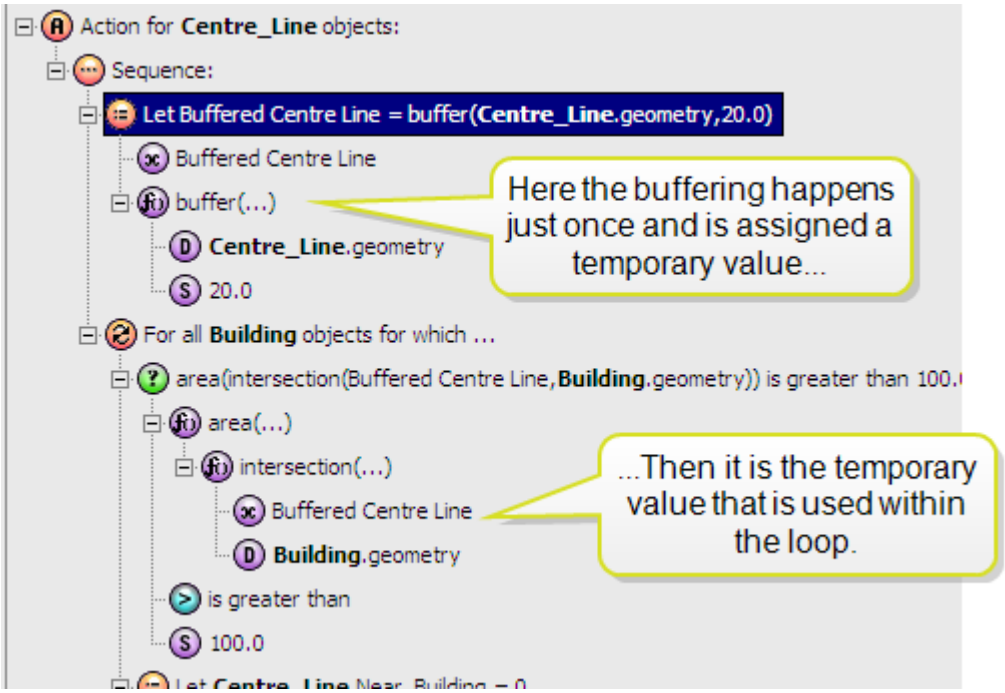

Figure 1-8: *Buffer is calculated only once*

Use Accurate Spatial Extents

The spatial extents of data read from Oracle are read (if they exist) from the Oracle spatial metadata table (USER_SDO_GEOM_METADATA).

The more accurate these extents are, the better the performance of spatial searches. It will still find and process data regardless of these extents (although if they are orders of magnitude wrong the Build Topology task might complain) but it will improve performance if they are roughly right. This will improve all conditions or operations which use the geometry as the first comparison, as well as the Build Topology task.

To see the current setting for the spatial extents, you can view the geometry attributes (see Viewing Geometry Attributes).

Share Tasks

When developing and testing rules and actions, you usually create a task for each rule or action within a session so that you can easily see the results for each task.

If you have multiple rules or actions that run on the same class name, then it is quicker to run these within the same task. This is because when an object is "grabbed" we can run all the relevant rules or actions while it is to hand. Otherwise, if the rules or actions are all in their own tasks, then all the objects in the relevant class are cycled through for each task.

Note: Be careful with actions; the order of how rules or actions are run within the same task is not guaranteed. For rules this does not matter as rules are stateless (i.e. they don't change the state of the data so the order doesn't matter) but for actions, you often rely on the order in which they are run. If this is the case then you will need multiple tasks to run your actions in order to guarantee the order in which they are run.

Efficient Input Mapping

When setting up Input Mapping for a data store, you can specify which attributes to include and which to exclude by selecting and deselecting attributes for each input class.

If you only need to access, update or "copy to" a subset of these attributes, then opening and committing a data store can be made faster by only enabling the attributes that you require.