let select

Last Modified on 25/05/2020 1:44 pm AEST

Assigns a sub-calculation to a variable that can then be used later in the calculation.

Keywords

```
let variable = calculation select result
```

Arguments

Argument	Data Type
variable	a label in the calculation
calculation	any
result	any

Comments

The **let** and **select** keywords can be used together to allow calculations to be simplified and organised into smaller sub-calculations.

The **let** keyword is used to associate some calculation to a with a variable name. That variable name can then be used within the *result* calculation that follows the **select** keyword.

```
let dueDate = [Assessment].[Due date]
select 'The assessment is due ' + dueDate
```

Multiple variables

The **let** keyword may be used multiple times to define multiple variables. A list of**let** keywords is concluded with a single **select** keyword. The calculation for each variable may make reference to other variables that were defined before it (but not after). In the following example, the reminderDate calculation may refer to the dueDate variable, and the overall result calculation can refer to both variables.

```
let dueDate = [Assessment].[Due date]
let reminderDate = dateadd(day, -7, dueDate)
select  'The assessment is due ' + dueDate + ' and a reminder will be sent ' + reminderDate
```

Variable scope

Typically variables are listed at the top of a calculation, but they may also be declared and used within sub-calculations. Doing so can some times be advantageous when organising a more complicated calculation that contains aggregation calculations. A variable name is *scoped* such that it may only used within the**select** calculation that it corresponds to, and within other **let** calculations that follow it that also belong to the same select keyword.

In general, the calculation: (let x = value select result), may be used with or without brackets in most places that any other calculation can be used. In the following example, variables are defined within the sum function to simplify the calculation of individual items.

```
sum(
  let x = [Order Items]
  let discount = iif(x.[Item quantity] >= 10, 20.0, 0)
  select x.[Item cost] * x.[Item quantity] - discount
)
```

Variable names

Variables names behave similar to field and relationship names. They:

- May be specified with or without square brackets.
- If there are no square brackets, then they may only consist of letters, numbers, and underscore (and further may not start with a number).
- If square brackets are used, then any name may be used so long as it does not itself consist of square brackets.

Variable names are case-insensitive. By convention, variable names are usually specified without square brackets, and tend to join multiple words together. The following are all legal variable names:

- x
- costOfShipping
- [Cost of shipping]

Variables that hold lists

ReadiNow calculation variables do not hold values as such. Rather, they are associated with sub-calculations. The distinction is subtle, but it can be helpful in understanding how variables will behave in certain circumstances.

For example, variables that return list results generally cannot be used within summarise functions such as **count** and **sum**. The reason for this is that summarise calculations often need to filter some list of records according to some external criteria that needs to be fixed to some constant value while the summarization is being performed. In the following calculation, the x variable is associated with a simple calculation for determining a maximum allowable transaction size. The **select** result then visits all related transactions, testing the amount of each transaction to the fixed value of x. Correspondingly, this calculation could not make meaninful sense if x were to hold a list of numbers instead.

```
let x = [Department].[Maximum allowable transaction]
select
count( [Transactions] where [Amount] > x )
```

Precedence

Care should be taken to avoid variable names that coincide with field and relationship names. However, if a variable name ends up being the same as a field name, relationship name, or an externally provided workflow variable, then the **let** keywords variable will be selected with the highest priority over the others to minimise the chance of a calculation becoming broken due to unrelated changes such as new fields being added to an object.

Examples

```
let fullName = [First name] + ' ' + [Last name]
select
'Dear ' + fullName + ', we have received your reservation under the name: ' + fullName
```