Controlling the processing of data using partition strategies

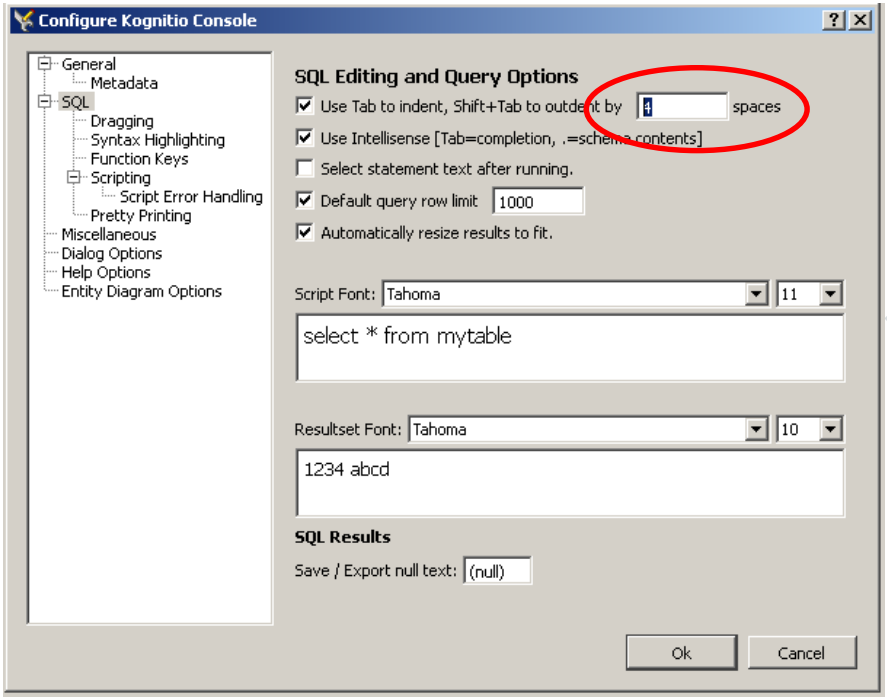
(Part 3 of external script technical training)

Click here to download the supporting SQL script

External Script Training Overview

This is the Python version

1. Introduction to external script interface
2. Controlling Script Processes – threads and nodes
3. Controlling Data Processing – partitions
4. Passing parameters through to scripts



Python is strict about code indentation as it is used to define code within “for”, “if/else” statements etc.

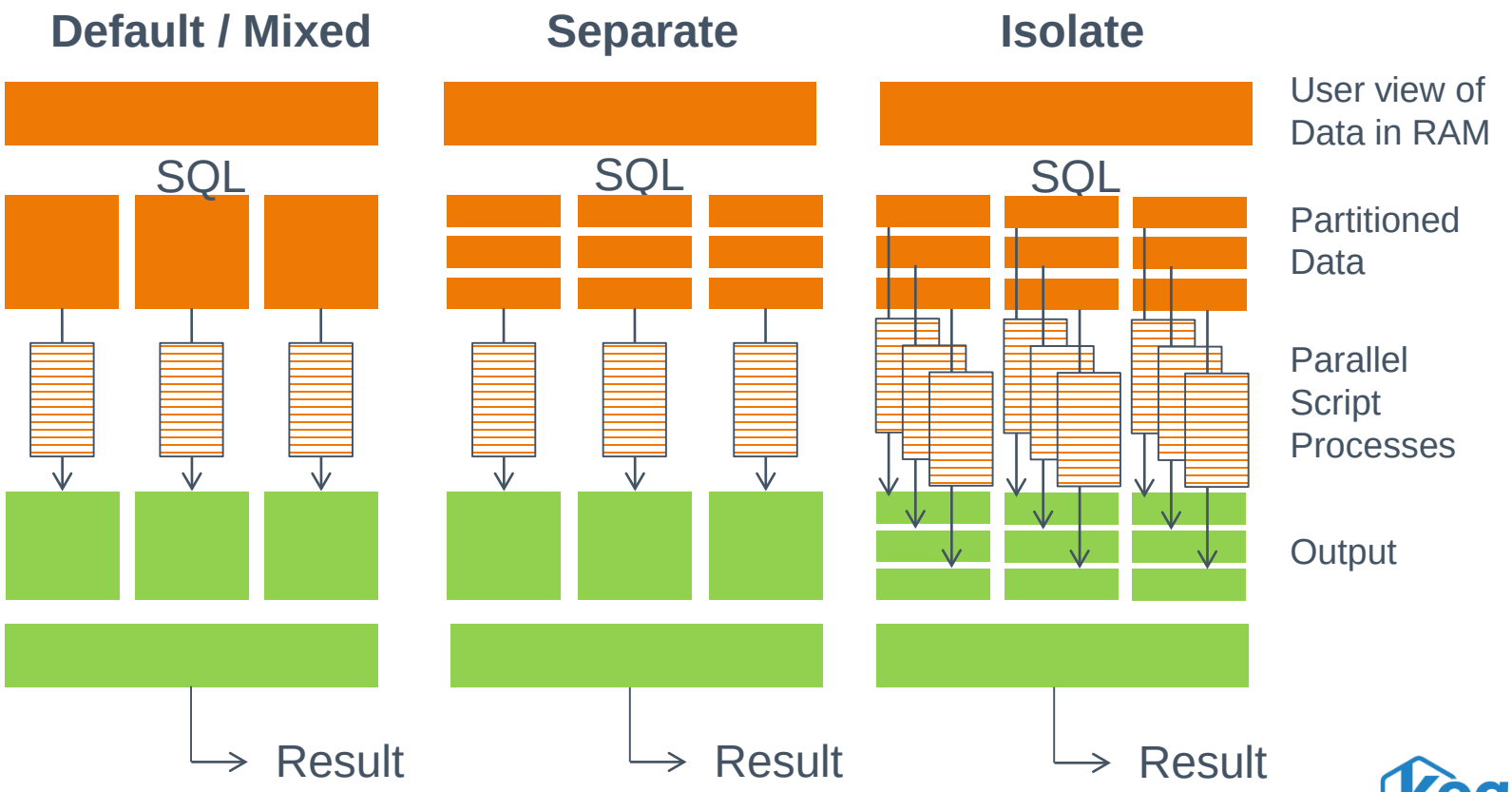
When using Kognitio console set the tab indentation to 4 to match this requirement.

Tools->Configure->SQL

Kognitio

Controlling data processing:
Partition Strategies Overview

- SQL Windowing style syntax controls bucketing and sorting of data
- Scripting modes control processing strategy



Kognitio

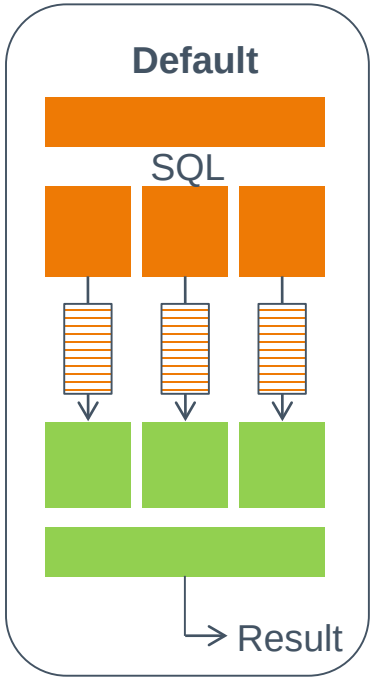
Controlling data processing:
Default Behaviour

```
create external script Kog_P_AvgOverPartID_Def
                  environment python
receives(Part_ID int, value float)
partition by Part_ID
sends(Part_ID int, avg1 float, count1 float)
script S'EOF(
import csv,sys
input = csv.reader(sys.stdin)
result = csv.writer(sys.stdout)
prevpartid=''      #Holds the previous partition id
sum1=0             #Initialise sum
count1=0           #Initialise count

for row in input:
    if len(row)>0:
        if row[0]==prevpartid:
            sum1=sum1+float(row[1])
            count1=count1+1
        else:
            if prevpartid!='':
                avg1=sum1/count1
                result.writerow([prevpartid,avg1,count1])
            prevpartid=row[0]
            sum1=float(row[1])
            count1=1

if count1!=0:
    avg1=sum1/count1
    result.writerow([prevpartid,avg1,count1])
)EOF';
```

Can control how to split the script input by declaring partition columns from receives. Using SQL windowing syntax

Script must detect changes in partition values

Reset vars for each partition



The default partitioning strategy:

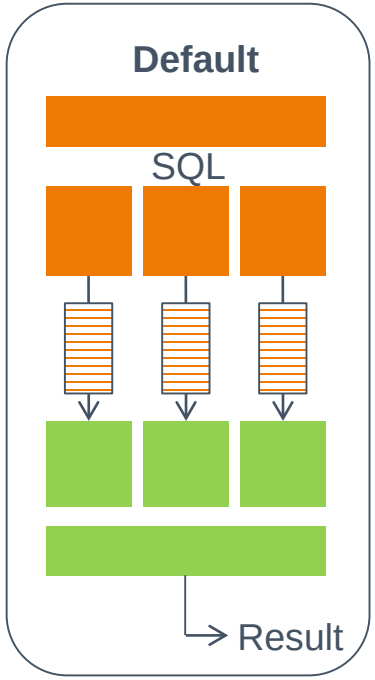
- All rows for a partition are sent to the same script invocation
- Followed by the rows from another partition
- This is done in parallel until all partitions are processed

Kognitio

Controlling data processing:
Default Behaviour

```
select Product_Name, Avg1 Avg_Spend, cast(Count1 as int) NumTrans
from ( external script Kog_P_AvgOverPartID_Def
         from (select PRODNO, PRICE/100.00
               from DEMO_RET.V_RET_SALE
               where storeno between 1 and 5)
        ) sq1
join DEMO_RET.V_RET_PRODUCT p
on sq1.part_id=p.prodno
order by part_id;
```

Prod avgs from the external script are joined to product lookup table to get results



Results (30 rows) | Metadata | Info

	PRODUCT_NAME	AVG_SPEND	NUMTRANS
1	Heinz Beans & Sausage 225G	0.37	21534
2	Batchelors Bigga Peas 300G	0.25	32642
3	Prizewinner Plum Tomatoes 227G	0.14	29042
4	D.Mario Chpd Toms/ Chilli 400G	0.27	26168
5	Morton Diced Carr+Turnip 300G	0.27	25431
6	Ragu Traditional Sauce 2X440G	1.88	17878

For the other partition strategies the python code is amended so that the same query can be run for each to yield the same results

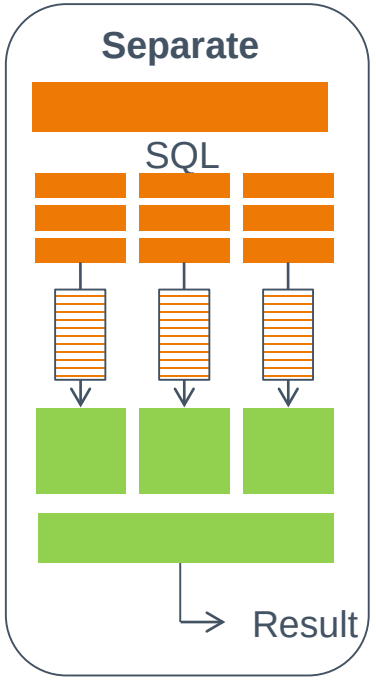
Kognitio

Controlling data processing:
Separate Partitions

```
create external script Kog_P_AvgOverPartID_Sep
                    environment python
receives(Part_ID int, value float)
partition by Part_ID
sends(Part_ID int, avg1 float, count1 float)
separate partitions
script S'EOF(
import csv,sys
input = csv.reader(sys.stdin)
result = csv.writer(sys.stdout)
sum1=0                          #Initialise sum
count1=0                        #Initialise count

for row in input:
    if len(row)>0:
        partid=row[0]
        sum1=sum1+float(row[1])
        count1=count1+1
    else:
        if count1!=0:
            avg1=sum1/count1
            result.writerow([partid,avg1,count1])
            partid=''
            sum1=0
            count1=0

if count1!=0:
    avg1=sum1/count1
    result.writerow([partid,avg1,count1])
)EOF';
```

Separate partition strategies must be declared after the sends command

Need to keep track of the partition value but not previous ones



The separate partitioning strategy:

- Same as default except a blank line is inserted after each partition

Works well for processes that requires all rows from a partition to be read prior to analysis and there are lots of partition values

Kognitio

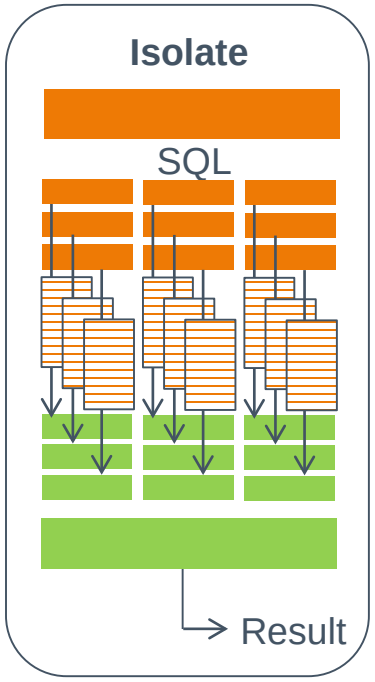
Controlling data processing:

Isolate Partitions

```
create external script Kog_P_AvgOverPartID_Iso
                    environment python
receives(Part_ID int, value float)
partition by Part_ID
sends(Part_ID int, avg1 float, count1 float)
isolate partitions
script S'EOF(
import csv,sys
input = csv.reader(sys.stdin)
result = csv.writer(sys.stdout)
sum1=0
count1=0

for row in input:
    if len(row)>0:
        partid=row[0]
        sum1=sum1+float(row[1])
        count1=count1+1

if count1>0:
    avg1=sum1/count1
    result.writerow([partid,avg1,count1])
)EOF';
```

Isolate partition strategies must be declared after the sends command

Script is more straightforward but remember to output partition value



The isolate partitioning strategy:

- All rows for a partition are sent to a new isolated script invocation
- Initially the max number of invocations are started
- Subsequent script invocations started when an existing one finishes

There is an overhead associated with invoking each external script so this works well for complex processes where this overhead is negligible compared to script execution time.
Also useful when number of partition values is small.

Kognitio

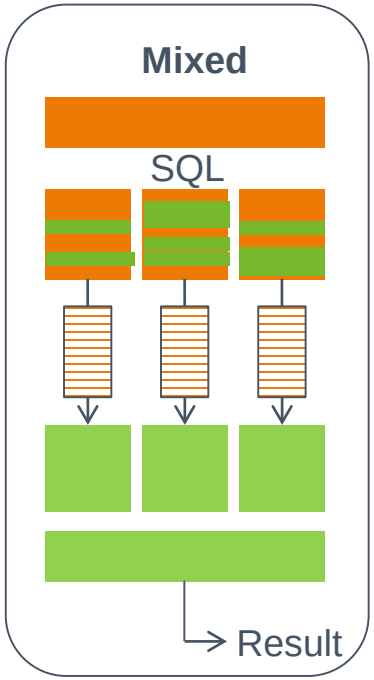
Controlling data processing:
Mixed Partitions

```
create external script Kog_P_AvgOverPartID_Mix environment python
receives(Part_ID int, value float)
partition by Part_ID
sends(Part_ID int, avg1 float, count1 float)
mix partitions
script S'EOF(
import csv,sys
input = csv.reader(sys.stdin)
result = csv.writer(sys.stdout)
partid=[]   #Intialise partition list
sum1=[]                 #Initialise sum list
count1=[]   #Initialise count list
avg1=[]                 #Initialise avg list
for row in input:
    if len(row)>0:
        id=row[0]
        value=float(row[1])
        if partid.count(id)>0:
            i=partid.index(id)
            sum1[i]=sum1[i]+value
            count1[i]=count1[i]+1
        else:
            partid.append(id)
            sum1.append(value)
            count1.append(1)
if len(partid)>0:
    avg1=[0]*len(partid)
    avg1= [x/y for x,y in zip(sum1,count1)]
    for id in partid:
            i=partid.index(id)
            result.writerow([id,avg1[i],count1[i]])
)EOF';
```

Mix partition strategies must be declared after the sends command

Script must keep track of all partitions and values

All results output at end of script



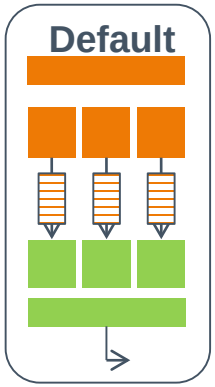
The mixed partitioning strategy:

- Similar to default
- All rows for a partition are sent to the same script invocation but rows for different partitions can be mixed

Kognitio

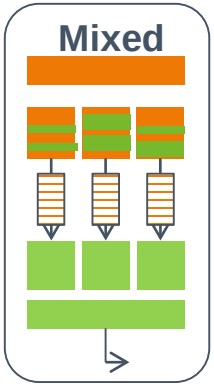
Controlling data processing:
When to use different partitions

Default



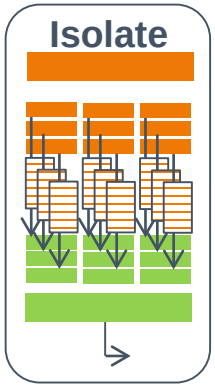
- Data can be streamed through the script without retention
- Number of rows per partition is relatively small (not in 100s millions)
- Output is complex

Mixed



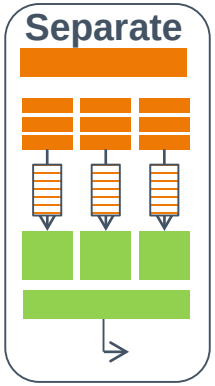
- Do not need to pass all rows for one partition before starting to pass next Minimises pre-script data storage
- Number of rows per partition is relatively large (in 100s millions)
- Output not too complex as needs tracking

Isolate



- Complex process - If overhead for invoking script is small compared to execution time of script
- Number of partitions is small (not in millions)

Separate



- Process requires all rows from a partition to be read **prior** to analysis
- Number of partitions is high (in the millions)

Kognitio