CS 327E Class 6

October 14, 2019

1) PTransforms such as `Pardo` mutate their input elements.

A. True
B. False

2) What kind of object does the `ParDo` transform expect?

A. A `DoFn` subclass
B. A `DoFn` super class
C. A `DoFn` abstract class

3) Does `ParDo` support random access to `PCollection` elements? For example, is the highlighted code allowed?

A. Yes
B. No

```
class ComputeWordLengthFn(beam.DoFn):
  def process(self, element):
    element0 = words[0]
    if len(element0) >= len(element):
       return [element0]


word_lengths = words | beam.ParDo(ComputeWordLengthFn())
```

4) `ParDo` resembles which SQL operation?

A. `FROM clause`
B. `WHERE clause`
C. `ORDER BY clause`
D. `JOIN clause`

5) `CoGroupByKey` resembles which SQL operation?

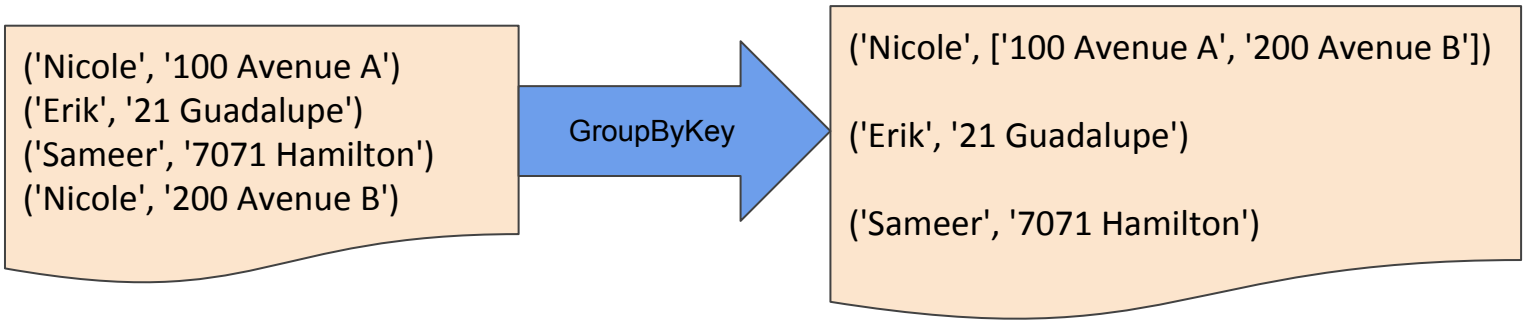
A. `FROM clause`
B. `WHERE clause`
C. `ORDER BY clause`
D. `JOIN clause`

Recall: `ParDo` Transform

- Maps 1 input element to (1, 0, many) output elements
- Invokes a user-specified function on each of the elements of the input `PCollection`
- User code is implemented as a subclass of `DoFn` with a `process(self, element)` method
- Input elements are processed independently and in parallel
- Output elements are bundled into a new `PCollection`
- Typical usage: filtering, formatting, extracting parts of data, performing computations on data elements

`GroupByKey` Transform

- Takes a `PCollection` as input where each element is a (key, value) pair
- Groups the values by unique key
- Produces a `PCollection` as output where each element is a (key, list(value)) pair
- Resembles `GROUP BY` in SQL



Demo: `Student_single.py`

```
git pull origin master
```

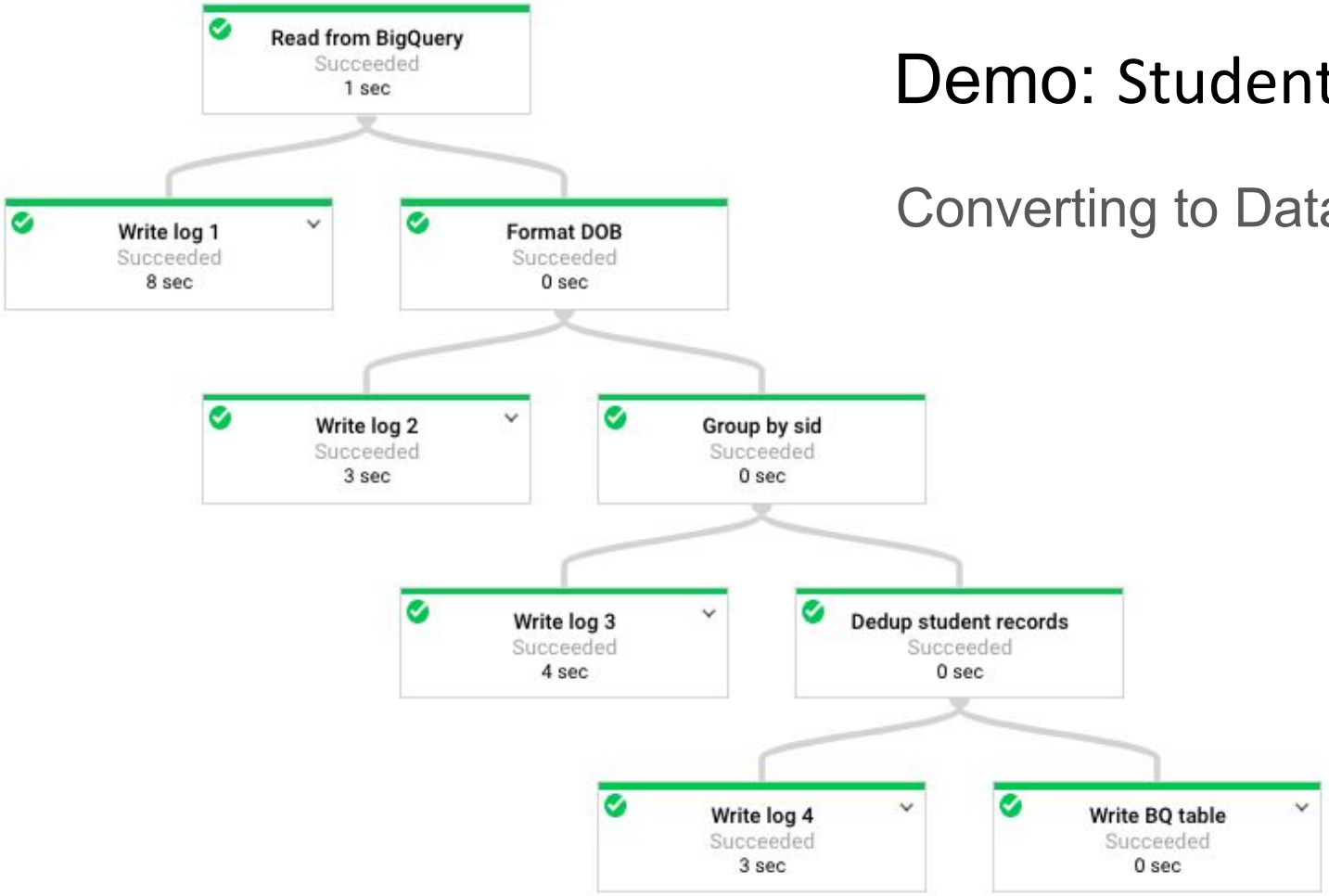
Hands-on Exercise 1

Run `Student_single.py`

iClicker Question 1

How many records are in the resulting Student table?

A. 0
B. 12
C. 15



Demo: Student_cluster.py

Converting to Dataflow pipeline

Hands-on Exercise 2

Create `Teacher_cluster.py` from `Teacher_single.py`

Run `Teacher_cluster.py` on Dataflow

iClicker Question 2

How many nodes are in the job's execution graph?

A. 3
B. 4
C. 9

`ParDo` Side Inputs

- A side input is an optional input passed to `DoFn`
- Passed as an extra argument to `process` method:

```
process(self, element, side_input1)
```

- Side inputs can be ordinary values or entire `PCollections`
- `DoFn` reads side inputs while processing an individual element
- Multiple side inputs per `DoFn` are supported:

```
process(self, element, side_input1, side_input2,
        side_inputn)
```

Demo: `Takes_single.py`

Show Side Inputs

`Flatten` Transform

- Takes a list of `PCollections` as input
- Produces a single `PCollection` as output
- Results contain all the elements from the input `PCollections`
- Note: Input `PCollections` must have matching schemas

```
a_pcoll = p | 'Read File 1' >> ReadFromText('oscars_data_archive.tsv')
b_pcoll = p | 'Read File 2' >> ReadFromText('oscars_data_2019.tsv')

# Union the two PCollections
c_pcoll = (a_pcoll, b_pcoll) | 'Merge PCollections'  >> beam.Flatten()
```

`CoGroupByKey` Transform

- Takes two or more `PCollections` as input
- Every element in the input is a (key, value) pair
- Groups values from all input `PCollections` by common key
- Produces a `PCollection` as output where each element is a (key, value) pair
- Output value is a list of dictionaries containing all data associated with unique key
- Analogous to the `FULL OUTER JOIN` in SQL

`CoGroupByKey` Transform

```
q1 = 'SELECT sid, cno, grade FROM college_modeled.Takes'
q2 = 'SELECT cno, cname FROM college_modeled.Class'

takes_pcoll = p | 'Run Q1' >> beam.io.Read(beam.io.BigQuerySource( query=q1))
class_pcoll = p | 'Run Q2' >> beam.io.Read(beam.io.BigQuerySource( query=q2))

takes_tuple = takes_pcoll | 'Takes Tuple' >> beam.ParDo(MakeTuple())
class_tuple = class_pcoll | 'Class Tuple' >> beam.ParDo(MakeTuple())

joined_pcoll = (takes_tuple, class_tuple) | 'Join' >> beam.CoGroupByKey()
```

Milestone 6

1) Requirements and rubric: assignment sheet

2) Debugging assistance: sign-up sheet