# CS 327E Class 8

November 12, 2018

1) The individual elements of a `PCollection` are **not** accessible by Beam Transforms.

A. True
B. False

## 2) Which Beam Transform can contain a boolean condition that specifies which elements from the input `PCollection` should be in the output `PCollection`?

A. `ParDo`
B. `GroupByKey`
C. `CoGroupByKey`
D. `Flatten`
E. None of the above

## 3) Which Beam Transform is equivalent to an `ORDER BY` clause in SQL?

A. `ParDo`
B. `GroupByKey`
C. `CoGroupByKey`
D. `Flatten`
E. None of the above

## 4) Which Beam Transform is equivalent to a `JOIN` in SQL?

A. `ParDo`
B. `GroupByKey`
C. `CoGroupByKey`
D. `Flatten`
E. None of the above

## 5) Which statement is **True** about the `GroupByKey` Transform?

A. `GroupByKey` groups all the elements in the input `PCollection` except for the first and last elements.

B. `GroupByKey` expects the elements of the input `PCollection` to contain multiple types (e.g. String, Integer, etc.).

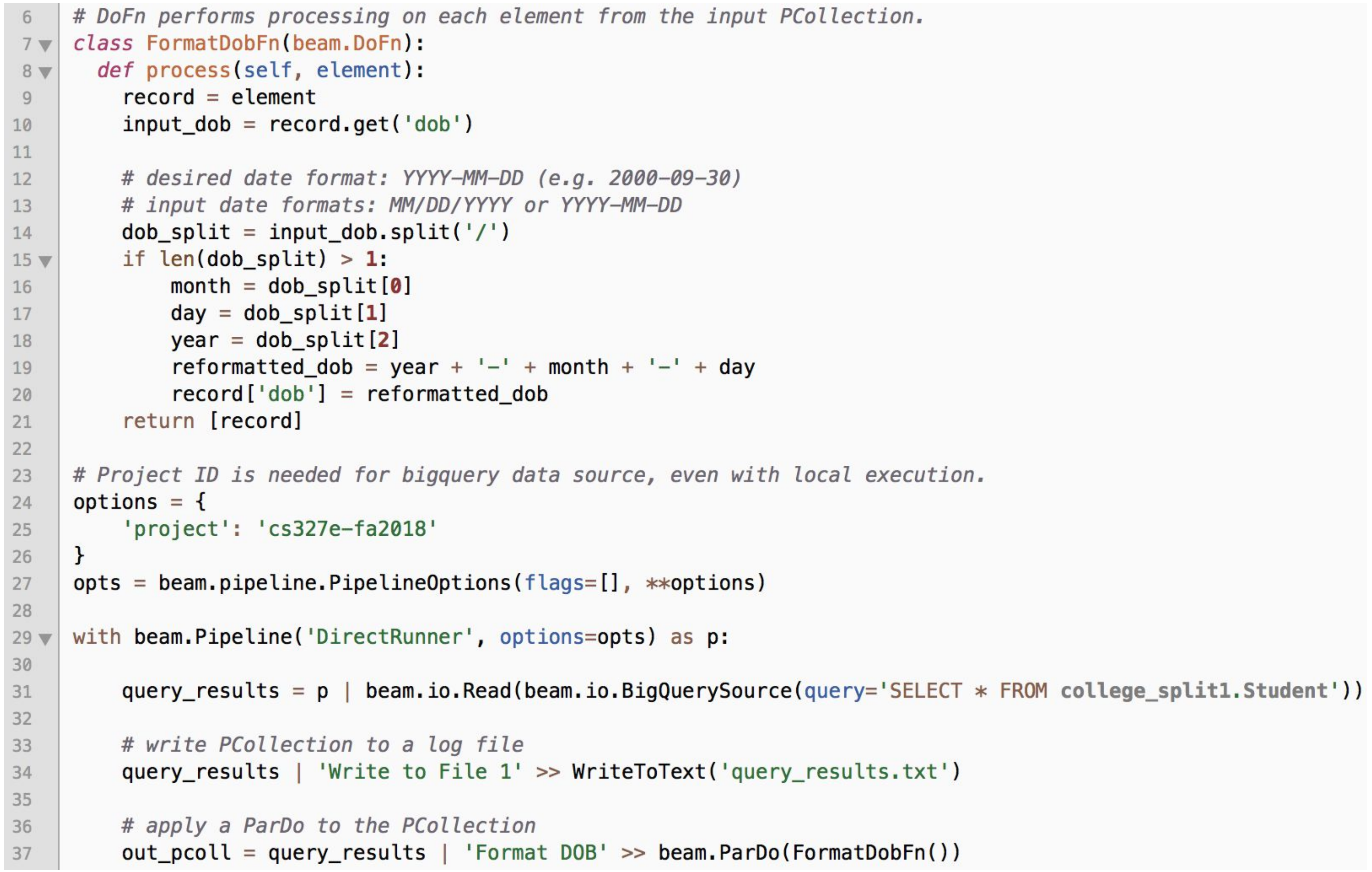
C. `GroupByKey` expects the elements of the input `PCollection` to be shaped as a (key, value) pair.

D. `GroupByKey` is analogous to a `GROUP BY` clause in SQL.

## 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` containing a user-defined function `process(self, element)`
- 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

## ParDo Example



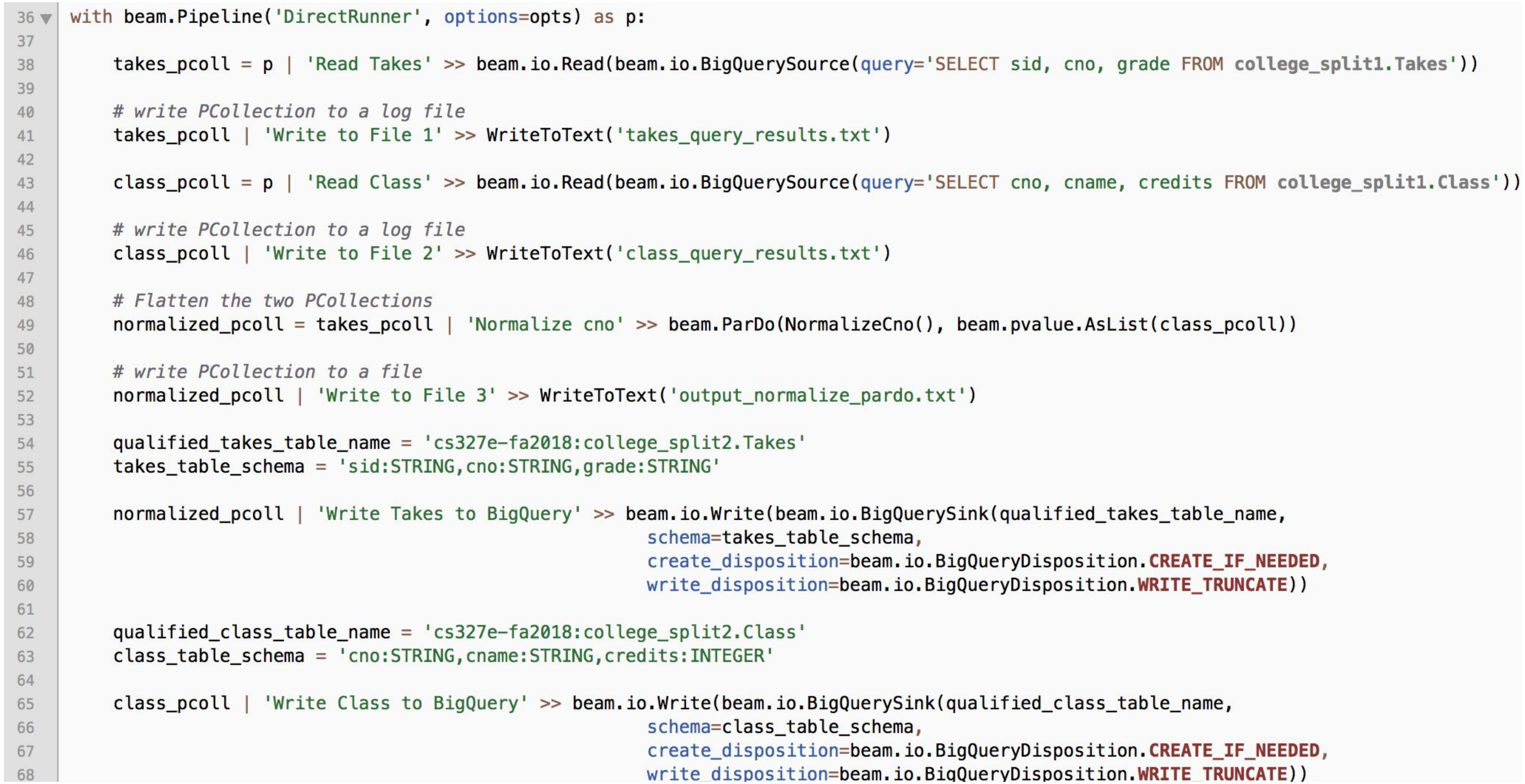
```python
# DoFn performs processing on each element from the input PCollection.
class FormatDobFn(beam.DoFn):
  def process(self, element):
    record = element
    input_dob = record.get('dob')

    # desired date format: YYYY-MM-DD (e.g. 2000-09-30)
    # input date formats: MM/DD/YYYY or YYYY-MM-DD
    dob_split = input_dob.split('/')
    if len(dob_split) > 1:
        month = dob_split[0]
        day = dob_split[1]
        year = dob_split[2]
        reformatted_dob = year + '-' + month + '-' + day
        record['dob'] = reformatted_dob
    return [record]

# Project ID is needed for bigquery data source, even with local execution.
options = {
    'project': 'cs327e-fa2018'
}
opts = beam.pipeline.PipelineOptions(flags=[], **options)

with beam.Pipeline('DirectRunner', options=opts) as p:

    query_results = p | beam.io.Read(beam.io.BigQuerySource(query='SELECT * FROM college_split1.Student'))

    # write PCollection to a log file
    query_results | 'Write to File 1' >> WriteToText('query_results.txt')

    # apply a ParDo to the PCollection
    out_pcoll = query_results | 'Format DOB' >> beam.ParDo(FormatDobFn())
```

Source File: https://github.com/cs327e-fall2018/snippets/blob/master/format_student_dob.py

## ParDo Side Input

- An optional input passed to `ParDo`’s `DoFn`
- Side input can be ordinary values or entire `PCollection`
- `DoFn` reads side input while processing an element
- Can have multiple side inputs per `DoFn`
- Passed as extra arguments to `process(self, element, side_input1, side_input2 ...)`

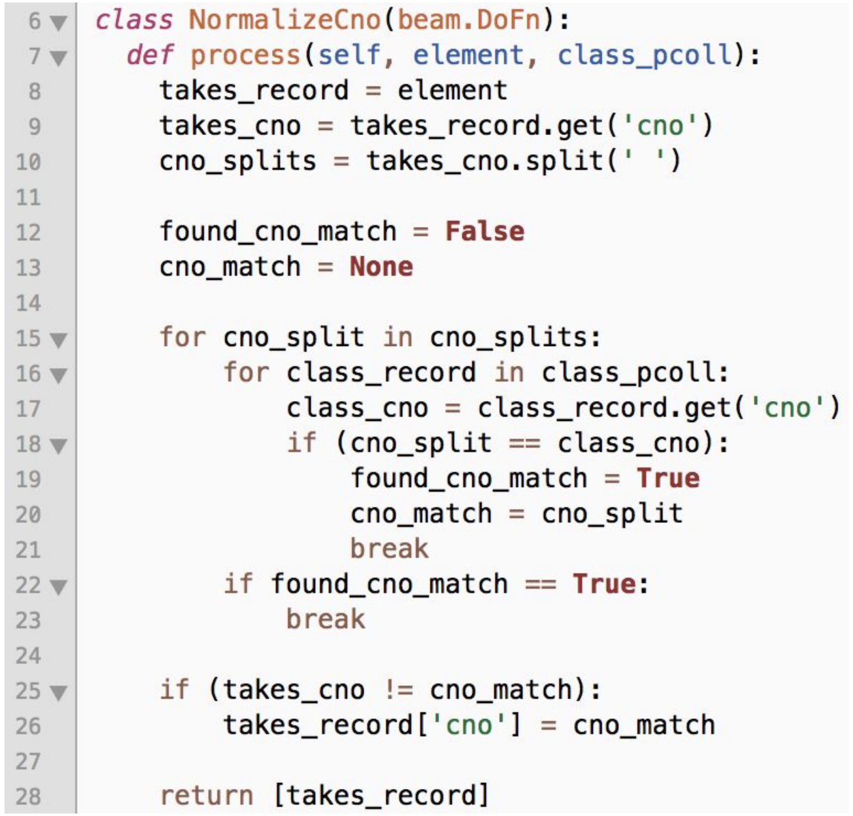
# Pardo with Side Input Example



```python
with beam.Pipeline('DirectRunner', options=opts) as p:

    takes_pcoll = p | 'Read Takes' >> beam.io.Read(beam.io.BigQuerySource(query='SELECT sid, cno, grade FROM college_split1.Takes'))

    # write PCollection to a log file
    takes_pcoll | 'Write to File 1' >> WriteToText('takes_query_results.txt')

    class_pcoll = p | 'Read Class' >> beam.io.Read(beam.io.BigQuerySource(query='SELECT cno, cname, credits FROM college_split1.Class'))

    # write PCollection to a log file
    class_pcoll | 'Write to File 2' >> WriteToText('class_query_results.txt')

    # Flatten the two PCollections
    normalized_pcoll = takes_pcoll | 'Normalize cno' >> beam.ParDo(NormalizeCno(), beam.pvalue.AsList(class_pcoll))

    # write PCollection to a file
    normalized_pcoll | 'Write to File 3' >> WriteToText('output_normalize_pardo.txt')

    qualified_takes_table_name = 'cs327e-fa2018:college_split2.Takes'
    takes_table_schema = 'sid:STRING,cno:STRING,grade:STRING'

    normalized_pcoll | 'Write Takes to BigQuery' >> beam.io.Write(beam.io.BigQuerySink(qualified_takes_table_name,
                                                      schema=takes_table_schema,
                                                      create_disposition=beam.io.BigQueryDisposition.CREATE_IF_NEEDED,
                                                      write_disposition=beam.io.BigQueryDisposition.WRITE_TRUNCATE))

    qualified_class_table_name = 'cs327e-fa2018:college_split2.Class'
    class_table_schema = 'cno:STRING,cname:STRING,credits:INTEGER'

    class_pcoll | 'Write Class to BigQuery' >> beam.io.Write(beam.io.BigQuerySink(qualified_class_table_name,
                                                      schema=class_table_schema,
                                                      create_disposition=beam.io.BigQueryDisposition.CREATE_IF_NEEDED,
                                                      write_disposition=beam.io.BigQueryDisposition.WRITE_TRUNCATE))
```

Source File: https://github.com/cs327e-fall2018/snippets/blob/master/normalize_takes_cno.py

# ParDo and Side Input Example



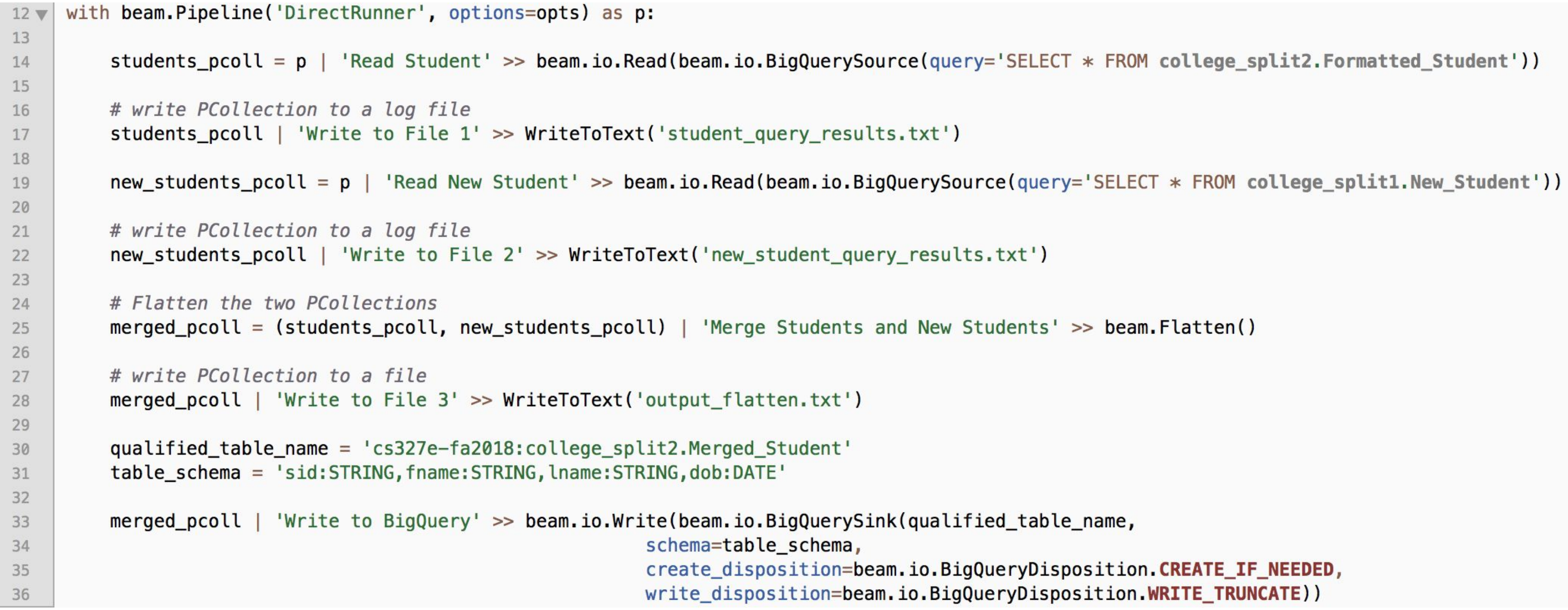
```python
class NormalizeCno(beam.DoFn):
  def process(self, element, class_pcoll):
    takes_record = element
    takes_cno = takes_record.get('cno')
    cno_splits = takes_cno.split(' ')

    found_cno_match = False
    cno_match = None

    for cno_split in cno_splits:
        for class_record in class_pcoll:
            class_cno = class_record.get('cno')
            if (cno_split == class_cno):
                found_cno_match = True
                cno_match = cno_split
                break
        if found_cno_match == True:
            break

    if (takes_cno != cno_match):
        takes_record['cno'] = cno_match

    return [takes_record]
```

Source File: https://github.com/cs327e-fall2018/snippets/blob/master/normalize_takes_cno.py

## 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

# Flatten Example



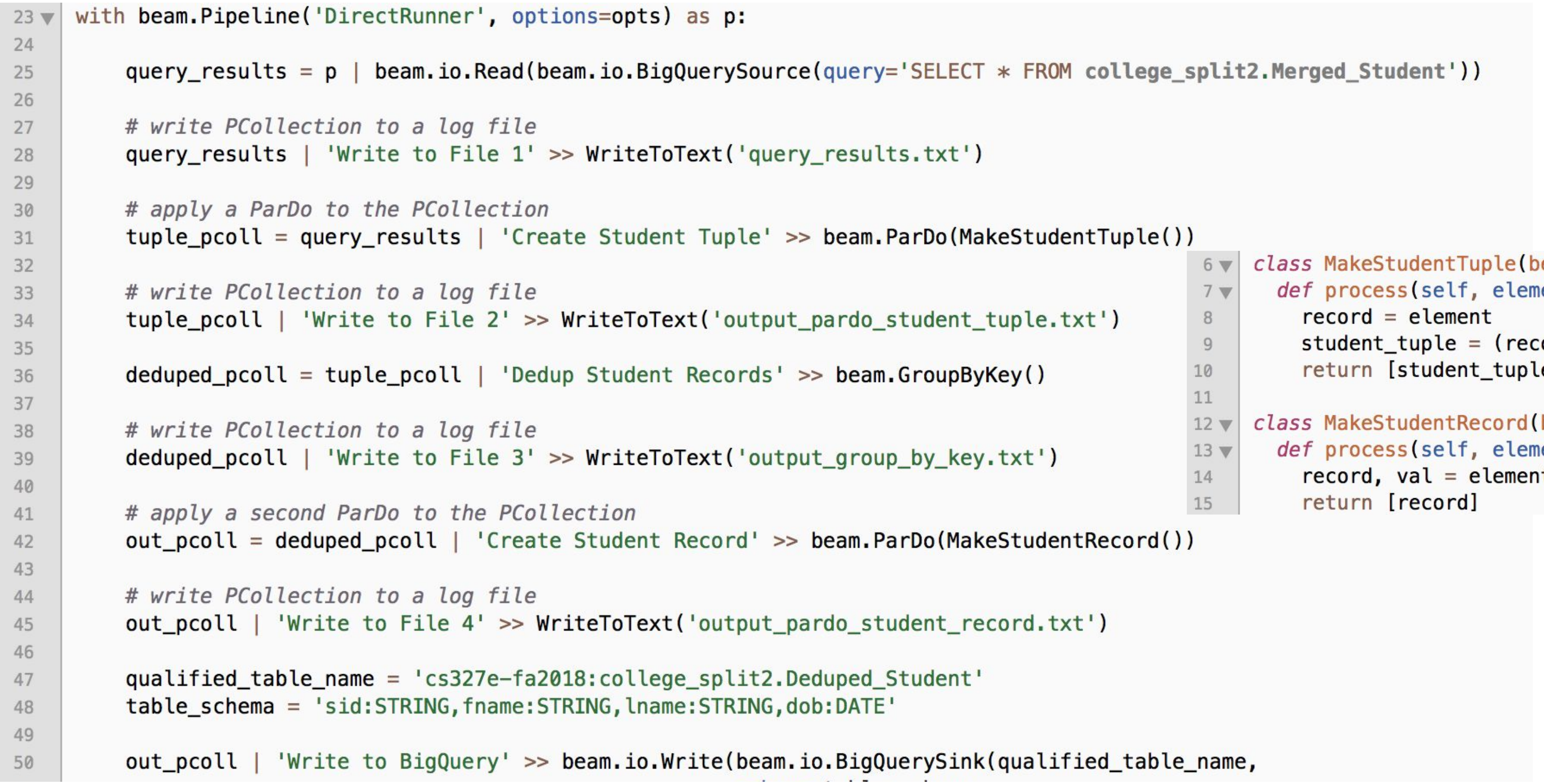
```python
with beam.Pipeline('DirectRunner', options=opts) as p:

    students_pcoll = p | 'Read Student' >> beam.io.Read(beam.io.BigQuerySource(query='SELECT * FROM college_split2.Formatted_Student'))

    # write PCollection to a log file
    students_pcoll | 'Write to File 1' >> WriteToText('student_query_results.txt')

    new_students_pcoll = p | 'Read New Student' >> beam.io.Read(beam.io.BigQuerySource(query='SELECT * FROM college_split1.New_Student'))

    # write PCollection to a log file
    new_students_pcoll | 'Write to File 2' >> WriteToText('new_student_query_results.txt')

    # Flatten the two PCollections
    merged_pcoll = (students_pcoll, new_students_pcoll) | 'Merge Students and New Students' >> beam.Flatten()

    # write PCollection to a file
    merged_pcoll | 'Write to File 3' >> WriteToText('output_flatten.txt')

    qualified_table_name = 'cs327e-fa2018:college_split2.Merged_Student'
    table_schema = 'sid:STRING,fname:STRING,lname:STRING,dob:DATE'

    merged_pcoll | 'Write to BigQuery' >> beam.io.Write(beam.io.BigQuerySink(qualified_table_name,
                                                       schema=table_schema,
                                                       create_disposition=beam.io.BigQueryDisposition.CREATE_IF_NEEDED,
                                                       write_disposition=beam.io.BigQueryDisposition.WRITE_TRUNCATE))
```

Source File: https://github.com/cs327e-fall2018/snippets/blob/master/merge_student_tables.py

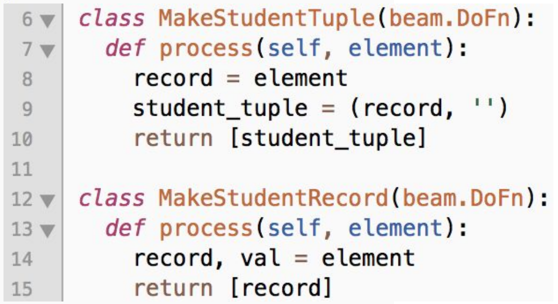
## GroupByKey Transform

- 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
- Related, but not analogous to `GROUP BY` in SQL

# GroupByKey Example



```python
with beam.Pipeline('DirectRunner', options=opts) as p:

    query_results = p | beam.io.Read(beam.io.BigQuerySource(query='SELECT * FROM college_split2.Merged_Student'))

    # write PCollection to a log file
    query_results | 'Write to File 1' >> WriteToText('query_results.txt')

    # apply a ParDo to the PCollection
    tuple_pcoll = query_results | 'Create Student Tuple' >> beam.ParDo(MakeStudentTuple())

    # write PCollection to a log file
    tuple_pcoll | 'Write to File 2' >> WriteToText('output_pardo_student_tuple.txt')

    deduped_pcoll = tuple_pcoll | 'Dedup Student Records' >> beam.GroupByKey()

    # write PCollection to a log file
    deduped_pcoll | 'Write to File 3' >> WriteToText('output_group_by_key.txt')

    # apply a second ParDo to the PCollection
    out_pcoll = deduped_pcoll | 'Create Student Record' >> beam.ParDo(MakeStudentRecord())

    # write PCollection to a log file
    out_pcoll | 'Write to File 4' >> WriteToText('output_pardo_student_record.txt')

    qualified_table_name = 'cs327e-fa2018:college_split2.Deduped_Student'
    table_schema = 'sid:STRING,fname:STRING,lname:STRING,dob:DATE'

    out_pcoll | 'Write to BigQuery' >> beam.io.Write(beam.io.BigQuerySink(qualified_table_name,
```



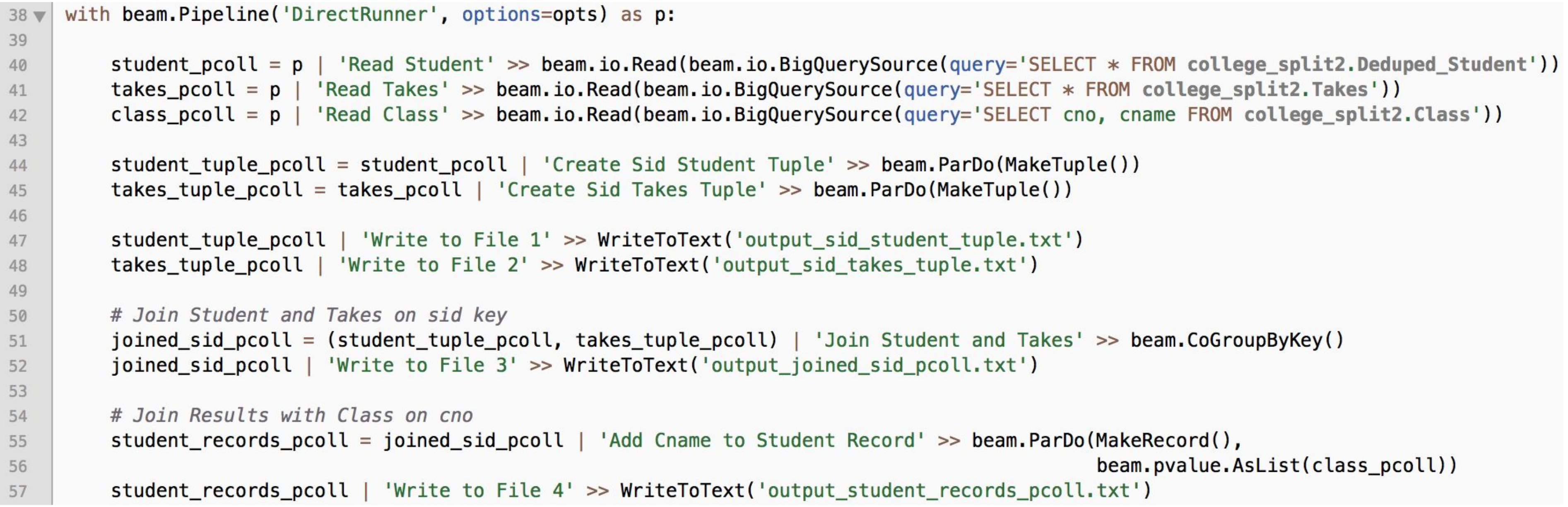
```python
class MakeStudentTuple(beam.DoFn):
  def process(self, element):
    record = element
    student_tuple = (record, '')
    return [student_tuple]

class MakeStudentRecord(beam.DoFn):
  def process(self, element):
    record, val = element
    return [record]
```

Source File: https://github.com/cs327e-fall2018/snippets/blob/master/dedup_student_table.py

## 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 tuple of dictionary lists containing all data associated with unique key
- Analogous to a `FULL OUTER JOIN` in SQL

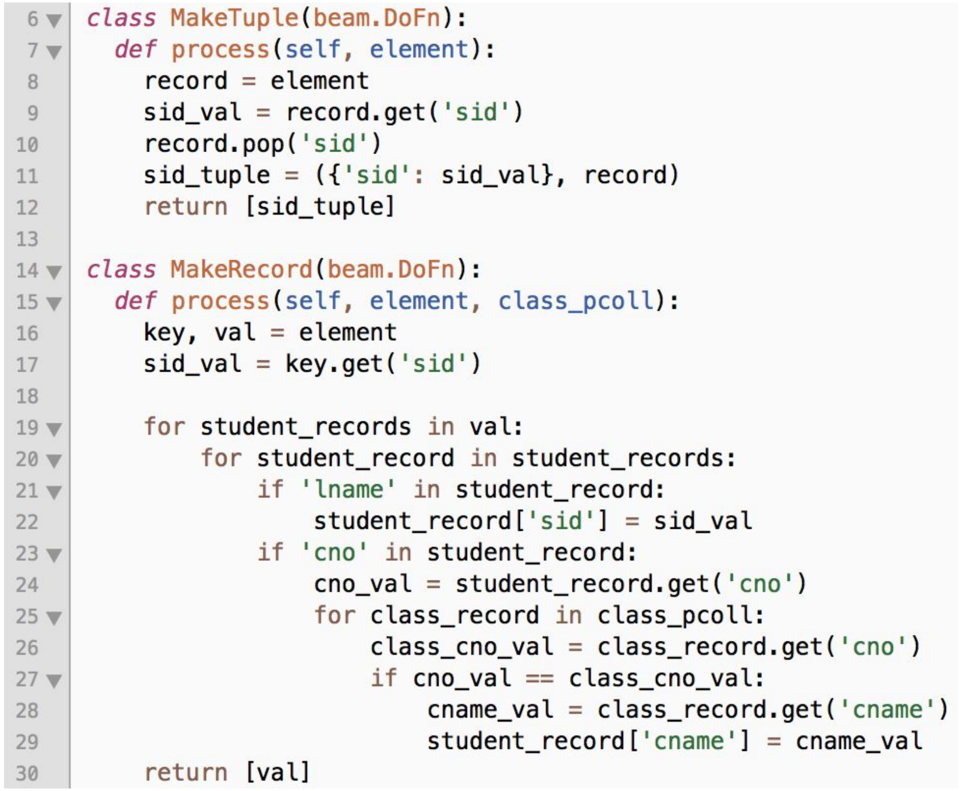
# CoGroupByKey Transform



```python
with beam.Pipeline('DirectRunner', options=opts) as p:

    student_pcoll = p | 'Read Student' >> beam.io.Read(beam.io.BigQuerySource(query='SELECT * FROM college_split2.Deduped_Student'))
    takes_pcoll = p | 'Read Takes' >> beam.io.Read(beam.io.BigQuerySource(query='SELECT * FROM college_split2.Takes'))
    class_pcoll = p | 'Read Class' >> beam.io.Read(beam.io.BigQuerySource(query='SELECT cno, cname FROM college_split2.Class'))

    student_tuple_pcoll = student_pcoll | 'Create Sid Student Tuple' >> beam.ParDo(MakeTuple())
    takes_tuple_pcoll = takes_pcoll | 'Create Sid Takes Tuple' >> beam.ParDo(MakeTuple())

    student_tuple_pcoll | 'Write to File 1' >> WriteToText('output_sid_student_tuple.txt')
    takes_tuple_pcoll | 'Write to File 2' >> WriteToText('output_sid_takes_tuple.txt')

    # Join Student and Takes on sid key
    joined_sid_pcoll = (student_tuple_pcoll, takes_tuple_pcoll) | 'Join Student and Takes' >> beam.CoGroupByKey()
    joined_sid_pcoll | 'Write to File 3' >> WriteToText('output_joined_sid_pcoll.txt')

    # Join Results with Class on cno
    student_records_pcoll = joined_sid_pcoll | 'Add Cname to Student Record' >> beam.ParDo(MakeRecord(),
                                                                                           beam.pvalue.AsList(class_pcoll))
    student_records_pcoll | 'Write to File 4' >> WriteToText('output_student_records_pcoll.txt')
```

Source File: https://github.com/cs327e-fall2018/snippets/blob/master/create_student_view.py

# CoGroupByKey Example



```python
class MakeTuple(beam.DoFn):
  def process(self, element):
    record = element
    sid_val = record.get('sid')
    record.pop('sid')
    sid_tuple = ({'sid': sid_val}, record)
    return [sid_tuple]

class MakeRecord(beam.DoFn):
  def process(self, element, class_pcoll):
    key, val = element
    sid_val = key.get('sid')

    for student_records in val:
        for student_record in student_records:
            if 'lname' in student_record:
                student_record['sid'] = sid_val
            if 'cno' in student_record:
                cno_val = student_record.get('cno')
                for class_record in class_pcoll:
                    class_cno_val = class_record.get('cno')
                    if cno_val == class_cno_val:
                        cname_val = class_record.get('cname')
                        student_record['cname'] = cname_val
    return [val]
```

Source File: https://github.com/cs327e-fall2018/snippets/blob/master/create_student_view.py

## First Problem

*Normalize the instructor values in the Teacher table.*

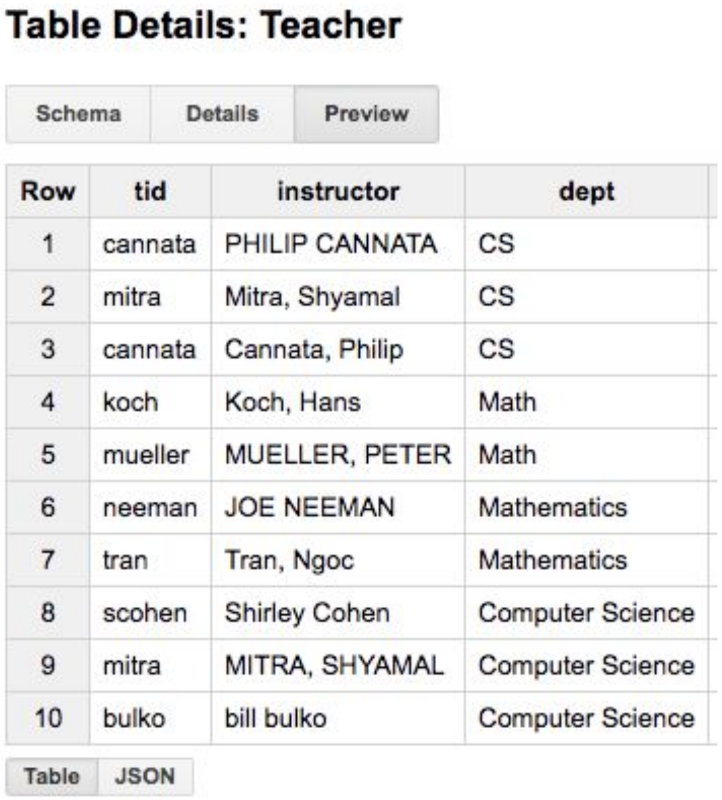


**Table Details: Teacher**

| Row | tid | instructor | dept |
|---|---|---|---|
| 1 | cannata | PHILIP CANNATA | CS |
| 2 | mitra | Mitra, Shyamal | CS |
| 3 | cannata | Cannata, Philip | CS |
| 4 | koch | Koch, Hans | Math |
| 5 | mueller | MUELLER, PETER | Math |
| 6 | neeman | JOE NEEMAN | Mathematics |
| 7 | tran | Tran, Ngoc | Mathematics |
| 8 | scohen | Shirley Cohen | Computer Science |
| 9 | mitra | MITRA, SHYAMAL | Computer Science |
| 10 | bulko | bill bulko | Computer Science |

# iClicker Question

*Normalize the instructor values in the Teacher table.*

Which Beam Transform is involved in this type of processing?

A. `ParDo`
B. `GroupByKey`
C. `CoGroupByKey`
D. `Flatten`

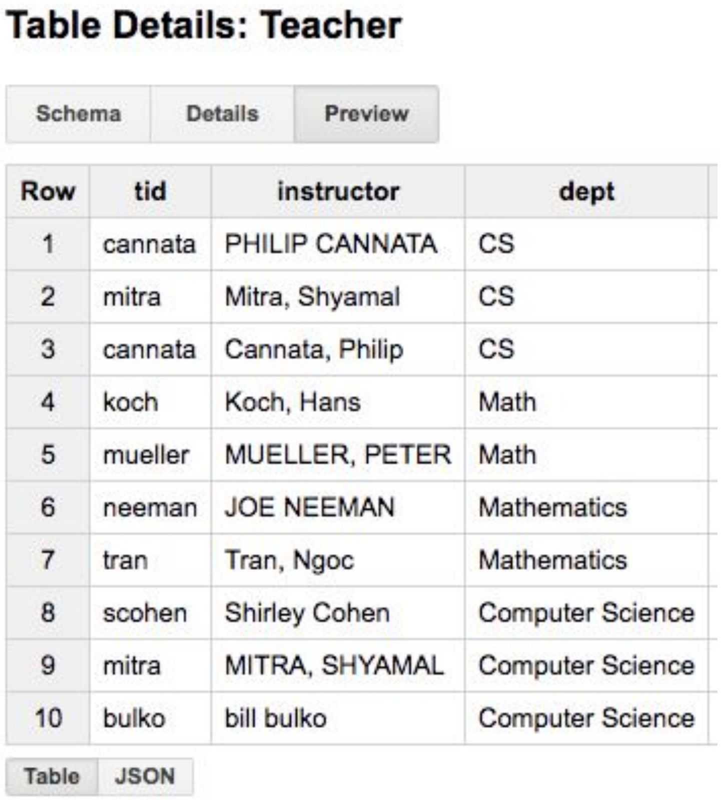


Table Details: Teacher

Schema | Details | Preview

| Row | tid | instructor | dept |
|---|---|---|---|
| 1 | cannata | PHILIP CANNATA | CS |
| 2 | mitra | Mitra, Shyamal | CS |
| 3 | cannata | Cannata, Philip | CS |
| 4 | koch | Koch, Hans | Math |
| 5 | mueller | MUELLER, PETER | Math |
| 6 | neeman | JOE NEEMAN | Mathematics |
| 7 | tran | Tran, Ngoc | Mathematics |
| 8 | scohen | Shirley Cohen | Computer Science |
| 9 | mitra | MITRA, SHYAMAL | Computer Science |
| 10 | bulko | bill bulko | Computer Science |

Table | JSON

## Second Problem

*Normalize the dept values in the Teacher table.*

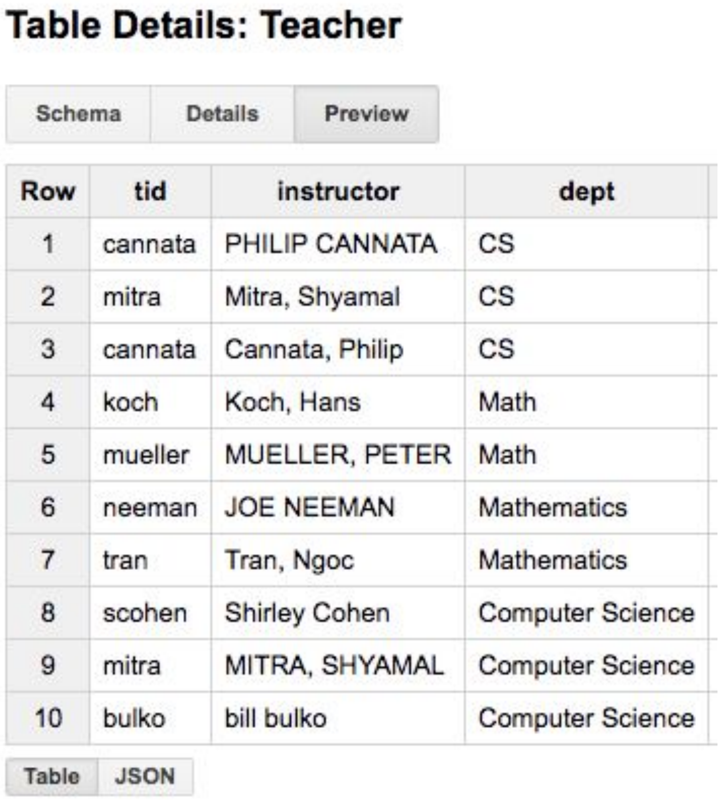


**Table Details: Teacher**

| Row | tid | instructor | dept |
|---|---|---|---|
| 1 | cannata | PHILIP CANNATA | CS |
| 2 | mitra | Mitra, Shyamal | CS |
| 3 | cannata | Cannata, Philip | CS |
| 4 | koch | Koch, Hans | Math |
| 5 | mueller | MUELLER, PETER | Math |
| 6 | neeman | JOE NEEMAN | Mathematics |
| 7 | tran | Tran, Ngoc | Mathematics |
| 8 | scohen | Shirley Cohen | Computer Science |
| 9 | mitra | MITRA, SHYAMAL | Computer Science |
| 10 | bulko | bill bulko | Computer Science |

# iClicker Question

*Normalize the dept values in the Teacher table.*

Which Beam Transform is involved in this type of processing?

A. `ParDo`
B. `GroupByKey`
C. `CoGroupByKey`
D. `Flatten`

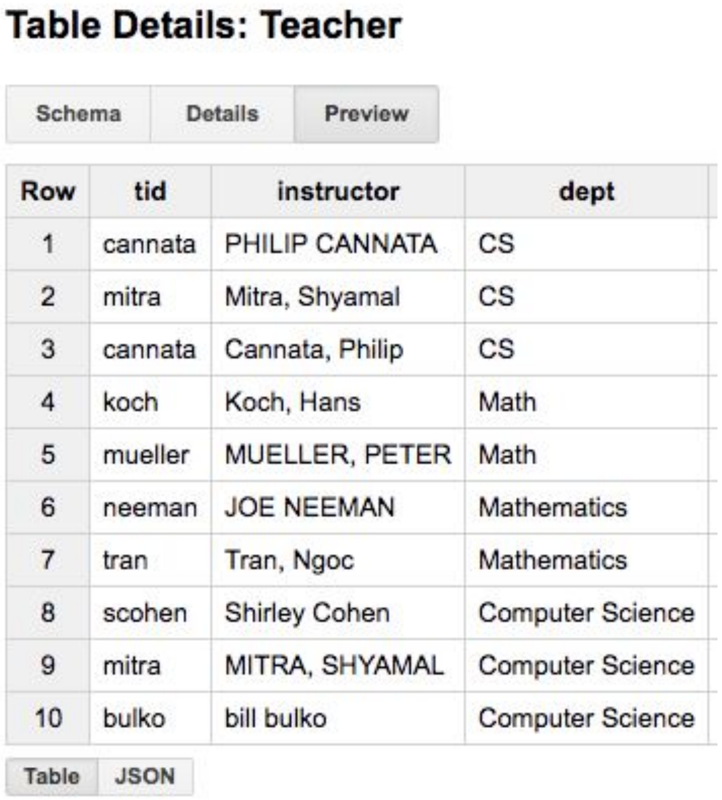


**Table Details: Teacher**

| Row | tid | instructor | dept |
|---|---|---|---|
| 1 | cannata | PHILIP CANNATA | CS |
| 2 | mitra | Mitra, Shyamal | CS |
| 3 | cannata | Cannata, Philip | CS |
| 4 | koch | Koch, Hans | Math |
| 5 | mueller | MUELLER, PETER | Math |
| 6 | neeman | JOE NEEMAN | Mathematics |
| 7 | tran | Tran, Ngoc | Mathematics |
| 8 | scohen | Shirley Cohen | Computer Science |
| 9 | mitra | MITRA, SHYAMAL | Computer Science |
| 10 | bulko | bill bulko | Computer Science |

## Third Problem

*Remove duplicate records from the Teacher table such that each instructor is stored only once.*

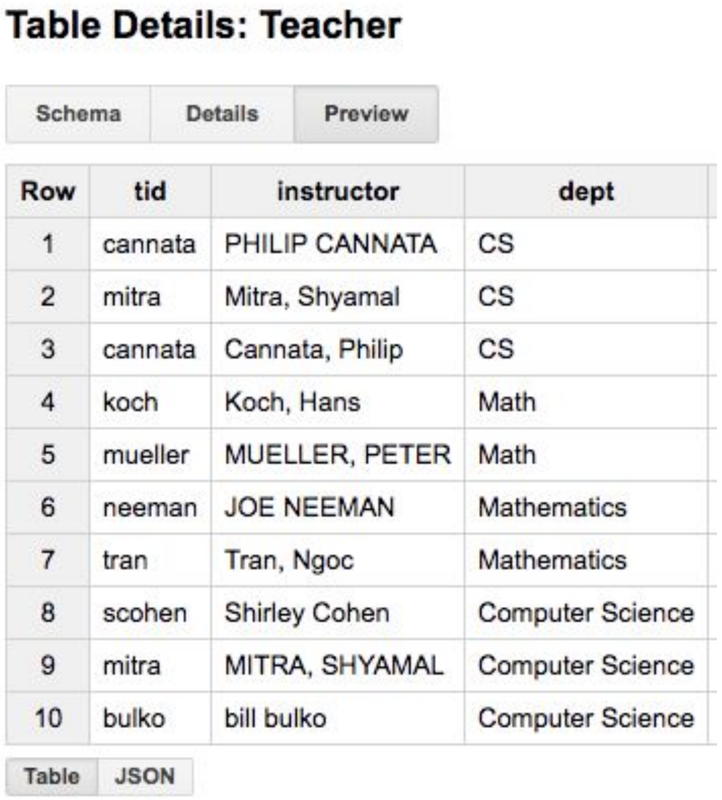


**Table Details: Teacher**

Schema | Details | Preview

| Row | tid | instructor | dept |
|---|---|---|---|
| 1 | cannata | PHILIP CANNATA | CS |
| 2 | mitra | Mitra, Shyamal | CS |
| 3 | cannata | Cannata, Philip | CS |
| 4 | koch | Koch, Hans | Math |
| 5 | mueller | MUELLER, PETER | Math |
| 6 | neeman | JOE NEEMAN | Mathematics |
| 7 | tran | Tran, Ngoc | Mathematics |
| 8 | scohen | Shirley Cohen | Computer Science |
| 9 | mitra | MITRA, SHYAMAL | Computer Science |
| 10 | bulko | bill bulko | Computer Science |

Table | JSON

# iClicker Question

*Remove duplicate records from the Teacher table such that each instructor is stored only once.*

Which Beam Transform(s) is involved in this type of processing?

A. `ParDo`
B. `ParDo and GroupByKey`
C. `GroupByKey`

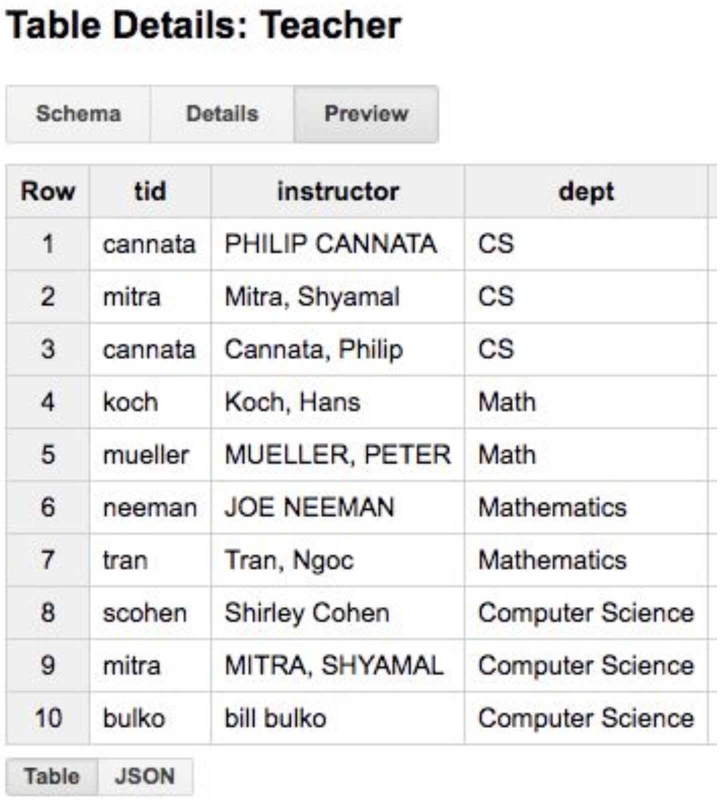


**Table Details: Teacher**

| Row | tid | instructor | dept |
|---|---|---|---|
| 1 | cannata | PHILIP CANNATA | CS |
| 2 | mitra | Mitra, Shyamal | CS |
| 3 | cannata | Cannata, Philip | CS |
| 4 | koch | Koch, Hans | Math |
| 5 | mueller | MUELLER, PETER | Math |
| 6 | neeman | JOE NEEMAN | Mathematics |
| 7 | tran | Tran, Ngoc | Mathematics |
| 8 | scohen | Shirley Cohen | Computer Science |
| 9 | mitra | MITRA, SHYAMAL | Computer Science |
| 10 | bulko | bill bulko | Computer Science |

# Milestone 7 Hints

Part 1:

- Your cross-dataset query descriptions should be clear, concise, and compelling.
- They will drive the requirements for Milestones 8 - 10.
- Get feedback on your cross-dataset queries next class by signing-up for a short review session.

Part 2:

- Review the Beam code samples in our snippets repo
- Run code samples on your environment by following instructions in README
- Sample data for your Beam Transforms can come from either a text file or BigQuery query