Working with Advanced Data Transformations

Overview

Understand the group by keyword and use aggregations on field values

Use joins to combine matching records from multiple relations

Use the union command to combine records together into one relation

Extract entities in bags into discrete records using the flatten command

Use real world data from the City of New York to perform analysis

Demo

Access and download the data for accident information for the City of New York

Grouping Records on the Same Key

Group By

ID	Product_ID	Quantity	Amount
o1	phone	1	199
o1	shoes	1	69
o2	book	2	22
o3	phone	1	149
o3	belt	2	19

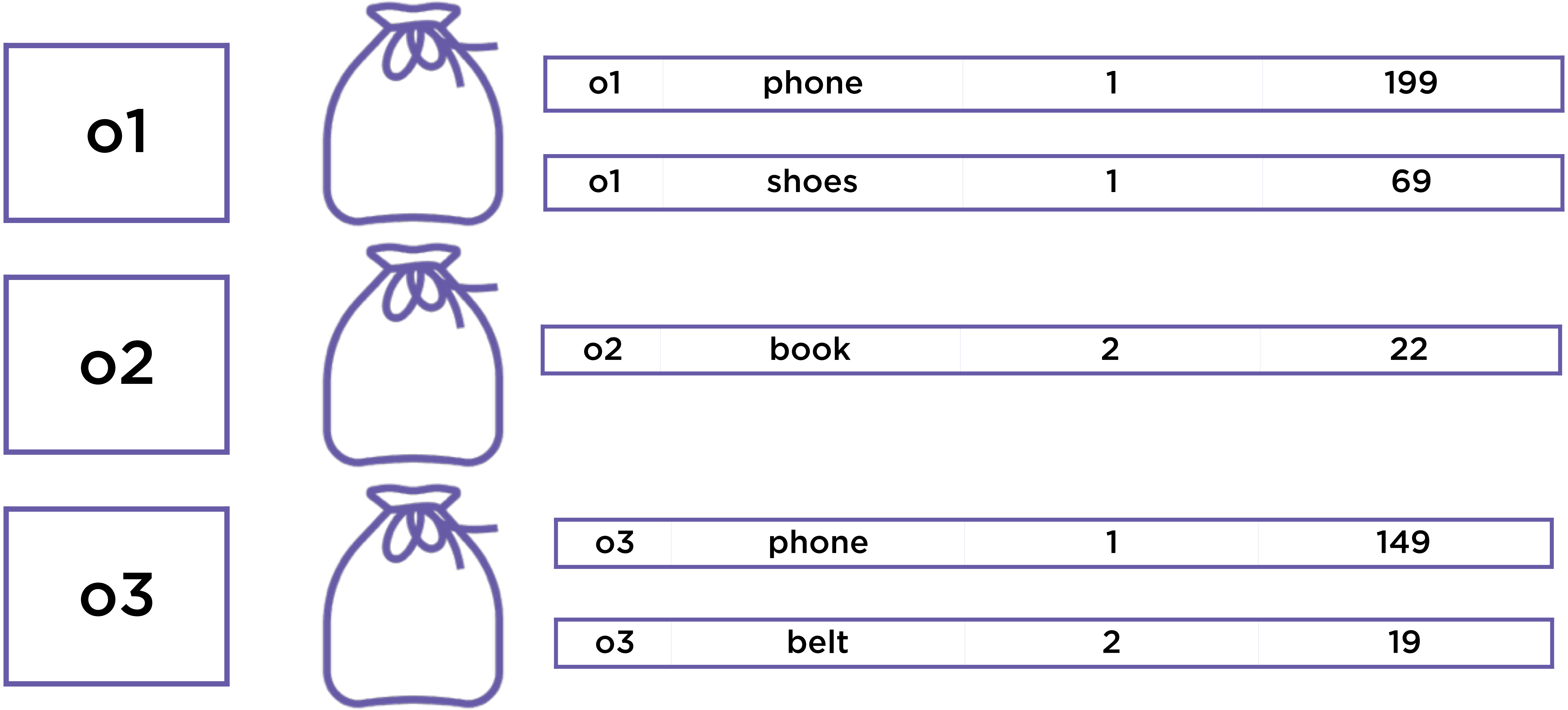
Tuple of fields

Group By

ID	Product_ID	Quantity	Amount
o1	phone	1	199
o1	shoes	1	69
o2	book	2	22
o3	phone	1	149
o3	belt	2	19

```
group orders by ID
```

Group By



o1

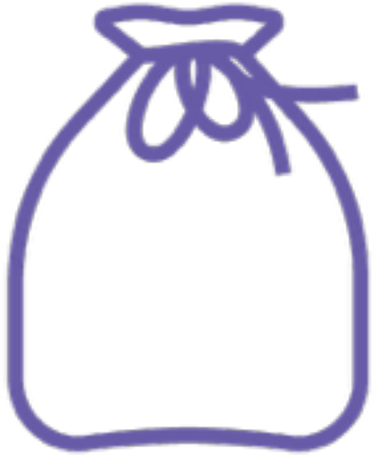
o2

o3

o1	phone	1	199
o1	shoes	1	69
o2	book	2	22
o3	phone	1	149
o3	belt	2	19

Group By

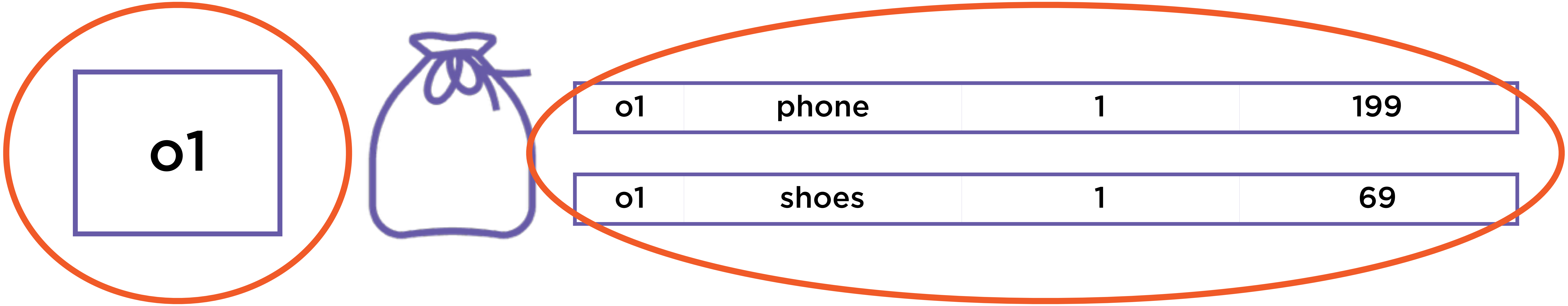
o1



o1	phone	1	199
o1	shoes	1	69

All records with the same key are grouped into a bag

Group By



o1

o1	phone	1	199
o1	shoes	1	69

`group orders by ID` creates a relation with 2 fields

key = field name “group”

value = bag with field name “orders”

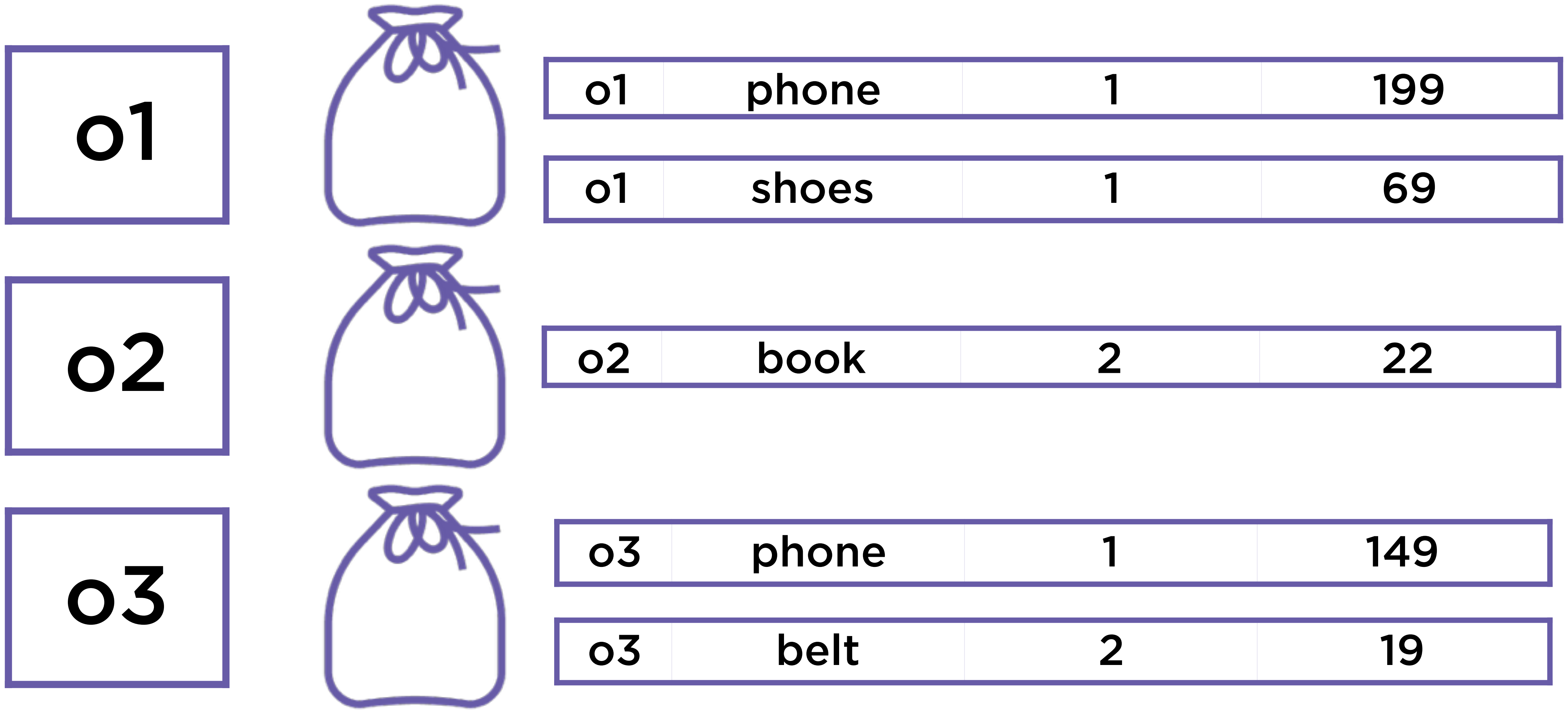
Demo

Use the group by command on the collisions data in preparation to performing aggregation operations

- group by reason for collisions across all boroughs
- group by collisions on a per borough basis

Performing Aggregations on Grouped Records

Aggregations on Groups



o1

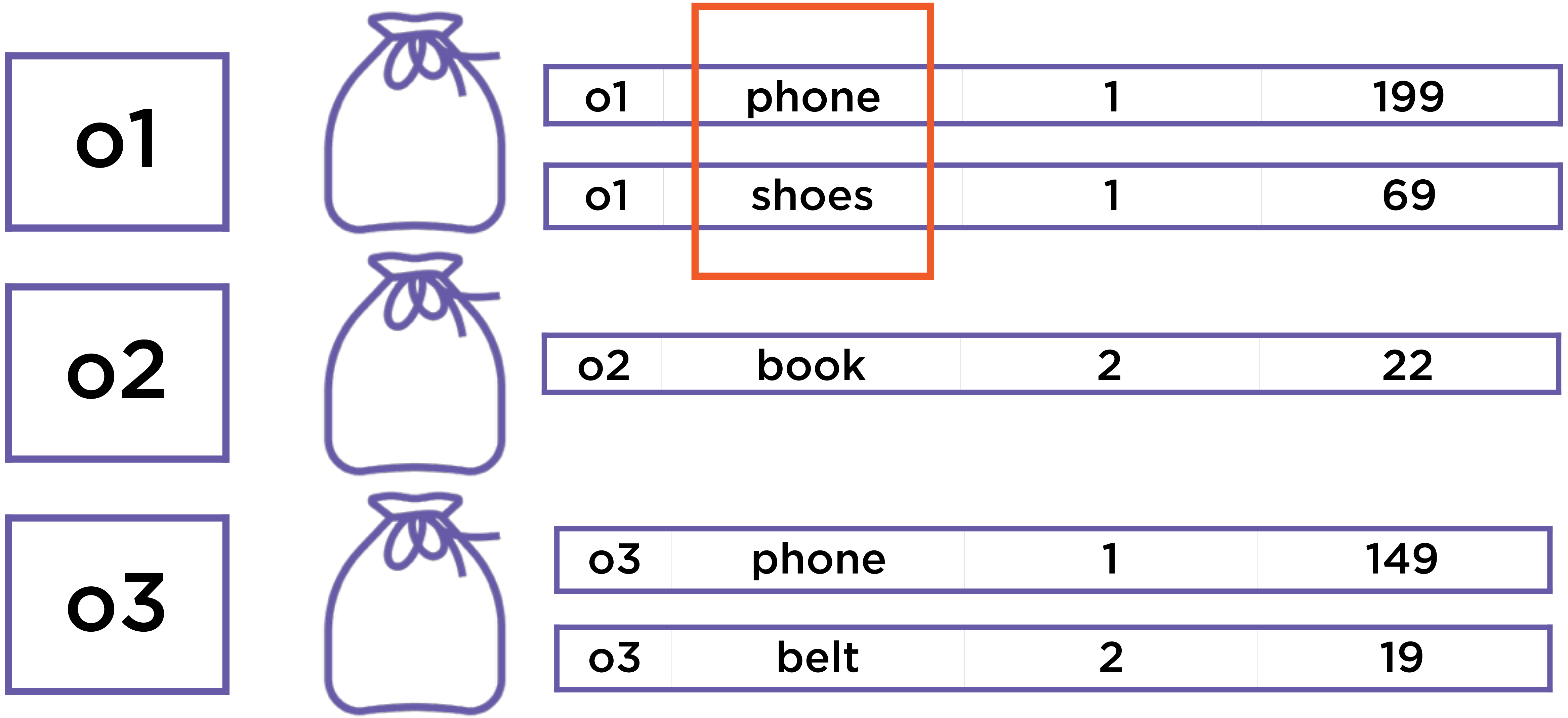
o2

o3

o1	phone	1	199
o1	shoes	1	69
o2	book	2	22
o3	phone	1	149
o3	belt	2	19

Aggregations are UDFs which can be applied to field values from multiple records

Aggregations on Groups



o1

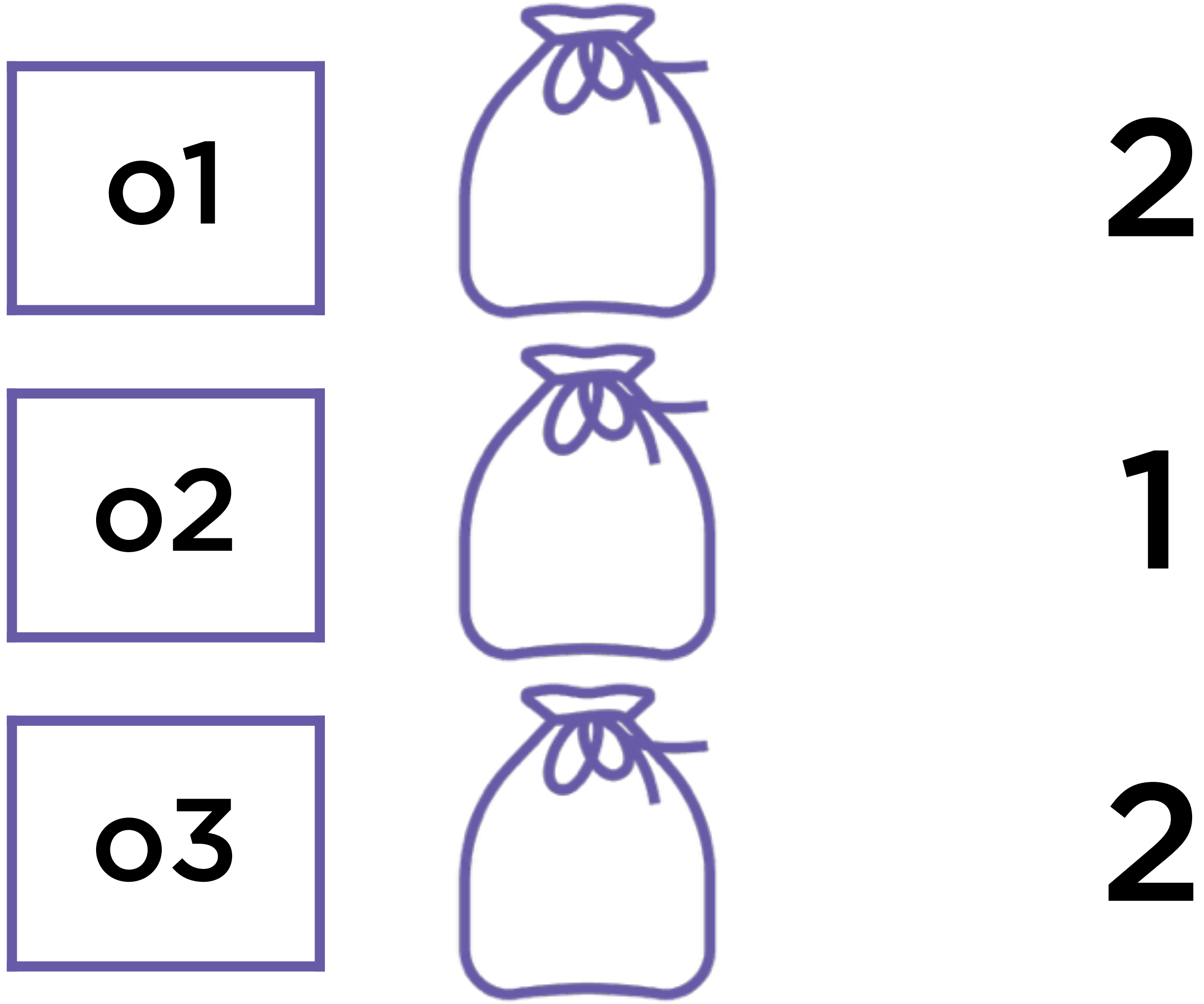
o2

o3

o1	phone	1	199
o1	shoes	1	69
o2	book	2	22
o3	phone	1	149
o3	belt	2	19

COUNT() the number of different products in each order

Aggregations on Groups



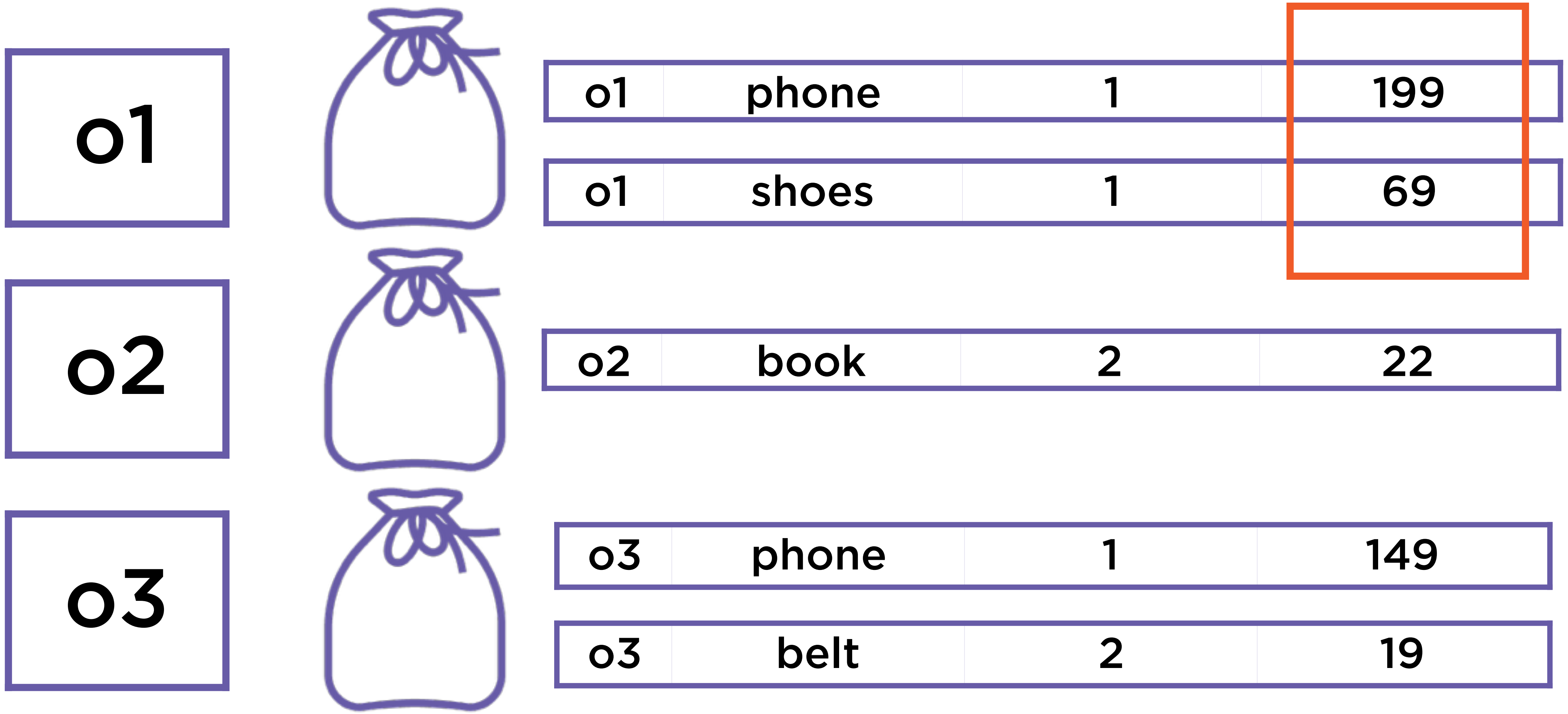
o1 2

o2 1

o3 2

COUNT() the number of different products in each order

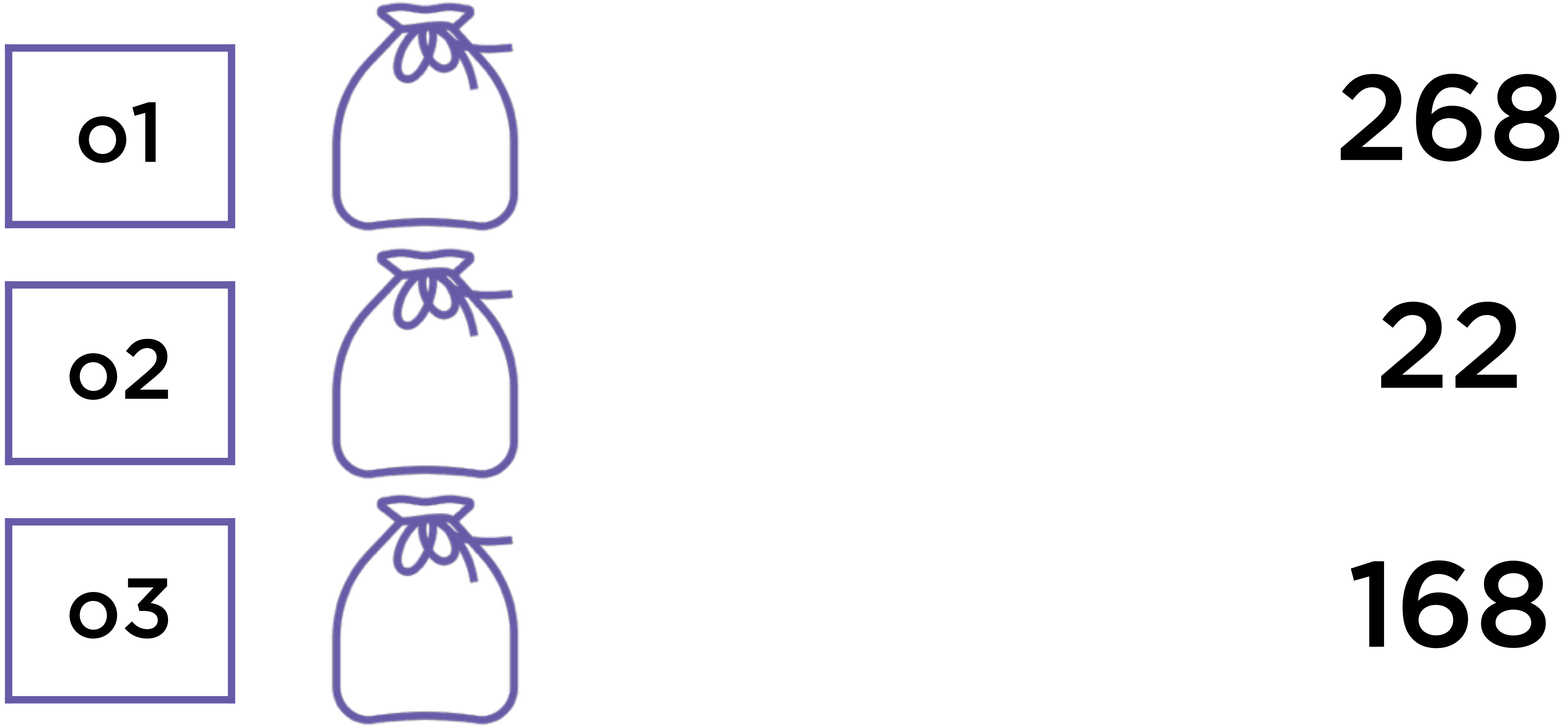
Aggregations on Groups



o1	o1	phone	1	199
	o1	shoes	1	69
o2	o2	book	2	22
o3	o3	phone	1	149
	o3	belt	2	19

SUM() the total amount spent per order

Aggregations on Groups



o1 268

o2 22

o3 168

SUM() the total amount spent per order

Demo

What kind of collision causes the most injuries in New York?

- use the SUM() aggregation

What boroughs have the most collisions?

- use the COUNT() aggregation

Join Operations in Pig

Joins

Name	Salary
Tom	1
John	1
Judy	150m

×

Name	Department
Judy	Google
Tom	GoogleX
John	Alphabet

Joins

Name	Salary	Department
Tom	1	GoogleX
John	1	Alphabet
Judy	150m	Google

Records from each relation matched on the join column

Joins

Name	Salary	Department
Tom	1	GoogleX
John	1	Alphabet
Judy	150m	Google

Pig provides support only for equi-joins

Demo

Perform join operations with 2 relations

Access individual fields from the joined relation using the :: operator

Types of Joins in Pig

Types of Joins

- Left Outer Join
- Right Outer Join
- Full Outer Join
- Self Join
- Cross Join

Types of Joins

Left Outer Join

Right Outer Join

Full Outer Join

Self Join

Cross Join

Left Outer Join

Name	Salary
Tom	1
John	1
Judy	150m

Name	Department
Emily	Google
John	GoogleX
Tom	Alphabet

Left Outer Join

Name	Salary
Tom	1
John	1
Judy	150m

Every record on the left table will be present in the result

- with a matching record
- padded with nulls

Left Outer Join

Name	Salary	Department
Tom	1	Alphabet
John	1	GoogleX
Judy	150m	NULL

Types of Joins

Left Outer Join

Right Outer Join

Full Outer Join

Self Join

Cross Join

Right Outer Join

Name	Salary
Tom	1
John	1
Judy	150m

Name	Department
Emily	Google
John	GoogleX
Tom	Alphabet

Right Outer Join

Every record on the right table will be present in the result

- with a matching record
- padded with nulls

Name	Department
Emily	Google
John	GoogleX
Tom	Alphabet

Right Outer Join

Name	Salary	Department
Emily	NULL	Google
John	1	GoogleX
Tom	1	Alphabet

Types of Joins

Left Outer Join

Right Outer Join

Full Outer Join

Self Join

Cross Join

Full Outer Join

Name	Salary
Tom	1
John	1
Judy	150m

Name	Department
Emily	Google
John	GoogleX
Tom	Alphabet

Full Outer Join

Name	Salary
Tom	1
John	1
Judy	150m

✕

Name	Department
Emily	Google
John	GoogleX
Tom	Alphabet

Records from both tables will be present in the result

- with a matching record
- padded with nulls

Full Outer Join

Name	Salary	Department
Emily	NULL	Google
John	1	GoogleX
Tom	1	Alphabet
Judy	150m	NULL

Types of Joins

Left Outer Join

Right Outer Join

Full Outer Join

Self Join

Cross Join

Self Join

Name	Salary
Tom	1
John	1
Judy	150m

×

Name	Salary
Tom	1
John	1
Judy	150m

Self Join

Name	Salary	Salary
Tom	1	1
John	1	1
Judy	150m	150m

Types of Joins

Left Outer Join

Right Outer Join

Full Outer Join

Self Join

Cross Join

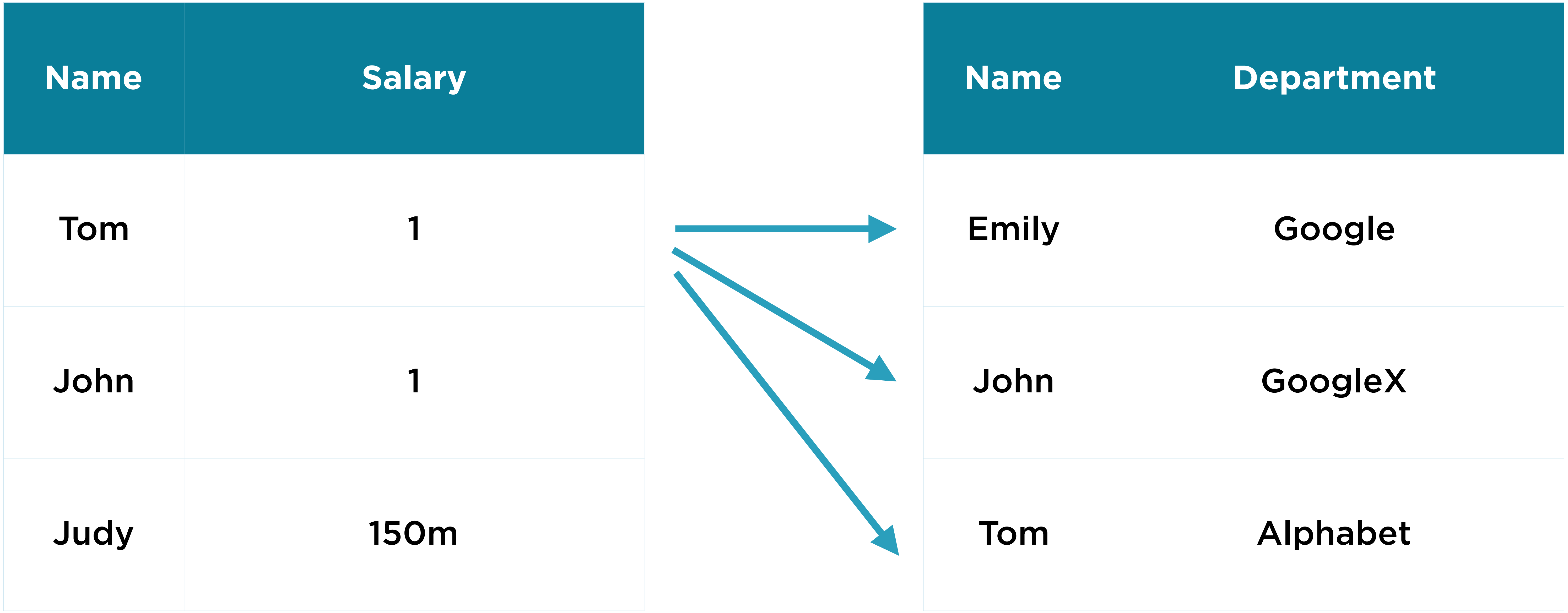
Cross Join

Name	Salary
Tom	1
John	1
Judy	150m

×

Name	Department
Emily	Google
John	GoogleX
Tom	Alphabet

Cross Join

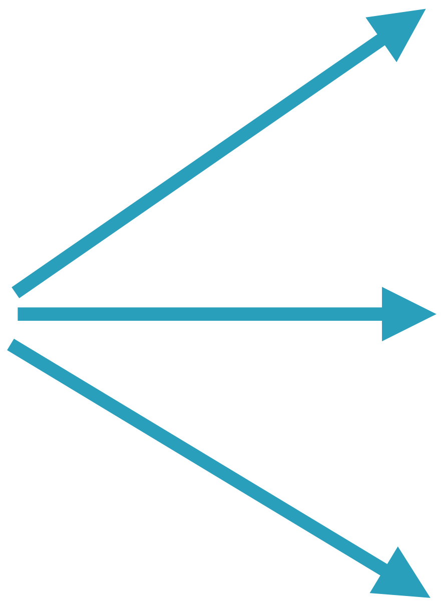


Name	Salary
Tom	1
John	1
Judy	150m

Name	Department
Emily	Google
John	GoogleX
Tom	Alphabet

Cross Join

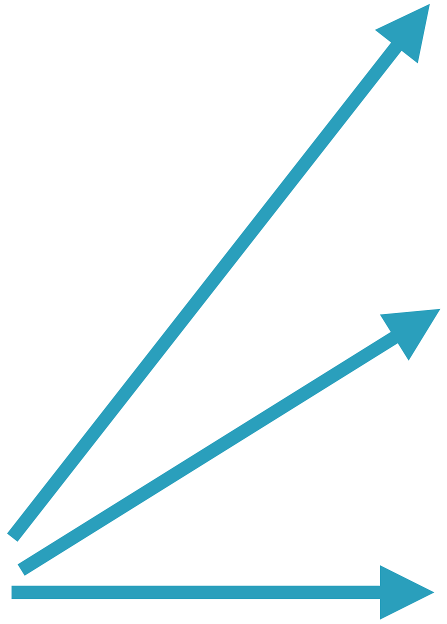
Name	Salary
Tom	1
John	1
Judy	150m



Name	Department
Emily	Google
John	GoogleX
Tom	Alphabet

Cross Join

Name	Salary
Tom	1
John	1
Judy	150m



Name	Department
Emily	Google
John	GoogleX
Tom	Alphabet

Cross Join

Name	Salary	Name	Department
Tom	1	Emily	Google
John	1	John	GoogleX
Judy	150m	Tom	Alphabet
Tom	1	Emily	Google
John	1	John	GoogleX
Judy	150m	Tom	Alphabet
Tom	1	Emily	Google
John	1	John	GoogleX
Judy	150m	Tom	Alphabet

Demo

Implement join operations in Pig

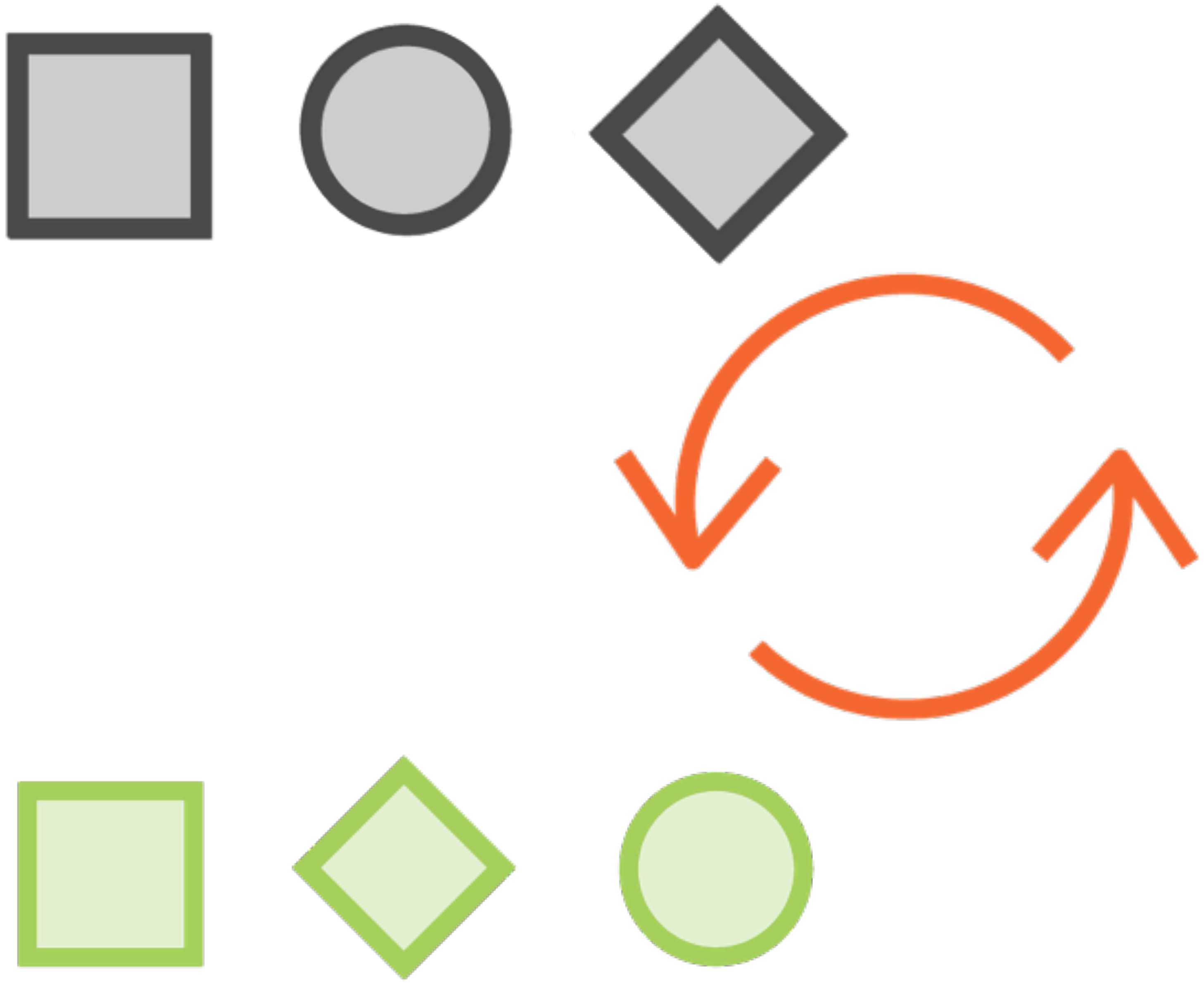
- left outer join

- self join

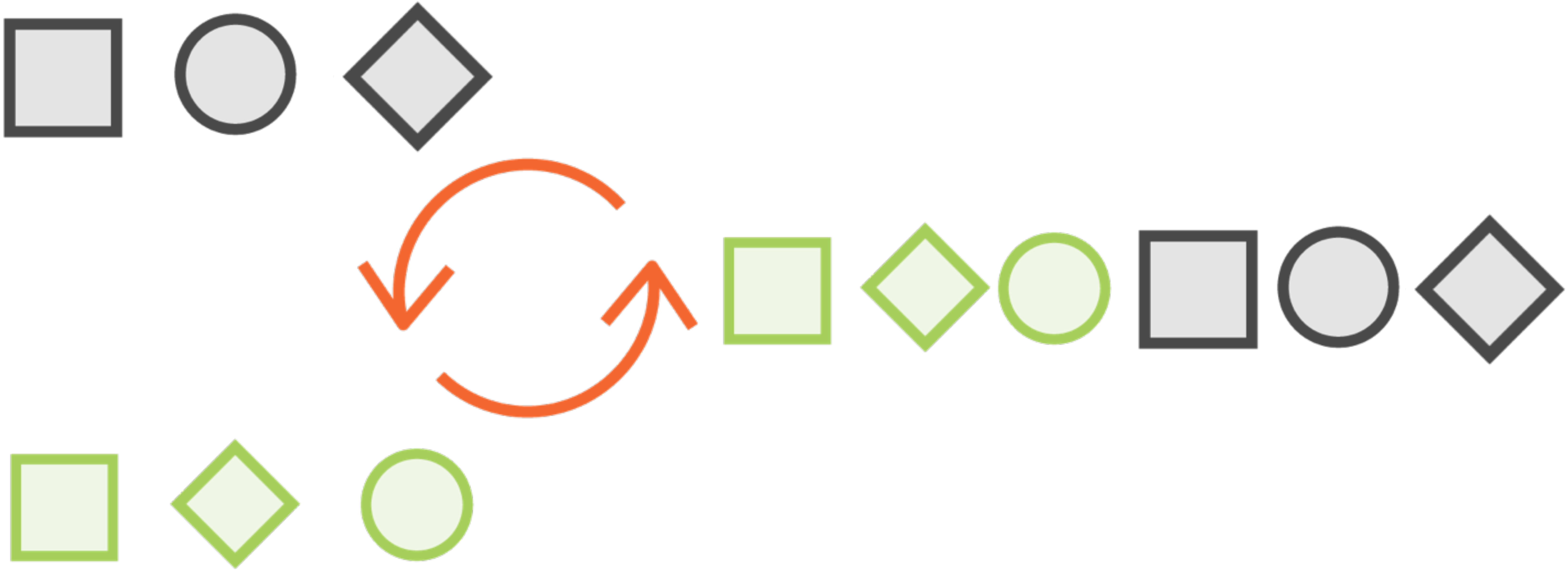
- cross join

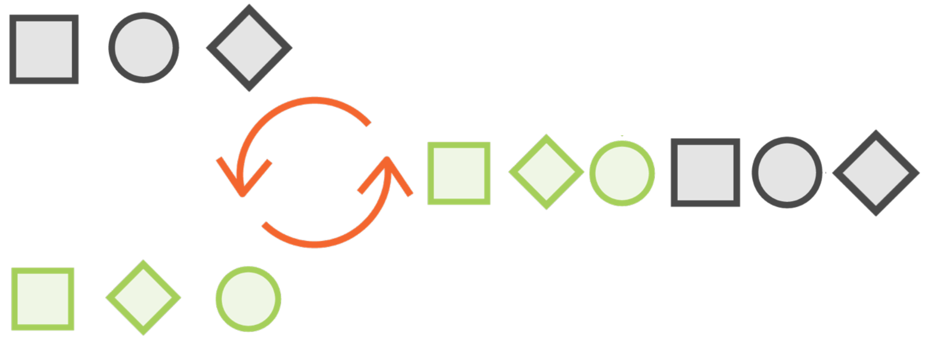
Unions in Pig

Union



Union





Union

The relations involved in a union should have:

- the same number of fields
- compatible schema

Does not preserve the order of tuples

Preserves duplicates

Demo

Implement a union between 2 relations which have the same schema

Unions with Different Schemas

```
R1: (a1: long, a2: long)
R2: (b1: long, b2: long, b3: long)

R1 union R2: null
```

Union When Schema Is Mismatched

```
R1: (a1: long, a2: long)
R2: (b1: long, b2: long, b3: long)

R1 union R2: null
```

Union When Schema Is Mismatched

```
R1: (a1: long, a2: long)
R2: (b1: long, b2: long, b3: long)

R1 union R2: null
```

Union When Schema Is Mismatched

```
R1: (a1: long, a2: long)
R2: (b1: long, b2: long, b3: long)

R1 union R2: null
```

Union When Schema Is Mismatched

```
R1: (a1: long, a2: long)
R2: (b1: (x: int, y: int), b2: long)

R1 union R2: (a1: bytearray, a2: long)
```

Union When Schema Types Are Not the Same

```
R1: (a1: long, a2: long)
R2: (b1: (x: int, y: int), b2: long)

R1 union R2: (a1: bytearray, a2: long)
```

Union When Schema Types Are Not the Same

```
R1: (a1: long, a2: long)
R2: (b1: (x: int, y: int), b2: long)

R1 union R2: (a1: bytearray, a2: long)
```

Union When Schema Types Are Not the Same

```
R1: (a1: long, a2: bytearray, a3: int)
R2: (b1: float, b2: chararray, b3: bytearray)

R1 union R2: (a1: float, a2: chararray, a3: int)
```

Compatible Types Will Be Cast to Higher Type

```
R1: (a1: long, a2: bytearray, a3: int)
R2: (b1: float, b2: chararray, b3: bytearray)

R1 union R2: (a1: float, a2: chararray, a3: int)
```

Compatible Types Will Be Cast to Higher Type

double > float > long > int > bytearray

```
R1: (a1: long, a2: bytearray, a3: int)
R2: (b1: float, b2: chararray, b3: bytearray)

R1 union R2: (a1: float, a2: chararray, a3: int)
```

Compatible Types Will Be Cast to Higher Type

double > float > long > int > bytearray

tuple | bag | map | chararray > bytearray

```
R1: (a1: long, a2: bytearray, a3: int)
R2: (b1: float, b2: chararray, b3: bytearray)

R1 union R2: (a1: float, a2: chararray, a3: int)
```

Compatible Types Will Be Cast to Higher Type

double > float > long > int > bytearray

tuple | bag | map | chararray > bytearray

```
R1: (a1: long, a2: bytearray, a3: int)
R2: (b1: float, b2: chararray, b3: bytearray)

R1 union R2: (a1: float, a2: chararray, a3: int)
```

Compatible Types Will Be Cast to Higher Type

double > float > long > int > bytearray

tuple | bag | map | chararray > bytearray

```
R1: (a1:(x:long, y:int), a2:{(n:float, m:chararray)})
R2: (b1:(g:chararray, h:float), b3:{(n:int, m:long)})

R1 union R2: (a1: (), a2: {()})
```

Different Inner Types

The union may result in an empty complex type

```
R1: (a1:(x:long, y:int), a2:{(n:float, m:chararray)})
R2: (b1:(g:chararray, h:float), b3:{(n:int, m:long)})

R1 union R2: (a1: (), a2: {()})
```

Different Inner Types

The union may result in an empty complex type

```
R1: (a1:(x:long, y:int), a2:{(n:float, m:chararray)})
R2: (b1:(g:chararray, h:float), b3:{(n:int, m:long)})

R1 union R2: (a1: (), a2: {()})
```

Different Inner Types

The union may result in an empty complex type

Union Onschema for Schema Mismatches

```
R1: (a1: long, a2: chararray)
R2: (b1: long, b2: float, b3: bytearray)

union onschema R1, R2

U: (a1: long, a2: chararray, b2: float, b3: bytearray)
```

Union Onschema Combines Schemas

```
R1: (a1: long, a2: chararray)
R2: (b1: long, b2: float, b3: bytearray)

union onschema R1, R2

U: (a1: long, a2: chararray, b2: float, b3: bytearray)
```

Union Onschema Combines Schemas

```
R1: (a1: long, a2: chararray)
R2: (b1: long, b2: float, b3: bytearray)

union onschema R1, R2

U: (a1: long, a2: chararray, b2: float, b3: bytearray)
```

Union Onschema Combines Schemas

```
R1: (a1: long, a2: chararray)
R2: (b1: long, b2: float, b3: bytearray)

union onschema R1, R2

U: (a1: long, a2: chararray, b2: float, b3: bytearray)
```

Union Onschema Combines Schemas

Demo

Implement union onschema between 2 relations which have only a few columns with matching schema

The Flatten Function

Flatten

User_ID	Username	Products_Bought
u123	John	{(phone), (book), (shoes), (shirt)}
u876	Jill	{(speakers)}
u654	Nina	{(handbag), (book)}

The flatten function is applied to a bag of tuples

Flatten

User_ID	Username	Products_Bought
u123		ook), (shoes), irt)}
u876		akers)}
u654	Nina	{(handbag), (book)}

{(handbag), (book)}

The products each user has bought is specified as a bag

Flatten

User_ID	Username	Products_Bought
u123	John	{(phone), (book), (shoes), (shirt)}
u876	Jill	{(speakers)}
u654	Nina	{(handbag), (book)}

Flattening a bag makes entity in the bag a separate record

Flatten

User_ID	Username	Products
u123	John	phone
u123	John	book
u123	John	shoes
u123	John	shirt
u876	Jill	speakers
u654	Nina	handbag
u654	Nina	book

Flatten

User_ID	Username	Products
u123	John	phone
u123	John	book
u123	John	shoes
u123	John	shirt
u876	Jill	speakers
u654	Nina	handbag
u654	Nina	book

Flatten

User_ID	Username	Products
u123	John	phone
u123	John	book
u123	John	shoes
u123	John	shirt
u876	Jill	speakers
u654	Nina	handbag
u654	Nina	book

Flattening an empty bag results in **null**

Demo

Use the flatten function with a bag of tuples

Summary

Used advanced Pig transformations such as:

- group by and aggregations
- join operations
- union operations
- flatten command

Analyzed real world data from the City of New York