```
data_set_2 = pig latin command ON data_set_1;

data_set_3 = pig latin command ON data_set_2;

data_set_4 = pig latin command ON data_set_3;

data_set_5 = pig latin command ON data_set_4;
```

THESE STEPS ARE VARIOUS DATA TRANSFORMATIONS

data_Set_2 = pig latin command ON data_Set_1;

data_Set_3 = pig latin command ON data_Set_2;

THESE STEPS ARE VARIOUS DATA TRANSFORMATIONS

data_Set_4 = pig latin command ON data_Set_3;

data_Set_5 = pig latin command ON data_Set_4;

LET'S TALK ABOUT THESE

BEFORE YOU DUMP DATA INTO A FILE OR ONTO SCREEN

1. SELECTING SPECIFIC FIELDS FROM THE RELATION

2. APPLYING SPECIFIC FUNCTIONS TO A FIELD

3. SELECTING A SPECIFIC NUMBER OF RECORDS, OR A SPECIFIC NUMBER OF DISTINCT VALUES

4. ORDERING RECORDS BASED ON SOME SPECIFIC COLUMN

5. FILTERING RECORDS BASED ON A CONDITION

6. GROUPING/AGGREGATING DATA BASED ON SPECIFIC COLUMNS

7. JOINING DATA OF ONE RELATION WITH ANOTHER RELATION

8. TRANSFORMING NESTED DATA IN A COLUMN TO MULTIPLE ROWS

9. SAMPLING RECORDS

1. SELECTING SPECIFIC FIELDS FROM THE RELATION

2. APPLYING SPECIFIC FUNCTIONS TO A FIELD

3. SELECTING A SPECIFIC NUMBER OF RECORDS, OR A SPECIFIC NUMBER OF DISTINCT VALUES

4. ORDERING RECORDS BASED ON SOME SPECIFIC COLUMN

5. FILTERING RECORDS BASED ON A CONDITION

6. GROUPING/AGGREGATING DATA BASED ON SPECIFIC COLUMNS

7. JOINING DATA OF ONE RELATION WITH ANOTHER RELATION

8. TRANSFORMING NESTED DATA IN A COLUMN TO MULTIPLE ROWS

9. SAMPLING RECORDS

1. SELECTING SPECIFIC FIELDS FROM THE RELATION

THIS IS DONE USING FOREACH AND GENERATE

1. SELECTING SPECIFIC FIELDS FROM THE RELATION

THIS IS DONE USING FOREACH

AND GENERATE

LET US ASSUME WE HAVE LOADED THIS TABLE INTO THE RELATION ORDERS WITH SCHEMA

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	01-Jan-15	1	Apple iPhone 4S
OD01	1	1	450	01-Jan-15	2	Apple iPhone 5S
OD01	1	1	450	01-Jan-15	3	Apple iPhone 6
OD02	2	2	400	02-Jan-15	1	Apple iPhone 4S
OD02	2	2	400	02-Jan-15	2	Apple iPhone 5S
OD02	2	2	400	02-Jan-15	3	Apple iPhone 6

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

THIS IS DONE USING FOREACH AND GENERATE

```
grunt> orders = load 'order_data' using PigStorage(',') as
(Order_ID:chararray, Customer_ID:int,
Order_Quantity:int,Order_Value:float,Order_Date:chararray,
Product_ID:int, Product_Name:chararray);
```

```
grunt> orders_customers = foreach orders generate
Order_ID,Customer_ID;
```

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

THIS IS DONE USING FOREACH AND GENERATE

```
grunt> orders_customers = foreach orders generate
Order_ID,Customer_ID;

grunt>dump orders_customers
```

GIVES

ORDER_ID	CUSTOMER_ID
OD01	1
OD01	1
OD01	1
OD02	2
OD02	2
OD02	2

```
grunt> orders_customers = foreach orders generate
Order_ID,Customer_ID;
```

```
grunt>dump orders_customers
```

GIVES

```
(OD01,1)
(OD01,1)
(OD01,1)
(OD02,2)
(OD02,2)
(OD02,2)
(OD03,2)
(OD03,2)
(OD03,2)
(OD04,3)
(OD04,3)
(OD04,3)
(OD05,4)
(OD05,4)
(OD05,4)
```

ORDER_ID	CUSTOMER_ID
OD01	1
OD01	1
OD01	1
OD02	2
OD02	2
OD02	2

THIS IS THE OUTPUT WE WILL SEE

```
grunt> orders_customers = foreach orders generate
Order_ID,Customer_ID;
```

THIS IS SIMILAR TO THE SELECT FROM COMMAND OF SQL

```
grunt> orders_customers = foreach orders generate
Order_ID,Customer_ID;
```

IF WE HAVEN'T SPECIFIED A SCHEMA FOR THE ORDERS TABLE WE WOULD USE THE $ NOTATION TO SELECT COLUMNS

```
grunt> orders_customers = foreach orders
generate $0,$1;
```

HOW DO WE CHOOSE FIELDS FROM COMPLEX DATA TYPES?

EXAMPLE TUPLE

(Order_ID: chararray,
Customer_ID: int)

(OD01,
1)

THE TUPLE WAS LOADED USING

```
order_customer = load 'order_customer' as
(order_customer_id:
tuple(Order_ID:chararray,Customer_ID:int));
```

TUPLE

THE TUPLE WAS LOADED USING

```
order_customer = load 'order_customer' as
(order_customer_id:
tuple(Order_ID:chararray,Customer_ID:int));
```

FIELDS IN A TUPLE CAN BE ACCESSED USING THE NAME OF THE FIELD AND "." OPERATOR

```
grunt> orders_only=foreach order_customer
generate order_customer_id.Order_ID;
```

GENERATES ORDER_ID ONLY

TUPLE

THE TUPLE WAS LOADED USING

```
order_customer = load 'order_customer' as
(order_customer_id:
tuple(Order_ID:chararray,Customer_ID:int));
```

FIELDS IN A TUPLE CAN BE ACCESSED USING THE NAME OF THE FIELD AND "." OPERATOR

```
grunt> orders_only=foreach order_customer
generate order_customer_id.$0;
```

GENERATES ORDER_ID ONLY

MAP

LET US ASSUME WE LOAD DATA FROM A FILE CALLED MAP_EXAMPLE.TXT

THE DATA LOOKS LIKE THIS

```
[Customer_Id#'101',Customer_Name#'Hari']
[Customer_Id#'102',Customer_Name#'Niki']
```

[KEY# VALUE]

WE LOAD THIS FILE USING

```
map_example = load 'map_example.txt' as
(Customer_Fields:MAP[chararray]);
```

MAP

WE LOAD THIS FILE USING

```
map_example = load 'map_example.txt' as
(Customer_Fields:MAP[chararray]);
```

IF WE DUMP map_example.txt

```
([Customer_Name#'Hari',Customer_Id#'101'])
([Customer_Name#'Niki',Customer_Id#'102'])
```

MAP

WE LOAD THIS FILE USING

```
map_example = load 'map_example.txt' as
(Customer_Fields:MAP[chararray]);
```

```
([Customer_Name#'Hari',Customer_Id#'101'])
([Customer_Name#'Niki',Customer_Id#'102'])
```

HOW TO GET 'VALUES' IN MAP USING 'KEY'?

MAP_NAME#'KEY'

```
grunt> customer_name = foreach map_example generate
Customer_Fields#'Customer_Name';
```

MAP

```
([Customer_Name#'Hari',Customer_Id#'101'])
([Customer_Name#'Niki',Customer_Id#'102'])
```

HOW TO GET 'VALUES' IN MAP USING 'KEY'?

MAP_NAME#'KEY'

```
grunt> customer_name = foreach map_example generate
Customer_Fields#'Customer_Name';
```

IF WE DUMP customer_name

```
('Hari')
('Niki')
```

BAG

CONSIDER THE TUPLE

(Order_ID: chararray,
Customer_ID: int,
Order_Quantity: int,
Order_Value: float,
Order_Date: chararray,
Product_ID: int,
Product_Name: chararray)

(OD01,
1,
1,
450.0
,2016-01-24
,1
,Apple iPhone 4S)

WE CREATE A BAG OF THESE TUPLES

```
order_bag = load 'order_bag' as
(order_customer_id: bag{tuple_name:
```

(Order_ID: chararray,
Customer_ID: int,
Order_Quantity: int,
Order_Value: float,
Order_Date: chararray,
Product_ID: int,
Product_Name: chararray)

TUPLE_SCHEMA

```
});
```

THE order_bag IN FILE LOOKS LIKE THIS

```
{(OD01),(1),(1),(450.0),(2016-01-24),(1),(Apple iPhone 4S)}
{(OD01),(1),(1),(450.0),(2016-01-24),(2),(Apple iPhone 5S)}
{(OD01),(1),(1),(450.0),(2016-01-24),(3),(Apple iPhone 6)}
{(OD02),(2),(2),(400.0),(2016-01-24),(1),(Apple iPhone 4S)}
{(OD02),(2),(2),(400.0),(2016-01-24),(2),(Apple iPhone 5S)}
{(OD02),(2),(2),(400.0),(2016-01-24),(3),(Apple iPhone 6)}
{(OD03),(2),(1),(350.0),(2016-01-25),(1),(Apple iPhone 4S)}
{(OD03),(2),(1),(350.0),(2016-01-25),(2),(Apple iPhone 5S)}
{(OD03),(2),(1),(350.0),(2016-01-25),(3),(Apple iPhone 6)}
{(OD03),(2),(1),(350.0),(2016-01-25),(3),(Apple iPhone 6)}
{(OD04),(3),(1),(500.0),(2016-01-25),(1),(Apple iPhone 4S)}
{(OD04),(3),(1),(500.0),(2016-01-25),(2),(Apple iPhone 5S)}
{(OD04),(3),(1),(500.0),(2016-01-27),(3),(Apple iPhone 6)}
{(OD05),(4),(1),(650.0),(2016-01-26),(1),(Apple iPhone 4S)}
{(OD05),(4),(1),(650.0),(2016-01-27),(2),(Apple iPhone 5S)}
{(OD05),(4),(1),(650.0),(2016-01-27),(3),(Apple iPhone 6)}
```

OUTPUT OF order_bag IN PIG

```
({(OD01),(1),(1),(450),(2016-01-24),(1),(Apple iPhone 4S)})
({(OD01),(1),(1),(450),(2016-01-24),(2),(Apple iPhone 5S)})
({(OD01),(1),(1),(450),(2016-01-24),(3),(Apple iPhone 6)})
({(OD02),(2),(2),(400),(2016-01-24),(1),(Apple iPhone 4S)})
({(OD02),(2),(2),(400),(2016-01-24),(2),(Apple iPhone 5S)})
({(OD02),(2),(2),(400),(2016-01-24),(3),(Apple iPhone 6)})
({(OD03),(2),(1),(350),(2016-01-25),(1),(Apple iPhone 4S)})
({(OD03),(2),(1),(350),(2016-01-25),(2),(Apple iPhone 5S)})
({(OD03),(2),(1),(350),(2016-01-25),(3),(Apple iPhone 6)})
({(OD03),(2),(1),(350),(2016-01-25),(3),(Apple iPhone 6)})
({(OD04),(3),(1),(500),(2016-01-25),(1),(Apple iPhone 4S)})
({(OD04),(3),(1),(500),(2016-01-25),(2),(Apple iPhone 5S)})
({(OD04),(3),(1),(500),(2016-01-27),(3),(Apple iPhone 6)})
({(OD05),(4),(1),(650),(2016-01-26),(1),(Apple iPhone 4S)})
({(OD05),(4),(1),(650),(2016-01-27),(2),(Apple iPhone 5S)})
({(OD05),(4),(1),(650),(2016-01-27),(3),(Apple iPhone 6)})
```

BAG

CONSIDER THE TUPLE

```
(Order_ID: chararray,
Customer_ID: int,
Order_Quantity: int,
Order_Value: float,
Order_Date: chararray,
Product_ID: int,
Product_Name: chararray)
```

```
(OD01,
1,
1,
450.0
,2016-01-24
,1
,Apple iPhone 4S)
```

WE CREATE A BAG OF THESE TUPLES

SINCE A BAG IS AN UNORDERED COLLECTION OF TUPLES, WE CAN'T INDEX AND RETRIEVE SPECIFIC TUPLES

BUT WE CAN CREATE/PROJECT A BAG OF FIELDS CONTAINED IN THE TUPLE

BAG

```
order_bag = load 'order_bag' as
(order_customer_id: bag{tuple_name:
    (Order_ID: chararray,
     Customer_ID: int,
     Order_Quantity: int,
     Order_Value: float,
     Order_Date: chararray,
     Product_ID: int,
     Product_Name: chararray)
});
```

TUPLE_SCHEMA

BUT WE CAN CREATE/PROJECT A BAG OF THE FIELDS CONTAINED IN THE TUPLE

```
grunt> orders_product_tuple = foreach order_bag
generate order_customer_id.(Order_ID,Product_Name);
```

BUT WE CAN CREATE/PROJECT A BAG OF THE FIELDS CONTAINED IN THE TUPLE

```
grunt> grunt> orders_product_tuple = foreach order_bag
generate order_customer_id.(Order_ID,Product_Name);
```

IT IS A BAG OF (Order_ID, Product_Name) TUPLES

```
({(OD01),(Apple iPhone 4S)})
({(OD01),(Apple iPhone 5S)})
({(OD01),(Apple iPhone 6)})
({(OD02),(Apple iPhone 4S)})
({(OD02),(Apple iPhone 5S)})
({(OD02),(Apple iPhone 6)})
({(OD03),(Apple iPhone 4S)})
({(OD03),(Apple iPhone 5S)})
({(OD03),(Apple iPhone 6)})
({(OD03),(Apple iPhone 6)})
({(OD04),(Apple iPhone 4S)})
({(OD04),(Apple iPhone 5S)})
({(OD04),(Apple iPhone 6)})
({(OD05),(Apple iPhone 4S)})
({(OD05),(Apple iPhone 5S)})
({(OD05),(Apple iPhone 6)})
```

1. SELECTING SPECIFIC FIELDS FROM THE RELATION

2. APPLYING SPECIFIC FUNCTIONS TO A FIELD

3. SELECTING A SPECIFIC NUMBER OF RECORDS, OR A SPECIFIC NUMBER OF DISTINCT VALUES

4. ORDERING RECORDS BASED ON SOME SPECIFIC COLUMN

5. FILTERING RECORDS BASED ON A CONDITION

6. GROUPING/AGGREGATING DATA BASED ON SPECIFIC COLUMNS

7. JOINING DATA OF ONE RELATION WITH ANOTHER RELATION

8. TRANSFORMING NESTED DATA IN A COLUMN TO MULTIPLE ROWS

9. SAMPLING RECORDS

1. SELECTING SPECIFIC FIELDS FROM THE RELATION

2. APPLYING SPECIFIC FUNCTIONS TO A FIELD

3. SELECTING A SPECIFIC NUMBER OF RECORDS, OR A SPECIFIC NUMBER OF DISTINCT VALUES

4. ORDERING RECORDS BASED ON SOME SPECIFIC COLUMN

5. FILTERING RECORDS BASED ON A CONDITION

6. GROUPING/AGGREGATING DATA BASED ON SPECIFIC COLUMNS

7. JOINING DATA OF ONE RELATION WITH ANOTHER RELATION

8. TRANSFORMING NESTED DATA IN A COLUMN TO MULTIPLE ROWS

9. SAMPLING RECORDS

2. APPLYING SPECIFIC FUNCTIONS TO A FIELD

FOREACH CAN APPLY EXPRESSIONS OR FORMULAS ALSO TO THE FIELDS

LET'S SAY WE WANT TO COMPUTE THE PRODUCT OF TWO FIELDS

LET'S SAY WE WANT TO COMPUTE THE PRODUCT OF TWO FIELDS

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

ORDER_QUANTITY * ORDER_VALUE

ORDER_QUANTITY * ORDER_VALUE

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

```
grunt> quantity_value_product = foreach orders
generate(Order_Quantity * Order_Value) as
quantity_value_product,Order_ID,Customer_ID;
```

THIS IS THE NAME OF THE NEW FIELD WHICH IS CALCULATED USING THESE TWO FIELDS

LET US ASSUME WE WANT TO COMPUTE THE PRODUCT OF TWO FIELDS

```
grunt> quantity_value_product = foreach orders
generate(Order_Quantity * Order_Value) as
quantity_value_product,Order_ID,Customer_ID;
```

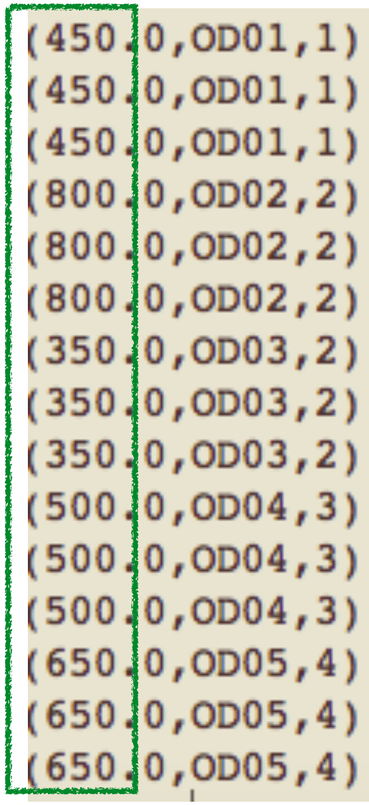
THIS IS THE NAME OF THE NEW FIELD WHICH IS CALCULATED USING THESE TWO FIELDS

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

THE OUTPUT OF THIS COMMAND IS

```
grunt> quantity_value_product = foreach orders
generate(Order_Quantity * Order_Value) as
quantity_value_product,Order_ID,Customer_ID;
```

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6



FOREACH CAN APPLY EXPRESSIONS OR FORMULAS TO FIELDS

2. APPLYING SPECIFIC FUNCTIONS TO A FIELD

FOREACH ALSO SUPPORTS UDFs

UDF = USER DEFINED FUNCTIONS

FOREACH ALSO SUPPORTS UDFs

UDF = USER DEFINED FUNCTIONS

PIG'S STRENGTH COMES FROM THE FACT THAT IT CAN BE EXTENDED USING USER-DEFINED FUNCTIONS

PIG'S STRENGTH COMES FROM THE FACT THAT IT CAN BE EXTENDED USING USER-DEFINED FUNCTIONS

UDFs CAN BE WRITTEN IN

JAVA
PYTHON
JPYTHON
RUBY
JAVASCRIPT

UDFs CAN BE WRITTEN IN

JAVA
PYTHON
JPYTHON
RUBY
JAVASCRIPT

APART FROM THE UDFS THAT YOU CAN WRITE THERE ARE MANY USEFUL BUILT-IN UDFs YOU CAN USE

APART FROM THE UDFS THAT YOU CAN WRITE THERE ARE MANY USEFUL BUILT-IN UDFs YOU CAN USE

PIG COMES PREPACKAGED WITH MANY UDFs THAT CAN BE DIRECTLY USED

PIG COMES PREPACKAGED WITH MANY UDFs
THAT CAN BE DIRECTLY USED

THERE ARE 4 TYPES OF BUILT-IN PIG FUNCTIONS

LOAD

STORE

EVALUATE

FILTER

LOAD THERE ARE 4 TYPES OF BUILT-IN PIG FUNCTIONS

WE HAVE ALREADY COVERED SOME OF THEM

Function	Location String indicates	Constructor arguments	Description
`HBaseStorage`	HBase table	The first argument is a string describing Pig field to HBase column family and column mapping. The second is an option string (optional).	Store data to HBase (see HBase).
`PigStorage`	HDFS file	The first argument is a field separator (optional; defaults to Tab).	Store text to HDFS in text format (see Store).

STORE LOAD

THERE ARE MANY STORE/LOAD FUNCTION AVAILABLE FOR VARIOUS TYPE OF FILE FORMATS

NAME	FILE FORMAT	LOAD/STORE
org.apache.pig.piggybank.storage.avro.AvroStorage() org.apache.pig.piggybank.storage.avro.AvroStorage('no_schema_check','$SCHEMA_FILE', '$PATH');	AVRO	LOAD,STORE
org.apache.pig.piggybank.storage.CSVExcelStorage() org.apache.pig.piggybank.storage.CSVExcelStorage('$DELIMITER', 'YES_MULTILINE', 'NOCHANGE', 'SKIP_INPUT_HEADER')	CSV	LOAD,STORE
org.apache.pig.piggybank.storage.JsonLoader() org.apache.pig.piggybank.storage.JsonLoader('$SCHEMA')	JSON	LOAD,STORE
com.mongodb.hadoop.pig.MongoLoader() com.mongodb.hadoop.pig.MongoLoader('$SCHEMA')	MongoDB L	LOAD,STORE

STORE LOAD

THEY'RE TYPICALLY THE SAME FUNCTIONS USED ALONG WITH THE LOAD AND STORE COMMANDS

NAME	FILE FORMAT	LOAD/STORE
org.apache.pig.piggybank.storage.avro.AvroStorage() org.apache.pig.piggybank.storage.avro.AvroStorage('no_schema_check','$SCHEMA_FILE', '$PATH');	AVRO	LOAD,STORE
org.apache.pig.piggybank.storage.CSVExcelStorage() org.apache.pig.piggybank.storage.CSVExcelStorage('$DELIMITER', 'YES_MULTILINE', 'NOCHANGE', 'SKIP_INPUT_HEADER')	CSV	LOAD,STORE
org.apache.pig.piggybank.storage.JsonLoader() org.apache.pig.piggybank.storage.JsonLoader('$SCHEMA')	JSON	LOAD,STORE
com.mongodb.hadoop.pig.MongoLoader() com.mongodb.hadoop.pig.MongoLoader('$SCHEMA')	MongoDB L	LOAD,STORE

PIG COMES PREPACKAGED WITH MANY UDFS THAT CAN BE DIRECTLY USED

THERE ARE 4 TYPES OF BUILT-IN PIG FUNCTIONS

LOAD

STORE

EVALUATE

FILTER

THERE ARE 4 TYPES OF BUILT-IN PIG FUNCTIONS

EVALUATE

FILTER

EVALUATION FUNCTIONS ARE THE MOST COMMON UDF FUNCTIONS

THERE ARE 4 TYPES OF BUILT-IN PIG FUNCTIONS

EVALUATE

FILTER

EVALUATION FUNCTIONS ARE THE MOST COMMON UDF FUNCTIONS

FILTER FUNCTIONS ARE USED WITH THE FILTER COMMAND

EVALUATION FUNCTIONS ARE THE MOST COMMON UDF FUNCTIONS

EVALUATION FUNCTIONS CAN BE CATEGORISED INTO THE FOLLOWING TYPES

EVALUATION FUNCTIONS CAN BE CATEGORISED INTO THE FOLLOWING TYPES

MATH FUNCTIONS

CHARARRAY FUNCTIONS

COMPLEX DATA TYPE FUNCTIONS

AGGREGATE FUNCTIONS

DATETIME FUNCTIONS

MATH FUNCTIONS

- Math Functions
  - ABS
  - ACOS
  - ASIN
  - ATAN
  - CBRT
  - CEIL
  - COS
  - COSH
  - EXP
  - FLOOR
  - LOG
  - LOG10
  - RANDOM
  - ROUND
  - SIN
  - SINH
  - SQRT
  - TAN
  - TANH

THESE FUNCTIONS ARE SIMILAR TO JAVA.LANG.MATH

ALL OF THEM ARE SUPPORTED IN PIG

MATH FUNCTIONS

- Math Functions
  - ABS
  - ACOS
  - ASIN
  - ATAN
  - CBRT
  - CEIL
  - COS
  - COSH
  - EXP
  - FLOOR
  - LOG
  - LOG10
  - RANDOM
  - ROUND
  - SIN
  - SINH
  - SQRT
  - TAN
  - TANH

THESE FUNCTIONS ARE SIMILAR TO JAVA.LANG.MATH

REMEMBER THAT THESE FUNCTION NAMES ARE UPPER CASE

MATH FUNCTIONS

HOW TO USE THESE FUNCTIONS?

- Math Functions
  - ABS
  - ACOS
  - ASIN
  - ATAN
  - CBRT
  - CEIL
  - COS
  - COSH
  - EXP
  - FLOOR
  - LOG
  - LOG10
  - RANDOM
  - ROUND
  - SIN
  - SINH
  - SQRT
  - TAN
  - TANH

```
quantity_value_product = foreach orders
generate(Order_Quantity * Order_Value) as
quantity_value_product,Order_ID,Customer_ID,
ROUND(Order_Value) as Trimmed_Value;
```

MATH FUNCTIONS

```
quantity_value_product = foreach orders
generate(Order_Quantity * Order_Value) as
quantity_value_product,Order_ID,Customer_ID,
ROUND(Order_Value) as Trimmed_Value;
```

- Math Functions
  - ABS
  - ACOS
  - ASIN
  - ATAN
  - CBRT
  - CEIL
  - COS
  - COSH
  - EXP
  - FLOOR
  - LOG
  - LOG10
  - RANDOM
  - ROUND
  - SIN
  - SINH
  - SQRT
  - TAN
  - TANH

THE OUTPUT IS

ROUNDING THE ORDER VALUE

```
(450.0,OD01,1,450)
(450.0,OD01,1,450)
(450.0,OD01,1,450)
(800.0,OD02,2,400)
(800.0,OD02,2,400)
(800.0,OD02,2,400)
(350.0,OD03,2,350)
(350.0,OD03,2,350)
(350.0,OD03,2,350)
(500.0,OD04,3,500)
(500.0,OD04,3,500)
(500.0,OD04,3,500)
(650.0,OD05,4,650)
(650.0,OD05,4,650)
(650.0,OD05,4,650)
```

EVALUATION FUNCTIONS CAN BE CATEGORISED INTO FOLLOWING TYPES

MATH FUNCTIONS

CHARARRAY FUNCTIONS

COMPLEX DATA TYPE FUNCTIONS

AGGREGATE FUNCTIONS

DATETIME FUNCTIONS

CHARARRAY FUNCTIONS

THESE FUNCTIONS PROVIDE WAYS TO MANIPULATE TEXT FIELDS

String Functions

- INDEXOF
- LAST_INDEX_OF
- LCFIRST
- LOWER
- REGEX_EXTRACT
- REGEX_EXTRACT_ALL
- REPLACE
- STRSPLIT
- SUBSTRING
- TRIM
- UCFIRST
- UPPER

THESE ARE THE THE CHARARRAY FUNCTIONS SUPPORTED IN PIG

CHARARRAY FUNCTIONS

THESE FUNCTIONS PROVIDE WAYS TO MANIPULATE TEXT FIELDS

String Functions

- INDEXOF
- LAST_INDEX_OF
- LCFIRST
- LOWER
- REGEX_EXTRACT
- REGEX_EXTRACT_ALL
- REPLACE
- STRSPLIT
- SUBSTRING
- TRIM
- UCFIRST
- UPPER

LET US USE ONE OF THEM

CHARARRAY FUNCTIONS

LET US USE ONE OF THEM

String Functions

- INDEXOF
- LAST_INDEX_OF
- LCFIRST
- LOWER
- REGEX_EXTRACT
- REGEX_EXTRACT_ALL
- REPLACE
- STRSPLIT
- SUBSTRING
- TRIM
- UCFIRST
- UPPER

```
grunt>quantity_value_product = foreach orders
generate(Order_Quantity * Order_Value) as
quantity_value_product,Order_ID,Customer_ID,
SUBSTRING(Order_ID,2,3) as Order_Number;
```

CHARARRAY FUNCTIONS

String Functions

- INDEXOF
- LAST_INDEX_OF
- LCFIRST
- LOWER
- REGEX_EXTRACT
- REGEX_EXTRACT_ALL
- REPLACE
- STRSPLIT
- SUBSTRING
- TRIM
- UCFIRST
- UPPER

```
grunt>quantity_value_product = foreach orders
generate(Order_Quantity * Order_Value) as
quantity_value_product,Order_ID,Customer_ID,
SUBSTRING(Order_ID,2,3) as Order_Number;
```

THE OUTPUT IS

TRIMMING THE 'OD' PART

GET THE SUBSTRING WHICH STARTS AT INDEX 2 AND ENDS AT INDEX 3

```
(450.0,OD01,1,01)
(450.0,OD01,1,01)
(450.0,OD01,1,01)
(800.0,OD02,2,02)
(800.0,OD02,2,02)
(800.0,OD02,2,02)
(350.0,OD03,2,03)
(350.0,OD03,2,03)
(350.0,OD03,2,03)
(500.0,OD04,3,04)
(500.0,OD04,3,04)
(500.0,OD04,3,04)
(650.0,OD05,4,05)
(650.0,OD05,4,05)
(650.0,OD05,4,05)
```

EVALUATION FUNCTIONS CAN BE CATEGORISED INTO FOLLOWING TYPES

MATH FUNCTIONS

CHARARRAY FUNCTIONS

COMPLEX DATA TYPE FUNCTIONS

AGGREGATE FUNCTIONS

DATETIME FUNCTIONS

AGGREGATE FUNCTIONS

THESE FUNCTIONS ARE SUPPORTED IN PIG

- AVG
- CONCAT
- COUNT
- COUNT_STAR
- DIFF
- IsEmpty
- MAX
- MIN
- SIZE
- SUM
- TOKENIZE

THESE FUNCTIONS WILL BE USED IN GROUP COMMANDS - MORE IN A BIT

EVALUATION FUNCTIONS CAN BE CATEGORISED INTO FOLLOWING TYPES

MATH FUNCTIONS

CHARARRAY FUNCTIONS

COMPLEX DATA TYPE FUNCTIONS

AGGREGATE FUNCTIONS

DATETIME FUNCTIONS

COMPLEX DATA TYPE FUNCTIONS

THESE FUNCTIONS EXIST TO HELP YOU MANIPULATE COMPLEX DATA TYPES

NAME (SIGNATURE)	RETURN TYPE	DESCRIPTION
`TOTUPLE(expression [, expression ...])`	`Tuple`	Converts one or more expressions to type tuple.
`TOBAG(expression [, expression ...])`	`DataBag`	Converts one or more expressions to type bag.
`TOMAP(key-expression, value-expression [, key-expression, value-expression ...])`	`Map`	Converts key/value expression pairs into a map.
`TOP(int topN, column, relation)`	`DataBag`	Returns the top-n tuples from a bag of tuples.

COMPLEX DATA TYPE FUNCTIONS

THESE FUNCTIONS ARE TYPICALLY RUN USING THE FOREACH COMMAND

NAME (SIGNATURE)	RETURN TYPE	DESCRIPTION
`TOTUPLE(expression [, expression ...])`	`Tuple`	Converts one or more expressions to type tuple.
`TOBAG(expression [, expression ...])`	`DataBag`	Converts one or more expressions to type bag.
`TOMAP(key-expression, value-expression [, key-expression, value-expression ...])`	`Map`	Converts key/value expression pairs into a map.
`TOP(int topN, column, relation)`	`DataBag`	Returns the top-n tuples from a bag of tuples.

COMPLEX DATA TYPE FUNCTIONS

NAME (SIGNATURE)	RETURN TYPE	DESCRIPTION
TOTUPLE(expression [, expression ...])	Tuple	Converts one or more expressions to type tuple.
TOBAG(expression [, expression ...])	DataBag	Converts one or more expressions to type bag.
TOMAP(key-expression, value-expression [, key-expression, value-expression ...])	Map	Converts key/value expression pairs into a map.
TOP(int topN, column, relation)	DataBag	Returns the top-n tuples from a bag of tuples.

LET US SEE HOW WE CAN USE THEM

```
grunt>order_customer_map = foreach orders
generate(TOMAP(Order_ID,Customer_ID));
```

THIS WILL GENERATE A MAP OF ORDER_ID AND CUSTOMER_ID

COMPLEX DATA TYPE FUNCTIONS

```
grunt>order_customer_map = foreach orders
generate(TOMAP(Order_ID,Customer_ID));
```

THIS WILL GENERATE A MAP OF ORDER_ID AND CUSTOMER_ID

```
([OD01#1])
([OD01#1])
([OD01#1])
([OD02#2])
([OD02#2])
([OD02#2])
([OD03#2])
([OD03#2])
([OD03#2])
([OD04#3])
([OD04#3])
([OD04#3])
([OD05#4])
([OD05#4])
([OD05#4])
```

COMPLEX DATA TYPE FUNCTIONS

NAME (SIGNATURE)	RETURN TYPE	DESCRIPTION
TOTUPLE(expression [, expression ...])	Tuple	Converts one or more expressions to type tuple.
TOBAG(expression [, expression ...])	DataBag	Converts one or more expressions to type bag.
TOMAP(key-expression, value-expression [, key-expression, value-expression ...])	Map	Converts key/value expression pairs into a map.
TOP(int topN, column, relation)	DataBag	Returns the top-n tuples from a bag of tuples.

LET US SEE ONE MORE EXAMPLE

```
grunt>order_bag = foreach orders generate
TOBAG(Order_ID,Customer_ID,Order_Quantity,Order_Value,Order_Date,
oduct_ID,Product_Name) as Order_Customer_ID;
```

THIS WILL GENERATE A BAG OF ORDER TUPLES

COMPLEX DATA TYPE FUNCTIONS

```
grunt>order_bag = foreach orders generate
TOBAG(Order_ID,Customer_ID,Order_Quantity,Order_V
oduct_ID,Product_Name) as Order_Customer_ID;
```

THIS WILL GENERATE A BAG OF ORDER TUPLES

```
({(OD01),(1),(1),(450.0),(2016-01-24),(1),(Apple iPhone 4S)})
({(OD01),(1),(1),(450.0),(2016-01-24),(2),(Apple iPhone 5S)})
({(OD01),(1),(1),(450.0),(2016-01-24),(3),(Apple iPhone 6)})
({(OD02),(2),(2),(400.0),(2016-01-24),(1),(Apple iPhone 4S)})
({(OD02),(2),(2),(400.0),(2016-01-24),(2),(Apple iPhone 5S)})
({(OD02),(2),(2),(400.0),(2016-01-24),(3),(Apple iPhone 6)})
({(OD03),(2),(1),(350.0),(2016-01-25),(1),(Apple iPhone 4S)})
({(OD03),(2),(1),(350.0),(2016-01-25),(2),(Apple iPhone 5S)})
({(OD03),(2),(1),(350.0),(2016-01-25),(3),(Apple iPhone 6)})
({(OD03),(2),(1),(350.0),(2016-01-25),(3),(Apple iPhone 6)})
({(OD04),(3),(1),(500.0),(2016-01-25),(1),(Apple iPhone 4S)})
({(OD04),(3),(1),(500.0),(2016-01-25),(2),(Apple iPhone 5S)})
({(OD04),(3),(1),(500.0),(2016-01-27),(3),(Apple iPhone 6)})
({(OD05),(4),(1),(650.0),(2016-01-26),(1),(Apple iPhone 4S)})
({(OD05),(4),(1),(650.0),(2016-01-27),(2),(Apple iPhone 5S)})
({(OD05),(4),(1),(650.0),(2016-01-27),(3),(Apple iPhone 6)})
```

EVALUATION FUNCTIONS CAN BE CATEGORISED INTO FOLLOWING TYPES

MATH FUNCTIONS

CHARARRAY FUNCTIONS

COMPLEX DATA TYPE FUNCTIONS

AGGREGATE FUNCTIONS

DATETIME FUNCTIONS

DATETIME FUNCTIONS

- Datetime Functions
  - AddDuration
  - CurrentTime
  - DaysBetween
  - GetDay
  - GetHour
  - GetMilliSecond
  - GetMinute
  - GetMonth
  - GetSecond
  - GetWeek
  - GetWeekYear
  - GetYear
  - HoursBetween
  - MilliSecondsBetween
  - MinutesBetween
  - MonthsBetween
  - SecondsBetween
  - SubtractDuration
  - ToDate
  - WeeksBetween
  - YearsBetween

THESE FUNCTIONS EXIST TO HELP YOU MANIPULATE THE DATETIME DATA TYPE

THE FUNCTION YOU MUST KNOW IS ToDate

DATETIME FUNCTIONS

THE FUNCTION YOU MUST KNOW IS ToDate

- Datetime Functions
  - AddDuration
  - CurrentTime
  - DaysBetween
  - GetDay
  - GetHour
  - GetMilliSecond
  - GetMinute
  - GetMonth
  - GetSecond
  - GetWeek
  - GetWeekYear
  - GetYear
  - HoursBetween
  - MilliSecondsBetween
  - MinutesBetween
  - MonthsBetween
  - SecondsBetween
  - SubtractDuration
  - ToDate
  - WeeksBetween
  - YearsBetween

THIS FUNCTION IS USED TO TYPECAST OTHER DATA TYPES INTO PIG'S DATETTIME DATA TYPE

THE FUNCTION YOU MUST KNOW IS ToDate

THIS FUNCTION IS USED TO TYPECAST OTHER DATA TYPES INTO PIG'S DATETTIME DATA TYPE

PIG HAS ONLY ONE TIME RELATED DATA TYPE AND ALL THESE FUNCTIONS WILL WORK ONLY ON THAT DATETIME DATA TYPE

THE FUNCTION YOU MUST KNOW IS ToDate

CONSIDER OUR ORDERS DATA IN THIS FORM

HMM..THIS DATE IS CURRENTLY STORED AS A STRING

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

HMM..THIS DATE IS CURRENTLY STORED AS A STRING

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

WE WILL LOAD THIS TABLE AND TRANSFORM ITS ORDER_DATE FIELD USING THE ToDate FUNCTION

```
grunt>orders = load 'order_data.csv' using PigStorage(',') as (Order_ID:chararray,
Customer_ID:int, Order_Quantity:int,Order_Value:float,Order_Date, Product_ID:int,
Product_Name:chararray);
```

```
grunt> time_temp = foreach orders generate
Order_ID,Customer_ID,ToDate(Order_Date,'yyyy-mm-dd') as Order_TimeStamp;
```

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

```
grunt>orders = load 'order_data.csv' using PigStorage(',') as (Order_ID:chararray,
Customer_ID:int, Order_Quantity:int,Order_Value:float,Order_Date, Product_ID:int,
Product_Name:chararray);

grunt> time_temp = foreach orders generate
Order_ID,Customer_ID,ToDate(Order_Date,'yyyy-mm-dd') as Order_TimeStamp;
```

THIS IS THE CURRENT STRING FORMAT IN THE ORDERS RELATION

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

```
grunt> time_temp = foreach orders generate
Order_ID,Customer_ID,ToDate(Order_Date,'yyyy-mm-dd') as Order_TimeStamp;
```

OUTPUT OF DUMP time_temp

```
(OD01,1,2016-01-24T00:01:00.000+05:30)
(OD01,1,2016-01-24T00:01:00.000+05:30)
(OD01,1,2016-01-24T00:01:00.000+05:30)
(OD02,2,2016-01-24T00:01:00.000+05:30)
(OD02,2,2016-01-24T00:01:00.000+05:30)
(OD02,2,2016-01-24T00:01:00.000+05:30)
(OD03,2,2016-01-25T00:01:00.000+05:30)
```

```
grunt> time_temp = foreach orders generate
Order_ID,Customer_ID,ToDate(Order_Date,'yyyy-mm-dd') as Order_TimeStamp;
```

NOW WE CAN USE ANY OF OUR DATETIME FUNCTIONS

- Datetime Functions
  - AddDuration
  - CurrentTime
  - DaysBetween
  - GetDay
  - GetHour
  - GetMilliSecond
  - GetMinute
  - GetMonth
  - GetSecond
  - GetWeek
  - GetWeekYear
  - GetYear
  - HoursBetween
  - MilliSecondsBetween
  - MinutesBetween
  - MonthsBetween
  - SecondsBetween
  - SubtractDuration
  - ToDate
  - WeeksBetween
  - YearsBetween

```
(OD01,1,2016-01-24T00:01:00.000+05:30)
(OD01,1,2016-01-24T00:01:00.000+05:30)
(OD01,1,2016-01-24T00:01:00.000+05:30)
(OD02,2,2016-01-24T00:01:00.000+05:30)
(OD02,2,2016-01-24T00:01:00.000+05:30)
(OD02,2,2016-01-24T00:01:00.000+05:30)
(OD03,2,2016-01-25T00:01:00.000+05:30)
```

```
(OD01,1,2016-01-24T00:01:00.000+05:30)
(OD01,1,2016-01-24T00:01:00.000+05:30)
(OD01,1,2016-01-24T00:01:00.000+05:30)
(OD02,2,2016-01-24T00:01:00.000+05:30)
(OD02,2,2016-01-24T00:01:00.000+05:30)
(OD02,2,2016-01-24T00:01:00.000+05:30)
(OD03,2,2016-01-25T00:01:00.000+05:30)
```

NOW WE CAN USE ANY OF OUR DATETIME FUNCTIONS

- Datetime Functions
  - AddDuration
  - CurrentTime
  - DaysBetween
  - GetDay
  - GetHour
  - GetMilliSecond
  - GetMinute
  - GetMonth
  - GetSecond
  - GetWeek
  - GetWeekYear
  - GetYear
  - HoursBetween
  - MilliSecondsBetween
  - MinutesBetween
  - MonthsBetween
  - SecondsBetween
  - SubtractDuration
  - ToDate
  - WeeksBetween
  - YearsBetween

```
grunt>order_month_table = foreach time_temp generate
Order_ID,Customer_ID,GetMonth(Order_TimeStamp);

grunt> dump order_month_table;
```

```
grunt>order_month_table = foreach time_temp generate
Order_ID,Customer_ID,GetMonth(Order_TimeStamp);

grunt> dump order_month_table;
```

OUTPUT OF THIS COMMAND IS

```
(OD01,1,1)
(OD01,1,1)
(OD01,1,1)
(OD02,2,1)
(OD02,2,1)
(OD02,2,1)
(OD03,2,1)
(OD03,2,1)
(OD03,2,1)
(OD04,3,1)
(OD04,3,1)
```

- Datetime Functions
  - AddDuration
  - CurrentTime
  - DaysBetween
  - GetDay
  - GetHour
  - GetMilliSecond
  - GetMinute
  - GetMonth
  - GetSecond
  - GetWeek
  - GetWeekYear
  - GetYear
  - HoursBetween
  - MilliSecondsBetween
  - MinutesBetween
  - MonthsBetween
  - SecondsBetween
  - SubtractDuration
  - ToDate
  - WeeksBetween
  - YearsBetween

1. SELECTING SPECIFIC FIELDS FROM THE RELATION

2. APPLYING SPECIFIC FUNCTIONS TO A FIELD

3. SELECTING A SPECIFIC NUMBER OF RECORDS, OR A SPECIFIC NUMBER OF DISTINCT VALUES

4. ORDERING RECORDS BASED ON SOME SPECIFIC COLUMN

5. FILTERING RECORDS BASED ON A CONDITION

6. GROUPING/AGGREGATING DATA BASED ON SPECIFIC COLUMNS

7. JOINING DATA OF ONE RELATION WITH ANOTHER RELATION

8. TRANSFORMING NESTED DATA IN A COLUMN TO MULTIPLE ROWS

9. SAMPLING RECORDS

1. SELECTING SPECIFIC FIELDS FROM THE RELATION

2. APPLYING SPECIFIC FUNCTIONS TO A FIELD

3. SELECTING A SPECIFIC NUMBER OF RECORDS, OR A SPECIFIC NUMBER OF DISTINCT VALUES

4. ORDERING RECORDS BASED ON SOME SPECIFIC COLUMN

5. FILTERING RECORDS BASED ON A CONDITION

6. GROUPING/AGGREGATING DATA BASED ON SPECIFIC COLUMNS

7. JOINING DATA OF ONE RELATION WITH ANOTHER RELATION

8. TRANSFORMING NESTED DATA IN A COLUMN TO MULTIPLE ROWS

9. SAMPLING RECORDS

3. SELECTING A SPECIFIC NUMBER OF RECORDS, OR A SPECIFIC NUMBER OF DISTINCT VALUES

THERE ARE TWO IDEAS HERE - DISTINCT VALUES AND A FIXED NUMBER OF ROWS

LET US TALK ABOUT DISTINCT VALUES FIRST

LET US TALK ABOUT DISTINCT VALUES FIRST

THIS IS DONE BY "DISTINCT" COMMAND

THIS COMMAND REMOVES DUPLICATE RECORDS

LET US TALK ABOUT DISTINCT VALUES FIRST

THIS IS DONE BY "DISTINCT" COMMAND

THIS COMMAND REMOVES DUPLICATE RECORDS

DISTINCT COMMAND IN PIG HAS LIMITED CAPABILITY

IT WORKS ONLY ON ENTIRE RECORDS AND NOT ON INDIVIDUAL FIELDS

DISTINCT COMMAND IN PIG HAS LIMITED CAPABILITY

IT WORKS ONLY ON ENTIRE RECORDS AND NOT ON INDIVIDUAL FIELDS

LET US SAY WE WANT UNIQUE ORDERS

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

IT WORKS ONLY ON ENTIRE RECORDS AND NOT ON INDIVIDUAL FIELDS

LET US SAY WE WANT UNIQUE ORDERS

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

```
grunt> orders_temp = foreach orders generate Order_ID;
grunt> distinct_orders = distinct orders_temp;
grunt> dump distinct_orders;
```

```
(OD01)
(OD02)
(OD03)
(OD04)
(OD05)
```

WE CANNOT DIRECTLY ACT ON THE ORDER_ID FIELD IN THE ORDERS TABLE

3. SELECTING A SPECIFIC NUMBER OF RECORDS, OR A SPECIFIC NUMBER OF DISTINCT VALUES

THERE ARE TWO IDEAS HERE - DISTINCT VALUES AND A FIXED NUMBER OF ROWS

NOW LET US TALK ABOUT FIXED NUMBER OF ROWS

NOW LET US TALK ABOUT FIXED NUMBER OF ROWS

THIS IS DONE VIA THE "LIMIT" COMMAND

FOLLOWED BY RELATION NAME AND N WHERE N IS NUMBER OF ROWS

THIS IS DONE VIA THE "LIMIT" COMMAND

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

LET US SAY WE WANT TO PRINT ONLY 3 ROWS TO THE SCREEN

LET US SAY WE WANT TO PRINT ONLY 3 ROWS TO THE SCREEN

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

```
grunt> orders_temp = limit orders 3;
grunt> dump orders_temp;
```

```
(OD01,1,1,450.0,2016-01-24,1,Apple iPhone 4S)
(OD01,1,1,450.0,2016-01-24,2,Apple iPhone 5S)
(OD01,1,1,450.0,2016-01-24,3,Apple iPhone 6)
```

1. SELECTING SPECIFIC FIELDS FROM THE RELATION

2. APPLYING SPECIFIC FUNCTIONS TO A FIELD

3. SELECTING A SPECIFIC NUMBER OF RECORDS, OR A SPECIFIC NUMBER OF DISTINCT VALUES

4. ORDERING RECORDS BASED ON SOME SPECIFIC COLUMN

5. FILTERING RECORDS BASED ON A CONDITION

6. GROUPING/AGGREGATING DATA BASED ON SPECIFIC COLUMNS

7. JOINING DATA OF ONE RELATION WITH ANOTHER RELATION

8. TRANSFORMING NESTED DATA IN A COLUMN TO MULTIPLE ROWS

9. SAMPLING RECORDS

1. SELECTING SPECIFIC FIELDS FROM THE RELATION

2. APPLYING SPECIFIC FUNCTIONS TO A FIELD

3. SELECTING A SPECIFIC NUMBER OF RECORDS, OR A SPECIFIC NUMBER OF DISTINCT VALUES

4. ORDERING RECORDS BASED ON SOME SPECIFIC COLUMN

5. FILTERING RECORDS BASED ON A CONDITION

6. GROUPING/AGGREGATING DATA BASED ON SPECIFIC COLUMNS

7. JOINING DATA OF ONE RELATION WITH ANOTHER RELATION

8. TRANSFORMING NESTED DATA IN A COLUMN TO MULTIPLE ROWS

9. SAMPLING RECORDS

4. ORDERING RECORDS BASED ON SOME SPECIFIC COLUMN

THIS IS DONE VIA A SIMPLE "ORDER BY" COMMAND

4. ORDERING RECORDS BASED ON SOME SPECIFIC COLUMN

THIS IS DONE VIA A SIMPLE "ORDER BY" COMMAND

YOU CAN SPECIFY WHETHER THE ORDER SHOULD BE ASCENDING OR DESCENDING

BY DEFAULT IT IS ASCENDING

YOU CAN SPECIFY WHETHER THE ORDER SHOULD BE ASCENDING OR DESCENDING

BY DEFAULT IT IS ASCENDING

CONSIDER OUR ORDERS DATA IN THIS FORM

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

LET US SAY WE WANT TO ORDER IT BY "ORDER_ID" IN DESCENDING ORDER

YOU CAN SPECIFY WHETHER THE ORDER SHOULD BE ASCENDING OR DESCENDING

LET US SAY WE WANT TO ORDER IT BY "ORDER_ID" IN DESCENDING ORDER

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

```
grunt> orders_desc = order orders by Order_ID desc;

grunt> dump orders_desc;
```

```
(OD05,4,1,650.0,2016-01-27,2,Apple iPhone 5S)
(OD05,4,1,650.0,2016-01-27,3,Apple iPhone 6)
(OD05,4,1,650.0,2016-01-26,1,Apple iPhone 4S)
(OD04,3,1,500.0,2016-01-27,3,Apple iPhone 6)
(OD04,3,1,500.0,2016-01-25,2,Apple iPhone 5S)
(OD04,3,1,500.0,2016-01-25,1,Apple iPhone 4S)
(OD03,2,1,350.0,2016-01-25,2,Apple iPhone 5S)
(OD03,2,1,350.0,2016-01-25,3,Apple iPhone 6)
(OD03,2,1,350.0,2016-01-25,1,Apple iPhone 4S)
(OD02,2,2,400.0,2016-01-24,3,Apple iPhone 6)
(OD02,2,2,400.0,2016-01-24,2,Apple iPhone 5S)
(OD02,2,2,400.0,2016-01-24,1,Apple iPhone 4S)
(OD01,1,1,450.0,2016-01-24,3,Apple iPhone 6)
(OD01,1,1,450.0,2016-01-24,2,Apple iPhone 5S)
(OD01,1,1,450.0,2016-01-24,1,Apple iPhone 4S)
```

1. SELECTING SPECIFIC FIELDS FROM THE RELATION

2. APPLYING SPECIFIC FUNCTIONS TO A FIELD

3. SELECTING A SPECIFIC NUMBER OF RECORDS, OR A SPECIFIC NUMBER OF DISTINCT VALUES

4. ORDERING RECORDS BASED ON SOME SPECIFIC COLUMN

5. FILTERING RECORDS BASED ON A CONDITION

6. GROUPING/AGGREGATING DATA BASED ON SPECIFIC COLUMNS

7. JOINING DATA OF ONE RELATION WITH ANOTHER RELATION

8. TRANSFORMING NESTED DATA IN A COLUMN TO MULTIPLE ROWS

9. SAMPLING RECORDS

1. SELECTING SPECIFIC FIELDS FROM THE RELATION

2. APPLYING SPECIFIC FUNCTIONS TO A FIELD

3. SELECTING A SPECIFIC NUMBER OF RECORDS, OR A SPECIFIC NUMBER OF DISTINCT VALUES

4. ORDERING RECORDS BASED ON SOME SPECIFIC COLUMN

5. FILTERING RECORDS BASED ON A CONDITION

6. GROUPING/AGGREGATING DATA BASED ON SPECIFIC COLUMNS

7. JOINING DATA OF ONE RELATION WITH ANOTHER RELATION

8. TRANSFORMING NESTED DATA IN A COLUMN TO MULTIPLE ROWS

9. SAMPLING RECORDS

5. FILTERING RECORDS BASED ON A CONDITION

THIS IS DONE VIA THE FILTER COMMAND

5. FILTERING RECORDS BASED ON A CONDITION

THE FILTER STATEMENT IS SIMILAR TO THE WHERE CLAUSE OF SQL

5. FILTERING RECORDS BASED ON A CONDITION

THE FILTER STATEMENT CONTAINS A PREDICATE

IF THE PREDICATE IS TRUE FOR A RECORD, THE RECORD BECOMES PART OF THE RESULT DATASET (RELATION)

THE FILTER STATEMENT CONTAINS A PREDICATE

IF THE PREDICATE IS TRUE FOR A RECORD, THE RECORD BECOMES PART OF THE RESULT DATASET(RELATION)

PREDICATES CONTAIN RELATIONAL OPERATORS

==

>

<

<=

!=

>=

AND FILTER FUNCTIONS

PREDICATES CONTAIN RELATIONAL OPERATORS

==

>

<

<=

!=

>=

AND FILTER FUNCTIONS

ALL OF THEM CAN BE APPLIED TO SCALAR DATA TYPES

PREDICATES CONTAIN RELATIONAL OPERATORS

==

!=

>

>=

<

<=

AND FILTER FUNCTIONS

ALL OF THEM CAN BE APPLIED TO SCALAR DATA TYPES

ONLY == AND != APPLY TO MAPS AND TUPLES

PREDICATES CONTAIN RELATIONAL OPERATORS

== > < <=

!= >=

AND FILTER FUNCTIONS

ALL OF THEM CAN BE APPLIED TO SCALAR DATA TYPES

ONLY == AND != APPLY TO MAPS AND TUPLES

NONE OF THEM CAN BE APPLIED TO BAGS

PREDICATES CONTAIN RELATIONAL OPERATORS

ALL OF THEM CAN BE APPLIED TO SCALAR DATA TYPES

ONLY == AND != APPLY TO MAPS AND TUPLES

NONE OF THEM CAN BE APPLIED TO BAGS

PREDICATES CONTAIN RELATIONAL OPERATORS

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

LET US SAY WE WANT TO SELECT ONLY THOSE ORDERS WHERE QUANTITY IS GREATER THAN OR EQUAL TO 2

LET US SAY WE WANT TO SELECT ONLY THOSE ORDERS WHERE QUANTITY IS GREATER THAN OR EQUAL TO 2

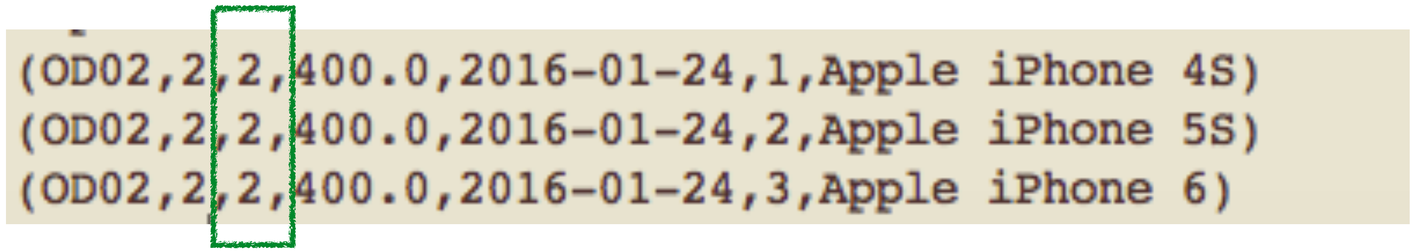
ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

```
grunt> orders_quantity_filter = filter orders by
Order_Quantity >=2;
grunt> dump orders_quantity_filter;
```

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

```
grunt> orders_quantity_filter = filter orders by Order_Quantity >=2;
grunt> dump orders_quantity_filter;
```

OUTPUT IS



PREDICATES CONTAIN RELATIONAL OPERATORS

==	>	<
!=	>=	<=

AND FILTER FUNCTIONS

NOW FOR FILTER FUNCTIONS

NOW FOR FILTER FUNCTIONS

AND FILTER FUNCTIONS

FILTER FUNCTIONS ARE BUILT-IN UDFs WHICH ARE USED WITH THE FILTER COMMAND

FILER FUNCTIONS ARE BUILT-IN UDFs WHICH ARE USED WITH FILTER COMMAND

AND FILTER FUNCTIONS

SOME IMPORTANT FILTER FUNCTIONS THAT ARE COMMONLY USED ARE

matches

IsEmpty

SOME IMPORTANT FILTER FUNCTIONS THAT ARE BEING USED ARE

matches

IsEmpty

matches IS USED WITH CHARARRAY TO SEE IF THE VALUE MATCHES AN EXPRESSION

matches IS USED WITH CHARARRAY TO SEE IF THE VALUE MATCHES AN EXPRESSION

matches

IsEmpty

LET US SAY WE WANT TO SELECT RECORDS WHERE ORDER_ID='OD02'

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

MATCHES

LET US SAY WE WANT TO SELECT RECORDS WHERE ORDER_ID='OD02'

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

```
grunt>orders_id_filter = filter orders by Order_ID matches 'OD02';
grunt> dump orders_id_filter;
```

MATCHES

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

```
grunt>orders_id_filter = filter orders by Order_ID matches 'OD02';
grunt> dump orders_id_filter;
```

```
(OD02,2,2,400.0,2016-01-24,1,Apple iPhone 4S)
(OD02,2,2,400.0,2016-01-24,2,Apple iPhone 5S)
(OD02,2,2,400.0,2016-01-24,3,Apple iPhone 6)
```

USING MATCHES, WE CAN SELECT STRINGS WHICH BEGIN OR END WITH A SPECIFIC EXPRESSION

MATCHES

LET US SAY WE WANT TO SELECT RECORDS WHERE PRODUCT_NAME STARTS WITH 'APPLE IPHONE 4';

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

```
grunt>product_id_filter = filter orders by Product_Name matches
'Apple iPhone 4.*';
grunt> dump product_id_filter;
```

MATCHES

```
grunt>product_id_filter = filter orders by Product_Name matches
'Apple iPhone 4.*';
grunt> dump product_id_filter;
```

```
(OD01,1,1,450.0,2016-01-24,1,Apple iPhone 4S)
(OD02,2,2,400.0,2016-01-24,1,Apple iPhone 4S)
(OD03,2,1,350.0,2016-01-25,1,Apple iPhone 4S)
(OD04,3,1,500.0,2016-01-25,1,Apple iPhone 4S)
(OD05,4,1,650.0,2016-01-26,1,Apple iPhone 4S)
```

NOTICE THE '.*' REGULAR EXPRESSION

SOME IMPORTANT FILTER FUNCTIONS THAT ARE BEING USED ARE

matches

IsEmpty

SOME IMPORTANT FILTER FUNCTIONS THAT ARE BEING USED ARE

matches

IsEmpty

THIS IS USED TO CHECK IF THE MAP OR BAG IS EMPTY

IsEmpty

LET US CONSIDER A RELATION (order_bag)

SCHEMA OF order_bag

```
grunt> describe order_bag;
order_bag: {order_customer_id: {TUPLE_NAME: (Order_ID: chararray,Customer_ID: int,Order_Quantity: int,Order_Value: float
,Order_Date: chararray,Product_ID: int,Product_Name: chararray)}}
```

HOW WILL THIS LOOK?

```
({(OD01,1,1,450.0,2016-01-24,1,Apple iPhone 4S)})
({(OD01,1,1,450.0,2016-01-24,2,Apple iPhone 5S)})
({(OD01,1,1,450.0,2016-01-24,3,Apple iPhone 6)})
({(OD02,2,2,400.0,2016-01-24,1,Apple iPhone 4S)})
({(OD02,2,2,400.0,2016-01-24,2,Apple iPhone 5S)})
({(OD02,2,2,400.0,2016-01-24,3,Apple iPhone 6)})
({(OD03,2,1,350.0,2016-01-25,1,Apple iPhone 4S)})
({(OD03,2,1,350.0,2016-01-25,2,Apple iPhone 5S)})
({(OD03,2,1,350.0,2016-01-25,3,Apple iPhone 6)})
({(OD04,3,1,500.0,2016-01-25,1,Apple iPhone 4S)})
({(OD04,3,1,500.0,2016-01-25,2,Apple iPhone 5S)})
({(OD04,3,1,500.0,2016-01-27,3,Apple iPhone 6)})
({(OD05,4,1,650.0,2016-01-26,1,Apple iPhone 4S)})
({(OD05,4,1,650.0,2016-01-27,2,Apple iPhone 5S)})
({(OD05,4,1,650.0,2016-01-27,3,Apple iPhone 6)})
```

IsEmpty

```
({(OD01,1,1,450.0,2016-01-24,1,Apple iPhone 4S)})
({(OD01,1,1,450.0,2016-01-24,2,Apple iPhone 5S)})
({(OD01,1,1,450.0,2016-01-24,3,Apple iPhone 6)})
({(OD02,2,2,400.0,2016-01-24,1,Apple iPhone 4S)})
({(OD02,2,2,400.0,2016-01-24,2,Apple iPhone 5S)})
({(OD02,2,2,400.0,2016-01-24,3,Apple iPhone 6)})
({(OD03,2,1,350.0,2016-01-25,1,Apple iPhone 4S)})
({(OD03,2,1,350.0,2016-01-25,2,Apple iPhone 5S)})
({(OD03,2,1,350.0,2016-01-25,3,Apple iPhone 6)})
({(OD04,3,1,500.0,2016-01-25,1,Apple iPhone 4S)})
({(OD04,3,1,500.0,2016-01-25,2,Apple iPhone 5S)})
({(OD04,3,1,500.0,2016-01-27,3,Apple iPhone 6)})
({(OD05,4,1,650.0,2016-01-26,1,Apple iPhone 4S)})
({(OD05,4,1,650.0,2016-01-27,2,Apple iPhone 5S)})
({(OD05,4,1,650.0,2016-01-27,3,Apple iPhone 6)})
```

SCHEMA OF order_bag

```
grunt> describe order_bag;
order_bag: {order_customer_id: {TUPLE_NAME: (Order_ID: chararray,Customer_ID: int,Order_Quantity: int,Order_Value: float
,Order_Date: chararray,Product_ID: int,Product_Name: chararray)}}
```

EACH RECORD OF THIS RELATION IS A BAG

IF YOU WANT TO REMOVE THOSE RECORDS WHERE THE BAG HAS NOTHING INSIDE IT, WE USE IsEmpty

BAGS ARE COMPLEX DATA TYPE, THEY CAN'T BE NULL. THEY WILL BE EMPTY

IsEmpty

```
({(OD01,1,1,450.0,2016-01-24,1,Apple iPhone 4S)})
({(OD01,1,1,450.0,2016-01-24,2,Apple iPhone 5S)})
({(OD01,1,1,450.0,2016-01-24,3,Apple iPhone 6)})
({(OD02,2,2,400.0,2016-01-24,1,Apple iPhone 4S)})
({(OD02,2,2,400.0,2016-01-24,2,Apple iPhone 5S)})
({(OD02,2,2,400.0,2016-01-24,3,Apple iPhone 6)})
({(OD03,2,1,350.0,2016-01-25,1,Apple iPhone 4S)})
({(OD03,2,1,350.0,2016-01-25,2,Apple iPhone 5S)})
({(OD03,2,1,350.0,2016-01-25,3,Apple iPhone 6)})
({(OD04,3,1,500.0,2016-01-25,1,Apple iPhone 4S)})
({(OD04,3,1,500.0,2016-01-25,2,Apple iPhone 5S)})
({(OD04,3,1,500.0,2016-01-27,3,Apple iPhone 6)})
({(OD05,4,1,650.0,2016-01-26,1,Apple iPhone 4S)})
({(OD05,4,1,650.0,2016-01-27,2,Apple iPhone 5S)})
({(OD05,4,1,650.0,2016-01-27,3,Apple iPhone 6)})
```

EACH RECORD OF THIS RELATION IS A BAG

```
grunt> describe order_bag;
order_bag: {order_customer_id: {TUPLE_NAME: (Order_ID: chararray,Customer_ID: int,Order_Quantity: int,Order_Value: float
,Order_Date: chararray,Product_ID: int,Product_Name: chararray)}}
```

IF YOU WANT TO REMOVE THOSE RECORDS WHERE THE BAG HAS NOTHING INSIDE IT, WE USE IsEmpty

```
grunt > product_order_bag = filter order_bag by not
IsEmpty(order_customer_id);
```

IsEmpty

LET US CONSIDER A MAP (map_example)

SCHEMA OF map_example

```
map_example: {Customer_Fields: map[chararray]}
```

HOW IT WILL LOOK LIKE?

```
([Customer_Name#'Hari',Customer_Id#'101'])
([Customer_Name#'Niki',Customer_Id#'102'])
```

IsEmpty

SCHEMA OF map_example

```
([Customer_Name#'Hari',Customer_Id#'101'])
([Customer_Name#'Niki',Customer_Id#'102'])
```

```
map_example: {Customer_Fields: map[chararray]}
```

EACH RECORD OF THIS RELATION IS A MAP

IF YOU WANT TO REMOVE THOSE RECORDS WHERE THE MAP HAS NOTHING INSIDE IT, WE USE IsEmpty

MAP IS COMPLEX DATA TYPE, IT CAN'T BE NULL. IT WILL BE EMPTY

IsEmpty

EACH RECORD OF THIS RELATION IS A MAP

```
([Customer_Name#'Hari',Customer_Id#'101'])
([Customer_Name#'Niki',Customer_Id#'102'])
```

```
map_example: {Customer_Fields: map[chararray]}
```

IF YOU WANT TO REMOVE THOSE RECORDS WHERE THE MAP HAS NOTHING INSIDE IT, WE USE IsEmpty

```
grunt > map_example_filter = filter map_example by not
IsEmpty(Customer_Fields);
```

WHEN YOU ARE LOOKING FOR NULL VALUES

YOU USE IS NULL

```
grunt>null_id_filter = filter orders by Order_ID IS NULL;
```

PREDICATES CONTAIN RELATIONAL OPERATORS

==	>	<
!=	>=	<=

AND FILTER FUNCTIONS

WITH ALL THESE OPERATORS YOU CAN ALSO USE AND, NOT AND OR LOGICAL OPERATORS

WITH ALL THESE OPERATORS YOU CAN ALSO USE AND, NOT AND OR LOGICAL OPERATORS

LET US SAY WE WANT TO SELECT RECORDS WHERE ORDER_ID='OD02' AND ORDER_QUANTITY IS NOT EQUAL TO 1

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

LET US SAY WE WANT TO SELECT RECORDS WHERE ORDER_ID='OD02'

AND

ORDER_QUANTITY IS NOT EQUAL TO 1

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

```
grunt>orders_id_quantity_filter = filter orders by Order_ID matches 'OD02'
and not (Order_Quantity == 1);

grunt> dump orders_id_quantity_filter;
```

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

```
grunt>orders_id_quantity_filter = filter orders by Order_ID matches 'OD02'
and not (Order_Quantity == 1);

grunt> dump orders_id_quantity_filter;
```

```
(OD02,2,2,400.0,2016-01-24,1,Apple iPhone 4S)
(OD02,2,2,400.0,2016-01-24,2,Apple iPhone 5S)
(OD02,2,2,400.0,2016-01-24,3,Apple iPhone 6)
```

1. SELECTING SPECIFIC FIELDS FROM THE RELATION

2. APPLYING SPECIFIC FUNCTIONS TO A FIELD

3. SELECTING A SPECIFIC NUMBER OF RECORDS, OR A SPECIFIC NUMBER OF DISTINCT VALUES

4. ORDERING RECORDS BASED ON SOME SPECIFIC COLUMN

5. FILTERING RECORDS BASED ON A CONDITION

6. GROUPING/AGGREGATING DATA BASED ON SPECIFIC COLUMNS

7. JOINING DATA OF ONE RELATION WITH ANOTHER RELATION

8. TRANSFORMING NESTED DATA IN A COLUMN TO MULTIPLE ROWS

9. SAMPLING RECORDS

1. SELECTING SPECIFIC FIELDS FROM THE RELATION

2. APPLYING SPECIFIC FUNCTIONS TO A FIELD

3. SELECTING A SPECIFIC NUMBER OF RECORDS, OR A SPECIFIC NUMBER OF DISTINCT VALUES

4. ORDERING RECORDS BASED ON SOME SPECIFIC COLUMN

5. FILTERING RECORDS BASED ON A CONDITION

6. GROUPING/AGGREGATING DATA BASED ON SPECIFIC COLUMNS

7. JOINING DATA OF ONE RELATION WITH ANOTHER RELATION

8. TRANSFORMING NESTED DATA IN A COLUMN TO MULTIPLE ROWS

9. SAMPLING RECORDS

6. GROUPING/AGGREGATING DATA BASED ON SPECIFIC COLUMNS

THIS IS DONE VIA THE GROUP BY COMMAND

6. GROUPING/AGGREGATING DATA BASED ON SPECIFIC COLUMNS

THIS IS DONE VIA THE GROUP BY COMMAND

THE GROUP BY STATEMENT DOES THE SAME THING AS THE SQL GROUP BY COMMAND..

..BUT VERY DIFFERENTLY

THE GROUP BY STATEMENT DOES THE SAME THING AS THE SQL GROUP BY COMMAND..
..BUT VERY DIFFERENTLY

LET US SAY WE WANT TO CALCULATE TOTAL ORDER_QUANTITY PER ORDER

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

LET US SAY WE WANT TO CALCULATE TOTAL ORDER_QUANTITY PER ORDER

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

WHAT WOULD BE THE SQL COMMAND FOR THIS?

```
SELECT Order_ID, SUM(Order_Quantity)
FROM Orders
GROUP BY Order_ID
```

WHAT WOULD BE THE SQL COMMAND FOR THIS?

```
SELECT Order_ID, SUM(Order_Quantity)
FROM Orders
GROUP BY Order_ID
```

IN PIG, THIS COMMAND WILL SPLIT INTO 2 PARTS

IN PIG, THIS COMMAND WILL SPLIT INTO 2 PARTS

```
SELECT Order_ID, SUM(Order_Quantity)
FROM Orders
GROUP BY Order_ID
```

THIS IS DONE IN THE 1ST PART

IN PIG, THIS COMMAND WILL SPLIT INTO 2 PARTS

THIS IS DONE IN THE 1ST PART

```
SELECT Order_ID, SUM(Order_Quantity)
FROM Orders
GROUP BY Order_ID
```

THE CORRESPONDING PIG COMMAND FOR THIS PART IS

```
grunt> groupd = group orders BY Order_ID;
```

IN PIG, THIS COMMAND WILL SPLIT INTO 2 PARTS

THIS IS DONE IN THE 1ST PART

```
SELECT Order_ID, SUM(Order_Quantity)
FROM Orders
GROUP BY Order_ID
```

NOTICE THAT NO AGGREGATION IS PERFORMED

```
grunt> groupd = group orders BY Order_ID;
```

IN PIG, THIS COMMAND WILL SPLIT INTO 2 PARTS

THIS IS DONE IN THE 1ST PART

```
SELECT Order_ID, SUM(Order_Quantity)
FROM Orders
GROUP BY Order_ID
```

GROUP COMMAND SIMPLY COLLECTS ALL THE RECORDS WITH THE SAME KEY (ORDER_ID) AND PUTS THEM IN THE BAG

```
grunt> groupd = group orders by Order_ID;
```

GROUP COMMAND SIMPLY COLLECTS ALL THE RECORDS WITH THE SAME KEY(ORDER_ID) AND PUTS THEM IN THE BAG

```
grunt> groupd = group orders by Order_ID;
```

'IN THE BAG' ???

Yes!

WHAT WILL THAT BAG LOOK LIKE?

GROUP COMMAND SIMPLY COLLECTS ALL THE RECORDS WITH THE SAME KEY(ORDER_ID) AND PUTS THEM IN THE BAG

```
grunt> groupd = group orders by Order_ID;
```

WHAT WILL THAT BAG LOOK LIKE?

```
grunt> dump groupd;
```

```
(OD01,{(OD01,1,1,450.0,2016-01-24,2,Apple iPhone 5S),(OD01,1,1,450.0,2016-01-24,1,Apple iPhone 4S),(OD01,1,1,450.0,2016-01-24,3,Apple iPhone 6)})
(OD02,{(OD02,2,2,400.0,2016-01-24,1,Apple iPhone 4S),(OD02,2,2,400.0,2016-01-24,2,Apple iPhone 5S),(OD02,2,2,400.0,2016-01-24,3,Apple iPhone 6)})
(OD03,{(OD03,2,1,350.0,2016-01-25,2,Apple iPhone 5S),(OD03,2,1,350.0,2016-01-25,3,Apple iPhone 6),(OD03,2,1,350.0,2016-01-25,1,Apple iPhone 4S)})
(OD04,{(OD04,3,1,500.0,2016-01-25,1,Apple iPhone 4S),(OD04,3,1,500.0,2016-01-25,2,Apple iPhone 5S),(OD04,3,1,500.0,2016-01-27,3,Apple iPhone 6)})
(OD05,{(OD05,4,1,650.0,2016-01-26,1,Apple iPhone 4S),(OD05,4,1,650.0,2016-01-27,2,Apple iPhone 5S),(OD05,4,1,650.0,2016-01-27,3,Apple iPhone 6)})
```

GROUP COMMAND SIMPLY COLLECTS ALL THE RECORDS WITH THE SAME KEY(ORDER_ID) AND PUT THEM IN THE BAG

```
grunt> groupd = group orders by Order_ID;
```

WHAT WILL THAT BAG LOOK LIKE?

```
grunt> dump groupd;
```

```
(OD01,{(OD01,1,1,450.0,2016-01-24,2,Apple iPhone 5S),(OD01,1,1,450.0,2016-01-24,1,Apple iPhone 4S),(OD01,1,1,450.0,2016-01-24,3,Apple iPhone 6)})
(OD02,{(OD02,2,2,400.0,2016-01-24,1,Apple iPhone 4S),(OD02,2,2,400.0,2016-01-24,2,Apple iPhone 5S),(OD02,2,2,400.0,2016-01-24,3,Apple iPhone 6)})
(OD03,{(OD03,2,1,350.0,2016-01-25,2,Apple iPhone 5S),(OD03,2,1,350.0,2016-01-25,3,Apple iPhone 6),(OD03,2,1,350.0,2016-01-25,1,Apple iPhone 4S)})
(OD04,{(OD04,3,1,500.0,2016-01-25,1,Apple iPhone 4S),(OD04,3,1,500.0,2016-01-25,2,Apple iPhone 5S),(OD04,3,1,500.0,2016-01-27,3,Apple iPhone 6)})
(OD05,{(OD05,4,1,650.0,2016-01-26,1,Apple iPhone 4S),(OD05,4,1,650.0,2016-01-27,2,Apple iPhone 5S),(OD05,4,1,650.0,2016-01-27,3,Apple iPhone 6)})
```

IT IS A BAG WITH WHERE ALL RECORDS HAVE

ORDER_ID = ‘OD01’

GROUP COMMAND SIMPLY COLLECTS ALL THE RECORDS WITH THE SAME KEY(ORDER_ID) AND PUT THEM IN THE BAG

```
grunt> groupd = group orders by Order_ID;
```

WHAT WILL THAT BAG LOOK LIKE?

```
grunt> dump groupd;
```

```
(OD01,{(OD01,1,1,450.0,2016-01-24,2,Apple iPhone 5S),(OD01,1,1,450.0,2016-01-24,1,Apple iPhone 4S),(OD01,1,1,450.0,2016-01-24,3,Apple iPhone 6)})
(OD02,{(OD02,2,2,400.0,2016-01-24,1,Apple iPhone 4S),(OD02,2,2,400.0,2016-01-24,2,Apple iPhone 5S),(OD02,2,2,400.0,2016-01-24,3,Apple iPhone 6)})
(OD03,{(OD03,2,1,350.0,2016-01-25,2,Apple iPhone 5S),(OD03,2,1,350.0,2016-01-25,3,Apple iPhone 6),(OD03,2,1,350.0,2016-01-25,1,Apple iPhone 4S)})
(OD04,{(OD04,3,1,500.0,2016-01-25,1,Apple iPhone 4S),(OD04,3,1,500.0,2016-01-25,2,Apple iPhone 5S),(OD04,3,1,500.0,2016-01-27,3,Apple iPhone 6)})
(OD05,{(OD05,4,1,650.0,2016-01-26,1,Apple iPhone 4S),(OD05,4,1,650.0,2016-01-27,2,Apple iPhone 5S),(OD05,4,1,650.0,2016-01-27,3,Apple iPhone 6)})
```

IT IS A BAG WITH WHERE ALL RECORDS HAVE

ORDER_ID = ‘ODO2’

GROUP COMMAND SIMPLY COLLECTS ALL THE RECORDS WITH THE SAME KEY(ORDER_ID) AND PUT THEM IN THE BAG

```
grunt> groupd = group orders by Order_ID;
```

WHAT WILL THAT BAG LOOK LIKE?

```
grunt> dump groupd;
```

```
(OD01,{(OD01,1,1,450.0,2016-01-24,2,Apple iPhone 5S),(OD01,1,1,450.0,2016-01-24,1,Apple iPhone 4S),(OD01,1,1,450.0,2016-01-24,3,Apple iPhone 6)})
(OD02,{(OD02,2,2,400.0,2016-01-24,1,Apple iPhone 4S),(OD02,2,2,400.0,2016-01-24,2,Apple iPhone 5S),(OD02,2,2,400.0,2016-01-24,3,Apple iPhone 6)})
(OD03,{(OD03,2,1,350.0,2016-01-25,2,Apple iPhone 5S),(OD03,2,1,350.0,2016-01-25,3,Apple iPhone 6),(OD03,2,1,350.0,2016-01-25,1,Apple iPhone 4S)})
(OD04,{(OD04,3,1,500.0,2016-01-25,1,Apple iPhone 4S),(OD04,3,1,500.0,2016-01-25,2,Apple iPhone 5S),(OD04,3,1,500.0,2016-01-27,3,Apple iPhone 6)})
(OD05,{(OD05,4,1,650.0,2016-01-26,1,Apple iPhone 4S),(OD05,4,1,650.0,2016-01-27,2,Apple iPhone 5S),(OD05,4,1,650.0,2016-01-27,3,Apple iPhone 6)})
```

IT IS A BAG WITH WHERE ALL RECORDS HAVE

ORDER_ID = ‘ODO3’

```
grunt> groupd = group orders by Order_ID;
```

WHAT IS THE SCHEMA OF THE BAG?

```
grunt> describe groupd;
```

DESCRIBE IS USED TO GET THE SCHEMA OF ANY RELATION

```
grunt> groupd = Group orders BY Order_ID;
```

WHAT IS THE SCHEMA OF THE BAG?

```
grunt> describe groupd;
```

```
groupd: {group: chararray,orders: {(Order_ID: chararray,Customer_ID: int,Order_Quantity: int,Order_Value: float,Order_Date: bytearray,Product_ID: int,Product_
Ncdwme: chararray)}}
```

```
groupd: {group: chararray,orders: {(Order_ID: chararray,Customer_ID: int,Order_Quantity: int,Order_Value: float,Order_Date: bytearray,Product_ID: int,Product_
Ncdwme: chararray)}}
```

RECORDS COMING OUT OF A GROUP BY STATEMENT HAVE TWO FIELDS

- THE KEY FIELD WHOSE NAME IS 'GROUP'
- THE BAG FIELD WHOSE NAME IS SAME AS THAT OF RELATION WHICH IS GROUPED

HERE THE NAME WILL BE 'ORDERS'

```
groupd: {group: chararray,orders: {(Order_ID: chararray,Customer_ID: int,Order_Quantity: int,Order_Value: float,Order_Date: bytearray,Product_ID: int,Product_
Ncdwme: chararray)}}
```

RECORDS COMING OUT OF GROUP BY STATEMENT HAS TWO FIELDS

- THE KEY FIELD WHOSE NAME IS 'GROUP'

```
(OD01,{(OD01,1,1,450.0,2016-01-24,2,Apple iPhone 5S),(OD01,1,1,450.0,2016-01-24,1,Apple iPhone 4S),(OD01,1,1,450.0,2016-01-24,3,Apple iPhone 6)})
(OD02,{(OD02,2,2,400.0,2016-01-24,1,Apple iPhone 4S),(OD02,2,2,400.0,2016-01-24,2,Apple iPhone 5S),(OD02,2,2,400.0,2016-01-24,3,Apple iPhone 6)})
(OD03,{(OD03,2,1,350.0,2016-01-25,2,Apple iPhone 5S),(OD03,2,1,350.0,2016-01-25,3,Apple iPhone 6),(OD03,2,1,350.0,2016-01-25,1,Apple iPhone 4S)})
(OD04,{(OD04,3,1,500.0,2016-01-25,1,Apple iPhone 4S),(OD04,3,1,500.0,2016-01-25,2,Apple iPhone 5S),(OD04,3,1,500.0,2016-01-27,3,Apple iPhone 6)})
(OD05,{(OD05,4,1,650.0,2016-01-26,1,Apple iPhone 4S),(OD05,4,1,650.0,2016-01-27,2,Apple iPhone 5S),(OD05,4,1,650.0,2016-01-27,3,Apple iPhone 6)})
```

GROUP BY IS DONE ON ORDER_ID

```
groupd: {group: chararray,orders: {(Order_ID: chararray,Customer_ID: int,Order_Quantity: int,Order_Value: float,Order_Date: bytearray,Product_ID: int,Product_
Ncdwme: chararray)}}
```

HERE THE NAME IS 'ORDERS'

```
(OD01,{(OD01,1,1,450.0,2016-01-24,2,Apple iPhone 5S),(OD01,1,1,450.0,2016-01-24,1,Apple iPhone 4S),(OD01,1,1,450.0,2016-01-24,3,Apple iPhone 6)}
(OD02,{(OD02,2,2,400.0,2016-01-24,1,Apple iPhone 4S),(OD02,2,2,400.0,2016-01-24,2,Apple iPhone 5S),(OD02,2,2,400.0,2016-01-24,3,Apple iPhone 6)}
(OD03,{(OD03,2,1,350.0,2016-01-25,2,Apple iPhone 5S),(OD03,2,1,350.0,2016-01-25,3,Apple iPhone 6),(OD03,2,1,350.0,2016-01-25,1,Apple iPhone 4S)}
(OD04,{(OD04,3,1,500.0,2016-01-25,1,Apple iPhone 4S),(OD04,3,1,500.0,2016-01-25,2,Apple iPhone 5S),(OD04,3,1,500.0,2016-01-27,3,Apple iPhone 6)}
(OD05,{(OD05,4,1,650.0,2016-01-26,1,Apple iPhone 4S),(OD05,4,1,650.0,2016-01-27,2,Apple iPhone 5S),(OD05,4,1,650.0,2016-01-27,3,Apple iPhone 6)}
```

- THE KEY FIELD WHOSE NAME IS 'GROUP'

- THE BAG FIELD WHOSE NAME IS SAME AS THAT OF RELATION WHICH IS GROUPED

```
groupd: {group: chararray,orders: {(Order_ID: chararray,Customer_ID: int,Order_Quantity: int,Order_Value: float,Order_Date: bytearray,Product_ID: int,Product_
Ncdwme: chararray)}}
```

HERE THE NAME IS 'ORDERS'

RECORDS COMING OUT OF GROUP BY STATEMENT HAS TWO FIELDS

```
(OD01,{(OD01,1,1,450.0,2016-01-24,2,Apple iPhone 5S),(OD01,1,1,450.0,2016-01-24,1,Apple iPhone 4S),(OD01,1,1,450.0,2016-01-24,3,Apple iPhone 6)})
(OD02,{(OD02,2,2,400.0,2016-01-24,1,Apple iPhone 4S),(OD02,2,2,400.0,2016-01-24,2,Apple iPhone 5S),(OD02,2,2,400.0,2016-01-24,3,Apple iPhone 6)})
(OD03,{(OD03,2,1,350.0,2016-01-25,2,Apple iPhone 5S),(OD03,2,1,350.0,2016-01-25,3,Apple iPhone 6),(OD03,2,1,350.0,2016-01-25,1,Apple iPhone 4S)})
(OD04,{(OD04,3,1,500.0,2016-01-25,1,Apple iPhone 4S),(OD04,3,1,500.0,2016-01-25,2,Apple iPhone 5S),(OD04,3,1,500.0,2016-01-27,3,Apple iPhone 6)})
(OD05,{(OD05,4,1,650.0,2016-01-26,1,Apple iPhone 4S),(OD05,4,1,650.0,2016-01-27,2,Apple iPhone 5S),(OD05,4,1,650.0,2016-01-27,3,Apple iPhone 6)})
```

- THE KEY FIELD WHOSE NAME IS 'GROUP'

- BAG OF TUPLES WITH THE NAME `orders`

THE BAG SIMPLY COLLECTS ALL THE RECORDS WITH THE SAME KEY(ORDER_ID)

NO AGGREGATION HAS BEEN PERFORMED YET

THIS IS CARRIED OUT IN A SEPARATE COMMAND

AGGREGATION WILL HAPPEN ON THE RECORDS COMING OUT OF GROUP COMMAND

AGGREGATION WILL HAPPEN ON THE RECORD COMING OUT OF GROUP COMMAND

COMING BACK TO THE ORIGINAL PROBLEM

```
SELECT Order_ID, SUM(Order_Quantity)
FROM Orders
GROUP BY Order_ID
```

WE HAVE TO SUM Order_Quantity

AGGREGATION WILL HAPPEN ON THE RECORD COMING OUT OF GROUP COMMAND

```
SELECT Order_ID, SUM(Order_Quantity)
FROM Orders
GROUP BY Order_ID
```

THIS IS DONE IN A SEPARATE COMMAND WHICH OPERATES ON groupd

```
grunt> groupd = group orders by Order_ID;
```

THIS IS DONE IN A SEPARATE COMMAND WHICH OPERATES ON groupd

```
grunt> groupd = group orders by Order_ID;
```

```
grunt> aggr_by_order = foreach groupd generate
group,SUM(orders.Order_Quantity);
```

THIS IS DONE IN A SEPARATE COMMAND WHICH OPERATES ON groupd

```
grunt> groupd = Group orders BY Order_ID;
```

```
grunt> aggr_by_order = foreach groupd generate
group,SUM(orders.Order_Quantity);
```

OUTPUT WILL BE:

```
(OD01,3)
(OD02,6)
(OD03,3)
(OD04,3)
(OD05,3)
```

```
grunt> groupd = group orders by Order_ID;
```

```
grunt> aggr_by_order = foreach groupd generate
group,SUM(orders.Order_Quantity);
```

TO UNDERSTAND THIS WE SHOULD KEEP THE SCHEMA OF `groupd` IN MIND

```
(OD01,{(OD01,1,1,450.0,2016-01-24,2,Apple iPhone 5S),(OD01,1,1,450.0,2016-01-24,1,Apple iPhone 4S),(OD01,1,1,450.0,2016-01-24,3,Apple iPhone 6)})
(OD02,{(OD02,2,2,400.0,2016-01-24,1,Apple iPhone 4S),(OD02,2,2,400.0,2016-01-24,2,Apple iPhone 5S),(OD02,2,2,400.0,2016-01-24,3,Apple iPhone 6)})
(OD03,{(OD03,2,1,350.0,2016-01-25,2,Apple iPhone 5S),(OD03,2,1,350.0,2016-01-25,3,Apple iPhone 6),(OD03,2,1,350.0,2016-01-25,1,Apple iPhone 4S)})
(OD04,{(OD04,3,1,500.0,2016-01-25,1,Apple iPhone 4S),(OD04,3,1,500.0,2016-01-25,2,Apple iPhone 5S),(OD04,3,1,500.0,2016-01-27,3,Apple iPhone 6)})
(OD05,{(OD05,4,1,650.0,2016-01-26,1,Apple iPhone 4S),(OD05,4,1,650.0,2016-01-27,2,Apple iPhone 5S),(OD05,4,1,650.0,2016-01-27,3,Apple iPhone 6)})
```

```
grunt> groupd = Group orders BY Order_ID;
```

```
grunt> aggr_by_order = foreach groupd generate
group,SUM(orders.Order_Quantity);
```

TO UNDERSTAND THIS WE SHOULD KEEP THE SCHEMA OF groupd IN MIND

```
(OD01,{(OD01,1,1,450.0,2016-01-24,2,Apple iPhone 5S),(OD01,1,1,450.0,2016-01-24,1,Apple iPhone 4S),(OD01,1,1,450.0,2016-01-24,3,Apple iPhone 6)})
(OD02,{(OD02,2,2,400.0,2016-01-24,1,Apple iPhone 4S),(OD02,2,2,400.0,2016-01-24,2,Apple iPhone 5S),(OD02,2,2,400.0,2016-01-24,3,Apple iPhone 6)})
(OD03,{(OD03,2,1,350.0,2016-01-25,2,Apple iPhone 5S),(OD03,2,1,350.0,2016-01-25,3,Apple iPhone 6),(OD03,2,1,350.0,2016-01-25,1,Apple iPhone 4S)})
(OD04,{(OD04,3,1,500.0,2016-01-25,1,Apple iPhone 4S),(OD04,3,1,500.0,2016-01-25,2,Apple iPhone 5S),(OD04,3,1,500.0,2016-01-27,3,Apple iPhone 6)})
(OD05,{(OD05,4,1,650.0,2016-01-26,1,Apple iPhone 4S),(OD05,4,1,650.0,2016-01-27,2,Apple iPhone 5S),(OD05,4,1,650.0,2016-01-27,3,Apple iPhone 6)})
```

THIS PART SAYS THAT WE WILL USE groupd STRUCTURE FOR FINAL COMPUTATION

```
grunt> groupd = Group orders BY Order_ID;
```

```
grunt> aggr_by_order = foreach groupd generate
group,SUM(orders.Order_Quantity);
```

TO UNDERSTAND THIS WE SHOULD KEEP THE SCHEMA OF groupd IN MIND

```
(OD01,{(OD01,1,1,450.0,2016-01-24,2,Apple iPhone 5S),(OD01,1,1,450.0,2016-01-24,1,Apple iPhone 4S),(OD01,1,1,450.0,2016-01-24,3,Apple iPhone 6)})
(OD02,{(OD02,2,2,400.0,2016-01-24,1,Apple iPhone 4S),(OD02,2,2,400.0,2016-01-24,2,Apple iPhone 5S),(OD02,2,2,400.0,2016-01-24,3,Apple iPhone 6)})
(OD03,{(OD03,2,1,350.0,2016-01-25,2,Apple iPhone 5S),(OD03,2,1,350.0,2016-01-25,3,Apple iPhone 6),(OD03,2,1,350.0,2016-01-25,1,Apple iPhone 4S)})
(OD04,{(OD04,3,1,500.0,2016-01-25,1,Apple iPhone 4S),(OD04,3,1,500.0,2016-01-25,2,Apple iPhone 5S),(OD04,3,1,500.0,2016-01-27,3,Apple iPhone 6)})
(OD05,{(OD05,4,1,650.0,2016-01-26,1,Apple iPhone 4S),(OD05,4,1,650.0,2016-01-27,2,Apple iPhone 5S),(OD05,4,1,650.0,2016-01-27,3,Apple iPhone 6)})
```

GROUP REPRESENTS HERE THE KEY WHICH IS order_id

GROUP REPRESENTS HERE THE KEY WHICH IS order_id

```
grunt> aggr_by_order = foreach groupd generate
group,SUM(orders.Order_Quantity);
```

```
(OD01,{(OD01,1,1,450.0,2016-01-24,2,Apple iPhone 5S),(OD01,1,1,450.0,2016-01-24,1,Apple iPhone 4S),(OD01,1,1,450.0,2016-01-24,3,Apple iPhone 6)})
(OD02,{(OD02,2,2,400.0,2016-01-24,1,Apple iPhone 4S),(OD02,2,2,400.0,2016-01-24,2,Apple iPhone 5S),(OD02,2,2,400.0,2016-01-24,3,Apple iPhone 6)})
(OD03,{(OD03,2,1,350.0,2016-01-25,2,Apple iPhone 5S),(OD03,2,1,350.0,2016-01-25,3,Apple iPhone 6),(OD03,2,1,350.0,2016-01-25,1,Apple iPhone 4S)})
(OD04,{(OD04,3,1,500.0,2016-01-25,1,Apple iPhone 4S),(OD04,3,1,500.0,2016-01-25,2,Apple iPhone 5S),(OD04,3,1,500.0,2016-01-27,3,Apple iPhone 6)})
(OD05,{(OD05,4,1,650.0,2016-01-26,1,Apple iPhone 4S),(OD05,4,1,650.0,2016-01-27,2,Apple iPhone 5S),(OD05,4,1,650.0,2016-01-27,3,Apple iPhone 6)})
```

```
(OD01,3)
(OD02,6)
(OD03,3)
(OD04,3)
(OD05,3)
```

```
grunt> groupd = Group orders BY Order_ID;
```

```
grunt> aggr_by_order = foreach groupd generate
group,SUM(orders.Order_Quantity);
```

TO UNDERSTAND THIS WE SHOULD KEEP THE SCHEMA OF groupd IN MIND

```
(OD01,{(OD01,1,1,450.0,2016-01-24,2,Apple iPhone 5S),(OD01,1,1,450.0,2016-01-24,1,Apple iPhone 4S),(OD01,1,1,450.0,2016-01-24,3,Apple iPhone 6)})
(OD02,{(OD02,2,2,400.0,2016-01-24,1,Apple iPhone 4S),(OD02,2,2,400.0,2016-01-24,2,Apple iPhone 5S),(OD02,2,2,400.0,2016-01-24,3,Apple iPhone 6)})
(OD03,{(OD03,2,1,350.0,2016-01-25,2,Apple iPhone 5S),(OD03,2,1,350.0,2016-01-25,3,Apple iPhone 6),(OD03,2,1,350.0,2016-01-25,1,Apple iPhone 4S)})
(OD04,{(OD04,3,1,500.0,2016-01-25,1,Apple iPhone 4S),(OD04,3,1,500.0,2016-01-25,2,Apple iPhone 5S),(OD04,3,1,500.0,2016-01-27,3,Apple iPhone 6)})
(OD05,{(OD05,4,1,650.0,2016-01-26,1,Apple iPhone 4S),(OD05,4,1,650.0,2016-01-27,2,Apple iPhone 5S),(OD05,4,1,650.0,2016-01-27,3,Apple iPhone 6)})
```

SUM IS AN BUILT-IN AGGREGATE FUNCTION WHICH IS USED TO COMPUTE SUM

SUM IS AN BUILT-IN AGGREGATE FUNCTION WHICH IS USED TO COMPUTE SUM

PIG SUPPORTS MANY AGGREGATE FUNCTIONS

- AVG
- CONCAT
- COUNT
- COUNT_STAR
- DIFF
- IsEmpty
- MAX
- MIN
- SIZE
- SUM
- TOKENIZE

PIG SUPPORTS MANY AGGREGATE FUNCTIONS

THESE AGGREGATE FUNCTIONS ARE APPLIED TO THE FIELDS

- AVG
- CONCAT
- COUNT
- COUNT_STAR
- DIFF
- IsEmpty
- MAX
- MIN
- SIZE
- SUM
- TOKENIZE

AND ALL OF THE FUNCTIONS SHOULD BE WRITTEN IN CAPITAL LETTERS

```
grunt> groupd = Group orders BY Order_ID;
```

```
grunt> aggr_by_order = foreach groupd generate
group,SUM(orders.Order_Quantity);
```

TO UNDERSTAND THIS WE SHOULD KEEP THE SCHEMA OF `groupd` IN MIND

WE USED **orders.Order_Quantity**
INSTEAD OF **Order_Quantity**

WE USED `orders.Order_Quantity` INSTEAD OF `Order_Quantity`

```
grunt> aggr_by_order = foreach groupd generate
group,SUM(orders.Order_Quantity);
```

THIS IS BECAUSE WE ARE COMPUTING SUM ON THE BAG

'orders.Order_Quantity'

WE USED `orders.Order_Quantity`
INSTEAD OF `Order_Quantity`

THIS IS BECAUSE WE ARE COMPUTING SUM ON THE BAG
'orders.Order_Quantity'

`orders.Order_Quantity` WILL BE A BAG
OF `Order_Quantity` FOR EACH KEY

THIS IS BECAUSE WE ARE COMPUTING SUM ON THE BAG

'orders.Order_Quantity'

orders.Order_Quantity WILL BE A BAG OF **Order_Quantity** FOR EACH KEY

```
(OD01,{(OD01,1,1,450.0,2016-01-24,2,Apple iPhone 5S),(OD01,1,1,450.0,2016-01-24,1,Apple iPhone 4S),(OD01,1,1,450.0,2016-01-24,3,Apple iPhone 6)})
(OD02,{(OD02,2,2,400.0,2016-01-24,1,Apple iPhone 4S),(OD02,2,2,400.0,2016-01-24,2,Apple iPhone 5S),(OD02,2,2,400.0,2016-01-24,3,Apple iPhone 6)})
(OD03,{(OD03,2,1,350.0,2016-01-25,2,Apple iPhone 5S),(OD03,2,1,350.0,2016-01-25,3,Apple iPhone 6),(OD03,2,1,350.0,2016-01-25,1,Apple iPhone 4S)})
(OD04,{(OD04,3,1,500.0,2016-01-25,1,Apple iPhone 4S),(OD04,3,1,500.0,2016-01-25,2,Apple iPhone 5S),(OD04,3,1,500.0,2016-01-27,3,Apple iPhone 6)})
(OD05,{(OD05,4,1,650.0,2016-01-26,1,Apple iPhone 4S),(OD05,4,1,650.0,2016-01-27,2,Apple iPhone 5S),(OD05,4,1,650.0,2016-01-27,3,Apple iPhone 6)})
```

THE BAG WILL BE CONSTRUCTED FROM THIS BAG

orders.Order_Quantity WILL BE BAG OF Order_Quantity FOR EACH KEY

```
(OD01,{(OD01,1,1,450.0,2016-01-24,2,Apple iPhone 5S),(OD01,1,1,450.0,2016-01-24,1,Apple iPhone 4S),(OD01,1,1,450.0,2016-01-24,3,Apple iPhone 6)})
(OD02,{(OD02,2,2,400.0,2016-01-24,1,Apple iPhone 4S),(OD02,2,2,400.0,2016-01-24,2,Apple iPhone 5S),(OD02,2,2,400.0,2016-01-24,3,Apple iPhone 6)})
(OD03,{(OD03,2,1,350.0,2016-01-25,2,Apple iPhone 5S),(OD03,2,1,350.0,2016-01-25,3,Apple iPhone 6),(OD03,2,1,350.0,2016-01-25,1,Apple iPhone 4S)})
(OD04,{(OD04,3,1,500.0,2016-01-25,1,Apple iPhone 4S),(OD04,3,1,500.0,2016-01-25,2,Apple iPhone 5S),(OD04,3,1,500.0,2016-01-27,3,Apple iPhone 6)})
(OD05,{(OD05,4,1,650.0,2016-01-26,1,Apple iPhone 4S),(OD05,4,1,650.0,2016-01-27,2,Apple iPhone 5S),(OD05,4,1,650.0,2016-01-27,3,Apple iPhone 6)})
```

THE BAG WILL BE CONSTRUCTED FROM THIS BAG

orders.Order_Quantity = {1,1,1}

WHILE GROUPING, USE

`relation_name.field_name`

instead of `field_name`

LET US DO ONE MORE EXAMPLE

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

LET US SAY WE WANT TO SEE HOW MANY PRODUCTS OUR CUSTOMER BUY

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

LET US SAY WE WANT TO SEE HOW MANY PRODUCTS OUR CUSTOMER BUY

THE GROUPING WILL HAPPEN ON CUSTOMER_ID

```
grunt> groupd = group orders by Customer_ID;
```

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

LET US SAY WE WANT TO SEE HOW MANY PRODUCTS OUR CUSTOMER BUY

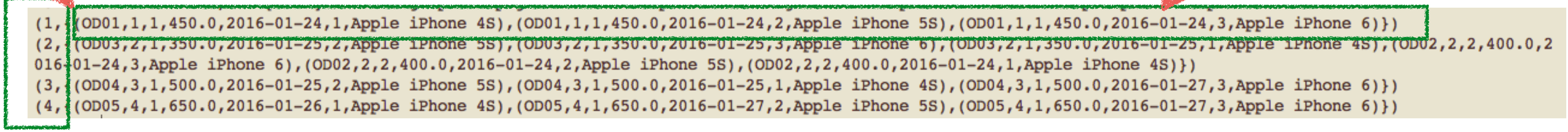
```
grunt> groupd = group orders by Customer_ID;
```

GROUPD WILL HAVE A KEY WITH NAME `group` AND A BAG WITH NAME `orders` HAVING ALL RECORDS WHICH HAVE SAME KEY

LET US SAY WE WANT TO SEE HOW MANY PRODUCTS OUR CUSTOMER BUY

```
grunt> groupd = group orders by Customer_ID;
```

GROUPD WILL HAVE A KEY WITH NAME **group** AND A BAG WITH NAME **orders** HAVING ALL RECORDS WHICH HAVE SAME KEY



```
(1,{(OD01,1,1,450.0,2016-01-24,1,Apple iPhone 4S),(OD01,1,1,450.0,2016-01-24,2,Apple iPhone 5S),(OD01,1,1,450.0,2016-01-24,3,Apple iPhone 6)})
(2,{(OD03,2,1,350.0,2016-01-25,2,Apple iPhone 5S),(OD03,2,1,350.0,2016-01-25,3,Apple iPhone 6),(OD03,2,1,350.0,2016-01-25,1,Apple iPhone 4S),(OD02,2,2,400.0,2016-01-24,3,Apple iPhone 6),(OD02,2,2,400.0,2016-01-24,2,Apple iPhone 5S),(OD02,2,2,400.0,2016-01-24,1,Apple iPhone 4S)})
(3,{(OD04,3,1,500.0,2016-01-25,2,Apple iPhone 5S),(OD04,3,1,500.0,2016-01-25,1,Apple iPhone 4S),(OD04,3,1,500.0,2016-01-27,3,Apple iPhone 6)})
(4,{(OD05,4,1,650.0,2016-01-26,1,Apple iPhone 4S),(OD05,4,1,650.0,2016-01-27,2,Apple iPhone 5S),(OD05,4,1,650.0,2016-01-27,3,Apple iPhone 6)})
```

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

LET US SAY WE WANT TO SEE HOW MANY PRODUCTS OUR CUSTOMER BUY

```
grunt> groupd = group orders by Customer_ID;
```

THEN PERFORM AN AGGREGATE FUNCTION ON GROUPD

```
grunt> aggr_by_cust = foreach groupd generate group,
COUNT(orders.Product_ID) as Total_Products;
```

```
grunt> groupd = group orders by Customer_ID;

grunt> aggr_by_cust = foreach groupd generate group,
COUNT(orders.Product_ID) as Total_Products;
```

THEN PERFORM AN AGGREAGATE FUNCTION ON GROUPD

OUTPUT OF THIS WILL BE

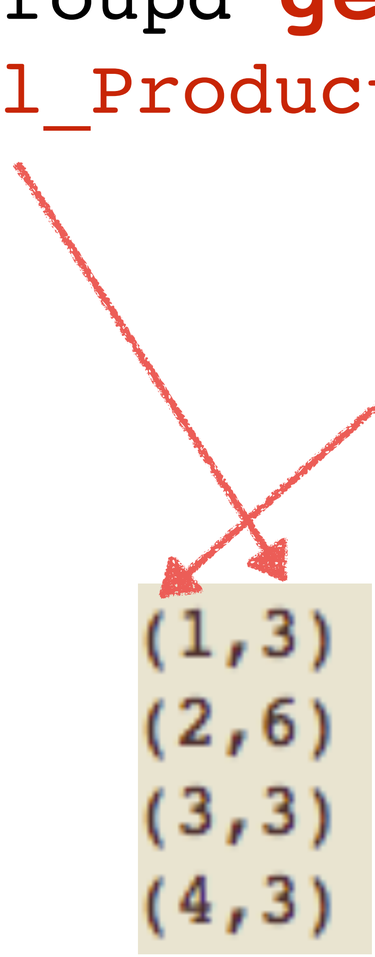
(Customer_ID,Total_Products)

```
grunt> groupd = Group orders BY Customer_ID;

grunt> aggr_by_cust = foreach groupd generate group,
COUNT(orders.Product_ID) as Total_Products;
```

OUTPUT OF THIS WILL BE

(CUSTOMER_ID,Total_Products)



NULL VALUES IN A GROUP

ALL RECORDS WITH THE NULL VALUE IN THE KEY ARE GROUPED TOGETHER

1. SELECTING SPECIFIC FIELDS FROM THE RELATION

2. APPLYING SPECIFIC FUNCTIONS TO A FIELD

3. SELECTING A SPECIFIC NUMBER OF RECORDS, OR A SPECIFIC NUMBER OF DISTINCT VALUES

4. ORDERING RECORDS BASED ON SOME SPECIFIC COLUMN

5. FILTERING RECORDS BASED ON A CONDITION

6. GROUPING/AGGREGATING DATA BASED ON SPECIFIC COLUMNS

7. JOINING DATA OF ONE RELATION WITH ANOTHER RELATION

8. TRANSFORMING NESTED DATA IN A COLUMN TO MULTIPLE ROWS

9. SAMPLING RECORDS

1. SELECTING SPECIFIC FIELDS FROM THE RELATION

2. APPLYING SPECIFIC FUNCTIONS TO A FIELD

3. SELECTING A SPECIFIC NUMBER OF RECORDS, OR A SPECIFIC NUMBER OF DISTINCT VALUES

4. ORDERING RECORDS BASED ON SOME SPECIFIC COLUMN

5. FILTERING RECORDS BASED ON A CONDITION

6. GROUPING/AGGREGATING DATA BASED ON SPECIFIC COLUMNS

7. JOINING DATA OF ONE RELATION WITH ANOTHER RELATION

8. TRANSFORMING NESTED DATA IN A COLUMN TO MULTIPLE ROWS

9. SAMPLING RECORDS

7. JOINING DATA OF ONE RELATION WITH ANOTHER RELATION

JOINING TWO DATA SETS IS AT THE HEART OF DATA TRANSFORMATIONS

7. JOINING DATA OF ONE RELATION WITH ANOTHER RELATION

JOINING TWO DATA SETS IS AT THE HEART OF DATA TRANSFORMATIONS

'JOIN BY'

JOINING TWO DATA SETS IS AT THE HEART OF DATA TRANSFORMATIONS

'JOIN BY'

LIKE HIVE, PIG PROVIDES SUPPORT ONLY FOR EQUI-JOINs

'JOIN BY'

LIKE HIVE, PIG PROVIDES SUPPORT ONLY FOR EQUI-JOINs

LET'S JOIN A COUPLE OF DATASETS

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S
OD01	1	1	450	2016-01-24	3	Apple iPhone 6
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 6

Customer_ID	First_Name	Second_Name	Contact_No	Created_Date	Email_ID
1	Tracy	Stokley	990000112345	2001-01-01	Tracy Stokley1@gmail.
2	Ken	Grigg	990000112346	2001-01-01	Ken Grigg2@gmail.co
3	Jacques	Sandoz	990000112347	2001-01-01	Jacques Sandoz3@gma
4	Rickie	Ciampa	990000112348	2001-01-01	Rickie Ciampa4@gmail
5	Andreas	Vicini	990000112349	2001-01-01	Andreas Vicini5@gmail

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-2	1	Apple iPhone 4S
OD01	1	1	450	2016-01-2	2	Apple iPhone 5S
OD01	1	1	450	2016-01-2	3	Apple iPhone 6
OD02	2	2	400	2016-01-2	1	Apple iPhone 4S
OD02	2	2	400	2016-01-2	2	Apple iPhone 5S
OD02	2	2	400	2016-01-2	3	Apple iPhone 6

Customer_ID	First_Name	Second_Name	Contact_No	Created_Date	Email_ID
1	Tracy	Stokley	990000112345	2001-01-01	Tracy Stokley1@gmail.com
2	Ken	Grigg	990000112346	2001-01-01	Ken Grigg2@gmail.com
3	Jacques	Sandoz	990000112347	2001-01-01	Jacques Sandoz3@gmail.com
4	Rickie	Ciampa	990000112348	2001-01-01	Rickie Ciampa4@gmail.com
5	Andreas	Vicini	990000112349	2001-01-01	Andreas Vicini5@gmail.com

```
customers = load 'customer_data.csv' using PigStorage(',') as
(Customer_ID:int,First_Name:chararray,Second_Name:chararray,Contact
_No:Long,Created_Date:chararray,Email_ID:chararray);

orders = load 'order_data.csv' using PigStorage(',') as
(Order_ID:chararray,Customer_ID:int,Order_Quantity:int,Order_Value:
float,Order_Date:chararray, Product_ID:int,
Product_Name:chararray);
```

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-2	1	Apple iPhone 4S
OD01	1	1	450	2016-01-2	2	Apple iPhone 5S
OD01	1	1	450	2016-01-2	3	Apple iPhone 6
OD02	2	2	400	2016-01-2	1	Apple iPhone 4S
OD02	2	2	400	2016-01-2	2	Apple iPhone 5S
OD02	2	2	400	2016-01-2	3	Apple iPhone 6

Customer_ID	First_Name	Second_Name	Contact_No	Created_Date	Email_ID
1	Tracy	Stokley	990000112345	2001-01-01	Tracy Stokley1@gmail.com
2	Ken	Grigg	990000112346	2001-01-01	Ken Grigg2@gmail.com
3	Jacques	Sandoz	990000112347	2001-01-01	Jacques Sandoz3@gmail.com
4	Rickie	Ciampa	990000112348	2001-01-01	Rickie Ciampa4@gmail.com
5	Andreas	Vicini	990000112349	2001-01-01	Andreas Vicini5@gmail.com

LET US ASSUME WE WANT TO JOIN THEM ON CUSTOMER_ID

THE COMMAND TO DO THAT WILL BE

```
grunt> joined= join customers by Customer_ID,orders by
Customer_ID;
```

```
grunt> joined= join customers by Customer_ID,orders by
Customer_ID;
```

THE OUTPUT IS A JOIN OF THE TWO TABLES on Customer_ID

```
(1,Tracy ,Stokley,990000112345,2001-01-01,Tracy Stokley1@gmail.com,OD01,1,1,450.0,2016-01-24,1,Apple iPhone 4S)
(1,Tracy ,Stokley,990000112345,2001-01-01,Tracy Stokley1@gmail.com,OD01,1,1,450.0,2016-01-24,2,Apple iPhone 5S)
(1,Tracy ,Stokley,990000112345,2001-01-01,Tracy Stokley1@gmail.com,OD01,1,1,450.0,2016-01-24,3,Apple iPhone 6)
(2,Ken ,Grigg,990000112346,2001-01-01,Ken Grigg2@gmail.com,OD03,2,1,350.0,2016-01-25,2,Apple iPhone 5S)
(2,Ken ,Grigg,990000112346,2001-01-01,Ken Grigg2@gmail.com,OD03,2,1,350.0,2016-01-25,3,Apple iPhone 6)
(2,Ken ,Grigg,990000112346,2001-01-01,Ken Grigg2@gmail.com,OD03,2,1,350.0,2016-01-25,1,Apple iPhone 4S)
(2,Ken ,Grigg,990000112346,2001-01-01,Ken Grigg2@gmail.com,OD02,2,2,400.0,2016-01-24,3,Apple iPhone 6)
(2,Ken ,Grigg,990000112346,2001-01-01,Ken Grigg2@gmail.com,OD02,2,2,400.0,2016-01-24,2,Apple iPhone 5S)
(2,Ken ,Grigg,990000112346,2001-01-01,Ken Grigg2@gmail.com,OD02,2,2,400.0,2016-01-24,1,Apple iPhone 4S)
(3,Jacques ,Sandoz,990000112347,2001-01-01,Jacques Sandoz3@gmail.com,OD04,3,1,500.0,2016-01-25,2,Apple iPhone 5S)
(3,Jacques ,Sandoz,990000112347,2001-01-01,Jacques Sandoz3@gmail.com,OD04,3,1,500.0,2016-01-25,1,Apple iPhone 4S)
(3,Jacques ,Sandoz,990000112347,2001-01-01,Jacques Sandoz3@gmail.com,OD04,3,1,500.0,2016-01-27,3,Apple iPhone 6)
(4,Rickie ,Ciampa,990000112348,2001-01-01,Rickie Ciampa4@gmail.com,OD05,4,1,650.0,2016-01-26,1,Apple iPhone 4S)
(4,Rickie ,Ciampa,990000112348,2001-01-01,Rickie Ciampa4@gmail.com,OD05,4,1,650.0,2016-01-27,2,Apple iPhone 5S)
(4,Rickie ,Ciampa,990000112348,2001-01-01,Rickie Ciampa4@gmail.com,OD05,4,1,650.0,2016-01-27,3,Apple iPhone 6)
```

FIELDS FROM CUSTOMERS

FIELDS FROM ORDERS

```
grunt> joined= join customers by Customer_ID,orders by
Customer_ID;
```

RECORDS FOR WHICH NO MATCHES ARE FOUND ARE DROPPED

```
(1,Tracy ,Stokley,990000112345,2001-01-01,Tracy Stokley1@gmail.com,OD01,1,1,450.0,2016-01-24,1,Apple iPhone 4S)
(1,Tracy ,Stokley,990000112345,2001-01-01,Tracy Stokley1@gmail.com,OD01,1,1,450.0,2016-01-24,2,Apple iPhone 5S)
(1,Tracy ,Stokley,990000112345,2001-01-01,Tracy Stokley1@gmail.com,OD01,1,1,450.0,2016-01-24,3,Apple iPhone 6)
(2,Ken ,Grigg,990000112346,2001-01-01,Ken Grigg2@gmail.com,OD03,2,1,350.0,2016-01-25,2,Apple iPhone 5S)
(2,Ken ,Grigg,990000112346,2001-01-01,Ken Grigg2@gmail.com,OD03,2,1,350.0,2016-01-25,3,Apple iPhone 6)
(2,Ken ,Grigg,990000112346,2001-01-01,Ken Grigg2@gmail.com,OD03,2,1,350.0,2016-01-25,1,Apple iPhone 4S)
(2,Ken ,Grigg,990000112346,2001-01-01,Ken Grigg2@gmail.com,OD02,2,2,400.0,2016-01-24,3,Apple iPhone 6)
(2,Ken ,Grigg,990000112346,2001-01-01,Ken Grigg2@gmail.com,OD02,2,2,400.0,2016-01-24,2,Apple iPhone 5S)
(2,Ken ,Grigg,990000112346,2001-01-01,Ken Grigg2@gmail.com,OD02,2,2,400.0,2016-01-24,1,Apple iPhone 4S)
(3,Jacques ,Sandoz,990000112347,2001-01-01,Jacques Sandoz3@gmail.com,OD04,3,1,500.0,2016-01-25,2,Apple iPhone 5S)
(3,Jacques ,Sandoz,990000112347,2001-01-01,Jacques Sandoz3@gmail.com,OD04,3,1,500.0,2016-01-25,1,Apple iPhone 4S)
(3,Jacques ,Sandoz,990000112347,2001-01-01,Jacques Sandoz3@gmail.com,OD04,3,1,500.0,2016-01-27,3,Apple iPhone 6)
(4,Rickie ,Ciampa,990000112348,2001-01-01,Rickie Ciampa4@gmail.com,OD05,4,1,650.0,2016-01-26,1,Apple iPhone 4S)
(4,Rickie ,Ciampa,990000112348,2001-01-01,Rickie Ciampa4@gmail.com,OD05,4,1,650.0,2016-01-27,2,Apple iPhone 5S)
(4,Rickie ,Ciampa,990000112348,2001-01-01,Rickie Ciampa4@gmail.com,OD05,4,1,650.0,2016-01-27,3,Apple iPhone 6)
```

FIELDS FROM CUSTOMERS

FIELDS FROM ORDERS

LET US ASSUME WE WANT TO JOIN THEM ON CUSTOMER_ID

THE COMMAND TO DO THAT WILL BE

```
grunt> joined= join customers by Customer_ID,orders by
Customer_ID;
```

WHAT IS THE SCHEMA OF 'joined'?

WHAT IS THE SCHEMA OF THE 'joined'?

```
grunt> joined= join customers by Customer_ID,orders by
Customer_ID;
grunt> describe joined;
```

```
grunt> describe joined;
joined: {customers::Customer_ID: int,customers::First_Name: chararray,customers::Second_Name: chararray,customers::Contac
hararray,customers::Email_ID: chararray,orders::Order_ID: chararray,orders::Customer_ID: int,orders::Order_Quantity: int,
der_Date: chararray,orders::Product_ID: int,orders::Product_Name: chararray}
```

WHAT IS THE SCHEMA OF THE 'joined'?

```
grunt> joined= join customers by Customer_ID,orders by
Customer_ID;
grunt> describe joined;
```

```
grunt> describe joined;
joined: {customers::Customer_ID: int,customers::First_Name: chararra
hararray,customers::Email_ID: chararray,orders::Order_ID: chararray,
der_Date: chararray,orders::Product_ID: int,orders::Product_Name: ch
```

YOU CAN SEE THAT ALL FIELDS OF **customers** RELATION ARE REPRESENTED AS

customers::field_name

```
grunt> joined= join customers by Customer_ID,orders by
Customer_ID;
grunt> describe joined;
```

```
y,customers::Second_Name: chararray,customers::Contact_No: long,customers::Created_Date: c
orders::Customer_ID: int,orders::Order_Quantity: int,orders::Order_Value: float,orders::Or
ararray}
```

ALL FIELDS OF **ORDERS** RELATION ARE REPRESENTED AS

orders::field_name

```
grunt> describe joined;
joined: {customers::Customer_ID: int,customers::First_Name: chararray,customers::Second_Name: chararray,customers::Contact_No: long,customers::Created_Date: c
hararray,customers::Email_ID: chararray,orders::Order_ID: chararray,orders::Customer_ID: int,orders::Order_Quantity: int,orders::Order_Value: float,orders::Or
der_Date: chararray,orders::Product_ID: int,orders::Product_Name: chararray}
```

customers::field_name

orders::field_name

USING THE FOREACH COMMAND AND KNOWLEDGE OF JOINED SCHEMA, WE CAN WRITE

```
grunt> temp = foreach joined generate customers::Customer_ID,
customers::Contact_No,orders::Order_ID,orders::Product_ID;
```

customers::field_name

orders::field_name

```
grunt> temp = foreach joined generate customers::Customer_ID,
customers::Contact_No,orders::Order_ID,orders::Product_ID;
```

THE OUTPUT IS

```
(1,990000112345,OD01,1)
(1,990000112345,OD01,2)
(1,990000112345,OD01,3)
(2,990000112346,OD03,2)
(2,990000112346,OD03,3)
(2,990000112346,OD03,1)
(2,990000112346,OD02,3)
(2,990000112346,OD02,2)
(2,990000112346,OD02,1)
(3,990000112347,OD04,2)
(3,990000112347,OD04,1)
(3,990000112347,OD04,3)
(4,990000112348,OD05,1)
(4,990000112348,OD05,2)
(4,990000112348,OD05,3)
```

customers::field_name

orders::field_name

```
grunt> temp = foreach joined generate customers::Customer_ID,
customers::Contact_No,orders::Order_ID,orders::Product_ID;
```

USING THIS WE CAN CHOOSE TO SHOW ONLY SPECIFIC FIELDS FROM EACH RELATION AFTER JOIN

```
grunt> temp = foreach joined generate customers::Customer_ID,
customers::Contact_No,orders::Order_ID,orders::Product_ID;
```

HOWEVER

```
grunt> temp = foreach joined generate customers::Customer_ID,
Contact_No,Order_ID,Product_ID;
```

IS ALSO CORRECT AND WILL GIVE THE SAME RESULT

WITHOUT :: OPERATIOR

```
grunt> temp = foreach joined generate customers::Customer_ID,
Contact_No,Order_ID,Product_ID;
```

IS ALSO CORRECT AND WILL GIVE THE SAME RESULT

customers:: and orders:: prefix needs to be used only when the field name is not unique

i.e FIELD NAME IS COMMON IN BOTH TABLES

`customers::` and `orders::` - prefixes need to be used only when the field name is not unique

PIG ALSO SUPPORTS

LEFT OUTER JOIN

RIGHT OUTER JOIN

FULL OUTER JOIN

SELF JOIN

CROSS JOIN

OUTER JOIN

THE RESULT INCLUDES ALL ROWS FROM THE FIRST RELATION ("LEFT OUTER JOIN"), OR THE SECOND RELATION ("RIGHT OUTER JOIN"), OR BOTH ("FULL OUTER JOIN"),

WITH NULLS BEING FILLED IN FOR THE MISSING FIELDS

OUTER JOIN

THE RESULT INCLUDES ALL ROWS FROM THE FIRST RELATION ("LEFT OUTER JOIN"), OR THE SECOND RELATION ("RIGHT OUTER JOIN"), OR BOTH ("FULL OUTER JOIN"),

WITH NULLS BEING FILLED IN FOR THE MISSING FIELDS

OUTER JOIN

THE RESULT INCLUDES ALL ROWS FROM THE FIRST RELATION ("LEFT OUTER JOIN"), OR THE SECOND RELATION ("RIGHT OUTER JOIN"), OR BOTH ("FULL OUTER JOIN"),

WITH NULLS BEING FILLED IN FOR THE MISSING FIELDS

OUTER JOIN

THE RESULT INCLUDES ALL ROWS FROM THE FIRST RELATION ("LEFT OUTER JOIN"), OR THE SECOND RELATION ("RIGHT OUTER JOIN"), OR BOTH ("FULL OUTER JOIN"),

WITH NULLS BEING FILLED IN FOR THE MISSING FIELDS

OUTER JOIN

THE RESULT INCLUDES ALL ROWS FROM THE FIRST RELATION ("LEFT OUTER JOIN"), OR THE SECOND RELATION ("RIGHT OUTER JOIN"), OR BOTH ("FULL OUTER JOIN"),

WITH NULLS BEING FILLED IN FOR THE MISSING FIELDS

OUTER JOIN

THE RESULT INCLUDES ALL ROWS FROM THE FIRST RELATION ("LEFT OUTER JOIN"), OR THE SECOND RELATION ("RIGHT OUTER JOIN"), OR BOTH ("FULL OUTER JOIN"),

WITH NULLS BEING FILLED IN FOR THE MISSING FIELDS

OUTER JOIN

OUTER JOINS ARE ONLY SUPPORTED WHEN PIG KNOWS THE SCHEMA OF THE RELATION FOR WHICH IT MIGHT HAVE TO FILL IN NULLS

PIG ALSO SUPPORTS

LEFT OUTER JOIN

RIGHT OUTER JOIN

FULL OUTER JOIN

SELF JOIN

CROSS JOIN

PIG ALSO SUPPORTS LEFT OUTER JOIN

LEFT OUTER JOIN IS DONE USING LEFT OUTER COMMAND

```
grunt> joined= join customers by Customer_ID left outer,orders
by Customer_ID;
```

LEFT OUTER JOIN

```
grunt> joined= join customers by Customer_ID left
outer,orders by Customer_ID;
```

PIG SHOULD KNOW THE SCHEMA OF `orders` Relation

BECAUSE NULLS WILL BE FILLED ON RIGHT SIDE

PIG ALSO SUPPORTS

LEFT OUTER JOIN

RIGHT OUTER JOIN

FULL OUTER JOIN

SELF JOIN

CROSS JOIN

PIG ALSO SUPPORTS RIGHT OUTER JOIN

RIGHT OUTER JOIN IS DONE USING RIGHT OUTER COMMAND

```
grunt> joined= join customers by Customer_ID right outer,orders
by Customer_ID;
```

PIG ALSO SUPPORTS RIGHT OUTER JOIN

```
grunt> joined= join customers by Customer_ID right outer,orders
by Customer_ID;
```

PIG SHOULD KNOW THE SCHEMA OF `customers` Relation

BECAUSE NULLS WILL BE FILLED ON LEFT SIDE

PIG ALSO SUPPORTS

LEFT OUTER JOIN

RIGHT OUTER JOIN

FULL OUTER JOIN

SELF JOIN

CROSS JOIN

PIG ALSO SUPPORTS FULL OUTER JOIN

FULL OUTER JOIN IS DONE USING FULL OUTER COMMAND

```
grunt> joined= join customers by Customer_ID full outer,orders
by Customer_ID;
```

PIG ALSO SUPPORTS FULL OUTER JOIN

```
grunt> joined= join customers by Customer_ID full outer,orders
by Customer_ID;
```

PIG SHOULD KNOW THE SCHEMA OF BOTH orders and customers Relation

PIG ALSO SUPPORTS

LEFT OUTER JOIN

RIGHT OUTER JOIN

FULL OUTER JOIN

SELF JOIN

CROSS JOIN

PIG ALSO SUPPORTS SELF JOIN

SELF JOINS ARE SUPPORTED BUT THE DATA WILL NEED TO BE LOADED TWICE:-(

```
grunt> orders1 = load 'order_data.csv' using PigStorage(',') as
(Order_ID:chararray,Customer_ID:int,Order_Quantity:int);

grunt> orders2 = load 'order_data.csv' using PigStorage(',') as
(Order_ID:chararray,Customer_ID:int,Order_Quantity:int);
```

SAME TABLE LOADED TWICE

SELF JOIN

SAME TABLE LOADED TWICE

```
grunt> orders1 = load 'order_data.csv' using PigStorage(',') as
(Order_ID:chararray,Customer_ID:int,Order_Quantity:int);

grunt> orders2 = load 'order_data.csv' using PigStorage(',') as
(Order_ID:chararray,Customer_ID:int,Order_Quantity:int);
```

```
grunt> joined= join orders1 by Customer_ID,orders2 by
Customer_ID;
```

THIS WILL WORK!

SELF JOIN

SAME TABLE LOADED TWICE

```
grunt> orders1 = load 'order_data.csv' using PigStorage(',') as
(Order_ID:chararray,Customer_ID:int,Order_Quantity:int);

grunt> orders2 = load 'order_data.csv' using PigStorage(',') as
(Order_ID:chararray,Customer_ID:int,Order_Quantity:int);
```

```
grunt> joined= join orders by Customer_ID,orders by
Customer_ID;
```

THIS WILL NOT WORK!

SELF JOIN

SAME TABLE LOADED TWICE

```
grunt> joined= join orders by Customer_ID,orders by
Customer_ID;
```

THIS WILL NOT WORK!

BECAUSE THE SAME RELATION `orders` CANNOT BE USED ON BOTH SIDES OF THE JOIN

This is a quirk of Pig

PIG ALSO SUPPORTS

LEFT OUTER JOIN

RIGHT OUTER JOIN

FULL OUTER JOIN

SELF JOIN

CROSS JOIN

PIG ALSO SUPPORTS CROSS JOIN

CROSS JOIN IS BASICALLY CARTESIAN PRODUCT OF TWO RELATIONS

IF RELATION A HAS N ROWS AND B HAS M ROWS, CROSS JOIN OF A AND B WILL HAVE (NXM) ROWS

PIG ALSO SUPPORTS CROSS JOIN

IF RELATION A HAS N ROWS AND B HAS M ROWS,
CROSS JOIN OF A AND B WILL HAVE (NXM) ROWS

CROSS JOIN GENERATES A LOT OF DATA

BE CAREFUL USING IT!

PIG ALSO SUPPORTS CROSS JOIN

CROSS JOIN IS DONE BY USING CROSS COMMAND

```
grunt> crossed= cross customers,orders;
```

CROSS CAN BE VERY USEFUL IF WE WANT TO DO NON EQUI-JOINS

CROSS CAN BE VERY USEFUL IF WE WANT TO DO NON EQUI-JOINS

LET US SAY WE WANT TO FIND THE INVALID ORDERS IN ORDER DATA

INVALID ORDERS ARE THOSE ORDERS WHERE CUSTOMER'S ACCOUNT CREATED DATE IS GREATER THAN ORDER DATE

CROSS CAN BE VERY USEFUL IF WE WANT TO NON EQUI-JOINS

INVALID ORDERS ARE THOSE ORDERS WHERE CUSTOMER'S ACCOUNT CREATED DATE IS GREATER THAN ORDER DATE

```
grunt> crossed= cross customers,orders;

grunt>invalid_orders = filter crossed by customers::Created_Date >=
orders::Order_Date;

grunt>invalid_orders_list = foreach invalid_orders generate Order_ID;
```

ORDER_ID	CUSTOMER_ID	ORDER_QUAN	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-2	1	Apple iPhone 4S
OD01	1	1	450	2016-01-2	2	Apple iPhone 5S
OD01	1	1	450	2016-01-2	3	Apple iPhone 6
OD02	2	2	400	2016-01-2	1	Apple iPhone 4S
OD02	2	2	400	2016-01-2	2	Apple iPhone 5S
OD02	2	2	400	2016-01-2	3	Apple iPhone 6

Customer_ID	First_Name	Second_Name	Contact_No	Created_Date	Email_ID
1	Tracy	Stokley	990000112345	2001-01-01	Tracy Stokley1@gmail.com
2	Ken	Grigg	990000112346	2001-01-01	Ken Grigg2@gmail.com
3	Jacques	Sandoz	990000112347	2001-01-01	Jacques Sandoz3@gmail.com
4	Rickie	Ciampa	990000112348	2001-01-01	Rickie Ciampa4@gmail.com

CROSS CAN BE VERY USEFUL IF WE WANT TO NON EQUI-JOINS

INVALID ORDERS ARE THOSE ORDERS WHERE CUSTOMER'S ACCOUNT CREATED DATE IS GREATER THAN ORDER DATE

```
grunt> crossed= cross customers,orders;

grunt>invalid_orders = filter crossed by customers::Created_Date >=
orders::Order_Date;

grunt>invalid_orderslist = foreach invalid_orders generate Order_ID;
```

THIS PREDICATE WILL HELP US IDENTIFY THE INVALID ORDERS

ORDER_ID	CUSTOMER_ID	ORDER_QUANTI	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD01	1	1	450	2016-01-24	1	Apple iPhone 4S
OD01	1	1	450	2016-01-24	2	Apple iPhone 5S

Customer_ID	First_Name	Second_Name	Contact_No	Created_Date	Email_ID
1	Tracy	Stokley	990000112345	2001-01-01	Tracy Stokley1@gmail.c

in addition to the
JOIN OPERATIONS

PIG ALSO SUPPORTS UNION

PIG ALSO SUPPORTS UNION

UNION CONCATENATES TWO DATASETS

UNION CONCATENATES TWO DATASETS

UNION DOES NOT REMOVE DUPLICATE RECORDS

UNION CONCATENATES TWO DATASETS

UNION DOES NOT REMOVE DUPLICATE RECORDS

UNION IN PIG IS LIKE UNION ALL IN SQL

UNION CONCATENATES TWO DATASETS

UNION DOES NOT REMOVE DUPLICATE RECORDS

UNION IN PIG IS LIKE UNION ALL IN SQL

```
grunt> file1 = load '/user/navdeepsingh/order_data/order_data/
date-1.csv';

grunt> file2 = load '/user/navdeepsingh/order_data/order_data/
date-2.csv';

grunt> total_data = union file1,file2;
```

BOTH FILES HAVE SAME SCHEMA OF ORDERS TABLE

```
grunt> file1 = load '/user/navdeepsingh/order_data/order_data/
date-1.csv';

grunt> file2 = load '/user/navdeepsingh/order_data/order_data/
date-2.csv';

grunt> total_data = union file1,file2;
```

UNLIKE SQL, UNION IN PIG DOESN'T NEED BOTH THE RELATIONS TO HAVE THE SAME SCHEMA

UNION OPERATION BASICALLY MEANS CONCATENATING TWO DATASETS

UNLIKE SQL, UNION IN PIG DOESN'T NEED BOTH THE RELATIONS TO HAVE THE SAME SCHEMA

IF SCHEMA OF ONE RELATION CAN BE CAST INTO SCHEMA OF OTHER RELATION, THE UNION WILL HAVE THAT SCHEMA

IF SCHEMA OF ONE RELATION CAN BE CAST INTO SCHEMA OF OTHER RELATION, THE UNION WILL HAVE THAT SCHEMA

IF SCHEMA OF ONE RELATION CANNOT BE CAST INTO SCHEMA OF OTHER RELATION, THE UNION WILL HAVE NO SCHEMA

IF SCHEMA OF ONE RELATION CANNOT BE CAST INTO SCHEMA OF OTHER RELATION, THE UNION WILL HAVE NO SCHEMA

Oh, and BTW -

IN CASE NEW COLUMNS ARE ADDED -

UNION ONSCHEMA

UNION ONSCHEMA

```
grunt> file1 = load '/user/navdeepsingh/sales/sales1' as
(Order_ID:Chararray,Customer_ID:int,Order_Quantity:Int,Order_Value:float);

grunt> file2 = load '/user/navdeepsingh/sales/sales2' as
(Order_ID:Chararray,Customer_ID:int,Order_Datetime:Chararray,Order_Value:f
loat);

grunt> total_data = union onschema file1,file2;

grunt> describe total_data;

total_data:
{Order_ID: bytearray,Customer_ID: int,Order_Quantity: int,Order_Value:
float,Order_Datetime: chararray}
```

UNION ONSCHEMA

```
grunt> file1 = load '/user/navdeepsingh/sales/sales1' as
(Order_ID:Chararray,Customer_ID:int,Order_Quantity:Int,Order_Value:float);

grunt> file2 = load '/user/navdeepsingh/sales/sales2' as
(Order_ID:Chararray,Customer_ID:int,Order_Datetime:Chararray,Order_Value:f
loat);

grunt> total_data = union onschema file1,file2;

grunt> describe total_data;

total_data:
{Order_ID: bytearray,Customer_ID: int,Order_Quantity: int,Order_Value:
float,Order_Datetime: chararray}
```

NOT PRESENT IN FILE 2

UNION ONSCHEMA

```
grunt> file1 = load '/user/navdeepsingh/sales/sales1' as
(Order_ID:Chararray,Customer_ID:int,Order_Quantity:Int,Order_Value:float);

grunt> file2 = load '/user/navdeepsingh/sales/sales2' as
(Order_ID:Chararray,Customer_ID:int,Order_Datetime:Chararray,Order_Value:f
loat);

grunt> total_data = union onschema file1,file2;

grunt> describe total_data;

total_data:
{Order_ID: bytearray,Customer_ID: int,Order_Quantity: int,Order_Value:
float,Order_Datetime: chararray}
```

NOT PRESENT IN FILE 1

UNION ONSCHEMA

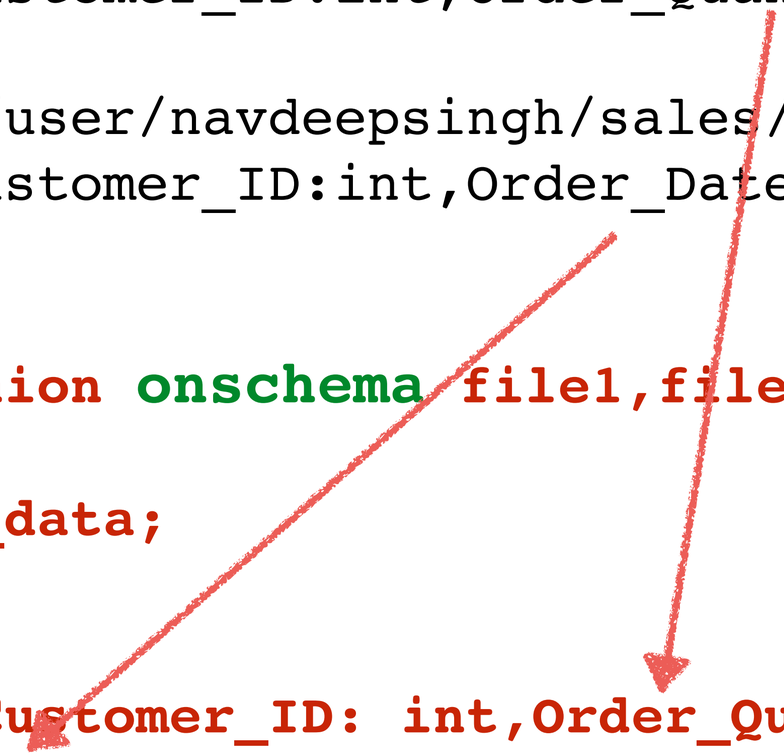
```
grunt> file1 = load '/user/navdeepsingh/sales/sales1' as
(Order_ID:Chararray,Customer_ID:int,Order_Quantity:Int,Order_Value:float);

grunt> file2 = load '/user/navdeepsingh/sales/sales2' as
(Order_ID:Chararray,Customer_ID:int,Order_Datetime:Chararray,Order_Value:f
loat);

grunt> total_data = union onschema file1,file2;

grunt> describe total_data;

total_data:
{Order_ID: bytearray,Customer_ID: int,Order_Quantity: int,Order_Value:
float,Order_Datetime: chararray}
```



SHARED SCHEMA CAN BE PRODUCED BY ADDING FIELDS AND IMPLICIT CASTING

USING ONSCHEMA

UNION ONSCHEMA

IF NO SHARED SCHEMA CAN BE PRODUCED BY ADDING FIELDS AND CASTING

AN ERROR WILL BE RETURNED

1. SELECTING SPECIFIC FIELDS FROM THE RELATION

2. APPLYING SPECIFIC FUNCTIONS TO A FIELD

3. SELECTING A SPECIFIC NUMBER OF RECORDS, OR A SPECIFIC NUMBER OF DISTINCT VALUES

4. ORDERING RECORDS BASED ON SOME SPECIFIC COLUMN

5. FILTERING RECORDS BASED ON A CONDITION

6. GROUPING/AGGREGATING DATA BASED ON SPECIFIC COLUMNS

7. JOINING DATA OF ONE RELATION WITH ANOTHER RELATION

8. TRANSFORMING NESTED DATA IN A COLUMN TO MULTIPLE ROWS

9. SAMPLING RECORDS