1. SELECTING SPECIFIC FIELDS FROM THE RELATION

2. APPLYING SPECIFIC FUNCTIONS TO A FIELD

3. SELECTING DISTINCT OR FIXED NUMBER OF RECORDS

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 DISTINCT OR FIXED NUMBER OF RECORDS

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

WE WILL SEE HOW WE CAN EXECUTE THESE OPERATIONS IN PIG ONE BY ONE

8. TRANSFORMING NESTED DATA IN A COLUMN TO MULTIPLE ROWS

THIS IS DONE VIA THE FLATTEN COMMAND

8. TRANSFORMING NESTED DATA IN A COLUMN TO MULTIPLE ROWS

THIS IS DONE VIA THE FLATTEN COMMAND

FLATTEN IS A MODIFIER THAT WORKS WITH FOREACH

8. TRANSFORMING NESTED DATA IN A COLUMN TO MULTIPLE ROWS

FLATTEN IS A MODIFIER THAT WORKS WITH FOREACH

FLATTEN operates on the BAG datatype

FLATTEN operates on the BAG datatype

FLATTEN PRODUCES A CROSS PRODUCT OF RECORDS IN THE BAG WITH ALL THE OTHER EXPRESSIONS IN THE GENERATE STATEMENT

FLATTEN operates on the BAG datatype

FLATTEN PRODUCES A CROSS PRODUCT OF RECORDS IN THE BAG WITH ALL THE OTHER EXPRESSIONS IN THE GENERATE STATEMENT

FLATTEN operates on the BAG datatype

FLATTEN PRODUCES A CROSS PRODUCT OF RECORDS IN THE BAG WITH ALL THE OTHER EXPRESSIONS IN THE GENERATE STATEMENT

FLATTEN operates on the BAG datatype

FLATTEN PRODUCES A CROSS PRODUCT OF RECORDS IN THE BAG WITH ALL THE OTHER EXPRESSIONS IN THE GENERATE STATEMENT

FLATTEN PRODUCES A CROSS PRODUCT OF RECORDS IN THE BAG WITH ALL THE OTHER EXPRESSIONS IN THE GENERATE STATEMENT

customer_product

CUSTOMER_ID	PRODUCTS_BOUGHT
1	(Apple iPhone 4S),(Apple iPhone 5S),(Apple iPhone 6)
2	(Apple iPhone 5S),(Apple iPhone 6),(Apple iPhone 4S)
3	(Apple iPhone 5S),(Apple iPhone 4S),(Apple iPhone 6)
4	(Apple iPhone 4S),(Apple iPhone 5S),(Apple iPhone 6)}

THIS IS A BAG

FLATTEN PRODUCES A CROSS PRODUCT OF RECORDS IN THE BAG WITH ALL THE OTHER EXPRESSIONS IN THE GENERATE STATEMENT

THIS IS A BAG

CUSTOMER_ID	PRODUCTS_BOUGHT
1	(Apple iPhone 4S),(Apple iPhone 5S),(Apple iPhone 6)
2	(Apple iPhone 5S),(Apple iPhone 6),(Apple iPhone 4S)
3	(Apple iPhone 5S),(Apple iPhone 4S),(Apple iPhone 6)
4	(Apple iPhone 4S),(Apple iPhone 5S),(Apple iPhone 6)}

`customer_product`

FLATTEN this on column

`products_bought`

FLATTEN PRODUCES A CROSS PRODUCT OF RECORDS IN THE BAG WITH ALL THE OTHER EXPRESSIONS IN THE GENERATE STATEMENT

FLATTEN this on column

products_bought

CUSTOMER_ID	PRODUCTS_BOUGHT
1	(Apple iPhone 4S),(Apple iPhone 5S),(Apple iPhone 6)
2	(Apple iPhone 5S),(Apple iPhone 6),(Apple iPhone 4S)
3	(Apple iPhone 5S),(Apple iPhone 4S),(Apple iPhone 6)
4	(Apple iPhone 4S),(Apple iPhone 5S),(Apple iPhone 6)}

customer_product

CUSTOMER_ID	PRODUCTS
1	Apple iPhone 4S
1	Apple iPhone 5S
1	Apple iPhone 6
2	Apple iPhone 5S
2	Apple iPhone 6
2	Apple iPhone 4S
3	Apple iPhone 5S
3	Apple iPhone 4S
3	Apple iPhone 6
4	Apple iPhone 4S
4	Apple iPhone 5S
4	Apple iPhone 6

FLATTEN this on column products_bought

customer_product

CUSTOMER_ID	PRODUCTS_BOUGHT
1	(Apple iPhone 4S),(Apple iPhone 5S),(Apple iPhone 6)
2	(Apple iPhone 5S),(Apple iPhone 6),(Apple iPhone 4S)
3	(Apple iPhone 5S),(Apple iPhone 4S),(Apple iPhone 6)
4	(Apple iPhone 4S),(Apple iPhone 5S),(Apple iPhone 6)}

CUSTOMER_ID	PRODUCTS
1	Apple iPhone 4S
1	Apple iPhone 5S
1	Apple iPhone 6
2	Apple iPhone 5S
2	Apple iPhone 6
2	Apple iPhone 4S
3	Apple iPhone 5S
3	Apple iPhone 4S
3	Apple iPhone 6
4	Apple iPhone 4S
4	Apple iPhone 5S
4	Apple iPhone 6

FLATTEN PRODUCES A CROSS PRODUCT OF RECORDS IN THE BAG

{(Apple iPhone 4S),
(Apple iPhone 5S),
(Apple iPhone 6)}

X

Customer_ID =1

Products_Bought is a bag

```
grunt> flattened_customer_product =  foreach customer_product generate
Customer_ID, flatten(Products_Bought) as Products;

grunt> dump flattened_customer_product;
```

customer_product

CUSTOMER_ID	PRODUCTS_BOUGHT
1	(Apple iPhone 4S),(Apple iPhone 5S),(Apple iPhone 6)
2	(Apple iPhone 5S),(Apple iPhone 6),(Apple iPhone 4S)
3	(Apple iPhone 5S),(Apple iPhone 4S),(Apple iPhone 6)
4	(Apple iPhone 4S),(Apple iPhone 5S),(Apple iPhone 6)}

flattened_customer_product

CUSTOMER_ID	PRODUCTS
1	Apple iPhone 4S
1	Apple iPhone 5S
1	Apple iPhone 6
2	Apple iPhone 5S
2	Apple iPhone 6
2	Apple iPhone 4S
3	Apple iPhone 5S
3	Apple iPhone 4S
3	Apple iPhone 6

Oh, and BTW, we got the customer_product table from orders

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

grunt> customer_product = foreach group generate group as
Customer_ID,orders.Product_Name as Products_Bought;
```

CUSTOMER_ID	PRODUCTS_BOUGHT
1	(Apple iPhone 4S),(Apple iPhone 5S),(Apple iPhone 6)
2	(Apple iPhone 5S),(Apple iPhone 6),(Apple iPhone 4S)
3	(Apple iPhone 5S),(Apple iPhone 4S),(Apple iPhone 6)
4	(Apple iPhone 4S),(Apple iPhone 5S),(Apple iPhone 6)}

IMPORTANT!

FLATTEN

WHEN AN EMPTY BAG IS FLATTENED, WE GET A NULL!

FLATTEN PRODUCES A CROSS PRODUCT

FLATTEN

WHEN AN EMPTY BAG IS FLATTENED, WE GET A NULL!

FLATTEN PRODUCES A CROSS PRODUCT

0 X SOME THING IS 0

0 (EMPTY BAG) X SOME THING IS 0 (NULL)

CO-GROUP

IS A MORE GENERAL FORM OF GROUP

CO-GROUP

IS A MORE GENERAL FORM OF GROUP

CO-GROUP RETURNS RECORDS THAT HAVE A KEY AND A BAG SIMILAR TO GROUP

CO-GROUP

IS A MORE GENERAL FORM OF GROUP

GROUP WORKS WITH ONLY 1 RELATION

CO-GROUP WORKS WITH MANY RELATIONS

CO-GROUP

GROUP WORKS WITH ONLY 1 RELATION

CO-GROUP WORKS WITH MANY RELATIONS

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.c
3	Jacques	Sandoz	990000112347	2001-01-01	Jacques Sandoz3@gma
4	Rickie	Ciampa	990000112348	2001-01-01	Rickie Ciampa4@gmai
5	Andreas	Vicini	990000112349	2001-01-01	Andreas Vicini5@gmail

LET US ASSUME WE HAVE TWO RELATIONS

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@g
2	Ken	Grigg	990000112346	2001-01-01	Ken Grigg2@gm
3	Jacques	Sandoz	990000112347	2001-01-01	Jacques Sandoz3@
4	Rickie	Ciampa	990000112348	2001-01-01	Rickie Ciampa4@
5	Andreas	Vicini	990000112349	2001-01-01	Andreas Vicini5@g

```
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);

cogrouped=cogroup customers by Customer_ID, 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

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

```
cogrouped=cogroup customers by Customer_ID, orders by Customer_ID;
```

```
(1,{(1,Tracy ,Stokley,990000112345,2001-01-01,Tracy Stokley1@gmail.com)},{(OD01,1,1,450.0,2016-01-24,1,Apple iPhone 4S),(OD01,1,1,450.0,2016-01-24,2,Apple iPh
one 5S),(OD01,1,1,450.0,2016-01-24,3,Apple iPhone 6)})
(2,{(2,Ken ,Grigg,990000112346,2001-01-01,Ken Grigg2@gmail.com)},{(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),(OD02,2,2,400.0,2016-0
1-24,1,Apple iPhone 4S)})
(3,{(3,Jacques ,Sandoz,990000112347,2001-01-01,Jacques Sandoz3@gmail.com)},{(OD04,3,1,500.0,2016-01-25,2,Apple iPhone 5S),(OD04,3,1,500.0,2016-01-25,1,Apple i
Phone 4S),(OD04,3,1,500.0,2016-01-27,3,Apple iPhone 6)})
(4,{(4,Rickie ,Ciampa,990000112348,2001-01-01,Rickie Ciampa4@gmail.com)},{(OD05,4,1,650.0,2016-01-26,1,Apple iPhone 4S),(OD05,4,1,650.0,2016-01-27,2,Apple iPh
one 5S),(OD05,4,1,650.0,2016-01-27,3,Apple iPhone 6)})
```

RECORDS FROM BOTH THE TABLES ARE PRESENT IN SAME RECORD!

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

```
cogrouped=cogroup customers by Customer_ID, orders by Customer_ID;
```

ONE BAG FOR THE RECORD FROM THE CUSTOMER RELATION

```
(1,{(1,Tracy ,Stokley,990000112345,2001-01-01,Tracy Stokley1@gmail.com)},{(
one 5S),(OD01,1,1,450.0,2016-01-24,3,Apple iPhone 6)})
(2,{(2,Ken ,Grigg,990000112346,2001-01-01,Ken Grigg2@gmail.com)},{(OD03,2,1,
OD03,2,1,350.0,2016-01-25,1,Apple iPhone 4S),(OD02,2,2,400.0,2016-01-24,3,Ap
1-24,1,Apple iPhone 4S)})
(3,{(3,Jacques ,Sandoz,990000112347,2001-01-01,Jacques Sandoz3@gmail.com)},
```

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
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

```
cogrouped=cogroup customers by Customer_ID, orders by Customer_ID;
```

ONE BAG FOR THE RECORDS FROM THE ORDERS RELATION

```
{(OD01,1,1,450.0,2016-01-24,1,Apple iPhone 4S),(OD01,1,1,450.0,2016-01-24,2,Apple iPh

,1,350.0,2016-01-25,2,Apple iPhone 5S),(OD03,2,1,350.0,2016-01-25,3,Apple iPhone 6),(
,Apple iPhone 6),(OD02,2,2,400.0,2016-01-24,2,Apple iPhone 5S),(OD02,2,2,400.0,2016-0

},{(OD04,3,1,500.0,2016-01-25,2,Apple iPhone 5S),(OD04,3,1,500.0,2016-01-25,1,Apple i

{(OD05,4,1,650.0,2016-01-26,1,Apple iPhone 4S),(OD05,4,1,650.0,2016-01-27,2,Apple iPh
```

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
5	Andreas	Vicini	990000112349	2001-01-01	Andreas Vicini5@gmail.com

```
cogrouped=cogroup customers by Customer_ID, orders by Customer_ID;
```

SCHEMA OF cogrouped IS

```
{group: int,customers: {(Customer_ID: int,First_Name:
chararray,Second_Name: chararray,Contact_No:
long,Created_Date: chararray,Email_ID: chararray)},orders:
{(Order_ID: chararray,Customer_ID: int,Order_Quantity:
int,Order_Value: float,Order_Date: chararray,Product_ID:
int,Product_Name: chararray)}}
```

```
{group: int,customers: {(Customer_ID: int,First_Name:
chararray,Second_Name: chararray,Contact_No:
long,Created_Date: chararray,Email_ID: chararray)},orders:
{(Order_ID: chararray,Customer_ID: int,Order_Quantity:
int,Order_Value: float,Order_Date: chararray,Product_ID:
int,Product_Name: chararray)}}
```

THE GROUP IS BY CUSTOMER_ID WHICH IS AN INTEGER

```
cogrouped=cogroup customers by Customer_ID, orders by Customer_ID;
```

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME

Customer_ID	First_Name	Second_Name	Contact_No	Created_Date	Email_ID

```
{group: int,customers: {(Customer_ID: int,First_Name:
chararray,Second_Name: chararray,Contact_No:
long,Created_Date: chararray,Email_ID: chararray)},orders:
{(Order_ID: chararray,Customer_ID: int,Order_Quantity:
int,Order_Value: float,Order_Date: chararray,Product_ID:
int,Product_Name: chararray)}}
```

CUSTOMERS IS A BAG FROM THE CUSTOMERS RELATION

```
cogrouped=cogroup customers by Customer_ID, orders by Customer_ID;
```

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME

Customer_ID	First_Name	Second_Name	Contact_No	Created_Date	Email_ID

```
{group: int,customers: {(Customer_ID: int,First_Name:
chararray,Second_Name: chararray,Contact_No:
long,Created_Date: chararray,Email_ID: chararray)},orders:
{(Order_ID: chararray,Customer_ID: int,Order_Quantity:
int,Order_Value: float,Order_Date: chararray,Product_ID:
int,Product_Name: chararray)}}
```

THE ORDERS BAG COMES FROM THE ORDERS RELATION

```
cogrouped=cogroup customers by Customer_ID, orders by Customer_ID;
```

ORDER_ID	CUSTOMER_ID	ORDER_QUANTITY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME

Customer_ID	First_Name	Second_Name	Contact_No	Created_Date	Email_ID

```
cogrouped=cogroup customers by Customer_ID, orders by Customer_ID;
```

SCHEMA OF cogrouped IS

```
cogrouped: {group: int,customers: {(Customer_ID: int,First_Name: chararray,Second_Name: chararray,Contact_No: long,Created_Date: chararray,Email_ID: chararray
)},orders: {(Order_ID: chararray,Customer_ID: int,Order_Quantity: int,Order_Value: float,Order_Date: chararray,Product_ID: int,Product_Name: chararray)}}
```

THE BAG PART OF cogroup HAS SCHEMA OF BOTH INPUTS

CO-GROUP HELPS EXECUTE SEMI-JOINS

CO-GROUP HELPS EXECUTE SEMI-JOINS

A LEFT SEMI JOIN IS A JOIN THAT RETURNS THE RECORDS ONLY FROM THE LEFT-HAND TABLE

CO-GROUP HELPS EXECUTE SEMI-JOINS

CUSTOMERS IS THE LEFT TABLE

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

ORDERS IS THE RIGHT TABLE

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 ASSUME CUSTOMERS IS THE LEFT TABLE AND ORDERS IS THE RIGHT TABLE

LEFT SEMI JOIN RETURNS THE CUSTOMERS DATA FOR ONLY THOSE CUSTOMERS WHO HAVE PLACED ANY ORDER

CUSTOMERS

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	Jacqu	Sandoz	990000112347	2001-01-01	Jacques

ORDERS

ORDER_ID	CUSTOMER_ID	ORDER_ QUANTI TY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAM
OD01	1	1	450	2016-01-24	1	Apple iPhone 4
OD01	1	1	450	2016-01-24	2	Apple iPhone 5
OD01	1	1	450	2016-01-24	3	Apple iPhone
OD02	2	2	400	2016-01-25	1	Apple iPhone 4

LEFT SEMI JOIN RETURNS THE CUSTOMERS DATA FOR ONLY THOSE CUSTOMERS WHO HAVE PLACED AN ORDER

CUSTOMERS

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

ORDERS

ORDER_ID	CUSTOMER_ID	ORDER_ QUANTI	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAM
OD01	1	1	450	2016-01-24	1	Apple iPhone
OD01	1	1	450	2016-01-24	2	Apple iPhone
OD01	1	1	450	2016-01-24	3	Apple iPhone
OD02	2	2	400	2016-01-25	1	Apple iPhone
OD02	2	2	400	2016-01-25	2	Apple iPhone
OD02	2	2	400	2016-01-25	3	Apple iPhone

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

CO-GROUP HELPS EXECUTE SEMI-JOINS

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

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

ORDER_ID	CUSTOMER_ID	ORDER_ QUANTI	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT
OD01	1	1	450	2016-01-24	1	Apple iPh
OD01	1	1	450	2016-01-24	2	Apple iPh
OD01	1	1	450	2016-01-24	3	Apple iP

CO-GROUP HELPS EXECUTE SEMI-JOINS

```
grunt> cogroupd = cogroup customers by Customer_ID, orders by Customer_ID;

grunt> semijoin = filter cogroupd by not IsEmpty(orders);

grunt> semijoin_dump = foreach semijoin generate flatten(customers);

grunt> dump semijoin_dump;
```

Customer_ID	First_Nam	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

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

ORDER_ID	CUSTOMER_ID	ORDER_ QUANTI	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT
OD01	1	1	450	2016-01-24	1	Apple iPh
OD01	1	1	450	2016-01-24	2	Apple iPh
OD01	1	1	450	2016-01-24	3	Apple iP

CO-GROUP HELPS EXECUTE SEMI-JOINS

```
grunt> cogroupd = cogroup customers by Customer_ID, orders by Customer_ID;
grunt> semijoin = filter cogroupd by not IsEmpty(orders);
grunt> semijoin_dump = foreach semijoin generate flatten(customers);
grunt> dump semijoin_dump;
```

COGROUP CUSTOMERS AND ORDERS

Customer_ID	First_Nam	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

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

ORDER_ID	CUSTOMER_ID	ORDER_ QUANTI	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT
OD01	1	1	450	2016-01-24	1	Apple iPh
OD01	1	1	450	2016-01-24	2	Apple iPh
OD01	1	1	450	2016-01-24	3	Apple iP

CO-GROUP HELPS EXECUTE SEMI-JOINS

```
grunt> cogroupd = cogroup customers by Customer_ID, orders by Customer_ID;

grunt> semijoin = filter cogroupd by not IsEmpty(orders);

grunt> semijoin_dump = foreach semijoin generate flatten(customers);

grunt> dump semijoin_dump;
```

IsEmpty IS A FILTER FUNCTION

Customer_ID	First_Nam	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

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

ORDER_ID	CUSTOMER_ID	ORDER_ QUANTI	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT
OD01	1	1	450	2016-01-24	1	Apple iPh
OD01	1	1	450	2016-01-24	2	Apple iPh
OD01	1	1	450	2016-01-24	3	Apple iP

CO-GROUP HELPS EXECUTE SEMI-JOINS

IsEmpty IS A FILTER FUNCTION

```
grunt> cogroupd = cogroup customers by Customer_ID, orders by Customer_ID;

grunt> semijoin = filter cogroupd by not IsEmpty(orders);

grunt> semijoin_dump = foreach semijoin generate flatten(customers);

grunt> dump semijoin_dump;
```

IsEmpty REMOVES RECORDS FOR THOSE CUSTOMERS WHERE THERE ARE NO ORDERS

Customer_ID	First_Nam	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

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

ORDER_ID	CUSTOMER_ID	ORDER_ QUANTI	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT
OD01	1	1	450	2016-01-24	1	Apple iPh
OD01	1	1	450	2016-01-24	2	Apple iPh
OD01	1	1	450	2016-01-24	3	Apple iP

CO-GROUP HELPS EXECUTE SEMI-JOINS

```
grunt> cogroupd = cogroup customers by Customer_ID, orders by Customer_ID;
```

```
cogroupd = {Co-group,BAG OF CUSTOMERS, BAG OF ORDERS}
```

```
grunt> semijoin = filter cogroupd by not IsEmpty(orders);

grunt> semijoin_dump = foreach semijoin generate flatten(customers);

grunt> dump semijoin_dump;
```

Customer_ID	First_Nam	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

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

ORDER_ID	CUSTOMER_ID	ORDER_ QUANTI	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT
OD01	1	1	450	2016-01-24	1	Apple iPh
OD01	1	1	450	2016-01-24	2	Apple iPh
OD01	1	1	450	2016-01-24	3	Apple iP

CO-GROUP HELPS EXECUTE SEMI-JOINS

FLATTEN GENERATES ONLY CUSTOMERS DATA FROM THE BAG

```
grunt> semijoin = filter cogroupd by not IsEmpty(orders);

grunt> semijoin_dump = foreach semijoin generate flatten(customers);

grunt> dump semijoin_dump;
```

Customer_ID	First_Nam	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

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

ORDER_ID	CUSTOMER_ID	ORDER_ QUANTI	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT
OD01	1	1	450	2016-01-24	1	Apple iPh
OD01	1	1	450	2016-01-24	2	Apple iPh
OD01	1	1	450	2016-01-24	3	Apple iP

CO-GROUP HELPS EXECUTE SEMI-JOINS

FLATTEN(orders) WILL GENERATE orders DATA

FLATTEN(customers) WILL GENERATE customers DATA

```
grunt> cogroupd = cogroup customers by Customer_ID, orders by Customer_ID;
grunt> semijoin = filter cogroupd by not IsEmpty(orders);

grunt> semijoin_dump = foreach semijoin generate flatten(customers);

grunt> dump semijoin_dump;
```

Customer_ID	First_Nam	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

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

ORDER_ID	CUSTOMER_ID	ORDER_ QUANTI	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT
OD01	1	1	450	2016-01-24	1	Apple iPh
OD01	1	1	450	2016-01-24	2	Apple iPh
OD01	1	1	450	2016-01-24	3	Apple iP

1. SELECTING SPECIFIC FIELDS FROM THE RELATION

2. APPLYING SPECIFIC FUNCTIONS TO A FIELD

3. SELECTING DISTINCT OR FIXED NUMBER OF RECORDS

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 DISTINCT OR FIXED NUMBER OF RECORDS

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

WE WILL SEE HOW WE CAN EXECUTE THESE OPERATIONS IN PIG ONE BY ONE

9. SAMPLING RECORDS

SAMPLING IS DONE VIA THE SAMPLE COMMAND

9. SAMPLING RECORDS

SAMPLING IS DONE VIA THE SAMPLE COMMAND

SAMPLING IS PRETTY BASIC IN PIG (RELATIVE TO HIVE)

SAMPLING IS DONE VIA THE SAMPLE COMMAND

SPECIFY THE SAMPLE % AS A DOUBLE BETWEEN 0 AND 1

SAMPLING IS DONE VIA THE SAMPLE COMMAND

SPECIFY THE SAMPLE % AS A DOUBLE BETWEEN 0 AND 1

THE SAMPLE COMMAND READS THROUGH ALL THE DATA AND THEN RETURNS A PERCENTAGE OF ROWS

SPECIFY THE SAMPLE % AS A DOUBLE BETWEEN 0 AND 1

THE SAMPLE COMMAND READS THROUGH ALL THE DATA AND THEN RETURNS A PERCENTAGE OF ROWS

```
grunt> file1 = sample orders 0.27;
```

THIS COMMAND WILL RETURN 27% OF THE ROWS

WE HAVE DONE THE BASIC STUFF

NOW LET US DO SOME COMPLICATED STUFF

NESTED FOREACH

NESTED FOREACH

LET US ASSUME WE WANT TO CALCULATE TOTAL NUMBER OF ITEMS BOUGHT BY CUSTOMERS

AND FIND WHICH ITEMS WERE BOUGHT MOST OFTEN

LET US ASSUME WE WANT TO CALCULATE TOTAL NUMBER OF ITEMS BOUGHT BY CUSTOMERS

AND FIND WHICH ITEMS WERE BOUGHT MOST OFTEN

WE ARE LOOKING FOR SOMETHING LIKE THIS

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

Customer_ID	Product_ID	Order_Quantity	Total_Items
2	3	3	7

WE ARE LOOKING FOR SOMETHING LIKE THIS

ORDER_ID	CUSTOMER_ID	ORDER_ QUANTI TY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD02	2	2	400	2016-01-25	1	Apple iPhone 4S
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	3	400	2016-01-25	3	Apple iPhone 6

Customer_ID	Product_ID	Order_Quantity	Total_Items
2	3	3	7

THE CUSTOMER BOUGHT 3 IPHONE 6 OUT OF THE TOTAL 7 ITEMS HE BOUGHT

NESTED FOREACH

LET US ASSUME WE WANT TO CALCULATE TOTAL NUMBER OF ITEMS BOUGHT BY CUSTOMERS

AND FIND WHICH ITEMS WERE BOUGHT MOST OFTEN

```
grunt> groupd = group orders by (Customer_ID,Product_ID);

grunt> temp = foreach groupd generate group.Customer_ID as Customer_ID,
group.Product_ID as Product_ID, SUM(orders.Order_Quantity) as no_of_items;

grunt> groupd2 = group temp by Customer_ID;
```

NESTED FOREACH

LET US ASSUME WE WANT TO CALCULATE TOTAL NUMBER OF ITEMS BOUGHT BY CUSTOMERS

AND FIND WHICH ITEMS WERE BOUGHT MOST OFTEN

```
grunt> groupd = group orders by (Customer_ID,Product_ID);
```

FIRST GROUP BY `Customer_ID, Product_ID`

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

```
grunt> groupd = group orders by (Customer_ID,Product_ID);
```

FIRST GROUP BY Customer_ID, Product_ID

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

THE GROUP HAS 2 FIELDS, CUSTOMER_ID AND PRODUCT_ID

```
grunt> groupd = group orders by (Customer_ID,Product_ID);
```

FIRST GROUP BY Customer_ID, Product_ID

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

THE REMAINING FIELDS FROM ORDERS BECOME PART OF THE ORDERS BAG WITHIN THE RELATION

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

```
grunt> temp = foreach groupd generate group.Customer_ID as Customer_ID,
group.Product_ID as Product_ID, SUM(orders.Order_Quantity) as no_of_items;
```

Next, generate a table Customer_ID, Product_ID and no_of_items

```
grunt> temp = foreach groupd generate group.Customer_ID as Customer_ID,
group.Product_ID as Product_ID, SUM(orders.Order_Quantity) as no_of_items;
```

Next, generate a table Customer_ID, Product_ID and no_of_items

(Customer_ID, Product_ID,no_of_items)

```
(1,1,1)
(1,2,1)
(1,3,1)
(2,1,3)
(2,2,3)
(2,3,3)
(3,1,1)
(3,2,1)
(3,3,1)
(4,1,1)
(4,2,1)
(4,3,1)
```

(Customer_ID,
Product_ID,no_of_items)

```
grunt> groupd2 = group temp by Customer_ID;
```

```
(1,1,1)
(1,2,1)
(1,3,1)
(2,1,3)
(2,2,3)
(2,3,3)
(3,1,1)
(3,2,1)
(3,3,1)
(4,1,1)
(4,2,1)
(4,3,1)
```

NEXT WE GROUP Temp ON CUSTOMER_ID

NEXT WE GROUP Temp ON CUSTOMER_ID

```
grunt> groupd2 = group temp by Customer_ID;
```

THE SCHEMA OF groupd2

```
groupd2: {group: int,temp: {(Customer_ID: int,Product_ID:
int,no_of_items: long)}}
```

```
groupd2: {group: int,temp: {(Customer_ID: int,Product_ID:
int,no_of_items: long)}}
```

THE SCHEMA OF groupd2

```
(1,{(1,3,1),(1,2,1),(1,1,1)})
(2,{(2,3,3),(2,2,3),(2,1,3)})
(3,{(3,3,1),(3,2,1),(3,1,1)})
(4,{(4,3,1),(4,2,1),(4,1,1)})
```

This bit here corresponds to CUSTOMER_ID=1

```
(1,{(1,3,1),(1,2,1),(1,1,1)})
(2,{(2,3,3),(2,2,3),(2,1,3)})
(3,{(3,3,1),(3,2,1),(3,1,1)})
(4,{(4,3,1),(4,2,1),(4,1,1)})
```

FROM THIS GROUP STRUCTURE, WE WILL COMPUTE

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

Customer_ID	Product_ID	Order_Quantity	Total_Items
2	3	3	7

WE NEED TWO ITEMS

1. ITEM WHICH WAS BOUGHT MOST OFTEN

2. TOTAL ITEMS BOUGHT

```
groupd2: {group: int,temp: {(Customer_ID: int,Product_ID:
int,no_of_items: long)}}
```

FROM THIS GROUP STRUCTURE, WE WILL COMPUTE

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

Customer_ID	Product_ID	Order_Quantity	Total_Items
2	3	3	7

WE NEED TWO ITEMS

1. ITEM WHICH WAS BOUGHT MOST OFTEN

2. TOTAL ITEMS BOUGHT

```
groupd2: {group: int,temp: {(Customer_ID: int,Product_ID:
int,no_of_items: long)}}
```

FROM THIS GROUP STRUCTURE, WE WILL COMPUTE

1. ITEM WHICH WAS BOUGHT MOST OFTEN

TO DO THIS:

SORT THE BAG ITEMS ON NO_OF_ITEMS IN DECREASING ORDER

AND SELECT THE TOPMOST TUPLE

```
groupd2: {group: int,temp: {(Customer_ID: int,Product_ID:
int,no_of_items: long)}}
```

FROM THIS GROUP STRUCTURE, WE WILL COMPUTE

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

Customer_ID	Product_ID	Order_Quantity	Total_Items
2	3	3	7

WE NEED TWO ITEMS

1. ITEM WHICH WAS BOUGHT MOST OFTEN

2. TOTAL ITEMS BOUGHT

```
groupd2: {group: int,temp: {(Customer_ID: int,Product_ID:
int,no_of_items: long)}}
```

FROM THIS GROUP STRUCTURE, WE WILL COMPUTE

2. TOTAL ITEMS BOUGHT

TO DO THIS:

SUM THE `ORDER_QUANTITY` OF ALL BAG ITEMS

```
groupd2: {group: int,temp: {(Customer_ID: int,Product_ID:
int,no_of_items: long)}}
```

1. ITEM WHICH WAS BOUGHT MOST OFTEN

SORT THE BAG ITEMS ON NO_OF_ITEMS IN DECREASING ORDER

AND SELECT THE TOPMOST TUPLE

2. TOTAL ITEMS BOUGHT

SUM THE ORDER_QUANTITY OF ALL BAG ITEMS

NESTED FOREACH

WE CAN DO MULTIPLE OPERATIONS ON A GROUP STRUCTURE IN A SINGLE FOREACH COMMAND

NESTED FOREACH

THE SYNTAX OF FOREACH COMMAND

```
grunt> result1 = foreach groupd2{
total_items = SUM(temp.no_of_items);
sorted_items = order temp by no_of_items desc;
highest_item = limit sorted_items 1;
generate Flatten(highest_item),total_items;
};
```

NESTED FOREACH

THE BRACES INDICATE THE OPERATIONS TO BE DONE FOR EVERY RECORD

```
grunt> result1 = foreach groupd2{
total_items = SUM(temp.no_of_items);
sorted_items = order temp by no_of_items desc;
highest_item = limit sorted_items 1;
generate Flatten(highest_item),total_items;
};
```

NESTED FOREACH

CALCULATE THE TOTAL NUMBER OF ITEMS PER CUSTOMER

```
grunt> result1 = foreach groupd2{
total_items = SUM(temp.no_of_items);
sorted_items = order temp by no_of_items desc;
highest_item = limit sorted_items 1;
generate group,flatten(highest_item),total_items;
};
```

THIS VARIABLE EXISTS PER CUSTOMER

VARIABLES IN A NESTED FOREACH EXIST FOR EACH RECORD!

NESTED FOREACH

```
grunt> result1 = foreach groupd2{
total_items = SUM(temp.no_of_items);
sorted_items = order temp by no_of_items desc;
highest_item = limit sorted_items 1;
generate Flatten(highest_item),total_items;
};
```

THIS SORTS AND FINDS THE PRODUCT A CUSTOMER BOUGHT THE MOST NUMBER OF TIMES

```
groupd2: {group: int,temp: {(Customer_ID: int,Product_ID:
int,no_of_items: long)}}
```

```
grunt> result1 = foreach groupd2{
total_items = SUM(temp.no_of_items);
sorted_items = order temp by no_of_items desc;
```

SORT THE BAG ITEMS ON NO_OF_ITEMS IN DECREASING ORDER

ORDER_ID	CUSTOMER_ID	ORDER_ QUANTI TY	ORDER_VALUE	ORDER_DATE	PRODUCT_ID	PRODUCT_NAME
OD02	2	3	400	2016-01-25	1	Apple iPhone 6
OD02	2	2	400	2016-01-25	2	Apple iPhone 5S
OD02	2	2	400	2016-01-25	3	Apple iPhone 4

```
grunt> result1 = foreach groupd2{
total_items = SUM(temp.no_of_items);
sorted_items = order temp by no_of_items desc;
highest_item = limit sorted_items 1;
```

AND SELECT THE TOPMOST TUPLE IN THE RELATION highest_item

Customer_ID	Product_ID	Order_Quantity	Total_Items
2	3	3	7

```
groupd2: {group: int,temp:
{(Customer_ID: int,Product_ID:
int,no_of_items: long)}}
```

```
grunt> result1 = foreach groupd2{
total_items SUM(temp.no_of_items);
sorted_items = order temp by no_of_items desc;
highest_item limit sorted_items 1;
generate Flatten(highest_item),total_items;
};
```

NOW WE USE FLATTEN TO GET 1 RECORD WITH ALL THE INFORMATION WE NEED

Customer_ID	Product_ID	Order_Quantity	Total_Items
2	3	3	7

```
groupd2: {group: int,temp:
{(Customer_ID: int,Product_ID:
int,no_of_items: long)}}
```

```
grunt> result1 = foreach groupd2{
total_items = SUM(temp.no_of_items);
sorted_items = order temp by no_of_items desc;
highest_item = limit sorted_items 1;
generate Flatten(highest_item),total_items;
};
```

USE generate TO GET THE FINAL FIELDS THAT WE'RE INTERESTED IN

NESTED FOREACH

ONE MORE EXAMPLE

COUNT DISTINCT NUMBER OF PRODUCTS PER PERSON

```
grunt> groupd = group orders by (Customer_ID,Product_ID);

grunt> temp = foreach groupd generate group.Customer_ID as Customer_ID,
group.Product_ID as Product_ID, SUM(orders.Order_Quantity) as no_of_items;

grunt> groupd2 = group temp by Customer_ID;

grunt> unique_order_customer = foreach groupd2{
total_products = temp.Product_ID;
unique_products = distinct total_products;
generate COUNT(unique_products) as unique_products,group;
};
```

NESTED FOREACH

COUNT DISTINCT NUMBER OF PRODUCTS PER PERSON

```
grunt> groupd = group orders by (Customer_ID,Product_ID);

grunt> temp = foreach groupd generate group.Customer_ID as Customer_ID,
group.Product_ID as Product_ID, SUM(orders.Order_Quantity) as no_of_items;

grunt> groupd2 = group temp by Customer_ID;
```

THE SAME RELATION AS EARLIER, FOR EVERY CUSTOMER THE PRODUCTS AND THE QUANTITY OF EACH PRODUCT HE HAS BOUGHT

```
groupd2: {group: int,temp: {(Customer_ID:
int,Product_ID: int,no_of_items: long)}}
```

NESTED FOREACH

```
groupd2: {group: int,temp: {(Customer_ID:
int,Product_ID: int,no_of_items: long)}}
```

```
grunt> unique_order_customer = foreach groupd2{
total_products = temp.Product_ID;
unique_products = distinct total_products;
generate COUNT(unique_products) as unique_products,group;
};
```

HERE IS WHERE WE GENERATE THE NUMBER OF UNIQUE PRODUCTS ANY CUSTOMER HAS BOUGHT

NESTED FOREACH

```
groupd2: {group: int,temp: {(Customer_ID:
int,Product_ID: int,no_of_items: long)}}
```

```
grunt> unique_order_customer = foreach groupd2{
total_products = temp.Product_ID;
unique_products = distinct total_products;
generate COUNT(unique_products) as unique_products,group;
};
```

THIS GENERATES A BAG WHICH INCLUDES ONLY THE PRODUCT IDS

NESTED FOREACH

```
groupd2: {group: int,temp: {(Customer_ID:
int,Product_ID: int,no_of_items: long)}}
```

```
grunt> unique_order_customer = foreach groupd2{
total_products = temp.Product_ID;
unique_products = distinct total_products;
generate COUNT(unique_products) as unique_products,group;
};
```

UNIQUE_PRODUCTS HOLDS THE DISTINCT PRODUCTS BOUGHT BY ANY CUSTOMER

NESTED FOREACH

```
groupd2: {group: int,temp: {(Customer_ID:
int,Product_ID: int,no_of_items: long)}}
```

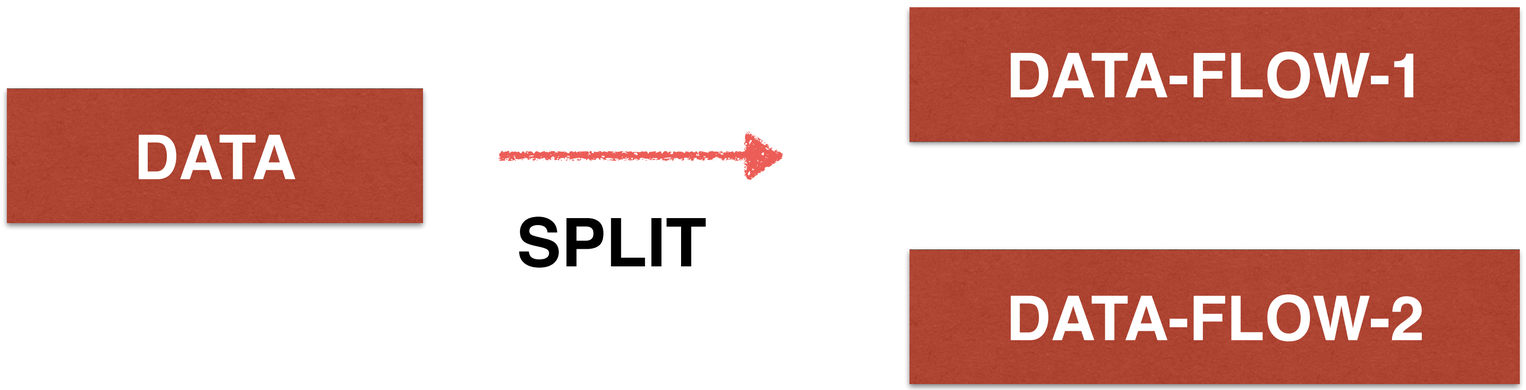
```
grunt> unique_order_customer = foreach groupd2{
total_products = temp.Product_ID;
unique_products = distinct total_products;
generate COUNT(unique_products) as unique_products,group;
};
```

GENERATE THE AGGREGATE COUNT OF UNIQUE PRODUCTS FOR ANY CUSTOMER

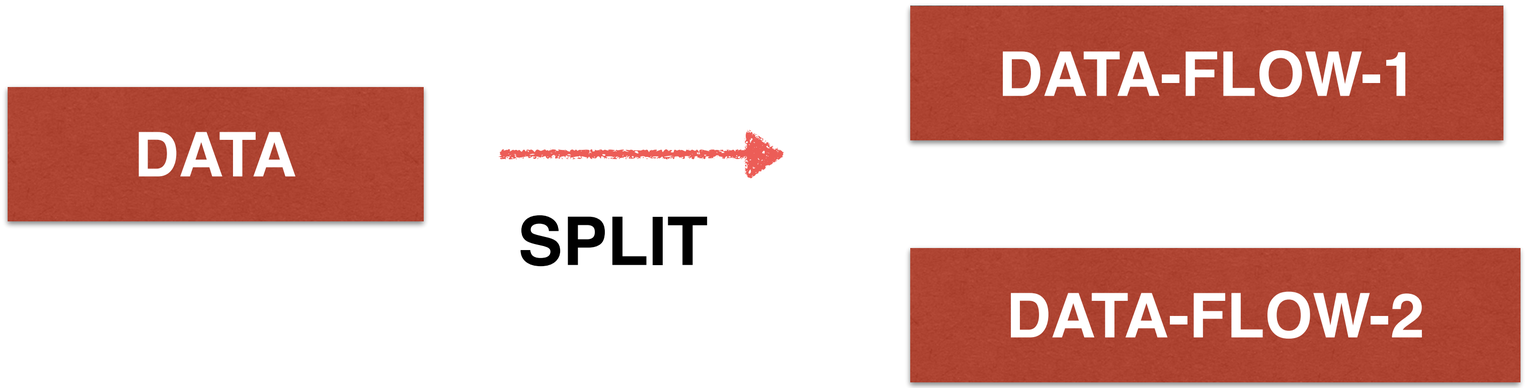
‘SPLIT’ COMMAND

'SPLIT' COMMAND

IT IS USED TO EXPLICITLY SPLIT DATA FLOWS



'SPLIT' COMMAND



SPLIT ALLOWS YOU TO SPLIT YOUR DATA IN AS MANY WAYS AS YOU LIKE

SPLIT ALLOWS YOU TO SPLIT YOUR DATA IN AS MANY WAYS AS YOU LIKE

```
grunt> split customers into customer_group_1 if
Customer_ID <=25,
customer_group_2 if Customer_ID <=50 and
Customer_ID >25,
customer_group_3 if Customer_ID <=75 and
Customer_ID >50,
customer_group_4 if Customer_ID >75;
```

```
grunt> split customers into customer_group_1 if Customer_ID <=25,
customer_group_2 if Customer_ID <=50 and Customer_ID >25,
customer_group_3 if Customer_ID <=75 and Customer_ID >50,
customer_group_4 if Customer_ID >75;
```

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
-	-	-	-	-	-
-	-	-	-	-	-
100	Andre as	Vicini	990000112349	2001-01-01	Andreas Vicini5@gmail.com

SPLIT

Customer_ID	First_ Name	Second_ Name	Contact_No	Created_ Date	Email_ID
1	Tracy	Stokley	990000112345	2001-01-01	Tracy Stokley1@gmail.com
25	-	-	-	-	-

customer_group_1

Customer_ID	First_ Name	Second_ Name	Contact_No	Created_ Date	Email_ID
26	-	-	-	-	-
50	-	-	-	-	-

customer_group_2

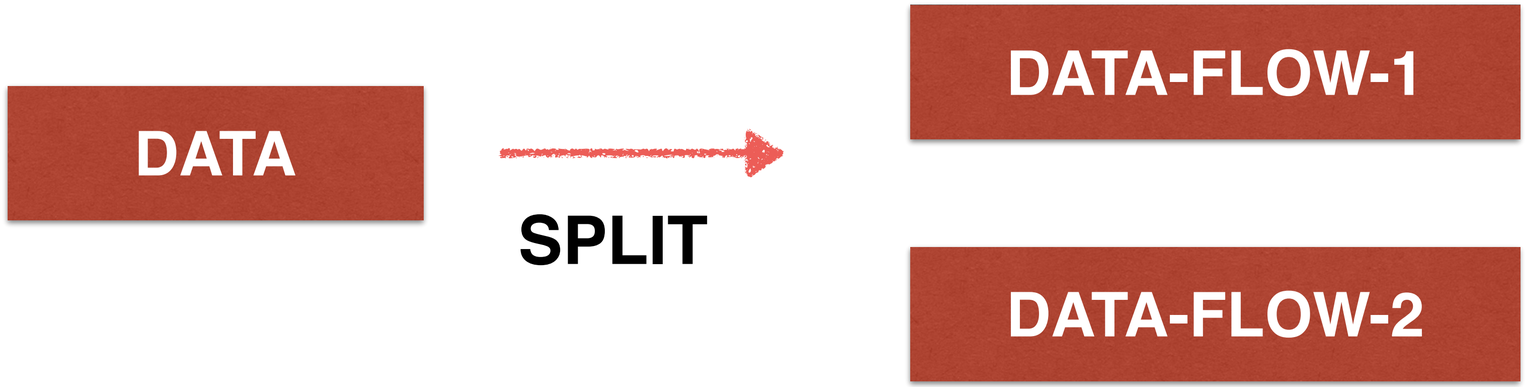
Customer_ID	First_ Name	Second_ Name	Contact_No	Created_Date	Email_ID
51	-	-	-	-	-
75	-	-	-	-	-

customer_group_3

Customer_ID	First_ Name	Second_ Name	Contact_No	Created_Date	Email_ID
76	-	-	-	-	-
100	Andrea s	Vicini	990000112349	2001-01-01	Andreas Vicini5@gmail.com

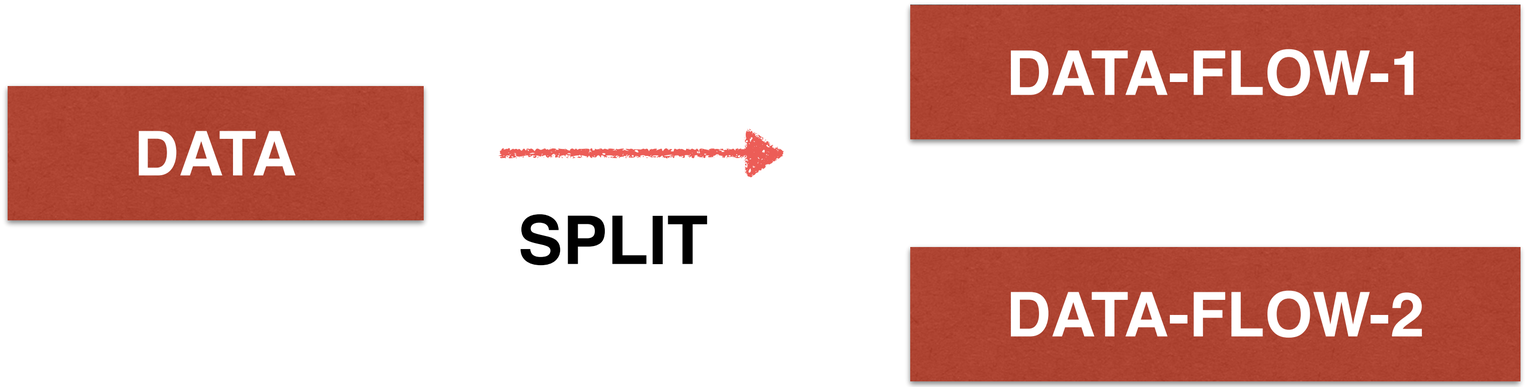
customer_group_4

'SPLIT' COMMAND



A SINGLE RECORD CAN GO TO MULTIPLE LEGS OF THE SPLIT

'SPLIT' COMMAND



IN THIS WAY - SPLIT IS NOT LIKE A SWITCH OR CASE STATEMENT

PIG PROVIDES VARIOUS DEBUGGING TOOLS

DESCRIBE

DUMP

EXPLAIN

ILLUSTRATE

PIG PROVIDES VARIOUS DEBUGGING TOOLS

DESCRIBE

DUMP

EXPLAIN

ILLUSTRATE

(We've covered these)

PIG PROVIDES VARIOUS DEBUGGING TOOLS

DESCRIBE

DUMP

EXPLAIN

ILLUSTRATE

EXPLAIN

1

SHOWS THE EXECUTION PLAN FOR A RELATION

```
grunt> explain cogrouped;

grunt> explain joined;
```

EXPLAIN

2

SHOWS THE EXECUTION PLAN FOR A PIG SCRIPT

```
grunt> explain PIG_SCRIPT.PIG;
```

EXPLAIN

PIG CONVERTS SCRIPTS INTO A SERIES OF MAPREDUCE JOBS

PIG DRAWS 3 PLANS

LOGICAL PLAN

PHYSICAL PLAN

MAPREDUCE PLAN

EXPLAIN

PIG DRAWS 3 PLANS

LOGICAL PLAN

PHYSICAL PLAN

MAPREDUCE PLAN

THE EXPLAIN COMMAND PRINTS
ALL 3 PLANS

EXPLAIN LOGICAL PLAN

PIG CONVERTS SCRIPTS INTO A SERIES OF MAPREDUCE JOBS

AFTER BASIC CHECKS, PIG DRAWS A LOGICAL PLAN WHICH DESCRIBES HOW IT WILL EXECUTE THE SCRIPT

EXPLAIN LOGICAL PLAN

AFTER BASIC CHECKS, PIG DRAWS A LOGICAL PLAN WHICH DESCRIBES HOW IT WILL EXECUTE THE SCRIPT

THE EXPLAIN COMMAND PRINTS THE LOGICAL PLAN FIRST

EXPLAIN LOGICAL PLAN

THE EXPLAIN COMMAND PRINTS THE LOGICAL PLAN FIRST

THE LOGICAL PLAN IS PRINTED DOWN UP

THE LOAD COMMAND, THE VERY FIRST WILL BE AT THE BOTTOM

```
#-----------------------------------------------
# New Logical Plan:
#-----------------------------------------------
grouped: (Name: LOStore Schema: group#3099:int,orders#3134:bag{#3143:tuple(Order_ID#3098:chararray,Customer_ID#3099:int,Order_Quantity#3100:int,Order_Value#31
01:float,Order_Date#3102:chararray,Product_ID#3103:int,Product_Name#3104:chararray)})
|
|---grouped: (Name: LOCogroup Schema: group#3099:int,orders#3134:bag{#3143:tuple(Order_ID#3098:chararray,Customer_ID#3099:int,Order_Quantity#3100:int,Order_Va
lue#3101:float,Order_Date#3102:chararray,Product_ID#3103:int,Product_Name#3104:chararray)})
    |   |
    |   Customer_ID:(Name: Project Type: int Uid: 3099 Input: 0 Column: 1)
    |
    |---orders: (Name: LOForEach Schema: Order_ID#3098:chararray,Customer_ID#3099:int,Order_Quantity#3100:int,Order_Value#3101:float,Order_Date#3102:chararray
,Product_ID#3103:int,Product_Name#3104:chararray)
        |   |
        |   (Name: LOGenerate[false,false,false,false,false,false,false] Schema: Order_ID#3098:chararray,Customer_ID#3099:int,Order_Quantity#3100:int,Order_Va
lue#3101:float,Order_Date#3102:chararray,Product_ID#3103:int,Product_Name#3104:chararray)ColumnPrune:OutputUids=[3104, 3098, 3099, 3100, 3101, 3102, 3103]Colu
mnPrune:InputUids=[3104, 3098, 3099, 3100, 3101, 3102, 3103]
        |   |   |
        |   |   (Name: Cast Type: chararray Uid: 3098)
        |   |   |
        |   |   |---Order_ID:(Name: Project Type: bytearray Uid: 3098 Input: 0 Column: (*))
        |   |   |
        |   |   (Name: Cast Type: int Uid: 3099)
        |   |   |
        |   |   |---Customer_ID:(Name: Project Type: bytearray Uid: 3099 Input: 1 Column: (*))
        |   |   |
        |   |   (Name: Cast Type: int Uid: 3100)
        |   |   |
        |   |   |---Order_Quantity:(Name: Project Type: bytearray Uid: 3100 Input: 2 Column: (*))
        |   |   |
        |   |   (Name: Cast Type: float Uid: 3101)
        |   |   |
        |   |   |---Order_Value:(Name: Project Type: bytearray Uid: 3101 Input: 3 Column: (*))
        |   |   |
        |   |   (Name: Cast Type: chararray Uid: 3102)
        |   |   |
        |   |   |---Order_Date:(Name: Project Type: bytearray Uid: 3102 Input: 4 Column: (*))
```

EXPLAIN

THE EXPLAIN COMMAND PRINTS THE LOGICAL PLAN FIRST
THE LOGICAL PLAN IS PRINTED DOWN UP

SOME OPTIMISATIONS ARE ADDED TO THIS LOGICAL PLAN

AFTER THIS, PIG PRODUCES A PHYSICAL PLAN

PHYSICAL PLAN

EXPLAIN PHYSICAL PLAN

AFTER THIS, PIG PRODUCES A PHYSICAL PLAN

THE PHYSICAL PLAN DESCRIBES THE PHYSICAL OPERATORS THAT PIG WILL USE TO EXECUTE THE SCRIPT

THE PHYSICAL PLAN DOESN'T REFERENCE MAPREDUCE JOBS

EXPLAIN PHYSICAL PLAN

AFTER THIS, PIG PRODUCES A PHYSICAL PLAN

THE PHYSICAL PLAN WILL LOOK LIKE LOGICAL PLAN BUT WILL HAVE MORE INFORMATION ON THE OPERATORS THAT WILL BE USED

THE PHYSICAL PLAN WILL LOOK LIKE LOGICAL PLAN BUT WILL HAVE MORE INFORMATION ON THE OPERATORS THAT WILL BE USED

```
#-----------------------------------------------
# Physical Plan:
#-----------------------------------------------
grouped: Store(fakefile:org.apache.pig.builtin.PigStorage) - scope-603
|
|---grouped: Package(Packager)[tuple]{int} - scope-600
    |
    |---grouped: Global Rearrange[tuple] - scope-599
        |
        |---grouped: Local Rearrange[tuple]{int}(false) - scope-601
            |   |
            |   Project[int][1] - scope-602
            |
            |---orders: New For Each(false,false,false,false,false,false,false)[bag] - scope-598
                |   |
                |   Cast[chararray] - scope-578
                |   |
                |   |---Project[bytearray][0] - scope-577
                |   |
                |   Cast[int] - scope-581
                |   |
                |   |---Project[bytearray][1] - scope-580
                |   |
                |   Cast[int] - scope-584
                |   |
                |   |---Project[bytearray][2] - scope-583
                |   |
                |   Cast[float] - scope-587
                |   |
                |   |---Project[bytearray][3] - scope-586
                |   |
                |   Cast[chararray] - scope-590
                |   |
                |   |---Project[bytearray][4] - scope-589
                |   |
                |   Cast[int] - scope-593
                |   |
                |   |---Project[bytearray][5] - scope-592
                |   |
                |   Cast[chararray] - scope-596
                |   |
                |   |---Project[bytearray][6] - scope-595
                |
                |---orders: Load(hdfs://localhost:9000/user/navdeepsingh/order_data.csv:PigStorage(',')) - scope-576
```

LOAD OPERATION IS DONE BY PigStorage

PATHS ARE ALSO SHOWN

EXPLAIN

THE PHYSICAL PLAN WILL LOOK LIKE LOGICAL PLAN
BUT WILL HAVE MORE INFORMATION ON THE
OPERATORS THAT WILL BE USED

AFTER THIS PIG WRITES MAPREDUCE PLAN

EXPLAIN

AFTER THIS PIG WRITES THE MAPREDUCE PLAN

THIS MAPREDUCE PLAN, IN TURN, HAS TWO PLANS

MAP PLANS

REDUCE PLANS

EXPLAIN

THIS MAPREDUCE PLAN, IN TURN, HAS TWO PLANS

MAP PLANS

REDUCE PLANS

MAPREDUCE PLAN DESCRIBES THE MAPS,
COMBINES AND REDUCES

THIS MAPREDUCE PLAN, IN TURN, HAS TWO PLANS

```
Map Plan
grouped: Local Rearrange[tuple]{int}(false) - scope-601
|   |
|   Project[int][1] - scope-602
|
|---orders: New For Each(false,false,false,false,false,false,false)[bag] - scope-598
    |   |
    |   Cast[chararray] - scope-578
    |   |
    |   |---Project[bytearray][0] - scope-577
    |   |
    |   Cast[int] - scope-581
    |   |
    |   |---Project[bytearray][1] - scope-580
    |   |
    |   Cast[int] - scope-584
    |   |
    |   |---Project[bytearray][2] - scope-583
    |   |
    |   Cast[float] - scope-587
    |   |
    |   |---Project[bytearray][3] - scope-586
    |   |
    |   Cast[chararray] - scope-590
    |   |
    |   |---Project[bytearray][4] - scope-589
    |   |
    |   Cast[int] - scope-593
    |   |
    |   |---Project[bytearray][5] - scope-592
    |   |
    |   Cast[chararray] - scope-596
    |   |
    |   |---Project[bytearray][6] - scope-595
    |
    |---orders: Load(hdfs://localhost:9000/user/navdeepsingh/order_data.csv:PigStorage(',')) - scope-576--------
Reduce Plan
grouped: Store(fakefile:org.apache.pig.builtin.PigStorage) - scope-603
|
|---grouped: Package(Packager)[tuple]{int} - scope-600--------
Global sort: false
----------------
```

THIS MAPREDUCE PLAN, IN TURN, HAS TWO PLANS

```
Map Plan
grouped: Local Rearrange[tuple]{int}(false) - scope-601
|   |
|   Project[int][1] - scope-602
|
|---orders: New For Each(false,false,false,false,false,false,false)[bag] - scope-598
    |   |
    |   Cast[chararray] - scope-578
    |   |
    |   |---Project[bytearray][0] - scope-577
    |   |
    |   Cast[int] - scope-581
    |   |
    |   |---Project[bytearray][1] - scope-580
    |   |
    |   Cast[int] - scope-584
    |   |
    |   |---Project[bytearray][2] - scope-583
    |   |
    |   Cast[float] - scope-587
    |   |
```

THIS MAPREDUCE PLAN, IN TURN, HAS TWO PLANS

```
|   |---Project[bytearray][2] - scope-583
|   |
|   Cast[bytearray] - scope-587
|   |
|   |---Project[bytearray][3] - scope-586
|   |
|   Cast[chararray] - scope-590
|   |
|   |---Project[bytearray][4] - scope-589
|   |
|   Cast[int] - scope-593
|   |
|   |---Project[bytearray][5] - scope-592
|   |
|   Cast[chararray] - scope-596
|   |
|   |---Project[bytearray][6] - scope-595
|
|---orders: Load(hdfs://localhost:9000/user/navdeepsingh/order_data.csv:PigStorage(',')) -
Reduce Plan
grouped: Store(fakefile:org.apache.pig.builtin.PigStorage) - scope-603
|
|---grouped: Package(Packager)[tuple]{int} - scope-600--------
Global sort: false
----------------
```

PIG PROVIDES VARIOUS DEBUGGING TOOL

DESCRIBE

DUMP

EXPLAIN

ILLUSTRATE

ILLUSTRATE

ILLUSTRATE TAKES A SAMPLE OF THE DATA AND RUNS IT THROUGH YOUR SCRIPT

ILLUSTRATE ENSURES THAT SOME RECORDS WILL DEFINITELY PASS THROUGH THE OPERATOR THAT REDUCES THE DATA

ILLUSTRATE

ILLUSTRATE ENSURES THAT SOME RECORDS WILL DEFINITELY PASS THROUGH THE OPERATOR THAT REDUCES THE DATA

IF NEEDED, ILLUSTRATE WILL MANUFACTURE SOME RECORDS SUCH THAT THEY PASS FILTERS

ILLUSTRATE

FOR EACH RELATION, ILLUSTRATE SHOWS RECORDS
AS THEY COME OUT OF THE RELATION

THIS IS TYPICALLY USED TO DEBUG YOUR SCRIPT
AND CATCH LOGICAL ERRORS

NESTED FOREACH

THE SYNTAX OF FOREACH COMMAND

```
grunt> groupd = group orders by (Customer_ID,Product_ID);

grunt> temp = foreach groupd generate group.Customer_ID as
Customer_ID, group.Product_ID as Product_ID,
SUM(orders.Order_Quantity) as no_of_items;

grunt> groupd2 = group temp by Customer_ID;

grunt> result1 = foreach groupd2{
total_items = SUM(temp.no_of_items);
sorted_items = order temp by no_of_items desc;
highest_item = limit sorted_items 1;
generate Flatten(highest_item),total_items;
};
```

ILLUSTRATE

```
grunt> illustrate result1;
```

```
(AliasName[line,offset]): M: groupd2[50,10] C:  R: result1[67,10],sorted_items[69,15],highest_item[70,15]
---------------------------------------------------------------------------------------------------------------------------------
| orders     | Order_ID:chararray      | Customer_ID:int      | Order_Quantity:int      | Order_Value:float      | Order_Date:chararray      | Product_ID:int      | Product_Name:chararray      |
---------------------------------------------------------------------------------------------------------------------------------
|            | OD02                    | 2                    | 2                       | 400.0                  | 2016-01-24                | 1                   | Apple iPhone 4S             |
|            | OD03                    | 2                    | 1                       | 350.0                  | 2016-01-25                | 1                   | Apple iPhone 4S             |
|            | OD03                    | 2                    | 1                       | 350.0                  | 2016-01-25                | 2                   | Apple iPhone 5S             |
|            | OD02                    | 2                    | 2                       | 400.0                  | 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             |
|            | OD03                    | 2                    | 1                       | 350.0                  | 2016-01-25                | 2                   | Apple iPhone 5S             |
---------------------------------------------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------------------------------------------------------------------------------------
| groupd     | group:tuple(Customer_ID:int,Product_ID:int)                 | orders:bag{:tuple(Order_ID:chararray,Customer_ID:int,Order_Quantity:int,Order_Value:float,Order_Date:chararray,Product_ID:int,Product_Name:chararray)}                    |
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------
|            | (2, 1)                                      | {}                                                                    |
|            | (2, 1)                                      | {}                                                                    |
|            | (2, 2)                                      | {}                                                                    |
|            | (2, 2)                                      | {}                                                                    |
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------

------------------------------------------------------------------------
| temp     | Customer_ID:int      | Product_ID:int      | no_of_items:long      |
------------------------------------------------------------------------
|          | 2                    | 1                   | 8                     |
|          | 2                    | 2                   | 2                     |
------------------------------------------------------------------------

-----------------------------------------------------------------------------------------------------------
| groupd2     | group:int      | temp:bag{:tuple(Customer_ID:int,Product_ID:int,no_of_items:long)}                        |
-----------------------------------------------------------------------------------------------------------
|             | 2              | {(2, 1, 8), (2, 2, 2)}                                                       |
-----------------------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------
| result1.sorted_items     | Customer_ID:int      | Product_ID:int      | no_of_items:long      |
-------------------------------------------------------------------------------------
|                          | 2                    | 1                   | 8                     |
|                          | 2                    | 2                   | 2                     |
-------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------
| result1.highest_item     | Customer_ID:int      | Product_ID:int      | no_of_items:long      |
-------------------------------------------------------------------------------------
|                          | 2                    | 1                   | 8                     |
|                          | 2                    | 2                   | 2                     |
-------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------------------------------
| result1     | highest_item::Customer_ID:int      | highest_item::Product_ID:int      | highest_item::no_of_items:long      | :long      |
-------------------------------------------------------------------------------------------------------------------
|             | 2                                  | 1                                 | 8                                   | 10         |
|             | 2                                  | 2                                 | 2                                   | 10         |
-------------------------------------------------------------------------------------------------------------------
```

This is the output of the ILLUSTRATE COMMAND

NESTED FOREACH

THE SYNTAX OF FOREACH COMMAND

```
grunt> groupd = group orders by (Customer_ID,Product_ID);

grunt> temp = foreach groupd generate group.Customer_ID as
Customer_ID, group.Product_ID as Product_ID,
SUM(orders.Order_Quantity) as no_of_items;

grunt> groupd2 = group temp by Customer_ID;

grunt> result1 = foreach groupd2{
total_items = SUM(temp.no_of_items);
sorted_items = order temp by no_of_items desc;
highest_item = limit sorted_items 1;
generate Flatten(highest_item),total_items;
};
```

ILLUSTRATE

```
grunt> illustrate result1;
```

(AliasName[line,offset]): M: groupd2[50,10] C: R: result1[67,10],sorted_items[69,15],highest_item[70,15]

orders	Order_ID:chararray	Customer_ID:int	Order_Quantity:int	Order_Value:float	Order_Date:chararray	Product_ID:int	Product_Name:chararray
	OD02	2	2	400.0	2016-01-24	1	Apple iPhone 4S
	OD03	2	1	350.0	2016-01-25	1	Apple iPhone 4S
	OD03	2	1	350.0	2016-01-25	2	Apple iPhone 5S
	OD02	2	2	400.0	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
	OD03	2	1	350.0	2016-01-25	2	Apple iPhone 5S

groupd	group:tuple(Customer_ID:int,Product_ID:int)	orders:bag{:tuple(Order_ID:chararray,Customer_ID:int,Order_Quantity:int,Order_Value:float,Order_Date:chararray,Product_ID:int,Product_Name:chararray)}
	(2, 1)	{}
	(2, 1)	{}
	(2, 2)	{}
	(2, 2)	{}

temp	Customer_ID:int	Product_ID:int	no_of_items:long
	2	1	8
	2	2	2

groupd2	group:int	temp:bag{:tuple(Customer_ID:int,Product_ID:int,no_of_items:long)}
	2	{(2, 1, 8), (2, 2, 2)}

result1.sorted_items	Customer_ID:int	Product_ID:int	no_of_items:long
	2	1	8

IT FIRST STARTS WITH ORDERS RELATION

NESTED FOREACH

THE SYNTAX OF FOREACH COMMAND

```
grunt> groupd = group orders by (Customer_ID,Product_ID);

grunt> temp = foreach groupd generate group.Customer_ID as
Customer_ID, group.Product_ID as Product_ID,
SUM(orders.Order_Quantity) as no_of_items;

grunt> groupd2 = group temp by Customer_ID;

grunt> result1 = foreach groupd2{
total_items = SUM(temp.no_of_items);
sorted_items = order temp by no_of_items desc;
highest_item = limit sorted_items 1;
generate Flatten(highest_item),total_items;
};
```

ILLUSTRATE

```
grunt> illustrate result1;
```

```
| orders | Order_ID:chararray | Customer_ID:int | Order_Quantity:int | Order_Value:float | Order_Date:chararray | Product_ID:int | Product_Name:chararray |
---------------------------------------------------------------------------------------------------------------------------------------------
| | OD02 | 2 | 2 | 400.0 | 2016-01-25 | 2 | Apple iPhone 5S |
| | OD03 | 2 | 1 | 350.0 | 2016-01-25 | 1 | Apple iPhone 4S |
| | OD03 | 2 | 1 | 350.0 | 2016-01-25 | 2 | Apple iPhone 5S |
| | OD02 | 2 | 2 | 400.0 | 2016-01-25 | 1 | Apple iPhone 4S |
| | OD03 | 2 | 1 | 350.0 | 2016-01-25 | 1 | Apple iPhone 4S |
| | OD03 | 2 | 1 | 350.0 | 2016-01-25 | 2 | Apple iPhone 5S |
---------------------------------------------------------------------------------------------------------------------------------------------
```

```
-----------------------
| groupd | group:tuple(Customer_ID:int,Product_ID:int) |
orders:bag{:tuple(Order_ID:chararray,Customer_ID:int,Order_Quantity:int,Order_Value:float,Order_Date:chararray,Product_ID:int,Product_Name:chararray)}
|
---------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------
| | (2, 1) | {} |
| | (2, 1) | {} |
| | (2, 2) | {} |
| | (2, 2) | {} |
---------------------------------------------------------------------------
```

GROUP BY HAPPENS ON TWO COLUMNS - CUSTOMER_ID, PRODUCT_ID

```
-----------------------------------------
| temp | Customer_ID:int | Product_ID:int | no_of_items:long |
-----------------------------------------
| | 2 | 1 | 8 |
| | 2 | 2 | 2 |
-----------------------------------------
```

```
-----------------------------------------------------------------------------
| groupd2 | group:int | temp:bag{:tuple(Customer_ID:int,Product_ID:int,no_of_items:long)} |
-----------------------------------------------------------------------------
| | 2 | {(2, 1, 8), (2, 2, 2)} |
-----------------------------------------------------------------------------
```

```
-----------------------------------------
| result1.sorted_items | Customer_ID:int | Product_ID:int | no_of_items:long |
-----------------------------------------
| | 2 | 1 | 8 |
| | 2 | 2 | 2 |
-----------------------------------------
```

NESTED FOREACH

THE SYNTAX OF FOREACH COMMAND

```
grunt> groupd = group orders by (Customer_ID,Product_ID);

grunt> temp = foreach groupd generate group.Customer_ID as
Customer_ID, group.Product_ID as Product_ID,
SUM(orders.Order_Quantity) as no_of_items;

grunt> groupd2 = group temp by Customer_ID;

grunt> result1 = foreach groupd2{
total_items = SUM(temp.no_of_items);
sorted_items = order temp by no_of_items desc;
highest_item = limit sorted_items 1;
generate Flatten(highest_item),total_items;
};
```

ILLUSTRATE

```
grunt> illustrate result1;
```

temp	Customer_ID:int	Product_ID:int	no_of_items:long
	2	1	8
	2	2	2

ILLUSTRATE SHOWS THAT THE TEMP TABLE IS GENERATED NEXT

NESTED FOREACH

THE SYNTAX OF FOREACH COMMAND

```
grunt> groupd = group orders by (Customer_ID,Product_ID);

grunt> temp = foreach groupd generate group.Customer_ID as
Customer_ID, group.Product_ID as Product_ID,
SUM(orders.Order_Quantity) as no_of_items;

grunt> groupd2 = group temp by Customer_ID;

grunt> result1 = foreach groupd2{
total_items = SUM(temp.no_of_items);
sorted_items = order temp by no_of_items desc;
highest_item = limit sorted_items 1;
generate Flatten(highest_item),total_items;
};
```

ILLUSTRATE

```
grunt> illustrate result1;
```

```
----------------------------------------------------------------------------------------------------------------------------------------------------
| groupd2     | group:int       | temp:bag{:tuple(Customer_ID:int,Product_ID:int,no_of_items:long)}                                                  |
----------------------------------------------------------------------------------------------------------------------------------------------------
|             | 2               | {(2, 1, 8), (2, 2, 2)}                                                                                             |
----------------------------------------------------------------------------------------------------------------------------------------------------
```

groupd2 is generated at this step by doing a group on temp relation on Customer_ID

NESTED FOREACH

THE SYNTAX OF FOREACH COMMAND

```
grunt> groupd = group orders by (Customer_ID,Product_ID);

grunt> temp = foreach groupd generate group.Customer_ID as
Customer_ID, group.Product_ID as Product_ID,
SUM(orders.Order_Quantity) as no_of_items;

grunt> groupd2 = group temp by Customer_ID;

grunt> result1 = foreach groupd2{
total_items = SUM(temp.no_of_items);
sorted_items = order temp by no_of_items desc;
highest_item = limit sorted_items 1;
generate Flatten(highest_item),total_items;
};
```

ILLUSTRATE

```
grunt> illustrate result1;
```

sorted_items is generated at this step by ordering temp relation on no_of_items

---

result1.sorted_items	Customer_ID:int	Product_ID:int	no_of_items:long
	2	1	8
	2	2	2

---

result1.highest_item	Customer_ID:int	Product_ID:int	no_of_items:long
	2	1	8
	2	2	2

---

result1	highest_item::Customer_ID:int	highest_item::Product_ID:int	highest_item::no_of_items:long	:long
	2	1	8	10

NESTED FOREACH

THE SYNTAX OF FOREACH COMMAND

```
grunt> groupd = group orders by (Customer_ID,Product_ID);

grunt> temp = foreach groupd generate group.Customer_ID as
Customer_ID, group.Product_ID as Product_ID,
SUM(orders.Order_Quantity) as no_of_items;

grunt> groupd2 = group temp by Customer_ID;

grunt> result1 = foreach groupd2{
total_items = SUM(temp.no_of_items);
sorted_items = order temp by no_of_items desc;
highest_item = limit sorted_items 1;
generate Flatten(highest_item),total_items;
};
```

ILLUSTRATE

```
grunt> illustrate result1;
```

highest_item is generated at this step by selecting top row of sorted_items for every Customer_ID

result1.highest_item	Customer_ID:int	Product_ID:int	no_of_items:long
	2	1	8
	2	2	2

result1	highest_item::Customer_ID:int	highest_item::Product_ID:int	highest_item::no_of_items:long	:long
	2	1	8	10
	2	2	2	10

NESTED FOREACH

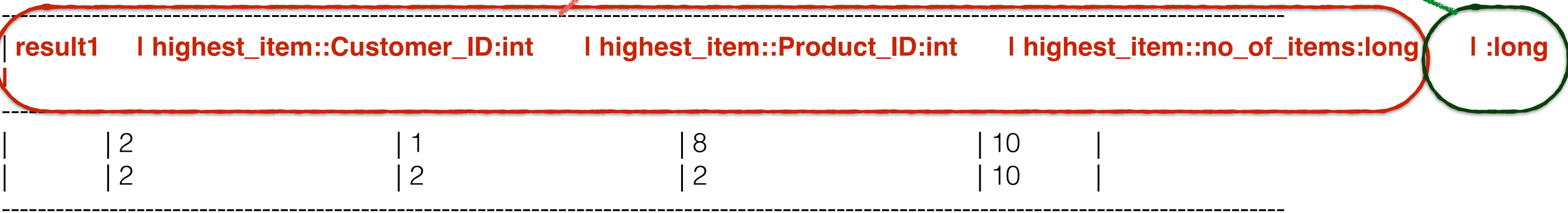
THE SYNTAX OF FOREACH COMMAND

```
grunt> groupd = group orders by (Customer_ID,Product_ID);

grunt> temp = foreach groupd generate group.Customer_ID as
Customer_ID, group.Product_ID as Product_ID,
SUM(orders.Order_Quantity) as no_of_items;

grunt> groupd2 = group temp by Customer_ID;

grunt> result1 = foreach groupd2{
total_items = SUM(temp.no_of_items);
sorted_items = order temp by no_of_items desc;
highest_item = limit sorted_items 1;
generate Flatten(highest_item),total_items;
};
```

ILLUSTRATE

```
grunt> illustrate result1;
```

result1 is finally generated at this step by flattening highest_items and adding last column of total_items



result1	highest_item::Customer_ID:int	highest_item::Product_ID:int	highest_item::no_of_items:long	:long
	2	1	8	10
	2	2	2	10

ILLUSTRATE

ILLUSTRATE TAKES A SAMPLE OF THE DATA AND RUNS IT THROUGH YOUR SCRIPT

ILLUSTRATE ENSURES THAT SOME RECORDS WILL DEFINITELY PASS THROUGH THE OPERATOR THAT REDUCES THE DATA

ILLUSTRATE

ILLUSTRATE ENSURES THAT SOME RECORDS WILL DEFINITELY PASS THROUGH THE OPERATOR THAT REDUCES THE DATA

IF NEEDED, ILLUSTRATE WILL MANUFACTURE SOME RECORDS SUCH THAT THEY PASS FILTERS

ILLUSTRATE

FOR EACH RELATION, ILLUSTRATE SHOWS RECORDS
AS THEY COME OUT OF THE RELATION

THIS IS TYPICALLY USED TO DEBUG YOUR SCRIPT
AND CATCH LOGICAL ERRORS

OPTIMIZING YOUR PIG LATIN SCRIPTS

OPTIMIZING YOUR PIG LATIN SCRIPTS

THERE ARE NUMBER OF WAYS IN WHICH YOU CAN OPTIMISE PIG LATIN SCRIPTS

THESE FALL BROADLY INTO 2 CATEGORIES

THERE ARE NUMBER OF WAYS IN WHICH YOU CAN OPTIMISE PIG LATIN SCRIPTS

THESE FALL BROADLY INTO 2 CATEGORIES

SPECIFIC COMMANDS/KEYWORDS

TRICKS/TACTICAL CHANGES (COMMON SENSICAL)
CAN BE USED IN ANY SCENARIO

SPECIFIC COMMANDS/KEYWORDS

MOST COMMON COMMANDS OR KEYWORDS THAT ARE USED TO OPTIMISE ARE

THE PARALLEL KEYWORD

DIFFERENT JOIN OPERATIONS

THE PARALLEL KEYWORD

ANY RELATIONAL OPERATION CAN BE MODIFIED WITH THE PARALLEL KEYWORD

A PARALLEL CLAUSE IS ONLY USED TO CONTROL REDUCE-SIDE PARALLELISM

A PARALLEL CLAUSE IS ONLY USED TO CONTROL REDUCE-SIDE PARALLELISM

MapReduce ALLOW USERS TO SET REDUCE-SIDE PARALLELISM

MAP-SIDE PARALLELISM IS CONTROLLED BY MapReduce AND HENCE IS NOT EXPOSED TO USERS IN PIG

THE PARALLEL KEYWORD

A PARALLEL CLAUSE IS ONLY USED TO CONTROL REDUCE-SIDE PARALLELISM

IT MAKES MORE SENSE ONLY FOR THOSE OPERATIONS THAT INVOLVE REDUCE PHASE

THE PARALLEL KEYWORD

ONLY MAKES SENSE FOR OPERATIONS THAT INVOLVE A REDUCE PHASE

Group

Order

Distinct

Join

Limit

Cogroup

Cross

THE PARALLEL KEYWORD

PARALLEL CLAUSE SETS THE NUMBER OF REDUCERS

```
grunt> groupd = group orders by (Customer_ID,Product_ID) Parallel 10;
```

THE MAPREDUCE JOB FOR THIS SCRIPT WILL HAVE 10 REDUCERS

THE PARALLEL KEYWORD

PARALLEL CLAUSE SETS THE NUMBER OF REDUCERS

```
grunt> groupd = group orders by (Customer_ID,Product_ID) Parallel 10;
```

PARALLEL COMMAND SETS 10 REDUCERS ONLY DURING EXECUTION OF THIS RELATION

THE PARALLEL KEYWORD

YOU CAN SET SCRIPT-WIDE DEFAULT PARALLEL VALUE USING

```
set default_parallel 5;
```

THE PARALLEL KEYWORD

MORE PARALLELISM IS NOT ALWAYS BETTER

PARALLELISM HAS

DATA TRANSFER COST

HIGH LATENCY COST

MORE PARALLELISM IS NOT ALWAYS BETTER

PARALLELISM HAS DATA TRANSFER COST

DATA THAT BELONGS TO A REDUCER HAS TO BE MOVED FROM EVERY MAPPER TO ITSELF

number of mappers: x

number of reducers: y

total number of Data Transfers = xy

MORE PARALLELISM IS NOT ALWAYS BETTER

PARALLELISM HAS DATA TRANSFER COST

AS NUMBER OF REDUCERS INCREASE, THE TRANSFER COST INCREASES EXPONENTIALLY

MORE PARALLELISM IS NOT ALWAYS BETTER

THERE ARE ALWAYS A LIMITED NUMBER OF REDUCERS

IT IS RECOMMENDED THAT NUMBER OF REDUCERS BE SET TO LESS THAN THE NUMBER OF REDUCE SLOTS AVAILABLE IN THE CLUSTER

MORE PARALLELISM IS NOT ALWAYS BETTER

IT IS RECOMMENDED THAT **NUMBER OF REDUCERS** BE SET TO LESS THAN THE NUMBER OF REDUCE SLOTS AVAILABLE IN THE CLUSTER

MORE PARALLELISM IS NOT ALWAYS BETTER

IT IS RECOMMENDED THAT NUMBER OF REDUCERS BE SET TO LESS THAN THE NUMBER OF REDUCE SLOTS AVAILABLE IN THE CLUSTER

MORE PARALLELISM IS NOT ALWAYS BETTER

IT IS RECOMMENDED THAT NUMBER OF REDUCERS BE SET TO LESS THAN THE NUMBER OF REDUCE SLOTS AVAILABLE IN THE CLUSTER

MORE PARALLELISM IS NOT ALWAYS BETTER

THERE ARE ALWAYS LIMITED NUMBER OF REDUCERS

IT IS RECOMMENDED THAT NUMBER OF REDUCERS BE SET TO LESS THAN THE NUMBER OF REDUCE SLOTS AVAILABLE IN THE CLUSTER

MORE PARALLELISM IS NOT ALWAYS BETTER

IT IS RECOMMENDED THAT NUMBER OF REDUCERS BE SET TO LESS THAN THE NUMBER OF REDUCE SLOTS AVAILABLE IN THE CLUSTER

IT CANNOT BE GREATER (THERE ARE NOT ENOUGH SLOTS)

HAVING FEWER REDUCERS THAN SLOTS ALLOW ROBUSTNESS IF ANY OF THE NODES GO DOWN

THE PARALLEL KEYWORD

PARALLEL WORKS ONLY IN HADOOP CLUSTER MODE

IT IS IGNORED IN LOCAL MODE

THE PARALLEL KEYWORD

IF YOU DO NOT SET PARALLELISM

PIG USES A HEURISTIC THAT WAS ADDED IN VERSION 0.8

TO GROSS ESTIMATE THE NUMBER OF REDUCERS

THE PARALLEL KEYWORD

PIG USES A HEURISTIC THAT WAS ADDED IN VERSION 0.8

THIS IS NOT A GOOD ALGORITHM

IT DOES NOT OPTIMIZE

THE PARALLEL KEYWORD

PIG USES A HEURISTIC THAT WAS ADDED IN VERSION 0.8

IT IS JUST A SAFETY NET

THE PARALLEL KEYWORD

IT IS RECOMMENDED THAT YOU SET A VALUE FOR PARALLELISM

THE PARALLEL KEYWORD

RULE OF THUMB

1 REDUCER PER 1 GB OF DATA

SPECIFIC COMMANDS/KEYWORDS

MOST COMMON COMMANDS OR KEYWORDS THAT ARE USED TO OPTIMISE ARE

THE PARALLEL KEYWORD

DIFFERENT JOIN OPERATIONS

DIFFERENT JOIN OPERATIONS

LET'S CONSIDER 4 JOIN OPTIMISATIONS

JOINING MULTIPLE INPUTS

JOINING SMALL TO LARGE DATA

JOINING SKEWED DATA

JOINING SORTED DATA

JOINING MULTIPLE INPUTS

CONSIDER THESE INPUTS

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST

TRADES

NAMES

SYMBOL	NAME
RIL	Reliance
NESTLEIND	Nestle

REVENUE

TOTAL REVENUE	NAME	SYMBOL
$ 4 Billion	Reliance	RIL
$ 0.5 Billion	Nestle	NESTLEIND

TRADES

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST

NAMES

SYMBOL	NAME
RIL	Reliance
NESTLEIND	Nestle

REVENUE

TOTAL_REVENUE	NAME	SYMBOL
$ 4 Billion	Reliance	RIL
$ 0.5 Billion	Nestle	NESTLEIND

join NAMES by Symbol, TRADES by Symbol, Revenue by Symbol

THESE ARE THE TABLES TO JOIN

JOINS

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST

TRADES

500 GB

NAMES

SYMBOL	NAME
RIL	Reliance
NESTLEIND	Nestle

50 MB

REVENUE

TOTAL REVENUE	NAME	SYMBOL
$ 4 Billion	Reliance	RIL
$ 0.5 Billion	Nestle	NESTLEIND

500 MB

LET US ASSUME SIZES OF THESE TABLES ARE

TRADES 500 GB

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST

NAMES 50 MB

SYMBOL	NAME
RIL	Reliance
NESTLEIND	Nestle

REVENUE 500 MB

TOTAL REVENUE	NAME	SYMBOL
$ 4 Billion	Reliance	RIL
$ 0.5 Billion	Nestle	NESTLEIND

join NAMES by Symbol, TRADES by Symbol, Revenue by Symbol

WE HAVE THREE TABLES HERE IN THE SEQUENCE

TRADES 500 GB

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST

NAMES 50 MB

SYMBOL	NAME
RIL	Reliance
NESTLEIND	Nestle

REVENUE 500 MB

TOTAL REVENUE	NAME	SYMBOL
$ 4 Billion	Reliance	RIL
$ 0.5 Billion	Nestle	NESTLEIND

join NAMES by Symbol, TRADES by Symbol, Revenue by Symbol

NAMES

TRADES 500 GB

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST

NAMES 50 MB

SYMBOL	NAME
RIL	Reliance
NESTLEIND	Nestle

REVENUE 500 MB

TOTAL REVENUE	NAME	SYMBOL
$ 4 Billion	Reliance	RIL
$ 0.5 Billion	Nestle	NESTLEIND

join NAMES by Symbol, TRADES by Symbol, Revenue by Symbol

NAMES

TRADES

TRADES 500 GB

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST

NAMES 50 MB

SYMBOL	NAME
RIL	Reliance
NESTLEIND	Nestle

REVENUE 500 MB

TOTAL REVENUE	NAME	SYMBOL
$ 4 Billion	Reliance	RIL
$ 0.5 Billion	Nestle	NESTLEIND

join NAMES by Symbol, TRADES by Symbol, Revenue by Symbol

NAMES TRADES

REVENUE

TRADES 500 GB

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST

NAMES 50 MB

SYMBOL	NAME
RIL	Reliance
NESTLEIND	Nestle

REVENUE 500 MB

TOTAL REVENUE	NAME	SYMBOL
$ 4 Billion	Reliance	RIL
$ 0.5 Billion	Nestle	NESTLEIND

NAMES TRADES REVENUE

IN THE JOIN MAPREDUCE, THE LAST TABLE IS STREAMED TO THE REDUCE OPERATION.

THE REMAINING TABLES ARE BUFFERED

TRADES 500 GB

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST

NAMES 50 MB

SYMBOL	NAME
RIL	Reliance
NESTLEIND	Nestle

REVENUE 500 MB

TOTAL REVENUE	NAME	SYMBOL
$ 4 Billion	Reliance	RIL
$ 0.5 Billion	Nestle	NESTLEIND

NAMES TRADES REVENUE

IN THE JOIN MAPREDUCE, THE LAST TABLE IS STREAMED TO THE REDUCE OPERATION.

THE REMAINING TABLES ARE BUFFERED

TRADES 500 GB

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST

NAMES 50 MB

SYMBOL	NAME
RIL	Reliance
NESTLEIND	Nestle

REVENUE 500 MB

TOTAL REVENUE	NAME	SYMBOL
$ 4 Billion	Reliance	RIL
$ 0.5 Billion	Nestle	NESTLEIND

NAMES TRADES REVENUE

IN THE JOIN MAPREDUCE, THE LAST TABLE IS STREAMED TO THE REDUCE OPERATION.

THE REMAINING TABLES ARE **BUFFERED**

TRADES

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST

NAMES 50 MB

SYMBOL	NAME
RIL	Reliance
NESTLEIND	Nestle

REVENUE 500 MB

TOTAL REVENUE	NAME	SYMBOL
$ 4 Billion	Reliance	RIL
$ 0.5 Billion	Nestle	NESTLEIND

NAMES TRADES REVENUE

THIS MEANS THAT 500 GB WORTH OF DATA IS KEPT IN MEMORY

500 GB

TRADES

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST

NAMES 50 MB

SYMBOL	NAME
RIL	Reliance
NESTLEIND	Nestle

REVENUE 500 MB

TOTAL REVENUE	NAME	SYMBOL
$ 4 Billion	Reliance	RIL
$ 0.5 Billion	Nestle	NESTLEIND

NAMES TRADES REVENUE

IN ORDER TO OPTIMIZE JOINS WE SHOULD MINIMIZE THE DATA HELD IN MEMORY

500 GB

TRADES 500 GB

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST

NAMES 50 MB

SYMBOL	NAME
RIL	Reliance
NESTLEIND	Nestle

REVENUE 500 MB

TOTAL REVENUE	NAME	SYMBOL
$ 4 Billion	Reliance	RIL
$ 0.5 Billion	Nestle	NESTLEIND

NAMES TRADES REVENUE

WRITE THE QUERY IN SUCH A MANNER THAT THE LARGEST TABLE APPEARS LAST

TRADES 500 GB

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST

NAMES 50 MB

SYMBOL	NAME
RIL	Reliance
NESTLEIND	Nestle

REVENUE 500 MB

TOTAL REVENUE	NAME	SYMBOL
$ 4 Billion	Reliance	RIL
$ 0.5 Billion	Nestle	NESTLEIND

NAMES

TRADES

REVENUE

IN THE JOIN MAPREDUCE THE MOST TABLE IS STREAMED TO THE REDUCER OPERATION. THE REMAINING TABLES ARE BUFFERED

SINCE TRADES IS THE LARGEST TABLE WE SHOULD TRY TO MAKE IT THE LAST IN THE SEQUENCE

TRADES 500 GB

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST

NAMES 50 MB

SYMBOL	NAME
RIL	Reliance
NESTLEIND	Nestle

REVENUE 500 MB

TOTAL REVENUE	NAME	SYMBOL
$ 4 Billion	Reliance	RIL
$ 0.5 Billion	Nestle	NESTLEIND

SINCE TRADES IS THE LARGEST TABLE WE SHOULD TRY TO MAKE IT THE LAST IN THE SEQUENCE

join NAMES by Symbol, Revenue by Symbol, TRADES by Symbol

ON A SIDE NOTE

join NAMES by Symbol, TRADES by Symbol, Revenue by Symbol

HIVE CONVERTS JOINS OVER MULTIPLE TABLES INTO A SINGLE MAP/REDUCE JOB IF THE SAME FIELDS ARE USED IN THE JOIN CLAUSES

ON A SIDE NOTE

join NAMES by Symbol, TRADES by Symbol, Revenue by Symbol

HIVE CONVERTS JOINS OVER MULTIPLE TABLES INTO A SINGLE MAP/REDUCE JOB IF THE SAME FIELDS ARE USED IN THE JOIN CLAUSES

CURRENTLY, PIG DOESN'T DO THAT OPTIMISATION!

DIFFERENT JOIN OPERATIONS

LET'S CONSIDER 4 JOIN OPTIMISATIONS

JOINING MULTIPLE INPUTS

JOINING SMALL TO LARGE DATA

JOINING SKEWED DATA

JOINING SORTED DATA

JOINING SMALL TO LARGE DATA

WHEN WE JOIN A SMALL DATASET

WHICH CAN BE LOADED IN THE MEMORY

TO A LARGE DATASET

THAT CAN NOT

WE CAN ASK PIG TO USE FRAGMENT-REPLICATE ALGORITHM

FRAGMENT-REPLICATE ALGORITHM

SMALLER TABLE IS READ FROM HDFS AND IS COPIED ONTO EACH MAPPER' LOCAL DISK

(REPLICATED)

BIGGER TABLE IS DISTRIBUTED OVER THE MAPPERS

(FRAGMENTED)

FRAGMENT-REPLICATE ALGORITHM

SYMBOL	SERIES	OPEN	HIGH	LOW	CLOSE	LAST

TRADES

500 GB

NAMES

SYMBOL	NAME
RIL	Reliance
NESTLEIND	Nestle

50 MB

join NAMES by Symbol, TRADES by Symbol using 'replicated'

join NAMES by Symbol, TRADES by Symbol using 'replicated'

using 'replicated'

TELLS PIG TO USE

THE FRAGMENT-REPLICATE ALGORITHM

FRAGMENT-REPLICATE ALGORITHM

IS BASICALLY

MAP SIDE JOIN

MAP SIDE JOINS

A MAP SIDE JOIN IS A JOIN THAT IS PERFORMED AS A MAP ONLY JOB

THE MAP SIDE JOIN DOESN'T NEED A REDUCER

FRAGMENT-REPLICATE ALGORITHM

- IN THIS ALGORITHM, THE SMALL TABLE NEEDS TO BE READ JUST ONCE

FRAGMENT-REPLICATE ALGORITHM

- THE ALGORITHM HELPS MINIMIZE THE COST INCURRED FOR SORTING AND MERGING IN THE SHUFFLE AND REDUCE STAGES OF MAPREDUCE

FRAGMENT-REPLICATE ALGORITHM

- FRAGMENT-REPLICATE ALGORITHM DOESN'T WORK IN CASE OF RIGHT OUTER JOIN AND FULL OUTER JOIN

LEFT:BIGGER TABLE

RIGHT:SMALLER TABLE

FRAGMENT-REPLICATE'S FEASIBLITY DEPENDS ON THE TYPE OF JOIN

INNER JOIN 

LEFT:BIGGER TABLE

LEFT OUTER JOIN 

RIGHT:SMALLER TABLE

RIGHT OUTER JOIN 

FULL OUTER JOIN 

FRAGMENT-REPLICATE'S FEASIBLITY DEPENDS ON THE TYPE OF JOIN

INNER JOIN ✓

LEFT OUTER JOIN ✓

RIGHT OUTER JOIN ✗

FULL OUTER JOIN ✗

LEFT:SMALLER TABLE

RIGHT:BIGGER TABLE

DIFFERENT JOIN OPERATIONS

LET'S CONSIDER 4 JOIN OPTIMISATIONS

JOINING MULTIPLE INPUTS

JOINING SMALL TO LARGE DATA

JOINING SKEWED DATA

JOINING SORTED DATA

JOINING SKEWED DATA

DURING JOIN ALGORITHM

ALL THE RECORDS FOR A GIVEN KEY
IS COLLECTED ON A SINGLE REDUCER

JOINING SKEWED DATA

ALL THE RECORDS FOR A GIVEN KEY IS COLLECTED ON A SINGLE REDUCER

SUPPOSE WE JOIN ORDERS DATA WITH SOME CALENDAR DATA ON THE DATE KEY

ALL THE RECORDS FOR A GIVEN KEY IS COLLECTED ON A SINGLE REDUCER

SUPPOSE WE JOIN ORDERS DATA WITH SOME CALENDAR DATA ON THE DATE KEY

THE REDUCERS WHICH HANDLE MAJOR SALES DAYS WILL HAVE TO PROCESS A LOT OF DATA COMPARED TO OTHER REDUCERS

SUPPOSE WE ARE JOINING ORDERS DATA WITH SOME CALENDAR DATA ON THE DATE KEY

THE REDUCERS WHICH HANDLE MAJOR SALES DAYS WILL HAVE TO PROCESS A LOT OF DATA COMPARED TO OTHER REDUCERS

PIG CAN'T START THE NEXT JOB UNTIL ALL THE REDUCERS HAVE FINISHED THE PREVIOUS JOB

PIG CAN'T START THE NEXT JOB UNTIL ALL THE REDUCERS HAVE FINISHED THE PREVIOUS JOB

SLOWEST REDUCER DEFINES THE LENGTH OF THE JOB

THAT IS WHY IS IMPORTANT TO HANDLE SKEW

JOINING SKEWED DATA

TO DEAL WITH THIS

PIG PROVIDES SKEW JOIN

```
join orders by Order_Date, calendar by Date using 'skewed'
```

join orders by Order_Date, calendar by Date using 'skewed'

using 'skewed'

TELLS PIG TO USE

SKEW JOIN

SKEW JOIN

IT DOES TWO THINGS

FIRST IT SAMPLES ONE INPUT FOR THE JOIN

IT TRIES TO IDENTIFY THOSE KEYS FOR WHICH THE DATA IS TOO BIG TO LOAD IN MEMORY

LET THESE KEYS BE X

SKEW JOIN

IT TRIES TO IDENTIFY THOSE KEYS FOR WHICH THE DATA IS TOO BIG TO LOAD IN MEMORY

LET THESE KEYS BE X

SECOND

FOR ALL THE KEYS APART FROM X, IT EXECUTES STANDARD JOIN

SKEW JOIN

IT TRIES TO IDENTIFY THOSE KEYS FOR WHICH THE DATA IS TOO BIG TO LOAD IN MEMORY

LET THESE KEYS BE X

SECOND

FOR ALL KEYS APART FROM X, IT EXECUTES STANDARD JOIN

BUT FOR THE KEYS BELONGING TO X, DEPENDING ON THE NUMBER OF RECORDS PER KEY AND AMOUNT OF DATA THAT A REDUCER CAN HANDLE, RECORDS ARE SPLIT ACROSS REDUCERS

SKEW JOIN

BUT FOR THE KEYS BELONGING TO X, DEPENDING ON THE NUMBER OF RECORDS PER KEY AND AMOUNT OF DATA THAT A REDUCER CAN HANDLE, RECORDS ARE SPLIT ACROSS REDUCERS

SKEW JOIN

BUT FOR THE KEYS BELONGING TO X, DEPENDING ON THE NUMBER OF RECORDS PER KEY AND AMOUNT OF DATA THAT A REDUCER CAN HANDLE, RECORDS ARE SPLIT ACROSS REDUCERS

REMEMBER THESE ARE KEYS WHOSE DATA IS TOO BIG TO FIT IN MEMORY

SKEW JOIN

BUT FOR THE KEYS BELONGING TO X, DEPENDING ON THE NUMBER OF RECORDS PER KEY AND AMOUNT OF DATA THAT A REDUCER CAN HANDLE, RECORDS ARE SPLIT ACROSS REDUCERS

SKEW JOIN

BUT FOR THE KEYS BELONGING TO X, DEPENDING ON THE NUMBER OF RECORDS PER KEY AND AMOUNT OF DATA THAT A REDUCER CAN HANDLE, RECORDS ARE SPLIT ACROSS REDUCERS

SKEW JOIN

BUT FOR THE KEYS BELONGING TO X, DEPENDING ON THE NUMBER OF RECORDS PER KEY AND AMOUNT OF DATA THAT A REDUCER CAN HANDLE, RECORDS ARE SPLIT ACROSS REDUCERS

SKEW JOIN

IT TRIES TO IDENTIFY THOSE KEYS FOR WHICH THE DATA IS TOO BIG TO LOAD IN MEMORY

LET THESE KEYS BE X

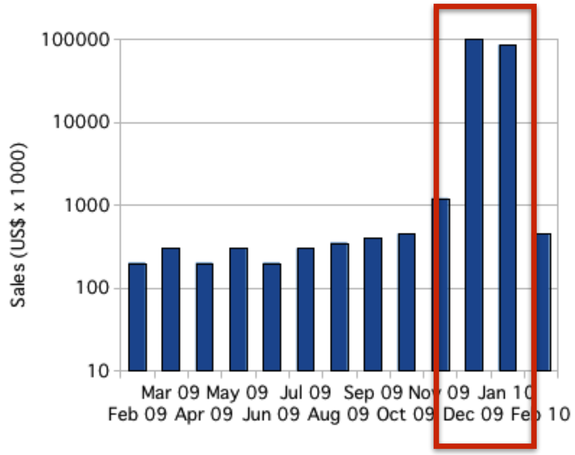
SECOND

FOR ALL KEYS APART FROM X, IT EXECUTES STANDARD JOIN

BUT FOR THE KEYS BELONGING TO X, DEPENDING ON THE NUMBER OF RECORDS PER KEY AND AMOUNT OF DATA THAT A REDUCER CAN HANDLE, RECORDS ARE SPLIT ACROSS REDUCERS

SKEW JOIN

LET US ASSUME THE JOIN FOR ORDERS IS DONE ON THE MONTH KEY



FIRST STEP

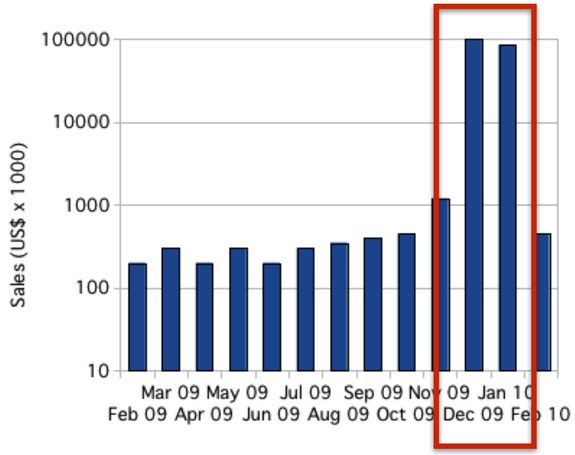
IT TRIES TO IDENTIFY THOSE KEYS FOR WHICH THE DATA IS TOO BIG TO LOAD IN MEMORY

DATA IS TOO BIG FOR KEY= DEC, JAN

SKEW JOIN

LET US ASSUME THE JOIN FOR ORDERS IS DONE ON MONTH KEY

DATA IS TOO BIG FOR KEY= DEC,JAN

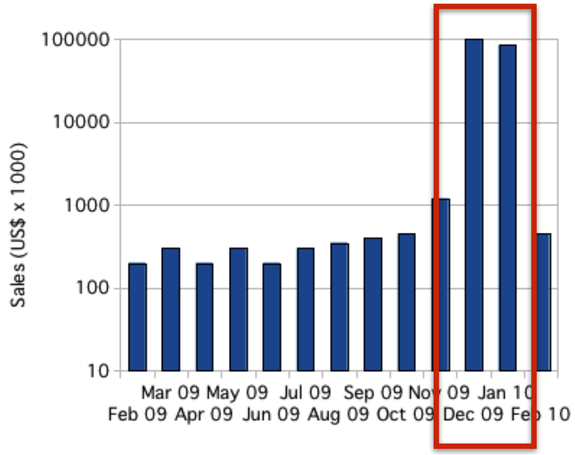


SECOND

FOR ALL KEYS APART FROM (DEC, JAN), IT EXECUTES STANDARD JOIN

SKEW JOIN

LET US ASSUME THE JOIN FOR ORDERS IS DONE ON MONTH KEY



DATA IS TOO BIG FOR KEY= DEC,JAN

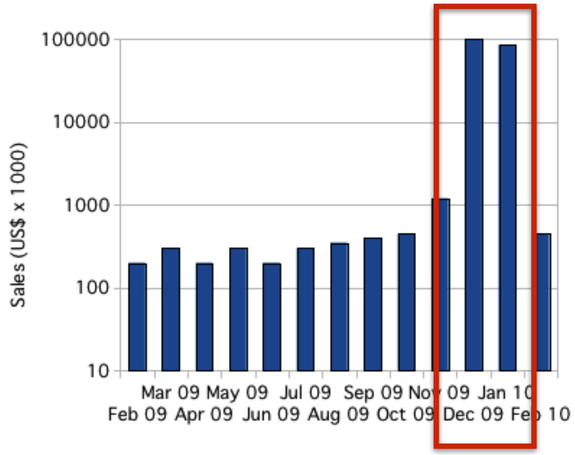
FOR ALL KEYS APART FROM (DEC,JAN), IT EXECUTES STANDARD JOIN

LET US ASSUME EACH REDUCER CAN HANDLE 5×10^5 RECORDS

AND JAN AND DEC EACH HAS 10×10^6

SKEW JOIN

DATA IS TOO BIG FOR KEY= DEC, JAN



LET US ASSUME EACH REDUCER CAN HANDLE 5×10^5 RECORDS
AND JAN AND DEC EACH HAS 10×10^6

NUMBER OF REDUCERS AMONG WHICH THE RECORDS WILL BE SPLIT IS $(10 \times 10^6)/(5 \times 10^5)$

```
number of reducers=20
```

SKEW JOIN

A SIMILAR ISSUE MIGHT ARISE WHILE GROUPING

PIG HANDLES THIS PROBLEM USING THE COMBINER PHASE

PIG OPERATORS AND BUILT-IN FUNCTIONS
USE COMBINERS WHENEVER POSSIBLE

BECAUSE OF ITS SKEW-REDUCING PROPERTIES

SKEW JOIN

A SIMILAR ISSUE MIGHT ARISE WHILE GROUPING

PIG HANDLES THIS PROBLEM USING THE COMBINER PHASE

COMBINERS BRING TOGETHER VALUES WHICH HAVE THE SAME KEY JUST AFTER THE MAP PHASE

BEFORE DATA IS SENT TO THE REDUCERS

BEFORE DATA IS SENT TO THE REDUCERS

DIFFERENT JOIN OPERATIONS

LET'S CONSIDER 4 JOIN OPTIMISATIONS

JOINING MULTIPLE INPUTS

JOINING SMALL TO LARGE DATA

JOINING SKEWED DATA

JOINING SORTED DATA

JOINING SORTED DATA

IF BOTH COLUMNS (ON WHICH JOINS ARE HAPPENING) ARE SORTED, IT BECOMES EASIER TO JOIN THEM

THIS IS CALLED SORT MERGE JOIN

SORT MERGE JOIN

STUDENT ID	NAME	SUBJECT CODE
2	ABC	101
3	CDE	99
1	FGH	44

STUDENTS

SUBJECT CODE	DESC
101	CHEMISTRY
99	PHYSICS
44	MATHS

SUBJECT_DESCRIPTION

SORTING

STUDENT ID	NAME	SUBJECT CODE
1	FGH	44
3	CDE	99
2	ABC	101

STUDENTS

SUBJECT CODE	DESC
44	MATHS
99	PHYSICS
101	CHEMISTRY

SUBJECT_DESCRIPTION

SORT MERGE JOIN

STUDENT ID	NAME	SUBJECT_CODE
2	ABC	101
3	CDE	99
1	FGH	44

STUDENTS

SUBJECT_CODE	DESC
101	CHEMISTRY
99	PHYSICS
44	MATHS

SUBJECT_DESCRIPTION

SORTING

STUDENT ID	NAME	SUBJECT_CODE
1	FGH	44
3	CDE	99
2	ABC	101

STUDENTS

SUBJECT_CODE	DESC
44	MATHS
99	PHYSICS
101	CHEMISTRY

SUBJECT_DESCRIPTIONN

MERGING

STUDENT ID	NAME	SUBJECT_CODE	DESC
1	FGH	44	MATHS
3	CDE	99	PHYSICS
2	ABC	101	CHEMISTRY

IF THE INPUTS ARE ALREADY SORTED, IT MAKES SENSE TO USE

SORT MERGE JOIN

join Students by Subject_Code, Subject_Description by Subject_Code using 'merge'

STUDENT ID	NAME	SUBJECT CODE
1	FGH	44
3	CDE	99
2	ABC	101

STUDENTS

SUBJECT CODE	DESC
44	MATHS
99	PHYSICS
101	CHEMISTRY

SUBJECT_DESCRIPTION

join Students by Subject_Code, Subject_Description by Subject_Code using 'merge'

using 'merge'

TELLS PIG TO USE

SORT-MERGE JOIN

THERE ARE NUMBER OF WAYS IN WHICH YOU CAN OPTIMISE PIG LATIN SCRIPTS

THESE FALL BROADLY INTO 2 CATEGORIES

SPECIFIC COMMANDS/KEYWORDS

TRICKS/TACTICAL CHANGES (COMMON SENSICAL)
CAN BE USED IN ANY SCENARIO

TRICKS/TACTICAL CHANGES (COMMON SENSICAL)

1. USE APPROPRIATE DATATYPES

SPECIFYING THE REAL DATA TYPE SPEEDS UP ARITHMETIC COMPUTATION

IT IS BETTER TO USE SMALLER DATA TYPES IF THEY SATISFY THE REQUIREMENTS

TRICKS/TACTICAL CHANGES (COMMON SENSICAL)

2. PROJECT EARLY AND OFTEN

IF YOU DON'T NEED A FIELD, DROP IT

THIS WILL REDUCE THE AMOUNT OF DATA THAT IS CARRIED THROUGH TO THE MAP AND REDUCE PHASE

TRICKS/TACTICAL CHANGES (COMMON SENSICAL)

3. FILTER EARLY AND OFTEN

FILTER REDUCES THE AMOUNT OF DATA THAT IS BEING PROCESSED

THIS WILL REDUCE THE AMOUNT OF DATA THAT IS CARRIED THROUGH TO THE MAP AND REDUCE PHASE

4. REDUCE THE NUMBER OF OPERATIONS

COMBINE FOREACH STATEMENTS WHEREVER POSSIBLE

TRICKS/TACTICAL CHANGES (COMMON SENSICAL)

5. USE THE LIMIT OPERATOR

WHENEVER WE ARE NOT INTERESTED IN FULL DATA AND ARE ONLY FOCUSED ON TOP ROWS, USE LIMIT OPERATOR

THIS WILL REDUCE THE AMOUNT OF DATA

TRICKS/TACTICAL CHANGES (COMMON SENSICAL)

6. PREFER DISTINCT OVER GROUP BY

WHEN YOU ARE TRYING TO EXTRACT UNIQUE VALUES
USE DISTINCT

DISTINCT IS FASTER AND MORE EFFICIENT

TRICKS/TACTICAL CHANGES (COMMON SENSICAL)

7. DROP NULLS BEFORE A JOIN

```
join orders by Customer_ID, customers by Customer_ID
```

IN PIG, JOIN IS REWRITTEN LIKE THIS

```
temp = cogroup orders by Customer_ID, customers by Customer_ID
foreach temp generate flatten(orders), flatten(customers)
```

7. DROP NULLS BEFORE A JOIN

```
join orders by Customer_ID, customers by Customer_ID
```

IN PIG, JOIN IS REWRITTEN LIKE THIS

```
temp = cogroup orders by Customer_ID, customers by Customer_ID
foreach temp generate flatten(orders), flatten(customers)
```

WORKS WITH NULL

RETURNS NULL WITH EMPTY BAGS

7. DROP NULLS BEFORE A JOIN

IN PIG, JOIN IS REWRITTEN LIKE THIS

```
temp = cogroup orders by Customer_ID, customers by Customer_ID
foreach temp generate flatten(orders), flatten(customers)
```

WORKS WITH NULL

RETURNS NULL WITH EMPTY BAGS

EVENTUALLY IN JOINS NULLS WILL BE DROPPED

DROP THEM BEFORE THE COGROUP OPERATION!

7. DROP NULLS BEFORE A JOIN

EVENTUALLY IN JOINS NULLS WILL BE DROPPED

DROP THEM BEFORE THE COGROUP OPERATION!

In one test where the key was null 7% of the time and the data was spread across 200 reducers, we saw a about a 10x speed up in the query by adding the early filters.

PIG APACHE WEBSITE