clear
help
history
quit

exec [-param parameter-name = perm-value] [-param parameter-name = perm-value] script.
or
run

run or exec | ~ .pig

| diff exec & run
run - commands in history.

Execution in batch mode

```
$ pig -x local simple-script.pig
$ pig -x mapReduce simple-script.pig
```

A.
operators

+	
−	
*	
/	division
%	modulus

Case when Then else end

```
Case f2 % 2
    when 0 Then
    'even'
    when 1 then
    "odd'
end.
```

Comparision

==	
!=	matches: Pattern match.
>=	
<=	
>	
<	

loading (or) reading data.

relationname → name to store data.

= load "filepath" using function as schema

"filepath": Hdfs / local based on Pig execution mode

function: load function
- BinStorage
- JsonLoader
- PigStorage
- TextLoader

schema = (col1, col2)
(or)
(col1: datatype, col2: dtype)

without schema, colours $01, $02, $03, $04, ---

Dump A → statment (relation)
↳ execute (run) piglatin statement & display output

Storing data:
-- single line comment
/* */ → multiline comment

Store Relation_name into 'path/location' [using function];

Store ~~Relatio~~ A into "/~~~/(~)" using Pig Storage.
automatically create dir.

Dignostic Operator:
↳ load → Just load data to specified relation
↳ to very load we us Dignostic operators.

4 different types of dignostic operators:
* Dump Operator
* Describe operator
* Explanation Operator
* Illustration Operator.

Dump Operator → run latin statement & give or display output on screen.
* used for debugging purpose.

Dump relation_name;

Describe operator → view schema of relation.
Describe relation name;
Explan operator → show(or) disblay logical, physical, Mapreduce execution plans of a relation.

illustrate operation → give step by step execution of sequence of statements

Group Operator

Group:- group the data in one (or) more statements.

Grouped_data = GROUP relation_name BY age;
↓
feild.

Group_multi_col_data = GROUP relation_name BY (age, city);

to group all columns
group by all columns

group_all = GROUP relation_name All;

—

Cogroup
→ Same like group operator
→ ~~Cog~~ Group work on → 1 relation
  CoGroup → 2 (or) more relations

CoGroup_data = Cogroup A1 by C1, A2 by C2;

(~ , { —— }, { (), () })

Join

Self-Join
Inner-Join
outer-Join - (left Join, right Join, full Join)

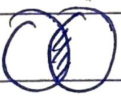
2 relations. customers = (id: int, name: chararray,
age: int, address: chararray,
salary: int);
orders = (oid: int, date: chararray,
customer_id: int, amount: int)

Self Join: Join table itself. → used when we want to compare data with itself.
* create 2 relations with same data, schema.

Syntax
$ relation_3 = Join Relation_1 by key, Relation2 by key
↳ Join customers_1 by id, customers_2 by id.

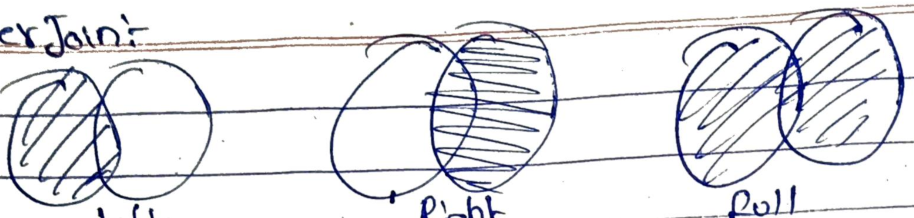
both same but Just renamed

equiJoin
Inner Join:



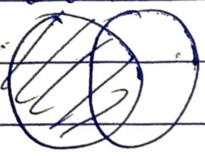
Customer_orders = Join Customers by Id, orders by customer_id.

OuterJoin:-

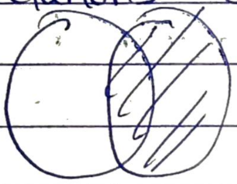


retornall rows atleastone relation.

leftouter
Relations= Join customers by ID left outer,
orders by customer_id;



Right outer
Relations= Join customers by ID Right Outer,
orders by customerID



FullouterJoin → return $\overset{\text{rowthat match}}{}$ atleast one relation.

Relation = Join customers by ID Full outer,
orders by columnID

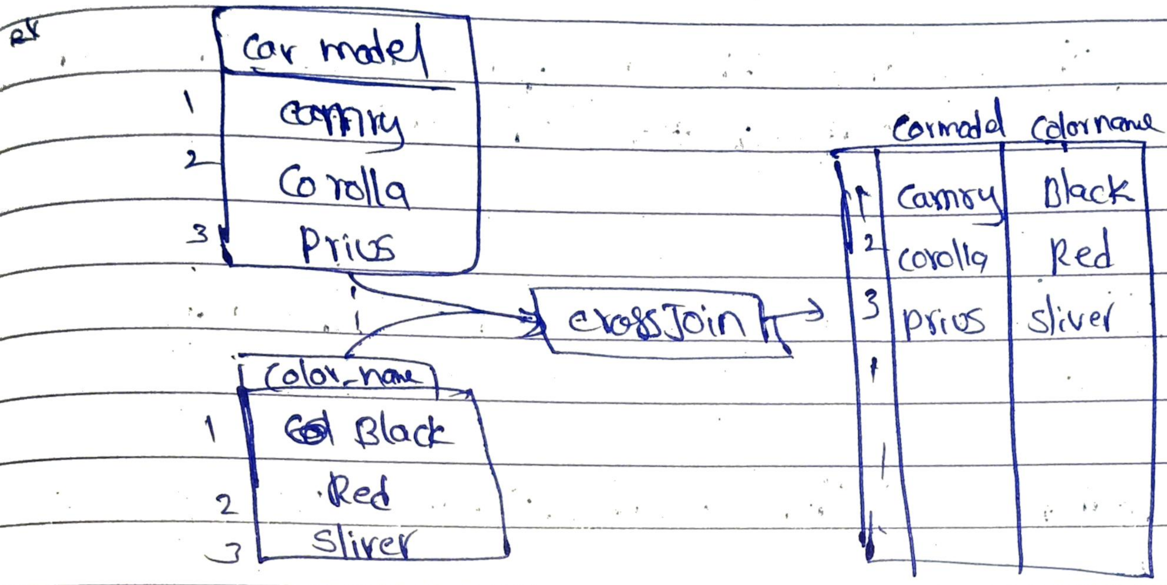
Join operation based on multiple keys:-

syntax
Relation3= Join $\overset{\text{employee}}{\text{~~customer~~}}$ by $\overset{\text{key1, key2}}{(\text{ID, JobID})}$,
employee_contact by (ID, JobID);

Join
or
cross operator:- —— cross product of two or more relations

↳ combine eachrow of one table ~~w~~ with another row of another table.

ex



	Car model
1	camry
2	Corolla
3	Prius

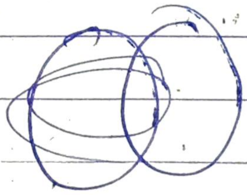
	Color_name
1	Black
2	Red
3	Sliver

	Carmodel	Colorname
1	Camry	Black
2	corolla	Red
3	prius	sliver

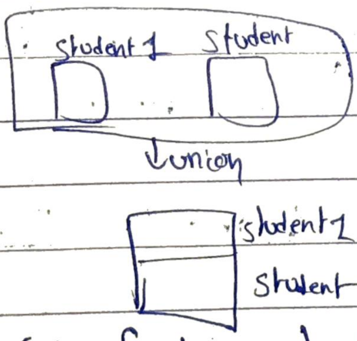
c

Relation3 = Cross Relation1, Relation2

union operator —— combining
merge two or more relations



ex



Student 1 = (id, firstname, lastname, phone, city) - 50 record
Student 2 = (id, firstname, lastname, phone, city) - 50
s3 = union Student 1, Student 2; ⟶ 100 records

Split → divide the relation into two or more.

spliting into 1

var value

split relation into new_relation_name if age>18

split ~~input~~relation into ~~new~~output_relation1 if condition1,
output_relation2 if condition2;

filter → to select required tuples ~~based on~~ from a relation based on condition.

Relation2 = Filter Relation_1_name by condition;

Distinct → remove the duplicate tuples from ~~tuples~~ relation

relation2 = Distinct Relation1;

for each → generate specific data transformations based on column data,

Relation_2 = ForEach Relation1 Generate columnnames;

to select id, age, city - foreach student

Relation3 = ForEach student_details Generate id, age, city;

order by ($\rightarrow$ sort by ~~columns~~ based on one or more fields)
order relation-name by column (ASC|DESC);
age DESC;

limit $\rightarrow$ select Top n num of Tuples
limit_data = LIMIT student_details 4;
Top 4 tuples

converting datatype
✓ date1-int
(chararray) date1

AVG
CONCAT.
COUNT
IN
MAX
MIN
SIZE
SUM
TOKENIZE

EVAL (evaluate) functions:-

~~Avg~~

Forfach relation Generate
Avg( );

AVG( ) → preciding use group all or group by

CONCAT(~~expres~~ firstnam, lastname).
↳ 2 or more expressions

Count(Age) — not include nullvalues

Count_STAR( ) - Include nullvalue

student_details
: Student_group_all = Group student_details All;

studentcount = FOREACH student_group_all Generate
COUNT_STAR(Student_details.
gpa)

DIFF(exp1, exp2) — compare bags
exp1 tuple1 = exp2 tuple2 means {} - empty bag
≠ return both