PIG the 2nd

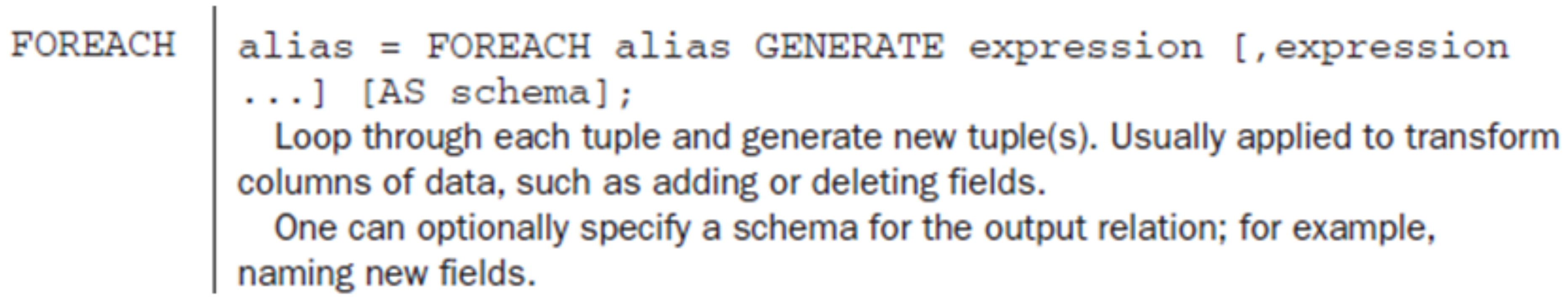
Arend Hintze

relational operators

- FOREACH
- FILTER
- ORDER BY
- SPLIT
- UNION
- DISTINCT
- GROUP
- JOIN

dataset:
A,1,2,3,m
B,1,2,3,m
C,2,2,2,f
…

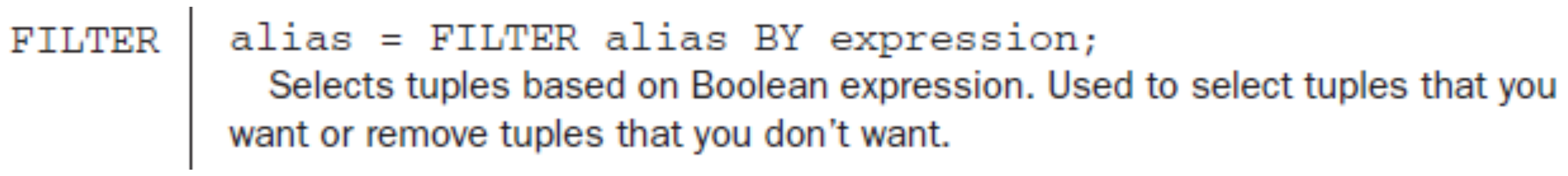
FOREACH



FOREACH	`alias = FOREACH alias GENERATE expression [,expression ...] [AS schema];` Loop through each tuple and generate new tuple(s). Usually applied to transform columns of data, such as adding or deleting fields. One can optionally specify a schema for the output relation; for example, naming new fields.

```
data = LOAD ‘grades.csv’ using PigStorage(‘,’) AS
           (name,g1:int,g2:int,g3:int,gender);
sums = FOREACH data GENERATE name,g1+g2+g3;
DUMP sums;
```

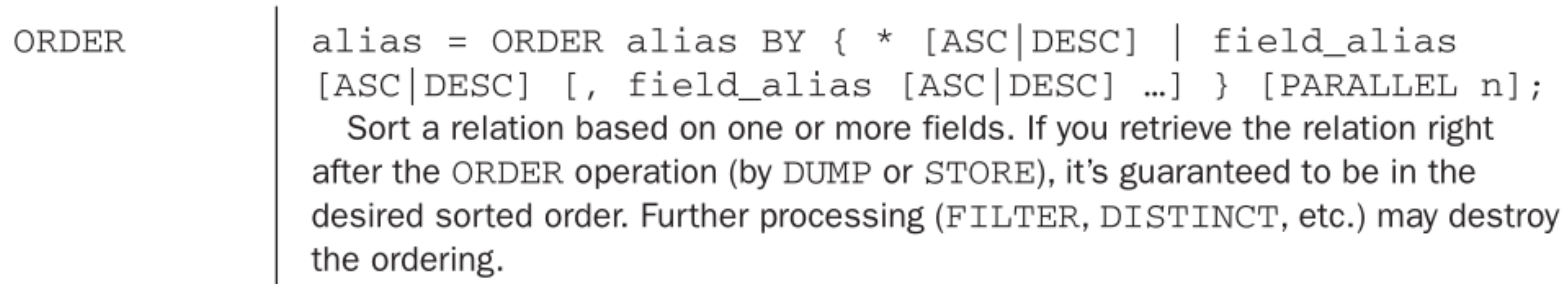
FILTER



FILTER	`alias = FILTER alias BY expression;` Selects tuples based on Boolean expression. Used to select tuples that you want or remove tuples that you don't want.

```
data = LOAD ‘grades.csv’ using PigStorage(‘,’) AS
           (name,g1:int,g2:int,g3:int,gender);
goodOnes = FILTER data BY g2>10;
DUMP goodOnes;
```

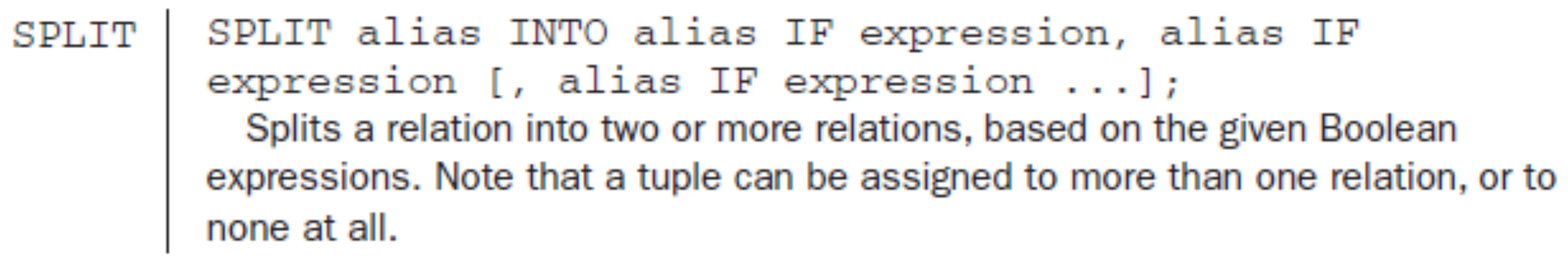
ORDER BY



ORDER	`alias = ORDER alias BY { * [ASC|DESC] | field_alias [ASC|DESC] [, field_alias [ASC|DESC] …] } [PARALLEL n];` Sort a relation based on one or more fields. If you retrieve the relation right after the ORDER operation (by DUMP or STORE), it's guaranteed to be in the desired sorted order. Further processing (FILTER, DISTINCT, etc.) may destroy the ordering.

```
data = LOAD ‘grades.csv’ using PigStorage(‘,’) AS
           (name,g1:int,g2:int,g3:int,gender);
myOrder = ORDER data BY name DESC;
DUMP myOrder;
```

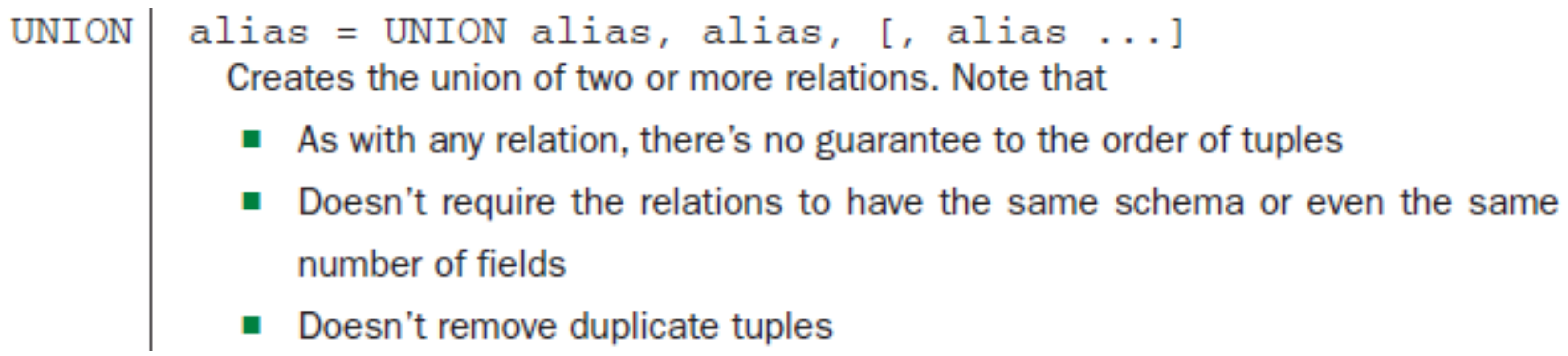
SPLIT



SPLIT	`SPLIT alias INTO alias IF expression, alias IF expression [, alias IF expression ...];` Splits a relation into two or more relations, based on the given Boolean expressions. Note that a tuple can be assigned to more than one relation, or to none at all.

```
data = LOAD ‘grades.csv’ using PigStorage(‘,’) AS
           (name,g1:int,g2:int,g3:int,gender);
sums = FOREACH data GENERATE name,g1+g2+g3;
SPLIT sums INTO high if $1>100, low if $1<=100;
DUMP low;
DUMP high;
```

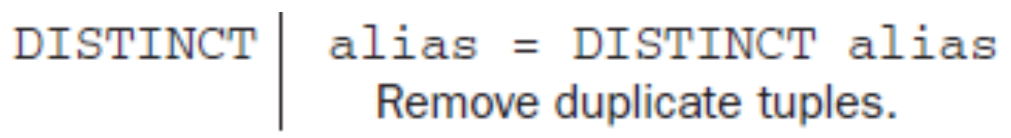
UNION



UNION	`alias = UNION alias, alias, [, alias ...]` Creates the union of two or more relations. Note that ■ As with any relation, there's no guarantee to the order of tuples ■ Doesn't require the relations to have the same schema or even the same number of fields ■ Doesn't remove duplicate tuples

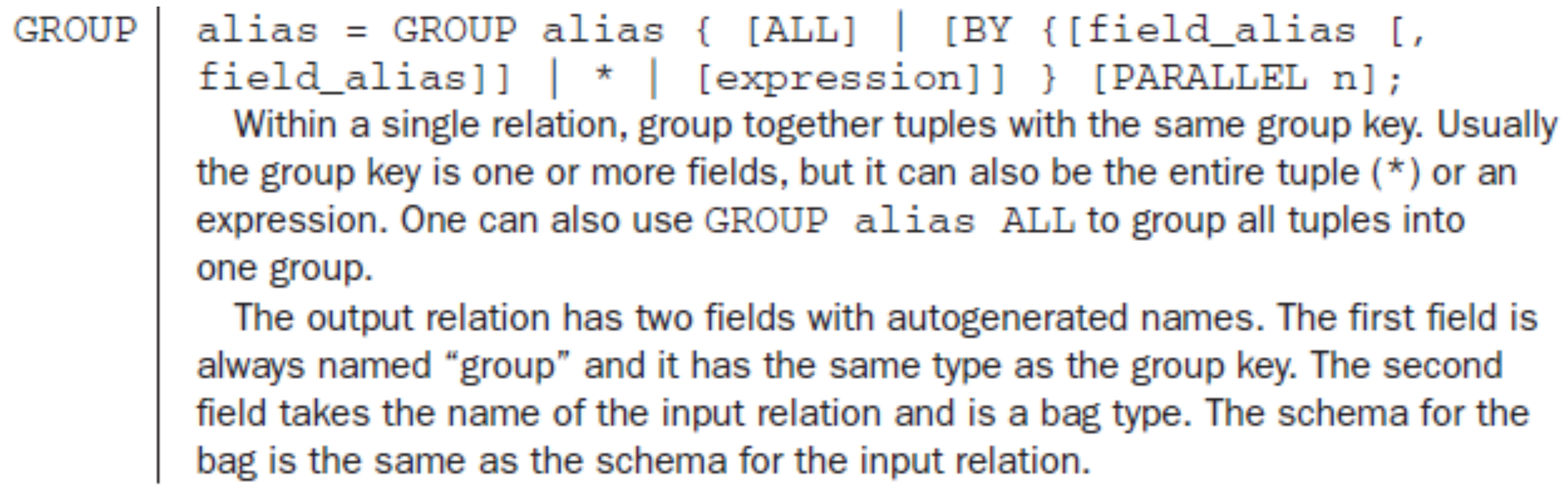
```
data = LOAD ‘grades.csv’ using PigStorage(‘,’) AS
           (name,g1:int,g2:int,g3:int,gender);
sums = FOREACH data GENERATE name,g1+g2+g3;
SPLIT sums INTO high if $1>100, low if $1<=100;
DUMP low;
DUMP high;
myU = UNION low,high;
DUMP myU;
```

DISTINCT



`DISTINCT`	`alias = DISTINCT alias` Remove duplicate tuples.

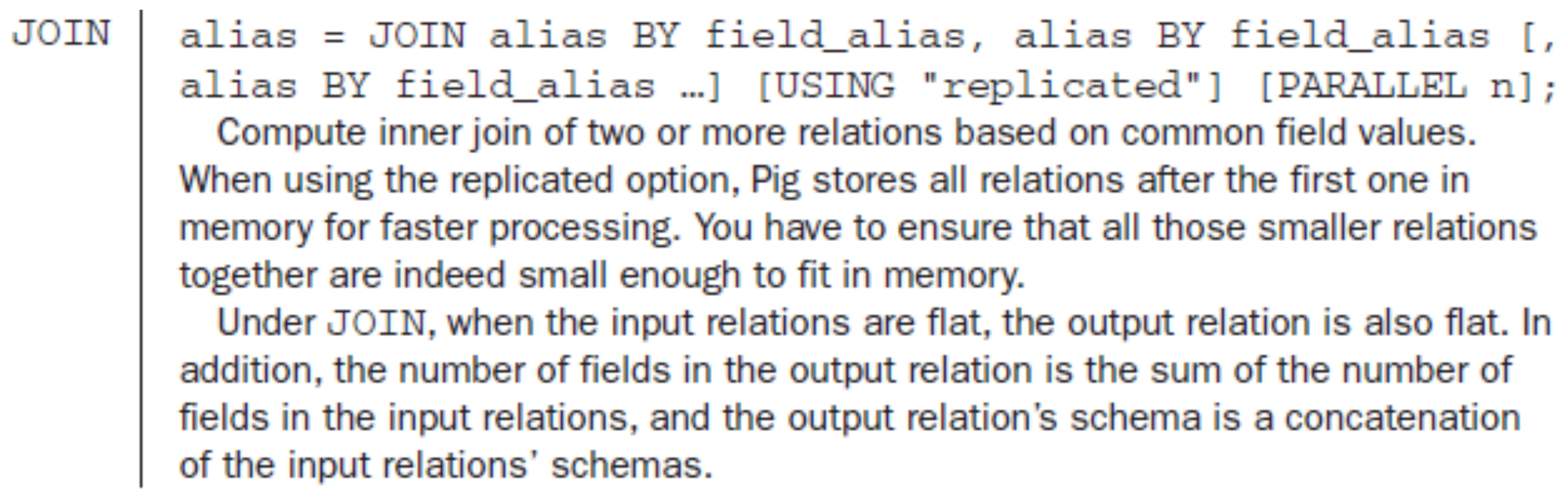
GROUP



GROUP	`alias = GROUP alias { [ALL] | [BY {[field_alias [, field_alias]] | * | [expression]] } [PARALLEL n];` Within a single relation, group together tuples with the same group key. Usually the group key is one or more fields, but it can also be the entire tuple (*) or an expression. One can also use `GROUP alias ALL` to group all tuples into one group. The output relation has two fields with autogenerated names. The first field is always named "group" and it has the same type as the group key. The second field takes the name of the input relation and is a bag type. The schema for the bag is the same as the schema for the input relation.

```
data = LOAD ‘grades.csv’ using PigStorage(‘,’) AS
           (name,g1:int,g2:int,g3:int,gender);
genders = GROUP data BY gender;
DUMP genders;
```

JOIN (inner join)



JOIN

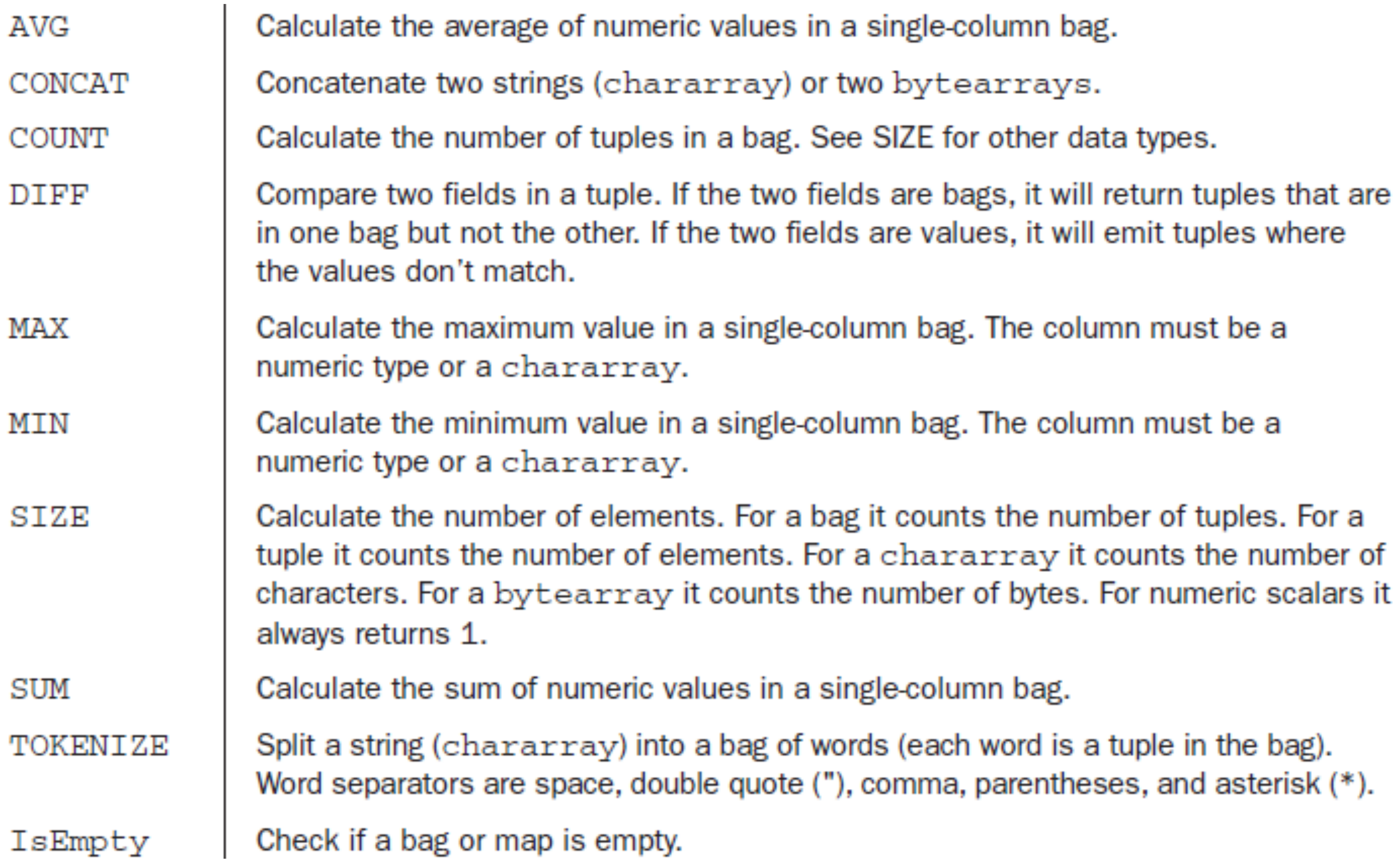
```
alias = JOIN alias BY field_alias, alias BY field_alias [,
alias BY field_alias …] [USING "replicated"] [PARALLEL n];
```

Compute inner join of two or more relations based on common field values. When using the replicated option, Pig stores all relations after the first one in memory for faster processing. You have to ensure that all those smaller relations together are indeed small enough to fit in memory.

Under JOIN, when the input relations are flat, the output relation is also flat. In addition, the number of fields in the output relation is the sum of the number of fields in the input relations, and the output relation's schema is a concatenation of the input relations' schemas.

```
dataA = LOAD ‘gradesA.csv’ using PigStorage(‘,’) AS
           (name,g1:int,g2:int,g3:int,gender);
dataB = LOAD ‘gradesB.csv’ using PigStorage(‘,’) AS
           (name,g1:int,g2:int,g3:int,gender);
j = JOIN dataA BY name,dataB BY name;
```

Built-In Functions



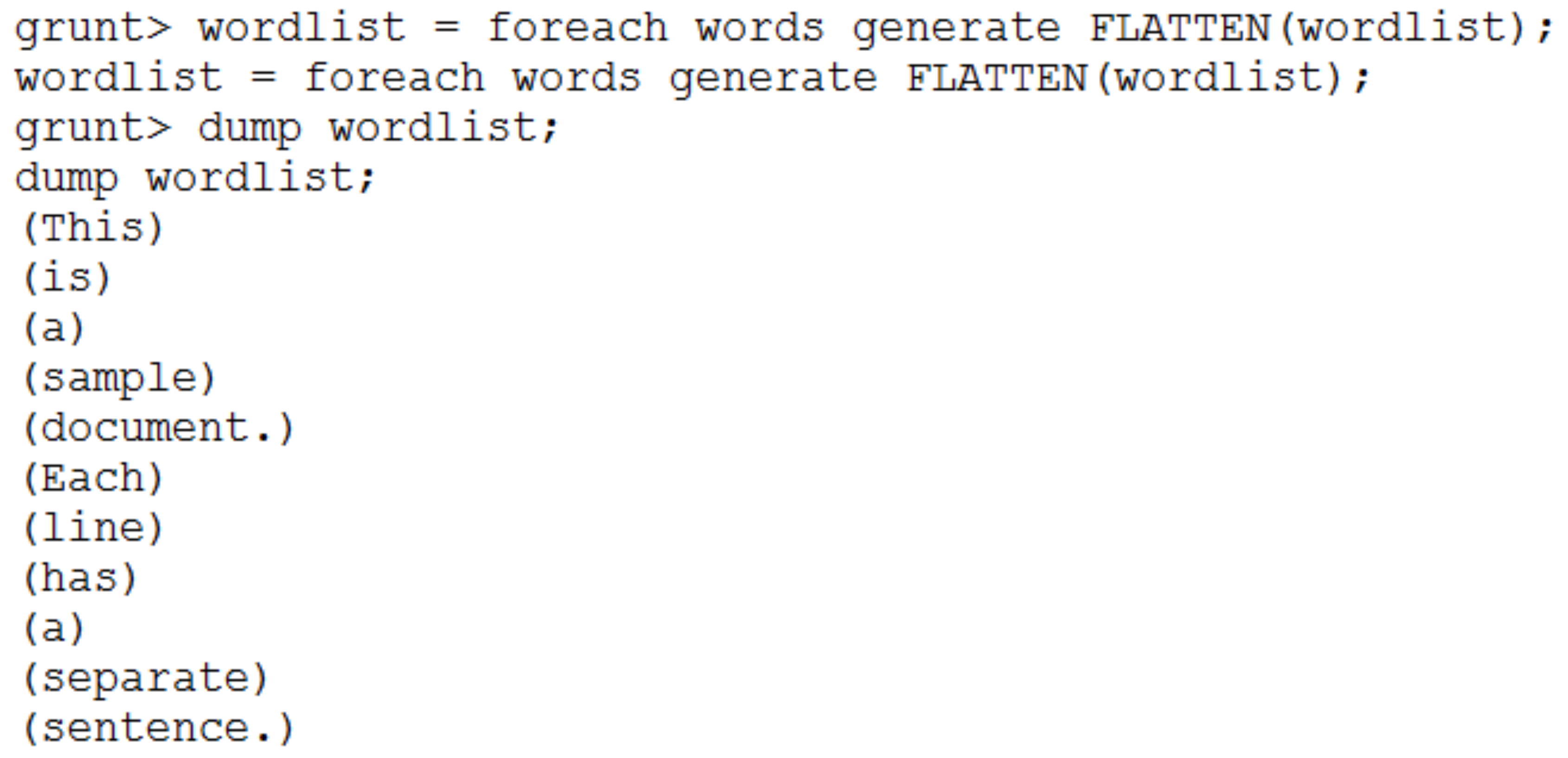
AVG	Calculate the average of numeric values in a single-column bag.
CONCAT	Concatenate two strings (chararray) or two bytearrays.
COUNT	Calculate the number of tuples in a bag. See SIZE for other data types.
DIFF	Compare two fields in a tuple. If the two fields are bags, it will return tuples that are in one bag but not the other. If the two fields are values, it will emit tuples where the values don't match.
MAX	Calculate the maximum value in a single-column bag. The column must be a numeric type or a chararray.
MIN	Calculate the minimum value in a single-column bag. The column must be a numeric type or a chararray.
SIZE	Calculate the number of elements. For a bag it counts the number of tuples. For a tuple it counts the number of elements. For a chararray it counts the number of characters. For a bytearray it counts the number of bytes. For numeric scalars it always returns 1.
SUM	Calculate the sum of numeric values in a single-column bag.
TOKENIZE	Split a string (chararray) into a bag of words (each word is a tuple in the bag). Word separators are space, double quote ("), comma, parentheses, and asterisk (*).
IsEmpty	Check if a bag or map is empty.

CASE sensitive!

FLATTEN

- flattens a nested datatype (bags for example)

  list of Bags -> list of all Bag elements

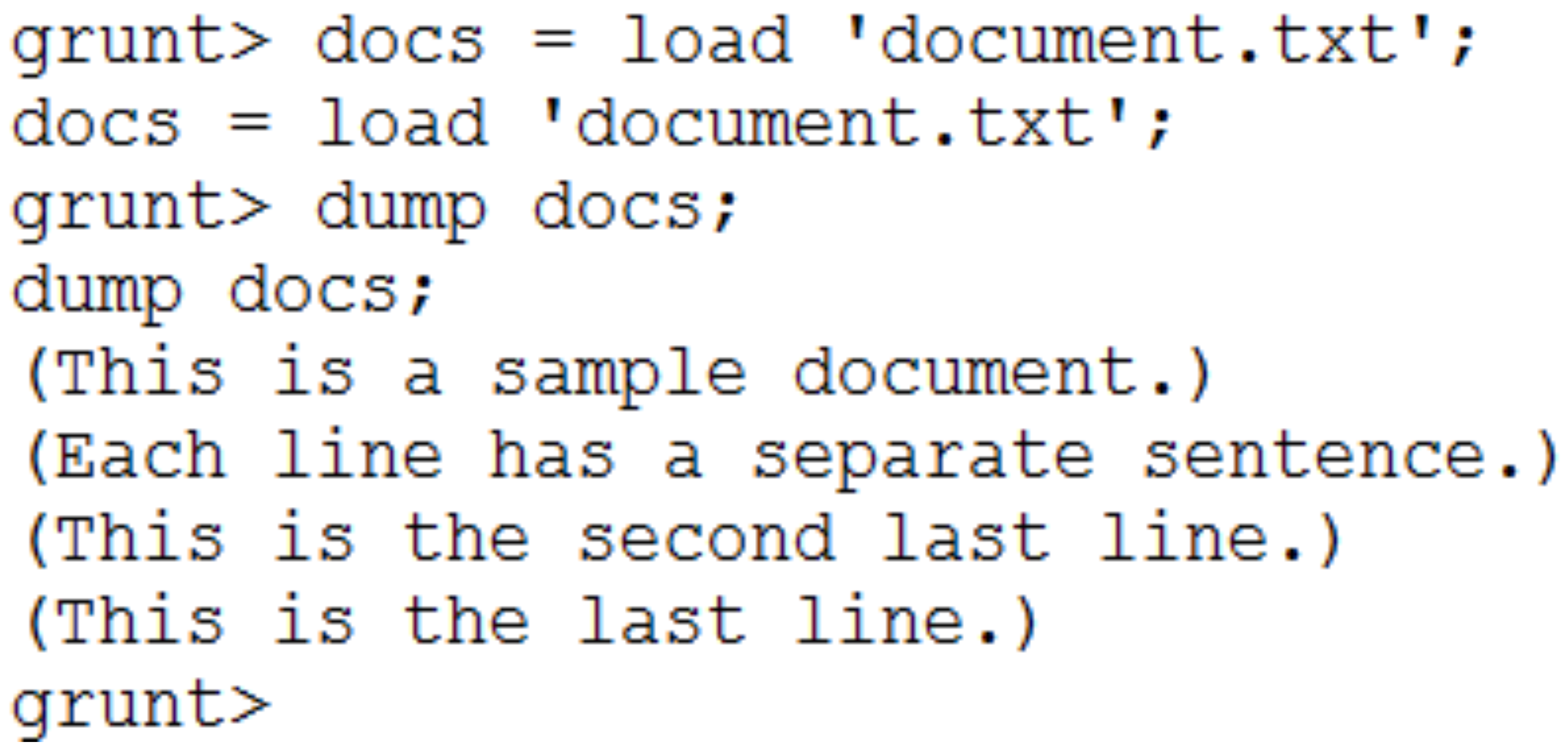


```
grunt> wordlist = foreach words generate FLATTEN(wordlist);
wordlist = foreach words generate FLATTEN(wordlist);
grunt> dump wordlist;
dump wordlist;
(This)
(is)
(a)
(sample)
(document.)
(Each)
(line)
(has)
(a)
(separate)
(sentence.)
```

WordCount in PIG

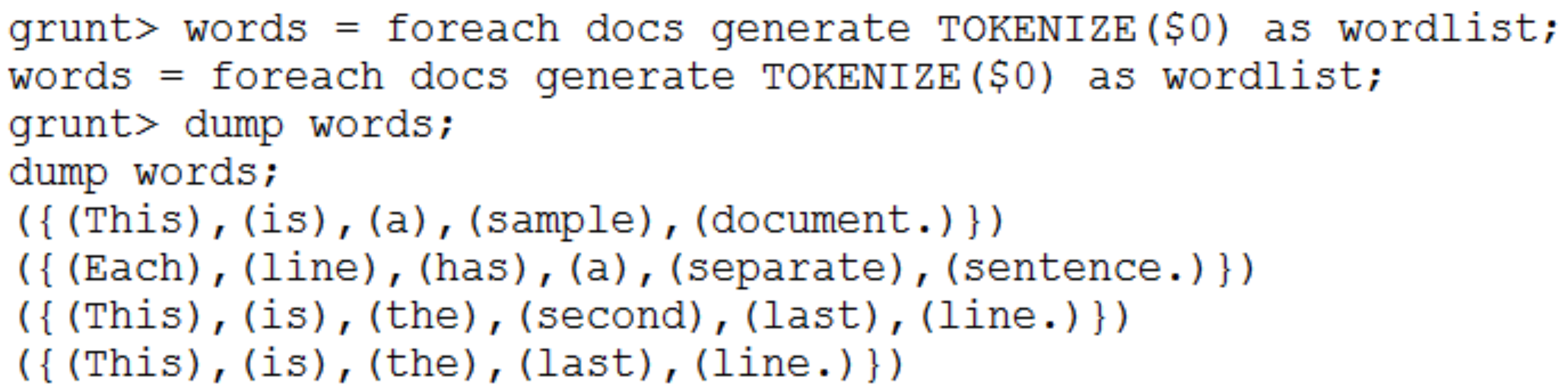
```
data = LOAD ‘file0’;
words = FOREACH data GENERATE TOKENIZE($0) AS wordlist;
allwords = FOREACH words GENERATE FLATTEN(wordlist),1;
grp = GROUP allwords BY $0;
counts = FOREACH grp GENERATE $0,SUM($1.$1);
DUMP counts;
```

Step 1:



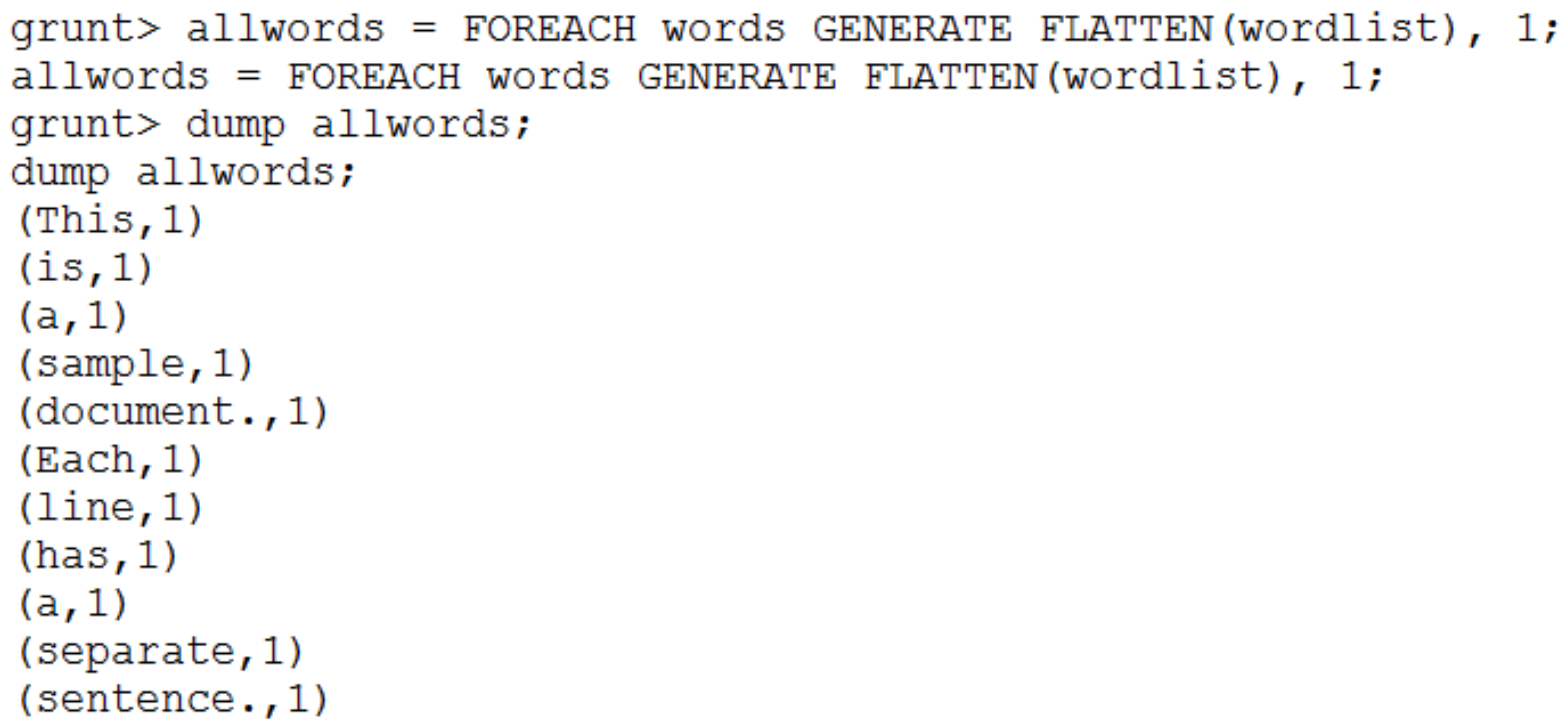
```
grunt> docs = load 'document.txt';
docs = load 'document.txt';
grunt> dump docs;
dump docs;
(This is a sample document.)
(Each line has a separate sentence.)
(This is the second last line.)
(This is the last line.)
grunt>
```

Step 2:



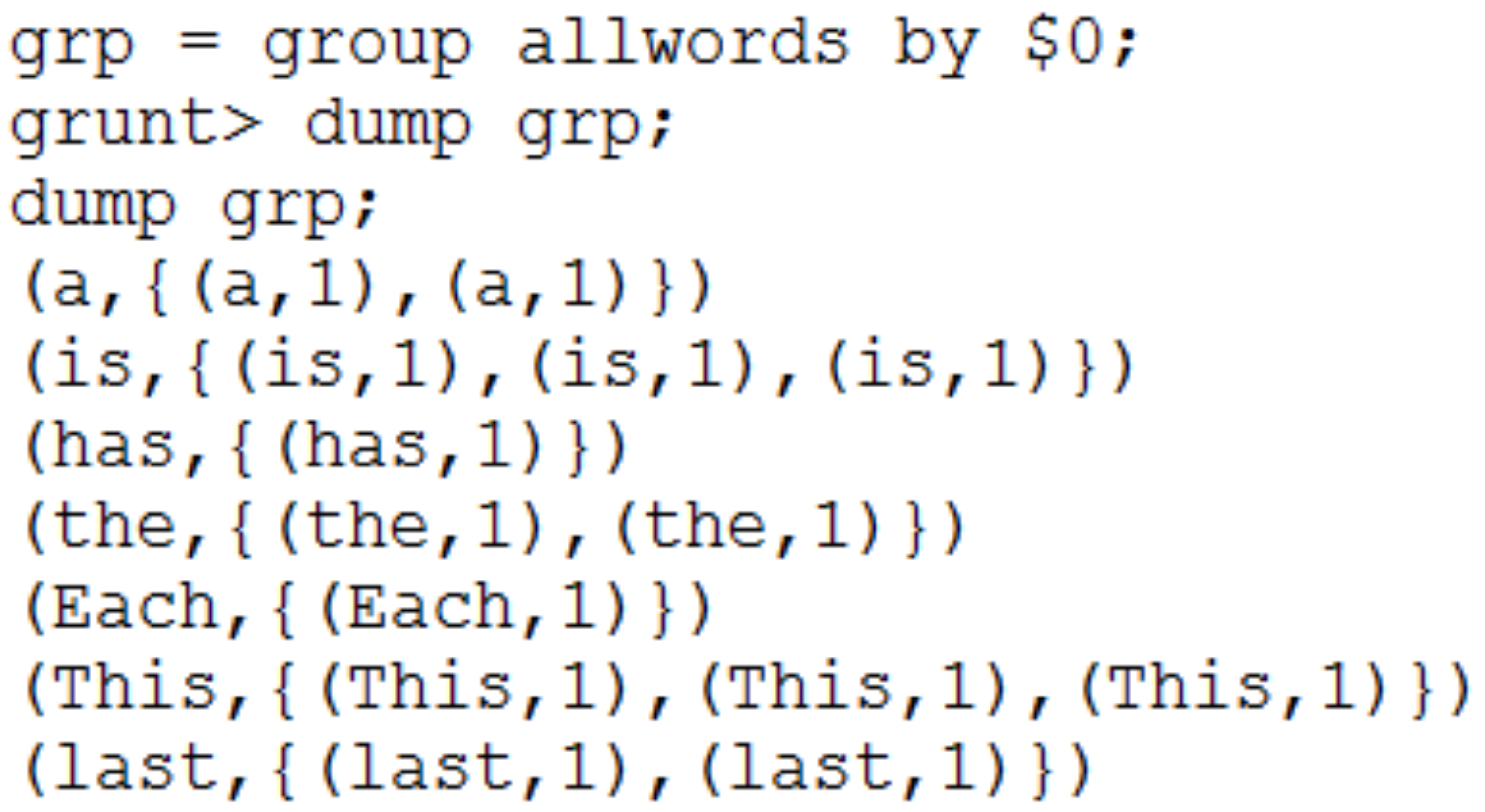
```
grunt> words = foreach docs generate TOKENIZE($0) as wordlist;
words = foreach docs generate TOKENIZE($0) as wordlist;
grunt> dump words;
dump words;
({(This),(is),(a),(sample),(document.)})
({(Each),(line),(has),(a),(separate),(sentence.)})
({(This),(is),(the),(second),(last),(line.)})
({(This),(is),(the),(last),(line.)})
```

Step 3:



```
grunt> allwords = FOREACH words GENERATE FLATTEN(wordlist), 1;
allwords = FOREACH words GENERATE FLATTEN(wordlist), 1;
grunt> dump allwords;
dump allwords;
(This,1)
(is,1)
(a,1)
(sample,1)
(document.,1)
(Each,1)
(line,1)
(has,1)
(a,1)
(separate,1)
(sentence.,1)
```

Step 4:

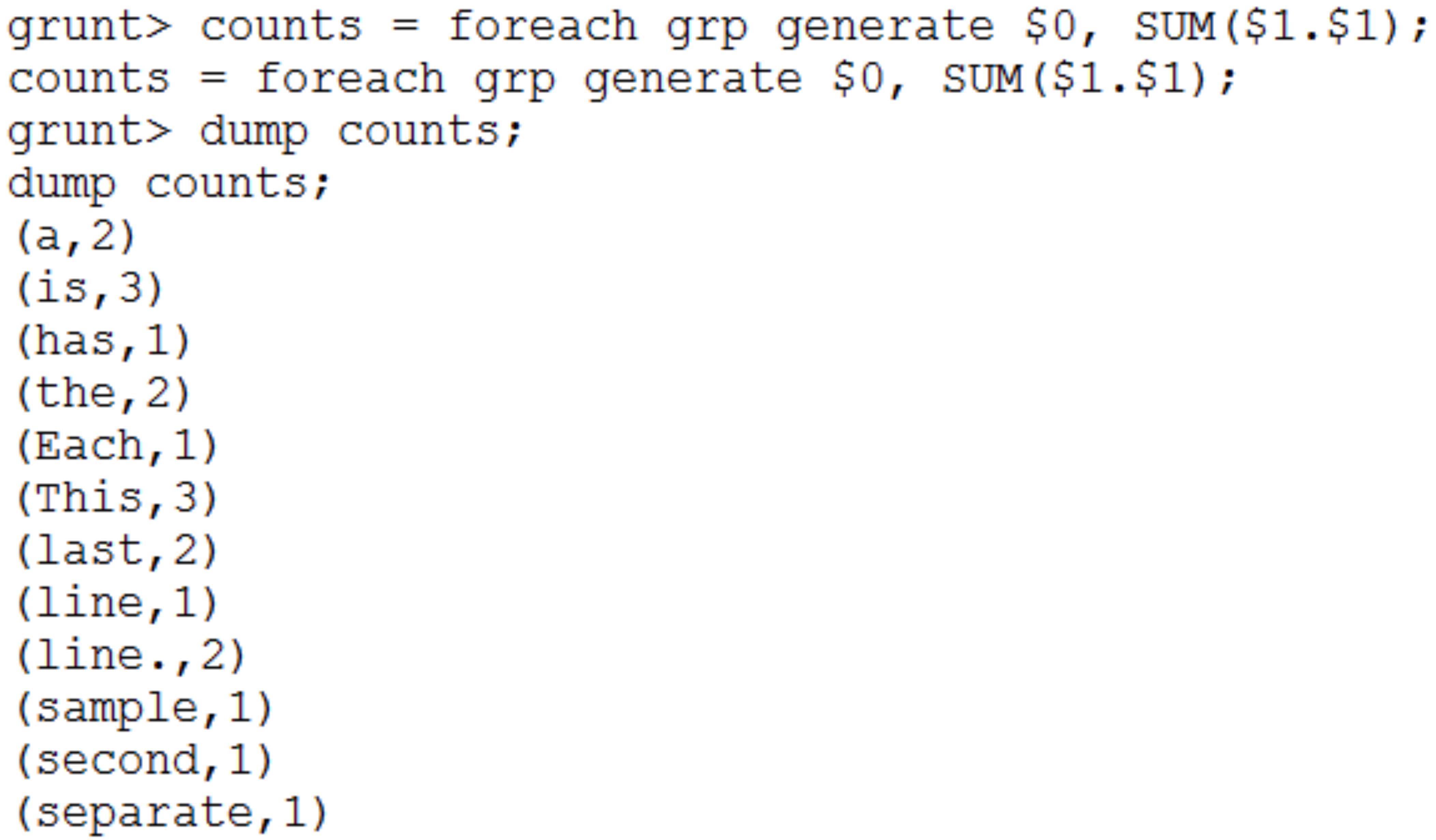


```
grp = group allwords by $0;
grunt> dump grp;
dump grp;
(a,{(a,1),(a,1)})
(is,{(is,1),(is,1),(is,1)})
(has,{(has,1)})
(the,{(the,1),(the,1)})
(Each,{(Each,1)})
(This,{(This,1),(This,1),(This,1)})
(last,{(last,1),(last,1)})
```

For the first row/record: (a, { (a,1), (a,1)})

- $0 = a
- $1 = (a,1)
- $1.$1 = 1 =>   sum($1.$1) = 2

Step 5:



```
grunt> counts = foreach grp generate $0, SUM($1.$1);
counts = foreach grp generate $0, SUM($1.$1);
grunt> dump counts;
dump counts;
(a,2)
(is,3)
(has,1)
(the,2)
(Each,1)
(This,3)
(last,2)
(line,1)
(line.,2)
(sample,1)
(second,1)
(separate,1)
```

Macros

- Macros provide a way to define reusable code (functions)
  - DEFINE <macroName> (<args>) RETURNS <returnValue> { theCode }
- Wordcount Example:

```
DEFINE wordcount(text) RETURNS counts {
   tokens = foreach $text generate TOKENIZE($0) as terms;
   wordlist = foreach tokens generate FLATTEN(terms) as word,1 as freq;
   groups = group wordlist by word;
   $counts = foreach groups generate group as word,SUM(wordlist.freq) as freq;
}
```