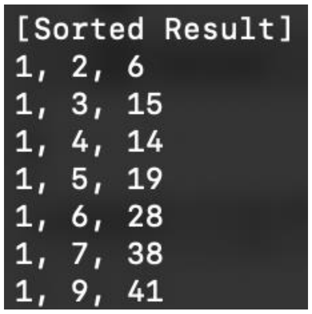
4.1A. Shortest Paths

In datalog:

```
database({
  warc(A: integer, B: integer, D: integer)
}).

pth(A,B,mmin<D>) <- warc(A,B,D), A = {startvertex}.
pth(A,B,mmin<D>) <- pth(A,C,Da), warc(C,B,Db), D = Da+Db.
qpth(A,B,min<D>) <- pth(A,B,D).
query qpth(A,B,D).
```

THE TEST SAMPLE:

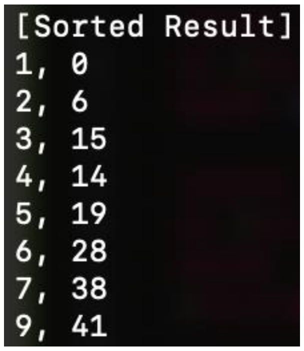


```
[Sorted Result]
1, 2, 6
1, 3, 15
1, 4, 14
1, 5, 19
1, 6, 28
1, 7, 38
1, 9, 41
```

In SQL:

```
create table warc(A integer, B integer, C integer);

WITH recursive pth (Node, min() AS Dist)
AS  (SELECT {startvertex}, 0)
       UNION
    (SELECT warc.B, pth.Dist+warc.C FROM pth, warc
     WHERE pth.Node = warc.A)
SELECT Node,Dist FROM pth
```

THE TEST SAMPLE:



```
[Sorted Result]
1, 0
2, 6
3, 15
4, 14
5, 19
6, 28
7, 38
9, 41
```

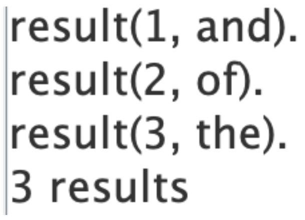
Comments:
The queries showcased above were applied to the dataset of 'wars.csv'.The usage of recursion is a monotonic aggregation, which enables us to aggregate the Datalog queries in this case. The same concept could also be applied to the recursion in our RaSQL.
The major difficulties that we encountered throughout the implementation include Datalog not allowing data duplicates, as well as understanding the syntax for RaSQL as it is slightly different from conventional databases. Besides the mentioned difficulties, we did not find ourselves having issues with other procedures in the process.

4.1D. Sorting in Lexicographical Order

In datalog:

```
name("the").
name("of").
name("and").

database({name(A: string)}).
export name(A).
order(J,A) <- name(A), J=1.
order(J1,A1) <- order(J,A),name(A1),A<A1,J1=(J+1).
result(J,min<A>) <- order(J,A).
export result(J,A).
```

THE TEST SAMPLE:

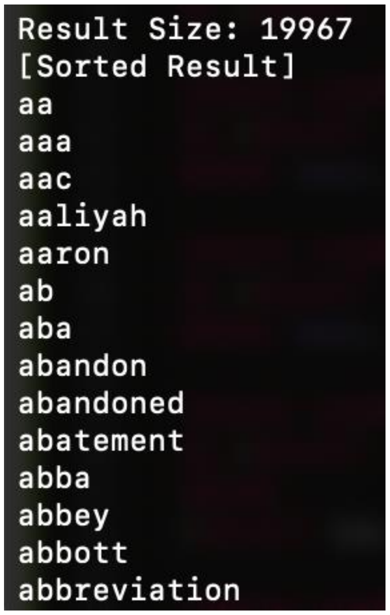


```
result(1, and).
result(2, of).
result(3, the).
3 results
```

In SQL:

```
create table name(A string);

SELECT A FROM name
ORDER BY A
```

THE TEST SAMPLE:


```
Result Size: 19967
[Sorted Result]
aa
aaa
aac
aaliyah
aaron
ab
aba
abandon
abandoned
abatement
abba
abbey
abbott
abbreviation
```

Comments:
For lexicographic sorting, we have used the dataset of 'names.csv', which appears to contain random terms. As presented above, we have implemented two solutions using Datalog and another two solutions using RaSQL. We have tested and validated all appraoches and they are all working properly. After the csv file containing the data is read, the words will be sorted either by selection sort or rank increment. Both are valid solutions which made use of recursion with monotonic aggregations. The final output contains a sorted list with indices in lexicographic order or the names in lexicographic order.

4.2 datalog and SQL on KNN method

4.2.1 introduction of KNN method

the k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression.[1] In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression:

In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.
In k-NN regression, the output is the property value for the object. This value is the average of the values of k nearest neighbors.

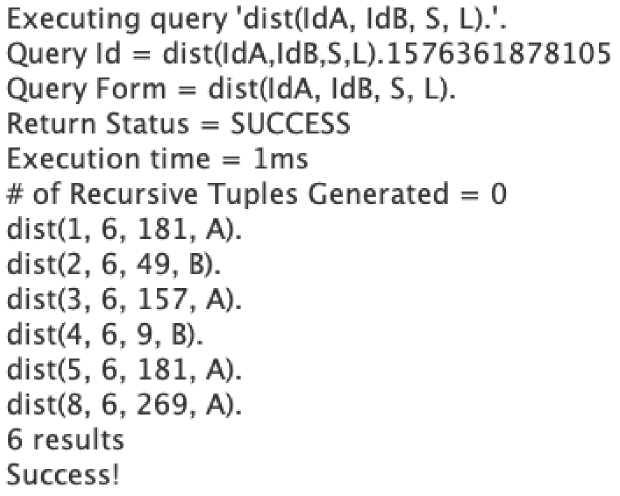
4.2.2 method of doing KNN – non recursive method

Non-Recursive: firstly, compute the Euclidean distance between the target data and every data in our training dataset; secondly, we order the training data using Euclidean distance; assume a certain, we pick the top k training data and assign to target data the majority classification.

In datalog:

```
% first step find all distance
dist(IdA, IdB, S, L) <- tr(IdA, X1a, X2a, L), te(IdB, X1b, X2b,_), S = (X1a - X1b)*(X1a - X1b) +
(X2a - X2b)*(X2a - X2b).
export dist(IdA, IdB, S, L).
```

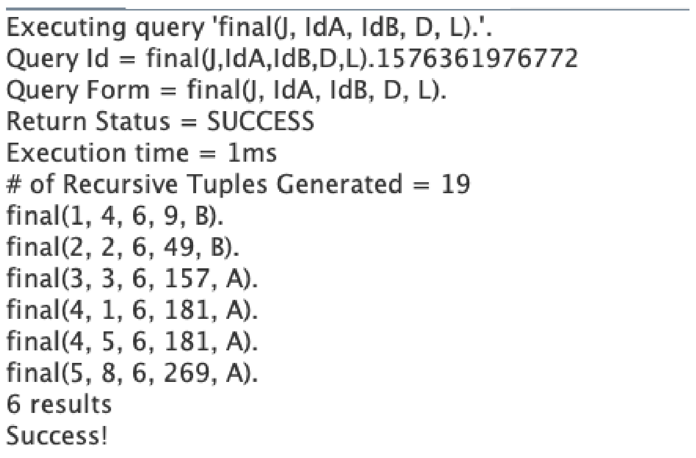
THE TEST SAMPLE:



```
Executing query 'dist(IdA, IdB, S, L).'.
Query Id = dist(IdA,IdB,S,L).1576361878105
Query Form = dist(IdA, IdB, S, L).
Return Status = SUCCESS
Execution time = 1ms
# of Recursive Tuples Generated = 0
dist(1, 6, 181, A).
dist(2, 6, 49, B).
dist(3, 6, 157, A).
dist(4, 6, 9, B).
dist(5, 6, 181, A).
dist(8, 6, 269, A).
6 results
Success!
```

```
% second step order the data using these Euclidean distance
order(J,IdA, IdB, S) <- dist(IdA, IdB, S, _), J=1.
order(J1,IdA1, IdB, S1) <- order(J,IdA, IdB, S),dist(IdA1, IdB, S1, _),S<S1,J1=(J+1).
result(J,IdB, min<S>) <- order(J,_, IdB, S).
final(J,IdA, IdB, D, L) <- result(J,IdB, D), dist(IdA, IdB, D, L).

export order(C, IdA, IdB, S).
export order(J,IdA, IdB, S).
export result(J, IdB, S).
export final(J,IdA, IdB, D, L).
```

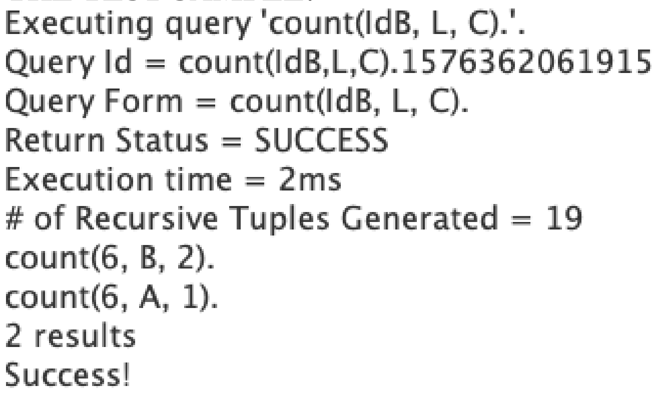
THE TEST SAMPLE:



```
Executing query 'final(J, IdA, IdB, D, L).'.
Query Id = final(J,IdA,IdB,D,L).1576361976772
Query Form = final(J, IdA, IdB, D, L).
Return Status = SUCCESS
Execution time = 1ms
# of Recursive Tuples Generated = 19
final(1, 4, 6, 9, B).
final(2, 2, 6, 49, B).
final(3, 3, 6, 157, A).
final(4, 1, 6, 181, A).
final(4, 5, 6, 181, A).
final(5, 8, 6, 269, A).
6 results
Success!
```

% third step given a certain k, find the top k training data and their labels and count the majority
count(IdB, L, count<IdA>) <- final(J,IdA, IdB, D, L), J <= 3.
export count(IdB, L, C).

THE TEST SAMPLE:



```
Executing query 'count(IdB, L, C).'.
Query Id = count(IdB,L,C).1576362061915
Query Form = count(IdB, L, C).
Return Status = SUCCESS
Execution time = 2ms
# of Recursive Tuples Generated = 19
count(6, B, 2).
count(6, A, 1).
2 results
Success!
```

In this sample test, we find given the new target ID=6, the label should be 'B', and we can also compare the true label your new target to the predicted label. In my test set, the new target ID=6 has the label 'B'.

In SQL:

In this part, I use the test dataset from (http://archive.ics.uci.edu/ml/machine-learning-databases/hill-valley/) and choose two features from them.

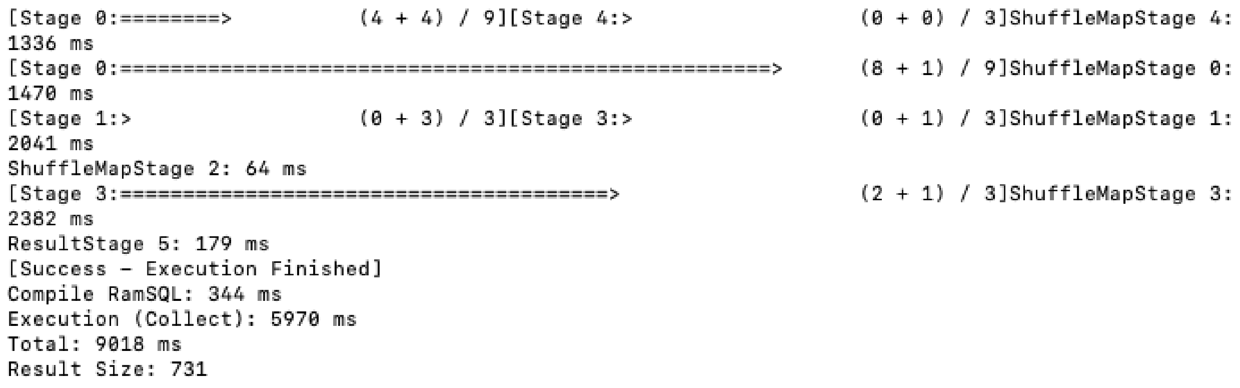
First step, we create two tables:

create table arc(ID integer, A float, B float, L string); # it is the training dataset
create table rc(ID integer, C float, D float, M string); # it is the testing dataset

Second step, we order the training data using Euclidean distance; assume a certain, we pick the top k training data and assign to target data the majority classification.

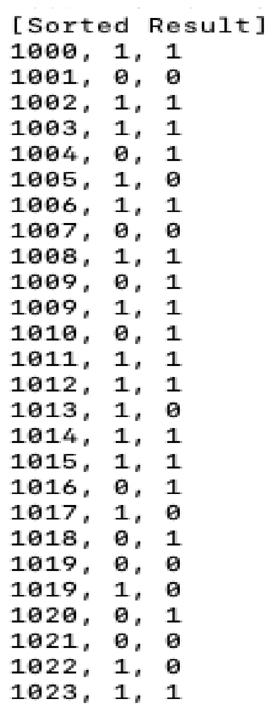
```
-- -- let us say k = 10 and find the label
CREATE VIEW knn(IDb, L, cou) AS
select IDb, L, count(IDa) as cou
from (
                select IDa, IDb, distance, L, rank() over(partition by IDb order by distance) as
rank
        from (
                select arc.ID AS IDa, rc.ID as IDb, (A-C)*(A-C) + (B-D)*(B-D) AS distance, L
                from arc, rc
        ) as merge
) as total
where rank <= 8
group by IDb, L;

SELECT ID, L, M
from rc,
(
        select k.IDb, k.L
        from knn k
        where k.cou >= ( select max(n.cou) from knn n where k.IDb = n.IDb)
) as final
where final.IDb = rc.ID;
```

THE TEST SAMPLE:



```
[Stage 0:========>          (4 + 4) / 9][Stage 4:>               (0 + 0) / 3]ShuffleMapStage 4:
1336 ms
[Stage 0:=====================================================>      (8 + 1) / 9]ShuffleMapStage 0:
1470 ms
[Stage 1:>                  (0 + 3) / 3][Stage 3:>               (0 + 1) / 3]ShuffleMapStage 1:
2041 ms
ShuffleMapStage 2: 64 ms
[Stage 3:========================================>               (2 + 1) / 3]ShuffleMapStage 3:
2382 ms
ResultStage 5: 179 ms
[Success - Execution Finished]
Compile RamSQL: 344 ms
Execution (Collect): 5970 ms
Total: 9018 ms
Result Size: 731
```

the results shown below (only a part):



```
[Sorted Result]
1000, 1, 1
1001, 0, 0
1002, 1, 1
1003, 1, 1
1004, 0, 1
1005, 1, 0
1006, 1, 1
1007, 0, 0
1008, 1, 1
1009, 0, 1
1009, 1, 1
1010, 0, 1
1011, 1, 1
1012, 1, 1
1013, 1, 0
1014, 1, 1
1015, 1, 1
1016, 0, 1
1017, 1, 0
1018, 0, 1
1019, 0, 0
1019, 1, 0
1020, 0, 1
1021, 0, 0
1022, 1, 0
1023, 1, 1
```

The accuracy is 0.523.

Next, let us choose different k (for k from 11 to 50) to test if there is any difference about the accuracy:

```
11 0.515702479338843
12 0.512396694214876
13 0.515702479338843
14 0.512396694214876
15 0.5256198347107438
16 0.5355371900826447
17 0.5289256198347108
18 0.5239669421487604
19 0.5305785123966942
20 0.5223140495867769
21 0.5256198347107438
22 0.5272727272727272
23 0.5355371900826447
24 0.5272727272727272
25 0.5239669421487604
26 0.5223140495867769
27 0.5173553719008265
28 0.5322314049586777
29 0.5256198347107438
30 0.515702479338843
31 0.5074380165289256
```

```
32 0.5041322314049587
33 0.5074380165289256
34 0.512396694214876
35 0.5140495867768595
36 0.5074380165289256
37 0.5107438016528926
38 0.5140495867768595
39 0.509090909090909
40 0.47107438016528924
41 0.47107438016528924
42 0.4727272727272727
43 0.4677685950413223
44 0.47768595041322315
45 0.4909090909090909
46 0.48760330578512395
47 0.5024793388429752
48 0.4925619834710744
49 0.5140495867768595
50 0.5140867469468595
```

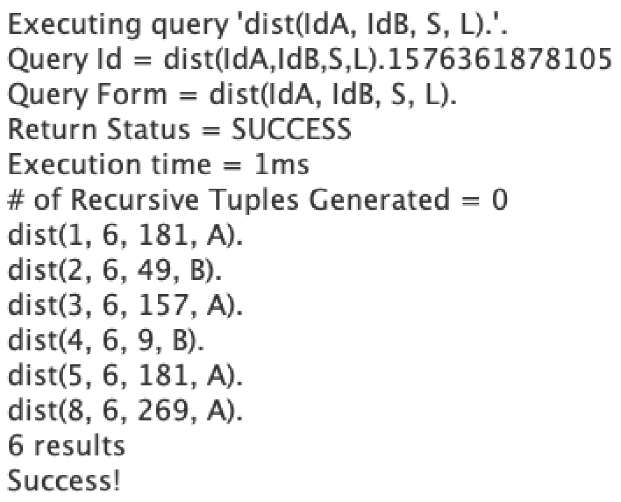
4.2.3 method of doing KNN – recursive method

Recursive method: Starting from the shortest distance found in 1 increase it by 20 or so until there are k or more points can be conted (using monotonic count) in that neighborhood. Finally classify according to the majority.

In datalog:

```
% first step find all distance
dist(IdA, IdB, S, L) <- tr(IdA, X1a, X2a, L), te(IdB, X1b, X2b,_), S = (X1a - X1b)*(X1a - X1b) +
(X2a - X2b)*(X2a - X2b).
export dist(IdA, IdB, S, L).
```

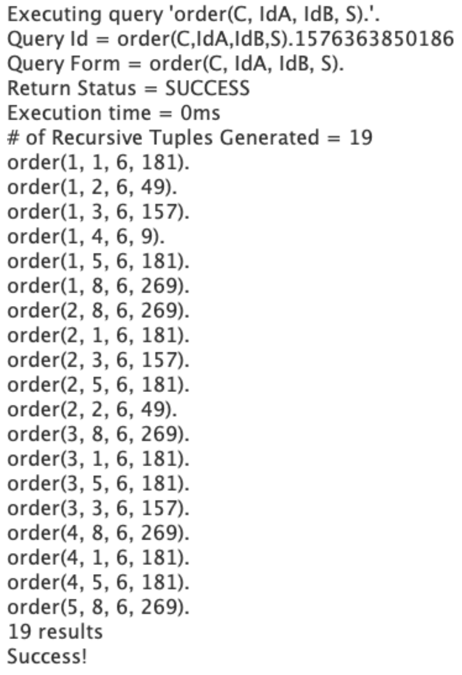
THE TEST SAMPLE:



```
Executing query 'dist(IdA, IdB, S, L).'.
Query Id = dist(IdA,IdB,S,L).1576361878105
Query Form = dist(IdA, IdB, S, L).
Return Status = SUCCESS
Execution time = 1ms
# of Recursive Tuples Generated = 0
dist(1, 6, 181, A).
dist(2, 6, 49, B).
dist(3, 6, 157, A).
dist(4, 6, 9, B).
dist(5, 6, 181, A).
dist(8, 6, 269, A).
6 results
Success!
```

% Second step use the recusive
order(IdA, IdB, S) <- dist(IdA, IdB, S, _), ~smaller(IdA, IdB, S).
smaller(IdA, IdB, S) <- dist(_, IdB, S2, _), dist(IdA, IdB, S, _), S2<S.
next(IdA1,IdB,S1,IdA2,S2) <- dist(IdA1, IdB, S1, _), dist(IdA1, IdB, S2, _),
~between(IdA1,IdB,S1,IdA2,S2).
between(IdA1,IdB,S1,IdA3,S3) <- dist(IdA1, IdB, S1, _), dist(IdA2, IdB, S2, _), dist(IdA3, IdB, S3, _), D1<D2, D2<D3.
order(IdA2,IdB,S2) <- order(IdA1,IdB,S1), next(IdA1,IdB,S1,IdA1,S2), S2 <= S1+20.
export order(IdA2,IdB,S2).

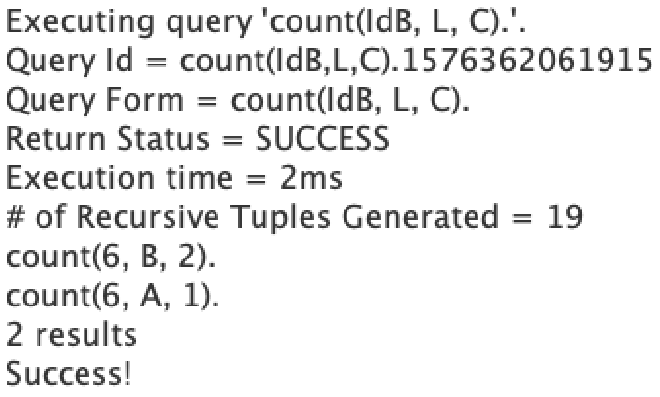
THE TEST SAMPLE:



```
Executing query 'order(C, IdA, IdB, S).'.
Query Id = order(C,IdA,IdB,S).1576363850186
Query Form = order(C, IdA, IdB, S).
Return Status = SUCCESS
Execution time = 0ms
# of Recursive Tuples Generated = 19
order(1, 1, 6, 181).
order(1, 2, 6, 49).
order(1, 3, 6, 157).
order(1, 4, 6, 9).
order(1, 5, 6, 181).
order(1, 8, 6, 269).
order(2, 8, 6, 269).
order(2, 1, 6, 181).
order(2, 3, 6, 157).
order(2, 5, 6, 181).
order(2, 2, 6, 49).
order(3, 8, 6, 269).
order(3, 1, 6, 181).
order(3, 5, 6, 181).
order(3, 3, 6, 157).
order(4, 8, 6, 269).
order(4, 1, 6, 181).
order(4, 5, 6, 181).
order(5, 8, 6, 269).
19 results
Success!
```

% the third step

```
increase(IdA2, IdB, S2,0) <- order(IdA1, IdB, S1), next(IdA1,IdB,S1,IdA2,S2), S2 <= S1+20.
increase(IdA2,IdB,S2,N2) <- increase(IdA3, IdB, S3,N3), next(IdA1,IdB,S1,IdA2,S2), S2 <= S3
+ N3, N2=N3+20, N3<120.
final(IdA,IdB, S) <- increase(IdA,IdB,S,_).
export final(IdA,IdB,S).

% given a certain k, find the top k training data and their labels and count the majority
count(IdB, L, count<IdA>) <- final(IdA,IdB,_),tr(IdA, _, _, L) .
export count(IdB, L, C).
```



```
Executing query 'count(IdB, L, C).'.
Query Id = count(IdB,L,C).1576362061915
Query Form = count(IdB, L, C).
Return Status = SUCCESS
Execution time = 2ms
# of Recursive Tuples Generated = 19
count(6, B, 2).
count(6, A, 1).
2 results
Success!
```

In SQL:

In this part, I use the test dataset from (http://archive.ics.uci.edu/ml/machine-learning-databases/hill-valley/) and choose two features from them.

First step, we create two tables:

```
create table arc(ID integer, A float, B float, L string);  # it is the training dataset
create table rc(ID integer, C float, D float, M string); # it is the testing dataset
```

Second step, we use recursive way to classify according to the majority.

```
WITH RECURSIVE next(A,B,d)
AS (
        select A,B,d from d, arc
        where B=Id and (B,d) in
        (select B, min(d) from d group by B))
UNION ALL
(select A,B,d from d, arc where B=Id and (B,d) in
        (select B, min(d) from d
                where (A,B,d) NOT in(
                        select A,B,d from next)
                group by B));
```

The accuracy is 0. 5325.

Comments:
In this dataset, we can see in difference methods(recursive and non-recursive) share the same accuracy and performance; in general, non-recursive method takes less time to run and it is easy to understand. Also, in the recursive solution, when adding 20 to the distance, might add too many tuples into the solution because there can be more than the k number of neighbors in that one addition.
In the recursive method, this is a monotonic aggregate.
Issue:
If the dataset size is very large, it is very hard to import all the data into Datalog, also, we cannot keep duplicates in Datalog. When we just import the same data in *fac*, Datalog only keep one data line.