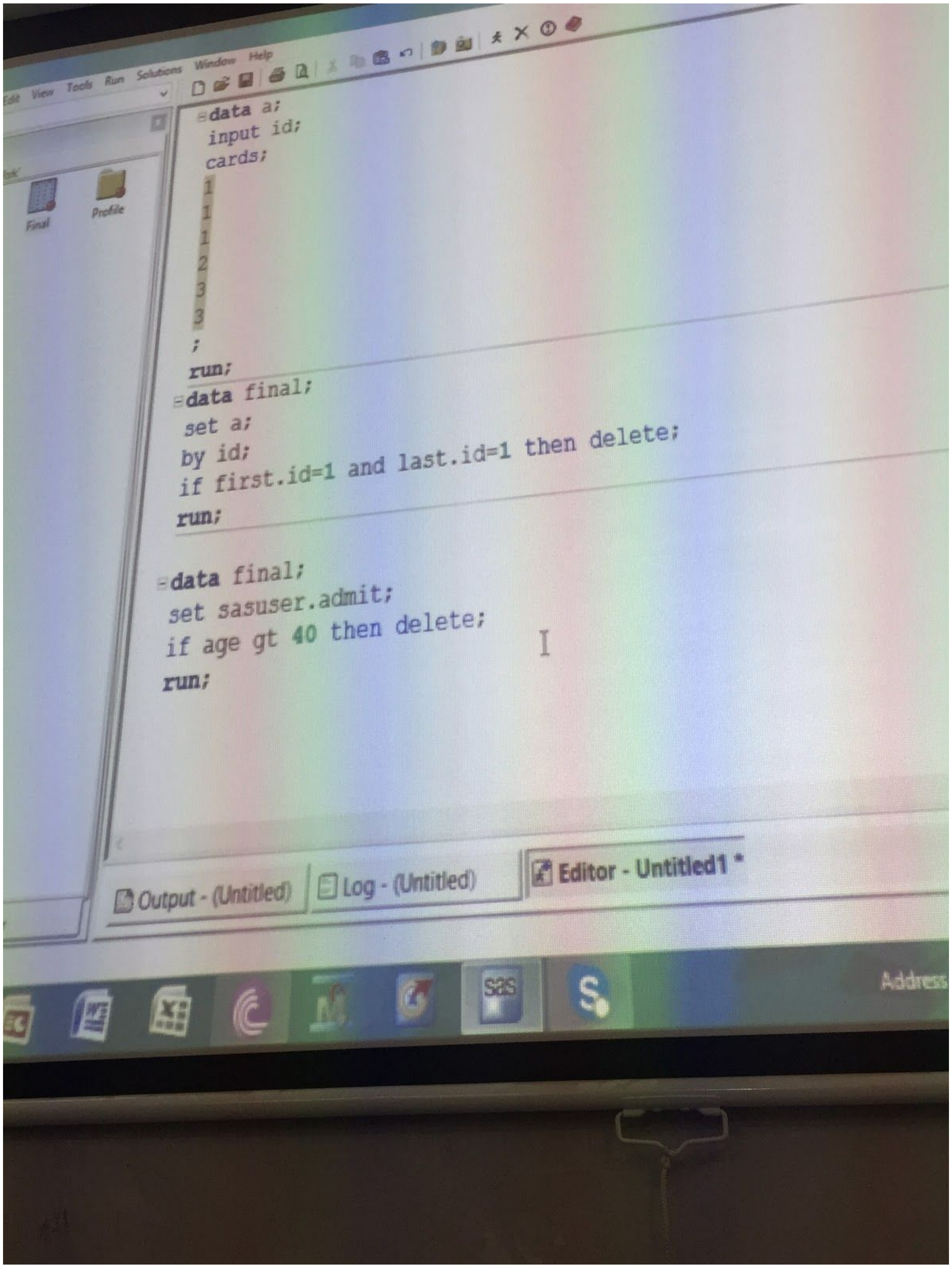
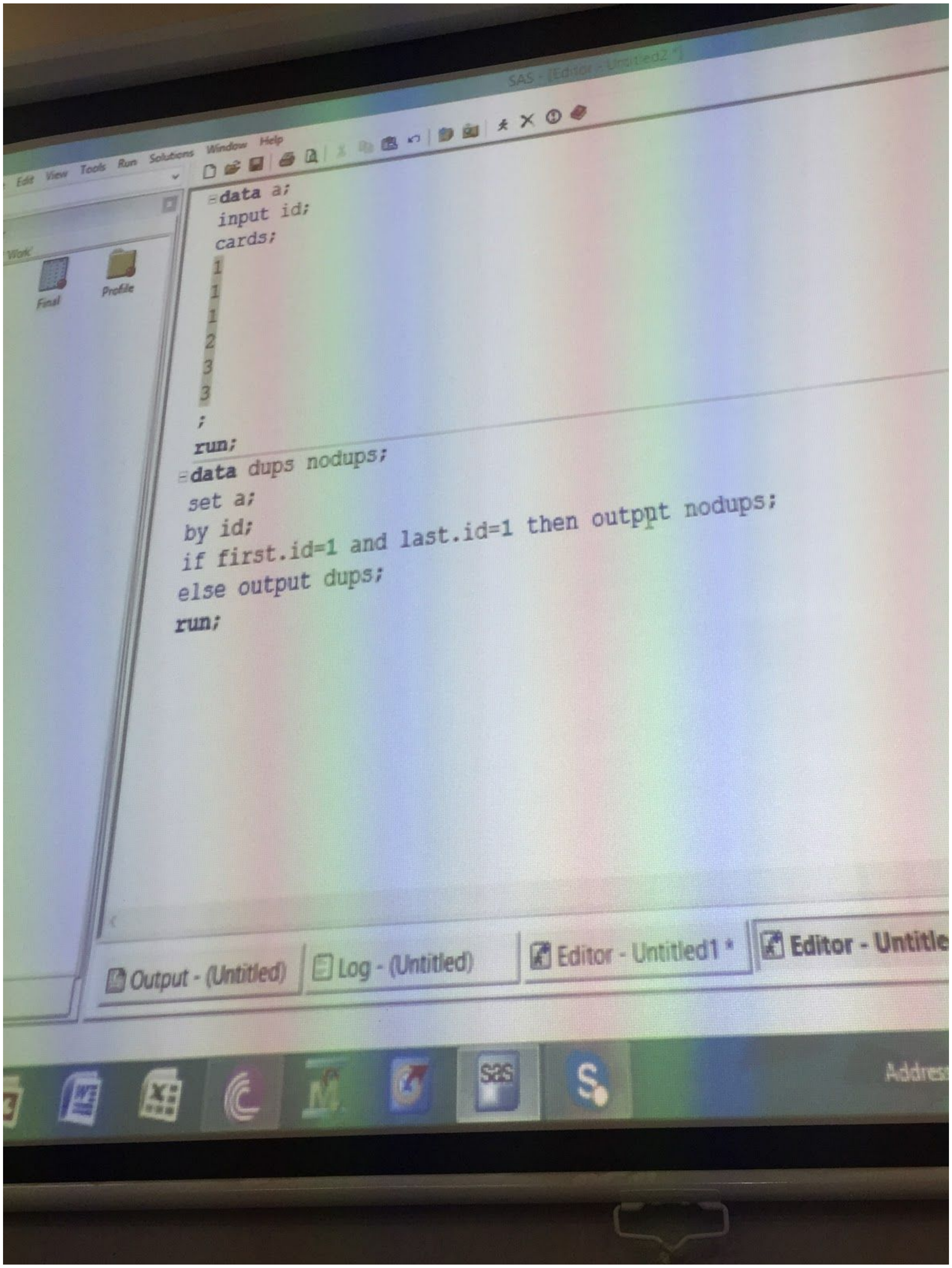
CLASS 9 NOTES



```
data a;
input id;
cards;
1
1
1
2
3
3
;
run;
data final;
set a;
by id;
if first.id=1 and last.id=1 then delete;
run;

data final;
set sasuser.admit;
if age gt 40 then delete;
run;
```



```
data a;
input id;
cards;
1
1
1
2
3
3
;
run;
data dups nodups;
set a;
by id;
if first.id=1 and last.id=1 then output nodups;
else output dups;
run;
```

CLASS-9

Program :

```
data a;
input id marks;
cards;
1    20
1    21
1    22
2    25
2    26
2    27
3    30
;
run;
```

		first.id	last.id	first.marks	last.marks
1	20	1	0	1	1
1	21	0	0	1	1
1	22	0	1	1	1
2	25	1	0	1	1
2	26	0	0	1	1
2	27	0	1	1	1
3	30	1	1	1	1

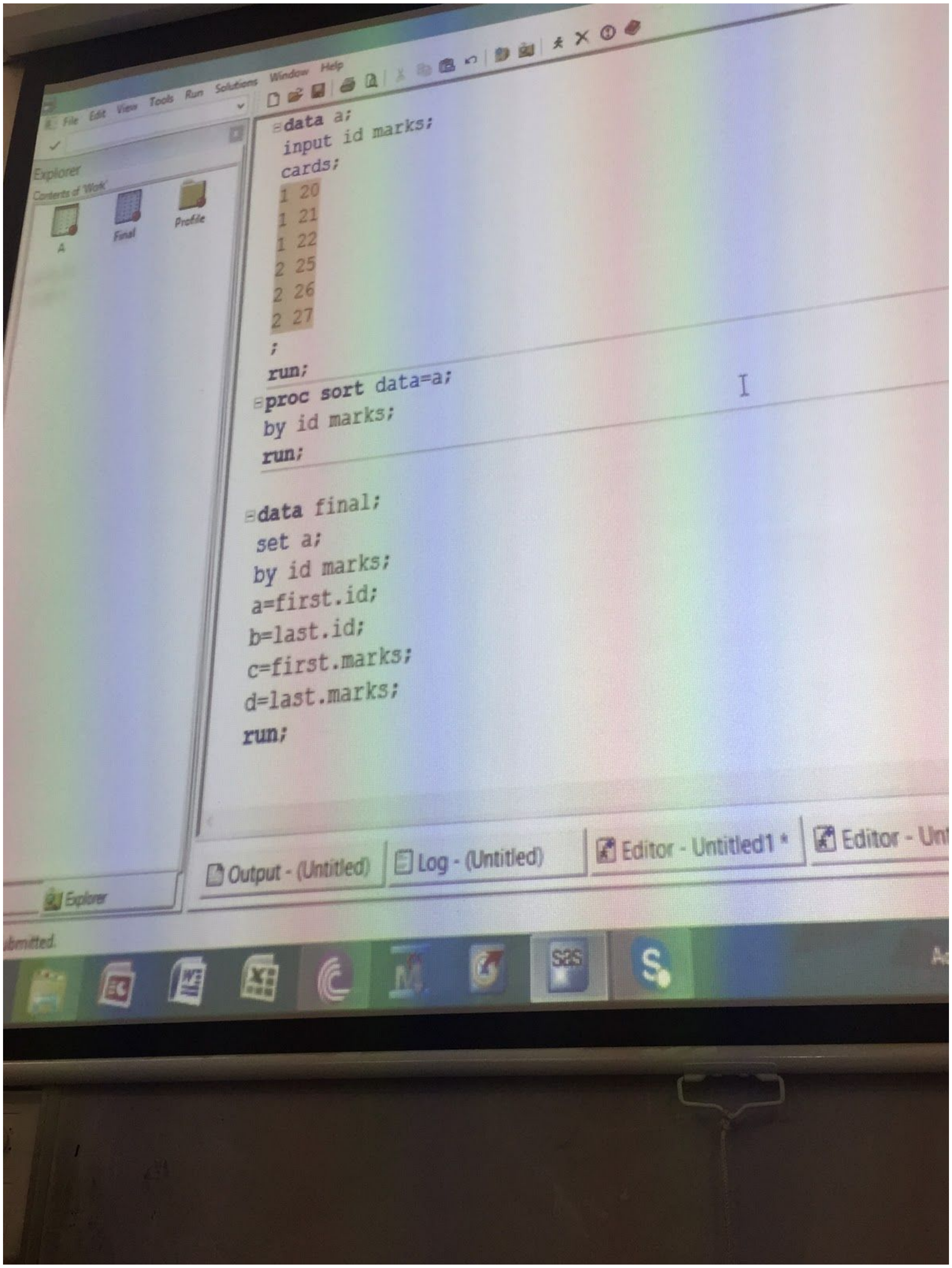
Suppose if we want the non-duplicates values to be deleted and duplicate values to be printed or vice-versa. then,

```
data final;
set a;
by id;          → If we want duplicates.
if first.id =1 and last.id =1 then delete;
run;
```

Another example

```
data a;
input id;
cards;
 1
 1
 1
 2
 3
 3
 ;
run;
```

```
data dups nodups;
set a;
by id;
if first.id=1 and last.id=1 then output
                                   nodups;
else output dups;
run;
```



```
data a;
 input id marks;
 cards;
1 20
1 21
1 22
2 25
2 26
2 27
;
run;
proc sort data=a;
 by id marks;
 run;

data final;
 set a;
 by id marks;
 a=first.id;
 b=last.id;
 c=first.marks;
 d=last.marks;
run;
```

```
data a;
input id marks;
cards;
1    20
1    21
1    22
2    25
2    26
2    27
;
run
```

```
proc sort data= a;
by id marks;
run;
```

```
data final;
set a;
by id marks;
a= first.id;
b= last.id;
c= first.marks;
d= last.marks;
run;
```

first.id	last.id
1	0
0	1
0	1
1	0
0	0
0	1

first.marks	last.marks
1	1
1	1
1	1
1	1
1	1
1	1

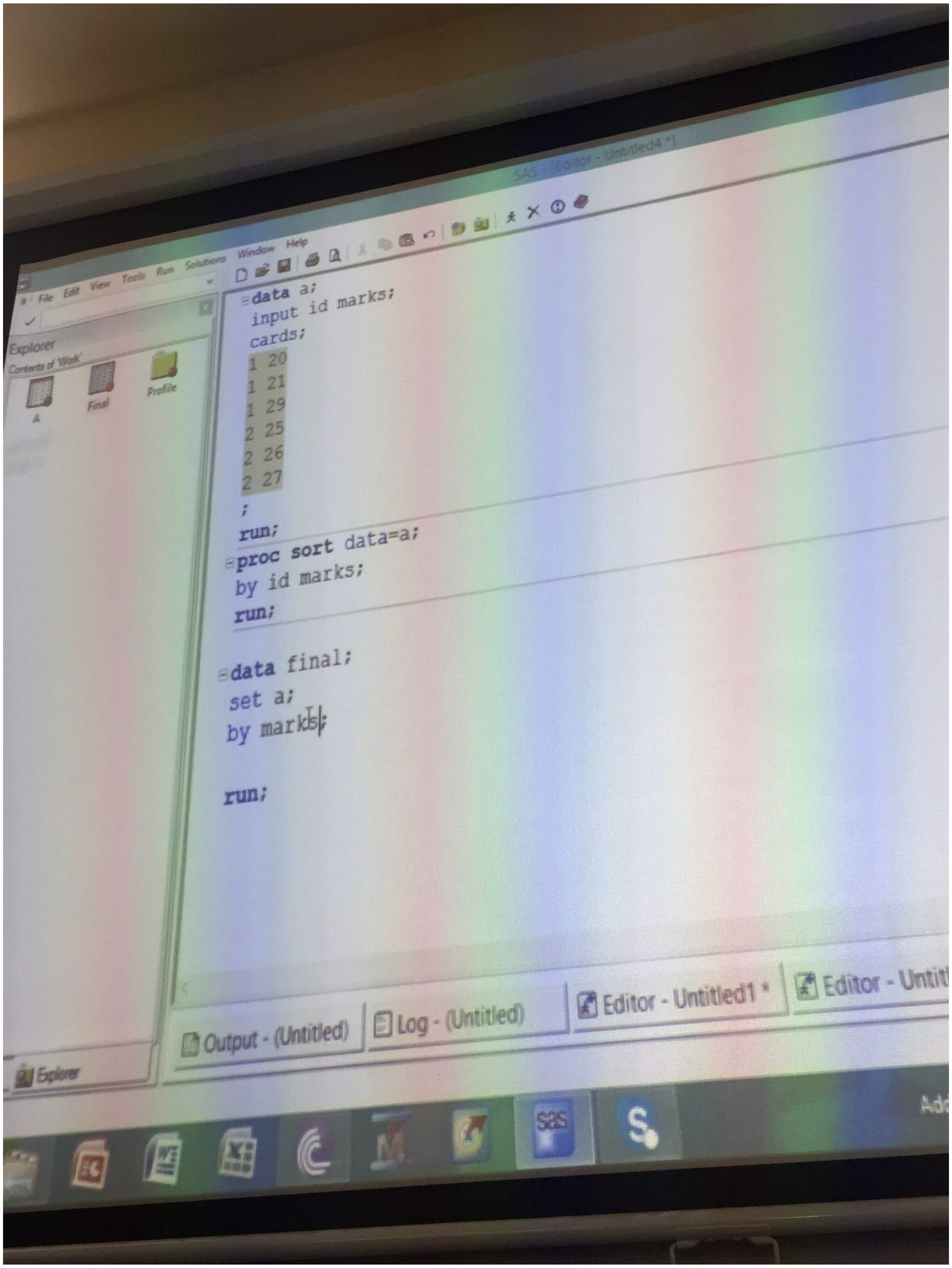
→ whatever variable we put in by, their first dot and last dot are made. Here we have two groups - primary(id) and secondary(marks) 'Id' will work according to the primary group and <u>marks</u> being the secondary group will work according to <u>id</u>.

<u>Program</u>

```
data a;
input id marks;
cards;
1    20
1    21
1    29
2    25
2    26
2    27
;
run;
```

```
proc sort data=a;
by id marks;        → This will run.
run;
```

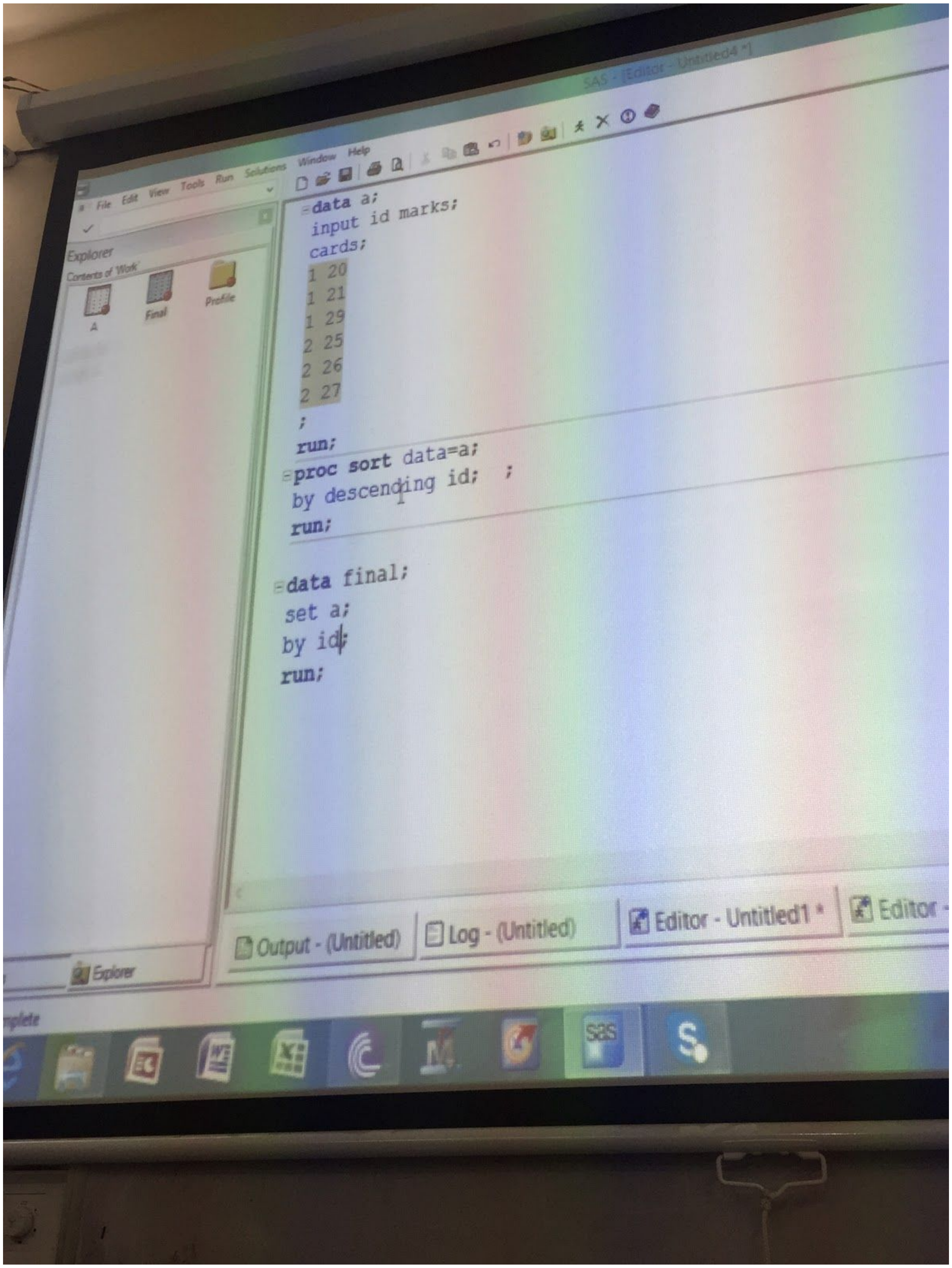
```
data final;
set a;              → This will show error because
by marks;             the dataset is not sorted by
run;                  marks, it is sorted by id.
```



```
data a;
 input id marks;
 cards;
1 20
1 21
1 29
2 25
2 26
2 27
;
run;
proc sort data=a;
by id marks;
run;

data final;
set a;
by marks;

run;
```



```
data a;
input id marks;
cards;
1 20
1 21
1 29
2 25
2 26
2 27
;
run;
proc sort data=a;
by descending id;  ;
run;

data final;
set a;
by id;
run;
```

Program

```
data a;
input id marks;
cards;
1  20
1  21
1  29
2  25
2  26
2  27
;
run;
```

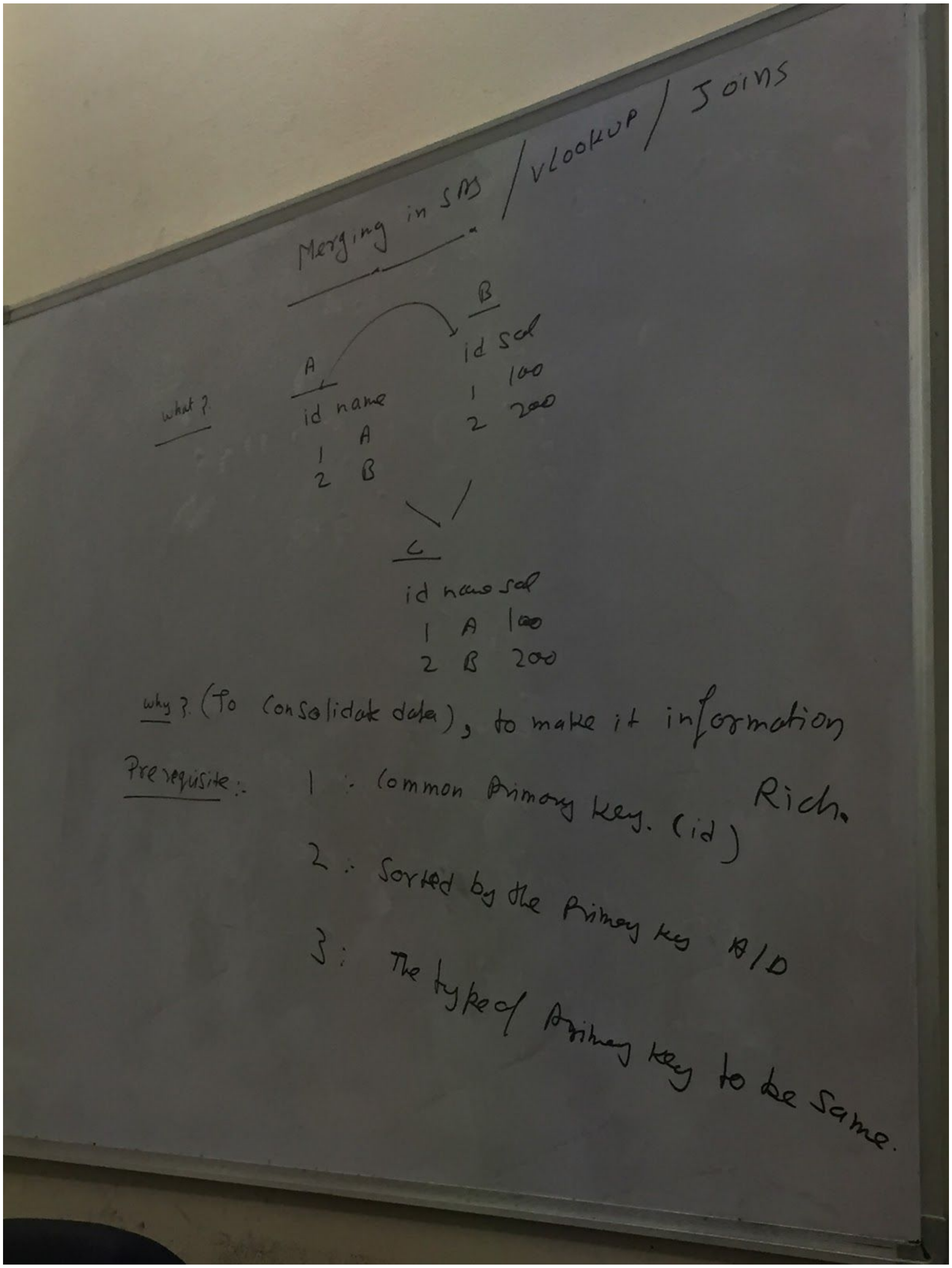
```
proc sort data=a;
by descending id;
run;
```

Here 'id' is sorted in descending order.

```
data final;
set a;
by id;
run;
```

This has to match

Here in ascending order. so, will show error.



Merging in SAS / VLOOKUP / Joins

what ?

A

id	name
1	A
2	B

B

id	sal
1	100
2	200

C

id	name	sal
1	A	100
2	B	200

why ? (To consolidate data), to make it information Rich.

Prerequisite :-

1 : Common Primary key. (id)

2 : Sorted by the Primary key A/D

3 : The type of Primary key to be Same.

MERGING in SAS / vlookup (Excel) / Join (Sql)

What is merging :

Merging is combining two or more datasets "head on" to make a child data set that has attributes (variables) from all the parent data set.

A

id	name
1	A
2	B

B

id	sal
1	100
2	200

Merge

C

id	name	sal
1	A	100
2	B	200

* In general, Appending increase rows and merging increases columns.

Here, C is the merged product of A and B or C is the child of A and B.

Why do we merge?

In order to consolidate data or to make it information rich, we do merging.

What are the pre-requisite (requirements) of merging:

1. Common key (Eg: 'id')
2. Sorted by the primary key (either ascending or descending).
3. The type of primary key should be same, (either character or numeric) eg- if it numeric on one side then it should be numeric on other side also.

How to do merging:

```
data a;
input id name $;
cards;
1   A
2   B
;
run;
```

```
data b;
input id sal;
cards;
1   100
2   200
;
run;
```

Merge

```
data c;          → child of 'a' and 'b'.
merge a b;       → Merge statement
by id;           → sorted by 'id'
run;
```

Output

id	name	sal
1	A	100
2	B	200

Types of merging

① 1 to 1 merging

Eg:

id	name
1	A
2	B

id	sal
1	100
2	200

↙
two bygroups

id	name	sal
1	A	100
2	B	200

Here, bygroups are two and each by group has single row in it. So, this type of merging is called <u>1-1 merging</u>.

<u>Note</u>: In 1-1 merging, no. of <u>rows</u> in the bygroup should be <u>one</u>.

<u>Another eg</u>:

①

id	name
1	A
2	B
3	C

id	sal
1	100
2	200

id	name	sal
1	A	100
2	B	200
3	C	.

$\longrightarrow$ numeric field

②

id	name
1	A
2	B

id	sal
1	100
2	200
7	300

$\longrightarrow$

id	name	sal
1	A	100
2	B	200
7	□	300

Character field

② 1- many merging

```
Data c;
merge a b;
by id;
run;
```

A

id	name
1	A
2	B
3	C

one to many

B

id	sub
1	H
1	M
2	E
3	M
3	S

Output

C

id	name	sub
1	A	H
1	A	M
2	B	E
3	C	M
3	C	S

3 bygroups

No. of bygroups are three, but in each bygroup no. of rows are one ~~are~~ more than one, so one to many merging.

③ Many to 1 merging

```
Data c;
merge b a;
by id;
run;
```

B

id	sub
1	H
1	M
2	E
3	M
3	S

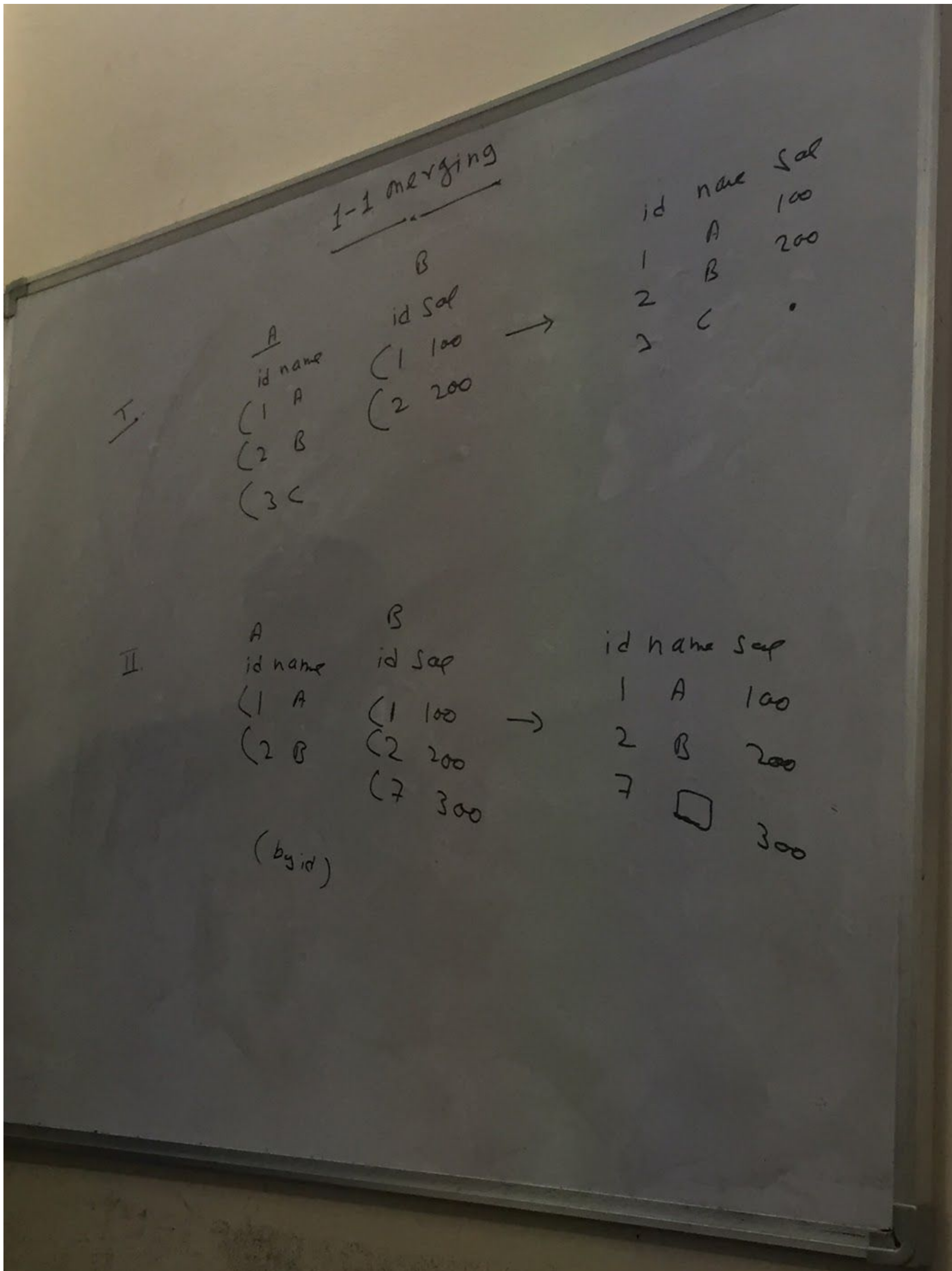
many to 1

A

id	name
1	A
	B
2	B
3	C

C

id	sub	name
1	H	A
1	M	A
2	E	B
3	M	C
3	S	C



1-1 merging

I.

A

id	name
1	A
2	B
3	C

B

id	sal
1	100
2	200

$\longrightarrow$

id	name	sal
1	A	100
2	B	200
3	C	•

II.

A

id	name
1	A
2	B

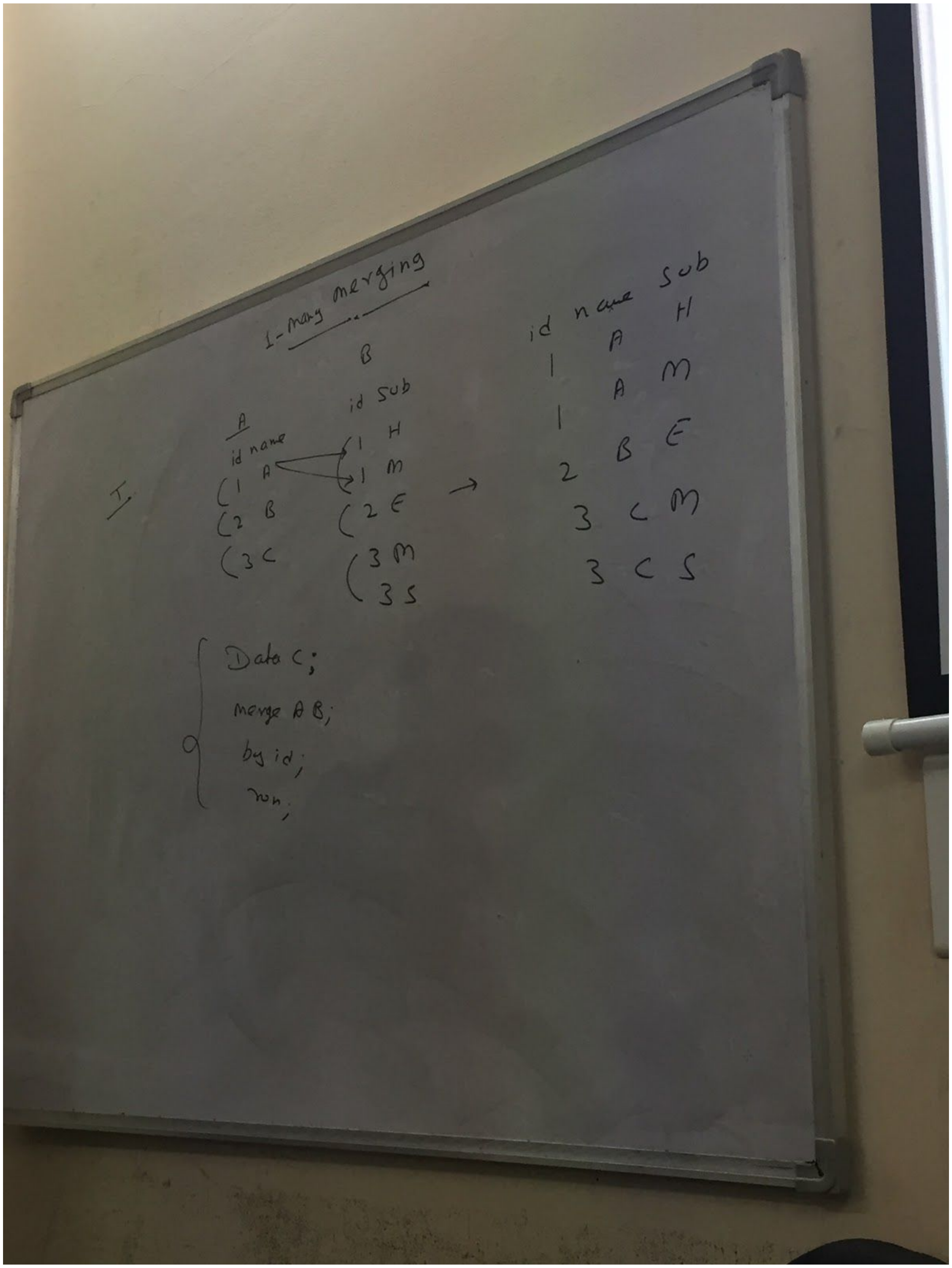
B

id	sal
1	100
2	200
7	300

(by id)

$\longrightarrow$

id	name	sal
1	A	100
2	B	200
7	□	300



1-many merging

I.

A

id	name
1	A
2	B
3	C

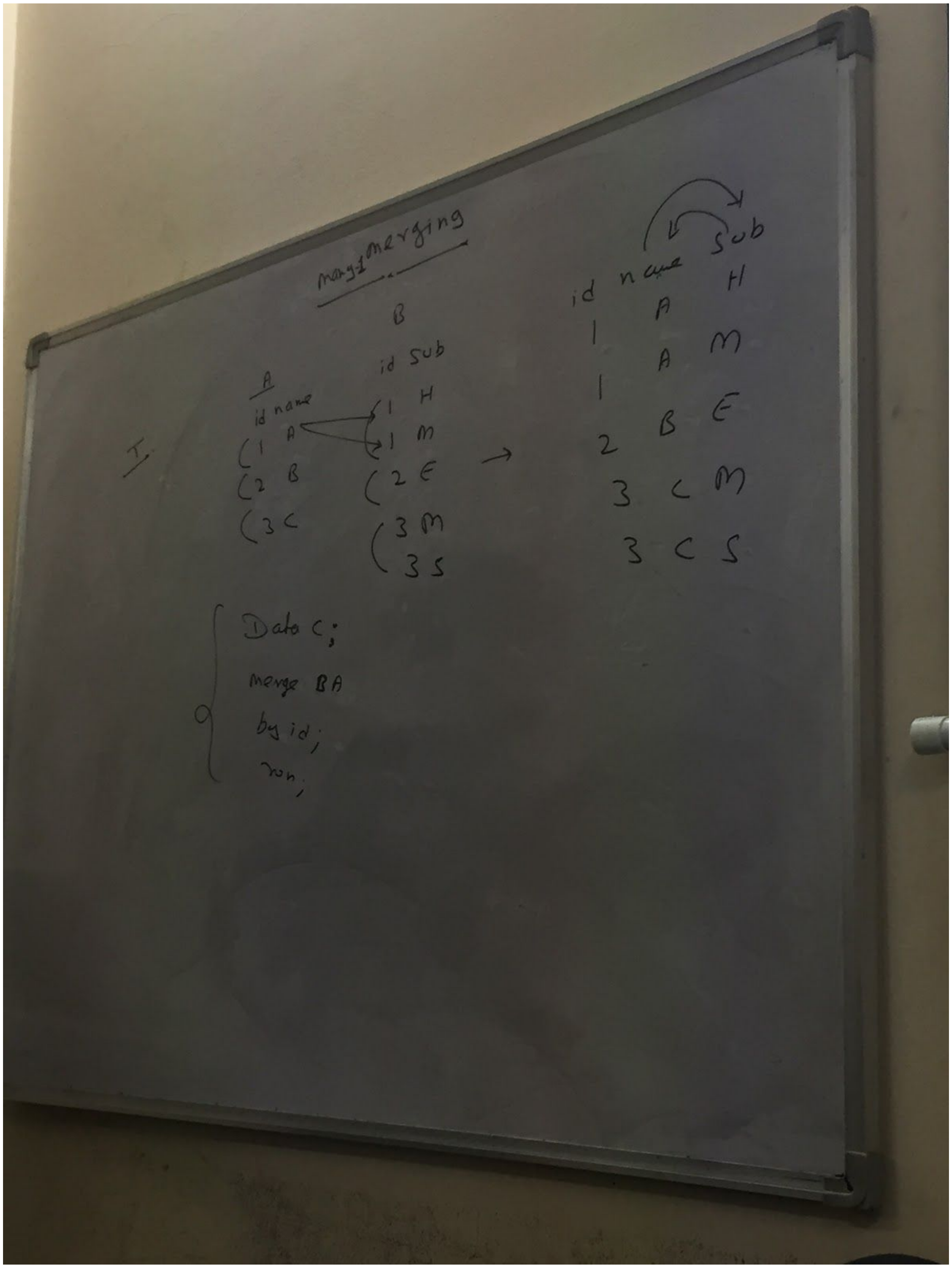
B

id	sub
1	H
1	M
2	E
3	M
3	S

$\rightarrow$

id	name	sub
1	A	H
1	A	M
2	B	E
3	C	M
3	C	S

```
Data C;
merge A B;
by id;
run;
```



many1 merging

I/

A

id	name
1	A
2	B
3	C

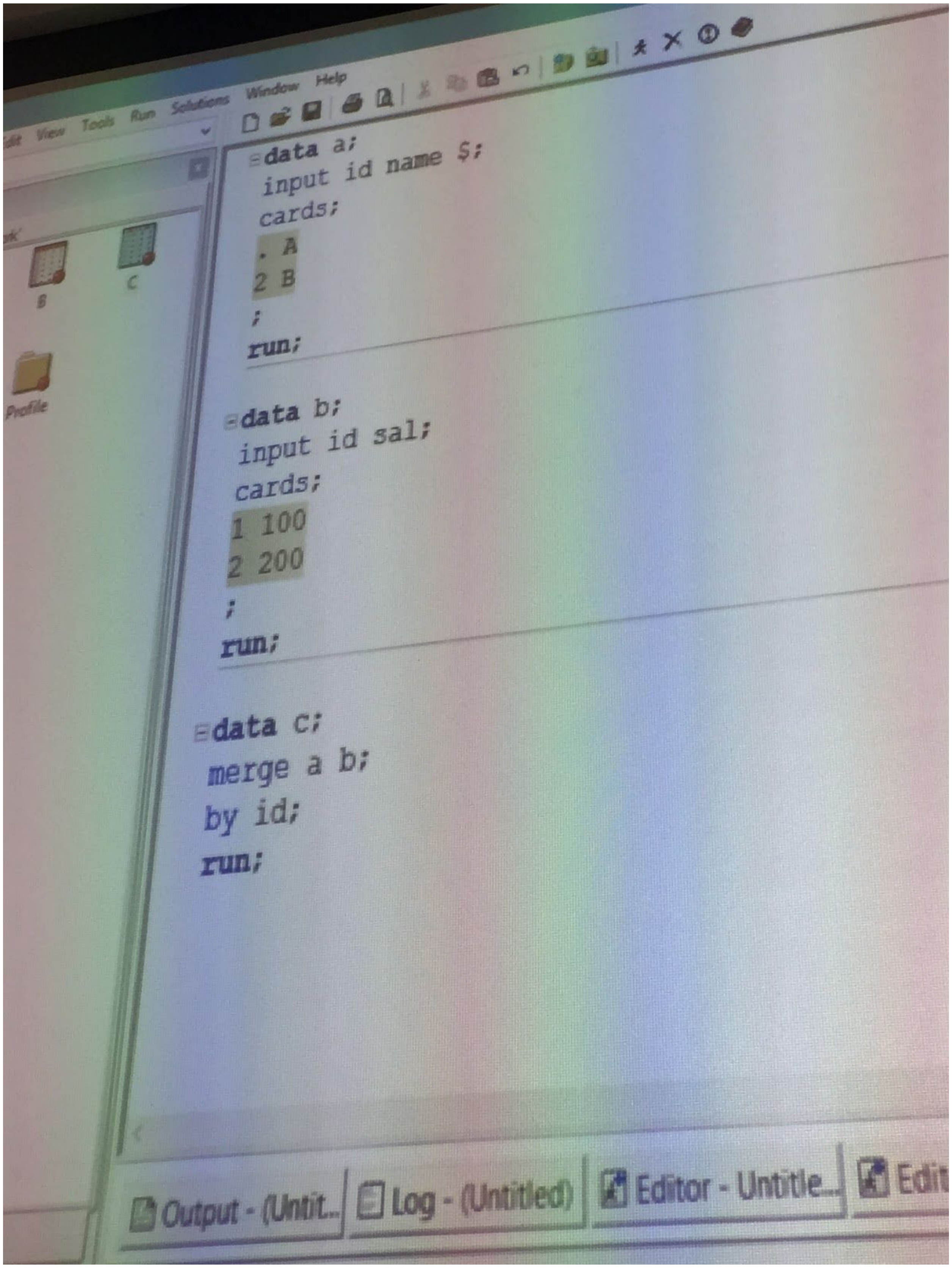
B

id	Sub
1	H
1	M
2	E
3	M
3	S

→

id	name	Sub
1	A	H
1	A	M
2	B	E
3	C	M
3	C	S

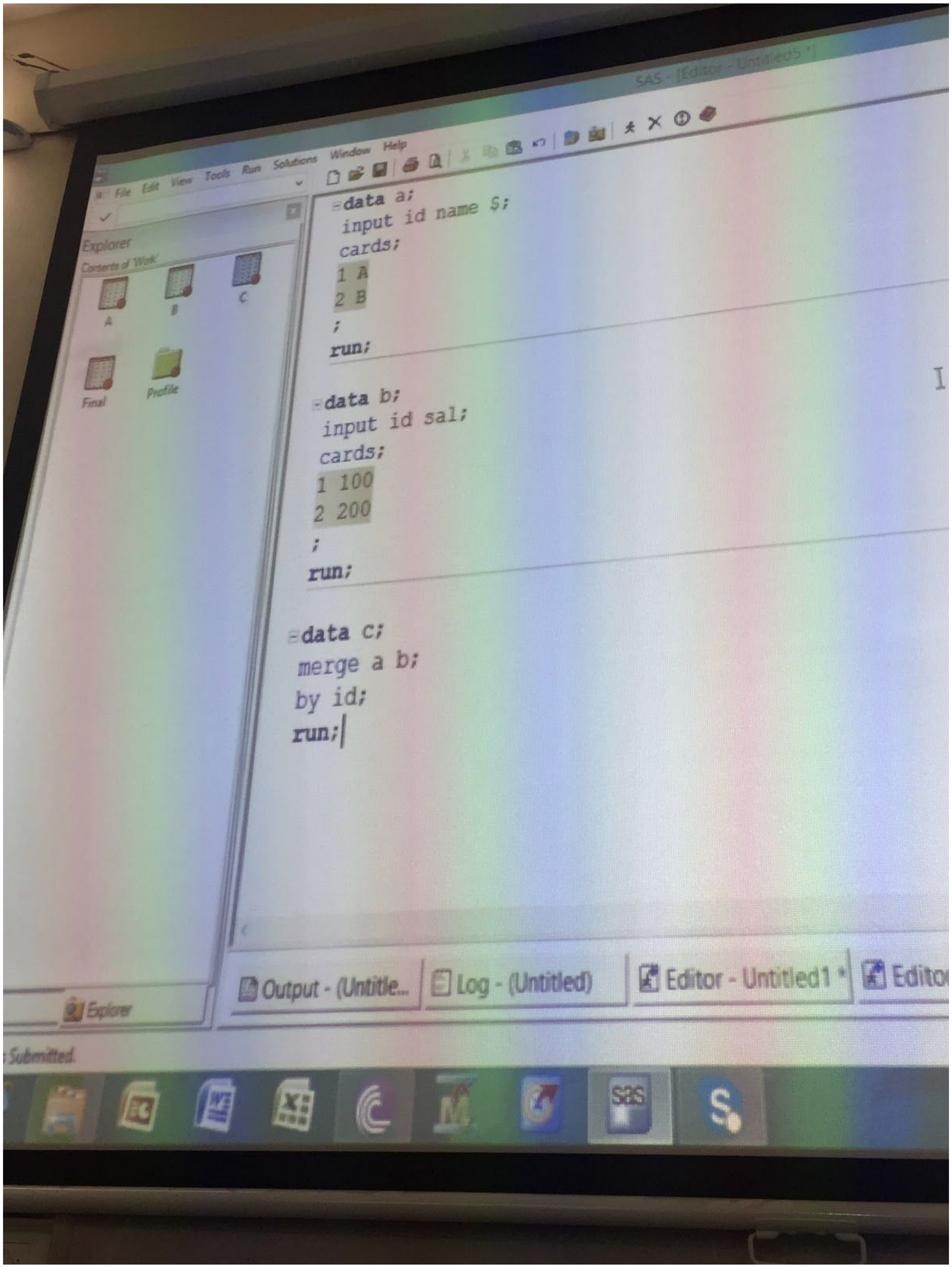
```
Data C;
merge BA
by id;
run;
```



```
data a;
 input id name $;
 cards;
. A
2 B
;
run;

data b;
 input id sal;
 cards;
1 100
2 200
;
run;

data c;
 merge a b;
 by id;
run;
```



```
data a;
 input id name $;
 cards;
1 A
2 B
;
run;

data b;
 input id sal;
 cards;
1 100
2 200
;
run;

data c;
 merge a b;
 by id;
run;
```

Interview Question (CASE STUDY - 1)

Merge behaves like append when you don't have common values.

Eg:

```
data a;
input id name$;
cards;
1 A
2 B
;
run;
```

```
data b;
input id sal;
cards;
11    100
22    200
;
run;
```

```
data c;
merge a b;
by id;
run;
```

Output

id	name	sal
1	A	
2	B	
11		100
22		200

CASE STUDY - 2

```
data a;
input id name$ age;
cards;
1   A   12
2   B   15
;
run;
```

```
data b;
input id sal age;
cards;
1   100   45
2   200   55
;
run;
```

```
data c;
merge a b;
by id;
run;
```

Here, we have four variables - id, name, age, sal.

But, the issue is whether the age from dataset 'a' will come or dataset 'b'.

Acc., to concept of merging <u>overlapping</u> is done.

Here, the age from dataset 'b' will overwrite dataset 'a' age's value.

But here also the condition is that dataset 'b' should have the <u>updated values</u> of age, only then it will over-write the age value of 'a'.

<u>Output</u>

id	name	sal	age
1	A	100	45
2	B	200	55

CASE STUDY - 3

Suppose, from the previous example, if dataset 'a' have the updated value of age and we don't want dataset 'b' age's value to overwrite 'a'. In that case, drop the variable of age from dataset 'b', so that it should not overwrite.

```
data c;
merge a b (drop = age);   → By this, output
by id;                       will not be
run;                         overlapped.
```

Output

id	name	sal	age
1	A	100	12
2	B	200	15

→ 'age' from dataset 'a'.

Program

```
data ctc;
input empid sal;
cards;
1   100
2   300
3   900
;
run;
```

```
data new;
input empid sal;
cards;
1   100
1   110
1   120
1   135
2   395
3   900
3   950
;
run;
```

Here, we want to update the data of 'ctc' by 'new' by emp id (which is sorted) means with every empid, the updated salary should come.

1s updated salary = 135
2s " " = 395
3s " " = 950

↓

Program →
```
data ctc;
update ctc new;
by empid;
run;
```

Output

	empid	sal
1	1	135
2	2	395
3	3	950