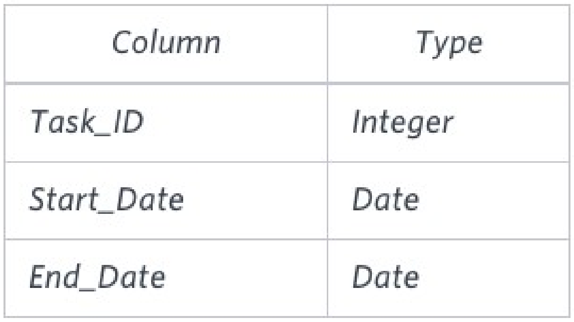
Task 1.You are given a table, *Projects*, containing three columns: *Task_ID*, *Start_Date* and *End_Date*. It is guaranteed that the difference between the *End_Date* and the *Start_Date* is equal to *1* day for each row in the table.

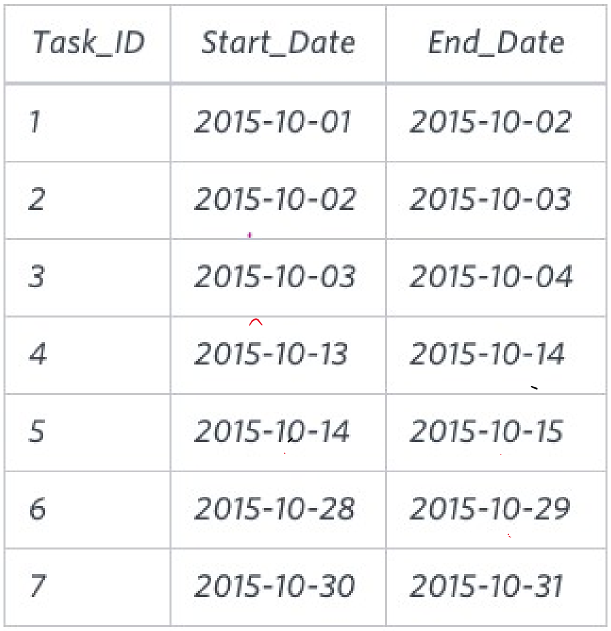


Column	Type
Task_ID	Integer
Start_Date	Date
End_Date	Date

If the *End_Date* of the tasks are consecutive, then they are part of the same project. Samantha is interested in finding the total number of different projects completed.

Write a query to output the start and end dates of projects listed by the number of days it took to complete the project in ascending order. If there is more than one project that have the same number of completion days, then order by the start date of the project.

Sample Input

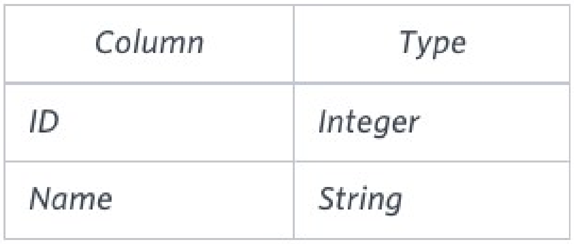


Task_ID	Start_Date	End_Date
1	2015-10-01	2015-10-02
2	2015-10-02	2015-10-03
3	2015-10-03	2015-10-04
4	2015-10-13	2015-10-14
5	2015-10-14	2015-10-15
6	2015-10-28	2015-10-29
7	2015-10-30	2015-10-31

Sample Output

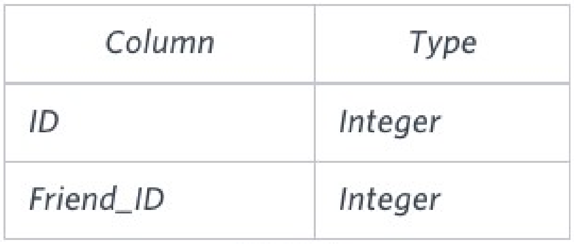
```
2015-10-28 2015-10-29
2015-10-30 2015-10-31
2015-10-13 2015-10-15
2015-10-01 2015-10-04
```

Task 2. You are given three tables: *Students*, *Friends* and *Packages*. *Students* contains two columns: *ID* and *Name*. *Friends* contains two columns: *ID* and *Friend_ID* (*ID* of the ONLY best friend). *Packages* contains two columns: *ID* and *Salary* (offered salary in $ thousands per month).



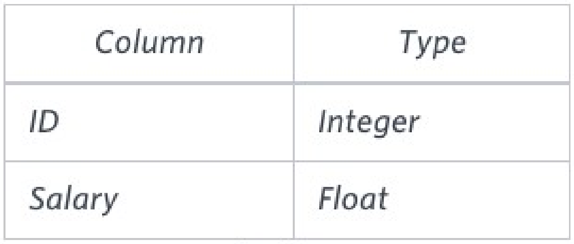
Column	Type
ID	Integer
Name	String

Students



Column	Type
ID	Integer
Friend_ID	Integer

Friends

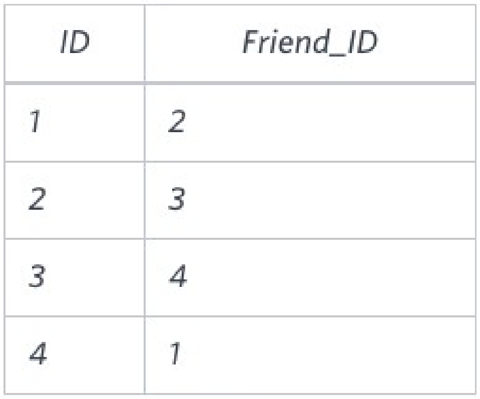


Column	Type
ID	Integer
Salary	Float

Packages

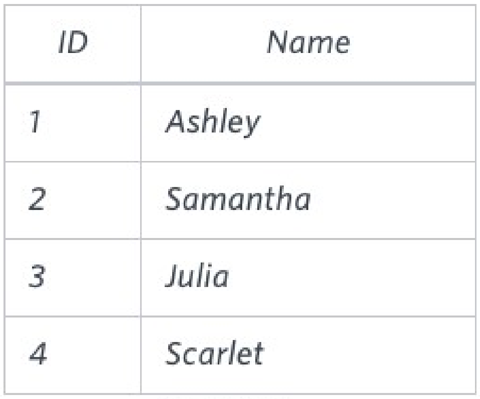
Write a query to output the names of those students whose best friends got offered a higher salary than them. Names must be ordered by the salary amount offered to the best friends. It is guaranteed that no two students got same salary offer.

Sample Input



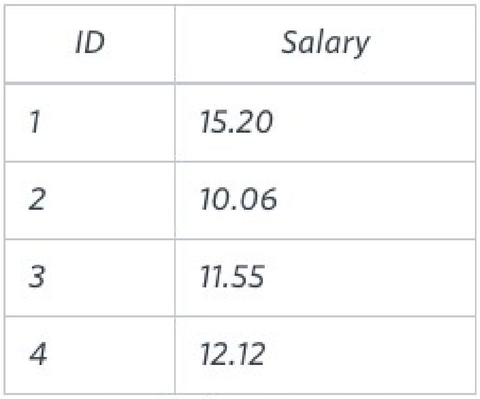
ID	Friend_ID
1	2
2	3
3	4
4	1

Friends



ID	Name
1	Ashley
2	Samantha
3	Julia
4	Scarlet

Students



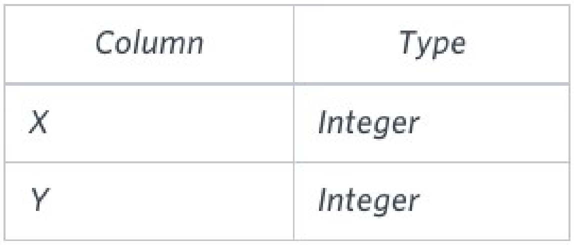
ID	Salary
1	15.20
2	10.06
3	11.55
4	12.12

Packages

Sample Output

```
Samantha
Julia
Scarlet
```

Task 3. You are given a table, *Functions*, containing two columns: *X* and *Y*.

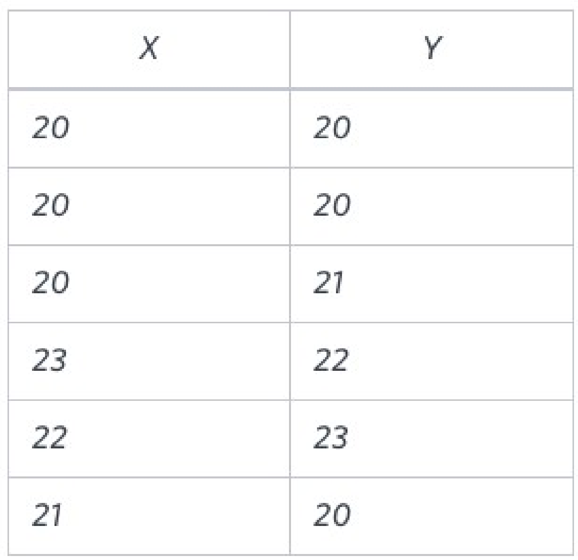


Column	Type
X	Integer
Y	Integer

Two pairs (X_1, Y_1) and (X_2, Y_2) are said to be *symmetric pairs* if $X_1 = Y_2$ and $X_2 = Y_1$.

Write a query to output all such *symmetric pairs* in ascending order by the value of *X*.

Sample Input



X	Y
20	20
20	20
20	21
23	22
22	23
21	20

Sample Output

```
20 20
20 21
22 23
```

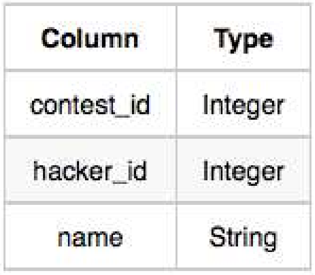
Task 4. Samantha interviews many candidates from different colleges using coding challenges and contests. Write a query to print the *contest_id*, *hacker_id*, *name*, and the sums of *total_submissions*, *total_accepted_submissions*, *total_views*, and *total_unique_views* for each contest sorted by *contest_id*. Exclude the contest from the result if all four sums are .

Note: A specific contest can be used to screen candidates at more than one college, but each college only holds screening contest.

Input Format

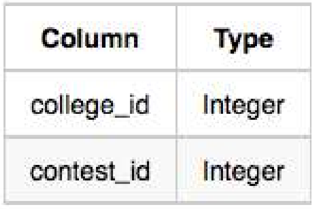
The following tables hold interview data:

- *Contests:* The *contest_id* is the id of the contest, *hacker_id* is the id of the hacker who created the contest, and *name* is the name of the hacker.



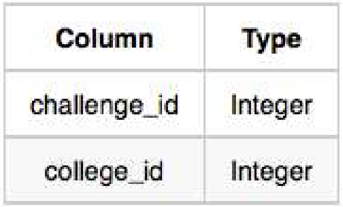
Column	Type
contest_id	Integer
hacker_id	Integer
name	String

- *Colleges:* The *college_id* is the id of the college, and *contest_id* is the id of the contest that Samantha used to screen the candidates.



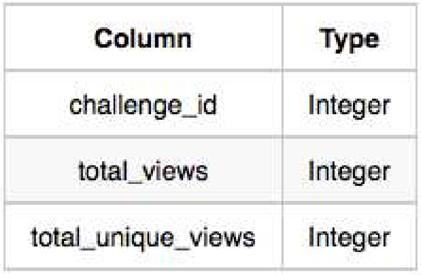
Column	Type
college_id	Integer
contest_id	Integer

- *Challenges:* The *challenge_id* is the id of the challenge that belongs to one of the contests whose contest_id Samantha forgot, and *college_id* is the id of the college where the challenge was given to candidates.



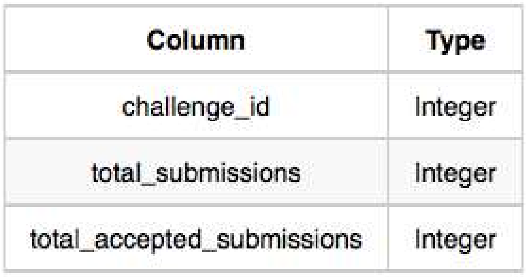
Column	Type
challenge_id	Integer
college_id	Integer

- *View_Stats:* The *challenge_id* is the id of the challenge, *total_views* is the number of times the challenge was viewed by candidates, and *total_unique_views* is the number of times the challenge was viewed by unique candidates.



Column	Type
challenge_id	Integer
total_views	Integer
total_unique_views	Integer

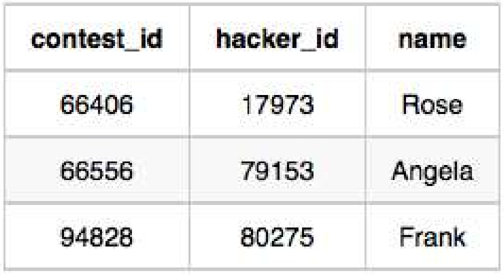
- *Submission_Stats:* The *challenge_id* is the id of the challenge, *total_submissions* is the number of submissions for the challenge, and *total_accepted_submission* is the number of submissions that achieved full scores.



Column	Type
challenge_id	Integer
total_submissions	Integer
total_accepted_submissions	Integer

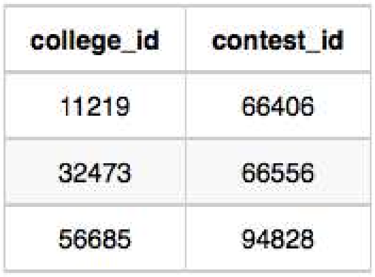
Sample Input

Contests Table:



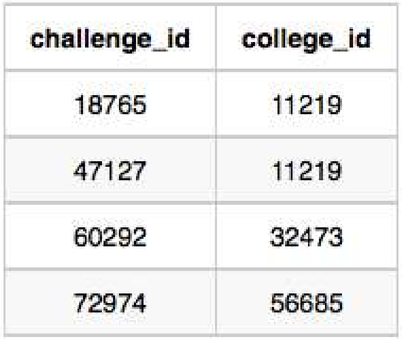
contest_id	hacker_id	name
66406	17973	Rose
66556	79153	Angela
94828	80275	Frank

Colleges Table:



college_id	contest_id
11219	66406
32473	66556
56685	94828

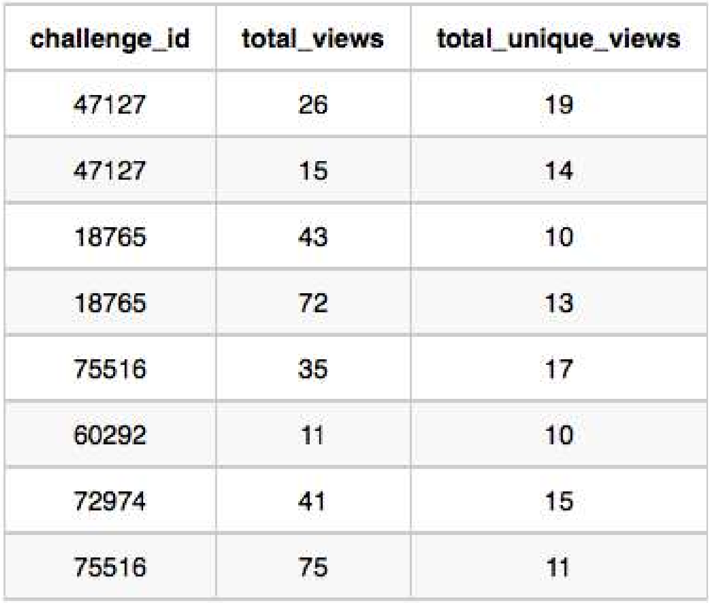
Challenges Table:



challenge_id	college_id
18765	11219
47127	11219
60292	32473
72974	56685

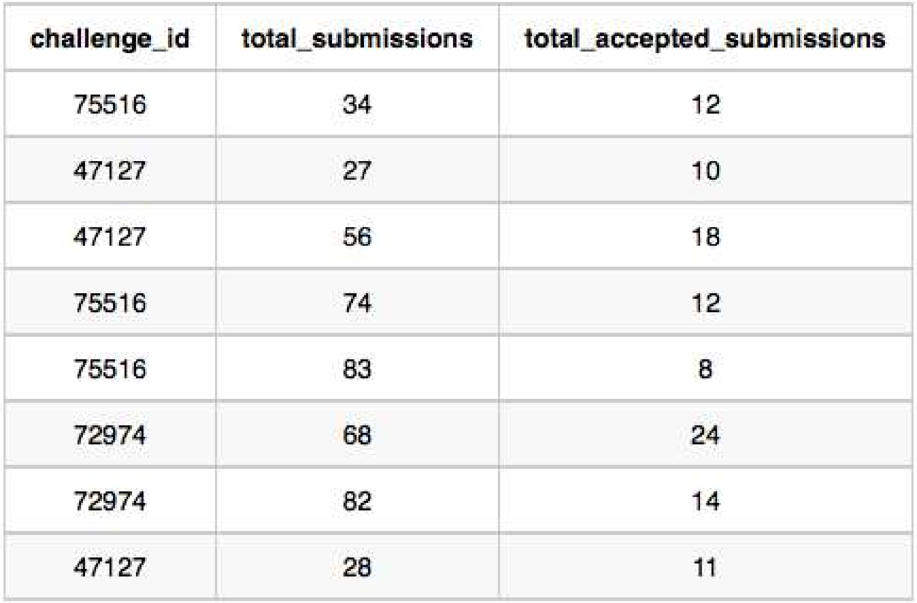
View_Stats

Table:



challenge_id	total_views	total_unique_views
47127	26	19
47127	15	14
18765	43	10
18765	72	13
75516	35	17
60292	11	10
72974	41	15
75516	75	11

Submission_Stats Table:



challenge_id	total_submissions	total_accepted_submissions
75516	34	12
47127	27	10
47127	56	18
75516	74	12
75516	83	8
72974	68	24
72974	82	14
47127	28	11

Sample Output

```
66406 17973 Rose 111 39 156 56
66556 79153 Angela 0 0 11 10
94828 80275 Frank 150 38 41 15
```

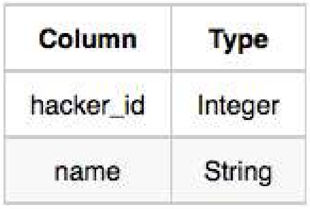
Task 5. Julia conducted a days of learning SQL contest. The start date of the contest was *March 01, 2016* and the end date was *March 15, 2016*.

Write a query to print total number of unique hackers who made at least submission each day (starting on the first day of the contest), and find the *hacker_id* and *name* of the hacker who made maximum number of submissions each day. If more than one such hacker has a maximum number of submissions, print the lowest *hacker_id*. The query should print this information for each day of the contest, sorted by the date.

Input Format

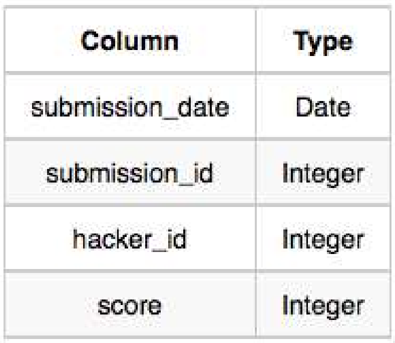
The following tables hold contest data:

- *Hackers:* The *hacker_id* is the id of the hacker, and *name* is the name of the hacker.



Column	Type
hacker_id	Integer
name	String

- *Submissions:* The *submission_date* is the date of the submission, *submission_id* is the id of the submission, *hacker_id* is the id of the hacker who made the submission, and *score* is the score of the submission.

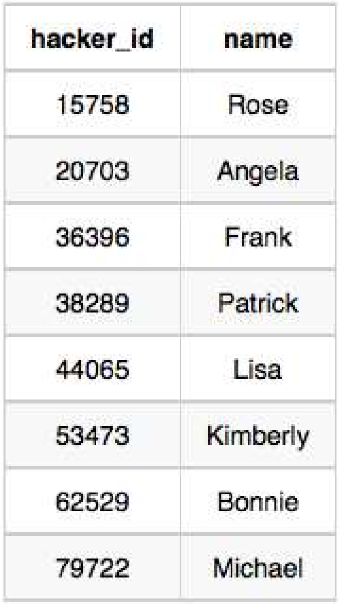


Column	Type
submission_date	Date
submission_id	Integer
hacker_id	Integer
score	Integer

Sample Input

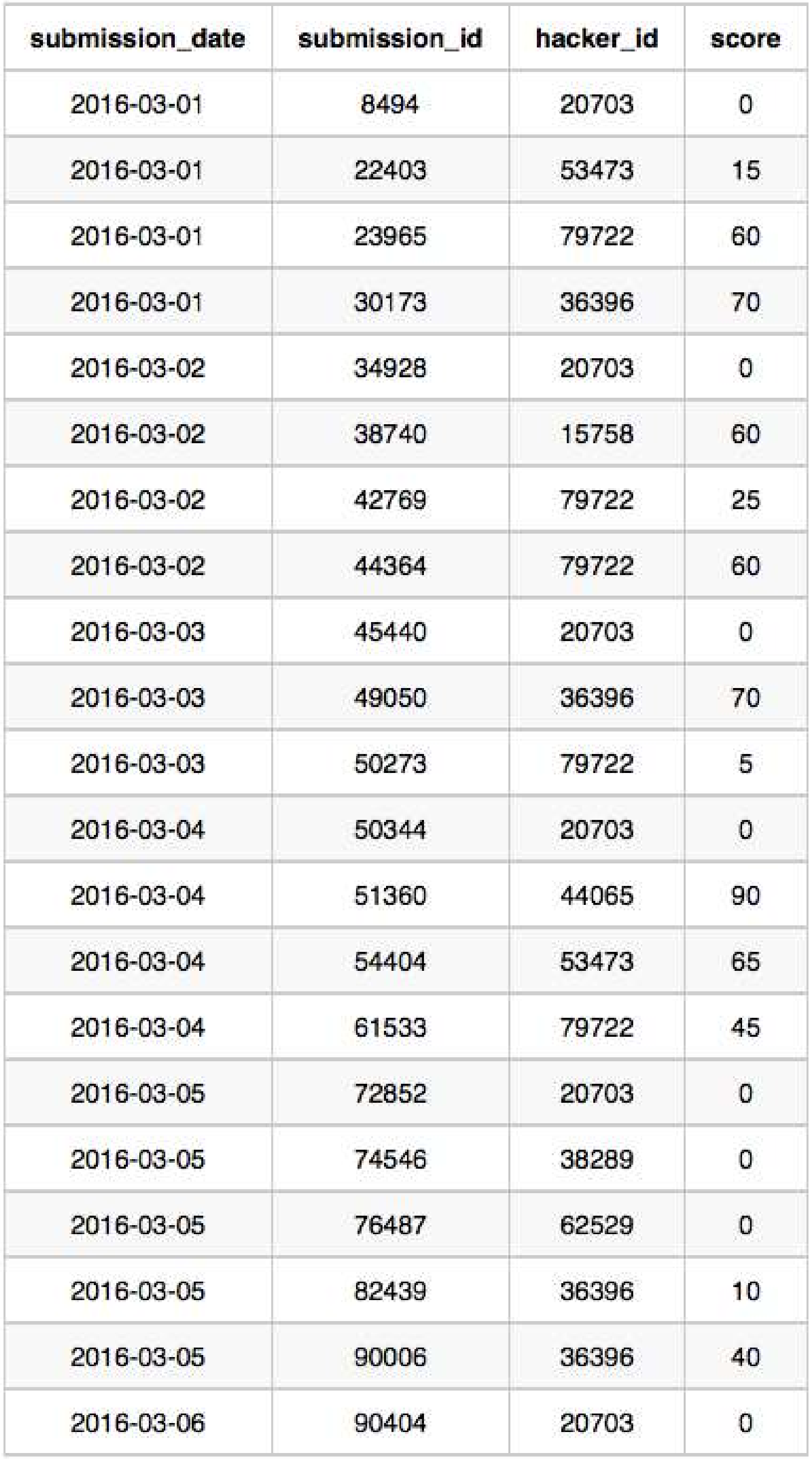
For the following sample input, assume that the end date of the contest was *March 06, 2016*.

Hackers Table:



hacker_id	name
15758	Rose
20703	Angela
36396	Frank
38289	Patrick
44065	Lisa
53473	Kimberly
62529	Bonnie
79722	Michael

Submissions Table:



submission_date	submission_id	hacker_id	score
2016-03-01	8494	20703	0
2016-03-01	22403	53473	15
2016-03-01	23965	79722	60
2016-03-01	30173	36396	70
2016-03-02	34928	20703	0
2016-03-02	38740	15758	60
2016-03-02	42769	79722	25
2016-03-02	44364	79722	60
2016-03-03	45440	20703	0
2016-03-03	49050	36396	70
2016-03-03	50273	79722	5
2016-03-04	50344	20703	0
2016-03-04	51360	44065	90
2016-03-04	54404	53473	65
2016-03-04	61533	79722	45
2016-03-05	72852	20703	0
2016-03-05	74546	38289	0
2016-03-05	76487	62529	0
2016-03-05	82439	36396	10
2016-03-05	90006	36396	40
2016-03-06	90404	20703	0

Sample Output

```
2016-03-01 4 20703 Angela
2016-03-02 2 79722 Michael
2016-03-03 2 20703 Angela
```

```
2016-03-04 2 20703 Angela
2016-03-05 1 36396 Frank
2016-03-06 1 20703 Angela
```

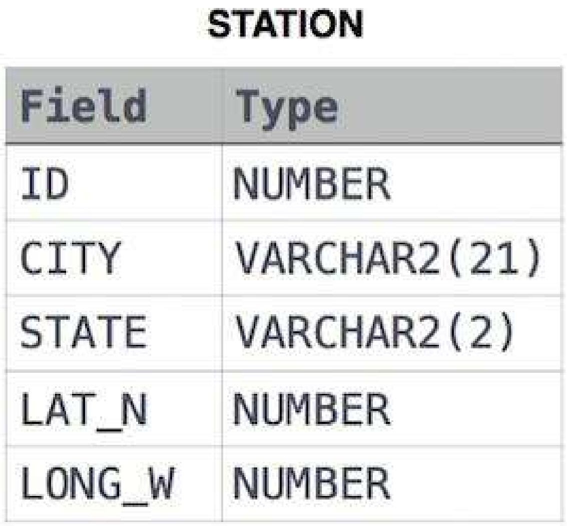
Task 6. Consider P1(a,b) and P2(c,d) to be two points on a *2D* plane.

- happens to equal the minimum value in *Northern Latitude* (*LAT_N* in **STATION**).
- happens to equal the minimum value in *Western Longitude* (*LONG_W* in **STATION**).
- happens to equal the maximum value in *Northern Latitude* (*LAT_N* in **STATION**).
- happens to equal the maximum value in *Western Longitude* (*LONG_W* in **STATION**).

Query the Manhattan Distance between points P1 and P2 and round it to a scale of decimal places.

Input Format

The **STATION** table is described as follows:



STATION

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where *LAT_N* is the northern latitude and *LONG_W* is the western longitude.

Task 7. Write a query to print all *prime numbers* less than or equal to 1000. Print your result on a single line, and use the ampersand (&) character as your separator (instead of a space).

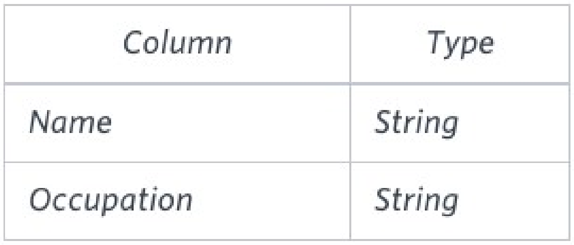
For example, the output for all prime numbers <=10 would be:

Task 8. Pivot the *Occupation* column in **OCCUPATIONS** so that each *Name* is sorted alphabetically and displayed underneath its corresponding *Occupation*. The output column headers should be *Doctor*, *Professor*, *Singer*, and *Actor*, respectively.

Note: Print **NULL** when there are no more names corresponding to an occupation.

Input Format

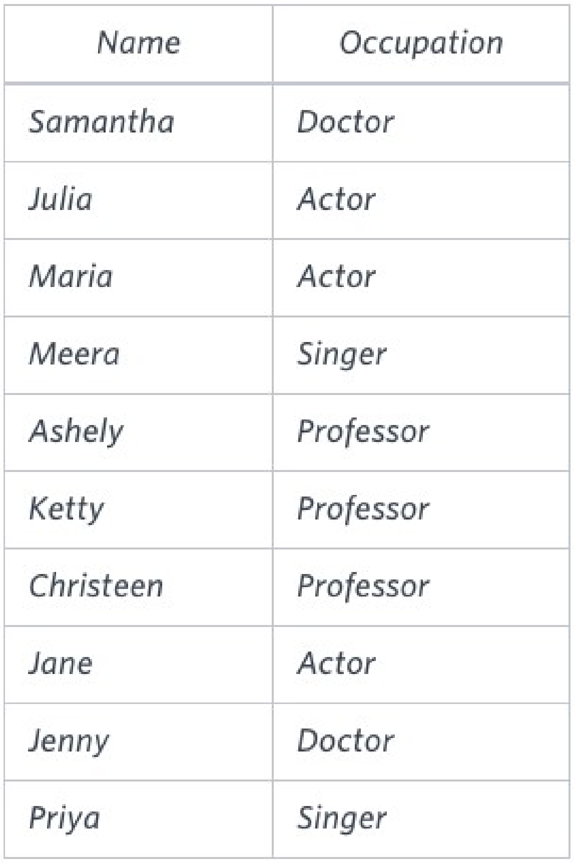
The **OCCUPATIONS** table is described as follows:



Column	Type
Name	String
Occupation	String

Occupation will only contain one of the following values: **Doctor**, **Professor**, **Singer** or **Actor**.

Sample Input

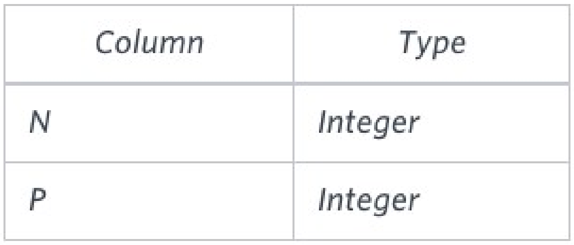


Name	Occupation
Samantha	Doctor
Julia	Actor
Maria	Actor
Meera	Singer
Ashely	Professor
Ketty	Professor
Christeen	Professor
Jane	Actor
Jenny	Doctor
Priya	Singer

Sample Output

```
Jenny    Ashley     Meera  Jane
Samantha Christeen  Priya  Julia
NULL     Ketty      NULL   Maria
```

Task 9. You are given a table, BST, containing two columns: N and P, where N represents the value of a node in $Binary\ Tree$, and P is the parent of N.

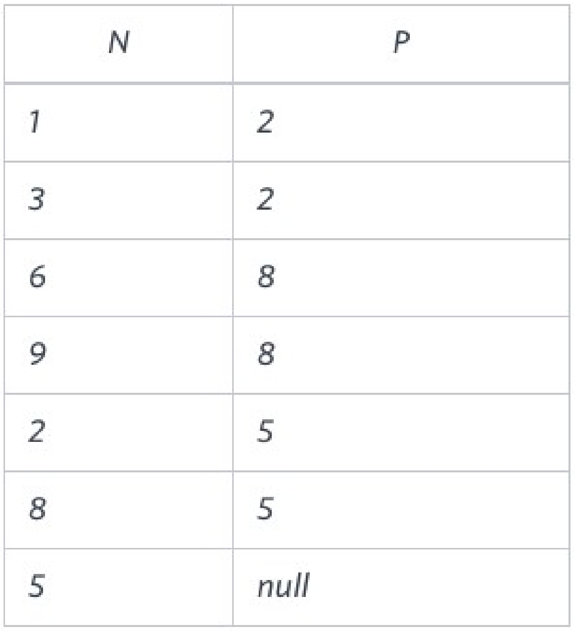


Column	Type
N	Integer
P	Integer

Write a query to find the node type of $Binary\ Tree$ ordered by the value of the node. Output one of the following for each node:

- $Root$: If node is root node.
- $Leaf$: If node is leaf node.
- $Inner$: If node is neither root nor leaf node.

Sample Input



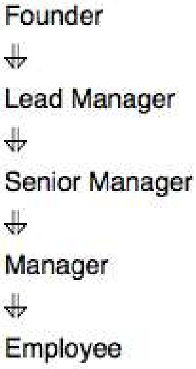
N	P
1	2
3	2
6	8
9	8
2	5
8	5
5	null

Sample Output

```
1 Leaf
2 Inner
3 Leaf
5 Root
6 Leaf
```

```
8 Inner
9 Leaf
```

Task 10. Amber's conglomerate corporation just acquired some new companies. Each of the companies follows this hierarchy:



Given the table schemas below, write a query to print the *company_code*, *founder* name, total number of *lead* managers, total number of *senior* managers, total number of *managers*, and total number of *employees*. Order your output by ascending *company_code*.

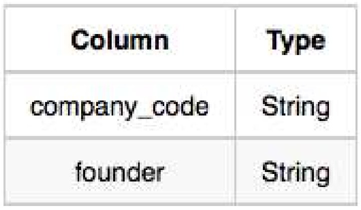
Note:

- The tables may contain duplicate records.
- The *company_code* is string, so the sorting should not be **numeric**. For example, if the *company_codes* are *C_1*, *C_2*, and *C_10*, then the ascending *company_codes* will be *C_1*, *C_10*, and *C_2*.

Input Format

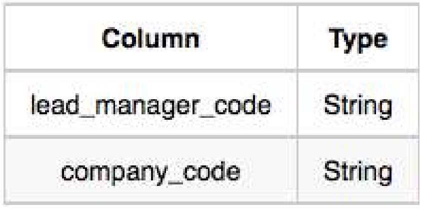
The following tables contain company data:

- *Company:* The *company_code* is the code of the company and *founder* is the founder of the company.



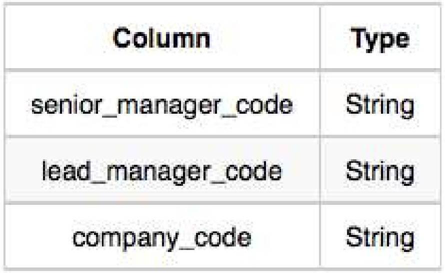
Column	Type
company_code	String
founder	String

- *Lead_Manager:* The *lead_manager_code* is the code of the lead manager, and the *company_code* is the code of the working company.



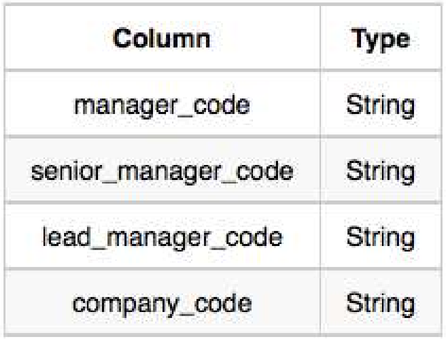
Column	Type
lead_manager_code	String
company_code	String

- *Senior_Manager:* The *senior_manager_code* is the code of the senior manager, the *lead_manager_code* is the code of its lead manager, and the *company_code* is the code of the working company.



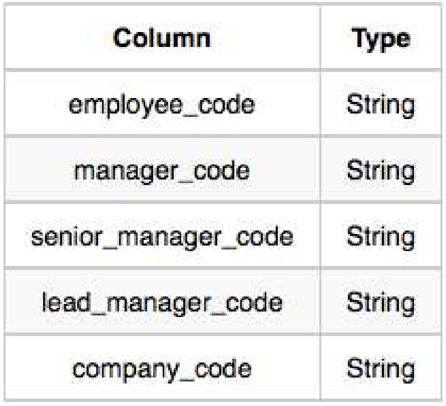
Column	Type
senior_manager_code	String
lead_manager_code	String
company_code	String

- *Manager:* The *manager_code* is the code of the manager, the *senior_manager_code* is the code of its senior manager, the *lead_manager_code* is the code of its lead manager, and the *company_code* is the code of the working company.



Column	Type
manager_code	String
senior_manager_code	String
lead_manager_code	String
company_code	String

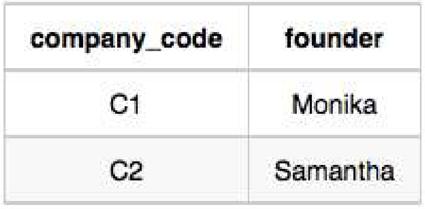
- *Employee:* The *employee_code* is the code of the employee, the *manager_code* is the code of its manager, the *senior_manager_code* is the code of its senior manager, the *lead_manager_code* is the code of its lead manager, and the *company_code* is the code of the working company.



Column	Type
employee_code	String
manager_code	String
senior_manager_code	String
lead_manager_code	String
company_code	String

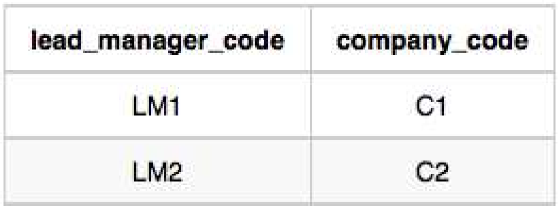
Sample Input

Company Table:



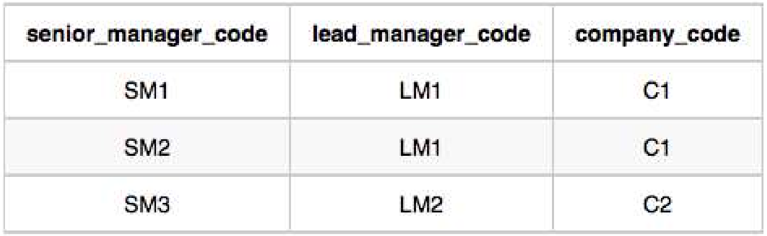
company_code	founder
C1	Monika
C2	Samantha

Lead_Manager Table:



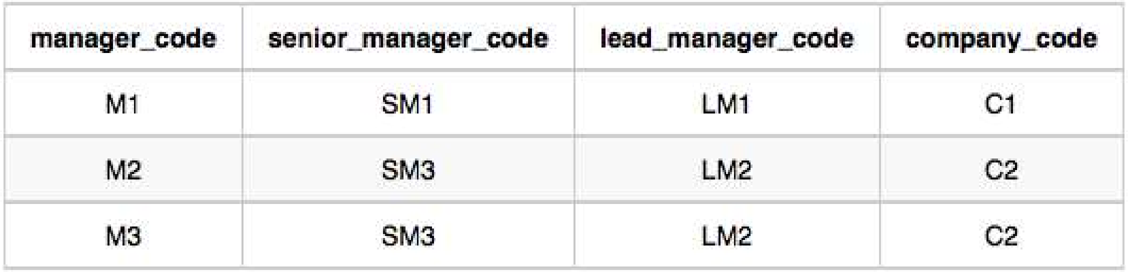
lead_manager_code	company_code
LM1	C1
LM2	C2

Senior_Manager Table:



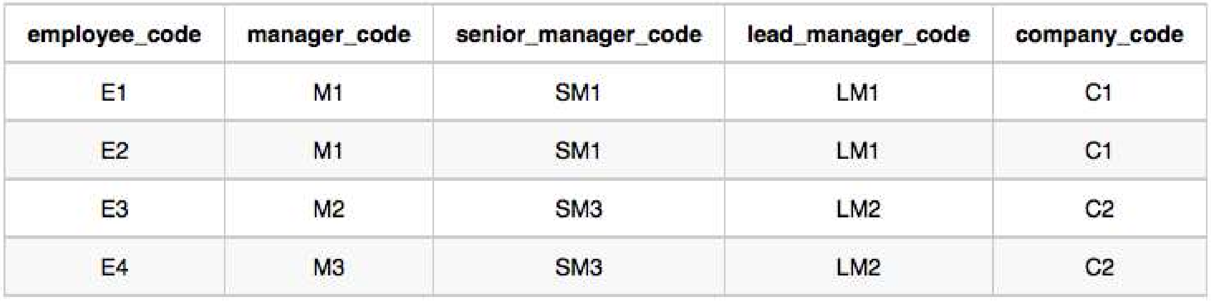
senior_manager_code	lead_manager_code	company_code
SM1	LM1	C1
SM2	LM1	C1
SM3	LM2	C2

Manager Table:



manager_code	senior_manager_code	lead_manager_code	company_code
M1	SM1	LM1	C1
M2	SM3	LM2	C2
M3	SM3	LM2	C2

Employee Table:

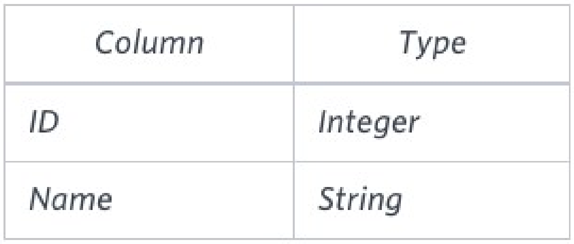


employee_code	manager_code	senior_manager_code	lead_manager_code	company_code
E1	M1	SM1	LM1	C1
E2	M1	SM1	LM1	C1
E3	M2	SM3	LM2	C2
E4	M3	SM3	LM2	C2

Sample Output

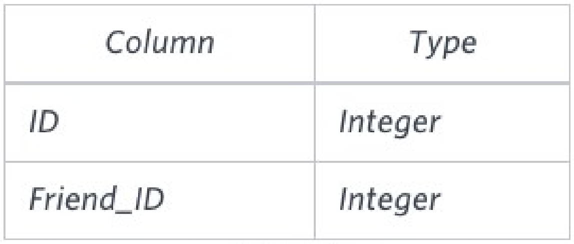
```
C1 Monika 1 2 1 2
C2 Samantha 1 1 2 2
```

Task 11. You are given three tables: *Students*, *Friends* and *Packages*. *Students* contains two columns: *ID* and *Name*. *Friends* contains two columns: *ID* and *Friend_ID* (*ID* of the ONLY best friend). *Packages* contains two columns: *ID* and *Salary* (offered salary in $ thousands per month).



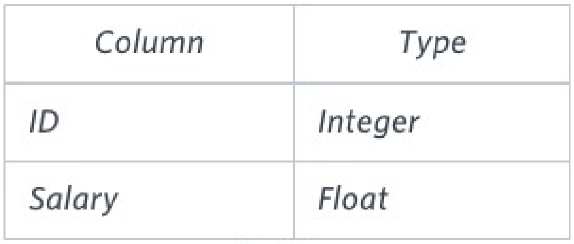
Column	Type
ID	Integer
Name	String

Students



Column	Type
ID	Integer
Friend_ID	Integer

Friends

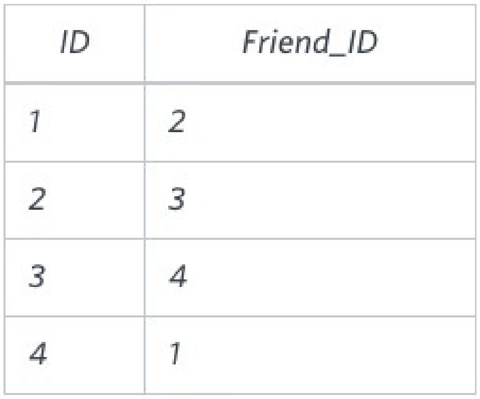


Column	Type
ID	Integer
Salary	Float

Packages

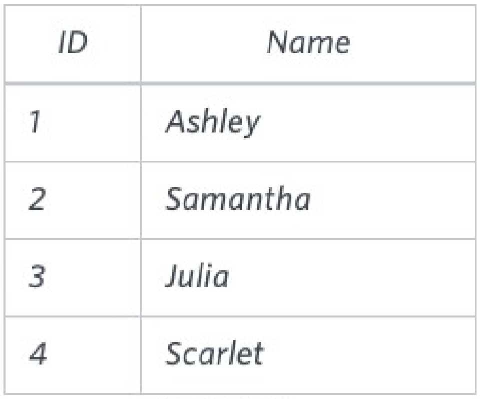
Write a query to output the names of those students whose best friends got offered a higher salary than them. Names must be ordered by the salary amount offered to the best friends. It is guaranteed that no two students got same salary offer.

Sample Input



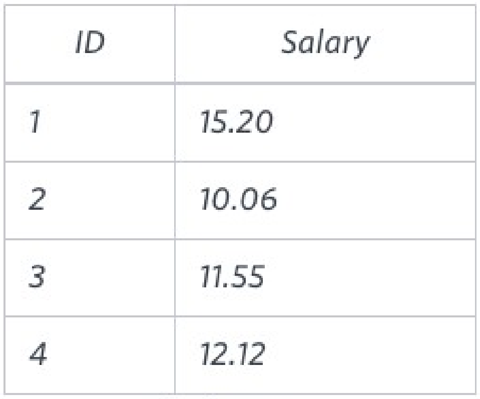
ID	Friend_ID
1	2
2	3
3	4
4	1

Friends



ID	Name
1	Ashley
2	Samantha
3	Julia
4	Scarlet

Students



ID	Salary
1	15.20
2	10.06
3	11.55
4	12.12

Packages

Sample Output

```
Samantha
Julia
Scarlet
```

Task 12. Display ratio of cost of job family in percentage by India and international (refer simulation data).

Task 13. Find ratio of cost and revenue of a BU month on month.

Task 14. Show headcounts of sub band and percentage of headcount (without join, subquery and inner query).

Task 15. Find top 5 employees according to salary (without order by).

Task 16. Swap value of two columns in a table without using third variable or a table.

Task 17. Create a user ,create a login for that user provide permissions of DB_owner to the user.

Task 18. Find Weighted average cost of employees month on month in a BU.

Task 19. Samantha was tasked with calculating the average monthly salaries for all employees in the **EMPLOYEES** table, but did not realize her keyboard's 0 key was broken until after completing the calculation. She wants your help finding the difference between her miscalculation (using salaries with any zeroes removed), and the actual average salary.

Write a query calculating the amount of error (i.e.: actual – miscalculated average monthly salaries), and round it up to the next integer.

Task 20. Copy new data of one table to another(you do not have indicator for new data and old data).