1. Query all columns for all American cities in the **CITY** table with populations larger than 100000.
The **CountryCode** for America is USA.
The **CITY** table is described as follows:

CITY

Field	Type
ID	NUMBER
NAME	VARCHAR2(17)
COUNTRYCODE	VARCHAR2(3)
DISTRICT	VARCHAR2(20)
POPULATION	NUMBER

2. Generate the following two result sets:
1.Query an *alphabetically ordered* list of all names in **OCCUPATIONS**, immediately followed by the first letter of each profession as a parenthetical (i.e.: enclosed in parentheses). For
example: AnActorName(A), ADoctorName(D), AProfessorName(P), and ASingerName(S).
2.Query the number of ocurrences of each occupation in **OCCUPATIONS**. Sort the occurrences in *ascending order*, and output them in the following format:
There are a total of [occupation_count] [occupation]s.
where [occupation_count] is the number of occurrences of an occupation in **OCCUPATIONS** and [occupation] is the *lowercase* occupation name. If more than one *Occupation* has the same [occupation_count], they should be ordered alphabetically.
Note: There will be at least two entries in the table for each type of occupation.
Input Format
The **OCCUPATIONS** table is described as follows:

Column	Type
Name	String
Occupation	String

the following values: **Doctor**, **Professor**, **Singer** or **Actor**.
Sample Input
An **OCCUPATIONS** table that contains the following records:

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

Sample Output

Ashely(P)
Christeen(P)
Jane(A)
Jenny(D)
Julia(A)
Ketty(P)
Maria(A)
Meera(S)
Priya(S)
Samantha(D)
There are a total of 2 doctors.
There are a total of 2 singers.
There are a total of 3 actors.
There are a total of 3 professors.

Explanation

The results of the first query are formatted to the problem description's specifications.
The results of the second query are ascendingly ordered first by number of names corresponding to each profession (2<=2<=3<=3), and then alphabetically by profession (doctor<= singer, and actor<=professor).

3. Employees With Missing Information

Table: Employees

Column Name	Type
employee_id	int
name	varchar

employee_id is the primary key for this table.
Each row of this table indicates the name of the employee whose ID is employee_id.

Table: Salaries

Column Name	Type
employee_id	int
salary	int

employee_id is the primary key for this table.
Each row of this table indicates the salary of the employee whose ID is employee_id.

Write an SQL query to report the IDs of all the employees with missing information. The information of an employee is missing if:
The employee's name is missing, or
The employee's salary is missing.
Return the result table ordered by employee_id in ascending order.
The query result format is in the following example.
Example 1:
Input:
Employees table:

employee_id	name
2	Crew
4	Haven
5	Kristian

Salaries table:

employee_id	salary
5	76071
1	22517
4	63539

Output:

employee_id
1
2

Explanation:
Employees 1, 2, 4, and 5 are working at this company.
The name of employee 1 is missing.
The salary of employee 2 is missing.

Example 1:
Input:
Logins table:

user_id	time_stamp
6	2020-06-30 15:06:07
6	2021-04-21 14:06:06
6	2019-03-07 00:18:15
8	2020-02-01 05:10:53
8	2020-12-30 00:46:50
2	2020-01-16 02:49:50
2	2019-08-25 07:59:08
14	2019-07-14 09:00:00
14	2021-01-06 11:59:59

Output:

user_id	last_stamp
6	2020-06-30 15:06:07
8	2020-12-30 00:46:50
2	2020-01-16 02:49:50

Explanation:
User 6 logged into their account 3 times but only once in 2020, so we include this login in the result table.
User 8 logged into their account 2 times in 2020, once in February and once in December. We include only the latest one (December) in the result table.
User 2 logged into their account 2 times but only once in 2020, so we include this login in the result table.
User 14 did not login in 2020, so we do not include them in the result table.

5. Customer Who Visited but Did Not Make Any Transactions

Table: Visits

Column Name	Type
visit_id	int
customer_id	int

visit_id is the primary key for this table.
This table contains information about the customers who visited the mall.

Table: Transactions

Column Name	Type
transaction_id	int
visit_id	int
amount	int

transaction_id is the primary key for this table.This table contains information about the transactions made during the visit_id.

Write a SQL query to find the IDs of the users who visited without making any transactions and the number of times they made these types of visits.
Return the result table sorted in any order. The query result format is in the following example.

Example 1:
Input:
Visits

visit_id	customer_id
1	23
2	9
4	30
5	54
6	96
7	54
8	54

Transactions

transaction_id	visit_id	amount
2	5	310
3	5	300
9	5	200
12	1	910
13	2	970

Output:

customer_id	count_no_trans
54	2
30	1
96	1

Explanation:
Customer with id = 23 visited the mall once and made one transaction during the visit with id = 12.
Customer with id = 9 visited the mall once and made one transaction during the visit with id = 13.
Customer with id = 30 visited the mall once and did not make any transactions.
Customer with id = 54 visited the mall three times. During 2 visits they did not make any transactions, and during one visit they made 3 transactions.
Customer with id = 96 visited the mall once and did not make any transactions.
As we can see, users with IDs 30 and 96 visited the mall one time without making any transactions. Also, user 54 visited the mall twice and did not make any transactions.

6. Daily Leads and Partners

Table: DailySales

```
+-------------+---------+
| Column Name | Type |
+-------------+---------+
| date_id     | date |
| make_name   | varchar |
| lead_id     | int |
| partner_id  | int |
+-------------+---------+
```

This table does not have a primary key.
This table contains the date and the name of the product sold and the IDs of the lead and partner it was sold to.
The name consists of only lowercase English letters.

Write an SQL query that will, for each date_id and make_name, return the number of distinct lead_id's and distinct partner_id's.

Return the result table in any order.

The query result format is in the following example.

Example 1:

Input:
DailySales table:

```
+-----------+-----------+---------+------------+
| date_id   | make_name | lead_id | partner_id |
+-----------+-----------+---------+------------+
| 2020-12-8 | toyota    | 0 | 1 |
| 2020-12-8 | toyota    | 1 | 0 |
| 2020-12-8 | toyota    | 1 | 2 |
| 2020-12-7 | toyota    | 0 | 2 |
| 2020-12-7 | toyota    | 0 | 1 |
| 2020-12-8 | honda     | 1 | 2 |
| 2020-12-8 | honda     | 2 | 1 |
| 2020-12-7 | honda     | 0 | 1 |
| 2020-12-7 | honda     | 1 | 2 |
| 2020-12-7 | honda     | 2 | 1 |
+-----------+-----------+---------+------------+
```

Output:

```
+-----------+-----------+--------------+-----------------+
| date_id   | make_name | unique_leads | unique_partners |
+-----------+-----------+--------------+-----------------+
| 2020-12-8 | toyota    | 2 | 3 |
| 2020-12-7 | toyota    | 1 | 2 |
| 2020-12-8 | honda     | 2 | 2 |
| 2020-12-7 | honda     | 3 | 2 |
+-----------+-----------+--------------+-----------------+
```

Explanation:
For 2020-12-8, toyota gets leads = [0, 1] and partners = [0, 1, 2]
while honda gets leads = [1, 2] and partners = [1, 2].
For 2020-12-7, toyota gets leads = [0] and partners = [1, 2]
while honda gets leads = [0, 1, 2] and partners = [1, 2].

7. Patients With a Condition

Table: Patients

+--------------+---------+
| Column Name | Type |
+--------------+---------+
patient_id	int
patient_name	varchar
conditions	varchar
+--------------+---------+
patient_id is the primary key for this table.
'conditions' contains 0 or more code separated by spaces.
This table contains information of the patients in the hospital.

Write an SQL query to report the patient_id, patient_name and conditions of the patients who have Type I Diabetes. Type I Diabetes always starts with DIAB1 prefix.

Return the result table in any order.

The query result format is in the following example.
Example 1:

Input:
Patients table:
+------------+--------------+--------------+
| patient_id | patient_name | conditions |
+------------+--------------+--------------+
1	Daniel	YFEV COUGH
2	Alice	
3	Bob	DIAB100 MYOP
4	George	ACNE DIAB100
5	Alain	DIAB201
+------------+--------------+--------------+
Output:
+------------+--------------+--------------+
| patient_id | patient_name | conditions |
+------------+--------------+--------------+
| 3 | Bob | DIAB100 MYOP |
| 4 | George | ACNE DIAB100 |
+------------+--------------+--------------+
Explanation: Bob and George both have a condition that starts with DIAB1.

8. Write a query identifying the *type* of each record in the **TRIANGLES** table using its three side lengths. Output one of the following statements for each record in the table:
•**Equilateral**: It's a triangle with sides of equal length.
•**Isosceles**: It's a triangle with sides of equal length.
•**Scalene**: It's a triangle with sides of differing lengths.
•**Not A Triangle**: The given values of *A*, *B*, and *C* don't form a triangle.
Input Format
The **TRIANGLES** table is described as follows:

Column	*Type*
A	*Integer*
B	*Integer*
C	*Integer*

Each row in the table denotes the lengths of each of a triangle's three sides.
Sample Input

A	*B*	*C*
20	*20*	*23*
20	*20*	*20*
20	*21*	*22*
13	*14*	*30*

Sample Output
Isosceles
Equilateral
Scalene
Not A Triangle

9. Julia just finished conducting a coding contest, and she needs your help assembling the leaderboard! Write a query to print the respective *hacker_id* and *name* of hackers who achieved full scores for *more than one* challenge. Order your output in descending order by the total number of challenges in which the hacker earned a full score. If more than one hacker received full scores in same number of challenges, then sort them by ascending *hacker_id*.

Input Format
The following tables contain 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

Difficulty: The *difficult_level* is the level of difficulty of the challenge, and *score* is the score of the challenge for the difficulty level.

Column	Type
difficulty_level	Integer
score	Integer

Challenges: The *challenge_id* is the id of the challenge, the *hacker_id* is the id of the hacker who created the challenge, and *difficulty_level* is the level of difficulty of the challenge.

Column	Type
challenge_id	Integer
hacker_id	Integer
difficulty_level	Integer

Submissions: The *submission_id* is the id of the submission, *hacker_id* is the id of the hacker who made the submission, *challenge_id* is the id of the challenge that the submission belongs to, and *score* is the score of the submission.

Column	Type
submission_id	Integer
hacker_id	Integer
challenge_id	Integer
score	Integer

Sample Input

Hackers Table:

hacker_id	name
5580	Rose
8439	Angela
27205	Frank
52243	Patrick
52348	Lisa
57645	Kimberly
77726	Bonnie
83082	Michael
86870	Todd
90411	Joe

Difficulty Table:

difficulty_level	score
1	20
2	30
3	40
4	60
5	80
6	100
7	120

Challenges Table:

challenge_id	hacker_id	difficulty_level
4810	77726	4
21089	27205	1
36566	5580	7
66730	52243	6
71055	52243	2

Submissions Table:

submission_id	hacker_id	challenge_id	score
68628	77726	36566	30
65300	77726	21089	10
40326	52243	36566	77
8941	27205	4810	4
83554	77726	66730	30
43353	52243	66730	0
55385	52348	71055	20
39784	27205	71055	23
94613	86870	71055	30
45788	52348	36566	0
93058	86870	36566	30
7344	8439	66730	92
2721	8439	4810	36
523	5580	71055	4
49105	52348	66730	0
55877	57645	66730	80
38355	27205	66730	35
3924	8439	36566	80
97397	90411	66730	100
84162	83082	4810	40
97431	90411	71055	30

Sample Output

90411 Joe

Challenges Table:

challenge_id	hacker_id	difficulty_level
4810	77726	4
21089	27205	1
36566	5580	7
66730	52243	6
71055	52243	2

Submissions Table:

submission_id	hacker_id	challenge_id	score
68628	77726	36566	30
65300	77726	21089	10
40326	52243	36566	77
8941	27205	4810	4
83554	77726	66730	30
43353	52243	66730	0
55385	52348	71055	20
39784	27205	71055	23
94613	86870	71055	30
45788	52348	36566	0
93058	86870	36566	30
7344	8439	66730	92
2721	8439	4810	36
523	5580	71055	4
49105	52348	66730	0
55877	57645	66730	80
38355	27205	66730	35
3924	8439	36566	80
97397	90411	66730	100
84162	83082	4810	40
97431	90411	71055	30

Sample Output
90411 Joe

10. You are given two tables: *Students* and *Grades*. *Students* contains three columns *ID*, *Name* and *Marks*.

Column	*Type*
ID	*Integer*
Name	*String*
Marks	*Integer*

Grades contains the following data:

Grade	*Min_Mark*	*Max_Mark*
1	*0*	*9*
2	*10*	*19*
3	*20*	*29*
4	*30*	*39*
5	*40*	*49*
6	*50*	*59*
7	*60*	*69*
8	*70*	*79*
9	*80*	*89*
10	*90*	*100*

Ketty gives *Eve* a task to generate a report containing three columns: *Name*, *Grade* and *Mark*. *Ketty* doesn't want the NAMES of those students who received a grade lower than *8*. The report must be in descending order by grade -- i.e. higher grades are entered first. If there is more than one student with the same grade (8-10) assigned to them, order those particular students by their name alphabetically. Finally, if the grade is lower than 8, use "NULL" as their name and list them by their grades in descending order. If there is more than one student with the same grade (1-7) assigned to them, order those particular students by their marks in ascending order.
Write a query to help Eve.

Sample Input

ID	Name	Marks
1	Julia	88
2	Samantha	68
3	Maria	99
4	Scarlet	78
5	Ashley	63
6	Jane	81

Sample Output

Maria 10 99
Jane 9 81
Julia 9 88
Scarlet 8 78
NULL 7 63
NULL 7 68

Note
Print "NULL" as the name if the grade is less than 8.

Explanation
Consider the following table with the grades assigned to the students:

ID	Name	Marks	Grade
1	Julia	88	9
2	Samantha	68	7
3	Maria	99	10
4	Scarlet	78	8
5	Ashley	63	7
6	Jane	81	9

So, the following students got *8*, *9* or *10* grades:
•*Maria (grade 10)*
•*Jane (grade 9)*
•*Julia (grade 9)*
•*Scarlet (grade 8)*

4. Table: Logins

Column Name	Type
user_id	int
time_stamp	datetime

(user_id, time_stamp) is the primary key for this table.
Each row contains information about the login time for the user with ID user_id.

Write an SQL query to report the latest login for all users in the year 2020. Do not include the users who did not login in 2020.
Return the result table in any order.
The query result format is in the following example.