MySQL Practice HackerRank Questions 1-23

Basic SELECT 1-13 Questions

Question 1: Range of Marks Scored

There is a database with the exam scores of every student. Write a query to print the maximum and minimum marks of the students. The result should be in the following format: MAX_MARKS MIN_MARKS

Schema

marks		
Name	**Type**	**Description**
ID	String	This is the student's ID. It is the primary key.
MARKS	Integer	These are the marks scored.

Sample Data Tables

marks	
ID	**MARKS**
abc123	30
def456	70
def123	50

Sample Output

```
70  30
```

Explanation:

Max marks = 70
Min marks = 30

Save your MySQL query code below

Question 2: The Superheroes Location

The location of the superheroes have been stored in the *SUPERHERO* table. Write a query to print the *IDs*, i.e., *SUPERHERO.ID* of the superheroes whose latitudes and longitudes both have a value smaller than *50*. The order output does not matter.

Schema

SUPERHERO		
Name	**Type**	**Description**
ID	Integer	A superhero ID in the inclusive range [1, 1000]. This field is the primary key.
NAMES	String	A superhero name. This field contains between *1* and *100* characters (inclusive).
LATITUDE	Float	The latitude of the superhero.
LONGITUDE	Float	The longitude of the superhero.

Output Format

The result should contain the *IDS* of the superheroes whose latitudes both have a value smaller than *50*.

```
SUPERHERO.ID
```

Sample Input

SUPERHERO			
ID	**NAME**	**LATITUDE**	**LONGITUDE**
1	Batman	50.23	85.45
2	Spiderman	65.43	65.66
3	Thor	45.34	30.89
4	Iron Man	85.34	80.98
5	Dead Pool	25.12	69.21
6	Hulk	30.34	20.98
7	Doctor Strange	40.45	40.56
8	Captain America	70.00	75.32

9	Avengers	81.32	90.84
10	Superman	85.32	45.78

Sample Output

```
3
6
7
```

Explanation

- For *Batman*, latitude = 50.23 and longitude = 85.45. So, neither is less than 50.
- For *Thor*, latitude = 45.34 and longitude = 30.89. So, both are less than 50.
- For *Deadpool*, latitude = 25.12 and longitude = 69.21. So, the latitude is less than 50 but the longitude is greater than 50.
- The remaining records are analyzed similarly.

Thor, Hulk, and *Doctor Strange* have latitudes and longitudes less than *50* so their *IDS* are printed.

Save your MySQL query code below

Question 3: Customers Credit Limit

A company maintains the data of its customers in the *CUSTOMER* table. Write a query to print the *ID*s and the *NAME*s of the customers who are from the USA and whose credit limit is greater than 100000, ordered by increasing *ID* number.

Output Format

The result should print the *ID*s and the *NAME*s of those customers who are from the USA and whose credit limit is greater than 100000, in ascending *ID* order an in the following format:

```
CUSTOMER.ID CUSTOMER.NAME
```

Sample Input

CUSTOMER			
ID	**NAME**	**COUNTRY**	**CREDITS**

1	Frances White	USA	200350
2	Carolyn Bradley	UK	15354
3	Annie Fernandez	France	359200
4	Ruth Hanson	Albania	1060
5	Paula Fuller	USA	14789
6	Bonnie Johnston	China	100243
7	Ruth Gutierrez	USA	998999
8	Ernest Thomas	Canada	500500
9	Joe Garza	UK	18782
10	Anne Harris	USA	158367

Sample Output

```
1 Frances White
7 Ruth Gutierrez
10 Anne Harris
```

Explanation

Description of some of the customers is given below:

- *Frances White* is from the *USA*, and his credit limit is *200350*, which is greater than *100000.*
- *Carolyn Bradley* is from the *UK*, and her credit limit is *15354* which is less than *100000.*
- *Paula Fuller* is from the *USA*, and her credit limit is *14789* which is less than *100000.*
- *Remaining records are analyzed similarly.*

So, *Frances White, Ruth Gutierrez,* and *Anne Harris* are from *USA* and credit is greater than *100000.*

Save your MySQL query code below

Question 4: The Superheroes Name

The information of the superheroes have been stored in the *SUPERHERO* table. Write a query to print the names, i.e., *SUPER.NAME* of the superheroes whose *NAME* has fewer than 7 characters. Sort the output in increasing order of their *ID*s.

Schema

SUPERHERO		
Name	**Type**	**Description**
ID	Integer	A superhero ID in the inclusive range [1, 1000]. This field is the primary key.
NAME	String	A superhero name. This field contains between *1* and *100* characters (inclusive).
LATITUDE	Float	The latitude of the superhero.
LONGITUDE	Float	The longitude of the superhero.

Output Format

The result should contain the names of the superheroes whose names have fewer than 7 characters. The result should be sorted in the increasing order of their *ID*s.

SUPERHERO.NAME

Sample Input

SUPERHERO			
ID	**NAME**	**LATITUDE**	**LONGITUDE**
1	Batman	50.0	85.0
2	Spiderman	65.0	65.0
3	Thor	45.0	30.0
4	Dead Pool	25.0	69.0
5	Hulk	30.0	20.0
6	Captain America	70.0	75.0
7	Superman	85.0	45.0

Sample Output

```
Batman
Thor
Hulk
```

Explanation

- For *Batman*, the id is *1*, and the name has *6* characters.
- For *Spiderman*, the id is *2*, and the name has *9* characters.
- For *Thor*, the id is *3*, and the name has *4* characters.
- Other records are analyzed similarly.

Batman, Thor, and Hulk have fewer than 7 characters in their names, so they are printed in increasing order of their ids: 1, 3, 5.

Save your MySQL query code below

Question 5: Undelivered Orders

A company maintains the information about its orders in the *ORDERS* table. Write a query to print the number of orders which are not yet declared, i.e., the *ORDERS.STATUS* is not equal to *DELIVERED*.

Schema

ORDERS		
Name	**Type**	**Description**
ID	Integer	A number in the inclusive range [1, 1000] which uniquely identifies the order.This field is the primary key.
ORDER_DATE	Date	A date when the order was placed.
STATUS	String	This is the order status. It can be PLACE, SHIPPED, IN TRANSIT, DELIVERED.
CUSTOMER_ID	Integer	A number in the inclusive range [1, 1000] which uniquely identifies the customer who placed the order.

Output Format

The output should be the count of orders that are not delivered yet.

```
COUNT_OF_ORDER_NOT_DELIVERED_YET
```

Sample Input

SUPERHERO			
ID	**ODER_DATE**	**STATUS**	**CUSTOMER_ID**
1	2003-01-06	PLACED	363
2	2003-01-06	PLACED	128
3	2003-01-06	IN TRANSIT	181
4	2003-01-06	DELIVERED	121
5	2003-01-07	DELIVERED	114
6	2003-01-07	IN TRANSIT	278
7	2003-01-07	PLACED	114
8	2003-05-05	IN TRANSIT	350
9	2003-05-05	DELIVERED	103

Sample Output

```
6
```

Explanation

Out of all the nine orders, the orders with *ID 4, 5,* and *9* are delivered while the rest six are still undelivered.

Save your MySQL query code below

Question 6: Order Management System

A retail company maintains the data of its customers in the *CUSTOMERS* table. Write a query to print the *ID*s and the *NAME*s of the customers, sorted by *CUSTOMER.NAME* in descending order. If two or more customers have the same *CUSTOMER.ID* in ascending order.

Schema

CUSTOMERS		
Name	**Type**	**Description**
ID	Integer	A customer ID in the inclusive range *[1, 1000].* This is the primary key.
NAME	String	A customer's name. This field contains between *1* and *100* characters (inclusive).
COUNTRY	String	The country of the customer.
CREDITS	Integer	The credit limit of the customer.

Output Format

The result should print the ids and the names of the customers, sorted by *CUSTOMER.NAME* in descending order. If two or more customers have the same *CUSTOMER.NAME,* then sort these by *CUSTOMER.ID* IN ascending order.

```
CUSTOMER.ID CUSTOMER.NAME
```

Sample Input

CUSTOMER			
ID	**NAME**	**COUNTRY**	**CREDITS**
1	Frances White	USA	200350
2	Carolyn Bradley	UK	15354
3	Annie Fernandez	France	359200
4	Ruth Hanson	Albania	1060
5	Paula Fuller	USA	14789
6	Bonnie Johnston	China	100243
7	Ruth Gutierrez	USA	998999
8	Ernest Thomas	Canada	500500
9	Joe Garza	UK	18782
10	Anne Harris	USA	158367

Sample Output

```
4 Ruth Hanson
7 Ruth Gutierrez
5 Paula Fuller
9 Joe Garza
1 Frances White
8 Ernest Thomas
2 Carolyn Bradley
6 Bonnie Johnston
3 Annie Fernandez
10 Anne Harris
```

Explanation
According to the lexicographical arrangement,

Ruth Hanson > Ruth Gutierrez > Paula Fuller > Joe Garza > Francis White > Ernest Thomas > Carolyn Bradley > Bonnie Johnston > Annie Fernandez > Anne Harris

There are no duplicate names, so all records are in descending alphabetical *NAME* order.

Save your MySQL query code below

Question 7: Examination Data
There is a database with exam scores. Write a query to print the names of the students who scored an even number of marks. The names should be listed in uppercase, alphabetically ascending. The result should be in the following format: NAME MARKS

Schema

exam		
Name	**Type**	**Description**
NAME	String	This is the student's name. Is the primary key.
MARKS	Integer	These are the marks obtained.

Sample Input

exam	
NAME	**MARKS**
Julia	10
Samantha	6
Jack	15

Sample Output

```
JULIA 10
SAMANTHA 15
```

Explanation

Julia and Samantha have an even number of marks, so they are included in the query.

Save your MySQL query code below

Question 8: Counting Hackos

A coding platform maintains all the participating hackers' data in the *HACKER* table. Write a query to print the names of all the hackers who have earned more than *100* hackos in less than *10* months. Print the output in the ascending order of their *ID*.

Schema

HACKOS		
Name	**Type**	**Description**
ID	Integer	A hacker ID in the inclusive range *[1, 1000]*. This is the primary key.
NAME	String	A hacker name. This field contains between *1* and *100* characters (inclusive).
MONTHS	Integer	The total number of months the hacker has been programming.
HACKOS	Integer	The total number of points the hacker gained per month.

The names should be printed in the ascending order of their *ID*.

Output Format

Each row of results must contain the name of a hacker who has earned more than *100* hackos in less than *10* months, in the following format:

```
HACKER.NAME
```

Sample Input

HACKOS			
ID	**NAME**	**MOTHS**	**HACKOS**
1	Frances White	5	20
2	Carolyn Bradley	2	10
3	Annie Fernandez	10	5
4	Ruth Hanson	5	15
5	Paula Fuller	6	15
6	Bonnie Johnston	8	12
7	Ruth Gutierrez	7	10
8	Ernest Thomas	4	30
9	Joe Garza	5	25
10	Anne Harris	7	15

Sample Output

```
Ernest Thomas
Joe Garza
Anne Harris
```

Explanation

The respective earnings of hackers since they started coding is described below:

- *Frances White* started coding *5* months ago and earned *20* hackos per month. So the total earned hackos are *5 x 20 = 100.*
- *Carolyn Bradley* started coding *2* months ago and earned *10* hackos per month. So the total earned hackos are *2 x 10 = 20.*

- *Annie Fernandez* started coding *10* months ago and earned *5* hackos per month. So the total earned hackos are *10 x 5 = 50.*
- *Ruth Hanson* started coding *5* months ago and earned *15* hackos per month. So the total earned hackos are *5 x 15 = 75.*
- *Paula Fuller* started coding *6* months ago and earned *15* hackos per month. So the total earned hackos are *6 x 15 = 90.*
- *Bonnie Johnston* started coding *8* months ago and earned *12* hackos per month. So the total earned hackos are *8 x 12 = 96.*
- *Ruth Gutierrez* started coding *7* months ago and earned *10* hackos per month. So the total earned hackos are *7 x 10 = 70.*
- *Ernest Thomas* started coding *4* months ago and earned *30* hackos per month. So the total earned hackos are *4 x 30 = 120.*
- *Joe Garza* started coding *5* months ago and earned *25* hackos per month. So the total earned hackos are *5 x 25 = 125.*
- *Anne Harris* started coding *7* months ago and earned *15* hackos per month. So the total earned hackos are *7 x 15 = 105.*

So, *Ernest Thomas, Joe Garza,* and *Anne Harris* have earned more than *100* hackos in less than *10* months.

Save your MySQL query code below

Question 9: Scoring System

The math scores of each student have been stored in the *STUDENT* table. Write a query to print the *ID* and the *NAME* of each of the three highest scoring students. Print the *NAME*s in descending order by *SCORE*, then ascending order by *ID* for matching *SCORE*s;

Schema

STUDENT		
Name	**Type**	**Description**
ID	Integer	A student ID in the inclusive range *[1, 1000]*. This field is the primary key.

NAME	String	A student's name. This field contains between *1* and *100* characters (inclusive).
SCORE	Float	The Math score of the student.

Output Format

The result should contain the *ID*s and the *NAME*s of the three highest scoring students. Print the records in descending order by *SCORE*, then ascending order by *ID* for matching *SCORE*s.

```
STUDENT.ID STUDENT.NAME
```

Sample Input

STUDENT		
ID	**NAME**	**SCORE**
1	Bob	50
2	John	65.5
3	Harry	45
4	Dick	85
5	Dev	25
6	Sid	98
7	Tom	90
8	Julia	70.5
9	Erica	81
10	Jerry	85

Sample Output

```
6 Sid
7 Tom
4 Dick
```

Explanation

The students are arranged in the descending order of their math scores, followed by the ascending order of their ids, as shown below:

Sid > Tom > Dick > Jerry > Julia > John > Bob > Harry > Dev

Dick's and Jerry's scores were the same, so they are shown in ID order.

Save your MySQL query code below

Question 10: The First Orders

A company maintains information about its orders in the *ORDERS* table. Write a query to print details of the earliest *five* orders (sorted by ORDER_DATE, ascending) that have not been delivered (i.e., *STATUS* is not *DELIVERED*). If there are more than five orders to choose from, select the ones with the lowest order ID. Sort the output in the increasing order of order *ID*. The output should contain *ID*, ORDER_DATE, STATUS, CUSTOMER_ID.

Schema

ORDERS	
column name	**column type**
ID	int
ORDER_DATE	date
STATUS	varchar(50)
CUSTOMER_ID	int

Sample Input

ORDERS			
ID	**ORDER_DATE**	**STATUS**	**CUSTOMER_ID**
10100	2003-01-06	PLACED	363
10101	2003-01-06	PLACED	128
10102	2003-01-06	IN TRANSIT	181

10103	2003-01-06	DELIVERED	121
10104	2003-01-07	IN TRANSIT	114
10106	2003-01-07	DELIVERED	278
10120	2003-01-07	PLACED	114
10122	2003-05-05	IN TRANSIT	350
10123	2003-05-05	DELIVERED	103

Sample Output

```
10100 2003-01-06 PLACE 363
10101 2003-01-06 PLACED 128
10102 2003-01-06 IN TRANSIT 181
10106 2003-01-07 IN TRANSIT 278
10120 2003-01-07 PLACED 114
```

Explanation

The orders with order numbers *10100*, *10101, 10102, 10106,* and *10120* are the earliest placed orders and also have order status, *not equal* to *DELIVERED*, so all their information in the increasing order of their order ID is printed.

Save your MySQL query code below

Question 11: Least Earning Locations

A ride hailing company has their DB structure in 3 major tables as described in *SCHEMA* below.

Write a query to fetch the city names along with earnings from each city. ‘Earnings’ are calculated as the sum of fares of all the rides taken in that city. The output should be structured as: cities.name earnings

The output is sorted ascending by earnings, then ascending by the city name.

Schema

There are 3 tables: CITIES, USERS, and RIDES.

CITIES		
Name	**Type**	**Description**
id	String	The assigned ID to the city presented as 32 character UUID.
name	String	The name of the city.

USERS		
Name	**Type**	**Description**
id	String	The assigned ID to the city presented as 32 character UUID.
city_id	String	The id of the city in which this user resides.
name	String	The name of the user.
email	String	The email of the user.

RIDES		
Name	**Type**	**Description**
id	String	The assigned ID to the ride presented as 32 character UUID.
user_id	String	The id of the user who took this ride.
distance	Integer	The traveling distance in this ride.
fare	Integer	The fare of this ride.

Sample Input

CITIES	
id	**name**
1	Cooktown
2	South Suzanne

USERS			
id	**city_id**	**name**	**email**
1	2	Robert Delgado	robertdelgado@hotmail.com
2	2	Thomas Williams	thomaswilliams@bradley.org
3	1	Michel Peterson	michelpeterson@hotmail.com
4	1	Bill Wheeler	billwheeler@gmail.com
5	1	David Lloyd	davidlloyd@gmail.com
6	1	Morgan Powers	morganpowers@hansen.biz

RIDES			
id	**user_id**	**distance**	**fare**
1	2	21	200
2	2	6	55
3	1	30	230
4	1	16	125
5	1	11	110

Output Format

```
South Suzanne 1050
Cooktown 1710
```

Explanation

1. In the city of *South Suzanne*, there are 2 users and the total fare of rides taken by those users is 1050.
2. In the city of *Cooktown*, there are 4 users and the total fare of rides taken by those users is 1710.

Save your MySQL query code below

Question 12: Student Rank

A university stores students' standardized test scores in a table named *STUDENT*. Student X placed 213th on the test.

Write a query to find Student's X's test score (i.e., the 213th highest *STUDENT.SCORE* in *STUDENT*.

Schema

STUDENT		
Name	**Type**	**Description**
ID	Integer	The student's unique ID. This is a primary key.
AGE	Integer	The student's age.
SCORE	Integer	The student's standardized test score.

Sample Input

STUDENT		
ID	**AGE**	**SCORE**
1	19	91
2	20	90
3	20	87
4	21	72
5	19	98
6	20	50

Explanation

This table's scores (from highest to lowest) are {*98, 91, 90, 87, 72, 50*}. As an example, the fourth-highest score is *87*. For the real query, find the 213th highest score.

Save your MySQL query code below

Question 13: Student Grades

Write a query to print the *ID* and *StudentGrade* for each record in the *STUDENT* table. Sort the output by student *ID*, ascending and use the following format.

```
Student STUDENT.ID has grade: StudentGrade
```

- If Score < 20, StudentGrade = F
- If 20 <= Score < 40, StudentGrade = D
- If 40 <= Score < 60, StudentGrade = C
- If 60 <= Score < 80, StudentGrade = B
- If Score >= 80, StudentGrade = A

Schema

STUDENTS		
Name	**Type**	**Description**
ID	Integer	primary key
SCORE	Integer	

Sample Input

STUDENT	
ID	**SCORE**
1	20
2	50
3	50
4	68
5	95

Sample Output

```
Student 1 has grade: D
Student 2 has grade: C
Student 3 has grade: C
Student 4 has grade: B
Student 5 has grade: A
```

Explanation

- The student with ID 1 has a SCORE of 20 and the grade D.
- 50 translates to C
- 68 translates to B
- 95 translates to A

Save your MySQL query code below

Basic Join 14 - 17 Questions

Question 14: Student's Major

A university maintains data on students and their majors in three tables: STUDENTS, MAJORS, and REGISTERS. The university needs a list of the STUDENT_NAME and MAJOR_NAME of the first 20 students. Do not sort the list.

Schema

STUDENTS		
Name	**Type**	**Description**
STUDENT_ID	Integer	The ID of a student. This is the primary key.
STUDENT_NAME	String	The name of the student.
STUDENT_AGE	Integer	The age of the student.

MAJORS		
Name	**Type**	**Description**
MAJOR_ID	Integer	The ID of a major. This is the primary key.
MAJOR_NAME	String	The name of a major.

REGISTER		
Name	**Type**	**Description**
STUDENT_ID	Integer	The ID of a student. This is a foreign key.

MAJOR_ID	Integer	The ID of a major. This is a foreign key.

Sample Input

STUDENTS		
STUDENT_ID	**STUDENT_NAME**	**STUDENT_AGE**
1	John	20
2	Masie	21
3	Harry	21

MAJORS	
MAJOR_ID	**MAJOR_NAME**
1000	Computer Science
2000	Biology
3000	Physics

REGISTER	
STUDENT_ID	**MAJOR_ID**
2	1000
3	3000
1	1000

Sample Output

```
John Biology
Masie Computer Science
Harry Physics
```

Save your MySQL query code below

Question 15: Trip Query

A travel and tour company has 2 tables that relate to customers: *FAMILIES* and *COUNTRIES*. Each tour offers a discount if a minimum number of people book at the same time.

Write a query to print the maximum number of discounted tours any 1 family in the *FAMILIES* table can choose from.

Schema

FAMILIES		
Name	**Type**	**Description**
ID	String	Unique ID of the family.
NAME	String	Name of the primary contact.
FAMILY_SIZE	Integer	Size of the family.

COUNTRIES		
Name	**Type**	**Description**
ID	String	Unique ID of the country.
NAME	String	Name of the country.
MIN_SIZE	Integer	Minimum size group to get a discount.

Sample Input

FAMILIES		
ID	**NAME**	**FAMILY_SIZE**
c00dac11bde74750b4d207b9c182a85f	Alex Thomas	9
eb6f2d3426694667ae3e79d6274114a4	Chris Gray	2

COUNTRIES		
ID	**NAME**	**FAMILY_SIZE**
jdjf89dgf78dfg8fdmdmdf9939393000v0	Bolivia	2
skkkkkk2k2373747599ssf0mwi23iudjd0	Cook Islands	4
cncnausujhaiauu7782890390dd003jf02	Brazil	4

Sample Output

```
3
```

Explanation

The Thomas family can choose from any of the 3 tours and qualify for the discount. The Gray family only qualifies for 1.

Save your MySQL query code below

Question 16: List the Course Names

Write a query to return a list of professor names and their associated courses. The results can be in any order but must not contain duplicate rows.

Schema

PROFESSOR		
Name	**Type**	**Description**
ID	Integer	Unique id, primary key
NAME	String	
DEPARTMENT_ID	Integer	This is a foreign key to DEPARTMENT.ID
SALARY	Integer	

DEPARTMENT		
Name	**Type**	**Description**
ID	Integer	Unique id, primary key

COURSE		
Name	**Type**	**Description**
ID	Integer	Unique id, primary key
NAME	String	
DEPARTMENT_ID	Integer	This is a foreign key to DEPARTMENT.ID
CREDITS	Integer	

SCHEDULE		
Name	**Type**	**Description**
PROFESSOR_ID	Integer	Foreign key to PROFESSOR.ID
COURSE_ID	Integer	Foreign key to COURSE.ID
SEMESTER	Integer	
YEAR	Integer	

Sample Input

PROFESSOR			
ID	**NAME**	**DEPARTMENT_ID**	**SALARY**
1	Alex Burton	5	7340
8	Jordan Diaz	1	17221
9	Drew Hicks	5	16613
2	Tyler Mathews	2	14521
10	Blake Foster	4	28526
3	Spencer Peters	1	10487
4	Ellis Marshall	3	6353
7	Morgan Lee	2	25796
5	Riley Peterson	1	35678
6	Peyton Fields	5	26648

DEPARTMENT	
ID	**NAME**
3	Biological Sciences
5	Technology
6	Humanities & Social Sciences

2	Clinical Medicine
4	Arts and Humanities
1	Physical Sciences

COURSE			
ID	**NAME**	**DEPARTMENT_ID**	**CREDITS**
9	Clinical Biochemistry	2	3
4	Astronomy	1	6
10	Clinical Neurosciences	2	5
1	Pure Mathematics and Mathematical Statistics	1	3
6	Geography	1	7
8	Chemistry	1	1
5	Physics	1	8
3	Earth Science	1	7
7	Materials Science and Metallurgy	1	5
2	Applied Mathematics and Theoretical Physics	1	5

SCHEDULE			
PROFESSOR_ID	**COURSE_ID**	**SEMESTER**	**YEAR**
5	3	6	2012
7	3	1	2013
5	7	6	2010
2	10	2	2004
5	1	1	2011
2	9	4	2005
7	10	6	2009

5	9	1	2014
9	9	5	2011

Sample Output

```
Tyler Mathews Clinical Biochemistry
Tyler Mathews Clinical Neuroscience
Drew Hicks Clinical Biochemistry
Morgan Lee Clinical Biochemistry
Morgan Lee Clinical Neuroscience
Morgan Lee Earth Science
Riley Peterson Earth Science
Riley Peterson Geography
Riley Peterson Materials Science and Metallurgy
Riley Peterson Pure Mathematics and Mathematical Statistics
```

Save your MySQL query code below

```
```

Question 17: Scheduling Errors

Write a query to return a list of professor names and their associated courses for all courses outside of their departments. There should be no duplicate rows, but they can be in any order.

The output should contain two columns: *professor.name, course.name.*

Schema

PROFESSOR		
Name	**Type**	**Description**
ID	Integer	Unique id, primary key
NAME	String	
DEPARTMENT_ID	Integer	Foreign key, department.id
SALARY	Integer	

DEPARTMENT

Name	Type	Description
ID	Integer	Unique id, primary key
NAME	String	

COURSE		
Name	**Type**	**Description**
ID	Integer	Unique id, primary key
NAME	String	
DEPARTMENT_ID	Integer	Foreign key, department.id
CREDITS	Integer	

SCHEDULE		
Name	**Type**	**Description**
PROFESSOR_ID	Integer	Foreign key, professor.id
COUSRE_ID	Integer	Foreign key, cousre_id
SEMESTER	Integer	
YEAR	Integer	

Sample Input

PROFESSOR			
ID	**NAME**	**DEPARTMENT_ID**	**SALARY**
1	Alex Daniels	4	7169
2	Drew Knight	1	9793
3	Jordan Myers	4	25194
4	Tyler Rodriguz	3	9686
5	Blake Gome	2	30860
6	Spencer George	5	10487

7	Ellis Vasquez	4	6353
8	Morgan Flores	1	25796
9	Riley Gilbert	5	35678
10	Peyton Stevens	2	26648

DEPARTMENT	
ID	**NAME**
3	Biological Sciences
5	Technology
6	Humanities & Social Sciences
2	Clinical Medicine
4	Arts and Humanities
1	Physical Sciences

COURSE			
ID	**NAME**	**DEPARTMENT_ID**	**CREDITS**
9	Clinical Biochemistry	2	3
4	Astronomy	1	6
10	Clinical Neurosciences	2	5
1	Pure Mathematics and Mathematical Statistics	1	3
6	Geography	1	7
8	Chemistry	1	1
5	Physics	1	8
3	Earth Science	1	7
7	Materials Science and Metallurgy	1	5
2	Applied Mathematics and Theoretical Physics	1	5

SCHEDULE			
PROFESSOR_ID	**COURSE_ID**	**SEMESTER**	**YEAR**
4	4	3	2003
3	3	1	2011
1	7	5	2011
7	7	1	2010
4	6	1	2001
9	3	1	2012
10	2	4	2009
1	1	3	2014
1	7	5	2008
1	7	5	2007

Sample Output

```
Tyler Rodriguez Astronomy
Jordan Myers Earth Science
Alex Daniels Materials Science and Metallurgy
Ellis Vasquez Materials Science and Metallurgy
Tyler Rodriguez Geography
Riley Gilbert Earth Sciences
Peyton Stevens Applied Mathematics and Theoretical Physics
Alex Daniels Pure Mathematics and Mathematical Statistics
Alex Daniels Applied Mathematics and Theoretical Physics
Alex Daniels Materials Science and Metallurgy
```

Explanation

1. Professor *Tyler Rodriguez's department_id* is *3,* but the *Astronomy* course's *department_id* is *1*.
2. Professor *Jordan Myers's department_id* is *4,* but the *Earth Science* course's *department_id* is *1*.

Save your MySQL query code below

Basic Aggregation 18 - 23 Questions

Question 18: Aggregate Marks

There is a database containing the marks of some students in various subjects. The data may contain any number of subjects for a student.

Retrieve the records of students who have a sum of marks greater than or equal to 500. The result should be in the following format: *STUDENT_ID SUM_OF_MARKS* sorted descending by *STUDENT_ID*.

Schema

marks		
Name	**Type**	**Description**
STUDENT_ID	Integer	This is the student's unique ID.
MARKS	Integer	These are the marks obtained.

Sample Input

marks	
STUDENT_ID	**MARKS**
1	450
2	200
3	260
2	300
3	250

Sample Output

```
3 510
2 500
```

Explanation

- *3* has a sum of 510, so it's printed in the query results.

- *1* has a sum of 450, so it does not get printed in the query results.
- *2* has a sum of 500, so it's printed in the query results.

Save your MySQL query code below

Question 19: Active Backlogs
Return a list of all students with at least one occurrence of a backlog item.

The result should be in the following format: *student.name*

Schema

student		
Name	**Type**	**Description**
ID	Integer	Unique id, the primary key.
NAME	String	

backlog		
Name	**Type**	**Description**
STUDENT_ID	Integer	Foreign key, student.id
SUBJECT_ID	String	

Sample Input

student	
ID	**NAME**
1	Chris
2	Sam
3	Alex

backlog

STUDENT_ID	SUBJECT_ID
1	abc123
3	def456

Sample Output

```
Alex
Chris
```

Explanation
Each student listed has at least one record in the backlog table.

Save your MySQL query code below

```
```

Question 20: Clumsy Administrator
A company maintains the records of all employees. The company pays the database administrator too little so the work has been quite clumsy. The administrator carelessly inserted the records of many employees into the employee records table multiple times. An employee's record is considered duplicate only if all columns (fields) of the employee's record are duplicated.

Write a query to find the names of employees whose records occur more than once in the table.

Schema

EMPLOYEE		
Name	**Type**	**Description**
NAME	String	The name of the employee.
PHONE	String	The telephone number of the employee.
AGE	Integer	The age of the employee.

Sample Input

EMPLOYEE

NAME	PHONE	AGE
Sam	1000040000	30
Alex	1000020000	60
Alex	1000020012	65
Sam	1000040000	30
Chris	1000012000	34
Chris	1000012000	34

Explanation
Here, the exact records of Sam and Chris occur more than once in the table. Hence, San and Chris are the employees in the resultant output.

Save your MySQL query code below

Question 21: Value of Properties Owned
There are two tables in a database of real estate owners. One has ownership information and the other has price information, in millions. An owner may own multiple houses, but a house will have only one owner.

Write a query to print the IDs of the owners who have at least 100 million worth houses and own more than 1 house. The order of output does not matter. The result should be in the format: *BUYER_ID TOTAL_WORTH*

Schema

house		
Name	**Type**	**Description**
BUYER_ID	Integer	Unique buyer ID.
HOUSE_ID	String	Unique house ID.

price

Name	Type	Description
HOUSE_ID	String	Unique house ID. The primary key
PRICE	Integer	The price of the house.

Sample Input

house	
BUYER_ID	**HOUSE_ID**
1	abc123
2	def456
3	abc456
1	def123
2	def789

price	
HOUSE_ID	**PRICE**
abc123	60
def456	20
abc456	120
def123	40
def789	70

Sample Output

```
1 100
```

Save your MySQL query code below

Question 22: The Beautiful Collection

A shopkeeper maintains the count of the different colored balls (Red, green, and blue) in the *COLLECTION* table. Each row of the table represents one of the following types:

- GOOD: *If the count of the red, green, and blue balls are equal.*
- BAD: *If the count of any two colored balls are equal, i.e., only one of the following conditions is true:*
    - Red balls count is equal to green balls.
    - Red balls count is equal to blue balls.
    - Green balls count is equal to blue balls.
- WORSE: *If all the colored balls have different counts.*

Write a query to print the type which is represented by each row of the table. Note that the output is case-sensitive, so make sure to output only GOOD, BAD or WORSE.

Schema

COLLECTIONS		
Name	**Type**	**Description**
RED	Integer	This describes the count of red balls. The count can be between *30* units and *100* units (inclusive).
GREEN	Integer	This describes the count of green balls. The count can be between *30* units and *100* units (inclusive).
BLUE	Integer	This describes the count of blue balls. The count can be between *30* units and *100* units (inclusive).

Output Format

Each row of the result should contain one of the types which are described above, in the following format. Note that the output is case-sensitive, so make sure to output only GOOD, BAD, WORSE.

```
TYPE_OF_COLLECTION
```

Sample Input

COLLECTION		
RED	**GREEN**	**BLUE**
65	65	87

50	50	50
30	50	100
40	50	90
92	50	50

Sample Output

```
BAD
GOOD
WORSE
WORSE
BAD
```

Explanation

The type of collection represented by each row is explained below:

- The count of the red balls is equal to the count of the green balls. But the count of blue balls is not equal to the count of red or green balls. So, the type is BAD.
- The count of the red, green, and blue balls are equal. So, the type is WORSE.
- The count of the red, green, and blue balls are different from each other. So, the type is WORSE.
- The count of the red, green, and blue balls are different from each other. So, the type is WORSE.
- The count of the green balls is equal to the count of the blue balls. But the count of red balls is not equal to the count of green or blue balls. So, the type is BAD.

Save your MySQL query code below

Question 23: City Revenue

A number of cities each have a number of agencies that estimate revenues. The average revenue of a city is defined as the average of all agencies' estimates of revenue for a city.

Write a query to print the **floor of the average revenue** of each city. The order of output does not matter. The result should be in the following format: *CITY_NAME AVERAGE_REVENUE*.

Schema

CITIES		
Name	**Type**	**Description**
CITY_CODE	Integer	The city's PINCODE. This is a primary key.
CITY_NAME	String	The city's name.

REVENUE		
Name	**Type**	**Description**
CITY_CODE	Integer	The city's PINCODE.
REVENUE	Integer	The estimated revenue.

Sample Input

CITIES	
CITY_CODE	**CITY_NAME**
1	New York
2	London
3	Paris

REVENUE	
CITY_CODE	**REVENUE**
1	10
2	5
2	10
3	15

Sample Output

```
New York 10
London 7
Paris 15
```

Explanation

- *New York* has only one estimate, so it is the average.
- *London* has two estimates. The average of those is *floor((5+9)/2) = 7.*
- *Paris* has only one estimate, so it is the average.

Save your MySQL query code below