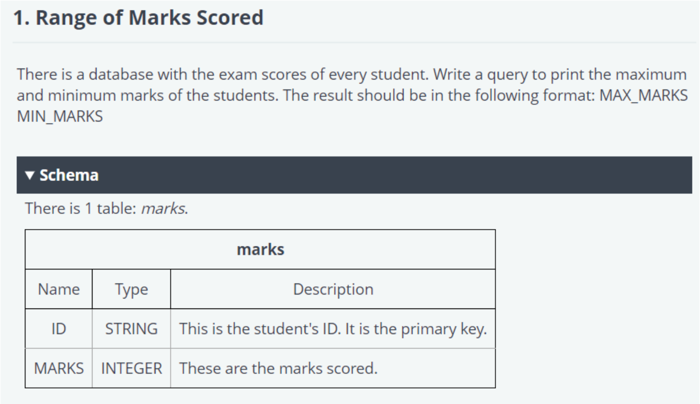


1. Range of Marks Scored

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

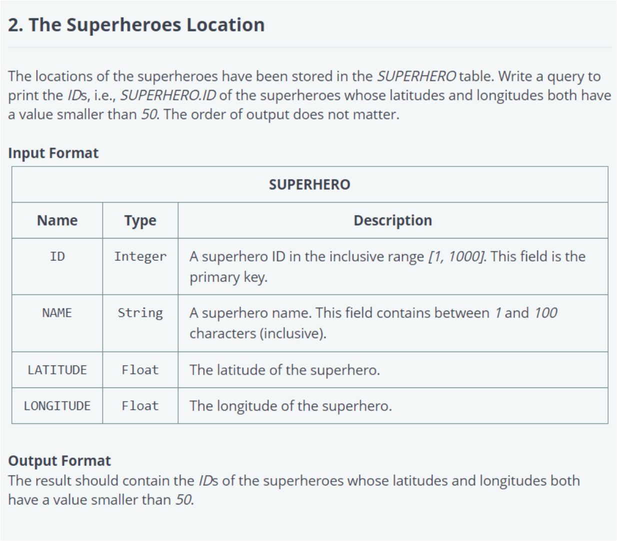
▼ **Schema**

There is 1 table: *marks*.

marks		
Name	Type	Description
ID	STRING	This is the student's ID. It is the primary key.
MARKS	INTEGER	These are the marks scored.

SELECT MAX(MARKS) AS MAX_MARKS , MIN(MARKS) AS MIN_MARKS

FROM marks;



2. The Superheroes Location

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

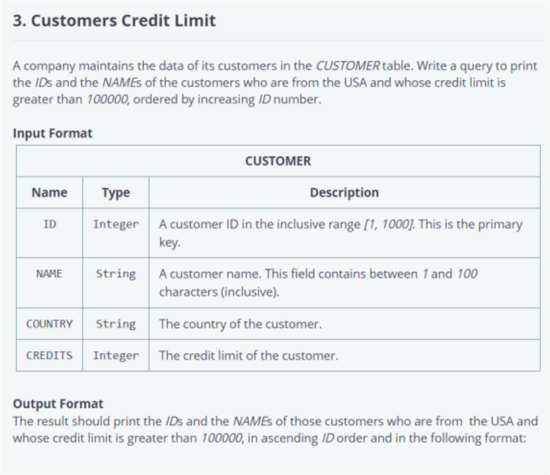
Input Format

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 *ID*s of the superheroes whose latitudes and longitudes both have a value smaller than *50*.

SELECT SUPERHERO.ID FROM SUPERHERO WHERE SUPERHERO.LATITUDE <50 AND SUPERHERO.LONGITUDE<50;



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.

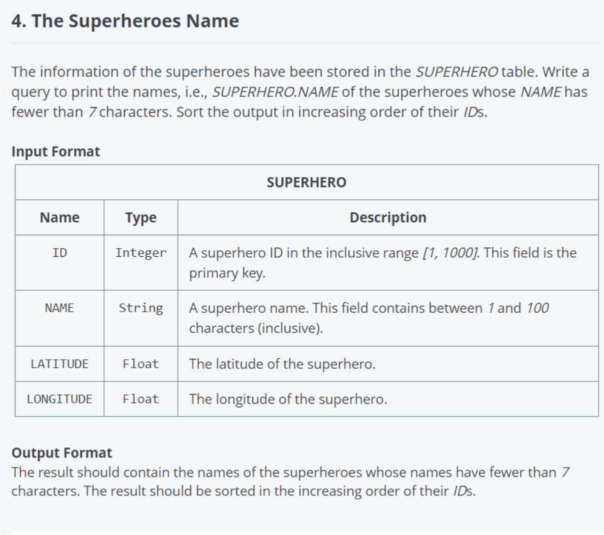
Input Format

CUSTOMER		
Name	**Type**	**Description**
`ID`	`Integer`	A customer ID in the inclusive range *[1, 1000]*. This is the primary key.
`NAME`	`String`	A customer 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 *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 and in the following format:

SELECT ID, NAME FROM CUSTOMER WHERE COUNTRY='USA'

AND CREDITS>100000 ORDER BY 1 DESC;



4. The Superheroes Name

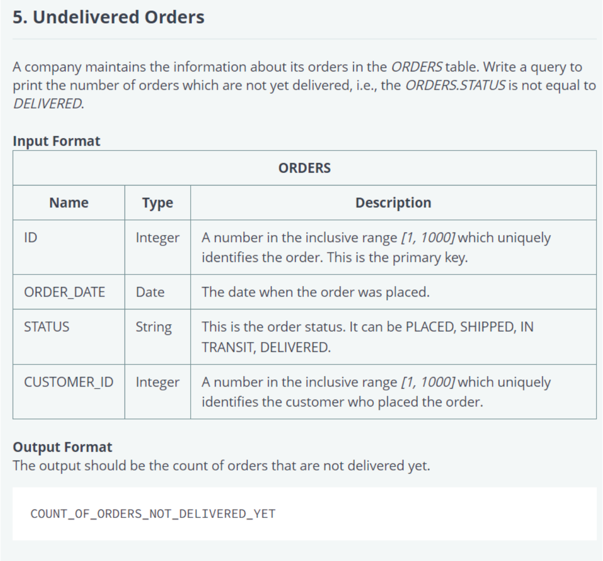
The information of the superheroes have been stored in the *SUPERHERO* table. Write a query to print the names, i.e., *SUPERHERO.NAME* of the superheroes whose *NAME* has fewer than *7* characters. Sort the output in increasing order of their *ID*s.

Input Format

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.

SELECT NAME FROM SUPERHERO WHERE LENGTH(NAME)<7 ORDER BY ID DESC;



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 delivered, i.e., the *ORDERS.STATUS* is not equal to *DELIVERED*.

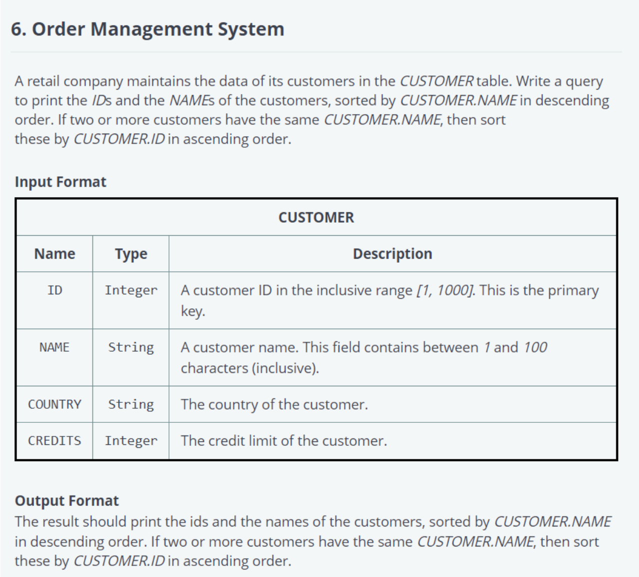
Input Format

ORDERS		
Name	**Type**	**Description**
ID	Integer	A number in the inclusive range *[1, 1000]* which uniquely identifies the order. This is the primary key.
ORDER_DATE	Date	The date when the order was placed.
STATUS	String	This is the order status. It can be PLACED, 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_ORDERS_NOT_DELIVERED_YET
```

SELECT COUNT(ID) FROM ORDERS WHERE STATUS <>'DELIVERED';



6. Order Management System

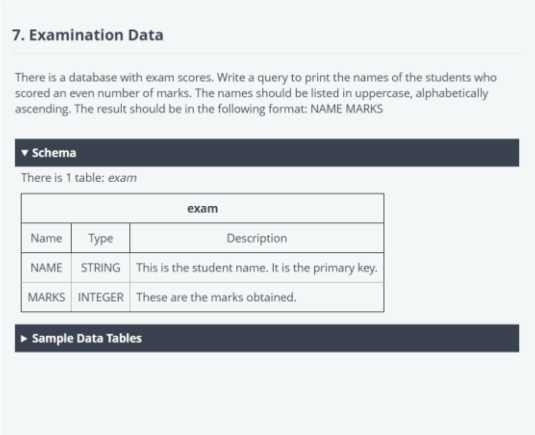
A retail company maintains the data of its customers in the *CUSTOMER* table. Write a query to print the *ID*s and the *NAME*s of the customers, sorted by *CUSTOMER.NAME* in descending order. If two or more customers have the same *CUSTOMER.NAME*, then sort these by *CUSTOMER.ID* in ascending order.

Input Format

CUSTOMER		
Name	**Type**	**Description**
ID	Integer	A customer ID in the inclusive range *[1, 1000]*. This is the primary key.
NAME	String	A customer 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.

SELECT ID, NAME FROM CUSTOMER ORDER BY NAME DESC,ID;



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

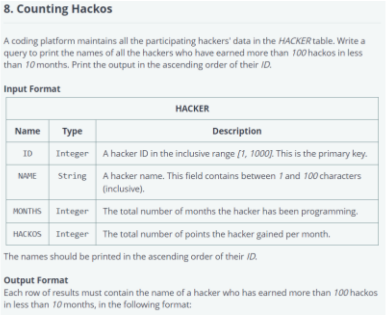
▼ Schema

There is 1 table: *exam*

exam		
Name	Type	Description
NAME	STRING	This is the student name. It is the primary key.
MARKS	INTEGER	These are the marks obtained.

▶ Sample Data Tables

```
SELECT UPPER(NAME) AS NAME,MARKS FROM exam WHERE MARKS%2=0 ORDER BY 1 ASC;
```



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*.

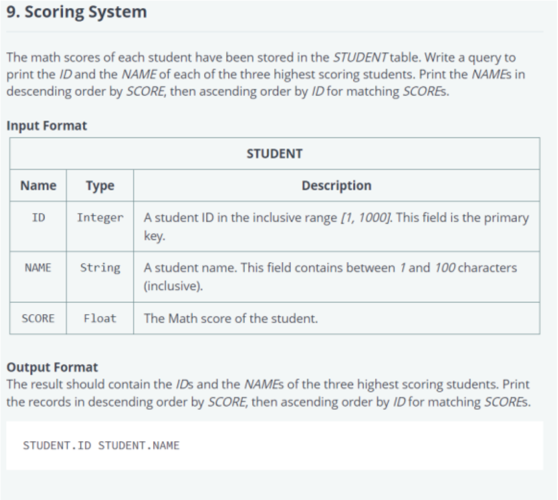
Input Format

HACKER		
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:

```
SELECT NAME FROM HACKER WHERE HACKOS*MONTHS>100 AND MONTHS<10;
```



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 *NAMEs* in descending order by *SCORE*, then ascending order by *ID* for matching *SCOREs*.

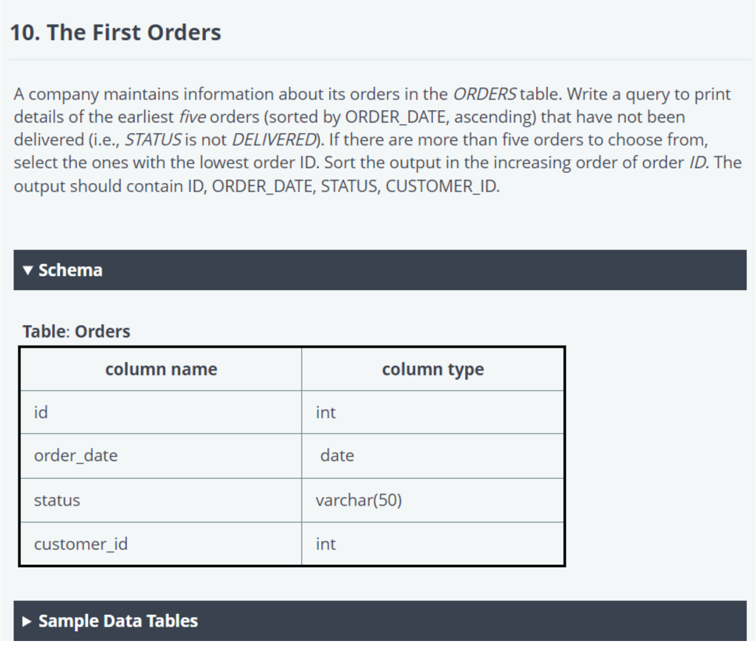
Input Format

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 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 *IDs* and the *NAMEs* of the three highest scoring students. Print the records in descending order by *SCORE*, then ascending order by *ID* for matching *SCOREs*.

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

```
SELECT ID ,NAME FROM STUDENT ORDER BY SCORE DESC,ID ASC LIMIT 3;
```



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.

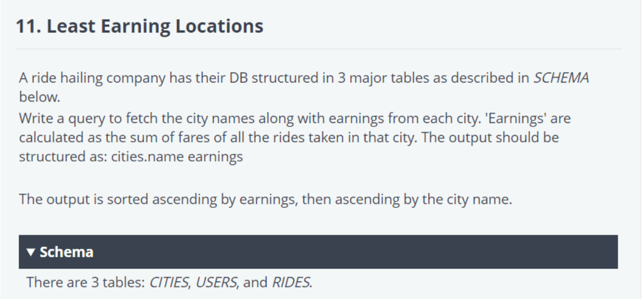
▼ Schema

Table: Orders

column name	column type
id	int
order_date	date
status	varchar(50)
customer_id	int

▶ Sample Data Tables

```
SELECT ID, ORDER_DATE ,STATUS, CUSTOMER_ID FROM ORDERS
WHERE STATUS<>'DELIVERED' ORDER BY 2 ASC,ID LIMIT 5;
```



11. Least Earning Locations

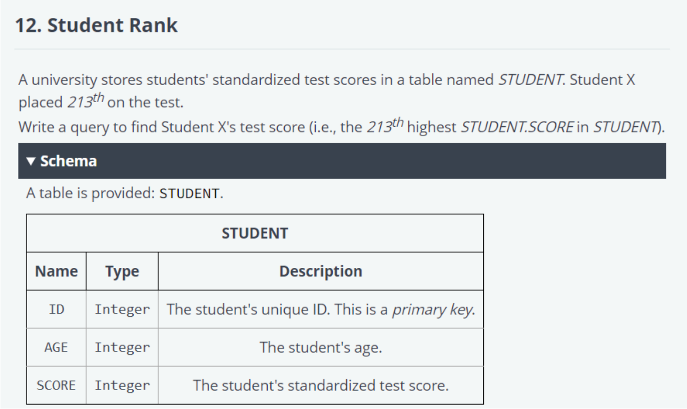
A ride hailing company has their DB structured in 3 major tables as described in *SCHEMA* below.
Write a query to fetch the city names along with earnings from each city. 'Earnings' are calculated as the sum of fares of all the rides taken in that city. The output should be structured as: cities.name earnings

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

▼ Schema

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

```
SELECT C.name,SUM(R.fare) AS 'Earnings' FROM CITIES C
JOIN USERS U
ON C.id = U.city_id
JOIN RIDES R
ON U.id=R.user_id
GROUP BY C.name
ORDER BY 2 ASC,1 ASC;
```



12. Student Rank

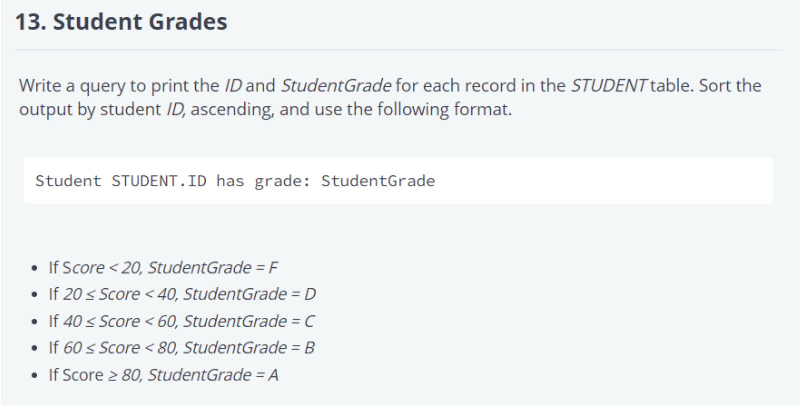
A university stores students' standardized test scores in a table named *STUDENT*. Student X placed *213*th on the test.
Write a query to find Student X's test score (i.e., the *213*th highest *STUDENT.SCORE* in *STUDENT*).

▼ Schema

A table is provided: STUDENT.

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.

```
/*
Enter your query here.
Please append a semicolon ";" at the end of the query
*/

SELECT SCORE  FROM STUDENT ORDER BY SCORE DESC LIMIT 212,1;
```



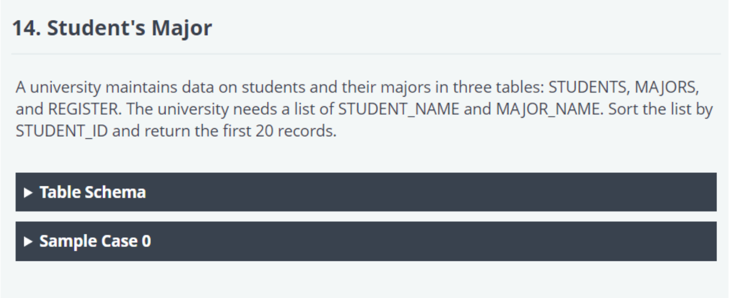
13. Student Grades

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*

```
/*
Enter your query here.
*/
SELECT
CONCAT('Student ',STUDENT.ID,' has grade:  ',
 CASE
WHEN SCORE>=20 AND SCORE<40 THEN 'D'
WHEN SCORE >=40 AND SCORE<60 THEN 'C'
WHEN SCORE>=60 AND SCORE<80 THEN 'B'
WHEN SCORE >=80 THEN 'A'
ELSE 'F'
END)AS StudentGrade
FROM STUDENT
ORDER BY STUDENT.ID ;
```

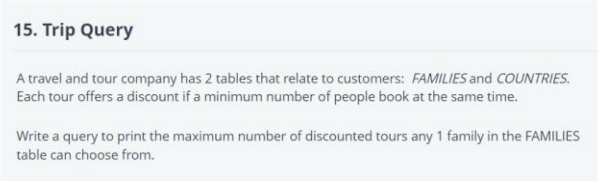


14. Student's Major

A university maintains data on students and their majors in three tables: STUDENTS, MAJORS, and REGISTER. The university needs a list of STUDENT_NAME and MAJOR_NAME. Sort the list by STUDENT_ID and return the first 20 records.

```
/*
Enter your query here.
Please append a semicolon ";" at the end of the query
*/
SELECT S.STUDENT_NAME,M.MAJOR_NAME FROM STUDENTS S
JOIN REGISTER R
ON S.STUDENT_ID=R.STUDENT_ID
JOIN MAJORS M
ON M.MAJOR_ID =R.MAJOR_ID
```

```
ORDER BY S.STUDENT_ID
LIMIT 20;
```

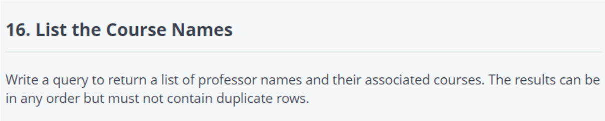


15. Trip Query

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.

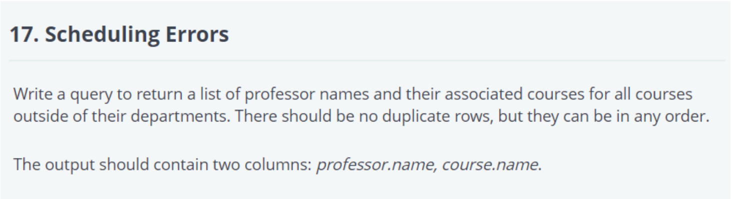
```
/*
Enter your query below.
Please append a semicolon ";" at the end of the query
*/
SELECT COUNT( CASE
WHEN F.FAMILY_SIZE >=C.MIN_SIZE THEN 'DISCOUNT'
ELSE NULL
END )AS 'TOURS'
 FROM FAMILIES  F
JOIN COUNTRIES C

where F.FAMILY_SIZE >=C.MIN_SIZE
GROUP BY F.NAME
ORDER BY 1 DESC
LIMIT 1
;
```



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.

```
SELECT DISTINCT
P.NAME,C.NAME
FROM PROFESSOR P
INNER JOIN SCHEDULE S
ON P.ID=S.PROFESSOR_ID
INNER JOIN COURSE C
ON S.COURSE_ID=C.ID
inner JOIN DEPARTMENT D
ON D.ID=C.DEPARTMENT_ID
order by 1 DESC;
```

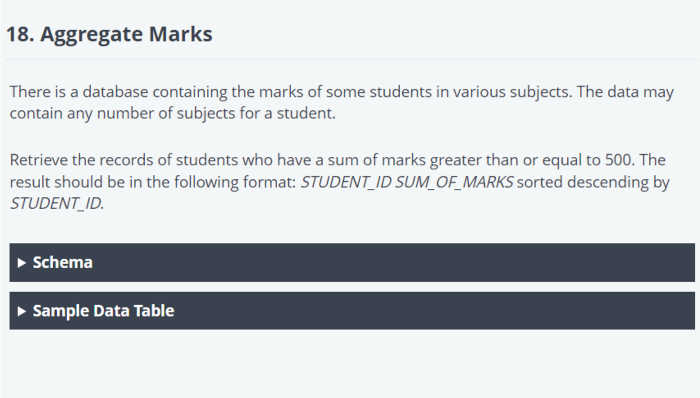


17. Scheduling Errors

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.*

```
SELECT DISTINCT
P.NAME,C.NAME
FROM PROFESSOR P
INNER JOIN SCHEDULE S
ON P.ID=S.PROFESSOR_ID
INNER JOIN COURSE C
ON S.COURSE_ID=C.ID
inner JOIN DEPARTMENT D
ON D.ID=C.DEPARTMENT_ID
WHERE P.DEPARTMENT_ID<>C.DEPARTMENT_ID;
```

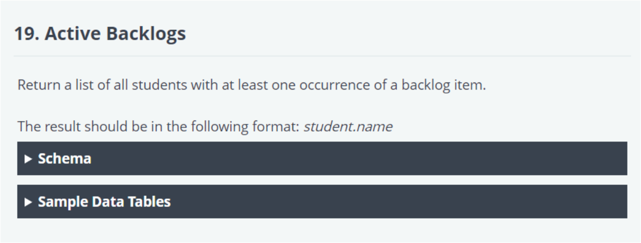


18. Aggregate Marks

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.*

```
/*
Enter your query below.
Please append a semicolon ";" at the end of the query
*/

SELECT STUDENT_ID,sum(MARKS) FROM marks
GROUP BY STUDENT_ID
HAVING SUM(MARKS)>=500
ORDER BY STUDENT_ID DESC;
```

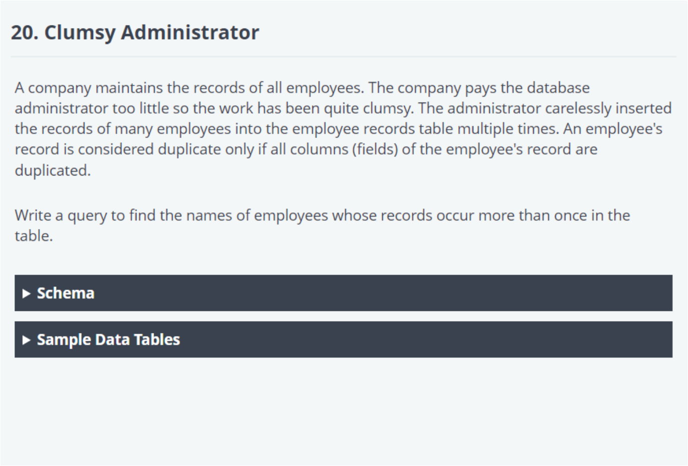


19. Active Backlogs

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*

```
SELECT DISTINCT student.NAME
FROM student
join backlog
on student.ID=backlog.STUDENT_ID
WHERE backlog.STUDENT_ID>=1
ORDER BY 1 ASC;
```

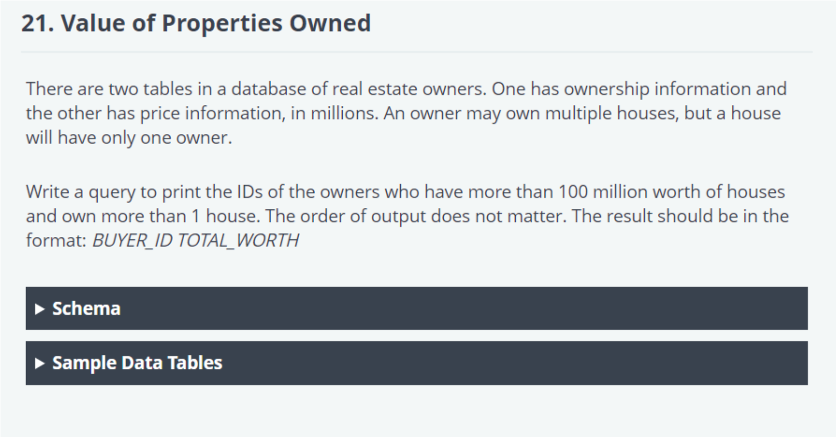


20. Clumsy Administrator

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.

```
SELECT DISTINCT NAME FROM EMPLOYEE
group by NAME,PHONE,AGE
HAVING COUNT(*)>1
ORDER BY NAME;
```

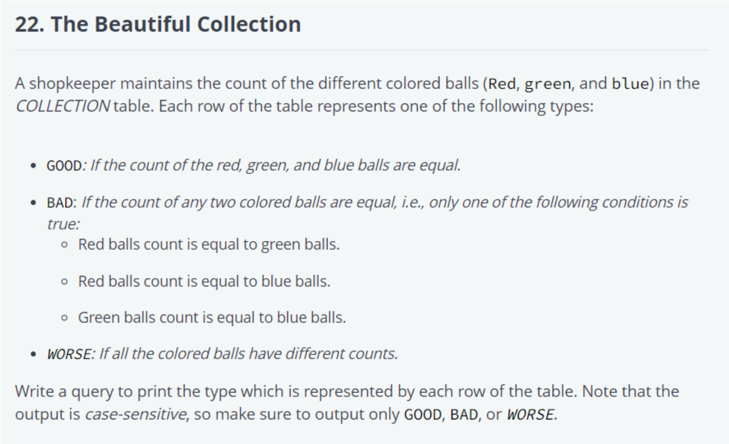


21. Value of Properties Owned

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 more than 100 million worth of 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*

```
SELECT H.BUYER_ID ,SUM(P.PRICE) AS TOTAL_WORTH
FROM house H
JOIN price P
ON H.HOUSE_ID=P.HOUSE_ID
GROUP BY H.BUYER_ID
HAVING SUM(P.PRICE)>100 AND COUNT(H.HOUSE_ID)>1;
```



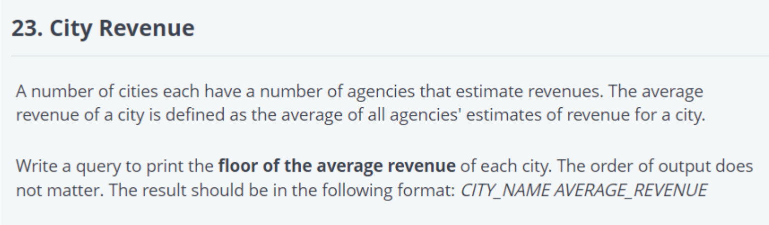
22. The Beautiful Collection

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`*.

```
SELECT
CASE
WHEN RED=BLUE AND BLUE=GREEN THEN 'GOOD'
WHEN RED=GREEN OR RED=BLUE OR GREEN=BLUE THEN 'BAD'
ELSE 'WORSE'
END AS 'TYPE_OF_COLLECTION'
FROM COLLECTION;
```



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*

```
SELECT C.CITY_NAME, FLOOR(
    SUM(R.REVENUE)/COUNT(R.CITY_CODE)) AS AVERAGE_REVENUE
    FROM CITIES C
    JOIN REVENUE R
    ON C.CITY_CODE=R.CITY_CODE
    GROUP BY C.CITY_NAME ;
```