**SQL Hackathon - Code Academy Berlin**

**Instructions**

* The document contains a set of questions in the domain of DBMS and MySQL.
* The **start time is 10 a.m**. and you have to **submit before 5:30 pm**
* In case you find some questions are not clear, please answer to the best of your understanding.
* Approach is much more valued than a correct answer.
* Submit the answers in the document itself and email it to your mentor ([shekhar@codeacademyberlin.com](mailto:shekhar@codeacademyberlin.com)) latest by the end time.
* As usual, you can take help from google ( however, **for conceptual questions, you are not allowed to look at google**).
* For any clarifications, please send a message to your mentor.
* All the questions generally have a schema associated with it, however, some questions might have more explanations with tables and data points (not necessarily though).
* You **DO NOT** have to create any table until and unless it is mentioned specifically.

###################################################################

###################################################################

1. What is the term Relational means in **RDBMS**.

**The term refers to the relations of data that is stored in different tables within a database. The distribution of data parts over several tables can save a lot of storage space.**

1. Can you explain Primary and Foreign Key and what is the importances of it?

**The Primary Key is the unique identifier of a record. The certain record can always be addressed with it.**

**The Foreign Key is a defined connection of one record to more data distributed over one or many other tables.**

1. What are the popular Data Types in MySQL?

**varchar, var, int, float, double, timestamp, datetime, boolean and more**

1. Can you explain the process of Database design process ( you can include what are the steps of database creation as well ).
2. **See and understand the incoming (streaming) or static data to be stored.**
3. **Define roughly the future tasks to fulfill with the data.**
4. **Analyze the data by it’s structure:   
   - What is important and needs to be kept?  
   - Which parts of the records are unique, which ones come as continuous duplicates? How are the different information parts connected to each other?**
5. **Divide the information in different tables and define their relations (including Primary and sufficient Foreign Keys in case of one- or many-to-many relations).**
6. **Define the single data types.**
7. **Set up the database and the tables with the designated data types.**
8. **Clean the data as far as possible.**
9. **Load the data.**
10. **Clean it (again/once more).**
11. **Remove duplicates if necessary.**
12. **Set Keys if they didn’t come with the data.**
13. **(Process or query the data.)**
14. Why is MySQL important in Data Science ? Do you prefer it over Pandas and if yes, in which scenarios ?

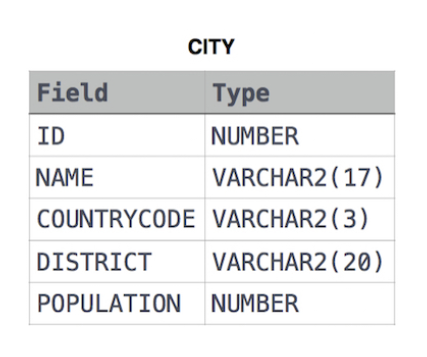
**MySQL can handle Big Data, Pandas unfortunately cannot.**

**Huge amounts of data can be stored and processed in clouds with SQL, also using APIs to other services as online visualization tools. SQL can also deal with data streams.**

1. When would you prefer Pandas over SQL.(give examples if possible)

**I personally prefer Pandas whenever I need to understand my data and serious data cleaning. To me Pandas is always helpful to find deviations in data quality or structure and in eliminating them.**

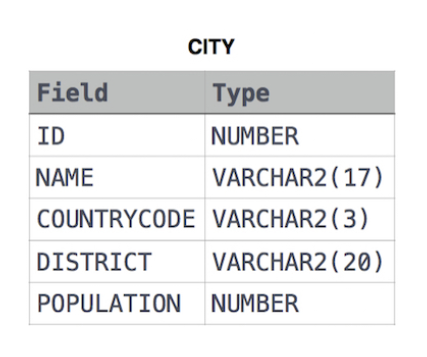
1. Query all columns for a city in **CITY** with the ID 1661.



**SELECT \* FROM CITY WHERE ID = 1661;**

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:



**SELECT \***

**FROM**

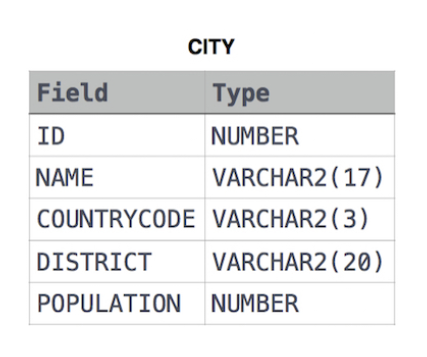
**CITY**

**WHERE**

**COUNTRYCODE = ‘USA‘ AND POPULATION > 100000;**

1. Query the **NAME** field for all American cities in the **CITY** table with populations larger than 120000. The CountryCode for America is ‘USA’.

The **CITY** table is described as above.



**SELECT**

**NAME**

**FROM**

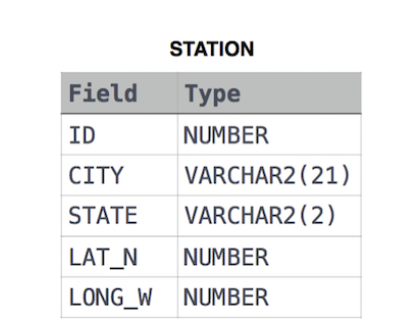
**CITY**

**WHERE**

**COUNTRYCODE = ‘USA‘ AND POPULATION > 120000;**

1. Query a list of **CITY** and **STATE** from the **STATION** table.

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



where **LAT\_N** is the northern latitude and **LONG\_W** is the western longitude.

**If it is supposed to be a distinct list:**

**SELECT**

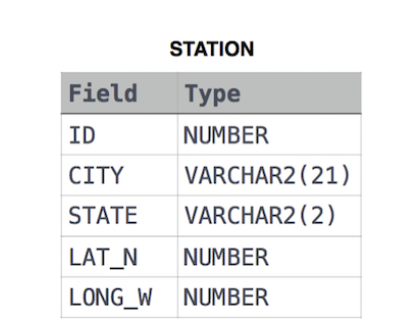
**DISTINCT(CITY, STATE)**

**FROM**

**STATION;**

1. Query a list of **CITY** names from **STATION** for cities that have an even **ID** number. Print the results in any order, but exclude duplicates from the answer.

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



where **LAT\_N** is the northern latitude and **LONG\_W** is the western longitude.

**SELECT**

**DISTINCT CITY**

**FROM**

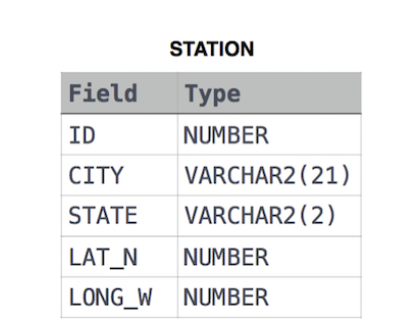
**STATION**

**WHERE**

**MOD(ID, 2) = 0;**

1. Find the difference between the total number of CITY entries in the table and the number of distinct CITY entries in the table.

The STATION table is described as follows:



where **LAT\_N** is the northern latitude and **LONG\_W** is the western longitude.

For example, if there are three records in the table with CITY values 'New York', 'New York',’Mumbai’, there are 2 different city names: 'New York' and ‘Mumbai’. The query returns 1, because:



**SELECT**

**COUNT(CITY) - COUNT(DISTINCT CITY)**

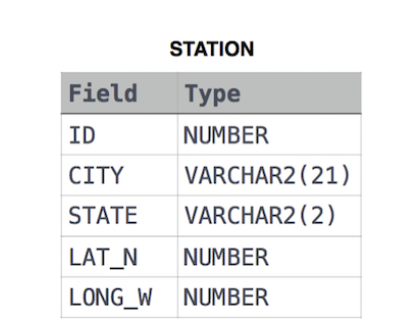
**FROM**

**STATION;**

1. Query the list of CITY names starting with vowels (i.e., a, e, i, o, or u) from STATION. Your result **cannot** contain duplicates.

**Input Format**

The STATION table is described as follows:



**SELECT**

**DISTINCT CITY**

**FROM**

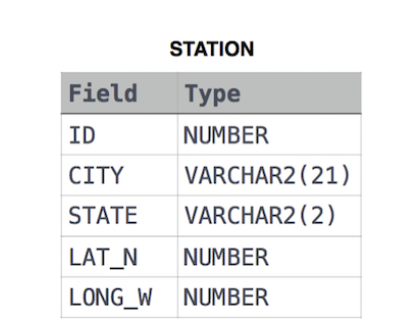
**STATION**

**WHERE**

**CITY RLIKE '^[AEIOU]‘;**

1. Query the two cities in STATION with the shortest and longest CITY names, as well as their respective lengths (i.e.: number of characters in the name). If there is more than one smallest or largest city, choose the one that comes first when ordered alphabetically.

The STATION table is described as follows:



**Sample Input**

For example, CITY has four entries: DEF, ABC, PQRS and WXY.

**Sample Output**

ABC 3

PQRS 4

For example, CITY has four entries: DEF, ABC, PQRS and WXY.

**Explanation**

When ordered alphabetically, the CITY names are listed as ABC, DEF, PQRS, and WXY, with lengths 3,3,4 and 3. The longest name is PQRS, but there are options for the shortest named city. Choose ABC, because it comes first alphabetically.

**Note**

**You can write two separate queries to get the desired output. It need not be a single query.**

**SELECT**

**MIN(CITY), MIN(LENGTH(CITY))**

**FROM**

**STATION**

**ORDER BY**

**CITY;**

**SELECT**

**MIN(CITY), MAX(LENGTH(CITY))**

**FROM**

**STATION**

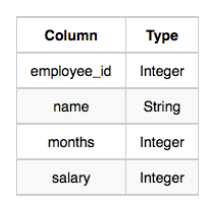
**ORDER BY**

**CITY;**

1. Write a query that prints a list of employee names (i.e.: the name attribute) for employees in Employee having a salary greater than $2000 per month who have been employees for less than 10 months. Sort your result by ascending employee\_id.

**Input Format**

The Employee table containing employee data for a company is described as follows:



where employee\_id is an employee's ID number, name is their name, months is the total number of months they've been working for the company, and salary is their monthly salary.

**Sample Input**

****

**Sample Output**

Angela

Michael

Todd

Joe

**Explanation**

Angela has been an employee for 1 month and earns 3443 per month.

Michael has been an employee for months and earns 2017 per month.

Todd has been an employee for months and earns 3396 per month.

Joe has been an employee for months and earns 3573 per month.

We order our output by ascending employee\_id.

**SELECT**

**name**

**FROM**

**Employee**

**WHERE**

**salary > 2000 AND months < 10**

**ORDER BY**

**employee\_id;**

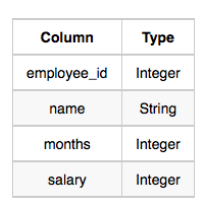
1. We define an employee's total earnings to be their monthly **salary \* months** worked, and the maximum total earnings to be the maximum total earnings for any employee in the Employee table.

Write a query to find the maximum total earnings for all employees as well as the total number of employees who have maximum total earnings. Then print these values as space-separated integers.

**Input Format**

The Employee table containing employee data for a company is described as

follows:



where employee\_id is an employee's ID number, name is their name, months

is the total number of months they've been working for the company, and

salary is their monthly salary.

**Sample Input**



**Sample Output**

69952 1

**Explanation**

The table and earnings data is depicted in the following diagram:



The maximum earnings value is 69952. The only employee with earnings

69952 is Kimberly, so we print the maximum earnings value (69952) and a

count of the number of employees who have earned 69952 (which is 1) as

two space-separated values.

**SELECT**

**CONCAT(MAX(months \* salary), ' ', (SELECT count(\*) AS resultcount**

**FROM (**

**SELECT MAX(rating \* review\_count)**

**FROM Employee) AS result))**

**FROM Employee;**

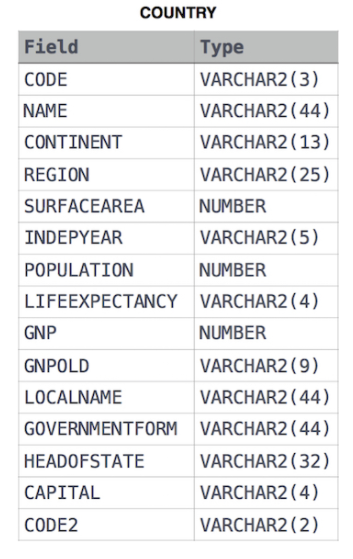
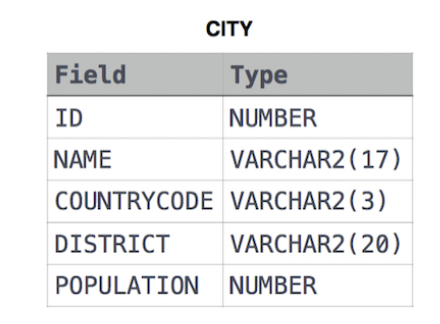
1. Given the **CITY** and **COUNTRY** tables, query the sum of the populations

of all cities where the CONTINENT is 'Asia'.

**Note:** CITY.CountryCode and COUNTRY.Code are matching key columns.

**Input Format**

The **CITY** and **COUNTRY** tables are described as follows:



**SELECT**

**SUM(POPULATION)**

**FROM**

**CITY**

**INNER JOIN**

**COUNTRY**

**ON**

**CITY.COUNTRYCODE = COUNTRY.CODE**

**WHERE**

**COUNTRY.CONTINENT LIKE 'Asia';**

OR

**WHERE**

**COUNTRY.CONTINENT = 'Asia';**

1. The Employee table holds all employees. Every employee has an Id, a salary, and there is also a column for the department Id.

+----+-------+--------+--------------+

| Id | Name | Salary | DepartmentId |

+----+-------+--------+--------------+

| 1 | Joe | 70000 | 1 |

| 2 | Jim | 90000 | 1 |

| 3 | Henry | 80000 | 2 |

| 4 | Sam | 60000 | 2 |

| 5 | Max | 90000 | 1 |

+----+-------+--------+--------------+

The Department table holds all departments of the company.

+----+----------+

| Id | Name |

+----+----------+

| 1 | IT |

| 2 | Sales |

+----+----------+

Write a SQL query to find employees who have the highest salary in each of the departments. For the above tables, your SQL query should return the following rows (order of rows does not matter).

+------------+----------+--------+

| Department | Employee | Salary |

+------------+----------+--------+

| IT | Max | 90000 |

| IT | Jim | 90000 |

| Sales | Henry | 80000 |

+------------+----------+--------+

**Explanation**:

Max and Jim both have the highest salary in the IT department and Henry has the highest salary in the Sales department.

**SELECT**

**Department.Name, Employee.Name, MAX(Employee.Salary)**

**FROM**

**Employee**

**INNER**

**JOIN Department**

**ON**

**Employee.DepartmentId = Department.Id**

**GROUP BY**

**Employee.DepartmentId;**

1. Create a database called **‘employee’**.

Then create and insert below values into the table **EmployeeWages**.

-- STEP 1: Create the table

CREATE TABLE EmployeeWages (

employee\_id INT PRIMARY KEY,

hourly\_rate SMALLMONEY,

weekly\_rate SMALLMONEY,

monthly\_rate MONEY,

CHECK(

hourly\_rate IS NOT NULL OR

weekly\_rate IS NOT NULL OR

monthly\_rate IS NOT NULL)

);

-- STEP 2: Insert data

INSERT INTO

EmployeeWages(

employee\_id,

hourly\_rate,

weekly\_rate,

monthly\_rate

)

VALUES

(1,60, NULL,NULL),

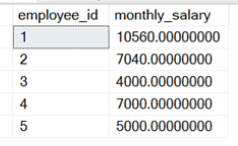
(2,40, NULL,NULL),

(3,NULL, 1000,NULL),

(4,NULL, NULL,7000),

(5,NULL, NULL,5000);

The task requires you to create the monthly salary table as below:



**CREATE TABLE monthly\_salary**

**AS**

**SELECT**

**employee\_id,**

**CASE**

**WHEN hourly\_rate IS NOT NULL THEN hourly\_rate \* 176**

**WHEN weekly\_rate IS NOT NULL THEN weekly\_rate \* 4**

**WHEN monthly\_rate IS NOT NULL THEN monthly\_rate**

**END AS monthly\_salary**

**FROM Employeewages;**