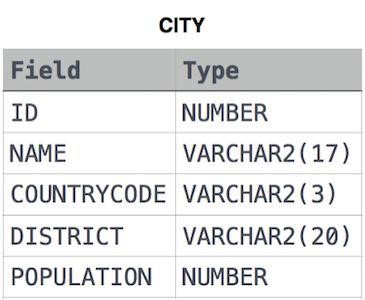
**SQL Questions**

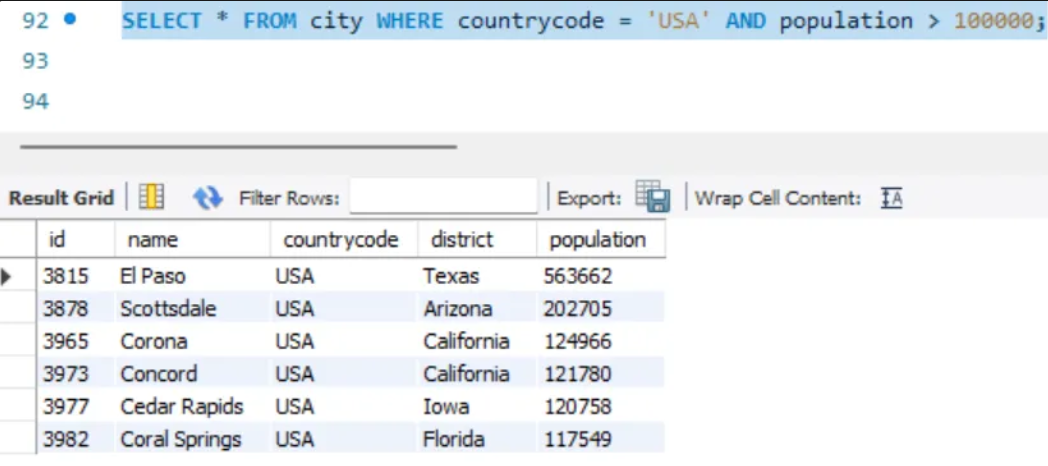
**Q1**. Query all columns for all American cities in the CITY table with populations larger than 100000.

The CountryCode for America is USA.

The CITY table is described as follows:

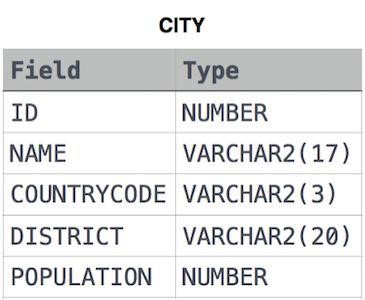


SELECT \* FROM city WHERE countrycode = 'USA' AND population > 100000;

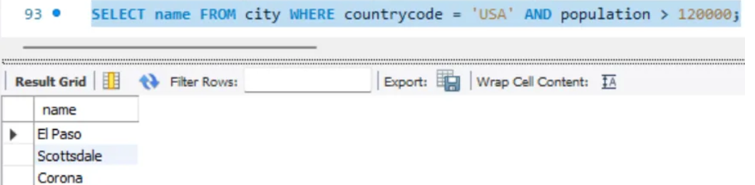


**Q2**. Query the NAME field for all American cities in the CITY table with populations larger than 120000. The CountryCode for America is USA.

The CITY table is described as follows:



SELECT name FROM city WHERE countrycode = 'USA' AND population > 120000;



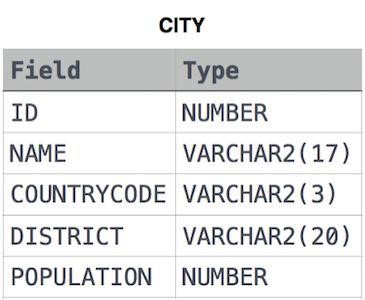
**Q3.** Query all columns (attributes) for every row in the CITY table. The CITY table is described as follows:   
A table with text and numbers

Description automatically generated

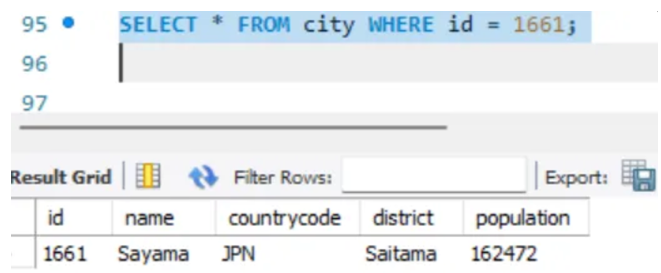
SELECT \* FROM city;



**Q4**. Query all columns for a city in CITY with the ID 1661. The CITY table is described as follows:

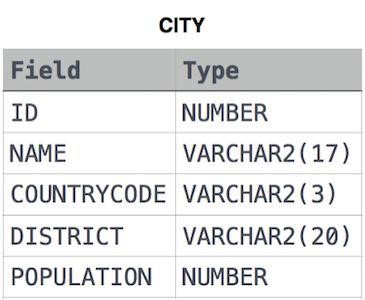


SELECT \* FROM city WHERE id = 1661;

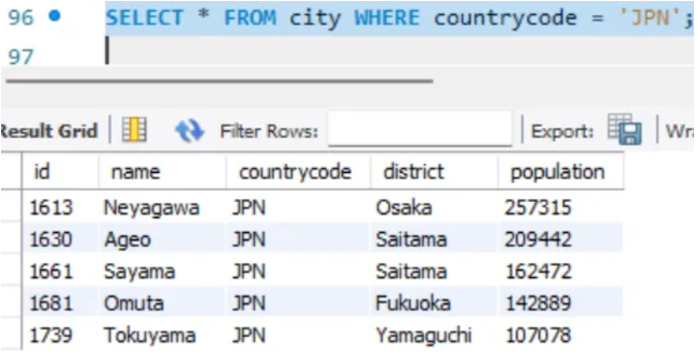


**Q5**. Query all attributes of every Japanese city in the CITY table. The COUNTRYCODE for Japan is JPN.

The CITY table is described as follows:

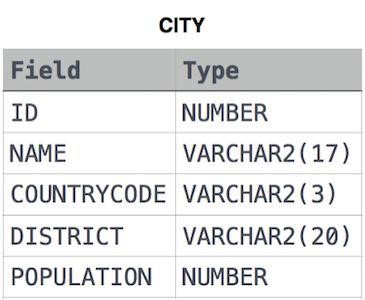


SELECT \* FROM city WHERE countrycode = 'JPN';

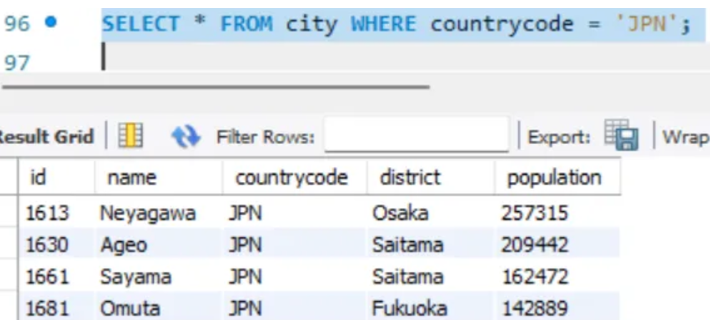


**Q6.** Query the names of all the Japanese cities in the CITY table. The COUNTRYCODE for Japan is JPN.

The CITY table is described as follows:



SELECT \* FROM city WHERE countrycode = 'JPN';



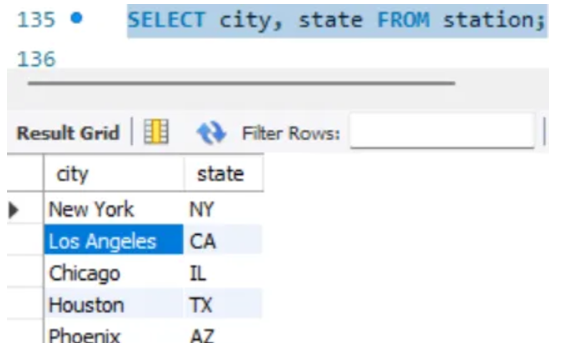
**Q7.** Query a list of CITY and STATE from the STATION table. The STATION table is described as follows:

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

A table with text and numbers

Description automatically generated

SELECT city, state FROM station;

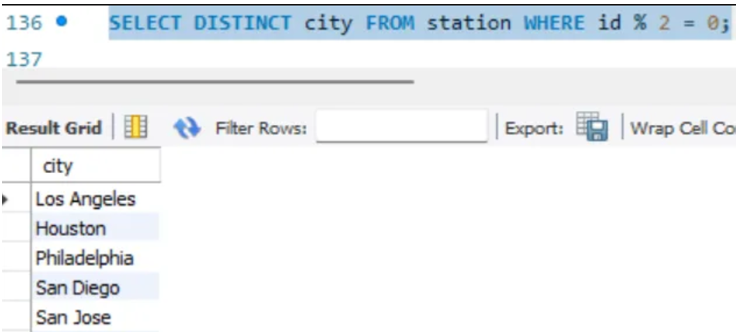


**Q8.** Query a list of CITY names from STATION for cities that have an even ID number. Print the results in any order, but exclude duplicates from the answer. The STATION table is described as follows:

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



SELECT DISTINCT city FROM station WHERE id % 2 = 0;



**Q9**. 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', 'Bengalaru', there are 2 different city names: 'New York' and 'Bengalaru'. The query returns , because total number of records - number of unique city names = 3-2 =1



SELECT (COUNT(city) - COUNT(DISTINCT city)) AS city\_difference FROM station;

A screenshot of a computer

Description automatically generated

**Q10.** 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:

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

Sample Input

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

Sample Output

ABC 3

PQRS 4



SELECT city, LENGTH(city) AS city\_length FROM station ORDER BY city\_length ASC, city LIMIT 1; -- shortest city  
A screen shot of a computer

Description automatically generated

SELECT city, LENGTH(city) AS city\_length FROM station ORDER BY city\_length DESC, city LIMIT 1; -- longest city

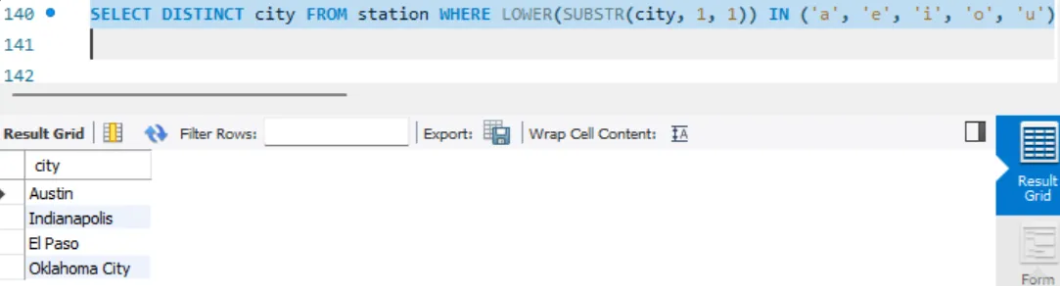
A screenshot of a computer

Description automatically generated  
**Q11**. 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:

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



SELECT DISTINCT city FROM station WHERE LOWER(SUBSTR(city, 1, 1)) IN ('a', 'e', 'i', 'o', 'u');



**Q12.** Query the list of CITY names ending with vowels (a, e, i, o, u) from STATION. Your result cannot contain duplicates.

Input Format

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 LOWER(SUBSTR(city, -1)) IN ('a', 'e', 'i', 'o', 'u');

A screenshot of a computer

Description automatically generated

**Q13.** Query the list of CITY names from STATION that do not start with vowels. Your result cannot contain duplicates.

Input Format

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 LOWER(SUBSTR(city, 1, 1)) NOT IN ('a', 'e', 'i', 'o', 'u');

A screenshot of a computer

Description automatically generated

**Q14.** Query the list of CITY names from STATION that do not end with vowels. Your result cannot contain duplicates.

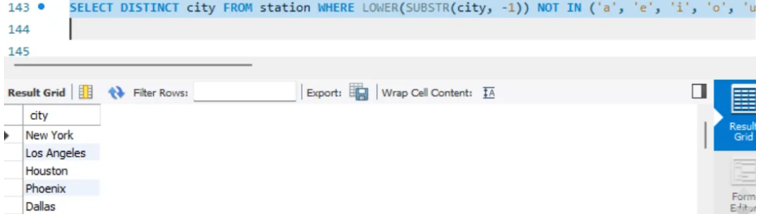
Input Format

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 LOWER(SUBSTR(city, -1)) NOT IN ('a', 'e', 'i', 'o', 'u');



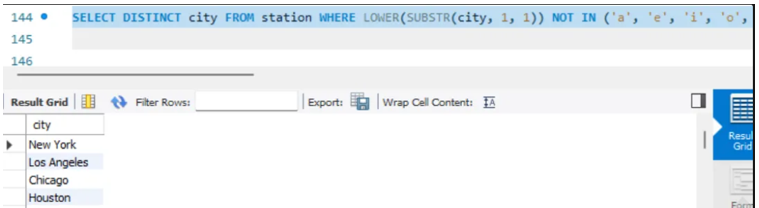
**Q15.** Query the list of CITY names from STATION that either do not start with vowels or do not end with vowels. Your result cannot contain duplicates.

Input Format

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 LOWER(SUBSTR(city, 1, 1)) NOT IN ('a', 'e', 'i', 'o', 'u') OR LOWER(SUBSTR(city, -1)) NOT IN ('a', 'e', 'i', 'o', 'u');



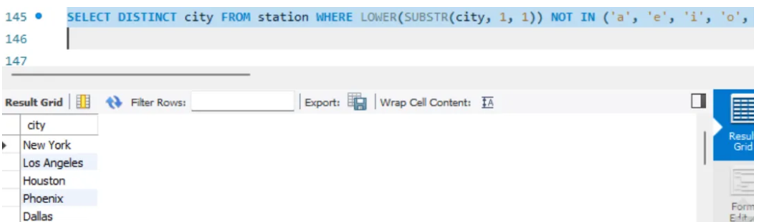
**Q16.** Query the list of CITY names from STATION that do not start with vowels and do not end with vowels. Your result cannot contain duplicates.

Input Format

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 LOWER(SUBSTR(city, 1, 1)) NOT IN ('a', 'e', 'i', 'o', 'u') AND LOWER(SUBSTR(city, -1)) NOT IN ('a', 'e', 'i', 'o', 'u');



**Q17.** Table: Product

|  |  |
| --- | --- |
| Column Name | Type |
| product\_id | int |
| product\_name | varchar |
| unit\_price | int |

product\_id is the primary key of this table.

Each row of this table indicates the name and the price of each product.

Table: Sales

|  |  |
| --- | --- |
| Column Name | Type |
| seller\_id | int |
| product\_id | int |
| buyer\_id | int |
| sale\_date | date |
| quantity | int |
| price | int |

This table has no primary key, it can have repeated rows.

product\_id is a foreign key to the Product table.

Each row of this table contains some information about one sale.

Write an SQL query that reports the products that were only sold in the first quarter of 2019. That is, between 2019-01-01 and 2019-03-31 inclusive.

Return the result table in any order.

The query result format is in the following example.

Input:

Product table:

|  |  |  |
| --- | --- | --- |
| product\_id | product\_name | unit\_price |
| 1 | S8 | 1000 |
| 2 | G4 | 800 |
| 3 | iPhone | 1400 |

Sales table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| seller\_id | product\_id | buyer\_id | sale\_date | quantity | price |
| 1 | 1 | 1 | 2019-01-21 | 2 | 2000 |
| 1 | 2 | 2 | 2019-02-17 | 1 | 800 |
| 2 | 2 | 3 | 2019-06-02 | 1 | 800 |
| 3 | 3 | 4 | 2019-05-13 | 2 | 2800 |

Output:

|  |  |
| --- | --- |
| product\_id | product\_name |
| 1 | S8 |

Explanation:

The product with id 1 was only sold in the spring of 2019.

The product with id 2 was sold in the spring of 2019 but was also sold after the spring of 2019.

The product with id 3 was sold after spring 2019.

We return only product 1 as it is the product that was only sold in the spring of 2019.

SELECT p.product\_id, p.product\_name, p.unit\_price FROM Product p

JOIN Sales s ON p.product\_id = s.product\_id

WHERE s.sale\_date BETWEEN '2019-01-01' AND '2019-03-31'

GROUP BY p.product\_id, p.product\_name, p.unit\_price

HAVING COUNT(DISTINCT CASE WHEN s.sale\_date > '2019-03-31' THEN 1 END)=0;

A screenshot of a computer

Description automatically generated

**Q18.** Table: Views

|  |  |
| --- | --- |
| Column Name | Type |
| article\_id | int |
| author\_id | int |
| viewer\_id | int |
| view\_date | date |

There is no primary key for this table, it may have duplicate rows.

Each row of this table indicates that some viewer viewed an article (written by some author) on some date.

Note that equal author\_id and viewer\_id indicate the same person.

Write an SQL query to find all the authors that viewed at least one of their own articles.

Return the result table sorted by id in ascending order. The query result format is in the following example.

Input:

Views table:

|  |  |  |  |
| --- | --- | --- | --- |
| article\_id | author\_id | viewer\_id | view\_date |
| 1 | 3 | 5 | 2019-08-01 |
| 1 | 3 | 6 | 2019-08-02 |
| 2 | 7 | 7 | 2019-08-01 |
| 2 | 7 | 6 | 2019-08-02 |
| 4 | 7 | 1 | 2019-07-22 |
| 3 | 4 | 4 | 2019-07-21 |
| 3 | 4 | 4 | 2019-07-21 |

Output:

|  |
| --- |
| id |
| 4 |
| 7 |

SELECT DISTINCT v.author\_id FROM Views v WHERE v.author\_id = v.viewer\_id ORDER BY v.author\_id ASC;

**A screenshot of a computer

Description automatically generated**

**Q19:** Table: Delivery

|  |  |
| --- | --- |
| Column Name | Type |
| delivery\_id | int |
| customer\_id | int |
| order\_date | date |
| customer\_pref\_delivery\_date | date |

delivery\_id is the primary key of this table.

The table holds information about food delivery to customers that make orders at some date and specify a preferred delivery date (on the same order date or after it).

If the customer's preferred delivery date is the same as the order date, then the order is called immediately; otherwise, it is called scheduled.

Write an SQL query to find the percentage of immediate orders in the table, rounded to 2 decimal places.

The query result format is in the following example.

Input:

Delivery table:

|  |  |  |  |
| --- | --- | --- | --- |
| delivery\_id | customer\_id | order\_date | customer\_pref\_ delivery\_date |
| 1 | 1 | 2019-08-01 | 2019-08-02 |
| 2 | 5 | 2019-08-02 | 2019-08-02 |
| 3 | 1 | 2019-08-11 | 2019-08-11 |
| 4 | 3 | 2019-08-24 | 2019-08-26 |
| 5 | 4 | 2019-08-21 | 2019-08-22 |
| 6 | 2 | 2019-08-11 | 2019-08-13 |

Output:

|  |
| --- |
| immediate\_percentage |
| 33.33 |

Explanation: The orders with delivery id 2 and 3 are immediate while the others are scheduled.

SELECT

ROUND(

(COUNT(CASE WHEN order\_date = customer\_pref\_delivery\_date THEN 1 END) \* 100.0)

/ COUNT(\*), 2) AS immediate\_order\_percentage

FROM Delivery;  
A screenshot of a computer

Description automatically generated  
  
 **Q20.**

Table: Ads

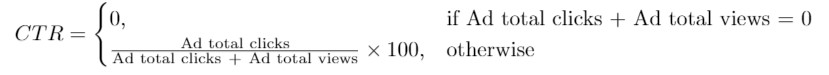
|  |  |
| --- | --- |
| Column Name | Type |
| ad\_id | int |
| user\_id | int |
| action | enum |

(ad\_id, user\_id) is the primary key for this table.

Each row of this table contains the ID of an Ad, the ID of a user, and the action taken by this user regarding this Ad.

The action column is an ENUM type of ('Clicked', 'Viewed', 'Ignored').

A company is running Ads and wants to calculate the performance of each Ad. Performance of the Ad is measured using Click-Through Rate (CTR) where:



Write an SQL query to find the ctr of each Ad. Round ctr to two decimal points.

Return the result table ordered by ctr in descending order and by ad\_id in ascending order in case of a tie.

The query result format is in the following example.

Input:

Ads table:

|  |  |  |
| --- | --- | --- |
| ad\_id | user\_id | action |
| 1 | 1 | Clicked |
| 2 | 2 | Clicked |
| 3 | 3 | Viewed |
| 5 | 5 | Ignored |
| 1 | 7 | Ignored |
| 2 | 7 | Viewed |
| 3 | 5 | Clicked |
| 1 | 4 | Viewed |
| 2 | 11 | Viewed |
| 1 | 2 | Clicked |

Output:

|  |  |
| --- | --- |
| ad\_id | ctr |
| 1 | 66.67 |
| 3 | 50 |
| 2 | 33.33 |
| 5 | 0 |

Explanation:

for ad\_id = 1, ctr = (2/(2+1)) \* 100 = 66.67 for ad\_id = 2, ctr = (1/(1+2)) \* 100 = 33.33 for ad\_id = 3, ctr = (1/(1+1)) \* 100 = 50.00 for ad\_id = 5, ctr = 0.00, Note that ad\_id = 5 has no clicks or views. Note that we do not care about Ignored Ads.  
  
SELECT ad\_id,

ROUND(SUM(CASE WHEN action = 'Clicked' THEN 1 ELSE 0 END) \* 100.0 /

SUM(CASE WHEN action IN ('Clicked', 'Viewed') THEN 1 ELSE 0 END), 2) AS ctr

FROM Ads

GROUP BY ad\_id

ORDER BY ctr DESC, ad\_id ASC;

A screenshot of a computer

Description automatically generated

**Q21.**

Table: Employee

|  |  |
| --- | --- |
| Column Name | Type |
| employee\_id | int |
| team\_id | int |

employee\_id is the primary key for this table.

Each row of this table contains the ID of each employee and their respective team.

Write an SQL query to find the team size of each of the employees.

Return result table in any order.

The query result format is in the following example.

Input:

Employee Table:

|  |  |
| --- | --- |
| employee\_id | team\_id |
| 1 | 8 |
| 2 | 8 |
| 3 | 8 |
| 4 | 7 |
| 5 | 9 |
| 6 | 9 |

Output:

|  |  |
| --- | --- |
| employee\_id | team\_size |
| 1 | 3 |
| 2 | 3 |
| 3 | 3 |
| 4 | 1 |
| 5 | 2 |
| 6 | 2 |

Explanation:

Employees with Id 1,2,3 are part of a team with team\_id = 8.

An employee with Id 4 is part of a team with team\_id = 7.

Employees with Id 5,6 are part of a team with team\_id = 9.  
  
SELECT e.employee\_id, e.team\_id, COUNT(\*) AS team\_size

FROM Employee e

JOIN Employee e2 ON e.team\_id = e2.team\_id

GROUP BY e.employee\_id, e.team\_id;

A screenshot of a computer

Description automatically generated

**Q22.**

Table: Countries

|  |  |
| --- | --- |
| Column Name | Type |
| country\_id | int |
| country\_name | varchar |

country\_id is the primary key for this table.

Each row of this table contains the ID and the name of one country.

Table: Weather

|  |  |
| --- | --- |
| Column Name | Type |
| country\_id | int |
| weather\_state | int |
| day | date |

(country\_id, day) is the primary key for this table.

Each row of this table indicates the weather state in a country for one day.

Write an SQL query to find the type of weather in each country for November 2019.

The type of weather is:

● Cold if the average weather\_state is less than or equal 15, ● Hot if the average weather\_state is greater than or equal to 25, and ● Warm otherwise.

Return result table in any order.

The query result format is in the following example.

Input:

Countries table:

|  |  |
| --- | --- |
| country\_id | country\_name |
| 2 | USA |
| 3 | Australia |
| 7 | Peru |
| 5 | China |
| 8 | Morocco |
| 9 | Spain |

Weather table:

|  |  |  |
| --- | --- | --- |
| country\_id | weather\_state | day |
| 2 | 15 | 2019-11-01 |
| 2 | 12 | 2019-10-28 |
| 2 | 12 | 2019-10-27 |
| 3 | -2 | 2019-11-10 |
| 3 | 0 | 2019-11-11 |
| 3 | 3 | 2019-11-12 |
| 5 | 16 | 2019-11-07 |
| 5 | 18 | 2019-11-09 |
| 5 | 21 | 2019-11-23 |
| 7 | 25 | 2019-11-28 |
| 7 | 22 | 2019-12-01 |
| 7 | 20 | 2019-12-02 |
| 8 | 25 | 2019-11-05 |
| 8 | 27 | 2019-11-15 |
| 8 | 31 | 2019-11-25 |
| 9 | 7 | 2019-10-23 |
| 9 | 3 | 2019-12-23 |

Output:

|  |  |
| --- | --- |
| country\_name | weather\_type |
| USA | Cold |
| Australia | Cold |
| Peru | Hot |
| Morocco | Hot |
| China | Warm |

Explanation:

Average weather\_state in the USA in November is (15) / 1 = 15 so the weather type is Cold.

Average weather\_state in Australia in November is (-2 + 0 + 3) / 3 = 0.333 so the weather type is Cold.

Average weather\_state in Peru in November is (25) / 1 = 25 so the weather type is Hot.

The average weather\_state in China in November is (16 + 18 + 21) / 3 = 18.333 so the weather type is warm.

Average weather\_state in Morocco in November is (25 + 27 + 31) / 3 = 27.667 so the weather type is Hot.

We know nothing about the average weather\_state in Spain in November so we do not include it in the result table.  
  
SELECT c.country\_name,

CASE

WHEN AVG(w.weather\_state) <= 15 THEN 'Cold'

WHEN AVG(w.weather\_state) >= 25 THEN 'Hot'

ELSE 'Warm'

END AS weather\_type

FROM Countries c

JOIN Weather w ON c.country\_id = w.country\_id

WHERE w.date BETWEEN '2019-11-01' AND '2019-11-30'

GROUP BY c.country\_name;

A screenshot of a computer program

Description automatically generated

**Q23.**

Table: Prices

|  |  |
| --- | --- |
| Column Name | Type |
| product\_id | int |
| start\_date | date |
| end\_date | date |
| price | int |

(product\_id, start\_date, end\_date) is the primary key for this table.

Each row of this table indicates the price of the product\_id in the period from start\_date to end\_date. For each product\_id there will be no two overlapping periods. That means there will be no two intersecting periods for the same product\_id.

Table: UnitsSold

|  |  |
| --- | --- |
| Column Name | Type |
| product\_id | int |
| purchase\_date | date |
| units | int |

There is no primary key for this table, it may contain duplicates.

Each row of this table indicates the date, units, and product\_id of each product sold.

Write an SQL query to find the average selling price for each product. average\_price should be rounded to 2 decimal places.

Return the result table in any order.

|  |  |  |  |
| --- | --- | --- | --- |
| product\_id | start\_date | end\_date | price |
| 1 | 2019-02-17 | 2019-02-28 | 5 |
| 1 | 2019-03-01 | 2019-03-22 | 20 |
| 2 | 2019-02-01 | 2019-02-20 | 15 |
| 2 | 2019-02-21 | 2019-03-31 | 30 |

The query result format is in the following example.  
Input:  
Prices table:

UnitsSold table:

|  |  |  |
| --- | --- | --- |
| product\_id | purchase\_date | units |
| 1 | 2019-02-25 | 100 |
| 1 | 2019-03-01 | 15 |
| 2 | 2019-02-10 | 200 |
| 2 | 2019-03-22 | 30 |

Output:

|  |  |
| --- | --- |
| product\_id | average\_price |
| 1 | 6.96 |
| 2 | 16.96 |

Explanation:

Average selling price = Total Price of Product / Number of products sold.

Average selling price for product 1 = ((100 \* 5) + (15 \* 20)) / 115 = 6.96 Average selling price for product 2 = ((200 \* 15) + (30 \* 30)) / 230 = 16.96  
  
SELECT u.product\_id,

ROUND(SUM(p.price \* u.units) / SUM(u.units), 2) AS average\_price

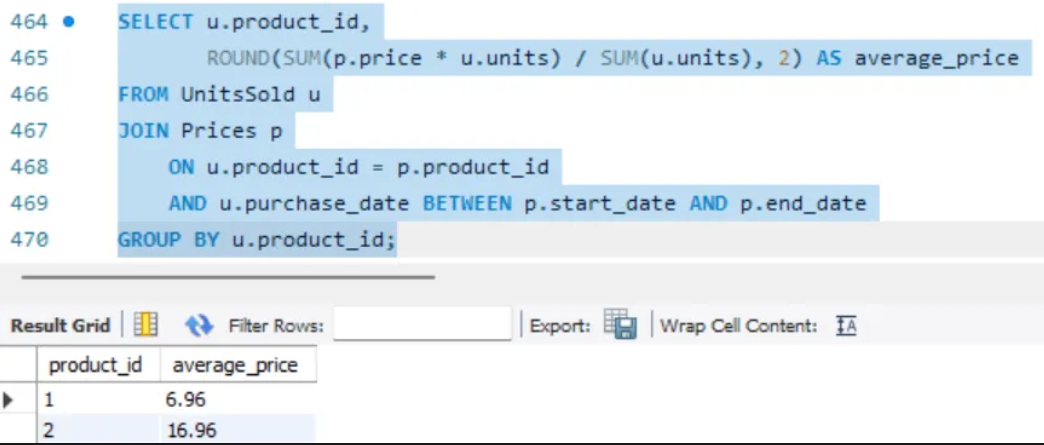
FROM UnitsSold u

JOIN Prices p

ON u.product\_id = p.product\_id

AND u.purchase\_date BETWEEN p.start\_date AND p.end\_date

GROUP BY u.product\_id;



**Q24.**

Table: Activity

|  |  |
| --- | --- |
| Column Name | Type |
| player\_id | int |
| device\_id | int |
| event\_date | date |
| games\_played | int |

(player\_id, event\_date) is the primary key of this table.

This table shows the activity of players of some games.

Each row is a record of a player who logged in and played a number of games (possibly 0) before logging out on someday using some device.

Write an SQL query to report the first login date for each player.

Return the result table in any order.

The query result format is in the following example.

Input:

Activity table:

|  |  |  |  |
| --- | --- | --- | --- |
| player\_id | device\_id | event\_date | games\_played |
| 1 | 2 | 2016-03-01 | 5 |
| 1 | 2 | 2016-05-02 | 6 |
| 2 | 3 | 2017-06-25 | 1 |
| 3 | 1 | 2016-03-02 | 0 |
| 3 | 4 | 2018-07-03 | 5 |

Output:

|  |  |
| --- | --- |
| player\_id | first\_login |
| 1 | 2016-03-01 |
| 2 | 2017-06-25 |
| 3 | 2016-03-02 |

**SELECT player\_id, MIN(event\_date) AS first\_login FROM Activity GROUP BY player\_id;  
  
A screenshot of a computer

Description automatically generated**

**Q25.**

Table: Activity

|  |  |
| --- | --- |
| Column Name | Type |
| player\_id | int |
| device\_id | int |
| event\_date | date |
| games\_played | int |

(player\_id, event\_date) is the primary key of this table.

This table shows the activity of players of some games.

Each row is a record of a player who logged in and played a number of games (possibly 0) before logging out on someday using some device.

Write an SQL query to report the device that is first logged in for each player.

Return the result table in any order.

The query result format is in the following example.

Input:

Activity table:

|  |  |  |  |
| --- | --- | --- | --- |
| player\_id | device\_id | event\_date | games\_played |
| 1 | 2 | 2016-03-01 | 5 |
| 1 | 2 | 2016-05-02 | 6 |
| 2 | 3 | 2017-06-25 | 1 |
| 3 | 1 | 2016-03-02 | 0 |
| 3 | 4 | 2018-07-03 | 5 |

Output:

|  |  |
| --- | --- |
| player\_id | device\_id |
| 1 | 2 |
| 2 | 3 |
| 3 | 1 |

**SELECT a.player\_id, a.device\_id**

**FROM Activity1 a**

**JOIN (**

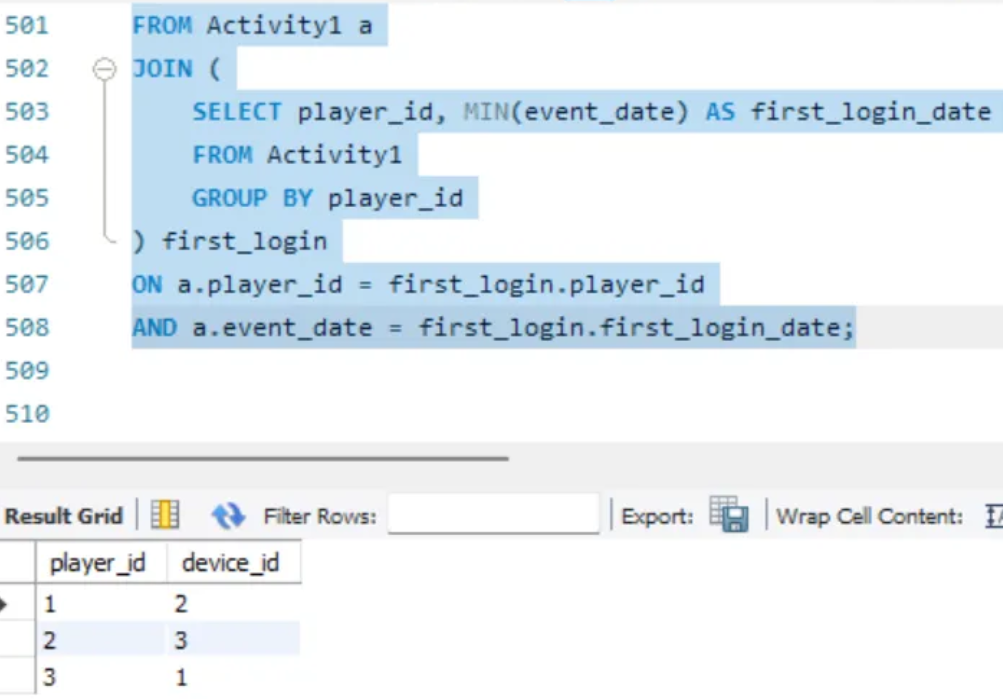
**SELECT player\_id, MIN(event\_date) AS first\_login\_date**

**FROM Activity1**

**GROUP BY player\_id**

**) first\_login**

**ON a.player\_id = first\_login.player\_id**

**AND a.event\_date = first\_login.first\_login\_date;  
  
**