Name: QIAN ZHANG

Date: 12/9/20.

Part A

Question 1

London is a city that could be found on the map of United Kingdom, and United Kingdom is a country that could be found on the map of British Isles, so London could be found on the map of British Isles.

Question 2

1. No, we could consider an address as a single entity, so saving the whole address in a table cell does not violate 1NF.
2. If we save the entire address into a table cell, and someday we need to modify the country information, we will need to update every single row of the address separately, which could result in update error. For example, we have a table which stores the addresses of residents in Puerto Rico, and someday Puerto Rico becomes an official state of United States. Therefore, if we save the entire address into a slot of table, we need to modify every row of address for residents in Puerto Rico, which takes much efforts.

Question 3

2NF: for a table to be in 2NF, it should first follow all the constraint of 1NF, so there is only 1 single and indivisible entity in every slot of the table. Moreover, all the non-prime attributes in 2NF should have the functional dependency on the whole set of candidate keys. If any of the non-prime attributes have the functional dependency on the proper subsets of candidate keys, instead of the whole set, the table is not in 2NF

3NF: for a table to be in 3NF, it should first follow all the constraints of 2NF and 1NF. Moreover, all the non-prime attribute should have a direct functional dependency on the whole set of candidate keys. If the dependency is transitive, for example, one non-prime attribute C is depended on the whole set of candidate keys of A and B, and another non-prime attribute D is depended on non-prime attribute C. Although there is functional dependency between non-prime attribute D and the whole set of candidate keys of A and B, this dependency is transitive, and the table is not in 3NF.

Question 4

1. Way 1: the name of attribute is too redundant and include unnecessary information

Way 2: does not explicitly identify the meaning of this attributes, which might confuse the user who tries to export value from the database.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Name | Description | Datatype |
| Example 1: | address | This attribute stores the address of all supermarket in United States | varchar(50) |
| Example 2 | price | This attribute stores the price of commodities in supermarket | decimal |
| Example 3 | purchase\_time | This attribute stores the time that a commodity is paid with the cashier | text |

Why it is bad:

* Example 1: The address might need space more than **varchar(50)** could store, but the maximum accepted size for **varchar** is fixed. Therefore, **text** is preferred.
* Example 2: The type **decimal** requires the exact precision. However, the precision of price of commodities might vary because of seasonal discount. Therefore, **float**, which does not have exact precision, is preferred.
* Example 3: It is better to use **timestamp** to store any date and time. Using the datatype **timestamp**, it is easier for user to access or update the time data. For example, if a user wants to change the value of hour of a timestamp, it will be much easier when the data is stored in **timestamp**.

Question 5

A lots error, such as update error or delete error, could happen if the data scientists use a folder full of spreadsheets as database. Also, using SQL database could improve the efficiency of accessing the database or running algorithms to analyze the data.

For example, if these data scientists work for the HR departments for the company, they will need to record the information of employees, including the salary, department or title for each individual, in the spreadsheets. If one department got cancelled and the data scientists simply delete the spreadsheet which corresponds to this cancelled department, a lot of necessary information about these employees who work in this department before it is canceled might be lost.

Moreover, if these employees are re-assigned to other departments, the data scientists might need to input their information into the new spreadsheets, which might take a lot of time and efforts and could be avoided if the data scientists use a relational paradigm.

Question 6

When we are designing the database for a start-up company, we might need to avoid of using SQL database at the very early stage of the start-up company, since a start-up company might not have a mature business model. Therefore, even the database designers design a standard SQL database for the start-up company, the time and efforts that the designers spend on meeting all the constraints of normalization might be wasted when the start-up company modifies its business model. Therefore, the start-up company might use a flexible database first, rather than a SQL database which takes enormous time and efforts to create and update.

Part B

Question 1

select dogs.name,customers.first\_name,customers.last\_name

from dogs inner join customers

on dogs.owner\_id = customers.id

Question 2

select dogs.id,count(\*) as number\_of\_bookings

from dogs inner join bookings

on dogs.id = bookings.dog\_id

group by dogs.id

Question 3

select bookings.id, customers.last\_name,sum(bookings.end-bookings.start) over (order by bookings.start)

from bookings inner join dogs

on bookings.dog\_id = dogs.id inner join customers

on dogs.owner\_id = customers.id

where bookings.kennel\_id = 7

and (bookings.start is null or (bookings.start is not null and bookings.end is null))

order by customers.last\_name

Question 4

select sum(capacity)

from kennels

Question 5

select dogs.owner\_id,sum(bookings.end-bookings.start) as amount\_of\_time

from dogs inner join bookings

on dogs.id = bookings.dog\_id

where bookings.start is not null

and bookings.end is not null

group by dogs.owner\_id

Question 6

select dogs.owner\_id,count(distinct dogs.id) as number\_of\_dog,sum(payments.amnount) as total\_payment

from dogs inner join bookings

on dogs.id = bookings.id inner join payments

bookings.id = payments.booking\_id

group by dogs.owner\_id

order by total\_payment desc

limit 10

Question 7

We could consider merge the table of customers and dogs by directly saving the first name and last name of dog owner inside the table of dogs, so we no longer need to join customers to dogs if we want to find one particular customer’ s total spending. The new table for dog will look like below

|  |  |
| --- | --- |
| dogs | |
| id | serial |
| email | text |
| name | text |
| age | text |
| gender | text |
| customer\_first\_name | text |
| customer\_last\_name | text |

However, this modification might result in data corruption when, for example, a customer who has multiple dogs changes his/her name. Then we need to update the name of the customer in every table of his/her dogs. It might result in update error if we fail to update every table, for example, when we are identifying one customers’ dogs given his/her first and last name.

Question 8

We could add an additional attribute to the table of bookings. For example, we could add a new Boolean attribute called **cancelled**, whose value is TRUE if the booking is cancelled, or FALSE if the booking is not cancelled.

We don’t want to delete bookings from the database directly because it might result in error under multiple various scenarios. For example, the customer might want to reinitiate the cancelled bookings. If we add an additional attribute to track the cancelled bookings like the modified schema above, we could simply modify a single value of data to reinitiated the booking. However, if we delete the booking from the database while the customer cancels the booking, we need to input all the data again, which takes additional efforts.

Moreover, the customer might make the payment before he/she cancels the bookings. Therefore, if we delete the booking from the database while the customer cancels the booking, we will fail to track the payment data and it might result in more trouble.