**Unit Test**

**Question 1**

Our marketing department has a spreadsheet with potential customers and their birthdates in the format d-m-yyyy or dd-mm-yyyy (both formats are used, including d-mm-yyyy and dd-m-yyyy). The marketing department wants to filter the file to retain only people born between 1985 and 2005 (inclusive). Regex details:

1. We do not need to identify valid dates, because all birthdates on the spreadsheet are valid. So you can match 44-55-1985, no problem.
2. For the years, you must describe the alternatives as follows (solutions like (1985)|(1986)|(1987) etc. are not accepted):

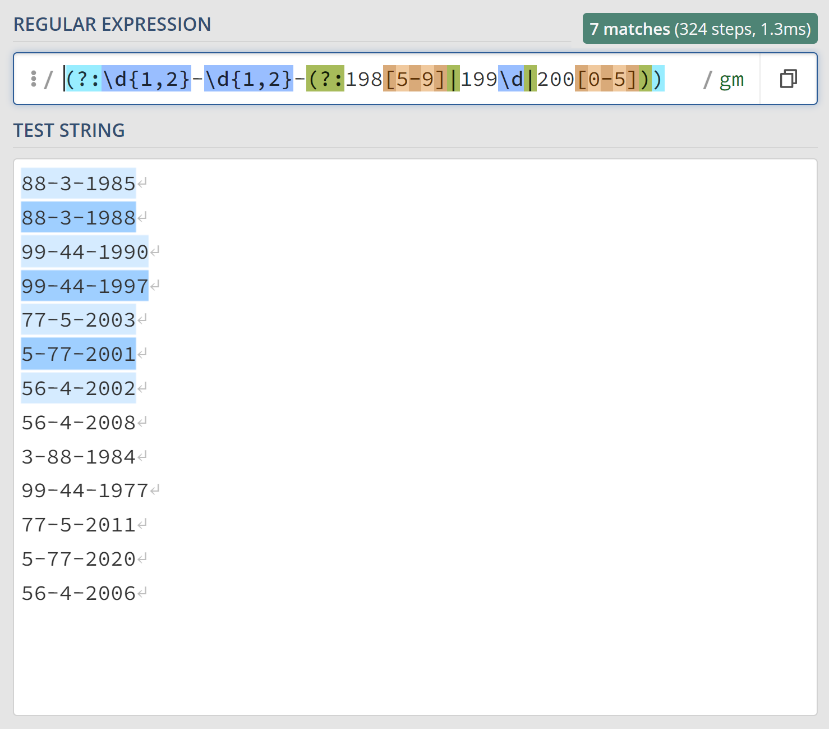
* 1985 – 1989
* 1990 – 1999
* 2000 - 2005

Your regex should match:

* 88-3-1985
* 88-3-1988
* 99-44-1990
* 99-44-1997
* 77-5-2003
* 5-77-2001
* 56-4-2002

Your regex should not match:

* 56-4-2008
* 3-88-1984
* 99-44-1977
* 77-5-2011
* 5-77-2020
* 56-4-2006



**Question 2**

EasyFit run classes for people who want to improve their fitness. Clients do exercises together under the instruction of a physiotherapist. EasyFit need the data structures to store these classes and their participants (the clients).

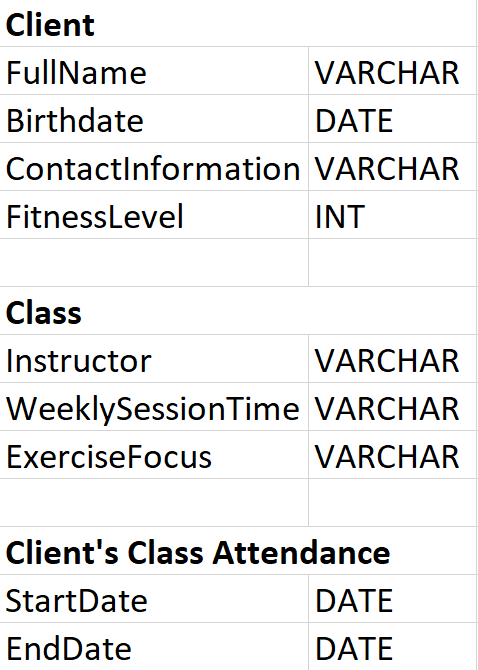
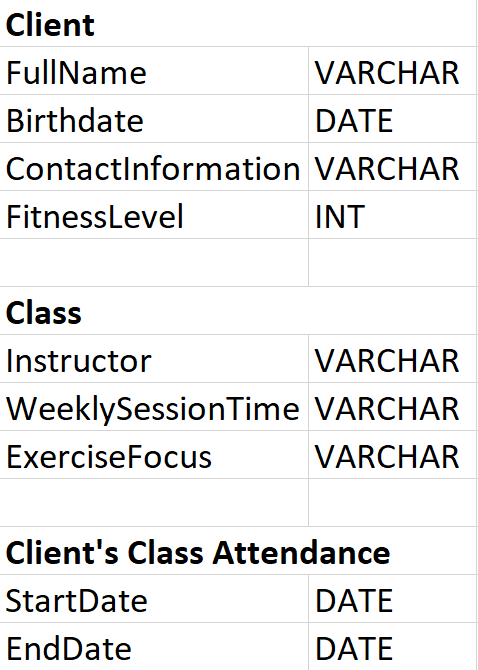
For each class, we need to record the instructor as well as the class's weekly session time (e.g. Monday 5pm), and the focus of the exercise (spine, upper body, etc.). Clients have a birthdate and contact information as well as a fitness level. Clients may move between classes (from a spine class to an upper body class, for example), so we need to know from when to when a person was attending a class. We are assuming that a client only attends one class at a time.

The information contains three main concepts:

* the client;
* the class;
* the client's class attendance.

For each of these concepts, make a list of attributes that are specific to the concept. For each attribute, add the datatype you would use.

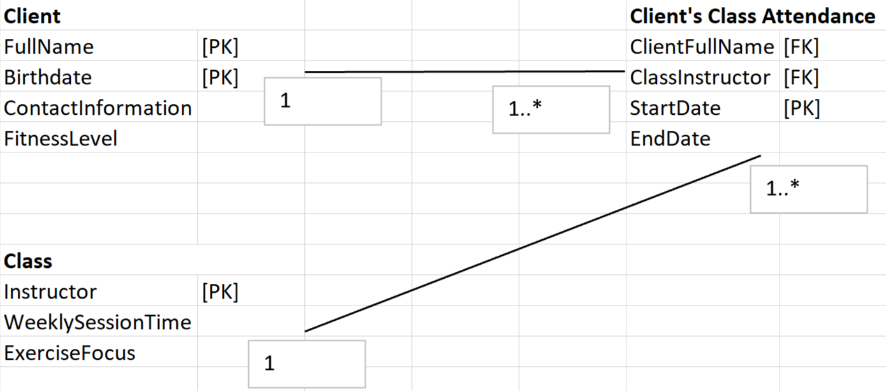
A white sheet with black text

Description automatically generated

**Question 3**

Based on your answer to Question 2, create and upload a UML diagram that shows the tables that capture the situation.

The UML diagram must include the cardinalities and the primary keys of each table must be defined (mark them as [PK]). Ensure that each client can be enrolled in a different classes over time, and each class can have multiple clients enrolled. For each key, mention possible drawbacks of the key you have chosen. Mark the foreign keys in the table ([FK]).



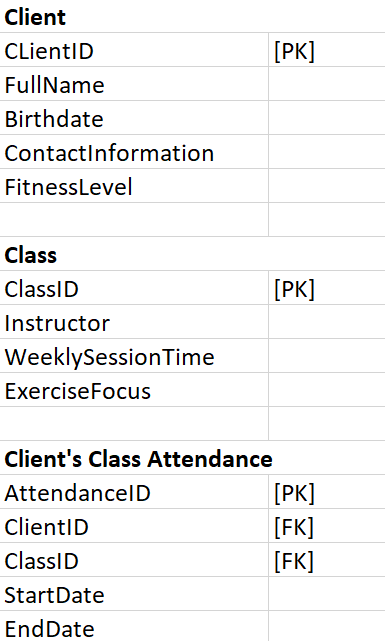
The Client's Class Attendance will reference Client and Class as foreign keys.

The relationships are:

* One Client can have many Client's Class Attendance records.
* One Class can have many Client's Class Attendance records.

**Question 4**

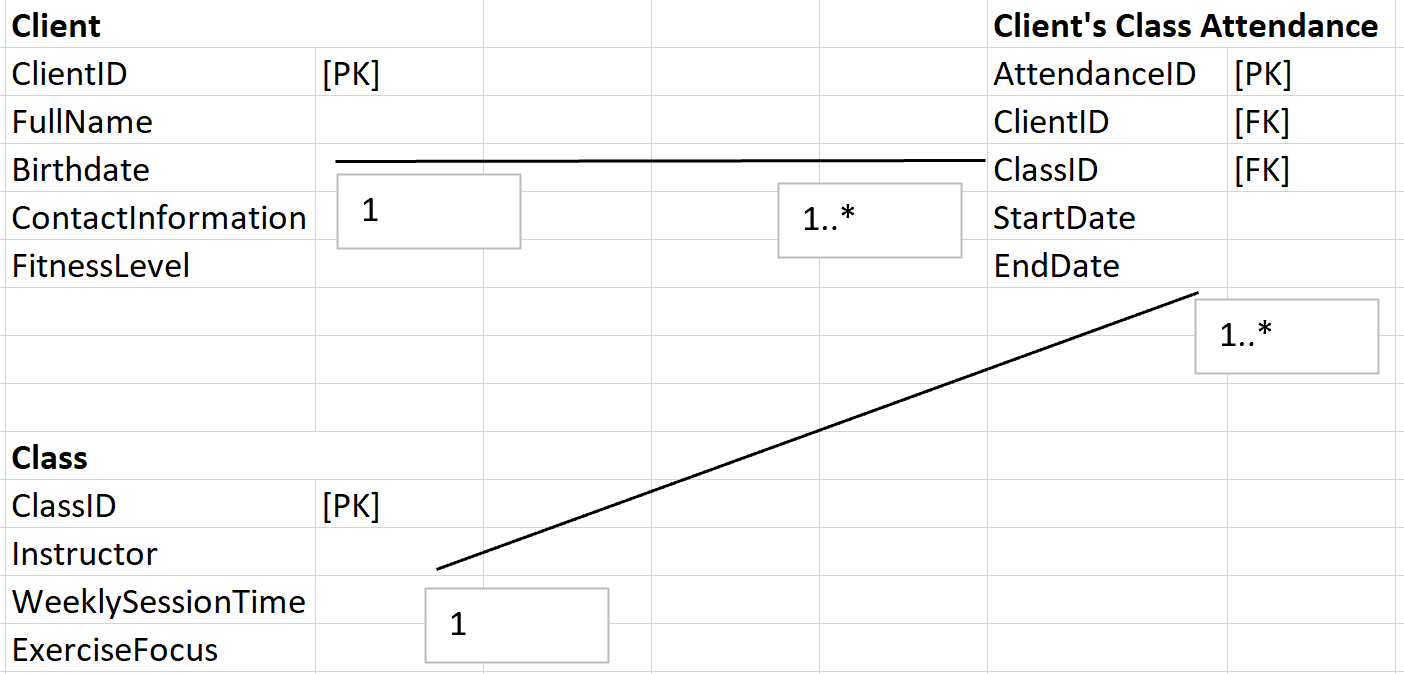
Change your model by replacing the composite keys with a surrogate. Adjust the foreign keys to match the new primary key where appropriate.

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

By using these surrogate keys as primary keys, it is much easier to manage and maintain the database, without worrying about changes in other attributes affecting the primary key. The relationships and connections between the tables remain the same as described in the previous message, but now they use the surrogate keys for linking records.

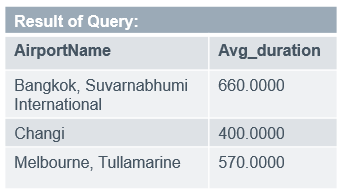


**Question 5**

These two tables are a part of an aviation database. They store all airports of the world (only an excerpt is shown) and all flights that run. The flights go from the departure airport (dep\_airport) to the destination airport (dest\_airport). Airports are identified by their IATA codes, flights are identified by a unique number.



We need to find out the average flight duration of flights departing from the dep\_airports. For the entries shown in the Airport and Flight tables, the result looks like this:



Write the query that creates this result.

SELECT a.name AS AirportName, AVG(f.mins) AS Avg\_duration FROM Flight AS f JOIN Airport AS a ON f.dep\_airport = a.airportID GROUP BY a.name;

* Use JOIN to combine data from the Flight and Airport tables based on the airportID and dep\_airport.
* SELECT is used to specify the name from the Airport table (aliased as AirportName) and the average mins from the Flight table (aliased as Avg\_duration) as the columns to return.
* The AVG(f.mins) function is used to calculate the average flight duration for each departure airport.
* The GROUP BY a.name clause is used to group the results by the airport name, so that the average is calculated separately for each airport.
* The semicolon (;) signifies the end of the SQL query.