Health Informatics 2016/2017
Assignment # ____1____
Student # 77020 **Name:** Joaquim Brotas Vasconcelos Esteves **Degree:** MEIC (7.5 ECTS)

Answer to Question: 1

Q1. One of the major difficulties of handling medical data is its „double nature", having to collect and integrate in the same record heterogeneous elements related to high and low level processes, such as i) lab tests, ii) images, iii) genetic data, iv) family's medical history and v) appraisals of emotional states. Please indicate the data types and/or representation formats that you would use for each of the above five information elements.

It is important to separate the way the data is stored, from the way it is represented. It's clear that the interface design should take top priority. Nonetheless it is necessary to clarify some points on the best way to organize the data in terms of computer science.

The typical computer design answer involving object oriented design must be disregarded forthwith. Since we're dealing with very complex types of data it appears to make sense, that the we should split something like a lab test or a patient's family medical history into a class/subclass and to create methods that would then tweak the representation of data. In a perfect world, this would be the optimal solution, whoever we must consider that medicine is both a constantly evolving subject and that not every hospital uses the same system. To an American system programmed in *python*, a Finnish patients *C++* object might as well be gibberish.

Hence the solution to the problem usually revolve around markup languages, such as XML or HL7. This type of approach provides us with many advantages, these types of documents are generally very light, easily encryptable quickly parsable and very flexible.

Let's consider one of the five examples given in this question namely lab tests. It is easy to see how to translate these to a markup language, one would start by defining the test with some sort of identifying tab followed by the various attributes that describe the results of the test. A new type of test would require nothing but a new document, and more attributes or a different way to describe attributes (Standard Kg vs English Stone) can be easily added without making old test results obsolete. Even images can be implemented with a markup language by simply describing the way they are encoded, for example:

<jpeg-base64> adsauda…….0ud88iuj8 </jpeg-base64>

In computer science data is nothing more than an organized collection of strings, ultimately the far more important and difficult task revolves around representing that data in a way that the user can benefit from.
To that lengthy and complex topic we can only regard with certainty that there is no single way display medical data. This is due to the fact that we are dealing with an excessively diverse use cases, from elderly computer illiterate patients to medical researchers, nurses and pharmacists. This is an area of study that must be dealt case by case, dividing each of the major areas of the use cases as much as possible and designing separate interfaces from there.

Health Informatics 2016/2017
Assignment # ____1____
Student # 77020 **Name:** Joaquim Brotas Vasconcelos Esteves **Degree:** MEIC (7.5 ECTS)

Answer to Question: 2

Q2. Given the imprecision of many medical terms, why do you think that serious instances of miscommunication among health care professionals are not more common? Explain also, in your own words, why is greater standardization of terminology necessary if computers rather than humans are to manipulate patient data?

The lack of major communications problems can the thanked in large part to the history of health care.

"Increasingly over the latter half of the 20th century, English became the Lingua Franca of medicine, in both international and intra-national communication." – Showell, C., Cummings, E. and Turner, P. (2010)

The recent information age in particular, has in many ways armed professionals, further dispelling miscommunication.

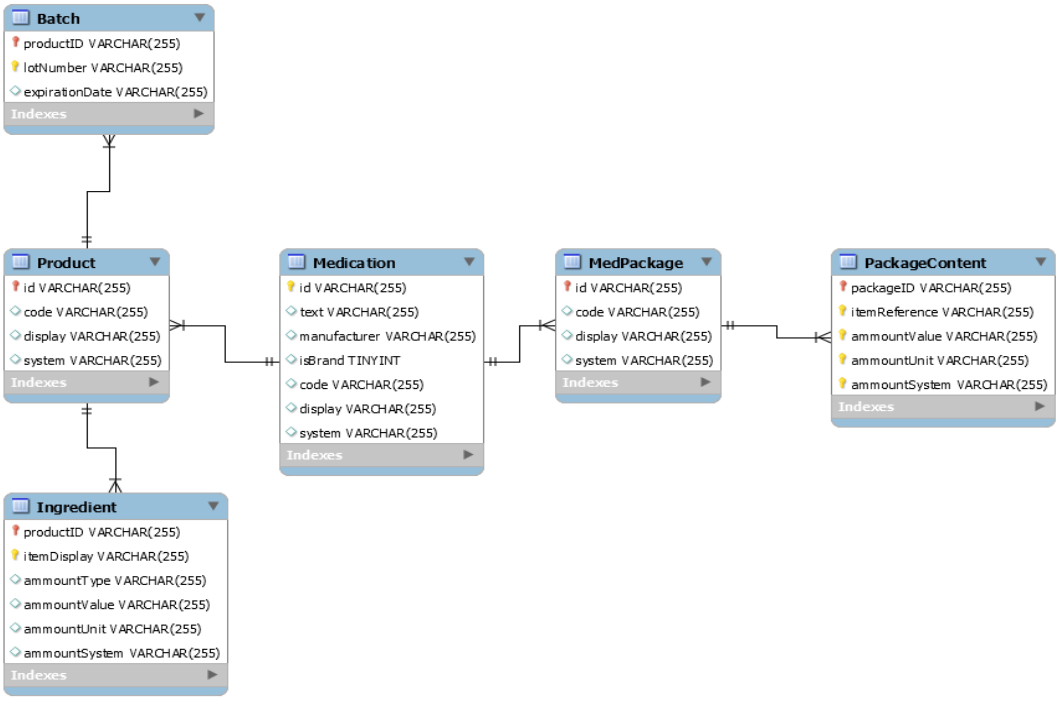
Nevertheless, computers have yet to master natural language, and as such a greater effort must be made in regards to the standardization of terminology in the medical field. Often, the medical professionals work is locally restrictive, as such, on top of the typical language barrier problems we should consider the prevalence of slang and shorthand terms. The computer systems must accommodate these issues and colloquial differences, as forcing an industry as large as medicine to adapt too quickly would result, at best, in a severe backlash and at worst the loss of human life. It's no small coincidence that the very start of this answer deals with serious instances of miscommunication, whose potential consequences could be catastrophic.

Ultimately however the hardest problem when it comes to terminology revolves around ambiguity. Experienced doctors are naturally accustomed to ambiguity, but computers have a rather appalling record when it comes dealing with ambiguous data.

Health Informatics 2016/2017
Assignment # ____1____
Student # 77020 **Name:** Joaquim Brotas Vasconcelos Esteves **Degree:** MEIC (7.5 ECTS)

Q3CS. Present an Entity-Relationship (ER) model of the medication resource. Convert also the ER model to the relational model, and implement the database in a database management system of your choice (suggestion: sqlite3). Your answer should contain (i) the ER model, and (ii) the SQL instructions to define the relational schema of the database.

EER Model:



Relational Model:

Medication (id ,text,manufacturer , isBrand , code , display , system)
unique(id)

Product(id,code,display,system)
id: FK Medication(id)

MedPackage(id,code,display,system)
id: FK Medication(id)

PackageContent (packageID,itemReference,ammountValue,ammountUnit,ammountSystem)
primary key(packageID,itemReference, ammountValue, ammountUnit, ammountSystem),
packageID: FK MedPackage(id)

Health Informatics 2016/2017
Assignment # ____1____
Student # 77020 **Name:** Joaquim Brotas Vasconcelos Esteves **Degree:** MEIC (7.5 ECTS)

Ingredient (productID,itemDisplay,ammountValue,ammountUnit,ammountSystem)
primary key(packageID,itemReference, ammountValue, ammountUnit, ammountSystem),
packageID: FK MedPackage(id)

Batch (productID,lotNumber,expirationDate)
ProductID: FK Product(id)
primary key (productID,lotNumber),

The schema is very large, it would perhaps increase readability if the reader would be so kind as to view the source code from the following URL, or the attached file:
https://github.com/JoaquimEsteves/Biomedical-Computing/blob/master/Delivery_1/schema.sql

```
drop table
if exists
Ingredient;
```

```
drop table if exists Batch;
drop table if exists PackageContent;
drop table if exists MedPackage;
drop table if exists Product;
drop table if exists Medication;
```

```
create table Medication (
    id varchar(255) not null unique,
    text varchar(255),
    manufacturer varchar(255),
       isBrand varchar(30),
       code varchar(255),
       display varchar(255),
       system varchar(255),
    primary key(id)
);

create table Product (
    id varchar(255) not null unique,
       text varchar(255),
       code varchar(255),
       display varchar(255),
       system varchar(255),
    primary key(id),
       foreign key(id) references Medication(id) ON DELETE CASCADE ON UPDATE
CASCADE
```

```
);


create table MedPackage (
	id varchar(255) not null unique,
	text varchar(255),
	code varchar(255),
	display varchar(255),
	system varchar(255),
	primary key(id),
	foreign key(id) references Medication(id) ON DELETE CASCADE ON UPDATE
CASCADE
);


create table PackageContent (
	packageID varchar(255) not null unique,
    itemReference varchar(255),
	ammountValue varchar(255),
	ammountUnit varchar(255),
	ammountSystem varchar(255),
    primary key(packageID,itemReference, ammountValue, ammountUnit,
ammountSystem),
    foreign key(packageID) references MedPackage(id) ON DELETE CASCADE ON UPDATE
CASCADE
);
create table Ingredient (
	productID varchar(255) not null,
	itemDisplay varchar(255),
	ammountType varchar(255),
	ammountValue varchar(255),
	ammountUnit varchar(255),
	ammountSystem varchar(255),
	primary key (productID, itemDisplay,ammountType),
	foreign key (productID) references Product(id) ON DELETE CASCADE ON
UPDATE CASCADE
);


create table Batch (
	productID varchar(255) not null unique,
	lotNumber varchar(255),
```

```
        expirationDate varchar(255),
        primary key (productID,lotNumber),
        foreign key (productID) references Product(id) ON DELETE CASCADE ON
UPDATE CASCADE
);
```

Q4CS. Write a program to populate the relational database from JSON files encoding medication resources according to the HL7-PHIR format (suggestion: use Python with the sqlite library4).

The program is very large, it would perhaps increase readability if the reader would be so kind as to view the source code from the following URL, or the attached file:
https://github.com/JoaquimEsteves/Biomedical-Computing/blob/master/Delivery_1/jsonConverter.py

```
#!/usr/bin/py
thon
        # -*- coding: utf-8 -*-
        import json
        import argparse
        import sqlite3 as lite
        import sys
        import base64
        from pprint import pprint


        def parseMedication(cur,con,data):
                if data["resourceType"] != "Medication":
                        print "This is not a medication!"
                        return
                medID =  "'"+json.dumps(data["id"])+"'"
                text = 'NULL'
                encodedText = 'NULL'
                manufacturer = 'NULL'
                isBrand = 'NULL'
                code = 'NULL'
                display = 'NULL'
                system = 'NULL'
                if 'text' in data :
                        #ALWAYS ENCODE PLAIN TEXT WHEN DEALING WITH JSONS
                        #JSONS ARE REALLY BLOODY FINNICKY
                        text = json.dumps(data["text"])
                        encodedText = "'"+str(base64.b64encode(text))+"'"
                if 'manufacturer' in data:
                        manufacturer =
"'"+json.dumps(data['manufacturer']["reference"])+"'"
                if 'isBrand' in data:
```

```
                isBrand = "'"+json.dumps(data['isBrand'])+"'"
        if 'code' in data:
                #For some SILLY REASON, code also apparently can have text!
                if 'text' in data['code']:
                        text = base64.b64decode(encodedText) + "Coding Text: " +
json.dumps(data['code']['text'])
                        encodedText = "'"+str(base64.b64encode(text))+"'"
                # coding = "'"+str(data['code']['coding'])+"'"
                if 'coding' in data['code']:
                        code = "'["
                        display = "'["
                        system = "'["
                        max_items = len(data['code']['coding'])
                        i = 0
                        while i < max_items:
                                code +=
json.dumps(data['code']['coding'][i]['code'])+","
                                display +=
json.dumps(data['code']['coding'][i]['display'])+","
                                system +=
json.dumps(data['code']['coding'][i]['system'])+","
                                i += 1
                        code += "]'"
                        display += "]'"
                        system += "]'"
        medicationInsert = "INSERT INTO Medication VALUES ("+medID+""
        medicationInsert +=
        ","+encodedText+","+manufacturer+","+isBrand+","+code+","+display+","+
system+");"
        cur.execute(medicationInsert)
        con.commit();
        cur.execute("SELECT * FROM Medication;")
        print "\n\n\n\nPrinting out the complete list of Medications!"
        print cur.fetchall()
        return medID


def parseProduct(cur,con,data,medID):
        text = 'NULL'
        encodedText = 'NULL'
        code = 'NULL'
        display = 'NULL'
        system = 'NULL'
        if 'form' in data["product"]:
```

```
                if 'coding' in data["product"]["form"]:
                        #For some SILLY REASON, code also apparently can have
text!
                        if 'text' in data["product"]["form"]:
                                text =
json.dumps(data["product"]["form"]['text'])
                                encodedText =
"'"+str(base64.b64encode(text))+"'"
                        # coding = "'"+str(data['code']['coding'])+"'"
                        if 'coding' in data["product"]["form"]:
                                code = "'["
                                display = "'["
                                system = "'["
                                max_items =
len(data["product"]["form"]['coding'])
                                i = 0
                                while i < max_items:
                                        code +=
json.dumps(data["product"]["form"]['coding'][i]['code'])+","
                                        display +=
json.dumps(data["product"]["form"]['coding'][i]['display'])+","
                                        system +=
json.dumps(data["product"]["form"]['coding'][i]['system'])+","
                                        i += 1
                                code += "]'"
                                display += "]'"
                                system += "]'"
        productInsert = "INSERT INTO Product VALUES
("+medID+","+encodedText+","+code+","+display+","+system+");"
        cur.execute(productInsert)
        con.commit();
        cur.execute("SELECT * FROM Product;")
        print "\n\n\n\nPrinting out the complete list of Products!"
        print cur.fetchall()
        if 'ingredient' in data["product"]:
                parseIngredient(cur,con,data,medID)
        if 'batch' in data["product"]:
                parseBatch(cur,con,data,medID)
        return

def parseIngredient(cur,con,data,medID):
        itemDisplay = 'NULL'
        ammountType = 'NULL'
        ammountValue = 'NULL'
```

```
                ammountUnit = 'NULL'
                ammountSystem = 'NULL'
                ingredientInsert = ""
                if "item" in data["product"]["ingredient"][0]:
                        max_items = len(data["product"]["ingredient"])
                        i = 0
                        while i < max_items:
                                itemDisplay =
json.dumps(data["product"]["ingredient"][i]["item"]["display"])
                                if "amount" in data["product"]["ingredient"][i]:
                                        ammountType = "'numerator'"
                                        ammountValue ="'" +
json.dumps(data["product"]["ingredient"][i]["amount"]["numerator"]["value"])
+ "'"
                                        ammountSystem ="'" +
json.dumps(data["product"]["ingredient"][i]["amount"]["numerator"]["system"])
+ "'"
                                        ammountUnit ="'"
+json.dumps(data["product"]["ingredient"][i]["amount"]["numerator"]["code"])
+ "'"
                                        ingredientInsert = "INSERT INTO Ingredient
VALUES
("+medID+","+itemDisplay+","+ammountType+","+ammountValue+","+ammountUnit+","
+ammountSystem+");"
                                        cur.execute(ingredientInsert)
                                        con.commit();
                                        ammountType = "'denominator'"
                                        ammountValue ="'" +
json.dumps(data["product"]["ingredient"][i]["amount"]["denominator"]["value"]
) + "'"
                                        ammountSystem ="'" +
json.dumps(data["product"]["ingredient"][i]["amount"]["denominator"]["system"
]) + "'"
                                        ammountUnit ="'"
+json.dumps(data["product"]["ingredient"][i]["amount"]["denominator"]["code"]
) + "'"
                                        ingredientInsert = "INSERT INTO Ingredient
VALUES
("+medID+","+itemDisplay+","+ammountType+","+ammountValue+","+ammountUnit+","
+ammountSystem+");"
                                        cur.execute(ingredientInsert)
                                        con.commit();
                                i += 1
        cur.execute("SELECT * FROM Ingredient;")
```

```
        print "\n\n\n\nPrinting out the complete list of Ingredients!"
        print cur.fetchall()


def parseBatch(cur,con,data,medID):
        lotNumber = 'NULL'
        expirationDate = 'NULL'
        if 'lotNumber' in data["product"]["batch"][0]:
                lotNumber = "'" +
json.dumps(data["product"]["batch"][0]["lotNumber"]) + "'"
        if 'expirationDate' in data["product"]["batch"][0]:
                expirationDate = "'" +
json.dumps(data["product"]["batch"][0]["expirationDate"]) + "'"
        batchInsert = "INSERT INTO batch VALUES
("+medID+","+lotNumber+","+expirationDate+");"
        cur.execute(batchInsert)
        con.commit();
        cur.execute("SELECT * FROM Batch;")
        print "\n\n\n\nPrinting out the complete list of Batches!!"
        print cur.fetchall()


def parsePackage(cur,con,data,medID):
        text = 'NULL'
        encodedText = 'NULL'
        code = 'NULL'
        display = 'NULL'
        system = 'NULL'

        if 'container' in data["package"]:
                #For some SILLY REASON, code also apparently can have text!
                if 'text' in data["package"]["container"]:
                        text = "Coding Text: " +
json.dumps(data["package"]["container"]["coding"]['text'])
                        encodedText = "'"+str(base64.b64encode(text))+"'"
                # coding = "'"+str(data['code']['coding'])+"'"
                if 'coding' in data["package"]["container"]:
                        code = "'["
                        display = "'["
                        system = "'["
                        max_items = len(data["package"]["container"]['coding'])
                        i = 0
                        while i < max_items:
```

```
                                code +=
json.dumps(data["package"]["container"]['coding'][i]['code'])+","
                                display +=
json.dumps(data["package"]["container"]['coding'][i]['display'])+","
                                system +=
json.dumps(data["package"]["container"]['coding'][i]['system'])+","
                                i += 1
                        code += "]'"
                        display += "]'"
                        system += "]'"
        packageInsert= "INSERT INTO MedPackage VALUES
("+medID+","+text+","+code+","+display+","+system+");"
        cur.execute(packageInsert)
        con.commit();
        cur.execute("SELECT * FROM MedPackage;")
        print "\n\n\n\nPrinting out the complete list of Packages!!"
        print cur.fetchall()
        if 'content' in data["package"]:
                parsePackageContent(cur,con,data,medID)
        return
def parsePackageContent(cur,con,data,medID):
#
        itemDisplay = 'NULL'
        ammountValue = 'NULL'
        ammountSystem = 'NULL'
        ammountUnit = 'NULL'
        if "item" in data["package"]["content"][0]:
                max_items = len(data["package"]["content"])
                i = 0
                while i < max_items:
                        itemDisplay =
"'"+json.dumps(data["package"]["content"][i]["item"]["reference"])+"'"
                        if "amount" in data["package"]["content"][i]:
                                # ammountType = "'numerator'"
                                ammountValue ="'" +
json.dumps(data["package"]["content"][i]["amount"]["value"]) + "'"
                                ammountSystem ="'" +
json.dumps(data["package"]["content"][i]["amount"]["system"]) + "'"
                                ammountUnit ="'"
+json.dumps(data["package"]["content"][i]["amount"]["unit"]) + "'"
                                contentInsert = "INSERT INTO PackageContent
VALUES("+medID+","+itemDisplay+","+ammountValue+","+ammountUnit+","+ammountSy
stem+");"
                                cur.execute(contentInsert)
```

```
                                con.commit();
                        i += 1
                cur.execute("SELECT * FROM PackageContent;")
                print "\n\n\n\nPrinting out the complete list of contents!"
                print cur.fetchall()
if __name__ == "__main__":
        con = lite.connect('myDB.sqlt')

        cur = con.cursor()
        print "Showing off some of the items on our database! Hopefully it
will be empty!\n"
        cur.execute("SELECT * FROM Medication;")
        print cur.fetchall()
        cur.execute("SELECT * FROM Product;")
        print cur.fetchall()

        print "Sucessfully connected to db!"

        try:
                while(True):
                        # waits for client input:
                        print "Ok just write down the name of the json file you
want to open up!\n\n\n"
                        input_data = raw_input()
                        with open(input_data) as data_file:
                                data = json.load(data_file)
                                dataTest = json.dumps(data)
                                try:
                                        medID = parseMedication(cur,con,data)
                                        if 'product' in data:
                                                parseProduct(cur,con,data, medID)
                                        if 'package' in data:
                                                parsePackage(cur,con,data,medID)

                                except ValueError:
                                        print "Error %s:" % e.args[0]
                                        break

                                # pprint(data)
        except KeyboardInterrupt, e:
                # if CTRL+C is pressed, then go for last step
                print "\nCTRL+C - Exiting user application."
                pass
```