Database Architectures

**Practical Assessment #2 (PR2):**

**XML Extension**

***Students:*** *Carlos Del Blanco Garcia*

*Jordi Bericat Ruz*

***Professor:*** *Maria Teresa Bordas Garcia*

***Term:*** *Autumn 2020-21 (Aula 1)*

*Table of Contents*

[Activity 1 1](#_Toc55333561)

[a) 1](#_Toc55333562)

[b) 3](#_Toc55333563)

[Activity 2 7](#_Toc55333564)

[a) 7](#_Toc55333565)

[b) 7](#_Toc55333566)

[c) 7](#_Toc55333567)

[Activity 3 8](#_Toc55333568)

[a) 8](#_Toc55333569)

[b) 8](#_Toc55333570)

[Bibliography 9](#_Toc55333571)

# Activity 1

## a)

*Initial considerations:*

Before proceeding with the xml file creation itself, we though It would be a good practice to graphically model the xml-tree structure[[1]](#footnote-2). This way we can have a”one-sight” outlook of the whole xml structure and therefore make the later translation into xml code easier:



*XML Code:*

<?xml version = **"1.0"** encoding = **"ISO-8859-1"** ?>

<!-- See comments section [1] -->

<metadata xmlns=**"http://www.uoc.edu/subjects/adb/ns"**

xmlns:gencat=**"http://www.gencat.cat/dadesobertes/ns"**>

<COVID idregion=**"43"**>

<patients\_hospitalized type=**"patients"**>

<men>**17**</men>

<women>**4**</women>

<average\_age>**56**</average\_age>

</patients\_hospitalized>

<patients\_ICU type=**"patients"**>

<men>**6**</men>

<women>**2**</women>

<average\_age>**64**</average\_age>

</patients\_ICU>

<accumulated\_cases>**3947**</accumulated\_cases>

</COVID>

<COVID idregion=**"17"**>

<patients\_hospitalized type=**"patients"**>

<men>**21**</men>

<women>**6**</women>

<average\_age>**58**</average\_age>

</patients\_hospitalized>

<patients\_ICU type=**"patients"**>

<men>**7**</men>

<women>**1**</women>

<average\_age>**67**</average\_age>

</patients\_ICU>

<accumulated\_cases>**7998**</accumulated\_cases>

</COVID>

<gencat:COVID day=**"12/8/2020"**><!-- See comments section [2] -->

<![!CDATA[...]]><!-- See comments section [3] -->

</gencat:COVID>

</metadata>

<!--

**################################################################################**

**# #**

**# COMMENTS SECTION #**

**# #**

**################################################################################**

**[0] -> INITIAL CONSIDERATIONS:** In our answer, it would be great to include a PNG

showing the xml tree structure like in the documentation (Module 2, page

42) before the xml code. EDIT: I've added a first approach in this

activity folder -> PNG DRAFT DONE!

**[1] ->** I don't really know if we should use the keyword "root" instead of

"metadata" to define the xml's tree root (most likely it doesn't matter,

but better ask it in the forum to be in the safe side).

**[2] ->** Here we specify the alias "gencat" to this xml element (COVID) in order

to refer to the custom namespace provided in the activity statement. This

is required for the sake of properly integrate the external COVID xml data

into our xml, due to the fact that this external data MIGHT NOT HAVE the

very same structure than the defined in the default namespace, from which

we "import" the default predefined xml dictionary. Summing-up; this way we

avoid name clashes between the two elements with (probably) different

structure but identical identification (COVID).

**[3] ->** the structure of the external "not-xml" data is not provided, so we

cannot define it (we use the [...] placeholder instead).

**[4] ->** CDATA[...] -> maybe we have to add the "summary of the latest data"

-->

## b)

The main purpose of the extensive markup language (xml) is to stablish a proper communication mechanism among applications. To achieve this, it is necessary to strictly define a structure of elements (known as vocabulary) which implies a set rules and constraint. Here is where xml schemas come into play, since they allow defining that so-called vocabulary with a very high degree of details regarding the application data particularities.

That said, to define the xml schema that will stablish the required vocabulary and set of rules for the xml structure proposed in this activity statement, we will proceed as follows:

**<!-- ################### definition of the xml schema #################### -->**

<?xml version = **"1.0"** encoding = **"ISO-8859-1"** ?>

<!-- see comments section [0] -->

<xsd:schema><!-- see comments section [1] -->

**<!-- ################### definition of simple elements ################## -->**

<xsd:element name=**"id\_type"**><!--see comments section [2.1.3]-->

<xsd:simpleType>

<xsd:restriction base=**"xsd:positiveInteger"**>

<xsd:maxInclusive value=**"9999"**/>

</xsd:restriction>

</xsd:simpleType>

</xsd:element>

<xs:element name=**"year"** type=**"xs:integer"**/><!--see comments section [2.2.1]-->

<xs:element name=**"month"** type=**"xs:integer"**/><!--see comments section [2.2.1]-->

<xs:element name=**"day"** type=**"xs:integer"**/><!--see comments section [2.2.1]-->

<xs:element name=**"idRegion"** type=**"xs:integer"**/><!--see comments section [2.3.1]-->

<xs:element name=**"description"** type=**"xs:string"**/> <!--see comm. section [2.3.2]-->

<xs:element name=**"hospitalized"** type=**"xs:integer"**/><!--see comm. section[2.4.1]-->

<xs:element name=**"ICU"** type=**"xs:integer"**/><!-- see comments section [2.4.2]-->

<xsd:element name=**"gender"**><!--see comments section [2.4.3]-->

<xsd:simpleType>

<xsd:restriction base=**"xsd:NMTOKEN"**>

<xsd:enumeration value=**"female"** />

<xsd:enumeration value=**"male"** />

<xsd:enumeration value=**"other"** />

</xsd:restriction>

</xsd:simpleType>

</xsd:element>

**<!-- ##################### definition of attributes ##################### -->**

<xs:attribute name=**"id"** type=**"xs:id\_type"**/><!-- see comments section [2.1.3] -->

**<!-- ##################### definition of complex elements ##################### -->**

<xsd:element name=**"date"**><!-- see comments section [2.2] -->

<xsd:complexType>

<xsd:sequence>

<xs:element ref=**"year"**/>

<xs:element ref=**"month"**/>

<xs:element ref=**"day"**/>

</xsd:sequence>

</xsd:complexType>

</xsd:element>

<xsd:element name=**"region"**><!-- see comments section [2.3] -->

<xsd:complexType>

<xsd:sequence>

<xs:element ref=**"idRegion"**/>

<xs:element ref=**"description"**/>

</xsd:sequence>

</xsd:complexType>

</xsd:element>

<xsd:element name=**"patients"**><!-- see comments section [2.4] -->

<xsd:complexType>

<xsd:sequence>

<xs:element ref=**"hospitalised"**/>

<xs:element ref=**"ICU"**/>

<xs:element ref=**"gender"**/>

</xsd:sequence>

</xsd:complexType>

</xsd:element>

**<!-- ##################### root element ##################### -->**

<xs:element name=**"COVID"**><!-- see comments section [2.1.1] -->

<xs:complexType>

<xs:sequence>

<xs:element ref=**"date"**/>

<xs:element ref=**"region"**/>

<xs:element ref=**"patients"** maxOccurs=**"10"**/>

</xs:sequence>

<xs:attribute ref=**"id"** use=**"required"**/><!--comm. sections [2.1.2] & [2.1.3]-->

</xs:complexType>

</xs:element>

<!—xml schema definition end -->

</xs:schema>

<!--

**################################################################################**

**# #**

**# COMMENTS SECTION #**

**# #**

**################################################################################**

**[0] ->** First, we should decide on a design method to define the xml schema.

As seen at Bibliography [#2], there are three available approaches:

1- "Simplest-yet-messy" approach: This way, "to create the schema we could

simply follow the structure in the XML document and define each element

as we find it". (footnote: literal citation from Bibliography [#2],

section: "Create an XML Schema").

2- "Divided Schema" approach: "The next design method is based on defining

all elements and attributes first, and then referring to them using the

ref attribute" (footnote: literal citation from Bibliography [#2],

section: "Divide the Schema"). This way gets easier to read and

maintain the xml code in complex structures.

3- "Use of Named Types" approach: "The third design method defines classes

or types, that enables us to reuse element definitions"(footnote: literal

citation from Bibliography [#2], section: "Named Types").

After analyzing the characteristics of each of all three approaches, we

decided to use the 2nd one (Divide the Schema) since we won't be upgrading

the xml structure (the assessment statement says nothing about scalability

and reusability of types), but at the same time we wanted to design a

readable (it has to be assessed) as well as maintainable xml schema (so both

team members / students who participate on its elaboration can better

understand and eventually improve it).

**[1] ->** The activity statement doesn't give any information about namespaces

(xmlns), hence we neither include any reference to it in the schema root

element declaration, nor to the "targetNamespace" attribute (the XML Schema

will be assigned to the NULL namespace).

**[2] -> NOTES ABOUT ELEMENTS & ATTRIBUTES DEFINITIONS:**

**[2.1] -> NODE "COVID":**

**[2.1.1] ->** The "COVID" node will be the root element of this xml schema.

On the other hand, we'll be considering that its sub-elements will

be appearing in the same order on the instance xml documents.

Therefore, will have to use the primitive "sequence" in the

"COVID" element definition.

**[2.1.2] -> ATTRIBUTE "id" ("is mandatory" constraint):** it must be declared

as a mandatory attribute, since it contains a "foreign key" value.

To do so, we have to explicitly declare it with the "use"

attribute, as well as the "required" attribute (references:

Bibliography [#1], page 49 & Bibliography [#3]).

**[2.1.3] -> ATTRIBUTE "id" ("1-9999" range constraint):** On the other hand,

this attribute needs to be restricted to values in between 1-9999,

thus, we also might have to declare this integer attribute as a

simpleType and then apply a "range" restriction to its values by

means of the "minInclusive" & "maxInclusive" attributes

(references: Bibliography [#1], page 52).

**[2.2] -> NODE "date":** Must be a complex type defined as a SEQUENCE (references:

Bibliography [#1], page 48).

**[2.2.1] -> ELEMENTS "year", "month" & "day" (in that order):** We could define

these as simple type sub-elements (derived from integer built-in

types) of the complex type element "date". This way we make sure

these three elements contain an Integer value (and nothing else).

**[2.3] -> NODE "region**": The activity statement does not stablish any constraint

about this element, hence we can simply declare it as a complexType

with no constraints at all.

**[2.3.1] -> ELEMENT "idRegion":** "integer" built-in simple type (references:

Bibliography [#1], page 50)

**[2.3.2] -> ELEMENT "description":** "string" built-in simple type (references:

Bibliography [#1], page 50)

**[2.4] -> NODE "patients":** Since this node can only be repeated as much as 10

times, we must stablish the cardinality of this complexType element by

means of the minOccurs and maxOccurs attributes (references: module 2,

page 49).

**[2.4.1] -> ELEMENT "hospitalized":** "integer" built-in simple type (references:

page 50, module 2)

**[2.4.2] -> ELEMENT "ICU":** "integer" built-in simple type (references: page 50,

module 2)

**[2.4.3] -> ELEMENT "gender":** Must be a simple Type derived from an NTOKEN

existing built-in type, and it has to be defined as an ENUMERATION

(references: module 2 page 48)

# Activity 2

## a)

## b)

## c)

# Activity 3

## a)

## b)

# Bibliography

1. UOC Resources 🡪 Databases Architectures Module 2: Relational Extensions
2. XML Schema Example at w3.org 🡪 <https://www.w3schools.com/xml/schema_example.asp>
3. XML Attributes definition examples 🡪 <https://www.w3schools.com/xml/schema_simple_attributes.asp>

1. As suggested in bibliography [#1], page 42. [↑](#footnote-ref-2)