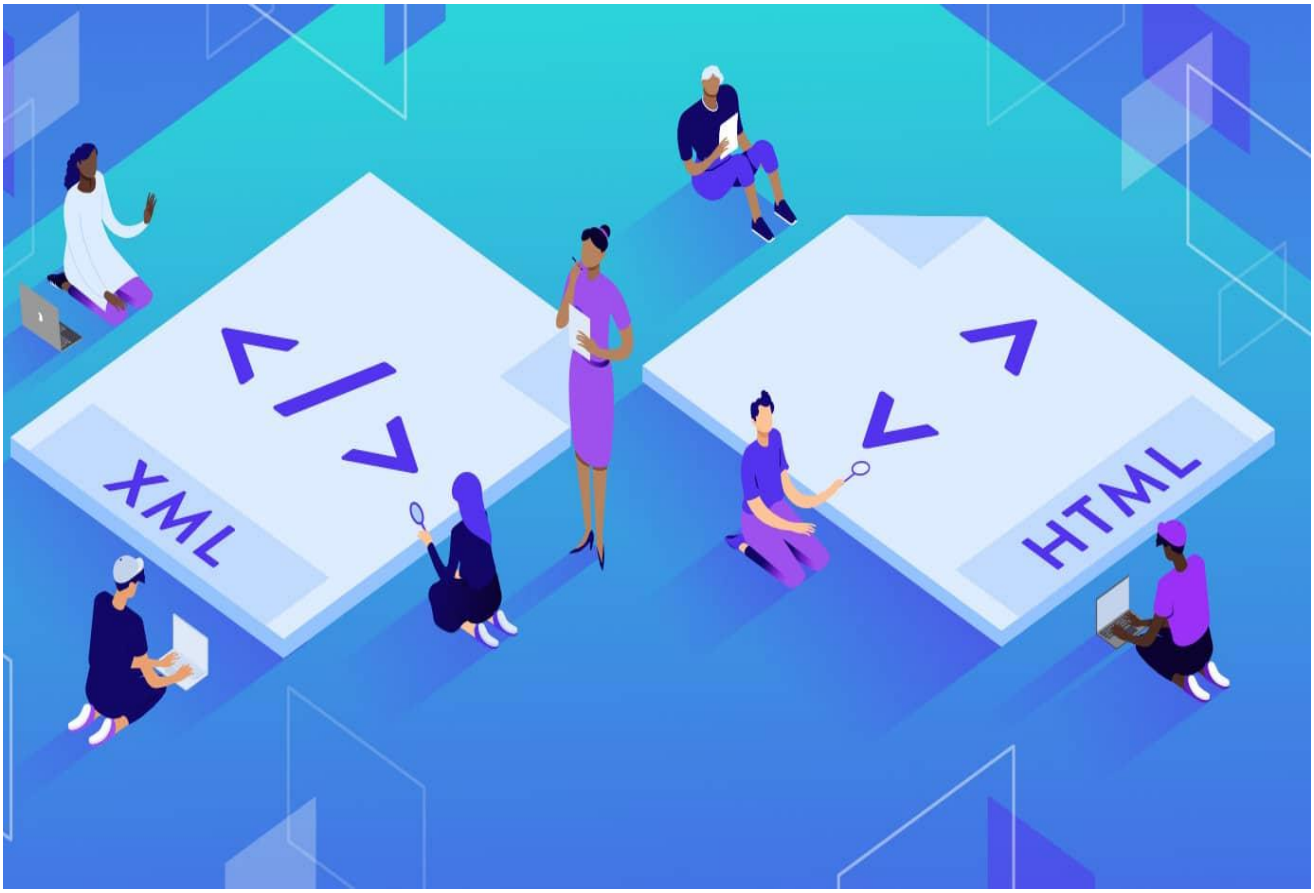
Data ScienceTech Institute

DataPipeline Part.1 – Group #2

XML Project : ESTATEPLATEFORME



Marie-Caroline Bertheau
Arnaud G
Ronald L
Damien Rondet

Introduction

In the context of enhancing our understanding of XML, we have modelled a collaborative real estate platform, stored as an xml database, which links different nature of involved entities: estate properties, professionals, prospects, sellers and lessors. Estate properties are localized in French Riviera (from Mandelieu-la-Napoule to Monaco). Our database is organized from the ESTATEPLATEFORME root, which encompasses five main sub-set (Figure 1):

- **PROSPECTS:** list of all the prospects with their contact details and search criteria registered on the platform,
- **IMMOPRODUCTS:** list of all the properties for rent or for sale with their prices and detailed descriptions,
- **SUPPLIERS:** list of professionals who can be expected to work on the properties for maintenance or technical reasons,
- **SELLERS:** list of the sellers and lessors attached to the Immoproducts,
- **TRANSACTIONS:** list of recorded transactions (rentals or sales) concluded (or in progress) between prospects and sellers.

For the goal of modelling our platform, we designed an XML schema (XSD file) which defined the structure, and the different objects contained in its database. We also took inspiration from existing estate platforms[1] .

Figure 1 | Representation of table's relations from ESTATEPLATEFORME database.

The primary and foreign keys are symbolized by ☑ and # respectively. Only principal nodes from each block are represented in this figure to keep this figure simple. We comment on all XML codes to bring the maximum information to our project.

[1] https://www.logic-immo.com/ https://www.bienici.com/ https://www.leboncoin.fr/

I- Schema part

We decided to create links between the different parts of the database using unique keys for each reference node inside each block. Only the suppliers table was not connected to others because we supposed this personal service activity proposed missions independently of the nature of real estate transactions. We generated 3 different primary keys using <xs:key> in REALESTATEPLATFORME, using key attributes of SELLERS, IMMOPRODUCTS, and PROSPECTS blocks as unique identifiers. This key creation step added a typical location in the schema, such as the REALESTATEPLATFORME/SELLERS/SELLER branch, thanks to <xs:selector>, while specifying the identifier with a unique value (for example, each seller with @SELLERref). These can then be used as foreign keys via <xs:keyref>, also placed into the REALESTATEPLATFORME root to ensure relational consistency and automatically validate the links between the various entities. For example, the XML file validation against the schema would not work if in TRANSACTIONS, there was a TRANSACTIONSELLER that would not exist in the SELLERS branch (via @SELLERref). During the execution of this step, the <xs:keyref> were a blocking point. This slowdown occurred because VS Code is much more permissive than the XML plugin for Notepad++. Indeed, VS Code can make <xs:keyref> work on its own. To avoid any errors, we chose to go with the rigidity and security of Notepad++.

Then we frequently used simple types (xs:simpleType) combined with restrictions (xs:restriction), enumerations (xs: enumeration) or even patterns (regex) to limit the possible values to precise format, to avoid typos and inconsistencies (“YES”, “Yes”, “Yess”...), to respect certain writing normalization like pattern for dates (\d{4}-\d{2}-\d{2}, compliant with ISO 8601 format YYYY-MM-DD). These instructions improve data quality and easy use during querying (transformation). Later, at query phase (request "c"), we encountered a difficulty with date formatting : keeping the regex formatting syntax would have led us to write our own date-sorting algorithm. To make the "c" request functional, we then adjusted the TRANSACTIONDATE type as "xs:date" (line 868), to take advantage of XSLT's native sorting ability. Other date fields have been left unchanged, for illustration purpose. So here we discovered that a trade-off had to be made between data control and functional usage.

The schema also defines several complex types such as ADDRESStype, CRITERIAtype or IMMOPRICEtype. They are defined by the succession of elements in the form of character strings and integers in a precise order. Then, we also created attributes (currency, surface area) to provide additional information in our database. These two complex structures make it possible to group together a logical set of related element fields under the same entity, making it easier to read and maintain the rules used in the file.

Then, we used other kinds of structures to control and constrain the validity of data. For example, min/maxOccurs tag is used to define the number of occurrences: minOccurs = “0” or “1” means that the element was optional (as “Exit”) or required at least one node (for example “Entry”); and maxOccurs = “1” or “unbounded” sets the upper limit on the number of elements (1 or infinity for respectively “Prospects” and “Entry”). The second type of

instruction was min/maxInclusive, a restriction that limited the range of numerical values, such as IMMONUMBEROFROOM in which the integer must be between 1 and 20.

Based on this schema and these database constructs, we performed a few queries using transformations in a proof-of-concept approach to test the quality of the structure, interoperability and use of the data. The results are obtained in HTML, JSON or XML files. In this way, we have queried our database using a business-oriented approach, considering both customer needs and commercial offers. This approach helps to provide structured analyses as part of a business intelligence approach.

II- Queries

a- Using the recusivity in the “Monaco suppliers query”

The XSLT code generates an HTML page with the list of suppliers (<SUPPLIER>) able to intervene (SUPPLIERZONEINTERVENTION) in the Monaco area or in a defined perimeter around Nice (up to 30 km). To do this, a filter (//SUPPLIER [SUPPLIERINTERVENTION = “MONACO”) or SUPPLIERINTERVENTION = “NICE” and (ZONEFLEXIBILITY=“UpTO20KM” or ZONEFLEXIBILITY=“UpTO30KM”) is applied to select the desired nodes.

From these, the XSLT code will be able to:

- Calculate automaticaly the total number of suppliers meeting the difined conditions by the folowwing code: <xsl:value-of select="count(//SUPPLIER[...])"/>,
- Activate recursion to generate the table using the match="/" template in HTML format (<html>, <head>, <body> tags) via the prior addition of a "Suppliers - Monaco" title (tab) and a subtitle with the total number of suppliers filtered(<h1>).

From the list of filtered suppliers (<xsl:with-param name="suppliers" .../>), the XSLT code initiates the recursion loop to fill the table. To do this we inspired the recommendations of chap 4 of the book "XSLT" by Dough Tidwell, 2001[2]. In fact, it activates the template named ‘recursive-supplier-list’, which at each iteration processes the first supplier in the list and displays it in the order in which it appears in the database. As long as there are still suppliers in the list (count($suppliers) > 0), it selected the first supplier ($suppliers[1]) to fill a row (<tr>) in the table and then process the rest of the list(‘$remaining’) as long as the condition $suppliers[position() > 1] was valid. At the end, the table contained the following attributes SUPPLIERACTIVITY (function), SUPPLIERNAME, SUPPLIERPRESTATION (work quality), SUPPLIERHOURSINTERVENTION (on-call duty), SUPPLIERMAIL, SUPPLIERTEL. We noted, in this transformation no statements enabling sorting (such as <xsl:sort>) in the XSLT code presented in this section. In fact, this function is normally used with <xsl:for-

[2] https://www.oreilly.com/library/view/xslt/0596000537/ch04s06.html

each> or <xsl:apply-templates> as indicated in the W3C specification[3]: '*The xsl:sort element can be used only within xsl:apply-templates or xsl:for-each*'. This prevents any deferred sorting strategy, particularly after recursive processing.

b- Overview on the real estate price market: Studio case without car parking

Secondly, we produced an XSLT code to have a rapid and structured macro-view of the market prices, for example studios (apartment with IMMONUMBEROFROOM=1) without parking by city. As XSLT 1.0 does not include a function for direct grouping (as group by in SQL), we used Muenchian's method (named after Steve Muench) as described in chapter 10 of the book "XSLT Cookbook" (2nd edition, Sal Mangano, 2005[4]). Briefly, this approach relies on <xsl:key> and generate-id() to extract the unique cities from which, the average price is calculated using the 'calculate-average' template by using sum() and count() functions previously defined. The output is structured with <City> and <Average_Price> tags for each city, with the price displayed in decimal format in XML format file.

c- XML to HTML : Rentings by Customer and descending Transaction date

As described below, we used again the Muenchian's method to have the transaction history of our platform. For this, we summarized in descending order, all past renting transactions by customer (PROSPECT object) and by date, including the relevant amount (euros) and the reference of the house/apartment (IMMOPRODUCT). To link this information, we used two "for-each" nested loops, one for TRANSACTION and the second for PROSPECT elements. In parallel, Muenchian's method allowed to group the transactions per prospect and determined the count of commercial operations for each targeted user for specific criteria.

d- XML to HTML : Matching PROSPECTS and IMMOPRODUCTS for a rent in NICE

In the previous scenario, we traced the transactional history of our database. Here we have built a query to correlate available offers with future customer requests to predict possibilities for rents in Nice. We compared prospects and estate properties by using this criteria "BUDGET >= IMMOPRICE". We wrote the code to find all matches between PROSPECT and IMMOPRODUCTS elements, without "for-each" loop. Instead, two XSL templates are used successively on each group.

[3] https://www.w3.org/TR/xslt.html

[4] https://learning.oreilly.com/library/view/xslt-cookbook-2nd/0596009747/ch10.html#xsltckbk2-CHP-10-SECT-3

e- XML request using Python

We performed a query using a python code to extract, from the XML file REALESTATEPLATEFORM, the list of suppliers working on hourly on-call duty and automatically publish it on an HTML page. We used the ElementTree module (xml.etree.ElementTree), which is a library integrated into Python. In this context, we use it to parse (ET.parse()) and manipulate the XML file. It also allows us to navigate the entire XML structure via the root element (getroot()), target <SUPPLIER> tags, and extract the relevant information using find() and findall().

f- XML to HTML : SellerInformation

The objective of this query is to be able to retrieve multiple pieces of information about each seller. To avoid null values or presentation issues, it was important to use the <xsl:if> command. For example, if the "Number of Listed Properties" is equal to 0, there is no need to provide further details about the type of properties.

g- XML to JSON: Transactions Status

The entire exercise of this scenario was to respect the structure of a JSON format. To adhere to this format, the entire output, our object, must be enclosed in curly braces {}. Then, for each key, a value must be associated, which can be a simple element (e.g., TotalTransactionAmount expecting a number) or a list (e.g., ProspectsWithTransactions expecting a list of IDs, or DetailedProspects which will take a list of objects {}). Therefore, in the JSON, we find a summary providing an overall status of transactions, followed by a list detailing all the prospects who have completed a transaction.

Conclusion

From our work, we designed a structured database validated by the syntax and the schema. To verify the efficiency of our XML code, we did queries with business contents. Following this project, we were able to draw comparisons with other tools (such as R and SQL) that seemed easier to use.

This led us to raise the following difficulties:

- data matching (join),
- sorting,
- Grouping.

XML's lack of flexibility becomes an advantage when it comes to exchange (serialization/deserialization) data between heterogeneous systems: XML provides great data control, produces documents that are both human-readable and machine-readable.