

# Données du Web - TD3 - XML-Relationnel

Basil Dalié - Yannis Naidja

Octobre 2019

# Chapitre 1

## Stockage schema-unaware : Verical-Edge vs Monet

- 1.1 Considérons le document XML pour la presse que vous avez proposé en réponse à la question1du TD1
- 1.2 À l'aide du langage SQL, implémenter les schémas de stockage Vertical-Edge et Monet associés au document. Ensuite, peupler les tables correspondantes

Voir document XML presse en annexe.

### 1.2.1 Vertical-Edge

```
1 CREATE TABLE PRESSE (  
2     source int,  
3     target int NOT NULL PRIMARY KEY,  
4     ordinal int,  
5     txtval varchar(30),  
6     numval int  
7 );  
8  
9 CREATE TABLE JOURNAL (  
10    source int,  
11    target int NOT NULL PRIMARY KEY,  
12    ordinal int,
```

```

13         txtval varchar(30),
14         numval int
15     );
16
17 CREATE TABLE ARTICLE (
18     source int,
19     target int NOT NULL PRIMARY KEY,
20     ordinal int,
21     txtval varchar(30),
22     numval int
23 );
24 CREATE TABLE TITRE (
25     source int,
26     target int NOT NULL PRIMARY KEY,
27     ordinal int,
28     txtval varchar(30),
29     numval int
30 );
31
32 CREATE TABLE AUTEUR (
33     source int,
34     target int NOT NULL PRIMARY KEY,
35     ordinal int,
36     txtval varchar(30),
37     numval int
38 );
39
40 CREATE TABLE CORPS (
41     source int,
42     target int NOT NULL PRIMARY KEY,
43     ordinal int,
44     txtval varchar(30),
45     numval int
46 );
47 CREATE TABLE JOURNALISTES (
48     source int,
49     target int NOT NULL PRIMARY KEY,
50     ordinal int,
51     txtval varchar(30),
52     numval int
53 );
54 CREATE TABLE JOURNALISTE (
55     source int,
56     target int NOT NULL PRIMARY KEY,
57     ordinal int,
58     txtval varchar(30),

```

```

59         numval int
60     );
61 CREATE TABLE JOURNALISTEID (
62     source int,
63     target int NOT NULL PRIMARY KEY,
64     ordinal int,
65     txtval varchar(30),
66     numval int
67 );
68
69 CREATE TABLE ANONYMOUS (
70     source int,
71     target int NOT NULL PRIMARY KEY,
72     ordinal int,
73     txtval varchar(30),
74     numval int
75 );
76 CREATE TABLE PSEUDO (
77     source int,
78     target int NOT NULL PRIMARY KEY,
79     ordinal int,
80     txtval varchar(30),
81     numval int
82 );
83
84 CREATE TABLE NOM (
85     source int,
86     target int NOT NULL PRIMARY KEY,
87     ordinal int,
88     txtval varchar(30),
89     numval int
90 );
91
92 CREATE TABLE PRENOM (
93     source int,
94     target int NOT NULL PRIMARY KEY,
95     ordinal int,
96     txtval varchar(30),
97     numval int
98 );
99
100 CREATE TABLE DIRECTEUR (
101     source int,
102     target int NOT NULL PRIMARY KEY,
103     ordinal int,
104     txtval varchar(30),

```

```

105         numval int
106     );
107
108     INSERT INTO PRESSE (target) VALUES
109         ↪ (0);
110     INSERT INTO JOURNAL (source, target, ordinal) VALUES (0,
111         ↪ 1, 1);
112     INSERT INTO NOM (source, target, ordinal, txtval) VALUES (1,
113         ↪ 2, 1, 'CNEWS');
114     INSERT INTO DIRECTEUR (source, target, ordinal) VALUES (1,
115         ↪ 3, 1);
116     INSERT INTO NOM (source, target, ordinal, txtval) VALUES (3,
117         ↪ 4, 1, 'Pepega');
118     INSERT INTO PRENOM (source, target, ordinal, txtval) VALUES (3,
119         ↪ 5, 2, 'Kekw');
120     INSERT INTO ARTICLE (source, target, ordinal) VALUES (1,
121         ↪ 7, 1);
122     INSERT INTO CORPS (source, target, ordinal, txtval) VALUES (7,
123         ↪ 10, 1, 'Des fake news');
124     INSERT INTO TITRE (source, target, txtval) VALUES (7,
125         ↪ 8, 'fake');
126     INSERT INTO AUTEUR(source, target, txtval) VALUES (7,
127         ↪ 9, 'j1');
128     INSERT INTO ARTICLE (source, target, ordinal) VALUES (1,
129         ↪ 11, 1);
130     INSERT INTO CORPS (source, target, ordinal, txtval) VALUES
131         ↪ (11, 14, 1, ' Encore des fake news');
132     INSERT INTO TITRE (source, target, txtval) VALUES
133         ↪ (11, 12, 'news');
134     INSERT INTO AUTEUR(source, target, txtval) VALUES
135         ↪ (11, 13, 'j1');
136     INSERT INTO JOURNALISTES (source, target, ordinal) VALUES (0,
137         ↪ 15, 2);
138     INSERT INTO JOURNALISTE (source, target, ordinal) VALUES
139         ↪ (15, 16, 1);
140     INSERT INTO JOURNALISTEID (source, target, txtval) VALUES
141         ↪ (16, 19, 'j1');
142     INSERT INTO NOM (source, target, ordinal, txtval) VALUES
143         ↪ (16, 17, 1, 'Vuillard');
144     INSERT INTO PRENOM (source, target, ordinal, txtval) VALUES
145         ↪ (16, 18, 1, 'Eric');
146     INSERT INTO JOURNALISTE (source, target, ordinal) VALUES
147         ↪ (15, 20, 2);
148     INSERT INTO JOURNALISTEID (source, target, txtval) VALUES
149         ↪ (20, 23, 'j2');

```

```

129 INSERT INTO NOM (source, target, ordinal, txtval)      VALUES
    ↪ (20, 21, 1, 'Dupont');
130 INSERT INTO PRENOM (source, target, ordinal, txtval)  VALUES
    ↪ (20, 22, 1, 'Jean');
131 INSERT INTO ANONYMOUS(source, target, ordinal, txtval) VALUES
    ↪ (20, 24, 1, 'non');

```

### 1.2.2 Monet

```

1 CREATE TABLE PRESSE (
2     node int,
3     txtval varchar(30),
4     numval int
5 );
6 CREATE TABLE PRESSE_JOURNAL (
7     node int,
8     txtval varchar(30),
9     numval int
10 );
11
12 CREATE TABLE PRESSE_JOURNAL_NOM (
13     node int,
14     txtval varchar(30),
15     numval int
16 );
17 CREATE TABLE PRESSE_JOURNAL_DIRECTEUR (
18     node int,
19     txtval varchar(30),
20     numval int
21 );
22
23 CREATE TABLE PRESSE_JOURNAL_DIRECTEUR_NOM (
24     node int,
25     txtval varchar(30),
26     numval int
27 );
28
29 CREATE TABLE PRESSE_JOURNAL_DIRECTEUR_PRENOM (
30     node int,
31     txtval varchar(30),
32     numval int
33 );
34
35 CREATE TABLE PRESSE_JOURNAL_ARTICLE (
36     node int,
37     txtval varchar(30),

```

```

38         numval int
39     );
40
41     CREATE TABLE PRESSE_JOURNAL_ARTICLE_CORPS (
42         node int,
43         txtval varchar(30),
44         numval int
45     );
46
47     CREATE TABLE PRESSE_JOURNAL_ARTICLE_TITRE (
48         node int,
49         txtval varchar(30),
50         numval int
51     );
52
53     CREATE TABLE PRESSE_JOURNAL_ARTICLE_AUTEUR (
54         node int,
55         txtval varchar(30),
56         numval int
57     );
58
59     CREATE TABLE PRESSE_JOURNALISTES (
60         node int,
61         txtval varchar(30),
62         numval int
63     );
64
65     CREATE TABLE PRESSE_JOURNALISTES_JOURNALISTE (
66         node int,
67         txtval varchar(30),
68         numval int
69     );
70
71     CREATE TABLE PRESSE_JOURNALISTES_JOURNALISTE_NOM (
72         node int,
73         txtval varchar(30),
74         numval int
75     );
76
77     CREATE TABLE PRESSE_JOURNALISTES_JOURNALISTE_PRENOM (
78         node int,
79         txtval varchar(30),
80         numval int
81     );
82
83     CREATE TABLE PRESSE_JOURNALISTES_JOURNALISTE_IDJ (

```

```

84         node int,
85         txtval varchar(30),
86         numval int
87     );
88
89     CREATE TABLE PRESSE_JOURNALISTES_JOURNALISTE_ANONYMISATION (
90         node int,
91         txtval varchar(30),
92         numval int
93     );
94
95     INSERT INTO PRESSE(node)
96     ↪ VALUES(0);
97     INSERT INTO PRESSE_JOURNAL(node)
98     ↪ VALUES(1);
99     INSERT INTO PRESSE_JOURNAL_NOM(node,txtval)
100    ↪ VALUES(2,'CNEWS');
101    INSERT INTO PRESSE_JOURNAL_DIRECTEUR(node)
102    ↪ VALUES(3);
103    INSERT INTO PRESSE_JOURNAL_DIRECTEUR_NOM(node,txtval)
104    ↪ VALUES(4,'Pepega');
105    INSERT INTO PRESSE_JOURNAL_DIRECTEUR_PRENOM(node,txtval)
106    ↪ VALUES(5,'KEKW');
107    INSERT INTO PRESSE_JOURNAL_ARTICLE(node)
108    ↪ VALUES(7);
109    INSERT INTO PRESSE_JOURNAL_ARTICLE_TITRE(node, txtval)
110    ↪ VALUES(8, 'fake');
111    INSERT INTO PRESSE_JOURNAL_ARTICLE_AUTEUR(node, txtval)
112    ↪ VALUES(9, 'j1');
113    INSERT INTO PRESSE_JOURNAL_ARTICLE_CORPS(node,txtval)
114    ↪ VALUES (10,'fakenews');
115    INSERT INTO PRESSE_JOURNAL_ARTICLE(node)
116    ↪ VALUES(11);
117    INSERT INTO PRESSE_JOURNAL_ARTICLE_TITRE(node, txtval)
118    ↪ VALUES(12, 'news');
119    INSERT INTO PRESSE_JOURNAL_ARTICLE_AUTEUR(node, txtval)
120    ↪ VALUES(13, 'j1');
121    INSERT INTO PRESSE_JOURNAL_ARTICLE_CORPS(node,txtval)
122    ↪ VALUES (14,' more fakenews');
123    INSERT INTO PRESSE_JOURNALISTES(node)
124    ↪ VALUES (15);
125    INSERT INTO PRESSE_JOURNALISTES_JOURNALISTE(node)
126    ↪ VALUES (16);
127    INSERT INTO PRESSE_JOURNALISTES_JOURNALISTE_NOM(node,txtval)
128    ↪ VALUES (17,'Vuillard');

```



```

112 INSERT INTO PRESSE_JOURNALISTES_JOURNALISTE_PRENOM(node,txtval)
    ↪ VALUES (18,'Eric');
113 INSERT INTO PRESSE_JOURNALISTES_JOURNALISTE_IDJ(node,txtval)
    ↪ VALUES (19, 'j1');
114 INSERT INTO PRESSE_JOURNALISTES_JOURNALISTE(node)
    ↪ VALUES (20);
115 INSERT INTO PRESSE_JOURNALISTES_JOURNALISTE_NOM(node,txtval)
    ↪ VALUES (21,'Dupont');
116 INSERT INTO PRESSE_JOURNALISTES_JOURNALISTE_PRENOM(node,txtval)
    ↪ VALUES (22,'Jean');
117 INSERT INTO PRESSE_JOURNALISTES_JOURNALISTE_IDJ(node,txtval)
    ↪ VALUES (23, 'j2');
118 INSERT INTO
    ↪ PRESSE_JOURNALISTES_JOURNALISTE_ANONYMISATION(node,txtval)
    ↪ VALUES (24, 'oui');

```

## 1.3 À l'aide du langage SQL, exprimer cinq requêtes XPath de votre choix sur chaque schéma

### 1.3.1 Le nom du journal

```
1 /presse/journal/nom
```

#### Vertical-Edge

```

1 SELECT N.txtval nom_journal
2 FROM PRESSE P,
3      JOURNAL J,
4      NOM      N
5 WHERE
6      P.Target = J.Source
7 AND   J.Target = N.Source;

```

#### Monet

```

1 SELECT txtval nom_journal
2 FROM   PRESSE_JOURNAL_NOM;

```

### 1.3.2 Le nom du directeur

```
1 /presse/journal/directeur/nom
```

#### Vertical-Edge

```

1 SELECT N.txtval nom_directeur
2 FROM   PRESSE      P,

```

```

3      JOURNAL    J,
4      DIRECTEUR D,
5      NOM        N
6 WHERE
7      P.Target = J.Source
8 AND   J.Target = D.Source
9 AND   D.Target = N.Source;

```

#### Monet

```

1 SELECT txtval nom_directeur
2 FROM PRESSE_JOURNAL_DIRECTEUR_NOM;

```

### 1.3.3 Le titre de tout les articles

```

1 /presse/journal/article/@titre

```

#### Vertical-Edge

```

1 SELECT T.txtval titre_article
2 FROM PRESSE P,
3      JOURNAL J,
4      ARTICLE A,
5      TITRE T
6 WHERE P.Target = J.Source
7 AND   J.Target = A.Source
8 AND   A.Target = T.Source;

```

#### Monet

```

1 SELECT txtval titre_article
2 FROM PRESSE_JOURNAL_ARTICLE_TITRE;

```

### 1.3.4 Le nombre d article

```

1 /presse/journal/count(article)

```

#### Vertical-Edge

```

1 SELECT COUNT(A.Target) nombre_d_article
2 FROM PRESSE P,
3      JOURNAL J,
4      ARTICLE A
5 WHERE P.Target = J.Source
6 AND   J.Target = A.Source;

```

## Monet

```
1 SELECT COUNT(A.node) nombre_d_article
2 FROM PRESSE_JOURNAL_ARTICLE A;
```

### 1.3.5 Les identifiants des journalistes

```
1 /presse/journal/journalistes/journaliste/@idJ
```

## Vertical-Edge

```
1 SELECT J3.txtval identifiant_de_journaliste
2 FROM PRESSE P,
3 JOURNALISTES J1,
4 JOURNALISTE J2,
5 JOURNALISTEID J3
6 WHERE P.Target = J1.Source
7 AND J1.Target = J2.Source
8 AND J2.Target = J3.Source;
```

## Monet

```
1 SELECT txtval identifiant_de_journaliste
2 FROM PRESSE_JOURNALISTES_JOURNALISTE_IDJ;
```

## Chapitre 2

# Stockage schema-aware : Verical-Edge vs Monet

2.1 A partir de la DTD pour les batiments presentee dans l'enonce du TD1, definir un schema de stockagerelationnel suivant la methode presentee en cours.

### 2.1.1 DTD batiment

```
1 <!DOCTYPE batiment [  
2 <!ELEMENT batiment (etage)+ >  
3 <!ELEMENT etage (description,(bureau+|salle+)) >  
4 <!ELEMENT description (#PCDATA) >  
5 <!ELEMENT bureau (code, personne*) >  
6 <!ELEMENT code (#PCDATA) >  
7 <!ELEMENT personne (#PCDATA) >  
8 <!ELEMENT salle (nombrePlaces) >  
9 <!ELEMENT nombrePlaces (#PCDATA) >]>
```

### 2.1.2 Suppression des symboles +

```
1 <!DOCTYPE batiment [  
2 <!ELEMENT batiment (etage*,etage) >  
3 <!ELEMENT etage (description,((bureau*,bureau)|(salle*,salle)) >  
4 <!ELEMENT description (#PCDATA) >  
5 <!ELEMENT bureau (code, personne*) >  
6 <!ELEMENT code (#PCDATA) >  
7 <!ELEMENT personne (#PCDATA) >  
8 <!ELEMENT salle (nombrePlaces) >
```

```
9 <!ELEMENT nombrePlaces (#PCDATA) >]>
```

### 2.1.3 Suppression de l'ordre et des correlations

```
1 <!DOCTYPE batiment [  
2 <!ELEMENT batiment (etage* | etage) >  
3 <!ELEMENT etage (description | bureau* | bureau | salle* |  
  ↪  salle)) >  
4 <!ELEMENT description (#PCDATA) >  
5 <!ELEMENT bureau (code | personne*) >  
6 <!ELEMENT code (#PCDATA) >  
7 <!ELEMENT personne (#PCDATA) >  
8 <!ELEMENT salle (nombrePlaces) >  
9 <!ELEMENT nombrePlaces (#PCDATA) >]>
```

### 2.1.4 Simplifications

$r \mid r^*$  est equivalent a  $r^*$

```
1 <!DOCTYPE batiment [  
2 <!ELEMENT batiment (etage*) >  
3 <!ELEMENT etage (description | bureau* | salle*) >  
4 <!ELEMENT description (#PCDATA) >  
5 <!ELEMENT bureau (code | personne*) >  
6 <!ELEMENT code (#PCDATA) >  
7 <!ELEMENT personne (#PCDATA) >  
8 <!ELEMENT salle (nombrePlaces) >  
9 <!ELEMENT nombrePlaces (#PCDATA) >]>
```

### 2.1.5 Representation sous forme de graphe

Voir graphe en annexe

### 2.1.6 Relations

- batiment(batimentID : integer, flagRoot : integer)
- etage(etageID : integer, batimentID : integer, description : string)
- bureau(bureauID : integer, etageID : integer, code : string)
- personne(personneID : integer, bureauID : integer)
- salle(salleID : integer, etageID : integer, nombreDePlace : integer)

### 2.1.7 Creation des tables

```
1 CREATE TABLE BATIMENT (  
2     batimentID int NOT NULL PRIMARY KEY,  
3     flagRoot int NOT NULL  
4 );
```

```

5
6 CREATE TABLE ETAGE (
7     etageID int NOT NULL PRIMARY KEY,
8     batimentID int NOT NULL,
9     description varchar(30),
10    CONSTRAINT fk_batimentID
11    FOREIGN KEY (batimentID)
12    REFERENCES BATIMENT (batimentID)
13 );
14
15 CREATE TABLE BUREAU (
16     bureauID int NOT NULL PRIMARY KEY,
17     etageID int NOT NULL,
18     code varchar(10),
19    CONSTRAINT fk_bureau_etageID
20    FOREIGN KEY (etageID)
21    REFERENCES ETAGE (etageID)
22 );
23
24 CREATE TABLE PERSONNE (
25     personneID int NOT NULL PRIMARY KEY,
26     bureauID int NOT NULL,
27    CONSTRAINT fk_bureauID
28    FOREIGN KEY (bureauID)
29    REFERENCES BUREAU (bureauID)
30 );
31
32 CREATE TABLE SALLE (
33     salleID int NOT NULL PRIMARY KEY,
34     etageID int NOT NULL,
35     nombreDePlace int,
36    CONSTRAINT fk_salle_etageID
37    FOREIGN KEY (etageID)
38    REFERENCES ETAGE (etageID)
39 );

```

## 2.2 Peupler les tables avec des lignes correspondants au document XML que vous propose en reponse a laquestion 1 du TD1

```

1 INSERT INTO BATIMENT (batimentID, flagRoot) VALUES (0,
  ↳ 1);
2 INSERT INTO ETAGE (etageID, batimentID, description) VALUES (0,
  ↳ 0, 'descriptionETAGE1');

```

```

3 INSERT INTO ETAGE (etageID, batimentID, description) VALUES (1,
  ↳ 0, 'descriptionETAGE2');
4 INSERT INTO SALLE (salleID, etageID, nombreDePlace) VALUES (0,
  ↳ 0, 25);
5 INSERT INTO SALLE (salleID, etageID, nombreDePlace) VALUES (1,
  ↳ 0, 50);
6 INSERT INTO BUREAU (bureauID, etageID, code) VALUES (0,
  ↳ 1, 'B02');
7 INSERT INTO PERSONNE (personneID, bureauID) VALUES (0,
  ↳ 0);
8 INSERT INTO BUREAU (bureauID, etageID, code) VALUES (1,
  ↳ 1, 'B03');
9 INSERT INTO PERSONNE (personneID, bureauID) VALUES (1,
  ↳ 1);
10 INSERT INTO PERSONNE (personneID, bureauID) VALUES (2,
  ↳ 1);

```

## 2.3 A l'aide du langage SQL, exprimer cinq requêtes XPath de votre choix

### 2.3.1 Les descriptions d'étage du bâtiment d'identifiant 0

```
1 //batiment[@id=0]/description/text()
```

#### Requête SQL

```

1 SELECT DESCRIPTION
2 FROM ETAGE
3 WHERE batimentID = 0;

```

### 2.3.2 Le nombre de place des salles de l'étage d'identifiant 0 du bâtiment d'identifiant 0

```
1 //batiment[@id=0]/etage[@id=0]/salle/nombredeplace
```

#### Requête SQL

```

1 SELECT S.NOMBREDEPLACE
2 FROM SALLE S JOIN ETAGE E ON S.etageID = E.etageID
3 WHERE E.batimentID = 0 AND E.etageID = 0;

```

### 2.3.3 Le nombre de personne travaillant dans des bureaux

```
1 count(//batiment/etage/bureau/personne)
```

#### Requête SQL

```
1 SELECT count(*)
2 FROM personne;
```

#### 2.3.4 Le code des bureaux de l etage 1 du batiment 0

```
1 //batiment[@id=0]/etage[@id=1]/bureau/@code
```

#### Requête SQL

```
1 SELECT B.code
2 FROM BUREAU B JOIN ETAGE E ON B.etageID = E.etageID
3 WHERE E.batimentID = 0 AND E.etageID = 1;
```

#### 2.3.5 Le nombre d'etage du batiment 0

```
1 count(//batiment[@id=0]/etage)
```

#### Requête SQL

```
1 SELECT COUNT(*)
2 FROM ETAGE
3 WHERE batimentID = 0;
```





## Chapitre 3

# Interval-encoding avec SAX

### 3.1 Illustrer l’encodage (1) begin/end et (2) Dewey de l’XML du document proposé pour les bâtiments

#### 3.1.1 BeginEnd

Voir arbre et graphe begin/end en annexe

#### 3.1.2 Dewey

Voir arbre Dewey en annexe

## Annexe A

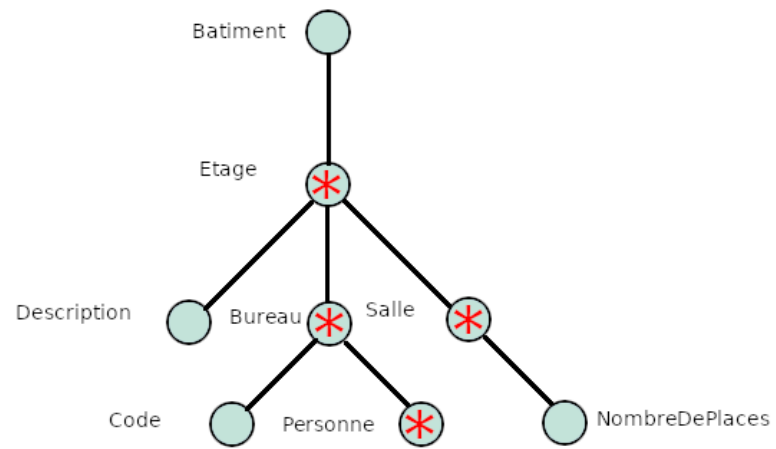
# XML - Presse

```
1  <presse>
2    <journal>
3      <nom>
4        Cnews
5      </nom>
6      <directeur>
7        <nom>
8          Pepega
9        </nom>
10       <prenom>
11         Kekw
12       </prenom>
13     </directeur>
14     <article titre="fake" auteur="j1">
15       <corps>
16         Des fake news
17       </corps>
18     </article>
19     <article titre="news" auteur="j1">
20       <corps>
21         Encore des fake news
22       </corps>
23     </article>
24   </journal>
25   <journalistes>
26     <journaliste idJ="j1">
27       <nom>
28         vuillard
29       </nom>
30       <prenom>
31         eric
```

```
32         </prenom>
33     </journaliste>
34     <journaliste idJ="j2" anonymisation="oui">
35         <nom>
36             Dupont
37         </nom>
38         <prenom>
39             Jean
40         </prenom>
41     </journaliste>
42 </journalistes>
43 </presse>
```

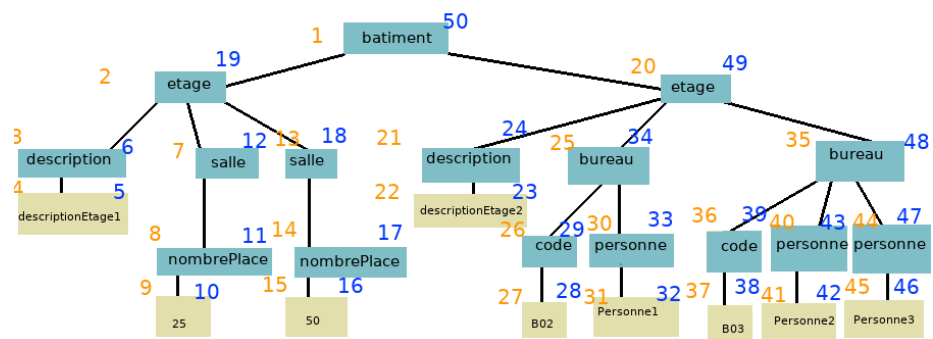
## Annexe B

# Representation sous forme de graphe



## Annexe C

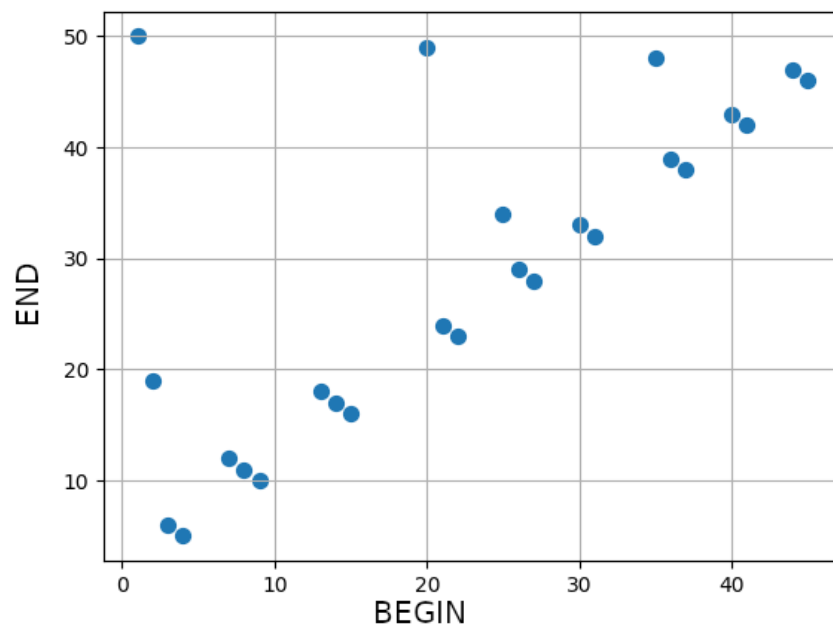
### Arbre begin/end pour le document batiment





## Annexe D

### Graphe begin/end pour le document batiment



## Annexe E

# Arbre Dewey pour le document bâtiment

