# XML - Semaine 5

Pierre Nerzic

février-mars 2022

Le cours de cette semaine présente :

- XQuery qui est une extension de XPath,
- les bases de données XML et leur interrogation avec XQuery.

XQuery est un sur-ensemble de XPath qui permet de faire de très nombreuses choses avec un document XML : en extraire des informations, le reformater et même le modifier.

On ne présentera ici que les bases fondamentales, mais les possibilités sont immenses.


Pierre Nerzic

# XQuery


Pierre Nerzic

# Présentation

XQuery est un langage permettant de traiter un document XML. Comme XSLT, il produit un document en sortie. Les instructions XQuery se placent au milieu d'une sorte de modèle XML. En sortie, les instructions XQuery sont remplacées par ce qu'elles calculent.

Les différences avec XSLT sont dans la syntaxe, les requêtes XQuery ne sont pas en XML[1]. D'autre part la norme XPath utilisée est la version 2.0 qui offre des possibilités supplémentaires.

XQuery permet d'exprimer des requêtes beaucoup plus complexes que XPath version 1.0. Il permet de faire des sortes de jointures. De fait, XQuery est aux *bases de données XML* ce que SQL est aux SGBD relationnels.

[1] En fait, si ! Il existe une syntaxe XML mais très peu lisible.

# Exemple initial

Pour un aperçu de ce qu'est XQuery, voici une feuille pour afficher albums.xml en HTML : 

```
<html><body>
<h2>Albums de Tintin</h2>
<table>{
   for $album in doc("albums.xml")//album[@serie="Tintin"]
   return
    <tr>
      <td>{$album/titre/text()}</td>
      <td>{$album//mois/text()}{" "}{$album//annee/text()}</td>
    </tr>
}</table>
</body></html>
```


Pierre Nerzic

# Traitement d'une feuille XQuery

Pour lancer le traitement en ligne de commande, on peut employer un outil appelé `galax-run`.

```
galax-run entrée > sortie
```

Par exemple : `galax-run albums.xq > albums.html`

Un autre logiciel, beaucoup plus puissant s'appelle `BaseX`, décrit sur la page wikipedia. Il permet de gérer une base de données XML et de l'interroger avec des requêtes XQuery. Voir la deuxième partie de ce cours.

# Bases de XQuery

XPath est un sous-ensemble de XQuery. C'est à dire que XQuery exécute n'importe quelle requête XPath directement.

Exemple de requête XQuery XPath sur le document XML : 

```
//album[@serie="Tintin" and date/mois="août"]/titre
```

Dans la suite, vous verrez qu'on met le mot clé `return` quand on veut générer quelque chose de différent du document d'origine. Il ne faut pas le mettre quand c'est une simple requête XPath.


Pierre Nerzic

# Scripts XQuery

En général, une requête XQuery se place dans un fichier source `requete.xq`. Ce fichier contient, soit uniquement une requête, soit du code XML dans lequel il y a des requêtes placées entre `{...}`.

Exemple de source : 

```
<html><body lang="fr">
Il y a { count( doc("albums.xml")/albums/album ) } albums.
</body></html>
```

- La fonction `doc("albums.xml")` retourne le document XML ; on lui applique une requête XPath.
- La fonction `count(collection)` compte le nombre de nœuds de la collection. Cette collection provient de l'expression XPath `/albums/album` appliquée au document `albums.xml`.

# Génération d'éléments XML

Pour produire des éléments ou des attributs en sortie, il y a une syntaxe ressemblant à RelaxNG :

- `element NOM { CONTENU1, CONTENU2, ... }`
- `attribute NOM { VALEUR }`

Exemple de script XQuery qui retourne exactement le même résultat que l'exemple précédent :

```
element html {
    element body {
        attribute lang { "fr" },
        "Il y a",
        count(doc("albums.xml")/albums/album),
        "albums."
    }
}
```


Pierre Nerzic

# Affectation de variables

XQuery permet de définir des variables. La syntaxe est :

```
let $NOM := VALEUR
return SORTIE
```

- La valeur est une expression XPath. Notez le $ devant les variables (comme en PHP).
- `return` indique ce qu'il faut produire en sortie.

Cet exemple est une variante des précédents : 

```
let $nombre := count( doc("albums.xml")/albums/album )
return <html><body>Il y a { $nombre } albums.</body></html>
```

Notez les `{...}` pour délimiter du code XQuery dans la clause `return`. Attention, il n'y a **pas de ;** à la fin du `let`.


Pierre Nerzic

# Affectations multiples

On peut faire plusieurs affectations successives, liées ou non : 

```
let $albums := doc("albums.xml")/albums/album
let $nombre := count( $albums )
let $min_annee := min( $albums/date/annee )
let $max_annee := max( $albums/date/annee )
return
    element html {
        element body {
            "Il y a", $nombre, "albums de",
            $min_annee, "à", $max_annee
        }
    }
```

Attention à ne pas mettre de ; dans cette requête. En fait, tout cela n'est qu'une seule instruction XQuery.

# Conditionnelles

XPath 2.0 fournit une structure conditionnelle :

```
if (CONDITION) then EXPR1 else EXPR2
```

C'est une expression dont la valeur est soit EXPR1, soit EXPR2.

Exemple : 

```
let $nombre := count( doc("albums.xml")/albums/album )
return <html><body>Il y a {
    if ($nombre > 20) then "de nombreux" else $nombre
} albums.</body></html>
```

Remarquez bien les mots clés `then` et `else` et ne les confondez pas avec les { des langages C et Java. Ici, les { et } permettent de passer de l'espace XML à l'espace XQuery.


Pierre Nerzic

# Conditionnelles (suite)

Du fait que ce soit une expression, on **ne peut pas** l'employer ainsi :

```
let $nombre := count( doc("albums.xml")/albums/album )
if ($nombre > 20) then return "beaucoup" else return "peu"
```

Par contre, il y a une structure pour cela :

```
let AFFECTATION where (CONDITION) return SORTIE
```

Exemple : 

```
let $nombre := count( doc("albums.xml")/albums/album )
where ($nombre > 20)
return <html><body>Il y a de nombreux albums.</body></html>
```

Cependant, il n'y a pas de clause `else` possible.

# Boucles

La puissance de XQuery vient des boucles `for`. Le schéma général est appelé FLWOR (*For Let Where OrderBy Return*) (et non pas FLOWR), dont voici le plus simple (les clauses sont optionnelles) :

```
for VARIABLE in COLLECTION
  return SORTIE
```

Exemple : 

```
for $album in doc("albums.xml")/albums/album
  return element tr { element td { $album/titre/text() } }
```

La collection est générée par l'expression `/albums/album`. C'est la liste de tous les éléments `<album>` du document. Notez la fonction `text()` pour récupérer le contenu texte du titre.

# Clause For

La clause `for` fait un parcours sur une collection :

- énumération d'entiers (`INF to SUP`), par exemple :

```
for $i in (1 to 10) return $i * $i
```

- requête XPath qui retourne une collection de nœuds XML :

```
let $albums := doc("albums.xml")//album
for $mois in distinct-values( $albums//mois )
  return element tr {
    count(//album[date/mois=$mois]), "en", $mois }
```

La fonction `distinct-values(collection)` retourne une collection sans doublons.

# Clause For sur des attributs

Quand on veut itérer sur des attributs, comme dans :

```
for $attr_numero in doc("albums.xml")//album/@numero
```

La variable `$attr_numero` contient le nœud XML de type *attribute* et non pas seulement la valeur de l'attribut.

Il faut extraire la valeur comme ceci :

```
let $numero := string($attr_numero)
```


Pierre Nerzic

# Clause Let

Il est possible d'insérer une ou plusieurs affectations avant et après la clause `for`, pour un calcul intermédiaire :

```
for VARIABLE1 in COLLECTION
  let VARIABLE2 := EXPRESSION
  return SORTIE
```

Exemple :

```
let $albums := doc("albums.xml")//album
for $mois in distinct-values( $albums//mois )
  let $nombre := count( $albums[date/mois=$mois] )
  let $titres := $albums[date/mois=$mois]/titre
  return element titres {
    attribute nombre { $nombre }, attribute mois { $mois },
    $titres
  }
```


Pierre Nerzic

# Clause `Where`

C'est une condition optionnelle dans la boucle pour filtrer les itérations. Elle ressemble à la clause `where` des requêtes SQL.

```
let $albums := doc("albums.xml")//album
for $album in $albums
  where $album/date/annee >= 1970
  return $album
```

Remarque: pour ça, on pourrait aussi écrire du XPath pur :

```
doc("albums.xml")//album[date/annee>=1970]
```


Pierre Nerzic

# Clause `Order by`

Cette clause optionnelle permet de classer les éléments à traiter dans la boucle. On peut rajouter `ascending` ou `descending` pour indiquer le sens. Si le critère de tri est un nombre, le placer dans `number(expression)`.

```
for VARIABLE in COLLECTION
  order by EXPRESSION
  return SORTIE
```

Exemple : 

```
for $album in doc("albums.xml")//album
  let $titre := $album/titre
  where starts-with($titre, "Tintin")
  order by number($album//annee) descending
  return $titre
```


Pierre Nerzic

# Boucles imbriquées

Exemple :

```
let $albums := doc("albums.xml")//album
for $mois in distinct-values( $albums//mois )
  return element mois {
    attribute nom { $mois },
    for $album in $albums
      where $album/date/mois = $mois
      return $album/titre
  }
```

La clause `where` est une sorte de condition de jointure entre les mois et les albums.

La boucle interne pourrait être remplacée par une simple expression XPath.


Pierre Nerzic

# XQuery pour modifier la base XML


Pierre Nerzic

# Requêtes en modification

Il existe un langage appelé XQuery Update Facility (XQUF), documenté sur cette page, qui permet de modifier les données XML. Il ajoute de nouvelles requêtes parmi lesquelles :

- `insert node` *chose* `into` *expression* : *chose* décrit un nouveau *node* (élément ou attribut) qui doit être inséré dans les données aux emplacements désignés par l'expression.
- `delete node` *expression* : supprime l'élément ou l'attribut désigné par l'expression.
- `replace [value of] node` *expression* `with` *chose* : remplace le nœud (ou sa valeur) désigné avec l'expression par la *chose*.

Ces requêtes modifient des `nodes`, ce sont toutes sortes de nœuds dans l'arbre XML concerné : éléments, attributs, textes, commentaires, CDATA, etc.

# Insertion d'éléments

On peut écrire l'élément à ajouter en syntaxe XML ou avec la syntaxe XQuery `element nom { contenu }`.

Exemple, on rajoute un nouvel album dans albums.xml et un sous-élément `<lu/>` dans l'album n°1 :

```
insert node
        <album numero="25">
            <titre>Tintin et Astérix contre Spirou</titre>
        </album>
    into /albums
insert node element lu {} into /albums/album[@numero=1]
```

**Important** : l'expression après le `into` doit désigner un élément unique.

# Insertion sur plusieurs éléments

Pour systématiser l'insertion sur plusieurs éléments, il faut simplement utiliser une expression FLWOR. Voici un exemple, on veut ajouter l'élément `<achat date="2021-12-24"/>` dans tous les albums à partir du n°10 :

```
for $album in /albums/album
  where $album/@numero >= 10
  return insert node <achat date="2021-12-24"/> into $album
```

L'astuce est de mettre le `insert` en tant que `return`.


Pierre Nerzic

# Insertion sur plusieurs éléments, suite

S'il y a plusieurs actions à faire sur le même album, on peut les regrouper dans le return en les entourant avec des parenthèses et les séparant par une virgule :

```
for $album in /albums/album
  where $album/@numero >= 10
  return (
    insert node <achat date="2021-12-24"/> into $album,
    insert node element lu {} into $album
  )
```


Pierre Nerzic

# Insertion d'attributs

Il suffit de l'écrire à l'aide de la syntaxe XQuery `attribute nom {'valeur'}`.

Exemple, on rajoute un attribut `editeur` pour l'album n°13 : 

```
insert node attribute editeur {'Casterman'}
    into /albums/album[@numero="13"]
```

**Important** : comme précédemment, l'expression `into` doit désigner un emplacement unique, mais on peut utiliser une boucle pour insérer le même attribut dans des éléments différents.


Pierre Nerzic

# Suppression d'éléments ou d'attributs

Voici trois exemples, on supprime l'album n°4, puis tous les albums parus en janvier, et enfin l'attribut `numero` des albums parus après 1950 (ça casse les données !) :

```
delete node /albums/album[@numero="4"]
delete node /albums/album[date/mois="janvier"]
delete node /albums/album[date/annee>1950]/@numero
```

Contrairement aux `insert`, `replace` et `rename`, un `delete` peut concerner plusieurs nodes.

# Remplacement d'éléments ou d'attributs

Voici un exemple, on remplace le contenu de l'élément `<titre>` de l'album n°1 :

```
replace value of node /albums/album[@numero=1]/titre
    with 'nouveau titre'
```

Dans cet exemple, son attribut `numero` devient -1 :

```
replace value of node /albums/album[@numero=1]/@numero
    with -1
```


Pierre Nerzic

# Remplacement d'éléments ou d'attributs

On peut aussi remplacer un élément par autre chose. Par exemple, remplacer l'élément <date> et ses descendants par tout autre chose (ça peut casser le schéma) :

```
replace node /albums/album[@numero=2]/date
    with <auteur nom="hergé"/>
```

Il existe aussi une requête pour renommer un élément ou un attribut (faire une boucle s'il y en a plusieurs) :

`rename node` *designation* `as` *nouveau nom*


Pierre Nerzic

# Remplacement de plusieurs sous-éléments

Au lieu de faire plusieurs requêtes pour modifier des sous-éléments liés, on peut les grouper dans des parenthèses : 

```
let $album := /albums/album[@numero=2]
return (
    replace node $album/date with <auteur nom="sempé"/>,
    replace value of node $album/@serie with 'sempe'
)
```

Notez la virgule entre les deux `replace` dans le `return`.


Pierre Nerzic

# Autres actions

Il y a de nombreuses autres actions possibles, soit des raffinements des actions comme `insert`, soit d'autres opérations plus spécifiques pour faire des sortes de transactions. Elles sont trop complexes pour être présentées ici et ne seront pas nécessaires en TP.

# Bases de données XML


Pierre Nerzic

# Présentation

Une base de données XML native incorpore des données directement au format XML et propose les langages XPath et XQuery pour les interroger.

Ces bases de données sont des sortes de bases NoSQL : *not only SQL*. La structuration des données et les méthodes d'interrogation sont d'un autre genre que les requêtes SQL sur une base relationnelle.

Nous verrons une autre approche en fin de période, l'intégration de XML dans PostgreSQL.

# Principe général d'un SGBD XML

Le SGBD stocke des « forêts » d'arbres XML qui peuvent provenir de différents documents XML ayant par exemple le même schéma.

Le SGBD fonctionne en mode client/serveur. Il exécute des requêtes de type XQuery à la demande des clients. Il existe des langages XML permettant de modifier les données, *XQuery Update*, et dans certains cas, le SGBD fournit un modèle REST[2] pour les clients.

Les requêtes sur un gros document sont rendues efficaces à l'aide d'index sur les éléments.

[2] Un serveur REST propose plusieurs méthodes d'interrogation et de modifications à l'aide de requêtes HTTP GET, PUT, POST, DELETE. Chaque requête est complète et indépendante des autres.

# Utilisation de BaseX

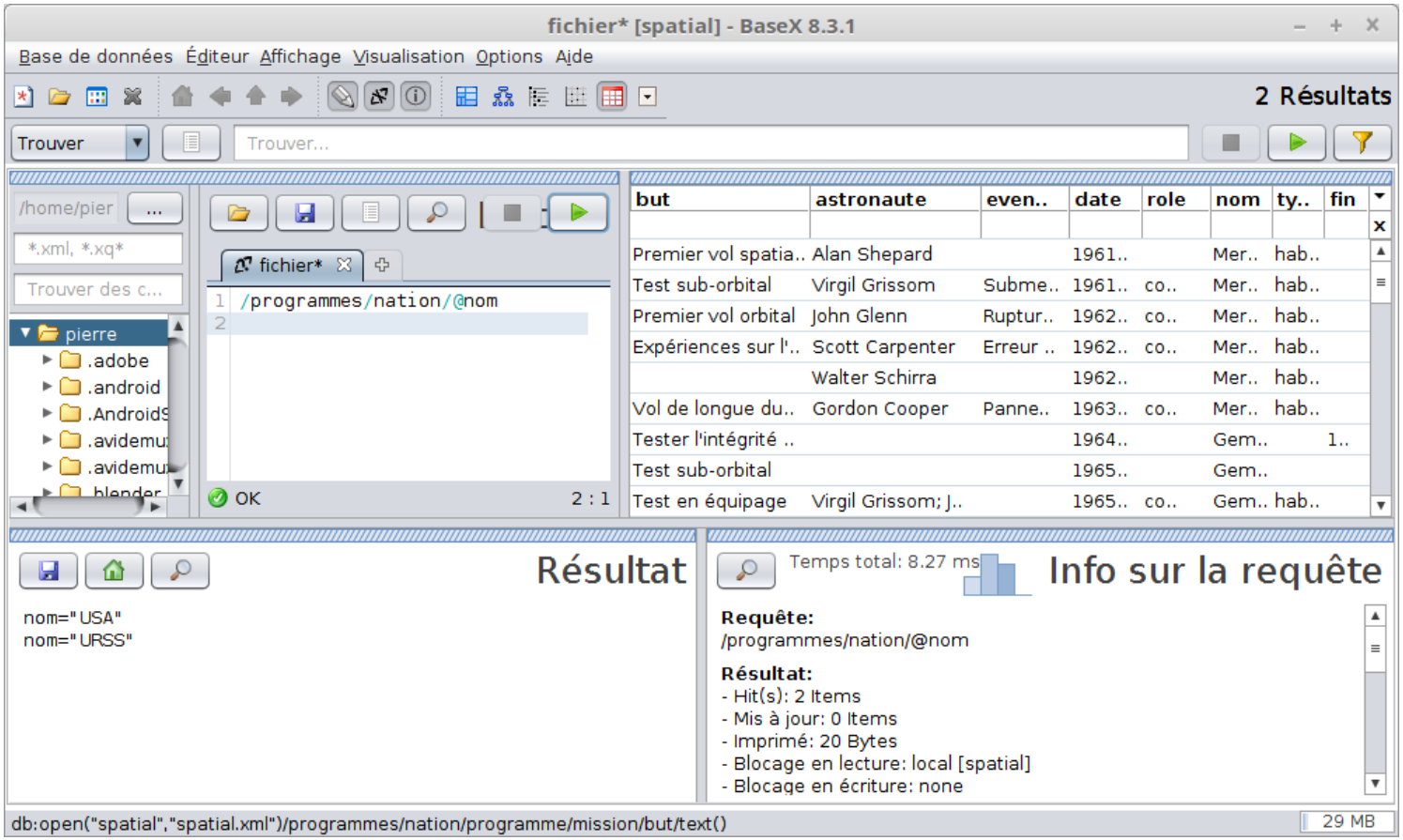
En TP, nous utiliserons BaseX. Il y a également eXist. Tous deux sont gratuits et open source. BaseX a été développé initialement par l'Université de Constance en Allemagne et il est maintenant sur GitHub, licence BSD.

Le SGBD BaseX propose :

- Interrogation à l'aide de XQuery (y compris version 3),
- Modification à l'aide de XQuery Update,
- Serveurs intégrés de type RESTful et WebDAV (mais il faut télécharger des archives spécifiques pour les avoir)
- Une interface utilisateur complète.

La documentation complète est disponible au format pdf sur cette page.

# Interface de BaseX



# Interface graphique (suite)

L'interface est composée de plusieurs panneaux :

- Une barre pour saisir directement des recherches texte ou des requêtes XQuery ou des commandes sur la base,
- Un éditeur de requête XQuery,
- Une vue tabulaire de la base XML,
- Les résultats de la requête, qu'on peut aussi afficher graphiquement selon le type des données,
- Des statistiques sur l'exécution de la requête.

Elle est documentée sur cette page.

En fait, c'est tellement complet qu'on ne pourra faire que quelques manipulations de découverte en TP, hormis des requêtes XQuery.

# Création d'une base de données XML

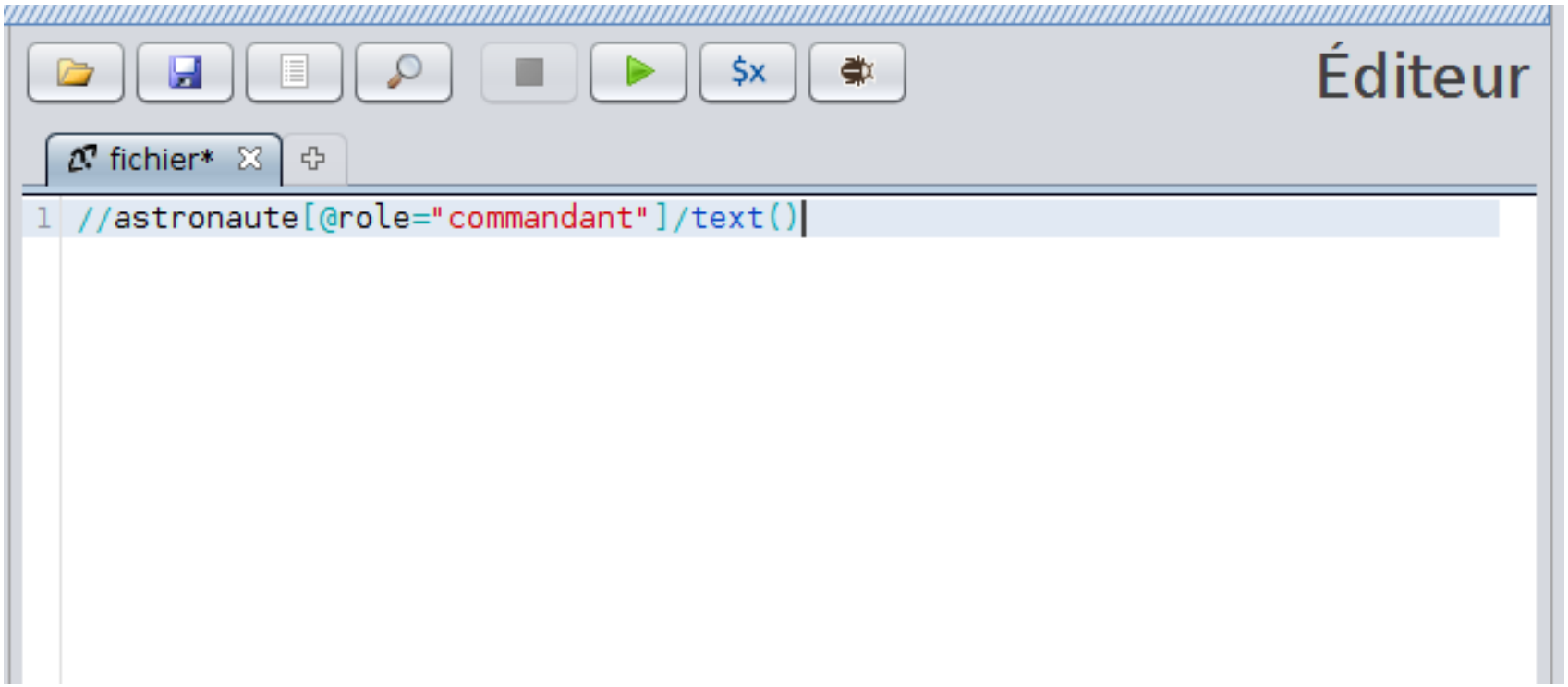
Il faut posséder un fichier de données au format XML. C'est une collection de données toutes similaires, par exemple comme la liste des albums de Tintin ou un annuaire téléphonique. Il faut que ces données puissent être distinguées entre elles, par exemple par un attribut identifiant.

Soit avec les menus, soit avec une commande (voir en TP), on fait prendre en charge ce fichier par BaseX.

| but | astronaute | evenement | date | role | nom | type | fin |
|---|---|---|---|---|---|---|---|
| Premier vol spatial américain | Alan Shepard | | 1961-05-05 | | Mercury 3 | habité | |
| Test sub-orbital | Virgil Grissom | Submersion de.. | 1961-06-21 | command.. | Mercury 4 | habité | |
| Premier vol orbital | John Glenn | Rupture de la f.. | 1962-02-20 | command.. | Mercury 6 | habité | |
| Expériences sur l'impesanteur | Scott Carpenter | Erreur de pilot.. | 1962-05-24 | command.. | Mercury 7 | habité | |
| | Walter Schirra | | 1962-10-03 | | Mercury 8 | habité | |
| Vol de longue durée | Gordon Cooper | Panne électriq.. | 1963-05-15 | command.. | Mercury 9 | habité | |
| Tester l'intégrité de la capsule | | | 1964-04-08 | | Gemini 1 | | 1964-.. |
| Test sub-orbital | | | 1965-01-19 | | Gemini 2 | | |
| Test en équipage | Virgil Grissom; John Young | | 1965-03-23 | command.. | Gemini 3 | habité | |

# Requête XQuery

On peut ensuite écrire des requêtes XPath ou XQuery dans la barre ou dans la zone d'édition et lancer l'exécution.





Pierre Nerzic