XQuery Cheatsheet

XPath Cheat Sheet

XPath (XML Path Language) is used to navigate and query XML documents, enabling users to select nodes or node-sets in an XML document.

1. Basic XPath Syntax

- **Absolute Path**: Starts from the root (`/`).

  ```
  /bookstore/book
  ```

- **Relative Path**: Starts from the current node (`.` or `//` for any level).

  ```
  ./book/author
  //author
  ```

- **Selecting Elements**:
  - Select all `book` elements:

    ```
    //book
    ```

  - Select `title` elements within `book` elements:

    ```
    /bookstore/book/title
    ```

2. XPath Axes

- **child**: Selects child nodes.

  ```
  child::book
  ```

- **parent**: Selects parent of the current node.

  ```
  parent::*
  ```

- **descendant**: Selects all descendants (children, grandchildren).

```
descendant::title
```

- **ancestor**: Selects all ancestors (parent, grandparent).

  ```
  ancestor::bookstore
  ```

- **following-sibling**: Selects all siblings after the current node.

  ```
  following-sibling::book
  ```

- **preceding-sibling**: Selects all siblings before the current node.

  ```
  preceding-sibling::title
  ```

3. Node Tests

- **Select All Elements**:

  ```
  //*
  ```

- **Select Specific Elements by Name**:

  ```
  //book
  ```

- **Wildcard for All Nodes**:

  ```
  /*
  ```

4. Predicates

Predicates help filter nodes based on conditions.

- **Select based on attribute value**:

  ```
  //book[@category='fiction']
  ```

- **Select the first `book` element**:

  ```
  (//book)[1]
  ```

- **Select nodes based on position**:
  - First `book`:

```
//book[position()=1]
```

- Last `book`:

  ```
  //book[last()]
  ```

5. XPath Functions

- **String Functions**
  - **starts-with()**: Checks if a string starts with a substring.

    ```
    //book[starts-with(title, 'Intro')]
    ```

  - **contains()**: Checks if a string contains a substring.

    ```
    //book[contains(description, 'science')]
    ```

- **Numeric Functions**
  - **sum()**: Calculates the sum of selected nodes.

    ```
    sum(//book/price)
    ```

- **Boolean Functions**
  - **not()**: Returns true if the argument is false.

    ```
    //book[not(price < 10)]
    ```

6. XPath Operators

- **Arithmetic Operators**: `+`, `-`, `*`, `div`
- **Comparison Operators**: `=`, `!=`, `<`, `<=`, `>`, `>=`
- **Boolean Operators**: `and`, `or`

XPath Examples

- **Select all books priced over 20**:

  ```
  //book[price > 20]
  ```

- **Select the title of books by a specific author**:

```
//book[author='John Doe']/title
```

FLWOR Expressions Cheat Sheet

FLWOR (For, Let, Where, Order by, Return) is used in **XQuery** to process and query XML data, similar to SQL.

1. FLWOR Syntax Structure

```
for $var in collection
let $var2 := expression
where condition
order by expression
return result
```

2. FLWOR Clauses

- **For**: Iterates over a sequence of items.

  ```
  for $book in doc("books.xml")//book
  ```

- **Let**: Binds variables to expressions.

  ```
  let $discount := $book/price * 0.9
  ```

- **Where**: Filters items based on a condition.

  ```
  where $book/price > 20
  ```

- **Order By**: Sorts results by specified criteria.

  ```
  order by $book/title ascending
  ```

- **Return**: Specifies the result of the query.

  ```
  return <discounted-book>{$book/title, $discount}</discounted-book>
  ```

3. Common FLWOR Examples

Retrieve All Book Titles

```
for $book in doc("books.xml")//book
return $book/title
```

Apply a Discount to Books Over a Certain Price

```
for $book in doc("books.xml")//book
let $discountPrice := $book/price * 0.9
where $book/price > 30
return <book>
           {$book/title, <discounted-price>{$discountPrice}</discounted-price>}
       </book>
```

Sort Books by Price in Descending Order

```
for $book in doc("books.xml")//book
order by $book/price descending
return $book
```

4. FLWOR with Aggregations

- **Count the Number of Books by Each Author**

```
for $author in distinct-values(doc("books.xml")//book/author)
let $count := count(doc("books.xml")//book[author = $author])
return <author name="{$author}">
           <book-count>{$count}</book-count>
       </author>
```

5. Grouping in FLWOR

- **Group Books by Category and Return Total Price for Each Group**

```
for $category in distinct-values(doc("books.xml")//book/@category)
let $totalPrice := sum(doc("books.xml")//book[@category = $category]/price)
return <category name="{$category}">
           <total-price>{$totalPrice}</total-price>
       </category>
```

Advanced FLWOR Techniques

Joining Data Using FLWOR

- **Join Books with Authors (From Separate Documents)**

```
for $book in doc("books.xml")//book,
    $author in doc("authors.xml")//author
where $book/author-id = $author/@id
return <book-details>
           {$book/title, $author/name}
       </book-details>
```

Nested FLWOR Expressions

- **Return Each Book with a List of Similar Books (e.g., by Genre)**

```
for $book in doc("books.xml")//book
return <book>
           {$book/title}
           <similar-books>
               {
                   for $similar in doc("books.xml")//book
                   where $similar/genre = $book/genre and $similar != $book
                   return $similar/title
               }
           </similar-books>
       </book>
```

Subqueries in Return Clause

- **Calculate Total Price for Books in Each Category**

```
for $category in distinct-values(doc("books.xml")//book/@category)
return <category name="{$category}">
           {
               let $total := sum(for $book in doc("books.xml")//book[@category =
$category] return $book/price)
               return <total-price>{$total}</total-price>
           }
       </category>
```