Capítulo 8

XML: Parte 2

Example Document

```
<BARS>
   <BAR name = “JoesBar”>
        <PRICE theBeer = “Bud”>2.50</PRICE>
        <PRICE theBeer = “Miller”>3.00</PRICE>
   </BAR> …
   <BEER name = “Bud” soldBy = “JoesBar
        SuesBar … ”/> …
</BARS>
```

XQuery: FLWR expressons

- **Comparación con SQL**

  for ⇔ SQL **from**
  where ⇔ SQL **where**
  order by ⇔ SQL **order by**

  return ⇔ SQL **select**
  let permite variables temporales, y no tiene equivalente en SQL

XQuery: FLWR expressons

- **Semántica de Expresión FLWR**
  - Cada **for** crea un loop.
  - **let** produce solo una definición local.
  - En cada iteración de los loops anidados, evaluar la cláusula **where**.
  - Si la cláusula **where** retorna TRUE, invocar la cláusula **return**, y agregar su valor a la salida.

XQuery: FLWR expressons

- La **cláusula for** usa expresiones XPath, y la variable en la cláusula for recorre los valores en la secuencia retornada por la expresión Xpath.
- Items en la **cláusula return** son texto XML a menos que estén englobados entre {}, en cuyo caso son evaluados.

XQuery: FLWR Expression

- **Ejemplo:** encontrar todas las cuentas con balance mayor que 400 donde cada resultado está englobado entre etiquetas <account_number> y </account_number>

XQuery: FLWR Expression

- **Ejemplo:** encontrar todas las cuentas con balance mayor que 400 donde cada resultado está englobado entre etiquetas <account_number> y </account_number>
  - **for** $x in /bank-2/account
    let $acctno := $x/@account_number
    where $x/balance > 400
    return <account_number> { $acctno } </account_number>
- La cláusula Let no se necesita realmente en la consulta anterior, y la selección puede ser hecha en Xpath. La consulta puede escribirse como:

XQuery: FLWR Expression

- **Ejemplo:** encontrar todas las cuentas con balance mayor que 400 donde cada resultado está englobado entre etiquetas <account_number> y </account_number>
  - **for** $x in /bank-2/account
    let $acctno := $x/@account_number
    where $x/balance > 400
    return <account_number> { $acctno } </account_number>
- La cláusula Let no se necesita realmente en la consulta anterior, y la selección puede ser hecha en Xpath. La consulta puede escribirse como:
  - **for** $x in /bank-2/account[balance>400]
    return <account_number> { $x/@account_number }
    </account_number>

XQuery: FLWR expressions

for <variable> **in** <expression>, . . .

- Las variables comienzan con $.
- La variable del **for** considera cada item en la secuencia obtenida por evaluar la expresión.
- **Semántica**: Cualquier cosa que siga al **for** es ejecutada una vez por cada valor de la variable.

XQuery : FLWR expressions

- **Ejemplo**: encontrar todos los nombres de cerveza, donde cada resultado está englobado entre etiquetas `<BEERNAME>` y `</BEERNAME>`

XQuery : FLWR expressions

- **Ejemplo**: encontrar todos los nombres de cerveza, donde cada resultado está englobado entre etiquetas `<BEERNAME>` y `</BEERNAME>`

  for $beer **in** document("bars.xml")/BARS/BEER/@name

  return <BEERNAME> {$beer} </BEERNAME>

- $beer recorre los atributos `name` de todas las cervezas.
- **¿Cuál es el resultado de la consulta?**

XQuery : FLWR expressions

- **Ejemplo**: encontrar todos los nombres de cerveza, donde cada resultado está englobado entre etiquetas `<BEERNAME>` y `</BEERNAME>`

  for $beer **in** document("bars.xml")/BARS/BEER/@name

  return <BEERNAME> {$beer} </BEERNAME>

- $beer recorre los atributos `name` de todas las cervezas.
- **¿Cuál es el resultado de la consulta?**
- <BEERNAME>Bud</BEERNAME>
  <BEERNAME>Miller</BEERNAME> . . .

XQuery: FLWR expressions

let <variable> := <expression>, . . .

- El valor de la variable es una **sequencia** de items definida por la expresión.
- Notar que **let** no causa iteración; **for** sí.
- Cuando una variable es ligada en una cláusula for o let, puede ser referenciada en cualquier lugar en esa FLWOR luego de la cláusula que la liga, *pero no antes.*

XQuery: FLWR expressons

- **Ejemplo**: encontrar todos los nombres de cerveza,
  - la lista de esos nombres deberá estar englobada entre las etiquetas `<BEERNAMES>` y `</BEERNAMES>`. Usar let.

XQuery: FLWR expressons

- **Ejemplo**: encontrar todos los nombres de cerveza,
  - la lista de esos nombres deberá estar englobada entre las etiquetas `<BEERNAMES>` y `</BEERNAMES>`. Usar let.

```
let $d := document("bars.xml")
let $beers := $d/BARS/BEER/@name
return <BEERNAMES> {$beers} </BEERNAMES>
```

- **¿Cuál es el resultado de la consulta?**

XQuery: FLWR expressons

- **Ejemplo**: encontrar todos los nombres de cerveza,
  - la lista de esos nombres deberá estar englobada entre las etiquetas `<BEERNAMES>` y `</BEERNAMES>`. Usar let.

```
let $d := document("bars.xml")
let $beers := $d/BARS/BEER/@name
return <BEERNAMES> {$beers} </BEERNAMES>
```

- **¿Cuál es el resultado de la consulta?**

```
<BEERNAMES>Bud Miller …</BEERNAMES>
```

XQuery: Expresiones englobadas

- Cuando un constructor de elemento contiene otra expresión XQuery entre llaves, esta se llama **expresión englobada** y su valor se convierte en parte del contenido del elemento en los resultados.

XQuery: Expresiones englobadas

- **Ejemplo**: Presentar usando HTML un encabezado con título `Product Catalog`; luego construir una lista no ordenada de items, donde cada ítem contiene los literales `number:` y `name:` seguidos cada uno de ellos por expresiones englobadas que proveen la información respectiva (cada una de ellas deberá evaluar a un valor atómico).

XQuery: Expresiones englobadas

- Queremos que la salida de la consulta sea:

```
<html>
<h1>Product Catalog</h1>
<ul>
    <li>number: 557, name: Fleece Pullover</li>
    <li>number: 563, name: Floppy Sun Hat</li>
    <li>number: 443, name: Deluxe Travel Bag</li>
    <li>number: 784, name: Cotton Dress Shirt</li>
</ul>
</html>
```

XQuery: Expresiones englobadas

- `fn:data($arg as item()*) as xs:anyAtomicType*`
- **`fn:data`** toma una secuencia de items y devuelve una secuencia de valores atómicos.
- El resultado de `fn:data` es la secuencia de valores atómicos producida por aplicar las siguientes reglas a cada ítem en `$arg`:
  - Si el ítem es un valor atómico, el mismo es retornado.
  - Si el ítem es un nodo:
    - Si el nodo no tiene un valor tipado, se emite un error
    - De otro modo, `fn:data()` retorna el valor tipado del nodo

XQuery: Expresiones englobadas

```
<html>
    <h1>Product Catalog</h1>
    <ul>{
        for $prod in doc("catalog.xml")/catalog/product
        return <li>number: {data($prod/number)}, name: {data($prod/name)}</li>
    }</ul>
</html>
```

XQuery: Constructores

- Los **constructores de elementos directos** usan una sintaxis que se parece mucho a XML.
  - Las etiquetas usan ‘<‘ y ‘>’,
  - los nombres deben ser nombres XML válidos,
  - toda start-tag debe tener una correspondiente end-tag.
- Además se pueden usar nombres prefijados e incluso se pueden incluir declaraciones de espacios de nombres.
- Los atributos dentro de un constructor de elemento directo deben tener nombres únicos.
- Constructores de elementos directos pueden contener otros constructores de elementos directos.

XQuery: Constructores

- **Ejercicio**: Presentar usando HTML un encabezado con título Product Catalog; luego producir un párrafo donde se indica la cantidad de productos que hay en el catálogo (**hint** usar función `count`).
  - Queremos que la salida de la consulta sea:

```
<html>
   <h1>Product Catalog</h1>
   <p>A <i>huge</i> list of 4 products.</p>
</html>
```

XQuery: Constructores

Query

```
<html>
    <h1>Product Catalog</h1>
    <p>A <i>huge</i> list of
   {count(doc("catalog.xml")//product)}  products.</p>
</html>
```

- El constructor de elemento `html` contiene constructores de elementos para `h1`.
- El constructor de elemento `p` contiene una combinación de contenido de datos y un constructor para el elemento i y la expresión englobada.

XQuery: Expresiones englobadas

- ***¿Qué se puede expresar con expresiones englobadas?***
- Es posible para una expresión englobada evaluar a una secuencia de atributos u otros nodos, valores atómicos o incluso combinaciones de nodos y valores atómicos.
- Si la expresión englobada evalúa a una secuencia de elementos y valores atómicos, el elemento resultado tiene **contenido mezclado**, donde el orden de los elementos hijo y datos carácter son preservados.

XQuery: Expresiones englobadas

- **Ejemplo**: Obtener una lista de ítems donde para cada ítem se tiene `number:` seguido del elemento `<number>.`

XQuery: Expresiones englobadas

- **Ejemplo**: Obtener una lista de ítems donde para cada ítem se tiene `number:` seguido del elemento `<number>`.

```
for $prod in doc("catalog.xml")/catalog/product
return <li>number: {$prod/number}</li>
```

- **¿Cuáles son los resultados de esta consulta?**

XQuery: Expresiones englobadas

- **Ejemplo**: Obtener una lista de ítems donde para cada ítem se tiene `number:` seguido del elemento `<number>`.

```
for $prod in doc("catalog.xml")/catalog/product
return <li>number: {$prod/number}</li>
```

- **¿Cuáles son los resultados de esta consulta?**

```
<li>number: <number>557</number></li>
<li>number: <number>563</number></li>
<li>number: <number>443</number></li>
<li>number: <number>784</number></li>
```

XQuery: Construcción de Atributos

- ***Se está construyendo un elemento E. ¿Cómo incluir atributos para E que provienen de otro elemento F?***
- **Solución**: Hacer que el constructor de elemento *E* contenga una expresión englobada que evalúa a uno o más atributos de *F*; esos atributos se convierten en atributos del elemento *E* bajo construcción.
  - Expresiones englobadas que evalúan a atributos deben aparecer primero en el constructor de contenido de elemento, antes que otros tipos de nodos.

XQuery: Construcción de Atributos

- **Ejercicio:** obtener la lista de elementos `li` que para cada producto produce atributo `dept` con valor de atributo `dept` de `prod` y elemento `<number>` de `prod`.

XQuery: Construcción de Atributos

- **Ejercicio:** obtener la lista de elementos `li` que para cada producto produce atributo `dept` con valor de atributo `dept` de `prod` y elemento `<number>` de `prod`.

XQuery: Construcción de Atributos

- **Ejercicio:** obtener la lista de elementos `pr` que para cada producto produce atributo `dept` con valor de atributo `dept` de `prod` y elemento `<number>` de `prod`.

  for $prod in doc("catalog.xml")/catalog/product

  return <pr>**{$prod/@dept}number: {$prod/number}**</pr>

- **¿Cuáles son los resultados de la consulta?**

XQuery: Construcción de Atributos

- **Ejercicio:** obtener la lista de elementos `pr` que para cada producto produce atributo `dept` con valor de atributo `dept` de `prod` y elemento `<number>` de `prod`.

```
for $prod in doc("catalog.xml")/catalog/product
return <pr>{$prod/@dept}number: {$prod/number}</pr>
```

- **¿Cuáles son los resultados de la consulta?**

```
<pr dept="WMN">number: <number>557</number></pr>
<pr dept="ACC">number: <number>563</number></pr>
<pr dept="ACC">number: <number>443</number></pr>
<pr dept="MEN">number: <number>784</number></pr>
```

XQuery: Expresiones englobadas

- ***¿Qué pasa con la salida si una expresión englobada evalúa a uno o más valores atómicos?***
- Esos valores son convertidos a `xs:string` e incluidos como contenido de datos carácter del elemento.
  - Cuando valores atómicos adyacentes aparecen en la secuencia de expresión, son separados por un espacio en el contenido del elemento.

XQuery: Expresiones englobadas

- **Ejemplo**: Sea {"x", "y", "z"}
- **¿Qué retorna esta consulta?**

XQuery: Expresiones englobadas

- **Ejemplo**: Sea {"x", "y", "z"}
- **¿Qué retorna esta consulta?**
- va a retornar x y z.
- **¿Cómo hacer para evitar los espacios?**

XQuery: Expresiones englobadas

- **Ejemplo**: Sea {"x", "y", "z"}
- **¿Qué retorna esta consulta?**
- va a retornar x y z.
- **¿Cómo hacer para evitar los espacios?**
- Para evitar los espacios se puede usar: {"x"}{"y"}{"z"},
- otra opción es usar la función concat: {concat("x", "y", "z")}.

XQuery: Expresiones englobadas

- ***Hasta aquí las expresiones englobadas solo incluían una expresión, ¿es esto obligatorio?***
- Expresiones englobadas pueden incluir más de una subexpresión dentro de llaves usando comas como separadores.

XQuery: Expresiones englobadas

- **Ejemplo**:

  for $prod in doc("catalog.xml")/catalog/product

  return <pr>{$prod/@dept,"string",5+3,$prod/number}</pr>
- **¿Qué resultado retorna esta consulta?**

XQuery: Expresiones englobadas

- **Ejemplo**:

  for $prod in doc("catalog.xml")/catalog/product

  return <pr>{$prod/@dept,"string",5+3,$prod/number}</pr>

- **¿Qué resultado retorna esta consulta?**

  <pr dept="WMN">string 8<number>557</number></pr>

  <pr dept="ACC">string 8<number>563</number></pr>

  <pr dept="ACC">string 8<number>443</number></pr>

  <pr dept="MEN">string 8<number>784</number></pr>

XQuery: Construcción de atributos

- ***¿Cómo construir directamente atributos de un elemento E?***
- **Solución**: Los atributos se pueden construir usando sintaxis XML.
  - Los valores de atributo pueden ser especificados usando un texto literal o expresiones englobadas o una combinación de los dos.

XQuery: Construcción de atributos

- **Ejemplo**: generar un documento con la siguiente salida:

```
<html> <h1 class="itemHdr">Product Catalog</h1>
<ul>
        <li dep="WMN">number: 557, name: Fleece Pullover</li>
        <li dep="ACC">number: 563, name: Floppy Sun Hat</li>
        <li dep="ACC">number: 443, name: Deluxe Travel Bag</li>
        <li dep="MEN">number: 784, name: Cotton Dress Shirt</li>
</ul> </html>
```

Se pide que los valores de los atributos para los ítems sean construidos usando expresiones englobadas.

XQuery: Construcción de atributos

```
<html>
    <h1 class="itemHdr">Product Catalog</h1>
    <ul>{
        for $prod in doc("catalog.xml")/catalog/product
        return <li dep="{$prod/@dept}">
                    number: {data($prod/number)},
                    name: {data($prod/name)}</li>
    }</ul>
</html>
```

- **Importante**: El atributo `dep` va a aparecer sin importar si hay un atributo `dept` del elemento `$prod`; si este elemento no tiene atributo `dept`, el valor del atributo `dep` va a ser el string vacío.

XQuery: Construcción de atributos

- **Ejemplo:** Escribir un encabezado “Product catalog”, luego para cada producto construir el atributo `class` cuyo valor es el valor del atributo `dept` de `prod`; además incluir el valor del elemento `number` de `prod`.

XQuery: Construcción de atributos

- **Ejemplo:** Escribir un encabezado “Product catalog”, luego para cada producto construir el atributo `class` cuyo valor es el valor del atributo `dept` de `prod`; además incluir el valor del elemento `number` de `prod`.

```
<xhtml:html
   xmlns:xhtml="http://www.w3.org/1999/xhtml">
   <xhtml:h1 class="itemHdr">Product Catalog</xhtml:h1>
   <xhtml:ul>{
      for $prod in doc("catalog.xml")/catalog/product
      return <xhtml:li class="{$prod/@dept}">number: {
            data($prod/number)}</xhtml:li>
   }</xhtml:ul>
</xhtml:html>
```

- **¿Qué resultados produce esta consulta?**

XQuery: Construcción de atributos

```
<xhtml:html xmlns:xhtml="http://www.w3.org/1999/xhtml">
    <xhtml:h1 class="itemHdr">Product Catalog</xhtml:h1>
    <xhtml:ul>
        <xhtml:li class="WMN">number: 557</xhtml:li>
        <xhtml:li class="ACC">number: 563</xhtml:li>
        <xhtml:li class="ACC">number: 443</xhtml:li>
        <xhtml:li class="MEN">number: 784</xhtml:li>
    </xhtml:ul>
</xhtml:html>
```

XQuery: Construcción de atributos

- **Problema:** Se quiere incluir elementos del documento de entrada, pero quiere hacer modificaciones menores tales como agregar o remover un hijo o un atributo.
- **Solución**:
  - Nuevos elementos o atributos deben ser creados usando constructores.
  - Si $E contiene todos los elementos de etiqueta <E> que aparecen en el documento, entonces:
    - `{$E/(@*, *)}` copia todos los atributos y subelementos de los elementos en $E.
    - `{$E/(@* except @attr-name, * except elem-name)}` copia todos los atributos y subelementos de los elementos en $E menos el atributo de nombre `attr-name` y el elemento de elemento de nombre `elem-name`.

XQuery: Construcción de atributos

- **Ejemplo**: incluir elementos `product` del documento de entrada, pero se agrega un atributo adicional `id` que es igual a la letra **P** más el número de producto.

XQuery: Construcción de atributos

- **Ejemplo**: incluir elementos `product` del documento de entrada, pero se agrega un atributo adicional `id` que es igual a la letra **P** más el número de producto.

```
for $prod in doc("catalog.xml")/catalog/product[@dept = 'ACC']
return <product id="P{$prod/number}">
            {$prod/(@*, *)}
        </product>
```

- **¿Qué resultados da esta consulta?**

XQuery: Construcción de atributos

```
<product dept="ACC" id="P563">
   <number>563</number>
   <name language="en">Floppy Sun Hat</name>
</product>
<product dept="ACC" id="P443">
   <number>443</number>
   <name language="en">Deluxe Travel Bag</name>
</product>
```

XQuery: Expresiones englobadas

- **Ejemplo**: suponga que quiere copiar algunos elementos de `product` del documento de entrada pero remover el hijo `number`.

XQuery: Expresiones englobadas

- **Ejemplo**: suponga que quiere copiar algunos elementos de `product` del documento de entrada pero remover el hijo `number`.

```
for $prod in doc("catalog.xml")/catalog/product[@dept = 'ACC']
return <product>
        {$prod/(@*, * except number)}
          </product>
```

- **¿Qué resultados arroja la consulta anterior?**

XQuery: Expresiones englobadas

- **Ejemplo**: suponga que quiere copiar algunos elementos de `product` del documento de entrada pero remover el hijo `number`.

```
for $prod in doc("catalog.xml")/catalog/product[@dept = 'ACC']
return <product>
        {$prod/(@*, * except number)}
          </product>
```

- **¿Qué resultados arroja la consulta anterior?**

```
<product dept="ACC">
   <name language="en">Floppy Sun Hat</name>
</product>
<product dept="ACC">
   <name language="en">Deluxe Travel Bag</name>
</product>
```

XQuery: reuniones

- Las reuniones son especificadas de una manera muy similar a SQL (era igualar atributos con igual nombre).
- **Ejercicio**: mostrar un listado donde cada línea consiste de un cliente y de una cuenta del mismo (usar **where**).

DTD: Elements

```
<!DOCTYPE bank [
    <!ELEMENT bank ( ( account | customer | depositor)+)>
    <!ELEMENT account (account_number branch_name balance)>
    <! ELEMENT customer(customer_name customer_street  customer_city)>
    <! ELEMENT depositor (customer_name account_number)>
    <! ELEMENT account_number (#PCDATA)>
    <! ELEMENT branch_name (#PCDATA)>
    <! ELEMENT balance(#PCDATA)>
    <! ELEMENT customer_name(#PCDATA)>
    <! ELEMENT customer_street(#PCDATA)>
    <! ELEMENT customer_city(#PCDATA)>
]>
```

XQuery: reuniones

- Las reuniones son especificadas de una manera muy similar a SQL (era igualar atributos con igual nombre).
- **Ejercicio**: mostrar un listado donde cada línea consiste de un cliente y de una cuenta del mismo (usar **where**).

```
for $a in /bank/account,
    $c in /bank/customer,
    $d in /bank/depositor
where   $a/account_number = $d/account_number
          and $c/customer_name = $d/customer_name
return <cust_acct> { $c $a } </cust_acct>
```

XQuery: reuniones

- **Ejercicio:** Expresar la misma consulta sin usar where y usando predicado de selección en XPath.

XQuery: reuniones

- **Ejercicio:** Expresar la misma consulta sin usar where y usando predicado de selección en XPath.

```
for $a in /bank/account
      $c in /bank/customer
      $d in /bank/depositor[
                  account_number = $a/account_number and
                  customer_name  = $c/customer_name]
return <cust_acct> { $c $a } </cust_acct>
```

XQuery: reuniones

```
<bank-1>
    <customer>
        <customer_name> Hayes </customer_name>
        <customer_street> Main </customer_street>
        <customer_city>    Harrison </customer_city>
        <account>
            <account_number> A-102 </account_number>
            <branch_name>     Perryridge </branch_name>
            <balance>             400 </balance>
        </account>
        <account>
            …
        </account>
    </customer>
      .
      .
</bank-1>
```

XQuery: reuniones

- **Ejercicio:** Se requiere convertir datos cumpliendo el DTD `bank` para la información en la estructura anidada usada en `bank-1`; o sea para cada cliente mostrar sus subelementos y listar todas las cuentas de ese cliente juntas .

XQuery: reuniones

- **Ejercicio:** Se requiere convertir datos cumpliendo el DTD `bank` para la información en la estructura anidada usada en `bank-1`; o sea para cada cliente mostrar sus subelementos y listar todas las cuentas de ese cliente juntas .

```
<bank-1> {
    for $c in /bank/customer
    return
        <customer>
          { $c/* }
          { for $d in /bank/depositor[customer_name = $c/customer_name],
                $a in /bank/account[account_number=$d/account_number]
          return $a }
       </customer>
} </bank-1>
```

  - $c/* denota todos los hijos del nodo al cual $c está ligado, sin la etiqueta de alto nivel que la abarca.
  - $c/text() da el contenido texto de un elemento sin todos los subelementos / etiquetas

XQuery: Expresiones Cuantificadas

- Una **expresión cuantificada** determina si algunos o todos los ítems en una secuencia cumplen con una determinada condición.
- Una expresión cuantificada siempre evalúa a un valor Booleano.
- Una expresión cuantificada es hecha de varias partes:
  - Un **cuantificador** (**some** o **every**).
  - Una o más cláusulas in que ligan variables a secuencias.
  - Una cláusula **satisfies** que contiene la **expresión de prueba**.
- La sintaxis de una expresión cuantificada es:

  (some|every) ($ <variable-name> in <expr> ,)+ satisfies <expr>

XQuery: Expresiones Cuantificadas

- **Semántica de** `some $x in` E_1 `satisfies` E_2
  1. Evalúa la secuencia E_1.
  2. Sea $x (cualquier variable) un item en la secuencia, y evaluar E_2.
  3. Retornar TRUE si E_2 tiene EBV TRUE para al menos un $x.
- Análogamente para: `every $x in` E_1 `satisfies` E_2

XQuery: Expresiones Cuantificadas

- **Observación**: las expresiones cuantificadas se pueden expresar usando expresiones FLWOR y operaciones sobre secuencias.
  - some variables satisfies condition **<=>**

    exists(for variables where condition return 1)
  - every variables satisfies condition **<=>**

    empty(for variables where not(condition) return 1)
- Sin embargo, una expresión cuantificada puede ser más compacta y fácil para optimizar implementaciones.

XQuery: Expresiones Cuantificadas

- **Ejemplo**: para averiguar si algún ítem en una orden son del departamento de accesorios:

XQuery: Expresiones Cuantificadas

- **Ejemplo**: para averiguar si algún ítem en una orden son del departamento de accesorios:

  some $dept in /order/item/@dept

  satisfies ($dept = "ACC")

- **Ejemplo**: para averiguar si todos los ítems en una orden son del departamento de accesorios:

  every $dept in /order/item/@dept

  satisfies ($dept = "ACC")

XQuery: Expresiones Cuantificadas

- **Ejercicio**: expresar la consulta encontrar los números de producto cuyo precio es mayor que todos los precios de los productos que no tienen un subelemento de tipo `discount` (ver filmina 36 de parte 1 de XML).

XQuery: Expresiones Cuantificadas

- **Ejercicio**: expresar la consulta encontrar los números de producto cuyo precio es mayor que todos los precios de los productos que no tienen un subelemento de tipo `discount` (ver filmina 36 de parte 1 de XML).

```
for $p in /prices/pricelist/prod
where every $x in /prices/pricelist/prod[not(discount)]/price
        satisfies $x < $p/price
return <prod num = "{$p/@num}"/>
```

XQuery: Expresiones Cuantificadas

- **Repaso**: el procesador prueba la expresión `satisfies` para cada ítem en la secuencia.
  - Si el cuantificador es `some`, retorna `true` si la expresión `satisfies` es `true` para alguno de los ítems.
  - Si el cuantificador es `every` retorna `true` si la expresión satisfies es `true` para todos los ítems.
- **¿Si no hay ítems en la secuencia qué sucede?**

XQuery: Expresiones Cuantificadas

- **Repaso**: el procesador prueba la expresión `satisfies` para cada ítem en la secuencia.
  - Si el cuantificador es `some`, retorna `true` si la expresión `satisfies` es `true` para alguno de los ítems.
  - Si el cuantificador es `every` retorna `true` si la expresión satisfies es `true` para todos los ítems.
- **¿Si no hay ítems en la secuencia qué sucede?**
  - una expresión con `some` siempre retorna `false`,
  - una expresión con `every` siempre retorna `true`.

XQuery: Expresiones Cuantificadas

- Se puede usar la función `not()` con una expresión cuantificada para expresar **ninguno** y **no todos**.
- **Ejemplo**: ningún ítem en una orden es del departamento de accesorios

```
not(some $dept in doc("catalog.xml")//product/@dept satisfies ($dept = "ACC"))
```

XQuery: Expresiones Cuantificadas

- Se pueden ligar múltiples variables en una expresión cuantificada separando las cláusulas por coma. Como con las cláusulas `for` de FLWORs el resultado es que toda combinación de los ítems en las secuencias es considerado.
- **Ejemplo**: la siguiente expresión evalúa a true.

  some $i in (1 to 3), $j in (10, 11)

  satisfies $j - $i = 7

XQuery: distinct-values

- La función **`distinct-values`** elige distintos valores atómicos de una secuencia.
  - Se determina si dos valores son distintos basados en la igualdad de sus valores usando el operador **`eq`**.
  - distinct-values acepta solo una secuencia de valores atómicos, no varias secuencias de múltiples valores.
- **Ejemplo**: Retornar todos los distintos valores del atributo `dept`.

XQuery: distinct-values

- La función **`distinct-values`** elige distintos valores atómicos de una secuencia.
  - Se determina si dos valores son distintos basados en la igualdad de sus valores usando el operador **`eq`**.
  - distinct-values acepta solo una secuencia de valores atómicos, no varias secuencias de múltiples valores.
- **Ejemplo**: Retornar todos los distintos valores del atributo `dept`.
  - distinct-values(doc("catalog.xml")//product/@dept)
- Es también común elegir un conjunto de combinaciones distintas de valores.
  - Uno de los beneficios mayores de los FLWORs es que pueden fácilmente combinar datos de varias fuentes.

XQuery: distinct-values

- **Ejemplo**: elegir a partir de `catalog` los distintas combinaciones de `department/product`.

XQuery: distinct-values

- **Ejemplo**: elegir a partir de `catalog` los distintas combinaciones de `department/product`.

```
let $prods := doc("catalog.xml")//product
for $d in distinct-values($prods/@dept),
$n in distinct-values($prods[@dept = $d]/number)
return <result dept="{$d}" number="{$n}"/>
```

- **¿Qué resultados arroja esta consulta?**

```
<result dept="WMN" number="557"/>
<result dept="ACC" number="563"/>
<result dept="ACC" number="443"/>
<result dept="MEN" number="784"/>
```

XQuery: tokenize

- La función

  `tokenize(xs:string?, xs:string) as xs:string*`

  divide un string en una secuencia de tokens.
  - El primer argumento es un string de entrada y el segundo argumento es un delimitador que separa tokens.
  - `tokenize` divide el string de entrada en cada ocurrencia del delimitador, resultando en una secuencia de `substrings (tokens)`, excluyendo el delimitador en si mismo.
  - Cada vez que el delimitador aparece al comienzo o fin del string de entrada el string vacío es agregado al comienzo o al final de la secuencia resultado.
  - **Ejemplo**: tokenize(“hello, world”, “, “) => (“hello”, “world”)
  - **Ejemplo**: tokenize(“azul verde rojo”, “ “) => (“azul”, “verde”, “rojo”)

XQuery: Following IDREF's

- **`fn: id($arg as xs:string*) as element()*`**
  - Retorna la secuencia de nodos elemento que tienen un valor ID que se corresponde con el valor de uno o más de los valores IDREF provistos en `$arg`.

XQuery: Following IDREF's

- Considerar el siguiente DTD

```
<!DOCTYPE BARS[
    <!ELEMENT BAR (ADDRESS)>
    <!ATTLIST BAR NAME ID #REQUIRED >
    <!ELEMENT ADDRESS #PCDATA >
    <!ELEMENT SELLS EMPTY>
    <!ATTLIST SELLS theBeer IDREF #REQUIRED>
    <!ATTLIST SELLS price CDATA #REQUIRED>
    <!ELEMENT BEER EMPTY>
    <!ATTLIST BEER name ID #REQUIRED>
    <!ATTLIST BEER soldBy IDREFS #REQUIRED>
]>
```

XQuery: Following IDREF's

Ejemplo: Encontrar los bares que venden cervezas que cuestan menos de 3.00.

XQuery: Following IDREF's

Ejemplo: Encontrar los bares que venden cervezas que cuestan menos de 3.00.

```
let $B := /BARS/SELLS[@price < 3.00]/(id(@theBeer))
for $C in distinct-values($B/tokenize(@soldBy,“ ”))
return <BAR name =“{$C}” />
```

XQuery: Order-By Clauses

- FLWR es realmente FLWOR: una cláusula **order-by** puede preceder el return.
- Sintaxis: **`order by`** <expression>
  - Opcionalmente: **`ascending`** o **`descending`**.
- La expresión es evaluada para cada elemento de salida.
- Determina la ubicación en la secuencia de salida.

XQuery: Order-By Clauses

- **Ejemplo**: retornar los clientes ordenados por nombre.

XQuery: Order-By Clauses

- **Ejemplo**: retornar los clientes ordenados por nombre.

```
for $c in /bank/customer
order by $c/customer_name
return <customer> { $c/* } </customer>
```

  - Usar **`order by`** `$c/customer_name` para ordenar en orden descendiente.

XQuery: Order-By Clauses

- **Ejemplo**: Mostrar los ítems de todas las órdenes ordenadas por color.

XQuery: Order-By Clauses

- **Ejemplo**: Mostrar los ítems de todas las órdenes ordenadas por color.

```
for $item in doc("order.xml")//item
order by $item/@color
return $item
```

- **¿Qué resultados arroja la consulta anterior?**

```
<item dept="WMN" num="557" quantity="1" color="black"/>
<item dept="MEN" num="784" quantity="1" color="gray"/>
<item dept="WMN" num="557" quantity="1" color="navy"/>
<item dept="MEN" num="784" quantity="1" color="white"/>
<item dept="ACC" num="563" quantity="1"/>
<item dept="ACC" num="443" quantity="2"/>
```

XQuery: Order-By Clauses

- Se puede ordenar en varios niveles de anidamiento.
- **Ejemplo**: ordenar por `customer_name`, y por `account_number` dentro de cada cliente (hint: completar el ejemplo de filmina 60).

XQuery: Order-By Clauses

- Se puede ordenar en varios niveles de anidamiento.
- **Ejemplo**: ordenar por `customer_name`, y por `account_number` dentro de cada cliente (hint: completar el ejemplo de filmina 60).

```
<bank-1> {
        for $c in /bank/customer
        order by $c/customer_name
        return
            <customer>
                    { $c/* }
                    { for $d in /bank/depositor[customer_name=$c/customer_name],
                          $a in /bank/account[account_number=$d/account_number] }
                    order by $a/account_number
                    return <account> $a/* </account>
            </customer>
} </bank-1>
```

XQuery: Agregación

- Las consultas a menudo son escritas para organizar o resumir información en categorías.
- Además de simplemente reagrupar ítems es a menudo deseable realizar cálculos en grupos.
  - Esto puede hacerse usando **funciones de agregación**.
  - Ejemplos de funciones de agregación son count, sum, min, max, avg. Estas funciones operan en secuencias.
  - max y min aceptan valores de cualquier tipo que sea ordenado.
  - sum y avg aceptan valores numéricos.
  - sum, min y max tratan datos no tipados como numéricos.

XQuery: Agregación

- **Ejemplo**: para cada departamento que tiene alguna orden imprimir como atributos las siguientes informaciones: nombre del departamento, cantidad de items en ordenes del departamento, cantidad de distintos productos en items de órdenes del departamento, cantidad total de productos en ítems de órdenes del departamento. Ordenar por departamento los resultados.

XQuery: Agregación

```
for $d in distinct-values(doc("order.xml")//item/@dept)
    let $items := doc("order.xml")//item[@dept = $d]
    order by $d
    return <department code="{$d}"
    numItems="{count($items)}"
    distinctItemNums="{count(distinct-values($items/@num))}"
    totQuant="{sum($items/@quantity)}"/>
```

XQuery: Agregación

```
<department code="ACC" numItems="2" distinctItemNums="2" totQuant="3"/>
<department code="MEN" numItems="2" distinctItemNums="1" totQuant="2"/>
<department code="WMN" numItems="2" distinctItemNums="1" totQuant="2"/>
```

- Observar que el **let** arma la clase de equivalencia.

XQuery: Agregación

- La secuencia pasada a una función de agregación puede contener nodos que son strings vacíos, incluso aunque el usuario puede pensar acerca de ellos como valores faltantes,
  - puede haber casos donde se quiere que los valores faltantes sean tenidos en consideración.
- El siguiente es un ejemplo de agregación en múltiples valores.

XQuery: Agregación

- **Ejemplo**: listar ordenando primero por departamento y luego por número de producto la cantidad de ítems de ese producto para ese departamento y la cantidad de productos pedidos de ese producto para ese departamento. Se pide representar a todas las informaciones de una línea de resultado como atributos.

XQuery: Agregación

```
let $allItems := doc("order.xml")//item
for $d in distinct-values($allItems/@dept)
for $n in distinct-values($allItems[@dept = $d]/@num)
let $items := $allItems[@dept = $d and @num = $n]
order by $d, $n
return <group dept="{$d}" num="{$n}"
        numItems="{count($items)}"
        totQuant="{sum($items/@quantity)}"/>
```

- **¿Qué resultados arroja esta consulta?**

```
<group dept="ACC" num="443" numItems="1" totQuant="2"/>
<group dept="ACC" num="563" numItems="1" totQuant="1"/>
<group dept="MEN" num="784" numItems="2" totQuant="2"/>
<group dept="WMN" num="557" numItems="2" totQuant="2"/>
```

XQuery: Agregación

- Nuevamente el **let** se usa para armar las clases de equivalencia.
- El uso de la función `distinct-values` es necesario, sino corro el riesgo de construir la misma clase de equivalencia varias veces y entonces se retornará el mismo resultado varias veces.
- Además de retornar valores agregados en los resultados de las consultas se puede restringir y ordenar los resultados en los valores agregados.

XQuery: Agregación

- **Ejemplo**: idem que la consulta anterior, pero ahora ordenando por cantidad de items de ese producto para ese departamento y siempre que la cantidad de productos pedidos de ese departamento sea mayor o igual a 2 (sino no se incluye en el resultado).

XQuery: Agregación

```
let $allItems := doc("order.xml")//item
for $d in distinct-values($allItems/@dept)
for $n in distinct-values($allItems/@num)
let $items := $allItems[@dept = $d and @num = $n]
where sum($items/@quantity) > 1
order by count($items)
return if (exists($items))
    then <group dept="{$d}" num="{$n}" numItems="{count($items)}"
      totQuant="{sum($items/@quantity)}"/>
    else ( )
```

- **¿Qué resultados arroja la consulta anterior?**

```
<group dept="ACC" num="443" numItems="1" totQuant="2"/>
<group dept="WMN" num="557" numItems="2" totQuant="2"/>
<group dept="MEN" num="784" numItems="2" totQuant="2"/>
```

XQuery: Agregación

- Observar que la cláusula **where** cumple el papel de `having` de SQL.
- Observar cómo se usa la expresión **order by** .