Azzolini Riccardo 2020-04-23

XQuery

1 Interazione tra clausole `for` e `let`

Nelle espressioni FLWOR, le clausole `for` e `let` possono essere combinate in modi diversi:

- Il caso visto finora è quello di un singolo `for` seguito da un singolo `let`:

  ```
  for $x in (1, 2)
  let $y := ("a", "b", "c")
  return ($x, $y)
  ```

  Il risultato è la sequenza ottenuta dalla concatenazione di due coppie `($x, $y)`, nelle quali `$x` assume rispettivamente i valori `1` e `2`, mentre `$y` ha il valore costante `"a", "b", "c"`:

  ```
  1, "a", "b", "c", 2, "a", "b", "c"
  ```

- Invece, usando un `let` per dichiarare `$x`, e un `for` per `$y`,

  ```
  let $x := (1, 2)
  for $y in ("a", "b", "c")
  return ($x, $y)
  ```

  si fissa a `1, 2` il valore di `$x`, e quello che varia a ogni iterazione è il valore di `$y`, quindi si ottiene la sequenza:

  ```
  1, 2, "a", 1, 2, "b", 1, 2, "c"
  ```

- È anche possibile usare due `for` (nidificati):

  ```
  for $x in (1, 2)
  for $y in ("a", "b", "c")
  return ($x, $y)
  ```

  Come in molti altri linguaggi di programmazione, questi cicli `for` nidificati generano tutte le possibili coppie dei valori contenuti nelle sequenze `1, 2` e `"a", "b", "c"`:

  ```
  1, "a", 1, "b", 1, "c", 2, "a", 2, "b", 2, "c"
  ```

- Infine, con due `let`,

```
let $x := (1, 2)
let $y := ("a", "b", "c")
return ($x, $y)
```

non si ha alcuna iterazione, e il risultato è semplicemente la concatenazione delle due sequenze di partenza:

```
1, 2, "a", "b", "c"
```

Nota: Siccome il calcolo dei valori di `$y` non dipende da `$x`, in tutti questi casi l'ordine delle due clausole può essere scambiato, senza influire sul risultato. Ad esempio, l'espressione

```
for $x in (1, 2)
let $y := ("a", "b", "c")
return ($x, $y)
```

può essere scritta equivalentemente come

```
let $y := ("a", "b", "c")
for $x in (1, 2)
return ($x, $y)
```

2 Esempio di interrogazione su più documenti

Una delle caratteristiche importanti di XQuery (anche se, per la verità, essa era già presente in XPath 2.0) è la possibilità di esprimere interrogazioni su più alberi XML.

Ad esempio, si considerano il solito albero delle ricette (che si suppone sia contenuto nel file `ricette.xml`) e il seguente documento (`frigorifero.xml`):

```
<frigorifero>
  <roba>uova</roba>
  <roba>latte</roba>
  ...
</frigorifero>
```

L'interrogazione per elencare tutte le ricette che hanno come ingrediente almeno una "roba" contenuta nel frigorifero è:

```
for $r in fn:doc("ricette.xml")//rcp:ricetta
for $i in $r//rcp:ingrediente/@nome
for $s in fn:doc("frigorifero.xml")//roba[text()=$i]
return fn:distinct-values($r/rcp:titolo/text())
```

Il risultato dei tre cicli `for` nidificati è una sequenza di triple di legami alle variabili `($r, $i, $s)`, dove:

- `$r` è legata a una ricetta;
- `$i` è legata al nome di un ingrediente;
- `$s` è legata a una "roba" con lo stesso nome dell'ingrediente.

Poi, nel `return`, vengono selezionati i titoli delle ricette che usano almeno una roba presente nel frigorifero, e si usa infine la funzione `fn:distinct-values` (analoga al `DISTINCT` di SQL) per eliminare eventuali duplicati dalla sequenza risultante.

Osservazione: In sostanza, quest'interrogazione esegue un join tra i dati relativi alle ricette e la roba nel frigorifero.

3 Esempio di ristrutturazione del risultato

Come esempio che sottolinei la capacità di ristrutturazione del risultato di un'interrogazione, considerando ancora l'albero delle ricette, si vuole creare un altro albero XML che, per ogni ingrediente, elenchi tutte le ricette in cui esso è utilizzato (in pratica, si tratta di un "ribaltamento" dell'albero delle ricette):

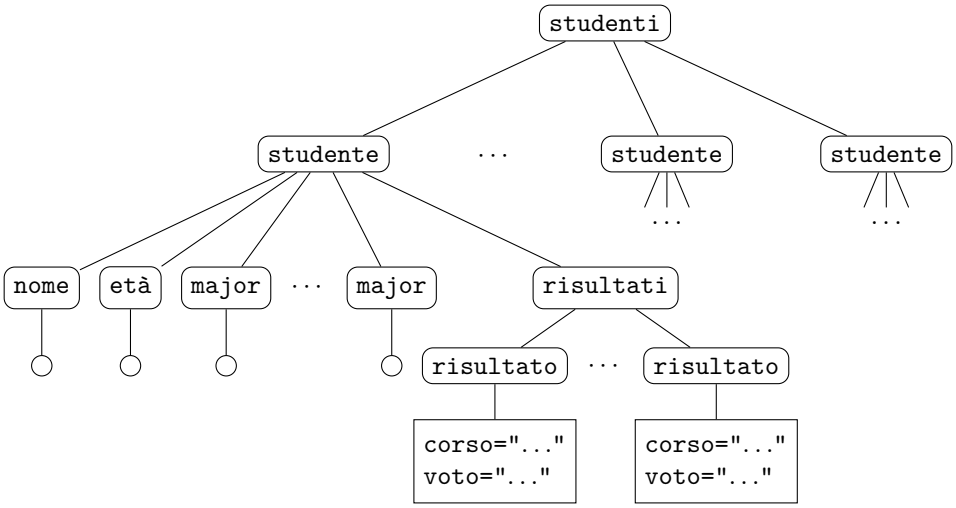
```
<ingredienti>{
  for $i in fn:distinct_values(
    fn:doc("ricette.xml")//rcp:ingrediente/@nome
  )
  return
    <ingrediente nome="{$i}">{
      for $r in fn:doc("ricette.xml")//rcp:ricetta
      where $r//rcp:ingrediente[@nome=$i]
      return <title>{$r/rcp:titolo/text()}</title>
    }</ingrediente>
}</ingredienti>
```

- Il `for` esterno genera un elemento ingrediente per ogni nome di ingrediente presente nelle ricette.
- Il `for` interno genera una sequenza di elementi `title` contenenti i nomi delle ricette che utilizzano l'ingrediente "corrente" del ciclo esterno.

4 Esempio di aggregazioni e ordinamento

L'esempio seguente mette in luce le capacità di ad aggregazione e ordinamento dei dati.

Dato l'albero XML degli studenti (`studenti.xml`),



elencare i nomi degli studenti (come semplice sequenza: qui non si vuole fare una ristrutturazione del risultato) in ordine di carriera accademica, ovvero:

1. per il numero di voti massimi (30) ricevuti, in ordine decrescente (prima gli studenti che hanno più 30);
2. in caso di pareggio, per il numero di major, in ordine decrescente;
3. infine, se si ha ancora un pareggio, in ordine crescente di età (prima i più giovani).

```
for $s in fn:doc("studenti.xml")//studente
order by
  fn:count($s/risultati/risultato[@voto="30"]) descending,
  fn:count($s/major) descending,
  xs:integer($s/age/text()) ascending
return $s/name/text()
```

Nota: Siccome il contenuto dell'elemento `age` è testuale, cioè una stringa, di default verrebbe applicato l'ordine lessicografico (ad esempio, `"7"` risulterebbe maggiore di `"20"`[1]). Allora, si usa la funzione di conversione esplicita `xs:integer` per trasformare tale stringa in un intero, in modo che l'ordinamento avvenga in base al valore numerico.

5 Sistema dei tipi

XQuery è un linguaggio tipato (ovvero è possibile determinare il tipo di un'espressione). Il sistema dei tipi di XQuery si integra con quello di XML Schema, ma ha anche degli elementi diversi.

[1] Nel caso degli studenti universitari, ci si aspetta che tutte le età siano a due cifre, quindi effettuare l'ordinamento direttamente sui valori testuali dovrebbe dare comunque il risultato desiderato.

In generale, i tipi di XQuery descrivono **sequenze**: una sequenza è un multi-insieme ordinato di **item**. Ciascun item può essere un **nodo** o un **valore atomico**.

I tipi di nodi sono di fatto analoghi a quelli dell'XPath Data Model:

- documento;
- elemento;
- attributo;
- dati testuali;
- namespace;
- processing instruction;
- commento.

Invece, i tipi di valori atomici sono in tutto 24: 19 sono i tipi primitivi di XML Schema, ai quali se ne aggiungono altri 5 per descrivere dati non previsti da XML Schema (ad esempio, gli intervalli temporali).

In un'interrogazione XQuery, si può usare il **type matching** per determinare se il risultato un'espressione è di un determinato tipo:

espr `instance of` *tipo*

Ad esempio:

```
fn:doc("ricette.xml")//ingrediente instance of element()
```

Con questo meccanismo, si può controllare che (parte di) un documento (albero) XML sia valido rispetto ai tipi di XQuery.

6 Punti deboli

Il linguaggio XQuery ha alcuni punti deboli, che, in pratica, ne hanno limitato l'adozione:

- è complicato;
- è incompleto per quanto riguarda la capacità di update (modifica dei dati, ad esempio nel contesto di un database);
- essendo (concettualmente) una generalizzazione di SQL, la sua applicazione principale avrebbe dovuto essere come linguaggio di interrogazione all'interno di DBMS XML nativi, ma questi ultimi non hanno avuto successo.

In generale, XML ha avuto successo come modello dati per l'interscambio, ma invece, nell'ambito della persistenza dei dati non è riuscito a scalzare il modello relazionale. Tuttavia, come già avvenuto per il modello object-oriented, sono state aggiunte funzionalità XML ai DBMS relazionali, e, in particolare, XQuery è stato integrato in SQL.