INDICE

INDICE DELLE FIGURE

INDICE DELLE TABELLE

INTRODUZIONE

L'evoluzione dei linguaggi per la creazione e la gestione dei database ha portato alla luce, negli ultimi anni, un approccio che si può definire "rivoluzionario".
Si è infatti passati dalla gestione dei dati secondo il modello relazionale a quella definita dal modello semi-strutturato.

Nel modello relazionale non viene richiesta all'utente la conoscenza della struttura del database, per questo motivo il modello relazionale può essere visto come un "modello statico".
Al contrario il modello semi-strutturato, introdotto con la creazione dell'XML, è conosciuto come "modello dinamico", come definito dal nome stesso del linguaggio: eXtensible Markup Language.

A causa delle enormi differenze di approccio tra i due modelli non è più possibile utilizzare il linguaggio di interrogazione "storico" per i database relazionali, cioè SQL 92, ma si sta sviluppando un nuovo linguaggio per interrogare file XML denominato XQuery (XML Query Language), attualmente disponibile nella versione 1.0.

1 – XML

XML è un linguaggio ideato nel 1996 dal W3C (World Wide Web Consortium) come evoluzione del linguaggio standard per le pagine web, ovvero l'HTML.
La differenza tra i due linguaggi risiede nella libertà fornita ai progettisti attraverso un approccio "personalizzato", essi possono infatti definire i propri tag, senza dover rispettare quelli base forniti dal linguaggio HTML.

L'estensibilità introdotta dall'XML e la sua derivazione dall'HTML non consentono però di sostenere che quest'ultimo verrà sostituito dal nuovo linguaggio, in quanto, nonostante l'XML sia un'evoluzione dell'HTML, la sua concezione è mirata ad un compito sostanzialmente diverso dal suo predecessore.
Negli ultimi anni si sta andando verso una combinazione tra XML ed HTML: il primo sarà utilizzato per definire i dati, mentre il secondo per la visualizzazione degli stessi.

1.1 – Namespace in XML

La libertà concessa al progettista di definire gli elementi con nomi a loro scelta può portare alla creazione di conflitti tra i nomi degli elementi in diversi documenti XML.

Riportiamo tabella 1.1 due elementi con lo stesso nome che possono portare a conflitti (qualora vengano aggiunti insieme), nonostante definiscano elementi diversi:

```
<Contenuto>
  <Studente>
    <Nome>Mario Rossi</Nome>
    <Matricola>001</Matricola>
  </Studente>
</Contenuto>
```

```
<Contenuto>
  <Libro>XQuery.The XML Language</Libro>
  <Libro>XQuery from the Experts</Libro>
</Contenuto>
```

Tabella 1.1: Conflitti sui tag XML

Come si può notare il tag “Contenuto” viene utilizzato per definire elementi diversi.

Per evitare la possibile creazione di conflitti si può procedere in questo modo:

```
<s:Contenuto>
  <s:Studente>
    <s:Nome>Luca Rossi</s:Nome>
  </s:Studente>
</s:Contenuto>
```

```
<l:Contenuto>
  <l:Libro>The XML Language</l:Libro>
  <l:Libro>XQuery from the Experts</l:Libro>
</l:Contenuto>
```

Tabella 1.2: Evitare conflitti sui tag XML

Tramite l’inserimento di un prefisso (separato dal nome dell’attributo o dell’elemento dal carattere “:”) è possibile definire i diversi ambiti degli elementi, ciò consente di eliminare la possibile creazione di conflitti.

1.2 –Descrizione della struttura di un file XML

Per definire gli elementi legali e la struttura di un file XML sono disponibili due diversi approcci: il DTD e l'XML Schema.
Il DTD (Document Type Definition) consente di definire, in un file esterno di estensione ".dtd", i vincoli relativi ad un documento XML.
La definizione dei vincoli di un documento XML non è indispensabile per la sua creazione, tuttavia è consigliabile la sua realizzazione, sia per ottenere una verifica sui dati che per rendere più semplice la comprensione della struttura e dei contenuti del documento, qualora un altro progettista si trovi in futuro a lavorare sullo stesso documento.
Si è però sviluppato un nuovo tipo di approccio alla definizione degli elementi legali di un documento XML, si tratta di XML Schema.
Quest'ultimo ha una sintassi basata su XML, il che rende migliore la definizione della struttura del file, dimostrandosi dunque uno strumento più efficace del DTD.

1.3 – Sintassi XML

Un documento di tipo XML deve avere gli elementi o gli attributi contenuti tra due tag:

- quello iniziale (definito tramite la sintassi <tag>);
- quello finale (definito tramite la sintassi </tag>).

Il W3C definisce alcune regole da rispettare, che sono comunque molto inferiori rispetto ai vincoli posti per i database relazionali.
Alcune restrizioni riguardano la presenza di caratteri accentati (sia nei tag che negli elementi/attributi), o di spazi nei tag.

Esistono quattro tipi di elementi a seconda del contenuto, come illustrato nella tabella 1.3.

TIPO	**CONTENUTI**
Element content	Altri elementi
Mixed content	Sia testo che altri elementi
Simple content	Testo
Empty content	Vuoto

Tabella 1.3: Tipi di elementi

Vediamo ora un esempio che illustra la struttura di un documento XML:

```
<Iscritti>
        <Persona>
                <Nome>Luca</Nome>
                <Cognome>Rossi</Cognome>
                <E-mail>Rossi@mail.it<E-mail>
        </Persona>
        <Persona>
                <Nome>Mario</Nome>
                <Cognome>Verdi</Cognome>
                <WebSite>www.ricercatori.it/Verdi</WebSite>
        </Persona>
</Iscritti>
```

La figura 1.1 rappresenta il documento XML appena introdotto:

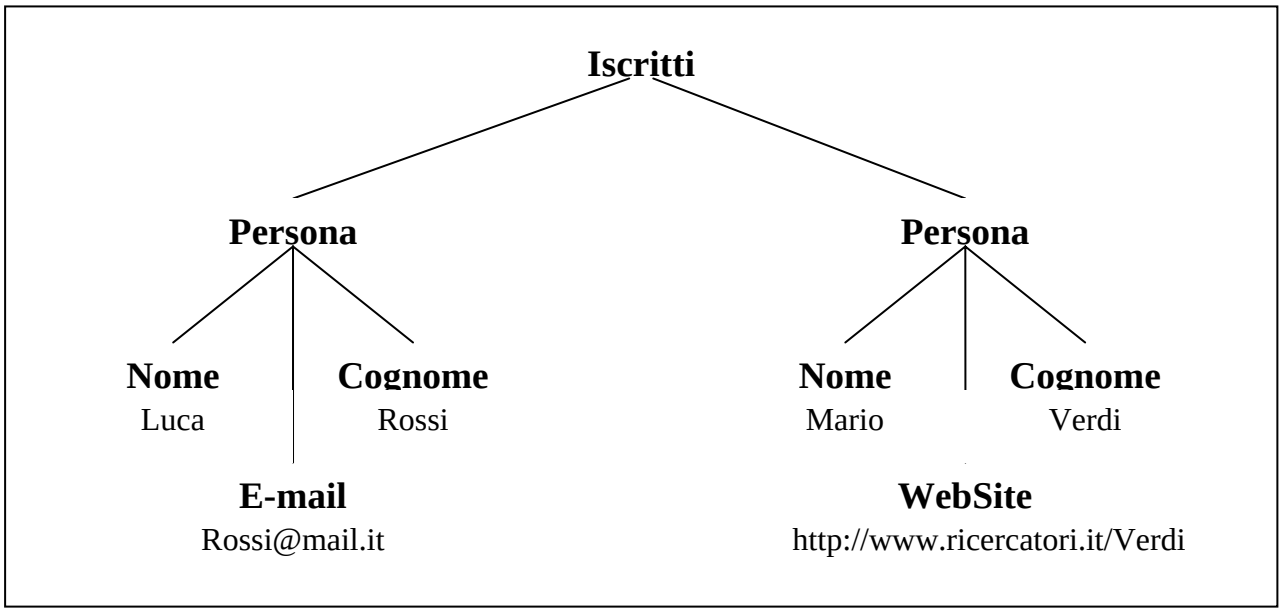

Figura 1.1: Dati Semi-Strutturati

Questo rende evidente la differenza tra database organizzati secondo il modello semi-strutturato e quelli strutturati secondo il modello relazionale.
In particolare si noti come, nonostante gli elementi “Persona” siano dello stesso tipo, essi contengono attributi diversi tra loro; ciò rende necessaria l’introduzione di un nuovo approccio alle interrogazioni su questo tipo di documento, le XQuery.

2 – XQuery

XQuery 1.0 è un linguaggio d'interrogazione per l'XML conciso ed allo stesso tempo molto flessibile ed efficace.
Esso è stato concepito a seguito di numerosi anni di lavoro tra singoli individui ed aziende di tutto il mondo; attualmente è sviluppato dal W3C.
Attualmente questo tipo di linguaggio di interrogazione si sta ancora sviluppando e solo in futuro si potrà avere un riscontro in termini di successo, le premesse sono ottime e tutto fa presagire che XQuery sarà per l'XML ciò che SQL è per i database relazionali.

2.1 – Perché XQuery?

Una domanda che ci si può porre facilmente è: "Perché XQuery anziché altri linguaggi esistenti per le Query?"; infatti esistono altri linguaggi per le Query, come ad esempio XPath, XSLT o SQL.
La domanda però risulta sbagliata in partenza, in quanto tutti i linguaggi appena citati sono ottimi per scopi diversi tra loro ed XQuery non si vuole proporre come un linguaggio che sostituisca gli altri.
La domanda però risulta interessante se viene posta in questo modo: "Quando utilizzare XQuery anziché altri linguaggi esistenti per le Query?".

La tabella 2.1 illustra quando risulta utile l'utilizzo di un linguaggio piuttosto che un altro.

LINGUAGGIO	**UTILITA'**
XPath 1.0	Utile solo quando si deve estrarre un nodo da un documento XML.
XSLT 1.0	Contiene le utilità di XPath ma è anche ottimo perché permette di trasferire file XML all'interno di documenti HTML, inoltre consente di introdurre variabili e namespace.
XQuery 1.0	Ha risultati simili ad XPath ma utilizza un approccio diverso, fornendo una grande efficienza per quanto riguarda la realizzazione di ordinamenti e di operazioni di join.
SQL	Utile quando si lavora su database relazionali.

Tabella 2.1: Utilità dei linguaggi

La definizione della struttura è forse il più grande difetto di XSLT; esso infatti è stato concepito prima dell'ideazione di XML Schema, questo non rende possibile lo sfruttamento dei vantaggi introdotti da quest'ultimo.

Da un primo sguardo alla tabella precedente sembrerebbe che XQuery sia preferibile rispetto ad XSLT ma ciò non è vero, o almeno non del tutto, in quanto XQuery si concentra sulla creazione di documenti XML e non sui documenti HTML e di testo.
Tuttavia XQuery ha un potenziale più elevato e un raggio d'azione più ampio rispetto ad XSLT, ma quest'ultimo, essendo un linguaggio presente da più tempo, ha un'implementazione più solida rispetto ad XQuery.
Con il tempo si pensa di superare questo problema, in quanto le potenzialità di XQuery saranno maggiormente evidenziate e portate alla luce attraverso la sua progressiva diffusione ed il conseguente perfezionamento.

2.2 – Struttura base di un'interrogazione XQuery

Come già detto in precedenza XQuery è un linguaggio molto conciso, la sua struttura risulta quindi di facile comprensione ed attuazione da parte dei programmatori.
Nei seguenti paragrafi verranno illustrate la struttura ed i costrutti utilizzati nelle XQuery su cui saranno effettuate le analisi.

La struttura base di un'XQuery è riconducibile a quella utilizzata con il linguaggio SQL, la sua sintassi è la seguente:

for $a in doc("file.xml")/PercorsoNodo[valore del nodo ricercato]
[where (condizioni)]
return
<Risultati>
{elementi da visualizzare nel risultato}
</Risultati>

Il risultato sarà visualizzato in un file di tipo XML con i nodi definiti dopo il comando "return".

Il costrutto "for … in" può essere sostituito dal costrutto "let … :="; nella tabella 2.2 viene visualizzato un confronto tra i due costrutti.

COSTRUTTO "FOR … IN"	**COSTRUTTO "LET … :="**
for $a in doc(<A/>,<B/>,<C/>) *return* *<Risultato>$a</Risultato>*	*let $a := doc(<A/>,<B/>,<C/>)* *return* *<Risultato>$a</Risultato>*
RISULTATO:	**RISULTATO:**
<Risultato> <A/> </Risultato> <Risultato> <B/> </Risultato> <Risultato> <C/> </Risultato>	<Risultato> <A/> <B/> <C/> </Risultato>

Tabella 2.2: Confronto tra for e let

La differenza tra i due costrutti risiede nell'iterazione dell'espressione:

- il costrutto "let … :=" associa alla variabile del risultato una tupla con tutti i valori associati ad essa;
- con l'utilizzo del costrutto "for … in" alla variabile vengono associati, singolarmente, tutti i valori presenti, con iterazione.

Passiamo ora alle condizioni utilizzate nella clausola "where".

2.3 – Clausola "where"

La clausola "where" ha lo stesso significato di quella omonima utilizzata in SQL, cambiano però le funzioni messe a sua disposizione.
Introduciamo ora quali sono le funzioni utilizzate al suo interno nelle XQuery che verranno prese in esame:

- "some … in … satisfies …",
- contains,
- exists.

Prima di procedere ad una spiegazione delle funzioni introdotte è bene precisare che "contains" ed "exists" possono essere utilizzate anche nella clausola "return", qualora in essa siano presenti espressioni condizionali.

2.3.1 – “some … in … satisfies …”

Questo tipo di funzione non ha corrispondenze nel linguaggio SQL e restituisce un valore di tipo “Boolean” (o true o false).
Il risultato sarà “true” quando alcuni valori della variabile introdotta dopo “some” soddisfa le condizioni espresse dopo “satisfies”, in caso contrario ritornerà “false”.
A titolo esemplificativo vediamo un’interrogazione che contiene questo costrutto:

```
for $a in doc("Prodotti.xml")//Articolo
where some $b in $a//giacenze satisfies($b=5000)
return
…
```

In questo caso il risultato sarà “true” se esiste almeno un elemento $b contenuto nel nodo “giacenze” della variabile $a che ha come valore “5000”.

É bene precisare che esiste anche il costrutto “where every … in … satisfies(condizione)”, il quale ritorna come risultato “true” solo se tutti gli elementi soddisfano le condizioni elencate, è infatti sufficiente un solo valore che non rispetta le condizioni per rendere il risultato “false”.

2.3.2 – Contains

Questa clausola è riconducibile al “LIKE” di SQL e serve per controllare se in un determinato nodo è contenuto un valore.
Riportiamo un esempio dove viene utilizzata la funzione contains:

```
for $a in doc(“Iscritti.xml”)//Professione
where contains($a,”Studente”)
return
…
```

In questo caso verranno selezionati i nodi del file “Iscritti.xml” in cui la professione è rappresentata dalla stringa “Studente”.

Mostriamo ora come si integrano le funzioni contains e “some … in … satisfies”:

```
for $a in doc(“Iscritti.xml”)//Persona
where some $b in $a//LingueConosciute
                    satisfies( contains($b,“Italiano”) or contains($b,”Inglese”))
return
…
```

Nell’esempio appena citato si selezionano le persone che hanno nel nodo “LingueConosciute” o la stringa “Italiano” o la stringa “Inglese”.

2.3.3 – Exists

La funzione exists serve per verificare l’esistenza di un nodo, può valere il viceversa ponendo il suffisso “not” prima della funzione stessa.
Vediamo ora un esempio dove vengono utilizzate sia la funzione “exists” che la funzione “not exists”:

```
for $a in doc(“Iscritti.xml”)//Persona
where exists($a//E-mail) and not exists($a//WebSite)
return
…
```

In questa situazione la Query filtra tutte le persone iscritte per le quali si ha l’indirizzo di posta elettronica, ma che allo stesso tempo non possiedono alcun riferimento per quanto concerne la pagina web personale.

Esiste poi un’alta gamma di funzioni che si possono utilizzare all’interno della clausola “where”, ma che ai fini delle XQuery che verranno testate non risultano utili quindi non saranno trattate.

2.4 – Return

Una volta formulata un’interrogazione attraverso i costrutti precedentemente elencati si deve procedere alla visualizzazione del risultato voluto.

A tal fine si usa l'espressione return, appartenente alle espressioni FLOWR così come le espressioni for, let, where viste in precedenza e come l'espressione "order by" che verrà trattata più avanti.

All'interno di return risiedono sia la struttura secondo la quale verranno mostrati i risultati che i nodi o gli elementi che si vorranno visualizzare.

Andiamo ora ad esaminare alcuni aspetti fondamentali della funzione return.

2.4.1 – Regole della funzione return

Esistono alcune regole da rispettare nella formulazione del risultato di un'interrogazione mediante l'utilizzo della funzione return, elenchiamo ora le principali:

- ogni tag utilizzato deve essere chiuso dopo averne elencato il contenuto ed esso non può contenere spazi o caratteri accentati nella sua dichiarazione;
- tramite l'utilizzo delle parentesi graffe si indica che sarà visualizzato il valore corrispondente alle variabili indicate tra esse, in caso contrario il valore che risulterà in output sarà esclusivamente quello contenuto tra i tag;
- per la visualizzazione di un attributo costituito da un elemento di testo dopo il nome della variabile andrà inserita la seguente stringa "/text()";

Prendendo in considerazione gli ultimi due punti vediamo, attraverso la tabella 2.3, come risultano diversi gli output di due interrogazioni apparentemente simili:

XQUERY	**RISULTATO**
for $a in doc("Iscritti.xml")//Persona* *return* *<Cognome>* *{$a//Cognome/text()} *</Cognome>*	<Cognome> Rossi </Cognome> <Cognome> Neri </Cognome>
for $a in doc("Iscritti.xml")//Persona *return* *<Cognome>* *$a//Cognome/text()* *</Cognome>*	<Cognome> $a//Cognome/text() </Cognome>

Tabella 2.3: Funzione return

Il semplice confronto appena effettuato mette in evidenza l'importanza delle regole da seguire, che, pur essendo in numero ridotto, definiscono alcuni vincoli fondamentali da rispettare.

2.4.2 – Espressioni condizionali

Ad un primo sguardo la formulazione dell'espressione return può ricondurre alla "SELECT" utilizzata con il linguaggio SQL.
Tuttavia un elemento di distinzione esiste, ed è quello riguardante le espressioni condizionali che XQuery mette a disposizione all'interno della return.
É infatti possibile ritornare un risultato solo se viene verificata una condizione specificata non in where, ma direttamente nella return.
Il costrutto è del tipo "if () then … else …", e ne mostriamo di seguito un esempio che formula in modo diverso l'interrogazione vista per la funzione exists, fornendo tuttavia lo stesso risultato:

```
for $a in doc("Iscritti.xml")//Persona
return
if(exists($a//E-mail) and not exists($a//WebSite)) then
<Risultato>{$a//Cognome/text()}</Risultato>
else()
```

In questo caso il risultato in output mostrerà solo i valori della variabile $a alla quale sono associati elementi in cui esiste l'attributo "E-mail", ma non l'attributo "WebSite".
L'utilizzo delle espressioni condizionali all'interno della return è utile, in particolar modo, quando si vuole dividere il risultato in diverse parti e strutture a seconda del verificarsi o meno della condizione o delle condizioni espresse.

Come accennato in precedenza, l'unica espressione di FLOWR che non è ancora stata citata è quella riguardante l'ordinamento del risultato.

2.5 – Ordinamenti

L'ordinamento del risultato in XQuery viene realizzato con la stessa sintassi utilizzata in SQL, cioè con il comando "order by … (ascending/descending)".
Se non viene specificata la clausola di ordinamento il risultato sarà visualizzato a seconda della sequenza dei risultati dell'espressione for.

Vediamo ora un esempio riguardante l'utilizzo degli ordinamenti in XQuery:

```
for $a in doc("Iscritti.xml")//Persona
order by $a//Cognome descending
return
<Risultato>
        <Cognome>
                {$a//Cognome/text()}
        </Cognome>
        <Nome>
                {$a//Nome/text()}
        </Nome>
</Risultato>
```

In questo caso il risultato sarà ordinato in base al cognome, in modo decrescente.

2.6 – Funzioni di aggregazione

Le funzioni di aggregazione che saranno utilizzate saranno le seguenti:

- distinct-values,
- empty,
- count.

2.6.1 – Distinct-values

Per ovviare al problema della duplicazione del risultato viene utilizzata la funzione di aggregazione distinct-values.
Vediamo un esempio nel quale si utilizza tale funzione:

```
for $a in distinct-values(doc("Iscritti.xml")//Cognome)
order by $a ascending
return
<Cognome>
{$a/text()}
</Cognome>
```

Il risultato sarà composto dai soli cognomi distinti, in ordine alfabetico crescente.

2.6.2 – Empty

La funzione di aggregazione empty viene utilizzata quando è necessario sapere se un nodo è vuoto oppure no.
Nel seguente esempio vogliamo ottenere l'elenco degli iscritti per i quali non è stata inserita la data di nascita:

```
for $a in doc("Iscritti.xml")//Persona
where empty($a//DataDiNascita)
return
<Risultato>
        <Cognome>
                {$a//Cognome/text()}
        </Cognome>
        <Nome>
                {$a//Nome/text()}
        </Nome>
</Risultato>
```

2.6.3 – Count

L'ultima funzione di aggregazione da introdurre è la funzione count, la quale, come per il linguaggio SQL, serve per effettuare il conteggio di un insieme di nodi.
Il conteggio può anche essere effettuato per valori distinti di una variabile, utilizzando al suo interno la funzione di aggregazione vista in precedenza, ovvero la funzione distinct-values.

Vediamo ora come si costruisce una XQuery che effettua il conteggio degli studenti iscritti nel database:

```
for $a in doc("Iscritti.xml")//Persona[Professione = "Studente"]
let $b := doc("fibre2fashion.xml")//[Professione = $a]
```

```
return
<Risultato>
<TotaleStudenti>
{count($b)}
</TotaleStudenti>
</Risultato>
```

3 – MonetDB/XQuery

Prima di procedere con il test delle XQuery poniamo la nostra attenzione sullo strumento software utilizzato per eseguire le interrogazioni: MonetDB/XQuery.
Questo software, sviluppato dal centro olandese CWI (Centrum voor Wiskunde en Informatica), è composto da due elementi:

- server,
- client.

Nato come parte del "Pathfinder project", il suo sviluppo e l'attività di ricerca ad esso correlato si è concentrato sulla scalabilità e sulla compilazione standard.

3.1 – Scalabilità dei Database

Gli ultimi decenni di ricerca hanno fornito DBMS (DataBase Management System) che salvano, recuperano e processano un'enorme quantità di dati in modo sempre più efficiente.
Questi gestori di database sono stati sviluppati ed implementati secondo diversi modelli, tra cui citiamo quello relazionale che è di gran lunga quello di maggior successo.
Lo sforzo nella ricerca Pathfinder ha spinto questo sviluppo un passo più avanti, oltre a fornire un efficiente supporto ai database relazionali per i dati XML (e di conseguenza al loro linguaggio di interrogazione XQuery).

Per illustrare la storia in maggior dettaglio vediamo ora una sezione relativa al benchmark.

3.1.1 – Benchmark

Il benchmark scelto dagli sviluppatori XQuery per valutare la funzionalità e le prestazioni dei prototipi implementati è stato XMark.
Questo benchmark, sviluppato sotto la direzione del CWI nel 2002, misura le prestazioni utilizzando un insieme di diverse specifiche XQuery, alcune delle quali sono notoriamente complesse da implementare in quanto esplorano tutte le caratteristiche più profonde del linguaggio XQuery.

La dimensione dei documenti XML da interrogare può andare dai 100KB ai 10GB.
Molte implementazioni hanno portato a risultati soddisfacenti fino ad un massimo di 1GB, mentre MonetDB/XQuery ha ottenuto risultati positivi fino a 10GB, come illustrato in figura 3.1.

	11 MB			110 MB			1.1 GB		11 GB
Q	Galax	X-Hive	MDB/XQ	Galax	X-Hive	MDB/XQ	X-Hive	MDB/XQ	MDB/XQ
1	0,06	0,37	0,05	0,72	1,29	0,41	9,9	1,2	13
2	0,03	0,45	0,07	0,31	1,75	0,30	33,0	2,4	25
3	0,14	0,65	0,28	1,76	5,66	1,51	25,1	12,5	126
4	0,22	0,10	0,08	2,91	1,00	0,45	18,1	3,8	36
5	0,05	0,13	0,05	0,63	0,90	0,16	20,7	1,2	11
6	1,30	1,07	0,02	13,29	10,17	0,05	178,1	0,3	3
7	2,68	1,57	0,03	30,01	24,84	0,07	278,4	0,4	4
8	0,16	0,85	0,14	2,12	3,51	0,75	49,1	10,4	208
9	113,23	32,25	0,20	*DNF*	12280,66	0,87	*DNF*	12,9	289
10	1,74	5,28	0,80	18,61	442,37	5,31	*DNF*	55,0	1882
11	2,62	98,91	0,18	*DNF*	19927,29	3,48	*DNF*	872,5	*DNF*
12	1,44	23,39	0,14	*DNF*	5100,19	1,66	*DNF*	150,7	*DNF*
13	0,03	0,10	0,07	0,66	1,03	0,22	12,9	1,3	13
14	1,92	0,72	0,17	99,53	11,16	1,40	110,2	13,7	959
15	0,02	0,03	0,09	0,20	0,49	0,28	10,6	1,7	16
16	0,03	0,03	0,11	0,46	0,52	0,26	10,9	1,8	18
17	0,06	0,09	0,07	0,82	0,85	0,30	11,8	2,8	26
18	0,07	0,08	0,04	0,73	0,64	0,13	14,8	0,9	9
19	1,17	0,67	0,11	14,73	12,15	0,55	254,5	5,3	88
20	0,28	0,11	0,24	2,98	1,40	0,62	24,6	4,9	50

Figura 3.1: XMark Benchmark

La figura mostra l'XMark Benchmark (20 query da processare) per il confronto tra Galax, X-Hive e MonetDB/XQuery per documenti con dimensione variabile tra 110MB a 11GB.

I risultati sono misurati in secondi (DNF=”Did Not Finish”, utilizzato quando il DBMS non è riuscito ad eseguire correttamente l’interrogazione).

Dai risultati prestazionali ottenuti potrebbe sembrare che XMark sia stato realizzato per promuovere MonetDB/XQuery, ciò non è vero in quanto il software è stato prodotto molto tempo dopo la realizzazione del benchmark in questione.

Passiamo ora alla spiegazione dell’altro punto sul quale si è concentrato il progetto Pathfinder, cioè la compilazione standard.

3.2 – Compilazione Standard

Il linguaggio XQuery è stato definito senza una forte presenza di soluzioni software precendenti che potessero aiutarne lo sviluppo.
Per questo motivo la sua progettazione ha avuto il vantaggio di essere “pulita”, cioè di non essere contaminata da altri linguaggi esistenti al momento della sua ideazione.

Attraverso un accurato studio, il compilatore utilizzato risulta compatibile con lo schema di esecuzione di interrogazioni XQuery.

Passiamo ora ad osservare le interfacce di MonetDB/XQuery.

3.3 – Interfacce di MonetDB/XQuery

MonetDB/XQuery può essere utilizzato in diverse applicazioni, tra cui MAPI (MonetDB API) risulta la più semplice.
MAPI è un’applicazione da linea di comando che permette di eseguire XQuery e di ritrovarne il risultato in un documento XML visualizzato attraverso il Web Browser di default.

Prima di eseguire ogni operazione riguardante le interrogazioni o l’inserimento/cancellazione di documenti alla collezione dovrà esser mandato in esecuzione il server MonetDB XQuery Server (illustrato in figura 3.2), giunto alla versione 4.8.2.

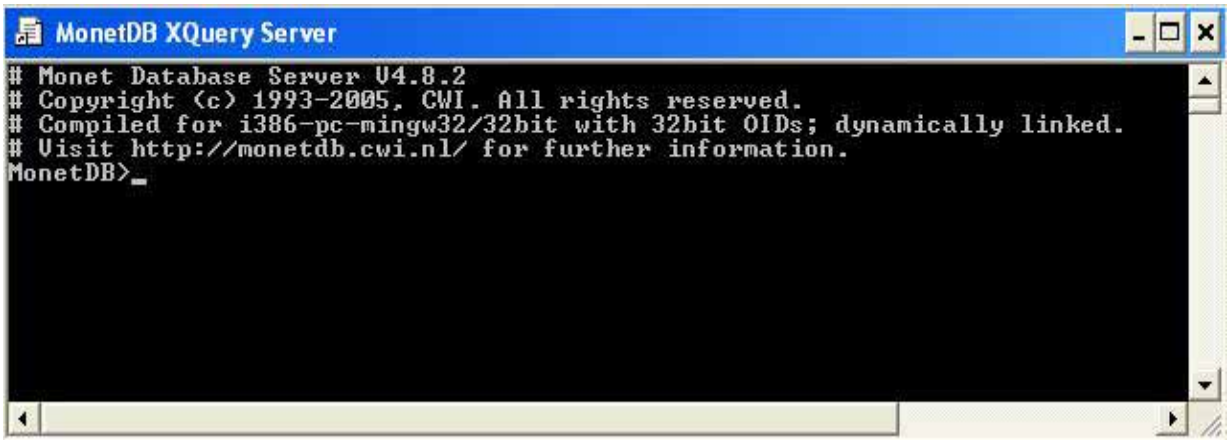

Figura 3.2: MonetDB XQuery Server

Vediamo ora come vengono salvati i documenti nella collezione apposita e come vengono processate le XQuery.

3.3.1 – Aggiungere/Cancellare documenti alla collezione

Il linguaggio XQuery non ha a disposizione comandi per aggiungere o cancellare documenti dalla collezione, quindi per poter effettuare tali operazioni è necessario utilizzare l'interfaccia MAPI con l'opzione MIL (MonetDB Intermediate Language).
Vediamo in figura 3.3 come viene aggiunto un documento di tipo XML alla collezione.
Da notare che quando si esegue il comando shred_doc("1","2") vanno inseriti come parametri:

1 - il percorso completo del file XML da aggiungere alla collezione,
2 - l'alias, ovvero il nome sotto il quale sarà disponibile nella collezione dei documenti, in questo modo quando si eseguiranno XQuery su un documento (nel caso della figura 3.3 il file HelloWorld.xml) non andrà specificato il percorso del file, ma solo l'alias.

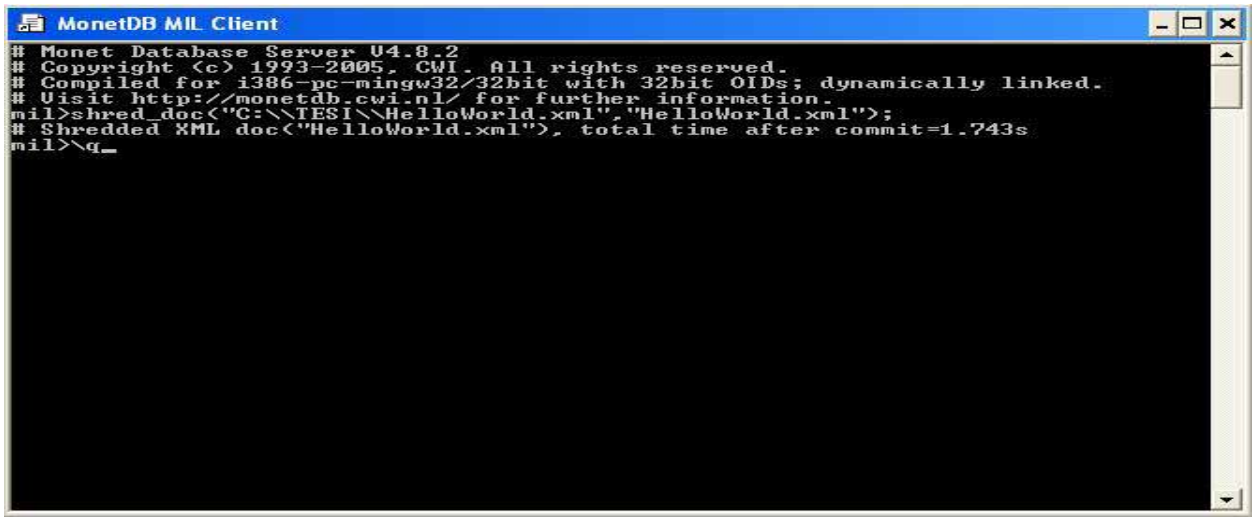

Figura 3.3: Inserimento di un file alla collezione

Da ricordare che al termine di ogni istruzione risulta necessario il carattere “;”.
In figura 3.3 viene anche mostrato il comando “\q”, utilizzato per terminare l’esecuzione di MonetDB MIL Client.

Nella figura 3.4 viene illustrata l’operazione inversa a quella appena considerata, cioè la cancellazione di un documento dalla collezione.

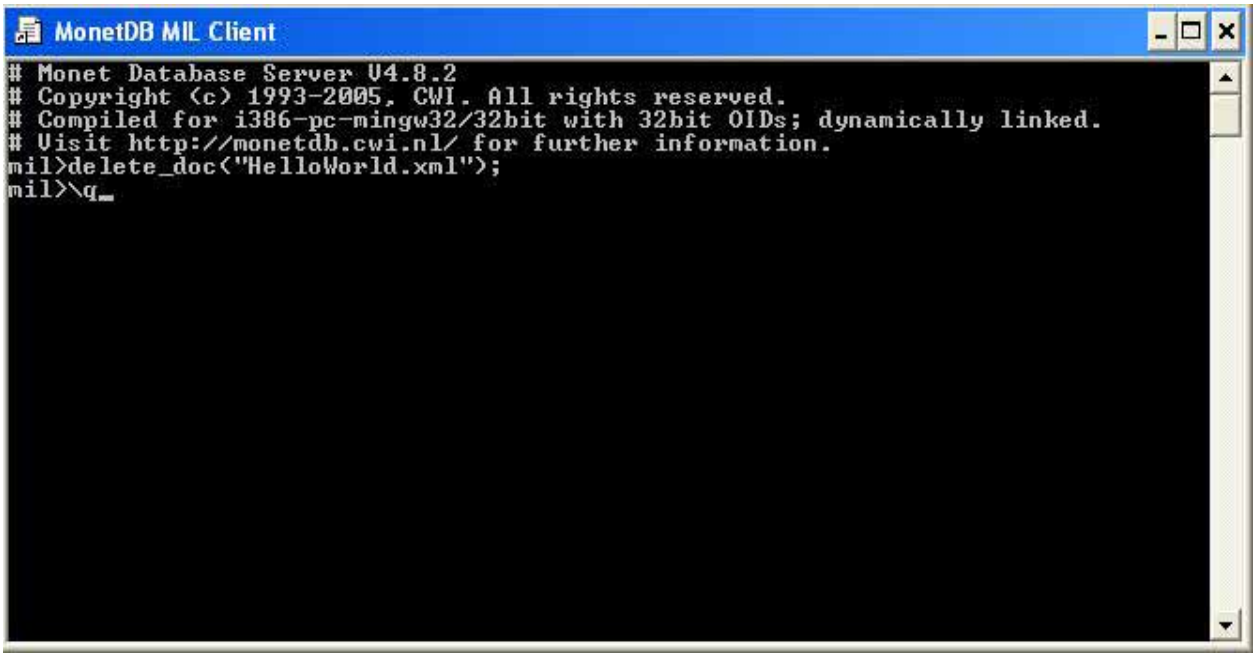

Figura 3.4: Cancellazione di un file dalla collezione

Nel caso in cui si vogliono eliminare dalla collezione tutti i documenti presenti basterà digitare il comando:

delete_all_docs();

Un ultimo comando che può essere molto utile utilizzando MonetDB MIL Client è quello relativo alla visualizzazione di tutti i documenti presenti nella collezione (figura 3.5).

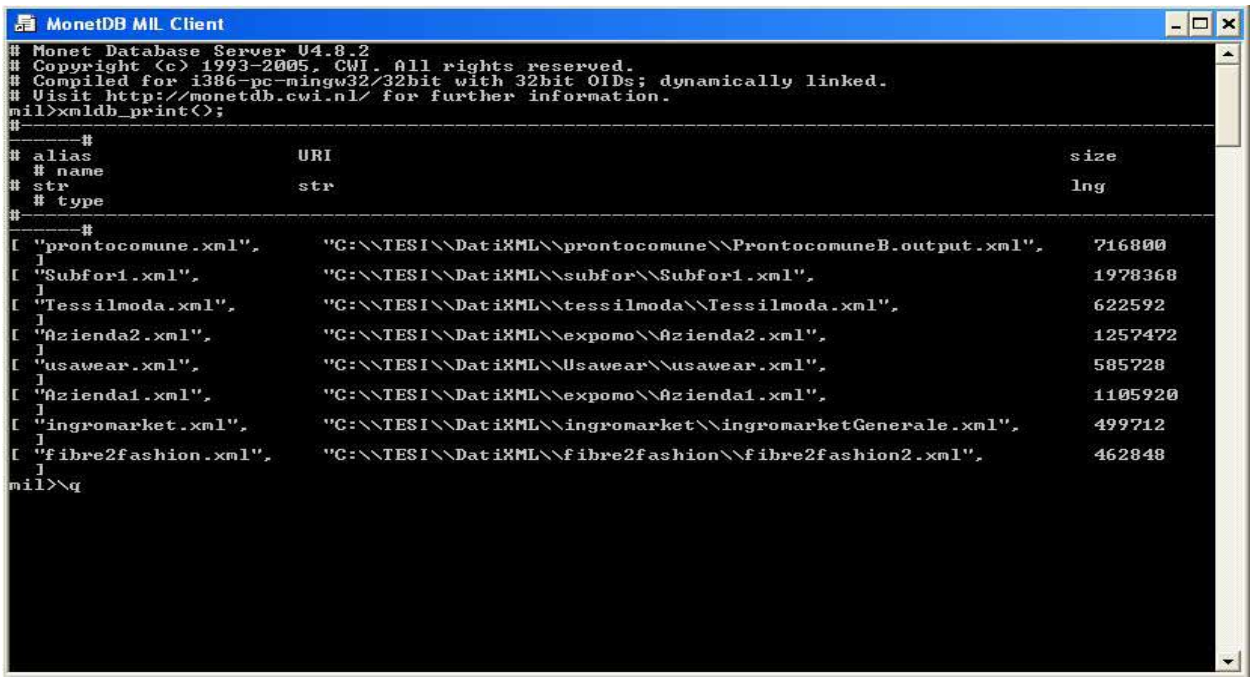

Figura 3.5: Collezione documenti

Attraverso il comando xmldb_print() vengono visualizzate, oltre ai documenti presenti nella collezione, anche alcune informazioni quali il percorso assoluto e la dimensione in byte dei file XML.

3.3.2 – Cache dei documenti

Se si prova ad eseguire un'interrogazione XQuery su un file XML non presente nella collezione dei documenti, MonetDB/XQuery proverà comunque a ricercare il file e ad aggiungerlo in modalità "on-the-fly".
Questo permette di processare XQuery su qualsiasi file XML, anche se è consigliabile aggiungere i file alla collezione mediante il comando shred_doc, in quanto il caricamento istantaneo dei documenti è ancora in fase sperimentale e non fornisce un elevato tasso di affidabilità.
Un altro svantaggio dell'attuale modo di caricamento "on-the-fly" dei documenti XML è che esso richiede un maggior numero di risorse (CPU e banda di rete), con conseguente diminuzione di prestazioni sul tempo di esecuzione dell'interrogazione.
Quando verrà processato un documento non presente nella collezione, l'equivalente dell'operazione eseguita con shred_doc sarà salvata in quella che viene chiamata "Document Cache", o cache dei documenti, con anche informazioni relative alla dimensione del file e alla data dell'ultima modifica.

Ogni volta che si eseguirà un file non presente nella collezione verrà eseguito un controllo nella cache dei documenti.
La “Document Cache” ha uno spazio limitato, indicato nel file “MonetDB.conf” presente nella cartella in cui è installato il software, per questo motivo una volta che essa risulterà piena il cache manager provvederà ad eliminare i documenti meno utilizzati della cache stessa.

3.3.3 – Esecuzione XQuery

Per eseguire un’interrogazione XQuery in ambiente MonetDB/XQuery bastano due semplici operazioni:

1 - lanciare MonetDB XQuery Server,
2 - fare doppio click sul file relativo alla XQuery.

I file contenenti le XQuery devono essere di estensione “.xq”, inoltre, come detto in precedenza, è preferibile avere il file XML da interrogare nella collezione dei documenti, per poter avere un vantaggio sia sull’utilizzo delle risorse che sulla riduzione del tempo di esecuzione relativo alla query stessa.

Al momento della richiesta di esecuzione di un’interrogazione verrà lanciato il file “MPFClient.bat” che prenderà come parametro in input il percorso relativo alla XQuery, a questo punto MonetDB/XQuery interpreta la query e può tornare tre diversi tipi di output:

- errore per mancata connessione al server (figura 3.6)

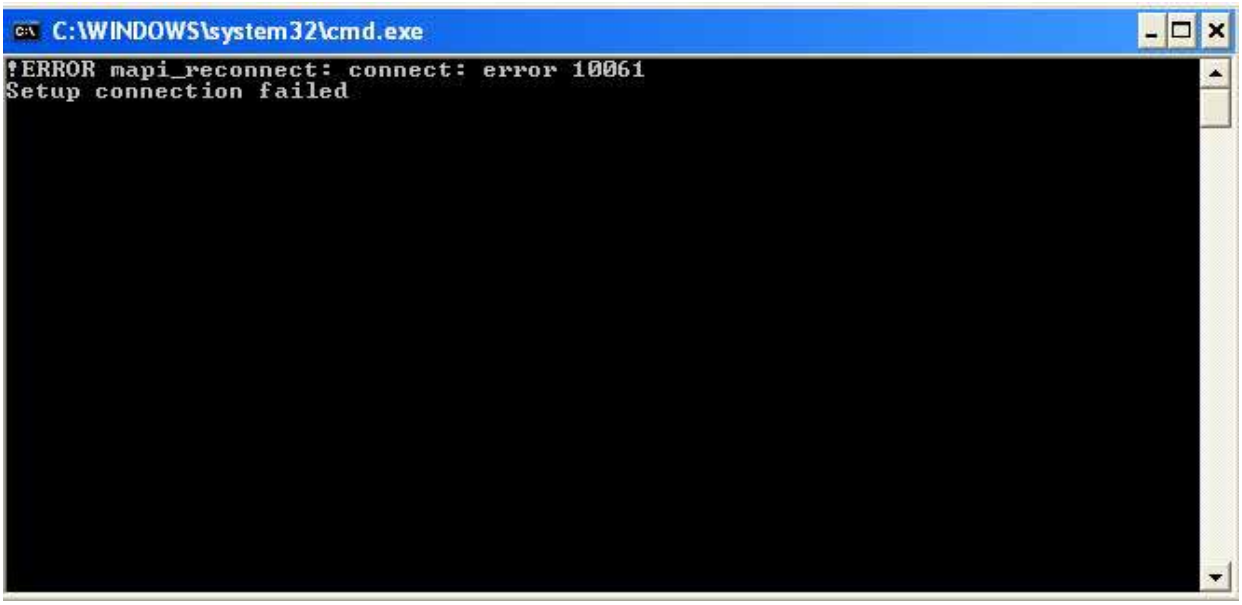

Figura 3.6: Errore connessione

- errore per sintassi non corretta all'interno dell'XQuery (figura 3.7), con conseguente chiusura del server che dovrà essere fatto ripartire

Figura 3.7: Errore XQuery

- corretta esecuzione dell'interrogazione e visualizzazione del risultato in una pagina XML, attraverso un Web Browser (figura 3.8)

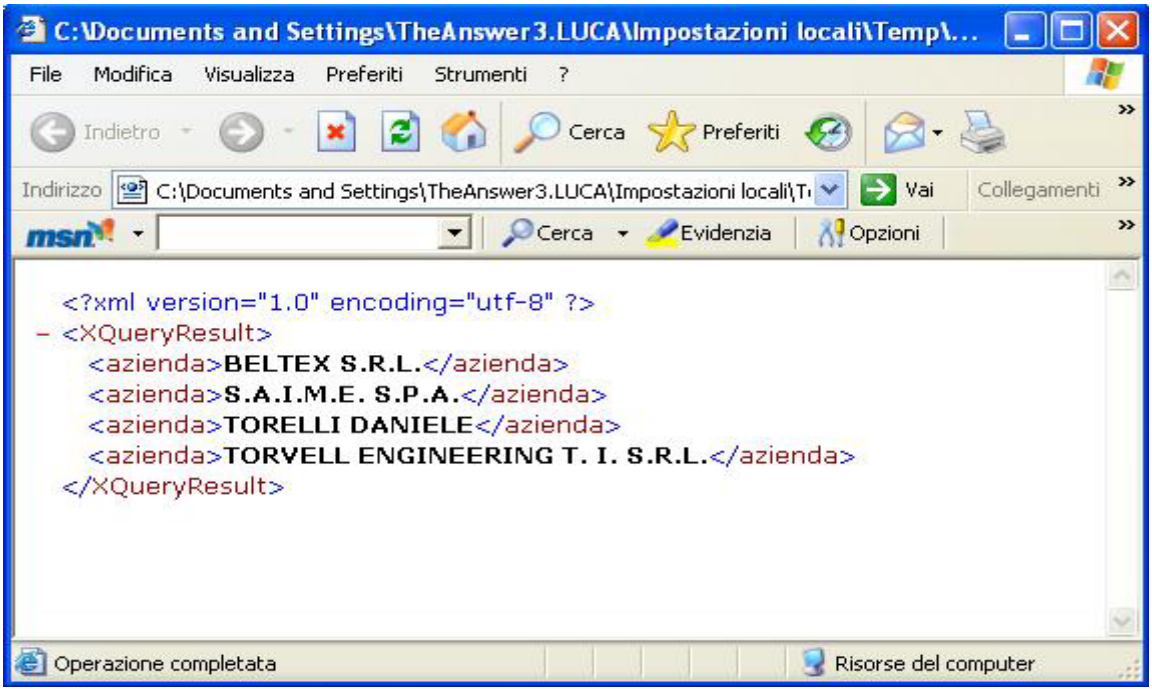

Figura 3.8: XQuery eseguita correttamente

In tutti e tre i casi citati viene creato un file XML con il risultato nella cartella temporanea dell'utente, se l'esecuzione dell'interrogazione non è andata buon fine tale documento sarà vuoto.

3.4 – Linguaggi XQuery supportati

MonetDB/XQuery è un'implementazione dello standard XQuery definito dal W3C.
Per quanto riguarda la copertura del linguaggio XQuery, si può dire che la maggior parte delle specifiche relative ad esso sono già supportate in questa release del software.
Tuttavia alcune specifiche risultano ancora mancanti, per poter conoscere quali sono già presenti, quali no e quali fanno parte della roadmap di sviluppo del software, le quali saranno disponibili nelle future release, si consulti l'Appendice B a fine elaborato.

Il linguaggio XPath risulta completamente supportato dal software, in quanto esso è un set di funzioni contenuto nel linguaggio XQuery.
Inoltre il team di sviluppo di MonetDB/XQuery sta lavorando allo sviluppo delle specifiche XUpdate, si pensa di rendere disponibili anch'esse nelle prossime release del software sviluppato dal CWI.

4 – Test ed Analisi

Procediamo ora con l'introduzione delle XQuery realizzate, dei file XML sui quali vengono effettuate tali interrogazioni e, successivamente, al test delle query stesse e all'analisi dei risultati ottenuti.
Ricordiamo che per l'esecuzione di tali test è stato utilizzato un PC Portatile Toshiba Satellite A10, le cui caratteristiche più significative, per quanto riguarda le operazioni eseguite, sono illustrate nella tabella 4.1.

CARATTERISTICHE	**COMPONENTI**
Processore:	Intel Celeron 2,20GHz
RAM:	256MB
Hard Disk:	30GB (4200RPM)
Sistema Operativo:	Windows XP Service Pack 2

Tabella 4.1: Caratteristiche Software/Hardware

Da precisare che le operazioni eseguite sono state svolte dopo aver terminato tutti i processi non necessari al normale funzionamento del computer, quindi si può affermare, in generale, che tutte le misurazioni effettuate risultano pressoché affidabili per poter procedere ad effettuare le analisi ad esse inerenti.

Introduciamo ora la descrizione relativa ai documenti XML su cui verranno effettuate in seguito le interrogazioni.

4.1 – File XML

I file XML che saranno interrogati riguardano la composizione di alcune ditte, o di articoli in vendita.
Tutti i file sono stati caricati in modo corretto nella collezione dei documenti di MonetDB/XQuery, nella tabella 4.2 vengono illustrati i dati relativi alla composizione dei database XML.
Se consideriamo l'ambiente MonetDB/XQuery, in particolare lo strumento MIL Client, riportiamo in tabella 4.3 i tempi di caricamento dei documenti nella collezione.

NOME FILE XML	RIFERIMENTO	TOTALE RIGHE	DIMENSIONE (in kByte)
Azienda1.xml	expomo	10.224	306
Azienda2.xml	expomo	9.208	365
fibre2fashion.xml	fibre2fashion	546	25
ingromarket.xml	ingromarket	1.399	54
prontocomune.xml	prontocomune	3.784	140
Subfor1.xml	subfor	16.828	965
tessilmoda.xml	tessilmoda	2.304	128
usawear.xml	usawear	1.992	87

Tabella 4.2: File XML utilizzati

La caratteristica più evidente che si può notare dalla tabella 4.3, risiede nel tempo necessario per aggiungere un file XML alla collezione dei documenti, il quale aumenta all'aumentare della dimensione (intesa come numero di elementi) dell'elemento da caricare.

Se si considera invece l'utilizzo di file attraverso la "Document Cache", i tempi di esecuzione di una XQuery risultano maggiori rispetto alla somma tra i tempi di caricamento, mediante il comando shred_doc, e dei tempi di esecuzione dell'interrogazione.

NOME FILE XML	TEMPO DI CARICAMENTO (in secondi)
Azienda1.xml	0,972
Azienda2.xml	0,881
fibre2fashion.xml	0,681
ingromarket.xml	0,721
prontocomune.xml	0,811
Subfor1.xml	1,372
tessilmoda.xml	0,801
usawear.xml	0,771

Tabella 4.3: Tempi di caricamento file XML

Vediamo ora le problematiche riguardanti la misurazione dei tempi di esecuzione delle XQuery con MPFClient.

4.2 – Problemi di misurazione dei tempi

I dati riguardanti le tempistiche di caricamento di documenti in ambiente MonetDB/XQuery vengono gestite dal software stesso, il quale, contestualmente al messaggio relativo al corretto caricamento del file XML, fornisce in output anche il tempo che è stato necessario per eseguire tale operazione.

Tuttavia la funzionalità appena citata non viene messa a disposizione per poter ottenere una corretta misurazione del tempo di esecuzione delle interrogazioni, perciò si è dovuto procedere con la creazione di un file in linguaggio C, che misura il tempo necessario per eseguire correttamente una XQuery.
Questo script in C utilizza la libreria "time.h" per poter eseguire operazioni sui tempi, infatti viene misurata la data (in secondi), sia prima che dopo l'esecuzione del programma e, successivamente, si procede con il calcolo della differenza.
Al momento della compilazione del programma mediante un qualsiasi compilatore C (nel nostro caso è stato utilizzato il compilatore Dev-C++ 4.9.9.2 della Bloodshed Software), viene creato, nella cartella in cui è presente il file C, un file eseguibile che permette di eseguire il programma successivamente senza dover eseguire tale operazione attraverso il compilatore.

Tale script viene illustrato di seguito con alcuni commenti relativi ad esso.

```
#include <stdio.h>
#include <conio.h>
#include <time.h>
#include <stdlib.h>
void main()
{ clock_t start, end;   /* in secondi dall' anno 1970 */
  double tempo;
  start = clock();      /* assegno alla variabile il valore del tempo prima del lancio */

  /* Lancio il programma... */
 system("C:\\Programmi\\CWI\\MonetDB-XQuery\\MPFclient.bat Percorso della XQuery");

 end = clock();          /* assegno alla variabile il valore del tempo dopo l'esecuzione */

 tempo = (double)(end - start) / CLOCKS_PER_SEC;
 printf("\nTempo trascorso :%6.6f sec", tempo);    /* mando in output il tempo di esecuzione */
 getch();
}
```

Introduciamo ora le XQuery sulle quali si procederà alla misurazione dei tempi ed al successivo confronto.

4.3 – Interrogazioni XQuery

Sono state realizzate e testate XQuery sui documenti XML, introdotti nei paragrafi precedenti, per un totale pari a quattordici.
Queste interrogazioni sono state suddivise in due diversi raggruppamenti:

- XQuery di selezione,
- XQuery di conteggio.

Ora saranno introdotti separatamente i due raggruppamenti con i relativi test effettuati e le considerazioni a riguardo.
Al termine di questa analisi interna ai singoli raggruppamenti, sarà effettuato un confronto tra le due tipologie di interrogazione utilizzate.

4.3.1 – XQuery di selezione

Questo primo raggruppamento, che chiameremo “Q1”, è composto da un insieme di interrogazioni che effettuano delle operazioni di ricerca e selezione di nodi, data una condizione relativa al contenuto di alcuni dei nodi che compongono il documento XML.

La struttura comune alle interrogazioni formulate è la seguente:

```
for $a in doc(“1")//2
where 3
order by 4
return
<Risultato>
{5}
</Risultato>
```

Dove i numeri indicano:

1 - l’alias assegnato al file XML da interrogare nel momento in cui il documento è stato inserito nella collezione;
2 - il percorso del nodo da assegnare alla variabile $a;
3 - la condizione che un determinato nodo deve soddisfare, espressa attraverso le funzioni di exists() o di “some … in … satisfies”, le quali sono state introdotte nel dettaglio in precedenza;
4 - l’elemento secondo il quale si ordina il risultato e il tipo di ordinamento (crescente o decrescente) voluto;
5 - gli elementi da visualizzare nel risultato e gli eventuali tag per suddividere in modo più leggibile il risultato.

Le interrogazioni facenti parte di questo primo raggruppamento sono state eseguite su tutti i documenti XML caricati nella collezione.

Procediamo ora con l’esecuzione delle interrogazioni ed alla conseguente visualizzazione dei rilevamenti effettuati sui tempi di esecuzione necessari per ognuna delle XQuery.

4.3.1.a – Test sul raggruppamento Q1

Vediamo nella tabella 4.4 le caratteristiche relative alle interrogazioni, a come esse filtrano i dati ed il tempo medio di esecuzione per il raggruppamento in esame.

XQuery	Nome File	Numero Elementi Totali	Numero Righe Selezionate	Tempo di Esecuzione (in secondi)
expomo1	Azienda1.xml	10.224	7	1,269
expomo2	Azienda2.xml	9.208	6	1,313
fibre1	fibre2fashion.xml	546	32	1,473
ingro1	ingromarket.xml	1.399	68	1,281
pronto2	prontocomune.xml	3.784	212	1,478
subfor1	Subfor1.xml	16.828	2.619	1,594
tessil2	Tessilmoda.xml	2.304	71	1,257
usa1	usawear.xml	1.992	302	1,365
		TEMPO MEDIO PER Q1		**1,379**

Tabella 4.4: Tempi di esecuzione raggruppamento Q1

I tempi relativi all'esecuzione delle singole interrogazioni si riferiscono alla media tra cinque esecuzioni delle stesse, escludendo la prima, la quale, in quanto successiva alla compilazione del programma in C, fornisce dati non affidabili e poco significativi alla valutazione che si vuole effettuare (i tempi per il primo lancio dell'XQuery risultano superiori ai tre secondi).

Passiamo ora all'analisi dei risultati ottenuti.

NOTA: Precisiamo che quando si parlerà di filtraggio si intenderà la percentuale di elementi scartati attraverso l'esecuzione dell'interrogazione.
Tale valore sarà ottenuto mediante la seguente operazione:
X = 100 – 100 *(Numero Righe Selezionate/Numero Elementi Totali)

Come si può notare i tempi di esecuzione risultano contenuti entro i due secondi, tali risultati devono però essere analizzati singolarmente prima di procedere a formulare considerazioni di insieme.

Per quanto riguarda l'esecuzione delle XQuery expomo1 ed expomo2 si può dire che la differenza in termini di tempistiche di esecuzione risiede sia nel numero delle righe filtrate, che nel numero di elementi da selezionare.

Parliamo ora della percentuale di filtraggio di tali interrogazioni; per quanto riguarda l'interrogazione expomo1 si ha un filtro di dati pari al 99,932% dei dati iniziali, mentre per quanto riguarda l'XQuery expomo2 tale filtro è pari al 99,935%.
Tuttavia tale differenza risulta molto ridotta, imputabile a possibili risorse in utilizzo per altri processi dalla CPU, per cui si può dire che i tempi possono essere considerati i medesimi per entrambe le interrogazioni.

L'XQuery denominata fibre1 è quella che opera sul file XML di minor dimensione, ed effettua un discreto filtraggio dei dati.
Questo filtraggio dei dati risulta pari al 94,139%, il quale risulta sensibilmente inferiore rispetto ai precedenti calcoli.
Questa ultima considerazione rende evidente che il tempo di esecuzione risulta più elevato rispetto alle interrogazioni fino ad ora considerate a causa del maggior numero di nodi che devono essere forniti in output.

Per quanto concerne l'interrogazione ingro1 possiamo dire che il suo tempo medio di esecuzione risulta tra i più bassi nel confronto con tutte le altre XQuery contenute nel raggruppamento attualmente in esame.
Il filtraggio relativo al file "ingromarket.xml" è del 95,139%, il quale in parte giustifica il suo basso tempo di esecuzione, il quale risulta tuttavia leggermente maggiore rispetto al tempo necessario per eseguire l'interrogazione expomo1 per motivi dovuti alla tempistica di caricamento dell'output del risultato.

L'interrogazione pronto2 esegue un filtraggio delle informazioni pari al 94,397%, questo la rende molto simile all'XQuery fibre1, ciò giustifica il maggior tempo di esecuzione rispetto alle altre interrogazioni fino ad ora esaminate.
La differenza di tempo con l'interrogazione fibre1 è dovuta al tempo di caricamento dei risultati in output.

L'interrogazione che prendiamo in esame ora è quella con il maggior tempo di esecuzione medio rispetto a tutte quelle che vengono eseguite nel raggruppamento Q1, vale a dire l'XQuery denominata subfor1.
Il suo tempo di esecuzione è pari a 1,594 secondi, con un filtraggio pari all'84,437%.
Questo ultimo valore, combinato al fatto che il documento XML su cui si effettua la selezione è quello contenente il maggior numero di elementi, rende chiaro il motivo per cui questa è l'interrogazione di Q1 con il maggior tempo di esecuzione.
Questa XQuery rappresenta la combinazione di tutte le cause di aumento del tempo di esecuzione, ma di questa caratteristica ce ne occuperemo nelle considerazioni conclusive sul raggruppamento Q1.

Passando ad analizzare l'interrogazione tessil2 si arriva a considerare l'esatto opposto rispetto all'ultima XQuery analizzata; si passa infatti da quella con il maggior tempo di esecuzione a quella che necessita del minor numero di secondi per la sua esecuzione all'interno del raggruppamento in esame.
Il suo ridotto tempo di esecuzione è dovuta soprattutto alla percentuale di filtraggio, pari al 96,918%, e la causa per la quale il suo tempo di esecuzione risulta lievemente inferiore a quello relativo alle XQuery expomo1 ed expomo2 risiede essenzialmente nello stato di occupazione della CPU al momento dell'esecuzione dei cinque tentativi relativi all'interrogazione tessil2.
Questa ultima motivazione è giustificata dal fatto che, come si può osservare nell'Appendice A situata nella parte finale dell'elaborato, in una delle cinque esecuzioni di expomo1, si è ottenuto un tempo pari a 2,143 secondi e, nella stessa situazione di esecuzione per expomo2, un tempo di esecuzione è risultato di 1,832 secondi.

L'ultima interrogazione che andiamo ad osservare è quella che risulta più vicina alla media di tutte le XQuery facenti parte del raggruppamento Q1, vale a dire la query denominata usa1.
Il valore del relativo al filtraggio è pari all'84,840% e sul suo tempo di esecuzione valgono le considerazioni enunciate anche per le altre interrogazioni.
Il suo tempo medio di esecuzione risulta tale a causa di un pressoché equo bilanciamento tra il valore di filtraggio ed il numero di elementi che vengono caricati in output.

Passiamo ora ad alcune considerazioni finali relative a ciò che è emerso dal test effettuato sulle XQuery di selezione appena esaminate.

4.3.1.b – Considerazioni finali sulle XQuery di selezione

Dalle considerazioni effettuate nel precedente paragrafo relativamente alle singole XQuery portano a definire quali sono, in generale, le cause che concorrono all'aumento del tempo di esecuzione di una XQuery di selezione.
Le cause principali sono essenzialmente tre:

- la percentuale di filtraggio dell'interrogazione;
- il numero di elementi del documento XML da interrogare;
- la dimensione dell'output.

Inoltre bisogna tenere conto delle operazioni che vengono eseguite dalla CPU, i quali, in alcune situazioni, possono portare ad un aumento non ordinario del tempo di esecuzione di un'interrogazione.

Il tempo medio di esecuzione di un interrogazione facente parte del raggruppamento Q1 è pari a 1,379 secondi.

Tale tempo è da considerarsi abbastanza soddisfacente, anche se bisogna considerare che le interrogazioni sono state eseguite su database reali, ma che sono comunque di dimensioni relativamente contenute rispetto a quelli di grandi aziende.
Questo è uno dei punti sui quali si cerca di migliorare l'ambiente MonetDB/XQuery, il quale risulta comunque il migliore tra quelli attualmente disponibili, come visto nella sezione relativa al benchmark (sezione 3.1.1).

Probabilmente le tempistiche di esecuzione possono essere ridotte con una diminuzione del tempo necessario per caricare l'output. Se si procede con un'attenta analisi si può concludere che quest'ultima problematica può essere superata con la creazione "ad-hoc" di un apposito strumento per la visualizzazione dei risultati delle XQuery con MonetDB/XQuery. Il caricamento di un Web Browser per visualizzare il risultato richiede un tempo eccessivo, che si può stimare intorno al 50% del tempo di esecuzione totale di un'interrogazione.
Conclusa l'analisi delle XQuery di selezione, procediamo ora con lo studio delle interrogazioni che effettuano delle operazioni di conteggio mediante l'utilizzo di apposite funzioni di aggregazione, vale a dire quelle di tipo contatore.

4.3.2 – XQuery di conteggio

Il secondo raggruppamento che viene preso in considerazione è quello denominato "Q2", il quale è composto da un insieme di interrogazioni che realizzano un contatore relativo ad aziende o articoli presenti in ogni categoria di cui sono composti i documenti XML che sono stati considerati nell'analisi.
Tuttavia tale operazione non è stata possibile su tutti i file XML caricati nella collezione dei documenti, in quanto, per i file "Subfor1.xml" e "usawear.xml", non si è trovato nessun nodo comune su cui poter effettuare un raggruppamento per poi utilizzare la funzione di aggregazione count() necessaria per realizzare il contatore.
La struttura comune alle interrogazioni formulate è la seguente:

```
for $a in doc("1")//2
let $b := doc("1")//3[2 = $a]
return
<Risultato>
{4}
<Totale>
{count($b//5)}
</Totale>
</Risultato>
```

Dove i numeri stanno ad indicare:

1 - l'alias assegnato al file XML da interrogare nel momento in cui il documento è stato inserito nella collezione;
2 - il percorso del nodo da assegnare alla variabile $a;
3 - il percorso del nodo da assegnare alla variabile $b, tale percorso deve fermarsi ad un livello superiore rispetto a quello della variabile $a e, attraverso l'utilizzo delle parentesi quadre, si deve specificare che il valore del nodo 2 deve essere uguale al valore assegnato alla variabile $a;
4 - gli elementi da visualizzare nel risultato e gli eventuali tag per suddividere in modo più leggibile il risultato;
5 - il percorso del nodo della variabile $b che deve essere sottoposto al conteggio, in questo caso può essere omesso e si può semplicemente contare la variabile $b, anche se nella maggior parte dei casi si specifica un nodo in particolare da contare.

Il problema più comune che può sorgere durante l'operazione di conteggio risiede nella dichiarazione della variabile $b, in quanto se siamo abituati ad effettuare interrogazioni in ambiente SQL non risulta necessaria tale dichiarazione, a causa della presenza della funzione di raggruppamento "group by" presente per SQL, ma non disponibile per quanto riguarda il linguaggio XQuery.
É bene sottolineare l'importanza del contenuto delle parentesi quadre presenti nella dichiarazione di $b, esse infatti permettono di effettuare un conteggio veritiero ed affidabile, ovviando al problema precedentemente introdotto riguardante la mancanza di funzioni di raggruppamento nel linguaggio XQuery.

Procediamo ora con l'esecuzione delle interrogazioni ed alla conseguente visualizzazione dei rilevamenti effettuati sui tempi di esecuzione necessari per ognuna delle XQuery.

4.3.2.a – Test sul raggruppamento Q2

In tabella 4.5 vengono illustrate le caratteristiche relative alle interrogazioni, a come esse filtrano i dati ed il tempo medio di esecuzione per il raggruppamento in esame.

Nel raggruppamento Q1 si consideravano le righe totali che venivano selezionate e poste in output, nel caso di Q2 ciò non risulta utile all'analisi, quindi si è passati a considerare il numero dei raggruppamenti che sono stati effettuati per il conteggio sugli elementi facenti parte di ciascuno di essi.

XQuery	Nome File	Numero Elementi Totali	Numero Raggruppamenti	Tempo di Esecuzione (in secondi)
pronto1	prontocomune.xml	3784	17	1,782
tessil1	Tessilmoda.xml	2304	8	1,283
expomo3	Azienda1.xml	10224	9	1,790
expomo4	Azienda2.xml	9208	4	1,638
fibre2	fibre2fashion.xml	546	3	1,390
ingro2	ingromarket.xml	1399	8	1,483
		TEMPO MEDIO PER Q2	**1,561**	

Tabella 4.5: Tempi di esecuzione raggruppamento Q2

Anche in questo caso i tempi relativi all'esecuzione delle singole interrogazioni fanno riferimento alla media tra cinque esecuzioni delle XQuery, escludendo la prima per gli stessi problemi del raggruppamento Q1 riguardanti la scarsa affidabilità del dato successivamente alla compilazione del programma in C che misura il tempo necessario all'esecuzione.

Passiamo ora allo studio dei risultati ottenuti.

NOTA: Precisiamo che, anche in questa situazione, quando si parlerà di filtraggio si intenderà la percentuale di elementi scartati attraverso l'esecuzione dell'interrogazione.
Tale valore sarà ottenuto mediante la seguente operazione:
X = 100 – 100 *(Numero Raggruppamenti/Numero Elementi Totali)

Come effettuato in precedenza, si procede anche per il raggruppamento Q2 ad un analisi delle singole interrogazioni, per poi poter arrivare a formulare considerazioni generali riguardanti il tipo di XQuery ora in esame.

L'interrogazione denominata pronto1, la quale opera il conteggio con un tempo elevato, ampiamente superiore al tempo medio che risulta per il raggruppamento in esame.
Tale rilevanza del tempo di esecuzione medio è dovuto a due motivi principali:

- il filtraggio, pari al 99,551%, che va interpretato in maniera diversa rispetto a prima, infatti, in questo secondo raggruppamento, esso indica che sono stati realizzati un numero elevato di raggruppamenti, quindi sono state effettuate un maggior numero di operazioni di conteggio;
- il numero dei raggruppamenti elevato, che porta ad un consistente output da caricare a video.

L’XQuery denominata tessil1 risulta quella facente parte del raggruppamento Q2 con in minor tempo di esecuzione.
Il filtraggio risulta discreto, ma il motivo a cui si può imputare il ridotto tempo medio di esecuzione risiede nel basso numero di elementi da caricare in output.
Un giusto bilanciamento tra filtraggio e dati da fornire in uscita sono la causa del basso tempo di esecuzione di quest’interrogazione.

L’interrogazione che viene ora esaminata è quella che presenta il più alto tempo medio di esecuzione, vale a dire l’XQuery denominata expomo3.
Il filtraggio è pari al 99,912% e ciò giustifica in parte il tempo di esecuzione.
Tuttavia ci sono altri due elementi che vanno a formulare tale tempo di esecuzione:

- il documento XML risulta quello di dimensioni maggiori tra quelli presi in esame per le XQuery di conteggio;
- le operazioni di conteggio risultano più lunghe.

La stretta vicinanza tra il tempo di esecuzione di pronto1 esaminato in precedenza e quello relativo a expomo3 è dovuta al fatto che, nella prima, è stato necessario un maggior tempo per il caricamento del risultato e per il numero maggiore di operazioni di conteggio, mentre nella seconda il tempo è stato elevato a causa della dimensione del documento da interrogare e dal numero dei conteggi effettuati.

L’XQuery expomo4 ha ottenuto un tempo di esecuzione non molto superiore al tempo medio per questo raggruppamento.
La causa alla base di questo tempo di esecuzione risiede nel bilanciamento tra le caratteristiche che finora abbiamo esaminato, cioè il tempo di caricamento del risultato, il filtraggio e le operazioni di conteggio da effettuare.

Il documento “fibre2fashion.xml” è quello con il minor numero di elementi, per cui il tempo medio di esecuzione dell’interrogazione formulata su di esso, denominata fibre2, decisamente sotto il tempo medio relativo a Q2.

L’ultima XQuery da considerare è ingro2 eseguita sul file “ingromarket.xml”.
Il motivo per il quale questa interrogazione risulta quella con il tempo di esecuzione più vicino alla media risulta abbastanza evidente.
Infatti, se si confrontano tutti i dati relativi a ingro2 con gli stessi relativi alle altre interrogazioni esaminate, si può notare che questa XQuery si pone al centro di tutte le altre, sia come numero di elementi del documento XML da interrogare che come numero di raggruppamenti su cui viene effettuato il conteggio e che viene fornito in output.

Anche in questa situazione è bene sottolineare come, anche se in maniera casuale e non sempre verificata, la CPU ed i processi che sono in esecuzione al momento del lancio delle interrogazioni possono influenzare le misurazioni effettuate sulle XQuery.

Per una visione dettagliata dei tempi di esecuzione di ogni tentativo di esecuzione effettuato, si consulti l'Appendice A posta a fine elaborato.

Dopo aver appena analizzato le XQuery di conteggio singolarmente, procediamo ora con le considerazioni conclusive relative alle interrogazioni del raggruppamento Q2 nel loro insieme, considerando quanto è appena emerso del test.

4.3.2.b - Considerazioni finali sulle XQuery di conteggio

Dalle analisi appena effettuate sulle singole interrogazioni risultano evidenti quali sono le cause che, per quanto riguarda le XQuery di conteggio, vanno a concorrere alla formazione del tempo di esecuzione di queste ultime.
Tali tempistiche sono imputabili a:

- percentuale di filtraggio delle informazioni;
- il numero di elementi del documento XML;
- il numero di risultati da mandare in output.

La percentuale di filtraggio ed il numero di elementi da interrogare concorrono alla formazione del tempo di esecuzione, in quanto ad essi è strettamente correlato un'altra importante caratteristica da non sottovalutare nella misurazione del tempo stesso, vale a dire le operazioni che la funzione di aggregazione count() del linguaggio XQuery deve eseguire.

Queste ultime operazioni introdotte vanno considerate sia come numero di volte che viene inizializzata la variabile di conteggio, sia per quanti elementi deve contare.
Se effettuiamo un confronto tra queste ultime due caratteristiche relative alla funzione di aggregazione count() si possono trarre le seguenti considerazioni:

- se un contatore dovrà essere inizializzato più volte, allora il tempo di esecuzione dell'interrogazione risulterà più elevato;
- se si dispone di un contatore che conteggia un determinato numero di elementi, al posto di un insieme di contatori che conteggiano, nel loro complesso, lo stesso numero di elementi, allora il tempo di esecuzione medio si riduce di circa 0,25 secondi.

Da queste ultime due considerazioni si giunge alla conclusione che, dove possibile, sarebbe preferibile effettuare interrogazioni che raggruppino al loro interno il maggior numero di nodi del documento XML che si sta esaminando.

Abbiamo ora esaminato le due tipologie XQuery formulate, procediamo ora con un confronto tra i due tipi diversi di interrogazioni analizzate, cercando di porre in evidenza quali sono le differenze strutturali e di esecuzione che portano ad un tempo medio di esecuzione diverso.

4.3.3 – Confronto tra i due tipi di XQuery

La differenza tra i tempi medi di esecuzione dei due tipi di interrogazione risulta pari a 0,182 secondi.
Tale differenza può sembrare contenuta, anche se quest'affermazione risulta errata se si considera che, solitamente, i database di grandi aziende contenuti nei file XML che dovranno essere interrogati nella realtà potranno essere decisamente più grandi rispetto a quelli esaminati in questo elaborato.
Questi ultimi non devono però essere sottovalutati, in quanto rappresentano comunque situazioni reali e la loro analisi risulta comunque affidabile se si tiene in considerazione ciò che è stato appena detto in relazione ad essi.

Nella tabella 4.6 vediamo un confronto tra le interrogazioni XQuery facenti parte dei due raggruppamenti che operano sullo stesso documento XML, con anche la relativa differenza tra i tempi medi.

XQuery	Nome File	Tempo di Esecuzione di Q1 (in secondi)	Tempo di Esecuzione di Q2 (in secondi)	Differenza (in secondi)
pronto2, pronto1	prontocomune.xml	1,478	1,782	0,304
tessil2, tessil1	Tessilmoda.xml	1,257	1,283	0,026
expomo1, expomo3	Azienda1.xml	1,269	1,790	0,521
expomo2, expomo4	Azienda2.xml	1,313	1,638	0,325
fibre1, fibre2	fibre2fashion.xml	1,473	1,390	-0,084
ingro1, ingro2	ingromarket.xml	1,281	1,483	0,202

Tabella 4.6: Differenza tra i tempi di esecuzione

Osservando la tabella 4.6 si possono formulare alcune considerazioni, che verranno elencate di seguito.

In generale le differenze risultano abbastanza evidenti, ciò è imputabile all'utilizzo della funzione di aggregazione count(), che costituisce il motivo principale per il quale l'esecuzione delle interrogazioni del raggruppamento Q2 richiede mediamente più tempo rispetto a quella delle XQuery contenute in Q1.

Si trovano però due casi che si possono definire anomali in tabella 4.6.

Il primo caso particolare che consideriamo è quello relativo alle interrogazioni eseguite sul documento "Tessilmoda.xml".
La differenza tra le due interrogazioni eseguite su tale file è decisamente inferiore rispetto alla media.
Ciò è imputabile alla maggiore differenza tra i valori del filtraggio, il quale si avvicina molto al 4% ed alla conseguente necessità di un maggior tempo per caricare l'output con il risultato dell'interrogazione.

Il secondo caso particolare che si considera è quello che coinvolge le interrogazioni eseguite sul documento "fibre2fashion.xml".
Il calcolo della differenza tra i tempi di esecuzione porta ad un risultato negativo ed è l'unico caso in cui ciò accade.
Il motivo per cui si ha un tempo di esecuzione più alto nell'interrogazione di selezione che in quella di conteggio può essere interamente imputato al caricamento dell'output ed all'utilizzo del contatore.
Il documento in questione è infatti quello di dimensioni più piccole ed è quello che fornisce il minor numero di raggruppamenti in output per quanto riguarda l'XQuery di tipo contatore e per questo motivo il tempo per tale esecuzione risulta basso.
Il contatore viene infatti azzerato solo tre volte e l'output è composto da un numero ridotto di dati da visualizzare.
Si può dunque concludere dicendo che il fatto che l'output da caricare è maggiore per l'interrogazione di selezione, sommato alla necessità di azzerare il contatore solo tre volte in quella di conteggio, costituiscono le cause per cui la differenza tra i tempi medi di esecuzione risulta negativa.

Nella sezione 3.3.2 si è accennata la modalità di esecuzione "on-the-fly" a disposizione di MonetDB/XQuery, anche se ancora in fase di sviluppo, andremo ora a valutare come variano i tempi di esecuzione delle interrogazioni eseguendo il caricamento del documento in tale modalità, anziché nel modo standard (cioè mediante l'utilizzo del comando shred_doc della libreria "pathfinder" di MonetDB/XQuery).

4.4 – XQuery in modalità “on-the-fly”

Come introdotto in precedenza MonetDB/XQuery fornisce una modalità di caricamento dei documenti nella collezione nuova e, per questo motivo, oltre a presentare alcuni vantaggi, presenta ancora troppi svantaggi.
Tale modalità è denominata “on-the-fly” e riguarda l’utilizzo di uno spazio di memoria detto “Document Cache”.

Alcuni svantaggi sono stati citati nella sezione 3.3.2 quindi riprendiamo solo i principali.

Essendo la cache dei documenti un’opzione nuova e tuttora poco utilizzata, ad essa viene assegnato un spazio minimo (quello di default è pari a 100MB).
Questo spazio può essere comunque modificato andando ad operare sul file contenuto nella cartella “etc” della directory di installazione del software, in particolare modificando il valore assegnato alla variabile “xquery_cacheMB” nel file “MonetDB.conf”.

Lo svantaggio forse più grande relativo a questo tipo di caricamento dei documenti risiede nella volatilità della “Document Cache”, la quale, una volta esaurito lo spazio che mette a disposizione, procede con la cancellazione dei documenti in essa inserita che sono stati utilizzati il minor numero di volte.
Questo non è accettabile qualora si operi spesso su determinati documenti e se l’inserimento di nuovi file non è dovuto alla sostituzione di quelli presenti.

Il vantaggio maggiore risiede nella possibilità di eseguire interrogazioni su documenti XML qualsiasi sia la loro posizione e soprattutto quando si ritiene che su tale file saranno effettuate poche XQuery in un arco di tempo limitato.

Nella sezione seguente andiamo ad analizzare i tempi di esecuzione delle interrogazioni formulate studiate in precedenza, eseguite però in modalità “on-the-fly”.

4.4.1 – Confronto tra caricamento standard e “on-the-fly”

Il confronto che verrà ora effettuato si basa sulla comparazione tra la somma dei tempi per aggiungere il documento XML alla collezione ed il tempo per eseguire la relativa interrogazione, e la semplice esecuzione dell’XQuery senza aver aggiunto il documento mediante il comando standard shred_doc.

Nelle tabelle 4.7 e 4.8 visualizziamo i tempi, rispettivamente di Q1 e Q2, a confronto con l'esecuzione in modalità "on-the-fly".
I tempi di esecuzione in modalità "on-the-fly" sono stati ottenuti come nei casi precedenti, ovvero come media tra cinque esecuzioni escludendo la prima.

In questo caso si è proceduto con la cancellazione della "Document Cache" al termine di ogni singola esecuzione.
Per la visualizzazione nel dettaglio dei rilevamenti effettuati si può consultare l'Appendice A posta a fine elaborato.

XQuery	Nome File	Q1 (in secondi)	Tempo di Esecuzione di Q1 in modalità "on-the-fly" (in secondi)
expomo1	Azienda1.xml	2,241	2,693
expomo2	Azienda2.xml	2,194	2,709
fibre1	fibre2fashion.xml	2,154	2,841
ingro1	ingromarket.xml	2,002	2,703
pronto2	prontocomune.xml	2,289	2,867
subfor1	Subfor1.xml	2,966	3,015
tessil2	Tessilmoda.xml	2,058	2,627
usa1	usawear.xml	2,136	2,797

Tabella 4.7: Confronto Q1-"on-the-fly"

XQuery	Nome File	Q2 (in secondi)	Tempo di Esecuzione di Q2 in modalità "on-the-fly" (in secondi)
pronto1	prontocomune.xml	2,593	2,793
tessil1	Tessilmoda.xml	2,084	2,691
expomo3	Azienda1.xml	2,762	3,015
expomo4	Azienda2.xml	2,519	2,715
fibre2	fibre2fashion.xml	2,071	2,683
ingro2	ingromarket.xml	2,204	2,699

Tabella 4.8: Confronto Q2-"on-the-fly"

Dall'analisi sulle tabelle 4.7 e 4.8 risulta evidente l'attuale svantaggio delle interrogazioni eseguite "on-the-fly", rispetto all'esecuzione in modo standard delle stesse.
La differenza media è superiore al mezzo secondo per quanto riguarda il raggruppamento Q1, mentre per quanto riguarda la interrogazioni facenti parte di Q2, tale differenza risulta di circa 0,4 secondi.
Questo pone in evidenza i motivi per cui conviene ancora utilizzare l'approccio "classico", anche se probabilmente il CWI provvederà al miglioramento ed allo sviluppo del nuovo approccio in modo da renderlo, se non più efficiente, almeno competitivo con quello attualmente in uso per MonetDB/XQuery, in quanto, attualmente, non mostra alcun punto di forza a fronte di un elevato numero di punti deboli.

5 – Conclusioni

A seguito delle analisi effettuate risultano evidenti le caratteristiche e le potenzialità, sia dello strumento software utilizzato che del linguaggio di formulazione delle interrogazioni che è stato posto al centro delle nostre analisi.
Un approccio attraverso il modello semi-strutturato consente una strutturazione più flessibile delle informazioni, anche se tuttavia gli strumenti che utilizzano ed interrogano tali informazioni sono ancora in fase di sviluppo e non esistono ancora strumenti software di alto livello che ne consentano l'utilizzo ed il conseguente sviluppo nella massa.
Grandi multinazionali del software operanti nell'ambito della costruzione, manipolazione ed interrogazione di basi di dati costituiti da informazioni semi-strutturate non hanno ancora adottato completamente questa tecnologia.
Oracle e Microsoft non hanno ancora rilasciato sul mercato dei DBMS in grado di formulare XQuery o di manipolare i dati semi-strutturati in modo efficiente come per il modello relazionale; c'è però da precisare che Oracle ha rilasciato per un breve periodo un tool per le XQuery in ambiente Oracle 10g, ma dopo poco tempo esso è stato tolto dal mercato.
Tutto questo ha favorito senz'altro lo sviluppo di strumenti software come MonetDB/XQuery, il quale risulta il più efficiente attualmente disponibile, anche se presenta alcune lacune.
La prima lacuna imputabile al DBMS utilizzato è la mancanza di un'interfaccia utente per poter realizzare, compilare ed eseguire le interrogazioni.
Un altro punto debole risiede nel fatto che, ogni volta che viene eseguita una XQuery errata nella sua sintassi, si deve riavviare il server.
Un ultimo difetto, che può collegarsi alla prima lacuna di MonetDB/XQuery citata, è la mancanza di uno strumento per la visualizzazione del risultato costruito "ad-hoc" per il software stesso.

A tal fine si è potuto notare che un'influenza significativa sulle tempistiche di esecuzione di un'interrogazione è esercitata della necessità di dover caricare un Web Browser per la visualizzazione del risultato.

Come è stato evidenziato durante l'analisi, per quanto riguarda il tempo di esecuzione delle interrogazioni si può dire che, a concorrere alla formazione di tale valore vi sono diversi elementi, come ad esempio la dimensione dell'elemento da interrogare, la percentuale di filtraggio dell'informazione, la dimensione dell'output da visualizzare e il tempo necessario per ottenere l'avvio del Web Browser di default.
La differenza principale tra le interrogazioni di conteggio e quelle di selezione risiede nella necessità, da parte delle prime, di inizializzare, per ogni raggruppamento, una variabile di tipo contatore ed eseguire successivamente il conteggio.

Tuttavia il fatto che sia il software che il linguaggio XQuery siano ancora in fase di sviluppo è di incoraggiamento per il futuro, in quanto le potenzialità, soprattutto di XQuery, sembrano molto elevate.
Fonti ufficiali, provenienti dalla rete, sostengono che i maggiori produttori di DBMS del mercato, Microsoft ed Oracle su tutti, stanno studiano e realizzando strumenti software più potenti ed "user-friendly" di quelli attualmente disponibili per poter utilizzare al meglio e con maggior efficienza la famiglia di linguaggi, come ad esempio quella relativa alle XQuery.

Con il tempo e con l'aiuto delle multinazionali dei DBMS si potrà dunque permettere una maggior diffusione dei dati semi-strutturati e di tutto ciò che serve per operare su essi; questo permetterà ad XQuery di diventare ciò che è nato per essere, cioè l'equivalente, per il modello semi-strutturato, del linguaggio standard di manipolazione ed interrogazione del modello relazionale, cioè il linguaggio SQL.

BIBLIOGRAFIA

(1) http://www.w3c.org

(2) http://monetdb.cwi.nl

(3) http://www.sourceforge.com

(4) “XQuery: The XML Query Language”, Micheal Brundage; Addison Wesley Publisher; Feb. 06,2004

(5) “XQuery from the Experts: A Guide to the W3C XML Query Language”, Howard Katz Editor, Don Chamberlin, Denise Draper, Mary Fernández, Michael Kay, Jonathan Robie, Michael Rys, Jérôme Siméon, Jim Tivy, Philip Wadler; Addison Wesley Publisher; Aug. 29,2003

Appendice A

RISULTATI DEI TENTATIVI DI ESECUZIONE DELLE XQUERY

- Raggruppamento Q1

Q1	expomo1	expomo2	fibre1	ingro1	pronto2	subfor1	tessil2	usa1
Tentativo1	2,143	1,261	1,261	1,562	1,341	2,143	1,151	1,191
Tentativo2	1,041	1,832	1,582	1,201	1,462	1,542	1,141	2,093
Tentativo3	1,051	1,091	1,942	1,181	1,772	1,381	1,702	1,151
Tentativo4	1,051	1,141	1,321	1,161	1,341	1,442	1,161	1,211
Tentativo5	1,061	1,241	1,261	1,301	1,472	1,462	1,131	1,181

Tabella A1: Tentativi su Q1

- Raggruppamento Q2

Q2	pronto1	tessil1	expomo3	expomo4	fibre2	ingro2
Tentativo1	1,702	1,412	1,702	1,432	1,171	1,281
Tentativo2	1,742	1,251	1,942	2,373	2,213	2,243
Tentativo3	1,952	1,251	1,922	1,482	1,161	1,281
Tentativo4	1,702	1,191	1,722	1,492	1,231	1,331
Tentativo5	1,812	1,311	1,662	1,412	1,172	1,281

Tabella A2: Tentativi su Q2

- Raggruppamento Q1 in modalità “on-the-fly”

Q1 “on-the-fly”	expomo1	expomo2	fibre1	ingro1	pronto2	subfor1	tessil2	usa1
Tentativo1	2,641	2,751	3,011	2,791	2,751	2,951	2,591	2,811
Tentativo2	2,851	2,781	2,801	2,511	2,811	2,951	2,591	2,791
Tentativo3	2,611	2,611	2,752	2,761	2,901	3,081	2,651	2,791
Tentativo4	2,751	2,651	2,691	2,761	2,901	3,131	2,699	2,801
Tentativo5	2,611	2,751	2,951	2,692	2,971	2,961	2,602	2,791

Tabella A3: Tentativi su Q1 “on-the-fly”

- Raggruppamento Q2 in modalità “on-the-fly”

Q2 “on-the-fly”	pronto1	tessil1	expomo3	expomo4	fibre2	ingro2
Tentativo1	2,911	2,701	2,991	2,751	2,691	2,761
Tentativo2	2,791	2,851	3,011	2,691	2,691	2,761
Tentativo3	2,781	2,701	3,081	2,691	2,701	2,611
Tentativo4	2,881	2,651	2,981	2,691	2,711	2,681
Tentativo5	2,601	2,551	3,011	2,751	2,621	2,681

Tabella A4: Tentativi su Q2 “on-the-fly”

Appendice B

XQUERY SUPPORT DI MONETDB/XQUERY

Riportiamo ora l'elenco delle funzioni supportate da MonetDB/XQuery. Alcune non sono ancora implementate ed altre non sono previste nel futuro prossimo.

yes	**Order Awareness** MonetDB/XQuery correctly implements document and sequence orders, as well as node identity.
yes	**XPath Location Steps** MonetDB/XQuery implements XQuery's *full axis feature*, i.e., we support all 12 XPath axes. Note, however, that we do not support node tests on *type annotations* as introduced with newer XQuery drafts.
yes	**FLWOR clauses** We support FLWOR clauses with full generality and arbitrary nesting, including positional variables.
yes	**Arithmetics, Logics, Conditionals** We support arithmetics, logics (`and`, `or`), and conditionals (`some`/`every`, `if-then-else`).
partly	**Node Construction** We fully support element, attribute, and text constructors, at arbitrary nesting depth. We have not implemented document, comment and processing-instruction constructors, yet.
yes	**Namespaces** MonetDB/XQuery fully supports namespaces.
yes	**Schema Import** MonetDB/XQuery implements the *Schema Import* feature. This is not well tested, though.
no	**Module Import** MonetDB/XQuery does not implement the *Module Import* feature.
partly	**XQuery Built-In Functions** We support a large set of built-in functions, as listed in our Function Library. We currently do *not* support functions that involve specific collation orders, as well as functions that involve date/time conversions.
yes	**User-Defined Functions** MonetDB/XQuery correctly deals with user-defined functions, with or without recursion. Current development efforts will lead to an even more efficient implementation that completely eliminates function call overhead.
partly	**Simple Types** We currently support the XQuery simple types `xs:integer`, `xs:decimal`, `xs:double`, `xs:string`, and `xs:boolean`. Note that we currently implement `xs:decimal` as a floating point number which may lead to rounding errors.
yes	**Static Typing** MonetDB/XQuery supports the *Static Typing* feature. (For experts: we actually support *structural* typing here, using Antimirov's algorithm.)

no	**Dynamic Typing** Our implementation of XQuery's `typeswitch` clause is still very limited. We currently allow type tests that can be decided at compile time, and tests for *atomic* types. Note that the lack of dynamic typing may also limit XQuery's casting functionalities.	
no	**Validation** MonetDB/XQuery does not yet support validation, though work is underway to close that gap.	

Function Library

Aggregates

fn:count	($srcval as item*) as xs:integer	Done
fn:avg	($srcval as xdt:anyAtomicType*) as xdt:anyAtomicType?	Done
fn:max	($srcval as xdt:anyAtomicType*) as xdt:anyAtomicType?	Done
fn:max	($srcval as xdt:anyAtomicType*, $collationLiteral as string) as xdt:anyAtomicType?	won't
fn:min	($srcval as xdt:anyAtomicType*) as xdt:anyAtomicType?	Done
fn:min	($srcval as xdt:anyAtomicType*, $collationLiteral as string) as xdt:anyAtomicType?	won't
fn:sum	($srcval as xdt:anyAtomicType*) as xdt:anyAtomicType?	Done
fn:sum	($arg as xdt:anyAtomicType*, $zero as xdt:anyAtomicType?) as xdt:anyAtomicType?	Done

Numeric

fn:number	() as xs:double	Done
fn:number	($srcval as item?) as xs:double	Done
fn:abs	($srcval as numeric?) as numeric?	Done
fn:ceiling	($srcval as numeric?) as numeric?	Done
fn:floor	($srcval as numeric?) as numeric?	Done
op:numeric-add	($operand1 as numeric, $operand2 as numeric) as numeric	Done
op:numeric-divide	($operand1 as numeric, $operand2 as numeric) as numeric	Done
op:numeric-equal	($operand1 as numeric, $operand2 as numeric) as xs:boolean	Done
op:numeric-greater-than	($operand1 as numeric, $operand2 as numeric) as xs:boolean	Done
op:numeric-integer-divide	($operand1 as xs:integer, $operand2 as xs:integer) as xs:integer	Done
op:numeric-less-than	($operand1 as numeric, $operand2 as numeric) as xs:boolean	Done
op:numeric-mod	($operand1 as numeric, $operand2 as numeric) as numeric	Done
op:numeric-multiply	($operand1 as numeric, $operand2 as numeric) as numeric	Done
op:numeric-subtract	($operand1 as numeric, $operand2 as numeric) as numeric	Done
op:numeric-unary-minus	($operand as numeric) as numeric	Done

op:numeric-unary-plus	($operand as numeric) as numeric	Done
fn:round	($srcval as numeric?) as numeric?	Done
fn:round-half-to-even	($srcval as numeric?) as numeric?	not yet
fn:round-half-to-even	($srcval as numeric?, $precision as integer) as numeric?	not yet
op:to	($firstval as xs:integer, $lastval as xs:integer) as xs:integer+	Done

Boolean

fn:boolean	($srcval as item*) as xs:boolean	done
fn:false	() as xs:boolean	done
fn:not	($srcval as item*) as xs:boolean	done
fn:true	() as xs:boolean	done

Comparison

op:base64Binary-equal	($value1 as xs:base64Binary, $value2 as xs:base64Binary) as xs:boolean	not yet
fn:deep-equal	($parameter1 as item*, $parameter2 as item*) as xs:boolean	Will
fn:deep-equal	($parameter1 as item*, $parameter2 as item*, $collationLiteral as string) as xs:boolean	Will
fn:compare	($comparand1 as xs:string?, $comparand2 as xs:string?) as xs:integer?	done
fn:compare	($comparand1 as xs:string?, $comparand2 as xs:string?, $collationLiteral as xs:string) as xs:integer?	done
op:boolean-equal	($value1 as xs:boolean, $value2 as xs:boolean) as xs:boolean	done
op:boolean-greater-than	($srcval1 as xs:boolean, $srcval2 as xs:boolean) as xs:boolean	done
op:boolean-less-than	($srcval1 as xs:boolean, $srcval2 as xs:boolean) as xs:boolean	done
op:hexBinary-equal	($value1 as xs:hexBinary, $value2 as xs:hexBinary) as xs:boolean	not yet

String

fn:concat	($op1 as xs:string?, $op2 as xs:string?, ...) as xs:string	done
fn:contains	($operand1 as xs:string?, $operand2 as xs:string?) as xs:boolean?	done
fn:contains	($operand1 as xs:string?, $operand2 as xs:string?, $collationLiteral as xs:string) as xs:boolean?	Won't
fn:default-collation	() as xs:anyURI?	Won't
fn:ends-with	($operand1 as xs:string?, $operand2 as xs:string?) as xs:boolean?	done
fn:ends-with	($operand1 as xs:string?, $operand2 as xs:string?, $collationLiteral as xs:string) as xs:boolean?	Won't
fn:lower-case	($srcval as xs:string?) as xs:string?	done

fn:matches	($input as xs:string?, $pattern as xs:string) as xs:boolean?	Will
fn:matches	($input as xs:string?, $pattern as xs:string, $flags as xs:string) as xs:boolean?	Will
fn:normalize-space	() as xs:string?	Done
fn:normalize-space	($srcval as xs:string?) as xs:string?	Done
fn:normalize-unicode	($srcval as xs:string?) as xs:string?	won't
fn:normalize-unicode	($srcval as xs:string?, $normalizationForm as xs:string) as xs:string?	won't
fn:starts-with	($operand1 as xs:string?, $operand2 as xs:string?) as xs:boolean?	Done
fn:starts-with	($operand1 as xs:string?, $operand2 as xs:string?, $collationLiteral as xs:string) as xs:boolean?	won't
fn:string	() as xs:string	Done
fn:string	($srcval as item?) as xs:string	Done
fn:string-join	($operand1 as xs:string*, $operand2 as xs:string) as xs:string	Done
fn:string-length	() as xs:integer?	Done
fn:string-length	($srcval as xs:string?) as xs:integer?	Done
fn:string-pad	($padString as xs:string?, $padCount as xs:integer) as xs:string?	Will
fn:replace	($input as xs:string?, $pattern as xs:string, $replacement as xs:string) as xs:string?	Will
fn:replace	($input as xs:string?, $pattern as xs:string, $replacement as xs:string, $flags as xs:string) as xs:string?	Will
fn:substring	($sourceString as xs:string?, $startingLoc as xs:double) as xs:string?	Done
fn:substring	($sourceString as xs:string?, $startingLoc as xs:double, $length as xs:double) as xs:string?	Done
fn:substring-after	($operand1 as xs:string?, $operand2 as xs:string?) as xs:string?	Done
fn:substring-after	($operand1 as xs:string?, $operand2 as xs:string?, $collationLiteral as xs:string) as xs:string?	won't
fn:substring-before	($operand1 as xs:string?, $operand2 as xs:string?) as xs:string?	Done
fn:substring-before	($operand1 as xs:string?, $operand2 as xs:string?, $collationLiteral as xs:string) as xs:string?	won't
fn:tokenize	($input as xs:string?, $pattern as xs:string) as xs:string*	not yet
fn:tokenize	($input as xs:string?, $pattern as xs:string, $flags as xs:string) as xs:string*	not yet
fn:upper-case	($srcval as xs:string?) as xs:string?	Done

Nodes

fn:context-item[1]	() as item?	Will
fn:data	($srcval as item*) as xdt:anyAtomicType*	Done
fn:distinct-nodes	($srcval as node*) as node*	Done

fn:distinct-values	($srcval as xs:anyAtomicType*) as xs:anyAtomicType*	done
fn:distinct-values	($srcval as xs:anyAtomicType*, $collationLiteral as xs:string) as xs:anyAtomicType*	Won't
fn:doc	($uri as xs:string?) as document?	done
op:except	($parameter1 as node*, $parameter2 as node*) as node*	done
fn:id	($srcval as xs:string*) as element*	done
fn:idref	($srcval as xs:string*) as node*	done
fn:input	() as node*	Won't
op:intersect	($parameter1 as node*, $parameter2 as node*) as node*	done
fn:local-name	() as xs:string	done
fn:local-name	($srcval as node?) as xs:string	done
fn:name	() as xs:string	done
fn:name	($srcval as node?) as xs:string	done
op:node-after	($parameter1 as node, $parameter2 as node) as xs:boolean	done
op:node-before	($parameter1 as node, $parameter2 as node) as xs:boolean	done
op:is-same-node	($parameter1 as node, $parameter2 as node) as xs:boolean	done
fn:node-kind[1]	($srcval as node) as xs:string	must
op:NOTATION-equal	($srcval1 as xs:NOTATION, $srcval2 as xs:NOTATION) as xs:boolean	Won't
fn:root	() as node	done
fn:root	($srcval as node) as node	done
fn:sequence-node-identical[1]	($parameter1 as node*, $parameter2 as node*) as xs:boolean?	not yet
op:union	($parameter1 as node*, $parameter2 as node*) as node*	done

Sequence

fn:collection	($srcval as xs:string) as node*	Will
op:concatenate	($seq1 as item*, $seq2 as item*) as item*	Will
fn:empty	($srcval as item*) as xs:boolean	done
fn:exactly-one	($srcval as item*) as item	done
fn:exists	($srcval as item*) as xs:boolean	done
fn:index-of	($seqParam as xs:anyAtomicType*, $srchParam as xs:anyAtomicType) as xs:integer*	Will
fn:index-of	($seqParam as xs:anyAtomicType*, $srchParam as xs:anyAtomicType, $collationLiteral as xs:string) as xs:integer*	Will
fn:insert-before	($target as item*, $position as xs:integer, $inserts as item*) as item*	not yet
fn:item-at	($seqParam as item*, $posParam as integer) as item?	Will
fn:last	() as xs:integer?	done
fn:one-or-more	($srcval as item*) as item+	done
fn:position	() as xs:integer?	done
fn:subsequence	($sourceSeq as item*, $startingLoc as xs:double) as item*	Will

fn:subsequence	($sourceSeq as item*, $startingLoc as xs:double, $length as xs:double) as item*	Will
fn:remove	($target as item*, $position as xs:integer) as item*	not yet
fn:zero-or-one	($srcval as item*) as item?	Done
fn:unordered	($sourceSeq as item*) as item*	Done

QName

fn:get-local-name-from-QName	($srcval as xs:QName?) as xs:string?	not yet
fn:get-namespace-from-QName	($srcval as xs:QName?) as xs:string?	not yet
fn:expanded-QName	($paramURI as xs:string, $paramLocal as xs:string) as xs:QName	not yet
fn:node-name	($srcval as node) as xs:QName?	not yet
op:QName-equal	($srcval1 as xs:QName, $srcval2 as xs:QName) as xs:boolean	not yet
fn:resolve-QName	($qname as xs:string, $element as element) as xs:QName	not yet

URI

op:anyURI-equal	($srcval1 as xs:anyURI, $srcval2 as xs:anyURI) as xs:boolean	not yet
fn:base-uri	($srcval as node) as xs:string?	not yet
fn:base-uri	() as xs:string?	not yet
fn:document-uri	($srcval as node) as xs:string?	not yet
fn:escape-uri	($uri-part as string, $escape-reserved as xs:boolean) as xs:string	not yet
fn:get-namespace-uri-for-prefix	($element as element, $prefix as xs:string) as xs:string?	not yet
fn:resolve-uri	($relative as xs:string) as xs:string	not yet
fn:resolve-uri	($relative as xs:string, $base as anyURI) as xs:string	not yet
fn:namespace-uri	() as xs:string	Done
fn:namespace-uri	($srcval as node?) as xs:string	Done

Lang

fn:lang	($testlang as xs:string) as xs:boolean	will
fn:translate	($srcval as xs:string?, $mapString as xs:string?, $transString as xs:string?) as xs:string?	will

Runtime

fn:error	() as none	not yet
fn:error	($srcval as item?) as none	not yet
fn:trace	($value as item*, $label as xs:string) as item*	not yet
fn:codepoints-to-string	($srcval as xs:integer*) as xs:string	not yet
fn:string-to-codepoints	($srcval as xs:string) as xs:integer*	not yet

Time

op:add-dayTimeDuration-to-date	($srcval1 as xs:date, $srcval2 as xdt:dayTimeDuration) as xs:date	won't
op:add-dayTimeDuration-to-dateTime	($srcval1 as xs:dateTime, $srcval2 as xdt:dayTimeDuration) as xs:dateTime	won't
op:add-dayTimeDuration-to-time	($srcval1 as xs:time, $srcval2 as xdt:dayTimeDuration) as xs:time	won't
op:add-dayTimeDurations	($srcval1 as xdt:dayTimeDuration, $srcval2 as xdt:dayTimeDuration) as xdt:dayTimeDuration	won't
op:add-yearMonthDuration-to-date	($srcval1 as xs:date, $srcval2 as xdt:yearMonthDuration) as xs:date	won't
op:add-yearMonthDuration-to-dateTime	($srcval1 as xs:dateTime, $srcval2 as xdt:yearMonthDuration) as xs:dateTime	won't
op:add-yearMonthDurations	($srcval1 as xdt:yearMonthDuration, $srcval2 as xdt:yearMonthDuration) as xdt:yearMonthDuration	won't
fn:adjust-date-to-timezone	($srcval as xs:date?) as xs:date?	won't
fn:adjust-date-to-timezone	($srcval as xs:date?, $timezone as xdt:dayTimeDuration?) as xs:date?	won't
fn:adjust-dateTime-to-timezone	($srcval as xs:dateTime?) as xs:dateTime?	won't
fn:adjust-dateTime-to-timezone	($srcval as xs:dateTime?, $timezone as xdt:dayTimeDuration?) as xs:dateTime?	won't
fn:adjust-time-to-timezone	($srcval as xs:time?) as xs:dateTime?	won't
fn:current-date	() as date	won't
fn:current-dateTime	() as dateTime	won't
fn:current-time	() as time	won't
fn:adjust-time-to-timezone	($srcval as xs:time?, $timezone as xdt:dayTimeDuration?) as xs:time?	won't
op:date-equal	($operand1 as xs:date, $operand2 as xs:date) as	won't

	xs:boolean	
op:date-greater-than	($operand1 as xs:date, $operand2 as xs:date) as xs:boolean	won't
op:date-less-than	($operand1 as xs:date, $operand2 as xs:date) as xs:boolean	won't
op:dateTime-equal	($operand1 as xs:dateTime, $operand2 as xs:dateTime) as xs:boolean	won't
op:dateTime-greater-than	($operand1 as xs:dateTime, $operand2 as xs:dateTime) as xs:boolean	won't
op:dateTime-less-than	($operand1 as xs:dateTime, $operand2 as xs:dateTime) as xs:boolean	won't
op:dayTimeDuration-equal	($operand1 as xdt:dayTimeDuration, $operand2 as xdt:dayTimeDuration) as xs:boolean	won't
op:dayTimeDuration-greater-than	($operand1 as xdt:dayTimeDuration, $operand2 as xdt:dayTimeDuration) as xs:boolean	won't
op:dayTimeDuration-less-than	($operand1 as xdt:dayTimeDuration, $operand2 as xdt:dayTimeDuration) as xs:boolean	won't
op:divide-dayTimeDuration	($srcval1 as xdt:dayTimeDuration, $srcval2 as xs:decimal) as xdt:dayTimeDuration	won't
op:divide-yearMonthDuration	($srcval1 as xdt:yearMonthDuration, $srcval2 as xs:decimal) as xdt:yearMonthDuration	won't
op:gDay-equal	($operand1 as xs:gDay, $operand2 as xs:gDay) as xs:boolean	won't
fn:get-day-from-date	($srcval as xs:date?) as xs:integer?	won't
fn:get-day-from-dateTime	($srcval as xs:dateTime?) as xs:integer?	won't
fn:get-days-from-dayTimeDuration	($srcval as xdt:dayTimeDuration?) as xs:integer?	won't
fn:get-hours-from-dateTime	($srcval as xs:dateTime?) as xs:integer?	won't
fn:get-hours-from-dayTimeDuration	($srcval as xdt:dayTimeDuration?) as xs:integer?	won't
fn:get-hours-from-time	($srcval as xs:time?) as xs:integer?	won't
fn:get-in-scope-namespaces	($element as element) as xs:string*	won't
fn:get-minutes-from-dateTime	($srcval as xs:dateTime?) as xs:integer?	won't
fn:get-minutes-from-dayTimeDuration	($srcval as xdt:dayTimeDuration?) as xs:integer?	won't
fn:get-minutes-from-time	($srcval as xs:time?) as xs:integer?	won't
fn:get-month-from-date	($srcval as xs:date?) as xs:integer?	won't
fn:get-month-from-dateTime	($srcval as xs:dateTime?) as xs:integer?	won't
fn:get-months-from-yearMonthDuration	($srcval as xdt:yearMonthDuration?) as xs:integer?	won't
fn:get-seconds-from-dateTime	($srcval as xs:dateTime?) as xs:decimal?	won't
fn:get-seconds-from-dayTimeDuration	($srcval as xdt:dayTimeDuration?) as xs:decimal?	won't
fn:get-seconds-from-time	($srcval as xs:time?) as xs:decimal?	won't
fn:get-timezone-from-date	($srcval as xs:date?) as xdt:dayTimeDuration?	won't
fn:get-timezone-from-dateTime	($srcval as xs:dateTime?) as xdt:dayTimeDuration?	won't

fn:get-timezone-from-time	($srcval as xs:time?) as xdt:dayTimeDuration?	won't
fn:get-year-from-date	($srcval as xs:date?) as xs:integer?	won't
fn:get-year-from-dateTime	($srcval as xs:dateTime?) as xs:integer?	won't
fn:get-years-from-yearMonthDuration	($srcval as xdt:yearMonthDuration?) as xs:integer?	won't
op:gMonth-equal	($operand1 as xs:gMonth, $operand2 as xs:gMonth) as xs:boolean	won't
op:gMonthDay-equal	($operand1 as xs:gMonthDay, $operand2 as xs:gMonthDay) as xs:boolean	won't
op:gYear-equal	($operand1 as xs:gYear, $operand2 as xs:gYear) as xs:boolean	won't
op:gYearMonth-equal	($operand1 as xs:gYearMonth, $operand2 as xs:gYearMonth) as xs:boolean	won't
fn:implicit-timezone	() as xs:dayTimeDuration?	won't
op:multiply-dayTimeDuration	($srcval1 as xdt:dayTimeDuration, $srcval2 as xs:decimal) as xdt:dayTimeDuration	won't
op:multiply-yearMonthDuration	($srcval1 as xdt:yearMonthDuration, $srcval2 as xs:decimal) as xdt:yearMonthDuration	won't
op:subtract-dates	($srcval1 as xs:date, $srcval2 as xs:date) as xdt:dayTimeDuration	won't
fn:subtract-dateTimes-yielding-dayTimeDuration	($srcval1 as xs:dateTime, $srcval2 as xs:dateTime) as xdt:dayTimeDuration	won't
fn:subtract-dateTimes-yielding-yearMonthDuration	($srcval1 as xs:dateTime, $srcval2 as xs:dateTime) as xdt:yearMonthDuration	won't
op:subtract-dayTimeDuration-from-date	($srcval1 as xs:date, $srcval2 as xdt:dayTimeDuration) as xs:date	won't
op:subtract-dayTimeDuration-from-dateTime	($srcval1 as xs:dateTime, $srcval2 as xs:dayTimeDuration) as xs:dateTime	won't
op:subtract-dayTimeDuration-from-time	($srcval1 as xs:time, $srcval2 as xs:dayTimeDuration) as xs:time	won't
op:subtract-dayTimeDurations	($srcval1 as xdt:dayTimeDuration, $srcval2 as xdt:dayTimeDuration) as xdt:dayTimeDuration	won't
op:subtract-times	($srcval1 as xs:time, $srcval2 as xs:time) as xdt:dayTimeDuration	won't
op:subtract-yearMonthDuration-from-date	($srcval1 as xs:date, $srcval2 as xdt:yearMonthDuration) as xs:date	won't
op:subtract-yearMonthDuration-from-dateTime	($srcval1 as xs:dateTime, $srcval2 as xdt:yearMonthDuration) as xs:dateTime	won't
op:subtract-yearMonthDurations	($srcval1 as xdt:yearMonthDuration, $srcval2 as xdt:yearMonthDuration) as xdt:yearMonthDuration	won't
op:time-equal	($operand1 as xs:time, $operand2 as xs:time) as xs:boolean	won't
op:time-greater-than	($operand1 as xs:time, $operand2 as xs:time) as xs:boolean	won't
op:time-less-than	($operand1 as xs:time, $operand2 as xs:time) as xs:boolean	won't

op:yearMonthDuration-equal	($operand1 as xdt:yearMonthDuration, $operand2 as xdt:yearMonthDuration) as xs:boolean	won't
op:yearMonthDuration-greater-than	($operand1 as xdt:yearMonthDuration, $operand2 as xdt:yearMonthDuration) as xs:boolean	won't
op:yearMonthDuration-less-than	($operand1 as xdt:yearMonthDuration, $operand2 as xdt:yearMonthDuration) as xs:boolean	won't

Tabella B1: Funzioni