DESAIN DAN IMPLEMENTASI MESIN QUERY UNTUK XML

Thomas Hans[1], Winang Sari Pradani[2], Dhinta Darmantoro[3]

Jurusan Teknik Informatika Sekolah Tinggi Teknologi Telkom
[2]wns@stttelkom.ac.id, [3]dhinta@ stttelkom.ac.id

Abstrak

Salah satu kekuatan XML (*Extensible Markup Languange*) adalah pada fleksibilitasnya dalam menyatakan beragam jenis informasi dari beragam sumber. Untuk mengoptimalkan memanfaatkan kemampuan XML, diperlukan mekanisme yang memungkinkan ekstraksi, seleksi, integrasi, dan transformasi dari informasi yang disimpan dalam bentuk XML. Kemampuan dalam melakukan query pada sumber data XML menjadi semakin penting seiring dengan semakin populernya XML, dimana akan semakin banyak informasi yang akan disimpan, dipertukarkan, dan disediakan dalam bentuk tersebut. XQuery [2] adalah suatu bahasa query untuk XML yang dikembangkan oleh *World Wide Web Consortium* (W3C). Bahasa ini diharapkan dapat menjadi bahasa query standar untuk XML. Penelitian ini bertujuan membuat suatu mesin query bernama XQEngine yang mampu memproses dan mengeksekusi bahasa XQuery. Tidak semua fitur bahasa XQuery akan didukung. Ada beberapa bagian dari bahasa XQuery yang tidak diimplementasikan. Ada pula bagian yang diimplementasikan dengan mengalami penyesuaian. Walaupun XQEngine tidak mendukung semua fitur bahasa XQuery, namun dari uji analisa kasus penggunaan, dapat dilihat bahwa XQEngine mampu menangani jenis-jenis query yang esensial. XQEngine dibangun menggunakan C++Builder® dengan platform Windows™. Versi awal dibangun pada platform Linux dengan bahasa pemrograman Java™.

Kata Kunci : XML, mesin query, *XQuery, XQEngine*

Abstract

Among the advantages of XML is its flexibility to convey information from many sources. To be able to maximally utilize XML, a mechanism enabling extraction, selection, integration, and transformation of information saved in XML is needed. The ability to do querying from XML source(s) is becoming important as XML is becoming more popular as well. This means that there will be more information saved, exchanged, and provided in XML. XQuery[2] is query language for XML data developed by World Wide Web Consortium (W3C). This language is expected to be standard language for XML. Thus, the aim of this research is implementing query engine called XQEngine capable of processing and executing XQuery language. However, not all features of Xquery language would be supported. Some parts of it would not be implemented and some others were implemented after adjustment. Even though XQEngine does not support all Xquery language features, the utilization case analysis shows that XQEngine is capable of handling essential query types. The research result is an engine called XQEngine that has association with subset and minor fitting of XQuery. The engine was built upon C++ Builder® in Windows™ platform. The earlier version was based on Java™ in Linux platform.

Keywords : XML, mesin query, XQuery, XQEngine

1. Pendahuluan

Extensible Markup Languange (XML) adalah suatu format dokumen yang digunakan dalam pertukaran data elektronik [4]. XML merupakan format tekstual, dimana sebuah dokumen XML terbentuk dari rangkaian karakter. Dalam dokumen XML, sebagian karakter membentuk data dokumen dan sebagian lagi membentuk *markup*. *Markup* digunakan untuk menyatakan struktur dari sebuah dokumen XML.

Dengan semakin populernya XML, semakin banyak informasi yang disimpan, dipertukarkan, dan disediakan dalam bentuk tersebut. Untuk bisa memanfaatkan kemampuan XML secara optimal, diperlukan sebuah mekanisme yang memungkinkan ekstraksi, seleksi, integrasi dan transformasi terhadap data yang disimpan dalam bentuk XML.

Salah satu mekanisme yang telah dikenal baik dan digunakan pada bidang basis data adalah bahasa query. Dengan bahasa query, seseorang dapat melakukan manipulasi data dengan mudah sesuai keperluannya. W3C telah merintis pengembangan sebuah bahasa query untuk dokumen XML yang dinamakan XQuery [2]. Bahasa ini diharapkan dapat menjadi bahasa query standar untuk XML.

Pada saat tulisan ini dibuat, pengembangan bahasa XQuery belum mencapai tahap akhir. Masih ada beberapa masalah dan konflik pada spesifikasi bahasa tersebut. Status saat ini dari spesifikasi XQuery adalah *working draft*. Ini berarti spesifikasi bahasa masih belum tetap dan masih ada

kemungkinan perubahan pada masa yang akan datang. Walaupun demikian, perkembangan XQuery pada saat ini telah sampai pada kondisi yang cukup stabil untuk bisa diimplementasikan.

Semakin populernya XML akan menyebabkan semakin banyak informasi yang akan disimpan, dipertukarkan, dan disediakan dalam bentuk XML. Kemampuan untuk melakukan query pada sumber data XML akan menjadi semakin penting.

XQuery adalah suatu bahasa query yang diharapkan dapat menjadi bahasa query standar untuk XML. Untuk dapat memanfaatkan dan menggunakan kemampuan XQuery, pengguna memerlukan suatu mesin query yang mampu memproses dan mengeksekusi bahasa XQuery.

Penelitian ini bertujuan unuk membuat suatu mesin query bernama XQEngine yang mampu memproses dan mengeksekusi bahasa XQuery. Namun, tidak semua fitur dari bahasa XQuery akan didukung. Ada beberapa bagian dari bahasa XQuery yang tidak diimplementasikan. Ada pula bagian yang diimplementasikan dengan mengalami penyesuaian. Selain itu, XQEngine dirancang dan diimplementasikan dengan tidak digunakan fasilitas skema XML, *namespace*, dan fungsi yang didefinisikan oleh pengguna.

2. Bahasa Formal

2.1 Spesifikasi Bahasa

Suatu bahasa pemrograman dapat didefinisikan dengan cara mendeskripsikan bagaimana bahasa itu terlihat (sintaks dari bahasa) dan apa arti dari bahasa tersebut (semantik dari bahasa). Untuk menentukan sintaks dari suatu bahasa, digunakan bentukan yang disebut sebagai *grammar*. Jenis *grammar* yang sering digunakan untuk menyatakan sintaks bahasa pemrograman adalah *context free grammar* [3][1].

Secara formal, derivasi satu langkah dari sebuah *grammar* G didefinisikan sebagai berikut: $\beta A\gamma \Rightarrow_G \beta\alpha\gamma$, dimana: $A \rightarrow \alpha \in P$ dan $\alpha, \beta, \gamma \in (V \cup \Sigma)^*$. Derivasi dengan nol atau lebih langkah dinyatakan dengan $\Rightarrow_G^*$. Bahasa yang dihasilkan oleh sebuah *grammar* G disebut sebagai L(G). Dimana $L(G) = \{ x \in \Sigma^* \mid S \Rightarrow_G^* x \}$.

1. Sebuah context free grammar G, terdiri dari empat bagian (V, Σ, P, S) dimana :
2. V adalah sebuah himpunan terhingga dari variabel (*non-terminal*).
3. Σ adalah sebuah himpunan terhingga dari alfabet (*terminal*).
4. P adalah sebuah himpunan terhingga dari aturan produksi dalam bentuk $A \rightarrow \alpha$ dimana $A \in V$ dan $\alpha \in (V \cup \Sigma)^*$.
5. S adalah simbol awal, dimana $S \in V$.

Parse tree adalah representasi grafik dari sebuah derivasi. Setiap simpul pada *parse tree* diberi label berupa sebuah *terminal* atau *non-terminal*. Simpul akar diberi label simbol awal. Setiap simpul interior diberi label sebuah *non-terminal*. Setiap simpul daun diberi label *terminal* atau ε. Hubungan ayah-anak antara simpul menyatakan sebuah step dari derivasi. Penggabungan dari semua label yang ada di node daun akan membentuk kalimat hasil derivasi. Sebuah *grammar* disebut ambigu, jika terdapat lebih dari satu *parse tree* untuk sebuah derivasi.

Parsing adalah suatu proses pembentukan (pencarian) *parse tree* dari sebuah kalimat berdasarkan sebuah *grammar*. Kebanyakan metoda parsing terbagi dalam dua kelas, yaitu *top-down* dan *bottom-up*. Perbedaan kedua kelas ini adalah pada urutan pembangunan simpul-simpul pada *parse tree*. Pada *top-down parsing*, konstruksi dimulai dari akar diteruskan ke anak-anaknya. Pada *bottom-up parsing*, konstruksi dimulai dari anak-anak lalu diteruskan ke akar.

2.2 Ekspresi Reguler

Ekspresi reguler dibangun dari operasi-operasi yang dilakukan terhadap bahasa, seperti *union*, *concatenation*, dan *kleene-closure*. Jika sebuah bahasa L(*e*) adalah himpunan semua kalimat yang bisa dihasilkan dari ekspresi reguler *e*, maka ekspresi reguler didefinisikan sebagai berikut [1]:

1. å adalah ekspresi reguler yang menyatakan kalimat kosong, atau L(å) = {å}.
2. Setiap anggota *a* dari Ó adalah ekspresi reguler dimana L(*a*) = {*a*}.
3. Jika p dan q adalah ekspresi reguler maka (*p*|*q*) juga adalah ekspresi reguler, dimana L(*p*|*q*) = L(*p*) ∪ L(*q*).
4. Jika *p* dan *q* adalah ekspresi reguler maka (*pq*) adalah ekspresi reguler, dimana L(*pq*) = L(*p*).L(*q*).
5. Jika *p* adalah ekspresi reguler, maka *p** adalah ekspresi reguler yang menyatakan (L(*p*))*.

2.3 Notasi EBNF

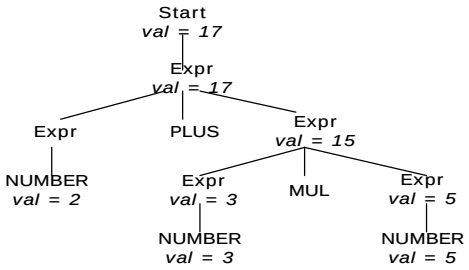
Notasi Extended Backus-Naur Form (EBNF) adalah notasi *context free grammar* yang dikembangkan dengan menambahkan ekspresi reguler pada aturan produksi [1]. Notasi EBNF lebih mudah ditulis dan lebih singkat dibandingkan dengan notasi *context-free grammar* murni.

Operator-operator yang digunakan dalam EBNF adalah * (*kleene closure*), +(perulangan satu atau lebih), ?(pilihan), | (*union*), dan [ ] (*range*).

2.4 Evaluasi Semantik

Untuk membantu dalam evaluasi nilai semantik, ditambahkan atribut pada tiap-tiap node yang ada pada *parse tree*, seperti dicontohkan pada Gambar 1. Tipe atribut bermacam-macam sesuai dengan kebutuhan, bisa berupa *string*, *float*, *integer* dan sebagainya. Atribut dari *terminal* biasanya

diambil dari kalimat leksikal yang bersangkutan. Sedangkan atribut dari *non-terminal* biasanya dihitung dari anak-anak *non-terminal* bersangkutan.



Gambar 1. Evaluasi semantik untuk ekspresi 2+3*5

3. Fitur Bahasa Xquery

3.1 Pengenalan

Bagian ini membahas mengenai fitur-fitur bahasa XQuery yang didukung oleh XQEngine. Pembahasan difokuskan pada fitur-fitur yang mengalami modifikasi. Fitur yang sama dengan spesifikasi XQuery tidak akan dibahas lebih lanjut. Keterangan yang lebih detail mengenai bahasa XQuery dapat dilihat pada spesifikasi yang diterbitkan oleh W3C [2] [7] [6].

Nama	:	*doQuery*
Aktor	:	*Querier*, entity yang memasukkan query
Masukan	:	query dalam bentuk *string*
Keluaran	:	Hasil query berupa dokumen XML atau berupa *fragment* XML
Deskripsi : 1. *Querier* memasukkan query yang hendak dieksekusi ke sistem. 2. Sistem kemudian akan mengeksekusi/ mengevaluasi query tersebut. 3. Jika eksekusi berhasil, sistem akan menampil-kan hasil query beserta waktu eksekusi query. 4. Jika eksekusi gagal, pesan kesalahan.		

3.2 Model Data

Model data menggambarkan informasi apa saja yang terdapat dalam sebuah dokumen XML. Model data juga berfungsi sebagai representasi nilai dari query, baik itu sebagai masukan, nilai antara, dan hasil akhir query. XQEngine mendukung model data query yang diterbitkan oleh W3C [7].

Nilai query adalah nilai dari model data. Semua nilai query, baik masukan, hasil antara, maupun hasil akhir, dinyatakan menggunakan model data. Nilai dari sebuah query bisa berupa sebuah *sequence* atau sebuah nilai *error*.

Sequence merupakan kumpulan terurut (*ordered collection*) dari nol atau lebih *item*. *Sequence* yang mempunyai sebuah elemen disebut sebagai *singleton sequence*. *Sequence* yang tidak mempunyai elemen disebut juga sebagai *empty sequence*. *Item* ada dua macam yaitu berupa *node* atau *atomic value*.

Node menyatakan informasi struktur dan data yang terdapat dalam dokumen XML. XQEngine mendukung enam macam node, yaitu: *document node, element node, attribute node, text node, processing instruction node* dan *comment node.* XQEngine tidak mendukung *namespace node.*

Atomic value digunakan untuk menyatakan nilai dari operasi-operasi tertentu pada query yang memerlukan suatu nilai atomik, misalnya operasi aritmetika, operasi perbandingan dan operasi logika. XQEngine hanya mendukung empat macam *atomic-value*, yaitu *string, integer, decimal,* dan *bool*.

Error adalah sebuah nilai spesial yang mengindikasikan bahwa sebuah kesalahan telah terjadi sewaktu eksekusi query. *Error* semacam ini disebut sebagai *runtime-error*. Ada jenis *error* lain, yang bisa terjadi sewaktu analisa statis query, error ini disebut sebagai *static-error*. Contoh *static-error*: pemanggilan fungsi yang tidak dikenali.

3.3 Ekspresi

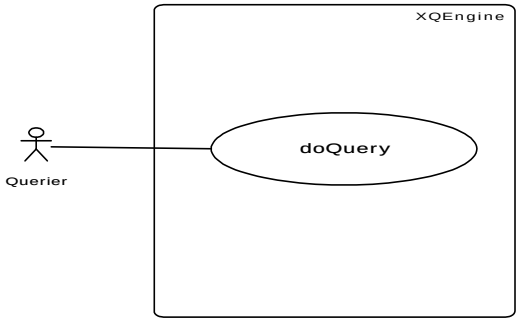
Query pada bahasa XQuery dibentuk dari ekspresi [2]. Ekspresi-ekspresi ini disusun dari *keyword*, symbol dan operator. Ekspresi dapat dikombinasikan dengan berbagai cara untuk membentuk ekspresi yang lebih kompleks, dimana sumber dari suatu ekspresi diambil dari ekspresi lainnya. Bahasa XQuery bersifat *case sensitive*. Semua *keyword* yang ada pada bahasa XQuery terdiri dari huruf kecil.

4. Analisis dan Desain XQEngine

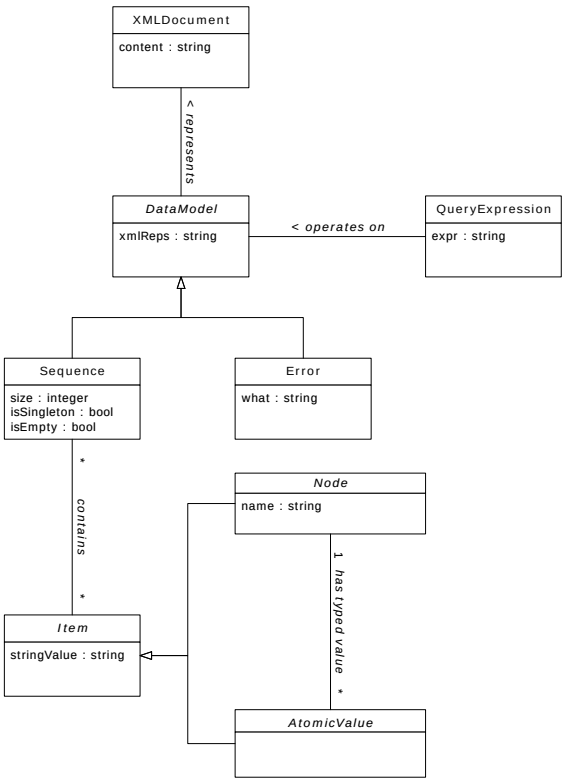
Pendekatan yang dipakai untuk analisa, perancangan, dan implementasi XQEngine adalah pendekatan berorientasi objek. Metodologi yang digunakan adalah metodologi *waterfall* [9]. Notasi diagram yang digunakan adalah notasi Unified Modeling Language (UML) [10].

4.2 *Use Case* Sistem

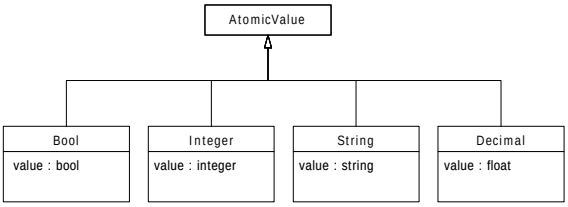
Use case untuk XQEngine didefinisikan sebagaimana pada Gambar 2.



Gambar 2. *Use Case Diagram* untuk XQEngine



Gambar 3. *Class Diagram Data Model*



Gambar 4. *Class Diagram Atomic Value*

4.3 Model Konseptual

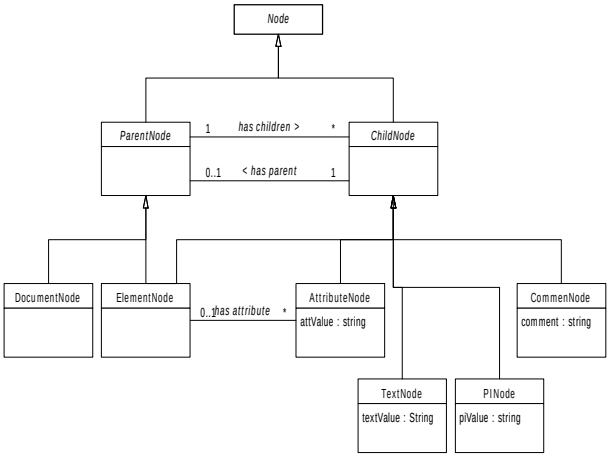
Model konseptual merupakan analisis terhadap konsep-konsep pada domain permasalahan, seperti dijelaskan pada Gambar 3. XQEngine adalah mesin query yang mengeksekusi bahasa XQuery, sehingga domain permasalahan XQEngine berpusat pada bahasa XQuery.

4.3.1 Model Data Query

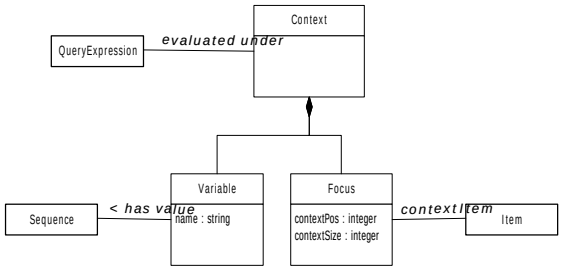
Operasi query pada bahasa XQuery tidak dilakukan langsung pada dokumen XML, namun pada model data query. Model data query merupakan representasi dari dokumen XML. Model data query digunakan sebagai nilai masukan, nilai antara dan nilai keluaran query.

Seperti terlihat pada Gambar 4, *Atomic Value* digunakan untuk menyatakan nilai dari operasi-operasi tertentu pada query yang memerlukan suatu nilai atomik, misalnya operasi aritmetika, operasi perbandingan dan operasi logika. XQEngine mendukung empat macam *AtomicValue*, yaitu *String*, *Integer*, *Decimal*, dan *Bool*.

Pada Gambar 5 ditunjukkan bagaimana *Node* menyatakan informasi struktur dan data yang terdapat dalam dokumen XML. Ada dua jenis *Node* yaitu *ParentNode* dan *ChildNode*. *ParentNode* mempunyai anak berupa *ChildNode*. *ChildNode* mempunyai *parent* berupa *ParentNode*. *Node* yang boleh berfungsi sebagai *parent* adalah *Document-Node* dan *ElementNode*. Sedangkan *node* yang hanya bisa berfungsi sebagai anak adalah *AttributeNode*, *TextNode*, *PINode*, dan *Comment-Node*. *ElementNode* bisa mempunyai anak dan juga bisa berfungsi sebagai anak. *ElementNode* memiliki atribut yang dinyatakan dengan *AttributeNode*.



Gambar 5. *Class Diagram Node*



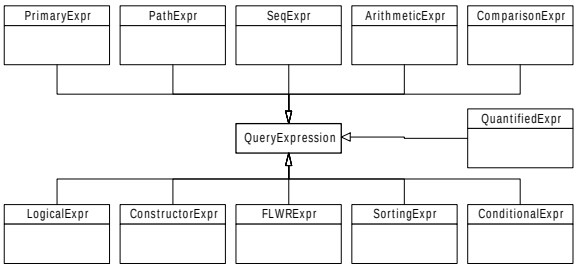
Gambar 6. *Class Diagram Context*

4.3.2 Ekspresi Query

Setiap ekspresi query dieksekusi dalam suatu konteks. Dicontohkan pada Gambar 6, konteks merupakan kumpulan informasi yang tersedia sewaktu eksekusi query. Konteks terdiri dari variabel dan fokus. Setiap variabel mempunyai nama dan berisi nilai yang telah diikat sebelumnya sewaktu evaluasi query. Fokus berisi informasi *item* yang sedang diproses saat itu.

Ada berbagai jenis ekspresi query sebagaimana dijelaskan pada Gambar 7, yaitu: ekspresi primer, ekspresi *path*, ekspresi *sequence*, ekspresi perbandingan, ekspresi logika, ekspresi konstruktor,

ekspresi FLWR, ekspresi *sort*, ekspresi kondisional dan ekspresi terkuantifikasi.



Gambar 7. Class Diagram QueryExpression

4.4 Desain XQEngine

Masukan untuk *XQEngine* adalah sebuah *string* yang berisi query yang hendak dieksekusi. Keluaran dari *XQEngine* adalah hasil query dalam bentuk *string* yang berupa dokumen XML atau *fragment* XML. *XQEngine* juga akan mengeluarkan lama waktu evaluasi untuk sebuah query.

Perbedaan antara sebuah dokumen XML dengan *fragment* XML adalah pada jumlah element teratasnya. Pada sebuah dokumen XML hanya boleh terdapat persis satu buah element yang menjadi element teratas, sedangkan pada fragment XML boleh terdapat nol atau lebih element yang menjadi element teratas. Tidak semua fragment XML adalah dokumen XML.

Evaluasi query terbagi menjadi beberapa tahap, yaitu tahap analisa query, diikuti oleh tahap eksekusi query lalu diikuti oleh tahap serialisasi, seperti dijelaskan pada Gambar 8. Tahap analisa query disebut juga sebagai tahap evaluasi statis, pada tahap ini struktur query akan dianalisa dalam rangka pembuatan rencana eksekusi. Rencana eksekusi ini kemudian akan dijalankan pada tahap eksekusi query. Tahap eksekusi query disebut juga tahap evaluasi dinamis.

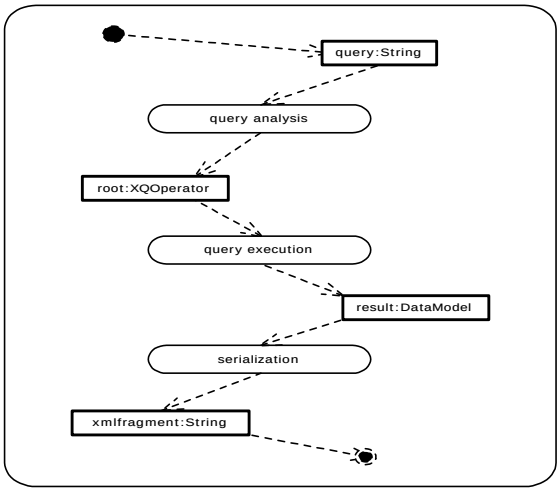
Hasil dari eksekusi query berupa nilai dari model data query. Karena model data query berupa representasi internal memori, maka nilai tersebut perlu diubah ke bentuk eksternal, yaitu ke bentuk dokumen XML atau *fragment* XML. Proses perubahan dari model data ke bentuk *fragment* XML disebut sebagai proses serialisasi.

4.4.1 Tahap Analisa Query

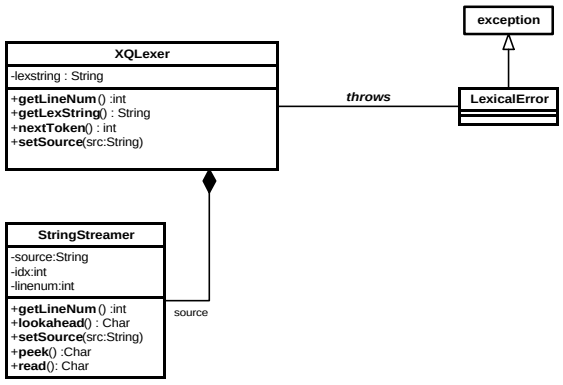
Tahap analisa query dibagi menjadi tiga tahap, yaitu analisa leksikal, analisa sintaks, dan analisa semantik query. Semua query yang valid memenuhi struktur sintaks yang dinyatakan melalui *grammar*.

Pada tahap analisa leksikal dilakukan proses pengenalan *token-token* (atau *non-terminal*) dari rangkaian karakter masukan. *Class* yang bertugas untuk menangani hal ini adalah *XQLexer*, dengan class diagram yang diperlihatkan pada Gambar 9. *XQLexer* menggunakan bantuan *StringStreamer* untuk menyediakan fasilitas *look-ahead* dari kalimat masukan yang akan dianalisa.

Analisa *token* dilakukan oleh *XQLexer* melalui fungsi *nextToken()*. *Token* direpresentasikan sebagai nilai *integer*. Jika ditemukan kesalahan dalam analisa *token*, maka ekspresi bertipe *LexicalError* akan dilempar. Fungsi *nextToken()* dipanggil berulang-ulang sampai semua *token* habis dibaca. Apabila tidak ada lagi *token* yang dapat dibaca, fungsi *nextToken()* mengembalikan nilai token 0. Fungsi *getLexString()* mengembalikan nilai *string* dari *token* yang baru dibaca melalui fungsi *nextToken()*. Fungsi *getLineNumber()* memberikan nomor baris dimana *token* terakhir dibaca.



Gambar 8. Aktivitas Diagram Evaluasi Query

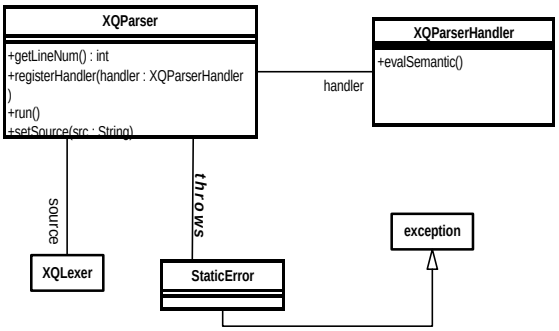


Gambar 9. *Class Diagram XQLexer*

Proses selanjutnya dari analisa query adalah analisa sintaks query. Analisa sintaks akan mencoba menemukan struktur kalimat dari kalimat masukan. Proses ini disebut sebagai proses *parsing*. Masukan dari proses ini adalah rangkaian *token* yang telah dibaca pada tahap analisa leksikal. *Class* yang bertugas untuk menangani analisa sintaks adalah *XQParser*, dengan class diagram pada Gambar 10.

Fungsi *run()* pada *XQParser* akan memulai proses *parsing*, *token-token* yang menjadi masukan diperoleh dari *XQLexer*. Fungsi *run()* akan melemparkan *StaticError* jika terjadi kesalahan

sintaks dimana struktur kalimat tidak memenuhi aturan dari *grammar*.

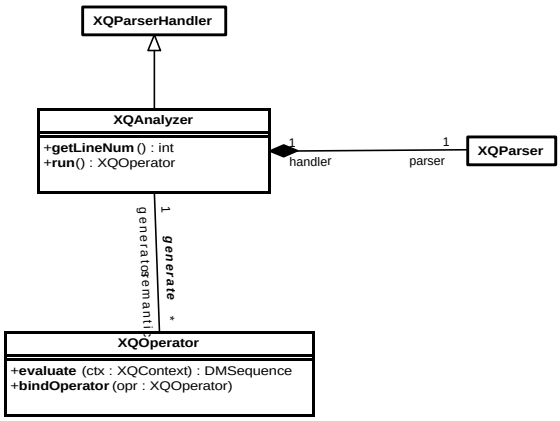


Gambar 10. *Class Diagram XQParser*

Fungsi *registerHandler()* digunakan untuk mendaftarkan sebuah objek bertipe *XQParser-Handler*. Untuk setiap aturan produksi *grammar* yang dikenali oleh *XQParser*, fungsi *evalSemantic()* pada *handler* akan dipanggil untuk menghitung nilai semantik.

Pada tahap analisa semantik dilakukan penyusunan rencana eksekusi query. *Class* yang bertugas untuk ini adalah *XQAnalyzer*, sebagaimana dijelaskan pada Gambar 11. *Class XQAnalyzer* merupakan turunan *XQParserHandler*, oleh karena itu *XQAnalyzer* dapat digunakan sebagai *handler* untuk menghitung nilai semantik untuk *XQParser*. Nilai semantik yang disusun oleh *XQAnalyzer* berupa rencana eksekusi query yang dinyatakan melalui *XQOperator*.

XQOperator menyatakan unit eksekusi terkecil dari suatu query. Sebuah operator akan menghasilkan nilai dan nilai ini dapat digunakan sebagai sumber untuk operator lain. Operator-operator dalam rencana eksekusi query akan saling berkaitan satu sama lainnya membentuk struktur hierarki pohon, dimana masukan dari sebuah operator diambil dari operator yang ada dibawahnya. Operator yang terletak paling atas yang biasa juga disebut sebagai *root-operator*, akan menghasilkan nilai yang merupakan hasil akhir dari query.

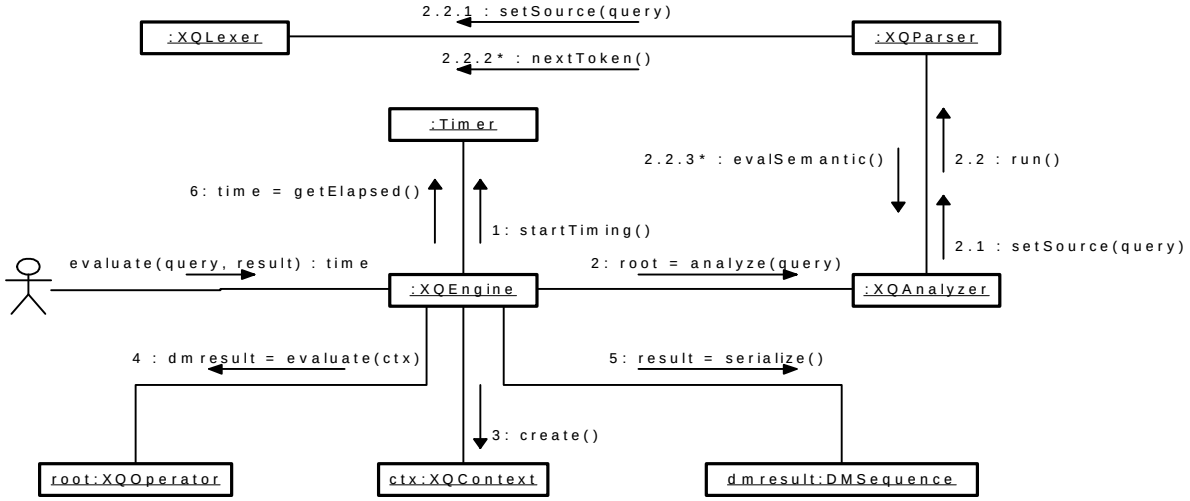


Gambar 11. *Class Diagram XQAnalyzer*

Ada berbagai macam operator yang didukung oleh XQEngine, semua operator merupakan turunan dari class *XQOperator*. Masing-masing operator mempunyai fungsi tersendiri dan memerlukan jumlah *operand* yang berbeda-beda. *Operand* dari suatu operator diikat pada waktu konstruksi objeck operator yang bersangkutan. Namun ada kalanya *operand* operator belum tersedia pada waktu suatu operator diciptakan. Fungsi *bindOperator()* digunakan untuk mengikat *operand* setelah sebuah operator telah diciptakan.

4.4.2 Tahap Eksekusi Query

Pada tahap eksekusi query dilakukan evaluasi terhadap rencana eksekusi query yang telah dibangun pada tahap sebelumnya. Diagram kolaborasi evaluasi query diperlihatkan pada Gambar 12. Eksekusi query dilakukan dengan memanggil fungsi *evaluate()*pada *root-operator*.



Gambar 12. Diagram Kolaborasi Evaluasi Query

Pada waktu eksekusi, operator memerlukan sebuah konteks evaluasi, yang direpresentasikan oleh class *XQContext*. *XQContext* digunakan untuk menyimpan *focus* dan variabel yang diikat selama eksekusi query. Setiap operator mengambil masukan dari operator di bawahnya (dalam pohon operator), kemudian mengolah nilai tersebut untuk dijadikan keluaran operator tersebut. Secara umum eksekusi operator berlangsung dari bawah ke atas, dengan operator pada bagian bawah dieksekusi terlebih dahulu sebelum operator di atasnya.

Operator beroperasi pada model data query yang merepresentasikan nilai masukan, nilai antara dan nilai akhir dari operator. Alur eksekusi dan nilai yang dihasilkan oleh query ditentukan oleh konfigurasi pohon operator yang telah disusun pada tahap analisa query sebelumnya. Representasi nilai query dinyatakan dalam *class DMSequence*. Nilai *Error* dinyatakan dalam class *RuntimeError* yang merupakan jenis class *Exception*.

4.4.3 Tahap Serialisasi Hasil Query

Proses serialisasi dilakukan dengan memanggil fungsi *serialize()* yang terdapat pada class *DMSequence*. Fungsi ini kemudian akan memanggil fungsi *serialize()* pada tiap-tiap objek *DMItem* yang dikandungnya. Fungsi *serialize()* pada *DMItem* akan menghasilkan representasi XML dari item yang bersangkutan. Jika *item* yang dikandung hanya satu dan *item* tersebut adalah *document node* atau *element node*, maka hasil serialisasi dalah sebuah dokumen XML. Jika tidak, maka hasil serialisasi adalah sebuah *fragment* XML.

5. Analisa Uji Kasus Penggunaan

5.1 Kasus Penggunaan Query

Pada Tabel 1 diperlihatkan hasil uji eksekusi query oleh XQEngine pada beberapa kasus penggunaan yang dianggap representatif. Kasus penggunaan diambil dari [8] dan [5]. Kasus-kasus penggunaan yang dipakai mengambil data dari beberapa dokumen XML. Pada contoh kasus penggunaan ini diasumsikan bahwa dokumen XML yang digunakan berada pada sistem file lokal.

Tabel 1 Hasil Evaluasi Kasus Penggunaan

No	Kasus Query	Kesamaan Hasil Query dengan Acuan [5]
1	Seleksi dan Ekstraksi	Ya
2	Perataan Struktur	Ya
3	Proyeksi	Ya
4	Transformasi Struktur	Ya
5	Join	Ya
6	Query pada Posisi	Ya
7	Sorting	Ya
8	Query Nama Elemen	Ya
9	Ekspresi Path Regular	Ya
10	Kuantifikasi	Ya

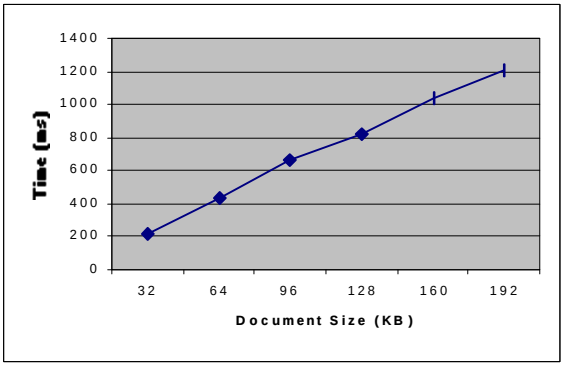
5.2 Kecepatan Evaluasi Query

Pada bagian ini diperlihatkan perbandingan kecepatan evaluasi untuk beberapa query. Untuk tiap-tiap query dilakukan evaluasi dengan ukuran dokumen masukan yang berbeda-beda.

5.2.1 Kasus 1: Seleksi dan Ekstraksi

Untuk kasus pertama, dilakukan pengamatan terhadap eksekusi query yang terdapat pada Tabel 1 nomor 1. Eksekusi query dilakukan sebanyak enam kali dengan ukuran file bib.xml yang berbeda-beda.

Penambahan ukuran dokumen bib.xml dilakukan dengan cara menambahkan elemen-elemen buku tambahan yang tidak memenuhi kondisi seleksi pada query seleksi dan ekstraksi Dengan demikian walaupun ukuran dokumen masukan bertambah, ukuran keluaran tetap sama. Hasil pengujian diperlihatkan pada Gambar 13.



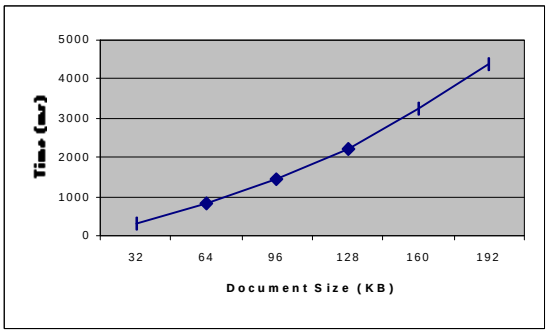
Gambar 13. Kecepatan seleksi dan ekstraksi vs dokumen

5.2.2 Kasus 2: Proyeksi

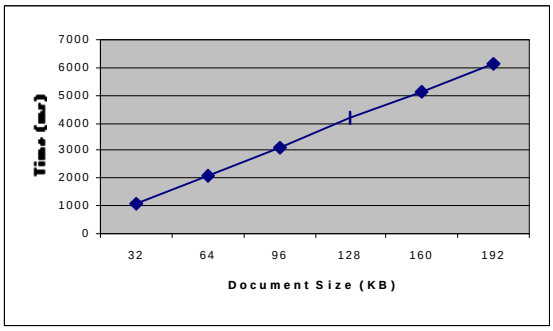
Untuk kasus kedua digunakan query yang terdapat pada Tabel 1 nomor.3. Query pada kasus ini melakukan proyeksi terhadap dokumen masukan. Berbeda pada query kasus pertama, pada query ini tidak ada kondisi seleksi sehingga ukuran keluaran query berbanding lurus dengan ukuran masukan query. Pada Gambar 14 dapat dilihat bahwa kecepatan proyeksi berbanding eksponensial terhadap ukuran dokumen masukan.

Jika dilihat dari jenisnya, query pada kasus kedua ini tidak lebih kompleks dibandingkan dengan query pada kasus pertama, malah lebih sederhana, karena tidak adanya seleksi. Yang membedakan antara kasus pertama dengan kedua adalah pada ukuran keluaran query.

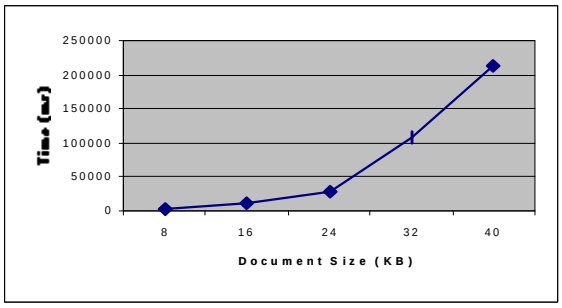
Pada kasus pertama ukuran keluaran query tetap, hal ini disebabkan karena adanya kondisi seleksi sehingga tidak semua elemen yang ada pada dokumen masukan akan menjadi hasil query. Sedangkan pada kasus kedua tidak ada kondisi seleksi sehingga semua elemen yang ada pada dokumen masukan akan ikut menjadi hasil query. Ini berarti ukuran keluaran ikut bertambah seiring dengan bertambahnya ukuran dokumen masukan.



Gambar 14. Kecepatan eksekusi proyeksi vs dokumen



Gambar 15. Kecepatan penggabungan vs dokumen



Gambar 16. Kecepatan evaluasi kuantifikasi vs dokumen

5.2.3 Kasus 3: Penggabungan

Kasus ketiga menggunakan query pada Tabel 1 nomor 5. Pada query ini dilakukan penggabungan data dari dua sumber dokumen yaitu bib.xml dan reviews.xml. Pada kasus ini ukuran dokumen yang divariasikan hanya ukuran dokumen bib.xml, sedangkan ukuran dokumen reviews.xml tetap. Hasil evaluasi query tidak dipengaruhi oleh penambahan ukuran dokumen, seperti terlihat di Gambar 15.

5.2.4 Kasus 4: Kuantifikasi

Untuk kasus ke-empat digunakan query yang ada pada Tabel 1 nomor 10, hasilnya diperlihatkan di Gambar 16. Ada banyak sekali operasi join yang digunakan oleh query ini, mulai dari yang berasal dari ekspresi FLWR hingga yang berasal dari ekspresi *every*.

6. Kesimpulan

Walaupun XQEngine tidak mendukung semua fitur bahasa XQuery, namun XQEngine mampu menangani jenis-jenis query yang esensial. Dalam melakukan query ke dokumen XML, pemrosesan skema bersifat opsional. Walaupun tidak ada dukungan terhadap skema, namun pengguna masih bisa melakukan query yang berguna. Dalam query sederhana, kecepatan eksekusi query berbanding lurus dengan ukuran dokumen masukan. Dalam query yang melibatkan join, eksekusi query bisa berkecepatan eksponensial. Ukuran keluaran query ikut mempengaruhi kecepatan evaluasi query. Artinya proses serialisasi ikut mengambil waktu eksekusi cukup signifikan dalam proses evaluasi query.

Pada implementasi XQEngine ini, rencana eksekusi query yang disusun langsung dieksekusi tanpa melalui optimasi lanjut. Dengan penambahan proses optimasi query diharapkan kecepatan eksekusi query dapat ditingkatkan lagi.

Daftar Pustaka

[1] Aho, A. V., R. Sethi, and J. D. Ullman. 1986. *Compilers Principles, Techniques, and Tools*. New York: Addison-Wesley Publishing Co.

[2] Boag, S., et. al. 2002. *XQuery 1.0: An XML Query Language*. W3C (workingdraft). http://www.w3.org/TR/xquery

[3] Brookshear, J. G. 1989. *Theory of Computation: Formal Languages, Automata, and Complexity*. The Benjamin/Cummings Publishing Co. Inc.

[4] Bray, T., et. al. 2000. *Extensible Markup Language (XML) 1.0 (Second Edition)*. W3C. http://www.w3.org/TR/REC-xml

[5] Chamberlin, D., et. al. 2002. *XML Query Use Cases.* W3C (working draft). http://www.w3.org/TR/xmlquery-use-cases

[6] Draper, D., et. al. 2002. *XQuery 1.0 Formal Semantics.* W3C (working draft). http://www.w3.org/TR/query-semantics

[7] Fernández, M., J. Marsh, and M. Nagy. 2002. *XQuery 1.0 and XPath 2.0 Data Model*. W3C (working draft). http://www.w3.org/TR/query-datamodel

[8] Fernandez, M., J. Simeon, and P. Wadler. 1999. *XML Query Languanges: Experiences and Exemplars* http://www.w3.org/1999/09/ql/docs/xquery.html

[9] Pressman, R. S. 1997. *Software Engineering: A Practitioner's Approach, 4th Ed.* McGraw-Hill Co. Inc.

[10] UML Revision Task Force. 2001. *OMG Unified Modeling Language Specification v. 1.4*. Object Management Group.