ISSN: 0000-0000 (print), 0000-0000 (online)

AIMS Journal

Journal homepage: www.jdconline.net/aims

Regular Paper

NoSQL Database Query Generation using an Automated Approach

Awad Ali[a], Muhammad Ibrahim[b*]

[a] *College of Computer Science & Information Systems, Najran University, Najran, 61441, Saudi Arabia*

A R T I C L E I N F O

Article history:
Received 17 January, 2016
Revised 29 March, 2017
Accepted 14 April, 2017

Keywords:
NoSQL databases
XQuery
XML

A B S T R A C T

It is observed NOSQL that the diversity of data source formats being manipulated on the web with the growth of XQuery as a language to query a widely used kind of data: XML. In addition to this, considering relational databases as a consolidated storage technology, a lack of an integrated mechanism to query different types of data sources offering language independence was noted. Our work presented a solution to the mentioned problem, showing mechanisms to parse and translate an SQL query to XQuery expressions, being able to connect them with an XQuery engine. In the most recent years, we saw the ascent of XQuery, a dialect locally intended to inquiry over XML archives. The Web, as we probably am aware today, makes an enormous utilization of XML reports, To give a coordination between these two universes, our work introduces a device equipped for arrange SQL inquiries into XQuery administrators prepared to be handled by a XQuery motor. Along these lines, we expect to give more flexibility to engineers, offering dialect autonomy by using SAT.

1. Introduction

The XML arrangement got to be broadly spread in the most recent decade and it is these days the standard organization for electronic information exchange. Besides, it is a standard that was destined to drive content presentation on the web, consequently being of most extreme significance in the advanced online applications. In any case, XQuery [1] is a powerful information programming dialect, whose usefulness goes past questioning and manipulating XML information. XQuery gives systems to inquiry these XML-based documents and to build them too.

It is essentially a useful programming dialect, with highlights like unlucky deficiency of reactions, completely compostable outflows, variable tidings, recursion, and higher-request capacities. For questioning, XQuery offers FLOWR expressions, a build influenced by the SQL punctuation, equipped for performing the standard operations—determination, projection, and join. Consequently, we can reason that XQuery really subsumes essential SQL usefulness [2].

The XQuery dialect consolidates the XML information model, in which information is displayed as XML sections. Dissimilar to other information models, XML does not have to comply with a construction, permitting a shifted level of structure, from all around defined plain information to profoundly settled progressions of self-assertive information things. In this manner, this progressive information representation offers both flexibility and extensibility, permitting us to speak to an expansive scope of information organizations.

1.1. Databases and Query Interfaces

In spite of every one of its focal points and broad use on the web, XML is not intensely embraced as capacity innovation, a zone where social databases SQL still rule. Local XML databases (XDBMS) are no doubt understood and inexhaustible as scientific models, however re- fundamental outlandish in big business applications [2]. A conceivable purpose behind this is that the high flexibility of the XML information model includes an extra layer of many-

* *Corresponding author.* Tel.: +0-000-000-0000 ; fax: +0-000-000-0000.
E-mail address: *im.ibrahim@ymail.com*

http://dx.doi.org/........

sided quality to no doubt understood database strategies, for example, streamlining, separation, records, etc. Subsequently, a XDBMS item from significant database sellers was not yet settled available.

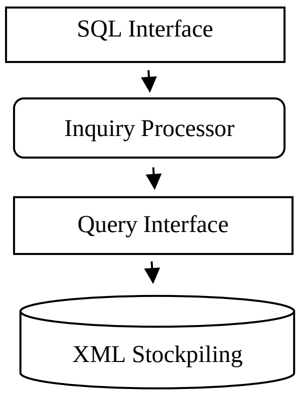

Fig 1. Framework Building Design

Given these angles, we can watch the absence of an instrument that coordinates both XQuery and SQL, permitting grammar autonomy notwithstanding the information stockpiling independence. The transcendence of XML as an information exchange form and the predominance of SQL as questioning dialect, oblige an instrument that permits us to handle both, by concealing expensive transformations done inside. Most endeavors in this course utilize a SQL question motor adjusted to manage XQuery, which includes encode the various leveled model of XML in tables [3].

Since XQuery is a more effective dialect than SQL, it is significant to utilize an XQuery processor as base to make this component. Going for more information oversee capacity, we show a structure equipped for translating SQL utilizing an XQuery motor. The SQL transforming foundation is relocated to XQuery, adding a parser to translate the SQL dialect, giving dialect freedom in the layer "over" the processor. This disposes of the need to revamp code to XQuery. Moreover, it permits the SQL designer to continue working with the framework without expecting to take in another question dialect. The system has the capacity map remote information models to XML, permitting the XQuery motor to process XML information, as well as relational tables.

1.2. Problems in writing XQuery

Big data, big user and high volume processing make the changes in software industries. SQL was dominant on industries but now industries are moving towards NOSQL non-relational databases which are capable to enable huge amount of users, data and processing efficiently. From NOSQL domain XQuery is most flexible and easy to handle every type of databases. Since XQuery is emerging as the standard of XML query languages, considerable effort has been directed towards developing XQuery query processor. The first approach is to implement a native XQuery processor; the other one is to develop an efficient and comprehensive SQL-to- XQuery query processor [2]

- How to parse SQL & XQuery?
- How to extract and integrate SQL and XQuery?
- How to map retrieved Query from SQL to XQuery?
- How transformation will help in database generation?
- What could be the possible results and their effect on Queries?

2. Related work

The significant endeavors in this SQL and XQuery [1-3] reconciliation region have just given information mapping and methods to import (send out) XML information into (from) social databases. The SQL/XML standard [12], for instance, is an augmentation of SQL that permits SQL questions to make XML structures with a couple XML distributed capacities. Some very much defunded social database motors, actualize this standard. Another apparatus that uses a social motor as base to question XML is the SQL Server from Microsoft [7]. It displays an expansion to manage XQuery in its motor, making conceivable to form SQL and XQuery in the same inquiry.

Uniquely in contrast to the beforehand specified advances, some methodologies utilize a local XQuery motor simultaneously. AquaLogic Information Administrations Stage [3] makes an interpretation of SQL inquiries into XQuery expressions to be prepared by an XQuery motor. Not at all like our methodology, has the AquaLogic arrangement needed to reprocess the XQuery question after the interpretation, as opposed to creating XQuery administrators straightforwardly from the SQL inquiry. Another pertinent related work is the IBM task named ROX (Halverson et al. 2004), it concentrates on questioning just over local XML records and look at execution results. On the other hand, the distributed content does not gives much clarification about the real interpretation.

An algorithm is outlined for translating an input STORED query into SQL. The algorithm uses inversion rules to create a single canonical data instance, intuitively corresponding to a schema. The structural component of the STORED query is then evaluated on this instance to obtain a set of results, for each of which a SQL query is generated incorporating the rest of the STORED query. a brief overview of how to translate the basic operations in a path expression query to SQL is provided. The operations described: (1) Returning an element with its children, (2) selections on values, (3) pattern matching, (4) optional predicates, (5) Predicates on attribute names and (6) regular path queries which can be translated into recursive SQL queries [2].

The binary association method deals with translating OQL-like queries into SQL. The class of queries they consider roughly corresponds to branching path expression queries in XQuery. In XRel, a core part of XPath called XPathCore is identified and a detailed algorithm for translating such queries into SQL is provided. Since with each element, a path id corresponding to the root-to-leaf path is stored, a simple path expression query like book/section/title gets efficiently evaluated. Instead of performing a join for each step of the path expression, all elements with a matching path id are extracted. Similar optimizations are proposed for branching path expression queries exploiting both path ids and the interval encoding.

Algorithms for translating order based path expression queries into SQL are provided. They provide translation procedures for each axis in XPath, as well as for positional predicates. Given a path expression, the algorithm translates one axis at a time in sequence. The dynamic intervals approach deals with a larger fragment of Xquery with arbitrarily nested FLWR expressions, element constructors and built-in functions including structural comparisons. The core idea is to begin with static intervals for each element and construct dynamic intervals for XML elements constructed in the query. Several new operators are proposed to efficiently implement the generated SQL queries inside the relational engine [7-8]. These operators are highly specialized and are similar to operators present in a native XML engine.

3. Used Approach

At this chapter the implementation details of the proposed methodology have also been described. Our work presented a solution to the mentioned problem, showing mechanisms to parse and translate an SQL query to XQuery expressions, being able to connect them with an XQuery engine. In the most recent years, we saw the ascent of XQuery, a dialect locally intended to inquiry over XML archives. The Web, as we probably am aware today, makes an enormous utilization of XML reports, To give a coordination between these two universes, our work introduces a device equipped for arrange SQL inquiries into XQuery administrators prepared to be handled by a XQuery motor.

3.1. XQuery Syntax Presentations

The following diagram indicates the basic structure of a query statement.

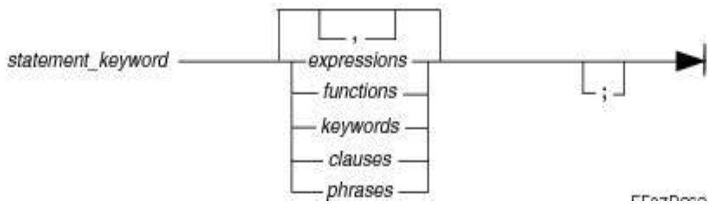

Fig 2. Syntax representation

This syntax element Specifies …

statement_keyword	the name of the statement.
expressions	literals, name references, or operations using names and literals.
functions	the name of a function and its arguments, if any.
keywords	special values introducing clauses or phrases or representing special objects, such as NULL..
clauses	subordinate statement qualifiers.
phrasesdata	attribute phrases.

3.2. SQL Query Parsing

The SQL Query Parser is a generated parser. LPG (Laxer Parser Generator) is used to generate the parser, which, based on a set of grammar rules, generates the Parser and Laxer source code in Java. The parser takes SQL DML statements as input and creates instances of SQL Query Model classes if the SQL is syntactically valid. In addition to the syntactic validation, the parser can also perform a semantic validation. The parser is extensible to support vendor specific dialects and custom source generation. The first step in our SQL query processing is to parse the query. Parsing is the process of analyze a text, consisting in a sequence of tokens, to determine its grammatical structure according to a given formal grammar [4-5]. Our Parser respects the SQL grammar, and produces a tree of tokens, conforming this grammar structure. It receives a string, the main rule of the parser is the statement rule, which consists in the select_statement rule, followed by a semicolon. The select_statement rule is composed by the sequence of sub-rules listed below:

- select_clause: Accepts prefixed column references, asterisk (all the columns from all specified tables), prefixed asterisks (all the columns from certain table), and expressions, which can be column references, function calls and etc.
- from_clause: Accepts table references and alias assignment. Besides, allows sub- queries, which are recursively defined as statement rules, but interpreted as table references by the queries above it.
- where_clause (optional): Accepts any predicate.
- groupby_clause (optional): Accepts column references that can be followed by an optional having_clause.
- orderby_clause (optional): Accepts column references, each one can be followed by the ASC or DESC optional modifiers.

3.3. SQL Lexicon

Here some SELECT, INSERT, UPDATE and DELETE core queries are parsed.

```
"SELECT" column_name, * "FROM" table_name "ORDER"  "BY" column_name  "ASC" | "DESC"  "," *
     "WHERE"  some_column compEXP some_value ";"
"INSERT"  "INTO"  table_name (column1,*)"VALUES" (value,*)  ";"
"DELETE"  "FROM" table_name "WHERE" some_column  compEXP some_value  ";"
"UPDATE"  table_name "SET"  (column  "="  value,*) "WHERE" some_column  compEXP some_value  ";"
```

3.4. XML Data Model

XQuery overcomes many of the limitations of XPath at the expense of introducing complexity: its rich semantics significantly increase the complexity of query evaluation and optimization. XQuery provides a feature called FLWOR expressions. The FLWOR acronym stands for *for*, *let*, *where*, *order by* and *return*. All FLWOR expressions start with a *for* or a let expression and end with a return expression. FLOWR expressions enable XQuery to: — perform iterations (for) — define variables (let) — order results (order by) — impose constraints and perform joins (where) and finally — construct custom formatted data (return).

3.5. Path Expression Queries

We refer to the class of path expression queries without predicates as simple path expression queries. For interval-based solutions, evaluating simple path expressions entails performing a range join for each step of the path expression. For example the query book/author/name translates into a three-way join. For id-based solutions, each parent-child (/) step translates into an equijoin, whereas recursion in the path expression (through //) requires a recursive SQL query. For path-based solutions, the path id can be used to avoid performing one join per step of the path expression.

3.6. Lexicon of Xquery Queries

Here some Xquery 3.0 update facility queries are parsed according to their syntax w3c standard. In analysis and extract reserved words and parameters and literal etc. After this lexical analysis, the token stream transform into Xquery.

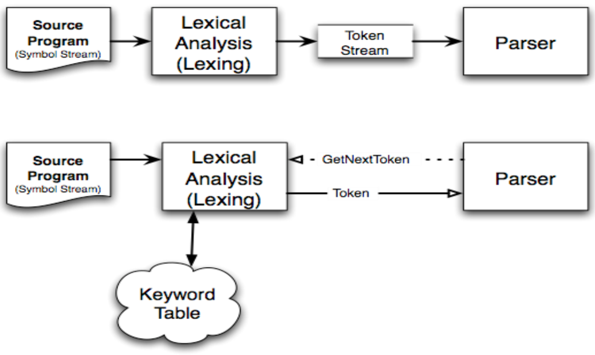
Source Program (Symbol Stream)
Lexical Analysis (Lexing)
Token Stream
Parser
Source Program (Symbol Stream)
Lexical Analysis (Lexing)
GetNextToken
Token
Parser
Keyword Table

Fig 3. Framework Building Design

```
•  ExprSingle         ::=         FLWORExpr
   | QuantifiedExpr
   | InsertExpr
   | DeleteExpr
   | RenameExpr
   | TransformExpr
   | OrExpr
•  FLWORExpr          ::=         (ForClause | LetClause)+ WhereClause? OrderByClause? "return"
   ExprSingle
•  ForClause          ::=         "for" "$" VarName TypeDeclaration? PositionalVar? "in" ExprSingle
   ("," "$" VarName TypeDeclaration? PositionalVar? "in" ExprSingle)*
•  LetClause          ::=         "let" "$" VarName TypeDeclaration? ":=" ExprSingle ("," "$"
   VarName TypeDeclaration? ":=" ExprSingle)*
•  WhereClause        ::=         "where" ExprSingle
•  OrderByClause      ::=         (("order" "by") | ("stable" "order" "by")) OrderSpecList
•  OrderModifier      ::=         ("ascending" | "descending")?
•  InsertExpr         ::=         "insert" ("node" | "nodes") SourceExpr InsertExprTargetChoice
   TargetExpr
•  DeleteExpr         ::=         "delete" ("node" | "nodes") TargetExpr
•  ReplaceExpr        ::=         "replace" ("value" "of")? "node" TargetExpr "with" ExprSingle
•  OrExpr             ::=         AndExpr ( "or" AndExpr )*
•  AndExpr    ::=         ComparisonExpr ( "and" ComparisonExpr )*
•  ComparisonExpr
•  GeneralComp        ::=         "=" | "!=" | "<" | "<=" | ">" | ">="
•  ValueComp          ::=         "eq" | "ne" | "lt" | "le" | "gt" | "ge"
•  NodeComp           ::=         "is" | "<<" | ">>"
•  PathExpr           ::=         ("/" RelativePathExpr?)
   | ("//" RelativePathExpr)
   | RelativePathExpr
•  RelativePathExpr::=   StepExpr (("/" | "//") StepExpr)*
•  Wildcard           ::=         "*"
•  Predicate          ::=         "[" Expr "]"
•  Literal         ::=    NumericLiteral | StringLiteral
•  VarRef          ::=    "$" VarName
•  sourceExpr       ::=   ExprSingle
•  TargetExpr       ::=   ExprSingle
•  NewNameExpr      ::=   ExprSingle
•  IntegerLiteral   ::=   Digits
```

3.7. SQL To XQuery Mapping Matrix

Here a model present to mapping SQL query to XQuery by which we can extract data. In the matrix we place the queries and mark the matching lexicon according to grammar rules of XQuery and SQL as well as matches the reserve words and their syntax. This approach helps to make the decision which XQuery syntax available against SQL query.

Input SQL

```
SELECT column_name, column_name
FROM  table_name
ORDER BY column_name ASC|DESC,
WHERE  column_name = Literal ;
```

Output XQuery

```
for  $VarName  in  PathExpr
where  ComparisonExpr and|or  ComparisonExpr
order by ascending|descending Literal
return  ExprSingle
```

Selection Matrix

Table 2: A matric for select query to XQuery mapping

	For	$	varName	In	PathExpr	Where	CompExpr	Order by	ascending	Descending	literal	return	Expsingle
SELECT		✓	✓		✓							✓	
Column_name			✓										
FROM	✓	✓	✓	✓	✓								
Table_name			✓										✓
ORDER BY								✓					
ASC									✓				
DESC										✓			
WHERE						✓							
CompExpr							✓						
Literal											✓		
;													

Following are the mapping details for the insert query of SQL to XQuery.

Input SQL
```
INSERT INTO table_name
Column_name,...)VALUES (literal,...);
```

Output Xquery
```
insert node|nodes SourceExpr into TargetExpr
```

Insertion Matrix

Table 3: A selection matric for insert query to XQuery mapping

	Insert	node	nodes	sourceExp	Into	TargetExp
INSERT	✓	✓	✓			
INTO					✓	
table_name						✓
column_name						✓
VALUES				✓		
Literal				✓		
;						

Following are the mapping details for the delete query of SQL to XQuery.

Input SQL
```
DELETE FROM table_name
WHERE some_column=some_value;
```

Output Xquery
```
delete  node|nodes  TargetExpr
literal ComparisonExpr literal]
```

Table 3.2: A selection matric for delete query to XQuery mapping

Input SQL
```
UPDATE  table_name
SET column1=value1,column2=value2,...
WHERE some_column=some_value;
```

Output Xquery

```
replace value of node TargetExpr [literal ComparisonExpr literal]with ExprSingle
```

Table 4: A selection matric for update query to XQuery mapping

	Replace	**Value of node**	**TargetExpr**	**[]**	**Literal**	***ComparisonExpr***	**with**	**ExprSingle**
UPDATE	✓	✓						
table_name			✓					
SET							✓	
some_colunm			✓					
CompExpr				✓		✓		
some_value					✓			✓
WHERE				✓				
;				✓				

4. Results and Discussion

The compilation time have been also measured, to calculate the time taken to process the translation of the SQL query to XQuery operators. Results show that all queries produced the exact same results, confirming correctness of all the tested constructions. Since the performance was not the focus of this study, the tests were ran in small tables, resulting in a significant time slice being considered by the compiler. However, when it comes to bigger storage files, the compilation time is still the same, while the total execution time of each query grows, decreasing the impact of the compilation time. Regardless of table size and time taken to execute some query, the translation process remains in the same time magnitude, being not noticeable to the user eyes, thus not significantly affecting the engine performance. It is possible to carry out some code optimizations, improving the query compilation time.

Table 5: A selection matric for update query to XQuery mapping

	delete	**Node**	**nodes**	**TargetExpr**	**[ ]**	**Literal**	***ComparisonExpr***
DELETE	✓	✓	✓				
FROM	✓						
table_name				✓			
WHERE					✓		
some_column				✓			
ComparisonExp							✓
some_value						✓	
;						✓	

5. Conclusion and Future Work

In our solution, an SQL parser has been developed. We introduce a very flexible tool, which supports storage independence in the layer "below" the query processor and language independence in the layer "above". In this way, we aimed to abstract as much as possible, in both ends, the process of querying data from different data sources. To validate the developed solution, some queries were run, aiming to check its correctness, which was confirmed with the obtained results. Considering the unconcern with the performance, the proposed objective of this work has been achieved. Dividing the process into three different steps reduced the work complexity and increased the organization. However, the translation step is still with some complicated manipulation and workarounds, which can be better depicted as future work, enhancing the code legibility and extensibility. Other related works are not as flexible as ours, but some of them offer more SQL constructions, in the regard of SQL coverage. In this way, it is interesting, as future work, to implement missing SQL constructions, such as count, outer join, union and others that are beyond the basic clauses, often used in the queries. An important topic to be improved as future work is the experiments. More solid tests can be executed, and consolidated benchmarks, such as TPCH can be used to verify the SQL standard coverage, correctness and performance of the developed SQL interface.

REFERENCES

[1] W3C. XQuery 1.0: An XML Query Language (Second Edition). 2010. Avail- able from Internet: <http://www.w3.org/TR/xquery/>.
[2] QUERY, W. X.; GROUPS, X. W. XML Path Language (XPath) 2.0 Standard. [S.l.]: Network Theory Ltd., 2010. ISBN 190696601X, 9781906966010.

[3] Jigyasu, S. et al. SQL to XQuery translation in the aqualogic data services platform. In: Proceedings of the 22nd International Conference on Data Engineer- ing. Washington, DC, USA: IEEE Computer Society, 2006. (ICDE '06), p. 97–. ISBN 0-7695-2570-9.

[4] Anders, B. (2006). Extensible Stylesheet Language (XSL) Version 1.1. W3C Recommendation 05 December 2006 http://www.w3.org/TR/xsl11/.

[5] Codd, E. F. Relational completeness of data base sublanguages. In: R. Rustin (ed.): Database Systems: 65-98, Prentice Hall and IBM Research Report RJ 987, San Jose, California, 1972.

[6] Eisenberg, A.; MELTON, J. Advancements in sql/xml. SIGMOD Record, v. 33, n. 3, p. 79–86, 2004.

[7] Garcia, R. Foundations Book II: Understanding SQL Server 2005 Sup- porting Technology (XML, XSLT, XQuery, XPath, MS Schemas, DTD's, Namespaces). [S.l.]: Lulu.com, 2007. ISBN 1430324465, 9781430324461.

[8] Halverson, A. et al. Rox: relational over xml. In: Proceedings of the Thirtieth international conference on Very large data bases - Volume 30. VLDB Endowment, 2004. (VLDB '04), p. 264–275. ISBN 0-12-088469-0. Available from In- ternet: <http://dl.acm.org/citation.cfm?id=1316689.1316714>.

[9] HSQLDB. 2012. Available from Internet: <http://hsqldb.org/>.

[10] Bajwa, I. S., Mumtaz, S., & Naveed, M. S. (2008). Database Interfacing using Natural Language Processing. European Journal of Scientific Research, 20(4), 844-851.

[11] Katz, H. et al. XQuery from the Experts: A Guide to the W3C XML Query Language. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003. ISBN 0321180607.

[12] W3C. XML Schema Part 2: Datatypes (Second Edition). 2004. Available from Internet: <http://www.w3.org/TR/xmlschema-2/>.

[13] WOOD, L. Programming the web: The w3c dom specification. IEEE Internet Computing, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 3, n. 1, p. 48–54, jan. 1999. ISSN 1089-7801. Available from Internet: <http://dx.doi.org/10.1109/4236.747321>.

[14] Naeem, M. A., Ullah, S., & Bajwa, I. S. (2012). Interacting with data warehouse by using a natural language interface. In Natural Language Processing and Information Systems (pp. 372-377). Springer Berlin Heidelberg.