Generating OLAP Queries from Natural Language Specification

M. Asif Naeem
School of Computing and Mathematical Sciences,
Auckland University of Technology, Auckland
New Zealand
+64(0)9 921 9999
mnae006@aucklanduni.ac.nz

Imran Sarwar Bajwa
Department of Computer Science &IT
The Islamia University of Bahawalpur
Pakistan
+92(0)62 925 5466
i.s.bajwa@cs.bham.ac.uk

ABSTRACT
Automatic translation of natural language (NL) questions to Structured Query Language (SQL) queries is a challenging task. It is a common knowledge that writing Online Analytical Processing (OLAP) queries for data warehouses is difficult, particularly, for the novice users. In this paper, we present a natural language processing based approach to automatically generate OLAP queries those can be used to communicate with the a data warehouse. In the presented approach, user provides queries in English and our approach process English queries and generate OLAP queries. In our approach, we incorporate OMG’s recent standard Semantic of Business Vocabulary and Business Rules (SBVR) to simplify the translation process of English to OLAP. SBVR is used in detailed semantic analysis of English queries. The presented approach is also implemented in Java as a prototype tool. To test the performance of the tool, an experimental study is also conducted. Results of the experimental study imply that our approach is capable in communicating with a data warehouse.

Categories and Subject Descriptors
D.3.3 [Programming Languages]: Language Contracts and Features – abstract data types, polymorphism, control structures.

General Terms
Algorithms, Management, Performance.

Keywords
Keywords are your own designated keywords.

1. INTRODUCTION
Automatic generation of semantics of a programming language from a natural language is challenging task in the field of computational linguistics for last couple of decades. Such automated approaches can have serious impact in business and social world. As, access of accurate data at the right time to the right place in the right format has become need of the modern business organizations. Access to correct data is significant in achieving business success as customers are typically interested in the latest information about products in order to purchase, pay and ship [1].

OLAP queries are typically involved in communication with the data warehouses. To communicate with a data warehouse, one need to first build a connection stream and then accesses the data contents from the data warehouse using a standardized interfacing mechanism [2]. Simple command shells are typically used and they are often incorporated within every distinct data warehouse product. These command shells are typically simple filters which helps a user to log on to the data warehouse, execute particular commands and receive output. These command shells provide access to the data warehouse from the machine on which the RDBMS (Relational Database Management System) is actually running [3]. After hooking to a particular data warehouse a user or a programmer requires an interface and typically that interface is provided by some technical languages. These languages are called query languages and are constituted of the SQL commands typically used for retrieving information from a data warehouse. OLAP is set of particular commands under SQL [4] which are specifically used to interact with data warehouse repositories.

It is a common knowledge that an application programmer has to write OLAP queries to communicate warehouse. OLAP queries are inherited from a declarative query language, SQL, but writing OLAP queries are difficult to write due to their inherent complexities. The questions are typically specified in SQL for large web-based and stand-alone applications. However, writing OLAP based queries for a data warehouse is a complex task specifically for the users having little or no knowledge of SQL queries. Moreover, it is quite difficult to remember the SQL commands and use them accurately. An ad-hoc solution was presented in [5] to provide a Q&A interface for data warehouse. However, there are a few short-comings in the presented approach. Firstly, the architecture does not provide the taxonomy used to perform the semantic analysis for the given user’s specification of OLAP queries. Secondly, the given approach only considers one type of OLAP queries named aggregation while the other two types drill-down and slicing-dicing are also equally important for query generation. Thirdly, the authors evaluated their approach based on the “number of entities found” only and they did not provide any data about the accuracy and the execution time taken to generate one query. An automated
approach to assist writing of OLAP queries can be a great help of data warehouse administrators and the novice users of a data warehouse.

In this paper, we exploit mapping at syntactic and semantic level from natural language to OLAP queries. Here, SBVR plays key role in shallow and deep semantic analysis of a natural language query. The presented approach is implemented to a prototype tool QueGen that can facilitate both users and software engineers. The tool is capable of generating typical OLAP queries from English specification of queries. The architecture of the tool is discussed in this paper. We also designed a dataset of question and queries and use them to test the working of the tool.

Rest of the paper is structured as: Section 2 describes the problem addressed in this paper. In Section 3 we present the approach used to generate OLAP queries from English text. An experimental study is presented in Section 4 to evaluate the performance of the tool. The related work to our approach is described in Section 5 and finally Section 6 concludes the paper.

2. PROBLEM DESCRIPTION
The typical OLAP queries applied on a data warehouse are multi-dimensional commands. Due to this characteristic users need extensive level technical skills to execute these commands. The situation becomes more critical when a low skilled person wants to access or analyze his business data from data warehouse. This requires more expertise and skills in terms of understanding and writing the accurate and functional queries. To illustrate it further we consider a multi-dimensional query in English for tables shown in Figure 1:

“I want the sum of units sold from fact sales for Coca-Cola in each country from 01-Jan-11 to 31-Dec-11.”

The OLAP query for the above example is given below.

```
SELECT country, SUM(Units_Sold)
FROM Fact_Sales
WHERE Product_Name="Coca_Cola"
AND date BETWEEN 01-Jan-11 AND 31-Dec-11
GROUP BY Country;
```

![Figure 1. A sketch of star schema for “Soft Drink Company’s” data warehouse](image)

If we observe the above query, a number of technical concepts like group function SUM, WHERE clause, a logical operator AND, a comparison operator BETWEEN and GROUP BY clause are involved. Therefore, it is difficult to write such type of queries for novel or those users who do not familiar with the above technical concepts.

These complex tasks can be simplified by providing an easy interface that can be well known to all kind of users. In order to resolve all such issues, automated software is needed, which facilitates both users and software engineers. For complex queries an expert programmer also requires assistance in terms of automatic query generation. He can use these queries after making appropriate adjustments and alterations in the automated generated queries with less effort in less time as compared to the traditional approaches.

3. FROM NL TO OLAP QUERIES
To address the problem described in Section 2, we present a novel approach that can not only process NL queries but also maps to OLAP queries. In the presented approach, OMG’s standard SBVR (Semantic of Business Vocabulary and Rules) [6] based logical representation is generated from English specification queries and then the SBVR based logical representation is mapped to SQL syntax. QueGen generates the intended OLAP queries that can be directly executed on a data warehouse. The proposed approach has robust ability to create code automatically without external environment. It provides a quick and reliable way to generate OLAP queries to save the time and budget of both the user and organizations.

![Figure 2. An Abstract Level Architecture of the Tool](image)

The presented approach works as a user provides the English specification of a query language and our approach lexically and syntactically analyzes (see Figure 2) the English specification of query using the Stanford POS tagger [12] and the Stanford parser [13]. We have used the off-shelf components for lexical and syntactic analysis as these tools provide high accuracy such as the Stanford POS tagger is 97.0% accurate [12], while the Stanford parser is 84.1% accurate [13]. However, for the semantic analysis of the English queries we have written a semantic analyzer that resolves all such issues, automated software is needed, which facilitates both users and software engineers. For complex queries an expert programmer also requires assistance in terms of automatic query generation. He can use these queries after making appropriate adjustments and alterations in the automated generated queries with less effort in less time as compared to the traditional approaches.

The presented approach works as a user provides the English specification of a query language and our approach lexically and syntactically analyzes (see Figure 2) the English specification of query using the Stanford POS tagger [12] and the Stanford parser [13]. We have used the off-shelf components for lexical and syntactic analysis as these tools provide high accuracy such as the Stanford POS tagger is 97.0% accurate [12], while the Stanford parser is 84.1% accurate [13]. However, for the semantic analysis of the English queries we have written a semantic analyzer that generates a logical representation based on SBVR vocabulary. Once the logical representation is extracted from the input text, the SBVR based logical representation is mapped to OLAP terms. In such SBVR to OLAP mapping, the appropriate OLAP terms"
such as such as keyword, table names, field names, field value, etc
are assigned to SBVR vocabulary items on the basis of identified
semantics. Finally, using these OLAP terms query generation
module generates the OLAP query which can be applied on a data
warehouse directly.

The English language statements are effortlessly converted into
OLAP queries by using QueGen tool. Select query is the common
query in OLAP that is used to choose a set of values from a data
warehouse table [7].

To evaluate the working and performance of QueGen, we
implemented a light weight data warehouse for a “Soft Drinks
Company” as an example. The “Soft Drinks Company” produced
variety of drinks like Coca-Cola, Sprite, Pepsi etc, and supplies
these products worldwide. The sketch of start schema we used to
implement the data warehouse for the company is shown in Figure
2. According to the figure there are three dimension tables namely
Dim_Date, Dim_Product, and Dim_Store which are connected
with a central fact table, Fact_Sales under one-to-many
relationship. Normally the three dimension tables contain master
and reference data while the fact table contains transactional data
that changes frequently [1].

In the incoming sections, we will show step-by-step that how our
approach generates the OLAP query from the example discussed
in Section 2.

3.1 Processing NL Queries
The first phase in NL to OLAP queries is detailed parsing of NL
text. The NL parsing phase is divided into lexical, syntactic and
semantic processing.

3.1.1 Lexical Processing
In lexical processing of English specification of an OLAP query,
following three steps are involved:

a. Tokenization. Identification of the token is the first step in
lexical processing. Example discussed in Section 2 is tokenized in
Figure 3:

[I] [want] [the] [sum] [of] [units] [sold] [from] [fact] [sales] [for] [Coca-Cola] [in] [each] [country] [from] [01-Jan-11] [to] [31-Dec-11] [.]

Figure 3. Tokenizing English specification of query

b. Parts-of-Speech (POS) Tagging. The tokenized text is further
passed to Stanford parts-of- speech (POS) tagger v3.0 to identify
the basic POS tags (see Figure 4). We have used the Stanford
POS tagger v3.0 to identify tokens and POS tags from NL text
due to its high accuracy that is 97% [12].

[I/PRP] [want/VBP] [the/DT] [sum/NNS] [of/IN] [units/NNS] [sold/VBN] [from/IN] [fact/NN] [sales/NNS] [for/IN] [Coca-Cola/NNP] [in/IN] [each/DT] [country/NNS] [from/IN] [01-Jan-11/JJ] [to/TO] [31-Dec-11/JJ] [./.]

Figure 4. POS tagging of text using the Stanford POS Tagger

c. Morphological Analysis. After POS tagging, the input text is
morphologically processed to separate the suffixes possibly
attached to the nouns and verbs e.g. a noun “units” is analyzed as
“unit+s” and “sales” is analyzed as “sale+s”.

3.1.2 Syntactic Processing
We have used the Stanford parser to parse the lexically analyzed
text and generate parse tree representation and syntactic
dependencies. An example of the syntactic dependencies is shown
in Figure 5.

nsubj(want-2, I-1)
root(ROOT-0, want-2)
det(sum-4, the-3)
dobj(want-2, sum-4)
prep_of(sum-4, units-6)
partmod(units-6, sold-7)
nn(sales-10, fact-9)
prep_from(sold-7, sales-10)

prep_for(sales-10, Coca-Cola-12)
det(country-15, each-14)
prep_in(sold-7, country-15)
prep_from(sold-7, 01-Jan-11-17)
prep_to(01-Jan-11-17, 31-Dec-11-19)

Figure 5. Parsing English text using the Stanford Parser

3.1.3 Semantic Analysis
In this semantic analysis phase, semantic role labeling [12] is
performed. As we aim to generate a SBVR based logical
representation, we have used the SBVR vocabulary types as the
possible semantic roles. We have chosen SBVR for semantic role
labeling because SBVR is an adopted standard and easy to map to
other formal languages as SBVR is based on higher order logic.
Commonly used SBVR based semantic role labels are Object
Type, Individual Concept, Verb phrase, Characteristics, fact type,
etc [15-16]. Mapping of English text to SBVR based semantic
roles is explained in the remaining part of this Section.

a. Object Types. All common nouns are represented as object
types.

b. Individual Concepts. All proper nouns are mapped to the
individual concepts.

c. Verb Concepts. The action verbs are represented as verb
concepts.

d. Characteristics. The attributes are adjectives and possessive
nouns (i.e. ending in’s or coming after of).

e. Quantifications. Determiners such as ‘a’ and ‘an’ are mapped to
uniqueness quantification (∃=1X). Tokens ‘each’, ‘all’, and
‘every’ are mapped to universal quantification (∀X). Similarly,
‘several’, ‘lot’, ‘much’, ‘more’, ‘some’, etc are mapped to
existential quantification (∃X). If the keywords like ‘more than’
or ‘greater than’ are used with n then solution quantifier (§X) is
mapped to At-most n Quantification. Similarly, if the keywords
like ‘less than’ or ‘smaller than’ are used with n then solution
quantifier is mapped to At-least n Quantification.
On the basis of the above defined mapping, the semantic roles for the example discussed in Section 2 are shown in Figure 6.

\[
\begin{align*}
[I] \{\text{want/Verb}\_\text{Concept}\} \{\text{the/DT}\} \\
\{\text{sum/Characteristic}\} \{\text{of/IN}\} \\
\{\text{units/Object}\_\text{Type}\} \{\text{sold/Verb}\_\text{Concept}\} \\
\{\text{from/IN}\} \{\text{fact sales/Object}\_\text{Type}\} \{\text{for/IN}\} \\
\{\text{Coca-Cola/Individual}\_\text{Concept}\} \{\text{in/IN}\} \\
\{\text{each/Universal}_\text{Quantification}\} \{\text{country/Object}\_\text{Type}\} \{\text{from/IN}\} \{\text{01-Jan-11/Characteristic}\} \{\text{to}\} \{\text{31-Dec-11/Characteristic}\} \\
\end{align*}
\]

Figure 6. Semantic interpretation of English text

Once the semantic role labeling is performed, the NL text is mapped to a logical representation as shown in the Figure 7. We have written a rule based analyzer that maps semantic roles to a SBVR based logical representation.

\[
\begin{align*}
\{\text{sold}\} \\
\{\text{characteristic} = (\text{sum} ? Z)\} \{\text{object}\_\text{type} = (\forall Y \sim (\text{units} ? Y))\} \{\text{individual}\_\text{concept} = (\text{Coca-Cola} ? W)\} \{\text{object}\_\text{type} = (\forall V \sim (\text{country} ? V))\} \{\text{characteristic} = (\text{01-Jan-11 ? U})\} \{\text{characteristic} = (\text{31-Dec-11 ? T})\} \\
\end{align*}
\]

Figure 7. SBVR based Logical representation.

First we need to create a BSP tree. A BSP-tree of a UML class model contains all the information regarding total number of classes in UML class model and their hierarchal structure. Instead of creating the BSP tree from a graphical scene, we create the BSP tree from XMI document. The first class in an XMI document is considered as a root. Rest of the classes in XMI document becomes child nodes of the root in the tree. All classes \(C_r, C_s, ..., C_n\) associated to a class \(C_a\) become children of the class \(C_a\). Moreover, the associations, generalizations and aggregations are used to identify the hierarchy of the nodes in the tree.

3.2 Extracting OLAP Query Elements

In this phase, finally the SBVR rule is further processed to extract the OO information. The extraction of each OO element from SBVR representation is described below:

3.2.1 Extracting Tables

Each Object Type or Individual Concept is mapped to the table names in the target data warehouse and if matches with a table name then that Object Type or Individual Concept is tagged as ‘Table Name’.

3.2.2 Extracting Fields

An Object Type or an Individual Concept that does not match to any table name, it is mapped to the field names of each table in the target data warehouse and if matches with any field name then that Object Type or Individual Concept is tagged as ‘Field Name’.

3.2.3 Extracting Functions

An Object Type, Individual Concept or a characteristic that does not match to any table name or field name is looked in a list of functions names and if matches with any function name then that Object Type or Individual Concept is tagged as ‘Function Name’.

3.2.4 Extracting Field Values

A Characteristic that is not a function is mapped to field value. Moreover, any Individual Concept that does not match to a table name or field name are considered as field name.

3.2.5 Extracting Keywords

The tokens in English text such as ‘show’, ‘list’, ‘select’, and ‘display’ are mapped to “select” keyword.

3.3 OLAP Query Generation

This is the final phase in generation of OLAP query from English specification of queries. In this phase the logical representation generated in semantic analysis phase and the keywords extracted in Section 3.2 are combined to generate a particular query. Finally, the SQL (OLAP) query is generated by embedding the extracted information in Section 3.2 in the following template:

\[
\text{SELECT} <\text{field-name}>, \quad [\text{function-name}(<\text{field-name}>)] \quad \text{FROM} <\text{field-name}> \quad \text{WHERE} <\text{field-name}> = <\text{field-value}> \quad \text{[AND} \quad <\text{field-name}> \quad \text{and ...] [GROUP BY} <\text{field-name}>];
\]

Figure 8. A Templates used for OLAP Query Generation

A snapshot of required OLAP query is shown in Figure 9. Moreover, the query shown in the Figure is same to that one which is already described in Section 2.

4. EXPERIMENTAL STUDY

We performed an extensive experimental evaluation of our proposed QueGen. In this section we illustrate the environment of our experiments and analyze the results that we obtained through various queries generation.
4.1 Experimental Setup
In order to implement QueGen and the prototype of data warehouse for a "Soft Drinks Company" we used the following hardware and data specifications.

4.1.1 Hardware Specifications
We carried out our experimentation on a Pentium-IV 2X2.13GHz machine with 3G main and 160G disk memory under WindowsXP. We implemented the experiment in Java using the Eclipse IDE 3.3.1.1. We also used built-in plugin provided by the Java API, to measure the processing time. We used MySQL 5.0 in order to build a light weight data warehouse.

4.1.2 Data Specification
We tested the performance and accuracy of QueGen using simple to complex level queries. We generated multi-dimensions queries to test QueGen. The inputs we provided to QueGen were in the form of natural English language.

4.2 Results Analysis
In this section we carried out the experimental study to evaluate the performance of QueGen. We evaluated the performance of QueGen with respect to accuracy and execution time. In both kinds of evaluations we applied three major types of OLAP queries namely consolidation, drill-down, and slicing and dicing. Before further proceeding it is important to present a brief overview of each type first. Consolidation is basically an aggregation of data that can be gathered or computed in single or multi dimensions. For example, to observe the overall sales trend the sales of all products by all countries can be accumulated. Contrarily, the drill-down is an approach which permits users to go through the details. For example, users can observe the sales of individual product in each country. Slicing and dicing is a technique where users can access a specific set of data of the cube (slicing) and the slices can be observed from the different angles (dicing).

4.2.1 Accuracy Evaluation
In this experiment we analyzed the accuracy rate of QueGen by applying all three categories of OLAP queries. Under each category we run QueGen for both simple and complex level queries and compared the generated queries with the original queries. On the basis of that we measured the accuracy level as shown in Table1.

<table>
<thead>
<tr>
<th>Query Types</th>
<th>Simple</th>
<th>Complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consolidation</td>
<td>92.45%</td>
<td>84.53%</td>
</tr>
<tr>
<td>Drill-down</td>
<td>87.58%</td>
<td>81.93%</td>
</tr>
<tr>
<td>Slicing &amp; Dicing</td>
<td>85.26%</td>
<td>80.73%</td>
</tr>
</tbody>
</table>

The accuracy rate for both simple and queries is shown in Figure 10. From the figure it can be observed that under all three OLAP categories the accuracy level for both simple and complex queries is significantly high. Moreover, in case of consolidation the accuracy level is even greater than the other two categories. The reason behind this is that because the queries under consolidation category are comparatively less complex than the other two categories. For more clarity we also present the concrete values for our measurements, as shown in Table1.

4.2.2 Execution Time
In our second experiment we tested the performance of QueGen in terms of execution time. We run both simple and complex queries under all three OLAP categories and measure the time taken by each query. To make our measurements more accurate, for each query we repeated our experiment for ten times and then take the average as a final result. The results of the experiment are shown in Figure 11.

<table>
<thead>
<tr>
<th>Query Types</th>
<th>Simple</th>
<th>Complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consolidation</td>
<td>4.82</td>
<td>6.75</td>
</tr>
<tr>
<td>Drill-down</td>
<td>7.53</td>
<td>9.13</td>
</tr>
<tr>
<td>Slicing &amp; Dicing</td>
<td>11.23</td>
<td>13.75</td>
</tr>
</tbody>
</table>

Figure 11 shows that for most complex type of queries QueGen does not take more than fifteen seconds. However, on the other hand for simple consolidate queries it takes even less than five seconds.
5. RELATED WORK
Various approaches have been presented to translate natural language queries to SQL queries to extract data from relational database [14]. A recent attempt [5] is made in the similar direction by proposing a natural language based interface to communicate with data warehouse. The approach consists of three main components, the Named Entity Recognizer (NER), the Question Translation, the Question Rewriting, and the Query Completion. The NER identifies the known objects of the data model into dimensions, measures and values of dimensions. The Question Translation component build a technical query from user’s provided scenario while Question Rewriting component rewrites the query if it is incorrect. The component Query Completion completes the query if user’s scenario is incomplete. Although this approach is a positive step in this area of research, there are some limitations which need to explore further. Firstly, the architecture does not provide the taxonomy used to perform the semantic analysis for the given user’s scenario. Secondly, the given approach only considers one type of OLAP queries named aggregation while the other two types drill-down and slicing-dicing are also equally important for query generation. Thirdly, the authors evaluated their approach based on the “number of entities found” only and they did not provide any data about the accuracy and the execution time taken to generate one query.

6. CONCLUSION
The task of writing OLAP queries is difficult and error prone for non-technical users or even medium level skilled persons. This task can be simplified by providing an easy interface that is more familiar and well known to the users. In this paper we proposed a novel automated query generator tool with name QueGen. The proposed tool has the capability to generate all three categories of OLAP queries. User needs to write the requirements in simple English language. The proposed tool analyzes the given script. After semantic analysis and mapping of associated information, it generates the intended OLAP queries that can be applied directly on data warehouse. The proposed tool provides a quick and reliable way to generate OLAP queries to save the time and cost of users and business organizations. We also evaluated the performance of the proposed tool with respect to both execution time and accuracy.

Currently, QueGen does not have the ability to generate complex queries which are normally used in the area of data mining. In future we have a plan to extend the feature of QueGen so that it can generate these kinds of queries.

7. References