IMPROVEMENT OF THE BILL OF MATERIALS (BOM) GENERATOR FOR PRODUCT VARIANTS

Sri Raharno[1]and Yatna Yuwana Martawirya[2]

Mechanical Engineering Department, Faculty of Mechanical Engineering and Aerospace Engineering, Institut Teknologi Bandung, Bandung, Indonesia, Tel: 62-22-2504243, Fax: 62-22-2534099,
e-mail: [1]harnos@staff.itb.ac.id; [2]yatna@ftmd.itb.ac.id

Received Date: December 25, 2012

Abstract

In short, a bill of materials (BOM) is a list of parts or components and quantities, which are required to manufacture a product. A BOM also describes the component structure of a product, usually as a hierarchical structure implemented within a relational database. Generally, to generate a BOM of a product that has no variant is a relatively simple process. On the other hand, there are problems in generating a BOM for a product with many variants. Since the number of variants may be large, it is impossible to design and maintain a BOM structure for each variant. The high number of components will certainly result in a time consuming BOM generation. Moreover, another challenge of data management associated with variety of products is data redundancy. In order to overcome the problems, previous research has developed a product data model using a single structure for many product variants. The research also has implemented a heuristic rule as a BOM generator. However, the implementation has shown that generating a BOM has been time consuming and required relatively complex codes. This research deals with an improvement of the BOM generator developed in previous research. The improvement involves reducing the duration of processes and simplifying the codes.

Keywords: Bill of Materials (BOM) generator, Improvement, Product variants

Introduction

The bill of material (BOM), which is a documentation technique on product structure, is used to demonstrate the structure and relations between the final product, subassemblies, as well as the corresponding quantities of the subordinate parts and materials of each assembly [1][2]. A structure model is proposed to record the product tree. Each object in the tree presents itself as a parent item or a child item. There are different forms of BOM during the product life cycle. For instance, the production stage involves the Engineering BOM (EBOM), the Process BOM (PBOM) and the Manufacturing BOM (MBOM). EBOM is one form of BOM that is widely used in material requirement planning and manufacture resource planning. EBOM is also the foundation of other BOM forms of a product. PBOM is used in the stage of processing of parts, which reflects the product assembly structure and sequences. MBOM includes all material items that are necessary in the manufacture of the product [3]. In this case, one of the challenges of data management associated with different forms of BOM is avoidance of the BOM databases redundancy [4].

In theory, the varieties derived from a product could be in hundreds of thousands. For instance, a car of common type could be assembled in millions of variants through all possible combinations of its assemblies. However, practically, the diversification of model into variants is limited to those assemblies and final products with few differences [1]. In a customer-oriented environment, generic products replace standardized models. A generic product is defined through a set of attributes, which may have a set of alternatives parts/variants. Since the number of variants may be large, it is difficult to design and maintain a BOM structure for each variant. A solution is to describe all product variants in one generic BOM [5][6][7]. The BOM for each product variant may then be generated from this structure by specifying attributes [8].

A BOM describes the component structure of a product, usually as a hierarchical structure implemented within a relational database. These descriptions include the relations between the end-product, subassemblies, and materials. The conventional approach for the implementation of these structures in an Enterprise Resource Planning (ERP) or a Product Data Management (PDM) system is to design a single BOM for each product variant. However, this becomes impossible in a customer-oriented production, where the generic product is defined through a set of attributes, which may have alternative values or variants [9].

Previous research has resulted in a product data model using a single structure for many product variants [10]. The product data model has been implemented using a relational database management system (RDBMS) and a BOM is generated by employing some queries based on a heuristic rule. There are many advantages of developing a bill of material generator based on a query language processor [11]. For instance, the advantages are (1) least amount of time required to developed and implement the bill of material generator, (2) database administration and maintenance are made simple by the ability to easily manipulate the stored data using query language commands and (3) the bill of materials generator could be expanded readily by adding a new object.

Previous problem solving approach is shown in Figure 1. The implementation of the heuristic rule as BOM generator primarily used a nested procedure to retrieve the product structure and to collect the material data. In this case, the RDBMS was only used to read and write the product structure data. This approach has two shortcomings, namely the process duration and the complexity of codes. Related to the process duration, to generate a BOM for product variants by retrieving data repeatedly using a nested procedure from the RDBMS and filtering data outside of the RDBMS certainly is more time consuming. Furthermore, retrieving product structure data and collecting material data require complex codes.

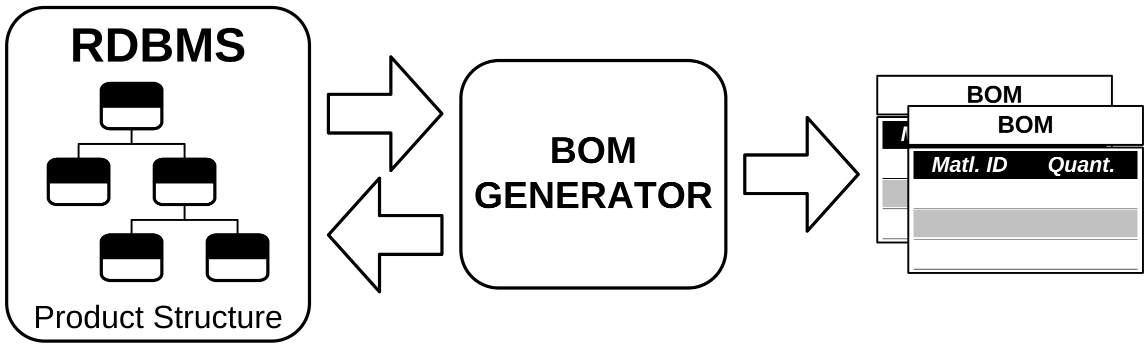


Figure 1. Previous problem solving approach [10]

ProposeD Problem Solving Approach

Based on the previous shortcomings, an improvement method is proposed. As shown in Figure 2, the proposed problem solving approach implemented the BOM generator as part of the RDBMS using object views (virtual object tables). In this case, the RDBMS used to not only read and write the product data but also to collect and filter material data from the product structure through object views.

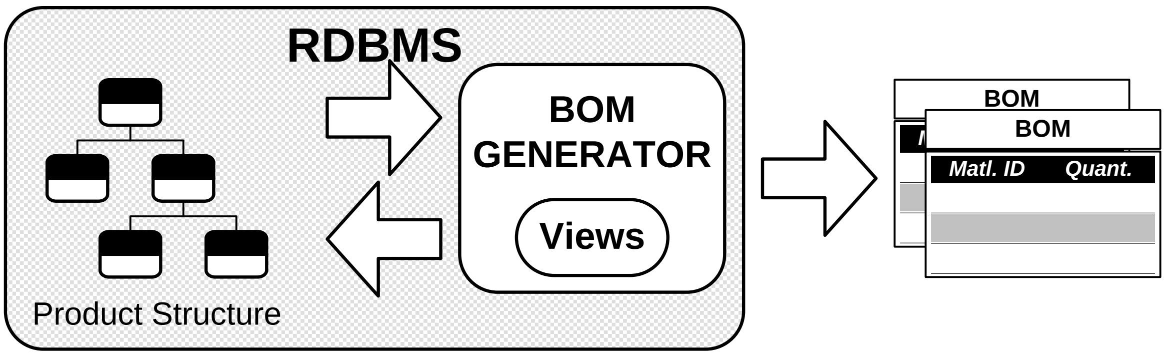


Figure 2. Proposed problem solving approach

There are several reasons to develop a BOM generator through creating object views of RDBMS. First, RDBMS organizes data in tables and relations between tables. The relationships that could be created among the tables enable a RDBMS to store huge amount of data efficiently and retrieve selected data effectively. Next, RDBMS has been used in hundreds of thousands application worldwide and it has been proven adequate for the job. Moreover, a RDBMS could process up to a few thousand transactions per second, thus a RDBMS is an ideal system for transaction processing and handling of complex query work loading. Lastly, a RDBMS has a native language called Structured Query Language (SQL) developed to work with it. Using the native language could reduce the processing time and result in easier communication with the BOM generator. Briefly, this approach has possibilities to reduce processing time and obtaining a BOM by using simpler codes.

Modelling of Product Structure

Before discussing the product structure and its modeling, first let’s look into the definitions of product models and product variants that are used in this research. A product model is defined as a group of products with certain identification and name. A product variant is a product model that has more detail specifications. A product model may have several product variants. For example, Airbus A-380 is a product model, while Airbus A380-8XX-000 with 569 tons MTOW, Airbus A380-8XX-001 with 510 tons MTOW and Airbus A380-8XX-002 with 569 tons MTOW are product variants.

The relation between a product model and its variants is shown in Figure 3. It may be seen that Model 01 has three product variants. They are variant VA, VB, and VC. For these variants, there is only one product structure model. In general, model of product structure is a part of product data model that provides information about the breakdown of parts that construct the product and the relations among final products, assemblies, subassemblies, and parts or components. As shown in Figure 3, product model M01 has 9 elements for its structure. The attributes of an element are shown at the left bottom side of the Figure 3. Furthermore, each element in the structure has an ownership mark. For example, variant VA, VB, and VC own the element with ID 0.0 and name Product X. It means that variants VA, VB, and VC have this element as part of their product structures.

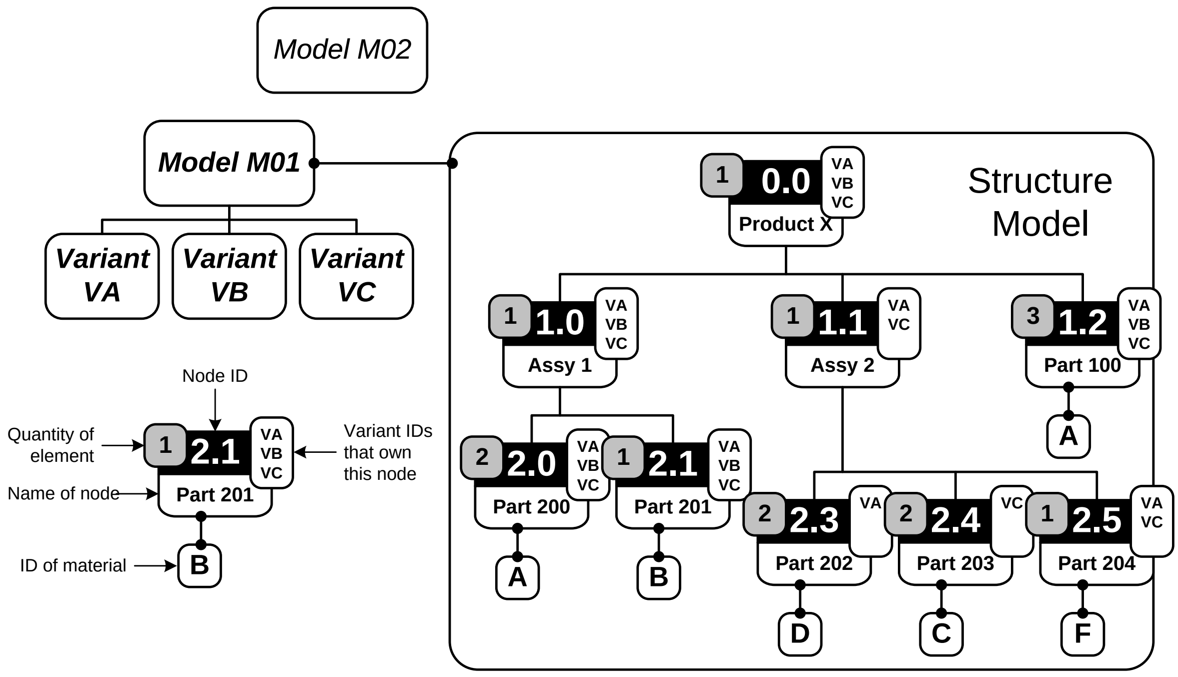
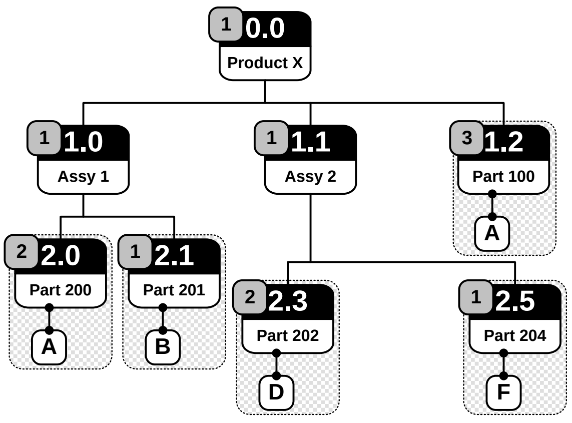


Figure 3. Relations between product models and product variants

As shown in Figure 3, the elements of product structure may have information about the material. For example, element 2.0 is made of material A, and its quantity is two. According to the structure, if material A from element 2.0 will be assembled with material B from element 2.1 then they will build an upper element namely element 1.0. In this model, not all of element has information about their materials. In this case, only elements that is at the bottom of the structure need to have information about the material.

Although the product data model only has one structure model for many variants, but by using ownership marks, it could generate a product structure from each variant. The product structure for variant VA, VB, and VC are shown in Figure 4 to Figure 6 respectively. Furthermore, the structure of each variant will be used to generate the BOM by collecting information about the materials and their quantities. As shown in Figure 4, the BOM of variant VA consists of material A from element 2.0 and 1.2 with the quantity of five, material B from element 2.1 with the quantity of one, material D from element 2.3 with the quantity of two and one material F from element 2.5.



BOM for VA

Matl. ID	Quant.
A	5
B	1
D	2
F	1

Figure 4. The product structure for variant VA and its BOM

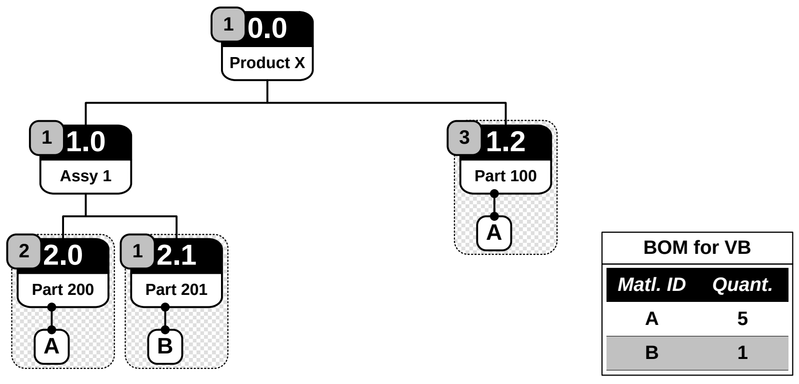


Figure 5. The product structure for variant VB and its BOM

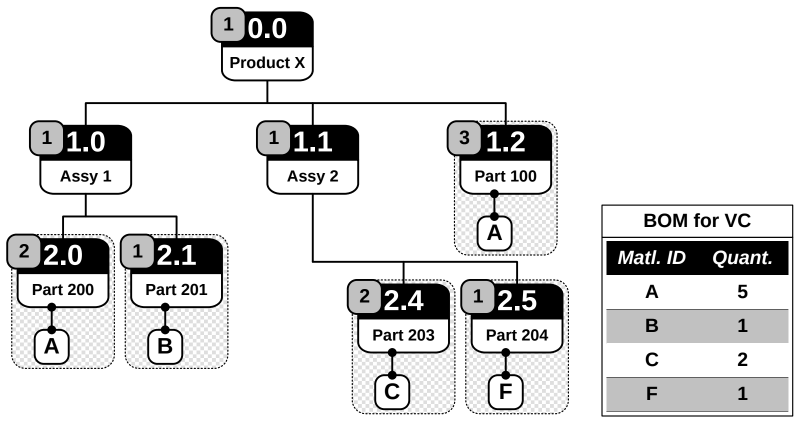


Figure 6. The product structure for variant VC and its BOM

Development of the BOM Generator

In this research, development of the BOM generator employed Oracle XE RDBMS. This is a RDBMS offered by Oracle, free to distribute on Windows and Linux platforms. This RDBMS is restricted for use as single CPU with a maximum of 4 GB of user data and 1 GB maximum memory, although it could be installed on a server with any amount of memory. The simplified physical data model as implementation of the product data model used in this research is shown in Figure 7.

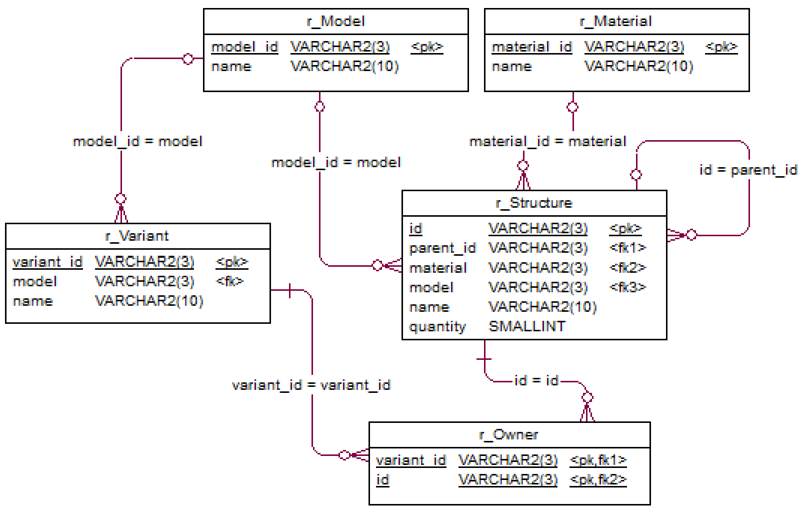
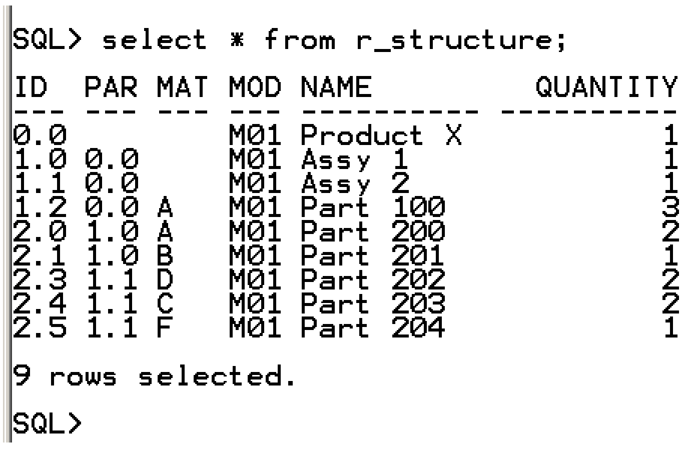


Figure 7. The simplified physical data model for the product data

Table 1 shows the required data for models, variants, and materials related with the physical data model above. The structure of product data used in this research is shown in Figure 8 and the relations between the elements of product structure and product variants are shown in Figure 9.

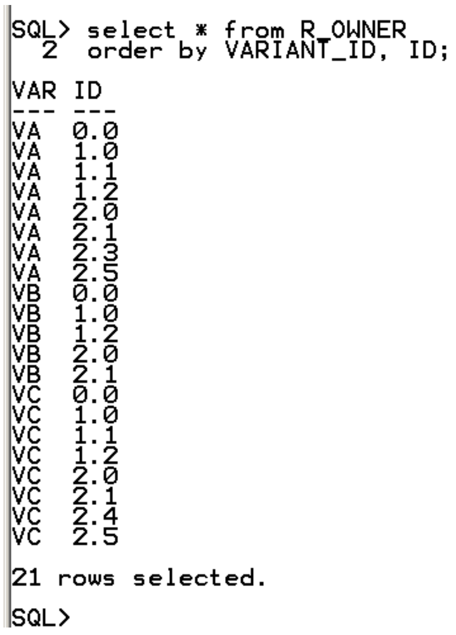
Table 1. Models, Variants, and Materials Data

	ID	Name		ID	Name
Model	M01	Model 01		A	Material A
	M02	Model 02		B	Material B
Variant	**ID**	**Name**	**Material**	C	Material C
	VA	Variant A		D	Material D
	VB	Variant B		E	Material E
	VC	Variant C		F	Material F



```
SQL> select * from r_structure;

ID  PAR MAT MOD NAME           QUANTITY
--- --- --- --- ---------- ----------
0.0         M01 Product X           1
1.0 0.0     M01 Assy 1              1
1.1 0.0     M01 Assy 2              1
1.2 0.0 A   M01 Part 100            3
2.0 1.0 A   M01 Part 200            2
2.1 1.0 B   M01 Part 201            1
2.3 1.1 D   M01 Part 202            2
2.4 1.1 C   M01 Part 203            2
2.5 1.1 F   M01 Part 204            1

9 rows selected.

SQL>
```

Figure 8. Product structure data

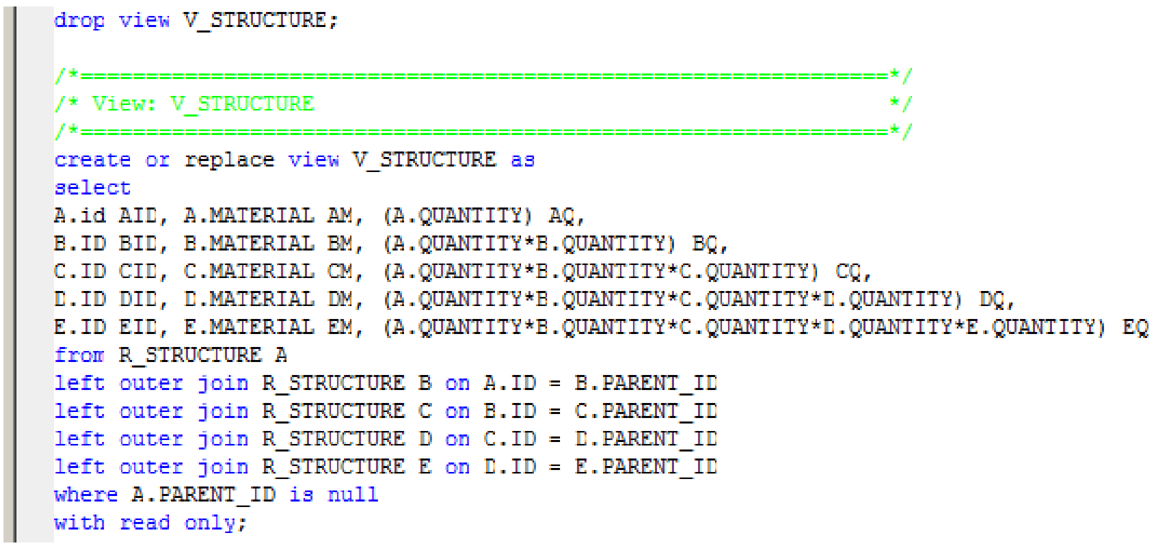


```
SQL> select * from R_OWNER
  2  order by VARIANT_ID, ID;

VAR ID
--- ---
VA  0.0
VA  1.0
VA  1.1
VA  1.2
VA  2.0
VA  2.1
VA  2.3
VA  2.5
VB  0.0
VB  1.0
VB  1.2
VB  2.0
VB  2.1
VC  0.0
VC  1.0
VC  1.1
VC  1.2
VC  2.0
VC  2.1
VC  2.4
VC  2.5

21 rows selected.

SQL>
```

Figure 9. Relations between elements of product and product variants

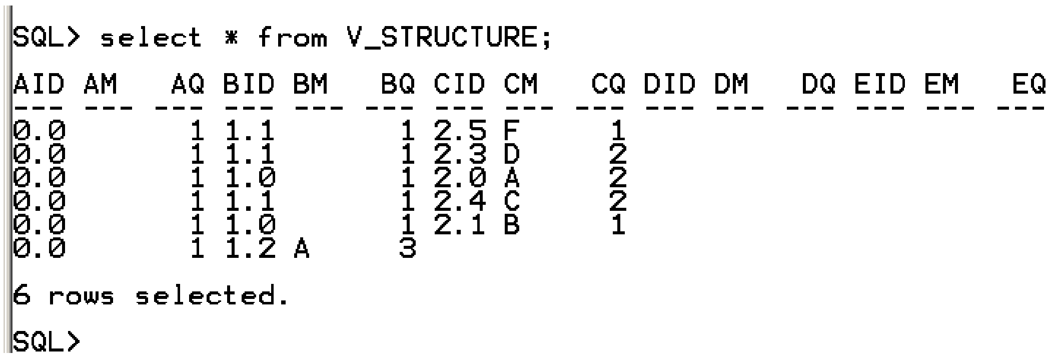
The basic concept employed in the development of BOM generator as part of a RDBMS is the use of views. In database terminology, a view consists of a stored query accessible as a virtual table in a relational database. Unlike ordinary tables in a relational database, a view is not part of the physical schema. It is a dynamic, virtual table computed or collated from data in the database. Changing the data in a table alters the data shown in subsequent of the view. Views provide advantages over tables, such as ability to represent a subset of data contained in a table, views could join and simplify multiple tables into a single virtual table, and views could hide the complexity of data. In a relational database, the primary mechanism for retrieving information from a database is the use of queries. Generally, a query consists of questions presented to the database in a predefined format and the RDBMS uses the Structured Query Language (SQL) as the standard query format.

The main view created in this research is a view to retrieve the path of a bottom element from the top level (product) based on the product structure. For example, based on Figure 2 the path for element 2.4 is element 0.0, element 1.1, and element 2.4. Furthermore, the script used for creating the view is shown in Figure 10 and the example data of the view is shown in Figure 11.



```
drop view V_STRUCTURE;

/*==============================================================*/
/* View: V_STRUCTURE                                            */
/*==============================================================*/
create or replace view V_STRUCTURE as
select
A.id AID, A.MATERIAL AM, (A.QUANTITY) AQ,
B.ID BID, B.MATERIAL BM, (A.QUANTITY*B.QUANTITY) BQ,
C.ID CID, C.MATERIAL CM, (A.QUANTITY*B.QUANTITY*C.QUANTITY) CQ,
D.ID DID, D.MATERIAL DM, (A.QUANTITY*B.QUANTITY*C.QUANTITY*D.QUANTITY) DQ,
E.ID EID, E.MATERIAL EM, (A.QUANTITY*B.QUANTITY*C.QUANTITY*D.QUANTITY*E.QUANTITY) EQ
from R_STRUCTURE A
left outer join R_STRUCTURE B on A.ID = B.PARENT_ID
left outer join R_STRUCTURE C on B.ID = C.PARENT_ID
left outer join R_STRUCTURE D on C.ID = D.PARENT_ID
left outer join R_STRUCTURE E on D.ID = E.PARENT_ID
where A.PARENT_ID is null
with read only;
```

Figure 10. The script for creating the view of path of the elements

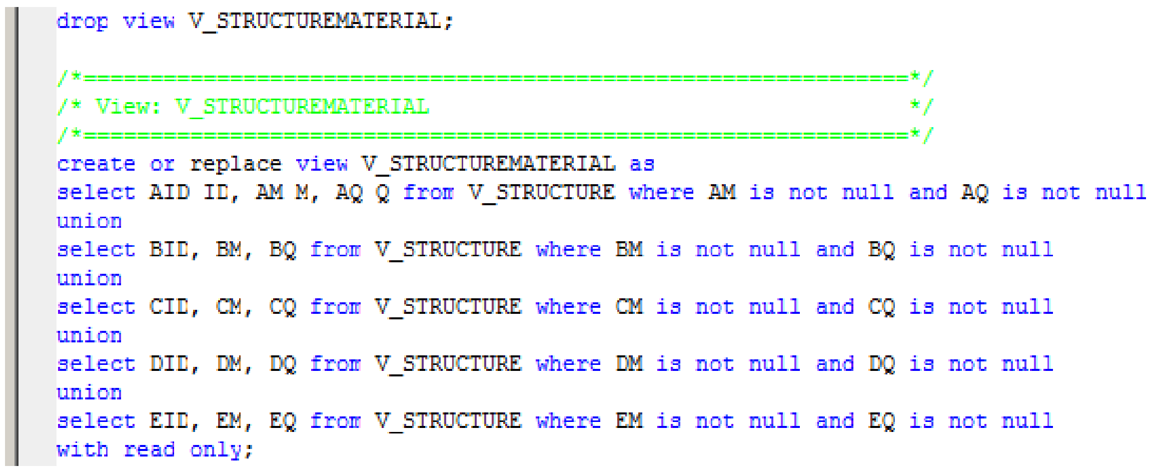


```
SQL> select * from V_STRUCTURE;

AID AM   AQ BID BM   BQ CID CM   CQ DID DM   DQ EID EM   EQ
--- --- --- --- --- --- --- --- --- --- --- --- --- --- ---
0.0       1 1.1       1 2.5 F     1
0.0       1 1.1       1 2.3 D     2
0.0       1 1.0       1 2.0 A     2
0.0       1 1.1       1 2.4 C     2
0.0       1 1.0       1 2.1 B     1
0.0       1 1.2 A     3

6 rows selected.

SQL>
```

Figure 11. The path of product structure elements

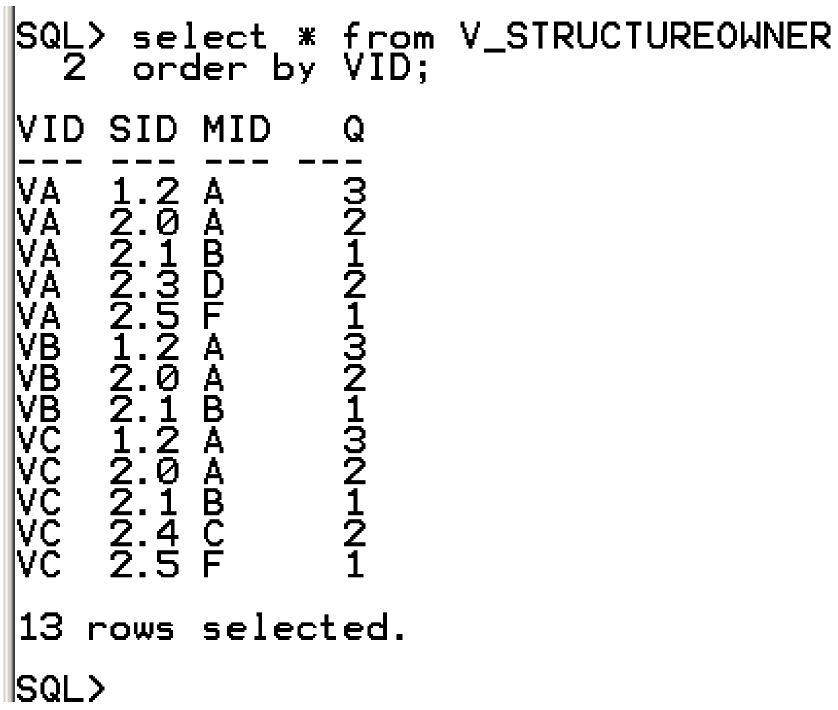
In the script for creating the view above, the depth of level is limited to five levels although the depth of level of the product structure is unlimited. If the depth of level of the product structure is more than five, then the view must be rebuilt in order to make the BOM generator function properly. Based on Figure 11, AID means the ID for elements of level 1 of the product structure, AM means the material ID for elements of level 1 of the product structure, and AQ means the quantity of material owned by elements of level 1 of the product structure. Then BID means the ID for elements of level 2 of the product structure and so on.

After the view of path of the bottom elements has been created, the second view is required to retrieve the elements that have a material from the previous view. In this view, the elements that have a material from column A, B, C, D, and E are collected into one virtual table. The script for creating this view is shown in Figure 12 and the result is shown in Figure 13.



```
drop view V_STRUCTUREMATERIAL;

/*==============================================================*/
/* View: V_STRUCTUREMATERIAL                                    */
/*==============================================================*/
create or replace view V_STRUCTUREMATERIAL as
select AID ID, AM M, AQ Q from V_STRUCTURE where AM is not null and AQ is not null
union
select BID, BM, BQ from V_STRUCTURE where BM is not null and BQ is not null
union
select CID, CM, CQ from V_STRUCTURE where CM is not null and CQ is not null
union
select DID, DM, DQ from V_STRUCTURE where DM is not null and DQ is not null
union
select EID, EM, EQ from V_STRUCTURE where EM is not null and EQ is not null
with read only;
```

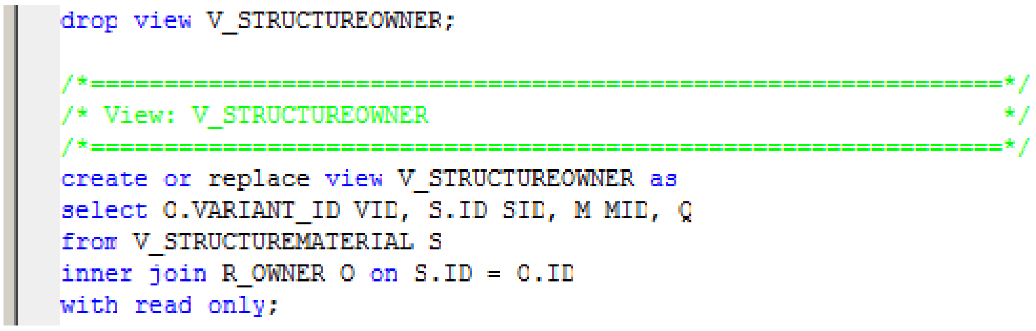
Figure 12. The script for creating the view of collecting elements that have a material



```
SQL> select * from V_STRUCTUREOWNER
  2  order by VID;

VID SID MID   Q
--- --- --- ---
VA  1.2 A     3
VA  2.0 A     2
VA  2.1 B     1
VA  2.3 D     2
VA  2.5 F     1
VB  1.2 A     3
VB  2.0 A     2
VB  2.1 B     1
VC  1.2 A     3
VC  2.0 A     2
VC  2.1 B     1
VC  2.4 C     2
VC  2.5 F     1

13 rows selected.

SQL>
```

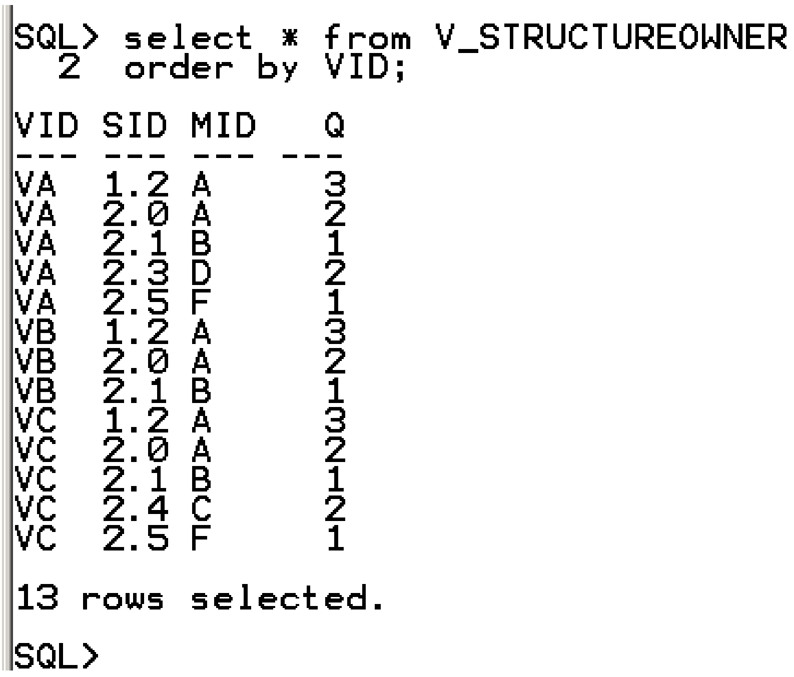
Figure 13. Data of elements that have a material

After creating the second view, the third view is required to join the second view with the product variants that own the elements in the second view. The script for creating this view is shown in Figure 14 and the result is shown in Figure 15.



```
drop view V_STRUCTUREOWNER;

/*==============================================================*/
/* View: V_STRUCTUREOWNER                                       */
/*==============================================================*/
create or replace view V_STRUCTUREOWNER as
select O.VARIANT_ID VID, S.ID SID, M MID, Q
from V_STRUCTUREMATERIAL S
inner join R_OWNER O on S.ID = O.ID
with read only;
```

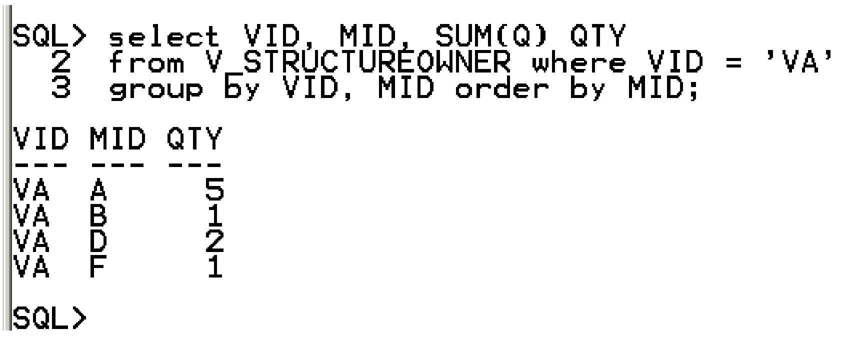
Figure 14. The script for creating the third view



```
SQL> select * from V_STRUCTUREOWNER
  2  order by VID;

VID SID MID   Q
--- --- --- ---
VA  1.2 A     3
VA  2.0 A     2
VA  2.1 B     1
VA  2.3 D     2
VA  2.5 F     1
VB  1.2 A     3
VB  2.0 A     2
VB  2.1 B     1
VC  1.2 A     3
VC  2.0 A     2
VC  2.1 B     1
VC  2.4 C     2
VC  2.5 F     1

13 rows selected.

SQL>
```

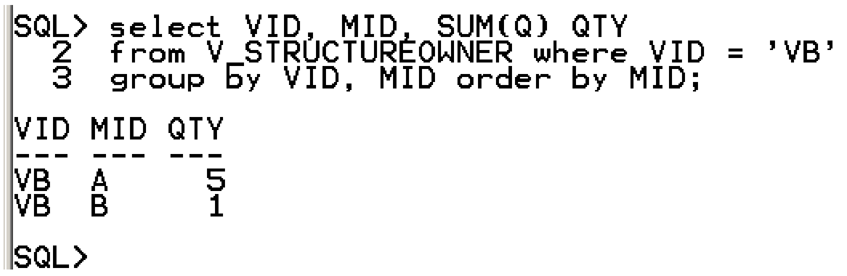
Figure 15. Retrieving data from the third view

After creating the third view, the list of BOM may be generated. By using the third view, an aggregate function, and supplying the product variant data, as shown in Figure 16 for the variant VA, the list of BOM will be retrieved. The query that is used to generate the BOM as shown in Figure 16 is much simpler than the codes that have been implemented for the heuristic rule. The query that is used to generate the BOM for variant VB and VB are shown in Figure 17 and Figure 18 respectively.



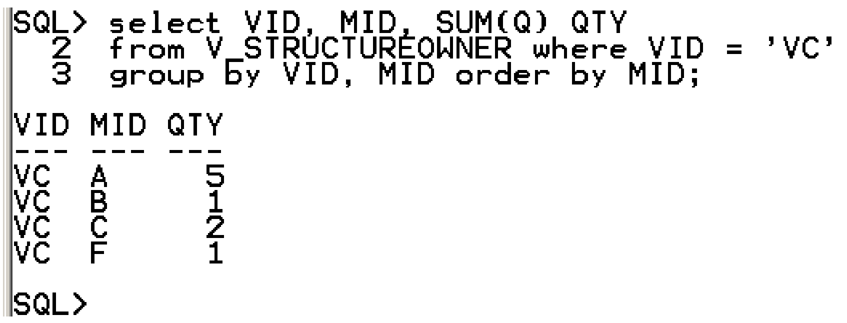
```
SQL> select VID, MID, SUM(Q) QTY
  2  from V_STRUCTUREOWNER where VID = 'VA'
  3  group by VID, MID order by MID;

VID MID QTY
--- --- ---
VA  A     5
VA  B     1
VA  D     2
VA  F     1

SQL>
```

Figure 16. A simple query to generate a BOM for variant VA



```
SQL> select VID, MID, SUM(Q) QTY
  2  from V_STRUCTUREOWNER where VID = 'VB'
  3  group by VID, MID order by MID;

VID MID QTY
--- --- ---
VB  A     5
VB  B     1

SQL>
```

Figure 17. A simple query to generate a BOM for variant VB

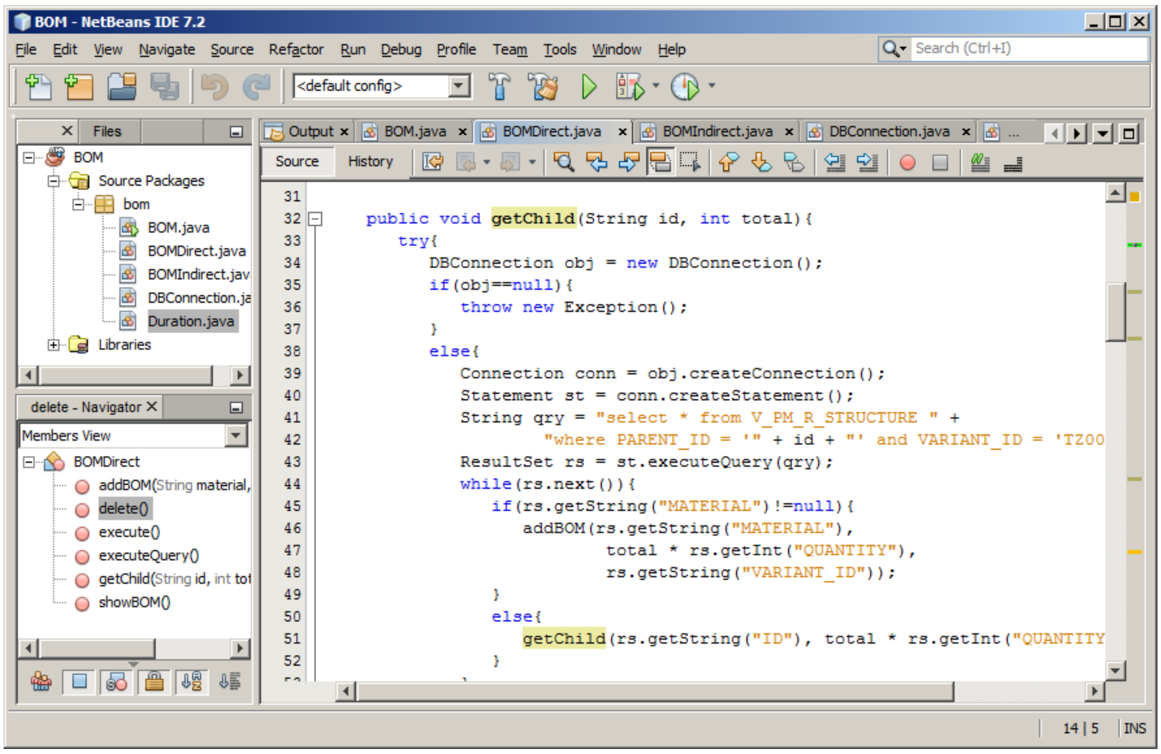


```
SQL> select VID, MID, SUM(Q) QTY
  2  from V_STRUCTUREOWNER where VID = 'VC'
  3  group by VID, MID order by MID;

VID MID QTY
--- --- ---
VC  A     5
VC  B     1
VC  C     2
VC  F     1

SQL>
```

Figure 18. A simple query to generate a BOM for variant VC

Performance Test

To evaluate the proposed method, a performance test was conducted. The basic idea of the test is to compare the process duration of previous and proposed methods. The test has been done by using a simple application written in Java language (see Figure 19). The application implements both the communication with the RDBMS from the proposed method and the heuristic rule from the previous method (see Figure 20 and 21).



```
    public void getChild(String id, int total){
        try{
            DBConnection obj = new DBConnection();
            if(obj==null){
                throw new Exception();
            }
            else{
                Connection conn = obj.createConnection();
                Statement st = conn.createStatement();
                String qry = "select * from V_PM_R_STRUCTURE " +
                        "where PARENT_ID = '" + id + "' and VARIANT_ID = 'TZ00
                ResultSet rs = st.executeQuery(qry);
                while(rs.next()){
                    if(rs.getString("MATERIAL")!=null){
                        addBOM(rs.getString("MATERIAL"),
                                total * rs.getInt("QUANTITY"),
                                rs.getString("VARIANT_ID"));
                    }
                    else{
                        getChild(rs.getString("ID"), total * rs.getInt("QUANTITY
                    }
```

Figure 19. A simple Java application to evaluate the performance

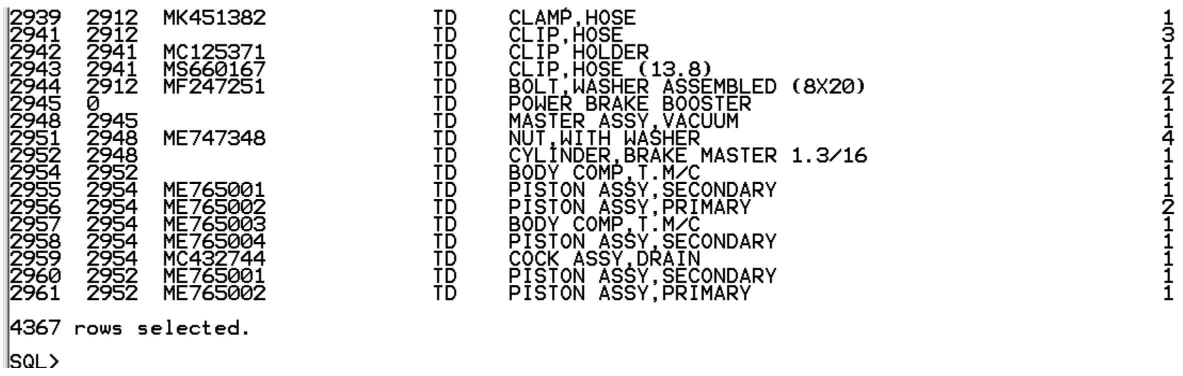
```
procedure BOMProposedMethod is:
input: string variant_id
  1. get the start time
  2. retrive BOM data for variant_id from database
  3. display the BOM
  4. get the finish time
  5. calculate the duration
end BOMProposedMethod
```

Figure 20. Pseudocode for proposed method

```
procedure BOMPreviousMethod is:
input: string variant_id
  1. get the start time
  2. retrieve root of structure for variant_id from database
  3. if root has material data, AddBOMData(material.id,  material.quantity,
     variant_id)
  4. otherwise, GetChild(root.id, root.quantity, variant_id)
  5. retrieve BOM data from temp table
  6. display the BOM
  7. get the finish time
  8. calculate the duration
end BOMPreviousMethod

procedure AddBOMData is:
input: string material_id, material_quantity, variant_id
  1. insert material_id, material_quantity, variant_id into temp table in
     database
end AddBOMData

procedure GetChild is:
input: string node_id, number quantity, string variant_id
  1. retrieve children of node_id from database
  2. for each node on the children
       1. if node has material data, AddBOMData(material.id,
          node.quantity * quantity, variant_id)
       2. otherwise, GetChild(node.id, node.quantity * quantity,
          variant_id)
end GetChild
```

Figure 21. Pseudocode for previous method

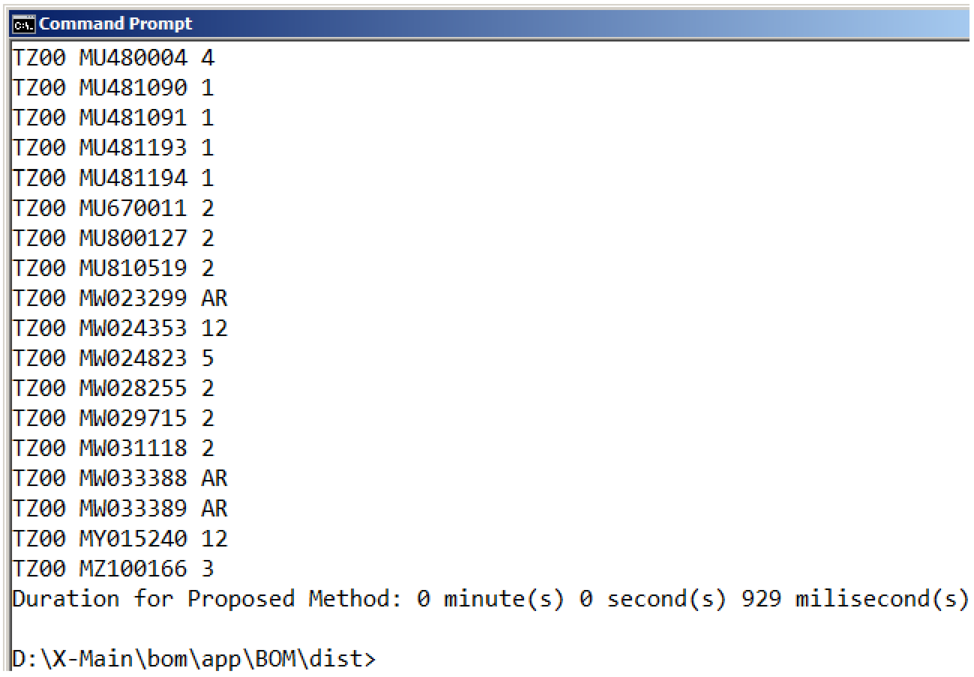
The primary steps used in the test are (1) obtaining the start time, (2) executing the process to generate the list of BOM, (3) obtaining the finish time and (4) calculating the duration. The test used a hierarchical structure of car with 4367 rows of data in comparing the performance. Partial data of hierarchical structure of car used in the test is shown in Figure 22.



```
2939  2912  MK451382          TD    CLAMP,HOSE                                1
2941  2912                    TD    CLIP,HOSE                                 3
2942  2941  MC125371          TD    CLIP HOLDER                               1
2943  2941  MS660167          TD    CLIP,HOSE (13.8)                          1
2944  2912  MF247251          TD    BOLT,WASHER ASSEMBLED (8X20)              2
2945  0                       TD    POWER BRAKE BOOSTER                       1
2948  2945                    TD    MASTER ASSY,VACUUM                        1
2951  2948  ME747348          TD    NUT,WITH WASHER                           4
2952  2948                    TD    CYLINDER,BRAKE MASTER 1.3/16              1
2954  2952                    TD    BODY COMP,T.M/C                           1
2955  2954  ME765001          TD    PISTON ASSY,SECONDARY                     1
2956  2954  ME765002          TD    PISTON ASSY,PRIMARY                       2
2957  2954  ME765003          TD    BODY COMP,T.M/C                           1
2958  2954  ME765004          TD    PISTON ASSY,SECONDARY                     1
2959  2954  MC432744          TD    COCK ASSY,DRAIN                           1
2960  2952  ME765001          TD    PISTON ASSY,SECONDARY                     1
2961  2952  ME765002          TD    PISTON ASSY,PRIMARY                       1

4367 rows selected.

SQL>
```

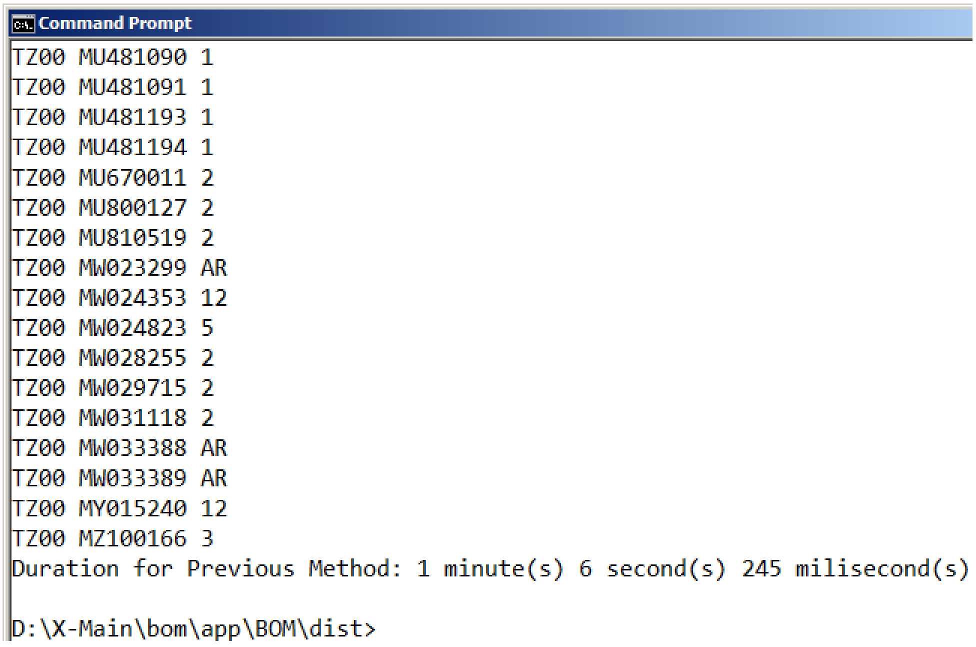
Figure 22. Partial data of hierarchical structure of car

The test was conducted in two phases. The first phase is evaluation of previous method and the second phase is evaluation of proposed method. Each phase is performed 10 times. Examples of screenshot of the application for evaluation of proposed and previous methods are shown in Figure 23 and Figure 24, respectively. Each screenshot of application contains information about BOM and the duration required to generate the BOM. Each line of BOM as shown in Figure 20 or Figure 21 represents the variant ID, the material ID and the quantity. In the list, the quantity AR means "as required". After executing the Java application 10 times for each method, the results of the tests are shown in Table 2.



```
TZ00 MU480004 4
TZ00 MU481090 1
TZ00 MU481091 1
TZ00 MU481193 1
TZ00 MU481194 1
TZ00 MU670011 2
TZ00 MU800127 2
TZ00 MU810519 2
TZ00 MW023299 AR
TZ00 MW024353 12
TZ00 MW024823 5
TZ00 MW028255 2
TZ00 MW029715 2
TZ00 MW031118 2
TZ00 MW033388 AR
TZ00 MW033389 AR
TZ00 MY015240 12
TZ00 MZ100166 3
Duration for Proposed Method: 0 minute(s) 0 second(s) 929 milisecond(s)

D:\X-Main\bom\app\BOM\dist>
```

Figure 23. A screenshot of the application for evaluation of proposed method



```
TZ00 MU481090 1
TZ00 MU481091 1
TZ00 MU481193 1
TZ00 MU481194 1
TZ00 MU670011 2
TZ00 MU800127 2
TZ00 MU810519 2
TZ00 MW023299 AR
TZ00 MW024353 12
TZ00 MW024823 5
TZ00 MW028255 2
TZ00 MW029715 2
TZ00 MW031118 2
TZ00 MW033388 AR
TZ00 MW033389 AR
TZ00 MY015240 12
TZ00 MZ100166 3
Duration for Previous Method: 1 minute(s) 6 second(s) 245 milisecond(s)

D:\X-Main\bom\app\BOM\dist>
```

Figure 24. A screenshot of the application for evaluation of previous method

Table 2. Results of Performance Test

	Duration for Previous Method				Duration for Proposed Method			
Test#	**min.**	**sec.**	**ms**	**Total (ms)**	**min.**	**sec.**	**ms**	**Total (ms)**
1	1	6	637	66,637	0	1	352	1,352
2	1	7	226	67,226	0	0	923	923
3	1	6	960	66,960	0	1	280	1,280
4	1	7	152	67,152	0	1	303	1,303
5	1	8	300	68,300	0	1	316	1,316
6	1	3	960	63,960	0	1	291	1,291
7	1	7	692	67,692	0	1	301	1,301
8	1	6	662	66,662	0	1	308	1,308
9	1	2	121	62,121	0	0	938	938
10	1	6	245	66,245	0	0	929	929
Statistics								
Average (ms)				66,296.0	Average (ms)			1,194.1
Minimun (ms)				62,121	Minimun (ms)			923
Maximum (ms)				68,300	Maximum (ms)			1,352
Range (ms)				6,179	Range (ms)			429

As shown in Table 2, the duration of previous method fluctuated between 62,121 ms to 68,300 ms with the average duration of 66,296.0 ms. The duration of the proposed method also fluctuated from 923 ms to 1,352 ms and the average of 1,194.1 ms. Based on the average durations, the proposed method took only 1.8% of previous method duration to generate the BOM from the same data. It shows that the proposed method has taken much less time in processing than the previous method.

Conclusions

Improvement of the BOM generator for product variants has been developed through object views. Based on the performance test, the proposed method average duration is 1.8% of that of the previous method to generate the BOM from the same data. It is because of the processing using some codes outside the RDBMS requires more time than processing using native commands. Furthermore, the command to obtain a list of a BOM for a product variant is relatively simple.

Although the proposed method takes less time in processing and is easier to use compared to the previous method, it has two limitations. The first limitation is the depth of levels of the product structure. Although the physical data model of the product structure has been developed to model the depth of levels without limitation, but the view for paths of the product structure (as shown in Figure 11) can only function properly to 5 depths of levels. If it is required to have more than 5 depths of levels, the view must be rebuilt. However, in practice the depth of levels of the product structure is rarely larger than 10 depths of levels. For example, the depth of levels for an automobile is only 6 or 7. The second limitation is the developed model of product has not specified a validity attribute for elements in the product structure. The attribute will be useful to limit the period of validity of an element, such as in the case of change of components. Hence, the proposed methods have yet to accommodate any changes of product.

References

[1] J. Guoli, G. Daxin, and F. Tsui, "Analysis and implementation of the BOM of a tree-type structure in MRP II," *Journal of Materials Processing Technology*, Vol. 139, No. 1-3, pp. 535-538, 2003.

[2] P.W. Stonebraker, "Restructuring the bill of material for productivity: A strategic evaluation of product configuration," *International Journal Production Economics*, Vol. 45, No. 1-3, pp. 251-260, 1996.

[3] S. Zhu, D. Cheng, K. Xue, and X. Zhang, "A unified bill of material based on STEP/XML," In: *The 10th International Conference on Computer Supported Cooperative Work in Design, 2006*, Nanjing, pp. 267–276, 2007.

[4] F.B. Watts, *Engineering Documentation Control Handbook*, Noyes Publication, New York, 2000.

[5] J. Jiao, M.M. Tseng, Q. Ma, and Y. Zhou, "Generic bill-of-materials-and-operations for high-variety production management," *Journal of Concurrent Engineering: Research and Application*, Vol. 8, No. 4, pp. 297-322, 2000.

[6] F. Erens, H. Hegge, E.A. van Veen, and J.C. Wortmann, "Generative bills-of-materials: An overview," In: *The IFIP WG5.7 Working Conference on Integration in Production Management Systems*, Eindhoven, pp. 93 – 113, 1992.

[7] J.W.M. Bertrand, M. Zuijderwijk, and H.M.H. Hegge, "Using hierarchical pseudo bills of materials for custom order acceptance and optimal material replenishment in assemble to order manufacturing of non modular products," *International Journal of Production Economics*, Vol. 66, No. 2, pp. 171-184, 2000.

[8] K.A. Olsen, and P. Saetre, "Describing products as executable programs: Variant specification in a customer-oriented environment," *International Journal of Production Economics*, Vol. 56-57, No. 1, pp. 495-502, 1998.

[9] J.C.H. Matias, H.P. Garcia, J.P. Garcia, and A.V. Idoipe, "Automatic generation of a bill of materials based on attribute patterns with variant specifications in a customer-oriented environment," *Journal of Materials Processing Technology*, Vol. 199, No. 1-3, pp. 431-436, 2008.

[10] Y.Y Martawirya, S. Raharno, and I. Nurhadi, "Development of bill of materials of product variants," In: *The 9th Asia Pacific Industrial Engineering and Management Systems Conference*, Bali, pp. 2869 – 2870, 2008.

[11] G. Nandakumar, "Bills of material processing with a SQL database," *Computers and Industrial Engineering Journal*, Vol. 18, No. 4, pp. 471- 483, 1990