# Rules Extractor from PLSQL

Sumit Dubey
Department of Computer Science
Manipal University
256 (Ground Floor), Okhla Industrial Estate-III,
New Delhi - 110020, India
sumitgpl@gmail.com

Kanchan Lata
Faculty of CS/IT
Dewan V.S.Institue of Engineering &Technology
Partapur Bypass Road, Meerut (U.P.)
kanchan.jssit@gmail.com

***Abstract*— The complexity of real production systems implies more difficulties to make an efficient monitoring and especially fault diagnosis. We propose a new method supporting the operator to find the cause and the origin of a fault. To obtain a diagnosis aid system that is both reactive and easy to configure, we define a set of artificial intelligence tools using neuron network techniques. The interest of these techniques is to combine the neural networks learning capabilities and PLSQL business logics and extracting the rules into natural language.**

***Keywords- Business rules extraction, PLSQL, natural language processing, neural network.***

## I. INTRODUCTION

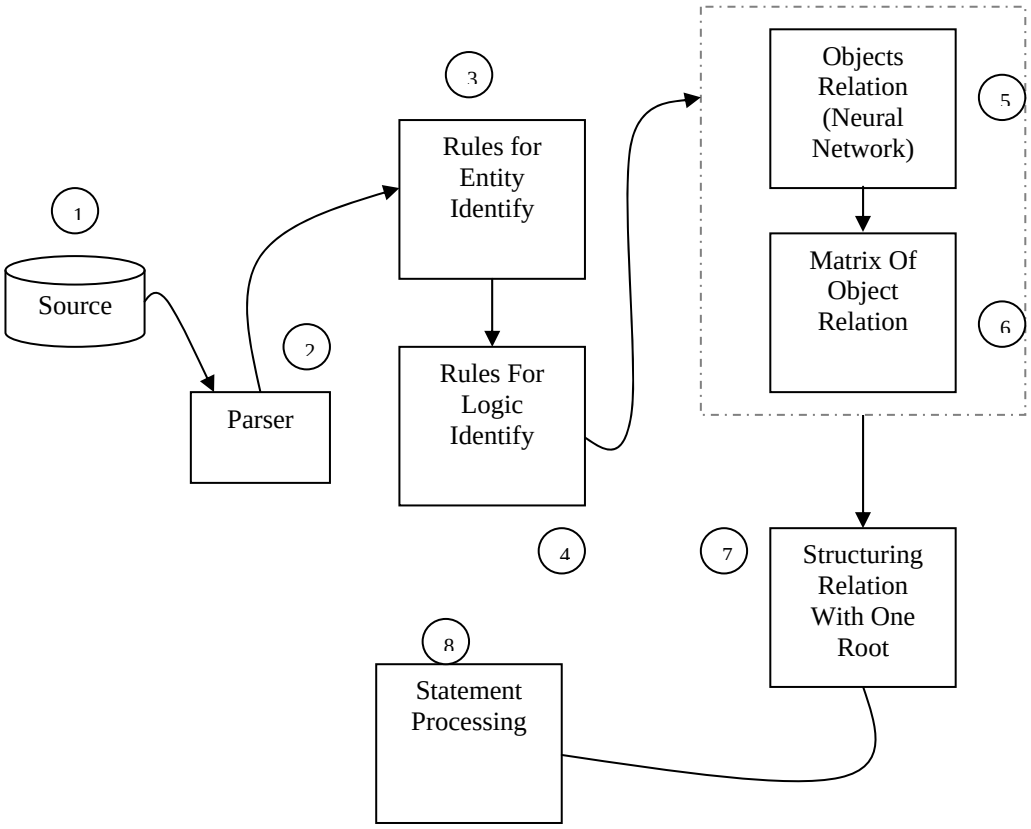
Oracle's Procedural Language/SQL or PL/SQL is a language that extends the ANSI/ISO standard, Structured Query Language (SQL). It is modeled after the Ada-83 programming language with several significant extensions. While many people think that PL/SQL is a purely procedural language, possibly because of the use of "Procedural" in its name, it implements a significant number of the above concepts. Support for each of the above concepts is discussed below.

- The concept of *class* is supported in PL/SQL by the *package* and *object type* constructs. PL/SQL packages and object types have attributes and behavior. The PL/SQL package should not be confused with other programming language package constructs which typically use the package as a way of grouping classes. The PL/SQL package concept is derived directly from the Ada-83 package concept.
- The concept of *object* is supported in PL/SQL by executing instances of packages and types. Packages are instantiated (loaded) upon first reference. Object and collection types require constructor calls just as for Java and C++.
- *Inheritance* for packages is supported on a limited basis in PL/SQL. Inheritance for packages is limited to single interface inheritance. Each package implements a single interface that is used to specify the public interface for the package. The interface, or package specification, is implemented in the package body. Object types support single inheritance for both interface inheritance and class inheritance.
- Packages and object types *encapsulate* data and methods. Packages provide for access control to the elements where elements defined in the package specification have public access and elements defined in the package body have package access. All elements defined in an object type have public access.
- *Abstraction* is supported only by object types. This is done through the creation of an object type that cannot be instantiated and is not final. Note that operations specified in package and object type specifications are inherently abstract.
- Method *polymorphism* is supported by packages and object types. Instance and type polymorphism is supported only by object types.

The above points show that PL/SQL is very close to being a fully object-oriented language. This attribute is helpful for the PLSQL code to UML creation.

Now let us see the drawback of above existing PLSQL to UML converter. In the above structure we are assuming whole block as a class and object and after conversion we have an overall block structure not at the code level. Take an example we have only one procedure (having multiple insertion and update) and using the above attribute we trying to convert into UML. Its only show the all the agents based on inputs and putting insert and update activity without going to check the dependencies based on insert and update condition. It's not going to tell about what exactly going on with all variables and their dependencies to insert and update data into database

## II. SYSTEM STRUCTURE




According to the above block structure, below are the detailed steps….

### *A. Source*

First In this section we are assuming we have a PLSQL source code, on which we are going to analyze for the further enhancement (analysis). This source may be the single PLSQL block which contains BEGIN and END condition without any variable declaration or it may be the full complicated PLSQL bock which having some parameter, some variable definition (may be with some cursor or row type), some loops, some condition and basis of these condition values may be insertion into table or manipulating other variable values.

In the second situation, we can say this is our procedure, function or a package definition, which we are going to analyze. So in this part we can submit a block of PLSQL code or a functional unit name with their type.

### *B. Parser*

Like the other programming parser, this is having same functionality to reading code line by line and word by word. We can use parser to directly take the input from the USER_SOURCE (this table having source code of all procedure, package, trigger etc.), if we are given only PLSQL block name. This table having source code line by line, so that will be easier to parser to break structure for the further below section use.

*C. Rules of entity identify*

This is section for identify the entity. We are passing whole PLSQL block's line to this unit and this unit is able to identify the embedded entity on the same line. This section not only identifying also creating a table structure like below to identify which entity comes fist and at which level. In this section we are including variables, condition and loop block etc into the entity. That means this section just having some set of rules to identify the PLSQL block entity by name not by logic. To identify the logic this section further going to call the next section which is logic identifies section.

After reading plsql code, it's going to make a sense of which logic is going on and what is its status. And based on which condition logic is going on its calling making an entry into below tables.

i) - conditional block: - it's having the structure like

Table 1

| unique# | Entity | start | end | next_entity |
|---|---|---|---|---|
| 1 | Begin | 1 | 300 | |
| 2 | if | 12 | 30 | 3 |
| 3 | if | 15 | 20 | |

Table 2

| unique# | sub_unique# | entity | start | end |
|---|---|---|---|---|
| 2 | 1 | if | 12 | 18 |
| 2 | 2 | elsif | 19 | 25 |
| 2 | 3 | else | 26 | 30 |

ii) - variable values: - it's having below structure

| var_unique# | var_entity | value | datatype | length |
|---|---|---|---|---|
| 1 | v_val1 | "value1" | varchar | 200 |
| 2 | v_val2 | "01-aug-2010" | date | - |

*D. Rules of logic identify*

In this section we are going to identify the logic behind the given PLSQL entity. As we know in the above section, into 'entity identify' we are identifying entity and then calling this 'logic identify' section to extract logic attached with this entity.

The major difference between both term 'entities identify' and 'logic identify'- in first we are identifying entity (like if, loop etc) in second we are extraction related logic with this entity (like variable dependencies). In first we are passing parameter line wise line, into second we are calling from first one with whole entity set

*E. Object relation (neural network) and Matrix of object relation*

In this section we are creating a neural network and an object dependencies matrix. Here we are taking variable as an object, because in a PLSQL block every conditional logic dependent on a variable or some combination of variable. Lets take the example of below PLSQL block, which having 6 if conditional block and their dependent statements.

```
If 1….
{
        Statement 1;
        If 2…..
         {
          Statement 2;
         }

          If 3…..
          {
            Statement 3;
```

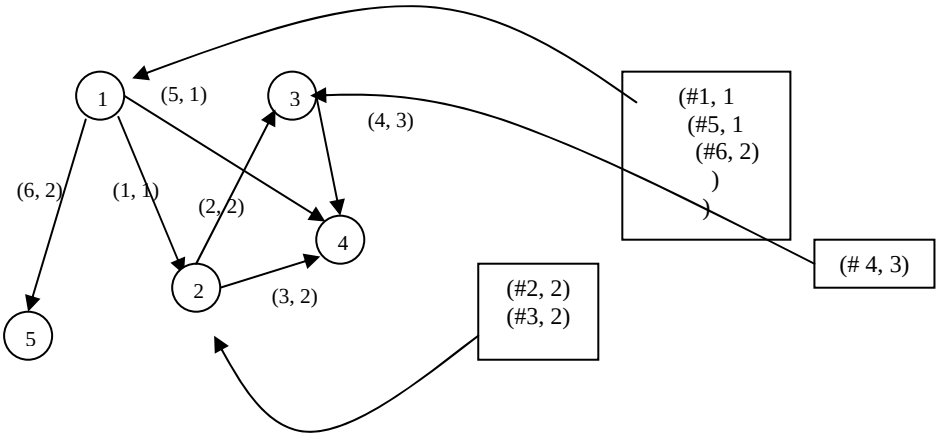
```
                    If 4…..
                     {
                       Statement 4;
                     }
                }
}

If 5….
{
     If 6…..
      {
        Statement 5 and 6;
      }
}
```

| X | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | | 5 | 6 | | |
| 2 | | 0 | 2 | 3 | | | |
| 3 | | | 0 | 4 | | | |
| 4 | | | | 0 | | | |
| 5 | | | | | 0 | | |
| 6 | | | | | | 0 | |
| 7 | | | | | | | 0 |

| id | condition | level | con_val_id |
|---|---|---|---|
| 1 | if1 | 1 | 1 |
| 2 | if2 | 2 | 2 |
| 3 | if3 | 2 | 3 |
| 4 | if4 | 3 | 4 |
| 5 | if5 | 1 | 5 |
| 6 | if6 | 2 | 6 |

After scanning the above block we can define a matrix like above, which is showing the object dependencies to others. And having one id for each object dependency. Because each dependencies coming with some if condition, so we can draw this dependencies id with those condition and their level (see the second matrix above).




After completing object dependencies matrix creation, we can directly create the neural network for the same PLSQL block. This neural network is a logical not actual. To store actual neural network we need to create RDBMS‘s tables.

**Scenario 1**:- in case we have multi value dependencies like the below

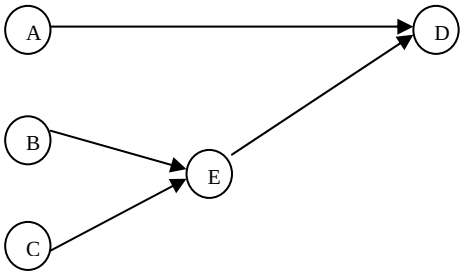
```
If (val_var1 =100) OR (val_var2=’TMP’ and val_var3=’XYZ’) then
   Val_name:=’SUMIT’;
End if;
```

In this scenario we need to create some node value between two nodes. Here in we need to create one node for val_var1=100, second one for val_var2=’TMP’, third for val_var2=’XYZ’ and fourth for the truth condition

acceptance for (val_var2='TMP' and val_var3='XYZ') and all above having threshold value 1. If node one condition is true then it'll return 1 other wise 0 and next node get activated if input get 1 because of threshold value is 1 and total should be >0.

Let's assign expression some id to easy to show in neural network like below

val_var1 → A, AD → 100
val_var2 → B, BE→ 'TMP'
val_var3 → C , CE → 'XYZ'
val_name → D, ED → 'TMP' + 'XYZ'.




So according to the above E would have …

val_var2 ; val_var3 → E

And for the above table structure would be like

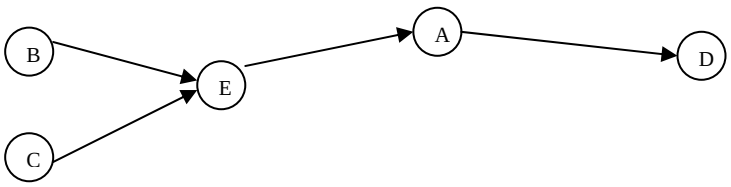
| rid | in | out | vertex_nm | property | value |
|---|---|---|---|---|---|
| V1 | | E1 | A | val_var1 | |
| V2 | | E2 | B | val_var2 | |
| V3 | | E3 | C | val_var3 | |
| V4 | E1;E4 | | D | val_name | 'SUMIT' |
| V5 | E2;E3 | E4 | E | val_var2 ;val_var3 | |

| rid | node1 | node2 | value |
|---|---|---|---|
| E1 | V1 | V4 | 100 |
| E2 | V2 | V5 | 'TMP' |
| E3 | V3 | V5 | 'XYZ' |
| E4 | V5 | V4 | 'TMP' + 'XYZ' |

**Scenario 2:-** in case we have multi value dependencies at different level like the below

```
If (val_var2='TMP' and val_var3='XYZ') then
   If (val_var1 =100) then
   val_name:='SUMIT';
   End if;
End if;
```

Threshold value and active condition are same as above




And for the above table structure would be like

| rid | in | out | vertex_nm | property |
|---|---|---|---|---|
| V1 | E4 | E1 | A | val_var1 =100 |
| V2 | | E2 | B | val_var2='TMP' |
| V3 | | E3 | C | val_var3='XYZ' |
| V4 | E1 | | D | val_name:='SUMIT' |
| V5 | E2;E3 | E4 | E | val_var2='TMP' and val_var3='XYZ' |

| rid | node1 | node2 |
|---|---|---|
| E1 | V1 | V4 |
| E2 | V2 | V5 |
| E3 | V3 | V5 |
| E4 | V5 | V4 |

*F. Structuring relation with one root*

In this section we are structuring the above whole neural network and object matrix into a one root structure. This one root structure will help to store the PLSQL block's integrated logic into one statement. This will also help to redraw and other conversion like into other programming language.

This will help to the back tracking of the logic, like we have the conditional logic structure and based on the below section Statement Processing, we have a statement for this logic. Now we want to add one more rule into our existing process, then we can easily identify where we need to make changes and what dependencies need to break and what need to add up.

This one root structure is very easy to draw; just need to identify the start and end point of the above neural network as well need to know the condition level. Like the above neural network we can see, 1 is don't have any in degree that means we can choose it as a starting point. After that next thing which we need to identifying is its weight. Into above neural network we can see, we have three out degree but we can select only one for start, so we are choosing (1, 1) based one minimum condition level and minimum condition value id. Same thing we need to continue for the further synapse and next node until we have finished.

Every time before moving to the next synapse, we need to maintain our root structure. After completing all network structure will be like below….

```
(#1->
   (#2->
      (#3->
        (#4->)
      )
      (#4->)
   )
   (#4->)
   (#5->)
)
```

*G. Statement processing*

In the statement processing we are assuming we are going to draw whole PLSQL block into an English sentence. This whole part will work on natural language processing (NLP) like identifying the part of speech and processing them as statement. For the statement processing we can divide it into two parts

1) - a summary statement whose showing the what going on into this PLSQL block as a summary.

2) - this section having detailed rules about all the logic written into the PLSQL block. This section is followed by the above first section. In this section we are trying to show all logic, after reading all dependencies and all inputs. If there are more then one dependencies level then we need to conceder all into one rule Like:-

```
If var_val_1 > 100 and var_val_2 = 'TMP' Then
var_name:='SUMIT';
End if;
```

As per the above condition logic, statement should parse like –

. **If var_val is greater then 100 and var_val2 equal to TMP then var_name will be SUMIT.**

Take the second scenario,

```
If var_val_1 > 100 and var_val_2 ='TMP' then
```

```
 If var_val_3 = '10-jan-2012' then
   Var_name:='SUMIT';
 Else
   Var_name:= 'TIMUS';
 End if;
End if;
```

For the above condition logic, statement will be:-

**if var_val_1 greater then 100 and var_val_2 equal to TMP then system will check the next condition var_val_3 equal to '10-jan-2012' and basis of this assign var_name equal to SUMIT , other wise it will be TIMUS.**

the above both scenario having a simple example, lets take an example which having the some complex INSERT INTO, UPDATE or DELETE condition…

.

## III. EXAMPLE

### *A. Existing Process*

We have a system which can calculate the no of pay, based on the master table's column bill_interval. Actually this bill_interval having value like 30, 60, 90, 180 that showing values respectively like 360/30=12, 360/ 60= 6, 360/90 = 4, 360/180 =2. That means if we are putting some value into the bill_interval, based on this automatically system will define how many billing should be there into system.

Code will be like …….

```
1   PROCEDURE create_billing (pv_offer_id IN NUMBER, pv_master_id IN NUMBER)
2   IS
3   lv_bill_interval   master_tab. bill_interval%TYPE;
4   lv_nopay                NUMBER;
5   lv_bill_date       DATE;
6   lv_ordr_price      NUMBER;
7   lv_order_id        NUMBER;
8   BEGIN
9   SELECT bill_interval
10  INTO lv_bill_interval
11  FROM master_tab
12  WHERE master_id = pv_master_id;
13
14  lv_nopay := 360 / lv_bill_interval;
15  lv_bill_date := SYSDATE;
16
17  SELECT (order_price / lv_nopay)
18  INTO lv_ordr_price
19  FROM order_offer
20  WHERE offer_id = pv_offer_id;
21
22  FOR rec INTO lv_nopay loop
23
24  SELECT order_seq.NEXTVAL
25  INTO lv_order_id
26  FROM DUAL;
27
28  INSERT INTO billing (order_id,order_date,order_amt)
29  VALUES (lv_order_id,lv_bill_date,lv_ordr_price);
30
31  lv_bill_date:=lv_bill_date+bill_interval;
32
```

```
33      END LOOP;
34      END;
```

Step1 Source:-

In this section we are passing the above whole procedure. If this procedure is not created into database then we need to pass whole code otherwise procedure name is sufficient.

Step2 Parser:-

This is having same functionality to reading code line by line and word by word. We can use parser to directly take the input from the USER_SOURCE (this table having source code of all procedure, package, trigger etc.), if we are given only PLSQL block name. This table having source code line by line, so that will be easier to parser to break structure for the further below section use.

Step3 Rules of entity identify: -

After moving into entity identify, we are identifying the different entity of the above PLSQL block. And table structure would be like

rec <= lv_nopay.

i) - conditional block

| unique# | Entity | start | end | next_entity |
|---|---|---|---|---|
| 1 | Proc | 1 | 34 | 2 |
| 2 | Begin | 3 | 34 | 3 |
| 3 | Select | 9 | 12 | 4 |
| 4 | Select | 17 | 20 | 5 |
| 5 | For | 22 | 33 | # |

| unique# | sub_unique# | entity | start | end |
|---|---|---|---|---|
| #5 | #1 | Select | 24 | 26 |
| #5 | #2 | Insert | 28 | 29 |

ii) - variable values

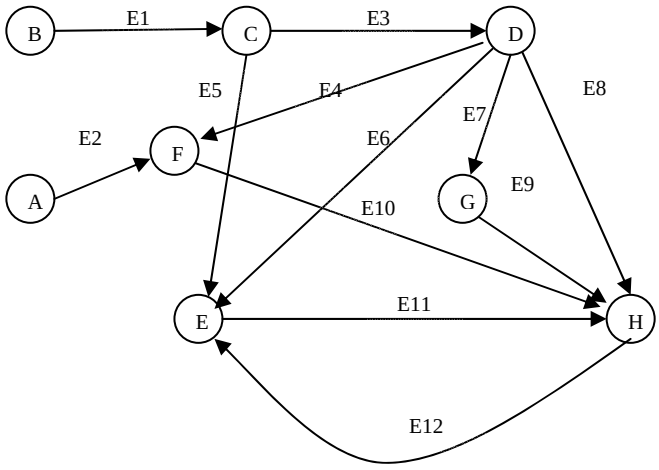
| var_unique# | var_entity | value | datatype | length |
|---|---|---|---|---|
| 1 | lv_bill_interval | | master_tab. bill_interval | TYPE |
| 2 | lv_nopay | | NUMBER | - |
| 3 | lv_bill_date | SYSDATE | DATE | - |
| 4 | lv_ordr_price | | NUMBER | - |
| 5 | lv_order_id | | NUMBER | - |
| 6 | pv_offer_id | | NUMBER | - |
| 7 | pv_master_id | | NUMBER | - |

Step4 Rules of logic identify: -

In this section we are passing code block as per the above conditional block line by line. That means firstly we are passing id 1 and then identifying what happing with that block and this block is related to which logic. Eg. Unique id 5 FOR, having some looping condition this condition getting fetched.

Step 5 & 6 Object relation (neural network) and Matrix of object relation: -

pv_offer_id → A, AF → #SELECT2
pv_master_id → B, BC→ #SELECT1
lv_bill_interval → C , CD → 360 / #C
lv_nopay → D, DF → #SELECT2
lv_bill_date → E
lv_ordr_price → F
lv_order_id → G
#INSERT1 → H



| rid | in | out | vertex_nm | property | Value |
|---|---|---|---|---|---|
| V1 | | | A | pv_offer_id | |
| V2 | | | B | pv_master_id | |
| V3 | | | C | lv_bill_interval | |
| V4 | | | D | lv_nopay | |
| V5 | | | E | lv_bill_date | '21-feb-2012' |
| V6 | | | F | lv_ordr_price | |
| V7 | | | G | lv_order_id | |
| V8 | | | H | #INSERT1 | |

| rid | node1 | node2 | Value |
|---|---|---|---|
| E1 | V2 | V3 | #SELECT2 |
| E2 | V1 | V6 | #SELECT1 |
| E3 | V3 | V4 | 360 / #C |
| E4 | V4 | V6 | #SELECT2 |
| E5 | V3 | V5 | #C + #E |
| E6 | V4 | V5 | # FOR |
| E7 | V4 | V7 | # FOR |
| E8 | V4 | V8 | # FOR |
| E9 | V7 | V8 | TRUE |
| E10 | V6 | V8 | TRUE |
| E11 | V5 | V8 | TRUE |
| E12 | V8 | V5 | TRUE |

Step7 structuring relation with one root: -

```
(#1->
   (#2->
          (#3->
            B -> C
          )
          (#->
            C -> D
          )
          (#->
            SYSDATE -> E
          )
          (#4->
            A + D -> F
          )
          (#5->
            D->
                  (#5, #1
                  )
                  (#5, #2
                    G + E + F -> H
                  )
                  (#
                    E + C -> E
                  )
          )
   )
)
```

Step4 statement processing: -

Summary statement:-

Based on two parameters Pv_offer_id and Pv_master_id, calculating the no of pay , bill date and order price. After all calculation we are inserting into billing table for all no of pay records.

Rules:-

1) - calculating bill interval based on input pv_master_id from table master_tab.
2) - calculating no of pay as 360/ billing interval.
3) - assigning bill date initial value as sysdate.
4) - calculating order price from order_offer based on input pv_offer_id.
5) - taking order id from sequence order_seq.

6) - inserting above calculated order id, bill date and order price into billing table.
7) - after insertion increasing bill date value with bill interval

*B. Enhancement*

Now we have explored above PLSQL procedure into neural network, object dependencies and one root structure. And also having rules and statement summary for the same.

Above system is working fine and generating a lot of revenue but after some time business analyze that they are doing fixed bill interval for a master id. But now time to give this liability to customer to choose billing interval.

According to new changes, rules will be like.

Changed Rules:-

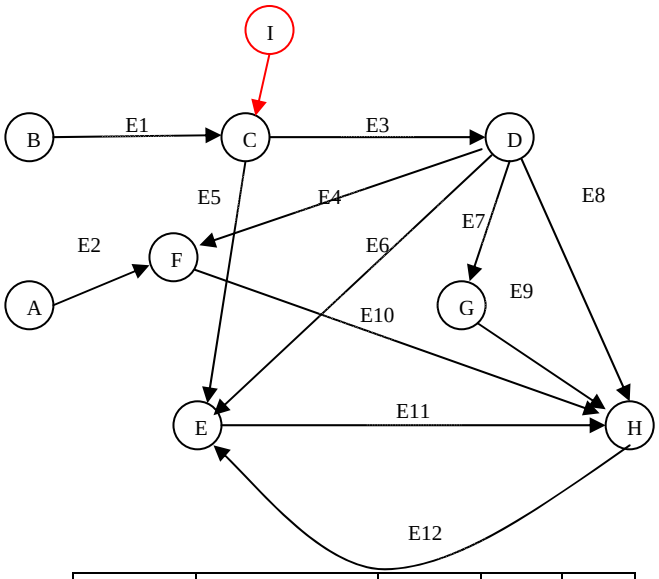
1) - calculating bill interval based on input pv_master_id from table master_tab if input bill interval is null, other wise input bill interval will be the base bill interval.
2) - calculating no of pay as 360/ billing interval.
3) - assigning bill date initial value as sysdate.
4) - calculating order price from order_offer based on input pv_offer_id.
5) - taking order id from sequence order_seq.
6) - inserting above calculated order id, bill date and order price into billing table.
7) - after insertion increasing bill date value with bill interval.

Changed structuring relation with one root :-

```
(#1->
   (#2->
         (#6->
               (#6, #1
                 I is NULL
                     (#3
                       B -> C
                     )
               )
               (#6, #2
               I - > C
               )
         )
         (#->
           C -> D
         )
         (#->
           SYSDATE -> E
         )
         (#4->
           A + D -> F
         )
         (#5->
           D->
               (#5, #1
               )
               (#5, #2
                 G + E + F -> H
               )
               (#
                 E + C -> E
               )
         )
   )
)
```

Changed Object relation (neural network) and Matrix of object relation:-

pv_offer_id → A, AF → #SELECT2
pv_master_id → B, BC→ #SELECT1
lv_bill_interval → C , CD → 360 / **#C**
lv_nopay → D, DF → #SELECT2
lv_bill_date → E
lv_ordr_price → F
lv_order_id → G
#INSERT1 → H
**pv_bill_interval → I**




Changed entity blocks:-

| unique# | Entity | start | end | next_entity |
|---|---|---|---|---|
| 1 | Proc | 1 | 37 | 2 |
| 2 | Begin | 3 | 37 | 3 |
| ~~**3**~~ | ~~**Select**~~ | ~~**9**~~ | ~~**12**~~ | ~~**4**~~ |
| 4 | Select | 20 | 23 | 5 |
| 5 | For | 25 | 36 | # |
| **6** | **IF** | **9** | **16** | **4** |

| unique# | sub_unique# | entity | start | end |
|---|---|---|---|---|
| #5 | #1 | Select | 24 | 26 |
| #5 | #2 | Insert | 28 | 29 |
| **#6** | **#1** | **IF** | **9** | **13** |
| **#6** | **#2** | **Select** | **10** | **13** |
| **#6** | **#3** | **Else** | **14** | **16** |

| var_unique# | var_entity | value | datatype | length |
|---|---|---|---|---|
| 1 | lv_bill_interval | | master_tab. bill_interval | TYPE |
| 2 | lv_nopay | | NUMBER | - |
| 3 | lv_bill_date | SYSDATE | DATE | - |
| 4 | lv_ordr_price | | NUMBER | - |
| 5 | lv_order_id | | NUMBER | - |
| 6 | pv_offer_id | | NUMBER | - |
| 7 | pv_master_id | | NUMBER | - |
| **8** | **pv_bill_interval** | | **NUMBER** | **-** |
| | | | | |

Changed Procedure

```
1       PROCEDURE create_billing (pv_offer_id IN NUMBER, pv_master_id IN NUMBER, pv_bill_interval
        IN NUMBER)
2       IS
3       lv_bill_interval    master_tab. bill_interval%TYPE;
4       lv_nopay                    NUMBER;
5       lv_bill_date        DATE;
6       lv_ordr_price       NUMBER;
7       lv_order_id         NUMBER;
8       BEGIN
9       If pv_bill_interval is null then
10      SELECT bill_interval
11      INTO lv_bill_interval
12      FROM master_tab
13      WHERE master_id = pv_master_id;
```

```
14      Else
15        lv_bill_interval:= pv_bill_interval;
16      end if;
17      lv_nopay := 360 / lv_bill_interval;
18      lv_bill_date := SYSDATE;
19
20      SELECT (order_price / lv_nopay)
21      INTO lv_ordr_price
22      FROM order_offer
23      WHERE offer_id = pv_offer_id;
24
25      FOR rec INTO lv_nopay loop
26
27      SELECT order_seq.NEXTVAL
28      INTO lv_order_id
29      FROM DUAL;
30
31      INSERT INTO billing (order_id,order_date,order_amt)
32      VALUES (lv_order_id,lv_bill_date,lv_ordr_price);
33
34      lv_bill_date:=lv_bill_date+bill_interval;
35
36      END LOOP;
37      END;
```

## IV. BENEFITS

- Through the object dependencies we can draw and data flow logic for any PLSQL code.
- In this paper we are converting PLSQL code into ENGLISH statement, which is helpful for the any person to understand the logic even he knows PLSQL or not.
- Some time we are facing issue of code conversion from one environment to other. In this paper we are including neural network to process PLSQL code, which is having a tightly bind logic and we can easily convert this into other programming language like java, c++.
- We have neural network for the PLSQL code that make sense for we can implement learning algorithm for this code.
- To create above learning algorithm, we can write separate logic and just need to validate our neural network of that source code.
- In the current time, mostly database forecasting tool using data for analyze the business and forecasting result. But as we know the above we can create learning algorithm using neural network that means we can create forecasting report based on existing business logic + data analysis. And this will be more accurate.
- We are creating a neural network using the PLSQL code that means we can do the backtracking of logic. This will help to the code rewriting from ENGLISH written rules.
- We have neural network and learning method that means based on current market business, we can suggest where we need to move on.

## V. CONCLUSION

After looking into above structure in detailed and example, we can convert a PLSQL code into Neural network structure and later it into a structured English language. Not only converting a PLSQL code into English language we can show those as a Business Rules which will be easier to understand the whole process without knowing the PLSQL. Later on we can create some interface, by which can convert this PLSQL code into other language like JAVA. In the current time, mostly database forecasting tool using data for analyze the business and forecasting result. But as we know the above we can create learning algorithm using neural network that means we can create forecasting report based on existing business logic + data analysis. And this will be more accurate.

## ACKNOWLEDGMENT

In the above example section, using a small program to show the processes. After using some machine learning algorithm we can make this system more clear and powerful.

## REFERENCES

[1] Kevin Loney : Oracle Database 10g - The complete reference

[2] Eunjee Song,Shuxin Yin,Indrakshi Ray : Using UML to Model Relational Database Operations

[3] Cynthia Krieger (1996). Neural Networks in Data Mining.

[4] Hehnut Schmid (1999): part-of-speech tagging with neural networks.

[5] Herbert Gómez Tobón and Áldrin Fredy Jaramillo Franco Business Rules Extraction from Business Process Specifications Written in Natural Language

[6] S.Rajasekaran , G.A. Vijayalakshmi : Neural Network,Fuzzy Logic,Genetic Algorithms Synthesis & Application.

[7] http://en.wikipedia.org/wiki/Artificial_neural_network