Algorithm 3 GenerateVPSQL(blocks, op, proj)

```
1: query = “”
2: aliaseMap = map<String, String>
3: blockQuery ← ϕ
4: m ← getTripleCount(blocks)
5: if n > 1
6:          buildJoin ← true
7: end if
8: for k = 1 to k = n do
9:     projections, conditions ← ϕ
10:   fromClause ← ϕ
11:   whereClause ← ϕ
12:   for i = 1 to m do
13:       if b == k  // b is the block number
14:       join_conditions ← ϕ
15:           if  isVar(tpi.Subject) then
16:               projections ← projections U (tpi.Subject →  Ti.Subject )
17:               aliaseMap[Ti.Subject] = tpi.Subject
18:           else
19:              conditions ← conditions U (Ti.Subject  = tpi.Subject)
20:           end if
21:           if  isVar(tpi.Object) then
22:              projections ← projections U (tpi.Object → Ti.Object )
23:          aliaseMap[Ti.Object] = tpi.Object
24:           else
25:          conditions ← conditions  U (Ti.Object = tpi.Object)
26:           end if
27:           for j = m to j = 1  do
28:         if b == k
29:             if tpi == 1
30:                  fromClause ←  predicate(i) at tpi assign to aliases Ti
31:                if isPresent(fi) then
32:                    conditions ← conditions U fi
33:                end if
34:            else
35:                  fromClause ← concate_str(fromClause, “JOIN”, predicate(i) at tpi assign to aliases Ti)
36:                if isVar(tpi.Subject)
37:                     if tpi.Subject == tpj.Subject
38:                              join_conditions ← join_conditions U (Ti.Subject == Tj.Subject)
39:                     end if
40:                     if tpi.Subject == tpj.Object
41:                              join_conditions ← join_conditions U (Ti.Subject == Tj.Object)
42:                     end if
43:                  end if
44:                     if isVar(tpi.Object)
45:                     if tpi.Object == tpj.Subject
46:                          join_conditions ← join_conditions U (Ti. Object == Tj.Subject)
47:                      end if
48:                     if tpi.Object == tpj.Object
49:                          join_conditions ← join_conditions U (Ti. Object == Tj.Object)
50:                      end if
51:                      end if
52:                end if
53:           end if
54:           end for
```

```
55:            if isPresent(f_i) then
56:          if isContains(f_i, "!bound")
57:              op[i] = "EXCEPT"
58:          else
59:               whereClause ← conditions U f_i
60:               end if
61:             end if
62:               fromClause ← fromClause U join_conditions
63:       end if
64:    end for
65:    if (buildJoin = true) then
66:        tempQuery ← getTempQuery(projections, fromClause, whereClause)
67:        Add tempQuery to vector blockQuery.
68:    else
69:    selectClause← getProjections(proj, isDistinct)
70:    tempQuery ← getTempQuery(selectClause, fromClause, whereClause)
71:          query = tempQuery // query is the final result if SPARQL Query does not have operator
       OPTIONAL or UNION.
72:    end if
73: end for
74: if n > 1 then
75:     blockJoinCond ←getBlockJoin(proj, aliaseMap, blocks)
76:          blockJoinCond → {cond_1, cond_2,….. cond_n-1}
77:    selectClauseLeftJoin ← getSelectClauseLeftJoin(proj, aliaseMap, blocks)
78:          selectClauseLeftJoin →{X_K. proj_K}  // k ≤ n
79:    selectClauseUnion ← getSelectClauseUnion(proj)
80:      // selectClauseUnion is the string representation of the projected elements present in Proj vector.
81:      for j = 1 to j = n – 1 do
82:           if op[j] == "LEFT OUTER JOIN" || op[j] == "EXCEPT"
83:          if j == 1 then
84:               query += concat_str(selectClauseLeftJoin, blockQuery[j-1], op[j], blockQuery[j], cond[j-1])
85:        else
86:               query += concat_strQuery(op[j], blockQuery[j], cond[j-1])
87:        end if
88:          end if
89:          if op[j] == 'UNION ALL' then
90:         if j ==  1 then
91:              query += concat_str(selectClauseUnion, blockQuery[j-1], op[j], blockQuery[j])
92:         else
93:              query += concat_strQuery(op[j], blockQuery[j], "")
94:         end if
95:           end if
96:       end for
97: end if
98: return query
```