Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Кафедра информатики

Отчёт

Лабораторная работа №4

Выполнил:	Проверил:
студент группы №853504	Чащин С.В.
Кузьма В.В.	

Минск 2021

ЗАДАНИЕ 1-2.

SELECT: на вход подается JSON/XML (на выбор студента), где указан тип запроса (SELECT), наименования выходных столбцов, наименование таблиц, условия объединения таблиц для запроса,условия фильтрации. Необходимо реализовать парс входных данных формирование запроса и выполнение его, на выход отдать курсор.

Вложенные запросы: доработать пункт 1 с тем, чтобы в качестве условия фильтрации можно было бы передать вложенный запрос (условия IN, NOT IN, EXISTS, NOT EXISTS). Сформировать запрос, выполнить его, на выход передать курсор.

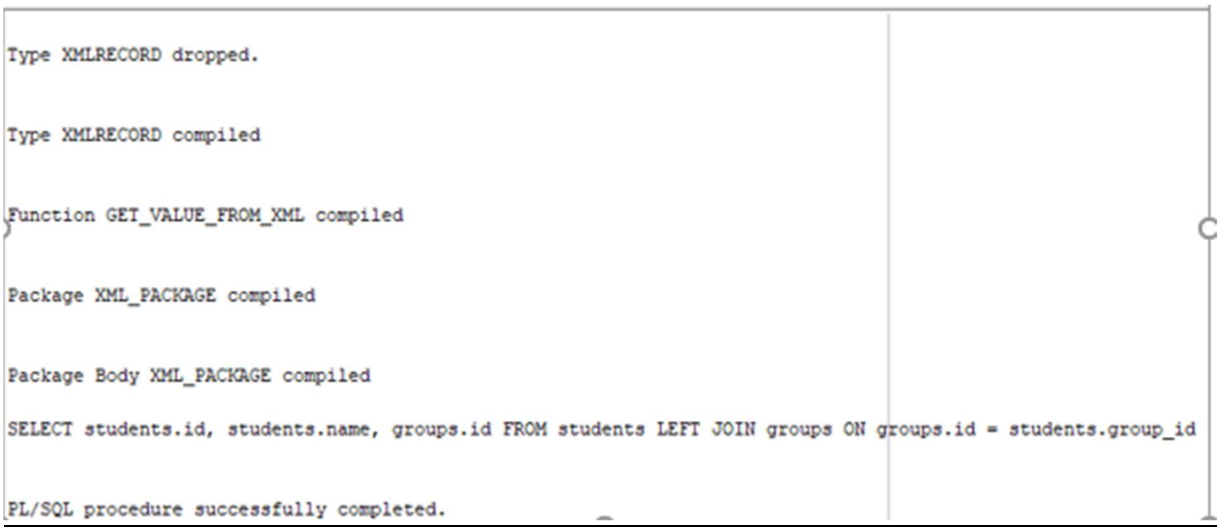
```
DROP TYPE XMLRecord;

CREATE TYPE XMLRecord IS TABLE OF VARCHAR2(1000);

/

CREATE OR REPLACE FUNCTION get_value_from_xml(xml_string IN VARCHAR2, xpath
IN VARCHAR2)

RETURN XMLRecord

AS

   records_length NUMBER :=0;

   current_record VARCHAR2(50) := ' ';

   xml_property XMLRecord := XMLRecord();

   i NUMBER := 1;

BEGIN

  SELECT EXTRACTVALUE(XMLTYPE(xml_string), xpath ||'[' || i || ']') INTO current_record
FROM dual;


  WHILE current_record IS NOT NULL LOOP

      i := i+1;

      records_length := records_length + 1;

      xml_property.extend;

      xml_property(records_length) := REPLACE(TRIM(current_record), ' ', '');

      SELECT EXTRACTVALUE(XMLTYPE(xml_string), xpath ||'[' || i || ']') INTO
current_record  FROM dual;

   END LOOP;

   return xml_property;

end get_value_from_xml;

/
```

```
CREATE OR REPLACE PACKAGE xml_package
AS
  FUNCTION process_select(xml_string IN varchar2) RETURN sys_refcursor;
  FUNCTION xml_select (xml_string in varchar2 ) RETURN varchar2;
  FUNCTION where_property (xml_string in varchar2 ) RETURN varchar2;
END xml_package;
/
CREATE OR REPLACE PACKAGE BODY xml_package
AS
   FUNCTION process_select(xml_string IN varchar2)
   RETURN sys_refcursor
   AS
      cur   sys_refcursor;
   BEGIN
      OPEN cur FOR xml_select(xml_string);
      RETURN cur;
   END process_select;

   FUNCTION xml_select(xml_string in varchar2 )
   RETURN varchar2
   AS
      tables_list XMLRecord := XMLRecord();
      columns_list XMLRecord := XMLRecord();
      filters XMLRecord := XMLRecord();
      join_type VARCHAR2(100);
      join_condition VARCHAR2(100);
      select_query VARCHAR2(1000) :='SELECT';
   BEGIN
      IF xml_string IS NULL THEN
         RETURN NULL;
      END IF;

      tables_list := get_value_from_xml(xml_string, 'Operation/Tables/Table');
      columns_list := get_value_from_xml(xml_string, 'Operation/OutputColumns/Column');
```

```
        select_query := select_query || ' ' || columns_list(1);
        FOR col_index IN 2..columns_list.count LOOP
            select_query := select_query || ', ' ||  columns_list(col_index);
        END LOOP;

        select_query := select_query || ' FROM '  || tables_list(1);
        FOR indx IN 2..tables_list.count LOOP
            SELECT EXTRACTVALUE(XMLTYPE(xml_string),'Operation/Joins/Join'  ||'[' || (indx
- 1) || ']/Type') INTO join_type  FROM dual;
            SELECT EXTRACTVALUE(XMLTYPE(xml_string),'Operation/Joins/Join'  ||'[' || (indx
- 1) || ']/Condition') INTO join_condition FROM dual;
            select_query := select_query || ' ' ||  join_type || ' ' || tables_list(indx) || ' ON ' ||
join_condition;
        END LOOP;

        select_query := select_query || where_property(xml_string);
        dbms_output.put_line(select_query);
        RETURN select_query;
    END xml_select;

    FUNCTION where_property (xml_string in varchar2 ) RETURN varchar2
    AS
        where_filters XMLRecord := XMLRecord();
        where_clouse VARCHAR2(1000) := ' WHERE';
        condition_body VARCHAR2(100);
        sub_query VARCHAR(1000);
        sub_query1 VARCHAR(1000);
        condition_operator VARCHAR(100);
        current_record VARCHAR2(1000);
        records_length NUMBER :=0;
        i NUMBER := 0;
    BEGIN
        SELECT EXTRACT(XMLTYPE(xml_string),
'Operation/Where/Conditions/Condition').getStringVal() INTO current_record  FROM dual;
```

```
        WHILE current_record IS NOT NULL LOOP
            i := i + 1;
            records_length := records_length + 1;
            where_filters.extend;
            where_filters(records_length) := TRIM(current_record);
            SELECT EXTRACT(XMLTYPE(xml_string), 'Operation/Where/Conditions/Condition' 
||'[' || i || ']').getStringVal() INTO current_record  FROM dual;
        END LOOP;


        FOR i IN 2..where_filters.count LOOP
            SELECT EXTRACTVALUE(XMLTYPE(where_filters(i)), 'Condition/Body') INTO 
condition_body  FROM dual;
            SELECT EXTRACT(XMLTYPE(where_filters(i)), 'Condition/Operation').getStringVal() 
INTO sub_query  FROM dual;
            SELECT EXTRACTVALUE(XMLTYPE(where_filters(i)), 
'Condition/ConditionOperator') INTO condition_operator  FROM dual;
            sub_query1 := xml_select(sub_query);


            IF sub_query1 IS NOT NULL THEN
                sub_query1:= '('|| sub_query1 || ')';
            END IF;
            where_clouse := where_clouse || ' ' || TRIM(condition_body) || ' ' || sub_query1 || 
TRIM(condition_operator) || ' ';
        END LOOP;


        IF where_filters.count = 0 THEN
            return ' ';
        ELSE
            return where_clouse;
        END IF;
    END where_property;
END xml_package;
/
SET SERVEROUTPUT ON;
DECLARE
```

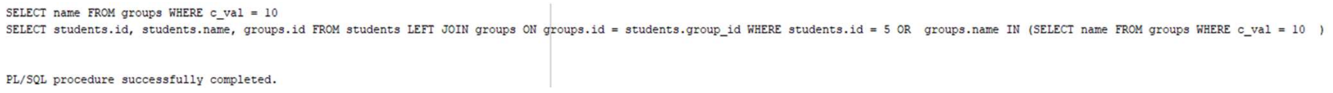
```
    cur   sys_refcursor;
BEGIN
    cur := xml_package.process_select(
    '<Operation>
        <QueryType>
            SELECT
        </QueryType>
        <OutputColumns>
            <Column>students.id</Column>
            <Column>students.name</Column>
            <Column>groups.id</Column>
        </OutputColumns>
        <Tables>
            <Table>students</Table>
            <Table>groups</Table>
        </Tables>
        <Joins>
            <Join>
                <Type>LEFT JOIN</Type>
                <Condition>groups.id = students.group_id</Condition>
            </Join>
        </Joins>
        <Where>
            <Conditions>
                <Body>students.id = 5</Body>
            </Conditions>
        </Where>
    </Operation>');
END;
```



```
Type XMLRECORD dropped.


Type XMLRECORD compiled


Function GET_VALUE_FROM_XML compiled


Package XML_PACKAGE compiled


Package Body XML_PACKAGE compiled

SELECT students.id, students.name, groups.id FROM students LEFT JOIN groups ON groups.id = students.group_id


PL/SQL procedure successfully completed.
```

```
SET SERVEROUTPUT ON;
DECLARE
   cur   sys_refcursor;
BEGIN
   cur := xml_package.process_select(
   '<Operation>
      <QueryType>
         SELECT
      </QueryType>
      <OutputColumns>
         <Column>students.id</Column>
         <Column>students.name</Column>
         <Column>groups.id</Column>
      </OutputColumns>
      <Tables>
         <Table>students</Table>
         <Table>groups</Table>
      </Tables>
      <Joins>
         <Join>
            <Type>LEFT JOIN</Type>
            <Condition>groups.id = students.group_id</Condition>
         </Join>
      </Joins>
```

```
        <Where>
          <Conditions>
            <Condition>
              <Body>students.id = 5</Body>
              <ConditionOperator>OR</ConditionOperator>
            </Condition>
            <Condition>
              <Body>groups.name IN</Body>
              <Operation>
                <QueryType>SELECT</QueryType>
                <OutputColumns>
                  <Column>name</Column>
                </OutputColumns>
                <Tables>
                  <Table>groups</Table>
                </Tables>
                <Where>
                  <Conditions>
                    <Condition>
                      <Body>c_val = 10</Body>
                    </Condition>
                  </Conditions>
                </Where>
              </Operation>
            </Condition>


          </Conditions>
        </Where>
      </Operation>');
```



```
SELECT name FROM groups WHERE c_val = 10
SELECT students.id, students.name, groups.id FROM students LEFT JOIN groups ON groups.id = students.group_id WHERE students.id = 5 OR  groups.name IN (SELECT name FROM groups WHERE c_val = 10  )


PL/SQL procedure successfully completed.
```

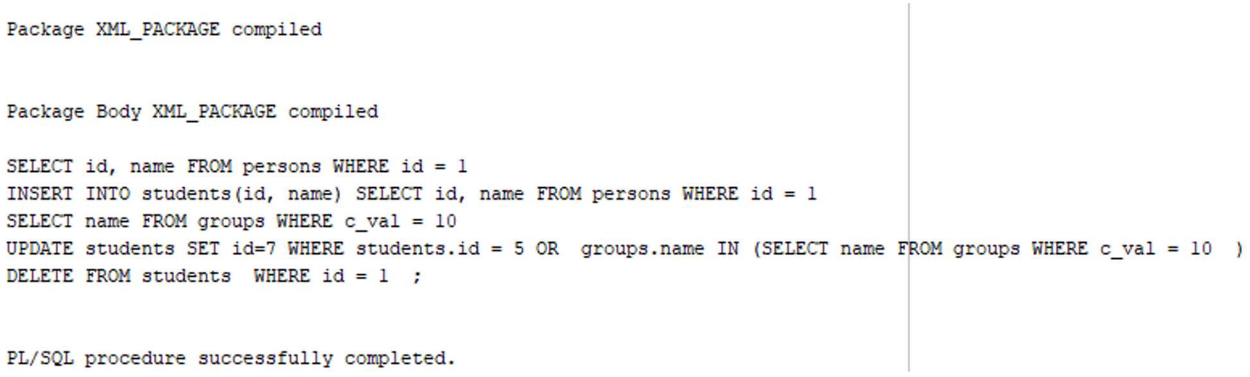
ЗАДАНИЕ 3.

DML: реализовать возможность в качестве структурированного файла передавать условия для генерации и выполнения запросов INSERT, UPDATE, DELETE, с реализацией возможности в качестве фильтра передавать как условия, так и подзапросы (Аналогично блоку 2)

```
CREATE OR REPLACE PACKAGE xml_package
AS
  FUNCTION process_select(xml_string IN varchar2) RETURN sys_refcursor;
  FUNCTION xml_select (xml_string in varchar2 ) RETURN varchar2;
  FUNCTION where_property (xml_string in varchar2 ) RETURN varchar2;
  FUNCTION xml_insert(xml_string in varchar2) RETURN varchar2;
  FUNCTION xml_update(xml_string in varchar2) RETURN varchar2;
  FUNCTION xml_delete(xml_string in varchar2) RETURN varchar2;
END xml_package;
/
CREATE OR REPLACE PACKAGE BODY xml_package
AS
   FUNCTION process_select(xml_string IN varchar2)
   RETURN sys_refcursor
   AS
      cur   sys_refcursor;
   BEGIN
      OPEN cur FOR xml_select(xml_string);
      RETURN cur;
   END process_select;

   FUNCTION xml_select(xml_string in varchar2 )
   RETURN varchar2
   AS
      tables_list XMLRecord := XMLRecord();
      columns_list XMLRecord := XMLRecord();
      filters XMLRecord := XMLRecord();
      join_type VARCHAR2(100);
```

```
    join_condition VARCHAR2(100);
    select_query VARCHAR2(1000) :='SELECT';
  BEGIN
    IF xml_string IS NULL THEN
      RETURN NULL;
    END IF;

    tables_list := get_value_from_xml(xml_string, 'Operation/Tables/Table');
    columns_list := get_value_from_xml(xml_string, 'Operation/OutputColumns/Column');

    select_query := select_query || ' ' || columns_list(1);
    FOR col_index IN 2..columns_list.count LOOP
      select_query := select_query || ', ' ||  columns_list(col_index);
    END LOOP;

    select_query := select_query || ' FROM '  || tables_list(1);
    FOR indx IN 2..tables_list.count LOOP
      SELECT EXTRACTVALUE(XMLTYPE(xml_string),'Operation/Joins/Join'  ||'[' || (indx
- 1) || ']/Type') INTO join_type  FROM dual;
      SELECT EXTRACTVALUE(XMLTYPE(xml_string),'Operation/Joins/Join'  ||'[' || (indx
- 1) || ']/Condition') INTO join_condition FROM dual;
      select_query := select_query || ' ' ||  join_type || ' ' || tables_list(indx) || ' ON ' ||
join_condition;
    END LOOP;

    select_query := select_query || where_property(xml_string);
    dbms_output.put_line(select_query);
    RETURN select_query;
  END xml_select;

  FUNCTION where_property (xml_string in varchar2 ) RETURN varchar2
  AS
    where_filters XMLRecord := XMLRecord();
    where_clouse VARCHAR2(1000) := ' WHERE';
    condition_body VARCHAR2(100);
    sub_query VARCHAR(1000);
    sub_query1 VARCHAR(1000);
    condition_operator VARCHAR(100);
    current_record VARCHAR2(1000);
    records_length NUMBER :=0;
    i NUMBER := 0;
  BEGIN
    SELECT EXTRACT(XMLTYPE(xml_string),
'Operation/Where/Conditions/Condition').getStringVal() INTO current_record  FROM dual;

    WHILE current_record IS NOT NULL LOOP
      i := i + 1;
      records_length := records_length + 1;
      where_filters.extend;
      where_filters(records_length) := TRIM(current_record);
      SELECT EXTRACT(XMLTYPE(xml_string), 'Operation/Where/Conditions/Condition'
||'[' || i || ']').getStringVal() INTO current_record  FROM dual;
    END LOOP;

    FOR i IN 2..where_filters.count LOOP
      SELECT EXTRACTVALUE(XMLTYPE(where_filters(i)), 'Condition/Body') INTO
condition_body  FROM dual;
      SELECT EXTRACT(XMLTYPE(where_filters(i)), 'Condition/Operation').getStringVal()
INTO sub_query  FROM dual;
      SELECT EXTRACTVALUE(XMLTYPE(where_filters(i)),
'Condition/ConditionOperator') INTO condition_operator  FROM dual;
      sub_query1 := xml_select(sub_query);

      IF sub_query1 IS NOT NULL THEN
        sub_query1:= '('|| sub_query1 || ')';
      END IF;
      where_clouse := where_clouse || ' ' || TRIM(condition_body) || ' ' || sub_query1 ||
TRIM(condition_operator) || ' ';
    END LOOP;

    IF where_filters.count = 0 THEN
```

```
        return ' ';
      ELSE
        return where_clouse;
      END IF;
    END where_property;

    FUNCTION xml_insert(xml_string in varchar2)
    RETURN varchar2
    AS
      values_to_insert varchar2(1000);
      select_query_to_insert varchar(1000);
      xml_values XMLRecord := XMLRecord();
      xml_columns_list XMLRecord := XMLRecord();
      insert_query VARCHAR2(1000);
      table_name VARCHAR(100);
      xml_columns VARCHAR2(200);
    BEGIN
      SELECT extract(XMLTYPE(xml_string), 'Operation/Values').getStringVal() INTO
  values_to_insert  FROM dual;
      SELECT EXTRACTVALUE(XMLTYPE(xml_string), 'Operation/Table') INTO
  table_name  FROM dual;

      xml_columns_list := get_value_from_xml(xml_string,'Operation/Columns/Column');
      xml_columns:='(' || xml_columns_list(1);

      FOR i in 2 .. xml_columns_list.count LOOP
        xml_columns := xml_columns || ', ' || xml_columns_list(i);
      END LOOP;

      xml_columns := xml_columns || ')';
      insert_query := 'INSERT INTO ' || table_name ||xml_columns;

      IF values_to_insert IS NOT NULL THEN
        xml_values := get_value_from_xml(values_to_insert,'Values/Value');
        insert_query := insert_query || ' VALUES' || ' (' || xml_values(1) || ')' ;
        FOR i in 2 .. xml_values.count LOOP
          insert_query := insert_query || ', (' || xml_values(i) || ') ';
        END LOOP;
      ELSE
        SELECT EXTRACT(XMLTYPE(xml_string), 'Operation/Operation').getStringVal()
INTO  select_query_to_insert  FROM dual;
        insert_query := insert_query || ' ' || xml_select(select_query_to_insert);
      END IF;
      RETURN insert_query;
    end xml_insert;



    FUNCTION xml_update(xml_string in varchar2)
    RETURN varchar2
    AS
      set_collection XMLRecord := XMLRecord();
      set_operations VARCHAR2(1000);
      update_query VARCHAR2(1000) := 'UPDATE ';
      table_name VARCHAR(100);
    BEGIN
      SELECT extract(XMLTYPE(xml_string), 'Operation/SetOperations').getStringVal() INTO
set_operations FROM dual;
      SELECT EXTRACTVALUE(XMLTYPE(xml_string), 'Operation/Table') INTO
table_name  FROM dual;
      set_collection := get_value_from_xml(set_operations,'SetOperations/Set');
      update_query := update_query || table_name || ' SET ' || set_collection(1);
      FOR i in 2..set_collection.count LOOP
        update_query := update_query || ',' || set_collection(i);
      END LOOP;
      update_query := update_query || where_property(xml_string);
      RETURN update_query;
    END xml_update;


    FUNCTION xml_delete(xml_string in varchar2)
    RETURN varchar2
    AS
        delete_query VARCHAR2(1000) := 'DELETE FROM ';
        table_name VARCHAR(100);
    BEGIN
        SELECT EXTRACTVALUE(XMLTYPE(xml_string), 'Operation/Table') INTO
table_name  FROM dual;
        delete_query := delete_query || table_name || ' ' || where_property(xml_string) || ';';
        RETURN delete_query;
    END xml_delete;
END xml_package;
/
SET SERVEROUTPUT ON;
BEGIN
    DBMS_OUTPUT.put_line(xml_package.xml_insert(
    '<Operation>
        <Type>INSERT</Type>
        <Table>students</Table>
        <Columns>
            <Column>id</Column>
            <Column>name</Column>
        </Columns>
        <Operation>
            <QueryType>SELECT</QueryType>
            <Tables>
                <Table>persons</Table>
            </Tables>
            <OutputColumns>
                <Column>id</Column>
                <Column>name</Column>
            </OutputColumns>
            <Where>
                <Conditions>
                    <Condition>
                    <Body>id = 1</Body>
                    </Condition>
                </Conditions>
            </Where>
        </Operation>
    </Operation>'));

    DBMS_OUTPUT.put_line(xml_package.xml_update(
    '<Operation>
        <Type>UPDATE</Type>
        <Table>students</Table>
        <SetOperations>
            <Set>id = 7</Set>
        </SetOperations>
        <Where>
            <Conditions>
                <Condition>
                    <Body>students.id = 5</Body>
                    <ConditionOperator>OR</ConditionOperator>
                </Condition>
                <Condition>
                    <Body>groups.name IN</Body>
                    <Operation>
                        <QueryType>SELECT</QueryType>
                        <OutputColumns>
                            <Column>name</Column>
                        </OutputColumns>
                        <Tables>
                            <Table>groups</Table>
                        </Tables>
                        <Where>
                            <Conditions>
                                <Condition>
                                    <Body>c_val = 10</Body>
                            </Condition>
                        </Conditions>
                    </Where>
                </Operation>
            </Condition>
        </Conditions>
      </Where>
    </Operation>'));
    DBMS_OUTPUT.put_line(xml_package.xml_delete(
    '<Operation>
      <Type>DELETE</Type>
      <Table>students</Table>
      <Where>
        <Conditions>
          <Condition>
            <Body>id = 1</Body>
          </Condition>
        </Conditions>
      </Where>
    </Operation>'));
```



```
Package XML_PACKAGE compiled


Package Body XML_PACKAGE compiled

SELECT id, name FROM persons WHERE id = 1
INSERT INTO students(id, name) SELECT id, name FROM persons WHERE id = 1
SELECT name FROM groups WHERE c_val = 10
UPDATE students SET id=7 WHERE students.id = 5 OR  groups.name IN (SELECT name FROM groups WHERE c_val = 10  )
DELETE FROM students  WHERE id = 1  ;


PL/SQL procedure successfully completed.
```

ЗАДАНИЕ 4-5.

DDL: реализовать возможность генерации и выполнения DDL скриптов CREATE TABLE и DROP TABLE. В качестве входных данных - структурированный файл с определением DDL-команды, названием таблицы, в случае необходимости (перечнем полей и их типов).

Доработать пункт 4 с тем, чтобы одновременно с созданием таблицы генерировался триггер по генерации значения первичного ключа.

```
CREATE OR REPLACE FUNCTION auto_increment_generator(table_name in varchar2)
RETURN varchar2
AS
   generated_script VARCHAR(1000);
BEGIN
   generated_script := 'CREATE SEQUENCE ' || table_name || '_pk_seq' || ';';
   generated_script :=  generated_script ||  'CREATE OR REPLACE TRIGGER ' || table_name || ' BEFORE INSERT ON ' || table_name || ' FOR EACH ROW '|| chr(10) ||
         'BEGIN  ' || chr(10) ||
         '   IF inserting THEN ' || chr(10) ||
         '       IF :NEW.ID IS NULL THEN ' || chr(10) ||
         '           SELECT ' || table_name || '_pk_seq'  || '.nextval INTO :NEW.ID FROM dual; '|| chr(10) ||
         '       END IF; '|| chr(10) ||
         '   END IF; '|| chr(10) ||
         'END;';
  RETURN generated_script;
END auto_increment_generator;


/
CREATE OR REPLACE PACKAGE xml_package
AS
  FUNCTION process_select(xml_string IN varchar2) RETURN sys_refcursor;
  FUNCTION xml_select (xml_string in varchar2 ) RETURN varchar2;
  FUNCTION where_property (xml_string in varchar2 ) RETURN varchar2;
  FUNCTION xml_insert(xml_string in varchar2) RETURN varchar2;
  FUNCTION xml_update(xml_string in varchar2) RETURN varchar2;
  FUNCTION xml_delete(xml_string in varchar2) RETURN varchar2;
  FUNCTION xml_drop(xml_string IN VARCHAR2) RETURN varchar2;
  FUNCTION xml_create(xml_string IN VARCHAR2) RETURN nvarchar2;
END xml_package;
/
```

```
CREATE OR REPLACE PACKAGE BODY xml_package
AS
  FUNCTION process_select(xml_string IN varchar2)
  RETURN sys_refcursor
  AS
    cur   sys_refcursor;
  BEGIN
    OPEN cur FOR xml_select(xml_string);
    RETURN cur;
  END process_select;

  FUNCTION xml_select(xml_string in varchar2 )
  RETURN varchar2
  AS
    tables_list XMLRecord := XMLRecord();
    columns_list XMLRecord := XMLRecord();
    filters XMLRecord := XMLRecord();
    join_type VARCHAR2(100);
    join_condition VARCHAR2(100);
    select_query VARCHAR2(1000) :='SELECT';
  BEGIN
    IF xml_string IS NULL THEN
      RETURN NULL;
    END IF;

    tables_list := get_value_from_xml(xml_string, 'Operation/Tables/Table');
    columns_list := get_value_from_xml(xml_string, 'Operation/OutputColumns/Column');

    select_query := select_query || ' ' || columns_list(1);
    FOR col_index IN 2..columns_list.count LOOP
      select_query := select_query || ', ' ||  columns_list(col_index);
    END LOOP;

    select_query := select_query || ' FROM '  || tables_list(1);
```

```
        FOR indx IN 2..tables_list.count LOOP
           SELECT EXTRACTVALUE(XMLTYPE(xml_string),'Operation/Joins/Join'  ||'[' || (indx
- 1) || ']/Type') INTO join_type  FROM dual;
           SELECT EXTRACTVALUE(XMLTYPE(xml_string),'Operation/Joins/Join'  ||'[' || (indx
- 1) || ']/Condition') INTO join_condition FROM dual;
           select_query := select_query || ' ' ||  join_type || ' ' || tables_list(indx) || ' ON ' ||
join_condition;
        END LOOP;


        select_query := select_query || where_property(xml_string);
        dbms_output.put_line(select_query);
        RETURN select_query;
    END xml_select;


    FUNCTION where_property (xml_string in varchar2 ) RETURN varchar2
    AS
        where_filters XMLRecord := XMLRecord();
        where_clouse VARCHAR2(1000) := ' WHERE';
        condition_body VARCHAR2(100);
        sub_query VARCHAR(1000);
        sub_query1 VARCHAR(1000);
        condition_operator VARCHAR(100);
        current_record VARCHAR2(1000);
        records_length NUMBER :=0;
        i NUMBER := 0;
    BEGIN
        SELECT EXTRACT(XMLTYPE(xml_string),
'Operation/Where/Conditions/Condition').getStringVal() INTO current_record  FROM dual;


        WHILE current_record IS NOT NULL LOOP
           i := i + 1;
           records_length := records_length + 1;
           where_filters.extend;
           where_filters(records_length) := TRIM(current_record);
           SELECT EXTRACT(XMLTYPE(xml_string), 'Operation/Where/Conditions/Condition'
||'[' || i || ']').getStringVal() INTO current_record  FROM dual;
```

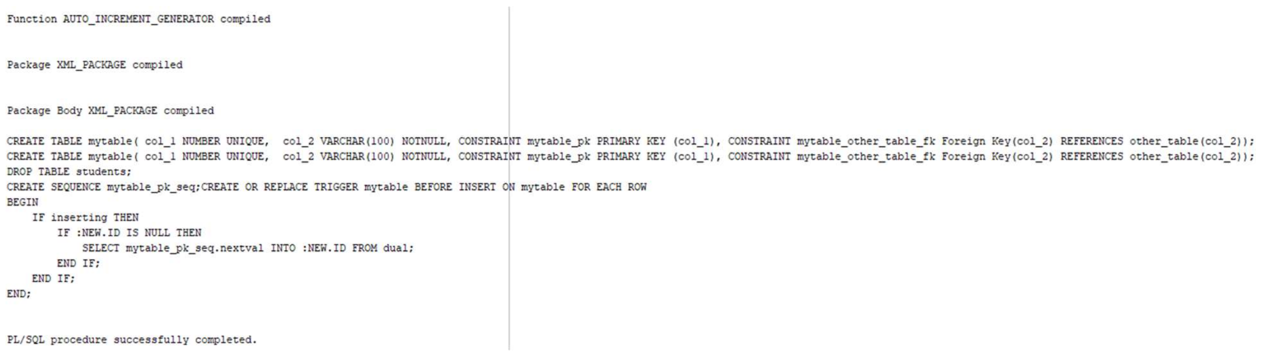
```
        END LOOP;


        FOR i IN 2..where_filters.count LOOP
            SELECT EXTRACTVALUE(XMLTYPE(where_filters(i)), 'Condition/Body') INTO
condition_body  FROM dual;
            SELECT EXTRACT(XMLTYPE(where_filters(i)), 'Condition/Operation').getStringVal()
INTO sub_query  FROM dual;
            SELECT EXTRACTVALUE(XMLTYPE(where_filters(i)),
'Condition/ConditionOperator') INTO condition_operator  FROM dual;
            sub_query1 := xml_select(sub_query);


            IF sub_query1 IS NOT NULL THEN
                sub_query1:= '('|| sub_query1 || ')';
            END IF;
            where_clouse := where_clouse || ' ' || TRIM(condition_body) || ' ' || sub_query1 ||
TRIM(condition_operator) || ' ';
        END LOOP;


        IF where_filters.count = 0 THEN
            return ' ';
        ELSE
            return where_clouse;
        END IF;
    END where_property;


    FUNCTION xml_insert(xml_string in varchar2)
    RETURN varchar2
    AS
        values_to_insert varchar2(1000);
        select_query_to_insert varchar(1000);
        xml_values XMLRecord := XMLRecord();
        xml_columns_list XMLRecord := XMLRecord();
        insert_query VARCHAR2(1000);
        table_name VARCHAR(100);
        xml_columns VARCHAR2(200);
```

```
  BEGIN
      SELECT extract(XMLTYPE(xml_string), 'Operation/Values').getStringVal() INTO
values_to_insert  FROM dual;
      SELECT EXTRACTVALUE(XMLTYPE(xml_string), 'Operation/Table') INTO
table_name  FROM dual;


      xml_columns_list := get_value_from_xml(xml_string,'Operation/Columns/Column');
      xml_columns:='(' || xml_columns_list(1);


      FOR i in 2 .. xml_columns_list.count LOOP
        xml_columns := xml_columns || ', ' || xml_columns_list(i);
      END LOOP;


      xml_columns := xml_columns || ')';
      insert_query := 'INSERT INTO ' || table_name ||xml_columns;


      IF values_to_insert IS NOT NULL THEN
        xml_values := get_value_from_xml(values_to_insert,'Values/Value');
        insert_query := insert_query || ' VALUES' || ' (' || xml_values(1) || ')' ;
        FOR i in 2 .. xml_values.count LOOP
          insert_query := insert_query || ', (' || xml_values(i) || ') ';
        END LOOP;
      ELSE
         SELECT EXTRACT(XMLTYPE(xml_string), 'Operation/Operation').getStringVal()
INTO  select_query_to_insert  FROM dual;
         insert_query := insert_query || ' ' || xml_select(select_query_to_insert);
      END IF;
      RETURN insert_query;
  end xml_insert;



  FUNCTION xml_update(xml_string in varchar2)
  RETURN varchar2
  AS
      set_collection XMLRecord := XMLRecord();
```

```
        set_operations VARCHAR2(1000);
        update_query VARCHAR2(1000) := 'UPDATE ';
        table_name VARCHAR(100);
    BEGIN
        SELECT extract(XMLTYPE(xml_string), 'Operation/SetOperations').getStringVal() INTO
set_operations FROM dual;
        SELECT EXTRACTVALUE(XMLTYPE(xml_string), 'Operation/Table') INTO
table_name  FROM dual;
        set_collection := get_value_from_xml(set_operations,'SetOperations/Set');
        update_query := update_query || table_name || ' SET ' || set_collection(1);
        FOR i in 2..set_collection.count LOOP
            update_query := update_query || ',' || set_collection(i);
        END LOOP;
        update_query := update_query || where_property(xml_string);
        RETURN update_query;
    END xml_update;

    FUNCTION xml_delete(xml_string in varchar2)
    RETURN varchar2
    AS
        delete_query VARCHAR2(1000) := 'DELETE FROM ';
        table_name VARCHAR(100);
    BEGIN
        SELECT EXTRACTVALUE(XMLTYPE(xml_string), 'Operation/Table') INTO
table_name  FROM dual;
        delete_query := delete_query || table_name || ' ' || where_property(xml_string) || ';';
        RETURN delete_query;
    END xml_delete;

    FUNCTION xml_drop(xml_string IN VARCHAR2)
    RETURN varchar2
    AS
        drop_query VARCHAR2(1000):='DROP TABLE ';
        table_name VARCHAR2(100);
    BEGIN
```

```
        SELECT EXTRACTVALUE(XMLTYPE(xml_string), 'Operation/Table') INTO
table_name  FROM dual;

        drop_query := drop_query || table_name || ';';

        RETURN drop_query;

    END xml_drop;


    FUNCTION xml_create(xml_string IN VARCHAR2)

    RETURN nvarchar2

    AS

        col_name VARCHAR2(100);

        col_type VARCHAR(100);

        parent_table VARCHAR2(100);

        constraint_value VARCHAR2(100);

        temporal_record XMLRecord := XMLRecord();

        temporal_string VARCHAR2(100);

        create_query VARCHAR2(1000):= 'CREATE TABLE';

        primary_constraint VARCHAR2(1000);

        auto_increment_script VARCHAR(1000);

        current_record VARCHAR2(1000);

        records_length NUMBER :=0;

        table_columns XMLRecord := XMLRecord();

        table_name VARCHAR2(100);

        col_constraints XMLRecord := XMLRecord();

        table_constraints XMLRecord := XMLRecord();

        i NUMBER := 0;

    BEGIN

        SELECT EXTRACTVALUE(XMLTYPE(xml_string), 'Operation/Table') INTO
table_name  FROM dual;

        create_query := create_query || ' ' || table_name || '(';


        SELECT EXTRACT(XMLTYPE(xml_string),
'Operation/Columns/Column').getStringVal() INTO current_record  FROM dual;

        WHILE current_record IS NOT NULL LOOP

            i := i + 1;

            records_length := records_length + 1;
            table_columns.extend;
            table_columns(records_length) := TRIM(current_record);
            SELECT EXTRACT(XMLTYPE(xml_string), 'Operation/Columns/Column' ||'[' || i ||
']').getStringVal()
            INTO current_record
            FROM dual;
        END LOOP;


        FOR i in 2..table_columns.count LOOP
            constraint_value := '';
            SELECT EXTRACTVALUE(XMLTYPE(table_columns(i)), 'Column/Name') INTO
col_name  FROM dual;
            SELECT EXTRACTVALUE(XMLTYPE(table_columns(i)), 'Column/Type') INTO
col_type  FROM dual;
            col_constraints :=
get_value_from_xml(table_columns(i),'Column/Constraints/Constraint');
            FOR i in 1..col_constraints.count LOOP
                constraint_value := constraint_value || ' ' || col_constraints(i);
            END LOOP;
            create_query := create_query || ' ' || col_name || ' ' || col_type || constraint_value;


            IF i != table_columns.count THEN
                create_query := create_query || ', ';
            END IF;
        END LOOP;


        SELECT extract(XMLTYPE(xml_string),
'Operation/TableConstraints/PrimaryKey').getStringVal()
        INTO  primary_constraint
        FROM dual;


        IF primary_constraint IS NOT NULL THEN
            temporal_record :=
get_value_from_xml(primary_constraint,'PrimaryKey/Columns/Column');
            temporal_string := temporal_record(1);
            FOR i in 2..temporal_record.count LOOP
                temporal_string := temporal_string || ', ' || temporal_record(i);
            END LOOP;
            create_query := create_query || ', CONSTRAINT ' || table_name || '_pk '|| 'PRIMARY
KEY (' || temporal_string || ')';
        ELSE
            auto_increment_script := auto_increment_generator(table_name);
            create_query := create_query || ', ID NUMBER PRIMARY KEY';
        END IF;

        table_constraints := XMLRecord();
        records_length := 0;
        i := 0;
        SELECT EXTRACT(XMLTYPE(xml_string),
'Operation/TableConstraints/ForeignKey').getStringVal() INTO current_record  FROM dual;
        WHILE current_record IS NOT NULL LOOP
            i := i + 1;
            records_length := records_length + 1;
            table_constraints.extend;
            table_constraints(records_length) := TRIM(current_record);
            SELECT EXTRACT(XMLTYPE(xml_string), 'Operation/TableConstraints/ForeignKey'
||'[' || i || ']').getStringVal()
            INTO current_record
            FROM dual;
        END LOOP;

        FOR i in 2..table_constraints.count LOOP
            SELECT EXTRACTVALUE(XMLTYPE(table_constraints(i)), 'ForeignKey/Parent')
INTO parent_table  FROM dual;
            temporal_record :=
get_value_from_xml(table_constraints(i),'ForeignKey/ChildColumns/Column');
            temporal_string := temporal_record(1);
            FOR i in 2..temporal_record.count LOOP
                temporal_string := temporal_string || ', ' || temporal_record(i);
            END LOOP;
            create_query:= create_query || ', CONSTRAINT ' || table_name || '_' || parent_table || '_fk '
||
                                        'Foreign Key' || '(' || temporal_string || ') ';
            temporal_record := get_value_from_xml(table_constraints(i),
'ForeignKey/ChildColumns/Column');
            temporal_string := temporal_record(1);
            FOR i in 2..temporal_record.count LOOP
               temporal_string := temporal_string || ', ' || temporal_record(i);
            END LOOP;
            create_query:= create_query || 'REFERENCES ' || parent_table || '(' || temporal_string || ')';

        END LOOP;
        create_query := create_query || ');' || auto_increment_script;
        DBMS_OUTPUT.put_line(create_query);
        return create_query;
   END xml_create;
END xml_package;
/
SET SERVEROUTPUT ON;
DECLARE
   generated_script VARCHAR(1000);
BEGIN
   DBMS_OUTPUT.put_line(xml_package.xml_create(
   '<Operation>
      <Type>CREATE</Type>
      <Table>mytable</Table>
      <Columns>
         <Column>
            <Name>col_1</Name>
            <Type>NUMBER</Type>
            <Constraints>
               <Constraint>UNIQUE</Constraint>
            </Constraints>
         </Column>
         <Column>
           <Name>col_2</Name>
```

```
          <Type>VARCHAR(100)</Type>
           <Constraints>
              <Constraint>NOT NULL</Constraint>
          </Constraints>
        </Column>
    </Columns>
    <TableConstraints>
        <PrimaryKey>
            <Columns>
                <Column>col_1</Column>
            </Columns>
        </PrimaryKey>
        <ForeignKey>
            <ChildColumns>
                <Column>col_2</Column>
            </ChildColumns>
            <Parent>other_table</Parent>
            <ParentColumns>
                <Column>id</Column>
            </ParentColumns>
        </ForeignKey>
    </TableConstraints>
  </Operation>'
  ));
  DBMS_OUTPUT.put_line(xml_package.xml_drop(
  '<Operation>
    <Type>DROP</Type>
    <Table>students</Table>
  </Operation>'));
  generated_script := auto_increment_generator('mytable') ;
  DBMS_OUTPUT.put_line(generated_script);
END;
```



```
Function AUTO_INCREMENT_GENERATOR compiled


Package XML_PACKAGE compiled


Package Body XML_PACKAGE compiled

CREATE TABLE mytable( col_1 NUMBER UNIQUE,  col_2 VARCHAR(100) NOTNULL, CONSTRAINT mytable_pk PRIMARY KEY (col_1), CONSTRAINT mytable_other_table_fk Foreign Key(col_2) REFERENCES other_table(col_2));
CREATE TABLE mytable( col_1 NUMBER UNIQUE,  col_2 VARCHAR(100) NOTNULL, CONSTRAINT mytable_pk PRIMARY KEY (col_1), CONSTRAINT mytable_other_table_fk Foreign Key(col_2) REFERENCES other_table(col_2));
DROP TABLE students;
CREATE SEQUENCE mytable_pk_seq;CREATE OR REPLACE TRIGGER mytable BEFORE INSERT ON mytable FOR EACH ROW
BEGIN
    IF inserting THEN
        IF :NEW.ID IS NULL THEN
            SELECT mytable_pk_seq.nextval INTO :NEW.ID FROM dual;
        END IF;
    END IF;
END;


PL/SQL procedure successfully completed.
```