Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Отчет к лабораторной работе №4

Выполнил:

студент гр. 953501

Кондрашов И.Д.

Проверил:

Чащин C. В.

Минск 2022

# 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>

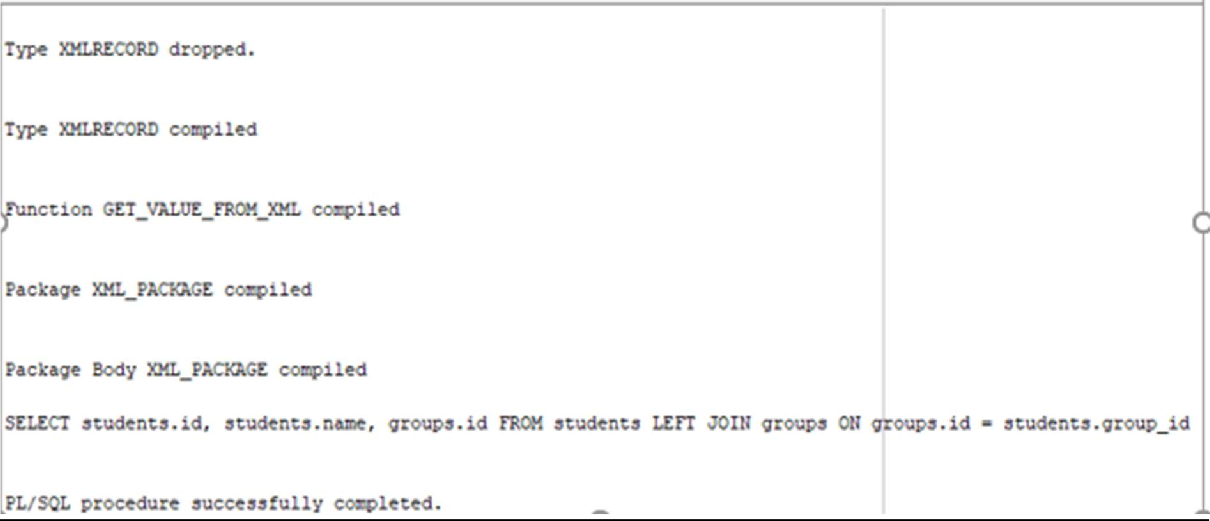
<Conditions>

<Body>students.id = 5</Body>

</Conditions>

</Where>

</Operation>'); END;



### 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>

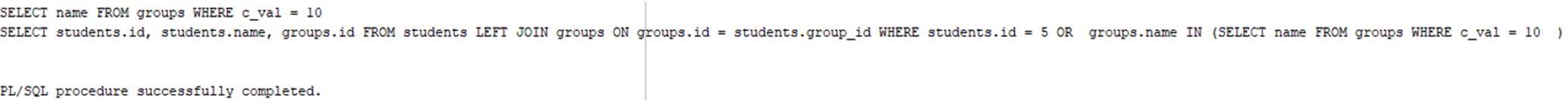
</Operation>

</Condition>

</Conditions>

</Where>

</Operation>');



# 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>

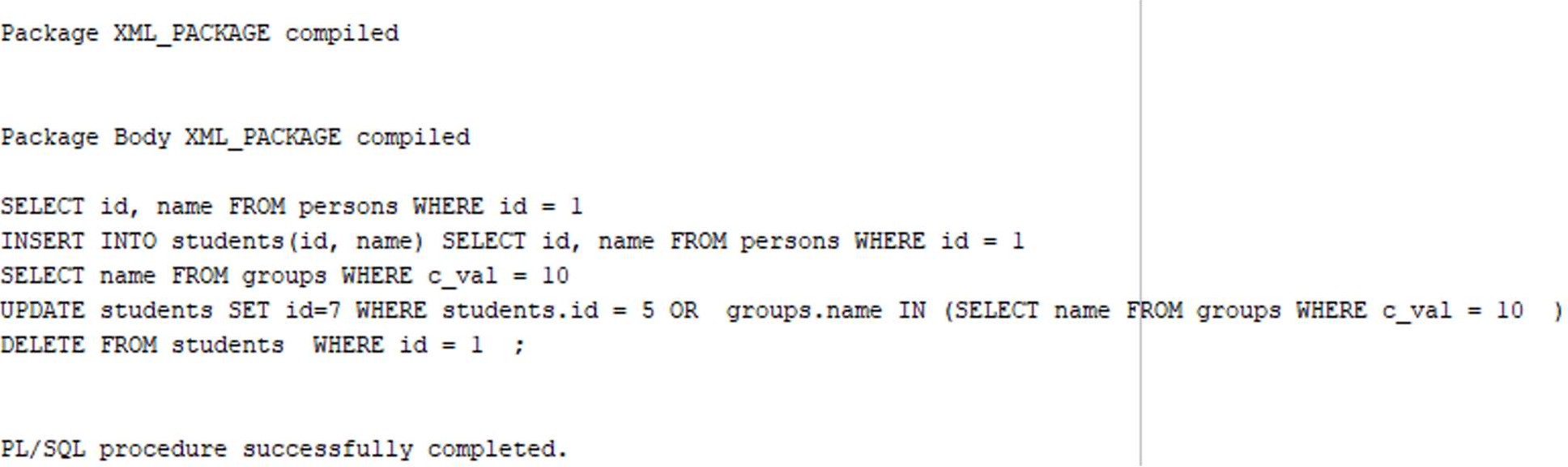
<Body>id = 1</Body>

</Condition>

</Conditions>

</Where>

</Operation>'));



# 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;

