Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Отчет по лабораторной работе №4

по дисциплине «Модели данных и системы управления базами данных»

Студент: гр. 953501

Кореневский С. А.

Проверил: Чащин С.В.

Минск 2022

Задание 1

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.

Задание 2

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.

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 TABLE json_tabl (
  id Number GENERATED ALWAYS AS IDENTITY(Start WITH 1 INCREMENT by 1),
  data  CLOB,
  CONSTRAINT json_documents_pk PRIMARY KEY (id),
  CONSTRAINT json_documents_json_chk CHECK (data IS JSON)
);
```

```
declare
 res VARCHAR2(600);
FUNCTION make_query(data_js IN VARCHAR2)
RETURN VARCHAR2
IS
   query VARCHAR2(600);
   query_type VARCHAR2(600);
   table_name VARCHAR2(600);
   counter NUMBER := 0;
   filters VARCHAR2(600);
   nested_filter VARCHAR2(600);
   set_t VARCHAR2(600);

BEGIN
   INSERT INTO json_tabl (data) VALUES (data_js);
```

```
   SELECT a.data.operation, a.data.tablen INTO query_type, table_name FROM json_tabl a where id = (SELECT
MAX(ID) FROM json_tabl) ;

   CASE query_type
      WHEN 'select' THEN
         query  := 'SELECT ';
         counter := 0;
         FOR res IN (SELECT value FROM json_table((SELECT a.data.fields FROM json_tabl a where
ID=(SELECT MAX(ID) FROM json_tabl)) , '$[*]' COLUMNS (value PATH '$')))
         LOOP
            IF counter != 0 THEN
               query := query || ', ' || res.value ;
            ELSE
               query := query || res.value;
               counter := counter + 1;
            END IF;
         END LOOP;

         query := query || ' FROM ' || table_name;

         counter := 1;


         FOR res IN (SELECT tablen,onn FROM json_table( (SELECT a.data.join FROM json_tabl a where
ID=(SELECT MAX(ID) FROM json_tabl) ) , '$[*]' COLUMNS (tablen PATH '$.tablen', onn PATH '$.onn')))
         LOOP
            query := query || ' JOIN ' || res.tablen;
            query := query || ' ON ' || res.onn;
         END LOOP;

         SELECT a.data.filters INTO filters FROM json_tabl a where ID=(SELECT MAX(ID) FROM json_tabl) ;

         IF filters is not NULL THEN
            query := query || ' where ';

            FOR res IN (SELECT type, condition,log_oper,query FROM json_table( (SELECT a.data.filters FROM
json_tabl a where ID=(SELECT MAX(ID) FROM json_tabl) )  , '$[*]'
               COLUMNS (type PATH '$.type',
                     condition PATH '$.condition',
                     log_oper PATH '$.log_oper',
                     fquery PATH '$.query'
                     )))
            LOOP
               query := query || res.condition;

               IF res.type = 'nested' THEN
```

```
                    Select a.data.filters.query INTO nested_filter from json_tabl a where ID=(SELECT MAX(ID) FROM
json_tabl);
                    query:= query || ' (' || make_query(nested_filter) || ') ' ;
                END IF;

                if res.log_oper is not NULL Then
                    query:= query || ' ' || res.log_oper || ' ' ;
                END IF;
            END LOOP;
        END IF;

    WHEN 'insert' THEN
        query:= 'INSERT INTO ' || table_name || ' (';
        counter := 0;
        FOR res IN (SELECT value FROM json_table( (Select a.data.fields from json_tabl a where ID=(SELECT
MAX(ID) FROM json_tabl)) , '$[*]' COLUMNS (value PATH '$')))
        LOOP
            IF counter != 0 then
                query := query || ', ' || res.value ;
            ELSE
                query := query || res.value;
                counter := counter + 1;
            END IF;
        END LOOP;

        query := query || ') VALUES(';

        counter := 0;
        FOR res IN (SELECT value FROM json_table( (Select a.data.valuess from json_tabl a where ID=(SELECT
MAX(ID) FROM json_tabl)) , '$[*]' COLUMNS (value PATH '$')))
        LOOP
            IF counter != 0 then
                query := query || ', ' || res.value ;
            ELSE
                query := query || res.value;
                counter := counter + 1;
            END IF;
        END LOOP;

        query := query || ');';

    WHEN 'update' THEN
    query := 'UPDATE ' || table_name || ' SET ';

    counter := 0;
    FOR res IN (SELECT value FROM json_table( (Select a.data.sett from json_tabl a where ID=(SELECT
```

```
MAX(ID) FROM json_tabl)) , '$[*]' COLUMNS (value PATH '$')))
        LOOP
          IF counter != 0 then
            query := query || ', ' || res.value ;
          ELSE
            query := query || res.value;
            counter := counter + 1;
          END IF;
        END LOOP;

      SELECT a.data.filters INTO filters FROM json_tabl a where ID=(SELECT MAX(ID) FROM json_tabl) ;

        IF filters is not NULL THEN
          query := query || ' where ';

          FOR res IN (SELECT type, condition,log_oper,query FROM json_table( (SELECT a.data.filters FROM
json_tabl a where ID=(SELECT MAX(ID) FROM json_tabl) )  , '$[*]'
            COLUMNS (type PATH '$.type',
                condition PATH '$.condition',
                log_oper PATH '$.log_oper',
                fquery PATH '$.query'
                )))
          LOOP
            query := query || res.condition;

            IF res.type = 'nested' THEN
              Select a.data.filters.query INTO nested_filter from json_tabl a where ID=(SELECT MAX(ID) FROM
json_tabl);
              query:= query || ' (' || make_query(nested_filter) || ') ' ;
            END IF;

            if res.log_oper is not NULL Then
              query:= query || ' ' || res.log_oper || ' ' ;
            END IF;
          END LOOP;
        END IF;
      WHEN 'delete' THEN

        query:= 'DELETE FROM ' || table_name ;
        SELECT a.data.filters INTO filters FROM json_tabl a where ID=(SELECT MAX(ID) FROM json_tabl) ;

        IF filters is not NULL THEN
          query := query || ' where ';

          FOR res IN (SELECT type, condition,log_oper,query FROM json_table( (SELECT a.data.filters FROM
json_tabl a where ID=(SELECT MAX(ID) FROM json_tabl) )  , '$[*]'
```

```
                COLUMNS (type PATH '$.type',
                        condition PATH '$.condition',
                        log_oper PATH '$.log_oper',
                        fquery PATH '$.query'
                        )))
              LOOP
                query := query || res.condition;

                IF res.type = 'nested' THEN
                  Select a.data.filters.query INTO nested_filter from json_tabl a where ID=(SELECT MAX(ID) FROM
json_tabl);
                  query:= query || ' (' || make_query(nested_filter) || ') ' ;
                END IF;

                if res.log_oper is not NULL Then
                  query:= query || ' ' || res.log_oper || ' ' ;
                END IF;
              END LOOP;
            END IF;

        ELSE
          query:= 'error';

    END case;
    DELETE FROM json_tabl where id=(SELECT MAX(ID) FROM json_tabl);
    return query;

END make_query;
```

Результат

JSON:

```
{
        "operation": "delete",
        "tablen": "university",
        "filters": [
            {
                "type": "primitive",
                "condition": "id > 3",
                "log_oper": "and"
            },
            {
                "type": "nested",
                "condition": "name in",
                "query": {
                    "operation": "select",
                    "fields": ["name"],
                    "tablen": "university",
                    "filters": [
                        {
                            "type": "primitive",
                            "condition": "lower(name) like ''%json%''"
                        }
                    ]
                }
            }
        ]
    }
```

SQL:

```
[2022-05-18 13:42:08] completed in 15 ms
DELETE FROM university where id > 3 and name in (SELECT name FROM university where lower(name) like '%json%')
```

JSON:

```
{
      "operation": "select",
      "fields":["name"],
      "tablen": "student",
    "filters": [
        {
            "type": "primitive",
            "condition": "id > 2",
            "log_oper": "and"
        },
        {
            "type": "nested",
            "condition": "name in",
            "query": {
                "operation": "select",
                "fields": ["name"],
                "table": "student",
                "filters": [
                    {
                        "type": "primitive",
                        "condition": "id % 2 = 0"
                    }
                ]
            }
        }
    ]
}
```

SQL:

```
[2022-05-18 13:44:43] completed in 33 ms
SELECT name FROM student where id > 2 and name in (SELECT name FROM  where id % 2 = 0)
```

JSON:

```
{
      "operation": "select",
      "fields":["student.name", "student_group.name as group_name",
"university.name as uni_name"],
      "tablen": "student",
      "join": [
            {
                  "tablen": "student_group",
                  "onn": "student.group_id = student_group.id"
            },
            {
                  "tablen": "university",
                  "onn": "student_group.university_id = university.id"
            }
      ],
      "filters": [
            {
                  "type": "primitive",
                  "condition": "student.id % 2 = 1"
            }
      ]
 }
```

SQL:

```
[2022-05-18 13:45:51] completed in 51 ms
SELECT student.name, student_group.name as group_name, university.name as uni_name FROM student JOIN student_group ON student
  .group_id = student_group.id JOIN university ON student_group.university_id = university.id where student.id % 2 = 1
```

JSON:

```
{
      "operation": "select",
      "tablen": "student_group",
      "fields":["id", "name"],
      "filters": [
            {
                  "type": "primitive",
                  "condition": "id % 2 = 0",
            }
      ]
 }
```

SQL:

```
[2022-05-18 13:50:17] completed in 31 ms
SELECT id, name FROM student_group where id % 2 = 0
```

JSON:

```
{
        "operation": "insert",
        "table": "student",
        "fields": ["name", "group_id"],
        "valuess": ["JSON student", 3]
}
```

SQL:

```
[2022-05-18 13:52:41] completed in 34 ms
INSERT INTO  (name, group_id) VALUES(JSON student, 3);
```

JSON:

```
{
    "operation": "update",
    "tablen": "university",
    "sett": ["name = concat(asd, name)"],
    "filters": [
        {
            "type": "primitive",
            "condition": "id > 3",
            "log_oper": "and"
        },
        {
            "type": "nested",
            "condition": "name in",
            "query": {
                "operation": "select",
                "fields": ["name"],
                "tablen": "university",
                "filters": [
                    {
                        "type": "primitive",
                        "condition": "lower(name) like 'json%'"
                    }
                ]
            }
        }
    ]
}
```

SQL:

```
[2022-05-18 13:54:19] completed in 35 ms
UPDATE university SET name = concat(asd, name) where id > 3 and name in (SELECT name FROM university where lower(name) like
 'json%')
```

Задание 4

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).

Задание 5

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

```
declare
 res VARCHAR2(32000);

FUNCTION make_trigger(pk_name VARCHAR2, table_name VARCHAR2)
RETURN VARCHAR2
IS
   query VARCHAR2(32000);
BEGIN
   query :='CREATE OR REPLACE TRIGGER pk_trigger
          BEFORE INSERT
          ON ' || table_name || '
          FOR EACH ROW
          DECLARE
            counter NUMBER;
            max_id NUMBER;
          BEGIN
            IF :new.'|| pk_name ||' IS NOT NULL THEN
               SELECT COUNT(*) INTO counter FROM '||table_name||' WHERE id=:new.'|| pk_name ||';
               IF counter > 0 THEN
                  raise_application_error(-20001,''Invalid '||pk_name||''');
               END IF;
            ELSE
               SELECT max('||pk_name||') INTO max_id FROM '||table_name||';
               IF max_id is NULL THEN
                  :new.'||pk_name||':=1;
               ELSE
                  :new.'||pk_name||':= max_id + 1;
               END IF;
            END IF;
          END;';
   return query;
END make_trigger;
```

```
FUNCTION make_query(data_js IN VARCHAR2)
RETURN VARCHAR2
IS
   query VARCHAR2(32000);
   query_type VARCHAR2(600);
   table_name VARCHAR2(600);
   pk_name VARCHAR2(600);

BEGIN
   INSERT INTO json_tabl (data) VALUES (data_js);

   SELECT a.data.operation, a.data.tablen INTO query_type, table_name FROM json_tabl a where id = (SELECT
MAX(ID) FROM json_tabl) ;
   CASE query_type
      WHEN 'create' THEN
         query:= 'CREATE TABLE ' || table_name || ' ('|| chr(10);
         FOR res IN (SELECT name, type, f_k, is_pk FROM json_table( (SELECT a.data.fields FROM json_tabl a
where ID=(SELECT MAX(ID) FROM json_tabl) )  , '$[*]'
               COLUMNS (type PATH '$.type',
                     name PATH '$.name',
                     is_pk PATH '$.is_pk',
                     f_k PATH '$.fk'
                     )))
            LOOP

               query := query || res.name || ' ' || res.type || ','||chr(10);

               IF res.f_k is not null THEN
                  query := query || ' FOREIGN KEY ' || res.name || ' ' || res.f_k;
               END IF;

               IF res.is_pk = 'true' THEN
                  pk_name := res.name;
               END IF;

            END LOOP;
         query := query || ');';

         IF pk_name is not null then
            query:= query || chr(10) || make_trigger(pk_name, table_name);
         END IF;
         DELETE FROM json_tabl where id=(SELECT MAX(ID) FROM json_tabl);
      WHEN 'drop' THEN
         query:= 'DROP TABLE ' || table_name || ' CASCADE CONSTRAINTS;';
```

```
        ELSE
            query := 'error';

    END CASE;
    DELETE FROM json_tabl where id=(SELECT MAX(ID) FROM json_tabl);
    return query;
END make_query;
```

Результат

JSON:

```
{
        "operation": "create",
        "tablen": "json_table1",
        "fields": [
            {
                "name": "id",
                "type": "integer not null",
                "is_pk": true
            },
            {
                "name": "name",
                "type": "varchar(100) not null"
            },
            {
                "name": "university_id",
                "type": "integer not null",
                "fk": "references university(id) on delete cascade"
            }
        ]
    }
```

SQL:

```
[2022-05-18 14:05:59] completed in 20 ms
CREATE TABLE json_table1 (
id integer not null,
name varchar(100) not null,
university_id integer not null,
FOREIGN KEY university_id references university(id) on delete cascade);
CREATE OR REPLACE TRIGGER pk_trigger
BEFORE INSERT
ON json_table1
FOR EACH ROW
DECLARE
counter NUMBER;
max_id NUMBER;
BEGIN
IF :new.id IS NOT NULL THEN
SELECT COUNT(*) INTO counter FROM json_table1 WHERE id=:new.id;
IF counter > 0 THEN
raise_application_error(-20001,'Invalid id');
END IF;
ELSE
SELECT max(id) INTO max_id FROM json_table1;
IF max_id is NULL THEN
:new.id:=1;
ELSE
:new.id:= max_id + 1;
END IF;
END IF;
END;
```

JSON:

```
{
    "operation": "drop",
    "tablen": "json_table1"
}
```

SQL:

```
[2022-05-18 14:10:03] completed in 39 ms
DROP TABLE json_table1 CASCADE CONSTRAINTS;
```