Санкт-Петербургский Национальный Исследовательский университет ИТМО

Факультет Программной Инженерии и Компьютерной Техники

Вариант №1(XML)

Лабораторная работа №3

По дисциплине

‘Низкоуровневое программирование‘

Выполнил:

Верещагин Егор р33312

Преподаватель:

Кореньков Юрий Дмитриевич

Санкт-Петербург 2023 г.

Задание

Задание 3

На базе данного транспортного формата описать схему протокола обмена информацией и воспользоваться существующей библиотекой по выбору для реализации модуля, обеспечивающего его функционирование.
Протокол должен включать представление информации о командах создания, выборки, модификации и удаления данных в соответствии с данной формой, и результатах их выполнения.
Используя созданные в результате выполнения заданий модули, разработать в виде консольного приложения две программы: клиентскую и серверную части. Серверная часть – получающая по сети запросы и операции описанного формата и последовательно выполняющая их над файлом данных с помощью модуля из первого задания. Имя фала данных для работы получать с аргументами командной строки, создавать новый в случае его отсутствия. Клиентская часть – в цикле получающая на стандартный ввод текст команд, извлекающая из него информацию о запрашиваемой операции с помощью модуля из второго задания и пересылающая её на сервер с помощью модуля для обмена информацией, получающая ответ и выводящая его в человеко-понятном виде в стандартный вывод.

Цели

Создать 2 консольных приложения, которые обмениваются информацией через XML по своему протоколу. Серверный модуль должен использовать функции, реализованные в первой ЛР и по сути представляет из себя удаленную СУБД. Клиентский модуль должен парсить запросы на языке AQL и пересылать их серверу.

Задачи

- Изучить доступные средства для парсинга и валидации XML.
- Спроектировать структуры данных для представления запроса.
- Реализовать публичный интерфейс для приведенных выше операций
- Реализовать тестовую программу для демонстрации работоспособности решения

Пример сеанса работы разработанных программ

Для запуска клиента и сервера необходимо передать параметры — для клиента это ip сервера и его порт, а для сервера — его адрес, порт и файл, который будет использоваться для хранения данных

/home/egor/CLionProjects/llp/cmake-build-debug/client 127.0.0.1 8080

/home/egor/CLionProjects/llp/cmake-build-debug/server 127.0.0.1 8080 db.db

Приведем примеры взаимодействия клиента и сервера

Для начала создадим тестовую таблицу

```
/home/egor/CLionProjects/llp/cmake-build-debug/client 127.0.0.1 8080
CREATE_TABLE TEST {i: INT, s: STRING}
^D
Server address: 127.0.0.1
Port: 8080

<?xml version="1.0"?>
<sqlQuery><requestType>CREATE_TABLE</requestType><tableName>TEST</tableName><fields><field><name>i</name><type>INT</type></field><field><name>s</name><type>STRING</type></field>

Message: Table successfully created!

Process finished with exit code 0
```

Попробуем добавить туда значения

```
/home/egor/CLionProjects/llp/cmake-build-debug/client 127.0.0.1 8080
INSERT {i: 123, s: "test"} IN TEST
^D
Server address: 127.0.0.1
Port: 8080
"test"
<?xml version="1.0"?>
<sqlQuery><requestType>INSERT</requestType><tableName>TEST</tableName><insertValues><field><name>i</name><value>123</value><type>INT</type></field><field><name>s</name><value>te

Message: Successfully inserted

Process finished with exit code 0
```

Теперь сделаем select

```
/home/egor/CLionProjects/llp/cmake-build-debug/client 127.0.0.1 8080
FOR u IN TEST
    RETURN u
^D
Server address: 127.0.0.1
Port: 8080


<?xml version="1.0"?>
<sqlQuery><requestType>SELECT</requestType><tableName>TEST</tableName><filters/></sqlQuery>

i: 123
s: test
Message: End of table

Process finished with exit code 0
```

Для чтения строк из бд происходитт несколько запросов, на последний возвращается сообщение, говорящее, что данные кончились(подробнее в следущем разделе)

Теперь обновим данные и проверим, что они обновились

```
/home/egor/CLionProjects/llp/cmake-build-debug/client 127.0.0.1 8080
FOR u IN TEST
    UPDATE {i : u.i, s: "new"} IN TEST
^D
Server address: 127.0.0.1
Port: 8080

"new"
<?xml version="1.0"?>
<sqlQuery><requestType>UPDATE</requestType><tableName>TEST</tableName><filters/><updateValues><field><isValueColumnName>true</isValueColumnName><name>i</name><value>i</value></fiel

Message: Successfully updated

Process finished with exit code 0
```

```
/home/egor/CLionProjects/llp/cmake-build-debug/client 127.0.0.1 8080
FOR u IN TEST
    RETURN u
^D
Server address: 127.0.0.1
Port: 8080


<?xml version="1.0"?>
<sqlQuery><requestType>SELECT</requestType><tableName>TEST</tableName><filters/></sqlQuery>

i: 123
s: NEW
Message: End of table

Process finished with exit code 0
```

Удаление строк и таблицы выглядит следующим образом(фильтр можно применять и к другим запросам, например SELECT или UPDATE)

```
/home/egor/CLionProjects/llp/cmake-build-debug/client 127.0.0.1 8080
FOR u IN TEST
    FILTER u.i == 123
    REMOVE u IN TEST^D
Server address: 127.0.0.1
Port: 8080


<?xml version="1.0"?>
<sqlQuery><requestType>DELETE</requestType><tableName>TEST</tableName><filters><filter><leftOp><isColumnName>true</isColumnName><value>i</value></leftOp><operator>==</operator><ri

Message: Successfully deleted

Process finished with exit code 0
```

```
/home/egor/CLionProjects/llp/cmake-build-debug/client 127.0.0.1 8080
DROP_TABLE TEST
^D
Server address: 127.0.0.1
Port: 8080

<?xml version="1.0"?>
<sqlQuery><requestType>DROP_TABLE</requestType><tableName>TEST</tableName></sqlQuery>

Message: Table successfully dropped!

Process finished with exit code 0
```

Описание решения

Для разработки использовалась библиотека LIBXML2 — одна из самых популярных библиотек под XML на C. К сожалению, библиотека не предоставляет функционал для автогенерации структур, поэтому построение сообщение производилось вручную. Как в клиенте, так и в сервере были реализованы модули net и xml, которые позволяют передавать сообщения и парсить их. Сообщение передается по обычным сокетам, парсинг вида команды происходит с помощью специального тега.

В xml на клиенте данные клались из ast. Функционал парсера был урезан до функциональности сервера — теперь нельзя использовать несколько вложенных FOR, если это не join, иначе коллекция в запросе должна быть одна. Для ее определения все дерево сканируется с помощью dfs и ищется имя таблицы для запроса. Дальше обрабатываются фильтры, если они есть — переменные в них заменяются на имена таблиц, которые им соответсвуют(то есть если было
FOR u IN COL

FILTER u.id == …
в xml в фильтре будет лежать не u, а COL(по-другому нельзя, бд не умеет в переменные)

Далее парсится само действие, которое надо сделать — оно определяется сканированием дерева на ноды определенного типа.

Дальше данные запроса зависят от его типа. Например, если это INSERT, ноды дерева будут преобразованы в теги

<field>

<name>…</name>

<value>…</value>

</field>

и т.д.(схема запроса и ответа дальше)
Сервер получает этот xml и в зависимости от типа запроса строит структуру request

```
enum requestType {
  CREATE_TABLE,
```

```
    DROP_TABLE,
    INSERT,
    SELECT,
    DELETE,
    UPDATE,
    JOIN,
    NEXT
};

struct createTableRequest {
    char *tableName;
    int32_t columnNum;
    enum DataType *types;
    const char **names;
};

struct dropTableRequest {
    char *tableName;
};

struct insertRequest {
    char *tableName;
    int32_t dataCount;
    void **data;
};

struct selectRequest {
    char *tableName;
    int32_t conditionCount;
    struct Condition *conditions;
};


struct deleteRequest {
    char *tableName;
    int32_t conditionCount;
    struct Condition *conditions;
};

struct updateRequest {
    char *tableName;
    int32_t updateColumnsCount;
    struct UpdateColumnValue *updateColumnValues;
    int32_t conditionCount;
    struct Condition *conditions;
};
```

```
struct JoinTable {
    char* name;
    char* alias;
};

struct joinRequest {
    int32_t selectColumnsNum;
    struct TableAliasAndColumn *selectColumns;
    int32_t joinTablesNum;
    struct JoinTable *joinTables;
    int32_t joinConditionNum;
    struct JoinCondition *joinCondition;
    int32_t filtersNum;
    struct JoinWhereCondition *joinWhereCondition;
};

struct request {
    enum requestType type;
    union {
        struct createTableRequest createTableRequest;
        struct dropTableRequest dropTableRequest;
        struct insertRequest insertRequest;
        struct selectRequest selectRequest;
        struct deleteRequest deleteRequest;
        struct updateRequest updateRequest;
        struct joinRequest joinRequest;
    };
};
```

Они отлично маппятся из xml, и почти совпадают со структурами, которые ждет сама бд — только надо кое-где поменять имя таблицы на TableHeader и т. д., чем занимается xml_server — он получает request из xml, преобразует в структуры бд и возвращает результат пользователю.

Все запросы от пользователя выполняются в отдельном потоке — это сделано для того, чтобы можно было несколько раз использовать клиентский сокет для передачи сообщений(главный поток ждет подключений и не может несколько раз обработать сообщение клиента)

Протокол.

Для протокола использован формат xsd — он используется для валидации сообщения. Запрос:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="sqlQuery">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="requestType">
```

```
            <xs:simpleType>
               <xs:restriction base="xs:string">
                  <xs:enumeration value="SELECT"/>
                  <xs:enumeration value="INSERT"/>
                  <xs:enumeration value="DELETE"/>
                  <xs:enumeration value="UPDATE"/>
                  <xs:enumeration value="CREATE_TABLE"/>
                  <xs:enumeration value="DROP_TABLE"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:element>
         <xs:element name="tableName" type="xs:string"/>
         <xs:choice>
            <xs:sequence>
               <!-- If requestType is DELETE or SELECT -->
               <xs:element name="filters" minOccurs="0">
                  <xs:complexType>
                     <xs:sequence>
                        <xs:element name="filter" maxOccurs="unbounded">
                           <xs:complexType>
                              <xs:sequence>
                                 <xs:element name="leftOp">
                                    <xs:complexType>
                                       <xs:sequence>
                                          <xs:element name="isColumnName" type="xs:boolean"/>
                                          <xs:element name="value" minOccurs="0">
                                             <xs:simpleType>
                                                <xs:union memberTypes="xs:string xs:int xs:double
xs:boolean"/>
                                             </xs:simpleType>
                                          </xs:element>
                                       </xs:sequence>
                                    </xs:complexType>
                                 </xs:element>
                                 <xs:element name="operator">
                                    <xs:simpleType>
                                       <xs:restriction base="xs:string">
                                          <xs:enumeration value="=="/>
                                          <xs:enumeration value="&lt;"/>
                                          <xs:enumeration value="&gt;"/>
                                       </xs:restriction>
                                    </xs:simpleType>
                                 </xs:element>
                                 <xs:element name="rightOp">
                                    <xs:complexType>
                                       <xs:sequence>
                                          <xs:element name="isColumnName" type="xs:boolean"/>
                                          <xs:element name="value" minOccurs="0">
                                             <xs:simpleType>
                                                <xs:union memberTypes="xs:string xs:int xs:double
xs:boolean"/>
                                             </xs:simpleType>
                                          </xs:element>
                                       </xs:sequence>
                                    </xs:complexType>
                                 </xs:element>
                              </xs:sequence>
                           </xs:complexType>
                        </xs:element>
                     </xs:sequence>
```

```
                </xs:complexType>
            </xs:element>
        </xs:sequence>
        <xs:element name="join">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="joinConditions" type="JoinConditionType"/>
                    <xs:element name="filterConditions" type="FilterConditionType"
minOccurs="0"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <!-- If requestType is INSERT -->
        <xs:element name="insertValues" minOccurs="0">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="field" maxOccurs="unbounded">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element name="name" type="xs:string"/>
                                <xs:element name="value" type="xs:string"/>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <!-- If requestType is UPDATE -->
        <xs:element name="updateValues" minOccurs="0">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="field" maxOccurs="unbounded">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element name="name" type="xs:string"/>
                                <xs:element name="type">
                                    <xs:simpleType>
                                        <xs:restriction base="xs:string">
                                            <xs:enumeration value="string"/>
                                            <xs:enumeration value="int"/>
                                            <xs:enumeration value="double"/>
                                            <xs:enumeration value="boolean"/>
                                        </xs:restriction>
                                    </xs:simpleType>
                                </xs:element>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <!-- If requestType is CREATE_TABLE -->
        <xs:element name="fields" minOccurs="0">
            <xs:complexType>
                <xs:sequence>
                    <xs:element name="field" maxOccurs="unbounded" minOccurs="0">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element name="name" type="xs:string"/>
                                <xs:element name="type" type="xs:string"/>
```

```
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:choice>
          </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:complexType name="JoinConditionType">
        <xs:sequence>
            <xs:element name="leftTable" type="xs:string"/>
            <xs:element name="leftColumn" type="xs:string"/>
            <xs:element name="operator" type="xs:string"/>
            <xs:element name="rightTable" type="xs:string"/>
            <xs:element name="rightColumn" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="FilterConditionType">
        <xs:sequence>
            <xs:element name="table" type="xs:string"/>
            <xs:element name="column" type="xs:string"/>
            <xs:element name="operator" type="xs:string"/>
            <xs:element name="value" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Ответ

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:complexType name="RowNodeType">
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="val" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="RowType">
        <xs:sequence>
            <xs:element name="rowNode" type="RowNodeType" minOccurs="0"
maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>

    <xs:element name="response">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="message" type="xs:string"/>
                <xs:element name="hasNext" type="xs:boolean"/>
                <xs:element name="row" type="RowType" minOccurs="0"/>
            </xs:sequence>
        </xs:complexType>
```

```
    </xs:element>

</xs:schema>
```

Запрос к серверу содержит имя таблицы, тип запроса и дальше — в зависимости от типа. Ответ от сервера может содержать 3 поля — message — сообщение для клиента, hasNext — сообщение клиенту, что еще есть данные для считывания, row — считанная строка

Выводы

В ходе выполнения лабораторной работы были соединены результаты 1 и 2 лр. Реализовано клиент-серверное взаимодействие с помощью протокола на xml.