**Лабораторная работа № 15. Использование XML**

**КУРНОСЕНКО СОФЬЯ**

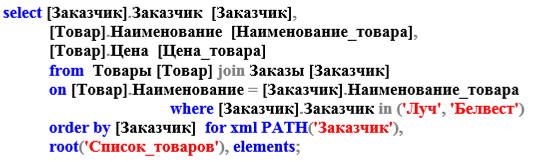
**XML** (Extensible Markup Language) – расширяемый язык разметки. XML-формат часто используется для обмена данными между компонентами информационных систем. При работе с базами данных важными являются две задачи: преобразование *табличных данных в XML-структуры* и преобразование *XML-структур в строки реляционной таблицы*.

1.Разработать сценарий создания XML-документа в режиме PATH из таблицы **TEACHER** для преподавателей кафедры ИСиТ.

***→ Теория (методичка)***

Для преобразования результата SELECT-запроса в формат XML в операторе SELECT применяется секция FOR XML. При этом могут использоваться режимы RAW, AUTO, PATH.

При использовании режима PATH каждый столбец конфигурируется независимо с помощью псевдонима этого столбца.



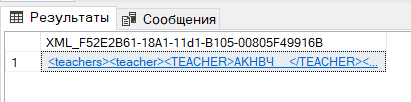
***→ Выполнение заданий***

select \* from TEACHER

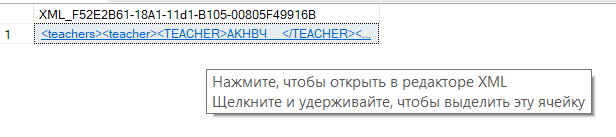
where PULPIT = 'исит'

for xml path('teacher'), root('teachers');

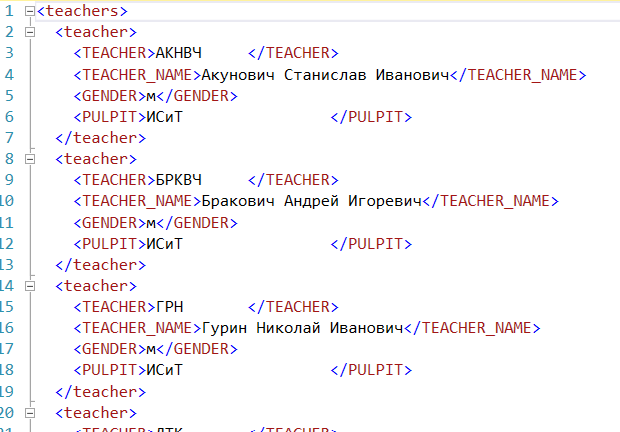
В результате получим ячейку таблицы со ссылкой на документ:

******

При наведении курсора:



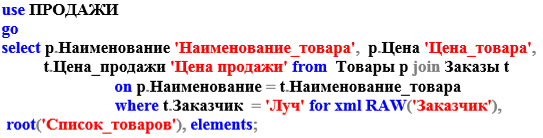
Щелкнем по ссылке и откроем документ (посмотрим частичный результат):



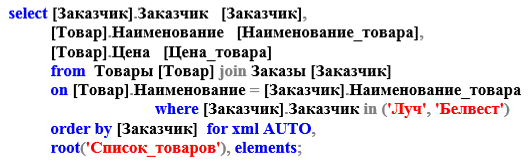
2. Разработать сценарий создания XML-документа в режиме AUTO на основе SELECT-запроса к таблицам **AUDITORIUM** и **AUDITORIUM\_TYPE,** которыйсодержит следующие столбцы: наименование аудитории, наименование типа аудитории и вместимость. Найти только лекционные аудитории.

***→ Теория (методичка)***

В режиме RAW в результате SELECT-запроса создается XML-фрагмент, состоящий из последовательности элементов с именем **row**. Каждый элемент **row** соответствует строке результирующего набора, имена его атрибутов совпадают с именами столбцов результирующего набора, а значения атрибутов равны их значениям. Чтобы раскрыть полностью XML-фрагмент в результирующем наборе надо по фрагменту дважды щелкнуть.



Особенность режима AUTO проявляется в многотабличных запросах. В этом случае режим AUTO позволяет построить XML-фрагмент с применением вложенных элементов.



***→ Выполнение заданий***

select AUDITORIUM\_NAME as [auditorium\_name],

rtrim(aud\_type.AUDITORIUM\_TYPE) as [type],

AUDITORIUM\_CAPACITY as [capacity]

from AUDITORIUM auditorium inner join AUDITORIUM\_TYPE aud\_type

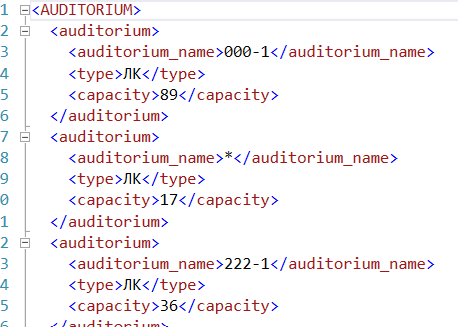
on auditorium.AUDITORIUM\_TYPE = aud\_type.AUDITORIUM\_TYPE

where aud\_type.AUDITORIUM\_TYPE in ('ЛК-К', 'ЛК')

for xml auto,

root ('AUDITORIUM'),

elements

******

3.Разработать XML-документ, содержащий данные о трех новых учебных дисциплинах, которые следует добавить в таблицу **SUBJECT**.

Разработать сценарий, извлекающий данные о дисциплинах из XML-документа и добавляющий их в таблицу **SUBJECT**.

При этом применить системную функцию **OPENXML** и конструкцию INSERT… SELECT.

***→ Теория (методичка)***

Пример преобразования XML-структуры в строки реляционной таблицы:

**use ПРОДАЖИ**

**go**

**declare @h int = 0,**

**@x varchar(2000) = ' <?xml version="1.0" encoding="windows-1251" ?>**

**<товары>**

**<товар="стол" цена="40" количество="5" />**

**<товар="стул" цена="10" количество="3" />**

**<товар="шкаф" цена="400" количество="1" />**

**</товары>';**

**exec sp\_xml\_preparedocument @h output, @x;** -- подготовка документа

**select \* from openxml(@h, '/товары/товар', 0)**

**with([товар] nvarchar(20), [цена] real, [количество] int )**

**exec sp\_xml\_removedocument @h;** -- удаление документа

Для преобразования XML-данных в строки таблицы предназначена функция **OPENXML**, которая принимает три входных параметра: дескриптор, выражение XPATH и целое положительное число, определяющее режим работы функции.

Дескриптор определяется процедурой SP\_XML\_PREPAREDOCUMENT, которая должна быть выполнена до SELECT-запроса, применяющего OPENXML. Процедура принимает в качестве входного параметра XML-документ (в формате строки) и возвращает дескриптор.

Выражение XPATH предназначено для выбора требуемых данных из исходного XML-документа.

Режим работы указывает на тип преобразования (0 − используется атрибутивная модель сопоставления, каждый XML-атрибут преобразовывается в столбец таблицы; 1− аналогично типу 0, но для необработанных столбцов применяется сопоставление на основе элементов XML-документа; 2 − используется сопоставление на основе элементов, каждый элемент преобразовывается в столбец таблицы).

С помощью выражения WITH должна быть указана структура формируемого результата.

Для того, чтобы извлечь данные о товарах из XML-документа и добавить их в таблицу **Товары**, надо заменить оператор

**select \* from openxml(@h, '/товары/товар', 0)**

**with([товар] nvarchar(20), [цена] real, [количество] int )**

на

**insert Товары select [товар], [цена], [количество]**

**from openxml(@h, '/товары/товар', 0)**

**with([товар] nvarchar(20), [цена] real, [количество] int )**

***→ Выполнение заданий***

Формируем документ и преобразуем его в строки таблицы:

go

declare @handler int = 0, -- дескриптор созданныго xml-документа

@xmlText varchar(2000) = '<?xml version="1.0" encoding="windows-1251"?>

<предметы>

<предмет код\_предмета="Зе" наименование="Зельеварение" кафедра="ох"/>

<предмет код\_предмета="А" наименование="Астрономия" кафедра="исит"/>

<предмет код\_предмета="За" наименование="Заклинания" кафедра="ох"/>

</предметы>';

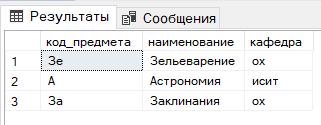
-- создание xml-документа из текста и возврат дескриптора для доступа к документу

exec sp\_xml\_preparedocument @handler output, @xmlText;

-- Для преобразования XML-данных в строки таблицы предназначена функция OPENXML

select \* from openxml(@handler, '/предметы/предмет', 0)

with ([код\_предмета] char(10), [наименование] varchar(100), [кафедра] char(20));

******

Добавление строк в таблицу:

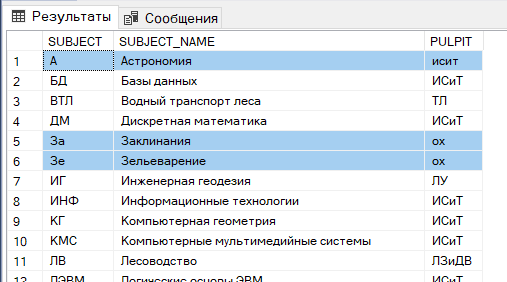
-- добавляем данные в таблицу SUBJECT

insert into SUBJECT

select \* from openxml(@handler, '/предметы/предмет', 0)

with ([код\_предмета] char(10), [наименование] varchar(100), [кафедра] char(20));

В результате в таблице:

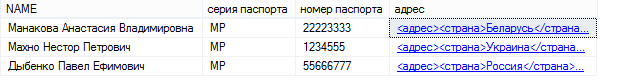


4.Используя таблицу **STUDENT** разработать XML-структуру, содержащую паспортные данные студента: серию и номер паспорта, личный номер, дата выдачи и адрес прописки.

Разработать сценарий, в который включен оператор INSERT, добавляющий строку с XML-столбцом.

Включить в этот же сценарий оператор UPDATE, изменяющий столбец **INFO** у одной строки таблицы **STUDENT** и оператор SELECT, формирующий результирующий набор, аналогичный представленному на рисунке.

В SELECT-запросе использовать методы QUERY и VALUE XML-типа.



***→ Теория (методичка)***

Пусть требуется создать таблицу **Поставщики**, содержащую адреса поставщиков товаров:

**create table Поставщики**

**( Организация nvarchar(50) primary key,**

**Адрес xml** -- столбец XML-типа

**);**

В таблицу помещается информация о поставщиках, например:

**insert into Поставщики (Организация, Адрес)**

**values ('Пинскдрев', '<адрес> <страна>Беларусь</страна>**

**<город>Пинск</город> <улица>Кирова</улица>**

**<дом>52</дом> </адрес>');**

**insert into Поставщики (Организация, Адрес)**

**values ('Минскдрев', '<адрес> <страна>Беларусь</страна>**

**<город>Минск</город> <улица>Кальварийская</улица>**

**<дом>35</дом> </адрес>');**

Для обновления, например, номера дома можно использовать оператор UPDATE:

**update Поставщики**

**set Адрес = '<адрес> <страна>Беларусь</страна>**

**<город>Минск</город> <улица>Кальварийская</улица>**

**<дом>45</дом> </адрес>'**

**where Адрес.value('(/адрес/дом)[1]','varchar(10)') = 35;**

Содержимое таблицы **Поставщики** показывает оператор SELECT:

**select Организация,**

**Адрес.value('(/адрес/страна)[1]','varchar(10)') [страна],**

**Адрес.query('/адрес') [адрес]**

**from Поставщики;**

***→ Выполнение заданий***

-- паспортные данные

declare @passportInfo\_1 varchar(2000) =

'<passport\_details>

<серия>MP</серия>

<номер>1234112</номер>

<дата\_выдачи>03.05.2022</дата\_выдачи>

<адрес\_прописки>

<страна>Беларусь</страна>

<город>Волковыск</город>

<улица>Восточная</улица>

<дом>65</дом>

</адрес\_прописки>

</passport\_details>';

declare @passportInfo\_2 varchar(2000) =

'<passport\_details>

<серия>MP</серия>

<номер>048765</номер>

<дата\_выдачи>04.05.2022</дата\_выдачи>

<адрес\_прописки>

<страна>Россия</страна>

<город>Иркутск</город>

<улица>Красноказачья</улица>

<дом>124</дом>

</адрес\_прописки>

</passport\_details>';

declare @passportInfo\_3 varchar(2000) =

'<passport\_details>

<серия>MP</серия>

<номер>09876543</номер>

<дата\_выдачи>05.05.2022</дата\_выдачи>

<адрес\_прописки>

<страна>Россия</страна>

<город>Воркута</город>

<улица>Ленина</улица>

<дом>37</дом>

</адрес\_прописки>

</passport\_details>';

-- для обновления

declare @passportInfo\_3\_update varchar(2000) =

'<passport\_details>

<серия>MP</серия>

<номер>09876543</номер>

<дата\_выдачи>05.05.2022</дата\_выдачи>

<адрес\_прописки>

<страна>Россия</страна>

<город>Воркута</город>

<улица>Транспортная</улица>

<дом>19</дом>

</адрес\_прописки>

</passport\_details>';

-- вставка строк в таблицу STUDENT

insert into STUDENT (IDGROUP, NAME, BDAY, INFO)

values (3, 'студент\_1', getdate(), @passportInfo\_1),

(3, 'студент\_2', getdate(), @passportInfo\_2),

(3, 'студент\_3', getdate(), @passportInfo\_3);

-- Обновление

update STUDENT

set INFO = @passportInfo\_3\_update

where NAME = 'студент\_3';

-- вывод добавленных строк

select NAME as [ФИО],

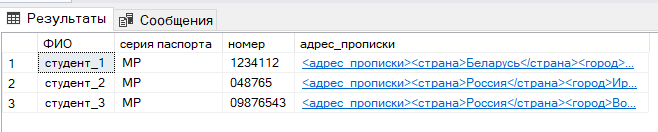
INFO.value('(/passport\_details/серия)[1]', 'varchar(5)') as [серия паспорта],

INFO.value('(/passport\_details/номер)[1]', 'varchar(15)') as [номер],

INFO.query('(/passport\_details/адрес\_прописки)') as [адрес\_прописки]

from STUDENT

where INFO is not null;

******

5. Изменить (ALTER TABLE) таблицу **STUDENT** в базе данных **UNIVER** таким образом, чтобы значения *типизированного* столбца с именем **INFO** контролировались коллекцией XML-схем (XML SCHEMACOLLECTION), представленной в правой части.

Разработать сценарии, демонстрирующие ввод и корректировку данных (операторы INSERT и UPDATE) в столбец **INFO** таблицы **STUDENT**, как содержащие ошибки, так и правильные.

Разработать другую XML-схему и добавить ее в коллекцию XML-схем в БД UNIVER**.**

***→ Теория (методичка)***

Пусть имеется объект XML SCHEMACOLLECTION с именем **Student**:

**use UNIVER**

**go**

**create xml schema collection Student as**

**N'<?xml version="1.0" encoding="utf-16" ?>**

**<xs:schema attributeFormDefault="unqualified"**

**elementFormDefault="qualified"**

**xmlns:xs="http://www.w3.org/2001/XMLSchema">**

**<xs:element name="студент">**

**<xs:complexType><xs:sequence>**

**<xs:element name="паспорт" maxOccurs="1" minOccurs="1">**

**<xs:complexType>**

**<xs:attribute name="серия" type="xs:string" use="required" />**

**<xs:attribute name="номер" type="xs:unsignedInt" use="required"/>**

**<xs:attribute name="дата" use="required" >**

**<xs:simpleType> <xs:restriction base ="xs:string">**

**<xs:pattern value="[0-9]{2}.[0-9]{2}.[0-9]{4}"/>**

**</xs:restriction> </xs:simpleType>**

**</xs:attribute> </xs:complexType>**

**</xs:element>**

**<xs:element maxOccurs="3" name="телефон" type="xs:unsignedInt"/>**

**<xs:element name="адрес"> <xs:complexType><xs:sequence>**

**<xs:element name="страна" type="xs:string" />**

**<xs:element name="город" type="xs:string" />**

**<xs:element name="улица" type="xs:string" />**

**<xs:element name="дом" type="xs:string" />**

**<xs:element name="квартира" type="xs:string" />**

**</xs:sequence></xs:complexType> </xs:element>**

**</xs:sequence></xs:complexType>**

**</xs:element>**

**</xs:schema>';**

Отформатированный вид для лучшего воприятия:

****

Документ XML-Schema, размещенный в коллекции **Student**, описывает XML-документ с корневым элементом **студент** (первый тэг **element**).

На втором уровне (внутри тега **студент**) расположено три тэга: **паспорт**, **телефон** и **адрес** (вложенные теги **element**). Причем тэг **паспорт** должен быть ровно один (атрибуты **maxOccurs** и **minOccurs**); тэг **телефон** является обязательным и может быть в количестве не более трех (атрибут **maxOccurs**); тэг **адрес** является обязательным, и количество таких тэгов не должно быть более одного.

Элементы третьего уровня (**страна**, **город**, **улица**, **дом**, **квартира**) присутствуют только внутри элемента **адрес**. Все эти элементы являются обязательными и должны присутствовать ровно один раз.

Данные в документе размещаются как значения атрибутов (теги **attribute**) или как значения, размещенные в теле элементов (**телефон**, **страна**, **город**, **улица**, **дом**, **квартира**).

Тип данных, размещаемых в атрибутах или теле элементов данных, определяется значением атрибута **type**.

Пример создания таблицы **STUDENT** с типизированной структурой:

**drop table STUDENT;**

**go**

**create table STUDENT**

**( IDSTUDENT integer identity(1000,1) primary key,**

**IDGROUP integer foreign key references GROUPS(IDGROUP),**

**NAME nvarchar(100),**

**BDAY date,**

**STAMP timestamp,**

**INFO xml(STUDENT),** -- типизированный столбец XML-типа

**FOTO varbinary**

**);**

***→ Выполнение заданий***

-- создаем схему и добавляем в БД UNIVERSITY

use UNIVERSITY

go

create xml schema collection Student as

N'<?xml version="1.0" encoding="utf-16" ?>

<xs:schema attributeFormDefault="unqualified"

elementFormDefault="qualified"

xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="студент">

<xs:complexType><xs:sequence>

<xs:element name="паспорт" maxOccurs="1" minOccurs="1">

<xs:complexType>

<xs:attribute name="серия" type="xs:string" use="required" />

<xs:attribute name="номер" type="xs:unsignedInt" use="required"/>

<xs:attribute name="дата" use="required" >

<xs:simpleType>

<xs:restriction base ="xs:string">

<xs:pattern value="[0-9]{2}.[0-9]{2}.[0-9]{4}"/>

</xs:restriction>

</xs:simpleType>

</xs:attribute>

</xs:complexType>

</xs:element>

<xs:element maxOccurs="3" name="телефон" type="xs:unsignedInt"/>

<xs:element name="адрес">

<xs:complexType>

<xs:sequence>

<xs:element name="страна" type="xs:string" />

<xs:element name="город" type="xs:string" />

<xs:element name="улица" type="xs:string" />

<xs:element name="дом" type="xs:string" />

<xs:element name="квартира" type="xs:string" />

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:sequence></xs:complexType>

</xs:element>

</xs:schema>';

-- удаляем из столбца INFO значения, ибо далее мы зотим установить для этих значений схему (шаблон),

-- а текущие данные с этом столбце не соответствуют этой схеме, из-за чего появится ошибка

update STUDENT

set INFO = NULL;

-- делаем INFO типизированным столбцом XML-типа

alter table STUDENT

alter column INFO xml(Student);

-- некорректные xml-данные

go

declare @passportInfo varchar(2000) =

'<passport\_details>

<серия>MP</серия>

<номер>09876543</номер>

<дата\_выдачи>05.05.2022</дата\_выдачи>

<адрес\_прописки>

<страна>Россия</страна>

<город>Воркута</город>

<улица>Ленина</улица>

<дом>28</дом>

</адрес\_прописки>

</passport\_details>';

insert into STUDENT (IDGROUP, NAME, BDAY, INFO)

values (3, 'студент', getdate(), @passportInfo);

-- корректные

go

declare @passportInfo varchar(2000) =

'<студент>

<паспорт серия="MP" номер="09876543" дата="05.05.2022"/>

<телефон>290231036</телефон>

<адрес>

<страна>Россия</страна>

<город>Воркута</город>

<улица>Ленина</улица>

<дом>28</дом>

<квартира>6</квартира>

</адрес>

</студент>';

insert into STUDENT (IDGROUP, NAME, BDAY, INFO)

values (3, 'студент', getdate(), @passportInfo);

-- убираем схему из определения типа столбца

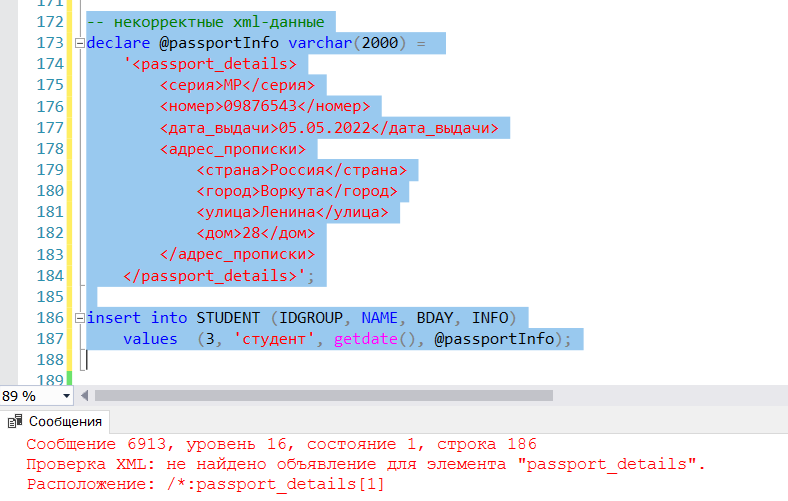
alter table STUDENT

alter column INFO xml;

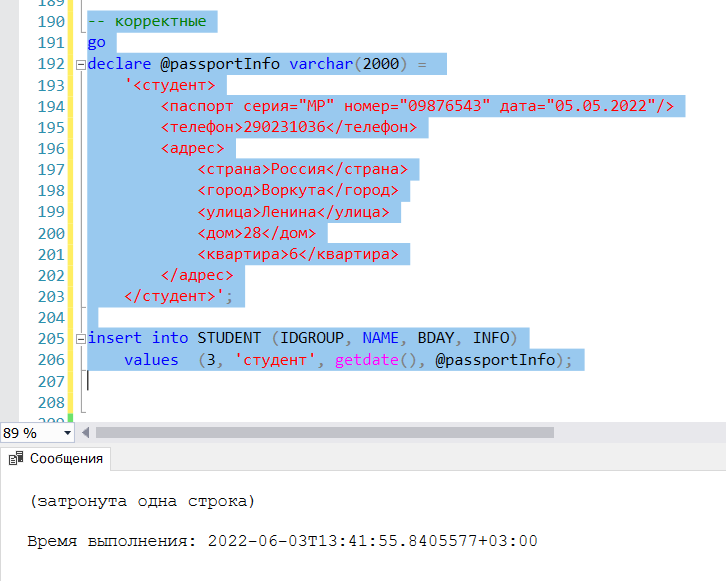
-- удаляем схему

drop XML SCHEMA COLLECTION Student;

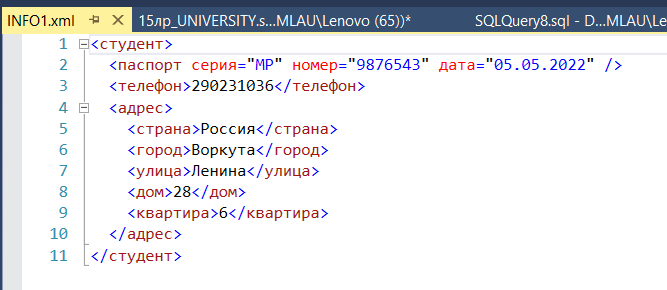
Теперь подходившие ранее данные не подходят из-за схемы:



Вставка корректных данных:



В итоге в ячейке будет храниться документ такого вида:



Создание своей схемы:

use UNIVERSITY

go

create xml schema collection MySchema as

N'<?xml version="1.0" encoding="utf-16" ?>

<xs:schema attributeFormDefault="unqualified"

elementFormDefault="qualified"

xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="паспорные\_данные">

<xs:complexType>

<xs:sequence>

<xs:element maxOccurs="1" minOccurs="1" name="серия" type="xs:string"/>

<xs:element maxOccurs="1" minOccurs="1" name="номер" type="xs:string"/>

<xs:element maxOccurs="1" minOccurs="1" name="дата\_выдачи">

<xs:simpleType>

<xs:restriction base="xs:string">

<xs:pattern value="[0-9]{2}.[0-9]{2}.[0-9]{4}"/>

</xs:restriction>

</xs:simpleType>

</xs:element>

<xs:element name="адрес\_прописки">

<xs:complexType>

<xs:sequence>

<xs:element name="страна" type="xs:string" />

<xs:element name="город" type="xs:string" />

<xs:element name="улица" type="xs:string" />

<xs:element name="дом" type="xs:string" />

<xs:element name="квартира" type="xs:string" />

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>';

update STUDENT

set INFO = NULL;

alter table STUDENT

alter column INFO xml(mySchema);

-- вставка строки

go

declare @passportInfo varchar(2000) =

'<паспорные\_данные>

<серия>MP</серия>

<номер>09876543</номер>

<дата\_выдачи>05.05.2022</дата\_выдачи>

<адрес\_прописки>

<страна>Россия</страна>

<город>Воркута</город>

<улица>Ленина</улица>

<дом>28</дом>

<квартира>6</квартира>

</адрес\_прописки>

</паспорные\_данные>';

insert into STUDENT (IDGROUP, NAME, BDAY, INFO)

values (3, 'студент', getdate(), @passportInfo);

-- убираем схему из определения типа столбца

alter table STUDENT

alter column INFO xml;

-- удаляем схему

drop XML SCHEMA COLLECTION mySchema;