



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика, искусственный интеллект и системы управления

КАФЕДРА Системы обработки информации и управления

**Методические указания к лабораторным работам
по курсу «Постреляционные базы данных»**

**Лабораторная работа №6
«Работа с полуструктурированными данными
в формате XML»**

Виноградова М.В., Зубаиров В.А., Волков А.С.

Под редакцией к.т.н. доц. Виноградовой М.В.

Москва, 2022 г.

ОГЛАВЛЕНИЕ

1. ЗАДАНИЕ.....	3
1.1. Цель работы.....	3
1.2. Средства выполнения.....	3
1.3. Продолжительность работы.....	3
1.4. Пункты задания для выполнения.....	3
1.5. Содержание отчета.....	4
2. ТЕОРЕТИКО-ПРАКТИЧЕСКИЙ МАТЕРИАЛ.....	5
2.1. Работа с СУБД PostgreSQL через pgAdmin.....	5
2.2. Язык XML.....	5
2.2.1. Синтаксис языка XML.....	5
2.2.2. Правильный и действительный XML-документы.....	7
2.3. Язык запросов XPath.....	7
2.3.1. Объектная модель документа DOM.....	7
2.3.2. Путевое выражение.....	9
2.3.3. Функции в языке XPath.....	11
2.3.4. Примеры запросов на языке XPath.....	13
2.4. Тип XML в PostgreSQL.....	14
2.4.1. Создание XML-значений.....	15
2.4.2. Обращение к XML-значениям.....	15
2.5. XML-функции в PostgreSQL.....	16
2.5.1. Создание XML-контента.....	16
2.5.2. Обработка XML.....	21
2.5.3. Отображение таблиц в XML.....	23
2.6. Пример работы с XML в PostgreSQL.....	26
2.6.1. Создание таблиц.....	26
2.6.2. Экспорт таблиц и запросов в XML.....	27
2.6.3. Создание XML на основе таблиц.....	30
2.6.4. Примеры запросов к XML-документу.....	34
3. КОНТРОЛЬНЫЕ ВОПРОСЫ.....	36
4. СПИСОК ИСТОЧНИКОВ.....	37

1. ЗАДАНИЕ

Лабораторная работа №6 «Работа с полуструктурированными данными в формате XML» по курсу «Постреляционные базы данных».

1.1. Цель работы

- Изучить языки запросов XPath и XQuery к XML-документам [1].
- Освоить методы работы с XML в постреляционных СУБД.
- Получить навыки экспорта в XML и запроса к XML-данным на примере СУБД PostgreSQL [2].

1.2. Средства выполнения

- СУБД PostgreSQL,
- Утилита PgAdmin [3].

1.3. Продолжительность работы

Время выполнения лабораторной работы 4 часа.

1.4. Пункты задания для выполнения

1. **(базовое)** Через PgAdmin соединиться с PostgreSQL и создать базу данных. В БД создать две-три связанные таблицы по теме, выданной преподавателем. Открыть таблицы на редактирование и заполнить тестовыми данными. Не менее 5 записей в каждой таблице. Иметь поля с NULL, висающие относительно соединения записи в обеих таблицах и несколько дочерних записей к одной родительской.
2. **(базовое)** В среде построения запросов PgAdmin продемонстрировать просмотр экспорт содержимого таблиц в xml в следующих вариантах:
 - все поля — элементы,
 - все поля — атрибуты,
 - добавление корневого элемента,
 - переименование строк,
 - получение xml-схемы по умолчанию,
 - отображение значений NULL.

3. **(расширенное)** В среде построения запросов PgAdmin продемонстрировать экспорт результата запроса с условием в XML и экспорт содержимого всех таблиц в xml произвольной структуры:
 - с атрибутами,
 - с дочерними элементами,
 - с атрибутами дочерних элементов.
4. **(расширенное)** В среде построения запросов создать сценарии для чтения xml из файла (взять xml документ сложной структуры, полученный ранее). Выполнить запросы с условием на получение данных в виде таблиц.
5. **(базовое)** В среде построения запросов создать сценарии для чтения xml из файла (взять xml документ сложной структуры, полученный ранее). На языке XPath выполнить запросы:
 - Проверки существования данных (атрибутов, элементов и их значений);
 - Извлечения данных (атрибутов, элементов и содержимого);
 - Получения фрагмента XML.
6. **(дополнительное)** Использовать конструкции Xpath и выполнить запросы к XML с условием, агрегацией и группировкой.

1.5. Содержание отчета

- Титульный лист;
- Цель работы;
- Задание;
- Тексты запросов и команд в соответствии с пунктами задания.
- Результаты выполнения запросов (скриншоты);
- Вывод;
- Список используемой литературы.

2. ТЕОРЕТИКО-ПРАКТИЧЕСКИЙ МАТЕРИАЛ

2.1. Работа с СУБД PostgreSQL через pgAdmin

Порядок работы с СУБД PostgreSQL [2] через утилиту pgAdmin [3] был подробно описан в Лабораторной работе №1 «Создание объектно-реляционной базы данных на примере СУБД PostgreSQL» по курсу «Постреляционные базы данных».

2.2. Язык XML

Язык XML (Extensible Markup Language) — расширяемый язык разметки [4]. Основан на языке SGML. Документ в формате XML содержит текстовые данные и тэги. Тэги формируют семантическую разметку документа, определяют смысловое назначение данных.

Язык XML задает в линейной форме полуструктурированные данные, тэги XML являются метками дуг графа полуструктурированных данных.

Пример документа XML:

```
<?xml version="1.0" standalone="yes"
      encoding=" Windows-1251" ?>
<book>
  <avtor>
    <fio>Текст</fio>
  </avtor>
  <izdanie city="Mos">Еще текст</izdanie>
  <bestbook/>
  <dt year="2000" />
</book>
```

2.2.1. Синтаксис языка XML

Каждый элемент задается **тэгом**:

```
<тэг> содержимое элемента </тэг>
```

где <тэг> — начальный (открывающий) тэг;

</тэг> — завершающий (закрывающий) тэг;

Между открывающим и закрывающим тэгом находится содержимое элемента.

Название тэга соответствует названию элемента:

```
<fio>Текст</fio>
```

Соответственно, следующим образом будет выглядеть **пустой элемент** (без содержимого):

```
<bestbook/>
```

Вложенность тэгов является произвольной, но строго иерархической.

Корневой элемент ровно один на весь документ и включает все прочие элементы. Он является обязательным. Например, в случае, если в XML-документе описываются характеристики конкретной книги, то корневым элементом будет:

```
<book> ... </book>
```

Атрибуты характеризуют элемент. Атрибуты указывают в открывающем тэге, их значения пишут в одинарных или двойных кавычках (атрибуты `city` и `year` в примере ниже):

```
<izdanie city="Mos"> ... </izdanie>  
<dt year="2000" />
```

XML-документ начинается строкой:

```
<?xml version="1.0" standalone="yes" encoding="Windows-1251" ?>
```

где `version` — версия документа (версии 1.0 или 1.1); `standalone` — самодостаточность документа (не требуются дополнительные файлы); `encoding` — кодировка элемента.

Комментарии указывают так:

```
<!-- комментарий -->
```

Названия тегов и атрибутов регистрозависимы.

Для отображения **служебных символов** используют их коды:

- `<` — знак «<»;
- `>` — знак «>»;
- `&` — знак «&»;
- `"` — знак «кавычки»;
- `&#код;` — символ с указанным кодом.

Для отображения **неформатированного текста** используют конструкцию:

```
<![CDATA[ текст ]]>
```

Для включения **инструкций обработки** используют конструкцию:

```
<?app .... ?>
```

2.2.2. Правильный и действительный XML-документы

XML-документ считается **правильным** («well formed»), если:

- содержит любые авторские тэги;
- не имеет определения схемы;
- записывает в линейной форме полуструктурированный граф.

XML-документ считается **действительным** («valid»), если:

- соответствует некоторой схеме, которая задает допустимый набор тэгов, их атрибутов и их вложенность;
- является средним между жесткой схемой (реляционной БД) и отсутствием схемы (полуструктурированными данными), поскольку в нем могут быть поля, которые не используются.

2.3. Язык запросов XPath

Языки XPath и XQuery используют для составления запросов к данным в XML. Оба языка разработаны консорциумом W3C.

Язык XPath (XML Path Language) — язык запросов к элементам XML документа [5]; предназначен для навигации по XML-документу; позволяет указать путь к элементу, условие выборки и извлекаемое значение.

2.3.1. Объектная модель документа DOM

Каждый XML-документ может быть представлен как дерево. Элементы и атрибуты документа будут узлами дерева, а отношение вложенности элементов или принадлежности атрибутов — дугами. Такое дерево называют **объектной моделью документа** (DOM — document object model).

При анализе любого XML-документа строится его объектная модель.

В качестве примера рассмотрим и будем использовать в дальнейшем XML-документ следующего содержания:

```

<?xml version="1.0" encoding="UTF8" standalone="no"?>
<!DOCTYPE db SYSTEM "file.dtd">
<db>
  <film idf="f1" toact="a1 a2" tos="s1">
    <name>Чапаев</name>
    <year>1934</year>
    <len>63</len>
    <type>драма</type>
  </film>
  <actor ida="a1" tof="f1">
    <fio>Иванов И.И. </fio>
    <edu>Щукинское училище</edu>
  </actor>
  <actor ida="a2" tof="f1">
    <fio>Бабочкин Б.А.</fio>
    <edu>Студия Певцова</edu>
  </actor>
  <studio ids="s1" tof="f1">
    <name>Ленфильм</name>
    <addr>Санкт-Петербург, Россия</addr>
  </studio>
</db>

```

Посмотрим DOM-модель документа. Рисунок 1 демонстрирует полученное дерево элементов.



Рисунок 1 - DOM-модель XML-документа

Язык XPath предназначен для перемещения по дереву и позволяет указать подмножество узлов документа, удовлетворяющих заданному условию.

Кроме элементов и атрибутов узлами считают комментарии, текстовое содержимое элементов, инструкции, пространства имен.

2.3.2. Путевое выражение

В запросе на языке XPath указывают направление движения по дереву документа от некоторого узла, направление задается осью.

Краткий синтаксис. Пример **путевого выражения** для извлечения названия фильма по его идентификатору (атрибуту idf):

```
/db/film[@idf="f1"]/name
```

Полный синтаксис выглядит так:

```
/child::db/child::film[attribute::idf="f1"]/child::name
```

В данном случае путь к элементу расшифровывается так:

- `/child::db/` — шаг первый, от корня документа взять дочерний элемент с названием `db`,
- `child::film[attribute::idf="f1"]/` — шаг второй, от полученного результата взять дочерний элемент с названием `film`, для которого значение атрибута `idf` равно «f1»,
- `child::name` — шаг третий, от полученного результата взять дочерний элемент с названием `name`

Слово `child` является **осью документа** и определяет отношение элемента к текущему узлу. Существует множество различных осей документа:

- `ancestor` — предки, то есть узлы выше текущего по иерархии вложенности;
- `parent` (родитель) — элемент, в который вложен данный;
- `self` — сам элемент;
- `child` — дети, то есть узлы, непосредственно вложенные в текущий элемент (по умолчанию);
- `descendant` — потомки, то есть узлы ниже текущего по иерархии вложенности;
- `ancestor-or-self` — предки или текущий;
- `descendant-or-self` — потомки или текущий;
- `attribute` — атрибут элемента;
- `namespace` — элемент с пространством имен (с атрибутом `xmlns`);
- `following` — следующие элементы после текущего по тексту документа, кроме его потомков;
- `following-sibling` — элементы после текущего по тексту документа на его уровне вложенности;
- `preceding` — предыдущие элементы до текущего, кроме его предков;
- `preceding-sibling` — предыдущие элементы на его уровне вложенности.

В путевом выражении выделяют:

- шаги адресации;

- предикат;
- условие проверки узлов.

Шаги адресации задают слева направо и разделяются знаком «/».

Шаг содержит:

- контекст — результат выполнения предыдущего шага (множество узлов). На первом шаге равен корню документа. От него вычисляют новый результат шага;
- ось (обязательна) — определяет направление движения на шаге;
- условие (обязательно) — условие выбора узлов по оси: либо имена узлов, либо знак * (все узлы);
- предикат (необязательный) — фильтр или вычисляемое выражение для фильтрации множества узлов.

В путевом выражении допускаются следующие **сокращения**:

- descendant (потомки) — «//»;
- parent (родитель) — «..»;
- self (сам элемент) — «.»;
- child (дети) — по умолчанию без обозначения;
- attribute — атрибут элемента «@имя».

2.3.3. Функции в языке XPath

Язык XPath располагает множеством различных функций [5], предназначенных для выполнения разного рода действий. Ниже приведены основные функции.

Функции работы с множествами:

- node() — множество всех узлов в т.ч. текстовых;
- text() — текстовый узел (//text() — все текстовые узлы);
- current() — текущий элемент;
- last() — номер последнего элемента в множестве;
- position() — номер элемента в множестве;
- count() — количество элементов в множестве;
- name(множество) — имя первого элемента в множестве;
- name() — имя элемента

Логические функции – функции для реализации логических операций:

- or — ИЛИ, and — И, not — отрицание;
- = равно, != не равно;
- > больше, < меньше, >= больше или равно, <= меньше или равно;
- boolean(объект) — возвращает true или false;
- true() — истина; false() — ложь.

Числовые функции – функции для работы с числовыми данными:

- + сумма, - разность, * умножение, div деление, mod остаток от деления по модулю;
- sum() — сумма элементов множества;
- round() — округление значения;
- number(значение) — приведение значения к числу.

Строковые функции – функции для работы со строковыми данными:

- string(объект) — приведение к строке;
- string-length(строка) — длина строки;
- contains(исходная_строка, подстрока) — проверка вхождения подстроки в строку;
- substring(строка, начало, количество) — извлечь подстроку начиная с символа (начало) и взять количество символов;
- starts-with(строка_исходная, подстрока) — проверка начала строки с подстроки;
- concat (s1, s2...) — конкатенация строк;
- translate(строкаA, AB , ab) — замена в 1-ой строке символов из букв 2-й строки на символы из 3-й строки.

Системные функции:

- document(объект) — возвращает документ;
- document(xml-документ) — возвращает множество узлов;
- collection(набор элементов без корня) — возвращает множество узлов.

2.3.4. Примеры запросов на языке XPath

Все приведенные ниже запросы будут выполняться к ранее описанному XML-документу с информацией о фильмах и актерах.

Перечислим несколько возможных **запросов к элементам**:

- `/db` — корень (возвращает весь документ);
- `/db/film` — все фильмы (возвращает все элементы `<film>..</film>` вместе с их содержимым и тэгами);
- `//film` — все фильмы (аналог предыдущего в сокращенном варианте);
- `//film/name` — элементы `name` внутри элементов `film`;
- `//name` — любые элементы `name` (дочерние для элементов `film` и `studio`);
- `//*` — все элементы;
- `/ **/name` — элементы `name` 3-ого уровня;
- `//actor | // studio` — объединение нескольких путей (актеров и студий).

Запросы к содержимому элемента будут выглядеть следующим образом:

- `//name/text()` — содержимое элементов `name`;
- `//fio [. = "Иванов И.И."]` — элемент `fio`, значение (текст) которого равно «Иванов И.И.».

Запросы к элементам с условием будут выполняться следующим образом:

- `//film[1]` — первый элемент `film`;
- `//film[last ()]` — последний элемент `film`;
- `//fio[position() mod 2 = 0]` — каждый второй элемент `fio`;
- `//film[@idf]` — все элементы `film`, имеющие атрибут `idf`;
- `//film[@*]` — все элементы `film`, имеющие атрибуты;
- `//film[not (@*)]` — все элементы `film`, не имеющие атрибутов;
- `//film[(len or type) and name]` — все все элементы `film`, имеющие вложенные элементы `len` или `type` и `name`;
- `//film[@idf = "f1"]` — элементы `film`, для которых атрибут `idf` равен «f1» (в данном случае один фильм);
- `//actor[count(edu) > 3]` — элементы `actor`, имеющие более 3-х вложенных элементов `edu`;

- `//*[start-with(name()), "a"]` — все элементы, названия которых начинаются с «а»;
- `//film[3[year="1950"]]` — третий фильм, если он снят в 1950 году;
- `//film[year="1950"][3]` — все фильмы, снятые в 1950, и взять из них третий.

Запросы к атрибутам описываются следующим образом:

- `//*[@*]` — все атрибуты всех элементов (возвращает набор значений);
- `//film/@idf` — значения атрибутов `idf` всех элементов `film`;
- `//film[year/text()>2000]/@tos` — значения атрибутов `tos` тех элементов `film`, где вложенный элемент `year` больше 2000 (идентификаторы всех студий, снявших фильмы после 2000 года).

Наконец, так можно описать **запросы на соединение элементов**:

- `//studio[@ids = //film[year>2012]/@tos]/name` — названия студий, снявших фильмы после 2012 года (студия, идентификатор которой указана в атрибуте фильма, снятого после 2012, и взять ее название);
- `//studio[not (@ids = //film[year>2012]/@tos)]/name` — названия студий, не снимавших фильмы после 2012 года (студия, такая, что не существует фильма, снятого после 2012, в атрибуте которого указан идентификатор данной студии, и взять название студии).

2.4. Тип XML в PostgreSQL

Тип `xml` предназначен для хранения XML-данных. Его преимущество по сравнению с обычным типом `text` в том, что он проверяет вводимые значения на допустимость по правилам XML и для работы с ним есть типобезопасные функции [6].

Тип `xml` может сохранять правильно оформленные «документы», в соответствии со стандартом XML, а также фрагменты «содержимого», определяемые как менее ограниченные «узлы документа» в модели данных XQuery и XPath. Другими словами, это означает, что во фрагментах содержимого может быть несколько элементов верхнего уровня или текстовых узлов. Является ли некоторое значение типа `xml` полным документом или фрагментом содержимого, позволяет определить выражение `xml-значение IS DOCUMENT`.

2.4.1. Создание XML-значений

Чтобы получить значение типа `xml` из текстовой строки, используйте функцию `xmlparse`:

```
XMLPARSE ( { DOCUMENT | CONTENT } value)
```

Примеры:

```
XMLPARSE (DOCUMENT '<?xml
version="1.0"?><book><title>Manual</title><chapter>...</chapter></book>')
XMLPARSE (CONTENT 'abc<foo>bar</foo><bar>foo</bar>')
```

Хотя в стандарте SQL описан только один способ преобразования текстовых строк в XML-значения, специфический синтаксис PostgreSQL тоже допустим:

```
xml '<foo>bar</foo>'
'<foo>bar</foo>'::xml
```

Обратная операция, получение текстовой строки из `xml`, выполняется с помощью функции `xmlserialize`:

```
XMLSERIALIZE ( { DOCUMENT | CONTENT } значение AS тип )
```

Здесь допустимый тип — `character`, `character varying` или `text` (или их псевдонимы).

2.4.2. Обращение к XML-значениям

Тип `xml` отличается от других тем, что для него не определены никакие операторы сравнения, так как чётко определённого и универсального алгоритма сравнения XML-данных не существует. Одно из следствий этого — нельзя отфильтровать строки таблицы, сравнив столбец `xml` с искомым значением. Поэтому обычно XML-значения должны дополняться отдельным ключевым полем, например `ID`. Можно также сравнивать XML-значения, преобразовав их сначала в текстовые строки, но заметьте, что с учётом специфики XML-данных этот метод практически бесполезен [6].

2.5. XML-функции в PostgreSQL

Функции и подобные им выражения, описанные в этом разделе, работают со значениями типа `xml`. Выражения `xmlparse` и `xmlserialize`, преобразующие значения `xml` в текст и обратно, здесь повторно не рассматриваются.

2.5.1. Создание XML-контента

Для получения XML-контента из данных SQL существует целый набор функций и функциональных выражений, особенно полезных для выдачи клиентским приложениям результатов запроса в виде XML-документов [7].

2.5.1.1. XML-комментарий

`xmlcomment(текст)`

Функция `xmlcomment` создаёт XML-значение, содержащее XML-комментарий с заданным текстом. Этот текст не должен содержать «- -» или заканчиваться знаком «-», чтобы результирующая конструкция была допустимой в XML. Если аргумент этой функции `NULL`, результатом её тоже будет `NULL`.

Пример:

```
SELECT xmlcomment('hello');
      xmlcomment
-----
<!--hello-->
```

2.5.1.2. Объединение нескольких XML-значений

`xmlconcat(xml[, ...])`

Функция `xmlconcat` объединяет несколько XML-значений и выдаёт в результате один фрагмент XML-контента. Значения `NULL` отбрасываются, так что результат будет равен `NULL`, только если все аргументы равны `NULL`.

Пример:

```
SELECT xmlconcat('<abc/>', '<bar>foo</bar>');
      xmlconcat
-----
```



```
<abc/><bar>foo</bar>
```

XML-объявления, если они присутствуют, обрабатываются следующим образом. Если во всех аргументах содержатся объявления одной версии XML, эта версия будет выдана в результате; в противном случае версии не будет. Если во всех аргументах определён атрибут `standalone` со значением «yes», это же значение будет выдано в результате. Если во всех аргументах есть объявление `standalone`, но минимум в одном со значением «no», в результате будет это значение. В противном случае в результате не будет объявления `standalone`. Если же окажется, что в результате должно присутствовать объявление `standalone`, а версия не определена, тогда в результате будет выведена версия 1.0, так как XML-объявление не будет допустимым без указания версии. Указания кодировки игнорируются и будут удалены в любых случаях.

Пример:

```
SELECT xmlconcat('<?xml version="1.1"?><foo/>', '<?xml version="1.1"
standalone="no"?><bar/>');
```

```
xmlconcat
```

```
-----
<?xml version="1.1"?><foo/><bar/>
```

2.5.1.3. XML-элемент

```
xmlelement(name имя [, xmlattributes(значение [AS атрибут] [, ...])]
[, содержимое, ...])
```

Выражение `xmlelement` создаёт XML-элемент с заданным именем, атрибутами и содержимым.

Примеры:

```
SELECT xmlelement(name foo);
```

```
xmlelement
```

```
-----
<foo/>
```

```
SELECT xmlelement(name foo, xmlattributes('xyz' as bar));
```

```

xmlelement
-----
<foo bar="xyz"/>

SELECT xmlelement(name foo, xmlattributes(current_date as bar),
'cont', 'ent');

xmlelement
-----
<foo bar="2007-01-26">content</foo>

```

Если имена элементов и атрибутов содержат символы, недопустимые в XML, эти символы заменяются последовательностями `_xNNNN_`, где `NNNN` — шестнадцатеричный код символа в Unicode. Например:

```

SELECT xmlelement(name "foo$bar", xmlattributes('xyz' as "a&b"));

xmlelement
-----
<foo_x0024_bar a_x0026_b="xyz"/>

```

Если в качестве значения атрибута используется столбец таблицы, имя атрибута можно не указывать явно, этим именем станет имя столбца. Во всех остальных случаях имя атрибута должно быть определено явно. Таким образом, это выражение допустимо:

```

CREATE TABLE test (a xml, b xml);
SELECT xmlelement(name test, xmlattributes(a, b)) FROM test;

```

А следующие варианты — нет:

```

SELECT xmlelement(name test, xmlattributes('constant'), a, b) FROM
test;
SELECT xmlelement(name test, xmlattributes(func(a, b))) FROM test;

```

Содержимое элемента, если оно задано, будет форматировано согласно его типу данных. Когда оно само имеет тип `xml`, из него можно конструировать сложные XML-документы. Например:

```
SELECT xmlelement(name foo, xmlattributes('xyz' as bar),
                  xmlelement(name abc),
                  xmlcomment('test'),
                  xmlelement(name xyz));
```

xmlelement

<foo bar="xyz"><abc/><!--test--><xyz/></foo>

Содержимое других типов будет оформлено в виде блока символьных данных XML. Это, в частности, означает, что символы <, > и & будут преобразованы в сущности XML. Двоичные данные (данные типа `bytea`) представляются в кодировке `base64` или в шестнадцатеричном виде, в зависимости от значения параметра `xmlbinary`. Следует ожидать, что конкретные представления отдельных типов данных могут быть изменены для приведения типов SQL и PostgreSQL в соответствие со стандартом XML Schema, когда появится его более полное описание.

2.5.1.4. Последовательность XML-элементов

```
xmlforest(содержимое [AS имя] [, ...])
```

Выражение `xmlforest` создаёт последовательность XML-элементов с заданными именами и содержимым.

Примеры:

```
SELECT xmlforest('abc' AS foo, 123 AS bar);
```

xmlforest

<foo>abc</foo><bar>123</bar>

```
SELECT xmlforest(table_name, column_name)
FROM information_schema.columns
WHERE table_schema = 'pg_catalog';
```

xmlforest

```
<table_name>pg_authid</table_name><column_name>rolname</column_name>
table_name>pg_authid</table_name><column_name>rolsuper</column_name>
...
```

Как показано во втором примере, имя элемента можно опустить, если источником содержимого служит столбец (в этом случае именем элемента по умолчанию будет имя столбца). В противном случае это имя необходимо указывать.

2.5.1.5. Корневой узел XML-значения

```
xmlroot(xml, version текст | нет значения [, standalone yes|no|нет
значения])
```

Выражение `xmlroot` изменяет свойства корневого узла XML-значения. Если в нём указывается версия, она заменяет значение в объявлении версии корневого узла; также в корневой узел переносится значение свойства `standalone`.

```
SELECT xmlroot(xmlparse(document '<?xml version="1.1"?
><content>abc</content>'),
           version '1.0', standalone yes);

           xmlroot
-----
<?xml version="1.0" standalone="yes"?>
<content>abc</content>
```

2.5.1.6. Агрегатный вызов

```
xmlagg(xml)
```

Функция `xmlagg`, в отличие от других описанных здесь функций, является агрегатной. Она соединяет значения, поступающие на вход агрегатной функции, подобно функции `xmlconcat`, но делает это, обрабатывая множество строк, а не несколько выражений в одной строке.

Пример:

```
CREATE TABLE test (y int, x xml);
INSERT INTO test VALUES (1, '<foo>abc</foo>');
```

```
INSERT INTO test VALUES (2, '<bar/>');
SELECT xmlagg(x) FROM test;
      xmlagg
-----
<foo>abc</foo><bar/>
```

Чтобы задать порядок сложения элементов, в агрегатный вызов можно добавить предложение ORDER BY. Например:

```
SELECT xmlagg(x ORDER BY y DESC) FROM test;
      xmlagg
-----
<bar/><foo>abc</foo>
```

Следующий нестандартный подход рекомендовался в предыдущих версиях и может быть по-прежнему полезен в некоторых случаях:

```
SELECT xmlagg(x) FROM (SELECT * FROM test ORDER BY y DESC) AS tab;
      xmlagg
-----
<bar/><foo>abc</foo>
```

2.5.2. Обработка XML

Для обработки значений типа xml с помощью выражений XPath 1.0 в PostgreSQL представлены функции xpath и xpath_exists [7].

```
xpath(xpath, xml [, nsarray])
```

Функция xpath вычисляет выражение XPath (аргумент xpath типа text) для заданного xml. Она возвращает массив XML-значений с набором узлов, полученных при вычислении выражения XPath. Если выражение XPath выдаёт не набор узлов, а скалярное значение, возвращается массив из одного элемента.

Вторым аргументом должен быть правильно оформленный XML-документ. В частности, в нём должен быть единственный корневой элемент.

В необязательном третьем аргументе функции передаются сопоставления пространств имён. Эти сопоставления должны определяться в двумерном массиве типа text, во второй размерности которого 2 элемента (т. е. это должен быть массив

массивов, состоящих из 2 элементов). В первом элементе каждого массива определяется псевдоним (префикс) пространства имён, а во втором — его URI. Псевдонимы, определённые в этом массиве, не обязательно должны совпадать с префиксами пространств имён в самом XML-документе (другими словами, для XML-документа и функции `xpath` псевдонимы имеют локальный характер).

Пример:

```
SELECT xpath('/my:a/text()', '<my:a
xmlns:my="http://example.com">test</my:a>',
            ARRAY[ARRAY['my', 'http://example.com']]);

xpath
-----
{test}
(1 row)
```

Для пространства имён по умолчанию (анонимного) это выражение можно записать так:

```
SELECT xpath('//mydefns:b/text()', '<a
xmlns="http://example.com"><b>test</b></a>',
            ARRAY[ARRAY['mydefns', 'http://example.com']]);

xpath
-----
{test}
(1 row)
```

```
xpath_exists(xpath, xml [, nsarray])
```

Функция `xpath_exists` представляет собой специализированную форму функции `xpath`. Она возвращает не весь набор XML-узлов, удовлетворяющих выражению XPath, а только одно логическое значение, показывающее, есть ли такие узлы. Эта функция равнозначна стандартному условию `XMLEXISTS`, за исключением того, что она также поддерживает сопоставления пространств имён.

Пример:

```
SELECT xpath_exists('/my:a/text()', '<my:a
xmlns:my="http://example.com">test</my:a>',
```

```
ARRAY[ARRAY['my', 'http://example.com']]);
```

```
xpath_exists
```

```
-----
```

```
t
```

```
(1 row)
```

2.5.3. Отображение таблиц в XML

Следующие функции отображают содержимое реляционных таблиц в значения XML [7]. Их можно рассматривать как средства экспорта в XML:

```
table_to_xml(tbl regclass, nulls boolean, tableforest boolean,  
targetns text)  
query_to_xml(query text, nulls boolean, tableforest boolean, targetns  
text)  
cursor_to_xml(cursor refcursor, count int, nulls boolean,  
tableforest boolean, targetns text)
```

Результат всех этих функций имеет тип xml.

`table_to_xml` отображает в xml содержимое таблицы, имя которой задаётся в параметре `tbl`. Тип `regclass` принимает идентификаторы строк в обычной записи, которые могут содержать указание схемы и кавычки. Функция `query_to_xml` выполняет запрос, текст которого передаётся в параметре `query`, и отображает в xml результирующий набор. Последняя функция, `cursor_to_xml` выбирает указанное число строк из курсора, переданного в параметре `cursor`. Этот вариант рекомендуется использовать с большими таблицами, так как все эти функции создают результирующий xml в памяти.

Если параметр `tableforest` имеет значение `false`, результирующий XML-документ выглядит так:

```
<имя_таблицы>  
  <row>  
    <имя_столбца1> данные </имя_столбца1>  
    <имя_столбца2> данные </имя_столбца2>  
  </row>
```

```

    <row>
        ...
    </row>

    ...
</имя_таблицы>

```

А если `tableforest` равен `true`, в результате будет выведен следующий фрагмент XML:

```

<имя_таблицы>
    <имя_столбца1> данные </имя_столбца1>
    <имя_столбца2> данные </имя_столбца2>
</имя_таблицы>

<имя_таблицы>
    ...
</имя_таблицы>

...

```

Если имя таблицы неизвестно, например, при отображении результатов запроса или курсора, вместо него в первом случае вставляется `table`, а во втором — `row`.

Параметр `nulls` определяет, нужно ли включать в результат значения `NULL`. Если он установлен, значения `NULL` в столбцах представляются так:

```

<имя_столбца xsi:nil="true"/>

```

Здесь `xsi` — префикс пространства имён XML Schema Instance. При этом в результирующий XML будет добавлено соответствующее объявление пространства имён. Если же данный параметр равен `false`, столбцы со значениями `NULL` просто не будут выводиться.

Параметр `targetns` определяет целевое пространство имён для результирующего XML. Если пространство имён не нужно, значением этого параметра должна быть пустая строка.

Следующие функции выдают документы XML Schema, которые содержат схемы отображений, выполняемых соответствующими ранее рассмотренными функциями:


```
table_to_xmlschema(tbl regclass, nulls boolean, tableforest boolean,  
    targetns text)  
query_to_xmlschema(query text, nulls boolean, tableforest boolean,  
    targetns text)  
cursor_to_xmlschema(cursor refcursor, nulls boolean, tableforest  
boolean,  
    targetns text)
```

Чтобы результаты отображения данных в XML соответствовали XML-схемам, важно, чтобы паре функций передавались одинаковые параметры.

Следующие функции выдают отображение данных в XML и соответствующую XML-схему в одном документе (или фрагменте), объединяя их вместе. Это может быть полезно там, где желательно получить самодостаточные результаты с описанием:

```
table_to_xml_and_xmlschema(tbl regclass, nulls boolean, tableforest  
boolean,  
    targetns text)  
query_to_xml_and_xmlschema(query text, nulls boolean, tableforest  
boolean,  
    targetns text)
```

В дополнение к ним есть следующие функции, способные выдать аналогичные представления для целых схем в базе данных или даже всей текущей базы данных:

```
schema_to_xml(schema name, nulls boolean, tableforest boolean,  
    targetns text)  
schema_to_xmlschema(schema name, nulls boolean, tableforest boolean,  
    targetns text)  
schema_to_xml_and_xmlschema(schema name, nulls boolean, tableforest  
boolean,  
    targetns text)  
  
database_to_xml(nulls boolean, tableforest boolean, targetns text)  
database_to_xmlschema(nulls boolean, tableforest boolean, targetns  
text)  
database_to_xml_and_xmlschema(nulls boolean, tableforest boolean,  
    targetns text)
```

2.6. Пример работы с XML в PostgreSQL

2.6.1. Создание таблиц

Приведенный ниже код создает две таблицы данных.

```
CREATE TABLE org
(
    oname varchar(20) PRIMARY KEY,
    city varchar(25),
    otype varchar(10) CHECK (otype in ('OOO', 'ZAO', 'OAO', 'Trest'))
);

create table Worker
(
    inn int primary key,
    fio varchar(50) not null,
    age int default 18,
    addr varchar(30) default 'Mos',
    working varchar(20) references org(oname)
    on update cascade on delete set null
);
```

Таблицы были наполнены данными. Рисунок 2 и Рисунок 3 демонстрируют содержимое заполненных таблиц.


	oname [PK] character v	city character v	otype character v
1	Antey	Mos	OAO
2	MGTU	Mos	Trest
3	RZD	RF	OAO
4	Stankin	Mos	OOO
5	Zenit	SPB	OOO
*			

Рисунок 2 – Содержимое таблицы org

	inn [PK] integer	fio character v	age integer	addr character v	working character v
1	2222	Semenova	18	Mos	Stankin
2	3333	Kurov	40	Mos	MGTU
3	4444	Udina	30	SPB	Zenit
4	5555	Usov	55	Vladimir	MGTU
5	11111	Petrov	18	Tver	
6	12345	Ivanov I.	20	Mos	Antey
*					

Рисунок 3 – Содержимое таблицы Worker

2.6.2. Экспорт таблиц и запросов в XML

Экспорт выполняется в окне запросов. Для записи результата в файл выбрать кнопку  в окне запросов и указать настройки экспорта (см. Рисунок 4).

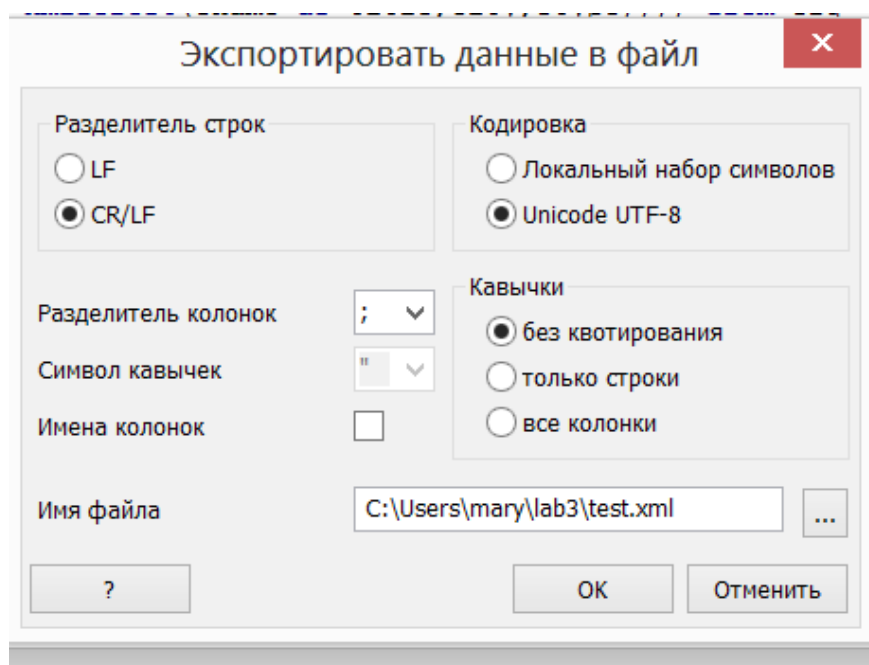


Рисунок 4 - Окно экспорта данных

Также можно использовать команду `copy` следующим образом:

```
copy (
select ...
) to 'd:/test.xml'
```

2.6.2.1. Экспорт таблицы

Выполним экспорт таблицы Worker, созданной ранее:

```
select table_to_xml('worker',true,true,'') ;
```

В результате получим следующее содержимое файла:

```
<worker xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <inn>12345</inn>
  <fio>Ivanov I.I.</fio>
  <age>20</age>
  <addr>Mos</addr>
  <working>Antey</working>
</worker>

<worker xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <inn>3333</inn>
  <fio>Kurov</fio>
  <age>40</age>
  <addr>Mos</addr>
  <working>MGТУ</working>
</worker>

<worker xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <inn>4444</inn>
  <fio>Udina</fio>
  <age>30</age>
  <addr>SPB</addr>
  <working>Zenit</working>
</worker>

<worker xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <inn>11111</inn>
  <fio>Petrov</fio>
  <age>18</age>
  <addr>Tver</addr>
  <working xsi:nil="true"/>
</worker>

<worker xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <inn>2222</inn>
```

```

    <fio>Semenova</fio>
    <age>18</age>
    <addr>Mos</addr>
    <working>Stankin</working>
  </worker>

  <worker xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <inn>5555</inn>
    <fio>Usov</fio>
    <age>55</age>
    <addr>Vladimir</addr>
    <working>MGТУ</working>
  </worker>

```

2.6.2.2. Экспорт результата запроса

Выполним экспорт результатов запроса, в котором отберем только сотрудников МГТУ:

```

select query_to_xml('select * from worker where
working='MGТУ'',true,true,'k') ;

```

Рисунок 5 демонстрирует содержимое полученного файла.

```

<?xml version="1.0"?>
- <table xmlns="k" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  - <row>
    <inn>3333</inn>
    <fio>Kurov</fio>
    <age>40</age>
    <addr>Mos</addr>
    <working>MGТУ</working>
  </row>
  - <row>
    <inn>5555</inn>
    <fio>Usov</fio>
    <age>55</age>
    <addr>Vladimir</addr>
    <working>MGТУ</working>
  </row>
</table>

```

Рисунок 5 - Содержимое файла XML

Также в настройках экспорта возможно добавление схемы, корневого элемента, экспорт схемы БД или всей БД [7].

2.6.3. Создание XML на основе таблиц

2.6.3.1. Создание элементов из строк таблиц, поле как содержимое

Следующий код извлекает строки из таблицы и создает XML-элементы, содержимое которых отражает значения полей таблицы.

```
select xmlelement(name org, oname) from org

"<org>Antey</org>"
"<org>RZD</org>"
"<org>MG TU</org>"
"<org>Stankin</org>"
"<org>Zenit</org>"
```

2.6.3.2. Создание элементов из строк таблиц, поле как атрибут

Следующий код извлекает строки из таблицы и создает XML-элементы, в которых поля таблицы передаются как атрибуты элементов.

```
select xmlelement(name org, xmlattributes(oname,city,otype)) from org

"<org oname="Antey" city="Mos" otype="OAO"/>"
"<org oname="RZD" city="RF" otype="OAO"/>"
"<org oname="MG TU" city="Mos" otype="Trest"/>"
"<org oname="Stankin" city="Mos" otype="000"/>"
"<org oname="Zenit" city="SPB" otype="000"/>"
```

2.6.3.3. Создание элементов из строк таблиц, переименование элемента

Следующий код извлекает строки из таблицы и создает XML-элементы, в которых поля таблицы передаются как атрибуты элементов, а также выполняется переименование элемента.

```
select xmlelement(name MyOrg, xmlattributes(oname as
title,city,otype)) from org

"<myorg title="Antey" city="Mos" otype="OAO"/>"
"<myorg title="RZD" city="RF" otype="OAO"/>"
"<myorg title="MG TU" city="Mos" otype="Trest"/>"
```

```
"<myorg title="Stankin" city="Mos" otype="000"/>"
"<myorg title="Zenit" city="SPB" otype="000"/>"
```

2.6.3.4. Создание элементов из строк таблиц, поле как дочерний элемент

Следующий код извлекает строки из таблицы и создает XML-элементы, в которых поля таблицы передаются как дочерние атрибуты элементов.

```
select xmlelement(name MyOrg, xmlelement(name title, oname),
xmlelement(name city, city), xmlelement(name type, otype)) from org

"<myorg><title>Antey</title><city>Mos</city><type>0A0</type></myorg>"
"<myorg><title>RZD</title><city>RF</city><type>0A0</type></myorg>"
"<myorg><title>MGTU</title><city>Mos</city><type>Trest</type></
myorg>"
"<myorg><title>Stankin</title><city>Mos</city><type>000</type></
myorg>"
"<myorg><title>Zenit</title><city>SPB</city><type>000</type></myorg>"
```

2.6.3.5. Получение последовательности элементов

Следующий код извлекает строки из таблицы и создает последовательность XML-элементов.

```
select xmlforest(oname,city,otype) from org

"<oname>Antey</oname><city>Mos</city><otype>0A0</otype>"
"<oname>RZD</oname><city>RF</city><otype>0A0</otype>"
"<oname>MGTU</oname><city>Mos</city><otype>Trest</otype>"
"<oname>Stankin</oname><city>Mos</city><otype>000</otype>"
"<oname>Zenit</oname><city>SPB</city><otype>000</otype>"
```

2.6.3.6. Получение последовательности элементов с дочерними элементами и переименованием

Следующий код извлекает строки из таблицы и создает последовательность XML-элементов, в которых поля таблицы передаются как дочерние элементы.

```

select xmlelement(name MyOrg, xmlforest(oname as title,city,otype))
from org

"<myorg><title>Antey</title><city>Mos</city><otype>OA0</otype></
myorg>"
"<myorg><title>RZD</title><city>RF</city><otype>OA0</otype></myorg>"
"<myorg><title>MGTU</title><city>Mos</city><otype>Trest</otype></
myorg>"
"<myorg><title>Stankin</title><city>Mos</city><otype>000</otype></
myorg>"
"<myorg><title>Zenit</title><city>SPB</city><otype>000</otype></
myorg>"

```

2.6.3.7. Добавление корневого элемента

Следующий код извлекает строки из таблицы и создает XML-элементы с добавлением корневого элемента.

```

select xmlroot( xmlelement(name MyOrg, xmlforest(oname as
title,city,otype)), version '1.1', standalone yes) from org

"<?xml version="1.1"
standalone="yes"?><myorg><title>Antey</title><city>Mos</city><otype>OA0</
otype></myorg>"
"<?xml version="1.1"
standalone="yes"?><myorg><title>RZD</title><city>RF</city><otype>OA0</
otype></myorg>"
"<?xml version="1.1"
standalone="yes"?><myorg><title>MGTU</title><city>Mos</city><otype>Trest</
otype></myorg>"
"<?xml version="1.1"
standalone="yes"?><myorg><title>Stankin</title><city>Mos</city><otype>000<
/otype></myorg>"
"<?xml version="1.1"
standalone="yes"?><myorg><title>Zenit</title><city>SPB</city><otype>000</
otype></myorg>"

```


2.6.3.8. Упаковка всех строк в один документ

Следующий код извлекает все строки из таблицы и создает один XML-документ, содержащий все строки (поля передаются как дочерние элементы). Рисунок 6 демонстрирует полученный XML-документ.

```
select xmlelement(name root, xmlagg( xmlelement(name MyOrg,
xmlforest(oname as title,city,otype)))) from org
```

```
<?xml version="1.0"?>
- <root>
  - <myorg>
    <title>Antey</title>
    <city>Mos</city>
    <otype>OAO</otype>
  </myorg>
  - <myorg>
    <title>RZD</title>
    <city>RF</city>
    <otype>OAO</otype>
  </myorg>
  - <myorg>
    <title>MGTU</title>
    <city>Mos</city>
    <otype>Trest</otype>
  </myorg>
  - <myorg>
    <title>Stankin</title>
    <city>Mos</city>
    <otype>OOO</otype>
  </myorg>
  - <myorg>
    <title>Zenit</title>
    <city>SPB</city>
    <otype>OOO</otype>
  </myorg>
</root>
```

Рисунок 6 - Содержимое XML-документа

2.6.3.9. Создание документа с произвольной структурой

Следующий код извлекает строки из таблицы, и создает XML-документ с произвольной структурой. Рисунок 7 демонстрирует полученный XML-документ.

```
copy (
  select xmlelement(name mydbs, xmlagg(xmlelement(name MyOrg,
xmlattributes(oname as title,city,otype),
  xmlconcat( (select xmlagg(xmlelement(name person,
xmlattributes(inn,age), fio ))
  from worker where working=oname)) ) )) from org
) to 'd:/test1.xml'
```

```

<?xml version="1.0"?>
- <mydbs>
  - <myorg title="Antey" otype="OAO" city="Mos">
    <person age="20" inn="12345">Ivanov I.I.</person>
  </myorg>
  <myorg title="RZD" otype="OAO" city="RF"/>
  - <myorg title="MGTU" otype="Trest" city="Mos">
    <person age="40" inn="3333">Kurov</person>
    <person age="55" inn="5555">Usov</person>
  </myorg>
  - <myorg title="Stankin" otype="OOO" city="Mos">
    <person age="18" inn="2222">Semenova</person>
  </myorg>
  - <myorg title="Zenit" otype="OOO" city="SPB">
    <person age="30" inn="4444">Udina</person>
  </myorg>
</mydbs>

```

Рисунок 7 - Содержимое XML-документа с произвольной структурой

2.6.4. Примеры запросов к XML-документу

Чтение XML из файла выполняется функцией `pg_read_file` (относительно каталога PGDATA). Следующий код преобразует строковые данные из файла в XML:

```
xmlparse(DOCUMENT pg_read_file('test1.xml'))
```

2.6.4.1. Проверка существования документа

Проверка существования документа выполняется следующим образом:

```
select xpath_exists('//person[@age=18 and ../@city='Mos'],'',
xmlparse(DOCUMENT pg_read_file('test1.xml')) ) ;
```

2.6.4.2. Получить названия организаций

Выведем все названия организаций:

```
select unnest(xpath('//myorg/@title', xmlparse(DOCUMENT
pg_read_file('test1.xml'))));
```

```

"Antey"
"RZD"
"MGTU"
"Stankin"
"Zenit"

```

2.6.4.3. Получить ФИО сотрудников

Выведем все ФИО сотрудников:

```
select unnest(xpath('//person/text()', xmlparse(DOCUMENT
pg_read_file('test1.xml'))));
```

2.6.4.4. Получить ФИО сотрудников МГТУ

Выведем ФИО только сотрудников из МГТУ:

```
select unnest(xpath('//person[../@title='MGTU']/text()',
xmlparse(DOCUMENT pg_read_file('test1.xml'))));
```

2.6.4.5. Получить количество сотрудников МГТУ

Получим количество сотрудников, работающих в МГТУ:

```
select xpath('count(//myorg[@title='MGTU']/person)',
xmlparse(DOCUMENT pg_read_file('test1.xml'))) AS w_cnt;
```

2.6.4.6. Количество сотрудников в организациях

Выведем количество сотрудников в каждой организации:

```
select xpath('@title',a) as org, xpath('count(person)',a) as cnt
from unnest(xpath('//myorg', xmlparse(DOCUMENT
pg_read_file('test1.xml')))) as a;
```

```
"{Antey}"; "{1}"
```

```
"{RZD}"; "{0}"
```

```
"{MGTU}"; "{2}"
```

```
"{Stankin}"; "{1}"
```

```
"{Zenit}"; "{1}"
```

3. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое XML формат и каковы его особенности?
2. Какими средствами можно работать с XML?
3. Как задать сохранение свойства в виде элемента, атрибута или текста?
4. Какие функции и методы позволяют выполнять импорт данных из XML?
5. Какие функции и методы позволяют выполнять экспорт данных в формате XML? Для таблиц и для запросов?
6. Назначение и синтаксис языка XPath.
7. Навигация и условные выражения в XPath.
8. Назначение и синтаксис языка XQuery.
9. Конструкции FLWOR в XQuery.
10. Операторы for, let, order by, return, where в XQuery.
11. Создание атрибутов и элементов XML.

4. СПИСОК ИСТОЧНИКОВ

1. Виноградов В.И., Виноградова М.В. Постреляционные модели данных и языки запросов: Учебное пособие. — М.: Изд-во МГТУ им. Н.Э. Баумана, 2017. — 100с. - ISBN 978-5-7038-4283-6.
2. PostgreSQL 14.2 Documentation. — Текст. Изображение: электронные // PostgreSQL : [сайт]. — URL: <https://www.postgresql.org/docs/14/index.html> (дата обращения: 28.04.2022)
3. pgAdmin 4 6.5 documentation. — Текст. Изображение: электронные // pgAdmin - PostgreSQL Tools : [сайт]. — URL: <https://www.pgadmin.org/docs/pgadmin4/6.5/index.html> (дата обращения: 26.04.2022)
4. Extensible Markup Language (XML) 1.0 (Fifth Edition). — Текст. Изображение : электронные // World Wide Web Consortium (W3C) : [сайт]. — URL: <https://www.w3.org/TR/xml/> (дата обращения: 28.04.2022)
5. XML Path Language (XPath) 3.1. — Текст. Изображение : электронные // World Wide Web Consortium (W3C) : [сайт]. — URL: <https://www.w3.org/TR/xpath-31/> (дата обращения: 28.04.2022)
6. PostgreSQL: Документация: 14: 8.13. Тип XML. — Текст. Изображение: электронные // Компания Postgres Professional: [сайт]. — URL: <https://postgrespro.ru/docs/postgresql/14/datatype-xml> (дата обращения: 28.04.2022)
7. PostgreSQL: Документация: 14: 9.15. XML-функции. — Текст. Изображение: электронные // Компания Postgres Professional: [сайт]. — URL: <https://postgrespro.ru/docs/postgresql/14/functions-xml> (дата обращения: 28.04.2022)