**ПРАКТИЧЕСКАЯ РАБОТА № 1.**

**РАЗРАБОТКА РЕЛЯЦИОННОЙ МОДЕЛИ ДАННЫХ**

**Цель работы**: научиться средствами СУБД разрабатывать базу данных и работать с ее объектами.

**Методические указания**

В данной работе разработка реляционных баз данных будет рассматриваться на примере СУБД PostgreSQL.

PostgreSQL — это мощная объектно-реляционная СУБД с открытым исходным кодом. PostgreSQL поддерживает базовую спецификацию SQL99. В данной СУБД используется клиент-серверная архитектура с распределением процессов между пользователями.

Разработка и последующая работа с базой данных может выполняться путем запуска терминальной программы **psql**, которая позволяет выполнять SQL-команды, либо с использованием инструмента управления базой данных с графическим интерфейсом **pgAdmin.**

Рассмотрим пример разработки БД, содержащей информацию о студентах и посещаемых ими занятиях с использованием интерфейса pgAdmin.

***1. Создание базы данных***

Создать БД можно посредством функций рабочей среды либо с использованием команд SQL, при этом необходимо задать БД определенные параметры.

***2. Создание таблиц базы данных***

Перед созданием таблиц необходимо создать схему.

Схемы можно использовать для разграничения прав пользователей, так как они обеспечивают дополнительный уровень распределения прав. В БД уже существует схема public по умолчанию, в которую можно добавлять общие таблицы.

После создания схемы необходимо добавить таблицы, при этом для каждого столбца необходимо задать тип данных.

PostgreSQL представляет большой набор встроенных типов данных, а также позволяет создать свой собственный тип посредством использования команды [CREATE TYPE](https://postgrespro.ru/docs/postgresql/9.4/sql-createtype).

Встроенные типы данных представлены в таблице 1.

Таблица 1 – Типы данных PostgreSQL

|  |  |  |
| --- | --- | --- |
| **Тип данных** | **Псевдонимы** | **Описание** |
| bigint | int8 | знаковое целое из 8 байт |
| bigserial | serial8 | восьмибайтное целое с автоувеличением |
| bit [(*n*)] |  | битовая строка фиксированной длины |
| bit varying [(*n*)] | varbit | битовая строка переменной длины |
| boolean | bool | логическое значение (true/false) |
| box |  | прямоугольник в плоскости |
| bytea |  | двоичные данные ("массив байт") |
| character [(*n*)] | char [(*n*)] | символьная строка фиксированной длины |
| character varying [(*n*)] | varchar [(*n*)] | символьная строка переменной длины |
| cidr |  | сетевой адрес IPv4 или IPv6 |
| circle |  | круг в плоскости |
| date |  | календарная дата (год, месяц, день) |
| double precision | float8 | число двойной точности с плавающей точкой (8 байт) |
| inet |  | адрес узла IPv4 или IPv6 |
| integer | int, int4 | знаковое четырёхбайтное целое |
| interval [*поля*] [(*p*)] |  | интервал времени |
| json |  | текстовые данные JSON |
| jsonb |  | двоичные данные JSON, разобранные |
| line |  | прямая в плоскости |
| lseg |  | отрезок в плоскости |
| macaddr |  | MAC-адрес |
| money |  | денежная сумма |
| numeric [(*p*, *s*)] | decimal [(*p*, *s*)] | вещественное число заданной точности |
| path |  | геометрический путь в плоскости |
| pg\_lsn |  | последовательный номер в журнале PostgreSQL |
| point |  | геометрическая точка в плоскости |
| polygon |  | замкнутый геометрический путь в плоскости |
| real | float4 | число одинарной точности с плавающей точкой (4 байта) |
| smallint | int2 | знаковое двухбайтное целое |
| smallserial | serial2 | двухбайтное целое с автоувеличением |
| serial | serial4 | четырёхбайтное целое с автоувеличением |
| text |  | символьная строка переменной длины |
| time [(*p*)] [without time zone] |  | время суток (без часового пояса) |
| time [(*p*)] with time zone | timetz | время суток с учётом часового пояса |
| timestamp [(*p*)] [without time zone] |  | дата и время (без часового пояса) |
| timestamp [(*p*)] with time zone | timestamptz | дата и время с учётом часового пояса |
| tsquery |  | запрос текстового поиска |
| tsvector |  | документ для текстового поиска |
| txid\_snapshot |  | снимок идентификатора транзакций |
| uuid |  | универсальный уникальный идентификатор |
| xml |  | XML-данные |

Столбцы и типы данных можно задать на этапе создания таблицы либо добавить их после.

***3. Введение ограничений целостности***

Целостность БД подразумевает соответствие информации в БД ее внутренней логике, заданным правилам и структуре.

Ограничения целостности – это специальные средства или утверждения, которые не допускают попадания в БД ошибочных данных (прим. тринадцатый месяц в году). Ограничения могут быть определены при создании таблицы (для каждого столбца в отдельности) или добавлены в таблицу позже с помощью специальной команды SQL.

Рассмотрим ограничения на значения столбцов:

1. Первичный ключ (primary key) – обеспечивает уникальность данных в столбцах, также используется для обеспечения ссылок на другие таблицы (посредством использования внешних ключей).

Первичный ключ для каждой строки таблицы должен быть уникальным и ненулевым.

1. Внешний ключ (foreign key) – применяется с определенным ранее первичным ключом либо ограничением уникальности в связанной таблице. При этом условие на значение внешнего ключа ставит в соответствие один или несколько столбцов таблицы идентичному набору столбцов другой таблицы, для которых определенно ограничение первичного ключа или уникальности.

Ограничение на значение столбцов первичного и внешнего ключа обеспечивают соответствие строк связанных таблиц, поэтому столбцы с данными ограничениями используются для реализации операций соединения таблиц.

Наглядное представление о структуре связей между таблицами в базе данных можно получить с помощью диаграмм «таблица-связь», на которых указываются ограничения primary key и foreign key и такая характеристика связей, как кардинальность связи.

1. Ограничение уникальности (unique) – позволяет запретить повторение значений в столбце таблицы. Не действует на столбец с ограничением первичного ключа, так как уникальный индекс данного столбца уже создан.

Столбец с ограничением уникальности может быть использован для связи с другими таблицами.

1. Проверочное значение (check) – устанавливает возможные в столбце значения. Может использоваться для определения диапазона значения, который может храниться в столбце.

***4. Ввод данных в таблицы***

Для ввода данных можно воспользоваться соответствующей командой редактирования таблицы либо с помощью операции INSERT языка SQL.

При вводе данных необходимо учитывать введенные ранее ограничения:

* значения внешнего ключа должны соответствовать значениям, на которые внешние ключи ссылаются;
* значение по умолчанию заполняется, если не введено другое значение и др.

Добавление вручную осуществляется путем редактирования данных, при этом, если для ключа было указано значение типа счетчик (serial), то его не обязательно заполнять вручную.

При добавлении данных с помощью команды INSERT важно соблюдать порядок следования столбцов.

**Порядок выполнения работы**

1. Установить СУБД PostgreSQL (postgresql.org​)
2. Проанализировать предметную область согласно выбранному варианту. При анализе определить необходимые для отображения информационной системы таблицы, столбцы и типы данных, и ограничения целостности.
3. Создать БД согласно варианту.
4. Ввести тестовые данные в созданные таблицы.

**Контрольные вопросы**

1. Чем реляционные базы данных отличаются от нереляционных?
2. Реляционная модель данных.
3. Классификация СУБД.
4. Клиент-серверная СУБД.
5. Ограничения целостности.
6. Что такое первичный ключ и его ограничение?
7. Что такое внешний ключ и его ограничение?

**ПРАКТИЧЕСКАЯ РАБОТА № 2.**

**ВЫПОЛНЕНИЕ ПРОСТЫХ SQL-ЗАПРОСОВ**

**Цель работы**: изучить язык запросов для выборки данных.

**Методические указания**

Для ввода и тестирования инструкций и операторов языка SQL можно использовать приложение Query Tool, входящее в комлект приложения PosgreSQL.

Для запуска Query Tool необходимо выбрать одноименную команду из меню Tools командного меню pgAdmin (рисунок 1).

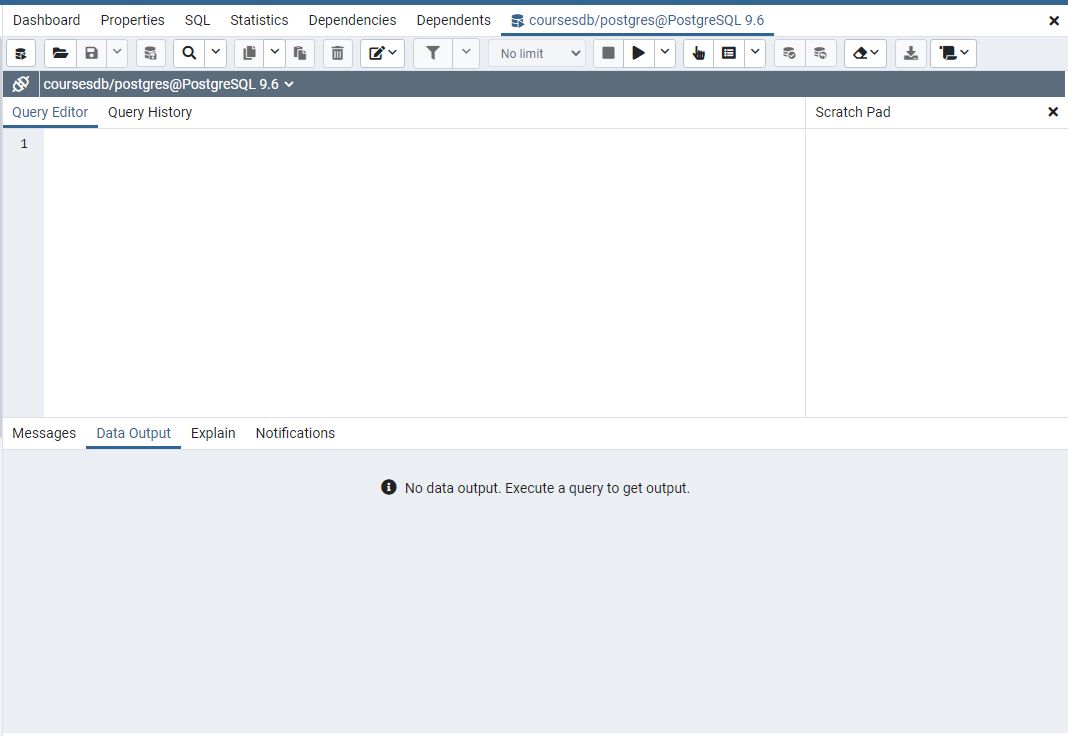


Рисунок 1 – Интерфейс Query Tool

Для выполнения простых выборок данных необходимо использовать оператор SELECT.

Оператор SELECT, в упрощенном виде, имеет следующую структуру:

**SELECT** [DISTINCT | ALL] select\_expression,...

**FROM** table\_references

[**WHERE** where\_definition]

[**GROUP BY** {unsigned\_integer | col\_name | formula}]

[**HAVING** where\_definition]

[**ORDER BY** {unsigned\_integer | col\_name | formula} [ASC | DESC], ...]

, где квадратные скобки ([]) обозначают необязательность выделенных в скобки конструкций.

Различные элементы данного оператора позволяют указать условия для выбора нужных данных и их обработки:

|  |  |
| --- | --- |
| SELECT | определяет результирующий набор столбцов |
| FROM | определяет базовое табличное выражение, которое используется предложениями оператора SELECT |
| WHERE | определяет условие, по которому отбираются результирующие строки |
| GROUP BY | группировка строк по результатам агрегатных функций |
| HAVING | отбор групп, получающихся в результате группировки |
| ORDER BY | сортировка результирующего набора строк |

**Порядок выполнения работы**

1. Наполнить базу данных согласно своему варианту задания тестовыми данными с учетом реальных пропорций количества строк в таблице. Минимальное количество строк в таблице - 3.

При этом в таблицах, имеющих связь с данной таблицей "один-ко-многим" должно быть соотношение количества строк 1:3.

Например, для БД Университета может быть три студенческих группы, в каждой не менее трех студентов. Для магазина - три категории товаров, в каждой не менее трех товаров.

1. Написать и отладить следующие виды SQL-запросов:

* Запрос на выбор всех данных из таблицы
* Запрос на выбор определенных полей (столбцов) из таблицы
* Запрос на выбор строк, соответствующих определенному условию WHERE с использованием операторов сравнения ( > < = )
* Запрос на выбор строк, соответствующих определенному условию WHERE с использованием BETWEEN
* Запрос на выбор строк, соответствующих определенному условию WHERE с использованием LIKE
* Запрос на выбор строк, соответствующих определенному условию WHERE с использованием IN
* Запрос с использованием сортировки строк результата ORDER BY
* Запрос с использованием ограничения на количество выводимых строк ORDER BY ... LIMIT
* Запрос, получающий декартово произведение связанных таблиц
* Запрос, получающий соединение связанных таблиц по ключу
* Запрос, получающий соединение таблицы со своей копией

**ПРАКТИЧЕСКАЯ РАБОТА № 3**

**ВЫПОЛНЕНИЕ SQL-ЗАПРОСОВ С ИСПОЛЬЗОВАНИЕМ АГРЕГИРУЮЩИХ ФУНКЦИЙ И ФУНКЦИИ ГРУППИРОВКИ**

**Цель работы**: изучить возможности агрегирующих функций и функций группировки.

**Методические указания**

Однородные в определенном отношении строки реляционной таблицы могут быть объединены в группы.

Для выполнения объединения данных по некоторому критерию используется модификатор GROUP BY, который делит таблицу на группы строк, каждая из которых имеет одинаковые значения в столбце, указанном в

GROUP BY.

Сама по себе группировка не имеет смысла группировка и обычно предшествует неким операциям, выполняемым над данными в группах.

В запросе на группировку записей должны быть выполнены следующие условия:

* группировка может выполняться по одному или нескольким столбцам;
* порядок группировки определяется порядком следования столбцов в списке GROUP BY;
* список полей группировки в предложении GROUP BY и список элементов предложения SELECT должны быть согласованы;
* в списке SELECT можно использовать только те поля, по которым выполняется группировка, а также вычисляемые поля, например на основе статистических функций: COUNT, SUM, AVG, MAX, MIN и др.

**Порядок выполнения работы**

Написать и отладить следующие виды SQL-запросов:

* Запрос на подсчет количества строк в таблице, удовлетворяющих заданному условию (COUNT)
* Запрос на подсчет среднего значения в каком-либо столбце таблицы (функция AVG)
* Запрос на подсчет суммы значений какого-либо столбца в таблице (функция SUM)
* Запрос на нахождение максимального значения в столбце таблицы (функция MAX)
* Запрос на нахождение минимального значения в столбце таблицы (функция MIN)
* Запрос на нахождение минимального значения в столбце таблицы бес использования функции MIN, применяя ORDER BY и LIMIT
* Запрос на нахождение максимального значения в столбце таблицы бес использования функции MAX, применяя ORDER BY DESC и LIMIT
* Запрос с группировкой строк и подсчетом значения любой агрегатной функции по каждой группе (GROUP BY)
* Запрос с соединением не менее, чем 2-х таблиц, группировкой строк и подсчетом значения любой агрегатной функции по каждой группе (GROUP BY)
* Запрос SELECT с использованием вложенного подзапроса SELECT
* Запрос INSERT с использованием вложенного подзапроса SELECT
* Запрос на объединение результатов запросов с использованием UNION
* Запрос на соединение таблиц с использованием JOIN (все виды)

**Контрольные вопросы**

1. Что такое агрегирующие функции и как они используются?
2. Как называются функции SUM, AVG, MAX, MIN, COUNT? Почему?
3. Какие требования необходимо выполнить для объединения двух запросов?
4. Как средствами SQL выполнить объединение, пересечение и разность двух таблиц?
5. Каковы особенности применения предложения UNION?
6. Какие виды JOIN реализованы в PostgreSQL?
7. Каковы особенности применения предложения JOIN?
8. Для чего предназначена конструкция WITH?

**ПРАКТИЧЕСКАЯ РАБОТА № 4**

**ПРЕДСТАВЛЕНИЕ, ПРОЦЕДУРЫ, ФУНКЦИИ, ТРИГГЕРЫ**

**Цель работы**: изучить определенные объекты баз данных, их возможности и научиться их разрабатывать.

**Методические указания**

PostgreSQL поддерживает объекты базы данных, которые реализуют исполняемый код, определяемый пользователем, например, представления, процедуры, функции и триггеры.

В рамках лабораторной работы необходимо изучить данные объекты, научиться их создавать и работать с данными объектами и результатами их выполнения.

*Представления*

Представление являет собой особый объект базы данных, который содержит данные, полученные запросом SELECT из другой таблицы или таблиц.

Упрощенный синтаксис создания представления выглядит следующим образом:

CREATE VIEW имя-представления [ ( имя-столбца [, ...] ) ]

AS запрос;

Имя представления и запрос в данной команде являются обязательными.

Представление упрощает обращение к БД, так как для работы с результатом какого-либо запроса можно обернуть его в представление и далее обращаться к этим результатам также, как к таблице:

SELECT \* FROM имя-представления

Однако в отличие от таблиц, представления не содержат данные – физически данные находятся в таблицах, к которым обращается представление.

*Хранимые процедуры*

Хранимая процедура – это объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере.

Создание процедуры в упрощенном виде осуществляется следующей командой:

CREATE [ OR REPLACE ] PROCEDURE

имя ( [ [ режим\_аргумента ] [ имя\_аргумента ] тип\_аргумента [ { DEFAULT | = } выражение\_по\_умолчанию ] [, ...] ] )

{ LANGUAGE имя\_языка

| TRANSFORM { FOR TYPE имя\_типа } [, ... ]

| AS 'определение'

} ...

, где имя – имя процедуры;

режим аргумента - IN (входной), INOUT (входной и выходной) или VARIADIC (переменный). По умолчанию подразумевается IN;

имя\_аргумента – имя аргумента, которое может использоваться в теле функции;

тип*\_*аргумента – тип данных аргумента;

выражение\_по\_умолчанию - выражение, используемое для вычисления значения по умолчанию, если параметр не задан явно;

имя\_языка – язык, на котором реализована функция (SQL, C и др.);

TRANSFORM { FOR TYPE имя\_типа } [, ... ] } – установка списка трансформации (при приведении типов SQL к типам данных других языков);

определение – реализация процедуры.

Вызов процедуры осуществляется с помощью инструкции CALL:

CALL имя([параметры])

*Функции*

Пользовательская функция - отличается от хранимой процедуры, тем что хранимая функция в общем случае возвращает только скаляр (то есть единичное значение), а процедура может возвращать набор значений.

Упрощенный синтаксис функции:

CREATE [ OR REPLACE ] FUNCTION

имя ( [ [ режим\_аргумента ] [ имя\_аргумента ] тип\_аргумента [ { DEFAULT | = } выражение\_по\_умолчанию ] [, ...] ] )

[ RETURNS тип\_результата

| RETURNS TABLE ( имя\_столбца тип\_столбца [, ...] ) ]

{ LANGUAGE имя\_языка

| TRANSFORM { FOR TYPE имя\_типа } [, ... ]

| WINDOW

| ROWS строк\_в\_результате

| SET параметр\_конфигурации { TO значение | = значение | FROM CURRENT }

| AS 'определение'

} ...

, где имя – имя функции;

режим аргумента - IN (входной), OUT (выходной), INOUT (входной и выходной) или VARIADIC (переменный). По умолчанию подразумевается IN;

имя\_аргумента – имя аргумента, которое может использоваться в теле функции;

тип*\_*аргумента – тип данных аргумента;

выражение\_по\_умолчанию - выражение, используемое для вычисления значения по умолчанию, если параметр не задан явно;

тип\_результата – тип возвращаемых данных;

имя\_столбца – имя выходного столбца;

тип\_столбца – тип данных, выходного столбца;

имя\_языка – язык, на котором реализована функция (SQL, C и др.);

TRANSFORM { FOR TYPE имя\_типа } [, ... ] } – установка списка трансформации (при приведении типов SQL к типам данных других языков);

WINDOW – указание для создания оконных функций (для функций на языке C);

определение – реализация функции.

*Триггеры*

Триггер – это хранимая процедура, которую пользователь не вызывает непосредственно, а исполнение которой обусловлено действием по модификации данных.

Упрощенный синтаксис создания триггера выглядит следующим образом:

CREATE [ CONSTRAINT ] TRIGGER имя { BEFORE | AFTER | INSTEAD OF } { событие [ OR ... ] }

ON имя\_таблицы

[ FOR [ EACH ] { ROW | STATEMENT } ]

[ WHEN ( условие ) ]

EXECUTE PROCEDURE имя\_функции ( аргументы )

, где имя – имя триггера;

BEFORE | AFTER | INSTEAD OF – определяет будет ли функция вызываться до, после или вместо события;

событие ***–*** здесь допускается INSERT, UPDATE [ OF имя\_столбца [, ... ] ], DELETE, TRUNCATE. параметр определяет событие, при котором будет срабатывать триггер;

имя\_таблицы – имя таблицы, для которой определен триггер;

ссылающаяся\_таблица – имя другой таблицы, на которую ссылается ограничение;

условие – логическое выражение, определяющее, будет ли выполняться функция триггера.

имя\_функции – пользовательская функция, объявленная как функция без аргументов и возвращающая тип trigger, которая будет вызываться при срабатывании триггера;

аргументы – необязательный список аргументов для функции.

**Порядок выполнения работы**

Перед или во время выполнения данной работы разрешается изменение структуры БД – добавление новых таблиц, переопределение связей, определение расчетных столбцов и т.д.

1. Создать следующие представления:

* Представление с вложенным запросом;
* Представление с группировкой;
* Представление с соединением;
* Изменяемое представление с условием WHERE.

1. Создать и отладить следующие запросы:

* Выборка данных из всех созданных представлений;
* Вставка/обновление в изменяемом представлении данных таких, что новые данные попадают под условие WHERE;
* Вставка/обновление в изменяемом представлении данных таких, что новые данные не попадают под условие WHERE.

1. Изменить представление так, чтобы вставка/обновление значений, не попадающих под условие WHERE, была запрещена.
2. Проверить запрос на вставку/обновление в измененном представлении.
3. Создать и отладить:

* Хранимую процедуру, выполняющую несколько инструкций;
* Хранимую процедуру с вводимым параметром (дата, идентификатор, значение столбца и т.д.)

1. Рассмотреть возможность запуска процедур по расписанию. При необходимости настроить запланированный запуск процедур.
2. Создать и отладить функции посредством вызова для отношения.
3. Создать триггеры на каждое событие к определенным отношениям в зависимости от логики заполнения, обновления и удаления данных. Рассмотреть возможность и реализацию вызова процедур в триггере.
4. Проверить работу триггеров.

**Вопросы к защите**

1. Представления, свойства, преимущества использования.
2. Хранимые процедуры, свойства, параметры, преимущества перед прямыми SQL-запросами к серверу.
3. Функции, свойства, параметры.
4. Триггеры, свойства, параметры.
5. Выполнение запроса согласно варианту.

**ПРАКТИЧЕСКАЯ РАБОТА № 5**

**РОЛИ И ПОЛЬЗОВАТЕЛИ**

**Цель работы**: изучить системные возможности для управления пользователями базы данных, ограничить выполнение определенных действий определенным пользователям.

**Порядок выполнения работы**

1. Просмотрите существующие роли в БД.
2. Создайте отдельные роли на чтение, вставку, обновление и удаление данных таблиц для схемы, созданной ранее.
3. Создайте пользователя с возможностью подключения к БД. Проверьте правильность выполненных действий. Присвойте пользователю права на создание и изменение БД.
4. Измените пароль для созданного пользователя. Определите срок действия пароля и проверьте правильность выполненных действий.
5. Создайте пользователя Admin и присвойте ему роль, обладающую полным доступ к БД.
6. Зайдите под учетной записью пользователя Admin и создайте пользователя User. Присвойте данному пользователю роль только для чтения записей из БД. Проверьте правильность выполненных действий.
7. Запретите пользователю User просмотр определенных таблиц и столбцов. Проверьте правильность выполненных действий.
8. Создайте пользователя Manager и разрешите ему просматривать и обновлять записи в БД. Просмотрите, какие роли определены для пользователя Manager.
9. Удалите пользователя Manager. Определите действия, которые необходимо выполнить для удаления пользователя.
10. Создайте группу ролей managers и добавьте ей права на чтение, вставку и обновление записей в БД.
11. Создайте заново пользователя Manager и добавьте его в группу ролей managers. Определите, как изменились права пользователя Manager при добавлении его в группу ролей.
12. Создайте пользователя, для которого будут определены возможности чтения, вставки, обновления и удаления данных для разных таблиц БД без добавления пользователя в группу ролей.
13. При необходимости создайте дополнительных пользователей для реализаций функций БД.

**Вопросы к защите**

Подготовить описание базы данных, полученной в ходе выполнения практических работ.