МИНИСТЕРСТВО СЕЛЬСКОГО ХОЗЯЙСТВА РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Башкирский государственный аграрный университет»

Кафедра информатики и
информационных технологий

Б1.О.18 БАЗЫ ДАННЫХ

Методические указания к лабораторным работам

РАЗРАБОТКА БАЗЫ ДАННЫХ В POSTGRESQL

для направления подготовки 09.03.03 Прикладная информатика
профиль подготовки Прикладная информатика цифровой экономики

Квалификация выпускника
бакалавр

Уфа 2021

Рекомендовано к изданию методической комиссией экономического факультета (протокол № 8 от 25.03.2021 г.)

Составитель: старший преподаватель Иванова Г.Р.

Ответственный за выпуск:
Заведующий кафедрой информатики и информационных технологий, д.т.н. доцент Беляева А.С.

г. Уфа, ФГБОУ ВО Башкирский ГАУ, кафедра информатики и информационных технологий

Содержание

Лабораторная работа № 1. Создание базы данных PostgreSQL

Целью работы является приобретение навыков создания баз данных в PostgreSQL.

1 Основные положения

PostgreSQL использует модель клиент/сервер. Сессия PostgreSQL состоит из следующих скооперированных процессов (программ):

- Серверного процесса (backend), который управляет файлами баз данных, осуществляет подключение к базам данным клиентских приложений и выполняет действия над базой данных, затребованные клиентами. Программа-сервер баз данных называется postgres
- Клиентского приложения пользователя (frontend), которое хочет выполнить операции с базой данных. Клиетнсткие приложения могут быть очень разными: ориентированными на работу с текстом, с графикой, с web-сервером или специальные инструменты обслуживания базы данных. Некоторые клиентские приложения поставляются в составе дистрибутива PostgreSQL, но подавляющее большинство пишется пользователями.

Для клиент-серверных приложений клиент и сервер обычно находятся на разных компьютерах. В этом случае, они соединяются по сети, через TCP/IP. Вы должны взять это на заметку, потому что файлы, которые доступны на клиентской машине могут быть недоступны (или доступны, используя другое имя файла) на машине-сервере.

Сервер PostgreSQL может управлять несколькими соединениями от клиентов. Чтобы осуществлять это, сервер запускает (через системный вызов "fork") новый процесс для каждого соединения. Начиная с момента запуска клиент и новый серверный процесс общаются друг с другом без помощи первоначального postgres процесса. Таким образом, процесс postgres запущен всегда и ожидает соединений от клиентов, после чего начинают работать клиент и соответствующий ему серверный процесс.

Первый тест, с помощью которого будет видно, что вы можете получить доступ к серверу баз данных - это попытка создать базу данных. Запущенный сервер PostgreSQLможет управлять множеством баз данных. Обычно, для каждого проекта или каждого пользователя используется отдельная база данных.

Возможно, что администратор вашей машины уже создал базу данных для вас. Он должен был бы сказать вам имя вашей базы данных. В этом случае вы можете пропустить данную секцию и перейти к следующей.

Для создания новой базы данных, в этом примере называющейся mydb, вы можете использовать следующую команду:

```
$ createdb mydb
```

Если команда завершилась без каких-либо сообщений, то данный шаг был пройден успешно и вы можете пропустить написанное далее до конца этой секции.

Если вы получите что-то похожее на:

```
createdb: command not found
```

то это означает, что продукт PostgreSQL не был правильно установлен. Или было установлено не все или путь поиска в вашем командном интерпретаторе не был установлен соответствующим образом. Попытайтесь вызвать эту команду, используя полный путь:

```
$ /usr/local/pgsql/bin/createdb mydb
```

Полный путь на вашей машине может быть другим. Свяжитесь с вашим администратором или проверьте инструкции по установке, чтобы исправить ситуацию.

Другое сообщение, которое вы можете получить:

```
createdb: could not connect to database postgres:
could not connect to server: No such file or directory
    No such file or directory
    Is the server running locally and accepting
    connections on Unix domain socket
"/tmp/.s.PGSQL.5432"?
```

Это означает, что сервер не был запущен или что он не был запущен так, как этого ожидает команда `createdb`. Снова, проверьте инструкции по установке или проконсультируйтесь с администратором.

Ещё одно сообщение, которое вы можете получить:

```
createdb: could not connect to database postgres: FATAL: user "joe" does not
exist
```

где вместо `joe` упоминается ваше имя регистрации. Такое сообщение появляется если администратор не создал для вас пользователя в PostgreSQL. Вам понадобится зарегистрироваться в операционной системе под именем того пользователя, из под которого работает PostgreSQL (обычно postgres), чтобы создать первого пользователя в СУБД. Для команды создания базы, вы также можете указать в PostgreSQL имя пользователя, которое отличается от вашего текущего имени пользователя в операционной системе; в этом случае вам необходимо задать имя пользователя PostgreSQL с помощью опции -U или установки переменной окружения PGUSER.

Если нужный пользователь существует, но у него нет прав, требуемых для создания базы данных, вы увидите следующее сообщение:

```
createdb: database creation failed: ERROR: permission denied to create database
```

Не каждый пользователь имеет авторизацию для создания новых баз данных. Если PostgreSQL отвергает ваши попытки создания баз данных, то администратору вашей машины нужно дать вам права для создания баз данных.

Вы можете также создавать базы данных с любыми другими именами. PostgreSQL позволяет вам создавать любое количество баз данных на одном сервере. Имена баз данных должны состоять из букв и цифр (вначале всегда должна быть буква) и быть не более 63 байт длиной. Довольно удобно создавать базу данных с таким же именем как у пользователя. Многие инструменты по умолчанию считают, что имя базы данных именно такое, так что

вы сможете не нажимать лишние кнопки. Чтобы создать такую одноимённую с именем пользователя базу данных, просто наберите:

```
$ createdb
```

Если вы не хотите использовать вашу базу данных в будущем, вы можете удалить ее. Например, если вы являетесь владельцем (создателем) базы данных `mydb`, вы можете уничтожить её, используя следующую команду:

```
$ dropdb mydb
```

(Для этой команды вы должны обязательно задать имя базы данных, она не будет считать, что имя базы данных - это имя текущего пользователя). Данная команда физически удалит все файлы, связанные с указанной базой данных так, что их нельзя будет восстановить, так что выполняйте эту операцию с большой осторожностью.

2 Содержание работы

1. Создайте таблицу «Книги» с полями Идентификатор (SERIAL), Название (20 символов), Автор (20 символов), Тип (1 символ), Количество (целое) (CREATE TABLE).
2. Создайте таблицу «Читатели» с полями Идентификатор (SERIAL), Имя (20 символов), Фамилия (20 символов), Тип (1 символ).
3. Создайте таблицу «Карточки» с полями Идентификатор (SERIAL), Идентификатор книги (целое), Идентификатор читателя (целое), Дата выдачи (дата), Дата возврата (дата).
4. Просмотрите структуру Таблиц (\d имя таблицы).
5. Измените таблицу «Книги», добавив первичный ключ к полю Идентификатор; новое поле Год издания (дата); (ALTER TABLE).
6. Измените таблицу «Читатели» добавив первичный ключ к полю Идентификатор.
7. Измените таблицу «Карточки» добавив первичный ключ к полю Идентификатор; внешние ключи к полям Идентификатор книги и Идентификатор читателя.

8. Просмотрите изменённую структуру таблиц.

9. Измените таблицу «Книги» удалив поле Год Издания.

10.Просмотрите изменённую структуру таблицы.

11.Удалите все таблицы (DROP TABLE).

12.Создайте таблицу «Книги» с полями, описанными в пункте 4, и свойствами, описанными пункте 5, одной командой CREATE TABLE. Выполните аналогичные действия с таблицами «Читатели» и «Карточки».

13.Показать таблицы преподавателю и удалить их.

Контрольные вопросы.

1. Синтаксис команд CREATE, ALTER, DROP.
2. Для чего нужны эти команды?
3. Какие типы могут иметь поля таблиц?

Лабораторная работа № 2. Создание, изменение и удаление данных

Целью работы является изучение методов создания, редактирования и удаления данных в PostgreSQL.

1 Основные положения

Таблица в реляционной базе данных состоит из строк и колонок. Каждая колонка имеет имя, а количество и порядок колонок являются постоянной величиной. Количество строк является переменной величиной — оно отражает количество данных, которое хранится в таблице в данный момент. SQL не гарантирует какого-то порядка строк в таблице. При чтении таблицы, строки выдаются в произвольном порядке, если только явно не затребована сортировка. Кроме того, SQL не назначает для строк уникальных идентификаторов, так что в таблице может существовать несколько полностью идентичных строк. Такое поведение является следствием математической модели, по которой работает SQL, но обычно оно нежелательно.

Каждая колонка имеет свой тип данных. Тип данных ограничивает список возможных значений, которые могут находится в этой колонке и определяет семантику данных, хранящихся в колонке, благодаря чему эти данные могут использоваться при каких-либо вычислениях. Например, если объявлено, что колонка имеет числовой тип данных, то в неё не могут быть помещены произвольные строки текста, а данные, хранимые в этой колонке могут быть использованы для математических вычислений. В противоположность этому, если объявлено, что колонка имеет тип данных символьная строка, то в неё можно поместить любой вид данных, но сами эти данные не могут быть использованы для математических вычислений, в то время как для этих данных доступны другие операции, например такие, как слияние строк.

PostgreSQL включает большой список встроенных типов данных, которые подходят для работы многих приложений. Пользователи также могут задавать свои собственные типы данных. Большинство встроенных типов данных имеют очевидные имена и семантику. Некоторые из наиболее часто используемых типов

— это integer для простых чисел, numeric для дробных чисел, text для символьных строк, date для дат, time для времени и timestamp для значений, содержащих и дату и время.

Чтобы создать какую-либо таблицу, используйте команду CREATE TABLE. В этой команде необходимо, как минимум, задать имя для новой таблицы, имена колонок и типы данных для этих колонок. Например:

```
CREATE TABLE my_first_table (
    first_column text,
    second_column integer
);
```

Создаётся таблица с именем my_first_table и двумя колонками. Первая колонка имеет имя first_column и тип данных text; вторая колонка имеет имя second_column и тип данных integer. Имена таблиц и колонок должны выбираться по синтаксическим правилам формирования идентификаторов. Имена типов обычно также являются идентификаторами, но есть некоторые исключения. Заметим, что список колонок заключается в круглые скобки, а описания колонок отделяются запятой.

Разумеется, что предыдущий пример сильно утрирован. Обычно, таблицам и колонками даются такие имена, которые говорят о том, какие данные в них хранятся. Вот пример, который более приближен к реальности:

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric
);
```

Тип numeric может хранить дробные значения, например такие как денежные.

Существует ограничение на количество колонок, которое может содержать таблица. В зависимости от типов данных колонок, оно колеблется между 250 и 1600. Однако, создание таблиц, содержащих такое множество колонок является

очень неэффективным подходом и часто является следствием некорректного проектирования базы данных.

Если какая-либо таблица вам больше не нужна, вы можете удалить её, используя команду DROP TABLE. Например:

```
DROP TABLE my_first_table;
DROP TABLE products;
```

Попытка удалить таблицу, которая не существует приведёт к ошибке. Тем не менее, во многих файлах с SQL скриптами выполняется безусловная попытка удалить каждую таблицу, перед её созданием, игнорируя сообщения об ошибках, так что скрипт работает существует ли данная таблица или нет. (Если вы хотите, вы можете использовать вариант `DROP TABLE IF EXISTS`, чтобы избежать получения сообщений об ошибке, но это не по стандарту SQL.)

2 Содержание работы

1. Создайте таблицы «Книги», «Читатели» и «Карточки» с полями описанными в Л.Р.№1.
2. Вставьте 10 записей в таблицу «Книги». В поле Тип указывайте тип книги: «к» - книга, «ж» - журнал, «м» - методичка. Записей с разными типами должно быть примерно одинаковое количество (INSERT).
3. Вставьте 10 записей в таблицу «Читатели». В поле Тип указывайте тип читателя: «с» - студент, «п» - преподаватель. Записей с разными типами должно быть примерно одинаковое количество.
4. Вставьте 20 записей в таблицу «Карточки».
5. Просмотрите записи в каждой из таблиц (SELECT).
6. В таблице «Книги» в каждой записи поменяйте значение поля Тип равное «к» на «у».
7. Просмотрите изменение в таблице «Книги».
8. В таблице «Книги» в каждой записи увеличьте на 200 значение поля Количество.
9. Просмотрите изменения в таблице «Книги».

10. Из таблицы «Карточки» удалите записи, у которых срок сдачи уже истек.

11.В таблице «Читатели» поменяйте местами преподавателей и студентов, т.е. в записях поменяйте местами значения «с» и «п» в поле Тип.

12.Убедитесь, что нельзя завести записи с одинаковым идентификатором.

Контрольные вопросы

1. Синтаксис команд INSERT, UPDATE, DELETE.
2. Для чего нужны эти команды?
3. Что будут делать команды UPDATE и DELETE без условия WHERE?

Лабораторная работа № 3. Запросы в PostgreSQL

Целью работы является изучение методологии работы с запросами в PostgreSQL.

1 Основные положения

Процесс извлечения или команда извлечения данных из БД, называется *запросом*. В SQL для выполнения запроса используется команда `SELECT`. Команда `SELECT` имеет синтаксис:

`[WITH` *`с_запросами`*`] SELECT` *`список_выборки`* `FROM` *`табличное_выражение`* `[`*`спецификация_сортировки`*`]`

В следующих подразделах подробно описывается список выборки, табличное выражение и спецификация сортировки. Запросы с `WITH` обсуждаются в конце, так как они являются дополнительной возможностью SQL.

Простейший запрос выглядит так:

```
SELECT * FROM table1;
```

Эта команда извлекает из таблицы table1, все строки и все колонки. Список выборки * означает все колонки, из представленного табличного выражения. Список выборки также может задавать подсписок доступных колонок или выполнять вычисления с использованием колонок. Например, если таблица table1 имеет колонки с именами a, b и c (и возможно ещё и другие) вы можете создать следующий запрос:

```
SELECT a, b + c FROM table1;
```

(предполагая, что b и c имеют числовой тип данных).

`FROM table1` – это простой вид табличного выражения: оно соответствует только одной таблице. Но табличные выражения могут быть и сложными конструкциями, которые базируются на таблицах, соединениях и подзапросах. Но вы также можете полностью опустить табличное выражение и использовать команду `SELECT` как калькулятор:

```
SELECT 3 * 4;
```

Такой способ полезен, если выражения в списке выбора возвращают разные результаты. Например, вы можете таким способом вызвать функцию:

```
SELECT random();
```

1.1. Табличные выражения

Табличное выражение предоставляет некую таблицу. Табличное выражение содержит предложение FROM, за которым необязательно следуют предложения WHERE, GROUP BY и HAVING. Простое табличное выражение просто указывает на какую-либо таблицу на диске, так называемую базовую таблицу, но для модификации и комбинирования базовых таблиц различными способами могут быть использованы более сложные выражения.

Необязательные предложения WHERE, GROUP BY и HAVING в табличном выражении, задают конвейер последовательных преобразований, выполняемых над таблицей, полученной из предложения FROM. Все эти преобразования создают виртуальную таблицу, которая предоставляет строки, которые затем передаются в список выборки для отбора строк на выходе запроса.

Предложение FROM

FROM Clause создаёт таблицу из одной или более других таблиц, которые задаются списком разделённых запятыми ссылок на таблицы:

```
FROM ссылка_на_таблицу [, ссылка_на_таблицу [, ...]]
```

Ссылка на таблицу может быть именем таблицы (возможно с указанием схемы) или производной таблицей, такой как подзапрос, соединением таблиц, или сложной комбинацией из них. Если в списке предложения FROM указано более одной ссылки на таблицу, они просто соединяются (см. ниже) в форму промежуточной виртуальной таблицы, которая может затем быть преобразована с помощью предложений WHERE, GROUP BY, и HAVING, и в конце-концов стать результатом всего табличного выражения.

Когда ссылка на таблицу является родительской таблицей в иерархии наследования, данная ссылка на таблицу выдаёт строки не только этой таблицы, но и всех её таблиц-потомков, если перед именем таблицы не указано ключевое

слово ONLY. Однако, ссылка на такую таблицу выдаёт только колонки, которые есть в этой таблице; любые другие колонки в таблицах-потомках игнорируются.

2 Содержание работы

1. Подключитесь к базе данных «library» (`\c название БД`).
2. Для изменения кодировки используйте команду `\encoding WIN866`
3. Определите какие таблицы входят в базу данных «library» и какова их структура (`\d`).
4. Просмотрите все записи в каждой таблице (`SELECT, FROM`).
5. Составьте запросы, которые бы выводили только студентов (тип «с») и только преподавателей (тип «п») из таблицы «readers» (`SELECT, FROM, WHERE`).
6. Составьте запрос, который бы выводил книги (тип «к») и методички (тип «м») из таблицы «books» (`SELECT, FROM, WHERE, AND`).
7. Составьте запрос, который бы выводил книги (тип «к»), название которых содержит слово «Анализ» с учётом регистра первой буквы (`SELECT, FROM, WHERE, AND, LIKE`).
8. Составьте запрос, который бы выводил список фамилий читателей, книги, которые за ними числятся, и срок сдачи упорядоченный по фамилиям (`SELECT, FROM, WHERE, AND, ORDER BY`).
9. Составьте запрос, который бы выводил количество читателей разного типа. Т.е. сколько студентов и сколько преподавателей. Список должен быть упорядочен по количеству читателей (`SELECT, COUNT(), FROM, GROUP BY, ORDER BY`).
10. Составьте запрос, который бы выводил список должников, книги и количество дней, на которое они задержали эти книги (`SELECT, COUNT(), FROM, ORDER BY, CURRENT_DATE`).

11.Составьте запрос, который бы выводил список читателей и количество книг, которые на них числятся. Сделайте два варианта: отсортированный по фамилиям и по количеству книг (SELECT, COUNT(), FROM, GROUP BY, ORDER BY).

12.Составьте запрос, который бы выводил список читателей у которых больше трех книг(SELECT, COUNT(), FROM, GROUP BY, ORDER BY, HAVING).

13.Составьте запрос, который бы выводил список 5 популярных книг с указанием количества читателей, взявших эти книги. Список должен быть отсортирован по убыванию (SELECT, COUNT(), FROM, GROUP BY, ORDER BY, LIMIT, DESC).

Контрольные вопросы.

1. Синтаксис команды SELECT.
2. Как выглядит типовой запрос с использованием группировки?
3. В чем различие WHERE и HAVING?

Лабораторная работа № 4. Внешние соединения

Целью работы является изучение методологии работы с запросами в PostgreSQL.

1 Основные положения

Соединённая таблица – это таблица, производная от двух других (реальных или в свою очередь производных) таблиц, полученая в соответствии с правилами определённого типа соединения. Доступные типы соединения: inner (внутреннее), outer (внешнее) и cross (перекрёсное).

Типы соединений

Перекрёстное соединение (cross join)

T1 CROSS JOIN *T2*

Для каждой комбинации строк из *T1* и *T2* (декартово произведение) соединённая таблица будет содержать строки, состоящие из всех колонок таблицы *T1*, за которым следуют все колонки из таблицы *T2*. Если таблицы имеют соответственно N и M строк, соединённая таблица будет иметь N * M строк.

FROM T1 CROSS JOIN T2 эквивалентно FROM T1, T2. Также это эквивалентно FROM T1 INNER JOIN T2 ON TRUE.

Квалифицированные соединения

T1 { [INNER] | { LEFT | RIGHT | FULL } [OUTER] } JOIN T2 ON логическое_выражение

T1 { [INNER] | { LEFT | RIGHT | FULL } [OUTER] } JOIN T2 USING (список_колонок_для_соединения)

T1 NATURAL { [INNER] | { LEFT | RIGHT | FULL } [OUTER] } JOIN T2

Слова INNER и OUTER являются необязательными во всех формах. INNER применяется по умолчанию; LEFT, RIGHT и FULL неявно указывают на внешнее (outer) соединение.

Условие соединения задаётся в предложениях ON или USING, или неявно, с помощью слова NATURAL. Условие соединения определяет какие строки из двух исходных таблиц будут считаться "совпавшими", как ниже описывается в деталях.

Предложение ON является наиболее обычным способом задать условие соединения: в нём указывается логическое выражение того же типа, что используется в предложении WHERE. Пара строк из *T1* и *T2* считается совпавшей, если для неё, выражение, заданное в ON принимает значение истина.

USING имеет более краткую форму: в ней указывается разделённый запятыми список имён колонок, которые должны быть общими в соединяемых таблицах и условия соединения, задавая равенство каждой из этих пар колонок. Таким образом, вывод JOIN USING будет состоять из одной колонки для каждой, являющейся равной, пары входных колонок, за которыми следует остальные колонки из каждой таблицы. Так, USING (a, b, c) эквивалентно ON (t1.a = t2.a AND t1.b = t2.b AND t1.c = t2.c) с тем исключением, что если ON используется там, где в результате будут две колонки a, b и c, то USING там, где будет только одна из них (и они будут выданы первыми, если используется SELECT *).

NATURAL – это краткая форма USING: она формирует список USING, содержащий все имена колонок, которые есть в обеих входных таблицах. Как и с USING, эти колонки появляются в выходной таблице только один раз. Если общих колонок нет, NATURAL ведёт себя как CROSS JOIN.

Возможные типы квалифицированных соединений:

INNER JOIN

Для каждой строки R1 из таблицы T1, соединённая таблица будет иметь строку для каждой строки в T2, которая удовлетворяет условию соединения с R1.

LEFT OUTER JOIN

Сначала выполняется INNER JOIN. Затем, для каждой строки в T1, которая не удовлетворяет условию соединения с любой строкой в T2, добавляется

соединённая строка с значениями NULL в колонках T2. Таким образом, соединённая таблица всегда имеет, по крайней мере, одну строку для каждой строки из T1.

`RIGHT OUTER JOIN`

Сначала выполняется `INNER JOIN`. Затем, для каждой строки в T2, которая не удовлетворяет условию соединения с любой строкой в T1, добавляется соединенная строка со значениями `NULL` в колонках T1. Этот тип соединения является обратным по отношению к `LEFT JOIN`: результирующая таблица всегда будет иметь хотя бы одну строку для каждой строки из T2.

`FULL OUTER JOIN`

Сперва выполняется `INNER JOIN`. Затем, для каждой строки в T1, которая не удовлетворяет условию соединения с любой строкой в T2, добавляется соединенная строка с значениями NULL в колонках T2. Также, для каждой строки T2, которая не удовлетворяет условию соединения с любой строкой в T1, добавляется соединённая строка с значениями `NULL` в колонках T1.

Соединения всех типов могут быть сцеплены вместе или скомпонованы: одна или обе таблицы *T1* и *T2* могут быть соединяющимися таблицами. Для управления порядком выполнения соединений, вокруг `JOIN` могут использоваться круглые скобки. В отсутствие скобок, предложения `JOIN` компонуются слева направо.

2 Содержание работы

1. Подключитесь к базе данных «library» (`\c название БД`).
2. Для изменения кодировки используйте команду `\encoding WIN866`
3. Определите какие таблицы входят в базу данных «library» и какова их структура (`\d`).
4. Составьте запрос, который бы выводил список всех читателей соединяя их с книжками, которые они взяли. Список должен включать читателей,

которые не брали книжек (SELECT, FROM, LEFT JOIN (в запросе используется 2 LEFT JOIN)).

5. Составьте запрос, который бы выводил список всех книжек соединяя их с читателями, которые их взяли. Список должен включать книжки, которых не брали (SELECT, FROM, LEFT JOIN (в запросе используется 2 LEFT JOIN)).

6. Составьте запрос, который бы выводил список всех читателей (Фамилия, Имя) с количеством книжек, которые они взяли. Список должен включать читателей, которые не брали книжек. Список должен быть отсортирован по количеству (SELECT, FROM, LEFT JOIN, COUNT(), GROUP BY).

7. Составьте запрос, который бы выводил список всех книжек с количеством читателей, которые их взяли. Список должен включать книжки, которых не брали (SELECT, FROM, LEFT JOIN, COUNT(), GROUP BY).

8. Составьте запрос аналогичный п.5, с указанием количества дней, на которые брались все книги, и с указанием среднего арифметического, минимума и максимума количества дней. Список должен быть отсортирован по количеству дней и по фамилиям (SELECT, FROM, LEFT JOIN, COUNT(), AVG(), MIN(), MAX(), GROUP BY, ORDER BY, CURRENT_DATE).

9. Составьте запрос аналогичный п.3, с указанием количества дней, на которые брались книги, и с указанием количества просроченных дней. Список должен быть отсортирован по количеству просроченных дней и по фамилиям (SELECT, FROM, LEFT JOIN, COUNT(), GROUP BY, ORDER BY, CURRENT_DATE (в запросе используется 2 LEFT JOIN)).

Контрольные вопросы

1. Синтаксис команды SELECT с использованием JOIN.
2. Как выглядит типовой запрос с использованием внешнего соединений?
3. В чем различие LEFT JOIN и RIGHT JOIN?

Лабораторная работа № 5. Расширения языка SQL. Представления

Целью работы является изучение представлений в PostgreSQL.

1 Основные положения

Предположим, что вас интересует составной список из погодных записей и координат городов, но вы не хотите каждый раз вводить весь этот запрос. Вы можете создать представление по данному запросу, фактически присвоить имя запросу, а затем обращаться к нему как к обычной таблице:

```
CREATE VIEW myview AS
    SELECT city, temp_lo, temp_hi, prcp, date, location
        FROM weather, cities
        WHERE city = name;

SELECT * FROM myview;
```

Активное использование представлений — это ключевой аспект хорошего проектирования баз данных SQL. Представления позволяют вам скрыть внутреннее устройство ваших таблиц, которые могут меняться по мере развития приложения, за надёжными интерфейсами.

Представления можно использовать практически везде, где можно использовать обычные таблицы. И довольно часто представления создаются на базе других представлений.

2 Содержание работы

1. Создайте новую базу данных.
2. Измените кодировку (`\encoding WIN866`).

3. Добавить данные из файла **X:\kav\DataBase\library.sql** (`\i имя файла`).

4. Определите какие таблицы входят в базу данных и какова их структура (`\d`).

5. Создать горизонтальное представление в которое входят только преподаватели с сортировкой по фамилии (`CREATE VIEW`).

6. Просмотрите структуру представления (`\d представление`).

7. Создать горизонтальное представление, содержащие Идентификатор, Фамилию, Имя в которое входят только студенты вашей группы с сортировкой по фамилии (`CREATE VIEW`).

8. Создать представление, содержащее список должников (с Идентификатором), книги и количество дней, на которое они задержали эти книги, с сортировкой по фамилии.

9. Добавить запись в таблицу **carts**. Проверить изменилось ли содержимое представления из п.7.

10. Создать два представления, как в п.6, таким образом, чтобы преподаватели были в одном, а студенты в другом.

11. Создать два представления, как в п.7, таким образом, чтобы преподаватели были в одном, а студенты в другом.

12. Создать представление, которое содержало список читателей у которых больше трёх книг.

13. Составьте представление аналогично п.7, с указанием количества дней, на которые брались книги, и с указанием количества просроченных дней. Список должен быть отсортирован по количеству просроченных дней и по фамилиям.

3 Контрольные вопросы.

1. Синтаксис команды CREATE VIEW.
2. Как выглядит типовой запрос к представлению?

3. В чем различие между представлением и таблицей?

Лабораторная работа № 6. Расширения языка SQL. Пользователи и права доступа.

Целью работы является изучение представлений в PostgreSQL.

1 Основные положения

Когда в базе данных создаётся объект, ему назначается владелец. Владельцем обычно становится роль, с которой был выполнен оператор создания. Для большинства типов объектов в исходном состоянии только владелец (или суперпользователь) может делать с объектом всё, что угодно. Чтобы разрешить использовать его другим ролям, нужно дать им *права*.

Существует несколько типов прав: `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE`, `REFERENCES`, `TRIGGER`, `CREATE`, `CONNECT`, `TEMPORARY`, `EXECUTE` и `USAGE`. Набор прав, применимых к определённому объекту, зависит от типа объекта (таблица, функция и т. д.)

Неотъемлемое право изменять или удалять объект имеет только владелец объекта.

Объекту можно назначить нового владельца с помощью команды `ALTER` для соответствующего типа объекта, например `ALTER TABLE`. Суперпользователь может делать это без ограничений, а обычный пользователь, только если он является одновременно текущим владельцем объекта (или членом роли владельца) и членом новой роли.

Для назначения прав применяется команда `GRANT`. Например, если в базе данных есть роль `joe` и таблица `accounts`, право на изменение таблицы можно дать этой роли так:

```
GRANT UPDATE ON accounts TO joe;
```

Если вместо конкретного права написать `ALL`, роль получит все права, применимые для объекта этого типа.

Для назначения права всем ролям в системе можно использовать специальное имя «роли»: PUBLIC. Также для упрощения управления ролями, когда в базе данных есть множество пользователей, можно настроить «групповые» роли.

Чтобы лишить пользователей прав, используйте команду REVOKE:

```
REVOKE ALL ON accounts FROM PUBLIC;
```

Особые права владельца объекта (то есть права на выполнение DROP, GRANT, REVOKE и т. д.) всегда неявно закреплены за владельцем и их нельзя назначить или отобрать. Но владелец объекта может лишить себя обычных прав, например, разрешить всем, включая себя, только чтение таблицы.

Обычно распоряжаться правами может только владелец объекта (или суперпользователь). Однако возможно дать право доступа к объекту «с правом передачи», что позволит получившему такое право назначать его другим. Если такое право передачи впоследствии будет отозвано, то все, кто получил данное право доступа (непосредственно или по цепочке передачи), потеряют его.

2 Содержание работы

1. Проверьте работоспособность пользователя «dummy625_№», где № ваш номер, пароль тот же что и у «u625_№».
2. Создайте новую базу данных от имени «dummy625_№» и создайте таблицы из Л.Р. №1.
3. Вставте по 5 записей в каждую таблицу.
4. Выдайте права **чтения** на все таблицы пользователю «u625_№» (GRANT).
5. Проверить наличие прав (SELECT).
6. Выдайте права **добавления** в таблицу «Книги» пользователю «u625_№».
7. Проверить наличие прав (INSERT).

8. Выдайте права **удаления** в таблице «Читатели» пользователю «u625_№».

9. Проверить наличие прав (`DELETE`).

10. Выдайте права **изменения** в таблице «Карточки» пользователю «u625_№».

11. Проверить наличие прав (`UPDATE`).

12. Создать горизонтальное представление, содержащие Идентификатор, Фамилию, Имя в которое входят только студенты с сортировкой по фамилии. Выдайте права **чтения** для этого представления пользователю «u625_№».

13. Проверить наличие прав (`SELECT`).

14. Забрать права **чтения** на все таблицы у пользователя «u625_№» (`REVOKE`).

15. Проверить наличие прав (`SELECT`).

16. Забрать права **добавления** в таблицу «Книги» у пользователя «u625_№».

17. Проверить наличие прав (`INSERT`).

18. Забрать права **удаления** в таблице «Читатели» у пользователя «u625_№».

19. Проверить наличие прав (`DELETE`).

20. Забрать права **изменения** в таблице «Карточки» у пользователя «u625_№».

21. Проверить наличие прав (`UPDATE`).

22. Забрать права **чтения** на представление у пользователя «u625_№».

23. Проверить наличие прав (`SELECT`).

3 Контрольные вопросы

1. Синтаксис команды GRANT, REVOKE.
2. Назначение прав доступа

Лабораторная работа № 7. WEB-интерфейс для работы с PostgreSQL

Целью работы является изучение способов управления сервером PostgreSQL через веб-интерфейс.

1 Основные положения

Администрирование баз данных всегда считалось достаточно сложной задачей. Стремление сделать этот процесс более простым и эффективным привело к созданию специализированных программных инструментов различного плана, в том числе позволяющих управлять базами данных с помощью web-интерфейса.

Скрипты **PhpPgAdmin** разработаны для работы с **СУБД PostgreSQL**. Данная разработка изначально базировалась на **PhpMyAdmin**. Сохранив основную идею **PhpMyAdmin**, **PhpPgAdmin** обзавелся и значительным количеством специфичных добавок. **PhpPgAdmin** позволяет администрировать несколько серверов, работать с полномочиями пользователей, управлять базами данных и таблицами, выполнять сложные **SQL**-запросы.

Установку **PhpPgAdmin** выполним из системы портов. Описание дистрибутива **PhpPgAdmin** в системе портов следующее:

```
phpPgAdmin is phpMyAdmin (for MySQL) ported to PostgreSQL.
phpPgAdmin is a
fully functional PostgreSQL administration utility. You can
use it to create
and maintain multiple databases and even multiple servers.
Features include:
 - create and drop databases
 - create, copy, drop and alter
   tables/views/sequences/functions/indicies/triggers
 - edit and add fields (to the extent Postgres allows)
 - execute any SQL-statement, even batch-queries
 - manage primary and unique keys
 - create and read dumps of tables
 - administer one single database
 - administer multiple servers
 - administer postgres users and groups
```

Непосредственно установка порта:

```
# cd /usr/ports/databases/phppgadmin && make install clean
&& rehash
```

По завершению установки добавим в **httpd.conf** такой блок:

```
Alias /pga/ "/usr/local/www/phpPgAdmin/"
<Directory "/usr/local/www/phpPgAdmin/">
          Options -Indexes
          AllowOverride Limit
          Order Deny,Allow
          Allow from all
</Directory>
```

Для того, чтобы изменения вступили в силу, даем команду на перезапуск Apache:

```
# apachectl graceful
```

После этого в строке браузера вводим ссылку **http://ip_сервера/pga/** и попадаем в интерфейс.

2 Содержание работы

1. Создайте базу данных.
2. Создайте домены и таблицы, описанные как «Большой пример»
3. Вставьте 5 записей в таблицы ***spec, teacher, kafedra.***
4. Вставьте 10 записей в таблицы ***student, uchplan, nagruzka.***
5. Вставьте 20 записей в таблицы ***ocenka.***
6. Составить запросы, которые бы выводили список предметов ведущихся на кафедрах с сортировкой по названию кафедры.
7. Создайте представление по п.6.
8. Составить запросы, которые бы выводили список студентов, предметы и оценки по этим предметам с сортировкой по оценкам (по убыванию) и фамилиям.
9. Создайте представление по п.8.

10. Вывести список предметов, кто их ведёт в каких группах и сколько пар с сортировкой по названию предмета.

11. Создайте представление по п.10.

12. Создайте представление выводящее список специальностей с количеством обучающихся студентов на данной специальности.

13. Создайте представление выводящее список кафедр с количеством преподавателей.

14. Создайте представление выводящее список предметов с общим количеством часов по всем семестрам.

15. Создайте базу данных «Библиотека». Название базы должно быть начинаться с Вашего сетевого логина (пример u625) без пробелов.

16. Создайте таблицу «Книги» с полями Идентификатор (целое, не принимающее пустых значений), Название (20 символов), Автор (20 символов), Тип (1 символ), Количество (целое) (CREATE TABLE).

17. Создайте таблицу «Читатели» с полями Идентификатор (целое, не принимающее пустых значений), Имя (20 символов), Фамилия (20 символов), Тип (1 символ).

18. Создайте таблицу «Карточки» с полями Идентификатор (целое, не принимающее пустых значений), Идентификатор книги (целое), Идентификатор читателя (целое), Дата выдачи (дата), Дата возврата (дата).

19. Измените таблицу «Книги», добавив первичный ключ к полю Идентификатор; индексы к полям Название, Автор, Тип; новое поле Год издания (дата); новое свойство AUTO_INCREMENT к полю Идентификатор (ALTER TABLE).

20. Измените таблицу «Читатели» добавив первичный ключ к полю Идентификатор; индексы к полям Фамилия, Тип; новое свойство AUTO_INCREMENT к полю Идентификатор.

21. Измените таблицу «Карточки» добавив первичный ключ к полю Идентификатор; новое свойство AUTO_INCREMENT к полю Идентификатор.

22. Вставьте 10 записей в таблицу «Книги». В поле Тип указывайте тип книги: «к» - книга, «ж» - журнал, «м» - методичка. Записей с разными типами должно быть примерно одинаковое количество (INSERT).

23. Вставьте 10 записей в таблицу «Читатели». В поле Тип указывайте тип читателя: «с» - студент, «п» - преподаватель. Записей с разными типами должно быть примерно одинаковое количество.

24. Вставьте 20 записей в таблицу «Карточки».

25. Просмотрите записи в каждой из таблиц.

26. В таблице «Книги» в каждой записи поменяйте значение поля Тип равное «к» на «у».

27. В таблице «Книги» в каждой записи увеличьте на 200 значение поля Количество.

28. Перейдите в базу данных lib.

29. Составьте запросы, которые бы выводили только студентов (тип «с») и только преподавателей (тип «п») из таблицы «readers» (SELECT, FROM, WHERE).

30. Составьте запрос, который бы выводил книги (тип «к») и методички (тип «м») из таблицы «books» (SELECT, FROM, WHERE, AND).

31. Составьте запрос, который бы выводил книги (тип «к»), название которых содержит слово «Анализ» (SELECT, FROM, WHERE, AND, LIKE).

32. Составьте запрос, который бы выводил количество читателей разного типа. Т.е. сколько студентов и сколько преподавателей. Список должен быть упорядочен по количеству читателей (SELECT, COUNT(), FROM, GROUP BY, ORDER BY).

3 Контрольные вопросы

1. Как настроить WEB-интерфейс для работы с PostgreSQL?

Библиографический список

1. Документация к Postgres Pro Standard 10.4.1. Режим доступа - https://postgrespro.ru/docs/postgrespro/10/index
2. Кара-Ушанов В. Ю. SQL – язык реляционных баз данных: Учебное пособие / Кара-Ушанов В.Ю., - 2-е изд., стер. - М.:Флинта, Изд-во Урал. ун-та, 2017. - 156 с.