

А. В. Некрасов, Л. Г. Пастухова

ПРИНЦИПЫ ИСПОЛЬЗОВАНИЯ СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ POSTGRESQL

Учебно-методическое пособие для самостоятельной работы

Текстовый электронный образовательный ресурс

Учебно-методическое пособие для самостоятельной работы по дисциплине
«Информационные технологии в строительстве» для студентов магистратуры,
обучающихся по направлению подготовки 08.04.01 Строительство

Подготовлено кафедрой гидравлики
Институт Строительства и Архитектуры

Екатеринбург
2025

Содержание

Вводные замечания	3
1. Работа в среде клиента psql.....	4
1.1. Запуск клиента	4
1.2. Команды управления метаданными	5
1.3. Команды модификации данных	7
1.4. Пример создания базы данных.....	9
1.4. Выборка данных	11
1.6. Экспорт и импорт данных	17
2. Работа в среде клиента pgAdmin III	19
3. Использование средств искусственного интеллекта для создания запросов ...	24
4. Задание для самостоятельной работы	26
Библиографический список	28

Вводные замечания

PostgreSQL (произносится как пост-грес-кью-эл) – современная свободно распространяемая система управления реляционными базами данных (СУБД) с широкими возможностями. Работа PostgreSQL осуществляется по технологии клиент-сервер. В этой связи необходимо остановиться на некоторых основных понятиях.

Сейчас сложно представить нашу жизнь без компьютерных сетей вообще и особенно без Интернета. Любая сеть это средство, предназначенное для совместного использования ресурсов нескольких компьютеров (слово «компьютер» здесь и далее используется условно для краткости, в общем случае это может быть любое современное цифровое устройство: смартфон, планшет и т. п.).

Далее речь пойдет о совместном использовании файлов и программ. Важно иметь в виду, что термины сервер и клиент используются в данном случае для обозначения программного обеспечения. Всем известными примерами клиентов являются различные браузеры. Браузеры взаимодействуют с тем или иным приложением веб-сервером: MS Internet Information Services, Apache и т. п. Браузер по нашей команде формирует специальное HTTP-сообщение, в котором указано какие данные и в каком виде он хочет получить от веб-сервера. Сообщения клиентов называются запросами. Сервер, получив такое сообщение, отправляет браузеру ответ (обычно это HTML-документ), в котором содержится нужная информация.

Если вернуться к PostgreSQL, то именно это приложение и является сервером. Для PostgreSQL разработано несколько клиентских приложений, например: psql, pgAdmin и т. п. Запросы можно формировать и с помощью приложений, разработанных на том или ином языке программирования.

Стоит заметить, что взаимодействие между клиентом и сервером всегда инициирует клиент, а сервер лишь отвечает клиенту (или сообщает может ли он предоставить услугу клиенту и если может, то на каких условиях). Клиентское и серверное программное обеспечение обычно установлено на разных устройствах, но они могут работать и на одном компьютере. В последнем случае такой компьютер выполняет функции виртуального сервера и называется localhost.

Основным средством взаимодействия пользователя с базой данных является язык SQL (Structured Query Language – язык структурированных запросов). SQL не является языком программирования и не имеет никаких средств для отображения и управления данными (операторов выбора, циклов и т. п.).

Установку PostgreSQL на домашние компьютеры желательно выполнять с официального сервера. В Интернете можно легко найти подробные пошаговые инструкции для выполнения этой процедуры.

1. Работа в среде клиента psql

1.1. Запуск клиента

Клиент psql – стандартное средство, входящее в любую версию PostgreSQL. Он достаточно удобен для решения повседневных задач по администрированию баз данных, для написания относительно простых запросов. Клиент имеет собственные команды, позволяющие ориентироваться в объектах, хранящихся в базе данных, и представлять их на экране.

Запуск приложения psql осуществляется из меню «Пуск», в котором необходимо ввести «SQL Shell»*. Вы увидите приглашение:

```
postgres=#
```

Здесь postgres – имя базы данных, к которой пользователь подключается по умолчанию. Она хранит служебную информацию, например, о пользователях, их базах данных и т. п.

Администрирование PostgreSQL в данном пособии не рассматривается. Ограничимся лишь некоторыми полезными командами. Далее для удобства читателя команды, которые нужно вводить с клавиатуры, выделяются **жирным** шрифтом.

Отметим, что в некоторых случаях PostgreSQL различает строчные и прописные буквы в командах. Далее команды SQL для наглядности будут записываться прописными буквами, а специальные команды psql – строчными.

Специальные команды psql всегда начинаются с обратной косой черты «\». Например, для завершения работы PostgreSQL необходимо выполнить команду:

```
postgres=#\q
```

Полный перечень команд с их краткими описаниями можно получить с помощью команды \?. Некоторые из них будут рассмотрены далее.

Для получения справки введите \help, или \help [имя команды]. Здесь и далее в квадратных скобках будем указывать необязательные параметры. Например:

```
postgres=#\help UPDATE
```

На экран будет выведено описание синтаксиса команды update.

Запуск PostgreSQL всегда осуществляется от имени администратора. Администратор с именем postgres имеет права суперпользователя (суперпользователь, создается автоматически и существует всегда). Он имеет полный доступ к базам данных, может определять новых пользователей, назначать им права доступа к данным и выполнять другие административные функции.

Для получения списка зарегистрированных на данном сервере пользователей и их баз данных необходимо выполнить команду:

* В данном пособии рассматривается приложение, установленное на компьютерах Института строительства и архитектуры УрФУ

```
postgres=#\list
```

Если символы кириллицы отображаются не верно, измените кодовую страницу:

```
postgres=#\! chcp 1251
```

Примечание. После знака «!» обязательно введите пробел.

Перечень упомянутых выше, а также других полезных команд, приведен в табл. 1.

1.2. Команды управления метаданными

Метаданные – это данные, которые описывают структуру и смысл других данных. В нашем случае это сами базы данных и их таблицы. Таким образом, команды управления метаданными предназначены для создания, изменения или удаления баз данных и содержащихся в них таблиц. Строго говоря, в состав современных баз данных сейчас включаются не только таблицы, но и другие объекты. Однако их изучение выходит за пределы нашего краткого курса. Поэтому мы рассмотрим лишь некоторые команды данной группы.

Таблица 1

Некоторые команды psql

Команда	Назначение
\?	Справка psql
\! chcp номер кодовой таблицы	Изменение кодовой таблицы
\h	Справка по всем командам SQL
\h команда	Справка по указанной команде SQL
\c база данных	Подключение к указанной базе данных
\d	Список таблиц активной базы данных
\d таблица	Список атрибутов указанной таблицы
\list	Список зарегистрированных пользователей и баз данных
\q	Завершение работы

Создание новой базы осуществляется командой SQL

```
CREATE DATABASE имя_базы_данных
```

Имя базы данных указывается латинскими буквами без пробелов с учетом регистра. Первым символом не может быть цифра. Рекомендуем использовать строчные буквы.

В качестве примера создадим базу данных с именем test. Выполните команду:

```
postgres=# CREATE DATABASE test;  
CREATE DATABASE
```

Обратите внимание на отсутствие символа «\» в начале команды и на точку с запятой в ее конце. Пока PostgreSQL не «увидит» точку с запятой, он будет считать, что ввод команды будет продолжен в следующей строке (таким образом, длинную команду можно разбить на несколько строк). Во второй строке приведенного примера видим повторение введенной команды. Это ответ сервера, показывающий, что команда выполнена.

Рекомендуем следить за ответами сервера. Если после нажатия клавиши Enter ответа не последовало, значит ввод команды не завершен.

Ранее введенные команды можно вновь вызвать на экран для повторного выполнения. При необходимости их можно откорректировать. Для это используйте клавиши управления курсором.

Для работы с созданной базой к ней необходимо подключиться командой

```
\с имя_базы_данных
```

Выполните:

```
postgres=# \с test
```

```
You are now connected to database "test" as user "postgres".
```

```
test=#
```

Ответом сервера является сообщение о подключении к базе данных test пользователя postgres. Приглашение сменилось на test=#.

База данных состоит из одной и более таблиц. У каждой таблицы есть имя. Атрибуты (столбцы) таблицы имеют имена и тип.

Для создания новой таблицы необходимо использовать команду

```
CREATE TABLE имя_таблицы (атрибут1 тип_атрибута1 [, атрибут2 тип_атрибута2]
                          [, ...] )
```

Здесь и далее необязательные параметры команды заключены в квадратные скобки []. Заметим, что это упрощенная форма команды. Некоторые ее дополнительные параметры мы рассмотрим далее.

PostgreSQL не только располагает большим количеством встроенных типов данных, но и позволяет пользователям создавать собственные типы. Мы ограничимся рассмотрением лишь нескольких встроенных типов:

- integer – целые числа;
- numeric – действительные числа;
- date – даты;
- text – текстовые строки;
- boolean – логический тип, принимающий значения true (истина) или false (ложь).

Помимо обычных значений, определяемых типом данных, атрибут может иметь неопределенное значение null. Его можно рассматривать как «значение не задано».

Создайте таблицу example с атрибутами (atr1, atr2 и т. д.) указанных выше типов:

```
test=# CREATE TABLE example(atr1 integer, atr2 numeric, atr3 text, atr4
boolean, atr5 date);
```

```
CREATE TABLE
```

Для изменения структуры (состава атрибутов и их типов) существующей таблицы используется команда ALTER TABLE. Применяя ее, можно добавить или удалить атрибут, задать ключевые атрибуты в существующей таблице и т. п.

Для добавления атрибута применяется команда:

```
ALTER TABLE имя_таблицы ADD атрибут тип_атрибута
```

Например, для добавления атрибута atr6 целого типа к таблице example нужно выполнить команду:

```
ALTER TABLE example ADD atr6 integer
```

Другая разновидность команды ALTER TABLE применяется для удаления атрибута:

```
ALTER TABLE имя_таблицы DROP атрибут
```

Например, для удаления atr6:

```
ALTER TABLE example DROP atr6
```

Для удаления таблицы из базы данных используется команда:

```
DROP TABLE имя_таблицы
```

Удалить всю базу данных можно командой:

```
DROP DATABASE имя_базы_данных
```

Конечно, использовать эти команды следует очень осторожно, поскольку восстановить удаленные данные невозможно.

1.3. Команды модификации данных

Команды этой группы предназначены для внесения в таблицы конкретных значений атрибутов, их изменения или удаления.

Для добавления в таблицу новой строки данных (кортежа) предназначена команда

```
INSERT INTO имя_таблицы ( атрибут1 [, атрибут2] [, ...]) VALUES ( значение_
атрибута1 [, значение_атрибута2][, ...] )
```

Заметим, что это одна из возможных форм команды, которая будет использоваться далее.

Для иллюстрации ее применения добавим строку данных в созданную ранее таблицу example.

Предварительно установите привычный формат дат ДД.ММ.ГГГГ:

```
test=# SET DATESTYLE TO GERMAN;
SET
```

Выполните команду:

```
test=# INSERT INTO example(atr1, atr2, atr3, atr4, atr5)
```

VALUES (123, 3.14,'АБВ', true, '13.01.2018');

INSERT 0 1

Обратите внимание, что в качестве разделителя целой и дробной части в действительных числах используется точка, значения текстовых атрибутов и даты заключены в апострофы (клавиша клавиатуры с буквой «Э»), а значения логических атрибутов указываются без сокращений – true (истина) или false (ложь).

Если при вводе команды не допущено синтаксических ошибок, СУБД подтвердит выполнение команды сообщением INSERT 0 1. Чтобы увидеть результат ее работы выполните команду:

```
test=# SELECT * FROM example;
```

В ответ на экран будет выведено содержимое таблицы example.

Команда SELECT далее будет рассмотрена подробно. Здесь заметим, что она может использоваться и для вывода результатов каких-либо вычислений, значений функций и т. д. Например, в случае выполнения арифметических вычислений она выглядит так:

SELECT выражение;

В частности, для вычисления произведения 7.5×3 введите

```
test=# SELECT 7.5 * 3;
```

Результат будет выведен на экран в виде таблицы.

При работе с атрибутами типа date может потребоваться узнать текущую дату. Для ее вызова предназначена функция CURRENT_DATE.

```
test=# SELECT CURRENT_DATE;
```

Рекомендуем внести в таблицу example еще несколько строк. Напоминаем, что команду INSERT можно вновь вызвать на экран клавишами управления курсором и вносить требуемые изменения.

Для удаления строк из таблицы используется команда

DELETE FROM имя_таблицы [WHERE логическое выражение]

Здесь «логическое выражение» – условие, которому должны удовлетворять значения атрибутов в удаляемых строках. При использовании данной команды требуется повышенное внимание. Если логическое выражение будет отсутствовать, то из таблицы удалятся все строки!

В качестве примера удалим из таблицы строку (строки), в которых выполняется условие atr1=123. Выполните команду:

```
test=# DELETE FROM example WHERE atr1=123;
```

Чтобы увидеть результат выполните команду:

```
test=# SELECT * FROM example;
```

Для изменения значений одного или нескольких атрибутов таблицы применяется команда

UPDATE имя_таблицы SET атрибут1 = значение1 [, атрибут2 = значение2] [, ...] [WHERE логическое выражение]

Как и в предыдущей команде, обычно необходимо указывать условие, которому должны удовлетворять значения атрибутов в изменяемых строках. В противном случае значения указанных атрибутов во всех строках таблицы станут одинаковыми. Рекомендуем самостоятельно проверить работу команды UPDATE на таблице example.

1.4. Пример создания базы данных

Создадим учебную базу decanat для учета успеваемости студентов. Она состоит из 3-х таблиц (подчеркнуты ключевые атрибуты)*:

r2(ngr, ggr),
r4(nzk, ngr, adr, fio),
r5(nzk, dis, sem, exm).

Таблица r2 содержит сведения об учебных группах (табл. 2). Она включает 2 атрибута: ngr – номер группы (text), ggr – год формирования группы (integer).

Атрибут ngr является ключевым, т. е. его значения уникальны в каждой строке таблицы.

Таблица 2

Пример таблицы r2

ngr	ggr
00502	2023
80401	2024

Таблица r4 содержит сведения о студентах (табл. 3). Она включает 4 атрибута: nzk – личный номер студента (text), fio – фамилия и инициалы (text), adr – домашний адрес, ngr – номер группы (text).

В данной таблице ключевым атрибутом является nzk. Кроме того, для обеспечения целостности базы данных значения ngr должны выбираться из таблицы r2.

Таблица 3

Пример таблицы r4

nzk	fio	adr	ngr
522	Петров А.К.	Вишневая, 3, к. 6	00502
534	Иванов И.С.	Липовая, 45, к. 65	00502
821	Крылов И.С.	Сосновая, 7, к. 43	80401

* Порядок проектирования этой базы данных рассматривается в курсе лекций

Таблица r5 (табл. 4) содержит сведения о результатах сдачи экзаменов по различным дисциплинам в разных семестрах. Она включает 4 атрибута: dis – наименование дисциплины (text), sem – номер семестра (integer), exm – оценка (integer), nzk – личный номер студента (text).

Таблица 4

Пример таблицы r5

dis	sem	exm	nzk
Математика	1	4	522
Математика	1	5	534
Физика	2	4	534
История	1	5	534
Физика	2	5	821
История	1	4	821
Математика	1	3	821
Математика	3	5	821
Философия	4	4	821

Ключом данной таблицы является совокупность трех атрибутов dis, sem и nzk. При этом значения nzk должны обязательно присутствовать в таблице r4.

PostgreSQL позволяет поддерживать целостность базы данных, следя за уникальностью значений ключей.

Рассмотрим последовательность работы.

Сначала создадим базу данных:

```
postgres=# CREATE DATABASE decanat;
```

Здесь и далее подтверждения PostgreSQL о выполнении команд приводиться не будут.

Сделаем базу данных decanat активной:

```
postgres=# \c decanat
```

Создадим таблицу r2:

```
decanat=#CREATE TABLE r2 (ngr TEXT PRIMARY KEY,  
ggr INTEGER);
```

В данной команде добавлена опция PRIMARY KEY для атрибута ngr, которая говорит о том, что данный атрибут является ключевым, т. е. значения в этом столбце должны быть уникальными, а неопределенные (пустые) значения не допускаются.

Внесем в таблицу конкретные значения согласно табл. 2:

```
decanat=#INSERT INTO r2(ngr, ggr) VALUES('00502', 2022),  
( '80401', 2023);
```

Как видим, команда позволяет вводить данные сразу в несколько строк таблицы. Ее результат позволяет вывести команда:

```
decanat=#SELECT * FROM r2;
```

В качестве эксперимента рекомендуем выполнить ту же команду INSERT еще раз. PostgreSQL сообщит о совпадении значений ключевого атрибута. Данные в таблицу внесены не будут.

Теперь создадим таблицу r4:

```
decanat=#CREATE TABLE r4(nzk TEXT PRIMARY KEY, fio TEXT,  
adr TEXT, ngr TEXT REFERENCES r2(ngr));
```

В этой команде присутствует уже знакомая опция PRIMARY KEY, которая объявляет личный номер ключевым атрибутом. Опция REFERENCES определяет ограничение ссылочной целостности. Иными словами, она показывает, что номера групп ngr в данной таблице должны соответствовать таблице r2.

Сформируйте самостоятельно команду INSERT для добавления к таблице r4 первой строки из табл. 2. Повторите команду с необходимыми изменениями для ввода второй и третьей строки. Результат проконтролируйте с помощью команды

```
decanat=#SELECT * FROM r4;
```

Рекомендуем также убедиться, что при попытке ввода несуществующего номера группы изменение таблицы не происходит.

Создадим таблицу r5:

```
decanat=#CREATE TABLE r5(dis TEXT, sem INTEGER,  
exm INTEGER, nzk TEXT REFERENCES r4(nzk),  
CONSTRAINT pk PRIMARY KEY(dis, sem, nzk));
```

Как и в предыдущем случае, здесь использована опция REFERENCES, с помощью которой устанавливается соответствие личных номеров nzk в таблицах r5 и r4. Опция CONSTRAINT создает ключ с именем pk, состоящий из трех атрибутов dis, sem и nzk. Таким образом, PostgreSQL не позволит внести в таблицу одному и тому же студенту две и более оценок по одной и той же дисциплине в семестре.

Сформируйте самостоятельно команду INSERT для добавления к таблице r5 первой строки из табл. 4. Повторите команду с необходимыми изменениями для ввода остальных строк. Результат проконтролируйте с помощью команды

```
decanat=#SELECT * FROM r5;
```

На этом ввод данных завершается. Для получения списка всех таблиц активной базы данных воспользуйтесь командой \d.

1.4. Выборка данных

Базы данных используются не только для хранения информации, но и для ее использования, что, вероятно, является наиболее важным. Для извлечения требуемых данных используется команда SELECT (выбрать). Ранее мы уже

неоднократно использовали ее простейшую форму. Далее мы рассмотрим наиболее часто используемые виды этой команды с учетом особенностей PostgreSQL.

В общем случае команда SELECT состоит из нескольких частей. Сначала рассмотрим ее «предварительный» синтаксис, который далее позволит лучше понять детали. Вот некоторые элементы (ключевые слова, опции) команды:

SELECT – что выводится (список атрибутов или выражений),

FROM – откуда (источники данных – список таблиц),

WHERE – условиям, которым должны удовлетворять выбираемые данные (логическое выражение),

GROUP BY – атрибуты, по которым осуществляется группировка данных,

HAVING – дополнительное логическое условие группировки,

ORDER BY – как следует упорядочить выбираемые данные.

Обязательными элементами команды являются только два первых – SELECT и FROM.

Далее будут рассмотрены конкретные примеры запросов к созданной базе данных decanat.

Простая проекция

В реляционной алгебре операция проекции представляет собой выборку из таблицы значений некоторых атрибутов без каких-либо условий. Именно эту операцию мы и выполняли ранее. Например, запрос

```
SELECT * FROM r4
```

позволяет извлечь из таблицы r4 все значения всех атрибутов (на это указывает символ *).

Если необходимо извлечь значения лишь некоторых атрибутов, то их нужно просто перечислить сразу за словом SELECT. Например,

```
SELECT r4.fio FROM r4
```

извлекает только фамилии студентов, а

```
SELECT r4.fio, r4.nzk FROM r4
```

фамилии студентов и их личные номера.

В данных примерах перед именем каждого атрибута через точку указано еще и имя таблицы. Поскольку в данном случае выборка осуществляется из одной таблицы, в этом нет необходимости. Тем не менее, рекомендуем указывать имя таблицы всегда, поскольку в более сложных случаях результат запроса может оказаться неправильным.

Для получения ответа на вопрос «По каким дисциплинам студенты сдавали экзамены?» выполните команду:

```
decanat=# SELECT r5.dis FROM r5;
```

В результате на экран будет выведен полный перечень дисциплин (табл. 5). Последовательность дисциплин может быть и иной, но главное то, что результат содержит много повторяющихся записей, что часто не может нас удовлетворить.

Для исключения дублирования используется дополнительная опция DISTINCT. Выполните команду:

```
decanat=# SELECT DISTINCT r5.dis FROM r5;
```

В результате на экран будет выведен полный перечень дисциплин без повторов (табл. 6).

Таблица 5

Перечень дисциплин из
таблицы r5

dis
Математика
Математика
Физика
История
Физика
История
Математика
Математика
Философия

Таблица 6

Перечень дисциплин из таблицы r5
без дублирования

dis
Математика
Физика
История
Философия

Еще одна дополнительная опция AS позволяет изменить в выводимой на экран таблице имя атрибута на любое иное. Попробуйте выполнить команду:

```
decanat=#SELECT DISTINCT r5.dis AS "Дисциплина" FROM r5;
```

Обратите внимание, что наименование колонки таблицы указывается в двойных кавычках.

Наряду с перечнем атрибутов можно выводить и результаты арифметических операций. Например, для пересчета экзаменационных оценок в 100-бальную шкалу их нужно умножить на 20. Это позволяет сделать такая команда:

```
decanat=#SELECT r5.nzk, r5.dis, r5.sem, r5.exm*20 FROM r5;
```

Сортировка

Для сортировки отобранных данных по одному или нескольким признакам используются ключевые слова ORDER BY, за которыми следует перечень атрибутов, по которым будет проводиться упорядочивание данных при их выводе на экран. Например, для вывода фамилий всех студентов по алфавиту нужно выполнить команду

```
decanat=# SELECT r4.fio FROM r4 ORDER BY r4.fio;
```

Если требуется противоположный порядок, то необходимо указать дополнительную опцию DESC. Попробуйте выполнить команду

```
decanat=# SELECT r4.fio FROM r4 ORDER BY r4.fio DESC;
```

Сортировку можно осуществлять и по нескольким атрибутам. Например, команда

```
decanat=#SELECT r5.dis, r5.nzk, r5.exm FROM r5  
ORDER BY r5.dis, r5.nzk;
```

осуществляет сортировку по дисциплинам и по личным номерам студентов.

Предлагаем самостоятельно сформировать команду, которая осуществляет сортировку по дисциплинам и оценкам в порядке их уменьшения.

Выбор по условию

Как ограничить число отображаемых записей? Пусть, например, нас интересуют фамилии студентов только из группы 00502. Для этого необходимо задать логическое условие с помощью опции WHERE, принимающее значения «ложь» или «истина». В результате выполнения такой команды будут выбраны только те данные, которые соответствуют значению «истина». В нашем случае запрос будет таким:

```
decanat=# SELECT r4.fio FROM r4 WHERE r4.ngr ='00502';
```

Напомним, что ранее мы определили атрибут ngr как текстовый. Именно поэтому номер группы заключен в апострофы.

Сложность логического выражения WHERE ничем не ограничена. В нем можно использовать любые знаки отношений: =, <> (или !=), >, >=, <, <=.

При построении логических выражений с использованием текстовых значений нужно иметь в виду некоторые особенности SQL.

Если выполнить команду

```
decanat=# SELECT r4.fio FROM r2 WHERE r4.fio ='П';
```

то на экран не будет выведено ни одной фамилии. Это вполне ожидаемый результат, поскольку в списке отсутствуют студенты с такой короткой фамилией. В то же время команда

```
decanat=# SELECT r4.fio FROM r4 WHERE r4.fio >'П';
```

найдет студента с фамилией Петров, поскольку эта строка длиннее отдельного символа П.

Выполните команду

```
decanat=#SELECT r4.fio FROM r4 WHERE r4.fio >'И';
```

В этом случае на экран будут выведены фамилии всех студентов из нашего списка. Как это можно объяснить?

Очень часто при отборе значений необходимо принимать во внимание сразу несколько логических условий. В таких случаях приходится их комбинировать с помощью логических операторов NOT, AND, OR.

Оператор NOT (не) выполняет логическое отрицание. Результат операции AND (и) будет истинным только тогда, когда истинны оба, связываемые им, выражения. Результат операции OR (или) будет истинным, когда хотя бы одно выражение является истиной.

При использовании логических операций необходимо принимать во внимание их приоритет. Если в выражении нет скобок, то сначала выполняются все операции NOT, затем AND, затем OR.

Использование логического оператора AND позволяет, например, вывести на экран фамилии, начинающиеся на определенную букву. В частности, команда

```
decanat=# SELECT r4.fio FROM r4 WHERE r4.fio >'К'  
AND r4.fio <'Л';
```

выводит единственную фамилию Крылов.

При работе с текстовыми значениями атрибутов может использоваться оператор LIKE, который проверяет их соответствие заданному шаблону. В шаблоне можно использовать знаки «%» (любое количество произвольных символов) или «_» (один символ).

Например, запрос

```
decanat=# SELECT r4.fio FROM r4 WHERE r4.fio LIKE 'И%';
```

позволяет найти фамилии всех студентов, фамилии которых начинаются на букву И.

Предлагаем вам самостоятельно составить запрос с шаблоном с использованием символа «_» и проверить его работу.

Запросы на основе нескольких таблиц

Попробуем ответить на вопрос «Кто сдавал экзамены во втором семестре?». Номера семестров хранятся в таблице r5, из нее можно извлечь личные номера студентов, но их фамилии находятся в таблице r4. Таким образом, необходимо извлечь данные из двух таблиц одновременно.

Попробуйте выполнить команду

```
decanat=# SELECT * FROM r4, r5;
```

Результатом ее выполнения будет довольно большая таблица, которая в реляционной алгебре называется прямым или декартовым произведением. Обратите внимание на принцип ее образования – к каждой строке одной таблицы добавляется каждая строка другой. Лишь некоторые строки этой таблицы содержат ответы на поставленный выше вопрос.

Уменьшить количество выводимых данных можно, принимая во внимание наличие общего атрибута nzk в таблицах r4 и r5:

```
decanat=# SELECT * FROM r4, r5 WHERE r4.nzk=r5.nzk;
```

Эта операция называется соединением. В окончательном виде запрос будет таким:

```
decanat=# SELECT * FROM r4, r5 WHERE r4.nzk=r5.nzk AND r5.sem=2;
```

Примечание. Запросы можно формулировать и в другом виде, указывая соединения с помощью опции JOIN. Предыдущий запрос с ее использованием записывается так:

```
SELECT * FROM r4 JOIN r5 ON r4.nzk=r5.nzk WHERE r5.sem=2
```

Обе формы эквивалентны, но первый способ представляется более наглядным. Поэтому далее будем использовать именно его.

Для иллюстрации работы с тремя отношениями дополним поставленный выше вопрос условием «Кто из этих студентов учится в группах, образованных в 2024 году?». Предлагаем составить запрос SQL самостоятельно. При этом необходимо добавить еще одно условие связи.

Группировка данных

Средства, которые были рассмотрены выше, невозможно применить для решения некоторых важных практических задач. Например, они не позволяют найти ответ даже на такой простой вопрос, как «Сколько студентов учится в каждой группе?». Для выполнения подобных вычислений в SQL используются специальные агрегатные функции. Они предназначены для выполнения вычислений над данными, попавшими в выборку.

Например: функция COUNT позволяет подсчитать количество строк в выборке; функции SUM и AVG вычисляют сумму и среднее значение атрибута, а MAX и MIN – определяют его максимальное и минимальное значение.

Атрибуты, к которым применяются данные функции должны быть объединены в группы с помощью опции GROUP BY. При группировке в одной строке результата размещается единственное значение, вычисленное на основании данных из нескольких строк выборки.

Вот как следует сформировать команду SQL для ответа на поставленный выше вопрос:

```
decanat=# SELECT r4.ngr, COUNT(*) FROM r4 GROUP BY r4.ngr;
```

Для вычисления средней оценки по каждой дисциплине необходимо выполнить запрос

```
decanat=# SELECT r5.dis, AVG(r5.exm) FROM r5 GROUP BY r5.dis;
```

Определить средние оценки по дисциплинам с учетом семестров, в которых они изучались, позволит запрос

```
decanat=# SELECT r5.dis, r5.sem, AVG(r5.exm) FROM r5  
GROUP BY r5.dis, r5.sem;
```

Предлагаем самостоятельно составить запрос для вычисления средних оценок каждого студента за каждый семестр. При этом на экран необходимо вывести не только личные номера, но и фамилии студентов.

Рассмотрим еще один пример, иллюстрирующий действие опции HAVING, которая позволяет наложить условия включения значений атрибутов в группу. Допустим, что нас интересуют фамилии студентов, получивших на экзаменах более одной пятерки. Запрос выглядит так:

```
decanat=# SELECT r4.fio FROM r4, r5 WHERE r4.nzk=r5.nzk  
AND r5.exm=5 GROUP BY r4.fio HAVING COUNT(*)>1;
```

Группировка осуществляется по фамилиям студентов, получившим пятерки, но учитываются лишь те, у которых их количество более единицы.

Подзапросы

Как показано выше, команда SELECT формирует таблицу, которая выводится на экран. В то же время эта команда может быть использована и в других командах языка SQL, точнее – в любом месте, где по смыслу должна находиться таблица. Такая вложенная команда SELECT, заключенная в круглые скобки, называется подзапросом.

Рассмотрим команду с подзапросом, которая позволяет получить ответ на вопрос «Кто и как сдал экзамен по истории?».

```
decanat=# SELECT r4.fio, (SELECT r5.exm FROM r5 WHERE r4.nzk=r5.nzk AND r5.dis='История') FROM r4;
```

Ее результат приведен в табл. 7. Если мысленно исключить из данной команды заключенный в скобки подзапрос, то получим просто список фамилий. Подзапрос формирует таблицу оценок по истории с учетом связи таблиц r4 и r5. Поскольку для студента с фамилией Петров результат не определен (он не сдавал экзамен), соответствующая клетка таблицы пуста.

Таблица 7

Оценки студентов по истории

fio	exm
Петров	
Крылов	4
Иванов	5

Почти аналогичный результат позволяет получить команда:

```
decanat=# SELECT r4.fio FROM r4 WHERE r4.nzk IN (SELECT r5.nzk FROM r5 WHERE r5.dis='История');
```

Она иллюстрирует также использование предиката IN, который позволяет определить содержатся ли значения в таблице, получаемой в подзапросе. В результате будут выведены лишь две фамилии – Крылов и Иванов.

Предлагаем повторить эту команду с использованием отрицания NOT IN. Почему в результате ее выполнения будет выведена лишь одна фамилия – Петров?

1.6. Экспорт и импорт данных

На практике довольно часто возникает необходимость передать результаты выборки или какую-либо таблицу целиком в какое-нибудь другое приложение, например, для их включения в текстовый документ, в таблицу Excel и т. п. Для экспорта данных из таблиц PostgreSQL предназначена команда COPY. У нее довольно много опций, но нам достаточно ограничиться следующим видом:

COPY имя_таблицы TO 'имя_файла' [формат_файла] [DELIMITER
'разделитель'] [HEADER]

В данной команде:

имя_таблицы – наименование экспортируемой таблицы активной базы данных;

имя_файла – наименование файла, в котором будет сохранена таблица, с указанием пути к нему;

формат_файла – формат записи файла; по умолчанию текстовый, допускается csv (англ. Comma-Separated Values – значения, разделенные запятыми);

DELIMITER – разделитель – задает символ, разделяющий столбцы в строках файла; по умолчанию это символ табуляции в текстовом формате и запятая в формате csv;

HEADER – при экспорте в первую строку файла будут записаны имена атрибутов таблицы.

Далее на конкретном примере рассмотрим применение команды COPY для экспорта данных в текстовые файлы типа csv.

Формат csv предназначен для передачи табличных данных между различными приложениями. На самом деле разделителем значений в каждой строке может быть символ табуляции, точка с запятой и др. Например, представленная выше табл.2 в таком формате с точками с запятой в качестве разделителей выглядит так:

```
nzk;fio;adr;ngr  
522;Петров А.К.;Вишневая, 3, к. 6;502  
534;Иванов В.С.;Липовая, 45, к.65;502  
821;Крылов И.С.;Сосновая, к. 43;80401
```

Первая строка этого текста содержит наименования атрибутов, а остальные строки соответствуют строкам таблицы.

Допустим, что необходимо экспортировать таблицу r5 в файл export.csv на внешнем устройстве «e:/» (флеш-карте). Эту операцию выполняет команда:

```
COPY r5 TO 'e:/export.csv' CSV DELIMITER ';' HEADER
```

Здесь указано, что разделителем является точка с запятой. Вместо нее можно указать и любой иной символ, если есть уверенность, что он отсутствует среди символов в значениях экспортируемых атрибутов. Файлы формата csv можно открыть с помощью электронной таблицы (рис. 1).

Обратите внимание, что атрибут nzk, хранящийся в базе данных как текст, воспринимается Excel как числовой. Об этом говорит выравнивание значений по правому краю ячеек (сравните с текстовым атрибутом dis).

С помощью команды COPY можно экспортировать результаты работы команды SELECT (подзапрос). Для этого всю команду необходимо указать в круглых скобках вместо имени таблицы. Например, команда:

```
COPY (SELECT * FROM r5) TO 'e:/export.csv' CSV DELIMITER ';' HEADER
```

абсолютно идентична предыдущей.

	A	B	C	D
1	dis	sem	exm	nzk
2	Математика	1	4	522
3	Математика	1	5	534
4	Математика	1	3	821
5	Физика	2	4	534
6	История	1	5	534
7	Физика	2	5	821
8	История	1	4	821
9	Математика	3	5	821
10	Философия	4	4	821
11				

Рис. 1. Файл export.csv в среде Excel

Команда COPY используется и при импорте данных в уже имеющуюся таблицу. В этом случае она имеет следующий формат:

COPY имя_таблицы (список атрибутов) FROM 'имя_файла' [формат_файла] [DELIMITER 'разделитель'] [HEADER]

2. Работа в среде клиента pgAdmin III

PgAdmin – графическое средство для администрирования PostgreSQL*. Это приложение упрощает основные задачи администрирования, отображает объекты баз данных, позволяет выполнять запросы SQL. В данном пособии рассматривается версия pgAdmin III. Запуск приложения осуществляется из меню Пуск.

В левой части основного окна расположен *Навигатор* (рис. 2), позволяющий обратиться к любому объекту базы данных. В качестве примера здесь показан вид *Навигатора* после создания двух баз данных test и decanat, рассмотренных ранее. База postgres создается автоматически при установке СУБД.

Последовательно раскрывая пункты *Навигатора* можно увидеть иерархическую структуру баз данных. На рис. 3 показаны объекты базы данных decanat, в том числе и созданные ранее таблицы.

Чтобы внести изменения в какую-либо таблицу достаточно выделить ее в окне Навигатора и выбрать пункт инструмент Редактирование на панели (рис. 4). Обратите внимание, что в заголовочной части таблицы указываются типы атрибутов и ключи (PK).

Для формирования запроса необходимо вызвать одноименный пункт меню (рис. 5) и ввести его в открывшемся окне. В нашем примере показан запрос «Кто и как сдавал экзамены во 2-ом семестре?». Для его выполнения необходимо выбрать соответствующий пункт меню окна Query.

* Основное отличие более современной версии pgAdmin IV заключается в том, что данное приложение работает в среде браузера.

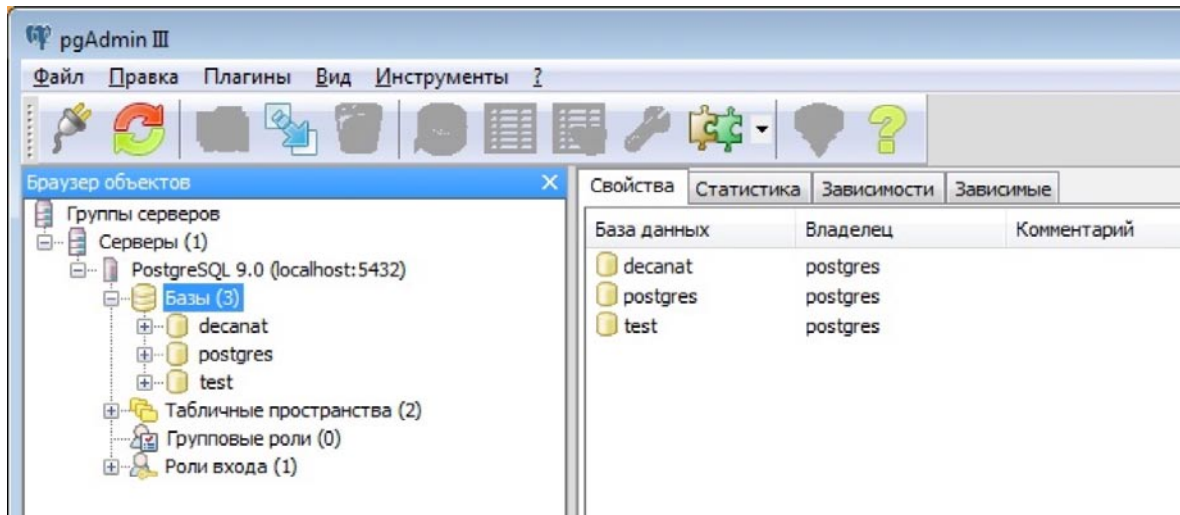


Рис. 2. Основное окно pgAdmin III

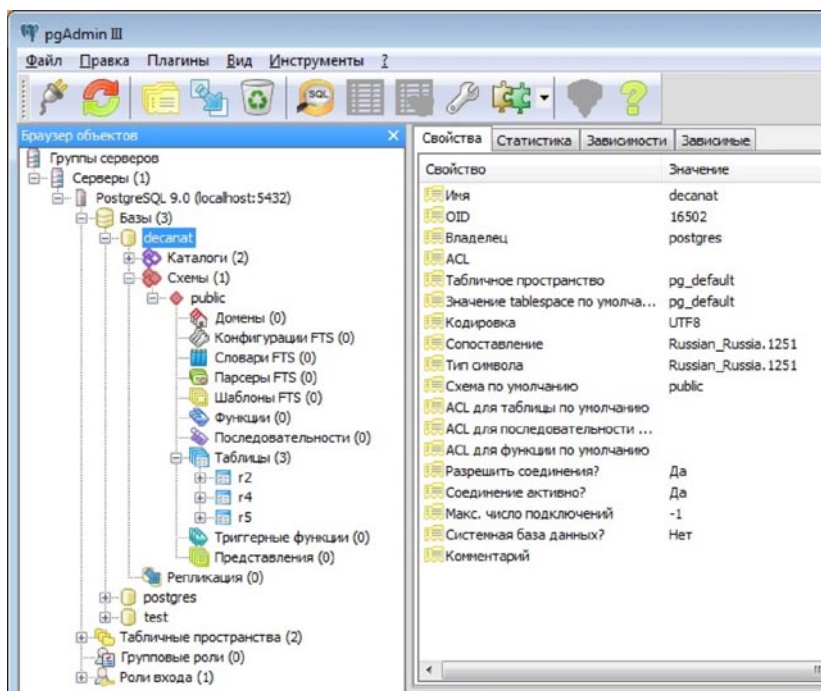


Рис. 3. Объекты базы данных decanat

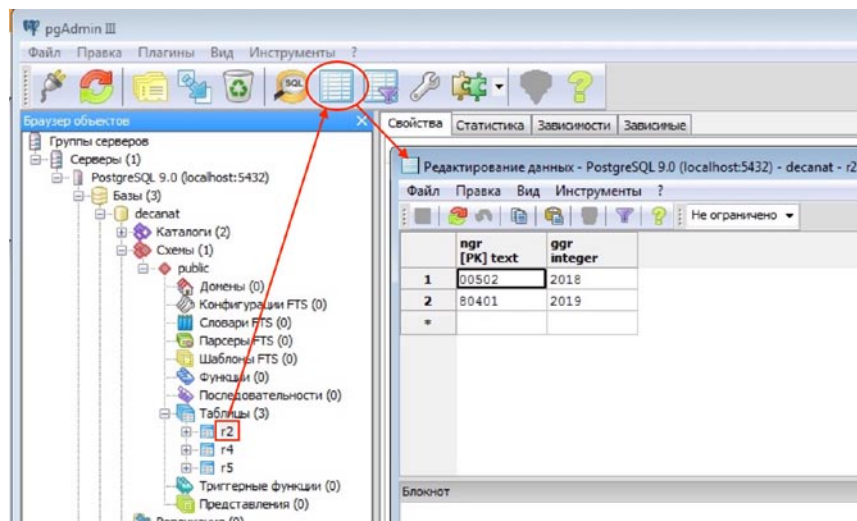


Рис. 4. Окно редактирования таблицы

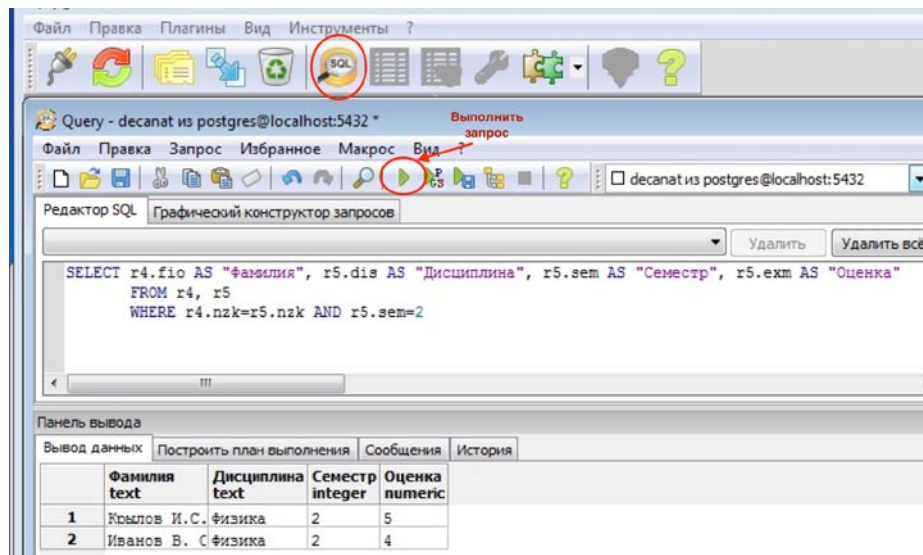


Рис. 5. Окно запросов *Query*

Рассмотрим последовательность действий для создания новой базы данных и ее таблиц.

Для создания новой базы данных воспользуйтесь соответствующим пунктом контекстного меню (правая клавиша мыши). В открывшемся окне укажите ее имя (в нашем примере test2) и владельца postgres (рис. 6).

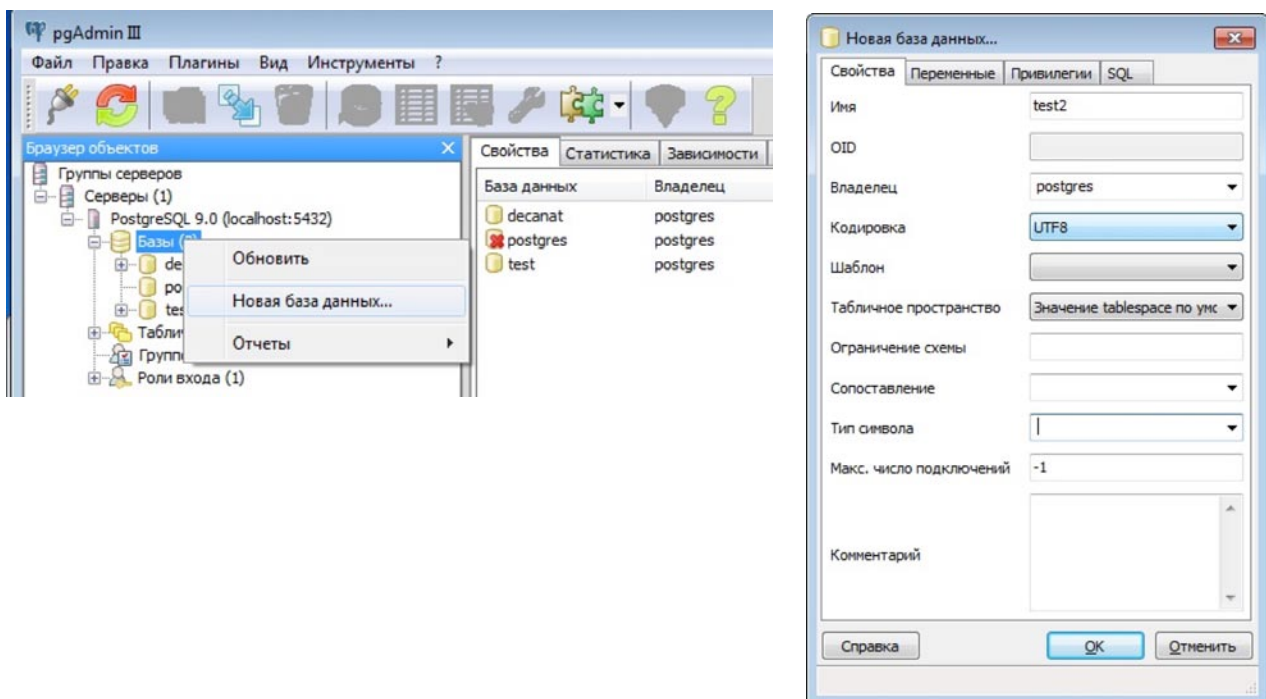


Рис. 6. Создание новой базы данных

В окне *Навигатора* раскройте содержимое базы данных test2 и в контекстном меню активизируйте пункт *Новая таблица* (рис. 7). На вкладке *Свойства* окна *Новая таблица* задайте ее имя (в данном примере r10).

Для создания атрибутов таблицы перейдите на вкладку *Колонки* и нажмите кнопку *Добавить*. Введите имя колонки и тип данных (выбирается из раскрывающегося списка). В качестве примера на рис. 8 показан состав таблицы r10 с двумя колонками.

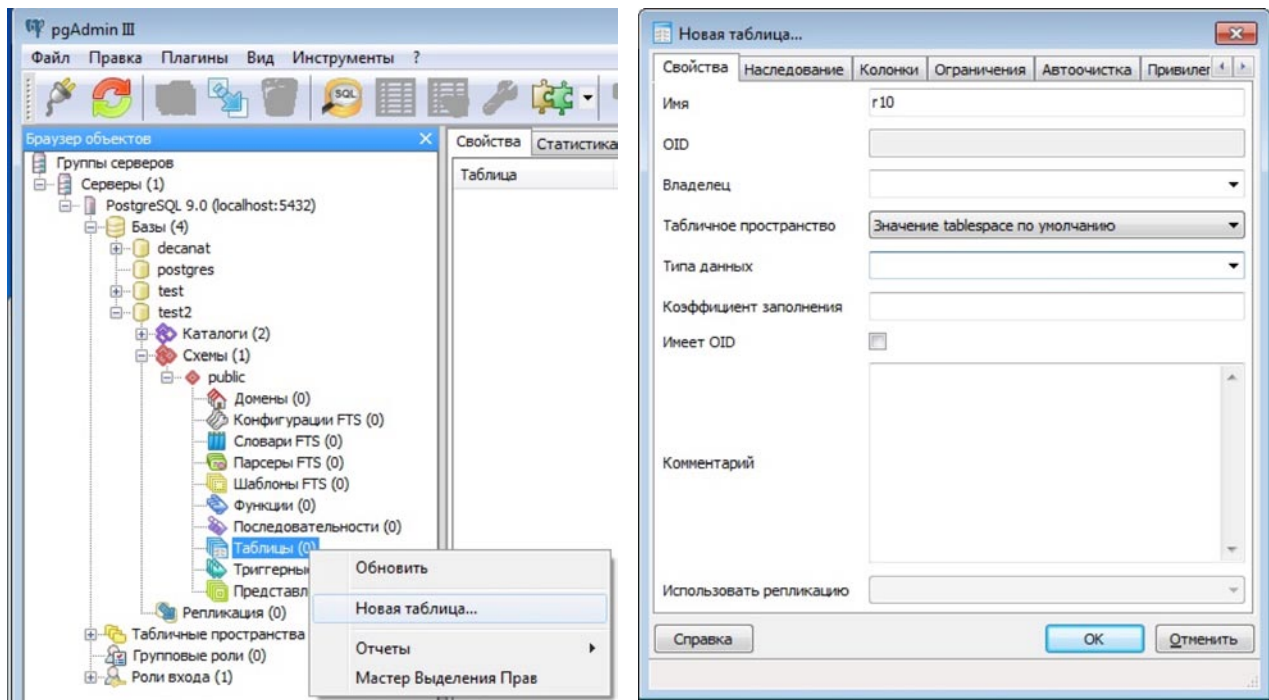


Рис. 7. Создание новой таблицы

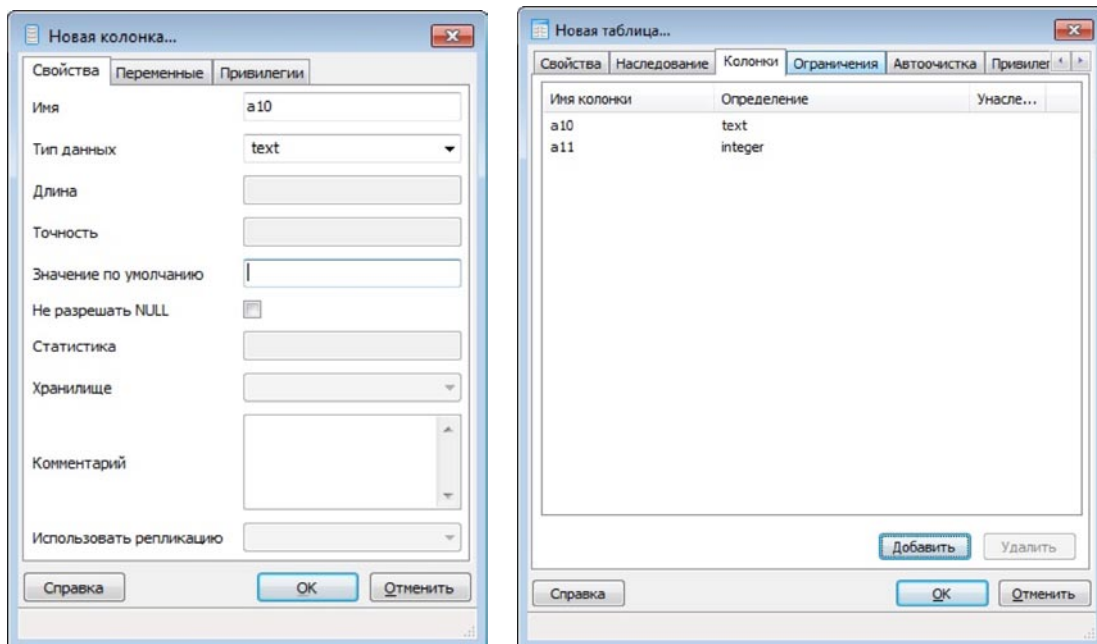


Рис. 8. Создание колонок (атрибутов) таблицы

Для каждой вновь создаваемой таблицы необходимо указать первичный ключ. Им может быть один или несколько атрибутов таблицы, совокупность значений которых уникальны для каждой строки.

Для создания первичного ключа перейдите на вкладку *Ограничения* и нажмите кнопку *Добавить* (рис. 9). В открывшемся окне введите произвольное имя ключа. Обратите внимание, что имя ключа нужно указать даже в том случае, если он состоит только из одного атрибута. . Если ключ не задан, то вы не сможете ввести данные в таблицу.

В нашем случае имя первичного ключа a11_k. Он будет создан на основе единственного атрибута a11. Для указания ключевых атрибутов перейдите на

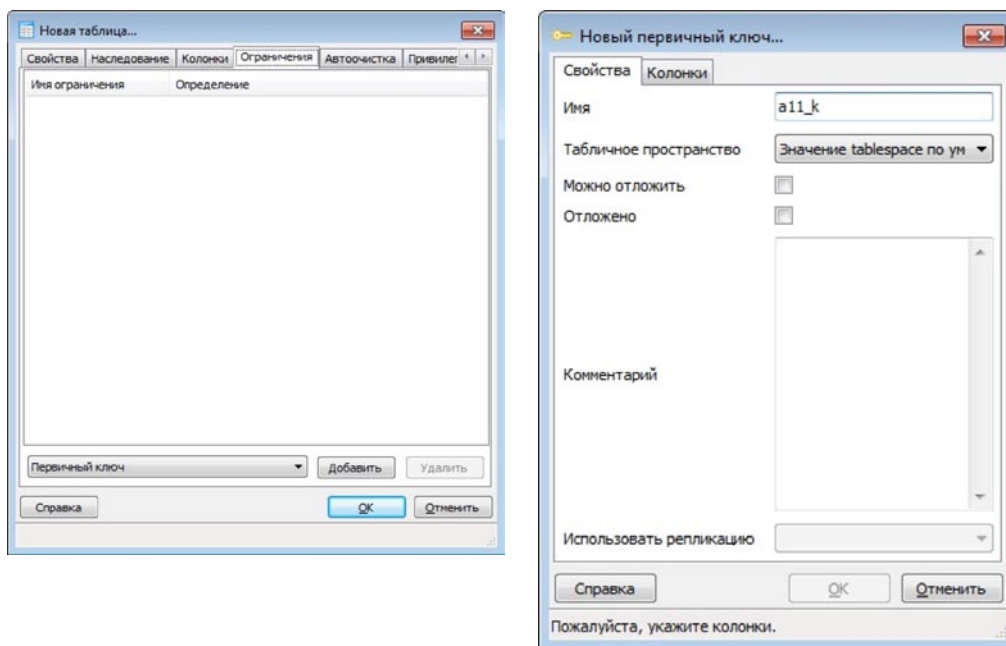


Рис. 9. Создание первичного ключа

вкладку *Колонки* окна *Новый первичный ключ* и выберите необходимые атрибуты из раскрывающегося списка ранее введенных колонок (рис. 10). Если ключ состоит из нескольких атрибутов нажмите кнопку *Добавить*.

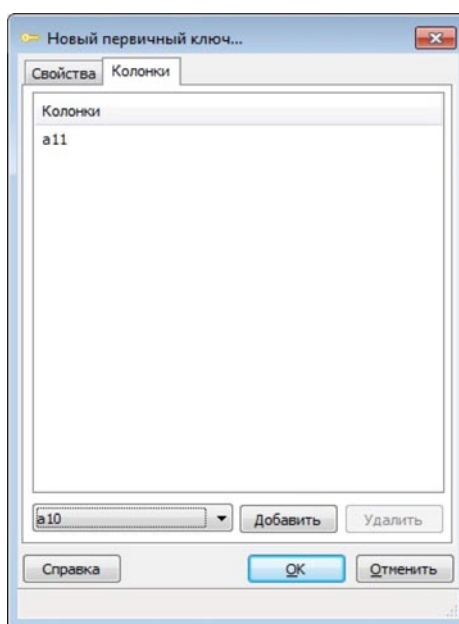


Рис.10. Окно выбора ключевых атрибутов

С целью обеспечения доступности данных целесообразно периодически сохранять базу на резервном носителе. Для создания резервной копии какой-либо базы данных, выберите ее в окне *Навигатора* и воспользуйтесь соответствующим пунктом контекстного меню (рис. 11). В открывшемся окне укажите внешнее устройство (в нашем примере E:\) и произвольное имя файла-копии.

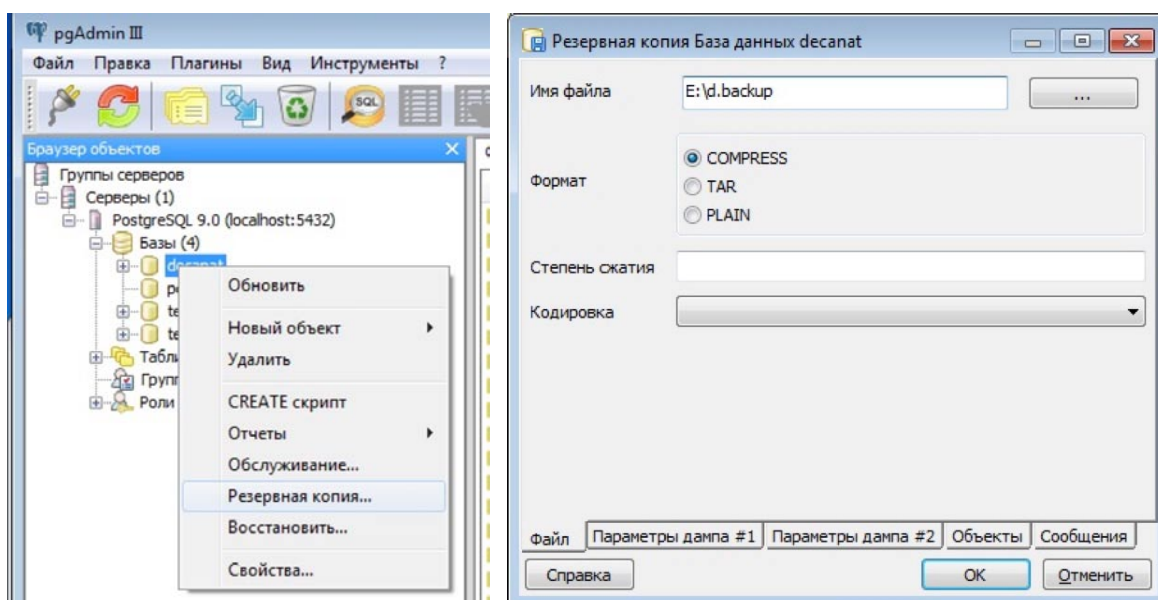


Рис. 11. Создание резервной копии базы данных

Для восстановления базы данных из резервной копии воспользуйтесь соответствующим пунктом контекстного меню *Навигатора*.

3. Использование средств искусственного интеллекта для создания запросов

Сейчас в Интернете можно найти много средств искусственного интеллекта (ботов нейросетей), позволяющих создавать SQL-запросы*. Независимо от используемого инструмента, качество получаемого результата существенно зависит от постановки задачи. Как минимум, необходимо описать структуру базы данных, а в ряде случаев привести примеры используемых таблиц.

Рассмотрим одно из таких средств – Wand.Tools. Используя таблицы базы данных decanat, попробуем с его помощью получить ответы на некоторые вопросы, поставленные выше.

Один из возможных вариантов постановки задачи для ответа на задание «Найти фамилии студентов групп образованных в 2024 году» показан на рис. 12. После нажатия кнопки *Сгенерировать SQL* довольно быстро будет получен текст запроса:

```
SELECT r4.fio FROM r4 JOIN r2 ON r4.ngr = r2.ngr
```

Ответ на задание «Определить средние оценки каждого студента за каждый семестр. Вывести заголовки колонок по-русски» выглядит следующим образом:

```
SELECT r4.fio AS 'Фамилия', r5.sem AS 'Семестр', AVG(r5.exm) AS 'Средняя
оценка' FROM r4 JOIN r5 ON r4.nzk = r5.nzk
GROUP BY r4.fio, r5.sem;
```

* Необходимо иметь в виду, что обычно эти средства имеют ограничения бесплатного использования по времени или по количеству символов, вводимых в чате.

Генератор SQL-запросов

Преобразуйте ваши текстовые инструкции в SQL-запросы или введите запрос, чтобы получить его объяснение.

Сгенерировать SQL Объяснить SQL

```
имеется три таблицы  
таблица r2 (ngr, ggr)  
ngr — уникальный номер учебной группы;  
ggr — год ее формирования  
  
таблица r4 (nzk, fio, adr, ngr)  
nzk — уникальный личный номер;  
fio — фамилия;  
adr — адрес  
  
таблица r5 (dis, sem, exm, nzk)  
dis — дисциплина;  
sem — семестр;  
exm — экзаменационная оценка  
Найти фамилии студентов групп образованных в 2024 году
```

Сгенерировать SQL

Рис. 12. Рабочий экран Wand.Tools
<https://wand.tools/sql/ru/> (дата доступа 10.11.2025)

В любом случае ответ бота следует проверить и при необходимости откорректировать с учетом особенностей используемой версии SQL. Например, в последнем запросе нужно заменить апострофы на двойные кавычки.

4. Задание для самостоятельной работы

Постановка задачи

База данных содержит сведения о деталях (узлах) самолетов, эксплуатируемых авиакомпанией, и их количестве на складах, расположенных в различных городах. База данных состоит из трех таблиц.

Таблица *det*, фрагмент которой показан в табл. 8, включает следующие атрибуты:

NP – уникальный номер (идентификатор) детали (узла) самолета, *text*;

NA – наименование детали (узла), *text*;

ND – номер (идентификатор) узла, частью которого является данная деталь, *text*.

Таблица 8

Фрагмент таблицы *det*

NP	NA	ND
211	0	Кресло
2114	211	Чехол
2116	211	Ремень
21163	2116	Пряжка
21164	2116	Крепление
206	0	Панель
2061	206	Кнопка
2066	206	Выключатель
2068	206	Вентилятор

Таблица *prs* (табл. 9) включает следующие атрибуты:

NP – уникальный номер (идентификатор) детали (узла) самолета, *text*;

AD – город, в котором расположен склад деталей, *text*;

QU – количество данных деталей на складе в городе, *integer*.

Таблица 9

Фрагмент таблицы *prs*

NP	AD	QU
211	Москва	106
211	Екатеринбург	28
211	Новосибирск	77
2114	Москва	6
2114	Новосибирск	28
2116	Екатеринбург	314
2116	Новосибирск	29
21164	Ростов	365

Таблица use (табл. 10) включает следующие атрибуты:

NP – уникальный номер (идентификатор) детали (узла) самолета, text;

TP – тип самолета, text;

NB – количество данных деталей (узлов), необходимых для самолета данного типа, integer.

Таблица 10

Фрагмент таблицы use

NP	TP	NB
211	A-320	180
211	A-330	406
2114	A-320	180
2114	A-330	406
2116	A-320	108
2116	A-330	406
21164	A-320	621
21164	A-330	734

Данные табл. 8-10 представлены также в виде текстовых файлов det.csv, prs.csv и use.csv (предоставляются преподавателем).

Задание

Создайте базу данных с произвольным именем.

Определите ключевые атрибуты и создайте указанные выше таблицы с соответствующими ключами.

Введите в таблицы любым способом данные из файлов det.csv, prs.csv, use.csv.

Составьте SQL-запросы для ответов на следующие вопросы:

1. В каких городах (без дублирования названий) расположены склады деталей (узлов)?

2. Какие детали (название) есть хотя бы в одном городе?

3. Какие детали (название) и в каких количествах входят в конструкцию самолетов каждого типа?

4. В каких городах имеются чехлы и в каком количестве?

5. Сколько ремней требуется для самолета каждого типа?

6. Сколько кресел в самолете А-320?

При вводе символа «А» возможно использование латинской и русской раскладки клавиатуры.

7. Сколько чехлов на складе в Новосибирске?

8. Какие детали (названия) имеются на складах в Москве и Екатеринбурге?

9. Какие детали (названия) и в каких городах имеются в количестве, достаточном для самолета А-320?

10. Определить количество городов, в которых имеются одинаковые детали (вывести их названия).

11. Сколько всего деталей каждого вида есть на складах?

12. Каких деталей меньше всего в каждом городе? Вывести их названия и количество.

При составлении запросов необходимо принять во внимание, что идентификаторы деталей пользователь не знает. Результат работы каждого запроса должен быть проиллюстрирован соответствующими таблицами, с заголовками колонок на русском языке. При выполнении задания допускается использование средств искусственного интеллекта.

Библиографический список

1. Лузганов П. В. Postgres. Первое знакомство / П. В. Лузганов, Е. В. Рогов , И. В. Левшин. – М.:Простгрес Профессиональный, 2023. Электронное издание: <https://postgrespro.ru/education/introbook> (дата доступа 10.11.2025)

2. Маркин А. В. Построение запросов и программирование на SQL: учеб. пособие для студентов вузов, обучающихся по специальности 230201 “Информ. системы и технологии” / А. В. Маркин. – М.: Диалог-МИФИ, 2008 .— 320 с.

3. Кара-Ушанов В. Ю. SQL – язык реляционных баз данных : учебное пособие для студентов, обучающихся по направлению подготовки 230100 “Информатика и вычислительная техника”, 230400 “Информационные системы и технологии” / В. Ю. Кара-Ушанов – Екатеринбург : Издательство Уральского университета, 2016. – 156 с.

4. Хернандес М. Д. SQL-запросы для простых смертных : Практическое руководство по манипулированию данными в SQL / М. Д. Хернандес, Д. Л. Вьескас; пер. с англ. М.: Лори, 2003 . – 460 с.

5. Тейлор А. SQL для “чайников” / А. Д. Тейлор ; пер. с англ. М.: Диалектика, 2001 .— 368 с.

6. Форта Б. Освой самостоятельно SQL. 10 минут на урок / Бен Форта; пер. с англ. – М.: Вильямс, 2006 .— 282 с.

7. Моисеенко С. И. SQL. Задачи и решения / С. И. Моисеенко .— СПб. : Питер, 2006 . – 255 с.

8. Уилтон П. SQL для начинающих / П. Уилтон, Д. Колби; пер. с англ. М.: Диалектика, 2006 .— 487 с.

Текстовый электронный образовательный ресурс

Некрасов Александр Васильевич
Пастухова Лилия Германовна

Принципы использования системы управления базами данных PostgreSQL

Учебно-методическое пособие для самостоятельной работы

Компьютерная верстка *А. В. Некрасова*

Рекомендовано Методическим советом УрФУ

Протокол №__ от _____2025
Электронный формат – pdf



<https://study.urfu.ru>