## **Создание и связывание таблиц базы данных в среде PostgreSQL**

**Основные сведения**

Рассмотрим следующие вопросы:

* создание и выбор базы данных;
* создание таблиц;
* столбцы и типы данных в PostgreSQL;
* создание индексов;
* удаление таблиц, индексов и баз данных;
* изменение структуры таблиц.

Базы данных, таблицы и индексы создаются в рамках графического ин­терфейса PostgreSQL – pgAdmin 4, но **мы будем использовать монитор PostgreSQL (клиент командной строки) –psql**, чтобы лучше понять структуру БД, таблиц и индексов.

***Чувствительность к регистру и идентификаторы****.*

* Имена БД подчиняются тем же правилам зависимости от регистра символов, каким следуют каталоги операционной системы. Имена таблиц следуют тем же правилам, что и имена файлов. Все остальное не зависит от регистра.
* Все идентификаторы, кроме имен псевдонимов, могут содержать до 64 сим­волов. Имена псевдонимов могут иметь до 255 символов.
* Идентификаторы могут содержать любые допустимые символы, но имена баз данных не могут содержать символы /, \ и . , а имена таблиц – символы . и /.
* Зарезервированные слова можно использовать для идентификаторов, если заключить их в кавычки.

***Комментарий в PostgreSQL****.* Аналогично SQL, начинается с двух дефисов (--), за которыми должен следовать пробел. Кроме того, PostgreSQL содержит ряд собственных комментариев. Shell-комментарий # действует аналогично – все, что расположено правее его, является текстом комментария. Комментарий /\* \*/ является многострочным – комментарий начинается с /\* и заканчивается, когда встретится завершение \*/.

***Примечание****.*

Данное методическое указание было составлено на версии PostgreSQl 15.6

Стоит учитывать, что при работе с psql в среде Windows можно столкнуться с проблемой отображения кириллицы. Способ решения данной проблемы:

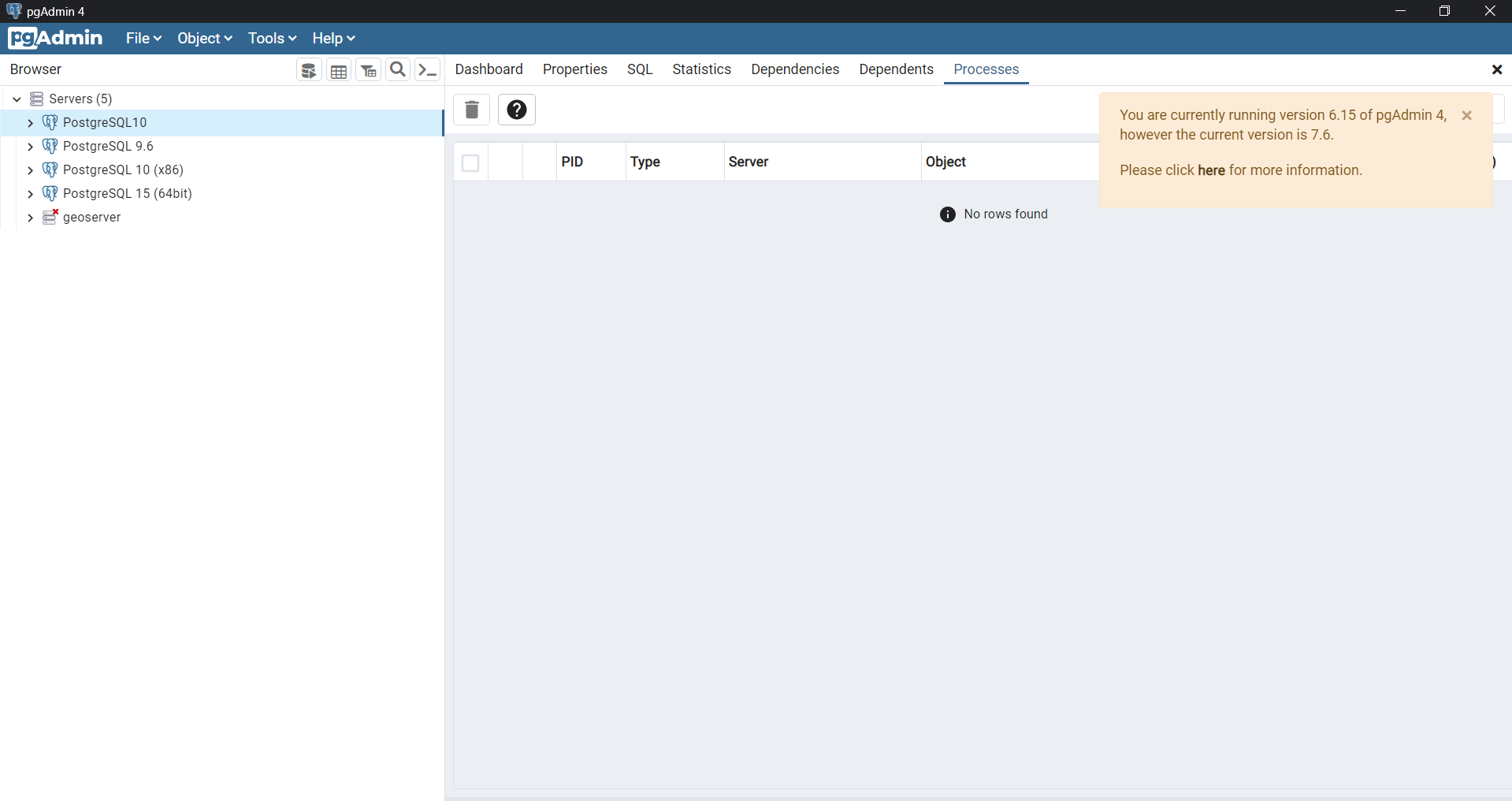
*\! chcp 1251*

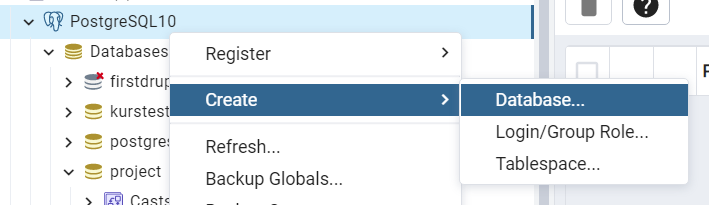
(каждую новую сессию нужно вводить заново).

***Создание и выбор базы данных.***

Работать можно в консольной утилите psql и в приложении с графическим интерфейсом (вводимые команды могут отличаться).

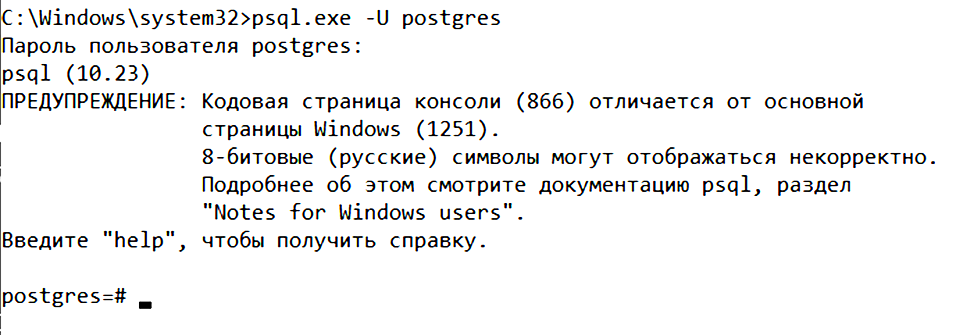
Для наглядности, pgAdmin 4 выглядит следующим образом:



И с помощью него создать базу данных выбрав нужный сервер:  


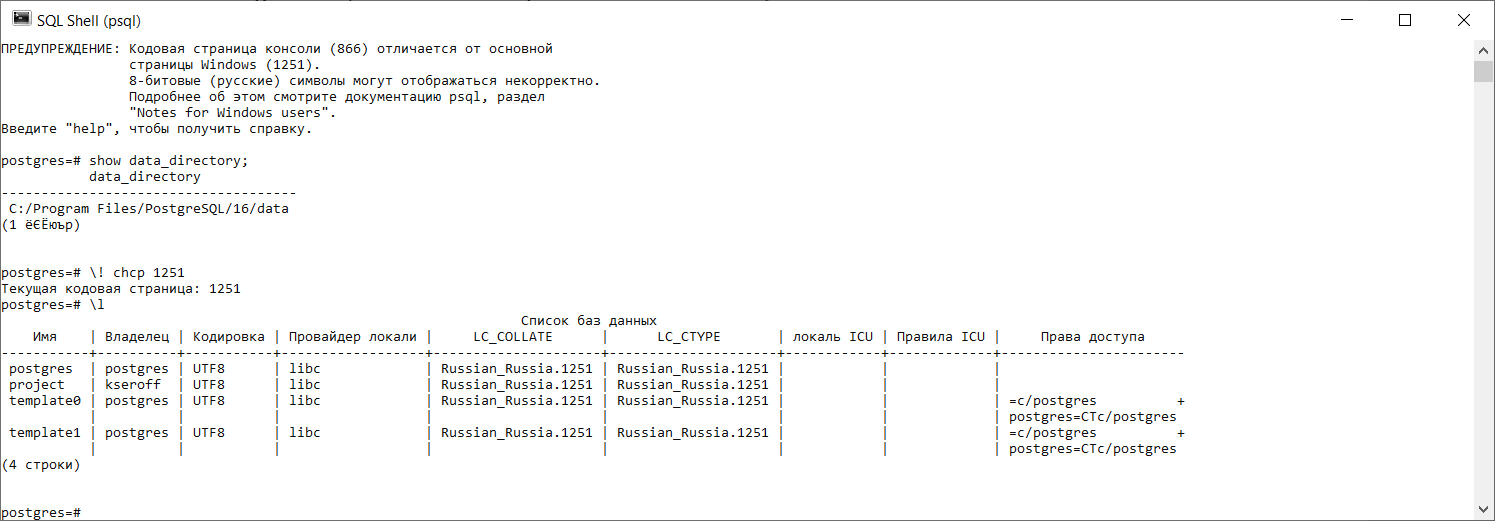
***Далее будем использовать консоль***. Открываем её, первые четыре строчки можно пропустить (оставить по умолчанию), нажав Enter, далее требуется ввести пароль (**ВАЖНО! Символы, введенные в сточку пароля, не видимы, введите пароль и нажмите Enter**), далее можно приступить к созданию БД.

-U postgres



Базы данных доступных в данный момен до создания новой:

Postgres=# \l



Создание БД осуществляется с помощью оператора

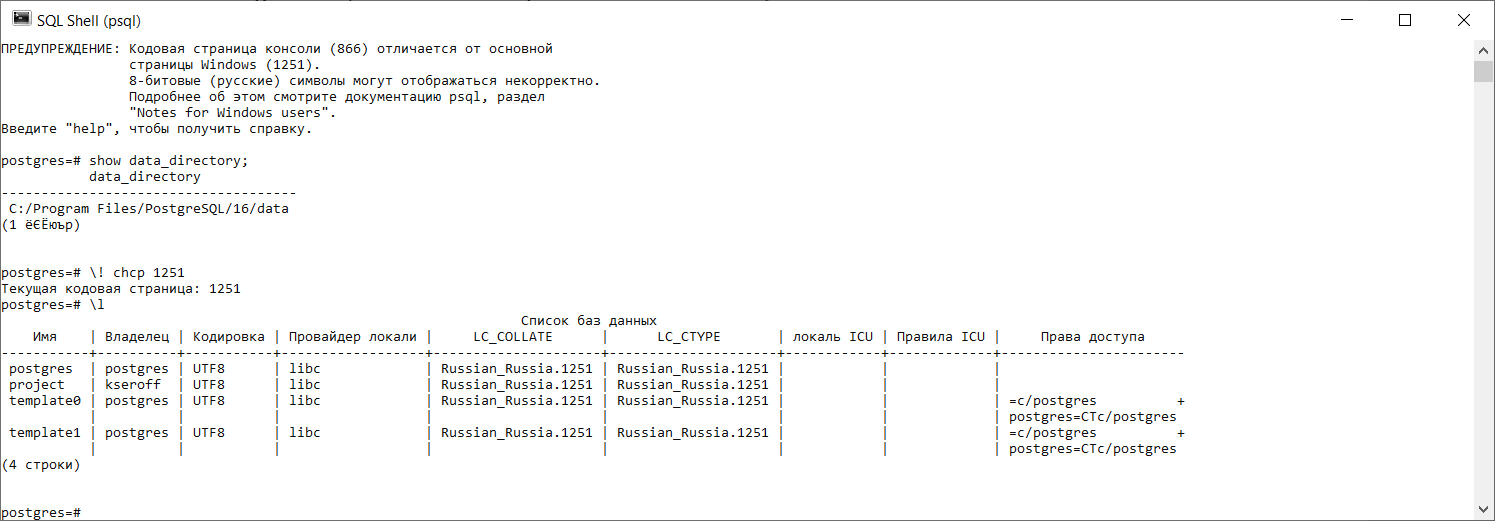
*create database имя\_базы\_данных owner имя\_админа(superuser) encoding кодировка template название\_шаблона;*

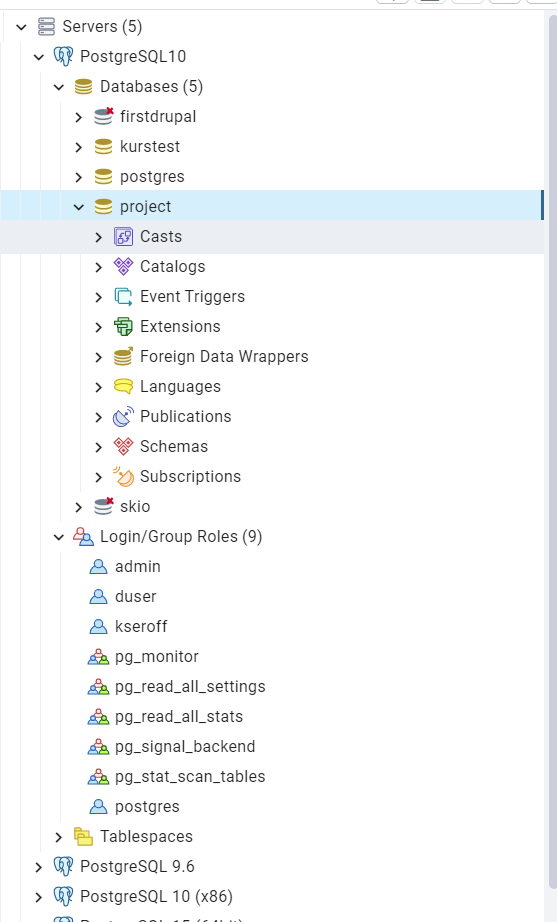
*create database project owner postgres template template0;*

**

Убедиться в том, что оператор выполнил задачу, можно с помощью оператора list – “\l”;

Он выведет все доступные БД.

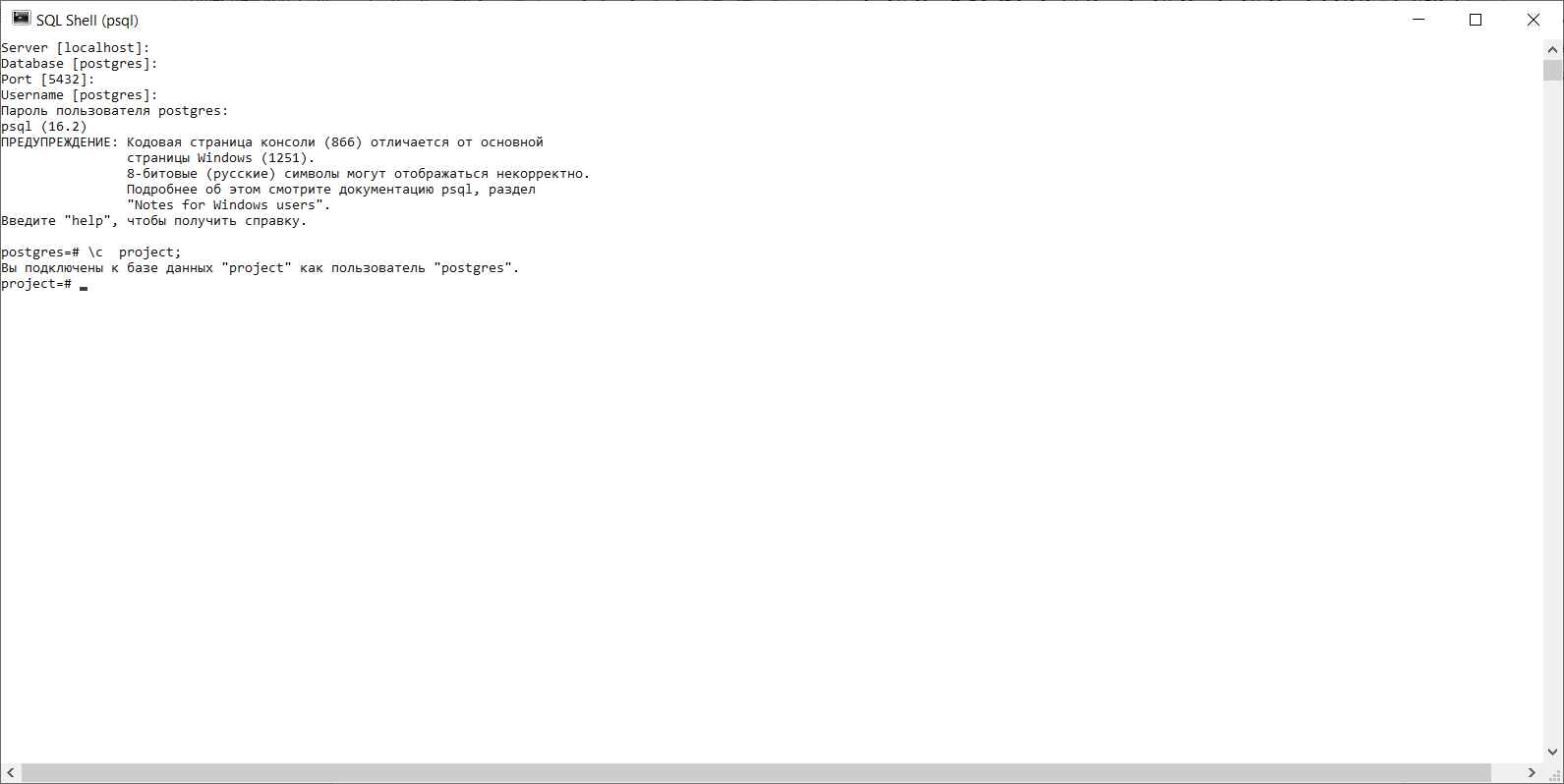
  
Также проверить правильно ли было создано можно в pg\_admins4:



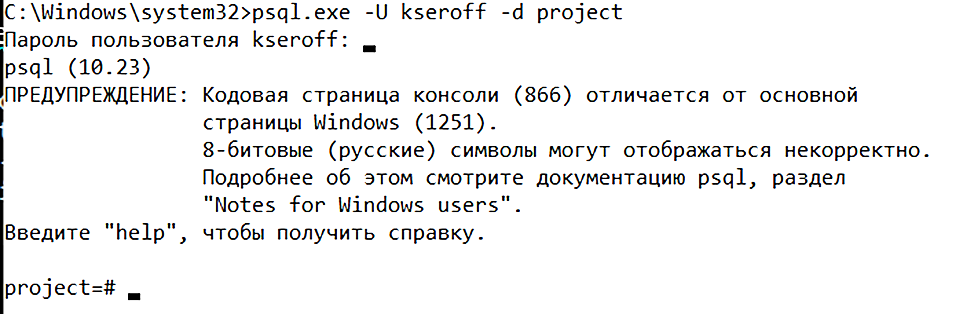
Теперь имеется пустая БД, ожидающая создания таблиц. Прежде чем работать с БД, необходимо выбрать эту БД с помощью оператора

*\c имя\_базы\_данных;*

*\c project;*



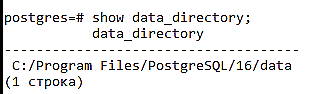
Или можно зайти в базу данных выйдя с помощью команды *\q ,*прописав следом *–psql.exe –U имя\_админа –d имя\_базы\_данных  
psql.exe –U kseroff –d project*

**

Теперь все действия по умолчанию будут применяться именно к этой БД.

Посмотреть, где физически хранится база данных можно с помощью оператора

show data\_directory;

****

**Создание таблиц**[3]**.**

Команда CREATE TABLE используется для создания новой пустой таблицы в текущей базе данных. При выполнении этой команды, владельцем таблицы будет пользователь, который ее создал. Если указано имя схемы (например, CREATE TABLE myschema.mytable ...), то таблица создается в указанной схеме, в противном случае она создается в текущей схеме. Временные таблицы, в свою очередь, существуют в специальной схеме, и поэтому при их создании нельзя указывать имя схемы.

Имя таблицы должно быть уникальным и отличаться от имен других отношений в этой схеме, таких как таблицы, последовательности, индексы, представления, материализованные представления или сторонние таблицы. При использовании команды CREATE TABLE будет также автоматически создан составной тип данных, соответствующий одной строке таблицы. Поэтому имя таблицы не может совпадать с именем существующего типа данных в данной схеме.

Для эффективного определения множества допустимых значений в таблице при добавлении или изменении строк возможно использование необязательных предложений, называемых ограничениями. Ограничения являются SQL-объектами, которые задают определенные правила проверки данных.

Ограничения могут быть определены двумя способами: в виде ограничений таблицы и в виде ограничений столбца. Ограничение столбца является частью определения столбца и применяется только к этому столбцу. Ограничение таблицы, в свою очередь, не является привязанным к конкретному столбцу и может охватывать несколько столбцов таблицы.

Более того, в целях удобства, особенно когда ограничение применяется только к одному столбцу, возможно записать любое ограничение столбца в виде ограничения таблицы.

Чтобы создать таблицу, необходимо иметь право USAGE для типов всех столбцов или типа в предложении OF, соответственно.

TEMPORARY или TEMP #

С таким указанием таблица создаётся как временная. Временные таблицы автоматически удаляются в конце сеанса или могут удаляться в конце текущей транзакции.

UNLOGGED #

С этим указанием таблица создаётся как нежурналируемая. Данные, записываемые в нежурналируемые таблицы, не проходят через журнал предзаписи.

IF NOT EXISTS #

Не считать ошибкой, если отношение с таким именем уже существует. В этом случае будет выдано замечание. Заметьте, что нет никакой гарантии, что существующее отношение как-то соотносится с тем, которое могло бы быть создано.

LIKE исходная\_таблица [ вариант\_копирования ... ] #

Предложение LIKE определяет таблицу, из которой в новую таблицу будут автоматически скопированы все имена столбцов, их типы данных и их ограничения на NULL.

PRIMARY KEY (ограничение столбца)

PRIMARY KEY ( имя\_столбца [, ... ] ) [INCLUDE ( имя\_столбца [, ...])] (ограничение таблицы)#

Ограничение PRIMARY KEY определяет, что столбец или столбцы таблицы могут содержать только уникальные (без повторений) значения, отличные от NULL. Для таблицы может быть задан только один первичный ключ, будь то ограничение столбца или ограничение таблицы.

EXCLUDE [ USING индексный\_метод ] ( элемент\_исключения WITH оператор [, ... ] ) параметры\_индекса [ WHERE ( предикат ) ] #

Предложение EXCLUDE определяет ограничение-исключение, которое гарантирует, что для любых двух строк, сравниваемых по указанным столбцам или выражениям с указанными операторами, результат не будет равен TRUE для всех сравнений.

REFERENCES внешняя\_таблица [ ( внешний\_столбец ) ] [ MATCH тип\_совпадения ] [ ON DELETE ссылочное\_действие ] [ ON UPDATE ссылочное\_действие ] (ограничение столбца)

FOREIGN KEY ( имя\_столбца [, ... ] ) REFERENCES внешняя\_таблица [ ( внешний\_столбец [, ... ] ) ] [ MATCH тип\_совпадения ] [ ON DELETE ссылочное\_действие ] [ ON UPDATE ссылочное\_действие ] (ограничение таблицы) #

Эти предложения определяют ограничение внешнего ключа, требующее, чтобы группа из одного или нескольких столбцов новой таблицы содержала только такие значения, которым соответствуют значения в заданных столбцах некоторой строки во внешней таблице. Если список целевых\_столбцов опущен, в качестве него используется первичный ключ целевой\_таблицы. В качестве целевых столбцов должны указываться столбцы неоткладываемого уникального ограничения или первичного ключа во внешней таблице. При этом пользователь должен иметь право REFERENCES во внешней таблице (либо для всей таблицы, либо только для целевых столбцов). Для добавления ограничения внешнего ключа требуется блокировка SHARE ROW EXCLUSIVE в целевой таблице. Заметьте, что нельзя определить ограничения внешнего ключа, связывающие временные и постоянные таблицы.

SET NULL [ ( имя\_столбца [, ... ] ) ] #

Установить во всех ссылающихся столбцах или в указанном подмножестве ссылающихся столбцов значения null. Подмножество столбцов можно указать только для действий ON DELETE.

SET DEFAULT [ ( имя\_столбца [, ... ] ) ] #

Установить во всех ссылающихся столбцах или в указанном подмножестве ссылающихся столбцов значения по умолчанию. Подмножество столбцов можно указать только для действий ON DELETE. (Если значения по умолчанию отличны от NULL, во внешней таблице должна быть строка, соответствующая набору этих значений; в противном случае операция завершится ошибкой.)

Если внешние столбцы меняются часто, будет разумным добавить индекс для ссылающихся столбцов, чтобы действия по обеспечению ссылочной целостности, связанные с ограничением внешнего ключа, выполнялись более эффективно.

USING метод [#](https://postgrespro.ru/docs/postgresql/16/sql-createtable#SQL-CREATETABLE-METHOD)

Это дополнительное предложение задаёт табличный метод доступа, который будет использоваться для сохранения содержимого новой таблицы; типом этого метода доступа должен быть TABLE. В случае отсутствия этого указания для новой таблицы выбирается метод доступа по умолчанию. За подробностями обратитесь к [default\_table\_access\_method](https://postgrespro.ru/docs/postgresql/16/runtime-config-client" \l "GUC-DEFAULT-TABLE-ACCESS-METHOD).

WITH ( параметр\_хранения [= значение] [, ... ] ) [#](https://postgrespro.ru/docs/postgresql/16/sql-createtable#SQL-CREATETABLE-PARMS-WITH)

Это предложение определяет дополнительные параметры хранения для таблицы или индекса; за подробностями обратитесь к разделу [Параметры хранения](https://postgrespro.ru/docs/postgresql/16/sql-createtable#SQL-CREATETABLE-STORAGE-PARAMETERS) ниже. В целях обратной совместимости предложение WITH для таблицы также может содержать указание OIDS=FALSE, отмечающее, что строки новой таблицы не должны содержать OID (идентификатор объекта); указание OIDS=TRUE более не поддерживается

DROP [#](https://postgrespro.ru/docs/postgresql/16/sql-createtable#SQL-CREATETABLE-PARMS-ON-COMMIT-DROP)

Временная таблица будет удалена в конце текущего блока транзакции. Если это секционированная таблица, будут удалены и все её секции. Если у таблицы есть потомки в иерархии наследования, они также будут удалены.

PostgreSQL поддерживает следующие типы данных, допустимые для столбцов:

* числовые;
* строковые;
* календарные;
* *null* – специальный тип, обозначающий отсутствие информации.

**Числовые типы данных**[4]**:**

* ***serial***: представляет автоинкрементирующееся числовое значение, которое занимает 4 байта и может хранить числа от 1 до 2147483647. Значение данного типа образуется путем автоинкремента значения предыдущей строки. Поэтому, как правило, данный тип используется для определения идентификаторов строки.
* ***smallserial***: представляет автоинкрементирующееся числовое значение, которое занимает 2 байта и может хранить числа от 1 до 32767. Аналог типа ***serial***для небольших чисел.
* ***bigserial***: представляет автоинкрементирующееся числовое значение, которое занимает 8 байт и может хранить числа от 1 до 9223372036854775807. Аналог типа ***serial***для больших чисел.
* ***smallint***: хранит числа от -32768 до +32767. Занимает 2 байта. Имеет псевдоним ***int2***.
* ***integer***: хранит числа от -2147483648 до +2147483647. Занимает 4 байта. Имеет псевдонимы ***int***и ***int4***.
* ***bigint***: хранит числа от -9223372036854775808 до +9223372036854775807. Занимает 8 байт. Имеет псевдоним ***int8***.
* ***numeric***: хранит числа с фиксированной точностью, которые могут иметь до 131072 знаков в целой части и до 16383 знаков после запятой. Данный тип может принимать два параметра ***precision*** и ***scale***: ***numeric(precision, scale)***.

Параметр ***precision*** указывает на максимальное количество цифр, которые может хранить число.

Параметр ***scale*** представляет максимальное количество цифр, которые может содержать число после запятой. Это значение должно находиться в диапазоне от 0 до значения параметра ***precisio***n. По умолчанию оно равно 0.

Например, для числа 23.5141 ***precision*** равно 6, а ***scale*** - 4.

* ***decimal:*** хранит числа с фиксированной точностью, которые могут иметь до 131072 знаков в целой части и до 16383 знаков в дробной части. То же самое, что и numeric.
* ***real***: хранит числа с плавающей точкой из диапазона от 1E-37 до 1E+37. Занимает 4 байта. Имеет псевдоним float4.
* ***double precision***: хранит числа с плавающей точкой из диапазона от 1E-307 до 1E+308. Занимает 8 байт. Имеет псевдоним float8.

**Типы для работы с валютой (денежными единицами)**

Для работы с денежными единицами определен тип ***money***, который может принимать значения в диапазоне от -92233720368547758.08 до +92233720368547758.07 и занимает 8 байт.

**Символьные типы**

* ***character(n)***: представляет строку из фиксированного количества символов. С помощью параметра задается задается количество символов в строке. Имеет псевдоним ***char(n)***.
* ***character varying(n)***: представляет строку из переменной длины. С помощью параметра задается задается максимальное количество символов в строке. Имеет псевдоним ***varchar(n)***.
* ***text***: представляет текст произвольной длины.

**Бинарные данные**

Для хранения бинарных данных определен тип ***bytea***. Он хранит данные в виде бинарных строк, которые представляют последовательность октетов или байт.

**Типы для работы с датами и временем**

* ***timestamp***: хранит дату и время. Занимает 8 байт. Для дат самое нижнее значение - 4713 г до н.э., самое верхнее значение - 294276 г н.э.
* ***timestamp with time zone***: то же самое, что и ***timestamp***, только добавляет данные о часовом поясе.
* ***date***: представляет дату от 4713 г. до н.э. до 5874897 г н.э. Занимает 4 байта.
* ***time***: хранит время с точностью до 1 микросекунды без указания часового пояса. Принимает значения от 00:00:00 до 24:00:00. Занимает 8 байт.
* ***time with time zone***: хранит время с точностью до 1 микросекунды с указанием часового пояса. Принимает значения от 00:00:00+1459 до 24:00:00-1459. Занимает 12 байт.
* ***interval***: представляет временной интервал. Занимает 16 байт.

Распространенные форматы дат:

* yyyy-mm-dd - 1999-01-08
* Month dd, yyyy - January 8, 1999
* mm/dd/yyyy - 1/8/1999

Распространенные форматы времени:

* hh:mi - 13:21
* hh:mi am/pm - 1:21 pm
* hh:mi:ss - 1:21:34

**Логический тип**

Тип ***boolea*n** может хранить одно из двух значений: ***true*** или ***false***.

Вместо ***true***можно указывать следующие значения: TRUE, 't', 'true', 'y', 'yes', 'on', '1'.

Вместо ***false*** можно указывать следующие значения: FALSE, 'f', 'false', 'n', 'no', 'off', '0'.

**Типы для представления интернет-адресов**

* ***cidr***: интернет-адрес в формате IPv4 и IPv6. Например, 192.168.0.1. Занимает от 7 до 19 байт.
* ***inet***: интернет-адрес в формате ***cidr/y***, где cidr – это адрес в формате IPv4 или IPv6, а ***/y*** - количество бит в адресе (если этот параметр не указан, то используется 34 для IPv4, 128 для IPv6). Например, 192.168.0.1/24 или 2001:4f8:3:ba:2e0:81ff:fe22:d1f1/128. Занимает от 7 до 19 байт.
* ***macaddr***: хранит MAC-адрес. Занимает 6 байт.
* ***macaddr8***: хранит MAC-адрес в формате EUI-64. Занимает 8 байт.

**Геометрические типы**

* ***point***: представляет точку на плоскости в формате (x,y). Занимает 16 байт.
* ***line***: представляет линию неопределенной длины в формате {A,B,C}. Занимает 32 байта.
* ***lseg***: представляет отрезок в формате ((x1,y1),(x2,y2)). Занимает 32 байта.
* ***box***: представляет прямоугольник в формате ((x1,y1),(x2,y2)). Занимает 32 байта.
* ***path***: представляет набор содиненных точек. В формате ((x1,y1),...) путь является закрытым (первая и последняя точка соединяются линией) и фактически представляет многоугольник. В формате [(x1,y1),...] путь является открытым Занимает 16+16n байт.
* ***polygon***: представляет многоугольник в формате ((x1,y1),...). Занимает 40+16n байт.
* ***circle***: представляет окружность в формате <(x,y),r>. Занимает 24 байта.

**Остальные типы данных**

* ***json***: хранит данные ***json*** в текстовом виде.
* ***jsonb***: хранит данные ***json*** в бинарном формате.
* ***uuid***: хранит универсальный уникальный идентификатор (UUID), например, a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11. Занимает 32 байта.
* ***xml***: хранит даные в формате XML.

Пример создания таблицы:

CREATE TABLE Client (

id\_client BIGSERIAL NOT NULL PRIMARY KEY,

cl\_address VARCHAR(255) NOT NULL,

cl\_company VARCHAR(255) NOT NULL,

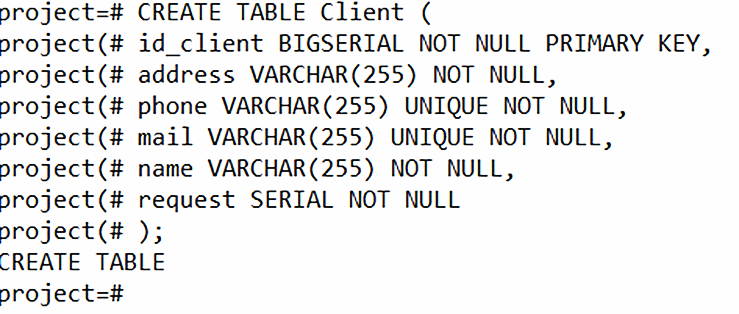
cl\_phone VARCHAR(255) UNIQUE NOT NULL,

cl\_mail VARCHAR(255) UNIQUE NOT NULL,

cl\_name VARCHAR(255) NOT NULL,

cl\_request SERIAL NOT NULL

);



**Обеспечение ссылочной целостности** задается конструкцией:

*foreign key [name\_key] (col1, ... ) REFERENCES tbl (tbl\_col, ... )*

*[ON DELETE {CASCADE | SET NULL | NO ACTION | RESTRICT | SET DEFAULT}] [ON UPDATE {CASCADE | SET NULL | NO ACTION | RESTRICT | SET DEFAULT}]*

Конструкция позволяет задать внешний ключ с необязательным именем *name\_key* на столбцах, которые задаются в круглых скобках (один или несколько). Ключевое слово *references* указывает таблицу *tbl*, на которую ссылается внешний ключ; в круглых скобках указываются имена столбцов. Необязательные конструкции *ON DELETE* и *ON UPDATE* позволяют задать поведение СУБД при удалении и обновлении строк из таблицы-предка. Параметры, следующие за этими ключевыми словами, имеют значения:

* *CASCADE –* при удалении или обновлении записи в таблице-предке, содержащей первичный ключ, записи со ссылками на это значение в таблице-потомке удаляются или обновляются автоматически;
* *SET NULL –* при удалении или обновлении записи в таблице-предке, содержащей первичный ключ, в таблице-потомке значения внешнего ключа, ссылающегося на таблицу-предка, устанавливаются в *null*;
* *NO ACTION –* при удалении или обновлении записей, содержащих первичный ключ, с таблицей-потомком никаких действий не производится;
* *RESTRICT –* если в таблице-потомке имеются записи, ссылающиеся на первичный ключ таблицы-предка, при удалении или обновлении записей с таким первичным ключом возвращается ошибка;
* *SET DEFAULT* – согласно стандарту *SQL*, при удалении или обновлении первичного ключа в таблице-потомке для ссылающихся на него записей в поле внешнего ключа должно устанавливаться значение по умолчанию.

**Создание индексов**. Индексы играют большую роль в БД, т. к. это основной способ ускорения их работы. Записи в таблице располагаются хаотически. Чтобы найти нужную запись, необходимо сканировать всю таблицу, на что уходит много времени. Идея индексов состоит в том, чтобы создать для столбца копию, которая постоянно будет поддерживаться в отсортированном состоя­нии. Это позволяет быстро осуществлять поиск по такому столбцу.

Все необходимые индексы формируются при создании таблицы. Индексированы будут все столбцы, объявленные как *PRIMARY KEY, KEY, UNIQUE* или *INDEX*. Индекс также можно добавить с помощью оператора *CREATE INDEX*. Перед выполнением оператор преобразуется в оператор *ALTER* *TABLE*. Например, создание индекса с именем *name* на основе поля *u\_name* из таблицы *users*:

*create index name on users (u\_name);*

Перед ключевым сло­вом *index* может присутствовать *UNIQUE*, требующее уникальности ограниче­ния.

Корректность таблиц в БД можно проверить с помощью оператора

*SHOW TABLES;*

Более подробную информацию о структуре таблицы дает команда

*DESCRIBE имя\_таблицы;*

**Переименование БД**.

ALTER DATABASE name RENAME TO newname.

**Удаление БД**.

Удалить всю БД вместе с ее содержимым можно командой:

*drop database [IF EXISTS] имя\_базы\_данных;*

**Сохранение БД.**

Сохранить базу данных ,создается специальный файл где хранится ее состояние, это можно сделать командой:  
 *PG\_DUMP [IF EXISTS] –Fc –f filename.dump*

**Удаление таблиц и индексов**.

Удалить таблицу можно с помощью оператора:

*drop table [IF EXISTS] имя\_таблицы;*

Удалить индекс можно с помощью оператора:

*drop index имя\_индекса on имя\_таблицы;*

**Изменение структуры таблиц.**

Изменить структуру су­ществующей таблицы можно с помощью оператора *ALTER TABLE*. Например, можно создать индекс *name* для таблицы *users* следующим образом:

*alter table users* *add index name (u\_name);*

Оператор *ALTER TABLE* является исключительно гибким, поэтому он имеет огромное множество дополнительных ключевых слов.

**Изменение Колонок таблице**.

Добавить столбец вы можете так. Новый столбец заполняется заданным для него значением по умолчанию (или значением NULL, если вы не добавите указание DEFAULT).

ALTER TABLE *table\_name*  
ADD *new\_column\_name column\_definition*,  
ADD new\_column\_name column\_definition,

При этом вы можете сразу определить ограничения столбца, используя обычный синтаксис:

ALTER TABLE *products* ADD COLUMN *description\_text* CHECK (*description <>* '');

## **Изменить столбец в таблице.**

ALTER TABLE *table\_name*  
ALTER COLUMN *column\_name* TYPE *column\_definition*;

## **Удалить столбец в таблице.**

ALTER TABLE *table\_name*  
DROP COLUMN *column\_name*;

## **Переименовать столбец в таблице.**

ALTER TABLE *table\_name*  
RENAME COLUMN *old\_name TO new\_name*;

## **Переименовать таблицу.**

ALTER TABLE *table\_name*  
RENAME TO *new\_table\_name*;

**Практическая работа**

При выполнении лабораторной работы необходимо для заданной предметной области средствами PostgreSQL:

* создать базу данных;
* создать таблицы, определить поля таблиц, индексы, учитывать ссылочную целостность;
* определить связи между таблицами и ограничения целостности;
* заполнить согласованными данными таблицы БД;
* при необходимости исправить введенную информацию;
* составить отчет по лабораторной работе.

**Пример выполнения работы**

Операторы создания БД contract имеют следующий вид (целесообразно создать в *Блокноте* текстовый файл и записать туда операторы).

CREATE TABLE Client (

id\_client BIGSERIAL PRIMARY KEY,

cl\_address VARCHAR(255) NOT NULL,

cl\_company VARCHAR(255) NOT NULL,

cl\_phone VARCHAR(255) UNIQUE NOT NULL,

cl\_mail VARCHAR(255) UNIQUE NOT NULL,

cl\_name VARCHAR(255) NOT NULL,

cl\_request SERIAL NOT NULL

);

CREATE TABLE Skill (

skill\_name VARCHAR(255) PRIMARY KEY,

skill\_DMS VARCHAR(255) NOT NULL,

skill\_language VARCHAR(255) NOT NULL,

skill\_SP VARCHAR(255) NOT NULL,

skill\_OS VARCHAR(255) NOT NULL

);

CREATE TABLE Department (

id\_department BIGSERIAL PRIMARY KEY,

dep\_task VARCHAR(255) NOT NULL

);

CREATE TABLE Employee (

id\_employee BIGSERIAL PRIMARY KEY,

em\_department BIGINT,

em\_name VARCHAR(255) NOT NULL,

em\_position VARCHAR(255) NOT NULL,

em\_phone VARCHAR(255) UNIQUE NOT NULL,

em\_mail VARCHAR(255) UNIQUE NOT NULL,

em\_skill VARCHAR(255) NOT NULL,

FOREIGN KEY (em\_department) REFERENCES Department(id\_department) ON DELETE SET NULL,

FOREIGN KEY (em\_skill) REFERENCES Skill(skill\_name) ON DELETE SET DEFAULT

);

CREATE TABLE Project (

id\_project BIGSERIAL PRIMARY KEY,

client BIGINT NOT NULL,

proj\_task VARCHAR(255) NOT NULL,

proj\_start\_date DATE NOT NULL,

proj\_end\_date DATE NOT NULL,

proj\_status BOOLEAN,

FOREIGN KEY (client) REFERENCES Client(id\_client) ON DELETE SET DEFAULT

);

CREATE TABLE Project\_Department (

id\_project BIGINT,

id\_department BIGINT,

PRIMARY KEY (id\_project, id\_department),

FOREIGN KEY (id\_project) REFERENCES Project(id\_project) ON DELETE CASCADE,

FOREIGN KEY (id\_department) REFERENCES Department(id\_department) ON DELETE CASCADE

);

**2) Вставка, удаление и обновление данных**

**Основные сведения**

Рассматриваются следующие вопросы:

* вставка данных с помощью оператора *INSERT*;
* удаление данных оператором *DELETE*;
* обновление данных с помощью оператора *UPDATE*.

После создания БД и таблиц перед разработчиком встает задача заполнения таблиц данными. В реляционных БД традиционно применяют три подхода:

* однострочный оператор *insert* – добавляет в таблицу новую запись;
* многострочный оператор *insert* – добавляет в таблицу несколько записей;
* пакетная загрузка, добавление данных из файла.

**Вставка данных с помощью оператора *INSERT***. Однострочный оператор *insert* может использоваться в нескольких формах. Упрощенный синтаксис первой формы:

*insert INTO имя\_таблицы [(имя\_столбца, ... )]*

*VALUES(выражение, ... );*

*Пример:*

INSERT INTO Client (cl\_address, cl\_company, cl\_phone, cl\_mail, cl\_name, cl\_request)

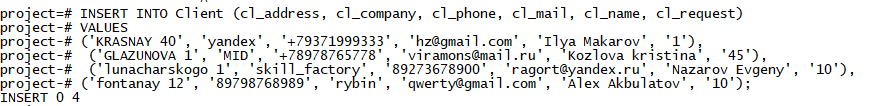
VALUES

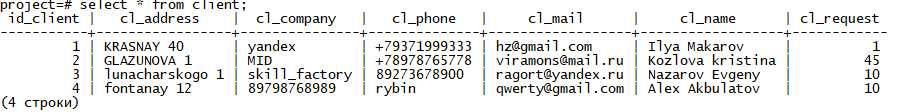
*('KRASNAY 40', 'yandex', '+79371999333', 'hz@gmail.com', 'Ilya Makarov', '1'),*

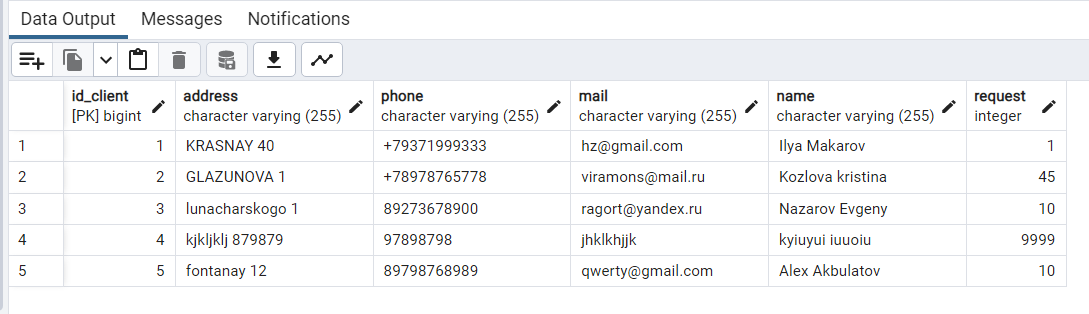
*('GLAZUNOVA 1', 'MID', '+78978765778', 'viramons@mail.ru', 'Kozlova kristina', '45'),*

*('lunacharskogo 1', 'skill\_factory', '89273678900', 'ragort@yandex.ru', 'Nazarov Evgeny', '10'),*

*('fontanay 12', '89798768989', 'rybin', 'qwerty@gmail.com', 'Alex Akbulatov', '10');*



**



Оператор вставляет новую запись в таблицу *имя\_таблицы*. Значения полей записи перечисляются в списке *(выражение, ... )*. Порядок следования столбцов задается списком *(имя\_столбца, ... )*. Список столбцов *(имя\_столбца, ... )* позволяет менять порядок следования столбцов при добавлении.

Оператор заносит в таблицу *имя\_таблицы* новую запись, столбец *имя\_столбца* в которой получает значение *выражение*.

Многострочный оператор *INSERT* совпадает по форме с однострочным оператором, но после ключевого слова *values* добавляется через запятую несколько списков *(выражение, ... )*.

**Удаление данных.** Для удаления записей из таблиц предусмотрен оператор *DELETE.*

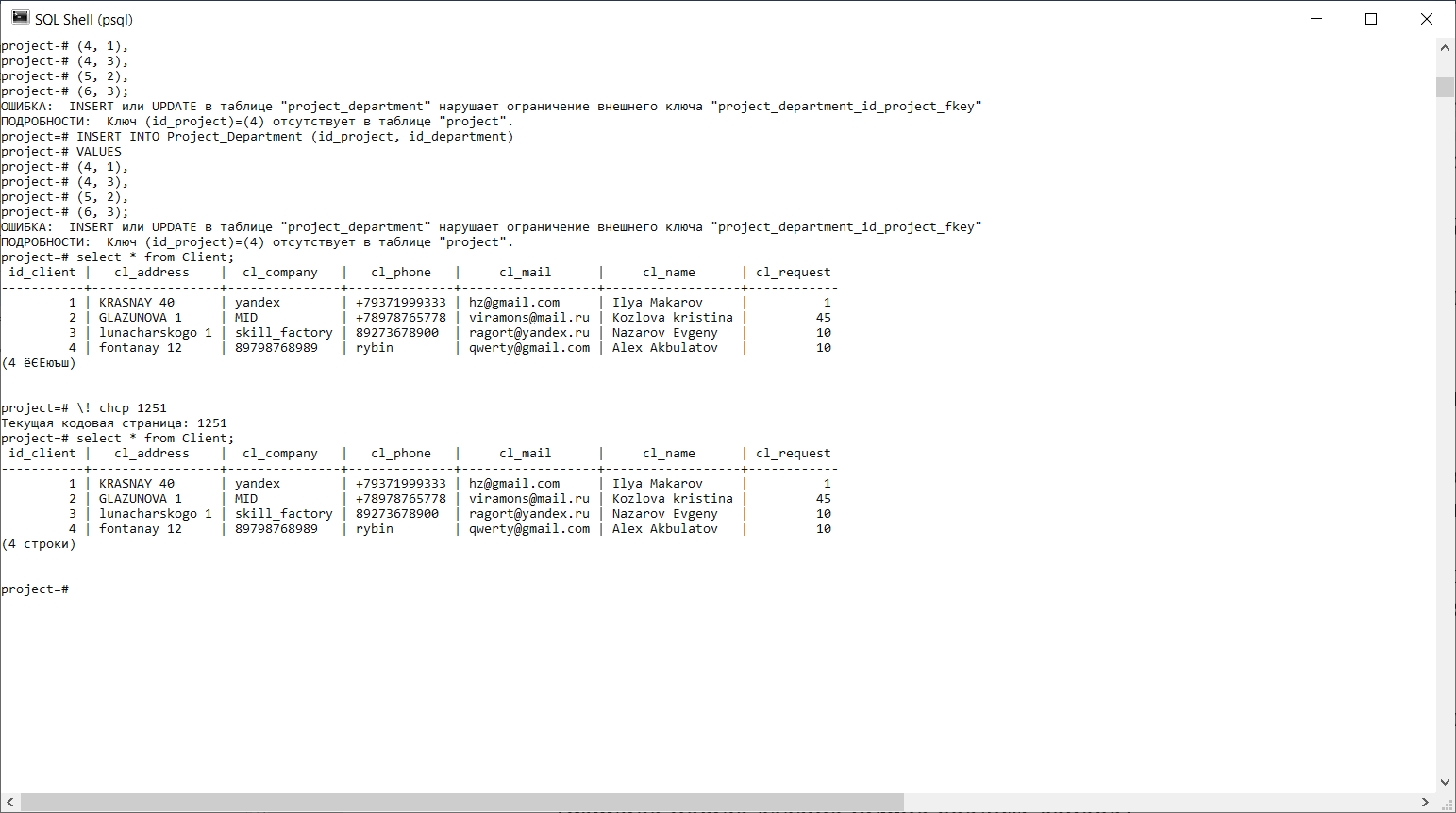
Оператор *DELETE* имеет следующий синтаксис:

*DELETE FROM имя\_таблицы*

*[Where* *условие];*

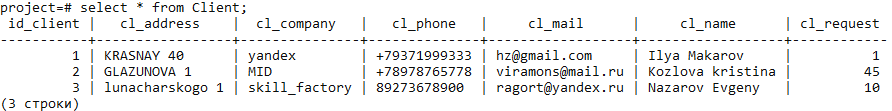
Оператор удаляет из таблицы *имя\_таблицы* записи, удовлетворяющие условию. В следующем примере из таблицы *Client* удаляются записи, имеющие значение первичного ключа *cli\_id* больше шести.

select \* from Client;

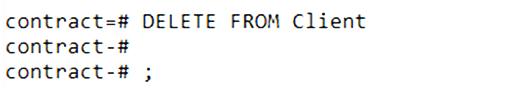


delete from Client where id\_client=4;

select \* from Client;



Если в операторе отсутствует условие *where*, удаляются все записи таблицы.



**Обновление данных.** Обновление данных (изменение значений полей в существующих записях) обеспечивает оператор *Update*.

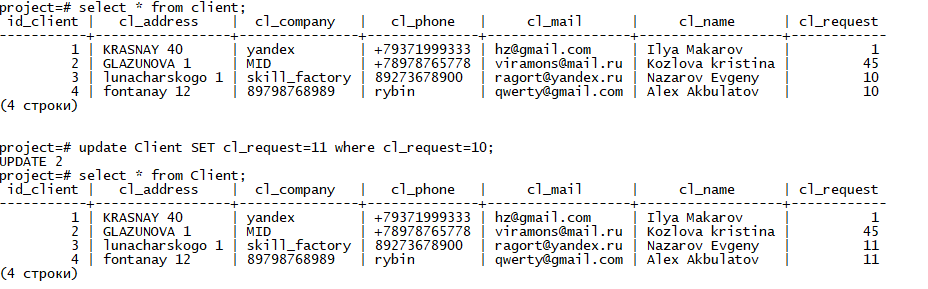
Оператор *UPDATE* позволяет обновлять отдельные поля в существующих записях. Имеет следующий синтаксис

*Update имя\_таблицы*

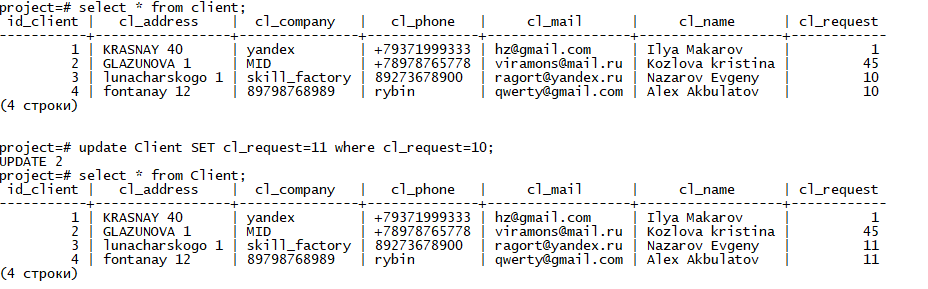
*SET имя\_столбца1= выражение1 [, имя\_столбца2 = выражение2 … ]*

*[WHERE условие];*

После ключевого слова *update* указывается таблица, которая изменяется. В предложении *set* указывается, какие столбцы обновляются и устанавливаются их новые значения. Необязательное условие *WHERE* позволяет задать критерий отбора строк (обновляться будут только строки, удовлетворяющие условию).

select \* from Client; 

update Client SET cl\_request=11 where cl\_request=10;



**Пример заполнения таблиц**

-- *Вставка квалификаций*

INSERT INTO Skill (skill\_name, skill\_DMS, skill\_language, skill\_SP, skill\_OS)

VALUES

('Разработчик ПО', 'MySQL, PostgreSQL', 'Python, Java', 'IntelliJ IDEA, PyCharm', 'Windows, Linux'),

('Системный администратор', 'PostgreSQL, Oracle', 'Bash, PowerShell', 'IntelliJ IDEA, Visual Studio Code', 'Linux, Windows'),

('Тестировщик', 'MySQL', 'Java, Python', 'IntelliJ IDEA', 'Linux, Windows'),

('Web-разработчик', 'MySQL, SQL Server', 'JavaScript, PHP', 'WebStorm, Visual Studio Code', 'Linux, Windows'),

('Аналитик данных', 'MySQL, SQL Server, Oracle', 'Python, R', 'PyCharm, RStudio', 'Linux, Windows'),

('Junior Разработчик', 'MySQL, PostgreSQL', 'C, C++, Java', 'IntelliJ IDEA, Visual Studio', 'Linux, Windows'),

('Middle Разработчик', 'MySQL, PostgreSQL', 'C++, Java, C#', 'IntelliJ IDEA, Visual Studio', 'Linux, Windows'),

('Senior Разработчик', 'MySQL, PostgreSQL', 'Java, C#', 'IntelliJ IDEA, Visual Studio', 'Linux, Windows');

-- *Вставка отделов*

INSERT INTO Department (dep\_task)

VALUES

('Разработка ПО'),

('Администрирование'),

('Тестирование');

-- *Вставка сотрудников*

INSERT INTO Employee (em\_department, em\_name, em\_position, em\_phone, em\_mail, em\_skill)

VALUES

(1, 'Иванов Иван', 'Разработчик', '1234567890', 'ivanov@example.com', 'Разработчик ПО'),

(1, 'Петров Петр', 'Разработчик', '0987654321', 'petrov@example.com', 'Разработчик ПО'),

(2, 'Сидоров Алексей', 'Системный администратор', '5678901234', 'sidorov@example.com', 'Системный администратор'),

(2, 'Козлова Мария', 'Системный администратор', '6789012345', 'kozlova@example.com', 'Системный администратор'),

(2, 'Николаев Иван', 'Системный администратор', '7890123456', 'nikolaev@example.com', 'Системный администратор'),

(3, 'Иванова Анна', 'Тестировщик', '8901234567', 'ivanova@example.com', 'Тестировщик'),

(3, 'Петрова Екатерина', 'Тестировщик', '9012345678', 'petrova@example.com', 'Тестировщик');

-- *Вставка клиентов*

INSERT INTO Client (cl\_address, cl\_company, cl\_phone, cl\_mail, cl\_name, cl\_request)

VALUES

*('KRASNAY 40',’yandex’, '+79371999333', 'hz@gmail.com', 'Ilya Makarov', '1'),*

*('GLAZUNOVA 1',’MID’, '+78978765778', 'viramons@mail.ru', 'Kozlova kristina', '45'),*

*('lunacharskogo 1',’skill\_factory’, '89273678900', 'ragort@yandex.ru', 'Nazarov Evgeny', '10'),*

*('fontanay 12', '89798768989',’rybin’, 'qwerty@gmail.com', 'Alex Akbulatov', '10');*

-- *Вставка проектов*

INSERT INTO Project (client, proj\_task, proj\_start\_date, proj\_end\_date, proj\_status)

VALUES

(3, 'Разработка приложения Погоды', '2024-02-14', '2024-03-14', false),

(2, 'Секретно', '2024-02-14', '2024-04-14', false),

(2, 'Секретно', '2024-02-14', '2024-05-14', false);

-- *Вставка отделов для проектов*

INSERT INTO Project\_Department (id\_project, id\_department)

VALUES

(1, 1),

(1, 3),

(2, 2),

(3, 3);