

Лабораторная работа №1. Создание базы данных в СУБД PostgreSQL.

Основы программирования на языке SQL.

Цель работы: Получить теоретические и практические навыки создания базы данных в СУБД PostgreSQL, изучить основные понятия и операторы, научиться работать в среде pgAdmin, сформировать знания и умения по программирования на языке SQL, приобрести практические навыки работы со средствами языка SQL для обновления, удаления и вставки данных в БД.

Средства выполнения

- СУБД PostgreSQL
- Средство администрирования pgAdmin

Пункты задания для выполнения

1. Изучить теоретические сведения лабораторной работы.
2. Создать базу данных в среде pgAdmin.
3. Создать таблицы (две с помощью запроса и две с помощью графического способа), первичные ключи, обеспечить ссылочную целостность базы данных (ограничение по внешним ключам) в соответствии с приведенным ниже рисунком.
4. Изменить одну из созданных таблиц с помощью команды ALTER TABLE.
5. Создать ограничение на уникальность по двум столбцам (например, имя + фамилия).
6. Создать ограничение на проверку данных (см. раздел *Использование проверочных ограничений*).
7. Создать значение по умолчанию.
8. Заполнить каждую таблицу базы данных (минимум 10 записей в каждую таблицу) с помощью команды INSERT. ПРИМЕЧАНИЕ: для просмотра содержания таблиц необходимо использовать запрос SELECT * FROM <название таблицы>
9. Изучить команду для обновления данных в таблицах.
10. Изучить команду для удаления данных из таблиц.
11. Создать SQL-скрипт (дамп) БД.
12. Создать диаграмму БД.

13. Вывести список таблиц и их структуру в терминале.
14. Защитить лабораторную работу.

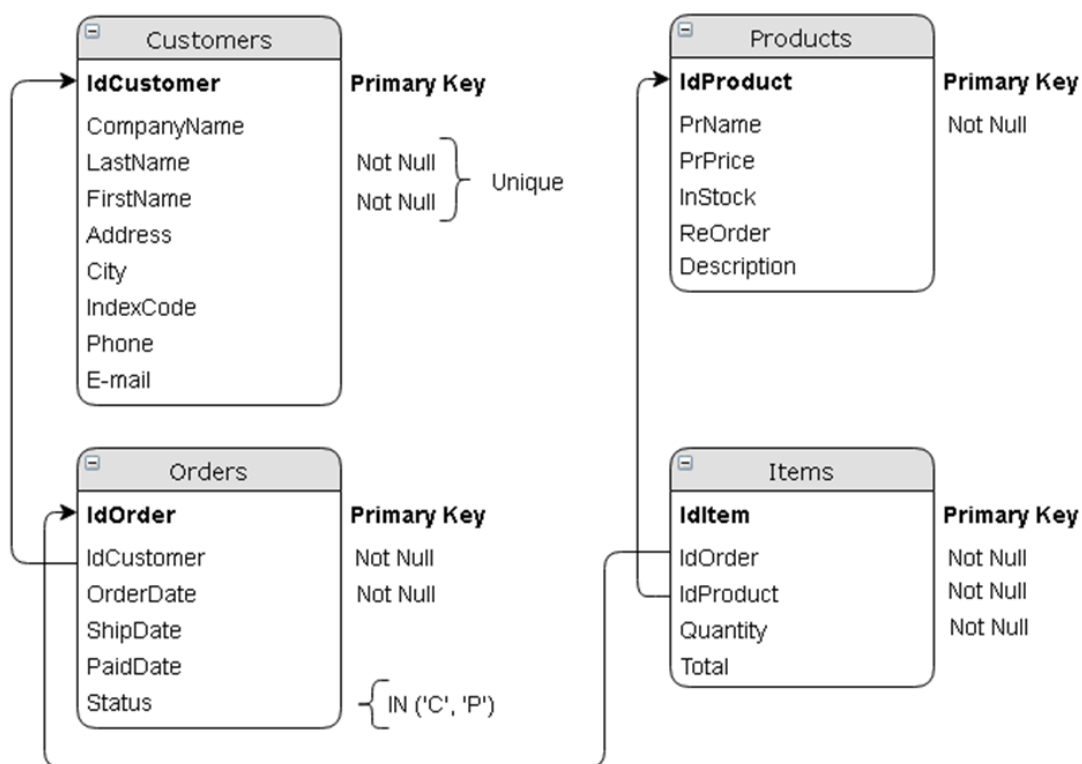
Схема таблиц

Таблица **Customers** будет хранить такую информацию о клиентах, как название фирмы, фамилия, имя, адрес, город, индекс, телефон.

Таблица **Orders** будет содержать подробную информацию о заказах — номер заказа, идентификатор покупателя, даты отправки и оплаты, а также статус заказа ('C' - Cancelled отменен, 'P' - Processed обработан, 'A' - adopted принят).

Таблица **Products** будет хранить такую информацию о продукте, как название, цена продукта, его количество на складе, необходимость перезаказа и описание.

Таблица **Items** содержит идентификатор заказа, идентификатор продукта в заказе и его количество, общую сумму по данной позиции.



Теоретическая часть

PostgreSQL является объектно-реляционной СУБД, т.е. объединяет реляционную модель с принципами объектно-ориентированного подхода.

Прим. В методических указаниях используется PostgreSQL версии 14.

Физическое размещение файлов баз данных в PostgreSQL

В PostgreSQL область, где хранятся файлы баз данных называется *кластер*. В файловой системе кластер хранится как каталог, содержащий все данные. Список баз данных подгружается из этого каталога при запуске сервера. В ОС Windows по умолчанию кластер расположен в *C:\Program Files\PostgreSQL\14\data*. Проверить и изменить расположение кластера можно из реестра: *Компьютер\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\postgresql-x64-14*. В переменной ImagePath путь, указанный после флага -D – это путь до каталога, являющегося кластером.

Подробнее про физическое размещение БД можно прочитать [здесь](#).

Компоненты СУБД PostgreSQL

- **Tables (Таблицы).** Таблицы базы данных предназначены для хранения данных. Таблица состоит из строк и столбцов. Число столбцов и их порядок фиксированы, для каждого столбца определено имя и тип данных. Число строк переменное и отражает текущее количество данных, находящихся в таблице.
- **Views (Представления).** По сути своей являются «виртуальными таблицами». Представление - это поименованный запрос SELECT. Indexes (Индексы). Существуют для поддержания вместе с данными информации об их упорядоченности по различным критериям, что позволяет существенно повысить производительность некоторых операций, в частности поиска данных. Индексы существуют непосредственно вместе с таблицами и не имеют смысла сами по себе. Индексирование может быть выполнено по одному или нескольким столбцам и произведено в любой момент.
- **Keys (Ключи).** Подобно индексам, ключи не существуют сами по себе. Ключ - один из типов ограничений целостности.
- **Defaults (Умолчания).** Не существуют отдельно от таблиц. Умолчания определяют, какие значения будут подставлены в поле данных при добавлении строки, если значение не задано явно. Rules (Правила). Механизм, предназначенный для установления ограничений на диапазон возможных значений поля таблицы или нового типа данных, определяемого пользователем.
- **Constraints (Ограничения целостности).** Определяют диапазон возможных значений полей таблицы.
- **Stored Procedures (Хранимые процедуры).** Поименованный набор команд SQL. Хранимые процедуры располагаются на сервере вместе с базой данных и могут запускаться различными пользователями, имеющими соответствующие права.

- **Triggers (Триггеры).** Не существуют отдельно от таблицы. Триггер - специальный вид хранимой процедуры. Триггер может выполняться при изменениях данных или событиях в базе данных.
- **User-defined data types, UDDT (Определяемые пользователем типы данных).** Предоставляет специальный аппарат для создания пользовательских типов данных.
- **User-defined functions, UDF (Определяемые пользователем функции).** Представляет собой набор команд SQL, сохраненных в специальном виде.
- **Users (Пользователи).** Определяет список пользователей базы данных. Служит для работы механизма защиты данных.
- **Roles (Роли).** Роли пользователей - важный механизм для организации разграничения доступа. Пользователю может быть назначена одна или больше ролей.

psql

Консольная утилита для работы с Postgres, подробнее [здесь](#).

Основные метакоманды:

- \connect db_name (\c db_name) – подключиться к базе с именем db_name
- \du – список пользователей
- \dp (или \z) – список таблиц, представлений, последовательностей, прав доступа к ним
- \di – индексы
- \ds – последовательности
- \dt – список таблиц
- \dt+ — список всех таблиц с описанием
- \dt *s* — список всех таблиц, содержащих s в имени
- \dv – представления
- \dS – системные таблицы
- \d+ – описание таблицы
- \l – список баз данных
- \d "table_name" – описание таблицы
- \i запуск команды из внешнего файла, например \i /my/directory/my.sql
- \pset – команда настройки параметров форматирования
- \echo – выводит сообщение
- \set – устанавливает значение переменной среды. Без параметров выводит список текущих переменных (\unset – удаляет).
- \? – справочник psql
- \help – справочник SQL
- \q (или Ctrl+D) – выход с программы

Запуск утилиты:

```
# sudo -u postgres psql
```

Пример создания БД из терминала при помощи psql:

```
# create database lab1;  
# \c lab1
```

Здесь же можно выполнять SQL-запросы к базе данных.

Прим. Флаг `-u` с аргументом `postgres` означает, что команда выполняется под пользователем с именем `postgres` (пользователь по умолчанию в PostgreSQL).

Утилита pgAdmin

Графический клиент для СУБД PostgreSQL. Предоставляет множество возможностей для работы с БД. В pgAdmin можно создавать базы данных и все ассоциированные с ними объекты (таблицы, представления, хранимые процедуры и др.), выполнять типовые задачи обслуживания баз данных, такие как резервное копирование и восстановление. Здесь также можно настраивать систему безопасности базы данных и сервера, просматривать журнал ошибок и многое другое.

Отключение и подключение БД

Отключение БД выполняется с помощью команды:

```
service postgresql stop
```

Подключение БД выполняется с помощью команды:

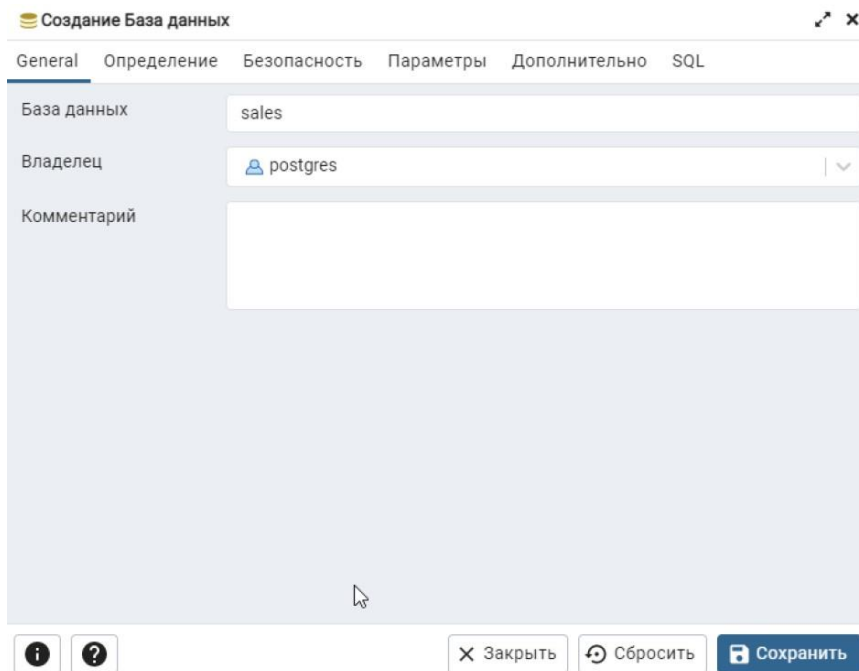
```
service postgresql start
```

Подключение к серверу и создание базы данных в среде pgAdmin

В *Обозревателе* раскройте выпадающий список *Servers*, затем выберите нужный сервер (по умолчанию там будет только сервер PostgreSQL) и введите пароль, который вы задали при установке Postgres. В *Обозревателе* появится список баз данных и откроется рабочая область.

Пример. Создать базу данных `sales`:

1. Щелкните правой кнопкой мыши (ПКМ) по элементу Базы данных в Обозревателе и выберите Создать > База данных...



Создание База данных

General | Определение | Безопасность | Параметры | Дополнительно | SQL

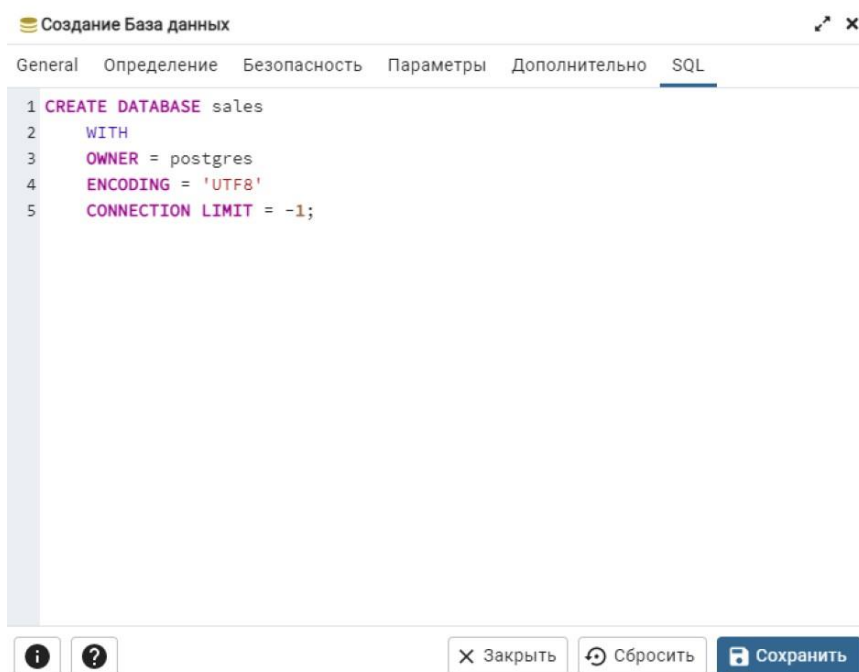
База данных: sales

Владелец: postgres

Комментарий:

Заккрыть Сбросить Сохранить

- Во вкладке *General* укажите название базы данных. *Прим.* Во вкладке SQL автоматически сгенерируется SQL-запрос на создание БД:



Создание База данных

General | Определение | Безопасность | Параметры | Дополнительно | SQL

```
1 CREATE DATABASE sales
2 WITH
3 OWNER = postgres
4 ENCODING = 'UTF8'
5 CONNECTION LIMIT = -1;
```

Заккрыть Сбросить Сохранить

- Нажмите *Сохранить*

Типы данных

В PostgreSQL используется следующие типы данных:

- Целочисленные типы данных**

Типы данных для хранения целых чисел (в скобках указан диапазон значений типа данных): smallint (-32768 .. +32767), int (-231 .. +(231-1)), bigint (-263 .. +(263-1)), занимают 2,4, 8 байтов в памяти соответственно.

- **Типы данных для хранения чисел с плавающей запятой**

real – вещественное число с точностью в пределах 6 десятичных цифр, занимает в памяти 4 байта.

double precision – вещественное число с точностью в пределах 15 десятичных цифр, занимает в памяти 8 байт.

- **Типы данных для хранения чисел с фиксированными точностью и масштабом**

numeric (precision, scale) где precision (точность) – максимальное количество десятичных разрядов числа (как слева, так и справа от десятичной запятой), а scale (масштаб) – максимальное количество десятичных разрядов числа справа от десятичной запятой. Объем занимаемой памяти варьируется в зависимости от этих параметров.

Тип decimal полностью эквивалентен типу numeric.

- **Последовательные типы**

Типы smallserial(1 .. 32767), serial (1 .. (231-1)), и bigserial (1 .. (263-1)) служат для создания столбцов с автоинкрементом. Занимают 2, 4, 8 байтов соответственно.

- **Символьные типы данных**

char(n), varchar(n) – строковые данные фиксированной длины. Оба эти типа могут хранить текстовые строки длиной до n символов. Если длина сохраняемой строки оказывается меньше объявленной, значения типа char будут дополняться пробелами; а тип varchar просто сохранит короткую строку;

text – строковый тип данных, позволяющий хранить строки произвольной длины.

- **Логический тип данных**

Тип boolean может иметь три состояния: true (также yes/on/1), false (также no/off/0) и третье состояние, «unknown», которое представляется SQL-значением NULL.

- **Типы данных даты и времени**

Timestamp (with/without time zone) – дата и время (с указанием часового пояса/без указания часового пояса), объем занимаемой памяти – 8 байт date – дата без указания времени, объем занимаемой памяти – 4 байта;

time (with/without time zone) – время без указания даты (с указанием часового пояса/без указания часового пояса), объем занимаемой памяти – 12 байт

Для типов, содержащих время, может быть указана точность, определяющая, сколько знаков после запятой должно сохраняться в секундах.

- **Денежный тип данных для хранения финансовой информации**

money – тип данных, представляющий денежные (валютные) значения. Объем занимаемой памяти – 8 байт.

Подробнее про типы данных [здесь](#).

Таблицы

Вся информация в базе данных хранится в таблицах. Таблицы состоят из строк или записей и столбцов или полей. Каждое поле имеет три характеристики:

- Имя поля - используется для обращения к полю;
- Значение поля - определяет информацию, хранимую в поле;
- Тип данных поля - определяет, какой вид информации можно хранить в поле.

Создание таблиц.

```
CREATE TABLE table_name (  
    column1 datatype ограничение(если есть),  
    column2 datatype ограничение(если есть),  
    column3 datatype ограничение(если есть),  
    ....  
);
```

где *column1* – название столбца, *datatype* – стандартный или пользовательский тип данных, *ограничение* – накладываемое на столбец ограничение данных.

Пример:

```
Query  Query History  
1 CREATE TABLE t1(field1 integer, field2 char(10));
```

Удаление таблиц.

```
DROP TABLE table_name;
```

Изменение определения таблиц.

```
ALTER TABLE
```

Примеры:

Добавление столбца в таблицу:

```
ALTER TABLE t1 ADD field3 char(15);
```


Изменение типа данных столбца таблицы:

```
ALTER TABLE t1 ALTER COLUMN field2 TYPE INTEGER USING field2::integer;
```

Удаление столбца таблицы:

```
ALTER TABLE t1 DROP COLUMN field1;
```

Переименование таблицы:

```
ALTER TABLE t1 RENAME TO t2;
```

Ограничения

Ограничение-проверка — наиболее общий тип ограничений. В его определении вы можете указать, что значение данного столбца должно удовлетворять логическому выражению (проверке истинности).

В Postgres поддерживаются следующие типы ограничений: на уникальность, на допустимость значения NULL, первичный ключ, внешний ключ, ограничения общего вида.

- **ограничения общего вида**

Пример синтаксиса:

```
CREATE TABLE products (  
    product_id integer,  
    prname varchar(50),  
    price numeric CHECK (price > 0)  
);
```

Вы можете также присвоить ограничению отдельное имя. Это улучшит сообщения об ошибках и позволит вам ссылаться на это ограничение, когда вам понадобится изменить его. Сделать это можно так:

```
price numeric CONSTRAINT positive_price CHECK (price > 0)
```

То есть, чтобы создать именованное ограничение, напишите ключевое слово CONSTRAINT, а за ним идентификатор и собственно определение ограничения. (Если вы не определите имя ограничения таким образом, система выберет для него имя за вас.)

- **ограничения NOT NULL**

Ограничение NOT NULL указывает, что столбцу нельзя присваивать значение пустое значение NULL.

Пример синтаксиса:

```
CREATE TABLE products (  
    product_id integer NOT NULL,  
    pname varchar(50),  
    price numeric  
);
```

- **ограничения уникальности**

Ограничения уникальности гарантируют, что данные в определённом столбце или группе столбцов уникальны среди всех строк таблицы. Ограничение записывается

```
CREATE TABLE products (  
    product_id integer UNIQUE,  
    pname varchar(50),  
    price numeric  
);
```

Чтобы определить ограничение уникальности для группы столбцов, запишите его в виде ограничения таблицы, перечислив имена столбцов через запятую:

```
CREATE TABLE example (  
    a integer,  
    b integer,  
    c integer,  
    UNIQUE (a, c)  
);
```

- **ограничение DEFAULT**

Установка для полей значений по умолчанию — это отличный способ избавить пользователя от излишней работы, если значения этих полей во всех записях, как правило, принимают одни и те же значения. Значение по умолчанию будет добавлено ко всем новым записям, если другое значение не указано.

Пример синтаксиса:

```
CREATE TABLE users (  
    user_id int NOT NULL,  
    uname varchar(50) NOT NULL,  
    fullname varchar(60),  
    gender int,  
    country varchar(50) DEFAULT 'Spain'  
);
```

- **первичные ключи**

Ограничение первичного ключа означает, что образующий его столбец или группа столбцов может быть уникальным идентификатором строк в таблице. Для этого требуется, чтобы значения были одновременно уникальными и отличными от NULL. Таким образом, таблицы со следующими двумя определениями будут принимать одинаковые данные:

```
CREATE TABLE products (
    product_id integer UNIQUE NOT NULL,
    prname varchar(100),
    price numeric
);

CREATE TABLE products (
    product_id integer PRIMARY KEY,
    prname varchar(100),
    price numeric
);
```

Первичные ключи могут включать несколько столбцов. Синтаксис похож на запись ограничений уникальности:

```
CREATE TABLE example (
    a integer,
    b integer,
    c integer,
    PRIMARY KEY (a, c)
);
```

При добавлении первичного ключа автоматически создаётся уникальный индекс-В-дерево для столбца или группы столбцов, перечисленных в первичном ключе, и данные столбцы помечаются как NOT NULL.

Таблица может иметь максимум один первичный ключ. (Ограничений уникальности и ограничений NOT NULL, которые функционально почти равнозначны первичным ключам, может быть сколько угодно, но назначить ограничением первичного ключа можно только одно.)

- **внешние ключи**

Внешние ключи позволяют установить связи между таблицами. Внешний ключ устанавливается для столбцов из зависимой, подчиненной таблицы, и указывает на один из столбцов из главной таблицы. Как правило, внешний ключ указывает на первичный ключ из связанной главной таблицы.

Общий синтаксис установки внешнего ключа на уровне таблицы:

```
[CONSTRAINT имя_ограничения]
FOREIGN KEY (столбец1, столбец2, ... столбецN)
REFERENCES главная_таблица (столбец_главной_таблицы1, столбец_главной_таблицы2)
[ON DELETE действие]
[ON UPDATE действие]
```

Для создания ограничения внешнего ключа после FOREIGN KEY указывается столбец таблицы, который будет представлять внешний ключ. После ключевого слова REFERENCES указывается имя связанной таблицы, а затем в скобках имя связанного столбца, на который будет указывать внешний ключ. После выражения REFERENCES идут выражения ON DELETE и ON UPDATE, которые задают действие при удалении и обновлении строки из главной таблицы соответственно.

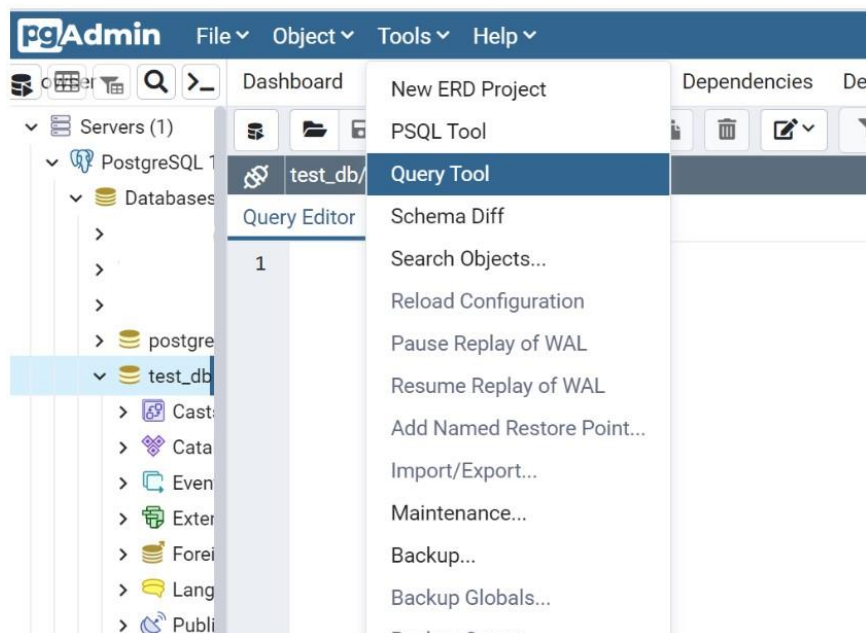
Например, определим две таблицы и свяжем их посредством внешнего ключа:

```
CREATE TABLE customers (
    cust_id INT PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
    age INT,
    first_name VARCHAR(20) NOT NULL,
    last_name VARCHAR(20) NOT NULL,
    phone VARCHAR(20) NOT NULL UNIQUE
);

CREATE TABLE orders (
    ord_id INT PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
    cust_id INT,
    created_at Date,
    FOREIGN KEY (cust_id) REFERENCES customers (cust_id)
);
```

Пример создания таблицы с помощью SQL-запросов

Для написания SQL-запросов откройте **Query Editor** в pgAdmin. Для этого откройте вкладку Tools и выберите Query Tool.



Создаем таблицу Orders, содержащую поля: id, customer_id, order_date, ship_date, paid_date, status.

```
CREATE TABLE IF NOT EXISTS orders(
    order_id int PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
    customer_id int REFERENCES customers(customer_id),
    order_date date NOT NULL default current_date,
    ship_date date,
    paid_date date,
    status char(1)
);
```

- IF NOT EXIST позволяет перед созданием таблицы проверить, что таблицы с таким названием не существует, это помогает избежать лишних ошибок.

Для того, чтобы для поля order_id применялось автоинкрементирование, нужно указать GENERATED BY DEFAULT AS IDENTITY при этом создается последовательность

из integer с автоинкрементом. Ограничение NOT NULL указывает, что ячейке столбца нельзя присваивать значение NULL.

- Для столбца order_id определено ограничение в виде первичного ключа (PRIMARY KEY).
- Для поля order_date определено значение по умолчанию в виде текущей даты. В этом случае при добавлении записи о новом заказе в случае пропуска этого поля оно будет автоматически заполняться значением системной даты.
- Ограничение REFERENCES указывает на то, что столбец является внешним ключом для таблицы customers. В скобках указан столбец, являющийся первичным ключом в этой таблице. Если в БД отсутствует таблица, на которую ссылается внешний ключ, запрос не выполнится. Поэтому сначала создайте таблицу customers, используя этот пример.

Другой вариант – не задавать для столбца customer_id ограничение внешнего ключа, а добавить его после создания таблицы с помощью команды ALTER TABLE.

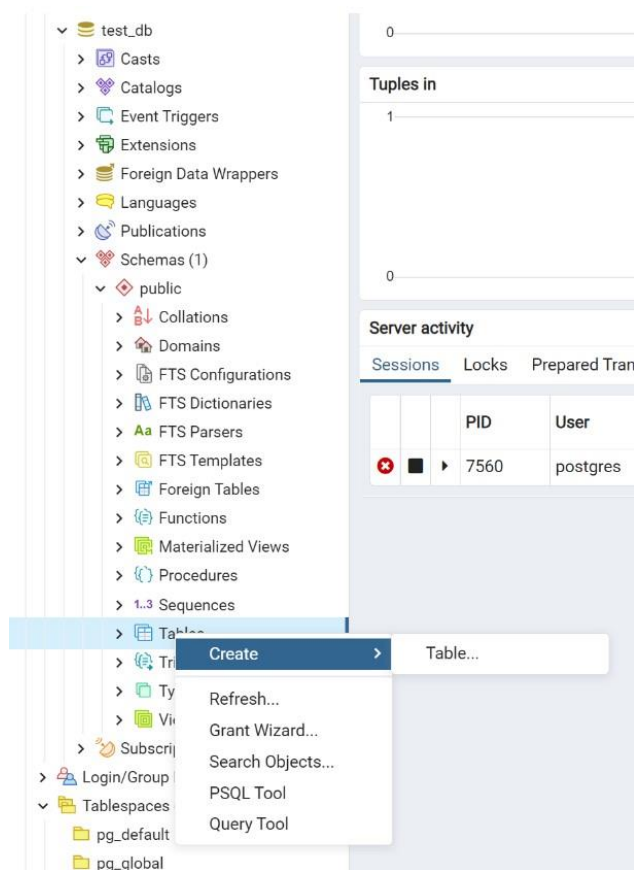
Пример команды для добавления ограничения внешнего ключа, если оно не было указано при создании таблицы:

```
ALTER TABLE orders ADD CONSTRAINT order_customer_fk FOREIGN KEY (customer_id)
REFERENCES customers(customer_id);
```

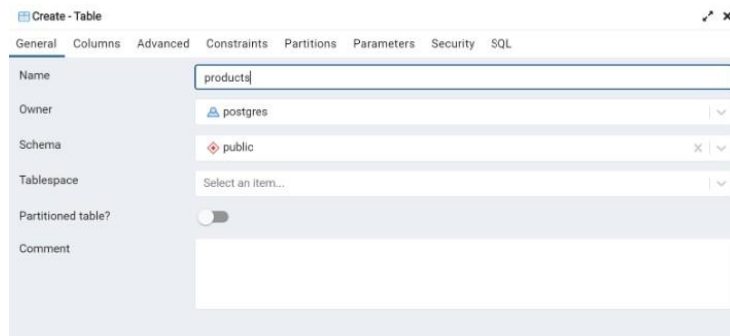
Пример создания таблицы с помощью графического интерфейса PgAdmin

Таблицы можно создавать и в графическом интерфейсе.

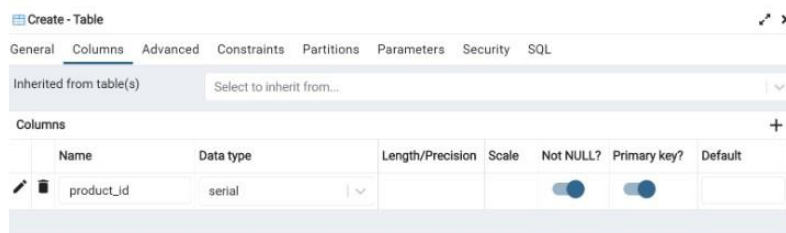
Для создания таблицы необходимо раскрыть Databases → Ваша база данных → Schemas. Нажать правой кнопкой мыши на Tables (Таблицы) и выбрать Create → Table.



В окне создания таблицы ввести название в поле Name.



Столбцы в таблицах можно создать при создании таблицы. Для добавления столбца надо перейти во вкладку Columns, нажать на знак + и ввести данные о столбце. Также при необходимости можно выбрать таблицу, от которой наследуется данная таблицы. Сохранить таблицу. Нажать кнопку «Save».



Использование проверочных ограничений (задание лабораторной работы)

Ограничения на проверку используются для ограничения данных, принимаемых полем, даже если они имеют корректный тип. Например, логично, что дата отправки не может быть раньше даты заказа. Рассмотрим, как создать ограничение на проверку.

1. Во вкладке Constraints=>Check таблицы orders добавьте новую запись.
2. Откройте окно редактирования ограничения и во вкладке Definition введите следующую запись в поле Check: `ship_date >= order_date`

The screenshot shows a database management interface for the 'orders' table. The 'Constraints' tab is active, and the 'Check' sub-tab is selected. A new constraint is being added with the name 'ship_date' and the check expression 'ship_date >= order_date'. The 'Definition' sub-tab is also visible, showing the same expression. At the bottom, there are buttons for 'Close', 'Reset', and 'Save'.

3. Сохраните изменения.

Создайте ограничения для полей `product_price`, `in_stock` таблицы **products** и `quantity` таблицы **items**, запрещающие ввод в них отрицательных значений. Протестируйте созданное вами ограничение, введя в таблицу несколько новых записей, с использованием инструкции INSERT.

Использование операторов INSERT, UPDATE, DELETE

В SQL заполнение таблиц производится при помощи следующей команды:

```
INSERT INTO <имя таблицы>(<список столбцов>) VALUES (<список значений>);
```

где *<имя_таблицы>* - таблица, куда вставляются данные, *<список столбцов>* - список полей, в которые вставляются данные (если он не указывается, то

подразумевается заполнение всех полей). В списке полей поля указываются через запятую, *<список_значений>* - значение полей для вставки через запятую.

Пример: Добавление записи в таблицу orders

```
INSERT INTO orders (order_id, customer_id, ship_date, status) VALUES (1, 1, '12/02/2023', 'S');
```

Из таблицы можно удалить все столбцы, либо отдельные записи. Это осуществляется командой:

```
DELETE FROM <Имя таблицы> [WHERE <Условие>]
```

где *<Условие>* - условие, которым удовлетворяют удаляемые записи, если условие не указано, то удаляются все столбцы таблицы.

Пример: Удалить записи из таблицы orders, у которых поле customer_id = 10.

```
DELETE FROM orders WHERE customer_id = 10;
```

Также существует команда TRUNCATE для быстрого удаления всех данных из таблицы.

```
TRUNCATE <Имя таблицы>
```

Значение полей таблицы можно обновить (изменить), используя следующую команду:

```
UPDATE <Имя таблицы> SET  
    <Имя поля1> = <Выражение1>,  
    [<Имя поля2> = <Выражение2>, ... ]  
    [WHERE <Условие>]
```

Здесь *<Имя поля1>*, *<Имя поля2>* - имена изменяемых полей; *<Выражение1>*, *<Выражение 2>* - значения, которые должны принять поля; *<Условие>* - условие, которым должны соответствовать записи, поля которых изменяем. В качестве выражения можно использовать математические формулы.

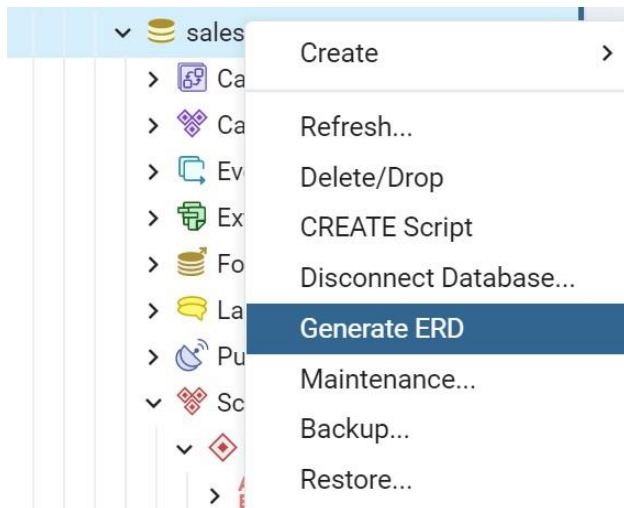
Пример: для таблицы orders установить сегодняшнюю дату отправки для всех обработанных заказов

```
UPDATE orders SET ship_date = CURRENT_DATE WHERE status='P';
```

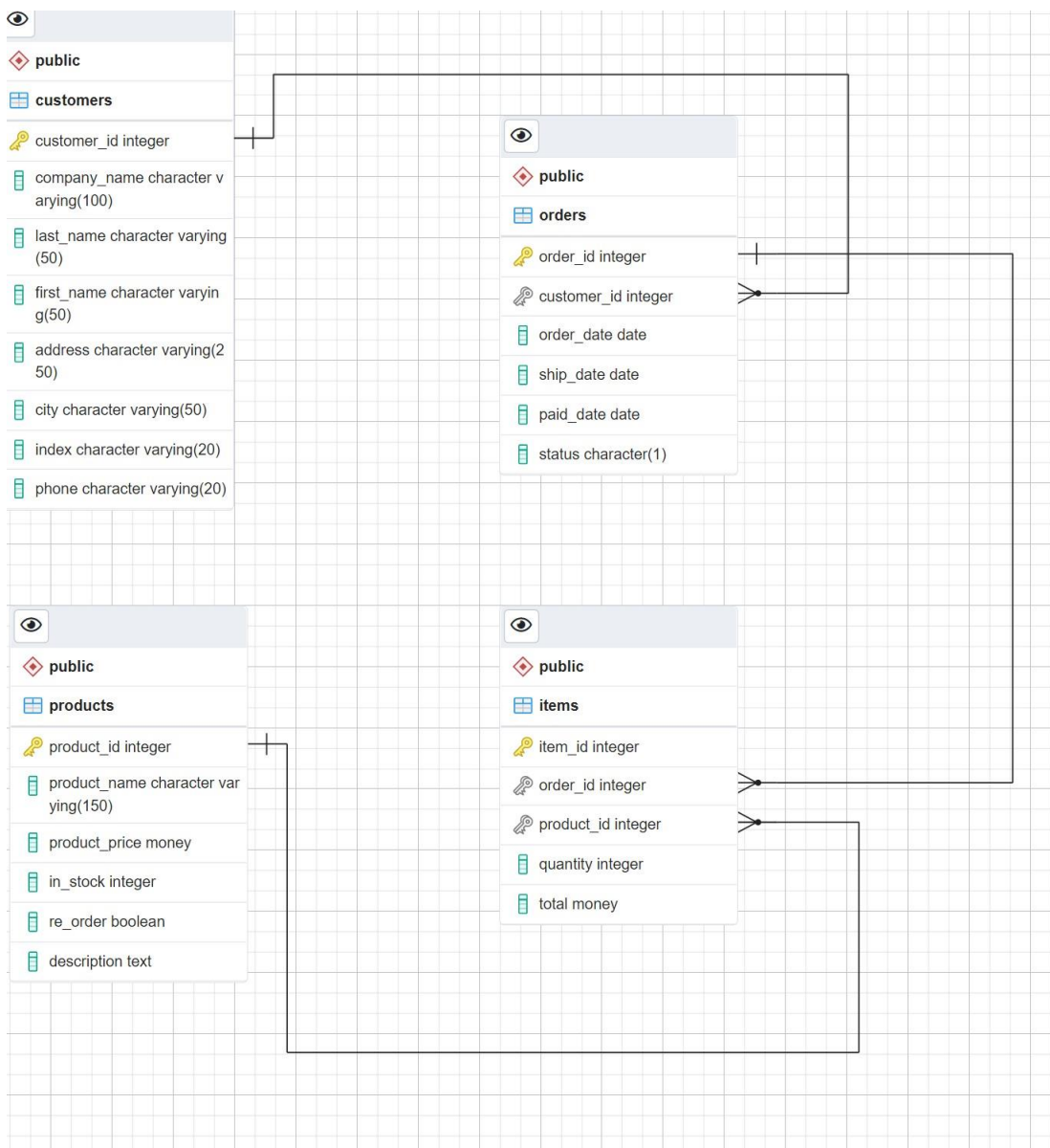
Для того, чтобы вывести все записи таблицы пропишем вместо имен столбцов, используется «*»:

Создание схемы БД

Для создания схемы БД щелкните ПКМ по элементу Базы данных в Обозревателе и выберите Generate ERD.



В результате должна получиться следующая схема:

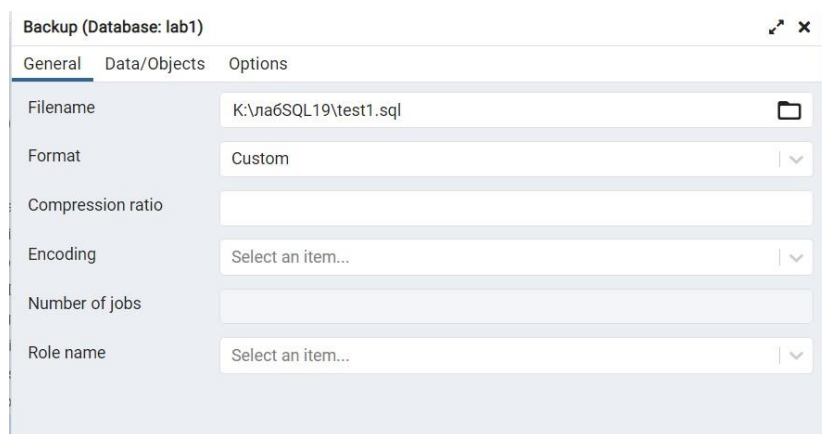


Создание дампа БД

Дамп базы данных – это файл, содержащий структуру и содержимое базы данных.

Создание дампа с помощью pgAdmin:

Заходим в pgAdmin, находим нужную базу данных, нажимаем правой кнопкой мыши на нее и выбираем Backup. В появившемся окне нажимаем на троеточие (выбор папки). Выбираем нужную папку, далее вводим название файла (предварительно нужно создать пустой файл с расширением .sql или .backup. Запускаем Backup.



Контрольные вопросы

1. Что такое СУБД?
2. В чем основное отличие объектно-реляционной СУБД?
3. Назовите основные компоненты БД. Какие функции они выполняют?
4. В чем заключается отличие таблиц от представлений?
5. Что такое psql?
6. Опишите структуру среды pgAdmin.
7. Какие типы данных используются в PostgreSQL?
8. Опишите структуру запроса для создания таблицы.
9. Что такое первичный ключ? Почему первичный ключ должен быть уникальным?
10. Как реализуется связь между таблицами? Объяснить на примере запроса создания связи между таблицами с помощью внешних ключей.
11. Для чего используются проверочные ограничения? Как создать ограничение на проверку?
12. Зачем используются значения по умолчанию? Как установить значение по умолчанию?
13. Опишите структуру команды INSERT для заполнения таблицы.
14. Опишите структуру команды DELETE для удаления записей из таблицы.

15. Опишите структуру команды UPDATE для записей из таблицы.
16. Для чего нужна команда TRUNCATE?
17. Для чего нужна команда ALTER TABLE?
18. Для чего нужна команда DROP TABLE?