Cодержание

[1. Введение. 3](#_Toc510791231)

[2. Задание. 4](#_Toc510791232)

[3. Создать таблицу БД с учетом ограничений целостности данных, содержащую не менее 15 записей. 5](#_Toc510791233)

[4. Используя оператор Select, выдать следующую информацию: 9](#_Toc510791234)

[5. SQL инъекция. 11](#_Toc510791235)

[6. Список литературы 11](#_Toc510791236)

[7. Приложение. Код Python 12](#_Toc510791237)

# Введение.

SQL (Structured Query Language — Структурированный язык запросов) — язык управления базами данных для реляционных баз данных. Сам по себе SQL не является Тьюринг-полным языком программирования, но его стандарт позволяет создавать для него процедурные расширения, которые расширяют его функциональность до полноценного языка программирования.

**DDL** (Data Definition Language / Язык определения данных). К этому типу относятся различные команды, которые создают базу данных, таблицы, индексы, хранимые процедуры и т.д. В общем определяют данные.

* В частности, к этому типу мы можем отнести следующие команды:
* **CREATE**: создает объекты базы данных (саму базу даных, таблицы, индексы и т.д.)
* **ALTER**: изменяет объекты базы данных
* **DROP**: удаляет объекты базы данных
* **TRUNCATE**: удаляет все данные из таблиц
* **SELECT**: извлекает данные из БД
* **UPDATE**: обновляет данные
* **INSERT**: добавляет новые данные
* **DELETE**: удаляет данные

# Задание.

«Автопарк».

Автопарк осуществляет обслуживание заказов на перевозку грузов, используя для этой цели свой парк автомашин и своих водителей. Водитель, выполнивший заказ, получает 20 % от стоимости перевозки.

Управление автопарком должно иметь сведения:

1. об автомашинах: номер машины, марка, пробег на момент приобретения, грузоподъемность;

2. о водителях: табельный номер, фамилия водителя, категория, стаж, адрес, год рождения;

3. о выполненных заказах: дата, фамилия водителя, номер машины, километраж, масса груза, стоимость перевозки.

Необходимо:

1. Создать таблицы БД с учетом ограничений целостности данных и заполнить не менее 15 записей.

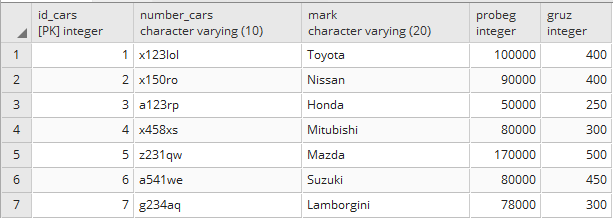
2. Используя оператор Select, выдать следующую информацию:

* по указанному водителю – перечень выполненных заказов за указанный период;
* по указанной машине – общий пробег и общую массу перевезенных грузов;
* по каждому водителю – общее количество поездок, общую массу перевезенных грузов, сумму заработанных денег;
* по автомашине с наибольшим общим пробегом – все сведения.

3. Обеспечить с помощью операторов Insert, Update, Delete обновление информации в указанных таблицах.

# Создать таблицу БД с учетом ограничений целостности данных, содержащую не менее 15 записей.

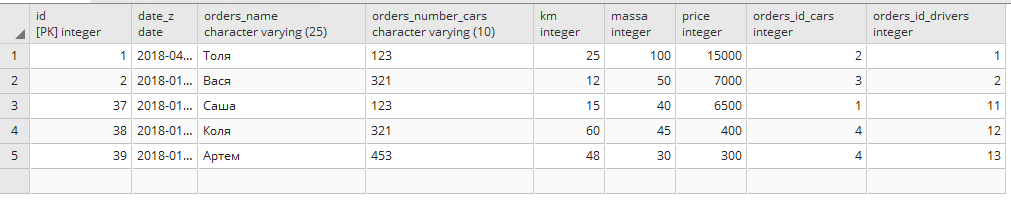
**Создание таблицы Cars**



**Создание таблицы Drivers.**



**Создание таблицы Orders.**



**Скрипт INSERT добавления в таблицу Cars новых записей**

def insert\_cars(conn):  
 cursor = conn.cursor()  
 number = input("Input number\_cars: ")  
 mark = input("Input mark: ")  
 probeg = input("Input probeg")  
 gruz = input("Input gruz: ")  
 sql = "INSERT INTO PUBLIC.\"cars\"(\"number\_cars\", \"mark\", \"probeg\", \"gruz\") VALUES (%s,%s,%s,%s)"  
 cursor.execute(sql, (number, mark, probeg, gruz))

Рисунок 1 Скрипт добавления в таблицу Cars новых записей

**Скрипт INSERT таблицы Driver новыми записями**

cursor = conn.cursor()  
number = input("Input number drivers: ")  
name = input("Input name: ")  
category = input("Input category: ")  
stage = input("Input hstager: ")  
adress = input("Input sadress ")  
year = input("Input year ")  
sql = "INSERT INTO PUBLIC.\"drivers\"(\"number\_drivers\", \"drivers\_name\", \"category\", \  
\"stage\", \"adress\", \"drivers\_year\") VALUES (%s,%s,%s,%s,%s,%s)"  
cursor.execute(sql, (number, name, category, stage, adress, year))  
conn.commit()

*Рисунок 2 Скрипт заполнения таблицы Driver*

**Скрипт INSERT таблицы Orders новыми записями**

cursor = conn.cursor()  
data = input("Input date: ")  
name = input("Input orders\_name ")  
cars = input("Innput orders\_nubmer\_cars: ")  
km = input("Input km: ")  
massa = input("Input massa: ")  
price = input("Input price ")  
id\_cars= input("Input id\_cars ")  
orders\_id\_drivers = input("Input orders\_id\_drivers ")  
sql = "INSERT INTO PUBLIC.\"orders\"(\"date\_z\", \"orders\_nubmer\_cars\", \"category\", \  
\"km\", \"adress\", \"massa\", \"price\", \"orders\_id\_cars\", \"orders\_id\_drivers\") VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s)"cursor.execute(sql, (data, name, cars, km, massa, price, id\_cars, orders\_id\_drivers))  
conn.commit()

**Скрипт DELETE**

deleteid = int(input("Input ID of orders you want to DELETE - "))  
 cursor = conn.cursor()  
 sql = "DELETE FROM PUBLIC.\"orders\" WHERE \"id\" = \'{}\' ".format(deleteid)  
 cursor.execute(sql)  
 conn.commit()  
 print("Succefull deleting")  
def edit\_orders(conn): ###############################################################3

try:  
 cursor = conn.cursor()  
 deleteid = int(input("Input ID of cars you want to DELETE - "))  
 cursor = conn.cursor()  
 sql = "DELETE FROM PUBLIC.\"cers\" WHERE \"id\_cars\" = \'{}\' ".format(deleteid)  
 cursor.execute(sql)  
 conn.commit()  
 print("Succefull deleting")  
except psycopg2.IntegrityError:  
 print("You need to delete orders!")  
 sys.exit()

try:  
 cursor = conn.cursor()  
 deleteid = int(input("Input ID of drivers you want to DELETE - "))  
 cursor = conn.cursor()  
 sql = "DELETE FROM PUBLIC.\"drivers\" WHERE \"id\_drivers\" = \'{}\' ".format(deleteid)  
 cursor.execute(sql)  
 conn.commit()  
 print("Succefull deleting")  
except psycopg2.IntegrityError:  
 print("You need to delete orders!")  
 sys.exit()

**Скрипт update для orders, cars, drivers**

def edit\_drivers(conn):  
 new\_id = input("Input firm ID you want to edit - ")  
 number = input("Input number drivers: ")  
 name = input("Input name: ")  
 category = input("Input category: ")  
 stage = input("Input hstager: ")  
 adress = input("Input sadress ")  
 year = input("Input year ")  
 cursor = conn.cursor()  
 sql = "UPDATE PUBLIC.\"drivers\" SET \"number\_drivers\" = %s, \"drivers\_name\" =%s, category=%s, stage,adress,drivers\_year=%s WHERE \"id\_drivers\" = \'{}\'".format(  
 new\_id)  
 cursor.execute(sql, (number, name, category, stage, adress, datetime(year)))  
 conn.commit()

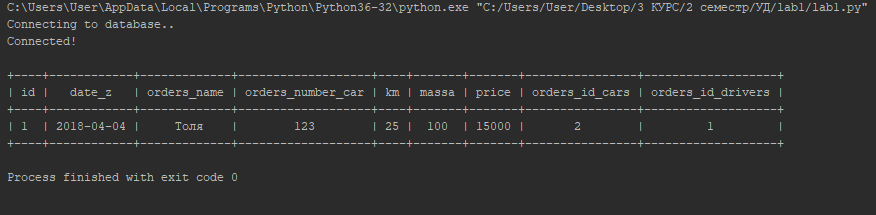
def edit\_cars(conn):  
 cursor = conn.cursor()  
 new\_id = input("Input provider ID you want to edit - ")  
 number = input("Input number\_cars: ")  
 mark = input("Input mark: ")  
 probeg = input("Input probeg")  
 gruz = input("Input gruz: ")  
 cursor = conn.cursor()  
 sql = "UPDATE PUBLIC.\"cars\" SET \"number\_cars\" = %s, \"mark\" =%s, probeg=%s, gruz=%s WHERE \"id\_cars\" = \'{}\'".format(new\_id)  
 cursor.execute(sql, (number, mark, probeg, gruz))  
 conn.commit()

def edit\_orders(conn): ###############################################################3  
 cursor = conn.cursor()  
 new\_id = input("Input provider ID you want to edit - ")  
 data = input("Input date: ")  
 name = input("Input orders\_name ")  
 cars = input("Innput orders\_nubmer\_cars: ")  
 km = input("Input km: ")  
 massa = input("Input massa: ")  
 price = input("Input price ")  
 id\_cars = input("Input id\_cars ")  
 orders\_id\_drivers = input("Input orders\_id\_drivers ")  
 cursor = conn.cursor()  
 sql = "UPDATE PUBLIC.\"orders\" SET \"date\_z\" = %s, \"orders\_name\" =%s, \"orders\_number\_cars\" =%s, \"km=%s\", \"massa=%s\", \"price=%s\", \"orders\_id\_cars=%s\",\"orders\_id\_drivers=\"%s WHERE \"id\" = \'{}\'".format(new\_id)  
 cursor.execute(sql, (data, name, cars, km,massa, price, id\_cars, orders\_id\_drivers))  
 conn.commit()

# Используя оператор Select, выдать следующую информацию:

**По указанному водителю – перечень выполненных заказов за указанный период**

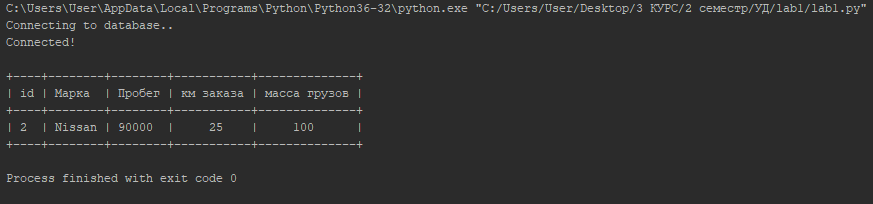
"SELECT id, date\_z, orders\_name, orders\_number\_cars, km, massa, price, orders\_id\_cars, orders\_id\_drivers FROM PUBLIC.cars, public.orders\  
where date\_z BETWEEN '2012-01-05 00:00:00' AND '2018-06-06 00:00:00' AND orders\_name = 'Толя'\  
GROUP BY id"



*Рисунок 4 Результат выполнения скрипта*

**По указанной машине – общий пробег и общую массу перевезенных грузов**

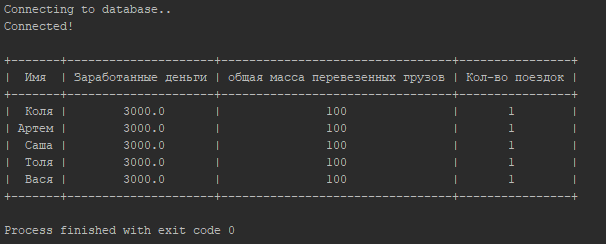
"SELECT id\_cars, mark, probeg, *sum*(orders.km),*sum*(massa) FROM PUBLIC.cars, public.orders\  
where id\_cars=orders\_id\_cars and id\_cars='2'\  
GROUP BY id\_cars"



*Рисунок 5 Результат выполнения скрипта*

**По каждому водителю – общее количество поездок, общую массу перевезенных грузов, сумму заработанных денег**

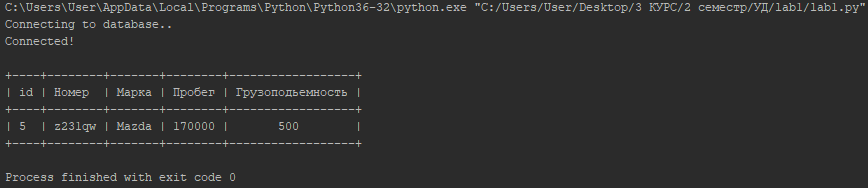
"SELECT drivers\_name, *SUM*(price)\*0.2 AS price, *SUM*(massa) AS massa, *COUNT*(drivers\_name) AS km FROM PUBLIC.drivers, public.orders\  
where date\_z BETWEEN '2012-01-05 00:00:00' AND '2018-06-06 00:00:00' AND orders\_name = 'Толя'\  
GROUP BY drivers\_name"



*Рисунок 6 Результат выполнения скрипта*

**По автомашине с наибольшим общим пробегом – все сведения**

SELECT \* FROM public.cars\  
WHERE probeg = (SELECT *MAX*(probeg) FROM public.cars)"

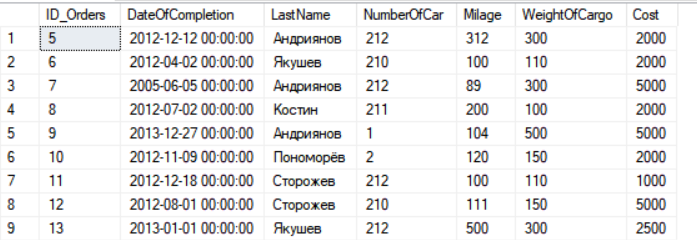


*Рисунок 7 Результат выполнения скрипта*

# SQL инъекция.

SELECT \* FROM Orders

WHERE NumberOfCar = ' ' or '1' = '1'



*Рисунок 8 Результат выполнения инъекции*

# Список литературы

1) Конноли Т., Бегг К. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. 3-е издание. - М.: Издательский дом "Вильямс", 2013. - 1440 с.

2) Дейт К. Дж. Введение в системы баз данных, 8-е издание. . - М.: Издательский дом "Вильямс", 2006. - 1328 с.

3.Кузнецов М. В., Симдянов И. В. MySQL5.- СПб.: БХВ-Петербург, 2006. - 1024 с.

4.Моисеенко С. И. SQL. Задачи и решения. - СПб.: Питер, 2006. - 256 с.: ил.

# Приложение. Код Python

import psycopg2 # Import library postgres  
from prettytable import PrettyTable  
import sys  
import datetime  
flag = True  
try:  
 print("Connecting to database..")  
 conn = psycopg2.connect("host=" + "localhost" + " port=" + "5432" + " dbname=" + "test" + " user=" + "postgres" + " password=" + "10217200")  
 print("Connected!\n")  
except psycopg2.ProgrammingError:  
 print("Connection error.")  
 flag = False  
  
  
def show\_drivers(conn):  
 cursor = conn.cursor()  
 cursor.execute("SELECT \* FROM drivers") # Selecting all columns from table seller  
 row = cursor.fetchone()  
 table = PrettyTable(('id\_drivers','number\_drivers','drivers\_name','category','stage','adress','driving\_year'))  
 while row is not None: # Fetching row  
 table.add\_row(row)  
 row = cursor.fetchone()  
 cursor.close()  
 return table  
  
def insert\_drivers(conn):  
 cursor = conn.cursor()  
 number = input("Input number drivers: ")  
 name = input("Input name: ")  
 category = input("Input category: ")  
 stage = input("Input hstager: ")  
 adress = input("Input sadress ")  
 year = input("Input year ")  
 sql = "INSERT INTO PUBLIC.\"drivers\"(\"number\_drivers\", \"drivers\_name\", \"category\", \  
 \"stage\", \"adress\", \"drivers\_year\") VALUES (%s,%s,%s,%s,%s,%s)"  
 cursor.execute(sql, (number, name, category, stage, adress, year))  
 conn.commit()  
  
  
def edit\_drivers(conn):  
 new\_id = input("Input firm ID you want to edit - ")  
 number = input("Input number drivers: ")  
 name = input("Input name: ")  
 category = input("Input category: ")  
 stage = input("Input hstager: ")  
 adress = input("Input sadress ")  
 year = input("Input year ")  
 cursor = conn.cursor()  
 sql = "UPDATE PUBLIC.\"drivers\" SET \"number\_drivers\" = %s, \"drivers\_name\" =%s, category=%s, stage,adress,drivers\_year=%s WHERE \"id\_drivers\" = \'{}\'".format(  
 new\_id)  
 cursor.execute(sql, (number, name, category, stage, adress, datetime(year)))  
 conn.commit()  
  
def delete\_drivers(conn):  
 try:  
 cursor = conn.cursor()  
 deleteid = int(input("Input ID of drivers you want to DELETE - "))  
 cursor = conn.cursor()  
 sql = "DELETE FROM PUBLIC.\"drivers\" WHERE \"id\_drivers\" = \'{}\' ".format(deleteid)  
 cursor.execute(sql)  
 conn.commit()  
 print("Succefull deleting")  
 except psycopg2.IntegrityError:  
 print("You need to delete orders!")  
 sys.exit()  
  
  
  
def show\_cars(conn):  
 cursor = conn.cursor()  
 cursor.execute("SELECT \* FROM cars") # Selecting all columns from table seller  
 row = cursor.fetchone()  
 table = PrettyTable(('id\_cars','number\_cars','mark','probeg','gruz'))  
 while row is not None: # Fetching row  
 table.add\_row(row)  
 row = cursor.fetchone()  
 cursor.close()  
 return table  
  
def delete\_cars(conn):  
 try:  
 cursor = conn.cursor()  
 deleteid = int(input("Input ID of cars you want to DELETE - "))  
 cursor = conn.cursor()  
 sql = "DELETE FROM PUBLIC.\"cers\" WHERE \"id\_cars\" = \'{}\' ".format(deleteid)  
 cursor.execute(sql)  
 conn.commit()  
 print("Succefull deleting")  
 except psycopg2.IntegrityError:  
 print("You need to delete orders!")  
 sys.exit()  
  
def edit\_cars(conn):  
 cursor = conn.cursor()  
 new\_id = input("Input provider ID you want to edit - ")  
 number = input("Input number\_cars: ")  
 mark = input("Input mark: ")  
 probeg = input("Input probeg")  
 gruz = input("Input gruz: ")  
 cursor = conn.cursor()  
 sql = "UPDATE PUBLIC.\"cars\" SET \"number\_cars\" = %s, \"mark\" =%s, probeg=%s, gruz=%s WHERE \"id\_cars\" = \'{}\'".format(new\_id)  
 cursor.execute(sql, (number, mark, probeg, gruz))  
 conn.commit()  
  
def insert\_cars(conn):  
 cursor = conn.cursor()  
 number = input("Input number\_cars: ")  
 mark = input("Input mark: ")  
 probeg = input("Input probeg")  
 gruz = input("Input gruz: ")  
 sql = "INSERT INTO PUBLIC.\"cars\"(\"number\_cars\", \"mark\", \"probeg\", \  
 \"gruz\") VALUES (%s,%s,%s,%s)"  
 cursor.execute(sql, (number, mark, probeg, gruz))  
  
def show\_orders(conn):  
 cursor = conn.cursor()  
 cursor.execute("SELECT \* FROM orders") # Selecting all columns from table seller  
 row = cursor.fetchone()  
 table = PrettyTable(('id','date\_z','orders\_name','orders\_nubmer\_cars','km','massa','price','orders\_id\_cars','orders\_id\_drivers'))  
 while row is not None: # Fetching row  
 table.add\_row(row)  
 row = cursor.fetchone()  
 cursor.close()  
 return table  
  
def insert\_orders(conn):  
 cursor = conn.cursor()  
 data = input("Input date: ")  
 name = input("Input orders\_name ")  
 cars = input("Innput orders\_nubmer\_cars: ")  
 km = input("Input km: ")  
 massa = input("Input massa: ")  
 price = input("Input price ")  
 id\_cars= input("Input id\_cars ")  
 orders\_id\_drivers = input("Input orders\_id\_drivers ")  
 sql = "INSERT INTO PUBLIC.\"orders\"(\"date\_z\", \"orders\_nubmer\_cars\", \"category\", \  
 \"km\", \"adress\", \"massa\", \"price\", \"orders\_id\_cars\", \"orders\_id\_drivers\") VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s)"  
 cursor.execute(sql, (data, name, cars, km, massa, price, id\_cars, orders\_id\_drivers))  
 conn.commit()  
  
  
def delete\_orders(conn):  
 deleteid = int(input("Input ID of orders you want to DELETE - "))  
 cursor = conn.cursor()  
 sql = "DELETE FROM PUBLIC.\"orders\" WHERE \"id\" = \'{}\' ".format(deleteid)  
 cursor.execute(sql)  
 conn.commit()  
 print("Succefull deleting")  
def edit\_orders(conn): ###############################################################3  
 cursor = conn.cursor()  
 new\_id = input("Input provider ID you want to edit - ")  
 data = input("Input date: ")  
 name = input("Input orders\_name ")  
 cars = input("Innput orders\_nubmer\_cars: ")  
 km = input("Input km: ")  
 massa = input("Input massa: ")  
 price = input("Input price ")  
 id\_cars = input("Input id\_cars ")  
 orders\_id\_drivers = input("Input orders\_id\_drivers ")  
 cursor = conn.cursor()  
 sql = "UPDATE PUBLIC.\"orders\" SET \"date\_z\" = %s, \"orders\_name\" =%s, \"orders\_number\_cars\" =%s, \"km=%s\", \"massa=%s\", \"price=%s\", \"orders\_id\_cars=%s\",\"orders\_id\_drivers=\"%s WHERE \"id\" = \'{}\'".format(new\_id)  
 cursor.execute(sql, (data, name, cars, km,massa, price, id\_cars, orders\_id\_drivers))  
 conn.commit()  
  
  
  
  
  
  
  
#по указанному водителю – перечень выполненных заказов за указанный период;  
def zap1(conn):  
 cursor = conn.cursor()  
 cursor.execute("SELECT id, date\_z, orders\_name, orders\_number\_cars, km, massa, price, orders\_id\_cars, orders\_id\_drivers FROM PUBLIC.cars, public.orders\  
 where date\_z BETWEEN '2012-01-05 00:00:00' AND '2018-06-06 00:00:00' AND orders\_name = 'Толя'\  
 GROUP BY id")  
 row = cursor.fetchone()  
 table = PrettyTable(('id', 'date\_z', 'orders\_name', 'orders\_number\_car', 'km', 'massa', 'price', 'orders\_id\_cars', 'orders\_id\_drivers'))  
 while row is not None:  
 table.add\_row(row)  
 row = cursor.fetchone()  
 cursor.close()  
 return table  
#по указанной машине – общий пробег и общую массу перевезенных грузов;  
def zap2(conn):  
 cursor = conn.cursor()  
 cursor.execute("SELECT id\_cars, mark, probeg, *sum*(orders.km),*sum*(massa) FROM PUBLIC.cars, public.orders\  
 where id\_cars=orders\_id\_cars and id\_cars='2'\  
 GROUP BY id\_cars")  
 row = cursor.fetchone()  
 table = PrettyTable(('id', 'Марка', 'Пробег', 'км заказа', 'масса грузов' ))  
 while row is not None:  
 table.add\_row(row)  
 row = cursor.fetchone()  
 cursor.close()  
 return table  
  
def zap3(conn):  
 cursor = conn.cursor()  
 cursor.execute("SELECT drivers\_name, *SUM*(price)\*0.2 AS price, *SUM*(massa) AS massa, *COUNT*(drivers\_name) AS km FROM PUBLIC.drivers, public.orders\  
 where date\_z BETWEEN '2012-01-05 00:00:00' AND '2018-06-06 00:00:00' AND orders\_name = 'Толя'\  
 GROUP BY drivers\_name")  
 row = cursor.fetchone()  
 table = PrettyTable(('Имя', 'Заработанные деньги', 'общая масса перевезенных грузов', 'Кол-во поездок'))  
 while row is not None:  
 table.add\_row(row)  
 row = cursor.fetchone()  
 cursor.close()  
 return table  
  
#• по автомашине с наибольшим общим пробегом – все сведения.  
def zap4(conn):  
 cursor = conn.cursor()  
 cursor.execute(" SELECT \* FROM public.cars\  
 WHERE probeg = (SELECT *MAX*(probeg) FROM public.cars)")  
 row = cursor.fetchone()  
 table = PrettyTable(('id', 'Номер', 'Марка', 'Пробег', 'Грузоподьемность'))  
 while row is not None:  
 table.add\_row(row)  
 row = cursor.fetchone()  
 cursor.close()  
 return table  
print(zap4(conn))  
conn.close()