Cодержание

[1. Введение. 3](#_Toc513144117)

[2. Задание. 4](#_Toc513144118)

[3. Создать таблицу БД с учетом ограничений целостности данных, содержащую не менее 15 записей. 5](#_Toc513144119)

[4. Используя оператор Select, создать пять запросов с агрегатными функциями 5](#_Toc513144120)

[5. Создать запрос: вывести всех водителей, у которых в таблице заказов масса груза от 50 до 120, стоимость перевозки меньше 4000, километраж больше 10. 8](#_Toc513144121)

[6. Список литературы 9](#_Toc513144122)

[7. Приложение. Код Python 9](#_Toc513144123)

# Введение.

* Агрегатные функции выполняют вычисления над значениями в наборе строк. В T-SQL имеются следующие агрегатные функции:
* **AVG**: находит среднее значение
* **SUM**: находит сумму значений
* **MIN**: находит наименьшее значение
* **MAX**: находит наибольшее значение
* **COUNT**: находит количество строк в запросе
* В качестве аргумента все агрегатные функции принимают выражение, которое представляет критерий дя определения значений. Зачастую, в качестве выражения выступает название столбца, над значениями которого надо проводить вычисления.
* Выражения в функциях AVG и SUM должно представлять числовое значение. Выражение в функциях MIN, MAX и COUNT может представлять числовое или строковое значение или дату.
* Все агрегатные функции за исключением COUNT(\*) игнорируют значения NULL.

# 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:

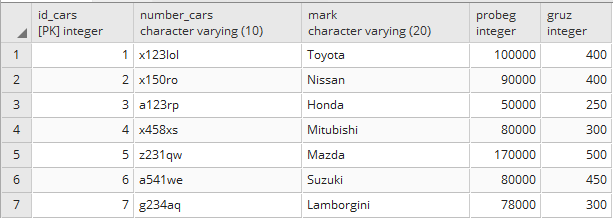
1. Создать таблицы БД с учетом ограничений целостности данных и заполнить не менее 15 записей.

2. Используя оператор Select, создать пять запросов с агрегатными функциями

3. Создать запрос: вывести всех водителей у которых в таблице заказо масса груза от 50 до 120, стоимость перевозки меньше 4000, километраж больше 10.

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

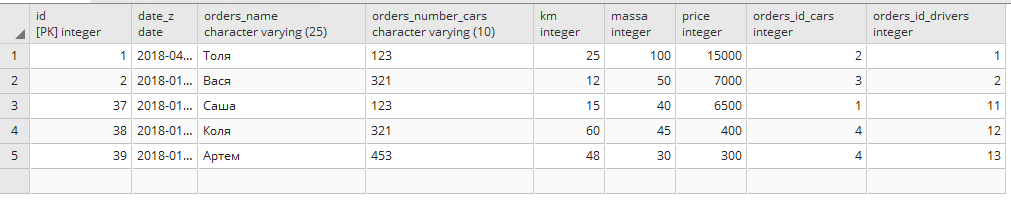
**Создание таблицы Cars**



**Создание таблицы Drivers.**

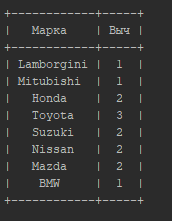


**Создание таблицы Orders.**



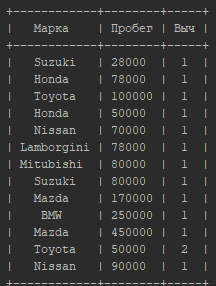
# Используя оператор Select, создать пять запросов с агрегатными функциями

1. **Подсчитать количество совпадающих марок машин. Агрегатная функция COUNT**



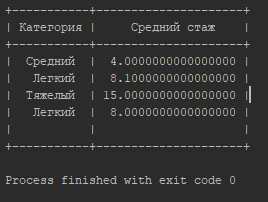
def zap5(conn):  
 cursor = conn.cursor()  
 cursor.execute("SELECT Mark, *COUNT*(\*) FROM Cars\  
 GROUP BY Mark;")  
 row = cursor.fetchone()  
 table = PrettyTable(('Марка', 'Выч'))  
 while row is not None:  
 table.add\_row(row)  
 row = cursor.fetchone()  
 cursor.close()  
 return table

1. **Агрегатная функция COUNT**



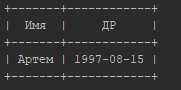
def zap6(conn):  
 cursor = conn.cursor()  
 cursor.execute("SELECT Mark, probeg, *COUNT*(\*) FROM Cars\  
 GROUP BY Mark, probeg")  
 row = cursor.fetchone()  
 table = PrettyTable(('Марка', 'Пробег', 'Выч'))  
 while row is not None:  
 table.add\_row(row)  
 row = cursor.fetchone()  
 cursor.close()  
 return table

1. **Агрегатная функция AVG**



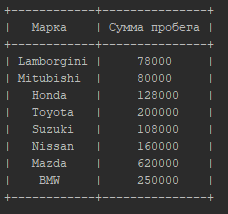
def zap7(conn):  
 cursor = conn.cursor()  
 cursor.execute("SELECT category, *AVG*(stage) FROM Drivers\  
 GROUP BY Category")  
 row = cursor.fetchone()  
 table = PrettyTable(('Категория', 'Средний стаж'))  
 while row is not None:  
 table.add\_row(row)  
 row = cursor.fetchone()  
 cursor.close()  
 return table

1. **Самый младший водитель**



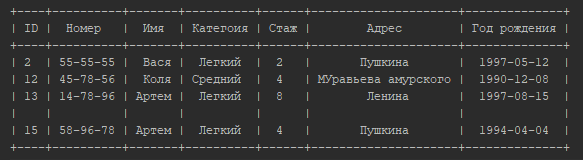
def zap8(conn):  
 cursor = conn.cursor()  
 cursor.execute("SELECT drivers\_name, drivers\_year FROM Drivers\  
 WHERE drivers\_year = (SELECT *MAX*(drivers\_year) from Drivers);")  
 row = cursor.fetchone()  
 table = PrettyTable(('Имя', 'ДР'))  
 while row is not None:  
 table.add\_row(row)  
 row = cursor.fetchone()  
 cursor.close()  
 return table

1. **Общий пробег марки**



def zap9(conn):  
 cursor = conn.cursor()  
 cursor.execute("SELECT Mark, *SUM*(probeg) FROM Cars\  
 GROUP BY Mark;")  
 row = cursor.fetchone()  
 table = PrettyTable(('Марка', 'Сумма пробега'))  
 while row is not None:  
 table.add\_row(row)  
 row = cursor.fetchone()  
 cursor.close()  
 return table

# Создать запрос: вывести всех водителей, у которых в таблице заказов масса груза от 50 до 120, стоимость перевозки меньше 4000, километраж больше 10.



def zap10(conn):  
 cursor = conn.cursor()  
 cursor.execute("SELECT drivers.\* FROM drivers\  
 JOIN orders ON Drivers.drivers\_name = orders.orders\_name\  
 WHERE orders.massa BETWEEN 10 AND 120\  
 AND orders.price < 4000\  
 AND orders.km > 10")  
 row = cursor.fetchone()  
 table = PrettyTable(('ID', 'Номер', 'Имя','Категоия','Стаж','Адрес','Год рождения'))  
 while row is not None:  
 table.add\_row(row)  
 row = cursor.fetchone()  
 cursor.close()  
 return table

# 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

#• Агрегатная функция COUNT: подсчитать количество одинаковых марок.  
def zap5(conn):  
 cursor = conn.cursor()  
 cursor.execute("SELECT Mark, *COUNT*(\*) FROM Cars\  
 GROUP BY Mark;")  
 row = cursor.fetchone()  
 table = PrettyTable(('Марка', 'Количество'))  
 while row is not None:  
 table.add\_row(row)  
 row = cursor.fetchone()  
 cursor.close()  
 return table  
  
#COUNT 2  
def zap6(conn):  
 cursor = conn.cursor()  
 cursor.execute("SELECT Mark, probeg, *COUNT*(\*) FROM Cars\  
 GROUP BY Mark, probeg")  
 row = cursor.fetchone()  
 table = PrettyTable(('Марка', 'Пробег', 'Выч'))  
 while row is not None:  
 table.add\_row(row)  
 row = cursor.fetchone()  
 cursor.close()  
 return table  
  
  
#AVG  
def zap7(conn):  
 cursor = conn.cursor()  
 cursor.execute("SELECT category, *AVG*(stage) FROM Drivers\  
 GROUP BY Category")  
 row = cursor.fetchone()  
 table = PrettyTable(('Категория', 'Средний стаж'))  
 while row is not None:  
 table.add\_row(row)  
 row = cursor.fetchone()  
 cursor.close()  
 return table  
  
#Самый младший водитель  
def zap8(conn):  
 cursor = conn.cursor()  
 cursor.execute("SELECT drivers\_name, drivers\_year FROM Drivers\  
 WHERE drivers\_year = (SELECT *MAX*(drivers\_year) from Drivers);")  
 row = cursor.fetchone()  
 table = PrettyTable(('Имя', 'ДР'))  
 while row is not None:  
 table.add\_row(row)  
 row = cursor.fetchone()  
 cursor.close()  
 return table  
  
#Общий пробег марки  
def zap9(conn):  
 cursor = conn.cursor()  
 cursor.execute("SELECT Mark, *SUM*(probeg) FROM Cars\  
 GROUP BY Mark;")  
 row = cursor.fetchone()  
 table = PrettyTable(('Марка', 'Сумма пробега'))  
 while row is not None:  
 table.add\_row(row)  
 row = cursor.fetchone()  
 cursor.close()  
 return table  
  
  
#5.Создать запрос: вывести всех водителей, у которых в таблице заказов масса груза от 50 до 120, стоимость перевозки меньше 4000, километраж больше 10  
def zap10(conn):  
 cursor = conn.cursor()  
 cursor.execute("SELECT drivers.\* FROM drivers\  
 JOIN orders ON Drivers.drivers\_name = orders.orders\_name\  
 WHERE orders.massa BETWEEN 10 AND 120\  
 AND orders.price < 4000\  
 AND orders.km > 10")  
 row = cursor.fetchone()  
 table = PrettyTable(('ID', 'Номер', 'Имя','Категоия','Стаж','Адрес','Год рождения'))  
 while row is not None:  
 table.add\_row(row)  
 row = cursor.fetchone()  
 cursor.close()  
 return table