

Министерство образования Республики Беларусь

Учреждение образования  
Белорусский государственный университет информатики и  
радиоэлектроники

Факультет компьютерного проектирования  
Кафедра инженерной психологии и эргономики  
Дисциплина: Базы данных

Лабораторная работа № 2

«Выборка данных с использованием предложения SELECT»

Выполнил:  
ст.гр. 113802  
Разумов Д.А.

Проверила:  
Василькова А.Н.

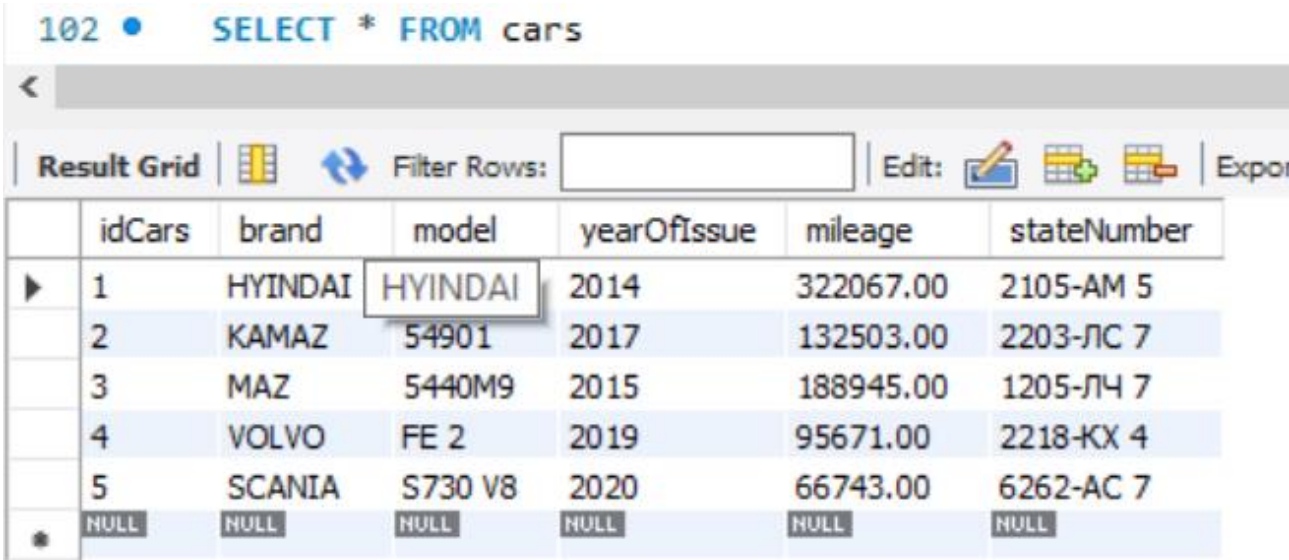
Минск 2022

Итак, в нашей БД Автогараж есть три таблицы: routs (маршруты), drivers (водители) и traffics (машины). И мы хотим посмотреть, какие данные в них содержатся. Для этого в SQL существует оператор SELECT. Синтаксис его использования, следующий:

SELECT что\_выбрать FROM откуда\_выбрать;

Вместо "что\_выбрать" мы должны указать либо имя столбца, значения которого хотим увидеть, либо имена нескольких столбцов через запятую, либо символ звездочки (\*), означающий выбор всех столбцов таблицы. Вместо "откуда\_выбрать" следует указать имя таблицы.

На рисунке 1 приведен пример запроса с оператором SELECT.



	idCars	brand	model	yearOfIssue	mileage	stateNumber
▶	1	HYINDAI	HYINDAI	2014	322067.00	2105-AM 5
	2	KAMAZ	54901	2017	132503.00	2203-ЛС 7
	3	MAZ	5440M9	2015	188945.00	1205-ЛЧ 7
	4	VOLVO	FE 2	2019	95671.00	2218-KX 4
	5	SCANIA	S730 V8	2020	66743.00	6262-AC 7
•	NULL	NULL	NULL	NULL	NULL	NULL

Рисунок 1 – Пример запроса с SELECT

Мы увидели все данные, которые вносили в эту таблицу. Но предположим, что мы хотим посмотреть только столбец brand. Для этого в запросе мы укажем имя этого столбца:

SELECT brand FROM routs;

На рисунке 2 выведен только столбец brand.

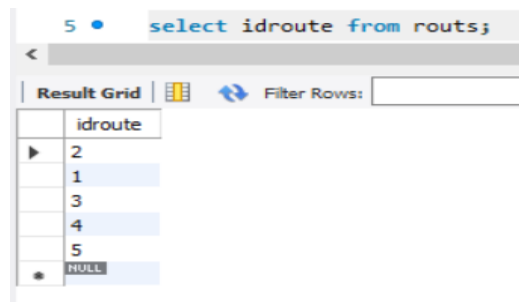


Рисунок 2 – Пример запроса SELECT только для одного столбца

Если мы захотим посмотреть, например, brand и model маршрутов, то перечислим интересующие столбцы через запятую:

`SELECT brand, model FROM routs;`

На рисунке 3 приведен данный пример.

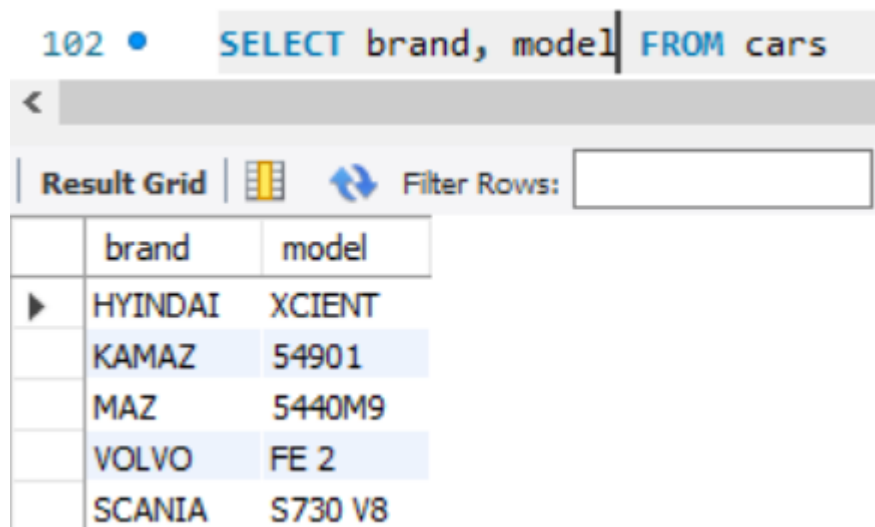


Рисунок 3 - Пример запроса SELECT для двух определенных столбцов

В настоящее время имеется 4 темы, а если их будет 100? Хотелось бы, чтобы они выводились, например, по алфавиту. Для этого в SQL существует ключевое слово ORDER BY после которого указывается имя столбца, по которому будет происходить сортировка. Пример работы данного оператора приведен на рисунке 4.

```

102 • SELECT startPoint, endPoint, trafficLength FROM traffics
103     ORDER BY trafficLength;

```

	startPoint	endPoint	trafficLength
▶	Солигорск	Киев	469
	Минск	Варшава	557
	Брест	Москва	1062
	Минск	Прага	1297
	Могилев	Берлин	1313

Рисунок 4 – Пример работы оператора ORDER BY

По умолчанию сортировка идет по возрастанию. Для сортировки по убыванию значений – необходимо использовать ключевое слово DESC. Пример работы данного оператора представлен на рисунке 5.

```

102 • SELECT startPoint, endPoint, trafficLength FROM traffics
103     ORDER BY trafficLength DESC;

```

	startPoint	endPoint	trafficLength
▶	Могилев	Берлин	1313
	Минск	Прага	1297
	Брест	Москва	1062
	Минск	Варшава	557
	Солигорск	Киев	469


Рисунок 5 – Пример работы операторов ORDER BY и DESC

Например, мы хотим узнать, у какого маршрута trafficLength > 400 или trafficLength < 600. Для этого в SQL есть ключевое слово WHERE, синтаксис у такого запроса, следующий:

SELECT имя\_столбца FROM имя\_таблицы WHERE условие;

Пример работы данного оператора показан на рисунке 6 - 8.

```
102 • SELECT startPoint, endPoint, trafficLength FROM traffics
103 WHERE trafficLength = 1062;
```

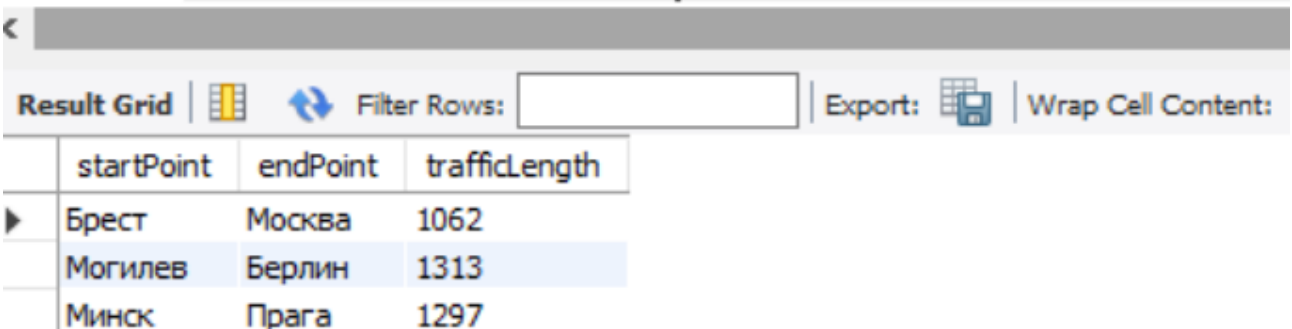


The screenshot shows a database query interface. At the top, a SQL query is entered: `SELECT startPoint, endPoint, trafficLength FROM traffics WHERE trafficLength = 1062;`. Below the query, there is a toolbar with options like 'Result Grid', 'Filter Rows', 'Export', and 'Wrap Cell Content'. The 'Result Grid' is active, displaying a table with three columns: 'startPoint', 'endPoint', and 'trafficLength'. The table contains one row: 'Брест', 'Москва', and '1062'.

startPoint	endPoint	trafficLength
Брест	Москва	1062

Рисунок 6 – Пример работы оператора '=' и оператора WHERE

```
102 • SELECT startPoint, endPoint, trafficLength FROM traffics
103 WHERE trafficLength > 800;
```

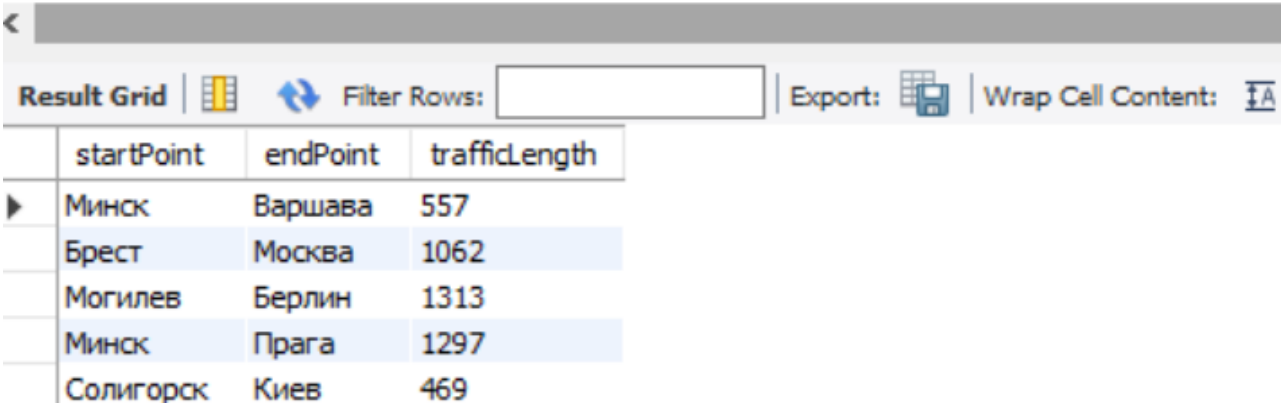


The screenshot shows a database query interface. At the top, a SQL query is entered: `SELECT startPoint, endPoint, trafficLength FROM traffics WHERE trafficLength > 800;`. Below the query, there is a toolbar with options like 'Result Grid', 'Filter Rows', 'Export', and 'Wrap Cell Content'. The 'Result Grid' is active, displaying a table with three columns: 'startPoint', 'endPoint', and 'trafficLength'. The table contains three rows: 'Брест', 'Москва', '1062'; 'Могилев', 'Берлин', '1313'; and 'Минск', 'Прага', '1297'.

startPoint	endPoint	trafficLength
Брест	Москва	1062
Могилев	Берлин	1313
Минск	Прага	1297

Рисунок 7 – Пример работы оператора '≥' и оператора WHERE

```
102 • SELECT startPoint, endPoint, trafficLength FROM traffics
103 WHERE trafficLength IS NOT NULL;
```



The screenshot shows a database query interface. At the top, a SQL query is entered: `SELECT startPoint, endPoint, trafficLength FROM traffics WHERE trafficLength IS NOT NULL;`. Below the query, there is a toolbar with options like 'Result Grid', 'Filter Rows', 'Export', and 'Wrap Cell Content'. The 'Result Grid' is active, displaying a table with three columns: 'startPoint', 'endPoint', and 'trafficLength'. The table contains five rows: 'Минск', 'Варшава', '557'; 'Брест', 'Москва', '1062'; 'Могилев', 'Берлин', '1313'; 'Минск', 'Прага', '1297'; and 'Солигорск', 'Киев', '469'.

startPoint	endPoint	trafficLength
Минск	Варшава	557
Брест	Москва	1062
Могилев	Берлин	1313
Минск	Прага	1297
Солигорск	Киев	469

Рисунок 8 – Пример работы оператора IS NOT NULL и оператора WHERE

Поиск с использованием метасимволов может осуществляться только в текстовых полях. Самый распространенный метасимвол - %. Он означает любые символы. Например, если нам надо найти слова, начинающиеся с букв "вел", то мы напишем LIKE 'вел%', а если мы хотим найти слова, которые содержат символы "клуб", то мы напишем LIKE '%клуб%'. Пример показан на рисунке 9.

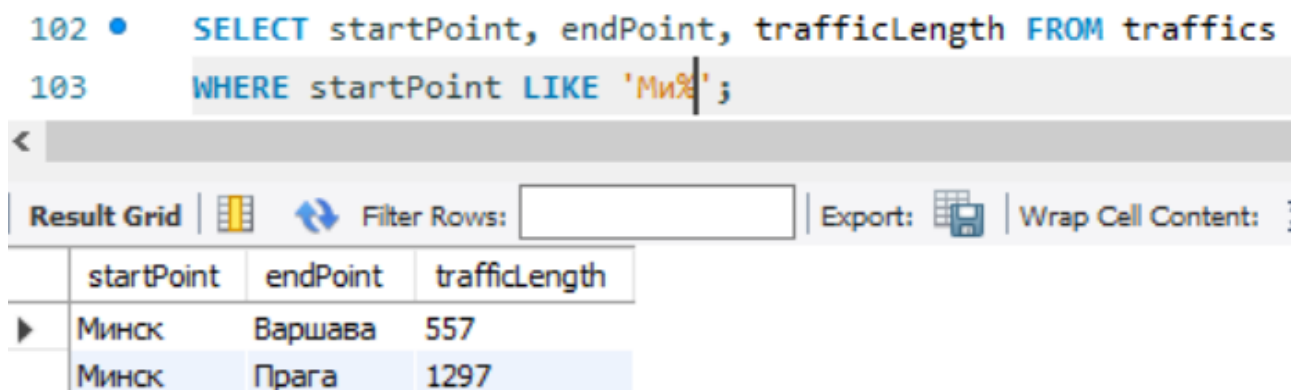


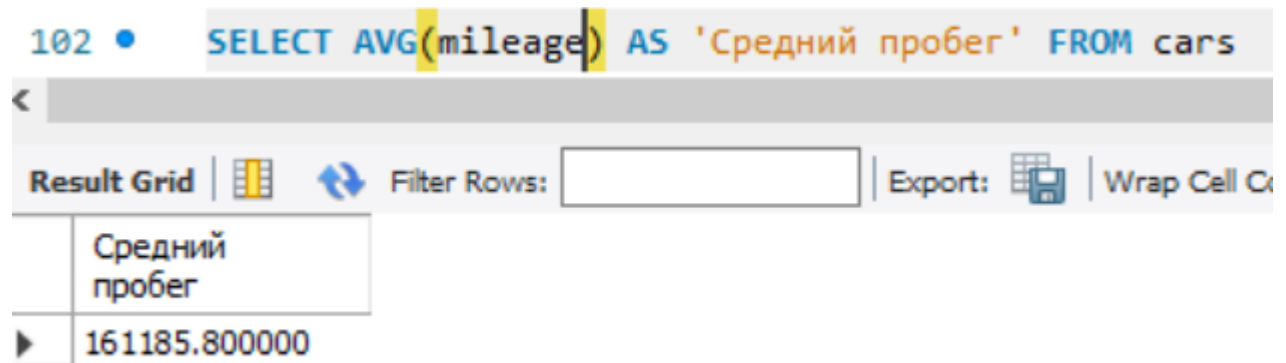
Рисунок 9 – Пример работы оператора LIKE и оператора WHERE

Агрегатные функции вычисляют некоторые скалярные значения в наборе строк. В MySQL есть следующие агрегатные функции:

- AVG: вычисляет среднее значение
- SUM: вычисляет сумму значений
- MIN: вычисляет наименьшее значение
- MAX: вычисляет наибольшее значение
- COUNT: вычисляет количество строк в запросе

Все агрегатные функции принимают в качестве параметра выражение, которое представляет критерий для 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 (например, столбец, который хранит числовые значения). Выражение в функциях MIN, MAX и COUNT может представлять числовое или строковое значение или дату. Все агрегатные функции за исключением COUNT (\*) игнорируют значения NULL.

С помощью оператора AVG вычислим среднее значение величины mileage. Результата показан на рисунке 10.

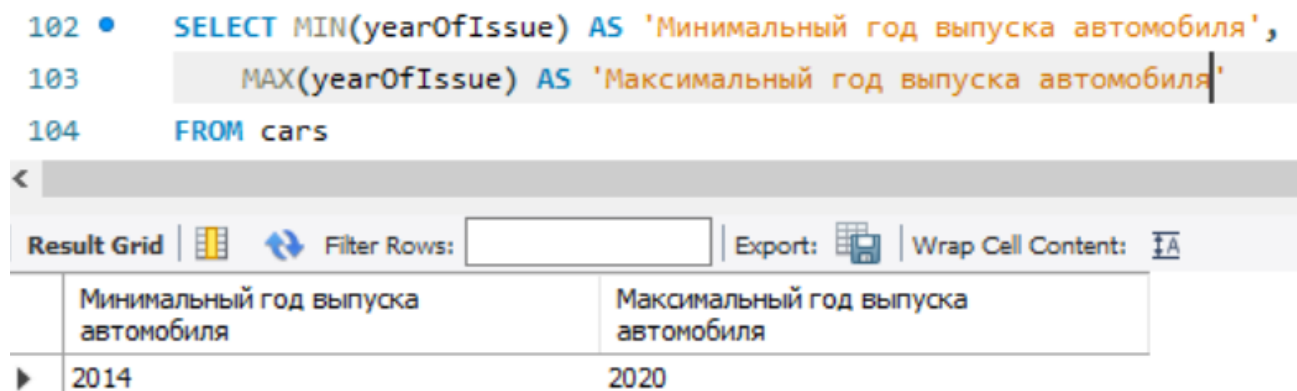


```
102 • SELECT AVG(mileage) AS 'Средний пробег' FROM cars
```

Средний пробег
161185.800000

Рисунок 10 – Пример работы оператора AVG

С помощью операторов MIN и MAX вычислим максимальное и минимальное значение величины year\_of\_issue. Результата показан на рисунке 11.

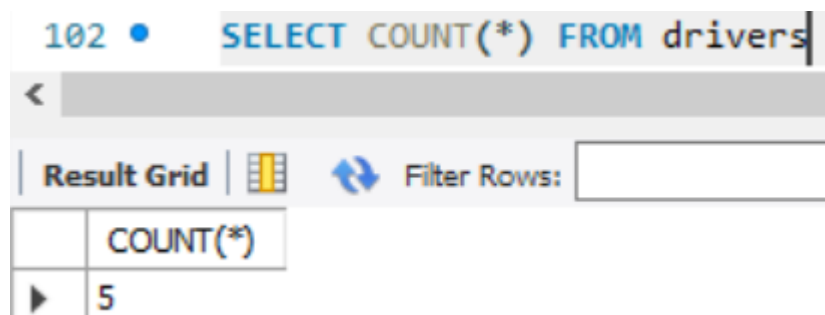


```
102 • SELECT MIN(yearOfIssue) AS 'Минимальный год выпуска автомобиля',  
103 MAX(yearOfIssue) AS 'Максимальный год выпуска автомобиля',  
104 FROM cars
```

Минимальный год выпуска автомобиля	Максимальный год выпуска автомобиля
2014	2020

Рисунок 11 – Пример работы операторов MIN и MAX

Подсчитаем общее количество кортежей в таблице drivers с помощью оператора COUNT. Результат показан на рисунке 12.

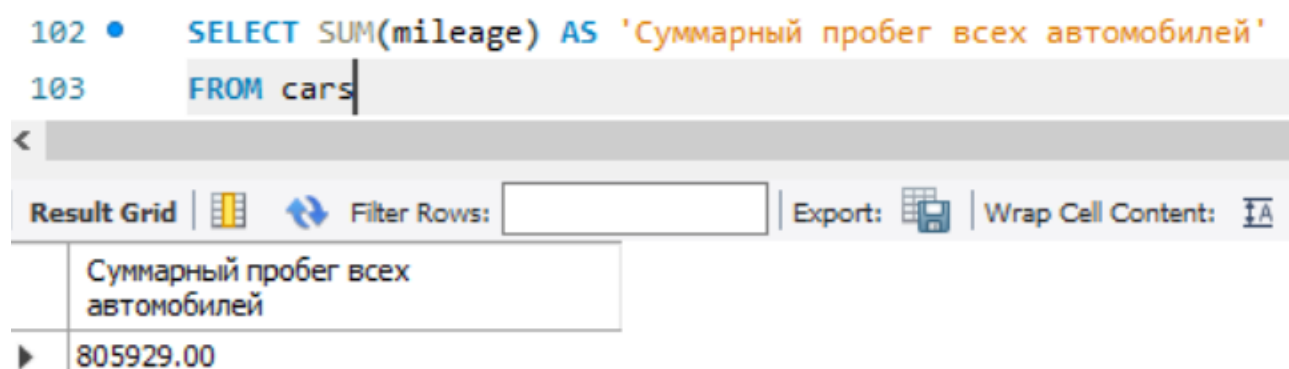


The screenshot shows a query editor with the SQL statement `SELECT COUNT(*) FROM drivers`. Below the editor, the 'Result Grid' tab is active, displaying a single row with the column header `COUNT(*)` and the value `5`. The interface includes a 'Filter Rows' input field and a 'Result Grid' icon.

COUNT(*)
5

Рисунок 12 – Пример работы оператора COUNT

Подсчитаем сумму значений величины mileage с помощью оператора SUM. Результат показан на рисунке 13.



The screenshot shows a query editor with the SQL statement `SELECT SUM(mileage) AS 'Суммарный пробег всех автомобилей' FROM cars`. Below the editor, the 'Result Grid' tab is active, displaying a single row with the column header `Суммарный пробег всех автомобилей` and the value `805929.00`. The interface includes a 'Filter Rows' input field, an 'Export' button, and a 'Wrap Cell Content' checkbox.

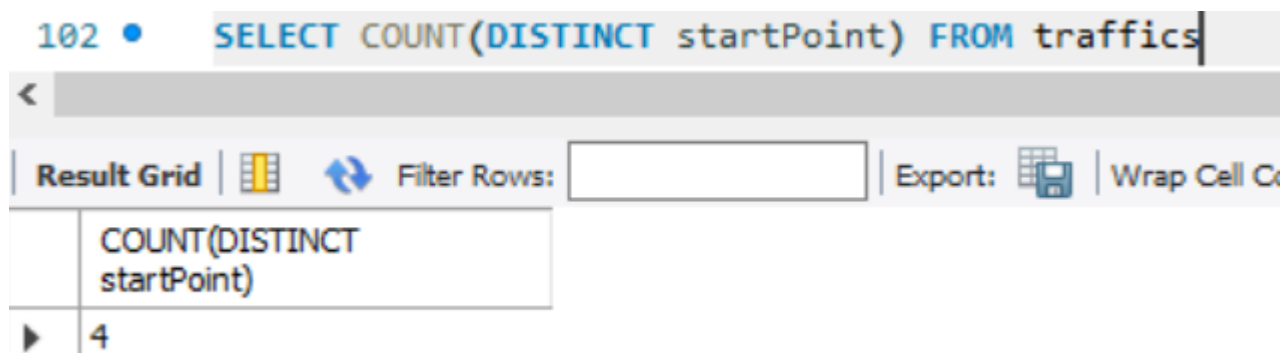
Суммарный пробег всех автомобилей
805929.00

Рисунок 13 – Пример работы оператора SUM

По умолчанию все вышеперечисленных пять функций учитывают все строки выборки для вычисления результата. Но выборка может содержать повторяющиеся значения. Если необходимо выполнить вычисления только над уникальными значениями, исключив из набора значений повторяющиеся данные, то для этого применяется оператор DISTINCT.



Подсчитаем количество уникальных значений начальной точки отправления startPoint. Результат показан на рисунке 14.



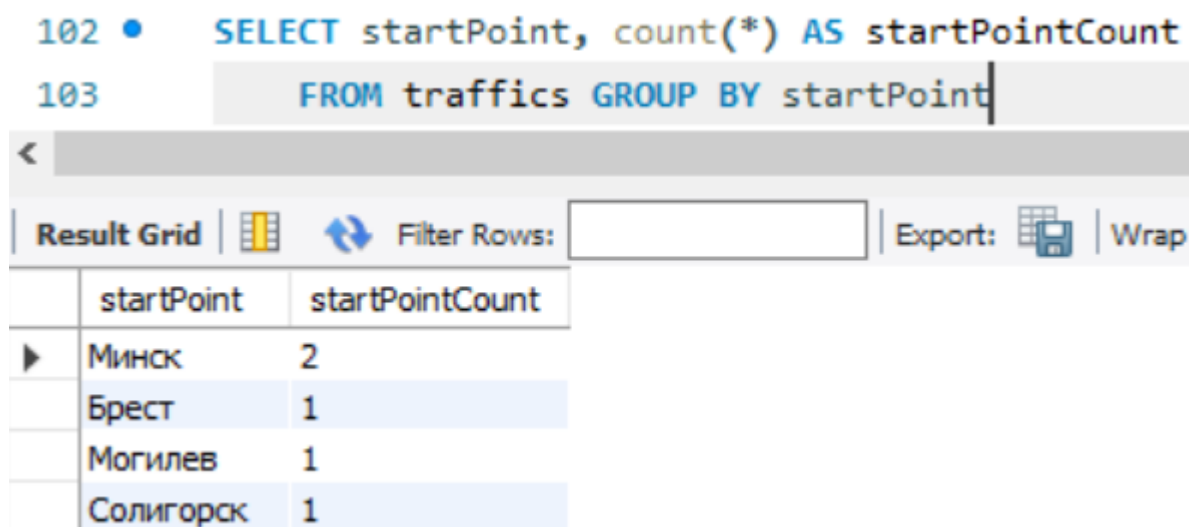
The screenshot shows a SQL query editor with the following SQL query: `SELECT COUNT(DISTINCT startPoint) FROM traffics`. Below the query, there is a toolbar with options like 'Result Grid', 'Filter Rows', 'Export', and 'Wrap Cell C'. The result is displayed in a table with one column and one row.

	COUNT(DISTINCT startPoint)
▶	4

Рисунок 14 – Пример работы оператора DISTINCT

Операторы GROUP BY и HAVING позволяют сгруппировать данные.

Они употребляются в рамках команды SELECT. Оператор GROUP BY определяет, как строки будут группироваться. Например, сгруппируем маршруты по точке отправления. Первый столбец в выражении SELECT – startPoint представляет название группы, а второй столбец – startPointCount представляет результат функции Count, которая вычисляет количество строк в группе. Результат показан на рисунке 15.



The screenshot shows a SQL query editor with the following SQL query: `SELECT startPoint, count(*) AS startPointCount FROM traffics GROUP BY startPoint`. Below the query, there is a toolbar with options like 'Result Grid', 'Filter Rows', 'Export', and 'Wrap'. The result is displayed in a table with two columns: 'startPoint' and 'startPointCount'.

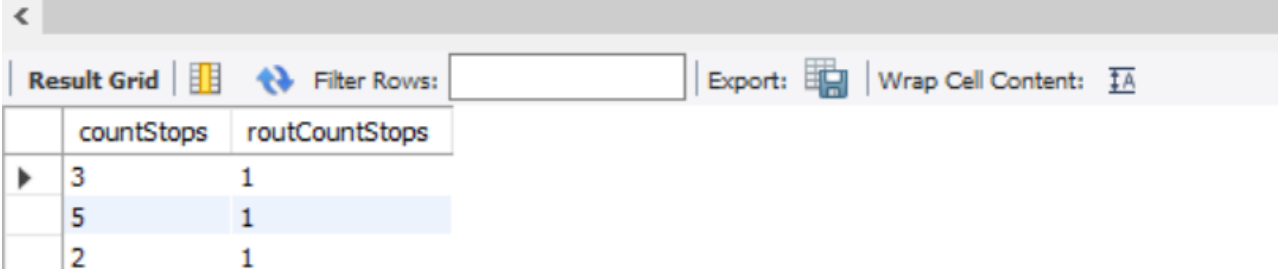
	startPoint	startPointCount
▶	Минск	2
	Брест	1
	Могилев	1
	Солигорск	1

Рисунок 15 – Пример работы оператора GROUP BY

Оператор HAVING позволяет выполнить фильтрацию групп, то есть определяет, какие группы будут включены в выходной результат. Использование HAVING во многом аналогично применению WHERE. Только есть WHERE применяется для фильтрации строк, то HAVING - для фильтрации групп.

Например, найдем все маршруты, для которых одинаковое количество остановок меньше 2. Результат показан на рисунке 16.

```
102 • SELECT countStops, COUNT(*) AS routCountStops
103     FROM traffics GROUP BY countStops HAVING routCountStops < 2
```

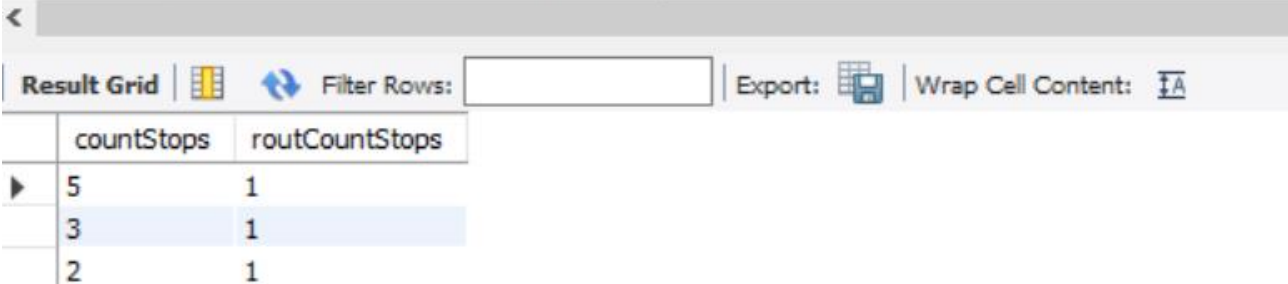


	countStops	routCountStops
▶	3	1
	5	1
	2	1

Рисунок 16 – Пример работы операторов GROUP BY и HAVING

Для этого же подсчета выведем данные по убыванию с помощью операторов ORDER BY и DESC. Результат показан на рисунке 17.

```
102 • SELECT countStops, COUNT(*) AS routCountStops
103     FROM traffics GROUP BY countStops HAVING routCountStops < 2
104     ORDER BY countStops DESC
```



	countStops	routCountStops
▶	5	1
	3	1
	2	1

Рисунок 17 – Пример работы операторов GROUP BY, HAVING, ORDER BY

Таким образом, исходя полученных результатов, мы ознакомились с основными операторами и агрегатными функциями, с сортировкой и группировкой в языке SQL в среде разработки MySQL Workbench.