

# Язык Structured Query Language (SQL)

Артамонов Ю.Н.

# Краткая характеристика языка SQL

Structured Query Language структурированный язык запросов – это универсальный язык высокого уровня для создания, модификации и управления данными в реляционных БД. SQL один из нескольких языков, который «вырос» из идеи использования реляционной модели данных для организации СУБД. В настоящее время SQL практически полностью доминирует в мире реляционных БД. Производители реляционных СУБД, первоначально использующие другие языки, сегодня переориентировались на SQL.

SQL впервые был выпущен на рынок компанией Oracle в 1979 году. В 1981 году IBM объявила о своем первом, основанном на SQL, программном продукте, SQL/DS. О создании собственных реляционных СУБД заявили компании Relational Technology и ряд других. К 1989 году насчитывалось более 25 различных SQL подобных СУБД, работающих на различных типах универсальных ЭВМ и ПЭВМ. SQL до 1986 года не имел официального стандарта. В 1986 году стандарт SQL был совместно опубликован Американским институтом стандартов (ANSI American National Standards Institute) и Международной организацией по стандартизации (ISO International Standards Organization). В 1989 и 1992 годах стандарт языка SQL пересматривался. В настоящее время действует стандарт SQL 92, который в основном представляет собой расширенное множество спецификаций стандарта SQL 89.

# Краткая характеристика языка SQL

SQL относятся к непроцедурным языкам высокого уровня. Так, на процедурном языке javascript нужно записать шаг за шагом все инструкции, необходимые для выполнения задания. В отличие от языка javascript, SQL просто декларирует, что нужно получить, а исполнение (т.е. как получить) возлагается на СУБД. При этом СУБД рассматривается как «черный ящик», и что происходит внутри него – пользователя БД не должно беспокоить. Его должно интересовать только получение правильного ответа на запрос к БД. Ограничивая пользователя в вопросах исполнения операторов, SQL позволяет упростить сами операции и обеспечивает гибкость при реализации баз данных и внутренней оптимизации операций.

Запрос к БД на SQL состоит из набора предложений (или «секций» запроса) с помощью которых указывается, какие данные необходимо получить и основе каких условий. С помощью единственного запроса на SQL можно соединить несколько таблиц, получив промежуточную (временную) таблицу, а затем вырезать из нее требуемые строки и столбцы (селекция и проекция).

# Функциональные категории команд SQL

Принято выделять функциональные команды SQL:

- Data Manipulation Language (DML), язык манипулирования данными - это набор команд, которые выбирают или задают данные в таблицах. Здесь различают запросы на выборку данных - SELECT, запросы на модификацию данных - INSERT, UPDATE, DELETE.
- Data Definition Language (DDL), язык описания данных - состоит из команд, которые создают объекты (таблицы, индексы, представления, и так далее) в базе данных.
- Data Control Language (DCL), язык управления данными - состоит из средств, которые определяют права (привилегии) пользователя для выполнения определенных действий в БД. Включает две основные команды: GRANT — применяется для присвоения привилегии; REVOKE — применяется для отмены привилегии.

Важно понимать, что это не различные языки, а команды SQL, сгруппированные по их функциям.

# Описание учебной базы

Для демонстрации работы различных команд языка SQL рассмотрим учебную базу данных class, состоящую из трех связанных таблиц.

Работники школы:

#	Табельный №	ФИО учителя	Должность/профессия	Пол	Оклад
1	100	Петров Станислав Васильевич	Учитель	М	50000
2	101	Петрова Валентина Григорьевна	Учитель	Ж	55000
3	102	Рыбакова Анна Ивановна	Завуч	Ж	90000
4	103	Федоров Юрий Васильевич	Директор	М	200000
5	104	Смирнов Антон Юрьевич	Учитель	М	70000

Классы:

№ класса	Табельный № учителя	Специализация класса	Классная комната
10А	101	Иностранные языки(английский ...	203
11Б	100	Математика и физика	212

Школьники:

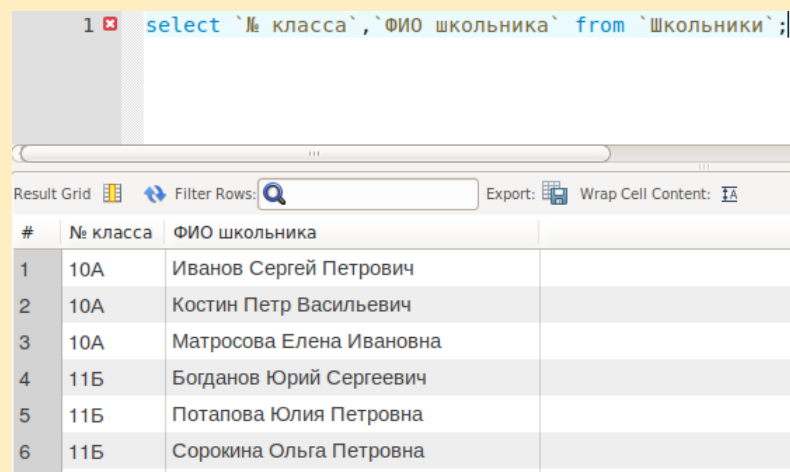
№ класса	Порядковый № в журнале	ФИО школьника	Пол
10А	1	Иванов Сергей Петрович	М
10А	2	Костин Петр Васильевич	М
10А	3	Матросова Елена Ивановна	Ж
11Б	1	Богданов Юрий Сергеевич	М
11Б	2	Потапова Юлия Петровна	Ж
11Б	3	Сорокина Ольга Петровна	Ж
11Б	4	Сидоров Андрей Петрович	М

# Команда SELECT

Запрос SELECT - наиболее часто используемая в SQL команда, которая позволяет вывести определенную информацию из таблиц в память. Эта информация обычно посылается непосредственно экран компьютера. Структура этой команды обманчиво проста, потому что нужно расширять ее, чтобы выполнить сложные оценки и обработки данных. Упрощенный синтаксис оператора SELECT имеет следующий вид:

SELECT <список атрибутов «производной (выходной)» таблицы» FROM <список таблиц, участвующих в запросе> WHERE <условия поиска и/или условия соединения таблиц (если в запросе участвует несколько таблиц)>;

В самой простой форме команда SELECT извлекает информацию из таблицы. Например, можно написать запрос, который выводит отдельные столбцы из таблицы Школьников, т.е. выбранные столбцы (атрибуты) и все строки (кортежи), в следующем виде:



The screenshot shows a database query window with a SQL editor at the top containing the query: `select `№ класса`, `ФИО школьника` from `Школьники`;`. Below the editor is a toolbar with icons for 'Result Grid', 'Filter Rows', 'Export', and 'Wrap Cell Content'. The 'Result Grid' is active, displaying a table with the following data:

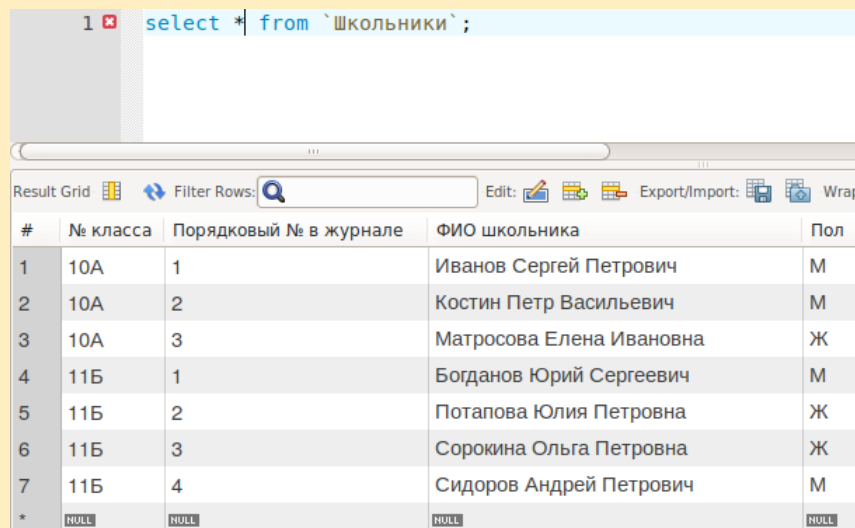
#	№ класса	ФИО школьника
1	10А	Иванов Сергей Петрович
2	10А	Костин Петр Васильевич
3	10А	Матросова Елена Ивановна
4	11Б	Богданов Юрий Сергеевич
5	11Б	Потапова Юлия Петровна
6	11Б	Сорокина Ольга Петровна

# Команда SELECT

В предложении FROM запроса указываются имена таблиц, используемых в качестве источника информации. В данном примере это имя одной таблицы Школьники.

Точка с запятой используется во всех интерактивных (т.е. выполняемых в диалоговом режиме) командах SQL, чтобы сообщать серверу БД, что запись команды завершена и она готова к выполнению.

Если в запросе нужно выбрать все атрибуты таблицы, то можно использовать сокращение «звездочка (\*)». Например, в этом случае предыдущий запрос можно было бы переписать в следующем виде:

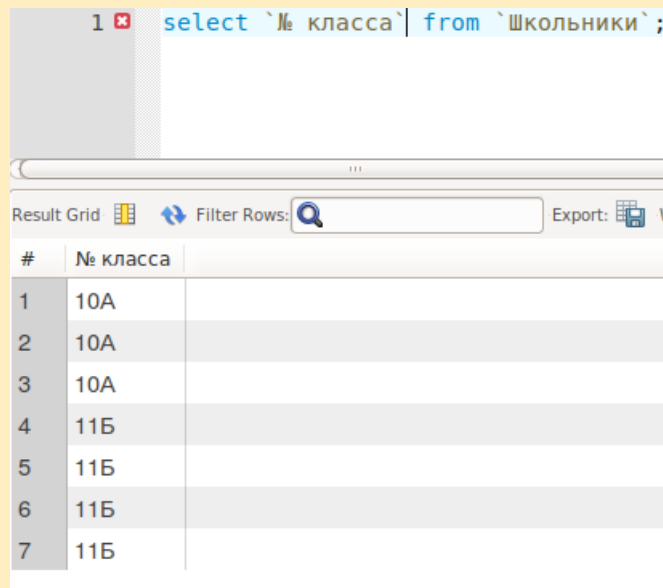


```
1 select * from `Школьники` ;
```

#	№ класса	Порядковый № в журнале	ФИО школьника	Пол
1	10А	1	Иванов Сергей Петрович	М
2	10А	2	Костин Петр Васильевич	М
3	10А	3	Матросова Елена Ивановна	Ж
4	11Б	1	Богданов Юрий Сергеевич	М
5	11Б	2	Потапова Юлия Петровна	Ж
6	11Б	3	Сорокина Ольга Петровна	Ж
7	11Б	4	Сидоров Андрей Петрович	М
*	NULL	NULL	NULL	NULL

# Удаление повторяющихся значений

Особенностью SQL запросов, отличающих их от операций реляционной алгебры (а именно от операции проецирования), является необходимость явного указания, что следует удалять повторяющиеся строки таблицы-результаты. Для этого используется ключевое слово **DISTINCT**. Это аргумент, который ставится сразу после ключевого слова **SELECT** и позволяет устранять повторяющиеся значения строк из результата запроса. Например, требуется получить все различные классы, в которых учатся школьники. Однако выполнив обычный запрос, мы получим повторения записей (фактически от каждого школьника):



```
1 select `№ класса` from `Школьники`;
```

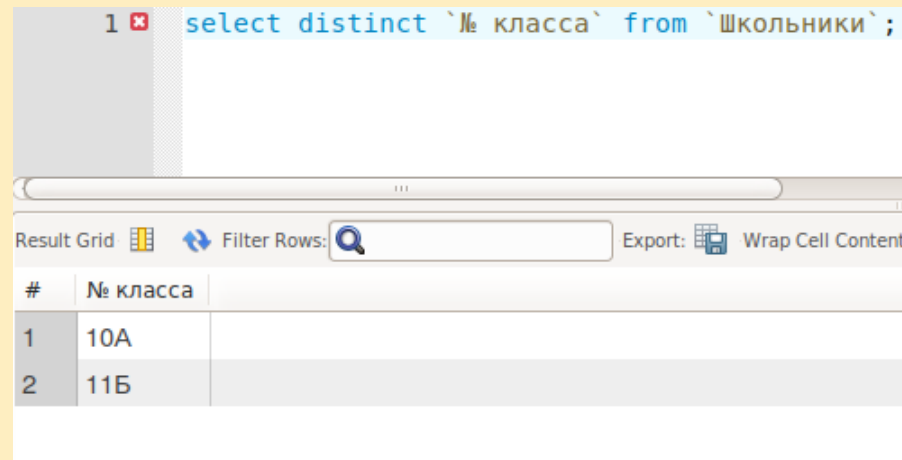
Result Grid

#	№ класса
1	10A
2	10A
3	10A
4	11Б
5	11Б
6	11Б
7	11Б



# Удаление повторяющихся значений

Выполним этот же запрос, но с использованием DISTINCT, получим требуемый результат:



The screenshot shows a SQL query editor with a single query in a text area. Below the text area is a horizontal scrollbar. Underneath the scrollbar is a toolbar with icons for 'Result Grid', 'Filter Rows', 'Export', and 'Wrap Cell Content'. Below the toolbar is a table with two columns: '#' and '№ класса'. The table contains two rows of data.

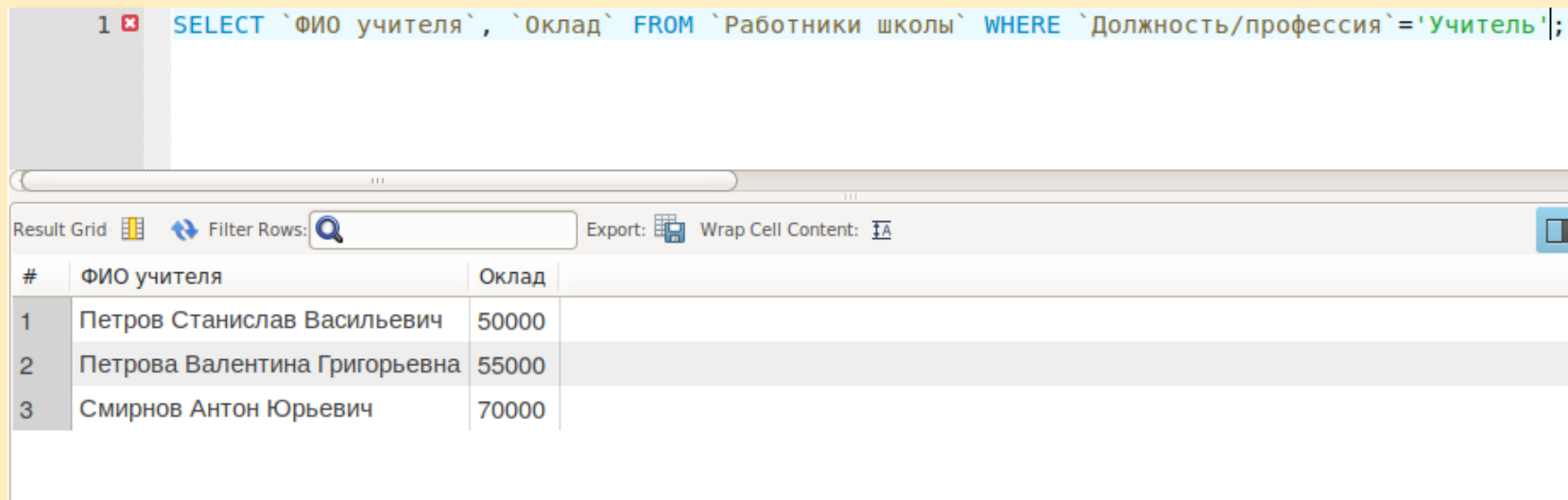
```
1 x select distinct `№ класса` from `Школьники`;
```

#	№ класса
1	10A
2	11Б

# Задание условий в запросах

Ключевое слово **WHERE** задает предложение команды **SELECT**, которое позволяет устанавливать ограничение на выбираемые строки. Для этого в запросе используются предикаты (условия) - команда **SELECT** извлекает только те строки из таблицы, для которых утверждение предиката верно.

Запрос: вывести имена и оклады всех работников школы в должности учителя.



The screenshot shows a database query interface. At the top, a SQL query is entered in a text box: `1 SELECT `ФИО учителя`, `Оклад` FROM `Работники школы` WHERE `Должность/профессия`='Учитель';`. Below the query box, there is a toolbar with options like 'Result Grid', 'Filter Rows', 'Export', and 'Wrap Cell Content'. The main area displays a table with the results of the query.

#	ФИО учителя	Оклад
1	Петров Станислав Васильевич	50000
2	Петрова Валентина Григорьевна	55000
3	Смирнов Антон Юрьевич	70000

Когда предложение **WHERE** присутствует в запросе, сервер БД просматривает всю таблицу и проверяет для каждой строки условие предиката.

- 1 Получить список всех работников школы мужского пола.
- 2 Получить список работников школы, у которых должностной оклад равен 50 тыс.руб.
- 3 Получить должности всех работников школы без повторений.

# Операторы сравнения

Операторы сравнения указывают на тип сравнения между двумя значениями в предикате. Операторы сравнения SQL приведены в таблице:

Символ	Отношение
=	равно
>	больше
<	меньше
>=	больше либо равно
<=	меньше либо равно
<>	не равно

Эти операторы имеют стандартные значения для численных значений. Для строковых значений их результат зависит от кодировки строки: ASCII или UTF-8. SQL сравнивает символьные значения в лексикографическом порядке кодов символов как это определено в соответствующей кодировке.

# Булевы операторы

Основные булевские операторы поддерживаемые в SQL выражениях - это AND, OR, и NOT: логическое «И», «ИЛИ» «Отрицание». Другие, более сложные, логические операторы (например, "исключающий или") могут быть сформированы из этих трех основных операторов.

# Примеры использования операторов сравнения и булевых операторов

Получить список работников школы мужского пола в должности учителя:

```
1 SELECT * FROM `Работники школы` WHERE `Должность/профессия`='Учитель' AND `Пол`='М';
```

#	Табельный №	ФИО учителя	Должность/профессия	Пол	Оклад	
1	100	Петров Станислав Васильевич	Учитель	М	50000	
2	104	Смирнов Антон Юрьевич	Учитель	М	70000	
*	NULL	NULL	NULL	NULL	NULL	

Получить список всех сотрудников школы, кроме директора школы:

```
1 SELECT * FROM `Работники школы` WHERE NOT(`Должность/профессия`='Директор');
```

#	Табельный №	ФИО учителя	Должность/профессия	Пол	Оклад	
1	100	Петров Станислав Васильевич	Учитель	М	50000	
2	101	Петрова Валентина Григорьевна	Учитель	Ж	55000	
3	102	Рыбакова Анна Ивановна	Завуч	Ж	90000	
4	104	Смирнов Антон Юрьевич	Учитель	М	70000	

# Примеры использования операторов сравнения и булевых операторов

Получить список всех сотрудников школы, кроме директора школы (другое решение):

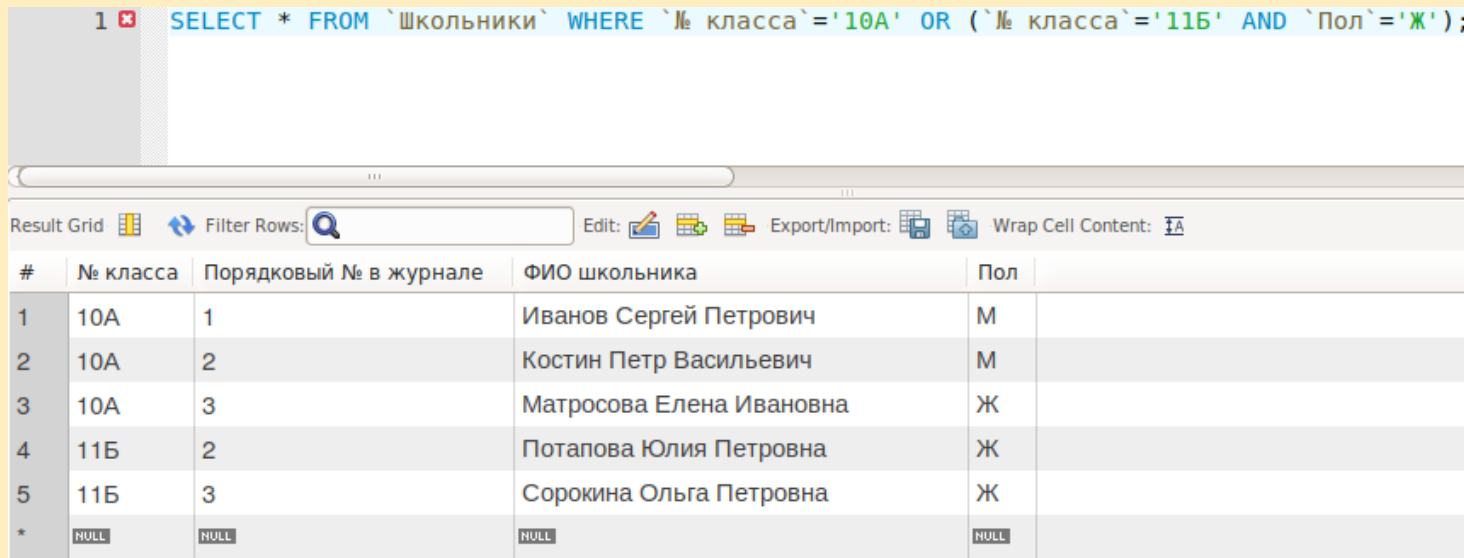
```
1 SELECT * FROM `Работники школы` WHERE `Должность/профессия` <> 'Директор';
```

Result Grid						
Filter Rows: <input type="text"/>						
Edit:     Export/Import:   Wrap Cell Content:						
#	Табельный №	ФИО учителя	Должность/профессия	Пол	Оклад	
1	100	Петров Станислав Васильевич	Учитель	М	50000	
2	101	Петрова Валентина Григорьевна	Учитель	Ж	55000	
3	102	Рыбакова Анна Ивановна	Завуч	Ж	90000	
4	104	Смирнов Антон Юрьевич	Учитель	М	70000	
*	NULL	NULL	NULL	NULL	NULL	

# Примеры использования операторов сравнения и булевых операторов

Получить список всех школьников из 10А класса, или всех девочек из 11Б класса:

```
1 x SELECT * FROM `Школьники` WHERE `№ класса`='10А' OR (`№ класса`='11Б' AND `Пол`='Ж');
```



#	№ класса	Порядковый № в журнале	ФИО школьника	Пол
1	10А	1	Иванов Сергей Петрович	М
2	10А	2	Костин Петр Васильевич	М
3	10А	3	Матросова Елена Ивановна	Ж
4	11Б	2	Потапова Юлия Петровна	Ж
5	11Б	3	Сорокина Ольга Петровна	Ж
*	NULL	NULL	NULL	NULL



# Примеры использования операторов сравнения и булевых операторов

Получить список школьников, у которых четный номер в журнале:

```
1 x SELECT * FROM `Школьники` WHERE (`Порядковый № в журнале`%2)=0;
```

Result Grid Filter Rows: Edit: Export/Import: Wrap Cell Content:

#	№ класса	Порядковый № в журнале	ФИО школьника	Пол
1	10А	2	Костин Петр Васильевич	М
2	11Б	2	Потапова Юлия Петровна	Ж
3	11Б	4	Сидоров Андрей Петрович	М
*	NULL	NULL	NULL	NULL

- 1 Получить список всех работников школы, у которых должностной оклад меньше 60 тыс. руб..
- 2 Получить список сотрудников и их окладов при увеличении окладов на 10%

# Оператор IN

Оператор IN определяет набор (список) значений, в который должно быть включено значение атрибута.

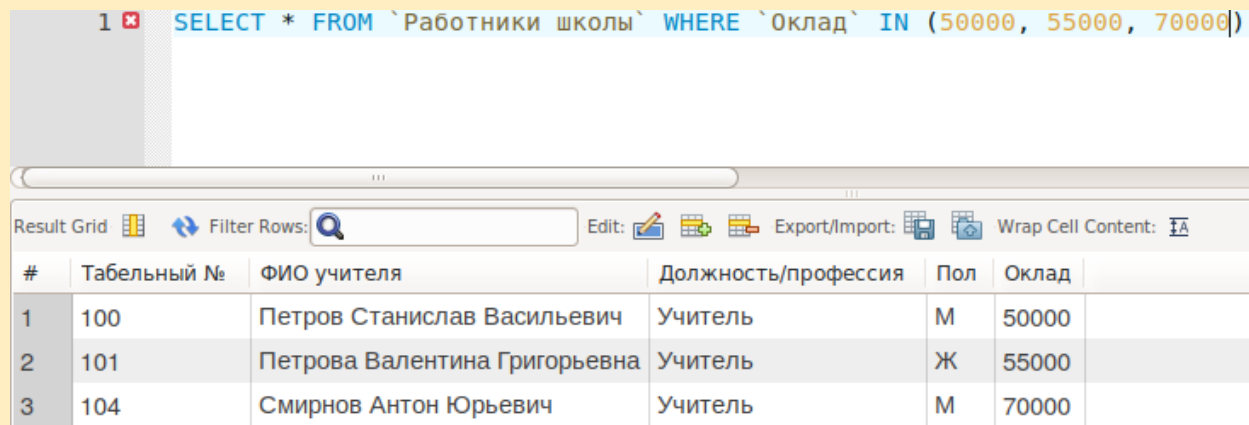
В принципе без оператора IN можно обойтись, используя логическое OR:

```
SELECT * FROM 'Работники школы' WHERE 'Должность/профессия'='Учитель'  
OR 'Должность/профессия' = 'Директор';
```

Однако оператор IN позволяет упростить запрос:

```
SELECT * FROM 'Работники школы' WHERE 'Должность/профессия' IN ('Учи-  
тель', 'Директор');
```

Оператор IN определяет набор значений с помощью списка имен, заключенных в круглые скобки и отделенных запятыми. Когда список состоит из числовых значений, а не строк, то одиночные кавычки опускаются:

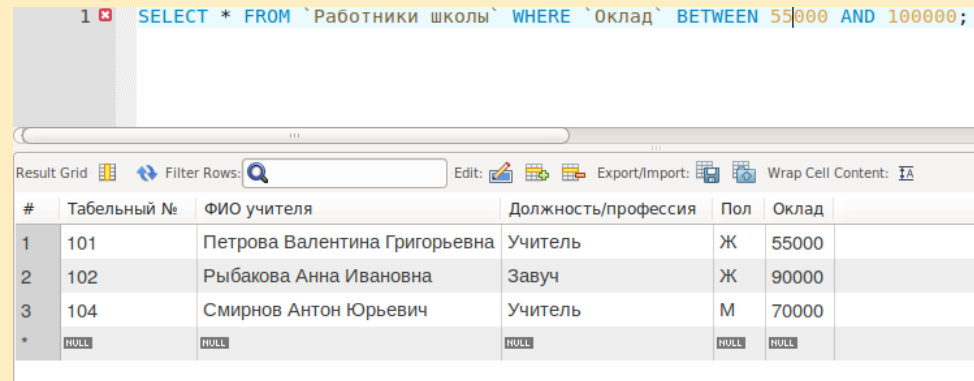


```
1 SELECT * FROM `Работники школы` WHERE `Оклад` IN (50000, 55000, 70000);
```

#	Табельный №	ФИО учителя	Должность/профессия	Пол	Оклад
1	100	Петров Станислав Васильевич	Учитель	М	50000
2	101	Петрова Валентина Григорьевна	Учитель	Ж	55000
3	104	Смирнов Антон Юрьевич	Учитель	М	70000

# Оператор BETWEEN

Оператор BETWEEN похож на оператор IN. Однако, в отличие от определения списка (как это делает IN), оператор BETWEEN определяет диапазон значений. Вводится ключевое слово BETWEEN с начальным Значением, ключевое слово-разделитель AND и затем конечное значение. BETWEEN «чувствителен» к порядку, т.е. первое значение в предложении должно быть меньше второго.

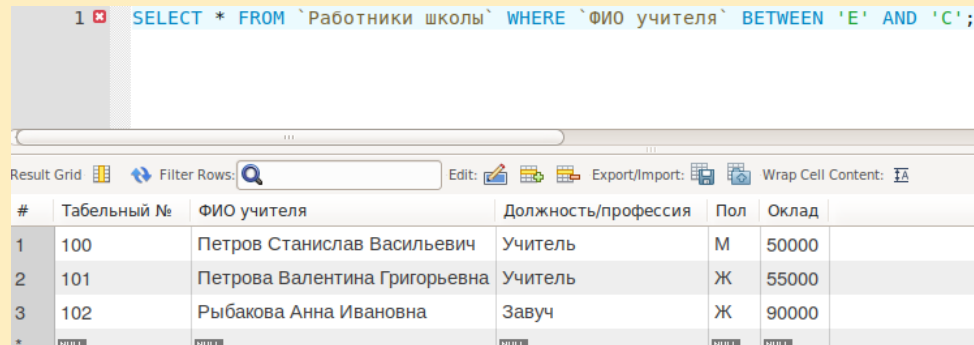


```
1 SELECT * FROM `Работники школы` WHERE `Оклад` BETWEEN 55000 AND 100000;
```

Result Grid

#	Табельный №	ФИО учителя	Должность/профессия	Пол	Оклад
1	101	Петрова Валентина Григорьевна	Учитель	Ж	55000
2	102	Рыбакова Анна Ивановна	Завуч	Ж	90000
3	104	Смирнов Антон Юрьевич	Учитель	М	70000
*	NULL	NULL	NULL	NULL	NULL

Оператор BETWEEN может работать с символьными полями в терминах ASCII, т.е. можно использовать BETWEEN, чтобы выбирать ряд значений из упорядоченных по алфавиту значений:



```
1 SELECT * FROM `Работники школы` WHERE `ФИО учителя` BETWEEN 'Е' AND 'С';
```

Result Grid

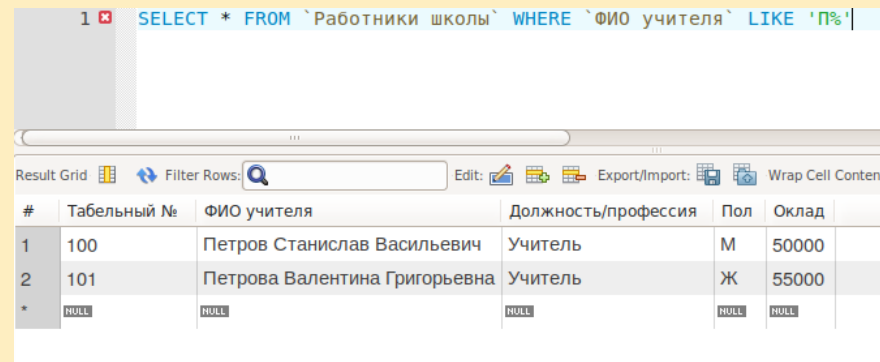
#	Табельный №	ФИО учителя	Должность/профессия	Пол	Оклад
1	100	Петров Станислав Васильевич	Учитель	М	50000
2	101	Петрова Валентина Григорьевна	Учитель	Ж	55000
3	102	Рыбакова Анна Ивановна	Завуч	Ж	90000
*	NULL	NULL	NULL	NULL	NULL

# Оператор LIKE

Оператор LIKE применим только к полям типа CHAR или VARCHAR, с которыми он используется, чтобы находить подстроки. При задании условий используются «групповые символы» (wildcards). Имеются два типа групповых символов используемых с оператором LIKE:

1. Символ подчеркивания («`_`») - замещает любой одиночный символ. Например, "b\_t будет соответствовать словам "bat' или 'bit', но не будет соответствовать 'brat'.
2. Знак процента («`%`») - замещает последовательность любого (в числе нулевого) числа символов. Например, %p%t будет соответствовать словам 'put', 'posit, или 'opt', но не 'spite'.

Отобразить всех сотрудников школы, фамилия которых начинается с буквы «П»:

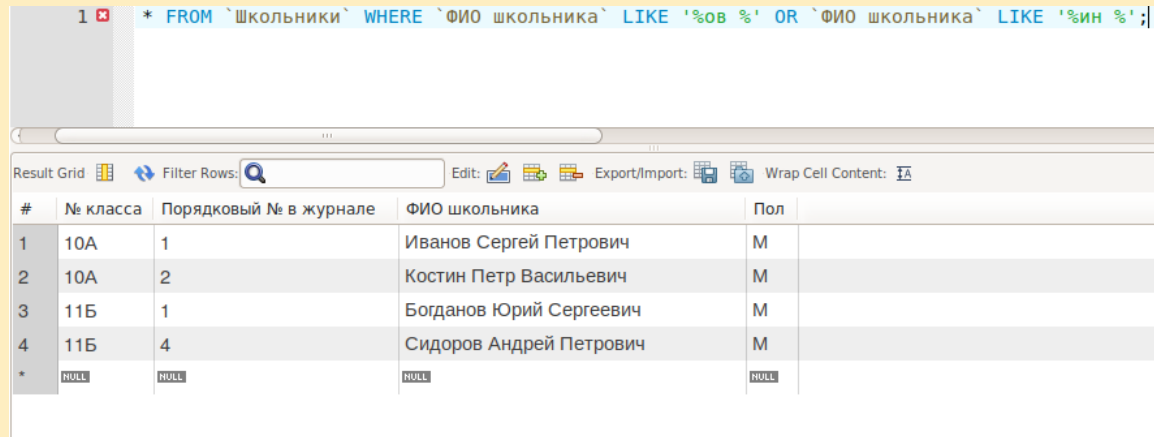


The screenshot shows a database query window with the following SQL query: `SELECT * FROM `Работники школы` WHERE `ФИО учителя` LIKE 'П%'``. Below the query, there is a toolbar with icons for 'Result Grid', 'Filter Rows', 'Edit', 'Export/Import', and 'Wrap Cell Content'. The results are displayed in a table with the following data:

#	Табельный №	ФИО учителя	Должность/профессия	Пол	Оклад
1	100	Петров Станислав Васильевич	Учитель	М	50000
2	101	Петрова Валентина Григорьевна	Учитель	Ж	55000
*	NULL	NULL	NULL	NULL	NULL

# Оператор LIKE

Отобразить всех школьников, фамилия которых заканчивается на «ов» или «ин»:

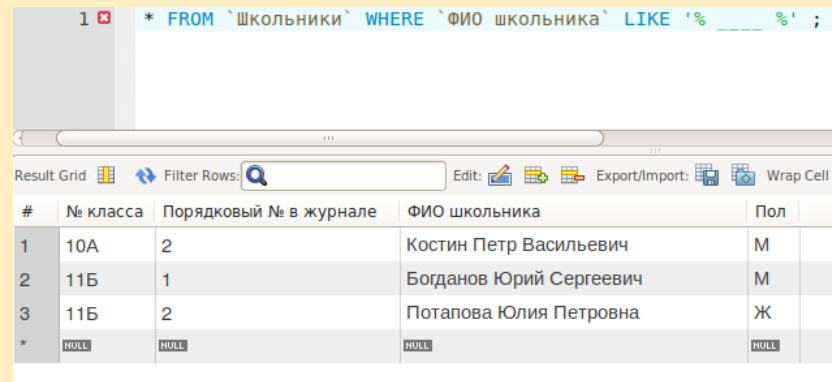


The screenshot shows a database query editor with a SQL query in the top pane and a result grid in the bottom pane. The query is: `* FROM `Школьники` WHERE `ФИО школьника` LIKE '%ов %' OR `ФИО школьника` LIKE '%ин %';`

The result grid displays the following data:

#	№ класса	Порядковый № в журнале	ФИО школьника	Пол
1	10A	1	Иванов Сергей Петрович	М
2	10A	2	Костин Петр Васильевич	М
3	11Б	1	Богданов Юрий Сергеевич	М
4	11Б	4	Сидоров Андрей Петрович	М
*	NULL	NULL	NULL	NULL

Отобразить всех школьников, имя которых состоит из 4 букв:



The screenshot shows a database query editor with a SQL query in the top pane and a result grid in the bottom pane. The query is: `* FROM `Школьники` WHERE `ФИО школьника` LIKE '% ____ %';`

The result grid displays the following data:

#	№ класса	Порядковый № в журнале	ФИО школьника	Пол
1	10A	2	Костин Петр Васильевич	М
2	11Б	1	Богданов Юрий Сергеевич	М
3	11Б	2	Потапова Юлия Петровна	Ж
*	NULL	NULL	NULL	NULL

# Работа с NULL значениями

В таблице могут храниться записи, которые не имеют значений для некоторого атрибута, например, потому, информация отсутствует, или это поле просто не заполнялось. SQL учитывает такой вариант, позволяя вводить значение NULL (пустой) в поле, вместо конкретного значения. Когда значение поля равно NULL, это означает, что СУБД специально промаркировала это поле как не Имеющее значения для этой строки. Это отличается от назначения полю «нуля» или пробела. NULL не имеет типа данных, поэтому оно может помещаться в атрибут любого типа. Тем ни менее, NULL в SQL часто упоминается как ноль.

Так как NULL указывает на отсутствие значения, вы не можете знать, каков будет результат сравнения с использованием NULL. Когда NULL сравнивается со значением, даже с другим таким же NULL, результат будет «неизвестен».

Поэтому выражение типа 'Фамилия школьника' = NULL или 'Фамилия школьника' IN (NULL) будет неизвестно, независимо от значения 'Фамилия школьника'. Для работы с NULL SQL имеет специальный оператор IS, который используется с ключевым словом NULL, для записи условий на значения NULL.

Запрос: Найти всех сотрудников школы, у которых не указан оклад:

```
SELECT * FROM 'Работники школы' WHERE 'Оклад' IS NULL;
```

Запрос: Найти всех сотрудников школы, у которых наоборот указан оклад:

```
SELECT * FROM 'Работники школы' WHERE NOT('Оклад' IS NULL);
```

- 1 Напишите запрос, который выводит всех сотрудников школы, табельный номер которых в диапазоне с 100 до 103 с использованием IN и BETWEEN.
- 2 Напишите запрос, который выводит сотрудников школы, у которых отчество начинается на букву «В».



# Агрегатные функции

Запросы могут вычислять обобщенное значение поля - это делается с помощью агрегатных функций:

- COUNT(<поле>) выдает число не-NULL значений поля результирующей таблицы.
- SUM(<поле>) - вычисляет арифметическую сумму всех выбранных значений поля.
- AVG(<поле>) вычисляет усреднение всех выбранных значений данного поля.
- MAX(<поле>) - вычисляет наибольшее из всех выбранных значений данного поля.
- MIN(<поле>) - вычисляет наименьшее из всех выбранных значений данного поля.

Агрегатные функции используются в предложении SELECT запроса, - они используют имя поля как аргумент. С функциями SUM и AVG могут использоваться только числовые поля. С функциями COUNT, MAX и MIN могут использоваться и числовые, и строковые атрибуты.

Функция COUNT считает число значений в данном столбце, или число строк в таблице. Когда она считает значения столбца, ее следует использовать с оператором DISTINCT, чтобы не подсчитывать повторения.

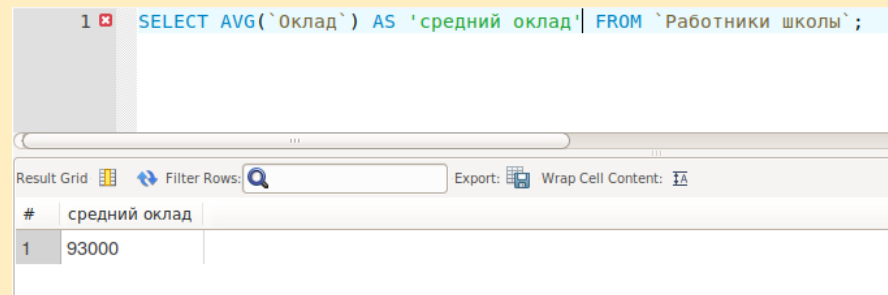
# Агрегатные функции

Чтобы подсчитать общее число строк в таблице, используют функцию COUNT со звездочкой вместо имени поля, как, например, в следующем примере:

```
SELECT COUNT (*) FROM 'Классы';
```

Функция «COUNT со звездочкой» подсчитывает все строки, включая «дубликаты».

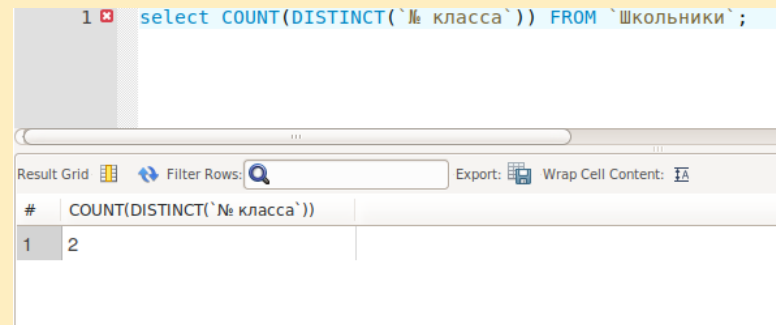
Запрос: Рассчитать средний оклад работников школы:



```
1 x SELECT AVG(`Оклад`) AS 'средний оклад' FROM `Работники школы`;
```

#	средний оклад
1	93000

Запрос: Рассчитать количество различных классов, в которых учатся школьники:

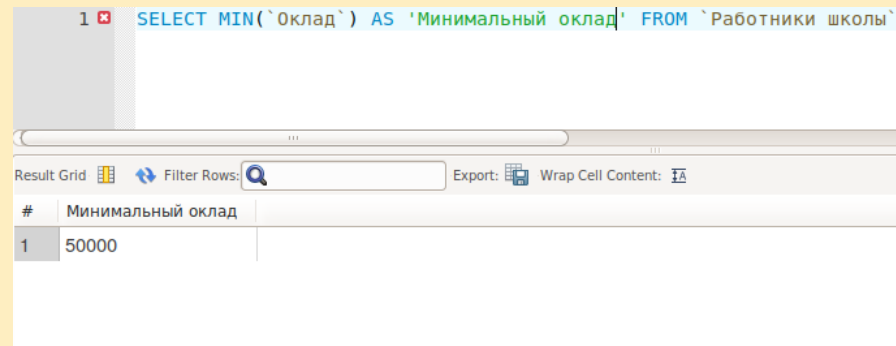


```
1 x select COUNT(DISTINCT(`№ класса`)) FROM `Школьники`;
```

#	COUNT(DISTINCT('№ класса'))
1	2

# Агрегатные функции

Запрос: Рассчитать минимальный оклад работников школы:



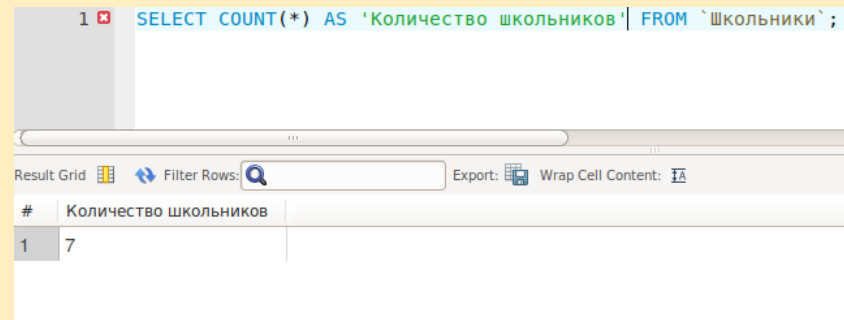
```
1 SELECT MIN(`Оклад`) AS 'Минимальный оклад' FROM `Работники школы`;
```

Result Grid

#	Минимальный оклад
1	50000

Filter Rows: Export: Wrap Cell Content:

Запрос: Рассчитать количество записей в таблице школьников:



```
1 SELECT COUNT(*) AS 'Количество школьников' FROM `Школьники`;
```

Result Grid

#	Количество школьников
1	7

Filter Rows: Export: Wrap Cell Content:

# Предложение GROUP BY

Предложение GROUP BY позволяет определять группу строк в терминах заданного атрибута и применять агрегатные функции к группам. Возможно формировать группы строк и применять к ним агрегатные функции в одном предложении SELECT.

Запрос: Рассчитать средний оклад по каждой должности:

```
1 SELECT `Должность/профессия`,AVG(`Оклад`) AS 'Средний оклад' FROM `Работники школы` GROUP BY `Должность/профессия`;
```

#	Должность/профессия	Средний оклад
1	Директор	200000
2	Завуч	90000
3	Учитель	58333.33333333336

Каждая группа состоит из всех строк с одним и тем же значением поля 'Должность/профессия', и функция AVG применяется отдельно для каждой такой группы. Таким образом, поле, к которому применяется GROUP BY, имеет по определению, константное значение на каждую группу.

# Предложение GROUP BY

Можно использовать GROUP BY с несколькими полями:

Запрос: Рассчитать суммарное количество школьников мальчиков и девочек в разбивке по классам:

```
1 select `№ класса`, `Пол`, COUNT(`Пол`) AS 'Количество' FROM `Школьники` GROUP BY `№ класса`, `Пол`;
```

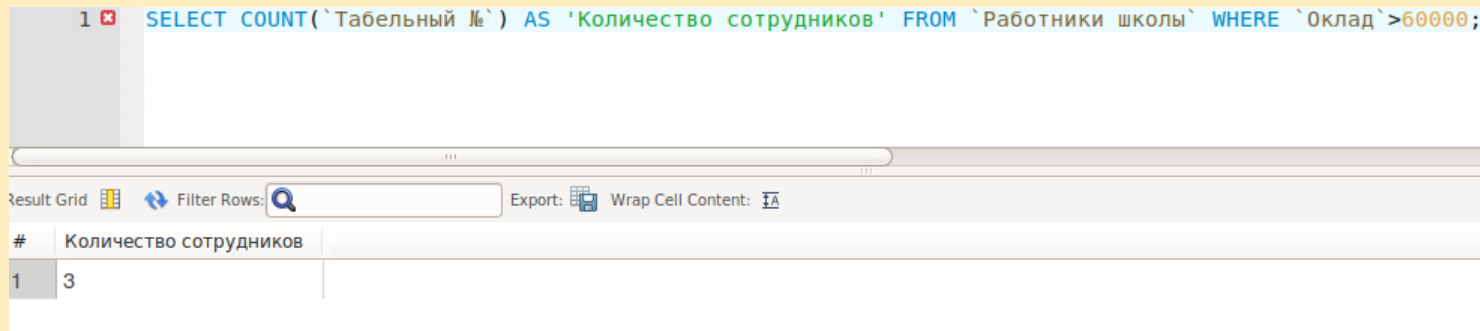
Result Grid Filter Rows: Export: Wrap Cell Content:

#	№ класса	Пол	Количество
1	10A	Ж	1
2	10A	М	2
3	11Б	Ж	2
4	11Б	М	2

# Предложение HAVING

НЕЛЬЗЯ использовать агрегатную функцию в предложении WHERE, поскольку предикаты формируют условия в терминах одной строки, а агрегатные функции относятся к группе строк.

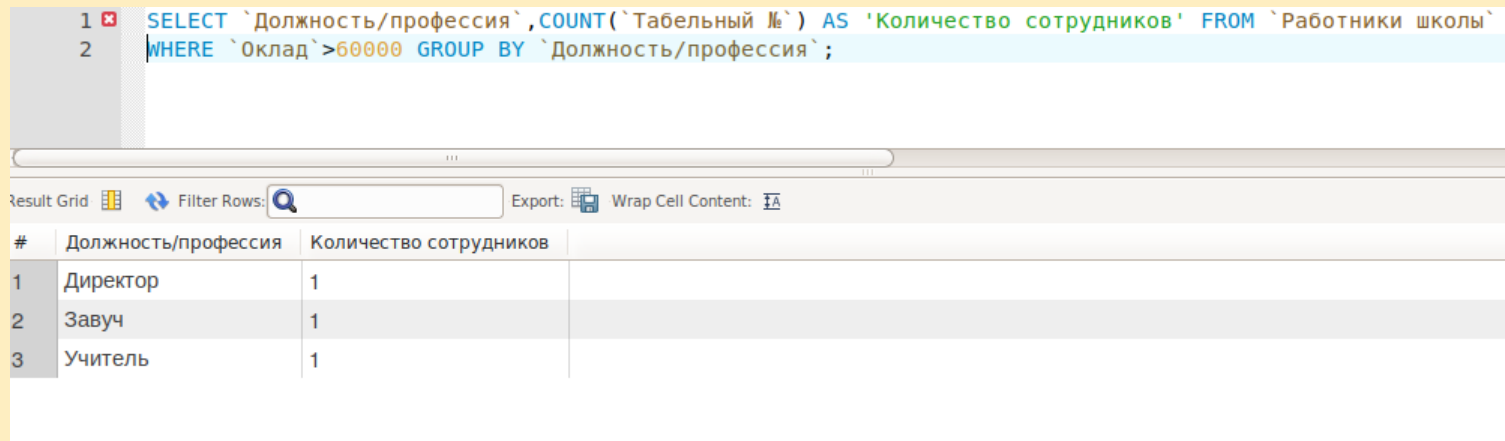
Запрос: Определить суммарное количество сотрудников с окладом больше 60 тыс. руб.



```
1 SELECT COUNT('Табельный №') AS 'Количество сотрудников' FROM 'Работники школы' WHERE 'Оклад' > 60000;
```

#	Количество сотрудников
1	3

Запрос: Определить суммарное количество сотрудников с окладом больше 60 тыс. руб. в разбивке по должности



```
1 SELECT 'Должность/профессия', COUNT('Табельный №') AS 'Количество сотрудников' FROM 'Работники школы'
2 WHERE 'Оклад' > 60000 GROUP BY 'Должность/профессия';
```

#	Должность/профессия	Количество сотрудников
1	Директор	1
2	Завуч	1
3	Учитель	1

# Предложение HAVING

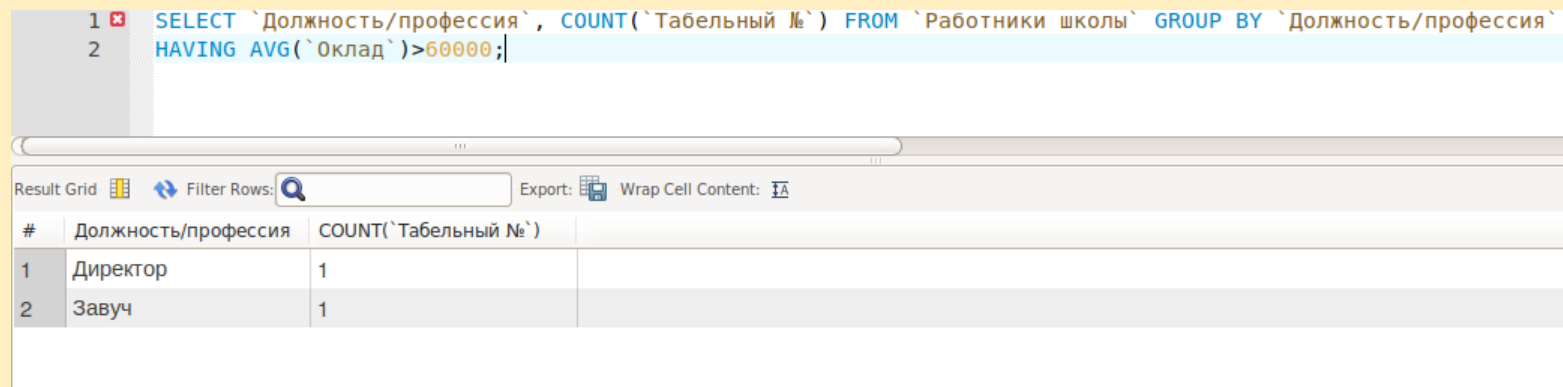
Запрос: Отобразить должности сотрудников, средний оклад которых превышает 60 тыс. руб. Если выполнить что-то такое:

```
SELECT 'Должность/профессия', COUNT('Табельный №') FROM 'Работники школы' WHERE AVG('Оклад')>60000 GROUP BY 'Должность/профессия';
```

То получим ошибку:

```
SELECT Должность/профессия FROM 'Работники школы' WHERE AVG('Оклад')>60000; Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'Работники школы' WHERE AVG('Оклад')>60000' at line 1 0,00038 sec
```

В этом случае нужно использовать предложение HAVING. Предложение HAVING определяет критерии, используемые для задания условий на группы, также как предложение WHERE задает условия для индивидуальных строк:



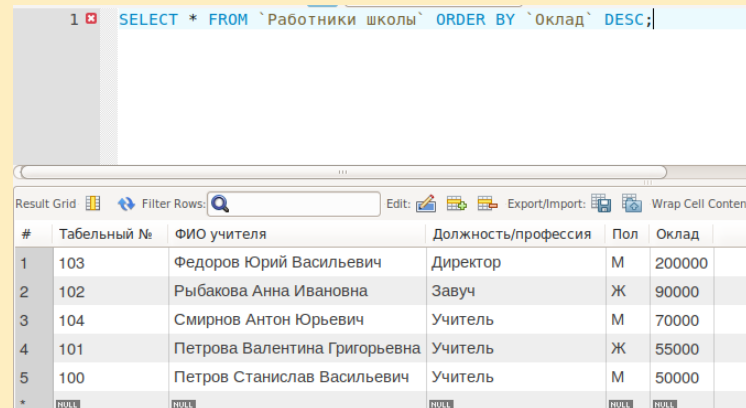
```
1 SELECT `Должность/профессия`, COUNT(`Табельный №`) FROM `Работники школы` GROUP BY `Должность/профессия`
2 HAVING AVG(`Оклад`)>60000;
```

Result Grid				Filter Rows: <input type="text"/>	Export:	Wrap Cell Content: <input type="checkbox"/>
#	Должность/профессия	COUNT('Табельный №')				
1	Директор	1				
2	Завуч	1				

## Упорядочение вывода

Таблицы содержат неупорядоченные (по определению отношения) наборы строк, и поэтому данные запроса, необязательно появляются какой-то определенной последовательности. Чтобы упорядочить результат запроса используют предложение `ORDER BY`. Эта команда упорядочивает вывод согласно значениям столбцов, указанных в предложении `ORDER BY`. Порядок сортировки можно определять по возрастанию (ключевое слово `ASC`) или по убыванию (ключевое слово `DESC`) для каждого столбца, указанного предложении. По умолчанию установлено возрастание.

Запрос: Вывести таблицу Работников школы, упорядоченную по величине оклада (т.е. по убыванию окладов):



The screenshot shows a database query editor with a SQL query in the top pane and a result grid in the bottom pane. The query is: `SELECT * FROM `Работники школы` ORDER BY `Оклад` DESC;`

The result grid displays the following data:

#	Табельный №	ФИО учителя	Должность/профессия	Пол	Оклад	
1	103	Федоров Юрий Васильевич	Директор	М	200000	
2	102	Рыбакова Анна Ивановна	Завуч	Ж	90000	
3	104	Смирнов Антон Юрьевич	Учитель	М	70000	
4	101	Петрова Валентина Григорьевна	Учитель	Ж	55000	
5	100	Петров Станислав Васильевич	Учитель	М	50000	
*	NULL	NULL	NULL	NULL	NULL	

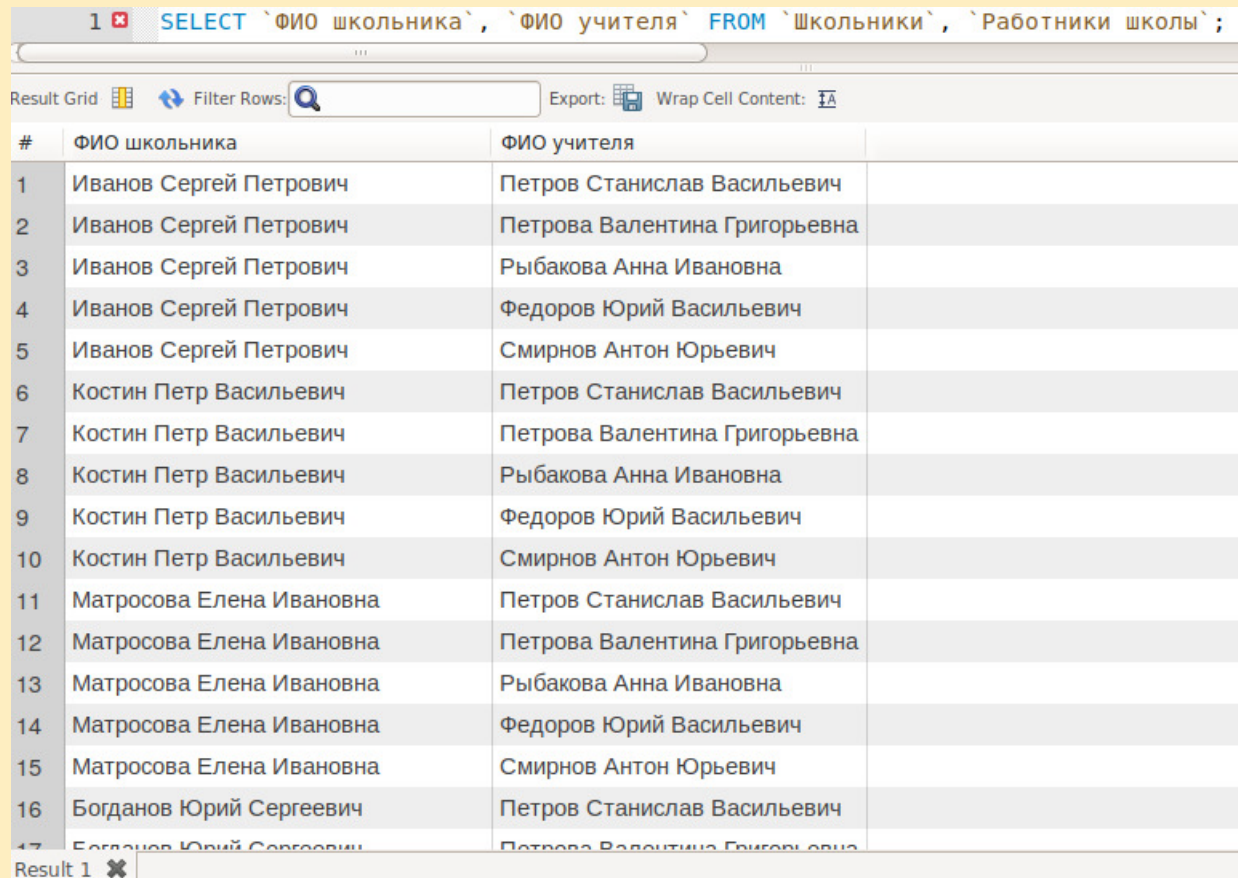
Можно сортировать по нескольким полям, указывая их через запятую. Предложение `ORDER BY` может использоваться совместно с предложением `GROUP BY`, для упорядочения групп `ORDER BY` всегда записывается последней строкой запроса.



# Соединение таблиц

SQL позволяет выбирать данные сразу из нескольких таблиц. Однако, если не вводить ограничений, то в результате все записи одной таблицы будут сочетаться со всеми записями другой, т.е. получим их декартово произведение:

Например, требуется выбрать напротив каждого школьника его учителя:



The screenshot shows a database query interface with the following SQL statement: `SELECT `ФИО школьника`, `ФИО учителя` FROM `Школьники`, `Работники школы`;` The result is displayed in a table with 4 columns. The first column is an index from 1 to 17. The second column lists the names of 5 students (Ivanov, Kostin, Matrosova, Bogdanov). The third column lists the names of 4 teachers (Petrov, Petrova, Rybakova, Fedorov, Smirnov). The fourth column is empty. This represents a Cartesian product where every student is paired with every teacher.

#	ФИО школьника	ФИО учителя	
1	Иванов Сергей Петрович	Петров Станислав Васильевич	
2	Иванов Сергей Петрович	Петрова Валентина Григорьевна	
3	Иванов Сергей Петрович	Рыбакова Анна Ивановна	
4	Иванов Сергей Петрович	Федоров Юрий Васильевич	
5	Иванов Сергей Петрович	Смирнов Антон Юрьевич	
6	Костин Петр Васильевич	Петров Станислав Васильевич	
7	Костин Петр Васильевич	Петрова Валентина Григорьевна	
8	Костин Петр Васильевич	Рыбакова Анна Ивановна	
9	Костин Петр Васильевич	Федоров Юрий Васильевич	
10	Костин Петр Васильевич	Смирнов Антон Юрьевич	
11	Матросова Елена Ивановна	Петров Станислав Васильевич	
12	Матросова Елена Ивановна	Петрова Валентина Григорьевна	
13	Матросова Елена Ивановна	Рыбакова Анна Ивановна	
14	Матросова Елена Ивановна	Федоров Юрий Васильевич	
15	Матросова Елена Ивановна	Смирнов Антон Юрьевич	
16	Богданов Юрий Сергеевич	Петров Станислав Васильевич	
17	Богданов Юрий Сергеевич	Петрова Валентина Григорьевна	

Как видно, для одного и того же школьника просто перечисляются все возможные работники школы.

Чтобы выполнить требуемый запрос, необходимо выполнить соединение таблиц. Соединение является основной операцией в реляционных базах данных, которая формирует производные таблицы для «содержательных» запросов.

Таблицы, участвующие в соединении, задаются списком в предложении FROM. Предикаты (условия запроса) могут ссылаться на любые столбцы производной таблицы.

Полное имя столбца состоит из имени таблицы, отделяемой точкой от собственно имени столбца. В предыдущих запросах можно было опускать имена таблиц, потому что запрос проводился только с одной таблицей. Когда делается запрос со многими таблицами, можно опускать имя таблицы, если ее столбцы имеют имена, отличные от имен остальных таблиц.

# Соединение таблиц

Пример: вывести ФИО школьника, его пол и специализацию класса, в котором он учится:

```
1 SELECT `Школьники`.`ФИО школьника`, `Классы`.`Специализация класса`, `Школьники`.`Пол`
2 FROM `Школьники`, `Классы` WHERE `Школьники`.`№ класса`=`Классы`.`№ класса`;
```

Result Grid Filter Rows: Export: Wrap Cell Content:

#	ФИО школьника	Специализация класса	Пол
1	Иванов Сергей Петрович	Иностранные языки(английский ...	М
2	Костин Петр Васильевич	Иностранные языки(английский ...	М
3	Матросова Елена Ивановна	Иностранные языки(английский ...	Ж
4	Богданов Юрий Сергеевич	Математика и физика	М
5	Потапова Юлия Петровна	Математика и физика	Ж
6	Сорокина Ольга Петровна	Математика и физика	Ж

Result 1

Соединение таблиц через соответствующие первичные и внешние ключи очень часто используется и называется естественным соединением.

В уникальных названиях полей можно не указывать таблицу, поэтому запрос можно чуть упростить:

```
SELECT 'ФИО школьника','Специализация класса','Пол' FROM 'Школьники', 'Классы' WHERE 'Школьники'. '№ класса'='Классы'. '№ класса';
```

# Соединение таблиц

В одном запросе можно соединять сразу несколько таблиц. Для этого вернемся к запросу: напротив каждого школьника указать его учителя:

```
1 SELECT `ФИО школьника`, `ФИО учителя` FROM `Школьники`, `Работники школы`, `Классы`
2 WHERE `Школьники`.`№ класса` = `Классы`.`№ класса`
3 AND `Классы`.`Табельный № учителя` = `Работники школы`.`Табельный №` ;
```

Result Grid Filter Rows: Export: Wrap Cell Content:

#	ФИО школьника	ФИО учителя	
1	Богданов Юрий Сергеевич	Петров Станислав Васильевич	
2	Потапова Юлия Петровна	Петров Станислав Васильевич	
3	Сорокина Ольга Петровна	Петров Станислав Васильевич	
4	Сидоров Андрей Петрович	Петров Станислав Васильевич	
5	Иванов Сергей Петрович	Петрова Валентина Григорьевна	
6	Костин Петр Васильевич	Петрова Валентина Григорьевна	
7	Матросова Елена Ивановна	Петрова Валентина Григорьевна	

Это пример естественного соединения. Соединения, которые используют условие равенства, называются эквисоединения (соединениями по равенству). Соединения по равенству - это наиболее распространенный вид соединения, но имеются и другие варианты.

# Соединение таблиц

Можно использовать любой из операторов сравнения в соединении соединения называют «тета-соединениями». Вот пример такого вида соединения.

Запрос: Вывести список работников школы, за которыми не закреплены классы:

```
1 SELECT DISTINCT `ФИО учителя` FROM `Работники школы`, `Классы`
2 WHERE `Классы`.`Табельный № учителя`+1< `Работники школы`.`Табельный №` ;
```

Result Grid Filter Rows: Export: Wrap Cell Content:

#	ФИО учителя
1	Рыбакова Анна Ивановна
2	Федоров Юрий Васильевич
3	Смирнов Антон Юрьевич

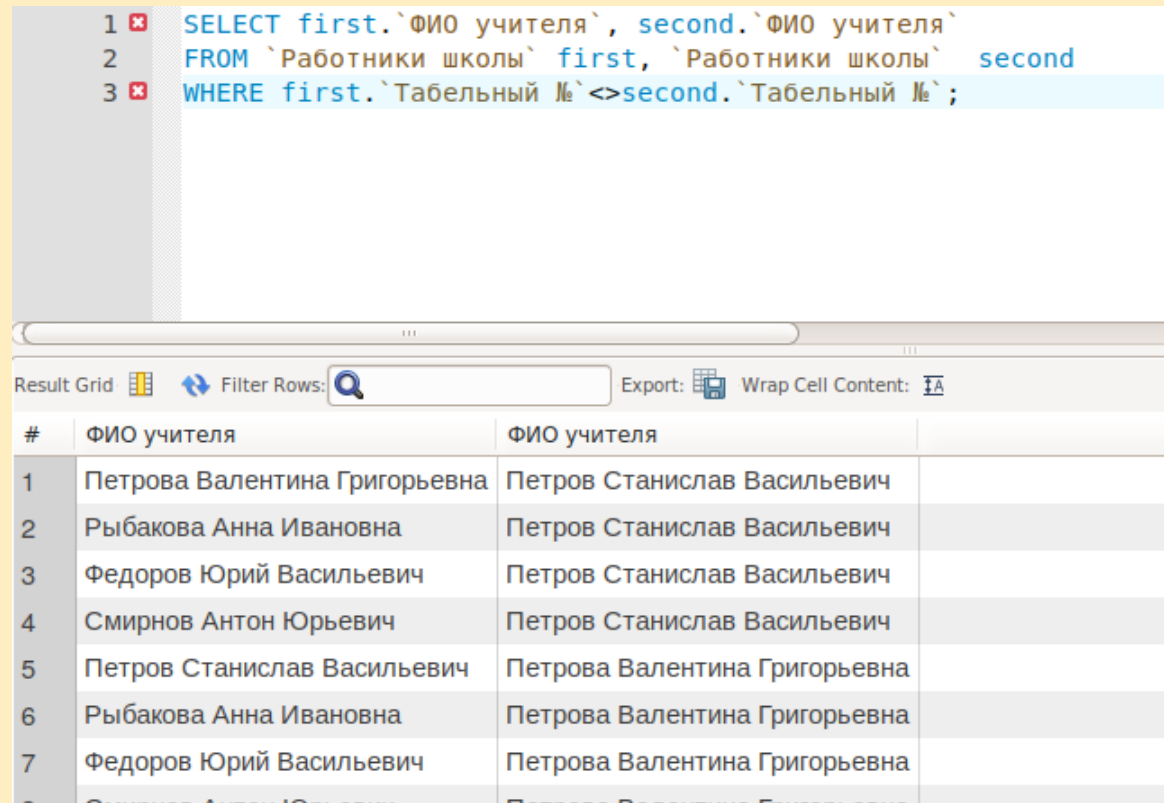
Данное решение является несколько искусственным, но демонстрирует использование тета-соединения.

# Соединение таблиц

Возможно соединение таблицы с самой собой, в этом случае соединение таблицы с собой выполняется, как объединение двух копий одной и той же таблицы. Таблица на самом деле не копируется, но SQL выполняет команду так, как если бы это было сделано. Другими словами, это соединение - такое же, как и любое другое соединение между двумя таблицами, за исключением того, что в данном случае обе таблицы идентичны.

Пример: построить всевозможные соединения школьных работников друг с другом, исключая соединение с самим собой:

```
1 SELECT first.`ФИО учителя`, second.`ФИО учителя`
2 FROM `Работники школы` first, `Работники школы` second
3 WHERE first.`Табельный №` <> second.`Табельный №`;
```



#	ФИО учителя	ФИО учителя
1	Петрова Валентина Григорьевна	Петров Станислав Васильевич
2	Рыбакова Анна Ивановна	Петров Станислав Васильевич
3	Федоров Юрий Васильевич	Петров Станислав Васильевич
4	Смирнов Антон Юрьевич	Петров Станислав Васильевич
5	Петров Станислав Васильевич	Петрова Валентина Григорьевна
6	Рыбакова Анна Ивановна	Петрова Валентина Григорьевна
7	Федоров Юрий Васильевич	Петрова Валентина Григорьевна
8	Смирнов Антон Юрьевич	Петрова Валентина Григорьевна

Синтаксис команды для объединения таблицы с собой, тот же что и для объединения многочисленных таблиц, в одном экземпляре. Когда соединяется таблица с собой, все «повторяемые» имена столбца, содержат префиксы имени таблицы. Чтобы ссылаться на эти столбцы внутри запроса, нужно иметь два различных имени для этой таблицы. Это можно сделать с помощью определения временных имен, называемых Псевдонимы (алиасы). Они определяются в предложении FROM — записывается имя таблицы, и через пробел, указывается ее псевдоним для данного запроса.

Запрос выполняется так, как если бы соединялись две таблицы называемые 'первая' и 'вторая'. Псевдонимы разрешают обработать одну таблицу как две независимых таблицы. Псевдонимы first и second были определены в предложении FROM. Псевдонимы существуют локально только для данной команды. Когда запрос завершен, псевдонимы, используемые в нем, больше не имеют значения.



## Соединение таблиц

Альтернативой рассмотренному способу записи соединения является появившаяся в стандарте ANSI SQL-92 инструкция JOIN. Для соединения двух таблиц предлагается следующий синтаксис:

```
SELECT <таблица>.<столбец> [... n] FROM <таблица1> {INNER | {LEFT |  
RIGHT | FULL} | CROSS } JOIN <таблица2> ON <условие>
```

Чтобы пояснить особенности работы разных типов такого соединения, рассмотрим примеры.

Для демонстрации создадим еще одну таблицу GROUPS, не связанную внешними ключами с остальными таблицами. Будет отмечать в ней сотрудников школы, которые проводили инструктаж по технике безопасности в различных классах:

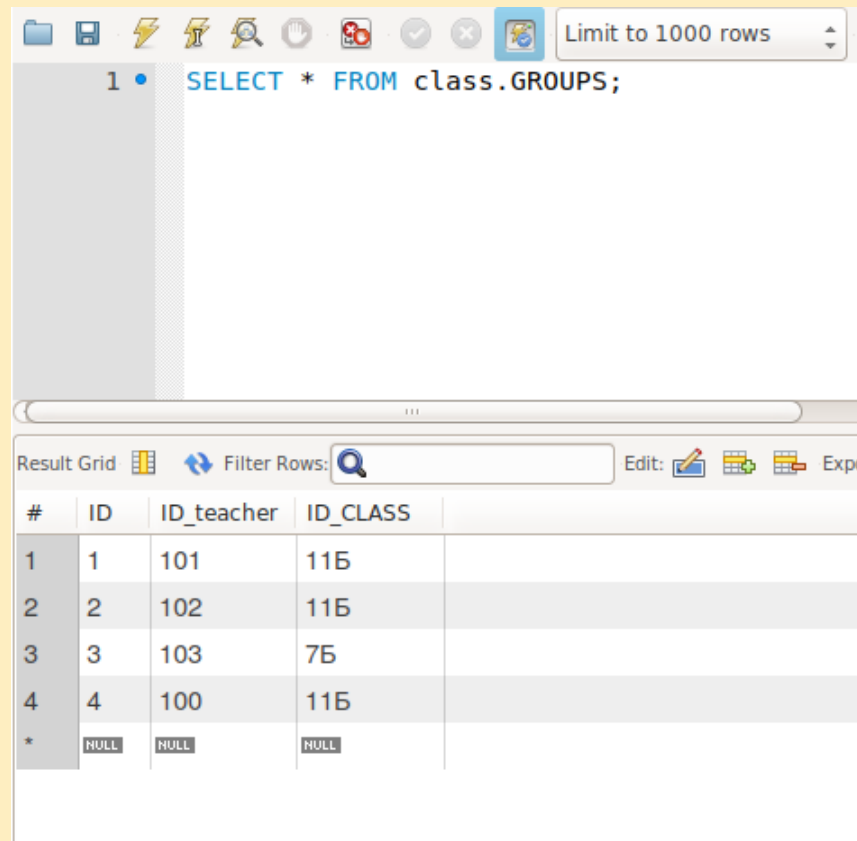
```
CREATE TABLE GROUPS ( ID INT PRIMARY KEY NOT NULL, ID _teacher  
INT, ID _CLASS VARCHAR(20));
```

```
INSERT INTO GROUPS(ID, ID _teacher, ID _CLASS ) VALUES (1,101, '11Б'),  
(2, 102, '11Б'), (3, 103, '7Б'), (4, 100,'11Б') ;
```



# Соединение таблиц

В результате этих действий получим следующую таблицу:



The screenshot shows a database query tool interface. At the top, there is a toolbar with various icons and a dropdown menu set to "Limit to 1000 rows". Below the toolbar, a SQL query is entered in a text area: `1 • SELECT * FROM class.GROUPS;`. Below the query editor, the "Result Grid" is displayed, showing the results of the query. The grid has columns labeled #, ID, ID\_teacher, and ID\_CLASS. The results are as follows:

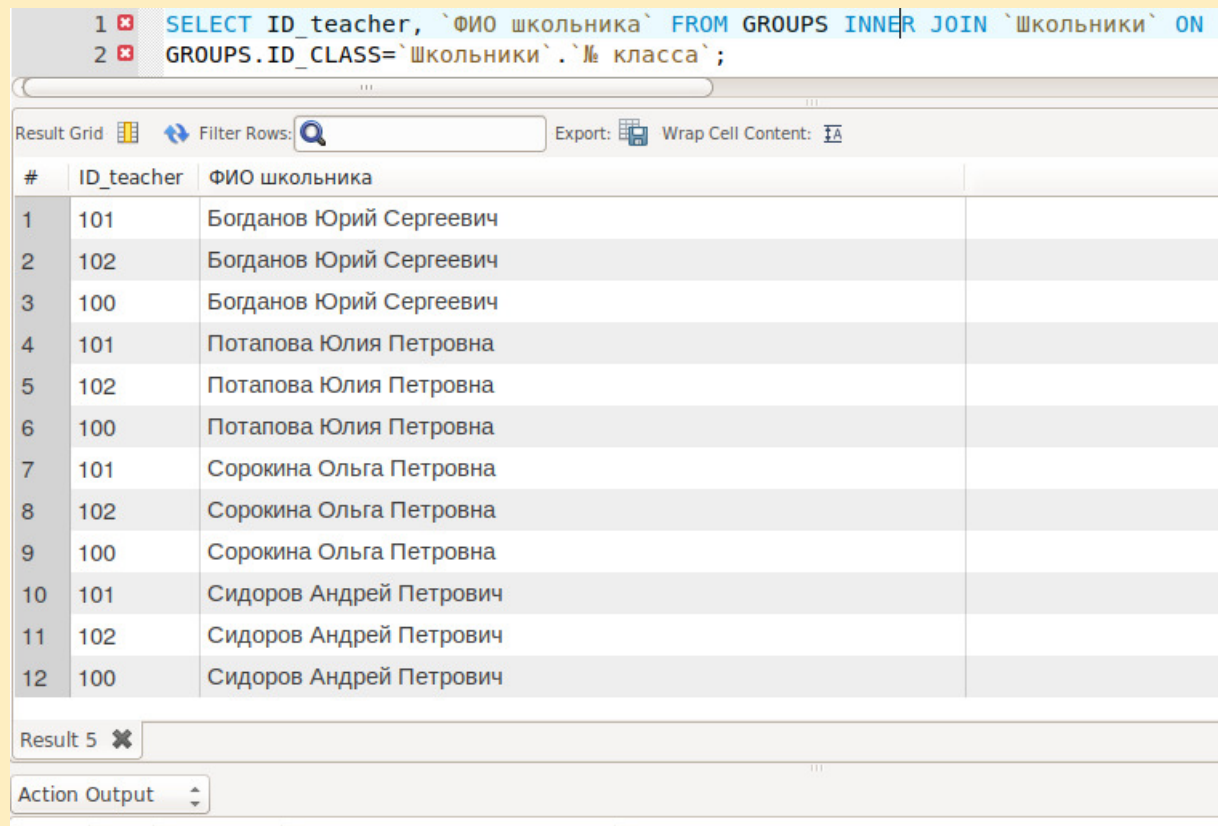
#	ID	ID_teacher	ID_CLASS
1	1	101	11Б
2	2	102	11Б
3	3	103	7Б
4	4	100	11Б
*	NULL	NULL	NULL

# Соединение таблиц

Пробуем получить список, у каких школьников работники школы проводили инструктаж:

Используем для этих целей несколько вариантов запросов JOIN:

Используем INNER:



The screenshot shows a database query interface. At the top, there are two lines of SQL code:

```
1 SELECT ID_teacher, `ФИО школьника` FROM GROUPS INNER JOIN `Школьники` ON  
2 GROUPS.ID_CLASS=`Школьники`.`класс`;
```

Below the query editor, there is a toolbar with options like "Result Grid", "Filter Rows", "Export", and "Wrap Cell Content". The main area displays a table with 12 rows of results. The table has three columns: "#", "ID\_teacher", and "ФИО школьника".

#	ID_teacher	ФИО школьника
1	101	Богданов Юрий Сергеевич
2	102	Богданов Юрий Сергеевич
3	100	Богданов Юрий Сергеевич
4	101	Потапова Юлия Петровна
5	102	Потапова Юлия Петровна
6	100	Потапова Юлия Петровна
7	101	Сорокина Ольга Петровна
8	102	Сорокина Ольга Петровна
9	100	Сорокина Ольга Петровна
10	101	Сидоров Андрей Петрович
11	102	Сидоров Андрей Петрович
12	100	Сидоров Андрей Петрович




At the bottom of the interface, there is a "Result 5" tab and an "Action Output" dropdown menu.

В результате получилось 12 записей, из тех для которых информация есть в обеих таблицах.

# Соединение таблиц

В принципе INNER эквивалентен уже рассмотренному соединению:

```
1 SELECT ID_teacher, `ФИО школьника` FROM GROUPS, `Школьники` WHERE
2 GROUPS.ID_CLASS=`Школьники`.`Ид класса`;
```

Result Grid  Filter Rows:  Export:  Wrap Cell Content: 

#	ID_teacher	ФИО школьника
1	101	Богданов Юрий Сергеевич
2	102	Богданов Юрий Сергеевич
3	100	Богданов Юрий Сергеевич
4	101	Потапова Юлия Петровна
5	102	Потапова Юлия Петровна
6	100	Потапова Юлия Петровна
7	101	Сорокина Ольга Петровна
8	102	Сорокина Ольга Петровна
9	100	Сорокина Ольга Петровна
10	101	Сидоров Андрей Петрович
11	102	Сидоров Андрей Петрович
12	100	Сидоров Андрей Петрович

# Соединение таблиц

```
1 SELECT ID_teacher, `ФИО школьника` FROM GROUPS LEFT JOIN `Школьники` ON
2 GROUPS.ID_CLASS=`Школьники`.`№ класса` ;
```

Result Grid Filter Rows: Export: Wrap Cell Content:

#	ID_teacher	ФИО школьника
1	101	Богданов Юрий Сергеевич
2	102	Богданов Юрий Сергеевич
3	100	Богданов Юрий Сергеевич
4	101	Потапова Юлия Петровна
5	102	Потапова Юлия Петровна
6	100	Потапова Юлия Петровна
7	101	Сорокина Ольга Петровна
8	102	Сорокина Ольга Петровна
9	100	Сорокина Ольга Петровна
10	101	Сидоров Андрей Петрович
11	102	Сидоров Андрей Петрович
12	100	Сидоров Андрей Петрович
13	103	NULL

Result 6

Как видно, в данном случае левая таблица объявляется главной, и даже если для какой либо записи из главной таблицы во вспомогательной таблице отсутствуют сопоставляемые значения, такая запись все равно включается в результат запроса. Например, для сотрудника ID\_teacher=103 13-я запись содержит null.

# Соединение таблиц

```
1 SELECT ID_teacher, `ФИО школьника` FROM GROUPS RIGHT JOIN `Школьники` ON
2 GROUPS.ID_CLASS=`Школьники`.`класс`;
```

Result Grid

#	ID_teacher	ФИО школьника
3	101	Сорокина Ольга Петровна
4	101	Сидоров Андрей Петрович
5	102	Богданов Юрий Сергеевич
6	102	Потапова Юлия Петровна
7	102	Сорокина Ольга Петровна
8	102	Сидоров Андрей Петрович
9	100	Богданов Юрий Сергеевич
10	100	Потапова Юлия Петровна
11	100	Сорокина Ольга Петровна
12	100	Сидоров Андрей Петрович
13	NULL	Иванов Сергей Петрович
14	NULL	Костин Петр Васильевич
15	NULL	Матросова Елена Ивановна

Result 7

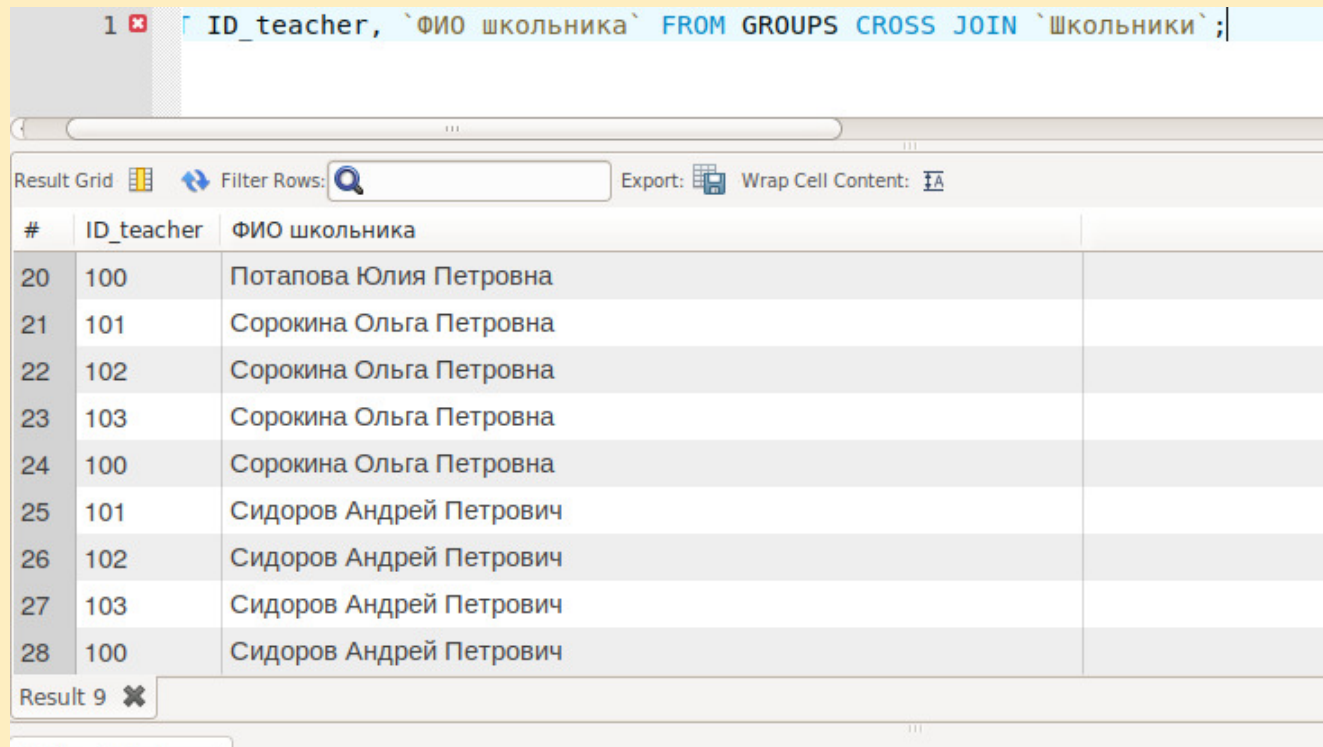
Action Output

Теперь же главной таблицей объявляется правая таблица, поэтому все записи из нее появятся в результате запроса.

Аналогичным образом, в результате запроса FULL должны объединиться результаты LEFT и RIGHT. Однако MySQL не поддерживает FULL JOIN.

# Соединение таблиц

CROSS JOIN дает альтернативный способ получить декартово произведение:



The screenshot shows a SQL query editor with a query window at the top containing the following SQL statement:

```
1 x ID_teacher, `ФИО школьника` FROM GROUPS CROSS JOIN `Школьники`;
```

Below the query window is a "Result Grid" with a search bar and "Export" and "Wrap Cell Content" buttons. The grid displays the results of the query, which is a Cartesian product of the GROUPS and Школьники tables. The grid has three columns: #, ID\_teacher, and ФИО школьника. The results are as follows:

#	ID_teacher	ФИО школьника
20	100	Потапова Юлия Петровна
21	101	Сорокина Ольга Петровна
22	102	Сорокина Ольга Петровна
23	103	Сорокина Ольга Петровна
24	100	Сорокина Ольга Петровна
25	101	Сидоров Андрей Петрович
26	102	Сидоров Андрей Петрович
27	103	Сидоров Андрей Петрович
28	100	Сидоров Андрей Петрович

At the bottom of the grid, it says "Result 9" with a close button.

# Подзапросы

Подзапросом называется оператор SELECT, вложенный в другой SELECT. Рассмотрим пример:

Запрос: получить список учеников, у которых школьный учитель имеет оклад более 50 тыс. руб.

```
1 Select `Школьники`.`ФИО школьника`, г.`ФИО учителя` FROM `Школьники`,  
2 (SELECT * FROM `Работники школы` WHERE `Должность/профессия`='учитель' AND `Оклад`>50000) г,  
3 `Классы` WHERE г.`Табельный №`=`Классы`.`Табельный № учителя` AND  
4 `Классы`.`№ класса`=`Школьники`.`№ класса`;
```

Result Grid				Filter Rows:		Export:		Wrap Cell Content:			
#	ФИО школьника	ФИО учителя									
1	Иванов Сергей Петрович	Петрова Валентина Григорьевна									
2	Костин Петр Васильевич	Петрова Валентина Григорьевна									
3	Матросова Елена Ивановна	Петрова Валентина Григорьевна									

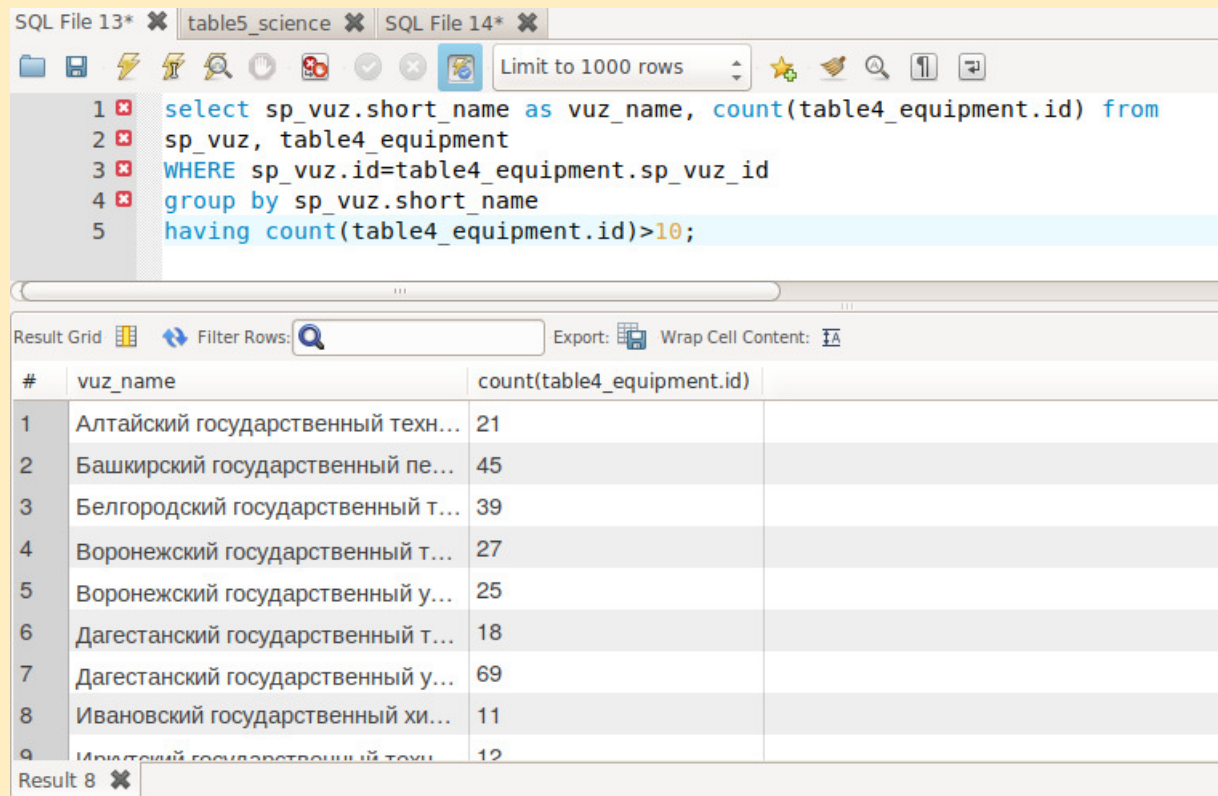
# Операция объединения

Совместимые по типу реляционные отношения (состоящие из одинаковых столбцов)  $A$  и  $B$  можно объединять с помощью операции **UNION**. Если в таблицах  $A$  и  $B$  будут совпадающие строки, дубликаты в результат не включаются.

Пример: Отобразить перечень вузов, у которых количество единиц оборудования больше 10 единиц или количество штатных преподавателей больше 100 человек.

Вначале выполним каждый запрос по отдельности:

1. Отбираем перечень вузов, у которых количество единиц оборудования больше 10 единиц.



The screenshot shows a SQL IDE interface with a query editor and a results grid. The query editor contains the following SQL code:

```
1 select sp_vuz.short_name as vuz_name, count(table4_equipment.id) from
2 sp_vuz, table4_equipment
3 WHERE sp_vuz.id=table4_equipment.sp_vuz_id
4 group by sp_vuz.short_name
5 having count(table4_equipment.id)>10;
```

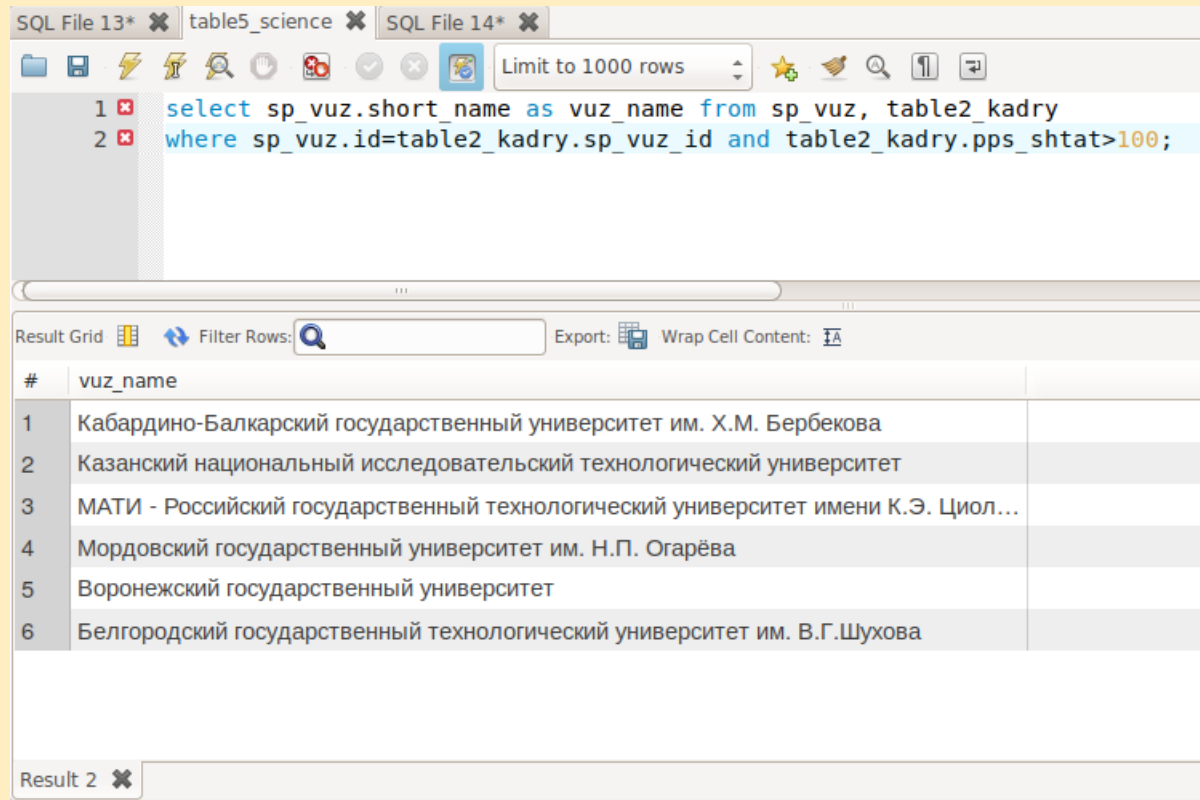
The results grid displays the following data:

#	vuz_name	count(table4_equipment.id)
1	Алтайский государственный техн...	21
2	Башкирский государственный пе...	45
3	Белгородский государственный т...	39
4	Воронежский государственный т...	27
5	Воронежский государственный у...	25
6	Дагестанский государственный т...	18
7	Дагестанский государственный у...	69
8	Ивановский государственный хи...	11
9	Иркутский государственный техн...	12



# Операция объединения

2. Отбираем перечень вузов, у которых количество штатных преподавателей больше 100 человек:



The screenshot shows a SQL IDE interface with two tabs: 'table5\_science' and 'SQL File 14\*'. The query editor contains the following SQL code:

```
1 select sp_vuz.short_name as vuz_name from sp_vuz, table2_kadry
2 where sp_vuz.id=table2_kadry.sp_vuz_id and table2_kadry.pps_shtat>100;
```

Below the query editor, the 'Result Grid' displays the results of the query. It includes a search bar, 'Filter Rows' button, 'Export' button, and 'Wrap Cell Content' checkbox. The results are shown in a table with two columns: '#' and 'vuz\_name'.

#	vuz_name
1	Кабардино-Балкарский государственный университет им. Х.М. Бербекова
2	Казанский национальный исследовательский технологический университет
3	МАТИ - Российский государственный технологический университет имени К.Э. Циол...
4	Мордовский государственный университет им. Н.П. Огарёва
5	Воронежский государственный университет
6	Белгородский государственный технологический университет им. В.Г.Шухова

At the bottom left, there is a tab labeled 'Result 2' with a close button.

# Операция объединения

## 3. Объединяем полученные запросы

The screenshot shows a SQL IDE with a query editor and a result grid. The query is a UNION of two SELECT statements. The first SELECT statement filters for universities with more than 10 equipment items, and the second SELECT statement filters for universities with more than 100 publications. The result grid shows the first four rows of the query results.

```
1 (select x.vuz_name from (select sp_vuz.short_name as vuz_name, count(table4_equipment.id) from
2 sp_vuz, table4_equipment
3 WHERE sp_vuz.id=table4_equipment.sp_vuz_id
4 group by sp_vuz.short_name
5 having count(table4_equipment.id)>10) x)
6 union
7 (select sp_vuz.short_name as vuz_name from sp_vuz, table2_kadry
8 where sp_vuz.id=table2_kadry.sp_vuz_id and table2_kadry.pps_shtat>100)
9
```

Result Grid

#	vuz_name
17	Оренобургский государственный университет
18	Петрозаводский государственный университет
19	Российский государственный педагогический университет им. А. И. Герцена
20	Российский химико-технологический университет имени Д.И. Менделеева

# Операция пересечения

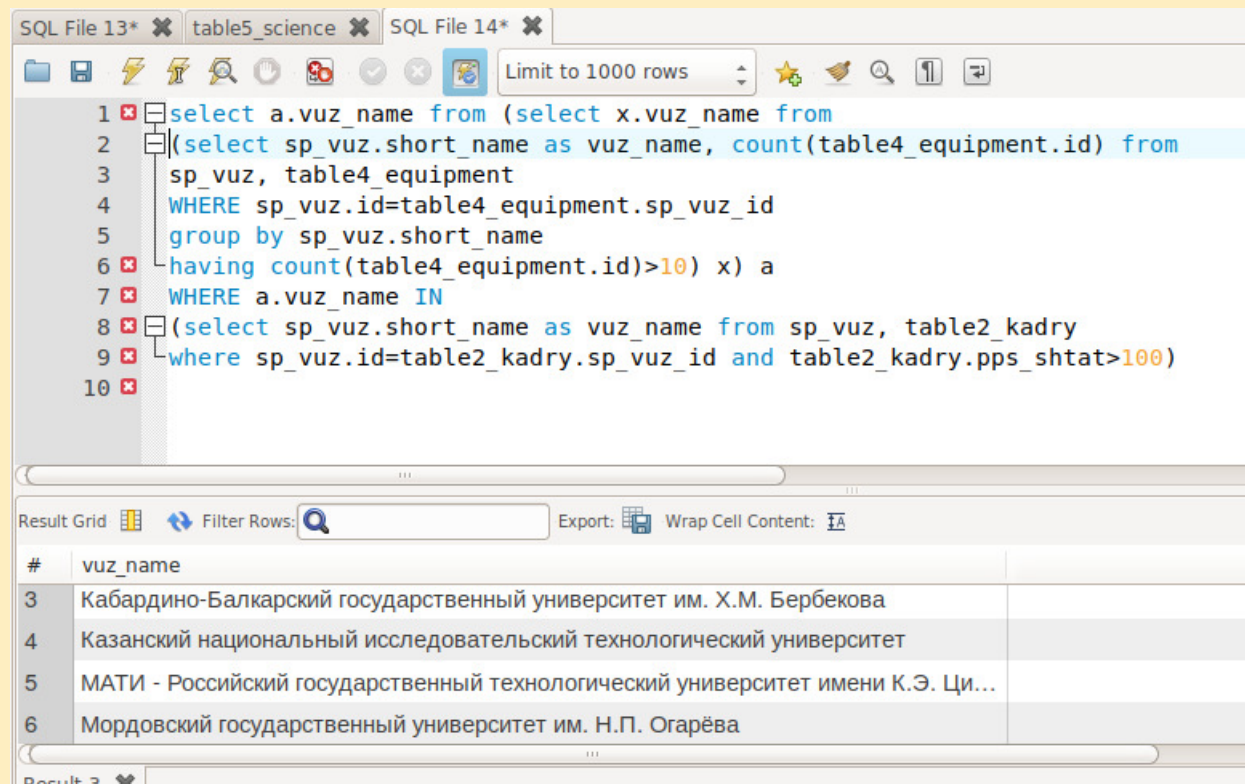
В классическом SQL есть операция перечення двух совместимых по типу реляционных отношений:

```
SELECT * FROM A
```

```
INTERSECT
```

```
SELECT * FROM B
```

Однако, в MySQL в чистом виде такая операция отсутствует. Ее можно имитировать используя предложения IN или EXISTS:



```
1 select a.vuz_name from (select x.vuz_name from
2 (select sp_vuz.short_name as vuz_name, count(table4_equipment.id) from
3 sp_vuz, table4_equipment
4 WHERE sp_vuz.id=table4_equipment.sp_vuz_id
5 group by sp_vuz.short_name
6 having count(table4_equipment.id)>10) x) a
7 WHERE a.vuz_name IN
8 (select sp_vuz.short_name as vuz_name from sp_vuz, table2_kadry
9 where sp_vuz.id=table2_kadry.sp_vuz_id and table2_kadry.pps_shtat>100)
10
```

Result Grid

#	vuz_name
3	Кабардино-Балкарский государственный университет им. Х.М. Бербекова
4	Казанский национальный исследовательский технологический университет
5	МАТИ - Российский государственный технологический университет имени К.Э. Ци...
6	Мордовский государственный университет им. Н.П. Огарёва

# Операция вычитания

В классическом SQL есть операция вычитания двух совместимых по типу реляционных отношений:

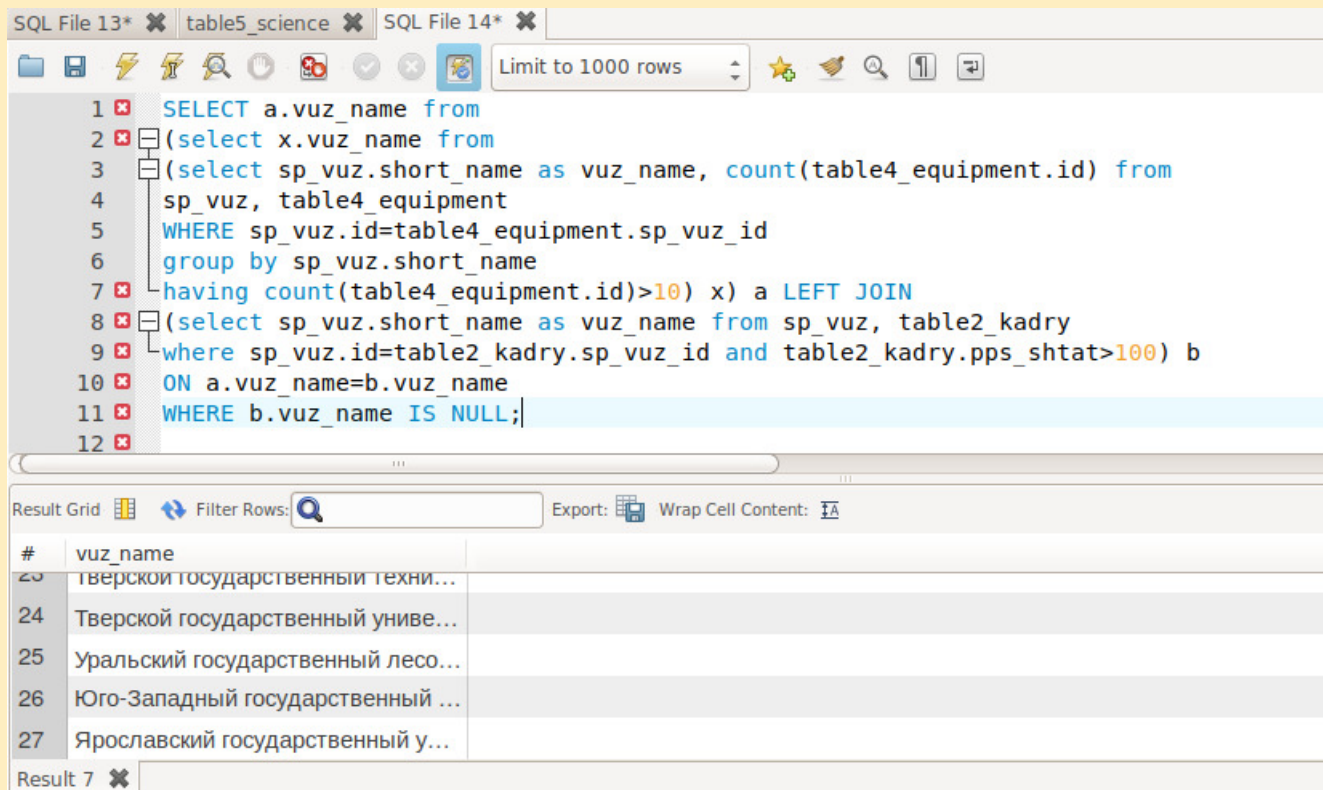
```
SELECT id FROM t1  
MINUS
```

```
SELECT id FROM t2;
```

Однако, в MySQL в чистом виде такая операция отсутствует. Ее можно имитировать используя JOIN:

```
SELECT t1.ID  
FROM t1  
LEFT JOIN t2 ON  
t2.ID=t1.ID  
WHERE t2.ID IS NULL;
```

# Операция вычитания



The screenshot displays a SQL IDE interface with a query editor and a results grid. The query is a complex SQL statement involving multiple subqueries, joins, and filters. The results grid shows a list of university names, with the first row partially visible and truncated.

```
1 SELECT a.vuz_name from
2 (select x.vuz_name from
3 (select sp_vuz.short_name as vuz_name, count(table4_equipment.id) from
4 sp_vuz, table4_equipment
5 WHERE sp_vuz.id=table4_equipment.sp_vuz_id
6 group by sp_vuz.short_name
7 having count(table4_equipment.id)>10) x) a LEFT JOIN
8 (select sp_vuz.short_name as vuz_name from sp_vuz, table2_kadry
9 where sp_vuz.id=table2_kadry.sp_vuz_id and table2_kadry.pps_shtat>100) b
10 ON a.vuz_name=b.vuz_name
11 WHERE b.vuz_name IS NULL;|
12
```

Result Grid

#	vuz_name
23	тверской государственный техни...
24	Тверской государственный униве...
25	Уральский государственный лесо...
26	Юго-Западный государственный ...
27	Ярославский государственный у...

Result 7

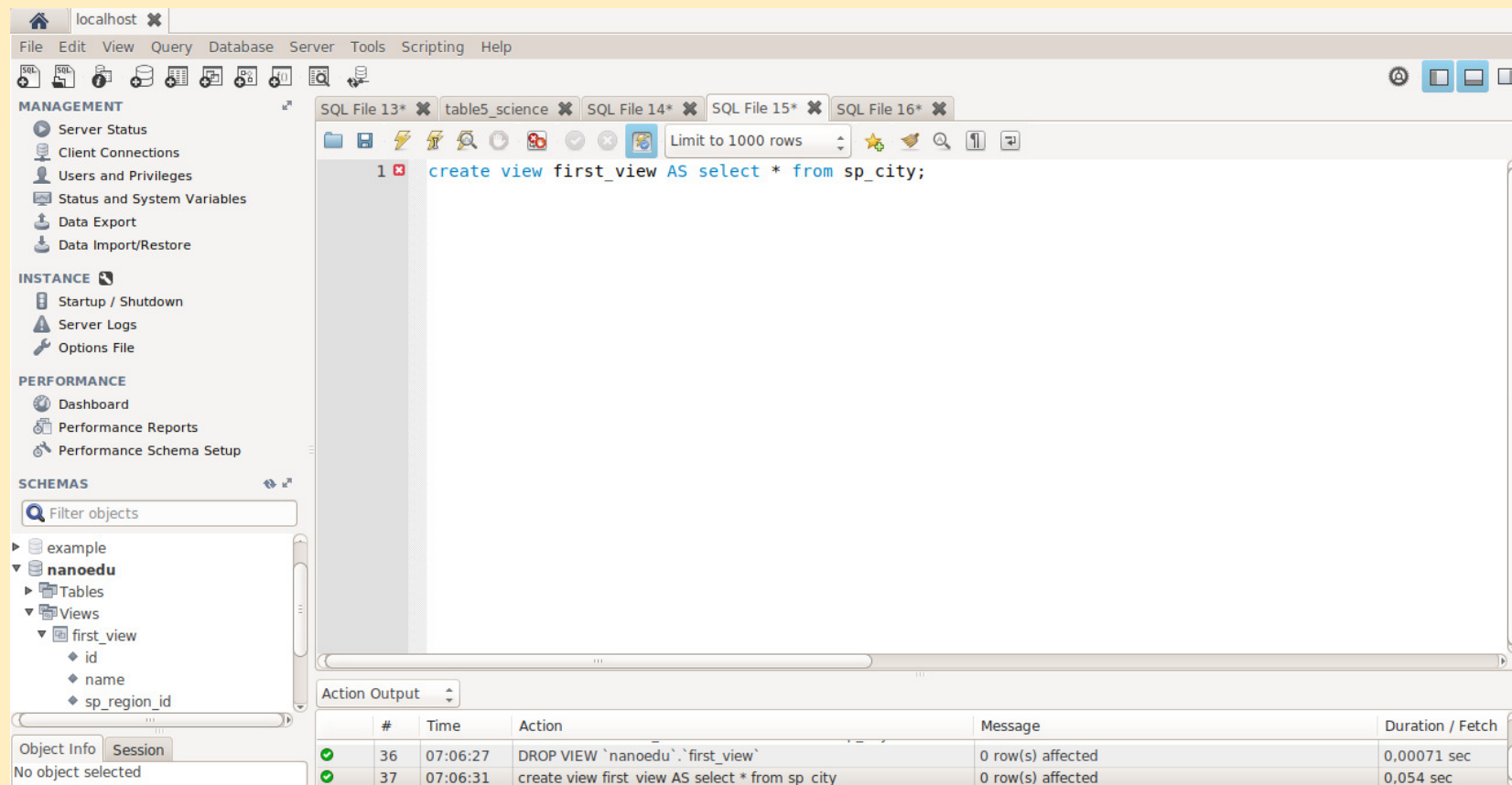
# Представления

Представление (англ. view) по сути, является хранимым в базе данных под отдельным именем запросом на выборку. Собственными хранимыми данными представление не обладает, а возвращает результат обработки выражения SELECT для текущего состояния базы данных.

Представление может быть создано следующей командой:

```
CREATE VIEW <название> AS <запрос>
```

Пример создания представления:



# Представления

Представление является именованным объектом базы данных, и после его создания можно из него запрашивать данные, как из обычной таблицы. Что касается добавления, изменения и удаления данных, то тут есть ряд ограничений. Некоторые представления могут быть обновляемы, некоторые — нет.

Зависит это от того, можно ли производимые изменения однозначно отобразить на те хранимые таблицы, на основе которых определено представление. Некоторые отличия здесь могут быть и у разных СУБД.

Удаление представления выполняется с помощью команды DROP:

```
DROP VIEW <название>
```

Нередко представления используются для задач разграничения доступа к данным. В подобных случаях, создается представление, возвращающее только те данные, к которым пользователь должен иметь доступ. А пользователю предоставляются права на чтение данных из представления и отбираются права на чтение данных из исходных таблиц.

Также с помощью представлений может обеспечиваться логическая независимость данных.

Многие СУБД хранят представления в БД в предварительно обработанном, откомпилированном виде. Это повышает скорость их выполнения по сравнению с аналогичным запросом.