

Запросы на языке SQL

Высшая Школа Цифровой Культуры

Университет ИТМО

[dc@itmo.ru](mailto:dc@itmo.ru)

**Оглавление**

[**Оператор SELECT** 3](#_Toc203129919)

[**Условия выборки** 8](#_Toc203129920)

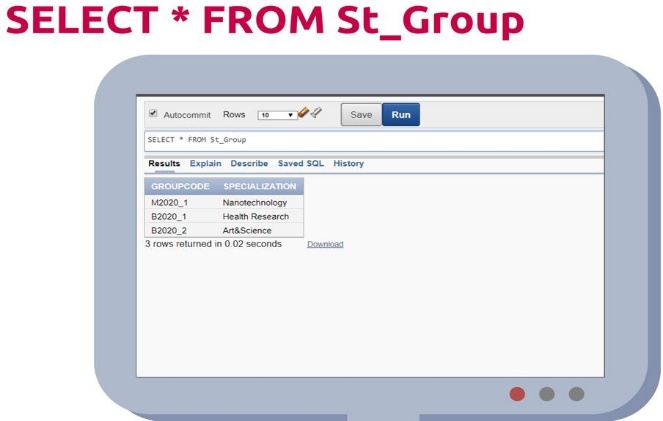
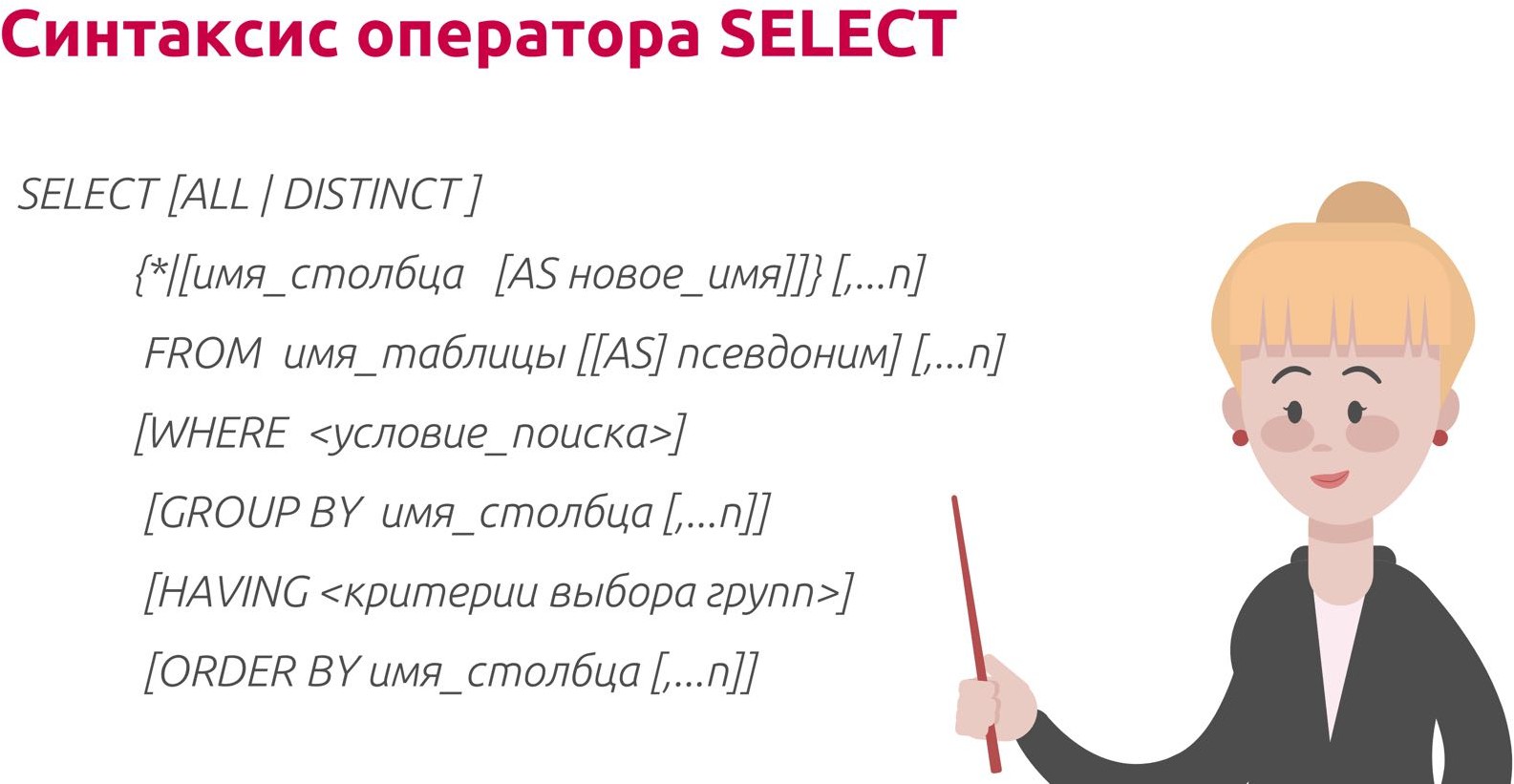
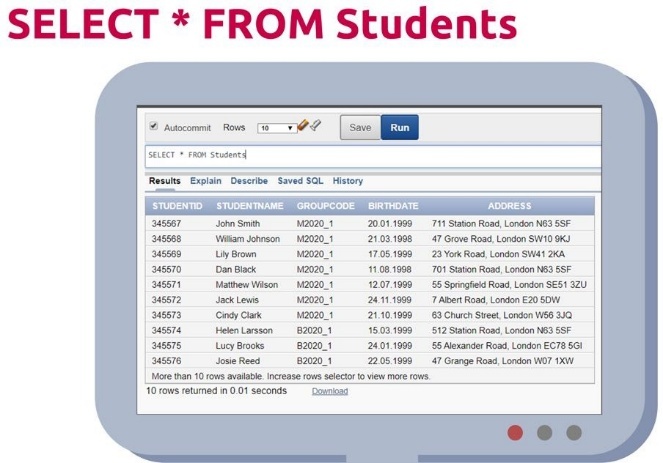
[**Агрегатные функции** 14](#_Toc203129921)

[**Вложенные запросы** 18](#_Toc203129922)

[**Теоретико-множественные операции** 23](#_Toc203129923)

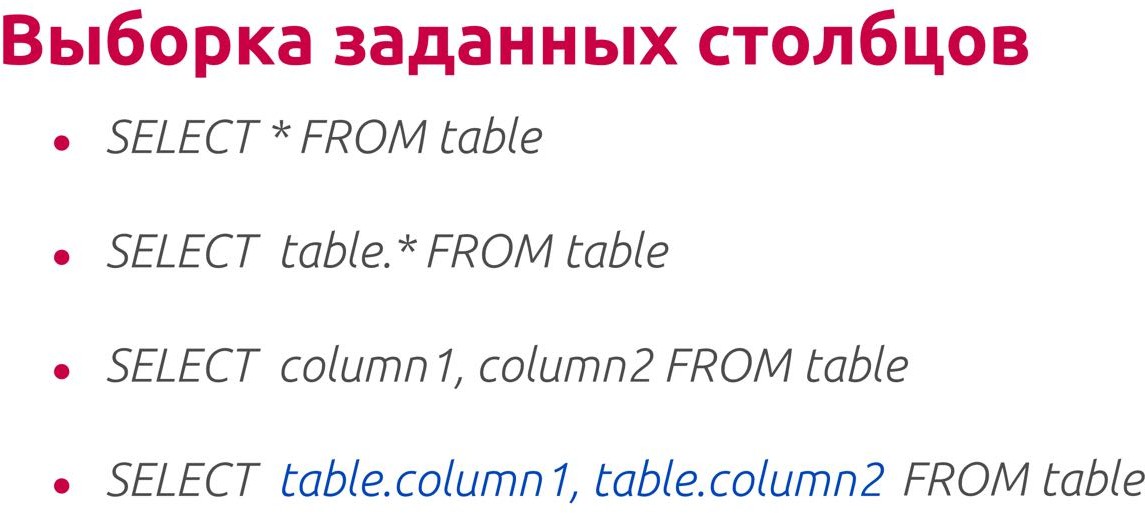
[**Соединение таблиц** 29](#_Toc203129924)

# **Оператор SELECT**

Итак, мы научились проектировать данные, создавать таблицы с заданными типами столбцов и ограничениями целостности и заполнять их значениями. Теперь приступим к написанию запросов. Для того, чтобы написать запрос к данным, есть всего один оператор – он называется SELECT. Всего один оператор, но с его помощью можно писать очень сложные запросы, которые извлекают данные из нескольких таблиц, проверяя множество критериев. Давайте посмотрим на его синтаксис:

* FROM – определяются имена используемых таблиц;
* WHERE – выполняется фильтрация строк объекта в соответствии с заданными условиями;
* GROUP BY – образуются группы строк , имеющих одно и то же значение в указанном столбце;
* HAVING – фильтруются группы строк в соответствии с указанным условием;
* SELECT – устанавливается, какие столбцы должны присутствовать в выходных данных;
* ORDER BY – определяется упорядоченность результата выполнения оператора.

Начнем с самого простого запроса – вывести полностью содержимое указанное таблицы. Имя таблицы указывается после слова FROM. После слова SELECT указывают выводимые столбцы. Если после SELECT указан знак \* (звездочка), это означает «все столбцы». Выведем содержимое таблицы Students и таблицы St\_Group.

Если строк таблицы очень много, то СУБД может отобразить некоторое ограниченное количество строк.

Если нужно выводить не все столбцы таблицы, а некоторые, определенные поля, то их имена надо указать после слова SELECT. Задать столбцы таблицы можно по-разному. Можно указать \* – все столбцы таблицы, можно явно перечислить названия столбцов, а можно перед именем столбца указать имя таблицы. Имя таблицы и имя столбца разделяют точкой. В запросе могут быть использованы несколько таблиц, и если в них встречаются одинаково названные столбцы, то надо обязательно указывать имя таблицы перед именем столбца, чтобы указать, про какой конкретно столбец идет речь. Можно указать все столбцы определенной таблицы, написав им# таблицы.\*.

В первом запросе

SELECT StudentId FROM Students

выводятся значения столбца StudentId таблицы Students – это номера зачеток студентов. Во втором случае

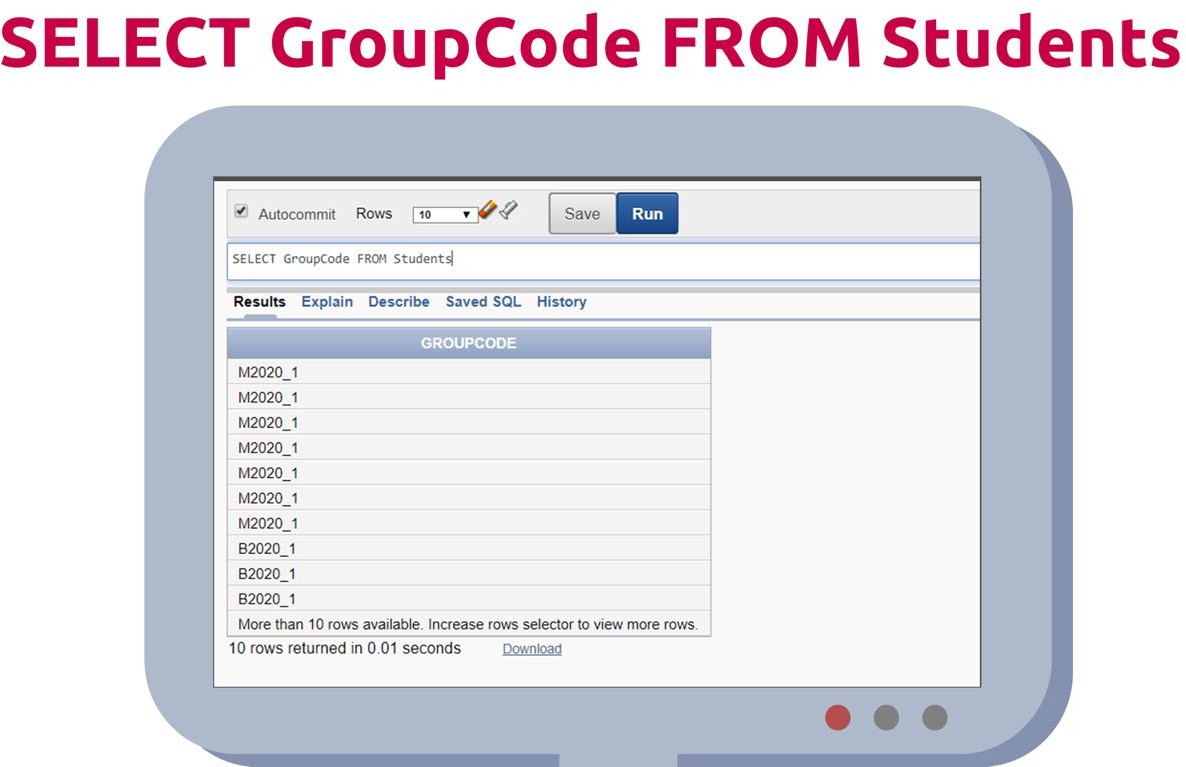
SELECT StudentName, GroupCode FROM Students

выведем StudentName, GroupCode таблицы Students. В каждой группе учится много студентов. Третий запрос

SELECT GroupCode FROM Students

выводит только номера групп, в которых учатся студенты.



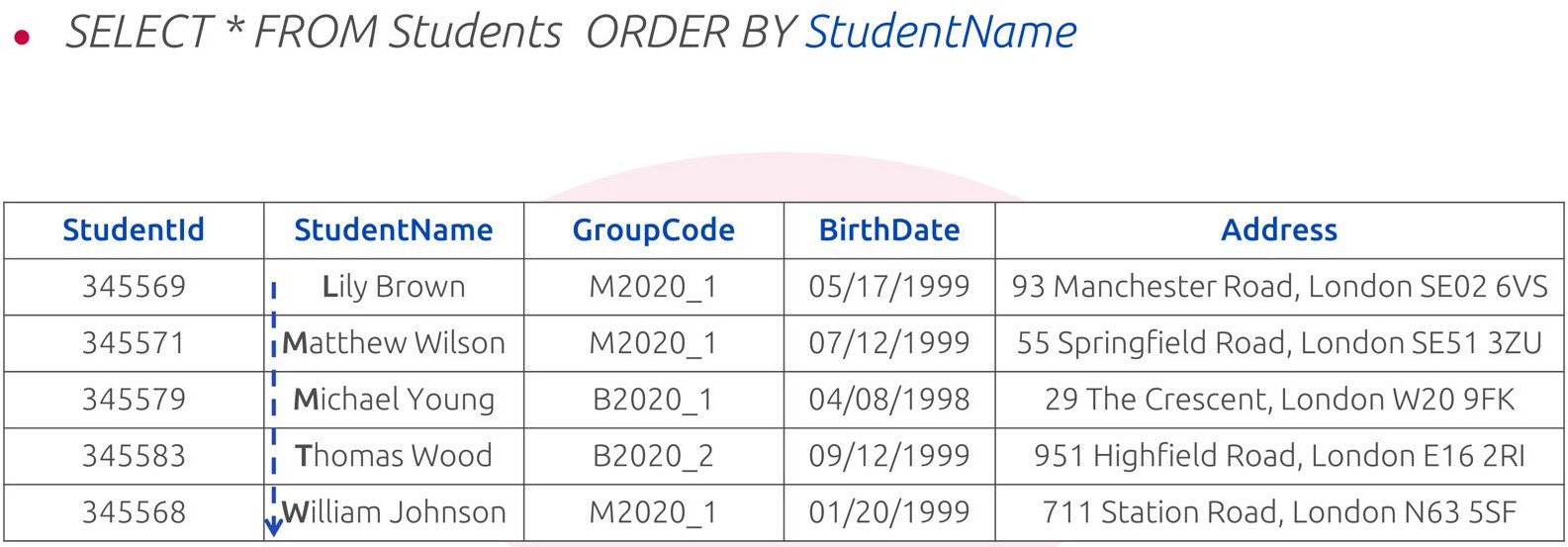
Наверное, Вам покажется странным, что один и тот же номер группы выводится несколько раз – при выполнении запроса значение поля GroupCode выводится для каждой строки, т.е. для каждого студента, невзирая на повторения.



По умолчанию выводятся все значения выборки, чтобы указать это в явном виде, можно перед именами полей указать ключевое слово ALL. Если нужно выводить только уникальные строки, то перед именем поля (или полей) необходимо написать слово DISTINCT, тогда дубликаты строк будут удалены из выборки. В запросе с кодами групп из таблицы Students каждый код группы будет выведен ровно по одному разу.

**Сортировка строк**

Если не указывать критерии сортировки, то строки выборки будут выводиться в произвольном порядке. Если нужно выводить строки в определенном порядке, то это можно сделать при помощи ключевого слова ORDER BY. После этого слово указывают столбец или столбцы, которые являются критерием ранжирования. Порядок сортировки может быть по возрастанию



ASC и по убыванию DESC. Если критерии заданы, но порядок не указан, то по умолчанию используется сортировка по возрастанию. Например, можно вывести все содержимое таблицы Students, упорядочив строки по возрастанию поля StudentName, или по Коду группы, или по Коду группы по убыванию:

SELECT \* FROM Students ORDER BY StudentName SELECT \* FROM Students ORDER BY GroupCode SELECT \* FROM Students ORDER BY GroupCode DESC

Чтобы дополнительно сортировать внутри группы, добавим еще и сортировку по имени.

SELECT \* FROM Students ORDER BY GroupCode DESC, StudentName

Можно выводить не все поля, а только имя студента и код группы:

SELECT StudentName, GroupCode FROM Students ORDER BY GroupCode DESC, StudentName

При этом можно производить сортировку даже по тем полям, которые не включены в итоговую выборку. Например, если в качестве критерия сортировки указать поле StudentId, то строки будут упорядочены по возрастанию этого критерия, а потом их них будут взяты только столбцы StudentName, GroupCode.

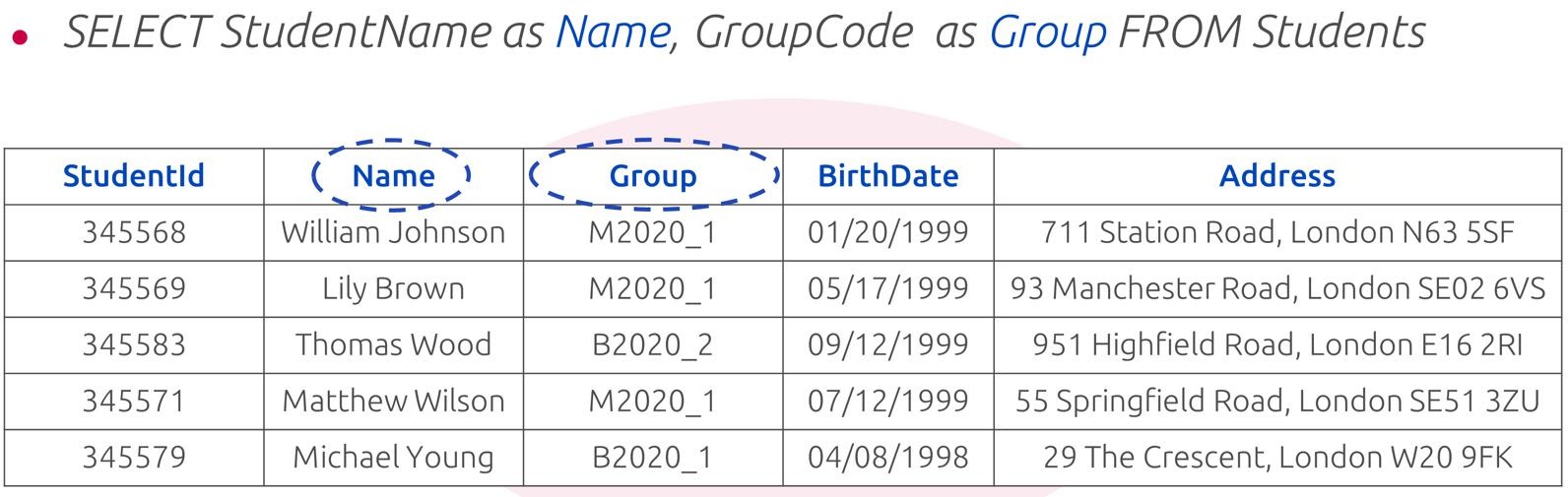
SELECT StudentName, GroupCode FROM Students ORDER BY StudentId

Вместо имен полей при сортировке можно указывать их порядковые номера в исходной таблице, но такой способ указания на столбцы не часто используют.

SELECT StudentName, GroupCode FROM Students ORDER by 2, 1

**Переименование атрибутов**

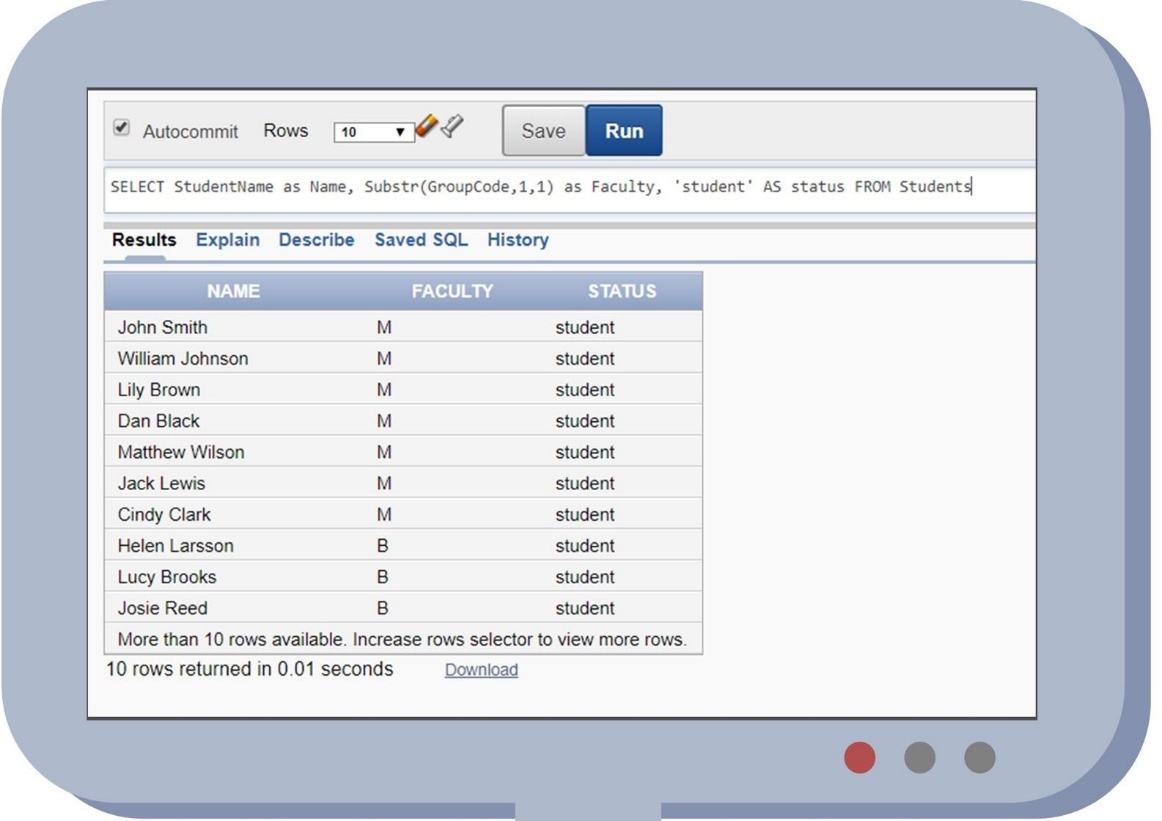
Столбцы выборки называются так, как были названы в исходной таблице. Если их нужно переименовать, то новое название можно указать после указания исходного названия столбца и ключевого слова as:

SELECT StudentName as Name, GroupCode as Group\_Name FROM Students

На самом деле, можно не только выводить значения столбцов таблицы, а составлять их них выражения. Для этого можно использовать арифметические операции, функции преобразования строк и пр. Например, взять первую букву от кода группы и назвать это поле факультетом.

SELECT StudentName as Name, Substr(GroupCode,1,1) as Faculty FROM Students

Или добавить в качестве дополнительного столбца выборки статус со значением Student.

SELECT StudentName as Name, Substr(GroupCode,1,1) as Faculty, 'student' AS status FROM Students

# **Условия выборки**

До сих пор мы задавали множество столбцов выборки, но выводили значения для всех строк таблицы, разве что удаляя дубликаты. Часто требуется вывести не все строки, а только те, которые соответствуют некоторому условию, которое называют условием выборки.

Условие указывают после ключевого слова WHERE. Каким может быть условие? Это некоторое логическое выражение, которое может быть составлено из названий столбцов, констант, операторов сравнения, арифметических операторов, функций и пр. Ну, например, самое простое – когда значение столбца сравнивают с константой. Например, вывести строки таблицы Studentds со значением StudentId = 345569.

SELECT \* FROM Students WHERE StudentId = 345569

В таком случае результат выборки будет единственной строкой, т.к. мы задали в качестве критерия точное значение первичного ключа. Если в качестве условия выборки указать код группы, например, M2020\_1, то результатом будут строки всех студентов в группе.

SELECT \* FROM Students WHERE GroupCode = 'M2020\_1'

Обратите внимание, как указываются константы: строки и даты указываются в кавычках, обычно в одинарных, а числа – без кавычек.

Итак, один из способов задать условие выборки – это операция сравнения. При таком виде условий значение одного выражения сравнивается со значением другого. В результат выборки попадают строки, для которых значение оператора сравнения истинно. Конечно, в качестве операции сравнения может выступать не только равенство, а знаки <>, >, >=, <, <=. Каждый из приведенных операторов сравнения — бинарный, то есть должен использоваться с двумя операндами. Результатом выполнения операторов сравнения является логическое значение, истина или ложь.

Например, выберем всех студентов, для которых значение StudentId < 345569, или всех студентов, у которых код группы не равен M2020\_1

SELECT \* FROM Students WHERE StudentId < 345569

SELECT \* FROM Students WHERE GroupCode <> 'M2020\_1'

Другой возможностью задать условие выборки является принадлежность множеству: проверяется, принадлежит ли результат вычислений выражения заданному множеству значений. Для проверки принадлежности используется ключевое слово IN. Элементами множества могут быть константы соответствующего типа, перечисленные через запятую, или результаты вложенного подзапроса.

Сделаем выборку из таблицы Students, задав в качестве условия принадлежность множеству. Например, зададим условие выборки так: код группы принадлежит множеству ’M2020\_1’, ’B2020\_1’

SELECT \* FROM Students WHERE GroupCode IN ('M2020\_1', 'B2020\_1')

При проектировании таблицы мы упоминали о том, что значения некоторых столбцов могут быть не заданы. Например, можно сохранить информацию о студенте, если мы не знаем дату его рождения. Или назначить для группы экзамен по определенному предмету, указав дату, но не определив сразу аудиторию. Как найти записи, чьи значения атрибутов не заданы?



При этом на первый взгляд кажется, что незаданная строка равна пустой строке, а незаданное число нулю. Но это не так. Для незаданных, неопределенных значений есть специальное обозначение – NULL. С такими значениями надо быть особенно аккуратными. Если обычные, заданные значения, мы всегда можем сравнить – равны они или нет, то с незаданными значениями и тут неопределенность. Если нам неизвестно, в каких аудиториях проводятся два разных экзамена, можем ли мы утверждать, что они проходят в одной и той же аудитории? Конечно, нет. Но и утверждать, что они проводятся в разных, мы тоже не можем. Точно также при сравнении на равенство заданных значений с незаданными мы получаем не истину или ложь, как в обычном случае, а неопределенное значение. Для проверки, является ли значение какого-либо столбца незаданным значением, необходимо использовать фразу IS NULL, что означает незаданное значение, и IS NOT NULL, что означает заданное, определенное значение.

Давайте напишем примеры запросов. Например, вывести всех студентов, у которых дата рождения не задана:

SELECT \* FROM Students WHERE BirthDate IS NULL

Или найдем экзамены, для проведения которых не определена аудитория:

SELECT \* from exam\_Sheet WHERE Classroom IS NULL

А теперь аудитория определена:

SELECT \* from exam\_Sheet WHERE Classroom IS NOT NULL



Для строковых значений есть возможность поиска по заданному шаблону. Например, может потребоваться найти все значения строкового столбца, начинающиеся на какую-то букву, или содержащие определенную последовательность символов.

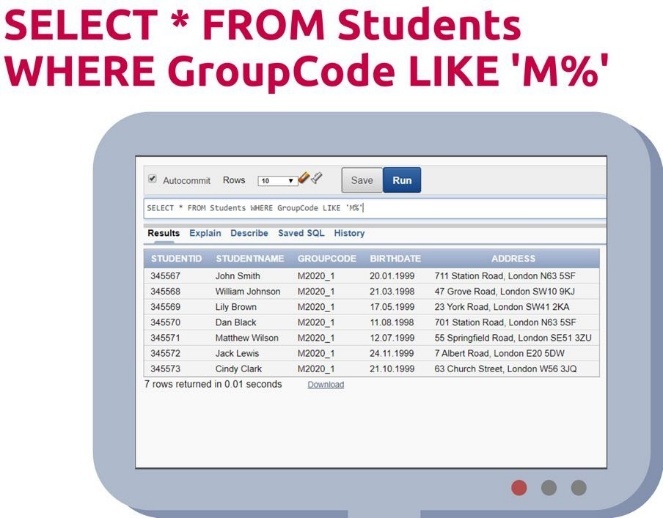
Для этого используется ключевое слово LIKE, которое указывается после имени столбца. После этого необходимо задать сам шаблон. В нем могут быть использованы специальные символы: % означает любую последовательность символов, в том числе и пустую, а \_ (подчеркивание) обозначает один любой символ.

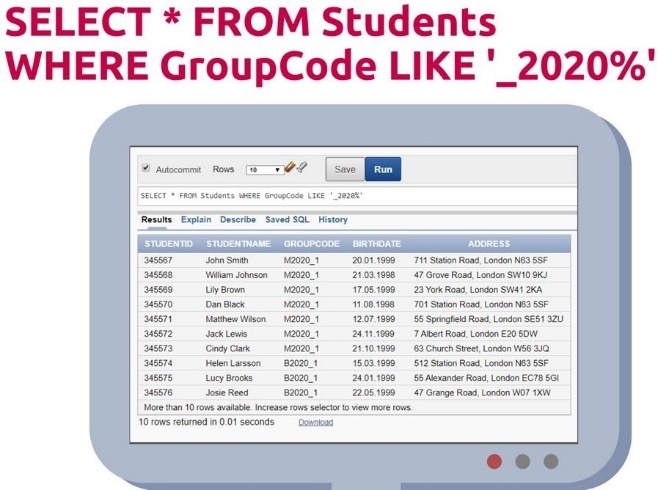
Приведем примеры:

’x%’ – любые строки, которые начинаются с буквы x;

’ab\_’ – строки длиной строго ь символа, причем первыми символами строки должны быть ab;

’%t’ – любая последовательность символов, которая обязательно заканчивается символом t;

’%car%’ – любая последовательность символов, содержащая слово car в любой позиции строки.



Применим поиск с шаблоном для строкового столбца GroupCode таблицы Students. Например, найдем всех студентов, чей код группы начинается с буквы M – в шаблоне после буквы % указывает произвольную последовательность символов:

SELECT \* FROM Students WHERE GroupCode LIKE 'M%'

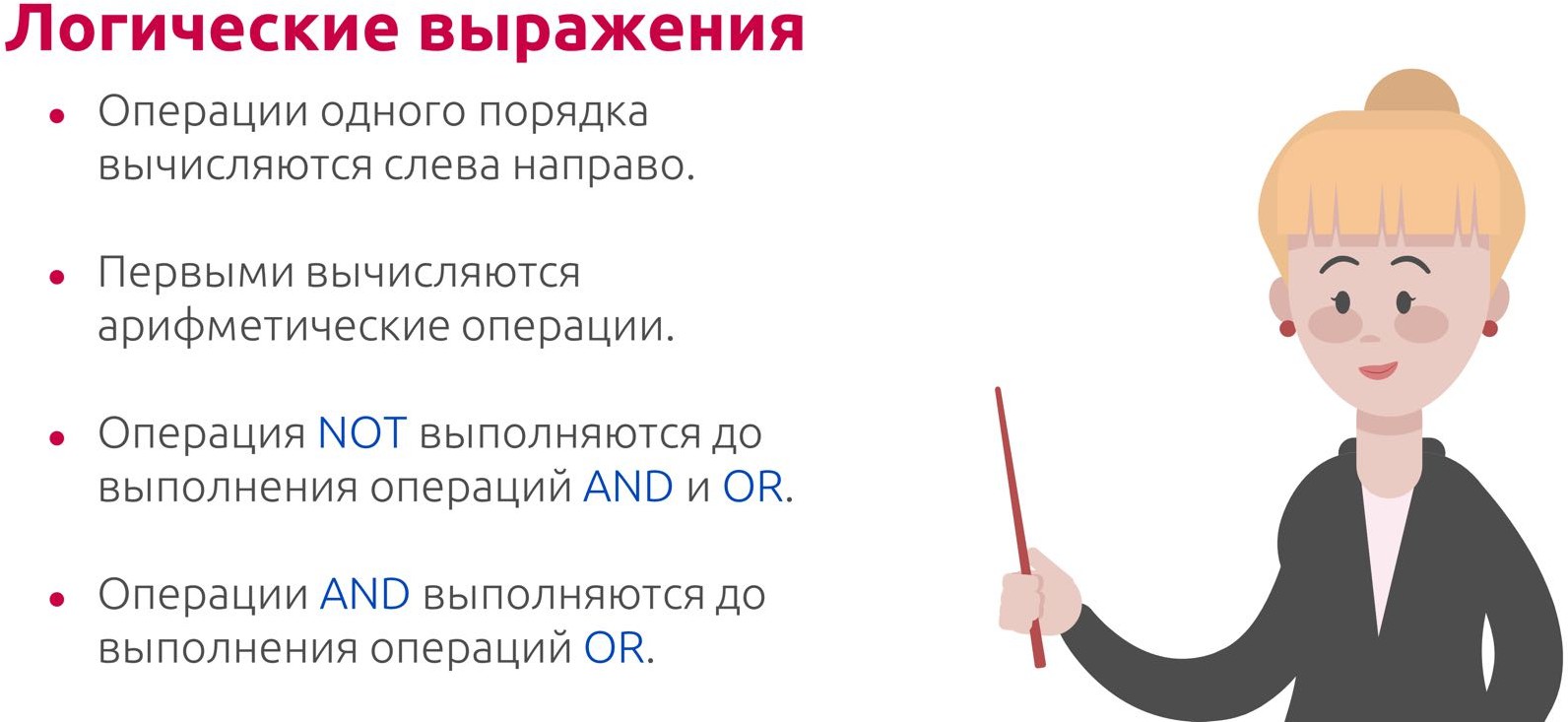
Следующий запрос выведет все группы 2020 года поступления – в шаблоне первый символ может быть любым, затем идут символы 2020, затем произвольная последовательность символов:

SELECT \* FROM Students WHERE GroupCode LIKE '\_2020%'

И еще один запрос с шаблоном – найти всех студентов, чьи фамилии начинаются на букву B. Поле StudentName содержит имена и фамилии студентов. Чтобы искать по шаблону именно фамилии, перед буквой B указан знак пробела, а до и после пробела и буквы B указан знак процента – любая последовательность символов:

SELECT \* FROM Students WHERE StudentName LIKE '% B%'

**Логические выражения**

Пока мы рассматривали только простые условия выборки, когда значения столбца, или выражения сравнивались с константами или значениями других выражений. Но часто требуется проверить сразу несколько критериев. Если нужно проверить несколько условий, то их можно соединять при помощи логических операций: Отрицания, логического умножения и логического сложения: NOT, AND, OR. Также можно использовать скобки. Логические выражения записываются после слова WHERE.

Начнем с самого простого логического выражения – с отрицания NOT. Мы искали строки, которые подходят под заданный шаблон при помощи предложения LIKE. Таким образом мы нашли всех студентов, которые живут на Station Road:

SELECT \* FROM Students WHERE address LIKE '% Station Road%'

Если перед словом LIKE мы укажем операцию отрицания, то запрос вернет всех, кто не живет на Station Road:

SELECT \* FROM Students WHERE address NOT LIKE '% Station Road%'

Теперь составим логическое выражение, соединив два условия: в адресе присутствует подстрока Station Road и дата рождения больше или равна 01/01/1999 (1 января 1999 года). Мы использовали логическую операцию AND, чтобы выполнялись оба указанных условия:

SELECT \* FROM Students

WHERE address LIKE '% Station Road%' AND

BirthDate >= '01/01/1999'

Еще один пример: выберем всех студентов, у которых значение поля StudentId меньше 345574 или больше 345586. Логическая операция OR требует выполнения хотя бы одного из связываемых условий:

SELECT \* FROM Students

WHERE StudentId < 345574 OR StudentId > 345586

# **Агрегатные функции**

Иногда нам нужно не просто выбрать какие-то строки из таблицы, а выполнить некоторую агрегирующую обработку этих строк – найти количество записей, сумму или среднее значение числового столбца, самую раннюю/позднюю дату и пр. Для решения таких задач нам помогают агрегатные функции.

Агрегатная функция — функция, которая выполняет операцию над набором значений столбца (или выражений) и возвращает в качестве результата единственное значение.

Все СУБД поддерживают как минимум пять стандартных агрегатных функций:

COUNT (Выражение) – определяет количество записей;

MIN (Выражение) – наименьшее из множества значений;

MAX (Выражение) – наибольшее из множества значений;

AVG (Выражение) – среднее значение;

SUM (Выражение) – сумма множества значений.

При этом аргументами функции суммы и среднего значения могут быть только числа, т.е. можно использовать либо столбцы с числовыми типами данных, либо числовые выражения, а остальные функции могут иметь в качестве аргументов также значения других типов. У функции COUNT в качестве аргумента может быть знак \*, что означает «все строки».



**Агрегирование для всей таблицы**

Агрегатные функции могут быть применены ко всей таблице, или к отдельным группам записей. Если агрегатная функция применяется ко всей таблице, то ее имя указывают после команды SELECT.

Например, найдем общее количество студентов:

SELECT COUNT(\*) FROM Students

Назовем результат Number\_Of\_Students:

SELECT COUNT(\*) AS Number\_Of\_Students FROM Students

Теперь выполним запрос

SELECT MAX(BirthDate) FROM Students

и таким образом найдем дату рождения самого юного студента. В одном запросе можно сразу использовать несколько функций агрегирования: например, найти сразу и количество студентов, и дату рождения самого юного студента.

А вот использовать в одном запросе вместе с функциями агрегирования значения столбцов без агрегирования нельзя. Например, мы только что нашли дату рождения самого юного студента – хотелось бы узнать, как же его зовут. Но запрос

SELECT StudentsName, MAX(BirthDate) FROM Students

является неправильным. Чтобы узнать, как же зовут самого юного студента, надо написать чуть более сложный запрос.

**Агрегирование для всей таблицы с условием**

Если вместе с функцией агрегирования указать условие выборки строк таблицы, то агрегатная функция будет применяться только к строкам, удовлетворяющим указанному условия.

Например, сосчитаем не всех студентов в таблице, а только тех, которые учатся в группе в группе M2020\_1:

SELECT COUNT(\*) FROM Students WHERE GroupCode='M2020\_1'

Или сосчитаем студентов сразу из двух групп: ’B2020\_1’ и ’B2020\_2’:

SELECT COUNT(\*) FROM Students

WHERE GroupCode='B2020\_1' OR GroupCode='B2020\_2'

При подсчете строк аргументов функции COUNT может быть не только знак \*, можно подсчитывать количество значений отдельных столбцов. Например, сосчитаем количество групп:

SELECT COUNT(GroupCode) FROM Students

Их оказывается столько же, сколько строк в таблице Students:

SELECT GroupCode FROM Students



Функция агрегирования считает не количество уникальных значений заданного столбца, а общее количество значений. Можно ли сосчитать количество разных групп? Конечно, тут нам поможет слово DISTINCT. Мы уже использовали его, чтобы вывести только уникальные значение групп из таблицы Students:

SELECT DISTINCT GroupCode FROM Students

С помощью этого ключевого слова мы можем и сосчитать количество уникальных групп:

SELECT COUNT(DISTINCT GroupCode) FROM Students

**Агрегирование с группировкой**

Мы нашли количество студентов из одной группы, другой, третьей. Но если групп окажется много, то неужели придется считать для каждой группы отдельно, задавая ее код в качестве условия? Есть способ проще, он называется группировкой. В качестве критерия можно указать столбец (или несколько столбцов), и те строки, в которых значения указанного критерия совпадут, попадут в одну группу.

Например, сгруппируем таблицу Students по значению поля GroupCode. Теперь функции агрегирования, указанные после слова SELECT, будут выполняться отдельно для каждой группы.

SELECT COUNT(\*) FROM Students GROUP BY GroupCode

Теперь мы можем узнать количество студентов в каждой группе отдельно. В качестве результата запроса мы получим столбец цифр, каждая из которых соответствует значению функции агрегирования для каждой группы. Только как теперь узнать, какая цифра соответствует какой группе?

Для этого надо указать вместе с функцией агрегирования название того столбца, по которому мы производили группировку:

SELECT GroupCode, COUNT(\*) FROM Students GROUP BY GroupCode

Только при группировке названия столбцов можно и нужно помещать после слова SELECT вместе с функциями агрегирования. Но названия других столбцов, по которым группировка не производилась, не могут быть использованы в запросе.

При использовании группировки можно использовать оператор HAVING – он позволяет производить фильтрацию групп. После конструкции HAVING можно писать агрегатные функции и задавать ограничения на значения столбцов, по которым проводится группировка.

Предположим, нам не нужно знать результат агрегирующей функции для каждой группы, а надо выводить только те группы, в которых результат агрегатной функции удовлетворяет заданному условию. Например, мы нашли количество студентов в каждой группе и вывели это количество вместе с номерами групп. Теперь выведем только те номера групп, в которых студентов меньше 8:

SELECT GroupCode FROM Students GROUP BY GroupCode

HAVING COUNT(\*) < 8

Следующий пример выводит номера групп и количество студентов в них, если в группе более ь студентов и название группы начинается с символа B.

SELECT GroupCode, COUNT(\*) FROM Students GROUP BY GroupCode

HAVING COUNT(\*) > ь AND GroupCode LIKE 'B%'

Можно ли после HAVING ограничивать не результат функции агрегирования, а значения отдельных столбцов? Например, написать ограничение на количество студентов и ограничение на значение поля StudentID < 345577?

SELECT GroupCode FROM Students GROUP BY GroupCode

HAVING COUNT(\*) < 8 AND StudentID < 345577

Так писать неправильно.

Если нужно выбрать строки, соответствующие определенному критерию, то нужно использовать для этого условие после ключевого слова WHERE. Тогда сначала из таблицы будут выбраны строки, для которых выполнено условие, затем эти строки будут сгруппированы, для каждой группы будет вычислена функция агрегирования, указанная после HAVING, и проверен ее результат:

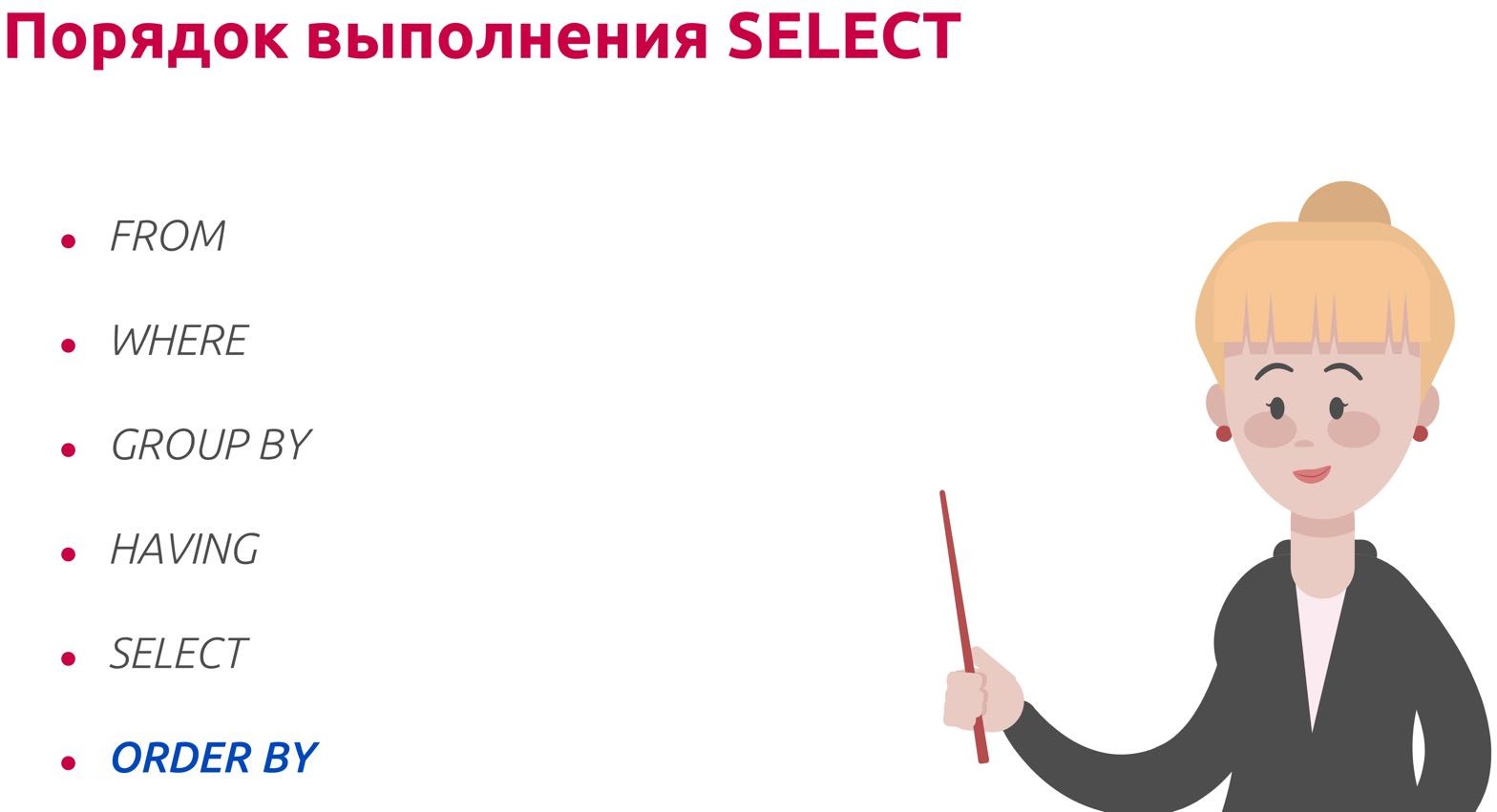
SELECT GroupCode FROM Students WHERE StudentID < 345577

GROUP BY GroupCode HAVING COUNT(\*) < 8

Можно применять сортировку к агрегированию с группировкой. Например, отсортируем список групп по убыванию, начав с самой многочисленной группы.

# **Вложенные запросы**

Давайте еще раз вернемся к описанию оператора SELECT, чтобы разобраться в порядке выполнения. Сначала идет обращение к таблице, указанной после предложения FROM. Затем из таблицы выбираются строки, удовлетворяющие условию, указанному после слова WHERE. После этого строки группируются согласно критерию после GROUP BY. Затем группы фильтруются согласно условию HAVING. После этого подсчитываются итоговые агрегатные функции или выбираются столбцы таблицы и, наконец, итоговая выборка сортируется в указанном порядке.

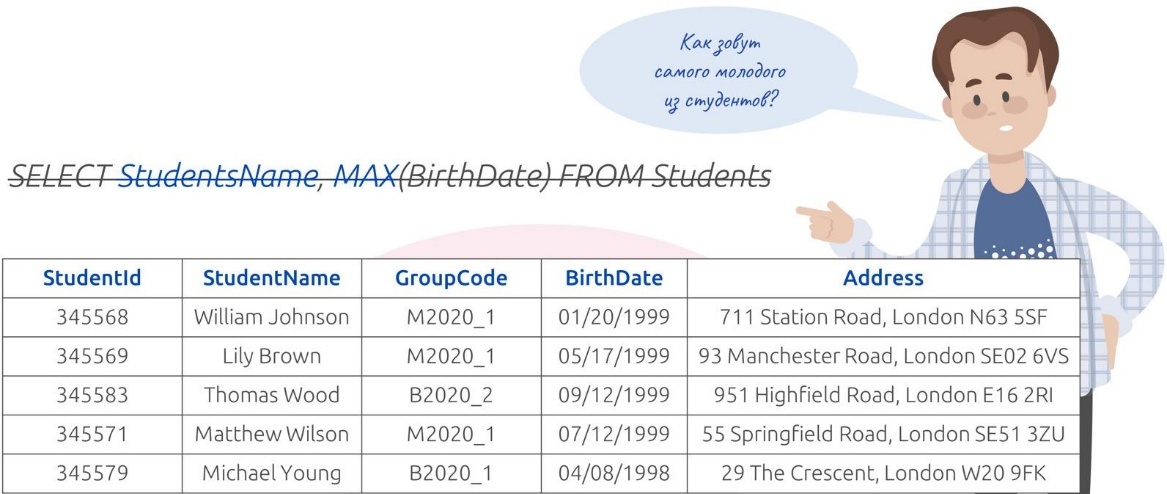


Запросы могут содержать в себе вложенные подзапросы. Вложенные запросы могут быть использованы после слова SELECT, вместо имени таблицы в запросе, и после слов WHERE и HAVING. Вложенные запросы нужны, если требуется соединить или сопоставить данные из нескольких таблиц. Вложенный запрос всегда заключается в скобки.

Вложенные запросы могут возвращать одно значение, а могут в качестве результата иметь целую таблицу. Бывают вложенные запросы, которые называют коррелированными, если они ссылаются на столбцы таблицы внешнего запроса.

Вспомним запрос, которым мы находили самого юного студента:

SELECT MAX(BirthDate) FROM Students

Теперь нужно узнать, как же зовут этого студента. Надеюсь, вы помните, что использовать рядом функции агрегирования и столбцы таблицы без группировки нельзя.

Нам поможет вложенный запрос. Мы найдем дату рождения самого юного студента, а затем выберем из таблицы тех (или того) студента, чья дата рождения совпала с найденным максимумом. Мы сравнивали значение столбца BirthDate с результатом вложенного запроса при помощи равенства, т.к. были уверены, что результатом вложенного запроса будет единственное скалярное значение.

SELECT StudentName FROM Students

WHERE BirthDate = (SELECT MAX(BirthDate) FROM Students)

Если результат вложенного запроса был бы множеством значений, итоговый запрос не мог бы быть выполнен.

Рассмотрим еще один пример. Предположим, нам нужно найти студентов, у которых специализация Nanotechnology (Нанотехнология). Для этого из таблицы St\_Group найдем значение GroupCode:

SELECT GroupCode FROM St\_Group

WHERE Specialization = 'Nanotechnology'

Теперь из таблицы Students найдем студентов по заданному коду группы:

SELECT StudentName FROM Students WHERE GroupCode='M2020\_1'

Мы написали два простых запроса, а на самом деле, их можно объединить в один. Подставим первый запрос вместо значения поля GroupCode во втором запросе.

SELECT StudentName FROM Students

WHERE GroupCode = (SELECT GroupCode FROM St\_Group WHERE Specialization = 'Nanotechnology')

Пока мы пользовались тем, что вложенный запрос возвращает единичный результат. Давайте сделаем так, чтобы в результате первого запроса находилось несколько значений. Найдем студентов, у которых специализация Нанотехнология или Rдравоохранение. Снова напишем запрос к таблице St\_Group, и теперь этот запрос вернет два значения поля GroupCode:

SELECT GroupCode FROM St\_Group

WHERE Specialization = 'Nanotechnology' OR Specialization = 'Health Research'

Этот запрос можно переписать через принадлежность множеству, где в качестве элементов множества выступают названия специализаций.

SELECT GroupCode FROM St\_Group

WHERE Specialization IN ('Nanotechnology', 'Health Research')

Элементы множества можно задавать перечислением констант, а можно формировать из результатов вложенного запроса. Напишем запрос, который найдет фамилии студентов специализации Nanotechnology и Health Research при помощи вложенного запроса:

SELECT StudentName FROM Students

WHERE GroupCode IN (SELECT GroupCode FROM St\_Group

WHERE Specialization = 'Nanotechnology' OR Specialization = 'Health Research')

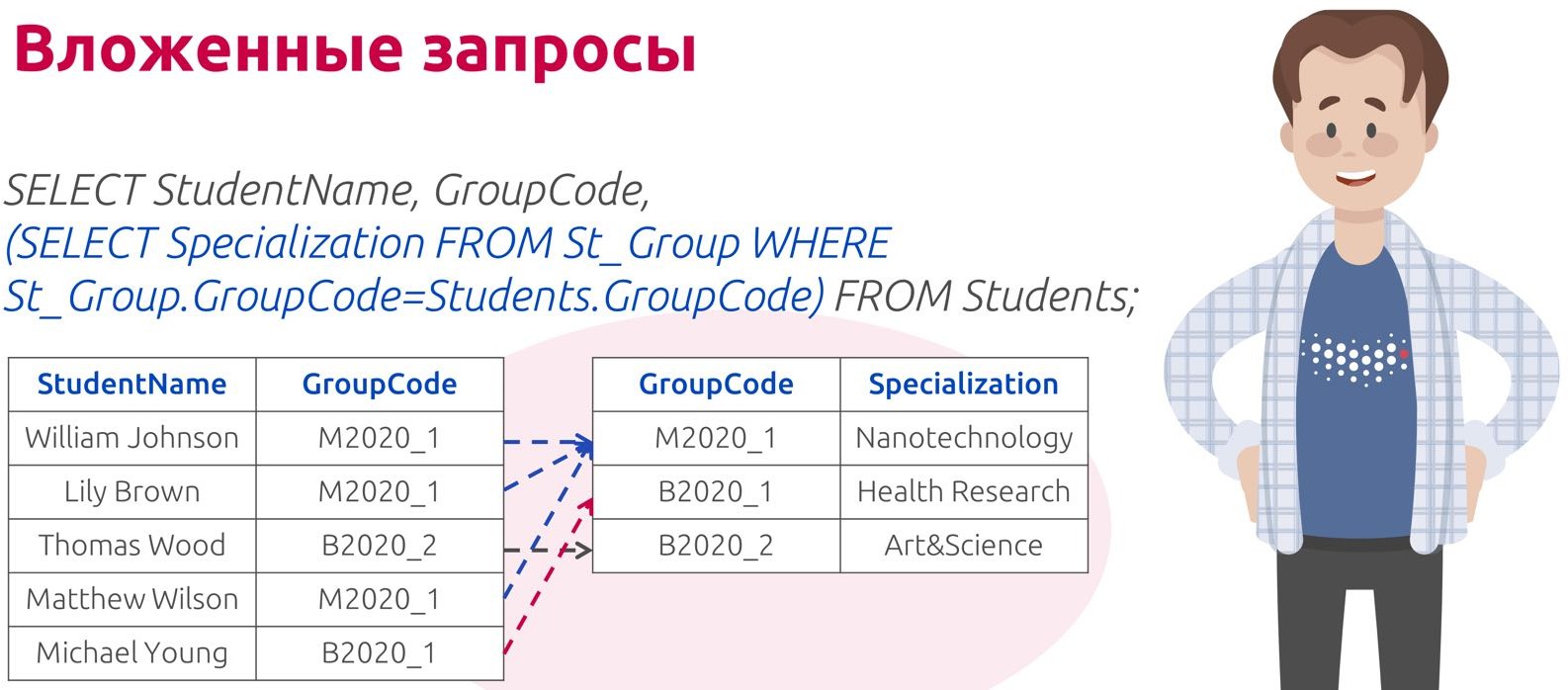
Вложенный запрос можно использовать и после предложения SELECT, вместе с названиями столбцов, но только в том случае, если он возвращает ровно одно значение.

Например, выберем список имен студентов и коды групп, где они учатся:

SELECT StudentsName, GroupCode FROM Students

Теперь добавим столбец со специализацией группы. Такого столбца нет в таблице Students, зато он есть в таблице St\_Group. При этом мы знаем, что для каждого кода группы есть ровно одна специализация.

SELECT GroupCode, Specialization FROM St\_Group

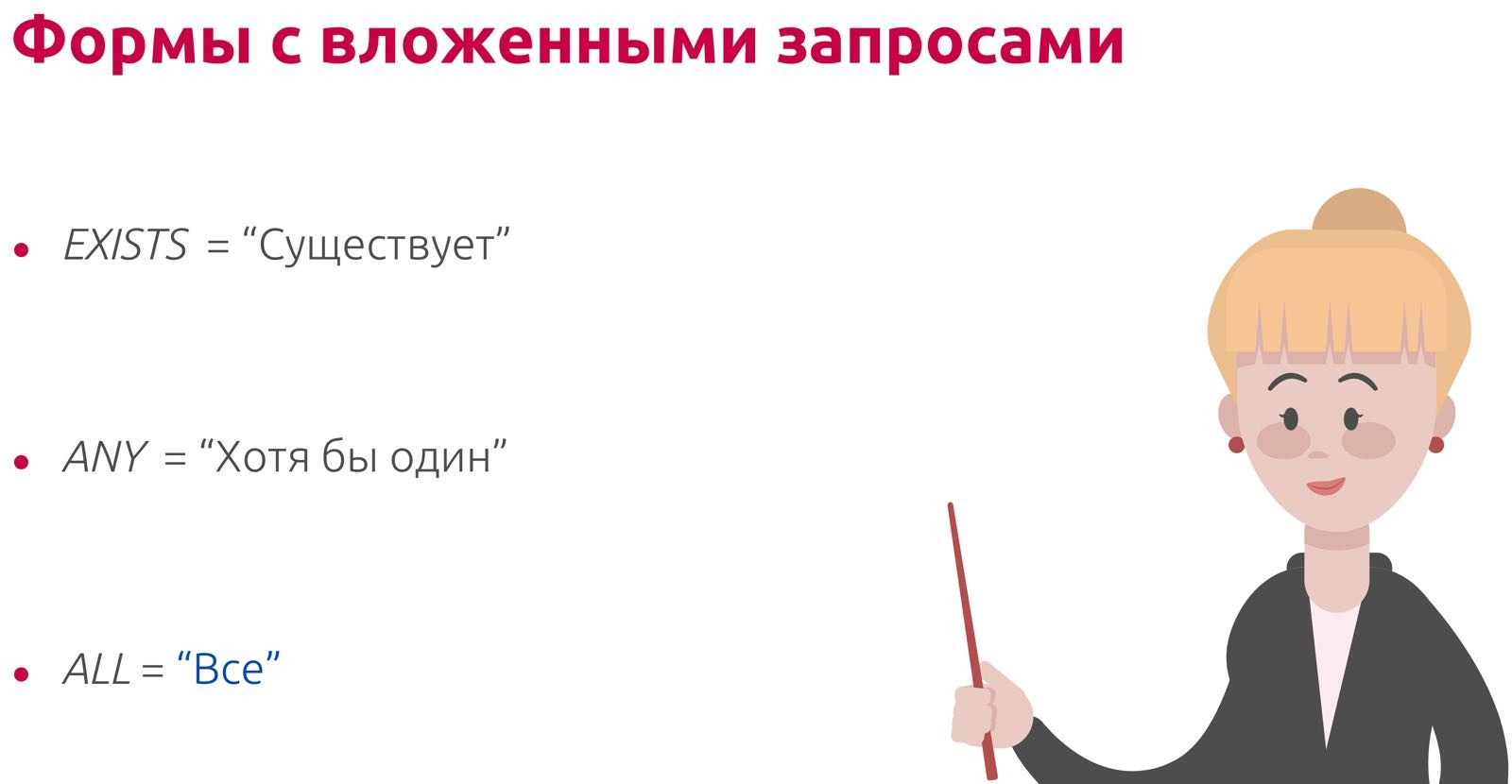
Тогда мы можем добавить для вывода специализации студента вложенный запрос, который возвращает поле специализация из таблицы St\_Group, сравнивая поле GroupCode из таблицы St\_Group со значением поля GroupCode студента:

SELECT StudentName, GroupCode, (SELECT Specialization FROM St\_Group

WHERE St\_Group.GroupCode=Students.GroupCode) FROM Students

Вы заметили, что в таблице участвуют две таблицы, и в обеих таблицах есть поле GroupCode, которое мы используем в запросе. Надо как-то показать, про какое поле идет речь. Для этого перед названием поля указывают имя таблицы. Теперь наш вложенный запрос стал коррелированным, так он сам находит значения из таблицы St\_Group, но использует внутри значение поля из таблицы Students, которое берется из внешнего запроса.

При использовании вложенных запросов можно применять операторы EXISTS, ANY и ALL, которые соответственно означают «Существует», «Хотя бы один» и «Все».



Их можно использовать следующим образом. Оператор EXISTS указывают после слова WHERE, за ним следует вложенный запрос, обычно коррелируемый. Оператор EXISTS возвращает значение истина, если существует хоть одна строка вложенного запроса.

SELECT \* FROM T

WHERE EXISTS (вложенный запрос)

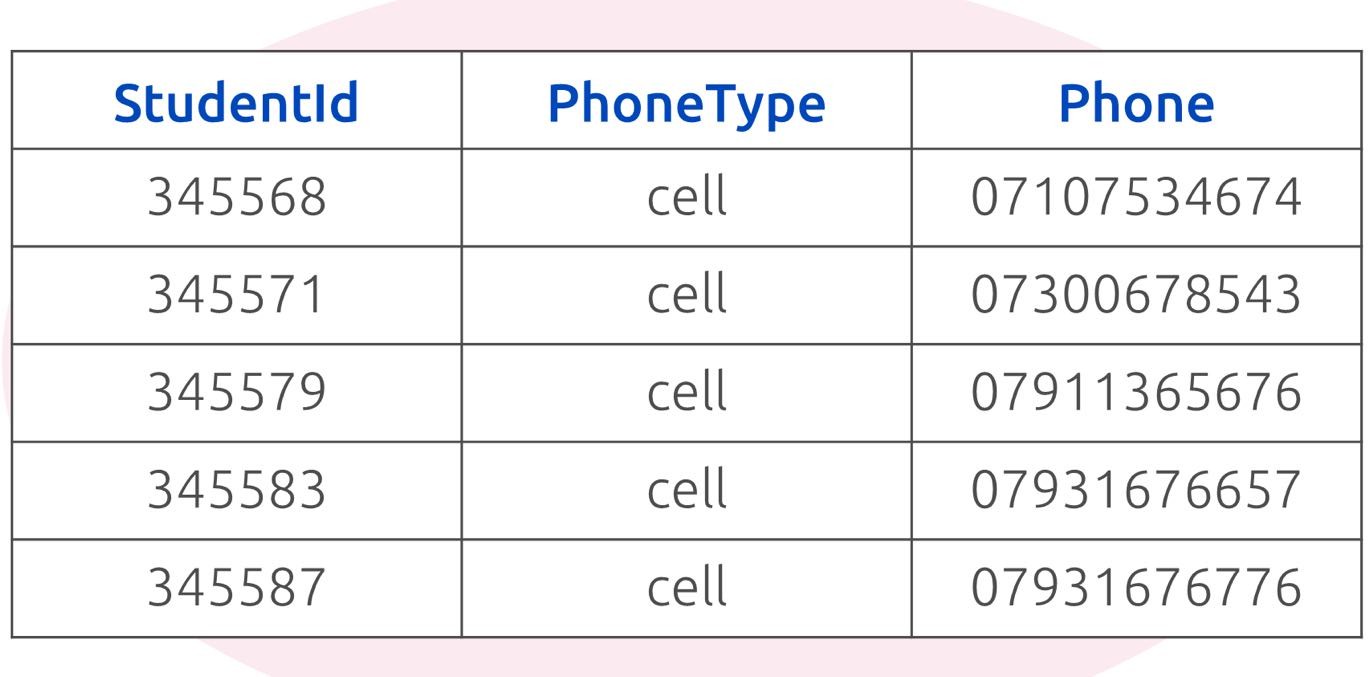
Например, найдем студентов, у которых в таблице PHONE\_LIST есть хоть один телефон. Напомним, что у каждого студента может быть несколько телефонов, а может не быть ни одного – между сущностями Студент и Телефон связь вида многие-ко-многим, модальность «может». В таблице PHONE\_LIST вместе с каждым номером телефона хранится номер зачетки студента, который показывает, кому принадлежит телефон. Во внешнем запросе будем искать номера зачеток и фамилии студентов, а во внутреннем – проверять, что для текущего номера зачетки есть хоть одна запись в таблице телефонов.

SELECT StudentID, StudentName FROM Students WHERE EXISTS (SELECT \* FROM PHONE\_LIST WHERE

Students.StudentID=PHONE\_LIST.StudentID)

В результате выполнения запроса мы видим номера зачеток и имена студентов, у которых есть хоть один телефон.

Если перед оператором EXISTS поставить NOT, то результирующая вы-

борка вернет нам тех, у кого не указано ни одного телефона. Теперь внешний запрос вернет все строки таблицы Students, для которых не существуют ни одной записи в таблице PHONE\_LIST:

SELECT StudentID, StudentName FROM Students WHERE NOT EXISTS (SELECT \* FROM PHONE\_LIST WHERE Students.StudentID=PHONE\_LIST.StudentID)

Оператор ALL, ANY можно использовать, если вложенный запрос возвращает один столбец, значения которого сравниваются с заданной скалярной величиной.



Например, найдем студентов, у которых есть хоть одна двойка за экзамен. Оценки студентов хранятся в таблице EXAM\_RESULT, и вложенный запрос возвращает все оценки для каждого студента из таблицы EXAM\_RESULT, а внешний запрос выбирает все записи из таблицы Students, для которых существует оценка, равная 2.

SELECT StudentID, StudentName FROM Students

WHERE 2 = ANY (SELECT Grade FROM EXAM\_RESULT WHERE

Students.StudentID=EXAM\_RESULT.StudentID)

При помощи оператора ALL найдем студентов, у которых оценки только 4 и 5. Внутренний запрос по-прежнему возвращает все оценки для каждого студента из таблицы EXAM\_RESULT, а внешний запрос находит те строки, для которых все оценки >3.

SELECT StudentID, StudentName FROM Students

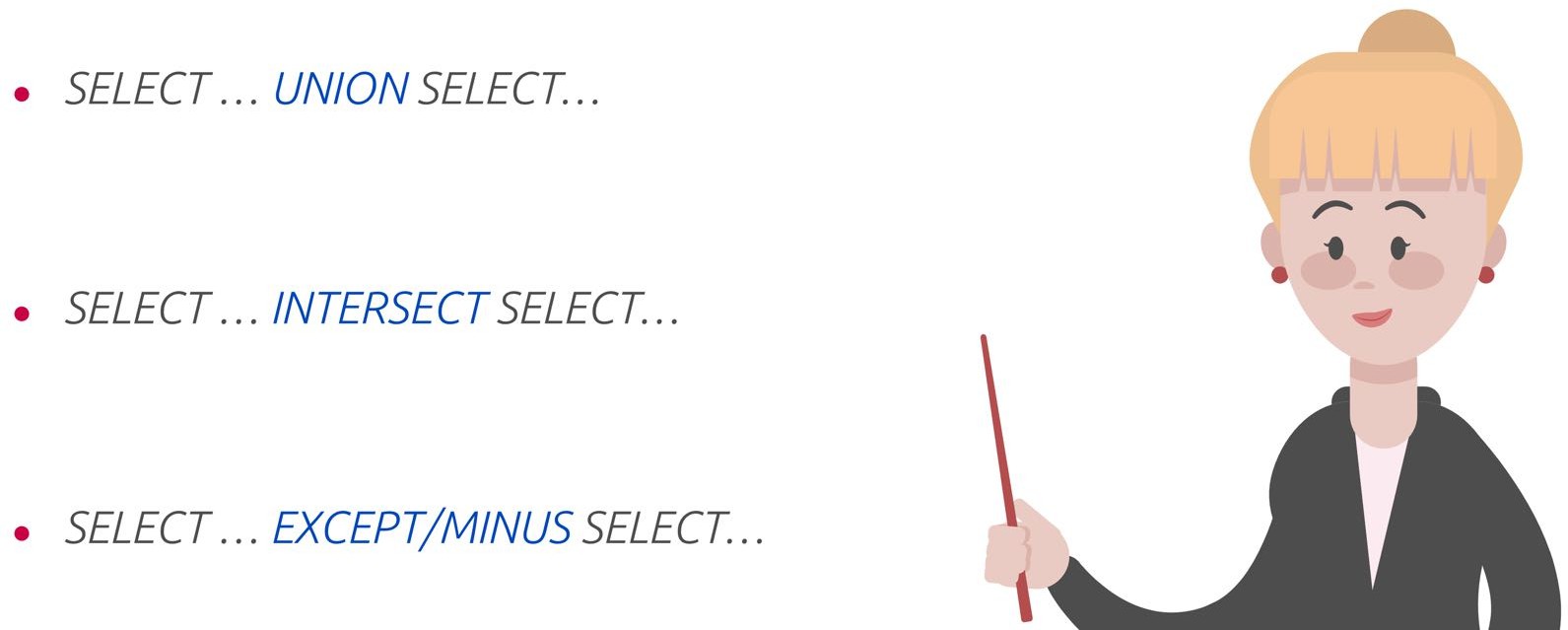
WHERE 3 < ALL (SELECT Grade FROM EXAM\_RESULT WHERE

Students.StudentID=EXAM\_RESULT.StudentID)

# **Теоретико-множественные операции**

Строки в реляционных таблицах можно рассматривать как множества, а с множествами можно производить теоретико-множественные операции: объединение, пересечение, разность, декартово произведение. С их помощью можно объединить результаты двух запросов, находить общую часть – строки, которые есть в обеих выборках, находить разность – строки, которые есть в одной выборке, но нет во второй, а также сопоставлять строки нескольких таблиц.

Для того, чтобы с выборками можно было производить эти операции, они должны быть совместимы по типу: в них должно быть одинаковое количество столбцов, и столбцы с одинаковым порядковым номером должны иметь совместимые типы данных. Какие типы данных считаются совместимыми?



Когда легко можно сделать преобразование одного типа в другой. Например, любое целое число можно сделать дробным, короткую строку длинной и т.д.

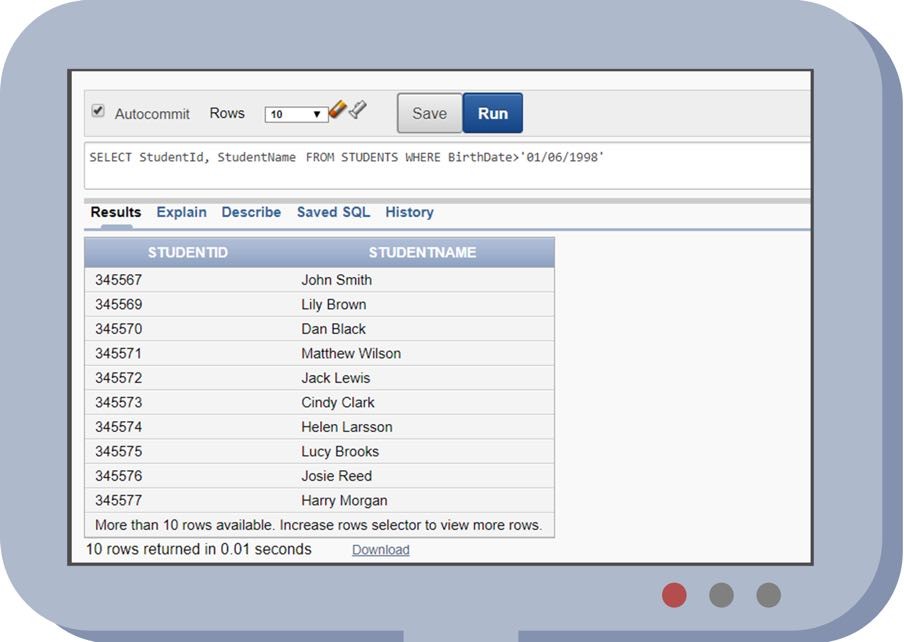
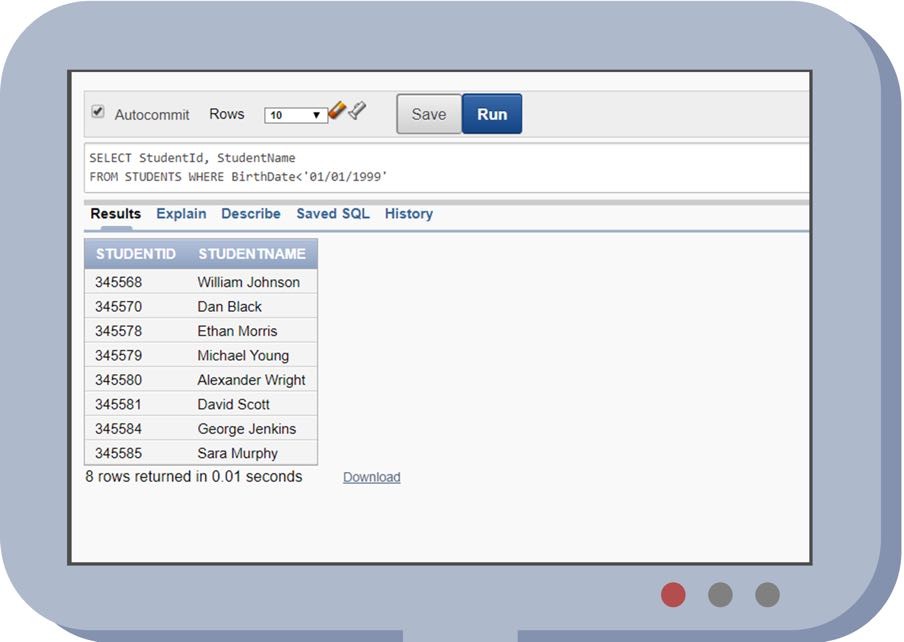
А вот, например, строку в число преобразовать нельзя. Так что типы столбцов очень важны для этих операций. А вот названия полей не обязательно должны совпадать.

Подготовим две выборки. Например, первый запрос возвращает нам поля StudentId и StudentName из таблицы Students со значением BirthDate < ’01/01/1999’:

SELECT StudentId, StudentName FROM STUDENTS WHERE BirthDate < '01/01/1999'

Другой запрос вернет нам тех, чьи даты рождения BirthDate > ’06/01/1998’:

SELECT StudentId, StudentName FROM STUDENTS WHERE BirthDate > '06/01/1998'

В наших выборках по два столбца – первый числовой и второй строковой. На этих данных продемонстрируем операции объединения, пересечения и разности.

Начнем с операции объединения. Для этого нужно соединить две выборки оператором UNION:

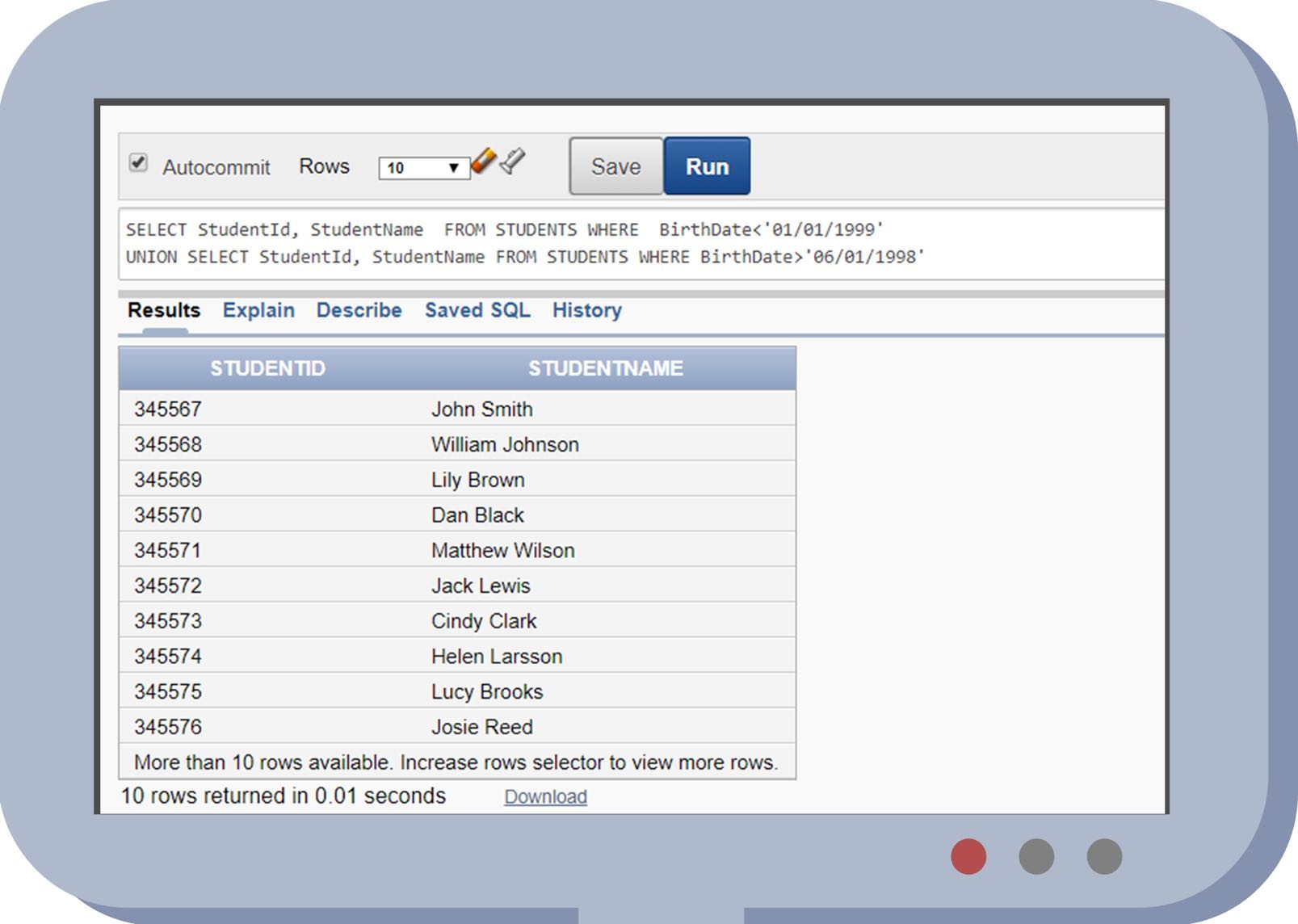
SELECT StudentId, StudentName FROM STUDENTS WHERE BirthDate < '01/01/1999'

UNION

SELECT StudentId, StudentName FROM STUDENTS WHERE BirthDate > '06/01/1998'

Заметим, что в исходных выборках были совпадающие записи – в результирующий набор они вошли без дублирования.

Если нужно включить в результирующий набор все строки, не взирая на повторение, то надо написать вместо UNION оператор UNION ALL:



SELECT StudentId, StudentName FROM STUDENTS WHERE BirthDate < '01/01/1999'

UNION ALL

SELECT StudentId, StudentName FROM STUDENTS WHERE BirthDate > '06/01/1998'

Теперь найдем пересечение двух выборок при помощи оператора INTERSECT:

SELECT StudentId, StudentName FROM STUDENTS WHERE BirthDate < '01/01/1999'

INTERSECT

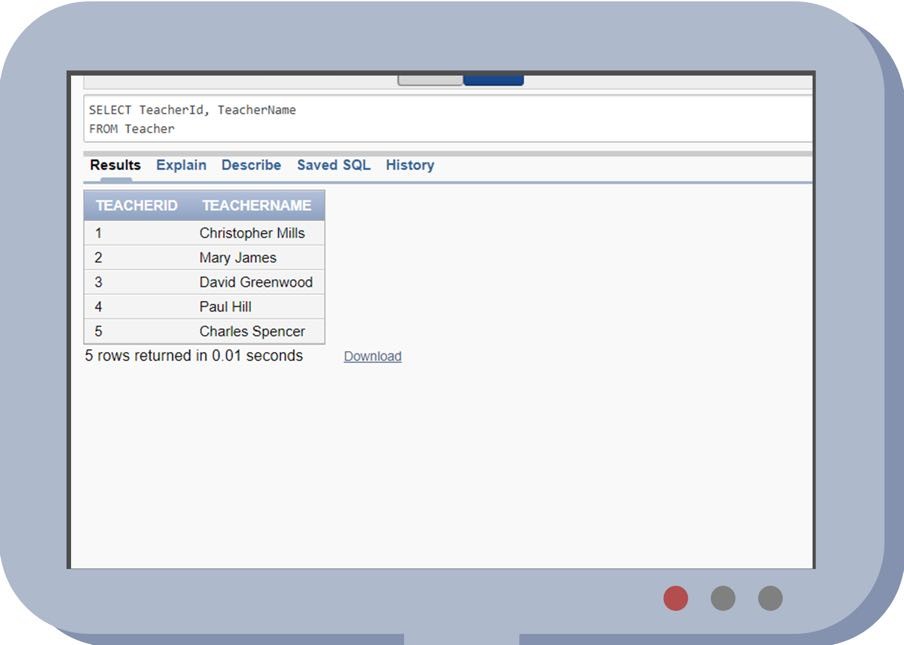
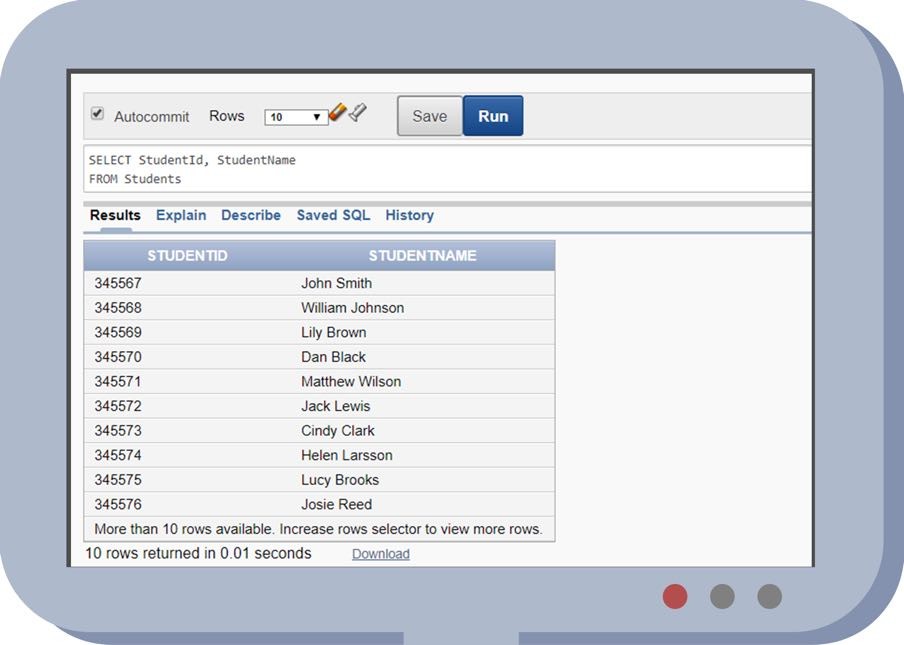
SELECT StudentId, StudentName FROM STUDENTS WHERE BirthDate > '06/01/1998'

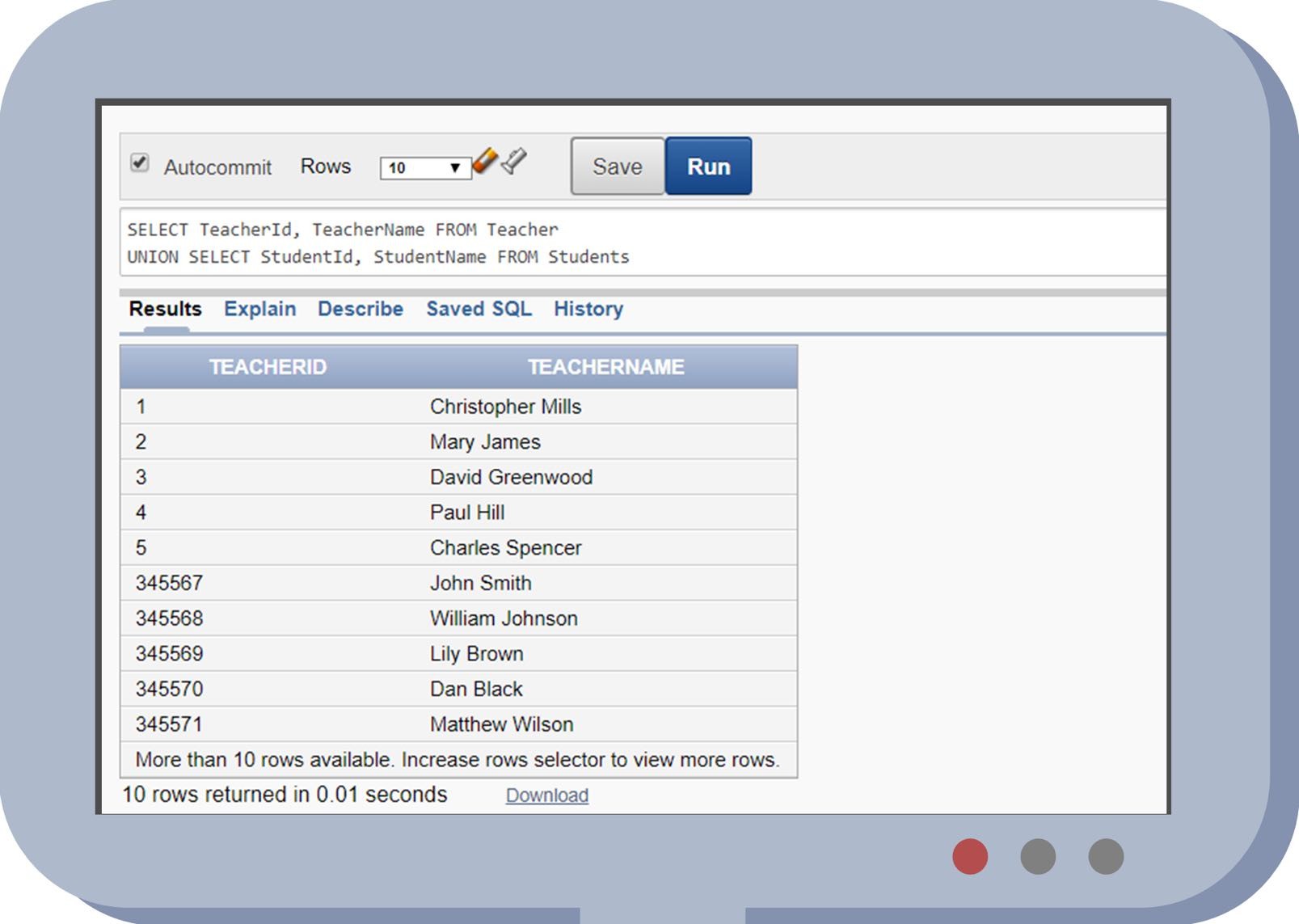
В результат пересечения попали четыре строки – это те строки, где дата рождения от 1 июня 1998 года до 1 января 1999.

Строки, которые есть в первой выборке, но не присутствуют во второй, можно найти при помощи оператора разности, который обозначается в некоторых системах словом EXCEPT, в некоторых – словом MINUS:

SELECT StudentId, StudentName FROM STUDENTS WHERE BirthDate < '01/01/1999'

MINUS

SELECT StudentId, StudentName FROM STUDENTS WHERE BirthDate > '06/01/1998'

Приведем еще один пример объединения. Сделаем выборку из таблицы Teachers: выберем поля TeacherId, TeacherName. Можно ли эту выборку объединить с выборкой из таблицы Students? В каждой из выборок два столбца, первые столбцы целочисленные, вторые строковые. Правда, на имя студента отведено 100 символов, а на имя преподавателя всего 50, но тем не менее эти типы данных совместимы.

SELECT TeacherId, TeacherName FROM Teacher SELECT StudentId, StudentName FROM Students

В результате объединения получится таблица, столбцы которой будут

называться так же, как и в первой из объединяемых таблиц, а длина строкового поля станет равна максимуму из длин соответствующих полей.

SELECT TeacherId, TeacherName FROM Teacher UNION

SELECT StudentId, StudentName FROM Students

Мы видим, что первый столбец выборки называется TeacherId, а второй – TeacherName, что не совсем корректно. Переименуем столбцы объединенной выборки в Id и Name:

SELECT TeacherId AS Id, TeacherName AS Name FROM Teacher

UNION

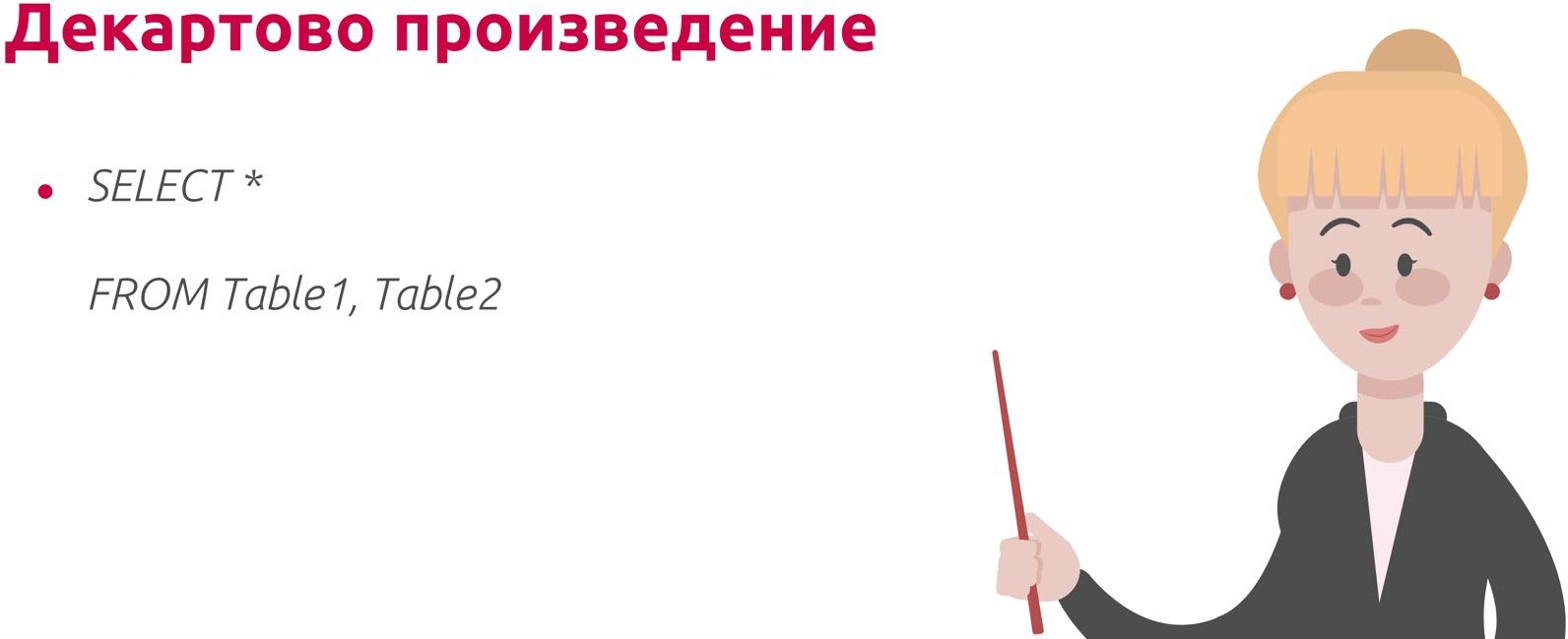
SELECT StudentId AS Id, StudentName AS Name FROM Students

Теперь первый столбец выборки называется Id, а второй – Name.

Чтобы не путать, кто есть кто, можно добавить в итоговую выборку еще один столбец – Status. Для преподавателей сделаем статус Teacher, а для студентов – Student:

SELECT

TeacherId AS Id, TeacherName AS Name, 'Teacher' AS Status FROM Teacher UNION

SELECT StudentId AS Id, StudentName AS Name, 'Student' AS Status FROM Students

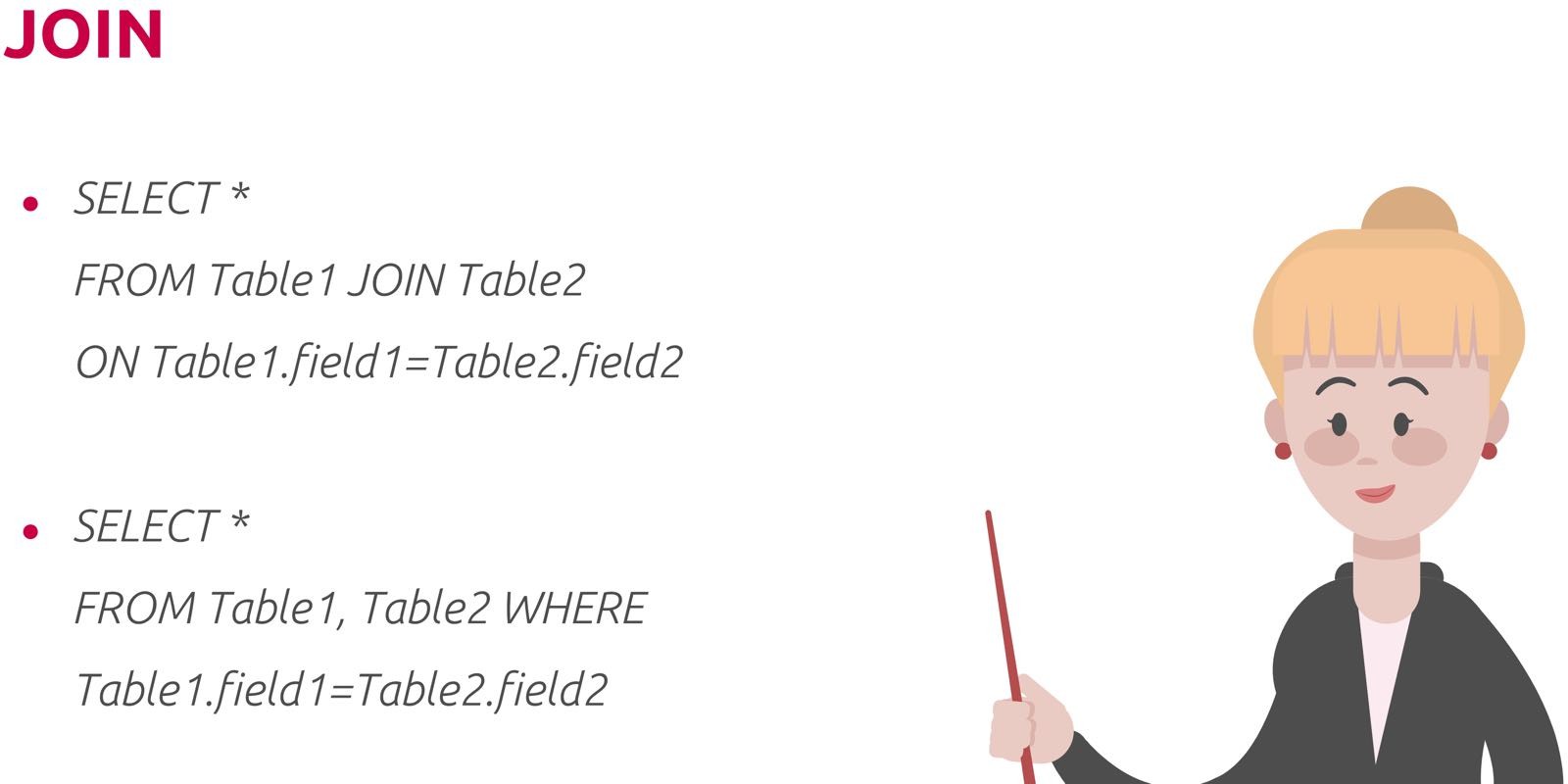
Последняя теоретико-множественная операция, которую мы рассмотрим – это декартово произведение. Если сделать декартово произведение двух таблиц, то получится множество, состоящее из всех возможных пар сочетаний, в котором первый элемент пары – строка первой таблицы, а второй элемент пары – строка из второй таблицы. Это очень емкая операция, количество строк итоговой выборки будет равно произведению количества строк первой и второй таблиц. Чтобы выполнить эту операцию, нужно после ключевого слова FROM через запятую перечислить таблицы, с которыми нужно выполнить декартово произведение.

Приведем пример декартова произведения, перемножив таблицу с учебными группами и таблицу с курсами. В итоге мы получим таблицу, в которой каждой группе будут сопоставлены все курсы, и, наоборот, каждому курсу все группы. Выберем из результата произведения три столбца: код группы, специализацию группы и название курса:

SELECT GroupCode, Specialization, CourseTitle FROM ST\_GROUP, COURSE

Итак, мы рассмотрели основные теоретико-множественные операции, которые часто используются при работе с базами данных.

# **Соединение таблиц**

Давайте вспомним, что таблицы служат не только для хранения информации об объектах, но и для реализации связей. Например, мы храним телефоны студентов в отдельной таблице. Чтобы сохранить связь, мы храним вместе с телефонным номером ключ от объекта Студент – номер зачетки, поле StudentId. Как узнать все телефонные номера, принадлежащие студенту? Надо взять его номер зачетки, и найти все записи из таблицы PHONE\_LIST, которые соответствуют этому номеру зачетки. Можно воспользоваться другим способом – соединить таблицы Students и PHONE\_LIST по полю StudentId.

Для этого существует специальная операция – JOIN. В общем виде она выглядит так: после FROM в операторе SELECT сначала указывают имя первой таблицы, затем ключевое слово JOIN, затем имя второй таблицы, затем ключевое слово ON, после которого указывают критерии соединения. Для соединения таблиц есть и другая форма записи – через декартово произведение с последующим условием.

Давайте соединим таблицу Students и таблицу PHONE\_LIST по совпадающим значениям поля StudentId. Обратите внимания, что это поле встречается в обеих таблицах, поэтому в критерии соединения надо указывать имя таблиц перед именем этого поля:

SELECT \* FROM Students JOIN PHONE\_LIST

ON Students.StudentId = PHONE\_LIST.StudentId

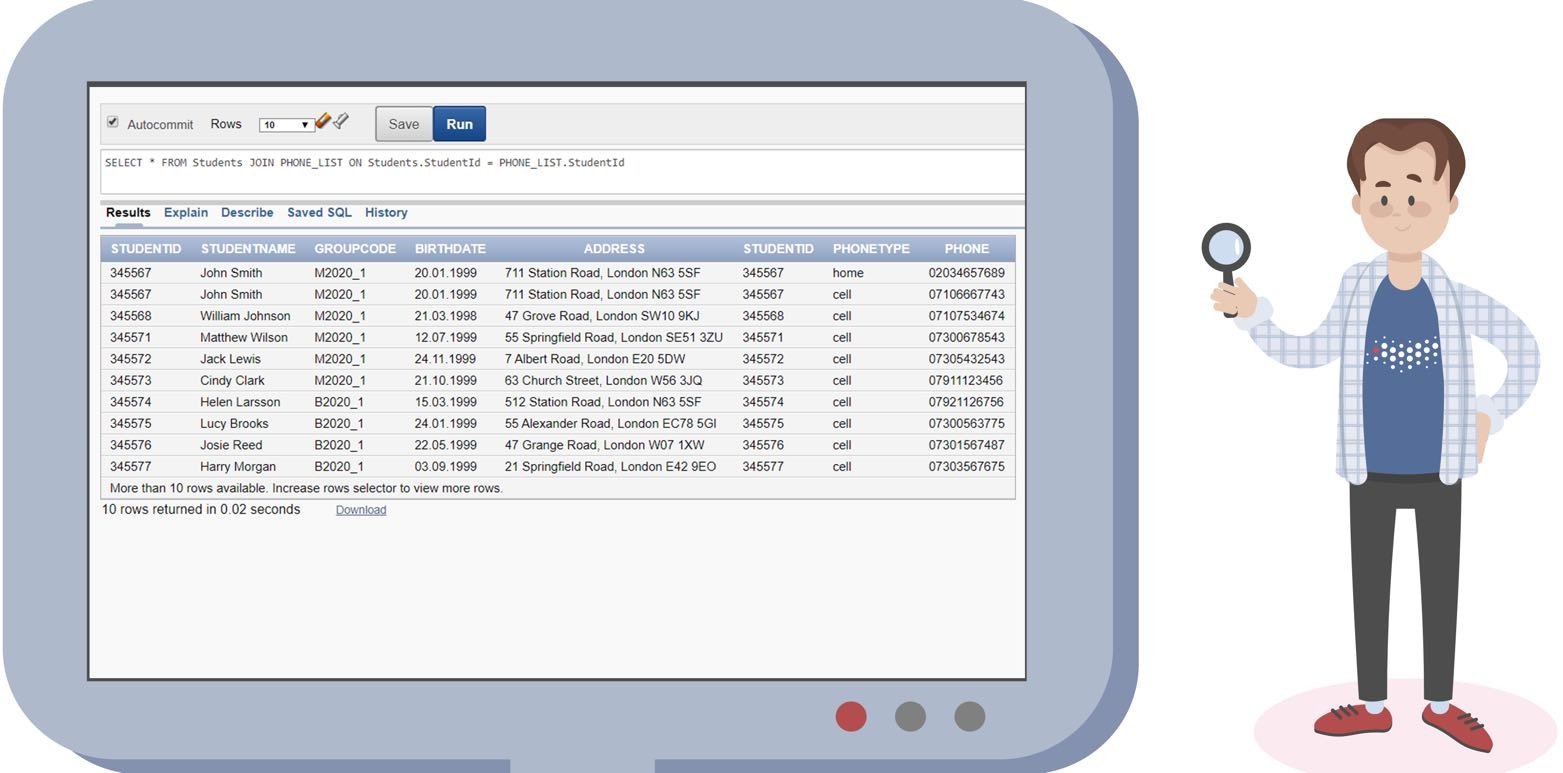
Приведем тот же запрос в другой форме записи – через декартово произведение с последующим условием:

SELECT \* FROM Students, PHONE\_LIST

WHERE Students.StudentId = PHONE\_LIST.StudentId

Теперь мы видим полную информацию о студентах и об их телефонах. Заметим, что в выборку не попали данные о студентах, у которых нет телефона. Например, мы не видим в итоговой выборке Лили Браун с номером зачетки 345569.

Обычное соединение называют еще «внутренним», т.к. в выборку попадают только строки, содержащие значения, используемые в условии соединения, которые находятся «внутри» как первой, так и второй таблицы.

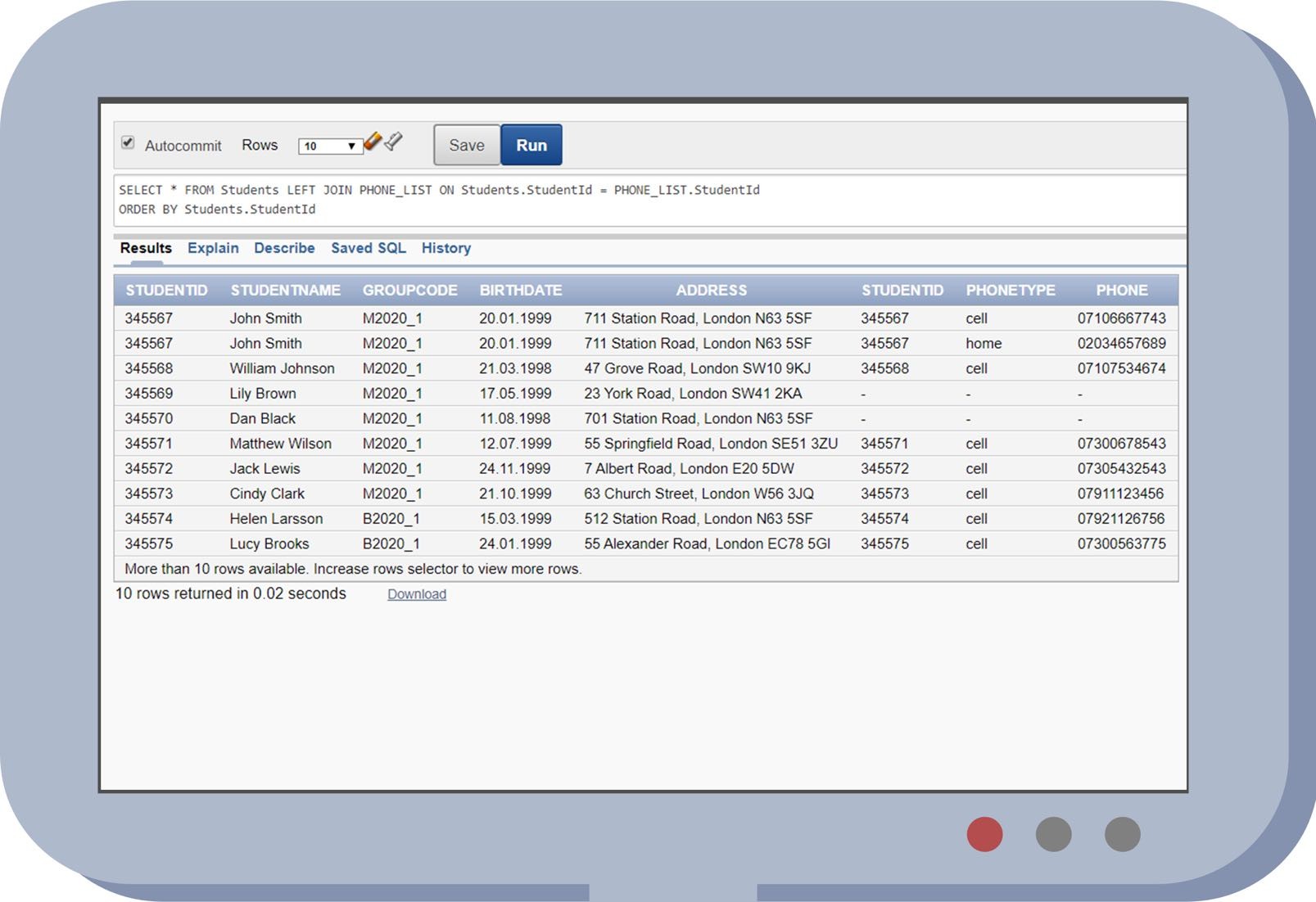


Если нужно включить в выборку всех студентов, несмотря на отсутствие телефонного номера, можно использовать левое соединение – LEFT JOIN. В левое соединение включаются все строки первой таблицы. Если их удалось соединить со строками второй таблицы, то они дополняются соответствующими значениями, а если не удалось – дополняются незаданными значениями.

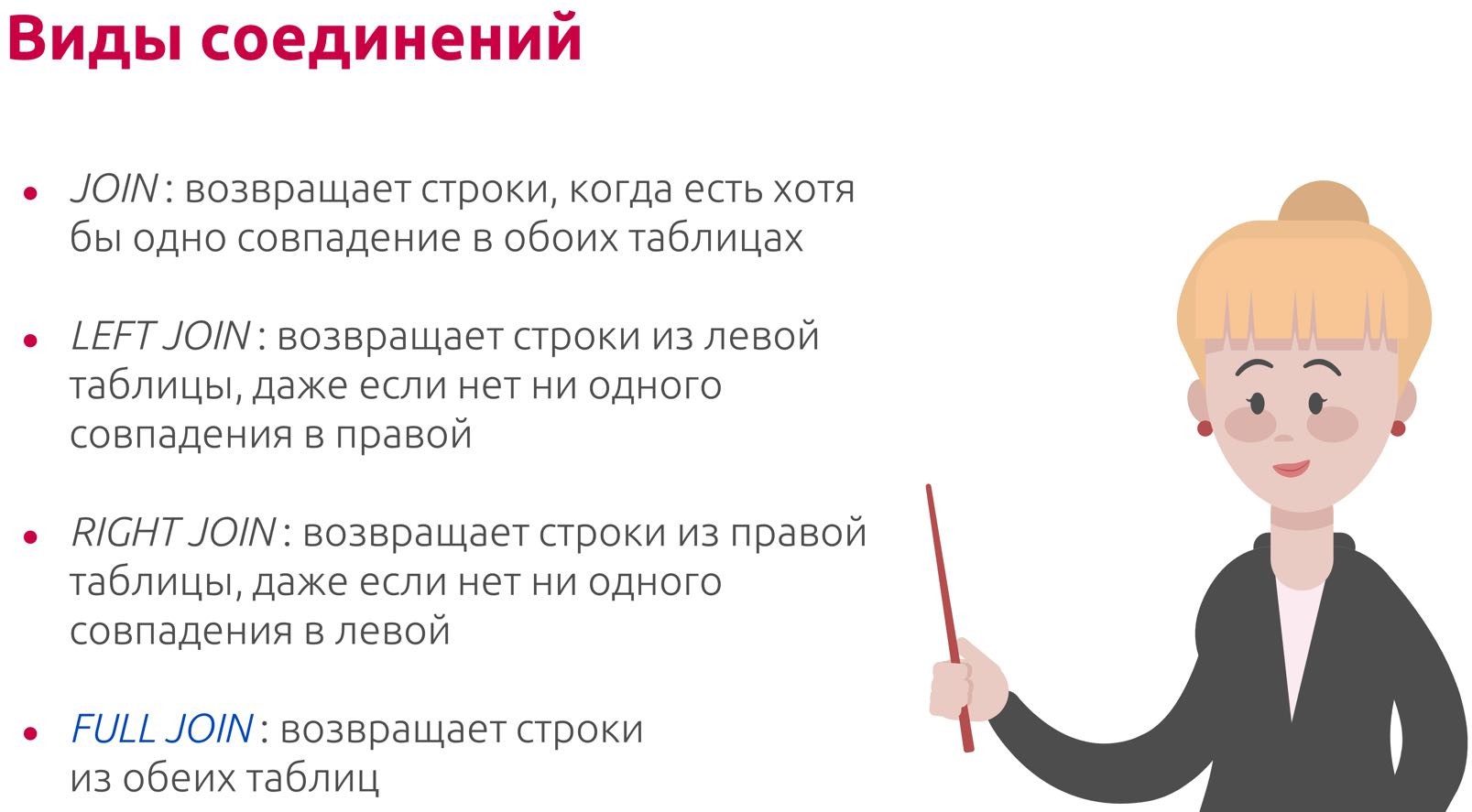
SELECT \* FROM Students LEFT JOIN PHONE\_LIST

ON Students.StudentId = PHONE\_LIST.StudentId ORDER BY Students.StudentId

Теперь мы видим список всех студентов, а у кого нет телефонного номера, на месте его значения стоит NULL – незаданное значение.



Кроме внутреннего и левого есть еще правое и полное соединения. RIGHT JOIN – возвращает строки из правой таблицы, даже если нет ни одного совпадения в левой; FULL JOIN – возвращает строки из обеих таблиц, соединяя там, где были выполнены условия соединения.



Пока мы соединяли разные таблицы. Но иногда нужно соединить таблицу саму с собой. Такое соединение часто называют SELF JOIN. Например, составим все возможные пары студентов. Как же это сделать? Если просто написать соединение таблицы самой с собой, то не понять, где в запросе указана какая таблица. Так соединить не удастся. SELF JOIN используется для соединения таблицы с ней самой таким образом, будто это две разные таблицы, временно переименовывая хотя бы одну из них.

Для симметричности переименуем оба включения таблицы Studentds в запросе – a и b. Соединять будем по неравенству полей StudentId, чтобы в пару не включили два раза одного и того же человека.

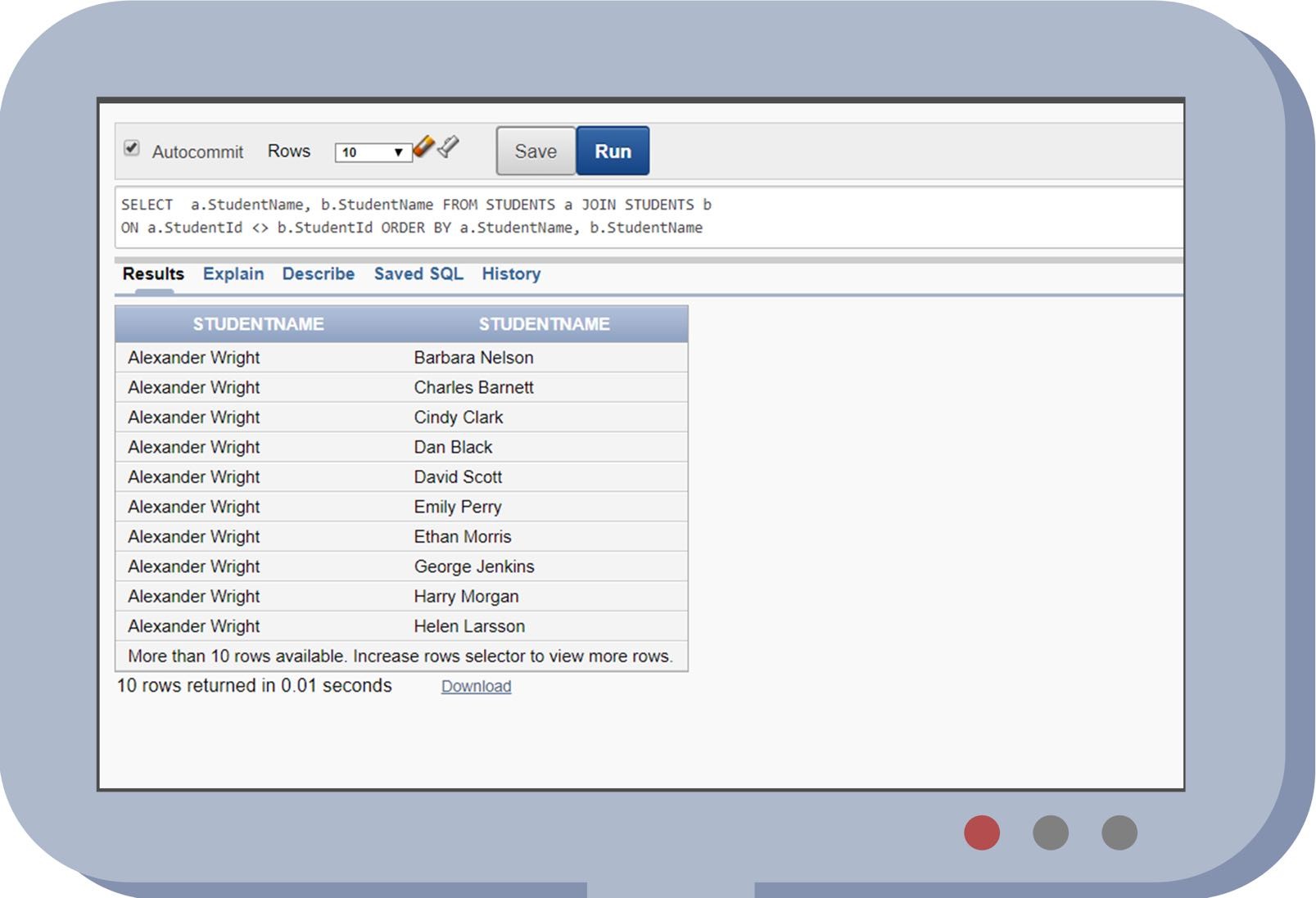
SELECT a.StudentName, b.StudentName FROM STUDENTS a JOIN STUDENTS b

ON

a.StudentId <> b.StudentId

ORDER BY a.StudentName, b.StudentName

В результирующей выборке мы видим имена для всех возможных пар студентов.



Соединять можно не только две таблицы, а три и более – столько, сколько нужно для выборки данных. Если нужно соединить несколько таблиц, то это делают последовательно: указывают имя первой таблицы, затем ключевое слово JOIN, имя второй таблицы и критерий соединения, затем снова пишут JOIN, имя третьей таблицы и критерий соединения.



Рассмотрим таблицу с описанием экзамена – в ней хранится номер экзамена, код группы, CourseId – указатель для сдаваемого курса, TeacherId – указатель для преподавателя, который будет принимать экзамен, аудитория и дата экзамена:

SELECT \* FROM EXAM\_SHEET



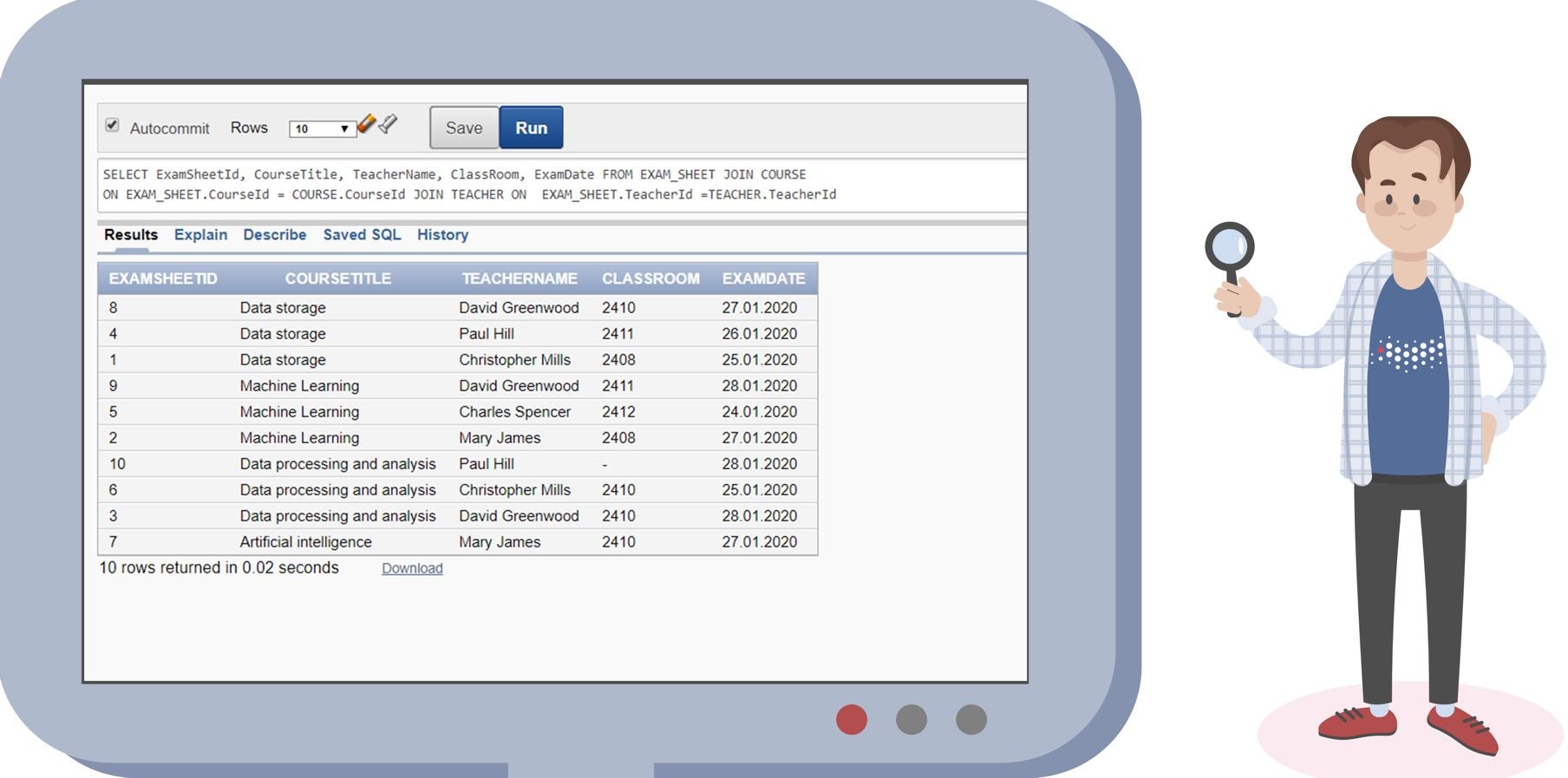
Чтобы получить в результате выборки номер экзамена, название предмета и имя преподавателя, надо таблицу EXAM\_SHEET соединить с таблицами COURSE и TEACHER. В примере запроса сначала соединяется таблица COURSE по полю CourseId, затем TEACHER по полю TeacherId. Порядок указания таблиц для операции соединения в данном случае не важен.

SELECT examSheetId, CourseTitle, TeacherName, ClassRoom, examDate FROM EXAM\_SHEET JOIN COURSE

ON EXAM\_SHEET.CourseId = COURSE.CourseId

JOIN TEACHER ON EXAM\_SHEET.TeacherId = TEACHER.TeacherId

Результат двойного соединения вы видите на рисунке.



Таким образом мы научились писать сложные запросы, указывая множество критериев.