

Запросы на языке SQL

Содержание

1	Оператор SELECT	2
2	Условия выборки	7
3	Агрегатные функции	13
4	Вложенные запросы	18
5	Теоретико-множественные операции	24
6	Соединение таблиц	29

1 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:

Синтаксис оператора SELECT

```
SELECT [ALL | DISTINCT]
      {*/[имя_столбца [AS новое_имя]]} [...n]
FROM  имя_таблицы [[AS] псевдоним] [...n]
[WHERE <условие_поиска>]
[GROUP BY имя_столбца [...n]]
[HAVING <критерии выбора групп>]
[ORDER BY имя_столбца [...n]]
```



- **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 указанной таблицы. Имя таблицы указывается после слова **FROM**. После слова **SELECT** указывают выводимые столбцы. Если после **SELECT** указан знак

SELECT * FROM Students

STUDENTID	STUDENTNAME	GROUPCODE	BIRTHDATE	ADDRESS
345567	John Smith	M2020_1	20.01.1999	711 Station Road, London NE3 5SF
345568	William Johnson	M2020_1	21.03.1996	47 Grove Road, London SW10 9KJ
345569	Lily Brown	M2020_1	17.05.1999	23 York Road, London SW14 2KA
345570	Dan Black	M2020_1	11.08.1998	701 Station Road, London NE3 5SF
345571	Matthew Wilson	M2020_1	12.07.1999	55 Springfield Road, London SE8 1JZU
345572	Jack Lewis	M2020_1	24.11.1999	7 Albert Road, London E20 9DW
345573	Cindy Clark	M2020_1	21.10.1999	63 Church Street, London W8B 3JQ
345574	Helen Larsson	B2020_1	15.03.1999	512 Station Road, London NE3 5SF
345575	Lucy Brooks	B2020_1	24.01.1999	55 Alexander Road, London EC7B 5GH
345576	Josee Reed	B2020_1	22.05.1999	47 Grange Road, London W67 1XW

SELECT * FROM St_Group

GROUPCODE	SPECIALIZATION
M2020_1	Nanotechnology
B2020_1	Health Research
B2020_2	AntiScience

* (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

Выборка заданных столбцов

- *SELECT * FROM table*
- *SELECT table.* FROM table*
- *SELECT column1, column2 FROM table*
- *SELECT table.column1, table.column2 FROM table*

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 **имя таблицы.***.

В первом запросе

```
SELECT StudentId FROM Students
```

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

```
SELECT StudentName, GroupCode FROM Students
```

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

```
SELECT GroupCode FROM Students
```

выводит только номера групп, в которых учатся студенты.

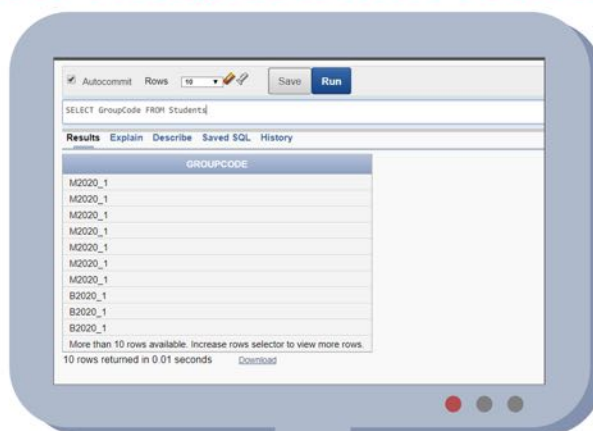
Выборка заданных столбцов

- *SELECT StudentId FROM Students*
- *SELECT StudentName, GroupCode FROM Students*
- *SELECT GroupCode FROM Students*

StudentId	StudentName	GroupCode	BirthDate	Address
345568	William Johnson	M2020_1	01/20/1999	711 Station Road, London N63 5SF
345569	Lily Brown	M2020_1	05/17/1999	93 Manchester Road, London SE02 6VS
345583	Thomas Wood	B2020_2	09/12/1999	951 Highfield Road, London E16 2RI
345571	Matthew Wilson	M2020_1	07/12/1999	55 Springfield Road, London SE51 3ZU
345579	Michael Young	B2020_1	04/08/1998	29 The Crescent, London W20 9FK

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.

SELECT GroupCode FROM Students



По умолчанию выводятся все значения выборки, чтобы указать это в явном виде, можно перед именами полей указать ключевое слово **ALL**. Ес-

Уникальные строки

- *SELECT GroupCode FROM Students*
- *SELECT ALL GroupCode FROM Students*
- *SELECT DISTINCT GroupCode FROM Students*

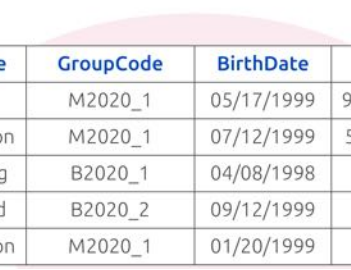
ли нужно выводить только уникальные строки, то перед именем поля (или

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

- *SELECT * FROM Students ORDER BY StudentName*



StudentId	StudentName	GroupCode	BirthDate	Address
345569	Lily Brown	M2020_1	05/17/1999	93 Manchester Road, London SE02 6VS
345571	Matthew Wilson	M2020_1	07/12/1999	55 Springfield Road, London SE51 3ZU
345579	Michael Young	B2020_1	04/08/1998	29 The Crescent, London W20 9FK
345583	Thomas Wood	B2020_2	09/12/1999	951 Highfield Road, London E16 2RI
345568	William Johnson	M2020_1	01/20/1999	711 Station Road, London N63 5SF

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 из 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 выборке, но такой способ указания на столбцы не часто используют.

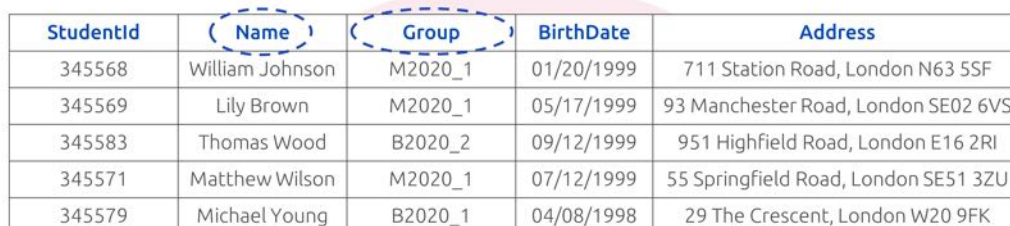
```
SELECT StudentName, GroupCode FROM Students ORDER by 2, 1
```

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
```

- *SELECT StudentName as **Name**, GroupCode as **Group** FROM Students*



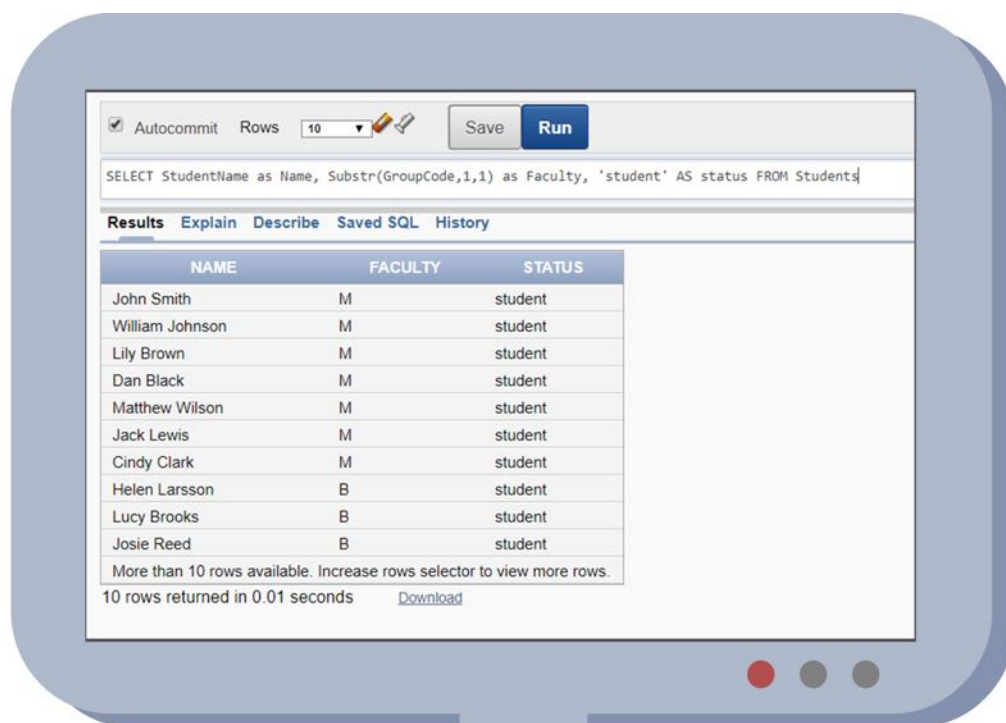
StudentId	(Name)	(Group)	BirthDate	Address
345568	William Johnson	M2020_1	01/20/1999	711 Station Road, London N63 5SF
345569	Lily Brown	M2020_1	05/17/1999	93 Manchester Road, London SE02 6VS
345583	Thomas Wood	B2020_2	09/12/1999	951 Highfield Road, London E16 2RI
345571	Matthew Wilson	M2020_1	07/12/1999	55 Springfield Road, London SE51 3ZU
345579	Michael Young	B2020_1	04/08/1998	29 The Crescent, London W20 9FK

На самом деле, можно не только выводить значения столбцов таблицы, а составлять из 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
```

2 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 `Students` 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

Условие выборки – сравнение

Сравнение: сравниваются результаты вычисления одного выражения с результатами вычисления другого.

< 345569

StudentId	StudentName	GroupCode	BirthDate	Address
345568	✓ William Johnson	M2020_1	01/20/1999	711 Station Road, London N63 5SF
345569	✗ Lily Brown	M2020_1	05/17/1999	93 Manchester Road, London SE02 6VS
345583	✗ Thomas Wood	B2020_2	09/12/1999	951 Highfield Road, London E16 2RI
345571	✗ Matthew Wilson	M2020_1	07/12/1999	55 Springfield Road, London SE51 3ZU
345579	✗ Michael Young	B2020_1	04/08/1998	29 The Crescent, London W20 9FK

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'
```

Условие выборки – принадлежность множеству

Принадлежность множеству: проверяется, принадлежит ли результат вычислений выражения заданному множеству значений.

\in
{M2020_1,
B2020_1}

StudentId	StudentName	GroupCode	BirthDate	Address
345568	William Johnson	M2020_1 ✓	01/20/1999	711 Station Road, London N63 5SF
345569	Lily Brown	M2020_1 ✓	05/17/1999	93 Manchester Road, London SE02 6VS
345583	Thomas Wood	B2020_2 ✗	09/12/1999	951 Highfield Road, London E16 2RI
345571	Matthew Wilson	M2020_1 ✓	07/12/1999	55 Springfield Road, London SE51 3ZU
345579	Michael Young	B2020_1 ✓	04/08/1998	29 The Crescent, London W20 9FK

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 принадлежности 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?

Условие выборки – незаданное значение атрибута

Значение NULL: проверяется, содержит ли данный столбец определитель NULL (неизвестное значение).



StudentId	StudentName	GroupCode	BirthDate	Address
345568	William Johnson	M2020_1	01/20/1999 ✗	711 Station Road, London N63 5SF
345569	Lily Brown	M2020_1	05/17/1999 ✗	93 Manchester Road, London SE02 6VS
345583	Thomas Wood	B2020_2	- ✓	951 Highfield Road, London E16 2RI
345571	Matthew Wilson	M2020_1	07/12/1999 ✗	55 Springfield Road, London SE51 3ZU
345579	Michael Young	B2020_1	04/08/1998 ✗	29 The Crescent, London W20 9FK

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
```

Условие выборки – шаблоны для строк

Соответствие шаблону: проверяется, отвечает ли некоторое строковое значение заданному шаблону.



StudentId	StudentName		GroupCode	BirthDate	Address
345568	William Johnson	✓	M2020_1	01/20/1999	711 Station Road, London N63 5SF
345569	Lily Brown	✓	M2020_1	05/17/1999	93 Manchester Road, London SE02 6VS
345583	Thomas Wood	✗	B2020_2	09/12/1999	951 Highfield Road, London E16 2RI
345571	Matthew Wilson	✓	M2020_1	07/12/1999	55 Springfield Road, London SE51 3ZU
345579	Michael Young	✗	B2020_1	04/08/1998	29 The Crescent, London W20 9FK

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 3 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.

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

STUDENTID	STUDENTNAME	GROUPCODE	BIRTHDATE	ADDRESS
345567	John Smith	M2020_1	20.01.1999	711 Station Road, London N63 5SF
345568	William Johnson	M2020_1	21.03.1998	47 Grove Road, London SW10 9KJ
345569	Lily Brown	M2020_1	17.05.1999	23 York Road, London SW14 2KA
345570	Dan Black	M2020_1	11.08.1998	701 Station Road, London N63 5SF
345571	Matthew Wilson	M2020_1	12.07.1999	55 Springfield Road, London SE51 3ZU
345572	Jack Lewis	M2020_1	24.11.1999	7 Albert Road, London E20 8DW
345573	Cindy Clark	M2020_1	21.10.1999	63 Church Street, London W66 3JQ

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

STUDENTID	STUDENTNAME	GROUPCODE	BIRTHDATE	ADDRESS
345567	John Smith	M2020_1	20.01.1999	711 Station Road, London N63 5SF
345568	William Johnson	M2020_1	21.03.1998	47 Grove Road, London SW10 9KJ
345569	Lily Brown	M2020_1	17.05.1999	23 York Road, London SW14 2KA
345570	Dan Black	M2020_1	11.08.1998	701 Station Road, London N63 5SF
345571	Matthew Wilson	M2020_1	12.07.1999	55 Springfield Road, London SE51 3ZU
345572	Jack Lewis	M2020_1	24.11.1999	7 Albert Road, London E20 8DW
345573	Cindy Clark	M2020_1	21.10.1999	63 Church Street, London W66 3JQ
345574	Helen Larsen	B2020_1	18.02.1999	612 Station Road, London N63 5SF
345575	Lucy Brooks	B2020_1	24.01.1999	56 Alexander Road, London EC78 5GR
345576	Joan Reed	B2020_1	22.06.1999	47 Grange Road, London W67 1XW

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

Условие выборки – шаблоны для строк

Соответствие шаблону: проверяется, отвечает ли некоторое строковое значение заданному шаблону.

Чьи фамилии начинаются на «B»?

StudentId	StudentName	GroupCode	BirthDate	Address
345568	William Johnson ✗	M2020_1	01/20/1999	711 Station Road, London N63 5SF
345569	Lily Brown ✓	M2020_1	05/17/1999	93 Manchester Road, London SE02 6VS
345583	Thomas Wood ✗	B2020_2	09/12/1999	951 Highfield Road, London E16 2RI
345571	Matthew Wilson ✗	M2020_1	07/12/1999	55 Springfield Road, London SE51 3ZU
345579	Michael Young ✗	B2020_1	04/08/1998	29 The Crescent, London W20 9FK

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

Логические выражения

- Операции одного порядка вычисляются слева направо.
- Первыми вычисляются арифметические операции.
- Операция **NOT** выполняется до выполнения операций **AND** и **OR**.
- Операции **AND** выполняются до выполнения операций **OR**.



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
```

3 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

```
SELECT COUNT(*) FROM Students
```



StudentId	StudentName	GroupCode	BirthDate	Address
345568	William Johnson	M2020_1	01/20/1999	711 Station Road, London N63 5SF
345569	Lily Brown	M2020_1	05/17/1999	93 Manchester Road, London SE02 6VS
345583	Thomas Wood	B2020_2	09/12/1999	951 Highfield Road, London E16 2RI
345571	Matthew Wilson	M2020_1	07/12/1999	55 Springfield Road, London SE51 3ZU
345579	Michael Young	B2020_1	04/08/1998	29 The Crescent, London W20 9FK

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 условию.

Например, сосчитаем не всех студентов в таблице, а только тех, которые учатся 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
```

Агрегирование уникальных строк

Сколько здесь разных групп?

```
SELECT COUNT(DISTINCT GroupCode) FROM Students
```

StudentId	StudentName	GroupCode	BirthDate	Address
345568	William Johnson	M2020_1	01/20/1999	711 Station Road, London N63 5SF
345569	Lily Brown	M2020_1	05/17/1999	93 Manchester Road, London SE02 6VS
345583	Thomas Wood	B2020_2	09/12/1999	951 Highfield Road, London E16 2RI
345571	Matthew Wilson	M2020_1	07/12/1999	55 Springfield Road, London SE51 3ZU
345579	Michael Young	B2020_1	04/08/1998	29 The Crescent, London W20 9FK

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 значения 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

Агрегирование с группировкой

Сколько студентов в каждой из групп?

```
SELECT COUNT(*) FROM Students GROUP BY GroupCode
```

StudentId	StudentName	GroupCode	BirthDate	Address
345568	William Johnson	M2020_1	01/20/1999	711 Station Road, London N63 5SF
345569	Lily Brown	M2020_1	05/17/1999	93 Manchester Road, London SE02 6VS
345583	Thomas Wood	B2020_2	09/12/1999	951 Highfield Road, London E16 2RI
345571	Matthew Wilson	M2020_1	07/12/1999	55 Springfield Road, London SE51 3ZU
345579	Michael Young	B2020_1	04/08/1998	29 The Crescent, London W20 9FK

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 3 студентов и название группы начинается с символа В.

```
SELECT GroupCode, COUNT(*) FROM Students
GROUP BY GroupCode
HAVING COUNT(*) > 3 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.

4 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.

Порядок выполнения **SELECT**

- *FROM*
- *WHERE*
- *GROUP BY*
- *HAVING*
- *SELECT*
- *ORDER BY*



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.

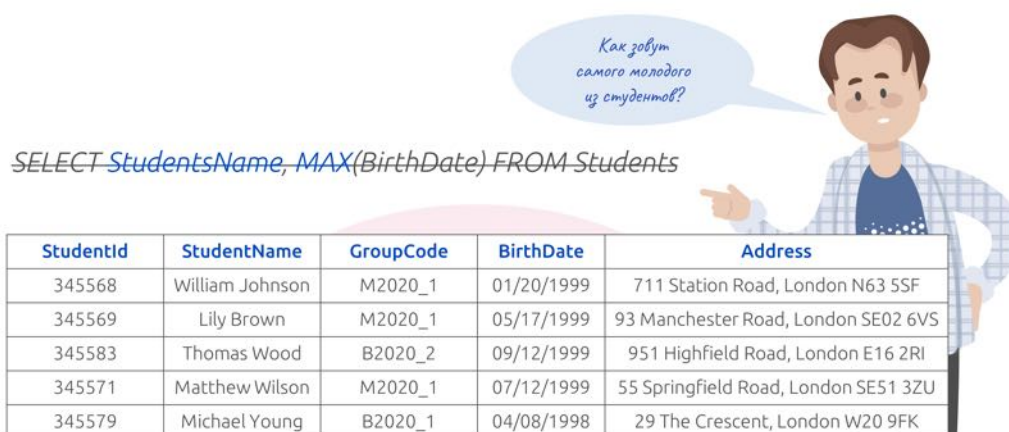
Вложенные запросы могут возвращать одно значение, а могут в качестве результата иметь целую таблицу. Бывают вложенные запросы, которые называют коррелированными, если они ссылаются на столбцы таблицы внешнего запроса.

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



Как зовут самого молодого из студентов?

```
SELECT StudentName, MAX(BirthDate) FROM Students
```

StudentId	StudentName	GroupCode	BirthDate	Address
345568	William Johnson	M2020_1	01/20/1999	711 Station Road, London N63 5SF
345569	Lily Brown	M2020_1	05/17/1999	93 Manchester Road, London SE02 6VS
345583	Thomas Wood	B2020_2	09/12/1999	951 Highfield Road, London E16 2RI
345571	Matthew Wilson	M2020_1	07/12/1999	55 Springfield Road, London SE51 3ZU
345579	Michael Young	B2020_1	04/08/1998	29 The Crescent, London W20 9FK

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 единственный результат. Давайте сделаем так, чтобы в результате первого запроса находилось несколько значений. Найдем студентов, у которых специализация **Нанотехнология** или **Здравоохранение**. 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
```

Вложенные запросы

```
SELECT StudentName, GroupCode,
(SELECT Specialization FROM St_Group WHERE
St_Group.GroupCode=Students.GroupCode) FROM Students;
```

StudentName	GroupCode	GroupCode	Specialization
William Johnson	M2020_1	M2020_1	Nanotechnology
Lily Brown	M2020_1	B2020_1	Health Research
Thomas Wood	B2020_2	B2020_2	Art&Science
Matthew Wilson	M2020_1		
Michael Young	B2020_1		



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».

Формы с вложенными запросами

- `EXISTS` = «Существует»
- `ANY` = «Хотя бы один»
- `ALL` = «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

StudentId	PhoneType	Phone
345568	cell	07107534674
345571	cell	07300678543
345579	cell	07911365676
345583	cell	07931676657
345587	cell	07931676776

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 выборка 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.

Операторы ALL/ANY

*SELECT * FROM T WHERE <выражение>
<оператор сравнения> ANY(вложенный запрос)*

*SELECT * FROM T WHERE <выражение>
<оператор сравнения> ALL(вложенный запрос)*



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**.

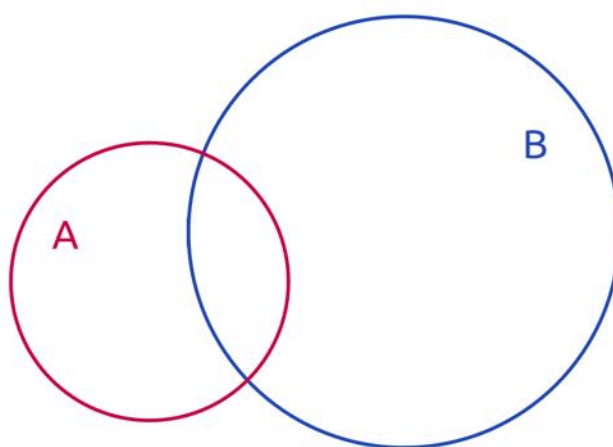
```
SELECT StudentID, StudentName FROM Students
WHERE 3 < ALL (SELECT Grade FROM EXAM_RESULT WHERE
Students.StudentID=EXAM_RESULT.StudentID)
```

5 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.

Теоретико-множественные операции

- *UNION*
- *INTERSECT*
- *EXCEPT*



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?

- *SELECT ... UNION SELECT...*
- *SELECT ... INTERSECT SELECT...*
- *SELECT ... EXCEPT/MINUS SELECT...*



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.

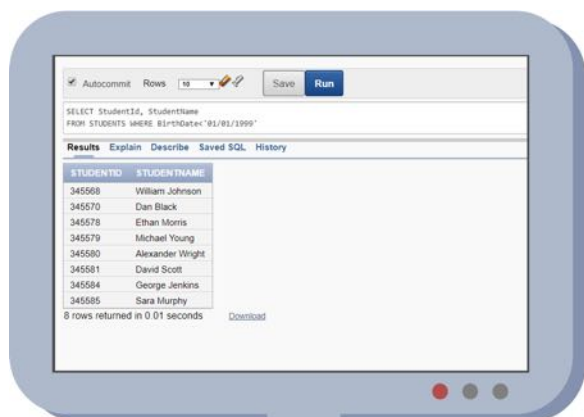
Подготовим две выборки. Например, первый запрос возвращает нам поля `StudentId` и `StudentName` из таблицы `STUDENTS` со значением `BirthDate < '01/01/1999'`:

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

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

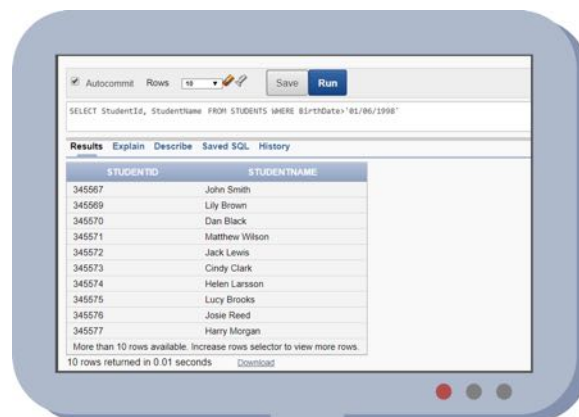
```
SELECT StudentId, StudentName FROM STUDENTS
WHERE BirthDate > '06/01/1998'
```

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



The screenshot shows a SQL query result with 8 rows. The columns are `STUDENTID` and `STUDENTNAME`. The data is as follows:

STUDENTID	STUDENTNAME
345568	William Johnson
345570	Dan Black
345578	Ethan Morris
345579	Michael Young
345580	Alexander Wright
345581	David Scott
345584	George Jenkins
345585	Sara Murphy



The screenshot shows a SQL query result with 10 rows. The columns are `STUDENTID` and `STUDENTNAME`. The data is as follows:

STUDENTID	STUDENTNAME
345567	John Smith
345569	Lily Brown
345570	Dan Black
345571	Matthew Wilson
345572	Jack Lewis
345573	Cindy Clark
345574	Helen Larsson
345575	Lucy Brooks
345576	Josie Reed
345577	Harry Morgan

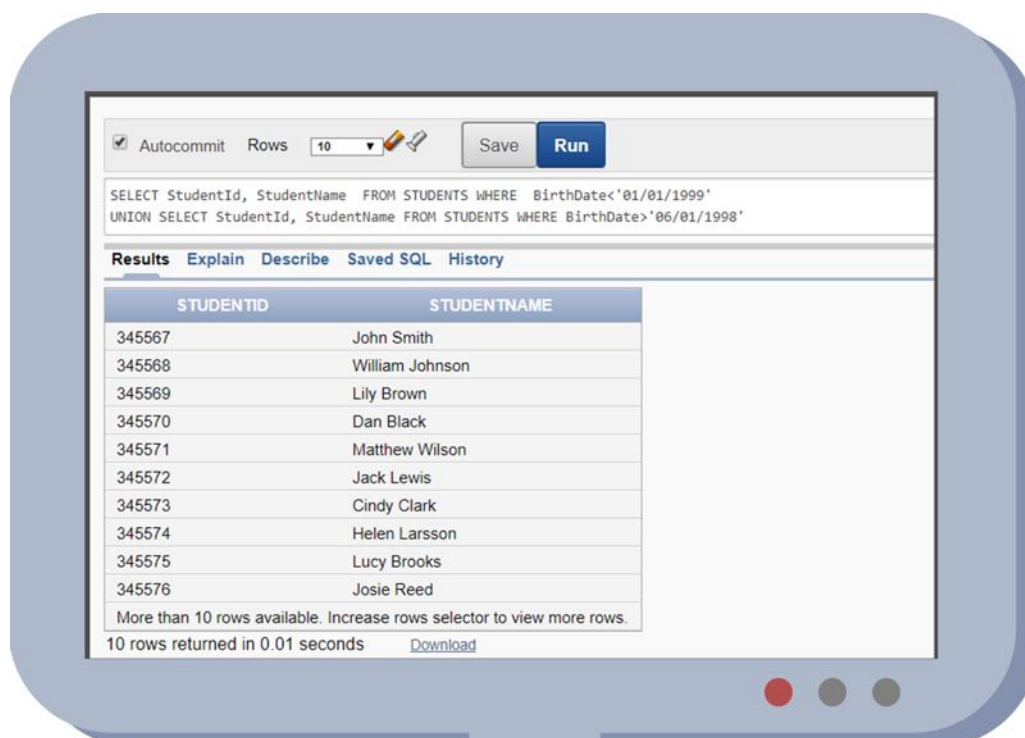
и разности.

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:

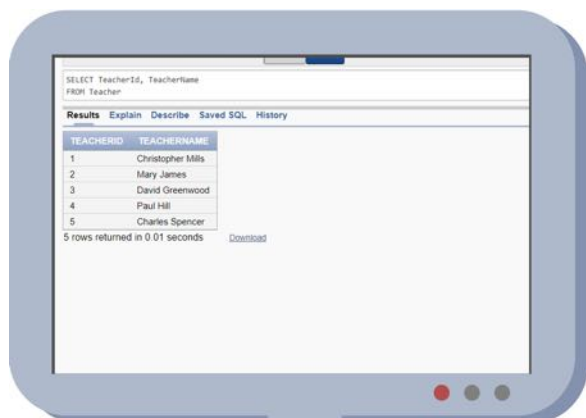
```
SELECT StudentId, StudentName FROM STUDENTS
WHERE BirthDate < '01/01/1999'
INTERSECT
SELECT StudentId, StudentName FROM STUDENTS
WHERE BirthDate > '06/01/1998'
```

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'
```

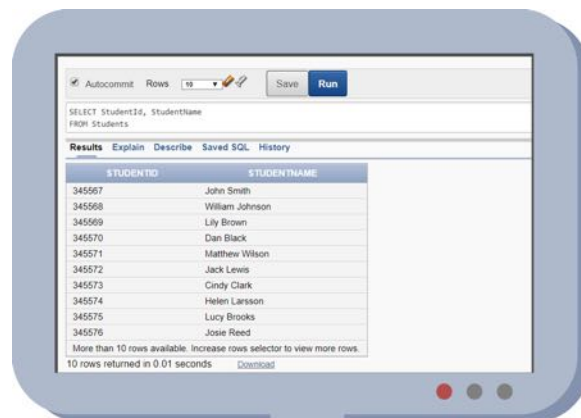
Приведем еще один пример объединения. Сделаем выборку из таблицы **Teachers**: выберем поля **TeacherId**, **TeacherName**. Можно ли эту выборку объединить с выборкой из таблицы **Students**? В каждой из выборок два столбца, первые столбцы целочисленные, вторые строковые. Правда, на имя



SELECT TeacherId, TeacherName
FROM Teacher

TEACHERID	TEACHERNAME
1	Christopher Mills
2	Mary James
3	David Greenwood
4	Paul Hill
5	Charles Spencer

5 rows returned in 0.01 seconds [Download](#)



SELECT StudentId, StudentName
FROM Students

STUDENTID	STUDENTNAME
345567	John Smith
345568	William Johnson
345569	Lily Brown
345570	Dan Black
345571	Matthew Wilson
345572	Jack Lewis
345573	Cindy Clark
345574	Helen Larsson
345575	Lucy Brooks
345576	Josie Reed

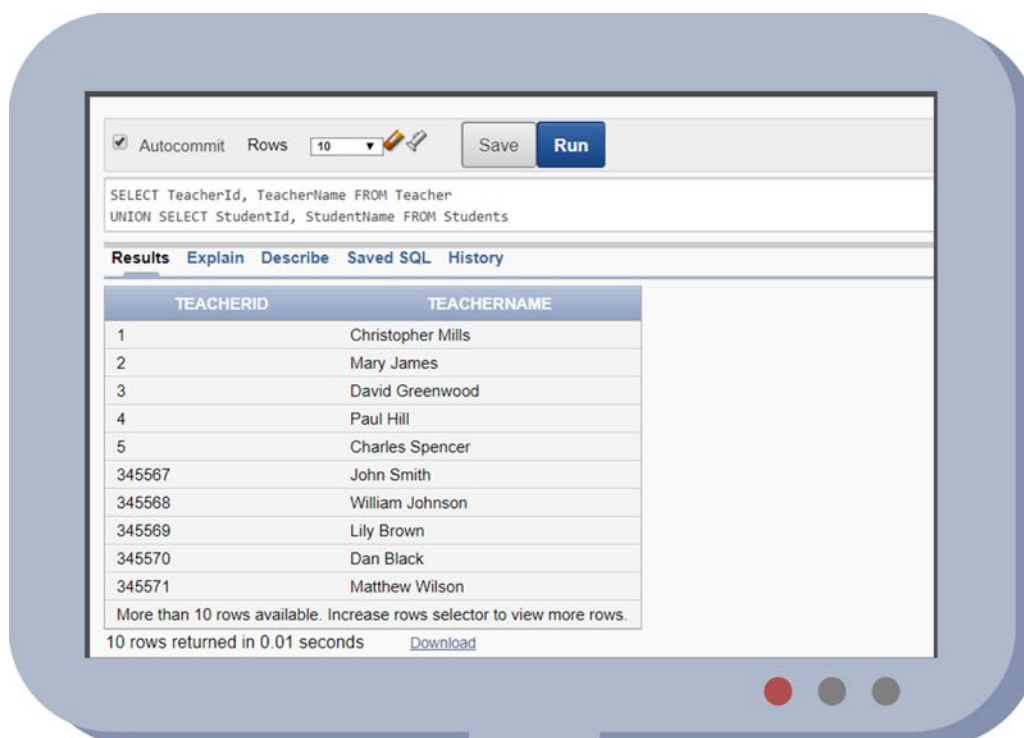
More than 10 rows available. Increase rows selector to view more rows.
10 rows returned in 0.01 seconds [Download](#)

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



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

TEACHERID	TEACHERNAME
1	Christopher Mills
2	Mary James
3	David Greenwood
4	Paul Hill
5	Charles Spencer
345567	John Smith
345568	William Johnson
345569	Lily Brown
345570	Dan Black
345571	Matthew Wilson

More than 10 rows available. Increase rows selector to view more rows.
10 rows returned in 0.01 seconds [Download](#)

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

Декартово произведение

- `SELECT *`
`FROM Table1, Table2`



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.

6 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

Операция внутреннего соединения



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 со-

STUDENTS + PHONE_LIST

StudentId	StudentName	GroupCode	Address	BirthDate	PhoneType	Phone
...

JOIN

- *SELECT **
FROM Table1 JOIN Table2
ON Table1.field1=Table2.field2
- *SELECT **
FROM Table1, Table2 WHERE
Table1.field1=Table2.field2



единения 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:

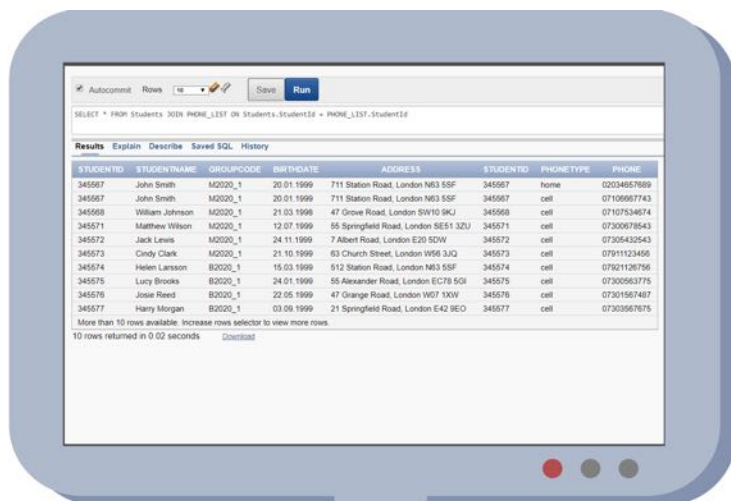
```
SELECT * FROM Students JOIN PHONE_LIST
ON Students.StudentId = PHONE_LIST.StudentId
```

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.

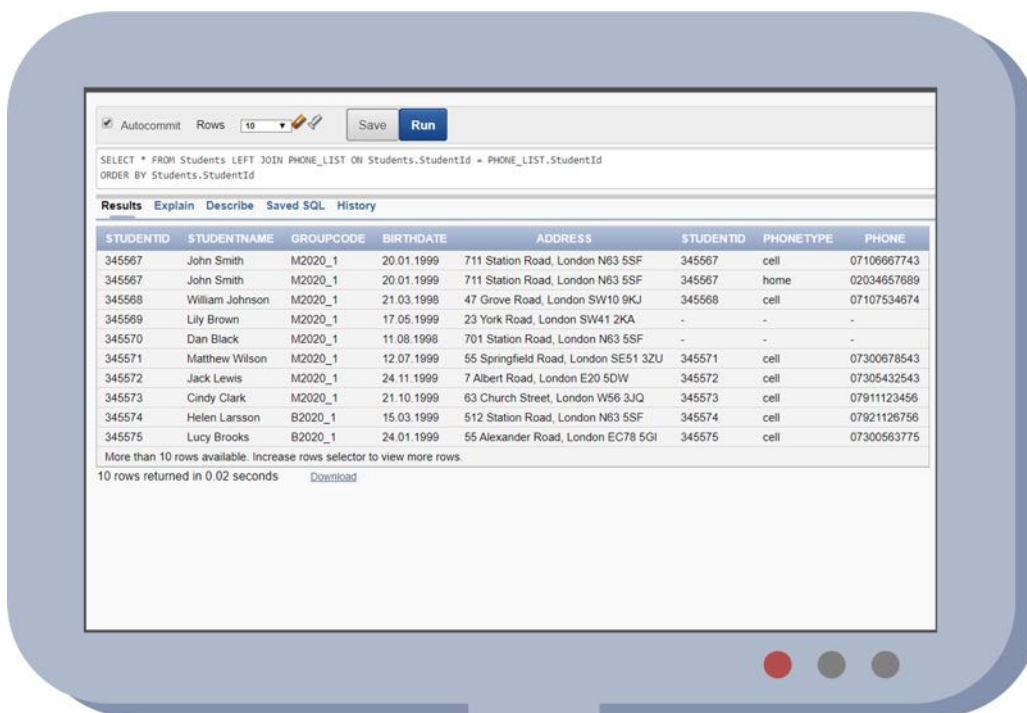
Обычное соединение называют еще «внутренним», т.к. в выборку попадают только строки, содержащие значения, используемые в условии соединения, которые находятся «внутри» как первой, так и второй таблицы.



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 незадаанными 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** – незадаанное 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.

Виды соединений

- *JOIN*: возвращает строки, когда есть хотя бы одно совпадение в обеих таблицах
- *LEFT JOIN*: возвращает строки из левой таблицы, даже если нет ни одного совпадения в правой
- *RIGHT JOIN*: возвращает строки из правой таблицы, даже если нет ни одного совпадения в левой
- *FULL JOIN*: возвращает строки из обеих таблиц



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

SELF JOIN

- ~~*SELECT **~~
FROM S JOIN S ON S.field1 = S.field1
- *SELECT a.*, b.**
FROM S as a
JOIN S as b
ON a.field1 = b.field2

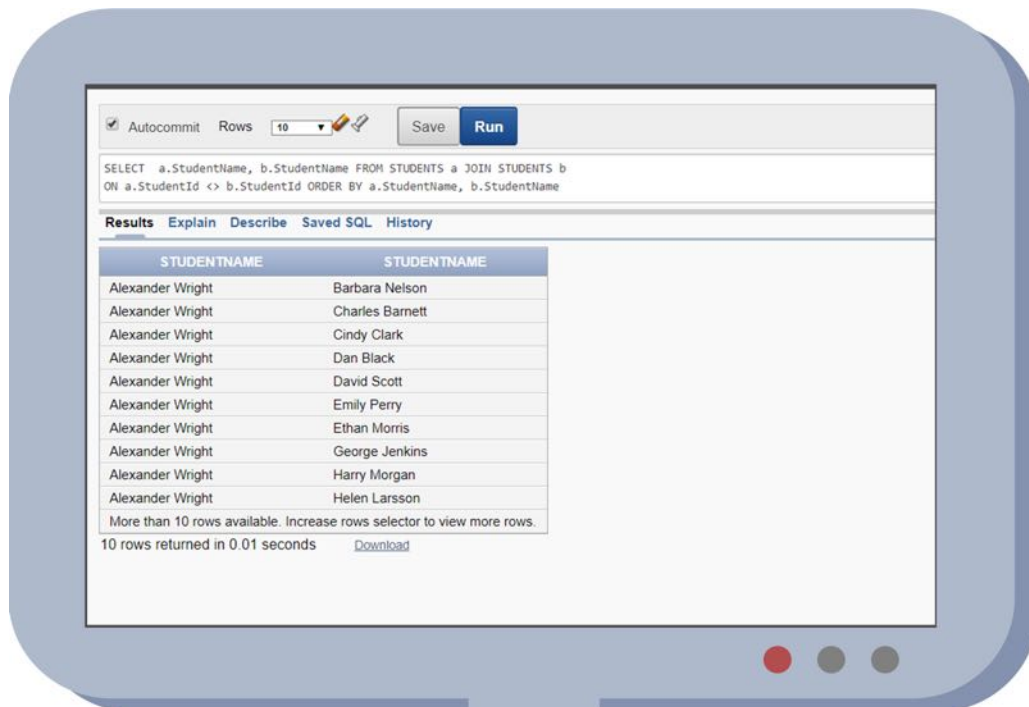


написать соединение таблицы самой с собой, то не понять, где в запросе указана какая таблица. Так соединить не удастся. **SELF JOIN** используется для соединения таблицы с ней самой таким образом, будто это две разные таблицы, временно переименовывая хотя бы одну из них.

Для симметричности переименуем оба включения таблицы **Students** 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.

Соединение трех таблиц

- *SELECT **
FROM Table1
JOIN Table2
ON Table1.field1 = Table2.field2
JOIN Table3
ON Table2.field3 = Table3.field4
- *SELECT **
FROM Table1, Table2, Table3
WHERE Table1.field1 = Table2.field2
AND Table2.field3 = Table3.field4



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:

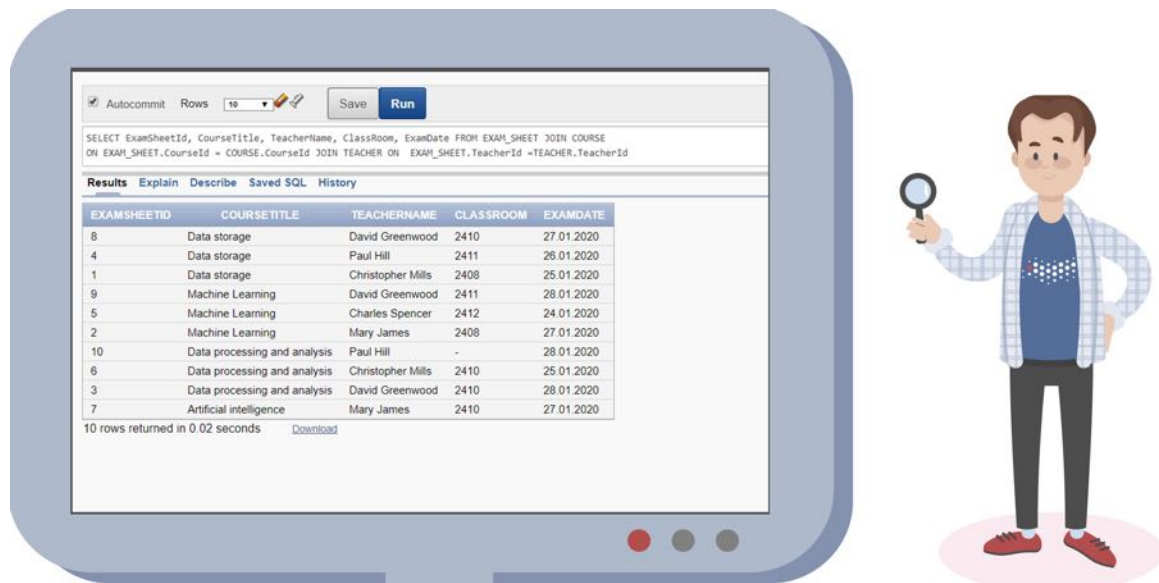
```
SELECT * FROM EXAM_SHEET
```

EXAM_SHEET					
ExamSheetId	GroupCode	CourseId	TeacherId	ClassRoom	ExamDate
...

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.