

**top**

КОМПЬЮТЕРНАЯ  
АКАДЕМИЯ

# Теория Баз Данных

# Unit 3

## Запросы SELECT, INSERT, UPDATE, DELETE

# Содержание

<b>1. Оператор SELECT .....</b>	<b>5</b>
1.1. Предложение SELECT .....	5
1.2. Предложение FROM .....	5
1.3. Предложение WHERE .....	12
1.4. Предложение ORDER BY .....	17
<b>2. Ключевые слова IN, BETWEEN, LIKE.....</b>	<b>20</b>
<b>3. Оператор INSERT .....</b>	<b>25</b>
<b>4. Оператор UPDATE.....</b>	<b>34</b>
<b>5. Оператор DELETE .....</b>	<b>36</b>
<b>6. Понятие транзакции.</b>	
<b>Использование транзакций.....</b>	<b>38</b>
<b>7. Домашнее задание .....</b>	<b>42</b>

# 1. Оператор SELECT

На прошлом занятии мы продемонстрировали вам пример SQL-запроса, который вернул фамилии студентов, родившихся в ноябре. На текущем занятии вы более подробно узнаете, какие операторы используются при написании запросов, и начнем мы с оператора **SELECT**.

## 1.1. Предложение SELECT

Предложение **SELECT** позволяет считывать необходимую информацию из базы данных. В самом простом варианте использования после инструкции **SELECT** указываются имена столбцов, значения которых требуется получить из одной или нескольких таблиц, названия этих столбцов разделяются между собой запятыми. Также после оператора **SELECT** можно указывать вычисляемые результаты и функции агрегирования, о которых вы узнаете в одном из последующих уроков.

Написание SQL-запроса с применением только одного оператора **SELECT** приведет к синтаксической ошибке, потому что необходимо использовать его в сочетании еще как минимум с одним оператором — **FROM**.

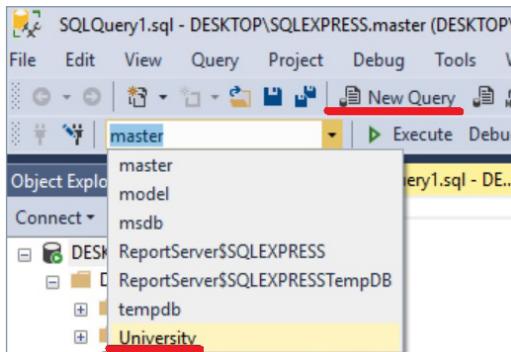
## 1.2. Предложение FROM

Предложение **FROM** используется для указания источника получения данных, после него прописывается название таблицы или список таблиц, разделенных между собой запятыми. Общая форма записи SQL-запроса

с использованием операторов **SELECT** и **FROM** выглядит следующим образом:

```
SELECT columnName1, columnName2, ...
FROM tableName;
```

Продемонстрируем простой запрос, позволяющий получить всю информацию из таблицы **Students** базы данных **University**, которая была создана нами на предыдущем уроке. Для этого откроем MS SQL Server Management Studio 17 и на панели инструментов нажмем кнопку **New Query** (Рисунок 1.1) или сочетание клавиш **Ctrl+N**, после чего в выпадающем списке необходимо выбрать требуемую БД (Рисунок 1.1).



**Рисунок 1.1.** Создание нового запроса

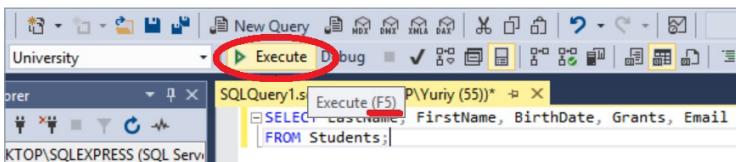
В появившемся окне напишем SQL-запрос:

```
SELECT LastName, FirstName, BirthDate,
Grants, Email
FROM Students;
```

При помощи этого запроса мы хотим получить значения столбцов **LastName**, **FirstName**, **BirthDate**, **Grants** и **Email** для всех записей из таблицы **Students**.

## 1. Оператор SELECT

Для того чтобы, написанный нами, запрос выполнился необходимо нажать кнопку Execute или клавишу F5 на клавиатуре (Рисунок 1.2).



**Рисунок 1.2.** Запуск запроса на выполнение

В результате SQL-запроса будет создана виртуальная таблица, в которой содержится вся запрошенная информация из физической таблицы **Students** базы данных **University** (Рисунок 1.3).

	LastName	FirstName	BirthDate	Grants	Email
1	Doe	John	1998-02-11	NULL	jh@net.eu
2	Jones	Jack	1997-11-05	1256.00	jj@net.eu
3	Johnson	Joshua	1997-05-23	NULL	jo@net.eu

**Рисунок 1.3.** Получение всех записей из таблицы **Students**

В языке SQL существует возможность написать запрос, в котором вместо списка столбцов таблицы можно указать символ \*:

```
SELECT *
FROM Students;
```

И хотя результат мы получим такой же (Рисунок 1.3) данный SQL-запрос будет выполняться дольше. Это происходит, потому что требуемые столбцы не указаны явным образом, и СУБД приходиться тратить время на определение и получение всех столбцов в таблице. Разни-

ца по времени при небольшом количестве записей будет не существенной, однако при больших объемах данных будет весьма заметна, поэтому настоятельно рекомендуется в SQL-запросе указывать все столбцы, информацию по которым необходимо получить.

При написании SQL-запросов можно формировать новые столбцы в виртуальной таблице, соединяя значения нескольких реальных столбцов при помощи символа `+`, в этом случае заголовок такого столбца будет **(No column name)**, если такое положение вещей вас не устраивает, то вы можете задать псевдоним (*alias*) этому столбцу при помощи оператора `AS`. Например, соединив в SQL-запросе имя и фамилию студента через пробел, укажем этому столбцу псевдоним `FullName`:

```
SELECT FirstName +' '+ LastName AS FullName,
BirthDate, Grants, Email
FROM Students;
```

Результат запроса представлен на рисунке 1.4.

	FullName	BirthDate	Grants	Email
1	John Doe	1998-02-11	NULL	jh@net.eu
2	Jack Jones	1997-11-05	1256.00	jj@net.eu
3	Joshua Johnson	1997-05-23	NULL	jo@net.eu

**Рисунок 1.4.** Использование псевдонима при формировании нового столбца

При написании SQL-запроса вы можете изменять данные в виртуальной таблице, применяя различные арифметические действия, и это никак не скажется на реальных данных. Допустим «ректор нашего университета» захотел узнать: «Как изменяются стипендии студентов, если их

## 1. Оператор SELECT

увеличить на 20 процентов?». Для того чтобы ответить на этот вопрос мы можем написать следующий запрос:

```
SELECT FirstName + ' ' + LastName AS FullName,  
Grants*1.2 AS [Plus 20 percent]  
FROM Students;
```

В предыдущем запросе продемонстрирована одна особенность при написании имен столбцов: если в названии столбца присутствуют пробелы, то это название необходимо брать в квадратные скобки. Результат этого SQL-запроса представлен на рисунке 1.5.

	FullName	Plus 20 percent
1	John Doe	NULL
2	Jack Jones	1507.200
3	Joshua Johnson	NULL

**Рисунок 1.5.** Использование арифметических операций

Оператор **SELECT** позволяет возвращать информацию в виде строки, которая формируется путем конкатенации (сцепления) определенной текстовой информации и значений из различных столбцов. Однако при этом указанные столбцы должны хранить текстовые данные, в противном случае при выполнении SQL-запроса мы получим ошибку (Рисунок 1.6).

The screenshot shows a SQL query window titled "SQLQuery1.sql - DE...ESKTOP\Yuriy (55)\*". The query is:

```
SELECT 'Student ' + LastName + ' receives ' + Grants  
FROM Students;
```

The "Results" tab is selected, displaying the following error message:

```
Msg 8114, Level 16, State 5, Line 1  
Error converting data type nvarchar to numeric.
```

**Рисунок 1.6.** Ошибка: недопустимое приведение типов данных

Данная ошибка произошла из-за невозможности преобразовать вещественные значения столбца **Grant** к строковому типу данных. Чтобы решить эту проблему в языке T-SQL существуют две стандартные функции **CAST()** и **CONVERT()**, которые позволяют осуществлять приведение данных к требуемому типу. Эти функции отличаются принимаемыми параметрами, но сравнив следующие запросы, вы разберетесь, как их использовать.

```
SELECT 'Student ' + LastName + ' receives ' +
CAST(Grants AS nvarchar(10))
FROM Students;
SELECT 'Student ' + LastName + ' receives ' +
CONVERT(nvarchar(10), Grants)
FROM Students;
```

В результате выполнения двух этих запросов мы получим одинаковые результаты (Рисунок 1.7).

(No column name)	
1	NULL
2	Student Jones receives 1256.00
3	NULL

**Рисунок 1.7.** Формирование SQL-запроса в виде строки

Наличие **NULL**-значений объясняется, тем, что у некоторых студентов стипендия имеет неопределенное значение, как убрать такие записи из результирующей таблицы вы узнаете в следующем подразделе.

В языке T-SQL совместно с предложением **SELECT** можно использовать оператор **TOP**, который позволяет получить первые записи либо в количественном, либо процентном отношении.

Допустим нам надо получить первые две записи в таблице **Students**, тогда запрос будет выглядеть следующим образом:

```
SELECT TOP 2 LastName, FirstName, BirthDate
FROM Students;
```

Результат SQL-запроса представлен на рисунке 1.8.

	LastName	FirstName	BirthDate
1	Doe	John	1998-02-11
2	Jones	Jack	1997-11-05

**Рисунок 1.8.** Использование оператора TOP

Если нам понадобится получить, например, 20 % первых записей из таблицы **Students**, тогда после числового значения необходимо добавить ключевое слово **PERCENT**, что демонстрируется в следующем SQL-запросе (результат на рисунке 1.9).

```
SELECT TOP 20 PERCENT LastName, FirstName,
BirthDate
FROM Students;
```

	LastName	FirstName	BirthDate
1	Doe	John	1998-02-11

**Рисунок 1.9.** Использование оператора TOP

Если в таблице вашей базы данных определенное поле содержит большое количество повторяющихся значений, а ваша задача избавиться от них в результирующей таблице, то вы можете использовать для этого оператор **DISTINCT**, который следует написать перед именем требуемого столбца, и тем самым вы исключите из результата все повторения. Например, нам надо определить какие

имена у студентов в «нашем учебном заведении», при этом следует исключить повторения:

```
SELECT DISTINCT FirstName
FROM Students;
```

Результат SQL-запроса представлен на рисунке 1.10.

	FirstName
1	Jack
2	John
3	Joshua

**Рисунок 1.10.** Использование оператора DISTINCT

### 1.3. Предложение WHERE

В тех случаях, когда необходимо отфильтровать получаемые данные, задав определенные условия, в SQL-запросе предусмотрено использование оператора **WHERE**, после которого прописываются требуемые условия. Общая форма записи запроса с использованием оператора **WHERE** выглядит следующим образом:

```
SELECT columnName1, columnName2, ...
FROM tableName
WHERE condition;
```

Для того чтобы в SQL-запросе указать необходимые ограничения, в языке T-SQL используются следующие операторы сравнения:

- = — равно;
- <> — не равно;
- > — больше;
- >= — больше или равно;

- $>$  — не больше чем;
- $<$  — меньше;
- $\leq$  — меньше или равно;
- $\neq$  — не меньше чем.

Например, необходимо вывести стипендию и фамилии всех студентов, у которых размер стипендии превышает 1000. SQL-запрос в этом случае будет выглядеть образом:

```
SELECT LastName, Grants
FROM Students
WHERE Grants > 1000;
```

В результате выполнения этого SQL-запроса вы получите следующую таблицу (Рисунок 1.11).

	Last Name	Grants
1	Jones	1256.00

**Рисунок 1.11.** Использование операторов сравнения

Следующий SQL-запрос позволяет получить информацию о студентах, в имени которых не больше чем четыре символа:

```
SELECT Id, LastName, FirstName, BirthDate, Grants, Email
FROM Students
WHERE LEN(FirstName) !> 4;
```

Для того чтобы определить количество символов (длину строки) в имени студента, мы используем функцию **LEN()**, которая принимает строковые данные. Эта функция является встроенной в MS SQL Server и не входит в стандарт языка T-SQL, то есть ее выполнение обеспечивает сама СУБД. В результате выполнения этого SQL-запроса вы получите следующую таблицу (Рисунок 1.12).

	Id	LastName	FirstName	BirthDate	Grants	Email
1	1	Doe	John	1998-02-11	NULL	jh@net.eu
2	2	Jones	Jack	1997-11-05	1256.00	jj@net.eu

**Рисунок 1.12.** Использование операторов сравнения

Для того чтобы в операторе **WHERE** задать несколько условий, необходимо использовать операторы объединения **AND** и **OR**.

Оператор **AND** объединяет два условия и возвращает истину, если они оба верны. Например, требуется получить информацию о студентах, которые родились осенью, такой SQL-запрос может выглядеть следующим образом:

```
SELECT LastName, FirstName, BirthDate
FROM Students
WHERE MONTH(BirthDate) >= 9 AND MONTH(BirthDate) <= 11;
```

Для того чтобы определить месяц рождения студентов мы воспользовались стандартной функцией языка T-SQL — **MONTH()**, которая позволяет получить номер месяца из переданной даты. Для того чтобы получить год и день, как часть даты, вы можете воспользоваться стандартными функциями **YEAR()** и **DAY()** соответственно.

В результате выполнения предыдущего SQL-запроса вы получите следующую таблицу (Рисунок 1.13).

	LastName	FirstName	BirthDate
1	Jones	Jack	1997-11-05

**Рисунок 1.13.** Использование оператора AND

Оператор **OR** объединяет два условия и возвращает истину, если хотя бы одно из условий верно. Например,

необходимо отобразить информацию о студентах, у которых либо четный год рождения, либо нечетный день рождения. Для того чтобы получить требуемый результат необходимо осуществить проверку соответствующих значений на кратность числу два, используя операцию деление по модулю (%).

```
SELECT LastName, FirstName, BirthDate
FROM Students
WHERE YEAR(BirthDate) % 2 = 0 OR DAY(BirthDate) % 2 <> 0;
```

В результате выполнения предыдущего SQL-запроса вы получите следующую таблицу (Рисунок 1.14).

	LastName	FirstName	BirthDate
1	Doe	John	1998-02-11
2	Jones	Jack	1997-11-05
3	Johnson	Joshua	1997-05-23

**Рисунок 1.14.** Использование оператора OR

В тех случаях, когда вам требуется проверить значения столбцов на неопределенность (значение `NULL`), использование операторов сравнения (`=`, `<>`, `>`, `<`, и т.д.) невозможно. Вместо них вам необходимо использовать оператор `IS NULL`. Например, для того чтобы получить список всех студентов, которые не получают стипендию, необходимо написать следующий запрос к базе данных:

```
SELECT LastName, FirstName, Grants
FROM Students
WHERE Grants IS NULL;
```

Результат SQL-запроса представлен на рисунке 1.15.

	Last Name	First Name	Grants
Id			
1	Doe	John	NULL
2	Johnson	Joshua	NULL

**Рисунок 1.15.** Использование оператора IS NULL

Существует еще один оператор — **NOT**, который используется, когда необходимо задать противоположное условие, допустим, требуется получить информацию о студентах, фамилии которых указывались при заполнении таблицы **Students**, то есть не равны **Doe**.

```
SELECT Id, LastName, FirstName, BirthDate, Grants, Email
FROM Students
WHERE NOT LastName = 'Doe';
```

Результат SQL-запроса представлен на рисунке 1.16.

	Id	Last Name	First Name	Birth Date	Grants	Email
1	2	Jones	Jack	1997-11-05	1256.00	jj@net.eu
2	7	Johnson	Joshua	1997-05-23	NULL	jo@net.eu

**Рисунок 1.16.** Использование оператора NOT

Синтаксис использования оператора **NOT** с **NULL**-значениями несколько отличается. Допустим, нам необходима информация обо всех студентах, которые получают стипендию, в этом случае SQL-запрос будет выглядеть следующим образом:

```
SELECT LastName, FirstName, Grants
FROM Students
WHERE Grants IS NOT NULL;
```

Результаты запроса будут, естественно, диаметрально противоположны запросу показанному на рисунке 1.15 (Рисунок 1.17).

	Last Name	First Name	Grants
1	Jones	Jack	1256.00

**Рисунок 1.17.** Использование оператора NOT с NULL-значениями

## 1.4. Предложение ORDER BY

При написании некоторых SQL-запросов существует необходимость упорядочить полученную информацию, тогда при написании запросов следует использовать предложение ORDER BY. Запись запроса с использованием оператора ORDER BY выглядит следующим образом:

```
SELECT columnName1, columnName2, ...
FROM tableName
ORDER BY columnName1 ASC | DESC, ...;
```

При помощи оператора ORDER BY можно указать количество столбцов, по которым производится сортировка и направление этой сортировки. Чтобы полученные данные были отсортированы по убыванию необходимо указать ключевое слово DESC (от *descending*) после соответствующего поля. Сортировка по возрастанию является сортировкой по умолчанию и ее можно не указывать, но если вы хотите сделать это явным образом, то следует воспользоваться ключевым словом ASC (от *ascending*).

В качестве примера использования оператора ORDER BY приведем SQL-запрос, возвращающий список студентов, отсортированный по дате рождения (Рисунок 1.18):

```
SELECT LastName, FirstName, BirthDate
FROM Students
ORDER BY BirthDate;
```

	LastName	FirstName	BirthDate
1	Johnson	Joshua	1997-05-23
2	Jones	Jack	1997-11-05
3	Doe	John	1998-02-11

Рисунок 1.18. Использование сортировки в SQL-запросе

При написании SQL-запроса существует возможность указать в операторе **ORDER BY** несколько полей с различным направлением сортировки. Допустим, полученный в результате выполнения запроса, список студентов надо отсортировать как по убыванию (столбец **LastName**), так и по возрастанию (столбец **FirstName**), для этого необходимо написать следующий SQL-запрос:

```
SELECT LastName, FirstName, BirthDate
FROM Students
ORDER BY LastName DESC, FirstName ASC;
```

Сортировка данных будет осуществляться следующим образом: все результаты будут отсортированы по фамилии и только в том случае, когда значение столбца **LastName** совпадет более чем в одной результирующей строке, запустится сортировка по имени студента (Рисунок 1.19).

	LastName	FirstName	BirthDate
1	Jones	Jack	1997-11-05
2	Johnson	Joshua	1997-05-23
3	Doe	John	1998-02-11

Рисунок 1.19. Использование сортировки по нескольким столбцам

## 2. Ключевые слова IN, BETWEEN, LIKE

При написании различных SQL-запросов существует потребность в указании сложного условия фильтрации полученных результатов с использованием нескольких операторов OR. Например, необходимо получить информацию о студентах, которых зовут John, David или Jack:

```
SELECT LastName, FirstName, BirthDate, Grants, Email
FROM Students
WHERE FirstName='John' OR FirstName='David'
OR FirstName='Jack';
```

Данный SQL-запрос вернет правильный результат (Рисунок 2.1), но в нем довольно часто используется оператор OR.

	LastName	FirstName	BirthDate	Grants	Email
1	Doe	John	1998-02-11	NULL	jh@net.eu
2	Jones	Jack	1997-11-05	1256.00	jj@net.eu

**Рисунок 2.1.** Запрос, использующий  
несколько операторов OR

Для того чтобы улучшить читабельность подобных запросов можно использовать ключевое слово IN, которое обеспечивает сравнение значения указанного столбца с любым значением из указанного множества. Переписав предыдущий запрос, мы получим тот же результат (Рисунок 2.1).

```
SELECT LastName, FirstName, BirthDate, Grants, Email
FROM Students
WHERE FirstName IN ('John', 'David', 'Jack');
```

При написании SQL-запросов, в которых требуется проверка на попадание значения определенного столбца в некий диапазон можно использовать оператор **AND**, а можно, в качестве альтернативы, применить ключевое слово **BETWEEN**. Ключевое слово **BETWEEN** прописывается после имени столбца, а после него необходимо указать нижнюю и верхнюю границу диапазона, разделенных оператором **AND**, границы диапазона также учитываются.

Допустим, необходимо определить студентов, которые родились в 1997 году. Следующие два запроса вернут одинаковый результат (Рисунок 2.2).

SQL-запрос, использующий оператор **AND**:

```
SELECT LastName, FirstName, BirthDate
FROM Students
WHERE BirthDate >= '1997-01-01'
AND BirthDate <= '1997-12-31';
```

SQL-запрос с использованием ключевого слова **BETWEEN**:

```
SELECT LastName, FirstName, BirthDate
FROM Students
WHERE BirthDate BETWEEN '1997-01-01'
AND '1997-12-31';
```

	Last Name	First Name	Birth Date
1	Jones	Jack	1997-11-05
2	Johnson	Joshua	1997-05-23

**Рисунок 2.2.** Результат попадания информации в указанный диапазон

Следует обратить ваше внимание на синтаксис записи даты рождения: дата указывается в последовательно-

сти год-месяц-день и берется в одинарные кавычки, как строковые данные. Такое представление даты определено драйвером ODBC (*Open Database Connectivity*), который обеспечивает взаимодействие приложений, написанных на различных языках программирования, с данными SQL Server.

Ключевое слово **BETWEEN** можно применять в SQL-запросах, требующих сравнения строковых значений, при этом сравнение будет осуществляться в соответствии с кодами символов. Также с ключевым словом **BETWEEN** можно использовать оператор **NOT**, например, необходимо вывести всех студентов, фамилии которых не начинаются на буквы **E**, **F** и **G**.

```
SELECT LastName, FirstName  
FROM Students  
WHERE LastName NOT BETWEEN 'E' AND 'G';
```

Результат, полученный после выполнения этого SQL-запроса, отображен на рисунке 2.3.

	Last Name	First Name
1	Doe	John
2	Jones	Jack
3	Johnson	Joshua

**Рисунок 2.3.** Использование ключевого слова **BETWEEN** с оператором **NOT**

В тех случаях, когда необходимо осуществить поиск по текстовым полям таблиц, существует возможность использовать шаблон, который можно задать, применив ключевое слово **LIKE**. Для того чтобы сформировать ша-

блон необходимо указывать определенные служебные символы или комбинации этих символов:

- % — соответствует любой последовательности символов от 0 и более;
- \_ — представляет любой одиночный символ;
- [] — задает последовательность или диапазон возможных символов;
- [^] — задает последовательность или диапазон символов, которые должны отсутствовать.

Продемонстрируем использование ключевого слова **LIKE** на следующих примерах. При помощи первого SQL-запроса мы хотим получить всех студентов, фамилия которых начинается на букву **J** или **M**, а имя заканчивается на букву **k**:

```
SELECT LastName, FirstName, BirthDate, Grants, Email
FROM Students
WHERE LastName LIKE '[JM]%' AND FirstName LIKE '%k';
```

Результат выполнения данного SQL-запроса вы можете увидеть на рисунке 2.4.

	LastName	FirstName	BirthDate	Grants	Email
1	Jones	Jack	1997-11-05	1256.00	jj@net.eu

**Рисунок 2.4.** Использование ключевого слова **LIKE**

Следующий запрос вернет информацию о студентах, у которых в адресе электронной почты вторая буква не совпадает ни с одной буквой из диапазона от **f** до **m**:

```
SELECT LastName, FirstName, BirthDate, Grants, Email
FROM Students
WHERE Email LIKE '_[f-m]%' ;
```

## 2. Ключевые слова IN, BETWEEN, LIKE

Результат выполнения предыдущего SQL-запроса (Рисунок 2.5).

	LastName	FirstName	BirthDate	Grants	Email
1	Johnson	Joshua	1997-05-23	NULL	jo@net.eu

**Рисунок 2.5.** Использование ключевого слова LIKE

## 3. Оператор INSERT

Во втором уроке, при создании таблицы мы заполняли ее значениями, используя средства MS SQL Server Management Studio. Однако довольно часто будет возникать необходимость заполнения таблиц программным путем и в этом вам поможет оператор **INSERT**.

Оператор **INSERT** может использоваться двумя способами. В первом случае необходимо указывать имена тех столбцов таблицы, в которые вы будете записывать соответствующие значения, указанные после ключевого слова **VALUES**, общая форма записи выглядит следующим образом:

```
INSERT INTO tableName (columnName1,  
                      columnName2, ...)  
VALUES (value1, value2, ...);
```

Если использовать такую форму оператора **INSERT**, то вам необходимо указывать, как минимум, все столбцы таблицы, которые не могут принимать неопределенное (**NULL**) значение. В противном случае при попытке вставить данные Management Studio выдаст ошибку (Рисунок 3.1).

При написании такого вида запросов не обязательно соблюдать правильный порядок следования имен столбцов в таблице. В следующем SQL-запросе мы укажем имена требуемых столбцов в произвольной последовательности, и не будем указывать столбцы, которые могут хранить неопределенные значения (Рисунок 3.2).

### 3. Оператор INSERT

SQLQuery1.sql - DE..ESKTOP\Yuriy (55)\* X

```
INSERT INTO Students(LastName, FirstName)
VALUES ('Wilson', 'Lily');
```

110 %

Messages

Msg 515, Level 16, State 2, Line 1  
Cannot insert the value NULL into column 'BirthDate',  
table 'University.dbo.Students'; column does not allow nulls.  
INSERT fails.  
The statement has been terminated.

**Рисунок 3.1.** Ошибка: невозможно вставить NULL-значение в столбец BirthDate

SQLQuery1.sql - DE..ESKTOP\Yuriy (55)\* X

```
INSERT INTO Students(BirthDate, FirstName, LastName)
VALUES ('1998-05-25', 'Lily', 'Wilson');
```

110 %

Messages

(1 row affected)

**Рисунок 3.2.** Использование оператора INSERT

Хотя вы видите уведомление MS SQL о корректном выполнении запроса (**1 row affected**), убедимся в правильности добавления данных самостоятельно, выполнив следующий запрос, результат на рисунке 3.3.

```
SELECT LastName, FirstName, BirthDate, Grants, Email
FROM Students;
```

	LastName	FirstName	BirthDate	Grants	Email
1	Doe	John	1998-02-11	NULL	jh@net.eu
2	Jones	Jack	1997-11-05	1256.00	jj@net.eu
3	Johnson	Joshua	1997-05-23	NULL	jo@net.eu
4	Wilson	Lily	1998-05-25	NULL	NULL

**Рисунок 3.3.** Проверка правильности добавления данных

При использовании оператора **INSERT** вторым способом названия столбцов таблицы не указываются, общая форма записи выглядит следующим образом:

```
INSERT INTO tableName  
VALUES (value1, value2, ...);
```

Если вы будете использовать такую форму оператора **INSERT**, то вам необходимо указывать значения для всех столбцов, которые не могут принимать неопределенное значение, обязательно соблюдая порядок их следования в таблице.

Допустим нам требуется добавить в таблицу студенку, которая не получает стипендию, но адрес ее электронной почты нам известен. В этом случае нам необходимо явно указать **NULL**-значение для стипендии, иначе в таблицу будут записаны неправильные данные:

```
INSERT INTO Students  
VALUES ('Evans', 'Grace', '1998-08-30',  
NULL, 'eg@net.eu');
```

Убедиться в правильности добавления записи вы можете самостоятельно, выполнив уже знакомый SQL-запрос (см. выше).

В языке T-SQL существует возможность заполнения существующей таблицы значениями, полученными в результате выполнения SQL-запроса. Для этих целей используется оператор **INSERT INTO SELECT**, применение которого осуществляется двумя способами.

В первом случае мы копируем все данные из таблицы, являющейся источником данных, и выглядит это следующим образом:

```
INSERT INTO destinationTable  
SELECT * FROM sourceTable  
WHERE condition;
```

Вторая форма записи предусматривает явное указание, как названий столбцов таблицы-источника, подлежащих копированию, так и названий столбцов таблицы-приемника данных:

```
INSERT INTO destinationTable (columnName1, columnName2, ...)  
SELECT columnName1, columnName2, ...  
FROM sourceTable  
WHERE condition;
```

Соответствующие столбцы этих таблиц не обязательно должны иметь одинаковые названия, необходимым требованием при использовании оператора **INSERT INTO SELECT** является соответствие типов данных в соответствующих столбцах исходной и целевой таблиц.

Для того чтобы проверить как это работает нам необходимо, для начала, создать в нашей базе данных **University** временную таблицу, которую мы назовем, например, **Temp**. Создать такую таблицу вы можете самостоятельно, освежив свои знания из урока №2. Название полей и их количество в таблице **Temp** не обязательно должно повторять все поля существующей таблицы **Students**, ограничимся только несколькими из них (Рисунок 3.4).

После того как мы создали временную таблицу **Temp**, заполним ее только теми сведениями о студентах (из таблицы **Students**), которые родились во второй половине года, требуемый SQL-запрос будет выглядеть следующим образом (Рисунок 3.5).

	Column Name	Data Type	Allow Nulls
Id	int	<input type="checkbox"/>	
LName	nvarchar(50)	<input checked="" type="checkbox"/>	
FName	nvarchar(50)	<input checked="" type="checkbox"/>	
BirthDate	nchar(10)	<input checked="" type="checkbox"/>	

Рисунок 3.4. Создание временной таблицы

```

SQLQuery1.sql - DE...ESKTOP\Yuriy (56)*  X
[ ] INSERT INTO Temp (LName, FName, BirthDate)
[ ]   SELECT LastName, FirstName, BirthDate
[ ]   FROM Students
[ ]   WHERE MONTH(BirthDate) > 6;
110 %  ◀ ▶
Messages
(2 rows affected)

```

Рисунок 3.5. Заполнение временной таблицы требуемыми данными

Выполнив следующий запрос к таблице `Temp`, мы убедимся в правильности добавления информации (Рисунок 3.6).

```

SELECT LName, FName, BirthDate
FROM Temp;

```

	LName	FName	BirthDate
1	Jones	Jack	1997-11-05
2	Evans	Grace	1998-08-30

Рисунок 3.6. Проверка правильности добавления данных

СУБД MS SQL Server предоставляет возможность создания новой таблицы на основе существующей, путем копирования значений всех указанных столбцов с ис-

пользованием оператора **SELECT INTO**, общая форма записи выглядит следующим образом:

```
SELECT columnName1, columnName2, ...
INTO newTable
FROM existingTable
WHERE condition;
```

Создадим в нашей базе данных **University** новую таблицу с информацией о студентах, электронный адрес которых известен, выполнив следующий запрос (Рисунок 3.7).

```
SQLQuery1.sql - DE...ESKTOP\Yuriy (57)*
SELECT LastName, FirstName, BirthDate, Email
INTO Temp1
FROM Students
WHERE Email IS NOT NULL;
110 %
Messages
(4 rows affected)
```

**Рисунок 3.7.** Создание новой таблицы на основе существующей

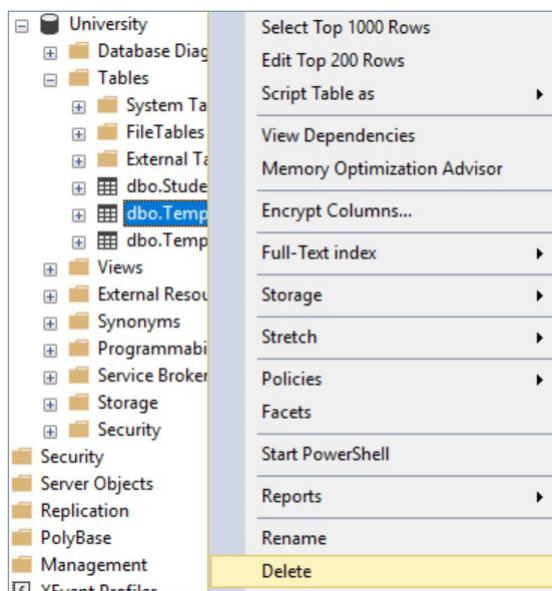
Для того чтобы проверить полученный результат, в следующем SQL-запросе мы специально не будем перечислять названия столбцов, результат на рисунке 3.8.

```
SELECT *
FROM Temp1;
```

	Last Name	First Name	Birth Date	Email
1	Doe	John	1998-02-11	jh@net.eu
2	Jones	Jack	1997-11-05	jj@net.eu
3	Johnson	Joshua	1997-05-23	jo@net.eu
4	Evans	Grace	1998-08-30	eg@net.eu

**Рисунок 3.8.** Проверка информации, записанной в новую таблицу

После того как мы продемонстрировали использование операторов **INSERT INTO SELECT** и **SELECT INTO**, удалим ненужные нам таблицы **Temp** и **Temp1** из базы данных **University**. Для этого необходимо нажать правой клавишей мыши на имени соответствующей таблицы в списке **Tables** базы данных **University** окна Object Explorer и в контекстном меню выбрать пункт Delete (Рисунок 3.9)

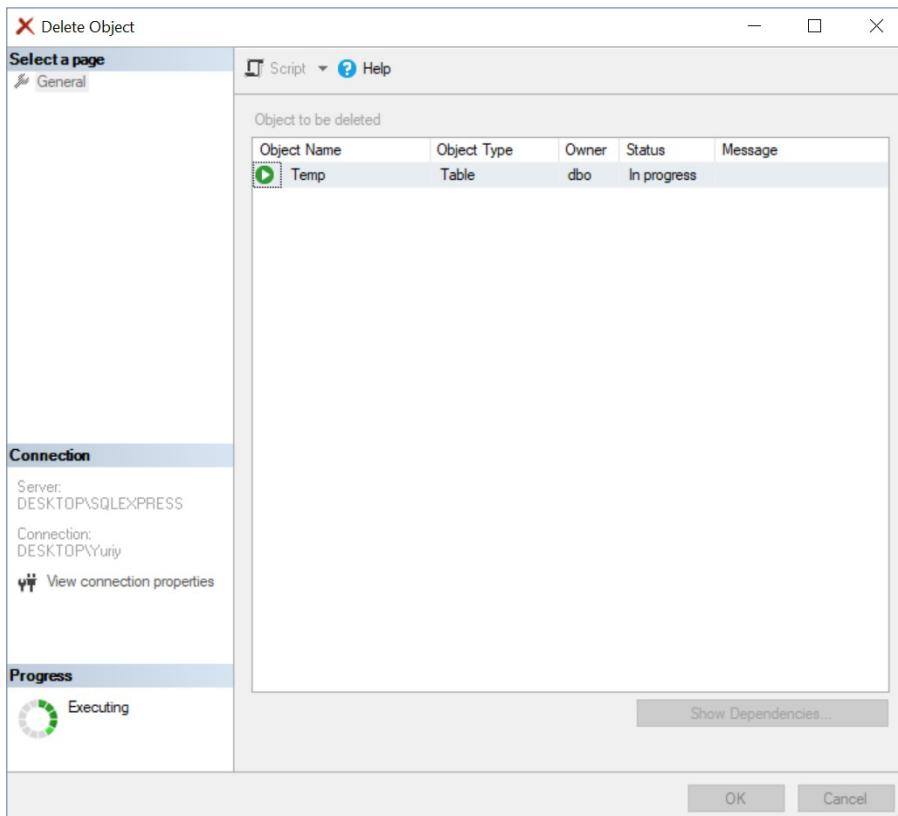


**Рисунок 3.9.** Удаление таблицы из базы данных (начало)

После этого появится окно Delete Object, в котором после нажатия кнопки OK вы увидите индикатор процесса удаления таблицы (Рисунок 3.10).

После того как окно закроется, таблица исчезнет из списка **Tables** вашей базы данных.

### 3. Оператор INSERT



**Рисунок 3.10.** Удаление таблицы из базы данных (завершение)

## 4. Оператор UPDATE

Для того чтобы изменить любую запись при помощи SQL-запроса используется оператор **UPDATE**, общая форма записи которого выглядит следующим образом:

```
UPDATE tableName  
SET columnName1 = value1, columnName2 = value2, .  
WHERE condition;
```

Использование оператора **WHERE** можно считать обязательным, ведь если вы не укажете условие, то изменятся значения в соответствующем столбце по всей таблице.

Смоделируем определенную фантастическую ситуацию: «ректор нашего университета» решил выплачивать минимальную стипендию всем студентам независимо от их успеваемости, так сказать, на покупку тетрадок, естественно нельзя оставить без внимания студентов, которые хорошо учатся — им надо повысить стипендию. Такая инициатива должна привести к изменению информации в базе данных, то есть нам необходимо выполнить запросы с использованием оператора **UPDATE**, которые будут выглядеть следующим образом (Рисунок 4.1).

В данном примере очень важен порядок выполнения SQL-запросов, если вначале установим стипендию отстающим студентам, тогда после увеличения стипендии всем студентам, которые учатся хорошо, у плохо успевающих студентов стипендия увеличится в два раза.

## 4. Оператор UPDATE

The screenshot shows a SQL query window titled 'SQLQuery1.sql - DE...ESKTOP\Yuriy (56)\*'. It contains the following SQL code:

```
-- increase of the scholarship  
UPDATE Students SET Grants += 500 WHERE Grants IS NOT NULL;  
  
-- appointment of a scholarship  
UPDATE Students SET Grants = 500 WHERE Grants IS NULL;
```

Below the code, the 'Messages' pane displays the results of the execution:

```
(1 row affected)  
(4 rows affected)
```

**Рисунок 4.1.** Изменение информации в таблице

Посмотрим, какие данные сейчас хранятся в столбце Grants таблицы Students (Рисунок 4.2).

```
SELECT LastName, FirstName, Grants  
FROM Students;
```

	LastName	FirstName	Grants
1	Doe	John	500.00
2	Jones	Jack	1756.00
3	Johnson	Joshua	500.00
4	Wilson	Lily	500.00
5	Evans	Grace	500.00

**Рисунок 4.2.** Проверка правильности изменения данных

## 5. Оператор DELETE

Для того чтобы удалить определенную информацию из таблиц базы данных программным путем необходимо использовать оператор **DELETE**, общая форма записи выглядит следующим образом:

```
DELETE FROM tableName  
WHERE condition;
```

Использовать оператор **DELETE** необходимо с особой осторожностью, так как можно случайно удалить очень важную информацию. Если вы не будете устанавливать граничное условие при помощи оператора **WHERE**, то ваш SQL-запрос удалит всю информацию из текущей таблицы.

Приведем пример использования оператора **DELETE**, допустим, нам необходимо удалить все записи из таблицы **Students**, у которых уникальный идентификатор (**Id**) больше 9 (Рисунок 5.1).

```
SQLQuery1.sql - DE...ESKTOP\Yuriy (56)* X
[ ] DELETE FROM Students
[ ] WHERE Id > 9;
110 % < >
Messages
(2 rows affected)
```

**Рисунок 5.1.** Удаление информации из таблицы

Как вы заметили, в данном случае были удалены две записи (**2 row affected**), при этом результат выполнения SQL-запроса в вашей базе данных может отличаться.

# 6. Понятие транзакции. Использование транзакций

Классический пример транзакции из повседневной жизни — перевод денежных средств с одного счета на другой. Естественно все хотят, чтобы деньги после снятия со счета-отправителя попали на счет-получателя, но если в момент осуществления этой операции произойдет сбой в системе, то необходимо чтобы деньги вернулись на счет-отправителя. Вообще транзакции окружают нас повсюду, и вы можете сами привести их примеры, подумайте.

Под транзакцией понимается последовательность действий, выполняемых как единое целое с возможность отмены каждого из них в случае ошибки.

СУБД MS SQL Server обеспечивает поддержку транзакций, которая характеризуется четырьмя важными свойствами известными под акронимом ACID:

- **Atomicity (атомарность)** — определяет целостность транзакции, то есть для успешного завершения транзакции должны выполниться либо все операции, составляющие данную транзакцию, либо ни одна из них;
- **Consistency (согласованность)** — это свойство обеспечивает целостность информации независимо от того успешно завершилась транзакция или нет;
- **Isolation (изолированность)** — означает, что все транзакции выполняются параллельно и не могут оказывать влияния друг на друга;

- **Durability** (*надежность*) — обеспечивает сохранность всех данных после фиксации успешно выполненной транзакции, независимо от возможных сбоев системы.

Существует три типа транзакций:

- **явные транзакции** — характеризуются явным указанием начала транзакции (`BEGIN TRAN` или `BEGIN TRANSACTION`), ее фиксации в случае успешного завершения (`COMMIT TRAN` или `COMMIT TRANSACTION`) и с возможностью прерывания (отката) транзакции при возникновении определенного условия (`ROLLBACK TRAN` или `ROLLBACK TRANSACTION`);
- **неявные транзакции** — транзакции, для которых начало и конец не указываются явным образом, такие транзакции происходят при выполнении следующих инструкций — `ALTER TABLE`, `CREATE`, `DROP`, `SELECT`, `INSERT`, `DELETE`, `UPDATE`, `GRANT`, `REVOKE`, `OPEN`, `FETCH`, `TRUNCATE TABLE` — каждая из них выполняется как отдельная транзакция;
- **автоматические транзакции** характерны тем что каждая успешно выполненная операция фиксируется, в противном случае откатывается, для того чтобы перейти в этот режим необходимо выполнить команду `SET IMPLICIT_TRANSACTION` с параметром `OFF`, для выхода `SET IMPLICIT_TRANSACTION ON`.

СУБД MS SQL Server создана как многопользовательская система, то есть система, которая позволяет одновременно использовать объекты баз данных нескольким

пользователями. В связи с этим существуют четыре возможные нарушения при работе с базами данных:

- ▷ чтение незафиксированных данных;
- ▷ неповторяемое чтение;
- ▷ фантомы;
- ▷ потерянные обновления.

Для того чтобы предотвратить конфликты, которые могут возникнуть при выполнении различных операций с одним и тем же объектом различными пользователями используются блокировки. MS SQL Server обеспечивает поддержку шести типов блокируемых ресурсов: база данных, таблица, экстент, страница, ключ и строка.

Также существует довольно большое количество типов самих блокировок, перечислим основные из них: разделяемые, исключительные, намеченные, блокировки обновления, блокировки схемы, блокировки массового обновления.

Блокировки, применяемые при выполнении транзакций, препятствуют взаимодействию с объектами другим процессам, тем самым могут служить причиной взаимной изоляции транзакций. Для управления степенью взаимной изоляции транзакций существуют четыре уровня изолированности транзакций:

- **READ UNCOMMITTED** — позволяет считывать данные, которые были изменены, но не были зафиксированы (не предотвращает ни одно из возможных нарушений);
- **READ COMMITTED** — не позволяет считывать данные, которые были изменены, но не были зафик-

сированы (используется по умолчанию) (предотвращает чтение незафиксированных данных);

- **REPEATABLE READ** — никакая транзакция не может изменять данные, считанные текущей транзакцией до ее завершения (предотвращает чтение незафиксированных данных и неповторяемое чтение);
- **SERIALIZABLE** — любые другие транзакции не могут вставлять/изменять/удалять данные, которые попадают в диапазон записей, удовлетворяющих условию, которое задано в операторе **WHERE** текущей транзакции (предотвращает чтение незафиксированных данных, неповторяемое чтение и фантомы).

Более подробная информация о транзакциях и их практическое использование будут рассмотрены в курсе MS SQL Server.

# 7. Домашнее задание

В домашнем задании к уроку №2 необходимо было создать базу данных «Больница» с таблицей «Patients», заполните эту таблицу произвольными данными с использованием оператора **INSERT** (не менее десяти записей) и напишите следующие SQL-запросы:

- вывести информацию обо всех пациентах, находящихся в больнице;
- показать данные о пациентах, которые лежат в определенном отделении;
- вывести названия всех отделений больницы;
- получить данные о пациентах, которые лежат в больнице больше месяца, отсортировав их по возрастанию даты поступления;
- вывести информацию о пациентах, которые были выписаны в прошлом месяце (вам поможет стандартная функция **GETDATE()** языка T-SQL, которая позволяет получить текущую дату);
- показать информацию о пациентах, которые лежали в больнице с октября по декабрь прошлого года в определенном отделении;
- вывести информацию о самом молодом пациенте и его возраст (для написания этого запроса вам следует самостоятельно изучить стандартную функцию **DATEDIFF()**, информацию о которой вы можете получить по следующей ссылке <https://docs.microsoft.com/en-us/sql/t-sql/functions/datediff-transact-sql>);

- получить информацию о пациентах, которые лежат в любых трех отделениях;
- показать всех пациентов, фамилия которых начинается на букву P;
- вывести данные о пациентах, которых лечит определенный врач с одинаковыми заболеваниями;
- получить данные о пациентах, пользующихся услугами определенного мобильного оператора;
- переименовать название определенного отделения;
- удалить всех пациентов, которые были выписаны больше чем полгода назад.

## 7. Домашнее задание

