Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Магнитогорский государственный технический университет им. Г.И. Носова»

И.И. Баранкова
У.В. Михайлова
Г.И. Лукьянов

РАЗРАБОТКА ПРИЛОЖЕНИЙ НА C# ДЛЯ РАБОТЫ С БАЗАМИ ДАННЫХ

*Утверждено Редакционно-издательским советом университета
в качестве практикума*

Магнитогорск
2018

УДК 004.658

Рецензенты:

кандидат технических наук,
заместитель директора СЦ,
ООО «ТЕХНОАП» Инжиниринг»
Д.В. Швидченко

кандидат физико-математических наук,
заведующий кафедрой физики,
ФГБОУ ВО «Магнитогорский государственный технический университет им. Г.И. Носова»
Ю.И. Савченко

Баранкова И.И., Михайлова У.В., Лукьянов Г.И.
Разработка приложений на C# для работы с базами данных [Электронный ресурс] : практикум / Инна Ильинична Баранкова, Ульяна Владимировна Михайлова, Георгий Игоревич Лукьянов ; ФГБОУ ВО «Магнитогорский государственный технический университет им. Г.И. Носова». – Электрон. текстовые дан. (1,36 Мб). – Магнитогорск : ФГБОУ ВО «МГТУ им. Г.И. Носова», 2018. – 1 электрон. опт. диск (CD-R). – Систем. требования : IBM PC, любой, более l GHz ; 512 Мб RAM ; 10 Мб HDD ; MS Windows XP и выше ; Adobe Reader 8.0 и выше ; CD/DVD-ROM дисковод ; мышь. – Загл. с титул. экрана.

Разработанное электронное издание соответствует требованиям ФГОС ВО и образовательной программе по специальности 100503 «Информационная безопасность автоматизированных систем», специализация «Обеспечение информационной безопасности распределенных информационных систем». Данное электронное издание полностью соответствует рабочей программе по дисциплине «Информационные технологии. Базы данных», «Языки программирования», «Технологии и методы программирования», а так же может использоваться в рамках изучения дисциплин: «Информатика».

В данном практикуме рассмотрена разработка приложения, для которого использовали язык программирования C# и СУБД MS SQL Server. Данное электронное издание освещает достаточный минимум, который необходим при работе с базами данных при разработке приложений на C#.

УДК 004.658

Содержание

ВВЕДЕНИЕ

В данном практикуме рассмотрены основы языка SQL на примере MS SQL Server. Рассмотрен достаточный минимум, который необходим при работе с базами данных.

Язык SQL подразделяется на несколько частей, из них 2 наиболее важные его части:

1. DDL – Data Definition Language (язык описания данных).
2. DML – Data Manipulation Language (язык манипулирования данными).

Сами СУБД имеют не удобный интерфейс для обычных пользователей. Так же стоит учитывать, что сервер один и на нем расположена система управления базами данных (СУБД), которая взаимодействует с БД, а устанавливать для каждого компьютера пользователя СУБД будет очень затратной задачей. В этом случае, будет удобно и выгодно разработать пользовательское приложение. Которое будет иметь удобный интерфейс и в нем будут уже заложены сформированные запросы. При этом имелся бы доступ к базе данных(БД) внутри компании через локальную сеть или могли бы работать удаленно.

В данном практикуме так же рассмотрена разработка такого приложения. Для него использовали язык программирования C# и СУБД MS SQL Server. MS Visual Studio позволяет создавать собственные БД, с помощью расширения использовать язык программирования C# и СУБД MS SQL Server. MS Visual Studio позволяет создавать собственные БД с помощью расширения SQLite, но в этом случае придется создавать приложение сервер, где будут обрабатываться данные, и отдельный клиент, который будет взаимодействовать с ним. Поэтому намного эффективней и выгодней будет использовать готовую СУБД + приложение для пользователей.

При создании приложения, которое работает с данными в базе данных, необходимо выполнить такие основные задачи, как определение строк подключения, вставка данных и выполнение хранимых процедур.

1. ЯЗЫК МАНИПУЛИРОВАНИЯ ДАННЫМИ (DML)

DML – Data Manipulation Language. Конструкция SELECT является самой главной конструкцией языка DML, т.к. за счет нее или ее частей осуществляется выборка необходимых данных из базы данных (БД).

Язык DML содержит следующие конструкции:

- SELECT – выборка данных;
- INSERT – вставка новых данных;
- UPDATE – обновление данных;
- DELETE – удаление данных;
- MERGE – слияние данных.

Если язык DDL больше статичен, т.е. при помощи него создаются жесткие структуры (таблицы, связи и т.п.), то язык DML носит динамический характер, здесь правильные результаты вы можете получить разными путями.

Приведенные примеры будут рассмотрены на основе БД Test, которая была рассмотрена в предыдущем пособии (Михайлова У.В., Баранкова И.И., Лукьянов Г.И. «Разработка БД В MS SQL Server с использованием SSMS»).

1.1. SELECT – оператор выборки данных

Рассмотрим, как выглядит оператор SELECT:

```
SELECT[DISTINCT]список_столбцов или *
FROM источник
WHERE фильтр
ORDERBYвыражение_сортировки
```

Вообще стоит отметить, что в диалекте MS SQL самая простая форма запроса SELECT может не содержать блока FROM, в этом случае вы можете использовать ее, для получения каких-то значений (рис. 1):

```
SELECT2+2/2, SYSDATETIME(); -- получение системной даты БД
```

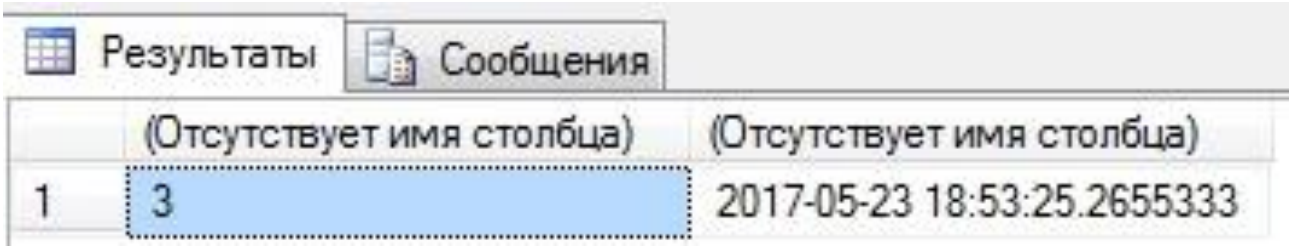


Рис. 1. Результат выполнения запроса

Стоит отметить, что выражение 2+2/2 выводит целое число, рассмотрим примеры получения вещественных чисел:

```
SELECT 12/10, -- 1
12./10, -- 1.2
123/10. - 1.2
```

Примечание. При других арифметических операциях действует та же самая логика, просто в случае деления этот нюанс более актуален. Поэтому обращайте внимание на тип данных числовых столбцов. В том случае если он целый, а результат вам нужно получить вещественный, то используйте преобразование, либо просто ставьте точку после числа указанного в виде константы.

Функцию CAST или CONVERT можно использовать для преобразования типов данных полей(ID - является типомint):

```
CAST(ID ASfloat)/100, -- используемфункциюCAST для преобразования в
тип float
CONVERT(float,ID)/100, -- используем функцию CONVERT для
преобразования в тип float
```

Если у нас есть несколько таблиц в блоке FROMто можно использовать псевдонимы. К примеру, если использовать 2 таблицы Студенты и Институты, то нам придётся использовать Студенты.ID, Институты.ID, для примера сократим Студенты:

```
SELECTemp.ID,emp.ФИО
FROMСтудентыASemp
--или
SELECTemp.ID,emp.ФИО
FROMСтудентыemp
```

Здесь можно использовать одно из двух, т.е. если вы задали псевдоним, то и пользоваться нужно будет им, а использовать имя таблицы уже нельзя.

Так оператор AS или знаком равенства можно задать напрямую для столбцов:

```
SELECT ID AS[Уникальный идентификатор]
FROM Студенты
-- =
SELECT'Уникальный идентификатор'=ID
FROM Студенты
```

1.2. Ключевое слово DISTINCT

Ключевое слово DISTINCT используется для того чтобы отбросить из результата запроса строки дубликаты. Грубо говоря, представьте, что сначала выполняется запрос без опции DISTINCT, а затем из результата выбрасываются все дубликаты.

Вот пример из статьи, все дубликаты помечены одним цветом (рис. 2):

```
SELECT Col1,Col2,Col3
FROM #Trash
```

```
SELECT DISTINCT Col1,Col2,Col3
FROM #Trash
```

Col1	Col2	Col3
A	A	A
A	B	C
C	A	B
A	A	B
B	B	B
A	A	B
A	A	A
C	A	B
C	A	B
A	A	B
A	NULL	B
A	NULL	B

Col1	Col2	Col3
A	NULL	B
A	A	A
A	A	B
A	B	C
B	B	B
C	A	B

Рис. 2. Пример работы DISTINCT

1.3. Предложение ORDERBY

Предложение ORDER BY используется для сортировки результата запроса.

```
SELECT  ФИО, Группа, Институт
FROM Студенты
ORDERBY  ФИО, Группа-- упорядочим по ФИО, а после по группе
```

После имени поля в предложении ORDER BY можно задать опцию DESC, которая служит для сортировки этого поля в порядке убывания:

```
SELECT  ФИО, Группа, Институт
FROM Студенты
ORDERBY  Институт DESC, --Убывание по инст.
ФИО, Группа
```

Для сортировки по возрастанию есть ключевое слово ASC, но так как сортировка по возрастанию применяется по умолчанию, то про эту опцию можно забыть.

Примечание. Стоит отметить, что в предложении ORDER BY можно использовать и поля, которые не перечислены в предложении SELECT (кроме случая, когда используется DISTINCT).

Так же в ORDERBYможно используя разные выражения, к примеру, **CONCAT** - Возвращает строку, являющуюся результатом объединения двух или более строковых значений. Принимает переменное число аргументов

строку и объединяет их в одну строку. Для этого требуется не менее двух входных значений. В противном случае возникает ошибка. Все аргументы неявно преобразуются в строковые типы и затем объединяются. Значения NULL неявно преобразуются в пустую строку. Если все аргументы имеют значение null, возвращается пустая строка типа varchar(1) возвращается.

```
SELECT CONCAT ('Happy ', 'Birthday ', 11, '/', '25')ASResult;
--
SELECT  ФИО, Группа, Институт
FROM Студенты
ORDERBYCONCAT(ФИО,' ',Институт)
```

Стоит отметить, что в случае использования предложения DISTINCT, в предложении ORDERBY могут использоваться только колонки, перечисленные в блоке SELECT. Т.е. после применения операции DISTINCT мы получаем новый набор данных, с новым набором колонок.

```
SELECTDISTINCTФИО, Группа
FROMСтуденты
ORDERBY Институт-- в этом случае не работает
```

1.4. Предложение TOP

TOP – ограничивает число строк, возвращаемых в результирующем наборе запроса до заданного числа или процентного значения. Если предложение TOP используется совместно с предложением ORDER BY, то результирующий набор ограничен первыми N строками отсортированного результата. В противном случае возвращаются первые N строк в неопределенном порядке.

Обычно данное выражение используется с предложением ORDER BY, и без него тоже используют:

```
SELECTTOP2*--возвращаем первые 2 строки
FROM Студенты
```

Так же можно указать слово PERCENT, для того чтобы вернулось процент строк из результирующего набора, к примеру вернуть 20% из общей информации:

```
SELECTTOP20PERCENT*
FROM Студенты
```

Теперь применим с группировкой:

```
SELECTTOP3 ФИО, Группа, Институт
FROM Студенты
ORDERBY  Институт DESC
```

Так же с TOP можно использовать опцию WITH TIES, которая поможет вернуть все строки в случае неоднозначной сортировки. Т.е. это предложение вернет все строки, которые равны по составу строкам, которые попадают в

выборку TOP N, в итоге строк может быть выбрано больше чем N. К примеру, выше добавим WITH TIES и увидим результат (рис. 3):

```
SELECTTOP3WITH TIES ФИО, Группа, Институт
FROM Студенты
ORDERBY  Институт DESC
```

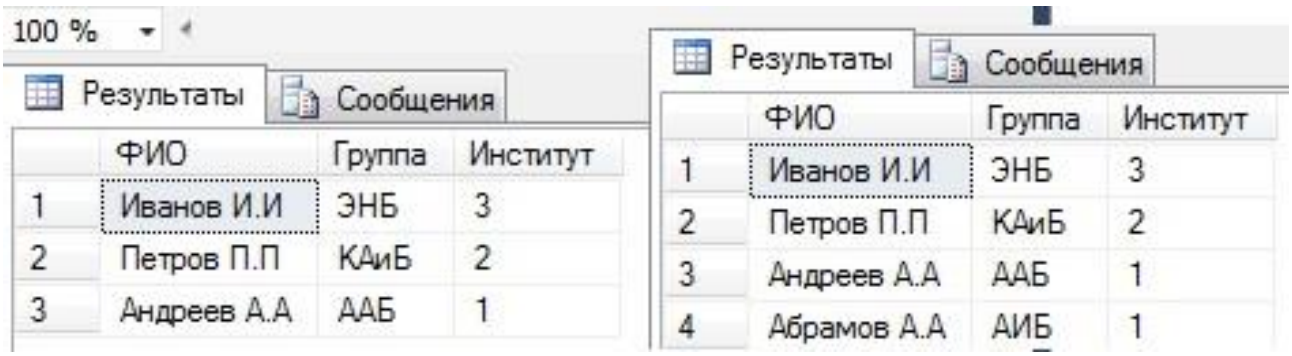


Рис. 3. –Использование предложения TOP с модификатором

В первом случае без WITH TIES, а в другом добавили.

Так же можно применить одновременно предложения DISTINCT и TOP. Порядок слов в операторе SELECT следующий, первым идет DISTINCT, а после него идет TOP, т.е. если рассуждать логически и читать слева-направо, то первым применится отброс дубликатов, а потом уже по этому набору будет сделан TOP.

```
SELECTDISTINCTTOP3WITH TIES Институт
FROMСтуденты
ORDERBYИнститутDESC
```

1.5. Предложение WHERE

Данное предложение служит для фильтрации записей по заданному условию. Предложение WHERE пишется до команды ORDER BY. Порядок применения команд к исходному набору Студенты следующий:

1. WHERE – если указано, то первым делом из всего набора Студенты идет отбор только удовлетворяющих условию записей
2. DISTINCT – если указано, то отбрасываются все дубликаты
3. ORDER BY – если указано, то делается сортировка результата
4. TOP – если указано, то из отсортированного результата возвращается только указанное число записей.

На рис. 4 показано как комбинировать:

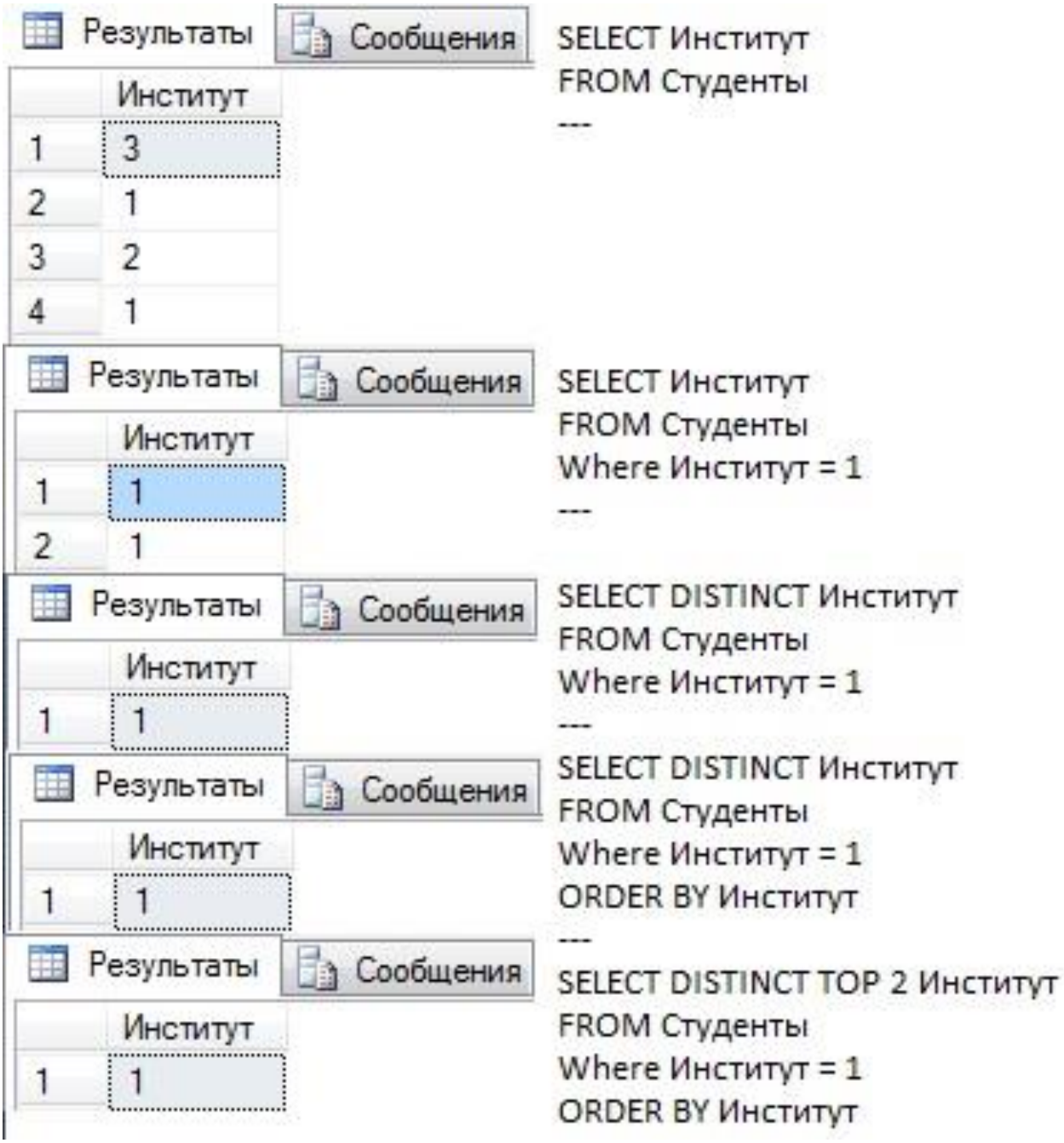


Рис. 4. Использование предложения WHERE

Примечание. Стоит отметить, что проверка на NULL делается не знаком равенства, а при помощи операторов IS NULL и IS NOT NULL.

BETWEEN – проверка на вхождение в диапазон, выглядит так:

```
проверяемое_значение[NOT]BETWEEN начальное_ значение AND конечное_
значение
```

Собственно, BETWEEN это упрощенная запись:

```
WHERE Значение>=20AND Значение<=30
-- на это
WHERE Значение BETWEEN20AND30
```

IN – проверка на вхождение в перечень значений:

```
проверяемое_значение[NOT]IN(значение1, значение2, …)
```

Так же упрощает запись:

```
WHERE Институт =3OR Институт =4
-- на
WHERE Институт IN(3,4)
```

Учтите, что искать NULL значения при помощи конструкции IN не получится, т.к. проверка NULL=NULL вернет так же NULL, а не True.

Примечание. Так же стоит упомянуть еще более коварную ошибку, связанную с NULL, которую можно допустить при использовании конструкции NOT IN. К примеру, выберем тех студентов, которых не состоят в 1 института или тех, у кого не указан институт (значение имеет NULL):

```
SELECT*
FROMСтуденты
WHEREИнститутNOTIN(1,NULL)
```

Но в итоге мы не получим не одной записи. Перепишем правильно в блоке WHERE:

```
SELECT*
FROMСтуденты
WHEREИнститутNOTIN(1)
ANDИнститутISNOTNULL
```

LIKE – проверка строки по шаблону. Этот оператор имеет следующий вид:

```
проверяемая_строка[NOT]LIKEстрока_шаблон[ESCAPEотменяющий_символ]
```

В «строке_шаблон» могут применяться следующие специальные символы:

- Знак подчеркивания «_» — говорит, что на его месте может стоять любой единичный символ.
- Знак процента «%» — говорит, что на его месте может стоять сколько угодно символов, в том числе и ни одного.

```
SELECT*
FROMСтуденты
--
WHEREФИОLIKE'Абр%'
--
WHEREФИОLIKE'%ов'
--
WHEREФИОLIKE'%ра%'
--
WHERE ФИО LIKE'_етров'-- 1 любой символ
--
WHERE ФИО LIKE'__тров'-- 2 любых символов
```

При помощи ESCAPE можно задать отменяющий символ, который отменяет проверяющее действие специальных символов «_» и «%». Данное предложение используется, когда в строке нужно непосредственно проверить наличие знака процента или знака подчеркивания.

К примеру, у нас в таблице ФИО содержит знак «%» и «_»: «Мих_М%М». То блок WHERE будет выглядеть так:

```
SELECT*
FROM Студенты
--
WHERE ФИО LIKE'%!_%'ESCAPE'!'-- строка содержит знак "_"
--
WHERE ФИО LIKE'%!%%'ESCAPE'!'-- строка содержит знак "%"
```

В случае проверки строки на наличие Unicode символов, нужно будет ставить перед кавычками символ N, т.е. N'…'. Но так как у нас в таблице все символьные поля в формате Unicode (тип nvarchar), то для этих полей можно всегда использовать : «N'Абр%'»

Если делать правильно, при сравнении с полем типа varchar (ASCII) нужно стараться использовать проверки с использованием '…', а при сравнении поля с типом nvarchar (Unicode) нужно стараться использовать проверки с использованием N'…'. Это делается для того, чтобы избежать в процессе выполнения запроса неявных преобразований типов. То же самое правило используем при вставке (INSERT) значений в поле или их обновлении (UPDATE).

Вне зависимости от региональных настроек в MSSQL можно использовать следующий синтаксис дат 'YYYYMMDD' (год, месяц, день слитно без пробелов). Такой формат даты MSSQL поймет всегда:

```
SELECT*
FROM Студенты
WHEREДРBETWEEN'19950101'AND'19961231;
```

В некоторых случаях, дату удобнее задавать при помощи функции DATEFROMPARTS:

```
SELECT*
FROMСтуденты
WHERE ДР BETWEEN DATEFROMPARTS(1995,01,01)AND
DATEFROMPARTS(1996,12,31)
```

Так же есть аналогичная функция DATETIMEFROMPARTS, которая служит для задания Даты и Времени.

Еще вы можете использовать функцию CONVERT, если требуется преобразовать строку в значение типа date или datetime:

```
CONVERT(date,'12.03.2015',104),
CONVERT(datetime,'2014-11-30 17:20:15',120)
```

Значения 104 и 120 указывают, какой формат даты используется в строке. Описание всех допустимых форматов вы можете найти в библиотеке MSDN.

1.6. Оператор CASE

Данный оператор позволяет осуществить проверку условий и возвратить в зависимости от выполнения того или иного условия тот или иной результат (таблица 1).

Таблица 1

Формы оператора Case

Первая форма:	**Вторая форма:**
CASE WHEN условие_1 THEN возвращаемое_значение_1 … WHEN условие_N THEN возвращаемое_значение_N [ELSE возвращаемое_значение] END	CASE проверяемое_значение WHEN сравниваемое_значение_1 THEN возвращаемое_значение_1 … WHEN сравниваемое_значение_N THEN возвращаемое_значение_N [ELSE возвращаемое_значение] END

Рассмотрим пример в первом случае с ELSE и без:

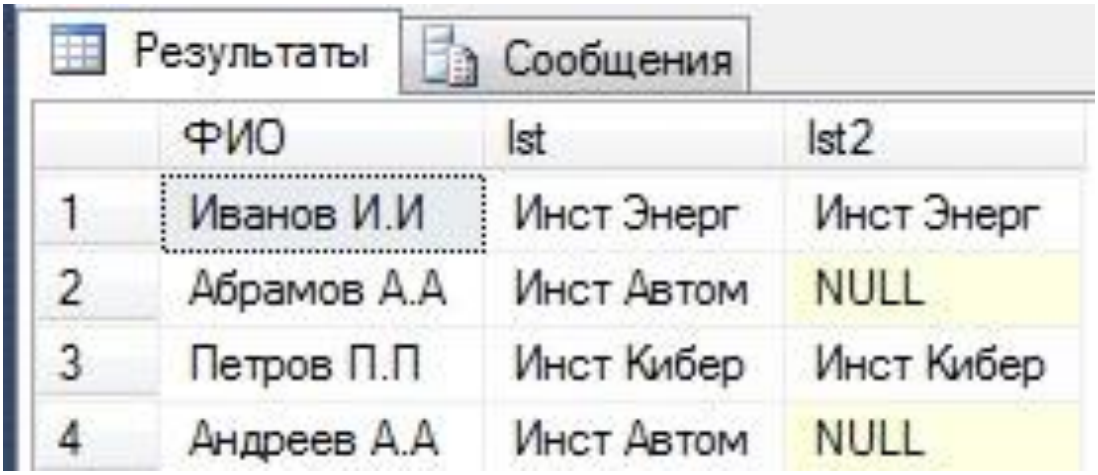
```
SELECTФИО,

CASE
WHENИнститут=3THEN'ИнстЭнерг'
WHENИнститут=2THEN'ИнстКибер'
ELSE'ИнстАвтом'
ENDIst,

CASE
WHENИнститут=3THEN'ИнстЭнерг'
WHENИнститут=2THEN'ИнстКибер'
END Ist2


FROMСтуденты
```

В итоге получим (рис. 5):



	ФИО	Ist	Ist2
1	Иванов И.И	Инст Энерг	Инст Энерг
2	Абрамов А.А	Инст Автом	NULL
3	Петров П.П	Инст Кибер	Инст Кибер
4	Андреев А.А	Инст Автом	NULL

Рис. 5. Результат выполнения оператора CASE

WHEN-условия проверяются последовательно, сверху-вниз. При достижении первого удовлетворяющего условия дальнейшая проверка прерывается и возвращается значение, указанное после слова THEN, относящегося к данному блоку WHEN.

Если ни одно из WHEN-условий не выполняется, то возвращается значение, указанное после слова ELSE. Если ELSE-блок не указан и не выполняется ни одно WHEN-условие, то возвращается NULL.

Для второй формы:

```
CASE Институт
WHEN3THEN'ИнстЭнерг'
WHEN2THEN'ИнстКибер'
ELSE'ИнстАвтом'
ENDIst
```

Теперь рассмотрим еще один пример, где студентам делаю добавки к стипендии в процентах, и узнаем, какие добавки получаться для каждого из них. Пример будет простой, создадим временную таблицу(ID, Name, Sl – текущая стипендия) и заполним его 3 студентами. Добавлять стипендию будем по ID:

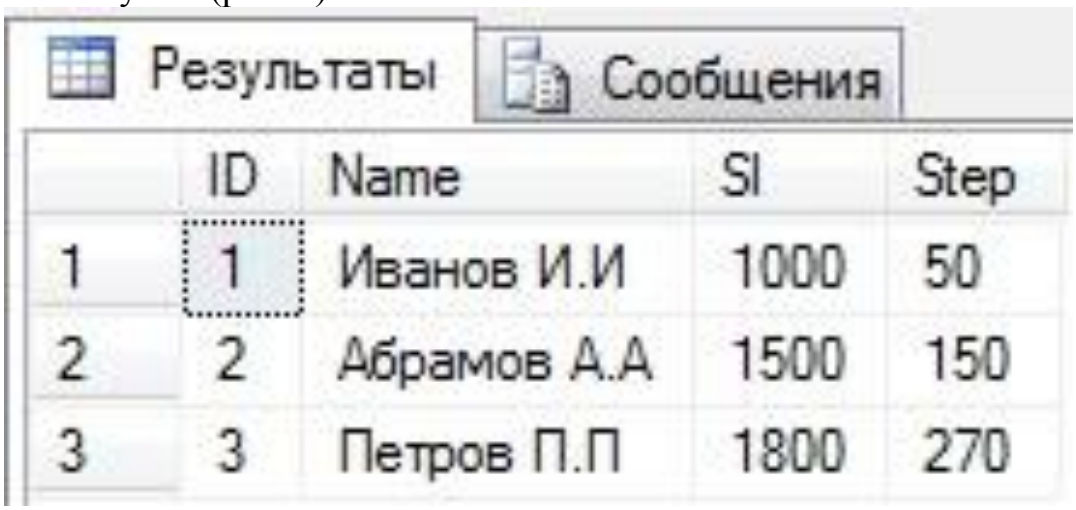
```
CreateTABLE #Test1 --создаем  временную таблицу
(
 ID int,
 Name nvarchar(30),
Slint
)

INSERTTest1(ID, Name, Sl)VALUES-- заполняемего
(1, N'Иванов И.И', 1000),
(2, N'Абрамов А.А', 1500),
(3, N'ПетровП.П', 1800)
-- выводимего
Select ID, Name, Sl,
Sl/100*
CASE ID
WHEN2THEN10-- 10%
```

```
WHEN3THEN15-- 15%
ELSE5-- остальнымпо 5%
end Step
From #Test1

DROPTABLE #Test1 --удаляемтабл
```

В итоге получим (рис. 6):



Результаты | Сообщения

	ID	Name	Sl	Step
1	1	Иванов И.И	1000	50
2	2	Абрамов А.А	1500	150
3	3	Петров П.П	1800	270

Рис. 6. Результат выполнения запроса

Так что, вторая форма – это всего лишь упрощенная запись для тех случаев, когда нам нужно сделать сравнение на равенство, одного и того же проверяемого значения с каждым WHEN-значением/выражением.

Есть упрощенная форма записи IIF, она есть только MSSQLc 2012 г, на предыдущих версиях она отсутствует, и скорее всего в других СУБД тоже. Она может использоваться для упрощенной записи конструкции CASE, в том случае если возвращаются только 2 значения. Конструкция IIF имеет следующий вид:

```
IIF(условие, true_значение, false_значение)
-- аналог
CASEWHENусловиеTHEN true_значениеELSE false_значениеEND
```

Если сравним, то примерно будетследуещее:

```
CASE
WHEN Институт=3THEN'ИнстЭнерг'
ELSE'ИнстАвтом'
ENDIst
---
IIF(Институт=3,'Инст Энерг','ИнстАвтом')
```

Конструкции CASE, IIF могут быть вложенными друг в друга:

```
CASE
WHEN Table1ID IN(1,2)THEN'A'
WHEN Table1ID =3THEN
CASETable2ID  --вложенный CASE
WHEN3THEN'B-1'
```

```
WHEN4THEN'B-2'
END
ELSE'C'
ENDCaseEND
---
IIF(Table1ID IN(1,2),'A',
    IIFTable1ID=3,CASE Table2ID WHEN3THEN'B-1'WHEN4THEN'B-2'END,'C'))
```

Так как конструкция CASE и IIF представляют собой выражения, которые возвращают результат, то мы можем использовать их не только в блоке SELECT, но и в остальных блоках, допускающих использование выражений, например, в блоках WHERE или ORDER BY.

```
ORDERBY
CASEWHENSl>=1500THEN1ELSE0END, -- выдать стипендию сначала тем у кого
она ниже 1500
Name-- дальше упорядочить список в порядке ФИО
--
WHERECASEWHENSl>=1500THEN1ELSE0END=1-- все записи у которых выражение
равно 1
-- Пример с NULL
Case
WHEN Институт=2THEN'ИнстКибер'
WHENИнститутis null THEN'безинст'
ELSE'Др. инст'
endendcase
```

И еще, CASEможно работать внутри агрегатный функции:

```
Select
SUM(CASEWHENColumnID=1THENColumn1END) Columns
SUM(CASEWHENColumnID=2THENColumn1END) Columns
FromTabl
```

1.7. Группировка данных GROUPBY

Группирует выбранный набор строк для получения набора сводных строк по значениям одного или нескольких столбцов или выражений. Возвращается одна строка для каждой группы. Агрегатные функции в списке <select> предложения SELECT предоставляют информацию о каждой группе, а не об отдельных строках.

В предложении GROUP BY можно указывать несколько полей «GROUP BY поле1, поле2, …, полеN», в этом случае группировка произойдет по группам, которые образовывают значения данных полей «поле1, поле2, …, полеN».

Для примера сгруппируем студентов в институтах:

```
SELECT Институт, COUNT(ФИО)as Кол_Студ
FROM Студенты
GROUPBY Институт
```

Из основного, стоит отметить, что в случае группировки (GROUP BY), в перечне колонок в блоке SELECT:

- Мы можем использовать только колонки, перечисленные в блоке GROUP BY.
- Можно использовать выражения с полями из блока GROUP BY.
- Можно использовать константы, т.к. они не влияют на результат группировки.
- Все остальные поля (не перечисленные в блоке GROUP BY) можно использовать только с агрегатными функциями (COUNT, SUM, MIN, MAX, …).
- Не обязательно перечислять все колонки из блока GROUP BY в списке колонок SELECT.

Примечание. Так же стоит отметить, что группировку можно делать не только по полям, но также и по выражениям. Для примера сгруппируем данные по студентам, по годам рождения:

```
SELECT
  CONCAT('Год рождения - ',YEAR(ДР))Год_рождения,
COUNT(*)Коунт
FROMСтуденты
GROUPBYYEAR(ДР)
-- болеесложныйпример
SELECT
CASE
WHENYEAR(ДР)>=1997THEN'от 1997'
WHENYEAR(ДР)>=1995THEN'1997-1995'
       ELSE'1995-1990'
ENDEndY,
COUNT(*)coun
FROMСтуденты
GROUPBY
CASE
WHENYEAR(ДР)>=1997THEN'от 1997'
WHENYEAR(ДР)>=1995THEN'1997-1995'
       ELSE'1995-1990'
END
```

Результат для обеих (рис. 7):

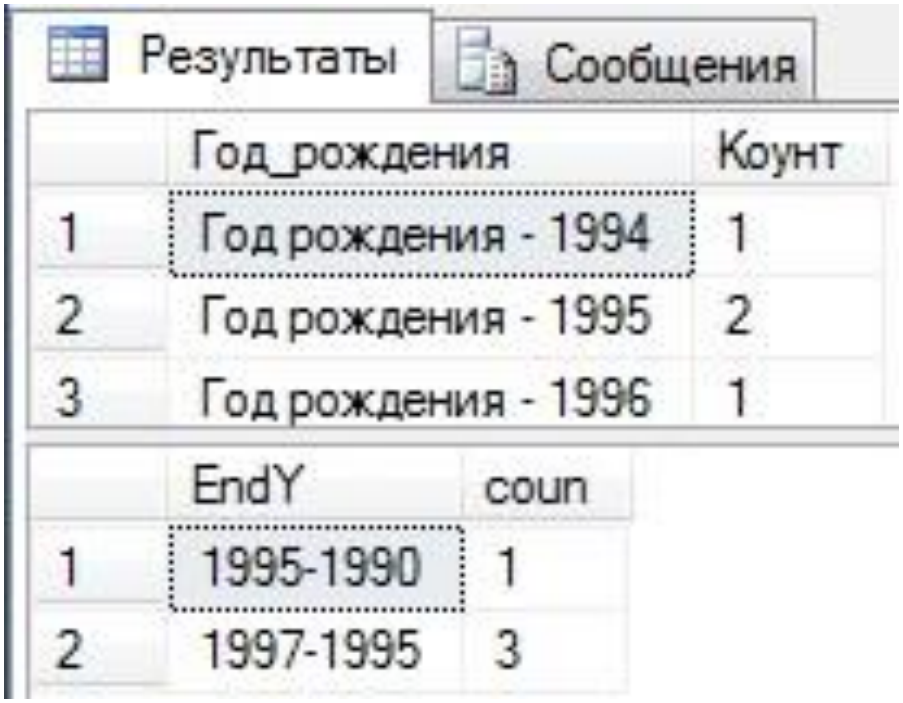


Рис. 7. Использование CASE

Т.е. в данном случае группировка делается по предварительно вычисленному для каждого CASE-выражению. Если убрать группировку и агрегатную функцию (COUNT), то в место 2 выражений получим 4.

Так же можем выполнить сортировку, она идет после группировки.

GROUP BY в купе с агрегатными функциями, одно из основных средств, служащих для получения сводных данных из БД, ведь обычно данные в таком виде и используются.

1.8. Условие HAVING

HAVING – чем-то подобен WHERE, только если WHERE-условие применяется к детальным данным, то HAVING-условие применяется к уже сгруппированным данным. По этой причине в условиях блока HAVING мы можем использовать либо выражения с полями, входящими в группировку, либо выражения, заключенные в агрегатные функции.

Для примера возьмем временную таблицу, где будем возвращать сгруппированные данные только по тем группам, у которых сумма стипендии всех студентов превышает 1000:

```
SELECT
GroupSt,
SUM(Sl)Степендия
FROM #Test1
GROUPBY GroupSt--группастудентов
HAVING SUM(Sl)>1000--больше 1000
```

Проведем сравнение WHEREиHAVING на примере:

```
SELECT
GroupSt,
SUM(Sl)Степендия
```

```
FROM #Test1
WHERE GroupSt='АИБ'--выборка
GROUP BY GroupSt--сделать группировку только по отобранным записям

SELECT
GroupSt,
SUM(Sl)Степендия
FROM #Test1
GROUP BY GroupSt--сделать группировку
HAVING GroupSt='АИБ'--наложить фильтр на результат группировки
```

На самом деле, несмотря на то, что эти два запроса выглядят по-разному, оптимизатор СУБД может выполнить их одинаково.

1.9. Многотабличные запросы и подзапросы

Для примеров изменим наши таблицы. Удалим через команду DROP таблицу Студент, потом Институт. И создадим 3 таблицы с ключами.

```
CREATETABLEИнституты
(
 ID int IDENTITY(1,1)NOTNULLCONSTRAINT PK_Институты PRIMARYKEY,
Наименование nvarchar(30)NOTNULL
)

CREATETABLEГруппы
(
 ID int IDENTITY(1,1)NOTNULLCONSTRAINT PK_ГруппыPRIMARYKEY,
Наименование nvarchar(30)NOTNULL
)

CREATETABLE[Студенты]
(
 ID int IDENTITY(1,1)NOTNULLCONSTRAINT PK_СтудентыPRIMARYKEY,
ФИО nvarchar(30)NOTNULL,
Паспорт int,
ГР date,
Стипендия int,
Группа int CONSTRAINT FK_Студенты_Группы
FOREIGNKEYREFERENCESГруппы(ID),
Институт int CONSTRAINT FK_Студенты_Институты FOREIGN KEY REFERENCES
Институты(ID)
)
```

Заполним созданные таблицы:

```
INSERT Институты(Наименование)VALUES
(N'Энергетика'), (N'Кибернетика'), (N'Автоматизация')
INSERT Группы(Наименование)VALUES
(N'ЭНБ'), (N'АИБ'), (N'ААБ'), (N'ИБ')

INSERT[Студенты](ФИО,Группа,Институт)VALUES
(N'Иванов И.И', 1, 1),
(N'Абрамов А.А', 2,3),
(N'Петров П.П', 3, 3),
```

```
(N'Андреев А.А',2, 3),
(N'Бодреев А.А', 2, 3),
(N'Фадеев Ф.А', 4, 2)

INSERT[Студенты](ФИО)VALUES(N'Михеев М.М')
INSERT Институты(Наименование)VALUES(N'Миханика')
```

1.10. Соединения JOIN

JOIN-соединения – операции горизонтального соединения данных. Здесь нам очень пригодится знание структуры БД, т.е. какие в ней есть таблицы, какие данные хранятся в этих таблицах и по каким полям таблицы связаны между собой. У нас структура состоит из 3-х таблиц (рис. 8):

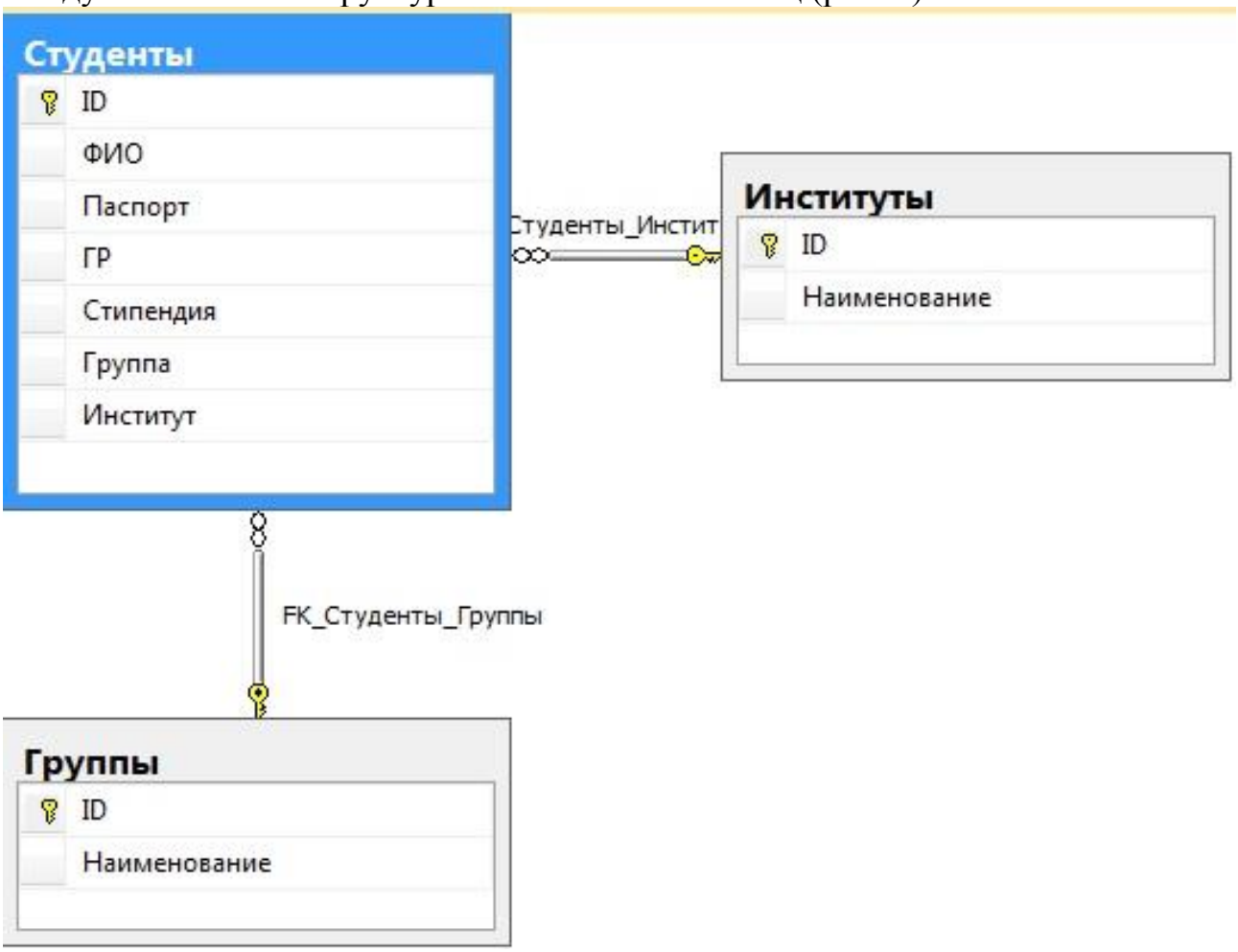


Рис. 8 - Диаграмма таблиц БД

Если говорить просто, то операции горизонтального соединения таблицы с другими таблицами используются для того, чтобы получить из них недостающие данные.

Существует пять типов соединения:

1. JOIN – левая таблица JOIN правая таблица ON условия соединения.

2. LEFT JOIN – левая таблица LEFT JOIN правая таблица ON условия соединения.
3. RIGHT JOIN – левая таблица RIGHT JOIN правая таблица ON условия соединения.
4. FULL JOIN – левая таблица FULL JOIN правая таблица ON условия соединения.
5. CROSS JOIN – левая таблица CROSS JOIN правая таблица.

Описание соединений указано в таблице 2.

Таблица 2

Описание соединений

Краткий синтаксис соединения	Полный синтаксис соединения	Описание соединения
JOIN	INNER JOIN	Из строк левой таблицы и правой таблицы объединяются и возвращаются только те строки, по которым выполняются условия соединения.
LEFT JOIN	LEFT OUTER JOIN	Возвращаются все строки левой таблицы (ключевое слово LEFT). Данными правой таблицы дополняются только те строки левой таблицы, для которых выполняются условия соединения. Для недостающих данных вместо строк правой таблицы вставляются NULL-значения.
RIGHT JOIN	RIGHT OUTER JOIN	Возвращаются все строки правой таблицы (ключевое слово RIGHT). Данными левой таблицы дополняются только те строки правой таблицы, для которых выполняются условия соединения. Для недостающих данных вместо строк левой таблицы вставляются NULL-значения.
FULL JOIN	FULL OUTER JOIN	Возвращаются все строки левой таблицы и правой таблицы. Если для строк левой таблицы и правой таблицы выполняются условия соединения, то они объединяются в одну строку. Для строк, для которых не выполняются условия соединения, NULL-значения вставляются на место левой таблицы, либо на место правой таблицы, в зависимости от того данных какой таблицы в строке не имеется.
CROSS JOIN	-	Объединение каждой строки левой таблицы со всеми строками правой таблицы. Этот вид соединения иногда называют декартовым произведением.

Полный синтаксис от краткого отличается только наличием слов INNER или OUTER. Обычно используют краткую версию.

Примеры:

```
SELECT Студенты.ID, Студенты.ФИО, Студенты.Институт, Институты.ID,
Институты.Наименование
FROM Студенты
JOIN Институты ON Студенты.Институт=Институты.ID

SELECT Студенты.ID, Студенты.ФИО, Студенты.Институт, Институты.ID,
Институты.Наименование
FROM Студенты
LEFTJOIN Институты ON Студенты.Институт=Институты.ID
```

Результат приведен на рис. 9:

JOIN

	ID	ФИО	Институт	ID	Наименование
1	1	Иванов И.И	1	1	Энергетика
2	2	Абрамов А.А	3	3	Автоматизация
3	3	Петров П.П	3	3	Автоматизация
4	4	Андреев А.А	3	3	Автоматизация
5	5	Бодреев А.А	3	3	Автоматизация
6	6	Фадеев Ф.А	2	2	Кибернетика

LEFT JOIN

	ID	ФИО	Институт	ID	Наименование
1	1	Иванов И.И	1	1	Энергетика
2	2	Абрамов А.А	3	3	Автоматизация
3	3	Петров П.П	3	3	Автоматизация
4	4	Андреев А.А	3	3	Автоматизация
5	5	Бодреев А.А	3	3	Автоматизация
6	6	Фадеев Ф.А	2	2	Кибернетика
7	7	Михеев М.М	NULL	N..	NULL

Рис. 9. Результат выполнения запроса

```
SELECT Студенты.ID, Студенты.ФИО, Студенты.Институт, Институты.ID,
Институты.Наименование
FROM Студенты
RIGHTJOIN Институты ON Студенты.Институт=Институты.ID

SELECT Студенты.ID, Студенты.ФИО, Студенты.Институт, Институты.ID,
Институты.Наименование
FROM Студенты
FULLJOIN Институты ON Студенты.Институт=Институты.ID
```

Результат приведен на рис. 10:

RIGHT JOIN

	ID	ФИО	Институт	ID	Наименование
1	1	Иванов И.И	1	1	Энергетика
2	6	Фадеев Ф.А	2	2	Кибернетика
3	2	Абрамов А.А	3	3	Автоматизац...
4	3	Петров П.П	3	3	Автоматизац...
5	4	Андреев А.А	3	3	Автоматизац...
6	5	Бодреев А.А	3	3	Автоматизац...
7	N..	NULL	NULL	4	Миханика

FULL JOIN

	ID	ФИО	Институт	ID	Наименование
1	1	Иванов И.И	1	1	Энергетика
2	2	Абрамов А.А	3	3	Автоматизация
3	3	Петров П.П	3	3	Автоматизация
4	4	Андреев А.А	3	3	Автоматизация
5	5	Бодреев А.А	3	3	Автоматизация
6	6	Фадеев Ф.А	2	2	Кибернетика
7	7	Михеев М.М	NULL	N..	NULL
8	N..	NULL	NULL	4	Миханика

Рис. 10. Результат выполнения запроса

```
SELECT Студенты.ID, Студенты.ФИО, Студенты.Институт, Институты.ID,
Институты.Наименование
FROM Студенты
CROSSJOIN Институты
```

Результат приведен на рис. 11:

	ID	ФИО	Институт	ID	Наименование
1	1	Иванов И.И	1	1	Энергетика
2	2	Абрамов А.А	3	1	Энергетика
3	3	Петров П.П	3	1	Энергетика
4	4	Андреев А.А	3	1	Энергетика
5	5	Бодреев А.А	3	1	Энергетика
6	6	Фадеев Ф.А	2	1	Энергетика
7	7	Михеев М.М	NULL	1	Энергетика
8	1	Иванов И.И	1	2	Кибернетика
9	2	Абрамов А.А	3	2	Кибернетика
10	3	Петров П.П	3	2	Кибернетика
11	4	Андреев А.А	3	2	Кибернетика
12	5	Бодреев А.А	3	2	Кибернетика
13	6	Фадеев Ф.А	2	2	Кибернетика
14	7	Михеев М.М	NULL	2	Кибернетика
15	1	Иванов И.И	1	3	Автоматизац...
16	2	Абрамов А.А	3	3	Автоматизац...
17	3	Петров П.П	3	3	Автоматизац...

CROSS JOIN

Результат выдаст 28 полей

Рис. 11. Результат выполнения запроса

Примечание. В многотабличных запросах, хоть и можно указать имя без псевдонима, в случае если имя не дублируется во второй таблице, но я бы рекомендовал всегда использовать псевдонимы в случае соединения, т.к. никто не гарантирует, что поле с таким же именем со временем не добавят во вторую таблицу, а тогда ваш запрос просто перестанет работать, выдавая сообщение, что не может понять к какой таблице относится данное поле.

А теперь разберем по порядку:

JOIN- по сути, данный запрос вернет только тех студентов, у которых указано значение «Институт», т.е. студенты числящих в институте.

LEFT JOIN - вернет всех студентов. Для тех студентов у которых не указан «Институт», поля «Институт.ID» и «Институты.Наименование» будут содержать NULL (NULL значения в случае необходимости можно обработать, например, при помощи ISNULL), т.е. когда нам важно получить данные по всем студентам, даже если они не числятся в институте.

RIGHTJOIN – аналогичнаLEFTJOIN, но наоборот. В этом случае отобразятся все институты, даже если в них нет студентов.

FULL JOIN - этот запрос важен, когда необходимо получить все данные по студентам и все данные по институтам. Соответственно получаем NULL-значения либо по студентам, либо по институтам.

CROSS JOIN –по сути, это объединение **LEFT JOIN и RIGHTJOIN.**

Рассмотрим еще один пример с использованием нескольких последовательных операций соединении. Если используется несколько операций соединения, то в таком случае они применяются последовательно сверху-вниз. Грубо говоря, после каждого соединения создается новый набор, и следующее соединение уже происходит с этим расширенным набором. Рассмотрим простой пример:

```
SELECT Студенты.ID, Студенты.ФИО, Институты.Наименование,
Группы.Наименование
FROM Студенты
JOIN Институты ON Студенты.Институт=Институты.ID
JOIN Группы ON Студенты.Институт=Группы.ID
```

Так же после этого можем использовать операторы WHERE, ORDER BY и т.д., они следует после JOIN

```
SELECTсписок_столбцов или *
FROM источник
WHERE фильтр
ORDERBYвыражение_сортировки
--роли источника выступает
TABLE_1
JOIN ... ON TABLE_1.Column_1=...Column_1
```

Так же на месте любой таблицы может стоять подзапрос. В свою очередь подзапросы могут быть вложены в подзапросы. И все эти подзапросы тоже представляют собой базовые конструкции. То есть базовая конструкция, это кирпичики, из которых строится любой запрос.

1.11. Связь при помощи WHERE-условия

Вернемся к предыдущему примеру:

```
SELECT Студенты.ID, Студенты.ФИО, Студенты.Институт, Институты.ID,
Институты.Наименование
FROM Студенты
RIGHTJOIN Институты ON Студенты.Институт=Институты.ID
```

Выполним его через WHERE:

```
SELECT Студенты.ID, Студенты.ФИО, Студенты.Институт, Институты.ID,
Институты.Наименование
FROM Студенты, Институты
Where Студенты.Институт=Институты.ID
```

Для оптимизатора запроса может быть и без разницы, как вы реализуете соединение (при помощи WHERE или JOIN), он их может выполнить абсолютно одинаково. Но из соображения понимаемости кода, рекомендуют стараться никогда не делать соединение таблиц при помощи WHERE-условия. WHERE-условия служат для фильтрации набора, и не нужно перемешивать условия служащие для соединения, с условиями, отвечающими за фильтрацию.

Обычно, в простых запросах, это не бросается в глаза, но если вы используете подзапросы и большое количество условий, то можно запутаться и не правильно реализовать запрос. Пример такого запроса:

```
Select Блюда.[ID Блюды], (cena.Стоимость+ Блюда.[Цена на
приготовления])as[Цена за блюдо]
From Блюда, (SelectSum(Продукты.Стоимость)as Стоимость, Состав.[ID
Блюда]
From Продукты innerjoin Состав on Продукты.[ID Продукта]= Состав.[ID
Продукта]
Where Продукты.[ID Продукта]in(Select Продукты.[ID Продукта]
From Продукты, (SelectDATEDIFF(day,GETDATE(),Поставка.Дата)as Д, [ID
Продукта], Количество From Поставка)AS t2
Where(t2.[ID Продукта]= Продукты.[ID Продукта])and((Продукты.[Срок
хранения]+ t2.Д)>0)and(t2.Количество >0))
Groupby Состав.[ID Блюда]
HavingCOUNT(Состав.[ID Продукта])in(SelectCOUNT(Состав.[ID
Продукта])as t from Состав Groupby[№ Приготавление]))ascena
Where Блюда.[ID Блюды]=cena.[ID Блюда];
```

1.12. Объединение UNION

UNION-объединения – операции вертикального объединения результатов запросов.Для примера выведем количество групп и студентов в каждом институте:

```
SELECTst.ИнститутInst, COUNT(st.Группа)asGruppa, COUNT(*) Student
FROMСтудентыst
WHEREst.Институт=1
GROUPBYst.Институт

SELECTst.ИнститутInst, COUNT(st.Группа)asGruppa, COUNT(*) Student
FROMСтудентыst
WHEREst.Институт=2
GROUPBYst.Институт

SELECTst.ИнститутInst, COUNT(st.Группа)asGruppa, COUNT(*) Student
FROMСтудентыst
WHEREst.Институт=3
GROUPBYst.Институт

SELECTst.ИнститутInst, COUNT(st.Группа)asGruppa, COUNT(*) Student
FROMСтудентыst
WHEREst.Институт=4
GROUPBYst.Институт
```

В итоге получим 4 таблицы с результатами. Так вот, если бы мы не знали, что существует операция группировки, но знали бы, что существует операция объединения результатов запроса при помощи UNION ALL, то мы могли бы склеить все эти запросы следующим образом:

```
SELECTst.ИнститутInst, COUNT(st.Группа)asGruppa, COUNT(*) Student
FROMСтудентыst
WHEREst.Институт=1
```

```
GROUPBYst.Институт
UNIONALL
SELECTst.ИнститутInst, COUNT(st.Группа)asGruppa, COUNT(*) Student
FROMСтуденты st
WHEREst.Институт=2
GROUPBYst.Институт
UNIONALL
SELECTst.ИнститутInst, COUNT(st.Группа)asGruppa, COUNT(*) Student
FROMСтуденты st
WHEREst.Институт=3
GROUPBYst.Институт
UNIONALL
SELECTst.ИнститутInst, COUNT(st.Группа)asGruppa, COUNT(*) Student
FROMСтуденты st
WHEREst.Институт=4
GROUPBYst.Институт
```

В место 4 таблиц получим одну (рис. 12):

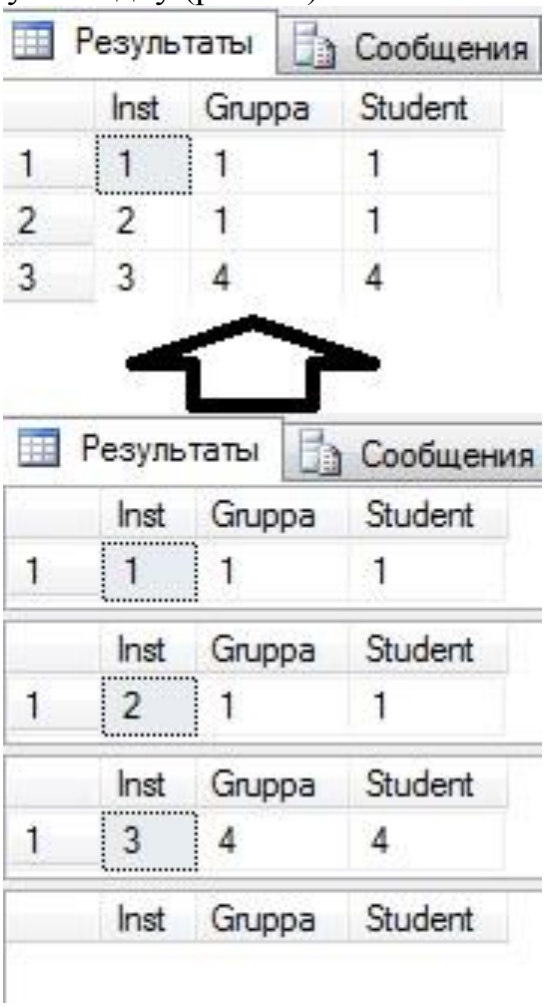


Рис. 12. Выполнение запроса с использованием UNIONALL

Т.е. UNIONALL позволяет склеить результаты, полученные разными запросами в один общий результат. Соответственно количество колонок в каждом запросе должно быть одинаковым, а также должны быть

совместимыми и типы этих колонок, т.е. строка под строкой, число под числом, дата под датой и т.п.

- UNION ALL - В результат включаются все строки из обоих наборов (A+B).
- UNION - В результат включаются только уникальные строки двух наборов DISTINCT(A+B).
- EXCEPT - В результат попадают уникальные строки верхнего набора, которые отсутствуют в нижнем наборе. Разница 2-х множеств DISTINCT(A-B).
- INTERSECT - В результат включаются только уникальные строки, присутствующие в обоих наборах. Пересечение 2-х множеств DISTINCT(A&B).

Примеры работы EXCEPT (рис. 13) и INTERSECT (рис. 14):

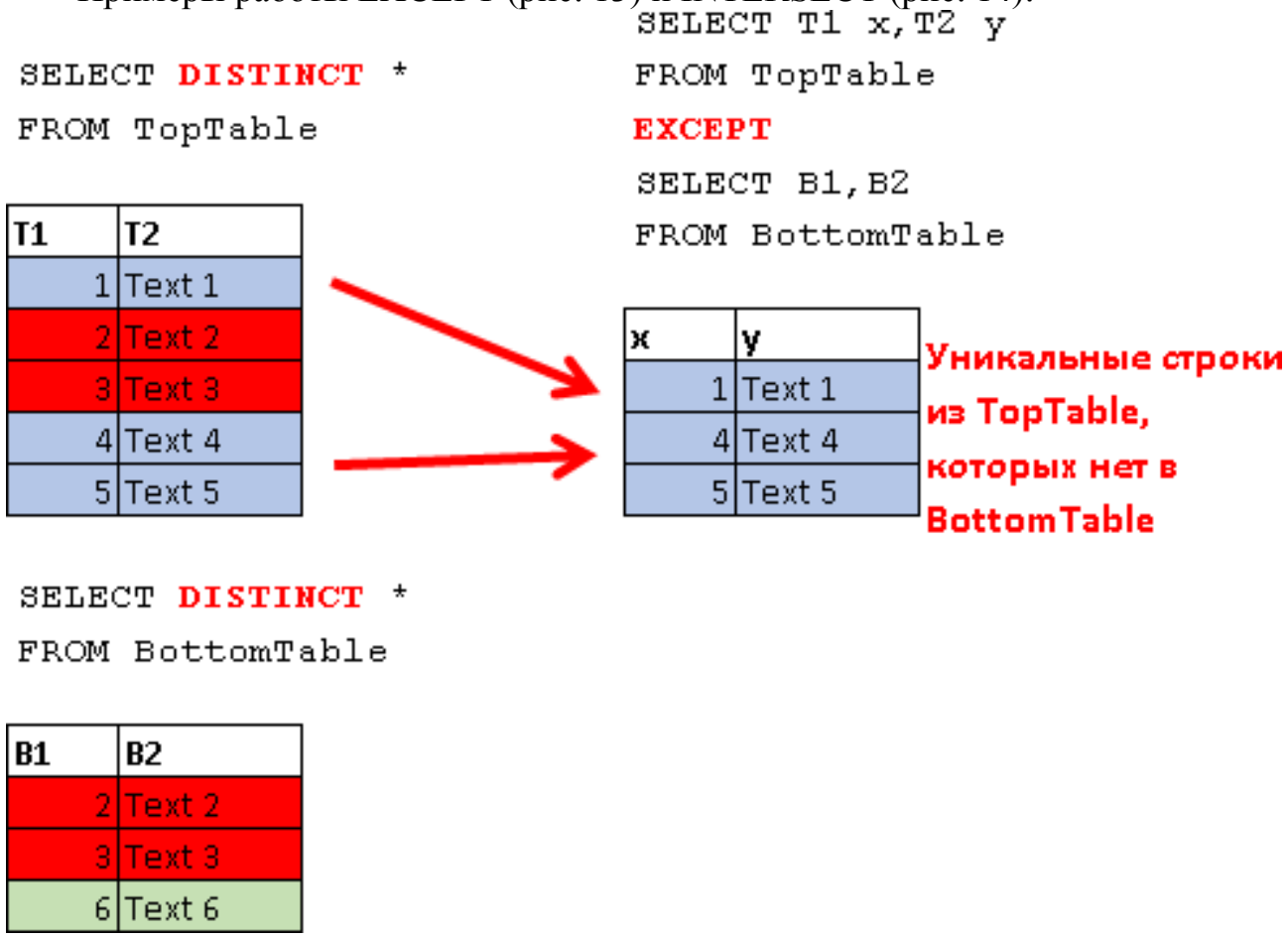


Рис. 1. Пример с EXCEPT

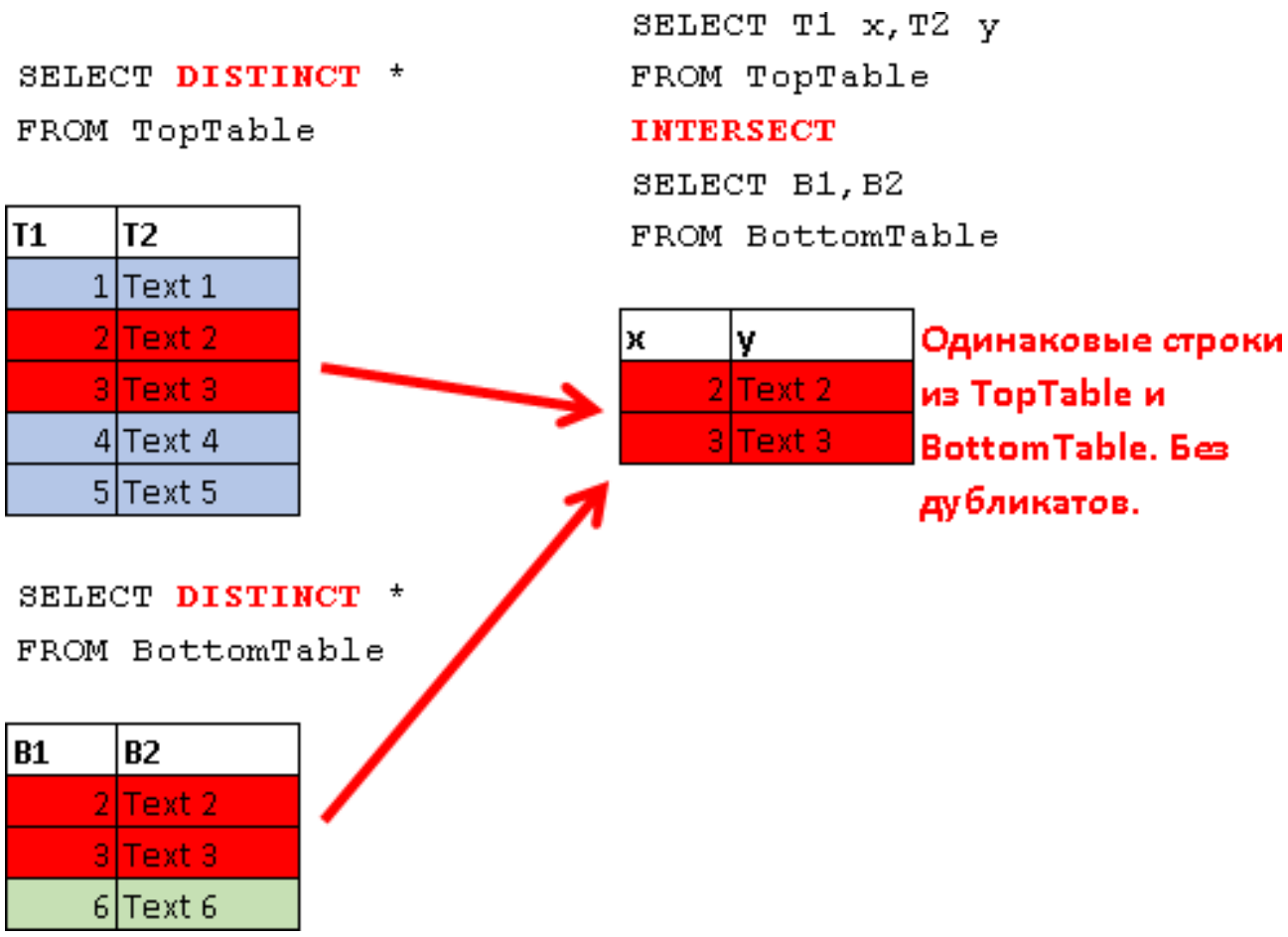


Рис. 2. Пример с INTERSECT

Вот в принципе и все, что касается вертикальных объединений, они намного проще, чем JOIN-соединения.

1.13. Создание подзапросов

Вложенным запросом называется запрос, помещаемый в инструкцию SELECT, INSERT, UPDATE или DELETE или в другой вложенный запрос. Подзапрос может быть использован везде, где разрешены выражения. Вложенный запрос по-другому называют внутренним запросом или внутренней операцией выбора, в то время как инструкцию, содержащую вложенный запрос, называют внешним запросом или внешней операцией выбора.

Многие инструкции языка Transact-SQL, включающие подзапросы, можно записать в виде соединений. Другие запросы могут быть осуществлены только с помощью подзапросов. В языке Transact-SQL обычно не бывает разницы в производительности между инструкцией, включающей вложенный запрос, и семантически эквивалентной версией без вложенного запроса. Однако в некоторых случаях, когда проверяется существование соединения, показывают лучшую производительность. В противном случае для устранения дубликатов вложенный запрос должен обрабатываться для получения каждого результата внешнего запроса. В таких случаях метод работы соединений дает лучшие результаты.

Пример, запроса для вывода студентов и в каком институте он состоит:

```
Selectst.ФИО, (Selectit.НаименованиеFrom Институты itWhere
it.ID=st.Институт)
From Студенты st
```

С помощью подзапроса мы вывели наименование института, вместо цифр. В данном случае подзапрос выполнится 7 раза, т.е. для каждого значения st.Институт.

Подзапрос в данном случае должен возвращать только одну строку и одну колонку. Если в подзапросе получается много строк, то используйте в нем либо TOP, либо какую-нибудь агрегатную функцию, чтобы в итоге получилась одна строка. Поэтому, я бы рекомендовал прибегать к использованию подзапросов с параметрами в самую последнюю очередь, т.к. когда вы не можете выразить запрос при помощи простых операций соединений, т.к. использование подзапросов в таких случаях может сильно понизить скорость выполнения запроса. Потому что подзапрос с параметром будет выполняться для каждого переданного ему параметра.

Пример:

```
Select it.ID, it.Наименование,
(SelectTOP1st.ФИОFromСтудентыstWHERE it.ID=st.Институт) Student,
(SelectTOP1st.ГруппаFromСтудентыstWHERE it.ID=st.Институт)grup
From Институты it
```

Можно применить конструкцию APPLY, которая имеет 2 формы – CROSS APPLY и OUTER APPLY. Конструкция APPLY позволяет избавиться от множества подзапросов, как в данном примере:

```
Select it.ID, it.Наименование, st1.fio, st1.grup
FromИнституты it
CROSS APPLY
(
SelectTOP1st.ФИОfio, st.Группagrup
FromСтудентыstWHERE it.ID=st.Институт
) st1
```

Здесь подзапрос блока CROSS APPLY выполнится для каждого значения строки из таблицы «Институты». Если подзапрос строки не вернет, то данный институт исключается из результирующего списка.

Если требуется, чтобы были возвращены все строки таблицы «Институты», то используйте следующую форму этого оператора OUTER APPLY:

```
Select it.ID, it.Наименование, st1.fio, st1.grup
FromИнституты it
OUTER APPLY
(
SelectTOP1st.ФИОfio, st.Группagrup
FromСтудентыstWHERE it.ID=st.Институт
) st1
```

В общем, достаточно полезный оператор, который в некоторых ситуациях сильно упрощающий запрос. Данный подзапрос так же сработает для каждой строки результирующего набора, т.е. для каждого переданного параметра, но сработает он намного эффективнее, чем в случае использования множества подзапросов. С прочими деталями в случае применения APPLY, например, для случая, когда подзапрос возвращает несколько строк.

Использование подзапросов в блоке WHERE. Выведем те институты, в которых числиться 1 студент:

```
Select it.ID, it.Наименование
From Институты it
Where(SelectCOUNT(*)FromСтудентыstWHERE it.ID=st.Институт)=1
```

Так как здесь мы используем оператор сравнения, то соответственно подзапрос должен возвращать максимум одну строку и одно значение, т.е. так же когда подзапрос используется и в блоке SELECT.

Конструкции EXISTS и NOT EXISTS. Позволяют проверить есть ли соответствующие условию записи в подзапросе:

```
Select it.ID, it.Наименование
From Институты it
WhereEXISTS(Select*FromСтудентыstWHERE it.ID=st.Институт)
--есть нет ни одного  Студента
Select it.ID, it.Наименование
FromИнституты it
WhereNOTEXISTS(Select*FromСтудентыstWHERE it.ID=st.Институт)
```

Здесь все просто – EXISTS возвращает True, если подзапрос возвращает хотя бы одну строку, и False, если подзапрос не возвращает строк. NOTEXISTS – инверсия результата.

Конструкция IN и NOT IN с подзапросом. Так же можно использовать его с подзапросом, который возвращает перечень этих значений:

```
Select*--институт где есть студенты
From Институты it
Where ID IN(Selectst.ИнститутFromСтудентыstWHEREst.Институтis not
null)

Select*--институт где нет студентов
From Институты it
Where ID not IN(Selectst.ИнститутFromСтудентыstWHEREst.Институтis not
null)
```

Примечание. Обратите внимание, что для исключения NULL значения используется условие (Department ID IS NOT NULL) в подзапросе. NULL значения в данном случае так же опасны (смотрите об этом в описании конструкции IN).

Подзапросы можно так же использовать во многих других блоках, например, в блоке HAVING, осуществлять проверку в конструкциях CASE, FROM чтобы использовать несколько выражений.

Конструкция WITH - это достаточно полезная конструкция особенно в случае работы с большими подзапросами. К примеру:

```
SELECT q1.x1,q1.y1,q2.x2,q2.y2
FROM
(
SELECT T1 x1,T2 y1
FROM TopTable
EXCEPT
SELECT B1,B2
FROM BottomTable
) q1
JOIN
(
SELECT T1 x2,T2 y2
FROM TopTable
EXCEPT
SELECT N1,N2
FROM NextTable
)q2
ONq1.x1=q2.x2
```

То же самое написанное при помощи WITH:

```
WITH q1 AS(
SELECT T1 x1,T2 y1
FROM TopTable
EXCEPT
SELECT B1,B2
FROM BottomTable
),
q2AS(
SELECT T1 x2,T2 y2
FROM TopTable
EXCEPT
SELECT N1,N2
FROM NextTable
)
-- основной запрос становится более прозрачным
SELECT q1.x1,q1.y1,q2.x2,q2.y2
FROM q1
JOIN q2 ON q1.x1=q2.x2
```

Как видим, большие подзапросы вынесены и поименованы в блоке WITH, что дало возможность разгрузить текст основного запроса и сделать его понятным.

Использование WITH по-другому называет CTE-выражениями: Общие табличные выражения (CTE — Common Table Expressions) позволяют существенно уменьшить объем кода, если многократно приходится обращаться к одним и тем же запросам. CTE играет роль представления, которое создается в рамках одного запроса и, не сохраняется как объект схемы.

1.14. Операторы модификации данных

Рассмотрим работу с операторами модификации данных:

- INSERT – вставка новых данных.
- UPDATE – обновление данных.
- DELETE – удаление данных.
- SELECT … INTO … – сохранить результат запроса в новой таблице.
- MERGE – слияние данных.
- TRUNCATE TABLE – DDL-операция для быстрой очистки таблицы.

Операции модификации данных очень сильно связаны с конструкциями оператора SELECT, т.к. по сути, выборка модифицируемых данных идет при помощи них. Рассмотрим только основные формы операторов модификации данных. За более подробной информацией по каждому оператору обращайтесь в MSDN.

1.14.1. Оператор INSERT

INSERT – вставка новых данных. В предыдущих темах мы его использовали, теперь рассмотрим подробней.

Данный оператор имеет 2 основные формы:

```
-- вставка в таблицу новой строки значения полей которой формируются
из перечисленных значений
INSERTINTO таблица(перечень_полей)VALUES(перечень_значений)
--

-- вставка в таблицу новых строк, значения которых формируются из
значений строк возвращенных запросом.
INSERTINTO таблица(перечень_полей)SELECTперечень_значенийFROM …
```

Рассмотрим пример:

```
INSERT[Студенты](ФИО, Паспорт, ГР, Стипендия, Группа, Институт)VALUES
(N'хавров И.И', 784568, null, 1300, 1, 1),
(N'Маликов А.А', 984565, null, 1500, 4, 1)
```

Хоть в этом случае можно и не указывать перечень полей, т.к. мы вставляем данные всех полей и в таком же виде, как они перечислены в таблице, т.е. не перечислять какие поля заполнять:

```
INSERT[Студенты]VALUES
(N'хавровИ.И', 784568, 02151995, NULL, 1, 1),
(N'Маликов А.А', 984565, null, 1500, 4, 1)
```

Примечание. Не рекомендуется использовать такой подход, особенно если данный запрос будет использоваться регулярно. Опять же это чревато тем, что структура таблицы может изменяться, в нее могут быть добавлены новые поля, или же последовательность полей может быть изменена, что еще опасней, т.к. это может привести к появлению логических ошибок во вставленных

данных. Поэтому лучше лишний раз не полениться и перечислить явно все поля, в которые вы хотите вставить значение.

Так же важно, чтобы при вставке были заданы значения для всех обязательных полей, которые помечены в таблице как NOT NULL.

Можно не указывать поля, у которых была указана опция IDENTITY или же поля, у которых было задано значение по умолчанию при помощи DEFAULT, т.к. в качестве их значения подставится либо значение из счетчика, либо значение, указанное по умолчанию.В случаях, когда значение поля со счетчиком нужно задать явно используйте опцию IDENTITY_INSERT.

Пример использований IDENTITY_INSERT:

```
SETIDENTITY_INSERTСтудентыON

INSERT[Студенты](ID,ФИО)VALUES
(101,N'Name1'),
(201,N'Name2'),
(301,N'Name3')

-- запрещаем добавление/изменение IDENTITY значения
SETIDENTITY_INSERTСтудентыOFF
```

Теперь рассмотрим через простой пример с использованием SELECT:

```
INSERTMyTable(PriKey, Description)
SELECTForeignKey, Description
FROMSomeView;
```

В этом случае VALUESзаменяется SELECT.

1.14.2. Оператор UPDATE

UPDATE – обновление данных. Данный оператор имеет так же 2 формы:

```
UPDATE таблица SET … WHERE условие_выборки
```

Обновляет строки таблицы, для которых выполняется условие выборки. Если предложение WHERE не указано, то будут обновлены все строки. Это можно сказать классическая форма оператора UPDATE.

```
UPDATE псевдоним SET … FROM …
```

Обновляет данные таблицы участвующей в предложении FROM, которая задана указанным псевдонимом. Конечно, здесь можно и не использовать псевдонимов, используя вместо них имена таблиц, но с псевдонимом на наш взгляд удобнее.

В примере обновим студентам стипендию:

```
UPDATE Студенты SET Стипендия=1000WHERE ID=1
UPDATE Студенты SET  Стипендия=1500WHERE ID=2
```

Можно обновить несколько столбцов:

```
UPDATE Студенты SET ГР ='19950505', Стипендия=1500WHERE ID=3
```

Пример для второго случая, где применялся псевдоним:

```
UPDATE e
SET
Группа=(SELECT ID FROMГруппыWHERE ID=e.Группа),
Институт=(SELECT ID FROMИнститутыWHERE ID=e.Институт)
FROM Студенты e
-- Или так
UPDATE e
SET
  Группа= g.ID,
  Институт= it.ID
FROMСтуденты e
LeftJOINГруппы g ON g.ID=e.Группа
LeftJOINИнституты it ON it.ID=e.Институт
```

1.14.3. Оператор DELETE

DELETE – удаление данных. Принцип работы DELETE похож на принцип работы UPDATE, и так же в MS SQL можно использовать 2 формы:

```
DELETE таблица WHEREусловие_выборки
```

Удаление строк таблицы, для которых выполняется условие выборки. Если предложение WHERE не указано, то будут удалены все строки. Это можно сказать классическая форма оператора DELETE (только в некоторых СУБД нужно писать DELETE FROM таблица WHERE условие выборки).

```
DELETE псевдоним FROM …
```

Удаление данных таблицы участвующей в предложения FROM, которая задана указанным псевдонимом. Конечно, здесь можно и не использовать псевдонимов, используя вместо них имена таблиц, но с псевдонимом на наш взгляд удобнее.

Для примера при помощи первого варианта:

```
--Удалим 2-х студентов
DELETE Студенты WHERE ID IN(4,5)
```

При помощи второго варианта удалим студента, у которого нет института или группы:

```
DELETE e
FROMСтуденты e
WHERE(e.ГруппаISNULL) or (e.ИнститутISNULL)
```

В заключение, по данным операторы модификации данных – INSERT, UPDATE и DELETE, очень легко понять интуитивно, когда умеешь пользоваться конструкциями оператора SELECT. Но все же старайтесь писать, как можно проще и понятней, обязательно предварительно проверяя, какие записи будут обработаны при помощи SELECT.

1.14.4. Оператор SELECT … INTO …

Данная конструкция позволяет сохранить результат выборки в новой таблице. Она представляет собой что-то промежуточное между DDL и DML.

Типы колонок созданной таблицы будут определены на основании типов колонок набора, полученного запросом SELECT. Если в выборке присутствуют результаты выражений, то им должны быть заданы псевдонимы, которые будут служить в роли имен колонок.

Для примера возьмем 2 таблицы и в новый таблице сохраним ИД, ФИО студента и наименование института, в котором он состоит и посмотри что получилось (рис.15):

```
Select e.ID, e.ФИО, it.Наименование
INTOStudentInfo
FROMСтуденты e
JOINИнституты it ON it.ID= e.Институт

SELECT*
FROMStudentInfo
```

	ID	ФИО	Наименование
1	1	Иванов И.И	Энергетика
2	2	Абрамов А.А	Автоматизация
3	3	Петров П.П	Автоматизация
4	4	Андреев А.А	Автоматизация
5	5	Бодреев А.А	Автоматизация
6	6	Фадеев Ф.А	Кибернетика
7	8	хавров И.И	Энергетика
8	9	Маликов А.А	Энергетика

Рис. 15. Результат выполнения запроса

Можно обновить список таблиц в инспекторе объектов и увидеть новую таблицу. Данную конструкцию иногда удобно применять при формировании очень сложных отчетов, которые требуют выборки из множества таблиц. В этом случае данные обычно сохраняют во временных таблицах (#). Т.е. предварительно при помощи запросов, мы сбрасываем данные во временные таблицы, а затем используем эти временные таблицы в других запросах, которые формируют окончательный результат.

Иногда данную конструкцию удобно использовать, чтобы сделать полную копию всех данных текущей таблицы. Вы можете сохранить копию либо всех данных таблицы, либо только тех данных, которых коснется модификация. Т.е. если что-то пойдет не так, вы сможете восстановить данные.

К примеру, можно создать другую базу, а там сохранять backup:

```
CREATEDATABASETestBekap
GO

SELECT*
INTOTestBekap.dbo.StudentBackup-- используемпрефиксИмяБаза.Схема.
FROM Студенты
```

1.14.5. Оператор MERGE

MERGE – слияние данных. Данный оператор хорошо подходит для синхронизации данных 2-х таблиц. Такой запрос может понадобиться при интеграции разных систем, когда данные передаются порциями из одной системы в другую.

В нашем случае, допустим, что стоит задача синхронизации таблицы «Студенты» с таблицей «StudentBackup». Допустим у нас в таблице «Студенты» некоторых студентов удалили, где то не правильно обновили данные или удалили часть информации. Для этого необходимо:

- Если для строки таблицы «Студенты»соответствия по ключу не нашлось, то необходимо сделать удаление таких строк из «Студенты».
- Если соответствие нашлось, то необходимо обновить строки «Студенты»данными соответствующей строки из «StudentBackup».
- Если строка есть в «StudentBackup», но ее нет в «Студенты», то ее необходимо добавить в «Студенты».

Выполним реализацию всей этой логики при помощи инструкции MERGE:

```
SELECT*
INTOStudentBackup
FROMСтуденты
---
MERGE Студентыtrg-- таблицаприемник
USINGStudentBackupsrc-- таблицаисточник
ON trg.ID=src.ID-- условиеслияния
-- 1. Строка есть в trg но нет сопоставления со строкой из src
WHENNOT MATCHED BY SOURCE THEN
DELETE
-- 2. Есть сопоставление строки trg со строкой из источника src
WHEN MATCHED THEN
UPDATESET
trg.Стипендия=src.Стипендия,
-- 3. Строка не найдена в trg, но есть в src
WHENNOT MATCHED BY TARGET THEN-- предложение BY TARGET
можноотпускать, т.е. NOT MATCHED = NOT MATCHED BY TARGET
INSERT(ФИО, Паспорт, ГР, Стипендия, Группа, Институт)
```

```
VALUES(src.ФИО,src.Паспорт,src.ГР,src.Стипендия,src.Группа,src.Институт);
```

Примечание. Данная конструкция должна оканчиваться «;».

После выполнения запроса сравните 2 таблицы, их данные должны быть одинаковыми.

Конструкция MERGE чем-то напоминает условный оператор CASE, она так же содержит блоки WHEN, при выполнении условий которых происходит то или иное действие, в данном случае удаление (DELETE), обновление (UPDATE) или добавление (INSERT). Модификация данных производится в таблице приемнике. В качестве источника может выступать запрос.

1.14.6. TRUNCATE TABLE

Данный оператор является DDL-операцией и служит для быстрой очистки таблицы, т.е. удаляет все строки из нее.

RUNCATE TABLE – удаляет все строки в таблице, не записывая в журнал удаление отдельных строк. Инструкция TRUNCATE TABLE похожа на инструкцию DELETE без предложения WHERE, однако TRUNCATE TABLE выполняется быстрее и требует меньших ресурсов системы и журналов транзакций.

Если таблица содержит столбец идентификаторов (столбец с опцией IDENTITY), счетчик этого столбца сбрасывается до начального значения, определенного для этого столбца. Если начальное значение не задано, используется значение по умолчанию, равное 1. Чтобы сохранить столбец идентификаторов, используйте инструкцию DELETE.

Инструкцию TRUNCATE TABLE нельзя использовать, если на таблицу ссылается ограничение FOREIGN KEY. Таблицу, имеющую внешний ключ, ссылающийся сам на себя, можно усечь.

```
TRUNCATETABLE Студенты
```

2. РАЗРАБОТКА ПРИЛОЖЕНИЯ НА ЯЗЫКЕ C# ДЛЯ РАБОТЫ В MS SQL SERVER

2.1. Подключение к СУБД

Для примера так же возьмем БД из предыдущего пособия (Михайлова У.В., Баранкова И.И., Лукьянов Г.И. «Разработка БД В MS SQL Server с использованием SSMS»). С начала необходимо создать приложение WindowsForms с помощью Visual C#.

Для приложения требуется подключение к СУБД. С нашим приложением его связывает тонкая нить в виде строки подключения. И от этой строки зависит очень многое. Строки подключение бывают разными и завися от самой СУБД (к примеру, у MySQL строка подключения отличается от MSSQLServer, даже у версии 2012 года есть отличия от версии 2014 года) и их настроек, так же стоит учитывать, как будет осуществляться подключение к БД: локально или удаленно.

Разберем некоторые строки подключения:

Стандартная безопасность:

```
Server = myServerAddress; Database=myDataBase; User Id =myUsername; Password=myPassword;
```

Надежноеподключение:

```
Server = myServerAddress; Database = myDataBase; Trusted_Connection= True;
```

Подключение к экземпляру SQL сервера:

```
Server = myServerName\myInstanceName; Database = myDataBase; User Id = myUsername; Password = myPassword;
```

Подключениечерез IP-адрес:

```
Data Source=190.190.200.100,1433; Network Library=DBMSSOCN;
Initial Catalog=myDataBase; User ID=myUsername; Password=myPassword;
```

User ID используется для аутентификации средствами SQL.Если с именем пользователя связан пароль, используется Password или Pwd.

Password или Pwd - поле пароля используется с User ID. Не имеет смысла вход в систему без имени пользователя, а только с паролем. Password и Pwd полностью взаимозаменяемы.

DataSource - название экземпляра SQL Servera, с которым будет идти взаимодействие. Это может быть название локального сервера, например, "EUGENEPC/SQLEXPRESS", либо сетевой адрес.

В качестве альтернативного названия параметра можно использовать **Server, Address, Addr и NetworkAddress,**к примеру, на MSDNпишут, что разницы между ними нет.

Integrated SercurityилиTrusted_Connection задает режим аутентификации. Может принимать значения true, false, yes, no и sspi. По умолчанию значение false. При обычном соединении, мы указываем имя пользователя и пароль в строке подключения, если TrustedConnection имеет значение. Тут могут использоваться имена локальных Windows пользователей (локальные учетные записи), имена пользователей домена AD, или иена

пользователей SQL сервера, при условии когда на SQL сервере включены оба типа аутентификации: Windows и SQL. В случае Trusted Connection может использоваться только режим Windows аутентификации SQL сервера. При этом имя пользователя и пароль не указываются в строке подключения. Вместо этого данные о пользователе берутся из текущего контекста безопасности приложения.

InitialCatalog и Database это просто два способа выбора базы данных, связанной с соединением.

NetworkLibrary или Net эта опция Network Library важна, если вы связываетесь с сервером по протоколу, отличному от TCP/IP. Значение по умолчанию для Network Library - это dbmssocn или TCP/IP. Доступны следующие опции:

- dbnmpntw (NamedPipes, именованные каналы),
- dbmsrpcn (Multiprotocol),
- dbmsadsn (AppleTalk),
- dbmsgnet (VIA),
- dbmsipcn (Shared Memory, разделяемая память),
- dbmsspxn (IPX/SPX)
- dbmssocn (TCP/IP).

Перейдем к самому приложению.

Вначале расположим компоненты на форму, сделаем 2 типа подключение локально и удаленно (рис. 16):

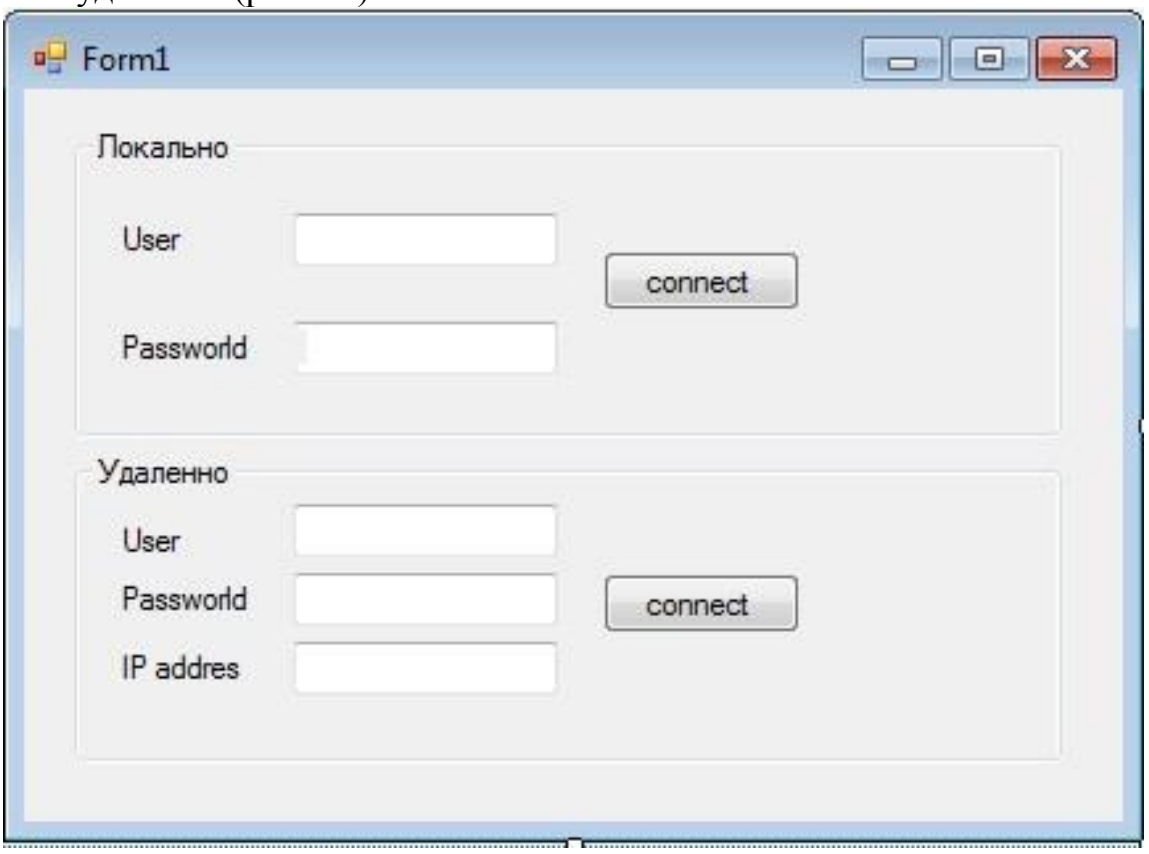


Рис. 36. Внешний вид приложения

Затем подключим класс:

```
Using System.Net;
using System.Data.SqlClient;
```

Создадим методы для локального и удаленного подключения. Для локального подключения:

```
Private void localconnect()
{
try
{
blockbutton(false);
string connectStr="Server=localhost; database=Test; User ID="+
textBox1.Text.ToString().Trim()+"; Password="+
textBox2.Text.ToString().Trim()+";";
SqlConnectiondbConnection=new SqlConnection(connectStr);
dbConnection.Open();
MessageBox.Show("Клиент подключился к БД");
dbConnection.Close();
blockbutton(true);
}
catch(SqlException e)
{
MessageBox.Show(e.Message);
blockbutton(true);
}
}
```

И для удаленного подключения:

```
Private void removeconnect()
{
IPAddressIpAd;// Для проверки правильного виденного IPa адреса
if(IPAddress.TryParse(textBox5.Text, out IpAd))
{
try
{
blockbutton(false);
string connectStr="Data
Source="+textBox5.Text.ToString().Trim()+",1433; Network
Library=DBMSSOCN;"+
"User ID="+textBox3.Text.ToString().Trim()+"; Password="+
textBox4.Text.ToString().Trim()+"; Initial Catalog=Test;";
SqlConnectiondbConnection=new SqlConnection(connectStr);
dbConnection.Open();
MessageBox.Show("Клиент подключился к БД");
dbConnection.Close();
blockbutton(true);
}
catch(SqlException e)
{
MessageBox.Show(e.Message);
blockbutton(true);
}
}
else
```

```
{
MessageBox.Show("Неверно виден IpAddres");
}
}
Private void removeconnect()
{
IPAddressIpAd;// Для проверки правильного виденного IPa адреса
if(IPAddress.TryParse(textBox5.Text, outIpAd))
{
try
{
blockbutton(false);
string connectStr="Data
Source="+textBox5.Text.ToString().Trim()+",1433; Network
Library=DBMSSOCN;"+
"User ID="+textBox3.Text.ToString().Trim()+"; Password="+
textBox4.Text.ToString().Trim()+"; Initial Catalog=Test;";
SqlConnectiondbConnection=new SqlConnection(connectStr);
dbConnection.Open();
MessageBox.Show("Клиент подключился к БД");
dbConnection.Close();
blockbutton(true);
}
catch(SqlException e)
{
MessageBox.Show(e.Message);
blockbutton(true);
}
}
else
{
MessageBox.Show("Неверно введен IpAddres");
}
}
```

Через обработчик кнопки запустим метод, и при удачном подключении получим сообщение о подключении (рис.17), иначе получим сообщение об ошибке от обработчика (рис. 18).

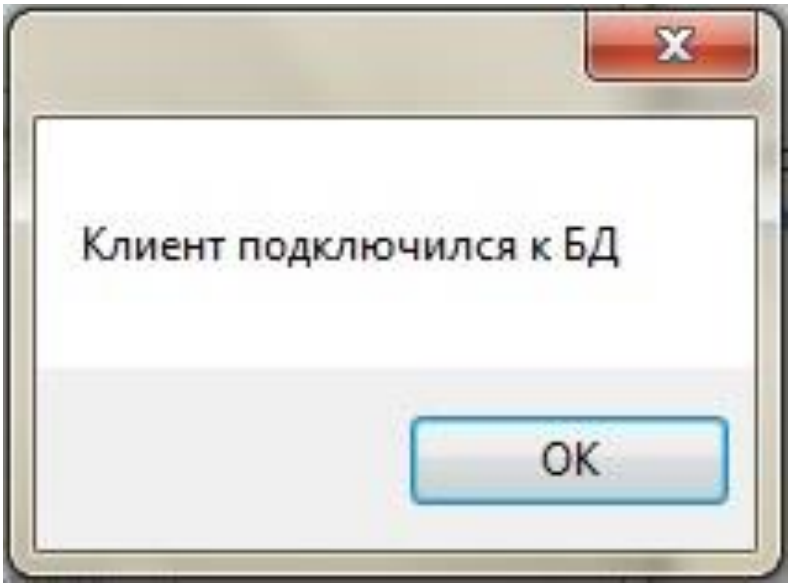


Рис. 17. Сообщение о подключении

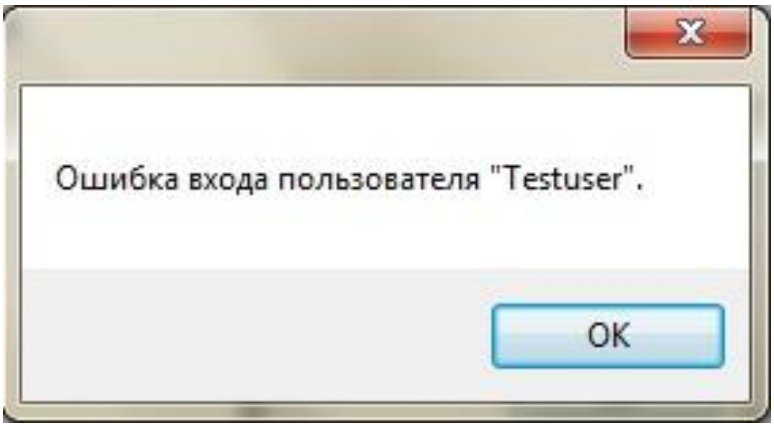


Рис. 18. Сообщение об ошибке подключения

SqlConnection - Представляет открытое подключение к базе данных SQL Server. Через него инициализирует новый экземпляр класса SqlConnection после получения строки, содержащей строку соединения.

Open() - открывает подключение к базе данных со значениями свойств, определяемыми объектом ConnectionString. (Переопределяет DbConnection.Open().) Соответственно Close() закрывает подключение.

2.2. Команды обработки запросов

2.2.1. Команды SqlCommand и SqlDataReader

Теперь создадим 2 форму (рис. 19) и рассмотрим команды.

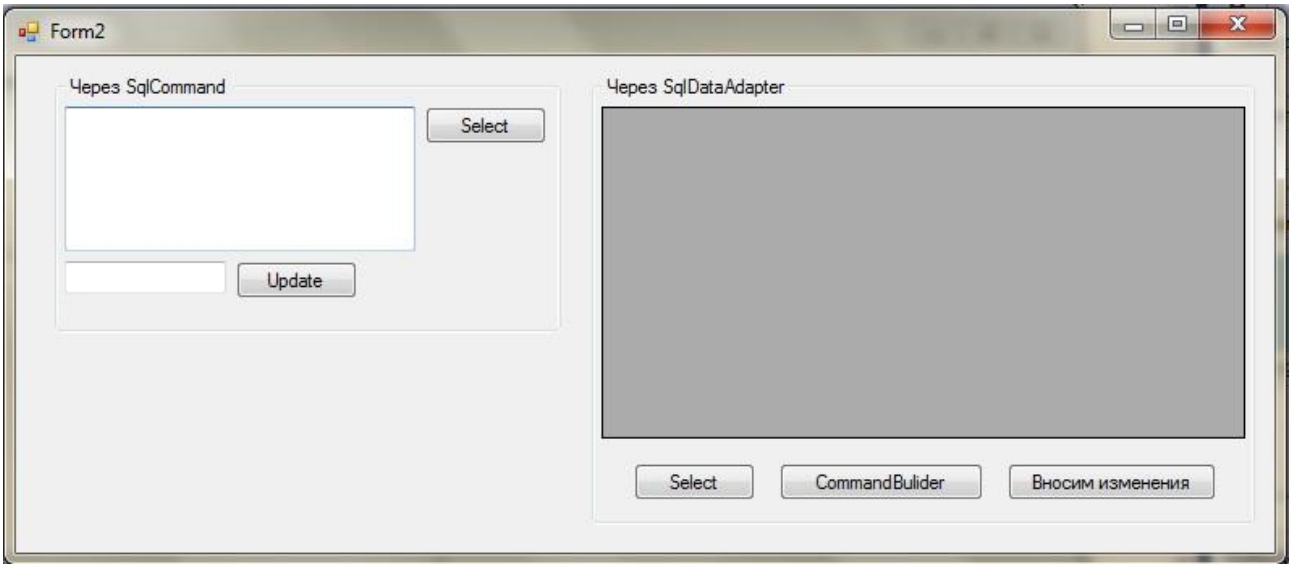


Рис. 19. Внешний вид второй формы приложения

После установки подключения мы можем выполнять к базе данных какие-либо команды, например, добавить в базу данных объект, удалить, изменить его или просто извлечь. Команды представлены объектом интерфейса System.Data.IDbCommand. Провайдер для MS SQL предоставляет его реализацию в виде класса SqlCommand. Этот класс инкапсулирует sql-выражение, которое должно быть выполнено.На пример, сделаем запрос на извлечение данных и выведем IDи ФИО полученные в ходе обработки запроса.

```
OpenSql();
stringqq="SELECT * FROM Студенты";
SqlCommandcomm=newSqlCommand();
comm.Connection=dbConnection;
comm.CommandText=qq;
SqlDataReader reader =comm.ExecuteReader();
textBox1.Text=reader.GetName(0)+"\t"+reader.GetName(1)+"\n";
while(reader.Read())
{
textBox1.AppendText(reader.GetInt32(0).ToString()+"\t"+reader.GetStri
ng(1)+" \n");
}
CloseSql();
```

В итоге получим (рис. 20):

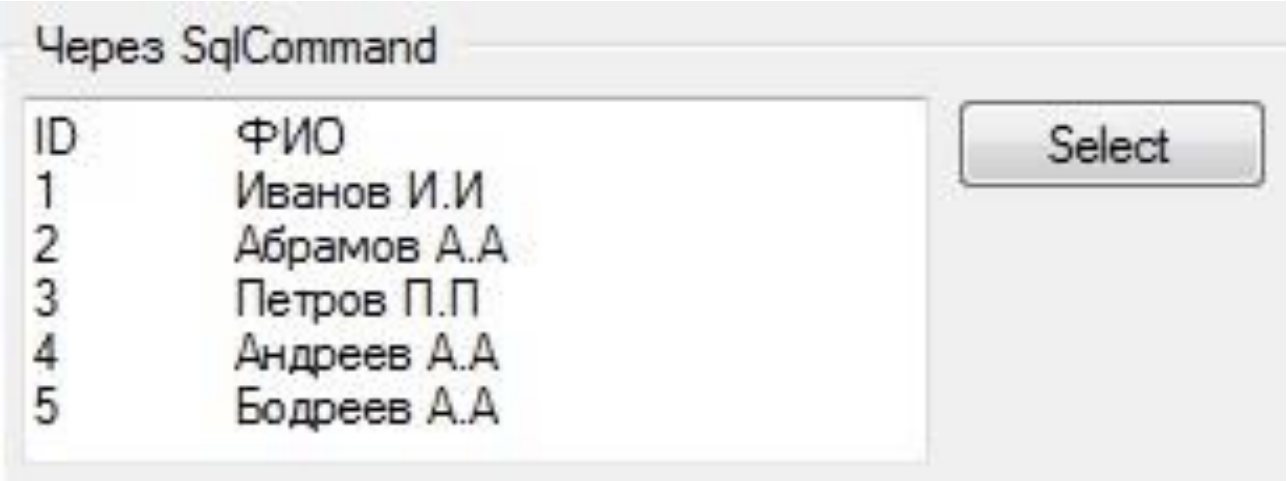


Рис. 20. Результат выполнения запроса

С помощью свойства **CommandText** устанавливается SQL-выражение, которое будет выполняться. В данном случае это запрос на получение всех объектов из таблицы «Студенты». А с помощью свойства **Connection** можно установить объект подключения SqlConnection.

Чтобы выполнить команду, необходимо применить один из методов SqlCommand:

- **ExecuteNonQuery**: просто выполняет sql-выражение и возвращает количество измененных записей. Подходит для sql-выражений INSERT, UPDATE, DELETE.

- **ExecuteReader**: выполняет sql-выражение и возвращает строки из таблицы. Подходит для sql-выражения SELECT.

- **ExecuteScalar**: выполняет sql-выражение и возвращает одно скалярное значение, например, число. Подходит для sql-выражения SELECT в паре с одной из встроенных функций SQL, как например, Min, Max, Sum, Count.

Этот метод возвращает объект **SqlDataReader**, который используется для чтения данных.

Reader.GetName() - выводит заголовки таблицы.

С помощью метода reader.Read() ридер переходит к следующей строке и возвращает булевое значение, которое указывает, есть ли данные для считывания.

reader.GetInt32(0) – возвращает столбец типа int, a GetString(1) типа text, т. е. стоит учитывать, какой тип имеет столбец и как расположены таблицы (самым 1 идет ID, а ФИО идет за ID).

Возьмем пример для обновления таблицы, где поменяем ФИО студента под ID = 1:

```
OpenSql();
stringqq="UPDATE Студенты SET ФИО = '"+ textBox2.Text+"' WHERE ID=1";
SqlCommandcomm=newSqlCommand();
comm.Connection=dbConnection;
comm.CommandText=qq;
SqlDataReader reader =comm.ExecuteReader();
CloseSql();
```

В итоге получим (рис.21):

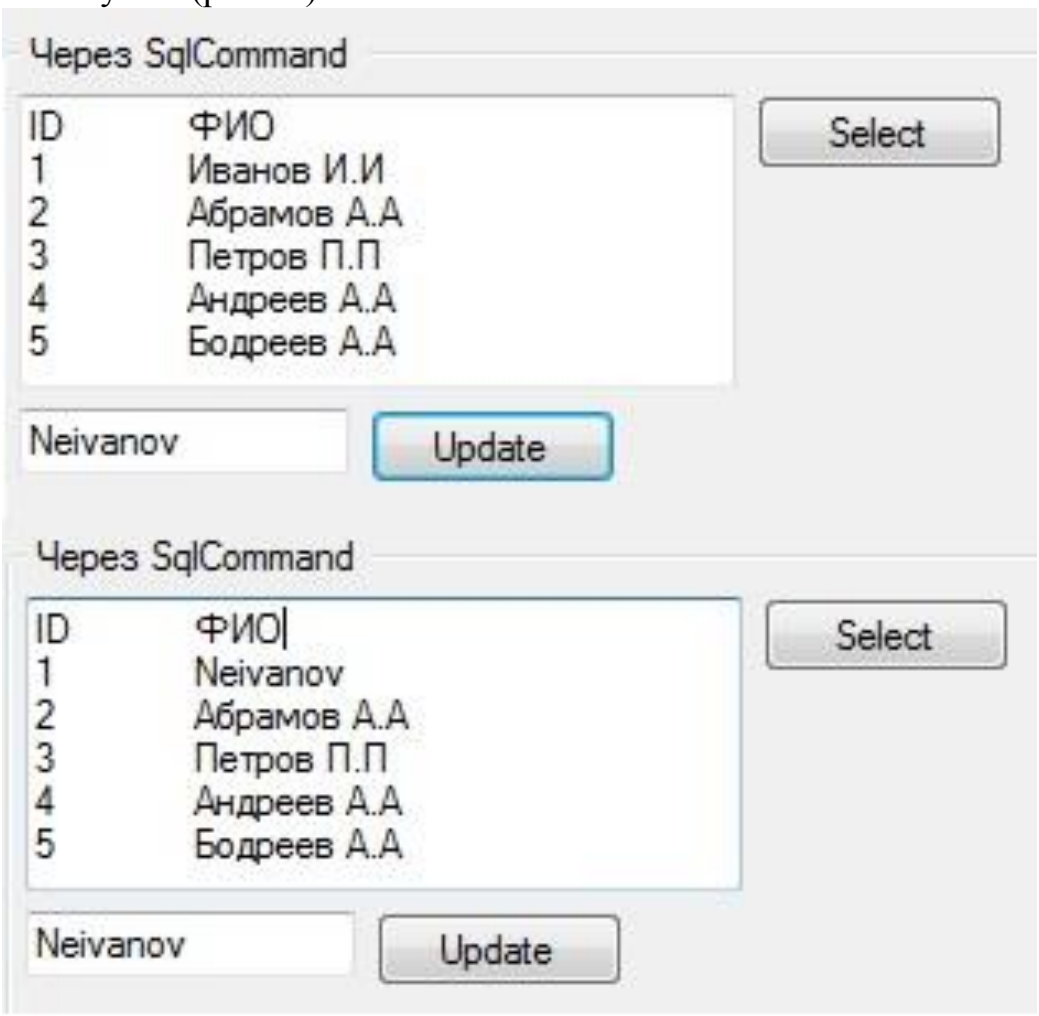


Рис. 21. Результат выполнения запроса

Для INSERET и DELETE анлогично.

2.2.2. Команды SqlDataAdapter и DataSet

Для заполнения таблицы (DataGridView) эта команда не удобна, слишком много писать программного кода, поэтому будем использовать **SqlDataAdapter** и **DataSet.**

Ранее для получения данных мы использовали объект SqlDataReader, с помощью которого построчно можно перебрать ответ от сервера базы данных. Но есть и другой способ, который демонстрирует использование объектов SqlDataAdapter и DataSet. DataSet представляет хранилище данных, с которыми можно работать независимо от наличия подключения, а SqlDataAdapter заполняет DataSet данными из БД.

Для начала создадим:

```
SqlDataAdapterda;
DataSetds;
```

Создадим запрос:

```
OpenSql();
stringqq="SELECT * FROM Студенты";
```

```
// СоздаемобъектDataAdapter
da=newSqlDataAdapter(qq, dbConnection);
// Создаемобъект Dataset
ds=newDataSet();
//Заполняем Dataset
da.Fill(ds,"Студенты");
// Отображаемданные
dataGridView2.DataSource=ds.Tables["Студенты"];
CloseSql();
```

В итоге получим (рис. 22):

ID	ФИО	Паспорт	ГР
1	Neivanov		
2	Абрамов А.А		
3	Петров П.П		05.05.1
4	Андреев А.А		
5	Бодреев А.А		
6	Фадеев Ф.А		
8	хавров И.И	784568	

Рис. 22. Результат выполнения запроса

SqlDataAdapter - представляет набор команд данных и подключение к базе данных, которые используются для заполнения DataSet и обновления базы данных SQL Server.

Fill(DataSet) - Добавляет или обновляет строки в DataSet.

В конструкторе формы в DataGridView загружаются данные. Для загрузки данных создается объект SqlDataAdapter, который принимает объект подключения и sql-выражение SELECT. Затем создается объект DataSet и с помощью метода adapter.Fill() в него загружаются данные. Дальше происходит установка источника данных для DataGridView.

В качестве источника устанавливается одна из таблиц в DataSet. Каждая таблица представляет объект DataTable, и в DataSet может быть определено несколько таких таблиц.

Получив данные в DataSet, мы можем производить с ними различными операции: удалять, изменять, добавлять новые записи. Однако все делаемые нами изменения автоматически не будут сохраняться в БД. Для этого нам еще надо вызвать метод Update объекта SqlDataAdapter, который заполнял DataSet.

Для модификации данных в БД в соответствии с изменениями в DataSetSqlDataAdapter использует команды InsertCommand, UpdateCommand и

DeleteCommand. Мы можем сами определить для этих команд sql-выражения, либо мы можем воспользоваться классом SqlCommandBuilder, который позволяет автоматически сгенерировать нужные выражения.

Добавим:

```
SqlCommandBuildercb;


OpenSql();
stringqq="SELECT * FROM Студенты";
// СоздаемобъектDataAdapter
da=newSqlDataAdapter(qq, dbConnection);
// создаемобъектSqlCommandBuilder
cb=newSqlCommandBuilder(da);
// Создаемобъект Dataset
ds=newDataSet();
//Заполняем Dataset
da.Fill(ds,"Студенты");
// Отображаемданные
dataGridView2.DataSource=ds.Tables["Студенты"];
CloseSql();
```

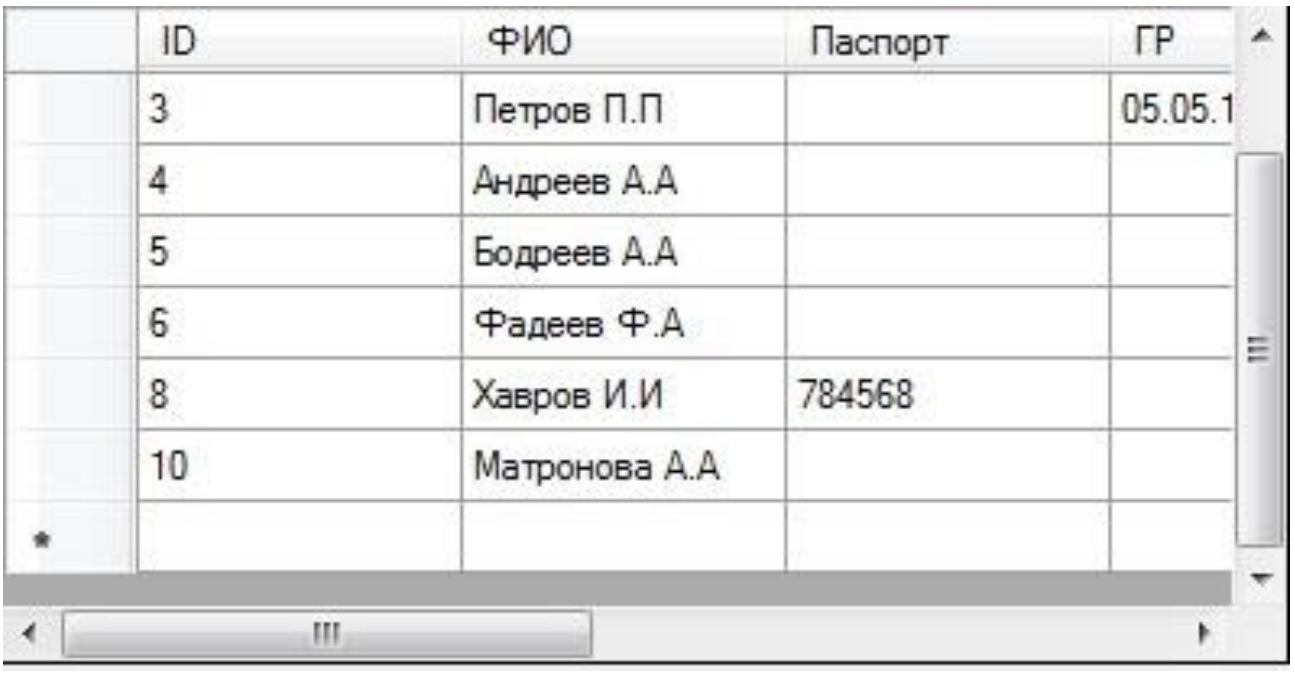
Получим то же самое, как и в первом случае, но теперь SqlDataAdapter имеет свойства SqlCommandBuilder, которое позволит нам автоматически выполнять UPDATE, DELETEили INSERT. Для этого изменения в таблице, добавим нового студента и удалим одного (рис. 23).

Создадимобработчик:

```
da.Update((DataTable)dataGridView2.DataSource);
```

Рис. 23. Изменения в таблице

В итоге получим (рис. 24):



	ID	ФИО	Паспорт	ГР
	3	Петров П.П		05.05.1
	4	Андреев А.А		
	5	Бодреев А.А		
	6	Фадеев Ф.А		
	8	Хавров И.И	784568	
	10	Матронова А.А		
*				

Рис. 24. Результат выполнения запроса

Все что описано выше – это основы и самые простые примеры, которых хватит для создания простого приложения. Можно так же использовать хранимые процедуры, они являются еще одной формой выполнения запросов к базе данных. Но по сравнению с ранее рассмотренными запросами, которые посылаются из приложения базе данных, хранимые процедуры определяются на сервере и предоставляют большую производительность и являются более безопасными.

3. ПРАКТИЧЕСКОЕ ЗАДАНИЕ

3.1. Создание Базы данных

Рассмотрим на примере БД для ресторана создание базы данных. В базе должна храниться следующая информация:

- меню ресторана, с указанием информации об изготавливаемых блюдах и их составе;
- поставка продуктов;
- информация о резервировании столиков, столики могут быть разного типа;
- информация о заказах ресторана, с указанием столика и принимающего заказ официанта.

Для разработки БД для ресторана используем СУБД MS SQL Server. Ресторан имеет склад, где хранятся продукты, для блюд, и напитки с указанием их количества и срока годности. При поставке продуктов и напитков учитывается их количество и дата поставки. Затем составляется меню для блюд: стоимость блюд, рассчитывается с учетом цены продуктов и требуемого количества. В ресторане отдельное меню напитков. Ресторан резервирует столики на будущее или на текущий момент, так же оформляет и хранит заказы посетителей и проводит оплату заказов с распечаткой чеков.

Для оболочки приложения, которое будет работать с базой данных, будем использовать C#. Доступ к приложению должен быть разделен: для администратора и для обслуживающего персонала. Администратор должен мониторить текущие и прошедшие заказы и резервированные столики, без оформления и проведения оплаты, заказывать продукты и напитки, составлять и убирать блюда из меню, нанимать и увольнять официантов. Персонал должен оформлять, изменять или отменять заказы и столики, проводить оплату заказов.

Для создания базы данных можно использовать SQL Server Management Studio или написать скрипт, который создаст базу данных. Рассмотрим наиболее простой способ создания БД. В SQL Server Management выбираем пункт меню «Создать запрос».

Для создания БД набираем следующий текст запроса T-SQL (рис. 25):

```
useMaster;
createdatabase Restaurant;
GO
```

После выполнения запроса (нажимаем «Выполнить» или клавишу F5) будет создана БД Restaurant.

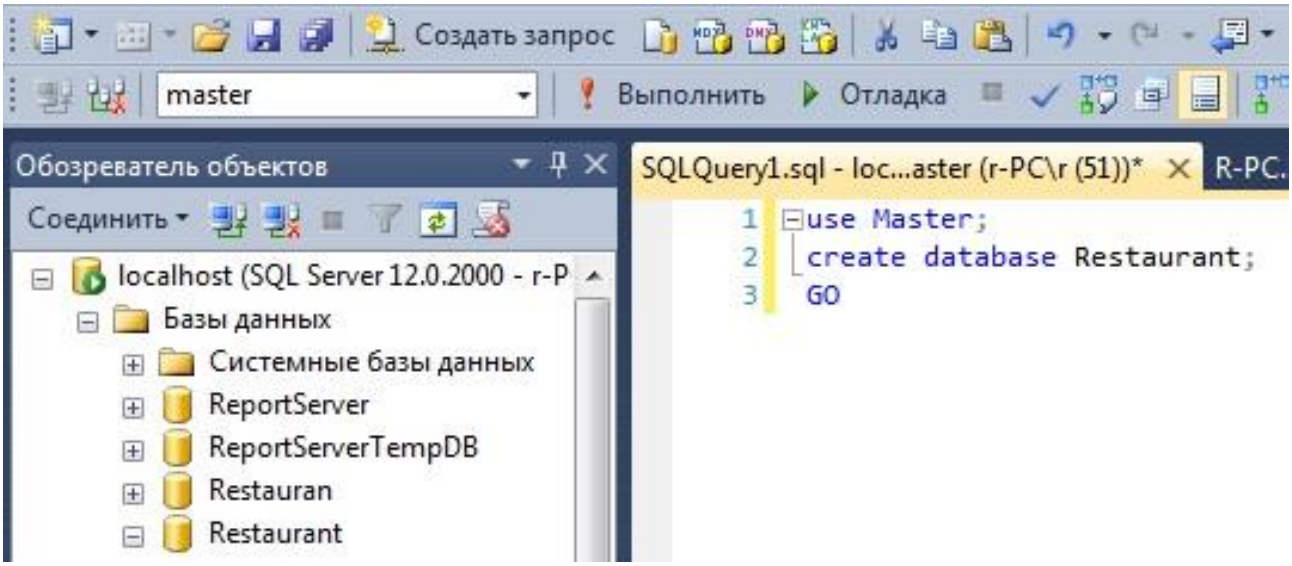


Рис.25. Создание БД Restaurant

Если посмотреть структуру вновь созданной БД, то можно увидеть, что в БД находится достаточно большое число объектов, например, были созданы пользователи БД, роли БД, схемы, системные представления и т.д. База данных создается на основе БД model, которая используется в качестве шаблона создаваемой БД. Если часто создаются базы данных, в которых используются одни и те же объекты пользователей, то можно внести их в БД model и они будут присутствовать во всех вновь создаваемых базах данных.

Примечание. При выполнении инструкции **CREATE DATABASE** первая часть базы данных создается путем копирования в нее содержимого базы данных model. Оставшаяся часть новой базы данных заполняется пустыми страницами. Если требуются, какие-то другие настройки, то можно через свойства БД изменить их.

3.2. Создание таблиц и связей

Для создания таблиц так же, как и для создания БД используется SQL Server Management Studio или пишутся скрипты. Существует специальная область SQL, называемая DDL (о ней мы говорили в предыдущем пособии (Михайлова У.В., Баранкова И.И., Лукьянов Г.И. «Разработка БД В MS SQL Server с использованием SSMS»)), которая специально работает над созданием объектов данных.

Для создания таблицы правой кнопкой мыши по БД вызываем контекстное меню (в нашем случае Restaurant) →Создать запрос (рис. 26).

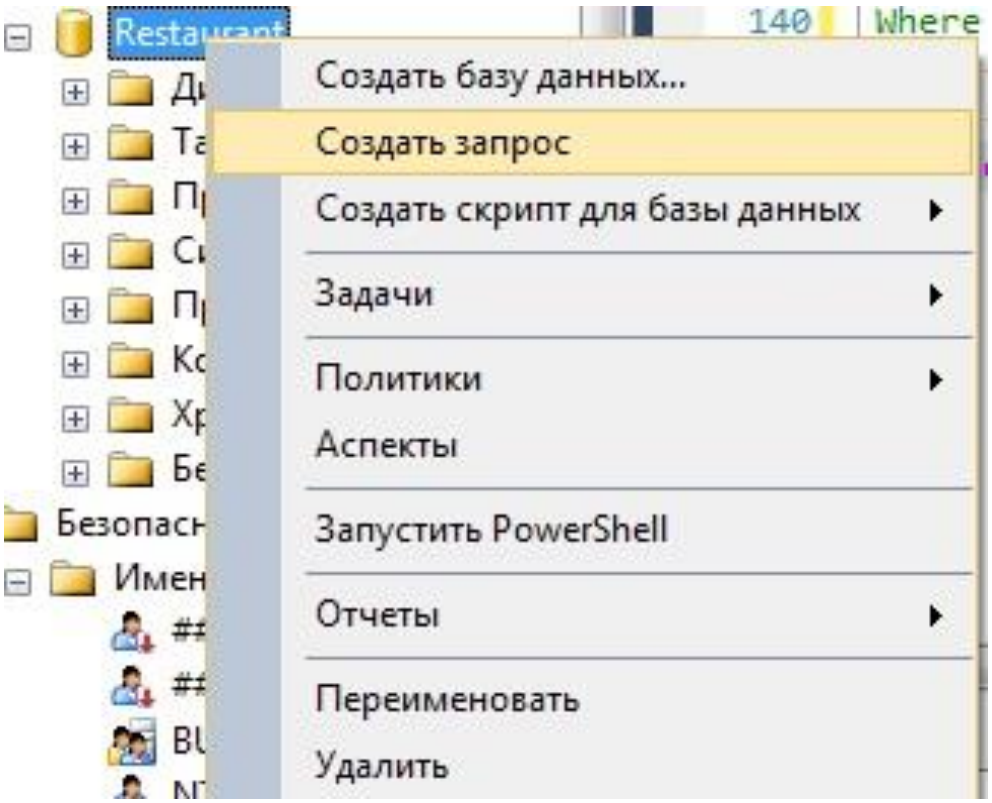


Рис. 26. Создание запросов

При этом создается файл с расширение SQL, в котором будут храниться запросы.

И так для создания таблицы в самом начале набираем инструкцию «use» и имя БД находящейся на сервере. Дальше идет создание таблиц и их параметры. Инструкция CREATE TABLE [имя таблицы] создает пустую таблицу с указанным именем. Команда CREATE TABLE задает имя таблицы, столбцы таблицы в виде описания набора имен столбцов, указанных в определенном порядке, а также она может определять главный и вторичные ключи таблицы. Кроме того, она указывает типы данных и размеры столбцов. Каждая таблица должна содержать, по крайней мере, один столбец

Пример:

```
use Restaurant

createtable Продукты
(
  [ID Продукта] intidentity(1,1)primarykey notnull,
Название nvarchar(50)notnull,
Категория nvarchar(30),
Стоимость real notnull,
  [Срокхранения] int notnull
)

CreateTable Поставка
(
  [№ Поставки] intidentity(1,1)primarykey notnull,
Датаdatenotnull,
Количествоrealnotnull,
```

```
  [ID Продукта] int foreign key references Продукты([Id Продукта])on delete
Cascadeon update cascade
)
```

Примечание. Имена таблиц должны соответствовать правилам для идентификаторов, а именно TABLE_NAME не может превышать 128 символов, за исключением имен локальных временных таблиц.

Аналогично примеру создайте все таблицы, которые входят в схему данных вашей базы данных (рис. 27).

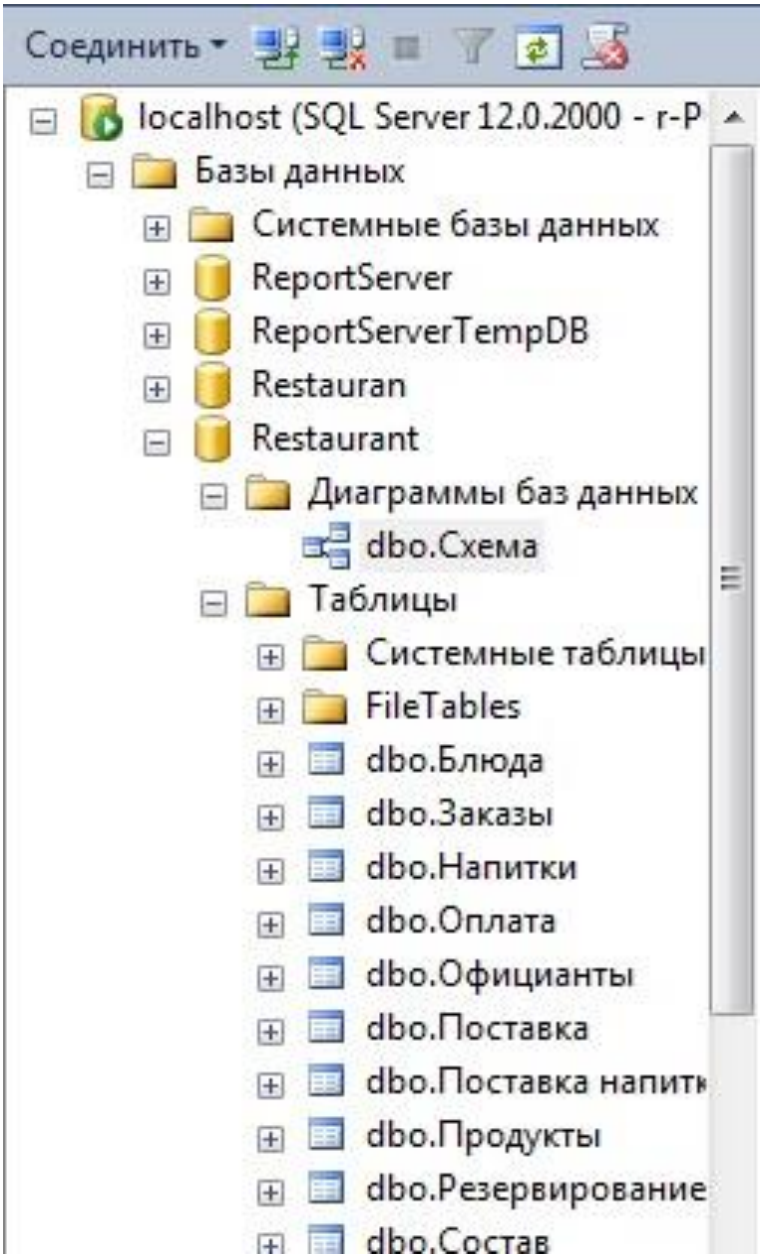


Рис. 27. Результат выполнения запроса на создание всех таблиц

Для редактирования таблиц правой кнопкой мыши на таблице вызывается контекстное меню → Изменить первые 200 строк… → откроется конструктор где можно напрямую заполнить таблицу (рис. 28).

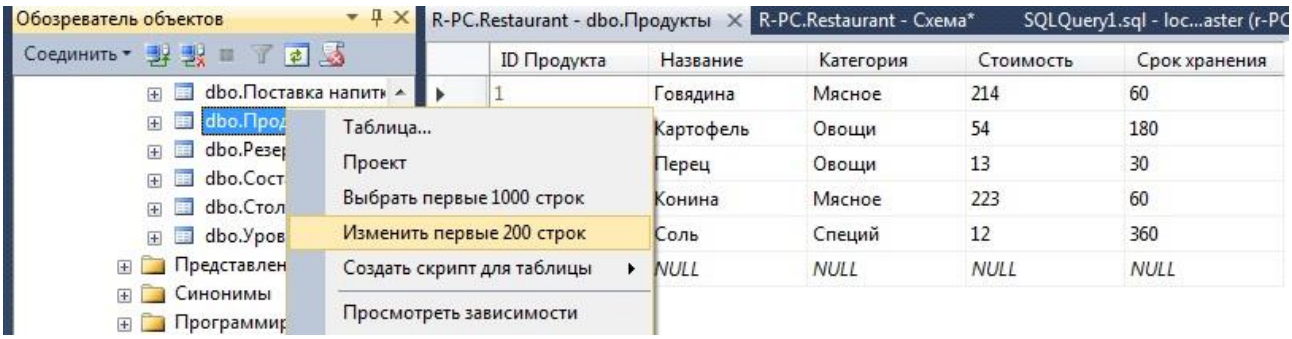


Рисунок 28 - Редактирование таблицы

В результате должна получиться схема данных базы данных Ресторан (рис. 29).

Рисунок 29 – Получившаяся схема данных БД Ресторан

3.3. Создание нескольких пользователей

Для нашего приложения требуется создать несколько пользователей для работы с БД. Для этого создаем имена входа с разными правами и привязываем их к БД.

Для этого выбираем пункт Безопасность → Имена входа → «Создать имя входа…» → Набираем имя → Проверка подлинности SqlServer→ Набираем пароль (рис. 30).

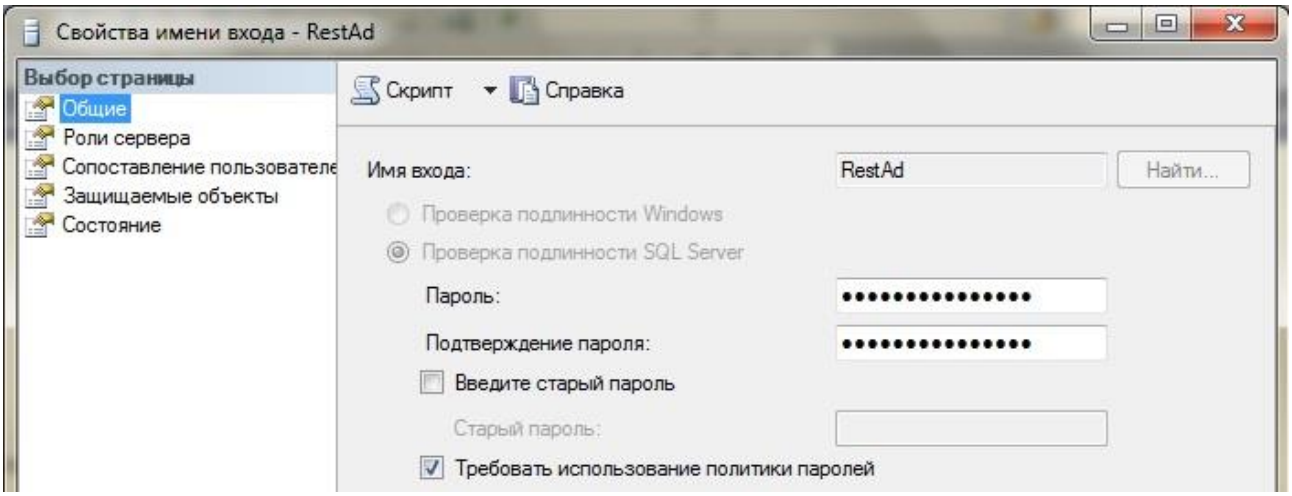


Рис. 30. Создание входа для различных пользователей

В Роли сервера ставим галочки public и sysadmin. В сопоставление пользователя ставим галочку для нашей БД (Restaurant), в результате создастся пользователя для БД. Затем в БД на вкладке Безопасность → Пользователи → Выбираем пользователя → Свойства → Членство (рис. 31).

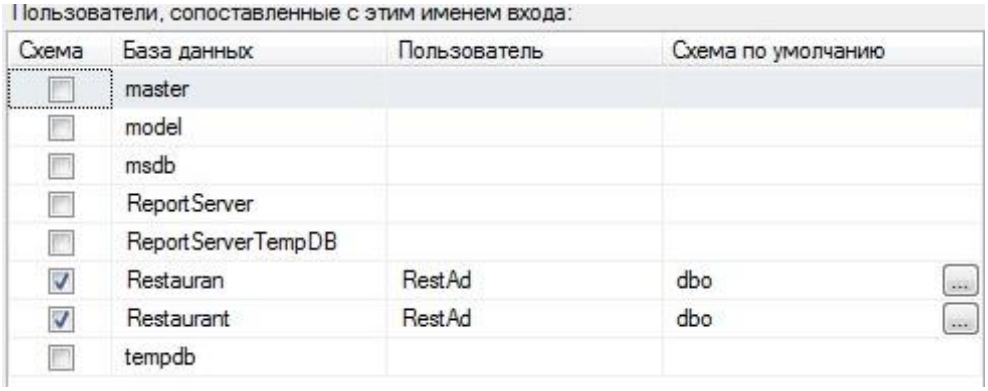


Рис. 31. Сопоставление пользователя

Oficante–этот пользователь которой в пользуется запросами «Select» «Update » «Delete» и ему даем права: **db_datareader, db_writer, db_dlladmin**

RestAd - кроме выше перечисленных так же отвечает за безопасность БД по этому добавляем **db_securityadmin**.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. RangaRengarajanT.K. SQLServer 2016 publicpreviewcomingthissummer [Электронный ресурс] – Режим доступа: https://blogs.technet.microsoft.com/dataplatforminsider/2015/05/04/sql-server-2016-public-preview-coming-this-summer/ свободный. – Загл. С экрана. – Яз.англ.
2. Михайлова У.В., Хусаинов А.А. Особенности и проблемы, возникающие при разработке моделей угроз информационной безопасности // Безопасность информационного пространства: сб. материалов XV Всеросс. науч.-практич. конф. студентов, аспирантов и молодых ученых. под ред. Д.И. Дик. Курган: ФГБОУ ВО «Курганский государственный университет». - 2016. С. 72-75.
3. Mikhailova U.V., Barankova I.I., Lu'yanov G.I. Automated control system of a factory rail way transport based on ZIGBEE // 2016 2nd International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM) Proseedings. - 2016.
4. Михайлова У.В., Ершов В.А. Способы организации и методы противодействия DOS/DDOS – атакам // Безопасность информационного пространства: сборник трудов XIII Всероссийской научно-практической конференции студентов, аспирантов и молодых учёных. Министерство образования и науки Российской Федерации, Южно-Уральский государственный университет, Кафедра «Безопасность информационных систем». -2015. С. 73-79.
5. Баранкова И.И., Михайлова У.В., Лукьянов Г.И. Прогнозирование локальных и внешних угроз на информационные серверы предприятия // Актуальные проблемы современной науки, техники и образования. - 2017. - Т. 1. С. 217-220.
6. Баранкова И.И., Михайлова У.В. Особенности формирования оценочных средств для сформированности компетенций специалиста по информационной безопасности // Информационное противодействие угрозам терроризма. - 2015. - Т. 2. № 25. С. 26-30.
7. Михайлова У.В., Лукьянов Г.И. Эффективность применения СЗИ от утечки по акустическим каналам // Вестник УрФО. Безопасность в информационной сфере. - 2014. - № 4 (14). С. 14-18.
8. Лукьянов Г.И., Михайлова У.В., Баранкова И.И., Коновалов М.В. Защита информации по виброакустическим каналам с использованием СЗИ «СОНАТА» // Актуальные проблемы современной науки, техники и образования. - 2015. - Т. 2. № 1. С. 186-188.
9. Баранкова И.И., Михайлова У.В., Лукьянов Г.И. Анализ методик оценки звукоизоляционных свойств ограждающих конструкций // Актуальные проблемы современной науки, техники и образования. - 2017. - Т. 1. № 1. С. 211-214.

10. Баранкова И.И., Михайлова У.В., Самохвал В.Д., Огонесян Ш.У. Анализ информационных угроз ВУЗА // Актуальные проблемы современной науки, техники и образования. - 2013. - Т. 2. - № 71. С. 157-159.
11. Коновалов М.В., Михайлова У.В., Хусаинов А.А., Санарбаев Р.Ж. Алгоритмы шифрования данных // Актуальные проблемы современной науки, техники и образования. - 2013. - Т. 2. - № 71. С. 159-161.
12. Михайлова У.В., Коновалов М.В., Гуринец К., Кучербаева Э.Ф. Идентификация личности // Актуальные проблемы современной науки, техники и образования. - 2013. - Т. 2. - № 71. С. 164-166.
13. Михайлова У.В., Аименева А.А., Полехина А.В. Технические средства защиты информации // Актуальные проблемы современной науки, техники и образования. - 2012. - Т. 2. - № 70. С. 27-30.
14. Михайлова У.В., Поступная А.П., Хасанова Е.Р. Защита информации по виброакустическим каналам // Актуальные проблемы современной науки, техники и образования. - 2012. - Т. 2. - № 70. С. 31-33.
15. Баранкова И.И., Михайлова У.В., Лукьянов Г.И. DLP система: защита от утечки информации. Анализ поиска WORDSEARCH // Актуальные проблемы современной науки, техники и образования. - 2016. - Т. 1. - № 1. С. 187-191.
16. Баранкова И.И. Компетентностно-ориентированный подход к формированию лабораторий учебно-тренировочных средств при подготовке специалистов в области информационной безопасности // Информационное противодействие угрозам терроризма. - 2015. - Т. 1. - № 25. С. 46-50.
17. Баранкова И.И., Гуринец К.Р., Хусаинов А.А., Санарбаев Р.Ж. Применение алгоритма шифрования методом Цезаря для шифрования данных, представленных в текстовом формате // Актуальные проблемы современной науки, техники и образования. - 2014. - Т. 2. - № 1. С. 152-155.
18. Носова Т.Н., Быкова Т.В., Булатов Р.Р., Михайлова У.В. Защита баз данных ORACLE / Актуальные проблемы современной науки, техники и образования. 2015. Т. 2. № 1. С. 188-191.
19. Баранкова И.И., Михайлова У.В., Лукьянов Г.И. Техническая защита информации: Учебное пособие. Электронное издание / Магнитогорск, 2017.
20. Баранкова И.И., Михайлова У.В., Лукьянов Г.И. Система прогнозирования транспортного графика электропоездов промышленного предприятия / Электротехнические системы и комплексы. 2017. № 3 (36). С. 66-70.

Учебное текстовое электронное издание

Баранкова Инна Ильинична
Михайлова Ульяна Владимировна
Лукьянов Георгий Игоревич

РАЗРАБОТКА ПРИЛОЖЕНИЙ НА C# ДЛЯ РАБОТЫ С БАЗАМИ ДАННЫХ

Практикум

1,36 Мб
1 электрон. опт. диск

г. Магнитогорск, 2018 год
ФГБОУ ВО «МГТУ им. Г.И. Носова»
Адрес: 455000, Россия, Челябинская область, г. Магнитогорск,
пр. Ленина 38

ФГБОУ ВО «Магнитогорский государственный
технический университет им. Г.И. Носова»
Кафедра информатики и информационной безопасности
Центр электронных образовательных ресурсов и
дистанционных образовательных технологий
e-mail: ceor_dot@mail.ru