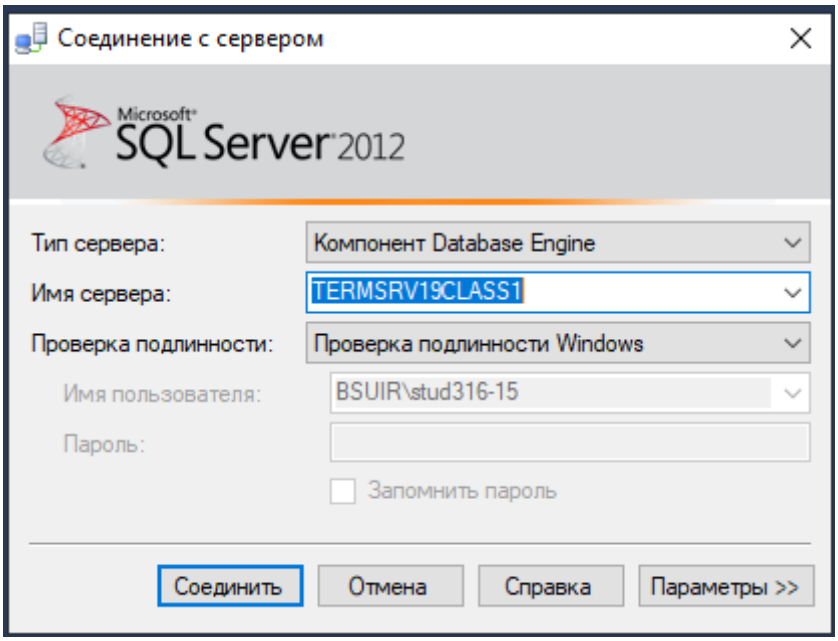
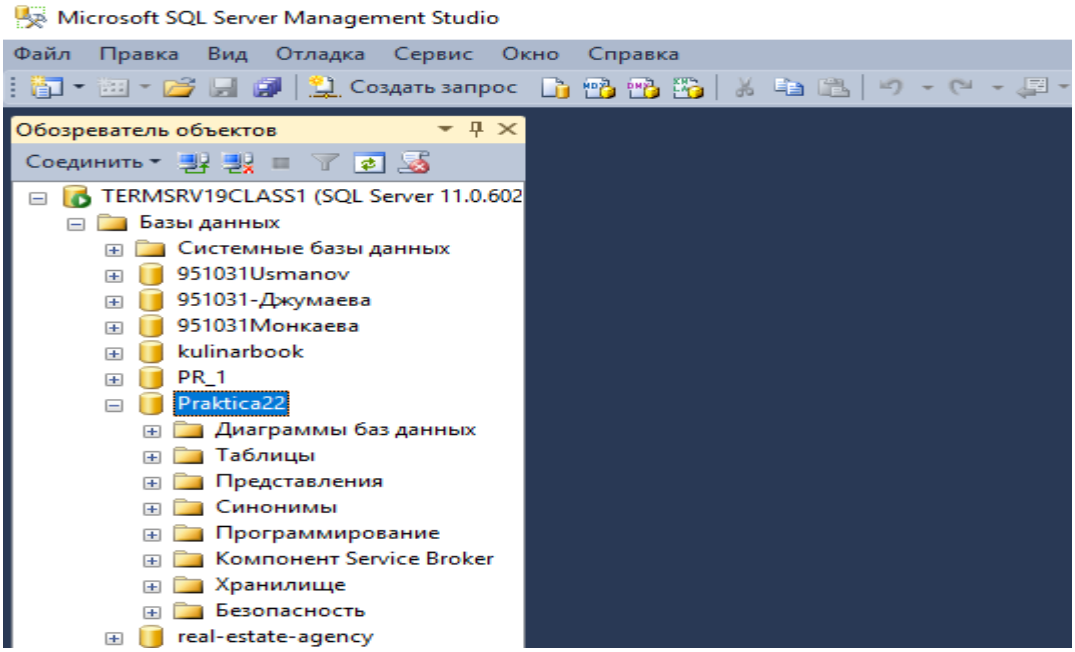
Лабораторная работа № 3.

Построение SQL запросов при помощи SQL Server Management Studio.

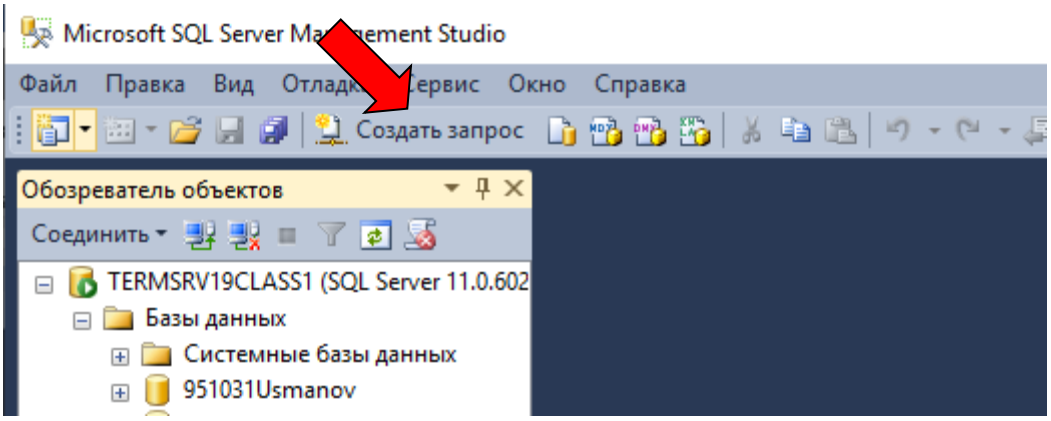
Для начала работы запустите среду SQL Server Management Studio известным Вам способом и установите соединение с сервером СУБД.



Раскройте в обозревателе объектов (слева) строку «Базы данных» и далее выберите ранее созданную Вами БД и также раскройте ее объекты. Выделите строку с названием Вашей БД.



Откройте окно редактора SQL запросов, нажав на панели инструментов среды SQL Server Management Studio кнопку «Создать запрос».



Прочитайте теоретический материал и выполните задания для самостоятельной работы (в конце) с оформлением отчета и обязательными скриншотами результатов.

1. Использование условий поиска для отбора строк.

В общем случае запросы начинаются с ключевого слова SELECT. Если необходимо получить все поля из таблицы, используют символ *, далее идет предложение FROM после которого указываются имена используемых таблиц.

Например: SELECT *
FROM STUDENTS;

Данный запрос выводит значения всех полей из таблицы STUDENTS. Если необходимо извлечь строго определенные поля, то они указываются через запятую после предложения SELECT.

При работе с данными часто необходимо устранять избыточные данные. Это реализуется при помощи команды DISTINCT. Т. о. DISTINCT не дает возможность данным дублироваться. Если вместо DISTINCT используется ALL, то это будет иметь противоположный эффект и дублирование строк сохранится.

Для использования условий поиска существует предложение WHERE.

Например: SELECT *
FROM STUDENTS
WHERE STIP>0;

В предложении WHERE могут использоваться следующие наборы операторов для сравнения:

- = равно,
- > больше,
- < меньше,
- <= меньше или равно,

- >= больше или равно,
- <> не равно.

Эти операторы имеют стандартные значения для числовых данных, а для символьных их определение зависит от кодов ASCII, они следуют в алфавитном порядке, причем заглавные буквы имеют код больше, чем строчные.

Стандартными булевыми операторами, используемыми в SQL являются: AND, OR, NOT.

В предложении SELECT в дополнение к рассмотренным выше операторам могут быть использованы операторы IN, BETWEEN, LIKE, IS NULL.

- IN определят набор значений из списка (аналогичен OR).
- BETWEEN определяет диапазон значений.
- LIKE – ищет подстроки.
- IS NULL – ищет неизвестные значения (со значением NULL).

Для получения итоговых данных используются следующие агрегатные функции:

- COUNT – производит подсчет количества строк или не NULL значений полей, которые выбрал запрос.
- SUM – рассчитывает арифметическую сумму всех выбранных значений данного поля.
- AVG – производит усреднение всех выбранных значений данного поля.
- MAX – находит и возвращает максимальное из всех значений данного поля.
- MIN – находит и возвращает минимальное значение из данного поля.

Для упорядочения вывода полей таблиц используют команду ORDER BY, позволяя сортировать вывод запроса согласно значениям в том или ином количестве выбранных столбцов. Если указывать несколько полей, то столбцы вывода должны быть упорядочены один внутри другого, при этом можно определить возрастание(ASC) или убывание (DESC). По умолчанию установлено возрастание.

Часто возникает необходимость в выборе информации из нескольких таблиц. Вариантом такого вывода является объединение результатов нескольких запросов, выполняющихся независимо друг от друга. Для размещения нескольких запросов вместе и объединения их вывода используют предложение UNION. Предложение UNION объединяет вывод двух или более SQL – запросов в единый набор строк и столбцов. Когда запросы подвергаются объединению, их столбцы вывода должны быть совместимы для объединения. Это означает для каждого запроса необходимость включения одинакового числа

столбцов в одинаковом порядке и при этом должна присутствовать совместимость типов.

2. Объединение таблиц.

В многотабличном запросе можно объединять данные таблиц, однако, та же самая методика может использоваться для объединения вместе двух копий одиночной таблицы. Для объединения таблицы с собой можно сделать каждую строку таблицы одновременно и комбинацией ее с собой и комбинацией с каждой другой строкой таблицы, а затем оценить каждую комбинацию в терминах предиката. Это позволяет легко создавать определенные виды связей между различными элементами внутри одиночной таблицы. Например, допускается изобразить объединение таблицы с собой как объединение двух копий одной и той же таблицы, причем она на самом деле не копируется, но SQL выполняет команду так, как если бы это было сделано.

Использование команды для объединения таблицы с собой аналогично тому приему, который используется для объединения нескольких таблиц. Когда объединяется таблица с собой, все повторяемые имена столбца заполняются префиксами имени таблицы, чтобы ссылаться к этим столбцам внутри запроса, необходимо иметь два различных имени для этой таблицы. Это можно сделать с помощью определения временных имен, называемых псевдонимами, которые определяются в предложении FROM запроса. Синтаксис в этом случае следующий после имени таблицы оставляют пробел, а затем должен следовать псевдоним для нее. В данном случае SQL ведет себя так, как если бы он соединял две различные таблицы, называемые предположим FIRST и SECOND, т е. псевдонимы разрешают одной и той же таблице быть обработанной независимо. Псевдонимы могут использоваться в предложении SELECT до их объявления в предложении FROM, однако SQL будет сначала допускать любые псевдонимы и может отклонить команду, если они не будут определены далее в запросе. Кроме того, необходимо помнить, что псевдоним существует только тогда, когда команда выполняется, а после завершения запроса псевдонимы, используемые в нем, больше не имеют никакого значения.

Например, если считать, что учебный предмет может вести только один преподаватель, то всякий раз в таблице PREDMET необходима проверка на это условие. При этом каждый раз, когда код предмета появляется в таблице PREDMET, он должен совпадать с соответствующим номером преподавателя. Следующая команда будет определять любые несогласованности в этой области:

```
SELECT FIRST.PNUM, FIRST.TNUM,
SECOND.PNUM,  SECOND.TNUM  FROM  PREDMET  FIRST,
PREDMET SECOND
```

```
WHERE FIRST.PNUM = SECOND.PNUM AND FIRST.TNUM <>
SECOND.TNUM
```

Вывода для данного примера не будет, т.к. данных, удовлетворяющих предикату в рассматриваемой таблице нет. Логика этого запроса достаточно проста: из таблицы будет выбираться очередная строка, и запоминаться под первым псевдонимом. После этого SQL начнет проверять ее в комбинации с каждой строкой таблицы под вторым псевдонимом. Если комбинация строк удовлетворяет предикату, то она выбирается для вывода, т. е. если будет найден учебный предмет, у которого выяснится несовпадение номера преподавателя в таблице под первым и вторым псевдонимом.

Объединение таблицы с собой - это наиболее часто встречающаяся ситуация, когда используются псевдонимы, однако их можно использовать в любое время для создания альтернативных имен для таблиц в запросе, например, в случае, если таблицы имеют очень длинные и сложные имена.

Более того, допускается использовать любое число псевдонимов для одной таблицы в запросе, хотя использование более двух в одном предложении SELECT часто будет излишеством. Например, для назначения стипендии на следующий семестр необходимо просмотреть все варианты комбинаций студентов с разными размерами стипендии: 25.50. 17.00 и 0.00 у. е. Тогда такой запрос будет выглядеть следующим образом:

```
SELECT FIRST.SFAM, SECOND.SFAM, THIRD.SFAM FROM
STUDENTS FIRST, STUDENTS SECOND, STUDENTS THIRD
WHERE FIRST.STIP = 25.50 AND SECOND.STIP = 17.00
AND THIRD.STIP = 0.00
```

Вывод для этого запроса:

```
SFAM        SFAM        SFAM
Поляков     Старова     Гриценко
Поляков     Старова     Котенко
Нагорный    Старова     Гриценко
Нагорный    Старова     Котенко
```

Как видно из результата, этот запрос находит все комбинации студентов с тремя различными размерами стипендии, поэтому первый столбец вывода состоит из студентов со стипендией 25.50, второй с 17.00 и последний - с 0 00. которые повторяются во всех возможных комбинациях. Интересно, что такой запрос не может быть выполнен с GROUP BY или ORDER BY, поскольку они сравнивают значения только в одном столбце вывода.

Существуют различные варианты оператора объединения JOIN.

- INNER JOIN – Объединяет записи (внутреннее объединение) на основе одного или более полей. Возвращаются только те записи, для которых имеет место соответствие по полям, указанным в JOIN.
- OUTER JOIN (LEFT, RIGHT) – (внешнее объединение)

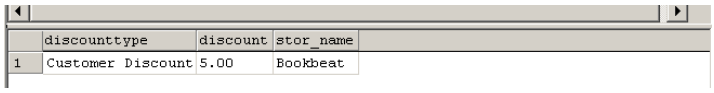
```
SELECT <список_полей>
FROM   <таблица,_которую_вы_считаете_левой>
<LEFT| RIGHT> [OUTER] JOIN
<таблица,_ которую_вы_считаете_правой>
ON <условие_объединения>
```

Опция LEFT OUTER JOIN – включает всю информацию из левой таблицы, а RIGHT OUTER JOIN – из правой таблицы.

Пусть нам необходимо знать все существующие скидки, размер каждой скидки, и какой из магазинов пользуется данной скидкой. В БД pubs есть таблицы discounts (скидки) и stores (магазины).

```
SELECT discounttype, discount, s.stor_name
FROM discounts d JOIN stores s
ON d.stor_id=s.stor_id
```

Результат:

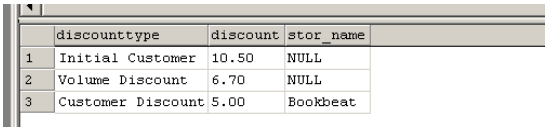


	discounttype	discount	stor_name
1	Customer Discount	5.00	Bookbeat

Этот запрос выдает только те скидки, для которых существует использующий их магазин. Результат не отвечает поставленной задаче.

```
SELECT discounttype, discount, s.stor_name
FROM discounts d
LEFT OUTER JOIN stores s
ON d.stor_id=s.stor_id
```

Результат:

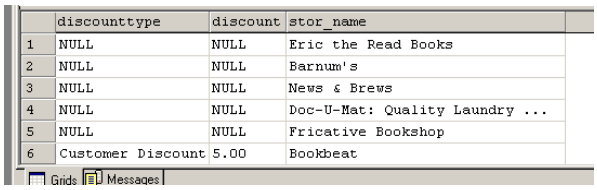


	discounttype	discount	stor_name
1	Initial Customer	10.50	NULL
2	Volume Discount	6.70	NULL
3	Customer Discount	5.00	Bookbeat

Для таблицы discounts в результат включены все строки.

```
SELECT discounttype, discount, s.stor_name
FROM discounts d
RIGHT OUTER JOIN stores s
ON d.stor_id=s.stor_id
```

Результат:



	discounttype	discount	stor_name
1	NULL	NULL	Eric the Read Books
2	NULL	NULL	Barnum's
3	NULL	NULL	News & Brews
4	NULL	NULL	Doc-U-Mat: Quality Laundry ...
5	NULL	NULL	Fricative Bookshop
6	Customer Discount	5.00	Bookbeat

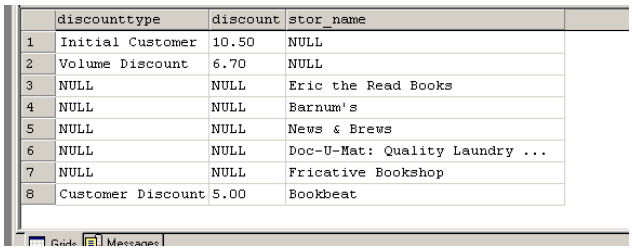
Grids Messages

Для таблицы stores в результат включены все строки. Кроме того, в результат включены, если есть соответствующие записи из discounts. Там где соответствующих записей нет, в полях таблицы discounts показаны значения NULL. Если всегда указывать таблицу discounts первой, а таблицу stores второй, то необходимо использовать LEFT OUTER JOIN, если нам нужны все скидки, и RIGHT OUTER JOIN, если нужны все магазины.

- FULL JOIN – Полное внешнее объединение таблиц получается из внутреннего объединения двух таблиц обычным способом. При этом каждая запись первой таблицы, которая не имеет связи ни с одной строкой во второй таблице, дополняется в результатах запроса значениями NULL. В то же время, каждая строка второй таблицы, которая не имеет связи ни с одной строкой первой таблицы, дополняется в результатах запроса значениями NULL.

```
SELECT discounttype, discount, s.stor_name
FROM discounts d
FULL OUTER JOIN stores s
ON d.stor_id=s.stor_id
```

Результат:



	discounttype	discount	stor_name
1	Initial Customer	10.50	NULL
2	Volume Discount	6.70	NULL
3	NULL	NULL	Eric the Read Books
4	NULL	NULL	Barnum's
5	NULL	NULL	News & Brews
6	NULL	NULL	Doc-U-Mat: Quality Laundry ...
7	NULL	NULL	Fricative Bookshop
8	Customer Discount	5.00	Bookbeat

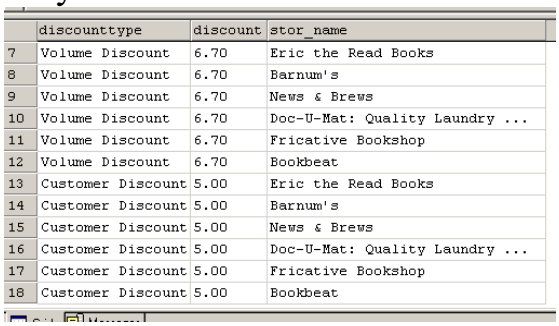
Grids Messages

- CROSS JOIN – В данном виде объединения отсутствует оператор ON (перекрестное объединение). Он объединяет

каждую запись с одной стороны JOIN с каждой записью с другой стороны JOIN.

```
SELECT discounttype, discount, s.stor_name
FROM discounts d
CROSS  JOIN stores s
```

Результат:



	discounttype	discount	stor_name
7	Volume Discount	6.70	Eric the Read Books
8	Volume Discount	6.70	Barnum's
9	Volume Discount	6.70	News & Brews
10	Volume Discount	6.70	Doc-U-Mat: Quality Laundry ...
11	Volume Discount	6.70	Fricative Bookshop
12	Volume Discount	6.70	Bookbeat
13	Customer Discount	5.00	Eric the Read Books
14	Customer Discount	5.00	Barnum's
15	Customer Discount	5.00	News & Brews
16	Customer Discount	5.00	Doc-U-Mat: Quality Laundry ...
17	Customer Discount	5.00	Fricative Bookshop
18	Customer Discount	5.00	Bookbeat

В таблице discounts было 3 записи, а в stores - 6, следовательно, в CROSS JOIN мы получим 3*6=18 записей. Данный тип объединения применяется, если необходимо построить декартово произведение всех значений объединяемых таблиц. Например, при тестировании системы перекрестным объединением можно получить большие массивы данных, не вводя их.

Альтернативный синтаксис объединений: Левое внешнее объединение двух таблиц записывается в команде WHERE в виде *=. Правое внешнее объединение двух таблиц записывается в команде WHERE в виде =*

3. Использование вложенных запросов.

В общем случае, запросы могут управлять другими запросами - это делается путем размещения запроса внутрь предиката другого, который использует вывод внутреннего запроса для установления верного или неверного значения предиката.

Чтобы выполнить основной запрос, SQL сначала должен оценить внутренний запрос (его называют подзапросом) внутри предложения WHERE. Происходит это традиционным образом, т. е. исполняется вложенный запрос, извлекающий необходимые для определения значения предиката данные, а только затем – основной. Разумеется, подзапрос должен выбрать только одно поле, а тип данных этого поля должен совпадать с тем значением, с которым он будет сравниваться

в предикате.
С другой стороны, возможна ситуация, когда подзапрос выдает в качестве результата несколько различных значений, что может сделать невыполнимым оценку предиката основного запроса, и команда выдаст ошибку. При использовании подзапросов в предикатах, основанных на реляционных операторах, обязательно нужно убедиться, что использован подзапрос, который будет отображать только одну строку вывода. Кроме того, при использовании подзапроса, который вообще не выводит никаких значений, основной запрос не выведет никаких значений: его предикат будет иметь неизвестное значение.
В некоторых случаях стоит использовать DISTINCT для того, чтобы в подзапросе получить одиночное значение.

В подзапросах допускается использование агрегатных функций - это удобно и по той причине, что они автоматически производят одиночное значение для любого числа строк, которое может быть использовано в основном предикате. Не стоит забывать, что сгруппированные агрегатные функции, определенные в терминах предложения GROUP BY, могут выдавать многочисленные значения, а значит, не допускаются в подзапросах такого характера. Даже если GROUP BY или HAVING используются так, что только одна группа значений выводится с помощью подзапроса, все равно команда будет отклонена.
Можно использовать подзапросы, которые производят любое число строк, если используется специальный оператор IN (операторы BETWEEN, LIKE, и IS NULL не могут использоваться с подзапросами). Опция IN - определяет набор значений предиката, одно из которых должно совпадать с другим. При использовании IN с подзапросом, SQL просто формирует этот набор из вывода подзапроса, а значит, допускается использование IN для того, чтобы выполнить такой подзапрос. В любой ситуации, где применяется реляционный оператор равенства (=), можно использовать IN. В отличие от первого, IN не может заставить запрос потерпеть неудачу, если подзапросом выбрано больше чем одно значение.

Когда используются подзапросы, в SQL имеется возможность обратиться к внутреннему запросу таблицы в предложении внешнего запроса FROM, с помощью соотнесенного подзапроса. При этом подзапрос выполняется неоднократно, по одному разу для каждой записи таблицы основного запроса. Строку внешнего запроса, для которой внутренний запрос каждый раз будет выполняться, будем называть текущей строкой. С учетом этого, процедура оценки, выполняемой соотнесенным подзапросом:

- выбор строки из таблицы во внешнем запросе - это текущая строка;

- сохранение значения текущей строки в псевдониме, имя которого определено в предложении FROM внешнего запроса;
- выполнение подзапроса, при этом везде, где найден псевдоним из внешнего запроса, используется значение из текущей строки (это принято называть *внешней ссылкой);*
- оценка предиката внешнего запроса на основе результатов подзапроса;
- описанная выше последовательность повторяется для следующей строки из таблицы внешнего запроса, и так до тех пор, пока все строки не будут проверены.

Соотнесенные подзапросы по своей сути близки к объединениям - обе конструкции включают проверку каждой записи одной таблицы с каждой записью другой или с псевдонимом из той же таблицы, при этом большинство операций у них похожи. Таким образом, применение вложенных запросов в целях использования их результата для управления другим запросом расширяет возможности SQL, позволяя выполнить большее количество функций.

4. Использование операторов EXISTS, ANY, ALL, и SOME.

Существуют следующие специальные операторы, которые всегда берут подзапросы в качестве аргументов - это EXISTS, ANY, ALL, и SOME
Оператор EXISTS используется для указания предикату на то, чтобы производить или не производить вывод в подзапросе, при этом EXISTS дает в качестве результата значение ИСТИНА или ЛОЖЬ. Это в свою очередь означает, что он может работать в предикате или в комбинации с другими булевскими выражениями - AND, OR, и NOT. Другими словами, EXISTS берет подзапрос, как аргумент и оценивает его как истинный, если он осуществляет любой вывод, или как ложный, если он не делает этого. Например, можно решить, извлекать ли данные из таблицы успеваемости, если в ней присутствуют отличные оценки. Это реализуется следующим образом:

```
SELECT * FROM USP
WHERE USP.OCENKA = 5
AND EXISTS (SELECT * FROM USP WHERE USP.OCENKA =
5);
```

При этом внутренний запрос выбирает все данные для всех студентов, получивших оценку 5. Оператор EXISTS во внешнем предикате отмечает, что некоторый вывод в подзапросе имел место, значит, это делает предикат верным. Подзапрос был выполнен только один раз

для всего внешнего запроса, и, следовательно, имеет одно значение во всех случаях - по этой причине EXISTS делает предикат верным или неверным для всех строк сразу. В последнем примере, если строго говорить, EXISTS должен быть установлен так, чтобы выбрать только один столбец. Однако он выбирает все столбцы во вложенном предложении SELECT *. Это не вызывает ошибки, поскольку при выборе EXISTS как одного столбца, так и всех столбцов, он просто замечает выполнение вывода из подзапроса, а полученные значения не использует.
В соотнесенном подзапросе, предложение EXISTS оценивается отдельно для каждой строки таблицы, имя которой указано во внешнем запросе, точно так же, как и другие операторы предиката, когда используется соотнесенный подзапрос. Это дает возможность использовать EXISTS как предикат, который генерирует различные значения для каждой записи таблицы, указанной в основном запросе. Следовательно, информация из внутреннего запроса будет сохраняться.

Существуют еще три специальных оператора, ориентируемых на подзапросы - это ANY. ALL и SOME, напоминающие EXISTS, которые воспринимают подзапросы, как аргументы. Однако эти операторы отличаются от EXISTS тем, что используются совместно с реляционными операторами, аналогично IN в подзапросах.
Операторы SOME и ANY работают одинаково. Различие в терминологии состоит в том, чтобы позволить пользователям употреблять тот термин, который наиболее однозначен. Основное преимущество в использовании ANY и ALL по сравнению с EXISTS заключается в том, что последний требует соотнесенных подзапросов, порой сложных для понимания. Кроме того, подзапросы с ANY или ALL могут выполняться один раз и иметь вывод, используемый для определения предиката для каждой строки основного запроса, а EXISTS требует, чтобы весь подзапрос повторно выполнялся для каждой строки основного запроса.
Оператор ALL работает таким образом, что предикат является верным, если каждое значение, выбранное подзапросом, удовлетворяет условию в предикате внешнего запроса.
Опция ALL гораздо более эффективно используется со знаком неравенства, т.е. с оператором < >. Предикат верен, если данное значение не найдено среди результатов подзапроса.
Рассмотрим такой запрос:

```
SELECT   * FROM USP
WHERE OCENKA < > ALL (SELECT OCENKA
FROM USP
WHERE UDATE = '06/10/1999')
```

Рассматриваемый запрос сделает следующее: подзапрос выберет все

оценки за 10/06/1999 - это будут 5 и 4. После этого, основной запрос выведет все записи с оценкой, не совпадающей ни с одной из них. Аналогичный запрос можно сформулировать с использование оператора NOT IN:

```
SELECT * FROM USP
WHERE OCENKA NOT IN (SELECT OCENKA
FROM USP
WHERE UDATE  = '06/10/1999')
```

Используя оператор ANY:

```
SELECT * FROM USP
WHERE NOT OCENKA = ANY (SELECT OCENKA
FROM USP
WHERE UDATE ='06/10/1999')
```

Вывод этих запросов такой же, как и первого. Важное различие между ALL и ANY - это способ действия в ситуации, когда подзапрос не возвращает никаких значений. Вообще говоря, во всех случаях, когда допустимый подзапрос не делает вывода, ALL автоматически возвращает значение ИСТИНА, а ANY -ЛОЖЬ.
Например, если нет никаких оценок за 10/06/2000, это означает, что запрос:

```
SELECT   *
FROM USP
WHERE OCENKA >= ANY (SELECT OCENKA
FROM USP
WHERE UDATE  = '06/10/2000')
```

не произведет никакого вывода, в то время как запрос

```
SELECT * FROM USP
WHERE OCENKA >= ALL (SELECT OCENKA
FROM USP
WHERE UDATE = '06/10/2000')
```

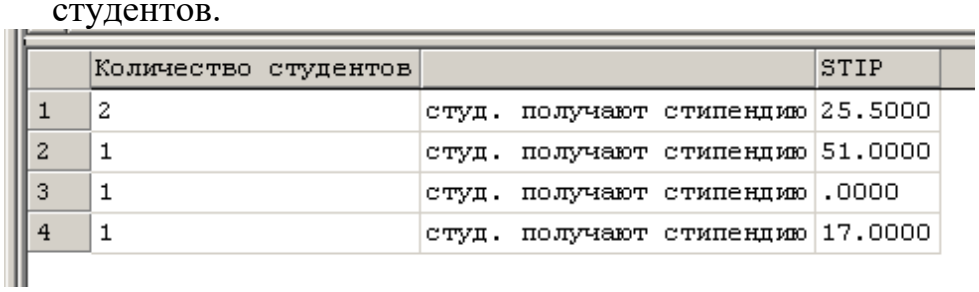
выведет всю таблицу USP. Действительно, если нет никаких оценок за 10/06/2000 то, естественно, ни одно из этих сравнений не имеет значения.

Задания на самостоятельную работу.

Выполнить запросы к БД и составить отчет.

1. Найти преподавателя с фамилией Казанко, Казанков, или Козанко.

2. Составить отчет о количестве студентов, получающих ту или иную стипендию с упорядочиванием по убыванию количества студентов.

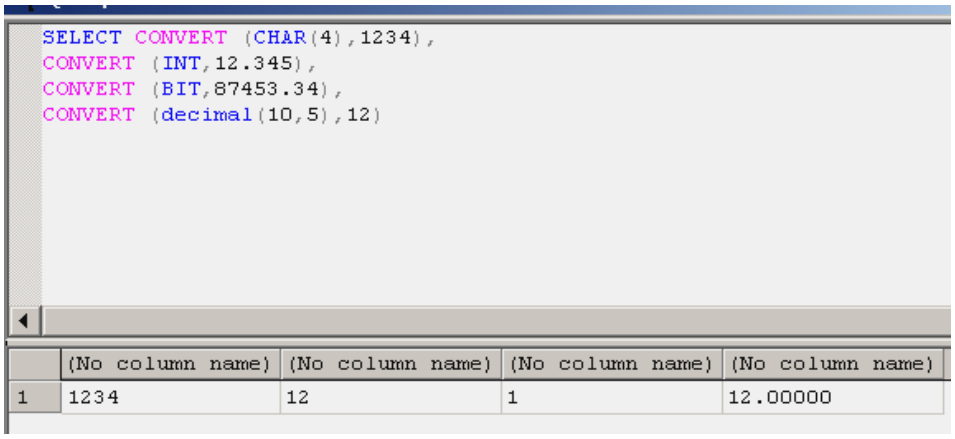


	Количество студентов		STIP
1	2	студ. получают стипендию	25.5000
2	1	студ. получают стипендию	51.0000
3	1	студ. получают стипендию	.0000
4	1	студ. получают стипендию	17.0000

3. С использованием подзапроса найдите данные на всех студентов, которые получали оценки 10/06/1999.

4. Вывести фамилии и номера всех студентов, которые получили более одной оценки. Для реализации задания используйте вложенный подзапрос.

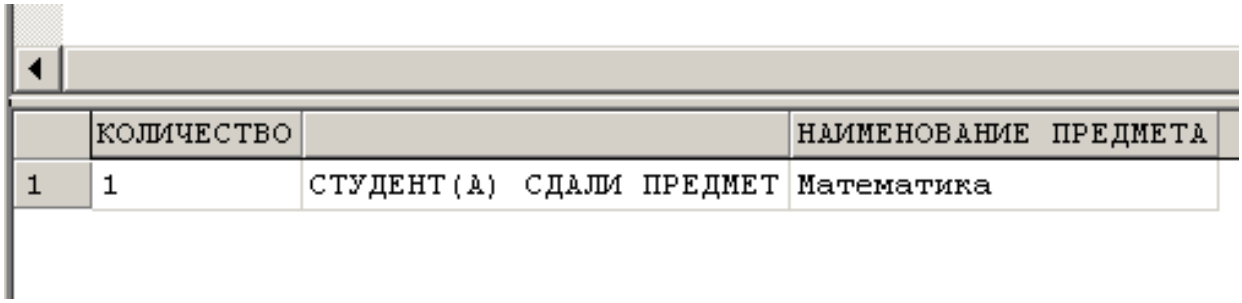
5. Установите среднее значение оценок за дни, для которых выполняется условие, что это среднее равно минимальной оценке в этот день. При вычислении среднего значения, для предотвращения округления до целого, воспользуйтесь функцией явного преобразования типов CONVERT(). Например:



```
SELECT CONVERT (CHAR(4),1234),
CONVERT (INT,12.345),
CONVERT (BIT,87453.34),
CONVERT (decimal(10,5),12)
```

	(No column name)	(No column name)	(No column name)	(No column name)
1	1234	12	1	12.00000

6. Определить количество студентов по дисциплине с PNUM=2003 с оценками выше средней по данному предмету.



	КОЛИЧЕСТВО		НАИМЕНОВАНИЕ ПРЕДМЕТА
1	1	СТУДЕНТ(А) СДАЛИ ПРЕДМЕТ	Математика

7. Вывести информацию о предмете с порядковым номером на 1 меньше, чем у предмета философия.

8. Найти предметы, содержащие знак подчеркивания в данных о названии предмета (предварительно добавить в таблицу PREDMET запись о предмете Ин_язык).