Лабораторная работа № 2

по дисциплине

«Базы данных»

на тему:

«Манипулирование данными с помощью языка Transact-SQL в СУБД Microsoft SQL Server 2008»

Составитель: канд. техн. наук Исмоилов М.И.

Лабораторная работа № 2. Манипулирование данными с помощью языка Transact-SQL в СУБД Microsoft SQL Server 2008.

Цель работы: Изучить синтаксические конструкции языка Transact-SQL, реализованные в СУБД Microsoft SQL Server 2008, применяемые для вставки, обновления, удаления и выборки данных.

Задание на лабораторную работу: Для каждого варианта индивидуального задания с помощью языка Transact-SQL необходимо выполнить действия, описанные в приложении А.

Ход выполнения работы

Использование основных синтаксических конструкций будет продемонстрировано на тестовой базе данных, физическая модель которой представлена на рис. 1.

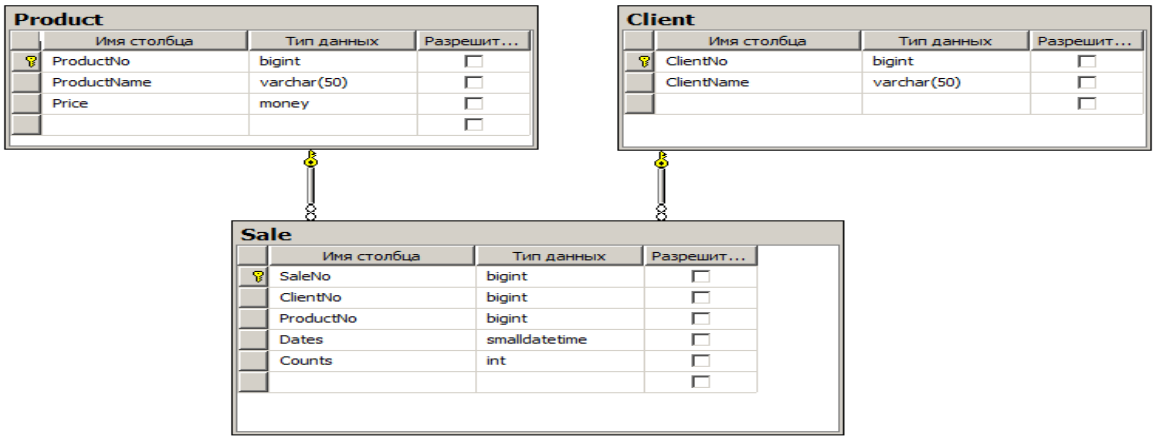

Рис. 1. - Физическая модель тестовой базы данных

Для того, чтобы создать структуру БД и заполнить таблицы начальными данными, необходимо открыть оснастку **SQL Server Management Studio** и подключиться к серверу СУБД. Затем необходимо создать тестовую БД (назовём её ProductSales) и нажать кнопку Создать запрос. В результате появится окно, внешний вид которого представлен на рис 2.

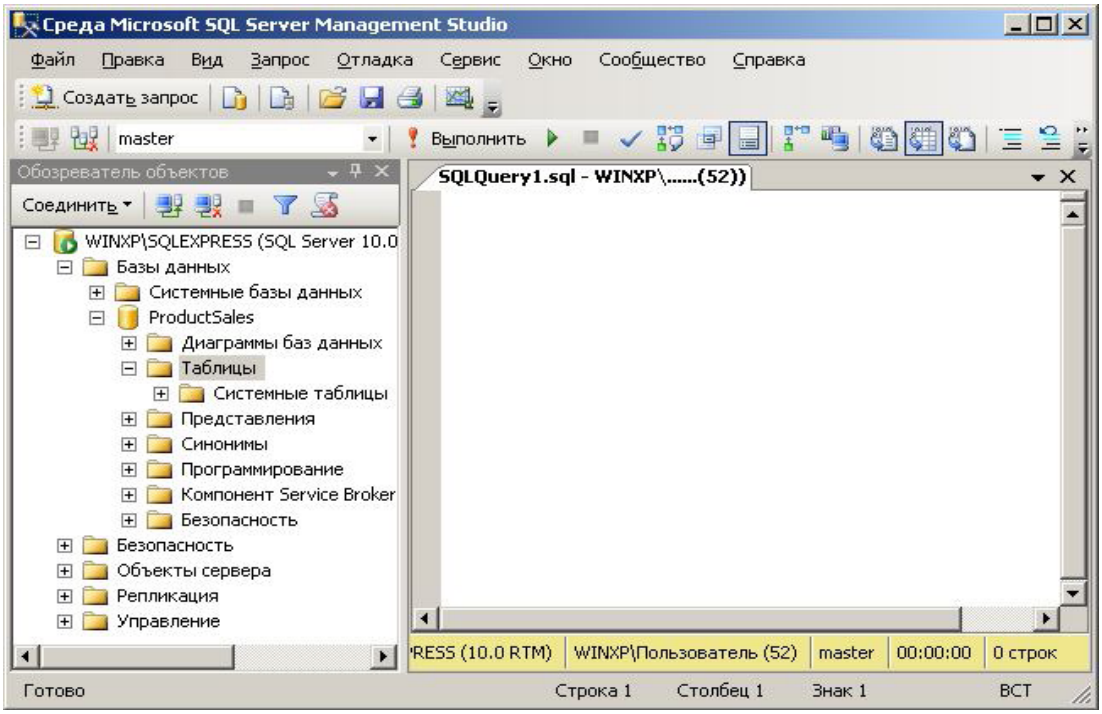

Рис. 2. – Окно ввода нового запроса в оснастке SQL Server Management Studio

В верхней части окна должна быть выбрана новая БД, т.к. именно в ней будут созданы требуемые таблицы. В среднюю часть рассматриваемого окна необходимо ввести программный код, представленный в листинге 1. Для запуска скрипта необходимо нажать кнопку Выполнить.

```
---Создание таблиц в БД
create table Product
(
ProductNo bigint identity(1,1) not null primary key,
ProductName varchar(50) not null,
Price money not null default 0
)
create table Client
(
ClientNo bigint identity(1,1) not null primary key,
ClientName varchar(50) not null
)
create table Sale
(
SaleNo bigint identity(1,1) not null primary key,
ClientNo bigint not null,
ProductNo bigint not null,
Dates smalldatetime not null,
Counts int not null default 0
)
--Создание внешних ключей для таблицы Sale
alter table Sale
add constraint FK_Sale_Client
foreign key(ClientNo)
references Client(ClientNo)
on update cascade
on delete cascade
alter table Sale
add constraint FK_Sale_Product
foreign key(ProductNo)
references Product(ProductNo)
on update cascade
on delete cascade
--Вставка значений в таблицу
  --Без указания имени полей Client
insert Client values('ИП Иванов И.И.')
--С указанием имён полей
insert Client(ClientName)
values('ООО Эдельвейс')
--Вставка нескольких значений
insert Client(ClientName)
select 'ИП Петров П.П.'
union all --Допускается дублирование
select 'ОАО Юг-Креатив'
--Вставка значений в таблицу Product
insert into Product --Несколько значений зез указания
полей
select 'Авторучка', 10
union
select 'Карандаш', 5
union
select 'Линейка', 4
union
select 'Ластик', 8
union
select 'Треугольник', 30
union
select 'Маркер', 20
--Вставка значений в таблицу Sale
insert Sale --Без указания полей
values(1,1,'2010-02-01T08:30:00',100)
--С указанием полей (в порядке объявления)
insert Sale(ClientNo,ProductNo,Dates,Counts)
values(2,1,'2010-02-07T09:15:00',30)
--С указанием полей (в произвольном порядке)
insert Sale(Dates,Counts, ClientNo,ProductNo)
values('2010-02-02T17:13:00',7,1,2)
--Несколько строк с указанием полей (в порядке
объявления)
insert into Sale(ClientNo,ProductNo,Dates,Counts)
select 3,2,'2010-03-03T15:10:00',10
union all
select 2,3,'2010-02-05T10:53:00',5
union all
select 1,3,'2010-03-07T14:07:00',3
union all
select 3,4,'2010-02-15T18:09:00',1
union all --Использование выражение в качестве
значения
select 3,4,'2010-02-16T12:28:00',2*5
union all
select 2,5,'2010-04-07T10:00:00',5
union all
select 1,5,'2010-04-08T09:56:00',2
union all
select 2,5,'2010-04-09T13:15:00',10
```

Листинг 1. – Программный код создания таблиц в БД и занесения в них данных

Несмотря, на то, что программный код хорошо документирован, он требует дополнительного рассмотрения. С помощью оператора create table происходит создание таблицы в текущей базе данных. При этом указываются имена всех полей и присвоенные им типы данных. Директива not null указывает на то, что для поля недопустимо пустое значение. Ключевое слово identity предписывает СУБД самостоятельно генерировать уникальное значение для поля. Директива primary key определяет первичный ключ в отношении. Для создания ограничения внешнего ключа используется директива alter table совместно с ключевыми словами foreign key.

Директива cascade требует каскадного изменения (удаления) значения внешнего ключа при соответствующем изменении (удалении) этого значения, записанного в поле первичного ключа.

Вставка значений в таблицы (см. листинг 1) выполняется с помощью оператора insert. Для добавления новой строки необходимо указать имя таблицы и значения для всех обязательных полей таблицы. Обязательными считаются поля, для которых определена директива not null (для первичных ключей указывается автоматически) и не прописано ни значение по умолчанию (default) ни свойство identity. Вставляемые значения указываются после ключевого слова values. Если порядок и количество соответствует порядку (и количеству) объявления полей в таблице (в директиве create table), то после имени таблицы названия столбцом можно не указывать. В противном случае указание столбцов, в которые заносятся значения – обязательно.

Отметим, что операторы вставки (insert), обновления (update) и удаления (delete) могут выполнять соответствующие операции только для строк одной единственной таблицы.

Результаты занесения значений в таблицы (выполнения скрипта из листинга 1) представлены на рис. 3.

WINXP\SQLEXPR... - dbo.Product | WINXP

	ClientNo	ClientName
▶	1	ИП Иванов И.И.
	2	ООО Эдельвейс
	3	ИП Петров П.П.
	4	ОАО Юг-Креатив
*	NULL	NULL

	ProductNo	ProductName	Price
▶	1	Авторучка	10,0000
	2	Карандаш	5,0000
	3	Ластик	4,0000
	4	Линейка	8,0000
	5	Маркер	30,0000
	6	Треугольник	20,0000
*	NULL	NULL	NULL

WINXP\SQLEXPRE...les - dbo.Sale

	SaleNo	ClientNo	ProductNo	Dates	Counts
▶	1	1	1	2010-02-01 08:30:00	100
	2	2	1	2010-02-07 09:15:00	30
	3	1	2	2010-02-02 17:13:00	7
	4	3	2	2010-03-03 15:10:00	10
	5	2	3	2010-02-05 10:53:00	5
	6	1	3	2010-03-07 14:07:00	3
	7	3	4	2010-02-15 18:09:00	1
	8	3	4	2010-02-16 12:28:00	10
	9	2	5	2010-04-07 10:00:00	5
	10	1	5	2010-04-08 09:56:00	2
	11	2	5	2010-04-09 13:15:00	10
*	NULL	NULL	NULL	NULL	NULL

Рис. 3. – Тестовые значения в БД

Оператор insert имеет следующий синтаксис [1]:

INSERT <имя таблицы> [(<имя поля>.,..)] VALUES

(<список выражений>) | <запрос>;

Перейдём к рассмотрению операторов update и delete, используемых для обновления и удаления значений соответственно. Для этого добавим одного клиента и два заказа для него, что выполняется соответствующим кодом, представленным на рис. 4 (слева).

```
insert   Client
              values('ООО Лотос')
```

ClientNo	ClientName
1	ИП Иванов И.И.
2	ООО Эдельвейс
3	ИП Петров П.П.
4	ОАО Юг-Креатив
5	ООО Лотос

```
insert Sale(ClientNo,ProductNo,Dates,Counts)
  values(5,1,'2010-02-03T17:15:00',30)

insert Sale(ClientNo,ProductNo,Dates,Counts)
  values(5,2,'2010-02-03T17:18:00',11)
```

SaleNo	ClientNo	ProductNo	Dates	Counts
1	1	1	2010-02-01 08:30:00	100
2	2	1	2010-02-07 09:15:00	30
3	1	2	2010-02-02 17:13:00	7
4	3	2	2010-03-03 15:10:00	10
5	2	3	2010-02-05 10:53:00	5
6	1	3	2010-03-07 14:07:00	3
7	3	4	2010-02-15 18:09:00	1
8	3	4	2010-02-16 12:28:00	10
9	2	5	2010-04-07 10:00:00	5
10	1	5	2010-04-08 09:56:00	2
11	2	5	2010-04-09 13:15:00	10
12	5	1	2010-02-03 17:15:00	30
13	5	2	2010-02-03 17:18:00	11

Рис. 4. – Добавление нового клиента и двух заказов

Результатом выполнения запроса явилось добавление в таблицу Client организации с название ООО Лотос и информации о продаже (вставка в таблицу Sale) ей авторучек (30 штук) и карандашей (11 штук) (рис. 4 справа).

Для изменения названия организации ООО Лотос на ОАО Лотос+ необходимо написать запрос, представленный на рис. 5.

```
update Client
  set ClientName='ОАО   Лотос+'
  where ClientNo=5
```

ClientNo	ClientName
1	ИП Иванов И.И.
2	ООО Эдельвейс
3	ИП Петров П.П.
4	ОАО Юг-Креатив
5	ОАО Лотос+

Рис. 5. – Обновление названия организации ООО Лотос на ОАО Лотос+

Обновление значений в строке выполняется с помощью оператора update, в котором указывается название обновляемой таблицы (или её псевдоним). После ключевого слова set указывается перечень полей, в которые заносятся новые значения. При этом обновляются значения во всех строках, удовлетворяющих предикату, записанному в директиве where. В нашем случае обновляется только одна строка, так как поле ClientNo первичный ключ и по определению уникален. Если в приведённом выше запросе убрать фразу where ClientNo=5, то все имеющиеся в БД организации будут названы ОАО Лотос+.

Рассмотрим более сложный запрос (рис. 6). В этом запросе в каждом заказе, сделанном организацией ОАО Лотос+, увеличивается количество купленной продукции. При этом если куплено более 20 единиц, то исходное значение увеличивается в 3 раза, если меньше или равно, то увеличивается в 2 раза.

```
 update s
 set Counts=Counts*case when Counts>20
   then 3
        else 2
        end
from Sale s
join Client c on
c.ClientNo=s.ClientNo
and c.ClientName='ОАО Лотос+'
```

SaleNo	ClientNo	ProductNo	Dates	Counts
1	1	1	2010-02-01 08:30:00	100
2	2	1	2010-02-07 09:15:00	30
3	1	2	2010-02-02 17:13:00	7
4	3	2	2010-03-03 15:10:00	10
5	2	3	2010-02-05 10:53:00	5
6	1	3	2010-03-07 14:07:00	3
7	3	4	2010-02-15 18:09:00	1
8	3	4	2010-02-16 12:28:00	10
9	2	5	2010-04-07 10:00:00	5
10	1	5	2010-04-08 09:56:00	2
11	2	5	2010-04-09 13:15:00	10
12	5	1	2010-02-03 17:15:00	90
13	5	2	2010-02-03 17:18:00	22

Рис. 6. – Сложный запрос обновления заказов, сделанных фирмой ОАО Лотос+

Сравнивая рис. 6 и рис. 4, можно сделать вывод, что заказ SaleNo = 12 увеличился с 30 единиц до 90, т.е. в 3 раза. Другой заказ (SaleNo=13) был увеличен в 2 раза, т.к. первичный размер заказа меньше 20.

Рассмотрим написанный запрос более подробно. После ключевого update указан псевдоним s, присвоенный таблице Sale в операторе from. Псевдонимы используют либо для более краткой формы записи названия таблицы (при многократном обращении к одной и той же таблицы), либо при соединении нескольких таблиц. В операторе set указано то, что требуется обновить поле Counts таблицы Sale, т.к. именно на неё ссылается псевдоним s. Расчёт нового значения выполняется на основе условного оператора case, который в отличие от реализации соответствующего оператора в высокоуровневых языках, в Transact-SQL возвращает значение. Синтаксис этого оператора имеет вид [1]:

```
--Синтаксис простого выражения case
CASE input_expression
    WHEN when_expression THEN result_expression [ ...n ]
    [ ELSE elseresultexpression ] END

--Синтаксис поискового выражения case, позволяющего конструировать сложные
выражения CASE
    WHEN Booleanexpression THEN resultexpression [ ...n ]
    [ ELSE elseresultexpression ]
END
```

Оператор case анализирует список условий и возвращает один из нескольких результатов. Выражение case имеет два формата:

- простое выражение case, используемое для определения результата сравнения выражения с набором простых выражений;
- поисковое выражение case, используемое для определения результата вычисления набора логических выражений.

Оба формата поддерживают дополнительный аргумент else. Выражение case может присутствовать в любой инструкции или предложении, которые разрешают использование допустимых выражений. Например, выражение case можно использовать в таких инструкциях, как select, update, delete и set, а также в таких предложениях, как selectlist, in, where, order by и having.

Рассмотрим синтаксические элементы более подробно:

inputexpression

Выражение, полученное при использовании простого формата выражения case. Аргумент input_expression представляет собой любое допустимое выражение.

WHEN whenexpression

Простое выражение, с которым сравнивается аргумент inputexpression при использовании простого формата функции case. Аргумент when_expression представляет собой любое допустимое выражение. Типы данных аргумента inputexpression и каждого из выражений whenexpression

должны быть одинаковыми или неявно приводимыми друг к другу.

THEN resultexpression

Выражение, возвращаемое, если сравнение выражений inputexpression и whenexpression дает в результате true или выражение booleanexpression вычисляется в true. Аргумент result expression представляет собой любое допустимое выражение.

ELSE elseresultexpression

Это выражение, возвращаемое, если ни одна из операций сравнения не дает в результате true. Если этот аргумент опущен и ни одна из операций сравнения не дает в результате true, функция case возвращает null. Аргумент else_result_expression представляет собой любое допустимое выражение. Типы данных аргумента else_result_expression и любого из аргументов result_expression должны быть одинаковыми или неявно приводимыми друг к другу.

WHEN Booleanexpression

Это логическое выражение, полученное при использовании поискового формата функции case. Аргумент boolean_expression представляет собой любое допустимое логическое выражение.

В рассматриваемом примере (рис. 6) необходимо было обновить поле из таблицы Sale для клиента ОАО Лотос+. Отметим, что в таблице Sale отсутствуют названия организаций (рис. 1) а имеются только её уникальные идентификаторы. Поэтому нам необходимо найти идентификатор клиента ОАО Лотос+ в таблице Client, определить её идентификатор, а затем найти все заказы в таблице Sale. В языке SQL это выполняется с помощью естественных внутренних соединений (оператор join). Т.е. строки, сохранённые в таблице Sale соединяются со строками, хранящимися в таблице Client по равенству значений полей ClientNo. При этом обновляться будут только строки, соответствующие клиенту ОАО Лотос+.

Синтаксис оператора update, выполняющего обновления значений строк имеет вид [1]:

```
[ WITH
<common_table_expression> [...n] ]
UPDATE
   [ TOP ( expression ) [ PERCENT ] ]
   { <object> | rowset functionlimited
    [ WITH ( <Table_Hint_Limited> [ ...n ] ) ]
   }
SET
      { column_name = { expression | DEFAULT |
        NULL } | { udt_column_name.{ {
        property_name = expression | field_name =
        expression } | method_name ( argument [ ,...n ] )
        } }
        | column_name { .WRITE ( expression , @Offset ,
        @Length ) } | @variable = expression | @variable =
        column = expression
      | column_name { += | -= | *= | /= | %= | &= | ^= | |= } expression |
      @variable { += | -= | *= | /= | %= | &= | ^= | |= } expression |
      @variable = column { += | -= | *= | /= | %= | &= | ^= | |= } expression
      } [ ,...n ]

   [ <OUTPUT Clause> ] [
   FROM{ <table_source> } [
   ,...n ] ] [ WHERE {
   <search_condition> | { [
   CURRENT OF
            { { [ GLOBAL ] cursor_name }
     | cursor_variable_name } ]
   } } ]
   [ OPTION ( <query_hint> [
,...n ] ) ] [ ; ]

<object>
```

```
::= {
   [ server_name . database_name . schema_name .
   | database_name .[ schema_name ] .
   | schema_name .
   ]
      table_or_view_name}
```

Рассмотреть синтаксические элементы более подробно можно в любом учебнике.

Для того, чтобы удалить клиента ОАО Лотос+ из БД необходимо выполнить запрос, представленный на рис. 7.

```
delete Client
where ClientName='ОАО Лотос+'
```

Table - dbo.Client* | WINXP.Product...s da

	ClientNo	ClientName
▶	1	ИП Иванов И.И.
	2	ООО Эдельвейс
	3	ИП Петров П.П.
	4	ОАО Юг-Креатив
*	NULL	NULL

Table - dbo.Sale* | WINXP.Product...s database.sql

	SaleNo	ClientNo	ProductNo	Dates	Counts	Price
▶	1	1	1	01.02.2010 8:30:00	100	5,0000
	2	2	1	07.02.2010 9:15:00	30	4,5000
	3	1	2	02.02.2010 17:13:00	7	2,0000
	4	3	2	03.03.2010 15:10:00	10	2,1000
	5	2	3	05.02.2010 10:53:00	5	15,0000
	6	1	3	07.03.2010 14:07:00	3	14,9000
	7	3	4	15.02.2010 18:09:00	1	7,0000
	8	3	4	16.02.2010 12:28:00	10	7,0000
	9	2	5	07.04.2010 10:00:00	5	17,0000
	10	1	5	08.04.2010 9:56:00	2	17,0000
	11	2	5	09.04.2010 13:15:00	10	17,0000
*	NULL	NULL	NULL	NULL	NULL	NULL

Рис. 7. – Удаление информации о фирме ОАО Лотос+ из БД

Для удаления значения из таблицы использована команда delete, вслед за которой указано название таблицы. В операторе where указано условие, которым удовлетворяет удаляемые строки. Как следует из рис. 7 вместе с клиентом были удалены все сделанные им заказы (см. рис. 6.). Это сделано потому, что при объявлении таблицы Sale указан внешний ключ ClientNo с опцией on delete cascade (Листинг 1).

Перейдём к рассмотрению синтаксических конструкций языка Transact-SQL, позволяющих извлекать информацию из БД, а именно, инструкцию select. Для выборки всех клиентов, сохранённых в БД, необходимо написать запрос, представленный на рис. 8.

```
select *
   from Client
```

ClientNo	ClientName
1	ИП Иванов И.И.
2	ООО Эдельвейс
3	ИП Петров П.П.
4	ОАО Юг-Креатив

Рис. 8. – Выборка информации о всех клиентах

После ключевого слова select указывается список полей. В нашем случае указан символ звёздочки (*), означающий выбрать все поля. После слова from указываются таблицы, из которых необходимо извлечь информацию.

Предыдущий запрос извлекал все строки из таблицы. Так как в любой таблице может содержаться огромное количество, строк, то чаще всего на результат выборки накладывают фильтр с помощью директивы where (рис. 9).

```
select *
  from Client
  where ClientNo=1
```

ClientNo	ClientName
1	ИП Иванов И.И.

Рис. 9. – Получение информации о клиенте, у которого в поле ClientNo указано значение 1

В операторе where указывается условие фильтрации, которое может состоять из одного (рис. 9) или нескольких критериев (рис. 10). На рис. 10 представлены два примера написания запроса, извлекающего информацию о тех клиентах, идентификаторы которых равны 1,2 или 5.

```
select *
from Client
where ClientNo in (1,2,5)
```

```
 select *
from Client
where ClientNo=1 or ClientNo=2
   or ClientNo=5
```

ClientNo	ClientName
1	ИП Иванов И.И.
2	ООО Эдельвейс

Рис. 10. – Извлечение информации о нескольких клиентах

Первая версия запроса (рис. 10, слева) использует оператор in, позволяющий проверить поле ClientNo на равенство хотя бы одному из значений, указанных в скобках. Вторая версия использует булевую бинарную операцию or (логическое «или»), позволяющую соединить несколько критериев в одно условие. На рис. 10 логическая операция применена для соединения нескольких критериев для одного поля, а на рис. 11 представлен запрос, в котором сформирован фильтр по двум полям.

```
select *
  from Client where (ClientNo=2) or
(ClientName like 'ИП%')
```

ClientNo	ClientName
1	ИП Иванов И.И.
2	ООО Эдельвейс
3	ИП Петров П.П.

Рис. 11. – Соединение критериев по нескольким полям с помощью операции or

С целью повышения наглядности, соединяемые критерии размещены в скобках, хотя в данном случае в (Transact-SQL) это необязательно. В запросе просматривается каждая строка таблицы Client и выводится в том случае, если либо идентификатор клиента равен 2, ли если значение поля ClientName начинается со строки «ИП», за которой следует неограниченное количество символов (данная маска задана с помощью символа %). Для проверки строкового поля на неточное соответствие значению используется ключевое слово like (с помощью которого описывается предикат), которое определяет совпадает ли указанная символьная строка с заданным шаблоном [1]. Шаблон может включать обычные символы и символы-шаблоны. Во время сравнения с шаблоном необходимо, чтобы его обычные символы в точности совпадали с символами, указанными в строке. Символы-шаблоны могут совпадать с произвольными элементами символьной строки. Использование символов-шаблонов с оператором like предоставляет больше возможностей, чем использование операторов сравнения строк = и <>.

Табл. 1. – Типы шаблонов Microsoft SQL Server 2008

Символ-шаблон	**Описание**	**Пример использования**
%	Любая строка длиной от нуля и более символов	Оператор where Название like '%компьютер%' выполняет поиск и выдает все названия, содержащие слово «компьютер»
_ (подчеркивание)	Любой одиночный символ	Оператор where фамилия_автора LIKE '_етров' выполняет поиск и выдает все имена, состоящие из шести букв и заканчивающиеся сочетанием «етров» (Петров, Ветров и
[ ]	Любой одиночный символ, содержащийся в диапазоне ([a-f]), т.е. в наборе ([abcdef])	Оператор where Фамилия_автора LIKE '[Л-С]омов' выполняет поиск и выдает все фамилии авторов, заканчивающиеся на «омов» и начинающиеся на любую букву в промежутке от «Л» до «С», например Ломов, Ромов, Сомов и т.п. При выполнении операции поиска в диапазоне символы, включенные в диапазон, могут изменяться в зависимости от правил сортировки
[^]	Любой символ, отсутствующий в диапазоне ([^a-f]) или в наборе ([^abcdef]).	Оператор where Фамилия_автора LIKE 'ив[^а]%' выполняет поиск и выдает все фамилии, начинающиеся на «ив», в которых третья буква отличается от «а»

В предыдущих запросах на выборку данных извлекались все поля таблицы и порядок следования строк был не определён. В следующем запросе (рис. 12) извлекаются названия всех организаций и результат сортируется в порядке возрастания.

```
select ClientName
  from Client
  order by ClientName
```

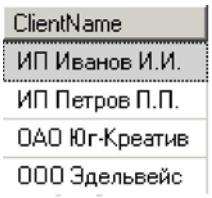

Рис. 12. – Сортировка организаций по названию

Для сортировки результирующего набора используется фраза order by, в которой указано поле сортировки. По умолчанию сортировка выполняется в порядке возрастания. Если требуется в порядке убывания, то необходимо после имени поля указать слово desc.

До текущего момента все запросы извлекали данные из одной таблицы, что осуществлялось указанием имени единственной таблицы в директиве from. На рис. 13 представлены две формы запросов, извлекающих информацию из всех трёх таблиц, изображённых на рис. 1. В запросе извлекается информация о продажах с указанием названия покупателя и продукта. Первая форма запроса (рис. 13) использует директиву where для соединения трёх таблиц (Client, Sale, Product), указанных после слова from, по полям первичных/внешних ключей. Т.к. в полях внешних ключей могут сохраняться значения, равные значениям первичных ключей других таблиц, то для организации соединений используется операция равенства (=), по обе стороны которой указаны имена соединяемых полей. Результирующий набор состоит из названия организации (c.ClientName), названия продукта (p.ProductName), все поля из таблицы Sale (s.*), а также вычисляемое поле, рассчитывающие сумму заказа (цена*количество) которое появляется в результирующем наборе под псевдонимом Summa (слово as является необязательным). Сортировка выполняется по клиенту, продукту и в порядке убывания (desc) даты продажи.

Вторая форма запроса (рис. 13) организует соединение с помощью директивы join. По умолчанию предполагается выполнение внутреннего соединения, поэтому фраза inner не обязательна. После слова on указываются поля (первичных/внешних ключей), по которым выполняется соединение. Использование фразы join вместо where более естественно описывает процедуру соединения таблиц.

```
select c.ClientName, p.ProductName, s.*,p.Price, p.Price*s.Counts as Summa
from Client c, Sale s, Product p
where c.ClientNo=s.ClientNo
and s.ProductNo=p.ProductNo
order by c.ClientName, p.ProductName, s.Dates desc.


select c.ClientName, p.ProductName, s.*, p.Price,
p.Price*s.Counts as Summa
from Client c
inner join Sale s on s.ClientNo=c.ClientNo join Product p on p.ProductNo=s.ProductNo
order by c.ClientName, p.ProductName, s.Dates desc
```

Результаты | Сообщения

	ClientName	ProductName	SaleNo	ClientNo	ProductNo	Dates	Counts	Price	Summa
1	ИП Иванов И.И.	Авторучка	1	1	1	2010-02-01 08:30:00	100	10,00	1000,00
2	ИП Иванов И.И.	Карандаш	3	1	2	2010-02-02 17:13:00	7	5,00	35,00
3	ИП Иванов И.И.	Ластик	6	1	3	2010-03-07 14:07:00	3	8,00	24,00
4	ИП Иванов И.И.	Маркер	10	1	5	2010-04-08 09:56:00	2	20,00	40,00
5	ИП Петров П.П.	Карандаш	4	3	2	2010-03-03 15:10:00	10	5,00	50,00
6	ИП Петров П.П.	Линейка	8	3	4	2010-02-16 12:28:00	10	4,00	40,00
7	ИП Петров П.П.	Линейка	7	3	4	2010-02-15 18:09:00	1	4,00	4,00
8	ООО Эдельвейс	Авторучка	2	2	1	2010-02-07 09:15:00	30	10,00	300,00
9	ООО Эдельвейс	Ластик	5	2	3	2010-02-05 10:53:00	5	8,00	40,00
10	ООО Эдельвейс	Маркер	11	2	5	2010-04-09 13:15:00	10	20,00	200,00
11	ООО Эдельвейс	Маркер	9	2	5	2010-04-07 10:00:00	5	20,00	100,00

Рис. 13. – Извлечение подробной информации о продажах

Отметим, что описанные формы запросов позволяют организовать внутреннее соединение, предполагающее вывод информации только для тех клиентов, которые сделали заказ и только о тех продуктах, которые хотя бы один раз были проданы. Если посмотреть на рис. 3, что видно, что на рис. 13 отсутствует организация ОАО Юг-Креатив и продукт Треугольник, т.к. они не участвовали в продажах.

Часто возникают задачи, при которых необходимо вывести информацию о клиентах и продажах, даже в том случае, когда клиент не совершал покупок. Так в следующем запросе (рис. 14) выводится информация о всех клиентах и датах сделанных ими покупок.

```
select c.ClientName, s.Dates, s.Counts
from Client c
left join Sale s on s.ClientNo=c.ClientNo
```

ClientName	Dates	Counts
ИП Иванов И.И.	2010-02-01 08:30:00	100
ИП Иванов И.И.	2010-02-02 17:13:00	7
ИП Иванов И.И.	2010-03-07 14:07:00	3
ИП Иванов И.И.	2010-04-08 09:56:00	2
ООО Эдельвейс	2010-02-07 09:15:00	30
ООО Эдельвейс	2010-02-05 10:53:00	5
ООО Эдельвейс	2010-04-07 10:00:00	5
ООО Эдельвейс	2010-04-09 13:15:00	10
ИП Петров П.П.	2010-03-03 15:10:00	10
ИП Петров П.П.	2010-02-15 18:09:00	1
ИП Петров П.П.	2010-02-16 12:28:00	10
ОАО Юг-Креатив	NULL	NULL

Рис. 14. – Выборка информации о всех клиентах и датах продаж

Для выполнения поставленной задачи использовалось левое внешнее соединение, реализуемое с помощью оператора left outer join (слово outer необязательное). Такое соединение подразумевает выборку всех строк, расположенных слева от оператора left join (в нашем случае из таблицы Client) и выборку только тех строк справа, которые соответствуют строкам слева (в нашем случае из таблицы Sale). Из рис. 3 видно, что компания ОАО Юг-Креатив не совершала покупок, поэтому в результирующей выборке (рис. 14) заполнены только поля, соответствующие таблицы Client (т.е. ClientName), все остальные поля, соответствующие другим таблицам (в нашем случае поля Dates и Counts таблицы Sale) будут иметь null-значения. Аналогичным образом для реализации поставленных задач можно использовать правое внешнее соединение (right join).

Модифицирована версия запроса, представленная на рис. 15, позволяет вывести информацию только о тех клиентах, которые ничего не приобретали.

```
select c.ClientName
  from Client c
  left join Sale s on
s.ClientNo=c.ClientNo
  where s.Dates is null
```

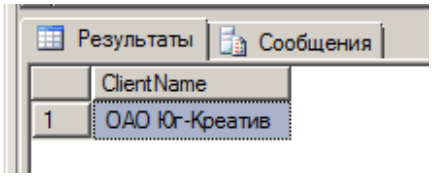

Рис. 15. – Выборка информации о клиентах, не сделавших ни единого заказа

Так как все поля заказа содержат null-значение, то для решения задачи достаточно проверить любое поле из таблицы с помощью операции is null. Т.к. null имеет особый смысл в теории БД, то для проверки наличия этого значения в поле таблицы используется отдельная операция. При этом нельзя использовать стандартную операция сравнения на равенство (=) в данном случае, т.к. null-значение не равно никакому другому, в том числе и null.

Все рассмотренные до текущего момента запросы выборки данных позволяли просмотреть детальную информацию, сформированную на основе строк, хранящихся в таблице. Одна часто возникает задача посчитать агрегированные (статистические) значения (сумму, среднее значение, минимальное значение и др) для всей выборки. На рис. 16 представлен пример запроса, который выводит количество покупок, количество проданного товара и сумму всех продаж, сделанных начиная с 01.03.2010 9:00.

```
select count(*)as 'Количество покупок', sum(s.Counts) as 'Количество товара' ,
sum(p.Price*s.Counts) as Summa
from Sale s
      inner join Product p on s.ProductNo= p.ProductNo
where s.Dates>='2010-03-01T09:00:00'
```

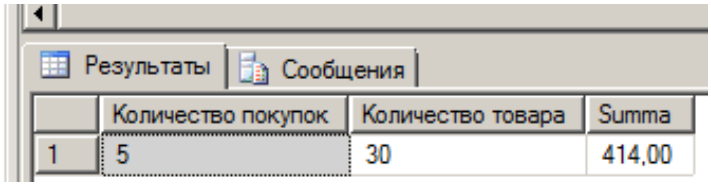

Рис. 16. – Расчет агрегированных значений

Отметим, что если выражению не присвоить псевдоним (для первого поля на рис. 16 слева), то в результирующем наборе оно не будет иметь имени. Представленный запрос имеет следующие особенности: в операторе select присутствуют только агрегирующие функции; условие, записанное в where применяется к каждой строке.

В таблице 2 представлено подробное описание всех агрегирующих функций, имеющихся в СУБД Microsoft SQL Server 2008 [1].

Табл. 1. – Агрегирующие (статистические) функции СУБД Microsoft SQL Server 2008

Функция/Описание	**Аргументы функции**
AVG([ALL|DISTINCT] expression) Возвращает среднее арифметическое группы значений. Значения NULL не учитываются.	***ALL*** Применяет статистическую функцию ко всем значениям. По умолчанию задается параметр all. ***DISTINCT***
CHECKSUM_AGG ([ALL|DISTINCT] expression) Возвращает контрольную сумму значений в группе. Значения NULL не учитываются.	
COUNT({[[ALL|DISTINCT] expression] | * }) Возвращает количество элементов в группе в виде результата типа int.	Аргументы ***ALL***, ***DISTINCT*** и **expression** используются также как и для предыдущих функций. ******* Указывается, что при возврате общего числа строк в таблице
COUNT_BIG({[[ALL|DISTINCT] expression] | * }) Возвращает количество элементов в группе в виде результата типа bigint.	
MAX([ALL|DISTINCT] expression) Возвращает максимальное значение в выражении.	Аргументы ***ALL***, ***DISTINCT*** и **expression** используются также как и для предыдущих функций. Однако параметр distinct не имеет смысла при использовании функции и доступен только для
MIN([ALL|DISTINCT] expression) Возвращает минимальное значение в выражении.	
SUM([ALL|DISTINCT] expression) Возвращает сумму значений в выражении.	Аргументы ***ALL***, ***DISTINCT*** и **expression** используются также как и для предыдущих функций.
STDEV([ALL|DISTINCT] expression) Возвращает статистическое стандартное отклонение значений в выражении.	
STDEVP([ALL|DISTINCT] expression) Возвращает статистическое среднеквадратичное отклонение совокупности значений в выражении.	
VAR([ALL|DISTINCT] expression) Возвращает статистическую дисперсию значений в выражении	
VARP([ALL|DISTINCT] expression) Возвращает статистическую дисперсию для заполнения значений в выражении.	

Часто статистические функции используются для расчёта значений не всей выборки в целом, а отдельных групп, сформированных по некоторым критериям. Так в запросе из рис. 17 извлекаются все продажи, в которых куплено более 10 единиц товара. Затем результирующий набор группируется по клиентам и для каждого клиента выводится его название, количество продаж и общая сумма всех покупок. Выборка сортируется по названию организации.

```
select c.ClientName, count(*) as Counts ,sum(p.Price*s.Counts) as Summa
from Product p join  Sale s on p.ProductNo=s.ProductNo
      join Client c on c.ClientNo=s.ClientNo where s.Counts>10
group by c.ClientName
order by c.ClientName
```

Результаты | Сообщения

	ClientName	Counts	Summa
1	ИП Иванов И.И.	1	1000,00
2	ООО Эдельвейс	1	300,00

Рис. 17. – Вывод агрегированных значений для покупателей

При группировке данных, выполняемой оператором group by в операторе select, могут присутствовать только столбцы, по которым выполняется группировка (c.ClientName) и агрегирующие функции, а также их комбинация.

Условие оператора where налагает ограничения на каждую строку выборки. Для наложения ограничений на значения, полученные после расчета статистических функций, используется фраза having. На рис. 18 для тех клиентов, которые совершили более трёх покупок на общую сумму более 100 рублей выводится количество продаж и общая сумма.

```
select c.ClientName, count(*) as Counts ,sum(p.Price*s.Counts) as Summa
    from Product p join  Sale s on p.ProductNo=s.ProductNo
            join Client c on c.ClientNo=s.ClientNo
    group by c.ClientName
    having sum(p.Price*s.Counts)>100 and count(*)>3
    order by c.ClientName
```

Результаты | Сообщения

	ClientName	Counts	Summa
1	ИП Иванов И.И.	4	1099,00
2	ООО Эдельвейс	4	640,00

Рис. 18. – Определение условий фильтрации для агрегированных значений

Выводы

В ходе выполнения лабораторной работы были изучены основные синтаксические конструкции, применяемые для манипулирования данными тестовой БД, реализованные в языке Transact-SQL.

Литература

1. Электронная документация по Microsoft SQL Server 2008 (Июль 2009г.), http://download.microsoft.com/download/A/D/3/AD3A804B-94A4-4F4E-A5B5-8E44E5937679/SQLServer2008_BOL_Jul2009_RUS.msi