Государственное бюджетное образовательное учреждение высшего образования
Московской области
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

Н.П. СИДОРОВА
Г.Н. ИСАЕВА

ОСНОВЫ T-SQL

Практикум по курсу «Базы данных»

Учебное пособие

Королев
2017

УДК 621.398

С 374

Сидорова Н.П**.,** Исаева Г.Н**.** Основы T-SQL.Практикум по курсу «Базы данных»: учебное пособие. – Королев МО: МГТОУ, 2017**.** – 60 с.

Рецензент: д.т.н., проф. Стреналюк Ю.В.

канд**.** техн**.** наук, доцент Зимин В.М.

Учебное пособие включает 8 практических работ, отражающих основные возможности работы с базами данных в среде SQLServer.Описание каждой практической работы содержит необходимые сведения о синтаксисе языка T-SQL и особенностях использования операторов языка T-SQL для решения задач создания объектов базы данных и обработки данных, программирования хранимых процедур и триггеров, а также решения задач администрирования баз данных. При описании синтаксических конструкций языка в пособии используется нотация Бэкуса-Наура.

Пособие составлено в соответствии с требованиями федерального государственного образовательного стандарта высшего профессионального образования (ФГОС) 09.03.03 «Прикладная информатика» профиль: Прикладная информатика в системах управления и будет полезно студентам других направлений подготовки при изучении особенностей использования T-SQL для работы с базами данных.

Содержание

ВВЕДЕНИЕ

Разработка баз данных (БД) с использованием серверных систем управления базами данных (СУБД), таких как Oracle, Microsoft SQL Server, MySql, предполагает использование мощного и простого языка, который позволяет выполнять все необходимые операции с БД её пользователям и администраторам. Таким языком для работы с реляционными БД стал язык SQL (Structured Query Language), достоинствам которого является наличие международных стандартов, широкая поддержка его реализации в различных средах программирования приложений с БД.

В настоящее время действует стандарт языка, принятый в 2003 году (SQL:2003) с небольшими модификациями, внесёнными позже в 2008 году. Этот стандарт поддерживается практически всеми разработчиками современных коммерческих реляционных СУБД, таких как Oracle, Microsoft SQL Server, MySql, Sybase и др. SQL с самого начала разрабатывался как полнофункциональный язык для работы с БД, т.е. он поддерживает выполнение любых действий по работе с БД. Он является основным.

В структуре SQL можно выделить следующие компоненты.

- ***SQL-DDL*** (Data Definition Language) – язык определения объектов БД и ограничений целостности. К ним относятся команды создания и удаления баз данных; создания, изменения и удаления таблиц; управления пользователями и т.д.
- ***SQL-DML*** (Data Manipulation Language) – язык манипулирования данными: добавление, изменение, удаление и извлечение данных, управления транзакциями.
- Операторы программирования.
- Средства администрирования.

В настоящее время существует большое количество реализаций языка, которые имеют специфические отличия, подчиняясь, тем не менее, единому стандарту. При выполнении практических работ используется реализация SQL в Microsoft SQL Server – Transact-SQL(T-SQL). Для этого используется компонент Database Engine – основная служба для хранения, обработки и защиты данных Microsoft SQL Server. Компонент Database Engine обеспечивает управляемый доступ и быструю обработку транзакций, достаточную для большинства приложений по обработке данных в организации. Он

используется для создания реляционных БД, для оперативной обработки транзакций или оперативной аналитической обработки данных.

Практикум включает 8 практических работ, которые предусматривают работу с ***SQL-DDL*** для создания объектов БД: таблиц, индексов, хранимых процедур и триггеров; ***SQL-DML*** для разработки запросов на обработку данных, средства среды ***SQL Server Management Studio*** для решения задач администрирования БД.

Язык T-SQL совместим с общей средой выполнения (Common Language Runtime – CLR) операционной системы Windows. Начиная с выпуска 2012 года, T-SQL стал одним из языков платформы NET.

При подготовке учебного пособия использовалась версия программного продукта SQL Server 2008 Express. Однако рассмотренные в практикуме возможности языка являются инвариантными и могут быть применены при работе с другими версиями и выпусками этой СУБД, поскольку основные фундаментальные операторы обработки данных являются практически одинаковыми.

При описании синтаксических конструкций языка в пособии используется нотация Бэкуса-Наура. Предполагается, что студенты знакомы с этим способом задания синтаксиса языков программирования.

Учебное пособие предназначено для студентов, обучающихся по направлению «Прикладная информатика» и может быть полезно студентам других направлений подготовки при изучении дисциплин, связанных с использованием Microsoft SQL Server для создания и использования БД.

Практическая работа 1. СОЗДАНИЕ СХЕМЫ БД

Цель работы – приобрести навыки использования оператора CreateTable и Insert.

1. Основные понятия

Таблица – основной объект для хранения информации в реляционной БД. Для создания описания таблицы БД используется оператор Create Table. В нём определяется состав столбцов таблицы, задается их описание и определяются правила целостности на уровне столбца и таблицы. Обязательными описателями являются имя таблицы, имя столбца, тип данных столбца. Синтаксис оператора приведён ниже.

CREATE TABLE *имя_таблицы*
(
{имя_столбцатип_данных ***[NOTNULL] [[PRIMARYKEY***|
UNIQUE]
[DEFAULT <*значение*>***]***
[IDENTITY [(начальное_значение***)]]***
[FOREIGN KEY
REFERENCES *имя_род_таблицы*
[(имя_столбца_род_таблицы ***)]***
[CHECK (<*условие* > ***)] [,...n]***
[ON UPDATE {CASCADE | NO ACTION }]
[ON DELETE {CASCADE | NO ACTION }]
}
);

Типы данных, хранящихся в столбце, определяются на основе разрешённых типов в данной реализации СУБД. Типы данных в SQL Server объединены в следующие категории:

- точные числа,
- символьные строки в Юникоде,
- приблизительные числа,
- двоичные данные,
- дата и время,
- символьные строки,
- дополнительные типы данных.

В зависимости от параметров хранения, некоторые типы данных в SQL Server относятся к следующим группам:

- типы данных больших значений: varchar(max), nvarchar(max) и varbinary(max);
- типы данных больших объектов: text, ntext, image, varchar(max), nvarchar(max), varbinary(max) и xml.

Описание наиболее употребляемых типов данных T-SQL приведены в таблице 1, характеристики числовых типов представлены в таблице 2. Полное описание типов данных можно найти на сайте разработчика ***technet.microsoft.com/ru-ru/library.***

Таблица 1– Описание наиболее употребляемых типов данных T-SQL

Тип	Описание
binary [(n)]	Хранит двоичные данные фиксированной длины размером в n байт, где n – значение от 1 до 8000. Размер хранения составляет n байт
varbinary [(n | max)]	Хранит двоичные данные переменной длины n, n может иметь значение от 1 до 8000; max означает максимальную длину хранения, которая составляет 2^31-1 байт. Размер хранения – это фактическая длина введённых данных плюс 2 байта. Он представляет собой простые битовые потоки
bit	Целочисленный тип данных, который может принимать значения 1, 0 или NULL
char [(n)]	Строковые данные фиксированной длины не в Юникоде. Аргумент n определяет длину строки, он должен быть значением от 1 до 8000. Размер при хранении составляет n байт
cursor	Тип данных для переменных или выходных параметров хранимых процедур, которые содержат ссылку на курсор
date	Описывает дату
datetime	Определяет дату, включающую время дня с долями секунды в 24-часовом формате
decimal[(p[,s])]	Хранит числовые данные с фиксированными точностью и масштабом. При использовании максимальной точности числа могут принимать значения в диапазоне от -10^38+1 до 10^38-1. p (точность) определяет максимальное количество десятичных разрядов числа (как слева, так и справа от десятичной запятой). Точность должна принимать значение от 1 до 38. По умолчанию для точности принимается значение 18. s (масштаб) – максимальное количество десятичных разрядов числа справа от десятичной запятой. Масштаб может принимать значение от 0 до p. Масштаб может быть указан только совместно с точностью. По умолчанию масштаб принимает значение 0; поэтому 0 <= s<= p
float [(n)]	Используется для числовых данных с плавающей запятой. n- задает точность

Продолжение табл.1

Тип	Описание
int, bigint, smallint и***tinyint***	Определяет точные числовые данные, использующие целые значения
money	Определяет данные, представляющие денежные (валютные) значения
nchar [(n)]	Строковые данные фиксированной длины в формате Юникод. Аргумент n определяет длину строки, он должен быть значением от 1 до 4000. Размер при хранении в два раза больше n байт
ntext	Этот тип данных представляет данные в формате Юникод переменной длины с максимальной длиной строки 2^30 – 1 (1 073 741 823) символов. Объём занимаемого этим типом пространства (в байтах) в два раза превышает введённую длину строки
numeric[(p[,s])]	Аналогично типу decimal
nvarchar [(n | max)]	Строковые данные переменной длины в формате Юникод. Аргумент n определяет длину строки, он может быть значением в диапазоне от 1 до 4000, max указывает, что максимальный размер для хранения равен 2^31-1 байт (2 ГБ)
real	Определяет числовые данные с плавающей запятой
smalldatetime	Определяет дату, сочетающуюся с временем дня. Время представлено в 24-часовом формате с секундами, всегда равными нулю (:00), без долей секунд
text	Этот тип данных представляет данные, отличные от данных Юникода с переменной длиной строки, представленные с использованием кодовой страницы сервера. Максимальная длина строки – 2^31 – 1 (2 147 483 647) символов
time	Определяет время дня без учёта часового пояса в 24-часовом формате
varbinary [(n | max)]	Двоичные данные переменной длины. n может иметь значение от 1 до 8000; max означает максимальную длину хранения, которая составляет 2^31-1 байт. Размер хранения – это фактическая длина введённых данных плюс 2 байта
varchar [(n | max)]	Строковые данные переменной длины не в формате Юникод. Аргумент n определяет длину строки, он может быть значением в диапазоне от 1 до 8000, max указывает, что максимальный размер для хранения равен 2^31-1 байт (2 ГБ). Размер хранения (в байтах) – это значение фактической длины введенных данных плюс 2 байта
xml (Transact-SQL)	Тип данных, в котором хранятся XML-данные. Можно хранить экземпляры xml в столбце либо в переменной типа xml.

Таблица 2 – Характеристики числовых типов данных T-SQL

Тип данных	Диапазон	Объём памяти
bigint	от -2^63 (-9 223 372 036 854 775 808) до 2^63-1 (9 223 372 036 854 775 807)	8 байт
int	от -2^31 (-2 147 483 648) до 2^31-1 (2 147 483 647)	4 байта
smallint	от -2^15 (-32 768) до 2^15-1 (32 767)	2 байта
tinyint	от 0 до 255	1 байт
float	- 1,79E+308 — -2,23E-308, 0 и 2,23E-308 — 1,79E+308	Зависит от значения n: при n= **1÷24 -** 4 байта при n= **25÷53 – 8 байтов**
real	- 3,40E + 38 — -1,18E - 38, 0 и 1,18E - 38 — 3,40E + 38	4 байта
money	От -922 337 203 685 477,5808 до 922 337 203 685 477,5807	8 байт
smallmoney	От -214 748,3648 до 214 748,3647	4 байта

Описание типов данных даты и времени T-SQL определены в таблице 3.

Таблица 3 – Описание типов данных даты и времени T-SQL

Тип данных	Формат	Диапазон	Точность	Объём памяти (в байтах)	Определяемая пользователем точность в долях секунды
time	чч:мм:сс[.нннннн]	от 00:00:00.0000000 до 23:59:59.9999999	100 наносекунд	от 3 до 5	да
date	ГГГГ-ММ-ДД	от 0001-01-01 до 9999-12-31	1 день	3	нет
smalldatetime	ГГГГ-ММ-ДД чч:мм:сс	от 01.01.1900 до 06.06.2079	1 минута	4	нет
datetime	ГГГГ-ММ-ДД чч:мм:сс[.ннн]	от 01.01.1753 до 9999-12-31	0,00333 секунда	8	нет

Важно! При работе с конкретной версией программного продукта целесообразно уточнить на сайте компании разработчика technet.microsoft.com поддерживаемые в используемой версии типы данных и их характеристики.

Для изменения структуры таблицы и описания её полей используется оператор ALTER TABLE, который позволяет добавлять

и удалять столбцы, создавать и уничтожать индексы, переименовывать столбцы таблицы:

ALTER TABLE *имя_таблицы*
{
[ALTER COLUMN *имя_столбца {новый_тип_данных* ***[NULL | NOT NULL]}]***
|
ADD { [*имя_столбца тип_данных]* *|* *имя_столбца* ***AS*** *выражение* ***} [,...n]***
|
DROP {COLUMN *имя_столбца****}[,...n]***
};

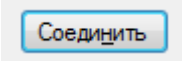
Для удаления описания таблицы используется оператор
DROP TABLE *имя_таблицы* ***[RESTRICT | CASCADE***]

2. Средства Management Studio для работы с БД Microsoft SQL Server

Используемый при выполнении практических работ выпуск SQL Server 2008 Compact Edition является бесплатной версией этого программного продукта. Он широко используется для обучения работой с Microsoft SQL Server. При необходимости использования дополнительных функций, его можно легко обновить до версий SQL Server более высокого класса.

Компонент Compact Edition является одним из сервисов SQL Server и предназначен для создания моделей базы данных, хранения, обработки и обеспечения безопасности данных. Он обеспечивает управляемый доступ к ресурсам и быструю обработку транзакций, что позволяет использовать его даже с самыми требовательными приложениями по обработке данных на предприятии.

Основу компонента Compact Edition составляет среда SQL Server Management Studio. Она является интегрированной средой для доступа, настройки, управления, администрирования и разработки всех компонентов SQL Server. Среда Management Studio позволяет разработчикам и администраторам, обладающим различными уровнями навыков, использовать SQL Server для решения практических задач.

Для работы с Compact Edition необходимо выбрать тип сервера, выбрать установленный сервер и выполнить соединения с ним, нажав кнопку 

(рис. 1).

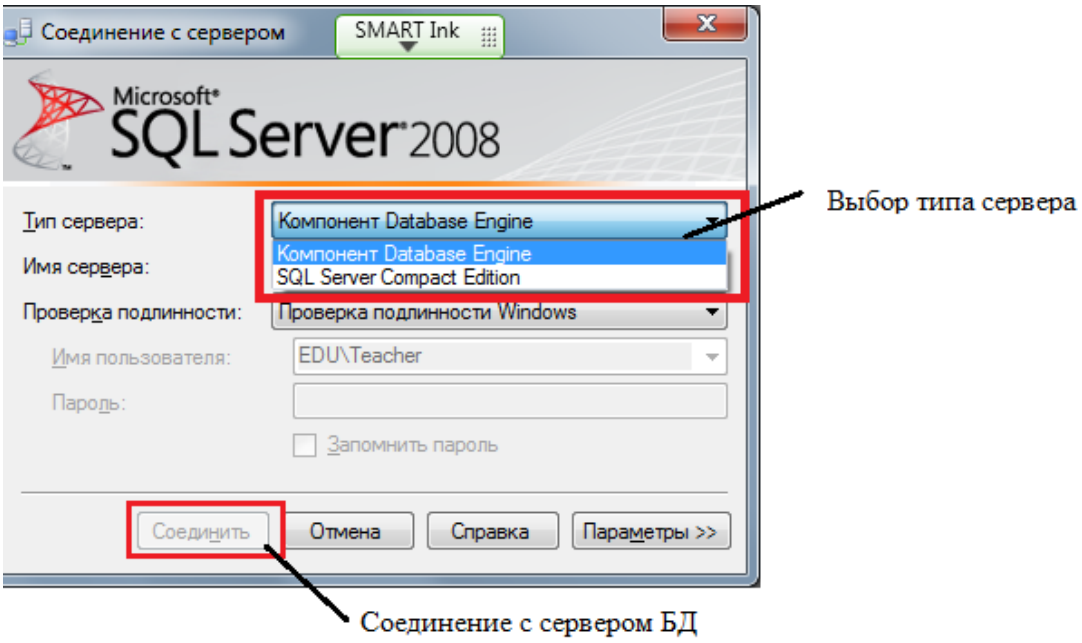


Рисунок 1 – Соединение с сревером БД

При соединении с БД открывается основное меню среды Microsoft SQL Server Management Studio (рис. 2), в котором можно создавать запросы.

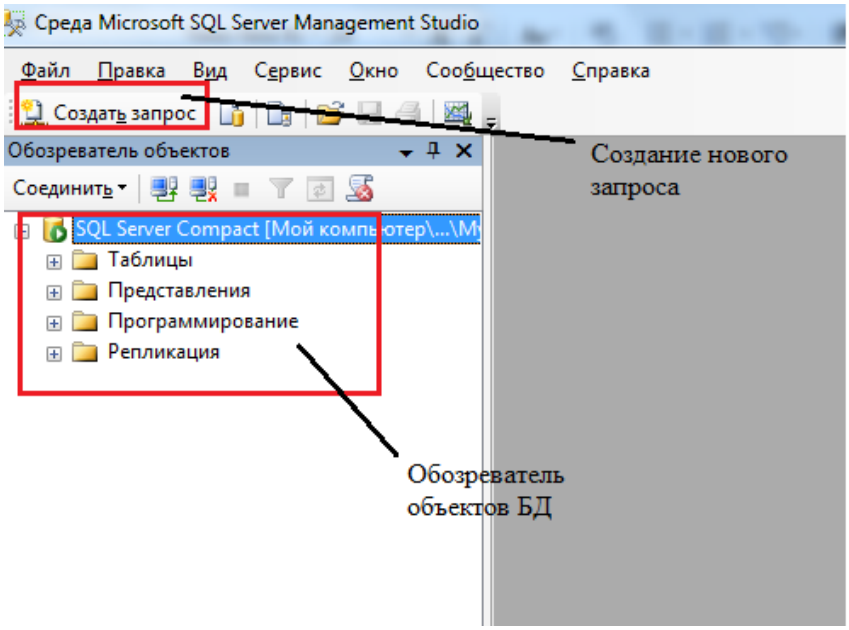
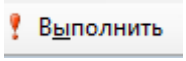


Рисунок 2 – Основное меню среды Microsoft SQL Server Management Studio

Для того чтобы выполнить запрос на работу с БД, необходимо описать команду на языке SQL в поле запроса и нажать кнопку 

(рис. 3)

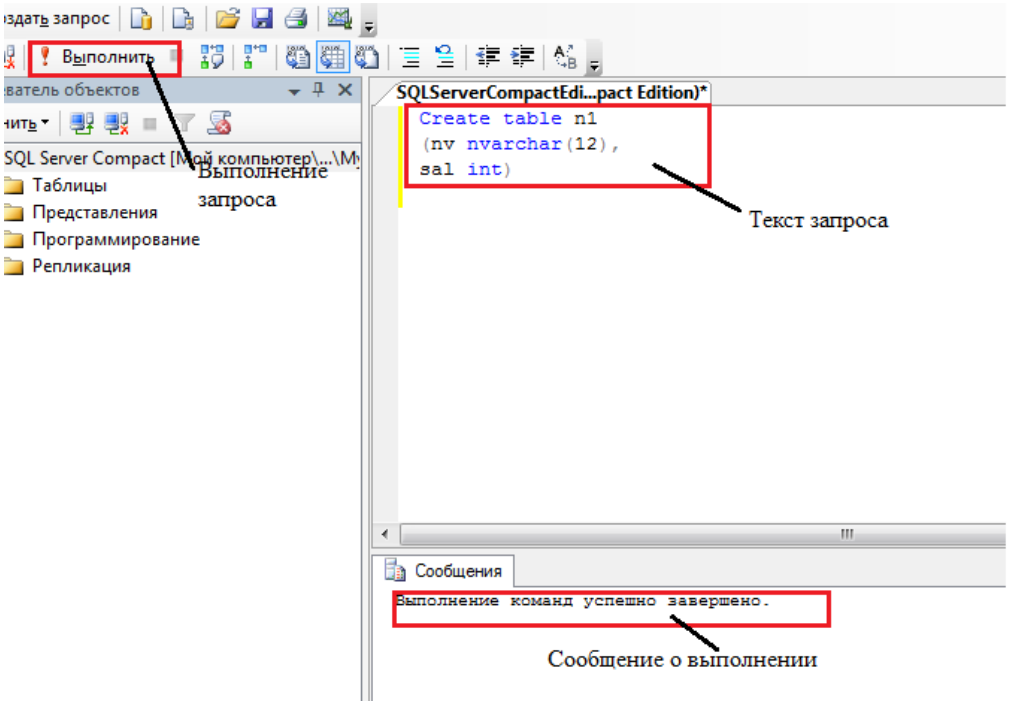


Рисунок 3 – Создание запроса к БД

В результате выполнения запросов к БД может измениться состав объектов. Обновлённый состав объектов и их структуру можно получить, обновив обозреватель объектов с помощью кнопки  (рис. 4).

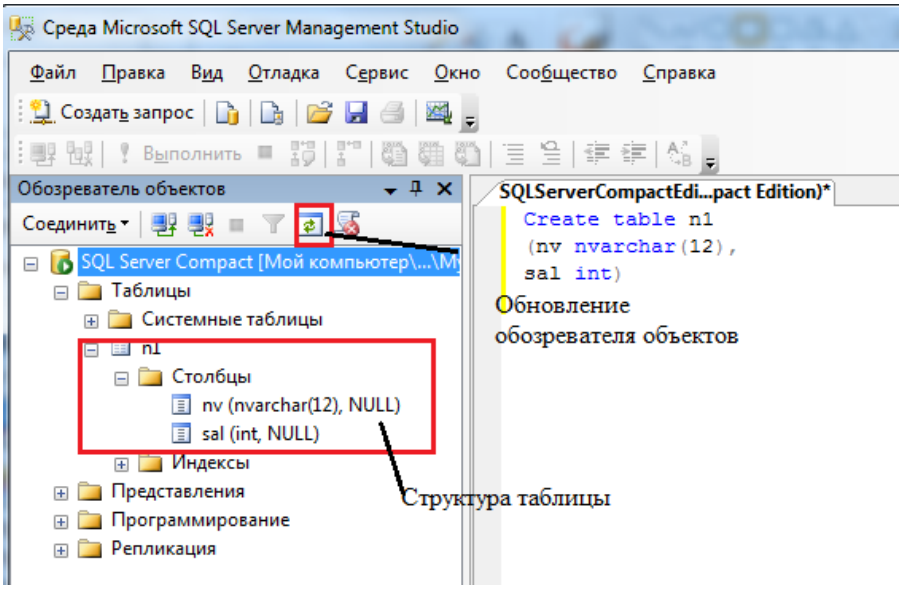


Рисунок 4 – Работа с обозревателем объектов

Описать структуру новой таблицы можно с помощью контекстного меню, которое открывается нажатием правой кнопки мыши на объекте таблицы в обозревателе объектов. В этом случае

открывается диалоговое окно (рис. 5), в котором можно задать поля создаваемой таблицы.

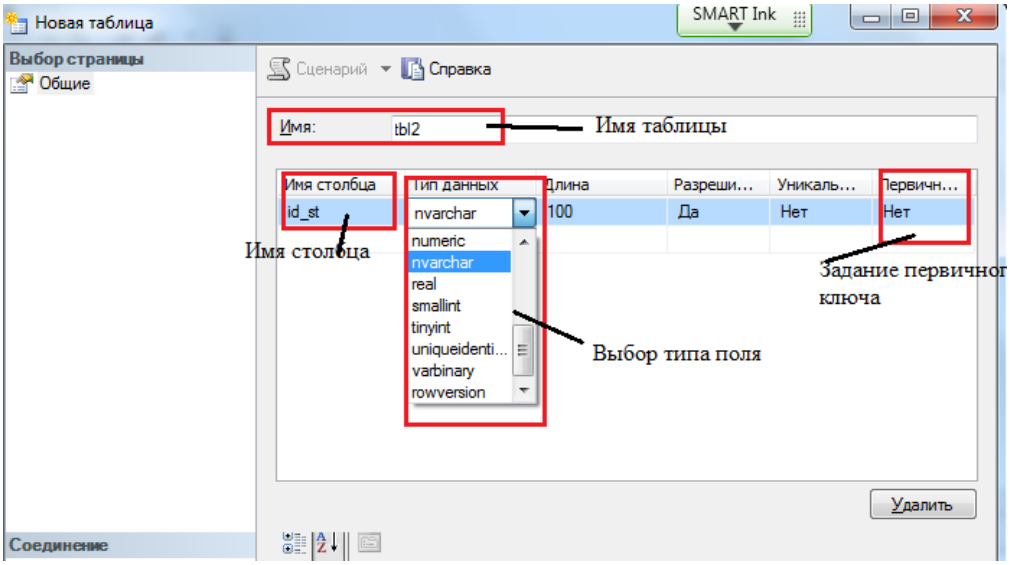


Рисунок 5 – Создание таблицы в диалоговом режиме

Изменение структуры таблицы может быть выполнено запросом с оператором ALTER TABLE. С его помощью можно добавить, удалить или изменить описание столбца таблицы. Для изменения описания столбца таблицы используется следующий вид этого оператора

ALTER TABLE *имя_таблицы*
ALTER COLUMN *имя_столбца {новый_тип_данных* ***[NULL | NOT NULL]}***

Пример использования такого оператора приведён на рисунке 6.

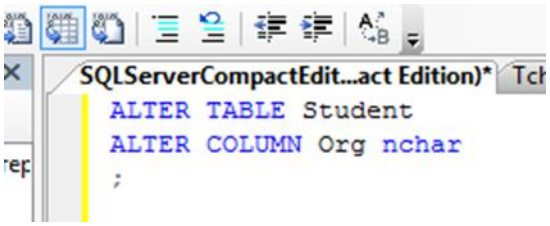


Рисунок 6 – Изменение описания столбца таблицы

При добавлении нового столбца в описание таблицы оператор ALTER TABLE принимает вид:

ALTER TABLE *имя_таблицы*
ADD { [*имя_столбца тип_данных]* | *имя_столбца* ***AS*** *выражение* ***} [,...n]***

Один оператор позволяет добавить несколько столбцов. Пример применения оператора добавления столбца в описание таблицы приведён на рисунке 7.

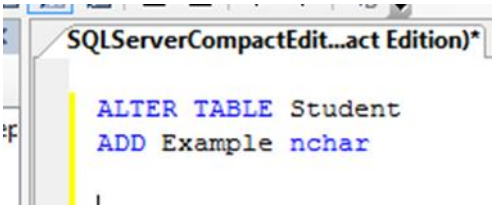


Рисунок 7 – Добавление описания столбца таблицы

Используя оператор ALTER TABLE, можно удалить описание одного или нескольких столбцов таблицы. В этом случае оператор принимает вид:

ALTER TABLE имя_таблицы
DROP {COLUMN имя_столбца}[,...n]

Пример применения оператора удаления столбца таблицы приведён на рисунке 8.

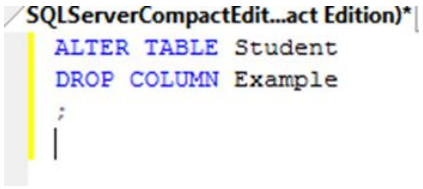


Рисунок 8 – Удаление описания столбца таблицы

3. Последовательность выполнения практической работы

1. Запустить программу SQL Server Management Studio, выполнить регистрацию и соединение с БД.
2. Средствами оператора CREATE TABLE создать описание таблиц БД, схема которой приведена в Приложении 1.
3. Используя оператор ALTER TABLE, изменить описание выбранного столбца таблицы.
4. Используя оператор ALTER TABLE, добавить описание нового столбца таблицы.
5. Используя оператор ALTER TABLE, удалить описание добавленного столбца таблицы.
6. Удалить описание таблицы (по указанию преподавателя).
7. Восстановить описание удалённой таблицы, используя встроенные средства SQL Management Studio.

8. Оформить отчёт по практической работе.

4. Методические указания

При определении описания полей таблицы не учитывать ограничения целостности на их значения.

5. Требования к оформлению отчёта

Отчёт по практической работе должен включать следующие разделы.

1. Титульный лист (Приложение 2).
2. Схема БД.
3. Операторы описания структуры всех таблиц БД.
4. Операторы изменения структуры таблицы.
5. Оператор удаления описания таблицы.
6. Скриншоты работы со средствами SQL Management Studio.
7. Выводы по работе.
8. Ответы на контрольные вопросы.

Контрольные вопросы

1. Определите назначение оператора Create Table.
2. Какие описатели полей в операторе Create Table являются обязательными?
3. Какие встроенные типы данных вам известны?
4. Почему следует избегать использования типов ntext, text и image при создании БД?
5. Для чего используется оператор Drop Table?
6. Определите назначение оператора Alter Table.

Практическая работа 2. ОБРАБОТКА ДАННЫХ ИЗ БД

Цель работы – приобрести навыки использования операторов Insert, Delete, Update для заполнения БД.

1. Основные понятия

Для добавления данных в таблицу используется оператор

INSERT INTO *<имя_таблицы>* ***[(****имя_столбца [,...n]* ***)]***
VALUES (*значение[,...n]****);***

Он позволяет добавить одну строку в таблицу данных. Список столбцов задаёт имена столбцов, которым будут присвоены значения в добавляемых записях, указанных после ключевого слова **VALUES.** Список может быть опущен. Тогда должны быть определены значения всех столбцов столбцы таблицы (кроме объявленных как счётчик). Порядок вводимых значений в этом случае должен совпадать с порядком описания столбцов при создании таблицы. Если в команде **INSERT** указывается конкретный список имен столбцов, то любые пропущенные в нём столбцы должны быть определены при создании таблицы с описателями **NULL** или **DEFAULT**.

При задании значений данных при вводе может быть использованы функции преобразования типов. В T-SQL определены универсальные функции **CONVERT** и **CAST**, которые преобразовывают значения одного типа в значения другого типа (если такие изменения возможны). Правила использования функций:

CONVERT (*тип_данных[(длина)], выражение [,стиль])*
CAST (*выражение* ***AS*** *тип_данных****)***

Аргумент ***стиль*** позволяет управлять стилем представления значений следующих типов данных*:* дата/время, денежный или нецелочисленный.

При вводе данных в БД можно в виде констант (т.е. конкретных значений) задать только числовые и символьные данные. При этом весе символьные константы содержатся между знаками ‘ ‘. Данные других типов требуют программного ввода (например, изображения) или использования явного преобразования типов (например, логические данные).

Пример оператора ввода данных приведён на рисунке 9.

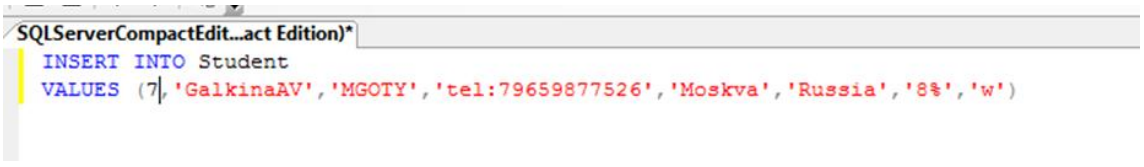
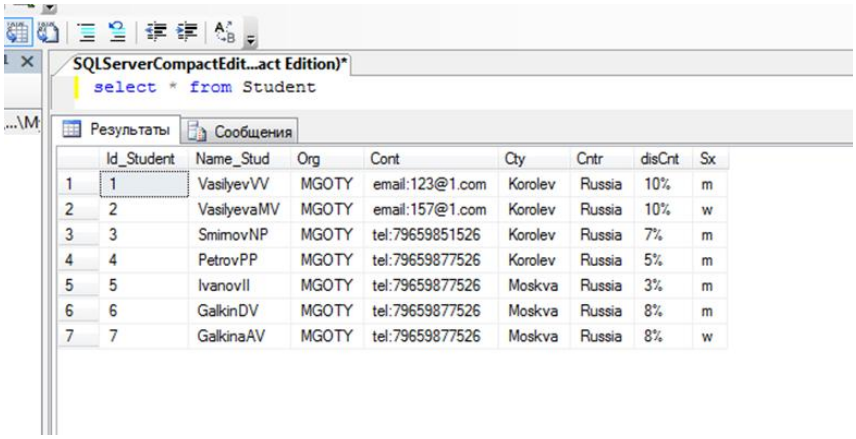


Рисунок 9 – Ввод данных в таблицу Student

Поверить результаты ввода данных в таблицу можно с помощью оператора

SELECT * FROM*имя_таблицы*

Результатом её выполнения будет список всех введённых строк (рис. 10).



	Id_Student	Name_Stud	Org	Cont	Cty	Cntr	disCnt	Sx
1	1	VasilyevVV	MGOTY	email:123@1.com	Korolev	Russia	10%	m
2	2	VasilyevaMV	MGOTY	email:157@1.com	Korolev	Russia	10%	w
3	3	SmimovNP	MGOTY	tel:79659851526	Korolev	Russia	7%	m
4	4	PetrovPP	MGOTY	tel:79659877526	Korolev	Russia	5%	m
5	5	IvanovII	MGOTY	tel:79659877526	Moskva	Russia	3%	m
6	6	GalkinDV	MGOTY	tel:79659877526	Moskva	Russia	8%	m
7	7	GalkinaAV	MGOTY	tel:79659877526	Moskva	Russia	8%	w

Рисунок 10 – Просмотр содержимого таблицы Student

Изменения данных в таблице выполняет оператор

UPDATE *имя_таблицы* ***SET*** *имя_столбца= <выражение>[,...n]* ***[WHERE*** *<условие_отбора>**]***

В предложении ***SET*** указываются имена одного и более столбцов, данные в которых необходимо изменить. Поэтому одним оператором можно изменить сразу несколько столбцов. Пример изменения значений столба таблицы приведён на рисунке 11.

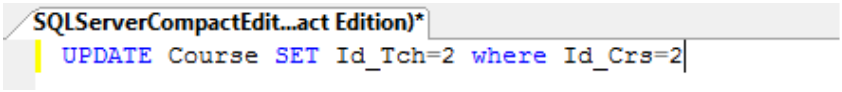


Рисунок 11 – Изменения значений столба ID_Tch таблицы Course

Для удаления строк таблицы применяется оператор

DELETE FROM *<имя_таблицы>*
[WHERE *<условие_отбора>**];***

Он позволяет удалить несколько строк из таблицы, которые удовлетворяют условию отбора (рис. 12). Если опустить предложение ***WHERE***, из таблицы будут удалены все записи, но сама таблица сохранится.

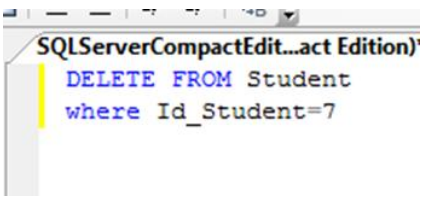


Рисунок 12 – Удаление строк из таблицы Student

2. Последовательность выполнения практической работы

1. Запустить программу SQL Server Management Studio, выполнить регистрацию и соединение с БД.
2. Средствами оператора INSERT добавить данные в созданные таблицы.
3. Используя команду UPDATE, изменить значение выбранного столбца таблицы.
4. Используя оператор DELETE, удалить строки из таблицы (по указанию преподавателя).
5. Проверить заполненные данные, используя встроенные средства SQL Management Studio.
6. Оформить отчёт по практической работе.

3. Требования к оформлению отчёта

Отчёт по практической работе должен включать следующие разделы.

1. Титульный лист.
2. Схема БД.
3. Исходные данные для заполнения таблиц.
4. Операторы работы с данными БД.
5. Скриншоты работы со средствами SQL Management Studio, отражающие содержимое таблиц.
6. Выводы по работе.
7. Ответы на контрольные вопросы.

Контрольные вопросы

1. Для чего используется оператор INSERT?
2. Какие описатели полей в операторе CREATE TABLE являются обязательными?
3. Какие встроенные типы данных вам известны?
4. Почему следует избегать использования типов ntext, text и image при создании БД?
5. Для чего используется оператор DROP TABLE?
6. Определите назначение оператора ALTER TABLE.
7. Какое соответствие должно быть между списком полей и списком значений в операторе INSERT?

Практическая работа 3. ПОИСК ДАННЫХ

Цель работы – приобрести навыки использования оператора SELECT при поиске данных из одной таблицы.

1. Основные понятия

Операция поиска данных в БД является наиболее часто используемой. Запрос на поиск данных выполняется в масштабе наборов данных, а не отдельных записей, поэтому один и тот же запрос может быть реализован несколькими способами, однако время выполнения таких запросов может существенно отличаться. Важно помнить, что оператор SELECT не изменяет данные в БД, а только производит их выборку в соответствии с заданными критериями.

Оператор SELECT имеет следующий формат:

SELECT [предикат ***]***
{ * | [имя_столбца ***[AS*** новое_имя***]] }*** [,...n]
FROM имя_таблицы ***[[AS]*** псевдоним***]*** [,...n]
[WHERE <условие_отбора>***]***
[GROUP BY имя_столбца [,...n]***]***
[HAVING <критерии выбора групп>***]***
[ORDERBYимя_столбца [,...n] ***];***

Ключевое слово WHERE позволяет определить условие поиска, определяющее те строки, которые должны быть выбраны при выполнении запроса. Условие отбора представляет собой логическое выражение, в котором могут быть использованы имена полей, логические операции ***AND, OR***и ***NOT***, операции сравнения, встроенные функции. ***Операция*** – это символ, который определяет выполняемое над одним или несколькими выражениями действие.

Операции делятся на следующие категории.

1. Арифметические операции.
2. Операции присваивания.
3. Побитовые операции.
4. Операции сравнения.
5. Логические операции.
6. Унарные операторы.

Набор операций, используемых при задании условия отбора, приведён в таблице 4.

Таблица 4 – Логические операции и кванторы

Операции	Результат
ALL	***TRUE,*** если весь набор сравнений даёт результат ***TRUE***
AND	***TRUE***, если оба булевых выражения дают результат ***TRUE***
ANY	***TRUE***, если хотя бы одно сравнение из набора даёт результат ***TRUE***
BETWEEN	***TRUE***, если операнд находится внутри диапазона
EXISTS	***TRUE***, если подзапрос возвращает хотя бы одну строку
IN	***TRUE***, если операнд равен одному выражению из списка или одной или нескольким строкам, возвращаемым подзапросом
LIKE	***TRUE***, если операнд совпадает с шаблоном
NOT	Логическое отрицание
OR	***TRUE***, если любое булево выражение равно ***TRUE***
SOME	***TRUE***, если несколько сравнений из набора дают результат ***TRUE***

Если в выражении присутствует несколько операций, то порядок их выполнения определяется приоритетом операций. Приоритет выполнения операций определяется в соответствии со следующими уровнями (от самого высокого к самому низкому).

1. () – выражения в скобках.
2. +, -, ~ – унарные операции.
3. *, /, % – арифметические операции типа умножения.
4. +, - – арифметические операции типа сложения.
5. =, >, <, >=, <=, <> – операции сравнения.
6. ^ (побитное исключающее ИЛИ), & (побитное И), | (побитное ИЛИ).
7. ***NOT***.
8. ***AND***.
9. ***ALL, ANY, BETWEEN, IN, LIKE, OR, SOME***.

Операции с одинаковым приоритетом вычисляются слева направо. Для изменения порядка выполнения операций используются скобки. Выражения в скобках вычисляются первыми.

Ключевое слово ***GROUP BY*** задаёт правило группировки выбранных данных. Группирование данных – это размещение данных в столбцах с повторяющимися значениями в определённом логическом порядке.

Ключевое слово ***HAVING*** даёт дополнительную возможность «профильтровать» отобранные записи. При этом надо учитывать следующие отличия применения условий, определённых после ключевых слов ***WHERE*** и ***HAVING:***

- ***WHERE*** накладывает ограничения на строки, ***HAVING*** – на группы;

- предложение ***WHERE*** отсеивает строки до группировки, а предложение ***HAVING*** – после;
- в условии поиска ***WHERE*** нельзя задавать агрегатные функции;
- в большинстве систем элементы предложения ***HAVING*** должны включаться в список выбора. На предложение ***WHERE*** подобное ограничение не распространяется.

Ключевое слово ***ORDER BY*** задаёт сортировку данных выходного набора в заданной последовательности. Сортировка может выполняться по нескольким полям, в этом случае они перечисляются за ключевым словом ***ORDER BY*** через запятую. По умолчанию реализуется сортировка по возрастанию, она задаётся ключевым словом ***ASC***. Для выполнения сортировки в обратной последовательности необходимо указать ключевое слово ***DESC.***

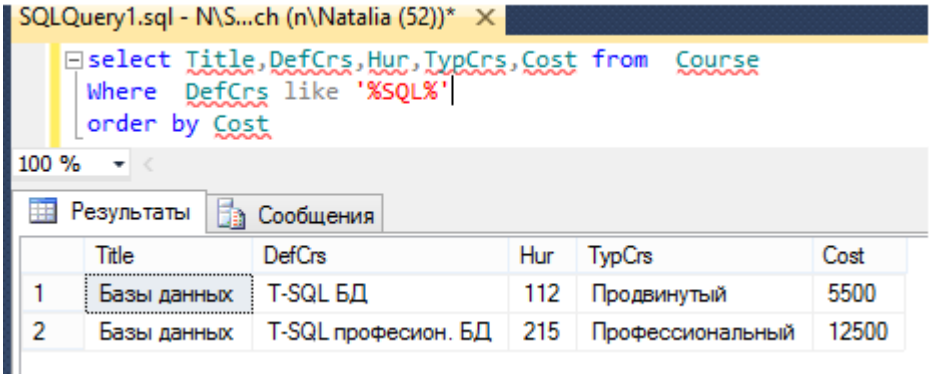
Оператор ***SELECT*** не определяет точные шаги, которые сервер базы данных должен предпринять, для его реализации. Это означает, что сервер БД проанализирует входную инструкцию и выберет эффективный способ извлечения запрошенных данных. Эту задачу решает оптимизатор запросов. Поэтому программист не должен планировать оптимальный ход выполнения запроса. Оператор ***SELECT*** определяет следующее:

- формат результирующего набора данных, который определяется, главным образом, в списке выбора, однако другие предложения, например, ***ORDER BY*** и ***GROUP BY***, также позволяют определить конечную форму результирующего набора;
- таблицы, в которых содержатся исходные данные, определяются после ключевого слова ***FROM***;
- условие отбора записей, которые задаются после ключевого слова WHERE в виде логического выражения;
- условия вывода найденных записей, которые определяются в предложениях ***HAVING, GROUPBY, ORDERBY.***

Следует отметить, что один запрос на поиск данных может быть реализован разными способами. В общем случае нет рекомендаций по выбору вида оператора ***SELECT***, и многое зависит от предпочтений программиста. Однако при поиске данных на основе символьных значений целесообразно использовать поиск по шаблону, задавая его с помощью операции ***LIKE.*** Для задания шаблона можно использовать следующие символы:

- % – вместо этого символа может быть подставлено любое количество произвольных символов;
- _ – заменяет один символ строки;
- [] – вместо символа строки будет подставлен один из возможных символов, указанный в этих ограничителях;
- [^] – вместо соответствующего символа строки будут подставлены все символы, кроме указанных в ограничителях.

Пример оператора ***SELECT*** и результат его обработки представлен на рисунке 13.



	Title	DefCrs	Hur	TypCrs	Cost
1	Базы данных	T-SQL БД	112	Продвинутый	5500
2	Базы данных	T-SQL професион. БД	215	Профессиональный	12500

Рисунок 13 – Поиск данных в таблице Course

2. Последовательность выполнения практической работы

1. Запустить программу SQL Server Management Studio, выполнить регистрацию и соединение с БД.

2. Добавить с помощью оператора INSERT данные в таблицы БД для выполнения поиска по заданным в п. 3 условиям.

3. Выполнить оператор SELECT для поиска данных из одной таблицы в соответствии с заданием.

3.1. Составить список названий курсов с указанием их стоимости, которые имеют тип «начальный».

3.2. Составить список названий курсов, у которых стоимость менее 1500 руб. и с которые имеют тип «начальный». Список отсортировать по названию курсов.

3.3. Составить список названий курсов, у которых стоимость более1500 руб. и с которые имеют тип, отличный от «начальный».

3.4. Составить список названий курсов с указанием их стоимости, в названии которых есть слово ‘SQL’. В результирующий список включить название курса, количество часов, тип курса, его стоимость. Список отсортировать по стоимости.

3.5. Составить список слушателей из заданной организации. Список должен содержать ФИО слушателя, контактные данные, размер скидки.

3.6. Составить список слушателей с фамилиями, начинающимися на букву К. Список должен содержать ФИО слушателя, контактные данные, название организации. Список отсортировать по названию организации.

3.7. Составить список слушателей-женщин с именем, начинающимися на букву А. Список должен содержать ФИО слушателя, контактные данные, название организации. Список отсортировать по названию организации.

3.8. Составить список слушателей заданной организации только женского пола и с именами, не заканчивающимися на букву «я» или «ь».

3.9. Составить список слушателей, которые имеют скидку на обучение в диапазоне от 3 до 1. Список должен содержать ФИО слушателя, контактные данные, название организации. Список отсортировать по фамилии слушателя.

3.10. Составить список преподавателей, у которых стоимость часа работы лежит в диапазоне от 600 руб. до 900 руб. Список должен содержать ФИО преподавателя, контактные данные, степень. Список отсортировать по фамилии преподавателя.

3.11. Составить список преподавателей, не имеющих степени и имеющих стоимость часа более 1000 руб. Список должен содержать ФИО преподавателя, контактные данные, стоимость часа работы. Список отсортировать по стоимости часа.

3.12. Составить список преподавателей только женского пола и с фамилиями, начинающимися на букву Б или В.

3.13. Составить список преподавателей, имеющих степень и со стоимостью часа менее 900 руб. Список должен содержать ФИО преподавателя, контактные данные, стоимость часа работы, степень. Список отсортировать по фамилии преподавателя.

3.14. Составить список преподавателей мужского пола, не имеющих степени, имя которых начинается на букву С. Список должен содержать ФИО преподавателя, контактные данные, степень, стоимость часа работы. Список отсортировать по степени и фамилии преподавателя.

3.15. Составить список преподавателей, имеющих степень, фамилия которых начинается на букву М и имеющих стоимость часа более 1000 руб. Список должен содержать ФИО преподавателя, контактные данные, стоимость часа работы. Список отсортировать по степени и стоимости часа.

4. Оформить отчёт по практической работе.

3. Методические указания

Если запрос допускает различные методы выборки данных из нескольких таблиц, обосновать используемый метод.

4. Требования к оформлению отчёта

Отчёт по практической работе должен включать следующие разделы.

1. Титульный лист.
2. Схема БД.
3. Текст запроса и соответствующий ему оператор выборки данных.
4. Обоснование выбранного метода реализации запроса.
5. Скриншоты результатов выполнения запросов в БД.
6. Выводы по работе.
7. Ответы на контрольные вопросы.

Контрольные вопросы

1. Как исключить повторяющиеся при выводе записи?
2. Какие символы могут использоваться в шаблоне?
3. Как задать сортировку данных по убыванию?
4. В чём отличие использования ключевых слов WHERE и HAVING?
5. Определите результат применения операции BETWEEN.
6. Какая операция проверяет принадлежность значения некоторому множеству?

Практическая работа 4.
ПРИМЕНЕНИЕ АГРЕГАТНЫХ ФУНКЦИЙ

Цель работы – приобрести навыки использования агрегатных функций и группировки при реализации запросов к БД.

1. Основные понятия

С помощью агрегатных (итоговых) функций в SQL-запросе можно получить статистические данные, вычисленные на основе множества отобранных значений. В T-SQL определены следующие основные агрегатные функции:

- ***Count (Выражение)*** – определяет количество записей в выходном наборе SQL-запроса;
- ***Min/Max (Выражение)*** – определяют наименьшее и наибольшее из множества значений в некотором поле запроса;
- ***Avg (Выражение)*** – эта функция позволяет рассчитать среднее для множества значений, хранящихся в определённом поле записей, отобранных запросом;
- ***Sum (Выражение)*** – вычисляет сумму множества значений, содержащихся в определённом поле записей, отобранных запросом.

Чаще всего в качестве выражения выступают имена столбцов. Выражение может вычисляться и по значениям нескольких таблиц. Агрегатные функции оперируют со значениями в единственном столбце таблицы или с арифметическим выражением и возвращают единственное значение. Функции ***COUNT***, ***MIN*** и ***MAX*** применимы как к числовым, так и к нечисловым полям. Функции ***SUM*** и ***AVG*** могут использоваться только в случае числовых полей. При вычислении результатов любых функций сначала исключаются все пустые значения. После этого требуемая операция применяется только к оставшимся конкретным значениям столбца. Исключением является функция ***COUNT(*),*** которая возвращает общее количество строк результирующей таблицы, – особый случай использования функции.

Агрегатные функции могут использоваться только в списке предложения ***SELECT*** и в составе предложения ***HAVING***.

Агрегатные функции используются так же, как имена полей в операторе SELECT. Когда агрегирующая функция используется в операторе выбора, который не содержит конструкцию GROUP BY, то

результатом будет одно итоговое значение независимо от наличия или отсутствия конструкции отбора WHERE (рис. 14).

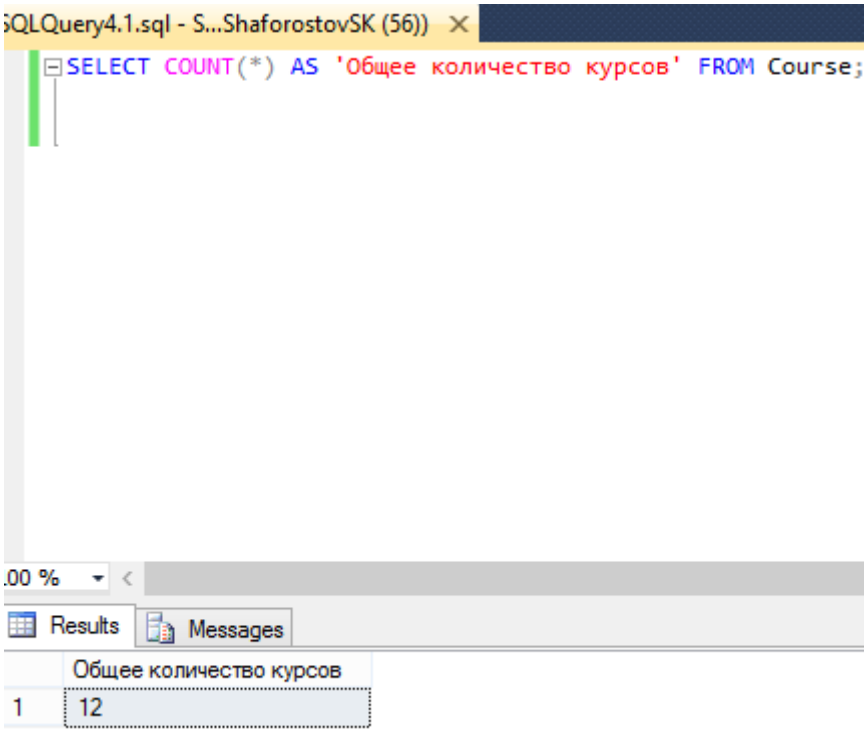


Рисунок 14 – Подсчет общего числа записей в таблице Course

Вместе с агрегатными функциями **SUM, AVG** и **COUNT** можно использовать опцию **distinct** (различные). Однако её нельзя использовать в функциях **MIN, MAX** и **COUNT(*)**. Если используется эта опция, то перед применением агрегирующей функции устраняются все дублирующие значения аргумента. Кроме того, если используется эта опция, то аргумент не может быть арифметическим выражением. Он должен состоять только из названия столбца таблицы.

Опция **distinct** должна быть расположена внутри скобок перед названием столбца. Например, для нахождения числа различных организаций слушателей курсов можно использовать следующий запрос:

```
SELECTCOUNT(DISTINCT Org)
FROM Student
```

Опция **GROUP BY** почти всегда появляется в операторе при вычислении агрегирующих функций. Она позволяет вычислить агрегирующую функцию для каждой группы отобранных записей.

Важно! Группировку можно проводить только по тем полям, имена которых указаны в списке выбора.

Пример выполнения запроса с агрегатной функцией и опцией **GROUP BY** приведен на рисунке 15.

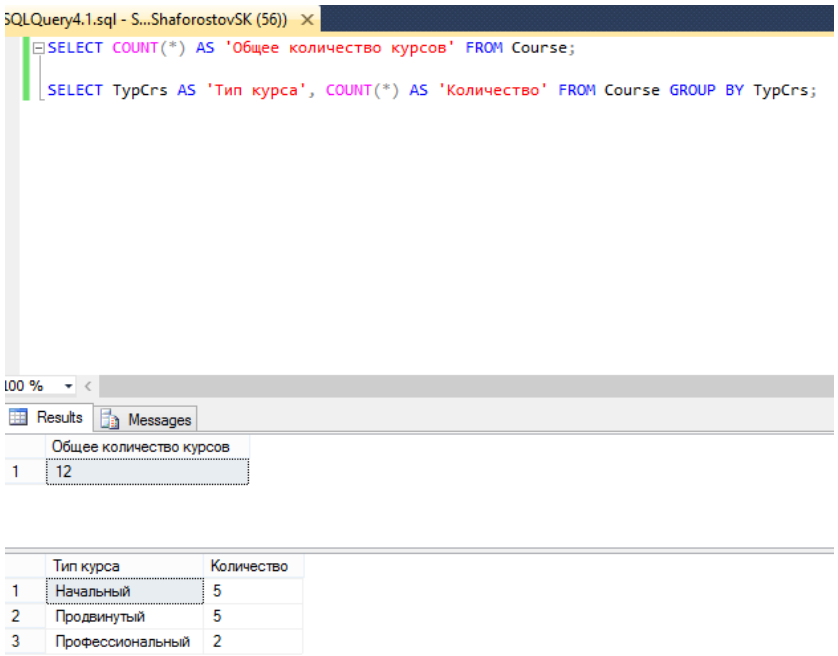


Рисунок 15 – Использование группировки при вычислении агрегатных функций

2. Последовательность выполнения практической работы

1. Запустить программу SQL Server Management Studio, выполнить регистрацию и соединение с БД.

2. Реализовать запросы к БД:

2.1. Составить отчёт по количеству учебных курсов каждого типа. Отчёт должен начинаться заголовком и цифрой, определяющей общее количество курсов. Далее следует таблица, включающая перечень типов учебных курсов с цифрами, не равными нулю.

2.2. Составить отчёт по количеству слушателей, приехавших из разных городов. Отчет должен начинаться с соответствующего заголовка и цифры – общего количества слушателей. Далее следует таблица, включающая перечень городов с цифрами, не равными нулю.

2.3. Составить отчёт о преподавателях учебных курсов. Отчёт должен начинаться с заголовка и цифры – общего количества преподавателей. Далее таблица, включающая перечень учебных степеней и количество преподавателей с заданной степенью с цифрами, не равными нулю.

2.4. Составить отчёт по стоимости часа преподавателей учебных курсов. Отчёт должен начинаться с заголовка и цифры – общего количества преподавателей. Далее таблица, включающая стоимость часа и количество преподавателей (с цифрами, не равными нулю).

2.5. Составить отчёт по длительности курсов. Отчёт должен начинаться с заголовка и цифры – общего количества курсов. Далее таблица, включающая перечень продолжительности курса и их количество (цифрами, не равными нулю).

2.6. Составить отчёт по курсам с учётом их типа. Отчёт должен начинаться с заголовка и цифры – общего количества курсов. Далее таблица, включающая тип курса и максимальную стоимость для этого типа.

2.7. Составить отчёт по курсам с учётом их типа. Отчёт должен начинаться с заголовка и цифры – общего количества курсов. Далее таблица, включающая тип курса, количество и среднюю стоимость для этого типа.

2.8. Составить отчёт преподавателям учебных курсов. Отчёт должен начинаться с заголовка и цифры – общего количества преподавателей. Далее таблица, включающая перечень учебных степеней, минимальную и максимальную стоимость часа для преподавателей с данной степенью.

2.9. Составить отчёт по средней стоимости часа преподавателей учебных курсов. Отчёт должен включать перечень учебных степеней и среднюю стоимость часа для преподавателей с данной степенью.

2.10. Составить отчёт о размере скидок на обучение. Отчёт должен включать общее количество слушателей. Далее таблица с указанием размера скидки и числа слушателей, имеющих эту скидку.

3. Оформить отчёт по практической работе.

3. Требования к оформлению отчёта

Отчёт по практической работе должен включать следующие разделы.

1. Титульный лист.
2. Схема БД.
3. Текст запроса и соответствующий ему оператор выборки данных.

4. Скриншоты результатов выполнения запросов в БД.
5. Выводы по работе.
6. Ответы на контрольные вопросы.

Контрольные вопросы

1. Для чего используются агрегатные функции?
2. Перечислите агрегатные функции, которые используются при работе с числовыми полями.
3. Как исключить суммирование повторяющихся значений?
4. В каких случаях используется группировка выводимых записей?
5. Какова особенность использования опции GROUPBY совместно с агрегатными функциями?

Практическая работа 5. РЕАЛИЗАЦИЯ СЛОЖНЫХ ЗАПРОСОВ

Цель работы – приобрести навыки использования оператора Select при поиске данных из нескольких взаимосвязанных таблиц.

1. Основные понятия

Как правило, извлекаемые из БД данные, содержатся в разных таблицах. Во время работы с данными, хранящимися в нормализованной БД, часто возникают ситуации, когда необходимая информация не может быть получена только из одной таблицы. В других ситуациях необходима выборка данных на основе условий, которые проверяются по данным другой таблицы.

При составлении запросов на выборку данных из нескольких таблиц можно использовать различные возможности T-SQL

- команду соединения;
- запросы с подзапросами.

Формирование результирующего набора данных зависит от того, какой вид оператора соединения используется. В составе языка T-SQL существует несколько команд, реализующих соединение: внутреннее соединение (***INNER JOIN***), внешнее левое соединение (***LEFTOUTER JOIN*** или ***LEFT JOIN***), внешнее правое соединение (***RIGHTOUTER JOIN*** или ***RIGHT JOIN***), полное внешнее соединение (***FULLOUTER JOIN*** или ***FULL JOIN***).

Конструкции ***INNER JOIN*** представляют собой наиболее распространенную разновидность ***JOIN***. Результатом внутреннего соединения, реализующего операцию внутреннего соединения реляционной алгебры, являются строки соединяемых таблиц, значения которых по всем соединяемым столбцам совпадают. Внутренние соединения исключают строки, не соответствующие ни одной строке в другой таблице.

При внешнем соединении в результирующую таблицу помещаются не только строки с совпадающими значениями, но и строки, для которых таких значений не нашлось.

При использовании внешнего левого соединения в результирующий набор будут помещены все строки из левой таблицы (указываемой первой). Если существует совпадение значений по соединяемым столбцам, то значения полей второй таблицы заносятся в результирующий набор. Если такого совпадения нет, то в качестве значений полей второй таблицы проставляется значение ***NULL.***

При использовании внешнего правого соединения в результирующий набор будут помещены все строки из правой таблицы (указываемой второй). Если существует совпадение значений по соединяемым столбцам, то значения полей первой таблицы заносятся в результирующий набор. Если такого совпадения нет, то в качестве значений полей первой таблицы проставляется значение ***NULL.***

При полном внешнем соединении в результирующий набор попадут все строки, как из правой, так и из левой таблицы. При совпадении значений по соединяемым столбцам результирующая строка будет содержать значения как из левой, так и из правой таблицы. Если такого совпадения нет, то соответствующие значения полей столбцов таблиц (левой или правой) получат значение ***NULL.***

Команда ***JOIN*** используется внутри опции ***FROM*** и имеет вид:

[{ INNER | { { LEFT | RIGHT | FULL } ***JOINON****условие_соединения*

Применение запросов с подзапросами связано с теми случаями, когда невозможно решить поставленную задачу посредством одного запроса. Если условие соединения указывает столбцы, их имена и типы данных могут не совпадать, однако, если типы данных не совпадают, столбцы должны либо быть совместимыми, либо иметь типы, которые SQL Server может неявно преобразовать. Если типы

данных не могут быть преобразованы неявно, условие соединения должно проводить явное преобразование типа данных при помощи функции ***CONVERT***.

Подзапрос – это запрос, содержащийся в выражении ключевого слова ***WHERE*** другого запроса с целью дополнительных ограничений на выводимые данные. Подзапросы называют также вложенными запросами.

Синтаксис оператора Select с подзапросом:

SELECT *имя_столбца*
FROM *таблица*
WHERE *предикат* (SELECT *имя__столбца*
FROM *таблица*
WHERE *условие*);

Подзапрос можно также использовать в выражении ключевых слов в ***WHERE*** или ***HAVING*** внешних операторов выбора ***SELECT***, вставки ***INSERT,*** обновления ***UPDATE*** или удаления ***DELETE***.

Существуют два основных типа подзапросов:

- подзапросы-выражения или скалярные подзапросы, они возвращают единственное значение;
- предикатные подзапросы, они возвращают список значений.

В тех случаях, когда результатом выполнения подзапроса является множество значений, для использования полученных в подзапросе значений, в основном операторе SELECT используются операции, которые применяются к множеству: предикаты ***IN, NOT IN, ALL*** и ***ANY (SOME), EXISTS*** (***NOT EXISTS***). Фактически любой запрос, который выражается через ***IN***, может быть альтернативным образом сформулирован также с помощью ***EXISTS***. Однако обратное высказывание несправедливо.

2. Последовательность выполнения практической работы

1. Запустить программу SQL Server Management Studio, выполнить регистрацию и соединение с БД.

2. Выполнить оператор SELECT для поиска данных из нескольких таблиц в соответствии с заданием.

2.1. Составить список слушателей заданного курса. Список включает ФИО слушателя, его контактные данные, название курса, его тип. Список отсортировать по фамилии слушателя.

2.2. Составить список слушателей курсов, стоимость которых менее 1500 руб. и с которые имеют тип «начальный». Список отсортировать по названию курсов и фамилии слушателя.

2.3. Составить список контрактов для слушателей, фамилия которых начинается на букву К. Список должен содержать ФИО слушателя, название курсов, на которые он заключал контракт, стоимость каждого курса. Список отсортировать по стоимости курса.

2.4. Составить список слушателей, с которыми занимается заданный преподаватель. В результирующий список включить название курса, количество часов, тип курса, ФИО слушателя, его контактные данные. Список отсортировать по названию курса и ФИО слушателя.

2.5. Составить список курсов, на которых обучались слушателей из заданной организации. Список должен содержать ФИО слушателя, контактные данные, название курса, тип курса, количество часов. Список отсортировать по названию курса и ФИО слушателя.

2.6. Составить список слушателей с фамилиями, начинающимися на букву К, которые обучались на курсах, имеющих тип «начальный». Список должен содержать ФИО слушателя, контактные данные, название организации, название курса, его стоимость. Список отсортировать по названию организации и ФИО слушателя.

2.7. Составить список курсов, на которые отведено максимальное количество часов. Список должен содержать название, тип, стоимость, количество часов. Список отсортировать по названию курса.

2.8. Составить список курсов, стоимость которых выше средней. Список должен содержать название курса, тип, стоимость и количество часов.

2.9. Составить список слушателей, которые имеют максимальную скидку на обучение. Список должен содержать ФИО слушателя, контактные данные, название организации. Список отсортировать по фамилии.

2.10. Составить список преподавателей, у которых стоимость часа работы выше средней. Список должен содержать ФИО преподавателя, контактные данные, степень. Список отсортировать по фамилии преподавателя.

2.11. Составить список слушателей, которые обучаются на курсах с максимальной стоимостью. Список должен содержать ФИО слушателя, контактные данные, название курса, его стоимость. Список отсортировать по названию курса и ФИО слушателя.

2.12. Составить список преподавателей, которые проводят занятия на курсах, в названии которых есть слово «SQL». Список должен содержать ФИО преподавателя, название курса, его тип и продолжительность. Данные отсортировать по ФИО преподавателя.

2.13. Составить список преподавателей, имеющих степень и преподающих на курсах, длительность которых более 36 часов. Список должен содержать ФИО преподавателя, контактные данные, степень, название курса и его стоимость. Список отсортировать по фамилии преподавателя.

2.14. Составить список преподавателей мужского пола, имеющих степень, стоимость часа работы которых меньше средней стоимости часа. Список должен содержать ФИО преподавателя, контактные данные, степень, стоимость часа работы. Список отсортировать по степени и фамилии преподавателя.

2.15. Составить список преподавателей, имеющих степень и стоимость часа более 3000 руб., которые ведут занятия на курсах, имеющих тип «начальный». Список должен содержать ФИО преподавателя, контактные данные, стоимость часа работы, название курса и его стоимость. Список отсортировать по степени и стоимости часа.

3. Оформить отчёт по практической работе.

3. Методические указания

Если запрос допускает различные методы выборки данных из нескольких таблиц, обосновать используемый метод.

4. Требования к оформлению отчёта

Отчёт по практической работе должен включать следующие разделы.

1. Титульный лист.
2. Схема БД.
3. Текст запроса и соответствующий ему оператор выборки данных.
4. Обоснование выбранного метода реализации запроса.

5. Скриншоты результатов выполнения запросов в БД.
6. Выводы по работе.
7. Ответы на контрольные вопросы.

Контрольные вопросы

1. Для чего используется команда Join?
2. Назовите разновидности команды Join.
3. Что такое подзапрос?
4. Почему возникает необходимость создания подзапросов?
5. Что такое скалярный подзапрос?
6. Какие подзапросы возвращают множество значений?

Практическая работа 6. РАЗРАБОТКА ХРАНИМЫХ ПРОЦЕДУР

Цель работы – приобрести навыки разработки пользовательских хранимых процедур.

1. Основные понятия

Хранимая процедура – это наиболее часто используемый в БД программный объект, который представляет собой оформленный особым образом сценарий действий (вернее, пакет). Хранимая процедура представляет собой набор операторов T-SQL, который компилируется системой SQL Server и сохраняется в виде отдельного объекта БД. Этот объект сохраняется в кэш-области памяти БД для процедур при первом выполнении хранимой процедуры, что позволяет использовать её без повторной компиляции.

Каждая хранимая процедура имеет собственное имя, связанное с программным кодом T-SQL. Хранимая процедура хранится и исполняется на сервере БД. Она может содержать практически любые конструкции или команды, исполнение которых реализуется в SQL Server. Хранимые процедуры можно использовать для изменения данных, возврата скалярных значений или целых результирующих наборов. Хранимые процедуры являются основным способом построения интерфейса, который должен использоваться приложениями для обращения к любым данным в БД. Хранимые процедуры позволяют не только управлять доступом к БД, но и изолировать код БД для упрощения контроля доступа к данным.

Выполнение хранимых процедур на стороне сервера БД имеет ряд преимуществ, к которым следует отнести следующее:

- хранимые процедуры хранятся в компилированном виде, поэтому выполняются быстрее, чем клиентские приложения или запросы;
- выполнение обработки данных на сервере БД, а не на р клиентской рабочей станции, значительно снижает нагрузку на локальную сеть;
- хранимые процедуры реализуют модульный подход, их внедрение и изменение не затрагивает клиентские приложения;
- хранимые процедуры можно рассматривать как важный компонент системы безопасности БД, если все клиентские приложения осуществляют доступ к данным с помощью хранимых процедур, то можно ограничить или полностью исключить прямой доступ к таблицам БД;
- хранимые процедуры скрывают от пользователя структуру базы данных и разрешают ему выполнение только тех операций, которые запрограммированы в хранимой процедуре, что также является важным элементом системы безопасности БД.

Хранимые процедуры T-SQL аналогичны процедурам в других языках программирования в том смысле, что они допускают входные параметры и возвращают выходные значения в виде параметров или сообщения о состоянии (успешное или неуспешное завершение). Все операторы процедуры обрабатываются при вызове процедуры.

Хранимые процедуры используются для группирования операторов T-SQL и любых логических конструкций, необходимых для выполнения задачи. Поскольку хранимые процедуры сохраняются в виде процедурных блоков, они могут использоваться различными пользователями БД для согласованного повторяемого решения одинаковых задач и даже в различных приложениях. Хранимые процедуры также позволяют поддерживать единый подход к управлению задачей, что помогает обеспечивать согласованное и корректное внедрение любых деловых правил. Использование хранимых процедур может повысить производительность и в других отношениях. Например, использование хранимых процедур для проверки условий сервера может повысить производительность за счёт снижения количества данных, которые должны передаваться между клиентским приложением и сервером БД (снижения объёма обработки, выполняемой на клиентской машине). Можно вызывать

хранимые процедуры из сценариев, пакетных заданий и интерактивных командных строк с помощью операторов T-SQL.

Хранимые процедуры также обеспечивают простой доступ к базе данных для пользователей. Пользователи могут осуществлять доступ к базе данных, не зная деталей архитектуры таблиц и без непосредственного доступа к данным таблиц, – они просто запускают процедуры, которые выполняют требуемые задачи. Тем самым хранимые процедуры помогают обеспечивать соблюдение деловых правил.

Различают системные, пользовательские и временные хранимые процедуры. Хранимые процедуры выполняются непосредственно на сервере БД, что обеспечивает более высокое их быстродействие, чем выполнение тех же операций средствами клиентских приложений. В теле хранимой процедуре могут быть использованы операторы ЯМД и операторы, реализующие логику обработки: присваивания, логического ветвления, цикла и др. (табл. 5).

Таблица 5 – Операторы программирования

Оператор	Описание
BEGIN...END.	Определяет блок операторов
GOTO label	Безусловный переход к метке ***label***
IF...ELSE.	Условный оператор
RETURN.	Безусловный выход
RETURN ([*integer_expression*])	Возвращаемое значение – это код возврата, причём часть
WHILE …***[BREAK]*** [***CONTINUE***]	Цикл с предусловием
...BREAK	Выход из цикла ***WHILE***
...CONTINUE	Продолжение цикла ***WHILE***
DECLARE	Объявление локальных переменных
PRINT.	Выдает заданное значение на экран
CASE	Позволяет выражениям принимать значение в зависимости от условий

Создание новой и изменение имеющейся хранимой процедуры осуществляется с помощью следующей команды:

{CREATE | ALTER } PROC[EDURE] *имя_процедуры*
[{@*имя_параметра тип_данных* ***} [VARYING]***
[=*значение_по_умолчанию****][OUTPUT]][,...n]***
[WITH { RECOMPILE | ENCRYPTION | RECOMPILE,
ENCRYPTION }]
[FOR REPLICATION]
AS
*Телопроцедуры****;***

Пример создания хранимой процедуры без параметров приведён на рисунке 16.

```
CREATE PROCEDURE LateShipments
AS
SELECT   RequiredDate,
           ShippedDate,
           Shippers.CompanyName
FROM     Orders, Shippers
WHERE    ShippedDate    > RequiredDate AND
           Orders.ShipVia = Shippers.ShipperID
GO
```

Рисунок 16 – Пример описания хранимой процедуры

Чтобы задать входные параметры в хранимой процедуре, укажите список этих параметров с символом @ перед именем каждого параметра, т.е. @ имя_параметра, например, @ shipper Name. Допускается задание до 1024 параметров в хранимой процедуре.

Для возврата значения параметра хранимой процедуры в вызывающую программу используйте ключевое слово OUTPUT после имени этого параметра. Чтобы сохранить значение в переменной, которую можно использовать в вызывающей программе, используйте при вызове хранимой процедуры ключевое слово OUTPUT.

Для создания локальных переменных используется ключевое слово DECLARE. При создании локальной переменной вы должны задать для неё имя и тип данных, а также поставить перед именем переменной символ @. При объявлении переменной ей первоначально присваивается значение NULL. Пример описания хранимой процедуры с параметрами приведён на рисунке 17.

```
CREATEPROCEDUREstudentContracts(@Name_Studvarchar(40),
@contractSumintOUTPUT) AS
BEGIN
SELECT @contractSum = SUM(Course.Cost) FROM Student
JOIN Contracts ON Contracts.Id_Stud = Student.Id_Stud
JOIN Course ON Course.Id_Crs = Contracts.Id_Crs
WHERE Name_Stud = @Name_Stud;
END
```

Рисунок 17 – Хранимая процедура с параметрами

При выполнении хранимой процедуры с параметрами необходимо учитывать тот факт, что результат e1 выполнения

должен быть сохранен в соответствующей переменной, которую нужно объявить перед выполнением процедуры (рис. 18)

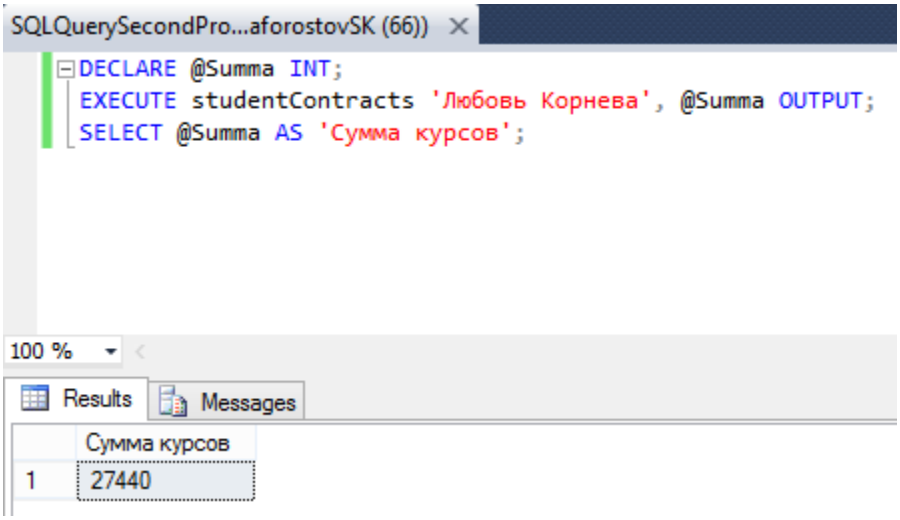


Рисунок 18 – Выполнение хранимой процедуры с параметрами

Запрос к реляционной БД может возвращать несколько строк (записей) данных. Однако приложение может обрабатывать лишь одну запись. В этой ситуации используется концепция курсора, который представляет собой указатель на ряд строк. Курсор в SQL – это область в памяти БД, которая предназначена для хранения последнего оператора SQL. Если текущий оператор – запрос к БД, в памяти сохраняется строка данных запроса, которая называется текущим значением, или текущей строкой курсора. Указанная область в памяти поименована и доступна для прикладных программ.

Обычно курсоры используются для выбора из БД некоторого подмножества хранимой в ней информации. В каждый момент времени прикладной программой может быть проверена одна строка курсора. Курсоры часто применяются в операторах SQL, встроенных в написанные на языках процедурного типа прикладные программы. Некоторые из них неявно создаются сервером базы данных, в то время как другие определяются программистами.

При работе с курсорами можно выделить следующие основные действия и связанные с ними операторы (табл. 6):

- создание или объявление курсора;
- открытие курсора, т.е. наполнение его данными, которые сохраняются в многоуровневой памяти;
- выборка из курсора и изменение с его помощью строк данных;

- закрытие курсора, после чего он становится недоступным для пользовательских программ;
- освобождение курсора, т.е. удаление курсора как объекта, поскольку его закрытие необязательно освобождает ассоциированную с ним память.

В некоторых случаях применение курсора неизбежно. Однако по возможности этого следует избегать и работать со стандартными командами обработки данных: SELECT, UPDATE, INSERT, DELETE. Помимо того, что курсоры не позволяют проводить операции изменения над всем объёмом данных, скорость выполнения операций обработки данных с помощью курсора существенно ниже, чем у стандартных средств SQL.

Таблица 6 – Операторы работы с курсорами.

Синтаксис оператора	Описание
DECLARE *имя_курсора* ***CURSOR [LOCAL | GLOBAL] [FORWARD_ONLY | SCROLL] [STATIC | KEYSET | DYNAMIC | FAST_FORWARD] [READ_ONLY | SCROLL_LOCKS | OPTIMISTIC] [TYPE_WARNING]*** ***FOR*** *SELECT_оператор* ***[FOR UPDATE [OF*** *имя_столбца[,...n]]]*	Создание или объявление курсора
OPEN *{{[**GLOBAL**]имя_курсора }*	Открытие курсора
FETCH [[NEXT | PRIOR | FIRST | LAST ***| ABSOLUTE*** *{номер_строки* *| @переменная_номера_строки}* ***| RELATIVE*** *{номер_строки | @переменная_номера_строки}]* ***FROM]{{[GLOBAL*** *]имя_курсора }| @имя_переменной_курсора }* *[**INTO** @имя_переменной [,...n]]*	Выбор данных
CLOSE *{имя_курсора | имя_переменной_курсора}*	Закрытие курсора
DEALLOCATE *{ имя_курсора |@имя_переменной_курсора }*	Освобождение курсора

При использовании ключевого слова ***LOCAL*** будет создан локальный *курсор*, который виден только в пределах создавшего его пакета, триггера, хранимой процедуры или пользовательской функции. По завершении работы пакета, триггера, процедуры или функции *курсор* неявно уничтожается. Чтобы передать содержимое *курсора* за пределы создавшей его конструкции, необходимо присвоить его параметру аргумент OUTPUT.

При использовании ключевого слова ***GLOBAL***, создается глобальный *курсор,* он существует до закрытия текущего соединения.

При использовании ключевого слова ***FORWARD_ONLY*** создается *последовательный курсор; выборку* данных можно осуществлять только в направлении от первой строки к последней.

При использовании ключевого слова ***SCROLL*** создается *прокручиваемый курсор*; обращаться к данным можно в любом порядке и в любом направлении.

При использовании ключевого слова ***STATIC*** создается *статический курсор*.

При использовании ключевого слова ***KEYSET*** создается ключевой курсор.

При использовании ключевого слова ***DYNAMIC*** создается *динамический курсор*.

Если для *курсора* ***READ_ONLY*** указать аргумент ***FAST_FORWARD***, то созданный *курсор* будет оптимизирован для быстрого доступа к данным. Этот аргумент не может быть использован совместно с аргументами ***FORWARD_ONLY*** и ***OPTIMISTIC***.

В *курсоре*, созданном с указанием аргумента OPTIMISTIC, запрещается *изменение* и *удаление строк*, которые были изменены после *открытия курсора*.

При указании аргумента ***TYPE_WARNING*** сервер будет информировать пользователя о неявном изменении типа *курсора*, если он несовместим с запросом ***SELECT***.

После *открытия курсора* происходит выполнение связанного с ним оператора ***SELECT***, выходные данные которого сохраняются в многоуровневой памяти.

После *открытия курсора* можно выбирать его содержимое (результат выполнения соответствующего запроса). При использовании ключевого слова ***FIRST*** будет возвращена самая первая строка полного результирующего набора курсора, которая становится текущей строкой.

При использовании ключевого слова ***LAST*** возвращается самая последняя строка курсора, она становится текущей строкой.

При использовании ключевого слова ***NEXT*** возвращается строка, находящаяся в полном результирующем наборе сразу же после текущей. Она становится текущей. По умолчанию команда ***FETCH*** использует именно этот способ выборки строк.

При использовании ключевого слова ***PRIOR*** возвращает строку, находящуюся перед текущей. Она становится текущей.

При использовании ключевого слова ***ABSOLUTE*** {номер_строки | @переменная_номера_строки} курсор возвращает строку по её абсолютному порядковому номеру в полном результирующем наборе курсора. Номер строки можно задать с помощью константы или как имя переменной, в которой хранится номер строки. Переменная должна иметь целочисленный тип данных. Указываются как положительные, так и отрицательные значения. При указании положительного значения строка отсчитывается от начала набора, отрицательного – от конца. Выбранная строка становится текущей. Если указано нулевое значение, строка не возвращается.

При использовании ключевого слова ***RELATIVE*** {кол_строки | @переменная_кол_строки} курсор возвращает строку, находящуюся через указанное количество строк после текущей. Если указать отрицательное значение числа строк, то будет возвращена строка, находящаяся за указанное количество строк перед текущей. При указании нулевого значения возвратится текущая строка. Возвращённая строка становится текущей.

В конструкции ***INTO*** @имя_переменной [,...n] задается список переменных, в которых будут сохранены соответствующие значения столбцов возвращаемой строки. Порядок указания переменных должен соответствовать порядку столбцов в *курсоре*, а тип данных переменной – типу данных в столбце *курсора*. Если конструкция ***INTO*** не указана, то поведение команды ***FETCH*** будет напоминать поведение команды S***ELECT*** – данные выводятся на экран.

После *закрытия курсор* становится недоступным для пользователей программы. При *закрытии* снимаются все блокировки, установленные в процессе его работы. *Закрытие* может применяться только к открытым *курсорам*. Закрытый, но не *освобождённый курсор* может быть повторно открыт. Не допускается закрывать неоткрытый *курсор*.

Закрытие курсора необязательно освобождает ассоциированную с ним память. Необходимо явным образом освободить её с помощью оператора *DEALLOCATE*. После освобождения курсора освобождается память, при этом становится возможным повторное использование имени курсора.

2. Последовательность выполнения практической работы

1. Запустить программу SQL Server Management Studio, выполнить регистрацию и соединение с БД.
2. Реализовать в виде хранимой процедуры статистические запросы, из задания к практической работе №5 (по заданию преподавателя).
3. Выполнить разработанную процедуру.
4. Разработать хранимую процедуру с входным и выходным параметрами для определения общей суммы контрактов заданного слушателя.
5. Выполнить разработанную процедуру.
6. Разработать процедуру по заданию преподавателя с использованием курсора (Приложение II).
7. Выполнить разработанную процедуру.
8. Оформить отчёт по практической работе.

3. Требования к оформлению отчёта

Отчёт по практической работе должен включать следующие разделы.

1. Титульный лист.
2. Задание на разработку хранимой процедуры.
3. Алгоритм реализации хранимой процедуры.
4. Операторы описания хранимых процедур.
5. Результаты выполнения хранимых процедур.
6. Выводы по работе.
7. Ответы на контрольные вопросы.

Контрольные вопросы

1. Для чего используются хранимые процедуры?
2. Какие виды хранимых процедур вы знаете?
3. Как задать параметры хранимой процедуры?
4. Для чего используется курсор в хранимых процедурах?
5. Как определить курсор в программе?
6. Назовите команды работы с курсором.
7. Какие режимы навигации определены в операторе Fetch?
8. Какой оператор позволяет удалить курсор?

Практическая работа 7.
РЕАЛИЗАЦИЯ ДЕКЛАРАТИВНЫХ ПРАВИЛ ЦЕЛОСТНОСТИ

Цель работы – применение статических правил для контроля целостности данных.

1. Основные понятия

При обработке данных необходима гарантия сохранения целостности данных в базе, поэтому важным этапом проектирования РБД является обеспечение целостности базы данных. Сервер СУБД включает специальные средства контроля целостности данных, которые не зависят от клиентских программ и связанных непосредственно с таблицами БД. Таким образом, принципиально не важно, каким образом выполнен доступ к БД: через SQL-интерфейс, через ODBC-драйвер из приложения, через Интернет. В любом из этих случаев контроль целостности данных осуществляет сервер. Если правил целостности данных нарушены, сервер известит клиента об ошибке.

К средствам контроля целостности данных относятся ограничители, которые определяются для отдельных столбцов таблицы и для таблицы целиком, а также триггеры, которые определяются для таблицы и выполняются в случае изменения данных в ней.

Ограничения целостности – это набор определённых правил, которые устанавливают допустимость данных и связей между ними в любой момент времени. Ограничения могут применяться на уровне столбцов и на уровне таблиц. Ограничения на уровне столбцов объявляются при создании столбца и применимы только к нему. Ограничения на уровне таблиц объявляются независимо от определений столбцов и могут применяться к одному или нескольким столбцам таблицы. Ограничения целостности имеют приоритет над триггерами, правилами и значениями по умолчанию.

Ограничители целостности для столбца таблицы – это элементарные проверки или условия, которые выполняются для операций вставки и изменения значения столбца. Если данные не соответствуют условиям проверки, то вставка или изменение отменяется. Ограничитель целостности представляет собой некоторое

логическое утверждение. Оно может быть истинным или ложным в зависимости от состояния БД.

При задании ограничений на уровне столбца можно использовать следующие виды ограничений:

- ограничение типа данных;
- ограничение первичного ключа – ***PRIMARY KEY***;
- ограничение внешнего ключа – ***FOREIGN KEY***;
- ограничение уникальности значений – ***UNIQUE***;
- ограничение значения ***NULL***;
- ограничение, задаваемое пользователем – ***CHECK***.

Ограничения, задаваемые пользователем на уровне столбца, определяются опцией ***CHECK:***

CHECK (<логическое_*условие****>)] [,…n]***

Ограничения на уровне таблицы задаются опцией ***CONSTRAINT:***

CONSTRAINT [*имя_ограничения] тип_ограничения [(столбец[,…])][предикат]*

CONSTRAINT [имя_ограничения] – начинает определение ограничения и определяет его имя. Если имя не задано, то система создаст имя автоматически. Лучше задавать ограничениям смысловые имена. Это поможет при возникновении ошибки определить, какое правило целостности было нарушено и внести необходимые изменения в данные.

Ограничение ***PRIMARY KEY (первичный ключ)*** определяет специальные случаи комбинирования ограничений ***UNIQUE*** и ***NOT NULL***.

Первичные ключи имеют следующие особенности:

- таблица может содержать только один первичный ключ;
- внешние ключи по умолчанию ссылаются на первичный ключ таблицы.

Первичный ключ может строиться на нескольких полях таблицы, т.е. быть составным. Составные ключи указываются через запятую после последнего поля, например:

primary key(ID_Student,ID_Lesson),

Ограничение ***FOREIGN KEY (внешний ключ)*** определяет, что это поле таблицы содержит ссылки на первичный ключ другой таблицы. Соответствие внешних и первичных ключей

поддерживается СУБД автоматически. Для этого необходимо задать описание при описании поля ограничение внешнего ключа:

FOREIGN KEY [(<список полей>.,..)] REFERENCES <имя таблицы> [(<список полей>)]

При определении ограничения внешнего ключа необходимо учитывать следующие требования:

- соответствие столбцов первичного и внешнего ключа по типу и размеру данных;
- если внешний ключ ссылается на первичный ключ другого отношения, имена полей первичного ключа можно не указывать;
- если внешний ключ составной, список полей, входящих в ключ, указывается после перечисления всех полей таблицы с ключевым словом ***FOREIGN KEY.***

Ограничение уникальности ***UNIQUE*** проверяет уникальное значение поля в пределах столбца таблицы. Если при создании таблицы для столбца указывается ограничение ***UNIQUE***, то СУБД блокирует любую попытку ввести в это поле какой-либо строки значение, уже содержащееся в том же поле другой строки. Это ограничение применимо только к тем полям, которые были объявлены с описателем ***NOT NULL.***

Ограничение значения ***NULL*** используется для указания того, что в данном столбце могут содержаться неопределённые значения. Если при описании поля задано ключевое слово ***NOT NULL,*** то будут отклонены любые попытки поместить значение ***NULL*** в данный столбец.

Ограничение ***CHECK (<условие>)*** используется для задания допустимых значений вводимых данных. Это ограничение задаёт множество возможных значений поля.

Синтаксис ограничения:

CONSTRAINT [имя_ограничения] CHECK (условие)

К одному и тому же столбцу таблицы можно применить несколько ограничений ***CHECK***, соединённых друг с другом логическими операторами ***AND*** и ***OR***. Они будут применимы в той последовательности, в которой происходило их создание.

Общие ограничения целостности позволяют определять соотношения данных разных столбцов в одной строке таблицы. Они определяются после описания последнего поля таблицы и задаются

так же, как и при определении правила для отдельного столбца. Однако при задании условия можно использовать имена столбцов этой таблицы. Пример описания общего правила приведен на рисунке 19.

```
ALTERTABLECrse2
ADDCONSTRAINTConstraintDocCHECK
((Doc IN('Справка','Сертификат')AND Hur BETWEEN 16 AND 36));
```

Рисунок 19 – Правило целостности для таблицы Crse2

2. Последовательность выполнения практической работы

1. Запустить программу SQL Server Management Studio, выполнить регистрацию и соединение с БД.
2. Добавить в таблицу Course столбец Doc, имеющий следующее описание:

Имя столбца	Тип данных	Комментарий
Doc	символьный	Вид документа, может принимать значение: удостоверение, диплом, сертификат, справка

3. Определить правила целостности для поля Doc.
4. Изменить описание таблиц, добавив описание правил целостности на основе данных таблицы 3 Приложения I.
5. Проверить заданные правила, используя оператор INSERT.
6. Определить правило целостности для таблицы Course на основе следующего описания.

Поле Doc может принимать значение «справка» или «сертификат» для курсов длительностью не более 36 часов. Если длительность курса находится в диапазоне от 72 до 126 часов, то поле Doc может принимать значение «сертификат» или «удостоверение». Если длительность курса более 126 часов, то поле Doc может принимать значение «удостоверение» или «диплом».

7. Проверить заданное правило, используя команду UPDATE для поля Hur.
8. Сформулировать свое ограничение целостности для таблицы Tchr и определить для него правило целостности.
9. Проверить заданное правило целостности, используя оператор INSERT.
10. Оформить отчёт по практической работе.

3. Требования к оформлению отчёта

Отчёт по практической работе должен включать следующие разделы.

1. Титульный лист.
2. Описание правил целостности.
3. Описание правил целостности на языке SQL.
4. Тестовые наборы данных для проверки правил целостности.
5. Результаты проверки правил целостности.
6. Выводы по работе.
7. Ответы на контрольные вопросы.

Контрольные вопросы

1. Для чего используются статические правила целостности?
2. Определите назначение конструкций CHECK и CONSTRAINT.
3. Назовите ограничения использования оператора CREATERULE.
4. Какие правила применяются для задания ограничений на значение атрибута?
5. Для чего используется описатель NOT NULL?
6. В чём важность задания ограничений целостности?

Практическая работа 8.
РЕАЛИЗАЦИЯ ДИНАМИЧЕСКИХ ПРАВИЛ ЦЕЛОСТНОСТИ

Цель работы – приобрести навыки использования внешних ключей разработки триггеров для обеспечения целостности данных.

1. Основные понятия

T-SQL позволяет контролировать строго определённые наборы или диапазоны значений для помещаемых в столбец данных. С целью обеспечения целостности данных, содержащихся в разных таблицах, можно использовать ограничения ***FOREIGN KEY*** (ограничения ссылочной целостности – referential integrity constraint), задаваемые при описании структуры таблицы. Для этого также можно использовать триггеры.

Ограничение ссылочной целостности не позволяет значениям из столбца одной таблицы принимать значения, кроме как из присутствующих в столбце другой таблицы. Это делается при помощи ограничителей ***FOREIGN KEY*** (внешний ключ) и ***REFERENCES*** (указатель ссылки). Таблица, содержащая ***FOREIGN KE***Y, считается родительской таблицей. Таблица, содержащая ***REFERENCES,*** считается дочерней таблицей. Внешний ключ и указатель ссылки могут находиться в одной таблице, т.е. родительская таблица одновременно является дочерней. Для обеспечения целостности данных, содержащихся в разных таблицах, можно использовать ограничения ***FOREIGN KEY***, задаваемые при описании структуры таблицы, и триггеры.

Ограничения ***FOREIGN KEY*** поддерживают связи между таблицами. Ограничения ссылочной целостности используются при каскадном удалении, т.е. при удалении записи в родительской таблице удаляются все записи с указанным ключом из дочерних таблиц, и, наоборот, при запрете удаления/модификации, т.е. при наличии зависимых записей в дочерних таблицах, значение ключа записи в родительской таблице нельзя удалить или модифицировать. Описатель внешнего ключа применяется для принудительного установления связи между данными в двух таблицах.

Важно! В столбцах, включенных во внешний ключ, могут использоваться только значения из столбцов первичного ключа.

Ограничение ***FOREIGN KEY*** может ссылаться на столбцы в таблицах этой же БД или столбцы этой же самой таблицы. Ограничение ***FOREIGN KEY*** не обязательно должно быть связано с ограничением ***PRIMARY KEY*** в другой таблице. Кроме того, с помощью этого ограничения могут указываться столбцы ограничения ***UNIQUE*** в другой таблице. Ограничение ***FOREIGN KEY*** может содержать значения ***NULL.***

Ограничение ***FOREIGN KEY*** обеспечивает целостность ссылок по следующим правилам: запрещается изменение данных в таблице первичного ключа, если такие изменения сделают недопустимой ссылку в таблице внешнего ключа. Если при попытке удалить строку в таблице первичного ключа или изменить значение этого ключа окажется, что удалённому или изменённому значению первичного ключа соответствует значение в ограничении ***FOREIGN KEY*** в другой таблице, то действие выполнено не будет. Для успешного

изменения или удаления строки с ограничением ***FOREIGN KEY*** можно использовать дополнительные предложения ON ***DELETE*** и ***ON UPDATE***, определяющими порядок действий с помощью специальных параметров:

- ***NO ACTION*** – удаления (изменения) не произойдет и будет выведено сообщение об ошибке;
- ***CASCADE*** – все строки с внешними ключами, указывающими на удаленную строку, также будут удалены (изменены);
- ***SET NULL*** – всем строкам с внешними ключами, указывающими на удалённую (изменённую) строку, присваивается значение ***NULL***;
- ***SET DEFAULT*** – всем строкам с внешними ключами, указывающим на удалённую (изменённую) строку, присваивается установленное для них значение по умолчанию.

Процедурные правила целостности позволяют проводить согласованные изменения данных, хранящихся во взаимосвязанных таблицах. ***Триггеры*** – это особый вид хранимых процедур, которые вызываются автоматически при возникновении ситуации, с которой связан триггер. Их удобно использовать в следующих случаях:

- для каскадных изменений в связанных таблицах БД (в большинстве случаев эти изменения можно более эффективно выполнить при помощи каскадных ограничений ссылочной целостности);
- для предотвращения случайных или неправильных операций ***INSERT, UPDATE*** и ***DELETE*** и реализации других более сложных ограничений, чем те, которые определены при помощи ограничения ***CHECK***, в отличие от ограничений ***CHECK***, триггеры могут ссылаться на столбцы других таблиц;
- для оценки состояние таблицы до и после изменения данных и предпринять действия на основе этого различия.

Пример реализации ссылочной целостности на основе внешнего ключа приведён на рисунке 20.

```
ALTERTABLEContract
ADDFOREIGNKEY(Id_Crs_FK)
REFERENCESCourse(Id_Crs)ON DELETE NO ACTION
```

Рисунок 20 – Описание ограничения ссылочной целостности на основе внешнего ключа

Триггеры, в отличие от ограничителей, могут выполнять сколь угодно сложные манипуляции над данными. Кроме контроля целостности при выполнении операций вставки и изменения данных, триггеры могут срабатывать и при удалении данных из таблицы. Можно также задавать порядок выполнения триггера относительно операции модификации БД: триггер может выполниться либо перед операцией модификации значения в таблице, либо непосредственно после такой операции.

Применение триггеров обеспечивает:

- прозрачный аудит модификации БД, который не зависит от клиентских программ и не контролируется ими, а также регистрацию событий, связанных с разграничением прав доступа к определённым таблицам или столбцам в таблицах;
- генерацию значений в столбцах одной таблицы на основе значений столбцов в другой таблице при модификации данных;
- работу с зависимыми таблицами в особенности, если они находятся на других узлах распределённой БД, чего нельзя сделать при помощи ограничителей.

При необходимости триггеры можно отключать (запрещать), а затем снова разрешать. Запрещение триггеров применяется обычно при массовых загрузках данных в таблицы извне для уменьшения времени загрузки.

Триггеры пишутся на процедурных расширениях SQL. Триггер не может иметь входных параметров и возвращать результат. В SQL Server существует два вида триггеров, которые различаются по способу его выполнения: ***AFTER*** и ***INSTEAD OF***.

1. ***AFTER***. ***Триггер*** выполняется после успешного выполнения вызвавших его команд. Если же команды по какой-либо причине не могут быть успешно завершены, триггер не выполняется. Следует отметить, что изменения данных в результате выполнения запроса пользователя и выполнение триггера осуществляется в теле одной

транзакции: если произойдёт откат триггера, то будут отклонены и пользовательские изменения. Можно определить несколько AFTER-триггеров для каждой операции (INSERT, UPDATE, DELETE). Если для таблицы предусмотрено выполнение нескольких AFTER-триггеров, то с помощью системной хранимой процедуры sp_settriggerorder можно указать, какой из них будет выполняться первым, а какой последним. По умолчанию в SQL Server все триггеры являются AFTER-триггерами.

2. ***INSTEAD OF***. Триггер вызывается вместо выполнения команд. В отличие от AFTER-триггера, INSTEAD OF-триггер может быть определён как для таблицы, так и для просмотра. Для каждой операции INSERT, UPDATE, DELETE можно определить только один INSTEAD OF-триггер.

Триггеры различают также по типу команд, на которые они реагируют. Существует три типа триггеров: ***INSERT TRIGGER, UPDATE TRIGGER, DELETE TRIGGER.***

1. ***INSERT TRIGGER*** – запускаются при попытке вставки данных с помощью команды INSERT. Использование триггеров INSTEAD OF в операциях INSERT. Если триггер INSTEAD OF определён в операциях INSERT для таблицы или представления, то триггер выполняется вместо инструкции INSERT. Ранние версии SQL Server поддерживают только триггеры AFTER, определённые для инструкции INSERT и других инструкций, изменяющих данные.

2. ***UPDATE TRIGGER*** – запускаются при попытке изменения данных с помощью команды UPDATE.

3. ***DELETE TRIGGER*** – запускаются при попытке удаления данных с помощью команды DELETE.

Триггер создается операцией

CREATE TRIGGER *имя_триггера*
BEFORE | AFTER<*триггерное_событие*>
ON<*имя_таблицы*>
[REFERENCING
<*список_старых_или_новых_псевдонимов*>*]*
[FOR EACH { ROW | STATEMENT}]
[WHEN(*условие_триггера*)*]*
<*тело_триггера*>

В теле триггера могут быть использованы специальные таблицы: *inserted* и *deleted,* которые создаёт сервер БД при

выполнении команд добавления, изменения и удаления записей. Структура таблиц inserted и deleted идентична структуре таблиц, для которой определяется триггер. Для каждого триггера создается свой комплект таблиц inserted и deleted, поэтому никакой другой триггер не сможет получить к ним доступ. В зависимости от типа операции, вызвавшей выполнение триггера, содержимое таблиц inserted и deleted может быть разным:

- команда INSERT – в таблице inserted содержатся все строки, которые пользователь пытается вставить в таблицу; в таблице deleted не будет ни одной строки; после завершения триггера все строки из таблицы inserted переместятся в исходную таблицу;
- команда DELETE – в таблице deleted будут содержаться все строки, которые пользователь попытается удалить; триггер может проверить каждую строку и определить, разрешено ли её удаление; в таблице inserted не окажется ни одной строки;
- команда UPDATE – при её выполнении в таблице deleted находятся старые значения строк, которые будут удалены при успешном завершении триггера, новые значения строк содержатся в таблице inserted, эти строки добавятся в исходную таблицу после успешного выполнения триггера.

Для получения информации о количестве строк, которое будет изменено при успешном завершении триггера, можно использовать функцию @@ROWCOUNT; она возвращает количество строк, обработанных последней командой. Следует подчеркнуть, что триггер запускается не при попытке изменить конкретную строку, а в момент выполнения команды изменения. Одна такая команда воздействует на множество строк, поэтому триггер должен обрабатывать все эти строки.

Если триггер обнаружил, что из 100 вставляемых, изменяемых или удаляемых строк только одна не удовлетворяет тем или иным условиям, то никакая строка не будет вставлена, изменена или удалена. Такое поведение обусловлено требованиями транзакции – должны быть выполнены либо все модификации, либо ни одной.

Триггер выполняется как неявно определённая транзакция, поэтому внутри триггера допускается применение команд управления транзакциями. В частности, при обнаружении нарушения ограничений целостности для прерывания выполнения триггера и

отмены всех изменений, которые пытался выполнить пользователь, необходимо использовать команду ROLLBACK TRANSACTION.

Для получения списка столбцов, изменённых при выполнении команд INSERT или UPDATE, вызвавших выполнение триггера, можно использовать функцию COLUMNS_UPDATED(). Она возвращает двоичное число, каждый бит которого, начиная с младшего, соответствует одному столбцу таблицы (в порядке следования столбцов при создании таблицы). Если бит установлен в значение "1", то соответствующий столбец был изменен. Кроме того, факт изменения столбца определяет и функция UPDATE (имя_столбца).

Пример описания триггера приведён на рисунке 21, а результат его выполнения – на рисунке 22.

```
CREATE TRIGGER changeIdTchr ON Tchr FORUPDATEAS
    DECLARE @oldIdTchr int, @newIdTchr int
    SELECT @oldIdTchr = Id_Tch FROM deleted
    SELECT @newIdTchr = Id_Tch FROM inserted
    UPDATE Course SET Id_Tch = @newIdTchr
    WHERE Course.Id_Tch = @oldIdTchr;
```

Рисунок 21 – Описание триггера

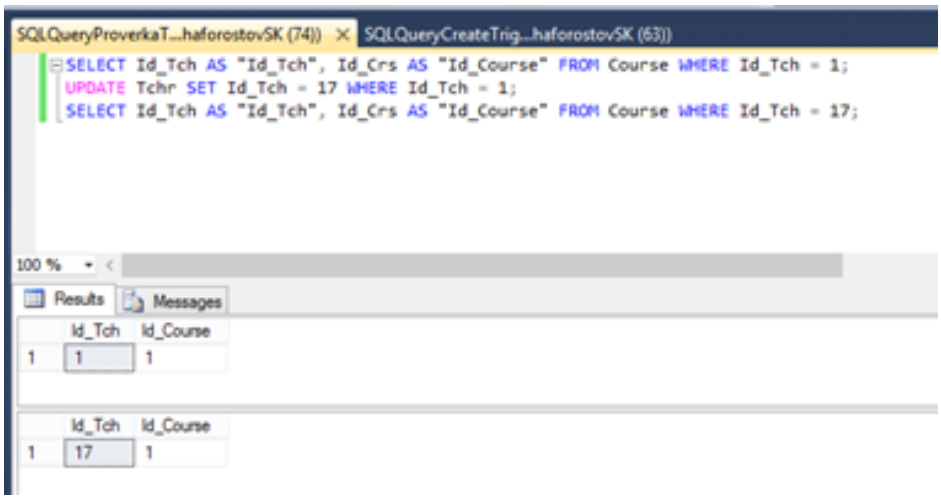


Рисунок 22 – Результат выполнения триггера

Отключить триггер можно командой:

DISABLE TRIGGER { *имя_триггера* [,...n] | ALL }
ON { *имя_таблицы* | ***DATABASE*** | ***ALL SERVER*** } [;]

Параметр ***ALL*** означает, что все триггеры в области действия предложения ON будут отключены.

Параметр ***DATABASE*** показывает, что триггер был создан или изменен для выполнения в области БД.

Параметр ***ALLSERVER*** показывает, что триггер был создан или изменен для выполнения в области сервера.

Включить уже существующий триггер можно командой:

ENABLE TRIGGER { *имя_триггера* [,...n] | ***ALL*** }
ON { *имя_таблицы* | ***DATABASE*** | ***ALL SERVER*** } [;]

Параметр ***ALL*** указывает, что включаются все триггеры, определённые в области предложения ON.

Параметр ***DATABASE*** указывает, что триггер был создан или изменен для выполнения в области базы данных.

Параметр ***ALL SERVER*** показывает, что триггер был создан или изменен для выполнения в области сервера.

2. Последовательность выполнения практической работы

1. Запустить программу SQL Server Management Studio, выполнить регистрацию и соединение с БД.
2. Изменить описание таблицы Contract, описав поле Id_Crs как внешний ключ, запрещающий изменение записей родительской таблицы.
3. Выполнить оператор удаления курса, с которым связана хотя бы одна запись таблицы Contract.
4. Написать триггер, который реализует каскадное изменение полей в связанных записях в таблицах Contract и Tchr при изменении кода преподавателя.
5. Проверить работоспособность триггера, выполнив команду изменения кода преподавателя в таблице Tchr.
6. Разработать триггер, изменяющий величину скидки для слушателя по следующему правилу.

Если стоимость всех контрактов слушателя лежит в диапазоне 50 000÷70 000 руб., то размер скидки – 5%; если стоимость всех контрактов слушателя лежит в диапазоне 70 000÷80 000 руб., то размер скидки – 6%; если стоимость всех контрактов слушателя лежит в диапазоне 80 000÷10 0000 руб., то размер скидки – 10%; если стоимость всех контрактов слушателя превышает 100 000 руб., то размер скидки устанавливается в 15%.

7. Проверить работоспособность триггера, добавив контракт для слушателя.
8. Оформить отчёт по практической работе.

3. Требования к оформлению отчёта

1. Титульный лист.
2. Описание правила целостности для каждого правила.
3. Алгоритм реализации правила целостности.
4. Описание правила целостности на языке SQL.
5. Тестовые наборы данных для проверки правила целостности.
6. Результаты проверки правил целостности.
7. Выводы по работе.
8. Ответы на контрольные вопросы.

Контрольные вопросы

1. Определите назначение триггеров.
2. Для чего используются вспомогательные таблицы deleted и inserted?
3. В чём разница выполнения триггеров after и instead of?
4. Опишите алгоритм стандартных правил целостности, обеспечивающих каскадное обновление данных.
5. Что такое ссылочная целостность?
6. В чём суть применения триггеров для контроля целостности данных?
7. Как можно реализовывать ограничения целостности на «домен»?

ЛИТЕРАТУРА

1. Дунаев В.В. Язык SQL для студента. – СПб.: БХВ-Петербург, 2007. – 312 с.

2. Советов Б.Я., Цехановский В.В. Базы данных. Теория и практика. – М.: Юрайт, 2012. – 324 с.

3. Пирогов, В.Ю. Информационные системы и базы данных: Организация и проектирование. – СПб.: БХВ-Петербург, 2009. – 528 с.

4. https://technet.microsoft.com/ru-ru/library/ms189826(v=sql.90).aspx

5. Бен-Ган И., Сарка Д., Талмейдж Р. Microsoft SQL Server 2012. Создание запросов: учебный курс Microsoft. – М.: Изд-во «Русская редакция», 2015. – 700 с.

6. Бондарь А. Microsoft SQL Server 2012. – СПб.: BHV-СПб, 2013. – 608 с.

7. Бен-Ган И. Microsoft SQL Server 2012. Основы T-SQL. – М.: Эксмо, 2015. – 400 с.

ПРИЛОЖЕНИЕ I

Схема БД и описание таблиц

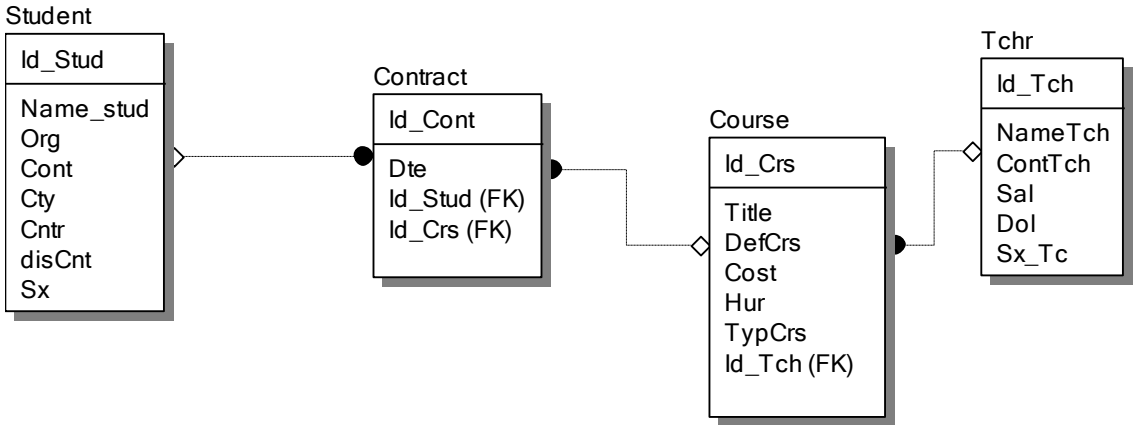


Рисунок 1 – Схема БД

Состав и описание таблиц БД

Таблица 1 определяет состав таблиц БД, в таблицах 2–5 приведены описания атрибутов каждой таблицы. В таблице 6 определены правила целостности для отдельных атрибутов.

Таблица 1 – Состав таблиц БД

Имя таблицы	Описание таблицы
Contract	Договор на оказание образовательной услуги
Course	Описание курса
Student	Сведения о слушателях
Tchr	Преподаватель

Таблица 2 – Описание столбцов таблицы ***Contract***

Имя столбца	Тип данных	Первичный ключ	Внешний ключ	Комментарий
Id_Cont	Integer	Yes	No	Номер контракта
Id_Stud	Integer	No	Yes	Код слушателя в БД
Dte	Datetime	No	No	Дата заключения
Id_Crs	Integer	No	Yes	Идентификатор курса

Таблица 3 – Описание столбцов таблицы ***Course***

Имя столбца	Тип данных	Первичный ключ	Внешний ключ	Комментарий
Id_Crs	Integer	Yes	No	Идентификатор курса
Title	varchar(20)	No	No	Название курса
DefCrs	varchar(20)	No	No	Краткое содержание курса (до 256 знаков)
Cost	Integer	No	No	Стоимость курса (не менее 500 руб.)
Id_Tch	Integer	No	Yes	Код преподавателя
Hur	Integer	No	No	Количество часов, не менее 16 часов
TypCrs	varchar(20)	No	No	Тип курса, может принимать значения начальный, продвинутый, профессиональный

Таблица 4 – Описание столбцов таблицы ***Student***

Имя столбца	Тип данных	Первичный ключ	Внешний ключ	Комментарий
Id_Stud	Integer	Yes	No	Код слушателя в БД
Name_stud	varchar(20)	No	No	Фамилия Имя отчество
Org	varchar(20)	No	No	Название организации
Cont	varchar(20)	No	No	Контактные данные
Cty	varchar(20)	No	No	Город проживания
Cntr	varchar(20)	No	No	Страна проживания
disCnt	Integer	No	No	Размер скидки на обучение
Sx	Char	No	No	Пол, может принимать значение «м» и «ж»

Таблица 5 – Описание столбцов таблицы ***Tchr***

Имя столбца	Тип данных	Первичный ключ	Внешний ключ	Комментарий
Id_Tch	Integer	Yes	No	Код преподавателя
NameTch	char(18)	No	No	Фамилия, имя отчество
ContTch	varchar(20)	No	No	Контактные данные
Sal	Integer	No	No	Стоимость 1 часа работы
Dol	varchar(20)	No	No	Степень
Sx_Tc	Char	No	No	Пол: «м» или «ж»
Id_Tch	Integer	Yes	No	Код преподавателя

Таблица 6 – Правила целостности для атрибутов

Имя атрибута	Имя таблицы	Min	Max
disCnt	Student	0	15
Sal	Tchr	500	10000
Hur	Course	16	

ПРИЛОЖЕНИЕ 2

Государственное бюджетное образовательное учреждение высшего образования
Московской области

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

Отчет по практической работе №_
по дисциплине БД

Выполнил(а) студент(ка) группы

Проверил(а)

Королев, 201_

Наталья Петровна Сидорова
Галина Николаевна Исаева

ОСНОВЫ T-SQL
Практикум по курсу «Базы данных»

Учебное пособие